Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

David Gaertner

Treppensteigen mit Laufrobotern: Untersuchung von Belohnungsfunktionen für Deep Reinforcement Learning

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

David Gaertner

Treppensteigen mit Laufrobotern: Untersuchung von Belohnungsfunktionen für Deep Reinforcement Learning

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Maschinenbau/Berechnung und Simulation
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

in Zusammenarbeit mit:
HAW Hamburg
Zentrum für industrielle Robotik
Berliner Tor 21
20099 Hamburg

Erstprüfer/in: Prof. Dr.-Ing. Thomas Frischgesell
Zweitprüfer/in: Prof. Dr. rer. nat. Sarah Hallerberg
Industrielle/r Betreuer/in: -

Abgabedatum: 03.07.2020

Zusammenfassung

Name des Studierenden
David Gaertner

Thema der Masterarbeit
Treppensteigen mit Laufrobotern: Untersuchung von Belohnungsfunktionen für Deep Reinforcement Learning

Stichworte
Reinforcement Learning, Fortbewegung, Laufroboter, Belohnungsfunktion, Treppensteigen

Kurzzusammenfassung
Die Untersuchung von Belohnungsfunktionen für Laufroboter liefert einen Beitrag, um die Leistungsfähigkeit von Reinforcement Learning in der Robotik zu steigern. Ein simuliertes Modell wird in der Fortbewegung über Treppen trainiert. Es sollen Parameter der Belohnungsfunktion identifiziert werden, die zielgerichtet spezifische Bewegungsformen hervorbringen.

Name of student
David Gaertner

Master thesis title
Climbing stairs with walking robots: Study of reward functions for Deep Reinforcement Learning

Keywords
reinforcement learning, locomotion, walking robot, reward function, stair climbing

Abstract
The study of reward functions for walking robots provides a contribution to enhance the capability of reinforcement learning in robotics. A simulated model is trained to surmount stairs. During this process parameters of the reward function should be identified that are suitable to induce a specific movement pattern.

Aufgabenstellung

Die Fortbewegung autonomer Roboter in unterschiedlichstem Gelände ist aktueller Gegenstand der Forschung. Wenngleich sich noch kein universelles, robustes Design etablieren konnte, haben sich Laufroboter in unwegsamem oder unstetem Gelände bisher als überlegen gegenüber fahrenden Systemen erwiesen.

In der Trajektorien- und Bahnplanung für Laufroboter halten zunehmend auch Techniken des maschinellen Lernens Einzug. In der näheren Vergangenheit sind zahlreiche Beiträge veröffentlicht worden, die erfolgreich Methoden des *Reinforcement-Learning* (Bestärkendes Lernen) auf verschiedene Aspekte der Fortbewegung anwenden. Ein wesentlicher Bestandteil des Reinforcement-Learning ist die Belohnungsfunktion (*reward function*).

Im Rahmen dieser Arbeit wird eine Reihe von Belohnungsfunktionen für das Training von Laufrobotern entwickelt. Während die Mehrheit der bisherigen Veröffentlichungen zu diesem Thema die Fortbewegung in der Ebene zum Gegenstand hat, liegt der Schwerpunkt der vorliegenden Forschungsarbeit auf dem Überwinden von Treppen und Hindernissen, als Beispiele einer natürlichen oder antropogenen Umgebung. Es werden verschiedene Belohnungsfunktionen in einem Reinforcement-Learning-Prozess erprobt und anhand der erlernten Bewegungsmuster, sowie dem Verlauf des Lernvorgangs bewertet. Grundlage der Bewertung ist ein zuvor zu entwickelndes Bewegungsprofil, sowie der zeitliche Aufwand des Lernvorgangs. Das Ziel ist es, geeignete Parameter für diesen Lernprozess zu identifizieren und ihren Einfluss auf das erlernte Verhalten zu bestimmen. Mithilfe dieser Erkenntnisse könnte die Effizienz des Gestaltungsprozesses von Belohnungsfunktionen gesteigert werden, indem eine zielgerichtete Abstimmung der Funktion zu einem spezifischen Anwendungsfall ermöglicht würde.

Als Trainingsobjekt wird das Simulationsmodell eines Laufroboters entwickelt. Seine kinematischen Eigenschaften sind angelehnt an reale Modelle aus thematisch ähnlichen Arbeiten und an vergleichbare kommerzielle Systeme. Um den Einfluss der Belohnungsfunktion identifizieren zu können, wird das Training für jede Funktion zu ansonsten einheitlichen Bedingungen durchgeführt. Die Implementierung des Reinforcement-Learning erfolgt in MATLAB, mit Simulink als Simulationsumgebung des Roboters. Darin wird eine physikalische Trainingsumgebung entsprechend der beschriebenen Aufgabe gestaltet. Die Einstellungen und Parameter betreffend des Reinforcement-Learning werden in der Vorbereitung des Trainings herausgearbeitet und bleiben in der Folge unverändert.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Beginnend in der zweiten Hälfte des vergangenen Jahrhunderts, halten Roboter bis heute mit zunehmender Geschwindigkeit Einzug in Arbeitswelt und Gesellschaft. Während sie aktuell hauptsächlich in der industriellen Fertigung eingesetzt werden, besitzen sie langfristig das Potential Umwälzungen in Wirtschaft und Gesellschaft zu bewirken, die vergleichbar sind mit den Folgen der Erfindung der Dampfmaschine oder der *Spinning Jenny*. Perspektivisch könnten Roboter den Menschen in vielen Lebensbereichen von seinen Aufgaben entbinden und damit Raum schaffen für eine grundlegende, strukturelle Neuausrichtung der Gesellschaft.

Die Gesamtheit der gegenwärtig im Einsatz befindlichen Roboter kann klassifiziert werden als mobile oder stationäre Systeme. Die Klasse der mobilen Roboter wird in deutlicher Mehrheit repräsentiert von fahrenden Robotern. Ihre Einsatzumgebung ist häufig bekannt oder sogar kontrolliert und räumlich begrenzt. Sie agieren in der Regel auf ebenem Untergrund mit fester Struktur. Typische Beispiele sind mobile Plattformen in Fertigungsstraßen, Transportsysteme in Lagerräumen oder Haushaltsroboter.

Laufroboter stellen eine zweite Klasse mobiler Roboter dar. Hinsichtlich der Mobilität in unwegsamem oder unstetem Gelände, sowie auf losen und wechselnden Untergründen, sind sie fahrenden Systemen überlegen. Sie sind damit geeignet um Aufgaben in anspruchsvoller und gegebenenfalls natürlicher Umgebung wahrzunehmen. Die möglichen Anwendungen sind vielfältig:

- Erkundung von unbekanntem Terrain (terrestrisch und extraterrestrisch)
- Sicherung und Überwachung
- Transport (z.B. Paketzustellung bis zur Wohnungstür)
- Dienstleistung und Service
- Unterhaltung und Sport

Dennoch sind sie bisher deutlich weniger verbreitet und sind noch vermehrt Gegenstand von Forschung und Entwicklung. Ein Grund dafür ist der Preis, mit dem die überlegene Mobilität gegenüber fahrenden Systemen erkauft wird. Da Laufroboter einerseits

keine systemimmanente Stabilität besitzen und andererseits komplexe Bewegungsmuster zur Fortbewegung erfordern, sind die Ansprüche an ihre Steuerung ungleich höher. In Kombination mit einer komplexen und möglicherweise dynamischen Umwelt, stellt dies eine der großen Herausforderungen der autonomen Fortbewegung dar.

Die Fortbewegung von Laufrobotern wird häufig in Form einer zyklischen Kombination von vordefinierten Bewegungssequenzen realisiert. Oftmals sind dies Imitationen natürlicher Vorbilder, sodass neben der Kinematik häufig auch spezielle Gangarten, etwa von Insekten, Säuge- oder Spinnentieren, technisch nachgebildet werden. Moderne Methoden der Trajektorien- und Bewegungsplanung bilden den Roboter und seine Umgebung in einem Modell ab und finden die nächste Aktion als Lösung eines algebraischen Optimierungsproblems.

Limitierender Faktor für den Erfolg solcher Modelle ist die Komplexität der Umgebung in der sie eingesetzt werden. Ein Roboter, der sich mithilfe einer starren Bewegungsroutine erfolgreich auf ebenem Untergrund vorwärts bewegt, stürzt gegebenenfalls in der schiefen Ebene. Ein Modell für die ganzheitliche Bewegungsplanung zwischen Start- und Zielort, kann scheitern wenn die Umgebung während der Ausführung nicht statisch ist. Reale und gegebenenfalls natürliche Umgebungen sind in der Regel zu komplex, um in manuell programmierten Bewegungsalgorithmen ausreichend abgebildet werden zu können.

Einen alternativen Lösungsansatz für den Umgang mit Komplexität bieten Methoden des maschinellen Lernens. In den vergangenen zehn Jahren sind eine Reihe von Beiträgen veröffentlicht worden, in denen Reinforcement-Learning (RL) erfolgreich eingesetzt wurde, um Problemstellungen der Robotik zu bearbeiten. Begünstigt durch die Weiterentwicklung von Algorithmen und die stetig anwachsende verfügbare Rechenleistung konnten auch für anspruchsvolle Aufgaben, wie die autonome Fortbewegung von Laufrobotern, Lösungsansätze gefunden werden. Triebfeder für diese Entwicklung ist die Hoffnung, dass RL durch einen iterativen, flexiblen Lernprozess Lösungen hervorbringt, die robuster und allgemeingültiger sind als es mit algebraischen Modellen möglich ist.

1.1. Motivation und Zielsetzung

Wesentlicher Bestandteil des RL ist die Belohnungsfunktion. Ein durch RL erlerntes Verhalten, die *Policy*, ist das Ergebnis der Interpretation der Belohnungsfunktion durch den Agenten (engl. *agent*) während des Lernvorgangs. In Bezug auf einen Laufroboter

hat sie damit unmittelbaren Einfluss auf Art und Qualität von gelernten Bewegungsformen.

Trotz ihrer elementaren Bedeutung gibt es bisher keine Arbeiten, die den praktischen Gestaltungsprozess einer Belohnungsfunktion betrachten. Der Schwerpunkt bisheriger Veröffentlichungen liegt auf dem Nachweis, dass Roboter Bewegungsarten mithilfe von RL erlernen können. In diesem Prozess verwendete Belohnungsfunktionen werden präsentiert, jedoch bleibt ihre Entstehung weitgehend unkommentiert. Sie geben wenig Aufschluss darüber, wie die Ausgestaltung der Belohnungsfunktion den Lernprozess und das erlernte Verhalten beeinflusst und von welcher Qualität die gelernten Bewegungsmuster sind.

Um RL effektiv und zielgerichtet zur Lösung der zu Beginn genannten Herausforderungen einsetzen und die Leistungsfähigkeit gegenüber den etablierten Methoden bewerten zu können, sind detailliertere Erkenntnisse über Einfluss und Wirkung der Belohnungsfunktion erforderlich. Es ist das Ziel dieser Arbeit, Erfahrungen in der Gestaltung von Belohnungsfunktionen für Laufroboter zu generieren.

Erfolgversprechende Lösungsansätze die während dieses Prozesses gefunden werden, können die zukünftige Forschungsarbeit zu diesem Thema beschleunigen. Zu verstehen, wie mit RL zielgerichtet Bewegungsmuster erlernt werden können, ist ein bedeutsamer Schritt, um dem Einsatz autonomer Roboter in unserer alltäglichen Umgebung näher zu kommen. Herausforderungen und Grenzen die im Zuge der Arbeit identifiziert werden, liefern einen Beitrag zur Standortbestimmung von RL in der Robotik.

1.2. Methodik und Struktur

Im Anschluss an eine theoretische Auseinandersetzung mit dem Gestaltungsprozess, wird eine Reihe von Belohnungsfunktionen entwickelt und im Training eines Laufroboters erprobt. Bewertet werden sowohl der Verlauf des Lernprozesses, als auch das erlernte Bewegungsmuster. Während die Bewertung des Lernprozesses eher statistischer Natur ist, wird die Qualität der Bewegung an praktischen Anforderungen für den Einsatz in alltäglicher und natürlicher Umgebung gemessen.

Die Erprobung erfolgt am Modell eines Laufroboters, dessen kinematischen und technischen Eigenschaften einerseits an den Anforderungen autonomer Fortbewegung und andererseits an realen, für den Einsatz in alltäglicher Umgebung geeigneten, Robotern orientiert sind. Als Trainingsumgebung dient ein Parcours mit dem Schwerpunkt

auf Treppen und Hindernisse, um einige praxisnahe Herausforderungen für Laufroboter abzubilden. Das Training des Roboters ist dabei ausschließlich auf die Simulation beschränkt. Versuche an einem realen System sind nicht vorgesehen.

Ausschließlicher Forschungsgegenstand dieser Arbeit ist die Belohnungsfunktion. Dabei steht ein praxisorientierter und empirischer Ansatz im Vordergrund. Die Versuche werden zu gleichbleibenden Trainingsbedingungen für alle getesteten Funktionen durchgeführt. Eine Variation oder Optimierung der Trainingsumgebung, des Robotermodells, sowie der Einstellungen und Parameter die das RL betreffen, ist hierin nicht enthalten.

Im nachfolgenden Kapitel wird ein Überblick über aktuelle Entwicklungen und Forschungsthemen aus den Bereichen Robotik und RL gegeben, soweit diese das Thema dieser Arbeit berühren. Kapitel 3 enthält einen Einstieg in die Methoden von RL und die damit verbundene Verwendung künstlicher, neuronaler Netze. Die zu untersuchenden Belohnungsfunktionen werden in Kapitel 5 entwickelt und vorgestellt. Nachfolgend wird die zur Erprobung verwendete Trainingsumgebung dokumentiert (Kapitel 4). Darin enthalten sind Erläuterungen zur getroffenen Wahl der Einstellungen und Versuchsbedingungen. Die Vorstellung und Bewertung der Trainingsergebnisse sind Inhalt von Kapitel 6. Kapitel 7 fasst die wesentlichen Erkenntnisse der Arbeit abschließend zusammen und gibt einen Ausblick auf weiterführende Fragestellungen.

2. Stand der Technik

Diese Arbeit steht im Kontext der Herausforderung, Laufroboter erfolgreich in alltäglicher und natürlicher Umgebung zu steuern und bewegen. Die damit verbundenen Fragestellungen sind in Teilen unabhängig davon, ob für die technische Umsetzung eines Systems Methoden des maschinellen Lernens oder klassische Ansätze aus der Robotik verwendet werden. Der erste Teil dieses Kapitels gibt deshalb einen Überblick über aktuelle Beiträge zur Fortbewegung von Laufrobotern, die nicht in Verbindung zu RL stehen.

RL ist als Teildisziplin maschinellen Lernens zunächst einmal unabhängig von der Robotik. Angewendet auf Problemstellungen wie die Fortbewegung von Laufrobotern, ist der zu erwartende Erfolg jedoch auch abhängig von der Leistungsfähigkeit des RL als Methode. Im zweiten Abschnitt werden daher aktuelle Entwicklungen und Forschungsschwerpunkte aus diesem Bereich betrachtet.

Im Anschluss daran wird ein Überblick über Beiträge gegeben, die RL in Verbindung bringen mit Fragestellungen aus der Robotik. Die vorgestellten Themen beinhalten mehrheitlich, aber nicht ausschließlich, die Forschung an Laufrobotern.

Im letzten Abschnitt werden Arbeiten behandelt, die die Gestaltung von Belohnungsfunktionen zum Thema haben.

2.1. Laufroboter

Ein wesentlicher Bestandteil der Forschung an Laufrobotern ist die technische Nachbildung natürlicher Vorbilder. Die Orientierung an der Natur entspringt der Annahme, dass dort über evolutionäre Prozesse im Verlauf von mehreren Millionen Jahren optimale Lösungen entstanden sind.

Ein Aspekt, dem vor diesem Hintergrund viel Beachtung geschenkt wird, ist die Wahl der Kinematik. (Gasparetto et al., 2008) analysieren die Eigenschaften von Spinnen und leiten daraus ein kinematisches Modell ab, welches deren Flexibilität und Beweglichkeit

abbildet. Darüber hinaus berücksichtigen sie die adhäsiven Eigenschaften des Spinnenfußes, um ein gültiges Modell auch für kletternde Roboter bereitzustellen. (Sarmah et al., 2018) verwenden für ihre Plattformen eine spinnenartige Kinematik, reduzieren aber die Anzahl der Beine. Neben Spinnen dienen auch Insekten (Pa et al., 2012) und Säugetiere, wie z.B. Hunde als natürliche Vorbilder (Spröwitz et al., 2018), (Zamani et al., 2018), (Bledt et al., 2018). Einen Gegenentwurf zu biomimetischen Kinematiken liefern (Moore et al., 2002). Sie verwenden den Hexapod RHex mit einer rotierenden, halbkreisförmigen Beingeometrie um ein stabiles Treppensteigen zu ermöglichen. Noch einen Schritt weiter gehen (Huang et al., 2017) und erlauben durch die Verwendung von radförmigen Füßen mit variabler Exzentrizität den fließenden Wechsel zwischen gehender und fahrender Kinematik.

Neben der Kinematik werden auch aus der Natur bekannte Gangarten imitiert. (Sarmah et al., 2018) und (Zamani et al., 2018) implementieren eine Bewegungssequenz, wie sie von Paarhufern praktiziert wird. (Biancardi et al., 2011) untersuchen die Gangarten von Vogelspinnen durch Videoanalyse und bewerten sie unter dem Aspekt der Energieeffizienz. (Moore et al., 2002) verwenden die unnatürliche Kinematik von *RHex* in Kombination mit Bewegungssequenzen, wie sie bei Insekten zu beobachten sind.

Der *MIT Cheetah 3* (Bledt et al., 2018) beherrscht mehrere natürliche Gangarten. Im Gegensatz zu starren Implementierungen ist er in der Lage die Gangart fließend zu wechseln und situativ anzupassen. Einen ähnlichen Ansatz verfolgen (Johnson et al., 2011) auf der Plattform *X-RHex*. Anstelle einer fixen Bewegungsart hinterlegen sie eine Reihe von primitiven Bewegungsmustern, die in Abhängigkeit der Umgebungswahrnehmung zur Fortbewegung kombiniert werden. Die Rekombination von Bewegungssequenzen ist auch Thema von (Fasih et al., 2009). Sie trainieren mithilfe eines genetischen Algorithmus ein neuronales Netz für verschiedene Verhaltensmuster. Im zweiten Schritt wird eine hierarchische Struktur eingeführt, in der eine übergeordnete Steuerung aus dem Katalog der gelernten Bewegungen auswählt und sie zur Erfüllung einer Aufgabe zusammenfügt. Ähnlich dazu und ebenfalls biologisch inspiriert ist das Modell von (Henley et al., 2004), die ein Hormonsystem zur Entscheidungsfindung in ihrer Steuerung abbilden.

Die Ansprüche an die Interaktion der Steuerung mit der Umgebung fallen unterschiedlich aus, was durch die Auswahl der Sensoren wiedergegeben wird. Der Hexapod *HITCR-II* (Zhanghe et al., 2014) verzichtet vollständig auf externe Sensoren. Die Bewegung wird geleitet von einer gewünschten Kraftverteilung an den Füßen, die zur Erfüllung einer Stabilitätsbedingung führt. Ein ähnlicher Ansatz wird von (Bledt et al., 2018) verwendet. Der MIT Cheetah 3 registriert unerwartete Kontakte mit der Umgebung durch Kollision mit seinen Füßen und passt seine Bewegung an, um Stabilität zu

wahren.

Welche Schwierigkeiten mit der Verarbeitung einer Außenwahrnehmung verbunden sind, zeigen andere Plattformen. Der Vierfüßler in (Sarmah et al., 2018) wechselt die Gangart bei Detektion eines Hindernisses von Gehen zu Klettern. Durch die Verwendung eines simplen Ultraschallsensors wird praktisch jedes Objekt als Treppe identifiziert. Obwohl die Ausstattung von X-RHex mit einem Lidar-System demgegenüber wesentlich fortschrittlicher ist, berichten (Johnson et al., 2011) von Schwierigkeiten bei der Wahl der richtigen Bewegungsstrategie. Ebenso dokumentieren (Fasih et al., 2009) Probleme im Umgang mit komplexen Umgebungen. Die Klassifizierung der Umwelt in diskrete Kategorien ist eine Herausforderung für diese Art der Steuerung. Besonders deutlich wird dies in der Arbeit von (Ilyas et al., 2018). Als Vorbereitung für die Entwicklung eines autonomen Laufroboters, trainieren sie ein neuronales Netz zur Lokalisierung von Treppen durch Bilderkennung. Das System soll die autonome Fortbewegung des Roboters in mehrstöckiger Umgebung ermöglichen. Der Algorithmus detektiert Treppen anhand der charakteristischen parallelen Kanten einzelner Stufen. Während Treppen mit hoher Wahrscheinlichkeit als solche erkannt werden, lösen auch andere Objekte wie Sitzgruppen, Leitern, Holzfußböden und Geländer eine Klassifizierung aus.

Eine wesentliche Hürde für die praktische Forschung an Laufrobotern ist der Umgang mit Fehlversuchen. Schäden und Verschleiß führen zu hohen Kosten und Zeitverlust. Ein Schwerpunkt liegt daher auf der Entwicklung von Designs für Roboter als Forschungsplattform. *Cheetah-cub* (Spröwitz et al., 2013) ist entwickelt worden, um komplexe Bewegungsmuster besser erforschen zu können. Durch ein spezielles Design der Beine, verfügt er über selbst-stabilisierende Eigenschaften. Die geringe Größe macht schwere Beschädigungen weniger wahrscheinlich. Oncilla (Spröwitz et al., 2018) ist als Open-Source-Forschungsplattform für die Untersuchung von Gangarten entwickelt worden, um die Forschung an Laufrobotern durch bessere Verfügbarkeit zu beschleunigen. Derselben Intention folgen (Dholakiya et al., 2019) mit *Stoch*, dessen Komponenten mehrheitlich im 3D-Druckverfahren hergestellt werden und so die Kosten für Herstellung und Reparatur klein halten.

Als Maßstab für die derzeitige technische Reife von Laufrobotern kann die kommerzielle Verfügbarkeit herangezogen werden. Alle bisher vorgestellten Modelle sind als Forschungsplattform konzipiert. Systeme, die für einen konkreten Anwendungsfall entwickelt wurden (Spielzeuge ausgenommen) und darüber hinaus frei zu erwerben sind, bilden bislang die Ausnahme. Der hundeähnliche Vierfüßler *Aibo*, ursprünglich als Unterhaltungsroboter entworfen, wird mit der Fähigkeit zur autonomen Überwachung von Wohnräumen beworben (*Sony* 2020). Einen ähnlichen Zweck, jedoch im industriellen Umfeld, erfüllt *ANYmal* ((*Anybotics* 2020)). Seine Abmessungen und Sensorik erlau-

ben die Fortbewegung in Umgebungen, die Treppen, Schrägen, Engstellen und Hindernisse beinhalten. Eine Nutzlast von bis zu 10 kg ermöglicht die Erweiterung der technischen Ausstattung oder die Verwendung als Transportmittel. In Umgebungen die einmalig gelernt wurden ist zudem autonome Navigation möglich. Hinsichtlich der technischen Merkmale und propagierten Einsatzzwecke direkt vergleichbar mit ANYmal ist *Spot* (*Boston Dynamics* 2020). Für das Jahr 2021 ist eine Erweiterung um einen Manipulator mit sechs Freiheitsgraden angekündigt, was die mögliche Aufgabenvielfalt steigern wird. Die Laufzeit von Spot ist mit 90 Minuten bei einer Höchstgeschwindigkeit von 1,6 $\mathrm{m\,s^{-1}}$ angegeben, gegenüber 2 bis 4 Stunden und 1,0 $\mathrm{m\,s^{-1}}$ für ANYmal.

Nicht frei erhältlich, aber für einen konkreten Einsatzzweck konzipiert ist *BigDog* (Raibert et al., 2008). Mit der Größe eines Esels soll der Vierfüßler als begleitender Transporter für Ausrüstung und Marschgepäck von Soldaten dienen. Die Besonderheit dieses Systems liegt im hydraulischen Antriebskonzept, das eine Nutzlast von 150 kg ermöglicht. Die Energiebereitstellung erfolgt über einen Verbrennungsmotor, was eine erhebliche Lärmentwicklung bedeutet. Infolgedessen wurde BigDog für nicht einsatztauglich befunden und die Weiterentwicklung eingestellt (*Guardian* 2020).

ANYmal und Spot repräsentieren den Stand der Technik für Laufroboter, wenngleich insgesamt von einem Stand der Forschung gesprochen werden kann. Dabei betreffen nicht alle Fragestellungen exklusiv das Feld der Robotik. Die angesprochene Interaktion mit der Umwelt mithilfe von Sensoren ist ein eigenständiges Forschungsthema. Ebenso ist die Begrenzung des Aktionsradius, beziehungsweise der Einsatzdauer, von elektrischen Systemen eine Frage der Energiebereitstellung, etwa durch Batterietechnologie.

2.2. Deep Reinforcement Learning

Die grundsätzliche Funktionsweise von RL wird häufig anhand von Beispielen wie *Grid World*, Backgammon oder einarmigen Banditen dargestellt. Praktische Anwendungen wie die Bild- und Messdatenverarbeitung oder komplexe Steuerungen sind im Gegensatz dazu häufig hochdimensional und bergen die Notwendigkeit, einen hohen Datendurchsatz oder große Datenmengen zu bearbeiten. Unter Einsatz der heute verfügbaren Rechenkapazitäten werden dabei regelmäßig Rechenzeiten erreicht, die, gemessen am Anwendungsfall, für nicht praktikabel befunden werden. Gleichzeitig ist bei steigender Rechenleistung ein Reboundeffekt hinsichtlich der Komplexität der ausgewählten Aufgaben zu verzeichnen. Es ist deshalb ein stetiges Bestreben, die RL-Methoden Leistungsfähiger und effizienter zu gestalten.

Die aktuellen Entwicklungen diesbezüglich können insgesamt in zwei Felder aufgeteilt werden. Einerseits ist dies die Verbesserung oder Neuentwicklung von Algorithmen, andererseits sind dies Untersuchungen zur richtigen Wahl von Trainingsparametern.

Mit der Einführung des *Deterministic-Policy-Gradient*-Theorems führen (Silver et al., 2014) eine neue Klasse von Algorithmen ein. Gegenüber des *Stochastic-Policy-Gradient*-Theorems ersetzen sie das Integral über den Aktionsraum (*action space*) durch eine lokale Abschätzung des Gradienten der *Action-Value-Function*. Die Berechnung des Gradienten der Policy wird dadurch wesentlich schlanker. In hochdimensionalen oder kontinuierlichen Aktionsräumen sind diese Algorithmen deutlich effizienter als die stochastischen Varianten.

Einen Beitrag zur Anwendbarkeit von RL auf hochdimensionale Probleme liefern (Mnih et al., 2015) mit dem *Deep-Q-Learning*-Algorithmus. Sie trainieren einen Agenten in 49 *Atari*-Videospielen. Die direkte Verwendung von Bildpunkten als Sensordaten anstelle von vorverarbeiteten Zuständen stellt ein Novum im RL dar. Sie erweitern den Ansatz des *Q-Learning* um die *Experience-Replay*-Technik und Lösen damit die Korrelation zwischen aufeinanderfolgenden Policy-Updates auf. Zudem werden Erfahrungen aus dem *Replay-Buffer* (Speicher der Erfahrungen) mehrfach für Parameter-Updates verwendet und damit effizienter genutzt. Des Weiteren überwinden sie Stabilitätsprobleme des Trainings, die bei der Repräsentation der *Q-Function* durch tiefe neuronale Netze auftreten. Eine Kopie des neuronalen Netzes sorgt für die Trennung der zu trainierenden Q-Function, von der für die Ermittlung der Zielwerte verwendeten.

Die Erkenntnisse der Arbeiten von (Silver et al., 2014) und (Mnih et al., 2015) werden von (Lillicrap et al., 2015) zusammengeführt im *Deep Deterministic Policy Gradient*-Algorithmus (DDPG). Er ermöglicht die direkte Anwendung von RL auf das Training von tiefen neuronalen Netzen in kontinuierlichen Zustands- und Aktionsräumen mit hochdimensionalen Sensordaten.

Explizit für die Ausnutzung moderner Rechnerarchitekturen entworfen ist der *Distributed Proximal Policy Optimization*-Algorithmus (DPPO). Darin vereinen (Heess; T.B. et al., 2017) Eigenschaften von *Trust Region Policy Optimization* (TRPO) (Schulman et al., 2015) und *Proximal Policy Optimization* (PPO) (Abbeel et al., 2016). Sie parallelisieren sowohl den Prozess des Datensammelns, als auch die Gradientenberechnung. Der leistungssteigernde Beitrag liegt in einer optimierten Koordination zwischen den Recheninstanzen, in Abhängigkeit des Trainingsverlaufs, die eine Skalierbarkeit der Rechenzeit mit der Rechneranzahl ermöglicht.

Mit dem *Soft Actor-Critic*-Algorithmus (SAC) zielen (Haarnoja; Zhou; Abbeel et al., 2018) darauf ab, Schwächen der bisher genannten Algorithmen, unter Beibehaltung ihrer Vorteile, zu beseitigen. Hauptaugenmerk ist das Beibehalten der Stabilität von *On-Policy*-Algorithmen wie TRPO und PPO, bei einer Effizienz wie sie der *Off-Policy*-Algorithmus DDPG ermöglicht.

Wichtiger Bestandteil von RL-Algorithmen sind Strategien zur Erkundung des Zustands- und Aktionsraumes. In Abkehr zu (Silver et al., 2014) verwenden (Lillicrap et al., 2015) anstelle einer stochastischen eine deterministische Policy mit einem *Ornstein-Uhlenbeck*-Prozess (Uhlenbeck et al., 1930), um ausreichende *Exploration* (Erkundung) sicherzustellen. Den von der *Behaviour Policy* ausgewählten Aktionen wird ein zufallsbasiertes Störsignal überlagert, dessen Wertebereich und Zeitverhalten steuerbar sind.

Dieser Ansatz wird von (Nauta et al., 2019) erneut aufgegriffen, um eine effiziente Exploration-Strategie zu entwickeln. Sie verwenden den Ornstein-Uhlenbeck-Prozess um eine *Brownsche Bewegung* des Agenten im Zustands- und Aktionsraum zu erreichen. Die Werte der Aktionen sollen nicht nur zufällig verrauscht werden, sondern in einer Art und Weise, dass die im Raum besuchten Punkte normalverteilt sind.

Einem ähnlichen Gedanken folgen (Haarnoja; Zhou; Abbeel et al., 2018) im SAC-Algorithmus mit einer Entropie-basierten Exploration-Strategie. Sie erweitern die RL-Zielfunktion um ein Maß für die Entropie der gelernten Policy. Der lernende Agent wird dadurch angehalten, die Summe der erhaltenen Belohnung zu maximieren und währenddessen möglichst zufällig zu agieren. Der Entropie-Term fördert die Erkundung des Zustands- und Aktionsraumes und bestärkt den Agent zusätzlich, Pfade zu verlassen, die wenig erfolgversprechend sind.

Neben den RL-Algorithmen selbst, hat die Wahl von Parametern für das Training neuronaler Netze großen Einfluss auf die Leistungsfähigkeit von RL. Die Entscheidung für einen Parameter wird oft in dem Spannungsfeld aus Geschwindigkeit und Stabilität des Trainings, sowie der Qualität des Ergebnisses getroffen. Werden Parameter hinsichtlich der Geschwindigkeit optimiert, divergiert gegebenenfalls der Lernprozess. Besitzt die Qualität des Gelernten hohe Priorität, ist die Geschwindigkeit möglicherweise unpraktikabel. Entsprechend viel Aufmerksamkeit erhält deshalb die Entwicklung von Strategien zur richtigen Wahl der Parameter.

Im Beitrag von (Smith et al., 2017) wird die Wechselwirkung der Lernrate mit der *Mini-Batch*-Größe und deren Einfluss auf die Leistung des gelernten Verhaltens diskutiert. Entgegen der verbreiteten Praxis, die Lernrate während des Trainings zu reduzieren, empfehlen sie stattdessen eine schrittweise Erhöhung der Mini-Batch-Größe. In Versuchen zur Bildklassifizierung erreichen sie damit vergleichbare Ergebnisse, wobei weni-

ger Updates der Netzwerkparameter nötig sind. Ihr Ansatz zielt darauf ab, steigende Rechenkapazitäten und die Möglichkeit zur massiven Parallelisierung des Lernprozesses auszunutzen. Das Trainingsziel soll mit weniger Updates von höherer Qualität schneller erreicht werden.

Ein häufig bei der Verwendung von großen Batch-Größen beobachtetes Problem, ist der *Generalization Gap*. Demzufolge generalisieren neuronale Netze schlechter wenn sie mit großen Batch-Größen trainiert wurden, gegenüber dem Training mit kleinen Batch-Größen. Um dem entgegenzuwirken entwickeln (Hoffer et al., 2017) ein Schema, dass den Verlauf der Lernrate in Abhängigkeit der Batch-Größe skaliert. Sie erreichen damit eine vergleichbare Generalisierungsfähigkeit für große und kleine Batch-Größen, bei gleicher Anzahl der Updates. Der von (Smith et al., 2017) angestrebte Geschwindigkeitsvorteil für große Batch-Größen wird dadurch egalisiert.

Dem gleichen Thema wenden sich (Keskar et al., 2016) mit einer ähnlichen Strategie zu. Anstatt den Verlauf der Lernrate mit der Batch-Größe zu skalieren, empfehlen sie eine zeitlich veränderliche Batch-Größe. Kleine Batch-Größen zu Beginn des Trainings sollen die freie Bewegung im Parameterraum begünstigen, große Batch-Größen im späteren Verlauf die Konvergenz sicherstellen.

Entgegen dem Trend von (Smith et al., 2017), (Hoffer et al., 2017) und (Keskar et al., 2016) sprechen sich (Masters et al., 2018) für die Wahl von kleineren Batch-Größen aus. Neben deren Einfluss auf die Generalisierung betrachten sie auch die Stabilität und das Konvergenzverhalten des Trainings. Kleine Batch-Größen wirkten sich demnach positiv auf die Stabilität aus und seien daher zu bevorzugen.

Die mitunter kontroverse Diskussion um die Wahl von Lernrate und Batch-Größe spiegelt einerseits die Dynamik in der aktuellen Forschung wieder und andererseits die Schwierigkeit leistungsfähige Standards zu entwickeln. Letzteres ist auch dem mehrheitlich empirischen Charakter der Forschungsarbeit geschuldet. Allgemeingültige Strategien und Techniken sind selten und gute Praktiken entwickeln sich nur langsam. Eine Zusammenstellung von Empfehlungen, für das Training tiefer neuronaler Netze, die sich als guter Ausgangspunkt erwiesen haben, liefert (Bengio, 2012). Einen Überblick über verschiedene Varianten und häufig verwendete Algorithmen von Gradientenverfahren, sowie deren Stärken und Schwächen, gibt (Ruder, 2016).

Ein Beispiel für eine erfolgreiche Strategie zur Wahl von Parametern, deren Leistungsfähigkeit theoretisch belegt ist, liefern (Glorot et al., 2010). Sie präsentieren ein Schema zur Initialisierung von *Weights* und *Biases* neuronaler Netze in Abhängigkeit der Schichtgrößen und weisen dessen günstige Eigenschaften für die Fortpflanzung von Gradienten nach.

Weiterentwickelt wird das Verfahren von (Glorot et al., 2010) in der Arbeit von (He et al., 2015). Sie konzentrieren sich dabei auf ein Initialisierungsschema, dass speziell auf die Eigenschaften der *Rectified Linear Unit*-Aktivierungsfunktion (ReLU) zugeschnitten ist und liefern auch hierfür einen theoretischen Nachweis.

Während der Erfolg der Schemata von (Glorot et al., 2010) und (He et al., 2015) abhängig ist von der gewählten Aktivierungsfunktion, bieten (Mishkin et al., 2015) mit der *Layer-Sequential Unit-Variance*-Initialisierung ein Verfahren an, dass diesbezüglich flexibel einsetzbar ist. Anstelle einer direkten, aber zufallsbasierten Initialisierung bestimmen sie günstige Startwerte in einem iterativen Prozess.

2.3. Robotik und Reinforcement Learning

RL hat in den vergangenen Jahren große Aufmerksamkeit durch die Robotik erfahren. Die Kombination von RL und Robotern beschäftigt Forscher schon länger. In einer umfassenden Studie tragen (Kober et al., 2013) Beiträge aus dem Zeitraum von 1988 bis 2012 zusammen und benennen offene Herausforderungen, die aus diesen Arbeiten hervorgehen.

Ein grundlegendes Problem erwächst aus dem Effekt der als *Reality-Gap* bekannt ist. RL verlangt das Sammeln einer großen Menge von Erfahrungen aus denen gelernt werden kann. Ein Prozess, für den sich die Simulation anbietet. Allerdings können Simulationsmodelle die Komplexität der Realität nicht vollumfänglich abbilden, sodass virtuell erlerntes Verhalten nicht direkt auf ein reales System übertragbar ist. Umgekehrt erfordern Versuche am echten Roboter Zeit und verursachen in einer Vielzahl von Trial-and-Error-Durchläufen Kosten, Verschleiß und Schäden. Zudem erfordern sie Abläufe in Echtzeit, was eine eigene Herausforderung für Algorithmen und Hardware darstellt.

Hinsichtlich des Rechenaufwands der benötigt wird, um ein optimales Verhalten zu lernen, ist mit dem von Robert Bellman formulierten *Curse of Dimensionality* (Bellman, 1957) umzugehen. Demnach ist die gesuchte Lösung Teil eines Raumes, dessen Größe exponentiell mit der Anzahl seiner Dimensionen anwächst. Während diese These im Zusammenhang mit diskreten Räumen entstanden ist, trifft ihre Implikation auch für Roboter zu, deren Zustands- und Aktionsraum hochdimensional oder gar kontinuierlich ist.

Die Studie stellt Lösungsansätze vor und diskutiert deren Vor- und Nachteile. Die nachfolgende Auswahl von Beiträgen stammt mehrheitlich aus den vergangenen vier Jahren und zeigt, dass die Herausforderungen noch immer aktuell sind.

Nachdem eine Reihe von Arbeiten das Problem der Trainingszeiten zu umgehen versucht, indem sie Teile der Lösung bereits vorgeben (vgl. (Kober et al., 2013)), unternehmen (Gu et al., 2016) den Versuch, reale Roboterarme mit RL, ohne Vorgaben oder menschlichen Eingriff, zu trainieren. Ihr Fokus liegt dabei auf dem effizienten Sammeln von Erfahrungen, um die Trainingszeit zu reduzieren. Sie präsentieren hierzu einen Algorithmus, der ein asynchrones, paralleles Lernen auf mehreren Robotern ermöglicht. Im Gegensatz zum Training in der Simulation ist das Sammeln von Erfahrungen hier teurer als die Berechnung der Policy-Updates. Die Erfahrungen von mehreren Robotern werden deshalb in einem gemeinsamen Replay-Buffer gesammelt und Updates von einem zentralen Server berechnet. Die Aktualisierung der Policy auf den Robotern erfolgt jeweils zu Beginn einer Episode, sodass der Sammelvorgang nicht unterbrochen werden muss. Es gelingt ihnen die Aufgabe des Türöffnens in weniger als drei Stunden zu lernen. Gleichzeitig stellen sie fest, dass mit ihrem Ansatz die Trainingszeit nicht direkt mit der Anzahl der Roboter skalierbar ist.

Eine Steigerung der Trainingseffizienz auf realen Systemen ist auch Gegenstand der Arbeit von (Haarnoja; Zhou; Ha et al., 2018). Auf der Plattform *Minitaur* erproben sie durchgängiges RL, um einen Laufroboter für die Fortbewegung in der Ebene zu trainieren. Sie verwenden den in Abschnitt 2.2 vorgestellten SAC-Algorithmus und erweitern ihn um eine automatische Anpassung des Temperaturparameters für den Entropie-Term. In Zuständen, für die bereits gute Aktionen gefunden wurden, wird die Temperatur reduziert. Die Exploration wird dadurch auf Bereiche des Zustandsraums konzentriert, in denen der relative Wert gewählter Aktionen mit großer Unsicherheit behaftet ist. Damit steigt der Nutzwert gesammelter Erfahrungen. Ähnlich wie (Gu et al., 2016), generieren sie Erfahrungen mithilfe des Roboters und führen die Berechnungen auf separaten Rechnern aus. Innerhalb von etwa zwei Stunden erreichen sie in 400 Trainingsepisoden eine stabile Fortbewegung. Eine verbleibende Schwierigkeit ist die Notwendigkeit des menschlichen Eingriffs. Wenn der Roboter fällt oder den Rand des Trainingsgeländes erreicht, muss er manuell zurückgesetzt werden.

Um das Problem der Dimensionalität zu reduzieren, verwenden (Heess; Wayne et al., 2016) einen Ansatz, der in Teilen vergleichbar ist mit der Arbeit von (Bledt et al., 2018) und (Johnson et al., 2011) aus Abschnitt 2.1. Sie führen eine aus der Neurobiologie inspirierte hierarchische Steuerungsstruktur ein. Die Wahrnehmung der Umwelt wird aufgeteilt in Signale der Selbstwahrnehmung (Propriozeption) und äußere Reize (Exterozeption). Eine übergeordnete Steuerung hat Zugang zu exterozeptiven Sensoren,

die Untergeordnete erhält nur propriozeptive Signale. In einem Vortraining lernt die untergeordnete Steuerung primitive Bewegungsmuster. Im Anschluss wird die übergeordnete Steuerung trainiert, eine Aufgabe, entsprechend ihrer Außenwahrnehmung, durch Stimulation der Bewegungsmuster in der untergeordneten Steuerung zu lösen.

(Yang et al., 2019) präsentieren einen hybriden Ansatz von RL und klassischer Steuerung. Ein neuronales Netz lernt ein Modell der Dynamik von Minitaur und plant die nächsten Schritte unter Berücksichtigung der Belohnungsfunktion. Das Modell ermittelt dabei nur den nächsten angestrebten Zustand. Die notwendigen Steuerungsbefehle werden durch eine Trajektorienplanung generiert. Mit dem Lernprozess am realen Roboter überwinden sie den Reality-Gap. Die hybride Lösung reduziert die Dimension des zu lernenden Problems, sodass in wenigen Versuchen stabile Bewegungsmuster entwickelt werden. Besonderes Merkmal des Modell-basierten Ansatzes, ist die Wiederverwendbarkeit des Modells bei einem Aufgabenwechsel. Während Modell-freie Techniken spezifisch für eine Aufgabe trainiert werden, kann das Dynamikmodell für mehrere Probleme verwendet werden (Yang et al., 2019).

Entgegen dem durchgängigen RL, wie es von (Heess; Wayne et al., 2016) und (Haarnoja; Zhou; Ha et al., 2018) praktiziert wird, zeigen (Kohl et al., 2004) mit ihrer abstrakten Version, wie RL benutzt werden kann, um das Potential klassischer Steuerungen besser auszunutzen. Sie beschäftigen sich mit der Konfiguration von Aibo im Rahmen der Roboter Fußball-Weltmeisterschaft. Die Gangart des Roboters wird von zwölf Parametern bestimmt, die in der Regel manuell festgelegt werden. Ein wesentliches Ziel der Parameterbestimmung ist die Maximierung der Höchstgeschwindigkeit, um einen Wettbewerbsvorteil zu erhalten. Mithilfe ihres Algorithmus finden sie eine Parameterauswahl, die einen schnelleren Gang erzeugt, als alle Mitbewerber zu dieser Zeit.

Den wahrscheinlich fortschrittlichsten und erfolgversprechendsten Ansatz für RL mit realen Laufrobotern präsentieren (Hwangbo et al., 2019) auf ANYmal. Ausgangspunkt ist ein Starrkörpermodell des Roboters. Aspekte wie Motordynamik, Eigendämpfung und Verzögerungen in den Steuersignalen, die den Reality-Gap ausmachen, sind darin nicht enthalten. Stattdessen trainieren sie diese Eigenschaften per *self-supervised learning* (Selbstüberwachtes Lernen) in einem neuronalen Netz, mit Sensordaten des echten Roboters. Das Modell und die Imperfektionen werden in der Simulation kombiniert, um mit RL eine Policy zu trainieren.

Im Gegensatz zu anderen Arbeiten erhalten sie so die vollkommene Flexibilität für den Lernprozess, die RL für komplexe Probleme attraktiv macht. Soweit bekannt, sind sie die ersten, denen die direkte Übertragung einer in der Simulation gelernten Policy auf den realen Roboter gelingt. Anstelle des Trainings am realen System schließen sie den

Reality-Gap durch maschinelles Lernen in der Simulation. Begünstigt durch fortschrittliche Software benötigen sie für einen Trainingslauf etwa elf Stunden und erreichen damit praxistaugliche Rechenzeiten.

2.4. Gestaltung von Belohnungsfunktionen

Die Wahl der richtigen Belohnungsfunktion für RL gewinnt zunehmend an Bedeutung, je komplexer die Probleme sind, die mit RL zu bearbeiten versucht werden. Für niedrigdimensionale Aufgaben wie Grid World reicht oftmals die einmalige Belohnung bei Erreichen des Zielfeldes aus, um den Trainingserfolg zu sichern. Die Wahrscheinlichkeit in hochdimensionalen Räumen zufällig die Lösung für eine Aufgabe zu finden ist dagegen ausgesprochen gering. Entlehnt aus der Psychologie ist daraus der Begriff des *Reward-Shaping* entstanden. Er beschreibt die Praxis, Belohnungsfunktionen so zu gestalten, dass nicht nur der erfolgreiche Abschluss einer Aufgabe belohnt wird, sondern auch die Leistung auf dem Weg dorthin bewertet wird. Dem Lernenden wird damit eine Orientierung in Richtung der Lösung vermittelt. Dieser Ansatz wird in einer Reihe von theoretischen Arbeiten aufgegriffen und weiterentwickelt.

Mit dem *Potential-based reward shaping* erweitern (Ng et al., 1999) den Grundgedanken um eine wichtige Eigenschaft. Demnach sollen Veränderungen einer Belohnungsfunktion im Sinne des Reward-Shaping nicht die Eigenschaften der optimalen Lösung beeinflussen. Sie soll dieselbe sein, die auch ohne die Anwendung von Reward-Shaping optimal ist. Während hier eine Bewertung der durchlaufenen Zustände vorgenommen wird, erweitern (Wiewiora et al., 2003) den Ansatz auch auf die gewählten Aktionen einer Policy, um den Leiteffekt weiter zu verstärken. Darauf aufbauend versuchen (Badnava et al., 2019) die gesammelten Erfahrungen und Belohnungen effizienter zu nutzen, indem sie das Gesamtergebnis mehrerer Episoden miteinander vergleichen. Der Agent soll dadurch verstärkt in Richtung der Episoden trainiert werden, in denen relativ gute Ergebnisse erzielt wurden. Alle drei Arbeiten erproben ihre Methoden anhand von Grid World mit unterschiedlichem Erfolg. Reward-Shaping im Kontext von kontinuierlichen Zustandsräumen, wie sie bei Robotern auftreten, erproben (Zou et al., 2019). Sie präsentieren einen Algorithmus, der für unterschiedliche Aufgaben innerhalb des gleichen Zustandsraums automatisches Reward-Shaping durchführt und demonstrieren die Wirksamkeit anhand der *Cart-Pole*-Aufgabe.

Insbesondere die Arbeit von (Zou et al., 2019) birgt einen vielversprechenden Ansatz, um effektive Belohnungsfunktionen zu finden. Nach Kenntnis des Autors werden die genannten Methoden selten in der Praxis angewendet. Belohnungsfunktionen werden stattdessen intuitiv und manuell gestaltet.

Einen Paradigmenwechsel in der Gestaltung von Belohnungsfunktionen propagieren (Heess; T.B. et al., 2017). Sie argumentieren, dass ein komplexes Verhalten nicht nur durch eine aufwendig gestaltete Belohnungsfunktion erreicht werden kann, sondern mit schlichten Funktionen und dem Training in vielfältiger, anspruchsvoller Umgebung. Der Mehrwert liegt in der größeren Freiheit des Agenten, der weniger Restriktion durch die Belohnungsfunktion erfährt. Die gelernte Policy soll so robuster und variantenreicher werden. In einer simulierten Umgebung mit zufällig generierten Strukturen trainieren sie Läufer mit verschiedenen Kinematiken und erreichen eine Vielzahl stabiler Bewegungsformen. Nachteil dieses Ansatzes ist der erhöhte Trainingsaufwand, der mit der gesteigerten Flexibilität einhergeht.

Einen Gegenentwurf zu RL mit Belohnungsfunktionen bildet *Inverse-Reinforcement-Learning*. Dabei wird davon ausgegangen, dass es schwieriger ist, ein komplexes Verhalten formal zu beschreiben als es vorzuführen. Der Agent lernt eine Demonstration zu imitieren und leitet daraus eine Belohnungsfunktion ab, für die dieses Verhalten optimal ist. Noch einen Schritt weiter gehen (Daniel et al., 2014) und betrachten in ihrer Arbeit Fälle, in denen die richtige Belohnungsfunktion nicht offensichtlich und auch die Demonstration durch einen Experten schwierig ist. Anstatt aus einer Vielzahl von Demonstrationen zu lernen, nutzen sie die menschliche Fähigkeit, einer Aktion einen relativen Wert zuzuordnen. In Experimenten mit einem Roboterarm lernt der Agent eine Policy zum Greifen eines Objekts. Als Rückmeldung zu seinen Versuchen erhält der Agent einen Bewertung durch den Experten und passt seine Belohnungsfunktion daran an.

3. Grundlagen

Mit dem Einsatz von RL für die Steuerung von Laufrobotern können diverse Elemente des klassischen Steuerungsentwurfs, wie Trajektorienplanung, Bewegungskoordination und Parametersuche, vom menschlichen Entwickler auf den Computer übertragen werden. Wie in Kapitel 2 angemerkt, steht RL grundsätzlich nicht im unmittelbaren Zusammenhang mit der Robotik, sondern stellt ein Rahmenkonzept für das automatisierte Lernen maschineller Entscheidungsfindung dar. Die zugrundeliegenden Ideen und Methoden sind allgemein formuliert und in ihrer Anwendbarkeit nicht auf die Robotik beschränkt. Diese Grundlagen werden im folgenden Abschnitt betrachtet. Im Zuge der Forschung wächst die Anzahl der Methoden und deren Varietäten stetig an, sodass kein Anspruch auf Vollständigkeit besteht. Der Fokus liegt auf den Konzepten, die in dieser Arbeit zur Anwendung kommen. Eine Einführung in die grundlegenden Methoden geben (Sutton et al., 2018). Weiterentwicklungen sind Schwerpunkt in (Wiering et al., 2012).

Im zweiten Abschnitt werden künstliche neuronale Netze behandelt. Ähnlich wie RL kein unbedingter Bestandteil der Robotik ist, sind neuronale Netze nicht zwingend erforderlich für die Umsetzung von RL. Sie besitzen jedoch diesbezüglich günstige Eigenschaften und werden deshalb häufig in Verbindung mit RL eingesetzt. Ihre Funktionsweise wird ebenfalls im Kontext ihrer Verwendung in dieser Arbeit beleuchtet.

3.1. Reinforcement Learning

Der Begriff des *Reinforcement* ist der Verhaltenspsychologie entlehnt und beschreibt die Förderung oder Unterdrückung eines bestimmten Verhaltens durch Belohnung oder Bestrafung. Ein Mechanismus, der bei menschlichen oder tierischen Lernprozessen zu beobachten ist. Der Ursprung des *Reinforcement Learning*-Ausdrucks im Zusammenhang mit maschinellen Lernprozessen wird Marvin Minsky in seiner Dissertation (Minsky, 1954) zugesprochen (Sutton et al., 2018, S. 20).

RL ist eine Teildisziplin des maschinellen Lernens (Abbildung 3.1). Als einer von drei Bereichen kann es abgegrenzt werden von *Supervised-Learning* (überwachtem Lernen)

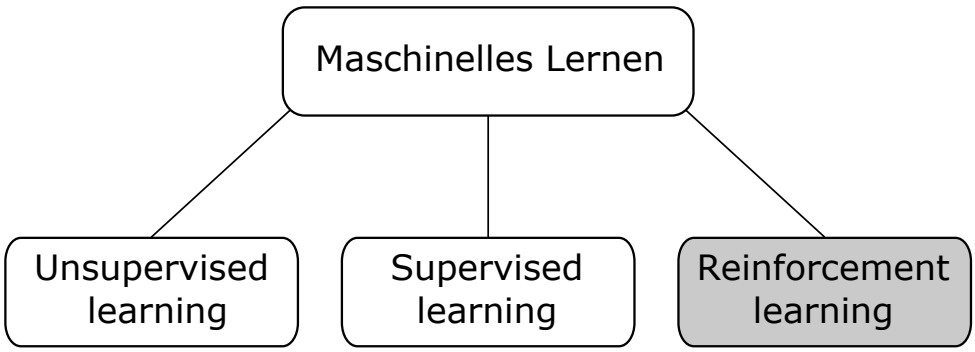

Abbildung 3.1.: Teildisziplinen maschinellen Lernens.

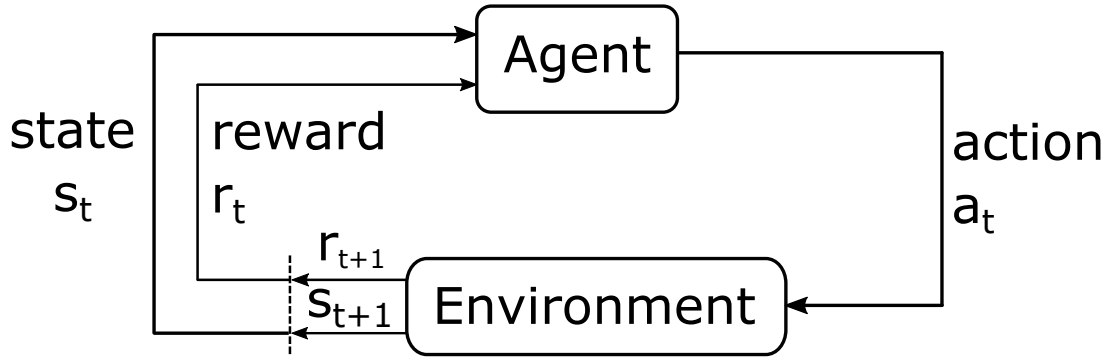

Abbildung 3.2.: Markov Entscheidungsprozess (in Anlehnung an (Sutton et al., 2018)).

und *Unsupervised-Learning* (unüberwachtem Lernen), wobei auch hybride Formen mit Eigenschaften aus mehreren Bereichen existieren. Sowohl Supervised-Learning als auch Unsupervised-Learning operieren mit vorgegebenen Datenmengen. In ersterem Fall werden Assoziationen zwischen Eingangssignalen und einer gewünschten Ausgabe gelernt, wie etwa die Klassifizierung von Hunde- und Katzenbildern auf Basis von Bildpunkten. Die Information über die angestrebte, richtige Ausgabe, ist anhand eines *Label* vorher bekannt und wird als Rückmeldung verwendet. Unüberwachtes Lernen verarbeitet ungekennzeichnete Daten und identifiziert Klassen, Muster oder Zusammenhänge darin, ohne dabei einem vorgegebenen Ziel unterworfen zu sein. Im Gegensatz dazu, sammelt RL Daten in Form von Erfahrungen erst durch Interaktion mit einer Umwelt. Der lernende Agent optimiert sein Verhalten in einem Trial-and-Error-Prozess unter Berücksichtigung einer erhaltenen Belohnung (engl. *reward*). Ein wesentlicher Unterschied besteht darin, dass der RL-Agent die Umwelt durch sein Handeln beeinflusst.

3.1.1. Markov Entscheidungsprozess

Formal kann das RL-Problem in der Regel als Markov-Entscheidungsprozess dargestellt werden (Abbildung 3.2), einem Modell für die sequentielle Entscheidungsfindung. Der Agent befindet sich zum Zeitpunkt t in einem Zustand (*state*) s und führt eine Aktion (*action*) a_t aus. Durch die dynamischen Eigenschaften der Umgebung (*Environment*), gelangt er damit in einen neuen Zustand s_{t+1} und erhält dafür eine Belohnung r_{t+1}. Der Begriff Environment beschreibt dabei nicht zwingend die räumliche Umgebung des Agenten, sondern das gesamte System mit dem er interagiert. Im Beispiel eines Roboters schließt dies unter anderem auch seine eigene Dynamik mit ein. Dieser Vorgang wird wiederholt, sodass eine Trajektorie:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots \tag{3.1}$$

entsteht. Die Dynamik dieses Prozesses kann als Wahrscheinlichkeitsverteilung dargestellt werden. Sie beschreibt, mit welcher Wahrscheinlichkeit der Agent, ausgehend von einem Zustand s und unter Ausführung der Aktion a, einen nächsten Zustand s' erreicht, wofür er eine Belohnung r erhält:

$$p(s', r \mid s, a) \doteq Pr\{s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a\} \tag{3.2}$$

Diese Verteilung hat die Eigenschaft

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \forall\, s \in S, a \in A(s), \tag{3.3}$$

wonach die Wahrscheinlichkeit des Auftretens eines Zustands s' und der Belohnung r nur vom Ausgangszustand s und der gewählten Aktion a abhängt. S bezeichnet den Zustandsraum und $A(s)$ die Menge aller möglichen Aktionen im Zustand s. Hieraus ergibt sich die formale Forderung an die Beschreibung des Zustands s, dass sie alle Informationen der zeitlichen Vergangenheit beinhalten muss, die relevant sind um eine Aussage über das Eintreten des nächsten Zustands zu treffen. Ein Zustand, der diese Anforderung erfüllt, besitzt die Markov-Eigenschaft. Sie ist von wesentlicher Bedeutung für den RL-Prozess, weil sie dem lernenden Agenten die Möglichkeit gibt, zielgerichtete Entscheidungen auf Basis seines aktuellen Zustands zu treffen.

Das langfristige Ziel des Agenten ist es, die Summe der erhaltenen Belohnungen, über eine Serie von Zeitschritten, durch seine Entscheidungen zu maximieren. Die Summe der erhaltenen Belohnungen kann dargestellt werden als der Ertrag G_t:

$$G_t \doteq r_{t+a} + r_{t+2} + r_{t+3} + \cdots + r_T \tag{3.4}$$

Die Definition des Ertrags nach Gleichung 3.4 setzt eine endliche Entscheidungssequenz mit der letzten Belohnung im Zeitschritt T voraus. Bei kontinuierlichen Aufgaben, in denen kein definierter Endzustand erreicht wird, können die Sequenzen theoretisch unendlich lang und der Ertrag entsprechend unendlich groß werden. Um dies zu vermeiden, wird der Ertrag, ähnlich einer Abzinsung in der Finanzmathematik, als Summe der diskontierten Belohnungen definiert:

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.5}$$

Der Diskontierungsfaktor (*discount factor*) γ, mit $0 \leq \gamma \leq 1$, führt für $\gamma < 1$ zu einem endlichen Ertrag. Neben dem mathematischen Vorteil nicht mit dem Unendlichkeitsbegriff umgehen zu müssen, kann der Planungshorizont des Agenten mit γ beeinflusst werden. Praktisch betrachtet, legt ein Agent mit $\gamma = 0{,}1$ mehr Wert auf die Maximierung kurzfristiger Belohnungen. Mit einem Diskontierungsfaktor nahe Eins werden langfristig erwartete Belohnungen höher bewertet.

Der Ertrag als Summe der Sequenz diskontierter Belohnungen kann in rekursiver Form ausgedrückt werden, als die Summe der unmittelbaren Belohnung und dem diskontierten, zukünftig erwarteten Ertrag:

$$\begin{aligned} G_t &\doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \\ &= r_{t+1} + \gamma G_{t+1} \end{aligned} \tag{3.6}$$

3.1.2. Value-Function und Policy

Damit ist die Form der Interaktion des Agenten mit der Environment und deren Ziel definiert. Um eine zielorientierte Entscheidung treffen zu können, benötigt der Agent noch einen Maßstab dafür, welcher Ertrag bei einer Aktion zu erwarten ist. Hierfür kann jedem möglichen Zustand ein Wert zugeordnet werden. Dieser ist nicht zuletzt auch davon abhängig, ob der Agent den Zustand tatsächlich erreichen wird.

Welche Zustände der Agent erreicht, bestimmt sein Verhalten bei der Auswahl seiner Aktionen, der Policy. Sie wird notiert als $\pi(a|s)$ und beschreibt die Wahrscheinlichkeit, dass in einem Zustand s eine Aktion a ausgewählt wird. Der Wert eines Zustands, bei gegebener Policy π, kann demnach definiert werden als der zu erwartende Ertrag, wenn der Agent in diesem Zustand startet und in der Folge Entscheidungen entsprechend der

Policy trifft (Sutton et al., 2018, S. 58). Er wird ausgedrückt durch die *Value-Function*:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid s_t = s] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right], \forall\, s_t = S \tag{3.7}$$

Analog dazu beschreibt die *Action-Value-Function* den Wert einer Aktion a im Zustand s, unter der Policy π, als den zu erwartenden Ertrag, wenn a in s ausgeführt wird und nachfolgend π angewendet wird:

$$q_\pi(s,a) \;\doteq\; \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{3.8}$$

Mit der rekursiven Formulierung des Ertrags aus Gleichung 3.6, kann der Wert eines Zustands ausgedrückt werden durch die unmittelbare Belohnung und den zu erwartenden Ertrag nachfolgender Zustände (Gleichung 3.9). Dies ist die *Bellman*-Gleichung der Value-Function:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \forall s \in S \end{aligned} \tag{3.9}$$

Analog dazu ist die Bellman-Gleichung der Action-Value-Function:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \\ &= \sum_{s'} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(s'|a') q_\pi(s', a')\right] \end{aligned} \tag{3.10}$$

Die Value-Function, respektive Action-Value-Function, können aus den gesammelten Erfahrungen des Agenten gelernt werden und sind in einfachen Fällen durch eine Wertetabelle darstellbar. Eine Policy zur Maximierung des Ertrags kann direkt daraus abgeleitet werden, indem jedem Zustand die Aktion mit dem höchsten zu erwartenden Ertrag zugewiesen wird. Sie wird als *Greedy-Policy* bezeichnet.

Der Vorteil der Action-Value-Function besteht darin, dass sie die Greedy-Policy bereits implizit enthält, wenn in jedem Zustand die Aktion mit dem höchsten Wert ausgewählt wird. Die Policy muss dann nicht mehr explizit gelernt werden. Im Umkehrschluss müssen für die Action-Value-Function Werte für alle möglichen Aktionen $A(s)$ gelernt werden. In hochdimensionalen Aktionsräumen ist dies ein wesentlicher Nachteil gegenüber der Value-Function.

Wie zu Beginn von Abschnitt 3.1 erwähnt, ist der Agent im RL darauf angewiesen, aus den Erfahrungen zu lernen, die er während des Trainings sammelt. Wählt der Agent seine Aktionen nach einer Greedy-Policy aus, wird er stets die Erfahrungen entlang der Trajektorie der Zustände mit den höchsten Werten generieren. Er nutzt dabei sein momentanes Wissen über die Value-Function um den Ertrag zu maximieren. Dieses Verhalten wird als *Exploitation* bezeichnet. Allerdings ist der Wert jedes Zustands zu diesem Zeitpunkt mit Unsicherheit behaftet, sodass eine Greedy-Policy auch suboptimale Aktionen auswählt. Um diese Unsicherheit zu verringern und den tatsächlichen Wert eines Zustands zu ermitteln, muss der Agent den Zustandsraum erkunden, *Exploration* betreiben. Dies stellt ein Problem für die Entscheidungsfindung dar, weil weder Exploration noch Exploitation in ausschließlicher Form betrieben werden kann, ohne in der Aufgabe zu scheitern (Sutton et al., 2018, S. 3).

Eine Balance zwischen beiden Strategien kann gefunden werden, indem der Agent mit einer gewissen Häufigkeit die jeweils beste Aktion auswählt und gelegentlich solche Aktionen ausführt, die ihn zu scheinbar schlechteren Zuständen bringen. Ein einfacher Weg dieses Verhalten umzusetzen sind ϵ-Greedy-Policies. Der Parameter ϵ bezeichnet dabei die Wahrscheinlichkeit, in einem Zustand eine suboptimale Aktion auszuwählen, sodass mit einer Wahrscheinlichkeit von $(1 - \epsilon)$ die Beste vorgezogen wird. Mit einer ausreichend großen Menge von Entscheidungen kann der Agent auf diesem Weg die beste Policy finden. Einige Alternativen zur ϵ-Greedy-Strategie sind bereits in Kapitel 2 beschrieben.

Eine Policy π ist dann als besser gegenüber einer anderen Policy π' einzustufen, wenn ihr zu erwartender Ertrag für alle Zustände größer oder zumindest gleichwertig ist. Dies ist gegeben, wenn $v_\pi(s) \geq v_{\pi'}(s)$ für alle $s \in S$ (Sutton et al., 2018, S. 62). Die zur optimalen Policy $\pi*$ gehörige optimale Value-Function, beziehungsweise Action-Value-Function ist damit definiert als:

$$v_*(s) \doteq \max_\pi v_\pi(s), \forall\, s \in S, \tag{3.11}$$

beziehungsweise

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a), \forall\, s \in S, a \in A(s). \tag{3.12}$$

Es existiert eine Reihe von Algorithmen, die zur Bestimmung optimaler Value-Functions, sowie dazugehöriger Policies geeignet sind und das bisher dargestellte RL-Problem lösen können. (Sutton et al., 2018) teilen die Methoden in die Kategorien:

- *Dynamic-Programming*,
- *Monte-Carlo-Methoden*
- und *Temporal-Difference-Learning*,

ein. Diese Methoden sind für RL insofern von grundlegender Bedeutung, als dass ihre Ansätze und Ideen in weiterentwickelten Techniken wieder aufgegriffen werden. Charakteristisches Merkmal von Algorithmen des Dynamic-Programming ist die Notwendigkeit eines perfekten Modells über die Dynamik der Environment. Monte-Carlo-Methoden sind ein Gegenentwurf dazu und verzichten auf die Bereitstellung vollständiger Information durch ein Modell. Stattdessen finden und verbessern sie Policies anhand von erfahrenen *Samples* im Zustandsraum. Temporal-Difference-Learning kann als hybrider Ansatz aus den beiden Erstgenannten betrachtet werden. Alle drei Kategorien beschreiben jedoch Methoden, die nur auf relativ niederdimensionale und insbesondere diskontinuierliche Zustandsräume praktisch anwendbar sind (Sutton et al., 2018).

3.1.3. Näherungsfunktionen

Um für praktische Probleme mit hochdimensionalen oder, wie im Beispiel des Roboters, kontinuierlichen Zustandsräumen Value-Functions ermitteln zu können, werden Näherungsfunktionen eingesetzt. Die tatsächliche Value-Function wird angenähert, durch eine parametrisierte Funktion mit dem Parametervektor w:

$$\hat{v}(s, w) \approx v_\pi(s) \tag{3.13}$$

Die Näherung besteht in der geringeren Dimensionalität d des Vektors w gegenüber dem Zustandsraum S ($d \ll S$). Die Näherungsfunktion kann nicht den exakten Wert der Value-Function für jeden Zustand abbilden, sodass eine Generalisierung über verschiedene Zustände stattfindet.

Eine Möglichkeit, die Werte der Näherungsfunktion an die tatsächliche Value-Function anzunähern, ist die Anwendung von Gradientenverfahren. Angenommen $J(w)$ ist eine differenzierbare Funktion, dann ist ihr Gradient beschrieben durch den Vektor der

partiellen Ableitungen bezüglich ihrer Parameter:

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix} \tag{3.14}$$

Die Regel zur Anpassung der Parameter kann gegeben sein als:

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w) \tag{3.15}$$

mit der Schrittweite α. Die Suche nach geeigneten Parametern für die Näherungsfunktion kann ausgedrückt werden, durch die Minimierung der *Loss-Function*, in diesem Fall der mittleren quadratischen Abweichung zwischen den Werten der Näherungsfunktion und der Value-Function):

$$J(w) = \mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}(s, w))^2 \right] \tag{3.16}$$

Die Anpassung der Parameter ist dann:

$$\Delta w = \alpha \mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}(s, w)) \nabla_w \hat{v}(s, w) \right] \tag{3.17}$$

wobei der Erwartungswert ausgedrückt werden kann durch Samples:

$$\Delta w \approx \alpha (v_\pi(s) - \hat{v}(s, w)) \nabla_w \hat{v}(s, w) \tag{3.18}$$

Die tatsächliche Value-Function $v_\pi(s)$ in den Gleichungen 3.16-3.18 ist immer noch unbekannt, sodass sie durch einen geeigneten Schätzwert ersetzt werden muss. Zwei mögliche Substitute sind in den Gleichungen 3.19-3.20 dargestellt:

$$\Delta w = \alpha (G_t - \hat{v}(s_t, w)) \nabla_w \hat{v}(s_t, w) \tag{3.19}$$

$$\Delta w = \alpha (r_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)) \nabla_w \hat{v}(s_t, w) \tag{3.20}$$

In Gleichung 3.19 ist der Wert der Value-Function, in Anlehnung an den Zielwert in Monte-Carlo-Methoden, ersetzt durch den effektiven Ertrag einer Sequenz von Zuständen. Gleichung 3.20 ist angelehnt an Temporal-Difference-Learning und definiert den Zielwert als die Summe der unmittelbaren Belohnung mit dem diskontierten Wert der angenäherten Value-Function für alle nachfolgenden Zustände. Dies ist nur eine Möglichkeit zur Approximation der tatsächlichen Value-Function. Die Menge der Näherungs- und Optimierungsverfahren ist ungleich größer und wird von (Sutton et al., 2018) vertieft behandelt.

3.1.4. Batch Methoden

Die im vorherigen Abschnitt beschriebenen Methoden verbessern die Näherung der Value-Function iterativ, mit den Erfahrungen des letzten Zeitschritts oder der letzten Trajektorie. Ein charakteristischer Makel liegt in der Effizienz dieser Methoden. Jede Erfahrung in Form der Belohnung oder des Ertrags wird nur einmalig als Zielwert für eine Anpassung der Parameter benutzt und verfällt anschließend. Dabei wird nicht berücksichtigt, wie gut die neuen Parameter tatsächlich mit den Zielwerten übereinstimmen. Um die Effizienz bezüglich der Erfahrungen zu steigern, werden Batch-Methoden in Verbindung mit Experience-Replay angewendet.

Erfahrungen, die der Agent während des Trainings sammelt, werden in einem Speicher D, dem Replay-Buffer, abgelegt. Für jede Anpassung der Parameter wird eine Menge M der Erfahrungen in zufälliger Weise aus dem Replay-Buffer entnommen. Die Menge M wird als *Mini-Batch* bezeichnet, weil üblicherweise $(M \ll D)$. Ähnlich wie in Abschnitt 3.1.3, kann das Optimierungsziel beispielsweise die Minimierung der mittleren quadratischen Abweichung (Gleichung 3.21) oder auch der Residuenquadratsumme (Gleichung 3.22) sein:

$$J(w) = \frac{1}{M} \sum_{i=1}^{M} (v_i^{\pi} - \hat{v}(s_i, w))^2 \tag{3.21}$$

$$J(w) = \sum_{i=1}^{M} (v_i^{\pi} - \hat{v}(s_t, w))^2 \tag{3.22}$$

Jede Erfahrung kann dadurch zu mehreren Parameteranpassungen beitragen, wobei die Häufigkeit der Verwendung unter Anderem von der Mini-Batch-Größe und der Länge des Replay-Buffer abhängt. Ein weiterer Vorteil ist die Dekorellation zwischen den für die Anpassung verwendeten Erfahrungen und den zuletzt Gesammelten. Der Einfluss kürzlicher Erfahrungen auf die Richtung der Parameteranpassung bleibt damit begrenzt, was den Lernprozess stabilisiert.

3.1.5. Policy-Gradient

Die in Abschnitt 3.1.4 behandelten Methoden finden und verbessern eine Policy auf Basis der Value-Function. Anstelle der Ableitung einer Policy von der Value-Function, kann die Policy mit Policy-Gradient-Methoden auch direkt ermittelt werden. Diese besitzen in der Regel bessere Konvergenzeigenschaften und sind effektiver im Umgang mit hochdimensionalen oder kontinuierlichen Aktionsräumen (Silver, 2015).

Eine Policy π ist nun definiert als die Wahrscheinlichkeitsverteilung in Zuständen s eine Aktion a auszuwählen, in Abhängigkeit der Parameter θ:

$$\pi_\theta(s|a) = \mathbb{P}[a|s,\theta] \tag{3.23}$$

Die Zielfunktion $J(\theta)$ sei ein Maß für die Leistungsfähigkeit oder den Wert der Policy in einem Zustandsraum:

$$J(\theta) = v_{\pi\theta}(S) \tag{3.24}$$

Es ist das Ziel, die Parameter der Policy unter Maximierung ihrer Leistungsfähigkeit anzupassen, wobei die Leistungsfähigkeit an der erhaltenen Belohnung gemessen wird. Ähnlich wie für die Approximation der Value-Function, werden hierfür Gradientenverfahren angewendet. Der Gradient der Policy ist definiert als Vektor der partiellen Ableitungen der Zielfunktion bezüglich ihrer Parameter:

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix} \tag{3.25}$$

Nach dem *Policy-Gradient-Theorem* kann der Gradient abgeschätzt werden als:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi\theta}\left[\nabla_\theta \log \pi_\theta(s|a) q_{\pi\theta}(s,a)\right] \tag{3.26}$$

Das Theorem drückt aus, dass die Parameter der Policy in Richtung der Wahrscheinlichkeit anzupassen sind, mit der gute Aktionen häufiger ausgewählt werden und Schlechte seltener, um den Wert der Policy zu steigern. Die Parameteranpassung kann auch für die Policy beschrieben werden, durch Gradient und Schrittweite:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \tag{3.27}$$

3.1.6. Actor-Critic

Die direkte Umsetzung der Policy-Gradient-Methode ist in Algorithmus (1) durch den *REINFORCE*-Algorithmus gegeben (Williams, 1992). Für die Anpassung der Parameter wird die unbekannte Action-Value-Function angenähert durch den Ertrag v_t der zurückliegenden Episode:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t \tag{3.28}$$

Diese Näherung induziert eine hohe Varianz in die Richtung und Größenordnung der Parameteranpassung, wodurch das Training des Agenten erheblich verlangsamt werden kann. Die Klasse der *Actor-Critic*-Algorithmen versucht diesen Makel zu beheben.

Algorithmus 1 REINFORCE (Monte-Carlo policy gradient)

Initialise θ arbitrarily
for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
  for $t = 1$ to $T - 1$ **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
  end for
end for
return θ

In Actor-Critic-Algorithmen werden zwei Näherungsfunktionen verwendet, um sowohl die zu lernende Policy, als auch die dazugehörige Action-Value-Function zu repräsentieren. Sie kombinieren damit die Methoden aus Abschnitt 3.1.3 und 3.1.5. Die Policy wird durch den Actor dargestellt und wählt während des Lernprozesses die Aktionen aus. Der Critic repräsentiert die Action-Value-Function und bewertet (kritisiert) die gewählte Aktion des Actor. Die Parameter beider Näherungsfunktionen werden während des Trainings fortwährend verbessert. Der Critic reduziert so die Varianz in den Parameteranpassungen aus Gleichung 3.28 und beschleunigt den Lernfortschritt.

Für die Approximation der Action-Value-Function in Actor-Critic-Algorithmen existieren diverse Varianten, die diesem gemeinsamen Zweck dienen. Unter anderem:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi\theta} \left[\nabla_\theta \log \pi_\theta(s|a) v_t \right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi\theta} \left[\nabla_\theta \log \pi_\theta(s|a) q_w(s,a) \right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi\theta} \left[\nabla_\theta \log \pi_\theta(s|a) A_w(s,a) \right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi\theta} \left[\nabla_\theta \log \pi_\theta(s|a) \delta \right] && \text{Temporal Difference Actor-Critic}
\end{aligned}
$$

In diesem Kapitel sind die grundlegenden Konzepte und Methoden des RL dargestellt worden. Die Abschnitte 3.1.3 und 3.1.5 haben die Verwendung von parametrisierten Näherungsfunktionen für Policy und Value-Function, beziehungsweise Action-Value-Function thematisiert. Der Begriff des *Deep Reinforcement Learning* ist nicht formal definiert. Er wird benutzt, um die häufig verwendete Kombination von RL mit mehrschichtigen neuronalen Netzen als Näherungsfunktionen zu beschreiben (François-Lavet et al., 2018). Sie werden im folgenden Abschnitt näher behandelt.

3.2. Neuronale Netze

Neuronale Netze sind ein praktisch universeller Ansatz um Näherungsfunktionen zu repräsentieren, die Eingaben auf gewünschte Ausgaben abbilden. Wie im vorherigen Abschnitt beschrieben, können ihre Parameter gelernt werden, indem beispielsweise der Fehler ihrer Ausgabe minimiert oder eine Zielfunktion maximiert wird. Ihr wesentlicher Vorteil gegenüber Polynomen als Näherungsfunktion ist die bessere Generalisierungsfähigkeit.

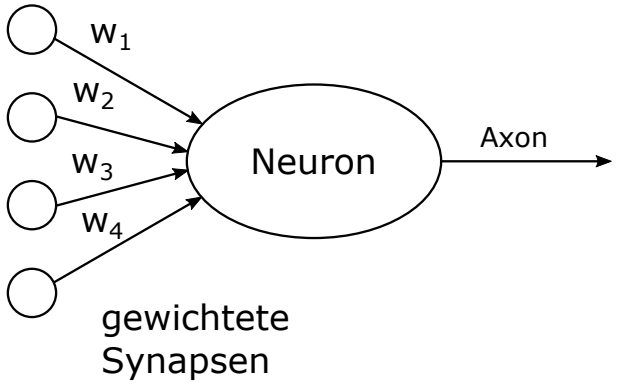

Abbildung 3.3.: Künstliches Modell eines Neurons (in Anlehnung an (Aggarwal, 2018)).

Aufbau und Funktionsweise neuronaler Netze sind inspiriert von den Kenntnissen über das Arbeitsprinzip von Nervenzellen in biologischen Organismen. Das menschliche oder tierische Nervensystem besteht aus einer Vielzahl Neuronen (Nervenzellen), die über Synapsen miteinander verbunden sind. Die Stärke der Verbindung zwischen zwei Neuronen, oder ganzen Regionen des Nervensystems, verändert sich in Abhängigkeit der Stimulation durch äußere Reize. Die erste technische Nachbildung dieses Mechanismus und Vorbild für neuronale Netze wie sie heute vielfach eingesetzt werden, ist das Perzeptron aus der Arbeit von (Rosenblatt, 1958). Es besteht aus einem einzelnen Neuron, dessen Eingangssignale mit Wichtungsfaktoren, den *Weights* versehen sind (Abbildung 3.3).

Der Wert eines Neurons wird beschrieben durch den Grad seiner Stimulation, die Aktivierung. Sie berechnet sich aus der Produktsumme der Eingangssignale x mit den Weights w und gegebenenfalls einem Bias-Faktor (3.4). Weights und Biases sind die Parameter der Näherungsfunktion, die im Zuge des Lernprozesses angepasst werden. Das Ausgangssignal y wird bestimmt durch Anwendung einer Aktivierungsfunktion σ auf die Aktivierung:

$$y = \sigma(w \cdot x) \tag{3.29}$$

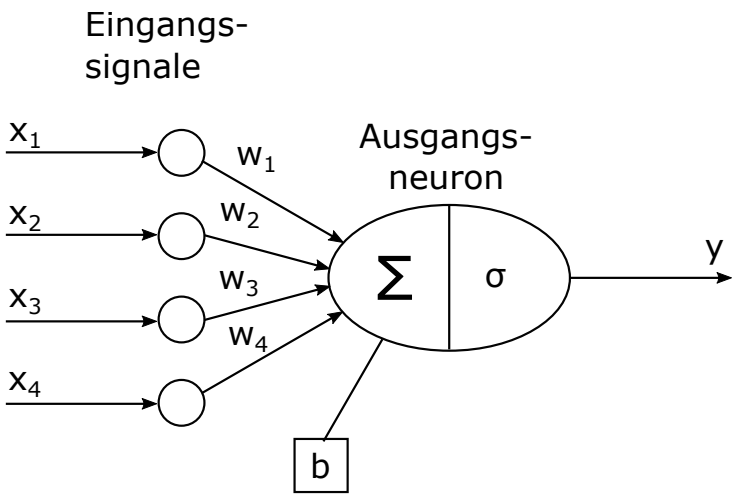

Abbildung 3.4.: Berechnung der Aktivität eines Neurons (in Anlehnung an (Aggarwal, 2018)).

Neuronale Netze werden in der Regel modular in Schichten mit einer Anzahl Neuronen pro Schicht aufgebaut. Mehrschichtige Netze bestehen aus je einer Ein- und Ausgangsschicht, sowie einer Anzahl verdeckter Zwischenschichten (engl. *hidden layers*). Neuronale Netze können anhand ihrer Architektur und ihres Signalflusses in Klassen eingeteilt werden. Drei häufig verwendete Arten sind:

- *Feedforward*-Netze,
- Rekurrente neuronale Netze (RNN),
- *Convolutional-Neural-Networks* (CNN).

In Abbildung 3.5 ist ein Feedforward-Netz mit zwei verdeckten Schichten dargestellt. Der Signalfluss ist unidirektional von der Eingangsschicht zu den Ausgangsneuronen. In rekurrenten Netzen werden Ausgangssignale von Neuronen teilweise zurückgeführt zu Eingängen von Neuronen derselben Schicht, einer vorherigen Schicht oder auf den Eingang des sendenden Neurons selbst. Eingesetzt werden sie in der Verarbeitung von Sprache, biologischer Daten und Datenreihen mit Korrelationen zwischen konsekutiven Zeitschritten. Convolutional-Neural-Networks verfügen über Filterfunktionen, um Werte einer Schicht auf einen Bereich der nächsten Schicht abzubilden. Sie finden häufig Verwendung in der Bildverarbeitung. Auf die Funktionsweise von RNN und CNN wird in dieser Arbeit nicht näher eingegangen. Details hierzu sind in (Aggarwal, 2018) gegeben.

3.2.1. Forward- und Backpropagation

Die Signalverarbeitung beim Training eines Feedforward-Netzes ist in zwei Phasen darstellbar. In der ersten Phase werden die Werte der Eingangsschicht entsprechend Gleichung 3.29 durch alle Schichten hindurch zu Ausgangssignalen verarbeitet. Wie in Abschnitt 3.1 dargestellt, wird für die Optimierung der Value-Function oder der Policy der Gradient der Zielfunktion bezüglich der Parameter der Näherungsfunktion verwendet. In der zweiten Phase wird deshalb der Gradient der Ausgabe, beziehungsweise der Zielfunktion bezüglich der Weights und Biases berechnet. Ausgangspunkt ist dabei das Ausgangsneuron, weswegen die Phase als *Backpropagation* bezeichnet wird. Die Gleichungen 3.30 und 3.31 skizzieren die Berechnung für das in Abbildung 3.6 dargestellte Beispiel mit zwei Neuronen und dem Gradienten der Loss-Function C:

$$\begin{aligned} C &= (a^{(L)} - y_{soll})^2 & \frac{\partial C}{\partial a^{(L)}} &= 2(a^{(L)}) \\ a^{(L)} &= \sigma(z^{(L)}) & \frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\ z^{(L)} &= w^{(L)} a^{(L-1)} + b^{(L)} & \frac{\partial z^{(L)}}{\partial w^{(L)}} &= a^{(L-1)} \end{aligned} \tag{3.30}$$

Der Gradient bezüglich des letzten Weights kann mithilfe der Kettenregel aufgelöst werden in die Reihe der Gradienten der beteiligten Komponenten:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} \tag{3.31}$$

Diese Reihe ist über alle Neuronen und Schichten des neuronalen Netzes fortsetzbar.

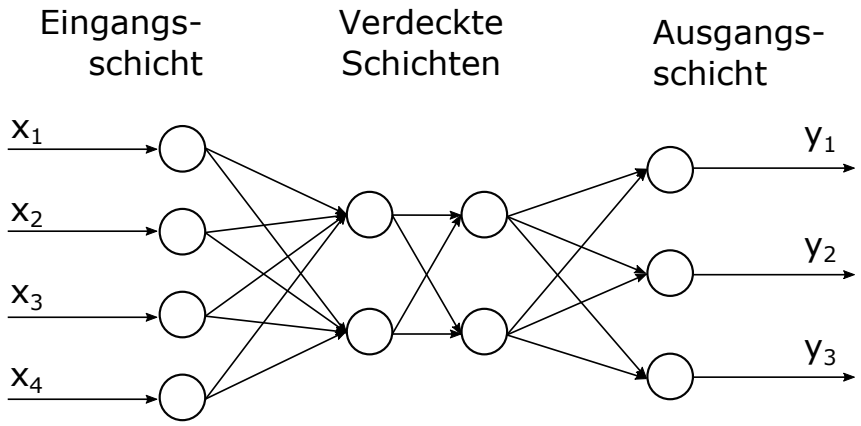

Abbildung 3.5.: Feedforward-Netz mit zwei verdeckten Schichten (in Anlehnung an (Aggarwal, 2018)).

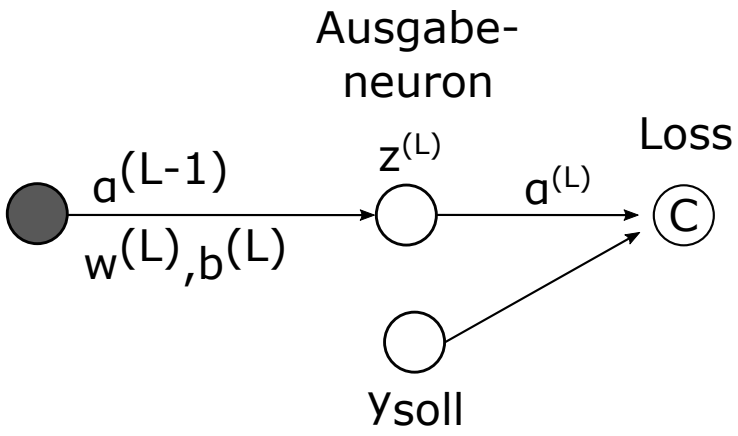

Abbildung 3.6.: Berechnung des Gradienten durch Backpropagation.

3.2.2. Praktische Schwierigkeiten des Trainings

Das Training neuronaler Netze birgt eine Reihe von Herausforderungen, wovon eine Auswahl hier betrachtet und Lösungsansätze besprochen werden.

Ein Problem, das nicht exklusiv bei neuronalen Netzen auftritt, aber auch hier berücksichtigt werden muss, ist das sogenannte *Overfitting*. Es kann insbesondere dann auftreten, wenn die Menge oder Vielfalt der Trainingsdaten zu klein oder die Approximationsfähigkeit des Netzes zu groß ist. Die gelernte Näherung bildet dann auch in den Trainingsdaten vorhandenes Rauschen im Detail ab und generalisiert in der Folge schlechter auf unbekannten Daten.

Eine mögliche Gegenmaßnahme ist Regularisierung. Dabei wird vom Betrag der Anpassung eines Parameters ein kleiner Anteil, bestimmt durch den Regularisierungsfaktor, abgezogen. Der neue Wert des Parameters stimmt in der Folge nicht in Gänze mit dem eigentlichen Zielwert für eine Anpassung überein und verhindert so eine zu scharfe Abbildung der Daten, die dieser Anpassung zugrunde liegen. Alternativ oder ergänzend dazu kann *Early-Stopping* angewendet werden. Prinzipiell wird hier das Training beendet, bevor die Parameter des Netzes die Trainingsdaten zu exakt annähern.

Insbesondere bei vielschichtigen, tiefen Netzen, besteht ein Risiko von sogenannten verschwindenden oder explodierenden Gradienten (engl. *vanishing / exploding gradients*). Durch die in Gleichung 3.31 dargestellte, multiplikative Verkettung von Gradienten, können diese in der Berechnung der Parameteranpassung exponentiell anwachsen (explodieren) oder zu sehr kleinen Werten abklingen (verschwinden). Das Training kann dadurch einerseits instabil und andererseits erheblich verlangsamt werden. Ob dieser Effekt eintritt hängt wesentlich von der gewählten Aktivierungsfunktion ab. Eine Auswahl ist in Abbildung 3.7 gegeben. Funktionen wie *tanh* oder *sigmoid* haben nur bei

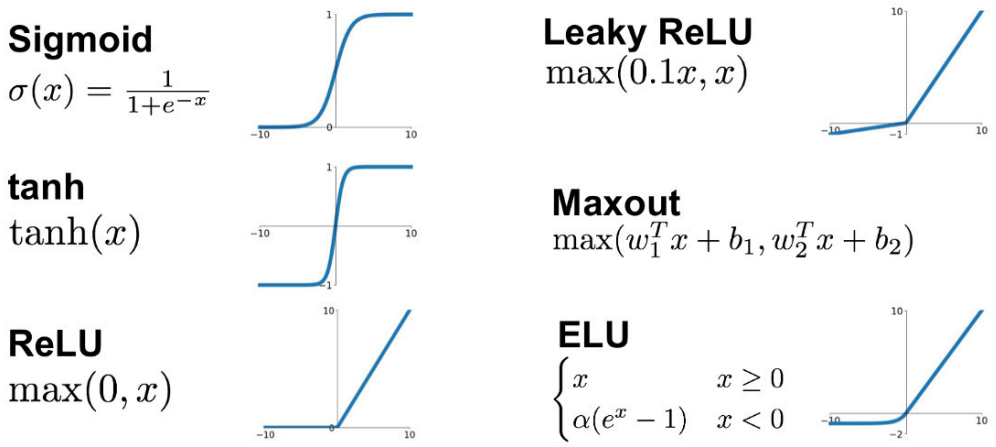

Abbildung 3.7.: Auswahl von Aktivierungsfunktionen (Pawan, 2019).

Aktivierungswerten nahe Null einen ausgeprägten Gradienten und erreichen abseits davon schnell einen Sättigungsbereich in denen er nahezu verschwindet.

Die *ReLU*-Funktion ist dagegen so definiert, dass der Gradient für positive Aktivierungen konstant Eins ist und für alle negativen Werte Null. Im positiven Bereich ist der Gradient damit stabil, wohingegen bei fortwährend negativen Aktivierungen das Problem des *Dying-ReLU* beziehungsweise der sterbenden Neuronen (*dying neurons*) auftreten kann. Im ungünstigsten Fall erfährt eine Reihe von Neuronen über das gesamte Training hinweg keine Gewichtsanpassung und trägt damit nicht zur Leistungsfähigkeit bei. Ein Versuch dieses Problem zu beheben sind die Varianten *Leaky-ReLU* und *Parameterized Leaky ReLU* (vgl. Kapitel 2), die auch bei negativer Aktivierung einen Gradienten besitzen.

Dem Problem der explodierenden Gradienten kann mit *Gradient-Clipping* begegnet werden. In der einfachsten Form wird der Gradient g bei überschreiten eines Schwellenwertes c auf diesen Wert begrenzt. Alternativ kann die Norm des Gradientenvektors herangezogen werden:

$$g \leftarrow c \cdot \frac{g}{||g||} \tag{3.32}$$

Bei der Verarbeitung von Eingangsdaten die in deutlich unterschiedlichen Größenordnungen vorliegen, sollten die Daten normalisiert werden. Andernfalls dominieren diese Signale die Aktivierung von Neuronen gegenüber kleineren Werten und verlangsamen die Optimierung.

Ein weiterer Aspekt, der die Stabilität des Trainings beeinflusst, ist die Initialisierung der Parameters eines neuronalen Netzes. Sie beeinflusst die Aktivierung der Neuronen, ebenso wie die Entwicklung der Gradienten über mehrere Schichten hinweg. Ein gängiges Verfahren ist die Ziehung von Zufallszahlen aus einer Verteilung, in Abhängigkeit der Neuronenanzahl von zwei verbundenen Schichten (vgl. Abschnitt 2.2).

4. Entwicklung der Trainingsumgebung

Bisherige Arbeiten die RL im Kontext der Robotik anwenden präsentieren vielversprechende Ergebnisse. Gleichzeitig wird darin immer wieder die Sensibilität der Erfolge gegenüber Änderungen in den Randbedingungen betont. Die Aufmerksamkeit dieses Kapitels ist deshalb auf die Entwicklung der Trainingsbedingungen gerichtet. Dies betrifft einerseits den Anteil der Simulation des Roboters in einer Trainingsumgebung und andererseits die Ausgestaltung des RL.

4.1. Modell des Roboters

Der Umgang mit den Eigenschaften des Robotermodells ist integraler Bestandteil der RL-Aufgabe. Um den praktischen Nutzen der verschiedenen Belohnungsfunktionen bewerten zu können, ist es deshalb erforderlich, das Modell anhand praxisnaher Anforderungen zu gestalten. Seine wesentlichen Merkmale müssen die Eigenschaften eines realen Roboters widerspiegeln. Umgekehrt formuliert erscheint es wenig sinnvoll, das Modell in einer Art und Weise zu abstrahieren, die ausschließlich auf eine Optimierung der Simulationsgeschwindigkeit abzielt. Gewonnene Erkenntnisse bezüglich der Wirkung einer Belohnungsfunktion wären dann mit hoher Wahrscheinlichkeit nicht auf praktische Anwendungen übertragbar.

Zu den Anforderungen, die der hier zu idealisierende, reale Roboter erfüllen sollte zählen:

1. die Fähigkeit, sich in natürlicher, insbesondere jedoch antropogener Umgebung fortzubewegen,

2. die Möglichkeit, einem tatsächlichen Zweck zu dienen, der über die reine Fortbewegung hinaus geht,

3. die Einsetzbarkeit in der Gegenwart von Menschen.

Roboter	Gewicht [kg]	Nutzlast [kg]	Länge [mm]	Breite [mm]	Höhe [mm]
ANYmal	50	10	1050	500	830
Spot	25	14	n.a.	n.a.	840
BigDog	113,4	150	1000	300	1250
HyQ	80	n.a.	1000	500	980
Titan-XIII	5,7	5	213,4	558,4	340
StarlETH	25	20	710	640	580
Cheetah 3	45	n.a.	600	256	800
Stoch	3,7	n.a.	400	270	320
Cheetah-cub	1,1	n.a.	205	100	158
Oncilla	5,1	n.a.	400	250	180

Tabelle 4.1.: Maße und Gewichte der Roboter

Eine ausführliche Designstudie zur möglichen Umsetzung dieser Kriterien ist nicht Bestandteil der vorliegenden Arbeit. Stattdessen wird auf bereits existierende Systeme zurückgegriffen. Einige der nachfolgend genannten wurden schon in Abschnitt 2.1 vorgestellt:

- ANYmal
- Spot
- BigDog
- HyQ
- Titan-XIII
- StarlETH
- Cheetah 3
- Stoch
- Oncilla
- Cheetah-cub

In ihrer Eigenschaft als Forschungsplattformen verfügen Stoch, Oncilla und Cheetah-cub nicht über eine Nutzlast und scheiden bezüglich des zweiten Kriteriums als Vorbild aus. Titan-XIII erlaubt eine Zuladung von 5 kg, ist aber mit einer Höhe von 340 mm zu klein für viele Umgebungen. StarlETH hat bei einem Eigengewicht von 25 kg eine beachtliche Nutzlast von weiteren 20 kg, benötigt jedoch in der vorgestellten Version eine externe Energieversorgung. Selbiges trifft auf HyQ zu. Drei der vier verbleibenden Roboter besitzen ein elektrisches Antriebskonzept. Ausschließlich BigDog wird hydraulisch betrieben. Wenngleich seine Nutzlast, Größe und Geschwindigkeit vorteilhaft sind, ist er mit einem Verbrennungsmotor ungeeignet für den Einsatz in geschlossenen Räumen und unvorteilhaft in direkter Nähe zu Menschen. Die technischen Merkmale des Modells, hinsichtlich Größe, Gewicht und Antriebsleistung, sind deshalb an ANYmal, Spot und Cheetah 3 angelehnt. In Tabelle 4.1 und 4.2 sind die technischen Daten aller Roboter zusammengefasst.

Roboter	Laufzeit [h]	Max. Geschwindigkeit [m s^{-1}]	Antrieb [-]	Leistung [-]
ANYmal	4	1,0	elektrisch	80 N m
Spot	1,5	1,6	elektrisch	n.a.
BigDog	2,5	3,1	hydraulisch	15 PS
HyQ	inf.	2,0	hydraulisch	120 N m
Titan-XIII	n.a.	1,38	elektrisch	n.a
StarlETH	inf.	1,0	elektrisch	40 N m
Cheetah 3	2	6,0	elektrisch	230 N m
Stoch	n.a.	0,6	elektrisch	3,5 N m
Cheetah-cub	inf.	1,42	elektrisch	2,0 N m
Oncilla	0,5	0,63	elektrisch	4,5 N m

Tabelle 4.2.: Leistungsdaten der Roboter

Die Wahl der geeigneten Kinematik erfolgt unter Berücksichtigung von zwei Aspekten. Zuerst wird sie an den Anforderungen der zu bewältigenden Aufgabe gemessen. Mit Blick auf die Komplexität der Einsatzumgebung erscheint eine große Vielfalt der möglichen Bewegungsformen unter Beibehaltung der Stabilität vorteilhaft.

Potentielle, natürliche Vorbilder sind Hunde, Menschen, Spinnen oder Insekten. Als Vorteile der hundeartigen Kinematik führen (Kitano et al., 2016) an, sie ermögliche schnellere Gangarten und führe zu geringerem Energieverbrauch, verglichen mit einer spinnenartigen Kinematik. Dieser Ansatz wird insbesondere bei Cheetah 3 verfolgt, der gemäß seines natürlichen Vorbilds auf die Maximierung der Laufgeschwindigkeit ausgerichtet ist. Ein wesentlicher Vorteil der humanoiden Kinematik mit aufrechtem Gang, gegenüber den vier- und vielfüßigen Modellen, ist die Befreiung zweier der vier Gliedmaßen von der Fortbewegung, sodass diese für andere Aufgaben zur Verfügung stehen. Im Gegenzug wird eine geringere Stabilität in Kauf genommen.

Bezogen auf die Stabilität in der Grundhaltung sind die Kinematiken von Spinnentieren und Insekten der hundeartigen und humanoiden überlegen. Der relativ niedrige Schwerpunkt und die größere Fläche des *Support Polygon* (Bereich, innerhalb dessen der Schwerpunkt bei Wahrung der statischen Stabilität liegen kann), wird durch die niedrige Positionierung des Körpers zwischen den abgespreizten Beinen erreicht. Darüber hinaus besitzen Spinnentiere und Insekten die größere Agilität, insbesondere bezüglich lateraler Bewegungen und Wendemanöver. Unter diesen Gesichtspunkten erscheint eine spinnenartige Kinematik als am besten geeignet.

Der zweite Aspekt ist die technische Ableitung des natürlichen Vorbilds. Es ist mit hoher Wahrscheinlichkeit anzunehmen, dass die Kinematik von Spinnentieren als Produkt ei-

nes evolutionären Prozesses nahezu optimal ist. Nicht immer offensichtlich sind dabei die Kriterien hinsichtlich derer Optimalität angestrebt wird und ihre Auswirkungen auf das Ergebnis. Geparden, als Vorbilder von Cheetah 3, sind die schnellsten Landtiere der Erde. Darüber hinaus besitzen sie die notwendigen motorischen Fähigkeiten um zu klettern. Letzteres spielt in der technischen Nachbildung keine Rolle. Hieraus erwächst die Frage, ob sie mit einer veränderten Kinematik, unter Verlust der Kletterfähigkeit, gegebenenfalls noch schneller sein könnten. Für die biomimetische Gestaltung eines Roboters bedeutet dies, dass zwischen den Anforderungen des natürlichen Vorbilds und denen an die technische Nachbildung zu unterscheiden ist. Anders als Spinnentiere, steht der Roboter nicht in einer Räuber-Beute-Beziehung zu seiner Umwelt und ist auch nicht auf die Fähigkeit angewiesen, an Wänden oder Decken manövrieren zu können. Die Gestaltung seiner Kinematik ist damit anderen Randbedingungen unterworfen.

Vor dem Hintergrund dieser Arbeit wird die Umsetzung der spinnenartigen Kinematik von zwei gegenläufigen Zielen bestimmt. Dem Grundgedanken des RL folgend, ist eine möglichst freie und flexible Kinematik anzustreben. Mit steigender Anzahl der Freiheitsgrade, wächst die Menge der realisierbaren Bewegungen, was schließlich zu einer vielseitigen Policy führt. Darüber hinaus vergrößert dies die Möglichkeiten, mithilfe der Belohnungsfunktion gezielt Einfluss auf das erlernte Verhalten zu nehmen. Umgekehrt betrachtet kann eine zu restriktive Kinematik die Menge der möglichen Lösungen derart reduzieren, dass die Wirkung der Belohnungsfunktion dabei unterminiert wird.

Dem entgegen steht die praktische Hürde des Curse of Dimensionality. Der nötige Rechenaufwand, um eine Bewegungsstrategie zu finden wächst mit den gegebenen Freiheiten des Modells.

Die gewählte Kinematik ist ein Kompromiss aus diesem Spannungsfeld. Charakteristisches Merkmal von Spinnentieren ist die Anzahl von acht Beinen. Hinsichtlich der

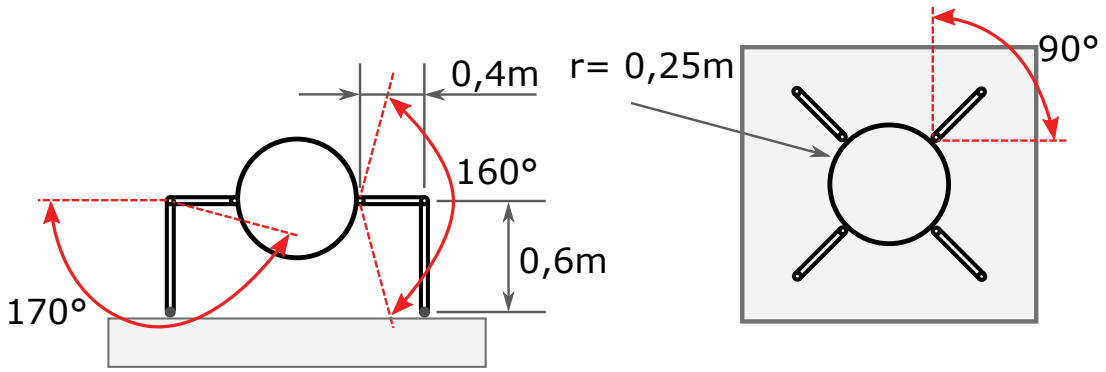

Abbildung 4.1.: Abmessungen und Aktionsradius des Robotermodells.

damit ausgeführten, natürlichen Gangarten ist eine Redundanz festzustellen. Der achtbeinige Gang wird demnach aus zwei vierbeinigen Bewegungssequenzen kombiniert (Biancardi et al., 2011). Für die Anforderungen des Laufroboters sind deshalb vier Beine ausreichend. Ausgangspunkt für die Kinematik der einzelnen Beine ist das Modell des Spinnenbeins von (Gasparetto et al., 2008). Sie reduzieren die tatsächliche Anzahl der Glieder und Gelenke auf ein Modell mit vier Freiheitsgraden pro Bein. Mit Blick auf die Modelle in vergleichbaren Arbeiten (vgl. Kapitel 2) wird in dieser Arbeit eine Kinematik mit drei motorisierten Freiheitsgraden pro Bein, insgesamt also zwölf Freiheitsgraden, verwendet. Eines der Scharniergelenke wird eingespart. Die Einschränkung der Bewegungsvielfalt gegenüber (Gasparetto et al., 2008) wird als wenig bedeutsam eingeschätzt, wohingegen ein signifikanter Vorteil hinsichtlich der Rechenzeit zu erwarten ist. Die Kinematik des Roboters und seine Bewegungsfreiheit sind in Abbildung 4.1 dargestellt.

4.2. Trainingsgelände und Sensorik

Der Agent wird mit zwei unterschiedlichen Umgebungen konfrontiert. In einem Basistraining wird die Fortbewegung in einer barrierefreien Umgebung auf ebenem Untergrund trainiert. Parallel dazu, wird dasselbe Training in einem Hindernisparcours ausgeführt. Erfolgreiche Agenten werden in einer dritten Umgebung erprobt, die sich von den Trainingsumgebungen unterscheidet.

In der Gestaltung des Parcours folgt diese Arbeit der Argumentation von (Heess; T.B. et al., 2017). Demnach ist eine anspruchsvolle und abwechslungsreiche Trainingsumgebung notwendige, wenngleich nicht hinreichende Bedingung, für das Erlernen einer robusten und vielseitigen Policy. Sie repräsentiert einen wesentlichen Teil des Zustandsraums, innerhalb dessen der Roboter Bewegungsstrategien erlernen soll. Je größer die Vielfalt der durchlaufenen Situationen im Training ist, desto höher ist die Wahrscheinlichkeit später eine Lösung für unbekannte, jedoch ähnliche Probleme zu finden.

Der Trainingsparcours ist eine Kombination aus ebenen Plattformen, Rampen, Stufen, Hindernissen und Spalten. Die Kombination und Abfolge der Elemente sieht eine graduelle Steigerung des Schwierigkeitsgrades vor, wobei Schwierigkeit als subjektives, menschliches Ermessen zu bewerten ist. Sichere Rückschlüsse über den Trainingsaufwand, der für die Bewältigung einzelner Abschnitte erforderlich ist, können daraus nicht gezogen werden. Der graduelle Charakter bezieht sich auf die ansteigende räumliche Dichte verschiedener Elemente. Dieser Aufbau soll dem Roboter die Möglichkeit

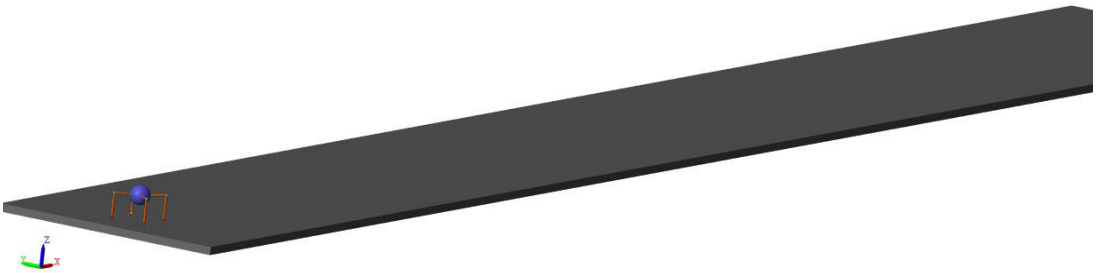

Abbildung 4.2.: Trainingsumgebung für das Basistraining.

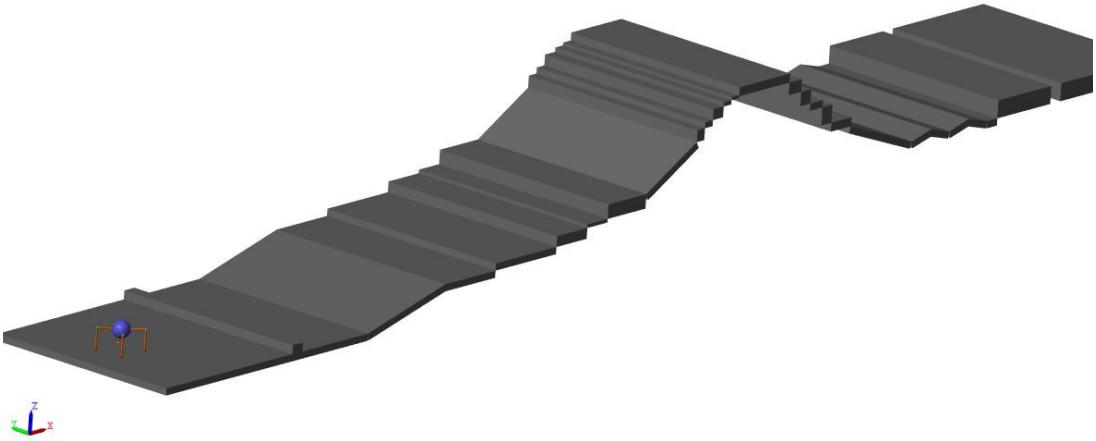

Abbildung 4.3.: Trainingsparcour mit Treppen und Hindernissen.

geben, in einfacher Umgebung Selbststabilisierung und grundlegende Bewegungen zu erlernen, bevor koordinativ anspruchsvollere Probleme zu lösen sind. Während das Training in der Ebene beginnt und erst nach mehreren Schritten das erste Hindernis auftritt, wird der Roboter nachfolgend in immer kürzeren Abständen mit neuen Situationen konfrontiert. Bezüglich des Treppensteigens wird die Annahme getroffen, dass das Emporsteigen leichter zu erlernen ist, als das Herabsteigen. Nahe dem Ende des Geländes ist ein Podest zu bewältigen, dessen Höhe ein zufälliges Überwinden, anders als bei Treppenstufen, unwahrscheinlich macht. Den Abschluss bildet ein Spalt zwischen der Zielplattform und dem Rest des Parcours. Im Gegensatz zum offenen, seitlichen Rand kann dieses Element nicht berücksichtigt werden indem es gemieden wird, sondern erfordert eine gezielte Steuerung und Koordination der Schrittweite, um einen Sturz zu vermeiden.

Das gesamte Trainingsgelände ist 40 m lang und 6 m breit. Die äußeren Dimensionen und die Anzahl der Strukturen sind ein Kompromiss aus der Forderung nach Vielfalt und deren Einfluss auf die Simulationsgeschwindigkeit. So findet eine Variation der Elemen-

te in Längsrichtung, nicht aber über die Breite des Parcours statt. Der Trainingsparcours und die Umgebung für das Basistraining sind in den Abbildungen 4.2 und 4.3 dargestellt.

Die Auswahl der Sensoren des Roboters erfolgt vorrangig unter der Prämisse, der Markov-Eigenschaft (vgl. Kapitel 3) hinreichend zu genügen, wobei die technische Umsetzbarkeit gegeben sein muss. Dies umfasst:

- die Orientierung des Körpers im Raum,
- die translatorische Geschwindigkeit,
- die Winkelgeschwindigkeit um die körpereigenen Achsen,
- die translatorische Beschleunigung,
- die Winkelposition, -geschwindigkeit und -beschleunigung der Gelenke,
- sowie die Normal- und Reibkraft an den Fußaufstandspunkten.

Für die Kontaktpaarung zwischen den Füßen des Roboters und dem Untergrund wird ein konstanter Reibungskoeffizient angenommen. Diese Vereinfachung stellt eine nicht unerheblich, jedoch notwendige Abstraktion der Realität dar. Modelle zur sensorischen Ermittlung von Reibbeiwerten sind noch Gegenstand der Forschung.

Die Kontur der Umgebung wird, ähnlich dem Vorgehen von (Heess; T.B. et al., 2017) durch ein eindimensionales Raster erfasst. An 21 äquidistanten Punkten entlang der Längsachse des Parcours, mit der aktuellen Position des Roboters in der Mitte, wird die relative Höhe der Umgebung zur Position des Körpers gemessen. Der Abstand zweier Messpunkte ist orientiert an der Größe der Strukturen. Es ist gewährleistet, dass zu jedem Zeitpunkt mindestens zwei Messpunkte zur Abtastung eines Elements benutzt werden. Technisch umsetzbar ist ein solches Verfahren etwa mit einem am Roboter montierten Lidar-System. Hierbei nicht berücksichtigt sind Abschattungseffekte und Selbstdetektion. Letzteres Problem kann mithilfe der bekannten Gelenkstellungen rechnerisch umgangen werden.

Darüber hinaus wird die relative Entfernung des Roboters von der Längsachse des Trainingsgeländes erfasst. Dies ist ein Kompromiss im Zusammenhang mit der eindimensionalen Geländewahrnehmung. Ein zweidimensionales Abtasten der Umgebung führte zu einer erheblich größeren Anzahl der Sensorsignale, die dem neuronalen Netz zugeführt würden. Da die Elemente des Trainingsgeländes nur in einer Dimension variiert werden, wäre ein Mehrwert ausschließlich durch die Detektion der seitlichen Ränder gegeben. Mit der lateralen Abweichung des Roboters aus der Mitte des Parcours, ist eine Verknüpfung zum Sturz bei Überschreiten der Ränder möglich.

Zusammen mit den zwölf Drehmomenten der Gelenkmotoren bilden die Sensoren eine *Observation* mit 91 Dimensionen. Die Daten werden vor der Übergabe an den Agenten auf einen Wertebereich von −1 bis 1 normiert. Die Trainingsbedingungen und -parameter des Agenten werden im nächsten Abschnitt diskutiert.

4.3. Trainingsparameter

Das Training des Agenten erfolgt unter Verwendung des DDPG-Algorithmus. Er ist geeignet für durchgängiges RL mit hochdimensionalen Observations (Lillicrap et al., 2015) und zeigt konkurrenzfähige Leistungen im Umgang mit physikalischen Steuerungsproblemen (Duan et al., 2016).

4.3.1. Actor-Critic-Repräsentation

Policy und Value-Function der Actor-Critic-Architektur werden durch neuronale Netze repräsentiert. Da die Gestaltung geeigneter Netzwerkstrukturen keinen universellen Regeln folgt, sondern vielmehr auf Erfahrungen beruht, entspricht die nachfolgende Architektur in weiten Teilen dem Netzwerk aus (Lillicrap et al., 2015). Das Actor-Netz umfasst zwei verdeckte Schichten mit 400 beziehungsweise 300 Neuronen. Der Agent soll die direkte Kontrolle über die Gelenkmotoren des Roboters lernen. Die Ausgabeschicht des Actor besteht dementsprechend aus zwölf Elementen, deren Ausgabe jeweils das Drehmoment für einen Freiheitsgrad bestimmt. Das Critic-Netz besteht ebenfalls aus zwei verdeckten Schichten mit 400 und 300 Elementen. Während der ersten Schicht nur die Observation zugeführt wird, sind in der zweiten Schicht auch die Eingänge der letzten Action berücksichtigt.

Die Ausgangsschicht des Actor-Netzes schließt mit einer *tanh*-Aktivierungsfunktion ab. Die Ausgabe der Drehmomente wird damit auf Werte im Bereich von −1 bis 1 beschränkt und kann entsprechend der gewünschten maximalen Leistung skaliert werden.

Als Aktivierungsfunktion für alle verdeckten Schichten wird die ReLU-Funktion verwendet. Im Vergleich zu anderen Funktionen hat sie nachweislich positiven Einfluss auf die Lerngeschwindigkeit eines neuronalen Netzes und zählt deshalb zu den derzeit am häufigsten verwendeten Aktivierungsfunktionen. In Kapitel 3 wurde bereits das Problem des *dying ReLU* thematisiert. Die weiterentwickelten Varianten LReLU und PReLU (He et al., 2015), versuchen dieses Problem zu umgehen, haben sich in der Praxis aber bisher nicht bewährt. Unter diesen Aspekten ist die ReLU-Funktion die sinnvollste Wahl.

Die Initialisierung der Weights und Biases in den verdeckten Schichten folgt dem Schema von (He et al., 2015) (vgl. Abschnitt 2.2). Die Ausgabeschicht beider Netzwerke wird, ähnlich wie von (Lillicrap et al., 2015) empfohlen, mit kleinen Werten aus einer stetigen Gleichverteilung initialisiert, um die Ausgabe zu Beginn des Trainings klein zu halten.

Der Optimierungsalgorithmus für Actor- und Critic-Netz ist *ADAM*. Der Algorithmus gilt als effizient hinsichtlich des Speicherbedarfs und benötigt nur wenige Hyperparameter (Kingma et al., 2014). Die Lernraten für Actor und Critic werden mit $1 \cdot 10^{-4}$ beziehungsweise $1 \cdot 10^{-3}$ festgelegt. Um *Overfitting* zu vermeiden wird *L2-Regularization* angewendet.

4.3.2. Parameter des RL-Agenten

Die Mini-Batch-Größe wird mit 32 relativ klein gewählt. Dafür spricht die Argumentation von (Masters et al., 2018) bezüglich der Stabilität des Trainings. Darüber hinaus wird kleinen Mini-Batch-Größen die Wirkung zugeschrieben, lokale Extremstellen und Sattelpunkte des Parameterraums zu überwinden, was zu einer besseren Generalisierung führt (Keskar et al., 2016), (Smith et al., 2017).

Der *Smoothing*-Faktor regelt die Stärke der Bindung zwischen den Parametern des Actor- und Critic-Netzwerks mit ihren jeweiligen *Target*-Netzwerken. (Haarnoja; Zhou; Abbeel et al., 2018) haben die Variation des Parameters über mehrere Größenordnungen verglichen. Sie beobachten eine destabilisierende Wirkung wenn der Parameter zu groß ($1 \cdot 10^{-1}$) gewählt wird und eine Verlangsamung des Trainings für zu kleine Werte ($1 \cdot 10^{-4}$). Ihren Erkenntnissen folgend wird der Faktor mit $1 \cdot 10^{-3}$ festgelegt.

Die Größe des Experience-Buffer und die *Sample-Time* (die Steuerungsfrequenz des Agenten), sind zwei Parameter, die in unmittelbarem Zusammenhang zueinander stehen. Im Vergleich zu Parametern wie Mini-Batch-Größe und Lernrate, behandeln andere Arbeiten nur selten die Kriterien, nach denen sie ausgewählt werden.

Der Experience-Buffer muss ausreichend dimensioniert werden, um die Korrelation zwischen kürzlich erhaltenen Erfahrungen und der Anpassung der Policy aufzulösen (Mnih et al., 2015). Die Größe des Buffers wird in der Anzahl der darin enthaltenen Schritte des Agenten bemessen. Für das Episoden-basierte Training des Roboters bedeutet dies, dass die Erfahrungen aller Schritte einer ausreichend großen Anzahl zurückliegender Trainingsversuche im Buffer gespeichert werden muss. Im einfachsten Fall ist er groß genug, um alle gesammelten Erfahrungen des Trainings zu speichern.

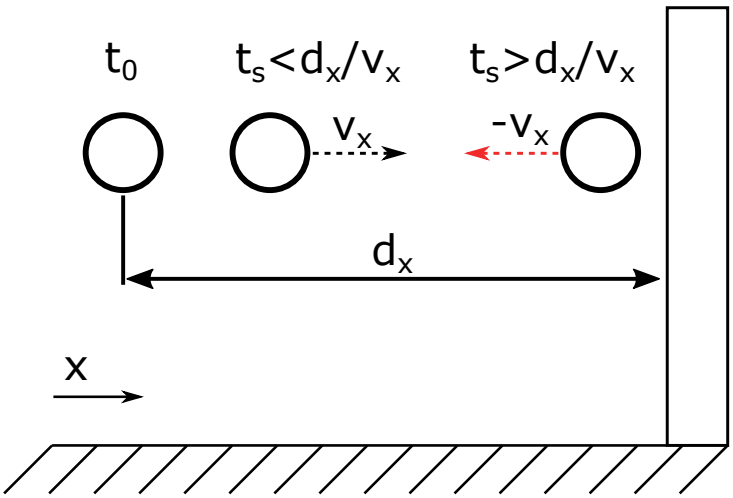

Abbildung 4.4.: Gedankenexperiment zur Sample-Time.

Die Anzahl der Schritte pro Episode ist abhängig von der Sample-Time und der Zeitdauer einer Episode. Die maximale Länge einer Trainingsepisode wird auf 40 s festgelegt. Dies ist eine Abschätzung der Zeit, die der Roboter maximal benötigt, um den Trainingsparcours zu beenden, wenn er eine erfolgreiche Policy anwendet. Davon ausgehend, dass er eine durchschnittliche Laufgeschwindigkeit von $1\,\mathrm{m\,s^{-1}}$, vergleichbar zu seinen technischen Vorbildern, erreicht, ist dies die Zeit, welche er für den 40 m langen Parcours benötigt.

Die Sample-Time sollte die Dynamik des zu steuernden Systems widerspiegeln, damit der Agent die richtigen Rückschlüsse bezüglich der zuvor gewählten Aktionen ziehen kann. Exemplarisch dafür ist der in Abbildung 4.4 dargestellte Fall. Angenommen, es sei die Aufgabe, einen Ball mit größtmöglicher Geschwindigkeit in eine bestimmte Richtung zu werfen, wäre die offensichtliche Lösung, möglichst viel Kraft dafür aufzuwenden. Prallt der Ball während des Flugs gegen eine Wand, wird seine Geschwindigkeit umgekehrt. Ob der Agent für den Wurf mit maximaler Kraft nun eine Belohnung erhält oder möglicherweise sogar bestraft wird, hängt (neben der Belohnungsfunktion) von der Sample-Time ab. Ist sie zu groß gewählt und der Ball befindet sich bei der Bewertung des neuen Zustands bereits wieder auf dem Rückweg, wird die Aktion fälschlicherweise negativ bewertet. Für ein komplexeres System wie den Roboter ist der Zusammenhang zwischen gewählter Aktion, Sample-Time und korrekter Bewertung nicht so offensichtlich wie in diesem Beispiel. Der Grundgedanke erscheint dennoch valide.

Die Notwendigkeit eines Kompromisses zwischen einem großen Experience-Buffer und einer kleinen Sample-Time ergibt sich aus dem begrenzten verfügbaren Arbeitsspeicher. Unter Berücksichtigung der Dynamik der Roboterbeine wird sie festgelegt auf $1 \cdot 10^{-2}$ s. Bei einer Buffer-Größe von $4 \cdot 10^{6}$ Schritten können mindestens die letzten

1000 Episoden darin gespeichert werden. Die tatsächliche Anzahl wird größer sein, weil die Mehrheit der Episoden nicht die maximale möglichen 4000 Schritte umfasst.

Der Ornstein-Uhlenbeck-Prozess zur Exploration ist in MATLAB implementiert als

$$N = N_{t-1} + \beta(u - N_{t-1})T_S + \rho c\sqrt{T_S} \tag{4.1}$$

mit den Parametern:

N_t = Noise
β = Anziehungskonstante bezüglich des Mittelwerts
u = Mittelwert
T_S = Sample-Time des Agenten
ρ = Varianz
κ = Zufallszahl aus der Normalverteilung

Es ist üblich die Varianz so zu wählen, dass

$$0{,}01 \cdot P \leq \rho\sqrt{T_S} \leq 0{,}1 \cdot P \tag{4.2}$$

wobei P die Breite des Wertebereichs der Aktionen des Agenten beschreibt (*MATLAB Documentation* 2020).

Die Varianz wird $4 \cdot 10^{-1}$, ihrer Abklingrate mit $1 \cdot 10^{-5}$, sowie die Anziehungskonstante mit $9 \cdot 10^{-1}$ festgelegt. Anfangs- und Mittelwert des Noise-Modells sind Null. Abbildung 4.5 zeigt exemplarisch die Entwicklung des Noise über eine Trainingsepisode in den Grenzen von Gleichung 4.2. Abgebildet ist ein Vergleich der gewählten Varianz gegenüber dem maximalen, minimalen und mittleren empfohlenen Wert. Mit der kleinen Abklingrate wird Exploration für die gesamte Dauer der Trainingsepisode gefördert. Varianz und Anziehungskonstante sind abgestimmt auf den Wertebereich des Agenten-Ausgangs von −1 bis 1. Der Noise ist normalverteilt im Wertebereich von −0,8 bis 0,8 (Abbildung 4.6), sodass innerhalb des Wertebereichs des Agenten umfassende Exploration gewährleistet ist, ohne die Bereichsgrenzen zu häufig zu verletzen.

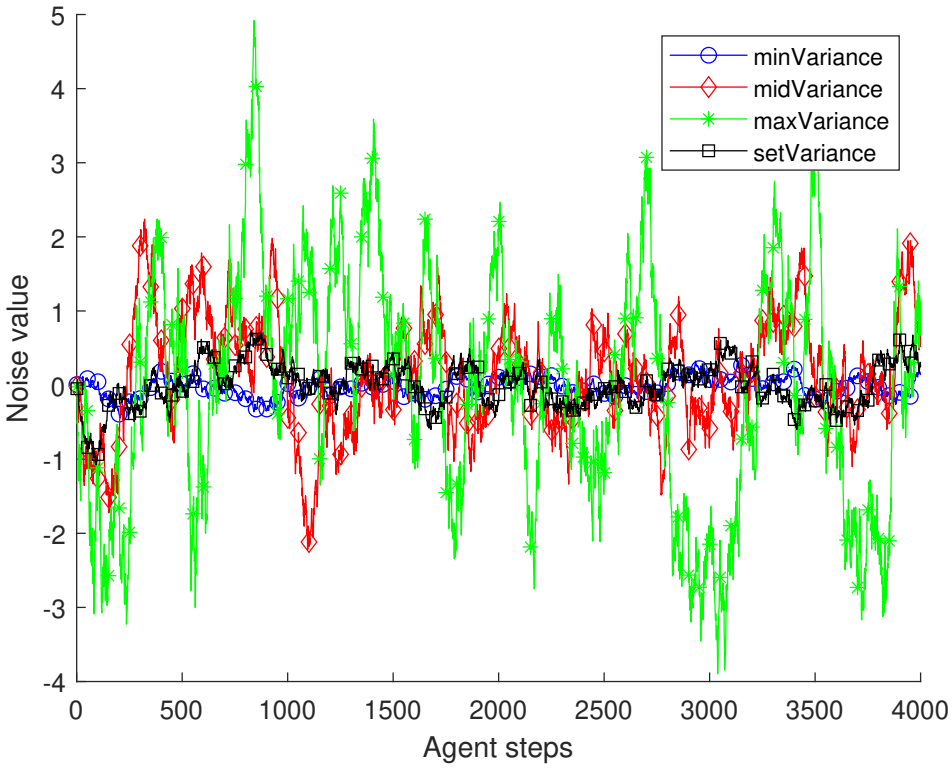

Abbildung 4.5.: Vergleich der Noise-Entwicklung in Abhängigkeit der Varianz.

Die Trainingsepisode wird abgebrochen, wenn:

- der Zielbereich erreicht wird,
- 40 s bzw. 4000 Schritte absolviert sind,
- Roll- oder Nickwinkel von 90° erreicht werden,
- der Körper mit dem Parcours kollidiert,
- ein Kniegelenk mit dem Parcours kollidiert,
- eine Selbstkollision zwischen Fuß und Körper auftritt.

Ein Trainingsdurchlauf des Roboters umfasst maximal 50 000 Episoden. Die Möglichkeit des Early-Stopping wird genutzt, wenn der Agent über eine Spanne von 250 Episoden durchschnittlich 95 % der maximalen Belohnung erhalten hat. Die Policy wird nach jeweils 32 Schritten des Agenten, sowie am Ende einer Episode aktualisiert.

Mit diesen Trainingsbedingungen wird ein robuster und stabiler Trainingsverlauf angestrebt. Im folgenden Kapitel wird die Gestaltung der für das Training verwendeten Belohnungsfunktionen behandelt.

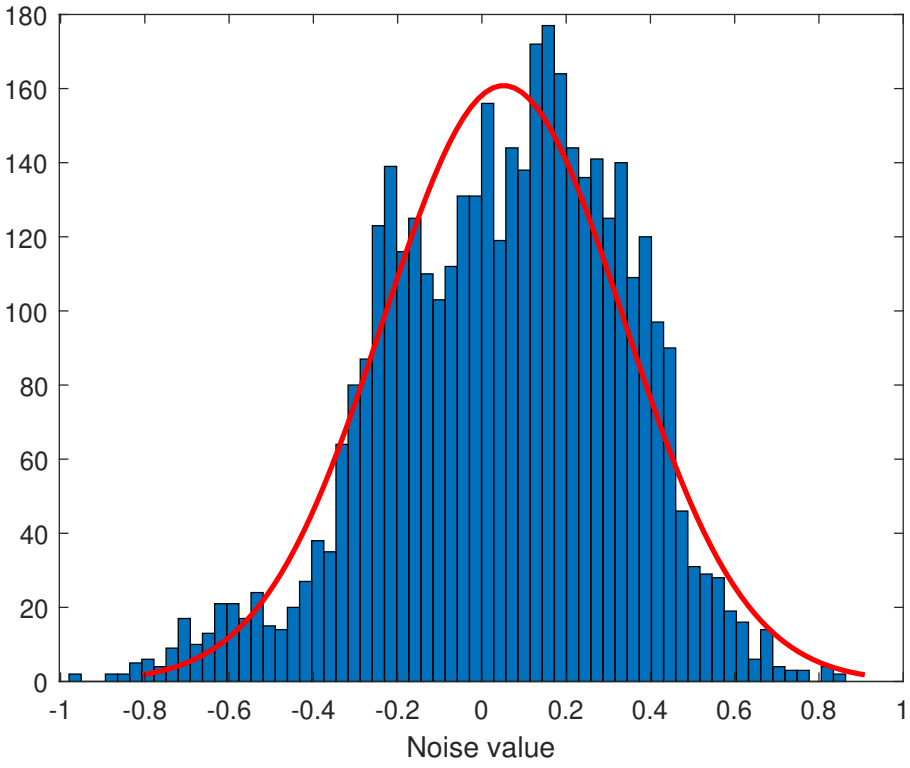

Abbildung 4.6.: Histogramm der Noise-Werte für die gewählte Varianz.

5. Belohnungsfunktionen

Dem Gestaltungsprozess von Belohnungsfunktionen im Kontext von RL in der Robotik wird bisher wenig Aufmerksamkeit geschenkt. In der Regel werden sie in einem zeitintensiven, iterativen Prozess händisch entwickelt (Heess; T.B. et al., 2017). Erfahrungen, die in dessen Verlauf gewonnen werden bleiben zumeist undokumentiert. Aus bisherigen Arbeiten in diesem Bereich lassen sich deshalb keine Strategien oder Richtlinien ableiten, die den Gestaltungsprozess beschleunigen oder die Ergebnisse verbessern könnten.

Dieses Kapitel dokumentiert den Entwicklungsprozess der Belohnungsfunktionen, mithilfe derer der Roboter das Treppensteigen erlernen soll. Zunächst erfolgt eine grundsätzliche Betrachtung der Entscheidungen, die im Zuge der Entwicklung zu treffen sind. Daran anschließend werden Belohnungsfunktionen aus anderen Veröffentlichungen diskutiert und bekannte Fehlversuche analysiert. Vor diesem Hintergrund werden im letzten Abschnitt dieses Kapitels die zu testenden Belohnungsfunktionen hergeleitet.

5.1. Zielformulierung

Vor dem Entwurf der geeigneten Belohnungsfunktion, steht die Formulierung des Ziels, welches damit verfolgt wird. Diese Feststellung erscheint zunächst trivial, kann aber weitreichende Konsequenzen für den Einsatz von RL haben, wie im Folgenden erläutert wird.

Die Arbeiten von (Heess; T.B. et al., 2017), (Lillicrap et al., 2015), (Heess; Wayne et al., 2016) und (Haarnoja; Zhou; Ha et al., 2018) haben alle das Ziel, Robotern mithilfe von RL das Laufen zu lehren oder benutzen diese Aufgabe, um die Wirksamkeit und Leistungsfähigkeit ihrer Methoden zu demonstrieren. Dabei steht die Frage im Vordergrund, ob es möglich ist, die Aufgabe mit RL zu lösen. Sie stellen keine expliziten Ansprüche an die Eigenschaften der erlernten Bewegungen. Während dies im Rahmen der Forschungsarbeiten legitim ist, erfordert die praktische Anwendung von RL, analog zum klassischen Steuerungsentwurf, hinreichende Vorgaben an denen das Ergebnis gemessen werden kann.

Der Humanoid von (Heess; T.B. et al., 2017) erlernt bemerkenswert robuste Bewegungsformen. Ein Merkmal seiner Policy sind relativ ruckartige, ausladende Bewegungen der Arme, möglicherweise ein Versuch um durch Trägheitseffekte die Balance zu wahren (Video [1]). Diese Bewegungen sind natürliche Nebenprodukte des offenen, flexiblen Lernprozesses und erinnern an Spandreln als Nebenprodukte evolutionärer Prozesse. Sie sind Teil einer erfolgreichen Fortbewegungsstrategie, wären aber für den praktischen Einsatz sicherlich unerwünscht. Über einen Eingriff in die Belohnungsfunktion könnten diese Bewegungen unterdrückt werden, mit dem Risiko, die Policy grundlegend zu ändern und neue Nebenprodukte zu erzeugen.

Je strikter und anspruchsvoller die Zielsetzung ist, desto wahrscheinlicher ist es, dass der RL-Prozess Effekte hervorbringt, die in Konflikt zu den Zielen stehen. Es ist demnach vorstellbar, dass eine Klasse komplexer Probleme existiert, für die durch händischen Entwurf keine geeignete Belohnungsfunktion gefunden werden kann. Es entbehrt nicht einer gewissen Ironie, dass die Flexibilität dieser Form des RL, die es für komplexe Probleme so attraktiv macht, gleichzeitig ein begrenzender Faktor für die Komplexität der Aufgaben ist, auf die es anwendbar ist. Die Schwierigkeit liegt darin, dass a priori nicht bekannt ist, ob ein definiertes Ziel mit RL und manuell gestalteten Belohnungsfunktionen erreichbar ist, sodass nur der Versuch bleibt.

5.2. Dimensionen einer Belohnungsfunktion

Wenn das Ziel einer RL-Aufgabe feststeht, muss es in die Komponenten der Belohnungsfunktion übersetzt werden. Die dabei zu treffenden Entscheidungen umfassen im Wesentlichen fünf Dimensionen:

1. Wahl der geeigneten Parameter
2. Anzahl der Parameter
3. Mathematische Gestalt der Parameter
4. Verhältnis / Gewichtung der Parameter zueinander
5. Zeitverhalten der Belohnungsfunktion

[1] `https://www.youtube.com/watch?v=hx_bgoTF7bs`

Für vergleichsweise simple Aufgaben wie *Cart-Pole*, bei der der Agent durch Manipulation eines Wagens, eine Stange auf selbigem balancieren muss, ist die Wahl der Parameter intuitiv möglich. Die Winkelposition der Stange ist hier ein unmittelbarer Maßstab für den Erfolg. Lautet die Aufgabe jedoch eine Partie Schach zu gewinnen, ist die Wahl weitaus weniger offensichtlich. Einerseits könnte eine große Zahl verbleibender Figuren auf dem Spielfeld erstrebenswert sein, andererseits sind Läufer, Springer und Dame womöglich wertvoller als die gleiche Anzahl Bauern. Zu dem Beispiel der Schach-Partie merkt (Marom et al., 2018) an, dass eine solche Parameterwahl bereits eine Lösungsstrategie implementiere und diese unter Umständen nicht optimal sei. Sie verletzt damit die Forderung von (Ng et al., 1999) an das Reward-Shaping. Alternativ dazu kann der Agent eine ausschließliche Belohnung am Ende einer gewonnenen Partie erhalten. Allerdings könnte dies so selten geschehen, dass der Lernprozess in einem Maß verlängert würde, das als unpraktikabel gilt. Auf hochdimensionale Probleme wie autonome Laufroboter trifft dies ebenso zu. Es ist also möglicherweise ein notwendiger Kompromiss bei der Parameterauswahl, einen Teil der Optimalität aufzugeben, um in vertretbarer Zeit solide Lösungen für praktische Probleme zu erhalten. Dieser Ansatz erscheint insbesondere vor dem Hintergrund nicht-konvexer Optimierungsprobleme vertretbar, bei denen die optimale Lösung in der Regel nicht bekannt ist.

Bezüglich der richtigen Anzahl der Komponenten sind zwei verschiedene Betrachtungsweisen möglich. Wie in Abschnitt 2.4 bereits beschrieben, befürworten (Heess; T.B. et al., 2017) ein minimalistisches Prinzip, das nur eben so viele Parameter eingebunden werden, wie zwingend erforderlich sind. Die Argumentation, den Lernprozess möglichst offen zu gestalten, um die Menge der möglichen Lösungen für den Agenten nicht unnötig zu beschränken ist aus Sicht der Theorie schlüssig. Von einem praktischen und wirtschaftlichen Standpunkt betrachtet stellt sich die Frage, ob eine Belohnungsfunktion mit mehr als der notwendigen Anzahl der Komponenten nicht benutzt werden kann, um die leitende Wirkung für den Agenten zu verstärken und das Training zu beschleunigen. In einem Vergleich zwischen diesen beiden Ansätzen sind vier verschiedene Ergebnisse denkbar. Dafür sei vorausgesetzt, dass die minimalistische Belohnungsfunktion eine optimale Policy zur Folge hat. Im schlechtesten Fall verändert ein zusätzlicher Parameter die Policy grundlegend, sodass die Zielvorgaben nicht mehr erfüllt werden. Ebenfalls kontraproduktiv wäre eine Verlängerung der Trainingszeit, selbst bei Aufrechterhaltung der optimalen Policy. In einem günstigeren Fall wird eine suboptimale, aber annehmbare Policy in signifikant kürzerer Zeit erlernt. Der ideale Ausgang ist schlussendlich die Wahrung der optimalen Lösung, bei einer Verkürzung der Trainingsdauer. Dass verschiedene Belohnungsfunktionen dieselbe optimale Policy für eine Aufgabe teilen können, beweisen (Ng et al., 1999).

Neben der Parameterwahl selbst, ist zu entscheiden durch welchen mathematischen

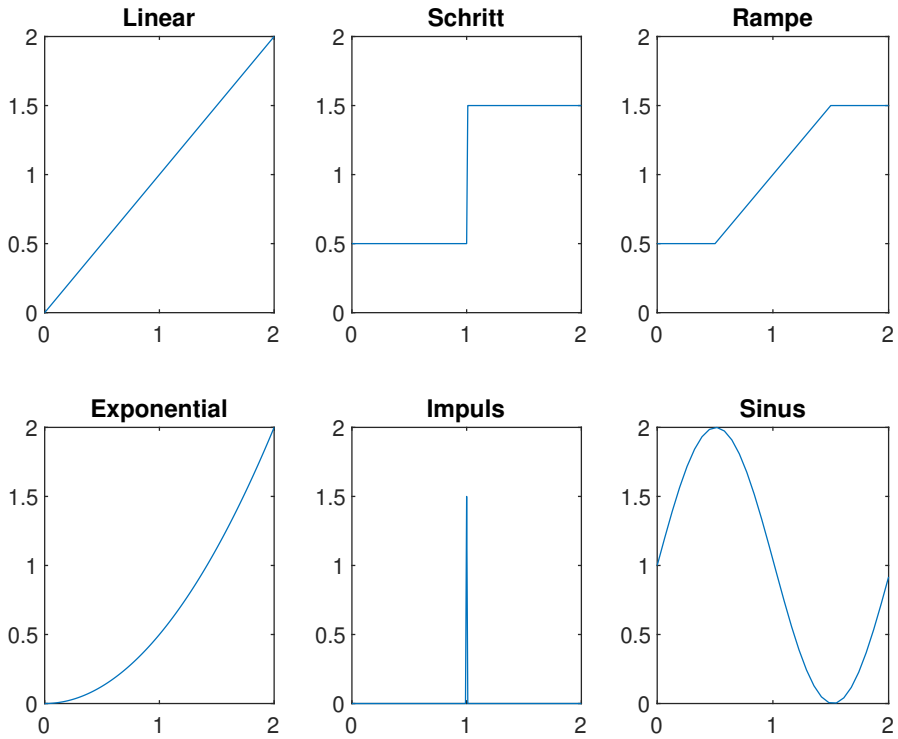

Abbildung 5.1.: Auswahl mathematischer Grundfunktionen

Funktionstyp die Belohnung oder Bestrafung ausgedrückt werden soll. Eine nicht erschöpfende Auswahl ist in Abbildung 5.1 gegeben. Die gegebenen Möglichkeiten werfen mehrere Fragen auf, die, soweit bekannt, bisher noch in keiner Arbeit thematisiert wurden:

- Welche Auswirkungen hat die Wahl von stetigen gegenüber unstetigen Funktionen?
- Wie beeinflussen lineare Terme Lernprozess und Policy im Vergleich zu Exponentialfunktionen?
- Gibt es Wechselwirkungen zwischen verschiedenen Funktionstypen innerhalb einer Belohnungsfunktion?

Über den Einfluss der mathematischen Gestalt einer Belohnungsfunktion ist bisher wenig bekannt. In der Regel bestehen manuell entworfene Belohnungsfunktionen aus einer Summe unabhängiger einzelner Komponenten in der Form:

$$r = \phi_1 + \phi_2 \tag{5.1}$$

Grundsätzlich denkbar sind auch Verknüpfungen von Komponenten als

$$r = \phi_1 + \phi_2(\phi_1) \tag{5.2}$$

sodass der Erhalt einer Komponente der Belohnung gebunden ist an die Erfüllung eines anderen Parameters.

Günstige Verhältnisse in der Gewichtung der Parameter zu finden gehört zu den bekannten Herausforderungen in der Gestaltung von Belohnungsfunktionen. Dabei ist die richtige Abstimmung nicht nur abhängig von der zu lösenden Aufgabe (Heess; T.B. et al., 2017), sondern ist auch unter Berücksichtigung des verwendeten RL-Algorithmus zu wählen (Haarnoja; Zhou; Ha et al., 2018), (Duan et al., 2016).

Zuletzt ist zu klären, ob die Belohnungsfunktion statisch oder in Abhängigkeit der Zeit gestaltet wird. Moderne Algorithmen sind in der Lage eine Policy mit veränderlicher Zielfunktion zu trainieren (Kingma et al., 2014). Dies eröffnet prinzipiell die Möglichkeit, den Lernvorgang für komplexe Aufgaben in mehrere Etappen mit leichter erreichbaren Zwischenzielen zu zerlegen. Ähnlich wie Kleinkinder nacheinander Sitzen, Krabbeln und Laufen lernen, könnten auch RL-Aufgaben aufgeteilt werden. Der wesentliche Unterschied ist, dass das menschliche Gehirn, im Gegensatz zum neuronalen Netz, die gelernten Etappenziele nicht wieder vergisst. Wenn die Belohnungsfunktion während des Trainings verändert wird, werden Weights und Biases dem neuen Ziel entsprechend überschrieben. Dieser Ansatz kann trotzdem erfolgreich sein. Er setzt jedoch voraus, dass die Parameter der Policy für die gewählten Zwischenziele bereits in der Nähe der optimalen Lösung für die finale Aufgabe liegen. Der Zusammenhang zwischen einem geeigneten Etappenziel und einer komplexeren Gesamtaufgabe ist damit mathematischer Natur und nicht zwingend so intuitiv wie das natürliche Vorbild. Es erscheint plausibel, dass ein Roboter, der seine maximal mögliche Laufgeschwindigkeit erlernen soll, davon profitieren kann, wenn er bereits eine stabile Gangart erlernt hat. Wenn er vor dem Erlernen eines stabilen Gangs die Selbststabilisierung im Stillstand als Zwischenziel nutzt, kann die zugrundeliegende Policy gänzlich unterschiedlich sein.

5.3. Belohnungsfunktionen aus der Robotik

In diesem Abschnitt werden die Belohnungsfunktionen aus Arbeiten betrachtet, die ähnliche Probleme mit RL bearbeiten wie die Vorliegende. Die Funktionen in den Gleichungen 5.3-5.7, beziehungsweise 5.8-5.10 sind entnommen aus den Beiträgen von (Duan et al., 2016) und (Heess; T.B. et al., 2017). Beide trainieren verschiedene Laufroboter

in der Simulation, wobei (Heess; T.B. et al., 2017) einen Teil der Belohnungsfunktionen von (Duan et al., 2016) übernommen und angepasst haben.

- Ant

$$r = v_x - 0{,}005||a||^2 - C_{contact} + 0{,}05 \tag{5.3}$$

v_x = Geschwindigkeit entlang der x-Achse
a = Steuersignal des Agenten
$C_{contact}$ = Faktor in Abhängigkeit der Kontaktkraft der Füße

- Hopper

$$r = v_x - 0{,}005||a||^2 + 1 \tag{5.4}$$

- Walker

$$r = v_x - 0{,}005||a||^2 \tag{5.5}$$

- Half-Cheetah

$$r = v_x - 0{,}05||a||^2 \tag{5.6}$$

- Simple und Full Humanoid

$$r = v_x - 5 \cdot 10^{-4}||a||^2 - C_{contact} - C_{deviation} + 0{,}2 \tag{5.7}$$

$C_{deviation}$ = Abweichung von der Vorwärtsbewegung

- Planar Walker

$$r = 10v_x + 0{,}5n_z - |\Delta h - 1{,}2| - 10I[\Delta h < 0{,}3] - 0{,}1||u||^2 \tag{5.8}$$

v_x = Geschwindigkeit entlang der x-Achse
n_z = Projektion der Hochachse des Roboters auf die globale Hochachse
Δh = Höhe des Torso über den Füßen
I = Indikatorfunktion

- Quadruped

$$r = v_x + 0{,}05n_z - 0{,}01||u||^2 \tag{5.9}$$

- Humanoid

$$r = \min(v_x, v_{max}) - 0{,}005(v_x^2 + v_y^2) - 0{,}05y^2 - 0{,}02||u||^2 + 0{,}02 \tag{5.10}$$

v_{max} = Zielgeschwindigkeit
v_y = laterale Geschwindigkeit
y = lateraler Abstand vom gewünschten Pfad

RL mit realen Robotern betreiben (Haarnoja; Zhou; Ha et al., 2018) und (Schuitema, 2012) (Gleichung 5.11-5.12).

- Minitaur

$$r = w_1(x_t - x_{t-1}) - w_2|\ddot{a}_t| - w_3|\phi| - w_4 \sum_{i \in \{1,2\}} \max(\bar{q} - q_i, 0) \tag{5.11}$$

x_t, x_{t-1} = Position zu den Zeitpunkten t und $t-1$
$\ddot{a}$ = Winkelbeschleunigung der Gelenke
ϕ = Rollwinkel des Torsos
$\bar{q}, q_i$ = Grenzwinkel und effektive Winkelposition der Vorderbeine
w_i = Wichtungsfaktoren

- Real Humanoid

$$r = 300\Delta x - 1 - 125I[f = 1] - 3E \tag{5.12}$$

Δx = zurückgelegte Wegstrecke
$I[f = 1]$ = Indikatorfunktion für Detektion eines Sturzes f
E = Energieverbrauch in Joule

(Randlov et al., 1998) trainieren einen Agent eine Fahrradmechanik zu einem 1000 m entfernten Punkt zu fahren mit Gleichung 5.13:

$$r = (4 - \psi_g^2)\, 4 \cdot 10^{-5} \tag{5.13}$$

ψ_g = Winkel zwischen Fahrtrichtung und Zielrichtung

Alle Belohnungsfunktionen haben gemeinsam, dass sie den Fortschritt der Aufgabe anhand eines Geschwindigkeitsvektors bewerten. Die realen Systeme zu den Gleichungen 5.11 und 5.12 bemessen die Geschwindigkeit aus der Distanz zwischen den Positionen zu zwei Zeitpunkten, was auf ihre Messtechnik zurückzuführen ist. (Randlov et al., 1998) verwenden den Winkel zwischen der Bewegungsrichtung des Fahrrads und dem Zielpunkt. Hier wird vorausgesetzt, dass der Agent in Bewegung ist.

Die Höhe der Belohnung wächst in fast allen Fällen linear mit dem Geschwindigkeitsvektor. Einzig die Belohnung des *Humanoid* von (Heess; T.B. et al., 2017) ist durch eine Rampenfunktion bei erreichen einer Zielgeschwindigkeit begrenzt. Ob diese Maßnahme messbare Auswirkungen hat ist nicht überliefert.

Der Geschwindigkeitsterm wird in fast allen Fällen um eine oder mehrere Größenordnungen stärker gewichtet, als die übrigen Komponenten.

Eine Gemeinsamkeit aller Roboteraufgaben ist die Bestrafung proportional zum Steuerungsaufwand in Form von Drehmomenten oder Energieverbrauch. Dieser Term wird im Allgemeinen benutzt, um harmonische und präzise Bewegungen zu fördern, Impulsive hingegen zu vermeiden.

Auffällig ist der Unterschied zwischen den Gleichungen 5.5 und 5.8. Obwohl die Kinematiken praktisch identisch sind, verwenden (Heess; T.B. et al., 2017) drei zusätzliche Terme für die Höhe des Torsos und eine aufrechte Haltung. Dies ist insofern bemerkenswert, als dass sie ausdrücklich minimalistische Belohnungsfunktionen anstreben. Der Grund für ihre Änderung sind höchstwahrscheinlich die unterschiedlichen Trainingsumgebungen. Während der *Walker* in einer freien Ebene trainiert, muss der *Planar Walker* Hürden überwinden und unter Hindernissen hindurch tauchen. Dieses Beispiel zeigt anschaulich, wie die Komplexität einer Aufgabe die Belohnungsfunktion beeinflussen kann.

Ein ähnlicher Zusammenhang wird beim Vergleich von *Hopper*, *Walker* und *Half-Cheetah* gegenüber *Ant* und *Humanoid* deutlich. Die Erstgenannten verfügen über weniger Freiheitsgrade und sind in ihren Bewegungen an ihre Symmetrieebene gebunden. Die höherdimensionalen Kinematiken profitieren demnach von der zusätzlichen Rückmeldung durch eine Belohnungsfunktion mit mehr Parametern.

Die zu den Simulationsversuchen gehörigen Funktionen sind weitgehend neutral gegenüber der Kinematik formuliert, auf der sie angewendet werden. Im Fall von *Minitaur* findet ein expliziter Eingriff in die Bewegungsfreiheit des Agenten statt. Mit dem letzten Term der Belohnungsfunktion versuchen (Haarnoja; Zhou; Ha et al., 2018) ein Einklappen der Vorderfüße unter dem Körper zu verhindern. Sie identifizieren dies als eine der Hauptursachen für Fehlversuche in ihrem Training. Dies spiegelt einen interessanten Aspekt von RL mit realen Systemen wider. Aus Sicht des RL ist die Einschränkung der Fußstellung unnötig und hindert den Agent möglicherweise daran, größere Schrittweiten zu erlernen. In diesem Fall wird sie jedoch für notwendig befunden, um das Sturzrisiko zu verringern und die Zahl der menschlichen Eingriffe zu reduzieren. Es wird also bewusst das Potential des RL beschnitten, um das praktische Training zu beschleunigen.

Ein Teil der Funktionen beinhaltet einen skalaren Term ohne Bezug zu einem Parameter (Gleichungen 5.10, 5.3, 5.4, 5.7). Dieser liefert dem Agenten eine konstante Belohnung in jedem Zeitschritt. Anschaulich ausgedrückt wird der Roboter damit für seine Existenz oder sein Überleben belohnt. Ein bekanntes Problem sind Versuche des Roboters, sich zu Beginn des Trainings, entsprechend der Belohnung für den Geschwindigkeitsvektor, möglichst effektiv in eine Richtung zu werfen. Kurzfristig ist damit eine hohe Belohnung zu erreichen. Direkt im Anschluss folgt in aller Regel der Sturz. Ähnlich wie die Bestrafung für Energieverbrauch, wird dieser Term genutzt, um ein lokales Optimum zu vermeiden. (Schuitema, 2012) wählt für diesen Zweck einen anderen Weg. Anstelle eines konstanten Überlebensbonus, wird dort eine Bestrafung für Stürze ausgegeben, die größer ist als die Summe der Belohnungen, die mit dem lokalen Optimum gesammelt werden können (Schuitema, 2012, S. 74).

5.4. Fehlerhafte Belohnungsfunktionen

RL-Agenten neigen dazu, Schwächen in der Gestaltung der Belohnungsfunktion in unerwarteter Weise auszunutzen. Zwei bekannte, einander ähnliche Fälle, sind die Experimente mit dem Atari-Videospiel *Coast Runners 7* und die im vorherigen Abschnitt vorgestellte Fahrrad-Aufgabe.

Im Videospiel steuert der Agent ein Rennboot in einem Rundkurs. Er erhält Punkte für die Platzierung beim Zieleinlauf, die benötigte Zeit, sowie für das Einsammeln von Elementen während des Rennens. Das Lernziel ist die Maximierung der Punktzahl. Anstatt der menschlichen Intuition zu folgen und das Rennen in möglichst kurzer Zeit beenden zu wollen, entwickelt der Agent die Strategie, um einen Punkt auf der Strecke zu kreisen, an dem er die immer wiederkehrenden Elemente einsammeln kann, ohne das Rennen zu beenden (Clark et al., 2016).

In einer ersten Version der Belohnungsfunktion erhält der Agent bei (Randlov et al., 1998) immer dann eine Belohnung, wenn er das Fahrrad in Richtung des Zielpunktes fährt. Daraus erwächst die Strategie, in Kreisen um den Startpunkt herumzufahren. Auf der Hälfte der Kreisfahrt, die dem Zielpunkt zugewandt ist, kann so praktisch beliebig viel Belohnung angehäuft werden. In der finalen Variante, ist die Belohnungsfunktion dahingehend angepasst, dass bei Entfernung vom Zielpunkt die betragsgleiche Menge der Belohnung abgezogen wird.

Ein weiteres Beispiel beschreiben (Kober et al., 2013) in ihrer Studie. Ein Roboterarm wird in dem Kinderspiel *Bilboquet* trainiert. Ziel des Spiels ist es, einen Ball, der an einem Faden vom Boden eines Bechers herabhängt, durch Schwingen in den Becher

zu befördern. Mit der ersten Belohnungsfunktion erhält der Agent mehr Belohnung, je näher der Ball dem Boden im inneren des Bechers während eines Versuchs war. Als Resultat schwingt der Roboterarm den Becher so, dass der Ball von unten an den Boden des Bechers schlägt, aber niemals darin gefangen wird. Im Sinne der Belohnungsfunktion ist dieses Verhalten nahezu optimal.

Bei diesen drei Beispielen geht es nicht um die Erkenntnis, dass eine ungünstige Belohnungsfunktion nicht zum gewünschten Ergebnis führt. Vielmehr ergibt sich hieraus, zusätzlich zu den Entscheidungen aus Abschnitt 5.2, eine weitere Fragestellung für den Gestaltungsprozess. Es erscheint ratsam, beim manuellen Entwurf einer Belohnungsfunktion nicht nur nach geeigneten Parametern und Gewichtungen, sondern auch aktiv nach Wegen zu suchen, wie die Funktion entgegen dem vorgesehen Zweck ausgenutzt werden kann. Im Fall des Roboterarms ist die Belohnungsfunktion nicht grundsätzlich ungeeignet, aber die optimale Lösung nur unwahrscheinlich durch Zufall zu entdecken. In den anderen Beispielen hat der Agent ein optimales, aber unerwünschtes Verhalten entwickelt. Eine Überprüfung der Möglichkeit, unbegrenzte Belohnung zu sammeln, hätte dies verhindern können.

5.5. Strategie für den Funktionsentwurf

Die Zielsetzung hinsichtlich der zu lernenden Bewegungsform orientiert sich an dem Anforderungsprofil des Roboters aus Abschnitt 4.1. Demnach ist das Primärziel eine stabile Form der Fortbewegung zu lernen, den Parcours zur überwinden und den Endbereich des Geländes zu erreichen. Dies deckt den ersten Teil des Profils ab. Darüber hinaus können aus der zweiten und dritten Anforderung folgende Sekundärziele abgeleitet werden:

- Energiesparende Bewegungsstrategie
- Harmonische Bewegungen
- Kollisionsvermeidung

Das Anstreben einer energieeffizienten Fortbewegung ist nach dem aktuellen Stand der Technik eine Notwendigkeit, um akzeptable, autonome Laufzeiten zu erreichen. Unter dem Begriff der harmonischen Bewegungen sind Bewegungsmuster zusammengefasst, die in der Gegenwart von Menschen annehmbar sind. Sie sollten zyklisch, fließend, dosiert und damit für Außenstehende vorhersehbar sein. Abrupte und chaotische Bewegungen, schnelle Richtungswechsel sowie kraftvolles Aufstampfen würden als irritierend oder bedrohlich wahrgenommen. Die Vermeidung von Kollisionen dient

dem Schutz des Roboters und der Umwelt, in der er sich bewegt. Hierbei ist die Einschränkung zu machen, dass in dieser Arbeit mit einem statischen Trainingsgelände gearbeitet wird. Der Umgang mit einer dynamischen Umgebung würde die Anforderungen an die Sensorik grundlegend verändern und an dieser Stelle zu weit führen.

Für eine extensive Erforschung der durch die Belohnungsfunktion gegebenen Einflussmöglichkeiten, wäre eine Variation von Funktionen über alle fünf Dimensionen aus Abschnitt 5.2 erstrebenswert. Allerdings ergäben sich bereits bei einer Variation über die ersten vier Dimensionen mit vier Varianten pro Dimension 256 mögliche Belohnungsfunktionen. Eine Größenordnung die im Rahmen dieser Arbeit nicht zu realisieren ist. Die Strategie für den Entwurf der Belohnungsfunktionen wird deshalb an die in diesem Kapitel diskutierten Aspekte angepasst.

Bei der Auswahl der Parameter werden solche bevorzugt, die keinen direkten Bezug zur Kinematik des Roboters haben. Der Trainingsprozess soll, im Sinne der Grundidee des RL, offen und flexibel gehalten werden. Des Weiteren könnte mit einer erfolgreichen neutralen Belohnungsfunktion die Übertragbarkeit auf andere Kinematiken erforscht werden.

Bezüglich der Parameteranzahl entsteht ein Interessenskonflikt mit der ersten Dimension. Um dem Agenten größtmögliche Freiheiten zu gewähren, ist das von (Heess; T.B. et al., 2017) propagierte Minimalprinzip anzuwenden. Dem gegenüber steht die Möglichkeit, den Trainingsprozess mit mehr Leitinformationen zu beschleunigen. Da eine größere Anzahl der Parameter auch gleichzeitig mit einem erhöhten Risiko einhergeht, ungeeignete Parameter zu verwenden, ist das Minimalprinzip vorzuziehen.

Die Belohnungsfunktionen werden deshalb in einem iterativen Prozess entworfen, erprobt und anschließend weiterentwickelt. Im Mittelpunkt steht zunächst die Erfüllung des Primärziels. Wenn dies erreicht wird, erfolgt eine Beurteilung der Policy hinsichtlich des Einhaltens der Sekundärziele. Bei Bedarf werden in der Folge weitere Komponenten zur Belohnungsfunktion hinzugefügt. Um die Struktur der Arbeit zu wahren, sind dennoch alle erprobten Funktionen am Ende diese Kapitels abgebildet.

Alle Terme der Belohnungsfunktion werden entweder konstant oder mit einer linearen Abhängigkeit zu einem Parameter gewählt. Die Priorisierung eines Parameters mithilfe der Gewichtung ist dadurch besser abzuschätzen, als beispielsweise bei einer Vermischung von Linear- und Exponentialfunktionen.

Die Gewichtung der Parameter untereinander erfolgt in Anlehnung an die erfolgreichen Beispiele aus Abschnitt 5.3. Komponenten, die auf die Erfüllung des Primärziels ausgerichtet sind, werden erheblich stärker gewichtet als die der nachrangigen Ziele. Die Summe der Belohnung, die für Parameter der Sekundärziele erreichbar ist, wird auf

5 % der maximal möglichen Belohnung für das Primärziel skaliert. Die Sekundärziele werden untereinander gleich gewichtet.

Eine Zeitabhängigkeit der Belohnungsfunktionen ist nicht vorgesehen. Um einen sicheren Nutzen daraus ziehen zu können, müssten zunächst geeignete Zwischenziele ermittelt werden, die den Lernprozess begünstigen.

5.6. Belohnungsfunktionen für Versuche

Das Primärziel wird, analog zu den diskutierten Arbeiten, anhand des Geschwindigkeitsvektors bewertet:

$$r = \vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{5.14}$$

Für die erdgebundene Fortbewegung ist die vertikale Komponente der Vektoren bedeutungslos. Aufgrund des Längen- und Breitenverhältnisses des Parcours, besitzt die laterale Komponente nur in unmittelbarer Nähe des Zielpunkts eine steuernde Wirkung sodass die Belohnung ausschließlich durch die Komponente in Längsrichtung des Trainingsgeländes ausgedrückt werden kann:

$$r = v_x \tag{5.15}$$

Als Startpunkt der Untersuchung ist diese Belohnungsfunktion grundsätzlich ausreichend. Um den bekannten Problemen lokaler Optima vorzubeugen, werden dennoch zwei Parameter zur Stabilisierung des Lernprozesses hinzugefügt. Der Agent erhält zusätzliche Belohnung, wenn er nicht mit maximalen Drehmomenten arbeitet und einen konstanten Bonus für längeres Überleben (Gleichung 5.16). Um die Struktur zu wahren sind die im weiteren Verlauf entwickelten Belohnungsfunktionen sind nachstehend aufgeführt (Gleichungen 5.17-5.19). Sie werden im Zuge der Trainingsauswertung näher besprochen.

$$\begin{aligned} r &= v_x + c_1 + c_2 \sum_{i=1}^{12} (1 - u_i^2) \\ c_i &= \begin{bmatrix} 0{,}0375 & 0{,}003125 \end{bmatrix} \end{aligned} \tag{5.16}$$

$$
\begin{aligned}
r &= v_x + c_1 + c_2 \sum_{i=1}^{12}(1 - u_i^2) + c_3 I[h_{min} < h < h_{max}](1 - \frac{|h - h_0|}{h_{max} - h_{min}}) \\
c_i &= \begin{bmatrix} 0{,}025 & 0{,}002083 & 0{,}025 \end{bmatrix}
\end{aligned}
\tag{5.17}
$$

$$
\begin{aligned}
r &= v_x + c_1 + c_2 \sum_{i=1}^{12}(1 - u_i^2) + c_3 I[h_{min} < h < h_{max}](1 - \frac{|h - h_0|}{h_{max} - h_{min}}) \\
&\quad + c_4 I[F_{cont} > 2] \\
c_i &= \begin{bmatrix} 0{,}0188 & 0{,}0016 & 0{,}0188 & 0{,}0188 \end{bmatrix}
\end{aligned}
\tag{5.18}
$$

$$
\begin{aligned}
r &= v_x + c_1 + c_2 \sum_{i=1}^{12}(1 - u_i^2) + c_3 I[h_{min} < h < h_{max}](1 - \frac{|h - h_0|}{h_{max} - h_{min}}) \\
&\quad + c_4 I[F_{cont} > 2] + c_5 I[f = 1] \\
c_i &= \begin{bmatrix} 0{,}0188 & 0{,}0016 & 0{,}0188 & 0{,}0188 & -135 \end{bmatrix}
\end{aligned}
\tag{5.19}
$$

Parameter:

v_x	=	Geschwindigkeit
u_i	=	Ausgangssignal des Agenten
h, h_{min}, h_{max}	=	aktuelle Höhe und Bereichsgrenzen
F_{cont}	=	Anzahl der Füße mit Bodenkontakt

6. Ergebnisse und Analysen

In diesem Kapitel sind die Trainingsergebnisse zu den Belohnungsfunktionen aus Abschnitt 5.6 dokumentiert. Die erlernten Bewegungsmuster werden zunächst für jeden Trainingsdurchlauf beschrieben. Gemeinsamkeiten und Unterschiede zwischen den Versuchen mit derselben Belohnungsfunktion, sowie auffällige Merkmale, werden analysiert. Wo ein Vergleich sinnvoll erscheint, wird eine Gegenüberstellung zwischen verschiedenen Funktionen vorgenommen. Jeder Abschnitt, der einer Belohnungsfunktion zugeordnet ist, schließt mit ihrer Weiterentwicklung für die nächste Versuchsreihe. Am Ende des Kapitels werden die gewonnenen Erkenntnisse zusammengefasst und es erfolgt eine kritische Betrachtung der gesamten Untersuchung.

6.1. Belohnungsfunktion 1

Abbildung 6.1 zeigt die Bewegungsfolge nach dem ersten Training. Der Roboter richtet sich auf, indem alle Beine durchgestreckt werden. Dabei ist die Bewegung so kraftvoll, dass alle vier Füße den Bodenkontakt für kurze Zeit verlieren. Die Hinterbeine verbleiben gestreckt und setzen erneut auf dem Boden auf. Gleichzeitig werden die Vorderbeine angewinkelt und an den Körper herangezogen. In dieser Konstellation wird kurzzeitig Geschwindigkeit aufgebaut. Der Roboter kippt schließlich nach vorne und stürzt unkontrolliert auf die Seite, weil die Vorderbeine in der angezogenen Haltung verbleiben.

Im zweiten Versuch beginnt die Bewegung ebenfalls mit einem Absprung aus der Startposition (Abbildung 6.2). Die Vorderbeine werden nach außen geklappt, während das hintere Beinpaar bereits gestreckt wird. Der Roboter kippt dadurch leicht nach vorne und verstärkt die Vorwärtsbewegung, indem nun alle vier Beine kraftvoll durchgestreckt werden. Ähnlich wie im ersten Versuch verlieren alle Füße den Bodenkontakt. Während der kurzen Flugphase werden die gestreckten Vorder- und Hinterbeine leicht vom Körper abgespreizt, was eine stabile Landung ermöglicht. Anschließend kippt der Roboter leicht über die angewinkelten Vorderbeine. Die Füße der noch immer gestreckten Hinterbeine verlassen erneut den Boden. Der Roboter versucht durch erneutes Strecken der Vorderbeine einen zweiten Absprung auszuführen und verharrt in dieser Pose bis der Sturz eintritt.

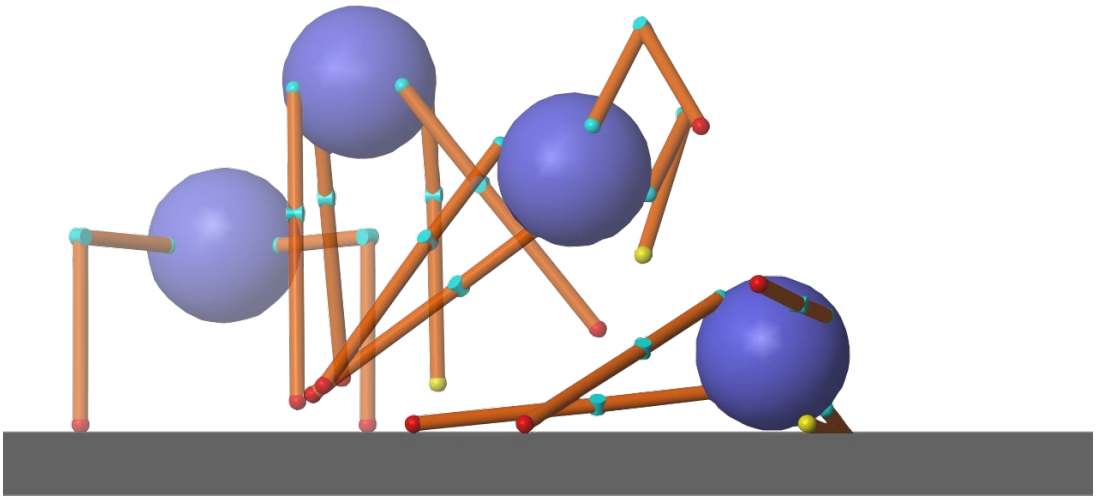

Abbildung 6.1.: Belohnungsfunktion 1, Bewegungssequenz des ersten Trainings.

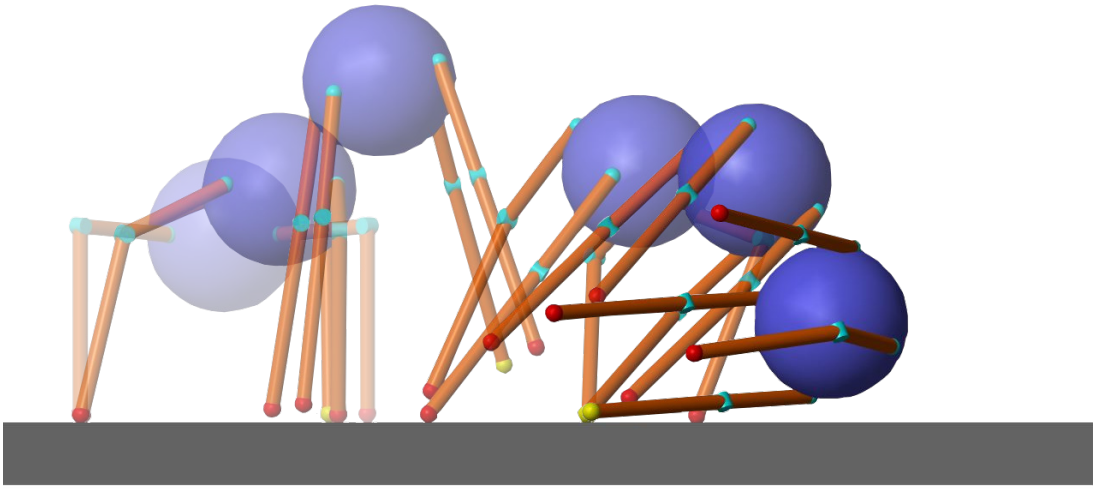

Abbildung 6.2.: Belohnungsfunktion 1, Bewegungssequenz des zweiten Trainings.

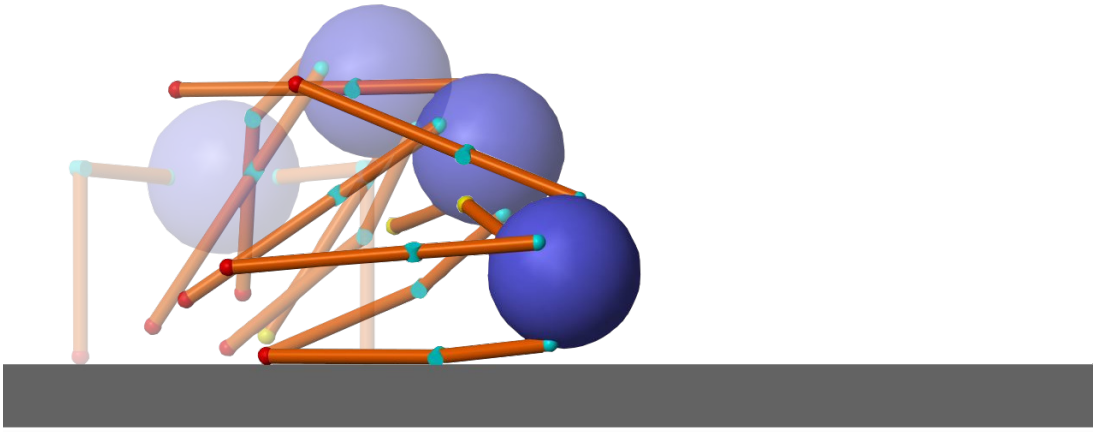

Abbildung 6.3.: Belohnungsfunktion 1, Bewegungssequenz des dritten Trainings.

Das Ergebnis des dritten Durchlaufs ist der Endphase der zweiten Bewegungssequenz ähnlich. Der Roboter verlässt die Startposition sprunghaft durch Strecken der Beine. Infolge einer Kippbewegung und keiner weiteren Veränderung der Beinhaltung nach dem Absprung, führt dies unmittelbar zu einem Sturz (Abbildung 6.3).

Die Bewegungsfolge des vierten Trainings ist in Abbildung 6.4 dargestellt. Die Hinterbeine werden durchgestreckt, während die vorderen zunächst in der angewinkelten Ausgangshaltung verbleiben. Der daraus resultierende Vortrieb ist stärker in horizontaler Richtung ausgeprägt als die vorherigen Versuche. Das vordere linke Bein setzt nach kurzer Schwingphase wieder vor dem Körper auf dem Boden auf. Die Hinterbeine verbleiben gestreckt und das vordere rechte Bein wird dem Linken nicht nachgezogen, sodass durch die Kollision des Kniegelenks mit dem Boden der Sturz eintritt.

In keinem der Versuche hat der Roboter eine stabile Form der Fortbewegung erlernt. Insgesamt kann das Verhalten als Ausnutzung lokaler Optima eingestuft werden. Durch die Sprungbewegung wird für kurze Zeit viel Geschwindigkeit in Richtung des Ziels aufgebaut und dafür ein relativ hoher Belohnungswert generiert. Die Kurzfristigkeit dieser Bewegungsplanung wird besonders im ersten, dritten und vierten Versuch deutlich. Nachdem der Roboter Geschwindigkeit aufgenommen hat, unternimmt er keine weiteren Versuche die Bewegung fortzusetzen oder zu stabilisieren, sondern verharrt in der letzten Pose, bis die Sequenz durch eine der Sturzbedingungen beendet wird. Die zweite Bewegungsfolge zeigt mit einer stabilen Landung nach dem anfänglichen Sprung zunächst einen vielversprechenden Ansatz. In der Folge wird jedoch auch hier das lokale Optimum ausgenutzt, indem sich der Roboter nach vorne wirft. Diese Bewegungsstra-

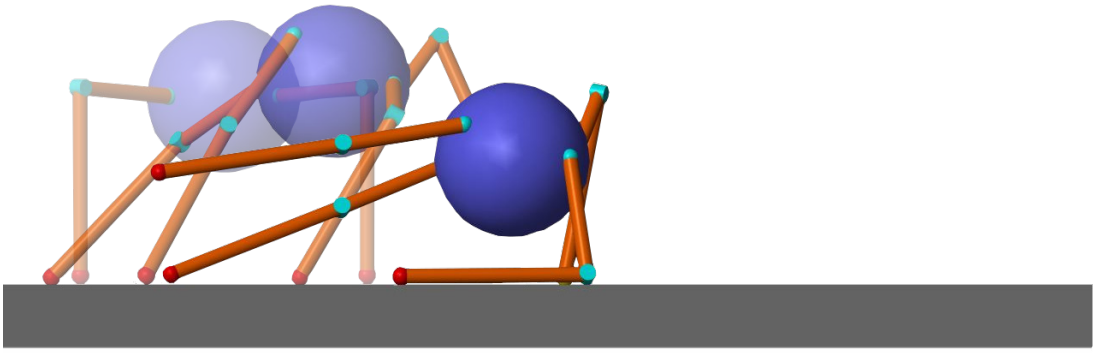

Abbildung 6.4.: Belohnungsfunktion 1, Bewegungssequenz des vierten Trainings.

tegie ist insofern bemerkenswert, als das der Parameter c_1 der Belohnungsfunktion explizit dafür vorgesehen ist, dieses Verhalten zu unterbinden.

Des Weiteren ist festzustellen, dass der Agent die günstigen Eigenschaften der Kinematik, hinsichtlich ihrer Stabilität in der Grundhaltung, zu beinahe keinem Zeitpunkt in der Bewegungsplanung berücksichtigt. Lediglich im zweiten Versuch befindet sich der Körper während der Landung zwischen den leicht abgespreizten Beinen. In allen anderen Fällen wird diese Stabilität durch das Strecken der Hinterbeine und das seitliche Aufklappen der Vorderbeine zugunsten eines größeren Vortriebs aufgegeben.

Für die zweite Versuchsreihe wird die Belohnungsfunktion um einen Parameter h für die Höhe des Körperschwerpunkts über dem Boden erweitert. Der Agent erhält zusätzliche Belohnung, wenn die Position des Körpers in einem Bereich von ± 100 mm um die Starthöhe von 600 mm liegt. Innerhalb dieses Bereichs wächst die Belohnung linear und erreicht ein Maximum bei Bewahrung der Starthöhe (vgl. Abschnitt 5.6).

Mit dem Höhenparameter wird das Ziel verfolgt, dem Agenten einen zusätzlichen Anreiz zu größerer Stabilität zu liefern. Er soll dazu angehalten werden, die Bewegung des Roboters nicht durch das Strecken der Beine mit einem kraftvollen Sprung zu beginnen. Die angestrebte Position des Körpers ist nur dann einzuhalten, wenn die Beine seitlich dazu abgespreizt werden. Mit einem anfänglichen Sprung, wie er bisher zu beobachten ist, wird der Korridor für diese zusätzliche Belohnung verlassen.

Ebenso soll die Gefahr des Kippens reduziert werden. Wenn der Agent bemüht ist, die geforderte Höhe nicht zu unterschreiten, kann der Kontakt das Körpers mit dem Boden

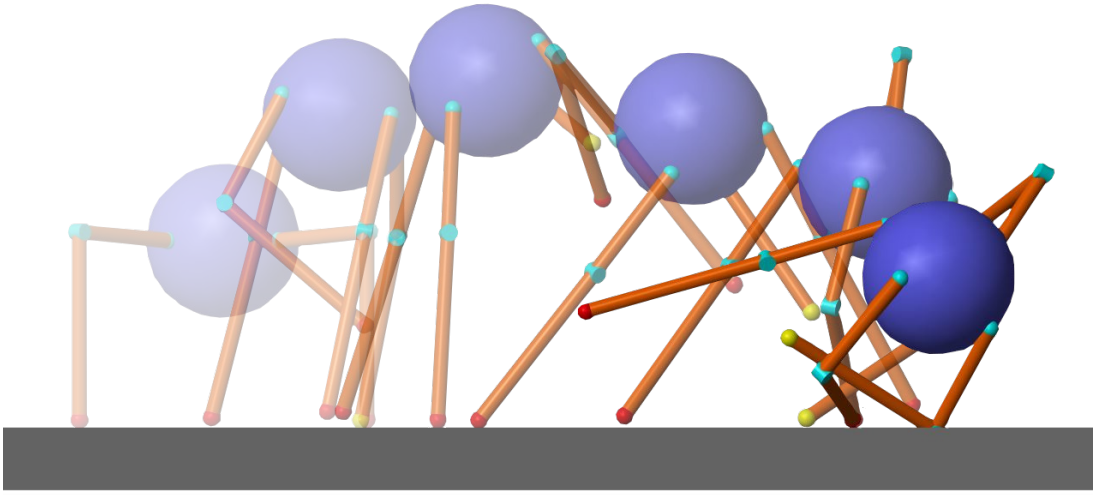

Abbildung 6.5.: Belohnungsfunktion 2, Bewegungssequenz des ersten Trainings.

als Sturzursache vermieden werden. Wenn es gelingt, den Agenten auf diesem Weg zu mehr Stabilität anzuleiten, verlängert dies gleichzeitig die Überlebenszeit in einer Episode und bietet mehr Zeit, um sichere Bewegungen zu erproben.

6.2. Belohnungsfunktion 2

Die im ersten Training mit der zweiten Belohnungsfunktion erlernte Bewegungsform ist in Teilen vergleichbar mit der ersten Sequenz des vorherigen Abschnitts. Sie beginnt mit dem sprunghaften Aufrichten des Roboters und dem kurzzeitigen Abheben aller Füße vom Boden. Noch während der kurzen Flugphase werden mit jedem Bein individuelle Bewegungen ausgeführt, die eine Schrittfolge andeuten. Durch die schlechte Koordination der Bewegung verliert der Roboter jedoch das Gleichgewicht und es kommt zum Sturz (Abbildung 6.5).

Die zweite Bewegungssequenz beginnt mit einem leichten, asymmetrischen Abstoßen durch die Hinterbeine, wodurch Geschwindigkeit aufgebaut wird. Mit dem vorderen rechten Bein wird ein koordinierter Schritt in Bewegungsrichtung ausgeführt. Währenddessen schwingt das hintere linke Bein ausgestreckt über den Torso hinweg. Zunächst baut der Roboter durch die auftretenden Trägheitskräfte zusätzliche Geschwindigkeit auf. In der Folge führt dies zu einer Kippbewegung des ganzen Körpers und schließlich zum Sturz (Abbildung 6.6).

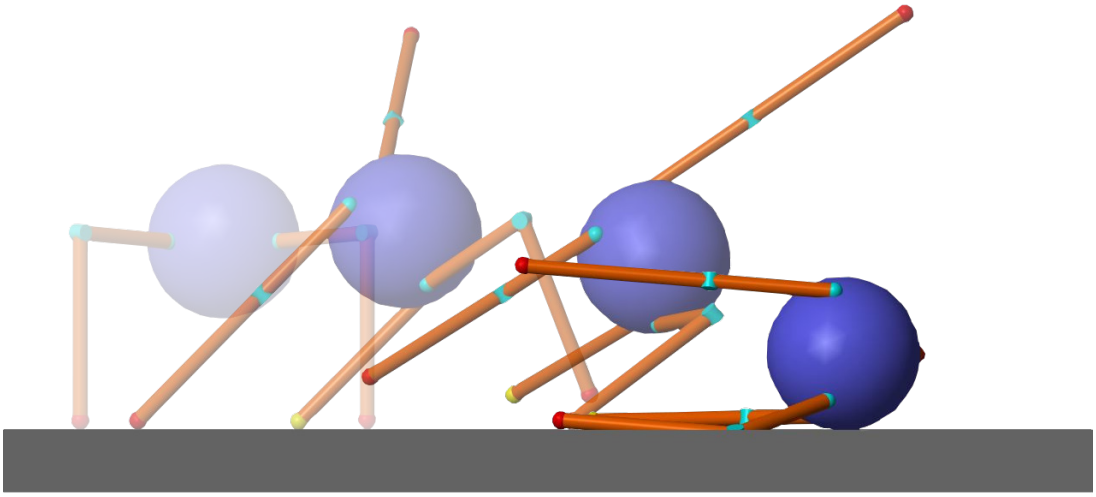

Abbildung 6.6.: Belohnungsfunktion 2, Bewegungssequenz des zweiten Trainings.

Im dritten Versuch wird zunächst das vordere rechte Bein angezogen. Anschließend werden alle Beine in einer kraftvoller Weise ausgestreckt. Dies wirft den Roboter in einer sprungartigen Bewegung nach vorne und es folgt unmittelbar der Sturz (Abbildung 6.7).

Die Bewegungsfolge des letzten Trainings ist in Abbildung 6.8 dargestellt. Aus einer leichten Hocke gegenüber der Startposition streckt der Roboter die Hinterbeine aus, um Geschwindigkeit in horizontaler Richtung aufzubauen. Der flache Winkel unter dem er sich gegen den Boden abstößt führt dazu, dass die Füße abgleiten und nur wenig Vortrieb erzeugt wird. Die Vorderbeine werden gleichzeitig unter dem Körper eingeklappt, sodass es zum Sturz auf die Kniegelenke kommt.

Die Trainingsreihe der zweiten Belohnungsfunktion hat, ebenso wie die der ersten, keine stabilen Gangarten hervorgebracht. Auffällig ist jedoch, dass die Trajektorie des Körpers im zweiten Versuch erkennbar flacher verläuft, als im Ersten und Dritten. In den Abbildungen 6.9 und 6.10 sind die Höhenverläufe des Körperschwerpunkts über dem Boden für die Versuchsreihe der ersten und zweiten Belohnungsfunktion gegenübergestellt. Demnach nutzt der Agent im zweiten Fall dieser Reihe die Möglichkeit, durch Einhalten der angestrebten Höhe zusätzliche Belohnung zu erhalten (Abbildung 6.11).

Ein besonderes Merkmal des zweiten Versuchs ist der Aufbau von Geschwindigkeit durch Ausnutzung von Trägheitseffekten. Die Einhaltung der Zielhöhe nimmt dem Agenten die Möglichkeit, größere Belohnung für seine Geschwindigkeit durch einen kraftvollen Absprung zu forcieren. Als Ausgleich dazu nutzt er das Überschwingen eines Beins,

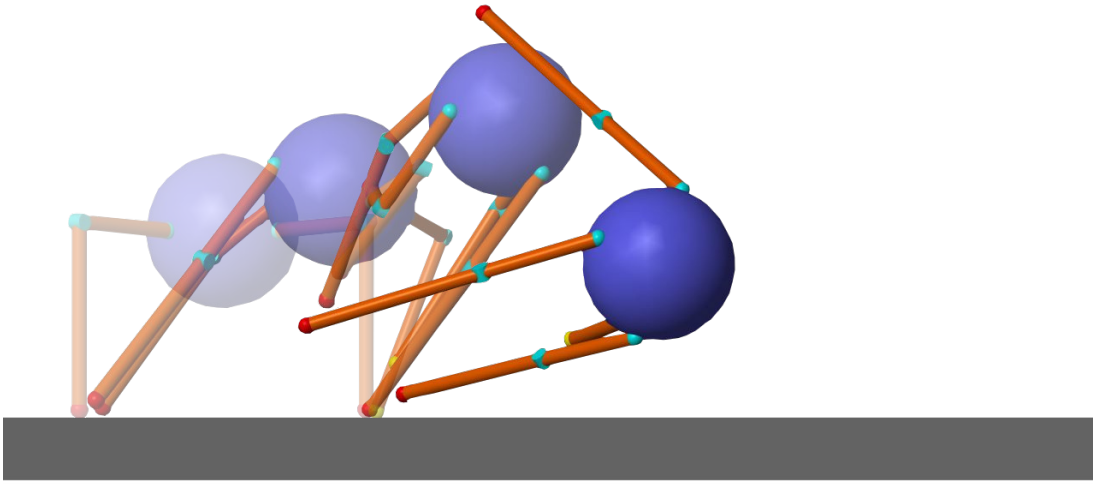

Abbildung 6.7.: Belohnungsfunktion 2, Bewegungssequenz des dritten Trainings.

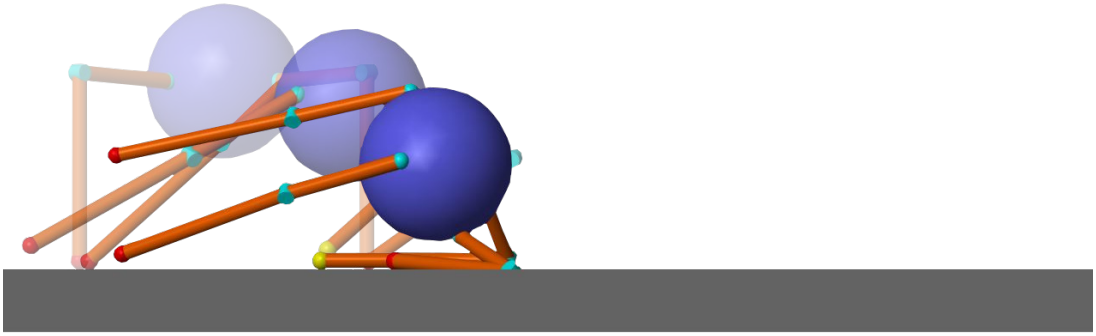

Abbildung 6.8.: Belohnungsfunktion 2, Bewegungssequenz des vierten Trainings.

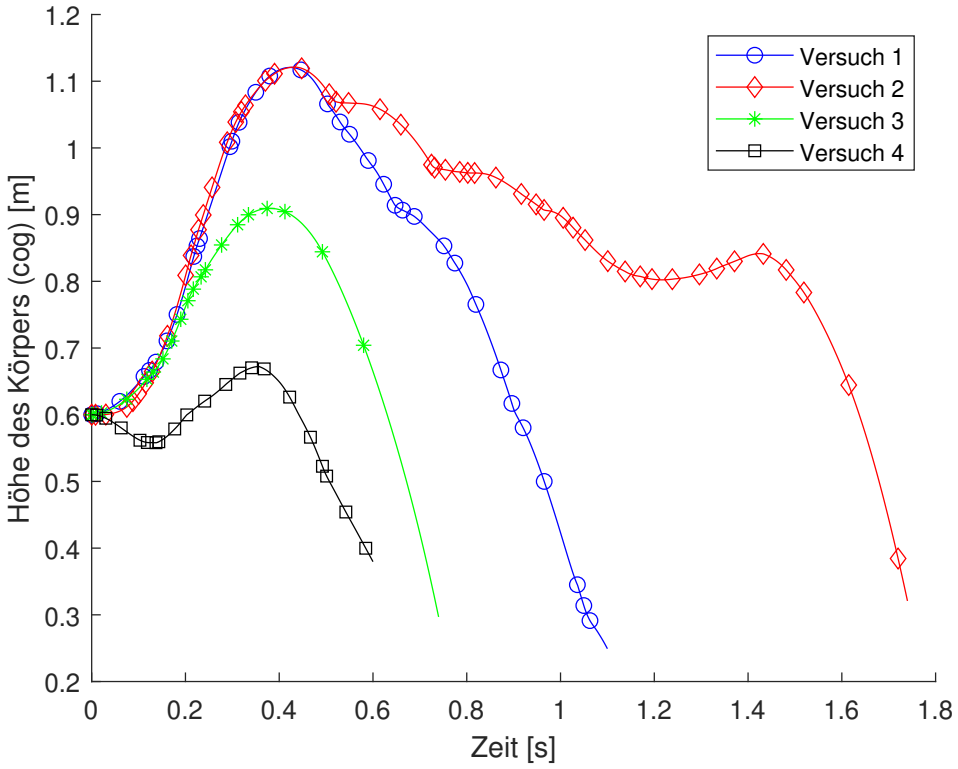

Abbildung 6.9.: Belohnungsfunktion 1, Höhen-Trajektorien des Körperschwerpunkts.

um Vortrieb zu erzeugen. Einerseits hat also der Parameter zur Zielhöhe des Körpers den gewünschten Effekt und verhindert in diesem Fall den unkontrollierten Sprung des Roboters. Andererseits führt die alternative Lösung des Agenten für den Aufbau von Geschwindigkeit ebenfalls zum Sturz. Der Versuch ist vergleichbar mit Bewegungssequenzen, in denen sich der Roboter unmittelbar nach vorne wirft. Der Unterschied ist, dass er diesmal eine flache Trajektorie für dieses Manöver anstrebt.

Gegenüber der ersten Belohnungsfunktion ist die Vielfalt der beobachteten Bewegungen mit dieser Funktion insgesamt größer. Die Bewegungsformen sind jedoch auch hier als Varianten von lokalen Optima einzustufen. Die Lernkurven in Abbildung 6.13 zeigen im Vergleich mit der ersten Belohnungsfunktion (Abbildung 6.12) keinen signifikanten Unterschied hinsichtlich der erhaltenen Belohnung. Wenngleich der Höhenparameter einen messbaren Einfluss auf die gelernten Bewegungsformen hat und der erwartete Effekt zu beobachten ist, so hat er nicht zu der gewünschten Stabilisierung geführt.

Um das lokale Optimum zu überwinden, benötigt der Agent weiterhin Erfahrungen mit längeren Bewegungssequenzen. Die Belohnungsfunktion wird deshalb um einen Parameter ergänzt, der die Anzahl der im Kontakt mit dem Boden befindlichen Füße berücksichtigt. In der nächsten Versuchsreihe erhält der Agent damit zusätzliche Belohnung,

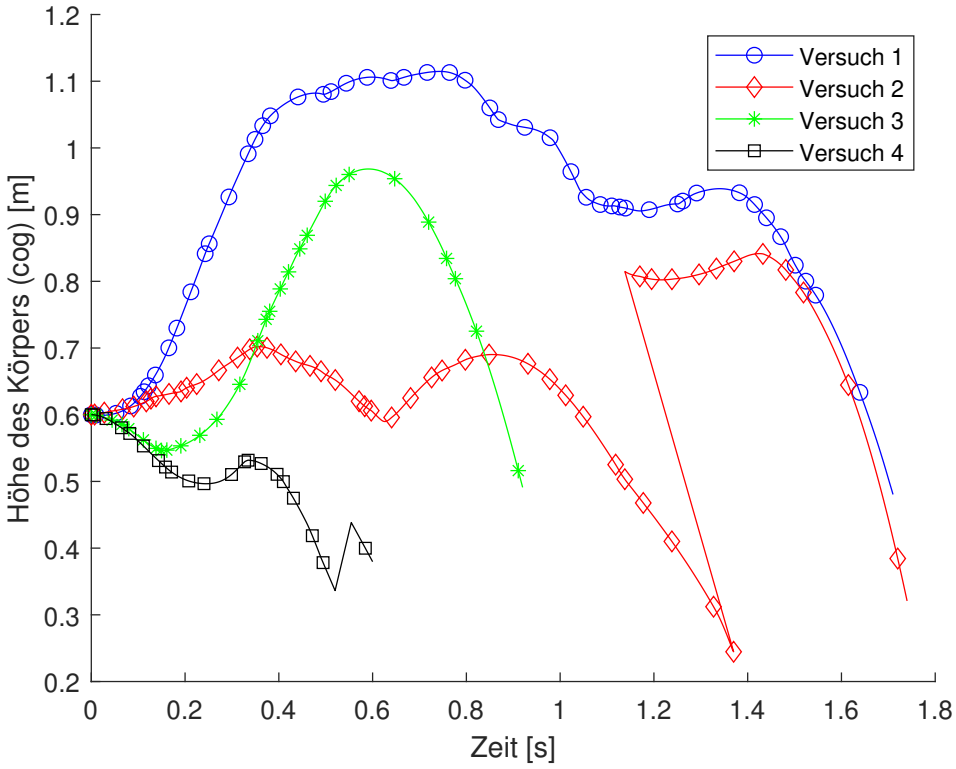

Abbildung 6.10.: Belohnungsfunktion 2, Höhen-Trajektorien des Körperschwerpunkts.

wenn mehr als zwei Füße den Boden berühren (vgl. Abschnitt 5.6).

Im Vergleich zu den Parametern der ersten beiden Belohnungsfunktionen, ist dies ein harter Eingriff in die Bewegungsfreiheit. Die erste Funktion liefert dem Agenten praktisch keine Vorgaben, wie die Aufgabe der Fortbewegung zu bewältigen ist. In der zweiten beeinflusst der Höhenparameter zwar die Trajektorie des Körpers, beschränkt dabei aber nicht die Bewegungsfreiheit der Beine. Mit dem neuen Parameter sind nur solche Bewegungen mit zusätzlicher Belohnung verknüpft, bei denen sich nur ein einzelnes Bein in der Flugphase befindet, während die Übrigen den Bodenkontakt halten. Die Anpassung ist angelehnt an die stabilste und koordinativ einfachste Gangart von Vierfüßlern, bei der jeweils nur ein Bein neu positioniert wird, während die verbleibenden drei Beine einen sicheren Stand gewährleisten.

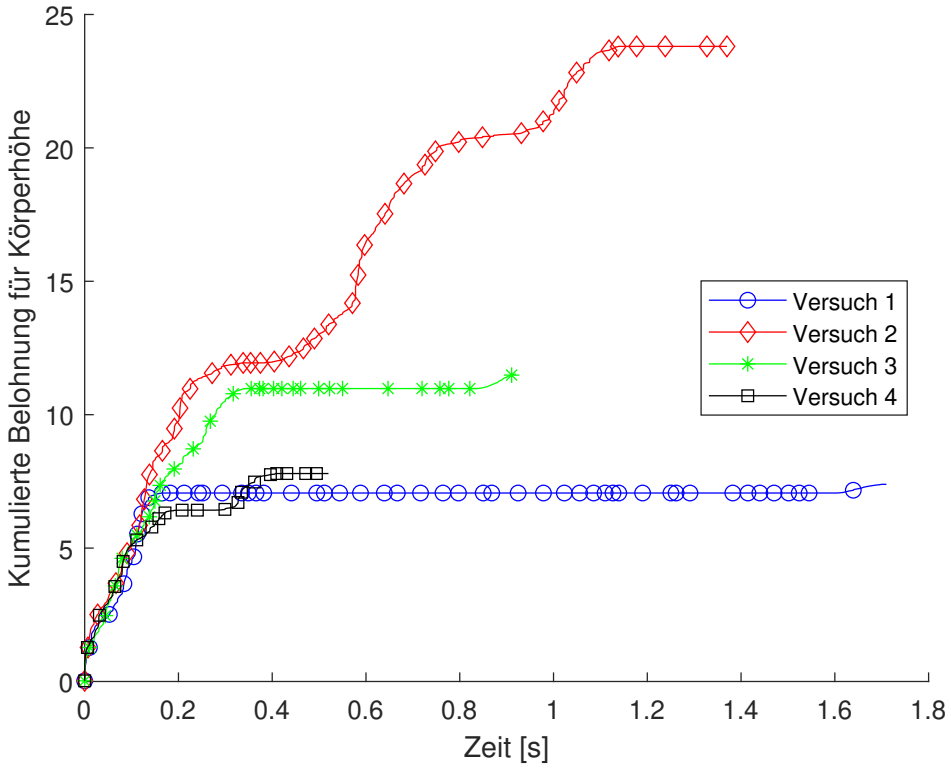

Abbildung 6.11.: Belohnungsfunktion 2, Belohnung für Einhaltung der Zielhöhe.

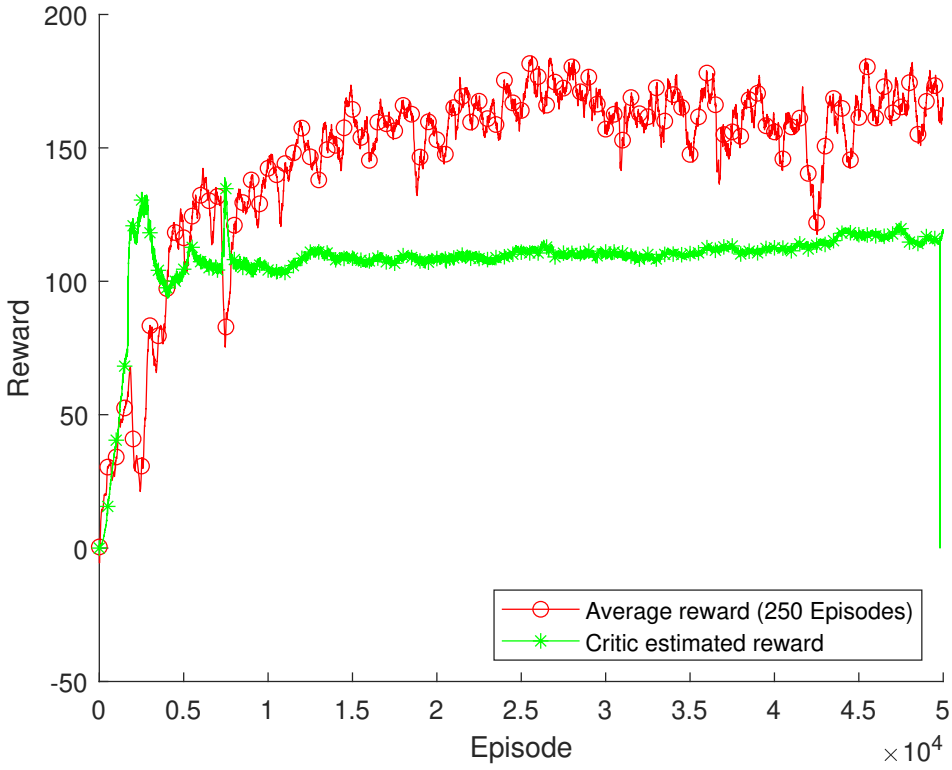

Abbildung 6.12.: Belohnungsfunktion 1, Trainingsstatistik des zweiten Versuchs.

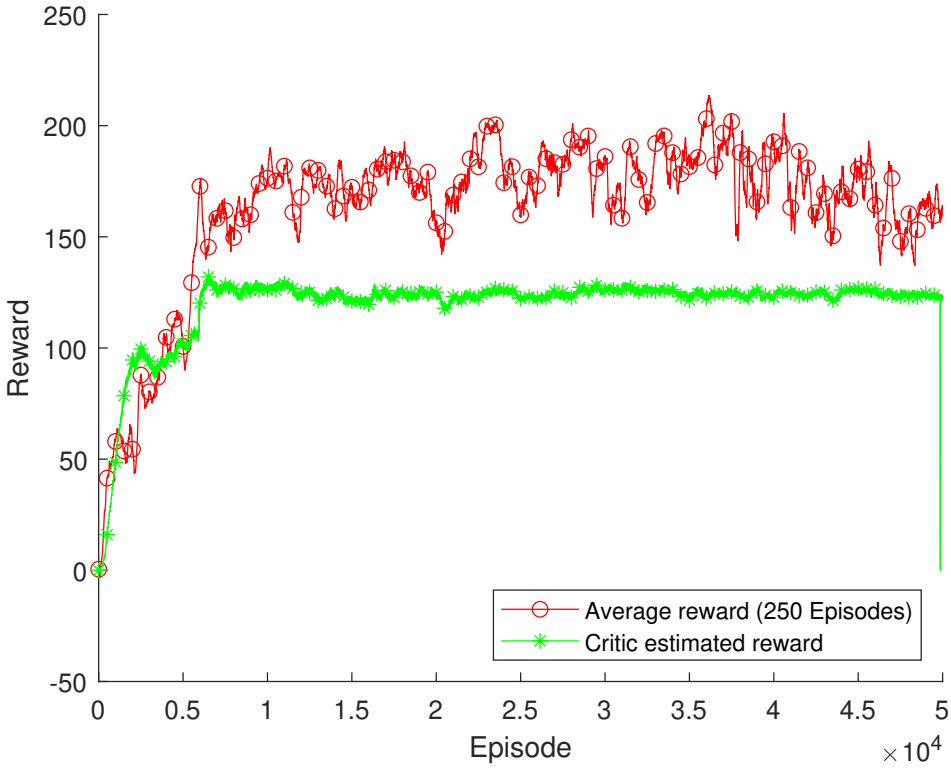

Abbildung 6.13.: Belohnungsfunktion 2, Trainingsstatistik des zweiten Versuchs.

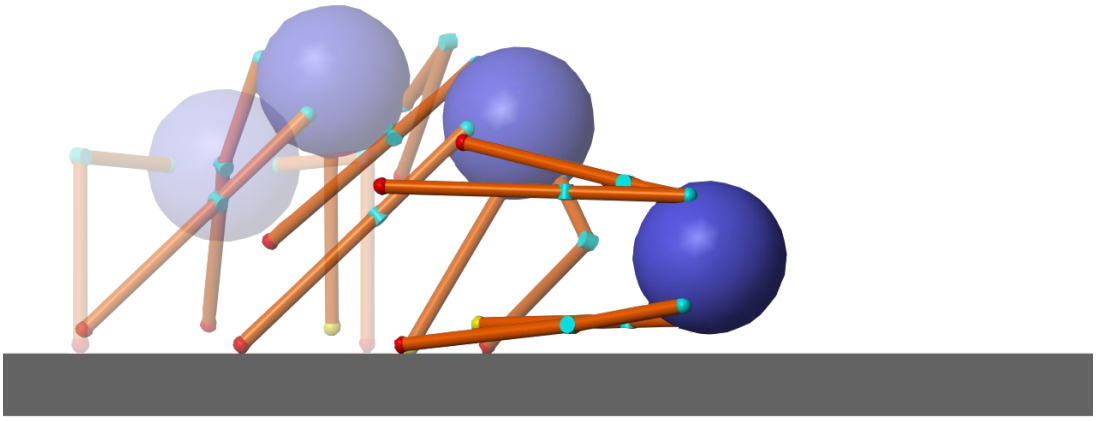

Abbildung 6.14.: Belohnungsfunktion 3, Bewegungssequenz des ersten Trainings.

6.3. Belohnungsfunktion 3

In der ersten Bewegungssequenz dieser Trainingsreihe (Abbildung 6.14) beginnt die Bewegung des Roboters mit dem Aufrichten aus der Grundhaltung. Sie ist vergleichbar mit dem dritten Versuch aus der zweiten Trainingsreihe, wenngleich die Bewegungen weniger kraftvoll sind. Auch hier baut der Roboter sprunghaft Geschwindigkeit auf und stürzt unmittelbar im Anschluss daran.

Im zweiten Versuch ist eine Bewegungsfolge zu beobachten, die den Ergebnissen der ersten Belohnungsfunktion ähnlich ist (Abbildung 6.15). Das Durchstrecken der Beine aus der Startpose heraus führt in einem Sprung zum Aufbau von Geschwindigkeit. Nach der Landung wird mit dem vorderen rechten Bein der Versuch unternommen, einen weiteren Schritt auszuführen. Das Verharren der übrigen Beine in gestreckter Haltung führt dabei zum Verlust der Stabilität und einem Kippen des Roboters bis hin zum Sturz.

Ebenfalls mit einem Sprung beginnt die dritte Bewegungssequenz (Abbildung 6.16). Wie schon in den gelernten Bewegungen der ersten Belohnungsfunktion beobachtet, werden dabei die Vorderbeine nach außen geklappt. Bei weiterhin ausgestreckten Beinen in Verbindung mit der erreichten Geschwindigkeit, führt dies zum Kippen des Roboters und dem anschließenden Sturz.

Im vierten Versuch springt der Roboter mit einer flachen Trajektorie aus der Startposition ab und stürzt unmittelbar zu Boden (Abbildung 6.17). Die Bewegungssequenz ist direkt vergleichbar mit dem vierten Versuch der zweiten Belohnungsfunktion. Auch hier

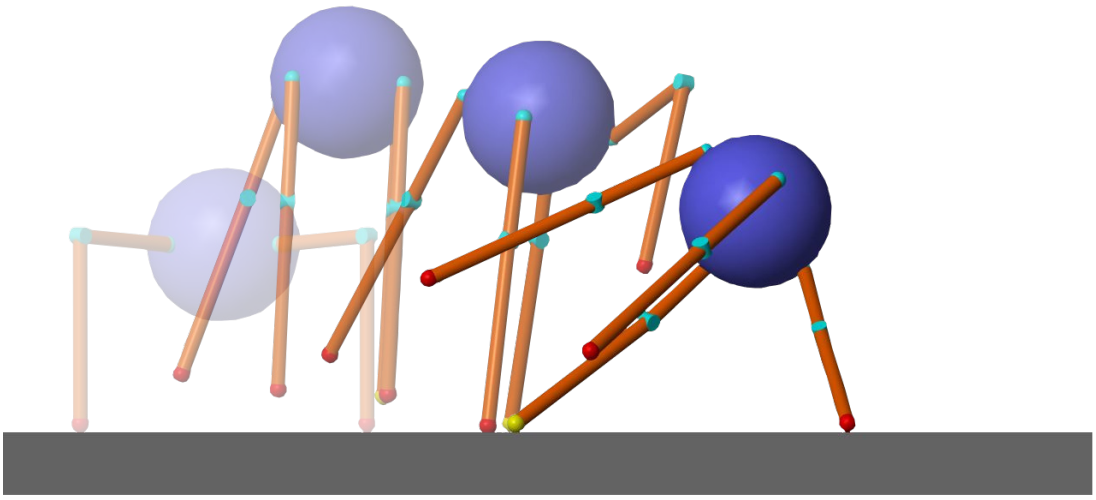

Abbildung 6.15.: Belohnungsfunktion 3, Bewegungssequenz des zweiten Trainings.

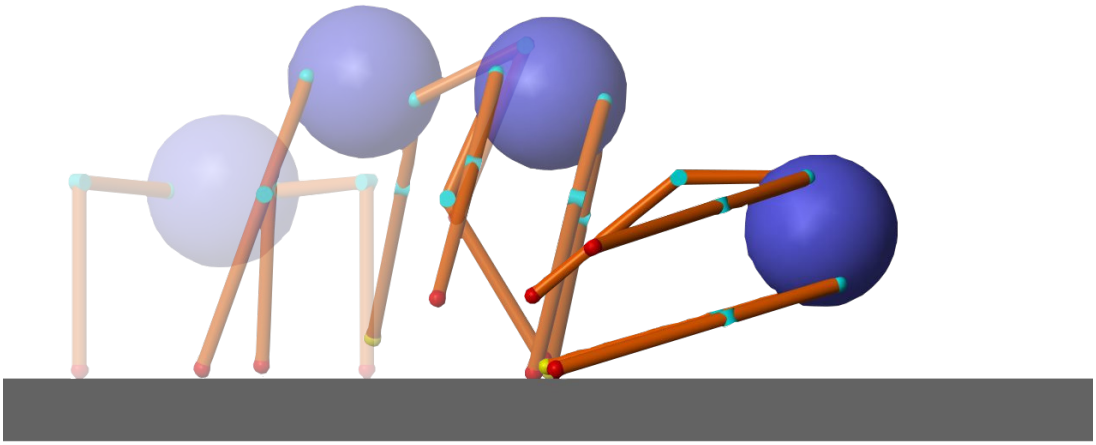

Abbildung 6.16.: Belohnungsfunktion 3, Bewegungssequenz des dritten Trainings.

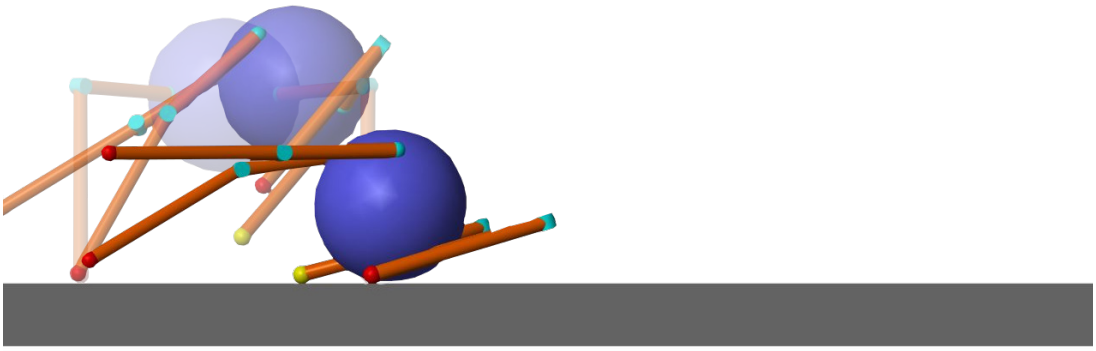

Abbildung 6.17.: Belohnungsfunktion 3, Bewegungssequenz des vierten Trainings.

gleiten die hinteren Füße ab, während die Vorderbeine unter dem Körper angezogen werden.

In den Abbildungen 6.18 und 6.19 sind die Belohnungen infolge des Bodenkontaktes der Füße für diese Versuchsreihe gegenübergestellt, mit den Belohnungen, die für die Bewegungssequenzen der zweiten Belohnungsfunktion diesbezüglich ausgeschüttet worden wären. Hieraus ist kein signifikanter Einfluss des neuen Parameters abzuleiten. Darüber hinaus ist in dieser Trainingsreihe keine weitere Stabilisierung der Bewegungsformen gegenüber den vorherigen Versuchen festzustellen.

Das Durchstrecken der Beine und das anschließende Beibehalten dieser Pose ist ein auffälliges Verhalten, das auch in den vorherigen Versuchen zu beobachten war. In Abbildung 6.20 sind die Drehmomentverläufe des hinteren rechten Beins gegenüber der Winkelposition der Gelenke aufgetragen. Daraus geht hervor, dass der Agent die Beinhaltung manifestiert, indem er die maximalen Drehmomente trotz der Positionierung am mechanischen Endanschlags aufrecht erhält.

Die hier beobachteten Bewegungen ähneln den Ergebnissen der ersten Belohnungsfunktion stärker als den Ergebnissen der zweiten. Der Einfluss des Höhenparameters ist nur im letzten Versuch feststellbar. Eine mögliche Ursache dafür und für den nicht nachweisbaren Einfluss des Parameters für den Fußkontakt, ist die Verschiebung der Gewichtung. Unter der gewählten Skalierung zwischen dem Primärziel der Fortbewegung und allen zusätzlichen Zielen, verringert sich der Anteil jedes sekundären Parameters an der maximalen Belohnung mit steigender Anzahl der zusätzlichen Parameter. Ihnen

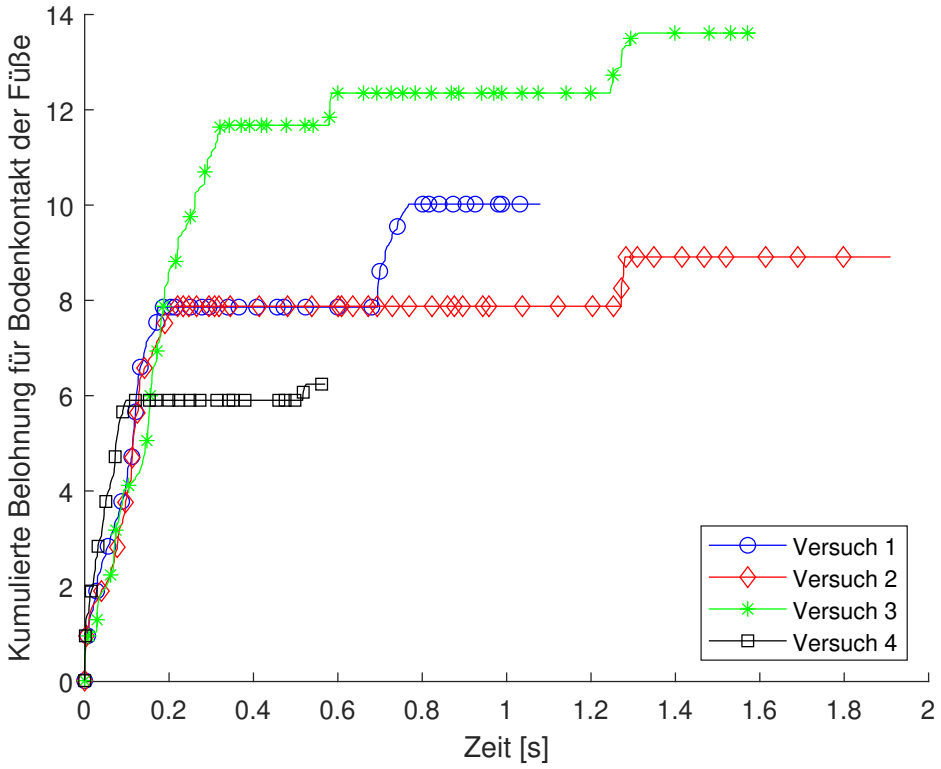

Abbildung 6.18.: Belohnungsfunktion 3, Belohnung für Füße im Bodenkontakt.

wird während des Lernprozesses weniger Bedeutung beigemessen, sodass die erlernten Bewegungsmuster größere Übereinstimmung mit denen der Belohnungsfunktion ohne Zusatzparameter zeigen.

Das Problem der lokalen Optima bleibt in dieser Trainingsreihe bestehen. Für die abschließende, vierte Belohnungsfunktion wird deshalb der Lösungsansatz von (Schuitema, 2012) erprobt. Die bisherigen Versuche haben in verschiedenen Varianten unter Ausnutzung des lokalen Optimums stets einen ähnlichen Gesamtbetrag der Belohnung generiert (vgl. Abbildungen 6.12 und 6.13). Der Betrag von 135 Punkten wird nun bei Erfüllung einer Sturzbedingung von der gesammelten Belohnung abgezogen.

6.4. Belohnungsfunktion 4

Die Bewegungssequenz des ersten Versuchs (Abbildung 6.21) beginnt mit dem Durchstrecken der Hinterbeine und dem Abstoßen vom Boden unter einem relativ flachen Winkel. Die Vorderbeine bleiben angewinkelt und werden, wie zuvor bereits beobachtet, nach außen geklappt. Ähnlich wie im letzten Durchlauf der zweiten Versuchsreihe,

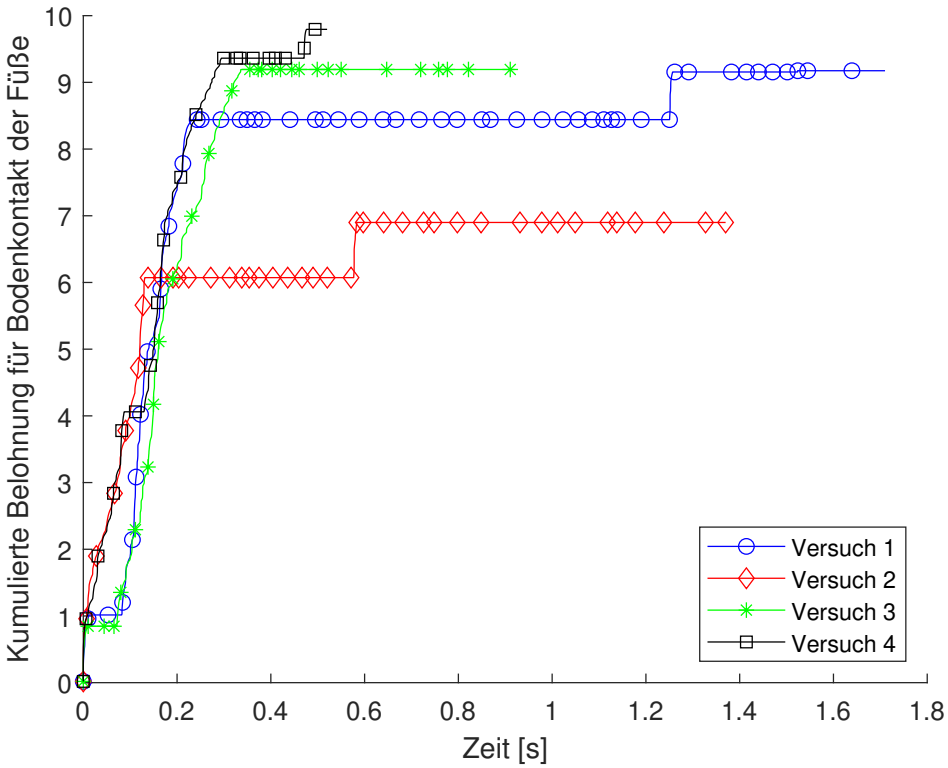

Abbildung 6.19.: Belohnungsfunktion 2, hypothetische Belohnung für Füße im Bodenkontakt.

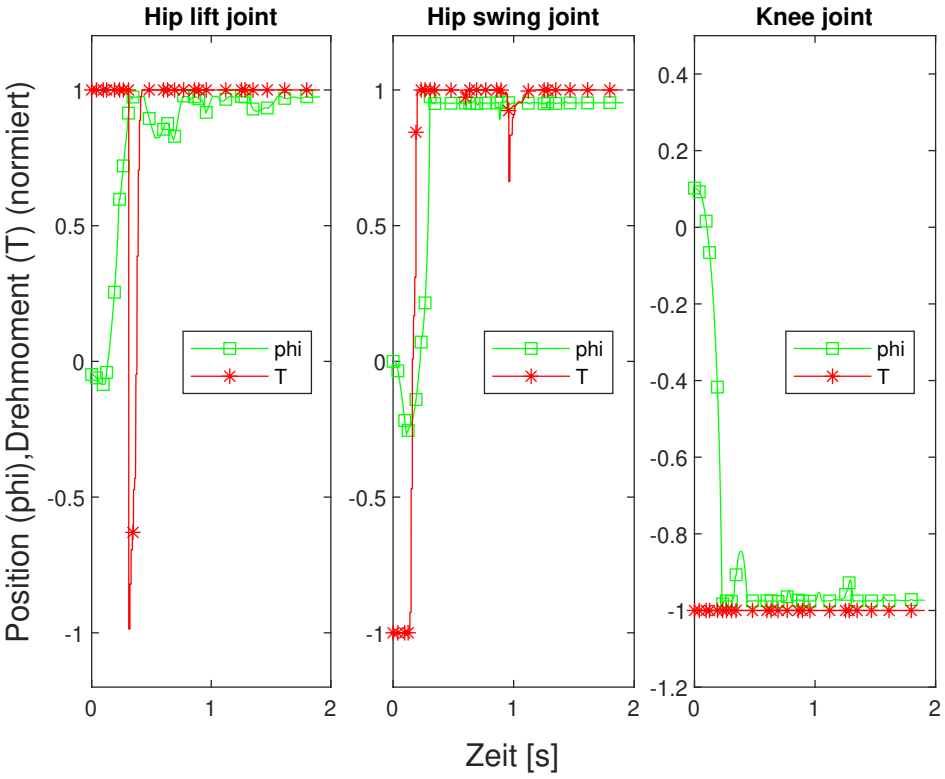

Abbildung 6.20.: Belohnungsfunktion 3, Verlauf von Drehmoment und Gelenkposition des hinteren, rechten Beins.

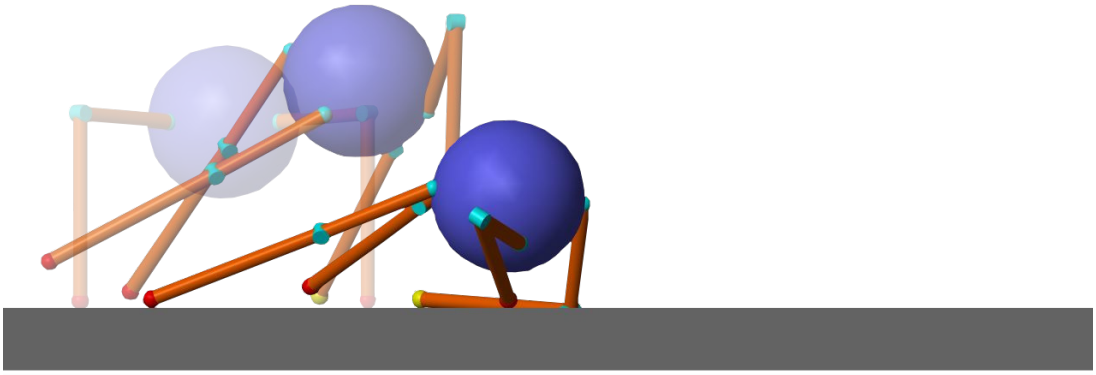

Abbildung 6.21.: Belohnungsfunktion 4, Bewegungssequenz des ersten Trainings.

endet die Sequenz mit der Kollision zwischen Kniegelenk und Boden.

Im zweiten Versuch (Abbildung 6.22) werden zunächst die Hinterbeine gestreckt, ohne dabei Geschwindigkeit aufzubauen. Die Vorderbeine werden nach außen geklappt und so weit wie möglich angezogen. Aus dieser geduckten Haltung heraus wird der Roboter durch kraftvolles Strecken der Vorderbeine nach vorne beschleunigt. Während der kurzen Flugphase ist ein Überschwingen des hinteren rechten Beins zu beobachten, wie es bereits im zweiten Versuch zur zweiten Belohnungsfunktion aufgetreten ist. Diese Bewegungsfolge schließt ebenfalls mit einem Sturz ab.

Für die pauschale Strafe bei Eintreten eines Sturzes ist kein positiver Effekt feststellbar. Die Trainingsstatistik in Abbildung 6.23 zeigt, dass die Lernkurve, im Vergleich zu den vorherigen Versuchsreihen, etwa um den Betrag der Bestrafung nach unten verschoben ist. In der nachträglichen Betrachtung erscheint diese Verschiebung schlüssig, weil jede Trainingsepisode, in der nicht das Ende des Trainingsparcours oder die maximale Anzahl der Schritte des Agenten erreicht wird, mit einem Sturz abschließt. Die verstärkende Wirkung dieser Strafe tritt demnach erst in Erscheinung, wenn eine geringfügige, aber stabile Fortbewegung erlernt wird. Als unmittelbare Gegenmaßnahme zu dem bisher beobachteten lokalen Optimum ist der Parameter daher nur wenig geeignet.

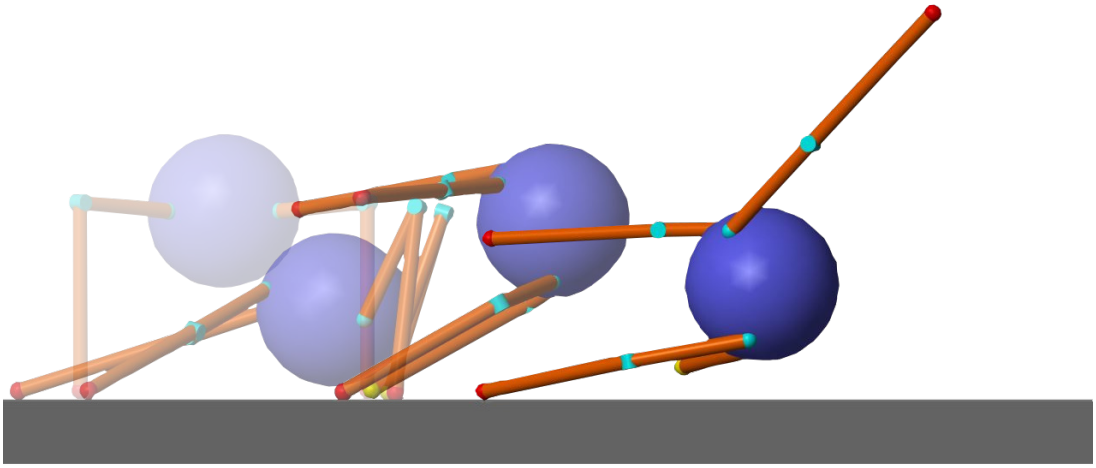

Abbildung 6.22.: Belohnungsfunktion 4, Bewegungssequenz des zweiten Trainings.

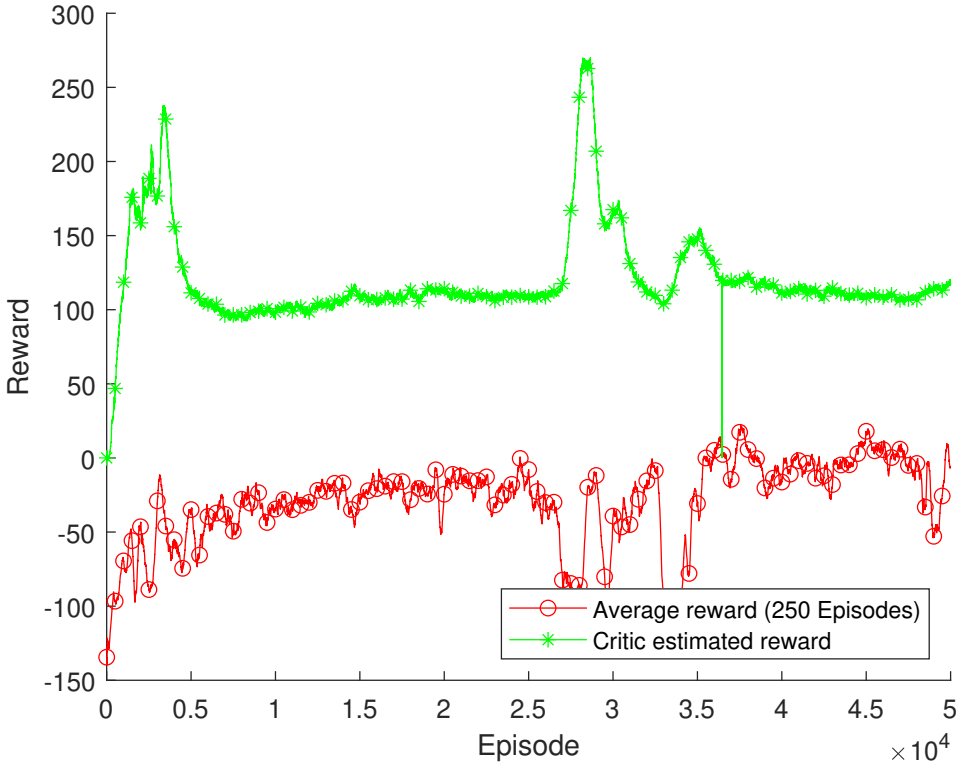

Abbildung 6.23.: Belohnungsfunktion 4, Trainingsstatistik des ersten Versuchs.

6.5. Parameter für weitere Belohnungsfunktionen

In dem zeitlich begrenzten Rahmen der vorliegenden Arbeit können keine weiteren Trainingsreihen durchgeführt werden. Zusätzliche Parameter, die für weitere Versuche in Betracht kommen, sind:

- die Orientierung des Körpers im Raum,
- die relativen Positionen des Körpers, der Gelenke und der Füße zueinander,
- die Orientierung des unteren Beinsegments gegenüber dem Boden,
- die Winkelbeschleunigung der Gelenke,
- das Zusammenspiel aus Gelenkposition und ausgeübtem Drehmoment.

Ähnlich dem Höhenparameter aus der zweiten Belohnungsfunktion, könnte zusätzliche Belohnung ausgeschüttet werden, wenn der Körper des Roboters eine annähernd aufrechte Position im Raum einnimmt. Einerseits würde dies Bewegungen begünstigen, die nicht zu dem häufig beobachteten Kippen führen. Andererseits schränkte der Parameter die Haltung beim Überqueren von Rampen oder Treppen ein, bei denen eine Neigung des Körpers natürlich erscheint.

Aus den relativen Positionen von Körper, Gelenken und Füßen können bevorzugte Konfigurationen entwickelt und mit zusätzlicher Belohnung verknüpft werden. Eine Positionierung der Füße unterhalb der Höhenkoordinate des Körperschwerpunkts oder des Kniegelenks erscheint sinnvoll, um ein Überschwingen der Beine über den Körper hinweg zu unterbinden. Im konkreten Fall des Treppensteigens kann dieser Parameter wiederum unerwünscht restriktiv für die Bewegungsfreiheit sein, wenn dies die Positionierung des Fußes auf einer höheren Ebene erforderte.

Die Fälle, in denen die Bewegungssequenz mit der Kollision zwischen Kniegelenk und Boden endet, könnten vermieden werden, indem eine lotrechte Pose des unteren Beinglieds gefördert würde. Gleichzeit würde dies die maximale Schrittweite des Roboters begrenzen. Dies könnte positiven Einfluss auf seine Stabilität haben, wobei die Mobilität einschränkt werden könnte.

Eine zusätzliche Belohnung für kleine Beträge der Winkelbeschleunigung in den Gelenken, könnte kraftvolle und ruckartige Bewegungen unterbinden und für gleichmäßige Trajektorien förderlich sein. Gegebenenfalls kommt dieser Parameter auch als Ersatz für die Bewertung der Drehmomente in Frage. Während in den erprobten Belohnungsfunktionen ein hoher Energieverbrauch pauschal mit einem Malus behaftet ist, würde dieser Ansatz nur solche Drehmomente bestrafen, die eine hohe Beschleunigung zur

Folge haben. Kraftintensive Aktionen die der Stabilisierung dienen, wären davon nicht betroffen.

Um den Missbrauch mechanischer Endanschläge zu unterbinden, könnte eine zusätzliche Bestrafung für Aktionen definiert werden, die in der Endposition höhere Drehmomente hervorbringen, als für die Wahrung der Position erforderlich ist.

Insbesondere die ersten drei der hier angeführten Parameter stellen weitere Einschränkungen der Bewegungsfreiheit dar und formen zunehmend die Eigenschaften einer konkreten Gangart. Dies ist nicht im Sinn des RL-Prozesses und würde eine zunehmende Abkehr von der in Abschnitt 5.5 entworfenen Strategie bedeuten. Unter diesem Aspekt ist die Verwendung von zusätzlichen Parametern nicht die bevorzugte Lösung für die Entwicklung einer stabilen Fortbewegung.

Im folgenden Abschnitt werden die Erkenntnisse der Trainingsprozesse zusammengefasst. Daran anschließend werden mögliche Gründe für die aufgetretenen Schwierigkeiten diskutiert und alternative Lösungswege skizziert.

6.6. Schlussbetrachtung

Eine erfolgreiche Strategie zur Fortbewegung des Roboters und für das Überwinden des Trainingsparcours konnte, unter den gegebenen Randbedingungen, mit keiner der Belohnungsfunktionen erlernt werden. In der Folge war es nicht möglich, in einer umfangreicheren Ganganalyse spezifische Merkmale einer stabilen Bewegungsart zu untersuchen und eine Verknüpfung zu den Parametern der Belohnungsfunktion herzustellen. Das höhere Ziel, Parameter für Belohnungsfunktionen zu finden, mit denen zielgerichtet bestimmte Eigenschaften der Fortbewegung in einem RL-Prozess erfüllt werden können, wurde damit nicht erreicht.

Die erste erprobte Belohnungsfunktion ist in ihrer Struktur und den Gewichtungen der Parameter vergleichbar mit den Funktionen aus anderen Beiträgen. Dennoch treten in Kombination mit dem in dieser Arbeit entwickelten Robotermodell ausschließlich lokale Optima auf. Dies wirft die Frage auf, inwiefern Belohnungsfunktionen erfolgreich zwischen verschiedenen Modellen übertragbar sind. Obwohl die Kinematik des Roboters vergleichbar ist mit dem Vierfüßler aus (Heess; T.B. et al., 2017) und ein Parameter der Belohnungsfunktion explizit der Vermeidung von lokalen Optima gilt, ist dessen Einfluss in dieser Versuchsreihe nicht feststellbar.

Der Versuch, den Agenten mit zusätzlichen Parametern schneller zu besseren Lösungen anzuleiten und dabei lokale Optima zu vermeiden, war nicht erfolgreich. Unter der

gegebenen Anzahl der Episoden ist keine Beschleunigung des Lernvorgangs festzustellen. Gleichwohl ist der Einfluss des Höhenparameters in der zweiten Belohnungsfunktion zu erkennen. Die Ergebnisse zeigen ein interessantes Beispiel dafür, wie der Agent die ihm gegebene Freiheit in der Lösungsfindung in unerwarteter Weise ausnutzt. Die Beschleunigung des Roboters durch Trägheitseffekte entspricht nicht den gewünschten Eigenschaften der Fortbewegung. Dieses Phänomen wurde in Abschnitt 5.3 bereits thematisiert und stellt ein grundlegendes Problem für das praxisorientierte Training von Laufrobotern dar.

Ebenfalls problematisch aus praktischer Sicht ist das im zweiten Versuch der dritten Belohnungsfunktion beobachtete Steuerungsverhalten des Agenten im Gelenkanschlag. Das Training war insgesamt nicht erfolgreich, sodass nicht sicher ist, ob dieser Effekt auch bei einer ausgebildeten Gangart auftreten würde. Die Stabilisierung der Beinhaltung durch Ausnutzung der mechanischen Begrenzung eines Gelenks ist sowohl aus energetischer Sicht, als auch unter dem Aspekt des Verschleiß unerwünscht. Ähnlich wie die Ausnutzung der Trägheitseffekte ist dies ein Ergebnis des RL-Prozesses, das im konventionellen Steuerungsentwurf keine Rolle spielt.

Weiterhin liefern die Versuchsreihen Anhaltspunkte dafür, welche Rolle die Gewichtung der Parameter in einer Belohnungsfunktion hat. Während der Effekt des Höhenparameters in der zweiten Versuchsreihe noch zu erkennen ist, sind die Zusatzparameter in den späteren Bewegungsfolgen praktisch nicht zu identifizieren. Das vereinzelte Auftreten flacher Trajektorien infolge der zweiten, dritten und vierten Belohnungsfunktion bestätigt nicht nur, dass die Verhältnisse zwischen den Parameter von Bedeutung sind, sondern zeigt auch, dass deren Effekte bei konstanter Gewichtung Schwankungen unterliegen. Dies lässt den Schluss zu, dass die Wirkung eines Parameters auch davon abhängt, ob der Agent dessen Mehrwert aufgrund seiner gesammelten Erfahrungen registriert.

6.7. Kritik

Eine eindeutige Ursache für den Misserfolg des Trainings kann nur schwer identifiziert werden, nicht zuletzt auch weil der Zufall ein wesentlicher Bestandteil des RL ist. In diesem Abschnitt wird deshalb eine Einschätzung vorgenommen, unter welchen Randbedingungen die Erfolgschancen der Untersuchung gesteigert werden könnten.

Die in Abschnitt 4.3 diskutierte Wahl der Trainingsparameter ist von zentraler Bedeutung für den RL-Prozess. Die Anzahl der möglichen Fehlerquellen stimmt dabei näherungsweise mit der Anzahl der Parameter überein. Eine erneute Betrachtung der getroffenen Auswahl erscheint daher angebracht.

Die Architektur und Größe des neuronalen Netzes sind durch die Arbeit von (Lillicrap et al., 2015) und die Vorversuche im Rahmen der vorliegenden Arbeit nachweislich geeignet, um die Lösung physikalischer Aufgaben in diesen Dimensionen zu erlernen. Bei der Wahl dazugehöriger Parameter, wie Lernrate, Mini-Batch-Größe, Smoothing-Faktor, dem Initialisierungsschema, sowie den Parametern des Optimierungsalgorithmus, liegt der Fokus auf der Robustheit und Stabilität des Lernprozesses. Die Trainingsstatistiken geben keinen Hinweis auf Instabilität oder Divergenz des Trainings, sodass dieses Ziel als erfüllt betrachtet werden kann. Es ist dennoch diskutabel, ob beispielsweise eine hierarchische Netzwerkarchitektur wie von (Heess; Wayne et al., 2016) bessere Voraussetzungen für die Komplexität dieser Aufgabe bietet. Ebenso ist davon auszugehen, dass mit einer extensiven Suche und Kalibrierung bessere Parameter gefunden werden können. Mit einer höheren oder gegebenenfalls variablen Lernrate könnte der Lernprozess beschleunigt werden. Ähnliches gilt für die übrigen Parameter. Insgesamt ist jedoch davon auszugehen, dass die getroffene Auswahl womöglich suboptimal, aber sehr wahrscheinlich nicht falsch ist.

Eine bedeutsame Hürde für die empirische Untersuchung von Belohnungsfunktionen ist die für jeden Trainingslauf benötigte Rechenzeit. In der Konzeption dieser Arbeit ist diesbezüglich eine Abschätzung getroffen worden, die sich im weiteren Verlauf als unzutreffend herausgestellt hat. Die tatsächlichen Rechenzeiten sind, zum deutlich überwiegenden Teil der Simulation des Roboters geschuldet, unter den gegebenen Bedingungen um ein Vielfaches größer als erwartet. Die Ursachen dafür liegen einerseits in der Schwierigkeit der gewählten Aufgabe und andererseits in der Grundlage anhand derer die Abschätzung getroffen wurde, wie im Folgenden erläutert wird.

Die gesamte Aufgabe weist vier separate Schwierigkeitsgrade auf:

- Laufen lernen mit Robotern und RL
- Treppensteigen mit Laufrobotern
- praktische Anforderungen an das Simulationsmodell
- empirische Untersuchung erfordert Vielzahl von Berechnungen

Die referenzierten Beiträge mit ähnlichen Aufgabenstellungen unterscheiden sich in einem oder mehreren dieser Aspekte von der vorliegenden Arbeit. Insbesondere das Modell des Roboters ist mit zwölf Freiheitsgraden komplexer als die Vierfüßler in (Duan et al., 2016) oder (Heess; T.B. et al., 2017). Gleichzeitig bilden die Erfahrungen aus diesen Beiträgen die Basis für die getroffene Abschätzung der Rechenzeit, sodass eine Diskrepanz zu der effektiven Rechenzeit plausibel ist. Darüber hinaus sind die Informationen über den Trainingsaufwand oftmals unvollständig, weil jeweils nur die Menge der

benötigten Episoden, der Schritte des Agenten oder der Zeitstunden dokumentiert wurden. Details über die Simulationsmodelle oder die verwendete Hardware sind in der Regel nicht gegeben. Abschließend ist anzumerken, dass zum Zeitpunkt der Aufgabengestaltung nicht sichergestellt war, welche Rechenkapazitäten für die Durchführung dieser Untersuchung zur Verfügung stehen würden. Die Summe der beschriebenen Unsicherheiten führte zur Fehleinschätzung der zu erwartenden Rechenzeiten.

Als Konsequenz mussten im Verlauf der Arbeit Kompromisse bezüglich der Anzahl der Berechnungen und des Umfangs jedes Trainingsversuchs eingegangen werden. In der Strategie für den Entwurf der untersuchten Belohnungsfunktionen wurde infolgedessen auf eine Variation der Gewichtung zwischen den Parametern verzichtet. Die Ergebnisse belegen jedoch die Bedeutung der Gewichtung, sodass eine Untersuchung dieser Dimension anzustreben ist. Die maximale Länge eines Trainings von 50 000 Episoden ist ein Kompromiss zugunsten der Anzahl der Berechnungen, die für eine empirische Untersuchung erforderlich sind. Eine Vergrößerung dieser Zahl kann die Chancen auf ein positives Trainingsergebnis signifikant erhöhen.

Um das Problem der Rechenzeiten zu beheben kann einerseits die Komplexität der Aufgabe reduziert und andererseits die eingesetzte Rechenkapazität vergrößert werden.

Mögliche Varianten um die Aufgabe zu vereinfachen sind:

- die Entwicklung einer einzelnen Belohnungsfunktion für das Treppensteigen,
- die Verwendung eines bezüglich der Simulationszeit optimierten Modells, unter Akzeptanz einer größeren Diskrepanz gegenüber der Realität,
- die Untersuchung einer einzelnen Gestaltungsdimension von Belohnungsfunktionen.

In der Nachbetrachtung ist der Misserfolg bezüglich des zentralen Forschungsziels zu einem bedeutenden Anteil dem Ungleichgewicht zwischen den Ambitionen der Aufgabe und den verfügbaren Ressourcen, in Form von Zeit und Rechenleistung, zuzurechnen. Dieses Ergebnis zeigt jedoch auch, dass der praktische Einsatz von RL in der Robotik gegenwärtig noch mehr als Herausforderung denn als fortschrittliche Lösungsmethode einzustufen ist.

7. Fazit

Modernen Methoden des RL wird das Potential zugeschrieben, Problemstellungen der Robotik handhaben zu können, für die mit konventionellen Techniken nur schwierig Lösungen zu finden sind. Aus dieser Perspektive heraus wurde in der vorliegenden Arbeit untersucht, wie die Gestaltung von Belohnungsfunktionen dazu beitragen kann, RL zielgerichtet für den Steuerungsentwurf von Laufrobotern einsetzbar zu machen. Als beispielhafte und aktuelle Herausforderung der Robotik behandelt die Forschungsarbeit die autonome Fortbewegung in Gegenwart von Treppen und Hindernissen.

Unter der Prämisse praktischer Anforderungen und in Anlehnung an den aktuellen Stand der Technik wurde das Simulationsmodell eines vierfüßigen Roboters entworfen und ein virtueller Trainingsparcours gestaltet. Des Weiteren wurden Trainingsbedingungen und Einstellungen für den RL-Prozess erarbeitet.

In einer grundlegenden theoretischen Auseinandersetzung wurde der manuelle Gestaltungsprozess von Belohnungsfunktionen betrachtet. Nach Kenntnis des Autors und auf Basis der Recherche in der Vorbereitung der vorliegenden Arbeit, ist dies der erste wissenschaftliche Beitrag, der den Entwurfsprozess explizit zum Gegenstand hat. Das Ergebnis der Analyse ist die Definition von fünf Dimensionen, innerhalb derer Entscheidungen für den Entwurf einer Belohnungsfunktion zu treffen sind, um mit RL ein gewünschtes Ergebnis zu erhalten.

Aus der theoretischen Betrachtung wurde eine Strategie zum Entwurf von Belohnungsfunktionen abgeleitet, um in einer Reihe von Trainingsversuchen den Einfluss ihrer Komponenten auf den Lernprozess und das erlernte Verhalten des Laufroboters zu ermitteln. In einem iterativen Prozess aus der Erprobung einer Belohnungsfunktion, der Analyse und Bewertung gelernter Bewegungsformen und der anschließenden Weiterentwicklung der Belohnungsfunktion, sollte der Roboter im Idealfall eine definierte Form der Fortbewegung über den Trainingsparcours erlernen.

Bei der Erprobung der Belohnungsfunktionen hat der Roboter in keinem Trainingsversuch eine stabile Form der Fortbewegung erlernt. Infolgedessen war es nicht möglich in einer Bewegungsanalyse geeignete Parameter herauszuarbeiten, um das Verhalten des Roboter zielgerichtet hinsichtlich der gewünschten Bewegungsform zu beeinflussen. Bezüglich der zentralen Forschungsfrage ist dies als Misserfolg einzuordnen. Die

wahrscheinlichste Ursache hierfür ist ein zu knapp bemessener Umfang der einzelnen Trainingsversuche. Aufgrund einer Fehleinschätzung der zu erwartenden Rechenzeiten und der zur Verfügung stehenden Rechenkapazitäten, musste ein Kompromiss der Trainingslänge gegenüber der Anzahl der Versuchsreihen geschlossen werden. Die Stabilität der Lernprozesse lässt darauf schließen, dass mit größerem Trainingsumfang bessere Resultate erzielbar gewesen wären.

In Bezug auf die praktische Anwendbarkeit von RL in der Robotik haben die Ergebnisse gezeigt, dass der offen gestaltete Lernprozess unerwünschte Nebeneffekte hervorbringen kann. In der vorliegenden Arbeit ist, ebenso wie in vergleichbaren Beiträgen, zu beobachten, dass der Agent dazu tendiert, Trägheitseffekte des Roboters zur Fortbewegung zu nutzen. Darüber hinaus ist festzustellen, dass die mechanischen Begrenzungen der Gelenke des Roboters ausgenutzt werden, um eine Beinpose unter Einsatz von unverhältnismäßigen Drehmomenten zu stabilisieren. Für reale technische Systeme ist dieses Verhalten problematisch und dokumentiert einen Nachteil von RL gegenüber dem klassischen Steuerungsentwurf.

Da das zentrale Ziel der vorliegenden Arbeit nicht erreicht wurde, bleibt die steuerbare Einflussnahme durch die Belohnungsfunktion weiterhin eine offene Forschungsfrage. Mögliche Wege um die Erfolgschancen zukünftiger Arbeiten zu diesem Thema zu steigern, sind die Reduzierung des Forschungsumfangs oder die Wiederaufnahme dieser Arbeit mit leistungsfähigeren Methoden und Ressourcen, die zum gegenwärtigen Zeitpunkt noch nicht zur Verfügung stehen.

Eine weitere Forschungsfrage die in der vorliegenden Arbeit aufgebracht, aber nicht abschließend behandelt werden konnte, ist die Frage nach den Grenzen der manuellen Gestaltung von Belohnungsfunktionen. Es ist zu erwarten, dass zukünftig, mit wachsender Rechenleistung und steigender Leistungsfähigkeit der Algorithmen, stetig komplexere Probleme mit RL bearbeitet werden. Vor diesem Hintergrund ist zu klären, in wie weit die menschliche Intuition mit dem manuellen Gestaltungsprozess dort von Bedeutung sein kann oder ob automatisierte Prozesse notwendig sein werden.

Die Erfahrungen dieser Arbeit dokumentieren keinen Beweis, sind aber sicherlich ein Indiz dafür, dass die Anwendung von RL auf praktische Probleme zum heutigen Zeitpunkt noch mehr eine Herausforderung als ein universelles Werkzeug ist. In Anbetracht der dynamischen Entwicklung dieser Disziplin in den vergangenen zehn Jahren, bleibt mit Spannung zu erwarten, welche Rolle RL in Zukunft einnehmen wird.

Literatur

[1] Abbeel, Pieter; Schulman, John: *Deep reinforcement learning through policy optimization. Tutorial at Neural Information Processing Systems*. 2016.

[2] Aggarwal, Charu C.: *Neural Networks and Deep Learning*. Cham,Schweiz: Springer Nature, 2018.

[3] *Anybotics*. 2020. URL: https://www.anybotics.com/.

[4] Badnava, Babak; Mozayani, Nasser: „A new Potential-Based Reward Shaping for Reinforcement Learning Agent“. In: *Computing Research Repository* (2019).

[5] Bellman, Richard: *Dynamic programming*. Princeton, New Jersey: Princeton University Press, 1957.

[6] Bengio, Yoshua: „Practical recommendations for gradient-based training of deep architectures“. In: *Computing Research Repository* (2012).

[7] Biancardi, Carlo et al.: „Biomechanics of octopedal locomotion: Kinematic and kinetic analysis of the spider Grammostola mollicoma“. In: *The Journal of experimental biology* 214 (2011), S. 3433–42.

[8] Bledt, Gerardo et al.: „MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot“. In: 2018.

[9] *Boston Dynamics*. 2020. URL: https://www.bostondynamics.com/spot.

[10] Clark, Jack; Amodei, Dario: *Faulty Reward Functions in the Wild*. 2016. URL: https://openai.com/blog/faulty-reward-functions/.

[11] Daniel, Christian et al.: „Active Reward Learning“. In: *Proceedings of Robotics: Science and Systems*. Berkeley, USA, 2014.

[12] Dholakiya, Dhaivat et al.: „Design, Development and Experimental Realization of a Quadrupedal Research Platform: Stoch“. In: *Computing Research Repository* (2019).

[13] Duan, Yan et al.: „Benchmarking Deep Reinforcement Learning for Continuous Control“. In: *Computing Research Repository* (2016).

[14] Fasih, Alireza; Chedjou, J.; Kyamakya, Kyandoghere: „Cellular Neural Networks-Based Genetic Algorithm for Optimizing the Behavior of an Unstructured Robot“. In: *International Journal of Computational Intelligence Systems* 2 (2009).

[15] François-Lavet, V. et al.: „An Introduction to Deep Reinforcement Learning“. In: *Foundations and Trends in Machine Learning* 11.3-4 (2018).

[16] Gasparetto, A.; Vidoni, R.; Seidl, T.: „Kinematic study of the spider system in a biomimetic perspective“. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, S. 3077–3082.

[17] Glorot, Xavier; Bengio, Yoshua: „Understanding the difficulty of training deep feedforward neural networks“. In: *AISTATS*. 2010.

[18] Gu, Shixiang et al.: „Deep Reinforcement Learning for Robotic Manipulation“. In: *Computing Research Repository* (2016).

[19] *Guardian*. 2020. URL: https://www.theguardian.com/technology/2015/dec/30/us-marines-reject-bigdog-robot-boston-dynamics-ls3-too-noisy.

[20] Haarnoja, Tuomas; Zhou, Aurick; Abbeel, Pieter et al.: „Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor“. In: *Computing Research Repository* (2018).

[21] Haarnoja, Tuomas; Zhou, Aurick; Ha, Sehoon et al.: „Learning to Walk via Deep Reinforcement Learning“. In: *Computing Research Repository* (2018).

[22] He, Kaiming et al.: „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“. In: *Computing Research Repository* (2015).

[23] Heess, Nicolas; T.B., Dhruva et al.: „Emergence of Locomotion Behaviours in Rich Environments“. In: (2017).

[24] Heess, Nicolas; Wayne, Gregory et al.: „Learning and Transfer of Modulated Locomotor Controllers“. In: *Computing Research Repository* (2016).

[25] Henley, Julian J.; Barnes, David P.: „An Artificial Neuro-Endocrine Kinematics Model for Legged Robot Obstacle Negotiation“. In: *8th ESA Workshop on Advanced Space Technologies for Robotics and Automation*. 2004, S. 134–142.

[26] Hoffer, Elad; Hubara, Itay; Soudry, Daniel: „Train longer, generalize better: closing the generalization gap in large batch training of neural networks“. In: *Advances in Neural Information Processing Systems 30*. Hrsg. von Guyon, I. et al. Curran Associates, Inc., 2017, S. 1731–1741.

[27] Huang, Jie et al.: „Walking Efficiency and Crossing Obstacle Ability of Variable Eccentric Wheel-legged Robot“. In: 2017.

[28] Hwangbo, Jemin et al.: „Learning agile and dynamic motor skills for legged robots“. In: *Science Robotics* 4.26 (2019).

[29] Ilyas, Muhammad et al.: „Staircase Recognition and Localization Using Convolution Neural Network (CNN) for Cleaning Robot Application“. 2018.

[30] Johnson, A. M. et al.: „Autonomous legged hill and stairwell ascent“. In: *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. 2011, S. 134–142.

[31] Keskar, Nitish Shirish et al.: „On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima“. In: *Computing Research Repository* abs/1609.04836 (2016).

[32] Kingma, Diederik; Ba, Jimmy: „Adam: A Method for Stochastic Optimization“. In: *International Conference on Learning Representations* (2014).

[33] Kitano, Satoshi et al.: „TITAN-XIII:sprawling-type quadruped robot with ability of fast and energy-efficient walking“. In: *ROBOMECH Journal* (3 2016).

[34] Kober, Jens; Bagnell, J.; Peters, Jan: „Reinforcement Learning in Robotics: A Survey“. In: *The International Journal of Robotics Research* 32 (2013), S. 1238–1274.

[35] Kohl, N.; Stone, P.: „Policy gradient reinforcement learning for fast quadrupedal locomotion“. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Bd. 3. 2004, 2619–2624 Vol.3.

[36] Lillicrap, Timothy et al.: „Continuous control with deep reinforcement learning“. In: *Computing Research Repository* (2015).

[37] Marom, Ofir; Rosman, Benjamin: *Belief Reward Shaping in Reinforcement Learning*. 2018.

[38] Masters, Dominic; Luschi, Carlo: „Revisiting Small Batch Training for Deep Neural Networks“. In: *Computing Research Repository* (2018).

[39] *MATLAB Documentation*. Version 2019b. 2020.

[40] Minsky, Marvin: „Neural Nets and the Brain Model Problem“. Dissertation. Princeton, 1954.

[41] Mishkin, Dmytro; Matas, Jiri: *All you need is a good init*. International Conference on Learning Representations, At San Juan, Puerto Rico, USA, 2015.

[42] Mnih, Volodymyr et al.: „Human-level control through deep reinforcement learning“. In: *Nature* 518 (2015), S. 529–33.

[43] Moore, E. Z. et al.: „Reliable stair climbing in the simple hexapod 'RHex'“. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation*. Bd. 3. 2002, 2222–2227 vol.3.

[44] Nauta, Johannes; Khaluf, Yara; Simoens, Pieter: „Using the Ornstein-Uhlenbeck Process for Random Exploration“. In: *COMPLEXIS*. 2019.

[45] Ng, Andrew Y.; Harada, Daishi; Russell, Stuart J.: „Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping“. In: *ICML*. 1999.

[46] Pa, P.S.; Wu, C.M.: „Design of a hexapod robot with a servo control and a man-machine interface“. In: *Robotics and Computer-Integrated Manufacturing* 28.3 (2012), S. 351–358.

[47] Pawan, Jain: *Complete Guide of Activation Functions*. 2019. URL: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044.

[48] Raibert, Marc et al.: „BigDog, the Rough-Terrain Quadruped Robot“. In: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, S. 10822–10825.

[49] Randlov, Jette; Alstrøm, Preben: „Learning to Drive a Bicycle Using Reinforcement Learning and Shaping.“ In: 1998, S. 463–471.

[50] Rosenblatt, Frank: „The perceptron - a probabilistic model for information storage and organization in the brain“. In: *Psychological Review* (65 1958).

[51] Ruder, Sebastian: „An overview of gradient descent optimization algorithms“. In: *Computing Research Repository* (2016).

[52] Sarmah, Anurag Narayan et al.: „A Bio-Inspired Implementation of Walking and Stair Climbing on a Quadruped Robot“. In: *Procedia Computer Science* 143 (2018). 8th International Conference on Advances in Computing and Communications (ICACC-2018), S. 671–677.

[53] Schuitema, Erik: „Reinforcement Learning on autonomous humanoid robots“. Dissertation. Technische Universität Delft, 2012.

[54] Schulman, John et al.: „Trust Region Policy Optimization“. In: *Proceedings of the 32nd International Conference on Machine Learning*. Bd. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, S. 1889–1897.

[55] Silver, David: *Reinforcement Learning*. Lecture. University College London, 2015.

[56] Silver, David et al.: „Deterministic Policy Gradient Algorithms“. In: *31st International Conference on Machine Learning, ICML 2014* 1 (2014).

[57] Smith, Samuel L.; Kindermans, Pieter-Jan; Le, Quoc V.: „Don't Decay the Learning Rate, Increase the Batch Size“. In: *Computing Research Repository* (2017).

[58] *Sony*. 2020. URL: https://us.aibo.com/.

[59] Spröwitz, Alexander et al.: „Towards Dynamic Trot Gait Locomotion: Design, Control, and Experiments with Cheetah-cub, a Compliant Quadruped Robot“. In: *The International Journal of Robotics Research* 32 (2013).

[60] Spröwitz, Alexander et al.: „Oncilla robot: a versatile open-source quadruped research robot with compliant pantograph legs“. In: *Computing Research Repository* (2018).

[61] Sutton, Richard S.; Barto, Andrew G.: *Reinforcement Learning - An introduction*. Cambridge, Massachusetts, London, England: The MIT Press, 2018.

[62] Uhlenbeck, G. E.; Ornstein, L. S.: „On the Theory of the Brownian Motion“. In: *Phys. Rev.* 36 (5 1930), S. 823–841.

[63] Wiering, Marco; Otterlo, Martijn van: *Reinforcement Learning. State-of-the-Art*. 1. Aufl. Berlin,Heidelberg: Springer, 2012.

[64] Wiewiora, Eric; Cottrell, Garrison; Elkan, Charles: „Principled Methods for Advising Reinforcement Learning Agents“. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, S. 792–799.

[65] Williams, R.J.: „Simple statistical gradient-following algorithms for connectionist reinforcement learning“. In: *Machine Learning* (8 1992), S. 229–256.

[66] Yang, Yuxiang et al.: „Data Efficient Reinforcement Learning for Legged Robots“. In: *Computing Research Repository* (2019).

[67] Zamani, Ali; Khorram, Mahdi; Moosavian, S. Ali A.: „Stable Stair-Climbing of a Quadruped Robot“. In: *Computing Research Repository* (2018).

[68] Zhanghe, He et al.: „Development of a Bionic Hexapod Robot for Walking on Unstructured Terrain“. In: *Journal of Bionic Engineering* 11 (2014), S. 176–187.

[69] Zou, Haosheng et al.: „Reward Shaping via Meta-Learning“. In: *Computing Research Repository* (2019).

Anhang
Inhaltsverzeichnis

A. Trainingsstatistiken

Dieser Anhang enthält die Lernkurven aller Trainingsversuche.

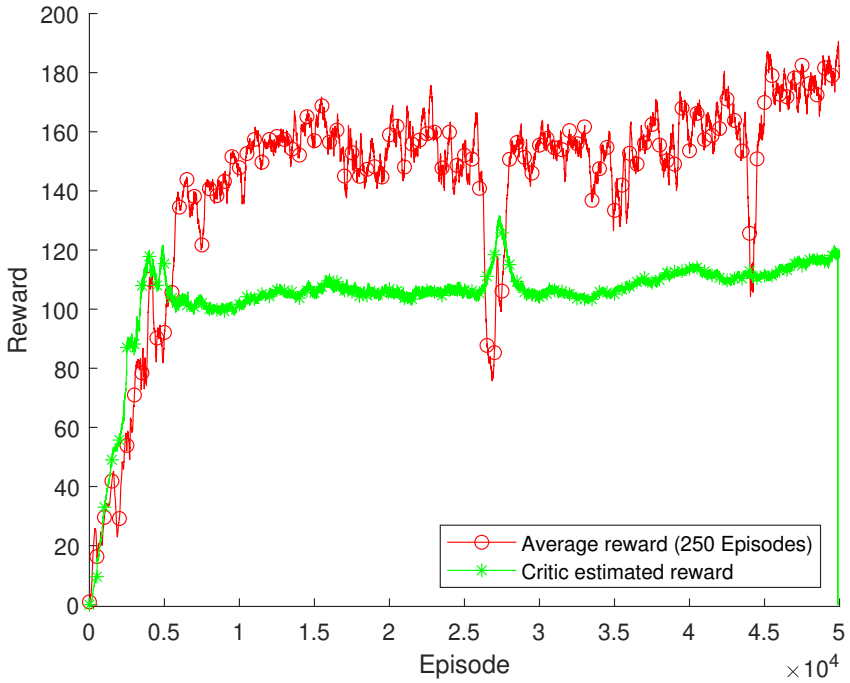

Abbildung A.1.: Belohnungsfunktion 1, Trainingsstatistik des ersten Versuchs.

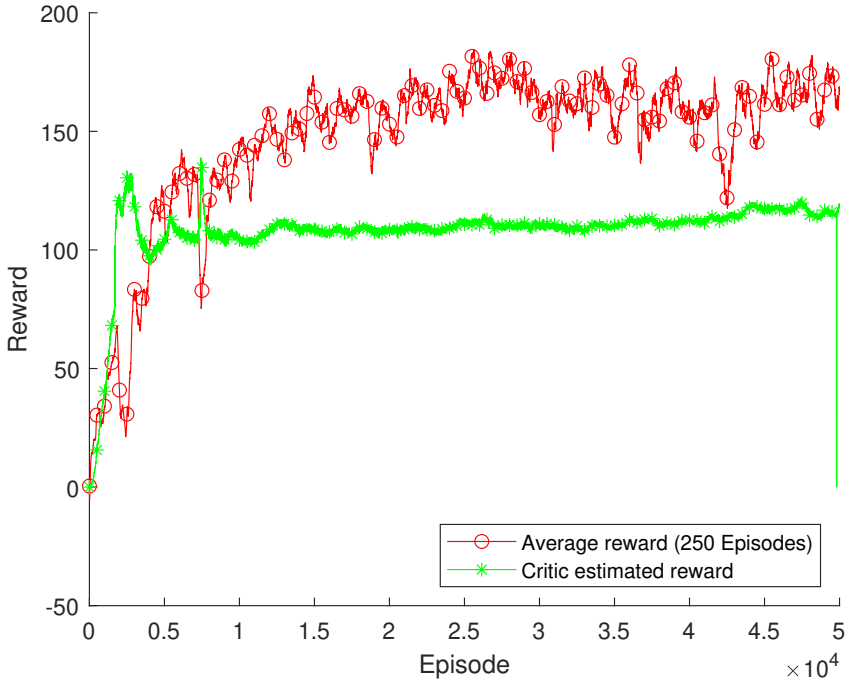

Abbildung A.2.: Belohnungsfunktion 1, Trainingsstatistik des zweiten Versuchs.

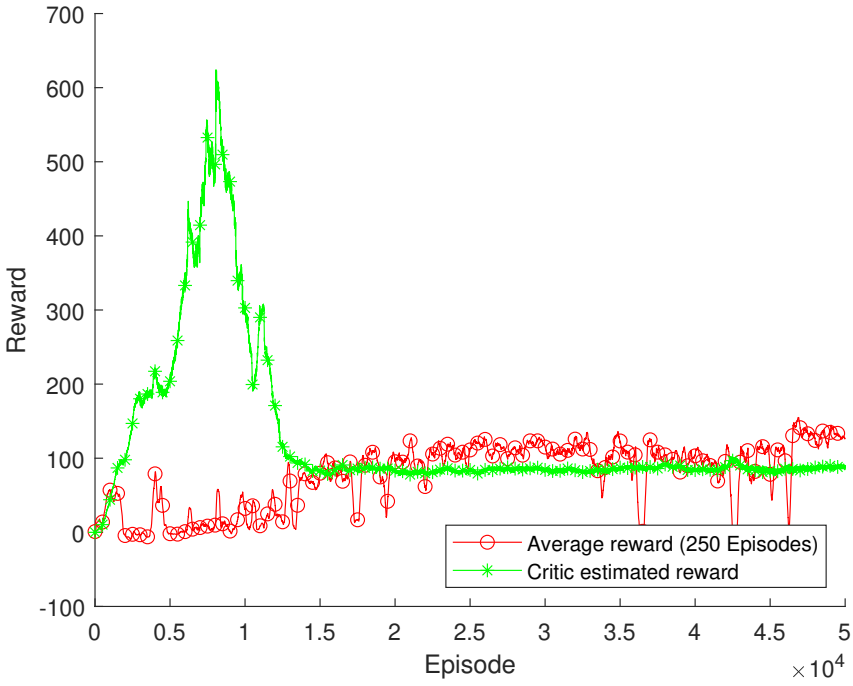

Abbildung A.3.: Belohnungsfunktion 1, Trainingsstatistik des dritten Versuchs.

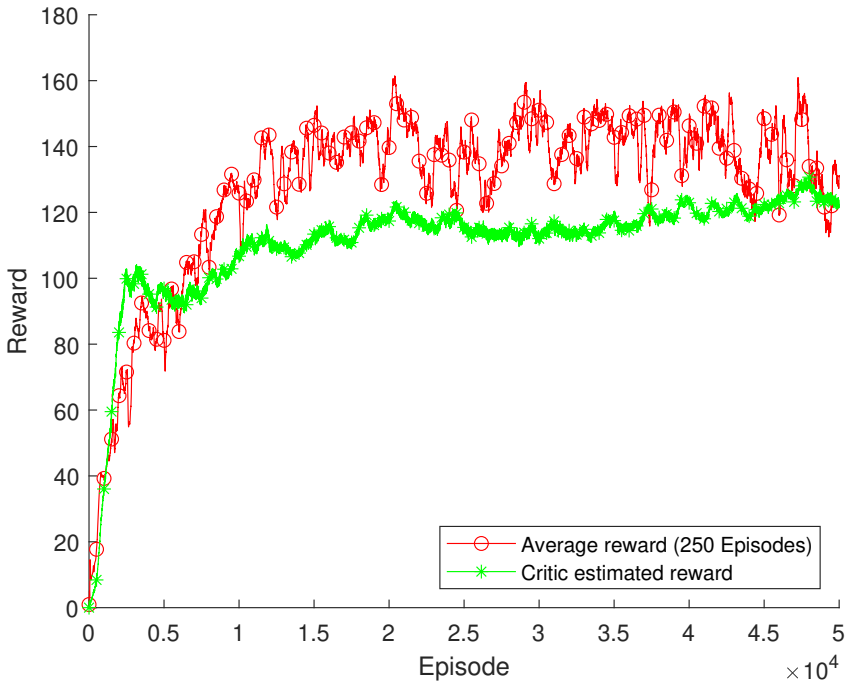

Abbildung A.4.: Belohnungsfunktion 1, Trainingsstatistik des vierten Versuchs.

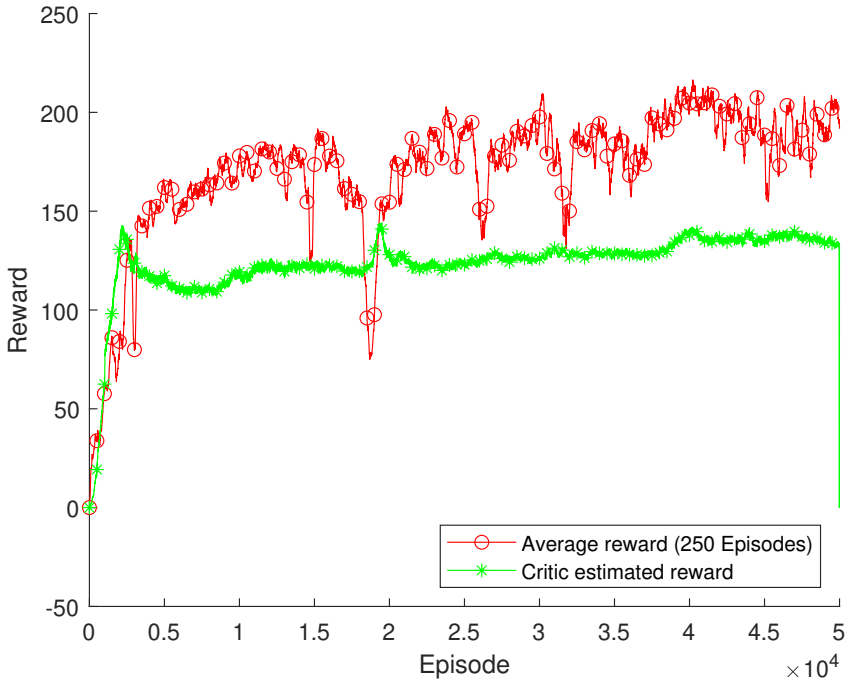

Abbildung A.5.: Belohnungsfunktion 2, Trainingsstatistik des ersten Versuchs.

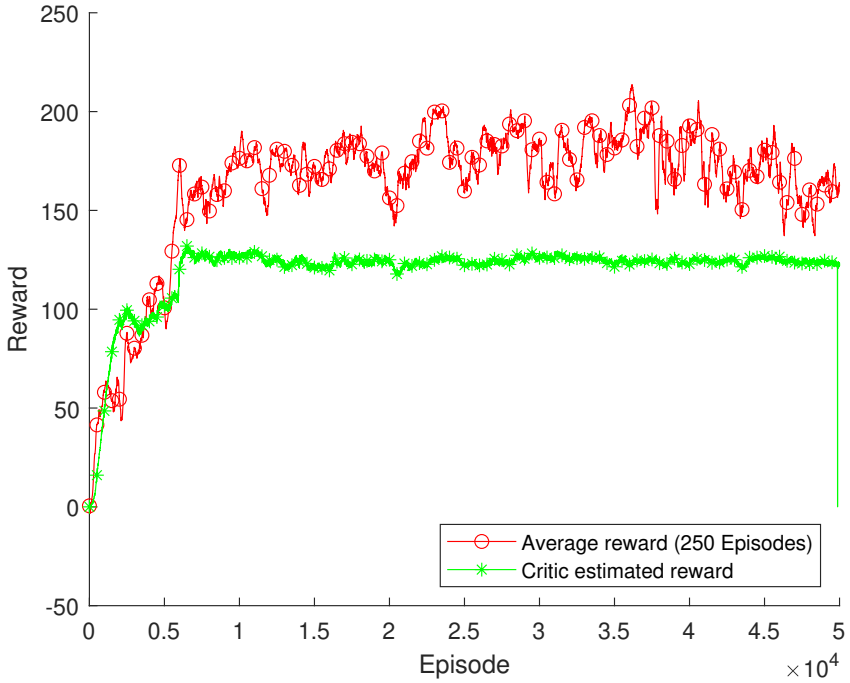

Abbildung A.6.: Belohnungsfunktion 2, Trainingsstatistik des zweiten Versuchs.

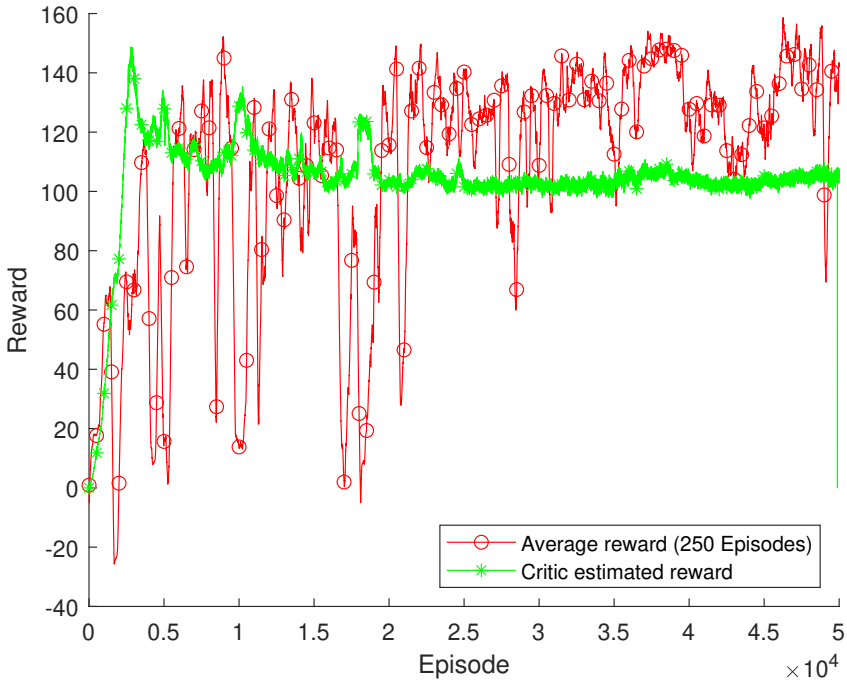

Abbildung A.7.: Belohnungsfunktion 2, Trainingsstatistik des dritten Versuchs.

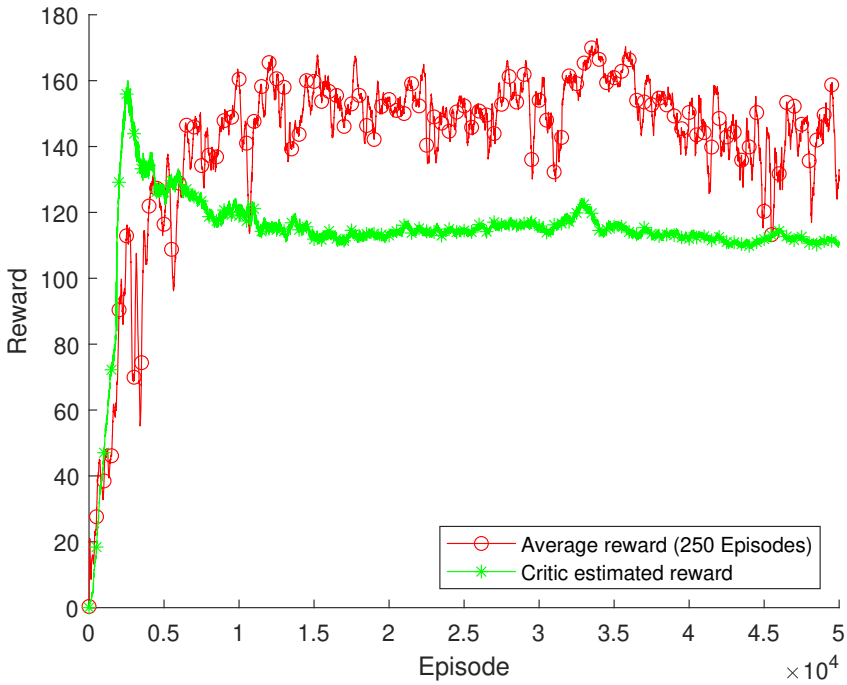

Abbildung A.8.: Belohnungsfunktion 2, Trainingsstatistik des vierten Versuchs.

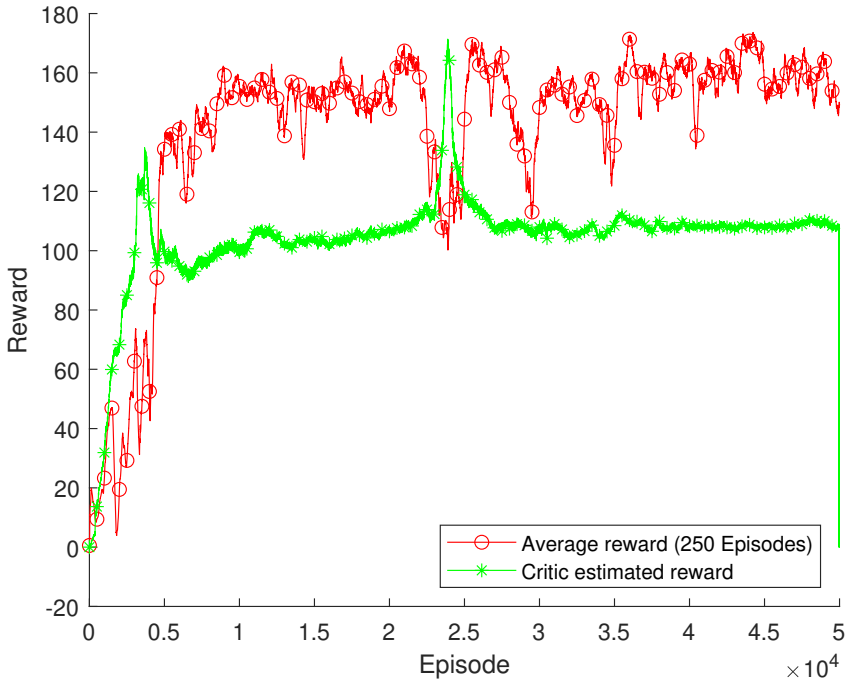

Abbildung A.9.: Belohnungsfunktion 3, Trainingsstatistik des ersten Versuchs.

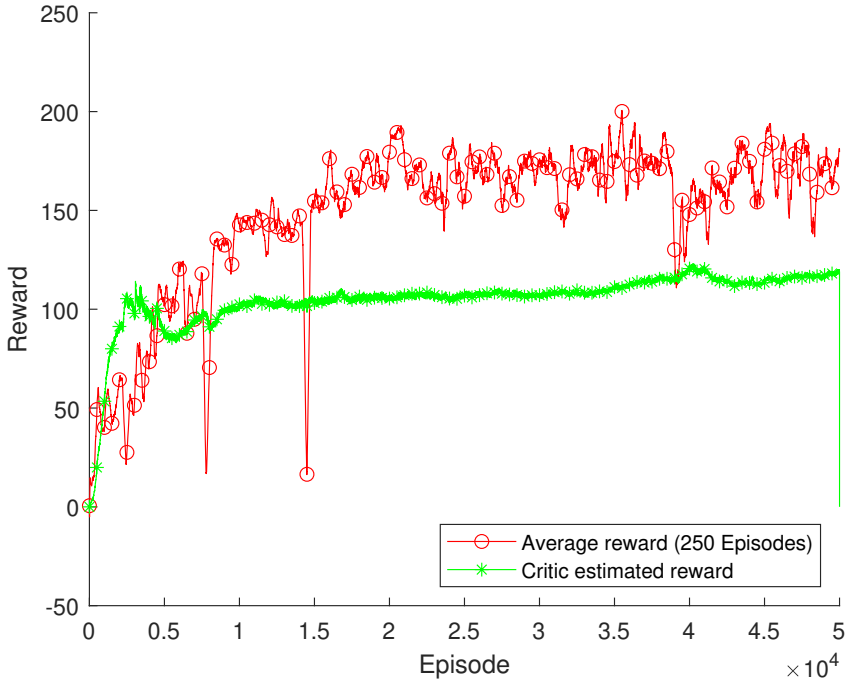

Abbildung A.10.: Belohnungsfunktion 3, Trainingsstatistik des zweiten Versuchs.

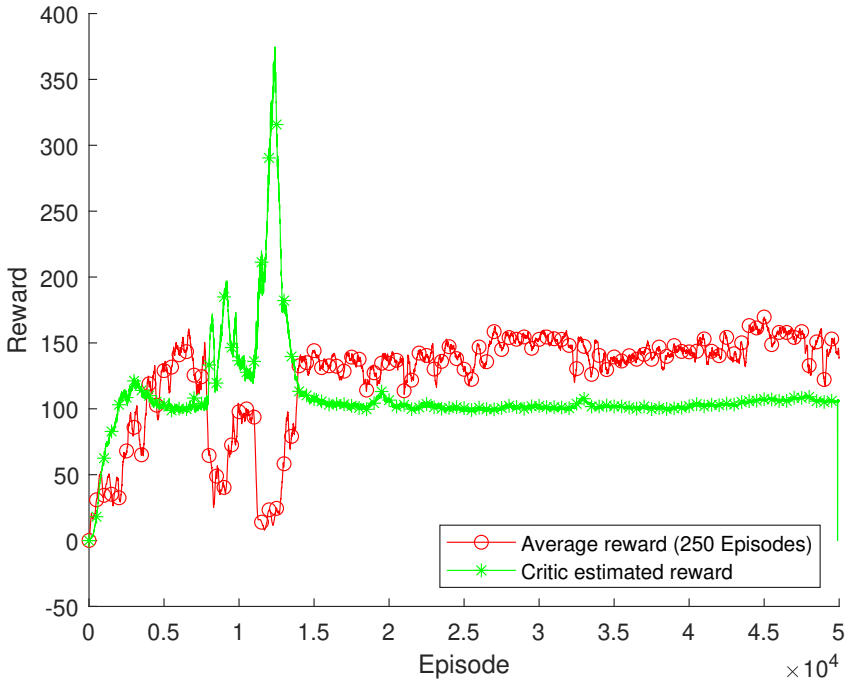

Abbildung A.11.: Belohnungsfunktion 3, Trainingsstatistik des dritten Versuchs.

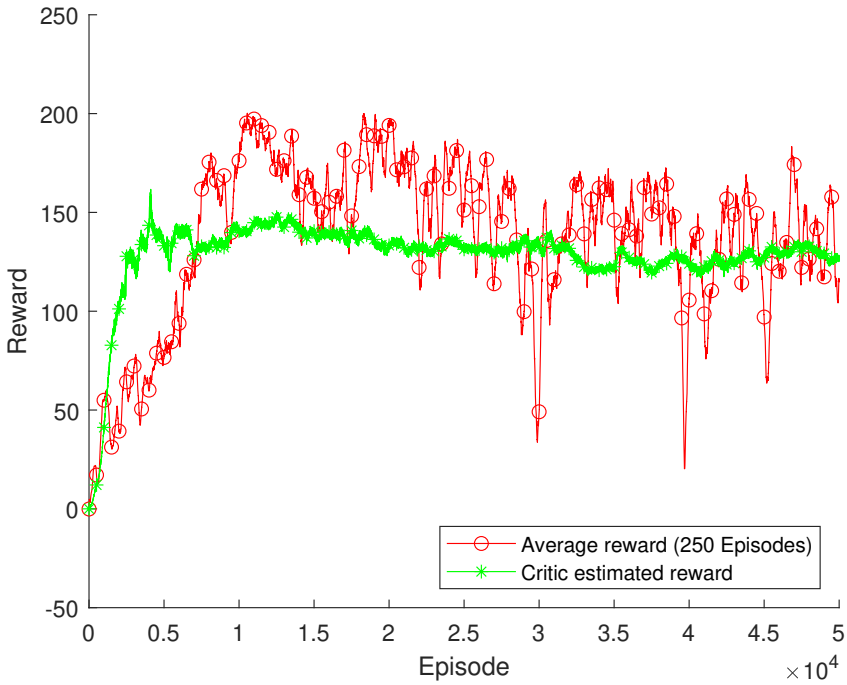

Abbildung A.12.: Belohnungsfunktion 3, Trainingsstatistik des vierten Versuchs.

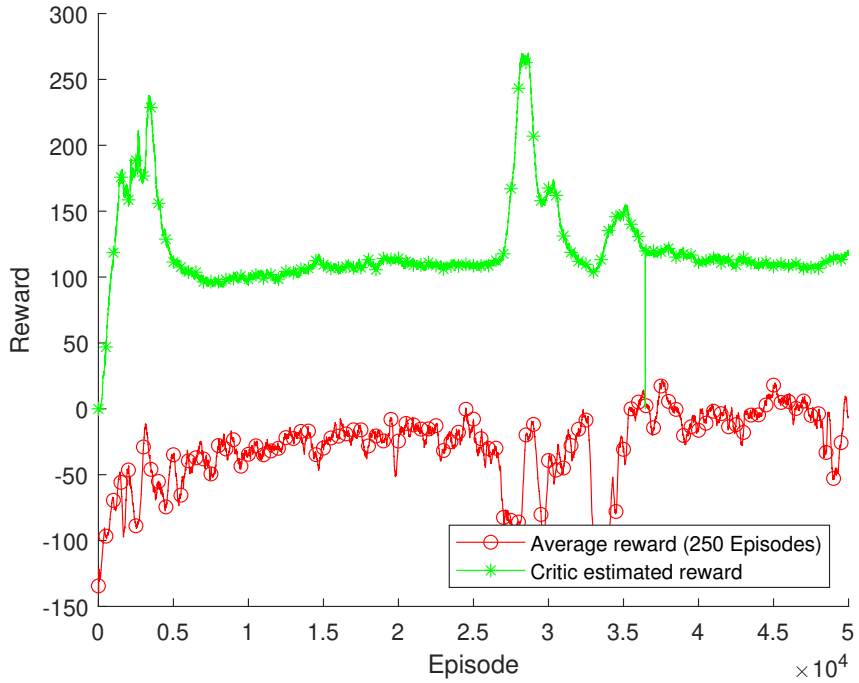

Abbildung A.13.: Belohnungsfunktion 4, Trainingsstatistik des ersten Versuchs.

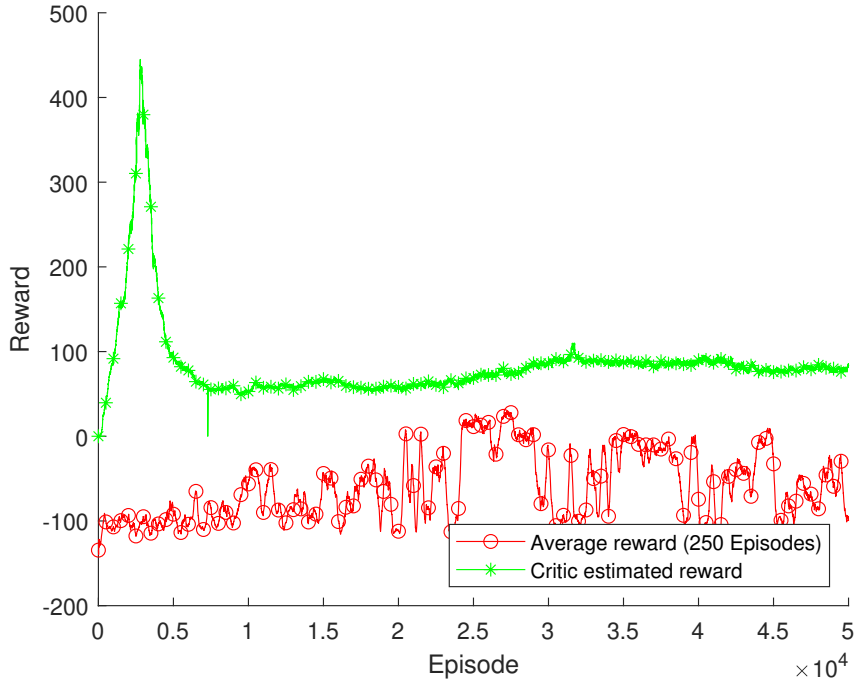

Abbildung A.14.: Belohnungsfunktion 4, Trainingsstatistik des zweiten Versuchs.