Universität Duisburg-Essen
Fakultät für Informatik
Abteilung für Software Engineering
Fachgebiet Intelligente Systeme

- Bachelorarbeit -
im Studiengang
Angewandte Informatik - Bachelor

Modellierung und Simulation eines quadrupedalen Roboters zum Erlernen von Bewegungsabläufen mit Methoden des Deep Reinforcement Learnings

Baran Demir

Erstprüfer: Prof. Dr. Josef Pauli
Zweitprüfer: Prof. Dr. Jens Krüger
Zeitraum: 11. August 2023 bis 17. November 2023

Zusammenfassung

Durchbrüche in den Bereichen Deep Learning und Robotik machen autonome intelligente Systeme immer allgegenwärtiger. Diese Systeme werden eines Tages potenziell in der Lage sein, den Menschen bezüglich kognitiver als auch motorischer Fähigkeiten zu übertreffen. Oft in der Gestalt eines Roboters könnten diese kognitiven Systeme Aufgaben erledigen, die für den Menschen nur schwer zu bewältigen oder inhärent mit Gefahren verbunden sind. Es werde an eine komplexe Rettungsaktion gedacht, möglicherweise in einem abgelegenen Waldgebiet. Dieses ist für Kraftfahrzeuge als auch für Menschen schwer zu durchforsten. Diese Bachelorarbeit widmet sich dem Erlernen von Bewegungsabläufen eines quadrupedalen Roboters, mit der Motivation, dass diese Roboterplattform durch ihre Flexibilität das Potenzial hat, eine Vielzahl an Aufgaben bewältigen zu können, insbesondere jene, die viel Geschick und Schnelligkeit erfordern. Für das Erlernen der Bewegungsabläufe wird die Methode des Deep Reinforcement Learnings verwendet, mit der Absicht, dass diese ein Maß an Optimalität liefert, das möglicherweise unter konventionellen Methoden nicht möglich ist. Durch Modellierung eines geeigneten Zustandsraumes und einer geeigneten Belohnungsfunktion soll ein Agent optimales Verhalten erlernen. Im Verlaufe dieser Bachelorarbeit wird auf verschiedene Herausforderungen des Deep Reinforcement Learnings im Allgemeinen und spezifisch im Kontext von (quadrupedalen) Robotern eingegangen. Ziel ist es, durch diese Erkenntnisse einen Recovery-Controller zu entwickeln. Das ist ein Controller, der im Vergleich zu einem konventionellen Controller eine Falldetektion und eine automatische Aufstehfunktion implementiert. Dieser Controller beinhaltet Teilbewegungen, welche nicht konventionell explizit programmiert, sondern durch Deep Reinforcement Learning erlernt werden.

Inhaltsverzeichnis

1 Einleitung

Die Robotik ist ein breites Feld. Entsprechend gibt es viele Herausforderungen, die durch eine Vielzahl von Methoden gelöst werden können. Dieses Kapitel gibt einen Überblick über das in dieser Bachelorarbeit betrachtete Problem, welche Lösungsansätze es für dieses gibt und inwiefern diese Fragestellungen untersucht werden sollen.

1.1 Motivation

Die Forschung der quadrupedalen Robotik hat im Verlaufe der letzten Jahre ein großes Maß an Aufmerksamkeit erringen können. Roboterplattformen wie MIT Cheetah (Massachusetts Institute of Technology) oder ANYmal (ETH Zürich) demonstrieren die Tauglichkeit dieses Systems in der echten Welt. ANYbotics, 2016 von der ETH Zürich gegründet, zeigt die kommerzielle Anwendung von ANYmal in der Inspektion und Überwachung von Industrieanlagen wie beispielsweise einer weit entfernten Bohrinsel, die für Menschen schwer erreichbar ist. Ein Problem bei diesem Anwendungsfall ist, dass physikalische Gegebenheiten unerwartete Probleme verursachen können, wie zum Beispiel das Ausrutschen des Roboters bei Nässe. Weitere Arbeiten beschäftigen sich mit der Exploration von Tunneln [Miller u. a. 2020]. Hier ist nicht nur die physikalische Erreichbarkeit ein Problem; Kommunikations- und Sensorstörungen müssen ebenfalls in Betracht gezogen werden. Doch auch wenn es nicht zu unerwarteten Störungen kommt, ist die Umgebung nicht exakt beschreibbar. Möglicherweise soll der Roboter ein waldiges, felsiges und/oder schlammiges Gebiet durchqueren, das stets Änderungen unterliegt und den Roboter zwingt sein Verhalten zu überprüfen und wenn notwendig zu verändern. Deshalb ist es besonders wichtig, eine Roboterplattform zu kreieren, die robust gegen allerlei Störquellen und adaptiv bezüglich verschiedener Umgebungen ist.
Diese Bachelorarbeit fokussiert sich auf die Adaptabilität des Roboters, indem Methoden untersucht werden, um robustes Verhalten auch bei Abweichungen von Standardparametern zu gewährleisten. Ferner liegt der Fokus auf dem Problem des Hinfallens. Dieses ist besonders kritisch im Kontext der oben beschriebenen Anwendungen. Ist der Roboter nicht in der Lage, sich eigenständig nach einem Fall aufzurichten, so muss ein menschlicher Operateur intervenieren. Das würde einen der größten Nutzen der quadrupedalen Robotik zunichtemachen, nämlich das autonome Erkunden von für Menschen unzugänglichen Räumen. Zu diesem Zweck ist es das Ziel der Bachelorarbeit, einen Controller mittels des Simulationsprogrammes Isaac Sim für den Unitree A1 zu implementieren. Der Unitree A1 ist ein quadrupedaler Roboter, der zur Ausstattung des Lehrstuhls für Intelligente Systeme der Universität Duisburg-Essen gehört. Ein Controller ist eine Software, die den Roboter steuert. Die Aufgabe des hier entwickelten Controllers soll es sein, automatisch einen Fall zu detektieren

und eine Prozedur einzuleiten, die den Roboter in einen operationsfähigen Zustand versetzt.

1.2 Problemstellung

Ein solcher Controller ist unter konventionellen Methoden besonders schwierig zu realisieren. Zunächst muss ein Konzept entwickelt werden, das die Grundfunktionalität des Roboters implementiert. Das ist in erster Linie das Laufen und das Aufstehen. Erfordert wird nicht nur Expertenwissen über die Regelung von Robotern im Allgemeinen, sondern auch im Spezifischen für quadrupedale Roboter. Weiter ist Wissen über das exakte Modell des verwendeten quadrupedalen Roboters notwendig. In einem Zustand s ist eine Aktion a zu wählen. Die Aufgabe des Entwicklers ist es, für alle Zustände, die auftreten können, die beste Aktion zu bestimmen. Daraus ergibt sich bereits die kritische Frage, ob das manuell festgelegte Verhalten optimal ist und woran Optimalität gemessen werden soll. Außerdem muss die korrekte Aktion eventuell für sehr viele Zustände betrachtet werden. Welche Aktion a ist zu wählen, wenn der Roboter im Zustand s_x stürzt? Welche Aktion a ist zu wählen, wenn der Roboter im Zustand s_y stürzt? Wie sollen Zustände s_x und s_y definiert und unterschieden werden? Diese Fragen sind beantwortbar, wenn der Zustands- und Aktionsraum überschaubar groß ist. Der Entwickler könnte die Zustände definieren, entscheiden, welche Aktion in einem Zustand die Beste nach seinem manuell festgelegten Kriterium ist und dieses Verhalten über bedingte Anweisungen (If-Else-Anweisungen) implementieren. Dieses Verfahren ist aber nicht realistisch, wenn der Zustands- und Aktionsraum so groß ist, dass ein Mensch ihn nicht mehr überblicken kann.
Deshalb wird die Methode des Deep Reinforcement Learnings in Betracht gezogen. Der Roboter soll die gewünschte Bewegung in der Simulation durch Befolgen eines Verhaltens, von hier an Policy π genannt, durch ein Signal selber erlernen. Dieses Signal wird im Reinforcement Learning Belohnung r genannt und bildet einen Zustand s auf eine reelle Zahl ab. Das Ziel ist, dass der Roboter in vielen verschiedenen Zuständen ein hohes Maß an Geschicklichkeit, Verlässlichkeit und Sensibilität erlernt. Das Reinforcement Learning bringt eine gewisse Komplexität mit sich. Wie ist dieses Belohnungssignal zu definieren? Wie stark sind die einzelnen Komponenten des Signales zu gewichten? Wie werden die gesammelten Signale ausgewertet, um die zukünftige Policy π festzulegen? Aus dieser Bachelorarbeit geht hervor, dass gerade Details in den Bewegungen des Roboters mit komplexen Belohnungsfunktionen einhergehen. Wenn die erlernte Policy π in der Simulation funktioniert, heißt das nicht, dass diese in der echten Welt funktionieren wird. Eine Simulation kann die reale Welt nicht exakt nachbilden. Des Weiteren ist zu beachten, dass für das betrachtete Problem der Zustands- und Aktionsraum so groß ist, dass diese den Einsatz neuronaler Netze erfordern. Das geht mit langen Berechnungszeiten einher. Außerdem ist die Erklärbarkeit des neuronalen Netzes und das damit verbundene Sicherheitsrisiko zu beachten, wenn der Entwickler dieses nicht nachvollziehen kann. Eine Möglichkeit, die Sicherheit des Systems zu erhöhen ist es, den Agenten bezüglich schlechter Aktionen mit negativen Signalen zu bestrafen. Schlechte Aktionen können zum Beispiel solche sein, die zu Selbstkollisionen führen. Doch dieser Lösungsansatz gewährleistet keine

Garantie. Die Sicherheit kann garantiert werden, indem, wie in [Smith u. a. 2022], der Aktionsraum stark eingeschränkt wird, doch das führt zum Verlust von Optimalität, falls optimale Aktionen außerhalb des eingeschränkten Bereichs liegen. Letztlich existiert der Ansatz eines analytischen Modells, das unsichere Aktionen, bevor sie vom Roboter ausgeführt werden, herausfiltert. Im Rahmen dieser Bachelorarbeit wurde keine wissenschaftliche Arbeit gefunden, die dieses Problem im Kontext quadrupedaler Roboter gelöst hat. Dieser Lösungsansatz bleibt eine offene Forschungsfrage.

1.3 Überblick

In Kapitel 2 werden grundlegende Konzepte der Bachelorarbeit dargestellt. Ein besonderer Schwerpunkt liegt auf dem Deep Reinforcement Learning. Es werden verschiedene Fragestellungen behandelt, deren Antworten weitere Methoden motivieren. Diese Fragestellungen beschäftigen sich mit Berechenbarkeitsaspekten bezüglich der Aktionsselektion und mit der Aktualisierungsregel für das zugrundeliegende neuronale Netzwerk. Zudem wird das Programmiergerüst „Gymnasium“ erklärt, das das für das Reinforcement Learning fundamentale Konzept des Markow-Entscheidungsproblems implementiert. Anschließend wird in Kapitel 3 das erste Teilsystem eingeführt, das Hinsetzen. Hier geht es darum, dass der Roboter erlernt, aus jeder willkürlichen Position, eine Sitzposition anzunehmen, aus der er aufstehen kann. Kapitel 4 erklärt, wie der Roboter das Aufstehen erlernen kann, um das in Kapitel 5 erlernte Laufen durchführen zu können. Die Bewegungen Hinsetzen, Aufstehen und Laufen sollen in Kapitel 6 in einem Controller zusammengefasst werden, sodass der Roboter gesteuert (Laufen) werden kann, mit der zusätzlichen Funktionalität der Falldetektion und des Aufrichtens (Hinsetzen und Aufstehen). Kapitel 7 ist der Abschluss dieser Bachelorarbeit. Es wird der Kern der aufgeführten Konzepte mit ihren Stärken und Schwächen hervorgehoben und abschließend wird eine weitere Methode vorgestellt, die die Vielseitigkeit des Aktionsraumes im Kontext des Reinforcement Learnings demonstriert.

2 Grundlagen des Deep Reinforcement Learnings

Das Grundlagenkapitel widmet sich dem theoretischen Hintergrund der in dieser Bachelorarbeit verwendeten Reinforcement Learning Methoden. Es werden Reinforcement Learning Algorithmen vorgestellt, wobei die Algorithmen Schwächen aufweisen, die die Einführung eines neuen Algorithmus motivieren. Als letzter Algorithmus wird Proximal Policy Optimization [Schulman u. a. 2017] vorgestellt. Dieser Algorithmus wurde im praktischen Teil der Bachelorarbeit verwendet. Abschließend wird das Simulationsprogramm und der Aktionsraum des Roboters erklärt.

2.1 Markow-Entscheidungsproblem

Die formale Grundlage für das Reinforcement Learning Problem ist ein Markow-Entscheidungsproblem, abgekürzt MDP (Markov Decision Process). Diese konkrete Ausprägung eines MDPs entstammt maßgeblich [Schulman u. a. 2015a], weil diese Arbeit der Grundstein für den weiteren Verlauf dieses Kapitels ist. Ein MDP wird definiert durch das Tupel $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, wobei $\mathcal{S}$ die Zustandsmenge, $\mathcal{A}$ die Aktionsmenge und $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ die Zustandsübergangsfunktion ist. Die Zustandsübergangsfunktion bildet ein Tripel aus jetzigem Zustand, jetziger Aktion und nächstem Zustand (s_t, a_t, s_{t+1}) auf eine Wahrscheinlichkeit ab. $r : S \to \mathbb{R}$ ist die Belohnungsfunktion, $\rho_0 : \mathcal{S} \to \mathbb{R}$ die Wahrscheinlichkeitsverteilung für den Anfangszustand und $\gamma \in (0, 1)$ der Diskontierungsfaktor. Dieser gibt den Wert einer l-Zeitschritte entfernten Belohnung $r(s_{t+l})$ zum Zeitschritt t an. Eine Belohnung $r(s_{t+l})$, die l Zeitschritte in der Zukunft liegt, ist nur γ^l mal so viel Wert wie wenn sie im jetzigen Zeitschritt erfahren werden würde. Sei π eine stochastische Policy. Das ist eine Wahrscheinlichkeitsverteilung, die für einen Zustand s bestimmt, wie wahrscheinlich es ist, eine Aktion a zu treffen: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Dann ist $\eta(\pi)$ die Erwartung an aufsummierten diskontierten erfahrenen Belohnungen, von hier an benannt als diskontierter erwarteter Gewinn:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right], \text{ mit} \tag{2.1}$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \tag{2.2}$$

Ferner wird definiert:

$$V^{\pi,\gamma}(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right], \tag{2.3}$$

$$Q^{\pi,\gamma}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right], \tag{2.4}$$

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t), \text{ mit} \tag{2.5}$$

$$a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \tag{2.6}$$

$V^{\pi,\gamma}$ gibt den diskontierten erwarteten Gewinn an, unter der Annahme, dass aus einem Zustand s_t der Policy π gefolgt wird. Diese Funktion wird von jetzt an Zustand-Wert-Funktion genannt. $Q^{\pi,\gamma}$ gibt den diskontierten erwarteten Gewinn an, unter der Annahme, dass aus einem Zustand s_t eine Aktion a_t getroffen wird und daraufhin der Policy π gefolgt wird. Diese Funktion wird von jetzt an Zustand-Aktion-Wert-Funktion genannt. Die Advantage-Funktion $A^{\pi,\gamma}$ setzt beide Größen ins Verhältnis zueinander. Gemäß obiger Definitionen lässt sich ein Maß für Optimalität definieren. Eine Policy π ist als besser definiert als eine andere Policy π', wenn $V^{\pi,\gamma}(s) \geq V^{\pi',\gamma}(s)$ für alle $s \in \mathcal{S}$ gilt. Es gibt immer mindestens eine Policy, die besser oder gleich allen anderen Policies ist. Diese Policy ist als optimal definiert. Es ergibt sich die optimale Zustand-Wert-Funktion $V_*(s) = \max_\pi V^{\pi,\gamma}(s)$, für alle $s \in \mathcal{S}$. Das gleiche gilt für die optimale Zustand-Aktion-Wert-Funktion $Q_*(s, a) = \max_\pi Q^{\pi,\gamma}(s, a)$, für alle $s \in \mathcal{S}$ und in s möglichen Aktionen $\mathcal{A}(s)$ [Sutton u. Barto 2018].

2.1.1 Deep Q-Learning

[1] Die betrachteten Methoden, eine Policy π zu erlernen, beschränken sich auf solche, die für hochdimensionale, kontinuierliche Zustands- und Aktionsräume geeignet sind. Hochdimensional und kontinuierlich impliziert, dass Zustands- und Aktionsraum zu groß sind, um in einer Lookup-Tabelle festgehalten zu werden. Das heißt, dass über verschiedene Zustände und Aktionen generalisiert werden muss, um der Größenordnung gerecht zu werden. Dies wiederum rechtfertigt den Einsatz tiefer neuronaler Netze. Die Verwendung tiefer neuronaler Netze zum Lösen hochdimensionaler Reinforcement Learning Probleme erlangte große Aufmerksamkeit durch [Mnih u. a. 2013], mittels „Deep Q-learning“ zum Lösen von Atari-Spielen. In dieser Arbeit ist allerdings nur der Zustandsraum hochdimensional. Der Aktionsraum $\mathcal{A} = \{1, ..., K\}$ ist diskret und niedrigdimensional. Ein für das Deep Q-Learning fundamentales Konzept ist die Bellmansche Optimalitätsgleichung [Sutton u. Barto 2018]:

$$Q_*(s_t, a_t) = \mathbb{E}\left[r(s_t) + \gamma \max_{a'} Q_*(s_{t+1}, a')|s_t, a_t\right] \tag{2.7}$$

Ist diese bekannt, so ist die optimale Aktion jene, die Gleichung 2.7 maximiert. Diese ist zunächst aber nicht bekannt und muss approximiert werden. In [Mnih u. a. 2013] geschieht das durch die iterative Minimierung der Loss-Funktion:

$$L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s_t, a_t; \theta_i))^2\right] \tag{2.8}$$

[1] Die in diesem Unterkapitel verwendete Notation weicht leicht von der in der Originalpublikation [Mnih u. a. 2013] verwendeten Notation ab, um den Kern der Aussagen hervorzuheben und um die hier gewählte Notation beibehalten zu können.

durch das Gradientenabstiegsverfahren. Das neuronale Netz Q_{θ_i} schätzt die Zustand-Aktion-Wert Funktion aus Gleichung 2.4. Weiter wird der Zielwert y_i der i-ten Iteration definiert

$$y_i = \mathbb{E}\left[r(s_t) + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{i-1}|s_t, a_t)\right]$$

Der Zielwert y_i legt fest, welche Funktion die Schätzfunktion Q_{θ_i} approximieren soll. Der Zielwert y_i ist so gewählt, dass durch das Optimieren der Loss-Funktion 2.8 die Parameter so verändert werden, dass die Bellmansche Optimalitätsgleichung 2.7 unter der Parametrisierung des letzten Zeitschrittes approximiert wird. Bei der Optimierung von Gleichung 2.8 werden die Parameter der vorherigen Iteration θ_{i-1} festgehalten. Das für die Bildung des Zielwertes y_i notwendige Quadrupel, die Erfahrung $\exp_t = (s_t, a_t, r(s_t), s_{t+1})$ entstammt einem Replay Buffer. Das Wiederverwenden von vergangenen Erfahrungen nennt sich Experience Replay [Lin 1992] und hat allgemein den Vorteil bereits erzeugte Daten wiederverwenden zu können. Beim Training wird gleichmäßig („uniformly“) aus dem Replay Buffer „gesamplet“. Das hat zur Folge, dass der Zielwert y_i durch eine Erfahrung gebildet werden kann, die einer Policy entstammt, deren Parameter nicht mehr den aktuellen Parametern entsprechen. Das nennt sich Off-Policy. Um zu verstehen, warum Off-Policy trainiert wird, anstatt nur mit den Daten zu trainieren, die ein Agent unter der aktuellen Parametrisierung in einem Spielzug erfährt, nehme der Leser an, dass der Agent in einem Spiel die Aktion *Links* wählt. Würde On-Policy trainiert werden, so würden die Trainingsdaten dominiert werden von Erfahrungen, die der linken Seite des Spielfeldes entstammen. Das hieße, dass die Policy nicht gleichermaßen für alle relevanten Zustände des Spiels trainiert wird. Das Samplen aus diesem Buffer hat somit das Ziel, Korrelationen in den Trainingsdaten zu brechen. Die Aktionsselektion des Deep Q-Learnings basiert darauf, über alle möglichen Aktionen in einem Zustand zu optimieren, um die Zustand-Aktion-Wert-Funktion zu maximieren:

$$a = \arg\max_a Q^*(s_t, a; \theta) \tag{2.9}$$

Für hochdimensionale kontinuierliche Probleme ist das Optimieren über die Zustand-Aktion-Wert-Funktion als Aktionsselektion suboptimal, weil der Zustands- und Aktionsraum sehr groß ist. Zahlreiche Anwendungsfälle erfordern eine sofortige Aktionsselektion. Das Lösen eines Optimierungsproblems für jeden erfahrenen Zustand widerspricht dem. Eine Diskretisierung des Aktionsraumes würde diesen zwar kleiner machen, führt aber möglicherweise zum Verlust der optimalen Aktion in einem Zustand und ist bezüglich der Größe noch immer schwierig zu lösen. Als Beispiel werde angenommen, dass der Aktionsraum des Unitree A1 Roboters diskretisiert wird. Auf jedes Gelenk sind maximal $\pm 33Nm$ anwendbar. Das heißt, dass eine mögliche Diskretisierung

$$\text{A1} = \{-33.0, -32.9, -32.8, ..., 0.0, ..., +32.8, +32.9, +33.0\}$$

mit $|\text{A1}| = 661$ sein könnte. Mit zwölf Freiheitsgraden ergeben sich somit 12^{661} mögliche Aktionen, über die in jedem Zustand optimiert werden müsste.

2.1.2 Policy Gradient Methoden

Soeben wurde beschrieben, dass eine Policy, die aus einem Zustand s eine Aktion a als ein Maximum über die Zustand-Aktion-Wert-Funktion berechnet, für hochdimensionale kontinuierliche Probleme ineffizient ist. Deshalb wird nun eine andere Klasse von Policies eingeführt, sogenannte Policy-Gradient Methoden [Sutton u. a. 1999]. Diese Policies benötigen keine Wert-Funktionen, um eine Aktion zu berechnen. Die Ausgabe der Aktion erfolgt direkt basierend auf der derzeitigen Parametrisierung der Policy. Die Wahrscheinlichkeit, dass ein Agent zu einem Zeitpunkt t eine Aktion a_t aus einem Zustand s_t unter einer Parametrisierung θ_t wählt ist $\pi(a_t, s_t, \theta_t)$. Eine einfache Policy Gradient Methode ist es, den Gradienten, bezüglich der Parameter der Policy, des in Gleichung 2.1 definierten diskontierten erwarteten Gewinns zu berechnen und die Parameter in die Richtung des Gradienten mittels des Gradientenaufstiegsverfahrens (Gradient-Ascent) zu aktualisieren. Nach [Schulman u. a. 2015b] ist der Policy Gradient allgemein definiert als:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k}\right] \tag{2.10}$$

mit der Aktualisierungsregel

$$\theta_{k+1} = \theta_k + \alpha_k g \tag{2.11}$$

Der Gradient $\nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k}$ zeigt in Richtung des steilsten Anstieges der logarithmischen Wahrscheinlichkeit eines Zustand-Aktion-Paares (s_t, a_t) im Parameterraum. Abhängig von Vorzeichen und Betrag von Ψ_t werden die Parameter entsprechend angepasst, um die Wahrscheinlichkeit in einem Zustand s_t eine Aktion a_t auszuführen, zu erhöhen oder zu verringern. Eine konkrete Ausprägung von Ψ_t wird in Abschnitt 2.1.2.3 erläutert.
Das Problem des Gradientenaufstiegsverfahrens liegt in der Wahl der Schrittgröße α. Das Verändern der parametrisierten Policy führt dazu, dass sich die Wahrscheinlichkeitsverteilung der auftretenden Zustände verändert, weil diese vom Verhalten bzw. von der parametrisierten Policy abhängt. Eine zu groß eingestellte Schrittgröße, bzw. ein einziger zu großer Schritt, kann dazu führen, dass der Agent nun in einem Teil des Zustandsraumes ist, der nicht relevant für das zugrundeliegende Reinforcement Learning Problem ist. Deshalb wird der Agent keine guten Belohnungen erhalten. Da die Trainingsdaten, anders als im Supervised-Learning, nicht schon im Voraus existieren, sondern erst während des Trainings erzeugt werden, leiden auch die Trainingsdaten. Es ist es fraglich, ob der Agent es schaffen wird aus diesen Trainingsdaten, sein Verhalten so zu verändern, dass er wieder in den relevanten Teil des Zustandsraumes gelangt. Ausgehend von dieser Problematik wird das Prinzip der Trust Region Policy Optimization erläutert.

2.1.2.1 Trust Region Policy Optimization

Die in diesem Unterkapitel aufgeführten Konzepte stammen aus [Schulman u. a. 2015a]. Trust Region Policy Optimization versucht die oben beschriebene Problematik zu umgehen, indem der Parametrisierung der Policy eine Bedingung aufgesetzt

wird, die sich darauf bezieht, wie weit sich die neue Policy von der alten Policy entfernen darf. Dieses Konzept liefert einen Algorithmus, der eine monoton steigende Folge von Policies garantiert. Die monotone Steigung bezieht sich auf den diskontierten erwarteten Gewinn der resultierenden Policies. Die Garantie basiert auf theoretischen Erkenntnissen aus [Kakade u. Langford 2002].

Diese Metrik definiert die diskontierte Wahrscheinlichkeit für das Auftreten eines Zustandes, über einen Zeithorizont:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + ... \tag{2.12}$$

Es gilt $s_0 \sim \rho_0$. Die Aktionen werden gemäß π selektiert.

Die folgende Gleichung definiert den erwarteten Gewinn einer Policy $\tilde{\pi}$ im Verhältnis zu einer anderen Policy π:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s_t, a_t) \tag{2.13}$$

$\rho_{\tilde{\pi}}(s)$ erschwert es über Gleichung 2.13 direkt zu optimieren. Stattdessen wird über eine lokale Approximation optimiert:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s_t, a_t) \tag{2.14}$$

Die lokale Approximation 2.14 nutzt $\rho_\pi(s)$ anstelle von $\rho_{\tilde{\pi}}(s)$ und ignoriert somit die mit Policy-Aktualisierungen einhergehenden Veränderungen der in Gleichung 2.12 beschriebenen Metrik. Für eine parametrisierte Policy π_θ, sodass $\pi_\theta(a|s)$ eine differenzierbare Funktion nach θ ist, gilt nach [Kakade u. Langford 2002] folgende Gleichheit für alle Parametervektoren θ_0

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}) \tag{2.15}$$

$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_0}, \tag{2.16}$$

Die Gleichheit der Gradienten impliziert, dass ein suffizient kleiner Schritt $\pi_{\theta_0} \to \tilde{\pi}$, der $L_{\pi_{\theta_{\text{old}}}}$ verbessert, auch η verbessert. Doch was ist ein „suffizient kleiner Schritt“? Seien $\epsilon = \max_{s,a} |A_\pi(s,a)|$, $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ und $D_{\text{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{KL}(\pi(\cdot|s)||\tilde{\pi}(\cdot|s))$. In [Schulman u. a. 2015a] wird gezeigt

$$\eta(\tilde{\pi}) = L_\pi(\tilde{\pi}) - C D_{\text{KL}}^{\max}(\pi, \tilde{\pi}), \tag{2.17}$$

Gleichung 2.17 inkludiert einen Strafterm, sodass das Abweichen von π bestraft wird. Die Strafe wird stärker, je größer der Faktor C ist. D_{KL} ist die Kullback-Leibler Divergenz und ein Maß für die Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen, in diesem Fall Policies. Es werde zur Kenntnis genommen, dass $L_\pi(\tilde{\pi})$ nur eine lokale Approximation von $\eta(\tilde{\pi})$, dem erwarteten Gewinn einer Policy $\tilde{\pi}$ im Verhältnis zu einer anderen Policy π ist. Deshalb reicht es nicht lediglich $L_\pi(\tilde{\pi})$ zu maximieren.

Die aufgeführten Überlegungen rechtfertigen ein Optimierungsproblem, dessen iterative Lösung garantiert, dass die diskontierten erwarteten Gewinne der resultierenden Policies eine monoton steigende Folge $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq ....$ bilden:

$$\underset{\pi_\theta}{\text{maximiere}} \ \left[L_{\pi_{\theta_{\text{old}}}}(\pi_\theta) - C D_{\text{KL}}^{\max}(\pi_{\theta_{\text{old}}}, \pi_\theta) \right] \tag{2.18}$$

Die Aktualisierungsschritte seien unter Verwendung des Strafterms sehr klein. Deshalb wird vorgeschlagen, die Eingrenzung der Abweichung der alten und neuen Policy durch eine Schranke δ festzulegen. Das Optimierungsproblem lautet, wobei $L_{\theta_{\text{old}}}$ zum besseren Verständnis ausgeschrieben wird:

$$\underset{\theta}{\text{maximiere}} \ \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s,a) \tag{2.19}$$

$$\text{beschränkt durch } \mathbb{E}_{s \sim \rho_{\theta_{old}}} \left[D_{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s), \pi_\theta(\cdot|s) \right] \right] \leq \delta \tag{2.20}$$

wobei die Beschränkung die erwartete KL-Divergenz ist, wenn Zustände gemäß der Zustandsverteilung der alten Policy gesehen werden. Dieses Optimierungsproblem wird als vereinfachter, zu schätzender Erwartungswert aufgeschrieben:

$$\underset{\theta}{\text{maximiere}} \ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \tag{2.21}$$

$$\text{beschränkt durch } \hat{\mathbb{E}}_t \left[D_{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta \tag{2.22}$$

Die Herleitung, die es ermöglicht, das Optimierungsproblem als Schätzung eines Erwartungswertes aufzuschreiben bzw. der Übergang von Gleichung 2.19 zu Gleichung 2.21, befindet sich in [Schulman u. a. 2015a], Kapitel 5. Sie wurde weggelassen, weil sie rein algebraisch ist. Das in diesem Unterkapitel gezeigte Konzept, die Abweichung zwischen alter und neuer Policy einzugrenzen, ist für den im praktischen Teil dieser Bachelorarbeit verwendeten, auf Trust Region Policy Optimization aufbauenden, Algorithmus namens Proximal Policy Optimization wichtig.

2.1.2.2 Proximal Policy Optimization

Die in diesem Unterkapitel aufgeführten Konzepte stammen aus der Publikation [Schulman u. a. 2017], die Proximal Policy Optimization (PPO) damit motiviert, dass diese Methode einfacher zu implementieren ist, allgemeiner und empirisch eine bessere Dateneffizienz aufweist als Trust Region Policy Optimization (TRPO). Die Stärke von TRPO, nämlich dass der Abweichung zwischen neuer und alter Parametrisierung einer Policy in einem Aktualisierungsschritt eine gewisse Schranke aufgesetzt wird, um exzessiv große Aktualisierungen zu vermeiden, wird beibehalten. Die Zielfunktion für PPO ist die folgende:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{2.23}$$

mit $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$. Zum besseren Verständnis möge der Leser annehmen, dass $\hat{A}_t$, die Schätzung der Advantage-Funktion 2.5, positiv ist. Das heißt, dass der Term

$r_t(\theta)\hat{A}_t$ größer wird, wenn r_t größer wird, also $\pi_\theta(a_t|s_t)$ größer wird. Uneingeschränkte Veränderungen der Parametrisierung der Policy können wie in Kapitel 2.1.2 beschrieben zu Divergenz führen. Das Einschränken der Policy passiert durch die min-Funktion und die Clip-Funktion. Ist $\hat{A}_t$ positiv und wird $\pi_\theta(a_t|s_t)$ so weit vergrößert, dass $r_t(\theta) > 1 + \epsilon$, so hat das keinen Einfluss mehr auf die Zielfunktion.

2.1.2.3 Generalized Advantage Estimation

Eine mögliche Instanziierung von Ψ_t in Gleichung 2.10 ist $\Psi_t = A^{\pi,\gamma}(s_t, a_t)$.

$$g^\gamma = \mathbb{E}\left[\sum_{t=0}^{\infty} A^{\pi,\gamma}(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{2.24}$$

Der Leser bemerke den Diskontierungsfaktor γ. In diesem Kontext reduziert dieser die Varianz auf Kosten der Einführung eines „Bias“ [Marbach u. Tsitsiklis 2003], [Kakade 2001], [Thomas 2014]. Der Bias stellt einen systematischen Fehler dar, der sich daraus ergibt, dass die erhaltenen Belohnungen durch den Diskontierungsfaktor verfälscht werden. Diese Wahl begründet sich damit, dass diese Form des Policy Gradienten eine geringe Varianz aufweist, weil die Advantage-Funktion einen sogenannten Baseline-Term verwendet, durch welchen es möglich ist, den Wert einer Aktion in einem Zustand $Q^\pi(s, a)$ in Bezug zu setzen mit dem Wert des Standardverhaltens der Policy $V^\pi(s)$ [Schulman u. a. 2015b]. Die Advantage-Funktion $A^{\pi,\gamma}$ ist eine nicht bekannte Größe und muss geschätzt werden. Mittels der Schätzung der Advantage-Funktion $A^{\pi,\gamma}$ kann der Policy-Gradient 2.24 geschätzt werden durch:

$$\hat{g} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty} \hat{A}_t^n \nabla_\theta \log \pi_\theta(a_t^n|s_t^n) \tag{2.25}$$

wobei N die Anzahl erfahrener Episoden ist, die für den Aktualisierungsschritt verwendet werden. Die in dieser Bachelorarbeit verwendete Implementierung von Proximal Policy Optimization macht Gebrauch von „Generalized Advantage Estimator“ für die Schätzung $\hat{A}_t$, definiert in [Schulman u. a. 2015b] wie folgt:

$$\hat{A}_t^{GAE(\gamma,\lambda)} := \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}^V, \tag{2.26}$$

mit $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$, wobei V die Zustand-Wert Funktion aus Gleichung 2.3 schätzt. λ ist ein Parameter, dessen Regulierung das Bias-Varianz Verhältnis beeinflusst:

$$GAE(\gamma, 0) : \hat{A} := \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.27}$$

$$GAE(\gamma, 1) : \hat{A} := \sum_{l=0}^{\infty}\gamma^l \delta_{t+l} = \sum_{l=0}^{\infty}\gamma^l r_{t+l} - V(s_t) \tag{2.28}$$

$GAE(\gamma, 1)$ liefert eine erwartungstreue Schätzung von Gleichung 2.24, unabhängig von der Präzision von $V(s_t)$, hat aber eine hohe Varianz, begründet in der hohen Anzahl aufzusummierender Terme. $GAE(\gamma, 0)$ liefert zwar nur eine erwartungstreue

Schätzung von Gleichung 2.24, falls die Schätzung V der Zustand-Wert-Funktion gleich der wahren Zustand-Wert-Funktion $V^{\pi,\gamma}$ ist, hat aber eine geringere Varianz. Zusammengefasst: „The generalized advantage estimator for $0 < \lambda < 1$ makes a compromise between bias and variance, controlled by parameter λ“ [Schulman u. a. 2015b]. In dieser Bachelorarbeit wurden die Parameter gewählt als $\gamma = 0.99$ und $\lambda = 0.95$.

2.2 Isaac Gym

Die Ergebnisse dieser Bachelorarbeit entstanden unter Verwendung des Simulationsprogrammes Isaac Sim [NVIDIA 2023a] und des Gymnasium-Programmiergerüsts von OpenAI [FaramaFoundation 2023]. Dieses Programmiergerüst wurde von NVIDIA weiter konkretisiert, sodass es auf Isaac Sim Anwendungen zugeschnitten ist. Das Resultat ist ein erweitertes Programmiergerüst, nämlich Isaac Gym bzw. „OmniIsaacGymEnvs“ [NVIDIA 2023b]. Die Besonderheit von Isaac Gym ist, dass Observationen, Belohnungen und Aktionspuffer alle auf der GPU prozessiert werden, sodass diese Daten nicht zwischen GPU und CPU transferiert werden müssen. Das ermöglicht einen deutlichen Geschwindigkeitszuwachs [Makoviychuk u. a. 2021].
Schematisch lässt sich das Gym-Programmiergerüst in verschiedene Funktionen gliedern. In der **Step**-Funktion wird die Simulationsumgebung um einen Zeitschritt erhöht. Die Eingabe für diese Funktion ist die Ausgabe (Aktion) des Policy-Netzwerkes (Die Policy π als neuronales Netzwerk). Sie lässt sich in den **Pre-Physics-Step** und in den **Post-Physics-Step** unterteilen. Im Pre-Physics-Step wird die von dem Policy-Netzwerk entgegengenommene Aktion a_t auf den Agenten angewendet. Möglicherweise wird die Ausgabe des Policy-Netzwerkes vom Anwender vorverarbeitet. Im Post-Physics-Step werden die Informationen, die der Agent nach Ausführen der Aktion a_t erfährt, verarbeitet. Zum einen geschieht das in der **Get-Observations**-Funktion. Nachdem der Agent die Aktion angewendet hat, wird die Zustandsvariable des Agenten s_t in den Zustand s_{t+1} überführt. Beispielsweise könnte sich die Geschwindigkeit des Agenten verändert haben, was erfasst werden muss für den nächsten Zeitschritt. Weiter existiert die **Calculate-Metrics**-Funktion. Basierend auf dem neuen Zustand des Agenten wird die erhaltene Belohnung berechnet. In **Reset** wird, falls eine bestimmte vordefinierte Bedingung eintritt, wie beispielsweise ein Time-Out, die Welt des Agenten in den Startzustand s_0 zurückgesetzt. In der realen Welt kann nicht garantiert werden, dass der Agent exakt im gleichen Zustand beginnt, weshalb es möglich ist, den Agenten in einen vom Standard-Startzustand abweichenden, randomisierten, Zustand zurückzusetzen. Isaac Gym verwendet standardmäßig eine Drittpartei Programmierbibliothek als Implementierung für Proximal Policy Optimization, die in dieser Bachelorarbeit verwendet wurde [Makoviichuk u. Makoviychuk 2021]. Das heißt, dass Proximal Policy Optimization nicht selber implementiert wurde. Im Programmiergerüst „OmniIsaacGymEnvs“ sind Dateien „anymal.py“ und „anymal_terrain.py“ enthalten. Die Software dieser Bachelorarbeit nutzt zu einem gewissen Grad bereits vorhandene Funktionalitäten dieser Dateien.

2.3 Aktionsraum

Ein weiterer Unterschied beim Training von Bewegungsabläufen mittels des Deep Reinforcement Learnings im Kontext von Robotern besteht im Aktionsraum. **Zielwinkel**: Die von der Policy zu wählende Aktion ist ein Vektor von Zielwinkeln. Die Aktion $a = \phi_d$ wird wie folgt berechnet: $\phi_d = ko_t + \phi_t$ oder $\phi_d = ko_t + \phi_n$. Die Ausgabe des Policy-Netzwerkes ist o_t und k ist ein einzustellender Skalierparameter. ϕ_t gibt die derzeitige und ϕ_n gibt eine standardmäßige Gelenkkonfiguration an. Die Wahl ist problemabhängig. Dieser Zielwinkel wird von einem PD-Regler umgewandelt in Drehmomente: $\tau^i = k_p^i(\hat{\phi}^i - \phi^i) + k_d^i(\hat{\dot{\phi}}^i - \dot{\phi}^i)$. τ^i gibt das berechnete Drehmoment für das i-te Gelenk an, dessen Winkelposition ϕ^i ist. $\dot{\phi}^i$ ist die Geschwindigkeit dieses Gelenkes, während $\hat{\phi}^i$ und $\hat{\dot{\phi}}^i$ Zielwerte sind, wobei die Zielgeschwindigkeit eines Gelenkes $\hat{\dot{\phi}}^i = 0$ ist. k_p^i, k_d^i sind manuell einzustellen. In dieser Bachelorarbeit wurden für alle Gelenke die Werte $k_p = 20.0$ und $k_d = 0.5$ gewählt. **Direkte Drehmomentregelung**: Eine andere Variante ist es, die Drehmomente direkt auf die Aktuatoren des Roboters zu senden, auch „Torque Control“ genannt. Im weiteren Verlaufe wird diese Aktionsform direkte Drehmomentregelung genannt. Die Ausgabe der Policy kann zunächst mit einem Skalierungsparameter k vorverarbeitet werden, ehe sie an den Roboter gesendet wird.
Einführend zur Fragestellung der Wahl des optimalen Aktionsraumes ist die Arbeit von [Peng u. Van De Panne 2017] zu nennen, welche drei Kriterien aufstellt. Das erste Kriterium ist die **Policy Leistung und Lerngeschwindigkeit:** Die Arbeit sagt, dass die direkte Drehmomentregelung für komplexe Fortbewegung besonders schlecht abschneidet, während der PD-Regler allgemein sehr gut abschneidet und schneller lernt. Ein weiteres Kriterium ist die **Policy Robustheit:** Die Robustheit der Policy wurde geprüft, indem der Roboter externen Störungen, in Form von randomisierten Krafteinflüssen auf den Rumpf, ausgesetzt wurde. Das Ergebnis lautet: „Overall, the PD policies are among the most robust for all the motions.“. Das letzte Kriterium ist die **Abtastrate:** Es wurde überprüft, wie sich die Leistung einer Policy in Abhängigkeit der Abtastrate und des Aktionsraumes verhält. Das Ergebnis sei, dass der PD-Regler eher mit niedrigeren Abtastraten umgehen kann als die direkte Drehmomentregelung. In den fortlaufenden Kapiteln werden diese Kriterien bezüglich der verschiedenen Bewegungsabläufe unter bestimmten Variationen evaluiert.

3 Bewegungsablauf: Hinsetzen

Die erste Art der Bewegung, die der Roboter erlernen soll, ist das Hinsetzen. Das Sitzen ist definiert als eine im Voraus bestimmte Gelenkkonfiguration, aus der der Roboter aufstehen kann. Das Ziel ist, dass der Roboter aus jeder beliebigen Konfiguration eine Sequenz von Aktionen ausführen kann, die ihn in die Sitzkonfiguration überführt. Die Erfüllung dieser Aufgabe ist die Bedingung für die Erfüllung der nächsten Aufgabe, dem Aufstehen, siehe Kapitel 4. Dieses Verhalten wird in der Literatur auch „Recovery“ genannt. Das Problem der Recovery ist nicht nur interessant im Kontext quadrupedaler Roboter. Es werde an ein Auto gedacht, das aus der Spur gerät oder an einen Helikopter, der die Balance verliert. Diese Situationen charakterisieren sich dadurch, dass sie den sicheren Zustandsraum verlassen und wieder in diesen zurückgeführt werden müssen.

3.1 Vorhandene Verfahren

Ein vorhandenes Verfahren ist es, den Roboter von der Ausgangslage über Zwischenzustände in die Sitzkonfiguration zu überführen [Kakiuchi u. a. 2017]. Die Fallerkennung geschieht durch das Vergleichen des Ist-Zustandes mit einem Soll-Zustand. Die Zustandserkennung geschieht per Sensoren und ist definiert als ein Vektor mit n Komponenten. Die Komponenten geben Auskunft über den Ort und die Geschwindigkeit des Roboters und weiteren für das Problem relevanten Informationen. Die Distanz zweier Zustandsvektoren x und y wird bemessen durch

$$d(x, y) = \sqrt{\sum_{i=0}^{n} \omega_i (x_i - y_i)^2} \tag{3.1}$$

wobei

$$\omega = (\omega_0, ..., \omega_n) \tag{3.2}$$

ein Vektor ist, der die Komponenten des Zustandes gewichtet. Ist die Distanz zwischen Soll und Ist-Wert zu groß, so wird angenommen, dass der Roboter gefallen ist, was die Hinsetzprozedur einleitet. Sowohl der Gewichtsvektor, als auch die Schwelle, die bestimmt, ab wann eine Distanz zu groß ist, müssen manuell eingestellt werden. Aus dem Ist-Zustand wird der gemäß Gleichung 3.1 naheliegendste, vorher definierte, Zwischenzustand angesteuert. Während eines Zustandsüberganges muss die Fallerkennungsprozedur aktiv sein. Die Ausprägung des Soll-Zustands während eines Überganges muss bereits bekannt sein und somit vorher bei einem erfolgreichen Zustandsübergang aufgenommen worden sein.

3.1.1 Nachteile

Das soeben vorgestellte Verfahren charakterisiert sich durch verschiedene Nachteile. **Optimalität der Zwischenzustände:** Die Zustände, die den Zustand unmittelbar nach einem Fall, und den Sitzzustand, aus dem der Roboter aufstehen kann, verbinden, müssen explizit vorher definiert werden. Was sind geeignete Zwischenzustände? Die Eignung kann sich dadurch kennzeichnen, dass die Zustandsabfolge möglichst viel Sicherheit gewährleistet. Zusätzlich ist die Zeit für eine vollständige Zustandsabfolge ein Kriterium. Möglicherweise heißt Optimalität eine Gewichtung der beiden Kriterien. Aber selbst wenn ein Kriterium ausgewählt wurde, stellt sich noch immer die Frage, wie die Zustände auszuwählen sind, sodass dieses Kriterium bestmöglich erfüllt wird. Und wie soll das gemessen werden? Das Reinforcement Learning Schema bietet durch das Konzept von Belohnungen eine Möglichkeit eine Metrik für den Erfolg einer Prozedur aufzustellen. **Anzahl der Zwischenzustände:** Soll die Hinsetzprozedur komplexer und detailreicher gestalten werden, so müssten mehr Zwischenzustände und entsprechende Übergänge implementiert werden. Gerade für das vorgestellte konventionelle Verfahren des Hinsetzens ist das schwierig. Die Soll-Zustände müssen im Voraus aufgezeichnet werden, was offensichtlich bei steigender Zustandsanzahl ein immer größerer Aufwand wird. Auch wenn es für das Aufnehmen selber einen Automatismus gibt, so muss dies dennoch bei jeder Erweiterung aufs Neue getan werden. Die Distanzschwelle müsste für jeden neuen Übergang manuell definiert werden. **Erweiterbarkeit:** Soll der Zustandsgraph erweitert werden, so gibt es die Gefahr, dass vorherige Zustandsübergänge nicht mehr gültig oder redundant sind. Es wäre manuell zu überprüfen, ob vorher zum Ziel führende Zustandsfolgen noch immer gültig sind, oder ob diese durch die Hinzunahme neuer Zustände zu Fehlverhalten führen.

3.2 Hinsetzen mittels Reinforcement Learning

Die in Kapitel 2.1 erklärten Ideen werden nun verwendet, damit der Unitree A1 erlernen kann, sich aus einer randomisierten Konfiguration, hinzusetzen. Dies soll der Roboter aus Erfahrungen (Trainingsdaten) erlernen, die er in der Simulation macht. Dazu wird der Unitree A1 in der Simulation aus der Luft auf den Boden abgeworfen. Die Gelenkkonfiguration wird zu Beginn einer jeden Episode randomisiert. Der gewählte Ansatz folgt maßgeblich dem in [Lee u. a. 2019] vorgestellten Ansatz.

3.2.1 Zustandsdarstellung

In Tabelle 3.1 ist die Darstellung der Zustandsvariablen zu sehen. Diese Wahl der Zustandsvariablen weicht von [Lee u. a. 2019] wie folgt ab: Die Referenzpublikation verwendet TSIF [Bloesch u. a. 2017] für die Schätzung der Position und Ausrichtung des Roboters. Wenn der Roboter fällt, so sind manche Schätzungen nicht mehr zuverlässig und werden deshalb aus der Zustandsvariablen exkludiert. Im Rahmen der Bachelorarbeit wurde beschlossen, das System nicht auf den realen Roboter zu übertragen, weshalb die Problematik außen vor gelassen wurde. Der zweite Unterschied ist, dass die „History of joint position error & velocity“ von der Zustandsvariablen ex-

kludiert wurde, bzw. die Historie nur einen Zeitschritt zurückreicht, um eine möglichst kompakte Zustandsdarstellung zu gewährleisten. Außerdem werden nicht Zielwinkelfehler in den Zustand inkludiert, sondern lediglich die Gelenkkonfiguration, gemessen im Bogenmaß. Ein Zielwinkelfehler ist der Fehler zwischen der Aktion als Zielwinkel und dem tatsächlich angenommen Winkel an einem Gelenk.

3.2.2 Belohnungsfunktion

Tabelle 3.2 stellt die Belohnungsfunktion für die Aufgabe des Hinsetzens dar. **Abweichung:** Gibt die Abweichung der jetzigen Gelenkkonfiguration von der Zielgelenkkonfiguration/Sitzkonfiguration an, wobei immer die kleinere Winkeldistanz gewählt wird. Die Abweichung wird in Form der euklidischen Norm bestraft. **Ausrichtung:** Die Ausrichtung des Roboters wird wie folgt bestimmt: Sei $[0, 0, -1]^T$ der Vektor, der die projizierte Schwerkraft des Roboters bei gerader Ausrichtung angibt (Soll-Wert). Der Vektor e_g gibt die derzeitige projizierte Schwerkraft des Roboters an (Ist-Wert). Die Distanz dieser Vektoren wird besonders stark gewichtet, weil es hoch prioritär ist, dass der Roboter schnellstmöglich gerade ausgerichtet ist. **Gelenkgeschwindigkeit, Gelenkbeschleunigung, Drehmoment, Aktionsrate:** Diese Strafterme sind sekundäre Ziele und sollen ermöglichen, dass der Roboter eine möglichst flüssige, energiesparende und sichere Bewegung erlernt. Es lässt sich allgemein argumentieren, dass eine hohe Gelenkgeschwindigkeit/-beschleunigung unsicherer ist als eine geringere. Höhere Drehmomente implizieren höheren Energieverbrauch. Die Aktionsrate ist ein Distanzmaß zwischen der im letzten und in diesem Zeitschritt gewählten Aktion. Die Aktionen sollen möglichst ähnlich zueinander sein, sodass die Bewegung möglichst „glatt“ ist. Da diese Terme nicht Teil des primären Zieles sind (und entsprechend in den anderen Teilsystemen ebenfalls verwendet werden) werden diese mit einem Kurrikulum-Faktor k_c multipliziert, eine Idee entnommen aus [Hwangbo u. a. 2019]. Zu Beginn ist $k_c = k_{c,0} \in (0, 1)$. Anschließend gilt folgende Aktualisierungsregel $k_{c,j+1} \leftarrow (k_{c,j})^{k_d}$. In dieser Bachelorarbeit wurde $k_0 = 0.3$, $k_d = 0.997$ gewählt und alle tausend Simulationszeitschritte aktualisiert. Diese Folge konvergiert gegen 1, sodass der komplette Strafterm mit fortlaufendem Training immer stärker in Erscheinung tritt. Diese Strategie ermöglicht es dem Agenten, sich zunächst auf die eigentliche Aufgabe des Trainings zu konzentrieren und später sekundäre Ziele anzuvisieren. [Hwangbo u. a. 2019] beschreibt:
„Low penalty on joint torque and velocity results in unnatural motions whereas high penalty on them results in a standing behavior. The main reason for the standing behavior is that such a behavior is already a good local minimum when there is high penalty associated with motion.“
Diesem Zitat wird entnommen, dass ein völliges außer Acht lassen der sekundären Ziele in „unnatürlichen Bewegungen“ mündet, aber auch, dass die sekundären Ziele in Konflikt mit den primären Zielen stehen können. Eine adaptive Gewichtung der sekundären Ziele durch den soeben erwähnten Kurrikulum-Faktor ist ein Lösungsansatz für dieses Problem.

Zustandsdarstellung
$\text{Lineargeschwindigkeit}_t$
$\text{Rotationsgeschwindigkeit}_t$
$\text{Projizierte Schwerkraft}_t$
$\text{Gelenkposition}_{t,t-1}$
$\text{Gelenkgeschwindigkeit}_{t,t-1}$
Aktion_{t-1}

Tabelle 3.1: Zustandsdarstellung für die Aufgabe des Hinsetzens

Belohnungsfunktion		
Metrik	Term	Gewicht
Abweichung	$\|\|\hat{\phi_j} - \phi_j\|\|$	-2.0
Ausrichtung	$\|\|[0,0,-1]^T - e_g\|\|^2$	-5.0
Gelenkgeschwindigkeit	$\|\|\dot{\phi}\|\|^2$	$-0.0003k_c$
Gelenkbeschleunigung	$\|\|\ddot{\phi}\|\|^2$	$-0.00001k_c$
Drehmoment	$\|\|\tau\|\|^2$	$-0.0002k_c$
Aktionsrate	$\|\|a_{t-1} - a_t\|\|^2$	$-0.01k_c$

Tabelle 3.2: Belohnungsfunktion für die Aufgabe des Hinsetzens

3.3 Evaluation

Das Erlernen dieses Bewegungsablaufes war erfolgreich. Es ist jedoch zu nennen, dass die Ausrichtung des Roboters zu Beginn einer Episode nicht gänzlich randomisiert werden konnte, weil das Simulationsprogramm einen Fehler warf. Deshalb wurde der Roboter zu Beginn einer Episode rückwärts ausgerichtet. Rückwärts heißt, dass der Rücken des Roboters zum Boden hin ausgerichtet wurde. Eine gewisse Randomisierung der Ausrichtung fand dennoch statt, weil die Randomisierung der Gelenke auch die Ausrichtung des Roboters beeinflusst hat. Evaluiert wird, wie viel Gewinn über 2000 Episoden erzielt wurde. Verglichen werden die in Kapitel 2.3 eingeführten Aktionsräume. **Zielwinkel**: Die Aktionen wurden wie folgt berechnet: $\phi_d = ko_t + \phi_t$. In dieser Bachelorarbeit wurde der Skalierparameter $k = 7$, für die Aufgabe des Hinsetzens, und $k = 10$ für die weiteren Aufgaben gewählt. Dies begründet sich dadurch, dass in [NVIDIA 2023b] für den Anymal Roboter [Hutter u. a. 2016], der größer ist als der Unitree A1, der Skalierparameter $k = 13.5$ gewählt wurde. Dieser Parameter hat sich bei den Experimenten dieser Bachelorarbeit als zu stark herausgestellt, weshalb empirisch je nach Aufgabe ein leicht abgesenkter Wert gewählt wurde. **Torque Control**: In [Chen u. a. 2022] wurde dieser Aktionsraum genauer untersucht. Diese Arbeit verwendet den gleichen Algorithmus, die gleiche Architektur für das neuronale Netz, das gleiche Simulationsprogramm und den gleichen Roboter wie in dieser Bachelorarbeit. Die ausgegebenen Aktionen wurden mit dem Skalar $k = 10$ multipliziert, ehe sie an den Roboter gesendet wurden. Dieser Wert hat sich bei den Experimenten als zu gering herausgestellt. In dieser Bachelorarbeit wird der Skalierparameter $k = 10000$ gewählt. Bei dieser Aufgabe hat sich die direkte Drehmomentregelung als erfolgreicher erwiesen. Zu sehen ist dies in Abbildung 3.1a. In dieser ist aber auch zu sehen, dass die Performanz der direkten Drehmomentregelung in der ca. 800. Episode einbricht. Die Performanz des PD-Reglers steigt langsamer,

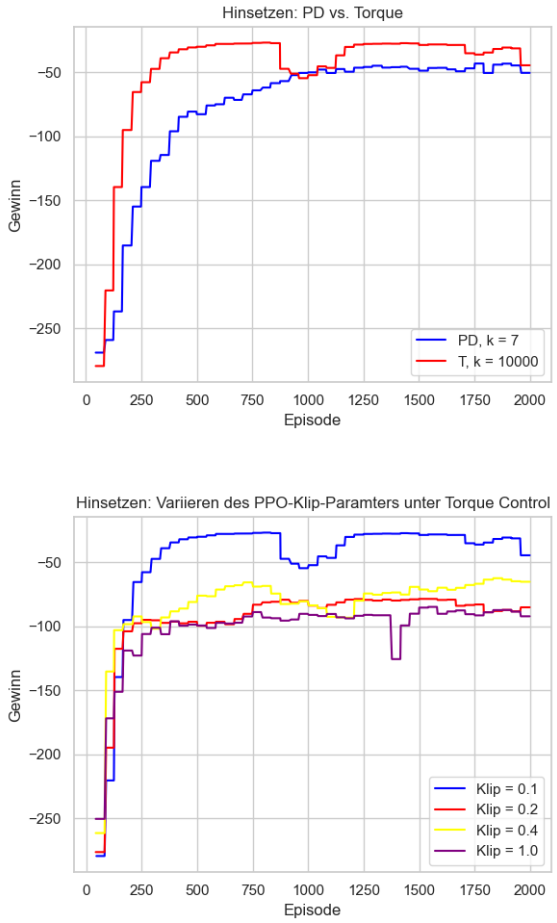


Abbildung 3.1: Die direkte Drehmomentregelung wird dem PD-Regler gegenübergestellt. Der PPO-Klip Parameter wird unter verschiedenen Werten gemessen

dafür aber stabiler. Es lässt sich die These aufstellen, dass der PD-Regler bei längerer Trainingszeit die direkte Drehmomentregelung übertreffen könnte. Bezüglich der Variation des PPO-Klip-Parameters ist zu sehen, dass die Wahl KLIP = 0.1 am besten abschneidet. Hervorzuheben ist die ca. 200. Episode. Alle anderen Variationen des Klip-Parameters können die Performanz nur geringfügig verbessern, während die Wahl KLIP = 0.1 die Performanz erheblich verbessert. Die anderen Variationen erreichen dieses Maß an Performanz in dem betrachteten Zeitabschnitt überhaupt nicht und nähern sich diesem auch nicht an. Es ist zu sehen, dass ein einziger schlechter Aktualisierungsschritt die Policy vom Kurs abbringen kann.

4 Bewegungsablauf: Aufstehen

Dieses Kapitel erklärt das zweite Teilsystem, dessen Funktion das Aufstehen des Roboters ist. Der Startzustand ist die Sitzkonfiguration, welche in Kapitel 3 näher erläutert wird. Die Gelenkkonfiguration wird randomisiert, um den Agenten auf Zustände vorzubereiten, die von der eigentlichen Startkonfiguration abweichen. Die Randomisierung geschieht durch die Multiplikation des Standardwertes des Gelenkes mit einem randomisiert ausgewählten Skalierfaktor im Intervall $[0.9; 1.1]$. Die Gelenkgeschwindigkeiten (m/s) werden randomisiert aus dem Intervall $[-0.4; 0.4]$ gewählt. Der Endzustand ist eine Gelenkkonfiguration, in der der Roboter eine Standposition annimmt. Aus dieser Standposition soll der Roboter in der Lage sein zu laufen, siehe Kapitel 5.

4.1 Vorhandene Verfahren

Wie für die Aufgabe des Hinsetzens bzw. der „Recovery“ gibt es für die Aufgabe des Aufstehens einen manuellen Lösungsansatz. In [Stückler u. a. 2006] wird gezeigt, wie es möglich ist, einen bipedalen Roboter aufstehen zu lassen. Diese Prozedur des Aufstehens wird in 4 Phasen aufgeteilt. **Phase 1**: Bewegung des Oberkörpers in eine aufrechte Sitzposition und Aufstützen der Arme hinter dem Rücken. Diese Phase kommt der Hinsetz/Recovery Prozedur gleich. **Phase 2**: Stütze die Arme so, dass eine Brückenposition angenommen wird. **Phase 3**: Bewegen des Massezentrums über die Füße, indem der Oberkörper nach vorne geschwungen wird. **Phase 4**: Richte den Körper gerade auf. Wie in Kapitel 3.1 müssen von einem Menschen manuell Zustände (der Anfang/das Ende einer Phase) definiert werden. Für jeden Zustandsübergang (Phase) muss eine Sequenz von Aktionen definiert werden. In Abbildung 3 aus [Stückler u. a. 2006] wird gezeigt, wie die Zielwinkel für jeden Zeitschritt explizit angegeben werden müssen für einen erfolgreichen Ablauf der Prozedur. Das wird mit steigender Anzahl von Zuständen immer aufwendiger.

4.2 Aufstehen mittels Reinforcement Learning

Die Aufstehprozedur ist für einen quadrupedalen Roboter prinzipiell nicht so komplex, dass diese mittels Reinforcement Learning entwickelt werden müsste. Dennoch ist es lohnenswert, das zu tun. **Testplattform**: Der quadrupedale Roboter kann als Testplattform dienen. Beispielsweise könnte ein Aspekt der Belohnungsfunktion für das Erlernen des Aufstehens eines quadrupedalen Roboters auch für andere Plattformen, wie einem humanoiden Roboter, verwendet werden. **Maßstab**: Die Aufgabe des Aufstehens lässt sich als Maßstab für das Testen eines Reinforcement Learning Algorithmus verwenden, wie in [Schulman u. a. 2015b].

Zustandsdarstellung
Höhe_t
$\text{Lineargeschwindigkeit}_t$
$\text{Rotationsgeschwindigkeit}_t$
$\text{Projizierte Schwerkraft}_t$
$\text{Gelenkposition}_{t,t-1}$
$\text{Gelenkgeschwindigkeit}_{t,t-1}$
Aktion_{t-1}

Tabelle 4.1: Zustandsdarstellung für die Aufgabe des Aufstehens

Belohnungsfunktion		
Metrik	Term	Gewicht
Abweichung	$\|\|\hat{\phi}_j - \phi_j\|\|$	-1.0
Ausrichtung	$\|\|[0,0,-1]^T - e_g\|\|^2$	-5.0
Gelenkgeschwindigkeit	$\|\|\dot{\phi}\|\|^2$	$-0.0003k_c$
Gelenkbeschleunigung	$\|\|\ddot{\phi}\|\|^2$	$-0.00001k_c$
Drehmoment	$\|\|\tau\|\|^2$	$-0.0002k_c$
Aktionsrate	$\|\|a_{t-1} - a_t\|\|^2$	$-0.01k_c$

Tabelle 4.2: Belohnungsfunktion für die Aufgabe des Aufstehens

4.2.1 Zustandsdarstellung

Die Zustandsdarstellung 4.1 ist die gleiche wie für die Aufgabe des Hinsetzens in Kapitel 3, mit dem Unterschied, dass die Zielkonfiguration eine andere ist und, dass die Höhe des Massezentrums des Roboters inkludiert wird, weil diese Größe das Aufstehen maßgeblich charakterisiert.

4.2.2 Belohnungsfunktion

Tabelle 4.2 zeigt die Belohnungsfunktion für das Aufstehen, welche nahezu identisch mit der des Hinsetzens ist. Das Gewicht des Terms *Abweichung* wurde auf -1.0 abgesenkt, weil sich gezeigt hat, dass der Roboter bei zu hoher Bestrafung dieses Terms gegen eine falsche Haltung konvergiert. Die Hypothese des Bacheloranden ist, dass der Strafterm der Abweichungsfunktion nicht monoton steigend ist. Der Roboter muss der These nach zeitweise höhere Kosten (höhere negative Zahlen) erfahren, um abschließend in der Position zu sein, die am wenigsten Kosten (globales Maximum) verursacht. Der Roboter entscheidet sich demnach in einem lokalen Maximum zu verweilen, wenn die Bestrafung für die Abweichung zu stark ist.

4.2.2.1 Höhe des Roboters

Es wäre auch möglich gewesen, die Höhe in die Belohnungsfunktion miteinzubeziehen, d.h. den Agenten in Abhängigkeit davon, ob er sich einer Zielhöhe nähert oder entfernt zu bestrafen. Diese Idee wurde zu Beginn berücksichtigt. Der Roboter wurde mit der Zielgelenkkonfiguration in der Simulation erzeugt und die korrespondierende

Höhe wurde abgelesen, notiert und in die Belohnungsfunktion integriert. Die Problematik, die sich ergeben hat, ist, dass der Roboter die Höhe erreicht hat, jedoch auf Kosten anderer Ziele wie der Orientierung oder der Gelenkkonfiguration.

4.2.2.2 Kontaktsensoren

Der Unitree A1 hat an seinen Füßen Kontaktsensoren. Die Messungen dieser Sensoren könnten in die Belohnungsfunktion integriert werden und es könnte eine Episode beendet werden, sobald der Roboter einen Fuß vom Boden abnimmt. Diese Vorgehensweise hat den Nachteil, dass die Trainingsgeschwindigkeit verlangsamt wird. In dieser Bachelorarbeit wurde das Programmiergerüst [NVIDIA 2023b] verwendet, welches erlaubt bis zu 4096 Roboter parallel trainieren zu lassen. Das Erzeugen von Fußsensoren für jeden dieser Roboter hat es speichertechnisch nicht mehr ermöglicht 4096 Roboter zu erzeugen, sondern nur noch 1024 Roboter. Das Berechnen der Sensorwerte für jeden einzelnen der 1024 Roboter beansprucht zudem Rechenzeit. Das Training resultierte auch ohne Verwendung von Fußkontaktsensoren in einer Policy, die den Roboter aufstehen lässt, ohne dass er einen Fuß abnimmt.

4.3 Evaluation

Das Erlernen dieses Bewegungsablaufes war erfolgreich. Die Konfiguration für die Aktionsräume ist Kapitel 3.3 zu entnehmen. Zusätzlich wurde der Skalierfaktor der Aktion k variiert. Auffällig ist, dass hohe Skalierfaktoren schneller höhere Belohnungen erzielen. Das lässt sich damit begründen, dass die angewendete Aktion durch die höhere Skalierung stärker ist. Stärker heißt bezüglich direkter Drehmomentregelung, dass höhere Drehmomente angewendet werden und bezüglich des PD-Reglers, dass größere Zielwinkel befehligt werden. Qualitativ hat sich aber gezeigt, dass die Stabilität der ausgeführten Bewegung unter hohen Skalierfaktoren leidet. Das drückt sich dadurch aus, dass der Roboter zittert und weiter Aktionen ausführt, obwohl er eigentlich bereits die Zielgelenkkonfiguration erreicht hat. Zu niedrige Skalierfaktoren (in der dritten Abbildung mit $k = 100$) haben sich beim Aufstehen als besonders problematisch bei der direkten Drehmomentregelung erwiesen, weil der Roboter nicht genügend Kraft aufwenden konnte, um sich vom Boden abzustützen.

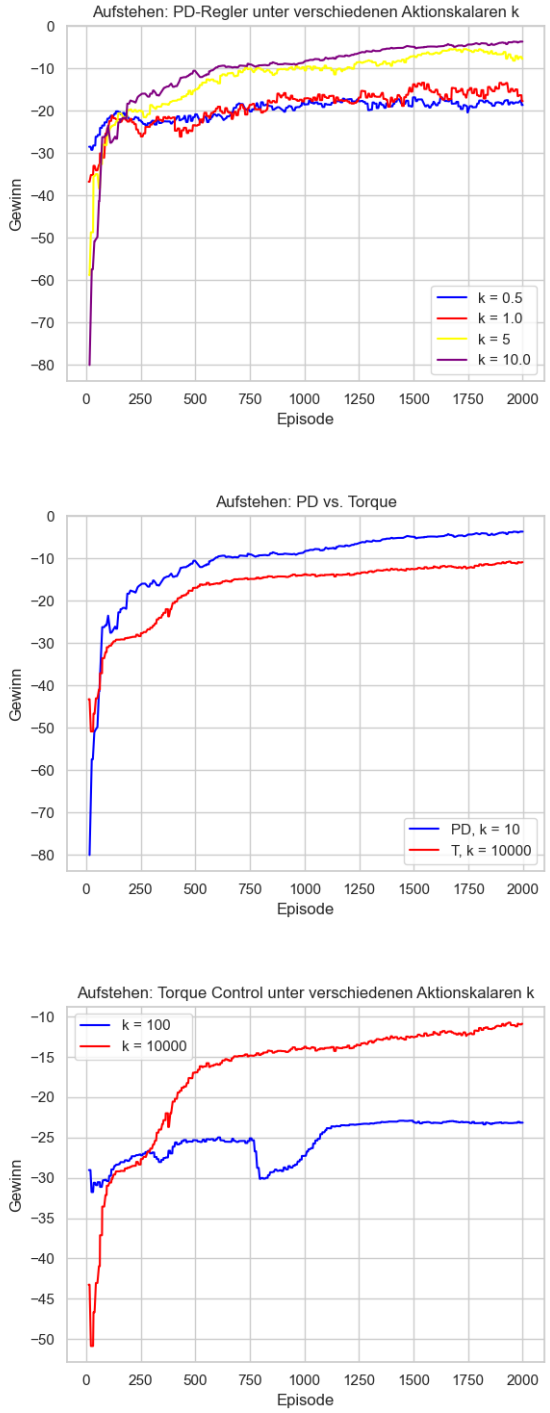


Abbildung 4.1: Der Skalierfaktor k des PD-Reglers und der direkten Drehmomentregelung wird unter verschiedenen Variationen ausgewertet. Der PD-Regler wird der direkten Drehmomentregelung gegenübergestellt

5 Bewegungsablauf: Laufen

Dieses Kapitel erklärt das dritte Teilsystem, dessen Ziel das Erlernen des Laufens ist. Zunächst wird ein konventionelles System vorgestellt. Danach wird als Vergleich das durch Reinforcement Learning entstandene System vorgestellt.

5.1 Vorhandene Verfahren

Im Rahmen des ALDURO-Projekts (**A**nthropomorphically **L**egged and Wheeled **Du**isburg **Ro**bot Alduro) [Germann u. a. 2005] soll die Aufgabe des Laufens eines vierbeinigen Baggers automatisiert werden. Mit 16-Freiheitsgeraden wäre es für einen Menschen sehr schwierig den Roboter/Bagger so zu kontrollieren, dass er eine sinnvolle und stabile Bewegung durchführt. Abhilfe soll ein System verschaffen, dass Nutzerbefehle auf valide Aktionen abbildet, welche anschließend an die Aktuatoren gesendet werden. Ein Teil dieses Systems, der *Motion Generator*, wird in diesem Kapitel vorgestellt.

5.1.1 FreeGait

Die konventionelle Methode, die vorgestellt wird, nennt sich *FreeGait Pattern Generator* und stammt aus [Germann u. a. 2005]. Die Aktion ist bestimmt als ein Laufschritt oder als eine Rumpfbewegung. Eine Frage ist, welches Bein als Nächstes angehoben wird. *FreeGait* wählt das nächste Bein, basierend darauf, welches am wahrscheinlichsten in einer ausgestreckten Position ist. Falls das ausgewählte Bein nicht erhöht werden kann, weil dies zu einer unstabilen Lage des Roboters führen würde, so wird vorher eine „korrektive Bewegung“ auf den Rumpf ausgeübt. Wie weit ein Bein ausgestreckt ist, wird für jedes Bein festgehalten in der Variablen *Proximity* und wird bestimmt in Abhängigkeit davon, wie nah die vier Hydraulik-Zylinder-Längen $x_i,\ \text{mit}\ i = 1..4$, eines Beines an ihrem „end-of-travel (eot)“ sind:

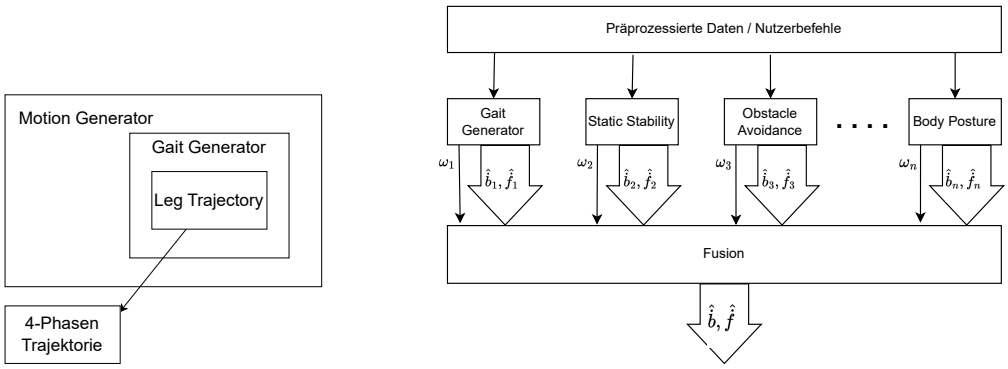
$$eot_i = -\ln(x_{min,i} - x_i) - \ln(x_{max,i} - x_i) \tag{5.1}$$

Zusätzlich ist von Interesse, wie schnell sich die Hydraulik-Zylinder-Längen ihrem „end-of-travel (eot)“ nähern. Dies wird ausgedrückt durch $v_{eot,i}$:

$$\delta_{eot,i} = \frac{d}{dx_i} eot_i = \frac{1}{x_{min,i} - x_i} + \frac{1}{x_{max,i} - x_i} \tag{5.2}$$

$$v_{eot,i} = \frac{d}{dt}(eot_i) = \frac{d}{dx_i}(eot_i)\frac{dx_i}{dt} = \delta_{eot,i}\frac{dx_i}{dt} = \delta_{eot,i} v_{x,i} \tag{5.3}$$

Im Algorithmus wird zur Definition der Variablen *Proximity* der Maximalwert $\delta_{eot,i} * sign(v_{eot,i})$ über alle Hydraulik-Zylinder betrachtet. Es wird gesagt, dass der *FreeGait*



(a) Die Bein Trajektorie wird über 4 Phasen geregelt

(b) Die von den für die Bewegung zuständigen Module befehltigen Körper und Fußgeschwindigkeiten werden in einer einzigen Ausgabe zusammengefasst

Abbildung 5.1: Abbildung des Motion Generator Moduls. Vereinfacht nachgezeichnet, entnommen aus [Germann u. a. 2005].

Algorithmus anfällig für Deadlocks ist und bessere Ergebnisse durch eine Sonderregel erzielt werden können.

5.1.1.1 Bein Trajektorie

Die Trajektorie der Beine basiert in [Germann u. a. 2005] auf einer „four-phase finite-state-machine“. Das Platzieren eines Beines wird in vier Phasen unterteilt. **Lift**: Nachdem ein Bein ausgewählt wurde, wird das Bein bis auf eine gemerkte Schritthöhe angehoben. **Swing**: Sobald der Kontaktsensor das Ablößen des Fußes vom Boden detektiert, setzt die Swing-Phase ein. Diese erhöht die Geschwindigkeit des Beines entsprechend der gewünschten Richtung. Je weiter das Bein an seine „kinematic margin“ gelangt - oder wenn die Bewegung durch den *Pattern Generator* gestoppt wird - desto mehr wird die Bewegung entschleunigt. **Lower**: Nach der Swing-Phase setzt die Lower-Phase ein. Diese Phase bewegt das Bein langsam nach unten bis eine geschätzte Bodenhöhe angenommen wird. Diese geschätzte Bodenhöhe basiert auf einer gemerkten Höhe des vorherigen Schrittes. Das Absetzen wird durchgeführt bis Boden detektiert wird. Falls nicht wird das Manöver abgebrochen. **Stance**: Nachdem Boden detektiert wurde, setzt die „Stance“-Phase ein, die den Roboter stehen lässt. In dieser Phase wird eine niedrige konstante Geschwindigkeit nach unten ausgeübt, um die Füße am Boden zu halten. Damit der Roboter sich nicht nach oben bewegt, übt das *Body Posture Modul* eine korrespondierende Geschwindigkeit nach unten auf den Rumpf aus. Die soeben aufgelisteten Phasen sind Teil des *Foot Trajectory Generator Moduls*. Parallel zu diesem existiert ein weiteres Modul, das *Obstacle Avoidance Modul*. Dieses Modul stellt einen Reflex bereit: Wenn ein Bein ein Hindernis während seiner Trajektorie trifft, so hat es in Abhängigkeit der Höhe des Hindernisses und der Position des Beines die Möglichkeit den Fuß zurückzuziehen oder das Hindernis zu durchqueren. Wenn ein Fuß abgesenkt wird, ohne den Boden zu finden, so sorgt dieses Modul dafür, dass der Fuß zurückgezogen wird. „Static Stability“ wird da-

durch erreicht, das Massezentrum des Roboters innerhalb eines „Support Polygons“ zu halten. Diese Aufgabe wird an das *Static Stability Modul* delegiert.

5.1.1.2 Zusammensetzen der Module

Die Bewegung gliedert sich in verschiedene, voneinander unabhängige Module. Diese Module können untereinander nicht kommunizieren. Sie erhalten als Eingabe präprozessierte Sensor-Daten und einen Befehl des Nutzers. In Abhängigkeit dieser Eingabe ist die Ausgabe jedes Moduls m eine Zielbewegung, welche sich in die Geschwindigkeiten des Körpers $\hat{\dot{b}}_m$ und in die Geschwindigkeit der vier Füße $\hat{\dot{f}}_m$ gliedert. Die Ausgaben der einzelnen Module werden im *Fusions Modul* in einer einzigen Ausgabe bzw. in einem einzigen Befehl durch eine gewichtete Addition der Befehlsvektoren der einzelnen Module zusammengefasst:

$$\left[\hat{\dot{b}}^T, \hat{\dot{f}}^T\right]^T = \frac{1}{n} \sum_{m=1}^{n} w_m \left[\hat{\dot{b}}_m^T, \hat{\dot{f}}_m^T\right]^T \tag{5.4}$$

wobei n die Anzahl der Module ist. Da die Module vollständig unabhängig voneinander sind, erfolgt die Koordinierung durch das Anpassen der Gewichte. Diese sind so anzupassen, dass beispielsweise das *Static Stability Modul* andere Module im Fall eines kritischen Zustandes überstimmen kann.

5.1.2 Nachteile

Zusätzlich zu den in Kapitel 3 und Kapitel 4 aufgeführten Nachteilen, sind weitere Probleme spezifisch für das oben vorgestellte Verfahren zu nennen. In Kapitel 5.1 wurde bereits die Anfälligkeit für einen Deadlock bezüglich der Wahl des nächsten Beines erwähnt. Der Vorschlag, eine Sonderregel zu verwenden, um ein besseres Ergebnis zu erzielen, ist fehleranfällig, weil Sonderregeln im Allgemeinen, gerade in größeren Systemen, mit anderen Funktionen in Konflikt stehen können, auch wenn das hier nicht der Fall sein mag. Als besonders kritisch hervorzuheben ist, dass das Einsetzen des kritischen *Static Stability Moduls* durch manuelles Anpassen der Gewichte reguliert wird. Mit zunehmender Anzahl von Modulen wird es schwieriger die Gewichtsparameter so einzustellen, dass das *Static Stability Moduls* immer genau dann einsetzt, wenn es benötigt wird. Zusätzlich stellt sich die Frage, wie das überprüft werden sollte. Das beeinflusst die Sicherheitsgarantie dieses Systems.

5.2 Laufen mittels Reinforcement Learning

Die verwendete Reinforcement Learning Methode richtet sich bezüglich der Zustandsdarstellung und Belohnungsfunktion maßgeblich nach [Hwangbo u. a. 2019] und [Aractingi u. a. 2023]. Die Ergebnisse aus [Hwangbo u. a. 2019] fordern konventionelle Methoden zum Steuern eines quadrupedalen Roboters heraus:
„Controllers presented in this paper, trained in a few hours in simulation, outperformed the best existing model-based controller running on the same robot, which were designed and tuned over many years. Our learned locomotion policies ran faster and

Zustandsdarstellung
Höhe_t
$\text{Lineargeschwindigkeit}_t$
$\text{Rotationsgeschwindigkeit}_t$
$\text{Geschwindigkeitsbefehl}_t$
$\text{Projizierte-Schwerkraft}_t$
Gelenkposition_t
$\text{Gelenkgeschwindigkeit}_t$
$\text{Gelenkkonfiguration}_{t,t-1}$
$\text{Gelenkgeschwindigkeit}_{t,t-1}$
Aktion_{t-1}

Tabelle 5.1: Zustandsdarstellung für die Aufgabe des Laufens

with higher precision while using less energy, torque, and computation.“
Dass das Erlernen des Laufens mittels Reinforcement Learning möglich ist, geht aus [Schulman u. a. 2017], [Schulman u. a. 2015a], und [Schulman u. a. 2015b] hervor, jedoch sind diese Ergebnisse auf die Simulation beschränkt.
Der Startzustand des Roboters ist die Standkonfiguration. Eine Episode terminiert, wenn das Zeitlimit der Episode überschritten wurde, oder wenn die Höhe des Roboters eine gewisse Schwelle unterschritten hat. Dann wird angenommen, dass der Roboter gestürzt ist. Bemerkenswert ist, dass das Erlernen des Laufens mittels des Deep Reinforcement Learnings weitestgehend plattformunabhängig ist. Der Entwurf der Belohnungsfunktion richtet sich nach [Hwangbo u. a. 2019] und [Aractingi u. a. 2023]. Die Belohnungsfunktionen dieser Publikationen wurden für die quadrupedalen Roboter ANYmal [Hutter u. a. 2016] und MIT Cheetah [Bledt u. a. 2018] entwickelt und funktionieren auch für den Unitree A1.

5.2.0.1 Zustandsdarstellung

Ein quadrupedaler Roboter wird durch einen Steuerhebel gesteuert. Durch Bedienen des Hebels kann dem Roboter eine gewisse Geschwindigkeit V_{cmd} befehligt werden. Mittels dieses Befehls wird der Roboter auch trainiert. Ein Vektor wird in die Zustandsdarstellung abgebildet, welcher die befehligte lineare Geschwindigkeit in die x- und y-Richtung $v^*_{linVel,xy}$ und die befehligte Drehrate $\omega^*_{angVel,z}$ entlang der z-Achse festhält. Die linearen Geschwindigkeiten und die Drehrate werden randomisiert zu Beginn einer jeden Episode respektiv aus den Intervallen $[-2.0,\ 2.0]$ (m/s), $[-1.0,\ 1.0]$ (m/s) und $[-3.14,\ 3.14]$ (rad/s) entnommen. Aus [Hwangbo u. a. 2019] geht hervor: „The joint state history was essential in training a locomotion policy.“. Die Autoren stellen die Hypothese auf, dass dies der Fall sein könnte, weil das Abbilden der vorherigen Gelenkkonfigurationen in die Zustandsdarstellung Kontaktdeketion ermöglicht. Das begründet die Entscheidung, die vorherige Gelenkkonfiguration und die vorherige Gelenkgeschwindigkeit in die Zustandsdarstellung abzubilden.

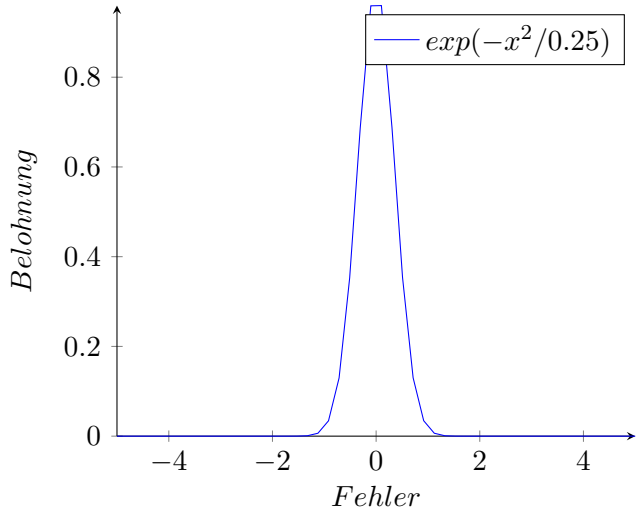
5.2.0.2 Belohnungsfunktion

Tabelle 5.2 gibt die Belohnungsfunktion für das Laufen an. Zentral sind die Teilfunktionen „Lineargeschwindigkeit x-y“ und „Rotationsgeschwindigkeit z“. Diese messen

Belohnungsfunktion		
Metrik	Term	Gewicht
Lineargeschwindigkeit x-y	$\exp(-\|\|v^*_{linVel,xy} - v_{base,xy}\|\|^2/0.25)$	2.0
Rotationsgeschwindigkeit z	$\exp(-(\omega^*_{angVel,z} - \omega_{angVel,z})^2/0.25)$	0.5
Lineargeschwindigkeit z	$-v^2_{linVel,z}$	$-1.2k_c$
Rotationsgeschwindigkeit x-y	$-\|\|\omega_{angVel,xy}\|\|^2$	$-0.05k_c$
Foot Clearance	$(p^{\max}_{f,z} - p_{f,i,z})^2 \|\|V_{f,xy,i}\|\|$	$-5.0k_c$
Abweichung	$\|\|\hat{\phi}_j - \phi_j\|\|$	$-0.5k_c$
Gelenkgeschwindigkeit	$\|\|\dot{\phi}\|\|^2$	$-0.0003k_c$
Gelenkbeschleunigung	$\|\|\ddot{\phi}\|\|^2$	$-0.00001k_c$
Drehmoment	$\|\|\tau\|\|^2$	$-0.0002k_c$
Aktionsrate	$\|\|a_{t-1} - a_t\|\|^2$	$-0.01k_c$

Tabelle 5.2: Belohnungsfunktion für die Aufgabe des Laufens

die Differenzen zwischen den derzeitigen Geschwindigkeiten des Roboters und den befohlenen. Ein naiver Ansatz ist es, die euklidische Norm als Distanzmaß zu verwenden. Laut [Hwangbo u. a. 2019] ist die Gefahr hier, dass der Agent die Episode beendet, indem er absichtlich hinfällt. Das tut er, weil er gerade zu Beginn des Trainings schlechtes Verhalten aufweist und so versucht den hohen Strafen, die er sonst beim Fortführen der Episode bekommen würde, zu entgehen. Stattdessen wird eine glockenartige Funktion empfohlen. Diese ist auf ihrem gesamten Definitionsbereich größer oder gleich null (keine Strafen) und gibt in Abhängigkeit des Fehlers eine nach oben beschränkte Belohnung aus:



Die Terme *Lineargeschwindigkeit z* und *Rotationsgeschwindigkeit x-y* sind Terme, die ungewünschte Bewegungen bestrafen. Der Rumpf des Roboters soll möglichst starr sein, sich also nicht linear entlang der z-Achse bewegen. Auch unerwünscht ist, dass er sich entlang der Roll- und Nick-Achse bewegt. Lediglich die Rotation um die Gierachse ist entsprechend des erhaltenen Befehls erlaubt. Die Gangart charakterisiert sich unter anderem durch die Höhe des Fußes zum Ende der Schwungphase. Es lässt sich argumentieren, dass ein durch grobes Terrain (hohes Gras, Gebüsche) laufender Roboter eine höhere Fußhöhe benötigt als einer, der auf glattem Boden läuft. Dies lässt

sich mittels des *Foot Clearance*-Belohnungsterms regeln. $p_{f,z}^{\max}$ gibt den Soll-Wert der Fußhöhe zum Ende der Schwung-Phase an und $p_{f,i,z}$ den Ist-Wert der Fußhöhe. Dieser Strafterm ist automatisch 0 wenn die Fußhöhe gleich der als maximal definierten ist. $||V_{f,xy,i}||$ gibt die absolute Geschwindigkeit eines Fußes entlang der x/y-Achse an. Die Füße sollen ihre maximale Geschwindigkeit zum Ende der Schwungphase erreichen. So erhält der Roboter eine Strafe, wenn er bei niedrigen Fußhöhen bereits eine hohe Fußgeschwindigkeit annimmt. Der Term *Abweichung* bestraft Abweichungen von der Standkonfiguration. Die für das Laufen notwendigen Gelenkkonfigurationen befinden sich in einem Bereich um die Standposition. In Abhängigkeit des Gewichts dieses Strafterms wird sich der Roboter mehr oder weniger weit von seiner Standkonfiguration entfernen [Aractingi u. a. 2023]. Dies kann die Schrittweite beeinflussen. Der Term kann allerdings dafür sorgen, dass der Roboter eine zu nah an der Standkonfiguration liegende Bewegung ausführt und somit nicht beweglich genug ist. Das Problem wird im Ausblick in Kapitel 7 näher betrachtet.

5.3 Evaluation

Das Erlernen dieses Bewegungsablaufes war erfolgreich. Wird die Aktion als einen vom Roboter anzunehmenden Zielwinkel-Vektor gewählt, so ergibt sich die Konstruktion der Aktion wie folgt: $\phi_d = ko_t + \phi_n$, wobei ϕ_n, anders als in Kapitel 4, nicht die derzeitige Gelenkkonfiguration, sondern die Standkonfiguration bzw. Nominalkonfiguration ist. Somit lassen sich die Zielwinkel in einem lokalen Umfeld um die Standkonfiguration des Roboters angeben. Das konzentriert den Suchraum über alle gültigen Aktionen direkt auf einen für das Laufen sinnvollen Bereich. Bei der direkten Drehmomentregelung hat sich gezeigt, dass es wichtig ist, die ausgegebenen Drehmomente zu skalieren, weil der Roboter sonst aufgrund von zu wenig Kraft fiel, was die Episode beendet.
Der erzielte Gewinn in einer Episode ist nicht die einzige Metrik, die zu betrachten ist. Ebenfalls relevant ist, wie robust der Roboter unter einer Policy ist. In Abbildung 6 aus [Chen u. a. 2022] wird das Ergebnis eines Experimentes vorgestellt, in welchem der Roboter getreten und anschließend dessen Höhe gemessen wird. Das Ergebnis lautet: „This kicking experiment demonstrates that the torque-based RL policy shows more robustness than the position-based RL policy“. Die Bachelorarbeit prüft die Robustheit, indem eine auf PD-Regler basierende und eine auf direkter Drehmomentregelung basierende Policy 2000 Episoden trainiert wurde. Anschließend wurde die trainierte Policy zehn Episoden in der Simulation von 32 Umgebungen/Robotern ausgeführt. Während der Ausführung und während des Trainings wurde der Roboter zufälligen Kräften ausgesetzt. Das geschieht, indem dem Roboter entlang allen Achsen zufällige Geschwindigkeiten im Intervall $[-0.5, 0.5]^3$ m/s aufgesetzt wurden. Insgesamt ist der Roboter, der per PD-Regler trainiert wurde 143 mal gefallen und der Roboter, der per direkter Drehmomentregelung trainiert wurde nur 101 mal. Das Experiment scheint die Erkenntnis aus [Chen u. a. 2022] zu bestätigen, aber es ist wichtig zu nennen, dass der PD-geregelte Roboter, weil er innerhalb von 2000 Episoden höhere Belohnungen erhalten hat, schneller läuft als der über direkter Drehmomentregelung trainierte Roboter. Schnelleres Laufen erhöht die Anfälligkeit für einen Sturz.

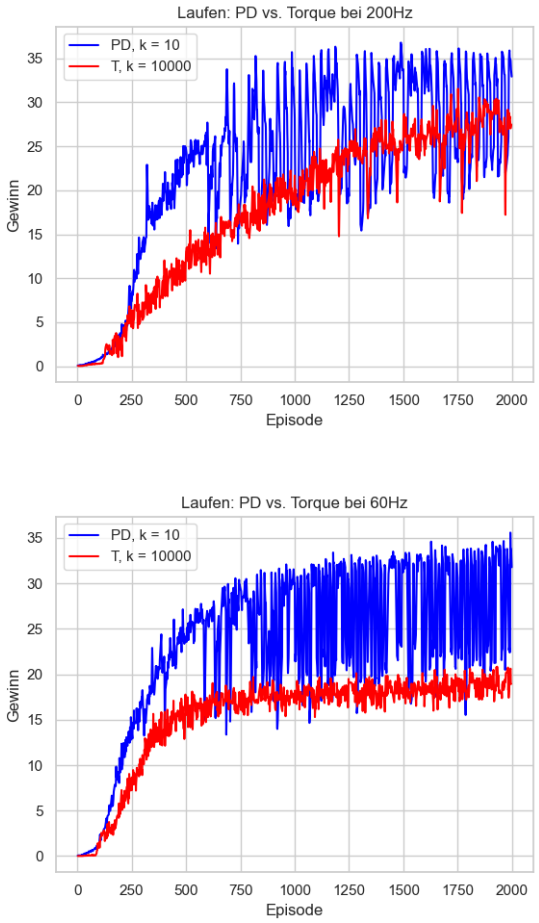


Abbildung 5.2: Der PD-Regler wird der direkten Drehmomentregelung unter 60Hz und 200Hz Policy-Abtastrate gegenübergestellt

Sowohl [Chen u. a. 2022] als auch [Peng u. Van De Panne 2017] schreiben, dass die direkte Drehmomentregelung höhere Abtastraten bezüglich der Policy erfordert. Zu diesem Zweck wird in dieser Bachelorarbeit das Laufen mit 200Hz und mit 60Hz trainiert. Abtastrate meint, wie oft das neuronale Netzwerk der Policy abgetastet wurde. Um das zu erreichen, wurde die Frequenz der Simulation insgesamt auf die Frequenz der gewünschten Policy-Abtastrate eingestellt. Beim Training erzielte der PD-Regler einen maximalen Gewinn von 35.58 (60Hz) und 36.78 (200Hz). Die direkte Drehmomentregelung erzielte einen maximalen Gewinn von 20.83 (60Hz) und 31.53 (200Hz). Das heißt, dass der PD-Regler unter Absenkung der Abtastrate 96.74% der maximalen Performanz erhalten hat, während es bei der direkten Drehmomentregelung lediglich 66.06% sind.

6 Zusammenführung der Bewegungsabläufe in einem Controller

Die Bewegungsabläufe Hinsetzen, Aufstehen und Laufen werden zusammengeführt. Das Konzept stammt aus [Lee u. a. 2019]. Das Zusammenführen einzelner erlernter Reinforcement Learning Policies nennt sich Hierarchical Reinforcement Learning. Die Kombination der Teilsysteme erfolgt entweder durch einen endlichen Automaten oder durch ein neuronales Netz. Anschließend wird erklärt, was zu beachten ist, wenn ein in der Simulation erlerntes System in die reale Welt transferiert werden soll.

6.1 Implementierung mit endlichem Automaten

Abbildung 6.1 stellt die Kombination der Einzelsysteme mittels eines endlichen Automaten dar. Die Zeichnung und Implementierung richtet sich nach [Lee u. a. 2019], Abschnitt G. Der Roboter leitet die Aufstehprozedur ein, nachdem der Winkel zwischen der z-Achse bezüglich des Roboters und der Basis z-Achse kleiner als 0.1° ist. Wenn der Roboter eine Höhe von über 0.2 Metern angenommen hat, wird der Lauf-Controller aktiviert. Dieser bleibt so lange aktiv, bis geschätzt wird, dass der Roboter in einem Zustand ist, in dem das Laufen nicht mehr sicher gewährleistet werden kann, z.B. wenn er fällt oder zu schief ist. Der Übergang vom Laufen zum Hinsetzen wird aktiviert, wenn der Winkel zwischen der z-Achse bezüglich des Roboters und der Basis z-Achse größer als 0.55° oder die Höhe kleiner als 0.2m ist. Die Wahl der Werte folgt keinem besonderen Schema, sondern basiert auf Trial-and-Error.

6.2 Implementierung mit neuronalem Netz

Die Regelung der Bewegungen kann von einem neuronalen Netz übernommen werden, welches ebenfalls mit einem Reinforcement Learning Algorithmus trainiert und entsprechend als Policy oder Behaviour Selector bezeichnet wird. Dieser Ansatz stammt aus [Lee u. a. 2019], aber wurde in dieser Bachelorarbeit nicht implementiert. Abbildung 6.2 stellt schematisch das Prinzip dar. Der Zustandsraum des Behaviour Selectors besteht aus der Vereinigung der Zustandsmengen der Bewegungsabläufe und dem Index der im letzten Zeitschritt gewählten Bewegung (letzte Aktion). Die Ausgabe des Behaviour Selectors ist ein Vektor $\{p_0, p_1, p_2\}$. Jeder Eintrag gibt die Wahrscheinlichkeit an, in einem Zustand s ein Teilmodul $a \in \{0, 1, 2\}$ auszuführen. Bei der Aktionsselektion wird diese Wahrscheinlichkeitsverteilung gesamplet. Die für das Training verwendete Belohnungsfunktion ist nahezu die gleiche wie für das Laufen. Der Implementierung mit neuronalem Netz könnte entgegnet werden, dass die

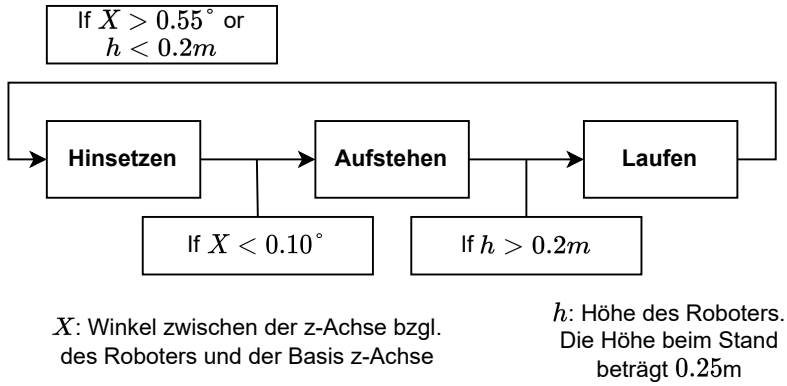


Abbildung 6.1: Recovery-Controller als endlicher Automat. Konzept und Abbildung modifiziert nachgezeichnet bzw. entnommen aus [Lee u. a. 2019]

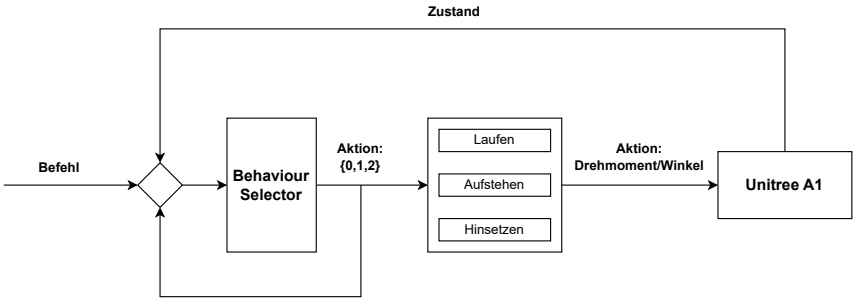


Abbildung 6.2: Recovery-Controller als neuronales Netzwerk. Konzept und Abbildung vereinfacht nachgezeichnet bzw. entnommen aus [Lee u. a. 2019]

geringe Komplexität der Regelung der einzelnen Bewegungsabläufe nicht die Verwendung eines neuronalen Netzes rechtfertigt. Dennoch zeigt dieses Konzept, dass der Aktionsraum abstrakter sein kann. Auch ist von Interesse, ob diese Art der Regelung grundsätzlich möglich ist. So kann diese Methode in Betracht gezogen werden, wenn das Gesamtsystem komplexer ist.

6.3 Evaluation

Das Zusammenführen der erlernten Bewegungsabläufe war TEILWEISE erfolgreich. Implementiert wurde nur der Ansatz des endlichen Automaten. Es hat funktioniert, den Roboter in einer randomisierten Position zu erzeugen, aus welcher er sich hingesetzt hat und anschließend aufgestanden ist. Beim Übergang zwischen dem Aufsteh-Controller und dem Lauf-Controller ist er gescheitert. Das Scheitern drückte sich dadurch aus, dass er sofort hingefallen ist, ohne auch nur ansatzweise zu laufen. Dieses Problem kann darauf zurückgeführt werden, dass es nicht gelungen war, den Roboter in der Simulation exakt in der Standposition zu erzeugen. Stattdessen fiel er immer aus der Luft auf den Boden. Das heißt, dass der gewünschte Startzustand

beim Erlernen des Laufens in der Simulation nicht exakt produziert werden konnte. Der Roboter sieht den eigentlichen Startzustand beim Befolgen des endlichen Automaten zum ersten Mal und ist entsprechend nicht vorbereitet. Ein Lösungsansatz wäre es, dem Roboter zu Beginn einer Episode beim Training des Laufens immer den Geschwindigkeitsbefehl $[0.0, 0.0, 0.0]$ zu geben. Gut ist das allerdings nicht, weil viel Rechenzeit für das Stehen aufgewendet werden würde, was eigentlich trivial sein sollte. Der beste Lösungsansatz wäre es, in der Simulation einen Weg zu finden, den Roboter exakt im gewünschten Startzustand zu erzeugen.

6.4 Sim2Real-Gap

Die gezeigten Konzepte wurden nur in der Simulation verwendet. Soll das System auf den echten Roboter übertragen werden, so ist die Sim-Zu-Real Kluft (in der Literatur: „Sim2Real Gap“) zu überschreiten. Die echte Welt wird unausweichlich anders sein als die Simulationsumgebung. Entsprechend ist es nicht trivial, ein in der Simulation trainiertes neuronales Netz ohne Weiteres auf einem realen Roboter auszuführen.

6.4.1 Domain Randomization

Eine Möglichkeit, die Sim2Real Gap zu überbrücken, nennt sich Domain Randomization [Peng u. a. 2018]. Anstatt eine einzige Simulationsumgebung für das Trainieren des neuronalen Netzes zu verwenden, wird das Netz mit vielen verschiedenen Umgebungen trainiert. Das heißt, dass verschiedene Parameter der Simulation, wie zum Beispiel Reibungskoeffizienten, Steifheit von Gelenken, Latenzen usw., randomisiert werden. Aus [Jakobi u. a. 1995] geht hervor, dass artifiziell, während des Trainings hinzugefügtes Rauschen, in der Größenordnung von „realistic quantities“, die Korrespondenz zwischen Simulation und echter Welt erhöht. Das Rauschen kann für die Zustandsvariable und die Aktion angewendet werden. Domain Randomization erfordert viel Rechenzeit, weshalb Arbeiten existieren, die auf Simulatoren verzichten [Böhm u. a. 2022].

6.4.2 Aktuatormodell

Aktuatoren steuern die Bewegung eines Roboters, jedoch wird ein Befehl möglicherweise aufgrund von internen Eigenheiten eines Aktuators nicht exakt ausgeführt wie befohlen. Würde eine in der Simulation trainierte Policy in der echten Welt ausgeführt werden, die diese Eigenschaft nicht berücksichtigt, so ist es denkbar, dass diese Policy in der echten Welt fehlschlagen wird. Diesem Problem wird in [Hwangbo u. a. 2019] entgegengekommen mittels eines Aktuatormodells. Durch Supervised-Learning wird ein neuronales Netz trainiert, das aus einer Historie von Gelenkfehlern (Differenz zwischen Ziel- und tatsächlich angenommener Position) und Gelenkgeschwindigkeiten eine „action- to-torque relationship“ herstellt. Der Trainingsdatensatz wurde in der echten Welt erstellt (es geht um die Schätzung der echten Motoren), indem dem ANYmal Roboter Fuß-Trajektorien in Form von Sinuswellen, unterschiedlicher Amplituden ($5 \sim 10cm$) und Frequenzen ($1 \sim 25Hz$) befehligt wurden, die dann per

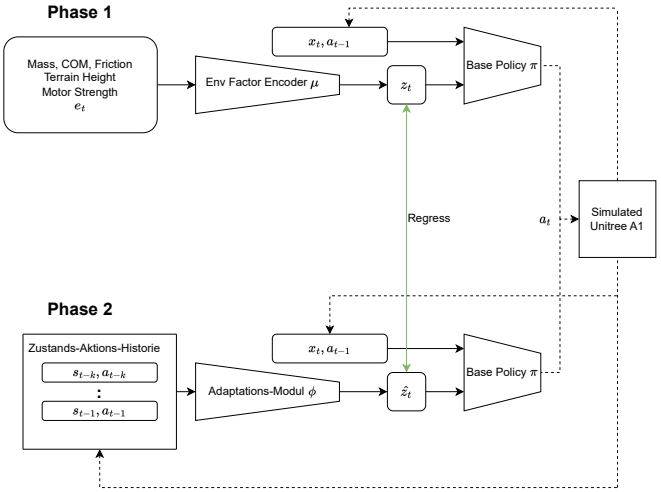


Abbildung 6.3: Rapid Motor Adaptation. Konzept und Abbildung leicht verändert nachgezeichnet bzw. entnommen aus [Kumar u. a. 2021]

inverser Kinematik in Gelenkpositionen umgewandelt wurden.
Inwiefern die Aktuatoren von einem Befehl abweichen, ist auch abhängig vom gewählten Aktionsraum. In Abbildung 4 aus [Chen u. a. 2022] geht hervor, dass die Abweichung zwischen Befehl und Realität stärker ist bei der Kontrolle über den PD-Regler als bei der direkten Drehmomentreglung.

6.4.3 Zustandsschätzung

Die Simulation enthält Informationen über Parameter, die in der echten Welt nicht bekannt sind. In Kapitel 3 wurde die Aufgabe des Hinsetzens vorgestellt. Dieses Modul soll nach Abbildung 6.1 aktiviert werden, sobald die Höhe des Roboterrumpfes unter 0.2m ist, das heißt es muss in der echten Welt eine Funktionalität existieren, um die Höhe zu schätzen. In [Lee u. a. 2019] steht geschrieben: „The estimated base height becomes unreliable when ANYmal falls.“. Aus diesem Grund wird in derselbigen Arbeit während der Simulation ein neuronales Netz h_ψ mittels Supervised-Learning zur Schätzung der Höhe trainiert. Die Eingabe ist die Ausrichtung und Gelenkkonfiguration. Das Training des Höhen-Schätzers findet parallel zum Training des Behaviour-Selectors statt, um die Rechenleistung auf tatsächlich besuchte Zustände zu konzentrieren. In jedem Simulationszeitschritt wird ein Paar (s_t, h_t) in einem Replay Buffer, siehe Kapitel 2, gespeichert. h_t ist dabei eine wahre Höhe, die der Simulation entnommen wird. Bei der Aktualisierung der Parameter ψ des Höhenschätzers werden K Exemplare aus dem Replay Buffer gesamplet und die Loss-Funktion $\sum_{j=0}^{K} ||h_j - h_\psi(s_j)||^2$ minimiert.

6.4.4 RMA - Rapid Motor Adaptation

Die Bewegung des Laufens ist unter verschiedenen Szenarien anders durchzuführen. Zu berücksichtigen sind die Art des Bodens, ob der Agent einer gewissen Last unter-

liegt oder ob er hoch- oder herunterläuft.
Rapid Motor Adaptation [Kumar u. a. 2021] funktioniert wie folgt: Sogenannte privilegierte Umweltinformationen env_t, die nur im Simulator existieren, werden in einen latenten Vektor z_t enkodiert. Die Umweltinformationen env_t beinhalten Informationen bezüglich der Reibung, der Nutzlast und Nutzlastposition auf dem Roboter und der Höhe des Terrains. Diese sind in der echten Welt ohne weiteres nicht verfügbar. In der Publikation wird beschrieben, dass sich aus der Abweichung des befehligten Zielwinkels und der tatsächlich angenommenen Position eines Gelenkes eine Abhängigkeit zu dem latenten Vektor z_t herstellen lässt. Aus diesem Grund wird vorgeschlagen, z_t aus einer Historie gesehener Zustände und Aktionen durch das Adaptations Modul ϕ_{RMA} zu schätzen:

$$\hat{z}_t = \phi_{RMA}(s_{t-k:t-1}, a_{t-k:t-1}) \tag{6.1}$$

Das neuronale Netz für die Schätzung $\hat{z}_t$ wird aktualisiert durch die Minimierung des Mean-Squared-Errors $\text{MSE}(\hat{z}_t, z_t) = \frac{1}{TN_{\text{env}}} \sum ||\hat{z}_t - z_t||^2$ durch das Gradientenabstiegsverfahren. T ist die Episodendauer und N_{env} ist die Anzahl der Simulationsumgebungen. Das ist möglich, weil der wahre latente Vektor z_t in der Simulation bekannt ist. Ein möglicher Ansatz für die Füllung des Trainingsdatensatzes ist es, eine bereits trainierte Basis-Policy π mit dem wahren latenten Vektor z_t auszuführen und in jedem Zeitschritt ein Trainingspaar $((s_{t-k:t-1}, a_{t-k:t-1}), z_t)$ einer Zustands-Aktions-Historie (Eingabe) und eines latenten Vektors (Ausgabe) zu speichern. Dies führt aber dazu, dass der Agent lediglich erfolgreiche Trajektorien sieht. Das widerspricht der Motivation von RMA. Das Adaptations Modul ϕ_{RMA} soll auch in unerwarteten Zuständen valide sein. Aus diesem Grund wird das Adaptations Modul On-Policy trainiert. Das heißt, dass die für das Training von ϕ_{RMA} verwendeten Daten unter Verwendung von ϕ_{RMA} entstehen. Die Basis-Policy π bestimmt die nächste Aktion während des Trainings von ϕ_{RMA} nicht durch $a = \pi(s_t, a_{t-1}, \mathbf{z_t})$, sondern durch den von ϕ_{RMA} geschätzten latenten Vektor: $a = \pi(s_t, a_{t-1}, \hat{\mathbf{z}}_\mathbf{t})$. Da ϕ_{RMA} randomisiert initialisiert wird und $\hat{z}_t$ zu Beginn unpräzise ist, wird der Agent Trainingspaare $((s_{t-k:t-1}, a_{t-k:t-1}), z_t)$ sehen, die er unter Verwendung des wahren latenten Vektors bei der Aktionsselektion nicht sehen würde.
Abbildung 6.3 stellt das Konzept grafisch dar: In **Phase 1** wird die Basis-Policy π trainiert, die unter Ausnutzung der enkodierten privilegierten Informationen eine gewisse Zielfunktion erfüllen soll. Um möglichst viele Ausprägungen enkodierter privilegierter Informationen zu sehen, sollte das Trainings-Terrain in der Simulation variiert werden. **Phase 2** stellt das soeben erklärte Verfahren dar, um das Adaptations Modul zu trainieren.

7 Zusammenfassung und Ausblick

Das Abschlusskapitel fasst die Erkenntnisse der Bachelorarbeit zusammen. In den vorherigen Kapiteln wurden verschiedene Aktionsräume vorgestellt: Die direkte Drehmomentregelung, die PD-Reglung und auf einer höheren Ebene, die Regelung einzelner vorher erlernter Bewegungsabläufe. Darauf aufbauend wird im Ausblick eine weitere Methode des Laufens vorgestellt. Diese zeigt, dass auch weitere abstraktere Aktionsräume existieren. In diesem Fall ist das ein Vektor, der die Parameter der Gangart reguliert.

7.1 Zusammenfassung

Diese Bachelorarbeit hat gezeigt, wie es möglich ist, mühsam entwickelte, möglicherweise fehleranfällige und suboptimale, plattformspezifische, konventionelle Methoden, durch per Deep Reinforcement Learning trainierte neuronale Netze zu ersetzen. Diese Methode benötigt kein Expertenwissen im klassischen Sinne. Die Fragestellung für den Entwickler ist nicht „Wie soll es funktionieren?“, sondern „Was soll funktioneren“. Die Antwort dieser Frage findet sich dann in der mathematisch formulierten Belohnungsfunktion wieder, in die die Komplexität ausgelagert wird. Nicht nur, dass es schwierig ist, die Terme richtig zu wählen. Es ist ebenfalls notwendig die Gewichtung richtig zu wählen. Aus [Booth u. a. 2023] geht hervor, dass 92% aller Reinforcement-Learning Praktizierenden ihre zuletzt entwickelte Belohnungsfunktion mittels Trial-und-Error entwarfen. Im Durchschnitt probierten die Probanden 4.1 verschiedene Belohnungsfunktionen aus. Auch in dieser Bachelorarbeit waren viele Anläufe notwendig. Die Terme wurden zwar gelungenen Publikationen entnommen, jedoch war es schwierig die richtige Gewichtung der Terme zu finden. Gekoppelt damit, dass für jeden neuen Anlauf das neuronale Netz neu trainiert werden muss, ist der Zeitaufwand für ein Projekt, das ein solches Verfahren implementieren will, entsprechend zu berechnen. Da die Belohnungsfunktion als Ganzes fungiert, ist es nicht direkt offensichtlich, welcher Term mit welcher Gewichtung zu Fehlverhalten geführt hat.

7.2 Ausblick

Das in Kapitel 5 vorgestellte Verfahren zum Erlernen des Laufens basiert auf Geschwindigkeitsbefehlen. Das Annehmen einer Geschwindigkeit ist im Allgemeinen allerdings nicht ausreichend für eine erfolgreiche Fortbewegung. Steile Wege erfordern, je nachdem, ob die Bewegung auf- oder abwärts erfolgt, mehr Kraft oder kleinere Schritte. Das Aufsteigen von Treppen erfordert höhere Fußhöhen. Basierend auf diesem Problem, stellt [Aractingi u. a. 2023] einen Lösungsansatz vor. Kernaspekt ist ein erweiterter Befehlsvektor:

$$\omega = (V_{cmd}, p_{f,z}^{\max}, c_{jpos}, k_p, k_d, \phi_n) \tag{7.1}$$

wobei c_{jpos} das Gewicht für den Term ist, der die Abweichung von der Nominalkonfiguration bestraft. In Abhängigkeit von dessen Höhe wagt der Roboter, horizontal betrachtet, kürzere oder längere Schritte. Die Idee ist es diese Parameter als variable Befehle anzusehen, in deren Abhängigkeit verschiedene Gangarten produziert werden sollen. Das Verfahren gliedert sich in zwei Policies. **Low-Level Policy**: Die Aufgabe der Low-Level Policy ist es, in Abhängigkeit der vom Befehlsvektor 7.1 festgelegten Parameter eine Laufbewegung zu produzieren. Es wird die Rapid Motor Adaptation (RMA) Strategie aus Kapitel 6.4.4 angewendet. Zusätzlich wurden in [Aractingi u. a. 2023] privilegierte Informationen „related to the robot's feet positions and contacts and the base linear velocity" enkodiert. Die Low-Level Policy wird mit verschiedenen Variationen des Befehlsvektors ω trainiert. Die Ausprägungen von ω werden zu Beginn einer Trainingsepisode oder zu zufälligen Zeiten während einer Episode bestimmt. **High-Level Policy:** Unter der Annahme, dass die Low-Level Policy erfolgreich unter verschiedenen Ausprägungen von ω trainiert wurde, ist es die Aufgabe der High-Level Policy, die Einträge des Befehlsvektors so zu regeln, dass der Roboter das Terrain erfolgreich traversieren kann. Basierend auf Standardwerten des Befehlsvektors ω werden je nachdem, wie die Situation es erfordert, Werte addiert oder subtrahiert. Die „Situation" charakterisiert sich durch den Zustand der Low-Level Policy und einer Höhen-Messung („Height Map") der Umgebung. Das Training der High-Level Policy basiert auf der Belohnungsfunktion $r_t^h = r_t^{vel} - 0.002|\tau_t^T \dot{\phi}_t|$, welche misst, ob eine Zielgeschwindigkeit eingehalten wurde unter der Bedingung, dass dies möglichst energieeffizient geschehen soll.
Unter der Annahme, dass alle bis hierhin gezeigten Konzepte erfolgreich implementiert wurden und einsatzfähig sind, lässt sich sagen, dass der Roboter viele verschiedene Arten von Geländern durchqueren kann. Falls er scheitert, weil er beispielsweise hinfällt, so erkennt er das und kann automatisch wieder aufstehen und weiterlaufen. Der nächste Schritt wäre es sich auf Aufgaben zu konzentrieren, die die kognitiven Fähigkeiten des Roboters erfordern. Denkbar wären Aufgaben, in denen der Roboter ein Gebiet durchqueren soll, unter der Zielfunktion möglichst wenig beschädigt zu werden und eine möglichst kurze Strecke zu hinterlegen. Die Aktionen würden die Gelenke nicht direkt kontrollieren, sondern den Roboter als Ganzes, in der Form von Geschwindigkeiten oder Koordinaten.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Aractingi u. a. 2023
ARACTINGI, Michel ; LÉZIART, Pierre-Alexandre ; FLAYOLS, Thomas ; PEREZ, Julien ; SILANDER, Tomi ; SOUÈRES, Philippe: A Hierarchical Scheme for Adapting Learned Quadruped Locomotion. (2023)

Bledt u. a. 2018
BLEDT, Gerardo ; POWELL, Matthew J. ; KATZ, Benjamin ; DI CARLO, Jared ; WENSING, Patrick M. ; KIM, Sangbae: Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE, 2018, S. 2245–2252

Bloesch u. a. 2017
BLOESCH, Michael ; BURRI, Michael ; SOMMER, Hannes ; SIEGWART, Roland ; HUTTER, Marco: The two-state implicit filter recursive estimation for mobile robots. In: *IEEE Robotics and Automation Letters* 3 (2017), Nr. 1, S. 573–580

Böhm u. a. 2022
BÖHM, Peter ; CHAPMAN, Archie ; POUNDS, Pauline: Training Directional Locomotion for Quadrupedal Low-Cost Robotic Systems via Deep Reinforcement Learning. (2022)

Booth u. a. 2023
BOOTH, Serena ; KNOX, W B. ; SHAH, Julie ; NIEKUM, Scott ; STONE, Peter ; ALLIEVI, Alessandro: The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In: *Proceedings of the AAAI Conference on Artificial Intelligence* Bd. 37, 2023, S. 5920–5929

Chen u. a. 2022
CHEN, Shuxiao ; ZHANG, Bike ; MUELLER, Mark W. ; RAI, Akshara ; SREENATH, Koushil: Learning torque control for quadrupedal locomotion. In: *arXiv preprint arXiv:2203.05194* (2022)

FaramaFoundation 2023
FARAMAFOUNDATION: *Zitat* `https://gymnasium.farama.org/content/basic_usage/` *[Zugriff am 20.10.2023 10:00].* 2023

Germann u. a. 2005
GERMANN, Daniel ; HILLER, Manfred ; SCHRAMM, Dieter u. a.: Design and control of the quadruped walking robot ALDURO. In: *International Symposium on Automation and Robotics in Construction (ISARC)*, 2005

Hutter u. a. 2016
Hutter, Marco ; Gehring, Christian ; Jud, Dominic ; Lauber, Andreas ; Bellicoso, C D. ; Tsounis, Vassilios ; Hwangbo, Jemin ; Bodie, Karen ; Fankhauser, Peter ; Bloesch, Michael u. a.: Anymal-a highly mobile and dynamic quadrupedal robot. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)* IEEE, 2016, S. 38–44

Hwangbo u. a. 2019
Hwangbo, Jemin ; Lee, Joonho ; Dosovitskiy, Alexey ; Bellicoso, Dario ; Tsounis, Vassilios ; Koltun, Vladlen ; Hutter, Marco: Learning agile and dynamic motor skills for legged robots. In: *Science Robotics* 4 (2019), Nr. 26, S. eaau5872

Jakobi u. a. 1995
Jakobi, Nick ; Husbands, Phil ; Harvey, Inman: Noise and the reality gap: The use of simulation in evolutionary robotics. In: *Advances in Artificial Life: Third European Conference on Artificial Life Granada, Spain, June 4–6, 1995 Proceedings 3* Springer, 1995, S. 704–720

Kakade 2001
Kakade, Sham: Optimizing average reward using discounted rewards. In: *International Conference on Computational Learning Theory* Springer, 2001, S. 605–615

Kakade u. Langford 2002
Kakade, Sham ; Langford, John: Approximately optimal approximate reinforcement learning. In: *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002, S. 267–274

Kakiuchi u. a. 2017
Kakiuchi, Yohei ; Kamon, Masayuki ; Shimomura, Nobuyasu ; Yukizaki, Sou ; Takasugi, Noriaki ; Nozawa, Shunichi ; Okada, Kei ; Inaba, Masayuki: Development of life-sized humanoid robot platform with robustness for falling down, long time working and error occurrence. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE, 2017, S. 689–696

Kumar u. a. 2021
Kumar, Ashish ; Fu, Zipeng ; Pathak, Deepak ; Malik, Jitendra: Rma: Rapid motor adaptation for legged robots. In: *arXiv preprint arXiv:2107.04034* (2021)

Lee u. a. 2019
Lee, Joonho ; Hwangbo, Jemin ; Hutter, Marco: Robust recovery controller for a quadrupedal robot using deep reinforcement learning. In: *arXiv preprint arXiv:1901.07517* (2019)

Lin 1992
Lin, Long-Ji: *Reinforcement learning for robots using neural networks.* Carnegie Mellon University, 1992

Makoviichuk u. Makoviychuk 2021
MAKOVIICHUK, Denys ; MAKOVIYCHUK, Viktor: *rl-games: A High-performance Framework for Reinforcement Learning.* `https://github.com/Denys88/rl_games`, May 2021

Makoviychuk u. a. 2021
MAKOVIYCHUK, Viktor ; WAWRZYNIAK, Lukasz ; GUO, Yunrong ; LU, Michelle ; STOREY, Kier ; MACKLIN, Miles ; HOELLER, David ; RUDIN, Nikita ; ALLSHIRE, Arthur ; HANDA, Ankur u. a.: Isaac gym: High performance gpu-based physics simulation for robot learning. In: *arXiv preprint arXiv:2108.10470* (2021)

Marbach u. Tsitsiklis 2003
MARBACH, Peter ; TSITSIKLIS, John N.: Approximate gradient methods in policy-space optimization of Markov reward processes. In: *Discrete Event Dynamic Systems* 13 (2003), S. 111–148

Miller u. a. 2020
MILLER, Ian D. ; CLADERA, Fernando ; COWLEY, Anthony ; SHIVAKUMAR, Shreyas S. ; LEE, Elijah S. ; JARIN-LIPSCHITZ, Laura ; BHAT, Akhilesh ; RODRIGUES, Neil ; ZHOU, Alex ; COHEN, Avraham ; KULKARNI, Adarsh ; LANEY, James ; TAYLOR, Camillo J. ; KUMAR, Vijay: Mine Tunnel Exploration Using Multiple Quadrupedal Robots. In: *IEEE Robotics and Automation Letters* 5 (2020), Nr. 2, S. 2840–2847. `http://dx.doi.org/10.1109/LRA.2020.2972872`. – DOI 10.1109/LRA.2020.2972872

Mnih u. a. 2013
MNIH, Volodymyr ; KAVUKCUOGLU, Koray ; SILVER, David ; GRAVES, Alex ; ANTONOGLOU, Ioannis ; WIERSTRA, Daan ; RIEDMILLER, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

NVIDIA 2023a
NVIDIA: *Zitat `https://docs.omniverse.nvidia.com/isaacsim/latest/overview.html` [Zugriff am 20.10.2023 10:00].* 2023

NVIDIA 2023b
NVIDIA: *Zitat `https://github.com/NVIDIA-Omniverse/OmniIsaacGymEnvs` [Zugriff am 20.10.2023 10:00].* 2023

Peng u. a. 2018
PENG, Xue B. ; ANDRYCHOWICZ, Marcin ; ZAREMBA, Wojciech ; ABBEEL, Pieter: Sim-to-real transfer of robotic control with dynamics randomization. In: *2018 IEEE international conference on robotics and automation (ICRA)* IEEE, 2018, S. 3803–3810

Peng u. Van De Panne 2017
PENG, Xue B. ; VAN DE PANNE, Michiel: Learning locomotion skills using deeprl: Does the choice of action space matter? In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2017, S. 1–13

Schulman u. a. 2015a
SCHULMAN, John ; LEVINE, Sergey ; ABBEEL, Pieter ; JORDAN, Michael ; MORITZ, Philipp: Trust region policy optimization. In: *International conference on machine learning* PMLR, 2015, S. 1889–1897

Schulman u. a. 2015b
SCHULMAN, John ; MORITZ, Philipp ; LEVINE, Sergey ; JORDAN, Michael ; ABBEEL, Pieter: High-dimensional continuous control using generalized advantage estimation. In: *arXiv preprint arXiv:1506.02438* (2015)

Schulman u. a. 2017
SCHULMAN, John ; WOLSKI, Filip ; DHARIWAL, Prafulla ; RADFORD, Alec ; KLIMOV, Oleg: Proximal policy optimization algorithms. In: *arXiv preprint arXiv:1707.06347* (2017)

Smith u. a. 2022
SMITH, Laura ; KOSTRIKOV, Ilya ; LEVINE, Sergey: A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. In: *arXiv preprint arXiv:2208.07860* (2022)

Stückler u. a. 2006
STÜCKLER, Jörg ; SCHWENK, Johannes ; BEHNKE, Sven: Getting Back on Two Feet: Reliable Standing-up Routines for a Humanoid Robot. In: *IAS* Citeseer, 2006, S. 676–685

Sutton u. Barto 2018
SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement learning: An introduction*. MIT press, 2018

Sutton u. a. 1999
SUTTON, Richard S. ; MCALLESTER, David ; SINGH, Satinder ; MANSOUR, Yishay: Policy gradient methods for reinforcement learning with function approximation. In: *Advances in neural information processing systems* 12 (1999)

Thomas 2014
THOMAS, Philip: Bias in natural actor-critic algorithms. In: *International conference on machine learning* PMLR, 2014, S. 441–448

Symbolverzeichnis

$\exp_t$	Erfahrung als Quadrupel $(s_t, a_t, r(s_t), s_{t+1})$
L_i	Loss Funktion der i-ten Iteration
y_i	Zielwert der Approximation
η	Erwartungswert des diskontierten Gewinnes bei Befolgen der Policy π
γ	Diskontierungsfaktor
$\hat{A}^{GAE(\gamma,\lambda)}$	General Advantage Estimator
$\mathcal{P}$	Zustandsübergangsfunktion
$\mathcal{S}$	Zustandsmenge
π	Policy
ρ_0	Wahrscheinlichkeitsverteilung für den Anfangszustand s_0
ρ_π	Wahrscheinlichkeitsverteilung für das Auftreten eines Zustandes über einen Zeithorizont unter der Policy π
$A^{\pi,\gamma}$	Advantage-Funktion
a_t	Aktion zum Zeitpunkt t
g	Policy Gradient
$Q^{\pi,\gamma}$	Zustands-Aktions-Wert-Funktion. Bildet unter einer Policy π ein Zustands-Aktions-Paar (s_t, a_t) auf einen erwarteten diskontierten Gewinn ab
$Q^{\pi,\gamma}$	optimale Zustands-Aktions-Wert-Funktion
r	Belohnungsfunktion
s_t	Zustand zum Zeitpunkt t
$V^{\pi,\gamma}$	Zustands-Wert-Funktion. Bildet unter einer Policy π einen Zustand s_t auf einen erwarteten diskontierten Gewinn ab
V_*	optimale Zustands-Wert-Funktion
$\mathcal{A}$	Aktionsmenge
$\hat{z}_t$	env_t enkodiert - geschätzt durch ϕ_{RMA}

$\omega^*_{angVel,z}$ befehligte Drehrate entlang z Achse

ϕ Gelenkkonfiguration des Roboters

ϕ_{RMA} Adaptations Modul - neuronales Netz zur Schätzung von z_t

τ Drehmoment

env_t privilegierte Umweltinformationen

e_g Projizierte Schwerkraft

eot_i End-of-Travel eines Roboterbeines i

h_ψ Höhenschätzer - neuronales Netz zur Schätzung der Roboterhöhe

k_c Kurrikulum-Faktor

$p^{\max}_{f,z}$ Zielfußhöhe zum Ende der Schwungphase

$v^*_{linVel,xy}$ befehligte lineare Geschwindigkeit entlang x und y Achse

$v_{eot,i}$ Geschwindigkeit zum End-of-Travel eines Roboterbeines i

z_t env_t enkodiert - wahr

C Skalierfaktor der Trust Region

D_{KL} Kullback Leibler Divergenz

L_π lokale Approximation von η

Versicherung an Eides Statt

Ich versichere an Eides Statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen übernommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Ich versichere an Eides Statt, dass ich die vorgenannten Angaben nach bestem Wissen und Gewissen gemacht habe und dass die Angaben der Wahrheit entsprechen und ich nichts verschwiegen habe. Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, namentlich die Strafandrohung gemäß § 156 StGB bis zu drei Jahren Freiheitsstrafe oder Geldstrafe bei vorsätzlicher Begehung der Tat bzw. gemäß § 163 StGB bis zu einem Jahr Freiheitsstrafe oder Geldstrafe bei fahrlässiger Begehung.

Ort, Datum

Unterschrift