HTWK Leipzig
Fakultät Informatik und Medien

Von der Simulation aufs Spielfeld: Reinforcement Learning für dynamische Schussbewegungen im Roboterfußball

Bachelorarbeit
von Felix Loos

Studiengang:	Informatik B.Sc.
Erstprüfer:	Prof. Dr. rer. nat. Jens Wagner
Zweitprüfer:	M. Sc. Tobias Jagla
Abgabedatum:	22. September 2025
Ort:	Leipzig

Kurzfassung

Diese Arbeit befasst sich mit der Entwicklung einer Schussbewegung für einen humanoiden Roboter mithilfe von Reinforcement Learning (RL). Im Gegensatz zu klassischen Ansätzen mit fest vorgegebenen Motorsequenzen erlaubt RL eine dynamische Anpassung an veränderliche Umgebungen und fördert eine stabile Ganzkörperkontrolle. Zu diesem Zweck wurde eine physikbasierte Simulation in NVIDIA Isaac Gym auf Basis der Booster Gym für den humanoiden T1-Roboter aufgebaut. Innerhalb dieser Umgebung wurde ein Belohnungsdesign entwickelt, das dichte Hilfsbelohnungen zur Steuerung des Lernprozesses mit spärlichen Zielbelohnungen kombiniert, um ein gezieltes Schussverhalten zu erlernen. Um den Reality Gap zu überbrücken, kamen Domain Randomization und gezieltes Finetuning zum Einsatz. Die erlernte Strategie konnte erfolgreich ohne zusätzliches Training in der realen Welt (Zero-Shot-Transfer) auf den Roboter übertragen werden. Die Ergebnisse zeigen, dass der Agent Schussbewegungen sowohl in der Simulation als auch auf der echten Hardware ausführen kann. Damit wird deutlich, dass RL in Kombination mit einem geeigneten Belohnungsdesign und Sim-to-Real-Methoden ein vielversprechender Ansatz für komplexe, dynamische Bewegungen humanoider Roboter ist und eine Grundlage für weitere Fähigkeiten wie Dribbeln oder Torwartverhalten bietet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Fußball stellt ein besonders schwieriges Problem für Roboter dar [1]. In einer sich kontinuierlich ändernden Umgebung müssen präzise Ganzkörperbewegungen ausgeführt werden. Eine anspruchsvolle und für das Spiel entscheidende Bewegung ist ein gezielter Schuss, der neben Präzision auch Gleichgewicht und Ballgefühl erfordert.

Im klassischen Ansatz werden feste Bewegungsabläufe einprogrammiert, bei denen die Motoren fixe Positionen in festgelegten zeitlichen Intervallen erreichen [2, 3]. Diese Sequenzen können allerdings nicht dynamisch auf Veränderungen in der Umgebung reagieren, was bedeutet, dass der Roboter während des Schusses leicht durch Hindernisse oder andere Roboter umgestoßen werden kann.

Um diese Limitationen zu umgehen, gibt es verschiedene Ansätze, die Stabilität und Exaktheit der programmierten Bewegung miteinander zu verhandeln [4]. Allerdings kommt auch dieser Ansatz an seine Grenzen, wenn mehr Dynamik gefordert ist. Jeder neue Aspekt, wie Reagieren auf unterschiedliche Ballpositionen, Ausfallschritte bei besonders starken Stößen oder Anpassen des Ziels, benötigt neue, unter Umständen aufwendige Implementierungen.

Maschinelle Lernverfahren sollen Abhilfe schaffen, indem sie komplexes dynamisches Verhalten in einer simulierten Umgebung weitgehend frei explorativ ermitteln, anstatt extern vorgegeben zu sein. Pionierprojekte für Spielfertigkeiten im Fußball, wie Dribbeln [5], Torwartverhalten [6] oder Schießen [7], haben vielversprechende erste Ergebnisse geliefert. Ambitioniertere Forschungsprojekte, die statt einzelner Bewegungen ganze Verhaltensmuster und Strategien lernen, wurden bereits erfolgreich in Simulationen [8] und mit echten Robotern [9] erprobt. Dabei sind Ansätze für ein vollständiges Spielverhalten im Vergleich zu spezialisierten Lernzielen sehr rechenintensiv und aufwendig in der Planung und Durchführung. Angesichts dessen betrachtet diese Arbeit nur das Erlernen einer einzelnen Bewegung, was ebenfalls perspektivisch ermöglicht, den Lernerfolg mit konkurrierenden oder zukünftigen Ansätzen leichter zu vergleichen.

Wenn sich die vielversprechenden Ansätze auch in Wettkampfumgebungen, wie dem *RoboCup*, beweisen sollten, werden wir in Zukunft dynamischeren, menschenähnlicheren und spannenderen Roboterfußball sehen können.

1.0.1 Zielsetzung

Ziel dieser Arbeit ist es, ein Schussverhalten für einen T1 Roboter von Booster Robotics [10] zu entwickeln. Das Verhalten soll mit dem maschinellen Lernverfahren, *Reinforcement Learning*, in einer Simulation trainiert und anschließend auf den echten Roboter übertragen werden. Es soll gezeigt werden, wie die Simulation und der Trainingsprozess aufgebaut sein müssen, um eine direkte Übertragung des

gelernten Verhaltens aus der Simulation auf den echten Roboter zu gewährleisten. Ein nachgelagertes Trainieren oder Adjustieren des Verhaltens mit echten Daten wäre somit nicht erforderlich. Im Mittelpunkt stehen das Belohnungsdesign, das das Lernziel festlegt und Qualität, wie Übertragbarkeit des Verhaltens, maßgeblich bestimmt, sowie die Evaluationsstrategien, mit denen dieses Design schrittweise erarbeitet wurde.

2 Grundlagen

Um die Umsetzung dieser Arbeit nachvollziehbar zu machen, werden in diesem Kapitel die notwendigen theoretischen und technischen Grundlagen erläutert. Zunächst wird auf die Rolle von Simulationsumgebungen eingegangen, die eine sichere und effiziente Möglichkeit bieten, komplexe Bewegungen zu trainieren und zu testen. Anschließend wird das Reinforcement Learning vorgestellt, das als zentrales Lernverfahren dient, um das gewünschte Verhalten des Roboters zu erlernen. Dabei werden sowohl die grundlegenden Prinzipien als auch spezifische Algorithmen beschrieben, die sich für kontinuierliche Ganzkörperbewegungen in der Robotik eignen. Darüber hinaus wird die Herausforderung des sogenannten Reality Gaps thematisiert, der beim Transfer von Strategien aus der Simulation in die reale Welt auftritt. Abschließend wird der verwendete Roboter, der Booster T1, vorgestellt. Seine Eigenschaften und technischen Spezifikationen bilden die Grundlage für das Training, sowie die spätere Evaluierung auf der realen Hardware. Damit legt dieses Kapitel das Fundament für die methodische Umsetzung, die im folgenden Kapitel beschrieben wird.

2.1 Simulation

Im Gegensatz zum klassischen Ansatz setzen moderne Verfahren des maschinellen Lernens auf die Arbeit in Simulationsumgebungen [11]. Dieses virtuelle Training erlaubt die Erzeugung vielfältiger, dynamischer Kontexte, die massenhafte Parallelisierung verschiedener Versuchsinstanzen und das Reduzieren von Risiken für Mensch und Maschine. Bei entsprechender Rechenleistung bieten diese Ansätze ein erhebliches Plus an Effizienz, Rigorosität und Sicherheit und sind dabei bemerkenswert günstig [12].

2.1.1 Anforderungen an Simulationen

Zentral ist eine realistische Physik-Engine. Diese muss Bewegungen von Körpern unter Kräften, wie Gravitation und Drehmomenten, auf Gelenken möglichst genau simulieren können. Speziell für das Simulieren der Robotermodelle ist eine genaue Rigid-Body-Dynamik nötig. Diese berechnet das Verhalten von Objekten unter Krafteinwirkung, die über Gelenken miteinander verbunden sind. Zusätzlich sollten Kollisionen und Reibungen von verschiedenen Objekten untereinander möglichst realistisch berechnet werden [13]. Ebenso wichtig ist die Möglichkeit, mit der Simulation detaillierte, virtuelle Umgebungen zu erzeugen, welche realistische Oberflächeneigenschaften und Lichtbedingungen beinhalten. Um vordefinierte Robotermodelle in die Umgebungen zu laden, ist es wichtig, dass gängige Modellformate

wie *Universal Scene Description* (USD) [14] oder *Unified Robot Description Format* (URDF) [15] unterstützt werden. Des Weiteren müssen Daten von Kameras, Kraftsensoren und Inertial Measurement Units (IMU) [1] realistisch gerendert bzw. generiert werden können. Zusammenfassend lässt sich sagen, dass eine passende Simulation für das Training von Bewegungen für Roboter einen digitalen Zwilling sowohl der physischen Plattform als auch der entsprechenden Umgebung schaffen muss. Dabei gilt: Eine authentischere Simulation erzielt realistischere Ergebnisse [16].

2.1.2 Funktionsweise von Simulationen

Zu Beginn einer Simulation wird eine virtuelle Umgebung erzeugt, in die die Robotermodelle geladen werden. Anschließend berechnet die Physik-Engine in diskreten Zeitschritten die zukünftigen Zustände. Dafür werden die Positionen und Geschwindigkeiten aller Objekte mit internen Kräften wie Motordrehmomenten und mit externen Kräften wie Schwerkraft oder Kontaktkräften verrechnet. Nach jedem Berechnungsschritt werden Sensordaten aus den Zuständen generiert und genutzt, um eine Aktion zu berechnen. Bei diesen Aktionen handelt es sich um Motorkräfte, die beim nächsten Berechnungsschritt umgesetzt werden. Je realistischer jeder der einzelnen Schritte ist, desto realistischer ist auch die gesamte Simulation [17, 18].

2.2 Reinforcement Learning

Reinforcement Learning wird in der Literatur als eigenständiges Lernparadigma zwischen Supervised und Unsupervised Learning beschrieben. Es basiert nicht auf direkter Beschriftung von Daten, sondern auf verzögertem Feedback, das über eine Belohnungsfunktion vermittelt wird [19]. Dabei wird durch Ausprobieren und automatisiertes Bewerten gelernt.

Im Reinforcement Learning interagiert ein *Agent* mit einer *Umgebung*. Der Agent wählt Aktionen, die den Zustand der Umgebung verändern, und erhält dafür Belohnungen. Ziel ist es, eine Strategie zu lernen, die den erwarteten langfristigen Gesamtertrag maximiert.

Der Zustandsraum umfasst alle möglichen Zustände, die über Aktionen erreichbar sind. Ein Aktionsraum umfasst alle möglichen Aktionen, die dem Agenten zur Verfügung stehen. Die Belohnungsfunktion R weist Zuständen (oder Zustands–Aktions-Paaren) numerische Belohnungen zu und spiegelt das Lernziel wider. Darauf auf-

[1] Eine *IMU* (Inertial Measurement Unit) ist ein Inertialsensorpaket aus Beschleunigungssensor(en) und Gyroskop(en), oft ergänzt um ein Magnetometer. Sie misst lineare Beschleunigungen $\mathbf{a}$ und Winkelgeschwindigkeiten $\boldsymbol{\omega}$ im Sensorkoordinatensystem und wird zur Zustands-/Pose-Schätzung (z. B. mit Filter- oder Optimierungsverfahren) verwendet.

bauend beschreibt die Wertfunktion V^π die erwartete Summe zukünftiger Belohnungen unter einer gegebenen Strategie (Policy) π:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad \text{mit } 0 \leq \gamma < 1,$$

wobei G_t die discounted Return-Summe, R_{t+k+1} die Belohnungen und γ der Discountfaktor ist, der bestimmt, wie stark zukünftige Belohnungen gegenüber unmittelbaren Belohnungen abgewertet werden. Die discounted Belohnungen sorgen dafür, dass auch weniger gute Aktionen in Kauf genommen werden, um später eine größere Belohnung zu erhalten.

Für das Beispiel Schießen ist eine mögliche Belohnungsfunktion die Geschwindigkeit des Fußes zum Ball. Je höher die Geschwindigkeit in Richtung Ball, umso härter fällt der Schuss aus. Andersherum, wenn der Fuß sich vom Ball weg bewegt, wird die Belohnung negativ und entspricht einer Bestrafung. Im Gegensatz dazu könnte eine Wertefunktion so erstellt werden, dass eine Geschwindigkeit weg vom Ball ebenfalls positiv bewertet wird, wenn das Ausholen des Beines später eine deutlich größere Belohnung ermöglicht.

Dabei werden zwei wesentliche Arten von Belohnungen unterschieden: dichte Hilfsbelohnungen, die jeden Zustand kontinuierlich bewerten und direktes Feedback geben, sowie spärliche Hilfsbelohnungen, die das Auftreten eines gewissen Zustandes belohnen [19]. Ein Beispiel für eine dichte Hilfsbelohnung ist die oben genannte Geschwindigkeit des Fußes in Richtung Ball, während ein Beispiel für eine spärliche Zielbelohnung das Berühren des Balls ist.

Die Strategie π des Agenten definiert, mit welcher Wahrscheinlichkeit er in einem Zustand s eine Aktion a auswählt. Ziel des Lernens ist es, diese Strategie so zu verbessern, dass die langfristig erwartete Belohnung maximiert wird.

Mit der Belohnungsfunktion und gegebenenfalls der Wertefunktion wird aus dem Zustandsraum eine Landschaft. Ergänzend kann auch der Aktionsraum der Landschaft hinzugefügt werden. Anschließend wird die Landschaft durch Ausprobieren verschiedener Aktionen in verschiedenen Zuständen erkundet.

Durch Exploration werden stichpunktartig einzelne Bereiche der Landschaft ausprobiert, ohne dass lokale Optima gesucht werden. Im Gegensatz dazu kann durch das Ausnutzen bekanntermaßen guter Aktionen (Exploitation) ein Bereich der Landschaft näher untersucht und lokale Optima gefunden werden.

Dieses Spannungsfeld wird als *Exploration-Exploitation-Dilemma* bezeichnet. Einerseits muss der Agent ausreichend explorieren, um neue und potenziell bessere Strategien zu entdecken. Andererseits muss er exploiten, also bereits bekannte gute Aktionen nutzen, um kurzfristig Belohnungen zu maximieren. Der Erfolg eines RL-Verfahrens hängt entscheidend davon ab, ein Gleichgewicht zwischen diesen beiden Anforderungen zu finden [20].

Die Umgebung ist ein Interface, in dem Aktionen Zustände zugeordnet werden, die von Belohnungsfunktionen bewertet werden. Über dieses Interface interagiert der Agent mit dem Resultat der Aktionen.

Die Anzahl der kontinuierlichen Interaktionen muss hoch genug gesetzt werden, damit die kumulierten Belohnungen konvergieren. Sobald die Belohnungen konvergieren, kann das Verfahren beendet werden. Allerdings lässt sich in der Regel nicht sagen, ob das gefundene Ergebnis einem globalen oder lediglich einem lokalen Optimum entspricht.

Ein Durchlauf des Verfahrens ist ein Trainingsdurchlauf. Im Folgenden kann das Resultat evaluiert werden und in einem weiteren Durchlauf können die Belohnungen angepasst werden (siehe Abschnitt 3.3.2).

2.2.1 Reinforcement Learning für Verhalten von Robotern

Das Lernen von Verhalten für Roboter lässt sich formal durch ein *Markov Decision Process* (MDP) beschreiben [21]. Ein MDP wird üblicherweise als Tupel

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \rho_0)$$

definiert. Die einzelnen Komponenten haben dabei folgende Bedeutung:

- $\mathcal{S}$ ist der Zustandsraum. Im Robotik-Kontext umfasst dieser alle möglichen Zustände, die durch die Sensorik beschrieben werden können, wie beispielsweise Gelenkwinkel, Gelenkgeschwindigkeiten, Kräfte oder externe Messungen durch Kameras oder IMUs. Die rohen Sensordaten werden dabei meist vorverarbeitet, etwa durch Normalisierung oder Filterung, um stabile Eingaben für das Lernverfahren zu gewährleisten [22]. Ein Beispiel für Normalisierung ist die Skalierung von Winkel- oder Geschwindigkeitswerten auf ein Intervall wie $[-1, 1]$, sodass alle Beobachtungen vergleichbare Größenordnungen aufweisen.

- $\mathcal{A}$ ist der Aktionsraum. Er umfasst alle möglichen Steuerbefehle an die Motoren des Roboters, wie zum Beispiel Sollpositionen, Geschwindigkeiten, Drehmomente oder auch erweiterte Parameter wie Steifigkeiten. Dabei sollten die physischen Limits der Motoren (zum Beispiel maximale Winkel oder Drehmomente) berücksichtigt werden, um den Aktionsraum sinnvoll zu beschränken und realistisch ausführbare Aktionen zu gewährleisten.

- $P(s' \mid s, a)$ ist die Übergangsfunktion. Sie beschreibt die Wahrscheinlichkeit, bei Ausführung einer Aktion $a \in \mathcal{A}$ aus einem Zustand $s \in \mathcal{S}$ in einen Folgezustand $s' \in \mathcal{S}$ überzugehen. Diese Dynamik ist in der Robotik meist durch die physikalischen Gesetze der Mechanik und die Dynamik der Motoren bestimmt.

- $R(s, a, s')$ ist die Belohnungsfunktion. Sie ordnet jedem Übergang von einem Zustand s über eine Aktion a in einen Folgezustand s' eine Belohnung $r \in \mathbb{R}$ zu. Die Belohnungsfunktion spiegelt das Lernziel wider, etwa die Fortbewegung des Roboters, das Greifen eines Objekts oder das Vermeiden von Hindernissen.

- $\gamma \in [0, 1)$ ist der Discountfaktor. Er bestimmt, wie stark zukünftige Belohnungen im Vergleich zu unmittelbaren Belohnungen gewichtet werden. Ein kleiner γ führt zu einem kurzfristig orientierten Verhalten, ein hoher γ zu einem langfristigen Verhalten.

- ρ_0 beschreibt die Anfangszustandsverteilung, aus der zu Beginn einer Episode der Startzustand s_0 gezogen wird.

Der Lernprozess verläuft typischerweise in diskreten Zeitschritten $t = 0, 1, 2, \ldots$. In jedem Schritt befindet sich der Agent (der Roboter) in einem Zustand $s_t \in \mathcal{S}$, wählt gemäß seiner Strategie (Policy) $\pi(a \mid s)$ eine Aktion $a_t \in \mathcal{A}$, wechselt mit Wahrscheinlichkeit $P(s_{t+1} \mid s_t, a_t)$ in einen Folgezustand s_{t+1} und erhält eine Belohnung

$$r_{t+1} = R(s_t, a_t, s_{t+1}).$$

Eine solche Abfolge $(s_t, a_t, r_{t+1}, s_{t+1})$ wird als *Transition* bezeichnet und beschreibt einen einzelnen Zeitschritt im Lernprozess.

Ziel ist es, eine Strategie π zu erlernen, welche die erwartete kumulierte und discounted Belohnung maximiert:

$$J(\pi) = \mathbb{E}_{\pi, P, \rho_0}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right].$$

Dabei beschreibt $J(\pi)$ die zu optimierende Zielfunktion, also den erwarteten Gesamtertrag einer Strategie π. Die Erwartung $\mathbb{E}_{\pi, P, \rho_0}$ berücksichtigt dabei die Wahrscheinlichkeiten der Strategie π, die Übergangsdynamik P der Umgebung und die Anfangszustandsverteilung ρ_0. Die Summe über alle Zeitschritte t aufsummiert die zukünftigen Belohnungen $R(s_t, a_t, s_{t+1})$, die mit dem Diskontfaktor γ abgewertet werden, sodass unmittelbare Belohnungen stärker gewichtet sind als weit entfernte. Das Ziel des Lernens ist es also, die Strategie π so zu verbessern, dass $J(\pi)$ möglichst groß wird.

Dieses formale Gerüst erlaubt es, ein Robotik-Szenario präzise zu beschreiben und verschiedene Algorithmen für das Lernen von Verhalten systematisch anzuwenden und zu vergleichen [23].

2.2.2 Reinforcement Learning Algorithmen

Ein weitverbreiteter Ansatz ist *Q-Learning*. Gelernt wird eine Wertefunktion (Q-Funktion) für Zustands-Aktions-Paare, welche klassischerweise tabellarisch angelegt wird [24].

Die Aktualisierung der Q-Funktion erfolgt iterativ nach der klassischen Q-Learning-Regel:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\Big).$$

Hierbei ist $Q(s_t, a_t)$ der bisherige Wert des Zustands-Aktions-Paares, $\alpha \in (0, 1]$ die Lernrate, r_{t+1} die erhaltene Belohnung nach Ausführung der Aktion a_t im Zustand s_t, und γ der Discountfaktor. Der Term $\max_{a'} Q(s_{t+1}, a')$ repräsentiert die Schätzung des besten zukünftigen Wertes im Folgezustand.

Das bedeutet: Wenn eine Aktion zu einer hohen Belohnung führt und zudem ein vielversprechender Folgezustand erreicht wird, wird der Q-Wert für (s_t, a_t) erhöht. Führt die Aktion dagegen zu einer geringen oder negativen Belohnung, wird der Q-Wert entsprechend abgesenkt. Auf diese Weise lernt der Agent durch wiederholte Interaktion mit der Umgebung, welche Aktionen langfristig vorteilhaft sind.

Der große Vorteil der Wertefunktion ist eine stabile Bewertung der Aktionen und Zustände. Demzufolge können Strategien statisch von der Q-Funktion abgeleitet werden. Die Aktionen werden so gewählt, dass der Wert der Q-Funktion maximiert wird. Dieses Verfahren kommt in hochdimensionalen Aktionsräumen allerdings an Grenzen, da alle Q-Werte für alle Zustands-Aktionspaare gespeichert werden müssen und bei jedem Optimierungsschritt in $\max_{a'} Q(s_{t+1}, a')$ durchsucht werden.

Eine Erweiterung des Ansatzes - *Deep Q-Learning* - ersetzt die tabellarische Q-Funktion durch ein neuronales Netz. Dabei wird versucht, die Q-Funktion zu approximieren. Folglich skaliert der Ansatz besser in hohen Dimensionen, da nur ein neuronales Netz ausgeführt werden muss.

Die Q-Funktion mit neuronalen Netzen wird ebenso wie ihr klassisches Pendant diskret ausgewertet und in Aktionen überführt. Der Aspekt macht dabei diesen Ansatz nicht direkt brauchbar für die Robotik, weil die Motoren auf Anweisungen in einem kontinuierlichen Format angewiesen sind. Um kontinuierliche Aktionen zu ermöglichen, kann die Auswertung der Q-Funktion ebenfalls durch ein neuronales Netz erfolgen. Diese Ansätze sind Q-basierte Actor-Critic-Varianten wie *Deep Deterministic Policy Gradient* (DDPG) [25] und *Soft Actor–Critic* (SAC) [26, 27], Diese Verfahren brauchen allerdings viele Zeitschritte und konvergieren langsam für komplexe Probleme. Das liegt vor allem daran, dass in hochdimensionalen Zustands- und Aktionsräumen sehr viele Transitionen benötigt werden, um ausreichend Erfahrung zu sammeln. Zusätzlich ist die Exploration in kontinuierlichen Aktionsräumen aufwendig, da zufällige Aktionen oft nicht sinnvoll sind

und dadurch viele Belohnungen uninformativ bleiben. Weiterhin können kleine Änderungen an Hyperparametern oder an der Umgebung das Lernen instabil machen, was die Konvergenz zusätzlich verlangsamt [28].

Ein weiterer häufig verwendeter Ansatz ist *Policy Gradient*, bei dem die Strategiefunktion direkt gelernt wird. Umgesetzt wird sie dabei als neuronales Netz und mit Gradient Ascent optimiert. Das neuronale Netz kann dabei kontinuierliche Werte zurückgeben und bei diskreten Werten deren Wahrscheinlichkeitsverteilung.

Bei Q-Learning (ohne neuronalen Netzen als Strategiefunktion) wird eine feste Strategiefunktion aus der Wertefunktion abgeleitet, somit konvergiert sie statisch. Hingegen hat Policy Gradient eine flexible Strategiefunktion unabhängig von einer Wertefunktion, die im Laufe des Trainings flexibel konvergieren kann [28]. Eine populäre Implementierung dieses Ansatzes ist *Proximal Policy Optimization* (PPO), welche Mechanismen einführt, die stabile Updates der Strategiefunktion beim Lernen begünstigen [29].

Moderne Implementierungen dieser beiden Ansätze wie SAC oder PPO nutzen die *Actor-Critic-Architektur*. Dabei gibt es eine Wertefunktion, den *Critic*, und eine Strategiefunktion, den *Actor*, die beide in Form eines neuronalen Netzes umgesetzt werden. Der Critic soll im Laufe des Trainings das Verhalten des Actors bewerten. Beim Lernen wird der Actor so optimiert, dass Verhalten auftritt, welches der Critic positiv bewertet. Der Critic lernt anhand von Belohnungen, das Verhalten des Actors und den Zustand der Umgebung zu bewerten [28]. Dabei kann er zusätzliche Informationen über den Zustand der Umgebung bekommen, die der Actor nicht erhält. Dies kann zu stabileren Bewertungen führen [30]. Die Actor-Critic-Architektur vereint die Vorteile von Wertefunktionen (stabile Beurteilungen der Aktionen) und die Vorteile der Strategiefunktion (flexible Strategiekonvergenz). Nachteilig ist jedoch, dass zwei neuronale Netze optimiert werden müssen. Das bedeutet höhere Komplexität und mehr Parameter, die angepasst werden müssen [26].

In dieser Arbeit wird mit der PPO-Implementierung gearbeitet, welche erprobt im Bereich der Robotik ist [31, 32, 33]. Trainiert wird die Strategie in der Nvidia Issac Gym Simulation und soll anschließend auf dem echten Roboter laufen. Hier ergibt sich jedoch das Problem, dass der Transfer nicht trivial möglich ist; zunächst muss der Simulation Gap überbrückt werden.

2.2.3 Übertragung in die Realität

Die größte Herausforderung ist der Reality Gap. Dieser beschreibt die Unterschiede zwischen Simulation und der echten Umgebung. Die Simulation kann zum Beispiel Aspekte wie Massen, Reibungen, Kontakte und Latenzen nicht genau abbilden. Dies sorgt dafür, dass Strategien, die in der Simulation funktionieren, sich nicht ohne Weiteres in die reale Welt übertragen lassen. Zusätzlich können Strategien in

der Simulation entwickelt werden, die spezifische Fehler der Simulation ausnutzen [34]. Ein Beispiel dasfür ist, wenn der Agent lernt, durch einen Boden ohne korrekt modellierte Kollision zu „gleiten“ oder sich durch unrealistisch niedrige Reibung schneller fortzubewegen. Solche Strategien funktionieren zwar in der Simulation, sind aber in der Realität physikalisch unmöglich und führen dort zu Fehlverhalten.

Eine spezielle Art des Reality Gaps stellt der Perception Gap dar. Dieser beschreibt die nicht genaue Abbildung von Sensorwerten wie Kamerabildern oder IMU-Werten. Ein gängiges Beispiel sind IMU-Werte, die in der Simulation rauschfrei sind. Eine Strategie könnte davon ausgehen, dass diese Werte auch in der realen Umgebung immer ideal sind. Das kann zu Fehlverhalten führen, wenn die echte IMU zum Beispiel durch Magnetfelder beeinflusst wird [35].

Die naheliegendste Lösung der oben genannten Probleme ist, die Simulation genauer zu machen. Dafür werden der Roboter und seine Umgebung möglichst genau vermessen und anschließend in der Simulation modelliert. Dieser Ansatz wird Systemidentifikation genannt [34, 36].

Allerdings ist es nicht möglich alle Parameter perfekt zu modellieren. Um dennoch die Wahrscheinlichkeit der Übertragbarkeit zu erhöhen, werden bei dem Ansatz der Domainrandomization die einzelnen Simulationsparameter bei jedem Trainingsdurchlauf zufällig eingestellt. Diese sind typischerweise die Physikparameter wie Masse oder Reibung, Offsets, Latenzen und Rauschen von Motoren, Sensorrauschen und Umgebungsparameter wie Positionen und Aussehen von Objekten. Auf diese Weise lernt der Agent generelles Verhalten in verschiedenen Umgebungen [35].

Wenn die realen Bedingungen in der Zufallsverteilung liegen, ist es möglich, die Strategie direkt ohne weiteres Lernen in die echte Umgebung zu übertragen (Zero-Shot) [34, 31]. Dabei gilt es, die Breite der Zufallsverteilung abzuwägen. Eine zu breite Verteilung kann zu suboptimalen und konservativen Strategien führen, während andererseits eine zu enge Verteilung dazu führen kann, dass die echten Parameter außerhalb liegen und somit keine Übertragung in die echte Umgebung möglich ist [34].

Insgesamt zeigt sich, dass der Reality Gap nicht vollständig vermieden werden kann. Durch Verfahren wie Systemidentifikation und Domainrandomization lässt sich die Lücke jedoch so weit verkleinern, dass Strategien mit höherer Wahrscheinlichkeit erfolgreich von der Simulation auf die reale Welt übertragen werden können. Damit wird der Simulationseinsatz trotz unvermeidbarer Abweichungen zu einem zentralen Werkzeug für das effiziente Training in der Robotik.

2.3 Booster T1 Roboter

Um schnell mit einem RL-Projekt starten zu können, sollte ein Roboter gewählt werden, für den es bereits erprobte Simulationsmodelle gibt. Folgend ist es möglich,

auf die Systemidentifikation zu verzichten. Weiterhin sollten Roboter bevorzugt werden, die besonders robust sind, da das Testen von Strategien auf echten Robotern zu instabilem Verhalten führen kann. Für dieses Projekt fiel die Wahl auf den T1 von Booster Robotics (siehe Abbildung 2.1). Ein großer Vorteil des Roboters ist die bereits existierende Trainings- und Simulationsumgebung Booster Gym. Diese beinhaltet bereits ein komplett aufgesetztes Training für das Laufen des T1.

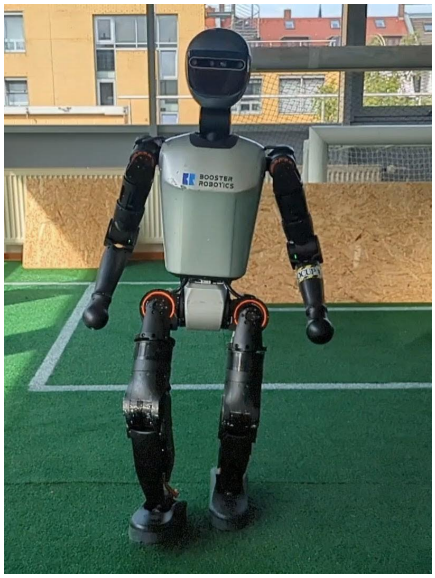


Abbildung 2.1: Booster T1 in der Laborumgebung.

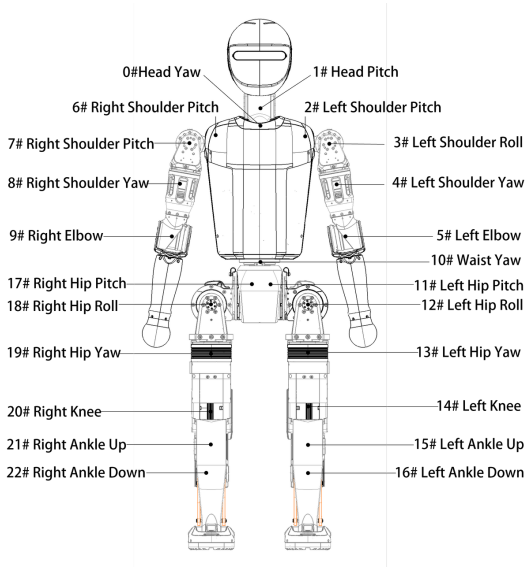


Abbildung 2.2: Gelenk- und Segmentbezeichnungen des Booster T1; entspricht der in *Booster Gym* verwendeten Nummerierung.

2.3.1 Aufbau des Roboters

Der Roboter wird von 23 Brushlessmotoren unterschiedlicher Größen und Stärken angetrieben (siehe Abbildung 2.2). Diese Motoren gelten als besonders robust und haben einen hohen Kraft-zu-Gewicht-Anteil [37]. Dabei kann der Motor im Knie 130 Nm an Drehmoment erzeugen, was auch für besonders harte Schüsse reicht. Als Kamerasystem dient eine *Intel Realsense D455*, die sowohl Tiefen- als auch Farbbilder der Umgebung erzeugen kann. Damit der T1 seine Ausrichtung wahrnehmen kann, besitzt er in seinem Oberkörper eine IMU die lineare Beschleunigungen und Winkelgeschwindigkeiten bereitstellt, aus denen die Ausrichtung errechnet werden kann. Für die Berechnungen und Ansteuerungen hat der T1 zwei Rechner

im Oberkörper. Der erste ist ein *Nvidia AGX Orin* mit 32 GB RAM und einer Grafikeinheit. Dieser wird für das Ausführen von rechenintensiven KI-Algorithmen genutzt. Ein weiterer Rechner beinhaltet eine *Intel i7 CPU* und 8 GB RAM, welcher für das Berechnen der Bewegungen benutzt wird.

3 Methodik

In diesem Kapitel wird die konkrete Umsetzung der Arbeit beschrieben. Aufbauend auf den in den Grundlagen dargestellten Konzepten werden hier die einzelnen Schritte detailliert vorgestellt, mit denen das Schussverhalten des T1-Roboters entwickelt, trainiert und schließlich auf den echten Roboter übertragen wurde. Dazu gehört zunächst die Beschreibung der eingesetzten Simulationsumgebung, die es ermöglicht, sowohl das Robotermodell, als auch die physikalischen Eigenschaften der Umgebung realitätsnah abzubilden. Anschließend wird der Trainingsprozess erläutert, in dem der Agent durch wiederholte Interaktion mit der Simulation das gewünschte Verhalten erlernt. Ein weiterer Schwerpunkt liegt auf dem Belohnungsdesign, das maßgeblich die Qualität und Stabilität des Lernprozesses bestimmt. Schließlich wird auf den Transfer von der Simulation auf den realen Roboter eingegangen, wobei die notwendigen Anpassungen und Herausforderungen des Sim-to-Real-Ansatzes thematisiert werden. Ziel dieses Kapitels ist es, die gesamte technische Pipeline transparent darzustellen und die einzelnen Umsetzungsschritte nachvollziehbar zu machen.

3.1 Simulation

Die Simulation bildet die Grundlage des Projekts. Sie umfasst sowohl die Umgebung als auch das Modell des Roboters. Als Ausgangspunkt der Entwicklung diente die *Booster Gym* [38]. Sie stellt ein vollständiges Set-up für Simulation und Training bereit und ist darauf ausgelegt, einem T1 das Laufen beizubringen. Die Booster Gym ist erprobt und hat gezeigt, dass sich damit Strategien entwickeln lassen, die auch auf den echten Roboter übertragbar sind. Technisch basiert sie auf der Nvidia *Isaac Gym* [39].

Im Folgenden wird der Aufbau der Isaac Gym erläutert. Dabei liegt der Fokus auf den Komponenten, die für das Schusstraining entscheidend sind.

3.1.1 Aufbau der Simulation

Grob lässt sich Issac Gym in drei Systeme aufteilen (siehe Abbildung 3.1). Erstens die Lernumgebung, welche die Logik für das Lernen beinhaltet. Zweitens die Environment-Logik, welche alle nicht-physikalischen Aspekte der Simulation verwaltet. Dies umfasst das Laden von Modellen, das Verwalten der Beobachtungen und Belohnungen und alle weiteren nicht-physikalischen Logiken. Als Drittes dient die Physik-Engine *PhysX* von Nvidia zum Berechnen aller physikalischen Vorgänge.

Die Lernumgebung bekommt Beobachtungen und Belohnungen von der Environment-Logik und gibt Aktionen an diese zurück. Die Kommunikation mit PhysX erfolgt

über die Isaac Gym Tensor API, welche die Aktionen weitergibt. Im Gegenzug liefert PhysX nach jedem Zeitschritt ein Update der physikalischen Zustände zurück, das anschließend in die Environment-Logik einfließt. Dort können mehrere Umgebungen parallel definiert werden. Jede dieser Umgebungen enthält stets einen Actor – in diesem Fall den Roboter – sowie die dazugehörige Umgebung. Sie sind physikalisch voneinander getrennt und interagieren nicht miteinander.

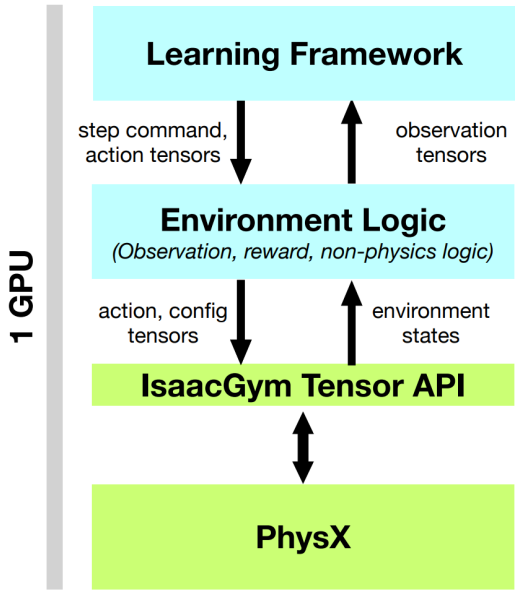


Abbildung 3.1: Isaac Gym – Komponenten und Datenfluss der RL-Simulationsarchitektur: Lernumgebung (Strategie/Training), Environment-Logik (Beobachtungen, Belohnungen, Aktionsweitergabe) und PhysX (GPU-basierte Physik) mit Kommunikation über die Isaac-Gym-Tensor-API. Quelle: Makoviychuk et al. (2021) [39].

Jede Umgebung stellt dabei einen eigenen physikalischen Raum dar, in dem Objekte, die als Rigid Bodies bezeichnet werden, existieren. Ein Rigid Body ist durch seine Form, Position, Rotation und Geschwindigkeit definiert. Der Actor kann aus mehreren Rigid Bodies bestehen, die durch Gelenke miteinander verbunden sind. Welche unterschiedliche Freiheitsgrade (Degrees of Freedom, DOF) aufweisen, die bestimmen, in welchen Richtungen sich das Gelenk bewegen kann; zum Beispiel hat ein Servomotor nur einen Freiheitsgrad, während ein Kugelgelenk zwei hat.

Mit der API können die wichtigsten Eigenschaften des Robotermodells abgefragt werden. Sie liefert Informationen zu Position, Orientierung sowie linearen und Winkelgeschwindigkeiten der Rigid Bodies. Darüber hinaus stellt sie für die

Freiheitsgrade Winkelpositionen und -geschwindigkeiten bereit und ermöglicht den Zugriff auf Kontaktkräfte zwischen Objekten.

Die Umgebung mit Beobachtungen und Belohnungen, der Lernprozess und PhysX-Parameter definieren zusammen einen *Task*. Der explizite Aufbau des Tasks für das Schießen soll im folgenden dargestellt werden.

3.1.2 Umgebung und Robotermodell

Das mit der Booster Gym gelieferte Robotermodell des T1 enthält bereits alle relevanten Motoren, Massen und Formen. Neben den sichtbaren Formen existieren vereinfachte Kollisionsformen. Sie bestimmen, wie Kollisionen berechnet werden, und reduzieren dabei die Komplexität der Simulation. In dem Modell des T1 werden Beine als Zylinder und Füße sowie Oberkörper als Rechtecke vereinfacht (siehe Abbildung 3.2). Damit findet eine vereinfachte Kollision zwischen Ball und Fuß statt. Das vergrößert den Reality-Gap, wird aber in Kauf genommen, um eine schnellere Simulation zu ermöglichen.

Der Ball wird als eine Kugel der Masse 0,2 kg und mit einem Durchmesser von 0,15 m modelliert. Der Task ist so aufgebaut, dass in einer Umgebung der Ball immer vor dem linken Fuß des Roboters platziert wird (siehe Abbildung 3.3). Das Koordinatensystem ist so ausgerichtet, dass der Roboter in Richtung X-Achse schaut, parallel zur Y-Achse steht und die Z-Achse nach oben zeigt. Die Umgebung ist nicht statisch aufgebaut, sondern wird zufällig initialisiert.

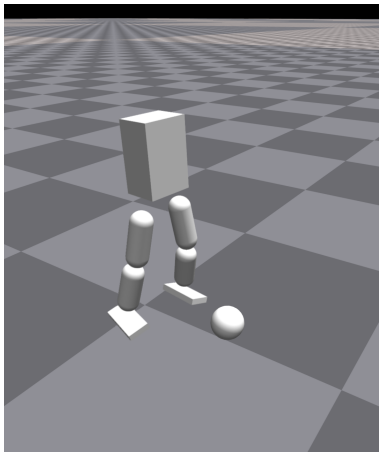


Abbildung 3.2: Vereinfachte Kollisionsformen im T1-Modell: Zylinder für Beinsegmente sowie Quader für Füße und Oberkörper.

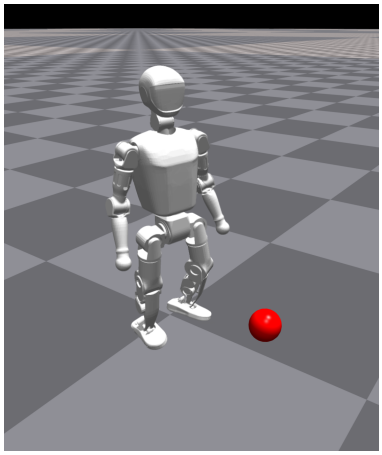


Abbildung 3.3: Schuss-Task in der Simulation: Ball initial vor dem linken Fuß des T1, Roboter blickt entlang der X-Achse (Y lateral, Z nach oben).

3.1.3 Domain Randomization

Um den Reality Gap zu verringern, kommt die Booster Gym bereits mit einigen Randomisierungen (siehe Tabelle A.3). So werden beim Initialisieren eines Roboters deren Parameter für die Freiheitsgrade und die Massen der einzelnen Bestandteile des Roboters zufällig aus einem gewissen Bereich gewählt. Dadurch erhält jeder simulierte Roboter leicht abweichende Eigenschaften. Das fördert Strategien, die auf unterschiedliche T1 Roboter übertragbar sind – denn auch die realen Roboter weichen in diesen Punkten voneinander ab.

Bei jedem Zurücksetzen einer Umgebung werden zudem weitere Parameter zufällig verändert. Dazu gehören die Anfangspositionen der Motoren, damit der Actor mit unterschiedlichen Startkonfigurationen zurechtkommt.

Um zum Lernen in diversen Ausgangssituationen beizutragen, werden die initiale Rotation des Roboters um die Z-Achse und die Position des Balls vor dem linken Fuß in einem gewissen Bereich zufällig gewählt.

Weiterhin wird eine Ansteuerungsverzögerung simuliert, indem die Weitergabe der Aktionen an die Physik-Engine um ein zufälliges Zeitintervall verzögert wird. Dies ist wichtig, da auch die reale Hardware Kommunikation und Signalübertragung nicht verzögerungsfrei ist und somit ein realistischeres Verhalten nachgebildet werden kann.

Um authentische Sensorwerte und Beobachtungen nachzustellen, wird ein Rauschen addiert, das nach dem Prinzip einer Gaußverteilung erzeugt wird.

3.2 Training

Unter Training versteht man den Prozess, bei dem der Agent durch wiederholte Interaktion mit der simulierten Umgebung ein Zielverhalten erlernt. Er sammelt dabei Erfahrungen und leitet aus diesen Verbesserungen für seine Strategie ab. Dieses Kapitel erläutert den Aufbau und Ablauf des Trainings.

3.2.1 Ablauf des Trainigs

Zu Beginn des Trainings werden alle Umgebungen wie in Abschnitt 3.1.2 beschrieben initialisiert. Um die Parallelisierbarkeit der Simulation auszunutzen, werden 4096 Umgebungen gleichzeitig erstellt. Die Anzahl an möglichen Umgebungen wird durch die Hardware des Trainingsrechners limitiert. Anschließend werden die einzelnen Simulationsschritte ausgeführt.

Ein Trainingsschritt entspricht 0,02 s. Zunächst werden die Aktionen des Actors relativ zu den Motor-Standardstellungen in Zielpositionen übersetzt. Anschließend laufen zehn Physikberechnungen (Physikschritte) à 0,002 s ab, wobei der

Verzögerungsparameter bestimmt, in welchem Schritt die Aktion an die Physikengine übergeben wird (vgl. Abschnitt 3.1.3).

Da die Physikengine Drehmomente erwartet, müssen die Zielpositionen erst umgewandelt werden. Dies geschieht nach folgenden Vorgehen:

$$\tilde{q}_i(t) = q_i^{\star}\big(t - \Delta t_{\text{act}}\big) \qquad \text{(verzögertes Ziel)} \tag{3.1}$$

$$\tau_i^{\text{PD}}(t) = K_{p,i}\big(\tilde{q}_i(t) - q_i(t)\big) - K_{d,i}\,\dot{q}_i(t) \qquad \text{(PD-Regler)} \tag{3.2}$$

$$\tau_i^{\text{fric}}(t) = \min\big(\tau_i^{\text{c}},\ |\tau_i^{\text{PD}}(t)|\big)\ \operatorname{sgn}\big(\tau_i^{\text{PD}}(t)\big) \qquad \text{(Coulomb-Reibung)} \tag{3.3}$$

$$\tau_i(t) = \operatorname{clip}\big(\tau_i^{\text{PD}}(t) - \tau_i^{\text{fric}}(t),\ -\tau_i^{\text{max}},\ \tau_i^{\text{max}}\big) \qquad \text{(Sättigung)} \tag{3.4}$$

Mit $\operatorname{clip}(x, a, b) = \max\{\min\{x, b\}, a\}$. Hierbei bezeichnet $q_i^{\star}$ die vom Actor vorgegebene Sollposition des Gelenks, die aufgrund der modellierten Verzögerung Δt_{act} erst zeitversetzt berücksichtigt wird. Der PD-Regler erzeugt daraus ein Drehmoment τ_i^{PD}, wobei $K_{p,i}$ die Steifigkeit und $K_{d,i}$ die Dämpfung darstellt. Um die statische Reibung zu berücksichtigen, wird zusätzlich ein Reibmoment τ_i^{fric} abgezogen, das durch die Reibschwelle τ_i^{c} begrenzt ist. Schließlich wird das resultierende Drehmoment τ_i auf die zulässigen Grenzwerte $\pm\tau_i^{\text{max}}$ beschnitten. Dem folgend wird ein Physikschritt ausgeführt und die Motorpositionen werden aktualisiert.

Sind alle Physikschritte abgeschlossen, erfolgt die Aktualisierung der Umgebungszustände. Dazu zählen der *Actor Root State* und der *Rigid Body State*, die Informationen über Position, Ausrichtung sowie Linear- und Winkelgeschwindigkeiten liefern. Ergänzend wird im *Contact Force State* die Kraft zwischen den sich berührenden Objekten erfasst.

Im Anschluss wird ermittelt, ob Stöße und Tritte auftreten und wie stark sie ausfallen. Die resultierenden Impulse werden im nächsten Physikschritt angewendet. Danach werden die Abbruchbedingungen geprüft. Tritt eine Bedingung ein, wird die gesamte Umgebung zurückgesetzt. Anschließend werden die Teilbelohnungen berechnet und zur Gesamtbelohnung aufsummiert. Zuletzt werden die Beobachtungen sowie die privilegierten Beobachtungen (nur für den Critic) erzeugt. Liegen Beobachtungen und Belohnungen vor, beginnt das Training.

3.2.2 Implementierung

Das Training ist nach der PPO-Implementierung [29] umgesetzt, die eine Actor Critic Architektur benutzt. Dies umfasst zwei neuronale Netze, deren Aufbau in Tabelle A.8 beschrieben ist. Der Aufbau des Trainings und die Einstellung der Parameter wurden von der Booster Gym übernommen (siehe Tabelle A.7 & Tabelle A.6). Als Eingabe bekommen der Actor und der Critic Beobachtungen (siehe Tabelle A.1 & Tabelle A.2), welche in einem asymmetrischen Trainingsprozess [30]

verwertet werden. Dafür bekommt der Critic mehr Informationen über die Umgebung als der Actor, welche die bereits erwähnten privilegierten Beobachtungen sind. Die Ausgabe des Actors sind die Abweichungen von den Standardmotorpositionen (siehe Abschnitt 3.2.1) der Unterkörpermotoren (Gelenk-ID 11-22, siehe Abbildung 2.2). Der Critic gibt einen einzelnen Wert zurück, der bewertet, wie gut ein aktueller Zustand ist. Ziel des Trainings ist es, dass der Actor lernt, wie er durch seine Aktionen in möglichst gut bewertete Zustände kommt. Der Critic hingegen lernt, wie er Zustände der Umgebung so bewerten kann, dass er den erwarteten zukünftigen Belohnungswert eines Zustands abschätzt und so dem Actor eine Orientierung für die Auswahl seiner Aktionen gibt.

In einer Trainingsepoche werden 24 Trainingsschritte ausgeführt. Dabei werden Erfahrungen in Form von Beobachtungen und Belohnungen angesammelt und für das anschließende Training benutzt. Das Optimieren findet in 20 Mini-Epochen statt. In jeder Mini-Epoche werden die Netzwerke einmal aktualisiert. Anschließend startet eine neue Epoche mit den aktualisierten Netzwerken. Details zu den Parametern des Trainings sind in Tabelle XX zu finden. Um diesen Trainingsprozess zu optimieren und Fehler zu beheben, ist eine wie folgend beschriebene Trainingsüberwachung entscheidend.

3.2.3 Trainingsüberwachung

Um einen Einblick in den Fortschritt und die Qualität des Trainings zu bekommen, werden verschiedene relevante Informationen im Loggingtool *Weights and Biases* [40] gespeichert. Dazu zählen unter anderem die einzelnen Belohnungen, wie stark die Vorhersage der Modelle vom gewünschten Zielwert abweicht (loss) und der Entropy-Wert des PPO. Ebenso wird alle 500 Epochen ein fünfsekündiges Video gespeichert. Da die gespeicherten Aufnahmen direkt einen unkomplizierten Einblick geben, ob das Hauptziel erreicht wurde, stellten sie sich als sehr nützliches Evaluationswerkzeug heraus. Ebenso konnte unerwünschtes Verhalten leicht identifiziert werden.

3.3 Lernsignale

Das Finden eines Belohnungsdesigns, das stabiles Schießen ermöglicht, ist die zentrale Aufgabe dieser Arbeit. Dafür wurden verschiedene Belohnungen eingeführt und getestet. Dieser Abschnitt soll neben den Belohnungen auch die Evaluierungsstrategie erläutern. Für eine vollständige Liste der Belohnungen, siehe Tabelle A.4.

3.3.1 Belohungsdesign

Die Belohnungen für den Schuss werden in zwei Phasen vergeben: zuerst für Zustände vor dem Schuss in Phase eins, in Phase zwei werden Belohnungen für Zustände nach dem Schuss vergeben.

Phase eins beinhaltet dichte Belohnungen, die den Agent in Richtung eines Schusses leiten sollen. Die wichtigste Belohnung wächst mit sinkender Distanz zwischen Fuß und Ball exponentiell. Weiterhin wird eine Oberkörperausrichtung in Schussrichtung belohnt, was eine richtige Ausrichtung des Agents vor dem Schießen bewirkt.

Alle Belohnungen, die in Phase 2 auftreten, sind spärliche Belohnungen, die nur vergeben werden, wenn der Agent den Ball schießt. Dazu zählt die Hauptbelohnung, die eine hohe Geschwindigkeit des Balles in eine Zielrichtung belohnt. Konkret bedeutet es, dass Verhalten belohnt werden soll, das zu einem starken Schuss führt. Je länger der Ball nach dem Schuss rollt, desto geringer fällt die Belohnung aus. Dies soll ermöglichen, dass Belohnungen, die für das Stehen nach dem Schuss zuständig sind, mit der Zeit überwiegen. Es hat sich gezeigt, wenn der Agent eine zu dominante Schussbelohnung bekommt, zu starke Schüsse erlernt werden, die das anschließende Stehen verhindern. Um dies zu vermeiden, wurde die maximale Belohnung für die Ballgeschwindigkeit in Zielrichtung begrenzt. Eine ergänzende Belohnung für die Ballbeschleunigung dient als spärliche Belohnung des Schießens.

Um das Überleben (nicht Erreichen einer Abbruchbedingung, siehe Abschnitt 3.3.3) des Agents zu belohnen, wird kontinuierlich eine Belohnung vergeben, solange der Agent lebt. Diese Belohnung ist vor dem Schießen halb so groß, wie danach. Es dient einerseits als spärliche Belohnung für das Schießen, und andererseits soll der Agent in der Phase vor dem Schießen ermutigt werden, auch riskantere Aktionen auszuprobieren, die zu einem verbesserten Schussverhalten führen.

Eine weitere, wichtige Gruppe von Belohnungen soll sicherstellen, dass der Roboter nicht umfällt. Eine Bestrafung des Agents erfolgt, je weiter seine Oberkörperhöhe vom Aufrechten Stehen (0,68 m) entfernt ist und nimmt quadratisch zu. Zusätzlich wird eine Bestrafung eingeführt, die das Nicht-Aufrechtstehen des Oberkörpers verhindern soll. Dafür wird der x- und y-Anteil des Gravitationsvektors im Koordinatensystem des Agents aufsummiert. Diese Bestrafung steigt ebenfalls quadratisch an, je schräger der Agent steht. Da Hinfallen mit einer Beschleunigung des Oberkörpers einhergeht, wird diese ebenfalls bestraft. Im gleichen Sinne wird eine Geschwindigkeit des Agents in der Z-Achse bestraft.

Damit der Agent lernt, nach dem Schuss stillzustehen und während des Schusses seinen Oberkörper möglichst wenig zu bewegen, wird er belohnt, je näher seine Geschwindigkeiten in der X- und Y-Achse an 0 m/s sind. Auch wird eine Winkelgeschwindigkeit in der Z-Achse von 0 rad/s belohnt und durch eine ergänzende

Bestrafung der Winkelgeschwindigkeit unterstützt.

Um einen Anreiz für möglichst energieeffiziente und ruhige Bewegungen zu schaffen, wurden verschiedene Bestrafungen eingeführt. Zentral ist dabei die Bestrafung der Aktionsraten. Diese entspricht der Distanz zwischen den aktuellen Motorpositionen und den Positionen aus dem letzten Trainingsschritt. Damit sollen große Änderungen in den Aktionen unwahrscheinlicher werden. Zusätzlich wird bestraft, wenn die Motoren hohe Drehmomente aufbringen müssen und diese Drehmomente nahe an die Hardwaregrenzen gelangen. Ähnlich werden Geschwindigkeit, Beschleunigung und Limits der Freiheitsgrade bestraft. Ergänzend wird eine hohe Leistung bestraft.

$$r_{\text{power}} = -\sum_i |\tau_i \cdot \dot{q}_i|$$

Dabei bezeichnet τ_i das vom i-ten Motor aufgebrachte Drehmoment und $\dot{q}_i$ dessen Winkelgeschwindigkeit. Das Produkt entspricht der Leistung des Motors. Durch die Summation über alle Motoren und das negative Vorzeichen wird erreicht, dass hohe Gesamtleistung als Bestrafung wirkt. Dies soll energieeffiziente Bewegungen ermöglichen.

Unkontrolliertes Fußverhalten ist ein Problem, das stabiles Stehen und Schießen erschwert. Um dies zu verhindern, wird der Agent bestraft, wenn seine Füße in der Yaw- (Drehung um Z-Achse) und Pitch-Ausrichtung (Drehung um X-Achse) von 0 Grad abweichen. Ergänzend dazu wird er bestraft, wenn die Yaw-Ausrichtungen der Füße unterschiedlich sind. Damit der Agent beim Schießen nicht seine Füße über den Boden schiebt, wird das Gleiten der Füße über den Boden bestraft.

Das Anpassen dieser Belohnungen stellt eine zentrale Aufgabe der Arbeit dar.

3.3.2 Evaluationsstrategien

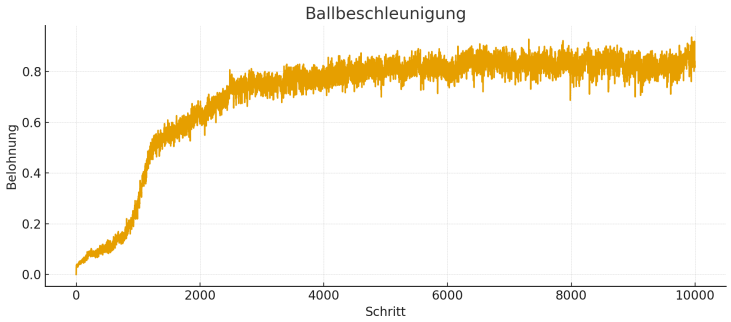
Belohnungen sollen dafür sorgen, dass gewünschtes Verhalten vermehrt auftritt und unerwünschtes Verhalten ausbleibt. Dafür müssen die Belohnungen passend skaliert sein und zum richtigen Zeitpunkt vergeben werden. Um dies zu bewältigen, wurden verschiedene Strategien entwickelt.

Zu Beginn lohnt es sich, mit einem minimalen Set an Belohnungen zu starten. Anfangs wurden die Belohnungen für das Lauftraining aus der Booster Gym benutzt. Diese sind nicht minimal für das Schießen, führen aber erprobterweise zu einem stabilen Verhalten. Ergänzt wird das Initialset durch die Hauptbelohnung, Ballgeschwindigkeit in Zielrichtung. Es hat sich herausgestellt, dass es sinnvoll ist, eine Hauptbelohnung zu haben, die in der Skalierung die restlichen Belohnungen übertrifft und darüber hinaus das Zielverhalten möglichst direkt belohnt.

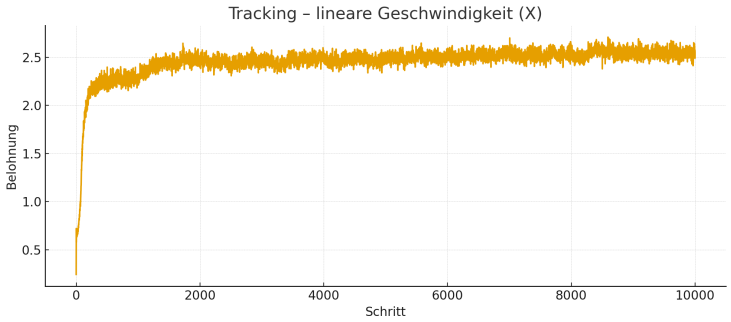
Die relative Skalierung der Belohnungen untereinander spielt eine entscheidende Rolle. Belohnungen mit höheren Werten werden stärker beim Training optimiert als solche mit niedrigeren Werten. Dies kann im Extremfall allerdings dazu führen,

dass wenn eine Belohnung zu dominant ist, andere Belohnungen völlig ignoriert werden. So führt eine übermäßige Belohnung der Ballgeschwindigkeit zum sehr starken Schießen des Agents, der jedoch anschließend umfällt. Die Skalierung entscheidet so über die Priorität der Belohnungen.

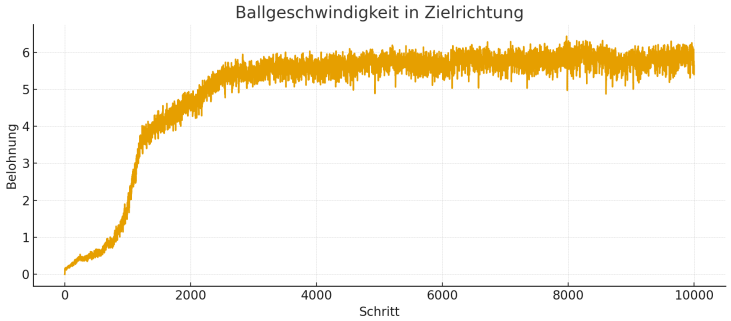
Das Einsehen und Überwachen der Skalierungen ist in Weights and Biases möglich. Abbildung 3.4 zeigt beispielhaft drei Belohnungen mit unterschiedlichen Skalierungen. Es ist zu sehen, dass die Hauptbelohnung (Abb. 3.4c) zu Beginn eine niedrigere Skalierung hat als die Belohnung für die lineare Geschwindigkeit x (Abb. 3.4b). Das lässt sich damit erklären, dass erst im späteren Trainingsverlauf der Agent lernt, den Ball zu schießen, und dadurch die Belohnung für die Ballgeschwindigkeit bekommt. Sodass erst nach ca. 1000 Trainingsepochen die Ballgeschwindigkeit dominanter als die Linare Geschwindigkeit wird. Daraus lässt sich ableiten, dass erst mit dem Trainingsfortschritt zu erkennen ist, ob die Skalierungen richtig gesetzt wurden.



(a) Ballbeschleunigung



(b) Lineare Geschwindigkeit x



(c) Ballgeschwindigkeit in Zielrichtung

Abbildung 3.4: Beispielhafte Skalierungen von Belohnungen im Trainingsverlauf. Dabei ist zu sehen, dass die Belohnung der Ballgeschwindigkeit in Zielrichtung erst im späteren Trainingsverlauf die Belohnung der linearen Geschindigkeit x als dominante Belohnung ablößt.

Dichte Belohnungen sollten eingeführt werden, wenn das gewünschte Verhalten trotz spärlicher Hauptbelohnung ausbleibt. Diese schränken die Exploration ein, können im Gegenzug den Agent schneller zu einem gewünschten Verhalten bringen. So führt das Belohnen einer geringen Distanz zwischen Fuß und Ball dazu, dass

der Agent schneller versteht, dass der Fuß zum Schießen in Richtung Ball bewegt werden muss, allerdings macht es im Gegenzug ein Ausholen mit dem Bein vor dem Schießen unwahrscheinlicher.

Einzelne Belohnungen können im Gegensatz zueinanderstehen. So werden schnelle Motorgeschwindigkeiten bestraft, die im Gegensatz zu den Belohnungen für einen harten Schuss stehen. Dies kann, bei falscher Skalierung, das Erlernen eines optimalen Verhaltens, bezogen auf einzelne Belohnungen, verhindern. Andererseits kann bei passender Skalierung ein Verhalten gefunden werden, welches beide Belohnungen berücksichtigt. Wie stark die beiden Belohnungen relativ zueinander skaliert sind, entscheidet dabei, welche Belohnung wie stark berücksichtigt wird.

Eine besondere Herausforderung bei dem Design und der Evaluation war der zweiphasige Charakter der Belohnungen. Das Wegfallen einzelner Belohnungen, sobald der Ball geschossen ist, interpretiert der Agent als Bestrafung, da die Gesamtbelohnung abfällt. Um dies zu kompensieren, wurden die dichten Belohnungen, die nur in der Vorschussphase oder in der Nachschussphase existieren, minimiert. Zusätzlich wurden frühere spärliche Belohnungen wie die Berührung des Balles ersetzt durch die Belohnung für die Ballgeschwindigkeit in Zielrichtung. Diese Belohnung tritt zwar nur spärlich auf, wird aber anschließend über einen längeren Zeitraum vergeben. Dies soll dazu beitragen, den Wegfall einzelner Belohnungen in der Nachschussphase zu kompensieren.

Um die Belohnungen für den Schuss auf Fehler zu überprüfen, reicht es nicht aus, sich die Gesamtskalierung wie in Abbildung 3.4 anzuschauen. Es ist zusätzlich erforderlich, sich die Skalierungen über den zeitlichen Verlauf anzuschauen (siehe Abbildung 3.5). Dabei ist zu erkennen, dass während des Schusses (siehe Anstieg der Belohnung der Ballgeschwindigkeit) eine starke Bestrafung für die Aktionsrate vergeben wird. Um eine in der Bilanz positive Gesamtbelohnung zu bekommen, muss beachtet werden, dass die Bestrafung ausreichend ausgeglichen wird. Sonst lernt der Agent nicht das Schießen, da er für die Schussbewegung insgesamt bestraft wird.

Neben den bereits ausgeführten Belohnungen sind Abbruchbedingungen eine weitere Möglichkeit, Einfluss auf das Erlernen des Verhaltens zu nehmen.

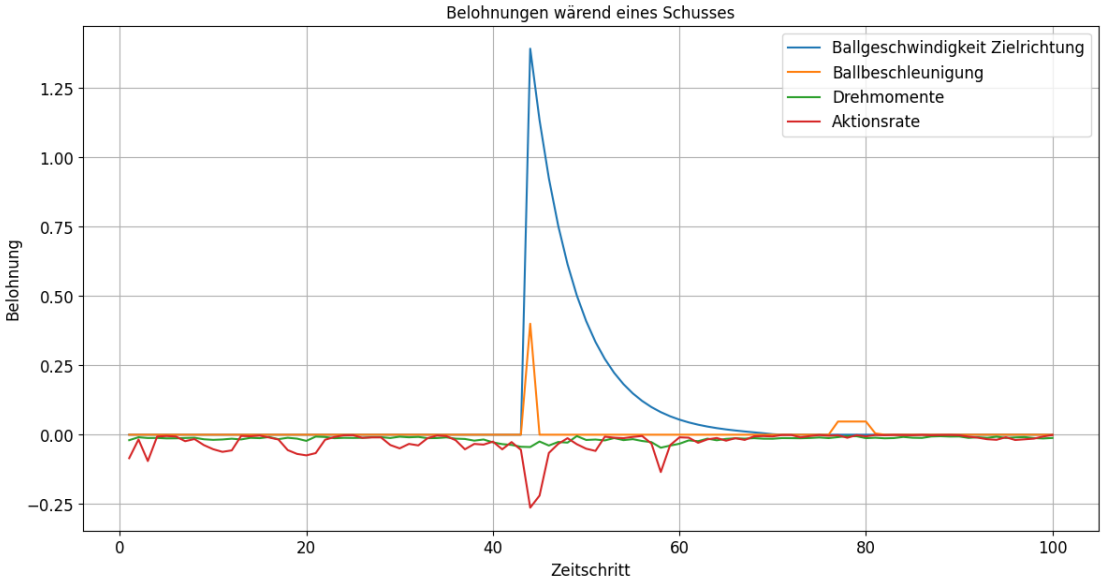


Abbildung 3.5: Verlauf von ausgewählten Belohnungen während eines Schusses. Das starke Ansteigen der Belohnung für die Ballbeschleunigung in Zielrichtung markiert den Zeitpunkt der Ballberührung.

3.3.3 Abbruchbedingungen

Wenn Abbruchbedingungen (siehe Tabelle A.5) eintreten, wird eine Umgebung zurückgesetzt und beendet eine kontinuierliche Sammlung von Beobachtungen und Belohnungen. Sie entscheiden, welche Erfahrungen und Belohnungen für den Agent zugänglich sind. Die Bedingungen sollten so gewählt werden, dass sie eine Abgrenzung zu Verhalten bilden, welches nicht mitgelernt werden soll. So ist das Aufstehen nach einem Fall nicht Teil des Schusstrainings.

Um dementsprechend eine Abgrenzung von dem Fallen zu ermöglichen, gibt es zwei Abbruchbedingungen, die bei einem Fall eintreten. Erstens, wenn der Oberkörper eine gewisse Höhe unterschreitet und zweitens, wenn die Geschwindigkeit des Oberkörpers zu groß wird.

Zum Abbruch kommt es auch, wenn der Ball zu lange nicht berührt wurde, oder eine gewisse Geschwindigkeit nach dem Schießen unterschreitet. Diese dienen dazu, den Agent von Belohnungen auszuschließen, die lediglich durch langes Stehen zustande kommen.

Eine weitere Bedingung bricht ab, wenn der Ball lange genug gerollt ist. Das entspricht einem Abbruch im Erfolgsfall und sorgt dafür, dass der Agent von vorn beginnen kann. Die letzte Bedingung bricht ab, wenn der Durchlauf zu lange dauert und keine andere Abbruchbedingung eintritt. Dies soll endlose Durchläufe unterbinden.

3.4 Sim-to-Real

Das finale Ziel ist, dass die gelernte Strategie auch auf einen echten T1 Roboter läuft. Dies ist allerdings nicht ohne Weiteres möglich. Sowohl das Transfersetup muss stimmen als auch der gesamte Trainings- und Simulationsprozess. Im Folgenden soll die Software beschrieben werden, welche die in der Simulation gelernte Strategie ausführt.

3.4.1 Set-up auf dem Roboter

Ausgeführt wird die Strategie auf dem Intel Rechner des T1, da dieser ausreichend Leistung besitzt, um das Actor-Modell auszuführen. Wichtig ist das Timing beim Ausführen der Strategie und beim Übergeben der Motorsignale. Die Strategie wird wie im Training alle 0,02 s ausgeführt und die Motorsignale werden im gleichen Zeitintervall wie der Simulationsschritt von 0,002 s übergeben. Dabei werden diese gedämpft, indem immer nur 20% des neuen Motorsignals übergeben werden:

$$\mathbf{q}_t^{\text{filtered}} = 0.8 \cdot \mathbf{q}_{t-1}^{\text{filtered}} + 0.2 \cdot \mathbf{q}_t^{\text{target}}$$

Hierbei ist $\mathbf{q}_t^{\text{filtered}}$ das geglättete Motorsignal zum Zeitpunkt t, $\mathbf{q}_{t-1}^{\text{filtered}}$ das vorherige geglättete Signal und $\mathbf{q}_t^{\text{target}}$ das aktuelle Zielsignal aus der Strategie. Durch diese rekursive Mischung wird erreicht, dass sprunghafte Änderungen im Motorsignal abgefedert werden und die Bewegungen des Roboters glatter und stabiler verlaufen.

Mit diesem Set-up können die Strategien auf dem echten Roboter getestet werden. Dabei müssen weitere Anpassungen an der Strategie vorgenommen werden, bevor ein stabiles Schussverhalten auf dem echten Roboter laufen kann.

3.4.2 Sim-to-Real-Transfer

Trotz vorbeugender Maßnahmen wie Randomization hat sich der Transfer der Strategie auf den echten Roboter als eine Herausforderung erwiesen. Eines der größten Probleme war ein Zittern im ganzen Körper, was zu hektischen Bewegungen geführt hat. Dadurch konnte der Roboter nicht stabil stehen und die ruckartigen Bewegungen waren sowohl für den Roboter als auch für den Menschen gefährlich. Ein weiteres Problem war, dass der Roboter vor jedem Schuss sehr lange gewartet hat bzw. gar nicht geschossen hat. Beide Probleme konnten mit Finetuning behoben werden. Dabei wird die bereits existierende Strategie weitertrainiert mit veränderten Parametern wie den Belohnungsskalierungen.

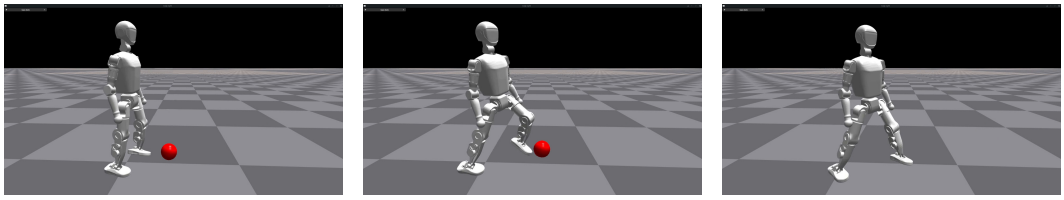
Um das Zittern zu verhindern, wurden diverse Bestrafungen erhöht die eine ruhige Bewegung ermöglichen sollen. Das umfasst eine höhere Bestrafung der Aktionsrate, damit schnelle und große Änderungen stärker sanktioniert werden; hohe Drehmomente, die vorwiegend bei ruckartigen Bewegungen auftreten; die Fuß-Yaw-Ausrichtung und den Unterschied zwischen den Füßen; sowie das Gleiten der

Füße über den Boden, um ein stabiles Fußverhalten zu erzwingen. Anschließend wurde die Strategie, die das Problem mit dem Zittern aufwies, weitertrainiert. Dies führte zu ruhigem und stabilem Verhalten.

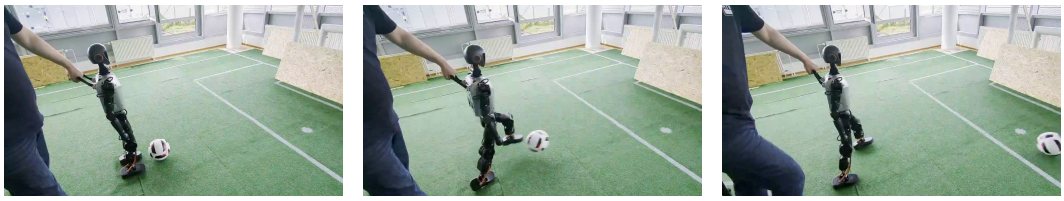
Um das Warten vor dem Schuss zu verhindern, wurde eine zusätzliche Bestrafung eingeführt. Sie steigt mit der Zeit an, in der der Ball nicht berührt wird, und fällt nach einem Schuss sofort weg. Dadurch wird nicht nur schneller geschossen, sondern der Wegfall der Bestrafung dient gleichzeitig als spärliche Belohnung. Mit dieser Anpassung konnte die bereits weitertrainierte Strategie erneut optimiert werden. Anschließend hat der Roboter deutlich schneller den Schuss ausgeführt. Das Ergebnis ist ein Zero-Shot Transfer des Verhaltens aus der Simulation in die echte Umgebung, ein Weitertrainieren mit echten Daten war nicht mehr nötig.

4 Diskussion

Um eine Schussbewegung für den T1 Roboter zu entwickeln, wurde eine Simulation entworfen, die den Roboter und seine Umgebung simuliert. Mit einem angepassten Belohnungsdesign ist es gelungen, einen Trainingsprozess umzusetzen, der in der Lage ist, eine Strategie zu finden, welche sowohl in der Simulation (siehe Abbildung 4.1a) als auch auf den echten Roboter funktioniert (siehe Abbildung 4.1b). Ein weiteres Trainieren mit echten Daten war nicht notwendig.



(a) Simulation



(b) Echter Roboter

Abbildung 4.1: Schuss-Sequenzen in der Simulation (oben) und auf dem echten Roboter (unten). Es ist zu erkennen, dass zum Schießen ein kraftvoller Ausfallschritt genutzt wird.

Dabei war das Einführen von dichten Belohnungen, die jeweils nur in der Phase vor dem Schießen auftreten, ein entscheidender Aspekt. Ausschließlich spärliche Zielbelohnungen führten nur sehr langsam zu einem Schussverhalten, da es nur selten vorkommt, dass der Roboter in der Simulation den Ball zufällig berührt. Die dichten Belohnungen konnten hingegen zuverlässig in Richtung Schussbewegung führen und verhalfen den Agent so schneller zu einem Schussverhalten.

Des Weiteren erwies sich das Finetuning als hilfreiche Maßnahme, um Transferprobleme zu lösen. Angepasst wurden dabei Belohnungen, die entweder neu hinzugefügt oder in der Skalierung verändert wurden. Das Verfahren konnte erfolgreich eingesetzt werden, um Probleme mit starkem Zittern und langem Warten vor dem Schießen zu beheben, die ausschließlich auf dem echten Roboter auftraten.

Bemerkenswert ist, dass die Trainingsparameter und die Modellearchitektur für das Laufen unverändert auch für das Schießen benutzt werden konnten. Das lässt

vermuten, dass diese ebenfalls für das Erlernen von weiteren Bewegungen genutzt werden können.

Die Qualität der Strategie lässt sich sowohl an der Schusshäufigkeit als auch an der erreichten Ballgeschwindigkeit in Zielrichtung messen. Dabei konnte in der Simulation eine Schusshäufigkeit von 84 % mit einer Durchschnittsballgeschwindigkeit von 6,75 m/s in Zielrichtung erreicht werden. Die Testdurchläufe wurden mit den gleichen Abbruchbedingungen und Randomisierungen wie im Training durchgeführt. Tabelle 4.1 fasst die wichtigsten Kennzahlen aus der Simulation zusammen.

Tabelle 4.1: Ergebnisse der Simulation des Schuss-Trainings

Kennzahl	Wert	Einheit
Anzahl Testdurchläufe	20.000	–
Erfolgreiche Schüsse	16.936	–
Umgefallene Roboter	365	–
Durchschnittliche Ballgeschwindigkeit in Zielrichtung	6,75	m/s
Standardabweichung Ballgeschwindigkeit in Zielrichtung	1,06	m/s
Maximale Ballgeschwindigkeit in Zielrichtung	11,55	m/s

Eine ähnliche Auswertung für die Strategie auf dem echten Roboter ist, aufgrund des aufwendigeren Set-ups für das Tracking des Balles, ausstehend.

Die Erkenntnisse dieser Arbeit decken sich mit den bekannten Stärken von Reinforcement Learning für kontinuierliche Ganzkörperprobleme, wie unbegrenzte Daten durch Simulation und das Erlernen komplexer Verhaltensweisen und deren Herausforderungen wie datenintensives Training, Reality Gap und hohe Anforderungen an das Belohnungsdesign [41, 21, 42, 43]. Aus den Erkenntnissen dieser Arbeit lassen sich dementsprechend folgende praktische Leitlinien ableiten:

- Phasen–bewusstes Belohnungsdesign
- Dichte Hilfsbelohnungen, die zum Schuss führen
- Finetuning gegen Transferprobleme

5 Fazit

In dieser Arbeit wurde gezeigt, dass durch eine geeignete Kombination aus realitätsnaher Simulation, gezieltem Belohnungsdesign und phasenbewusstem Training, ein komplexes Ganzkörperverhalten wie das Schießen erfolgreich mit Reinforcement Learning erlernt und auf einen realen T1-Roboter übertragen werden kann. Das zentrale Ziel, eine in der Simulation trainierte Strategie ohne zusätzliches Training mit realen Daten direkt zu übertragen, konnte erreicht werden.

Besondere Herausforderungen wie der Sim-to-Real-Gap traten in Form unerwünschten Verhaltens, wie dem Zittern des Körpers und Warten vor dem Schießen, auf. Diese Schwierigkeiten konnten durch Finetuning in der Simulation und gezielte Anpassungen im Belohnungsdesign erfolgreich entschärft werden. Damit liefert die Arbeit ein praxistaugliches Vorgehen, das Zero-Shot-Deployment trotz unvermeidbarer Diskrepanzen zwischen Simulation und Realität ermöglicht.

Die wesentlichen Beiträge dieser Arbeit lassen sich wie folgt zusammenfassen:

- Entwicklung einer RL-Pipeline für das Schießen auf humanoiden Robotern,
- systematisches Reward-Design als Blaupause für dynamische Bewegungen,
- ein einfaches Sim-to-Real-Rezept mit Finetuning zur Reduktion von Transferproblemen.

Damit konnte gezeigt werden, dass RL ein vielversprechender Ansatz ist, um hochdynamische, kontinuierliche Bewegungen für humanoide Roboter zu realisieren. Das Vorgehen lässt sich neben dem Schuss auch auf anderes Verhalten wie Dribbeln oder Torwartbewegungen anwenden. Somit leistet die Arbeit einen Beitrag zur Weiterentwicklung agiler und anpassungsfähiger Robotersysteme, die langfristig das Spielgeschehen im Roboterfußball auf ein neues Niveau heben können und die Spiele spannender machen.

6 Ausblick

Zentral für die Weiterentwicklung werden das Testen der Strategie auf weiteren T1 Robotern und die Einführung eines Evaluationssystems für den echten Roboter sein. Dies wird die Robustheit der Strategie prüfen. Bis jetzt wurde der Schuss nur isoliert als einzelne Bewegung betrachtet. Zukünftig muss ein Schuss im Spiel selbst funktionieren. Dies sorgt noch einmal für ganz andere Herausforderungen und Ansprüche an Timing, Stabilität und Genauigkeit. Um dem Roboter mehr Kontrolle über den Schuss zu geben, ist es überlegenswert, Parameter einzuführen, die Aspekte des Schusses einstellen. Diese könnten etwa die Schussstärke oder das Schussziel vorgeben, um den Schuss in diversen Situationen passend einzusetzen. Zuletzt soll auch weiteres Verhalten mit RL umgesetzt werden. Erste Experimente mit dem Dribbeln weisen darauf hin, dass dieses Verhalten ebenfalls von RL profitieren kann. Das Ziel soll sein, das gesamte Verhalten mit RL zu lernen. Dabei ist es eine offene Frage, ob einzelne Bewegungen getrennt trainiert und von einer übergeordneten Strategie koordiniert werden oder direkt ein Gesamtverhalten Ende-Zu-Ende gelernt wird.

Literatur

[1] RoboCup Federation. "Objective," RoboCup Federation, besucht am 21. Sep. 2025. Adresse: https://www.robocup.org/objective.

[2] C. S. Lin, P. R. Chang und J. Y.-S. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Transactions on Automatic Control*, Jg. 28, S. 1066–1074, 1983. Adresse: https://api.semanticscholar.org/CorpusID:119783730.

[3] J. Bobrow, S. Dubowsky und J. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, Jg. 4, Nr. 3, S. 3–17, 1985. DOI: 10.1177/027836498500400301. eprint: https://doi.org/10.1177/027836498500400301. Adresse: https://doi.org/10.1177/027836498500400301.

[4] A. Böckmann und T. Laue, "Kick motions for the NAO robot using dynamic movement primitives," in *RoboCup 2016: Robot World Cup XX*, S. Behnke, R. Sheh, S. Sarıel und D. D. Lee, Hrsg., Cham: Springer International Publishing, 2017, S. 33–44, ISBN: 978-3-319-68792-6. DOI: 10.1007/978-3-319-68792-6_3.

[5] Y. Ji, G. B. Margolis und P. Agrawal, *DribbleBot: Dynamic Legged Manipulation in the Wild*, 3. Apr. 2023. DOI: 10.48550/arXiv.2304.01159. arXiv: 2304.01159[cs]. besucht am 3. Okt. 2024. Adresse: http://arxiv.org/abs/2304.01159.

[6] X. Huang u. a., *Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning*, 10. Okt. 2022. DOI: 10.48550/arXiv.2210.04435. arXiv: 2210.04435[cs]. besucht am 12. Sep. 2025. Adresse: http://arxiv.org/abs/2210.04435.

[7] Y. Ji u. a., *Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot*, 1. Aug. 2022. DOI: 10.48550/arXiv.2208.01160. arXiv: 2208.01160[cs]. besucht am 17. Apr. 2025. Adresse: http://arxiv.org/abs/2208.01160.

[8] S. Liu u. a., *From Motor Control to Team Play in Simulated Humanoid Football*, 25. Mai 2021. DOI: 10.48550/arXiv.2105.12196. arXiv: 2105.12196[cs]. besucht am 23. Apr. 2025. Adresse: http://arxiv.org/abs/2105.12196.

[9] T. Haarnoja u. a., "Learning agile soccer skills for a bipedal robot with deep reinforcement learning," *Science Robotics*, 10. Apr. 2024, Publisher: American Association for the Advancement of Science. DOI: 10.1126/scirobotics.adi8022. besucht am 3. Okt. 2024. Adresse: https://www.science.org/doi/10.1126/scirobotics.adi8022.

[10] "Booster T1, Made for Developers," Booster Robotics, besucht am 21. Sep. 2025. Adresse: https://www.boosterobotics.com/booster-t1/.

[11] Sutikno, "(PDF) an overview of emerging trends in robotics and automation," *ResearchGate*, 24. Juli 2025. DOI: 10.11591/ijra.v12i4.pp405-411. besucht am 1. Sep. 2025. Adresse: https://www.researchgate.net/publication/379231219_An_overview_of_emerging_trends_in_robotics_and_automation.

[12] H. Choi u. a., "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proceedings of the National Academy of Sciences of the United States of America*, Jg. 118, Nr. 1, e1907856118, 28. Dez. 2020. DOI: 10.1073/pnas.1907856118. besucht am 1. Sep. 2025. Adresse: https://pmc.ncbi.nlm.nih.gov/articles/PMC7817170/.

[13] M. Guo, Y. Jiang, A. E. Spielberg, J. Wu und K. Liu, "Benchmarking rigid body contact models," in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, ISSN: 2640-3498, PMLR, 6. Juni 2023, S. 1480–1492. besucht am 20. Sep. 2025. Adresse: https://proceedings.mlr.press/v211/guo23b.html.

[14] Pixar Animation Studios, *OpenUSD Documentation (Universal Scene Description)*, https://openusd.org/docs/, Zugriff am 20.09.2025, 2025.

[15] Open Robotics, *Unified Robot Description Format (URDF) — XML Spezifikation*, https://wiki.ros.org/urdf/XML/model, Version vom 24.03.2023, Zugriff am 20.09.2025, 2023.

[16] J. Collins, S. Chand, A. Vanderkop und D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, Jg. 9, S. 51 416–51 431, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3068769. besucht am 1. Sep. 2025. Adresse: https://ieeexplore.ieee.org/document/9386154/.

[17] S. Anderson, "NSF/NIST/DOD workshop on using modeling and simulation in robotics: Pre-workshop slides (2018)," 2018.

[18] C. K. Liu und D. Negrut, "The role of physics-based simulators in robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, Jg. 4, S. 35–58, Volume 4, 2021 3. Mai 2021, Publisher: Annual Reviews, ISSN: 2573-5144. DOI: 10.1146/annurev-control-072220-093055. besucht am 1. Sep. 2025. Adresse: https://www.annualreviews.org/content/journals/10.1146/annurev-control-072220-093055.

[19] R. S. Sutton und A. G. Barto, "Reinforcement learning: An introduction,"

[20] P. Ladosz, L. Weng, M. Kim und H. Oh, "Exploration in Deep Reinforcement Learning: A Survey," *Information Fusion*, Jg. 85, S. 1–22, Sep. 2022, ISSN: 15662535. DOI: 10.1016/j.inffus.2022.03.003. arXiv: 2205.00824[cs]. besucht am 19. Sep. 2025. Adresse: http://arxiv.org/abs/2205.00824.

[21] J. Kober, J. A. Bagnell und J. Peters, "Reinforcement learning in robotics: A survey,"

[22] S. Levine, C. Finn, T. Darrell und P. Abbeel, "End-to-end training of deep visuomotor policies,"

[23] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994, ISBN: 978-0471727828.

[24] C. J. C. H. Watkins und P. Dayan, "Q-learning," *Machine Learning*, Jg. 8, Nr. 3–4, S. 279–292, Mai 1992. DOI: 10.1007/BF00992698. Adresse: https://link.springer.com/article/10.1007/BF00992698.

[25] T. P. Lillicrap u. a., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. arXiv: 1509.02971 [cs.LG]. Adresse: https://arxiv.org/abs/1509.02971.

[26] T. Haarnoja u. a., *Soft Actor-Critic Algorithms and Applications*, 29. Jan. 2019. DOI: 10.48550/arXiv.1812.05905. arXiv: 1812.05905[cs]. besucht am 1. Sep. 2025. Adresse: http://arxiv.org/abs/1812.05905.

[27] T. Haarnoja, A. Zhou, P. Abbeel und S. Levine, "Off-policy maximum entropy deep reinforcement learning with a stochastic actor,"

[28] D. Han, B. Mulyana, V. Stankovic und S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, Jg. 23, Nr. 7, S. 3762, Jan. 2023, Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s23073762. besucht am 1. Sep. 2025. Adresse: https://www.mdpi.com/1424-8220/23/7/3762.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, *Proximal Policy Optimization Algorithms*, 28. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. arXiv: 1707.06347[cs]. besucht am 1. Sep. 2025. Adresse: http://arxiv.org/abs/1707.06347.

[30] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba und P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *Robotics: Science and Systems XIV*, Robotics: Science und Systems Foundation, 26. Juni 2018, ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.008. besucht am 8. Sep. 2025. Adresse: http://www.roboticsproceedings.org/rss14/p08.pdf.

[31] OpenAI u. a., *Learning Dexterous In-Hand Manipulation*, 18. Jan. 2019. DOI: 10.48550/arXiv.1808.00177. arXiv: 1808.00177[cs]. besucht am 2. Sep. 2025. Adresse: http://arxiv.org/abs/1808.00177.

[32] M. Andrychowicz u. a., "Learning Dexterous In-Hand Manipulation," *The International Journal of Robotics Research*, Jg. 39, Nr. 1, S. 3–20, 2020. DOI: 10.1177/0278364919887447. Adresse: https://matthiasplappert.com/publications/2020_OpenAI_Dexterous-Manipulation_IJRR.pdf.

[33] N. Rudin, D. Hoeller, P. Reist und M. Hutter, "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning," in *Proceedings of the 5th Conference on Robot Learning (CoRL)*, Ser. Proceedings of Machine Learning Research, Bd. 164, PMLR, 2022, S. 91–100. Adresse: https://proceedings.mlr.press/v164/rudin22a.html.

[34] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger und J. Peters, "Robot Learning From Randomized Simulations: A Review," *Frontiers in Robotics and AI*, Jg. 9, 11. Apr. 2022, Publisher: Frontiers, ISSN: 2296-9144. DOI: 10.3389/frobt.2022.799893. besucht am 3. Okt. 2024. Adresse: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2022.799893/full.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba und P. Abbeel, *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*, 20. März 2017. DOI: 10.48550/arXiv.1703.06907. arXiv: 1703.06907[cs]. besucht am 1. Sep. 2025. Adresse: http://arxiv.org/abs/1703.06907.

[36] J. Collins, D. Howard und J. Leitner, "Quantifying the Reality Gap in Robotic Manipulation Tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, ISSN: 2577-087X, Mai 2019, S. 6706–6712. DOI: 10.1109/ICRA.2019.8793591. besucht am 14. Okt. 2024. Adresse: https://ieeexplore.ieee.org/document/8793591.

[37] M. Tiboni, A. Borboni, F. Vérité, C. Bregoli und C. Amici, "Sensors and Actuation Technologies in Exoskeletons: A Review," *Sensors (Basel, Switzerland)*, Jg. 22, Nr. 3, S. 884, 24. Jan. 2022, ISSN: 1424-8220. DOI: 10.3390/s22030884. besucht am 3. Sep. 2025. Adresse: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8839165/.

[38] Y. Wang, P. Chen, X. Han, F. Wu und M. Zhao, "Booster Gym: An End-to-End Reinforcement Learning Framework for Humanoid Robot Locomotion," *arXiv preprint arXiv:2506.15132*, 2025.

[39] V. Makoviychuk u. a., *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*, 25. Aug. 2021. DOI: 10.48550/arXiv.2108.10470. arXiv: 2108.10470[cs]. besucht am 3. Sep. 2025. Adresse: http://arxiv.org/abs/2108.10470.

[40] L. Biewald, *Experiment Tracking with Weights & Biases*, https://www.wandb.com/, Software available from wandb.com, 2020.

[41] W. Zhao, J. P. Queralta und T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," *CoRR*, Jg. abs/2009.13303, 2020. arXiv: 2009.13303. Adresse: https://arxiv.org/abs/2009.13303.

[42] E. Ratner, D. Hadfield-Menell und A. D. Dragan, "Simplifying Reward Design through Divide-and-Conquer," *CoRR*, Jg. abs/1806.02501, 2018. arXiv: 1806.02501. Adresse: http://arxiv.org/abs/1806.02501.

[43] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín und P. Stone, *Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes*, 2024. arXiv: 2408.03539 [cs.RO]. Adresse: https://arxiv.org/abs/2408.03539.

A Zusatzmaterial

Tabelle A.1: Beobachtungen des Actors (normalisiert und ggf. verrauscht gemäß Konfiguration).

Größe	Dim.	Bezugssystem	Kurzbeschreibung
Projizierte Gravitation $\mathbf{g}_{\text{base}}$	3	Actor	Orientierungs-Surrogat: Gravitationsvektor in Actor-KS; optional Rauschen & Normierung.
Actorkörper-Winkelgeschw. $\boldsymbol{\omega}_{\text{base}}$	3	Actor	Aktuelle Winkelgeschwindigkeit; optional Rauschen & Normierung.
Relative Ballposition (x, y), $\mathbf{p}^{xy}_{\text{ball/base}}$	2	Actor	Ballposition relativ zur Actor (nur x, y); optional Rauschen & Normierung.
Gelenkwinkel-Offset $\mathbf{q} - \mathbf{q}_0$	23	Gelenkraum	Abweichung zu Default-Winkeln; optional Rauschen & Normierung.
Gelenkgeschwindigkeiten $\dot{\mathbf{q}}$	23	Gelenkraum	Aktuelle DOF-Geschwindigkeiten; optional Rauschen & Normierung.
Aktionen $\mathbf{a}$	12	—	Letzte Aktor-Kommandos (bereits auf $\pm a_{\text{clip}}$ begrenzt).

Tabelle A.2: Privilegierte Beobachtungen (nur Critic).

Größe	Dim.	Bezugssystem	Kurzbeschreibung
Actor-COM-Offsets & Massenskalierung	4	—	Randomisierungsgrößen für Actor (COM x, y, z und Masse).
Actor-Linear-geschwindigkeit $\mathbf{v}_{\text{base}}$	3	Actor	Aktuelle lineare Geschwindigkeit; optional Rauschen & Normierung.
Actorhöhe h über Terrain	1	Welt	z-Abstand der Actor zur Terrainhöhe; optional Rauschen & Normierung.
Ball-Linearge-schwindigkeit (x, y)	2	Welt	Ballgeschwindigkeit (nur x, y); roh (ohne Rauschen/Normierung).
Fußposition links (x, y)	2	Welt	Position des linken Fußes in x, y; roh.
Fußposition rechts (x, y)	2	Welt	Position des rechten Fußes in x, y; roh.
Externe Schubkräfte $\mathbf{f}_{\text{push}}$	3	lokaler Actor	Auf die Actor wirkende Kräfte; mit Normierung.
Externe Schubmomente $\boldsymbol{\tau}_{\text{push}}$	3	lokaler Actor	Auf die Actor wirkende Momente; mit Normierung.

Tabelle A.3: Domain Randomization: Verteilungen/Intervalle, Anwendungsmomente und Erläuterungen ($\Delta t = 0.02\,\mathrm{s}$).

Randomisierung	**Formel**	**Anwendung**	**Kurzbeschreibung**
DOF-Startwinkel	$\mathbf{q}_0 \leftarrow \mathbf{q}_{\text{def}} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.05]$	Bei jedem Reset	$\mathbf{q}_0$ sind die initialen Gelenkwinkel (rad), $\mathbf{q}_{\text{def}}$ die Default-Pose; σ ist die Standardabweichung des additiven Gauß-Rauschens.
Basis-Yaw	$\psi_0 \leftarrow 0 + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.1]$	Bei jedem Reset	ψ_0 ist der anfängliche Gierwinkel um die Vertikalachse (rad); σ bestimmt die Streuung.
Basis-Lineargeschwindigkeit x, y	$\mathbf{v}_{0,xy} \leftarrow \mathbf{0} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.1]$	Bei jedem Reset	$\mathbf{v}_{0,xy} \in \mathbb{R}^2$ ist die anfängliche planare Lineargeschwindigkeit (m/s); σ ist die Rauschstärke.
Ball-Startposition relativ vor Roboter	$d_x \sim \mathcal{U}[0.25, 0.4], \quad d_y \sim \mathcal{U}[0, 0.2]$	Bei jedem Reset	d_x (Vorwärts-) und d_y (Seit-)Offset relativ zum Roboter; Platzierung vor dem linken Fuß im Roboter-KS, vertikal auf Bodenniveau plus Ballradius $r = 0{,}05\,\mathrm{m}$.
Aktuationsverzögerung	$\delta \in \{0, \ldots, 9\} \cdot 0.002\,\mathrm{s}$	Bei jedem Reset	δ ist die Verzögerung der Aktionsweitergabe (0–9 Physik-Substeps $\Rightarrow$ 0–0.018 s).

Fortsetzung auf der nächsten Seite

Fortsetzung von Tab. A.3

Randomisierung	**Formel**	**Anwendung**	**Kurzbeschreibung**
Beobachtungsrauschen: Gravitation	$\mathbf{g} \leftarrow \mathbf{g} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.01]$	Jeder Simulationsschritt	$\mathbf{g} \in \mathbb{R}^3$ ist der (projizierte) Gravitationsvektor; σ ist die Standardabweichung des additiven Rauschens.
Beobachtungsrauschen: Basis-Lin./Ang.-Geschw.	$\mathbf{v} \leftarrow \mathbf{v} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.05]$; $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.1]$	Jeder Simulationsschritt	$\mathbf{v} \in \mathbb{R}^3$ ist die lineare Geschwindigkeit (m/s), $\boldsymbol{\omega} \in \mathbb{R}^3$ die Winkelgeschwindigkeit (rad/s); σ gibt die Rauschstärke an.
Beobachtungsrauschen: Höhe	$h \leftarrow h + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.02]$	Jeder Simulationsschritt	h ist die Basis-Höhe über Terrain (m); σ ist die Standardabweichung des Rauschens.
Beobachtungsrauschen: Ball-Relativposition (2D)	$\mathbf{p}^{xy} \leftarrow \mathbf{p}^{xy} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.01]$	Jeder Simulationsschritt	$\mathbf{p}^{xy} = (x, y)$ ist die Ballposition relativ zum Roboter in der Ebene (m); σ bestimmt die Rauschstärke.
Beobachtungsrauschen: DOF-Offsets/Geschw.	$\Delta\mathbf{q} \leftarrow \Delta\mathbf{q} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.01]$; $\dot{\mathbf{q}} \leftarrow \dot{\mathbf{q}} + \mathcal{N}(0, \sigma^2),\ \sigma \in [0, 0.1]$	Jeder Simulationsschritt	$\Delta\mathbf{q}$ sind Gelenk-Offsets zur Default-Pose (rad), $\dot{\mathbf{q}}$ Gelenkgeschwindigkeiten (rad/s); σ sind die jeweiligen Rauschstärken.
PD-Gains (pro DOF)	$K_p \leftarrow K_p \cdot s,\ s \sim \mathcal{U}[0.95, 1.05]$; $K_d \leftarrow K_d \cdot s',\ s' \sim \mathcal{U}[0.95, 1.05]$	Bei Initialisierung	K_p Proportional-Steifigkeit, K_d Dämpfung; s, s' sind unabhängige uniforme Skalenfaktoren.

Fortsetzung auf der nächsten Seite

Fortsetzung von Tab. A.3

Randomisierung	**Formel**	**Anwendung**	**Kurzbeschreibung**
DOF-Reibung (pro DOF)	$\mu \leftarrow \mu + u,\ u \sim \mathcal{U}[0, 2.0]$	Bei Initialisierung	μ ist die (trockene) Gelenkreibung (dimensionslos); u ist eine uniforme additive Komponente.
Fußkontakt-Eigenschaften (Shapes)	$\mu \sim \mathcal{U}[0.1, 2.0], \quad c \sim \mathcal{U}[0.5, 1.5], \quad e \sim \mathcal{U}[0.1, 0.9]$	Bei Initialisierung	μ Reibungskoeffizient, c effektive Compliance/Weichheit, e Restitutionskoeffizient (Elastizität beim Stoß).
Basiskörper: Schwerpunkt & Masse	$\mathbf{c} \leftarrow \mathbf{c} + \mathcal{U}([-0.1, 0.1]^3); \quad m \leftarrow m \cdot s,\ s \sim \mathcal{U}[0.8, 1.2]$	Bei Initialisierung	$\mathbf{c} \in \mathbb{R}^3$ Schwerpunktlage (m) des Basiskörpers; m dessen Masse; s positiver Skalierungsfaktor.
Andere Körper: Schwerpunkt & Masse	$\mathbf{c}_i \leftarrow \mathbf{c}_i + \mathcal{U}([-0.005, 0.005]^3); \quad m_i \leftarrow m_i \cdot s,\ s \sim \mathcal{U}[0.98, 1.02]$	Bei Initialisierung	$\mathbf{c}_i$ (m) und m_i sind Schwerpunkt und Masse jedes nichtbasalen Körpers; kleinere Amplituden als beim Basiskörper.

Tabelle A.4: Belohnungsterme mit Skalierung der Finetuningiterationen (FT). Negative Skalen wirken als Bestrafungen. „—“ = nicht vorhanden. Werte in Klammern entsprechen der Skalierung in Phase 2 (Schuss)

Name	**Formel**	**Base**	**FT Zittern**	**FT Warten**	**Beschreibung**
Ballgeschwindigkeit in Zielrichtung	$r = \mathrm{clip}\big((\mathbf{v}_B \cdot \hat{\mathbf{d}})\, e^{-t_{\mathrm{roll}}/\tau},\, 0,\, v_{\max}\big)$	10	5	10	$\mathbf{v}_B$: Ballgeschwindigkeit; $\hat{\mathbf{d}}$: Einheitsvektor Ball→Ziel; t_{roll}: Zeit seit Rollbeginn; τ: Zerfallszeit; $v_{\max}$: Kappung. Hauptterm für „starken Schuss“.
Ballbeschleunigung in Zielrichtung	$r = r_{\max} \tanh\big(\max(0, a_x - \lvert a_y \rvert)/s\big)$	0.25	0.15	0.25	a_x, a_y: Ballbeschleunigungs-Komponenten (vorwärts/lateral); s: Skala; $r_{\max}$: Kappung.
Annäherung Fuß–Ball (Ball ruht)	$r = \mathrm{clip}\big(\exp(-d_{FB}/\sigma_{\mathrm{prox}}), 0, r_{\max}\big)$	10 (0)	10 (0)	10 (0)	d_{FB}: kleinster Fuß–Ball-Abstand; σ_{prox}: Nähe-Skala; $r_{\max}$: Obergrenze. Inaktiv in Schussphase
Körperausrichtung für Schuss	$r = \mathrm{clip}\big(\exp\big((\cos\theta_{\mathrm{ziel}} - 1)/\sigma\big), 0, r_{\max}\big)$	1	1	1	θ_{ziel}: Winkel zw. Vorwärtsrichtung und Ziel; σ: Empfindlichkeit; $r_{\max}$: Kappung.
Überleben	$r = 1$	1	0.5	0.25 (0.5)	Konstante Belohnung pro Schritt.

Fortsetzung auf der nächsten Seite

Name	Formel	Base	FT Zittern	FT Warten	Beschreibung
Warten	$r = \left(\frac{t}{T}\right)^2$	—	—	-1 (0)	t: vergangene Zeit; T: Referenz (Skalierung durch „Episode-Progress-Faktor“). Inaktive während Schussphase.
Tracking lin. Geschw. (x)	$r = \exp\big(-v_x^2/\sigma\big)$	1	1	1	v_x: Vorwärtsgeschwindigkeit des Oberkörpers; σ: Breite der Gauß-Glättung. Belohnt kleine $\lvert v_x\rvert$.
Tracking lin. Geschw. (y)	$r = \exp\big(-v_y^2/\sigma\big)$	1	1	1	v_y: Seitwärtsgeschwindigkeit des Oberkörpers; σ: Breite der Gauß-Glättung.
Tracking rot. Geschw. (yaw)	$r = \exp\big(-\omega_z^2/\sigma\big)$	0.25	0.25	0.25	ω_z: Yaw-Winkelgeschwindigkeit des Oberkörpers; σ: Breite. Belohnt kleine $\lvert\omega_z\rvert$.
Oberkörperhöhe	$r = (h - h^\star)^2$	-200	-200	-200	h: Oberkörperhöhe; $h^\star = 0{,}68\,\mathrm{m}$ Zielhöhe. Quadratische Abweichungsstrafe.
Orientierung (Kippung)	$r = g_x^2 + g_y^2$	-20	-20	-20	g_x, g_y: Anteile des Gravitationsvektors im Körper-KS (x/y). Bestraft Schräglage.

Fortsetzung auf der nächsten Seite

Name	**Formel**	**Base**	**FT Zittern**	**FT Warten**	**Beschreibung**
Drehmomente	$r = \sum_i \tau_i^2$	$-1{\cdot}10^{-4}$	$-3{\cdot}10^{-4}$	$-2{\cdot}10^{-4}$	τ_i: Motordrehmoment im DOF i; glättet energiereiche Befehle.
Drehmoment-„Müdigkeit“	$r = \sum_i \min\left((\lvert\tau_i\rvert/\tau_i^{\max})^2,\, 1\right)$	-0.01	-0.01	-0.01	$\tau_i^{\max}$: zulässiges Motormaximum; bestraft Nähe zum Limit (Sättigung bei 1).
Leistung	$r = \sum_i \max(\tau_i\, \dot{q}_i,\, 0)$	-0.001 (-0.002)	-0.002	-0.002	$\dot{q}_i$: Gelenkgeschwindigkeit. Bestraft positive (eingespeiste) Leistung.
Lin. Geschw. z	$r = v_z^2$	-1.5	-1.5	-1.5	v_z: Vertikalgeschwindigkeit des Oberkörpers.
Rot. Geschw. x, y	$r = \omega_x^2 + \omega_y^2$	-0.1	-0.1	-0.1	ω_x, ω_y: Roll-/Pitch-Winkelgeschwindigkeit.
Gelenkgeschwindigkeit	$r = \sum_i \dot{q}_i^2$	$-3{\cdot}10^{-4}$	$-3{\cdot}10^{-4}$	$-3{\cdot}10^{-4}$	$\dot{q}_i$: DOF-Geschwindigkeit; dämpft schnelle Bewegungen.
Gelenkbeschleunigung	$r = \sum_i \left((\dot{q}_i - \dot{q}_i^{\text{alt}})/\Delta t\right)^2$	$-1{\cdot}10^{-7}$	$-1{\cdot}10^{-7}$	$-1{\cdot}10^{-7}$	$\dot{q}_i^{\text{alt}}$: Vorwert; Δt: Zeitschritt. Dämpft Ruck.
Basis-Beschleunigung	$r = \left\lVert(\mathbf{v}^{\text{base}} - \mathbf{v}^{\text{alt}})/\Delta t\right\rVert^2$	$-1{\cdot}10^{-5}$	$-1{\cdot}10^{-5}$	$-1{\cdot}10^{-5}$	$\mathbf{v}^{\text{base}}$: lineare Oberkörper-Geschwindigkeit; $\mathbf{v}^{\text{alt}}$: Vorwert.
Aktionsrate	$r = \sum(a_t - a_{t-1})^2$	-0.25 (-0.5)	-1.5 (-0.5)	-1.5	a_t: Aktionsvektor im Schritt t; glättet Kommandos.

Fortsetzung auf der nächsten Seite

Name	Formel	Base	FT Zittern	FT Warten	Beschreibung
Gelenkpositions-Limits	$r = \sum \mathbb{1}\{\lvert q_i\rvert > \text{Limit}\}$	-1	-1	-1	q_i: Gelenkwinkel; Indikatorstrafe bei Überschreitung weicher Grenzen.
Gelenkgeschw.-Limits	$r = \sum \max(\lvert\dot{q}_i\rvert - \dot{q}_i^{\max}, 0)$	0	0	0	$\dot{q}_i^{\max}$: erlaubte DOF-Geschwindigkeit; hier inaktiv.
Drehmoment-Limits	$r = \sum \max(\lvert\tau_i\rvert - \tau_i^{\max}, 0)$	0	0	0	$\tau_i^{\max}$: Drehmomentgrenze; hier inaktiv.
Kollisionen	$r = \sum \mathbb{1}\{\text{Kontakt verb. Teile}\}$	-1	-1	-1	Indikatorstrafe für Kontakte auf definierten Robotorteilen (Rumpf/Gliedmaßen).
Fuß-Schlupf (bei Kontakt)	$r = \sum \left\lVert \frac{\mathbf{p}_{\text{Fuß}} - \mathbf{p}_{\text{Fuß}}^{\text{alt}}}{\Delta t} \right\rVert^2$	-0.2	-1	-1	$\mathbf{p}_{\text{Fuß}}$: Fußposition; reduziert Gleiten am Boden.
Fuß-Vertikalgeschw.	$r = \sum v_{z,\text{Fuß}}^2$	0	0	0	$v_{z,\text{Fuß}}$: vertikale Fußgeschwindigkeit; hier inaktiv.
Fuß-Rollwinkel	$r = \sum \phi_{\text{roll}}^2$	-0.3	-0.3	-0.3	ϕ_{roll}: Rollwinkel der Füße relativ Boden.
Fuß-Yaw-Differenz	$r = \big(\text{wrap}(\psi_R - \psi_L)\big)^2$	-1	-3	-3	$\psi_{L/R}$: Yaw-Ausrichtung link/rechts; wrap: Winkel in $(-\pi, \pi]$.
Fuß-Yaw-Mittel vs. Körper	$r = \big(\text{wrap}(\psi_{\text{Body}} - \frac{\psi_L + \psi_R}{2})\big)^2$	-1	-3	-3	ψ_{Body}: Yaw des Oberkörpers; bestraft verdrehte Fußstellung.

Fortsetzung auf der nächsten Seite

Name	**Formel**	**Base**	**FT Zittern**	**FT Warten**	**Beschreibung**
Körperwinkel (Pitch/Roll)	$r = \frac{1}{0.1+(\phi^2+\theta^2)^2} - 1$	0.25	0.1	0.1	ϕ: Roll; θ: Pitch des Oberkörpers; kleine Winkel werden belohnt.

Tabelle A.6: Optimizer- und Trainingsparameter

Größe	**Wert**	**Erläuterung**
Optimierungsverfahren	Adam	Standard-Optimizer für stochastische Gradientenverfahren in kontinuierlichen Steueraufgaben.
Lernrate (Start)	1×10^{-5}	Anfangswert für die Schrittweite der Parameteraktualisierung.
LR-Anpassung nach KL-Abweichung	Faktor 1.5 um Ziel-KL 0.01	Erhöht/senkt die Lernrate, wenn die mittlere KL-Divergenz deutlich unter/über dem Ziel liegt (stabilisiert Policy-Updates).
Gradienten-Clipping (global)	1.0 (L2-Norm)	Begrenzung der Gradientenlänge zur Vermeidung numerischer Instabilitäten und Explodierender Gradienten.
Entropie-Gewichtung	-0.01	Negativer Koeffizient im Verlust: fördert höhere Entropie der Policy (explorativere Aktionen).
Begrenzungs-Term für Mittelwerte	aktiv (auf $[-1, 1]$)	Quadratische Strafe, falls die Aktionsmittelwerte den zulässigen Bereich verlassen; verhindert Sättigung außerhalb des Intervalls.
Wertfunktions-Verlust	MSE	Quadratischer Fehler zwischen geschätztem Wert und Ziel-Rückgabe.
Discountfaktor	0.995	Gewichtet zukünftige Belohnungen; nahe 1 für langfristige Ziele.
GAE-Glättung (λ)	0.95	Bias–Varianz-Abwägung in der Vorteilsschätzung (Generalized Advantage Estimation).

Fortsetzung auf der nächsten Seite

Größe	Wert	Erläuterung
Normierung der Vorteile	z-Score	Zentrierung und Skalierung der Vorteile je Batch (Numerikstabiler PPO-Update).
Rollout-Länge	24 Schritte ($\approx 0.48\,\mathrm{s}$)	Anzahl Schritte pro Sammlung vor einem Update; Zeit basierend auf 0.02 s pro Schritt.
Optimierungsdurchläufe pro Update	20 Mini-Epochen	Wie oft über denselben Rollout-Batch iteriert wird (Datenwiederverwendung).
Speicherintervall	alle 100 Updates	Periodisches Sichern von Checkpoints/Logs.
Maximale Updates	10,000	Obergrenze für die Anzahl Training-Iterationen.

Tabelle A.7: Simulationsparameter für den Schuss-Task in Isaac Gym.

Parameter	Wert	Beschreibung
Physik-Zeitschritt (pro Substep)	0.002 s	Zeitinkrement, mit dem *PhysX* die Dynamik integriert.
Aktualisierungsintervall der Aktionen	10 Substeps	So lange wird eine vom Actor gelieferte Aktion gehalten, bevor die nächste an die Tensor-API/PhysX weitergegeben wird (Action-Haltezeit).
Abgeleitet: Zeit pro RL-Schritt	0.02 s	Zeit zwischen zwei Policy-Aktionen (Aktualisierungsintervall × Physik-Zeitschritt).
Schwerkraft (Welt)	$(0, 0, -9.81)\,\mathrm{m/s^2}$	Gravitationsbeschleunigung; Z-Achse zeigt nach oben.
Physik-Engine / Solver	PhysX (TGS)	GPU-basierte Kollision und Dynamik; TGS-Solver für stabile Kontakte.
Solver-Iterationen	Pos: 4, Vel: 1	Iterationszahlen zur Auflösung von Positions- bzw. Geschwindigkeitsnebenbedingungen je Schritt.

Fortsetzung auf der nächsten Seite

Parameter	**Wert**	**Beschreibung**
Kontaktparameter	Kontakt-Offset 0.02 m, Rest-Offset 0 m	Einstellungen für die Kontaktentstehung bzw. das „Rasten“ von Kontakten.
Gelenkregelung (PD)	$K_p = \{200, 200, 50\}$, $K_d = \{5, 5, 1\}$	Proportional- und Dämpfungsanteile für Hüfte/Knie/Sprunggelenk; bestimmen die resultierenden Stellmomente.
Skalierung der Aktionen	1.0	Faktor, mit dem die vom Actor ausgegebenen Kommandos in Stellgrößen/Drehmomente umgesetzt werden.
Bodenmodell (Reibung/Elastizität)	$\mu = 1.0$, $e = 0$	Reibung und Rückprallelastizität der ebenen Umgebung.
Parallelisierte Umgebungen	4096	Anzahl gleichzeitig simulierter, physikalisch getrennter Umgebungen (kein Austausch zwischen ihnen).
Beobachtungen des Actors	44	Länge des Beobachtungsvektors, der der Lernumgebung pro Schritt bereitgestellt wird.
Zusatzbeobachtungen (nur Wertfunktion)	20	Privilegierte Größen, die ausschließlich der Critic verwendet.
Aktionsraum (kontinuierlich)	12	Anzahl kontinuierlicher Stellgrößen, die der Actor ausgibt.
Startpose des Roboters	Höhe ≈ 0.72 m	Ausgangslage des Basiskörpers; weitere Startgrößen werden beim *Zurücksetzen einer Umgebung* zufällig erzeugt (Domain Randomization).

Tabelle A.5: Abbruchbedingungen mit formaler Definition und Kurzbeschreibung ($\Delta t = 0.02\,\mathrm{s}$).

Bedingung	Formel	Kurzbeschreibung
Sturz / niedrige Rumpfhöhe	$h := z_{\text{base}} - h_{\text{terrain}} < 0.45\,\mathrm{m}$	Abbruch, wenn die Rumpfhöhe h unter 0,45 m fällt. Dabei ist z_{base} die Höhe des Actor-Basiskorpers (Oberkörper) in Welt-z und h_{terrain} die Terrainoberfläche an der Standposition.
Zu hohe Basisgeschwindigkeit	$s := \|\mathbf{v}_{\text{base}}\|^2 + \|\boldsymbol{\omega}_{\text{base}}\|^2 > 50$	Abbruch, wenn das kombinierte Geschwindigkeitsmaß s den Grenzwert 50 überschreitet. Hierbei ist $\mathbf{v}_{\text{base}}$ die lineare Geschwindigkeit des Oberkörpers und $\boldsymbol{\omega}_{\text{base}}$ dessen Winkelgeschwindigkeit.
Erfolgsabbruch: Ball rollt lang genug	$\forall \tau \in [0, 2\,\mathrm{s}]$: $v_{\text{ball},x}(t-\tau) > 0.1\,\mathrm{m/s}$	Abbruch im Erfolgsfall: Die Ballgeschwindigkeit in x-Richtung $v_{\text{ball},x}$ bleibt durchgängig länger als 2 s über 0,1 m/s. t ist die aktuelle Zeit; dies entspricht ca. 100 Steps bei $\Delta t = 0.02\,\mathrm{s}$.
Ball zu lange still	$\forall \tau \in [0, 2\,\mathrm{s}]$: $\|\mathbf{v}_{\text{ball}}(t-\tau)\| \leq 0.1\,\mathrm{m/s}$	Abbruch, wenn der Ball mindestens 2 s praktisch stillsteht. $\mathbf{v}_{\text{ball}} \in \mathbb{R}^3$ ist die Ballgeschwindigkeit in Weltkoordinaten, und 0,1 m/s definiert den „still“-Schwellenwert ($\approx$ 100 Steps).
Ball zu lange in Bewegung	$\forall \tau \in [0, 5\,\mathrm{s}]$: $\|\mathbf{v}_{\text{ball}}(t-\tau)\| > 0.1\,\mathrm{m/s}$	Abbruch, wenn der Ball länger als 5 s rollt. $\mathbf{v}_{\text{ball}}$ wie oben; 0,1 m/s ist der Bewegungs-Schwellenwert ($\approx$ 250 Steps).
Episoden-Timeout	$t \geq 7\,\mathrm{s}$ (bzw. $N_{\text{steps}} \geq \lceil 7/\Delta t \rceil = 350$)	Abbruch nach Erreichen der maximalen Episodendauer. t ist die verstrichene Episodenzeit, N_{steps} die Anzahl Simulationsschritte ($\Delta t = 0.02\,\mathrm{s} \Rightarrow 350$ Steps).

Tabelle A.8: Schichtweiser Aufbau der Actor–Critic-Architektur. n_{obs} = Anzahl Beobachtungen, n_{priv} = Anzahl privilegierter Beobachtungen, n_{act} = Anzahl Aktionen.

Netz	Schicht	Typ	Eingabe	Ausgabe	Bemerkung
Critic	1	Linear	$n_{\text{obs}} + n_{\text{priv}}$	256	Eingabe ist $[\,\text{obs}, \text{priv}\,]$
	2	ELU	256	256	
	3	Linear	256	256	
	4	ELU	256	256	
	5	Linear	256	128	
	6	ELU	128	128	
	7	Linear	128	1	Skalarer Wert $V(s)$
Actor	1	Linear	n_{obs}	256	
	2	ELU	256	256	
	3	Linear	256	128	
	4	ELU	128	128	
	5	Linear	128	128	
	6	ELU	128	128	
	7	Linear	128	n_{act}	Ausgabematrix der Mittelwerte $\boldsymbol{\mu} \in \mathbb{R}^{n_{\text{act}}}$
	8	*(Parameter)*	–	n_{act}	Trainierbarer Vektor $\log \boldsymbol{\sigma} \in \mathbb{R}^{n_{\text{act}}}$, initialisiert mit -2.0; $\boldsymbol{\sigma} = \exp(\log \boldsymbol{\sigma})$
	9	*Policy*	–	–	Stochastische Politik: $\pi(\mathbf{a} \mid \mathbf{s}) = \mathcal{N}\big(\boldsymbol{\mu}(\mathbf{s}), \operatorname{diag}(\boldsymbol{\sigma}^2)\big)$

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel „Von der Simulation aufs Spielfeld: Reinforcement Learning für dynamische Schussbewegungen im Roboterfußball“ selbstständig und ohne unerlaubte Hilfe angefertigt habe. Alle verwendeten Quellen sind kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Leipzig, 22. September 2025
Felix Loos