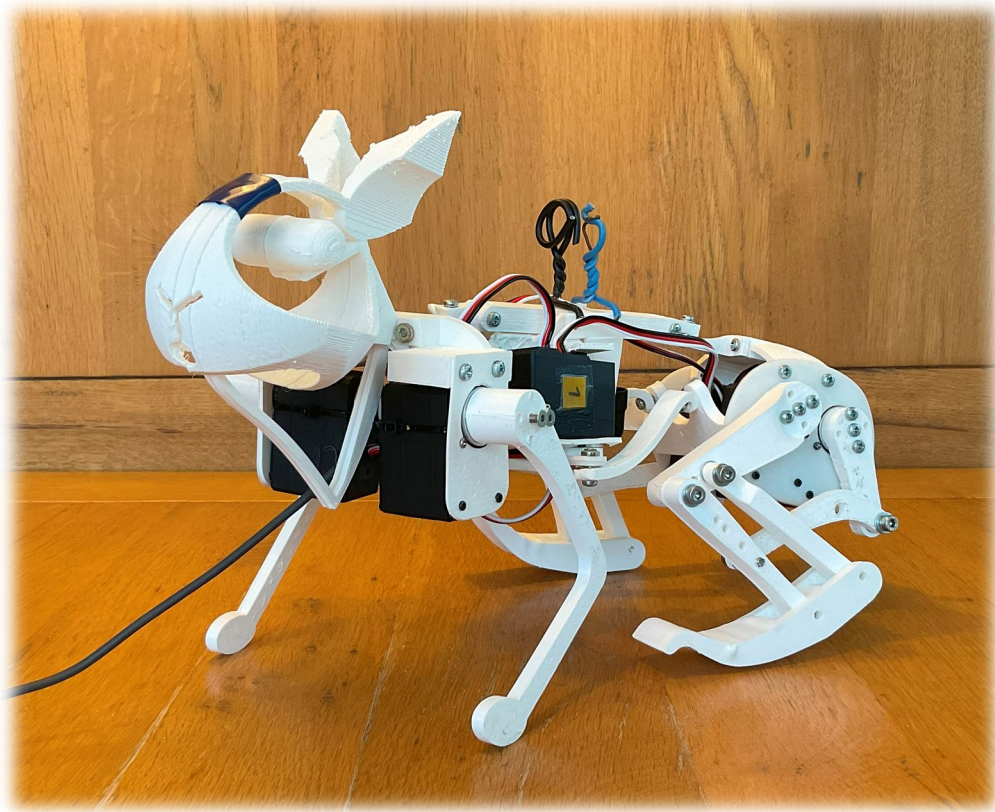


Naturnahe Bewegungsnachbildung

Ein Roboter-Kaninchen lernt das Springen

Konstruktion und maschinelles Lernen



Verfasser:

Kevin Leutwyler

Klasse G21D

Eingereicht bei:

Jonathan Hanselmann

Inhaltsverzeichnis

1	Einleitung.....	1
1.1	Projektziel.....	2
1.2	Fragestellungen.....	2
1.3	Vorgehen.....	2
2	Theoretische Grundlagen.....	3
2.1	Biologie	3
2.1.1	Zwergkaninchen	3
2.2	Robotik	3
2.2.1	Kinematischer Aufbau.....	3
2.2.2	Endeffektoren	4
2.2.3	Freiheitsgrad (DOF).....	4
2.2.4	PID-Regler.....	4
2.2.5	URDF (eng. unified robot description format).....	4
2.3	Maschinelles Lernen.....	5
2.3.1	Reinforcement Learning (RL).....	5
2.3.2	Neuronales Netzwerk	7
2.3.3	Proximal Policy Optimization (PPO).....	8
2.3.4	Soft Actor Critics (SAC)	9
2.3.5	Hyperparameter.....	10
3	Methode und Material	11
3.1	Material	12
3.1.1	Hardware.....	12
3.1.2	Software	12
3.2	Teilprojekt A: Entwurfs- / Realisierungsphase des Roboter Kaninchens	12
3.2.1	Analyse der Anatomie des Zwergkaninchens	13
3.2.2	Erstes Design als Entwurf.....	14
3.2.3	Inverse Kinematik der Hinterläufe	17
3.2.4	Verwendete Technik.....	19
3.2.5	Mechanische Konstruktion.....	22
3.3	Teilprojekt B: Simulation	23
3.3.1	Simulations-Umgebung	23
3.3.2	Experten-Bewegungsabläufe.....	26
3.3.3	Imitations-Reward-Funktion	28
3.3.4	Trainieren von RL-Agenten.....	31
3.4	Teilprojekt C: Anwenden der Experten Bewegungen am realen Roboter.....	34
4	Resultate.....	35
4.1	A) Ergebnis Roboter Kaninchen	35
4.1.1	Erfolgreicher Zusammenbau.....	35
4.1.2	Konstruktive Schwierigkeiten	37
4.2	B) Ergebnisse des Trainings des simulierten Roboter-Kaninchens	38
4.2.1	Vergleich PPO und SAC.....	38
4.2.2	Natürliche Terrains und ihre Auswirkungen auf Bewegungen	39

4.2.3	Fortbewegungsvergleich.....	42
4.3	C) Ergebnisse bei der Steuerung des realen Roboters.....	42
4.3.1	Vergleich zwischen Simulation und realen Umgebung.....	42
4.3.2	Beste Ergebnisse in der realen Welt.....	42
5	Diskussion.....	43
5.1	A) Vergleich mit realen Zwergkaninchen	43
5.1.1	Ähnlichkeiten und Unterschiede.....	43
5.2	B) Erlente Bewegungen.....	44
5.2.1	Vergleich PPO und SAC	44
5.2.2	Einfluss der Umgebung auf die Bewegung des Roboters	44
5.3	C) Unterschiede beim Springen des realen Roboter-Kaninchens	45
5.3.1	Unterschiede zwischen Simulation und Realität.....	45
6	Fazit.....	46
7	Literaturverzeichnis	47
8	Abbildungsverzeichnis	49
9	Anhang.....	I
9.1	Komplette Beschreibung der Reward-Funktion.....	I
9.1.1	Pose Reward rtp :.....	I
9.1.2	Velocity Reward rtv :	I
9.1.3	End-Effector Reward rte :.....	II
9.1.4	Joint Torque Reward rtt :.....	II
9.1.5	Root Pose Reward $rtrp$:.....	II
9.1.6	Root Velocity Reward $rtrv$:.....	II
9.1.7	Action-Reward $rtwa$:	III
9.1.8	Power-Reward $rtpw$:	III
9.2	Hyperparameter.....	III
	Eigenständigkeitserklärung	IV

Abstract

Mit dieser Arbeit ist es gelungen, die Anatomie eines Zwergkaninchens als Roboter nachzubilden. Die Fortbewegung, sowie auch Sprungakrobatik konnte in einer realitätsnahen Simulation mit einem Reinforcement Learning (RL) trainiert werden und Bewegungsabläufe am physischen Roboter getestet werden.

Die Arbeit gliedert sich in drei Teilprojekte: A, B und C. Die Disposition bildet die Grundlage für die Konzeption des Roboter-Kaninchens.

Teilprojekt A umfasst die Entwurfsphase, in der das biologische Vorbild eines Kaninchens analysiert und ein Roboter-Kaninchen konstruiert wird, inklusive dessen mechanischem Aufbau.

In Teilprojekt B wird eine Simulationsumgebung für das Roboter-Kaninchen entwickelt, in der mithilfe von Reinforcement Learning das Vorwärtsbewegen in verschiedenen Terrains trainiert wird. Dabei kommen die bekannten Algorithmen SAC und PPO zum Einsatz.

Teilprojekt C befasst sich schließlich mit dem Testen der in der Simulation erlernten Bewegungsabläufe am realen Roboter-Kaninchen.

In den Resultaten wurden gemäss den grundlegenden Fragestellungen Fortbewegungsstrategien des Roboter-Kaninchen auf den unterschiedlichen Terrains verglichen und der Bewegungsapparat auf seine Stabilität geprüft. Ausserdem zeigen aufgelistete Videos das Roboter-Kaninchen bei der Inbetriebnahme.

Der Roboter konnte erfolgreich nach biologischem Vorbild eines Zwergkaninchens konstruiert und Bewegungen wie das Vorwärtsbewegen konnten erlernt werden. Der Vergleich von PPO- und SAC-Algorithmen zeigte, dass SAC effizienter und besser die natürlichen Bewegungen eines Kaninchens imitierte. Zudem konnten interessante Beobachtungen gemacht werden, beispielsweise dass in Umgebungen mit unterschiedlichem Terrain unterschiedliche Fortbewegungsstrategien erlernt wurden.

Bei der Anwendung der trainierten Bewegungen auf den realen Roboter wurden Unterschiede zwischen der Simulation und Realität gefunden, welche das direkte Übernehmen der Simulationsbewegungen auf die reale Welt erschwerte.

In der Diskussion wird das Endresultat entsprechend den Fragenstellungen besprochen.

Der vollständige Quellcode sowie alle entwickelten Hilfstools sind im folgenden GitHub Repository einsehbar: https://github.com/RouvensLab/Rabbit_Project_v3

Resultate der trainierten RL-Agenten und des realen Roboters wurden als Videos auf dem folgenden GoogleDrive-Link abgespeichert:

<https://drive.google.com/drive/folders/1SwDRL51srnyJ0N2n3uRUVIfwi2vgcFA3?usp=sharing>

Vorwort

In der Natur findet man viele ausgeklügelte Methoden der Fortbewegung. Seit Jahrzehnten inspirieren die Bewegungsstrategien von Tieren die Robotik. So verwundert es nicht, dass auch ich von der Fortbewegung und Sprungtechnik meiner Hauskaninchen zu dieser Maturaarbeit angeregt wurde.

Ich bin jedes Mal aufs Neue erstaunt, wie präzise meine Kaninchen von einem Punkt zum anderen springen können und das mit Sprungdistanzen, die bis zu ihrem 6.5-fachen Körperlänge reichen (1).

Besonders die Sprungweite und die Präzision haben mich dazu angeregt, die technische Umsetzung eines Roboter Kaninchens in Angriff zu nehmen. Mein Ziel war es, ein Robotermodell zu entwickeln, das eine möglichst ähnliche, anatomische Struktur besitzt. Dabei sollte es auch in der Lage sein, die Bewegungen eines Kaninchens durch ein Deep Learning Netzwerk möglichst naturgetreu nachzuahmen.

Ein besonderes Anliegen war es mir, die Schwächen der technischen Nachbildung im Vergleich zum natürlichen Körper durch ein lernfähiges System zu kompensieren. So sollte der Roboter durch maschinelles Lernen in der Lage sein, sich gezielt zu bewegen.

Dieses Projekt hat mich über ein Jahr lang begleitet und mir die Möglichkeit gegeben, meine Fähigkeiten in der Konstruktion, Robotik, Elektronik und dem maschinellen Lernen zu vertiefen. Während ich bereits Erfahrungen im Bau von Robotern aus anderen ähnlichen Projekten gesammelt hatte, stellte dieses Projekt eine neue Herausforderung dar.

Im Speziellen die Durchführung des Robotertrainings mit Reinforcement Learning, war eine meiner ersten vertieften Erfahrungen im Bereich maschinellem Lernen und hat mir neue Perspektiven eröffnet.

Danksagung:

Ein besonderer Dank gilt meinem Maturaarbeit-Betreuer Jonathan Hanselmann für seine wertvolle Unterstützung und Begleitung bei dieser Arbeit. Ebenfalls möchte ich mich bei meinen Eltern bedanken. Sie standen mir in allen emotionalen Höhen und Tiefen zur Seite.

1 Einleitung

Diese Arbeit untersucht maschinelles Lernen, insbesondere Reinforcement Learning, als Teilbereich dieser Technologie.

Der Hintergrund des Projektes liegt in der Herausforderung, dass herkömmliche Roboter stark limitiert sind. Um einem Roboter Flexibilität und Anpassungsfähigkeit zu verleihen, muss er in der Lage sein, selbstständig zu "denken" und auf seine Umgebung zu reagieren.

Traditionelle Steuerungssysteme basieren auf starren, vom Menschen programmierten Abläufen, wodurch die Handlungsmöglichkeiten eines Roboters stark eingeschränkt bleiben.

Da herkömmliche Roboter-Algorithmen nicht in der Lage sind, flexibel auf sich ändernde oder unvorhersehbare Umgebungen zu reagieren, stossen sie schnell an ihre Grenzen.

Auch wenn Programmierer versuchen könnten, die Steuerung durch immer kompliziertere Logik zu verbessern, um auf mehr Situationen vorbereitet zu sein, würde das letztlich zu einem übermässig komplexen und unhandlichen Code führen.

Letztendlich wird **maschinelles Lernen** als Lösung benötigt, um die Flexibilität und Anpassungsfähigkeit der Roboter zu gewährleisten.

1.1 Projektziel

Das Projektziel ist ein Roboter-Modell eines Zwergkaninchens zu konstruieren und ihm mithilfe von Reinforcement Learning das Fortbewegen und Springen beizubringen. Zusätzlich wird angestrebt, die erlernten Vorwärtsbewegungen auf das reale Roboter-Modell zu übertragen, um die Verbindung zwischen virtueller Simulation und realer Anwendung zu erforschen.

1.2 Fragestellungen

Folgende Fragestellungen begleiten dieses Projekt:

- A) Ist es möglich ein Zwergkaninchen anatomisch sehr ähnlich nachzubilden.
- B) Wird ein Reinforcement Learning KI den anatomischen Aufbau des Roboter-Kaninchen ausnutzen und ein ähnliches Bewegungsmuster erlernen, wie das reale Kaninchen? Hat das Terrain der simulierten Umgebung einen Einfluss auf den schlussendlich trainierten Agenten?
- C) Kann das reale Roboter-Kaninchen zu einer Vorwärtsbewegen gebracht werden?

1.3 Vorgehen

Die in der Konzeptphase (Disposition) entwickelte Idee, ein Roboter-Kaninchen zu bauen und zu trainieren, wurde im Teilprojekt A der Entwurfs- und Realisierungsphase weiter ausgearbeitet. Zunächst wurde die Anatomie eines Zwergkaninchens genau ausgemessen, skizziert und entworfen. In der Realisierungsphase entstand aus diesen Entwürfen mithilfe des CAD-Programms Fusion 360 ein 3D-Modell, das anschliessend mit Hilfe eines 3D-Druckers hergestellt wurde.

Im Teilprojekt B wurde das erstellte 3D-Modell in einer Computersimulation mit Reinforcement Learning trainiert und getestet. Dabei lag der Fokus auf dem Erlernen von Sprungbewegungen und der Nachverfolgung eines Ziels.

Die erlernten Bewegungsabläufe konnten im Teilprojekt C dann anschliessend auf dem realen Roboter getestet werden. Die Resultate mit den Grundlegenden Fragestellungen wurden im Anschluss ausführlich diskutiert und Antworten formuliert.

2 Theoretische Grundlagen

Dieses Kapitel stellt Grundlagen vor, die für die Konstruktion und deren Steuerung durch maschinelles Lernen für eine Kaninchnachbildung relevant sind.

Ausserdem werden Fachbegriffe erläutert, die für das Verständnis des Projekts von zentraler Bedeutung sind. Weitere Details können in den referenzierten Quellen nachgelesen werden. Wichtige Begriffe sind in dieser Arbeit **fett** hervorgehoben.

2.1 Biologie

In diesem Abschnitt sind alle Begriffe zur Biologie zusammengefasst, die für diese Arbeit relevant sind.

2.1.1 Zwergkaninchen

Unter Zwergkaninchen wird eine Gruppe kleiner Hauskaninchen verstanden (Gattung aus der Familie der Hasen). Das in diesem Projekt benannte Roboter-Kaninchen ist also eine Nachbildung eines Zwergkaninchen. (2)

2.2 Robotik

In diesem Abschnitt sind alle Begriffe zur Robotik zusammengefasst, die für diese Projekt relevant sind.

2.2.1 Kinematischer Aufbau

Der kinematische Aufbau eines Roboters beschreibt die Anordnung und Anzahl der Achsen, die an der Bewegung beteiligt sind. Es gibt zwei Hauptarten von Gelenken: Rotationsgelenke und Translationsgelenke. Diese werden durch Gelenkkoordinaten und Endeffektoren beschrieben. Gelenkkoordinaten liegen in den Rotationsgelenken. Diese Koordinaten können je nach Perspektive entweder einen relativen Ursprung (**Root-Koordinaten**) oder einen absoluten Ursprung (World-Koordinaten) haben.

2.2.2 Endeffektoren

Endeffektoren sind Werkzeuge wie zum Beispiel ein Greifarm. Auch einfache Strukturen wie Fuss, Unterschenkel oder Oberschenkel können als Endeffektor bezeichnet werden. (3)

2.2.3 Freiheitsgrad (DOF)

Der Freiheitsgrad (DOF, engl. degrees of freedom) beschreibt die Anzahl unabhängiger Bewegungsmöglichkeiten eines Roboters. Ein frei beweglicher Körper im Raum hat sechs Freiheitsgrade. Drei davon beschreiben die Position im Raum (x-,y-,z-Koordinaten), die anderen drei die Orientierung im Raum (Drehwinkel φ_x , φ_y , φ_z). Die Anzahl Freiheitsgrade entspricht daher oft der Anzahl Gelenke eines Roboters. (4)

Zum Beispiel haben die Hinterläufe des Roboter-Kaninchens, die das 2-DOF parallel Roboterarm Konzept brauchen, zwei Freiheitsgrade, da es durch zwei Servomotoren gesteuert wird, welche die Änderung des Endeffektors in x- und y-Richtung verändern können.

2.2.4 PID-Regler

Als Kontrollsysteme werden die integrierten PID-Regler der Servos verwendet. Ein PID-Regler erhält von dem seriellen Bus eine gewünschte Soll-Position. Der PID-Regler vergleicht diese mit der aktuellen Position und regelt anhand der vorgegebenen internen Regel Parameter P (Proportionalanteil), I (Integralanteil), D (Differentialanteil) und den Zeiteinheiten den Servo an die gewünschte Soll Position. Der Regler fährt somit mit einer ihm vorgegeben maximal möglichen Geschwindigkeit und Genauigkeit die Soll Position eigenständig an. (5)

2.2.5 URDF (eng. unified robot description format)

Um einen selbst konstruierten Roboter in eine Physiksimulation zu importieren, muss das 3D-Modell in ein URDF (Unified Robot Description Format) umgewandelt werden. URDF-Fileformate repräsentieren den ganzen Roboter mit seinen Gelenken, Motoren und Sensoren als XML-File. (6)

2.3 Maschinelles Lernen

In diesem Abschnitt sind alle Begriffe zum maschinellen Lernen zusammengefasst, die für dieses Projekt relevant sind.

Maschinelles lernen wurde erstmals in den 1950er Jahren angewendet und ist bis heute ein ständig weiterentwickeltes System des Lernens. Es ist ein Teilbereich der künstlichen Intelligenz (KI) (7) und bezieht sich auf die Fähigkeit, in einer Umgebung vorzuschauen und angemessen zu agieren (8).

Dazu ist es notwendig, relevante Umgebungsdaten wahrzunehmen, zu verarbeiten und zu verstehen (7). Maschinelles Lernen verwendet Lernalgorithmen, um komplexe Probleme zu verstehen und zu lösen, die sich nicht mit einfachen Regeln beschreiben lassen (8).

Maschinelles Lernen lässt sich in weitere Untergebiete unterteilen, die für spezifische Aufgaben jeweils besser geeignet sind als andere.

Um ähnliche Probleme zu lösen, die zum Beispiel Tiere in der Natur antreffen, entwickelte man eine Unterkategorie des maschinellen Lernens, das sogenannte Reinforcement Learning (RL) (9).

2.3.1 Reinforcement Learning (RL)

Reinforcement Learning (RL, deutsch Bestärkendes Lernen) ist eine Methode des maschinellen Lernens, mit der ein Agent zu einer Aufgabe trainiert wird. Dieser Agent erlernt eigene Erfahrungen und Strategien (engl. policy), um eine möglichst grosse Gesamtbelohnung (eng. total discounted reward) über die Zeit zu erzielen. Dem Agenten wird keine optimale Vorgehensweise vorgegeben, sondern sein gesamter Lernprozess basiert ausschliesslich auf seinen eigenen gesammelten Erfahrungen, die er durch Ausprobieren sammelt. RL ist eine Annäherung an das Lernverhalten in der Natur. (9) (10)

Reinforcement Learning bietet viele verschiedene Lernalgorithmen (z.B. SAC, PPO) (10), die je nach Art der Aufgabe und dem angestrebten Ziel optimiert sind. Das grundlegende Schema bleibt jedoch immer gleich.

Theoretische Grundlagen

Das System besteht aus einer Umgebung (eng. environment) und dem Agenten. Der Agent erhält in jedem Zeitabschnitt (eng. time step) einen Zustand (eng. state) der Umgebung. Auf Grundlagen dieser Beobachtung muss sich der Agent für eine Aktion entscheiden, die dann in der Umgebung ausgeführt wird. Die Umgebung gibt dem Agenten durch einen reward (Belohnung) eine Rückmeldung, wie gut seine Aktion war. Dieser Prozess wiederholt sich, bis die Umgebung signalisiert, dass der Agent entweder "gescheitert" ist, also einen grossen Fehler gemacht hat, oder die vorgegebene Zeit (t) abgelaufen ist. Danach wird die Umgebung zurückgesetzt und eine neue **Episode** startet. (10) (9)

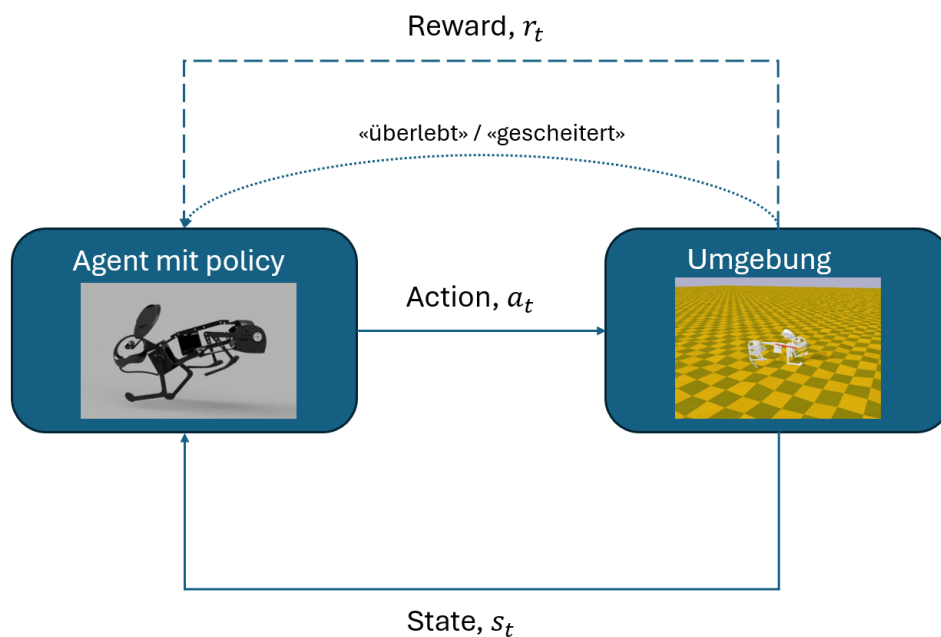


Abb. 1 Illustrierung des Lernprozesses (Eigene Abbildung)

2.3.2 Neuronales Netzwerk

Ein typischer RL-Algorithmus verwendet meist neuronale Netzwerke. Diese Netzwerke bestehen, ähnlich wie unser Gehirn, aus vielen miteinander verbundenen Neuronen. Wenn ein Neuron ein ausreichend starkes Signal erhält, wird es aktiviert und sendet seinerseits ein Signal an weitere Neuronen. (11)

In der Deep Learning-Welt sind diese Neuronen in Schichten (engl. layers) organisiert, wobei Verbindungen zwischen Neuronen ausschliesslich zwischen benachbarten Schichten entstehen. (Siehe Abb. 2).

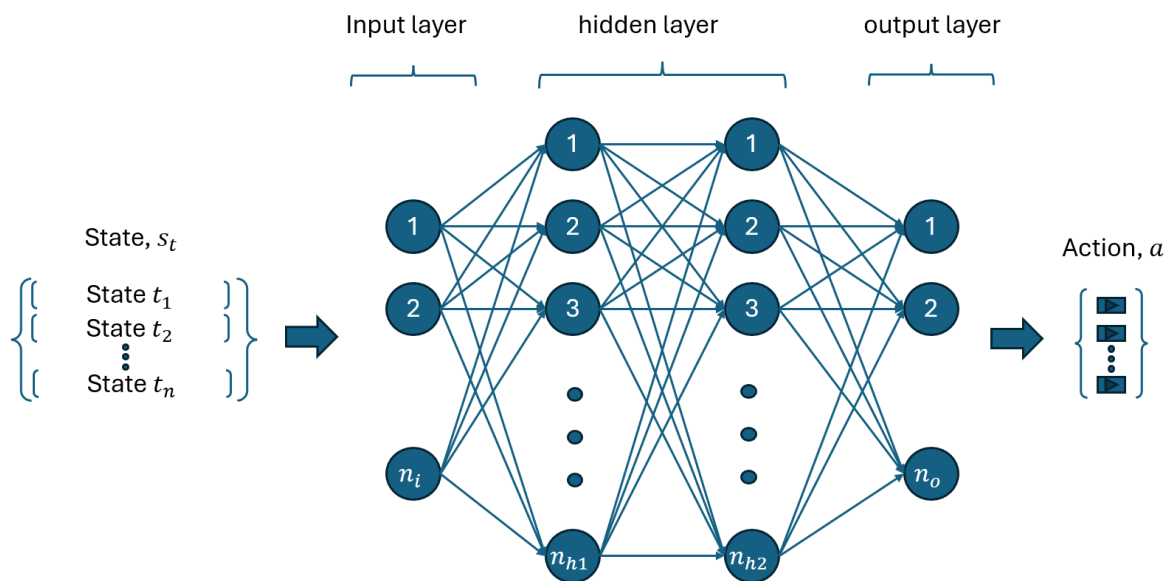


Abb. 2 Illustrierung eines typischen Policy-Netzwerkes. (eigene Abbildung)

Am Anfang und am Ende des neuronalen Netzwerkes befindet sich die Eingabeschicht (engl. input layer) und die Ausgabeschicht (engl. output layer). Der Eingabeschicht werden die States s übergeben und die Ausgabeschicht gibt zum gegebenen State eine Aktion aus. Die inneren Schichten sind die hidden Layers, diese bilden das grundlegende neuronale Netz ab (12).

Meist umfassen die RL-Algorithmen zwei neuronale Netzwerke. Eines dient dazu, die Umgebung zu verstehen und herauszufinden, wie der discounted reward mit dem State zusammenhängt. Und das zweite Netzwerk wird darauf trainiert, Aktionen auszugeben, die eine grosse Belohnung garantieren.

In der RL-Welt gibt es verschiedene Algorithmen, die je nach Aufgabe bevorzugt werden. Solche Entscheidungen werden beispielsweise durch die Effizienz (engl. sample efficiency), die Grösse des Models oder die Aufgabenkomplexität gefällt.

Da z.B. eine Sprungbewegung eines Kaninchens in einer 3-Dimensionalen Umgebung abläuft und eine komplexe Aufgabe darstellt, wurde besonderer Fokus auf die beiden Algorithmen PPO und SAC von der Stablebaselines3 Bibliothek gelegt (10).

2.3.3 Proximal Policy Optimization (PPO)

Der PPO-Algorithmus (eng. proximal policy optimization) ist einer der populärsten und erfolgreichsten On-Policy-RL-Algorithmen.

Das ganze PPO-Konstrukt besteht aus zwei neuronalen Netzwerken der Value-Funktion und Policy Funktion. (13) (14) (15)

2.3.3.1 Value-Funktion

Die Value-Funktion besteht aus einem neuronalen Netzwerk, welches die Umgebung kennen lernt. Es versucht zu verstehen welche States zu guten oder schlechten rewards führen. Das bedeutet, dass das neuronale Netzwerk eine Verbindung zwischen dem erhaltenen Reward und den übergebenen States herstellt. Die Funktion sagt also für einen gegebenen State eine mögliche Belohnung (discounted reward) vorher. (14) (13)

2.3.3.2 Policy-Funktion

Die Policy ist ebenfalls ein neuronales Netzwerk, welches mittels Value-Funktion trainiert wird. Der aktuelle State wird der Policy übergeben, und sie wandelt diesen Zustand über eine Gausssche Verteilung in eine Aktion (engl. action) um.

Die Policy gibt also eine Action aus, die an die Umgebung geschickt wird. Da die Policy darauf trainiert wird, immer den besten vorhergesagten discounted Reward zu erzielen, kann diese zu einseitigem Lernen führen. Das bedeutet, dass die Policy immer den Weg wählt, der nach der Value-Funktion zu einem möglichst hohen Reward führt. (14) (13)

2.3.4 Soft Actor Critics (SAC)

Der SAC-Algorithmus (Soft Action Critics) ist ein Off-Policy-RL-Algorithmus. Was ihn besonders macht, ist seine Lernstrategie, die auf maximaler Entropiestärke basiert. Ein Agent mit grosser Entropie verhält sich chaotisch und unvorhersehbar. Kurz gesagt, versucht der Agent, sich so unvorhersehbar wie möglich zu verhalten. Diese Strategie fördert die Exploration und ermöglicht es dem Agenten, mehr Erfahrungen anzueignen. (16)

Der SAC-Algorithmus lässt sich in zwei Hauptkomponenten unterteilen: den Actor und den Critic.

2.3.4.1 Actor

Der Actor ist ein neuronales Netzwerk, dass durch den Critic bzw. die Q-Funktion trainiert wird. Auf Basis des aktuellen State gibt der Actor eine Wahrscheinlichkeitsverteilung für die möglichen Aktionen aus, die der Umgebung geschickt wird. Im Gegensatz zu deterministischen Methoden wie bei PPO, bei denen eine einzige Aktion gewählt wird, verwendet SAC eine stochastische Policy. Das bedeutet, dass der Actor verschiedene Aktionsmöglichkeiten anbietet, von denen eine zufällig ausgewählt wird. (17)

2.3.4.2 Critic

Der Critic hat eine ähnliche Funktion wie die Value-Funktion, nur dass bei off-policy Methoden von der Q-Funktion gesprochen wird. Der Unterschied liegt darin, dass die Q-Funktion anhand des States und der ausgeführten Aktion vorhersagt, wie gross die Belohnung (Q-Value) sein wird. Die Value-Funktion von PPO hingegen erhält nur den State und sagt die Belohnung vorher, ohne die tatsächlich ausgeführte Aktion zu kennen. (16)

Ein weiteres wichtiges Merkmal von Off-Policy-RL-Algorithmen wie SAC ist die Nutzung eines Replay-Buffers.

2.3.4.3 *Replay Buffer*

Dieser Zwischenspeicher bewahrt gesammelte Erfahrungen über eine bestimmte Zeit auf. Die Grösse des Buffers (eng. buffer size) bestimmt, wie viele Erfahrungen gespeichert werden können.

Eine Erfahrung enthält immer Informationen zum State, der ausgeführten Aktion und der dafür erhaltenen Belohnung. Der Replay-Buffer bringt mehr Stabilität in den Lernprozess, da vergangene Erfahrungen mehrfach genutzt werden können, was dazu beiträgt, dass gute Aktionen weniger schnell vergessen werden (18).

Im Vergleich zum SAC-Algorithmus kann PPO gerade nur die aktuell gelernten Erfahrungen verarbeiten und muss darum ständig neue Daten sammeln. (19) (20) (17) (16)

2.3.5 Hyperparameter

Hyperparameter ist eine Sammlung von übergeordnetem Parameter, welche die Lerngeschwindigkeit und Effizienz eines RL-Algorithmus beeinflussen. (18)

3 Methode und Material

Im Folgenden wird ein grober Überblick über die angewendeten Methoden des Projekts gegeben, der die Arbeitsphasen in drei Teilprojekte unterteilt:

1. Teilprojekt A: Entwurfs- und Realisierungsphase des Kaninchen-Roboters

In diesem Teilprojekt wurde das Roboter-Kaninchen entworfen, konstruiert und schliesslich zusammengebaut.

2. Teilprojekt B: Virtuelles Training und Simulation des 3D-Modells

Das in Teilprojekt A erstellte 3D-Modell des Roboter-Kaninchens wurde in einer Computersimulation mithilfe von Reinforcement Learning (RL) trainiert, um das Vorwärtsbewegen und Springen zu erlernen.

3. Teilprojekt C: Steuerung des realen Roboters

Der zusammengebaute Roboter wurde angesteuert, und die in Teilprojekt B erlernten Bewegungsabläufe wurden am Modell angewandt.

Anschliessend wurden die Ergebnisse aus den Teilprojekten A, B und C analysiert und in der Diskussion im Hinblick auf die Zielsetzungen und Fragestellungen des Projekts ausführlich besprochen.

3.1 Material

Nachfolgend sind die Software und Hardware aufgeführt, die für den Bau des Roboterkaninchens verwendet wurden:

3.1.1 Hardware

- Computer, NVIDIA GeForce GTX 1650 Ti Grafikkarte
- 3D-Drucker daVinci 1.0 Pro von der Firma XYZprinting und PLA-Filament in weiss
- Servo (ST3215 (21)), Servoshield (ESP32 Serial Bus Servo Driver (22))

3.1.2 Software

- Fusion360
- Arduino IDE
- VS-Code als IDE
- Python 3.12 mit nachfolgenden Hauptbibliotheken:

Gymnasium ist eine Open-Source-Python-Bibliothek zum Entwickeln und Vergleichen von Reinforcement-Learning-Algorithmen durch Bereitstellung einer Standard-API für die Kommunikation zwischen Lernalgorithmen und Umgebungen. Ausserdem steht diese Umgebungen zum üben und austesten bereit (23).

StableBaselines3 (SB3) ist eine Reihe zuverlässiger Implementierungen von Reinforcement-Learning-Algorithmen wie auch Tensorboard innerhalb PyTorch (10)

3.2 Teilprojekt A: Entwurfs- / Realisierungsphase des Roboter Kaninchens

In einer Entwurfsphase entstand die Konstruktion wie auch das Design des Roboter-Kaninchens. Durch viele verschiedene iterative Schritte, Tests und Verbesserungen konnte ein mechanischer Aufbau iterativ in einem einfachen Design entworfen werden. Begonnen hat dies mit der Analyse der Anatomie des realen Zwergkaninchens. In diesem Projekt wurde bewusst das Zwergkaninchen als Referenz verwendet.

3.2.1 Analyse der Anatomie des Zwergkaninchens

Unter den Kaninchen gibt es viele verschiedene Arten, die sich stark in ihrer Größe unterscheiden.

Eine Analyse eines Röntgenbildes, unterstützt durch eine angefertigte Skizze (Abb. 3) der Hinterläufe des Kaninchens, bestätigt die anfängliche Vermutung, dass die Längenverhältnisse von Fuß, Unterschenkel und Oberschenkel etwa gleich sind.

Durch Messungen an meinem Zwergkaninchen, insbesondere der Fusslänge, Körperlänge und Körperbreite, konnten die Größen der Unterschenkel und Oberschenkel mit den Verhältnissen berechnet werden.

Die skizzierte Abb. 3 sowie die Tabelle 1 mit den selbst ausgemessenen Zwergkaninchenkörperteile sind die Grundlagen für das anschliessende durchgeführte Design und Konstruktion des Roboter-Kaninchens.

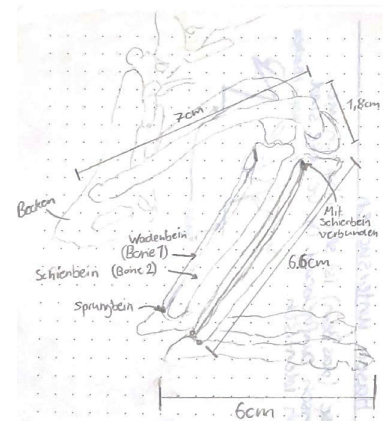


Abb. 3 Skizze des Röntgenbildes mit berechneten Längenangaben.

Körperteil	Grösse
Oberschenkel	70 mm
Unterschenkel	66 mm
Fusslänge (gemessen)	60 mm
Körperlänge (Hinterteil bis Kopf)	340 - 450 mm
Körperhöhe	160 – 180 mm
Gesamtes Körpergewicht	1.25 – 1.4 kg

Tabelle 1 Auflistung der ermittelten Grössen der Körperteile eines Zwergkaninchens.

3.2.2 Erstes Design als Entwurf

Das aktuelle Design wurde in der Entwurfsphasen anhand von bereits zwei Vorgänger Versionen und der ermittelten anatomischen Struktur weiterentwickelt. Somit ist das in diesem Projekt beschriebene Roboter Kaninchen die dritte Generation. In diesem Projekt wird ausschliesslich von der dritten Version ausgegangen.

Der Roboter besteht aus drei Hauptteilen: Unterkörper, Mittelkörper und Oberkörper, die durch eine Wirbelsäule verbunden sind, ähnlich wie bei einem echten Zwergkaninchen (Abb. 4).

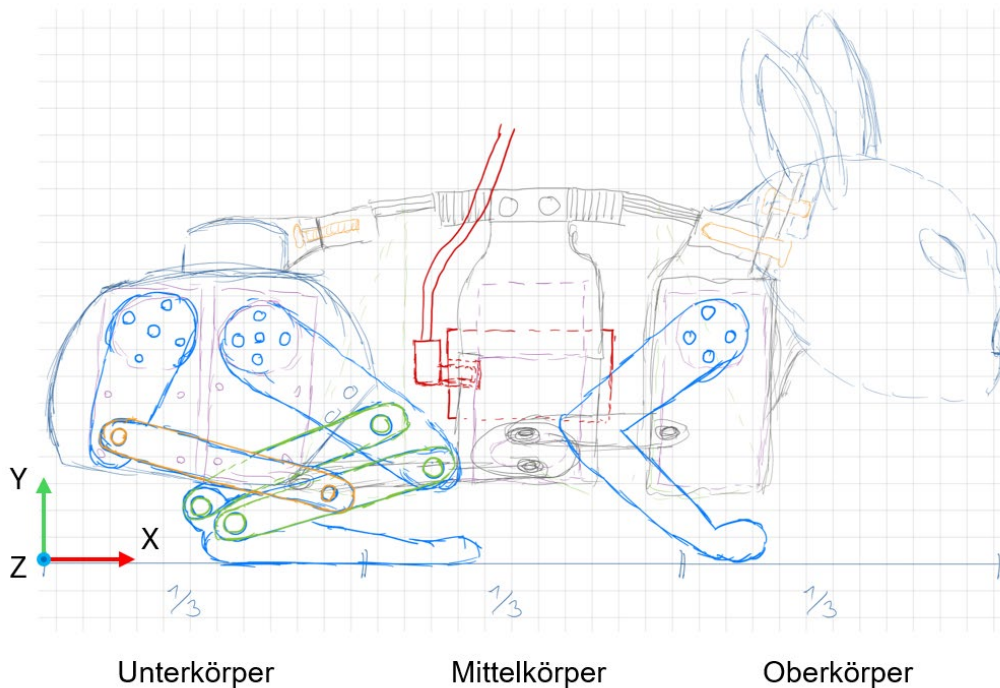


Abb. 4 Skizze der drei Sektionen des Roboter-Kaninchens (Eigene Abbildung)

3.2.2.1 UNTERKÖRPER:

Der Unterkörper beherbergt vier Servomotoren, jeweils zwei auf jeder Seite. An diesen Servo paaren sind die Hinterläufe mit jeweils zwei Freiheitsgraden (2-DOF) angebracht. Das bedeutet, dass sowohl der Oberschenkel als auch der Unterschenkel jeweils mit einem Servo angesteuert werden können. Beim Fuss wurde auf einen Servomotor verzichtet.

Beobachtungen der Sprungbewegungen von Kaninchen zeigen, dass sich der Fuss fast ausschliesslich in Abhängigkeit vom Oberschenkel bewegt.

Daher wurde der Fuss über eine Parallelverbindung mit dem Oberschenkel verbunden (siehe Abb. 5).

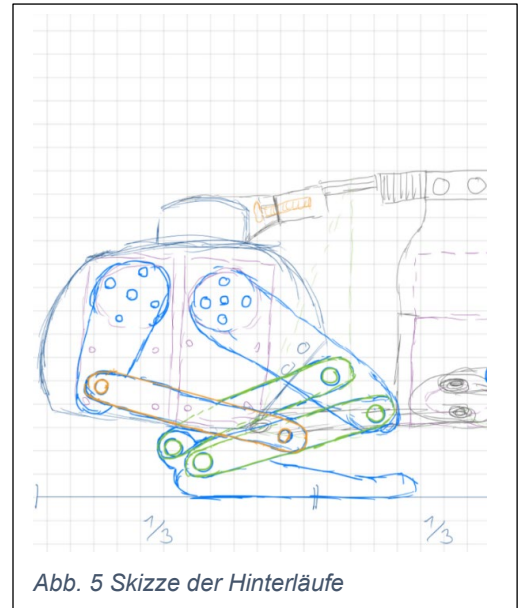


Abb. 5 Skizze der Hinterläufe

3.2.2.2 MITTELKÖRPER:

Der Mittelteil des Körpers basiert auf einer fünfteiligen Wirbelsäule, an deren hinterem Ende der Unterkörper und am vorderen Ende der Oberkörper befestigt ist. In der Mitte der Wirbelsäule ist ein Bauchgestell angebracht, das das ESP32-Servoshield beherbergt. Hier sind auch die beiden Servomotoren untergebracht, die für die Bewegungen von Ober- und Unterkörper zuständig sind. Mit diesen Servos kann die Wirbelsäule gesteuert werden, um den Rücken zu beugen, zu strecken sowie seitlich nach links und rechts zu krümmen.

Die Wirbelsäule hat einen Freiheitsgrad von zwei und ist somit fähig sich um die Z-Achse und die Y-Achse zu krümmen.

Der Root ist beim Mittelstück der Wirbelsäule definiert.

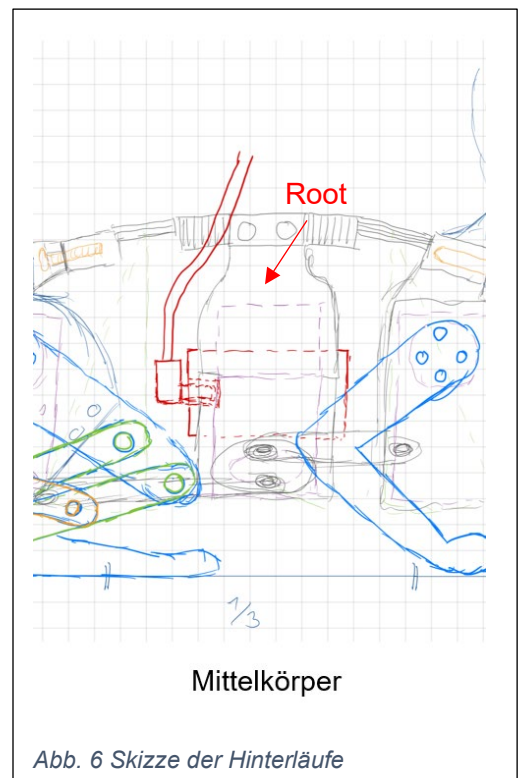
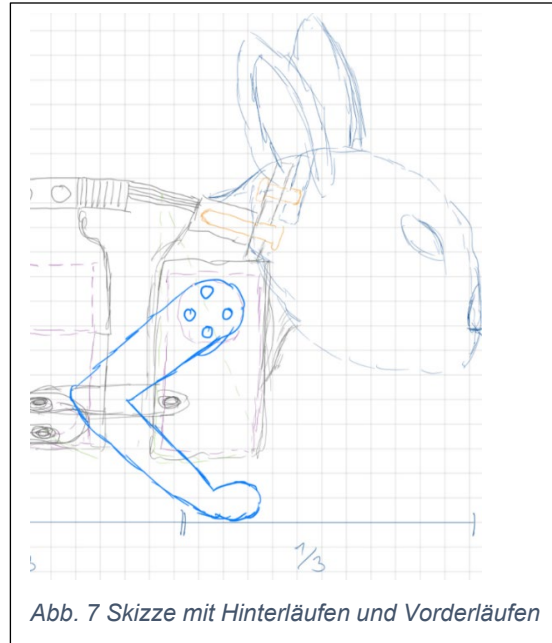


Abb. 6 Skizze der Hinterläufe

3.2.2.3 OBERKÖRPER:

Der Oberkörper des Roboters besteht aus Kopf, Schultern und zwei Servomotoren. Die starren Vorderläufe sind an der Schulterkonstruktion befestigt. Beobachtungen von Kaninchen zeigten, dass auf zwei oder sogar drei Servos für die Vorderläufe verzichtet werden kann. Es wurde jedoch in Betracht gezogen, den Unterschenkel je nach Winkel des Oberschenkels über eine zusätzliche Verbindung zu den Schultern so zu steuern, dass er sich streckt oder zusammenzieht. Idealerweise würden sich die Vorderläufe in der Mitte anziehen und entweder nach vorne oder hinten strecken.



3.2.3 Inverse Kinematik der Hinterläufe

Um die Bewegungen der Hinterläufe effizient zu steuern, wurde ein 2-DOF paralleles Roboterarm-Konzept verwendet. Bei dieser Konstruktion arbeiten zwei Servomotoren, um die komplexen Bewegungen des Oberschenkels und Unterschenkels nachzuahmen. Der erste Motor (Motor 1) steuert die Rotation des Oberschenkels, während der zweite Motor (Motor 2) über ein Hebelsystem die Bewegung des Knies beeinflusst. Da Motor 2 nicht direkt am Kniegelenk angebracht ist, sondern über Hebel agiert, muss die Bewegung präzise koordiniert werden.

Um die Steuerung des Kniewinkels zu vereinfachen und nicht direkt über den Motorwinkel zu arbeiten, ist es notwendig, die Inverse Kinematik zu berechnen. Diese ermöglicht es, den gewünschten Kniewinkel in den entsprechenden Motorwinkel (β) zu übersetzen. In der Simulation wurde das Knie direkt gesteuert. Darum musste der Winkel des zweiten Motors (β) entsprechend berechnet werden, um den gewünschten Kniewinkel (α_2) zu erreichen.

Im Folgenden wird die Herleitung der Inversen Kinematik beschrieben, die auf der Berechnung der Gelenkwinkel und Hebellängen basiert. Die Skizze (Abb. 8) dient der Veranschaulichung.

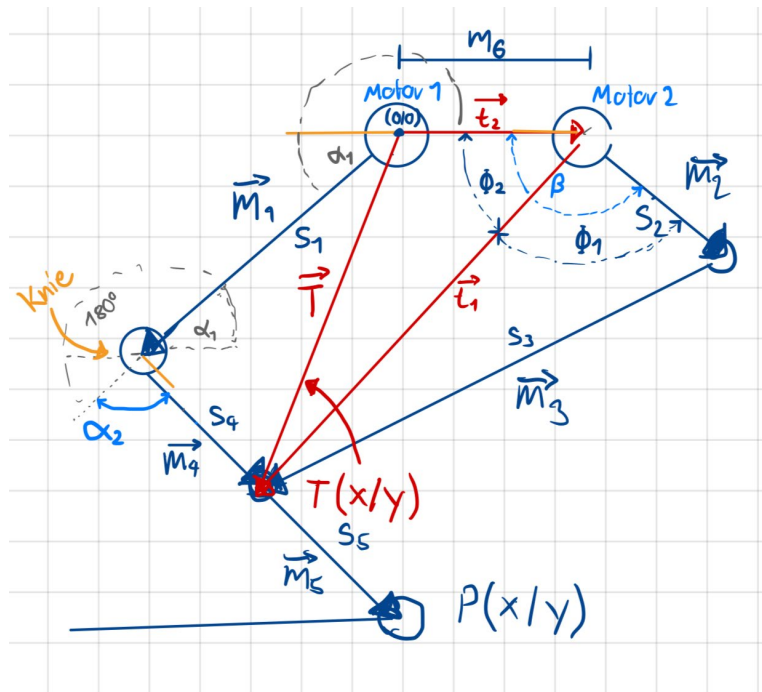


Abb. 8 Skizze des Hinterbeins (eigene Abbildung)

Zuerst werden die Vektoren \vec{m}_1 , \vec{m}_4 und \vec{m}_5 unter der Verwendung der gegebenen Winkel α_1 und α_2 sowie der Gelenklängen s_1 , s_4 , s_5 berechnet.

$$\vec{m}_1 = S_1 \begin{bmatrix} \cos(\alpha_1) \\ \sin(\alpha_1) \end{bmatrix}, \quad \vec{m}_4 = S_4 \begin{bmatrix} \cos(\alpha_1 + \alpha_2) \\ \sin(\alpha_1 + \alpha_2) \end{bmatrix}, \quad \vec{m}_5 = S_5 \begin{bmatrix} \cos(\alpha_1 + \alpha_2) \\ \sin(\alpha_1 + \alpha_2) \end{bmatrix} \quad (3.1)$$

Um das Verhältnis zwischen dem Kniewinkel α_2 und dem Motorwinkel φ_2 zu bestimmen, muss der Vektor \vec{t}_1 aus dem Vektor \vec{t}_2 und Vektor \vec{T} berechnet werden.

$$\vec{t}_1 = \vec{T} - \vec{t}_2 = (\vec{m}_1 + \vec{m}_4) - \vec{t}_2 \quad (3.2)$$

Mit den Längen $t_1 = \|\vec{t}_1\|$ und S_2 und S_3 kann mittels Kosinussatz eine Gleichung aufgestellt werden, die nach ϕ_1 aufgelöst wird. ϕ_1 repräsentiert den Winkel zwischen den Vektoren \vec{t}_1 und \vec{m}_2 .

$$\phi_1 = \cos^{-1} \left(\frac{S_2^2 - S_3^2 + t_1^2}{2S_2t_1} \right) \quad (3.3)$$

Daraufhin wird der Winkel vom Vektor \vec{t}_1 mithilfe des Tangens berechnet, um den verbleibenden Winkel der gesamten Rotation von Motor 2, ϕ_2 , zu ermitteln. Das Resultat ist der Winkel ϕ_2 .

$$\phi_2 = \begin{cases} \tan^{-1} \left(\frac{y}{x} \right), & x > 0 \\ \tan^{-1} \left(\frac{y}{x} \right) + \pi, & x < 0 \text{ and } y \geq 0 \\ \tan^{-1} \left(\frac{y}{x} \right) - \pi, & x < 0 \text{ and } y < 0 \\ \frac{\pi}{2}, & x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2}, & x = 0 \text{ and } y < 0 \end{cases} \quad (3.4)$$

Diese beiden ermittelten Winkel ϕ_1 und ϕ_2 werden addiert, um die vollständige Rotation β beim Motor 2 zu erhalten.

$$\beta = \phi_1 + \phi_2 \quad (3.5)$$

Die Endeffektorkoordinate P(x/y) an der Ferse wird durch Vektoraddition erhalten.

$$P = \vec{m}_1 + \vec{m}_4 + \vec{m}_5 \quad (3.6)$$

3.2.4 Verwendete Technik

Die Untersuchung der Kaninchenanatomie lieferte wichtige Erkenntnisse über die technischen Anforderungen, die erfüllt werden müssen, damit das Roboter-Kaninchen in der Lage ist zu springen.

3.2.4.1 Aktoren

Durch mehrere kleine Testversuche mit unterschiedlichen Bewegungsmechanismen, wie Servomotoren und übersetzten DC-Motoren, wurde grob ermittelt, welche technischen Komponenten sich am besten für eine kompakte Nachbildung des Zwergkaninchens eignen. Aufgrund ihrer Einfachheit und Kompaktheit wurden schliesslich die folgenden Strukturen und Komponenten ausgewählt:

UART Serial Bus Control

Up To 253 Servos Can Be Connected In Series At The Same Time

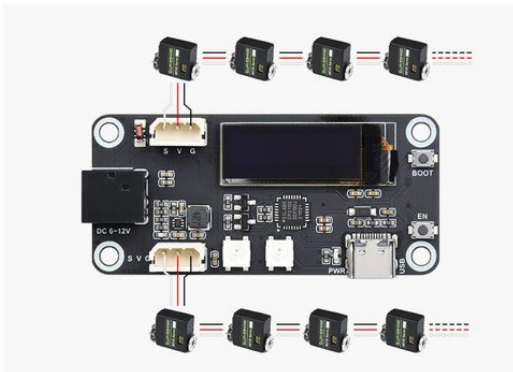


Abb. 9 Gewählte Struktur zur Ansteuerung des Roboters (22)

3.2.4.2 Ansteuerungsstruktur

Alle Servomotoren sind seriell miteinander verbunden. Das ESP32-Servoshield, das über zwei serielle Ausgänge verfügt, ist symmetrisch links und rechts mit jeweils vier Servos verbunden. Dank der seriellen Verbindungen der Servos können bei Bedarf weitere Servos, ohne ein Kabelsalat zu verursachen, hinzugefügt werden. Jeder Servo hat eine eindeutige ID, die auf einem Aufkleber auf dem jeweiligen Servo vermerkt ist.



Stk.	Komponenten	Eigenschaften / Bemerkungen
8	<p><i>Servomotoren ST3215 Servo (21)</i></p> 	<p>Serial servo</p> <p>Drehmoment: 30 kg/cm</p> <p>Firma: Waveshare</p>
1	<p><i>ESP32 Serial Bus Servo Driver (22)</i></p> 	<p>Expansion Board</p> <p>Firma: Waveshare</p>

Tabelle 2 Ausgewählte Komponenten Materialliste (eng. BoM)

Diese elektronischen Komponenten wurden von der Firma Bastelgarage.ch bezogen.

3.2.4.3 Steuerung

Für die Steuerung wurde ein dazu passendes Servoshield (24) verwendet, das bereits einen programmierbaren ESP32-Mikrocontroller integriert hat. Im Gegensatz zu herkömmlichen Servos bieten diese Servomotoren eine grosse Vielzahl von Parameter und Messdaten, die ausgelesen und eingestellt werden können. Darunter gehört die Messung der Winkelposition, Winkelgeschwindigkeit, das Drehmoments (Torque), und die Stromaufnahme. Es ist ausserdem möglich, Positionsgrenzen für den Winkel festzulegen und die PID-Regelparameter anzupassen. Die serielle Ansteuerung der Servos hatte einen erheblichen Mehraufwand in der Softwareentwicklung erfordert, ermöglicht aber langfristig eine präzisere Steuerung. Mit der entwickelten Software konnte das Roboter-Kaninchen in Betrieb genommen und erste Tests durchgeführt werden.

Da zukünftig ein RL-Netzwerk die Steuerung übernehmen soll, wurde entschieden, die umfangreichen Berechnungen auf einem externen Computer durchzuführen. Die Verbindung

zwischen Computer und ESP32 erfolgt über einen seriellen Bus, der sich als die effizienteste Methode zum Datenaustausch zwischen den beiden Geräten herausstellte.

3.2.4.4 Stromversorgung

Die Stromversorgung erfolgt über eine 12 V- Speisung mit maximalen 10A. Diese wird direkt ins ESP32-Servoshield eingespeist. Der Roboter-Kaninchen-Prototyp verwendet acht Servomotoren. Ein einzelner Servomotor zieht im blockierten zustand bis zu 2.7 A bei 12V. Unter Berücksichtigung der 2.7A für die acht Servos ergibt sich ein Spitzenwert von 21.6 A. Da jedoch nicht alle Servos gleichzeitig belastet werden, wurde eine Versorgung mit 10A gewählt.

3.2.5 Mechanische Konstruktion

Anhand der anatomischen Messdaten und der ausgewählten technischen Komponente konnte eine anatomisch dem Zwergkaninchen angepasste 3D Geometrie des Roboters erstellt werden.

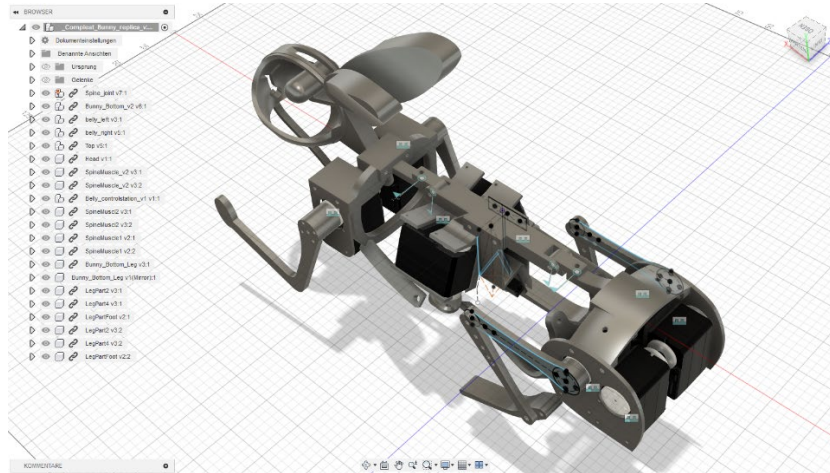


Abb. 10 Das Roboter-Kaninchen zusammengebaut mit den Servos in Fusion360. (Eigene Abbildung)

Das entwickelte Design wurde mithilfe des CAD-Programms Fusion 360 von Autodesk in einer 3D-Konstruktion erstellt. Zur Herstellung der entworfenen Teile aus PLA-Kunststoff kam ein 3D-Drucker zum Einsatz. Die gedruckten Teile wurden mit Schrauben zusammengefügt, wobei einfache Schrauben als Achsen für die Gelenke dienen.

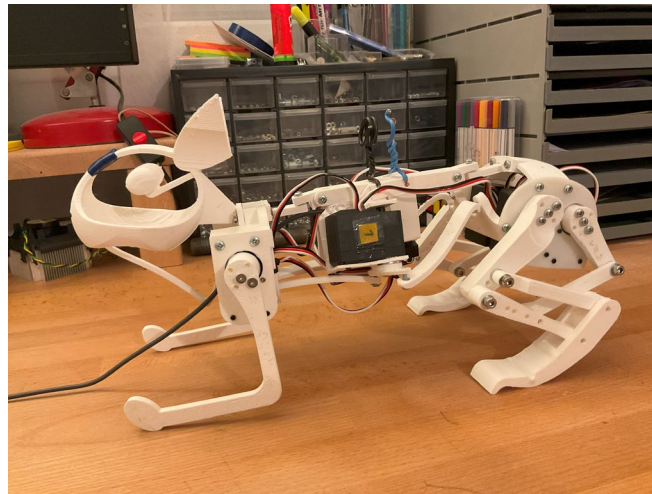


Abb. 11 3D Gedruckte Teile zusammengebaut zum Roboter-Kaninchen (Eigene Abbildung)

3.3 Teilprojekt B: Simulation

Um das Training eines Roboters zu beschleunigen, wird dieser als sogenannter Agent in einer virtuellen Umgebung trainiert. Dabei läuft das Training oft in beschleunigter Zeit ab und es können mehrere Agenten (Roboter) parallel trainiert werden. Dies verkürzt die normalerweise langsamen Trainingsprozesse, die in der realen Welt stattfinden würden.

Diese Agenten werden mit Hilfe von Reinforcement Learning (RL) trainiert, wobei viele ineinandergreifende Hyperparameter und Umgebungsparameter perfekt aufeinander abgestimmt werden müssen. Bereits kleinste Änderungen an diesen Parametern können zu komplett unterschiedlichen Trainingsergebnissen führen.

Im Laufe dieses Projekts wurden etwa 34 verschiedene Trainingsversuche unternommen, um die Auswirkungen dieser Parameter zu verstehen und zu optimieren. Mit der Zeit wurden wertvolle Erkenntnisse gesammelt, die in nachfolgenden Versionen des Trainingsprozesses eingeflossen sind.

In diesem Projekt wird die bisher erfolgreichste Methode vorgestellt, mit der das Roboter-Kaninchen in der Simulation trainiert wurde und mit der auch Experimente ausgetestet wurden. Das Programm für das Trainieren des Agenten mittels Reinforcement Learning (RL) besteht immer aus zwei Hauptkomponenten: Der Umgebung (Environment) und dem Lernalgorithmus.

3.3.1 Simulations-Umgebung

Die Umgebung ist eine simulierte 3D-Welt, in der das Roboter-Kaninchen als lernender Agent agiert. Um das konstruierte Roboter-Kaninchen mit seinen Gelenken und physikalischen Parametern in einer virtuellen Umgebung abzubilden, wurde die Pybullet-Physiksimulation (25) verwendet.

Diese Umgebung kann URDF (Universal Robotic Description Format) (6) Files verwenden, um den Roboter (Agenten) in der Simulation zu verwenden. Da Fusion 360 von Haus aus keine Funktion zur direkten Konvertierung in ein URDF bietet, wurde eine Erweiterung (26) verwendet. Mit diesen URDF-Informationen konnte das Roboter-Kaninchen korrekt in der Physiksimulation dargestellt werden.

Die ganze Umgebung basiert auf einer Standardumgebung, die von der Standard API «gymnasium» (23) bereitgestellt wird. Dadurch kann die später gebrauchte RL-Bibliothek Stable Baselines3 (10) mit dieser Umgebung interagieren.

Alle relevanten Parameter von der Umgebung müssen dem Environment-Objekt beim Initialisieren übergeben werden. Diese Parametrisierung macht es so einfacher, eine Vielzahl von verschiedenen Experimenten durchzuführen und die dazugehörigen Parametereinstellung gleich zu speichern.

Im Folgenden werden alle möglichen Parametereinstellungen aufgelistet, um einen Einblick in die Vielfalt der möglichen Experimente zu erhalten:

Parameter	Beschreibung
ModelType	SAC oder PPO
Rewards_type	z.B. «Modified_Peng_et_al_reward»
Observation_type	States-Arten, die dem Agenten übergeben werden.
Simulation_stepSize	$X \cdot 0.01s$ werden nach einem Step gerendert
Obs_time_space	Wie viele Sekunden kann der Agent in die Vergangenheit schauen.
maxSteps	Maximale Anzahl Steps die in einer Epoche gemacht werden dürfen. Sonst wird diese Epoche geschlossen.
Restriction_2D	Wenn True: Wird der Roboter durch äussere Kräfte auf eine gerade Linie beschränkt. Er bewegt sich in diesem Fall in einer 2D- Umgebung.
Terrain_type	«flat_terrain» / «uneven_terrain» / «random_terrain»
Recorded_movement_file_path_list	Eine Liste mit Paths zu Experten-Bewegungen

Tabelle 3 Parameter der Umgebung

Wichtige API-Befehle für den Simulationsablauf ist der Reset-Befehl wie auch der Step-Befehl, die im nachfolgenden genauer erklärt sind:

3.3.1.1 Reset-Befehl:

Der Reset-Befehl setzt den Roboter und seine Gelenke in ihre Startposition zurück. Alle Umgebungsinformationen werden ebenfalls zurückgesetzt, und es wird die erste Beobachtung des Roboters (observation, state) dem RL-Algorithmus übergeben.

3.3.1.2 Step-Befehl:

Der Step-Befehl ist ein simulierter Zeitabschnitt. Dabei nimmt er von dem RL-Algorithmus ein Aktionsarray (eine Liste von Soll-Winkelwerten) entgegen, das an die Servos übergeben wird. Mithilfe eines PID-Reglers ändern diese Servos ihre Winkelposition entsprechend diesen Soll-Winkeln.

Nach jedem Step wird das nächste Observation-Array, ein Reward (Imitation-Reward), Termination-Zustand und Truncated-Zustand dem RL-Agenten für den nachfolgenden Step mitgegeben.

3.3.1.3 Observation-Array

Dieses enthält Informationen über den Zustand des Agenten. Es umfasst in der Regel 10 historische Observation-Arrays, die sowohl die aktuellen Zustände als auch Informationen aus der Vergangenheit enthalten. Diese Observations beinhalten unter anderem die auf die Gelenke wirkenden Kräfte, Gelenkwinkelpositionen und die Richtung des Zieles vom Roboter aus gesehen.

3.3.1.4 Termination-Zustand:

Die Umgebung gibt ein Terminations-Zustand aus, wenn die Episode natürlich beendet wird. Ein natürliches Ende tritt ein, wenn der Roboter umfällt und «scheitert». Weitere unerlaubte Umstände sind das Berühren des Bodens mit dem Kopf, Bauch oder Hinterteil - nur die Füße und Pfoten dürfen den Boden berühren. Zudem kann Trainingszeit gepaart werden, wenn eine maximal erlaubte Distanz zum Vorbild (Experten) eingehalten werden muss.

3.3.1.5 Truncated-Zustand:

Die Umgebung gibt ein Truncated-Zustand als True aus, wenn die Episode aufgrund einer Zeitbegrenzung (definiert durch maxSteps) abgebrochen wird. Dadurch können endlos lange Episoden vermieden werden, die das Trainieren verlangsamen würden.

3.3.2 Experten-Bewegungsabläufe

Ein separates Hilfstool (Abb. 12) wurde entwickelt, um manuell Expertenbewegungen zu erstellen. Ein Experte ist in der Lage, eine Bewegungsabfolge zyklisch in einer Schleife zu wiederholen. Die Bewegungsabläufe wurden in der Physiksimulation getestet und anschliessend über mehrere Zyklen hinweg aufgezeichnet. Die Aufzeichnungen erfassen alle möglichen States der Umgebung, wobei für das spätere Trainieren nur bestimmte States verwendet werden. Zu den wichtigsten aufgezeichneten Parametern gehören:

Variablenname	Beschreibung	Formel
<i>action</i>	Die ausgeführte Aktion des Roboters	\hat{a}_t
<i>pos_array</i>	Root-Position des Roboters. Befindet sich in der Mitte der Wirbelsäule.	x_t^{root}
<i>euler_array</i>	Euler-Winkel bei Rootposition	q_t^{root}
<i>vel_array</i>	Root-Geschwindigkeit	\hat{x}_t^{root}
<i>joint_angles</i>	Liste mit Gelenkwinkel	\hat{q}_t^j
<i>joint_torques</i>	Liste mit Drehmomenten, die auf die Gelenke wirken.	$\hat{\tau}_t^j$
<i>component_coordinates_world</i>	Weltkoordinaten der Roboterkomponenten.	\hat{x}_e^t

Tabelle 4 Auflistung der wichtigen aufgezeichneten Zustände.

Diese aufgezeichneten Daten wurde später in der Imitation-Reward-Funktion verwendet, um Bewegungsunterschiede zwischen dem lernenden Agenten und den aufgezeichneten Expertenbewegungen zu berechnen. Basierend auf diesen Differenzen erhielt der lernende Agent entsprechende Belohnungen.

Eine Aufzeichnung umfasst eine komplette Episode. Das bedeutet, dass alle relevanten Daten kontinuierlich vom Startpunkt bis zum Ende der Episode (entweder durch das «Sterben» des Agenten oder durch das Erreichen der maximalen Zeit) erfasst werden.

Mit der Hilfs-App wurden viele verschiedene Bewegungsabläufe erstellt und aufgezeichnet. Einige liessen den Roboter geradeaus springen, andere bewegten ihn in Slalomlinien, Kurven nach

rechts oder links, oder mit Pausen. Diese Vielfalt an Bewegungen half dem lernenden Agenten, verschiedene Muster zu erlernen und sein Verhalten anzupassen.

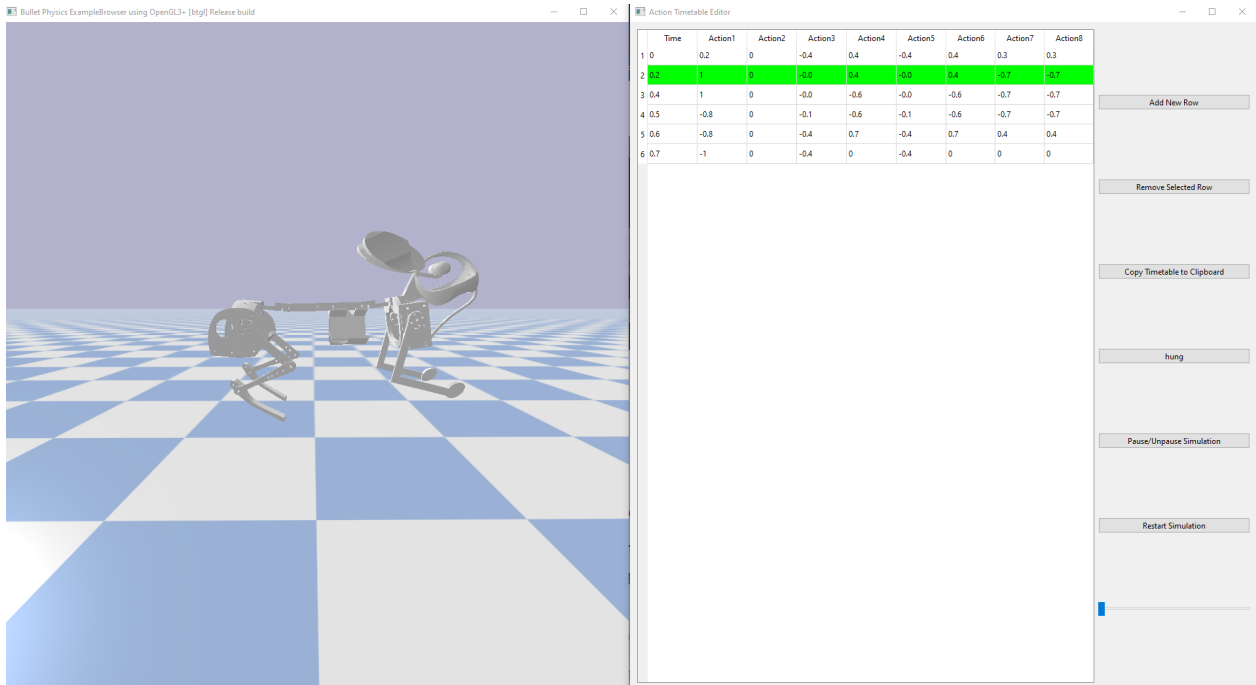


Abb. 12 ByBulld links mit ergänztem Benutzer interface. (Eigene Abbildung)

Das Hilfstool (Abb. 12) dient zur manuellen Erstellung von Expertenbewegungen. Auf der linken Seite wird die Simulationsumgebung angezeigt, in der sich der Roboter bewegt. Rechts befindet sich ein Kontrollpanel mit Steuerelementen zur Bedienung der Umgebung und der Bewegungsabfolge. In der Tabelle sind die Bewegungsschritte aufgelistet, wobei die aktuelle Bewegungsposition des Roboters grün hervorgehoben ist.

3.3.3 Imitations-Reward-Funktion

Das Erstellen einer effektiven Belohnungsfunktion (eng. *reward-function*) stellt eine grosse Herausforderung dar. Diese Funktion muss so gestaltet sein, dass der Agent kontinuierlich in die gewünschte Richtung gelenkt wird. Idealerweise soll der Agent für jeden kleinen Schritt, der ihn dem Ziel näherbringt, belohnt werden. Um dies zu erreichen, kamen verschiedene Methoden zum Einsatz. In diesem Ansatz wurde auf eine Imitationsstrategie zurückgegriffen (27) (28).

Die Reward-Funktion wurde in der «force_imitation» Umgebung implementiert. Nach vielen verschiedenen Versuchen, eine optimale Reward-Funktion zu entwickeln, wird nun die erfolgreichste Version erläutert.

Die Methode zur Berechnung des Rewards, die hier als «Modified_Peng_et_al_reward» bezeichnet wird, basiert auf einer ähnlichen Herangehensweise wie die der Arbeit von Peng_et_al. (29). Im Wesentlichen wird der Reward basierend auf der Genauigkeit der Nachahmung eines Experten berechnet. Der Agent erhält eine Belohnung, die proportional zur Übereinstimmung seines Verhaltens mit den Trajektorien eines zuvor erstellten Experten ist.

Dieser Ansatz gewährleistet, dass der Agent bestrebt ist, den Bewegungen des Experten so genau wie möglich zu folgen. Jede Abweichung von der vorgegebenen Trajektorie führt zu einer geringeren Belohnung, während eine genaue Nachahmung entsprechend belohnt wird.

3.3.3.1 Die Basisstruktur der Rewardfunktion

Die Basisstruktur jeder Teil-Reward-Funktion $R(d)$ ist eine Exponentialfunktion mit der Basis e (der Eulerschen Zahl):

$$R(d) = \exp(b \cdot d^2) \quad (3.7)$$

Der Exponent besteht aus $b \cdot d^2$, wobei der Wert b die «Fallgeschwindigkeit» der Kurve bestimmt. Diese Funktion $R(d)$ liefert einen Reward basierend auf der Distanz d , wobei das Ziel darin besteht, die Distanz zwischen dem Wert des lernenden Agenten und dem Wert des Experten so klein wie möglich zu halten. Durch die Quadrierung der Distanz ergibt sich eine Gaußsche Kurve, die am Anfang und Ende abgerundet ist. Je kleiner die Distanz, desto stabiler ist die Belohnung, wie in der Abb. 13 ersichtlich ist.

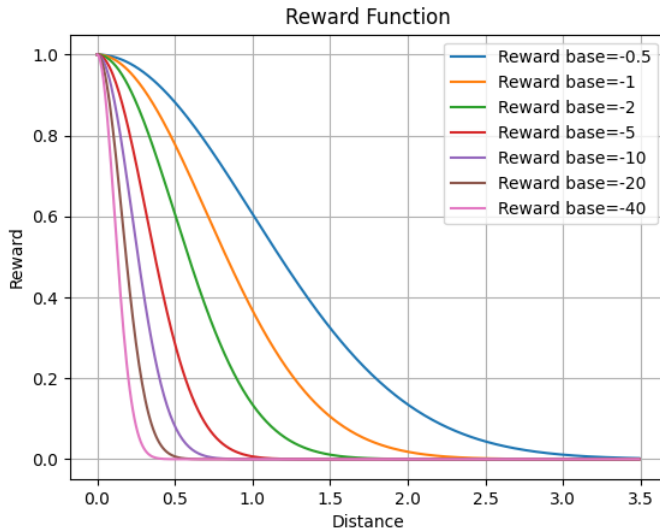


Abb. 13 Darstellung der Reward vs. Distanzen zwischen dem Experten und des lernenden Agenten. (Eigene Abbildung)

Aufbauend auf dieser gaußschen Funktion wurden mehrere Teil-Reward-Funktionen erstellt, die schliesslich summiert werden, um den Gesamt-Reward r_t zu bilden.

Der Gesamt-Reward wird bei jedem Zeitschritt wie folgt berechnet:

$$r_t = w_p \cdot r_t^p + w_v \cdot r_t^v + w_e \cdot r_t^e + w_t \cdot r_t^t + w_{rp} \cdot r_t^{rp} + w_{rv} \cdot r_t^{rv} + w_a \cdot r_t^{wa} + w_{lt} \cdot r_t^{pw} \quad (3.8)$$

Die Gewichte der einzelnen Teil-Reward-Funktionen sind wie folgt definiert:

$$w_p = 0.5, \quad w_v = 0.05, \quad w_e = 0.15, \quad w_t = 0.05, \quad w_{rp} = 0.1, \quad w_{rv} = 0.05, \quad w_a = 0.15, \quad w_{lt} = 0.05 \quad (3.9)$$

Zur Übersicht werden im Folgenden nur die am stärksten gewichteten Teil-Reward-Funktionen beschrieben. Eine vollständige Beschreibung ist im Anhang detailliert erläutert.

3.3.3.2 Pose Reward r_t^p :

Der Pose Reward ermutigt den Agenten, die Differenz der Gelenkwinkelpositionen zwischen dem Lernenden Roboter und dem Experten so klein wie möglich zu halten. Die Differenz zwischen den Winkelpositionen des Experten \hat{q}_t^j und des Agenten q_t^j wird berechnet und durch 2π geteilt, um die relative Differenz der Winkel (in Radiant) zu bestimmen. Die quadrierte Differenz wird mit der Fallgeschwindigkeit $b = -20$ multipliziert als Exponent in der Exponentialfunktion genutzt:

$$r_t^p = \exp \left(-20 \cdot \frac{\sum_j (q_t^j - \hat{q}_t^j)^2}{(2\pi)^2} \right) \quad (3.10)$$

3.3.3.3 End-Effector Reward r_t^e :

Der End-Effector-Reward zielt darauf ab, die Distanz zwischen den End-Effektor-Positionen des Experten und des Agenten zu minimieren. Die Distanz wird als Differenz der Koordinaten der relevanten Bauteile des Experten \hat{x}_e^t und des Agenten x_e^t berechnet. Die Fallgeschwindigkeit wurde hier steil mit $b = -20$ gewählt:

$$r_t^e = \exp \left(-20 \cdot \sum_e (\hat{x}_e^t - x_e^t)^2 \right) \quad (3.11)$$

3.3.3.4 Root Pose Reward r_t^{rp} :

Der Root-Pose-Reward minimiert die Raumdistanz und Winkeldifferenz zwischen dem Root-Punkt des Agenten (x_t^{root}, q_t^{root}) und des Experten ($\hat{x}_t^{root}, \hat{q}_t^{root}$). Die Fallgeschwindigkeit b der Kurve liegen bei -40 für die Raumdistanz und bei -20 für die Winkeldifferenz:

$$r_t^{rp} = \exp \left(-40 \cdot \|\hat{x}_t^{root} - x_t^{root}\|^2 - 20 \cdot \left(\frac{\|\hat{q}_t^{root} - q_t^{root}\|}{2\pi} \right)^2 \right) \quad (3.12)$$

3.3.3.5 Action-Reward r_t^{wa} :

Die Aktionen des Agenten a_t werden mit denen des Experten \hat{a}_t verglichen. Die Fallgeschwindigkeit wurde hier steil mit $b = -5$ gewählt.

$$r_t^{wa} = \exp(-5 \cdot \|\hat{a}_t - a_t\|^2) \quad (3.13)$$

3.3.4 Trainieren von RL-Agenten

In diesem Abschnitt wird das Trainingsverfahren, die Analyse der Ergebnisse sowie die verwendeten Algorithmen, Hyperparameter und die RL-Architektur beschrieben, die für die im Resultat aufgeführten Trainingsexperimente verwendet wurden.

3.3.4.1 *Grundlegendes Trainingsverfahren*

Da ein Trainingsversuch auf dem genutzten Laptop mehrere Stunden bis Tage dauern konnte, musste jeder Versuch sorgfältig geplant und auf Fehler überprüft werden, um unnötigen Zeitverlust zu vermeiden. Die Dokumentation der Umgebungs- und Hyperparameter war wichtig, um spätere Trainingsversuche nachvollziehen und reproduzieren zu können. Während des Trainings wurde zudem ein Logbuch geführt.

3.3.4.2 *Analyse der Trainings*

Zur einfachen Analyse der Trainingsergebnisse wurde Tensorboard genutzt. Diese Benutzeroberfläche liest das Logbuch aus und erstellt Graphiken wie Lern- und Überlebensdauerkurven. So konnte bereits während des Trainings entschieden werden, ob der Versuch fortgeführt oder abgebrochen werden sollte. Ein eigens entwickeltes Tool ermöglichte es, trainierte Agenten in verschiedenen Umgebungen zu testen und deren Überlebensdauer sowie erhaltenen Rewards zu vergleichen.

3.3.4.3 *Algorithmen*

Für das Training wurden hauptsächlich die Algorithmen PPO und SAC verwendet. Anfangs kam hauptsächlich PPO zum Einsatz, aber Vergleiche mit SAC zeigten, dass SAC für dieses Projekt besser geeignet war.

3.3.4.4 Hyperparameter:

Um faire Vergleiche zwischen PPO und SAC zu ermöglichen, wurden ähnliche Hyperparameter verwendet:

Parameter	Value SAC	Value PPO
Default Parameters	Standard SAC	Standard PPO
Learning_rate	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
Batch_size	10'000	10'000
Buffer_size	500'000	-
policy_kwargs	pi = [512, 256], qf=[512, 256]	pi: [512, 256], vf: [512, 256]
Gamma	0.95	0.95
Number of CPUs	10	10

Tabelle 5 Verwendete Hyperparameter von PPO und SAC.

3.3.4.5 RL-Architektur:

Die RL-Architektur beider Algorithmen basiert auf demselben neuronalen Netzwerk, das in zwei Hidden Layers von 512 auf 256 Neuronen reduziert wird. Dies gilt für den PPO- und SAC-Actor sowie für die Q- und V-Funktion.

Die folgende Abbildung zeigt die RL-Architektur (Abb. 15):

- Die States s werden dem Actor-Netzwerk mit zwei Hidden Layers (512 und 256 Neuronen) zugeführt. Die Ausgabe ist eine Aktion a in einem Aktionsraum mit 8 Variablen.
- Aktion a und State s werden dem Critic-Netzwerk zugeführt, das denselben Aufbau hat und einen Q-Value liefert.
- Nach einer bestimmten Anzahl von Episoden wird das Actor-Netzwerk durch das Critic-Netzwerk und den Replay-Buffer trainiert.

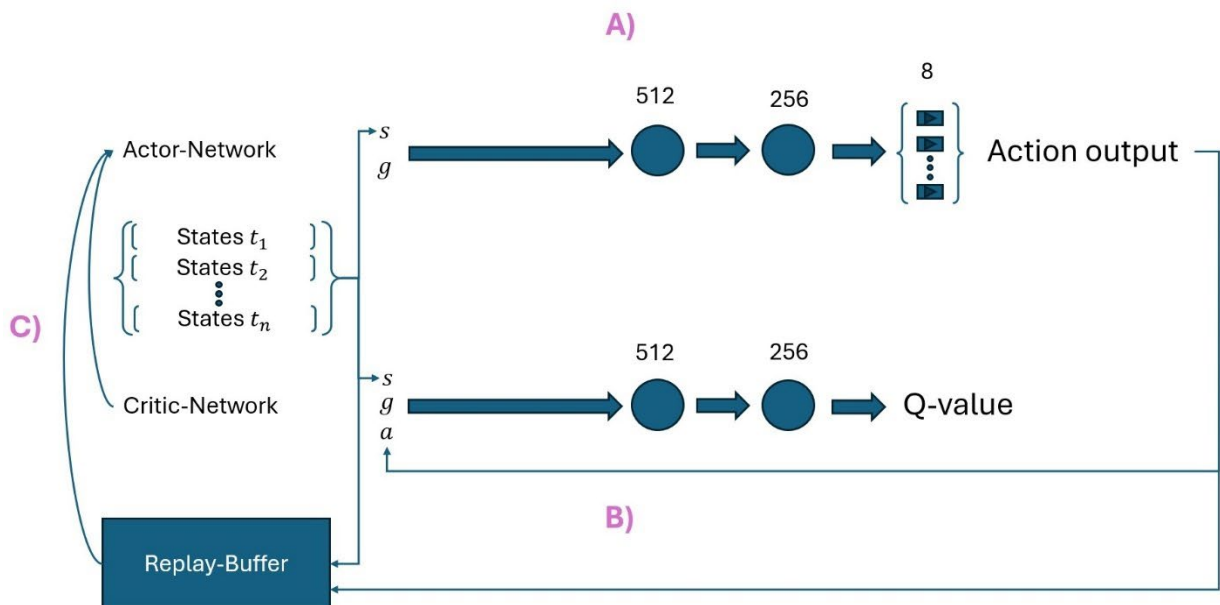


Abb. 14 Graphische Darstellung der SAC-Architektur. (Eigene Abbildung)

3.4 Teilprojekt C: Anwenden der Experten Bewegungen am realen Roboter

Das Hilfstool, das zur Erstellung der Expertenbewegungen verwendet wurde, ermöglicht auch das Abspielen dieser Bewegungen sowie der trainierten Agenten auf dem echten Roboter. Um die Bewegungen des realen und des virtuellen Roboters besser vergleichen zu können, kann der virtuelle Roboter gleichzeitig betrieben werden. Alle Funktionen des Benutzerinterfaces, wie Pausieren, Verlangsamen oder Neustarten der Bewegungsabfolge, können ebenfalls genutzt werden.

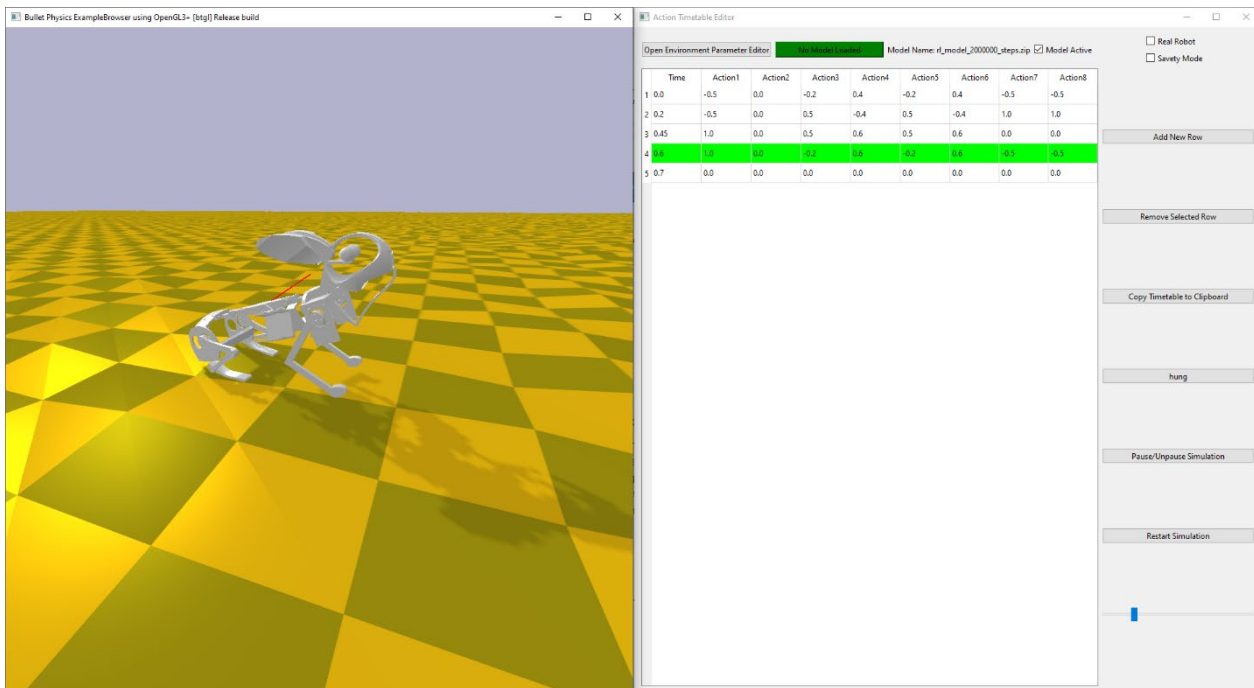


Abb. 15 Hilfstool, für anwenden der gelernte Agentenbewegungen vergleich mit den Experten daten. (Eigene Abbildung)

4 Resultate

In den folgenden Unterkapiteln werden die Ergebnisse der Teilprojekte A, B und C präsentiert, um die gestellten Fragen zu diskutieren. Zur besseren Übersicht ist die Struktur wieder in die drei Teilprojekte unterteilt:

1. **Teilprojekt A)** Ergebnisse der Entwurfsphase des Roboter-Kaninchens
Hier werden die Erfolge und Herausforderungen beim Entwurf, der Konstruktion und dem Zusammenbau des Roboter-Kaninchens beschrieben.
2. **Teilprojekt B)** Ergebnisse des Trainings des simulierten Roboter-Kaninchens
Es werden verschiedene Fortbewegungsstrategien der trainierten RL-Agenten vorgestellt. Zudem werden Trainingsergebnisse von Agenten präsentiert, die in unterschiedlichen Umgebungen mit verschiedenen Terrains trainiert wurden.
3. **Teilprojekt C)** Ergebnisse bei der Steuerung des realen Roboters
Es wird untersucht, wie gut der Roboter die in der Simulation erlernten Bewegungen in der realen Welt umsetzen und sich fortbewegen kann.

4.1 A) Ergebnis Roboter Kaninchen

In diesem Abschnitt werden die Erfolge und Herausforderungen bei der Planung, Konstruktion und dem Zusammenbau des Roboter-Kaninchens erläutert.

4.1.1 Erfolgreicher Zusammenbau

Der konstruierte Roboter konnte gemäss dem biologischen Vorbild eines Zwergkaninchens nachgebaut werden. Diese Nachbildung wurde nach einem genauen biologischen Vorbild einem Zwergkaninchen bezüglich der Struktur und Dimensionen aufgebaut. Der Roboter kann sich vorwärts bewegen, sich mit zwei Servos bücken (zusammenziehen) und strecken sowie den Rücken seitlich nach rechts und links krümmen.

Resultate

Trotz anfänglicher Schwierigkeiten wurde der Zusammenbau des Roboters erfolgreich abgeschlossen.

Der Rücken kann nun in alle notwendigen Richtungen gebogen werden (Abb. 16, Abb. 17, Abb. 18, Abb. 19). Und die speziell konstruierten 2-DOF parallel-Hinterläufe wie auch Vorderläufe des Roboters liessen sich ebenfalls in die gewünschten Richtungen ansteuern.

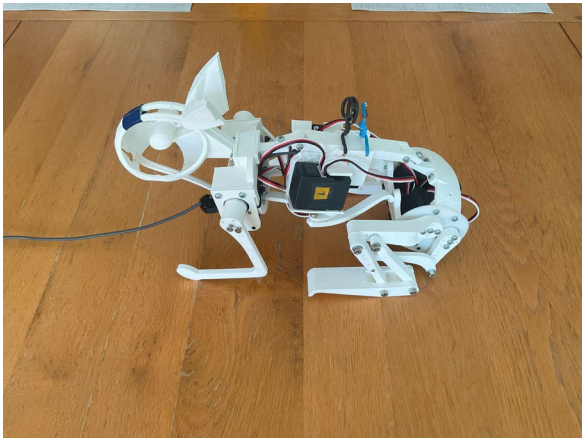


Abb. 16 Zusammengezogene Haltung

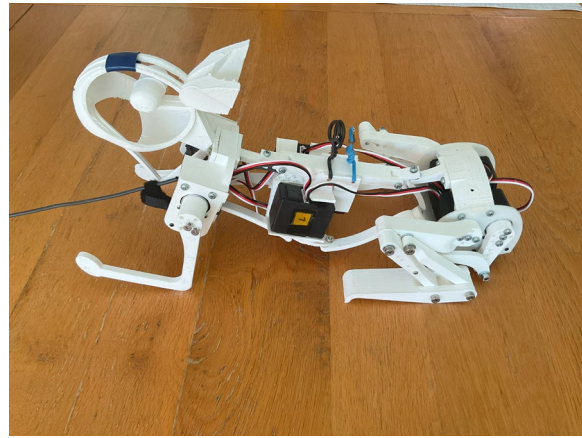


Abb. 17 Gestreckte Haltung

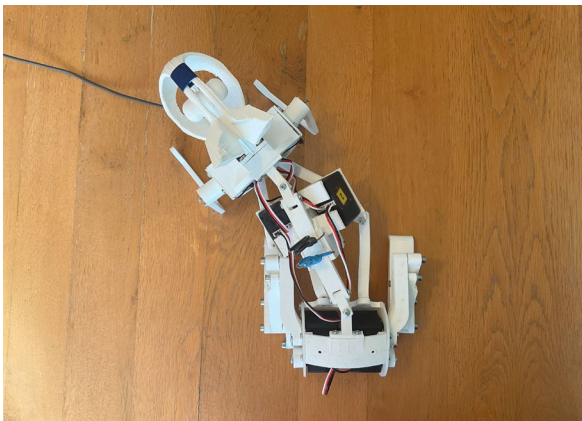


Abb. 18 leicht linksgekrümmte Haltung

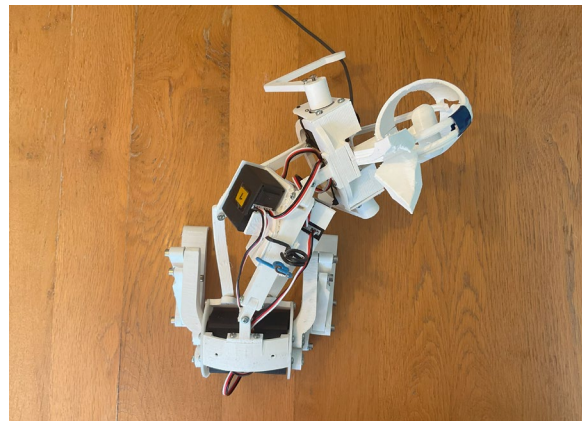


Abb. 19 starke rechtsgekrümmte Haltung

4.1.2 Konstruktive Schwierigkeiten

Ein zentrales Problem bei der Konstruktion war die Stabilität der Hinterläufe. Die anfänglich dünnen Hinterläufe wurden von leistungsstarken Servos oft verbogen oder auseinandergerissen. In der unteren Abbildung (Abb. 20) ist ein Bein zu sehen, bei dem der Oberschenkel direkt beim Motorgelenk gebrochen ist.



Abb. 20 Brüchigere Versionen der Hinterläufe.

Oberhalb der Abbildung ist ebenfalls ein Oberschenkel abgebildet, der nach aussen verbogen wurde. In den meisten Fällen war der Oberschenkel unstabil.

4.2 B) Ergebnisse des Trainings des simulierten Roboter-Kaninchens

Mehrere Agenten wurden erfolgreich trainiert, wobei der SAC-Algorithmus effizientere Ergebnisse als der PPO-Algorithmus erzielte. Die Imitation-Reward-Funktion "Modified_Peng_et_al_reward" motivierte jeden Agenten, Bewegungen nachzuahmen. Videoaufnahmen der Agenten sind auf dem folgenden GoogleDrive enthalten und über den jeweiligen Dateipfad zugänglich: <https://drive.google.com/drive/folders/1SwDRL51srnyJ0N2n3uRUVIfwi2vgcFA3?usp=sharing>

4.2.1 Vergleich PPO und SAC

Wie bereits in der theoretischen Grundlage erwähnt, lernt der SAC-Algorithmus zeiteffizienter als PPO. Der PPO-Algorithmus konnte in einer Trainingszeit von 6.5 Stunden etwa 10 Millionen neue Steps (Zeitschritte) lernen. Der SAC-Algorithmus benötigte hingegen 16.2 Stunden, um die gleiche Anzahl an Schritte zu lernen. Allerdings erzielte SAC in diesen wenigen Schritten fast exponentiell bessere Ergebnisse.

Die Abb. 21 zeigt die Gesamtsumme der belohnten Punkte (total discounted reward), die während einer Episode gesammelt wurden. Abb. 22 zeigt den durchschnittlichen Reward pro Schritt (relativer Reward), unabhängig von der Dauer der Episode.

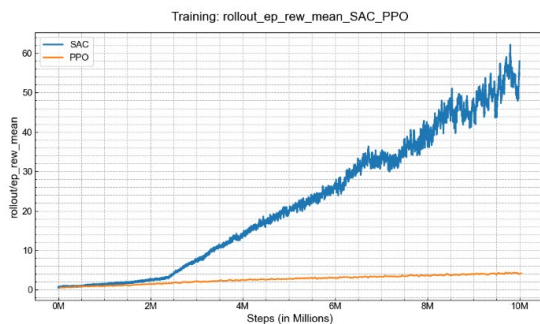


Abb. 21 Reward von PPO und SAC über 10Mio Steps.

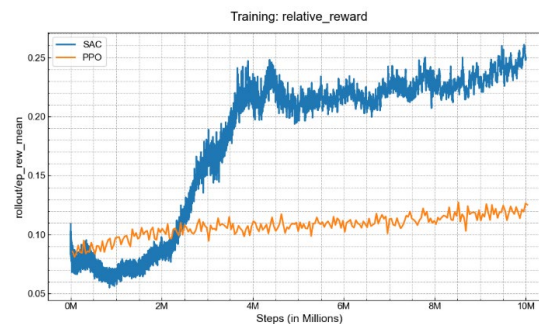
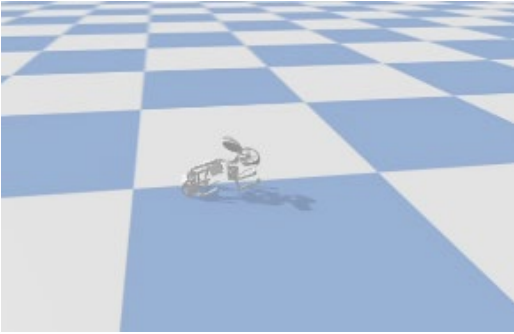
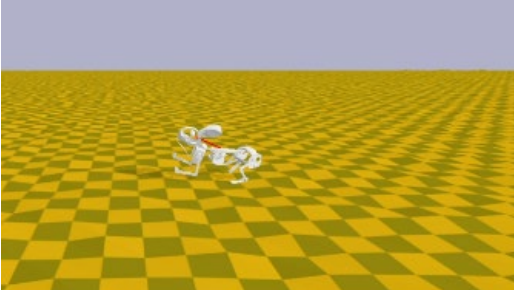
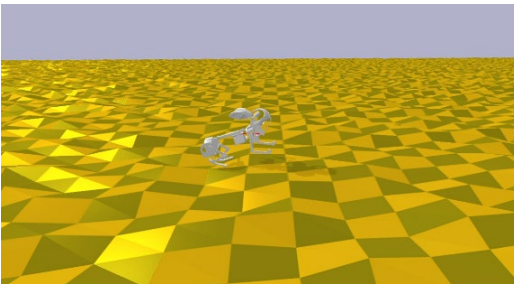


Abb. 22 Relative Reward von PPO und SAC

4.2.2 Natürliche Terrains und ihre Auswirkungen auf Bewegungen

Eine weitere Untersuchung bestand darin, herauszufinden, ob die Umgebung mit dem Terrain die trainierte Bewegung des Agenten beeinflussen kann. Dafür wurden drei verschiedene Umgebungen mit verschiedenem Terrain untersucht. Die erste Umgebung besteht ausschliesslich aus einem flachen Boden, die zweite Umgebung hat flache Hügel und die dritte Umgebung hat viele steile Hügel, sodass diese mit einem verwurzelten Waldboden zu vergleichen ist.

Im Folgenden sind die Umgebungen mit dazugehörigen Videoaufnahmen aufgelistet:

<p>Umgebung 1</p> <p>Dateipfad:</p> <p><i>Experiment_Terrain/evaluationUmgebung_1.mp4</i></p> <p><i>Experiment_Terrain/trainingUmgebung_1.mp4</i></p>	 <p><i>Abb. 23 Agent in flachen Umgebung 1</i></p>
<p>Umgebung 2</p> <p>Dateipfad:</p> <p><i>Experiment_Terrain/evaluationUmgebung_2.mp4</i></p> <p><i>Experiment_Terrain/trainingUmgebung_2.mp4</i></p>	 <p><i>Abb. 24 Agent in mittel unebener Umgebung 2</i></p>
<p>Umgebung 3</p> <p>Dateipfad:</p> <p><i>Experiment_Terrain/evaluationUmgebung_3.mp4</i></p> <p><i>Experiment_Terrain/trainingUmgebung_3.mp4</i></p>	 <p><i>Abb. 25 Agent in sehr unebener Umgebung 3</i></p>

Resultate

Die Lernkurven (Abb. 26) der Agenten in Umgebung 1 und 2 ähneln sich stark. Der Agent in Umgebung 3 zeigt eine flachere Lernkurve und erreichte am Ende des Trainings keinen höheren Reward als die Agenten in den anderen Umgebungen.

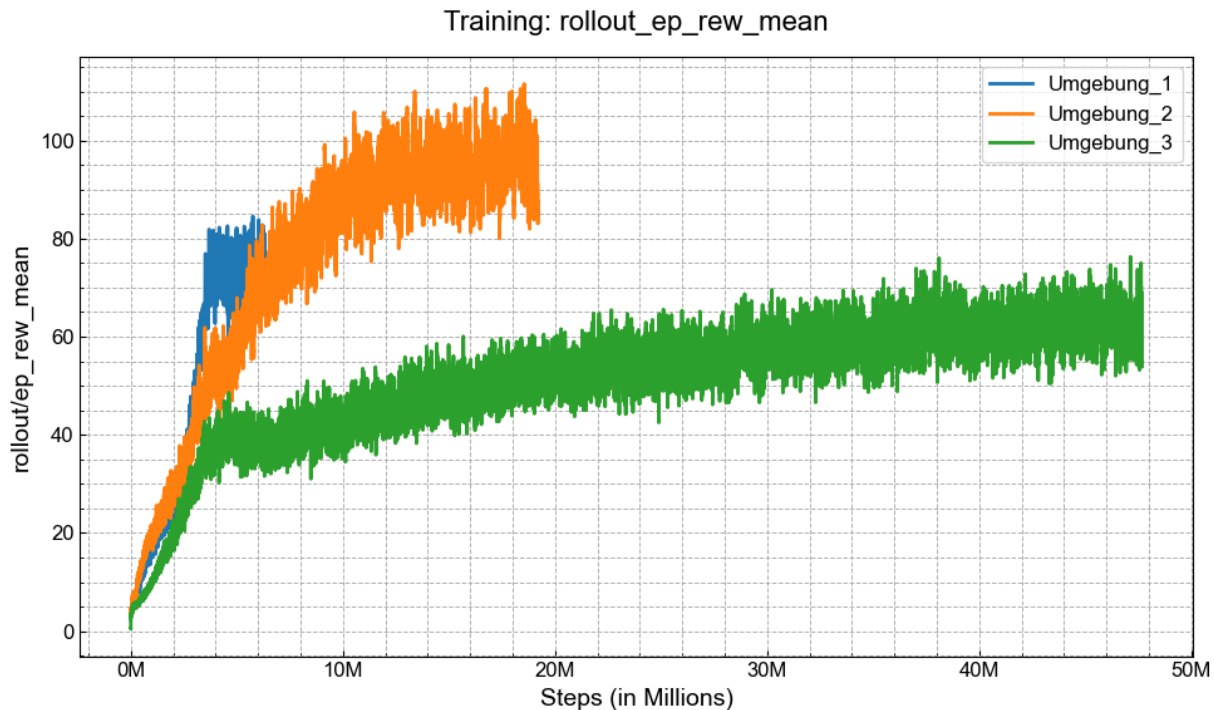


Abb. 26 Lernkurve der Agenten in den drei verschiedenen Umgebungen

Bei der Analyse der Videoaufnahmen fällt auf, dass der Agent in Umgebung 3 die Bewegung eines Kaninchens am ehesten nachahmte.

- In Umgebung 1 bewegt sich der Agent vorsichtig, bleibt nahe am Boden und schiebt sich fast ausschliesslich mit den Hinterbeinen vorwärts, während die Vorderbeine über den Boden gleiten.
- In Umgebung 2 nutzt der Agent seine Vorderbeine etwas mehr, bewegt sich aber dennoch in kleinen, unnatürlichen Sprüngen nach vorne.
- In Umgebung 3 verwendet der Agent sowohl Vorder- als auch Hinterbeine, um sich fortzubewegen, was der natürlichen Bewegung eines Zwergkaninchens stark ähnelt.

Resultate

Für einen fairen Vergleich wurden alle trainierten Agenten in jeder Umgebung über 50 Episoden hinweg getestet. Die Graphiken zeigen die durchschnittliche Belohnung (Abb. 27) und die durchschnittliche Überlebensdauer (Abb. 28) der Agenten:

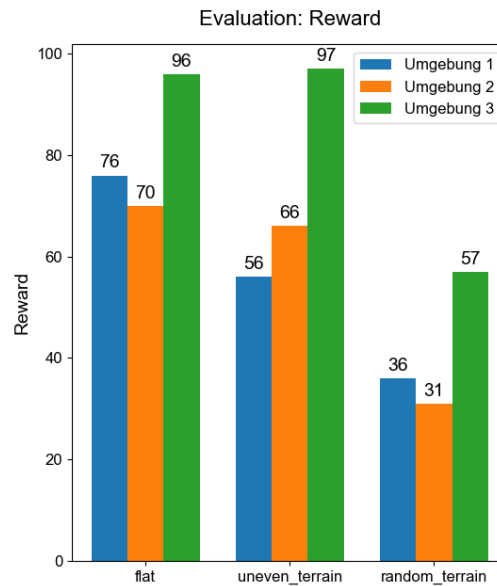


Abb. 27 Vergleich der durchschnittlich erhaltenen Belohnung der Agenten aus den verschiedenen Umgebungen

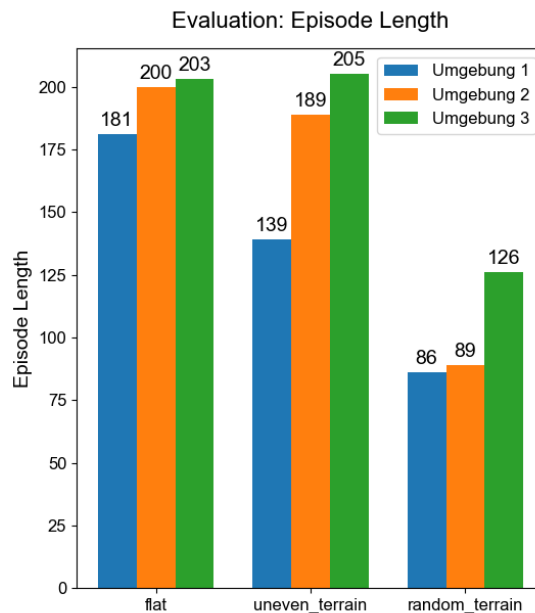


Abb. 28 Vergleich der durchschnittlichen Überlebensdauer der Agenten aus den verschiedenen Umgebungen

4.2.3 Fortbewegungsvergleich

Zur Veranschaulichung der Unterschiede und Gemeinsamkeiten in der Fortbewegung zwischen dem Roboter und einem echten Kaninchen sind im folgenden Ordner Videoaufnahmen hinterlegt. Diese enthalten Aufnahmen des simulierten Roboterkaninchens sowie zwei Referenzvideos von echten Kaninchen: [*Fortbewegungsvergleich/*.mp4*](#)

4.3 C) Ergebnisse bei der Steuerung des realen Roboters

Die Übertragung der Bewegungsabläufe aus der Simulation auf den realen Roboter verlief insgesamt gut. Allerdings gab es Bewegungen, die in der Simulation problemlos funktionierten, aber in der Realität nicht umsetzbar waren. Der Roboter konnte seine Hinterläufe nicht schnell genug nach vorne ziehen oder sich effektiv vom Boden abstossen. Im nächsten Kapitel werden die Bewegungen in der Simulation und der realen Welt verglichen, gefolgt von einem Kapitel mit den erfolgreichsten Ergebnissen in der realen Welt.

4.3.1 Vergleich zwischen Simulation und realen Umgebung

Im folgenden Ordner sind Videos abgelegt, die einen direkten Vergleich der Bewegungsabläufe von Experten und der trainierten RL-Agenten mithilfe des Hilfstools in der Simulation und in der realen Umgebung zeigen: [*Vergleich_zwischen_Simulation_und_realen_Umgebung/*.mp4*](#)

4.3.2 Beste Ergebnisse in der realen Welt

Einige Expertenbewegungen haben es dennoch geschafft, das Roboter-Kaninchen in der realen Welt in Bewegung zu bringen. Interessanterweise schnitten diese Bewegungsabläufe in der Simulation schlechter ab. Die besten Fortbewegungsergebnisse in der realen Welt können im folgenden Ordner als Videos eingesehen werden. Zusätzlich werden die entsprechenden Bewegungen dieser Experten in der Simulation gezeigt: [*Beste_Ergebnisse/*.mp4*](#)

5 Diskussion

In diesem Kapitel werden die drei Teilprojekte und deren Ergebnisse im Zusammenhang analysiert und diskutiert, um Antworten auf die in der Einleitung formulierten Fragestellungen zu geben:

- **Teilprojekt A)** Ist es möglich ein Zwergkaninchen anatomisch sehr ähnlich nachzubilden.
- **Teilprojekt B)** Wird ein Reinforcement Learning KI den anatomischen Aufbau des Roboter Kaninchen ausnutzen und ein ähnliches Bewegungsmuster erlernen, wie das reale Kaninchen? Hat das Terrain der simulierten Umgebung einen Einfluss auf den schlussendlich trainierten Agenten?
- **Teilprojekt C)** Kann das reale Roboter-Kaninchen zu einer Vorwärtsbewegen gebracht werden?

5.1 A) Vergleich mit realen Zwergkaninchen

In diesem Abschnitt werden die Ähnlichkeiten und Unterschiede in der Bauweise, und der Form zwischen dem echten Zwergkaninchen und dem Roboterkaninchen betrachtet.

5.1.1 Ähnlichkeiten und Unterschiede

Basierend auf den Ergebnissen des Aufbaus zeigt das Roboterkaninchen hinsichtlich der Gelenkdimensionen, Körpergrösse und des Aussehens deutliche Ähnlichkeiten zu einem Zwergkaninchen. Trotz der Verwendung von nur acht Servomotoren kommt die Beweglichkeit des Roboters erstaunlich nahe an die eines echten Kaninchens heran. Lediglich die Beweglichkeit der Vorderläufe und des Rückens ist etwas eingeschränkt.

Ein wesentlicher Unterschied liegt in den Antrieben, da Kaninchen Muskeln haben, die linear arbeiten, während Servomotoren rotieren. Darüber hinaus besitzen Kaninchen viele verteilte Muskeln, wohingegen der Roboter nur einen Motor pro Gelenk hat. Ein Servomotor mit einem Drehmoment von 3 Nm (entspricht 30 kg auf 1 cm) kann schnell ein Kunststoffteil aus PLA mit einem Füllungsgrad von 25 % verbiegen oder brechen. Besonders die Oberschenkelbereiche sind aufgrund des Hebelgesetzes stark belastet.

Es wurde grosser Wert daraufgelegt, die Komponenten so leicht und kompakt wie möglich zu gestalten, um den Proportionen eines echten Kaninchens zu entsprechen.

Allerdings musste erkannt werden, dass PLA-Kunststoff andere Eigenschaften als Knochen aufweist, insbesondere hinsichtlich der inneren Struktur. Aus diesem Grund wurden einige besonders beanspruchte Teile dicker und mit einem Füllungsgrad von 60 % verstärkt ausgedruckt.

5.2 B) Erlente Bewegungen

Die Agenten, die mit der in der Methode beschriebenen Lernalgorithmen trainiert wurden, konnten erfolgreich kaninchenähnliche Fortbewegungsmuster erlernen. Zudem haben sie gezeigt, dass sie lange überleben und dabei einem Ziel hinterherspringen können. Diese Agenten haben drei Bewegungsarten gelernt, die sie je nach Zielabstand und Orientierung anwenden. Die Bewegungsarten umfassen das Sitzen, gemächliche Nachspringen und schnelle Nachspringen eines Ziels.

5.2.1 Vergleich PPO und SAC

Der Vergleich von PPO und SAC zeigt, dass der SAC-Algorithmus zeiteffizienter lernt, was besser zum Verhalten eines Kaninchens passt. SAC verarbeitet ständig seine früheren Erfahrungen, sodass weniger Bewegungsversuche nötig sind, um bereits Gelerntes zu wiederholen. Ein mit PPO trainierter Roboter würde in der realen Welt mehr Versuche und somit mehr Zeit benötigen, um Fortschritte zu machen, da die reale Zeit begrenzt ist.

Ausserdem ist der SAC-Algorithmus besser darin, neue Aktionen zu entdecken. Er wählt basierend auf der Entropiestrategie Aktionen, die zu neuen Zuständen führen, die dann als Erfahrungen gespeichert werden. Im Gegensatz dazu wählt der PPO-Algorithmus immer eine Aktion, die auf bisherigen Erfahrungen den grössten Erfolg verspricht.

5.2.2 Einfluss der Umgebung auf die Bewegung des Roboters

Das Terrain-Experiment zeigte, dass die Umgebung die Fortbewegungsstrategie stark beeinflusst. Wildkaninchen leben meist an Waldrändern, in Heidelandschaften und lichten Wäldern-Gebiete mit unebenem, hügeligem Boden (30). Die dritte Testumgebung im Experiment ähnelt diesen natürlichen Bedingungen. Der Agent, der in dieser Umgebung trainierte, schnitt besser ab als die Agenten aus flacheren Umgebungen, sowohl in der Überlebensdauer als auch bei der Belohnung.

Das Experiment zeigt viele Parallelen zur Realität: Sowohl der Roboter als auch echte Tiere passen ihre Bewegungen an ihre Umgebung an.

5.3 C) Unterschiede beim Springen des realen Roboter-Kaninchens

Das reale Roboter-Kaninchen konnte mit neuen Expertenbewegungen teilweise vorwärts bewegt werden, aber diese Bewegungen sind noch weit von denen in der Simulation entfernt. Bewegungsabläufe, die in der Simulation gute Ergebnisse zeigten, wie die der trainierten RL-Agenten, funktionierten in der realen Welt nicht wie erwartet. Im folgenden Kapitel werden die möglichen Unterschiede zwischen der Simulation und der Realität diskutiert.

5.3.1 Unterschiede zwischen Simulation und Realität

Bei der Übertragung der in der Simulation entwickelten Bewegungen auf den realen Roboter traten deutliche Unterschiede auf. Zunächst wurde vermutet, dass die manuell erstellten Expertenbewegungen nicht präzise genug waren, doch auch die trainierten Bewegungen verbesserten die Sprungtechnik des Roboters nicht.

Besonders grosse Unterschiede zeigten sich bei den 2-DOF-Parallel-Beinen, die mithilfe der Inversen Kinematik berechnet werden. In der Realität bewegt sich der Unterschenkel langsamer, da er mit dem Servo im Becken verbunden ist. Der Unterschenkel-Servo muss sich immer mit dem Oberschenkel-Servo synchronisieren, um den Kniewinkel stabil zu halten, während der Servo in der Simulation direkt am Knie angebracht ist. Das führt dazu, dass der Unterschenkel entweder zu spät oder zu schnell reagiert.

Weitere Unterschiede könnten an den verschiedenen PID-Regler-Parametern zwischen den simulierten und realen Servos liegen. Die realen Servos reagieren langsamer und können ihre Positionen nicht so schnell wechseln wie in der Simulation.

Zusätzlich könnten physikalische Parameter wie Seitenreibung (`lateralFriction`), Gelenkdämpfung (`jointDamping`) und andere Simulationswerte die Unterschiede erklären, da sie die Bewegungsrealität beeinflussen.

6 Fazit

Der Bau eines Roboter-Kaninchens eröffnete mir eine faszinierende, bisher unentdeckte Welt der Naturwissenschaft und Technik. Die Möglichkeit, die Bewegungsabläufe der Kaninchen zu verstehen und zu rekonstruieren, erfüllte mich sehr.

Gleichzeitig musste ich jedoch erkennen, dass mein Projekt sehr anspruchsvoll und aufwendig war, da das Konstruieren des Modells und das Trainieren in der Simulation und Anwendung viel Zeit und Hintergrundwissen in Anspruch nahm und dies teils mit kleineren Rückschlägen verbunden war. Es benötigte viel Durchhaltevermögen, um trotz zahlreicher Herausforderungen Fortschritte zu erzielen und nicht aufzugeben.

Der direkte Nutzen des Roboter-Kaninchens für eine Anwendung in der Industrie (zum Beispiel Spielzeugbranche oder in anderen Gebieten) stand für mich nicht im Vordergrund. Die Erfahrungen, die ich beim Konstruieren komplexer Systeme und beim Erforschen von RL-Algorithmen gesammelt habe, werden mir in zukünftigen Studien und Projekten von grossem Wert sein.

Für die Zukunft sehe ich grosses Potenzial in der weiteren Optimierung von RL-Methoden und deren Anwendung auf das Erlernen von natürlichen Bewegungsmustern.

Obwohl das Projekt nicht immer reibungslos verlief, verspüre ich den Wunsch, mich weiteren Experimenten in diesem Bereich zu widmen und das Projekt weiterzuentwickeln.

Zum Beispiel müssen am konstruierten Roboter-Kaninchen noch Verbesserungen an den Vorderläufen, Hinterläufen und dem Rücken vorgenommen werden. Darüber hinaus sollte die Simulation genauer an die Realität angepasst werden, und die RL-Agenten müssen auf mehr Interferenzen trainiert werden, damit sie direkt in der realen Welt eingesetzt werden können.

7 Literaturverzeichnis

1. Why-do-bunnies-hop. *Wonderopolis*. [Online] [Zitat vom: 2. Oktober 2024.]
<https://wonderopolis.org/wonder/why-do-bunnies-hop>.
2. Wikipedia. *Zwergkaninchen (Hauskaninchen)*. [Online]
[https://de.wikipedia.org/wiki/Zwergkaninchen_\(Hauskaninchen\)](https://de.wikipedia.org/wiki/Zwergkaninchen_(Hauskaninchen)).
3. Wikipedia. *Robot end effector*. [Online] https://en.wikipedia.org/wiki/Robot_end_effector.
4. Wikipedia. *Freiheitsgrad*. [Online] [Zitat vom: 3. Oktober 2024.]
<https://de.wikipedia.org/wiki/Freiheitsgrad>.
5. Zurich Instruments. *Principles of PID Controllers*. [Online] [Zitat vom: 3. Oktober 2024.]
<https://www.zhinst.com/ch/de/resources/principles-of-pid-controllers>.
6. URDF. *Formant*. [Online] [Zitat vom: 3. Oktober 2024.] <https://formant.io/resources/glossary/urdf>.
7. Künstliche Intelligenz. *Wikipedia*. [Online] [Zitat vom: 3. Oktober 2024.]
https://de.wikipedia.org/wiki/K%C3%BCnstliche_Intelligenz.
8. Maschinelles Lernen. *Wikipedia*. [Online] [Zitat vom: 1. Oktober 2024.]
https://de.wikipedia.org/wiki/Maschinelles_Lernen.
9. Bestärkendes Lernen. *Wikipedia*. [Online] [Zitat vom: 3. Oktober 2024.]
https://de.wikipedia.org/wiki/Best%C3%A4rkendes_Lernen.
10. readthedocs. *stable-baselines3*. [Online] [Zitat vom: 2. Oktober 2024.] <https://stable-baselines3.readthedocs.io/en/master/>.
11. neural-networks. *IBM*. [Online] [Zitat vom: 5. Oktober 2024.] <https://www.ibm.com/topics/neural-networks>.
12. datasolut. *Deep Learning: Definition, Beispiele & Frameworks*. [Online] [Zitat vom: 3. Oktober 2024.]
<https://datasolut.com/was-ist-deep-learning/>.
13. RL_Tips. *stable-baselines3.readthedocs.io*. [Online] [Zitat vom: 2. Oktober 2024.] https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html.
14. PPO. *stable-baselines3.readthedocs.io*. [Online] [Zitat vom: 3. Oktober 2024.] <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>.
15. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization Algorithms. *arxiv*. [Online] [Zitat vom: 3. Oktober 2024.] <https://arxiv.org/abs/1707.06347>.
16. Asad Ali Shahid, Dario Piga, Francesco Braghin, Loris Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Springer Link*. [Online] 09. 02 2022. [Zitat vom: 3. Oktober 2024.] <https://link.springer.com/article/10.1007/s10514-022-10034-z>.
17. SAC. <https://stable-baselines3.readthedocs.io>. [Online] openAi. [Zitat vom: 1. Oktober 2024.]
<https://stable-baselines3.readthedocs.io/en/master/modules/sac.html>.
18. Medium. *RL - Tips on Reinforcement Learning*. [Online] [Zitat vom: 4. Oktober 2024.] <https://jonathan-hui.medium.com/rl-tips-on-reinforcement-learning-fbd12111775>.

19. Wikipedia. *Temporal Difference Learning*. [Online]
https://de.wikipedia.org/wiki/Temporal_Difference_Learning.
20. Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arxiv.org*. [Online] [Zitat vom: 4. Oktober 2024.] <https://arxiv.org/abs/1801.01290>.
21. ST3215 Servo. *Waveshare*. [Online] [Zitat vom: 3. Oktober 2024.]
https://www.waveshare.com/wiki/ST3215_Servo.
22. ESP32 Serial Bus Servo Driver Expansion Board. *bastelgarage*. [Online] [Zitat vom: 3. Oktober 2024.]
<https://www.bastelgarage.ch/esp32-serial-bus-servo-driver-expansion-board-1-2251?search=ESP32%20servo%20bus>.
23. An API standard for reinforcement learning with a diverse collection of reference environments. *Gymnasium*. [Online] [Zitat vom: 3. Oktober 2024.] <https://gymnasium.farama.org/index.html>.
24. Servo Driver with ESP32. *Waveshare*. [Online] [Zitat vom: 3. Oktober 2024.]
https://www.waveshare.com/wiki/Servo_Driver_with_ESP32.
25. pybullet. *Bullet Real-Time Physics Simulation*. [Online] [Zitat vom: 3. Oktober 2024.]
<https://pybullet.org/wordpress/>.
26. github. *fusion2urdf*. [Online] [Zitat vom: 3. Oktober 2024.] <https://github.com/syuntoku14/fusion2urdf>.
27. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *arxiv*. [Online] [Zitat vom: 3. Oktober 2024.] <https://arxiv.org/abs/1804.02717>.
28. Learning Agile Robotic Locomotion Skills by Imitating Animals. *arxiv*. [Online] [Zitat vom: 2. Oktober 2024.] <https://arxiv.org/abs/2004.00784>.
29. Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel. Deepmimic: Example-guided deep reinforcement. *ACM Trans. Graph.* [Online] [Zitat vom: 2. Oktober 2024.]
<http://doi.acm.org/10.1145/3197517.3201311>.
30. Wildkaninchen. *wildhueter*. [Online] [Zitat vom: 3. Oktober 2024.] <https://www.wildhueter-st-hubertus.de/wildkaninchen-2>.
31. Value Iteration and Q-learning. *core-robotics.gatech.edu*. [Online] <https://core-robotics.gatech.edu/2021/01/19/bootcamp-summer-2020-week-3-value-iteration-and-q-learning/>.
32. Reinforcement Learning with Stable Baselines 3. *Youtube*. [Online]
<https://youtube.com/playlist?list=PLQVvvaa0QuDf0O2DWwLZBfJeYY-JOeZB1&si=RBlpIW48cYgfkYXe>.
33. Physics Simulator w/ Robot Dog. *Youtube*. [Online]
<https://youtube.com/playlist?list=PLQVvvaa0QuDenVbxP4LXYZoGbjfgP-Y5i&si=YSih4fM3drSwvnIx>.

8 Abbildungsverzeichnis

Abb. 1 ILLUSTRIERUNG DES LERNPROZESSES (EIGENE ABBILDUNG)	6
Abb. 2 ILLUSTRIERUNG EINES TYPISCHEN POLICY-NETZWERKES. (EIGENE ABBILDUNG)	7
Abb. 3 SKIZZE DES RÖNTGENBILDES MIT BERECHNETEN LÄNGENANGABEN.	13
Abb. 4 SKIZZE DER DREI SEKTIONEN DES ROBOTER-KANINCHENS (EIGENE ABBILDUNG)	14
Abb. 5 SKIZZE DER HINTERLÄUFE	15
Abb. 6 SKIZZE DER HINTERLÄUFE	15
Abb. 7 SKIZZE MIT HINTERLÄUFEN UND VORDERLÄUFEN	16
Abb. 8 SKIZZE DES HINTERBEINS (EIGENE ABBILDUNG)	17
Abb. 9 GEWÄHLTE STRUKTUR ZUR ANSTEUERUNG DES ROBOTERS (22)	19
Abb. 10 DAS ROBOTER-KANINCHEN ZUSAMMENGEBAUT MIT DEN SERVOS IN FUSION360.(EIGENE ABBILDUNG)	22
Abb. 11 3D GEDRUCKTE TEILE ZUSAMMENGEBAUT ZUM ROBOTER-KANINCHEN (EIGENE ABBILDUNG).....	22
Abb. 12 BYBULLED LINKS MIT ERGÄNZTEM BENUTZER INTERFACE. (EIGENE ABBILDUNG)	27
Abb. 13 DARSTELLUNG DER REWARD VS. DISTANZEN ZWISCHEN DEM EXPERTEN UND DES LERNENDEN AGENTEN. (EIGENE ABBILDUNG)	29
Abb. 14 GRAPHISCHE DARSTELLUNG DER SAC-ARCHITEKTUR. (EIGENE ABBILDUNG)	33
Abb. 15 HILFSTOOL, FÜR ANWENDEN DER GELERNTEN AGENTENBEWEGUNGEN VERGLEICH MIT DEN EXPERTEN DATEN. (EIGENE ABBILDUNG)	34
Abb. 16 ZUSAMMENGEZOGENE HALTUNG	36
Abb. 17 GESTRECKTE HALTUNG	36
Abb. 18 LEICHT LINKSGEKRÜMMTE HALTUNG	36
Abb. 19 STARKE RECHTSGEKRÜMMTE HALTUNG	36
Abb. 20 BRÜCHIGERE VERSIONEN DER HINTERLÄUFE.	37
Abb. 21 REWARD VON PPO UND SAC ÜBER 10Mio STEPS.	38
Abb. 22 RELATIVE REWARD VON PPO UND SAC	38
Abb. 23 AGENT IN FLACHEN UMGEBUNG 1	39
Abb. 24 AGENT IN MITTEL UNEBENER UMGEBUNG 2	39
Abb. 25 AGENT IN SEHR UNEBENER UMGEBUNG 3	39
Abb. 26 LERNKURVE DER AGENTEN IN DEN DREI VERSCHIEDENEN UMGEBUNGEN	40
Abb. 27 VERGLEICH DER DURCHSCHNITTlich ERHALTENEN BELOHNUNG DER AGENTEN AUS DEN VERSCHIEDENEN UMGEBUNGEN	41
Abb. 28 VERGLEICH DER DURCHSCHNITTlichen ÜBERLEBENSDAUER DER AGENTEN AUS DEN VERSCHIEDENEN UMGEBUNGEN	41

9 Anhang

9.1 Komplette Beschreibung der Reward-Funktion

Durch diese gaussische Funktion wurden mehrere Teil-Reward-Funktionen erstellt, die schliesslich summiert werden, um den Gesamt-Reward r_t zu bilden. Der Gesamt-Reward wird bei jedem Zeitschritt wie folgt berechnet:

$$r_t = w_p \cdot r_t^p + w_v \cdot r_t^v + w_e \cdot r_t^e + w_t \cdot r_t^t + w_{rp} \cdot r_t^{rp} + w_{rv} \cdot r_t^{rv} + w_a \cdot r_t^{wa} + w_{lt} \cdot r_t^{pw} \quad (9.1)$$

Die Gewichte der einzelnen Teil-Reward-Funktionen sind wie folgt definiert:

$$w_p = 0.5, \quad w_v = 0.05, \quad w_e = 0.15, \quad w_t = 0.05, \quad w_{rp} = 0.1, \quad w_{rv} = 0.05, \quad w_a = 0.15, \quad w_{lt} = 0.05 \quad (9.2)$$

9.1.1 Pose Reward r_t^p :

Der Pose Reward ermutigt den Agenten, die Differenz der Gelenkwinkelpositionen zwischen dem Lernenden Roboter und dem Experten so klein wie möglich zu halten. Die Differenz zwischen den Winkelpositionen des Experten \hat{q}_t^j und des Agenten q_t^j wird berechnet und durch 2π geteilt, um die relative Differenz der Winkel in Radianten zu bestimmen. Die quadrierte Differenz wird als Exponent in der Exponentialfunktion genutzt:

$$r_t^p = \exp \left(-20 \cdot \frac{\sum_j (q_t^j - \hat{q}_t^j)^2}{(2\pi)^2} \right) \quad (9.3)$$

9.1.2 Velocity Reward r_t^v :

Der Velocity-Reward wird auf ähnliche Weise berechnet. Er bewertet die Differenz zwischen den Winkelgeschwindigkeiten \dot{q}_t^j des lernenden Agenten und $\hat{\dot{q}}_t^j$ des Experten. Auch hier wird die Differenz quadriert und in die Exponentialfunktion integriert, um eine glatte Belohnungskurve zu erzeugen, wobei $b = -0.5$ gewählt wurde.

$$r_t^v = \exp \left(-0.5 \cdot \frac{\sum_j (\dot{q}_t^j - \hat{\dot{q}}_t^j)^2}{(2\pi)^2} \right) \quad (9.4)$$

9.1.3 End-Effector Reward r_t^e :

Der End-Effector-Reward zielt darauf ab, die Distanz zwischen den End-Effektor-Positionen des Experten und des Agenten zu minimieren. Die Distanz wird als Differenz der Koordinaten der relevanten Bauteile des Experten \hat{x}_e^t und des Agenten x_e^t berechnet. Die Fallgeschwindigkeit wurde hier steil mit $b = -20$ gewählt:

$$r_t^e = \exp \left(-20 \cdot \sum_e (\hat{x}_e^t - x_e^t)^2 \right) \quad (9.5)$$

9.1.4 Joint Torque Reward r_t^t :

Der Joint-Torque-Reward ermutigt den Agenten, einen ähnlichen Kraftaufwand wie der Experte aufzuwenden. Als Distanzwert d wird die Torquedifferenz zwischen dem Experten $\hat{\tau}_t^j$ und dem lernenden Agenten τ_t^j verwendet. Die Fallgeschwindigkeit wurde hier glatt mit $b = -0.5$ gewählt:

$$r_t^t = \exp \left(-0.5 \cdot \sum_j (\tau_t^j - \hat{\tau}_t^j)^2 \right) \quad (9.6)$$

9.1.5 Root Pose Reward r_t^{rp} :

Der Root-Pose-Reward minimiert die Raumdistanz und Winkeldifferenz zwischen den Root-Punkt des Agenten (x_t^{root}, q_t^{root}) und des Experten ($\hat{x}_t^{root}, \hat{q}_t^{root}$). Die Fallgeschwindigkeit b der Kurve liegen bei -40 für die Raumdistanz und bei -20 für die Winkeldifferenz:

$$r_{rp}^t = \exp \left(-40 \cdot \|\hat{x}_{root}^t - x_{root}^t\|^2 - 20 \cdot \left(\frac{\|\hat{q}_{root}^t - q_{root}^t\|}{2\pi} \right)^2 \right) \quad (9.7)$$

9.1.6 Root Velocity Reward r_t^{rv} :

Der Root-Velocity-Reward bewertet die Übereinstimmung Positions- und Euler-Geschwindigkeiten zwischen dem lernenden Agenten ($\dot{x}_t^{root}, \dot{q}_t^{root}$) und dem Experten ($\hat{\dot{x}}_t^{root}, \hat{\dot{q}}_t^{root}$). Hier wurde eine flache Fallgeschwindigkeit $b = -0.5$ gewählt.

$$r_{rv}^t = \exp \left(-0.5 \cdot \|\hat{\dot{x}}_{\text{root}}^t - \dot{x}_{\text{root}}^t\|^2 - 0.5 \cdot \left(\frac{\|\hat{\dot{q}}_{\text{root}}^t - \dot{q}_{\text{root}}^t\|}{2\pi} \right)^2 \right) \quad (9.8)$$

9.1.7 Action-Reward r_t^{wa} :

Die Aktionen des Agenten a_t werden mit denen des Experten \hat{a}_t verglichen. Die Fallgeschwindigkeit wurde hier steil mit $b = -5$ gewählt.

$$r_t^{wa} = \exp(-5 \cdot \|\hat{a}_t - a_t\|^2) \quad (9.9)$$

9.1.8 Power-Reward r_t^{pw} :

Der Power-Reward belohnt den Agenten dafür, sparsam mit seiner Energie umzugehen. Die Leistung wird als Produkt des Drehmoments τ_t^j und der Winkelgeschwindigkeit \dot{q}_t^j der Servogelenke berechnet. Eine sehr flache Fallgeschwindigkeit $b = -0.01$ wurde gewählt.

$$r_t^{pw} = \exp \left(-0.1 \cdot \sum_j (\tau_t^j \cdot \dot{q}_t^j)^2 \right) \quad (9.10)$$

9.2 Hyperparameter

Im Folgenden sind die fast identischen Hyperparameter aufgelistet, um die Vergleichbarkeit zwischen PPO und SAC zu gewährleisten:

Parameter	Value SAC	Value PPO
Default Parameters	Standard SAC	Standard PPO
Learning_rate	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
Batch_size	10'000	10'000
Buffer_size	500'000	-
policy_kwargs	pi = [512, 256], qf=[512, 256]	pi = [512, 256], vf: [512, 256]
Gamma	0.95	0.95
Number fo CPUs	10	10

Tabelle 6 Verwendete Hyperparameter von PPO und SAC.

Eigenständigkeitserklärung

Ich erkläre hiermit, dass ...

... diese Arbeit weder ganz oder teilweise abgeschrieben noch kopiert, aus dem Internet oder von einem KI-Tool übernommen wurde.

... der Quellenachweis gemäss den Vorgaben des Handbuches Projekte korrekt und vollständig ist.

... die dargestellten Daten und Resultate von den Unterzeichnenden selbst und gemäss den Vorgaben des Handbuches Projekte erhoben und verarbeitet wurden.

Niederlenz, 16.Oktober 2024

Kevin Leutwyler