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

Baran Demir

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

30. November 2023

Outline

Einleitung [1] [2]

- Entwicklung quadrupedaler Robotik im Vormarsch
- Inspektion industrieller Anlagen wie Bohrinseln (ANYbotics)
- Exploration von Tunneln (DARPA Subterranean Challenge)
- Die Welt ist chaotisch. Viele Situationen und unvorhersehbare Störungen
- Wie kann korrektes Verhalten auch bei unerwarteten Zuständen ermöglicht werden?
- Was ist zu tun, wenn der Roboter fällt? Eingreifen eines menschlichen Operateurs vernichtet den Zweck autonomer Systeme

Einleitung - Aufgabenstellung [3] [4]

- **Ziel:**
  - Erlernen verschiedener Bewegungen mit Deep Reinforcement Learning
  - Zusammenfassen der Bewegungen in einen Controller
  - Automatisierte Falldetektion mit anschließender Aufstehfunktion
  - Konzept maßgeblich inspiriert von [Lee u. a. 2019] („Robust Recovery Controller“)
  - Evaluation verschiedener Steuerungen und Konfigurationen

- Betrachtete Steuerungen:
  - **Direkte Drehmomentregelung („Torque Control“)**: Ausgabe des neuronalen Netzwerkes wird direkt auf Gelenke angewendet
  - **PD-Regler**: Ausgabe des neuronalen Netzwerkes ist Zielwinkel $\hat{\phi}^i$ welcher von PD-Regler auf Drehmoment abgebildet wird.

Einleitung - Lösungsansatz

- Training im Simulationsprogramm Isaac-Sim unter Verwendung des Programmiergerüsts Isaac-Gym
- Das Training jeder Bewegung ist durch drei Dateien definiert
- Eine Datei unitreea1_bewegung.py um das Programmiergerüst Gym zu implementieren (Anwenden der Aktion, Erfassen des Zustandes, Belohnungsfunktion)
- Eine Datei UnitreeA1Bewegung.yaml, um allgemeine Simulationsparameter zu konfigurieren (Initialisierungsinformationen, Gewichte für die Belohnungsfunktion, Steuerung, Episodendauer)
- Eine Datei UnitreeA1BewegungPPO.yaml, um den Trainingsalgorithmus zu konfigurieren

Deep Reinforcement Learning - MDP [5]

- Markow-Entscheidungsproblem (MDP) ist formale Grundlage des Reinforcement Learnings
- Zustandsmenge $\mathcal{S}$
- Aktionsmenge $\mathcal{A}$
- Zustandsübergangsfunktion $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
- Belohnungsfunktion $r : S \to \mathbb{R}$
- Wahrscheinlichkeitsverteilung für den Anfangszustand $\rho_0 : \mathcal{S} \to \mathbb{R}$
- Diskontierungsfaktor $\gamma \in (0, 1)$
- Policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$
- Diskontierter erwarteter Gewinn $\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right]$

Deep Reinforcement Learning - MDP

- Zustand-Wert-Funktion:

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

- Zustand-Aktion-Wert-Funktion:

$$Q^{\pi,\gamma}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right] \tag{2}$$

- Advantage-Funktion $A^{\pi,\gamma}$ setzt die Funktionen ins Verhältnis zueinander:

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t) \tag{3}$$

Deep Reinforcement Learning - Deep Q-Learning [6]

- Zustands- und Aktionsraum hochdimensional und kontinuierlich
- Look-Up Tabelle nicht möglich $\Longrightarrow$ Policy π als neuronales Netzwerk
- Deep RL erlangte große Aufmerksamkeit durch Deep Q-Learning
- Idee: Bellmansche Optimalitätsgleichung approximieren:

$$Q_*(s_t, a_t) = \mathbb{E}\left[r(s_t) + \gamma \max_{a'} Q_*(s_{t+1}, a')|s_t, a_t\right] \tag{4}$$

- Minimierung der Loss Funktion:

$$L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s_t, a_t; \theta_i))^2\right] \tag{5}$$

- Zielwert y_i so festgelegt, dass Bellmansche Optimalitätsgleichung approximiert wird:

$$y_i = \mathbb{E}\left[r(s_t) + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{i-1}|s_t, a_t)\right] \tag{6}$$

Deep Reinforcement Learning - Deep Q-Learning

- Aktionsselektion als Optimierungsproblem

$$a = \arg\max_a Q^*(s_t, a; \theta) \tag{7}$$

- Zustands- und Aktionsraum sehr groß, deshalb suboptimal
- Verbesserungsmöglichkeit: Aktionsraum diskretisieren
- Nachteil: Verlust optimaler Aktion
- Nachteil: Noch immer schwierig zu lösen
- Beispiel: Unitree A1 Roboter mit $\pm 33 Nm$ auf jedem Gelenk als diskretisierter Aktionsraum:

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

- Es ergeben sich mit zwölf Freiheitsgraden $|\text{A1}|^{12} = 661^{12}$ potenzielle Aktionen

Deep Reinforcement Learning - Policy Gradient [7]

- Policy Gradient Methoden berechnen die Aktion als Wahrscheinlichkeit $\pi(a_t, s_t, \theta_t)$
- Policy Gradient allgemein definiert als

$$\nabla_\theta \eta(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{8}$$

  mit der Aktualisierungsregel

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \eta(\pi_{\theta_k}) \tag{9}$$

- Mögliche Ausprägung als $\Psi_t = A^{\pi,\gamma}(s_t, a_t)$
- Falsche Wahl von α im Reinforcement Learning besonders problematisch

Deep Reinforcement Learning - Policy Gradient [5]

- Trainingsdaten werden während des Trainings erzeugt und hängen von Parametrisierung θ ab
- Parametrisierung θ wird verändert $\implies$ Verteilung auftretender Zustände und Belohnungen wird verändert
- Ungünstige Veränderung der Parametrisierung θ $\implies$ Agent nun in irrelevantem Zustandsraum
- Nächster Aktualisierungsschritt basiert auf unter ungünstiger Policy π_θ erzeugten Trainingsdaten
- Weiterer Verlauf des Trainings ungewiss. Möglicherweise Divergenz

Deep Reinforcement Learning - TRPO [5]

- Trust Region Policy Optimization (TRPO) umgeht das Problem durch eine feste Schranke

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \tag{10}$$

$$\text{beschränkt durch } \mathbb{E}_{s\sim\pi_{\theta_k}} \left[D_{KL} \left[\pi_\theta(\cdot|s), \pi_{\theta_k}(\cdot|s) \right] \right] \leq \delta \tag{11}$$

- $\mathcal{L}(\theta_k, \theta)$ misst die Performanz der Policy π_θ relativ zu der alten Policy π_{θ_k} unter Daten der alten Policy π_{θ_k}

$$\mathcal{L}(\theta_k, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_k}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k},\gamma}(s,a) \right] \tag{12}$$

Deep Reinforcement Learning - PPO [8]

- Proximal Policy Optimization sei einfacher (bezüglich Verständnis, Lösung und Implementierung):

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta) \right] \tag{13}$$

- mit

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}\gamma}(s, a), \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k},\gamma}(s, a) \right) \tag{14}$$

- Beispiel: Sei $A^{\pi_{\theta_k},\gamma}(s, a)$ positiv. Der Term $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}\gamma}(s, a)$ wird größer, wenn $\pi_\theta(a|s)$ größer wird.
- Uneingeschränktes Verändern der Parameter θ kann zu Divergenz führen
- Lösung: Einschränken der Veränderung von θ durch Clip-Parameter ϵ
- Wenn $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} > 1 + \epsilon$ dann keinen Einfluss mehr auf Zielfunktion wegen des min-Terms

Bewegungsablauf Hinsetzen - Zustand

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

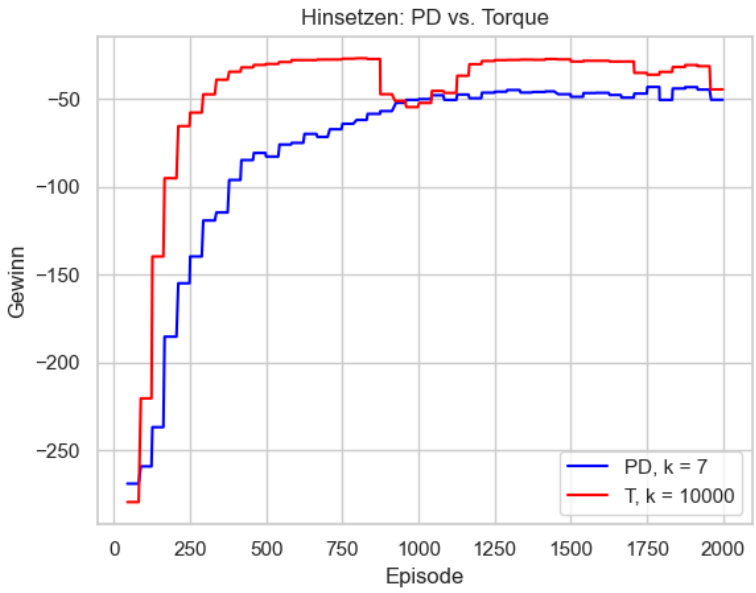
- Roboter wird in der Simulation aus der Luft abgeworfen
- Gelenkkonfiguration wird randomisiert im Rahmen der Gelenkgrenzen
- Rücken zum Boden ausgerichtet
- Ziel: Sequenz von Aktionen ausführen, um eine Sitzkonfiguration zu erreichen
- Roboter soll aus der Sitzkonfiguration aufstehen können

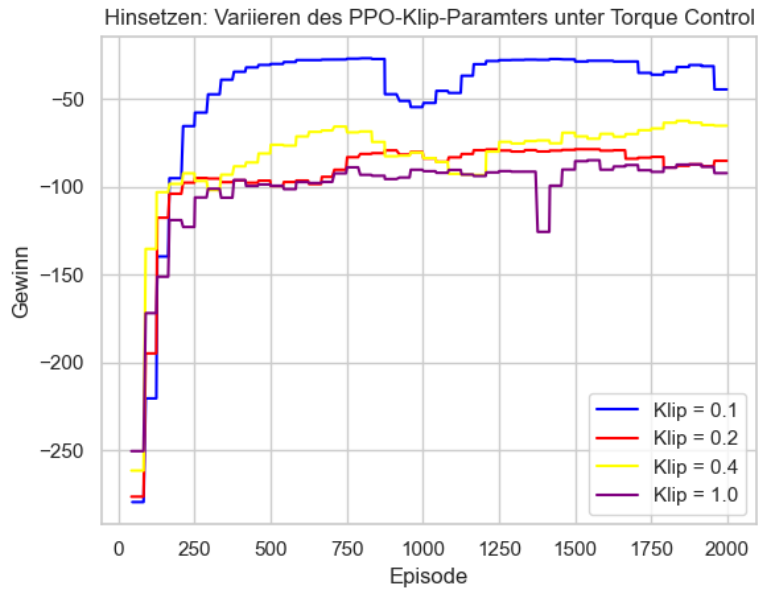
Bewegungsablauf Hinsetzen - Belohnungsfunktion [9]

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

- Kurrikulum-Faktor k_c für adaptive Gewichtung sekundärer Ziele
- Zu Beginn $k_{c,0} = 0.3$
- Danach Aktualisierung $k_{c,j+1} \leftarrow (k_{c,j})^{k_d}$, mit $k_d = 0.997$
- Folge konvergiert gegen 1 $\implies$ Strafterme werden mit fortlaufendem Training stärker

Bewegungsablauf Hinsetzen - Evaluation





- Direkte Drehmomentregelung schneidet besser ab bei dieser Aufgabe. Training jedoch instabiler
- Die Belegung $\epsilon = 0.1$ schneidet signifikant besser ab als andere Belegungen

Bewegungsablauf Aufstehen - Zustand

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

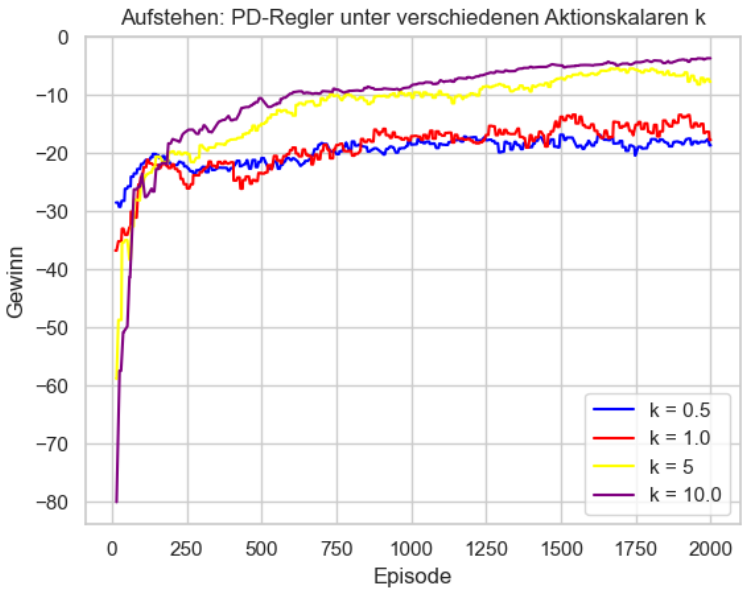
- Roboter startet aus Sitzkonfiguration
- Leichte Randomisierung der Gelenkkonfiguration und Gelenkgeschwindigkeit
- Ziel: Sequenz von Aktionen ausführen, um eine Standkonfiguration zu erreichen, aus der der Roboter laufen kann.
- Unterschied zum Hinsetzen: Höhe in die Zustandsdarstellung inkludiert

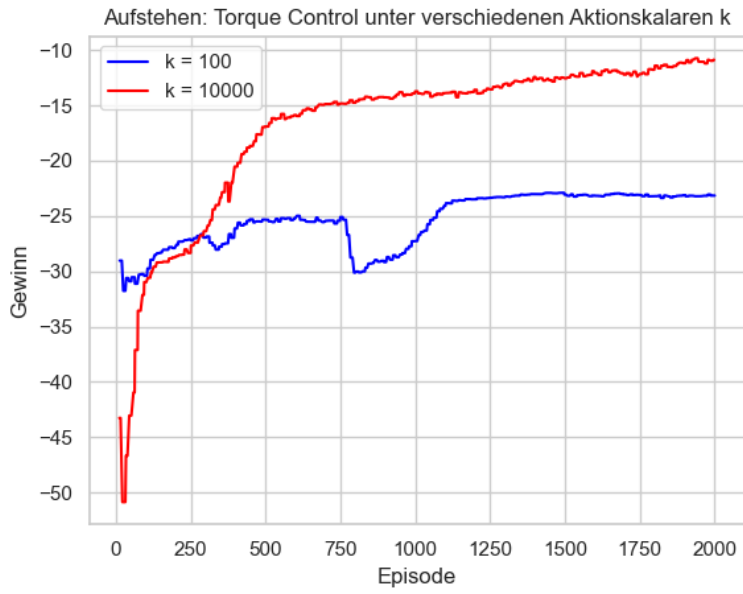
Bewegungsablauf Aufstehen - Belohnungsfunktion

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

- Nahezu identisch zum Hinsetzen. Gewicht für die Abweichung von -2.0 auf -1.0 betragsmäßig abgesenkt
- Keine Fußkontaktsensoren. Beanspruchen Speicher und Rechenleistung
- Höhe nicht in Belohnungsfunktion miteinbezogen. Resultierte in fehlerhaftem Stand

Bewegungsablauf Aufstehen - Evaluation





- **Direkte Drehmomentregelung:** Zu kleines Aktionsskalar k führt dazu, dass Roboter zu wenig Kraft zum Aufstehen hat. Schwierigere Wahl, weil Wertebereich größer.
- **PD-Regler:** Erzielter Gewinn besser, aber Bewegung ist instabiler

Bewegungsablauf Laufen - ALDURO [10]

- **ALDURO: A**nthropomorphically **L**egged and Wheeled **Du**isburg **Ro**bot
- Laufender Bagger mit 16 Freiheitsgraden. Schwierig manuell zu steuern
- Vorgeschlagene Lösung: Automatisierung der Steuerung durch unabhängige Module
- **Gait Generator:** Generieren der Gangart
- **Static Stability:** Rumpf des Roboters in einer Sicherheitszone halten
- **Obstacle Avoidance:** Entscheiden, ob ein Hindernis durchquert oder vermieden werden soll
- **Body Posture:** Ermöglichen einer Standposition

Bewegungsablauf Laufen - ALDURO

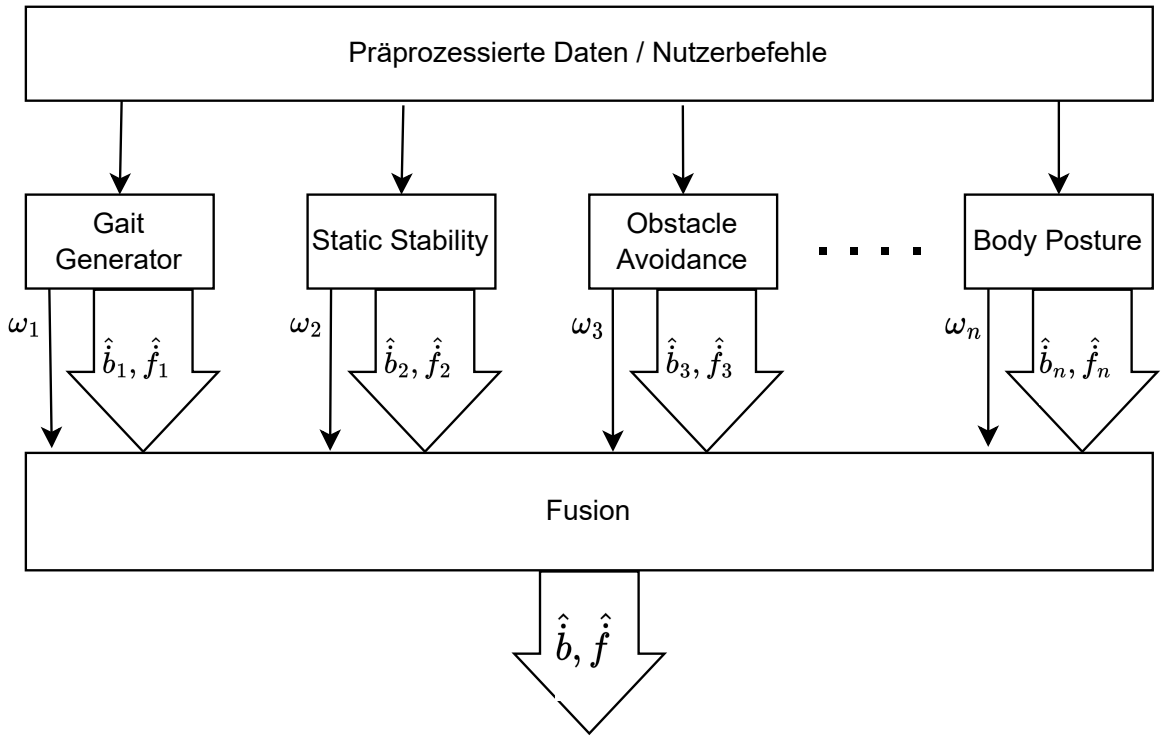


Abbildung 3: Nachgezeichnet entnommen aus [Germann u. a. 2005]

Bewegungsablauf Laufen - ALDURO

- Ausgabe jedes Moduls sind Körpergeschwindigkeiten $\hat{\dot{b}}_m$ und Fußgeschwindigkeiten $\hat{\dot{f}}_m$
- Module kommunizieren nicht miteinander. Koordination ausschließlich über Anpassung der Gewichte
- Einzelausgaben der n Module werden zusammengefasst als

$$\left[\hat{\dot{b}}^T, \hat{\dot{f}}^T\right]^T = \frac{1}{n} \sum_{m=1}^{n} w_m \left[\hat{\dot{b}}_m^T, \hat{\dot{f}}_m^T\right]^T \tag{15}$$

- Nachteil: Einsetzen des sicherheitskritischen Moduls hängt von Wahl der Gewichte w ab
- Nachteil: Bein-Selektions-Algorithmus nach Publikation Deadlock anfällig. Lösung über Sonderregel

Bewegungsablauf Laufen

Controllers presented in this paper, trained in a few hours in simulation, outperformed the best existing model-based controller running on the same robot, which were designed and tuned over many years. Our learned locomotion policies ran faster and with higher precision while using less energy, torque, and computation. (Hwangbo u. a. 2019)

Bewegungsablauf Laufen - Zustand

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

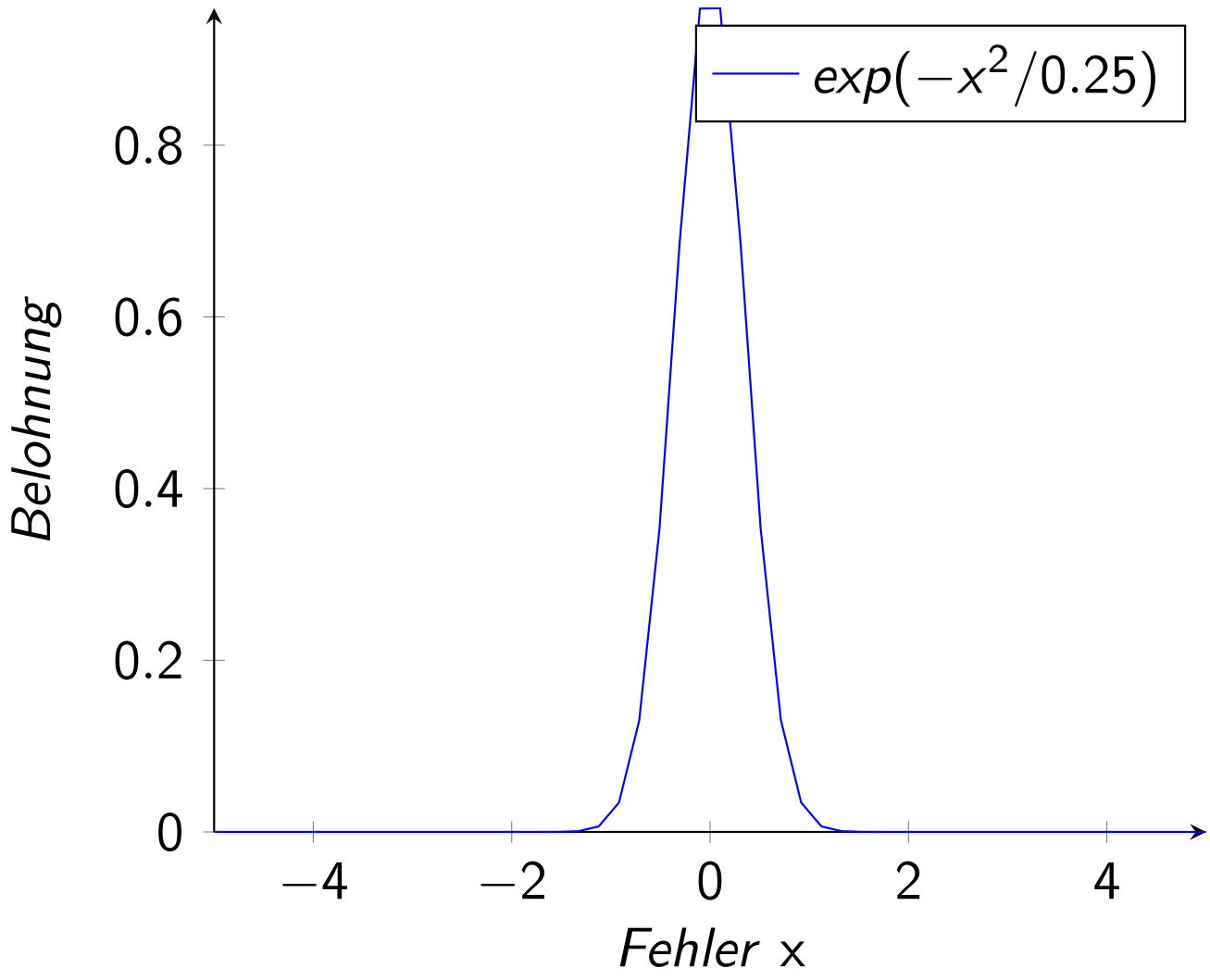
- Geschwindigkeitsbefehl als lineare Geschwindigkeit in die x- und y-Richtung $v^*_{linVel,xy}$ und Drehrate $\omega^*_{angVel,z}$ entlang der z-Achse
- Geschwindigkeitsbefehle werden bei jeder Episode neu bestimmt
- Episode endet, wenn Zeitlimit erreicht wird oder wenn Roboter zu niedrig ist
- Roboter wird während des Trainings randomisiert geschubst

Bewegungsablauf Laufen [11][9]

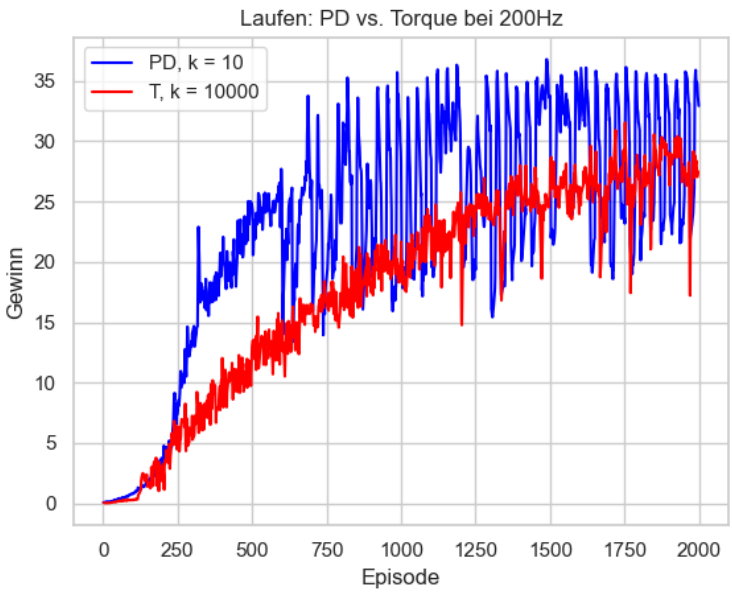
Belohnungsfunktion		
Metrik	Term	Gewicht
Lineargeschwindigkeit x-y	$\exp(-\|\|v^*_{linVel,xy} - v_{linVel,xy}\|\|^2/0.25)$	2.0
Rotationsgeschwindigkeit z	$\exp(-(\omega^*_{angVel,z} - \omega_{angVel,z})^2/0.25)$	0.5
Lineargeschwindigkeit z	$-v^2_{linVel,z}$	$-1.2k_c$
Rotationsgeschwindigkeit x-y	$-\|\|\omega_{angVel,xy}\|\|^2$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

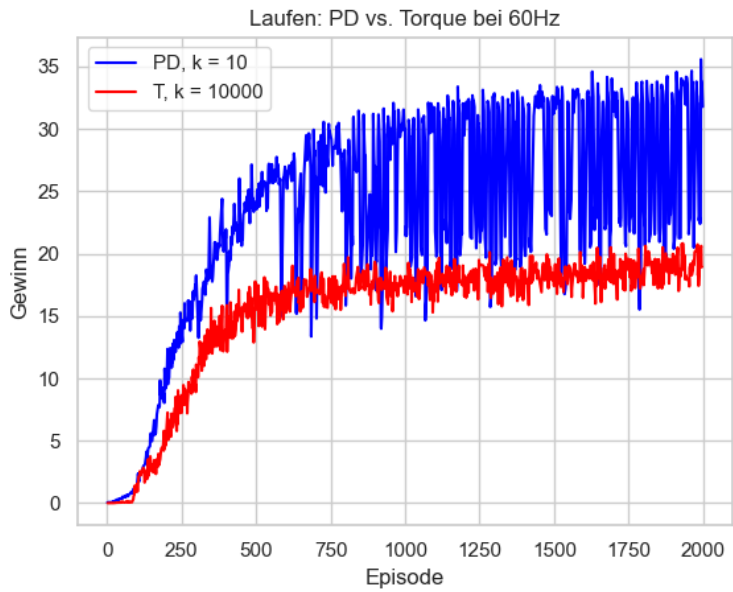
- **Lineargeschwindigkeit z** und **Rotationsgeschwindigkeit x-y** dienen der Stabilität
- Durch **Foot Clearance** Höchstgeschwindigkeit des Fußes zum Ende der Schwungphase
- **Abweichung** bestraft Entfernung von der Standkonfiguration
- **Lineargeschwindigkeit x-y** und **Rotationsgeschwindigkeit z** messen den Fehler zwischen Soll- und Ist-Geschwindigkeit. Keine Bestrafung, sondern beschränkte Belohnung

Bewegungsablauf Laufen



Bewegungsablauf Laufen - Evaluation





Bewegungsablauf Laufen - Evaluation

- Erzielter maximaler Gewinn:
  - **PD-Regler:** 35.58 (60Hz) und 36.78 (200Hz). Performanz-Erhalt von 96.74%
  - **Direkte Drehmomentregelung:** 20.83 (60Hz) und 31.53 (200Hz). Performanz-Erhalt von 66.06%
- Anzahl der Stürze nach Training über 10 Episoden mit 32 Robotern:
  - **PD-Regler:** 143
  - **Direkte Drehmomentregelung:** 101
- Zielwinkel werden in lokaler Umgebung der Standkonfiguration konstruiert als: $\phi_d = ko_t + \phi_n$.
- Möglich, dass direkte Drehmomentregelung robuster ist.
- **Aber:** PD-geregelter Roboter läuft schneller. Entsprechend anfälliger für einen Sturz
- Video: https://youtu.be/O-dw0-IyGEE

Zusammenführung der Bewegungsabläufe - FSM

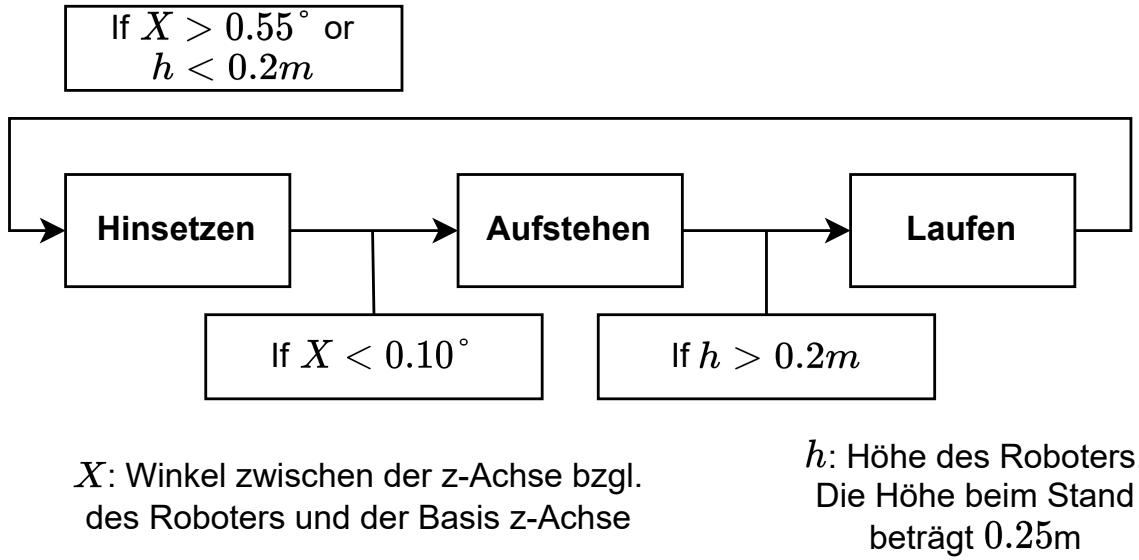


Abbildung 4: Modifiziert entnommen aus [Lee u. a. 2019]

- Separierung in einzelne Teilbewegungen, weil Belohnungsfunktionen sich widersprechen
- Implementierung über endlichen Automaten
- Übergang Hinsetzen $\rightarrow$ Aufstehen funktioniert
- Übergang Aufstehen $\rightarrow$ Laufen funktioniert nicht
- Video: https://youtu.be/FdP7DghXz5g

Zusammenführung der Bewegungsabläufe - neuronales Netz [12]

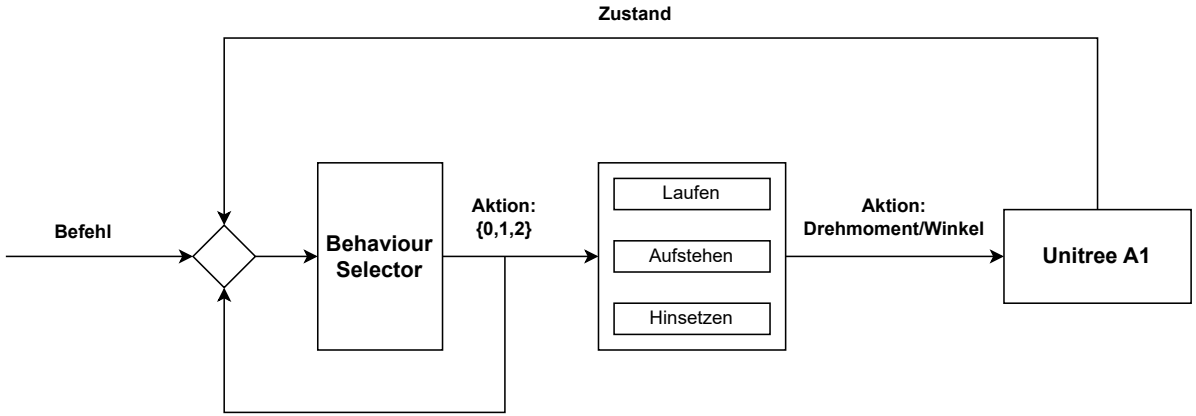


Abbildung 5: Simplifiziert entnommen aus [Lee u. a. 2019]

- Auswahl der Teilbewegung wird als Policy π_θ aufgefasst
- Zustand ist Vereinigung der Zustände der Teilbewegungen
- Belohnungsfunktion ist (nahezu) die des Laufens
- Ausgabe von π_θ ist $\{p_0, p_1, p_2\}$. Jeder Eintrag gibt die Wahrscheinlichkeit an, in einem Zustand s ein Teilmodul $a \in \{0, 1, 2\}$ auszuführen
- Aktionsselektion durch $a_t \sim \pi_\theta(a|s_t)$

Sim2Real Gap [12] [13] [14]

- Echte Welt nicht exakt wie die Simulation. Problem ist die Sim-Zu-Real Kluft (Literatur: Sim2Real Gap)
- Domain Randomization:
  - Policy robuster machen durch Randomisieren verschiedener Parameter (Reibungskoeffizienten, Steifheit von Gelenken, Latenzen usw.)
  - Artifizielles Verrauschen von Beobachtungen und Aktionen.
- Aktuatormodell:
  - Verhalten eines Aktuators weicht aufgrund interner Eigenschaften eventuell von Befehl ab
  - Analytisches Modell schwierig
  - Aktuatormodell trainieren, um Aktuatorverhalten basierend auf Befehl zu prognostizieren
- Kollisionen bestrafen

Sim2Real Gap - State estimation [12]

- Zustand in Simulation muss in echter Welt verfügbar sein
- Beispiel: Schätzung der Höhe nach Fall unzuverlässig
- Lösung: Ausnutzung der Simulation, um Höhe durch neuronales Netz h_ψ zu schätzen
- Während des Trainings Paare (s_t, h_t) speichern. Paare sind bekannt aus der Simulation
- Im Aktualisierungsschritt K Paare aus dem Speicher samplen und folgende Funktion minimieren

$$\sum_{j=0}^{K} ||h_j - h_\psi(s_j)||^2 \tag{16}$$

Sim2Real Gap - Rapid Motor Adaptation [15]

- Terrain der echten Welt höchst variabel (Schlamm, Gras, felsig, Sand, Nutzlast). Motorik muss bei Veränderung der Umgebung sofort angepasst werden
- Zustandsdarstellung um nur in der Simulation bekannten enkodierten privilegierten Informationen z_t ergänzen
- Die Information z_t ist in der echten Welt nicht bekannt
- Fehler zwischen Befehl und tatsächlich angenommener Position eines Gelenkes gibt Aufschluss über z_t
- Trainieren eines Adaptations Moduls ϕ_{RMA} das unter Eingabe der Zustands-Aktions-Historie den latenten Vektor z_t schätzt

$$\hat{z}_t = \phi_{RMA}(s_{t-k:t-1}, a_{t-k:t-1}) \tag{17}$$

Sim2Real Gap - Rapid Motor Adaptation

- Naive Variante den Datensatz für das Training von ϕ_{RMA} zu generieren: Bereits trainierte Base-Policy π ausführen mit wahrem z_t
- Aktionsselektion als $a = \pi(s_t, a_{t-1}, \mathrm{z_t})$ und Paare $((s_{t-k:t-1}, a_{t-k:t-1}), z_t)$ speichern.
- Problem: Agent sieht nur erfolgreiche Trajektorien!
- Robustere Variante wählt Aktion als $a = \pi(s_t, a_{t-1}, \hat{\mathrm{z}}_\mathrm{t})$ um Paare $((s_{t-k:t-1}, a_{t-k:t-1}), z_t)$ zu speichern
- Agent wird vom Standardfall abweichende Trajektorien sehen, weil ϕ_{RMA} randomisiert initialisiert wird und die Schätzungen $\hat{z}_t$ zu Beginn unpräzise sind

Sim2Real Gap - Rapid Motor Adaptation

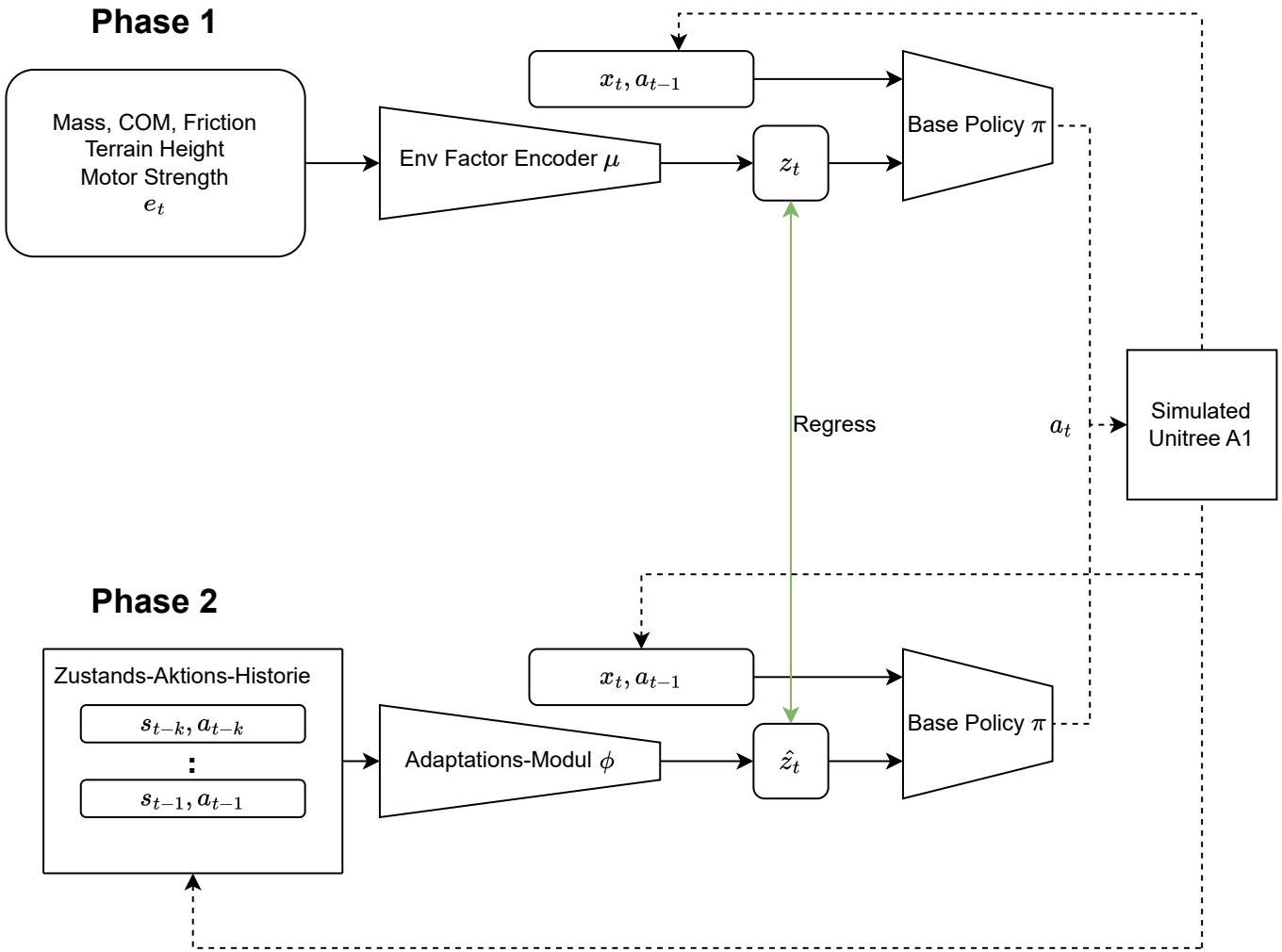


Abbildung 6: Rapid Motor Adaptation. Abbildung modifiziert nachgezeichnet bzw. entnommen aus [Kumar u. a. 2021]

Ausblick [11]

- Bislang gesehene Aktionsräume: Direkte Drehmomentregelung, Zielwinkel und Behavior Selector
- Abstraktere Aktionsräume möglich. Betrachtet werde die Aktion bzw. der Befehlsvektor

$$\omega = (V_{cmd}, p_{f,z}^{\max}, c_{jpos}, k_p, k_d, \phi_n) \tag{18}$$

- Variieren des Vektors $\Longrightarrow$ Variieren der Gangart
- **Low-Level Policy**: Trainieren des Laufens unter verschiedenen Ausprägungen des Befehlsvektors
- **High-Level Policy**: Unter Annahme, dass Training der Low-Level Policy erfolgreich war, wird Befehlsvektor in Abhängigkeit des Zustands und einer „Height Map“ reguliert
- Wenn alle bis hierhin gezeigten Konzepte erfolgreich implementiert worden sind, kann der Roboter laufen (und aufstehen beim Fall). Fokus danach auf kognitive Aufgaben wie Path Planning

Zusammenfassung [16]

- Es ist möglich, komplexe Bewegungsabläufe über Deep Reinforcement Learning zu erlernen
- Optimales (?) Verhalten in allen während des Trainings gesehenen Zuständen
- Plattformunabhängige Belohnungsfunktionen. Belohnungsfunktionen funktionieren für MIT-Cheetah, ANYmal und Unitree A1
- Kein klassisches Expertenwissen notwendig. Frage ist nicht „Wie?“, sondern „Was?“
- Schwierigkeit: Belohnungsfunktion und Zustandsdarstellung
- Studie besagt, dass 92% (von 24 Probanden) aller RL-Experten ihre zuletzt entwickelte Belohnungsfunktion per Trial-und-Error entwarfen
- Auch wenn Terme bekannt sind, ist es schwierig Gewichte richtig einzustellen.

Zusammenfassung

„Wie würdest du deine Bachelorarbeit jemandem ohne technischen Hintergrund erklären?“

Danksagung

Vielen Dank an **Fatih Özgan** für die Betreuung während der gesamten Zeit der Bachelorarbeit und des Bachelorseminars.

Danksagung

Vielen Dank an das Fachgebiet Intelligente Systeme für die Bereitstellung des Rechnerraumes und fürs Zuhören

Referenzen (im Rahmen des Kolloquiums) I

[1] Ian D. Miller u. a. „Mine Tunnel Exploration Using Multiple Quadrupedal Robots“. In: *IEEE Robotics and Automation Letters* 5.2 (2020), S. 2840–2847. DOI: 10.1109/LRA.2020.2972872.

[2] Marco Hutter u. a. „Anymal-a highly mobile and dynamic quadrupedal robot“. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, S. 38–44.

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

[4] Viktor Makoviychuk u. a. „Isaac gym: High performance gpu-based physics simulation for robot learning“. In: *arXiv preprint arXiv:2108.10470* (2021).

Referenzen (im Rahmen des Kolloquiums) II

[5] John Schulman u. a. „Trust region policy optimization“. In: *International conference on machine learning*. PMLR. 2015, S. 1889–1897.

[6] Volodymyr Mnih u. a. „Playing atari with deep reinforcement learning“. In: *arXiv preprint arXiv:1312.5602* (2013).

[7] John Schulman u. a. „High-dimensional continuous control using generalized advantage estimation“. In: *arXiv preprint arXiv:1506.02438* (2015).

[8] John Schulman u. a. „Proximal policy optimization algorithms“. In: *arXiv preprint arXiv:1707.06347* (2017).

[9] Jemin Hwangbo u. a. „Learning agile and dynamic motor skills for legged robots“. In: *Science Robotics* 4.26 (2019), eaau5872.

Referenzen (im Rahmen des Kolloquiums) III

[10] Daniel Germann, Manfred Hiller, Dieter Schramm u. a. „Design and control of the quadruped walking robot ALDURO“. In: *International Symposium on Automation and Robotics in Construction (ISARC)*. 2005.

[11] Michel Aractingi u. a. „A Hierarchical Scheme for Adapting Learned Quadruped Locomotion“. In: (2023).

[12] Joonho Lee, Jemin Hwangbo und Marco Hutter. „Robust recovery controller for a quadrupedal robot using deep reinforcement learning“. In: *arXiv preprint arXiv:1901.07517* (2019).

[13] Xue Bin Peng u. a. „Sim-to-real transfer of robotic control with dynamics randomization“. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, S. 3803–3810.

Referenzen (im Rahmen des Kolloquiums) IV

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

[15] Ashish Kumar u. a. „Rma: Rapid motor adaptation for legged robots“. In: *arXiv preprint arXiv:2107.04034* (2021).

[16] Serena Booth u. a. „The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Bd. 37. 5. 2023, S. 5920–5929.