Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Teamprojekt

RoboCrawler - Ein zwei-armiger selbstlernender Krabbelroboter

Markus Käppeler, Julian Dietsche,
Sebastian Rätzer, Jan Kaiser

Konstanz, 17. Dezember 2019

Zusammenfassung (Abstract)

Thema:	RoboCrawler - Ein zwei-armiger selbstlernender Krabbelroboter	
Autoren:	Markus Käppeler	markus.kaeppeler@htwg-konstanz.de
	Julian Dietsche	julian.dietsche@htwg-konstanz.de
	Sebastian Rätzer	sebastian.raetzer@htwg-konstanz.de
	Jan Kaiser	jan.kaiser@htwg-konstanz.de
Betreuer:	Prof. Dr. Oliver Bittel	bittel@htwg-konstanz.de
	Dipl.-Inf.-Wiss. Jürgen Keppler	juergen.keppler@htwg-konstanz.de

Im Mittelpunkt steht ein selbstlernender Roboter, der in KI- und Robotik-Vorlesungen und -Laboren als Vorführmodell Anwendung finden soll. Das Modell besitzt zwei baugleiche Arme mit jeweils zwei Freiheitsgraden. Der Roboter soll selbständig mit Methoden des Bestärkenden Lernen (Reinforcement Learning) eine Vorwärtsbewegung lernen und ausführen können. Ziel des Projektes ist es, ein Anschauungsmodell für künstliche Intelligenz zu schaffen. Unser Robotermodell wurde inspiriert durch einen Roboter, der mit einem Arm eine Vorwärtsbewegung lernt [1]. Da die Laufzeit des Lernalgorithmus exponentiell von der Anzahl der ansteuerbaren Arme abhängt, ergibt sich eine besondere Herausforderung, für Vorführungszwecke kurze Lernzeiten zu erreichen. Unter diesem Aspekt werden Möglichkeiten zur Traversierung des Zustandsraumes - während der Laufzeit - untersucht, um den Lernprozess zu verkürzen.
Der Roboter kommuniziert über ein Drathlosnetzwerk mit einer Grafischen Benutzeroberfläche, wodurch der Lernalgorithmus veranschaulicht dargestellt wird. Um die Entwicklung sowie das Testen des Algorithmus schneller voranzutreiben gibt es einen realitätsnahen Simulator.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der herkömmlichen Programmierung hat das Computersystem einen Satz an Regeln nach dem es agiert. Im Vergleich dazu beschäftigt sich das bestärkende Lernen mit Methoden des maschinellen Lernens, durch die das System (in unserem Fall ein Roboter) eine eigene Handlungsvorschrift lernt, ohne etwaige Regeln zu kennen.

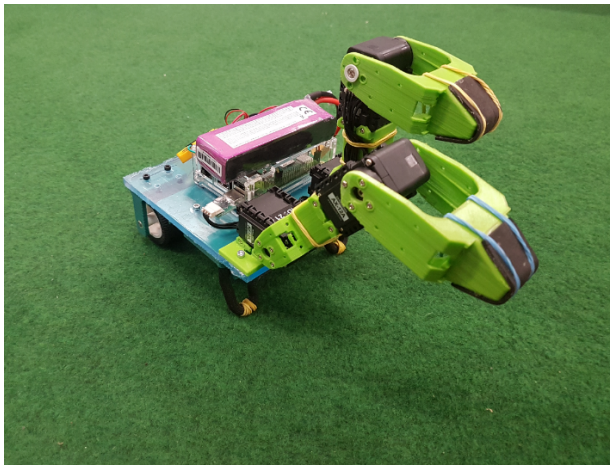

Abbildung 1: Der RoboCrawler auf einem Kunstrasen.

Eine solche Handlungsvorschrift, oder auch Strategie genannt, kann über Erfahrungswerte ermittelt werden, ausschlaggebend hierfür ist die Art und Weise des Feedbacks, welche der Agent (das lernende System) für seine Aktionen erhält. Die Art des erhaltenen Feedbacks - man spricht von der Belohnung - hängt von der Situation und der dazu korrespondierenden Entscheidung des Agenten ab (siehe Abb. 2). Dabei kann das Feedback positiv, negativ oder neutral sein. Im Verlauf der Lernphase approximiert der Agent eine Funktion, welche Zustände auf einen Wert abbildet, die sogenannte Value Function (hier Wertfunktion). Ein Zustand ist hier beispielsweise die Pose der Roboterarme. Aus der Wertfunktion kann der Krabbelroboter eine Verhaltensstrategie ableiten, also eine Regel, welche für jede seiner möglichen Armstellungen bestimmt, welche Folgebewegung die meiste Belohnung verspricht.

In kurz bedeutet dies, dass ein Agent über das Ausprobieren verschiedener Aktionen in einer Vielzahl von Zuständen danach strebt, seine Strategie immer weiter zu optimieren, um langfristig die größtmögliche Belohnung zu erhalten.

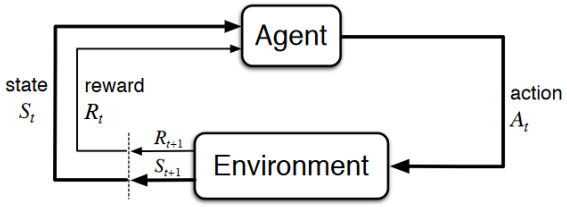

Abbildung 2: Bestärkendes Lernen nach Sutton und Barto.

Für die Robotik ist dies ein wichtiges Ziel, da viele Roboter nicht mehr von Hand eingelernt werden müssen. Ein Greifarmroboter kann von sich aus lernen verschiedene Objekte zu greifen, ohne vorher eine feste Vorschrift zu haben. Oder ein mobiler Staubsaugerroboter kann während der Reinigung eigenständig ermitteln, welche die beste Reinigungsroute ist. In unserem Fall lernt der Roboter - mit Hilfe von Feedbacksignalen - eigenständig zu krabbeln.

Ein Krabbelroboter mit einem Arm ist zunächst etwas trivialer, da der Bewegungsraum stark eingeschränkt ist und die entstehende Strategie meist ähnlich aussieht. Fügt man einen zweiten Arm hinzu, erhöht sich der Zustands- und Aktionsraum um ein Vielfaches, sodass die zu erlernende Bewegung nicht mehr so leicht zu erkunden ist. Außerdem kann die Zielstrategie abhängig vom Untergrund, auf dem sich der Roboter bewegt, und einigen weiteren Aspekten, variieren.
Ziel dieses Projektes ist die Implementierung eines Roboters, der trotz genannten Umständen, in möglichst kurzer Zeit lernt, sich vorwärts zu bewegen, sodass er für kurze Vorführungen tauglich ist.

Die Implementierung ist stark an die Beschreibung des Lernalgorithmus in [1] gelehnt, unterscheidet sich jedoch um wesentliche Erweiterungen, welche für die Ausweitung auf zwei Arme zwingend notwendig sind. In den folgenden Abschnitten wird zunächst auf den Aufbau des Roboters eingegangen. Der verwendete Algorithmus und dessen Implementierung in einer Python Simulationsumgebung wird im Anschluss beschrieben. Zu guter Letzt erklären wir kurz die Integration in ROS (Robot Operating System), sodass der Algorithmus schlussendlich auf dem Roboter ausführbar ist.

2 Stand der Technik und verwandte Arbeiten

Die wohl ähnlichste Arbeit ist der ein-armige Roboter “The Crawler”, aus dem Jahr 2009 nach [1] oder der Krabbelroboter von Jim Demello [2], welcher zusätzlich ein neuronales Netz verwendet, welcher in seiner Implementierung einen Zusatzaufwand - und keinen Gewinn - mit sich bringt. Es gibt jedoch eine größere Vielfalt an ähnlichen Projekten, welche anstatt zwei Armen, zwei Beine verwenden. Bei “Bipod-Walkern” gibt es ebenfalls verschiedene Modelle, so sind die Beine meist entlang der Querachse angeordnet, oder aber auch entlang der Längsachse wie bei [3].
Bei Google Brains zwei-beinigem Roboter wird ein tiefes neuronales Netz zum Anwenden der Maximum-Entropie-Methode, einer Methode basierend auf Bayesscher Statistik, verwendet. Diese Kombination ermöglicht die komplexe Verarbeitung einer hohen Informationsdichte, trotz mangelnder vorab verfügbarer Information. Laut Angabe von [3] ist der Roboter dazu fähig einen stabilen Gang innerhalb von zwei Stunden zu lernen.
Das Laufen mit zwei Beinen benötigt eine größere Bandbreite an Informationen um verstanden und gelernt zu werden, da der Roboter umkippen kann. Damit geht einher, dass die Entwicklung dadurch erschwert wird, dass die Peripherie während dem Lernprozess Kollisionsschäden erleiden kann. Die Umsetzung einer akkuraten physikalischen Umgebung erweist sich ebenfalls als schwieriger, da eine drei-dimensionale Simulation, welche das Kippverhalten beinhaltet, nötig wird.
Dadurch, dass unser Krabbelroboter nicht umkippt, müssen wir nur zwei Dimensionen simulieren. Die Verwendung eines neuronalen Netzes ist durch die geringe Komplexität und Informationsdichte nicht nötig, stattdessen wenden wir heuristische Methoden an, um die Lerndauer mit zwei Armen möglichst nah an jene eines ein-armigen Roboters anzunähern.

3 Der Roboter

3.1 Hardware

Als zentrale Steuereinheit für den Roboter (Abb. 3) wird ein Raspberry Pi 3 Model B+ verwendet. Der Raspberry Pi ist auf dem Rumpf direkt über der Achse der Räder montiert. Dem Roboter stehen zwei Arme mit jeweils zwei Gelenken zur Verfügung. Jedes Gelenk wird durch einen Dynamixel AX-12A Servo bewegt.

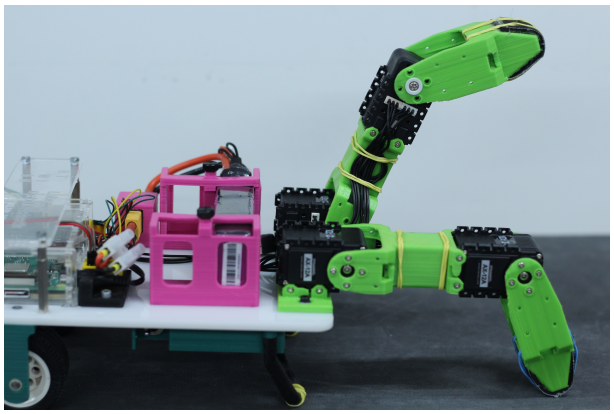

Abbildung 3: Zwei-armiger Robot Crawler

Die vier Servos werden über das USB-Modul USB2AX v3.2b angesteuert. Als Energiequelle für die Elektronik werden zwei LiPo-Akkus mit jeweils 11.1 V eingesetzt. Die Betriebsspannung von 5 Volt für den Raspberry Pi wird von einem Spannungsregler erzeugt. Dieser, sowie ein Ein-/Ausschalter, befinden sich unter dem Rumpf des Roboters. Die Bereifung setzt sich aus zwei LEGO Rädern der Größe 49,6 mm x 28 mm zusammen. Am rechten Rad ist ein Inkrementalgeber verbaut, der die Bewegung des Roboters registriert. Dieser wird mit 5 Volt versorgt. Die Ausgangssignale dieses Sensors werden über einen Pegelwandler, der die 5 Volt auf 3,3 Volt regelt, an den GPIO Pins des Raspberry Pi ausgelesen. Die beiden Arme sowie sämtliche Halterungen und Stützen kommen aus einem 3D Drucker. Um auf dem von uns getesteten Untergrund(Teppichboden) eine möglichst schnelle Bewegung zu ermöglichen, haben wir an den Spitzen der beiden Arme Schleifpapier angebracht, damit die Haftung auf dem Untergrund besser ist.

3.2 Algorithmus

Unser Algorithmus basiert auf dem Value Iteration Verfahren, wie er in [4] und [5] beschrieben ist. Ein Grund hierfür ist die leichtere Nachvollziehbarkeit des Lernfortschrittes des Roboters, welche bei uns in einer grafischen Oberfläche über die Summe der Value Tabelle dargestellt wird und zu jedem Zeitschritt aktualisiert wird. In jedem Zeitschritt $t \in N_0$ befindet sich unser Agent in einem Zustand $s_t = (s_1, s_2, s_3, s_4)$, wobei s_i ein diskreter Winkelwert des i-ten Gelenks ist. Der Agent führt im Zustand s_t eine Aktion $a_t = (a_1, a_2, a_3, a_4) \in A$ aus, wobei a_i die Zustandsänderung des i-ten Drehgelenks angibt. Durch Ausführung einer Aktion a in einem Zustand s gelangt der Agent in einen Folgezustand s'. Die Zustandsübergangsfunktion $\delta(s,a) = s'$ ist dem Agenten bekannt. Verbunden mit einem Zustandsübergang ist eine Belohnung $r(s,a)$. Die Belohnung $r(s,a)$ beobachtet der Agent im Laufe des Verfahrens.

Hierbei ist es wichtig zu wissen, dass das Belohnungssystem vom Menschen oder einer höheren Ebene als dem Agenten vorgegeben ist. In unserem Fall entspricht die zurückgelegte Distanz der Belohnung. Diese kann auch negativ ausfallen.
Hauptziel des Agenten ist das Erlernen der Wertfunktion $V(s)$, die angibt, welche Belohnungen der Agent in Zukunft maximal zu erwarten hat, wenn er in Zustand s die beste bekannte Strategie π (greedy policy) verfolgt (siehe Beispielstrategie in Abb. 4). Damit die Werte der Wertfunktion endlich werden, werden weiter in der Zukunft liegende Belohnungen mit einem Diskontierungsfaktor $\gamma = 0.9$ versehen.

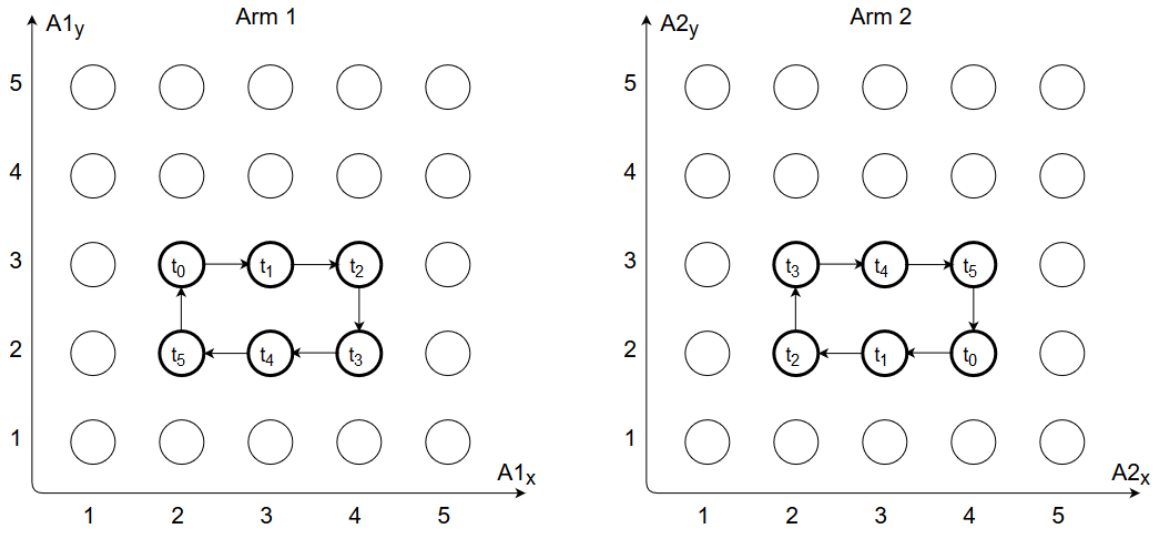

Abbildung 4: Zeitversetzte Strategie bei einem 5x5 Zustandsraum beider Arme

Der im Folgenden dargestellte Algorithmus 1 initialisiert die Wertfunktion $V(s)$ und die Belohnungsfunktion $r(s,a)$ mit 0 und startet im Zustand $state = (1,1,1,1)$. In Abhängigkeit von ε wählt der Roboter entweder eine Zufallsaktion a (Zeile 7, Explorationsprinzip) oder eine Aktion a, die die zukünftige Belohnung maximiert (Zeile 9 Exploitation Prinzip). Der Agent führt die Aktion a durch. Mit der beobachteten Belohnung wird $r(s,a)$ für den aktuellen Zustand und der entsprechenden Aktion aktualisiert (Zeile 12 bis 15). In Zeile 17 wird die Value-Funktion $V(s)$ für jeden Zustand s verbessert.
Um die Trainingszeit kurz zu halten werden folgende Einstellungen vorgenommen und zwei Tricks angewandt. Zum Einen wird der Aktionsraum eingeschränkt. Es darf nur ein Gelenk eines Arms in einem Schritt bewegt werden. So sind pro Arm maximal 4 Aktionen (Gelenk 1 hoch, Gelenk 1 runter, Gelenk 2 hoch und Gelenk 2 runter) möglich. Damit sind in einem Schritt maximal 16 (4 (Arm 1) * 4 (Arm 2)) Aktionen möglich. Lässt man zu, dass alle

Gelenke gleichzeitig bewegt werden dürfen, wobei ein Gelenk jeweils nach oben, nach unten oder nicht bewegt werden kann, so würden sich maximal 81 ((3 (Gelenk 1) * 3 (Gelenk 2)) (Arm 1) * (3 (Gelenk 1) * 3 (Gelenk 2)) (Arm 2)) Aktionen ergeben. Also deutlich mehr. Zusätzlich wird der Zustandraum auf 3x5 pro Arm festgelegt. So ergeben sich bei zwei Armen insgesamt 225 (3x5x3x5) Zustände.
Neben den beschriebenen Einstellungen konnte die Lernzeit durch die Symmetrie der Arme weiter verkürzt werden. Hierbei nehmen wir zwei Zusammenhänge als gegeben an. Einerseits kann die Belohnung einer Aktion auch bei der gespiegelten Aktion a_m und dem gespiegelten Zustand s_m aktualisiert werden (Zeile 13). Desweiteren ist zu erwarten, dass bei einer umgekehrten/rückwärts ausgeführten Aktion a_r die umgekehrte Belohnung abgeworfen wird. Zu der umgekehrten Bewegung existiert wiederum eine dazu gespiegelte Bewegung. Dadurch ergeben sich zwei weitere Werte, die in der Belohnungstabelle aktualisiert werden können (Zeile 14 und Zeile 15).

Algorithm 1 ValueIteration des zwei-armigen Roboters

1: Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in S$
2: Initialize r arbitrarily, e.g., $r(s,a) = 0$, for all $(s,a) \in S \times A$
3: $state \leftarrow (g_t = 1, g_x = 1, g_y = 1, g_z = 1)$
4: **loop**
5: $\quad \xi \leftarrow$ rand(0..1)
6: $\quad$ **if** $\xi < \varepsilon$ **then**
7: $\quad\quad a \leftarrow \underset{a \in A(state)}{rand}(a)$
8: $\quad$ **else**
9: $\quad\quad a \leftarrow \underset{a \in A(state)}{argmax}(r(state,a)) + \gamma V(\delta(state,a))$
10: $\quad succesorState \leftarrow \delta(state,a)$
11: $\quad reward \leftarrow$ observe $reward(state,a)$
12: $\quad r(state,a) \leftarrow reward$
13: $\quad r(state_m,a_m) \leftarrow reward$
14: $\quad r(successorState,a_r) \leftarrow -reward$
15: $\quad r(successorState_m,a_{rm}) \leftarrow -reward$
16: $\quad$ **for all** $s \in S$ **do**
17: $\quad\quad V(s) \leftarrow \max_{a \in A(s)} [r(s,a) + \gamma V(\delta(s,a))]$
18: $\quad state \leftarrow successorState$

Bei der Implementierung des Algorithmus wurden keine spezifischen Frameworks für die Anwendung von Methoden des maschinellen Lernens verwendet. Alle angewandten Algorithmen wurden unter Verwendung der Python Bibliothek “numpy” implementiert.

3.3 Roboter Simulation

Der Roboter in seiner physischen Struktur ist mit Hilfe des Python Frameworks Box2D [6] modelliert und befindet sich in einer physikalischen Simulation, in der die Erdanziehungskraft, Gewichte der Roboterkomponenten und sowohl die Reibung, als auch die Widerstände des Untergrundes und der Roboterarme berücksichtigt werden.
Der Roboter in der Simulation hat keine Wahrnehmung, jedoch messen wir die zurückgelegte Strecke mit Hilfe gegebener Funktionen. Die Strecke, die der Roboter innerhalb einer Iteration zurücklegt entspricht der Belohnung, die er erhält. Durch die virtuelle Simulation konnten bereits vor der Konstruktion des realen Modells sinnvolle Armlängenverhältnisse, Anzahl an Winkelschritten und Platzierung der Stütze bestimmt werden, die dann in den Bau des physischen Roboters eingeflossen sind. Durch die Simulation ist außerdem eine beschleunigte Abschätzung über die Zeitdauer des Lernvorgangs möglich, ohne den echten Roboter betreiben zu müssen.

Das Robotermodell ist flexibel konfigurierbar. Die Konfiguration und das Verfolgen des Lernvorganges ist über eine grafische Oberfläche möglich, welche mit Hilfe des PyQt5 [7] Frameworks implementiert wurde (siehe Abb. 5).

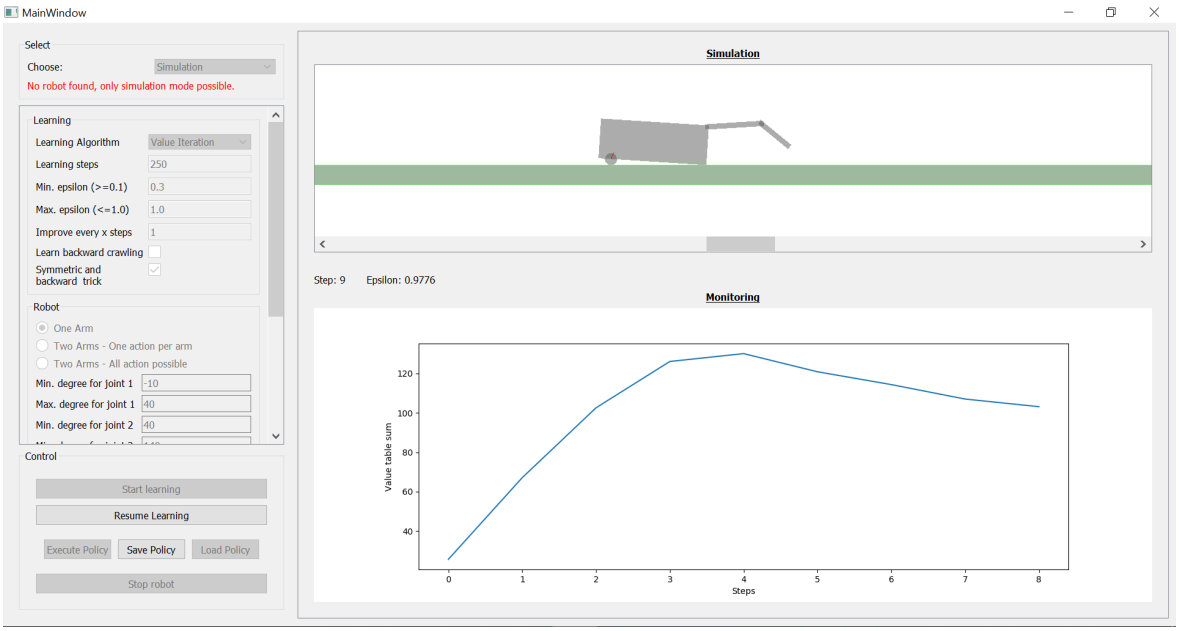

Abbildung 5: GUI zur Konfiguration der Simulation und der Motorgeschwindigkeit des Roboters.

Unter dem Segment “Monitoring” wird die Summe des Wertebereichs der Wertfunktion in Abhängigkeit der vergangenen Zeitschritte visualisiert. Die Maße des Roboters in der Simulation sind den Längen- und Höhenmaßen des realen Roboters angepasst. Über die GUI ist einstellbar, ob das Lernen in der Simulation dargestellt wird oder ob dabei der reale Roboter angesteuert wird. Diese Option ist nur verfügbar wenn sich der Roboter im selben Netzwerk befindet. Für schnelle Vorführungszwecke und längere Lernzeiten ist es auch möglich eine bereits gelernte Strategie zu laden, dementsprechend können diese auch - zu jedem Zeitschritt - gespeichert werden.

3.4 Roboter Software - Robot Operating System (ROS)

Zur Ansteuerung und Kommunikation mit dem Roboter wird das Robot Operating System (ROS) [8] verwenden. ROS ist ein modulares Software-Framework, welches dabei hilft Roboteranwendungen zu implementieren. Das Framework bietet viele Bibliotheken sowie hilfreiche Tools für die Entwicklung. Durch die Aufteilung in Module kann eine Kommunikation über das Netzwerk stattfinden. Somit ist die Möglichkeit gegeben, die Anwendung auf verschiedenen PCs oder Robotern auszuführen. Als Betriebssystem wird dabei die Linux Distribution Ubuntu verwendet, die auf dem Raspberry Pi installiert ist. Die Kommunikation zwischen den Sensoren wird mit Prinzipien wie Pulisher/Subscriber oder Service/Client um-

gesetzt. Aus den Signalen des Inkrementalgebers wird eine Distanz berechnet. Diese wird auf ein ROS Topic publiziert. Durch das Abonnieren des selben Topics erhält der Lernalgorithmus die Belohnung für eine ausgeführte Aktion. Mit Erhaltenen der Belohnung berechnet der Lernalgorithmus den nächsten Schritt. Das Service/Client Konzept von ROS wird dazu verwendet, um die Winkel an die Servos zu übermitteln. Das Lernverfahren kann über die GUI gestartet und gestoppt werden.

4 Ergebnisse

Aus den bisherigen Simulationen ergab sich, dass der ein-armige Roboter circa 100 Iterationen benötig um stets einer sichere Laufstrategie zu entwickeln, wohingegen der zwei-armige Roboter kaum mehr als die doppelte Anzahl benötigt, nämlich circa 250 Iterationen. Trotz des vierfach größeren Aktionsraumes und fünfzehnfach größeren Zustandsraumes im Vergleich zu einem ein-armigen Roboter ist es also möglich, dass der Roboter in einer nichtproportionalen Lernzeit eine sinnvolle Strategie erlernt.
Um ein Gefühl dafür zu geben was das zeitlich bedeutet, wurde gemessen wie viele Aktionen der reale Roboter innerhalb einer Minute durchführen kann. Bei einer sinnvollen Motorgeschwindigkeit - darunter verstehen wir ein Tempo, bei dem die Roboterarme noch Haftung am Grund haben - können 200 Iterationen innerhalb einer Minute ausgeführt werden. Lernt er also mit zwei Armen, so benötigt er keine zwei Minuten bis ein ausgereiftes Laufverhalten vorführbar ist.

5 Fazit und Ausblick in die Zukunft

In unserem Projekt wurde ein Krabbel-Roboter entwickelt, welcher selbstständig lernt, vorwärts zu krabbeln und in Seminaren, Vorlesungen oder auf Messen vorgeführt werden kann. Der Krabbelroboter kann entweder ein-armig oder zwei-armig betrieben werden. Im ein-armigen Fall werden beide Arme synchron bewegt. Der Roboter lernt in weniger als einer Minute ein sinnvolles Vorwärtskrabbeln. Im zwei-armigen Fall benötigt der Roboter etwa drei Minuten Zeit, bis er sich sinnvoll vorwärts bewegen kann.
Vor Inkorporation der zuvor genannten heuristischen Techniken (siehe Textabschnitt aus 3.2) benötigte der Roboter im zwei-armigen Fall circa zehn Minuten bis zu einer gelernten Strategie, welche ihn vorwärts brachte.
Ursprünglich war angedacht Prioritized Sweeping einzusetzen, eine Technik aus [4] und [9], dieser Plan wurde jedoch verworfen. Beim Prioritized Sweeping werden gezielt Vorgänger-

zustände von Zuständen betrachtet, die besonders hohe Bewertungen erzielt haben. Dadurch konvergiert die Value-Funktion schneller. Diese Technik macht erst dann Sinn, wenn der Agent bereits zu Genüge gelernt hat und sich in der Werttabelle ein positiver Pfad abbildet, der verfolgt werden kann. Dies hat in Kombination Verfolgung einer Greedy Policy nur eine geringe Wirkung, da beide Techniken denselben Effekt erzielen.

6 Danksagung

Wir bedanken uns herzlich bei Prof.Dr. Oliver Bittel und Jürgen Keppler für ihre tatkräftige Unterstützung im Projekt, in Form von regelmäßiger Betreuung, intellektuellen Beiträgen am Projekt, das Bauen des Roboters seitens Jürgen Keppler sowie das Drucken der Komponenten, die Bereitstellung der finanziellen Mittel und der Arbeitsumgebung, und vor allem für das Ermöglichen dieses Projektes.

Literatur

[1] Michel, Tokic, Wolfgang Ertel und Joachim Fessler: *The crawler, a class room demonstrator for reinforcement learning*. In: *Twenty-Second International FLAIRS Conference*, 2009.

[2] Demello, Jim: *Machine Learning Crawler Robot Using Reinforcement Learning, Neural Net and Q-Learning*. `https://www.instructables.com/id/Q-Learning-machine-Learning-Crawler/`, besucht: 20.08.2019.

[3] Haarnoja, Tuomas *et al.*: *Learning to Walk via Deep Reinforcement Learning*. Technischer Bericht, Google Brain, Berkeley Artificial Intelligence Research, University of California, Berkeley, 2019.

[4] Sutton, Richard S, Andrew G Barto *et al.*: *Introduction to Reinforcement Learning*, Band 135. MIT press Cambridge, 1998.

[5] Ertel, Wolfgang und Nathanael T Black: *Grundkurs Künstliche Intelligenz*. Springer, 2018.

[6] *Box2D*. `https://box2d.org/`, besucht: 20.08.2019.

[7] *PyQt5*. `www.riverbankcomputing.com/software/pyqt/intro/`, besucht: 20.08.2019.

[8] *ROS*. `http://wiki.ros.org/de/ROS/Introduction`, besucht: 20.08.2019.

[9] Lee, Mark: *9.4 prioritized sweeping*. `http://incompleteideas.net/book/first/ebook/node98.html`, besucht: 11.08.2019.