DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Erlernen von Hindernisumfahrung mithilfe von Reinforcement Learning

Studienarbeit (T3_3101)

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Yannik Schiebelhut

Abgabedatum:	22. Mai 2023
Bearbeitungszeitraum:	14.10.2022 - 22.05.2023
Matrikelnummer, Kurs:	3354235, TINF20B1
Gutachter der Dualen Hochschule:	Florian Stöckl

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit (T3_3101) mit dem Thema:

Erlernen von Hindernisumfahrung mithilfe von Reinforcement Learning

gemäß § 5 der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 21. April 2023

Schiebelhut, Yannik

Abstract

- Deutsch -

Platzhalter

Abstract

- English -

Placeholder

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung

2 Grundlagen

2.0.1 Machine Learning

Besonderer Fokus auf Reinforcement Learning und Begriffsdefinitionen

2.0.2 Unity

kurze Einführung zur Unity-Engine allgemein ML-Agents, Historie und Funktionsweise

2.0.3 Vektorgeometrie

Abstandsbestimmung und Winkel zwischen Vektoren

3 State of the Art

4 Konzeptionierung

4.1 Beschreibung der Projekt-Basis

Die Zielsetzung dieser Arbeit soll aufbauend auf einer Vorgängerarbeit realisiert werden, welche vor einigen Jahren ebenfalls im Rahmen einer Studienarbeit durchgeführt wurde. Hier soll nun zunächst die Vorgehensweise der Vorgängerarbeit in ihren Grundzügen erläutert werden.

4.1.1 Aufbau und Simulation des Roboters

Kernelement der Arbeit ist ein vierbeiniger, 3D-gedruckter Roboter, der in seiner Anatomie einer Spinne gleicht. Der Roboter besteht aus einer rechteckigen Zentralplatte. An jeder Ecke dieser Platte ist ein Bein angebracht. Die Beine des Roboters bestehen jeweils aus 3 separaten Teilen. Folglich hat jedes Bein zwei Gelenke. [1, S. 52] Jedes dieser Gelenke wird mit einem Servomotor des Typs SG90 XY realisiert, welche sich in einem Aktionsradius von 180° bewegen können. Die Neutralstellung der Beine ist die jeweilige Mittelposition des Servomotors. Angegeben wird der Aktionsradius des Servos als Wert im Bereich 0° - 180°, die Mittelstellung befindet sich also bei 90°. [1, S. 38] Die verwendeten Servomotoren bewegen sich nur auf einer Achse. An der Stelle, an der die Beine mit dem Körper verbunden sind, befindet sich ein weiterer Servomotor. Mithilfe der verbauten Servomotoren ist es möglich, jedes Bein anzuziehen oder auszustrecken und es nach vorne beziehungsweise hinten zu bewegen.

Angesteuert werden die Servomotoren mit einem ESP8266 Mikrocontroller, welcher mittels I^2C Steuersignale an ein Servo-Breakout-Board sendet, an welchem wiederum die Servomotoren angeschlossen sind. Die Stromversorgung erfolgt über eine Batterie, damit sich der Roboter autark von einer Stromquelle im Raum bewegen kann. [1, S. 54] Die aufgelisteten Bauteile werden so mit Kabelbindern an der Zentralplatte befestigt, dass sie nicht mit der Bewegungsfreiheit der Beine interferieren. Am Roboter ist keinerlei Sensorik verbaut. [1, S. 36]

Da das Training des Roboters in der Realität zu lange dauern würde und dabei außerdem die Gefahr drohen würde, den Roboter zu beschädigen, wurde eine Simulationsumgebung aufgebaut, in der das Training des Roboters erfolgt. Diese ist in Unity realisiert. In Unity ist es möglich, dasselbe 3D-Modell zu importieren, das auch für den Druck der Teile des Roboters verwendet wird, was eine akkurate Umsetzung der Dimensionen sicherstellt. Die größte Schwierigkeit einer Simulation des Roboters besteht darin, die Servomotoren abzubilden. Im Gegensatz zu einem Computerprogramm ist es Bauteilen in der realen Welt nicht möglich, instantan einen gezielten Zustand anzunehmen. Das heißt, wird für einen Servomotor ein neuer Winkel vorgegeben, so braucht dieser eine bestimmte Zeit, bis er diesen erreicht hat. Diese Zeit ist einerseits von der normalen Bewegungsgeschwindigkeit des Servomotors abhängig, andererseits stellt die Last, die der Motor bewegt, einen weiteren Einflussfaktor dar. Die Vorgängerarbeit implementiert eine Software-Simulation der Servomotoren, bei denen zumindest der Aspekt der normalen Bewegungsgeschwindigkeit berücksichtigt wird, sowie verwandte, allgemeine Charakteristika der Bewegung. [1, S. 37] Zusätzlich werden in Unity die ungefähren Gewichte der einzelnen Teile eingetragen, um eine akkurate Simulation der physikalischen Kräfte zu ermöglichen, die auf den Roboter einwirken. Die Simulation dieser allgemeinen Physik wird bereits als Grundfunktion der Unity-Engine bereitgestellt und muss nicht separat implementiert werden.

irgendwo mehr zu Unity schreiben

4.1.2 Training des Roboters

Mit der fertig aufgebauten Simulation kann nun der Roboter für alle erdenklichen Szenarien trainiert werden. Dazu müssen Rahmenbedingungen des Trainings definiert werden. Primär sind dies die Beobachtungen, die der Trainingsalgorithmus macht, die Aktionen, die er ausführen kann und die Belohnung, die er als Feedback für seine Handlungen erhält. Die Zielsetzung der Vorgängerarbeit bestand aus einer reinen Vorwärtsbewegung. Da der reale Roboter über keine Sensorik verfügt und lediglich die Ansteuerwinkel seiner Servomotoren kennt, waren dies auch die einzigen Beobachtungen, die dem Algorithmus zugänglich gemacht wurden. Als Aktionen kann der Algorithmus beliebige Zielwinkel für die Servomotoren setzen. Die Belohnung für den Roboter bestand in der Vorgängerarbeit aus der Streckendifferenz, die er nach vorne zurückgelegt hat. Bestraft wurde der Roboter für ein Umkippen, um zu verhindern, dass bei der späteren Übertragung auf den realen Roboter die Steuerelektronik beschädigt wird. Um das Training zu beschleunigen, sind in

der Vorgängerarbeit mehrere Agenten nebeneinander in derselben Umgebung instanziiert. Nach einer Einstellung der Hyperparameter wurden Modelle mit den Reinforcement-Learning-Algorithmen SAC und PPO trainiert. Dabei wurde festgestellt, dass in diesem Anwendungsfall die Ergebnisse des PPO-Algorithmus denen des SAC-Algorithmus deutlich überlegen sind.

Die Bewegungsart, die der Roboter sich bei diesen Trainingsbedingungen antrainiert, ist keine klassische Form des natürlichen Laufens. Stattdessen führen die Ergebnisse des Trainings dazu, dass der Algorithmus den Roboter mit einem der Hinterbeine einknicken lässt und ihn dann sprungartig nach vorne katapultiert.

4.1.3 Übertragung in die Realität

Die Implementierung der Vorgängerarbeit sieht auch eine Übertragung der Ergebnisse auf den realen Roboter vor. Zu diesem Zwecke können die Steuersignale, die der Roboter in der Simulation erhält, über eine serielle Verbindung übertragen werden und direkt auf dem realen Roboter angewandt werden.

Die unternommenen Versuche waren jedoch leider nicht erfolgreich. Als Ursache fällt die Vermutung auf Hardwarefehler, da der Roboter – in der Luft gehalten – die Bewegungen korrekt spiegelt. Wird der Roboter jedoch auf den Boden gestellt, knickt er unter seinem Gewicht sofort ein und kann die gelernte Laufmethodik nicht anwenden.

4.2 Einschränkungen (und Übertragungsprobleme)

- bislang sollte der Roboter nur geradeaus laufen
- Roboter verfügt über keinerlei Sensorik
- kennt nur den aktuellen Winkel der Beine/Servomotoren
- Training erfolgte rein in der Simulation
- die Simulationsumgebung kennt den Zustand (z.B. Neigung) des Roboters und kann anhand dessen den Wert der Belohnungsfunktion berechnen und an den Roboter zurückmelden

- Roboter kennt seinen eigenen Zustand nicht
- bisheriges Modell wird lediglich in der Praxis angewandt, unter der Annahme, dass bei korrekt gelerntem Modell keinerlei Zusatzinformationen notwendig sind, um den Roboter seine Aufgabe erfüllen zu lassen: geradeaus zu laufen

Probleme in der erweiterten Aufgabenstellung:

- der Roboter soll nun einem vorgegebenen Pfad folgen
- dabei soll der Pfad für jeden Durchlauf dem Roboter individuell vorgegeben werden können
- der Roboter soll NICHT einen vorgegebenen Pfad lernen und danach immer von diesem Pfad ausgehen
- dem Roboter muss also ein Pfad mitgegeben werden können
- Positionierungsinformationen benötigt?
- einerseits nein, theoretisch kann der Roboter seine aktuelle Position anhand seiner vergangenen Bewegungen vom Ausgangspunkt bestimmen
- andererseits ja, da der Roboter auf dem Boden rutschen kann (zumindest in der Realität), das Berechnen von Entfernungen den gesamten Algorithmus stark verkompliziert und unter Umständen durch kleinere Abweichungen sehr ungenau ausfallen kann
- mögliche Lösung in der Simulation: Positionierungsinformationen innerhalb der Simulationsumgebung für Roboter freigeben
- diese Informationen könnten dem Roboter später durch andere Sensoren geliefert werden
- wenn das Informationsformat des neuen Sensors umgewandelt wird in das bisherige Format (zum Beispiel durch ein externes Modul), könnte ein anderer Sensor Plug-And-Play in das trainierte Modell integriert werden
- außerdem soll der Roboter Hindernisse auf seinem Weg erkennen und gezielt umgehen können
- danach soll auf den Pfad zurückgekehrt werden

- dafür wird eine Hindernis-/Kollisionserkennung benötigt
- aktuell hat der Roboter keinerlei solche Sensorik
- vereinfacht soll für die Hinderniserkennung in der Simulation die Kollisionserkennung für die Beine verwendet werden
- dadurch muss der Roboter mit seinem Hindernis zusammenstoßen, um es wahrzunehmen
- in der Realität wäre natürlich eine Hinderniserkennung sinnvoll, mit der Hindernisse bereits vor einer Kollision erkannt werden können (z.B. LiDAR, Kameras oder ähnliche), die Integration solcher Systeme würde jedoch den Umfang dieser Arbeit erheblich übersteigen
- hier soll eher ein Proof-of-Concept für die selbstständige Umsteuerung von Hindernissen erarbeitet werden
- Genauigkeitsprobleme beim Übertragen der Springbewegung: daher genauere Einschränkungen für den Algorithmus
- bisher bewegt sich der Roboter nach Training in einer springenden Bewegung fort
- diese Bewegungsform ist sehr kraftaufwändig und instabil, der Roboter schwankt dabei stark um die horizontale Achse und kann seine exakte Landeposition nur bedingt steuern
- vor allem relevant, falls dem Roboter keine Positionierungsinformationen zur Verfügung gestellt würden, da dann jeder Millimeter zählt
- aber auch zur Erhöhung der allgemeinen Genauigkeit und Reduzierung der Fehler, wäre es sinnvoll, die Fortbewegung des Roboters zu stabilisieren
- mögliche Maßnahme zur Einschränkung der Bewegung: restriktive Anpassung der Belohnungsfunktion, wenn sich die Höhe des Mittelteils des Roboters zu stark verändert oder dieser spürbar die Neigung zum Horizont verändert, wird der Agent bestraft
- Insgesamt soll keine Übertragung des implementierten Modells auf den realen Roboter stattfinden

- wie oben beschrieben wäre eine Implementierung von diversen Sensoren vonnöten, was den Umfang dieser Arbeit deutlich übersteigt

4.3 Wahl der Simulationsumgebung

- in der Vorgängerarbeit wurde bereits über verschiedene mögliche Simulationsumgebungen geschrieben
- die Bedingungen haben sich leicht geändert
- eine mögliche Software (MuJoCo) ist mittlerweile nicht mehr kostenpflichtig, was damals einer der Gründe war, die gegen diese Software gesprochen haben
- andererseits soll diese Arbeit an die vorangegangene anknüpfen
- wenn die Simulationsumgebung gewechselt würde, würde man im Grunde genommen kaum Ergebnisse der Vorgängerarbeit aufgreifen sondern in vielen Gesichtpunkten von 0 beginnen
- deshalb soll weiterhin Unity verwendet werden
- Unity bietet jedoch die Möglichkeit, die standardmäßige Physics-Engine gegen andere nach dem Plugin-Prinzip auszutauschen
- insofern könnte realistisch und mit geringem Aufwand in Betracht gezogen werden, MuJoCo als Physics-Engine in Unity einzubinden, um somit das Ergebnis des Trainings durch eine andere/verbesserte Physiksimulation zu verbessern
- auch wäre es möglich, die Ergebnisse der verschiedenen Umgebungen zu vergleichen
- da jedoch keine Übertragung auf einen realen Roboter erfolgt, dürfte in diesem Kontext kein spürbarer Unterschied zu beobachten sein / ein beobachteter Unterschied könnte nicht hinsichtlich seiner Aussagekraft eingeordnet werden

4.4 Geplante Realisierung

1. alte Umgebug und Lernergebnisse des Roboters rekonstruieren; bringt Probleme mit sich, da einige der verwendeten Komponenten einer starken Entwicklung unterliegen/unterlagen, weshalb potenziell Anpassungen vorzunehmen sind, um die alte Umgebung weiterhin verwenden zu können oder die Umgebung auf einen aktuellen Stand der Technik migriert werden sollte, um von Verbesserungen im verwendeten Tooling zu profitieren und eine zukunftssichere Basis zu bieten
2. Format entwerfen, wie dem Roboter ein Pfad mitgeteilt werden kann, dem dieser folgen soll; ein Pfad wird hierbei voraussichtlich aus mehreren Punkten bestehen, die sich entlang seines Verlaufs befinden
3. Belohnungsfunktion anpassen; oben erwähnt Anpassungen hinsichtlich Laufstabilität; außerdem muss ein Abweichen vom direktesten möglichen Weg bestraft werden / zu einer ausbleibenden oder sehr geringen Belohnung führen
4. Hindernisse ergänzen; hierfür sollte die Kollisionserkennung der Unity-Engine verwendet werden können; mithilfe des Hinzufügens von Kollisionsmodellen für in der Simulationsumgebung platzierten Objekte, sollte der Roboter automatisch nicht mehr durch Hindernisse hindurch gehen können; größte Herausforderung sollte hier die Adaption der Belohnungsfunktion werden, damit diese den Roboter nicht daran hindert, den Pfad zu verlassen, das Hindernis zu umgehen und anschließend auf den Pfad zurückzukehren und eine deutlich gesteigerte Belohnung zu erhalten; gleichzeitig darf der Roboter sich nicht zu frei im Raum bewegen, sondern sollte so dicht wie möglich am vorgegebenen Pfad bleiben

5 Implementierung

6 Bewertung der Ergebnisse

7 Fazit / Future Work

ein Titel oder ggf zwei Kapitel?

Literaturverzeichnis

[1] Waidner, D. *Erlernen von Bewegungsabläufen durch Reinforcement Learning.* Techn. Ber. DHBW Karlsruhe, 2020.