Linienfolger auf Basis eines Raspberry Pi's und Q-Learning

Czogalla, Dennis-Immanuel
Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Str. 10, 68163 Mannheim

***Zusammenfassung*—Im Rahmen der Lehrveranstaltung Künstliche Intelligenz für autonome Systeme an der Hochschule Mannheim wurde ein Roboter auf Basis eines Raspberry Pi's entwickelt und programmiert. Ziel war es, dass dieser Roboter mit Hilfe von Reinforcement Learning (Q-Learning) lernt einer Linie zu folgen. Zunächst wurde hierfür eine Simulation geschrieben, die die Problemstellung aus der Realen Welt stark vereinfacht repräsentiert. Anhand dieser Simulation konnten mögliche Probleme im Voraus identifiziert und der Algorithmus soweit optimiert werden, dass er relativ zügig zu einem guten Ergebnis kommt. Anschließend konnte der Code mit minimalen Änderungen auf den Roboter übertragen und erfolgreich ausgeführt werden. Dabei wurden auch Lösungen entwickelt, die es nicht in die finale Version geschafft haben, die aber in dieser Arbeit kurz aufgeführt werden.**

Inhaltsverzeichnis

1. Einleitung

Der Roboter besitzt ein simples Kettenlaufwerk, bestehend aus jeweils einem Laufrad und einem Antriebsrad pro Seite. Er verfügt über einen Lichtsensor, der an drei verschiedenen Positionen Werte von seiner Umgebung auslesen kann. Die Werte stellen verschiedene Helligkeitsstufen dar. Das gesamte Fahrgestell, sowie die vier Räder wurden 3D gedruckt. Angetrieben wird er von einem Arduino Uno, zwei Servo Motoren und sechs AAA Batterien.

Ziel ist es, dass der Roboter selbständig lernt mit Hilfe des Lichtsensors und dem Einsatz von Q-Learning (einer Variante des Reinforcement Learning) einer schwarzen Linie zu folgen. Hierfür soll der Arduino durch einen Raspberry Pi ersetzt werden. Die Stromversorgung muss dementsprechend angepasst werden und besteht aus einer Powerbank für den Raspberry Pi, sowie vier AAA Batterien für die Servo-Motoren. Der Lichtsensor ist für den Arduino vorgesehen und arbeitet mit 5V. Der Raspberry Pi arbeitet jedoch mit 3.3V und so muss der Pegel mit Hilfe von Widerständen angepasst werden. Um das Programm auf dem Roboter bequem starten und stoppen zu können, wird ein Taster hinzugefügt, der bei Betätigung einen Befehl an den Raspberry sendet und das Programm startet bzw. stoppt.

2. Exkurs Q-Learning

Reinforcement Learning beschreibt ein Vorgehen, bei dem ein Agent (z.B. der Roboter) mit seiner Umgebung interagiert und für jede seiner durchgeführten Aktionen einen entsprechenden Reward bekommt. Diese Rewards werden im Vorfeld festgelegt und können positiv oder negativ sein. Die Umgebung wird als eine Menge von Zuständen dargestellt, über die der Agent iteriert. Für jeden Zustand stehen ihm alle Aktionen zur Auswahl. Gesucht werden dabei die Aktionen, die den Reward maximieren. Dieser Vorgang stellt das Lernen des Agenten dar.

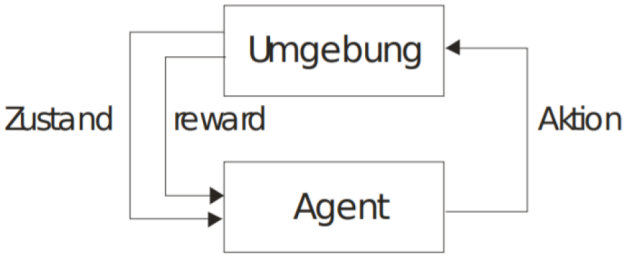

Abbildung 1. Reinforcement Learning

Das Q-Learning ist eine Variante des Reinforcement Learnings.

Hierbei kommt eine Bewertungsfunktion zum Einsatz, die eine Aktion in Abhängigkeit mit dem Zustand, in den der Agent durch die Aktion gelangt ist, bewertet und als sogenannten Q-Values abspeichert. Die Bewertungsfunktion sieht folgendermaßen aus [3] :

$$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$\max_{a'} Q(s', a')$ = Zustand mit höchsten Q-Value (wenn alle gleich, dann zufällig)
a = Action
s = State
α = Lernrate (wie schnell Änderungen übernommen werden, Wert zwischen 0 und 1)
r = Reward
γ = Discountfactor (Zustände, die weiter in der Zukunft liegen, werden geringer gewichtet)

Abbildung 2. Bewertungsfunktion Q-Learning

3. Das Projekt

3.1. Verwendete Hardware

- Raspberry Pi 3+
- Hamma Joy Powerbank 5200 mAh (Stromversorgung RPI)
- 2x Servos AR3603HB
- 4x AAA Batterien (Stromversorgung Servos)
- Lichtsensor (Sensorarray)
- Fahrgestell 3D gedruckt
- 3x Widerstände für Lichtsensor
- Taster
- Kabelbinder

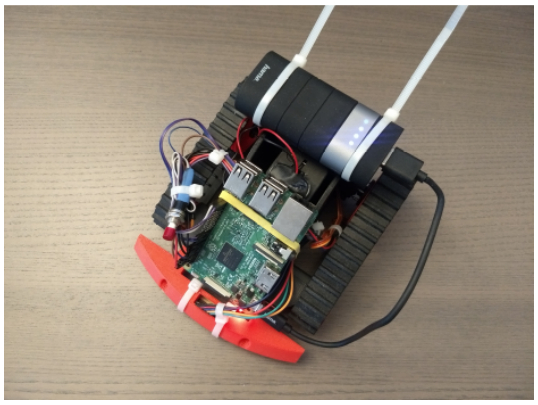

Abbildung 3. Roboter

3.2. Verwendete Software

- Raspbian Stretch Lite [4]
- Pycharm Community Edition [5]
- Pigpio zum Steuern der Servos [6]
- Tkinter Library für Simulation [7]
- Putty [9]
- Python 3.X

3.3. Simulation

Die Simulation ist in Python 3.X geschrieben und stellt eine stark vereinfachte Version der Problemstellung dar [2]. Das Spielfeld besteht aus 20x20 Kacheln, die entweder schwarz oder weiß sein können. Es können unterschiedliche Muster dargestellt werden. Hierfür wird mit Hilfe eines Grafikprogramms wie z.B. Photoshop ein Bild generiert (20x20 Pixel), das z.B. einen schwarzen Kreis enthält. Das Programm liest das Bild ein, wandelt die Informationen in schwarz-weiß um und färbt dementsprechend die richtigen Stellen auf den Spielfeld ein (vgl. Abbildung 5 auf Seite 2, vgl. Abbildung 6 auf Seite 3).

Der Roboter bzw. Akteur wird in Form eines roten Rechtecks dargestellt, das drei Felder hintereinander abdeckt. Dies entspricht technisch auch dem Lichtsensor, der nochmals in drei Sensoren aufgeteilt ist, die jeweils einen Helligkeitswert lesen können. Der Akteur hat drei verschiedene Bewegungsmöglichkeiten: Vorwärts, Rechtsdrehung, Linksdrehung. Das Spielfeld ist unbegrenzt (ähnlich wie bei dem Spiel Snake), d.h. der Akteur befindet sich nach dem Verlassen von z.B. der linken Spielfeldgrenze, an der rechten Spielfeldgrenze. Gleiches gilt für oben und unten.

Das Reinforcement Learning bzw. Q-Learning in der Simulation sieht folgendermaßen aus: Der Sensor ist in drei Sektoren unterteilt, die jeweils weiß oder schwarz sein können und verfügt damit über $2^3 = 8$ mögliche Kombinationen. Multipliziert man die Anzahl aller möglichen Kombinationen mit der Anzahl aller möglichen Bewegungsmöglichkeiten pro Sektor (Vorwärts, Links, Rechts), so erhält man die $2^3*3 = 27$ Zustände für das Q-Learning.

Die Bewertung dieser Zustände sieht folgendermaßen aus:

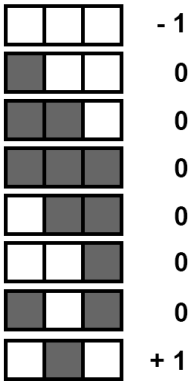

Abbildung 4. Rewards in Simulation

D.h. die Kombination Weiß Weiß Weiß erhält für alle Bewegungsmöglichkeiten (Vorne, Links, Rechts) einen negativen Reward von -1. Und für die Kombination Weiß Schwarz Weiß sowie alle Bewegungsmöglichkeiten den positiven Reward +1. Alle anderen Zustände und ihre Aktionen erhalten einen Reward von 0.

3.4. Umsetzung des Q-Learnings auf dem Roboter

Zunächst musste herausgefunden werden, wie die Servo-Motoren mit Hilfe des Raspberry Pis angesteuert werden. Hierfür wird die Bibliothek Pigpio [6] verwendet, mit der es möglich ist die Geschwindigkeit der Motoren einzustellen. Es wurden die Aktionen festgelegt und anschließend ein erster Test durchgeführt. Die Logik bzw. der Code für das Q-Learning konnte direkt aus der Simulation entnommen werden und musste nicht angepasst werden.

Nach den ersten Test stellte sich jedoch heraus, dass die drei festgelegten Aktionen nicht gut funktionieren, da das Spielfeld begrenzt ist und der Roboter dazu neigte, es recht zügig zu verlassen.

Es wurden diverse Möglichkeiten durchgetestet wie beispielsweise ein Suchalgorithmus, um wieder zum

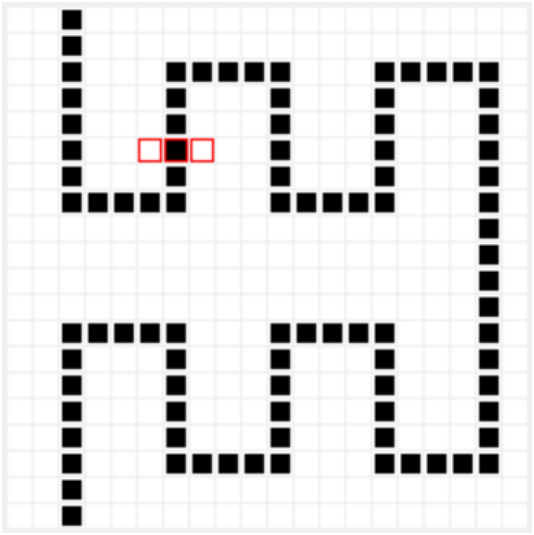

Abbildung 5. Linienform

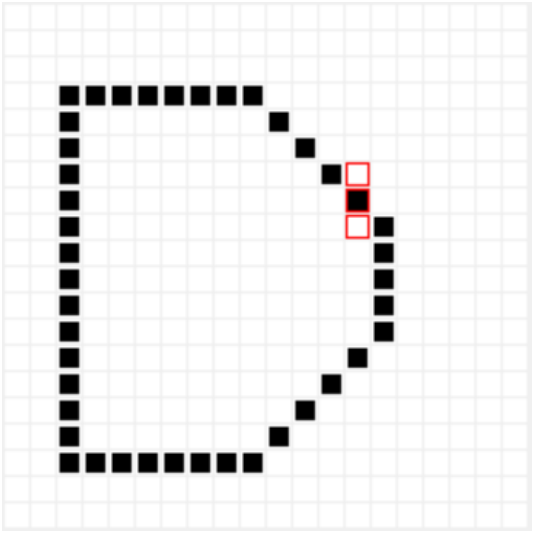

Abbildung 6. weitere Linienform

Spielfeld zurückzukehren, falls nach längerer Zeit kein positiver Reward mehr erfolgte. Näheres dazu im nächsten Kapitel.

Die beste Lösung hierfür war jedoch die Einführung einer zusätzlichen vierten Aktion (Rückwärts), die gleich zwei Verbesserungen mit sich brachte. Zum einen blieb der Roboter eher auf bzw. in der Nähe des Spielfelds und zum anderen war das erlernte Ergebnis deutlich präziser als mit nur drei Aktionsmöglichkeiten. Damit ergaben sich insgesamt 2^3*4 = 32 Zustände für das Q-Learning.

Es wurden zwei zusätzliche positive Rewards hinzugefügt und die Bewertung der restlichen Rewards wurde geändert (vgl. Abbildung 7 auf Seite 3). Die zusätzlichen geringer gewichteten Rewards sollen dem Roboter helfen, den Zustand mit dem besten Reward zu erreichen. Dieser ist nicht so leicht zu erreichen, da der Roboter exakt auf der Linie ausgerichtet sein muss. Oft befindet er sich jedoch zur Linie angewinkelt und verpasst so diesen Zustand [1].

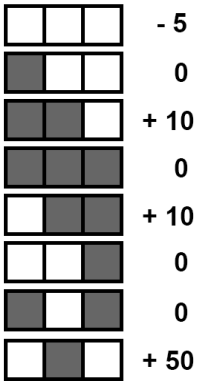

Abbildung 7. Rewards im realen Einsatz

3.5. Lösungen, die nicht funktionierten

Die drei Aktionen pro Zustand stellten sich als nicht zufriedenstellend heraus. Der Roboter verließ bereits nach kurzer Zeit das Spielfeld, da er zu diesem Zeitpunkt noch nicht genügend gelernt hatte und nach dem Zufallsprinzip vorging. Um dem entgegenzuwirken war ein Algorithmus notwendig, der den Roboter auf dem Spielfeld behielt und das ohne Einsatz von zusätzlichen Sensoren.

Zunächst wurde mit einer vierten Aktion experimentiert, bei der der Roboter sich um 180 Grad drehte. Schnell stellte sich heraus, dass diese Aktion dem Lernen eher schadete, da der Roboter beispielsweise sich an einer idealen Stelle befand, diese durch den Dreh jedoch verlor.

Die Idee mit der Drehung wurde jedoch nicht komplett fallen gelassen, sondern weiterentwickelt. Sie wurde nicht als vierte Aktion betrachtet, sondern als Verhalten, das nach einer bestimmten Zeit eintritt, wenn der Zustand mit dem besten Reward nicht gefunden wurde. Dies schien halbwegs zu funktionieren, nur passierte es, dass der Roboter manchmal dennoch vom Spielfeld abkam und nicht wieder zurückkehrte.

Als letzte Verbesserung wurde schließlich ein weiteres Verhalten hinzugefügt, das beim Suchen der Linie helfen sollte. Der Algorithmus sieht wie folgt aus:

Führe Q-Learning durch. Wurde nach einer bestimmten Zeit der Zustand mit dem höchsten Reward (weiß, schwarz, weiß) nicht gefunden, stoppe Q-Learning, dann drehe um 180 Grad und fahre eine bestimmte Zeit vorwärts. Sollte während dieser Zeit die Linie gefunden werden, brich ab und fahre mit dem Q-Learning fort.

Letztendlich funktionierte diese Lösung nicht zuverlässig. Während der Entwicklung dieser Lösung wurde auch mit anderen Aktionen experimentiert bzw. damit, um wie viel Grad sich der Roboter drehen sollte. Dabei wurde zufällig entdeckt, dass die Aktion Rückwärts-fahren zu einem sehr zufriedenstellenden Ergebnis führte. Zudem war sie auch sehr einfach und wurde damit in die finale Version des Roboters übernommen [1].

3.6. Roboter Bedienung

Gegenüber einem Arduino besitzt der Raspberry Pi den Nachteil, dass er über ein komplettes Betriebssystem verfügt. Das macht ihn für den Einsatz bei Robotern langsam und umständlich in der Bedienung. Der Arduino startet sein Script, sobald er Strom bekommt. Beim

Raspberry muss erstmal der Befehl zum Ausführen des Scripts ausgeführt werden. Um dieser Problematik entgegenzuwirken wurde zunächst auf das Betriebssystem Raspbian Stretch Lite [4] gesetzt. Dabei handelt es sich um eine abgespeckte Variante ohne GUI und nur mit den notwendigsten Werkzeugen. Der Bootvorgang ist selbst auf einem Raspberry Zero sehr schnell (ca. 10 Sekunden).

Um jetzt noch das Script auf dem Raspberry bequem ausführen lassen zu können, wurde ein Taster hinzugefügt. Für den Taster existiert ein Python Script, dass nach jedem Bootvorgang automatisch ausgeführt wird. Wenn der Taster gedrückt wird, startet das Script den Algorithmus für das Q-Learning. Wird er erneut betätigt, so stoppt das Script den Algorithmus.

4. Ausblick

Der Raspberry Pi bietet die Möglichkeit, eine Kamera anzuschließen und anzusteuern. Der Roboter könnte mit Hilfe der Kamera die Linie erkennen und entsprechend Aktionen ausführen. Es würde erneut Reinforcement Learning zum Einsatz kommen. Für die Kamera bzw. das Erkennen der Linien mit Hilfe der Kamera wäre ein Neuronales Netz notwendig. Es gibt bereits Bibliotheken wie OpenCV, mit denen sich Objekte bzw. Linien erkennen lassen können [8]. Denkbar wäre es, die Library für die Erkennung der Linien zu verwenden und in Kombination mit dem Lichtsensor den Roboter mit Q-Learning darauf zu trainieren, einer Linie zu folgen. Der Lichtsensor spielt hierbei nur für das Training eine Rolle. Er dient dazu, die von der Kamera empfangenen Bilder mit einem Reward zu bewerten. Wenn der Roboter beispielsweise auf der Linie ausgerichtet ist und der Lichtsensor dies als den Zustand mit dem besten Reward erkennt, dann wird dem aktuellem Bild der Kamera dieser Reward zugewiesen.

Literatur

[1] Dennis-Immanuel Czogalla. *Python Code Q-Learning Robot*. 2018. URL: https://github.com/PhonierDeluxe/RaspberryBot/blob/master/LFRL_newest.py (besucht am 20.07.2018).

[2] Dennis-Immanuel Czogalla. *Python Code Simulation*. 2018. URL: https://github.com/PhonierDeluxe/RaspberryBot/blob/master/LightSensorRL.py (besucht am 20.07.2018).

[3] Jörn Fischer. *Vorlesungsfolien Maschinelles Lernen*. 2018. URL: http://services.informatik.hs-mannheim.de/~fischer/lectures/MLE_Files/MLE.pdf (besucht am 20.07.2018).

[4] Raspberry Pi Foundation. *Raspbian Stretch lite*. 2018. URL: https://www.raspberrypi.org/downloads/raspbian/ (besucht am 20.07.2018).

[5] JetBrains. *PyCharm Community Edition*. 2018. URL: https://www.jetbrains.com/pycharm/download/#section=windows (besucht am 20.07.2018).

[6] Pigpio. *Pigpio Library*. 2018. URL: http://abyz.me.uk/rpi/pigpio/ (besucht am 20.07.2018).

[7] Python. *Tkinter Library*. 2018. URL: https://docs.python.org/3/library/tk.html (besucht am 20.07.2018).

[8] Adrian Rosebrock. *Accessing the Raspberry Pi Camera with OpenCV and Python*. 2015. URL: https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/ (besucht am 20.07.2018).

[9] Simon Tatham. *Putty*. 2018. URL: https://www.putty.org (besucht am 20.07.2018).