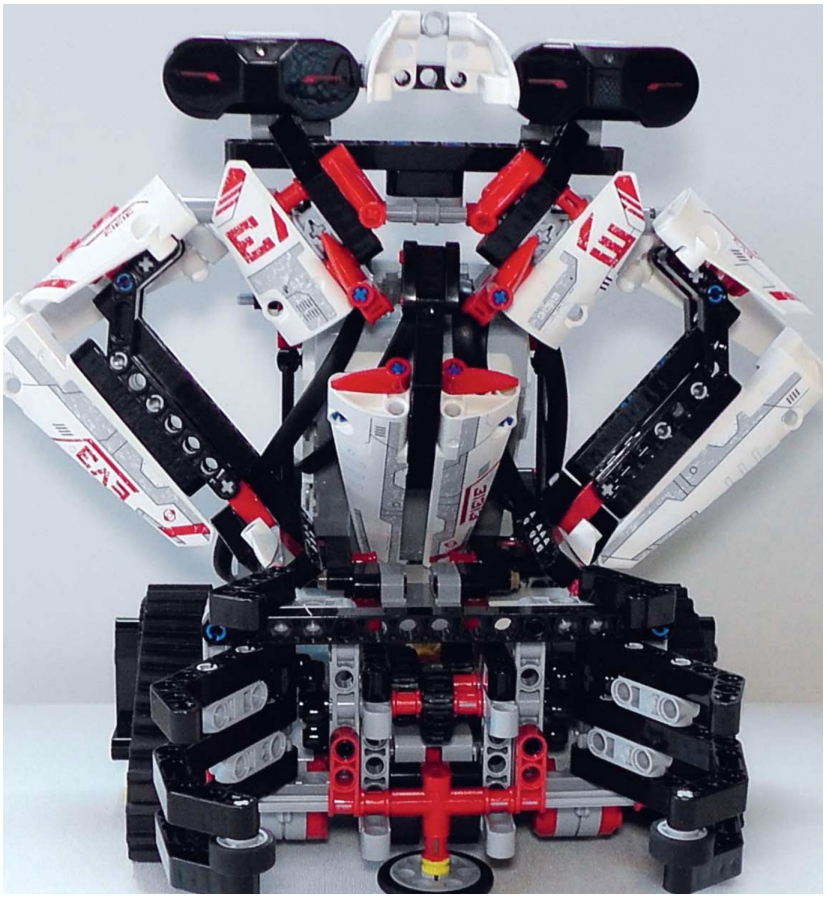
Laboranleitung - Kennenlernen von Künstlicher Intelligenz

Yoshua Schlenker, Lara Knödler

7. März 2024



Inhaltsverzeichnis

1 Einleitung **3**

2 Kontext Künstliche Intelligenz **3**

3 GesturBot **5**
3.1 Allgemeine Beschreibung 6
3.2 Aufbau des Roboters 7
3.3 Datensatz erstellen 7
3.4 Modell trainieren 8
3.4.1 Neuronales Netz 8
3.4.2 K-nearest-neighbors 10
3.5 Modell im Roboter verwenden 11
3.6 Tipps 12

4 Fazit **13**

5 Quellen **13**

1 Einleitung

Künstliche Intelligenz spielt in der heutigen Welt eine immer größere Rolle. Sei es im Autonomen Fahren, in Computerspielen oder in der Satzvervollständigung auf WhatsApp - das Anwendungsgebiet von KI wird immer größer und oftmals wird der Einsatz von KI gar nicht bemerkt. Da es für die meisten Personen relativ schwer ist, sich etwas unter dem Begriff Künstliche Intelligenz vorzustellen, wird in diesem Laborpraktikum versucht, erste Einblicke in die Künstliche Intelligenz, insbesondere dem Machine Learning zu geben.

2 Kontext Künstliche Intelligenz

Der Begriff Künstliche Intelligenz – in Englisch Artificial Intelligence, kurz AI - beschreibt den Versuch, menschliches Lernen und Denken auf Computer zu übertragen und ihm so eine gewisse Intelligenz zu verleihen. Genauer gesagt versucht sie Informationen aus Eingabedaten zu erkennen. Ein Teilgebiet der Künstlichen Intelligenz ist das Machine Learning wie in der unteren Abbildung zu sehen ist. Es beschäftigt sich mit der Entwicklung von Algorithmen und Modellen, die es Computern ermöglichen, aus Daten zu lernen und selbstständig Muster zu erkennen ohne explizit programmiert werden zu müssen.

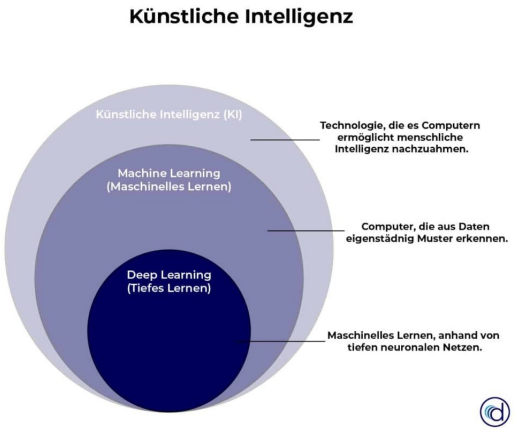


Abbildung 1: Gliederung KI

In den letzten Jahren machte dieser Zweig besonders große Fortschritte aufgrund von immer höheren Rechenleistungen, mehr Speicherplatz und allgemein die Verfügbarkeit von großen Datenmengen. Diese drei Faktoren sind die Grundvoraussetzungen für Machine Learning.

Das Gebiet Machine Learning lässt sich weiter unterteilen: Es beinhaltet die drei Teilbereiche Supervised Learning, auf Deutsch Überwachtes Lernen, Unsupervised Learning, übersetzt Unüberwachtes Lernen und Reinforcement Lear-

ning, nämlich Bestärkendes Lernen. Die Unterschiede der einzelnen Methoden lassen sich in den beiden Abbildungen unten gut sehen:

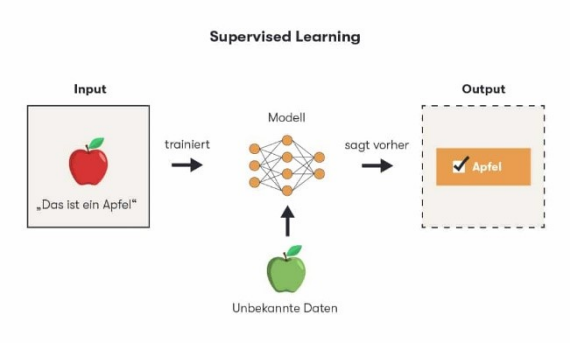


Abbildung 2: Überblick Supervised Learning

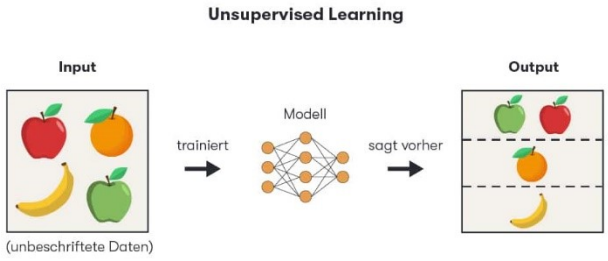


Abbildung 3: Überblick Unsupervised Learning

Unsupervised Learning verwendet nicht gelabelte Daten um Muster und Ähnlichkeiten in Daten zu erfassen und diese in Gruppen zusammenzufassen. Diesen Vorgang nennt man Clustering. Nicht gelabelte Daten bedeutet, dass es keine Ausgabevariablen gibt, die dem Modell während des Training zur Verfügung gestellt werden.

Ein Beispiel des weit verbreiteten Algorithmus kMeans findet sich in untenstehender Abbildung.

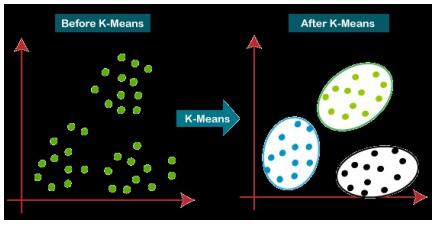


Abbildung 4: Clustering mit kMeans

Reinforcement Learning unterscheidet sich etwas von den beiden anderen Teilgebieten. Es verwendet anders wie die beiden anderen Ansätze keinen statischen Datensatz. Hier soll ein sogenannter Agent, meist eine Software/Algorithmus, durch Interaktion mit einer dynamischen Umgebung lernen, bestimmte Aktionen auszuführen. Der Agent geht dabei nach dem Trial-and-Error Verfahren vor und bekommt Rückmeldung in Form von Belohnungen von der Umgebung, je nachdem wie erfolgreich seine ausgeführte Aktion war. Das Ziel ist die Maximierung der Belohnungen über die Zeit hinweg. Die beiden nachfolgenden Links führen zu zwei Beispielvideos zu Reinforcement Learning: Hide and Seek oder Laufen lernen.

Dieser Versuch beschäftigt sich jedoch mit dem Supervised Learning was besonders bekannt ist durch den Begriff der Neuronalen Netze oder Deep Learning. Das Modell beim Supervised Learning wird dabei mit gelabelten Trainingsdaten trainiert. Es soll die Beziehungen zwischen den Eingabedaten und zugehörigen Ausgabedaten erlernen, um später Vorhersagen auf neue, nicht gelabelte Daten treffen zu können. Diesen Vorgang nennt man Klassifikation.

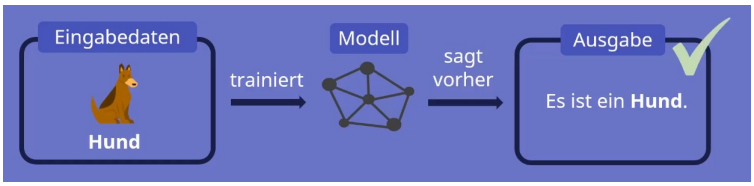


Abbildung 5: Beispiel Klassifikation

3 GesturBot

In diesem Versuch soll eine leicht modifizierte Version des Lego Mindstorms Gripp3r Roboter durch Gesten gesteuert werden können. Die Grundidee dafür kommt von Detlef Heinze, der diesen Versuch mit der internen Entwicklungsumgebung von Lego Mindstorms durchgeführt hat. Dies ist der Link zu seinem GitHub Repository: Heinze GitHub. Wir verwenden jedoch ausschließlich MATLAB für diesen Versuch, da dies relevanter für das Studium ist und zudem deutlich handlicher als die mitgelieferte Programmierumgebung von Lego.

Hier ist der Link zu unserem GitHub Repository: GitHub GesturBot

Am besten ist es einfach, den Master Branch zu klonen, so befinden sich bereits alle wichtigen Unterlagen wie die .m Files in dem Ordner. Dafür befindet sich rechts oben ein grüner Button mit der Aufschrift *Code*. Diesen einfach drücken und danach *Download ZIP* wählen. Nun müsst ihr nur noch das ZIP-Archiv in einen Ordner eurer Wahl entpacken.

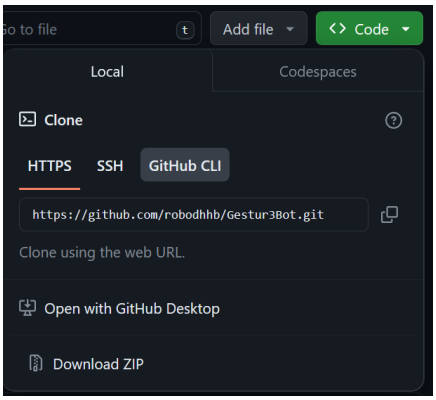


Abbildung 6: Button

3.1 Allgemeine Beschreibung

Ziel des Versuchs ist es, den Roboter mit Handgesten zu steuern. Dabei sind prinzipiell nachfolgende sieben Gesten möglich, wir verwenden jedoch nur drei der Gesten. Einerseits müssen dann nicht so viele Trainingsdaten aufgenommen werden, andererseits ist die Klassifikation zuverlässiger.

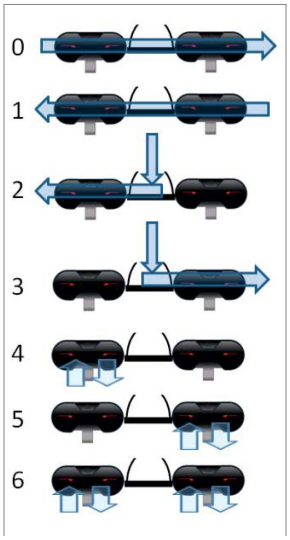


Abbildung 7: Gesten des GesturBot

Wir empfehlen, die Gesten 0, 1 und 6 zu verwenden. Geste 0 wird zum Vorwärtsfahren verwendet, Geste 1 zum Rückwärtsfahren und Geste 6 zum Stoppen. Diese Gesten liefern relativ unterschiedliche Abstandswerte, was die Klassifikation vereinfachen sollte, weil das Modell dann die Muster zwischen

Abstand und zusammenhängender Klasse besser versteht. Um die Geste 6 zu erzeugen müssen beide Hände zuerst auf den jeweiligen IR-Sensor zubewegt werden und dann zurückgezogen werden. Entsprechend funktionieren die Gesten 4 und 5, jedoch nur mit einer Hand auf einen der beiden Sensoren.

Für die Erkennung der Gesten sind die beiden IR-Sensoren zuständig. Sie messen den Abstand zwischen einem Objekt, zum Beispiel eurer Hand, und dem Sensor, indem sie die Intensität des reflektierten Infrarotlichts des Objektes messen. Damit dies gut funktioniert sind die Gesten mit der Faust oder mit der flachen Hand auszuführen. Außerdem hat der GesturBot eine Nase zwischen den IR-Sensoren, die verhindern soll, dass zurückgestrahltes Licht des einen Sensors auf den anderen Sensor fällt. Zudem müssen die Gesten mit mittlerer Geschwindigkeit ausgeführt werden damit die IR-Sensoren die Abstandswerte überhaupt detektieren können. Ein Beispielvideo findet sich im Ordner *Lab Description*.

Jede Geste besteht aus einem Vektor mit 32 Sensorwerten. Von jedem Sensor kommen dabei 16 Werte. Im der Abbildung unten findet sich eine Beispielmessung der Geste 0. Die erste Zeile kommt vom linken Sensor, die zweite Reihe vom rechten IR-Sensor, jeweils vom Benutzer aus gesehen.

```
17 12 12 10 11 20 42 54 54 55 55 56
55 55 55 55
42 39 36 34 32 24 12 12  8  4  8  8
39 39 44 45
```

Abbildung 8: Beispiel Geste 0

Den Weg der Hand kann man hier sehr gut beobachten: Die Zeit läuft von links nach rechts. Dabei werden als erstes die Abstände des linken Sensors kleiner und dann größer, während die Werte des rechten IR-Sensors erst groß sind, dann kleiner werden und schließlich wieder größer.

3.2 Aufbau des Roboters

Dieser Abschnitt ist nur relevant falls der Roboter nicht aufgebaut ist. Ansonsten könnt ihr direkt mit Abschnitt 3.2 *Datensatz erstellen* weitermachen.

Als aller erstes muss der Roboter leicht modifiziert werden, indem ein zweiter IR-Sensor angebracht wird. Hier findet ihr die Bauanleitungen, davon wird die für den Gripp3r-Roboter benötigt: Anleitungen. Alternativ haben wir euch die Anleitung auch in unserem GitHub Repository abgelegt.

Mission 1 und Mission 2 sind so durchzuführen wie in der Lego Anleitung beschrieben. Mission 3 wird jetzt bis zu Schritt 36 durchgeführt, danach müsst ihr euch an den drei Bildern im Ordner *Robot Build* orientieren.

3.3 Datensatz erstellen

Dieser Abschnitt beschäftigt sich mit dem Erstellen der Trainings- und Validierungsdaten für die Modelle. Das Erstellen wird mit dem Skript collectTrainDa-

ta.m bzw. collectValidationData.m gemacht werden.

Die Trainingsdaten sind dazu da, das Modell, wie zum Beispiel ein Neuronales Netz, zu trainieren. Anhand diesen Daten lernt das Modell also die Zusammenhänge und Muster. Die Validierungsdaten sind dazu da, zu überprüfen, wie gut das Modell auf neue Eingaben reagiert. Diese Daten werden **nicht** für das Training verwendet.

Beide Files sind ähnlich aufgebaut. Als erstes werden Objekte initialisiert, die eine Verbindung zu dem Brick und den IR-Sensoren aufbauen. Anschließend werden in einer äußeren Schleife die Samples gezählt und in der inneren for-Schleife die 32 Werte aus dem jeder Sample besteht. Diese Messungen werden dann Zeile für Zeile in der Variable *sensorData* gespeichert, sobald ein Abstand von 28 unterschritten wurde. Spalte 34 jedes Samples stellt die zugeordnete Gestennummer dar. Zum Schluss werden die Daten als ein .csv File abgespeichert.

Jede Geste muss beim Erstellen des Datensatzes 50 mal wiederholt werden. Zusätzlich zu den drei Gesten mit denen der Roboter gesteuert werden soll braucht es noch eine Geste die keine Geste darstellt. Werden, wie vorher beschrieben, die Gesten 0, 1 und 6 ausgewählt, so wiederholt man am besten die anderen vier Gesten, damit diese als ungültige Geste erlernt werden.

3.4 Modell trainieren

Dieser Abschnitt beschäftigt sich mit dem Training des Modells. Dafür werden die beiden Skripte trainNeuralNetwork.m und trainKnn.m gebraucht. Hier geht es um das Erlernen von Mustern in den Daten, um das Gelernte nachher auf neue Bedingungen übertragen zu können.

3.4.1 Neuronales Netz

Neuronale Netze orientieren sich an der Funktionsweise des menschlichen Gehirns. Unser Gehirn nimmt Reize, also Daten zum Beispiel über unsere Augen auf. Diese Daten werden dann von den Nervenzellen in unserem Gehirn, den Neuronen, in immer tiefere Ebenen weitergeleitet und dort ausgewertet, bis am Ende zum Beispiel ein Hund erkannt wird. Ein Neuron empfängt winzig kleine elektrische Signale über die Dendriten und leitet diese über das Axon an die Endknöpchen weiter, falls ein bestimmter Schwellwert überschritten wurde. Diese leiten das Signal an mehrere Tausend andere Nervenzellen weiter.

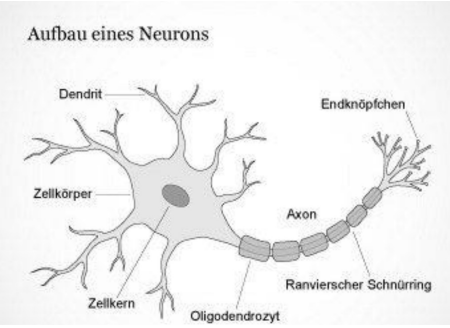


Abbildung 9: Neuron

Ab Geburt ist ein Großteil der Neuronen bereits vorhanden, diese müssen werden lediglich noch richtige verbunden, sodass zum Beispiel ein Hund erkannt wird. Diese Verbindungen nennt man Synapsen. Zudem ist es möglich, dass bei dem Lernprozess eine Verstärkung einiger Axonen beinhaltet, so können Reize schneller weitergeleitet werden.

Ein Künstliches Neuron (Perceptron), siehe Abbildung 10, besteht aus einem Eingabevektor x, einem Gewichtsvektor w, einem Bias b, einer Aktivierungsfunktion ϕ und einem Ausgabewert y. Der Gewichtsvektor w gibt an wie stark jede einzelne Eingabe in das Neuron gewichtet wird. Zusammen mit dem Bias ergeben sie v, den Wert eines Neurons:

$$v = w1x1 + w2x2 + b \tag{1}$$

Dieser Wert wird auf eine Aktivierungsfunktion geleitet, die dann entscheidet ob der Wert über einem Schwellenwert liegt und an den Ausgang weitergeleitet wird. Für ϕ gibt es mehrere Möglichkeiten wie die Relu Funktion, die Sigmoid Funktion oder die tanh Funktion.

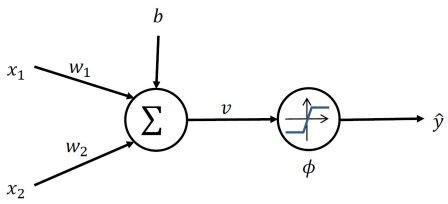


Abbildung 10: Perceptron

Ziel des Trainings eines Neuronalen Netzes ist das Anpassen der Gewichte nachdem diese zufällig initialisiert wurden, siehe Abbildung 11.

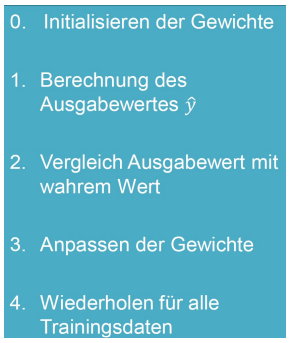


Abbildung 11: Lernregel Supervised Learning

Ordnet man mehrere Neuronen hintereinander an, so entsteht ein Neuronales Netz. Unser verwendetes Netz hat 32 Eingabewerte, eine Hidden Layer mit 20 Neuronen und eine Ausgabeschicht mit 4 Neuronen.

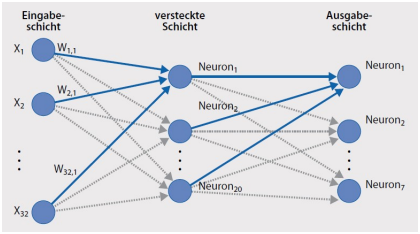


Abbildung 12: Neuronales Netz des GesturBot

Im MATLAB File werden zuerst die Trainings- und Validierungsdaten eingelesen und normalisiert. Dann wird das Neuronale Netz aufgebaut. Dies geschieht indem die einzelnen Schichten mit jeder Aktivierungsschicht angelegt werden. Zusätzlich werden noch einige Optionen für das Training angegeben. Während des Trainings wird in bestimmten Abständen die Güte des NN anhand der Validierungsdaten automatisch bestimmt. Das trainierte Netz wird abschließend im Ordner *Model* abgespeichert.

3.4.2 K-nearest-neighbors

K-nächste-Nachbarn ist eine sehr einfache Klassifikationsmethode. Ein neuer Eingang x wird aufgrund der am häufigsten vorhandenen Klasse der k nächsten Nachbarn zugeordnet. Der Parameter k muss zu Beginn festgelegt werden. Anders wie bei Neuronalen Netzen jedoch muss das Modell **nicht** trainiert werden, sondern es wird direkt eine Vorhersage getroffen. Der Algorithmus berechnet die Distanz des neuen Datenpunktes zu allen anderen Datenpunkten und wählt

dann die k Datenpunkte mit der kürzesten Distanz aus. Durch Mehrheitsentscheid der k nächsten Nachbarn wird die Klasse bestimmt. Die Abbildung unten veranschaulicht diesen Vorgang.

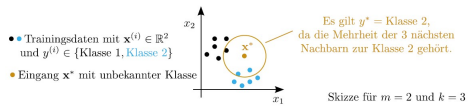


Abbildung 13: Beispiel Knn

Um zu vermeiden dass große Werte einen großen Einfluss auf den Abstand haben können die Daten normiert werden. Zusätzlich ist die Wahl des Parameters k entscheidend. Wählt man k zu klein kann sich Overfitting ergeben, wählt man k jedoch zu klein, so werden die Merkmale der Daten schlecht berücksichtigt.

Im Skript werden zuerst die Trainings- und Validierungsdaten geladen. k wird zu 5 gewählt, das wurde durch Probieren herausgefunden. Auf Basis der Trainingsdaten wird ein KNN Modell erstellt, es müssen aber keine Parameter gelernt werden. Anhand der Validierungsdaten wird die Güte des Knn Algorithmus berechnet. Falls die Klassifikationsgüte hoch genug ist wird das Modell abgespeichert. Ansonsten muss an den Parameter der Funktion *fitcknn()* gespielt werden.

3.5 Modell im Roboter verwenden

Als letzter Schritt wird das bereits trainierte Modell verwendet, um Vorhersagen auf neue, unbekannte Daten zu machen. Dafür werden die beiden MATLAB Skripte gesturBotKNN und gesturBotNN verwendet. Sobald das Skript gestartet wurde, könnt ihr Gesten ausführen und damit den Roboter steuern.

Die beiden Skripte sind relativ ähnlich aufgebaut. Zuerst wird das jeweilige Modell eingelesen. Nachdem die ganzen Komponenten des Roboters initialisiert wurden (Brick, Sensoren, Motoren) läuft die Zustandsmaschine des Roboters für 30 Sekunden, danach wird das Programm automatisch beendet(die Zeit kann auch gerne höher eingestellt werden -¿ Variable *maxExecutionTime*). Es werden Daten aufgenommen, sobald ein Abstand von 30 unterschritten wurde. Das Modell gibt dann für diesen einen Sample eine Vorhersage ab. Je nachdem welche Geste detektiert wurde werden die entsprechenden Motoren angesteuert.

Jetzt sollte der Roboter endlich laufen. Falls doch nicht findet ihr im nächsten Abschnitt *Tipps*, die hoffentlich helfen. Gerne kann auch an dem Roboter mit verschiedene Einstellungen beim Training des Netzes gespielt werden oder es können andere Gesten verwendet werden.

3.6 Tipps

Das ganze Projekt ist sehr empfindliche gegenüber äußere Einflüsse. Deswegen gibt es hier ein paar Tipps, die hilfreich sein könnten, um den Versuch erfolgreich abzuschließen.

1. Batterien: Es ist wichtig, dass die Batterien im Lego Roboter gut geladen sind. Sobald sie eine Genze von 50 Prozent unterschreiten, kann es sein, dass die Werte der IR-Sensoren nichtmehr richtig gemessen werden. Stellt also sicher, dass die Batterien ausreichend geladen sind. Ihr seht es einerseits auf dem Display des Roboters, andererseits sollte der Roboter auch ein Warning in der Konsole ausgeben, falls der Akkustand zu niedrig ist.

   Ebenfalls bei zu niedrigem Akkustand kann es sein, dass die Lego Ev3 Objekte nicht initialisiert werden können, sei es das Brick Objekt, ein Motor Objekt oder ein IR-Sensor Objekt. Es taucht diese Fehlermeldung auf:

   ```
   Error using legoev3
   Failed to connect to the EV3 brick. Please check the USB connection.

   Error in gesturBotNN (line 14)
   ev3brick = legoev3('USB');
   ```

   Abbildung 14: Fehlermeldung Ev3 Brick

   Das Problem kann gelöst werden, indem der Wordspace gecleart wird und das Programm erneut gestartet wird.

2. Geschwindigkeit der Gesten: Es ist wichtig, dass die Gesten mit der richtigen Geschwindigkeit, nicht zu schnell und nicht zu langsam, ausgeführt werden. Alle 40ms wird ein Wert der Sensoren gelesen. Wird die Geste zu schnell ausgeführt, so werden teilweise Veränderungen der Abstandswerte nicht bemerkt, weil die Abtastfrequenz zu langsam ist. Führt man die Geste zu langsam aus, fehlen zum Beispiel die Werte des einen Sensoren, da die Hand noch gar nicht bei diesem Sensor angekommen ist. An dem Beispielvideo im Ordner *Lab description* kann sich orientiert werden.

3. Man darf nicht zu weit von dem Robot entfernt sein. Die Datenaufnahme beginnt erst, wenn ein Abstand von 28 unterschritten wurde. Dies wurde so gewählt, damit die Person vor dem Roboter stehen kann ohne dass die ganze Zeit (ungültige Gesten) erkannt werden.

4. Standort: Es ist zu empfehlen, den gesamten Versuch an einem Ort zu machen, der gut mit Tageslicht durchflutet wird. So können die IR-Sensoren das reflektierte Licht der Hand besser messen.

5. Hand: Außerdem ist es wichtig, die Gesten mit der Faust oder der flachen Hand auszuführen. So wird ebenfalls das Licht besser reflektiert.

6. Die Nummer der Geste, die in der Konsole oder auf dem Display des Roboters angezeigt wird, entsprechen nicht der Nummerierung aus Abbildung 7. Geste 1 ist die Geste zum Vorwärtsfahren, mit Geste 2 wird das Rückwärtsfahren assoziiert, mit Geste 3 das Stoppen und Geste 4 stellt keine zulässige Geste dar. Es können natürlich auch andere Gesten aus den 7 möglichen verwendet werden, die Nummerierung bleibt dennoch die selbe.

4 Fazit

Abschließend gibt es noch ein paar finale Worte zu dem Versuch. Der Versuch ist nicht der komplexeste und auch nicht der spannendste. Aber er bringt die Vorgehensweise von Künstlicher Intelligenz näher. So werden zu aller erst bei jeder Anwendung mit KI Daten benötigt. In unserem Fall sind das nicht viele Daten. Man stelle sich jedoch die ungeheure Menge an benötigten Daten vor, die für das Training von Anwendungen wie ChatGPT benötigt werden. All diese Daten müssen erst einmal gesammelt werden und dann in ein Format gebracht werden, die das Machine Learning Modell verstehen kann. Dieser Aufwand ist nicht zu unterschätzen und macht wahrscheinlich auch nicht den meisten Spaß.

Als nächstes muss das Modell trainiert werden. Bei unserem Roboter ging das Training lediglich einige Sekunden. Je größer die Modelle jedoch werden desto länger dauert auch das Training. Man muss teilweise mehrere Stunden oder Tage warten bis das Training abgeschlossen ist. Viel schlimmer wird es, wenn das Training nach dieser Zeit nicht erfolgreich war. Dann beginnt der ganze Vorgang von vorne.

Hinzu kommt, dass das Modell meistens eine Art Black Box darstellt, in die man nicht hinein schauen kann. Man bekommt als Benutzer lediglich den Input und den Output mit, die innere Funktionsweise ist oft leider undurchsichtig. Die Fehler in der KI dann zu finden gestaltet sich oft schwierig. So kann mach sich auch fragen, inwiefern KI Modelle mit Entscheidungen vertraut werden können, wenn sie in kritischen Bereichen wie zum Beispiel dem Gesundheitswesen eingesetzt werden.

5 Quellen

Vorlesung Supervised und Deep Learning von Herr Raff und Herr Schubert
Artikel: Special: Künstliche Intelligenz - KI für Lego-Roboter von Detlef Heinze
https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html#:~:text=K