Technology
Arts Sciences
TH Köln

Reinforcement Learning am realen und simulierten Cart-Pole-Swing-Up Pendulum im Vergleich

Bachelorarbeit

vorgelegt an der Technischen Hochschule Köln

Campus Gummersbach

im Studiengang Informatik

Matrikelnummer: 11034938

ausgearbeitet von:

Yendoukon Nayante

Erster Prüfer: Prof. Dr. Wolfgang Konen

Zweiter Prüfer: Prof. Dr. Patrick Tichelmann

Gummersbach, im September 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Bestärkendes Lernen oder verstärkendes Lernen (englisch reinforcement learning) steht für eine Reihe von Methoden des maschinellen Lernens, bei denen ein Agent selbstständig eine Strategie erlernt, um erhaltene Belohnungen zu maximieren. Dabei wird dem Agenten nicht vorgezeigt, welche Aktion in welcher Situation die beste ist, sondern er erhält zu bestimmten Zeitpunkten eine Belohnung, die auch negativ sein kann. Anhand dieser Belohnungen approximiert er eine Nutzenfunktion, die beschreibt, welchen Wert ein bestimmter Zustand oder eine bestimmte Aktion hat [1].

Im Rahmen einer Praxisprojektarbeit [2] wurden verschiede Reinforcement Learning Algorithmen auf das Cart-Pole-Swing-Up angewendet. Mit der implementierten DQN-Klasse konnten gute Ergebnisse erzielt werden.

In dieser Bachelorarbeit sollen die Erkenntnisse aus dem Praxisprojekt auf ein reales Pendel angewendet und verglichen werden.

1.1 Verwandte Arbeiten

Reinforcement Learning (im folgenden RL genannt) gewinnt heutzutage immer mehr an Bedeutung und zahlreiche Arbeiten wurden in diesem Bereich durchgeführt. Es hat sich auch herausgestellt, dass das Cart-Pole- oder Cart-Pole-Swing-Up-Modell gerne verwendet wird, um die verschiedenen Algorithmen im Bereich des RL zu vergleichen. Beim Cart-Pole geht es darum, das Pendel zu balancieren, was einfacher ist im Vergleich zum Cart-Pole-Swing-Up, mit dem sich diese Arbeit befasst. Beim Cart-Pole-Swing-Up muss das Pendel erst aufgeschwungen und dann balanciert werden. In der Simulation verwenden wir das Cart-Pole-Swing-Up-Modell von Freeman [3] mit kleinen Änderungen, wie bei der Praxisprojektarbeit [2]. Für die Lösung des Cart-Pole-Swing-Up-Problems wurde bei Freeman [3] der Population-based Reinforcement Algorithmus verwendet. Bei der Arbeit von Riedmiller [4] wurde kein Schlitten verwendet. Die Aktionen basierten auf der Drehung durch einen DC Motor an der Pendeldrehachse und es standen zwei Aktionen zur Verfügung (DC Motor Achse dreht sich nach rechts oder links). Durch die Drehung des DC-Motors wird das Pendel in Bewegung versetzt. Riedmiller [4] verwendet ein nicht lineares neuronales Netz, um den Agenten zu trainieren, wie bei Anderson [5]. Bei Anderson [5] gibt der Output Layer neben der üblichen Ausgabe der Q-Funktion, auch die Änderung im Zustand $(S_{t-1} - S_t)$ aus, wohingegen wir in unserer Simulation aus dem Praxisprojekt [2] nur die Q-Funktion ausgeben. Außerdem verwendet Anderson [5] beim Cart-Pole-Swing-Up-Problem, neben der Zustandsvariablen auch die Aktion als Input für das neuronale Netz. Bei Anderson [5] wird ein neuronales Netz zunächst trainiert, um Veränderungen der Zustandsvariablen auf der Grundlage des aktuellen Zustands und Aktionen zu bestimmen, dann wird dasselbe neuronale Netz mit vortrainierten Gewichten weiter trainiert, um den Q-Wert zu approximieren und die Summe der zukünftigen Rewards vorherzusagen. Camilo [6] zeigt in seinem Artikel auch eine Lösung des Cart-Pole-Swing-Up-Problems in der Simulation unter Anwendung des ActorCritic

Algorithmus. Dabei wurde der Einfluss der Reibung auf das Lernverhalten des Agenten gründlich untersucht. Es sind auch mehrere Berichte und Videos über reale und verschiedene inverse Doppelpendel [7], inverse Trippelpendel [8] im Internet zu finden, aber in den meisten Fällen, wurde das Cart-Pole-Swing-Up-Problem in diesen Fällen unter Anwendung der Regelungstechnik gelöst. In dieser Arbeit verwenden wir die DQN-Klasse aus dem Praxisprojekt [2] und die Implementierung unterscheidet sich von der zuvor genannten verwandten Arbeit. Außerdem ist das reale Pendel, auf das sich diese Arbeit bezieht, im Labor für künstliche Intelligenz von Prof. Dr. Tichelmann an der TH Köln konzipiert worden und besitzt andere Rahmenbedingung als die meisten untersuchten Cart-Pole-Swing-Up-Pendel.

1.2 Aufgabenstellung

In der Praxisprojektarbeit [2] haben wir das Cart-Pole-Swing-Up-Problem gründlich untersucht und am Ende eine optimierte Lösung gefunden. Mit dieser Lösung wollen wir jetzt an den realen Versuch herangehen, mit der Hoffnung, dass die optimierte Lösung aus der Simulation sich auch im Versuch bewährt. Im Labor für künstliche Intelligenz von Prof. Dr. Tichelmann an der TH Köln wurde in der Vergangenheit gleichermaßen vorgegangen, um das Problem des Cart-Pole-Swing-Up zu lösen. Dabei hat man zwar bei der Simulation gute Ergebnisse erzielt, im realen Versuch jedoch nicht. Dies zeigt, dass die Rahmenbedingungen des Versuchs sich von denen der Simulation unterscheiden. Andernfalls wäre zu erwarten gewesen, dass man aus einem guten Ergebnis in der Simulation auf ein ebenso gutes Ergebnis beim realen Versuch schließen kann. Es ist klar, dass im realen Versuch viele Reibungskräfte auftreten, die weder in den vorangegangenen Simulationen im Labor von Prof. Dr. Tichelmann noch in den hier beschriebenen Simulationen berücksichtigt wurden. Außerdem braucht wahrscheinlich der Antrieb, der den Schlitten steuert, eine bestimmte Mindestzeit, um den Schlitten in Bewegung zu versetzen. Alle diese Faktoren haben wir nicht in der Simulation berücksichtigt. Dies könnte untersucht werden, falls sich die Simulationsergebnisse im Versuch nicht bestätigen lassen. Es kann aber auch sein, dass die DQN-Klasse aus der Praxisprojektarbeit [2] angepasst und die Parameter der neuronalen Netze optimiert werden müssen. Es ist auch klar, dass, wenn man das Pendel bei einem großen Anschlagwinkel starten lässt, der Agent größere Reward-Werte am Anfang bekommt als mit kleinem Startwinkel. Deswegen muss der Einfluss der Startbedingung auf das Lernverhalten des Agenten auch untersucht werden.

Der implementierte DQN-Algorithmus ist im Praxisprojektbericht (im Folgenden PPB) [2], ab Seite 30 beschrieben. Deswegen werden wir in dieser Bachelorarbeit den Code nicht nochmal beschreiben.

Bevor wir auf den realen Versuch eingehen, schauen wir uns die Funktionsweise des Inverted Pendulums in der Simulation und im Versuch an.

2 Das simulierte und reale Cart-Pole-Swing-Up Pendulum

Das Cart-Pole-Swing-Up ist ein physikalisches System, das sowohl in der Simulation als auch im Versuch aus einem fahrbaren Schlitten und einem daran angebrachten Pendel besteht. Das System soll erlernen, den Schlitten so zu bewegen, dass der Pendelstab aufgeschwungen und durch balancierende Ausgleichsbewegungen im Zenit gehalten wird. Um dieses Ziel zu erreichen, schauen wir uns zuerst die Funktionsweise des Systems in der Simulation und deren Ergebnisse an, bevor wir zum Versuchsaufbau kommen.

2.1 Das Cart-Pole-Swing-Up in der Simulation

In der Praxisprojektarbeit [2] ist das Cart-Pole-Swing-Up in OpenAI Gym [9] simuliert.

Gym ist eine Open Source Software, die von der Firma OpenAI entwickelt wird und eine umfassende Sammlung von Umgebungen mit einheitlicher Schnittstelle für RL-Experimente anbietet. Abbildung 1 zeigt das Cart-Pole-Swing-Up.

Folgende Schritte wurden bei der Simulation zugrunde gelegt, um die Ergebnisse zu bekommen:

1. Laden der untersuchten Umgebung

Beim Praxisprojekt [2] wurde eine Klasse mit dem Klasse-Namen CartPoleSwingUpEnv() für die Umgebung bereitgestellt. Diese Klasse erbt vom Gym-Environment und stellt uns u. a. folgende Methoden bereit:

- reset(self): setzt den Zustand der Umgebung zurück und gibt eine Observation zurück.
- step(self, action): Ausführung einer Aktion in der Umgebung für einen Zeitschritt. Gibt eine Observation, den Reward, die Variable done und eine Info zurück. Die Variable done ist ein Boole'scher Wert, der True ist, wenn die Episode vorbei ist. Info gibt zusätzliche Informationen, die von der jeweiligen Umgebung abhängig sind und kann für RL Methoden gewöhnlich ignoriert werden.

Außerdem kann durch die Klassen-Definition, eine andere Klasse die Cart-Pole-Swing-Up-Klasse aufrufen und alle Methoden der Klasse können benutzt werden.

2. Definition der Netzarchitektur

Hier legen wir eine separate Klasse für die Netzarchitektur an und können die Vererbung der objektorientierten Programmierung voll nutzen. In der Netzarchitektur legen wir u. a. die Anzahl der Neuronen, die Optimierungsverfahren und die Aktivierungsfunktion fest.

3. Die Agent-Klasse

Die Hauptfunktion dieser Klasse ist, die Daten aus der Cart-Pole-Swing-Up Umgebung zu sammeln und einen Agenten mit der festgelegten Netzarchitektur zu trainieren. Deswegen

ruft diese Klasse die beiden vordefinierten Klassen nämlich die Cart-Pole-Swing-Up-Klasse und die Netzarchitektur-Klasse auf. Alle drei Klassen sind im Anhang zu finden.

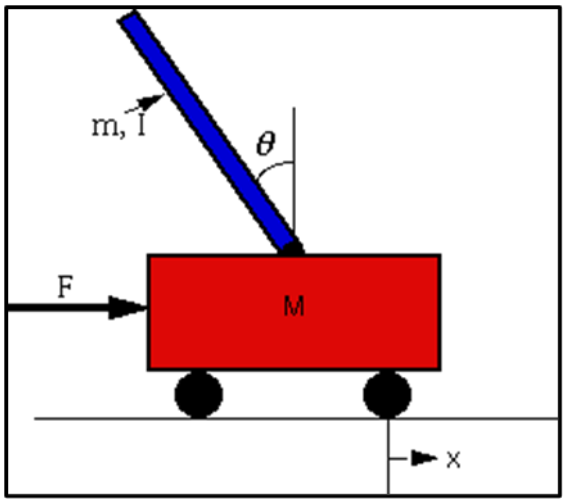

Abbildung 1: Beispiel Inverted Pendulum [10]

Nach dem Training werden 100 Episoden gespielt und totalReward, zenitCounter und firstTimestepAtZenit Diagramme aufgezeichnet.

- Das Diagramm totalReward zeigt die Summe aller Rewards in jeder Episode in Abhängigkeit der Episode.
- Im firstTimestepAtZenit-Diagramm (im folgenden 1stTimesAtZenit) wird für jede Episode, die Anzahl der Zeitschritte dargestellt, nach der das Pendel zum ersten Mal am Zenit angekommen ist (eine Episode besteht aus maximal 1000 Zeitschritte).
- Bei dem Diagramm zenitCounter wird die Anzahl der Schritte während einer Episode bei denen das Pendel im Zenit steht, gezählt und in Abhängigkeit der Episode dargestellt.

Wir definieren das Pendel als im Zenit stehend, wenn das Pendel einen Winkel kleiner als fünf Grad hat und die Winkelgeschwindigkeit kleiner als 6 rad/s ist.

Die Basisergebnisse mit zwei Hidden Layers mit einer Größe von jeweils 64 Neuronen und der Aktivierungsfunktion Relu am Ausgang eines jeden Hidden Layer, ist in Tabelle 1 zusammengefasst.

Ergebnis für 100 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	897	7	1000	0
zenitCounter	835	16	keine	keine
1stTimesAtZenit	163	16	keine	keine

Tabelle 1: Standardabweichungen und Mittelwerte der DQN-Klasse Ergebnisse nach dem Training

2.2 Das reale Cart-Pole-Swing-Up

Das reale Cart-Pole-Swing-Up wurde im Labor für angewandte künstliche Intelligenz von Prof. Tichelmann an der TH Köln konzipiert. Es handelt sich hierbei um einen realen Pendelprüfstand, an dem diverse Reinforcement-Learning-Algorithmen auf das Funktionieren unter realen Bedingungen untersucht und optimiert werden können. Zur Ansteuerung des Schlittens wird ein Mikrocontroller (Raspberry Pi) verwendet. Die gesammelten Daten zur Bewertung der Bewegungen werden in einer Cloud zwischengespeichert (hierbei wird die Cloud „Dropbox“ benutzt, siehe nächster Abschnitt) und von dort aus auf einen Rechner mit höherer Leistungsfähigkeit übertragen, um dort das neuronale Netz zu trainieren. Danach wird das neuronale Netz wieder an den Mikrocontroller übergeben.

Wir haben bei dem realen Pendel drei Systeme, die miteinander kommunizieren, um einen Agenten zu trainieren.

- Der Mikrocontroller (Raspberry Pi): steuert den Schlitten, um das Pendel in Bewegung zu versetzen. Wir können den Mikrocontroller als unsere Gym-Umgebung ansehen und die wichtigsten Funktionen aus OpenAI Gym [9], die nötig sind, um Daten aus der Umgebung zu generieren, hier implementieren.
- Der Cloud-Speicher: hier verwenden wir Dropbox, um die Daten zu speichern.
- Der Hochleistungs-PC: auf dem PC wird das neuronale Netz trainiert.

Der reale Versuchsaufbau ist in Abbildung 2 zu sehen.

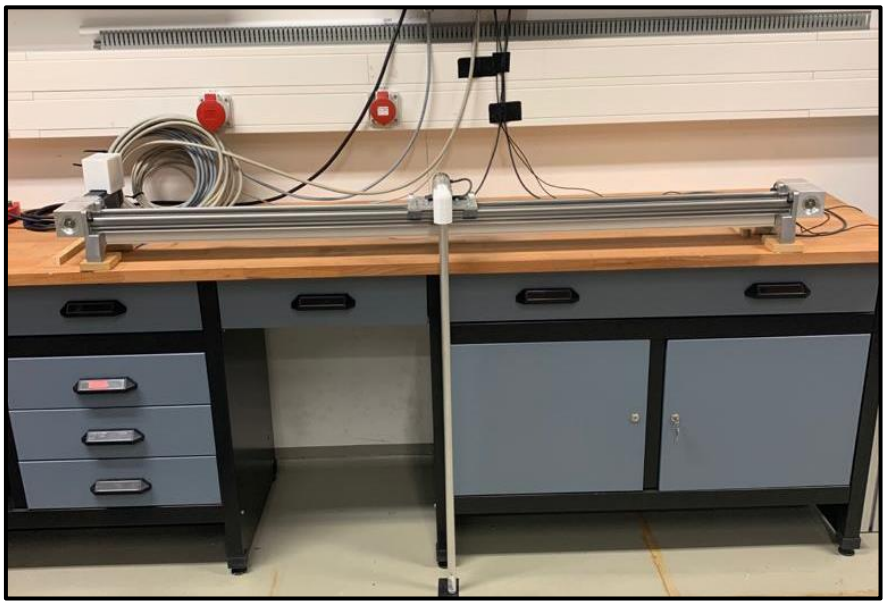

Abbildung 2: Der reale Pendelprüfstand

Im Gegensatz zu der Simulation, wo wir nur einen Rechner benutzen, um den Agenten zu trainieren, benötigen wir beim realen Pendel zwei Rechner und einen Cloudspeicher.

Jetzt schauen wir uns die Funktionsweise des Cloudspeichers Dropbox an.

2.2.1 Cloudspeicher: Dropbox

Dropbox ist ein Cloudspeicher, der uns ermöglicht Daten zwischen verschiedenen Geräten zu senden. Er funktioniert ähnlich wie OneDrive von Microsoft, wo wir auch unsere Daten manuell zwischen dem eigenen Computer und Microsoft OneDrive verschieben können, bietet aber zusätzlich die Funktion, Daten anhand von Python Code zwischen den verschiedenen Geräten zu senden: das ist gerade für unseren Mikrocontroller Raspberry und den PC zum Trainieren des Netzes sehr nützlich, da wir Python Programme für das RL benutzen. Im Folgenden beschreiben wir kurz die Installationen, die für diese Bachelorarbeit nötig sind, um die Kommunikation zwischen dem Raspberry und dem PC über Dropbox zu ermöglichen.

a) Einrichten der Dropbox auf dem PC

Zuerst muss man das Dropbox Programm auf den Geräten einrichten, danach legt man einen User Account an [11]. Unter dem folgenden Link kann man Dropbox herunterladen und installieren:

https://www.dropbox.com/de/install.

Für die Synchronisierung der Daten kann man optional einen Ordner auf dem eigenen Computer einlegen. Nur die Daten dieses Ordners werden auf der Dropbox Seite sichtbar sein.

Durch die Installation wird einen Dropbox-Ordner mit einem Dropbox-Icon auf dem PC erstellt. In diesem Ordner kann man andere Ordner oder Daten ablegen und die Änderung werden sofort auf dem Dropbox-Browser sichtbar.

Abbildung 3 zeigt den Dropbox-Browser. Die gleiche Ordnerstruktur ist auf dem PC zu finden (z. B.: MeinPC\Dropbox\Apps).

Abbildung 3: Dropbox Seite und Synchronisierte Ordner

b) Einrichten von Dropbox auf dem Raspberry

Die gleiche Vorgehensweise zur Installation von Dropbox auf dem PC könnte man hier wieder anwenden. Leider hat Dropbox keinen ARM-Client für das Linux-Betriebssystem veröffentlicht. Deswegen müssen wir zur Nutzung von Dropbox auf dem Mikrocontroller eine Dropbox-App und ein Python-Skript verwenden.

Auf der folgenden Seite ist gut erklärt, wie man Dropbox auf dem Raspberry einrichtet:

https://pimylifeup.com/raspberry-pi-dropbox/

Für das Pendel haben wir eine App mit dem Namen „RaspiPendel“ angelegt und in dieser App einen Unterordner mit dem gleichen Namen. Auf dem PC kann man dann zum Beispiel über den Pfad: MeinPC\Dropbox\Apps\RaspiPendel\RaspiPendel den Ordner der App erreichen und somit auch Daten hoch-/ runterladen. Das gleiche kann man auf dem Raspberry (im folgenden Raspi) mithilfe des Python-Skripts machen.

Für den Daten-Transfer zwischen dem Raspi und der Dropbox-App legen wir einen Ordner auf dem Raspi fest (z.B.: /home/raspi3b+/Desktop/NN_transfer/PPBot_6/). Jetzt können wir ein Skript schreiben, um Daten von dem Raspi auf Dropbox hoch-/ runterzuladen.

Der folgende Code zeigt zum Beispiel, wie wir eine Datei Namens „LetzteErfahrung.csv“ von dem Raspi auf Dropbox laden können:

```
1   PathCloud = '/RaspiPendel/'
2   PathHome='/home/raspi3b+/Desktop/NN_transfer/PPBot_6/'
3   db = dropbox.Dropbox('AccessToken')
4   f  = open(PathHome+'LetzteErfahrung.csv','rb')
5   upname = PathCloud +  'LetzteErfahrung.csv'
6   db.files_upload(
7                   f.read(), upname,
8                    mode=dropbox.files.WriteMode("overwrite")
9                         )
10  f.close()
```

Code 1. Python Skript Beispiel zum Laden von Daten vom Raspberry auf Dropbox

In Zeile 1 von Programmcode 1, legen wir den Dropbox App Ordner fest, in Zeile 2 den Ordner auf dem Raspi. In Zeile 3 legen wir eine Variable Namens db mit dem „ACCESSTOKEN“ der erzeugten App an. Mit dieser Variablen sind wir in der Lage, Dropbox Funktionen zu nutzen, wie:

- files_upload: hochladen von Daten auf Dropbox.
- files_download_to_file: herunterladen von Daten von Dropbox zu einem lokalen Ordner.
- files_list_folder: um Inhalte von einem Ordner zu sichten.

Weitere Funktionen sind auf der Seite https://www.kite.com/python/docs/dropbox.Dropbox zu finden. Die restlichen Codezeilen vom Programmcode 1 dienen zum Hochladen der Datei „LetzteErfahrung.csv“ vom Raspberry auf Dropbox.

Die Datei LetzteErfahrung.csv kann somit auch von Dropbox auf den PC heruntergeladen werden, obwohl die Datei von dem Raspi kommt. Somit können die beiden Rechner (PC und Raspi) die Daten austauschen. Sie benutzen als „Übermittler“ die Dropbox-App.

Ähnlich wie im Code 1 können wir auch Daten von Dropbox auf den Raspi herunterladen. Im Programmcode 2 zum Beispiel, sind wir in der Lage, die Datei model_new.h5 von Dropbox auf den Raspi zu laden. Die Ordner PathCloud und PathHome sind die gleichen wie im Programmcode 1.

```
1   db = dropbox.Dropbox('ACCESSTOKEN')
2   db.files_download_to_file(PathHome+'model_new.h5',
3   PathCloud+'model_new.h5')
```

Code 2. Python Skript Beispiel zum Herunterladen von Daten von Dropbox auf Raspi

Nachdem wir den Cloudspeicher auf den beiden Rechnern eingerichtet haben, müssen wir schauen, wie wir auf einer Seite die Variablen, mit denen wir das Netz auf dem PC trainieren, bekommen können und wie wir auf der anderen Seite, das trainierte Modell auf das reale Pendel anwenden können.

c) Kommunikation zwischen Raspberry und PC

Bei der Simulation im OpenAI Gym [9] sind die Observation, der Reward, done (Spielzustand) durch das zugrunde liegende Environment gegeben. Anhand von diesen Werten kann man das neuronale Netz trainieren.

Die gleiche Vorgehensweise wird auch am realen Pendel umgesetzt. Der Raspberry, auf dem das Pendel gesteuert wird, ist unser Environment und liefert uns wie beim Gym-Environment die Observation, den Reward und done.

Der Austausch von Daten zwischen dem Mikrocontroller (Raspberry) und dem PC geschieht folgendermaßen:

1. Der Mikrocontroller speichert die Zustände, den Reward, den Spielzustand und die Aktionen während einer Episode in einer Liste mit dem Namen LetzteErfarung.csv.
2. Die Liste LetzteErfarung.csv und eine Textdatei MeldungRaspi.csv bestehend aus dem String „1“ (es kann auch ein leerer String sein), werden am Ende der Episode

auf der Dropbox gespeichert. Die Textdatei soll dem PC signalisieren, dass die aktuelle Episode beendet wurde und die Daten zum Trainieren des Netzes (die Liste LetzteErfahrung.csv) nun vorliegen.

3. Der Rechner holt dann die Liste LetzteErfahrung.csv aus der Dropbox, löscht die Textdatei MeldungRaspi.csv, speichert das neue trainierte Netz (model_new.h5) und eine neue Textdatei MeldungPC.csv auf der Dropbox. Wie bei der Textdatei MeldungRaspi.csv, teilt die Textdatei MeldungPC.csv dem Mikrocontroller mit, dass das neue trainierte Netz fertig ist.
4. Der Mikrocontroller holt dann das trainierte Netz aus Dropbox, löscht die Textdatei MeldungPC.csv und das Ganze wiederholt sich, bis der Lernprozess beendet wird.

Die Kommunikation zwischen Raspberry, Dropbox und PC lässt sich als Komponentendiagramm visualisieren. Abbildung 4 zeigt ein Beispiel für die Kommunikation zwischen den drei Systemen.

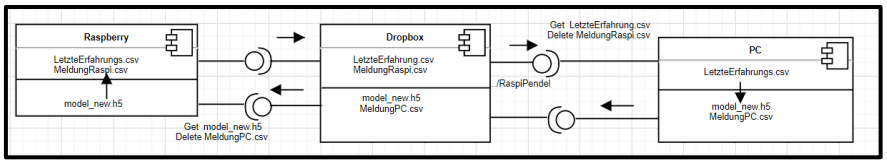

Abbildung 4: Komponentendiagramm Raspberry-Dropbox-PC

Bei der Simulation in OpenAI Gym [9] hat, wie bereits erwähnt, die DQN-Klasse das beste Ergebnis geliefert. Im nächsten Kapitel schauen wir uns die Implementierung der DQN-Klasse für die beiden Komponenten PC und Raspberry genauer an.

3 Programmcode für das reale Pendel

Für das Trainieren des Netzes und für die Nutzung des Netzes, um zum Beispiel eine Aktion zu bestimmen, wird das DQN-Klasse-Modell aus dem Praxisprojekt [2] verwendet (siehe Praxisprojektbericht [2] ab Seite 30). Die DQN-Klasse wurde bereits ausführlich während des Praxisprojekts erklärt, daher werden wir nur auf den wichtigsten Aspekt zum Trainieren auf dem PC und für die Nutzung des trainierten Netzes auf dem Raspberry eingehen.

In der Simulation in OpenAI Gym [9] hatten wir drei Klassen definiert (dqn.py, CartPoleSwingUpEnv.py, dqnAgent.py) um das Modell zu trainieren. Im Versuch benötigen wir dagegen auf dem PC keine CartPoleSwingUp-Klasse, da die Umgebungsdaten vom Raspi kommen. Auf dem Raspi benötigen wir auch keine dqnAgent-Klasse zum Trainieren des Netzes, da das Training auf dem schnellen Rechner (also dem PC) stattfindet. Wir benötigen aber auf beiden Rechnern die Netzarchitektur-Klasse dqn.py. Die zwei Klassen auf dem PC nennen wir Pendel_PC_4Obs_class.py und dqn_PC_class.py. Auf dem Raspi sind die Klassen: Pendel_Raspi_4Obs_class.py und dqn_Raspi_class.py.

Die DQN-Netz-Klasse wird im Konstruktor [12] von den beiden Klassen Pendel_PC_4Obs_class.py und Pendel_Raspi_4Obs_class.py aufgerufen. Zum Start des Versuches, legt der PC das DQN-Modell (neuronales Netzwerk) auf der Dropbox ab und der Raspi holt sich das Modell. Dies ist nötig, damit die beiden Netze (auf dem PC und Raspi) am Anfang des Trainings kompatibel sind. Sollten zum Beispiel beide Rechner verschiedene Netzarchitekturen haben, bekommt man eine Fehlermeldung beim Start des Versuches. Wie im PPB [2] beschrieben wurde, beginnt der PC das Netz zu trainieren, wenn die gesamte Trainingsliste (englisch replay buffer) mindestens eintausend Datensätze beinhaltet. In folgender Beschreibung gehen wir davon aus, dass dies zutrifft.

Der Raspi generiert bei der ersten Episode die Trainingsdaten mit untrainiertem Netz, sendet die Daten an den PC über Dropbox. Dann holt der PC die Daten, legt wieder das gleiche untrainiertes Netz auf der Dropbox ab (Begründung siehe unten) und informiert den Raspi über die Textdatei MeldungPC.csv, dass er weitermachen kann.

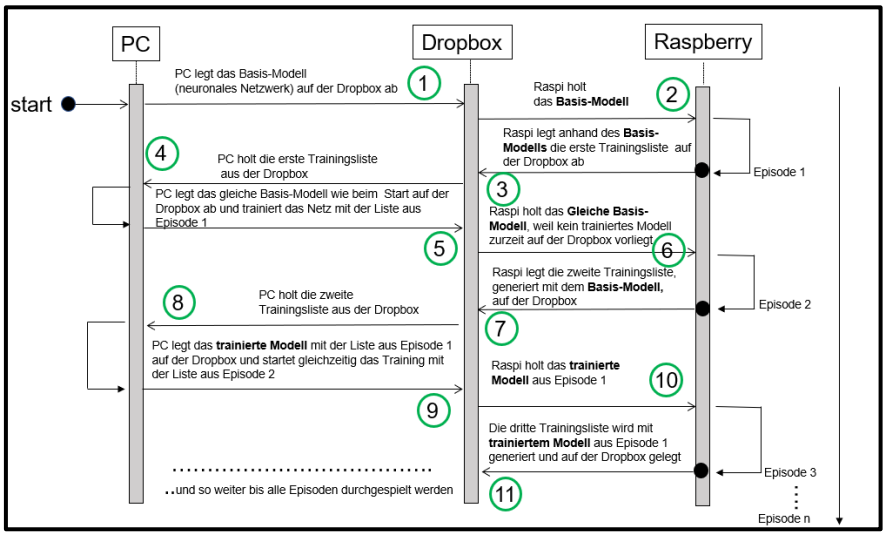

Abbildung 5: Datenaustausch zwischen Raspberry Dropbox und PC im Verlauf der Episoden

Bei der zweiten Episode bekommt der PC wieder die Liste von der zweiten Episode über Dropbox, legt jetzt aber ein trainiertes Modell basierend auf der Liste aus den ersten Episoden auf der Dropbox ab, sodass der Raspi Trainingsdaten von den dritten Episoden mit dem fertig trainierten Modell aus Episode 1 generiert. Das wiederholt sich, bis die eingestellte Episodenanzahl durchgespielt wurde. Der Raspi ist zwei Schritte voraus mit den Episoden. Dies ist nötig, damit man Zeit sparen kann. Ansonst würde der Raspi nach Episode 1, warten bis der PC das Modell trainiert hat und dann mit diesem Modell die Trainingsdaten der Episode 2 generieren. Die ganze Vorgehensweise für die ersten drei Episoden lässt sich mit der Abbildung 5 gut visualisieren. Die eingekreisten Nummern auf Abbildung 5 zeigen die Reihenfolge vom Datenaustausch. Im Programmcode gehen wir auf die einzelnen Nummern genauer ein.

3.1 dqn_PC_class.py Klasse

Die dqn_PC-Klasse ist der DQN-Klasse aus dem Praxisprojekt [2] ähnlich. Die Netzarchitektur besteht aus zwei Hidden Layers mit einer Größe von jeweils 64. Als Aktivierungsfunktion wird Relu oder Tanh am Ausgang eines jeden Hidden Layer gelegt.

Der Input für den ersten Layer ist die Observation bestehend aus 4 Tupeln (θ, $\dot{\theta}$, y, $\dot{y}$) und der letzte Layer besteht aus drei Ausgängen, da wir drei Aktionen haben (Links, Rechts und Stopp).

Als Optimizer wird Adam benutzt, als Loss-Funktion der mean-square-error. Code 3 zeigt die Netzarchitektur.

```
1     def build_model(self) -> Model:
2           input_state = Input(shape=self.state_shape)
3           x = Dense(units=64)(input_state)
4           x = Activation("relu")(x)
5           x = Dense(units=64)(x)
6           x = Activation("relu")(x)
7           q_value_pred = Dense(self.num_actions)(x)
8           model = Model(
```

```
9               inputs=input_state,
10              outputs=q_value_pred
11          )
12          model.compile(
13              loss="mse",
14              optimizer=Adam(learning_rate=self.learning_rate)
15          )
16          return model
```

Code 3. Netzarchitektur DQN-Klasse

Für die DQN-Klasse werden neben der Definition der Netzarchitektur weiteren Methoden wie zum Beispiel das Laden des Modells oder die Aktualisierung der Netzgewichte bereitgestellt [2].

3.2 Pendel_PC_4Obs_class.py Klasse

Die Klasse Pendel_PC_4Obs_class.py (im Folgenden PendelPC-Klasse) funktioniert ähnlich wie die dqnAgent.py Klasse aus dem Praxisprojekt [2]. Hier gehen wir auf die neu implementierten Methoden ein.

Im Code 4 ist ein Abschnitt aus dem Konstruktor [12] zu sehen.

```
1   PathDropbox = 'MeinPC/Dropbox/Apps/RaspiPendel/RaspiPendel/'
2   MODEL_PATH = "MeinPC/Desktop/temp_joel/dqn_new.h5"
3   DROPBOX_MODEL = "MeinPC/Desktop/temp_joel/model_new.h5"
4   TARGET_PATH = "MeinPC/Desktop/temp_joel/Target_new.h5"
5   class Agent():
6       def __init__(self):
7           # DQN Env Variables
8           self.observations = 4
9           self.actions = 3
10          # DQN Agent Variables
11          self.LetzteErfahrung = np.zeros((1,11))
12          self.Erfahrung  = np.zeros((1,11))
13          self.Imax = 1000
14          self.replay_buffer_size = 50_000
15          self.train_start = 1_000
16          self.memory: Deque =
collections.deque(maxlen=self.replay_buffer_size)
17          self.gamma = 0.99
18          self.alpha =0.99
19          self.epsilon = 1.0
20          self.epsilon_min = 0.01
21          self.epsilon_decay = 0.995
22          # DQN Network Variables
23          self.state_shape = self.observations
24          self.learning_rate = 1e-3
25          self.dqn = DQN(
26              self.state_shape,
27              self.actions,
28              self.learning_rate
```

```
29          )
30          self.target_dqn = DQN(
31              self.state_shape,
32              self.actions,
33              self.learning_rate
34          )
35          #update target and save model on dropbox
36          self.target_dqn.update_model(self.dqn)
37          self.batch_size = 32
38          self.dqn.save_model(PathDropbox+'model_new.h5')
```

Code 4. Code-Abschnitt Pendel_PC_4Obs_class.py Konstruktor

In Zeile 1 legen wir den Ordner der Dropbox-App auf dem PC fest und speichern in diesem Ordner das DQN-Modell mit dem Befehl in Zeile 38 unter dem Namen model_new.h5 ab. Zeile 38 entspricht z. B. der Nummer 1, 5 oder 9 auf der Abbildung 5.

Eine der wichtigsten Methoden der PendelPC-Klasse ist, die Trainingsdaten von der Dropbox herunterzuladen. Diese Funktion wird von der Methode loadExperienceList() bereitgestellt.

```
1   def loadExperienceList(self):
2       self.lastExperience = np.zeros((1, 11))
3       print('Überprüfen, ob der Raspi die Episode beendet hat...')
4       while True:
5           try:
6               if os.path.isfile(PathDropbox + 'MeldungRaspi.csv'):
7                   time.sleep(1)
8                   os.remove(PathDropbox + 'MeldungRaspi.csv')
9                   break
10          except FileNotFoundError:
11              pass
12      print('MeldungRaspi gefunden')
13      # load last experience
14      self.lastExperience = open(PathDropbox +
'LetzteErfahrung.csv')
15
16      self.lastExperience = np.genfromtxt(self.lastExperience,
delimiter=",", skip_header=0)
17      # save file "MeldungPC" on dropbox
18      try:
19          np.savetxt(PathDropbox + 'MeldungPC.csv', [1], fmt='%i')
20          print('Meldung an Raspi gesendet')
21      except:
22          time.sleep(0.01)
23      # append experience List
24      if self.lastExperience.ndim <= 1 or self.lastExperience.size
<= 1:
25          self.lastExperience = self.lastExperience.reshape(-1, 1)
26      try:
27          self.experience = np.append(self.experience,
self.lastExperience, axis=0)
28      except:
```

```
29          os.remove(PathDropbox + 'LetzteErfahrung.csv')
30          self.lastExperience = np.zeros((1, 11))
31          time.sleep(0.01)
32      np.savetxt('GesamteErfahrung.csv', self.experience,
delimiter=',', fmt='%1.7f')
33      print('Länge gesamte Erfahrung',
np.shape(self.experience)[0])
34      return self.lastExperience
```

Code 5. loadExperienceList() Methode

Beim Aufruf der Methode loadExperienceList() wird in der while-Schleife so lange geprüft, ob die Textdatei MeldungRaspi.csv auf Dropbox vorhanden ist, bis die Datei gefunden wurde. Dann wird sie gelöscht und die while-Schleife ist beendet. Jetzt wissen wir, dass eine Trainingsliste vorhanden ist und diese Liste wird in die festgelegte leere Liste self.last-Experience Zeile für Zeile kopiert. Die Liste self.lastExperience erhält nur die Daten der aktuellen Episode. Deswegen wird diese Liste am Anfang der Methode mit dem Befehl: self.lastExperience =np.zeros((1,11)) auf eine leere Liste, bestehend aus einer Zeile und 11 Spalten zurückgesetzt, um die alten Daten aus der vorherigen Episode zu löschen. Die elf Spalten stehen für die zwei Zustände (aktueller Zustand und nächster Zustand, insgesamt 8 Variablen), den Reward (eine Variable), die Aktion, die vom aktuellen Zustand zum nächsten Zustand geführt hat (eine Variable) und die Variable done.

Die Liste self.lastExperience wird in einer Liste Namens self.experience gespeichert, bevor sie für die nächste Episode auf null gesetzt wird und die Liste self.experience als Exceltabelle unter GesamteErfahrung.csv gespeichert wird. Somit beinhaltet die Liste GesamteErfahrung.csv am Ende der Episoden, alle Trainingsdaten, die vom Raspi generiert wurden. Manchmal kommt es vor, dass Raspi nur eine Liste bestehend aus einer Zeile während einer Episode generiert, und sie daher nur eine Dimension hat. In diesem Fall kommt es zu einer Fehlermeldung, wenn man diese Liste an die zweidimensionale self.experience Liste anhängen will. Um diesen Fehler zu umgehen, müssen wir zuerst die Liste vom Raspi in eine zweidimensionale Liste umwandeln (siehe Zeile 24 bis 25 im Code 5). Am Ende der Methode geben wir die Liste self.lastExperience als Rückgabe aus.

Die nächste wichtigste Methode ist die train() Methode. In dieser Methode wiederum rufen wir u. a. die Methode loadExperienceList() auf und speichern die Rückgabe in einer Variablen namens listOfstates ab. Danach speichern wir die Zustände, die Variablen Done, Reward und die Aktion über die Methode remember() in Zeile 25, in der Liste self.memory ab. Code 6 zeigt einen Abschnitt aus der train() Methode.

```
1   def train(self, num_episodes: int):
2       last_rewards: Deque = collections.deque(maxlen=5)
3       best_reward_mean = 0.0
4       for episode in range(1, num_episodes + 1):
5           total_reward = 0.0
6           rewardWithoutBoni = 0.0
7           print('Episode=', episode)
8           listOfstates = self.loadExperienceList()
9           for I in range(0, np.shape(listOfstates)[0]):
10              state = listOfstates[I, 0:4]
```

```
11              state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
12              action = int(listOfstates[I, 4])
13              reward = (listOfstates[I, 5]).astype(np.float32)
14              next_state = listOfstates[I, 6:10]
15              next_state = np.reshape(next_state, newshape=(1, -
1)).astype(np.float32)
16              done = bool(int(listOfstates[I, 10]))
17              if abs(next_state[0][0]) > 175 / 180 and
abs(next_state[0][1]) < 6 / 40:
18                  self.zenitCounter += 1
19                  if self.FlagZenit == 0:
20                      self.IZenit = I
21                      self.FlagZenit = 1
22              rewardTheta = (-np.cos(next_state[0][0] * np.pi) +
1.0) / 2.0
23              rewardX = np.cos((next_state[0][2]) * (np.pi / 2.0))
24              rewardWithoutBoni += rewardTheta * rewardX
25              self.remember(state, action, reward, next_state,
done)
26              self.replay()
27              total_reward += reward
28              state = next_state
```

Code 6. Abschnitt aus der train() Methode

Eine Episode besteht aus maximal 1000 Zeitschritte, d.h. die Variable listOfstates in Zeile 8 von Code 6 kann maximal 1000 Zeilen besitzen. In Zeile 10 bis 16, entpacken wir die Werte von der Variable listOfstates in die Variablen state, next_state, action, reward, done. Die restlichen Codezeilen in der train() Methode im Anhang sind für die Erzeugung der Diagramme wie bei der OpenAI Gym-Simulation.

3.3 Pendel_Raspi_4Obs_class.py Klasse

Die Hauptfunktion von der Pendel_Raspi_4Obs_class.py Klasse (im Folgenden Pendel-Raspi-Klasse) ist, die Trainingsliste für den PC zur Verfügung zu stellen. In der OpenAI Gym-Simulation gab es zwei wichtigste Methoden, die uns die Zustände der Umgebung geliefert haben: die reset() und step() Methode: wir orientieren uns an OpenAI Gym [12] und implementieren unter anderem auch diese beiden Methoden in der PendelRaspi-Klasse.

Für die Steuerung des Prüfstandes müssen zuerst ein paar Bibliotheken und Methoden importiert werden. In diesem Abschnitt gehen wir nur auf die Methoden ein, die für uns wichtig sind, um die Trainingsdaten zu sammeln.

Bei der Konzipierung des realen Prüfstands wurde eine Methode Namens readall() implementiert. Diese Methode liefert uns den gleichen Zustand aus 4-Tupel wie bei der OpenAI Gym-Simulation in dieser Reihenfolge:

- Der Winkel: θ
- Die Winkelgeschwindigkeit: $\dot{\theta}$

- Die Position des Schlittens im Folgenden y genannt (in der Simulation x genannt)
- Die Geschwindigkeit des Schlittens: $\dot{y}$ ($\dot{x}$ in der Simulation)

In der Simulation war die Reihenfolge von dem 4-Tupel: x, $\dot{x}, \theta$ und $\dot{\theta}$. Eine weitere Besonderheit ist, dass in der Simulation der Zenit Punkt bei 0 Grad ist und im realen Pendel bei 180 Grad. Außerdem ist die Winkeleinheit in der Simulation Radiant und im realen Pendel in Grad. Abbildung 6 zeigt die Winkelangabe in der Simulation und im Versuch.

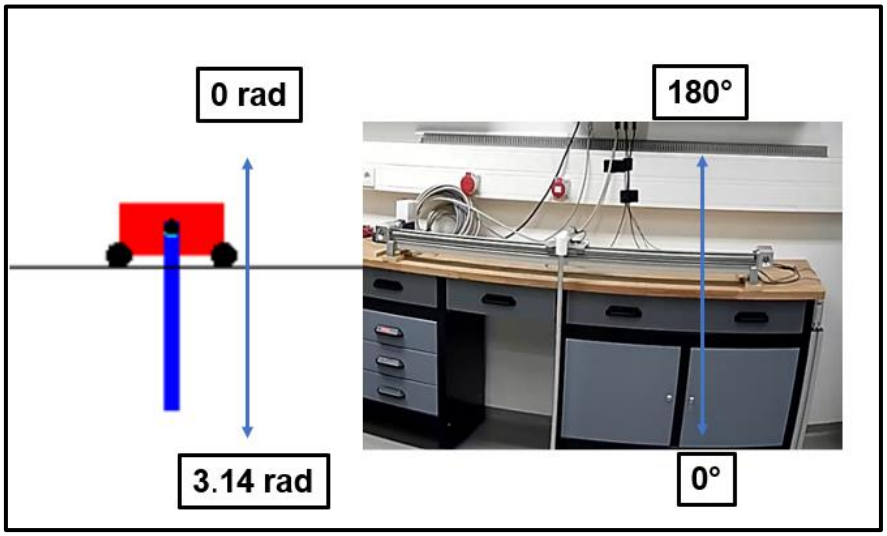

Abbildung 6: Winkelangabe in der Simulation (links) und im Versuch (rechts)

Im Konstruktor der PendelRaspi-Klasse legen wir ein paar Parameter fest und laden vor der ersten Episode das Modell aus Dropbox. Code 7 zeigt den Abschnitt aus dem Konstruktor, der für das Laden des Modells aus Dropbox zuständig ist. Dieser Code ist dem Code 2 ähnlich, außer, dass wir das Ganze um einen try-except-Block erweitert haben. Sonst ist der Konstruktor neben anderen Variablen ähnlich dem PendelPC-Konstruktor aufgebaut.

```
1   while Flag==0:
2           try:
3               db = dropbox.Dropbox('ACCESSTOKEN')
4               db.files_download_to_file(PathHome+'model_new.h5',
                PathCloud+'model_new.h5')
6               self.model_new.load_model(PathHome+'model_new.h5')
7               Flag=1
8           except requests.exceptions.ConnectionError:
9               print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
10          print('Es läuft jetzt eine neue Lesson...')
```

Code 7. Laden vom DQN-Modell aus Dopbox

Nach dem Konstruktor rufen wir die bereitgestellten Methoden von dem Pendelprüfstand auf. Diese sind nämlich:

- readall(): liefert uns die Zustände der Umgebung und entspricht der Lösung unserer Differenzialgleichung bei der Simulation.

- halt(): damit wird der Schlitten angehalten, d.h. der DC-Motor, der für die Steuerung des Schlittens zuständig ist, gibt keine Pulse mehr aus. Diese Funktion entspricht der Aktion Halt bei der Simulation.
- rechts(): der Schlitten fährt nach rechts (negative y-Werte), das ist die Aktion Rechts bei der Simulation.
- links(): der Schlitten fährt nach links (positive y-Werte), entspricht der linken Aktion bei der Simulation.
- Mitte(): Schlitten fährt in die Mitte $(y = 0)$, entspricht $x = 0$ in der Simulation.

Für das Training benötigen wir die Aktion als Integer-Werte. Deswegen wandeln wir in der Methode actionHandler() die Funktionen für die Schlittenbewegung in Integer um und legen gleichzeitig mit der Variable Pause = 0.1 eine Zeitspanne von 0.1 s für die Aktionsdauer an. Eine Pause zwischen den Aktionen ist wichtig, damit der Motor seine Wirkung realisieren kann (bei einer Pause von 0 hätte eine Aktion gar keine Wirkung). In der while-Schleife geben wir die Zustandswerte aus, damit wir die Abbruchbedingung prüfen können.

```
1   def actionHandler(self, action):
2       if action == 0:
3           self.rechts()
4           Pause = 0.1
5       elif action == 1:
6           self.halt()
7           Pause = 0.1
8       else:
9           self.links()
10          Pause = 0.1
11      Start = time.time()
12      dt = 0
13      while dt < Pause:
14          u, udot, y, ydot = self.read_all()
15          if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
16              self.mitte()
17              time.sleep(0.3)
18              break
19          dt = time.time() - Start
```

Code 8. actionhandler() Methode

Nach diesen Methoden legen wir die reset() und die step() Methoden an. Diese beiden Methoden haben die gleichen Funktionalitäten wie bei OpenAI Gym-Umgebung:

- In der reset() Methode lassen wir den Schlitten über eine Zeitspanne von 0.2 s nach rechts fahren und prüfen dabei die Abbruchbedingung. Danach legen wir eine Pause von 0.06 s ein, lassen den Schlitten nach links fahren, setzen eine Pause von 0.06 in die while-Schleife ein, damit wir möglichst kleine Winkel bekommen und rufen am Ende die Funktion readall() auf, um die Umgebungswerte in den Variablen $(u, \dot{u}, y, \dot{y})$ zu speichern. Die reset() Methode dient einfach dazu, das Pendel in

Schwingung mit kleinem Ausschlagswinkel zu bringen (Ausgangszustand jeder Episode).

- Die step() Methode bekommt eine Aktion als Eingabe und liefert uns die Zustandsvariablen der Umgebung. Die step() Methode ruft dabei die actionHandler() Methode auf, um anhand der Aktion die Schlittenbewegung zu bestimmen. Hier werden auch die Reward-Funktion und die Abbruchbedingung, wie bei der Simulation definiert. Die Abbruchbedingung ist so definiert, dass der Schlitten sich nur zwischen -390 mm und 390 mm bewegen darf und die absolute Winkelgeschwindigkeit 40 rad/s nicht überschreitet. Außerdem sorgen wir mit der Variable self.t_limit dafür, dass eine Episode maximal aus 1000 Zeitschritte besteht. Die Definition der Abbruchbedingung über die Variable done ist im Code 9 (Zeile 43 bis 48) zu sehen.

```
1    def reset(self):
2        self.mitte()
3        time.sleep(0.1)
4        Rechts = 0.2
5        Start1 = time.time()
6        dt1 = 0
7        while dt1 < Rechts:
8            self.rechts()
9            u, udot, y, ydot = self.read_all()
10           if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
11               self.mitte()
12               time.sleep(0.3)
13               break
14           dt1 = time.time() - Start1
15       time.sleep(0.06)
16       Links = 0.2
17       Start2 = time.time()
18       dt2 = 0
19       while dt2 < Links:
20           self.links()
21           time.sleep(0.06)
22           u, udot, y, ydot = self.read_all()
23           if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
24               self.mitte()
25               time.sleep(0.3)
26               break
27           dt2 = time.time() - Start2
28       u, udot, y, ydot = self.read_all()
29       self.state = (u, udot, y, ydot)
30       u_n, udot_n, y_n, ydot_n = copy(self.state)
31       obs = (u_n / self.u_max, udot_n / self.u_dot_threshold,
y_n / self.y_threshold, ydot_n / self.y_dot_max)
32       self.t = 0
33       return np.array(obs)
34
```

```
35  def step(self, action):
36      Boni = 0.0
37      self.actionHandler(action)
38      u, udot, y, ydot = self.read_all()
39      self.state = (u, udot, y, ydot)
40      u_n, udot_n, y_n, ydot_n = copy(self.state)
41      obs = (u_n / self.u_max, udot_n / self.u_dot_threshold,
y_n / self.y_threshold, ydot_n / self.y_dot_max)
42      self.t += 1
43      done = bool(
44          y < -self.y_threshold
45          or y > self.y_threshold
46          or abs(udot) > self.u_dot_threshold
47          or self.t >= self.t_limit
48      )
49      reward_theta = (-np.cos((u * np.pi) / 180) + 1.0) / 2.0
50      reward_x = np.cos((y / self.y_threshold) * (np.pi /
2.0))
51      reward = reward_theta * reward_x
52      return np.array(obs), reward, done
```

Code 9. reset() und step() Methoden

Die 4-Tupel der Umgebungsvariable werden wie bei der Simulation auf ihren jeweiligen maximalen Wert normalisiert, bevor sie von den Methoden reset() und step() als Rückgabewert ausgegeben werden.

Die Methode experienceList() ermöglicht uns die Zustandswerte in eine Liste zu speichern. Dafür wurde die Liste self.Erfahrungsvalue und self.LetzteErfahrung im Konstruktor angelegt. Beide Listen bestehen am Anfang wie die Liste self.lastExperience in der PendelPC-Klasse, aus einer Zeile und elf Spalten. Die Liste self.Erfahrungsvalue wird in jedem Zeitschritt mit den 4 Tupeln des Ausgangszustands, der Aktion, dem Reward, dem Folgezustandstupel und der Variable done gefüllt, und an die Liste self.LetzteErfahrung angehängt, sodass deren Länger in der Zeilen-Dimension während einer Episode anwächst. In dieser Methode reduzieren wir auch den Epsilon-Greedy Wert, jedes Mal, wenn diese Methode aufgerufen wird und solang $\varepsilon > 0.01$ ist.

```
1  def experienceList(self, state, action, reward, next_state,
done):
2      self.ErfahrungValue[0, 0:4] = state
3      self.ErfahrungValue[0, 4] = action
4      self.ErfahrungValue[0, 5] = reward
5      self.ErfahrungValue[0, 6:10] = next_state
6      self.ErfahrungValue[0, 10] = done
7      self.LetzteErfahrung = np.append(self.LetzteErfahrung,
self.ErfahrungValue, axis=0)
8      if self.epsilon > self.epsilon_min:
9          self.epsilon *= self.epsilon_decay
```

Code 10. experienceList() Methode

Die Methode saveExperienceListOnDropox() dient zum Hochladen der Trainingsdaten auf die Dropbox und zum Speichern der Textdatei „MeldungRaspi.txt“, damit der PC das Training fortsetzen kann (Punkt 3, 7, 11 auf der Abbildung 5). Die beiden Codeabschnitte zum Hochladen der Trainingsdaten und die Textdatei sind ähnlich wie das Beispiel Code 1 außer, dass wir hier wieder einen try-except-Block verwenden, um mit Fehlern bei der Datenübertragung umzugehen.

```
1   def saveExperienceListOnDropox(self):
2       self.LetzteErfahrung = self.LetzteErfahrung[1:, :]
3       # Save Erfahrung on Dropbox
4       np.savetxt(PathHome + 'LetzteErfahrung.csv',
self.LetzteErfahrung, fmt='%1.7f', delimiter=',')
5       try:
6           f = open(PathHome + 'LetzteErfahrung.csv', 'rb')
7           upname = PathCloud + 'LetzteErfahrung.csv'
8           db.files_upload(f.read(), upname,
9
mode=dropbox.files.WriteMode("overwrite")
10                          )
11          f.close()
12      except requests.exceptions.ConnectionError:
13          print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
14          # send file "MeldungRaspi.csv" to Dropbox
15      print('Meldung an den PC, dass die Episode beendet
wurde...')
16      try:
17          np.savetxt(PathHome + 'MeldungRaspi.csv', [1], fmt='%i')
18          print('Medlung an PC... ')
19          f = open(PathHome + 'MeldungRaspi.csv', 'rb')
20          upname = PathCloud + 'MeldungRaspi.csv'
21          db.files_upload(f.read(), upname,
22
mode=dropbox.files.WriteMode("overwrite")
23                          )
24          f.close()
25      except requests.exceptions.ConnectionError:
26          print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
```

Code 11. saveExperienceListOnDropox() Methode

Mit der Methode loadCurrentModel() aus Code 12, laden wir das aktuelle Modell von der Dropbox auf den Raspi (Nummer 2,6,10 auf der Abbildung 5). Dabei wird zuerst beim Aufruf dieser Methode, in einer while-Schleife ständig geprüft, ob die Datei „MeldungPC.csv“ vorhanden ist. Wenn die Datei MeldungPC.csv vorhanden ist, dann ist auch das neu trainierte Modell model_new.h5 vorhanden. Danach wird geprüft, ob die Datei MeldungPC.csv bereits gelöscht wurde. Wenn dies der Fall ist, lädt der Raspi das neue Modell und generiert damit die nächsten Trainingsdaten.

```
1   def loadCurrentModel(self):
2       print('Überprüfen ob der PC das Training des neuronalen
Netzes beendet hat...')
3       Flag = 0
4       while Flag == 0:
5           try:
6               List = db.files_list_folder(path="/RaspiPendel/")
7               for entrie in List.entries:
8                   if entrie.name == 'MeldungPC.csv':
9                       print('MeldungPC erhalten...')
10                      db.files_delete(PathCloud + 'MeldungPC.csv')
11                      Flag = 1
12          except requests.exceptions.ConnectionError:
13              print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
14
15      print()
16      # check if file "MeldungPC.csv" is already deleted
17      Flag1 = 0
18      Flag2 = 0
19      while Flag1 == 0:
20          try:
21              List = db.files_list_folder(path="/RaspiPendel/")
22              for entrie in List.entries:
23                  if entrie.name == 'MeldungPC.csv':
24                      print('MeldungPC  noch nicht gelöscht...')
25                      Flag2 = 0
26                  else:
27                      Flag2 = 1
28                      Flag1 = 1
29              if Flag2 == 1:
30                  print('MeldungPC  jetzt gelöscht...')
31          except requests.exceptions.ConnectionError:
32              print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
33      # load current model
34      print('Laden des neu trainierten neuronalen Netzes vom
PC...')
35      Flag3 = 0
36      while Flag3 == 0:
37          try:
38              db.files_download_to_file(PathHome + 'model_new.h5',
39                                        PathCloud +
'model_new.h5')
40              self.model_new.load_model(PathHome + 'model_new.h5')
41              Flag3 = 1
42          except requests.exceptions.ConnectionError:
43              print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
```

Code 12. loadCurrentModel() Methode

Die letzte wichtigste Methode ist collectExperience(). Hier spielen wir die Episoden durch und rufen die zuvor genannten Methoden auf, um die Trainingsdaten zu generieren. Diese Methode ist ähnlich wie die train() Methode auf dem PC außer, dass hier kein Training stattfindet. Die restlichen Methoden (siehe Pendel_Raspi_4Obs_class.py im Anhang), sind zur Generierung der Diagramme wie in der Simulation.

```
1       def collectExperience(self, num_episodes):
2           for E in range(1, num_episodes + 1):
3               total_reward = 0.0
4               print('Es bgeinnt jetzt eine neue Lesson...')
5
print('**********************************************************
**********************')
6               state = self.reset()
7               state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
8               for I in range(1, self.I_Max + 1):  # Iteration
9                   action = self.get_action(state)
10                  next_state, reward, done = self.step(action)
11                  next_state = np.reshape(next_state, newshape=(1,
-1)).astype(np.float32)
12                  self.experienceListe(state, action, reward,
next_state, done)
13                  total_reward += reward
14                  state = next_state
15                  if done:
16                      break
17              print('Episode:{} Reward:{}'.format(E,
total_reward))
18              self.mitte()
19              time.sleep(120)
20              self.saveExperienceListeOnDropox()
21              self.LetzteErfahrung = np.zeros((1, 11))
22              self.loadCurrentModel()
23              print('Neuronales Netz wird trainiert ...')
24          GPIO.cleanup()
```

Code 13. collectExperience() Methode

Nachdem wir uns mit der groben Struktur des Programmcodes vertraut gemacht haben, schauen wir uns in dem nächsten Abschnitt besondere Kenngrößen an, die für die Beschreibung des Schwingverhaltens des Inverted Pendulums wichtig sind.

4 Physikalische Kenngrößen

Das Ziel dieser Untersuchung ist vor allem einen Näherungswert für die Dämpfung der Schwingung im Versuch zu bekommen, um diese Erkenntnis in der Simulation verwenden zu können. Wir sind auch in der Lage, mit dieser Untersuchung Kenntnisse über den Einfluss der Masse vom Schlitten auf das Schwingen des Pendels zu gewinnen, sowie über die normale Schwingungsdauer, wenn wir eine bestimmte Aktion über einen längeren Zeitraum ohne Unterbrechung laufen lassen.

Tabelle 2 zeigt die Rahmenbedingungen des aktuellen Versuchsaufbaus. Es werden in Zukunft verschiedene Pendel mit unterschiedlichen Massen und Längen verwendet, um den Einfluss der Pendel-Eigenschaften auf das Lernverhalten des Agenten zu beurteilen. Ein paar Kenngrößen wie die Schwingungsdauer sind aber auch auf andere Pendel übertragbar, solange die gleiche Pendellänge wie im vorliegenden Versuch verwendet wird (~940 mm).

Physikalische Größe	Wert
Masse Schlitten	0.6 kg
Masse Pendel	0.5 kg
Verfahrweg Schlitten	-390 mm bis 390
Länge Pendel	940 mm
Maximale Schlitten-Geschwindigkeit	~100 mm/s
Maximale Winkelgeschwindigkeit	40 rad/s

Tabelle 2: Rahmenbedingungen Versuchsaufbau

4.1 Kenngrößen realer Versuch

Um die Kenngrößen zu erfassen, lassen wir den Winkel und den Verfahrweg über eine längere Zeitspanne aufnehmen und danach als Diagramm darstellen. Dafür benötigen wir einen kleinen Programmcode, um die Daten zu sammeln.

```
1   def saveDictionary(self,anglePositionDict):
2       with open('realpendel.csv', 'w') as f:
3           for key in anglePositionDict.keys():
4               f.write("%s,%s\n"%(key,anglePositionDict[key]))
5
6   def pendelParameter(self,num_episodes=1):
7       for E in range(1,num_episodes+1):
8           my_dict={}
```

```
9           state = env.reset()
10          Start1=time.time()
11          dt1=0
12          while dt1< 100:
13              self.halt()
14              u,udot,y,ydot = self.read_all()
15              my_dict[dt1] =[u,y]
16              dt1=time.time()-Start1
17          self.saveDictionary(my_dict)
18          self.mitte()
19          time.sleep(3)
```

Code 14. Programmcode zur Erfassung einer Kenngröße des realen Pendels

Im Code 14 legen wir zuerst eine Funktion an, die ein Python-Dictionary in eine csv-Datei speichert. In der Funktion pendelParameter(), lassen wir die reset() Funktion aufrufen, danach lassen wir in einer while-Schleife mit einer maximalen Durchlaufzeit von 100 Sekunden bei stehendem Schlitten (Aufruf der Funktion halt()), die Funktion readall() aufrufen und speichern den Winkel und die Position des Schlittens im Dictionary ab. Der Schlüssel-Wert für das Python-Dictionary ist die Zeitspanne zwischen den Aufnahmen.

4.1.1 Winkelaufnahme im realen Versuch

Wir lassen die aufgenommenen Winkel für einen kleinen Wertebereich über die Zeit darstellen und erhalten das Diagramm aus Abbildung 7.

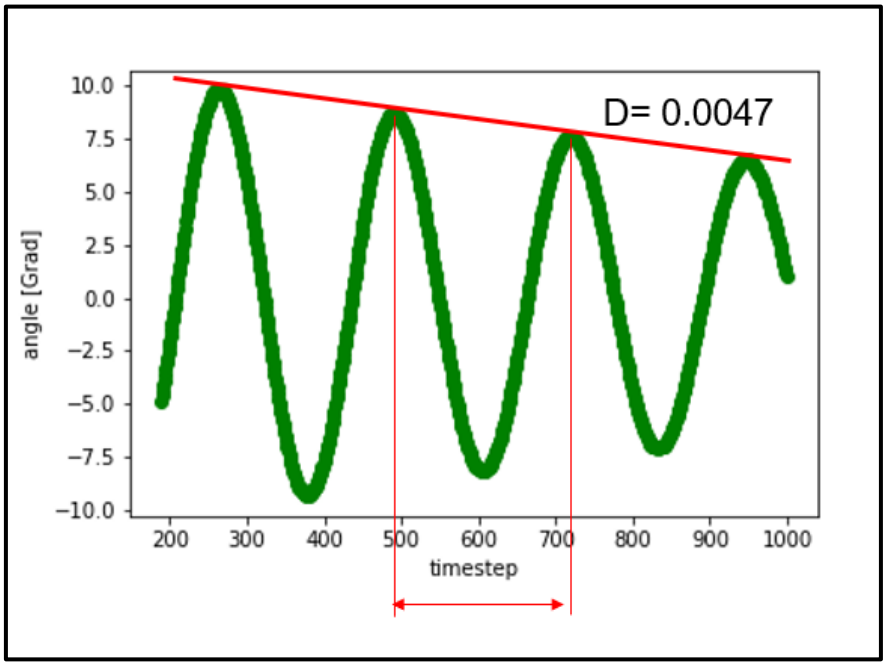

Abbildung 7: Realer Versuch Winkelaufnahme über Zeitschritte

Die Dämpfung D~0.0047 und die Schwingungsdauer bekommen wir, indem wir die positiven maximalen Amplituden-Werte (Y-Werte) als Funktion der Zeit (X-Werte) darstellen. Folgende Werte sind ermittelt worden:

$$X_{Werte} = [\,264, 490, 719, 947]$$

$$Y_{Werte} = [9.75, 861, 755, 6.5]$$

Die „normale“ Schwingungsdauer (~ 226 Zeitschritte) erhält man dann, wenn man zwei aufeinander folgende X-Werte subtrahiert (z.B $490 - 264 = 226$) und die Dämpfung (D=0.0047), indem wir die Steigung der Geraden ermitteln, die sich bei halblogarithmischer Auftragung der maximalen Amplituden über der Zeit ergibt. Da keine vollständige Abbklingkurve aufgenommen werden konnte, wurde die Steigung näherungsweise anhand einer nicht-logarithmischen Auftragung ermittelt (Abbildung 8).

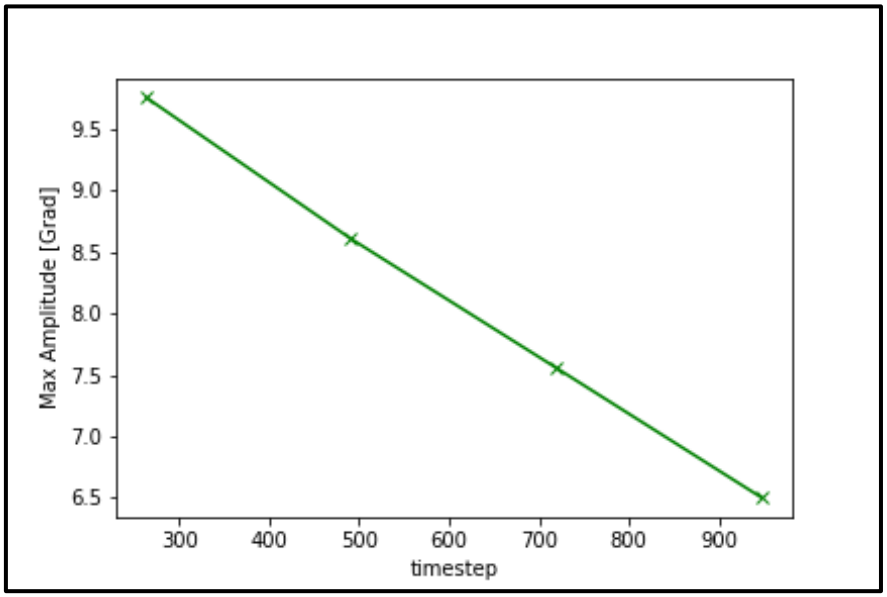

Abbildung 8: Maximale Amplitude über Zeitschritte des realen Pendels

4.1.2 Beeinflussung der Schlittenposition durch Pendelreaktionskräfte im realen Versuch

Wir lassen das Pendel wie bei der Winkelaufnahme frei schwingen und nehmen die Werte der Schlittenposition auf. Danach lassen wir das Diagramm der Schlittenverschiebung über die Zeit darstellen (Abbildung 9).

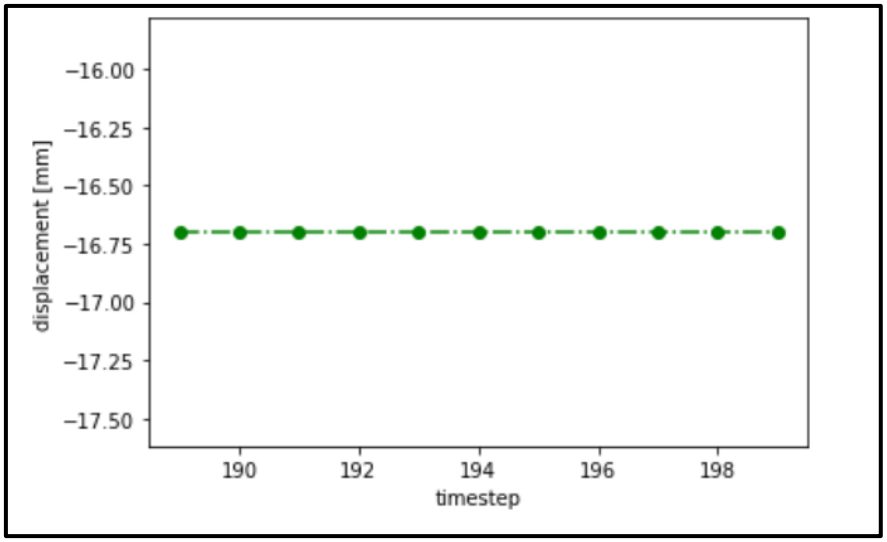

Abbildung 9: Schlittenposition über die Zeit im realen Versuch

Die Abbildung 9 zeigt, dass die Mitbewegung des Schlittens vernachlässigbar ist, wenn man das Pendel frei schwingen lässt.

4.2 Kenngröße in der Simulation

In der Simulation wurde eine Dämpfung von D= 0.0095 eingestellt, der maximale Verfahrweg des Schlittens lagt zwischen -2.4 mm und 2.4 mm. Da wir jetzt die Versuchsrahmenbedingungen kennen, macht es Sinn die Werte in der Simulation den realen Werten anzunähern.

- Der Dämpfungswert wird von 0.0095 auf 0.005 geändert (im Versuch wurde D= 0.0047 ermittelt). Mit diesem Wert sah die simulierte Pendelbewegung viel realistischer aus. Wie bereits in der Praxisprojektarbeit [2] erwähnt, ist keine Dämpfung in der ursprünglichen Cart-Pole-Swing-Up Umgebung von Freeman [3] implementiert. Mit dem neuen Dämpfungswert können wir uns an das reale Verhalten eines physikalischen Pendels annähern.
- Im Versuch ist der maximale Verfahrweg 390 mm und in der Simulation 2.4 mm. Es macht auch hier zwecks besserer Vergleichbarkeit Sinn, den Wert in der Simulation an den Versuchswert anzupassen. Allerdings müssen wir bei einem Wert von $x_{threshold} = 390$ die Größe der grafischen Darstellung der simulierten Pendelbewegung sehr stark skalieren, da man sonst kaum die Bewegung des Schlittens am Bildschirm sehen kann. Aus diesem Grund setzen wir $x_{threshold} = 40$ und passen folgende Variablen in der Methode render() im CartpoleSwingUpEnv.py an:

```
1   screen_width = 1600 # before was 600
2   screen_height = 600  # before was 400
3   scale = 18#before was screen_width / world_width
4   carty = screen_height / 2  # TOP OF CART
5   polelen = 120 * self.l  # bevor was scale* self.l
```

Code 15. Angepasste Variablen in render() Methode

Die Rahmenbedingungen in der Simulation sind in Tabelle 3 zusammengefasst. Bis auf den Verfahrweg, die Dämpfung und die render() Methode, nutzen wir die gleichen Einstellungen, die wir bei der Praxisprojektarbeit [2] benutzt haben.

Für die Darstellung des Winkels und der Schlittenbewegung, haben wir wie beim realen Versuch einen kleinen Programmcode geschrieben, um die Werte aufzunehmen (siehe Code 16).

Damit wir mehr Daten sammeln können, haben wir im CartpoleSwingUpEnv.py die Anzahl der Zeitschritte von 1000 auf 5000 erhöht. Nach der Aufnahme der Daten, setzen wir die Variable für die Einstellung der Zeitschritte auf ihren ursprünglichen Wert von 1000 ($self.t\_limit = 1000$) zurück. Außerdem lassen wir über die Methode step() für diese Simulation, nur den Winkel und die Position des Schlittens zurückgeben. Bei der reset() Methode, lassen wir das Pendel bei einem Anfangswinkel von $\sim 10\ Grad$ los und nehmen dann die Werte auf.

Physikalische Größe	Wert
Masse Schlitten	0.5 kg
Masse Pendel	0.5 kg
Verfahrweg Schlitten	-40 mm bis 40
Länge Pendel	600 mm
Maximale Schlitten-Geschwindigkeit	40 mm/s
Maximale Winkelgeschwindigkeit	40 rad/s

Tabelle 3: Rahmenbedingungen Simulation

```
1   import gym
2   from carpoleSwingUpEnv import CartPoleSwingUpEnv
3   env= CartPoleSwingUpEnv()
4   episodes = 1
5   def dictWriting(my_dict):
6       with open('aufgabeAC_simuladion_D005_time02.csv', 'w') as f:
7           for key in my_dict.keys():
8               f.write("%s,%s\n"%(key,my_dict[key]))
9   for episode in range(episodes):
10      my_dict={}
11      state = env.reset()
12      for i in range(5000) :
13          action=0
14          theta, x, _ = env.step(action)
15          my_dict[i] = [theta, x]
16          if done:
17             break
18      dictWriting(my_dict)
```

Code 16. Simulation Kenngröße Erfassung

In Abbildung 10 ist der Winkelverlauf über den Zeitschritt dargestellt. Bei der Darstellung haben wir nur Zeitschritte zwischen 1 und 500 berücksichtigt, um einen kleinen Grafikbereich zu haben. Wie beim Versuch, lassen wir hier auch die Dämpfung über die Steigung der maximalen Amplitude darstellen und bekommen mit einer Dämpfung von $D = 0.0048 \approx 0.005$, den gleichen Wert, den wir zuvor eingestellt haben. In der Abbildung 11 sieht man auch die Zeitschritte zwischen den maximalen Amplituden.

Wie im Versuch lassen wir auch die X_{Werte} und Y_{Werte} der maximalen Amplituden aufnehmen (siehe Abbildung 10 und 11).

$$X_{Werte} = [51, 151, 251, 352, 453]$$

$$Y_{Werte} = [9.7, 9.2, 8.7, 8.24, 7.8]$$

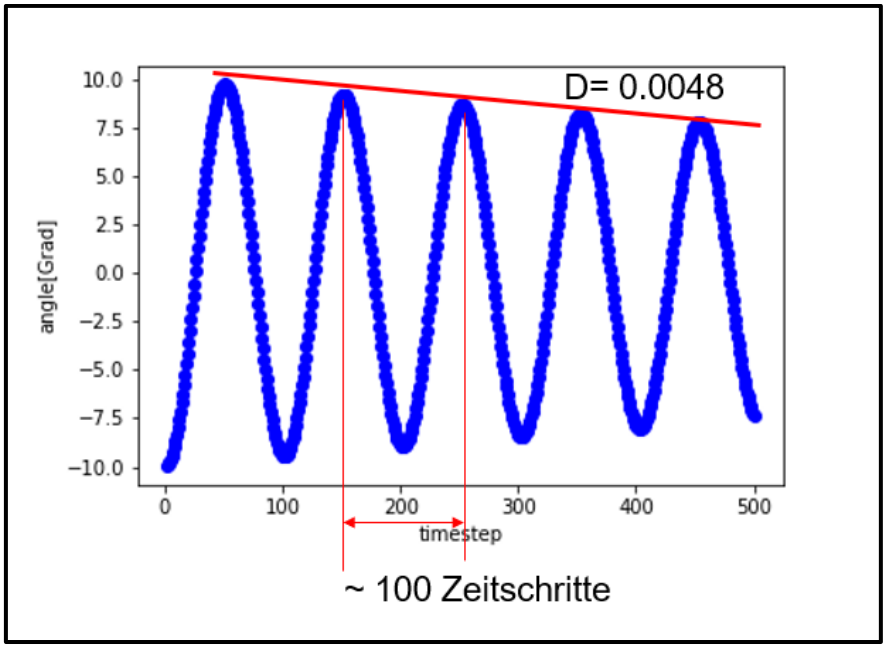

Abbildung 10: Simulation Winkelaufnahme über Zeitschritte

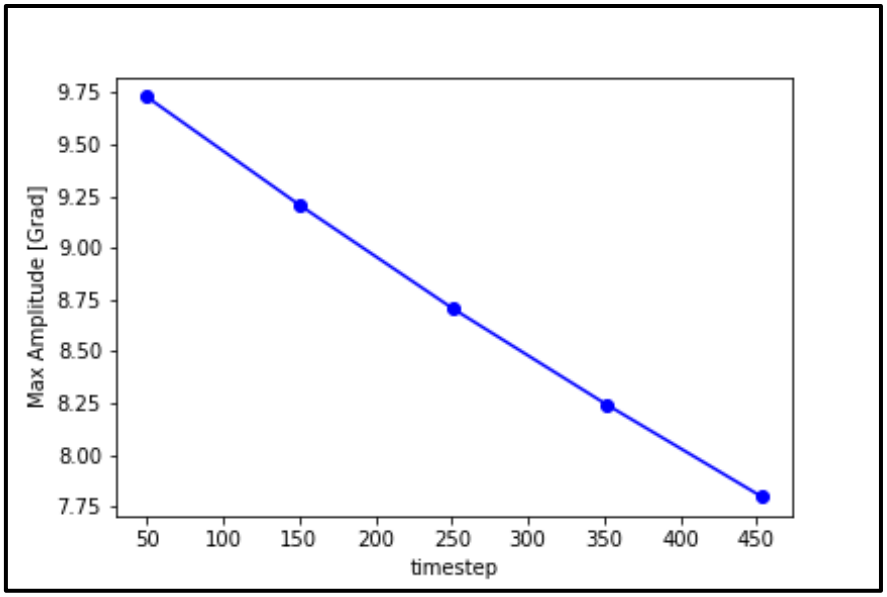

Abbildung 11: Schwingungsdauer in der Simulation

4.2.1 Beeinflussung der Schlittenposition durch Pendelreaktionskräfte in der Simulation

Wir lassen auch in der Simulation den Schlitten stehen (Aktion =0) und nehmen die Verschiebung aufgrund der Pendelschwingung auf. Abbildung 12 zeigt, dass der Schlitten Werte zwischen 0 und -0.1 annehmen kann. In der Simulation können wir demnach den Einfluss der Pendelreaktionskräften auf die Schlittenbewegung vernachlässigen.

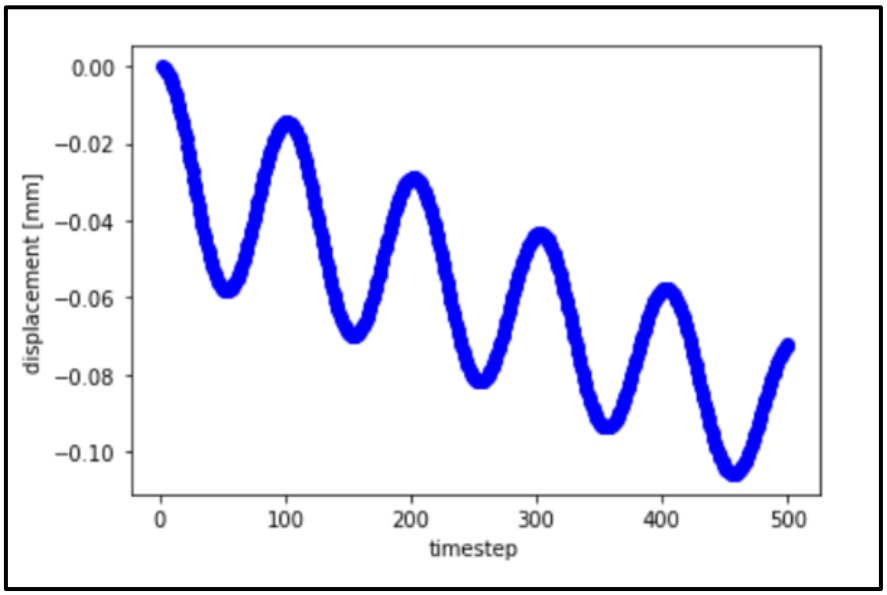

Abbildung 12: Schlittenposition über Zeitschritte in der Simulation.

Folgende Erkenntnisse können wir aus den ermittelten physikalischen Größen gewinnen:

- Eine „normale“ Schwingung dauert 2-mal länger im Versuch als in der Simulation, d. h., wenn wir in der Simulation 0.1s benötigen, um von einem Zustand in den nächsten überzugehen, benötigen wir für eine vergleichbare Zustandsänderung im Versuch 0.2 s. Es stehen beim Training des Agenten verschiedenen Aktionen zur Verfügung und eine Aktion dauert weniger als die eingestellten einhundert Sekunden in Code 14 Zeile 12, sodass die Schwingungsdauer beim Training anders als die ermittelte „normale“ Schwingungsdauer sein wird. Dies macht deutlich, dass die Simulation, zumindest in dieser Konfiguration nicht direkt mit dem realen Versuch vergleichbar ist.
- Es tritt sowohl in der Realität als auch in der Simulation keine Mitbewegung des Schlittens auf, wenn man das Pendel frei schwingen lässt: der Schlitten hat somit keinen großen Einfluss auf den Dämpfungswert des Pendels.

Aufgrund der geänderten Simulationseinstellungen in Tabelle 3 im Vergleich mit denen aus der Projektarbeit [2], (anderer Dämpfungswert und anderer Verfahrweg), macht es Sinn, den RL-Algorithmus in der Simulation mit den neuen Einstellungen zu wiederholen.

5 Basis-Ergebnisse

5.1 Ergebnisse Simulation

Verschiedenen Einstellungen wurden ausprobiert, um das beste Ergebnis zu bekommen. Hier wird nur das Ergebnis mit der besten Einstellung präsentiert. Weiteren Ergebnisse befinden sich im Anhang unter dem Abschnitt G.

Zusätzlich zu den Werten der Tabelle 3, wurden folgende Einstellungen vorgenommen, um das beste Ergebnis aus der Simulation zu bekommen:

- Die Aktivierungsfunktion ist jetzt $tanh$ und nicht $relu$ wie es in der Praxisprojektarbeit [2] war.
- In der Klasse dqnAgent wurde $batch\_size = 64$ (in PPB war $batch\_Size = 32$), $\varepsilon_{initial} = 0.2$ (in PPB war $\varepsilon_{initial} = 1$) eingestellt.
- Der Dämpfungswert wurde von 0.0095 auf 0.005 geändert.
- Für das Training haben wir 400 Episoden eingestellt.

In der Simulation hat der Agent allein durch die Änderung von $x_{theshold}$ von 2.4 mm auf 40 mm, mit vielen Einstellbereichen zu tun, was das Lernen komplizierter macht. Aus diesem Grund werden wir ab jetzt die Simulation mindestens zweimal wiederholen und das beste Ergebnis wird immer gezeigt.

Die Diagramme für die Ergebnisse, die wir vom Praxisprojekt kennen, werden dargestellt.

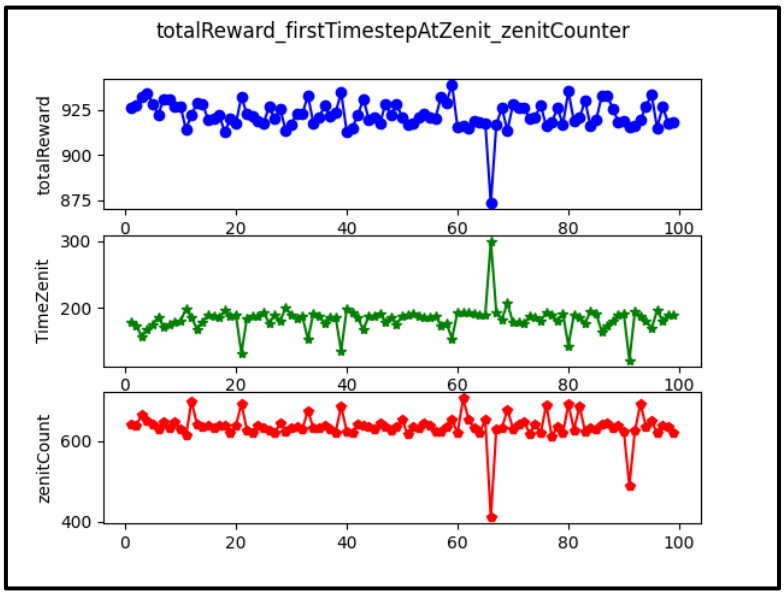

Abbildung 13: Ergebnis Basis-Modell Simulation

Die Tabelle 4 zeigt die Evaluierung von 100 Episoden nach dem Training.

Der zenitCounter-Wert ist schlechter im Vergleich zu den Praxisprojekt-Ergebnissen (siehe Praxisprojektbericht, Seite 38). Das ist aber das beste Ergebnis, das wir mit minimaler Änderung an der ursprünglichen Netzarchitektur und aus zwei Simulationen erzielt haben.

Wir nehmen die gleiche Einstellung für die Simulation und starten den ersten Versuch.

Ergebnis für 100 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	921	8	1000	0
zenitCounter	635	33	keine	keine
1stTimesAtZenit	183	18	keine	Keine

Tabelle 4: Standardabweichungen und Mittelwerte Basis-Modell Simulation

5.2 Ergebnisse realer Versuch

Für den realen Versuch wählen wir die gleichen Einstellungen für die Netzarchitektur und für die Agent-Klasse wie bei der Simulation. Wie in der Simulation lassen wir den Versuch mindestens zweimal wiederholen und das Ergebnis mit dem besten Reward-Wert wird gezeigt.

Die eingestellte reset() und die step() Methoden sind die gleichen wie im Code 9.

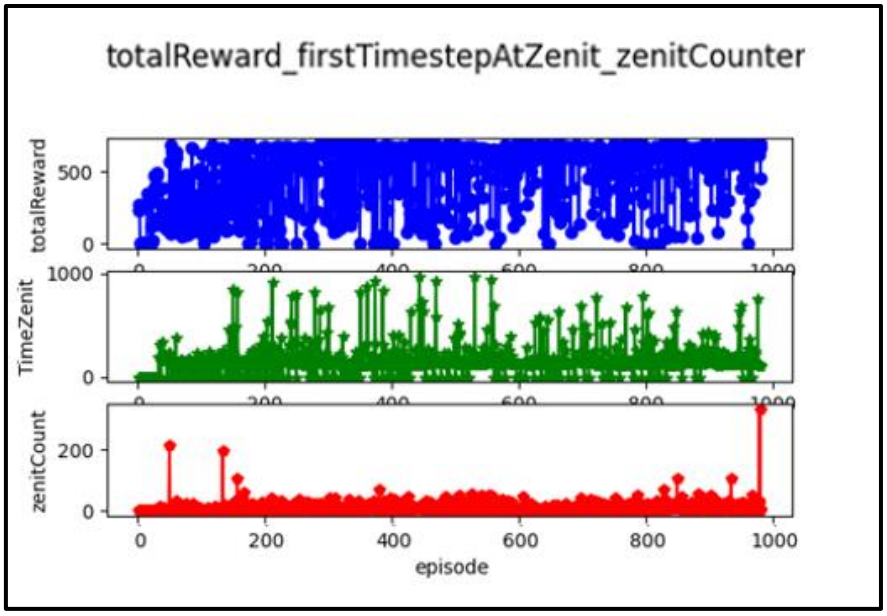

Abbildung 14: Trainingsverlauf Basis-Modell realer Versuch

Die Ergebnisse sind in Tabelle 5 zusammengefasst. In der Simulation lag der durchschnittliche Mittelwert mit der gleichen Einstellung bei 905 für den Reward, 679 für zenitCounter, im Versuch haben wir dagegen 604 und 3 für die jeweils gleichen Parameter. Die Ergebnisse der Simulation spiegeln das Versuchsergebnis nicht wider, das müssen wir in den folgenden Abschnitten genauer untersuchen.

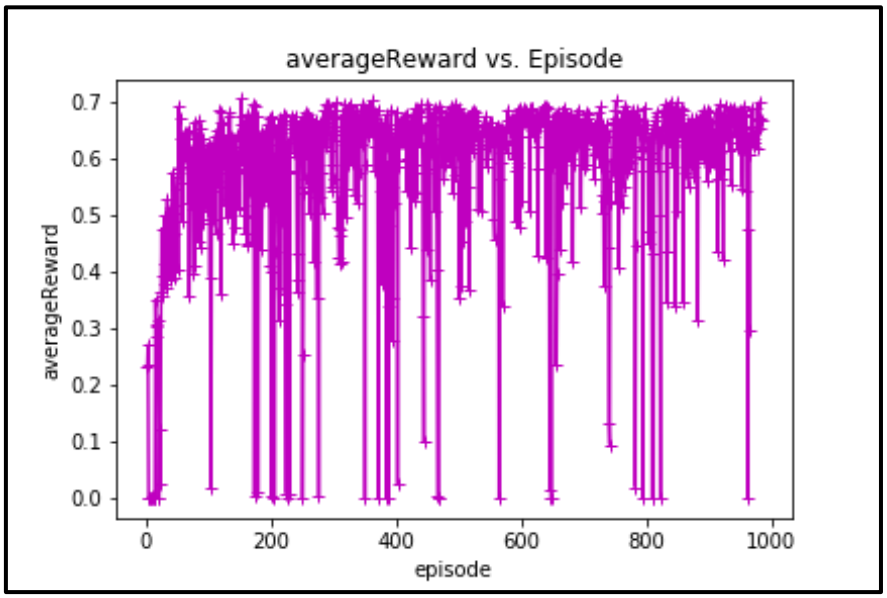

Abbildung 15: Average-Reward beim Training im Versuch

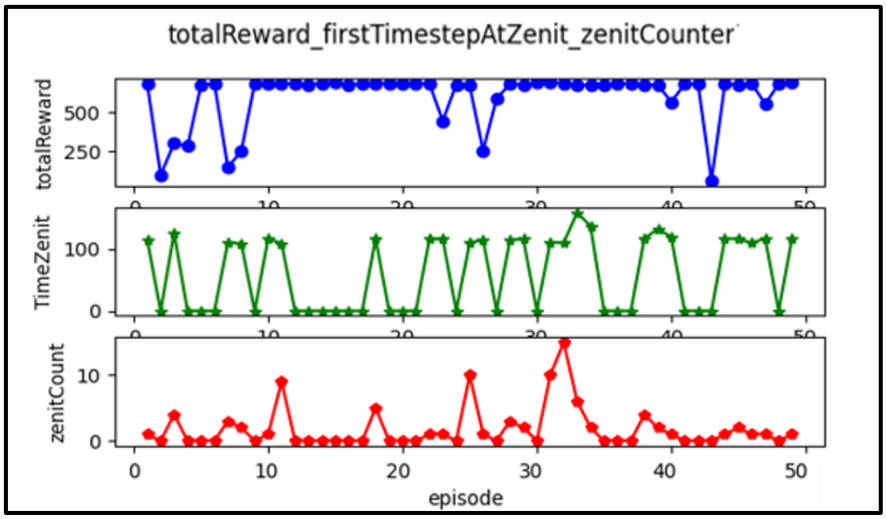

Abbildung 16: Auswertung Versuch Basis-Ergebnis

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	604	175	1000	0
zenitCounter	3	2	keine	keine
1stTimesAtZenit	60	59	keine	Keine

Tabelle 5: Standardabweichungen und Mittelwerte Basis-Modell Versuch

6 Analyse des Signaldurchlaufs

Mit der Schlittenbewegung kann das Pendel verschiedene Winkelpositionen annehmen. Dies gilt sowohl im Versuch als auch in der Simulation. Es ist dann wichtig, die beiden Kenngrößen, die die Schlittenbewegung vollständig beschreiben, nämlich die Position (x oder y) und die Geschwindigkeit des Schlittens ($\dot{x}$, oder $\dot{y}$) in der Simulation als auch im Versuch zu analysieren. Wir lassen im Folgenden die Geschwindigkeit und die Position des Schlittens über verschiedenen Zeitschritte aufnehmen und stellen dies graphisch dar.

6.1 Schlittenbewegung in der Simulation

Wir entwerfen hierfür ein kleines Programm, um x und $\dot{x}$ Werte in einer Liste aufzunehmen (siehe Code 17).

```
1   import gym
2   import numpy as np
3   from carpoleSwingUpEnv import CartPoleSwingUpEnv
4   env= CartPoleSwingUpEnv()
5   episodes = 1
6   def dictWriting(my_dict):
7       with open('avs_simulation.csv', 'w') as f:
8           for key in my_dict.keys():
9               f.write("%s,%s\n"%(key,my_dict[key]))
10  for episode in range(episodes):
11      my_dict={}
12      state = env.reset()
13      for i in range(5000) :
14          action=1
15          x, x_dot, reward, done, _ = env.step(action)
16          my_dict[i] = [x, x_dot]
17          if done:
18             break
19      dictWriting(my_dict
```

Code 17. Aufnahme der Schlittenbewegung in der Simulation

In Code 17 definieren wir nach dem Import eine Funktion mit dem Namen dictWriting(), die eine Liste, bestehend aus Wert und Schlüssel Paaren als Argument erhält und daraus eine Exceltabelle bereitstellt. Danach geben wir im CartpoleSwingUpEnv() dem Schlitten in einer for-Schleife 5000 mal hintereinander den Befehl sich nach rechts zu bewegen (action=1) und nehmen dabei die Position (x) und die Geschwindigkeit ($\dot{x}$) in die Liste auf.

Abbildung 17 zeigt die Position und die Geschwindigkeit in Abhängigkeit vom Zeitschritt.

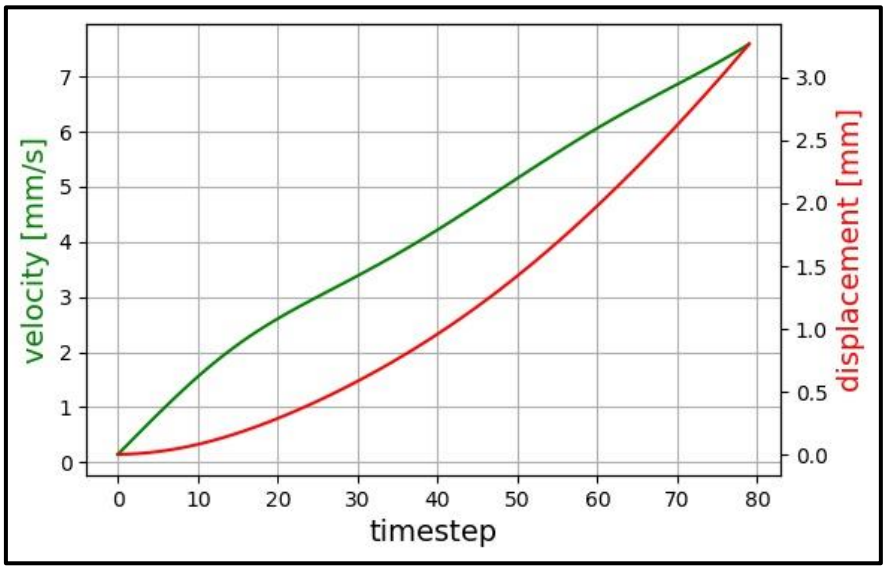

Abbildung 17: Schlittenposition und Geschwindigkeit in der Simulation.

Da es aufgrund des in der for-Schleife ständig wiederholten Befehls nach rechts zu fahren (durch Aufbringen einer Kraft F auf den Schlitten) zu keiner Verzögerung zwischen den Aktionen kommt, zeigt sich in Abbildung 17 ein kontinuierlicher Anstieg der Geschwindigkeit sowie der Position des Schlittens.

6.2 Schlittenbewegung im Versuch

Ähnlich wie in der Simulation haben wir hier ein kleines Programm geschrieben, um die Position des Schlittens (y) und die Geschwindigkeit ($\dot{y}$) in einer Liste aufzunehmen (siehe Code 18).

Der Code 18 ist dem Code 17 ähnlich mit dem einzigen Unterschied, dass der Schlüsselwert die Zeit für die Ausführung der aktuellen Aktion ist und eine separate Funktion namens getAVS() für die Aufnahme der Daten bereitgestellt wird.

```
1   def dictWriting(self,my_dict):
2       with open('avs_versuch.csv', 'w') as f:
3           for key in my_dict.keys():
4               f.write("%s,%s\n"%(key,my_dict[key]))
5   def getAVS(self,num_episodes):
6       for E in range(1,num_episodes+1):
7           my_dict={}
8           Start1=time.time()
9           dt1=0
10          while dt1< 6:
11              self.rechts()
12              u,udot,y,ydot = self.read_all()
13              if (abs(y)>self.y_threshold or
abs(udot)>self.u_dot_threshold):
14                  self.mitte()
15                  time.sleep(0.3)
16                  break
17              my_dict[dt1] =[u,y]
```

```
18              dt1=time.time()-Start1
19          self.dictWriting(my_dict)
20          self.mitte()
21          time.sleep(0.3)
```

Code 18. Aufnahme der Schlittenbewegung im Versuch

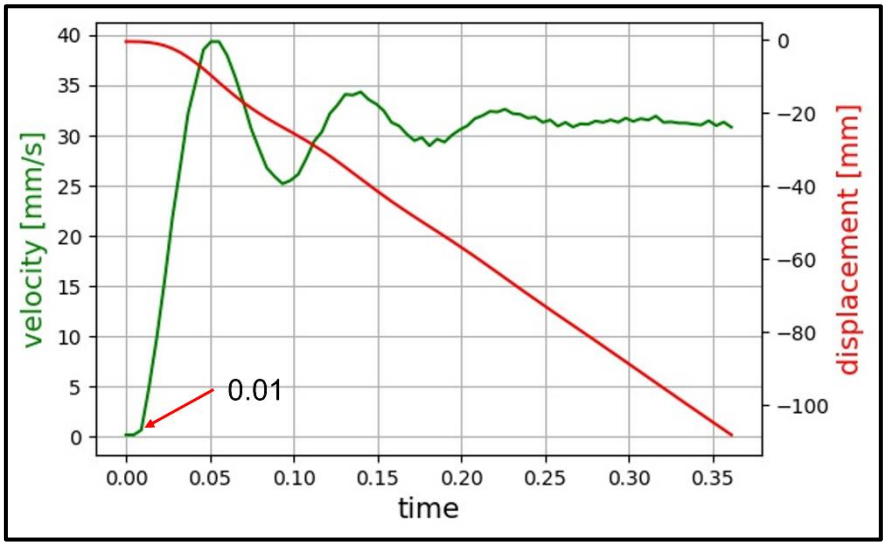

Abbildung 18: Schlittenposition und Geschwindigkeit im Versuch

In Abbildung 18 kann man erkennen, dass für die Ausführung einer Aktion mindestens eine Zeit $t = 0.01\ s$ benötigt wird. Es gibt eine Verzögerungszeit bei der Ausführung einer Aktion im Versuch. Außerdem haben wir bei reset() und step() Methode eine Aktion über eine Zeitspanne t (z.B. t= 0.1s in step()) durchführen lassen.

Wir halten Folgendes fest:

- In der Simulation folgt auf das Ende einer Aktion unmittelbar eine neue Aktion, sofern die Episode nicht beendet ist.
- Im realen Versuch gibt es eine Verzögerung (im folgenden Delay) wegen der Signalübertragung zwischen zwei aufeinander folgenden Aktionen.
- Neben der Verzögerung am realen Pendel wird eine Aktion in der step() Methode mindestens über eine Zeitspanne T (z. B. T=0.1 s) durchgeführt bevor die nächste Aktion kommt: diese Vorgehensweise wird als Sample and Hold bezeichnet.

Im nächsten Abschnitt implementieren wir das Sample and Hold (im folgenden SH) und die Verzögerung (im folgenden Delay) in der Simulation. Danach übertragen wir die beste Einstellung aus der Simulation auf den Versuch.

7 Sample Hold und Delay in der Simulation.

Für die Simulation mit den Einstellungen Sample Hold (im folgenden SH) und Delay müssen wir die train() Methode in der dqnAgent.py Klasse anpassen.

Code 19 zeigt die geänderten Codeabschnitte in der train() Methode.

```
1   SAMPLE_HOLD = 4 # als Beispiel
2   DELAY= 2        # als Beispiel
3   def train(self, num_episodes: int):
4       last_rewards: Deque = collections.deque(maxlen=5)
5       best_reward_mean = 0.0
6       for episode in range(1, num_episodes + 1):
7           total_reward = 0.0
8           timestepAtzenit=0
9           state = self.env.reset()
10          state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
11          initialAction = self.get_action(state)
12          # introduce delay with FIFO list actionList:
13          actionList = [initialAction for i in range(DELAY)]
14          while True:
15              timestepAtzenit+=1
16              action= actionList.pop(0)
17              next_state, reward, done, _ = self.env.step(action)
18              next_state = np.reshape(next_state, newshape=(1, -
1)).astype(np.float32)
19              if (len(actionList)<DELAY):
20                  self.remember(state, action, reward, next_state,
done)
21                  n_act  = self.get_action(next_state)
22                  # introduce sample & hold with for loop:
23                  [actionList.append(n_act) for i in
range(SAMPLE_HOLD)]
```

Code 19. Berücksichtigung von Sample Hold und Delay in der train() Methode

Zuerst legen wir zwei globale Variablen fest:

- Die Variable SAMPLE_HOLD legt fest, über wie viele Iterationen eine Aktion ausgeführt werden muss, bevor eine neue Aktion folgt.
- Mit der Variable Delay, legen wir fest, nach wie vielen Iterationen die neue Aktion eingreift.

Danach erstellen wir eine Liste von Aktionen (actionList) der Länge gleich Delay (siehe Zeile 13), in dieser Liste wird die Aktion Delay-mal gespeichert. In der Zeile 23 füllen wir die Liste actionList mit neuen Aktionen, sobald die actionList-Länge kleiner als Delay ist.

Für ein besseres Verständnis nehmen wir zum Beispiel Delay (oder transmit time) = 2 und SAMPLE_HOLD = 5 und betrachten wir Abbildung 19.

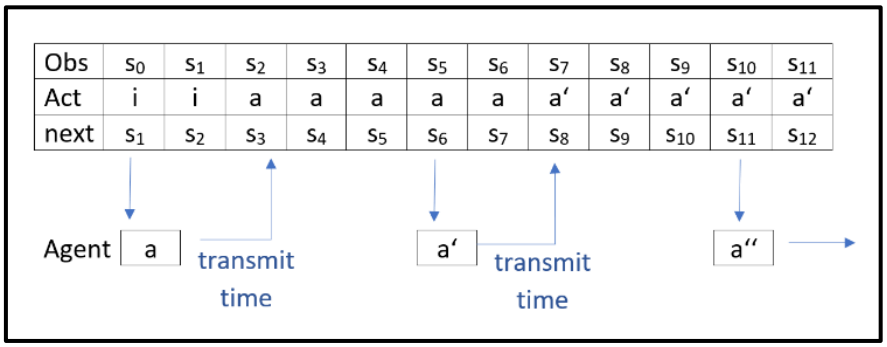

Abbildung 19: Beispiel SAMPLE_HOLD=5 und DELAY=2

In Abbildung 19 führt der Agent die Aktion i im Zustand S_0 aus und diese Aktion wird Delay-mal an die Liste actionList angehängt.

Für die Bestimmung des nächsten Zustands wird die erste Aktion aus der actionList genommen und gleichzeitig aus der Liste gelöscht. Der Agent ruft die Funktion env.step($action = i$) auf und geht in den nächsten Zustand S_1 über. Jetzt ist aber die Länge der actionsList gleich 1 und somit kleiner als DELAY=2: es wird dann basierend auf dem neuen Zustand S_1 eine neue Aktion a durch Aufruf der Funktion get_action($state = S_1$) bestimmt und diese Aktion a wird SAMPLE_HOLD-mal an die Liste angehängt. Damit kommt die Aktion a ab Zustand S_2 zur Ausführung. Beim Wechseln vom Zustand S_5 in den Zustand S_6 sinkt die Länge der actionsList wieder unter DELAY. Der Agent bestimmt die neue Aktion a', hängt diese SAMPLE_HOLD-mal an die ActionList an usw.

Um den Einfluss der Parameter SAMPLE_HOLD und DELAY auf die Ergebnisse zu untersuchen, starten wir folgende Simulationen:

- Simulation mit DELAY als diskrete Variable und SAMPLE_HOLD= 1 (ohne Sample_Hold).
- Simulation mit SAMPLE_HOLD als diskrete Variable und DELAY =1 (ohne Delay).
- Simulation mit SAMPLE_HOLD und DELAY beide als diskrete Variable und gleichgesetzt.

7.1 Simulation mit Delay als Variable

In dieser Simulation setzen wir im Code 19 SAMPLE_HOLD = 1 und starten verschiedene Simulationen, in denen die Variable Delay Werte von 1 bis 8 annehmen kann.

Für die Auswertungen der Varianten erstellen wir eine Kopie der CartPoleSwingUpEnv.py Klasse mit dem Namen CarpoleSwingUpEnvFixState.py und nehmen folgende Anpassungen an der neuen Umgebungsklasse vor:

- Wir erstellen zuerst aus der ursprünglichen Klasse CartPoleSwingUpEnv.py anhand der reset() Methode eine Liste mit einhundert Zuständen bestehend aus vier Tupel $(x, \dot{x}, \theta, \dot{\theta})$. Diese Liste wird im Konstruktor der Klasse CartpoleSwingUpEnvFixState aufgerufen.

- Beim ersten Aufruf der Mothode reset() in der CartpoleSwingUpEnvFixState-Klasse, wird der erste Eintrag der Liste ausgegeben und gleichzeitig aus der Liste entfernt. Bei dem zweiten Aufruf wird ebenso der zweite Eintrag entnommen und entfernt und so weiter, bis die 100 Episoden vorbei sind.

Mit der CartpoleSwingUpEnvFixState-Klasse, wird gewährleistet, dass bei der Auswertung mit verschiedenen Varianten die Anfangszustände gleich sind (dient zum besseren Vergleich).

Code 20 zeigt einen Abschnitt aus der CartpoleSwingUpEnvFixState Klasse. Außer der Anpassung im Konstruktor und der neuen reset() Methode ist die Klasse gleich der CartpoleSwingUpEnv.py-Klasse.

```
1   FileForEvaluation="fixStates.csv"
2   logger = logging.getLogger(__name__)
3   class CartPoleSwingUpEnvFixState(gym.Env):
4           ..........
5           ..........
6           ..........#weitere Codezeilen
7           self.stateListe=[]
8           self.stateCollecter= np.zeros(4)
9       with open(FileForEvaluation, newline='', encoding='utf-8')
as f:
10          reader = csv.reader(f)
11          for row in reader:
12              self.stateCollecter[0]=float(row[0])
13              self.stateCollecter[1]=float(row[1])
14              self.stateCollecter[2]=float(row[2])
15              self.stateCollecter[3]=float(row[3])
16              self.stateListe.append(self.stateCollecter)
17              self.stateCollecter= np.zeros(4)
18      def reset(self):
19          self.state = self.stateListe.pop(0)
20          x_n, x_dot_n, theta_n, theta_dot_n = copy(self.state)
21          obs = (x_n / self.x_threshold, x_dot_n /
self.theta_dot_threshold, theta_n / np.pi, theta_dot_n /
self.theta_dot_threshold)
22          self.steps_beyond_done = None
23          self.t = 0  # timestep
24          return np.array(obs)
```

Code 20. Codeabschnitt aus der CartPoleSwingUpEnvFixState Klasse

Mit den jeweiligen festgelegten Werten für Sample Hold und Delay, wird jede Simulation dreimal wiederholt und die Endergebnisse gemittelt. Dies ist wichtig, da die Modelle viel komplexer sind, sodass wir uns nicht auf ein einziges Testergebnis verlassen können. Dabei wird der Agenten bei der ersten und zweiten Simulation mit 600 Episoden trainiert und mit 1000 Episoden bei der dritten Simulation.

Nach dem Training werden die Agenten nach 100 Episoden evaluiert und der Mittelwert des gesamten Rewards wird als Endergebnis für die entsprechende Variante genommen (siehe Tabelle 6).

Die Ergebnisse der Simulation sind in Tabelle 6. Abbildung 20 zeigt den Verlauf des gesamten Rewards in Abhängigkeit des Delay-Wertes.

Es lässt sich erkennen, dass für Delay= 8 der Reward für den untersuchten Delay-Wertebereich minimal ist.

Variante	total_reward			Mittelwert
	Agent 1 (600 Episode)	Agent 2 (600 Episode)	Agent 3 (1000 Episode)	
SH1_D1	905	904	847	885
SH1_D2	844	866	822	844
SH1_D3	574	517	893	661
SH1_D4	220	849	712	594
SH1_D5	724	488	917	710
SH1_D6	740	395	878	671
SH1_D7	436	685	522	548
SH1_D8	348	407	428	394

Tabelle 6: Reward-Ergebnis mit Delay als Variable und ohne Sample_Hold

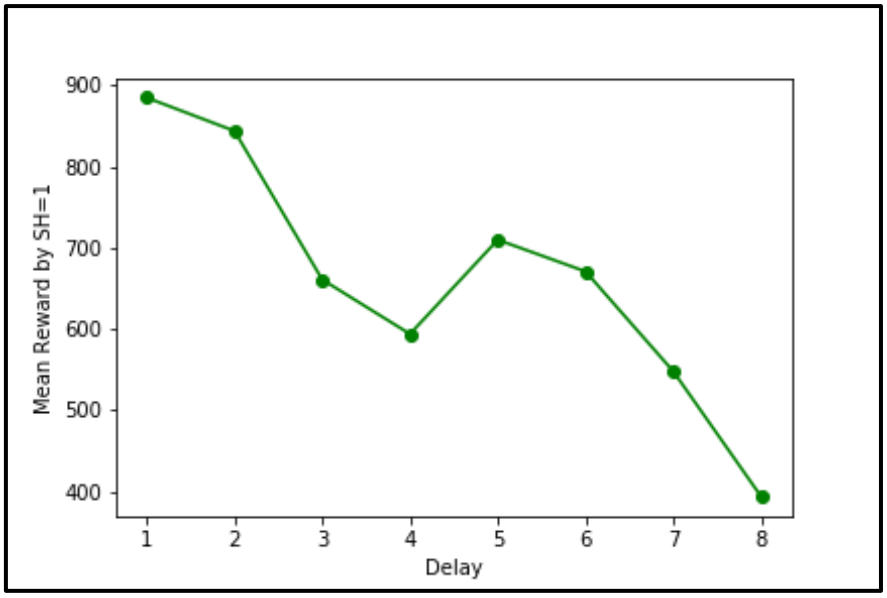

Abbildung 20: Mean-Reward in Abhängigkeit von Delay

7.2 Simulation mit SAMPLE HOLD als Variable

Wir wiederholen die Simulation und variieren diesmal die globale Variable SAMPLE_HOLD (im folgenden SH) und stellen Delay=1 ein (ohne Berücksichtigung von Delay). Wieder visualisieren wir die Ergebnisse als Diagramm und als Tabelle.

Variante	total_reward			Mittelwert
	Agent 1 (600 Episode)	Agent 2 (600 Episode)	Agent 3 (1000 Episode)	
SH1_D1	905	904	847	885
SH2_D1	881	832	839	851
SH3_D1	814	819	526	720
SH4_D1	680	801	342	608
SH5_D1	341	632	788	587
SH6_D1	607	606	447	553
SH7_D1	346	739	471	519
SH8_D1	474	539	591	535

Tabelle 7: Reward-Ergebnis mit SH Variable und ohne Delay

Das Ergebnis bei SH = 7 stellt sich als worst-case dar, wenn wir Delay nicht berücksichtigen und die Variable SH Werte von 1 bis 8 annehmen lassen.

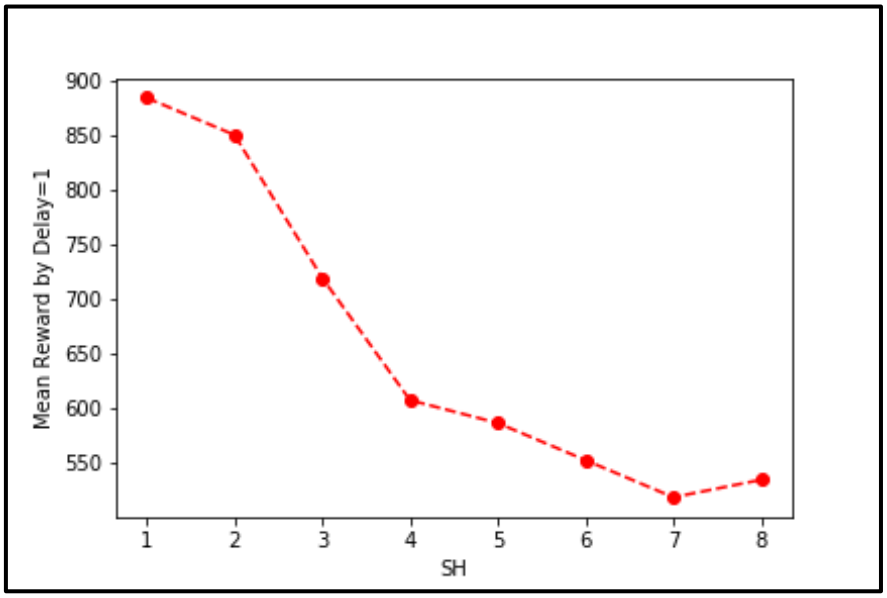

Abbildung 21: Mean-Reward in Abhängigkeit von SH

7.3 Simulation mit SAMPLE HOLD und Delay als Variable

Die Variablen SH und Delay werden gleichgesetzt und wir lassen wieder den Mean-Reward Wert in Abhängigkeit der beiden Variablen darstellen.

Variante	total_reward			Mittelwert
	Agent 1 (600 Episode)	Agent 2 (600 Episode)	Agent 3 (1000 Episode)	
SH1_D1	905	904	847	885
SH2_D2	589	893	786	756
SH3_D3	841	506	748	698
SH4_D4	651	633	625	636

Tabelle 8: Reward-Ergebnis mit SH und Delay Variable

Bei SH = Delay = 4 ist der Mean-Reward minimal für die untersuchten Wertebereiche (siehe Abbildung 22).

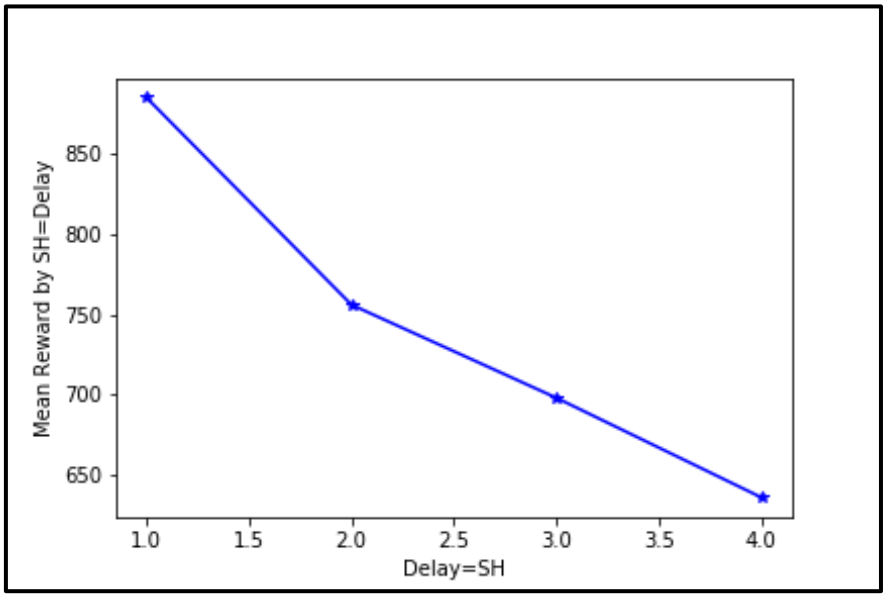

Abbildung 22: Mean-Reward in Abhängigkeit von SH und Delay

Die Ergebnisse sind viel schlechter als das Simulationsergebnis aus Abschnitt 5.1 und die Verläufe des Mean-Rewards über die Variablen ist bei einer bestimmten Variablen Größe nicht mehr eindeutig. Mit zunehmender variabler Größe ist das Lernverhalten viel komplexer und es kommen zu viele Streuungseffekten in den Ergebnissen vor.

Bei den Ergebnissen aus Abschnitt 7.1 stellt sich beim Delay=8 der worst-case ein. Wir müssen aber die Variable SH auch berücksichtigen, wenn wir die Versuchsergebnisse verbessern wollen, da die beiden Variablen, wie wir im Abschnitt 6 gesehen haben, Einfluss auf das Versuchsergebnis haben. Aus diesem Grund nehmen wir die Variante SH=4 und Delay=4 unter die Lupe und versuchen die Ergebnisse zu verbessern (SH=1 und Delay=1 stellen zwar den best case in allen drei Simulationen dar, sind aber unrealistisch, da Sie im realen Versuch größer eins waren). Außerdem liegt der Reward-Wert der Variante SH=4 und Delay=4 bei 636 und passt ungefähr zu dem Basis-Ergebnis aus dem Versuch, wo wir einen Wert von 604 hatten. Natürlich man kann auch eine andere Variante für die weitere

Untersuchung nehmen. Man muss nur darauf achten, dass SH- und Delay-Wert größer eins sind.

Ergebnis für 100 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	651	133	1000	0
zenitCounter	77	32	keine	keine
1stTimesAtZenit	225	95	keine	Keine

Tabelle 9: Standardabweichungen und Mittelwerte Variante SH=Delay=4

Abbildung 23 zeigt die Auswertung von einer der drei Varianten mit SH=4 und Delay=4, die Standardabweichung und der Mittelwert stehen in der Tabelle 9.

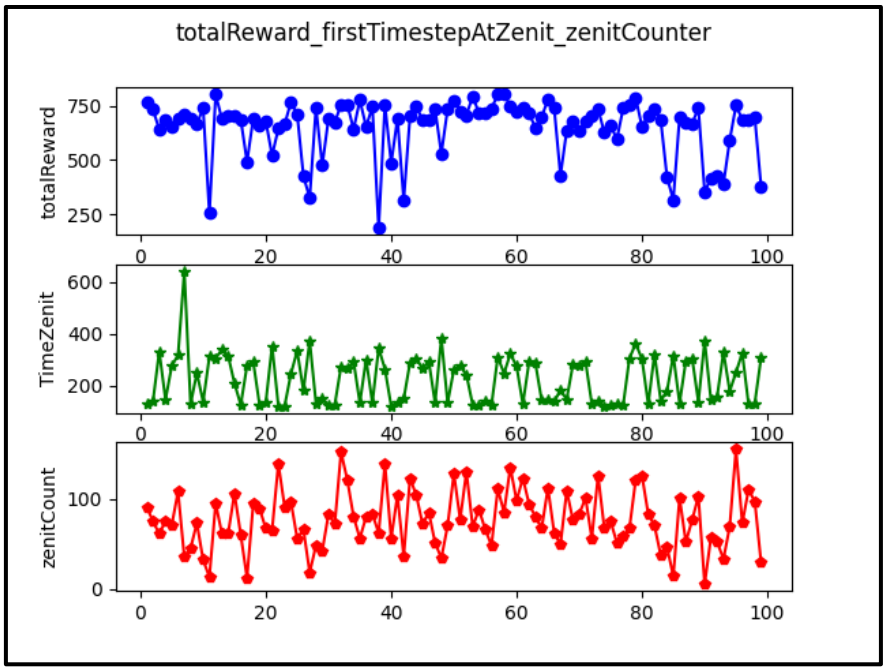

Abbildung 23: Ergebnis SH=Delay=4

Der Reward und vor allem auch der zenitCounter ist sehr schlecht gegenüber dem Ergebnis aus der Simulation in Abschnitt 5.1.

Wenn wir also das Ergebnis SH=4 und Delay=4 mit einer bestimmten Einstellung verbessern, damit wir insbesondere bessere zenitCounter-Werte bekommen, können wir mit den gleichen Einstellungen ein viel besseres Ergebnis im Versuch erwarten. In dem nächsten Abschnitt versuchen wir bessere Werte für den Mean-Reward und den zenitCounter zu erzielen.

7.4 Optimierung des Ergebnisses SH=4 und Delay=4

Aus dem Ergebnis vom Abschnitt 7.3 kann man in der graphischen Darstellung der Pendelbewegung sehen, dass das Pendel viel schneller dreht, öfter am Zenit vorbeikommt. Das einzige Problem ist, dass das Pendel am Zenit nicht stehen bleibt. Das gleiche Verhalten kann man auch in dem ersten Versuchsergebnis aus Abschnitt 5.2 beobachten. Mit der Berücksichtigung von SH und Delay in der Simulation, korrespondieren die beiden Modelle (Versuch und Simulation) gut. Daraus kann man schließen, dass der Agent anhand der verwendeten Reward-Funktion zwar gelernt hat, den Zenit zu erreichen, aber nicht dort stehen zu bleiben. Wir müssen also dem Agenten anhand der Reward-Funktion dazu anregen die Winkelgeschwindigkeit klein zu halten, wenn er am Zenit angekommen ist. Aus diesem Grund definieren wir in der step() Methode eine zusätzliche Variable Namens Boni, die eine Kombination aus Winkelgeschwindigkeit und Winkel ist und fügen diese Variable dem Reward hinzu. Für die Boni-Definition fügen wir in der step() Methode folgenden Codeabschnitt ein:

```
1   Boni= 0.0
2   if(abs(theta)<(5*np.pi/180) and abs(theta_dot)<6):
3       Boni=10.0
```

Code 21. Boni-Definition

Wir vergeben dem Agenten einen Wert von Zehn, wenn das Pendel am Zenit ist (Winkel kleiner als 5 in der Simulation bzw. größer 175 im Versuch) und die Winkelgeschwindigkeit kleiner sechs ist.

Wir ändern den Namen der ursprünglichen Reward-Funktion ohne Boni zu „Reward_NoBoni“ und addieren diese zu den Boni, um die endgültige Reward-Funktion zu erhalten:

$$Reward\_\theta = (\cos(\theta) + 1)/2 \tag{1}$$

$$Reward\_x = cos\left(\left({}^{x}/_{x_{threshold}}\right) \bullet {}^{\pi}/_{2}\right) \tag{2}$$

$$Reward\_NoBoni = Reward\_\theta \bullet Reward\_x \tag{3}$$

$$Reward = Reward\_NoBoni + Boni \tag{4}$$

Wie die Gleichung 4 zeigt, ist der neue Reward der bisherige Reward plus Boni.

In der Step() Methode geben wir nach wie vor nur den Reward aus. Um den Einfluss der Boni auf den Reward besser zu erfassen, fügen wir zu den bisherigen Diagrammen noch ein zusätzliches Diagramm mit dem Namen RewardNoBoni hinzu. Bei diesem Diagramm lassen wir die Summe aller RewardNoBoni aus Gleichung 3 für jede Episode darstellen. Wenn wir keine Boni berücksichtigen, haben dieses Diagramm und das totalReward Diagramm den gleichen Verlauf. Die folgende Abbildung und Tabelle zeigen die Ergebnisse aus einer Simulation. Abbildung 24 zeigt den Trainingsverlauf, Abbildung 25 zeigt die Evaluierung nach dem Training und in Tabelle 10 sind die Ergebnisse zusammengefasst. Mit den Boni ändert sich der Zielwert vom Reward von 1000 auf 11000, weil jede Episode aus 1000

Zeitschritte besteht und in einem Zeitschritt ein maximaler Reward-Wert von 11 (bestehend aus einem Grundreward von 1 plus einem Boni von 10) erreicht werden kann. Es ist sehr bemerkenswert, wie wir allein durch das Einfügen von den Boni den Reward_NoBoni von 651 auf 808 und vor allem den zenitCounter von 77 auf 369 verbessern können. Die graphische Darstellung der Pendelbewegung zeigt, dass das Pendel öfter im Zenit steht als in der Basis-Simulation ohne Boni.

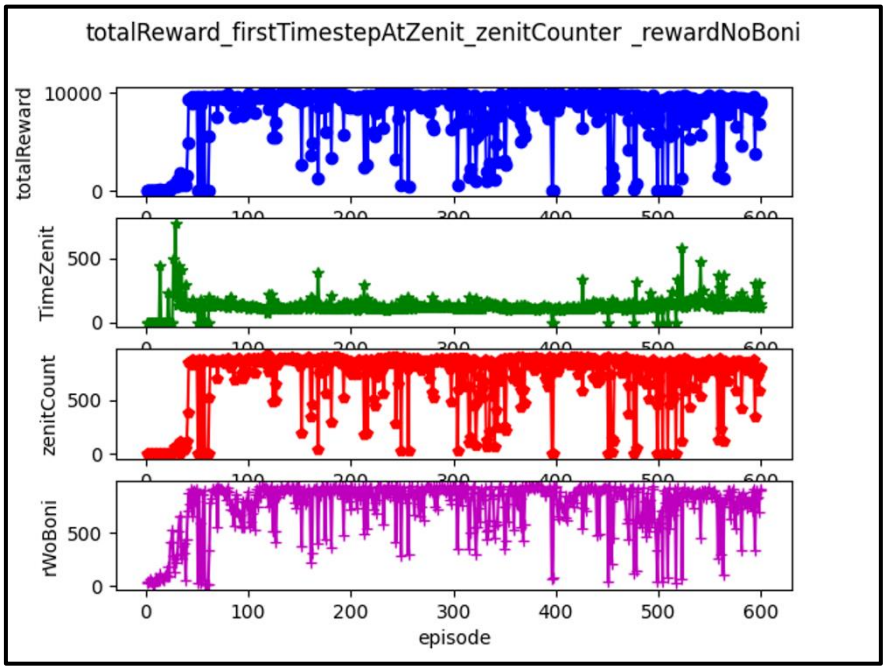

Abbildung 24: Trainingsverlauf SH=Delay=4 mit Boni

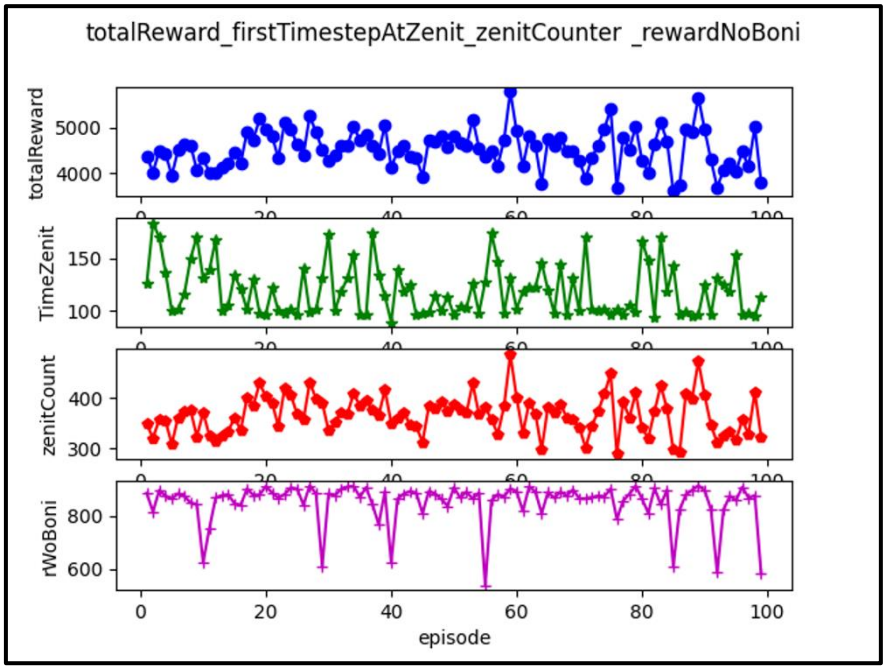

Abbildung 25: Evaluation SH=Delay=4 mit Boni

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	4534	417	11000	0
zenitCounter	369	38	keine	keine
1stTimesAtZenit	119	24	keine	Keine
Reward ohne Boni	808	123	1000	0

Tabelle 10: Standardabweichungen und Mittelwerte SH=Delay=4 mit Boni

Wir haben es durch das Einfügen der Variable Boni geschafft, das Pendel öfters zum Zenit aufschwingen zu lassen, selbst wenn der durchschnittliche Gesamtreward mit 4534 weit vom Zielwert von 11000 entfernt ist. Es ist auch klar, dass wir den Zielwert von 11000 nie erreichen werden, weil das Pendel ein paar kleine Schwingungen am Anfang macht, bevor es den Zenit erreicht. Diese Schwingungen nehmen einerseits ein paar Zeitschritte in Anspruch, anderseits führen die kleinen Schwingungen zu niedrigeren Reward-Werten.

Im nächsten Abschnitt fügen wir auch die Variable Boni in der step() Methode des Codes für den realen Versuch ein und lassen den Agenten neu trainieren.

8 Versuchsergebnisse unter Berücksichtigung von Boni

Aus dem Abschnitt 7.4 haben wir die Erkenntnisse gewonnen, dass die Performanz des Agenten mit den Boni gesteigert werden kann. Im realen Versuch erweitern wir den Reward also mit den Boni und lassen den Versuch wiederholen.

Code 22 zeigt die neue Reward-Definition in der step() Methode der Pendel_Raspi_4Obs_class.py-Klasse.

```
1   if abs(obs[0])>175/180 and abs(obs[1])<6/self.u_dot_threshold:
2       Boni=10.0
3   reward_theta = (-np.cos((u*np.pi)/180)+1.0)/2.0
4   reward_x = np.cos((y/self.y_threshold)*(np.pi/2.0))
5   reward = reward_theta*reward_x + Boni
```

Code 22. Boni Definition im Versuch

Wie in der Simulation geben wir der Variable Boni einen Wert von 10, wenn der Winkel größer als 175 Grad und die Winkelgeschwindigkeit kleiner als sechs ist. Neben den Boni machen wir auch Untersuchungen mit verschiedenen Parametern. Hier wird nur das Ergebnis mit der besten Einstellung gezeigt. Im Anhang befindet sich eine Tabelle mit den Ergebnissen der untersuchten Parameter.

8.1 Erstes Versuchsergebnis mit Boni

Wir lassen neben dem gesamten Reward auch den Reward_NoBoni darstellen, um den Einfluss der Boni auf das Lernverhalten des Agenten besser zu analysieren.

Abbildung 26 zeigt den Trainingsverlauf, Abbildung 27 und Tabelle 11 zeigen die Ergebnisse nach dem Training.

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	9024	719	1000	0
zenitCounter	822	70	keine	keine
1stTimesAtZenit	109	9	keine	Keine
Reward ohne Boni	790	98	1000	0

Tabelle 11: Standardabweichungen und Mittelwerte Versuchsergebnis mit Boni

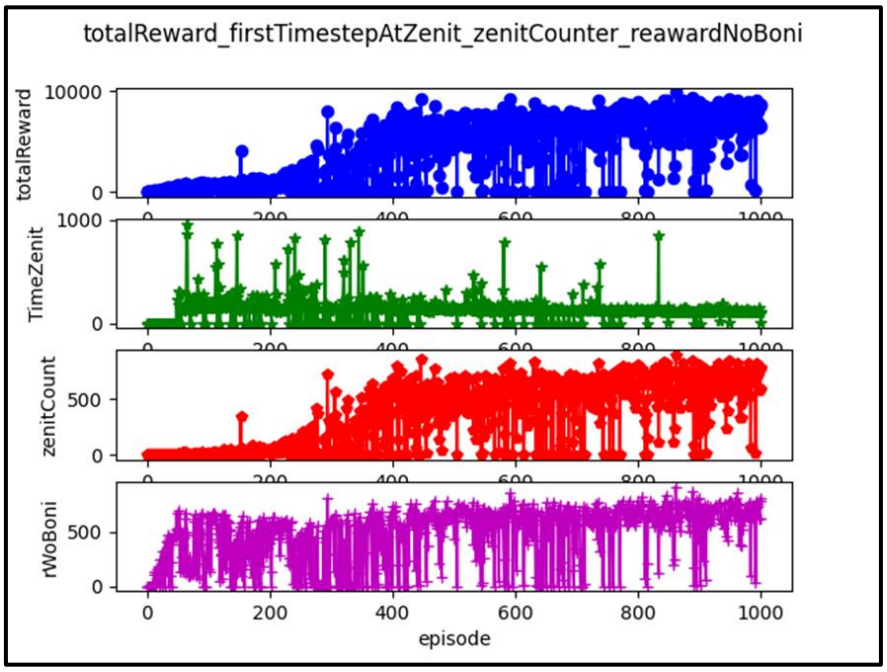

Abbildung 26: Trainingsverlauf realer Versuch mit Boni

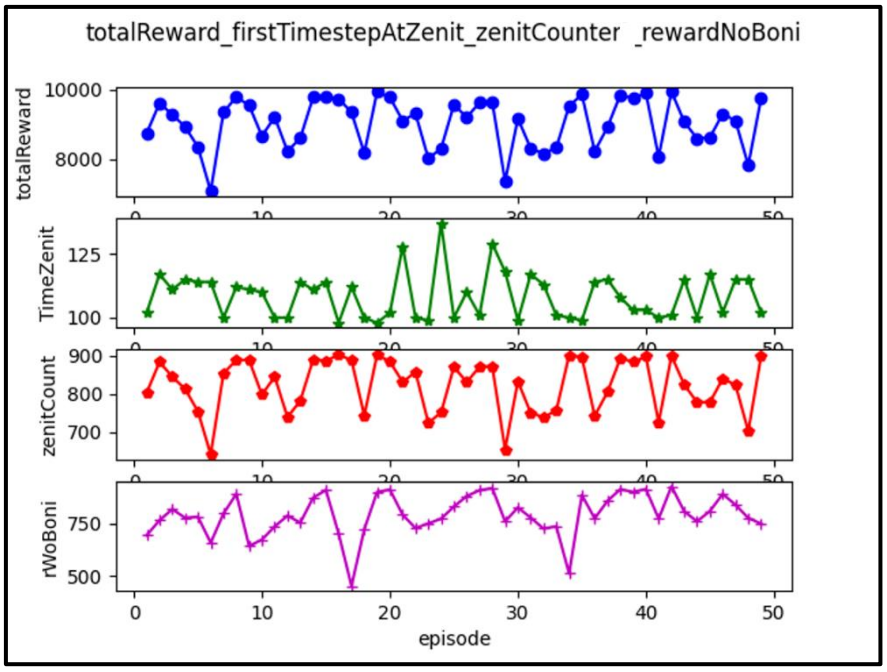

Abbildung 27: Auswertung Versuchsergebnis mit Boni nach dem Training

Die Ergebnisse sind viel besser als das Basis-Ergebnis ohne Boni aus dem Abschnitt 5.2. Aus den Tabellen 5 und 11 wird dies ersichtlich. Allein durch die Boni haben wir den Reward_NoBoni von 604 auf 790 und den zenitCounter von 3 auf 822 verbessert. Das ist ein gutes Ergebnis. Auch eine Wiederholung des Versuches mit anderen Agenten zeigt ähnliche Ergebnisse. Abbildung 28 zeigt das Pendel vor und nach dem Training.

Mit der verwendeten reset() Methode aus der Pendel_Raspi_4Obs_class.py-Klasse kann der Startwinkel in jeder Episode nur Werte zwischen -10 und 10 annehmen. Dies ist gewollt, damit der Agent mit möglichst kleinen Startwinkeln trainiert wird.

Abbildung 28: Pendel-Zustand vor (links) und nach dem Training (rechts)

Das Diagramm in Abbildung 27 zeigt die Ergebnisse mit einem kleinen Startwinkel beim Training. Mit der gleichen Einstellung (also kleine Startwinkel) wurde die Auswertung (Testlauf nach dem Training) gemacht.

In den nächsten Abschnitten untersuchen wir das Lernverhalten des Agenten mit kleinem und größerem Startwinkel.

8.2 Einfluss des Startwinkels auf die Ergebnisse des Trainings und des Testlaufs nach dem Training

8.2.1 Startwinkel kleiner 10 Grad im Training und größer 30 Grad im Testlauf nach dem Training (mit bereits trainiertem Agenten)

Das Ergebnis während des Trainings und die Auswertung nach dem Training jeweils mit Startwinkel kleiner 10 Grad kennen wir bereits (siehe Abschnitt 8.1). Wir passen jetzt die reset() Methode an, sodass der Anfangswinkel im Testlauf nach dem Training Werte größer 30 Grad annehmen kann und lassen den Agenten wie zuvor mit 50 Episoden evaluieren. Zur Anpassung der ursprünglichen reset() Methode auf größere Startwinkel, definieren wir eine Funktion resetWrapper(), lassen das trainierte Modell in einer for-Schleife laufen und geben am Ende der Schleife den Zustand aus.

```
1   def resetWrapper(self):
2       self.reset_model.load_model(PATH_RESET_MODEL)
3       state= self.reset()
4       state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
5       for _ in range(30):
6           action= self.get_actionForReset(state)
7           next_state, _, _ = self.step(action)
8           stateNotReshape= copy(next_state)
9           next_state = np.reshape(next_state, newshape=(1, -
1)).astype(np.float32)
10          state= next_state
11      self.t = 0
12      return   np.array(stateNotReshape)
```

Code 23. resetWrapper() Methode

Die resetWrapper() Methode ist ähnlich der evaluateDqn() Methode, die wir verwenden, um den Agenten nach dem Training zu evaluieren. Wie in der evaluateDqn() Methode, benötigt resetWrapper() ein dqn-Netz und eine Methode, die die Aktion mit dem maximalen Reward liefert. Aus diesem Grund haben wir ein zusätzliches Netz (reset_model) mit der gleichen Architektur im Konstruktor angelegt und eine neue Methode get_actionForReset() definiert.

```
2   self.reset_model = DQN(
3   state_shape=self.observations,
4   num_actions=self.actions,
5   learning_rate=0.001)
6   def get_actionForReset(self, state):
7       return np.argmax(self.reset_model(state))
```

Code 24. reset_modell und get_ actionForReset() Methode

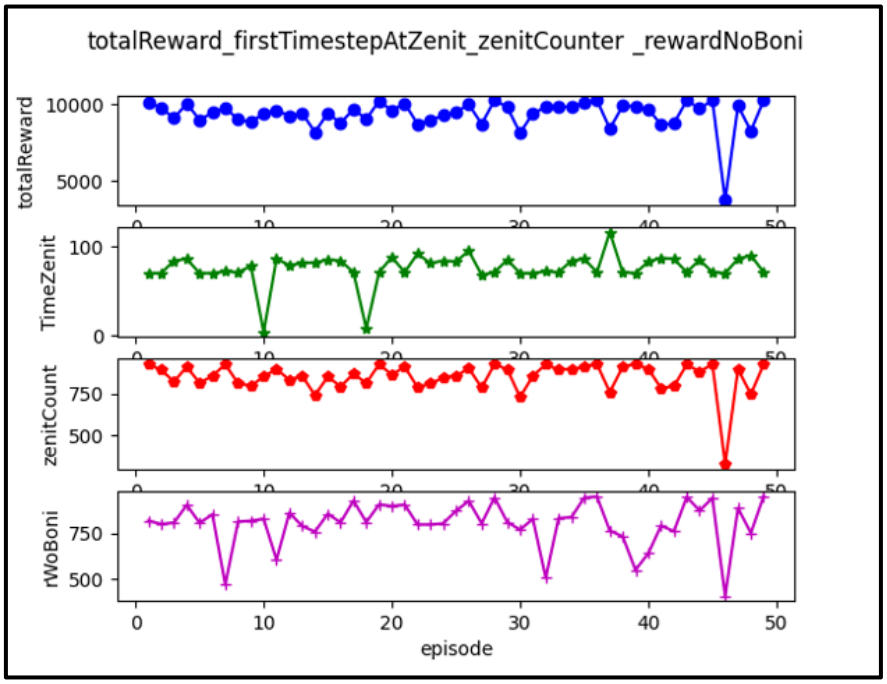

Abbildung 29: Ergebnisse des Testlaufs mit größerem Startwinkel, auf Basis des Trainings mit kleinerem Startwinkel

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	9345	999	1000	0
zenitCounter	852	95	keine	keine
1stTimesAtZenit	76	17	keine	Keine
Reward ohne Boni	808	123	1000	0

Tabelle 12: Ergebnisse des Testlaufs mit größerem Startwinkel, auf Basis des Trainings mit kleinerem Startwinkel

Wie erwartet, sind die Ergebnisse mit größerem Startwinkel besser als das Basis-Ergebnis. Mit größerem Startwinkel benötigt der Agent eine kleinere Anzahl an Iterationen, um am Zenit anzukommen, außerdem ist der Reward_NoBoni von Anfang an größer.

Nun lassen wir den Agenten mit größerem Winkel trainieren und machen die Auswertung zuerst mit gleichem Startwinkel wie im Training und danach mit kleinerem Startwinkel als im Training.

8.2.2 Startwinkel größer 30 Grad

Für den Anfangszustand rufen wir vor jeder Episode die resetWrapper() Methode und nicht mehr die „Basis-Reset“ Methode auf und lassen den Agenten mit Startwinkel größer 30 Grad trainieren. Danach folgt ein Testlauf mit dem gleichem Startwinkel.

Abbildung 30 und 31 zeigen den Trainingsverlauf und das Ergebnis nach dem Training.

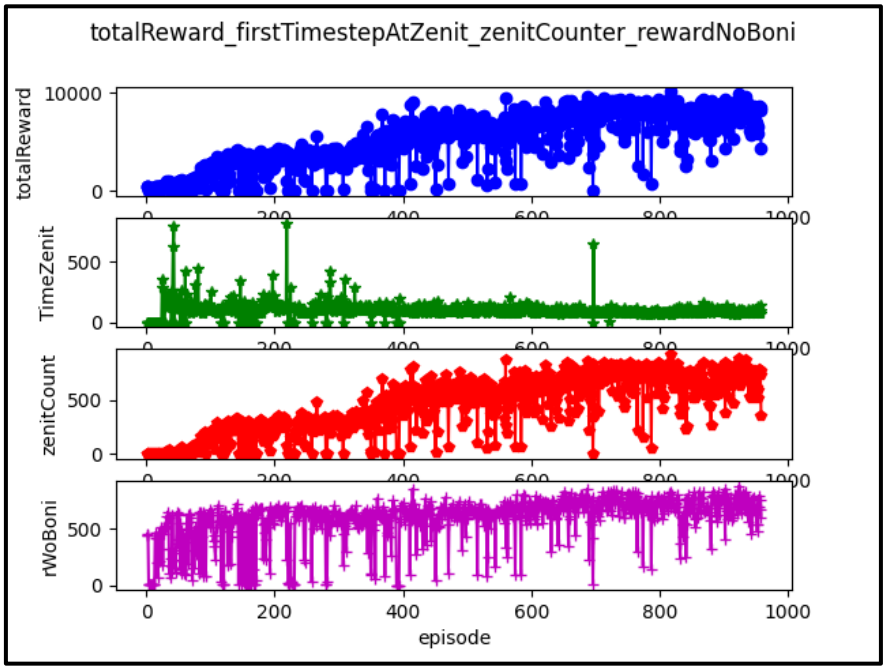

Abbildung 30: Trainingsverlauf Agent mit größerem Startwinkel

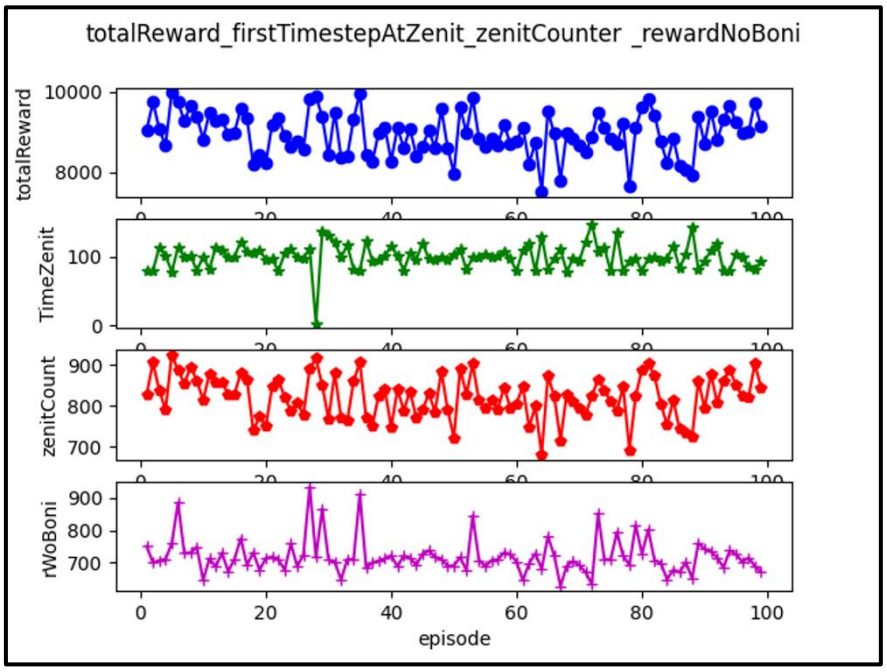

Abbildung 31: Auswertung und Training mit größerem Startwinkel

Entgegen unserer Erwartung ist das Ergebnis der Auswertung mit größerem Startwinkel insbesondere der totalReward und der Reward ohne Boni nicht besser als das Basis-Ergebnis aus Tabelle 11. Ein Vergleich der Trainingsdiagramme zeigt aber, dass der Agent von den Episoden zwischen 0 und 200 viele größere Reward Werte bekommt als das Basis-Ergebnis mit kleinen Startwinkeln beim Training. Außerdem muss man die Versuche mehrfach wiederholen, um eine finale Schlussfolgerung ziehen zu können.

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	8957	540	11000	0
zenitCounter	821	53	keine	keine
1stTimesAtZenit	100	18	keine	Keine
Reward ohne Boni	719	53	1000	0

Tabelle 13: Auswertung des Testlaufs nach dem Training mit größerem Startwinkel, auf Basis des Trainings mit größerem Startwinkel

Wir machen jetzt mit dem gleichen Trainingsmodell die Auswertung mit kleinen Startwinkel. Die Ergebnisse sind in Tabelle 14 zusammengefasst. Wie erwartet, sind die Werte schlechter als die Werte aus Tabelle 13. Der Agent braucht mit kleinen Startwinkel eine gewisse Anzahl an Iterationen mehr, bis er Zuständen mit größerem Winkel erreicht, diese kleinen Winkel führen zu den kleinen Reward-Werten.

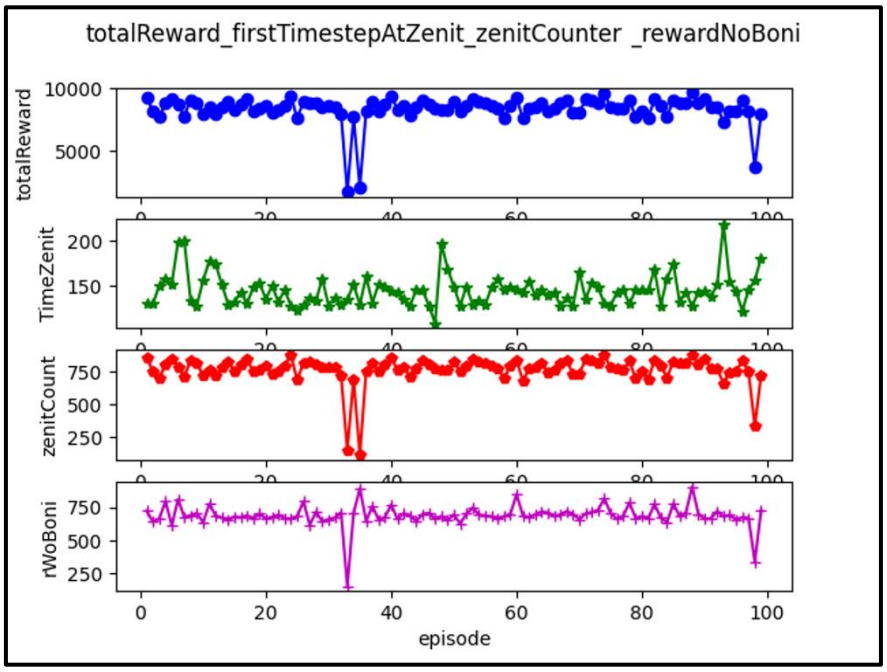

Abbildung 32: Testlauf mit kleinerem Startwinkel, auf Basis des Trainings mit größerem Startwinkel

Ergebnis für 50 Episoden			Zielwerte (mean, std)	
Diagramm	mean	std	mean	std
totalReward	8334	1149	11000	0
zenitCounter	762	111	keine	keine
1stTimesAtZenit	100	18	keine	Keine
Reward ohne Boni	690	83	1000	0

Tabelle 14: Auswertung mit kleinerem Startwinkel Training mit größerem Startwinkel

9 Zusammenfassung

Wir haben es geschafft, das reale Pendel zum Aufschwingen zu bringen und über längere Zeit im Zenit zu halten. Folgende Gründe könnten den Durchbruch und diesen Erfolgt gegenüber früheren gescheiterten Versuchen erklären:

- Für die Zustandsbeschreibung vom Schlitten haben wir neben der Position auch die Geschwindigkeit berücksichtigt. Somit konnte der Zustand vom Schlitten vollständig beschrieben werden. In den früheren Versuchen wurde die Geschwindigkeit vom Schlitten nicht berücksichtigt. Eine Wiederholung des Versuches aus Abschnitt 8.1 mit drei Zustandsvariablen ($\theta,\ \dot{\theta},\ y$) zeigt schlechte Ergebnisse. Der maximale Reward-Wert nach 780 Trainings-Episoden lag bei 210 (Zielwert: 11000) und der zenitCounter bei 0.
- Die Reward-Funktion in Form des Grundrewards „Reward_NoBoni“ (siehe Gleichung 3) unter Berücksichtigung des Bonusrewards (Boni genannt), hat sich als geeignet erwiesen, dem Agenten die nötigen Informationen zu liefern, um das Trainingsziel (das Balancieren im Zenit) zu erreichen. In früheren Versuchen war Boni auch berücksichtigt worden und die Definition ist der jetzigen gleich, aber der Reward_NoBoni wurde folgendermaßen definiert:

$$Reward\_NoBoni = \left|\frac{\theta}{180}\right| + 0.1 \bullet \left(1 - \left|\frac{x}{x_{threshold}}\right|\right) \tag{5}$$

  Eine Wiederholung des Versuches aus Abschnitt 8.1 mit $Boni = 1$ und dem Grundreward aus Gleichung 5, wie in den früheren Versuchen, zeigt nach 1000 Trainings-Episoden und dem Testlauf mit fünfzig Episoden, einen Reward-Wert von 1160 (Zielwert: 2100) und der zenitCounter liegt bei 349.

  Die Definition von Reward_NoBoni in der früheren Implementierung scheint nicht optimal für das Cart-Pole-Swing-Up-Problem zu sein.

- Eine Einsicht in eine DQN-Klasse aus dem früheren Versuch zeigt viele Unterschiede gegenüber der jetzigen Implementierung. Als Optimizer wurde zum Beispiel SGD benutzt, mit diesen Algorithmen konnte kein gutes Ergebnis während des Praxisprojekts in der Simulation erzielt werden. Außerdem war der Replay Buffer in den früheren Versuchen eine Liste ohne Begrenzung, was eventuell dazu führen kann, dass über mehreren Episoden teilweise die gleichen Daten zum Training benutzt werden und der Lernprozess des Agenten dadurch erschwert wird.

Anhand der Simulation waren wir in der Lage den DQN-Algorithmus für das Cart-Pole-Swing-Up-Problem zu optimieren und die Komplexität des realen Pendels vorab genauer in der Simulation zu untersuchen. Im realen Versuch wurde das Training mit der besten Einstellung viermal wiederholt und die Ergebnisse wurden bestätigt. Es ist empfehlenswert, die Simulation sowie den realen Versuch zu wiederholen, damit man bessere Statistikwerte erfassen kann. Es gibt auch einen Spruch, der besagt: „Ein Versuch ist kein Versuch“. Das

bedeutet, dass man mehrere Versuche machen muss, bevor man eine endgültige Schlussfolgerung ziehen kann.

Beim realen Versuchen kommt es manchmal vor, dass ein Verbindungsproblem auf dem Raspberry mit Dropbox (dropbox.exception.InternalServerError) oder mit der BUS-Verbindung auftritt. In diesem Fall hilft meistens ein Neustart des Versuchs auf dem Raspberry. Auf dem Hochleistungsrechner kommt es auch zum Verbindungsproblem mit der Dropbox. Hier hilft es die Dropbox-App neu zu starten, nachdem man den Prozess im Task-Manager beendet hat. Eine Ab- und Anmeldung mit den Kontodaten auf der Dropbox-Seite kann auch in manchen Fällen helfen.

Diese Arbeit ist ein guter Grundstein für weitere Untersuchungen. Im Praxisprojekt (siehe PPB Abschnitt 6.2) gab es weitere Algorithmen, wie die ACER und PPO Algorithmen, die für die Simulation gut funktioniert haben. Eine Anwendung dieser Algorithmen auf das reale Pendel könnte getestet werden. Eine weitere Untersuchung könnte die Variation der Aktionsdauer sein, um einerseits die beste Aktionsdauer anderseits die kritische Dauer, unter der der Agent nicht erfolgreich lernen kann, zu ermitteln.

10Anhang

A. cartPoleSwingUp.py

```
"""
Cart pole swing-up: Original version from:
https://github.com/zuoxingdong/DeepPILCO/blob/master/cartpole_swingup.py
"""
import logging
import math
import gym
from gym import error, spaces, utils
from gym.utils import seeding
import numpy as np
from copy import deepcopy, copy
logger = logging.getLogger(__name__)
class CartPoleSwingUpEnv(gym.Env):
    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second': 50
    }
    def __init__(self):
        self.g = 9.82  # gravity
        self.m_c = 0.5  # cart mass
        self.m_p = 0.5  # pendulum mass
        self.total_m = (self.m_p + self.m_c)
        self.l = 0.6  # pole's length
        self.m_p_l = (self.m_p * self.l)
        self.force_mag = 10.0
        self.dt = 0.01  # seconds between state updates
        self.b = 0.00001  # initial value 0.1 friction coefficient => frictionless
        self.t = 0  # timestep
        self.t_limit = 1000
        self.pole_frict = 0.005  # 0.0095
        self.x_threshold = 40
        self.theta_dot_threshold = 40 # same value will be used to normalise x_dot
        high = np.array([self.x_threshold, self.theta_dot_threshold, np.pi, self.theta_dot_threshold], dtype=np.float32)
        self.action_space = spaces.Discrete(3)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)
        self.seed()
        self.viewer = None
        self.state = None
        # add from joel
        self.steps_beyond_done = None
    def seed(self, seed=None):
```

```
39          self.np_random, seed = seeding.np_random(seed)
40          return [seed]
41      def step(self, action):
42          Boni = 0.0
43          err_msg = "%r (%s) invalid" % (action, type(action))
44          assert self.action_space.contains(action), err_msg
45          x, x_dot, theta, theta_dot = self.state
46          force = self.force_mag if action == 1 else -
self.force_mag if action == 2 else 0
47          s = math.sin(theta)
48          c = math.cos(theta)
49          xdot_update = (-2 * self.m_p_l * (
50                      theta_dot ** 2) * s + 3 * self.m_p * self.g
* s * c + 4 * force - 4 * self.b * x_dot) / (
51                                      4 * self.total_m - 3 *
self.m_p * c ** 2)
52          thetadot_update = (-3 * self.m_p_l * (theta_dot ** 2) *
s * c + 6 * self.total_m * self.g * s + 6 * (
53                      force - self.b * x_dot) * c) / (4 * self.l *
self.total_m - 3 * self.m_p_l * c ** 2)
54          x = x + x_dot * self.dt
55          theta = theta + theta_dot * self.dt
56          x_dot = x_dot + xdot_update * self.dt
57          theta_dot = (1 - self.pole_frict) * theta_dot +
thetadot_update * self.dt
58          # wrapped theta to 2*pi
59          if theta > np.pi:
60              theta = -np.pi + (theta - np.pi)
61          elif theta < -np.pi:
62              theta = np.pi - (-theta - np.pi)
63          # Normalization
64          self.state = (x, x_dot, theta, theta_dot)
65          x_n, x_dot_n, theta_n, theta_dot_n = copy(self.state)
66          obs = (x_n / self.x_threshold, x_dot_n /
self.theta_dot_threshold, theta_n / np.pi, theta_dot_n /
self.theta_dot_threshold)
67          self.t += 1
68          done = bool(
69              x < -self.x_threshold
70              or x > self.x_threshold
71              or abs(theta_dot) > self.theta_dot_threshold
72              or self.t >= self.t_limit
73          )
74          if (abs(theta)<(5*np.pi/180) and abs(theta_dot)<6):
75            Boni =10.0
76          reward_theta = (np.cos(theta) + 1.0) / 2.0
77          reward_x = np.cos((x / self.x_threshold) * (np.pi /
2.0))
78          reward = reward_theta * reward_x + Boni
79          return np.array(obs), reward, done, {}
80      def reset(self):
```

```
81          self.state = np.random.normal(loc=np.array([0.0, 0.0,
np.pi, 0.0]), scale=np.array([0.2, 0.2, 0.2, 0.2]))
82          x_n, x_dot_n, theta_n, theta_dot_n = copy(self.state)
83          obs = (x_n / self.x_threshold, x_dot_n /
self.theta_dot_threshold, theta_n / np.pi, theta_dot_n /
self.theta_dot_threshold)
84          self.steps_beyond_done = None
85          self.t = 0  # timestep
86          return np.array(self.state)
87      def close(self):
88          if self.viewer is not None:
89              self.viewer.close()
90              self.viewer = None
91          return
92      def render(self, mode='human', close=False):
93          if close:
94              if self.viewer is not None:
95                  self.viewer.close()
96                  self.viewer = None
97              return
98          screen_width = 1600
99          screen_height = 600  # before was 400
100         world_width = 5  # max visible position of cart
101         scale = 18#b was screen_width / world_width
102         carty = screen_height / 2  # TOP OF CART
103         polewidth = 6.0
104         polelen = 120 * self.l  # 0.6 or self.l
105         cartwidth = 40.0
106         cartheight = 20.0
107         if self.viewer is None:
108             from gym.envs.classic_control import rendering
109             self.viewer = rendering.Viewer(screen_width,
screen_height)
110             l, r, t, b = -cartwidth / 2, cartwidth / 2,
cartheight / 2, -cartheight / 2
111             cart = rendering.FilledPolygon([(l, b), (l, t), (r,
t), (r, b)])
112             self.carttrans = rendering.Transform()
113             cart.add_attr(self.carttrans)
114             cart.set_color(1, 0, 0)
115             self.viewer.add_geom(cart)
116             l, r, t, b = -polewidth / 2, polewidth / 2, polelen
- polewidth / 2, -polewidth / 2
117             pole = rendering.FilledPolygon([(l, b), (l, t), (r,
t), (r, b)])
118             pole.set_color(0, 0, 1)
119             self.poletrans = rendering.Transform(translation=(0,
0))
120             pole.add_attr(self.poletrans)
121             pole.add_attr(self.carttrans)
122             self.viewer.add_geom(pole)
123             self.axle = rendering.make_circle(polewidth / 2)
```

```
124             self.axle.add_attr(self.poletrans)
125             self.axle.add_attr(self.carttrans)
126             self.axle.set_color(0.1, 1, 1)
127             self.viewer.add_geom(self.axle)
128             # Make another circle on the top of the pole
129             self.pole_bob = rendering.make_circle(polewidth / 2)
130             self.pole_bob_trans = rendering.Transform()
131             self.pole_bob.add_attr(self.pole_bob_trans)
132             self.pole_bob.add_attr(self.poletrans)
133             self.pole_bob.add_attr(self.carttrans)
134             self.pole_bob.set_color(0, 0, 0)
135             self.viewer.add_geom(self.pole_bob)
136             self.wheel_l = rendering.make_circle(cartheight / 4)
137             self.wheel_r = rendering.make_circle(cartheight / 4)
138             self.wheeltrans_l =
rendering.Transform(translation=(-cartwidth / 2, -cartheight / 2))
139             self.wheeltrans_r =
rendering.Transform(translation=(cartwidth / 2, -cartheight / 2))
140             self.wheel_l.add_attr(self.wheeltrans_l)
141             self.wheel_l.add_attr(self.carttrans)
142             self.wheel_r.add_attr(self.wheeltrans_r)
143             self.wheel_r.add_attr(self.carttrans)
144             self.wheel_l.set_color(0, 0, 0)  # Black, (B, G, R)
145             self.wheel_r.set_color(0, 0, 0)  # Black, (B, G, R)
146             self.viewer.add_geom(self.wheel_l)
147             self.viewer.add_geom(self.wheel_r)
148             self.track = rendering.Line(
149                 (screen_width / 2 - self.x_threshold * scale,
carty - cartheight / 2 - cartheight / 4),
150                 (screen_width / 2 + self.x_threshold * scale,
carty - cartheight / 2 - cartheight / 4))
151             self.track.set_color(0, 0, 0)
152             self.viewer.add_geom(self.track)
153         if self.state is None: return None
154         x = self.state
155         cartx = x[0] * scale + screen_width / 2.0  # MIDDLE OF
CART
156         self.carttrans.set_translation(cartx, carty)
157         self.poletrans.set_rotation(x[2])
158         self.pole_bob_trans.set_translation(-self.l *
np.sin(x[2]), self.l * np.cos(x[2]))
159         return self.viewer.render(return_rgb_array=mode ==
'rgb_array')
```

B. cartPoleSwingUpEnvFixState.py

```
"""
Cart pole swing-up: Original version from:
https://github.com/zuoxingdong/DeepPILCO/blob/master/cartpole_swingup.py
"""
import logging
import csv
import math
import gym
from gym import error, spaces, utils
from gym.utils import seeding
import numpy as np
from copy import deepcopy, copy
FileForEvaluation="fixStates.csv"
logger = logging.getLogger(__name__)
class CartPoleSwingUpEnvFixState(gym.Env):
    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second': 50
    }
    def __init__(self):
        self.g = 9.82  # gravity
        self.m_c = 0.5  # cart mass
        self.m_p = 0.5  # pendulum mass
        self.total_m = (self.m_p + self.m_c)
        self.l = 0.6  # pole's length
        self.m_p_l = (self.m_p * self.l)
        self.force_mag = 10.0
        self.dt = 0.01  # seconds between state updates
        self.b = 0.00001  # intial value 0.1 friction coefficient => frictionless
        self.t = 0  # timestep
        self.t_limit = 1000
        self.pole_frict = 0.005
        self.x_threshold = 40
        self.theta_dot_threshold = 40 # same value will be used to narmalise x_dot
        high = np.array([self.x_threshold, self.theta_dot_threshold, np.pi, self.theta_dot_threshold], dtype=np.float32)
        self.action_space = spaces.Discrete(3)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)
        self.seed()
        self.viewer = None
        self.state = None
        self.steps_beyond_done = None
        self.stateListe=[]
        self.stateCollecter= np.zeros(4)
```

```
41          with open(FileForEvaluation, newline='', encoding='utf-
8') as f:
42              reader = csv.reader(f)
43              for row in reader:
44                  self.stateCollecter[0]=float(row[0])
45                  self.stateCollecter[1]=float(row[1])
46                  self.stateCollecter[2]=float(row[2])
47                  self.stateCollecter[3]=float(row[3])
48                  self.stateListe.append(self.stateCollecter)
49                  self.stateCollecter= np.zeros(4)
50      def seed(self, seed=None):
51          self.np_random, seed = seeding.np_random(seed)
52          return [seed]
53      def step(self, action):
54          Boni = 0.0
55          err_msg = "%r (%s) invalid" % (action, type(action))
56          assert self.action_space.contains(action), err_msg
57          x, x_dot, theta, theta_dot = self.state
58          force = self.force_mag if action == 1 else -
self.force_mag if action == 2 else 0
59          s = math.sin(theta)
60          c = math.cos(theta)
61          xdot_update = (-2 * self.m_p_l * (
62                         theta_dot ** 2) * s + 3 * self.m_p * self.g
* s * c + 4 * force - 4 * self.b * x_dot) / (
63                                          4 * self.total_m - 3 *
self.m_p * c ** 2)
64          thetadot_update = (-3 * self.m_p_l * (theta_dot ** 2) *
s * c + 6 * self.total_m * self.g * s + 6 * (
65                         force - self.b * x_dot) * c) / (4 * self.l *
self.total_m - 3 * self.m_p_l * c ** 2)
66          x = x + x_dot * self.dt
67          theta = theta + theta_dot * self.dt
68          x_dot = x_dot + xdot_update * self.dt
69          theta_dot = (1 - self.pole_frict) * theta_dot +
thetadot_update * self.dt
70          # wrapped theta to 2*pi
71          if theta > np.pi:
72              theta = -np.pi + (theta - np.pi)
73          elif theta < -np.pi:
74              theta = np.pi - (-theta - np.pi)
75          # Normalization
76          self.state = (x, x_dot, theta, theta_dot)
77          x_n, x_dot_n, theta_n, theta_dot_n = copy(self.state)
78          obs = (x_n / self.x_threshold, x_dot_n /
self.theta_dot_threshold, theta_n / np.pi, theta_dot_n /
self.theta_dot_threshold)
79          self.t += 1
80          done = bool(
81              x < -self.x_threshold
82              or x > self.x_threshold
83              or abs(theta_dot) > self.theta_dot_threshold
```

```
84              or self.t >= self.t_limit
85          )
86          if (abs(theta)<(5*np.pi/180) and abs(theta_dot)<6):
87            Boni =10.0
88          reward_theta = (np.cos(theta) + 1.0) / 2.0
89          reward_x = np.cos((x / self.x_threshold) * (np.pi /
2.0))
90          reward = reward_theta * reward_x + Boni
91          return np.array(obs), reward, done, {}
92      def reset(self):
93          self.state = self.stateListe.pop(0)
94          x_n, x_dot_n, theta_n, theta_dot_n = copy(self.state)
95          obs = (x_n / self.x_threshold, x_dot_n /
self.theta_dot_threshold, theta_n / np.pi, theta_dot_n /
self.theta_dot_threshold)
96          self.steps_beyond_done = None
97          self.t = 0  # timestep
98          return np.array(obs)
99      def close(self):
100         if self.viewer is not None:
101             self.viewer.close()
102             self.viewer = None
103         return
104     def render(self, mode='human', close=False):
105         if close:
106             if self.viewer is not None:
107                 self.viewer.close()
108                 self.viewer = None
109             return
110         screen_width = 1600
111         screen_height = 600  # before was 400
112         world_width = 5  # max visible position of cart
113         scale = 18#b was screen_width / world_width
114         carty = screen_height / 2  # TOP OF CART
115         polewidth = 6.0
116         polelen = 120 * self.l  # 0.6 or self.l
117         cartwidth = 40.0
118         cartheight = 20.0
119         if self.viewer is None:
120             from gym.envs.classic_control import rendering
121             self.viewer = rendering.Viewer(screen_width,
screen_height)
122             l, r, t, b = -cartwidth / 2, cartwidth / 2,
cartheight / 2, -cartheight / 2
123             cart = rendering.FilledPolygon([(l, b), (l, t), (r,
t), (r, b)])
124             self.carttrans = rendering.Transform()
125             cart.add_attr(self.carttrans)
126             cart.set_color(1, 0, 0)
127             self.viewer.add_geom(cart)
128             l, r, t, b = -polewidth / 2, polewidth / 2, polelen
- polewidth / 2, -polewidth / 2
```

```
129             pole = rendering.FilledPolygon([(l, b), (l, t), (r,
t), (r, b)])
130             pole.set_color(0, 0, 1)
131             self.poletrans = rendering.Transform(translation=(0,
0))
132             pole.add_attr(self.poletrans)
133             pole.add_attr(self.carttrans)
134             self.viewer.add_geom(pole)
135             self.axle = rendering.make_circle(polewidth / 2)
136             self.axle.add_attr(self.poletrans)
137             self.axle.add_attr(self.carttrans)
138             self.axle.set_color(0.1, 1, 1)
139             self.viewer.add_geom(self.axle)
140             # Make another circle on the top of the pole
141             self.pole_bob = rendering.make_circle(polewidth / 2)
142             self.pole_bob_trans = rendering.Transform()
143             self.pole_bob.add_attr(self.pole_bob_trans)
144             self.pole_bob.add_attr(self.poletrans)
145             self.pole_bob.add_attr(self.carttrans)
146             self.pole_bob.set_color(0, 0, 0)
147             self.viewer.add_geom(self.pole_bob)
148             self.wheel_l = rendering.make_circle(cartheight / 4)
149             self.wheel_r = rendering.make_circle(cartheight / 4)
150             self.wheeltrans_l =
rendering.Transform(translation=(-cartwidth / 2, -cartheight / 2))
151             self.wheeltrans_r =
rendering.Transform(translation=(cartwidth / 2, -cartheight / 2))
152             self.wheel_l.add_attr(self.wheeltrans_l)
153             self.wheel_l.add_attr(self.carttrans)
154             self.wheel_r.add_attr(self.wheeltrans_r)
155             self.wheel_r.add_attr(self.carttrans)
156             self.wheel_l.set_color(0, 0, 0)  # Black, (B, G, R)
157             self.wheel_r.set_color(0, 0, 0)  # Black, (B, G, R)
158             self.viewer.add_geom(self.wheel_l)
159             self.viewer.add_geom(self.wheel_r)
160             self.track = rendering.Line(
161                 (screen_width / 2 - self.x_threshold * scale,
carty - cartheight / 2 - cartheight / 4),
162                 (screen_width / 2 + self.x_threshold * scale,
carty - cartheight / 2 - cartheight / 4))
163             self.track.set_color(0, 0, 0)
164             self.viewer.add_geom(self.track)
165         if self.state is None: return None
166         x = self.state
167         cartx = x[0] * scale + screen_width / 2.0  # MIDDLE OF
CART
168         self.carttrans.set_translation(cartx, carty)
169         self.poletrans.set_rotation(x[2])
170         self.pole_bob_trans.set_translation(-self.l *
np.sin(x[2]), self.l * np.cos(x[2]))
171         return self.viewer.render(return_rgb_array=mode ==
'rgb_array')
```

C. dqn.py

```
# same class for dqn_Raspi_class.py and dqn_PC_class.py
1   import numpy as np
2   from tensorflow.keras.layers import Activation
3   from tensorflow.keras.layers import Dense
4   from tensorflow.keras.layers import Input
5   from tensorflow.keras.models import Model
6   from tensorflow.keras.optimizers import Adam
7   class DQN(Model):
8       def __init__(self, state_shape: int, num_actions: int,
learning_rate: float):
9           super().__init__()
10          self.state_shape = state_shape
11          self.num_actions = num_actions
12          self.learning_rate = learning_rate
13          self.internal_model = self.build_model()
14      def build_model(self) -> Model:
15          input_state = Input(shape=self.state_shape)
16          x = Dense(units=64)(input_state)
17          x = Activation("tanh")(x)
18          x = Dense(units=64)(x)
19          x = Activation("tanh")(x)
20          q_value_pred = Dense(self.num_actions)(x)
21          model = Model(
22              inputs=input_state,
23              outputs=q_value_pred
24          )
25          model.compile(
26              loss="mse",
27              optimizer=Adam(learning_rate=self.learning_rate)
28          )
29          return model
30      def call(self, inputs: np.ndarray) -> np.ndarray:
31          return self.internal_model(inputs).numpy()
32      def fit(self, states: np.ndarray, q_values: np.ndarray):
33          self.internal_model.fit(x=states, y=q_values, verbose=0)
34      def update_model(self, other_model: Model):
35
self.internal_model.set_weights(other_model.get_weights())
36      def load_model(self, path: str):
37          self.internal_model.load_weights(path)
38      def save_model(self, path: str):
39          self.internal_model.save_weights(path)
40  if __name__ == "__main__":
41      dqn = DQN(
42          state_shape=4,
43          num_actions=3,
44          learning_rate=0.001
45      )
46      dqn.internal_model.summary()
```

D. fixStates.csv

```
-0.0181444,-0.0372689,3.2291592,-0.3977114
-0.2239800,-0.0040213,3.2230235,0.0624181
0.0999986,0.2202447,3.0664396,0.1643383
-0.1206628,0.1137099,3.3020050,0.1807922
0.3390817,-0.0228308,2.9456543,-0.1971823
-0.2282394,0.4138532,3.0610459,-0.2538281
0.0491612,-0.2149289,3.0780013,-0.0275859
-0.1547821,-0.1103357,3.2060784,-0.2427627
0.1867016,-0.0465274,3.0751532,0.0213804
-0.1954725,-0.1330547,3.5307716,0.0255828
-0.1218419,0.2076001,3.4372726,-0.4667846
0.1011203,-0.2349090,2.9160672,0.0926761
-0.0775722,0.4278093,3.0410539,-0.2276670
-0.2246038,-0.0029375,2.8592155,-0.1127911
0.2860348,-0.0228498,3.1130596,-0.0505363
0.0602057,0.1187861,3.1007008,-0.2941609
0.2095502,-0.2177506,2.9013159,-0.0188761
-0.0515664,-0.0067958,3.0709789,0.0823522
0.1228192,-0.2523888,3.2183184,-0.0611118
-0.2194874,0.2085349,3.3456960,0.1315084
0.3399699,-0.0885230,3.1000837,0.1983911
-0.0676719,-0.0308610,3.1007931,-0.2110950
0.0321129,-0.2879029,3.1123554,-0.1175412
0.0156280,-0.1516777,2.9643970,0.2347598
0.0724631,0.0617225,3.0496035,0.3073896
-0.2331213,-0.1594757,3.1011312,0.0441267
-0.0430484,-0.0795588,3.1869360,0.2670179
-0.0641014,0.0559266,3.2638531,-0.5821237
-0.1455251,-0.0163203,3.4105803,-0.1931791
0.0367459,0.3643917,3.1219745,-0.2788052
-0.2087293,-0.1543799,3.1436375,-0.3792555
-0.0429527,0.0269541,2.9287162,0.0795340
-0.0571857,-0.0326500,2.8700151,-0.2087783
0.1963970,-0.0934473,2.7756170,-0.0342860
-0.2518966,0.1067593,3.2373967,0.0035477
-0.0126298,0.1705660,3.0368359,0.3192790
-0.1018448,-0.0960350,3.2384875,-0.1247828
0.1912736,0.1420507,3.4285288,-0.2133477
0.2453085,-0.0804293,3.0753926,-0.0018955
-0.2816699,-0.0830021,2.6534659,-0.0147403
0.0559094,-0.3963115,3.2614500,0.1748106
-0.1210216,0.0689083,3.2929249,0.1030958
0.2930963,0.1327532,2.9134526,-0.0097204
-0.0173160,-0.0939122,3.0333004,0.2869536
-0.1938844,-0.3986582,3.0106515,-0.1328338
0.0759046,0.1791385,3.0033424,-0.2001640
0.0256208,-0.3341581,3.2369574,0.1330409
-0.3489252,0.0429605,3.0445276,-0.2147442
0.0160416,-0.2146952,3.3724046,0.1010329
0.1431772,0.1850725,2.9402800,0.3745606
```

```
0.0827173,-0.2512718,2.9955411,-0.1962611
-0.5058667,-0.1688560,2.8670899,-0.0878942
-0.0768706,0.0072491,3.1544000,-0.1706255
0.0229424,0.0134602,3.1263686,-0.1070018
-0.1085748,0.0479248,3.5272805,-0.1452584
-0.4176510,0.0124003,3.1428322,-0.0439331
0.1774355,0.0943218,3.0506109,0.1630511
0.0186550,-0.2177817,3.0244830,-0.1217582
-0.3327312,0.4323594,2.7908725,-0.0932434
0.0623278,-0.2532593,2.9083483,0.2256219
-0.3807703,0.0702986,3.2322388,-0.3731435
0.1037910,0.3688966,2.9170704,-0.2947049
0.1272420,0.0937511,2.9745850,0.0501791
-0.0027308,-0.2468777,3.3495527,-0.2062578
0.3091095,0.1537279,3.3094888,0.1587296
-0.1583278,0.0444451,3.2244866,-0.0543033
0.3935008,0.3120586,3.1160218,0.0948834
0.4537229,-0.0440953,3.3132946,-0.1390201
0.1219980,0.1297427,2.9467506,0.1721565
-0.1530108,0.0800774,2.9667254,-0.2012086
0.2829378,0.1401814,2.9788743,-0.0066642
0.1457605,0.1852873,3.2084314,-0.1585560
0.3603167,0.1551108,3.2276420,-0.0933915
-0.0866430,0.0482265,3.1238339,-0.2269347
0.2338768,-0.0360764,3.2042615,-0.0111483
-0.1666796,-0.2975622,3.1993875,-0.1521532
-0.1568801,-0.2356022,3.1156271,-0.0758954
-0.1151713,-0.0652865,2.8464950,-0.1081410
-0.2214626,0.1674129,2.8198504,-0.4196186
-0.1212625,0.1875964,3.0988535,-0.0444575
-0.5311547,-0.3596119,3.1978832,-0.2992852
0.0494399,0.0141164,3.0469677,0.4067464
0.0193488,0.2061676,3.3819292,-0.4865677
-0.4301699,0.2398461,3.1390890,0.0985958
-0.0681897,-0.0588242,3.4245929,-0.3068797
-0.2163202,-0.1768019,3.2869419,-0.1614941
0.2273849,-0.2930407,3.0831436,0.2123613
0.2202991,-0.2877622,3.0626309,0.3365812
-0.0092871,-0.0328053,3.3119508,0.4535608
0.1176982,-0.0174894,3.1624262,0.1988779
-0.2998236,0.1035697,3.3369632,-0.0919616
-0.1306507,-0.3066973,3.4153166,-0.1692908
-0.7201659,0.0784664,3.3996336,-0.1414291
-0.2937266,-0.0643808,3.0993272,0.0704777
-0.1672570,-0.1151412,2.9624049,-0.0024998
0.0151899,0.0916900,2.9986297,0.6406102
0.3349534,0.1226685,3.1548453,-0.1198144
-0.0346609,-0.0844746,3.2860483,-0.1026092
-0.1839327,0.3246719,3.4112611,0.1840697
0.1220840,-0.2366406,3.8346504,0.3090561
0.2795950,0.2255267,3.0197590,-0.0949184
-0.1142156,-0.4015067,3.1447004,0.0018792
```

E. Pendel_PC_4Obs_class.py

```
1   # import necessary libraries
########################################################################
########################
2   import dropbox
3   import sys
4   import numpy as np
5   import csv
6   import random
7   from random import seed
8   from random import random, randint
9   import tensorflow as tf
10  from tensorflow import keras
11  from tensorflow.keras import layers
12  import time
13  import datetime
14  from datetime import datetime
15  import matplotlib
16  import matplotlib.pyplot as plt
17  matplotlib.rc('figure', max_open_warning=0)
18  matplotlib.use('Agg')
19  import os
20  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # Keras Meldungen
ausschalten
21  import math
22  from dqn_PC_class import DQN
23  import collections
24  from typing import Deque
25  import random
26  import csv
27  #
========================================================================
========================================
28  PathDropbox = 'C:/Users/OptiPlex
7050/Dropbox/Apps/RaspiPendel/RaspiPendel/'
29  MODEL_PATH = "C:/Users/OptiPlex
7050/Desktop/temp_joel/dqn_new.h5"
30  DROPBOX_MODEL = "C:/Users/OptiPlex
7050/Desktop/temp_joel/model_new.h5"
31  TARGET_PATH = "C:/Users/OptiPlex
7050/Desktop/temp_joel/Target_new.h5"
32  FILE_PATH = "C:/Users/OptiPlex 7050/Desktop/temp_joel/"
33  #
========================================================================
==========
34  class Agent:
35      def __init__(self):
36          # DQN Env Variables
37          self.observations = 4
```

```
38          self.actions = 3
39          # DQN Agent Variables
40          self.lastExperience = np.zeros((1, 11))
41          self.experience = np.zeros((1, 11))
42          self.Imax = 1000
43          self.replay_buffer_size = 50_000
44          self.train_start = 1_000
45          self.memory: Deque =
collections.deque(maxlen=self.replay_buffer_size)
46          self.gamma = 0.99
47          self.alpha = 0.99
48          self.epsilon = 0.2
49          self.epsilon_min = 0.01
50          self.epsilon_decay = 0.995
51          # DQN Network Variables
52          self.state_shape = self.observations
53          self.learning_rate = 1e-3
54          self.dqn = DQN(
55              self.state_shape,
56              self.actions,
57              self.learning_rate
58          )
59          self.target_dqn = DQN(
60              self.state_shape,
61              self.actions,
62              self.learning_rate
63          )
64          self.target_dqn.update_model(self.dqn)
65          self.batch_size = 64
66          self.dqn.save_model(PathDropbox + 'model_new.h5')
67          # curves variable
68          self.FlagZenit = 0
69          self.IZenit = 0
70          self.rewardWithoutBoniValue = np.zeros((1, 1))
71          self.rewardWithoutBoniPlot = np.zeros((1, 1))
72          self.rewardWithoutBoniData = np.zeros((1, 1))
73          self.totalRewardValue = np.zeros((1, 1))
74          self.totalRewardPlot = np.zeros((1, 1))
75          self.averageRewardValue = np.zeros((1, 1))
76          self.averageRewardPlot = np.zeros((1, 1))
77          self.zenitCounter = 0
78          self.zenitValue = np.zeros((1, 1))
79          self.zenitPlot = np.zeros((1, 1))
80          self.firstTimestepAtZenitValue = np.zeros((1, 1))
81          self.firstTimestepAtZenitPlot = np.zeros((1, 1))
82      def loadExperienceList(self):
83          self.lastExperience = np.zeros((1, 11))
84          print('Überprüfen, ob der Raspi die Episode beendet
hat...')
85          while True:
86              try:
```

```
87                  if os.path.isfile(PathDropbox +
'MeldungRaspi.csv'):
88                      time.sleep(1)
89                      os.remove(PathDropbox + 'MeldungRaspi.csv')
90                      break
91              except FileNotFoundError:
92                  pass
93          print('MeldungRaspi gefunden')
94          # ------------------------------------------------------
----
95          # load last experience
96          self.lastExperience = open(PathDropbox +
'LetzteErfahrung.csv')
97
98          self.lastExperience = np.genfromtxt(self.lastExperience,
delimiter=",", skip_header=0)
99          # ------------------------------------------------------
-----------------------------------------
100         # save file "MeldungPC" on dropbox
101         try:
102             np.savetxt(PathDropbox + 'MeldungPC.csv', [1],
fmt='%i')
103             print('Meldung an Raspi gesendet')
104         except:
105             time.sleep(0.01)
106         # append experience List
107         if self.lastExperience.ndim <= 1 or
self.lastExperience.size <= 1:
108             self.lastExperience = self.lastExperience.reshape(-
1, 1)  ###<--------------make lastExperience ndim=2
109         try:
110             self.experience = np.append(self.experience,
self.lastExperience, axis=0)
111         except:
112             os.remove(PathDropbox + 'LetzteErfahrung.csv')
113             self.lastExperience = np.zeros((1, 11))
114             time.sleep(0.01)
115         np.savetxt('GesamteErfahrung.csv', self.experience,
delimiter=',', fmt='%1.7f')
116         print('Länge gesamte Erfahrung',
np.shape(self.experience)[0])
117         return self.lastExperience
118
119     def collectCurvesData(self, total_reward, Izenit,
zenitCounter, reawardWithoutBoni, average_reward):
120         self.totalRewardValue[0] = total_reward
121         self.totalRewardPlot = np.append(self.totalRewardPlot,
self.totalRewardValue, axis=0)
122         np.savetxt(FILE_PATH + "totalRewardListList.csv",
self.totalRewardPlot, fmt='%1.7f', delimiter=',')
123         self.firstTimestepAtZenitValue[0] = Izenit
```

```
124         self.firstTimestepAtZenitPlot =
np.append(self.firstTimestepAtZenitPlot,
self.firstTimestepAtZenitValue, axis=0)
125         np.savetxt(FILE_PATH + "firstTimestepAtZenitList.csv",
self.firstTimestepAtZenitPlot, fmt='%1.7f',
126                    delimiter=',')
127         self.zenitValue[0] = zenitCounter
128         self.zenitPlot = np.append(self.zenitPlot,
self.zenitValue, axis=0)
129         np.savetxt(FILE_PATH + "zenitCounterList.csv",
self.zenitPlot, fmt='%1.7f', delimiter=',')
130         self.rewardWithoutBoniValue[0] = reawardWithoutBoni
131         self.rewardWithoutBoniPlot =
np.append(self.rewardWithoutBoniPlot, self.rewardWithoutBoniValue,
axis=0)
132         np.savetxt(FILE_PATH + "rewardWithoutBoniList.csv",
self.rewardWithoutBoniPlot, fmt='%1.7f', delimiter=',')
133         self.averageRewardValue[0] = average_reward
134         self.averageRewardPlot =
np.append(self.averageRewardPlot, self.averageRewardValue, axis=0)
135         np.savetxt(FILE_PATH + "averageRewardList.csv",
self.averageRewardPlot, fmt='%1.7f', delimiter=',')
136
137     def plotcurvesOneGraph(self, totalRewardPlot: np.ndarray,
firstTimestepAtZenitPlot: np.ndarray, zenitPlot: np.array
138                            , rewardWithoutBoniPlot: np.array):
139         x = [i for i in range(totalRewardPlot.shape[0])]
140         y1 = totalRewardPlot
141         y2 = firstTimestepAtZenitPlot
142         y3 = zenitPlot
143         y4 = rewardWithoutBoniPlot
144         fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1)
145
fig.suptitle('totalReward_firstTimestepAtZenit_zenitCounter_rewardN
oBoni')
146         ax1.plot(x[1:], y1[1:], marker='o', color='b')
147         ax1.set_ylabel('totalReward')
148         ax2.plot(x[1:], y2[1:], marker='*', color='g')
149         ax2.set_ylabel('TimeZenit')
150         ax3.plot(x[1:], y3[1:], marker='p', color='r')
151         ax3.set_ylabel('zenitCount')
152         ax4.plot(x[1:], y4[1:], marker='+', color='m')
153         ax4.set_xlabel('episode')
154         ax4.set_ylabel('rWoBoni')
155         plt.savefig(FILE_PATH +
"Reward_firstTimestepAtZenit_zenitCounter_rewardNoBoni.png")
156         plt.close('all')
157         plt.clf()
158         plt.cla()
159
160     def plotcurves(self, totalRewardPlot: np.ndarray,
firstTimestepAtZenitPlot: np.ndarray, zenitPlot: np.ndarray,
```

```
161                     rewardWithoutBoniPlot: np.array,
averageRewardPlot: np.array):
162         x = [i for i in range(totalRewardPlot.shape[0])]
163         plt.figure(1)
164         plt.plot(x[1:], totalRewardPlot[1:], 'o-')
165         plt.title("total Reward vs. Episode")
166         plt.xlabel('episode')
167         plt.ylabel('totalReward')
168         plt.draw()
169         plt.savefig(FILE_PATH + "totalRewardPlot.png")
170         plt.pause(0.001)
171         plt.figure(2)
172         plt.plot(x[1:], firstTimestepAtZenitPlot[1:],
marker='*', color='g')
173         plt.title("1stTimeAtZenit vs. Episode")
174         plt.xlabel('episode')
175         plt.ylabel('1stTimeAtzenit')
176         plt.draw()
177         plt.savefig(FILE_PATH + "1stTimeAtzenitPlot.png")
178         plt.figure(3)
179         plt.plot(x[1:], zenitPlot[1:], marker='p', color='r')
180         plt.title("zenitCounter vs. Episode")
181         plt.xlabel('episode')
182         plt.ylabel('zenitCounter')
183         plt.draw()
184         plt.savefig(FILE_PATH + "zenitCounterPlot.png")
185         plt.pause(0.001)
186         plt.figure(4)
187         plt.plot(x[1:], rewardWithoutBoniPlot[1:], marker='+',
color='m')
188         plt.title("rewardWithoutBoni vs. Episode")
189         plt.xlabel('episode')
190         plt.ylabel('rewardWithoutBoni')
191         plt.draw()
192         plt.savefig(FILE_PATH + "rewardWithoutBoniPlot.png")
193         plt.pause(0.001)
194         plt.figure(5)
195         plt.plot(x[1:], averageRewardPlot[1:], marker='+',
color='m')
196         plt.title("averageReward vs. Episode")
197         plt.xlabel('episode')
198         plt.ylabel('averageReward')
199         plt.draw()
200         plt.savefig(FILE_PATH + "averageRewardPlot.png")
201         plt.pause(0.001)
202         plt.close('all')
203         plt.clf()
204         plt.cla()
205
206     def remember(self, state, action, reward, next_state, done):
207         self.memory.append((state, action, reward, next_state,
done))
```

```
208
209     def replay(self):
210         if len(self.memory) < self.train_start:
211             return
212         minibatch = random.sample(self.memory, self.batch_size)
213         states, actions, rewards, states_next, dones =
zip(*minibatch)
214         states = np.concatenate(states).astype(np.float32)
215         states_next =
np.concatenate(states_next).astype(np.float32)
216         q_values = self.dqn(states)
217         q_values_next = self.target_dqn(states_next)
218         for i in range(self.batch_size):
219             a = int(actions[i])
220             done = int(dones[i])
221             if done == 1:
222                 q_values[i][a] = rewards[i]
223             else:
224                 q_values[i][a] = (1 - self.alpha) *
(q_values[i][a]) + self.alpha * (rewards[i] + \
225
self.gamma * np.max(
226                                 q_values_next[i]))
227         with tf.device("/gpu:0"):
228             self.dqn.fit(states, q_values)
229
230     def train(self, num_episodes: int):
231         last_rewards: Deque = collections.deque(maxlen=5)
232         best_reward_mean = 0.0
233         for episode in range(1, num_episodes + 1):
234             total_reward = 0.0
235             rewardWithoutBoni = 0.0
236             print('Episode=', episode)
237             listOfstates = self.loadExperienceList()
238             for I in range(0, np.shape(listOfstates)[0]):
239                 state = listOfstates[I, 0:4]
240                 state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
241                 action = int(listOfstates[I, 4])
242                 reward = (listOfstates[I, 5]).astype(np.float32)
243                 next_state = listOfstates[I, 6:10]
244                 next_state = np.reshape(next_state, newshape=(1,
-1)).astype(np.float32)
245                 done = bool(int(listOfstates[I, 10]))
246                 if abs(next_state[0][0]) > 175 / 180 and
abs(next_state[0][1]) < 6 / 40:
247                     self.zenitCounter += 1
248                     if self.FlagZenit == 0:
249                         self.IZenit = I
250                         self.FlagZenit = 1
251                 rewardTheta = (-np.cos(next_state[0][0] * np.pi)
+ 1.0) / 2.0
```

```
252                 rewardX = np.cos((next_state[0][2]) * (np.pi /
2.0))
253                 rewardWithoutBoni += rewardTheta * rewardX
254                 self.remember(state, action, reward, next_state,
done)
255                 self.replay()
256                 total_reward += reward
257                 state = next_state
258                 if done:
259                     self.target_dqn.update_model(self.dqn)
260                     self.dqn.save_model(PathDropbox +
"model_new.h5")
261                     self.dqn.save_model(DROPBOX_MODEL)
262                     last_rewards.append(total_reward)
263                     current_reward_mean = np.mean(last_rewards)
264                     if current_reward_mean > best_reward_mean:
265                         best_reward_mean = current_reward_mean
266                         self.dqn.save_model(MODEL_PATH)
267                         self.target_dqn.save_model(TARGET_PATH)
268                         print(f"Episode: {episode} best_Reward:
{total_reward}")
269                     break
270
print('**************************************************************
**********************')
271             self.collectCurvesData(total_reward, self.IZenit,
self.zenitCounter, rewardWithoutBoni, total_reward / I)
272             self.FlagZenit = 0
273             self.IZenit = 0
274             self.zenitCounter = 0
275             self.plotcurvesOneGraph(self.totalRewardPlot,
self.firstTimestepAtZenitPlot, self.zenitPlot,
276                                     self.rewardWithoutBoniPlot)
277         self.dqn.save_model(PathDropbox + "model_new.h5")
278         self.dqn.save_model(MODEL_PATH)
279         self.target_dqn.save_model(TARGET_PATH)
280         self.plotcurves(self.totalRewardPlot,
self.firstTimestepAtZenitPlot, self.zenitPlot,
self.rewardWithoutBoniPlot,
281                         self.averageRewardPlot)
282
283 if __name__ == "__main__":
284     agent = Agent()
285     StartZeit = time.time()
286     agent.train(num_episodes=1000)
287     TrainingsDauer = (time.time() - StartZeit) / 60
288     print(f"Train time :{TrainingsDauer}")
```

F. Pendel_Raspi_4Obs_class.py

```
1   # import necessary libraries
#########################################################################
########################
2   import smbus  # todo check smbus vs smbus2
3   import time
4   import os
5   import sys
6   import RPi.GPIO as GPIO
7   import pigpio
8   import numpy as np
9   import scipy as sci
10  import scipy.interpolate
11  from scipy.interpolate import UnivariateSpline
12  import matplotlib
13  import matplotlib.pyplot as plt
14  from scipy.optimize import curve_fit, root, fsolve
15  import csv
16  import random
17  from random import seed
18  from random import random, randint
19  import xlsxwriter
20  import datetime
21  from datetime import datetime
22  sys.path.append('/usr/lib/python3.7/MyModules')
23  import PVS_Initialization as PVS
24  # import I2C_Calc as I2CC
25  # Definitions
#########################################################################
#####################################
26  I2C_Address_STM32_Motorcontroller = 0x0F  # 15
27  I2C_Address_STM32_Encoder_Axis = 0x0A  # 10
28  I2C_Address_STM32_Encoder_Pendulum = 0x08  # 8
29  IRecLCV = []
30  I2C_Motorcontroller_Request_Array = [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0]
31  I2C_Encoder_Axis_Request_Array = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
32  I2C_Encoder_Pendulum_Request_Array = [0, 0, 0, 0, 0, 0, 0, 0]
33  # Initialization
#########################################################################
###################################
34  # Reset µCs
#########################################################################
########################################
35  PVS.reset_µC()
36  # Reset Motor
#########################################################################
#####################################
37  PVS.reset_motor()
```

```
38  # setup I2C Bus and search for addresses
#####################################################################
############
39  print('I2C-Adresscheck in progress. Please wait.... ')
40  os.system('sudo i2cdetect -y 1 \n')
41  bus = smbus.SMBus(1)
42  # check I2C Adresses for match
#####################################################################
######################
43  PVS.I2C_match(bus, I2C_Address_STM32_Motorcontroller,
I2C_Address_STM32_Encoder_Axis,
44                I2C_Address_STM32_Encoder_Pendulum)
45  # Initialize I2C Slave Modes for µCs
#####################################################################
################
46  PVS.initialize_Slave_Mode(bus,
I2C_Address_STM32_Motorcontroller, I2C_Address_STM32_Encoder_Axis,
47                            I2C_Address_STM32_Encoder_Pendulum)
48  # Initialize homing Axis
#####################################################################
############################
49  Homing_Check = PVS.homing(bus,
I2C_Address_STM32_Motorcontroller)
50  # Initialize measuring Axis and position cart in the middle of
axis ##################################################
51  Measuring_Check = PVS.measure_Axis(bus,
I2C_Address_STM32_Motorcontroller)
52  # Zero Position of Axis
#####################################################################
############################
53  PVS.zero_axis_coordinates(bus, I2C_Address_STM32_Encoder_Axis)
54  # Initialize axis orientation test
#####################################################################
#################
55  Orientation_Check = PVS.axis_orientation_test(bus,
I2C_Address_STM32_Motorcontroller, I2C_Address_STM32_Encoder_Axis)
56  # Initialize software limit switch test
#####################################################################
############
57  Software_Limit_Switch_Check = PVS.software_limit_test(bus,
I2C_Address_STM32_Motorcontroller,
58
I2C_Address_STM32_Encoder_Axis)
59  # Zero Position of Pendulum
#####################################################################
########################
60  time.sleep(60)
61  PVS.zero_pendulum_coordinates(bus,
I2C_Address_STM32_Encoder_Pendulum)
62  # Test Encoder Pendulum
#####################################################################
############################
```

```
63  Encoder_Pendulum_Check = PVS.encoder_pendulum_test(bus,
I2C_Address_STM32_Motorcontroller,
64
I2C_Address_STM32_Encoder_Pendulum)
65  # Set Parameter Motorcontroller
########################################################################
####################
66  # PVS.set_Motorcontroller_Parameter()
67  # Final output before start
########################################################################
########################
68  if ((Homing_Check == True)
69          and (Measuring_Check == True)
70          and (Orientation_Check == True)
71          and (Software_Limit_Switch_Check == True)):
72      # and (Encoder_Pendulum_Check == True)):
73      print('All checks successful! Press any key to start
learning algorithm')
74      input()
75  else:
76      print('NOT all check were successful. Are you sure to
continue?\n')
77      print(Homing_Check, Measuring_Check, Orientation_Check,
Software_Limit_Switch_Check, Encoder_Pendulum_Check)
78      continue_condition = info_query()
79      if (continue_condition == True):
80          pass
81      else:
82          print('OK, system will shut down in 5 seconds.\nTry
check cycle again.')
83          time.sleep(5)
84          sys.exit()
85  # Random Test Loop
########################################################################
################################
86  '''uncoment this command to test PVS functions
87              for normal use or learning comment the following
lines out and proceed with your code afterwards'''
88
89  # PVS.set_speed_acc(bus, I2C_Address_STM32_Motorcontroller, 100,
45000)
90  # for i in range (0, 1):
91  #    C = PVS.rand_test_loop(bus,
I2C_Address_STM32_Motorcontroller, I2C_Address_STM32_Encoder_Axis,
I2C_Address_STM32_Encoder_Pendulum)
92  # PVS.save_to_logfile(C)
93  # print('saved to logfile')
94
95
96
97
```

```
98  # Here starts NN-Code
#####################################################################
###################
99  import dropbox
100 import os
101 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # Keras Meldungen
ausschalten
102 import tensorflow as tf
103 from tensorflow import keras
104 from tensorflow.keras import layers
105 import requests
106 from dqn_Raspi_class import DQN
107 from copy import deepcopy, copy
108 import collections
109 tf.enable_eager_execution()
110 #
=====================================================================
=======================================
111 PathCloud = '/RaspiPendel/'
112 PathHome = '/home/raspi3b+/Desktop/NN_transfer/PPBot_6/'  #
<<<<<------ DAS MUSS KONTROLLIERT WERDEN !!!
113 PATH_TRAINED_MODEL =
'/home/raspi3b+/Desktop/NN_transfer/PPBot_6/model_new.h5'
114 PATH_EVAL =
'/home/raspi3b+/Desktop/NN_transfer/PPBot_6/Evaluation/'
115 db = dropbox.Dropbox(ACCESSTOKEN')
116 #
=====================================================================
=======================================
117 class CartPole:
118     def __init__(self):
119         self.observations = 4
120         self.actions = 3
121         # self.model_new = model_new
122         self.E_Max = 1000  # 99999
123         self.I_Max = 1000  # 5000
124         self.epsilon = 0.2  # 0.2 if stating training
125         self.epsilon_min = 0.01
126         self.epsilon_decay = 0.995
127         self.LetzteErfahrung = np.zeros((1, 11))
128         self.ErfahrungValue = np.zeros((1, 11))
129         self.state = None
130         self.y_threshold = 390
131         self.u_max = 180
132         self.u_dot_threshold = 40
133         self.y_dot_max = 100
134         self.t = 0  # timestep
135         self.t_limit = 1000  # 1000
136         # curves variable
137         self.FlagZenit = 0
138         self.IZenit = 0
139         self.rewardWithoutBoniValue = np.zeros((1, 1))
```

```
140         self.rewardWithoutBoniPlot = np.zeros((1, 1))
141         self.rewardWithoutBoniData = np.zeros((1, 1))
142         self.totalRewardValue = np.zeros((1, 1))
143         self.totalRewardPlot = np.zeros((1, 1))
144 
145         self.averageRewardValue = np.zeros((1, 1))
146         self.averageRewardPlot = np.zeros((1, 1))
147         self.zenitCounter = 0
148         self.zenitValue = np.zeros((1, 1))
149         self.zenitPlot = np.zeros((1, 1))
150         self.firstTimestepAtZenitValue = np.zeros((1, 1))
151         self.firstTimestepAtZenitPlot = np.zeros((1, 1))
152         self.statisticRandomState = {}
153         # load model
154         Flag = 0
155         self.model_new = DQN(
156             state_shape=self.observations,
157             num_actions=self.actions,
158             learning_rate=0.001)
159         while Flag == 0:
160             try:
161                 db = dropbox.Dropbox('ACCESSTOKEN')
162                 db.files_download_to_file(PathHome +
'model_new.h5',
163                                           PathCloud +
'model_new.h5')
164                 self.model_new.load_model(PathHome +
'model_new.h5')
165                 Flag = 1
166             except requests.exceptions.ConnectionError:
167                 print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
168         print('Es läuft jetzt eine neue Lesson...')
169     def read_all(self):
170         bus.write_byte(I2C_Address_STM32_Encoder_Pendulum, 102)
171         I2C_Encoder_Pendulum_Request_Array =
bus.read_i2c_block_data(I2C_Address_STM32_Encoder_Pendulum, 102, 8)
172         u = PVS.rotpos(I2C_Encoder_Pendulum_Request_Array[0],
I2C_Encoder_Pendulum_Request_Array[1])
173         udot =
PVS.rotspeed(I2C_Encoder_Pendulum_Request_Array[6],
I2C_Encoder_Pendulum_Request_Array[5],
174 
I2C_Encoder_Pendulum_Request_Array[4],
I2C_Encoder_Pendulum_Request_Array[3],
175 
I2C_Encoder_Pendulum_Request_Array[2])
176 
177         bus.write_byte(I2C_Address_STM32_Encoder_Axis, 102)
178         I2C_Encoder_Axis_Request_Array =
bus.read_i2c_block_data(I2C_Address_STM32_Encoder_Axis, 102, 10)
```

```
179         y = PVS.linpos(I2C_Encoder_Axis_Request_Array[3],
I2C_Encoder_Axis_Request_Array[2],
180                        I2C_Encoder_Axis_Request_Array[1],
I2C_Encoder_Axis_Request_Array[0])
181
182         bus.write_byte(I2C_Address_STM32_Encoder_Axis, 102)
183         I2C_Encoder_Axis_Request_Array =
bus.read_i2c_block_data(I2C_Address_STM32_Encoder_Axis, 102, 10)
184         ydot = PVS.linspeed(I2C_Encoder_Axis_Request_Array[8],
I2C_Encoder_Axis_Request_Array[7],
185                             I2C_Encoder_Axis_Request_Array[6],
I2C_Encoder_Axis_Request_Array[5],
186                             I2C_Encoder_Axis_Request_Array[4])
187
188         return u, udot, y, ydot
189
190     def halt(self):
191         bus.write_byte(I2C_Address_STM32_Motorcontroller, 3)
192
193     def rechts(self):
194         bus.write_byte(I2C_Address_STM32_Motorcontroller, 4)
195
196     def links(self):
197         bus.write_byte(I2C_Address_STM32_Motorcontroller, 5)
198
199     def mitte(self):
200         bus.write_byte(I2C_Address_STM32_Motorcontroller, 8)
201
202     def reset(self):
203         self.mitte()
204         time.sleep(0.1)
205         Rechts = 0.2
206         Start1 = time.time()
207         dt1 = 0
208         while dt1 < Rechts:
209             self.rechts()
210             u, udot, y, ydot = self.read_all()
211             if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
212                 self.mitte()
213                 time.sleep(0.3)
214                 break
215             dt1 = time.time() - Start1
216         time.sleep(0.06)
217         Links = 0.2
218         Start2 = time.time()
219         dt2 = 0
220         while dt2 < Links:
221             self.links()
222             time.sleep(0.06)
223             u, udot, y, ydot = self.read_all()
```

```
224             if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
225                 self.mitte()
226                 time.sleep(0.3)
227                 break
228             dt2 = time.time() - Start2
229         u, udot, y, ydot = self.read_all()
230         self.state = (u, udot, y, ydot)
231         u_n, udot_n, y_n, ydot_n = copy(self.state)
232         obs = (u_n / self.u_max, udot_n / self.u_dot_threshold,
y_n / self.y_threshold, ydot_n / self.y_dot_max)
233         self.t = 0
234         return np.array(obs)
235
236     def get_action(self, state):
237         if np.random.rand() <= self.epsilon:
238             return np.random.randint(self.actions)
239         else:
240             return np.argmax(self.model_new(state))
241
242     def get_actionForPlay(self, state):
243         return np.argmax(self.model_new(state))
244
245     def actionHandler(self, action):
246         if action == 0:
247             self.rechts()
248             Pause = 0.1
249         elif action == 1:
250             self.halt()
251             Pause = 0.1
252         else:
253             self.links()
254             Pause = 0.1
255         # Aktionsdauer
256         Start = time.time()
257         dt = 0
258         while dt < Pause:
259             u, udot, y, ydot = self.read_all()
260             # Abbruchbedingungen
261             if abs(y) > self.y_threshold or abs(udot) >
self.u_dot_threshold:
262                 self.mitte()
263                 time.sleep(0.3)
264                 break
265             dt = time.time() - Start
266
267     def step(self, action):
268         Boni = 0.0
269         self.actionHandler(action)
270         u, udot, y, ydot = self.read_all()
271         self.state = (u, udot, y, ydot)
272         u_n, udot_n, y_n, ydot_n = copy(self.state)
```

```
273         obs = (u_n / self.u_max, udot_n / self.u_dot_threshold,
y_n / self.y_threshold, ydot_n / self.y_dot_max)
274         self.t += 1
275         if abs(obs[0]) > 0.97 and abs(obs[1]) < 6 /
self.u_dot_threshold:
276             Boni = 10.0
277         done = bool(
278             y < -self.y_threshold
279             or y > self.y_threshold
280             or abs(udot) > self.u_dot_threshold
281             or self.t >= self.t_limit
282         )
283         reward_theta = (-np.cos((u * np.pi) / 180) + 1.0) / 2.0
284         reward_x = np.cos((y / self.y_threshold) * (np.pi /
2.0))
285         reward = reward_theta * reward_x + Boni
286         return np.array(obs), reward, done
287
288     def experienceList(self, state, action, reward, next_state,
done):
289         self.ErfahrungValue[0, 0:4] = state
290         self.ErfahrungValue[0, 4] = action
291         self.ErfahrungValue[0, 5] = reward
292         self.ErfahrungValue[0, 6:10] = next_state
293         self.ErfahrungValue[0, 10] = done
294         self.LetzteErfahrung = np.append(self.LetzteErfahrung,
self.ErfahrungValue, axis=0)
295         if self.epsilon > self.epsilon_min:
296             self.epsilon *= self.epsilon_decay
297
298     def saveExperienceListOnDropox(self):
299         self.LetzteErfahrung = self.LetzteErfahrung[1:, :]
300         # Save Erfahrung on Dropbox
301         np.savetxt(PathHome + 'LetzteErfahrung.csv',
self.LetzteErfahrung, fmt='%1.7f', delimiter=',')
302         try:
303             f = open(PathHome + 'LetzteErfahrung.csv', 'rb')
304             upname = PathCloud + 'LetzteErfahrung.csv'
305             db.files_upload(f.read(), upname,
306
mode=dropbox.files.WriteMode("overwrite")
307                             )
308             f.close()
309         except requests.exceptions.ConnectionError:
310             print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
311             # send file "MeldungRaspi.csv" to Dropbox
312         print('Meldung an den PC, dass die Episode beendet
wurde...')
313         try:
314             np.savetxt(PathHome + 'MeldungRaspi.csv', [1],
fmt='%i')
```

```
315             print('Medlung an PC... ')
316             f = open(PathHome + 'MeldungRaspi.csv', 'rb')
317             upname = PathCloud + 'MeldungRaspi.csv'
318             db.files_upload(f.read(), upname,
319
mode=dropbox.files.WriteMode("overwrite")
320                             )
321             f.close()
322         except requests.exceptions.ConnectionError:
323             print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
324
325     def loadCurrentModel(self):
326         print('Überprüfen ob der PC das Training des neuronalen
Netzes beendet hat...')
327         Flag = 0
328         while Flag == 0:
329             try:
330                 List =
db.files_list_folder(path="/RaspiPendel/")
331                 for entrie in List.entries:
332                     if entrie.name == 'MeldungPC.csv':
333                         print('MeldungPC erhalten...')
334                         db.files_delete(PathCloud +
'MeldungPC.csv')
335                         Flag = 1
336             except requests.exceptions.ConnectionError:
337                 print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
338         print()
339         # check if file "MeldungPC.csv" is already deleted
340         Flag1 = 0
341         Flag2 = 0
342         while Flag1 == 0:
343             try:
344                 List =
db.files_list_folder(path="/RaspiPendel/")
345                 for entrie in List.entries:
346                     if entrie.name == 'MeldungPC.csv':
347                         print('MeldungPC  noch nicht
gelöscht...')
348                         Flag2 = 0
349                     else:
350                         Flag2 = 1
351                         Flag1 = 1
352                 if Flag2 == 1:
353                     print('MeldungPC  jetzt gelöscht...')
354             except requests.exceptions.ConnectionError:
355                 print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
356         # load current model
```

```
357         print('Laden des neu trainierten neuronalen Netzes vom
PC...')
358         Flag3 = 0
359         while Flag3 == 0:
360             try:
361                 db.files_download_to_file(PathHome +
'model_new.h5',
362                                           PathCloud +
'model_new.h5')
363                 self.model_new.load_model(PathHome +
'model_new.h5')
364                 Flag3 = 1
365             except requests.exceptions.ConnectionError:
366                 print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
367
368     def collectCurvesData(self, total_reward, Izenit,
zenitCounter, reawardWithoutBoni, average_reward):
369         self.totalRewardValue[0] = total_reward
370         self.totalRewardPlot = np.append(self.totalRewardPlot,
self.totalRewardValue, axis=0)
371         np.savetxt(PATH_EVAL + "totalRewardListList.csv",
self.totalRewardPlot, fmt='%1.7f', delimiter=',')
372         self.firstTimestepAtZenitValue[0] = Izenit
373         self.firstTimestepAtZenitPlot =
np.append(self.firstTimestepAtZenitPlot,
self.firstTimestepAtZenitValue, axis=0)
374         np.savetxt(PATH_EVAL + "firstTimestepAtZenitList.csv",
self.firstTimestepAtZenitPlot, fmt='%1.7f',
375                    delimiter=',')
376         self.zenitValue[0] = zenitCounter
377         self.zenitPlot = np.append(self.zenitPlot,
self.zenitValue, axis=0)
378         np.savetxt(PATH_EVAL + "zenitCounterList.csv",
self.zenitPlot, fmt='%1.7f', delimiter=',')
379         self.rewardWithoutBoniValue[0] = reawardWithoutBoni
380         self.rewardWithoutBoniPlot =
np.append(self.rewardWithoutBoniPlot, self.rewardWithoutBoniValue,
axis=0)
381         np.savetxt(PATH_EVAL + "rewardWithoutBoniList.csv",
self.rewardWithoutBoniPlot, fmt='%1.7f', delimiter=',')
382         self.averageRewardValue[0] = average_reward
383         self.averageRewardPlot =
np.append(self.averageRewardPlot, self.averageRewardValue, axis=0)
384         np.savetxt(PATH_EVAL + "averageRewardList.csv",
self.averageRewardPlot, fmt='%1.7f', delimiter=',')
385
386     def plotcurvesOneGraph(self, totalRewardPlot: np.ndarray,
firstTimestepAtZenitPlot: np.ndarray, zenitPlot: np.array
387                            , rewardWithoutBoniPlot: np.array):
388         x = [i for i in range(totalRewardPlot.shape[0])]
389         y1 = totalRewardPlot
```

```
390         y2 = firstTimestepAtZenitPlot
391         y3 = zenitPlot
392         y4 = rewardWithoutBoniPlot
393         fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1)
394
fig.suptitle('totalReward_firstTimestepAtZenit_zenitCountert_reward
NoBoni')
395         ax1.plot(x[1:], y1[1:], marker='o', color='b')
396         ax1.set_ylabel('totalReward')
397         ax2.plot(x[1:], y2[1:], marker='*', color='g')
398         ax2.set_ylabel('TimeZenit')
399         ax3.plot(x[1:], y3[1:], marker='p', color='r')
400         ax3.set_ylabel('zenitCount')
401         ax4.plot(x[1:], y4[1:], marker='+', color='m')
402         ax4.set_xlabel('episode')
403         ax4.set_ylabel('rWoBoni')
404         plt.savefig(PATH_EVAL +
"Reward_firstTimestepAtZenit_zenitCounter_rewardNoBoni.png")
405         plt.close('all')
406         plt.clf()
407         plt.cla()
408
409     def plotcurves(self, totalRewardPlot: np.ndarray,
firstTimestepAtZenitPlot: np.ndarray, zenitPlot: np.ndarray,
410                    rewardWithoutBoniPlot: np.array,
averageRewardPlot: np.array):
411         x = [i for i in range(totalRewardPlot.shape[0])]
412         plt.figure(1)
413         plt.plot(x[1:], totalRewardPlot[1:], 'o-')
414         plt.title("total Reward vs. Episode")
415         plt.xlabel('episode')
416         plt.ylabel('totalReward')
417         plt.draw()
418         plt.savefig(PATH_EVAL + "totalRewardPlot.png")
419         plt.pause(0.001)
420         plt.figure(2)
421         plt.plot(x[1:], firstTimestepAtZenitPlot[1:],
marker='*', color='g')
422         plt.title("1stTimeAtZenit vs. Episode")
423         plt.xlabel('episode')
424         plt.ylabel('1stTimeAtzenit')
425         plt.draw()
426         plt.savefig(PATH_EVAL + "1stTimeAtzenitPlot.png")
427         plt.figure(3)
428         plt.plot(x[1:], zenitPlot[1:], marker='p', color='r')
429         plt.title("zenitCounter vs. Episode")
430         plt.xlabel('episode')
431         plt.ylabel('zenitCounter')
432         plt.draw()
433         plt.savefig(PATH_EVAL + "zenitCounterPlot.png")
434         plt.pause(0.001)
435         plt.figure(4)
```

```
436         plt.plot(x[1:], rewardWithoutBoniPlot[1:], marker='+',
color='m')
437         plt.title("rewardWithoutBoni vs. Episode")
438         plt.xlabel('episode')
439         plt.ylabel('rewardWithoutBoni')
440         plt.draw()
441         plt.savefig(PATH_EVAL + "rewardWithoutBoniPlot.png")
442         plt.pause(0.001)
443         plt.figure(5)
444         plt.plot(x[1:], averageRewardPlot[1:], marker='+',
color='m')
445         plt.title("averageReward vs. Episode")
446         plt.xlabel('episode')
447         plt.ylabel('averageReward')
448         plt.draw()
449         plt.savefig(PATH_EVAL + "averageRewardPlot.png")
450         plt.pause(0.001)
451         plt.close('all')
452         plt.clf()
453         plt.cla()
454
455     def collectExperience(self, num_episodes):
456         for E in range(1, num_episodes + 1):
457             total_reward = 0.0
458             print('Es bgeinnt jetzt eine neue Lesson...')
459
print('**************************************************************
**********************')
460             state = self.reset()
461             state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
462             for I in range(1, self.I_Max + 1):  # Iteration
463                 action = self.get_action(state)
464                 next_state, reward, done = self.step(action)
465                 next_state = np.reshape(next_state, newshape=(1,
-1)).astype(np.float32)
466                 self.experienceList(state, action, reward,
next_state, done)
467                 total_reward += reward
468                 state = next_state
469                 if done:
470                     break
471             print('Episode:{} Reward:{}'.format(E,
total_reward))
472             self.mitte()
473             time.sleep(120)
474             self.saveExperienceListOnDropox()
475             self.LetzteErfahrung = np.zeros((1, 11))
476             self.loadCurrentModel()
477             print('Neuronales Netz wird trainiert ...')
478         GPIO.cleanup()
479
```

```
480     def dictWriting(self, my_dict):
481         # my_dict = {'1': 'aaa', '2': 'bbb', '3': 'ccc'}
482         with open('daempfung.csv', 'w') as f:
483             for key in my_dict.keys():
484                 f.write("%s,%s\n" % (key, my_dict[key]))
485 
486     def pendelParameter(self, num_episodes):
487         for E in range(1, num_episodes + 1):
488             my_dict = {}
489             state = self.reset()
490             Start1 = time.time()
491             dt1 = 0
492             while dt1 < 100:
493                 self.halt()
494                 u, udot, y, ydot = self.read_all()
495                 my_dict[dt1] = [u, y]
496                 dt1 = time.time() - Start1
497             self.dictWriting(my_dict)
498             self.mitte()
499             time.sleep(0.3)
500         GPIO.cleanup()
501 
502     def getAVS(self,num_episodes):
503         for E in range(1,num_episodes+1):
504             my_dict={}
505             Start1=time.time()
506             dt1=0
507             while dt1< 6:
508                 self.rechts()
509                 u,udot,y,ydot = self.read_all()
510                 if (abs(y)>self.y_threshold or
abs(udot)>self.u_dot_threshold):
511                     self.mitte()
512                     time.sleep(0.3)
513                     break
514                 my_dict[dt1] =[u,y]
515                 dt1=time.time()-Start1
516             self.dictWriting(my_dict)
517             self.mitte()
518             time.sleep(0.3)
519 
520     def statisticEvaluateDqn(self, dictEvaluateDqn):
521         with open('statisticrandomState.csv', 'w') as f:
522             for key in dictEvaluateDqn.keys():
523                 f.write("%s,%s\n" % (key + ":",
dictEvaluateDqn[key]))
524 
525     def evaluateDqn(self):
526         self.model_new.load_model(PATH_TRAINED_MODEL)
527         episodes = 49
528         listTotalReward = []
529         listZenitCounter = []
```

```
530         listZenitTimestep = []
531         listRewardNoBoni = []
532         dictvariable = ["totalReward", "zenitTimestep",
"zenitCounter", "rewardNoBoni"]
533         for episode in range(1, episodes + 1):
534             timestepAtzenit = 0
535             state = self.reset()
536             state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
537             score = 0.0
538             rewardWithoutBoni = 0.0
539             for I in range(1, self.I_Max + 1):
540                 timestepAtzenit += 1
541                 action = self.get_actionForPlay(state)
542                 next_state, reward, done = self.step(action)
543                 next_state = np.reshape(next_state, newshape=(1,
-1)).astype(np.float32)
544                 if abs(next_state[0][0]) > 175 / 180 and
abs(next_state[0][1]) < 6 / 40:
545                     self.zenitCounter += 1
546                     if self.FlagZenit == 0:
547                         self.IZenit = timestepAtzenit
548                         self.FlagZenit = 1
549                 rewardTheta = (-np.cos(next_state[0][0] * np.pi)
+ 1.0) / 2.0
550                 rewardX = np.cos((next_state[0][2]) * (np.pi /
2.0))
551                 rewardWithoutBoni += rewardTheta * rewardX
552                 score += reward
553                 state = next_state
554                 if done:
555                     listTotalReward.append(score)
556                     listZenitCounter.append(self.zenitCounter)
557                     listZenitTimestep.append(self.IZenit)
558                     listRewardNoBoni.append(rewardWithoutBoni)
559                     print('Episode:{} Score:{}'.format(episode,
score))
560                     break
561             self.collectCurvesData(score, self.IZenit,
self.zenitCounter, rewardWithoutBoni, score / timestepAtzenit)
562             self.FlagZenit = 0
563             self.IZenit = 0
564             self.zenitCounter = 0
565             self.rechts()
566             time.sleep(0.1)
567             self.mitte()
568             time.sleep(90)
569         self.statisticRandomState[dictvariable[0]] =
{np.mean(listTotalReward), np.std(listTotalReward)}
570         self.statisticRandomState[dictvariable[1]] =
{np.mean(listZenitTimestep), np.std(listZenitTimestep)}
```

```
571         self.statisticRandomState[dictvariable[2]] =
{np.mean(listZenitCounter), np.std(listZenitCounter)}
572         self.statisticRandomState[dictvariable[3]] =
{np.mean(listRewardNoBoni), np.std(listRewardNoBoni)}
573         self.statisticEvaluateDqn(self.statisticRandomState)
574         self.plotcurvesOneGraph(self.totalRewardPlot,
self.firstTimestepAtZenitPlot, self.zenitPlot,
575                                 self.rewardWithoutBoniPlot)
576         self.plotcurves(self.totalRewardPlot,
self.firstTimestepAtZenitPlot, self.zenitPlot,
self.rewardWithoutBoniPlot,
577                         self.averageRewardPlot)
578         print(f"meanReward:{np.mean(listTotalReward)},
stdReward: {np.std(listTotalReward)}")
579         print(f"mean1stTimeAtZenit:{np.mean(listZenitTimestep)},
stdTimeAtZenit: {np.std(listZenitTimestep)}")
580         print(f"meanZeniCounter:{np.mean(listZenitCounter)},
stdzeniCounter: {np.std(listTotalReward)}")
581         print(f"meanRewNoBoni:{np.mean(listRewardNoBoni)},
stdzeniCounter: {np.std(listRewardNoBoni)}")
582         GPIO.cleanup()
583
584     def playEpisode(self, num_episodes: int):
585         # load current model
586         print('Laden des neu trainierten neuronalen Netzes vom
PC...')
587         Flag = 0
588         while Flag == 0:
589             try:
590                 db.files_download_to_file(PathHome +
'model_new.h5',
591                                           PathCloud +
'model_new.h5')
592                 self.model_new.load_model(PathHome +
'model_new.h5')
593                 Flag = 1
594             except requests.exceptions.ConnectionError:
595                 print('Dropbox nicht erreichbar. Bitte
Internetverbindung prüfen')
596         for episode in range(1, num_episodes + 1):
597             total_reward = 0.0
598             state = self.reset()
599             state = np.reshape(state, newshape=(1, -
1)).astype(np.float32)
600             for I in range(1, self.I_Max + 1):
601                 action = self.get_actionForPlay(state)
602                 next_state, reward, done = self.step(action)
603                 next_state = np.reshape(next_state, newshape=(1,
-1)).astype(np.float32)
604                 total_reward += reward
605                 state = next_state
606                 if done:
```

```
607                     print('Episode:{} Reward:{}'.format(episode,
total_reward))
608                     self.mitte()
609                     time.sleep(3)
610                     break
611         GPIO.cleanup()
612
613 if __name__ == "__main__":
614     env = CartPole()
615     # uncomment the line below for damping extimation
616     # env.pendelParameter(1)
617     StartZeit = time.time()
618     env.collectExperience(num_episodes=1000)
619     TrainingsDauer = (time.time() - StartZeit) / 60
620     print(f"Train time :{TrainingsDauer}")
621     # env.playEpisode(10)
622     env.evaluateDqn()
```

G. Simulation: Parameter Einstellung

run	Einstellung	result: total_reaward		Mean
		agent1 (400 episode)	agent2 (400 episode)	
bb00	x_threshold=40, alpha=gamma=0.99 Epsilon_initial=0.2 Dämpfung=0.005 batch_size= 32 Aktivierungsfukction: **RELU**	785	735	760
bb01	x_threshold=40 alpha=gamma=0.99 Epsilon_initial=0.2 Dämpfung=0.005 batch_size= 32 Aktivierungsfunktion: **Tanh**	706	864	785
bb02	x_threshold=40 alpha=gamma=0.99 Epsilon_initial=0.2 Dämpfung=0.005 **batch_size= 64** Aktivierungsfunktion: **RELU**	816	900	858
bb03	x_threshold=40 alpha=gamma=0.99 Epsilon_initial=0.2 Dämpfung=0.005 **batch_size= 64** **Aktivierungsfunktion: Tanh**	921	905	913

H. Versuchsergebnisse

dqnAgent Parameter	Ergebnis (mean, std) dqn Netz			
	totalReward	zenitCounter	1stTimeAtZenit	Reward ohne Boni
Epsilon=1, batch_size=64, Aktivierungsfunktion Tanh, Boni=10	Mean=7930, std=3117	Mean=726, std=288	Mean=99, std=34	Mean=659, std=255
Epsilon=0.2, batch_size=64, Aktivierungsfunktion Tanh, Boni=10	**Mean=9024, std=719**	**Mean=822, std=70**	**Mean=109, std=9**	**Mean=790, std=98**
Epsilon=0.2, batch_size=64, Aktivierungsfunktion Tanh, **Boni=1**	Mean=1474, std=222	Mean=711, std=128	Mean=125, std=13	Mean=758, std=102
Epsilon=0.2, batch_size=64, Aktivierungsfunktion Tanh, **Boni=0**	Mean=604, std=175	Mean=3, std=2	Mean=60, std=59	Mean=604, std=175
Epsilon=0.2, **batch_size=32**, Aktivierungsfunktion Tanh, Boni=10	Mean=4383, std=3901	Mean=400, std=360	Mean=77, std=65	Mean=363, std=331
Epsilon=1, batch_size=64, Aktivierungsfunktion **ReLU**, Boni=10	Mean=8963, std=1300	Mean=819, std=120	Mean=121, std=6	Mean=763, std=139

11Literaturverzeichnis

[1] Wikipedia, 30 10 2020. [Online]. Available: https://de.wikipedia.org/wiki/Best%C3%A4rkendes_Lernen. [Zugriff am 28 08 2021].

[2] Yendoukon Nayante, „Reinforcement Learning am simulierten Inverted Pendulum unter Anwendung verschiedener Algorithmen," 12 09 2021. [Online]. Available: https://github.com/barkname/Praxisprojektbericht/blob/master/Praxisprojektbericht_Nayante_Rev01.pdf. [Zugriff am 12 09 2021].

[3] C. Daniel Freeman and Luke Metz and David Ha, „Learning to Predict Without Looking Ahead: World Models Without Forward Prediction," 2019.

[4] Riedmiller, M., „Neural reinforcement learning to swing-up and balance a real pole," in *In 2005 IEEE International Conference on Systems, Man and Cybernetics*, 2005.

[5] Anderson, C. W., Lee, M., & Elliott, D. L., „Faster reinforcement learning after pretraining deep networks to predict state dynamics," in *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[6] Manrique Escobar, C. A., Pappalardo, C. M., & Guida, D, „A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole," Applied Sciences, 10(24), 9013, 2020.

[7] P. D.-I. H. Schwarz. [Online]. Available: https://www.uni-due.de/imperia/md/content/srs/forschung/msrt_paper/1994/fb07-94.pdf. [Zugriff am 03 September 2021].

[8] REX Controls, 18 Januar 2016. [Online]. Available: https://www.rexcontrols.com/oldweb/rexcontrols.com/articles/control-algorithm-fights-gravity. [Zugriff am 03 September 2021].

[9] Brockman, Greg and Cheung, Vicki and Pettersson, Ludwig and Schneider, Jonas and Schulman, John and Tang, Jie and Zaremba, Wojciech, „Openai gym," 2016. [Online]. Available: arXiv preprint arXiv:1606.01540. [Zugriff am 27 September 2021].

[10] T. München. [Online]. Available: https://www5.in.tum.de/wiki/index.php/Pendulum_Project.

[11] Dropbox, „Dropbox," 2014. [Online]. Available: http://www. dropbox. com.. [Zugriff am 27 September 2021].

[12] „Wikipedia," 31 01 2021. [Online]. Available: https://de.wikipedia.org/wiki/Konstruktoren_und_Destruktoren. [Zugriff am 08 09 2021].

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Gummersbach, den 16. September 2021

Unterschrift

Yendoukon Nayante