

HEILBRONN UNIVERSITY

## MASTER PROJECT

# Weiterentwicklung eines autonomen Airhockestisch mithilfe von Deep Reinforcement Learningmethoden

*Denise Baumann, Martin Haag*

supervised by

Prof. Dr. -Ing. Nicolaj C. Stache  
&  
Pascal Graph

April 30, 2022

MASTER PROJET

**Weiterentwicklung eines autonomen  
Airhockestisch mithilfe von Deep  
Reinforcement Learningmethoden**

at



Author:	Denise Baumann, Martin Haag
Matriculation Number:	210908, 194980
Course of Studies:	MAS, MMR
Supervisor:	Prof. Dr. -Ing. Nicolaj C. Stache & Pascal Graph
Time Frame:	Summer Semester / Winter Semester

# Contents

<b>1 Einleitung</b>	<b>1</b>
1.1 Vorstellung des Projekts . . . . .	1
1.2 Aufgabenstellung . . . . .	1
1.3 Zielsetzung . . . . .	2
1.4 Vorgehensweise . . . . .	2
<b>2 Reinforcement Learning</b>	<b>3</b>
2.1 Grundlagen . . . . .	3
2.2 Reinforcement Learning Algorithmus . . . . .	6
2.2.1 Proximal Policy Optimization (PPO) . . . . .	6
2.2.2 Soft Actor-Critic (SAC) . . . . .	7
<b>3 Hardwareanalyse</b>	<b>8</b>
3.1 Aufbau . . . . .	8
3.2 Komponentenaustausch . . . . .	10
<b>4 Simulationsumgebung</b>	<b>16</b>
4.1 Auswahl der Umgebung . . . . .	16
4.2 Unity . . . . .	16
4.2.1 Kurze Einführung . . . . .	17
4.2.2 Airhockey Projekt . . . . .	18
<b>5 Rewards und Training</b>	<b>22</b>
5.1 Rewards und Umgebungsparameter . . . . .	22
5.2 Spielmodi . . . . .	24
5.3 Trainingsstrategie und Zwischenergebnisse . . . . .	25
5.3.1 Stufenplan . . . . .	26
5.3.2 Netzwerkauswahl und Zwischenergebnisse . . . . .	27
5.4 Trainingsverlauf und Ergebnis . . . . .	33
5.4.1 Gegnerwahl . . . . .	33
5.4.2 Qualitätskriterien . . . . .	34

## CONTENTS

---

<b>6 Realer Demonstrator</b>	<b>35</b>
6.1 Geschwindigkeitsmessungen . . . . .	35
6.2 Benutzeroberfläche . . . . .	38
6.2.1 Ansicht der Oberfläche . . . . .	38
6.2.2 Programmablauf der GUI-Anwendung . . . . .	40
6.3 Kalibrierung . . . . .	44
6.4 Puck und Robotererkennung . . . . .	46
6.5 Spielverhalten . . . . .	49
<b>7 Fazit</b>	<b>50</b>

# 1 | Einleitung

In Einleitungskapitel werden die Rahmenbedingungen des Projekts erläutert. In diesem Zuge sollen die Ausgangssituation und das Ziel genauer behandelt werden.

## 1.1 Vorstellung des Projekts

Airhockey ist ein hochdynamisches Geschicklichkeitsspiel, bei dem zwei Personen gegeneinander antreten. Hierfür sind die Spieler mit jeweils einem Pusher ausgestattet. Ziel des Spiels ist es, den auf einem Luftfilm gleitenden Puck im gegnerischen Tor zu versenken. Durch die hohen Geschwindigkeiten, welche sowohl der Puck als auch die Pusher erreichen, besteht ein immenser Anspruch an die Mechanik sowie an die informationsverarbeitenden Systeme des gesamten Roboters. Um das Potenzial künstlicher Intelligenz anhand eines anschaulichen Demonstrators aufzuzeigen, möchte das Zentrum für maschinelles Lernen (ZML) in Heilbronn einen autonomen Air-Hockey-Roboter entwickeln. Dieser soll mithilfe eines Reinforcement Learning Algorithmus selbstständig das Air-Hockeyspiel erlernen und im Spiel gegen einen Menschen gewinnen. Bei dieser Form des Machine Learning muss ein Agent mit einer unbekannten Umgebung interagieren und daraus Erkenntnisse erzielen, also lernen. Das Lernen erfolgt auf der Basis von Belohnungen und Bestrafungen. Jede Aktion, die der Agent ausführt, wird dahingehend bewertet, ob sie im weiteren Verlauf zum Ziel (hier: ein Tor zu erzielen) oder im Gegensatz dazu zu einem Rückschlag (hier: ein Gegentor) führt.

## 1.2 Aufgabenstellung

Am Zentrum für maschinelles Lernen (ZML) wurde bereits in einer vorangegangenen Arbeit an einem selbst spielenden Airhockeytisch gearbeitet. Dazu wurde bereits ein Airhockeytisch mit der benötigten Hardware ausgestattet. Die Kinematik basiert dabei auf der des Roboters von jjRobots [4]. Des Weiteren wurde darin schon, in der Simulationsumgebung Unity, der Grundstein für das Training eines Reinforcement Learning Algorithmus gelegt. Eine Teilaufgabe ist nun, das Training des Agenten weiter voranzutreiben.

Zusätzlich zum Softwarebereich soll auch die vorhandene Hardware gründlich

überprüft werden und bei Bedarf die Komponenten ausgetauscht werden. Für das abschließende Testspiel am realen Aufbau muss außerdem noch eine Bildverarbeitung implementiert werden.

### 1.3 Zielsetzung

Ein Ziel dieses Masterprojekts ist das erfolgreiche Training eines Agenten, sodass dieser in der Lage ist, gegen einen menschlichen Spieler anzutreten und diesen auch zu besiegen. Ein weiteres Ziel ist die Übertragung der aus der Simulation gewonnenen Ergebnisse auf den realen Airhockeytisch.

### 1.4 Vorgehensweise

Zuerst werden die Grundlagen für das Reinforcement Learning gelegt. Anschließend wird auf die genaueren Gegebenheiten vor Ort eingegangen und eine umfangreiche Hardwareanalyse durchgeführt. Anschließend wird die Simulationsumgebung erläutert, in der die Umsetzung von diesem Projekt realisiert wird. Bevor aber ein Spiel am realen Demonstrator möglich ist, muss der Software Agent erst in dieser Simulationsumgebung trainiert werden. Bevor ein Überblick über den aktuellen Stand des Hardweraufbaus erarbeitet werden kann, werden zunächst projektspezifische Grundlagen geklärt. Abschließend erfolgt eine Evaluation der Ergebnisse.

## 2 | Reinforcement Learning

In diesem Kapitel wird ein erster Einblick in das Reinforcement Learning gegeben, welches ein Teilbereich des Machine Learning ist. Des Weiteren wird genauer auf die verwendeten Netzwerkarchitekturen eingegangen.

### 2.1 Grundlagen

Machine Learning allgemein ist ein Teilbereich von Artificial Intelligence (AI). Im Zusammenhang mit Daten und Algorithmen wird hierbei versucht, das menschliche Verhalten zu imitieren und zu verbessern [2]. Ein typisches Beispiel, das im Zusammenhang mit AI zurzeit sehr präsent ist, ist das autonome Fahren.

Das Machine Learning lässt sich in drei Teilbereiche untergliedern:

- Supervised Learning

Supervised Learning ist die in Forschungsarbeiten laut [10] meist verwendete Trainingsart im Bereich des Machine Learning. Bei dieser Art erfolgt das Lernen anhand von ausgewählten Beispielen, das von einem erfahrenen und fachkundigen Betreuer beaufsichtigt wird. Für jede Situation aus dieser Beispielmenge gibt es eine korrekte Aktion, die der Agent ausführen soll. Das Ziel ist es, diese Entscheidungen zu verallgemeinern und auf andere Situationen auszuweiten. Diese Form des Lernens ist aber nicht geeignet für das Lösen von interaktiven Problemen [10].

- Unsupervised Learning

Beim Unsupervised Learning geht es darum, Strukturen zu finden und diese in verschiedene Cluster einzufügen. Diese Algorithmen können ohne ein menschliches Eingreifen versteckte Muster entdecken oder Datengruppierungen durchführen. Deshalb ist diese Form des Lernens zum Beispiel ideal für die Bilderkennung [3].

- Reinforcement Learning

Beim Reinforcement Learning wird im Gegensatz zu den oben genannten Lernverfahren nicht auf Grundlage von Datensätzen, sondern mittels des

so genannten "trial and error" Verfahrens, also durch ausprobieren und scheitern, trainiert. Der Anreiz zum Lernen wird durch Belohnungen erzielt, denn jede mögliche Action des Agenten wird bewertet und anschließend je nachdem belohnt oder bestraft. Es wird also keine Lösung für eine bestimmte Situation vorgegeben, sondern der Agent muss selbst herausfinden, womit er die größte Belohnung erzielen kann. Das Reinforcement Learning bietet eine Vielzahl von Einsatzgebieten, wie zum Beispiel im Gamingbereich oder auch in der Robotik.

Das Prinzip eines Reinforcement Learning Algorithmus ist in der nachfolgenden Abbildung 2.1 gezeigt.

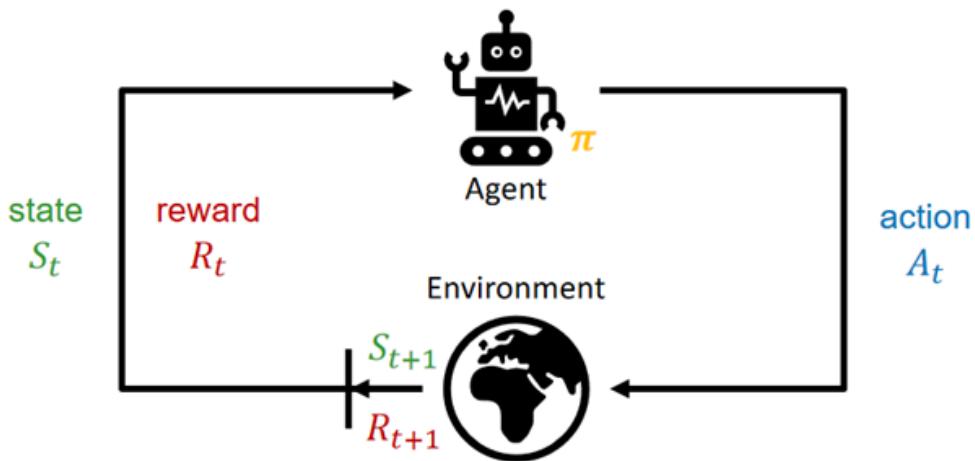


Figure 2.1: Übersicht des Reinforcement Learning Prinzips

- Agent

Im Agent verbirgt sich der Algorithmus der AI. Er empfängt den aktuellen State und den Reward aus der vorangegangenen Action und entscheidet unter Berücksichtigung der Policy, welche Action nun die beste Lösung im Verlauf der Episode bringen wird. Das Ziel des Agenten ist, einen möglichst großen Reward zu erzielen. Bei diesem Projekt ist der Agent der Pusher des Roboters.

- Environment

Das Environment zeigt die Gegebenheiten, auf deren Grundlage das Problem gelöst werden soll. Es gibt dem Agenten den Reward aus seiner vorangegangenen Action zurück. In diesem Projekt ist das Environment ein Airhockeytisch.

- Action

Eine Action wird vom Agenten zum Environment geschickt. Dort wird

anschließend diese dann ausgeführt. In diesem Projekt bilden die neun Richtungsmöglichkeiten die Menge aller möglichen Actions.

item State

Ein State ist der aktuelle Zustand des Spiels, nachdem eine Action im Environment durchgeführt wurde. Der State in diesem Projekt wird über die Kamera aufgenommen und weitergegeben.

- Reward

Mithilfe von Rewards wird der Agenten trainiert. Zu jedem Zeitpunkt gibt das Environment an den Agenten diesen Reward zurück. Ziel des Agenten ist es, den größtmöglichen Reward am Episodenende zu erreichen. Die Rewards in diesem Projekt werden genauer im Kapitel 5.1 erläutert.

- Policy  $\pi$

Die Policy bestimmt das Verhalten des Agenten. Es nimmt den aktuellen Zustand des Environments auf und wählt die darauffolgende Action aus.

- Episode

Eine Episode ist ein vorgeschriebener Zeitraum, nach dem das Environment wieder in den Ursprungszustand zurückgesetzt wird. Sie können entweder nach einer bestimmten Zeit terminiert werden oder aber abhängig von einem State beendet werden. Bei diesem Projekt zum Beispiel endet eine Episode unter anderem, wenn ein Tor erzielt wurde. Danach wird das Environment wieder auf die Ausgangslage zurückversetzt und es kann weiter trainiert werden. Der Agent behält jedoch seinen Trainingsfortschritt aus den vorangegangenen Episoden bei.

Die Abfolge von State (S), Reward (R) und Action (A) ist zu Beginn wie folgt festgelegt (Der Index gibt an, um welchen Zeitschritt es sich handelt):  $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$

Zu Beginn beeinflusst das Verhalten des Agents nur der aktuelle State  $S_0$ . Unter Berücksichtigung der Policy führt der Agent eine Action  $A_0$  zum State  $S_0$  durch. Das Environment führt nun eine Beurteilung des daraus resultierenden neuen State  $S_1$  durch und gibt dem Agent dementsprechend einen neuen Reward  $R_1$  zurück. Diese Schleife wird solange durchgegangen, bis das Ende der Episode erreicht wurde.

## 2.2 Reinforcement Learning Algorithmus

Die Grundlage des Reinforcement Learning ist die einfache mathematische Beschreibung des Problems als Markow Decision Process. Hierbei werden nur der State und die Action, welche unmittelbar zuvor aufgetreten sind, berücksichtigt. Ein Reinforcement Algorithmus kann in zwei Bereiche aufgeteilt werden [6]:

- Model-Based

Der Hauptbestandteil der Model-Based Methode ist die Planung einer Lösung des Problems. Der Agent analysiert das Environment, versucht es zu verstehen und dementsprechend eine Action auszuwählen.

- Model-Free

Bei der Model-Free Methode wird nicht das Modell des Environments genau analysiert und erlernt, sondern das Lernen erfolgt nur durch Probieren.

Des Weiteren können die Methoden dahingehend unterschieden werden, ob sie On-Policy oder Off-Policy sind. Der Unterschied ist, dass bei Off-Policy nur der nächste State berücksichtigt wird, während bei On-Policy auch noch die aktuelle Action in den Entscheidungsprozess mit einfließt [9]. Des Weiteren ist es bei Off-Policy Algorithmen möglich, aus den Erfahrungen, die in der Vergangenheit gesammelt wurden, zu lernen [1].

Es existiert eine Vielzahl an Algorithmen. Jeden einzeln zu beschreiben würde den Rahmen dieser Arbeit sprengen. Deshalb werden im Folgenden nur zwei Beispiele gegeben. Da später das Training mit dem Proximal Policy Optimization (PPO) und dem Soft Actor-Critic (SAC) durchgeführt wird (vgl. Kapitel 5.3.2), werden diese beiden Netzwerke nun genauer erläutert.

### 2.2.1 Proximal Policy Optimization (PPO)

Da die Algorithmen der Reinforcement Learning Methoden schwer zu handhaben sind und sehr aufwendig in der Abstimmung sind, wurde der PPO Algorithmus entwickelt. Das Prinzip hierbei ist, möglichst ein Gleichgewicht zwischen einer einfachen Implementierung, einer einfachen Abstimmung und der Komplexität der Beispiele zu finden. Dazu wird in jedem Schritt die Kostenfunktion minimiert, jedoch darf die Abweichung zur bisherigen Policy nur gering sein [5]. Gemäß [1] wird bei der Verwendung des ML-Agents Toolkit in Unity (vgl. Kapitel 4) als Grundeinstellung dieser Algorithmus ausgewählt. Die Erfahrung hat gezeigt, dass dieser einen großen Einsatzbereich hat und stabiler ist als viele andere.

### 2.2.2 Soft Actor-Critic (SAC)

Im Unterschied zum PPO Algorithmus wird beim SAC auf die Erfahrungen aus den vorangegangenen Zeitpunkten zurückgegriffen und die Erkenntnisse daraus zufällig wieder herangezogen. Der Vorteil gegenüber dem PPO ist dadurch, dass weniger Stichproben gebraucht werden. Ein SAC bietet sich dann an, wenn man ein schweres und langsames Environment hat [1].

## 3 | Hardwareanalyse

Diese Projektarbeit baut auf mehreren Vorgängerprojekten auf. Bei der Übergabe wurden einige Mängel bezüglich der Konstruktion festgestellt. Deshalb wird in diesem Kapitel eine umfangreiche Hardwareanalyse durchgeführt. Der Demonstrator besteht aktuell aus einem Airhockeytisch, einer Kamera und der Mechanik des Roboters, der den Roboterspieler verkörpert. Diese Mechanik besteht aus zwei Linearachsen, die eine Bewegung des “Pushers” in einer Ebene ermöglichen. Angesteuert werden die beiden Motoren mittels eines Arduinos. Das Arduino wird mit einer seriellen Schnittstelle erreicht. Bei den Motoren handelt es sich um Schrittmotoren, die mit einem Riemen einen Schlitten auf einer Führung entlang der Linearachsen bewegen. Um mehr Stabilität und Dynamik zu erreichen, wurde eine H-Anordnung für den Aufbau der Führungswellen gewählt.

### 3.1 Aufbau

Für eine bessere Übersicht sind in den nachfolgenden Abbildungen 3.1 und 3.2 die einzelnen Komponenten eingetragen, welche im Folgenden genauer betrachtet werden.

Die Funktionsweise des hier bereits realisierten Aufbaus besteht aus zwei Elektromotoren (1), die jeweils ein Zahnrad (4) antreiben. Über einen fest gespannten Zahnriemen und mehrere Umlenkrollen kann der Schlitten inklusive des Pushers (7) in neun mögliche Richtungen entlang der Führungswellen (2, 6) bewegt werden. Diese sind von den jeweiligen Laufrichtungen der zwei Motoren abhängig. Zur Aufnahme des Spielfelds für ein späteres Spiel gegen den trainierten Agenten ist oberhalb des Mittelpunkts eine Kamera (9) auf einer Querstrebe befestigt.

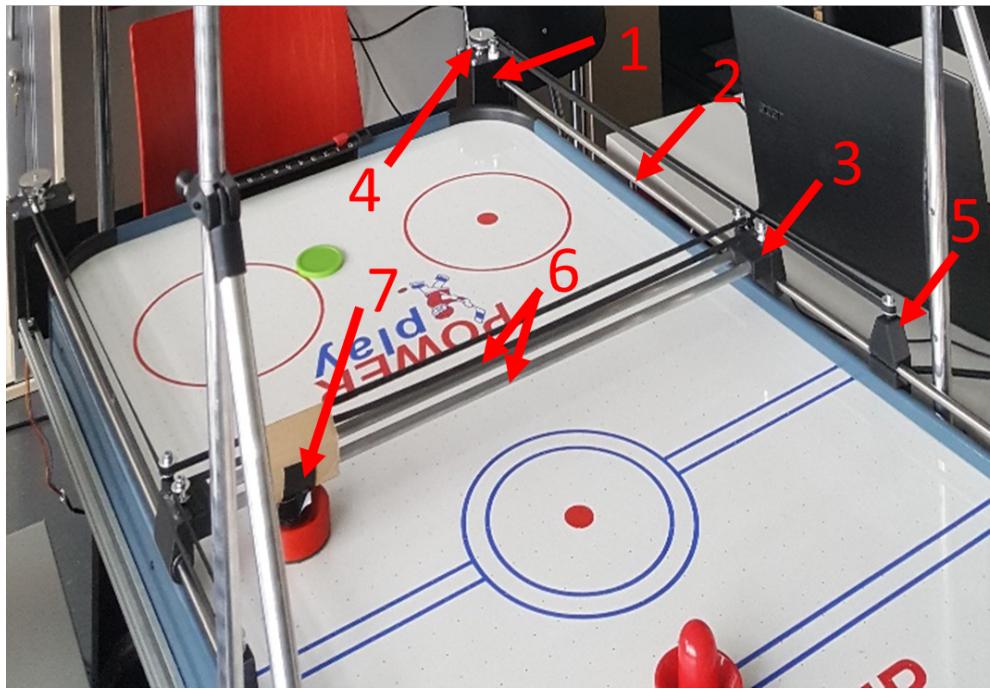


Figure 3.1: (1) Motor und Motorgehäuse; (2) Führungswelle (Längsrichtung); (3) Schlitten auf der Führungswelle (Längsrichtung); (4) Zahnrad; (5) Hintere Halterung der Führungswelle (Längsrichtung); (6) Führungswelle (Querrichtung); (7) Aufnahmeschlitten für den Pusher des Trainierten Agenten

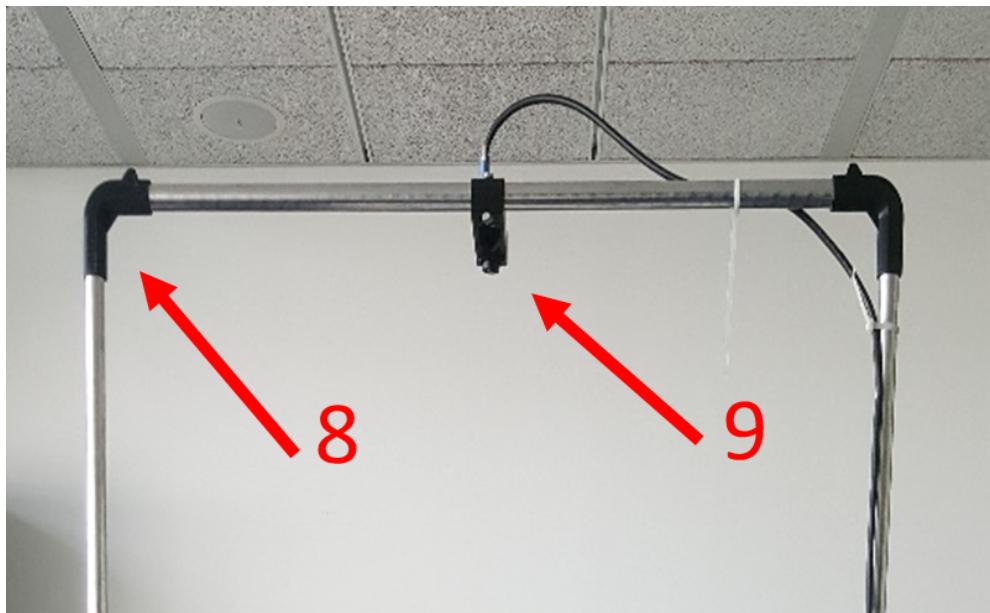


Figure 3.2: (8) Befestigung der Querstrebe für die Kamerahalterung; (9) Kamera

## 3.2 Komponentenaustausch

Die Komponenten des Hardwareaufbaus werden für eine bessere Übersicht in drei Gruppen eingeteilt. Die erste Gruppe "Bewegung des Roboters" beinhaltet alle Komponenten, die für das dynamische Bewegen des Pushers (7) beteiligt sind (vgl. Abbildung 3.1). Die Gruppe zwei "Bilderfassung" enthält die Kamera inklusive dem Halter und dem dazugehörigen Aufbau (vgl. Abbildung 3.2). Mit der letzten Gruppe "Original Airhockeytisch" ist der Originalaufbau des Airhockeytisches, so wie dieser gekauft wurde, gemeint.

### Bewegung des Roboters:

Die Bewegungen des Pushers (7) sind nur dann ideal, wenn der Zahnriemen fest gespannt wird. Aufgrund eines fehlenden Endanschlags am hinteren Halter (5) der Führungswelle in Längsrichtung (2) und des zu festen Spannen des Zahnriemens wirkte eine zu große Kraft  $F$  am Motorengehäuse (1), sodass dieses leicht nach vorne geneigt war (Winkel  $\alpha$ ). Begünstigt durch die Aussparungen des HHN Logos und diese zuvor beschriebene Kraft  $F$  brach das Gehäuse (1) an einigen Stellen auseinander. In der Abbildung 3.3 sind beide Effekte zu erkennen.

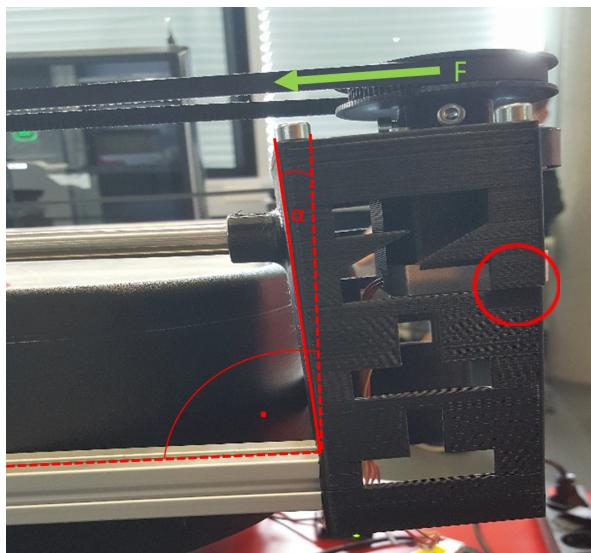


Figure 3.3: Neigung des Motorgehäuses um Winkel  $\alpha$  aufgrund der Kraft  $F$  und der daraus resultierende Riss am Motorengehäuse ist rot umrandet.

### CHAPTER 3. HARDWAREANALYSE

---

Zur Lösung dieses Problems wurde zunächst das Motorgehäuse (1) ohne Aussparungen und mit verstärkter Wanddicke 3D gedruckt. Auch neu gedruckt wurden die hinteren Halterungen der Führungswelle in Längsrichtung (5), sodass diese Welle (2) nun mit einem Keil in der Halterung festgeklemmt werden kann und somit der Kraft F entgegenwirkt. In der Abbildung 3.4 ist diese hintere Halterung, durch den die Welle mithilfe des Keils (rot umkreist) fixiert wird, dargestellt.



Figure 3.4: Hintere Halterung der Führungswelle in Längsrichtung mit rot gekennzeichnetem Keil zur Verklemmung dieser Welle.

Im Zuge der gesamten Stabilisierung des Zahnrämenantriebs wurden die zwei Antriebszahnräder aus dem 3D Drucker durch industriell hergestellte Aluminiumzahnräder ersetzt. Des Weiteren wurden auch die 3D gedruckten Umlenkrollen für die Spannvorrichtung durch industriell Gefertigte ersetzt. In der nachfolgenden Abbildung 3.5 ist diese Spannvorrichtung aus dem 3D Drucker rot umkreist dargestellt.



Figure 3.5: Umlenkrolle für die Spannvorrichtung aus dem 3D Drucker ist rot markiert.

Eine weitere Auffälligkeit war die Verdrehung der Führungswellen in Querrichtung (6). In Abbildung 3.6 ist dies deutlich erkennbar, denn die Führungswellen sollten parallel zu den blauen Linien auf dem Spieltisch stehen. Das ist aber nicht der Fall.

Diese Verdrehung hatte zur Folge, dass das Ansteuern von einem bestimmten Punkt mit dem Pusher (7) fast nicht möglich war, da z. B. bei einer reinen Querbewegung auch ein kleiner Anteil an Längsbewegung als Störung dazu kam.

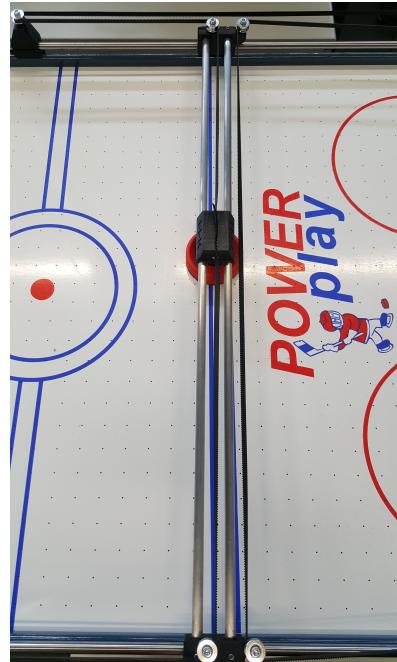


Figure 3.6: Verdrehung der Führungswellen in Querrichtung

Gelöst wurde dieses Problem durch eine Vergrößerung der Einfassungen der Querstreben in den Schlitten (3) auf der Führungswelle in Längsrichtung. Abbildung 3.7 zeigt den Vergleich zwischen der neuen verbauten Version und der alten, darunter auf dem Tisch liegenden Version. Der Unterschied ist eine Erweiterung der Einfassungen der Querstreben.

Bei anschließenden Tests konnten keine Richtungsabweichungen mehr festgestellt werden.

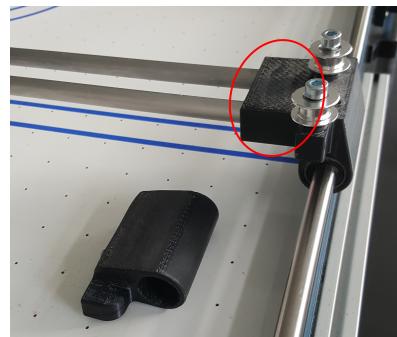


Figure 3.7: Vergleich der Schlitten (3): Neue Version ist verbaut, die alte Version liegt auf dem Tisch darunter (Orientierung: Der alte Schlitten ist um 90° Richtung Tischplatte gedreht).

Bilderfassung:

Eine weitere Komponente, die verbessert werden musste, war die Befestigung der Querstrebe für die Kamerahalterung (8). Bei der bisherigen Lösung wurde diese obere Stange, auf der die Kamera (9) mittig fest fixiert ist, links und rechts in die Bögen aus dem 3D Drucker (8) lose eingesteckt. Um eine Verdrehung der Kamera (9) bzw. der Stange zu verhindern, wurde diese mit Madenschrauben an den Plastikverbindungsstücken fixiert. Durch zu festes Zudrehen der Madenschrauben sind beide Halterungen jedoch gebrochen.

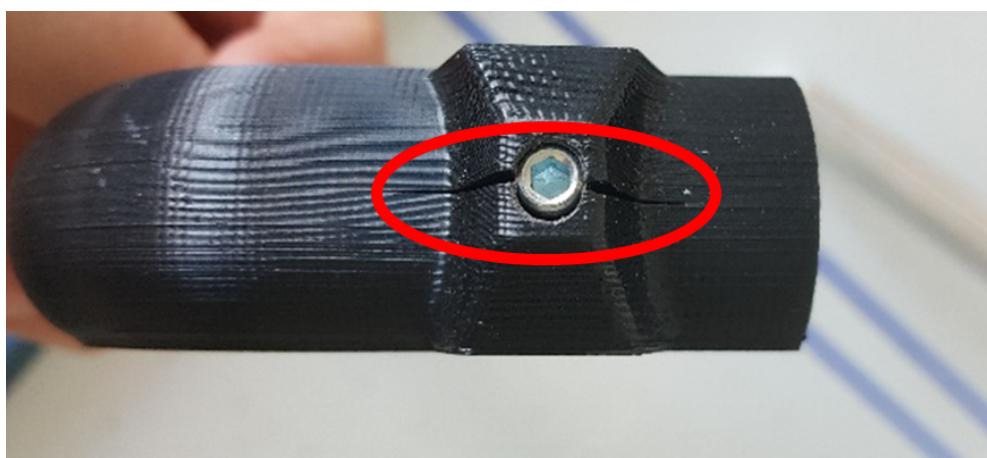


Figure 3.8: Riss am Befestigungsbogen für die Querstrebe der Kamerahalterung.

Da der Schwerpunkt der Kamera (9) inklusive der Halterung leicht versetzt zum Schwerpunkt der oberen Stange liegt, wirkt dieses Moment gegen die Fixierungen mit den Madenschrauben. Dies hatte zur Folge, dass sich diese Stange und die darauf fixierte Kamera langsam drehten. Damit veränderte sich das Sichtfeld der Kamera kontinuierlich, bis es zum Schluss nicht mehr den gesamten Spieltisch umfasste.

Um dieses Problem zu umgehen, wurde ein anderer Ansatz zur Befestigung der Querstrebe gewählt. Die neu konstruierten 90° Bögen (8) besitzen Aussparungen in Längsrichtung, sodass das Ende des Bogens leicht zusammengedrückt werden kann. Die Querstrebe wird nun mithilfe von Rohrschellen darin fest verklemmt. Der Vorteil an einer Fixierung mit einer Rohrschelle gegenüber einer mit Madenschraube ist, dass die Kraft nun nicht konzentriert auf eine kleine Stelle wirkt, sondern auf den kompletten Umfang verteilt wird.

Original Airhockeytisch:

Die größte Herausforderung bei der Optimierung des Hardwareaufbaus bestand darin, dass der Pusher (7) in der Nähe der Tischkante, insbesondere der Torlinie, oftmals ins Stocken gerät oder sogar ganz stecken bleibt. Eine genauere Betrachtung der Spielplatte zeigte, dass diese in der Spieltischmitte leicht absinkt und an den Banden wieder ansteigt. Es ist möglich, den Pusher wenige Millimeter in der Höhe zu verstellen, indem entweder Distanzscheiben in die innen liegende Verschraubung hinzugefügt oder entfernt werden. Da diese Höhenverschiebung erst gegen Ende des Projekts durchgeführt wurde, kam es bereits beim Testen zu Rissbildungen am Pusher (7). In der Abbildung 3.9 können die Risse links und rechts jeweils an der Aufnahme der Führungswellen (6) erkannt werden.



Figure 3.9: Risse am Pusherschlitten entlang der Einfassung der Führungswellen in Querrichtung.

### CHAPTER 3. HARDWAREANALYSE

---

Wenn jedoch der Abstand zwischen Tisch und Pusher zu groß ist, kann es passieren, dass der Puck dazwischen eingeklemmt wird. Leider ist es mit dem aktuellen Aufbau nicht möglich, beide Probleme gleichzeitig zu lösen.

Des Weiteren reduziert sich die Geschwindigkeit des Pucks beim Überqueren der Aufkleber auf dem Tisch und durch Verschmutzungen an der Oberfläche. Dadurch kommt es zu unerwarteten Schwankungen, was die Übertragung von Simulationsergebnissen auf den realen Aufbau nachteilig beeinflusst.

## 4 | Simulationsumgebung

Ohne eine angemessene Simulationsumgebung ist das ganze Projekt undenkbar. Nicht nur, dass das Training am realen Demonstrator schon wegen des Zeitaufwandes praktisch nicht möglich ist, auch die Konsistenz der Umgebung ist infrage zu stellen. Die Integration der vielen Rewards sind mit der Bildverarbeitung auch komplizierter und in einer Simulation zusätzlich präziser. Da wegen vielen Gründen eine Simulation nötig ist und diese auch einen großen Teil der Arbeit ausgemacht hat, wird im folgenden Kapitel das Programm Unity vorgestellt. Außerdem wird unser Unity Projekt hinsichtlich der Implementierung und der Nutzung vorgestellt.

### 4.1 Auswahl der Umgebung

Da bereits aus einer vorhergegangenen Arbeit ein Projekt in Unity vorhanden war, ist die Entscheidung hier sehr schnell gefallen. Selbst ohne diesen Aspekt ist Unity aber eine gute Wahl. Neben einer Python Schnittstelle, die für Reinforcement Learning generell immer nützlich ist, kann Unity auch noch mit einer großen Community und sehr guter Dokumentation punkten. Dadurch ist die Einarbeitungsphase relativ kurz und angenehm. Hinzu kommt noch die Toolbox ML-Agents. Diese stellt bereits Agenten zur Verfügung, bei denen nur noch Hyperparameter gewählt werden müssen. Dies ebnet die Möglichkeit, schnelle und unkomplizierte mit dem Training zu beginnen.

### 4.2 Unity

Unity ist eine von Unity Technologies entwickelte Multiplattformentwicklungsumgebung zum Erstellen von Videospiele. Für dieses Projekt ist die Anwendung zwar kein Videospiel im herkömmlichen Sinn, aber die Physikengine ist hierfür trotzdem nützlich. Neben dreidimensionalen Umgebungen bietet Unity auch einen 2D-Modus an, der für unsere Anwendung ausreichend ist. Die Programmiersprache unserer Wahl ist c#, jedoch ist es auch möglich, in UnityScript und Boo benutzerdefiniertes Verhalten zu programmieren.

Für dieses Projekt wurde die Unity Version 2020.1.6f1 genutzt.

Die Version des ML-Agents Toolkits, die zum Einsatz kam, lautet 2.1.0-exp.1

## CHAPTER 4. SIMULATIONSUMGEBUNG

---

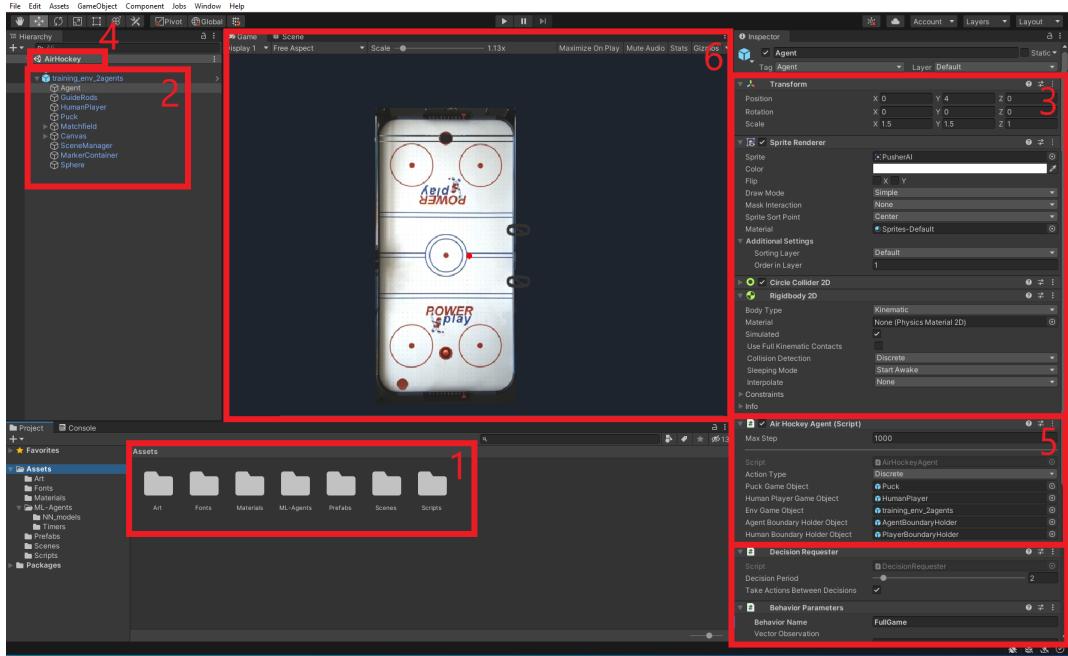


Figure 4.1: Benutzeroberfläche von Unity mit Markierungen

### 4.2.1 Kurze Einführung

Die kurze Einführung in Unity soll anhand des Userinterfaces geschehen. Diese Einführung ist keineswegs vollständig und soll auch nur ein grobes Verständnis für Fachfremde ermöglichen. Die Markierungen im Bild ?? werden im Folgenden erklärt.

- Assets (1)

Assets beinhaltet Diverses. Neben Grafiken, vorgefertigten Materialien und Szenen sind hier auch die Skripte zu finden.

- GameObjects (2)

GameObjects sind das zentrale Element von Unity. Jedes Objekt, jede Kamera und jede Grafik ist durch ein GameObject definiert. Die Funktionalitäten eines GameObjects werden durch die Components hinzugefügt. Die GameObjects sind in einem Hierarchiebaum in der Scene eingeordnet.

- Components (3)

Components geben den GameObjects ihre Funktionalität. Ein GameObject kann mehrere Components haben. Beispiele für Components sind Transform (zuständig für die Position), Collider (zuständig für die Interaktion in der Physiksimulation) und auch Skript.

- Scene (4)

Eine Scene (Szene) besteht aus GameObjects. Es ist im Prinzip die Wurzel des Hierarchiebaums. Zur Spieleentwicklung könnten hier unterschiedliche Level als unterschiedliche Scenes interpretiert werden. In unserem Fall sind nur zwei Scenes vorhanden: Eine mit einem Spielfeld zum selbst spielen und testen und eine mit acht Feldern für ein beschleunigtes Training.

- Script (5) :

Scripts sind auch Components. Sie bieten die Möglichkeit, das Verhalten selbst zu definieren. Es kann sich in Scripts auf GameObjects bezogen werden. Mit dieser Hilfe können Parameter wie das Material oder die Position geändert werden. Objekte können auch entfernt oder eingefügt werden.

- Visualisation (6) :

In diesem Bereich kann sowohl das Bild einer Kamera und damit die Ansicht im Spiel beobachtet werden als auch eine Darstellung aller GameObjects in der Scene. Objekte können hier auch verschoben oder gedreht werden.

### 4.2.2 Airhockey Projekt

In diesem Unterkapitel werden die einzelnen GameObjects und die wichtigsten Components im Projekt vorgestellt und erklärt. Ziel ist es dabei nicht auf jedes Detail einzugehen, sondern die Arbeit für Folgeprojekte zu erleichtern. Rein optische Aspekte, wie zum Beispiel die Anzeige des Spielstandes, werden nicht betrachtet. Außerdem wird nur auf die Scene mit einem Feld eingegangen, dann die Anpassungen, die zum Kopieren gemacht werden müssen, sind nicht maßgeblich. Die folgenden GameObjects sind auch in der Abbildung 4.2.1 zu finden.

#### Agent :

Der Agent ist eines der zentralen GameObjects. Er hat eine Sprite Renderer Component. Dadurch kann er mit einer .png Datei visualisiert werden. Der Agent ist also ein Objekt, das im Spiel zu sehen ist. Auf der rechten Seite ist die Ansicht, wie sie auch im Projekt zu sehen ist, dargestellt.

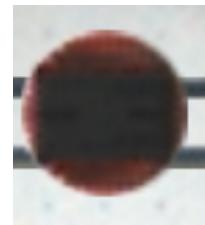


Figure 4.2:  
Ansicht des  
Agenten in  
Unity

Das Agent GameObject enthält auch eine Circle Colider Component. Mit der Rigidbody Component zusammen wird die Phsyik (Reibung, Bewegung, Kollision) simuliert.

Ein weiterer Component des Agent Objekts ist das Script Airhockey Agent. Die Funktion von diesem Script ist die Interaktion mit der ML-Agents Toolbox. Es werden sowohl die anfallenden Rewards entsprechend des Spielverlaufes an das Netzwerk zurückgegeben, als auch die Actions entgegengenommen und damit die Umgebung (Bewegung des Agenten) beeinflusst. Zu Episodenbeginn werden in diesem Script auch die Positionen von Agent und Spieler (Pusher) zurückgesetzt.

Des Weiteren beinhaltet das Agent GameObject auch noch eine Decision Requester Component. Mit ihrer Hilfe wird regelmäßig ,entsprechend des Parameters Decision Periode, eine Action vom Netzwerk angefordert. Die Behavior Parameter Componente ist, neben Decision Requester, zur Parametrisierung der Nutzung des ML-Agents Toolkits nötig. Hier kann die Dimension sowohl des Actionspaces als auch die der Observation festgelegt werden. Auch Angaben zur Inferenz können hier gemacht werden.

### HumanPlayer :

HumanPlayer ist das GameObject des zweiten Spielers. Es hat auch eine Sprite Renderer Component. Die Visualisierung ist rechts zu sehen. Da dieses Objekt auch am Spiel teilnimmt, hat es auch die Components Circle Colider und Rigidbody.



Figure 4.3:  
Ansicht des  
Pushers in  
Unity

HumanPlayer hat auch die Components Decission Requester und Behavior Parameters. Damit wird das Selfplay ermöglicht. In unserer Implementierung wird die Version des Agenten, die den HumanPlayer bewegt nicht trainiert. Das ML-Agents Toolkit wird hier nur zur Inferenz genutzt.

Das Objekt enthält auch ein Script. Im Gegensatz zum Airhockey Agents Script werden hier aber keine Rewards zurückgegeben, sondern nur die Observations und die Actions behandelt. Das reicht auch aus, denn dieser Agent soll ja nicht ständig mittrainieren, sondern nur das Selfplay ermöglichen. Wenn er zu schwach wird, wird einfach die stärkere Version vom Agenten übertragen. Im Script ist auch die Option, den HumanPlayer selbst zu steuern, implementiert. Entweder mit der Tastatur oder mit einem Controller kann so der Agent in der Simulation herausgefordert werden.

### Puck :

## CHAPTER 4. SIMULATIONSUMGEBUNG

---

Der Puck ist auch ein Objekt, das wie auch der HumanPlayer und der Agent am Spiel teilnimmt. Deshalb hat es auch die Components Circle Collide und Rigidbody. Die Components, die für das ML-Agents Toolkit nötig sind, werden hier aber nicht verwendet. Trotzdem gibt es auch hier ein c# Script, das das Verhalten des Pucks bestimmt.



Figure 4.4:  
Ansicht des  
Pucks in Unity

Abhängig vom Spielszenario werden in diesem Script die Startposition und die Startgeschwindigkeit des Pucks festgelegt. Auch der Spielstand wird hier mitgezählt. Die Circle Collider Componente des Pucks erlaubt es, ein Event bei Kollision mit anderen Objekten auszulösen. Wenn der Spielfeldrand am Tor des Agenten getroffen wird, kann das mithilfe eines GameObjects ohne Rendererer erkannt werden. Es kann mit einer Box Collider Componente ein Event ausgelöst werden, das den Spielstand anpasst. Das gleiche System wird auch auf der anderen Seite angewendet.

### Matchfield :

Das GameObject Matchfield hat selbst nur eine Sprite Renderer Component, die ein Bild des Airhockeytisches zeigt. Jedoch sind diesem Objekt hierarchisch Weitere untergeordnet. Diese untergeordneten Objekte sind aber alle ohne Rendererer Component und deshalb in der Spielansicht unsichtbar. Sie sind nur wegen ihrer Position interessant. Sie dienen dazu, das Spielfeld in Zonen einzuteilen. Die zulässigen Positionen von Agent, HumanPlayer und Puck werden damit auf das Spielfeld oder die jeweilige Hälfte limitiert. Die GameObjects, die zum Torezählen genutzt werden, sind auch Childobjects (hierarchisch untergeordnete Objekte) des Matchfields.



Figure 4.5:  
Spielfeldbegrenzungen  
in Unity

Das Matchfield hat kein Script, es wird sich nur von anderen Scripts auf die GameObjects von Matchfield bezogen. Die Transform Component, die die Position eines Objects angibt, wird dabei ausgelesen. Mit vier Objekten lässt sich damit ein rechteckiger Bereich festlegen.

### training\_env\_2agents :

Diesem Objekt sind, abgesehen von der Kamera, alle anderen GameObjects untergeordnet. Es dient als Container für das ganze Spiel. Das Objekt dient

hauptsächlich zwei Zwecken:

- Das ganze Spielfeld kann einfacher kopiert und verschoben werden. Die Positionen der Childobjects bleiben beim Verschieben des übergeordneten Objekts relativ zu diesem unverändert. Felder können dadurch schnell über und nebeneinander kopiert werden.
- Das envScript ist in diesem GameObject ein Component. In diesem Script werden alle Rewards festgelegt. Auch der Spielmodus wird hier ausgewählt. Näheres zu den Rewards und den Spielmodi wird im Kapitel 4.2.2 erläutert.

# 5 | Rewards und Training

In diesem Kapitel wird auf das Training eingegangen. Hierbei werden die unterschiedlichen Rewards erklärt, die genutzten Netzwerke, die Hyperparameter und die einzelnen Schritte im Training. Dazu wird auch das envScript aus Unterkapitel 4.2.2 noch mal genauer beleuchtet.

## 5.1 Rewards und Umgebungsparameter

Hier sollen als erstes alle implementierten Rewards vorgestellt werden, damit in den folgenden Teilen der Beschreibungen des Trainings klar ist, welche Effekte sie bedingen.

- taskType  
Legt fest, welcher Spielmodus gewählt wird. Siehe dazu Abschnitt 5.2.
- V\_max\_puck  
Legt die Maximalgeschwindigkeit des Pucks fest. Einheit ist dabei nicht m/s, der Tisch ist in der Realität kleiner als in Unity. Die Ermittlung der Maximalgeschwindigkeit kann im Abschnitt 6.1 nachvollzogen werden.
- V\_max\_robo  
Legt die Maximalgeschwindigkeit des Roboters fest. Messungen erfolgten analog zu denen der Maximalgeschwindigkeit des Pucks.
- V\_max\_human  
Legt die Maximalgeschwindigkeit des HumanPlayer fest. Messungen erfolgten analog zu denen der Maximalgeschwindigkeit des Pucks.
- neghumanGoalReward  
Dieser Reward wird vergeben, wenn der Puck in das Tor des Agenten trifft. Da es ein Gegentor ist, sollte der Reward negativ gewählt werden.
- agentGoalReward  
Dieser Reward wird vergeben, wenn der Puck in das Tor des Human-Player trifft.

- avoidBoundaries

Dieser Reward wird vergeben, wenn der Roboter die Bande berührt. Er sollte negativ sein.

- avoidDirectionChanges

Dieser Reward wird vergeben, wenn der Roboter die Bewegungsrichtung ändert. Er sollte negativ sein. Der Betrag wird mit dem Betrag der Bewegungsänderung im letzten Zeitschritt multipliziert. Da dieser Reward in jedem Zeitschritt vergeben werden kann, sollte dieser Reward betragsmäßig sehr klein gewählt werden.

- stayCenteredReward

Dieser Reward belohnt den Agenten, wenn er weder links noch rechts am Spielfeldrand platziert ist. Damit kann das Verteidigungsverhalten verbessert werden. Der Reward wird zu jedem Zeitschritt vergeben und sollte deshalb sehr niedrig gewählt werden. Der Reward sinkt proportional mit der Distanz des Pushers vom Rand.

- negoffCenteredReward

Dieser Reward ist vergleichbar mit dem stayCenteredReward. Jedoch ist dieser negativ zu wählen, denn er fällt betragsmäßig am höchsten aus, wenn der Agent an der Bande steht (links oder rechts).

- encouragePuckMovement

Dieser Reward belohnt die Puckbewegungsgeschwindigkeit. Er wird zu jedem Zeitschritt vergeben und sollte deshalb klein sein.

- encouragePuckContact

Dieser Reward belohnt Kontakte mit dem Puck.

- contacthalf

Dieser Parameter ist kein Reward sondern eine Booleanvariable. Wird sie zu True gesetzt hat das zur Folge, dass der Reward encouragePuckContact jedes mal, wenn er vergeben wird, halbiert wird. Damit wird verhindert, dass der Roboter alleine spielt oder den Puck einklemmt um den encouragePuckContact Reward auszunutzen.

- playForwardReward

Dieser Reward wird vergeben, wenn der Puck von hinten getroffen wird, also in Richtung gegnerisches Hälften geschossen wird.

- negplaybackReward

Dieser Reward wird vergeben, wenn der Puck von vorne getroffen wird, also in die falsche Richtung geschossen wird. Er sollte negativ gewählt werden.

- negStepReward

Dieser Reward wird in jedem Fall zu jedem Zeitschritt vergeben. Er soll

ein mögliches Zeitspiel des Agenten verhindern. Dazu muss er negativ sein.

- [negMaxStepReward](#)

Dieser Reward wird vergeben, wenn die Maximallänge einer Episode erreicht wird und das Spiel deshalb abgebrochen werden muss (wird nur im Training abgebrochen, nicht im Spiel). Er sollte negativ sein.

- [behindPuckReward](#)

Dieser Reward wird zu jedem Zeitschritt, zu dem sich der Agent näher am eigenen Tor befindet als der Puck gegeben. Er sollte niedrig gewählt werden, da diese Rewardvergabe in jedem Zeitschritt möglich ist.

- [behindPuckReward](#)

Dieser Reward ist exklusiv für das Training des Spielmodus Defending. Er wird jedes Mal vergeben, wenn ein Schuss erfolgreich verteidigt wurde.

## 5.2 Spielmodi

In diesem Abschnitt werden die implementierten Spielmodi beschrieben. Mit ihrer Hilfe kann die Eignung eines Netzwerk oft schneller festgestellt werden, als mit einem ganzen Spiel, da die anderen Spielsituationen nicht so komplex sind. Ein weiterer Vorteil von einfacheren Spielszenarien ist, dass damit der Agent Stufe für Stufe trainiert werden kann.

- [Reaching](#)

Das Ziel in diesem Spielmodus ist es, den Puck zu erreichen. Dazu wird der Puck zu Beginn der Episode an einem zufälligen Ort auf der Spielfeldseite des Agenten platziert. Da HumanPlayer bei diesem Szenario keine Rolle spielt wird seine Bewegung gestoppt. Die Episode endet mit der Kollision von Agent und Puck oder beim Erreichen der maximalen Simulationsschritte. Damit ist dieser Modus von sehr niedriger Komplexität und auch für einen untrainierten Agenten schnell erlernbar.

- [Scoring](#)

Im Scoring Modus geht es nur ums Zielen und darum, das Tor zu treffen. Hier wird die Episode nicht beendet, wenn der Puck getroffen wurde, sondern wenn der Puck entweder ein Tor erreicht oder die Bande trifft. Wie beim Reaching steht der HumanPlayer unbeweglich neben seinem Tor. Da mit einem diskreten Actionspace gearbeitet wird, ist dieser Modus nicht viel verwendet worden. Wegen des diskreten Actionspace kann der Agent sich nicht präzise genug platzieren, um zu zielen. Sollte in einem Folgeprojekt jedoch mit einem kontinuierlichen Actionspace gearbeitet werden, kann dieser Modus wieder interessant werden.

- [Defending](#)

Hier hat der Agent das Ziel, sein Tor zu verteidigen. Auch in diesem

Modus spielt der HumanPlayer keine Rolle. Er steht dabei unbeweglich neben seinem Tor. Der Agent wird zu Beginn der Episode an eine zufällige Position auf seiner Spielfeldseite platziert. Defending ist der einzige Modus, bei dem der Puck nicht ohne Geschwindigkeit auf das Feld gesetzt wird. Der Puck wird hier von der Seite des HumanPlayer aus mit einem zufälligen Bewegungswinkel zwischen  $70^\circ$  und  $-70^\circ$  auf die Seite des Agenten geschossen. Auch die Startposition variiert zufällig. Beendet wird die Episode, wenn entweder ein Tor kassiert wurde oder der Ball abgewehrt wurde. Das ist der Fall, wenn die X-Komponente der Geschwindigkeit des Pucks negativ wird, sich der Puck also auf die Hälfte des HumanPlayer zurückbewegt. In diesem Modus ist ein schneller Lernfortschritt nur mit dem defenceReward zu erwarten (bei fast allen Netzwerken).

- FullGame

FullGame ist selbsterklärend. Der Puck wird an einer zufälligen Position auf dem ganzen Spielfeld ins Spiel gebracht. Die Kontrahenten Agent und HumanPlayer werden auf ihre jeweilige Seite gesetzt und beiden ist die Bewegung im Rahmen der Parameter in envScript erlaubt. Beendet wird die Episode nur, wenn ein Tor erzielt wird oder die maximale Anzahl an Simulationsschritten erreicht wird. Diesen Modus zu meistern ist das Ziel der Trainings und eine Teilaufgabe des Projekts. Er ist zu komplex, um einen untrainierten Agenten ohne das stufenweise Ändern von Rewards zu trainieren

Modus	Reaching	Scoring	Defending	FullGame
Puck	Agent Seite	Agent Seite	Schuss auf Tor	Ganzes Feld
Agent	Agent Seite	Agent Seite	Agent Seite	Agent Seite
HumanPlayer	neben Tor	neben Tor	neben Tor	Agent Seite
Episodenende	Puckkontakt	Tor, Bande	Tor verteidigt	Tor

Figure 5.1: Übersicht über die Spielmodi

### 5.3 Trainingsstrategie und Zwischenergebnisse

In diesem Abschnitt wird das Vorgehen beim Training beschrieben und die Ergebnisse des Trainings in der Unity Umgebung gezeigt. Um die Ergebnisse einordnen zu können, werden sowohl mit Graphen der kumulativen Rewards über eine Episode als auch mit den Ergebnissen aus Spielen von Agenten untereinander gearbeitet. Zusätzlich wird eine Einschätzung der Ästethik des Spiels betrachtet.

### 5.3.1 Stufenplan

Da das Spiel im FullGame Modus sehr komplex ist und viele Sachen zu beachten sind, was sich auch an der Vielzahl an sinnvollen Rewards wider-spiegelt, muss der Agent einige Vortrainings abschließen, um so Eigenschaften nacheinander zu erlernen. Dazu haben wir uns einen Stufenplan, der in der Abbildung 5.3.1 zu sehen ist, erarbeitet.

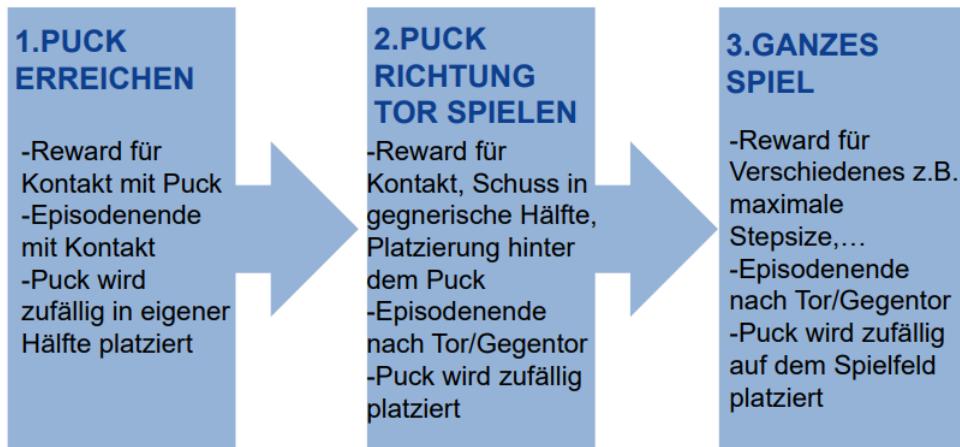


Figure 5.2: Stufenplan des Trainings

- 1. Puck erreichen

Der erste Lernschritt des Agenten ist es, den Puck zu erreichen. Diese Fähigkeit kann im Reaching Modus schnell mit effektiv nur einem Reward (encouragePuckContact) erlernt werden. Ein Nachteil beim Üben im Reaching Modus ist, dass bewegte Ziele nicht vorkommen und somit diese auch noch nicht trainiert werden können.

- 2. Puck richtung Tor spielen

Bei diesem Schritt wird auf dem vortrainierten Agenten aufgebaut. Das Ziel ist es hier den Puck in Richtung des gegnerischen Tors zu spielen. Es macht Sinn, diese Fähigkeit zuerst im Reaching Modus zu üben und erst danach in den FullGame Modus zu wechseln. Dadurch ist der Anstieg an Komplexität geringer. Hier werden die Rewards playForwardReward und negplaybackReward als Hauptrewards genutzt. Es hat sich jedoch gezeigt, dass die Rewards encouragePuckContact und behindPuckReward hier auch nützlich sind. Dadurch wird verhindert, dass der Agent, falls er vor dem Puck erscheint, die ganze Episode abwartet, um den negplaybackReward zu verhindern. Mit einem ähnlichen Satz an Rewards kann dann auch im FullGame Modus das Training fortgesetzt werden. Die Stärke des Gegners spielt dabei keine große Rolle, da keine negativen Rewards für Gegentore gegeben werden.

- 3. Ganzes Spiel

Diese Stufe im Plan ist mit Abstand die Zeit intensivste. Da hier das erste mal Rewards und Bestrafungen für Tore genutzt werden, ist die Wahl des Gegners nun ein sehr wichtiges Thema. Ist der Gegner zu schwach, kann nicht viel gelernt werden. Ist der Gegner zu stark, wird der Agent schnell passiv und versucht das Spiel auszubremsen um ein Gegentor zu verhindern. Bei diesem Schritt ist also die Balance zwischen der Anregung zum Spiel durch Rewards, wie negStepReward und encouragePuckMovement, und der Herausforderung durch den Gegner nicht einfach zu schaffen. Hinzu kommt, dass einige Reward Konstellationen zu unerwünschtem Verhalten führen. Ist beispielsweise der behindPuckReward hoch, der Gegner stark (in Vergleich zum Agenten) und der neghumanGoalReward hoch, neigt der Agent schnell dazu, sich nahe ans eigene Tor zu stellen und das Ende der Episode abzuwarten. Ist der Reward avoidDirectionChanges hoch, kann es passieren, dass der Roboter dazu neigt, oft bis an den Rand zu fahren. Der stayCenteredReward kann hier entgegenwirken.

Dadurch, dass oft unvorhergesehenes schlechtes Verhalten erlernt wird, ist es wichtig, den vorherigen Stand des Agenten zu sichern und gegebenenfalls den Fortschritt zu verwerfen, weil dieser eher einem Rückschritt gleicht. Dann kann mit einem neuen Reward- oder Parametersatz oder mit einem anderen Gegner ein weiterer Versuch vom gesicherten Stand aus probiert werden.

### 5.3.2 Netzwerkauswahl und Zwischenergebnisse

In diesem Abschnitt sollen die Ergebnisse von zwei Netzwerken und ihren Hyperparametern verglichen werden. In diesem Stil wurden noch andere Konstellationen von Hyperparametern verglichen. Nach Abwägung der Trainingsgeschwindigkeit gegen die Komplexität, wurde sich dann für die Parametersätze, welche in den Konfigurationsdateien zu finden sind, entschieden.

Das ML-Agents Toolkit stellt sowohl PPO als auch SAC Agenten zur Verfügung. Um nicht unnötige Ressourcen zu verschwenden, macht es jedoch keinen Sinn, zwei unterschiedliche Agenten zu trainieren. Deshalb wurde hier entschieden, sowohl einen PPO als auch einen SAC Agenten, jeweils das gleiche Einstiegstraining durchlaufen zu lassen. Anhand der Leistung dabei wurde dann die Wahl entschieden, mit welchem Netzwerk das weitere Training stattfinden soll.

Das Einstiegstraining erfolgte für beide Agenten mit genau den gleichen Rewards, in der gleichen Umgebung und mit den gleichen Parametern. Es beinhaltet insgesamt fünf Einzeltrainings, nach denen Rewards oder Parameter angepasst wurden.

- 1: Im Reaching Modus wird nur mit dem encouragePuckContact Reward das Netzwerk für 500k (500 000) Episoden trainiert.

- 2: Das zweite Training erfolgt immer noch im Reaching Modus, jedoch wird der encouragePuckContact Reward reduziert und der playForwardReward dazugenommen. Auch ein kleiner negStepReward wird genutzt. Das Training endet nach 1M (1 000 000) Episoden.
- 3: Dieses Training erfolgt nun im FullGame Modus. Neben den Rewards für Tore werden hier einige andere Rewards hinzugenommen um das Spielverhalten zu formen. Als Gegner wird ein anderer PPO Agent verwendet, der vorher schon etwas besser trainiert wurde. Dessen Geschwindigkeit wird aber stark limitiert um das Spiel fair zu halten. Es wird so für weitere 1M Episoden trainiert.
- 4: Nach dem letzten genannten Trainingsschritt ist aufgefallen, dass sich der Agent oft weit am Spielfeldrand befindet und so die Verteidigung vernachlässigt. Deshalb wurden für dieses Training die Rewards angepasst. Auch die Geschwindigkeit des HumanPlayers wurde erhöht, um dem Agenten besser gewachsen zu sein. So wurde dann bis insgesamt 3M Episoden trainiert.
- 5: Dieses Training unterscheidet sich nicht vom Vierten. Mit den gleichen Voraussetzungen wurde das Training bis 4M fortgesetzt

Die Ergebnisse dieser Trainings sind im Folgenden dargestellt. In den Abbildungen sind jeweils die Graphen des PPO-Agenten und des SAC-Agenten übereinandergelegt. Der linke Graph zeigt dabei die Entwicklung des kumulativen Rewards über die Anzahl der verstrichenen Trainingsepisoden, der rechte die Entwicklung der Episodenlänge. Da die Trainings aufeinander aufbauen, sind in den Graphen der fortgeschrittenen Trainings die Vorangegangenen auch enthalten.

1. Training bis 500k

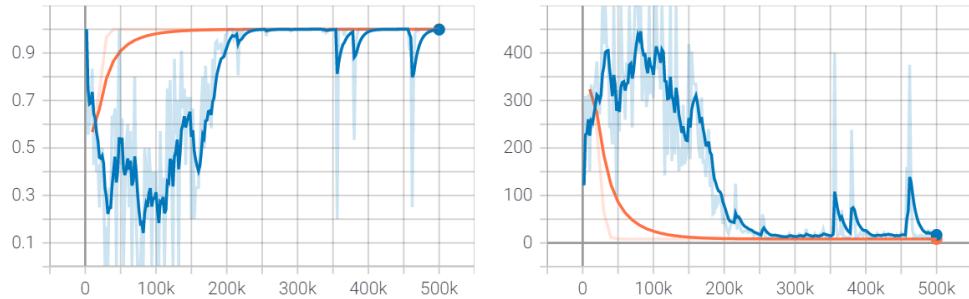


Figure 5.3: Orange: PPO-Agent, Blau: SAC-Agent

Der maximal erreichbare kumulative Reward ist in dieser Trainingsphase 1. Am Graph ist zwar zu erkennen, dass der PPO-Agent dieses Ergebnis schneller und ohne Ausreißer erreicht, aber der SAC-Agent erreicht das Ziel auch. Beobachtet man die beiden Agenten in der Unity-Umgebung bei der Aufgabe, sind sie vom Verhalten her praktisch nicht zu unterscheiden. Beide bewegen sich auf sehr kurzem Weg auf den Puck zu.

2. Training bis 1M

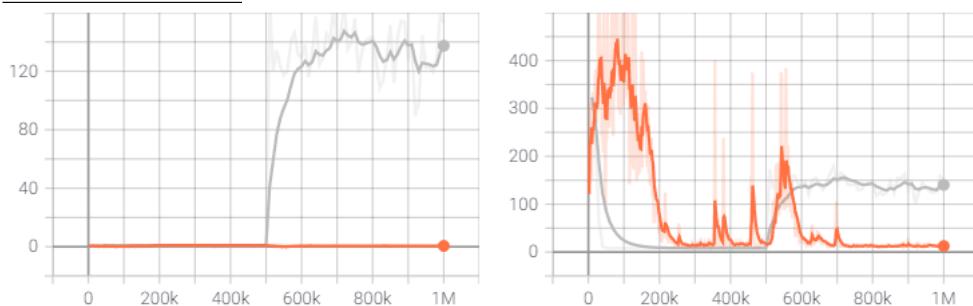


Figure 5.4: Grau: PPO-Agent, Orange: SAC-Agent

In diesem Trainingsabschnitt hat der PPO-Agent nach den Rewards klar den Vorteil. Jedoch kann man an der Episodenlänge sehen, dass die Aufgabe nicht zufriedenstellend erlernt wurde. Da die Episode beim Puckkontakt beendet wird, sollten sie sehr kurz ausfallen. Beim Beobachten des Spielverhaltens ist dies auch festzustellen. Der PPO-Agent neigt dazu, in der Nähe des Pucks in einen "Zitterzustand" überzugehen, welchen schnelle Richtungsänderungen charakterisieren. Geht er nicht in diesen Zustand, trifft er den Puck aber öfter von der richtigen Seite, als von der Falschen. Auch der SAC-Agent trifft den Puck öfter von der richtigen Seite, als von der Falschen. Er zeigt aber die Verhaltensauffälligkeit mit dem Zittern nur äußerst selten und ist damit insgesamt besser, als der PPO-Agent nach diesem Training in der zugehörigen Aufgabe.

### 3. Training bis 2M

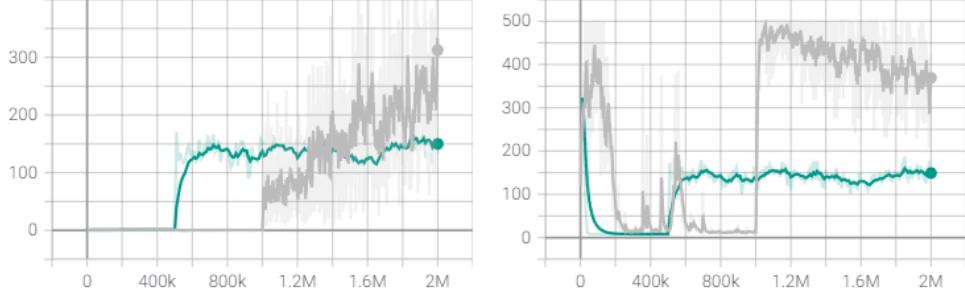


Figure 5.5: Grün: PPO-Agent, Grau: SAC-Agent

Der SAC-Agent hat in dieser Trainingsperiode seinen Rückstand in der Kategorie kumulativer Reward nicht nur verkürzt, sondern hat den PPO-Agent sogar überholt. Die Episodenlänge hat sich dabei erhöht. Da das Training im FullGame Modus erfolgte und diese Agenten jetzt dafür geeignet sein sollten, ist eine gute Möglichkeit sie zu vergleichen, das direkte Spiel gegeneinander. Hier schneidet der PPO-Agent eindeutig besser ab als der SAC-Agent. Er schlägt diesen mit 10:5. Außerdem ist sein Spielverhalten deutlich besser. Während der SAC-Agent sich oft unkontrolliert bewegt, ohne den Puck zu spielen (höhere Episodenlänge), lässt sich bei seinem PPO-Kontrahenten deutlich erkennen, dass er nicht nur zielstrebig auf den Puck zufährt, sondern ihn meist auch in die richtige Richtung spielt.

### 4. Training bis 3M

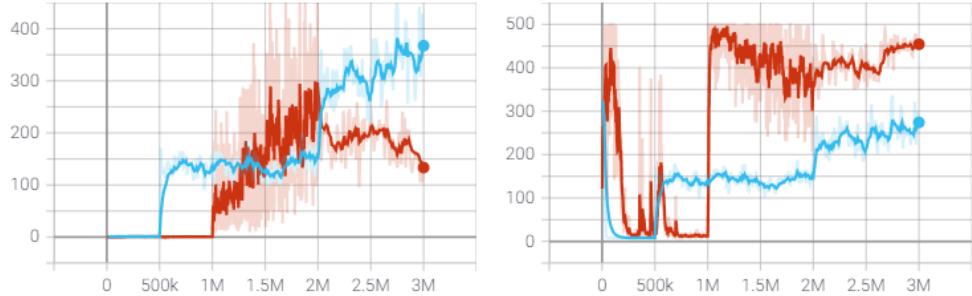


Figure 5.6: Blau: PPO-Agent, Orange: SAC-Agent

Bei diesem Training überholt der PPO-Agenten den SAC- Agenten in Sachen kumulativer Reward wieder. Auch beim Spielverhalten ist er ihm noch überlegen und schlägt ihn mit 10:4. Das Verhalten des SAC-Agenten hat sich selbst nach dem Training noch nicht deutlich verbessert. Er ist immer noch sehr passiv, wenn er einen Kontakt macht, ist dieser jedoch meist nicht schlecht. Insgesamt ist der SAC-Agent aber immer noch weit von einem Sieg entfernt.

### 5. Training bis 4M

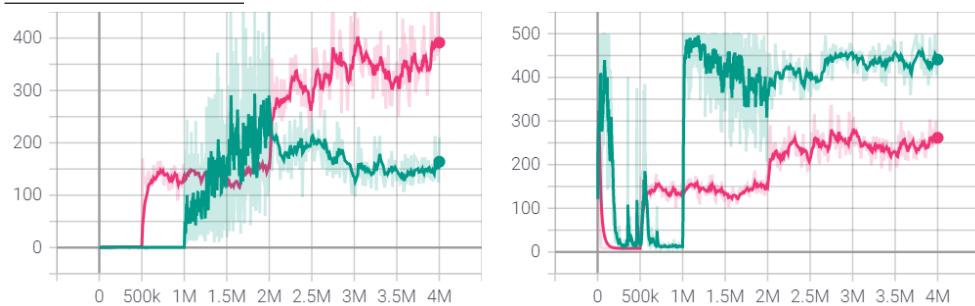


Figure 5.7: Rosa: PPO-Agent, Grün: SAC-Agent

In der letzten Trainingsetappe stagnieren nicht nur die kumulativen Rewards weitgehend, sondern auch das Spielverhalten ist weitgehend unverändert. Der PPO-Agent ist dem SAC-Agenten immer noch voraus.

Insgesamt haben die Erfahrungen im gesamten Training gezeigt, dass der PPO-Agent gegenüber dem SAC-Agenten für diese Anwendung im Vorteil ist. Bei genau gleichem Training hat er ein weit besseres, aktiveres, intelligenter wirkendes Spielverhalten entwickelt. Zusätzlich hat es den Anschein, dass sich die Verhaltensentwicklung beim PPO-Agenten besser abschätzen lässt. Die erwünschten Verhaltensmuster sind dadurch einfacher zu erreichen. Aus diesen Gründen wurde für den größten Teil der Trainingsbemühungen mit einem PPO-Agenten, der Parametrisierung des hier verwendeten Netzwerkes gearbeitet.

## 5.4 Trainingsverlauf und Ergebnis

Der vorhergegangene Abschnitt 5.3.2 hat bereits einen guten Einblick über das Vorgehen beim Training gegeben. Dieser soll noch einige Informationen zur Gegnerauswahl, zum Fortschritt, zu den Kriterien eines guten Agenten und zu den besten bereits erzielten Ergebnissen geben.

### 5.4.1 Gegnerwahl

Im letzten Abschnitt war zu erkennen, dass die ersten Trainings im Reach-ing Modus absolviert werden und deshalb dabei kein Gegenspieler benötigt wird. Diese Trainings sind jedoch ziemlich simpel und können schnell erfolgreich abgeschlossen werden. Der größte Teil des Trainings wird also mit einem Gegenspieler durchgeführt. Um zu verhindern, dass der Agent entweder nichts lernt (Gegner zu schwach) oder sich weigert, den Puck zur gegnerischen Hälfte zu spielen, weil er einen gefährlichen Schuss befürchtet (Gegner zu stark), muss ein adäquater HumanPlayer als Trainingspartner genutzt werden. Im Verlauf des Trainings bietet es sich an, die Methode des Selfplay zu nutzen. Hierbei bespielt sich der Agent selbst beziehungsweise eine Version von sich selbst. Dabei ist darauf zu achten, dass man das Netzwerk des Agenten nicht zu oft oder zu selten auf den HumanPlayer überträgt. Bei zu häufigen Transfers besteht die Gefahr, dass die Gegenspieler "zusammen spielen" und somit zum Beispiel einem Kontaktreward schnell einen Torreward obsolet machen. Um einen solchen Effekt zu erzielen, muss das Netzwerk aber oft übertragen oder gar für beide Spieler gleichermaßen trainiert werden. Bei zu geringen Transfers kann ein Training ohne vernünftige Herausforderung an Effizienz verlieren. Ein weiteres Problem des Selfplay ist, dass man zu Beginn gar keine Ergebnisse hat, welche als Gegner verwendet werden können. Deshalb ist in diesem Projekt auch ein festprogrammierter Spieler implementiert. Dieser bewegt sich stets hinter den Puck, wenn auf seine Seite gespielt wird, und spielt diesen zurück. Da die Richtung des Rückspiels aber nicht beachtet wird, ist die Gefahr eines Gegentores nicht so besonders groß. Besonders bei gedrosselter Geschwindigkeit ist er also ein idealer Einstiegsgegner.

### 5.4.2 Qualitätskriterien

Um einen guten Agenten trainieren zu können, muss zunächst geklärt werden, was ein guter Agent ist. Im Gegensatz zu einem Läufer, den man nur nach einem Kriterium bewertet (seiner benötigten Zeit um eine Strecke zurückzulegen), kann ein Agent nicht so einfach objektiv bewertet werden.

Eine erstrebenswerte Eigenschaft ist es, dass gegen möglichst viele unterschiedliche Gegner aus möglichst vielen Spielsituationen ein Tor erzielt wird. Analog dazu ist es auch erstrebenswert, gegen möglichst viele unterschiedliche Gegner aus möglichst vielen Spielsituationen kein Gegentor zu kassieren. Auch wenn viele Spielentscheidungen beide genannten Eigenschaften gleichzeitig begünstigen, kann doch eine besonders riskante Spielweise sowohl die Wahrscheinlichkeit, ein Tor zu machen als auch die Wahrscheinlichkeit eines Gegentores erhöhen. Ist demnach ein besonders defensives oder ein offensiveres Vorgehen besser? Da der Grad der Offensivneigung jedoch schwer zu messen ist und nicht mal klar ist, ob sie erwünscht ist, wird in dieser Arbeit hauptsächlich auf die Tordifferenz der Agenten geachtet. Diese Bewertung wurde dann im Spiel gegen einen älteren Agenten durchgeführt. Um zu verhindern, dass der Agent anfällig für einen menschlichen Spieler wird, wurde seine Spielfähigkeit von Zeit zu Zeit auch von einer Person per Tastatur oder Controller validiert. Logischerweise ist es jedoch nicht möglich, gegen einen menschlichen Spieler zu trainieren. Bei diesen Testspielen hat sich herausgestellt, dass bei den fortgeschritteneren Agenten einige dabei sind, die manuell nicht mehr besiegt werden können und teilweise sehr hoch in diesen Tests gewonnen haben. Diese Niederlage der menschlichen Testspieler ist jedoch ein Zeichen für ein gutes Trainingsergebnis.

Neben der Fähigkeit, viele Tore zu schießen, ohne im Gegenzug viele zu kassieren, ist auch die Ästhetik der Bewegungen nicht unwichtig. Da das Ziel des Projekts ein Demonstrator ist, ist ein nachvollziehbares Spielverhalten sinnvoll. Ein weiterer Punkt ist die Lärmbelastung. Auch wenn in Unity das Zittern keinen Nachteil mit sich bringt, kommt es dadurch am realen Tisch durch die Motoren zu schrillen, unangenehmen Geräuschen. Das sollte beim Training beachtet werden.

# 6 | Realer Demonstrator

In diesem Kapitel werden die nötigen Punkte zum Übergang auf den Demonstrator angesprochen. Dazu gehören die Messungen von Maximalgeschwindigkeiten, die Bildverarbeitung, Sicherheitsvorkehrungen, um Kollision und Schaden zu verhindern und ein Benutzerinterface für den Bediener des Demonstrators. Alle Bildverarbeitungsschritte wurden in Python mit der OpenCV Bibliothek durchgeführt.

## 6.1 Geschwindigkeitsmessungen

Bei den Geschwindigkeitsmessungen wurden die Maximalgeschwindigkeiten von Roboter und Puck gemessen. Während die Maximalgeschwindigkeit des Roboters relativ einfach erreicht werden kann, ist die des Pucks abhängig vom Spieler. Die hier verwendete Messung wurde deshalb an einem Schuss durchgeführt, dessen Geschwindigkeit als ausreichend schnell bewertet wurde. Da diese Messung vom menschlichen Können abhängt, gilt diese Maximalgeschwindigkeit nur als grober Richtwert. Gemessen wurde anhand von Videomaterial der Kamera. Mit dem Programm Tracker des Open Source Physics Projekts [7] wurde das Material analysiert. Diese Software erlaubt es, einen Punkt (Puck oder Roboter) von Frame zu Frame zu verfolgen. Mit der Bildwiederholungsrate und einem Maßstab können nun Werte ermittelt werden. Da diese Werte für das Training in der Softwareumgebung von Nöten sind, wurde als Maßstab nicht die Abmessung in Metern des realen Demonstrators genutzt, sondern die Länge in Unity-Längeneinheiten. Die Ergebnisse sind wie folgt ausgefallen:

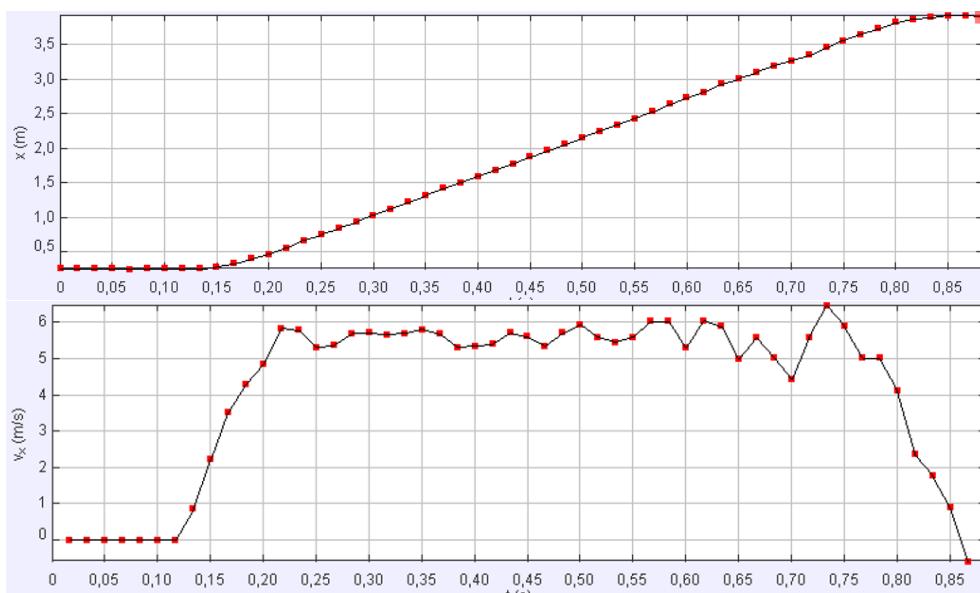


Figure 6.1: Messwerte des Roboters

Das obere Diagramm in Abbildung 6.1 zeigt den zurückgelegten Weg des Roboters über der verstrichenen Zeit seit Beginn der Messung. Das Untere zeigt den Geschwindigkeitsverlauf über der Zeit. Die Maximalgeschwindigkeit ist also in etwa 6m/s. Da im Wegdiagramm in sehr guter Näherung eine Gerade während der Bewegung zu sehen ist, kann die Geschwindigkeit während dieser Zeit als konstant angenommen werden. Rechnet man die Durchschnittsgeschwindigkeit über den Zeitraum von 0.2 bis 0.75 Sekunden aus, so erhält man bei einem zurückgelegten Weg von 3.2 Längeneinheiten (LE) eine Geschwindigkeit von 5.8 LE/s. Nach Rundung wurde mit dem Wert 6 LE/s trainiert.

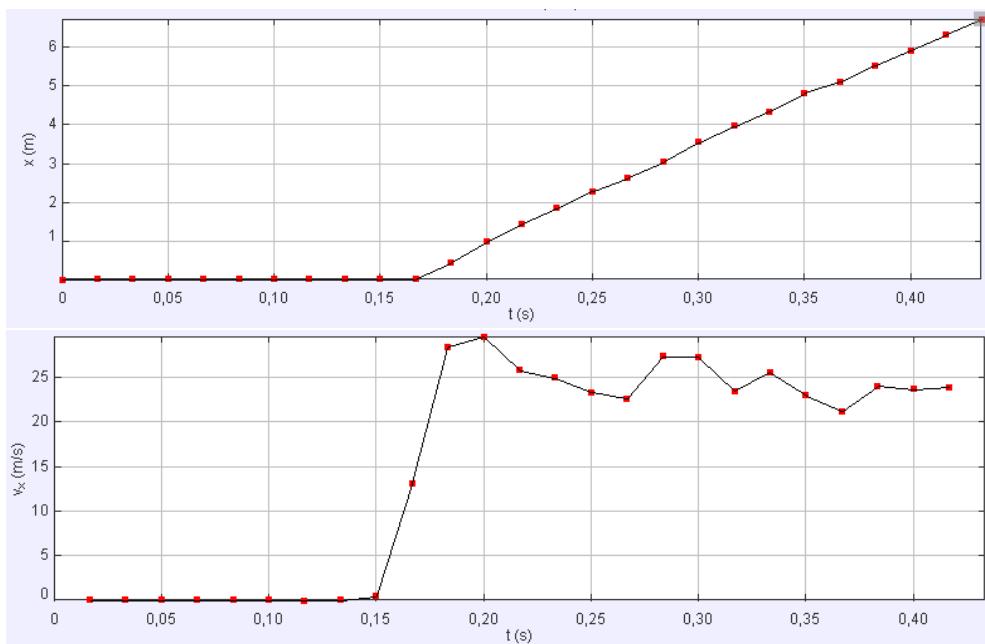


Figure 6.2: Messwerte des Pucks

Diese Diagramme sind von der Messung aus der Bewegung des Pucks. Auch hier zeigt das Obere den Verlauf des zurückgelegten Weges und das Untere den Geschwindigkeitsverlauf über der Zeit. Rechnet man hier die Steigung der Geraden im Wegdiagramm mit einer Wegdifferenz von  $6.8$  LE und einer verstrichenen Zeit von  $0.33$  Sekunden aus, so ergibt sich ein Wert von  $20.6$  LE/s. Da die Puckgeschwindigkeit bei anderen Spielern noch höher sein könnte, wurde für die Simulation in Unity und damit auch für die Trainings eine Maximalgeschwindigkeit des Pucks von  $25$  LE/s angenommen.

## 6.2 Benutzeroberfläche

Das graphical user interface (GUI) wurde in Python geschrieben. Die Bibliothek Tkinter wurde dabei genutzt. Diese Bibliothek wurde unter einer Pythonlizenz veröffentlicht und ist so frei für alle zugänglich. Dadurch ist der Einsatz der GUI in jedem Fall rechtlich unbedenklich.

### 6.2.1 Ansicht der Oberfläche

Die Folgende Abbildung zeigt das Interface. Die markierten Bereiche und Elemente werden im Folgenden erklärt.

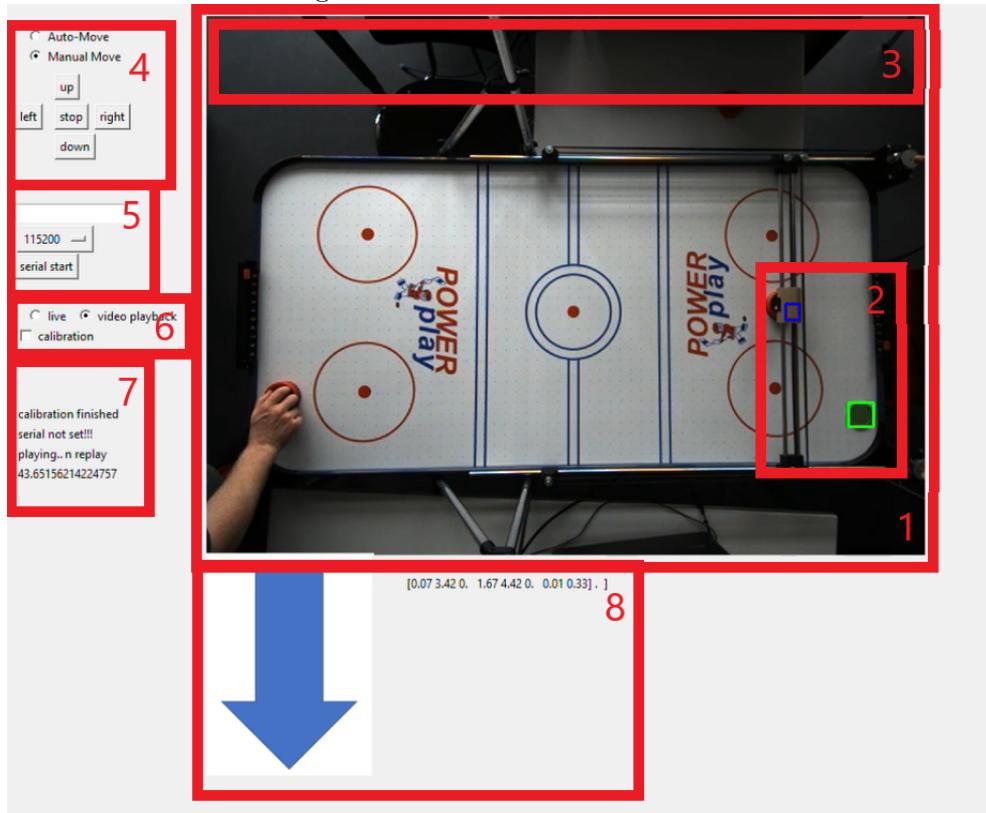


Figure 6.3: Benutzeroberfläche

## CHAPTER 6. REALER DEMONSTRATOR

---

1:

Hier ist das Kamerabild oder eine Videosequenz zu sehen. Das Bild wird, je nach Zustand, um einige Informationen ergänzt.

2:

Eine Ergänzung zum Kamerabild ist die Markierung von Roboter und Puck, sofern sie erkannt werden. Der Puck wird mit einem grünen und der Roboter mit einem blauen Rechteck markiert.

3:

In diesem Bereich wird eine Meldung gegeben, wenn Puck oder Roboter nicht erkannt werden.

4:

In diesem Bereich sind die Bedienelemente für das manuelle Verfahren. Mit den beiden Knöpfen kann eine Auswahl zwischen einem autonomen Bewegungsmodus (Auto-Move) und dem manuellen Verfahrmodus betätigt werden. Ist der manuelle Modus ausgewählt, kann mit den darunter liegenden Feldern der Roboter bewegt werden. Im Auto-Move Modus obliegt die Entscheidung, sich zu bewegen dem Agenten. Der Agent kann im Gegensatz zum manuellen Modus auch diagonal verfahren.

5:

Hier kann die serielle Kommunikation gesteuert werden. Im Eingabefenster muss der Port eingegeben werden (z.B. COM4). Darunter ist ein Drop-Down Menü, in dem die Baudrate gewählt werden kann. Mit dem Knopf serial start wird die Kommunikation aufgebaut. Besteht schon eine, wird die alte erst beendet, bevor eine neue aufgebaut werden kann.

6:

Im diesem Ausschnitt kann die Videoquelle gewählt werden. Bei "live" wird auf das aktuelle Kamerabild zugegriffen, bei video playback kann eine Aufnahme genutzt werden. Das ist besonders nützlich, wenn man keinen Zugang zum Airhockey-Tisch hat und somit von der Ferne aus weiter arbeiten kann. In der kleinen Box darunter kann der Kalibrierungsmodus initialisiert werden. Die Box bleibt dann ausgewählt, bis die Kalibrierung erfolgreich abgeschlossen wurde.

7:

In diesem Bereich werden nur Statusmeldungen angezeigt. Er enthält keine Bedienelemente. Die erste Zeile gibt Aufschluss über den Status der Kalibrierung. Die Zweite informiert über die serielle Kommunikation. Die Dritte enthält eine Aussage über den Spielmodus. In der Letzten wird angegeben, mit welcher Frequenz die ganze Anwendung läuft.

8:

Hier sind Eingang und Ausgang des neuronalen Netzwerkes angegeben. Der Vektor enthält die Informationen zum Spielgeschehen aus der Bildverarbeitung und der Pfeil zeigt das Ergebnis der Interferenz an.

### 6.2.2 Programmablauf der GUI-Anwendung

In diesem Abschnitt wird der Ablauf der einzelnen programmtechnischen Schritte erklärt. Zu beachten ist dabei, dass zwei Prozesse parallel ablaufen müssen. Das liegt daran, dass das Interface kontinuierlich überwacht und auf Eingaben überprüft werden muss. Gleichzeitig müssen die anderen Vorgänge zur Bildverarbeitung, zur Kommunikation etc. ablaufen. Der Prozess für die Funktion der Oberfläche und der Prozess für den standardmäßigen Ablauf sind also immer aktiv, beziehungsweise laufen in einer Schleife und werden kontinuierlich wiederholt. Wenn eine Benutzerinteraktion ausgelöst wird, wird diese in die Schleife der Interfacefunktion oder der Standardabläufe eingefügt. Wird beispielsweise ein Haken in der Box zur Kalibrierung erfasst, ändert das den Standardablauf, wird der Knopf für die serielle Kommunikation gedrückt, verlängert das die Schleife des Interfaces. Alle Funktionen, die direkt durch ein Bedienelement hervorgerufen werden, verlängern die Interfaceschleife, der Rest, bei dem nur ein Parameter angepasst wird, wird während des Standardablaufs abgearbeitet.

Funktionen der Interfaceschleife sind:

- manuelle Bewegung
- Start der seriellen Kommunikation
- Wechsel der Videoquelle

Wie die anderen Funktionen abgearbeitet werden ist in der Abbildung 6.4 dargestellt. Im Ablaufgraphen stellen die grünen Rechtecke Programmschritte dar. Haben die Rechtecke links und rechts einen Doppelstrich, bedeutet das, dass diese Prozesse komplexer sind und noch genauer erläutert werden. Die orangen Rauten stehen für eine Entscheidung. Die Sechsecke stellen Anfang und Ende einer Schleife dar.

Zu Beginn des Programms wird die Visualisierung erstellt. Dazu wird ein Objekt der Klasse Window angelegt. Dieses hat durch Vererbung aus der Tkinter Bibliothek die nötigen Methoden und Attribute, um das Fenster mit der mainloop Methode zu erstellen und die beiden Schleifen zu starten.

Im Standardablauf wird als Erstes überprüft, ob schon eine Videoquelle festgelegt wurde. Ist das nicht der Fall, so ist der Durchlauf der Schleife schon vorbei.

Als Nächstes wird geprüft, ob eine Kalibrierung durchgeführt werden soll. Ist das der Fall, wird ein Kalibrierungsversuch unternommen. Scheitert dieser, so wird sofort zum Update des Interfacebildes gesprungen. Ist er erfolgreich, so kann der Standardablauf weiter verfolgt werden.

Die nächste Abfrage ist, ob schon kalibriert wurde. Das ist wichtig, denn es besteht auch die Möglichkeit, dass noch nicht kalibriert wurde und der Haken

## CHAPTER 6. REALER DEMONSTRATOR

---

im Interface noch nicht gesetzt wurde.

Nun erfolgt der Schritt, bei dem die Inputdaten für den Agenten aus dem Kamerabild ermittelt werden.

Wenn Puck und Roboter erkannt werden, kommt es zu einem Spiel. Dabei wird die Ausgabe des Netzwerkes nach einer Sicherheitsüberprüfung ausgeführt.

Ist der Fall eingetreten, dass nur der Roboter erkannt wird, ist davon auszugehen, dass der Puck nicht im Spiel ist. Dann wird der Roboter in eine Ausgangsposition mittig vor dem Tor gefahren, um auf die Fortsetzung des Spiels zu warten.

Da es durchaus vorkommen kann, dass durch die Lichtverhältnisse oder Ähnliches für ein oder zwei Einzelbilder nicht Puck und Roboter erkannt werden, obwohl beide am Tisch sind, wird in diesem Fall der letzte Bewegungsbefehl wiederholt. Dazu wird überprüft, wie lange die Berechnung des letzten Befehls zurückliegt. Ist er zu alt, wird zurück in die Ausgangsposition verfahren.

Wenn der Roboter zu lange nicht mehr erkannt wurde, wird jede Bewegung verhindert, um eine Kollision zu verhindern. Dabei ist es egal, ob der Puck noch detektiert wird.

Als letzter Punkt der Schleife steht das Update des Bilds für das Interface. Das erfolgt immer, außer es gibt keine Videoquelle.

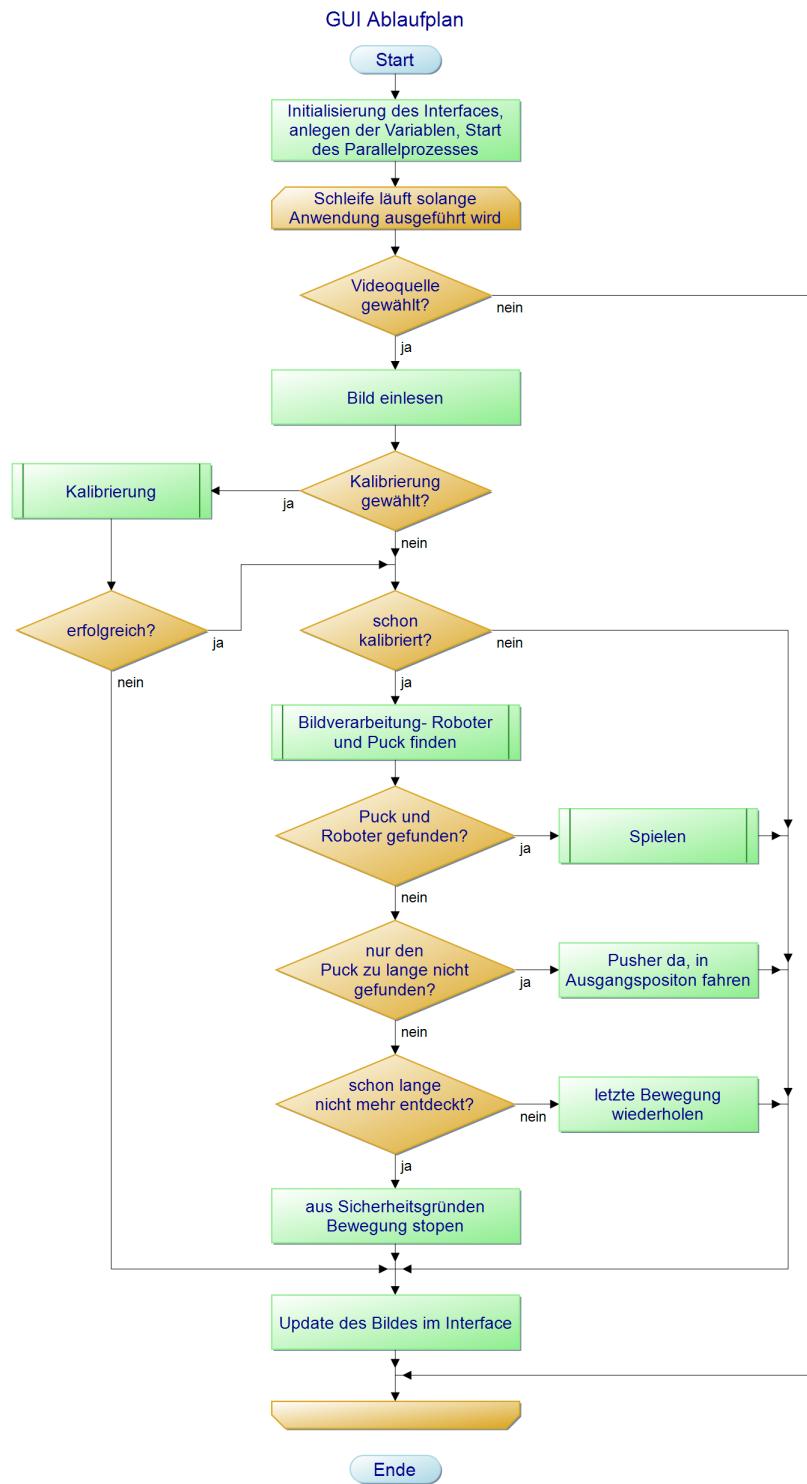


Figure 6.4: Programmablaufplan des Standardablaufes

Der Programmablaufplan in der Abbildung 6.5 zeigt, wie der Ablauf im Unterbereich Spielen, wie er in der Abbildung 6.4 zu sehen ist, ist. Dabei wird als erstes aus den ermittelten Werten aus dem Kamerabild mit dem neuronalen Netzwerk eine vorgesehene Bewegung errechnet. Danach wird überprüft, ob der Agent zu nahe am Rand steht. Ist das nicht der Fall, kann die Bewegung direkt ausgeführt werden. Dafür wird erst das Interface aktualisiert und dann der Bewegungsauftrag seriell abgeschickt. Steht der Agent jedoch gefährlich nahe am Rand, muss überprüft werden, ob die Bewegung noch weiter Richtung Rand führen würde. Wenn das der Fall ist, wird die Bewegung angepasst, bevor sie im Interface angezeigt und abgeschickt wird. Steht der Roboter beispielsweise am unteren Rand und möchte sich nach linksunten bewegen, wird die Bewegung auf links geändert. Möchte er sich nur nach unten bewegen, kommt es zu einem Bewegungsabbruch.

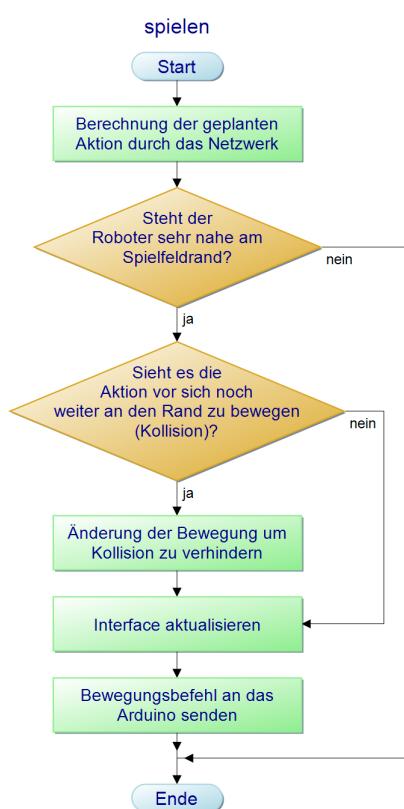


Figure 6.5:  
Programmablaufplan  
des Spielvorganges

### 6.3 Kalibrierung

Bei der Kalibrierung geht es darum, dass Spielfeld vor der Kamera zu vermessen und auf das Spielfeld in der Unity Umgebung zu übertragen. Das ist wichtig, da der Agent in der Softwareumgebung trainiert wurde und dementsprechend auch Größen in Unity Längeneinheiten angegeben werden müssen. Angaben in Metern oder Pixeln müssen also umgerechnet werden. Da vorausgesetzt wird, dass die Kamera senkrecht über dem Spielfeld befestigt ist und andere Verzerrungseffekte vernachlässigbar sind, kann das Problem zweidimensional betrachtet werden. Das bedeutet, dass die Felder zueinander verdreht um die Z-Achse, verschoben um die X- und Y-Achse und skaliert entlang der X- und Y-Achse sind. Eine Verkipfung um die X- und Y-Achsen sowie eine Verschiebung in Z-Richtung fallen weg.

Um all diese Parameter berechnen zu können, müssen auf dem realen Spielfeld mindestens drei Punkte ihrem Gegenstück aus der Unity Version zugeordnet werden.

Als markante Punkte wurden die vier roten Kreise auf dem Spielfeld ausgewählt. Um diese auf dem Kamerabild zu erfassen, muss zuerst ein bestimmter Farbbereich extrahiert werden. Dazu wird das Ausgangsbild in den HSV-Farbraum überführt. Statt der Farbwerte für die Kanäle rot, grün und blau, sind Farben nun in durch die Kategorien Hue (Farbwert), Saturation (Reinheit, Sättigung) und Value (Helligkeit) beschrieben. Da eine Verstärkung der Beleuchtung nun nur noch eine Änderung am V-Wert (Value) zur Folge hat, wird die Separation eines Objekts dadurch viel stabiler.

## CHAPTER 6. REALER DEMONSTRATOR

---

Nachdem der gewünschte vordefinierte Farbraum herausgefiltert wurde, bleibt ein Binärbild, in dem alle Pixel, die im Originalbild in den Farbraum fallen, weiß sind. Mit Hilfe der in OpenCV implementierten HoughCircles Funktion [8] können nun die Mittelpunkte der Kreise bestimmt werden. Dazu ist von Vorteil, dass die Größe vorher schon relativ genau bekannt ist.

Die Mittelpunkte werden danach mit vorher festgelegten Bereichen abgeglichen. Da die Kamera fest moniert ist, sollte nun klar sein, ob ein Kreis linksoben, rechtsoben, linksunten oder rechtsunten ist. Gibt es mindestens drei Übereinstimmungen, kann der Mittelpunkt des Spielfeldes und die Ausrichtung sowie die Skalierung relativ zum Unity-Feld, ermittelt werden. Mit diesem Wissen kann nun mittels zweier Funktionen die Umrechnung von realen in Unity-Koordinaten und umgekehrt durchgeführt werden.

Im Vergleich zum Programmablaufplan in der Abbildung 6.4 ist der aus Abbildung 6.6 deutlich überschaubarer. Er ist bis auf eine Verzweigung geradlinig.

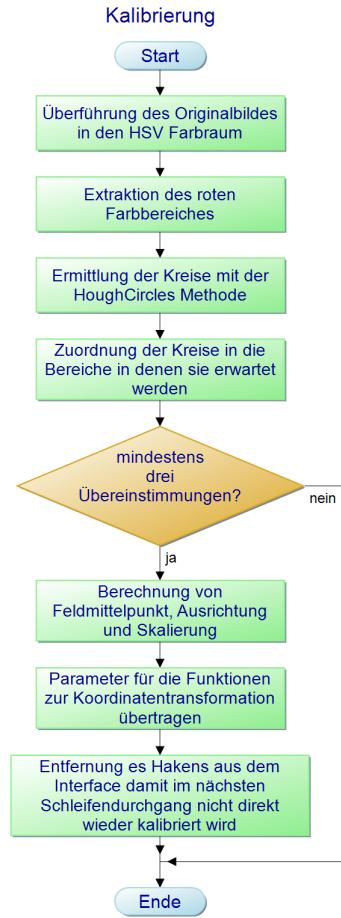


Figure 6.6:  
Programmablaufplan  
der Kalibrierung

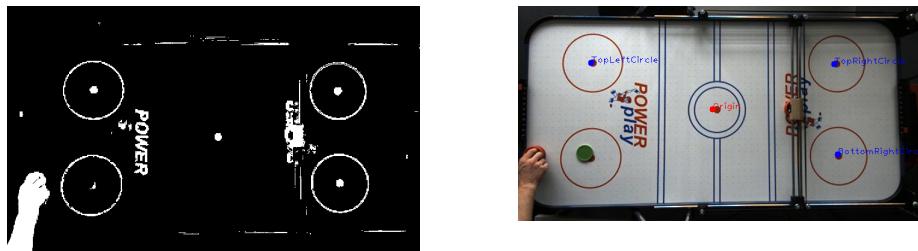


Figure 6.7: Zwischenergebnisse der Kalibrierung

Das linke Bild zeigt das Binärbild, welches entsteht, wenn nur ein bestimmter Farbbereich gewählt wird. Neben den Kreisen wird auch ein Teil der Schrift und der Arm des menschlichen Spielers mit binarisiert. Nach der HoughCircles Methode und der Zuordnung kann der Mittelpunkt des Spielfelds, wie man auf der rechten Seite erkennen kann, ziemlich gut ermittelt werden.

## 6.4 Puck und Robotererkennung

Um am realen Demonstrator den Agenten aus der Simulation nutzen zu können, müssen alle Eingabewerte, die während des Trainings von Unity bereitgestellt wurden, hier anders ermittelt werden. Da viele Sensoren teuer und aufwendig zu implementieren sind, werden hier alle Daten aus den Kamerabildern bezogen. Die benötigten Daten sind neben der Position und Geschwindigkeit des Pucks auch die Position des Roboters. Alle drei Daten sind als X- und Y-Wertepaare nötig. Wie diese Informationen erlangt werden sollen, soll in den nachfolgenden Abschnitten erläutert werden.

Der Vorgang ist bei Puck und Roboter vergleichbar. Die Hauptunterschiede sind die unterschiedlichen Farbräume, der Roboter kann sich nur in seiner Hälfte aufhalten und die Geschwindigkeit ist nur für den Puck zu berechnen. Zwar gibt es noch weitere kleine Unterschiede, beispielsweise bei der Erosion, aber bei beiden Erkennungen wird nach dem gleichen Ablauf verfahren.

Die Grundlage der Bildverarbeitung ist auch hier die Extraktion bestimmter Farbbereiche. Zuerst wird dazu wieder in den HSV-Raum übergegangen, um dann ein Binärbild mit den erwünschten Bereichen in weiß zu erhalten. Da hier nicht mit der HoughCircles Funktion schnell ein Ergebnis erreicht werden kann, wurde danach eine kleine Erosion durchgeführt, um das Binärbild übersichtlicher zu gestalten. Nach dieser Erosion sollten kaum mehr weiße Flecken außer der gewünschten Kontur übrig sein. Aus den verbleibenden Konturen wird dann die Größte herausgesucht. Passt ihre Größe zu der des gesuchten Objekts (Puck oder Roboter), wird ihre Position ermittelt. Hierzu werden die vorkalibrierten Funktionen verwendet. Im Unity Koordinatensystem sind die Werte dann geeignet für den Agenten. Aus der aktuellen Position der letzten Position und der verstrichenen Zeit wird dann noch die Geschwindigkeit berechnet.

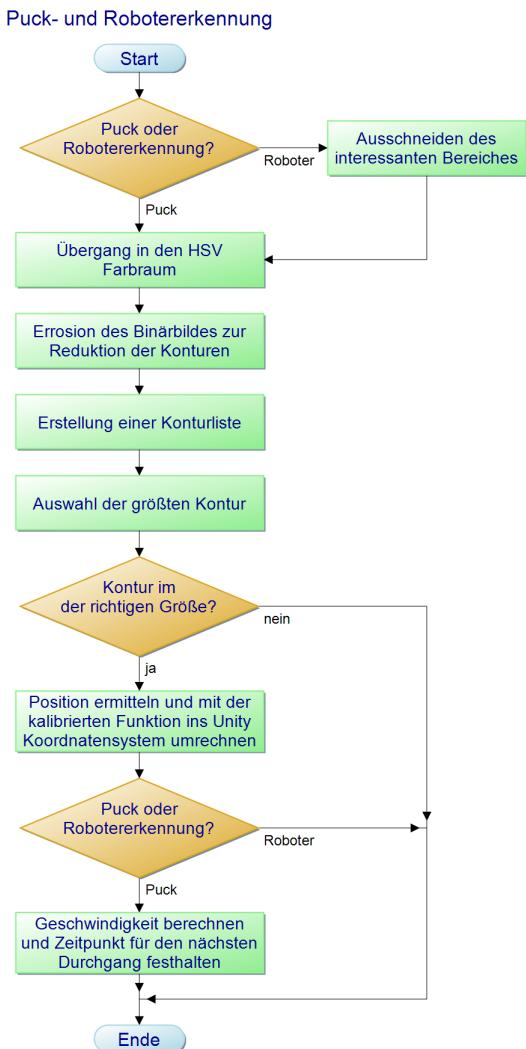


Figure 6.8: Programmablaufplan der Puck- und Robotererkennung

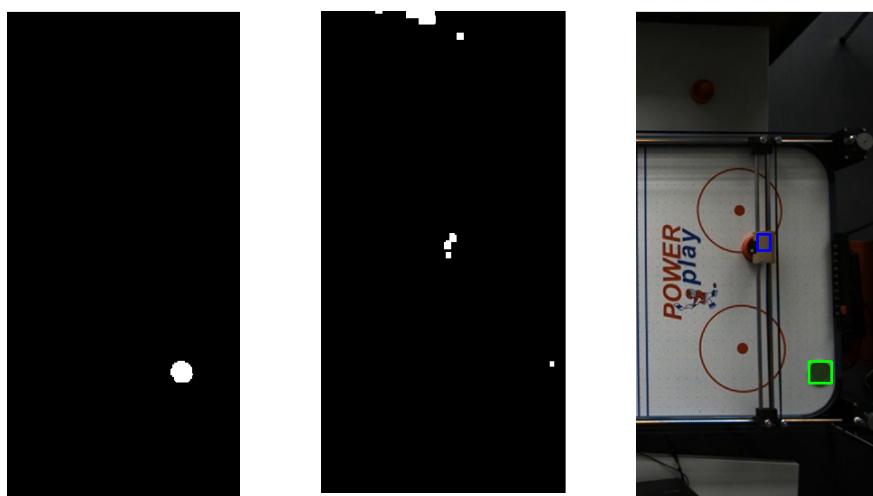


Figure 6.9: Zwischenergebnisse der Puck- und Robotererkennung

In der Abbildung 6.9 ist dreimal der gleiche Feldabschnitt zu sehen. Links ist das erodierte Binärbild aus der Puckererkennung. Hier ist die Kontur des Pucks eindeutig zu erkennen. In der Mitte ist das erodierte Binärbild aus der Robotererkennung zu sehen. Durch schlechte Lichtverhältnisse und eine schlechte Farbwahl für den Roboter ist das Binärbild nicht besonders gut. Da jedoch der Bereich, in dem sich der Roboter befinden kann, bekannt ist, kann die größte Kontur am oberen Bildrand ausgeschlossen werden. Der rechte Teil der Abbildung zeigt, dass Puck und Roboter richtig erkannt wurden.

## 6.5 Spielverhalten

Dieser Abschnitt soll kurz darstellen, wie sich der reale Demonstrator im Vergleich zu seinem digitalen Zwilling verhalten hat.

Ein großes Problem ist durch die Plattenbeschaffenheit aufgetreten. Die Platte ist nicht ganz eben. Dadurch fängt der Puck an manchen Stellen ohne Kontakt an zu gleiten. Solange der Puck schnell unterwegs ist, ist die Ablenkung durch die Unebenheit vernachlässigbar. Ein weiteres Problem, das von der Plattenkrümmung herrührt, ist, dass der Roboter sich an manchen Stellen einklemmen kann. Da die Linearführungen den Roboter nur auf einer Ebene führen, ändert sich also der Abstand zwischen Tischplatte und Führung. Besonders am Rand und in den Ecken kann es deshalb zum Steckenbleiben führen. Manchmal kann sich der Roboter unter Quietschen, Rattern und Schrittverlust an den Motoren wieder befreien, aber selbst dieser Fall ist höchst unerfreulich. An besonders niedrigen Stellen kann es dagegen dazu kommen, dass sich der Puck zwischen Tischplatte und Roboter klemmt. Auch hier ist ein Spielabbruch meist die einzige vernünftige Lösung.

Neben den Defiziten, die durch die Hardware auftreten, kommen noch andere dazu. Der Roboter kann bei Weitem nicht die Bildwiederholungsrate, die im Interface angezeigt wird, auf den Tisch bringen. Ein Teil des Zeitverlustes kommt von der Puck- und Robotererkennung. Ein weiterer Teil tritt bei der Kommunikation mit dem Arduino auf. Dadurch ist die Bewegung deutlich ruckelnder, als es zu erwarten war. Dass es einen kontinuierlichen Action-space gibt, in dem alle möglichen Aktionen mit Ausnahme von Stehenbleiben, nur die maximale Geschwindigkeit durchgeführt werden können, verbessert die Situation nicht besonders.

Insgesamt ist die Leistung am Tisch unbrauchbar. Verglichen mit dem Spiel in Unity, in dem der Agent fast nicht zu besiegen ist, ist dieses Ergebnis noch nicht für eine Demonstration geeignet.

## 7 | Fazit

Zusammenfassend kann gesagt werden, dass das Ziel eines autonom spielenden Airhockeytisches nicht erreicht wurde. Zwar wurden einige gute Zwischenergebnisse hervorgebracht, aber der reale Demonstrator erfüllt seinen Zweck noch nicht.

Im Rahmen der Arbeit wurde die Simulation weiter entwickelt und mit einigem Trainingsaufwand wurden Agenten erzielt, die zeigen, dass die gewählten Netzwerkarchitekturen für die Aufgabe geeignet sind. In der Softwareumgebung wurde das Projekt zufriedenstellen bearbeitet, es gibt aber auch hier noch Steigerungspotenzial.

Die meisten Probleme gibt es noch beim Übergang auf den realen Tisch. Neben den Hardwaremängeln sind auch die Kommunikation und die Latenz durch die Anwendung aus Kapitel 6 nicht optimal. Auch hier könnte noch Aufwand betrieben werden.

Neben den genannten Verbesserungsmöglichkeiten bietet der Airhockey Tisch aber noch andere Ideen an, die in Folgeprojekten interessant werden könnten. Eine Möglichkeit wäre es, einen kontinuierlichen Actionspace für einen weiteren Agenten zu wähle. Das könnte dazu beitragen, dass der Roboter sich am Ende flüssiger bewegt. Auch ein Wechsel auf eine andere Roboterkinematik könnte Vorteile mit sich bringen. Ein Roboter mit Unterarm und Oberarm könnte den Pusher auf der Platte aufliegen lassen und wäre so möglicherweise weniger anfällig gegen Unebenheiten.

## CHAPTER 7. FAZIT

---

### Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit ohne fremde Hilfe und nur unter Verwendung der angegebenen Hilfsmittel selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer Autoren entnommen sind, haben wir kenntlich gemacht.



Martin Haag



Denise Baumann

# Bibliography

- [1] ML-agents toolkit overview. <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md>, letzter Aufruf: 30.04.2022.
- [2] IBM Cloud Education. Unsupervised learning. <https://www.ibm.com/cloud/learn/unsupervised-learning>, letzter Aufruf: 25.04.2022, .
- [3] IBM Cloud Education. Machine learning. <https://www.ibm.com/cloud/learn/machine-learning>, letzter Aufruf: 25.04.2022, .
- [4] jjRobots Ltd. Air hockey robot evo. <https://www.jjrobots.com/the-open-source-air-hockey-robot/>, letzter Aufruf: 24.03.2020.
- [5] Filip Wolski Prafulla Dhariwal Alec Radford John Schulman, Oleg Klimov. Proximal policy optimization. <https://openai.com/blog/openai-baselines-ppo/>, letzter Aufruf: 30.04.2022.
- [6] J. Zico Kolter. Introduction to reinforcement learning. <https://icaps-conference.org/fileadmin/alg/conferences/icaps18/summerschool/lectures/Lecture5-rl-intro.pdf>, letzter Aufruf: 27.04.2022.
- [7] Open Source Physics. Tracker. <https://physlets.org/tracker/>.
- [8] Harvey Rhody. Lecture 10: Hough circle transform. *Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology*, 2005.
- [9] Julia Rieger. Reinforcement learning. <https://aracom.de/reinforcement-learning/>, letzter Aufruf: 27.04.2022.
- [10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.