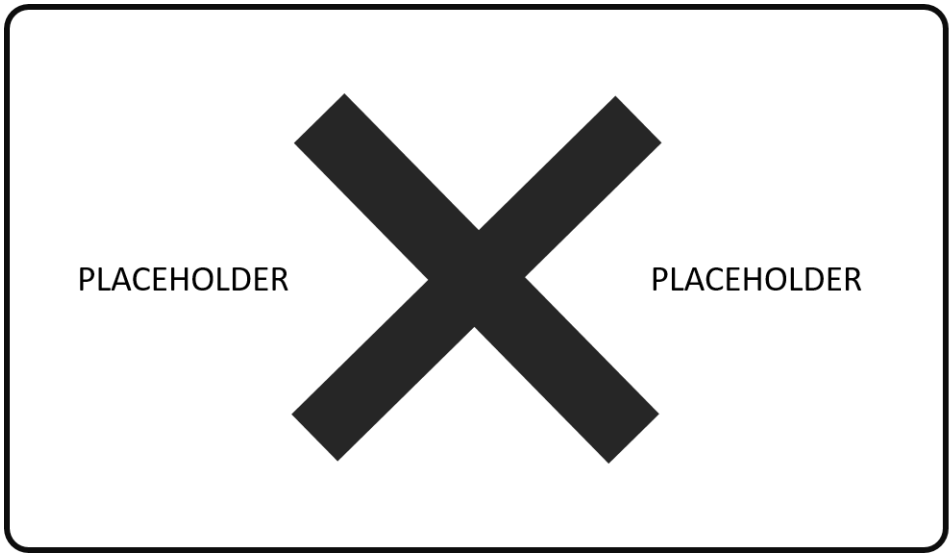

Heilbronn University

Master Project

Thesis Title

Denise Baumann, Martin Haag

supervised by

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

April 29, 2022

Master Thesis

Thesis Title

at

Author:	Denise Baumann, Martin Haag
Matriculation Number:	123456, 194980
Course of Studies:	ASE, MMR
Supervisor:	Prof. Dr. -Ing. Nicolaj Stache & Pascal Graph
Time Frame:	Summer Semester / Winter Semester

Contents

1 | Einleitung

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

1.1 Ausgangssituation

Airhockey ist ein hochdynamisches Geschicklichkeitsspiel, bei dem zwei Personen gegen einander spielen. Hierfür sind die Spieler mit 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 den Puck, als auch die Pusher erreichen, besteht ein immenser Anspruch an die Mechanik, sowie an die informationsverarbeitenden Systeme des gesamten Roboters. 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 [1] Das Zentrum für maschinelles Lernen (ZML) möchte einen selbstlernenden und spielenden AirHockeyRoboter entwickeln. Dafür statteten Maschinenbaustudenten einen Airhockeytisch mit einem Kamerahalter und einem Roboter aus, welcher auf dem Konzept der Firma jjRobots basiert.

2 | 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 wage ich in Frage 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.

2.1 Auswahl der Umgebung

Da bereits aus einer vorhergegangenen Arbeit und 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 Pythonschnittstellte, die für Reinforcemet Learning immer nützlich werden kann kann Unity auch noch mit einer großen Community und sehr guter Dokmentation 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. Eine schnelle, unkomplizierte Möglichkeit mit dem Training zu beginnen.

2.2 Unity

Unity ist eine von Unity Technologies entwickelte multiplattform Entwicklungsumgebung zum Erstellen von Videospielen. Unsere Anwendungist zwar kein Videospiel im herkömmlichne Sinn, aber von der Physikengine können wir trotzdem gebrauch machen. 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.
Wir haben die Unity Version 2020.1.6f1 genutzt.
Die Version des ML-Agents Toolkits, die zum Einsatz kam, lautet 2.1.0-exp.1

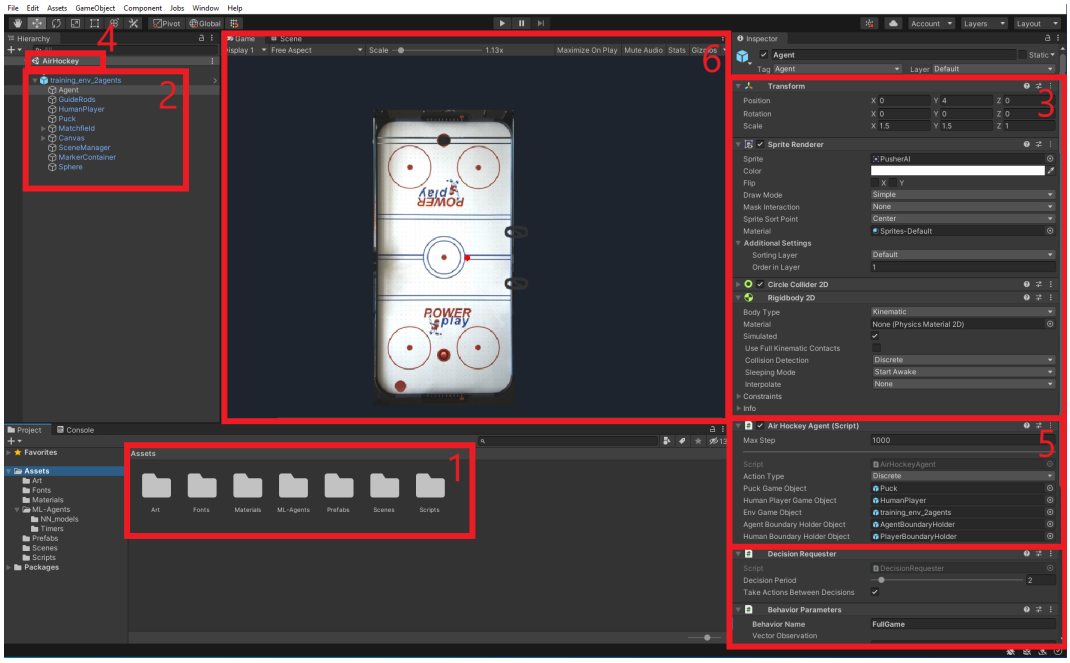

Figure 2.1: Benutzeroberfläche von Unity mit Markierungen

2.2.1 Kurze Einführung

Die kurze Einführung in Unity soll anhand des User Interfaces geschehen. Diese 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 zenrale Konzept 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 Hierariebaum 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 Hierarachiebaums. Zur Spieleentwicklung könnten hier unterschiedliche Level als underschiedliche Scenes interpretiert werden. In unserem Fall sind nur zwei Scenen vorhanden: eine mit einem Spielfeld zum selbst spielen und testen und eine mit acht Feldern zum beschleunigten Training.

- Script (5) :
  Scripts sind auch Components. Sie sind die Möglichkeit selbst Verhalten zu definieren. Es kann sich in Scripts auf GameObjects bezogen werden. Mit ihrer Hilfe können Parameter wie Material oder 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.

2.2.2 Airhockey Projekt

In diesem Unterkapitel werden die einzelnen GameObjects und die wichtigstens 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 2.2.1 zu finden.

Agent :

Das Agent ist eines der zentralen GameObjects. Es hat eine Sprite Renderer Component. Dadurch kann es mit einer .png Datei Visuallisiert werden. Der Agent ist also ein Objekt, dass im Spiel zu sehen ist. Auf der rechten Seite ist zu sehen, wie es im Projekt zu sehen ist.

Figure 2.2: 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 in die anfalleden Rewards entsprechend des Spielverlaufes an das Netzwerk zurück gegeben, als auch die Actions entgegen genommen und damit die Umgebung (Bewegung des Agenten) beeinflusst. Zu Episodenbeginn werden in diesem Script auch die Positionen von Agent und Spieler (Pusher) zurück gesetzt.
Des weiteren beinhaltet das Agent GameObject auch noch eine Decission Requester Componente. Mit ihrer Hilfe wird regelmäßig, entsprechend des Parameters Decision Periode, eine Action vom Netzwerk angefordert. Die Behavior Parameter Componente ist, neben Decison Requester, zur Parametrisierung der Nutzung des ML Agents Toolkits nötig. Hier kann die Dimension sowohl des Actionspaces als auch die der Obseration festgelegt werden. Auch Angaben zur Interferenz 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 diese Objekt auch am Spiel teilnimmt hat es auch die Components Circle Collider und Rigidbody.

Figure 2.3: Ansicht des Pushers in Unity

HumanPlayer hat auch die Components Decission Requester und Behavior Parameters. Damit wird das Selfplay ermöglicht. In unserer Implemetierung wird die Verion des Agenten, die den HumanPlayer bewegt nicht trainiert. Das ML Agents Toolkit wird hier nur zur Interferenz genutzt.
Das Objekt enthält auch ein Script. Im Gegensatz zum Airhockey Agents Script werden hier aber eine Rewards zurück gegeben sondern nur die Observations und die Actions behandelt. Das reicht auch aus, denn dieser Agent soll ja garnicht ständig mittrainieren, sondern nur das Selfplay ermöglichen. Wenn er zu schwach wird, wird einfach die stärkere Version vom Agent 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 herrausgefordert werden.

Puck :

Abhänig vom Spielszenario wird in diesem Script die Startposition und die Startgeschwindigkeit des Pucks festgelgt. Auch der Spielstand wird hier

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

Figure 2.4: Ansicht des Pucks in Unity

mitgezählt. Die Circle Collider Componente des Pucks erlaubt es ein Event bei Kollision mit anderen Objekten auszulösen. Wenn der Spielfeldrand am Tor des Agenten getroffe wird, kann das mit Hilfe eines GameObjects ohne Renderer 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 Renderer Component und deshalb in der Spielansicht unsichtbar. Sie sind nur wegen ihrer Position interessant. Sie dienen um das Spielfeld in Zonen einzuteilen. Die zulässigen Positionen von Agent, HumanPlayer und Puck werden damit auf das Spielfeld oder die jeweilige Hälfte limitiert. Die GameObjekts, die zum Tore zählen genutzt werden sind auch Childobjekte (hierarchisch untergeordnete Objekte) des Matchfieldes.

Figure 2.5: 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 Zecken:

- Das ganze Spielfeld kann einfacher kopiert und verschoben werden. Die Positionen der Childobjekte bleiben beim Verschieben des übergeordneten Objekts relativ zu diesem unverändert. Felder könne 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 wir im Kapitel 2.2.2 erläutert.

3 | 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 2.2.2 nochmal genauer beleuchtet.

3.1 Rewards und Umgebungsparameter

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

- taskType
  Legt fest, welcher Spielmodus gewählt wird. Siehe dazu Abschnitt 3.2.
- 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 4.1 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 gegeben, wenn der Puck in das Tor des Agenten trifft. Da es ein Gegentor ist sollte der Reward negativ gewählt werden.
- agentGoalReward
  Dieser Reward wird gegeben, wenn der Puck in das Tor des HumanPlayer trifft.

- avoidBoundaries
  Dieser Reward wird gegeben, wenn der Roboter die Bande berührt. Er sollte negativ sein.

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

- stayCenteredReward
  Dieser Reward belohnt es weder links noch rechts am Spielfeldrand zu sein. Damit kann das Verteidigungsverhalten verbessert werden. Der Reward wird zu jedem Zeitschritt gegeben und sollte deshalb sehr niedig gewählt werden. Der Reward sinkt proportional mit der Distanz vom Rand.

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

- encouragePuckMovement
  Dieser Reward belohnt Puckbewegungsgeschwindigkeit. Er wird zu jedem Zeitschritt gegeben 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, das der Reward encouragePuckContact jedes mal wenn er gegeben wird halbiert wird. Damit wird verhindert, dass der Roboter alleine spielt oder den Puck einklemmt um den encouragePuckContact Reward auszunutzen.

- playForwardReward
  Dieser Reward wird gegeben, wenn der Puck von hinten getroffen wird, also in die richtige Richtung geschossen wird.

- negplaybackReward
  Dieser Reward wird gegeben, 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 gegeben. Er soll langweiliges Zeitspiel verhindern. Dazu muss er negativ sein.

- negMaxStepReward
  Dieser Reward wird gegeben, wenn die Maximallänge einer Episode erreicht wird und das Spiel deshalb abgebrochen wird(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 er in jedem Zeitschritt geholt werden kann.

- behindPuckReward
  Dieser Reward ist exklusiv für das Training des Spielmoduses Defending. Er wird jedes mal gegeben, wenn ein Schuss erfolgreich verteidigt wurde.

3.2 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 traininiert werden kann.

- Reaching
  In diesem Spielmodus ist es das Ziel 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 gestopt. Die Episode endet mit der Kollision von Agent und Puck oder beim Erreichen der maximalen Simulationsschritten. 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 Tor 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 wiede 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 Geschwindgkeit auf das Feld gesetzt wird. Der Puck wird hier von der Seite des HumanPlayer aus mit einem zufälligen Bewegungswinkel zwischen 70 und -70 auf die Seite des Agenten geschossen. Auch die Startposition wird variiert. 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 zurück bewegt. 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 unseres Masterprojekts. 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 3.1: Übersicht über die Spielmodi

3.3 Trainingsstrategie und Zwischenergebnisse

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

3.3.1 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 wiederspiegelt, muss der Agent einige Vortrainings abschließen um so Eigenschaften nacheinander zu erlernen. Dazu haben wir uns einen Stufenplan der in der Abbildung 3.3.1 zu sehen ist, erarbeitet.

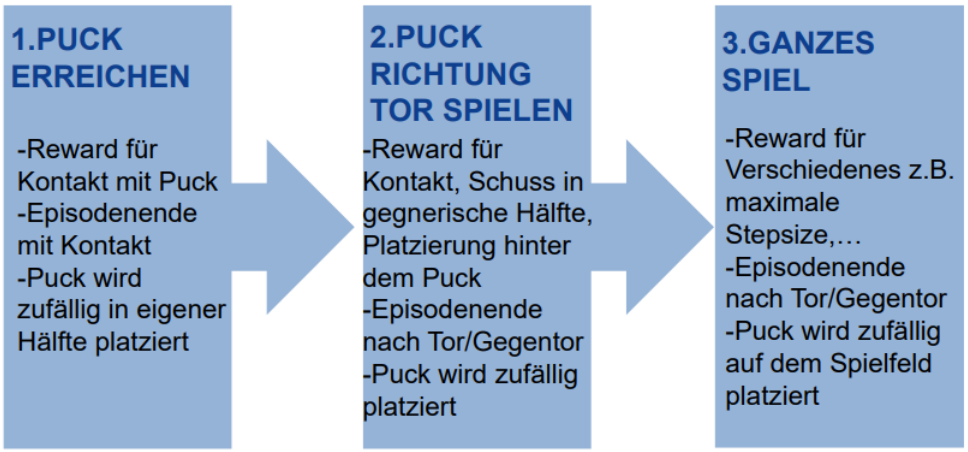

Figure 3.2: Stufenplan des Trainings

- 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 so noch nicht trainiert werden.

- 2. Puck richtung Tor spielen
  Bei diesem Schritt wird auf dem vortraininierten 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 Hautrewards genutzt. Es hat sich jedoch gezeigt, das 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 gegebe werden.

- 3. Ganzes Spiel
  Diese Stufe im Plan ist mit Abstand die zeitintensivste. 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 Herrausfordernung 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 entgegen wirken.
  Dadurch das oft unvorhergesehenes schlechtes Verhalten erlernt wird ist es nach unserer Erfahrung wichtig den vorherigen Stand des Agenten zu sichern und gegebenenfalls den Vortschritt zu verwerfen weil es ehr ein Rückschritt ist. Dann kann mit einem neuen Reward- oder Parametersatz oder mit einem anderen Gegner ein weiterer Versuch vom gesicherten Stand aus probiert werden.

3.3.2 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 Hyperparemetern verglichen. Nach Abwägung der Trainingsgeschindigkeit gegen die Komplexität wurde sich dann für die Parametersätze, die die in den Konfigurationsdateien zu finden sind, entschieden.
Das ML-Agents Toolkit stellt sowohl PPO als auch SAC Agenten zur Verfügung. Um nicht unnötig Ressourcen zu verschwenden macht es jedoch keine Sinn zwei unterschiedliche Agenten zu trainieren. Deshalb haben wir um uns entscheiden sowohl einen PPO als auch einen SAC Agenten jeweils das gleiche Einstiegstraining durchlaufen zu lassen. Anhand der Leistung dabei haben wir dann entschieden mit welchem Netzwerk wir weiter trainieren.

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 wir nur mit dem encoragePuckContact 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 hinzu genommen um das Spielverhalten zu formen. Als Gegner wird ein anderer PPO Agent verwendet, der vorher schon etwas besser traininert 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 HumanPlayer 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 Vorraussetzungen wurde das Training bis 4M fortgesetzt

Die Ergebnisse dieser Trainings sind im folgenden Dargestellt. Im den Abbildungen sind jeweils die Graphen des PPO-Agenten und des SAC-Agenten übereinander gelegt. Der linke Graph zeigt dabei die Entwicklung des komulativen 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 ver vorrangegangenen auch enthalten.

1. Trainig bis 500k

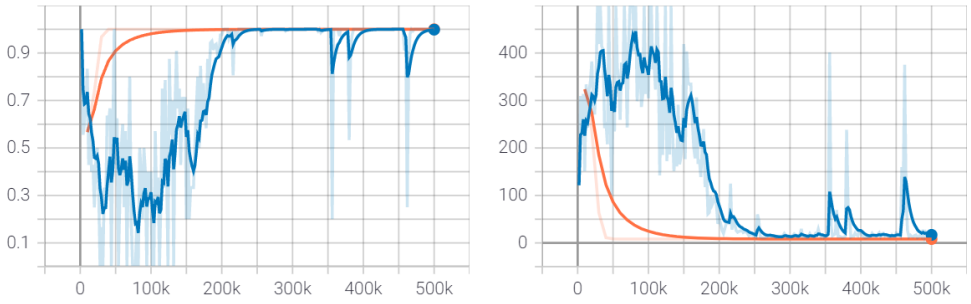

Figure 3.3: Orange: PPO-Agent, Blau: SAC-Agent

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 Unityumgebung bei der Aufgabe sind sie vom Verhalten her praktisch nicht zu unterscheiden. Beide bewegen sich auf sehr kurzem Weg auf den Puck zu.

2. Trainig bis 1M

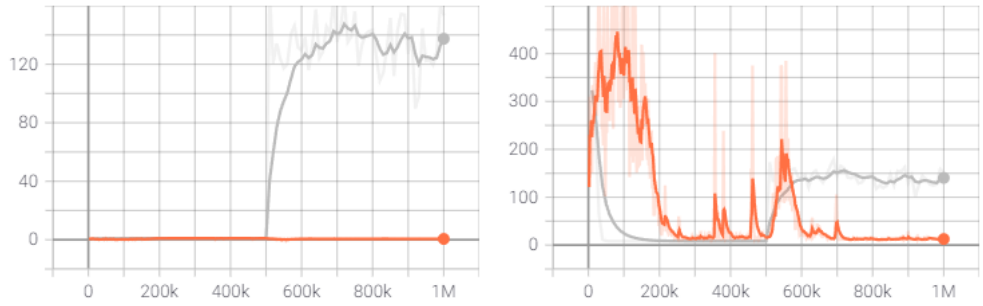

Figure 3.4: Grau: PPO-Agent, Orange: SAC-Agent

In diesem Trainingsabschnitt hat der PPO-Agent nach Reward klar den Vorteil. Jedoch kann man an der Episodenlänge sehen, dass die Aufgabe nicht zufriedenstellent erlernt wurde. Da die Episode beim Puckkontakt beendet wird sollten sie sehr kurz ausfallen. Beim Beobachten des Spielverhalten ist dieses Verhalten auch festzustellen. Der PPO-Agent neigt dazu in der Nähe des Pucks in einen Zitterzustand überzugehen. 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 selte und ist damit insgesamt besser als der PPO-Agent nach diesem Training in der zugehörigen Aufgabe.

3. Trainig bis 2M

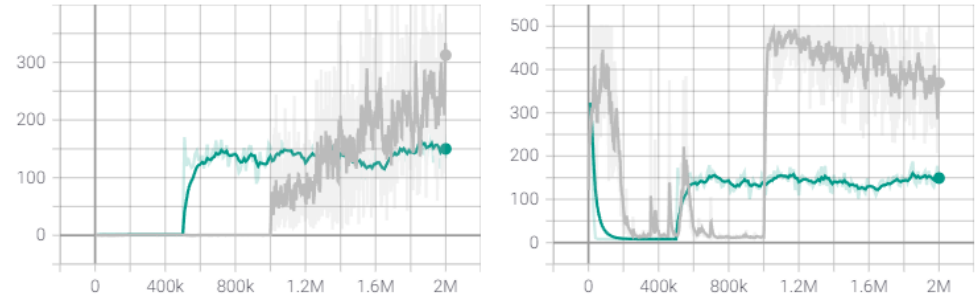

Figure 3.5: Grün: PPO-Agent, Grau: SAC-Agent

Der SAC-Agent hat in dieser Trainingsperiode seinen Rückstand in der Kategorie kumulativer Reward nicht nur verkürzt, sondern hat PPO sogar überholt. Die Episodenlänge hat sich dabei erhöht. Da das Trainig im FullGame Modus erfolgte und diese Agenten jetzt dafür geeignet sein sollten ist eine gute Möglichkeit sie zu vergleichen das direkte Spiel gegeineinander. Hier hat der PPO-Agent eindeutig die Nase vorne. Er schlägt den SAC-Agenten mit 10:5. Außerdem ist sein Spielverhalten deutlich besser. Währen 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. Trainig bis 3M

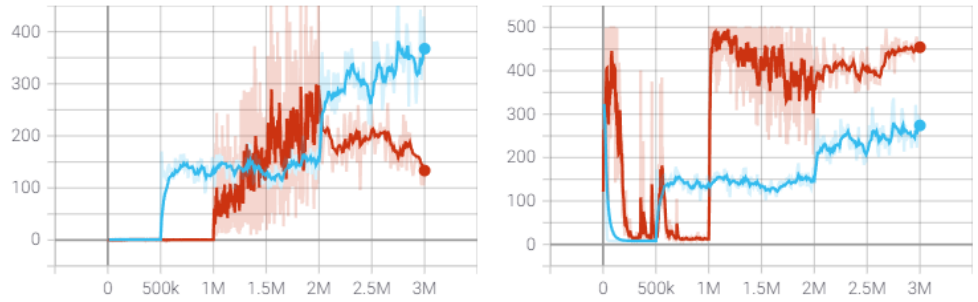

Figure 3.6: Blau: PPO-Agent, Orange: SAC-Agent

Bei diesem Training überholt der PPO-Agenten den SAC- Agenten in Sachen kumulativer Reward wieder. Auch beim Spielverhalten ist er ihm noch überleben 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. Trainig bis 4M

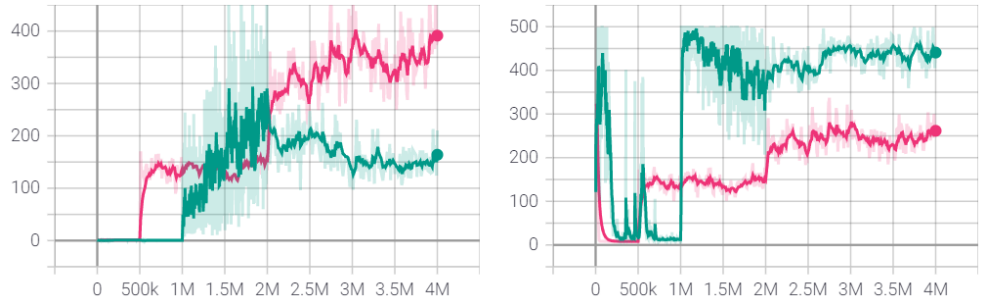

Figure 3.7: Rosa: PPO-Agent, Grün: SAC-Agent

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 vorraus.

Insgesamt haben unsere Erfahrungen gezeigt, dass der PPO-Agent gegenüber dem SAC-Agnenten in unserem Fall im Vorteil ist. Bei genau gleichem Training hat er ein weit besseres, aktiveres, inteligent werkenderes Spielverhalten entwickelt. Zusätzlich hat es den Anschein, als ob sich die Verhaltensentwicklung beim PPO-Agenten besser abschätzen lässt. Die erwünschten Verhaltensmuster sind dadurch einfacher zu erreichen. Aus diesen Gründen haben wir für den größten Teil unserer Trainingsbemühungen mit einem PPO-Agenten der Parametrisierung des hier verwendeten Netzwerkes gearbeitet.

3.4 Trainingsverlauf und Ergebnis

Der vorhergegangene Abschnitt 3.3.2 hat bereits einen guten Einblick über das vorgehen beim Training gegeben. Dieser soll noch einige Informationen zur Gegenerwahl, zum Fortschritt, zu den Kriterien eines guten Agenten und zu den besten bereits erzielten Ergebnissen geben.

3.4.1 Gegnerwahl

Im letzten Abschnitt war zu erkennen, dass die ersten Trainings im Reaching 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 Gegenspieler durchgeführt. Um zu verhindern, dass der Agent entweder nichts lernt (Gegner zu schwach) oder sich weigert den Puck rüber zu spielen weil er einen gefährlichen Schuss befürchtet (Gegner zu stark) muss ein ade quater 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 selbt. Dabei ist darauf zu achten, dass man das Netzwerk des Agente nicht zu oft oder zu selten auf den HumanPlayer überträgt. Tut man es zu oft läuft man Gefahr, dass die Gegenspieler "zusammen spielen" und so mit zum Beispiel einem Kontaktreward schnell einen Torreward obsolet machen. Um einen solchen Effekt zu erzielen muss das Netzwerk aber sehr oft übertragen oder gar für beide Spieler gleichermaßen trainiert werden. Tut man es zu selten kann ein Training ohne vernünftige Herrausforderung an Effizienz verlieren. Ein weiteres Problem des Selfplay ist, dass man zu Beginn gar kein Ergebins hat, dass als Gegner verwendet werden kann. Deshalb ist in unserem Projekt auch ein festprogrammierter Spieler implementiert. Dieser bewegt sich stets hinter den Puck wenn auf seine Seite gespielt wird und spielt 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.

3.4.2 Qualitätskriterien

Um einen guten Agenten traininieren zu können muss man zuerst wissen, was ein guter Agent ist. Im Gegensatz zu einem Läufer, denn man nur nach einem Kriterium bewertet (seiner benötigten Zeit um eine Strecke zurück zu legen) kann ein Agent so einfach objektiv bewertet werden.
Eine erstrebenswerte Eigenschaft ist, 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 Warscheinlicheit 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, haben wir uns darauf beschränkt die Tordifferenz der Agenten zu bewerten. Das haben wir hauptsächlich durch das 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 traininieren. Bei diesen Testspielen hat sich herrausgestellt, dass bei den fortgeschrittneren Agenten einige dabei, die wir manuell nicht mehr besiegen können und teilweise hoch gegen uns gewonnen haben. Selten hat sich eine Niederlage so gut angefühlt.
Neben der Fähigkeit viele Tore zu schießen ohne viel zu kassieren ist auch die Ästethik der Bewegung nicht unwichtig. Da das Ziel des Projekts ein Demostrator ist, ist ein nachvollziehbares Spielverhalten sinnvoll. Ein weiterer Punkt ist die Lärmbelastung. Auch wenn in Unity zittern keinen Nachteil mit sich bringen kommt es dadurch am realen Tisch durch die Motoren zu schrillen, unangenehmen Geräuschen. Das sollte beim Training beachtet werden.

4 | 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 Bedierner des Demonstrators. Alle Bildverarbeitungsschritte wurden in Python mit der OpenCV Bibliothek durchgeführt.

4.1 Geschwindigkeitsmessungen

Bei den Geschwindigkeitsmessungen wurden die Maximalgeschwindigkeit von Roboter und Puck gemessen. Während die Maximalgeschwindigkeit des Roboters relativ einfach erreicht werden kann ist die des Pucks abhänig vom Spieler. Unsere Messung wurde deshalb an einem Schuss durchgeführt, denn wir nach bestem Gewissen als ausreichend schnell bewerten würden. Gemessen wurde anhand von Videomaterial der Kamera. Mit dem Programm Tracker des Open Source Physics Projekts [2] wurde das Material analysier. 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 ist wurde als Maßstab nicht die Abmessung in Metern des realen Demonstrators genutzt sondern die Länge in Unitylängeneinheiten. Die Ergebnisse sind wie folgt ausgefallen:

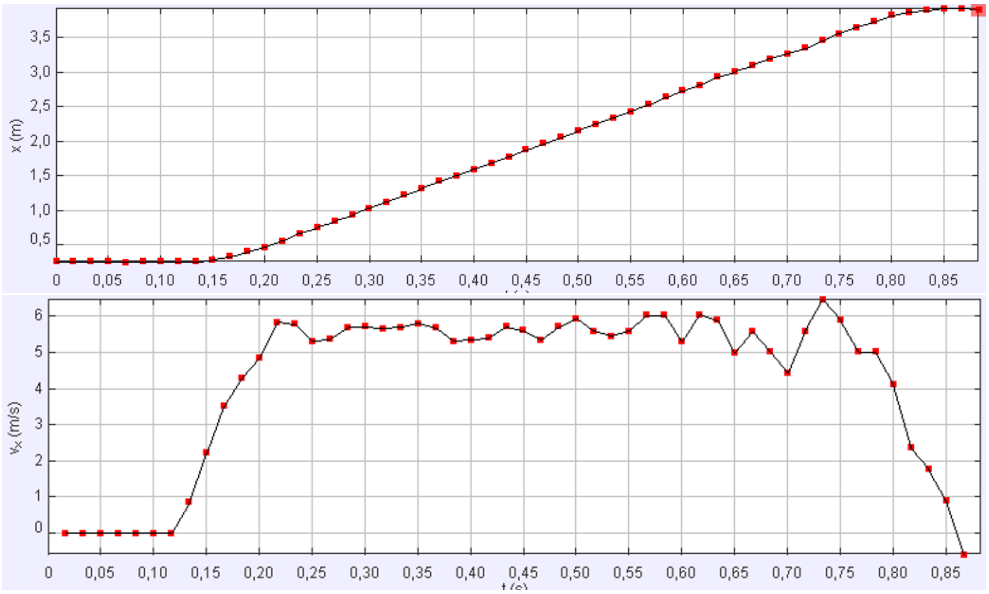

Figure 4.1: Messwerte des Roboters

Das obere Diagramm in Abbildung 4.1 zeigt den zurückgelegten Weg des Roboters über die verstrichene Zeit seit Beginn der Messung. Das untere zeigt den Geschwindigkeitsverlauf über die 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. Rechnent man die Durchschnittsgeschwindigkeit über den Zeitraum von 0.2 bis 0.75 Sekunden aus so erhält man bei einem zurückgelegten Weg von 3.2 (LE)Längenenheiten eine Geschwindigkeit von 5.8 LE/s. Nach Rundung wurde mit dem Wert 6 LE/s traininiert.

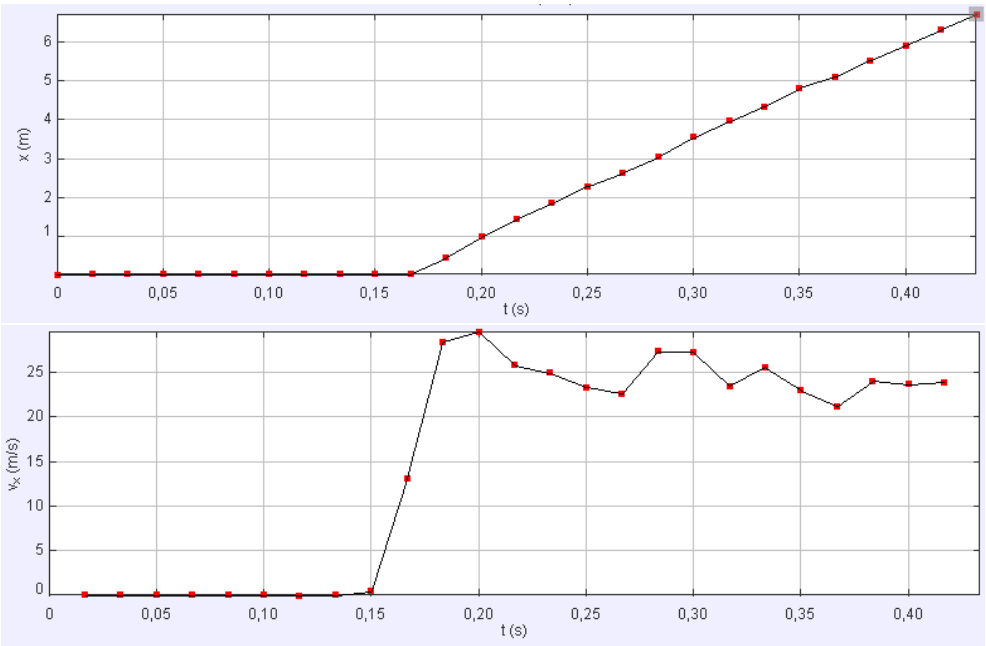

Figure 4.2: Messwerte des Pucks

Diese Diagramme sind von der Messung der Bewegung des Pucks. Auch hier zeigt da obere den Verlauf des zurückgelegten Weges und das untere den Geschwindigkeitserlauf über die 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.

4.2 Benutzeroberfläche

Das graphical use 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.

4.2.1 Ansicht der Oberfläche

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

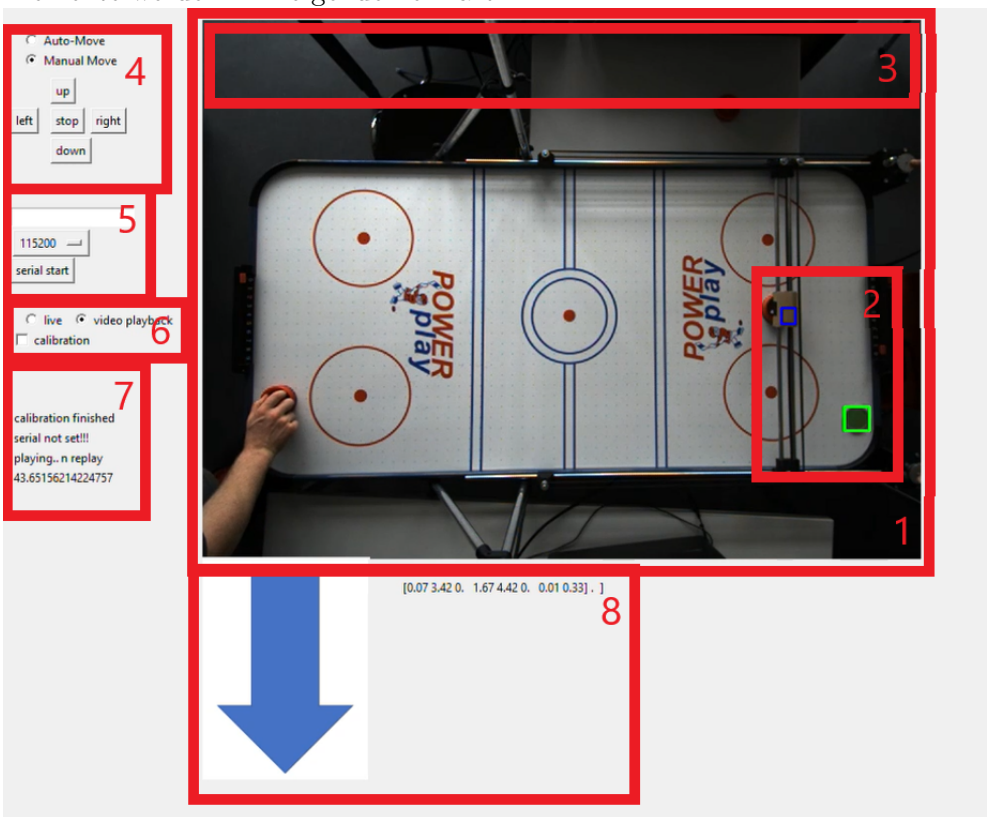

Figure 4.3: Benutzeroberfläche

1:
Hier ist das Kamerabild oder eine Videosequenz zu sehen. Das Bild wird wird, je nach Zustand, um einige Informationen ergänzt.
2:
Eine Ergänzung zum Kamerabild ist die Markierung von Roboter und Puck sofern sie erkannt wurden. Der Puck wird mit einem grünen und der Roboter mit einem blauen Rechteck markiert.
3:
In diesem Bereich wird eine Meldung gegeben wenn, wenn Puck oder Robote 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 gemacht 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 Eingabefenste 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 wird.
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 zuhause am Schribtisch am Projekt gearbeitet werden muss und man unabhänig vom Zugang zum Airhockey-Tisch sein will. In der kleinen Box darunter kann der Kalibrierungsmodus initialilisiert werden. Die Box bleibt dann ausgewählt bis die Kalibrierung erfolgreich abgeschlossen wurde.
7:
In diesem Bereich werden nur Statusmeldungen gemacht. 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 Ergebniss der Interferenz an.

4.2.2 Programmablauf der GUI-Anwendung

In diesem Abschnitt wird der Ablauf der einzelnen programmtechischen Schritte erklärt. Zu beachten ist dabei, dass zwei Prozess 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 standartmäßigen Ablauf sind also immer aktiv, beziehungsweise laufen in einer Schleife und werden koninuierlich wiederholt. Wenn eine Benutzerinteraktion ausgelöst wird, wird diese in die Schleife der Interfacefunktion oder der Standartabläufe eingefügt. Wird beispielsweiße ein Haken in der Box zur Kalibrierung erfasst ändert das den Standartablauf, wird der Knopf für die serielle Kommunikation gedrückt verlängert das die Schleife des Interfaces. Alle Funktionen, die direkt durch ein Bedienelement hervorgerufen weden verlängern die Interfaceschleife, der Rest, bei dem nur ein Parameter angepasst wird, wird während des Standartablaufen abgearbeitet.
Funktionen der Interfaceschleife sind:

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

Wie die anderen Funktionen abgearbeitet werden ist in der Abbildung 4.4 dargestellt. Im Ablaufgraphen stelle die grünen Rechtecke Programmschritte dar. Habe die Rechtecke links und rechts einen Doppelstrich bedeutet das, dass diese Prozesse komplexer sind und noch genauer erläutert werde. Die orangenen Rauten stehen für eine Entscheidung. Die Sechsecke stellen Anfang und Ende der Schleife dar.

Zu Beginn des Programms wird die Viualisierung 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 Standartablauf 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 Standartablauf 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

im Interface noch nicht gesetzt wurde.
Nun erfolgt der Schritt, bei dem die Input Daten für den Agenten aus dem Kamerabild ermittelt werden.
Wenn Puck und Roboter erkannt werden kommt es zu Spielen. Dabei wird die Ausgabe des Netzwerkes nach einer Sicherheitsüberprüfung ausgeführt.
Ist es der Fall das 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 fahren und auf die Fortsetzung des Spiels warten.
Da es durchaus vorkommen kann, dass durch die Lichtverhältnisse oder Ähnlichen 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 zurück liegt. Ist er zu alt wird 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.

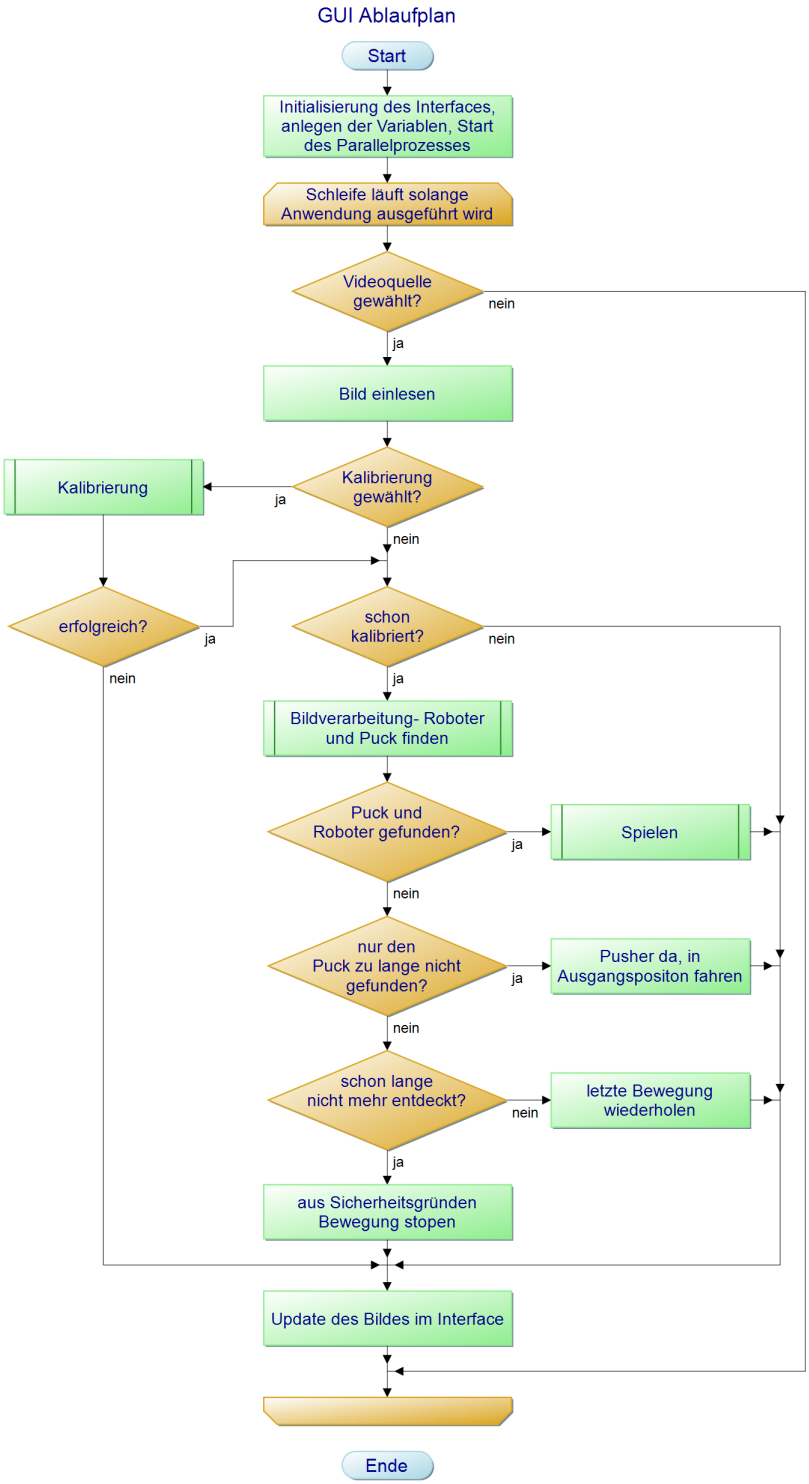

Figure 4.4: Programmablaufplan des Standartablaufes

4.3 Kalibrierung

Bei der Kalibrierung geht es darum das 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 traininiert wurde und dem entsprechend auch Größen in Unity Längeneinheiten angegeben werden müssen. Angaben in Metern oder Pixeln müssen also umgerechnet werden. Da wir davon ausgehen können, dass die Kamera senkrecht über dem Spielfeld hängt und andere Verzerrungseffekte vernachlässigbar sind, können wir das Problem zweidimensional betrachten. Das bedeutet, dass die Felder zueinander verdreht um die Z-Achse, verschoben um die X- und Y-Achse und scaliert entlang der X- und Y-Achse sind. Eine Verkippung 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 haben wir die vier roten Kreise auf dem Spielfeld gewählt. Um diese auf dem Kamerabild zu erfassen muss zuerst ein bestimmter Farbberich 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, Sattigkeit), und Value (Helligkeit) beschrieben. Da eine Verstärkung der Beleuchtung nun nur noch eine Änderung am V-Wert (Value) zur Folge hat wird die Seperation eines Objekts dadurch viel stabiler.

Nachdem der gewünschte vordefinierte Farbraum herausgefiltert wurde bleibt ein Binärbild, in dem alle Pixel die im Orginalbild in den Farbraum fallen, weiß sind. Mit hilfe der in OpenCV implementierten HoughCircles Funktion [3] 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 Berichen abgeglichen. Da die Kamera fest moniert ist sollte nun klar sein, ob ein Kreis links oben, rechts oben, links unten oder rechts unten ist. Gibt es midestens drei Übereinstimmungen kann der Mittelpunkt des Spielfeldes und die Ausrichtung sowie die Skalierung relativ zum Unityfeld ermittelt werden. Mit diesem Wissen kann nun mittels zweier Funktionen die Umrechnung von realen in Unitykoordinaten und umgekehrt durchgeführt werden.
Im Vergleich zum Programmablaufplan in der Abbildung 4.4 ist der aus Abbildung 4.5 deutlich überschaubarer. Er ist bis auf eine Verzweigung geradlinig.

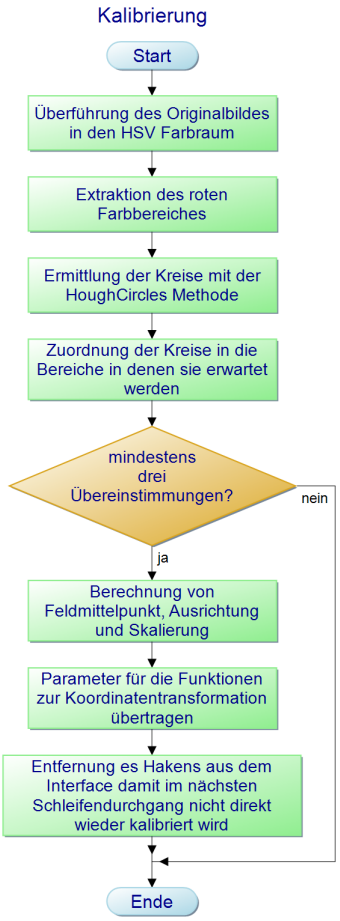

Figure 4.5: Programmablaufplan der Kalibrierung

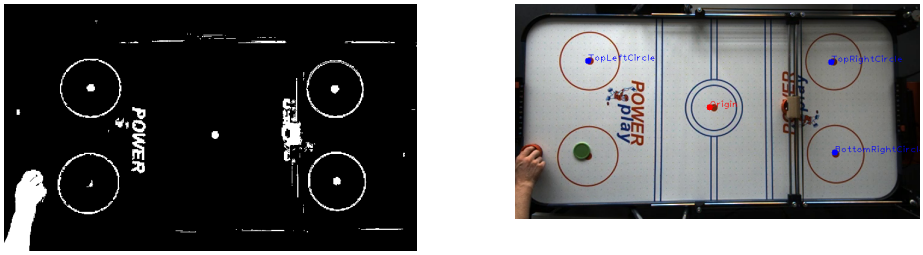

Figure 4.6: Zwischenergebnisse der Kalibrierung

Das linke Bild das Binärbild, das 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.

4.4 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 aufwändig zu implementieren sind haben wir uns entschlossen alle Daten aus den Kamerabildern zu beziehen. Die benötigten Daten sind Position des Pucks, Geschwindigkeit des Pucks und Position des Roboters. Alle drei Daten sind als X- und Y-Wertepaar nötig. Wie diese Imformationen erlangt werden soll in den folgen Abschnitten klar werden.

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

Grundlage der Bildverarbeitung ist auch hier die Extaktion 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 Ergebniss erreicht werden kann wir danach eine kleine Errosion durchgeführt um das Binärbild übersichtlicher zu machen. Nach dieser Errosion 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 gesuchte Objekts (Puck oder Roboter) wird ihre Positon ermittelt. Hierzu werden die vorkalibrierten Funktionen verwendet. Im Unity Koordinatensystem sind die Werte dann geeignet für den Agenten. Aus der aktuellen Position, der letzen Position und der verstrichenen Zeit wird dann noch die Geschwindigkeit berechnent.

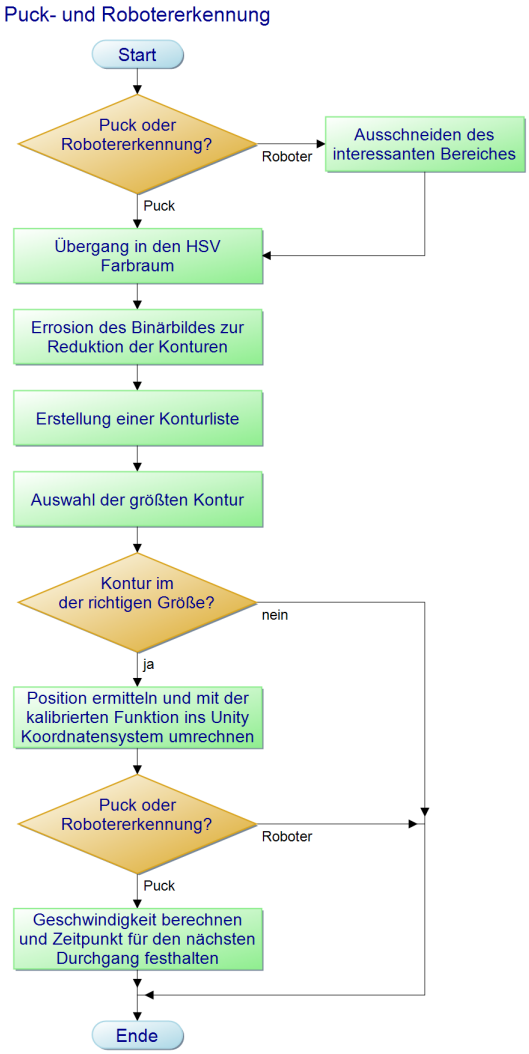

Figure 4.7: Programmablaufplan der Puck- und Robotererkennung

Figure 4.8: Zwischenergebnisse der Puck- und Robotererkennung

In der Abbildung 4.8 ist dreimal der gleiche Feldabschnitt zu sehen. Links ist das errodierte Binärbild aus der Puckerkennung. Hier ist die Kontur des Pucks eindeutig zu erkennen. In der Mitte ist das errodierte 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.

4.5 Spielverhalten

Dieser Abschnitt soll kurz darstellen wie sich der reale Demonstrator im Vergleich zu seinem digtalen Zwilling verhalten hat.
Ein großes Problem ist durch die Plattenbeschaffenheit aufgetretten. 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 her rührt, ist das der Roboter sich an manchen Stellen einklemmen kann. Da die Linerfühungen den Roboter auf einer Ebene führen ändert sich also der Abstand zwischen Tischplatte und Führung. Besonders am Rand und in den Ecken kann es deshalb zum stecken bleiben 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 auftretten 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 ruckeliger als es zu erwarten war. Das es einen kontinuierlichen Actionspace gibt, in dem alle möglichen Aktionen, mit Außnahme von Stehenbleiben, Vollgas sind verbesset die Situation nicht besonders.
Insgesamt ist die Leistung am Tisch unbrauchbar. Verglichen mit dem Spiel in Unity, dass fast nicht zu gewinnen ist, ist dieses Ergebnis noch nicht für eine Demonstration geeignet.

Bibliography

[1] jjRobots Ltd. Air hockey robot evo. https://www.jjrobots.com/the-open-source-air-hockey-robot/, letzter Aufruf: 24.03.2020.

[2] Open Source Physics. Tracker. https://physlets.org/tracker/.

[3] Harvey Rhody. Lecture 10: Hough circle transform. *Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology*, 2005.