HAW HAMBURG

BACHELOR THESIS
Alexander Wessels

Entwicklung und prototypischer Aufbau eines durch Reinforcement Learning gesteuerten Flipper-Automaten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Alexander Wessels

Entwicklung und prototypischer Aufbau eines durch Reinforcement Learning gesteuerten Flipper-Automaten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marc Hensel
Zweitgutachter: Prof. Dr. Ulrike Herster

Eingereicht am: 18. März 2024

Alexander Wessels

Thema der Arbeit

Entwicklung und prototypischer Aufbau eines durch Reinforcement Learning gesteuerten Flipper-Automaten

Stichworte

Reinforcement Learning, OpenAI Gym, Mikrocontroller, Schrittmotoren

Kurzzusammenfassung

Die Arbeit behandelt die Entwicklung eines Deep Learning Agents, welcher die Flipper eines Flipper-Automaten steuert. Zusätzlich wird ein Prototyp entwickelt.

Alexander Wessels

Title of Thesis

Development and prototypical construction of a reinforcement learning-controlled pinball machine

Keywords

Reinforcement Learning, OpenAI Gym, Microcontroller, Stepper motors

Abstract

The thesis deals with the development of a deep learning agent that controls the rackets of a pinball machine. In addition, a prototype is developed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Abkürzungsverzeichnis

FPS Frames Per Second

GPIO General Purpose Input Output

IC Integrated Circuit

RAM Random Access Memory

PWM Pulse-Width Modulation

DQN Deep Q-Network

UML Unified Modeling Language

DIP Dual In-line-Package

1 Einleitung

In diesem Abschnitt werden kurz die Motivation, das Ziel und die Gliederung der Arbeit dargestellt.

1.1 Motivation

Das Spiel Pinball (deutsch: Flipper), wie wir es heute kennen, entstand 1947, als dem Vorgängerspiel die charakteristischen Hebel (Flipper) hinzugefügt wurden. Das Vorgängerspiel basierte darauf, eine Kugel mithilfe einer Spannfeder durch ein Labyrinth zu schießen und eine möglichst hohe Punktzahl zu erreichen. Bagatelle-Automaten, wie sie genannt wurden, wurden im 18. Jahrhundert in Frankreich erfunden. Abbildung 1.1 zeigt einen solchen Automaten aus dem Jahr 1935. [46]

Abbildung 1.1: Bagatelle-Automat: Record Golf (Jahr 1935) [62]

Aufgrund der geringen Beeinflussbarkeit der Bagatelle-Automaten wurden diese als Glücksspiel angesehen und 1942 in New York City und anderen amerikanischen Großstädten verboten. Der neue Automat mit den Hebeln erbte das Verbot seines Vorgängers und wurde ebenfalls untersagt. [55]
Erst 1976 erstritt Roger Sharpe, ein passionierter Flipper-Spieler, vor Gericht die Aufhebung des Verbots für den Flipper. Er erreichte dies, indem er dem Richter und einer Kommission von Rechtsexperten den Verlauf der Kugel vorhersagte. Damit unterstrich er, dass es bei dem Spiel um Geschicklichkeit und Präzision geht. [55]
Während des Verbots wurden 1958 die ersten Flipper nach Deutschland importiert. Der Preis eines solchen Automaten entsprach damals dem eines neuen Volkswagen, ca. 4000 DM. Die Anzahl nahm trotzdem immer weiter zu, bis Ende der 70er Jahre ca. 200.000 Automaten in öffentlichen Einrichtungen aufgestellt waren. Allerdings markierte dies auch den Höhepunkt des Flippers, da durch die neuartigen Mikroprozessoren eine neue Art von Spielen aufkam: die Arcadespiele. Heutzutage sind die Flipper-Automaten fast vollständig aus dem öffentlichen Raum verschwunden. Im Jahr 2007 gab es bei der Spielhallen-Kette Merkur nur noch 34 Flipper in 200 Filialen. Dennoch gibt es Menschen, die sich sehr für diese Automaten interessieren und sie auch sammeln. Es gibt sogar eine Rangliste der weltweit besten Spieler, die „World Pinball Player Rankings“, und jährliche Meisterschaften. [46]
Aufgrund dessen, dass Roger Sharpe eine Logik nachweisen und Vorhersagen über den zukünftigen Verlauf der Kugel treffen konnte, ist dieses Spiel ideal für Reinforcement Learning geeignet. Weiterer Vorteil ist, dass der Flipper ein geschlossenes System ist. Dies bietet den Vorteil, dass man das System nach kurzer Anpassung, autonom und ohne Eingriffe selbständig lernen lassen kann. Im Rahmen dieser Arbeit wird ein durch Reinforcement Learning gesteuerter Flipper-Automat entwickelt. Dafür wird ein Prototyp und eine virtuelle Umgebung entwickelt. Die virtuelle Umgebung soll die Möglichkeit bieten vorab und ohne großen Aufwand ein solches System anzulernen, da dort flexibler Änderungen vorgenommen werden können. Des Weiteren sollen die ermittelten Parameter des Reinforcement Learning aus der virtuellen Umgebung auf den Prototypen übertragbar sein. Diese Arbeit soll zeigen, dass Reinforcement Learning auch bei hochdynamischen Systemen Anwendungen finden kann und dient somit als Basis für weitere wissenschaftliche Arbeiten.

1.2 Ziel

Das Ziel wird in Form der SMART-Methode ausgedrückt, um solide und belastbar zu sein.

Spezifisch:
Die Entwicklung eines Reinforcement-Learning-Systems, dass in der Lage ist, Ziele auf dem Spielfeld zu treffen. Dies beinhaltet das Training und Testen des Systems in einer virtuellen Umgebung, sowie die Entwicklung eines prototypischen Aufbaus. Bestandteile sind die Auslegung der Aktoren, die Implementierung der Steuerung und die Anpassung des Reinforcement-Learning-Systems auf die Hardware.

Messbar:
Die Umsetzung wird anhand der Fähigkeit des Systems bewertet, das Ziele mit der Kugel zu treffen. Diese Bewertung trifft sowohl auf den physischen Aufbau, als auch auf die virtuelle Umgebung zu. Somit hat man den Vorteil, die Umsetzbarkeit in der virtuellen Testumgebung zu testen, bevor man Ressourcen in den physischen Aufbau steckt.

Attraktiv:
Die Entwicklung eines solchen Systems führt zum besseren Verständnis des maschinellen Lernens und bietet des Weiteren eine solide Grundlage für weitere Forschungsarbeiten in diesem Bereich.

Realistisch:
Durch das reiche Informationsangebot im technischen Feld des Reinforcement-Learning sowie dem großen Open-Source Angebot im Bereich der Mikrokontroller kann die Erreichbarkeit gewährleistet werden.

Terminiert:
Der Zeitrahmen, in dem dieses Ziel erreicht werden soll, entspricht dem einer Bachelorarbeit.

1.3 Gliederung

Diese Arbeit ist in acht Teile gegliedert. Kapitel 2 befasst sich mit den Grundlagen. In Kapitel 3 werden die Anforderungen spezifiziert, welche dann in Kapitel 4 zu einem Konzept formuliert werden. In Kapitel 5 und Kapitel 6 wird dann anhand des Konzepts eine Umsetzung erarbeitet. Kapitel 7 befasst sich anschließend mit der Evaluation des entworfenen Systems, hierbei werden die in Kapitel 3 spezifizierten Anforderungen in Augenschein genommen. Abschließend befasst sich Kapitel 8 mit dem Fazit und einem Ausblick auf zukünftige Arbeiten.

2 Grundlagen

Dieses Kapitel befasst sich mit den fundamentalen Grundlagen, die für die Arbeit benötigt werden. Zu Anfang werden Grundlagen zu Flipper-Automaten beschrieben, danach die Grundlagen zu Schrittmotoren, dann werden physikalische Formeln von Kollisionen beschrieben und zum Schluss wird auf das Reinforcement Learning eingegangen.

2.1 Flipper-Automaten

In diesem Abschnitt wird der Aufbau von Flipper-Automaten beschrieben.

2.1.1 Aufbau

Der zentrale Dreh- und Angelpunkt eines Flipperautomats ist das rechteckige Spielfeld, auf dem das gesamte Spielgeschehen stattfindet. Auf dem Spielfeld sind Objekte platziert, die als Hindernisse dienen. Diese Objekte haben die Form von Rampen, Bumpern und Löchern. Die Flipper, welche auch die Namensgeber für den deutschen Namen sind, sind bewegliche Hebel, die durch den Spieler gesteuert werden können. Die Flipper sind trichterförmig mit einem Winkel vom typischerweise 35° zur Waagerechten angeordnet und befinden sich am unteren Ende des Spielfelds. Die Flipper dienen dazu, die Kugel im Spielfeld hin und her zu schießen. Die 35° entstammen der grafischen Berechnung aus Abbildung 2.1. Die Kugel kann durch eine Federstange in das Spielfeld gebracht werden. In Abbildung 2.1 ist ein exemplarisches Spielfeld gezeigt.
Die Maße und der Aufbau von Flippern unterscheiden sich sehr stark von Modell zu Modell und von Hersteller zu Hersteller, da Flipper themenbezogen sind und die Gestaltung der Spielfelds vom jeweiligen Thema abhängig ist. Zudem ist es schwierig Literatur zu finden, die Maße und den Aufbau beschreibt. Lediglich in Online-Foren findet man Information über diese Themen. Nachfolgend sind gängige Größen und Maße aufgelistet.

- Die Maße des Spielfelds betragen 51,5 cm x 107 cm. [13]

- Die Länge eines Flipperfingers beträgt 7,6 cm (3 Zoll). [14]

- Im Ruhezustand sind die Flipper trichterförmig mit einem Winkel vom 35° zur Waagerechten angeordnet (grafische Ermittlung aus Abbildung 2.1).

- Die Flipper bewegen sich in einem Bereich von 70°. Dieser resultiert aus dem zweifachen Winkel der Ruheposition.

- Die Banden links und rechts von den Flippern weisen den gleichen Winkel wie die Flipper auf (siehe Abbildung 2.1).

- Der Ball wird mithilfe eines Plungers (obengenannte Federstange) in das Spielfeld geschossen. [36]

2.1.2 Steuerung und Mechanik

Die Steuerung von älteren Flipperautomaten wurde elektromechanisch realisiert, mit Relais, Spulen, Motoren und Schaltern. Die Flipper wurden mithilfe von Elektromagneten bewegt, ein Beispiel dafür ist in Abbildung 2.2 zu sehen. Die neueren Modelle werden mithilfe von ICs, Transistoren und Prozessoren gesteuert, somit konnten neue Spielabläufe und Modi mit neuen Aufgaben umgesetzt werden. Diese Modelle bieten eine Vielzahl von Interaktionsmöglichkeiten in und um das Spielfeld herum an. Die Bewegung der Flipper durch Elektromagneten blieb jedoch die gleiche.

2.2 Schrittmotoren

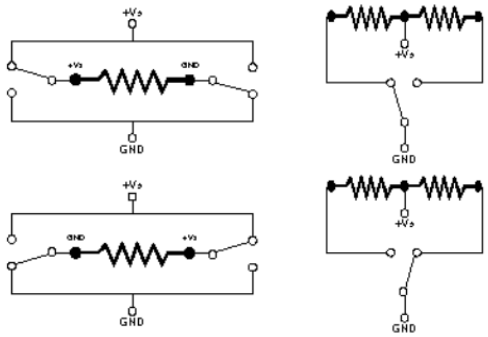
In diesem Abschnitt werden kurz die Grundlagen zu Schrittmotoren erklärt, um ein besseres Verständnis über die Funktionsweise und die Stellparameter zu bekommen.

Abbildung 2.1: Spielfeld [33]

2.2.1 Funktionsweise

Ein Schrittmotor ist ein synchroner Elektromotor, welcher mit Gleichstrom betrieben wird. Er besitzt Statorspulen, die durch die anliegende Spannung ein Magnetfeld ausbilden. Der Rotor richtet sich dann nach diesem aus. Durch gezieltes Ansteuern der Wicklungen, entsteht eine Drehung. Der Rotor ist dabei bürstenlos und besteht je nach Schrittmotor-Typ aus Eisen, bei welchem das Magnetfeld nach Ausschalten verschwindet, oder aus Permanentmagneten. [41]
Es gibt zwei verschiedene Ansteuerungstechniken der Spulen, die bipolare und die unipolare. Beim bipolaren Betrieb wird die Stromrichtung durch die Spulen umgekehrt. Bei unipolaren Betrieb gibt es einen Mittelabgriff, an dem die Versorgungsspannung anliegt. Abwechselnd werden dann die Spulenenden mit Masse verbunden und so der Stromkreis geschlossen. In Abbildung 2.3a ist eine bipolare Beschaltung und in Abbildung 2.3b ist eine unipolare Beschaltung zu sehen. [41]
Ein wichtiger Parameter bei Schrittmotoren ist die Auflösung einer Umdrehung. Eine gängige Größe ist dabei 1,8° pro Schritt. Um eine volle Umdrehung durchzuführen,

Abbildung 2.2: Elektromagnetischer Flipper [12]



(a) Bipolar (b) Unipolar

Abbildung 2.3: Schrittmotorbeschaltung [56]

müssen also 200 Schritte absolviert werden. Durch ausgeklügelte Steuerungsarten lassen sich jedoch auch Positionen zwischen den eigentlichen Schritten ansteuern. [41]

2.2.2 Stellparameter

Die Motornennspannung, die bei Schrittmotoren angegeben wird, ist im Vergleich zur Versorgungsspannung meistens sehr viel niedriger. Die Motornennspannung bezieht sich auf die konstante Spannung, die angelegt werden kann, ohne dass der Motor thermische Schäden erleidet. Dieser Betrieb wird auch als „Konstantspannungsbetrieb “ bezeichnet. Heutzutage sind Kostantstromsteuerungen bei Schrittmotoren sehr verbreitet, da das Drehmoment stark von der Stromstärke abhängig ist. Bei diesen Steuerungen sorgt ein Stromregler dafür, dass der Motor unabhängig von der Versorgungsspannung maximal

mit dem Motornennstrom betrieben wird. Aus diesem Grund ist die Versorgungsspannung der Steuerungen um ein vielfaches höher als die Motornennspannung des Schrittmotors. [60, 61]
Ein weiterer Aspekt, der berücksichtigt werden sollte, ist die steigende Impedanz bei hohen Drehzahlen. Das maximale Drehmoment des Schrittmotors liegt bei geringen Drezahlen vor, je höher die Drehzahl wird, desto geringer ist das Drehmoment. Wie bereits beschrieben, ist das Drehmoment stark von der Stromstärke abhängig. Die Stromstärke wird durch die Impedanz beeinflusst. Die Impedanz X_L steigt mit steigender Umschaltfrequenz und verringert so die Stromstärke und das Drehmoment (siehe Formel 2.1). Dabei ergibt sich die Frequenz f aus der Drehzahl und die Induktivität L ist eine spezifische Größe der verbauten Spulen. Ab einer bestimmten Drehzahl wird das Drehmoment zu gering, um den Rotor einen Schritt weiter zu drehen. Ist dies der Fall, spricht man von einem Schrittverlust. Bei hohen Drehzahlen sollte deshalb eine Verifizierung der zurückgelegten Drehung mittels Encoder, Endlagen oder anderen Methoden vorgenommen werden. [60, 61, 41]

$$X_L = 2\pi f \cdot L \tag{2.1}$$

2.3 Kollisionen und Stöße

Es gibt zwei Arten von Kollisionen, die elastische und die unelastische Kollision. Diese beiden Arten unterscheiden sich im Wesentlichen darin, wie sie kinetische Energie erhalten.
Bei elastischen Kollisionen bleiben sowohl Impuls als auch kinetische Energie erhalten. Die Summe der kinetischen Energie von allen beteiligten Objekten vor der Kollision ist gleich der kinetischen Energie nach der Kollision. Zusätzlich kommt es zu keiner dauerhaften Verformung, wie man auch aus dem Namen schlussfolgern kann. [52].
Bei einer unelastische Kollision bleibt nur der Impuls erhalten. Die kinetische Energie wird in andere Energieformen umgewandelt, wie zum Beispiel Wärme oder auch Schall. Bei dieser Art von Kollision bleibt eine dauerhafte Verformung bestehen. Wenn die beiden kollidierenden Körper sich nach dem Zusammenstoß zu einem Körper vereinen, spricht man von einem vollkommen unelastischen Kollision. [52] Tabelle 2.1 fasst diese Informationen kurz zusammen.

Tabelle 2.1: Übersicht elastische und unelastische Kollision

	Elastische Kollision	**Unelastische Kollision**
Impuls	Impuls bleibt erhalten	Impuls bleibt erhalten
Kinetischer Energie	Kinetische Energie ändert seine Energieform nicht	Kinetische Energie wird in andere Energieformen umgewandelt
Verformung	Keine dauerhafte Verformung	Dauerhafte Verformung
Beispiel	Billardkugeln	Autounfall

2.3.1 Eindimensionale Kollisionen

Als Erstes werden Kollisionen im eindimensionalen Raum betrachtet, um einen grundlegenden Kenntnisstand zu schaffen.

Elastische Kollision

Kollidieren zwei Körper mit einer Anfangsgeschwindigkeit elastisch, resultieren daraus zwei neue Geschwindigkeiten. Da sowohl der Impulserhaltungssatz als auch der Energieerhaltungssatz gilt, entsteht ein System mit zwei Gleichungen und zwei Unbekannten, welches eindeutig lösbar ist. Nachfolgend werden die Formeln für die resultierenden Geschwindigkeiten $v_1^{'}$ und $v_2^{'}$ gezeigt. Dabei ist m_1 und m_2 die Masse und v_1 und v_2 die Geschwindigkeiten des jeweiligen Körpers. [52]

$$v_1^{'} = \frac{(m_1 - m_2)v_1 + 2m_2v_2}{m_1 + m_2} \tag{2.2}$$

$$v_2^{'} = \frac{2m_1v_1 + (m_2 - m_1)v_2}{m_1 + m_2} \tag{2.3}$$

2.3.2 Spezialfall der elastischen Kollision

Ein Spezialfall der elastischen Kollision ist der, bei der eine der Massen sehr viel größer als die andere ist ($m_2 >> m_1$). Ein Beispiel dafür ist die Kollision mit einer Wand. Für die resultierende Geschwindigkeit erhält man dann folgende Formel [52]:

$$v_1^{'} = 0 - v_1 \tag{2.4}$$

Der zweite Körper befindet sich nach der Kollision immer noch in Ruhe. Der Impuls p_1 des ersten Körpers dreht sich vollständig um, er „prallt ab“. Die mathematische Beschreibung dieses Sachverhalts ist in Formel 2.7 zu sehen. Dabei ist $p_1^{'}$ der resultierende Impuls. [52]

$$p_1^{'} = p_2 - p_1 \tag{2.5}$$
$$= (-m_1 v_1) - (m_1 v_1) \tag{2.6}$$
$$= -2 \cdot m_1 \cdot v_1 \tag{2.7}$$

Der Impuls ändert sich also nicht nur, sondern er verdoppelt sich auch.

2.3.3 Vollkommen unelastische Kollision

Bei einer unelastischen Kollision tritt eine weitere Unbekannte auf, nämlich ΔE welche die Energie beschreibt, die in andere Formen umgewandelt wird. Dieser Wert ist abhängig von den Körpern. Der Einfachheit halber wird hier also nur die vollkommen unelastische Kollision betrachtet. Bei dieser wird angenommen, dass die Körper nach der Kollision zu einem „verschmelzen “, somit können die resultierenden Geschwindigkeiten gleichgesetzt werden und eine Unbekannte entfällt. Nachfolgend wird die Formel für die resultierende Geschwindigkeit v gezeigt. [52]

$$v = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2} \tag{2.8}$$

2.3.4 Zweidimensionale Kollisionen

Nachfolgend wird kurz die zweidimensionale elastische Kollision betrachtet.

Elastische Kollision

Das Prinzip der zweidimensionalen elastischen Kollision beruht auf dem der eindimensionalen elastischen Kollision. Dafür muss zunächst der Abstandsvektor zwischen den beiden Körpern zum Zeitpunkt der Kollision ermittelt werden (siehe Abbildung 2.4). Der

Geschwindigkeitsvektor wird dann in zwei Komponenten unterteilt, parallel und orthogonal zum Abstandsvektor. Die Komponente orthogonal zum Abstandsvektor verändert sich nicht. Die Komponente parallel zum Abstandsvektor kann man nach dem Prinzip der eindimensionalen elastischen Kollision lösen. [40]

Spezialfall der elastischen Kollision

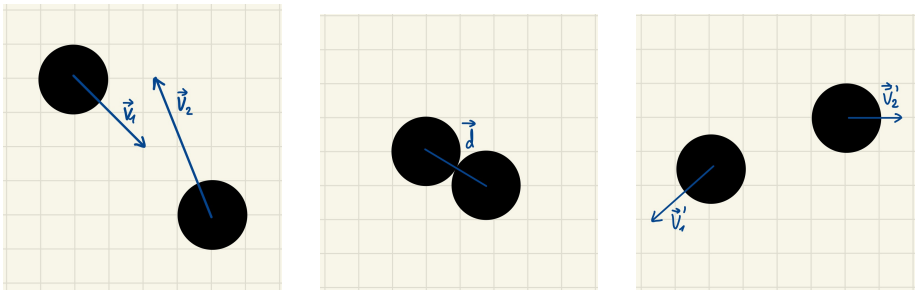
Für den Spezialfall, dass die beiden Körper rund sind und die gleiche Masse haben, ergeben sich für die Ausgangsgeschwindigkeitsvektoren folgende Gleichungen [40]:

$$v_1' = v_1 - \frac{\mathbf{d} \cdot (\mathbf{v_1} - \mathbf{v_2})}{|\mathbf{d}|^2} \cdot \mathbf{d} \tag{2.9}$$

$$v_2' = v_2 + \frac{\mathbf{d} \cdot (\mathbf{v_1} - \mathbf{v_2})}{|\mathbf{d}|^2} \cdot \mathbf{d} \tag{2.10}$$

2.4 Reinforcement Learning

Das Konzept des Reinforcement Learning hat seine Wurzeln in der Psychologie, wo es darauf abzielt, dass eine Handlung durch positive Konsequenzen verstärkt wird, was wiederum die Wahrscheinlichkeit erhöht, dass diese Handlung in ähnlichen Situationen erneut ausgeführt wird. Ähnlich wie beim Training eines Hundes nutzt man positive Bestärkung, um gewünschtes Verhalten zu fördern. Dadurch steigt die Wahrscheinlichkeit, dass der Hund diese Handlungen in ähnlichen Situationen wiederholt. Diese Prinzipien finden auch in der Entwicklung von Reinforcement-Learning-Algorithmen Anwen-



(a) Vor Kollision (b) Während Kollision (c) Nach Kollision

Abbildung 2.4: Zweidimensionale Ballkollision

dung, wo Belohnungen dazu dienen, dem System zu lehren, welche Aktionen erwünscht sind und welche vermieden werden sollen. Das Reinforcement Learning hat sich als Schnittstelle zwischen verschiedenen Forschungsgebieten etabliert, Abbildung 2.5 stellt diese Forschungsgebiete dar. Reinforcement Learning ist neben Supervised Learning und Unsupervised Learning ein wesentlicher Bestandteil des maschinellen Lernens, welches wiederum eine Teilmenge der Künstlichen Intelligenz (KI) darstellt. [54]
Im *Supervised Learning* wird dem System eine Eingabe zusammen mit einer spezifischen Kategorisierung (in Englisch: Label) präsentiert. Hierbei erwartet man, dass das System eine Beziehung oder Muster zwischen der Eingabe und der zugehörigen Kategorisierung erkennt. Diese Methode wird häufig verwendet, um Modelle zu trainieren, die in der Lage sind, aus neuen Daten korrekte Klassifizierungen vorzunehmen. [54]
Im Gegensatz dazu erhält das System bei *Unsupervised Learning* lediglich Eingabedaten ohne spezifische Kategorisierung. Ziel ist es, innerhalb dieser Daten Muster, Zusammenhänge oder Strukturen zu erkennen, um ähnliche Datenelemente zu gruppieren oder Charakteristiken aus den Daten zu extrahieren. Unsupervised Learning wird oft verwendet, um Muster in großen Datensätzen zu entdecken, die auf den ersten Blick nicht offensichtlich sind. [54] In Abbildung 2.6 sind die Bereiche des maschinellen Lernens abgebildet. Im Folgenden sind die zentralen Bereiche aufgelistet, die ein Reinforcement-Learning-System enthält. Anschließend werden diese näher erläutert.

- Agent
- Rewards (in Deutsch: Belohnung)
- Environment (in Deutsch: Umgebung)
- State (in Deutsch: Zustand)
- Policy (in Deutsch: Richtlinie)
- Value function (in Deutsch: Werte-Funktion)

Agent: Der Agent ist der aktive Teil des Systems, der über Intelligenz verfügt und eigenständig Entscheidungen darüber trifft, welche Handlungen in der Umgebung durchgeführt werden sollen. Er ist das Kernelement, das in der Lage ist, auf Basis von Beobachtungen, Informationen und den gelernten Belohnungen, Strategien zu entwickeln und Aktionen zu wählen, die auf bestimmte Ziele oder Anforderungen abzielen. [54]

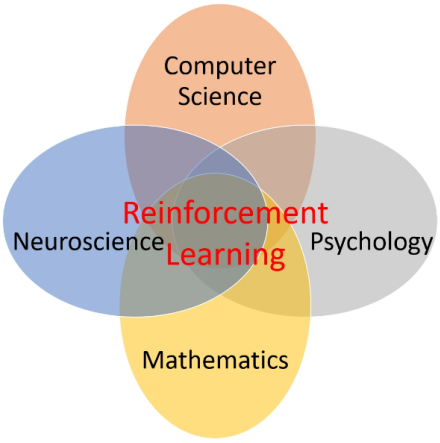


Abbildung 2.5: Forschungsbereiche des Reinforcement Learning [54]

Rewards: Die Belohnung, die oft als R_t bezeichnet wird, ist typischerweise eine skalare Größe, die dem Agenten als Feedback bereitgestellt wird, um sein Lernen zu steuern. Sie repräsentiert das Signal, das dem Agenten sagt, ob seine Aktion in einer bestimmten Situation positiv oder negativ war.
Das primäre Ziel des Agenten besteht darin, die Summe der erhaltenen Belohnungen über die Zeit zu maximieren. Dieses Signal fungiert als Anreizmechanismus und dient als Maß dafür, wie gut der Agent im Verlauf seiner Handlungen vorankommt. [54]

Environment: Die Umgebung ist das Umfeld, in dem der Agent Aktionen ausführt. Die Abbildung 2.7 zeigt den Zusammenhang zwischen Agent und Umgebung.
In jedem Zeitschritt t erhält der Agent eine Beobachtung (Englisch: Observation) O_t aus der Umgebung, die dann eine Aktion A_t auslöst. Aus der Aktion resultiert eine Belohnung R_t, die der Agent mit der nächsten Beobachtung erhält O_{t+1}. Dieser Prozess wird so lange wiederholt, bis ein Endzustand erreicht wird. [54]

State: Der Prozess des Wiederholens sorgt dafür, dass sich eine Abfolge von Beobachtungen O_i, Aktionen A_i und Belohnungen R_i entwickelt, die sich als Historie wie folgt definieren lässt:

$$H_t = \{O_1, A_1, R_1\}, \{O_2, A_2, R_2\}, ..., \{O_t, A_t, R_t\} \tag{2.11}$$

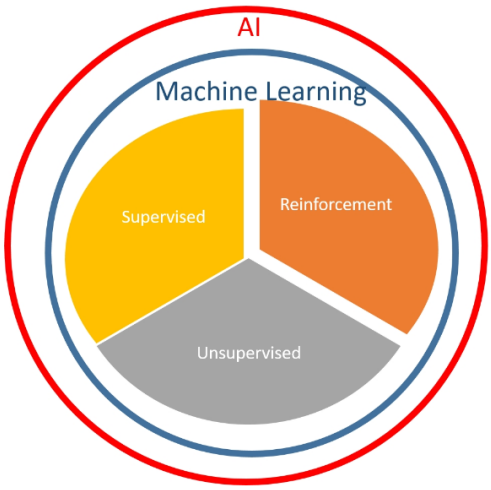


Abbildung 2.6: Bereiche des maschinellen Lernens [54]

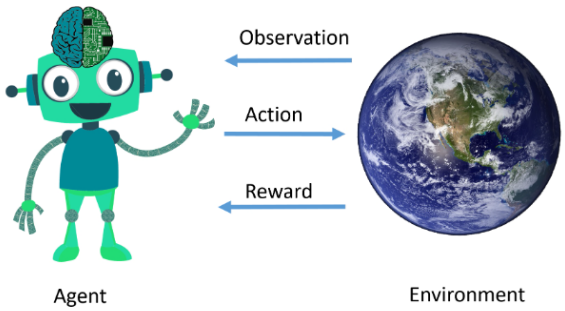


Abbildung 2.7: Zusammenhang zwischen Agent und Umgebung [54]

Der Zustand S (State) setzt sich dann aus einer Funktion zusammen, deren Bestandteil die Historie ist.

$$S_t = f(H_t) \tag{2.12}$$

Es gibt einen State für die Umgebung und einen für den Agenten. Normalerweise ist der State der Umgebung nicht sichtbar für den Agenten („teilweise beobachtbare Umgebungen“), es gibt jedoch Umgebungen, die ihren State veröffentlichen. Diese werden dann als „vollständig beobachtbare Umgebungen“ bezeichnet. [54]

Policy: Die Richtlinie wird als π dargestellt und beschreibt, welche Aktion bei einem gegebenen Zustand ausgeführt werden soll. Man kann also sagen, dass der Agent der Richtlinie folgt. Es gibt zwei verschiedene Arten von Richtlinien, die *deterministische* Richtlinie und die *stochastische* Richtlinie. Bei der *deterministischen* Richtlinie gibt es nur eine Aktion, die ausgeführt werden kann. Bei der *stochastischen* Richtlinie gibt es mehrere Aktionen, die ausgeführt werden können. Hierbei gibt die Richtlinie einen Wahrscheinlichkeitswert für jede Aktion zurück. Mathematisch bedeutet dies:

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2.13}$$

P ist dabei das Zustandsübergangsmodell.

Value function: Die Value function beinhaltet die Vorhersage über die Belohnung des Agenten. Man unterscheidet zwischen *State-value function* und *Action-value function*.
Die *State-value function* wird üblicherweise als $v(s)$ dargestellt. Sie stellt die Vorhersage des Agenten über die zukünftige Belohnung dar, basierend auf dem Zustand s und unter der Policy π. Mathematisch kann man diesen Zusammenhang wie folgt darstellen. [43]

$$v_\pi(s) = E\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\Big] \tag{2.14}$$

Dies bedeutet, dass der Wert des Zustands s unter der Policy π die erwartete Summe der gewichteten zukünftigen Belohnung ist. Dabei ist γ der Gewichtungsfaktor und eine Zahl zwischen [0,1]. Typischerweise liegt dieser zwischen 0.95 und 0.99.
Die *Action-value function* für eine Policy π wird üblicherweise als $q_\pi(s_t, a_t)$ dargestellt. Sie beschreibt, wie sinnvoll es ist, eine Aktion auszuführen, wenn man in einem gegebenen Zustand ist. Mathematisch kann man diesen Zusammenhang wie folgt darstellen [43]:

$$q_\pi(s, a) = E\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\Big] \tag{2.15}$$

2.4.1 Reinforcement learning Algorithmen

Das Ziel des Algorithmus ist es die Belohnung zu maximieren. Dies kann durch zwei Ansätze erzielt werden.
Der erste Ansatz ist, die Entscheidung des Algorithmus auf Grundlage der Belohnung

entweder positiv oder negativ zu bewerten. Dieser Fall lässt eine kontinuierliche Entwicklung zu, der Algorithmus verbessert sich mit jeder Bewertung selbst. [48]
Der zweite Ansatz bewertet, genau wie der erste Ansatz, die Aktionen durch die erhaltenen Belohnungen, jedoch in einer Trainings-Phase. Wenn die Aktionen des Systems zuverlässig sind, findet keine Bewertung der Aktionen mehr statt. [48]
Um die nachfolgenden Beschreibungen zu verstehen, werden hier kurz die Begriffe *on-policy* und *off-policy* erklärt.
On-policy: Die Aktualisierung erfolgt auf Basis von Aktionen, die von der Richtlinie bestimmt werden. Kurz gesagt, verwendet der Algorithmus seine aktuelle Richtlinie, um Aktionen auszuführen. Aus den Erfahrungen dieser Aktionen wird die Richtlinie aktualisiert. [58]
Off-policy: Die Aktualisierung erfolgt auf Basis von Aktionen, die nicht unbedingt von der aktuellen Richtlinie bestimmt werden. Kurz gesagt, führt der Algorithmus zufällige Aktionen aus, um die Umgebung zu erkunden und somit seine Richtlinie zu aktualisieren.
Im Folgenden werden die am häufigsten verwendeten TDL-Algorithmen (eng. Temporal Difference Learning) aufgelistet und kurz beschrieben. Temporales Differenzlernen bedeutet hierbei, dass diese Art von Algorithmen, basierend auf der Differenz zwischen erwarteten und tatsächlichen Belohnungen, Aktualisierungen vornimmt. [58]

Q-Learning

Der Q-Learning Algorithmus ist ein off-policy Algorithmus. Zentraler und wichtigster Bestandteil ist dabei die Q-Tabelle, welche dem Algorithmus hilft die richtige Aktion auszuwählen. Die Anzahl der Spalten richtet sich dabei nach den möglichen Zuständen, die Zeilen sind die möglichen Aktionen. Zu Anfang ist die Tabelle leer, beziehungsweise mit Nullen gefüllt. Um die Tabelle mit sinnvollen Werten zu füllen, werden zufällige Aktionen ausgeführt, wodurch die Zustände „erkundet“ werden, diese Herangehensweise wird auch „Exploration-Phase“ genannt. In der darauffolgenden „Exploitation-Phase“ wird das zuvor erlangte Wissen ausgenutzt, um Aktionen mit der höchsten Belohnung vorauszusagen. Über den sogenannten Epsilon-Wert kann definiert werden, wie oft sich der Algorithmus in den Phasen befindet. Die Werte in der Q-Tabelle, werden dabei mit jeder ausgewählten Aktion angepasst, dies passiert solange, bis ein spezifizierter Endpunkt

erreicht wird. Der Q-Wert für ein Tabellenfeld lässt sich dabei wie folgt berechnen [42]:

$$Q(S_t, A_t) = (1 - \alpha) \cdot Q(S_t, A_t) + \alpha(R_t + \lambda \cdot Q(S_{t+1}, \alpha)) \tag{2.16}$$

- S_t :Zustand zum Zeitpunkt t
- S_{t+1} :Zustand zum Zeitpunkt t+1
- A : Aktion
- R : Reward
- α : Lernrate
- λ : Discount Factor: Faktor, der dafür sorgt, dass neue Ereignisse einen weniger starken Einfluss haben als vergangene Ereignisse

SARSA

Der State-Action-Reward-State-Action (SARSA-)Algorithmus ist ein on-policy Algorithmus. Der Algorithmus benötigt für die Aktualisierung der Action-value function Zustands-Aktions-Paare. Insgesamt werden folgende Informationen benötigt: Zustand (State), Aktion (Action), Belohnung (Reward), nächster Zustand (State) sowie die nächste Aktion (Action). Daraus bildet sich auch der Name SARSA. Der Algorithmus basiert auf dem des Q-Learning und besitzt genau wie dieser eine Aktionswerttabelle. Der Unterschied ist, dass der SARSA-Algorithmus eine on-policy verfolgt. [48, 44]

Deep Q-Learning

Deep Q-Learning ist eine Weiterentwicklung von Q-Learning, dabei wird ein neuronales Netz verwendet, um Q-Werte zu approximieren. Das Netzwerk bekommt als Eingabe den Zustand und gibt als Ausgabe die Q-Werte für alle möglichen Aktionen aus. Die Verwendung eines neuronalen Netzes ermöglicht es ähnliche Zustände zu generalisieren, wodurch neue Situationen besser bewältigt werden können. Dies ist auch hilfreich bei Umgebungen, die eine Vielzahl von Zuständen besitzen. [48]

2.4.2 OpenAI Gym

OpenAI Gym ist eine Open-Source-Toolbox, die eine umfangreiche Auswahl an virtuellen Testumgebungen für intelligente Agenten-Algorithmen bereitstellt. Diese Toolbox bietet eine standardisierte Programmierschnittstelle (API), um mit den verschiedenen Umgebungen zu interagieren. Der Lernprozess in OpenAI Gym erfolgt über ein episodisches Verfahren. Die Erfahrungen des Agenten werden auf Episoden zurückgeführt. Jede Episode beginnt damit, dass die Umgebung in einen anfänglichen Zustand versetzt wird, der entsprechend den definierten Anforderungen zufällig ist. Eine Episode endet, sobald die Umgebung einen festgelegten Endzustand erreicht hat. Neben den bereits vordefinierten Testumgebungen bietet OpenAI Gym auch die Möglichkeit, maßgeschneiderte Umgebungen zu erstellen, die spezifische Anforderungen erfüllen. Dadurch können Entwickler Umgebungen erschaffen, die genau auf die Bedürfnisse ihrer Algorithmen und Forschungsziele zugeschnitten sind. In Abbildung 2.8, kann man verschiedene Umgebungen sehen, die OpenAI Gym zur Verfügung stellt. [54]

Umgebungen

Umgebungen werden je nach Eigenschaften in Kategorien eingeteilt. Nachfolgend werden diese mit einer kurzen Beschreibung aufgeführt.

Classic Control: Es gibt fünf klassische Steuerungsumgebungen: Acrobot, CartPole, Mountain Car, Continuous Mountain Car und Pendulum. Diese sind stark konfigurierbar und verfügen über die Möglichkeit Störgrößen auf die Aktionen anzuwenden. Diese Umgebungen sind leicht durch intelligente Agenten zu lösen. [9]

Box2D: Diese Kategorie umfasst Spielzeugspiele, die auf der Box2D-Physik-Engine basieren und PyGame für das Rendering verwenden. Auch diese Umgebungen zeichnen sich durch ihre Anpassungsfähigkeit aus und ermöglichen es, sie nach den Bedürfnissen der Anwender zu konfigurieren. Sie bieten eine breite Palette an Konfigurationsmöglichkeiten an, die es ermöglichen, die Spielparameter, die Physik oder andere Aspekte der Umgebung entsprechend anzupassen. [8]

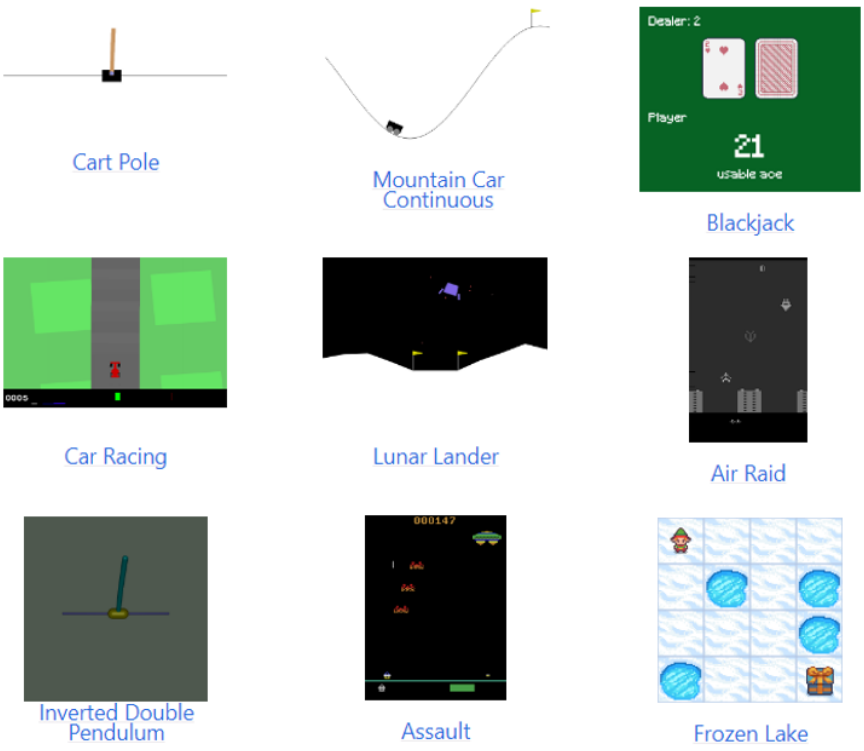


Abbildung 2.8: Verschiedene Testumgebungen von OpenAI Gym [3]

Toy Text: Diese Umgebungen zeichnen sich durch ihre Einfachheit aus und verfügen über vergleichsweise kleine Zustands- und Aktionsräume. Diese Eigenschaften machen sie äußerst geeignet für die Anwendung von Reinforcement-Learning-Algorithmen. Die klare und übersichtliche Struktur der Umgebungen ermöglicht es den Algorithmen, schnell Muster zu erkennen und effektive Handlungsstrategien zu erlernen. [38]

MuJoCo: MuJoCo, eine Abkürzung für Multi-Joint Dynamics with Contact, repräsentiert eine leistungsstarke Physik-Engine, die Forschung und Entwicklung in diversen Bereichen wie Robotik, Biomechanik, Grafik, Animation und anderen Disziplinen, die eine schnelle und präzise Simulation erfordern, unterstützt. Innerhalb der Umgebungen sind die MuJoCo-basierten Szenarien oft als anspruchsvolle einzustufen. Diese Realitätsnähe macht sie zu einem herausfordernden Testgebiet für fortgeschrittene Algorithmen des Reinforcement Learnings und der künstlichen Intelligenz. Sie erfordern oft komplexere

Strategien und eröffnen Forschern und Entwicklern die Möglichkeit, robuste und vielseitige Lösungen für komplexe Probleme zu erarbeiten. [27]

Atari Die Atari-Kategorie bietet eine Vielzahl an Arkade-Spielen, die man abbilden kann. Diese Umgebungsschnittstellen sind Wrapper, die auf der Arcade Learning Environment (ALE) aufsetzen. Als Input des Agenten werden Bildschirmbilder oder das RAM des Spiels verwendet. [54]

Benutzerdefinierte Umgebung

Eine benutzerdefinierte Umgebung erbt von *gym.Env* und ist somit eine abgeleitete Klasse, die auf der Basisklasse aufbaut. Es ist äußerst wichtig einige Methoden und Attribute anzulegen, da sonst Komplikationen auftreten können. Abbildung 2.9 dient dabei zur Veranschaulichung, zudem werden nachfolgend die Methoden und Attribute erklärt. [26]
In dem *metadata* Attribut werden verschiedene Render-Modi angegeben, wobei es ratsam ist, immer auch *None* als unterstützten Modus anzugeben, um potenzielle Fehler zu vermeiden. Dies muss aber nicht unbedingt in dem *metadata* Attribut passieren, sondern kann in der *__init__* Methode umgesetzt werden.
In der *__init__* Methode werden grundlegende Variablen initialisiert, die für das Funktionieren der Umgebung entscheidend sind. Neben den benutzerdefinierten Variablen gehören *self.observation_space* und *self.action_space* zu Variablen, die initialisiert werden müssen. Dabei ist es wichtig dies sorgfältig zu tun, da sie die Grenzen und Eigenschaften der beobachtbaren Zustände und der möglichen Aktionen definieren.
Der *self.observation_space* gibt die Grenzen und Merkmale der beobachtbaren Zustände an, beispielsweise die Dimensionen des Spielfeldes und/oder andere relevante Informationen, die der Agent wahrnehmen kann. Auf der anderen Seite definiert *self.action_space* die möglichen Aktionen, die der Agent ausführen kann, wie zum Beispiel die Bewegungen in einem Spiel oder andere Interaktionsmöglichkeiten innerhalb der Umgebung. Diese beiden Variablen bilden die Grundlage für das Verhalten des Agenten mit der Umgebung und sind essentiell für eine erfolgreiche Anwendung von Reinforcement-Learning-Algorithmen in der benutzerdefinierten Umgebung. [26]
Eine weitere essenzielle Methode innerhalb der Umgebung ist die *reset* Methode. Diese Funktion setzt die Umgebung zurück und versetzt sie in ihren Anfangszustand. Es bietet sich hierbei die Möglichkeit, eine gewisse Zufälligkeit zu integrieren, um dem Agenten eine vielfältigere Erfahrung in der Umgebung zu ermöglichen. Die Einführung von Variationen

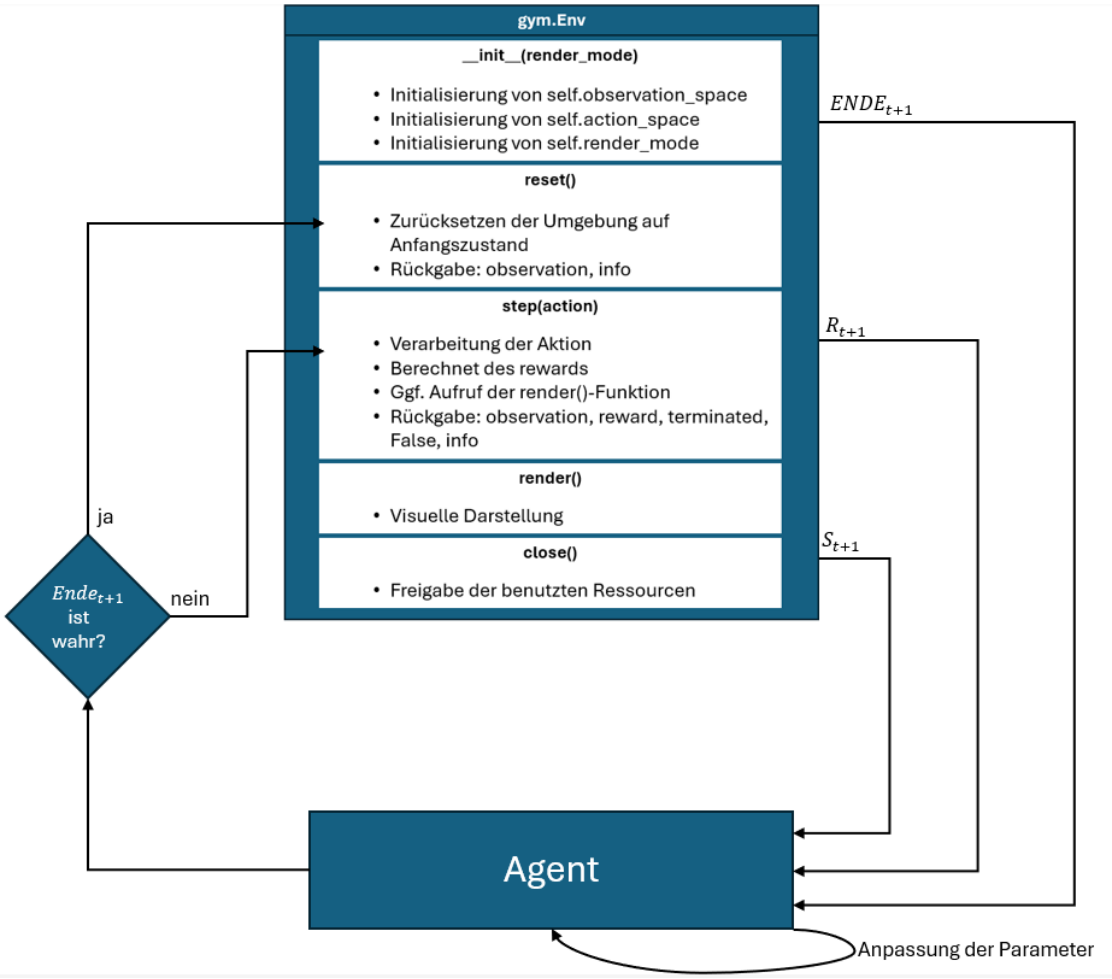


Abbildung 2.9: Eigenschaften der OpenAI-Gym-Umgebung

kann dazu beitragen, dass die künstliche Intelligenz neue Muster und Strategien erlernt. Die *reset* Methode gibt einen Wert zurück, der als Observation bezeichnet wird. Dieser Observations-Wert muss sich innerhalb der definierten Grenzen des *self.observation_space* befinden. Zusätzlich zur Observation gibt es die Option, einen Info-Wert zurückzugeben, der spezifische Informationen enthalten kann. [26]
Die *step*-Methode bildet das Herzstück der Logik innerhalb der Umgebung. Sie erwartet als Eingabe einen Aktions-Wert, der innerhalb der Methode verarbeitet wird. Die Rückgabewerte sind *observation*, *reward*, *terminated*, *False* und *info*. Der *observation* Parameter repräsentiert den aktuellen Zustand der Umgebung nach der Ausführung der Aktion. Diese Information ist entscheidend für den Agenten, um seinen nächsten Schritt

zu planen. Der *reward*-Wert gibt an, wie die ausgeführte Aktion bewertet wird. Diese Rückmeldung hilft dem Agenten, seine Handlungen anhand ihrer Ergebnisse zu bewerten und zu verbessern. Der *terminated*-Wert gibt an, ob ein vordefinierter Endzustand in der Umgebung erreicht wurde. Dieser Parameter signalisiert, ob das Spiel oder die Aufgabe beendet ist. Der vierte Parameter, *False*, ist ein Platzhalter, der für zukünftige Erweiterungen oder spezielle Anwendungsfälle genutzt werden kann. Zusätzlich bietet die *step*-Methode den *info*-Wert an, der detaillierte Informationen liefern kann, die für die Ausführung des Agenten relevant sein können. Es ist empfehlenswert, eine Abfrage über den aktuellen Rendermodus einzubauen, damit das Generieren der grafischen Oberfläche aus der step-Methode heraus angestoßen werden kann. [26]
Um die Umgebung visuell darzustellen, existiert eine *render*-Methode, die häufig auf anderen Python-Bibliotheken wie beispielsweise Pygame aufbaut. Je nach gewähltem Rendermodus variiert die Menge der gerenderten Informationen. Daher ist es ratsam, Abfragen zum aktuellen Rendermodus einzubauen, um das Rendern entsprechend anzupassen. Zusätzlich erfordert die *render*-Methode die Berücksichtigung des FPS-Werts (Frames pro Sekunde), der die Bildwiederholrate festlegt. Die Handhabung dieses Wertes innerhalb der Methode ist wichtig, da er die Geschwindigkeit und Flüssigkeit der visuellen Darstellung beeinflusst. Die Anpassung des Rendermodus und des FPS-Werts in der render-Methode ermöglicht eine Steuerung der visuellen Darstellung je nach den Anforderungen des Agenten, der Umgebung oder der spezifischen Trainingsphase. [26]
Die *close*-Methode ist wichtig, um sicherzustellen, dass nach der Nutzung der Umgebung, alle Ressourcen ordnungsgemäß geschlossen werden und um Speicher und andere Systemressourcen freizugeben. Durch die Nutzung vorgefertigter Funktionen aus etablierten Bibliotheken wird die Implementierung der close-Methode vereinfacht und ermöglicht eine effiziente Freigabe von Ressourcen. [26]

2.4.3 TensorFlow

TensorFlow ist eine Open-Source-Softwarebibliothek, die sich auf neuronale Netze spezialisiert hat. Die Bibliothek wurde 2015 vom Google-Team entwickelt. Sie unterstützt gängige Programmiersprachen wie Python, C++, Java, R und Go. Der Datentransfer läuft über die Keras API (Application Programming Interface- deutsch: Programmierschnittstellen). Diese ist seit dem Update 2.0 die Standartschnittstelle für die Interaktion mit TensorFlow geworden. TensorFlow-Modelle werden als Data Flow Graph (Datenflussdiagramme) dargestellt. Diese Graphen bestehen aus Knoten, welche mathematische

Operationen abbilden, und Kanten, die Daten zwischen den Knoten transportieren. Die Daten werden dabei als Tensoren bezeichnet. Die Tensoren sind dabei multidimensionale Datenstukturen, die Zahlen, Vektoren, Matrixen oder n-dimensionale Arrays darstellen können. [45]

Datenflussdiagramme (Data Flow Graph)

TensorFlow speichert die Modelle als Datenflussdiagramme, dadurch kann man die Modelle auch in Umgebungen verwenden, die keinen Phython-Interpreter besitzen. [20] Zudem sind Graphen leicht zu optimieren, was dem Compiler folgende Vorteile bietet:

- Separierung von unabhängigen Teilbereichen, um diese gegebenenfalls auf Threads oder Geräten aufzuteilen.
- Vereinfachung von arithmetischen Operationen, indem man gemeinsame Teilausdrücke eliminiert.

Es existiert ein eigenes Optimierungs-System, welches die oben genannten Punkte und noch weitere Leistungsverbesserungen umsetzt. [20]
Zusammenfassend kann man also sagen, dass Datenflussdiagramme extrem nützlich sind, TensorFlow schneller und effizienter macht und zusätzliche Parallelität zulässt. Ein kleines Beispiel für einen solchen Graphen ist in Abbildung 2.10 gezeigt. Dazu wird der Code aus Listing 2.1 verwendet.

```
1 w = a + b
2 x1 = a - c
3 y = x1 + d
4 x2 = a + c
5 z = y + e
```

Listing 2.1: Datenflussdiagramme Beispielcode [63]

Keras (API):
Keras ist eine sehr fähige Programmierschnittstelle, die sich auf maschinelles Lernen spezialisiert hat. Sie erleichtert das Erstellen von Modellen, in dem sie Anwendungen benutzerfreundlicher macht. Sie minimiert die Anzahl von Benutzerinteraktionen und konzentriert sich auf die Geschwindigkeit, Wartungsfreundlichkeit und Einsatzfähigkeit des Codes. Zudem laufen die erstellten Modelle auf vielen verschiedenen Oberflächen (z.B. Server, Mobil, Browser). [23]

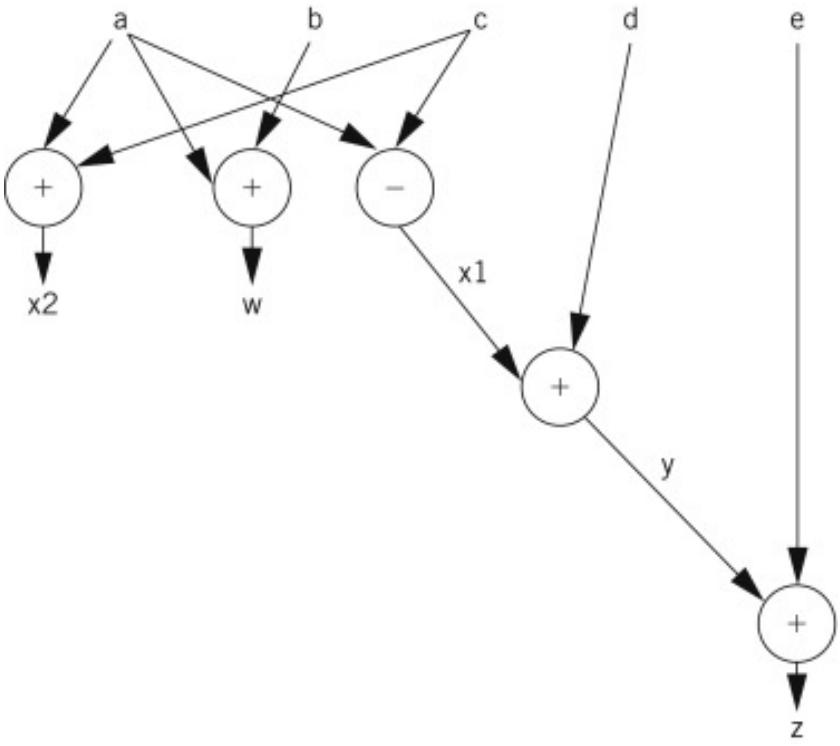


Abbildung 2.10: Datenflussdiagramm zu Listing 2.1 [63]

3 Anforderungsanalyse

Die Anforderungsanalyse bildet das Fundament für ein erfolgreiches System, indem sie die spezifischen Anforderungen, die an das System gestellt werden, erfasst und dokumentiert. Diese Phase legt den Grundstein für die spätere Entwicklung und ist zudem auch die Grundlage für die folgende Evaluation. In diesem Abschnitt wird nicht über Lösungswege diskutiert, sondern vielmehr der Fokus auf die detaillierte Erarbeitung der Anforderungen gelegt. Angefangen wird damit, dass die Stakeholder betrachtet und vorgestellt werden. Daraufhin werden die virtuelle Umgebung und der prototypische Demonstrator separat betrachtet.

3.1 Stakeholder

Stakeholder sind Interessensgruppen, die ein Interesse an dem Gelingen des Projekts haben. Sie können dabei in unterschiedlichen Bereichen des Projekts involviert sein. Die Interessensgruppen des Projekts sind nachfolgend kurz beschrieben.

Auftraggeber
Der Auftraggeber dieser Arbeit ist Herr Prof. Dr. Hensel, der als betreuender Professor und Erstprüfer fungiert. Sein Hauptinteresse besteht darin, eine Plattform für weitere Arbeiten zu schaffen. Neben der fachlichen Qualität ist sein Fokus auf den damit verbundenen Erkenntnisgewinn gerichtet, insbesondere auf die Implementierung eines physikalischen Systems, welches ohne manuelles Eingreifen selbständig trainieren kann. Des Weiteren wird eine positive externe Wirkung und eine robuste Grundlage für weitere Abschlussarbeiten im Bereich des Reinforcement Learning angestrebt.

Soft- und Hardware-Entwickler
Der Entwickler der Soft- und Hardware ist der Ersteller dieser Arbeit. Im Vordergrund

des Soft- und Hardware-Entwicklers steht die Erstellung einer funktionsfähigen Gesamtlösung. Das Hauptinteresse besteht darin, diese im Rahmen einer Bachelorarbeit zu bewältigen.

Aufbauende Projekte durch andere Studierende
Durch weiterführende Projekte anderer Studierenden am selben System, im Ganzen oder nur in Teilen, ist eine gute Dokumentation erforderlich. Diese dient zum besseren Verständnis und einer schnelleren Einarbeitungsphase. Das Hauptinteresse dieser Gruppe besteht also in einer guten Dokumentation und einer einfach zu steuernden Plattform. Dies betrifft sowohl die Hardware, als auch die Software.

3.2 Virtuelle Umgebung

Bei der Definition von Anforderungen wird zwischen funktionalen (**F**) und nicht-funktionalen (**NF**) Anforderungen unterschieden. Diese Anforderungen werden im Evaluationsprozess aufgenommen und überprüft.
Funktionale Anforderungen legen fest, welche konkreten Aufgaben oder Funktionen das System erfüllen muss. Sie beschreiben die spezifischen Handlungen oder Aktionen, die das System ausführen soll.
Nicht-funktionale Anforderungen hingegen konzentrieren sich auf die Art und Weise, wie das System diese Funktionen ausführen oder die Dienste erbringen soll.
In diesem Abschnitt werden die Anforderungen an die virtuelle Umgebung (**VU**) beschrieben.

3.2.1 Anwendungsfälle

In Abbildung 3.1 ist das Anwendungsfall-Diagramm der virtuellen Umgebung zu sehen. Das Diagramm bietet eine Übersicht über die involvierten Teilsysteme. In den Tabellen 3.1 bis 3.6 sind die Anwendungsfälle kurz beschrieben.

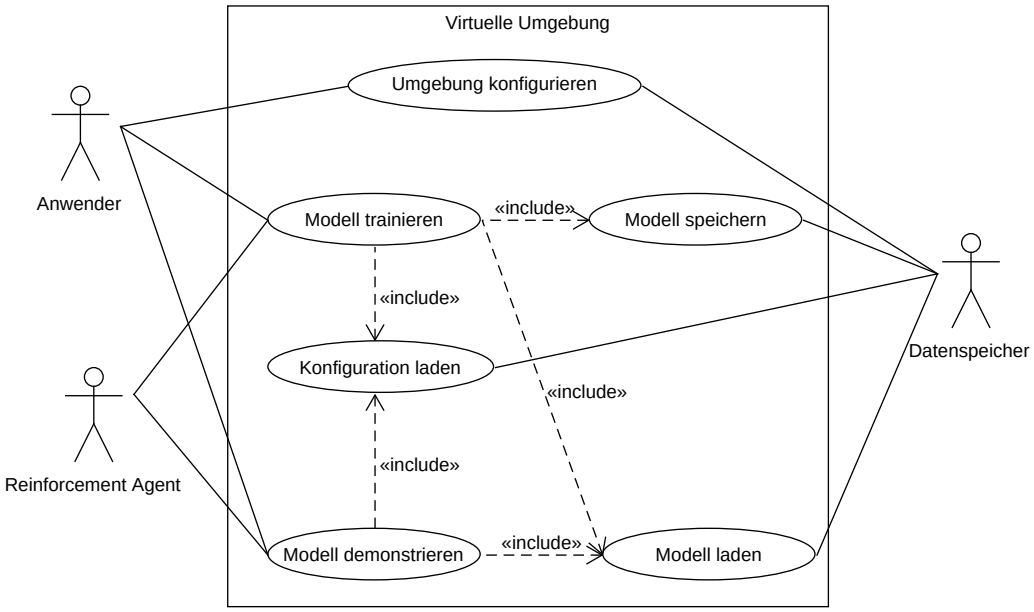


Abbildung 3.1: Anwendungsfall-Diagramm für die virtuelle Umgebung

Tabelle 3.1: Anwendungsfall virtuelle Umgebung: Umgebung konfigurieren

Name	Umgebung konfigurieren
Beschreibung	Die Umgebungs-Parameter der virtuellen Umgebung können angepasst werden.
Ergebnis	Die virtuelle Umgebung wurde konfiguriert.
Involvierte Aktoren/ Teilsysteme	Anwender Datenspeicher
Ablauf	Der Anwender ändert die gewünschten Parameter an zentraler Stelle im Programm.

Tabelle 3.2: Anwendungsfall virtuelle Umgebung: Modell trainieren

Name	Modell trainieren
Beschreibung	Der Reinforcement Agent trainiert in Testdurchläufen seine Fähigkeiten.
Ergebnis	Der Reinforcement Agent hat Fähigkeiten erlernt.
Involvierte Aktoren/ Teilsysteme	Anwender Reinforcement Agent
Ablauf	1) Benutzer gibt die Anzahl der gewünschten Durchläufe an. 2) Benutzer startet das Training.

Tabelle 3.3: Anwendungsfall virtuelle Umgebung: Modell demonstrieren

Name	Modell demonstrieren
Beschreibung	Der Reinforcement Agent demonstriert seine Fähigkeiten.
Ergebnis	Der Reinforcement Agent hat die erlernten Fähigkeiten demonstriert.
Involvierte Aktoren/ Teilsysteme	Anwender Reinforcement Agent
Ablauf	1) Benutzer gibt die Anzahl der gewünschten Durchläufe an. 2) Benutzer startet die Demonstration.

Tabelle 3.4: Anwendungsfall virtuelle Umgebung: Konfiguration laden

Name	Konfiguration laden
Beschreibung	Die Umgebungs-Parameter, die konfiguriert wurden, werden angewandt.
Ergebnis	Die virtuelle Umgebung wird mit den angegeben Parametern erstellt.
Vorbedingung	Die virtuelle Umgebung wurde konfiguriert. Und das Modell soll trainiert bzw. demonstriert werden.
Involvierte Aktoren/ Teilsysteme	Datenspeicher
Ablauf	Die virtuelle Umgebung wird mit angegeben Parametern erzeugt.

Tabelle 3.5: Anwendungsfall virtuelle Umgebung: Modell speichern

Name	Modell speichern
Beschreibung	Die Reinforcement-Parameter, die in voraufgegangenem Training ermittelt wurden, werden in einer Datei gespeichert.
Ergebnis	Man kann zu jeder Zeit auf trainierte Zustände zurückgreifen.
Vorbedingung	Das Modell wurde trainiert.
Involvierte Aktoren/ Teilsysteme	Datenspeicher
Ablauf	Es wird eine Datei erzeugt oder überschrieben.

Tabelle 3.6: Anwendungsfall virtuelle Umgebung: Modell laden

Name	Modell laden
Beschreibung	Die Reinforcement-Parameter, die im voraufgegangenem Training ermittelt wurden, können aus einer Datei gelesen werden.
Ergebnis	Der Agent muss nicht nach jedem Start neu trainiert werden.
Vorbedingung	Das Modell soll trainiert bzw. demonstriert werden.
Involvierte Aktoren/ Teilsysteme	Datenspeicher
Ablauf	Es wird auf eine bestehende Datei zugegriffen.

3.2.2 Anforderungen

VU-F1: Es gibt die Möglichkeit, die durch Training ermittelten Parameter in einer HDF5 Datei zu speichern.
Dieses Format ist mittlerweile offener Standard, um große numerische Daten schnell zu speichern. [50] Zudem wird es durch bereits bestehende Reinforcement Learning Bibliotheken verwendet. [59]

VU-F2: Es existiert ein Modus, welcher als Trainings-Modus bezeichnet wird. Dieser ist dazu gedacht, das System anzulernen, indem es Testdurchläufe absolviert und die daraus entstehenden Daten analysiert. Die Anzahl der Testdurchläufe soll variabel sein, damit man unterschiedliche Fähigkeitsgrade anlernen kann.

VU-F3: Es existiert ein Modus, welcher als Demonstrations-Modus bezeichnet wird. Dieser ist dazu gedacht, dem Anwender die Fähigkeiten des Systems zu demonstrieren. Die Anzahl der zu demonstrierenden Durchläufe soll variabel sein, um mehreren Systemantworten des Systems betrachten zu können.

VU-F4: Die Kollisionen sollen nach Abschnitt 2.2 physikalisch korrekt abgebildet werden, damit eine möglichst hohe Vergleichbarkeit zwischen realem und virtuellem Spiel gewährleisten werden kann.

VU-F5: Die Kugel wird nach einem Durchgang (Kugel ist am Flippern vorbei) zufällig am oberen Ende des Spielfelds platziert.
Diese Anforderung ersetzt eine Ballrückführung und sorgt für einen kontinuierlichen Spielfluss.

VU-NF1: Die grafische Darstellung der Umgebung ist dreidimensional, um dem Anwender eine ansehnliche Darstellung zu bieten.

VU-NF2: Bei der Entwicklung des Programms ist darauf zu achten, eine klare Struktur zu schaffen, um eine einfache Lesbarkeit und Wartbarkeit zu gewährleisten.
Dazu zählt die Gliederung zusammenhängender Strukturen in Klassen, das Erstellen von

Methoden und die Benutzung von aussagekräftigen Variablennamen. Dies vereinfacht unter anderem auch zukünftige Weiterentwicklungen.

VU-NF3: Die virtuelle Umgebung besitzt verhältnismäßig die gleichen Maße, die in Abschnitt 2.1 aufgelistet sind.
Dies dient dazu, eine möglichst hohe Vergleichbarkeit zwischen realem und virtuellem Aufbau gewährleisten zu können.

VU-NF4: Die Flipper sollen trichterförmig angeordnet werden. Der Winkel zur Waagerechten beträgt hierbei 35°.
Der Wert von 35° ergibt sich durch geometrisches Ausmessen eines bereits existierenden Flippers. [16]

VU-NF5: Die Flipper bewegen sich insgesamt um 70°.
Der Flipper hat in Ruheposition einen Winkel von 35° zur Waagerechten. Damit der Ball jeden Bereichs des Spielfelds erreichen kann, bewegt sich der Flipper 35° über die Waagerechte.

VU-NF6: Die virtuelle Umgebung soll konfigurierbar sein, um Änderungen der Maße schnell umsetzen/ändern zu können. Zu den konfigurierbaren Teilen zählt die Größe des Spielfeldes, die Größe und Position der Flipper, der Abstand zwischen den Flippern, die Größe der Elemente auf dem Spielfeld, die Neigung der Platte sowie die Größe und das Gewicht der Kugel.

3.3 Prototypischer Demonstrator

Nachfolgend werden die Anforderungen an den prototypischen Demonstrator (**DEMO**) beschrieben. Der Demonstrator wird der Übersicht halber in Allgemeine- (**A**), Hardware- (**HW**) und Softwareanforderungen (**SW**) unterteilt.

3.3.1 Anwendungsfälle

In Abbildung 3.2 ist das Anwendungsfall-Diagramm des prototypischen Aufbaus zusehen. In den Tabellen 3.7 bis 3.10 sind die Anwendungen beschrieben.

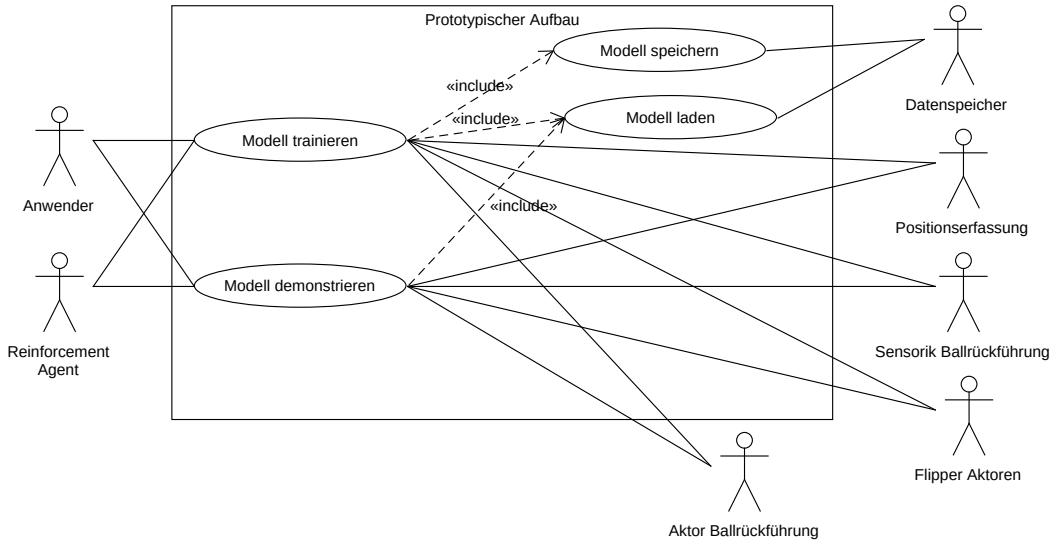


Abbildung 3.2: Anwendungsfall-Diagramm für den prototypischen Aufbau

Tabelle 3.7: Anwendungsfall prototypischer Aufbau: Modell trainieren

Name	Modell trainieren
Beschreibung	Der Reinforcement Agent trainiert in Testdurchläufen seine Fähigkeiten.
Ergebnis	Der Reinforcement Agent hat Fähigkeiten erlernt.
Involvierte Aktoren/ Teilsysteme	Anwender Reinforcement Agent Positionserfassung Sensorik Ballrückführung Aktor Flipper Aktor Ballrückführung
Ablauf	1) Der Benutzer gibt die Anzahl der gewünschten Durchläufe an. 2) Der Benutzer startet das Training.

Tabelle 3.8: Anwendungsfall prototypischer Aufbau: Modell demonstrieren

Name	Modell demonstrieren
Beschreibung	Der Reinforcement Agent demonstriert seine Fähigkeiten.
Ergebnis	Der Reinforcement Agent hat die erlernten Fähigkeiten demonstriert.
Involvierte Aktoren/ Teilsysteme	Anwender Reinforcement Agent Positionserfassung Sensorik Ballrückführung Aktoren Flipper Aktor Ballrückführung
Ablauf	1) Der Benutzer gibt die Anzahl der gewünschten Durchläufe an. 2) Der Benutzer startet die Demonstration.

Tabelle 3.9: Anwendungsfall prototypischer Aufbau: Modell speichern

Name	Modell speichern
Beschreibung	Die Reinforcement-Parameter, die in voraufgegangenem Training ermittelt wurden, werden in einer Datei gespeichert.
Ergebnis	Der Benutzer kann zu jeder Zeit auf trainierte Zustände zurückgreifen.
Vorbedingung	Das Modell wurde trainiert.
Involvierte Aktoren/ Teilsysteme	Datenspeicher
Ablauf	Es wird eine Datei erzeugt oder überschrieben.

Tabelle 3.10: Anwendungsfall prototypischer Aufbau: Modell laden

Name	Modell laden
Beschreibung	Die Reinforcement-Parameter, die in voraufgegangenem Training ermittelt wurden, können aus einer Datei gelesen werden.
Ergebnis	Der Agent muss nicht nach jedem Start neu trainiert werden.
Vorbedingung	Das Modell soll trainiert bzw. demonstriert werden.
Involvierte Aktoren/ Teilsysteme	Datenspeicher
Ablauf	Es wird auf eine bestehende Datei zugegriffen.

3.3.2 Anforderungen

Allgemein

Dieser Abschnitt stellt allgemeine Anforderungen an den Demonstrator. Dazu zählen Kosten, Funktionsweise und Sicherheitsaspekt.

DEMO-A-NF1: Die Projektkosten dürfen 150 € für Neuanschaffungen nicht überschreiten.
Dieser Wert erschließt sich aus den Vorgaben des Departments für Informations- und Elektrotechnik an der HAW Hamburg.

DEMO-A-NF2: Die Gefährdung durch Bälle außerhalb des Spielfeldes ist zu vermeiden, um Verletzungen vorzubeugen.

DEMO-A-NF3: Die Gefährdung einer Person durch Elektrizität ist auszuschließen.

Hardware

Im Folgenden werden Hardware-Anforderungen an den Demonstrator gestellt. Dazu zählen alle Anforderungen, die den physischen Aufbau betreffen.

DEMO-HW-F1: Die Flipper bewegen sich insgesamt um 70°.
Der Flipper hat in Ruheposition einen Winkel von 35° zur Waagerechten. Damit der Ball jeden Bereichs des Spielfelds erreichen kann, bewegt sich der Flipper 35° über die Waagerechte.

DEMO-HW-F2: Nachdem der Ball an den Flippern vorbei gefallen ist, soll der Ball spätestens nach 5 s wieder auf dem Spielfeld sein.
Dies dient dazu, den Spielfluss konstant zuhalten und die Zeit für den Lernprozess in einem kalkulierbaren Rahmen zu halten.

DEMO-HW-NF1: Das Ziel, welches getroffen werden soll, wird als Kreis auf dem Spielfeld dargestellt. Sobald der Ball sich auf dem Kreis befindet wird dem Reinforcement System eine Belohnung übergeben.
Bei handelsüblichen Flipper-Automaten, gibt es diverse Ziele, die getroffen werden können [62]. In diesem Fall wird sich für einen Kreis auf dem Spielfeld entschieden, da dies am wenigsten Ressourcen in Anspruch nimmt.

DEMO-HW-NF2: Das Spielfeld soll die Abmessungen von 50x34 cm aufweisen.
Wie in Abschnitt 2.1 beschrieben ist die Größe von Flippern sehr variabel. Die gängigen Maßen bewegen sich um 107 x 51,5 cm herum [35]. Um das gängige Seitenverhältnis ungefähr beizubehalten, aber dennoch genügend Platz in der Breite für Auf- und Einbauten zu haben, wird sich für die Maße 50 x 34 cm entschieden. Ein Grund für die kleinere Dimensionierung ist, dass der Prototyp transportabel sein soll und platzsparend gelagert werden kann.

DEMO-HW-NF3: Die Flipper sollen eine Länge von 4,5 cm aufweisen.
Wie in Abschnitt 2.1 beschrieben, beträgt die Länge eines Flipper üblicherweise 3 Zoll, also 7,6 cm. Aufgrund der verringerten Spielfeldgröße wird sich in diesem Fall für eine Länge von 4,5 cm entschieden.

DEMO-HW-NF4: Das Spielfeld soll eine Neigung von 7° betragen, damit der Ball durch die Schwerkraft immer Richtung Flipper rollt.
Die 7° entstammen der allgemein gängigen Praxis. [37]

DEMO-HW-NF5: Die Auswahl der Elektronikbauteile sollte auf eine breite Kompatibilität ausgerichtet sein, um eine zukünftige Ersetzung oder eine anderweitige Weiterverwendung zu gewährleisten.

DEMO-HW-NF6: Die bewegbaren Teile sind farblich rot zu kennzeichnen, um auf die ausgehende Gefahr von diesen Bauteilen hinzuweisen.

DEMO-HW-NF7: Die Flipper sollen trichterförmig angeordnet werden. Der Winkel zur Waagerechten beträgt hierbei 35°.

Der Wert von 35° ergibt sich durch geometrisches Ausmessen eines bereits existierenden Flippers. [16]

DEMO-HW-NF8: Die Bauteile sollen so ausgewählt werden, dass sie für künftige Projekte an der HAW wiederverwendbar sind.

Software

Im Folgenden werden Software-Anforderungen an den Demonstrator gestellt. Dazu zählt vor allem die Steuerung der Hardware.

DEMO-SW-F1: Es soll möglich sein, Parameter des Reinforcement Learnings aus der virtuellen Umgebung auf das Reinforcement Learning des Prototypen zu übertragen.

DEMO-SW-F2: Es existiert ein Modus, welcher als Demonstrations-Modus bezeichnet wird. Dieser ist dazu gedacht, die Fähigkeiten des Systems dem Anwender zu demonstrieren. Die Anzahl der zu demonstrierenden Durchläufe soll variabel sein, um mehreren Systemantworten des Systems betrachten zu können.

DEMO-SW-F3: Es existiert ein Modus, welcher als Trainings-Modus bezeichnet wird. Dieser ist dazu gedacht, das System anzulernen, indem es Testdurchläufe absolviert und die daraus entstehenden Daten analysiert. Die Anzahl der Testdurchläufe soll variabel sein, damit man unterschiedliche Fähigkeitsgrade anlernen kann.

DEMO-SW-F4: Die Ball-Position soll mittels eines Kamerasystems erfasst werden. Zudem soll aus dem Kamerabild automatisch das Spielfeld extrahiert, entzerrt und auf die Maße des Spielfelds umgerechnet werden.
Durch dieses Vorgehen erhält man einen Bereich mit festen Grenzen, in dem sich die Kugel befinden kann. Dies fördert die Vergleichbarkeit zwischen realem und virtuellem Aufbau.

DEMO-SW-F5: Die Bewegungen der Flipper soll synchron zueinander sein. Dies bedeutet, dass erst eine neue Bewegung ausgeführt werden kann, sofern ein oder beide

Flipper die obere oder untere Endlage erreicht hat.
Mögliche Bewegungen: Beide Flipper schlagen aus, jeweils ein Flipper schlägt aus, beim Zurückfahren eines Flippers kann der andere wieder ausschlagen.

4 Konzept

Dieser Abschnitt beschäftigt sich damit, ein Konzept aus den unter Kapitel 3 aufgeführten Anforderungen zu erarbeiten. Wie bei den Anforderungen wird zwischen virtueller Umgebung und Demonstrator unterschieden.

4.1 Virtuelle Umgebung

In diesem Abschnitt wird ein Konzept für die virtuelle Umgebung erstellt.

4.1.1 Entwicklungsumgebung

In diesem Abschnitt wird sich damit beschäftigt, welche Programmiersprache verwendet wird, sowie welche Bibliotheken und Frameworks zur Verwendung kommen.
Es gibt zum Zeitpunkt der Recherche (2024) insgesamt über 350 Programmiersprachen [57], zwischen denen ausgewählt werden kann. Da das Kernelement dieser Arbeit Künstliche Intelligenz ist, ist es sinnvoll zuschauen, welche der gängigen Sprachen das größte Angebot zu diesem Thema hat. Python ist dabei sehr weit vorne, da es umfangreiche Bibliotheken besitzt, die sich auf Bereiche der Künstlichen Intelligenz spezialisiert haben. Zudem ist der Support, den man im Internet erhält beachtlich, da die Sprache sowohl anfängerfreundlich, als auch für den professionellen Gebrauch geeignet ist. Eine Alternative zu Python wäre Java, dessen größter Vorteil die Plattformunabhängigkeit ist. Diese sorgt dafür, dass Anwendungen ohne Änderung auf mehreren Plattformen eingesetzt werden können. Da in diesem Fall nicht die Plattformunabhängigkeit der wichtigste Punkt ist, sondern die Unterstützung durch Bibliotheken, wird sich für Python entschieden. [47]
Bei der Auswahl von eines Reinforcement Learning Umgebungs-Framework, wird sich für OpenAI Gym entschieden. Die Auswahl des Reinforcement Learning Agenten-Frameworks fällt auf Tensorflow. Diese Entscheidungen wurden getroffen, da Tensorflow bereits

passgenaue Lösungen/Lösungsansätze für die Umgebungen von OpenAI Gym anbietet, auf die aufgebaut werden kann. Zusätzlich gibt es bereits vorgefertigte Methoden, für das Training und Demonstrieren eines Agenten sowie das Speichern bzw. Laden der ermittelten Parameter. Somit wären die Anforderungen VU-F1, VU-F2 und VU-F3 bereits erfüllt. [11]
Da durch Anforderung VU-NF1 gefordert wird, dass der virtuelle Flipper dreidimensional dargestellt werden soll, wird sich für die Python Bibliothek *VPython* entschieden, da bei dieser bereits Kenntnisse durch Arbeiten von Herrn Prof. Dr. Hensel bestehen.

4.1.2 Algorithmus

Die Auswahl des Algorithmus wird auf die in Abschnitt 2.2.1 aufgeführten Algorithmen begrenzt. Aus den drei Algorithmen wird der Deep Q-Learning Algorithmus verwendet, da dieser am besten mit großen Zustandsräumen umgehen kann, was bei dem Flipper von Vorteil ist, da die Position des Balls ein großer Zustandsraum ist.

4.1.3 Kollisionen und Spielfeld

Wie in VU-F4 gefordert, soll die Kollision physikalisch korrekt abgebildet werden. Da bei den Kollisionen keine bleibenden Verformungen auftreten, handelt es sich um eine elastische Kollision. Für die Umsetzung werden die physikalischen Formeln aus Abschnitt 2.2 betrachtet. Zudem wird auf bereits selbständig erarbeiteten Umsetzungen aus dem Wahlpflichtfach *Introduction to Computer Graphics* von Herrn Prof. Dr. Jünemann zurückgegriffen. In Anforderung VU-F5 wird gefordert, dass der Ball nach einem Durchlauf am oberen Spielfeldrand platziert werden soll, um die Ballrückführung wie beim prototypischen Aufbau zu imitieren. Durch die Verwendung der Bibliothek *VPython* ist das Positionieren des Balls einfach umzusetzen. Genau wie bei der Anforderung VU-F5, gilt bei den Anforderungen VU-NF3 (verhältnismäßig gleiche Maße) und VU-NF4 (Flipper sollen trichterförmig angeordnet werden), dass durch die Verwendung der Bibliothek *VPython* die Umsetzung recht einfach realisiert werden kann.

4.1.4 Ablauf

Die Interaktion des Anwenders mit dem System erfolgt durch kurze Programmieranweisungen in der main-Methode. Die Eingaben, die man dort tätigen kann, sind entweder ein

Training anzustoßen, sich die Fähigkeiten demonstrieren zu lassen oder die Parameter zu speichern bzw. zu laden. Sofern ein Training absolviert werden soll, wird die angegebene Anzahl von Schritten abgearbeitet. Der Anwender hat die Möglichkeit die dort ermittelten Parameter in einer Datei zu speichern, vorausgesetzt die Programmieranweisung wurde zuvor getätigt. Nun hat man die Möglichkeit sich die Fähigkeiten demonstrieren zu lassen. In dem Fall, dass zuvor schon Parameter ermittelt wurden, lassen diese sich vor dem Training bzw. dem Demonstrieren laden.

4.2 Prototypischer Demonstrator

Der prototypische Demonstrator umfasst sowohl den physischen Aufbau, als auch die Steuerung für den Aufbau.

4.2.1 Auswahl der Kugel

Bei der Kugel kann zwischen einem Tischkickerball, einer GraviTrax-Kugeln und einer Murmel entschieden werden. Diese wurden ausgewählt, da sie weit verbreitet und leicht zu beschaffen sind.

- Tischkickerball hat einen Durchmesser von ca. 32-36 mm und wiegt 20 bis 30 g. [24]
- GraviTrax-Kugel hat einen Durchmesser von ca. 12,7 mm und wiegt 9 g. [25]
- Die verbreitetsten Murmeln haben einen Durchmesser von ca. 16 mm und wiegen 5,5 g. [18]

Der Durchmesser und das Geweicht einer Standard-Flipperkugel beträgt ca. 27 mm und 81 g [15], da die Maße des Spielfeld nach Anforderung DEMO-HW-NF3 ungefähr halbiert wurden, würde der verhältnismäßig korrekte Durchmesser ca. 13,5 mm und das Gewicht ca. 40 g betragen. Bei den aufgeführten Kugeln findet man keine Kugel, bei der beide Anforderungen passen. Ein verhältnismäßig korrekter Durchmesser hat eine höhere Priorität als das Gewicht, da bei abweichenden Durchmesser alle anderen Bauteile angepasst werden müssten. Aus diesem Grund wird sich für die GraviTrax-Kugel entschieden, da sie näher am optimalen Durchmesser liegt und zudem noch schwerer ist als eine 16 mm Murmel.

4.2.2 Rahmen

Der Rahmen lässt sich durch verschiedene Materialien realisieren. In diesem Fall wurden Aluprofile und Kantholz betrachtet. Diese beiden Varianten werden nachfolgend anhand von zwei Kriterien, Wiederverwendbarkeit und Anschaffungskosten, verglichen und nach Vor- und Nachteilen abgewogen.

Wiederverwendbarkeit

Beide Materialien müssen auf die Länge der Platte zugeschnitten werden. Dieser Schritt lässt sich nicht vermeiden und verändert das Material bleibend. Betrachtet man davon abgesehen die weiteren erforderlichen Eingriffe, kristallisieren sich deutliche unterscheide. Die Aluprofile können weitestgehend ohne bleibende Veränderung wiederverwendet werden, da diese mit Hilfe von Nutensteine befestigt werden können. Nutensteine lassen sich einfach in die Nut einschieben und besitzen ein Gewinde, um verschraubt zu werden. Zudem gibt es passende Endstücke, die die Kanten abdecken und noch viele weitere Einsätze. Durch diese Eigenschaften ist der Aufbau sehr flexibel. Holz hingeben muss in weiteren Arbeitsschritten bearbeitet werden, um Verbindungen zu schaffen. Diese sind irreversibel und zeitintensiv.

Anschaffungskosten

Die Anschaffungskosten sind in Tabelle 4.1 zu sehen. Die Preise stammen aus dem Jahr 2023 und können variieren. Das Spielfeld soll nach der Anforderungen DEMO-HW-NF3 Maße von 50x34 cm betragen und soll nach Anforderungen DEMO-A-NF1 das Budget von 150 € nicht übersteigen. Um eine möglichst hohe Bande zu erhalten, wird der Rahmen auf der Platte montiert, somit muss die Breite des Rahmens auf die Maße des Spielfeldes addiert werden. Bei der Breite und Höhe des Rahmens wird sich für 20 mm entschieden, da im Anforderung DEMO-A-NF2 gefordert wird, dass der Ball keine Gefährdung außerhalb des Spielfelds darstellen soll. Durch eine Höhe von 20 mm soll dies gewährleistet werden. Somit ergeben sich die neuen Maße von 54x38 cm. Der Umfang beträgt dann 184 cm. Der Preis bezieht sich auf die 184 cm.

Tabelle 4.1: Übersicht Rahmen-Kosten [1, 31]

	Aluprofil	Kantholz
Preis in €/184 cm	15,33	5,14

Auswertung

Nach dem Vergleich der beiden Materialien wird sich für die Aluprofile entschieden. Dadurch werden weitere aufwendige Arbeitsschritte vermieden. Der Preisunterschied ist prozentual beachtlich, jedoch betragsmäßig gering und im Budget. Ausschlaggebend ist auch die Wiederverwendbarkeit der Aluprofile, die in Anforderung DEMO-HW-NF9 festgehalten ist.

4.2.3 Basisplatte

Für die Konstruktion der Basisplatte gibt es die Möglichkeit, diese als Aluverbund-, Holz- oder PVC-Platten zu realisieren. Diese drei Varianten werden nachfolgend anhand von zwei Kriterien, Wiederverwendbarkeit und Anschaffungskosten, verglichen und anschließend nach Vor- und Nachteilen abgewogen.

Wiederverwendbarkeit

Es ist unvermeidbar, dass Löcher in die Platte gebohrt werden. Im nachfolgenden Text wird dies nicht berücksichtigt, da davon ausgegangen wird, dass große Teile der Platte unversehrt bleiben und gegebenenfalls wiederverwendet werden können.
Bei der Wiederverwendbarkeit unterscheiden sich die Platten nicht stark von einander. Der einzige Unterschied ist, dass sowohl die Aluverbundplatte als auch die PVC-Platte für den Innen- und Außenbreiche genutzt werden können, da diese wetterbeständig sind. Nicht jede Art von Multiplexplatten ist wetterbeständig, dafür muss ein wasserbeständiger Leim verwendet werden oder die Platte muss nachträglich behandelt werden. [28] Dies ist mit zusätzlichem Aufwand verbunden. Die Unterscheidung ist wichtig, da zukünftige Arbeiten diese Platten im Außenbereich nutzen könnten.

Anschaffungskosten

Die Anschaffungskosten der verschiedenen Platten sind in Tabelle 4.2 zusehen. Es ist zu erwähnen, dass diese Preise aus dem Jahr 2023 stammen und variieren können. Nach DEMO-HW-NF3 soll das Spielfeld eine Größe 50x34 cm haben und nach Anforderungen DEMO-A-NF1 soll das Budget nicht 150€ übersteigen. Kombiniert man dies mit dem Durchmesser des Rahmen, erhält man die Maße für die Gesamtplatte, diese beträgt dann 54x38 cm. Der Preis bezieht sich somit auf die Größe der Gesamtplatte. Es wurde eine wasserbeständige Multiplexplatte ausgewählt, um eine bessere Vergleichbarkeit untereinander zu erhalten.

Auswertung

Nach der Sichtung und der Recherche ist zu erkennen, dass die Materialien eine ähnliche Wiederverwendbarkeit haben. Da das System portabel sein soll, wird sich für die Aluverbundplatte entschieden, diese überzeugt durch ihre Stabilität und Leichtigkeit. Die Multiplexplatte wäre in der kleinsten Materialstärke immer noch doppelt so dick wie Aluverbund- oder PVC-Platte, wodurch das Gewicht des Gesamtsystems merklich zunehmen würde. Die PVC-Platte ist deutlich günstiger als die andern Varianten, was in diesem Fall jedoch keinen großen Unterschied ausmacht und das Budget aus den Anforderungen DEMO-A-NF1 nicht ausreizt.

4.2.4 Elemente auf dem Spielfeld

Die Flipper und Hindernisse auf dem Spielfeld lassen sich sowohl aus 3D-gedruckten Teilen als auch aus Holz fertigen. Die Verbindung zwischen der Spielfeldplatte und den Hindernissen auf dem Spielfeld soll mittels Bolzen bzw. Stifte auf der Unterseite der Hindernisse und Löchern auf der Spielfeldplatte beziehungsweise Verschraubungen am Alurahmen realisiert werden. Elemente, die den Ball in eine Richtung leiten, sollen am

Tabelle 4.2: Übersicht: Platten-Kosten [2, 7, 30]

	Aluverbund	Multiplex	PVC
Materialstärke in mm	3	6,5	3
Preis	15,33 €	15,80 €	7,88 €

Rahmen montiert werden. Die dadurch entstehenden Elemente haben komplexere Formen, wodurch die Arbeit mit Holz deutlich zeitaufwendiger wäre. Aufgrund der Einfachheit und der großen Flexibilität von 3D-Druck ist eine Gegenüberstellung wie bei dem Spielfeld und Rahmens nicht nötig. Es wird sich für 3D-gedruckte Elemente entschieden.

4.2.5 Ballrückführung

Die Ballrückführung besteht aus einer Einheit, die den Ball zurück ins Spielfeld bringt (*Rückführungseinheit*), und einer *Auffangeinheit*, die den Ball nach Erreichen des Endzustands zu der Rückführungseinheit leitet.

Auffangeinheit

Da das Spielfeld nach Anforderung DEMO-HW-NF4 eine Neigung von 7° betragen soll, kann sich die Schwerkraft zu nutze gemacht werden. Der Endzustand wird erreicht, sobald der Ball an den Flippern vorbei gefallen ist. Der Ball wird dann mithilfe von einer Bahn, zu der unteren rechten Ecke geleitet, in der die *Rückführungseinheit* platziert ist. Auch hier kann wieder zwischen 3D-Druck und Holz ausgewählt werden. Jedoch ergeben sich nach kurzer Überlegung deutliche Vorteile, die für den 3D-Druck sprechen.
Zum Einen können mit 3D-gedruckten Elementen einfacher Kurven oder andere komplexere Formen realisiert werden. Bei einer Holzkonstruktion wäre dies nur mit viel Zeit, aufwändigen Arbeitsschritten und speziellem Werkzeug zu bewältigen. Hinzu kommt auch die bessere Handhabung im Falle einer Beschädigung. Die 3D-gedruckten Elemente, lassen sich einfacher reproduzieren, da die Dateien für den 3D-Drucker schon existieren und nicht neu entwickelt werden müssen. Zudem sind Elemente aus 3D-Druck im Gegensatz zu Holz deutlich leichter und tragen somit dazu bei, das Gesamtgewicht auf einem niedrigen Niveau zu halten.

Rückführungseinheit

Die Rückführungseinheit sorgt dafür, den Ball wieder ins Spielfeld zu bringen. Hierfür gibt es mehre Möglichkeiten, einerseits wie beim Original, mit einer Metallstange inmitten einer Feder, die heruntergezogen werden kann oder mittels eines alleinstehenden Elektromotors. Die Umsetzung eines alleinstehenden Elektromotor bietet den Vorteil,

dass keine weiteren Komponenten in diesen Prozess involviert sind, zudem bietet sich an den Motor am Rahmen zu befestigen. Die erste Variante beinhaltet mechanische Komponenten, außerdem müsste das Spannen der Feder auf andere weise als beim Original passieren, da beim Original der Spieler die Feder auf Spannung bringt. Auf Grundlage der genannten Faktoren wird sich für den alleinstehenden Elektromotor entschieden, um Komplikationen durch mechanische Komponenten auszuschließen.

4.2.6 Flipper

Die Flipper sind die einzige Interaktionsmöglichkeit mit dem System. Nach Anforderung DEMO-HW-NF7 sollen diese trichterförmig eingebaut werden. Zudem sollen die Flipper nach Anforderung DEMO-HW-NF6 rot gekennzeichnet sein, um die von der Bewegung ausgehende Gefahr optisch hervorzuheben. Bei der Entscheidung welches Material für die Flipper verwendet wird, kann die gleiche Argumentation wie bei den Elementen auf dem Spielfeld angewendet werden. Folglich werden für die Flipper rote 3D-gedruckte Elemente verwendet. In diesem Schritt wird auch die in Anforderung DEMO-HW-NF3 geforderten Länge von 4,5 cm berücksichtigt. Für die Bewegung muss ein elektrischer Antrieb ausgewählt werden. Hierbei kann zwischen einem Elektromagneten wie im Original oder einem Elektromotor entschieden werden.
Die Umsetzung mittels eines Elektromagneten wie beim Original, würde einen weiteren Aufbau mit Endlagen sowie weitere bewegliche Teile fordern (siehe Abbildung 2.2). Die Verwendung eines Elektromotors würde nur das Anbringen des Flippers an der Welle erfordern. Als Elektromotortyp wurde ein Schrittmotor gewählt, da dieser präzise Teildrehungen ausführen kann, was durch die 70° in Anforderung DEMO-HW-F1 gefordert wird. Zudem ist die trichterförmige Ausgangsstellung in Anforderung DEMO-HW-NF7 auch kein Problem für Schrittmotoren. Ein weiterer Aspekt ist, dass Elektromotoren vielseitigere Einsatzmöglichkeiten haben, was für die Wiederverwendbarkeit nach Anforderung DEMO-HW-NF5 spricht. Aufgrund dessen, dass der Schrittmotor all diese Anforderung besser erfüllt, wird sich für eben diesen entschieden.

4.2.7 Steuerung

Anders als bei der virtuellen Umgebung, bei der alles ohne externe Systeme auf einem Rechner umgesetzt werden kann, wird sich bei der Steuerung des prototypischen Aufbaus dafür entschieden eine Unterteilung in drei Bereiche vorzunehmen. Diese drei

Bereiche werden nachfolgend erklärt und konzipiert. In Abbildung 4.1 sieht man das Zusammenspiel dieser drei Bereiche.

Zentralsteuerung

Genau wie bei der virtuellen Umgebung wird sich für die Programmiersprache Python entschieden, die Entscheidungsfindung ist dieselbe (siehe Abschnitt 4.2.1). Ebenso wird das Reinforcement Learning und die Umgebung von OpenAI Gym übernommen.
OpenAI Gym wird jedoch nicht mehr dazu genutzt, eine virtuelle Umgebung zu erzeugen. Es wird viel mehr die Struktur und die Kompatibilität mit dem Reinforcement Learning System ausgenutzt. Somit werden in OpenAI Gym keine virtuellen Bauteile mehr angesteuert, sondern die physischen Aktoren. Genauso wird nicht mehr eine virtuelle Ballposition an den RL-Agenten (Reinforcement Learning) übergeben, sondern die Position des Balls.
Ein Vorteil der Übernahme ist der, dass die Anforderungen DEMO-SW-F1, DEMO-SW-F2 und DEMO-SW-F3 direkt erfüllt sind. Für das in Anforderung DEMO-SW-F1 geforderte importieren von ermittelten Parametern aus der virtuellen Umgebung, ist nur das Laden der entsprechenden HDF5-Datei notwendig.

Positionserfassung

Für die Positionserfassung und die Umsetzung von DEMO-SW-F4 gibt es kaum eine andere Variante, als die Positionserfassung mithilfe einer Kamera umzusetzen. Der Vorteil dieser Variante ist, dass man einen einfachen Aufbau hat, da man die Kamera nur von einer höheren Position auf das Spielfeld ausrichten muss. Zudem ist die Bildverarbeitung weit verbreitet und angewandt, wodurch die Anwendung und Umsetzung keine große Arbeitsinvestition fordert. [49] Die Position des Balls wird dann dem Python Programm übermittelt, in der es dann den Zustand der Umgebung definiert.

Aktorsteuerung

Für die Aktorsteuerung gibt es die Möglichkeit, diese mithilfe von einem Mikrocontroller umzusetzen. Der Mikrocontroller sollen in diesem Fall als Schnittstelle zwischen Rechner und Aktoren interagieren. Es gibt jedoch viele verschiedene Modelle und Ausführungen von Mikrocontrollern. Da der Mikrocontroller keine große Steuerungsrolle übernimmt,

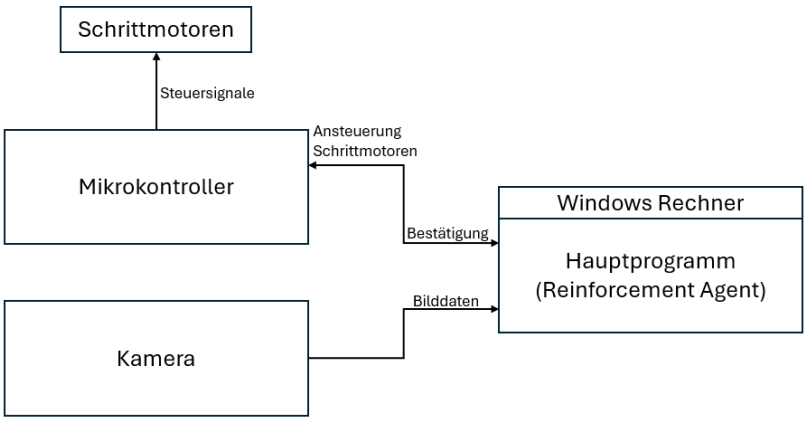


Abbildung 4.1: Aufteilung der Steuerung

wäre ein Raspberry Pi und oder ähnliche leistungsstarke Modelle zu überdimensioniert und teuer, weswegen diese im Weiteren nicht berücksichtigt werden.
Eine simplere Alternative wäre ein Arduino-Modell, diese sind weniger leistungsstark, bieten jedoch eine breite Auswahl an bereits existierenden Bibliotheken, auf die zurückgegriffen werden kann. Zudem kann über die serielle Schnittstelle, der angeschlossene Rechner Befehle zum Ansteuern der GPIO-Pins an den Arduino übermitteln. Gängige Modelle sind der Uno, Nano und Mega, diese sind in Tabelle 4.3 dargestellt. Die Leistungsparameter werden in diesem Fall nicht mit aufgelistet, da diese kein ausschlaggebendes Kriterium sind. In diesem Projekt werden drei Schrittmotoren verbaut, für die Ansteuerung dieser werden jeweils drei Pins benötigt, wobei einer davon ein PWM-Pin ist. Es wären also alle Modelle in der Lage diese Aufgabe zu bewältigen. Somit fällt der Arduino Mega aufgrund des Preises raus. Zwischen Arduino Uno und Nano gibt es nach der Tabelle 4.3 kaum Unterschiede. Aufgrund der Vorrätigkeit bei Herrn Prof. Dr. Hensel und der Einfachheitshalber, wird sich für einen Arduino Uno entschieden.

Tabelle 4.3: Übersicht Arduino-Modell [6, 5, 4]

	Arduino Uno	Arduino Nano	Arduino Mega
Digital I/O Pins	14	14	54
Digital I/O Pins mit PWM	6	6	14
Preis	15,70 €	15,50 €	35,60 €

5 Entwicklung der virtuellen Umgebung

In diesem Kapitel wird die Entwicklung der virtuellen Umgebung, wie in Kapitel 4 beschrieben, umgesetzt. Außerdem werden Schwierigkeiten und Probleme, die bei der Umsetzung aufgetreten sind, beschrieben. Dieses Kapitel ist unterteilt in, die Konstruktion des Modells, der Umsetzung von bewegten Elementen, der physikalischen Kollision und das Anwenden von Reinforcement Learning. Zur besseren Übersicht ist in Abbildung 5.1 ein UML-Diagramm abgebildet, welches die Programmstruktur erkennen lässt.

5.1 Aufbau der Umgebung

In diesem Abschnitt wird der dreidimensionale Aufbau beschrieben.

5.1.1 OpenAI Gym

Die Konstruktion der Umgebung wird unter Verwendung von OpenAI Gym in Python erstellt. Die Erstellung mithilfe von OpenAI Gym kann in Kapitel 2 nachgeschaut werden. Nachfolgend werden zusätzliche Schritte aufgezählt, die alle im Rahmen einer benutzerdefinierten Umgebung von OpenAI Gym verwendet wurden.

5.1.2 Modell

Das Modell wird dreidimensional dargestellt und ist in Abbildung 5.2 zusehen. Zur Umsetzung wurde VPython [39] benutzt, welches eine Python-Bibliothek ist, die sich auf 3D-Grafikmodelle spezialisiert hat. Die erstellten Grafikmodelle werden dann über ein Browserfenster dargestellt. Die Generierung des Tisches und die Steuerung der Flipper wurde in die Klasse *Table* ausgelagert, diese Klasse ist auch in Abbildung 5.1 zu sehen. Da die Bibliothek bereits in anderen Arbeiten verwendet wurde, gab es Beispiel-Code,

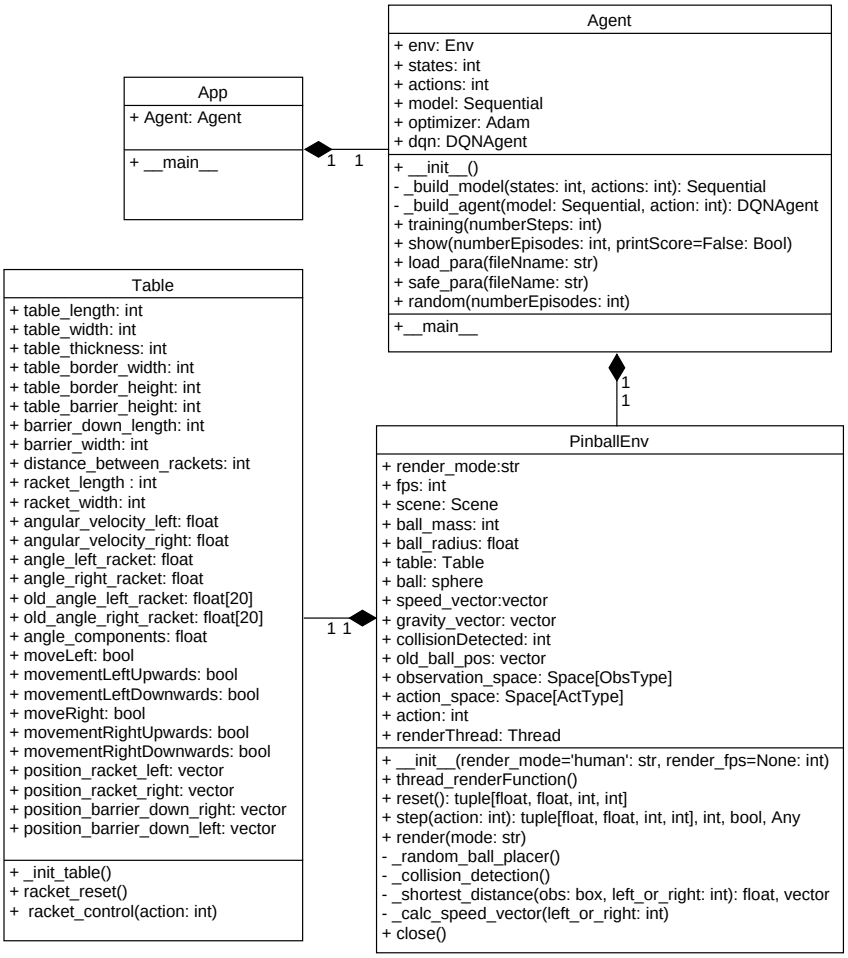


Abbildung 5.1: Klassenübersicht der virtuellen Umgebung

an dem man sich in den ersten Schritten orientieren konnte. Mit vordefinierten Objekten, wie *box*, konnte das Spielfeld konstruiert werden. Diese Objekte sind in allen Bereichen einstellbar, wodurch es möglich war die Objekte passgenau abzustimmen und Rotationsachsen zum Drehen der Objekte hinzuzufügen. Die Kugel ist ein Objekt des Typs *sphere*. Somit konnte man mit mehreren Instanzen dieser vorgefertigten Objekte das Spielfeld konstruieren. Das Modell wurde bei der Initialisierung einmalig generiert. Die Methode dafür ist in Listing 5.1 zu sehen. Die Vektoren in den Objekten dienen nur als Platzhalter, da der vollständige Code zu viel Platz eingenommen hätte.
Bevor man etwas anzeigen kann, muss man das *scene* Attribut zuweisen und dieses parametrieren. Das *scene* Attribut ist dafür verantwortlich eine Leinwand (eng. canvas)

zu erzeugen, auf welcher die Objekte dargestellt werden. Dieses Attribut besitzt genau wie die Objekte diverse Einstellungsmöglichkeiten, so kann man die Höhe und Breite der Leinwand anpassen, die Betrachtungsposition und den Betrachtungswinkel der Szene konfigurieren und den zusehenden Ausschnitt vergrößern oder verkleinern. Die Leinwand wird nach der Initialisierung im Browserfenster angezeigt. Die eingestellten Parameter sind in Listing 5.2 zu sehen.

5.2 Bewegte Elemente

Dieser Abschnitt befasst sich mit den bewegten Elementen in der virtuellen Umgebung und wie diese umgesetzt wurden.

5.2.1 Flipper und Kugel

In der virtuellen Umgebung gibt es drei bewegte Objekte, die beiden Flipper und die Kugel. Die Kugel besitzt einen Geschwindigkeitsvektor, welcher in jedem Durchlauf auf den Kugel-Positionsvektor aufaddiert wird. Um die Gravitation zu simulieren, gibt es einen Gravitationsvektor, der nur eine Z-Komponente besitzt. Dieser wird in jedem Durchlauf auf den Geschwindigkeitsvektor aufaddiert. Dadurch wird gewährleistet, dass die Kugel immer wieder in Richtung der Flipper „fällt“. Die Flipper werden mithilfe einer Drehachse, an dem jeweilig nicht schwenkenden Ende, bewegt. Jeder Flipper besitzt eine Winkelgeschwindigkeit, die aussagt, wie weit sich der Flipper pro Durchlauf bewegt. Die auf dem Bildschirm angezeigten Bewegungen sind ein Zusammenspiel aus der Winkelgeschwindigkeit und den Durchläufen pro Sekunde. Bei einem Winkelgeschwindigkeitswert von 0,25 und 50 FPS, benötigt der Flipper ca. 0,1 s und nimmt dabei sechs Positionen ein. Die Steuerung der Flipper wird in der Klasse *Table* in der Methode *racket_control* behandelt. In dieser wird die aktuelle Aktion ausgewertet und der zugehörige Flipper vor oder zurück bewegt. Wird ein Flipper mehrmals hintereinander angesteuert, bleibt er in der jeweiligen Endlage. Ein Flipper kann nur bewegt werden, wenn keine andere Bewegung aktiv ist. In Listing 5.3 ist die Auswertung der Aktion zu sehen.

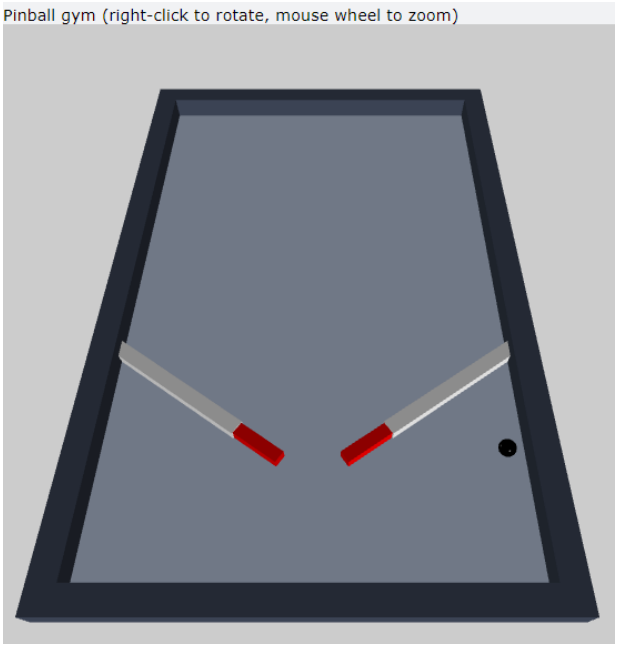


Abbildung 5.2: Modell der virtuellen Umgebung

5.2.2 Kollision

In der virtuellen Umgebung gibt es zwei verschiedene Arten von Kollisionen, eine mit *unbewegten Objekten* und eine mit *bewegten Objekten*. Beide Kollisionen wurden nach dem physikalischen Prinzip der elastischen Kollisionen umgesetzt, für mehr grundlegende Information zu Kollisionen siehe Kapitel 2.
Bei der Kollision mit *unbewegten Objekten* muss man zwischen Objekten parallel zu einer Achse und nicht parallelen Objekten unterscheiden. Bei Objekten parallel zu einer Achse kehrt sich nur der Geschwindigkeitswert der zugehörigen Achse um. Bei nicht parallelen Objekten muss der gesamte Geschwindigkeitsvektor neu berechnet werden, die dafür verwendete Formel ist 2.10 auf Seite 12. Bei dieser wird $v_2 = -v_1$ gesetzt, da sich die Geschwindigkeit bei statischen Objekten umkehrt. Dieser Sachverhalt ist in Formel 2.4 auf Seite 10 beschrieben. Der neue Geschwindigkeitsvektor wird in der Methode *_calc_speed_vector* berechnet. Um eine Berührung der Kugel mit einem Objekte zu überprüfen, wurde die Methode *_shortest_distance* angelegt. Diese benötigt die Position der Kugel, das zu überprüfende Objekte und die Information, in welche Richtung das Objekt „geneigt“ ist, um die kürzeste Entfernung zu dem Objekt zu berechnen. Es wird nun mit einer definierten Schrittweite durch die Achse des Objekts iteriert. Nach jeder

```
def _init_table(self):
  box(pos=vector(), size=vector(), color=vector()) #table
  box(pos=vector(), size=vector(), color=vector()) #right boarder
  box(pos=vector(), size=vector(), color=vector()) #left boarder
  box(pos=vector(), size=vector(), color=vector()) #lower boarder
  box(pos=vector(), size=vector(), color=vector()) #upper boarder

  #Barrier
  self.barrier_down_right = box(pos=vector(), size=vector(vector())
  self.barrier_down_right.rotate(angle=self.angle_components, axis=vector
    (0,1,0))
  self.barrier_down_left = box(pos=vector(), size=vector(vector())
  self.barrier_down_left.rotate(angle=-self.angle_components, axis=vector
    (0,1,0))

  #Creating the Rackets
  self.racket_right = box(pos=vector(), size=vector(), color=vector())
  self.racket_right.rotate(angle=self.angle_components, axis=vector(0,1,0))
  self.racket_left = box(pos=vector(), size=vector(), color=vector())
  self.racket_left.rotate(angle=-self.angle_components, axis=vector(0,1,0))
```

Listing 5.1: Methode _init_render_objects

Iteration wird der Abstand zur Kugel mithilfe von *math.hypot* berechnet und der kleinste Abstandswert zurückgegeben. Abbildung 5.3 zeigt einer Darstellung dieser Methode. In Listing 5.4 ist die Methode dargestellt.
Bei der Kollision mit *bewegten Objekten* muss berücksichtigt werden, dass diese eine eigene Geschwindigkeit haben. Im Falle des Flippers muss zudem der Ort des Kontakts berücksichtigt werden, da die Geschwindigkeit über die Länge des Flippers variiert. Dafür wurde das Verhältnis zwischen Kontaktposition und gesamter Länge gebildet und mit einem Flipper-Geschwindigkeitswert multipliziert. Zudem muss darauf geachtet werden, dass der Flipper beim Zurückfahren auf seine Ausgangsposition einen umgekehrte Geschwindigkeit hat und in diesem Fall keinen Impuls auf die Kugel überträgt. Dies wurde mit einer Abfrage der aktuelle Bewegungsrichtung realisiert, da diese sich beim Zurückfahren umkehrt. Der resultierende Geschwindigkeitsvektor der Kugel wurde mit der gleichen Formel wie bei *unbewegten antiparallelen Objekten* berechnet (Formel 2.10 auf Seite 12).
Es hat sich durch Probleme und Glitches gezeigt, dass es sinnvoller ist die Geschwindigkeiten der Objekte gering zuhalten und dafür die Durchläufe pro Sekunde zu erhöhen. Dies hat den Vorteil, dass die Objekte mehr Positionen annehmen, was essentiell für die Kollisionserkennung ist. Würde man dies nicht tun, könnte es passieren, dass Objekte sich durch andere Objekte bewegen.

```
# Set VPython scene
self.scene = scene

#scene dimensions
self.scene.width = 600
self.scene.height = 600
self.scene.range = 3.5
self.scene.center = vector(0, 0, 0)
self.scene.camera.pos = vector(175, 400, 75)
self.scene.camera.axis = vector(0, -550, -350)

# Set scene properties
self.scene.title = 'Pinball gym (right-click to rotate, mouse wheel to zoom)'
self.scene.caption = "\n"
self.scene.autoscale = False
self.scene.background = PinballEnv.__colors['background']
```

Listing 5.2: Szenen Parameter

5.3 Programmierung

Nachfolgend wird die Umsetzung des Reinforcemente Agenten beschrieben. Der Agent wurde in der Klasse *Agent* untergebracht (siehe Abbildung 5.1).

5.3.1 Deep Reiforcement Learning

Für die Umsetzung des Deep Reiforcement Learning wird TensorFlow im Zusammenspiel mit Keras benutzt. Genauere Informationen zu der Deep-Learning-Bibliothek Keras oder dem Framework TensorFlow sind in Kapitel 2 zu finden.
Im ersten Schritt wurde ein Deep Learning Model angelegt. Dafür wurde eine Methode erstellt, die als Rückgabewert das fertige Model ausgibt. Diese Methode ist in Listing 5.5 zu sehen.
Der Methode *build_model* wird der Zustandsraum sowie die möglichen Aktionen übergeben. Es wird ein sequentielles Modell verwendet. Dies ermöglicht es ein neurales Netz zu definieren, welches sequentiell durch die angelegten Schichten läuft. Dem sequentiellen Modell werden zuerst die Eingangsdaten übermittelt, die durch *Flatten* in eine eindimensionale Form gebracht werden. Danach folgt eine *Dense*-Schicht, die vollständig mit der vorherigen Schicht verbunden ist. Die 32 steht für die Anzahl der Neuronen und „relu“ („Rectified Linear Unit“) steht für die Aktivierungsfunktion. Es wurde sich für 32 Neuronen entschieden, da bereits Modelle für OpenAI Gym Umgebungen mit diesem

```
1  if action == 0 and not self.movementLeftUpwards and not self.
       movementLeftDownwards and not self.movementRightUpwards and not self.
       movementRightDownwards:
2              self.moveLeft = False
3              self.moveRight = False
4  elif action == 1 and not self.movementLeftUpwards and not self.
       movementLeftDownwards and not self.movementRightUpwards and not self.
       movementRightDownwards:
5              self.moveLeft = True
6              self.moveRight = False
7  elif action == 2 and not self.movementLeftUpwards and not self.
       movementLeftDownwards and not self.movementRightUpwards and not self.
       movementRightDownwards:
8              self.moveLeft = False
9              self.moveRight = True
10 elif action == 3 and not self.movementLeftUpwards and not self.
       movementLeftDownwards and not self.movementRightUpwards and not self.
       movementRightDownwards:
11             self.moveLeft = True
12             self.moveRight = True
```

Listing 5.3: Auswertung racket_control

Wert initialisiert wurden. [51] Die Anzahl der Neuronen und die Anzahl der Schichten insgesamt sorgt dafür, dass das Netz komplexere Beziehungen aus Daten erlernen kann. Eine Erhöhung dieser Anzahl sorgt jedoch auch dafür, dass mehr Rechenkapazität bereit gestellt werden muss. Deswegen ist die Auslegung des neuronalen Netzes sehr auf die Komplexität der Aufgabe auszurichten. [34] Im Falle des Flippers wurde sich für vier Schichten entschieden, da dieser keine allzu komplexen Sachverhalte beinhaltet. Es ist jedoch lohnenswert die Schichten und die Anzahl der Neuronen in den Schichten zu variieren, um ein bestmöglich passendes neuronales Netz zu erzeugen, sowie Erfahrungen für die Auslegung zukünftiger Netze zu schaffen.
Als nächster Schritt wurde ein Agent angelegt. Dafür wurde eine Methode erstellt, die als Rückgabewert einen DQN-Agent ausgibt. Diese Methode ist in Listing 5.6 zu sehen. Als Policy wurde sich für die *BoltzmannQPolicy* entschieden, da diese Richtlinie für Q-Learning-basierte Systeme ausgelegt wurde und sich in Kapitel 4 für ein Deep Q-learning Algorithmus entschieden wurde. Danach wird ein Speicher festgelegt. Zum Schluss wird ein DQN-Agent angelegt, welcher die vorher definierte Richtlinie und Speicher anwendet. Zudem werden dem Agenten noch die möglichen Aktionen *nb_actions* und der Aktualisierungszeitpunkt *target_model_update* mitgeteilt. Der Aktualisierungszeitpunkt sagt aus, wie oft das Zielmodell mit dem Gewicht des aktuellen Modells aktualisiert wird. Es hat Auswirkungen darauf, wie effektiv und stabil das Training verläuft. Auch hier ist es loh-

```
def _shortest_distance(self, obs, left_or_right):
  obs_top = obs.pos + (obs.axis*left_or_right)/2
  obs_bot = obs.pos - (obs.axis*left_or_right)/2

  point = self.ball.pos
  ref_point = obs_bot
  ref_vec = obs.axis * left_or_right
  nearest_point = ref_point

  div = 0.05
  factor = 0.01
  small = 1000

  while factor <=1:
    a=ref_vec*factor
    point2=ref_point + a
    length=math.hypot(point.x-point2.x, point.z-point2.z)
    if length<small:
      small=length
      nearest_point = point2
    factor=factor+div

  return small, nearest_point
```

Listing 5.4: Methode _shortest_distance

nenswert diesen Wert zu variieren, um zu schauen, wann der Agent am Effektivsten ist. In der Methode *training* wird dem Agenten eine Anzahl von Schritten übermittelt, die er in einem Training absolvieren soll. Der Methode *show*, wird eine Anzahl von Episoden übermittelt, die anschließend vom Agenten demonstriert werden sollen.

5.3.2 Speicherung von Modellen

Um ein Modell nach dem Trainingslauf in einer Datei zu speichern, reicht folgender Befehl *dqn.save_weights*. Es wird dann eine Datei des Typs .h5f im Ordner des Python Projekts

```
def build_model(states, actions):
  model = Sequential()
  model.add(Flatten(input_shape=(1,states)))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(actions, activation='linear'))
  return model
```

Listing 5.5: Methode build_model

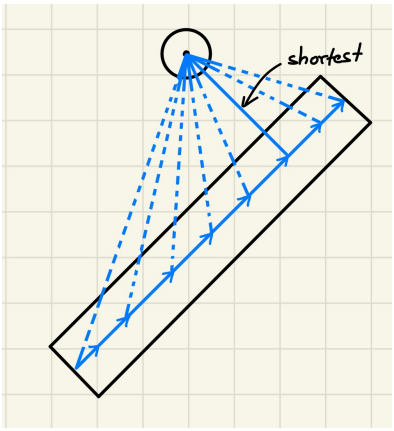


Abbildung 5.3: Methode shortestDistanc

```
def build_agent(model, actions):
  policy = BoltzmannQPolicy()
  memory = SequentialMemory(limit= 100, window_length=1)
  dqn = DQNAgent(model=model, memory= memory, policy=policy, nb_actions=
    actions, nb_steps_warmup=10, target_model_update=1e-2)
  return dqn
```

Listing 5.6: Methode build_agent

erstellt, beziehungsweise überschrieben, falls schon eine Datei vorhanden ist. Der Ausdruck ist Bestandteil der Methode *safe_model* in der Klasse *Agent* und ist in Listing 5.7 aufgeführt. Das Laden des Modells geschieht über den Befehl *dqn.load_weights*. Dieser ist Bestandteil der Methode *load_model* und ist in Listing 5.8 beschrieben.

```
def safe_model(self, fileName):
  self.dqn.save_weights(fileName, overwrite=True)
```

Listing 5.7: Methode safe_model

```
def load_model(self, fileName):
  self.dqn.load_weights(fileName)
```

Listing 5.8: Methode load_model

6 Entwicklung des prototypischen Aufbaus

In diesem Kapitel wird die Detaillösung, aus dem in Kapitel 4 beschriebenen Konzept, umgesetzt. Es wird zwischen der Hardware und der Software unterschieden.

6.1 Hardware

Dieser Abschnitt befasst sich mit dem Design der Hardwarekomponenten des Prototypen. Es werden das Spielfeld zusammen mit dem Rahmen, die 3D-gedruckten Elemente, die Schrittmotoren und die Kamerahalterung vorgestellt.

6.1.1 Spielfeld und Rahmen

Für das Spielfeld wird eine Aluverbundplatte benutzt, für die Konstruktion des Rahmens werden Aluprofile verwendet. Das Spielfeld hat die Maße von 50x34 cm, die Aluprofile sind quadratisch und haben eine Seitenlänge von 20 mm. Da die Aluprofile auf der Aluverbundplatte montiert werden, muss auf die Maße der Platte noch jeweils zwei mal die Breite der Aluprofile drauf gerechnet werden. Die Aluprofile werden mithilfe von 3D-gedruckten Klemmen an die Platte montiert, dabei werden die Klemmen per Nutstein und Schraube an dem Aluprofil befestigt (siehe Abbildung 6.1). Zudem werden in die Ecken rechtwinklige Winkel in die Nut eingelassen und mit Schrauben befestigt, dies sorgt für eine zusätzliche Stabilität des Rahmens. Abbildung 6.2 zeigt eine Draufsicht des Spielfelds mit Rahmen und den Maßen. Alle benötigten Komponenten samt Preis sind in Tabelle 6.1 aufgelistet. Die Preise können durch wirtschaftliche und politische Einflüsse schwanken und beziehen sich auf Februar 2024. Die Klemmen, welche den Rahmen und die Platte verbinden werden im Abschnitt 6.1.2 aufgelistet, da diese eine Teilmenge der 3D-gedruckten Teile sind.

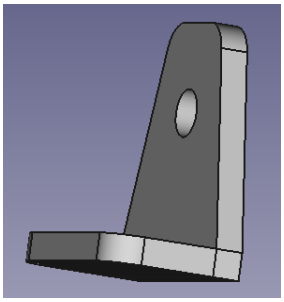


Abbildung 6.1: Klemme für Rahmen und Platte

Tabelle 6.1: Kosten für Spielfeld und Rahmen [2, 1, 17, 19]

	Preis pro Stück	Menge	Preis insgesamt
Aluverbundplatte	15,04 €	1	15,04 €
Rahmen	18,32 € pro 190 cm	184 cm	17,74 €
Nutsteine M4	0,16 €	8	1,28 €
Innenwinkel	1,70 €	4	6,80 €
Total			40,68 €

6.1.2 3D-gedruckte Elemente

In diesem Abschnitt wird kurz die Erarbeitung der 3D-gedruckten Objekte beschrieben. Zu diesen Teilen zählen unter anderem die Halterungen für die Motoren, die Elemente auf dem Spielfeld, sowie Stützen/Füße für den gesamten Aufbau. Eine Liste mit den gesamten Teilen ist in Tabelle 6.2 zu sehen. Da der Zugang zu einem 3D-Drucker schwierig war, hat sich Herr Prof. Dr. Hensel bereit erklärt diese Arbeit zu übernehmen. In Abbildung 6.3a und 6.3b sind zwei Bilder der Banden Elemente gezeigt. Der resultierende Aufbau des Spielfelds mit 3D-gedruckten Elementen ist in Abbildung 6.4 zu sehen. Die Dateien zu den 3D-gedruckten Teilen können im GitHub-Repository [1] von Herrn Prof. Dr. Hensel eingesehen werden.

6.1.3 Schrittmotor

Zum Bewegen der Flipper werden NEMA 17 Schrittmotoren verwendet, das Modell heißt 17HS19-2004S1. Die Signale kommen dabei vom Arduino Uno.

[1] https://github.com/MarcOnTheMoon/ai_pinball

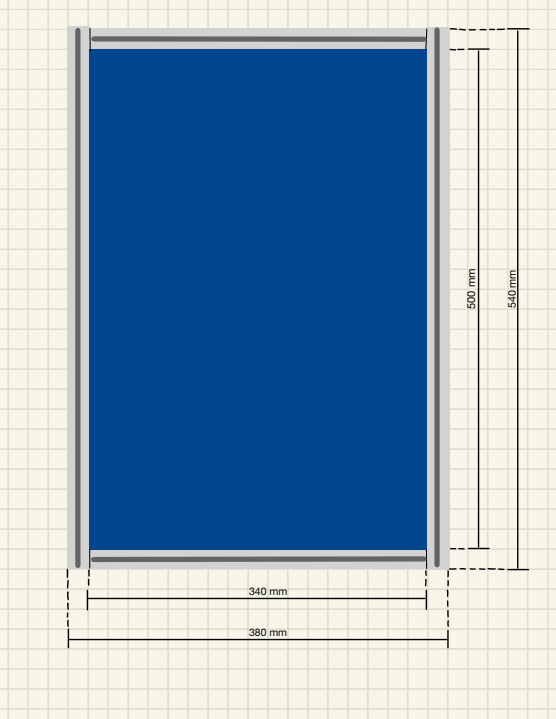
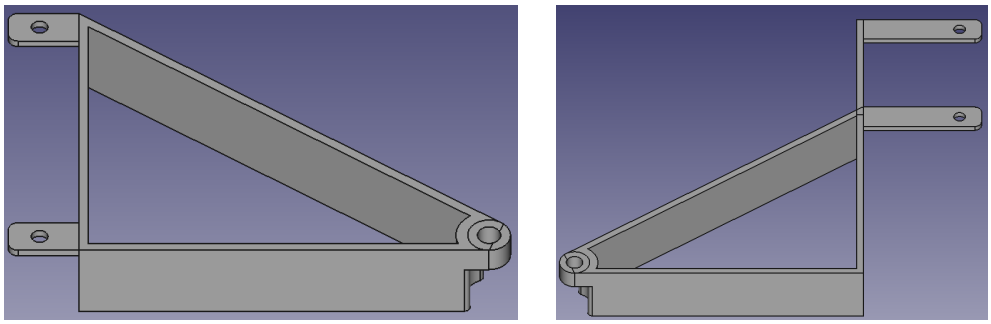


Abbildung 6.2: Maße des Spielfelds mit Rahmen

Zu Beginn wurde versucht die Flipper mit dem Schrittmotormodell ACT 16HS2404LP1X zu realisieren. Nach kurzer Testphase fiel jedoch auf, dass das Drehmoment des Modells zu gering war. Durch akustische Analyse wurde die maximale Geschwindigkeit der Kugel ermittelt. Dafür wurde die Kugel gegen den Rahmen am oberen Ende der Spielfelds gerollt, die Kugel rollte dann gegen die Flipper. Die Zeit zwischen den Aufprallgeräuschen



(a) Bande links (b) Bande rechts

Abbildung 6.3: 3D-gedruckte Elemente

Tabelle 6.2: Liste 3D-gedruckter Teile

	Menge
Flipper	2
Bande links	1
Bande rechts	1
Halterung für Motor-Strebe	2
Motorhalterung	3
Kugelrückführungsschläger	1
Klemmen für Rahmen und Platte	8
Füße vorne	2
Füße hinten	2
Starttunnel	1
Rückführungsrampe (2x 110 mm, 1x 90 mm)	1
Befestigung Rückführungsrampe	1
Kugelführung über Starttunnel	1

wurde gemessen ebenso wie der zurückgelegte Weg. Abbildung 6.5a und Abbildung 6.5b zeigen das obere und das untere Aufprallgeräusch.
Die berechnete Zeit beträgt 0,8 s, die zurückgelegte Strecke ca. 37 cm. Bei einer Neigung von 7° resultiert eine Beschleunigung von ca. 1,2 m/s^2. Die Geschwindigkeit vor dem Aufprall lässt sich nach Formel 6.1 berechnen. Wobei a die Beschleunigung und s die Strecke ist.

$$v = \sqrt{2as} \tag{6.1}$$

Nach Einsetzten der bekannten Werte erhält man:

$$v = \sqrt{2 \cdot 1,2\ m/s^2 \cdot 0,37\ m} = 0,942\ m/s$$

Es wird nun geschaut, ob ein NEMA17 Schrittmotor mit der Modellnummer 17HS19-2004S2 ausreicht. Dafür wird die Geschwindigkeit des Flippers berechnet. Um das maximale Drehmoment zu erhalten wird aus der Drehmoment-Kennlinie 6.6 die Drehzahl für das maximale Drehmoment ermittelt. Es wird eine Drehzahl von 300 Umdrehungen pro Minute ausgewählt, da das Drehmoment nach dieser kontinuierlich abfällt. Für eine Umdrehung benötigt der Motor also 200 ms. Die Geschwindigkeit am äußersten Ende des Flippers lässt sich nach Formel 6.2 berechnen. Dabei ist v die Geschwindigkeit, r die

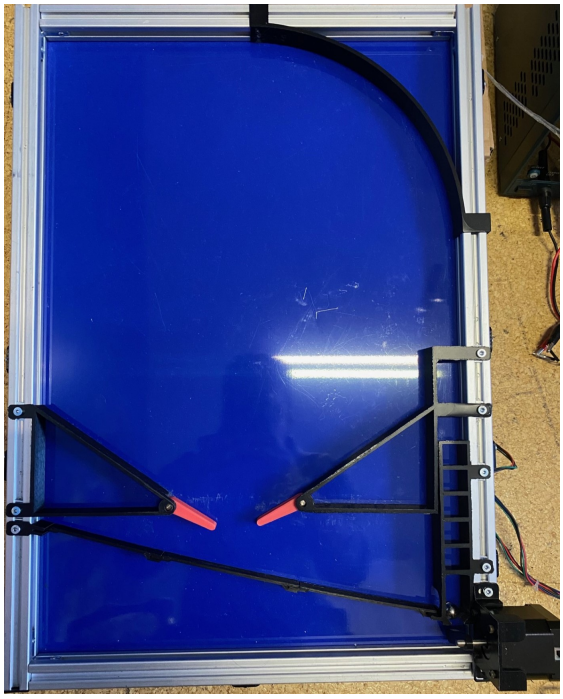


Abbildung 6.4: Realer Aufbau

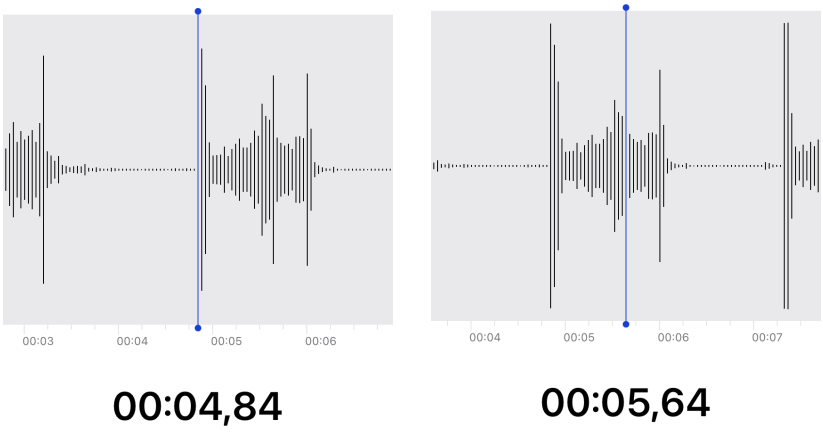
Länge des Flippers und t die Zeit die der Flipper für eine Umdrehung benötigt.

$$v = \frac{2\pi \cdot r}{t} \tag{6.2}$$

Nach Einsetzen der bekannten Werte erhält man:

$$v = \frac{2\pi \cdot 0,04\ m}{0,2\ s} = 1,257\ m/s$$

Wenn die Kugel nun auf den ausschlagenden Flipper trifft ergibt sich eine Geschwindigkeitsdifferenz von ca. 2,198 m/s. Um die Kraft zu berechnen wird Formel 6.3 verwendet. Dabei ist F die Kraft, m die Masse, v die Geschwindigkeit vor beziehungsweise nach dem Aufprall und s die Strecke. Die Geschwindigkeit vor dem Aufprall und nach dem Aufprall sind betragsmäßig gleich groß jedoch entgegensetzt (Verluste werden vernachlässigt). Die Kontaktzeit wird mit 5 ms angenommen, dieser Wert ergibt sich aus Messungen mit einer



(a) Aufprall oben (b) Aufprall unten

Abbildung 6.5: Audiodatei der Aufpralle

Kamera und einer Kraftmessplatte. [53, 64]

$$F = m \cdot \frac{v_{vor} - v_{nach}}{t} \tag{6.3}$$

Mit $v_{nach} = -v_{vor}$ ergibt sich:

$$F = m \cdot \frac{2v_{vor}}{t} \tag{6.4}$$

Nach Einsetzen der bekannten Werte erhält man:

$$F = 0,01\ kg \cdot \frac{2 \cdot (2,198\ m/s)}{0.005\ m/s} = 8,792\ N$$

Die Formel für das Drehmoment ist in Formel 6.5 dargestellt. Dabei ist M das Drehmoment und l die Länge des Hebelarms.

$$M = F \cdot l \tag{6.5}$$

Nach Einsetzen der bekannten Werte erhält man:

$$M = 8,792\ N \cdot 4\ cm = 35,168\ Ncm$$

Es wird ein Drehmoment von ca. 35,2 Ncm benötigt. Nach erneuter Analyse der Drehmoment-Kennlinie 6.6 ist zu erkennen, dass dieser Wert mit dem Schrittmotor 17HS19-2004S2

zu erreichen ist.
Zur Ansteuerung der Schrittmotoren wurde der Motortreiber TB6600 verwendet. Dieser ist für Anwendungen mit Mikroprozessoren ausgelegt und kann durch eine externe Spannungsversorgung hohe Leistungen bereitstellen. An dem TB6600 ist lediglich der Motornennstrom und die Schritte pro Umdrehung per DIP-Schalter einzustellen. Den Treibern wird eine Spannung von 24 V bereitgestellt. Bei dem TB6600 handelt es sich um einen Strom-regelnden Motortreiber, für weiter Informationen siehe Abschnitt 2.2. Der Anschlussplan der Schrittmotoren ist in Abbildung 6.7 zu sehen.
Für die Befestigung der Schrittmotoren, welche für die Drehung der Flipper verantwortlich sind, musste eine Aluprofil-Querstrebe am Rahmen befestigt werden. Für die Befestigung am Rahmen wurden 3D-gedruckte Verbindungsstücke konstruiert (siehe Abbildung 6.8a). Die Halterung der Motoren an den Aluprofilen wurde ebenfalls mit 3D-gedruckten Elementen realisiert (siehe Abbildung 6.8b). Alle verwendeten Komponenten samt Preis sind in Tabelle 6.3 dargestellt. Die reale Querstrebe samt Halterung und Schrittmotoren ist in Abbildung 6.9 beschrieben.

6.1.4 Kamerasystem

Da die Kugelposition mithilfe eines Bildverarbeitungssystem erfasst werden soll, wird eine Kamera sowie eine Halterung für die Kamera benötigt. Als Kamera wird die Jelly Comb W06 verwendet, da diese bei Herr Prof. Dr. Hensel vorrätig war.
Für die Kamerahalterung wird ein dreibeiniges Kamerastativ benutzt, auf dem die Kamera montiert wird. Verwendet wird das Kamerastativ Alpha 1000 von Cullmann. Bei der Entscheidung für dieses Stativ, war ebenso wie bei der Kamera die Vorrätigkeit bei Herrn Prof. Dr. Hensel ausschlaggebend. In Tabelle 6.4 sind diese beiden Komponenten samt Preis aufgelistet.

Tabelle 6.3: Komponenten Schrittmotorsteuerung [29, 22, 1, 6]

	Preis	Menge	Preis insgesamt
Schrittmotoren NEMA 17	10,20 €	3	30,60 €
Motor Treiber TB6600	22,90 €	3	68,70 €
Motorquerstrebe	18,32 € pro 190 cm	38 cm	3,67 €
Arduino Uno	18,70 €	1	18,70 €
Spannungsversorgung	1	1	1
Total			121,67 €

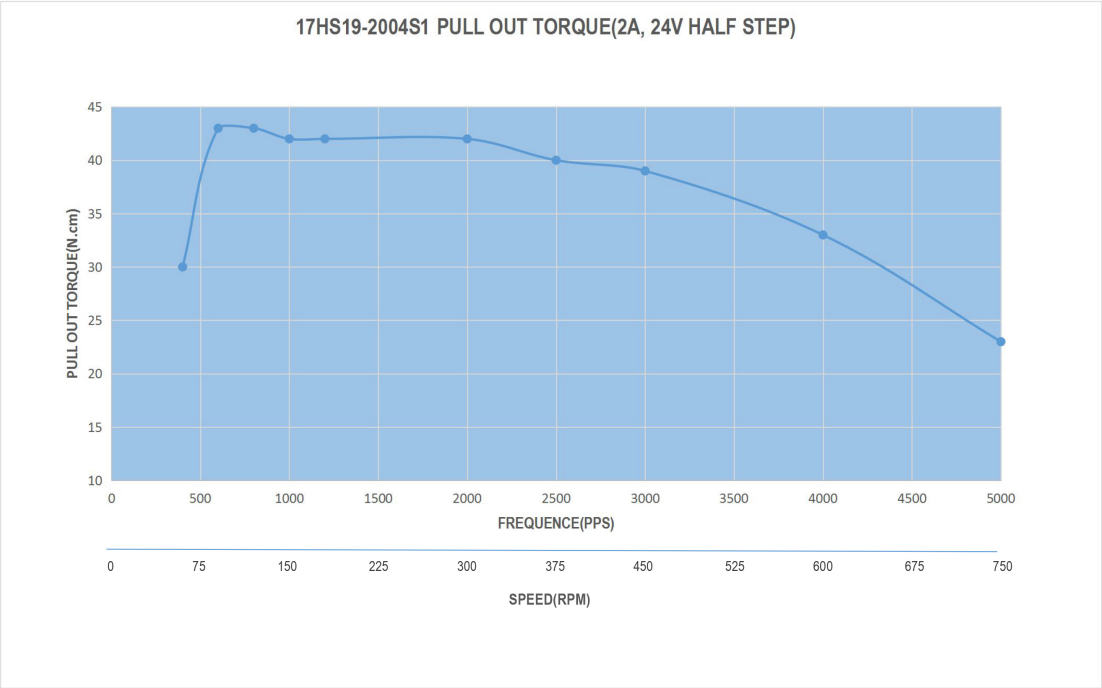


Abbildung 6.6: Drehmomentkennlinie NEAM17 [29]

Tabelle 6.4: Komponenten Kamerasystem [21, 10]

	Preis
Kamera Jelly Comb W06	29,75€
Kamerastativ Cullmann Alpha 1000	14,06€
Total	43,81€

6.2 Steuerung

Die Steuerung für den prototypischen Aufbau ist in drei Teilsysteme gegliedert. Die Hauptsteuerung läuft über ein Python Programm auf einem Rechner, für die Positionserfassung wird eine Kamera per USB an den Rechner angeschlossen. Bei der Ansteuerung der Schrittmotoren dient ein Arduino Uno, der per USB angeschlossen ist, als Schnittstelle, über die, die Schrittmotoren ihre Signale bekommen. Dieser Abschnitt ist in eben genau diese drei Teilsysteme gegliedert.

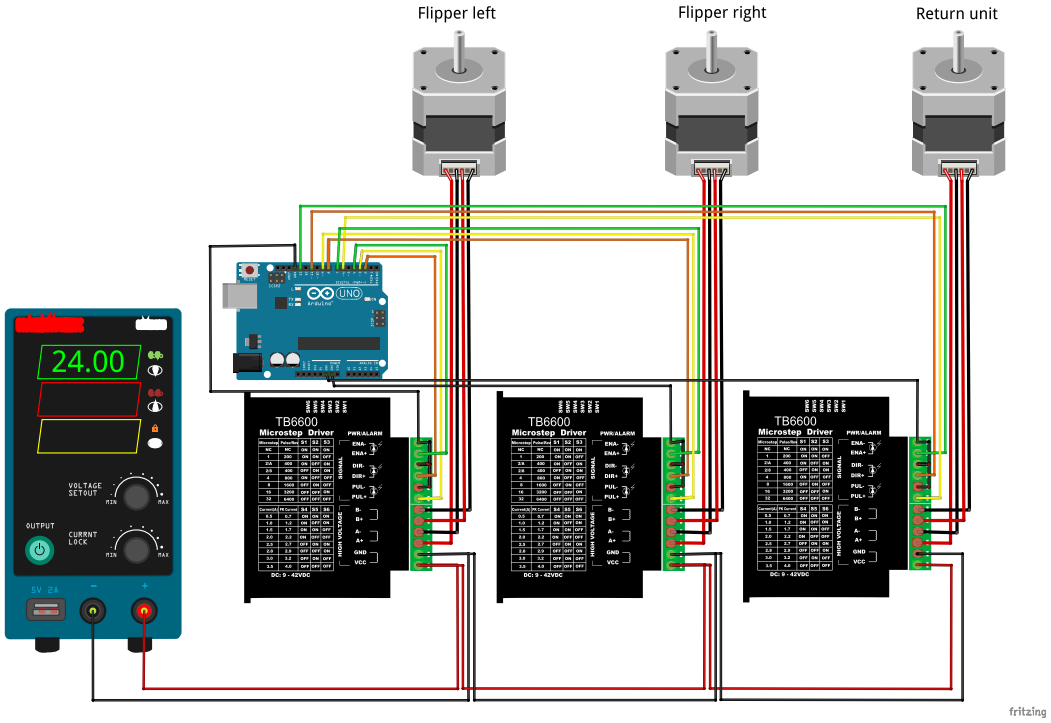
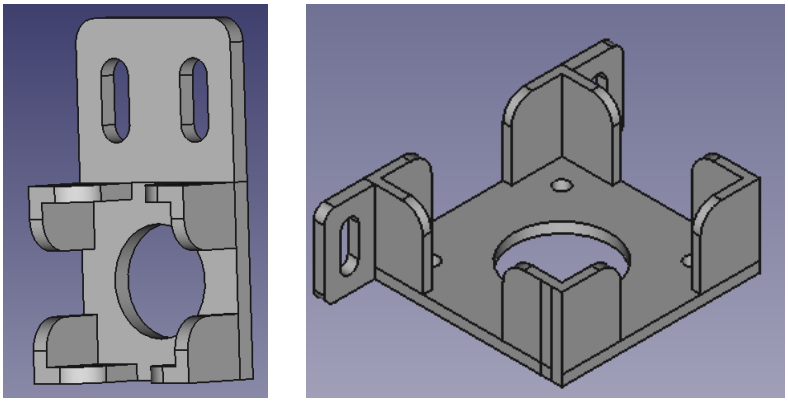


Abbildung 6.7: Anschlussplan für die Schrittmotoren

6.2.1 Zentralsteuerung

Ein Teil der Zentralsteuerung basiert auf der Steuerung der OpenAI-Umgebung. Es wird die Kompatibilität zwischen Reinforcement Agent und der Umgebung von OpenAI Gym ausgenutzt. In den benötigten Methoden werden dann nicht mehr die virtuellen Aktoren angesteuert, sondern die Aktoren des prototypischen Aufbaus. In Abbildung 6.10 ist ein Aktivitätsdiagramm dargestellt. In Abbildung 6.11 ist das Klassendiagramm für die Steuerung dargestellt. Wie man sieht, wird aus der Klasse *Env* die Bildverarbeitung und die Kommunikation mit dem Arduino gestartet. Dies ist auch in Listing 6.1 zu sehen. Man sieht, wie die Verbindung mit Arduino und Kamera aufgebaut wurde. Es ist wichtig, dem Kamerasystem Zeit zu geben, bevor das Programm weiterläuft, da das Programm sonst abstürzt, dies ist mit *sleep(15)* realisiert. Da nach einer abgeschlossenen Drehung des Schrittmotores eine Verifizierung vom Arduino zum Rechner übermittelt wird, würde das Steuerungsprogramm warten, bis die Verifizierung eingetroffen ist. Aus diesem Grund wird die Ansteuerung und das Verifizieren in einen Thread ausgelagert. Solange auf eine Verifizierung gewartet wird kann auch keine weitere Ansteuerung, vorgenommen werden.



(a) Verbindungsstück Rahmen-Strebe

(b) Motorhalterung

Abbildung 6.8: 3D-gedruckte Halterungen

```
self._arduino = ArduinoCOM(serialCOM=6)
reply = self._arduino.readLine()
print('Device ready: ' + reply)

self.positionX = 0
self.positionZ = 0

#connect to camera
self.cameraThread = threading.Thread(target=self.thread_function_camera)
self.cameraThread.start()

#wait until the camera system is set up
sleep(15)
```

Listing 6.1: Verbindungsaufbau mit Arduino und Kamera

Die Ansteuerung des linken Flippers ist in Listing 6.2 zu sehen. Es ist zu erkennen, dass die Variable *self.hold_left* während des Prozesse auf eins gesetzt wird. Dies signalisiert der Steuerung, dass momentan eine Bewegung ausgeführt wird.

```
def thread_function_left(self):
  self.hold_left = 1
  self._arduino.writeString('L>')
  reply = self._arduino.readLine()
  self.hold_left = 0
```

Listing 6.2: Ansteuerung linker Flipper

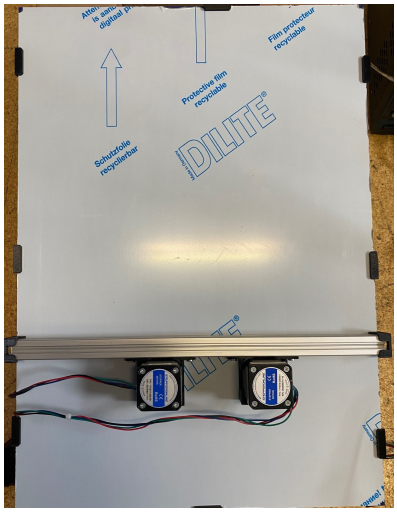


Abbildung 6.9: Montierte Motorquerstrebe

6.2.2 Bildverarbeitung

Die Position der Kugel, die in der virtuellen Umgebung ein Attribut des Objekts war, wird nun durch die Bildauswertung der Kamera geliefert. Dafür wird das Kamerabild durch ein Python Programm ausgewertet und liefert die Position der Kugel an die Zentralsteuerung. Der Programmaufbau für diese Aufgabe ist ebenfalls in Abbildung 6.11 als Klasse *App* zu sehen. Die Auswertung des Bildes wurde von Herrn Prof. Dr. Hensel durchgeführt und zur Verfügung gestellt. Das Programm für die Bildauswertung läuft in einem separaten Thread, um eine kontinuierliche Positionserfassung zu gewährleisten. Es wurden vorerst 10 FPS ausgewertet. Bei zukünftigen Arbeiten müsste dieser Wert eventuell angepasst werden, um schnelle Bewegungen der Kugel nachverfolgen zu können.

6.2.3 Ansteuerung Schrittmotoren

Die Ansteuerung der Schrittmotoren wird über einen Arduino Uno im Zusammenspiel mit dem TB6600 Motortreibern durchgeführt. Der Arduino bekommt dabei Befehle von der Zentralsteuerung und steuert die notwendigen Pins an. Nach Abschluss einer Bewegung wird ein *Ok*-Signal zur Zentralsteuerung übermittelt. Die Kommunikation zwischen Arduino und Rechner findet über die serielle Schnittstelle statt, die der Arduino auch bei Datenübermittlungen aus dem seriellen Monitor der Ardunio IDE nutzt. Die Verarbeitung, der vom Rechner übermittelten Werte, wird in der Methode

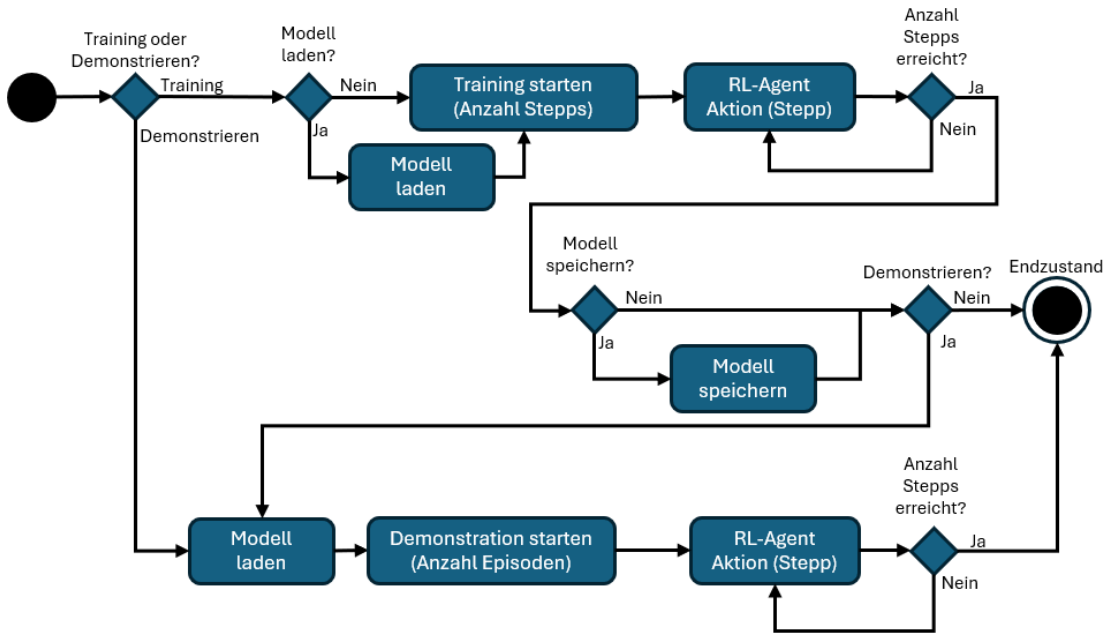


Abbildung 6.10: Aktivitätsdiagramm

processReceivedCommands umgesetzt und ist in Listing 6.3 zu sehen. Bei der Umsetzung der Schrittmotorsteuerung wurden bereits bestehende Bibliotheken genutzt und den Bedürfnissen dieses Systems angepasst. Die Programmstruktur ist in Abbildung 6.12 zu sehen. Bei der Schrittmotorsteuerung ist es wichtig, dass es eine Beschleunigungsphase am Anfang und eine Bremsphase am Ende einer Bewegung gib. Ohne diese Vorkehrung treten schnell Schrittverluste bei hohen Drehzahlen auf. Dies wurde in der Methode *action* umgesetzt. Die Überprüfung in welcher Phase man sich gerade befindet ist in Listing 6.4 zu sehen.

```
if (receivedCount > 0) {
   for (int i = 0; i < receivedCount; i++) {
     switch (receivedData[i]) {
       case 'L':                    // Stepper motor left
         moveSteps(1);
         break;
       case 'R':                    // Stepper motor right;
         moveSteps(2);
         break;
       case 'B':                    // Both Stepper motors;
         moveSteps(3);
         break;
       case 'S':                    // Return Unit;
         moveSteps(4);
         break;
       case 'N':                    // normal setting;
         moveSteps(0);
         break;
       case '>':                    // Send acqknowledge
         Serial.println("ok");
         break;
     }
   }
  }
}
```

Listing 6.3: Methode processReceivedCommands

```
if (stepsstepped < maxAccTime) //Accerlations phase
  {
    acAcTime = min(acAcTime + 1, maxAccTime);
    acF = (acAcTime / maxAccTime) * (maxStepsps - minStepspsStart) +
    minStepspsStart;
  }
else if (abs(steps) - stepsstepped < maxBreakTime) //Breaking phase
  {
    acAcTime = min(max(acAcTime - 1, 0), maxBreakTime);
    acF = (acAcTime / maxBreakTime) * (maxStepsps - minStepspsEnd) +
    minStepspsEnd;
  }
else //Max Speed
  {
     acF = maxStepsps;
  }
```

Listing 6.4: Geschwindigkeitsphasen

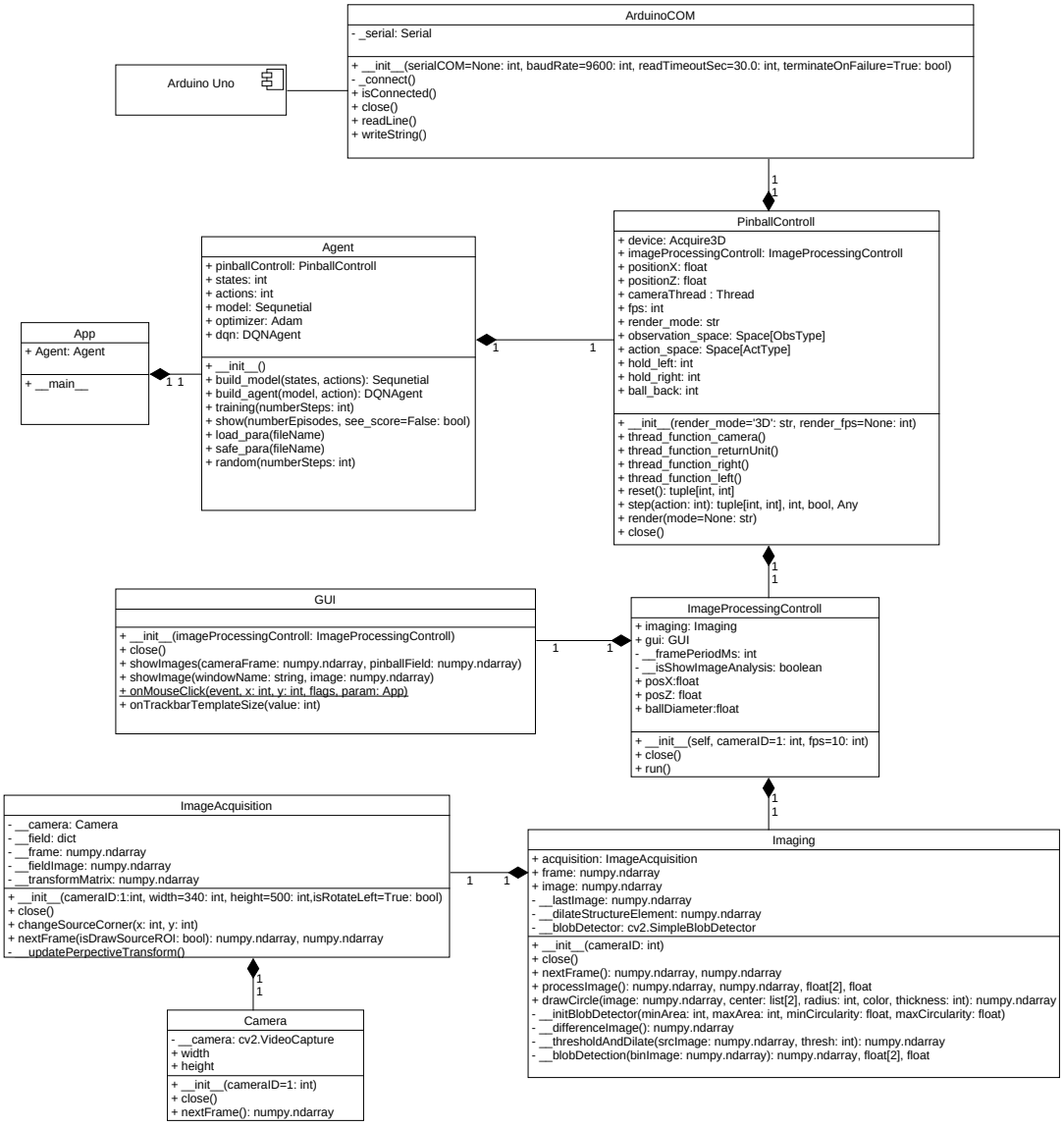


Abbildung 6.11: Klassenübersicht Steuerung

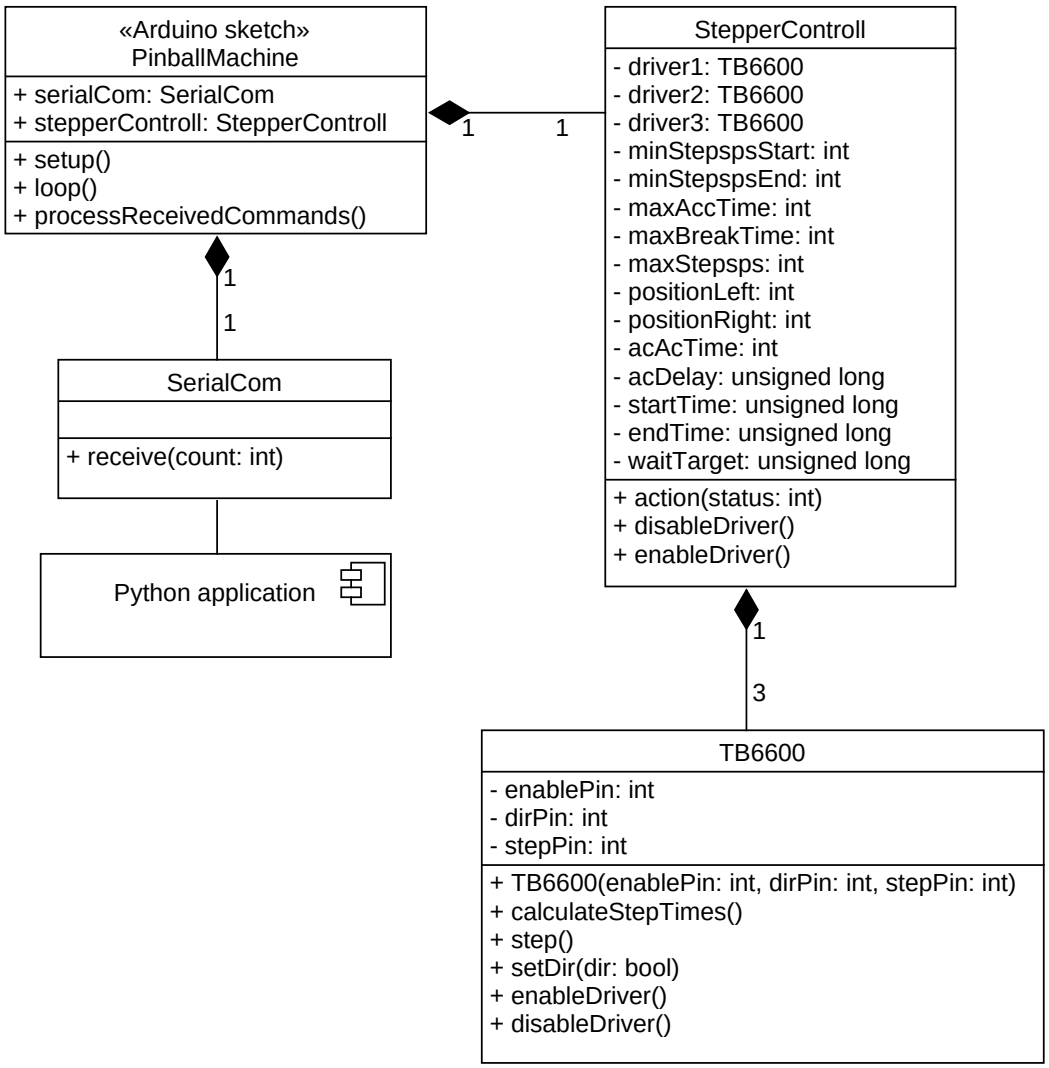


Abbildung 6.12: Klassenübersicht Arduino-Steuerung

7 Evaluation

In diesem Abschnitt geht es um die Auswertung der Anforderung aus Kapitel 3, diese werden Punkt für Punkt überprüft. Dabei wird zwischen virtueller Umgebung und prototypischen Demonstrator unterteilt. Bei komplexeren Anforderungen wird ein geeignetes Testverfahren verwendet, um eine sinnvolle Bewertung vornehmen zu können.

7.1 Virtuelle Umgebung

In diesem Abschnitt wird die virtuelle Umgebung evaluiert. Zuerst werden die gestellten Anforderungen geprüft, anschließend werden einzelne Anfordern genauer betrachtet und getestet und abschließend wird eine Bewertung vorgenommen.

7.1.1 Anforderungsprüfung

Dieser Abschnitt befasst sich mit den Anforderungen aus Abschnitt 3.2 und prüft inwiefern diese *erfüllt*, *teilweise erfüllt* oder *nicht erfüllt* wurden.

VU-F1: *Erfüllt:* Durch die Verwendung von Keras, lassen sich einfach Dateien im HDF5-Format generieren und abspeichern.

VU-F2: *Erfüllt:* Es gibt einen Trainings-Modus. Diesem kann man eine gewünschte Anzahl von Durchläufen übergeben, die dann abgearbeitet werden.

VU-F3: *Erfüllt:* Es gibt die Möglichkeit erlernte Fähigkeiten zu demonstrieren. Genau wie bei VU-F2 kann die Anzahl der gewünschten Durchläufe übergeben werden.

VU-F4: *Teilweise erfüllt:* Die Kollisionen sind nach den physikalischen Gesetzen umgesetzt, jedoch wurde für Reibungs- und Aufprallverluste durch federnde Bauteile ein konstanter Wert angenommen.

VU-F5: *Erfüllt:* Die Kugel taucht nach Endzustand am oberen Spielfeldrand auf. Sie wird zufällig entlang der kompletten Breite platziert, jedoch nicht direkt über der Lücke zwischen den Flippern.

VU-NF1: *Erfüllt:* Die virtuelle Umgebung ist dreidimensional dargestellt.

VU-NF2: *Erfüllt:* Es wurde darauf geachtet, passende Klassen zu erstellen und eindeutige Variablennamen zu benutzen.

VU-NF3: *Teilweise erfüllt:* Die Breite wurden um 2,5 cm verringert, da dieser Platz beim prototypischen Demonstrator für die Startvorrichtung verwendet wird, welcher keine Verwendung in der virtuellen Umgebung findet.

VU-NF4: *Erfüllt:* Die Flipper wurden trichterförmig mit einem Winkel von 35° zur Waagerechten auf dem Spielfeld platziert.

VU-NF5: *Erfüllt:* Die Bewegung der Flipper deckt einen Bereich von 70° ab.

VU-NF6: *Erfüllt:* Für die Konfigurierbarkeit wurden Variablen definiert, welche durch den Benutzer leicht konfigurierbar sind.

7.1.2 Funktionstest

Ein wichtiges Kriterium für die Beurteilung des Gesamtziels ist die Anzahl der Schritte pro Episode. Je mehr Schritte bewältigt wurden, desto länger war der Reinforcement Agent in der Lage die Kugel im Spiel zu halten. Aus diesem Grund wird die Anzahl der Schritte pro Episode über 100 Episoden analysiert. Als Referenzwert wird eine Methode programmiert, die zufällig ausgewählte Aktionen in der virtuellen Umgebung ausführt. Der Zeittakt der Aktionen von der Zufallsmethode und dem Reinforcement Learning wurden identisch gewählt, um eine gemeinsame Bezugsgröße zu erhalten. Diese betrug ca. 20 ms, da dies auch die Aktualisierungsrate der Umgebung ist.
In den ersten Versuchen wurde dem Agenten eine Belohnung übermittelt, wenn er die Kugel solange wie möglich im Spiel hält. Bei dieser Variante entwickelte sich schnell die Fähigkeit des Agenten die Flipper dauerhaft in der oberen Endlage zu halten. Dadurch wurde die Kugel in der Vertiefung zwischen Flipper und Bande gehalten und das vorgegebene Ziel erfüllt.
Um das Spiel dynamischer zu gestalten, wurde dieses Verhalten in den nachfolgenden Tests bestraft. Jedoch wurde erst bestraft, wenn die Flipper eine gewisse Zeit in der oberen Endlage waren, um den Agenten die Möglichkeit zu geben, diese Eigenschaft auszunutzen. Die Zeit belief sich auf 20 Durchläufe, was bei 50 FPS eine Zeit von ca. 400 ms entspricht. Die Bestrafung für dieses Verhalten wurde mit -5 bestraft, dass durch-

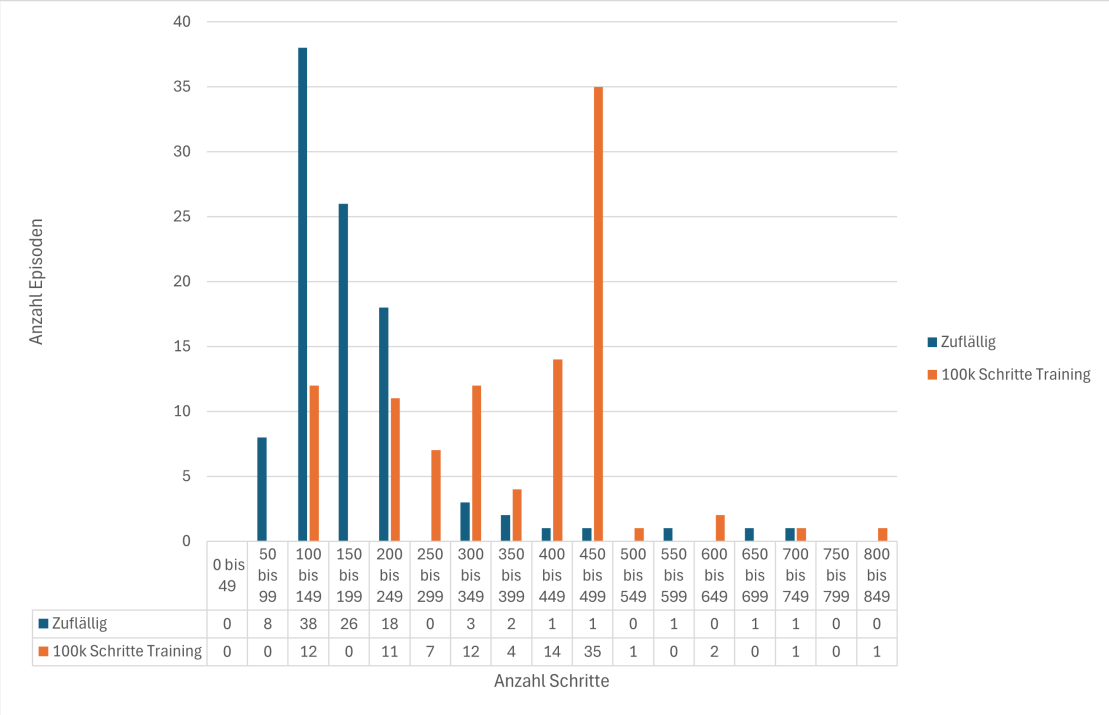
fallen der Kugel mit -100. Für das im Spiel halten wurde pro Aktion eine Belohnung von 1 übergeben. Ein Vergleich der Fähigkeiten ist in Abbildung 7.1 zu sehen. Die orangen Balken zeigen die Anzahl der Schritte pro Episode des trainierten Agenten, in blau ist die Zufallsmethode zu sehen. Es ist jedoch zu erwähnen, dass der Stand, der in der Abbildung dargestellt ist, leicht anders aufgebaut war, dazu zählt das Belohnungssystem, sowie leicht abweichende Kollisionen. Dies ändert aber nichts an der Aussagefähigkeit der Abbildung. Die Mittelwerte und der Median des Funktionstest sind in Tabelle 7.2 zu sehen. Es ist zu erkennen, dass die durchschnittlichen Schritte bei Verwendung eines Reinforcement Agenten ca. doppelt so groß sind, wie bei zufälligen Aktionen. Der Median, bei dem extreme Abweichungen herausgefiltert werden, ist mehr als doppelt so groß. Es ist jedoch auch zu erwähnen, dass es Optimierungspotential gibt, vor allem bei der Vergabe der Belohnung, da man durch sinnvoll platzierte Belohnungen den Agenten in eine gewünschte Richtung „lenken“ kann. Zudem gibt es bei der Generierung eines Reinforcement Agenten diverse Stellschrauben, durch die dieser angepasst werden kann.
Diese Arbeit zeigt vor allem die Machbarkeit, ein hoch dynamisches System durch einen Reinforcement Agenten zu steuern. Weitere Optimierungen können Bestandteil aufbauender Arbeiten sein.

7.1.3 Kollisionstest

Eine korrekte Abbildung der Kollisionen in der virtuellen Umgebung ist wichtig für die Vergleichbarkeit und auch für die Parameterübertragbarkeit zwischen virtueller und realer Umgebung. Um die Vergleichbarkeit zu testen wird in der Simulation die Kugel auf die Bande zurollen gelassen, genauso wie beim realen Aufbau. Von diesem Prozess wird ein Video aufgenommen. Durch die Analyse des zurückgelegten Weges können dann Rückschlüsse über die Vergleichbarkeit gemacht werden. Dabei ist die Startposition der Kugel 7 cm vom linken Rahmen gelegen. In Abbildung 7.2a ist der zurückgelegte Weg der virtuellen Kollision und in Abbildung 7.2b der zurückgelegte Weg der realen Kollision als rot gestrichelte Linie zu sehen. Es kann festgestellt werden, dass die zurückgelegten

Tabelle 7.2: Mittelwert und Median des Funktiontests

	Mittelwert	Median
Zufällige Aktionen	183	154
Aktionen durch Reinforcement Agenten	367	413



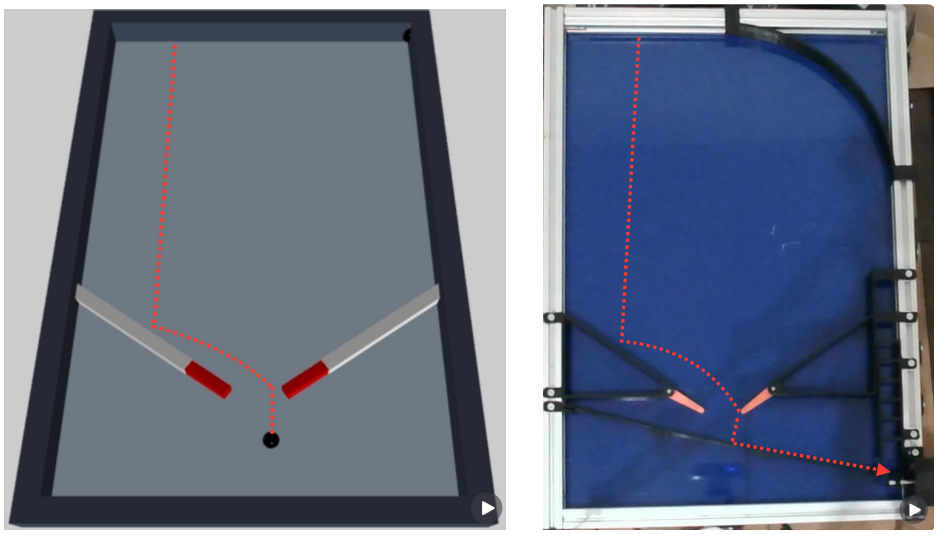
	0 bis 49	50 bis 99	100 bis 149	150 bis 199	200 bis 249	250 bis 299	300 bis 349	350 bis 399	400 bis 449	450 bis 499	500 bis 549	550 bis 599	600 bis 649	650 bis 699	700 bis 749	750 bis 799	800 bis 849
Zuflällig	0	8	38	26	18	0	3	2	1	1	0	1	0	1	1	0	0
100k Schritte Training	0	0	12	0	11	7	12	4	14	35	1	0	2	0	1	0	1

Abbildung 7.1: Fähigkeiten im Vergleich

Wege nahezu identisch sind. Es ist jedoch zu berücksichtigen, dass vor allem beim realen Aufbau eine leichte Verzerrung des Bildes vorliegt, da die Kamera nicht zentral über dem Spielfeld platziert werden konnte. Zudem ist eine leichte Schieflage des Spielfeldes zuerkennen, diese resultiert aus der Messungenauigkeit beim Einstellen der Spielfeldneigung.

In Abbildung 7.3 wurden die Linien übereinandergelegt, die blau gestrichelte Linie ist dabei der zurückgelegte Weg in der virtuellen Umgebung, die rot gestrichelte Linie ist der reale zurückgelegte Weg. Es musste darauf geachtet, dass die Bilder und die Spielfelder in den Bildern die gleichen Proportionen haben, damit bei der Überlagerung der Linien keine Fehler auftreten. Es ist deutlich zu erkennen, das die Linien fast identisch sind. Der etwas bauchigere Linienverlauf nach dem Kontakt mit der Bande bei der realen Linie, lässt sich auf die leichte Federwirkung der Bauteile zurückführen, die es bei virtuellen Bauteilen nicht gibt.

Es ist jedoch auch zu erwähnen, dass gewisse Bereiche in der virtuellen Umgebung immer noch zu Fehlern und Problemen führen können, zum Beispiel die Bereiche zwischen Bande und Flipper. Dort kann es zu einer falschen Berechnung des neuen Geschwin-



(a) Virtuelle Kollision (b) Reale Kollision

Abbildung 7.2: Vergleich Kollisionen

digkeitsvektors kommen, da die Kollisionsbereiche sich überlappen. Außerdem wird das „rollen“ direkt auf der Bande und den Flippern noch nicht ganz sauber abgebildet.

7.1.4 Zeittest

Ebenso wie die Kollision dazu beiträgt die virtuelle Umgebung realitätsnah abzubilden, ist ein weiterer wichtiger Faktor die Zeit. Es sollte die gleiche Zeit in der virtuellen Umgebung vergehen wie in der Realität. Um die Zeit für den real zurückgelegten Weg zu stoppen wurde auf die Videoaufnahmen aus Abschnitt 7.1.2 zurückgegriffen. Die Zeitauswertung in der virtuellen Umgebung wurde programmiertechnisch umgesetzt. Es wurde geschaut, wie lange der zurückgelegte Weg jeweils dauert. Die Auswertung der Zeiten ist in Tabelle 7.3 zu sehen. Die Abweichung zwischen realer und virtueller Zeit liegt bei 0,02 s. Durch diesen Wert kann also angenommen werden, dass die Physik

Tabelle 7.3: Zeit real vs. virtuell

	Zeit in Sekunden
Real	1,4
Virtuell	1,38

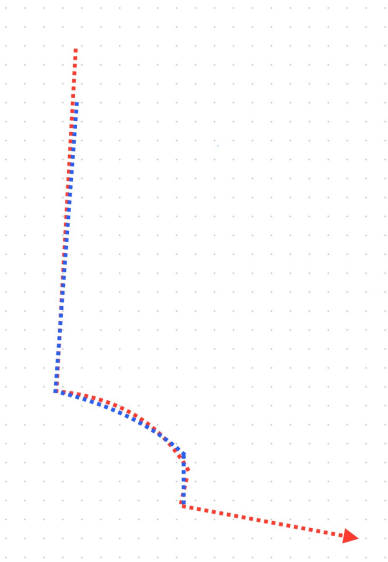


Abbildung 7.3: Auswertung Kollision (blau: Virtuell, rot: Real)

der virtuellen Umgebung, der der Realität nahekommt und das Ziel eine detailgetreue Umgebung zu schaffen als erfolgreich anzusehen ist.

7.1.5 Bewertung

Zusammenfassend kann man sagen, dass die virtuelle Umgebung in großen Teilen abgeschlossen ist. Die Physik wird realistisch umgesetzt und bildet eine solide Grundlage. Jedoch müsste an machen Stellen Debugging betrieben werden, wie zum Beispiel bei überlappenden Kollisionsbereichen und langsamen Bewegungen auf der Bande und den Flippern. Außerdem stimmt die Zeit, die ein Flipper zum Ausschlagen benötigt, noch nicht mit der des realen Aufbaus überein, diesbezüglich müsste der Geschwindigkeitswert der Flipper und die Durchlaufrate der Umgebung abgestimmt werden. Es ist möglich einfach Änderungen an der Umgebung vorzunehmen. Somit kann im weiteren Verlauf das Spielfeld mit Hindernissen besetzt werden und noch weitere Funktionen eingebracht werden. Die Steuerung durch einen Reinforcement Agenten konnte ebenso positive Ergebnisse erzielen, jedoch gibt es in diesem Bereich noch Optimierungspotential. Es wäre sinnvoll das Belohnungssystem zu optimieren oder die Parameter des Reinforcement Modells anzupassen um den Agenten in die richtige Richtung zu lenken.

7.2 Prototypischer Demonstrator

In diesem Abschnitt wird der prototypische Demonstrator begutachtet. Im ersten Schritt werden die gestellten Anforderungen geprüft, anschließend werden einzelne Anforderungen genauer betrachtet und abschließend wird eine Bewertung vorgenommen.

7.2.1 Anforderungsprüfung

Die in Abschnitt 3.3 aufgelisteten Anforderungen, werden im Folgenden überprüft. Es wird ebenfalls in Allgemein, Hardware und Software unterschieden.

Allgemein

DEMO-A-NF1: *Erfüllt:* Die Auflistung und Einhaltung der Projektkosten ist in Abschnitt 7.2.2 zu sehen.

DEMO-A-NF2: *Erfüllt:* Der Rahmen und die Kugel wurden extra so konzipiert, dass die Kugel nicht aus dem Spielfeld gelangen kann. Durch Fremdkörper oder andere Zustände ist es möglich, dass die Kugel jedoch aus dem Spielfeld gelangen kann. Für absolute Sicherheit könnte das Spielfeld nachträglich noch mit Plexiglas abgedeckt werden.

DEMO-A-NF3: *Erfüllt:* Dadurch, dass eine maximale Spannung von 24 V verwendet wird, fällt das System in die Kategorie der Schutzkleinspannung nach DIN VDE 0100, Teil 410. Nach dieser kann auf einen Schutz gänzlich verzichtet werden, wenn die Spannung kleiner als 25 V Wechselspannung beziehungsweise 60 V Gleichspannung ist. [32]

Hardware

DEMO-HW-F1: *Erfüllt:* Durch die Verwendung von Schrittmotoren ist der Bewegungsgrad frei zu wählen. Es wurden 70° realisiert.

DEMO-HW-F2: *Teilweise erfüllt:* Die Kugel ist nach spätesten 5 s wieder auf dem Spielfeld. Diese Anforderung wird in Abschnitt 7.2.3 genauer analysiert.

DEMO-HW-NF1: *Nicht erfüllt:* Es wurde eine Änderung diesbezüglich vorgenommen. Das Ziel des Agenten ist es, die Kugel so lange wie möglich im Spiel zuhalten. Die Realisierung mittels eines platzierten Zieles auf dem Spielfeld kann in weiteren Arbeiten aufgenommen werden.

DEMO-HW-NF2: *Erfüllt:* Die Maße des Spielfelds entsprechen den geforderten 50x34 cm.

DEMO-HW-NF3: *Erfüllt:* Die Länge der Flipper entsprechen den geforderten 4,5 cm.

DEMO-HW-NF4: *Erfüllt:* Das Spielfeld hat eine Neigung von 7°.

DEMO-HW-NF5: *Erfüllt:* Es wurden nur Elektronikbauteile verwendet, die auf eine breite Kompatibilität ausgerichtet sind. Somit ist das Ersetzen bzw. die anderweitige Verwendung dieser Bauteile gewährleistet.

DEMO-HW-NF6: *Erfüllt:* Die beiden Flipper sowie der „Schläger“ der Rückführungseinheit sind in rot.

DEMO-HW-NF7: *Erfüllt:* Die Flipper wurden trichterförmig mit einem Winkel von 35° zur Waagerechten platziert. Da Schrittmotoren verwendet wurden, lässt sich dieser Wert auch noch nachträglich einfach anpassen.

DEMO-HW-NF8: *Erfüllt:* Es wurde darauf geachtet, dass die verwendeten Bauteile für künftige Projekte wiederverwendbar sind. Abseits der 3D-gedruckten Bauteile wurden nur die Aluprofile in ihrer Länge und die Aluverbundplatte durch zwei Löcher dauerhaft verändert.

Software

DEMO-SW-F1: *Erfüllt:* Durch die Verwendung von Keras, lassen sich einfach Dateien in HDF5-Format generieren und abspeichern.

DEMO-SW-F2: *Erfüllt:* Es gibt einen Trainings-Modus. Diesem kann man eine gewünschte Anzahl von Durchläufen übergeben, die dann abgearbeitet werden.

DEMO-SW-F3: *Erfüllt:* Es gibt die Möglichkeit erlernte Fähigkeiten zu demonstrieren. Genau wie bei DEMO-HW-F2 kann die Anzahl der gewünschten Durchläufe variiert werden.

DEMO-SW-F4:	*Teilweise erfüllt:* Die Kugelposition wird durch ein Kamerasystem erfasst. Das Spielfeld muss jedoch per Hand markiert werden. Die automatische Erkennung kann Bestandteil weiterführender Arbeiten sein.
DEMO-SW-F5:	*Erfüllt:* Die synchrone Flipperbewegung ist möglich. Auch die versetzte Bewegung bei Erreichen der Endlage ist umgesetzt.

7.2.2 Kostenkalkulation

In diesem Abschnitt werden die gesamten Kosten für das Projekt aufgeführt. Es wird zwischen den Kosten der HAW, den Kosten von Herrn Prof. Dr. Hensel und vorrätigen Materialien unterschieden. Die vorrätigen Materialien stammen dabei von vorangegangenen Arbeiten. Zudem muss erwähnt werden, dass zwei Prototypen erstellt wurden, damit eine zeitgleiche Bearbeitung stattfinden konnte. Dieses Vorgehen hat erhebliche Austauschwege und Zeit gespart. Des Weiteren wurden die Kosten für ein Netzteil nicht berücksichtigt, da im Rahmen dieser Arbeit ein privates Labornetzteil verwendet wurde. Die Auflistung der Kosten ist in Tabelle 7.7 zu sehen.
Wie man erkennt liegen die Kosten für Neuanschaffungen insgesamt bei 393,36 €. Die von der HAW Hamburg übernommenen Kosten lagen jedoch innerhalb der geforderten 150 €. Die Arbeit wurde zu großen Teilen von Herrn Prof. Dr. Hensel privat mitfinanziert, da die Vorgaben der HAW, bezüglich der Beschaffung von Materialien, sehr einschränkend und zeitkritisch mit Bezug auf Abschlussarbeiten sind.
Letztendlich ist durch diese Arbeit aber eine Basis für zukünftige Projekte entstanden, bei denen die Anschaffungskosten sehr viel geringer ausfallen werden, da die grundlegende Hardware bereits vorhanden ist. Mit dieser Begründung sind die Kosten für Neuanschaffung akzeptierbar.

7.2.3 Rückführungs-Zeittest

Dieser Test bezieht sich auf die Zeit in Anforderung DEMO-HW-F2, bei der maximal 5 s zwischen dem Passieren der Flipper und der Rückführung ins Spielfeld liegen darf. Zu diesem Zweck wurde ein Video von dem Aufbau aufgenommen, mit diesem wurde dann geschaut, wie lange die Kugel benötigt. Als „Passiert“ gilt die untere Kante der Flipper. Als „wieder im Spielfeld“ gilt die Kugel, wenn diese aus dem Rückführungtunnel ausgetreten ist. In Abbildung 7.4 ist die Zeitauswertung zu sehen. Rechnet man nun die

Tabelle 7.7: Gesamtkosten

	Vorrätig	Kosten HAW	Kosten Prof. Hensel
Aluverbundplatte		49,88 €	
Alu-Profile/Eisenwaren	X		
Schrittmotoren		18,40 €	84,96 €
Schrittmotortreiber			56,97 €
Berührungssensoren		2,80 €	
Entwicklung Startvorrichtung			19,45 €
Filament	X		21,99 €
Schaltschrank		78,41 €	47,81 €
Arduino-Boards	X		
Kamera			12,69 €
Kamerastativ	X		
Total		149,49 €	243,87 €

Differenz aus, kommt man auf 3,1 s. Dieser Wert liegt unterhalb der geforderten 5 s und hält die Anforderung somit ein. Es muss jedoch erwähnt werden, dass die Zeit in gewissen Situationen länger als 5 s beträgt. Zum Beispiel kann es passieren, dass die Kugel bei sehr kleinen Geschwindigkeiten auf der Rampe liegen bleibt und somit das weitere Trainieren blockiert. In Weiterentwicklungen könnte die Rampe angepasst werden oder die Neigung des Flipper leicht erhöht werden, um einen insgesamt höheres Gefälle zu erzeugen. Zudem kann es bei dem Schrittmotor, welcher die Kugel zurück in Spielfeld bringt, zu Schrittverlusten kommen. Dies könnte durch Optimierungen in der Steuerung angepasst werden oder durch Volldrehung. Im letzteren Fall müsste der Aufbau leicht angepasst werden, da eine Volldrehung bei dem aktuellen Aufbau platztechnisch nicht möglich ist.

7.2.4 Bewertung

Zusammenfassend kann man sagen, dass die Steuerung der Aktoren mit Daten aus der Bildverarbeitung funktioniert. Jedoch kommt es bei der Bildverarbeitung noch zu Problemen durch die Bewegungen der Aktoren. Diese Bewegungen werden dann als Bewegung der Kugel interpretiert, in zukünftigen Arbeiten müsste man also schauen, dass diese Bewegungen durch die Bildverarbeitung herausgefiltert werden. Außerdem kann es dazu kommen, dass Schrittverluste bei den Schrittmotoren auftreten. Es wäre also sinnvoll, die Schrittmotoren nach jeder Episode neu zu kalibrieren. Dies könnte mit Hilfe

Abbildung 7.4: Rückführungszeit-Analyse

von Endlagensensoren oder über die Bildverarbeitung umgesetzt werden. Die Parameterübertragung bezüglich des Reinforcement Learning von der virtuellen Umgebung zum prototypischen Aufbau ist möglich, jedoch konnten wegen der unvollständigen Bildverarbeitung keine Tests durchgeführt werden.
Man kann jedoch sagen, dass durch den prototypischen Aufbau eine solide Grundlage erschaffen wurde, auf die in weiteren Arbeiten aufgebaut werden kann.

8 Fazit und Ausblick

Das Ziel war es, ein System zu entwickeln, welches das Spiel des Flippers durch Reinforcement Learning erlernt. Bestandteil war es eine virtuelle Umgebung und ein prototypischen Demonstrator zu entwickeln und alle dazugehörigen Auslegungen vorzunehmen. Sowohl die virtuelle Umgebung, als auch der prototypische Demonstrator wurden in Zusammenarbeit mit Herrn Prof. Dr. Hensel umgesetzt. Bei der Umsetzung des Reinforcement Agenten gibt es jedoch noch Optimierungsmöglichkeiten. Im Zuge der Optimierung sollten verschiedene Agenten erprobt werden, zudem wäre es sinnvoll auch die übergeben Parameter, also die *observation*, zu variieren und zu analysieren.
Eine weitere Anmerkung für zukünftige Arbeiten an den Flipper-Automaten wäre auch das Anbringen einer integrierten Kamerahalterung. Diese könnte mittels Aluprofilen am Rahmen befestigt werden. Eine solche Vorrichtung hätte den Vorteil, dass man immer dieselbe Kameraposition eingestellt hätte. Darüber hinaus wäre es platzsparend, da das Kamerastativ nicht neben dem Spielfeld platziert werden müsste. Nach Absolvierung dieser beiden Punkte hätte man eine beeindruckendes Beispiel für ein hoch dynamisches System, welches durch eine KI gesteuert werden kann.
Nach Selbstreflexion der Arbeitsweise ist erwähnenswert, dass eine Zunahme der Strukturierung insgesamt zu verzeichnen war. Die „Strukturierung-Kennlinie“ stieg dabei kontinuierlich über die gesamte Zeit der Arbeit an. Dies bezieht sich sowohl auf die schriftlichen Ausarbeitung, als auch auf die programmiertechnische Umsetzung. Zudem fiel das Schreiben der Arbeit mit zunehmender Zeit leichter, was auch dazu führte, dass ich immer neue Ideen bezüglich der Ausarbeitung und des Aufbaus bekam. Eine weitere Erkenntnis war, wie sinnvoll UML-Diagramme sind. Zu Anfang der Arbeit waren diese Diagramme nur flüchtig aus Vorlesungen bekannt. Mit Voranschreiten der Ausarbeitung fiel jedoch auf, wie wichtig diese für eine gute Dokumentation und vor allem auch für das eigene Verständnis waren.
Zusammenfassend und abschließend kann man sagen, dass diese Arbeit eine solide Grundlage für weitere Arbeiten am System selbst ist, aber auch eine Motivation für weitere persönliche wissenschaftliche Arbeiten geschaffen hat.

Literaturverzeichnis

[1] *ALU Profil Aluprofil 20x20mm*. URL https://tktrading24.de/ALU-Profil-Aluprofil-20x20mm-Nut-6-Aluminium-Systemprofil-Typ-I-190cm-1900mm. – Zugriffsdatum: 20.12.2023

[2] *Aluverbundplatten*. URL https://expresszuschnitt.de/Aluverbundplatten-DILITE-blau?_gl=1*1hj9z5x*_up*MQ..&gclid=EAIaIQobChMIv_mnnKPGggMVIYVoCR1eZgsZEAAYASAAEgKwhfD_BwE. – Zugriffsdatum: 20.12.2023

[3] *An API standard for reinforcement learning with a diverse collection of reference environments*. URL https://gymnasium.farama.org/. – Zugriffsdatum: 15.12.2023

[4] *Arduino Mega*. URL https://www.reichelt.de/de/en/arduino-mega-2560-atmega1280-usb-arduino-mega-p119696.html?search=arduino+mega&&r=1. – Zugriffsdatum: 10.03.2024

[5] *Arduino Nano*. URL https://www.reichelt.de/de/en/arduino-nano-v3-atmega-328-mini-usb-arduino-nano-p142943.html?search=arduino+nano&&r=1. – Zugriffsdatum: 10.03.2024

[6] *ArduinoUno*. URL https://www.reichelt.de/de/en/arduino-uno-rev-3-atmega328-usb-arduino-uno-p119045.html?PROVID=2788&gad_source=1&gclid=CjwKCAiA0bWvBhBjEiwAtEsoW843GlROy3tVWc9m9yneO681kKzhwrYhHd8DgGQ9H24Cjqzl4cfRoRoCJ5AQAvD_BwE&&r=1. – Zugriffsdatum: 10.03.2024

[7] *Birke Platte Multiplex wasserfest BB/BB*. URL https://expresszuschnitt.de/Multiplexplatte-Birke-Multiplex?_gl=1*1rs2v3c*_up*MQ..&gclid=EAIaIQobChMIuMmBwu2dgwMVS4poCR2zngjfEAAYASAAEgKAAvD_BwE. – Zugriffsdatum: 20.12.2023

[8] *Box2D*. URL https://gymnasium.farama.org/environments/box2d/. – Zugriffsdatum: 26.11.2023

[9] *Classic Control*. URL https://gymnasium.farama.org/environments/classic_control/. – Zugriffsdatum: 26.11.2023

[10] *Cullmann Alpha 1000*. URL https://www.idealo.de/preisvergleich/OffersOfProduct/201584299_-alpha-1000-cullmann.html. – Zugriffsdatum: 12.03.2024

[11] *Environments*. URL https://www.tensorflow.org/agents/tutorials/2_environments_tutorial. – Zugriffsdatum: 06.03.2024

[12] *Flipper-Assembly*. URL https://flipperteile.com/de/Flipperfinger-und-Mechanik/komplette-Einheiten/Williams-Flipper-Assembly-links--mit-Spule-FL-xxx--Druckfeder-u--Wolframkontakt.html. – Zugriffsdatum: 31.01.2024

[13] *Flipper selber bauen*. URL https://flipperteile.com/de/flipper-selber-bauen/das-gehaeuse.html. – Zugriffsdatum: 31.01.2024

[14] *Flipperfinger*. URL https://flipperteile.com/de/Flipperfinger-und-Mechanik/Flipperfinger/Flipperfinger--orig-Ref--Nr-----BALLY----C-611-1--Flipperfinger----fuer-schraubbare-Flipperachse--Farbe-rot-76mm-original-Ersatzteil.html. – Zugriffsdatum: 31.01.2024

[15] *Flipperkugeln*. URL https://www.flipperteile.de/de/Ersatzteile/Flipperkugeln/. – Zugriffsdatum: 20.12.2023

[16] *Goldball*. URL https://www.pinball-shop.de/product_info.php?p_id=1147&hd=1&mcID=17. – Zugriffsdatum: 17.02.2024

[17] *Hammermutter Nutenstein*. URL https://tktrading24.de/Hammermutter-Nutenstein-M4-Nut-6-fuer-Aluprofil-20-er-1-Stueck. – Zugriffsdatum: 17.02.2024

[18] *Häufig gestellte Fragen zu Murmeln*. URL https://www.murmelprinz.de/service/fragen-zu-murmeln/. – Zugriffsdatum: 06.03.2024

[19] *Innenwinkel Aussenwinkel*. URL https://tktrading24.de/Innenwinkel-Aussenwinkel-M5-Nut-6-Profilverbinder-fuer-Aluprofil-20-er-1-Stueck. – Zugriffsdatum: 17.02.2024

[20] *Introduction to graphs and tf.function*. URL https://www.tensorflow.org/guide/intro_to_graphs. – Zugriffsdatum: 15.12.2023

[21] *Jelly Comb W06*. URL https://www.idealo.de/preisvergleich/OffersOfProduct/201231529_-w06-jelly-comb.html. – Zugriffsdatum: 12.03.2024

[22] *JOY-IT TB6600*. URL https://www.reichelt.de/de/en/stepper-motor-driver-9-42-vdc-4-a-joy-it-tb6600-p365537.html?CCOUNTRY=445&LANGUAGE=fr&GROUPID=9630&START=0&OFFSET=16&SID=92476fad14139ac147ad15279adcabaa1a1a326691a63ed545c3b&LANGUAGE=EN&&r=1. – Zugriffsdatum: 08.03.2024

[23] *Keras*. URL https://keras.io/. – Zugriffsdatum: 15.12.2023

[24] *Kickerbälle*. URL https://www.sport-thieme.de/Freizeitspiele/Tischkicker/Kickerb%C3%A4lle#:~:text=Kickerb%C3%A4lle%20gibt%20es%20in%20unterschiedlichen,und%20verhindern%20einen%20fl%C3%BCssigen%20Spielverlauf.. – Zugriffsdatum: 20.12.2023

[25] *Kugeln von GraviTrax®*. URL https://service.ravensburger.de/Marken_und_Produkte/Ravensburger_Produkte/GraviTrax%C2%AE/Kugeln_von_GraviTrax%C2%AE#:~:text=Die%20Kugel%2C%20die%20beim%20Tiptube,fungiert%2C%20hat%208%20mm%20Durchmesser.. – Zugriffsdatum: 20.12.2023

[26] *Make your own custom environment*. URL https://www.gymlibrary.dev/content/environment_creation/. – Zugriffsdatum: 26.11.2023

[27] *MuJoCo*. URL https://gymnasium.farama.org/environments/mujoco/. – Zugriffsdatum: 26.11.2023

[28] *Multiplexplatte*. URL https://de.wikipedia.org/wiki/Multiplexplatte. – Zugriffsdatum: 20.12.2023

[29] *Nema 17 Bipolar*. URL https://www.omc-stepperonline.com/de/nema-17-bipolar-59ncm-84oz-in-2a-42x48mm-4draehte-w-1m-kabel-volle-d-cut-welle-17hs19-2004s2. – Zugriffsdatum: 08.03.2024

[30] *PVC Hartschaum Platte weiß*. URL https://expresszuschnitt.de/PVC-Hartschaumplatten?_gl=1*1rs2v3c*_up*MQ..&gclid=EAIaIQobChMIv_mnnKPGggMVIYVoCR1eZgsZEAAYASAAEgKwhfD_BwE. – Zugriffsdatum: 20.12.2023

[31] *Quadratleiste Kiefer 20 mm x 20 mm*. URL https://www.obi.de/leisten/quadratleiste-kiefer-20-mm-x-20-mm-laenge-2400-mm/p/1040195?wt_mc=gs.pla.Wohnen.Bodenbelaege.LaminatParkettVinylboeden&wt_cc1=664643021&wt_cc2=1040195&wt_cc4=c&wt_cc5=140694732450&wt_cc8=pla-online&wt_cc9=32127398845&storeId=&gad_source=1&gclid=EAIaIQobChMIhI-umZiegwMV0lNBAh084QcIEAQYAiABEgIHOfD_BwE&et_uk=8d7473dd9d724091be02061d831af0b2. – Zugriffsdatum: 20.12.2023

[32] *Schutzkleinspannung*. URL https://sicherheitsingenieur.nrw/glossar/schutzkleinspannung/. – Zugriffsdatum: 26.02.2024

[33] *Schwarz Weiss Bilder*. URL https://www.pinterest.de/pin/who-dunnit-pinball--80783387046755688/. – Zugriffsdatum: 28.02.2024

[34] *The Sequential model*. URL https://keras.io/guides/sequential_model/. – Zugriffsdatum: 11.03.2024

[35] *Spielfläche*. URL https://flipperteile.com/de/flipper-selber-bauen/spielflaeche.html#:~:text=F%C3%BCr%20die%20Spielfl%C3%A4che%20ist%20folgendes,x%2051%2C5cm%20(19mm). – Zugriffsdatum: 28.11.2023

[36] *Teil 5: Einen analogen Plunger im Virtual Pinball installieren*. URL https://www.virtual-pinball-cabinet.com/virtual-pinball-installation-des-analogen-plungers/. – Zugriffsdatum: 28.02.2024

[37] *Tipps und Tricks*. URL https://flipper-fan.de/html/tipps_-_tricks.html#:~:text=Die%20optimale%20Spielfeld%2DNeigung%20liegt,ist%20aber%20auf%20Dauer%20todlangweilig.. – Zugriffsdatum: 23.12.2023

[38] *Toy Text*. URL https://gymnasium.farama.org/environments/toy_text/. – Zugriffsdatum: 26.11.2023

[39] *VPyhton*. URL https://vpython.org/. – Zugriffsdatum: 22.01.2024

[40] *Zweidimensionaler elastischer Stoß*. URL https://de.wikipedia.org/wiki/Sto%C3%9F_(Physik). – Zugriffsdatum: 13.12.2023

[41] *Was sind Schrittmotoren, welche Typen gibt es und wie funktionieren sie?* URL https://kem.industrie.de/elektromotoren/was-sind-schrittmotoren-welche-typen-gibt-es-und-wie-funktionieren-sie/. – Zugriffsdatum: 26.02.2024, 2021

[42] *Q-Learning – einfach erklärt*. URL https://databasecamp.de/ki/q-learning. – Zugriffsdatum: 05.03.2024, 2023

[43] *Reinforcement Learning - Developing Intelligent Agents*. URL https://deeplizard.com/learn/video/eMxOGwbdqKY. – Zugriffsdatum: 13.12.2023, 2023

[44] *SARSA – Machine Learning mit verstärkendem Lernen*. URL https://datascientest.com/de/sarsa-machine-learning-mit-verstaerkendem-lernen. – Zugriffsdatum: 05.03.2024, 2024

[45] Antonio Gulli, Sujit P.: *Deep Learning with TensorFlow 2 and Keras*. Packt Publishing, 2019. – ISBN 9781838823412

[46] Benjamin Maack: Kultspielzeug Flipperautomat. In: *Spiegel* (2008). – URL https://www.spiegel.de/geschichte/kultspielzeug-flipperautomat-a-948020.html. – Zugriffsdatum: 21.11.2023

[47] Chornaya, Jenia: Die besten Programmiersprachen zum Coden von KI. In: *HubSpot* (2024). – URL https://blog.hubspot.de/website/ki-programmiersprachen. – Zugriffsdatum: 06.03.2024

[48] Ciaburro, Giuseppe: *Keras Reinforcement Learning Projects*. Packt Publishing, 2018. – ISBN 9781789342093

[49] Dadhich, Abhinav: *Practical Computer Vision*. Packt Publishing, 2018. – ISBN 9781788297684

[50] Fandango, Armando: *Python Data Analysis*. Packt Publishing, 2017. – ISBN 9781787127487

[51] Joo, John: Deep Reinforcement Learning. In: *Domino* (2024). – URL https://domino.ai/blog/deep-reinforcement-learning. – Zugriffsdatum: 11.03.2024

[52] Michael A. Runde: *Eindimensionale Stoßvorgänge*. URL https://physikbuch.schule/collisions-1d.html. – Zugriffsdatum: 28.11.2023, 2023

[53] Michael Vollmer, Klaus-Peter M.: *Von Bällen und Schlägern*. URL https://pro-physik.de/nachrichten/von-baellen-und-schlaegern. – Zugriffsdatum: 17.03.2024

[54] Palanisamy, Praveen: *Hands-On Intelligent Agents with OpenAI Gym*. Packt Publishing, 2018. – ISBN 9781788836579

[55] Reich, Stephan: Die besten Programmiersprachen zum Coden von KI. In: *Spiegel* (2020). – URL https://blog.hubspot.de/website/ki-programmiersprachen. – Zugriffsdatum: 06.03.2024

[56] Schaad, St.: Schrittmotor kurz erklärt. In: *DeltronAG*. – URL https://wiki.bu.ost.ch/infoportal/_media/hardware/sysp/bauteile/schrittmotor_kurz_erklaert_d.pdf. – Zugriffsdatum: 26.02.2024

[57] Schmole, Melanie: Welche Programmiersprache solltest du lernen? In: *get in IT* (2024). – URL https://www.get-in-it.de/magazin/bewerbung/it-skills/welche-programmiersprache-lernen. – Zugriffsdatum: 17.02.2024

[58] Suran, Abhishek: *On-Policy v/s Off-Policy Learning*. URL https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f. – Zugriffsdatum: 10.03.2024, 2020

[59] Thilakarathne, Haritha: *Using Hierarchical Data Format (HDF5) in Machine Learning*. URL https://naadispeaks.blog/2021/06/30/using-hierarchical-data-format-hdf5-in-machine-learning/, 2021

[60] Thorsten Ostermann: *Die Sache mit der Spannung*. URL https://www.schrittmotor-blog.de/die-sache-mit-der-spannung/. – Zugriffsdatum: 26.02.2024, 2011

[61] Thorsten Ostermann: *Die richtige Spannung für Schrittmotorsteuerungen – Kriterien zur Auswahl*. URL https://www.schrittmotor-blog.de/die-r

ichtige-spannung-fuer-schrittmotorsteuerungen/. – Zugriffsdatum: 26.02.2024, 2016

[62] Wikipedia: *Flipperautomat*. URL http://de.wikipedia.org/w/index.php?title=Flipperautomat&oldid=238508257. – Zugriffsdatum: 21.11.2023

[63] Wolf, Marilyn: *Deep Learning with TensorFlow 2 and Keras*. Morgan Kaufmann, 2017. – ISBN 9780128053874

[64] ZIMMERMANN F., WILHELM T.: *Fußball im Physikunterricht*. URL https://www.thomas-wilhelm.net/veroeffentlichung/Fussball_Computer.pdf. – Zugriffsdatum: 17.03.2024

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original