Universität Bremen
Fachbereich 3: Mathematik und Informatik

Masterarbeit

Training neuronaler Netze für Perzeption und Schlagbewegung eines ballspielenden Roboters ohne reale Daten

Arne Neisser
Matrikelnummer: 4434156

Erstprüfer:	Prof. Dr.-Ing. Udo Frese
Zweitprüfer:	Prof. Dr. Ralf Bachmayer

1. Juni 2023

Universität Bremen

Offizielle Erklärungen von

Nachname: _____ Vorname: _____

Matrikelnr.: _____

A) Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.
Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht.
Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein.
Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

B) Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach u. Jahr.

☒ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

C) Einverständniserklärung über die Bereitstellung und Nutzung der Bachelorarbeit / Masterarbeit / Hausarbeit in elektronischer Form zur Überprüfung durch Plagiatssoftware

Eingereichte Arbeiten können mit der Software *Plagscan* auf einen hauseigenen Server auf Übereinstimmung mit externen Quellen und der institutionseigenen Datenbank untersucht werden.
Zum Zweck des Abgleichs mit zukünftig zu überprüfenden Studien- und Prüfungsarbeiten kann die Arbeit dauerhaft in der institutionseigenen Datenbank der Universität Bremen gespeichert werden.

☒ Ich bin damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum Zweck der Überprüfung auf Plagiate auf den *Plagscan*-Server der Universität Bremen hochgeladen wird.

☐ Ich bin ebenfalls damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum o.g. Zweck auf dem *Plagscan*-Server der Universität Bremen hochgeladen u. dauerhaft auf dem *Plagscan*-Server gespeichert wird.

☐ Ich bin nicht damit einverstanden, dass die von mir vorgelegte u. verfasste Arbeit zum o.g. Zweck auf dem *Plagscan*-Server der Universität Bremen hochgeladen u. dauerhaft gespeichert wird.

Mit meiner Unterschrift versichere ich, dass ich die oben stehenden Erklärungen gelesen und verstanden habe. Mit meiner Unterschrift bestätige ich die Richtigkeit der oben gemachten Angaben.

_____ _____

Datum, Ort Unterschrift

Inhaltsverzeichnis

1 Einleitung

In dieser Masterarbeit untersuchen wir den Einsatz von neuronalen Netzen zur Verbesserung eines Ballspielroboters in zwei Anwendungsbereichen: der Erkennung eines Balls mithilfe eines echtzeitfähigen Convolutional neuronalen Netzwerks (CNN), das ausschließlich auf synthetischen Daten trainiert wurde, und einer Trajektorienplanung für das Zurückschlagen eines Balls, basierend auf einem zweiten neuronalen Netz. Der Einsatz von neuronalen Netzen zur Objekterkennung und Lokalisierung ist in der Robotik ein aktuelles und vielversprechendes Forschungsthema, da sie in der Lage sind, komplexe Muster und Zusammenhänge in Daten zu erkennen. Dadurch können Objekte in Echtzeit zuverlässig erkannt und verfolgt werden. In Kombination mit simulierten Trainingsdaten können neuronale Netze auf weit entfernten Anwendungsbereichen trainiert werden und im realen Einsatz präzise Ergebnisse liefern.

Die Trajektorienplanung für den Schlagarm des Ballspielroboters ist ein wichtiger Faktor für die Gesamtleistung des Systems. Um einen Ball erfolgreich schlagen zu können, muss die richtige Flugbahn des Balls vorhergesagt und eine entsprechende Bewegung des Schlagarms erzeugt werden. Dies erfordert die Integration von Sensordaten für die Erkennung und die Berücksichtigung von verschiedensten Faktoren. Dazu gehören die Geschwindigkeit und Richtung des Balls, die Bewegung des Roboters und möglicherweise auch externe Einflüsse wie Wind und Reibung.

1.1 Motivation

Das Ballspielen als Problem strahlt auf den ersten Blick nicht gerade vor Sinnhaftigkeit und ist trotzdem die Grundlage dieser und vieler weiterer Arbeiten auf dem Gebiet der Robotik. Auch weitere, zuerst spielerisch anmutende, Aufgaben und Probleme werden oft in der Forschung genutzt. Dabei können technische und methodische Neuerungen ausprobiert werden oder sie dienen als Entwicklungsdomänen, um komplexe und sehr spezifische Probleme herunterzubrechen und insgesamt zu verallgemeinern. Diese spielerischen Aufgaben haben in der Robotik speziell wichtige Anwendungen. Durch das Lösen dieser Art von Probleme können Fortschritte in der Entwicklung von Robotern gemacht werden, die in der Lage sind in der realen Welt zu agieren und mit der realen Welt zu interagieren. Dies hat weitreichende Auswirkungen auf verschiedene Anwendungsbereiche, deren Gesamtprozess für einen Einsatz von Robotik zu komplex ist. Die Kombination guter Ergebnisse kleinerer Problemstellungen können so komplexe Zielszenarien abbilden und ermöglichen die Forschung in komplexen Bereichen. Von der Industrie, bis hin zu Haushaltsgeräten und sogar in der Pflege und dem Gesundheitswesen können einfache, spielerische Szenarien Teile eines komplexen Arbeitsablaufes abbilden. Probleme können klein und einfach definiert und von Robotern gelöst werden. In ersten Schritten werden damit Menschen unterstützt und Vorgehensweisen erprobt. Erlangte Ergebnisse fließen in spätere Studien ein und können dort weiter genutzt werden. Damit wird zusätzlich die Akzeptanz von Robotern in der Arbeitswelt langsam gestärkt. In dem ausgewählten Szenario für diese Arbeit definiert das Ballspielen einen sehr zeitkritischen, fortschreitenden Vorgang ohne Platz für Fehler oder große Korrekturen. Der Ball muss innerhalb weniger Millisekunden nach dem Verlassen der Hand erkannt, seine Flugbahn vorhergesagt und ein Interaktionszeitpunkt sowie -ort im Aktionsradius des Roboters aus einer Vielzahl

Möglicher ausgewählt werden. Die Planung der Interaktion muss die physikalischen Grenzen berücksichtigen und trotzdem einen adäquaten Impuls auf den Ball auswirken. Insgesamt stellt dieses Szenario eine enorme Herausforderung dar, da es hohe Anforderungen an die Genauigkeit und Geschwindigkeit der Verarbeitung sensorischer Daten stellt. Zusätzlich wird das Potential von simulierten Trainingsdaten immer größer. Die Leistungen in der Modellierung täuschend echter Welten werden dabei immer mehr genutzt, um zusammen mit datengetriebenen Verfahren die Datensätze zu vergrößern und diversifizieren. Die Fortschritte im Bereich der Simulationstechnologie haben es ermöglicht, hochrealistische Szenarien zu erschaffen, die den realen Bedingungen sehr ähnlich sind. Simulierte Trainingsdaten bieten kostengünstige und zeitsparende Alternativen zur Datenerfassung und ermöglichen es Unternehmen und Forschungseinrichtungen, sich verstärkt auf die Entwicklung und Verbesserung ihrer visuellen Systeme zu konzentrieren. Durch die Verwendung simulierter Trainingsdaten können verschiedene Szenarien und Bedingungen einfach angepasst und variiert werden, um das Verhalten des visuellen Systems unter verschiedenen Situationen zu untersuchen. Dies ermöglicht den Entwicklern, ihr Modell gezielt zu trainieren und auf spezifische Anforderungen vorzubereiten. In Kombination mit realen Daten können simulierte Trainingsdaten die Effizienz und Vielseitigkeit der Entwicklung von visuellen Systemen weiter steigern und deren Fähigkeit verbessern, den Herausforderungen der realen Welt gerecht zu werden.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, den Einsatz verschiedener neuronaler Netze in der Robotik an einem bestehenden Roboter in einem bestehenden Szenario zu evaluieren. Durch den Einsatz von state-of-the-art neuronaler Netze soll ein System entwickelt werden, das in der Lage ist, geworfene Bälle zuverlässig und präzise in Echtzeit zu erkennen und eine effektive Schlagbewegung zu planen. Außerdem soll eine potenzielle Integration des entwickelten Systems am bestehenden Roboter evaluiert werden. Das Zielszenario des Roboters ist das Erkennen eines geworfenen Balles während der Flugphase und die Durchführung einer effizienten Schlagbewegung mit dem End-Aktuator. Das Ziel des Schlags ist ein zurückspielendes Verhalten zur werfenden Person. Dabei soll die Erkennung des Balles, die Vorhersage über die Flugbahn und die Planung der Schlagbewegung frühzeitig abgeschlossen sein, um einen möglichst präzisen und effizienten Schlag zu erlauben. Ein weiteres Ziel dieser Arbeit ist die Entwicklung dieses Systems ohne die Verwendung von aufgenommenen Daten aus dem Zielszenario des Roboters.

Die Arbeit beinhaltet die Erläuterungen der Grundlagen über den bestehenden Roboter und verwendeter Techniken zur Erzeugung einer Schlagtrajektorie, Entwicklung und Training eines CNN für die visuelle Ballerkennung zusammen mit der systematischen Erzeugung synthetischer Trainingsdaten, Entwicklung und Training eines zweiten neuronalen Netzes mittels anwendungsspezifischer Verlustfunktion zur Szenarienerfüllung des ballspielenden Roboters und die Evaluierung beider erarbeiteter Systeme.

2 Stand der Technik

2.1 Verwandte Arbeiten

In einem vorangegangenen Projekt von Hasselbring, Frese & Röfer [12] des Deutschen Forschungszentrum für Künstliche Intelligenz, Cyber-Physical Systems, wurden optimale Trajektorien für Roboter durch eine modellbasierte Loss-Funktion an einem neuronalen Netz gelernt. Die modellbasierte Loss-Funktion definiert dabei das Optimierungsziel der Trajektorien. Dieser Ansatz steht einerseits dem Lernen von einer vorher berechneten optimalen Trajektorie und andererseits dem Reinforcement-Learning entgegen. Die durchgeführte Studie zeigt in Simulation qualitativ ein erfolgversprechendes Ergebnis durch die Einhaltung von definierten technischen Grenzen und der Erzeugung brauchbarer Bewegungstrajektorien. Elsisi et al. [8] entwickelten 2021 einen Algorithmus zum Einstellen und Verbessern eines PID Controllers mittels neuronalen Netzen für die Kontrolle eines Roboterarms. Eine vorgegebene Trajektorie soll mittels PID Controller vom Roboterarm ausgeführt werden, wobei die Leistung stark von einzelnen Parametern des Controllers abhängt. Ein entwickelter iterativer Algorithmus verbessert diese Parameter, sodass einer kubischen Positionstrajektorie gefolgt werden kann und gegenüber Baseline Ansätzen die Setzungszeit als auch das Überschwingen verringert wird. Anders als bei Hasselbring et al. und ähnlicher zu Elsisi et al. nutzen wir Ausgaben des neuronalen Netzes als Parameter für die Trajektorienerzeugung und optimieren diese zur Erfüllung der Rückspielaufgabe.

Van Zandycke und De Vleeschouwer stellten 2019 [24] eine Erkennung von Bällen basierend auf einem Farbbild und einem Differenzbild aufeinanderfolgender Zeitpunkte vor. Durch die Nutzung beider Bilder erreichten Sie eine Erkennungsleistung von bis zu 70% mit einem Durchsatz von 12 bis 24 Bildern pro Sekunde. 2022 [21] nutzten Sie die entwickelte Erkennungsmethode um die Lösung von einer Erkennung auf Bildern zu einer Erkennung im 3D-Raum. Dafür wird zusätzlich der Radius erkannt, sodass eine Berechnung mit der genormten Größe von Basketbällen und den Kameraparametern die Distanz eingegrenzt werden kann. Sie erreichten in guten Konditionen einen durchschnittlichen Distanzfehler zum Ball von 1,8 Metern. Dies ist nach eigenen Aussagen keine genaue Positionsvorhersage, die ermittelten Werte können jedoch als Basis für Spielstatistiken benutzt werden.

2.2 Doggy

Der Ballspielroboter Doggy ist ein 2,21 Meter großer mobiler Roboter, dessen Entwicklungsprozess durch mehrere Iterationen und verschiedenen wissenschaftlichen Arbeiten begleitet wird. Die grundsätzliche Idee war die Entwicklung eines Unterhaltungsroboters der mit Menschen interagiert. Zu Beginn entwickelte Hammer 2011 [11] einen Vorläufer *Piggy* (Abbildung 2.1), der als Akteur im Kinderspiel *Schweinchen in der Mitte* als Schweinchen agieren sollte. Dabei war das Ziel den Einsatz von Robotern und entwickelten Algorithmen im Umfeld von Menschen zu evaluieren. 2013 stellten Laue, Birbach, Hammer und Frese [16] die Zusammenführung mehrerer Arbeiten als ballspielenden Roboter vor und führten weitere Versuche durch. In einer Revision des Roboters, begleitet und entwickelt in der Dissertation von Schüthe [19], soll die Mensch-Roboter-Interaktion dann nicht nur durch die Aktion des Ballspielens erzeugt werden. Durch die Implementierung einer weiteren Bewegungsachse, der Gier-Achse, entstand ein weiterer Freiheitsgrad für die Bewegung des

Schlägers. Damit sind natürlichere und weitere Bewegungen möglich. Reaktives Verhalten auf Geräusche im Raum, als auch eine Animatronik in der Rolle eines Hundes zusammen mit einem neuen Kostüm führten von dem Ballspielroboter *Piggy* zum Unterhaltungsroboter *Doggy*. Außerdem soll Interaktion in Form einer Abwandlung des Spiels *Ball über die Schnur* stattfinden. Dabei soll der Fokus der Interaktion nicht auf einem kompetitiven sondern kollaborativen Verhalten beider Parteien liegen, sodass ein Zusammenspiel beider Parteien gefördert wird.

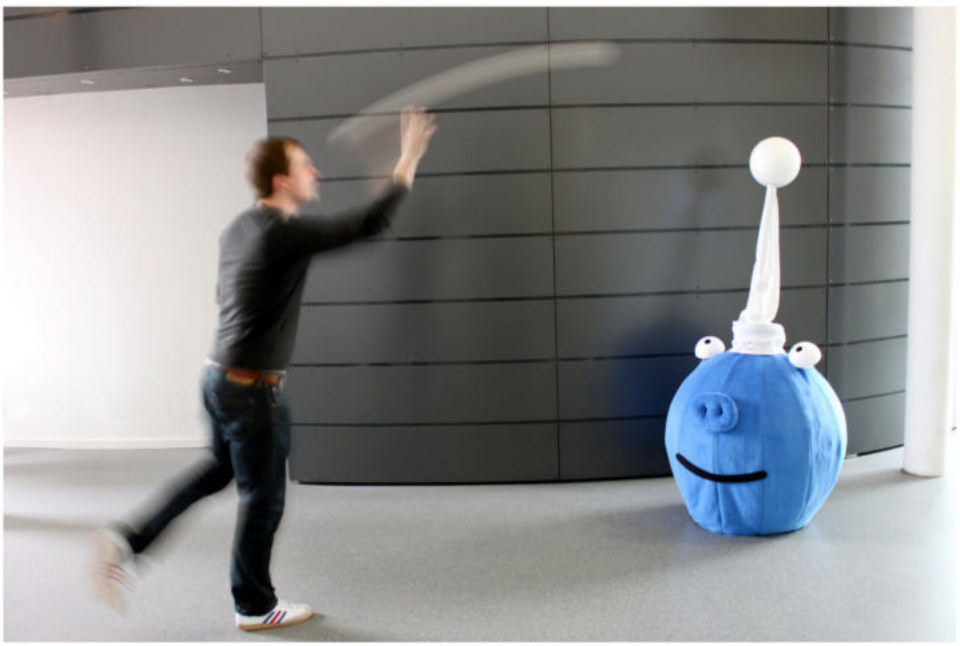

Abbildung 2.1: Vorentwicklung vom *Doggy*: Roboter *Piggy* als Schweinchen verkleidet während eines Wurfs. Grafik von Birbach, 2015 [2].

2.2.1 Zielszenario

Das Zielszenario definiert das Wunschverhalten und die Interaktion zwischen Mensch und Roboter. Der Spieler beginnt mit einem Wurf in Richtung des Roboters aus einer Distanz zwischen 3 und 5 Metern. Der Ball soll dabei so in den Bewegungsradius des Roboters geworfen sein, dass ein kollaboratives Zusammenspiel möglich ist. Ein Zusammenstoß zwischen Endaktuator und Ball soll herbeigeführt werden, sodass ein Zurückspielen erzeugt wird. Der retournierte Ball soll dann abwechselnd vom Menschen und vom Roboter gespielt werden ohne den Boden zu berühren. Ein geworfener Ball ist dabei zwischen 400ms und 900ms in der Luft, bevor er auf den Aktionsradius des Roboters trifft. Um einen sinnvollen Schlag zu erzeugen, muss innerhalb der ersten 100ms nach erster Erkennung eine Schlagtrajektorie berechnet sein und eine Bewegung initiiert werden. Das Zielszenario sieht vor, dass der Roboter in möglichst vielen Einsatzgebieten funktioniert und damit ein breites Spektrum an visuellen Störfaktoren abdeckt. In aufgenommenen Videos von Schüthe, 2017 ist ein Zusammenschnitt[1] einer Demonstration zu sehen. Der Roboter ist unter freiem Himmel aufgebaut und wurde über mehrere Stunden von Kindern mit Bällen bespielt. Der Aufbau war ein Teil der durchgeführten Tests des Gesamtsystems. Dabei sind wenige Bälle erreichbar zurückgespielt worden, sodass eine Entwicklung eines genaueren Schlags als Quintessenz des Tests hervorging. Abbildung 2.3 zeigt ebenfalls einen Einsatz von Doggy bei dem eine Kollision mit einem geworfenen Ball angestrebt wird.

[1] `https://www.informatik.uni-bremen.de/agebv2/downloads/videos/doggyOpenCampusDay.mp4`

2.2.2 Aufbau und relevante Komponenten

Mechanische Komponenten

Der Roboter *Doggy* besteht aus 4 physischen Komponenten die durch 3 Drehgelenke miteinander verbunden sind. Abbildung 2.2 zeigt den Roboter in verschiedenen Ansichten: v.l.n.r. in der namensgebenden Verkleidung, der Aufbau mit sichtbaren Komponenten, das konstruierte CAD Modell und zuletzt einen schematischen Aufbau der Komponenten. Im CAD Modell und im schematischen Aufbau sind korrespondierende Elemente in der gleichen Farbe dargestellt. Der Definition nach rotieren Drehgelenke um ihre Z-Achse, sodass in der CAD Zeichnung die Gelenke J1, J2 und J3 mit angezeigten Koordinatensystemen eine Drehung um die blau dargestellte Z-Achse erwirken. Die Drehgelenke werden in dieser Thesis als X, Y und Z bzw. als 1., 2. und 3. Element in Vektoren referenziert, wobei die Gier-Achse (yaw, J1) als erstes Element, die Nick-Achse (pitch, J2) als zweites Element und die Roll-Achse (roll, J3) als drittes Element festgelegt ist. Der gesamte Roboteraufbau ist in neutraler, aufrechter Position 221cm hoch, wobei 100cm die Höhe der Basis sind und 121cm der bewegliche Schläger. Die breiteste Stelle des Roboters ist die Basis mit rund 50cm im Durchmesser. Die Basis beinhaltet das Rechensystem und die Netzteile für die gesamte Stromversorgung der Motoren und des Systems. An der Basis ist über die Gier-Achse die Hüfte des Roboter angebunden. Zwei Kameras sind direkt links und rechts in der Hüfte verbaut und sie beinhaltet räumlich den restlichen Bewegungsapparat. Die Gelenke werden über Gleichstrommotoren mit einer Zahnriemen-Umsetzung bewegt. Durch die kompakte Bauweise der Hüfte ist die Bewegungsfreiheit des Schlägers sowohl nach hinten, als auch nach links und rechts relativ stark eingegrenzt. Der Bewegungsradius und die Reichweite des Endaktuators ist damit Kuppelförmig aufgebaut.

Visuelle Komponenten

Für die Erkennung von fliegenden Bällen ist im bestehenden Roboter ein Stereokamerasystem verbaut. Links und rechts in der Hüfte des Roboters auf einer Höhe von 83,6cm sind im Abstand von 44,7cm zwei Industriekameras verbaut, die mit 50Hz monochrome Kamerabilder in einer Auflösung von 768x576 Pixeln aufnehmen. Die Kameras sind in einem 35 Grad Winkel zum Horizont angebracht und besitzen ein horizontales Blickfeld von 57 Grad. Eine Registrierung des gleichen Objekts auf beiden Kamerabildern ermöglicht eine Lokalisierung des Objekts relativ zum Kamerasystem, und damit zum Roboter. Durch ein von Birbach 2015 [2] entwickeltes Erkennungs- und Verfolgungssystem für geworfene Bälle, werden Vorhersagen über Flugtrajektorien erstellt, die für eine entsprechende Bewegung des Roboters genutzt werden.

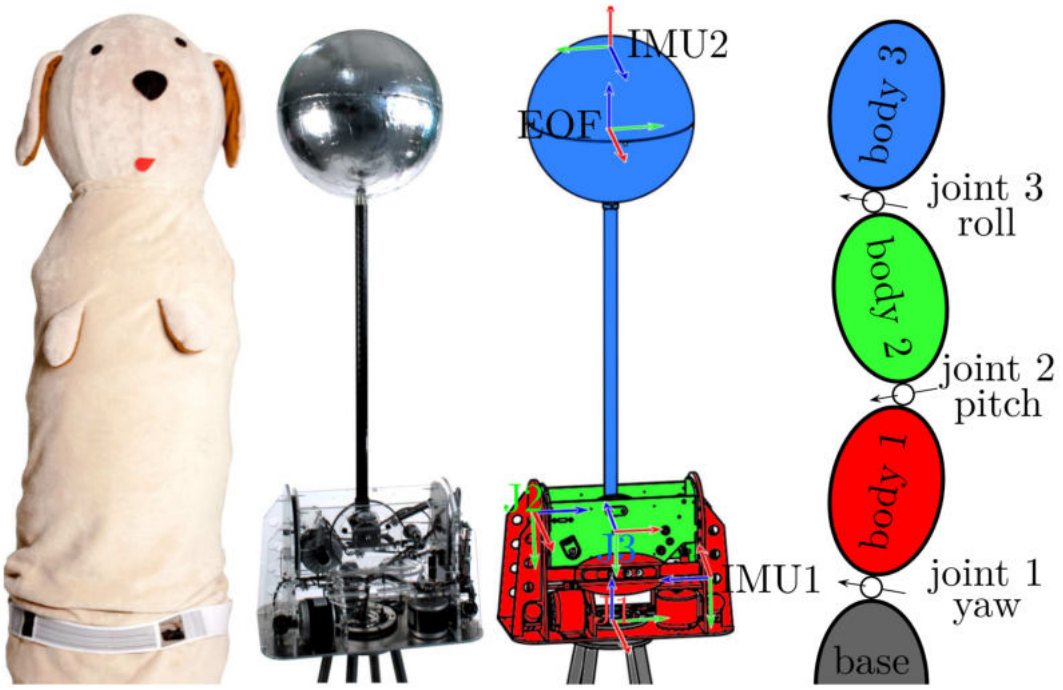

Abbildung 2.2: Unterhaltungsroboter Doggy v.l.n.r. in Verkleidung, mit sichtbaren Komponenten, das konstruierte CAD Modell, schematischer Aufbau. Gier-Achse (yaw, J1), die Nick-Achse (pitch, J2), Roll-Achse (roll, J3) mit zugehörigen Koordinatensystemen und Einfärbungen der zugehörigen Bauteile. Grafik von Schüthe, 2017 [19].

Abbildung 2.3: Unterhaltungsroboter Doggy in Aktion. Grafik von Schüthe, 2017 [19].

2.3 Convolutional Neural Network

Convolutional Neural Network bedeutet im Deutschen *gefaltetes neuronales Netzwerk* und weist auf die besondere Art der Verbindungen zwischen den Schichten des Netzwerks hin. Bei Eingabedaten in ein neuronales Netz sind herkömmliche voll- oder teilvermaschte neuronale Netze durch ihre Eigenart und den resultierenden Berechnungsaufwand stark in der Eingabegröße limitiert. Anstelle der Berechnung eines Neurons durch die Gewichtung aller Neuronen der vorherigen Schicht, setzt eine Convolutional-Schicht auf eine lokal-Vermaschung. Dabei wird die Lokalität der Vermaschung eines Neurons über einen Filter fester Größe bestimmt. Die Vermaschung der vorherigen Neuronen wird von den Filtern bestimmt, sodass ein Neuron nur Informationen eines lokalen Bereiches der vorherigen Schicht verarbeiten kann. Diese Abbildung der Daten auf ein Neuron wird als Faltung bezeichnet. Durch die alleinige Parametrisierung des Filters wird hier eine geringe Anzahl an Gewichten verwendet, die im Gegenzug zu vollvermaschten Netzen nicht mit der Datendimensionen skaliert. Jedoch wird das rezeptive Feld des Neurons lokal auf Bildbereiche begrenzt. Um ein einzelnes Neuron Informationen über das gesamte Eingabedatum tragen zu lassen, muss das rezeptive Feld des Neurons die gesamten Eingabedaten umfassen. Im Gegenzug zu vollvermaschten Netzen können Positionsinformationen erkannter Merkmale tief in das Netz getragen werden. Durch die Tiefe eines Convolutional neuronalen Netzes steigt die Informationsdichte, jedoch nimmt die Positionsresolution stark ab. Der Vorteil Positionen von Informationen und den Kontext der Informationen in den Bilddaten im neuronalen Netz nutzen zu können hat die Verarbeitung von Bildern stark vorangetrieben. 2012 setzte sich ein Convolutional Neural Network erstmals an die Spitze der Bildklassifizierungsalgorithmen. AlexNet, vorgestellt von Krizhevsky et al. 2012 [14], dominierte erstmals den Bildklassifizierungswettbewerb der ImageNet Challenge. Das AlexNet erreichte eine Fehlerrate die 10,8% niedriger war, als die des Zweitplatzierten. Seit 2012 dominieren CNN-basierte Ansätze die Wettbewerbe um Bildverarbeitungsaufgaben wie Klassifizierung oder Segmentierung.

2.4 U-Net

Das Segmentieren von Objekten auf Bildern erfordert neben der Klassifikation oder der Erkennung von Objekten auf Bildern zusätzlich deren Position und Größe auf dem Eingabebild. Ronneberger et al. [18] stellten 2015 das U-Net vor, eine besondere Netzarchitektur die sowohl das Identifizieren von Objekten, als auch die Rückführung dieser Information auf die Bildebene erfolgreich auf Basis von Convolutional-Schichten durchführt. Dazu werden hauptsächlich zwei Vorgänge konzipiert, eine kontrahierende, verengende Encoder-Struktur zur Informationskonzentration und eine expansive, ausdehnende Decoder-Struktur zur Rückrechnung konzentrierter Informationen auf die Eingabedaten. Da beide Strukturen konträre Ziele im Bezug zur Eingabedimension haben, werden sie oft in der namensgebenden Form des Buchstaben U visuell dargestellt. In Abbildung 2.4 ist die gesamte Architektur des vorgestellten U-Net abgebildet.

Encoder

Die Encoder-Struktur des U-Net von Ronneberger et al. für biomedizinische Bildsegmentierung folgt einer typischen Architektur für konventionelle Convolutional neuronale Netze und besteht aus mehrmaliger Anwendung von Blöcken bestehend aus zwei hintereinander gereihten ReLU-aktivierten Convolutional-Schichten mit 3x3 Filtergröße und einer abschließenden 2x2 Max-Pooling Schicht. Die Kanalzahl der Convolutional-Schichten verdoppelt sich nach jedem Block, wobei gleichzeitig die Auflösung der Feature-Maps beim Max-Pooling halbiert wird. Durch das faltende Verhalten der Convolutional-Schichten haben frühe Neuronen im Netz ein sehr kleines rezeptives Feld und können nur kleine, lokale Merkmale abbilden. Die

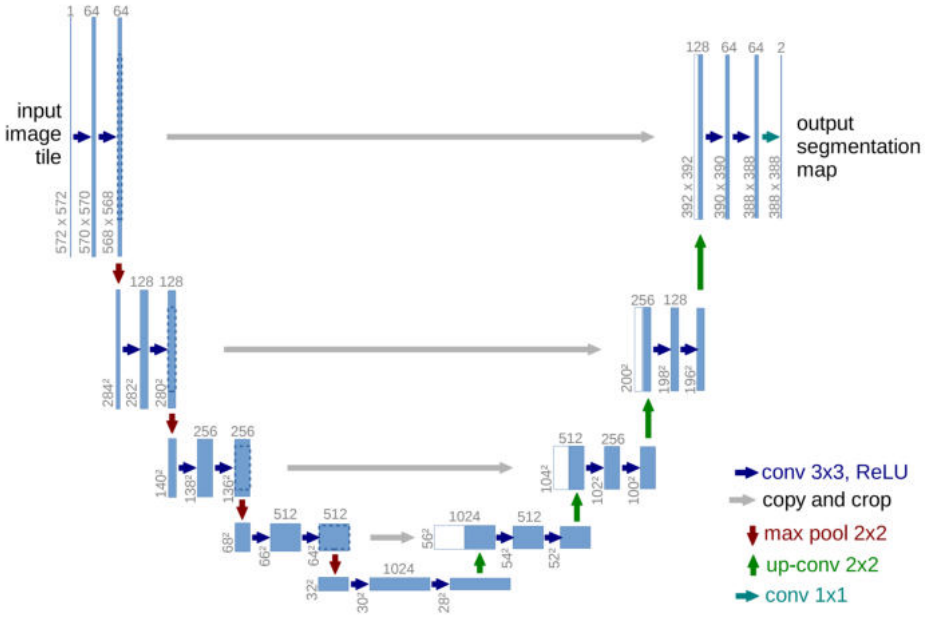

Abbildung 2.4: U-Net Architektur nach Ronneberger et al. [18]. Jede blaue Box stellt mehrere Kanäle einer Feature-Map dar. Die Anzahl an Kanälen ist über der Box angegeben. Die Feature-Map Dimension ist jeweils an der linken unteren Ecke angegeben. Weiße Boxen zeigen kopierte Feature-Maps an. Die Pfeile zeigen verwendete Operationen.

Aneinanderreihung der Schichten erweitert das rezeptive Feld späterer Neuronen, wodurch mit zunehmender Tiefe immer komplexere Strukturen abgebildet werden können.

Decoder

In der Decoder-Struktur des Netzes sollen die konzentrierten, komplexen Informationen der Encoder-Struktur zurück auf die Eingabedaten gerechnet werden, um dort eine Aussage über die Zugehörigkeiten oder Abgrenzungen der Bildteile zu treffen. Ähnlich wie beim Encoder werden die Informationen durch die Anwendung von gleichen Blöcken verarbeitet. Innerhalb eines Blocks werden die Daten zuerst um den Faktor 2 hochskaliert. Anschließend wird von einer 2x2-Convolutional-Schicht die Kanalzahl halbiert. Damit wurde, genau umgekehrt zur Encoder-Struktur, die Auflösung vergrößert und die Menge an Kanälen verringert. Zu diesen Daten werden dann die extrahierten Informationen aus dem Encoder des korrespondierenden Blockes gleicher Auflösung hinzugefügt. Danach folgen innerhalb des Blockes, genau wie bei der Encoder-Struktur, zwei ReLU-aktivierte 3x3-Convolutional-Schichten. Nach mehrmaliger Anwendung dieser Blöcke passen die Feature-Maps von den Dimensionen zu den Eingabedaten, sodass eine letzte 1x1-Convolutional-Schicht die Features auf beliebig viele Klassen zusammenfügt. Die Querverbindungen von Daten aus dem Encoder zum Decoder in jeder Ebene werden als Abkürzungs- oder Shortcut-Verbindungen bezeichnet. Gefundene, komplexe Merkmale im Netz können sich nur aus gefundenen simpleren Strukturen ergeben. Die Shortcut-Verbindungen liefern dem Decoder hochaufgelöste, klare Strukturen und die hochskalierten Daten liefern die Bereiche der gesuchten Objekte. Dadurch kann der gesamte Decoder eine hohe Informationsdichte zusammen mit einer hohen Auflösung durch das Netz propagieren. Eine letzte Ausgabeschicht erzeugt eine passende Maske zum Eingabebild auf der eine prozentuale Zugehörigkeit jedes Pixels zu den Klassen dargestellt ist.

2.5 Splines

Ein Spline ist eine kontinuierliche Funktion, die zur Interpolation von Datenpunkten in einem bestimmten Funktionsraum verwendet wird. Diese Funktion ist eine stückweise Polynomfunktion, bei der jeder Abschnitt des Funktionsraums durch ein separates Polynom mit einer vorgegebenen Gradstufe beschrieben wird. Die Übergänge zwischen den Abschnitten sind so gestaltet, dass eine kontinuierliche Funktion erzeugt wird. In welcher Form die Übergänge zwischen den Funktionsräumen gestaltet werden, definiert die Kontinuität des Splines. Verschiedene Spline Verfahren besitzen verschiedene Kontinuitäten und setzen meist eine vorab definierte Anzahl an Knotenpunkten voraus, an denen die Polynome aneinandergereiht werden. Neben den am häufigst genutzten Bézier Splines [10] gibt es noch weitere Splines wie z.b. den Hermite Spline [17], den Catmull-Rom Spline [4] oder den B-Spline [7] mit verschiedensten Eigenschaften und anwendungsbezogenen Vorteilen. Da in dieser Arbeit ausschließlich Bézier Splines verwendet werden, werden diese genauer erläutert.

2.5.1 Bézier Splines

Eine quadratische Bézierkurve kann durch den Prozess der linearen Interpolation zwischen drei Kontrollpunkten P_1, P_2 und P_3 definiert werden. Der Parameter t bestimmt die relative Lage eines Punktes auf einer geraden Linie zwischen zwei Kontrollpunkten. Durch Interpolation zwischen zwei geraden Linien, die durch jeweils zwei Kontrollpunkte definiert werden, kann ein Punkt P berechnet werden, der auf der quadratischen Bézierkurve liegt. Abbildung 2.5 zeigt die Interpolation und die sich ergebene Kurve.

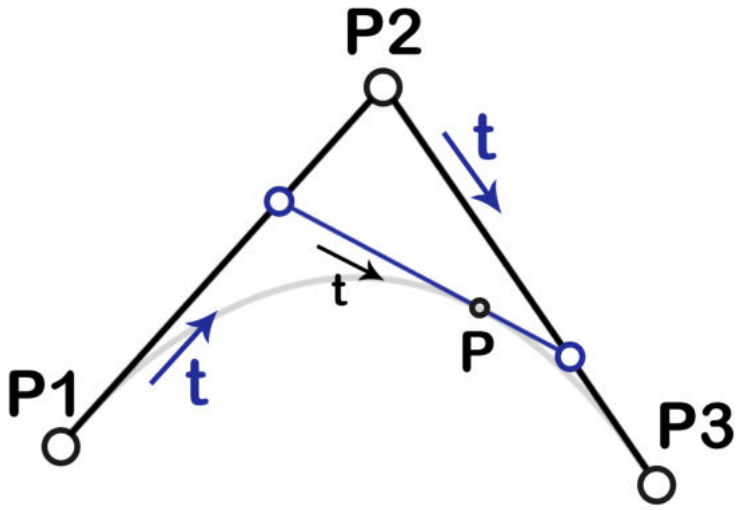

Abbildung 2.5: Eine Interpolation mit den Kontrollpunkten P_1, P_2 und P_3. Durch die Interpolation zwischen den Punkten P_1-P_2 und P_2-P_3 mit dem Parameter t und einer erneuten Interpolation zwischen den sich ergebenen Punkten erhält man den Punkt P auf der quadratischen Bézierkurve definiert durch die Kontrollpunkte.

Eine kubische Bézierkurve definiert sich durch 4 Kontrollpunkte zwischen denen die Lineare Interpolation im gleichen Verfahren auch einen einzigen Punkt P in Abhängigkeit von t definiert. Dieser Punkt liegt auf der kubischen Bézierkurve und verläuft im Intervall $[0, 1]$ genau durch den ersten und letzten Kontrollpunkt. Besonders hervorzuheben ist die

Eigenschaft von Bézierkurven, dass die Kurve an den Intervallgrenzen genau tangential zur Geraden zwischen dem ersten und zweiten bzw. dem dritten und vierten Kontrollpunkt verläuft. Abbildung 2.6 zeigt die Interpolation zwischen den 4 Kontrollpunkten P1 bis P4, auf denen jeweils interpolierte Punkte liegen. Interpoliert man zwischen diesen Punkten ergeben sich neue Punkte zwischen denen interpoliert wird. Jeweils erfolgt die Interpolation mit dem Parameter t.

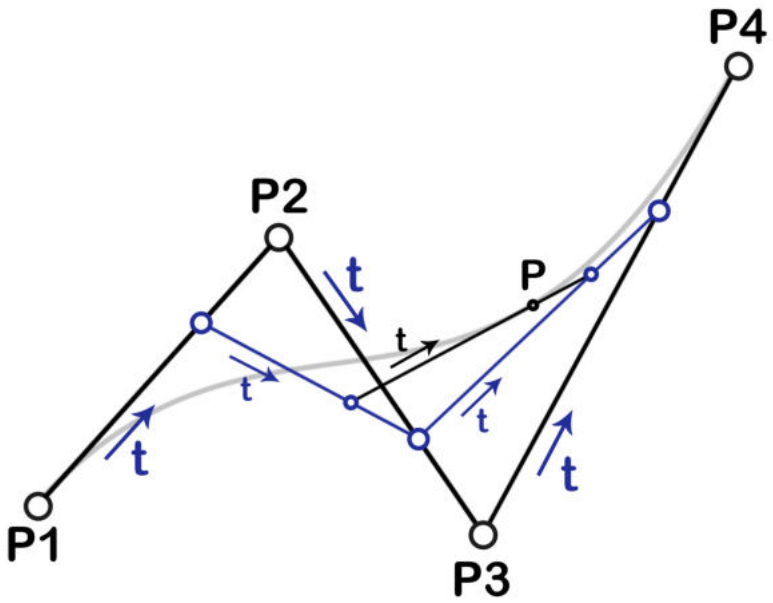

Abbildung 2.6: Eine Interpolation mit den Kontrollpunkten P_1, P_2, P_3 und P_4. Durch die Interpolation zwischen den Punkten P_1-P_2, P_2-P_3 und P_3-P_4 mit dem Parameter t und einer Interpolation zwischen den sich ergebenen Punkten erhält man zwei weitere Punkte. Durch eine Interpolation zwischen diesen Punkten ergibt sich Punkt P, der auf der quadratischen Bézierkurve liegt, die sich durch die Kontrollpunkte definiert.

Diese rekursive lineare Interpolation zwischen den Kontrollpunkten ist bekannt als DeCasteljau Algorithmus und wurde Anfang der 1960er Jahre von Paul de Faget de Casteljau entwickelt. Dabei wird so lange zwischen den Punkten interpoliert, bis sich nur noch ein einziger Punkt ergibt. Bei Bézierkurven n-ter Ordnung werden damit $n-1$ Ebenen der Rekursion benötigt. Das Faktorisieren der Kontrollpunkte innerhalb dieser Berechnungen ergibt die Bernstein Form (2.1) der Bézier Kurve. Hier ist für jeden Kontrollpunkt der Kurve in Abhängigkeit von t ein Polynom angegeben, dass den Einfluss des Kontrollpunktes auf die Kurve beschreibt. Diese Polynome sind auch als Bernsteinpolynome n-ten Grades bekannt. Eine weitere Darstellungsform der Bézier Kurve ist die Matrixform in Formel 2.2. Die Kontrollpunkte sind isoliert von den numerischen Faktoren und dem Parameter t. Diese Form zeigt die Möglichkeit der effizienten Berechnung, da Berechnungen vorher durchgeführt und immer wiederverwendet werden können. Nach Umstellen der Terme nach dem Parameter t ergibt sich außerdem eine weitere Form der Darstellung, die Polynomialkoeffizienten Darstellung (2.3). Jede dieser Darstellungen hat Vor- und Nachteile und dient in verschiedenen Kontexten zur besseren Veranschaulichung der Zusammenhänge.

$$\begin{aligned} P = {} & P_1(-t^3 + 3t^2 - 3t + 1) + \\ & P_2(3t^3 - 6t^2 + 3t) + \\ & P_3(-3t^3 + 3t^2) + \\ & P_4(t^3) \end{aligned} \tag{2.1}$$

$$P(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \tag{2.2}$$

$$\begin{aligned} P = {} & P_1 + \\ & t(-3P_1 + 3P_2) + \\ & t^2(3P_1 - 6P_2 + 3P_3) + \\ & t^3(-P_1 + 3P_2 - 3P_3 + P_4) \end{aligned} \tag{2.3}$$

Ein kubischer Bézier Spline ist eine Aneinanderreihung mehrerer Bézierkurven. Die Kontinuität eines Splines beschreibt die Eigenschaften aller Übergänge der Kurven innerhalb des Splines und kann in verschiedene, hierarchisch angeordnete Klassen C^n eingeteilt werden. Dabei besitzt jede Klasse C^n auch die Eigenschaften der vorangegangenen Klasse C^{n-1}. C^0 Kontinuität beschreibt die allgemeine Kontinuität eines Splines. Damit sind Endpunkte aneinanderliegender Splines an den gleichen Positionen. C^1 beschreibt die Kontinuität der ersten Ableitung der Kurven. Damit sind die Steigungen aneinander grenzender Kurven an den Übergängen gleich. C^2 ist die Kontinuität der zweiten Ableitung, der Beschleunigung der Kurven, an den Kurvenübergängen. Im gleichen Muster beschreiben weitere Kontinuitäten die Konsistenz der Änderung der entsprechenden Kurvenübergängen.

3 Ballerkennung

Das Ziel der hochperformanten Ballerkennung besteht darin, einen geworfenen Ball während seiner Flugphase auf monochromen Bildern genau und zuverlässig zu erfassen. Hierfür liefert ein Stereokamerasystem mit einer Frequenz von 50 Hz monochrome, parallel aufgenommene Bilder deren Belichtungszeit als auch die Blendenöffnung je nach Einsatzgebiet stark variieren. Sobald ein Ball im Aufmerksamkeitsfeld des Roboters in der Luft ist, soll der Ball im 3D Raum erfasst und während der gesamten Flugphase verfolgt werden. Basierend auf den Messungen wird eine Flugtrajektorie berechnet und durch weitere Messungen während des Fluges korrigiert.

Eine Ballerfassung ist die Erkennung eines geworfenen Balles in Position und Radius in Pixelwerten auf den Kamerabildern. Dafür wird ein neuronales Netz trainiert, dass fliegende Bälle auf den Kamerabildern maskiert und zusammen mit einem iterativen Algorithmus einen entsprechenden Kreis in Position und Radius pro Ball erzeugt. Besonders hervorzuheben ist dabei, dass dieses Training ohne reale Bilddaten stattfindet und allein auf synthetisch generierten Daten für die Erfüllung des Zielszenarios lernt.

3.1 Methodik

Grundsätzlich besteht die Problemstellung in der hocheffizienten Lösung einer semantischen Segmentierungsaufgabe durch Trennung von Vordergrund und Hintergrund. Der Vordergrund besteht aus einem Ball mit Fussball-typischem Schwarz-Weiß Hexagon Muster und einer leicht reflektierenden, glatten Oberfläche. Der Hintergrund ist, je nach Anwendungsgebiet des Roboters, verschieden und hat immer eine unterschiedliche Belichtung. Außerdem können Extrembelichtungen sowohl im Vordergrund, als auch im Hintergrund auftreten und mit direkter Sonneneinstrahlung oder Schattenbereichen, die durch Hindernisse verursacht werden, Teile des Bildes überstrahlen oder verdecken. Auch die Bewegung und Rotation des Balles, besonders bei hohen Belichtungszeiten in Kombination mit hoher Fluggeschwindigkeit, kann Bewegungsverzerrungen und -unschärfe ins Erscheinungsbild des Balles bringen. Ein neuronales Netz soll geworfene, relevante Bälle auf den Eingabedaten segmentieren und ein Konturenerkenner die segmentierten Bälle in Position und Größe letztendlich erkennen. Durch die vielen verschiedenen Einsatzszenarien des Roboters ist es für das Netz besonders wichtig eine robuste Leistung zu gewährleisten, um den Ball im Vordergrund mit der hohen Variabilität im Hintergrund erfolgreich zu erkennen. Um eine ausreichende Abdeckung dieser Varianten zu gewährleisten, muss der Datensatz aus einer großen Vielfalt an unterschiedlichen Szenarien generiert werden. Hierfür wird eine Zusammensetzung aus Vordergrund- und Hintergrundbildern vorgenommen, die einen möglichst großen Anwendungsbereich abdeckt.

3.1.1 Synthetische Trainingsdaten

Neben der Möglichkeit der hohen Variabilität synthetischer Trainingsdaten für neuronale Netze bringt diese Methode weitere Vorteile mit sich. Mit einer hinreichenden Menge an Variablen für die Datengenerierung ist der Raum erzeugbarer Trainingsdaten nahezu unendlich. Durch die Synthetisierung der Trainingsdaten können komplexe oder unbekannte Zielszenarien für die Anwendung generiert werden. Ein Generierungsprozess ist immer wiederholbar und spart gegenüber einer manuellen Trainingsdatenerzeugung besonders

auf lange Sicht Ressourcen. Teile des Generierungsprozesses können je nach Anwendung ersetzt oder erweitert werden, sodass die initiale Investition über die Wiederverwendung verschwindend gering wird. Darüber hinaus kann durch die Kontrolle der Variablen eine Reduktion von Bias und Varianz im Modell erzielt werden und dazu eine Überanpassung des Modells an die Trainingsdaten vermieden werden.

Mit Blick auf die Präzision eines Modells liefert ein Generierungsprozess synthetischer Trainingsdaten, aufgrund der genauen Kontrolle der Variablen, immer die genauste Grundwahrheit. Das Training von Modellen auf echten Daten ist immer durch die erbrachte Annotationsleistung limitiert. Fehler oder Inkonsistenzen beim Erzeugen der Trainingsdaten haben nach Enderes 2021 [9] verschiedene Auswirkungen auf die Genaugigkeit von trainierten Modellen. Fard et al. erkennen 2017 [20], dass besonders CNN sehr robust gegen unvoreingenommene Fehlkennzeichnungen, also nahezu zufällige Verwechslungen bei der Erstellung der Grundwahrheit, sein. Hingehen sollen voreingenommene Fehlkennzeichnungen, konstante Verwechselung ähnlicher Klassen oder konstante Fehler schnell einen nahezu linearen Fehler in der Menge der falschen Zuordnungen verursachen. Diese Problematiken sind große limitierende Faktoren beim Training von datengetriebenen Lernverfahren, treten bei einem automatisierten Generierungsprozess jedoch nicht auf.

Vordergrund

Für die Synthese von geworfenen Bällen ist ein 3D Modell eines Fussballs in der Modellierungsplattform Blender [6] erstellt worden. Ein Ball aus dem Zielszenario stand zur Verfügung, sodass Maße und Materialeigenschaften ähnlich übertragen werden konnten. Um eine genaue Kontrolle bei der späteren Komposition der Trainingsdaten zu behalten werden die Vordergrundbilder als 256x256 Pixel Große Patches mit transparentem Hintergrund aus der Blender Engine gerendert. Dabei ist die Position bei konstanter Größe genau im Zentrum festgelegt, sodass sich beim Erzeugen mehrerer Bilder nur die Beleuchtung und Rotation des Balles ändern. Die Bälle besitzen auf den Patches eine feste Größe von 64 Pixeln im Radius. Die Beleuchtung der Szene besteht aus mehreren Lichtern mit verschiedenen Intensitäten. Beim Generierungsprozess der Bilder nehmen die Lichter zufällige Positionen relativ zum Ball ein und der Ball eine zufällige Rotation. Diese Konfigurationen können eine große Menge an unterschiedlichen Momentaufnahmen eines Balles mit verschiedenen Lichtquellen und Beleuchtungsrichtungen erzeugen. Zusätzlich können Extrembelichtungen jeglicher Art erzeugt werden. Die Grundwahrheit der Bälle ergibt sich durch den Alphakanal der gerenderten Patches. Dieser beinhaltet in jedem Pixel die Sichtbarkeit des Balles.

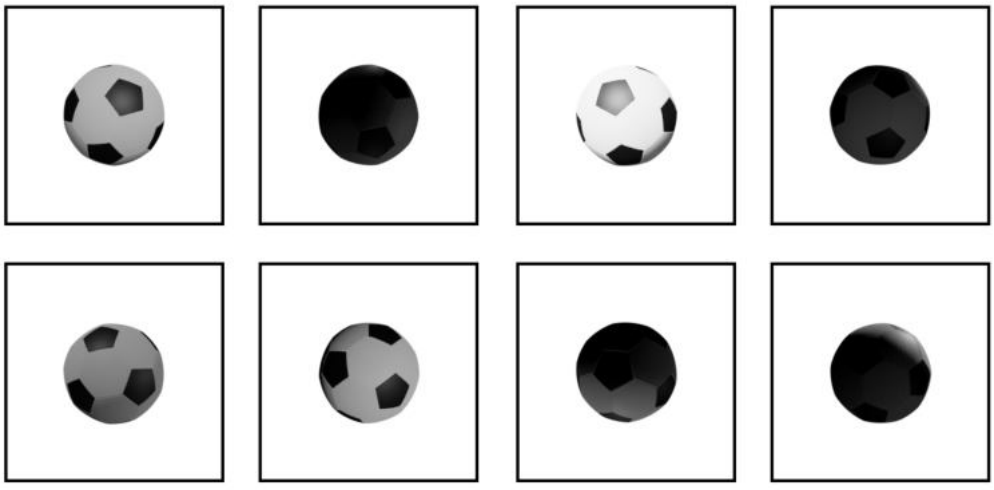

Abbildung 3.1: Auswahl von Ballpatches für den Vordergrund der Komposition.

Hintergrund

Der Hintergrund besteht aus künstlich erzeugten Bildern verschiedener *text2image* KI Tools. Sie zeigen eine offene Fläche im Vordergrund und geometrische Formen wie Gebäude, Wände oder Räume oder freie Sicht auf den Himmel im Hintergrund. Sie weisen keine besonderen Merkmale hinsichtlich der Lichtverhältnisse auf, da diese bei der Komposition der Trainingsdaten hinzugefügt und kontrolliert werden. Die ausgewählten Bilder wurden auf die Größe 768x576 Pixel skaliert, sodass sie das gleiche Format wie die Kamerabilder aus dem Zielsystem aufweisen. Da die ausgewählten Bilder zum Großteil einen RGB Farbraum aufweisen, das Zielszenario allerdings nur monochrome Bilder verarbeitet, nutzen wir die verschiedenen Farbkanäle als weiteren Faktor zur Vergrößerung der Varianz der Trainingsdaten.

Abbildung 3.2: Auswahl von Hintergrundbildern die mit *text2image* KI-Tools generiert wurden. Sie sollen einen möglichst großen Raum aus Umgebungen des Zielszenarios abbilden.

Komposition

Vor der Zusammensetzung eines Trainingsdatums aus Vorder- und Hintergrund werden verschiedene Transformationen auf den Bildern angewendet. Die hohe Variabilität der Trainingsdaten wird in diesem Schritt hergestellt und der Merkmalsraum endgültig aufgespannt. Unschärfen und andere Artefakte des Zielszenarios werden hier zusätzlich hinzugefügt um

die Trainingsdaten den realen Daten anzugleichen. Dies dient dazu, das erlernte Modell robust gegenüber Veränderungen im Bildinhalt und Bildumfeld zu machen. Wichtig ist dabei zu erwähnen, dass Ball-Patch und Hintergrundbild getrennt voneinander transformiert werden, da im Gegensatz zur identischen Transformation die Unabhängigkeit die Varianz verstärkt.

Der Ball-Patch wird so skaliert, dass ein Ball einen Radius von maximal 32 Pixel und mindestens 6,4 Pixel aufweist. Durch die Veränderung der Größe des Patches wird ein Abstand zur Kamera simuliert. Der Patch wird zufällig bis zu 360 Grad gedreht. Ein gaußscher Weichzeichner wird verwendet, um eine allgemeine Unschärfe verschiedener Stärke auf den Ball zu bekommen. Dieser dient zur Nachahmung von Änderungen in der Rotation des Balles während der Belichtungszeit. Eine weitere, gerichtete Unschärfe mit zufälliger Stärke und Richtung stellt eine Bewegungsunschärfe dar. Diese ist in realen Daten durch große Änderungen in der Position verursacht. Zuletzt werden die Grauwerte des Patches mit einem variablen Faktor zwischen 0,8 und 1,5 multipliziert und bei dem maximalen Grauwert begrenzt. Dadurch können leichte Helligkeits- als auch Kontrastunterschiede und zusätzliche Extrembelichtungen erzeugt werden.

Die Grundwahrheit dieses Patches wird lediglich mit einer sehr leichten gaußschen Unschärfe belegt und dann mit skaliert. Das liegt daran, dass alle anderen Transformationen nicht die Grundwahrheit der Position des Balls im Raum verändern, sondern nur die visuelle Erscheinung im System. Diese Verzerrungen werden durch systematische Faktoren und physikalische Einschränkungen hervorgerufen.

Das Hintergrundbild wird ebenfalls vor der Zusammensetzung transformiert. Eine zufällige, quadratische gaußsche Unschärfe soll den Hintergrund weichzeichnen. Die Grauwerte werden mit einem Faktor zwischen 0,8 und 2,3 multipliziert und zwischen 0 bis 15 Prozent aufgehellt. Diese Parameter spannen die Trainingsdaten weit auf, führen dabei allerdings durchaus zu vergleichbaren Extrembedingungen, wie sie auf dem aktuellen Robotersystem auftreten. Zuletzt wird das Hintergrundbild zwischen -15 Grad und +15 Grad gedreht und bis zu 1,8x vergrößert. Diese Transformationen sind bei der Bildgröße sehr rechenintensiv und werden beim Training nur mit einer Wahrscheinlichkeit von 33 Prozent pro Trainingsdatum durchgeführt. Bei der Zusammensetzung des Trainingdatums wird darauf geachtet, dass der zu erkennende Ball immer vollständig im Bild platziert ist. Abbildung 3.3 zeigt ein fertiges Trainingsdatum aus der Komposition. Dabei wurde das Hintergrundbild in Graustufen konvertiert, leicht im Uhrzeigersinn gedreht und vergrößert. Der Kontrast des Hintergrunds wurde angepasst und die allgemeine Helligkeit erhöht. Der Ball wurde skaliert auf einen Radius von 11 Pixeln, gedreht und eine gaußsche Unschärfe von 3 Pixeln Größe. Eine Bewegungsunschärfe in diagonaler Richtung mit 6 Pixeln Größe ist ebenfalls dazu gekommen. Der Ball selber ist zusätzlich heller mit etwas weniger Kontrast. In Abbildung 3.4 ist ein weiteres Trainingsdatum dargestellt mit einer Vergrößerung des Ballbereichs und einer Visualisierung der Grundwahrheit.

Nachverarbeitung

Durch die Konstruktion der Ausgabeschicht des verwendeten neuronalen Netzes, einer Sigmoid-Aktivierungsschicht, befindet sich die Segmentierungsinformation in den jeweiligen Pixelwerten des Ausgabebildes. In jedem Pixel steht die Wahrscheinlichkeit, zu der dieser Pixel zu einem Ball gehört. Die Ränder und das Umfeld eines segmentierten Balles weisen damit keine harten Grenzen sondern Abstufungen auf. Um einen erkannten Ball aus der Ausgabeschicht des neuronalen Netzes zu extrahieren sind damit weitere Schritte notwendig um Mittelpunkt und Größe zu bestimmen.

Zur Trennung zwischen erkanntem Vordergrund und Hintergrund werden die Grauwerte mit einem Grenzwert binärisiert. Ein Algorithmus zum Extrahieren von Konturen auf einem Bild liefert die Konturen aller vorhergesagten Bereiche auf dem Bild. Ein weiterer

Abbildung 3.3: Ein beispielhaftes fertiges Trainingsdatum aus der Komposition. Der Hintergrund ist in Grautönen, wurde gedreht und vergrößert, der Kontrast des Bildes wurde zusammen mit der Helligkeit erhöht, der Ball hat eine zufällige Rotation, eine Bewegungsunschärfe in diagonaler Richtung und eine generelle Unschärfe.

Algorithmus *Smallest enclosing disks (balls and ellipsoids)* von Welzl [23] liefert zu einer Menge an Punkten die Position und den Radius des kleinsten umschließenden Kreises. Wird der größte Bereich eines Balles korrekt detektiert, sodass sich die Form des korrekten Kreises trotzdem abzeichnen lässt, liefert der angewandte Algorithmus trotzdem die korrekte Position und Größe des Balles.

Abbildung 3.4: Ein beispielhaftes fertiges Trainingsdatum aus der Komposition. Der Hintergrund ist in Grautönen, wurde gedreht und vergrößert, der Kontrast des Bildes wurde zusammen mit der Helligkeit erhöht, der Ball hat eine zufällige Rotation, eine Bewegungsunschärfe in diagonaler Richtung und eine generelle Unschärfe.

3.2 Implementierung

Für die Implementierung des neuronalen Netzes werden die Frameworks TensorFlow 2 [1] und Keras [5] in der Programmiersprache Python verwendet. Zusammen erlauben sie eine schnelle und einfache Implementierung eines neuronalen Netzes und bieten weitere Funktionen um sowohl den Trainingsprozess, als auch die Datengenerierung effizient zu gestalten.

3.2.1 Neuronales Netz

Das neuronale Netz wurde als U-Net [18] implementiert und besteht grundsätzlich aus 2D-Convolution Blöcken mit Batchnormalisierungs- und ReLU-Aktivierungsschichten. Zwischen den Blöcken werden Shortcut-Verbindungen zwischen Decoder- und Encoderstufe eingefügt und die Feature-Dimension über ein MaxPooling konzentriert. Eine Besonderheit bei der Implementierung ist die Umsetzung einer frühen, biologisch inspirierter ON/OFF ReLU Aktivierung einer Convolution Schicht. Kim et al. [13] stellten 2015 eine doppelte Nutzung der Filter in einer Convolution Schicht vor. Die Filter der ersten Ebene werden einerseits durch eine konventionelle ReLU als auch durch eine negative ReLU Funktion aktiviert. So können sowohl die gelernten Strukturen der Filter als auch ihre negativen Gegenstücke genutzt werden. Die Ergebnisse von Kim et al. zeigen in Experimenten eine Steigerung der Genauigkeit in Modellen von 1,72% und einer schnelleren Lernrate gegenüber eines konventionellen Modells. Mit einer Filtergröße von 7x7 in der ON/OFF-ReLU aktivierten ersten Convolution Schicht können diese Filter abseits von simplen Strukturen auch komplexere Zusammenhänge lernen und ihre negativen Gegenstücke

werden durch die Aktivierung erzeugt und genutzt. Alle anderen Convolution Schichten im Modell lernen 3x3 Filter und sind mit konventionellen ReLU-Schichten aktiviert. Im Encoder werden die Feature Maps in 3 Stufen bis auf eine Größe von 192x144 Pixel verringert und im Decoder werden erzeugte Features wieder auf die Eingabegröße von 768x576 Pixel, jeweils mit den Shortcut Informationen, vergrößert.

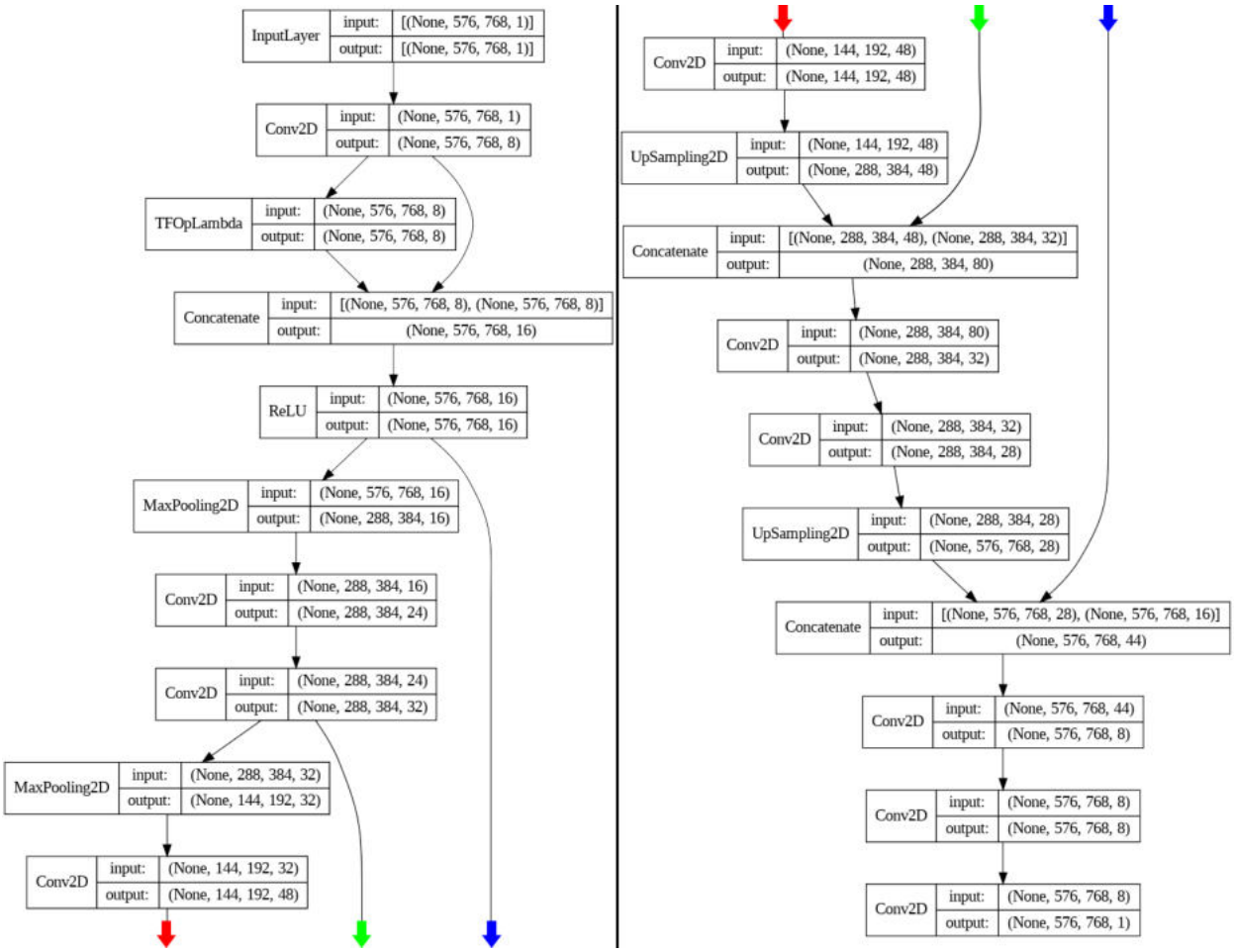

Abbildung 3.5: Verwendetes Netz für das Ballerkennungssystem. Mit Eingabeschicht oben links und Ausgabe unten rechts. Farbige Pfeile markieren seitenübergreifenden Datenfluss zur besseren Visualisierung.

3.2.2 Training

Für das Training des neuronalen Netzes ist ein Datengenerator in Keras implementiert worden, der die untransformierten Hintergrund- und Vordergrundbilder als Eingabe bekommt. Mittels der Komposition und den Tranformationen können Trainingsbilder mit großen Varianzen zusammengesetzt werden. Das Training des neuronalen Netzes kann, dank Keras, direkt mit dem Generator gestartet werden. Dabei sind Funktionen in dem Generator implementiert worden, sodass unterschiedliche Teilschritte der Komposition modifiziert und von außen gesteuert werden können. Dies erlaubt das Training mit unterschiedlichen Modi um in verschiedenen Phasen des Trainings eine genaue Kontrolle über die Eingabedaten zu haben.

Die gesamte Komposition findet während des Trainings im Generator Batchweise statt und verlangsamt das Training um ein Vielfaches. Da nach einem Training auf einem Batch nicht das gesamte Potential dessen aufgebraucht ist, ist eine sehr einfache Opti-

mierung das Wiederverwenden des Batches in kommenden Epochen. Grundsätzlich kann ein neuronales Netz von den gleichen Trainingsdaten mehrere Epochen lang lernen. Das Potential eines Samples in den Trainingsdaten ist verloren, sobald ein Overfitting auf den Daten stattfindet. Dabei lernt das Netz einen Zusammenhang zwischen Grundwahrheit und irrelevanten, zufälligen Informationen der Eingabedaten. Die Leistung des Systems wird auf den Trainingsdaten besser, verschlechtert sich aber auf der eigentlichen Aufgabe. Die Effektivität von Trainingsprozessen mit einer konstanten Menge an Trainingsdaten kehrt sich mit dem Beginn des Overfitting ins Negative. Die Menge der Trainingsdaten bietet in diesem Prozess kein Verbesserungspotential mehr. Ein guter Generierungsprozess von Daten wirkt diesem Problem mit der Variabilität und der Varianz der Daten entgegen und sorgt für die stetige Generierung von Verbesserungspotential. Das Optimieren des Prozesses mit der Wiederverwendung von Trainingsdaten ist eine sehr speicherintensive Optimierung, da alle Batches einer Epoche gleichzeitig im Speicher gehalten werden. Je nach Aufwand des Generierungsprozesses kann diese Optimierung große Zeitersparnisse beim Training mit sich bringen. Mit der seriellen Implementation des gesamten Generierungs- und Trainingsprozesses, zusammen mit der Verwendung großer Bilddaten und der Nutzung komplizierter Verzerrungsoperationen, spart diese Optimierung bei einmaliger Wiederverwendung der Trainingsdaten ein Drittel der Trainingsdauer gegenüber der wiederholten Neugenerierung. In diesem Training skaliert die Einsparung theoretisch asymptotisch gegen $\frac{2}{3}$, in der Realität ist eine Einsparung von ungefähr 50% realisierbar. Die sinnvolle Anzahl an Wiederverwendungen ist dabei der einschränkende Faktor.

3.3 Experimente

Zur Evaluation der entwickelten Erkennungsmethode von Bällen werden mehrere Experimente in qualitativer- und quantitativer Form durchgeführt. Ein händisch annotierter Datensatz von Bildern aus dem Zielszenario dient als Evaluationsgrundlage für die Bewertung der Erkennungsleistung.

3.3.1 Datensatz

Die Evaluation des Systems am echten Roboter ist durch weitere Forschungsarbeiten an *Doggy* während der Evaluationsphase dieser Thesis leider nicht gegeben. Zu Beginn wurden jedoch Datensätze von geworfenen Bällen am Roboter aufgezeichnet mit denen eine rein visuelle Evaluation auf einem anderen System möglich ist. Der Datensatz besteht aus 1225 Bildern auf denen 1223 Bälle in Mittelpunktposition und Radius von Hand annotiert wurden. Da beide Werte händisch als Ganzzahlen annotiert wurden, das entwickelte Erkennungssystem aber eine Erkennung im präziseren Zwischenpixelbereich liefert, ist hier bei der Auswertung ein gewisser Fehler zu gewähren. Der Datensatz beinhaltet ganze Würfe des Balles von einem Menschen aus unterschiedlichsten Positionen und Distanzen an drei verschiedenen Orten. Grundsätzlich wurden Szenarien mit unterschiedlichen Belichtungen gewählt, wobei verschiedene Extrembelichtungen des Balles und des Hintergrundes vorkommen.

3.3.2 Erkennung

Das neuronale Netz für die Erkennung wurde spezifisch für das Zielszenario trainiert, um eine möglichst frühe und genaue Erkennung eines gerade los geworfenen Ball zu liefern. Die Annotation des Datensatzes ganzer Würfe enthält Bälle, die nicht mehr relevant für die Erkennung und damit für die Verarbeitung im System sind. Die Sichtung der Daten zeigt, dass der relevante Erkennungsbereich auf den Bildern bei einem Ballradius von 7 Pixel anfängt und bei 16 Pixel aufhört. Ein kleinerer Radius des Balles stellt entweder

eine zu große Entfernung des Balles zum Robotersystem dar oder würde mit typischen Geschwindigkeiten eine spätere Erkennung bei kleinerer Distanz zum Roboter erlauben. Bälle mit einem größeren Radius als 16 Pixel stellen eine zu kleine Entfernung zu den Kameras dar und sind damit auch für die Erkennung und Verarbeitung im System irrelevant. Meist sind diese Bälle am Roboter abprallend oder ganz am Bewegungsradius des Roboters vorbei geworfen. Mit dieser Einschränkung bleiben im annotierten Datensatz von den 1223 Bällen noch 805 relevante Bälle für die Erkennung übrig.

In Abbildung 3.6 ist eine Auswahl von Bildern, die Bälle zeigen, die relevant für die Erkennung sind. Sie weisen teilweise Bewegungsunschärfe und Überblendungen auf und haben einen Radius von 7 bis 16 Pixel. Hingegen zeigt Abbildung 3.7 eine Auswahl an annotierten Bällen, die nicht relevant für eine Erkennung sind. Diese wurden zur Vollständigkeit des Datensatzes trotzdem annotiert, sind aber irrelevant für das Gesamtsystem. Sie zeigen abprallende oder vorbei geworfene Bälle und weisen einen Radius über 16 Pixel auf.

Abbildung 3.6: Auswahl von aufgenommenen Bildern mit relevanten Bällen für die Erkennung aus dem Zielszenario.

Um die Leistung des Netzes auszuwerten, wird eine Fehlertoleranz definiert, die zwischen erfolgreicher Erkennung und Falscherkennung differenziert. Eine Abweichung von $\frac{r}{5}$ als euklidische Distanz zwischen dem Mittelpunkt des annotierten Balls und erkanntem Ball mit r als Radius des annotierten Balles modelliert die Wichtigkeit einer genauen Erkennung in größerer Distanz. Neben dem Positionsfehler wird auch die Toleranz des erkannten Radius mit einer Abweichung von $\frac{r}{5}$ modelliert. Wird ein Ball erkannt und liegt innerhalb dieser beiden Toleranzen, gilt er als erfolgreich erkannt. Damit ergibt sich eine variable Toleranzgrenze von 1,4 Pixeln bei den kleinsten erkannten Bällen von 7 Pixeln im Radius, bis hin zu einer Toleranz von 3,2 Pixeln bei den größten erkannten Bällen mit 16 Pixeln Radius.

Abbildung 3.7: Auswahl von aufgenommenen Bildern mit irrelevanten Bällen für das Gesamtsystem.

3.3.3 Qualitative Auswertung

Für die qualitative Auswertung der Ergebnisse wurden Beispiele aus dem Datensatz gesucht, bei denen Besonderheiten hervorgehoben werden.

Experiment 1: Leicht erkennbarer Ball mit Bewegungsunschärfe

Ein gerade los geworfener Ball besitzt auf diesem Bild eine relativ hohe Geschwindigkeit. Auf dem aufgenommenen Bild ist um den Ball in Bewegungsrichtung eine Verzerrung in Form von Bewegungsunschärfe zu sehen. Das Netz erkennt den runden Ball korrekt genau in der Mitte der visuellen Erscheinung. Dies ist ein perfektes Beispiel für die Erkennungsleistung bei leicht erkennbaren Bällen. Keine Extrembelichtung überblendet Bildbereiche um den Ball herum. Das Muster des Balls ist klar zu erkennen, da keine zu große Unschärfe auftritt. Die Bewegungsunschärfe ist ebenfalls nicht zu stark, sodass die Mitte der Erscheinung als Ball detektiert werden kann. Dargestellt ist die Erkennung in Abbildung 3.8.

Experiment 2: Schwer erkennbarer Ball mit Überblendung

Der geworfene Ball ist seit einiger Zeit in der Luft und nähert sich seiner maximalen Höhe. Die vertikale Geschwindigkeit ist relativ gering, sodass keine Bewegungsunschärfe auftritt. Der Ball wird von einem überbelichteten Fenster teilweise überblendet, wodurch Bereiche der visuellen Erscheinung mit dem Hintergrund verschmelzen. Abgebildet wird dies in Abbildung 3.9. Das Netz detektiert den Ball trotz seiner eher ovalen Erscheinung und setzt sowohl Radius als auch Mittelpunkt korrekt.

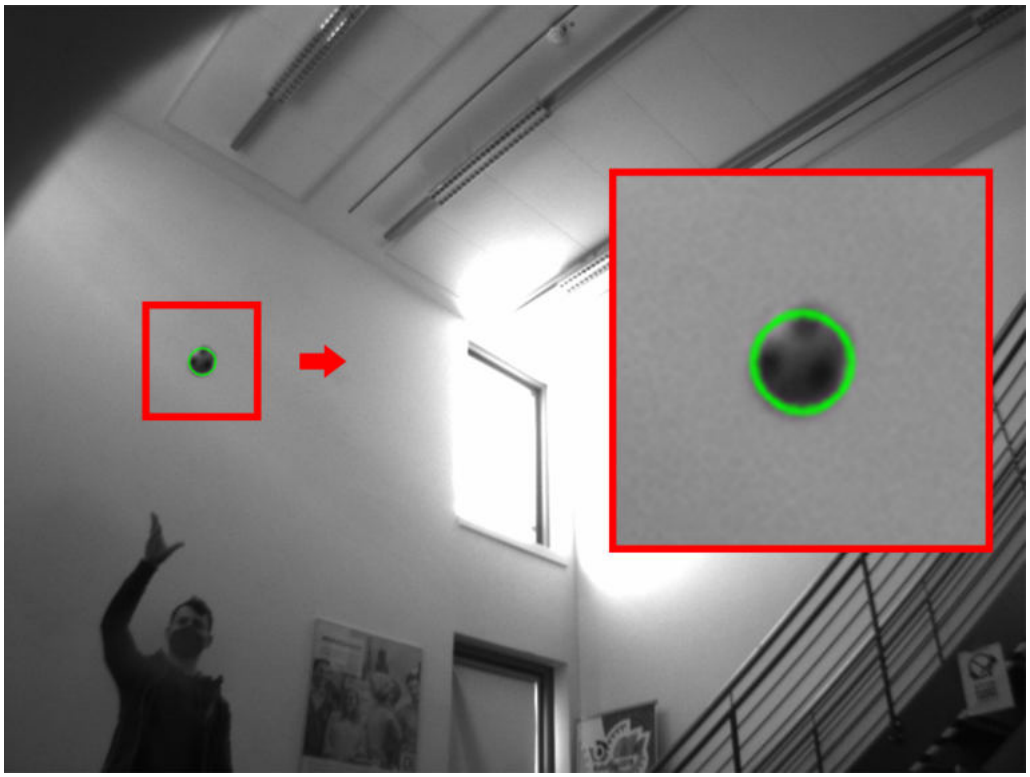

Abbildung 3.8: Experiment 1, Ball mit Bewegungsunschärfe ohne schwierige Erkennungseigenschaften. Korrekte Vorhersage in Grün durch entwickeltes System. Rechte Seite zeigt Vergrößerung.

Experiment 3: Konsistenz eines Wurfes mit Bewegungsunschärfe

Ein Wurf eines Balles nicht in den Aktionsradius des Roboters. Der Ball wird in einer ungefähren Entfernung von 3 Metern in einem hohen Bogen zum Roboter geworfen. Während des Fluges treten keine Extrembelichtungen oder extremen Unschärfen auf, sodass dies einen einfach erkennbaren Wurf darstellt. In 37 aufeinander folgenden Bildern wird der Ball in 35 Bildern korrekten erkannt. In Abbildung 3.10 ist eine Überlagerung der Bilder mit den erkannten Bällen des Wurfes zu sehen. Die Flugbahn des Balls hat einen Knick, da sich der Roboter während der Aufnahme bewegt.

Experiment 4: Konsistenz eines Wurfes mit Überblendung

Ein Teil eines Wurfes mit 24 aufeinander folgenden Bildern von Bällen ist in Abbildung 3.11 dargestellt. Die erkannten Bälle (in Grün) sind chronologisch mit Annotierung (in Blau) aufgereiht. Auch hier zeigt sich die Konsistenz der Erkennungsleistung deutlich. Bei extremer Überblendung wird keine Vorhersage zu dem Ball getroffen. Nach genauerer Auswertung mehrere Würfe zeigt sich, dass eine einseitige Überblendung des Balls, hervorgerufen durch eine sehr starke seitliche Beleuchtung, der Erkennung weniger Probleme bereitet, als eine Überblendung an zwei Seiten. In diesem Beispiel ist eine zweiseitige Überblendung durch eine Beleuchtung von hinten hervorgerufen worden.

3.3.4 Quantitative Auswertung

Mit der Auswertung von 805 Bildern auf denen jeweils genau ein Ball zu erkennen ist und der Anwendung der definierten Toleranz $\frac{r}{5}$ als Positions- als auch Radiusfehler erkennt das Netz 693 von 805 (86%) Bällen korrekt. Bei der Anwendung von strengeren Toleranzen von

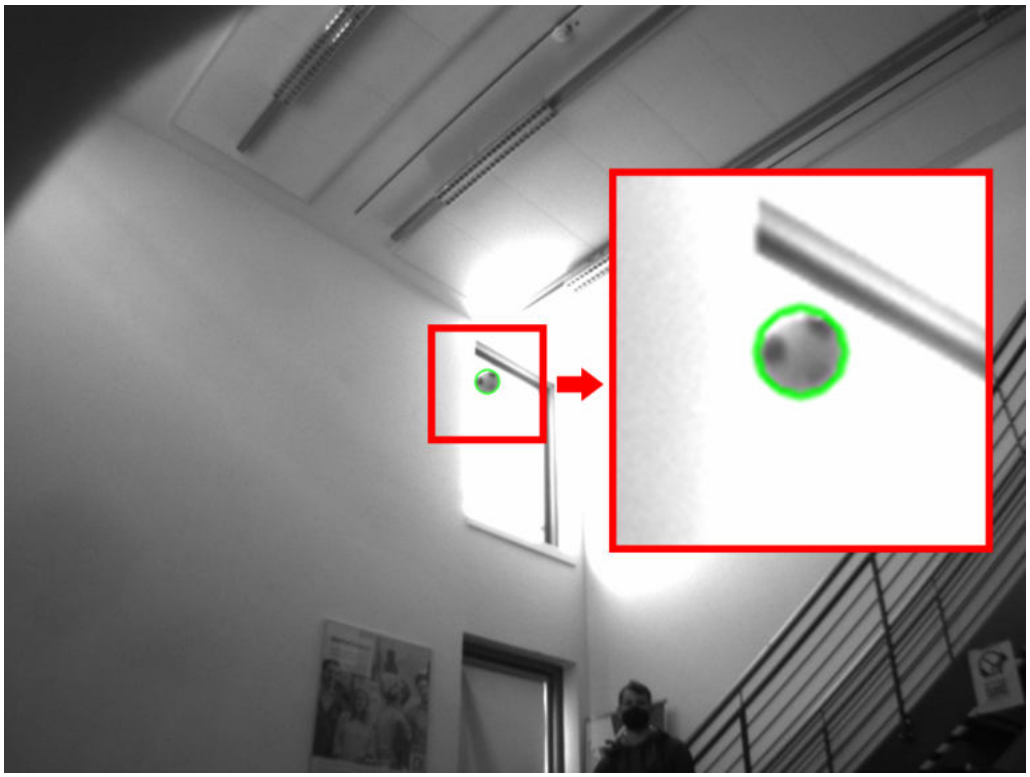

Abbildung 3.9: Experiment 2, Ball ohne Bewegungsunschärfe vor einem blendenen Fenster. Teilbereiche des Balls werden überblendet und lassen sich nicht vom Hintergrund differenzieren. Segmentierung durch entwickeltes System trotzdem erfolgreich. Rechte Seite zeigt Vergrößerung.

$\frac{r}{10}$ werden immer noch 451 (56%) Bälle erfolgreich erkannt. Eine Sichtung der erkannten Bälle zeigt, dass Bälle ohne spezielle Besonderheiten wie große generelle Unschärfe, große Bewegungsunschärfe oder Überblendung durch Extrembelichtungen sehr zuverlässig erkannt werden. Eine weitere Auswertung dieser Bälle zeigt, dass von 680 leicht zu erkennenden Bällen das Netz 642 (94%) mit strenger Toleranz von $\frac{r}{10}$ erkennt.

Das bestehende System liefert auf den gleichen Bildern mit der definierten Toleranz von $\frac{r}{5}$ für 457 Bälle eine korrekte Erkennung und erreicht damit eine Erkennungsleistung von 57%. Bei der strengeren Toleranz von $\frac{r}{10}$ werden noch 231 (29%) der geworfenen Bälle erkannt. Bei genauerer Betrachtung der nicht erkannten Bälle fällt auf, dass sowohl das Muster des Balls auf leicht zu erkennenden Bällen, als auch die beidseitigen Überblendungen der Ballränder dem bestehenden System Schwierigkeiten bereiten.

Eine Auswertung und ein Vergleich der falsch-positiven Erkennungen ist in diesem Kontext nicht sinnvoll. Das bestehende System ist so entwickelt, dass die Erkennungskomponente immer die 10 wahrscheinlichsten Kreise findet und dem restlichen System mitteilt. Dieses ist genau darauf entwickelt falsche Erkennungen auszusortieren und kann dadurch mit einer großen Menge an Falscherkennungen umgehen. Das neuronale Netz schneidet dabei ebenfalls besser ab, sodass im Schnitt pro erkanntem Ball auf den Testdaten nur 0,8 weitere falsche Erkennungen gemacht wurden. Wertet man bei den Erkennungen des bestehenden Systems nur relativ sicher erkannte Bälle, so bleiben trotzdem pro erkanntem Ball 4,5 Falscherkennungen bestehen.

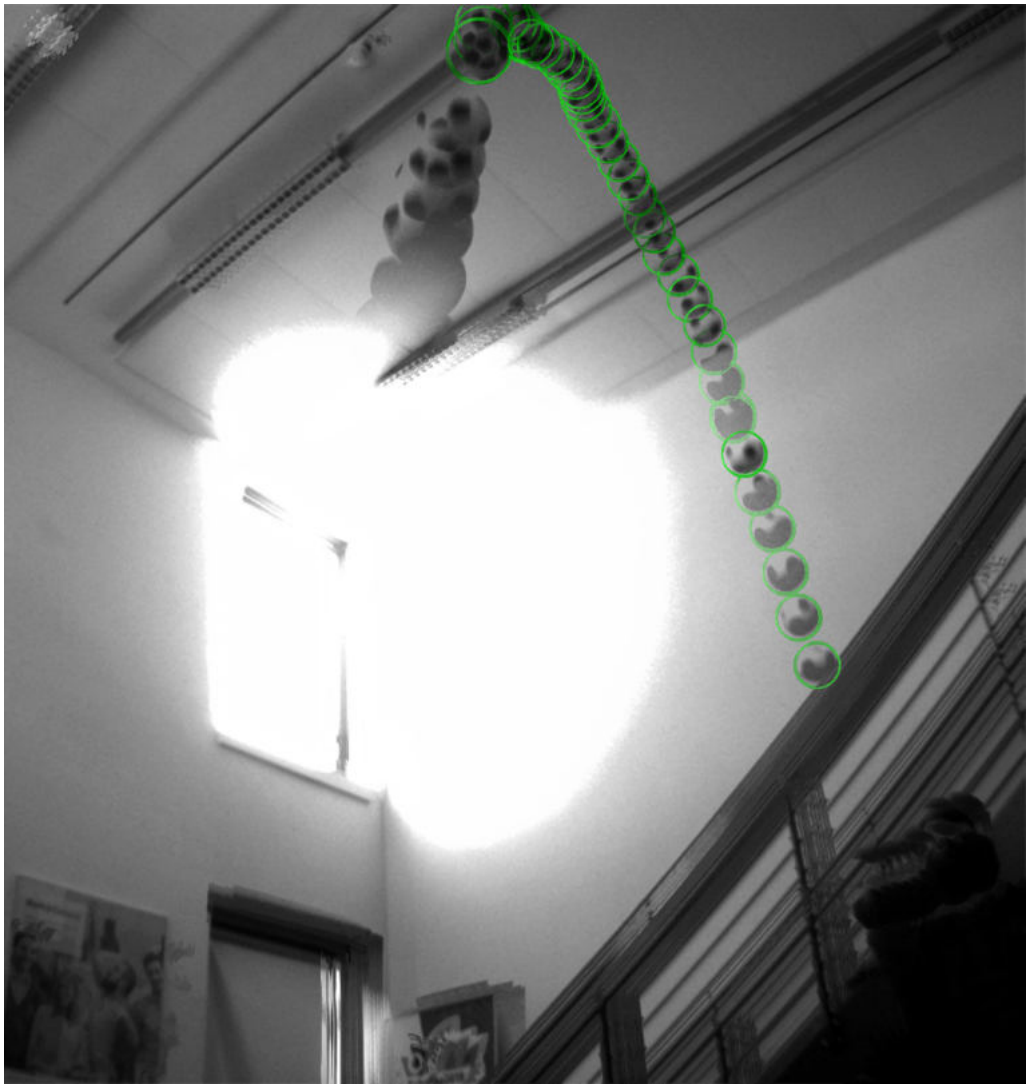

Abbildung 3.10: Experiment 3, eine Überlagerung mehrerer Bilder einer Flugbahn eines Balles. Grüne Umrandung zeigt erkannte Bälle durch das entwickelte System. Größere Bälle werden nicht erkannt, da diese irrelevant für das System sind.

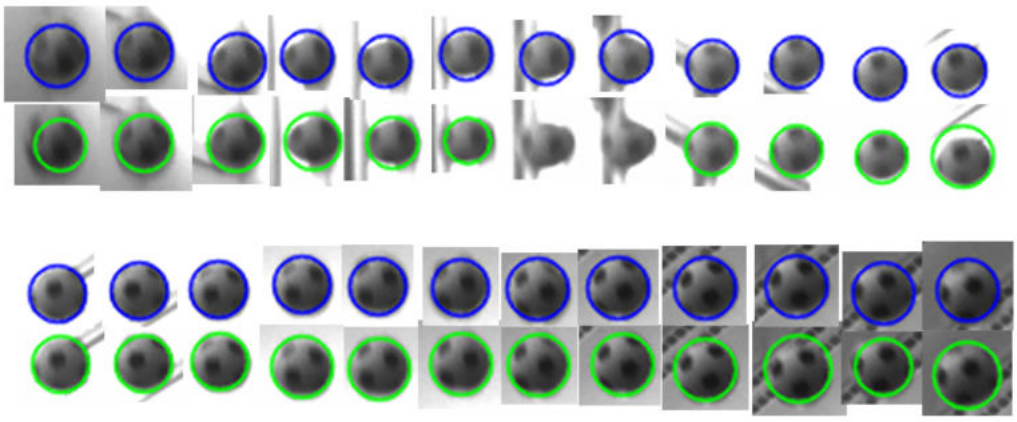

Abbildung 3.11: Experiment 4, eine Aufreihung der erkannten Bälle (in Grün) mit jeweiliger zugehöriger Grundwahrheit (blau).

3.4 Anwendung im Robotersystem

Durch die eingeschränkte Auswertung ohne die verbaute Hardware des Roboters kann für die Anwendung des erarbeiteten Systems im realen Umfeld nur eine Richtung bestimmt werden. Die Einrichtung auf dem System, das für die Auswertung benutzt wurde, lässt eine Idee über eine mögliche Anwendung im Roboter gewinnen. Das benutzte Auswertungssystem beinhaltet einen Intel Core i5-12400 Prozessor mit 6 physischen Prozessorkernen und einer 2,5 GHz Taktung mit Intel Turbo Boost auf maximal 4,4 GHz. Für eine schnelle Inferenz wird eine NVIDIA GeForce RTX 3080 Grafikkarte mit 8960 NVIDIA CUDA-Recheneinheiten und 12 GB VRAM Speicher genutzt. Mit der Nutzung der gleichen Bibliotheken für eine Inferenz des neuronalen Netzes und der Nachverarbeitung der Ergebnisse ist eine Verarbeitungszeit eines einzelnen Bildes in 11,9 Millisekunden möglich. Durch das Stereokamerasystem des Roboters sind immer zwei Bilder gleichzeitig für die Verarbeitung vorgesehen. Durch die Nutzung von Parallelverarbeitung mit Hardwarebeschleunigung kann über eine gleichzeitige Verarbeitung beider aufgenommener Bilder eine Gesamtzeit unter 16 Millisekunden erreicht werden. Diese Zeiten beinhaltet das Laden der Bilder von der Festplatte, die Inferenz des Netzes, die Nachverarbeitung der Netzausgabe, die Konturenfindung der semantischen Segmentierung und der anschließenden Berechnung der Ballpositionen und Radien. Da das bestehende System mit beiden Kameras in 50Hz aufzeichnet, bleibt für die gesamte Verarbeitung beider Bilder weniger als 20 Millisekunden. Zusammen mit den anderen Komponenten des Systems, dem Betriebssystem des Roboters und dem Zusammenspiel der einzelnen Softwareteile, reicht die Verarbeitungszeit eines Bildes nicht für eine zuverlässige Nutzung im aktuellen Umfeld.

Mit dem bestehenden Robotersystem und der aktuellen Hardware ist eine Nutzung der erarbeiteten Ballerkennung nicht denkbar. Nicht nur reicht die Bearbeitungszeit eines Bildes nicht einmal theoretisch auf dem Evaluationssystem, sondern ist die verbaute Hardware am Robotersystem über 10 Jahre alt. Der Prozessor des Evaluationssystems ist laut verschiedener Analysen von *cpu-monkey.com* mehr als 2,5-mal in Einzelkern- und Mehrkernberechnungen schneller. Ein viel größeres Problem stellt die nicht vorhandene Parallelrecheneinheit des Robotersystems dar, denn im Evaluationssystem wird ein aktuelles Spitzenmodell von Grafikkarten für diese Aufgabe verwendet. Für die Beschleunigung der Inferenz eines Convolutional-basierten neuronalen Netzes ist eine entsprechende Hardwarebeschleunigung unbedingt notwendig, um ein echtzeitfähiges System zu betreiben.

Vor der Idee des Einsatzes neuronaler Netze im Robotersystem von *Doggy* war die Problematik der Nutzung bildverarbeitender neuronaler Netze bekannt. Die Entwicklung des Netzes zeigt, dass für den Einsatz zukünftiger, rechenintensiver Methoden das Robotersystem modernisiert werden muss. Dennoch kann über eine Reduktion des Aufnahmeintervalls des Kamerasystems auf 40 Hz oder sogar 30 Hz genug Zeit geschaffen werden, dass das Evaluationssystem in der Form als Steuerungseinheit des Roboters denkbar wäre. Das gesamte Bild wird für die Ballerkennung im Netz verarbeitet, obwohl es typische Bereiche für erfolgreiche und sinnvolle Erkennungen gibt. Diese werden teilweise durch das Zielszenario definiert und erlauben eine gewisse Reduktion der Bildbereiche und damit eine schnellere Berechnung.

4 Schlagbewegung

Wird ein Ball erfolgreich vom visuellen System erkannt und verfolgt, wird eine Vorhersage über die Flugbahn vom bestehenden System erstellt. Schneidet diese Flugbahn den bekannten Aktionsradius des Schlagarms, ist es möglich, eine geeignete Schlagtrajektorie für den Arm zu planen. Dabei sind einige Dinge zu beachten, um einen erfolgreichen, effektiven Schlag zu erwirken. Im Sinne des Zielszenarios ist die Durchführung eines erfolgreichen Schlages die wichtigste Komponente der gesamten Interaktion zwischen Mensch und Roboter. Die Schlagbewegung ist die einzige Komponente des Roboters, welche die Gesamtleistung des Systems zum Benutzer vermittelt. Die planende Komponente muss sich dabei auf das gesamte Zusammenspiel aller Komponenten verlassen. Somit ist die Schlagbewegung maximal so gut, wie die anderen Komponenten des Systems. Als einzige agierende Komponente im gesamten Robotersystem ist die Schlagbewegung somit Hauptaugenmerk in der Leistung des Roboters. Neben der Effektivität des Schlags zählen noch andere Faktoren in Bewertung des Zusammenspiels zwischen Mensch und Maschine.

Menschen besitzen ein intuitives Verständnis für einen erfolgreichen Schlag oder allgemeiner, für eine erfolgreiche, effektive Bewegung. Das Perfektionieren von Bewegungen um größtmögliche Effektivität ist durch die durchgehende Anwesenheit von Sport so tief in dem Wesen Mensch verankert, dass dieses Verständnis in vielen Fällen stimmt. So finden humanoide Roboter nach Kupferberg et al. [15] eine höhere Akzeptanz bei Menschen, wenn sie ein menschliches Bewegungsprofil aufweisen. Eine erfolgreiche und effektive Bewegung im Zielszenario definiert sich durch eine flüssige Bewegung zum ankommenden Ball mit einer geeigneten Schlagtrajektorie, die auf die Flugbahn des Balls abgestimmt ist. Dabei ist es wichtig, dass die Bewegung des Schlagarms präzise und kraftvoll ausgeführt wird, um einen erfolgreichen Schlag zu erzielen. Anwendungsspezifisch ist es zusätzlich wichtig, dass ein geeigneter Punkt des Endaktuators die Berührung mit dem Ball durchführt, um eine möglichst effektive Kraftübertragung und Schlagrichtung zu erzielen.

Neben den ästhetischen und effektiven Anforderungen an den Schlag, ist eine sinnvolle und nutzbare Methode zur Trajektorienerzeugung zu wählen, die zu den technischen Gegebenheiten des existierenden Roboters passt.

4.1 Methodik

Für die Erzeugung einer Schlagtrajektorie muss die Bewegung für den Roboter pro Gelenk bestimmt werden. Als lineare Interpolationsmethode für die Erzeugung von Trajektorien werden Bézier-Splines für alle Aktuatoren genutzt. Bézier-Splines gewähren lokale Kontrolle an den Kontrollpunkten, sind differenzierbar und mit leichten Anpassungen C^1 kontinuierlich. Damit eignen sie sich perfekt für die Planung von physikalischen Bewegungen in der Realität. Die Eingabe des entwickelte System werden aktuelle Position der Roboteraktuatoren q_0, aktuelle Geschwindigkeit der Roboteraktuatoren $\dot{q}_0$, der Zeitpunkt des Schnittes von Balltrajektorie und Aktionsraum, also der Schlagzeitpunkt t, Koordinaten des Balles zum Schlagzeitpunkt p_t und Geschwindigkeit des Balles zum Schlagzeitpunkt $\dot{p}_t$. Daraus ergibt sich die Menge der Eingabeparameter: $(q_0, \dot{q}_0, t, p_t, \dot{p}_t)$ mit $q_0, \dot{q}_0, p_t, \dot{p}_t \in \mathbb{R}^3$ und $t \in \mathbb{R}$.

Als Ausgabe liefert das Netz Parameter für die Berechnung zweier Schlagtrajektorien: einer Schlagtrajektorie mit einer Kurve bis zum Zeitpunkt des Schlages und einer Kurve für das Zurückkehren vom Schlag zur Startposition. Der Ausgabeparameter t ist der vom Modell

vorhergesagter Zeitpunkt, zu dem der Kontakt zwischen Schläger und Ball erwartet wird, q_t sind die Gelenkpositionen des Roboters zum Zeitpunkt des Schlags, b_t ist ein Bézierkurven spezifischer Punkt, der als dritter Kontrollpunkt in der Schlag-Kurve verwendet wird und die Geschwindigkeit der Gelenke beim Kontakt festlegt, t_r ist die Dauer der zweiten, zurückkehrenden Kurve. Die Ausgabe besteht somit aus den Parametern (t, q_t, b_t, t_{ret}) mit $q_t, b_t \in \mathbb{R}^3$ und $t, t_{ret} \in \mathbb{R}$. Zusammen mit den Eingabeparametern ergibt sich die Funktion:

$$\Omega(q_0, \dot{q}_0, t, p_t, \dot{p}_t) = (t_h, q_t, b_t, t_r) \tag{4.1}$$

Dabei ist zu beachten, dass die Parameter $q_0, \dot{q}_0$ und q_t aus dem Gelenkwinkelraum des Roboters sind und grundsätzlich die Position im Bogenmaß angeben. p_t und $\dot{p}_t$ geben Position und Geschwindigkeiten im Welt-Koordinatensystem an. Der Roboter befindet sich in dem Welt-Koordinatensystem im Ursprung und der Ball wird grundsätzlich aus der positiven Richtung zum Ursprung geworfen.

4.1.1 Bewegungstrajektorie

Zur Berechnung einer Trajektorie aus den Ausgabeparametern des Netzes wird ein Bézier-Spline aufgestellt, der diese Parameter nutzt. Dieser Spline muss verschiedene Eigenschaften besitzen, um in der realen Welt anwendbar zu sein.

Spline

Der Bewegungsspline ergibt sich aus den Teilkurven des Schlags und dem Zurückkehren zur Startposition. Mit den vorgestellten Parametern ergeben sich so die Kontrollpunkte für die beiden Splines:

$$b_{hit} = \begin{bmatrix} q_0 \\ \dot{q}_0 \\ b_t \\ q_t \end{bmatrix} \tag{4.2}$$

$$b_{ret} = \begin{bmatrix} q_t \\ q_t + \dot{q}_t(t_r/t_h) \\ 0 \\ q_0 \end{bmatrix} \tag{4.3}$$

Der zusammengesetzte Bézier-Spline ergibt sich im Intervall $[0, t_h + t_r]$ somit aus den folgenden Teilkurven in Matrixform:

$$B_{hit}(u) = \begin{bmatrix} 1 & u/t_h & (u/t_h)^2 & (u/t_h)^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} q_0 \\ \dot{q}_0 \\ b_t \\ q_t \end{bmatrix} \text{ mit } u \in [0, t_h] \tag{4.4}$$

$$B_{ret}(u) = \begin{bmatrix} 1 & u_{dt} & u_{dt}^2 & u_{dt}^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} q_t \\ q_t + \dot{q}_t(t_r/t_h) \\ 0 \\ q_0 \end{bmatrix}$$
$$\text{mit } u \in [t_h, t_h + t_r] \text{ und } u_{dt} = (u - t_h)/t_r \tag{4.5}$$

Kontinuität

Damit der erstellte Spline in der Realität verwendet und am Roboter ausgeführt werden kann, müssen zur Definition eines Bézier-Splines bestimmte Bedingungen erfüllt sein und Vorkehrungen getroffen werden. Eine Bewegung in der Realität ist immer C^0 und in den meisten Fällen auch C^1 kontinuierlich. C^0 kontinuierlich bedeutet, dass keine plötzlichen Positionssprünge stattfinden. C^1 kontinuierlich bedeutet, dass auch die Änderung der Positionsänderung, also die Geschwindigkeit oder die Ableitung keine Sprünge aufweisen darf. Sei ein Bézier-Spline zusammengesetzt aus den Kurven A und B, jeweils mit den Kontrollpunkten 1, 2, 3 und 4 und A(X) der X-te Kontrollpunkt von A, dann ergeben sich folgende Kontinuitäten:

$$C^0 : A(4) = B(1) \tag{4.6}$$

$$C^1 : A'(4) = B'(1) \tag{4.7}$$

Länge

Eine Teilkurve eines Bézier-Splines ist eine Funktion im Intervall $[0, 1]$ und dient nur der Interpolation zwischen Kontrollpunkten. Mit der Verwendung eines Zeitrahmens muss auf die korrekte Skalierung der Parameter geachtet werden. Setzt man zwei Kurven unterschiedlicher Länge zusammen, so muss man in Einhaltung der Kontinuität besonders auf die C^1 Kontinuität achten und die mittleren Kontrollpunkte 2 und 3 passend in Abhängigkeit von den Längen der vorangehenden und nachfolgenden Kurve skalieren.

4.1.2 Verlustfunktion

Mit der Funktion 4.1 können Parameter für eine Bewegungstrajektorie des Schlägers zu einer Position im Raum und die Rückkehr dessen zur Startposition berechnet werden. Um ein neuronales Netz zu trainieren, kann eine Verlustfunktion verwendet werden. Diese bewertet die Ausgabe des Netzes und misst die Leistung basierend auf einer bestimmten Grundwahrheit für die spezifischen Eingabeparameter. Beim traditionellen Training wird zu jedem Eingabedatum vorher eine Grundwahrheit definiert. Die Vorhersage des Netzes versucht beim Training nun diese Grundwahrheit für jedes Eingabedatum vorherzusagen. Die Hoffnung dabei ist, dass nicht nur für die Menge der Trainingsdaten eine gute Vorhersage gefunden wird, sondern dass über diese Menge hinaus eine Generalisierung der Eingabedaten stattfindet und auch für unbekannte Daten eine gute Vorhersage getroffen werden kann.

Das verwendete Loss unterscheidet sich von herkömmlichen Verlustfunktionen und ähnelt eher einer Kostenfunktion. Zu einem Eingabedatum gibt es keine explizite Grundwahrheit. Diese steckt implizit verteilt in mehreren Verlustfunktionen, bei denen nicht direkt zwischen Vorhersage und Grundwahrheit verglichen, sondern die Vorhersage anhand mehrere Eigenschaften bewertet und dann der Verlust berechnet wird. Die Ausgabeparameter werden dabei in mehreren Funktionen verwendet, um zusammen eine abstrahierte Gesamtleistung zu ermitteln. Diese wird dann wiederum traditionell dazu genutzt, das Modell schrittweise anzupassen, um eine bessere Vorhersage zu treffen. Es wird dadurch keine *genauere* Vorhersage erreicht, sondern Werte ermittelt, die diese Funktionen minimieren.

Das Loss ist eine balancierte Kombination aus verschiedenen Teilfunktionen, um die ausgegebenen Parameter des Netzes zu bewerten. Zur Bewertung einer Ausgabe werden verschiedene Funktionen benutzt, um die physikalischen Einschränkungen des Roboters widerzuspiegeln. Dazu zählen der eingeschränkte Bewegungsradius des Roboters durch die Bauweise des Roboters, die maximalen Geschwindigkeiten und maximalen Drehmomente der Achsen. Außerdem wird der aufgabenspezifische Verlust errechnet, der sich auf das

Zurückspielen des Balls bezieht. Dazu zählt das Treffen des Balls, die erwirkte Geschwindigkeit und die Richtung. Als Letztes wird ein allgemeiner Aufwandsverlust erstellt, um unnötigen Bewegungen oder zeitlichen Verschwendungen entgegenzuwirken.

Bewegungsgrenzen

Die Position jedes Gelenks zu einem Zeitpunkt t wird in q_t (4.8) einzeln modelliert. Diese ergeben sich durch die Diskretisierung des aufgestellten Splines. Das Ausreizen des Aktionsradius führt zur Möglichkeit höherer Geschwindigkeiten und Kräfte beim Spielen des Balles, eine Überschreitung der Grenzen ist physisch nicht möglich und führt deshalb zu großen Verlusten. Die Bewegungsbereiche der verschiedenen Gelenke des Roboters sind im vorhandenen System angegeben. Dabei wird die Gier-Achse zwischen -80 und +80 Grad, die Nick-Achse zwischen -55 und +90 Grad und die Roll-Achse zwischen -42 und +45 Grad eingeschränkt. Die ungleichen Einschränkungen der Achsen und Richtungen sind durch die Bauweise der Gelenke entstanden. Definiert werden die Grenzen als q_{max} (4.9) und q_{min} (4.10). Um dieses Verhalten im Loss zu modellieren definiert Formel 4.11 die Verlustfunktion für das Überschreiten der Bewegungsgrenzen.

$$q_t = \begin{bmatrix} q_{yaw} \\ q_{pitch} \\ q_{roll} \end{bmatrix} rad \tag{4.8}$$

$$q_{max} = \begin{bmatrix} 1.39626 \\ 1.57079 \\ 0.78539 \end{bmatrix} rad \tag{4.9}$$

$$q_{min} = \begin{bmatrix} -1.39626 \\ -0.95993 \\ -0.73303 \end{bmatrix} rad \tag{4.10}$$

$$L_{pos} = \sum_t (max(q_t, q_{max}) - q_{max}) + \sum_t abs(min(q_t, q_{min}) - q_{min}) \tag{4.11}$$

Geschwindigkeitsgrenzen

Die Geschwindigkeit eines kubischen Bézier-Splines kann als quadratischer Bézier-Spline modelliert und diskretisiert werden. $\dot{b}$ (4.12) definiert die Kontrollpunkte eines quadratischen Splines, der die Geschwindigkeit eines Splines b, in Abhängigkeit seiner Kontrollpunkte $b(1)$ bis $b(4)$, angibt. Andererseits ist die Geschwindigkeit auch die Änderung der Position über die Zeit und kann von aufeinander folgenden Positionen im Spline durch Formel 4.13 direkt berechnet werden. Mit kleinerem zeitlichen Abstand zwischen den Diskretisierungen ist diese Methode ebenfalls hinreichend genau. Die Geschwindigkeit der Gelenke wird für jeden Zeitschritt pro Gelenk definiert (4.14). Unabhängig von der Methode zur Bestimmung der Geschwindigkeit können die Motoren des Roboters laut Herstellerangaben zusammen mit der Übersetzung zwischen Antrieb- und Drehgelenk in den verschiedenen Winkeln 180 Grad pro Sekunde erreichen. In Formel 4.15 werden die maximalen Winkelgeschwindigkeiten der Motoren pro Sekunde modelliert. Die errechneten Geschwindigkeiten müssen zur Evaluation von rad/t auf rad/s umgerechnet werden.

Die Geschwindigkeits-Verlustfunktion 4.16 lässt eine Nutzung des möglichen Geschwindigkeitsraumes zu und bestraft, ähnlich zur Bewegungsgrenzen-Verlustfunktion, nur Überschreitungen des Maximums. In Zusammenhang mit der aufgabenspezifischen Verlustfunktion ermuntert dies die Durchführung eines kraftvollen Schlags.

$$\dot{b} = \begin{bmatrix} b(2) - b(1) \\ b(3) - b(2) \\ b(4) - b(3) \end{bmatrix} \tag{4.12}$$

$$\dot{q}_t = (q_{t+1} - q_t)/dt \tag{4.13}$$

$$\dot{q}_t = \begin{bmatrix} \dot{q}_{yaw} \\ \dot{q}_{pitch} \\ \dot{q}_{roll} \end{bmatrix} rad/s \tag{4.14}$$

$$\dot{q}_{max} = \begin{bmatrix} 3.14159 \\ 3.14159 \\ 3.14159 \end{bmatrix} rad/s \tag{4.15}$$

$$L_{mov} = \sum_t (max(abs(\dot{q}_t), \dot{q}_{max}) - \dot{q}_{max}) \tag{4.16}$$

Drehmomentgrenzen

Um eine effektive Schlagbewegung zu erzielen, führen wir außerdem eine Limitierung der aufgebrachten Kräfte ein. Die Trägheit der Komponenten, zusammen mit der Anbringung und Bauweise des Bewegungsapparates des Roboters, führen zu unterschiedlichen Aufwänden bei Bewegungen des Schlägers. Um die Trägheit der Komponenten zu simulieren, kann die Open-Source Bibliothek Pinocchio [3] verwendet werden. Pinocchio bietet verschiedenste effiziente Funktionen zur Berechnung der Dynamik von Roboter-Modellen und kann dabei auch Ableitungen dieser Berechnungen erstellen. Durch die hochperformante Implementierung des rekursiven Newton-Euler Algorithmus [22], vorgestellt von Walker und Orin im Jahr 1982, kann die Bibliothek mit der Eingabe eines Roboter-URDF-Modells ein Drehmoment τ für jedes Gelenk berechnen. Es löst die Funktion $\tau = (q, \dot{q}, \ddot{q})$ für ein Robotermodell und kann dazu auch die Ableitungen $\frac{\partial q}{\partial t}$, $\frac{\partial \dot{q}}{\partial t}$ und $\frac{\partial \ddot{q}}{\partial t}$ berechnen. Die Ableitungen der Berechnung werden für die Fehlerrückführung beim Training des neuronalen Netzes in diesem Fall explizit benötigt, da die automatische Differenzierung von TensorFlow nicht bei externen Bibliotheken funktioniert und diese somit explizit berechnet werden müssen.

Die Nutzung der Pinocchio-Bibliothek im TensorFlow-Berechnungsgraphen schränkt die automatische Differenzierung ein, benötigt einen aufwändigeren Prozess bei der Berechnung und verlängert somit die Durchführungszeiten beim Training des neuronalen Netzes. Stellvertretend für die Drehmoment-Verlustfunktion ist im ersten Ansatz ein Beschleunigungsverlust implementiert worden. Die Beschleunigungs-Verlustfunktion 4.21 dient als erste Annäherung an die Drehmoment-Verlustfunktion und ist durch ihre Ähnlichkeit zur Geschwindigkeits-Verlustfunktion schnell und effizient zu berechnen. Für erste Schritte im Training des neuronalen Netzes liefert die Beschleuningungs-Verlustfunktion eine gute Basis für das Erzeugen geeigneter Trajektorien, für die Nutzung des neuronalen Netzes am Roboter ist ein Training mit der Drehmoment-Verlustfunktion sinnvoll.

Ähnlich zu den Geschwindigkeiten kann die Beschleunigung als Ableitung der Geschwindigkeit direkt aus dem Bézier-Spline als ein solcher (4.17) aufgestellt werden. Gleichzeitig ist es aber auch möglich, aus einem aufgestellten Spline über die diskretisierten Positionen mit Berechnung der Geschwindigkeiten auch die Beschleunigung als Änderung der Geschwindigkeit zu berechnen (4.18). Bei der Berechnung der Geschwindigkeit wird die Änderung über die Differenz von aufeinanderfolgenden Positionen bestimmt. Durch diese Art von Berechnung wird ein zeitlicher Versatz erzeugt. Kleinere Diskretisierungsschritte können dieser Fehler minimieren. Wird die Beschleunigung allerdings auf diesen Fehlern mit gleicher Berechnungsmethode über die Differenzen der Geschwindigkeiten aufgebaut,

können größere Abweichungen entstehen. Um die Auswirkungen zu verringern, bestimmen wir die Beschleunigung zum Zeitpunkt t durch die Änderung von $\dot{q}_{t-1}$ zu $\dot{q}_t$ und wirken dem zeitlichen Versatz mit dem Index-Versatz um 1 entgegen. Durch die Diskretisierung in Zeitschritte müssen auch die Beschleunigungswerte pro Schritt skaliert werden. Da die berechneten Geschwindigkeiten schon eine Skalierung beinhalten, muss hier eine erneute Skalierung nur auf die Länge eines Zeitschrittes erfolgen. Für die maximale Beschleunigung der drei Achsen wird in Gleichung 4.20 als obere Grenze $\pi/2 = 1.57080 rad/s^2$ angenommen.

$$\ddot{b} = \begin{bmatrix} \dot{b}(2) - \dot{b}(1) \\ \dot{b}(3) - \dot{b}(2) \end{bmatrix} \tag{4.17}$$

$$\ddot{q}_t = (\dot{q}_t - \dot{q}_{t-1})/dt \tag{4.18}$$

$$\ddot{q}_t = \begin{bmatrix} \ddot{q}_{yaw} \\ \ddot{q}_{pitch} \\ \ddot{q}_{roll} \end{bmatrix} rad/s^2 \tag{4.19}$$

$$\ddot{q}_{max} = \begin{bmatrix} 1.57080 \\ 1.57080 \\ 1.57080 \end{bmatrix} rad/s^2 \tag{4.20}$$

$$L_{acc} = \sum_t (max(abs(\ddot{q}_t), \ddot{q}_{max}) - \ddot{q}_{max}) \tag{4.21}$$

Die notwendigen Drehmomente für eine Bewegung an einer Position q_t, mit Geschwindigkeit $\dot{q}_t$ und Beschleunigung $\ddot{q}_t$ liefert die Bibliothek Pinocchio durch einen Aufruf der *rnea*-Funktion als τ (4.22). Dieses beinhaltet die nötigen Drehmomente pro Gelenk. Da die Motoren baugleich sind, kann ein Schwellwert für jedes Gelenk verwendet werden. Durch empirische Messungen des bestehenden Systems durch Schüthe, 2017 [19] können die Gelenke ungefähr bis zu 25 Nm an Kraft aufbringen. Eine Überschreitung eines sinnvollen Grenzwertes bei 20 Nm (4.23) bringt einen quadratischen Verlust in die Verlustfunktion 4.24 ein. Außerdem wird zusätzlich ein genereller Verlust bei Motorbewegungen addiert, damit zwischen möglichen ineffizienten und effizienten Schlagbewegungen unterschieden werden kann.

$$\tau_t = rnea(q_t, \dot{q}_t, \ddot{q}_t) Nm \tag{4.22}$$

$$\tau_{max} = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} Nm \tag{4.23}$$

$$L_{tor} = \sum_t (max(\tau_t, \tau_{max}) - \tau_{max})^2 + \tau_t \tag{4.24}$$

4.1.3 Aufgaben Loss

Durch die Anwendung der bisher definierten Dynamikverluste soll eine Trajektorie für den Roboter erzeugt werden, die innerhalb der Machbarkeit und somit den physischen Grenzen des Roboters liegt. Durch den generellen Verlust bei jeglicher Bewegung durch die Drehmoment-Verlustfunktion wird bislang keine Erfüllung des Zielszenarios erwirkt und die optimale Trajektorie lässt den Schläger in einer neutralen, energiesparenden Position. Das Zurückspiel-Verhalten des Roboters soll durch eine aufgabenspezifische Verlustfunktion erreicht werden. Sowohl die Stärke des Kontakts, als auch die Richtung des zurückgespielten Balles tragen zur Aufgabenerfüllung bei. Durch die aufgestellte Kinematik des Roboters

kann zu jedem diskretisierten Zeitpunkt die Position des Endeffektors im Raum bestimmt werden. Der Parameter t_h, der aus dem neuronalen Netz ausgegeben wird, bestimmt den Zeitpunkt des erwarteten Schlages. Damit wird auch der Zeitpunkt bestimmt, bei dem zwischen den bestimmten Trajektorien des Schlages und der Rückkehr umgeschaltet wird.

Die aufgabenspezifische Verlustfunktion besteht aus zwei Komponenten, die jedoch nicht gleichzeitig aktiv sind. Zum einen wird der Schlag des Balles zum Zeitpunkt t_h in Stärke und Richtung bewertet, zum anderen kann der Fall eintreten, dass keine Kollision zwischen Endeffektor und Ball entsteht. Wird keine Kollision zum Zeitpunkt t_h erreicht, so soll die Verlustfunktion dennoch zum Schlagen anregen und bewertet damit den Abstand zwischen Endeffektor und Ball zum eigentlichen Schlagzeitpunkt. Aber auch der gewählte Schlagzeitpunkt selber kann durch die Ausgabe des neuronalen Netzes so gewählt sein, dass sich der Ball außerhalb des Aktionsradius des Schlägers befindet und dadurch kein Schlag möglich ist.

Distanzverlust

Die Kinematik f (4.25) des Roboters liefert uns zu einer Gelenkkonfiguration q die Position des Effektors p_e (4.26) im Weltkoordinatensystem. Die Kinematik stammt aus der Entwicklung des Roboters von Schüthe, 2017 [19], und ist in der referenzierten Dissertation im Appendix A.1. genauer erläutert. Grundsätzlich wird der Schläger als Vektor modelliert und durch Transformationsmatrizen ins Weltkoordinatensystem transformiert. Durch den montierten, kugelförmigen Kopf am Ende des Schlagarms (Endeffektor) kann die Oberfläche als Punkt mit Radius r_e definiert werden. Ein geworfener Ball wird ebenfalls durch die Position des Mittelpunktes p_b mit Radius r_b definiert, wodurch sich der echte Abstand zwischen den beiden Körpern durch Formel 4.27 errechnen lässt. Durch die Auswertung des Abstands zwischen Ball und Schläger zum Zeitpunkt t_h lässt sich ermitteln, ob ein Zusammenstoß erreicht wurde. Ist also zum Zeitpunkt t_h die Distanz kleiner als ein festgelegter Schwellwert, so wird anstelle von der Distanzverlust-Funktion die Schlagverlust-Funktion benutzt, um die erzeugte Kollision zu bewerten. Ist der Abstand allerdings größer, so wird die quadratische Distanz als Distanzverlust-Funktion (4.28) definiert.

$$f(q) = \begin{pmatrix} (c_1 s_2 c_3 - s_1 s_3)1010.069\text{mm} \\ (s_1 s_2 c_3 + c_1 s_3)1010.069\text{mm} \\ 1010.069\text{mm}\ c_2 c_3 + 1013.2\text{mm} \end{pmatrix}, \text{mit } c_i = cos(q_i) \text{ und } s_i = sin(q_i) \quad (4.25)$$

$$p_e = f(q) \quad (4.26)$$

$$\delta p = \|p_e - p_b\| - r_e - r_b \quad (4.27)$$

$$L_{dis} = (\delta p)^2 \quad (4.28)$$

Schlagverlust

Durch die Auswertung der Distanz zwischen Ball und Endeffektor kann ermittelt werden, ob ein Zusammenstoß erreicht wurde. Um einen realen Zusammenprall analytisch zu modellieren, bedarf es einen massiven Aufwand. Eine Simulation des Zusammenpralls kann während der Verlustberechnung nicht durchgeführt werden, da dies den Differenzierungsgraphen von TensorFlow überfordern würde. Auch eine vorberechnete Lookup-Tabelle mit der späteren Suche von Ergebnissen ist, wie bei dem Vorgänger von *Doggy* von Hammer 2011 [11] nicht

möglich. Bei der vorherigen Berechnung und späteren Nachschlagen stehen Eingabe und Ausgabe in keiner modellierbaren Berechnung im Verhältnis und eine Backpropagation wäre ebenfalls nicht möglich. Um dennoch eine Annäherung des Verhaltens zu modellieren, bedienen wir uns den Prinzipien des elastischen Stoßes als auch des ideal plastischen Stoßes in 3 Dimensionen. Dabei wird die Kollision, bestimmend durch den Restitutionskoeffizienten k (4.29), als elastischer- und plastischer Zusammenstoß modelliert. Dafür werden die Geschwindigkeiten der Körper (4.30) vom Mittelpunkt in Richtung des Zusammenpralls projiziert (Formel 4.31 und 4.32), woraus sich für die beiden kollidierenden Körper jeweils v_1 und v_2 ergeben. Mittels Formel 4.33 und 4.34 berechnen sich die neuen Geschwindigkeiten v_1' und v_2'. Durch die Rückprojektion auf die Achsen kann so ein Ergebnis dieses Zusammenpralls ausgewertet werden. Dabei sind m_1 und m_2 die Massen des Schlägers und des Balls. Der Restitutionskoeffizient wurde durch eine kleine Versuchsreihe empirisch ermittelt, bei der die Abprallhöhe bei einem Fall aus einem Meter Höhe gemessen wurde. Durch die Abhängigkeit der Höhe proportional zum Quadrat der Geschwindigkeit ergibt sich der Koeffizient durch Formel 4.29.

$$k = \sqrt{\frac{h'}{h}} \tag{4.29}$$

$$v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} m/s \tag{4.30}$$

$$\delta p = \begin{bmatrix} \delta p_x \\ \delta p_y \\ \delta p_z \end{bmatrix} m \tag{4.31}$$

$$v_n = v_x \frac{\delta p_x}{\|\delta p\|} + v_y \frac{\delta p_y}{\|\delta p\|} + v_z \frac{\delta p_z}{\|\delta p\|} \tag{4.32}$$

$$v_1' = \frac{m_1 v_1 + m_2 v_2 - m_2 (v_1 - v_2) k}{m_1 + m_2} \tag{4.33}$$

$$v_2' = \frac{m_1 v_1 + m_2 v_2 - m_2 (v_2 - v_1) k}{m_1 + m_2} \tag{4.34}$$

Das gewünschte Zielszenario sieht ein Zurückspielen des Balles von Doggy vor. Dieses Verhalten kann einfach erreicht werden, wenn die Geschwindigkeit des Balles nach dem Schlag die Geschwindigkeit vor dem Schlag negiert. Dieses Verhalten lässt den Ball auf der Flugtrajektorie wieder zur werfenden Person zurückkehren. Ein zu starkes Zurückspielen ist mit diesem Verhalten ausgeschlossen und es benötigt keine explizite Angabe einer Zielposition des Empfängers. Nimmt man diese Definition von Zurückspielen als perfekt an, gibt es durch die Konstruktion des Aktionsradius von Doggy Würfe im Aktionsradius des Schlägers, die allerdings nicht perfekt zurückgespielt werden können. Zwar bietet der kugelförmige Endeffektor eine große Varianz an verschieden angewinkelten Flächen, dennoch kann eine aktive Kraft auf den Ball nur tangential zum Bewegungsradius wirken. In Formel 4.35 ist die Geschwindigkeit des Balles in den 3 Raumachsen modelliert. Die Verlustfunktion 4.36 definiert die Umkehrung der Geschwindigkeit durch die Summe der vorherigen Geschwindigkeit mit der wirkenden Geschwindigkeiten des Balles v_b' nach dem Schlag. Für v_b' kann die entsprechende Formel v_2' aus 4.34 benutzt werden.

$$v_b = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} m/s \tag{4.35}$$

$$L_{act} = (v_b + v_b')^2 \tag{4.36}$$

4.2 Implementierung

Die Implementierung des Netzes und der Verlustfunktionen erfolgt in Python mit TensorFlow 2 [1] und Keras [5]. Zusammen bilden sie eine leistungsfähige Plattform für die Erstellung und das Trainieren von neuronalen Netzen. TensorFlow 2 bietet eine Vielzahl von Tools und Funktionen, die es ermöglichen, komplexe Modelle zu erstellen und die Trainingsprozesse zu optimieren. Keras wiederum ist eine einfache, aber dennoch leistungsstarke API, die das Erstellen von Modellen erleichtert und eine schnelle Entwicklung ermöglicht.

TensorFlow erzeugt vor einer Berechnung zuerst einen Berechnungsgraphen. Dieser ist eine abstrakte Repräsentation des mathematischen Modells, der die Abhängigkeiten der Operationen in Form eines gerichteten Graphen darstellt. Jede Berechnung ist ein Knoten in dem Graphen und die Verkettung von mehreren Berechnungen stellen die Kanten zwischen den Knoten dieses Graphen dar. Um eine gezielte Optimierung der Gewichte des Netzes vorzunehmen, ist es wichtig, dass alle Operationen, die in diesem Graphen modelliert werden, differenzierbar sind. Der Prozess der Rückführung des errechneten Verlusts aus der Verlustfunktion durch die Berechnungen auf die einzelnen Parameter des Modells heißt Backpropagation. Durch die Rückführung des Verlusts auf die Parameter des Netzes können gezielt Änderungen vorgenommen werden. Eine vollständige Erläuterung der Funktionsweise ist in der Dokumentation von TensorFlow [1] zu finden.

Die erarbeitete Methodik setzt keinen notwendigen Startzustand in Hinblick auf Position und Geschwindigkeit des Roboters voraus. Für die Implementierung vereinfachen wir dennoch die Funktionen und setzen Startposition, Startgeschwindigkeit, Endposition und Endgeschwindigkeit fest. Hierfür wird die neutrale, statische Position des Roboters mit keiner Geschwindigkeit genau gerade über den Gelenkmittelpunkten gewählt. Der Roboter verbraucht in der neutralen Position keine Energie für die Erhaltung der Position. Dies führt zu einem neutralen, fest definierten Verhalten des Schlägers zusammen mit einer möglichst uneingeschränkten Machbarkeit für die Reaktion auf Würfe in dem gesamten Bewegungsradius. Außerdem schränkt es den Merkmalsraum für das Training des neuronalen Netzes ein und sollte damit das Training vereinfachen.

4.2.1 Neuronales Netz

Das verwendete neuronale Netz besteht aus einer Reihe von Dense-Schichten, die durch *leaky ReLU*-Aktivierungsschichten aktiviert werden. Das Netz bekommt die Eingabeparameter als batch_size $\times$ 13 Vektor. Die teilweise 3-dimensionalen Parameter sind dabei entpackt und als einzelne Komponenten aneinander gereiht. Als Ausgabe liefert das Netz einen batch_size $\times$ 8 Vektor, der die 3-dimensionalen Ausgabeparameter ebenfalls in flacher Struktur enthält. Die *leaky ReLU*-aktivierten Dense-Schichten besitzen eine Kanalzahl von 896, 448, 224 und 112, die Ausgabeschicht ist eine linear aktivierte Dense-Schicht mit 8 Kanälen. Die Abbildung 4.1 zeigt die simple Struktur und den sequentiellen Datenfluss. Die Größe und Dichte des Netzes sind frei gewählt und durch empirische Versuche festgesetzt worden.

4.2.2 Lernverfahren

Splines

Zu Beginn der Berechnung des Losses pro Batch werden für jedes Sample im Batch die beiden Teilkurven des Splines mit jeweils 100 Schritten diskretisiert. Diese Anzahl an Schritten ist festgelegt, sodass die automatische Differenzierung die Berechnungsgradienten für die Backpropagation automatisch erstellen kann. Wäre diese Anzahl variierbar, würde der Berechnungsgraph bei jedem Sample verschieden sein und müsste bei jedem Sample

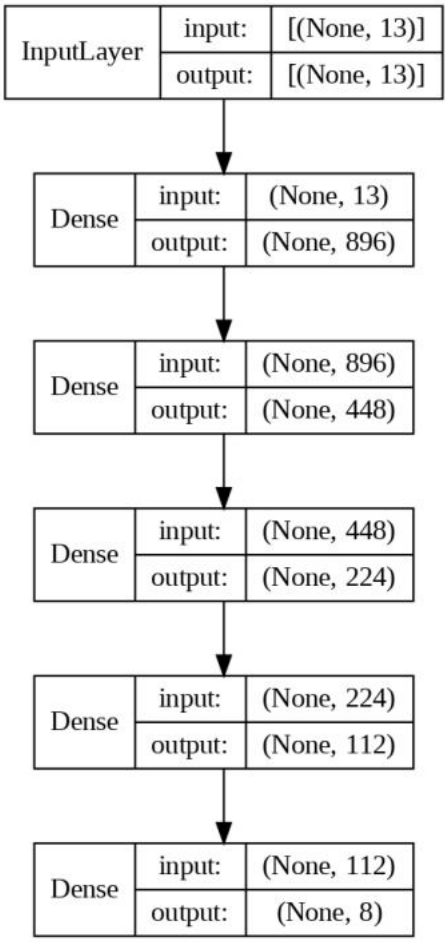

Abbildung 4.1: Verwendete Netzwerkstruktur für die Ausgabe von Parametern für die Erstellung der Schlagtrajektorie.

neu erstellt werden. Bei der Inferenz später am Roboter kann der Spline mit beliebigen Zeitintervallen diskretisiert werden, sodass dieser zu den Spezifikationen der Steuerungseinheit des Roboters passt.

Verlustfunktionen

Die Diskretisierung beider Teilkurven in eine gleiche Anzahl an Schritten, allerdings nicht in gleich große Schritte führt zu einer unterschiedlichen zeitlichen Skalierung. Dies liegt an der frei bestimmbaren jeweiligen Länge der Kurven. Die Berechnung der Geschwindigkeiten und Beschleunigungen der Kurven muss daher getrennt erfolgen und dann zusammengesetzt werden. Dabei sind die einzelnen Kurven an den End- bzw. Startpunkten mit den Informationen der jeweiligen anderen Kurve aufzufüllen.

Die jeweiligen Verlustfunktionen besitzen alle verschiedene Wertebereiche, Skalierungen und Einheiten, sodass eine einfache Summe der Teilverluste in der Gesamtfunktion falsche Minima zwischen den Gegebenheiten entstehen. Dazu steht neben der Einhaltung der physischen Grenzen auch die effiziente und erfolgreiche Aufgabenerfüllung mit minimalem Aufwand. Eine Skalierung der Teilverluste durch Faktoren kann die Wichtigkeit der Einhaltung verschiedener Maße bestimmen und zu einem schnelleren und effizienteren Training beitragen.

Drehmomente

Für die Implementierung der Verlustfunktion für die Überschreitung von Drehmomentgrenzen, als auch der generellen Nutzung von Energie, ist die *rnea*-Funktion der Pinocchio Bibliothek in die Verlustberechnung zu integrieren. Die Berechnung erfordert die Gelenkpositionen, Gelenkgeschwindigkeiten und Gelenkbeschleunigungen zusammen mit der Beschreibung des Robotermodells im URDF Dateiformat. In dem Robotermodell sind physische Eigenschaften des Systems definiert, die zur Berechnung der Drehmomente notwendig sind. Dazu zählen Position, Größe, Trägheit, Gewicht und physikalische Verkettung der Gelenke. Zur Ansteuerung dieser externen Bibliothek wird aus der TensorFlow Loss Funktion heraus über eine vorgefertigte Funktion von TensorFlow für jeden Zeitschritt der Trajektorie eine einfache Python Funktion aufgerufen. Diese berechnet über den Aufruf der Python-Bindings der eigentlichen C++ Bibliothek Pinocchio die Drehmomente. Zurückgegeben werden sowohl die Drehmomente pro Gelenkwinkel, als auch eine Funktion für die Gradientenberechnung dieser Funktion. Pinocchio bietet neben der RNEA Funktion zur Drehmomentberechnung auch die Funktion zur Berechnung der RNEA Derivates, also der Ableitung der Funktion jeweils nach den Eingabeparametern Position, Geschwindigkeit und Beschleunigung. Die Drehmoment-Berechnungsfunktion zusammen mit der Berechnungsfunktion des Gradienten bilden einen funktionierenden Knoten im TensorFlow Berechnungsgraphen und können somit im Training verwendet werden, um den Verlustgradienten pro Gewicht des Netzes aufzustellen.

4.3 Experimente

Im Anschluss an die Entwicklung des Schlagsystems werden eine Reihe von Experimenten durchgeführt, um die Leistung des entwickelten Systems zu evaluieren. Blender bietet neben der Möglichkeit 3D Objekte zu modellieren auch die Möglichkeit, die gesamte 3D Szene über Pythonskripte zu steuern. Für die generelle Umsetzung wurde eine 3D Blenderszene mit einem vereinfachten Robotermodell und einem Ball modelliert. Um eine korrekte Eingabe für das erarbeitete System zu erhalten, wird ein zufälliger Punkt auf dem Bewegungsradius des Roboters berechnet. Eine passende Geschwindigkeit wird außerdem mit der Hilfe von Zufallsvariablen berechnet, um eine realistische Flugbahn im Sinne des Zielszenarios zu erhalten. Zusammen mit einer zufälligen Dauer der Flugbahn ist damit die Bewegungstrajektorie des Balls definiert. Die Position des Balles auf dieser Flugbahn kann zu jedem Zeitpunkt bestimmt werden. Dabei ist zu erwähnen, dass diese Modellierung ohne die Berücksichtigung von Luftwiderstand und Wind passiert. Zudem vereinfacht angenommen, dass 100 Millisekunden nach Beginn der Flugbahn des Balles das Gesamtsystem den Ball erkannt, verfolgt und eine Vorhersage über die Position und Geschwindigkeit des Balles beim Auftreffen auf den Bewegungsradius getroffen hat. Innerhalb dieser Verzögerung hat das neuronale Netz ebenfalls eine Vorhersage für die Parameter des Bewegungssplines des Schlägers getroffen und beginnt sofort die ermittelte Bewegung durchzuführen. Implementiert ist sowohl die Balltrajektorie als auch die Schlägertrajektorie durch Keyframes der Position und Rotation in Blender. Ab dem Schlagzeitpunkt ist die Positionskontrolle des Balles über die Keyframes deaktiviert und die Physiksimulation von Blender übernimmt die Kontrolle. Der Schläger fungiert in diesem Szenario als passiver Starrkörper. Zum Zeitpunkt des Schlages wird der Zustand durch die Physiksimulation berechnet, sodass die Kraftübertragung zwischen Schläger und Ball korrekt berechnet werden kann. Nach dem Zusammenstoß wird die Position des Schlägers wieder von den Keyframes kontrolliert und bewegt sich entlang der zweiten berechneten Trajektorie. Da das vorhandene System am Roboter die Position des Schlägers über einen Regler kontrolliert, ist die Wiederaufnahme der Positionskontrolle nach dem Schlag in der Simulation ähnlich zu dem Verhalten des physischen Systems.

Um ein Szenario für ein Experiment zu erstellen, wird die Position und die Geschwindigkeit des Balles auf dem Bewegungsradius des Roboters so festgelegt, dass die Eigenschaften eines Schlags theoretisch erreicht, oder, je nach Art des Experiments, nicht erreicht werden können. Faktoren des Zurückschlagens sind die Machbarkeit des perfekten Zurückspielens zur Startposition des Balles, die Umkehrung der Flugtrajektorie durch das Umkehren der Fluggeschwindigkeit und Winkel, die generelle Machbarkeit eines Schlages an der Grenze des Bewegungsradius des Roboters und die generelle Machbarkeit einer sinnvollen Trajektoriendurchführung in der Flugzeit des Balles.

4.3.1 Qualitative Auswertung

Experiment 1: Wurf in Richtung des Roboters

Das erste Experiment stellt einen Wurf aus dem perfekten Zielszenario dar. Der Ball wird in eine perfekte Position des Bewegungsradius geworfen, sodass die Flugtrajektorie des Balles durch einen Schlag theoretisch umgekehrt werden kann und der Ball genau zum Startpunkt zurückkehrt.

Der Ball wird relativ gerade vor dem Roboter in einer Distanz von 4,5m in einer Höhe von 0,8m los geworfen. Der Ball trifft in 0,715 Sekunden auf den Aktionsradius des Schlägers. Der Punkt auf dem Aktionsradius ist 0,27m vor dem Zentrum des Roboters auf einer Höhe von 1,94m mit einem horizontalen Versatz von 0,15m auf. Das neuronale Netz wählt den angestrebten Schlagzeitpunkt etwas früher, sodass die Schlagtrajektorie in 0,7 Sekunden endet. Der Zusammenprall zwischen Ball und Schläger findet auf einer Höhe von 2,1m zentral mit dem Endeffektor statt. Durch den Zusammenstoß mittig im Bewegungsradius des Roboters wird der Ball zentral getroffen und kann dadurch bei der Rückkehr wieder an Höhe gewinnen. Die Flugbahn des Balls wird durch den Schlag gut umgekehrt, sodass der Ball 1,16m vom Startpunkt gefangen werden kann. Der Ball wird in einem guten Winkel zurückgespielt, hat allerdings zu wenig Geschwindigkeit, um an die Startposition zurückzukehren. Nach dem Schlag nutzt der Roboter 3,75 Sekunden, um die Schlagbewegung zu bremsen und zur Startposition zurückzukehren. Zum Schlagzeitpunkt nutzen die Motoren akkumuliert 10,5 Nm, davon in der Nick-Achse 8 Nm, in der Roll-Achse 2 Nm und in der Gier-Achse 0,5 Nm. Für die gesamte Bewegung werden, trotzdem 945 Nm aufgewendet. Auf den Abbildungen der Positionen (4.2), Geschwindigkeiten (4.3) und Drehmomenten (4.4) ist zu sehen, dass die Bewegungstrajektorie innerhalb der festgelegten Grenzen stattfindet und eine Durchführung dieses Schlages am Roboter möglich ist. Der effektive Abbau der Schlaggeschwindigkeit ist in Abbildung 4.2 durch die Nutzung des Bewegungsraumes gut zu sehen, wodurch Drehmoment und Energie gespart wird. In Abbildung 4.4 kann man zu Beginn der Bewegungstrajektorie einen leichten Impuls am Roll- und Nickgelenk sehen, gefolgt von einer Gegenbewegung zur Generierung von Geschwindigkeit in Richtung des Balles. Ab dem Moment des erwarteten Zusammentreffens wird eine bremsende Kraft größtenteils an der Nick-Achse ausgewirkt. Zusammen mit Abbildung 4.2 ist bei dem Nachschwung nach dem Schlag zu sehen, dass die Nick-Achse langsam in die Nähe der Bewegungsgrenze kommt, die Überschreitung jedoch durch eine stärkere Nutzung der Bremskraft vermieden wird. Während der Bewegungstrajektorie werden keine Gelenkgeschwindigkeiten erreicht, die außerhalb der Grenzwerte liegen. In Abbildung 4.3 ist gut zu sehen, dass der Geschwindigkeitsraum nicht ausgereizt wird und der Ball nicht so fest wie möglich geschlagen wird, sondern eine genaue Wahl getroffen wurde, um den Ball zum Abwurfpunkt zurückzuspielen. Abbildung 4.6 zeigt die Flugbahn des Balles im Aktionsradius des Roboters. Der Auf- und Abprallwinkel ist sehr ähnlich, sodass eine gute Retoure des Balles möglich ist.

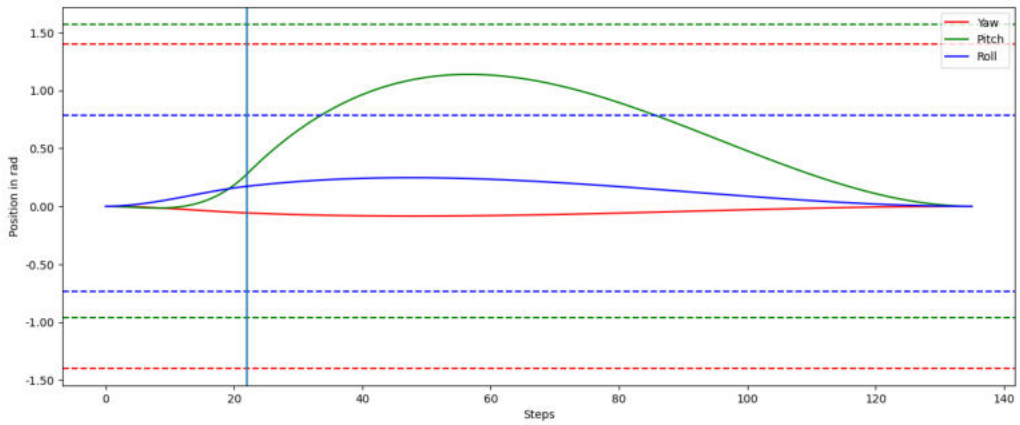

Abbildung 4.2: Gelenkpositionen während der Schlagtrajektorie im Experiment 1. Farblich gestrichelte Linien stellen Bewegungsgrenzen der zugehörigen Achse dar. Vertikaler Strich markiert Schlagzeitpunkt.

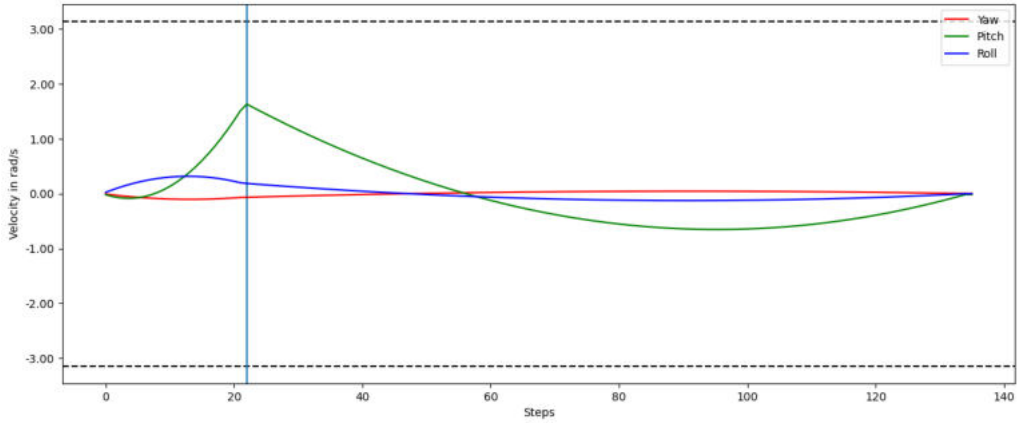

Abbildung 4.3: Gelenkgeschwindigkeiten während der Schlagtrajektorie im Experiment 1. Gestrichelte Linien stellen Geschwindigkeitsgrenzen dar. Vertikaler Strich markiert Schlagzeitpunkt.

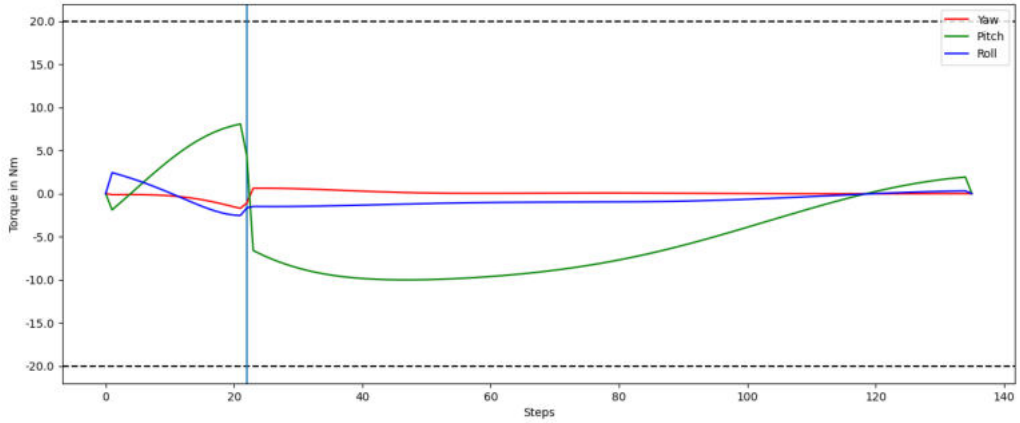

Abbildung 4.4: Notwendige Drehmomente während der Schlagtrajektorie im Experiment 1. Gestrichelte Linien stellen Drehmomentgrenzen dar. Vertikaler Strich markiert Schlagzeitpunkt.

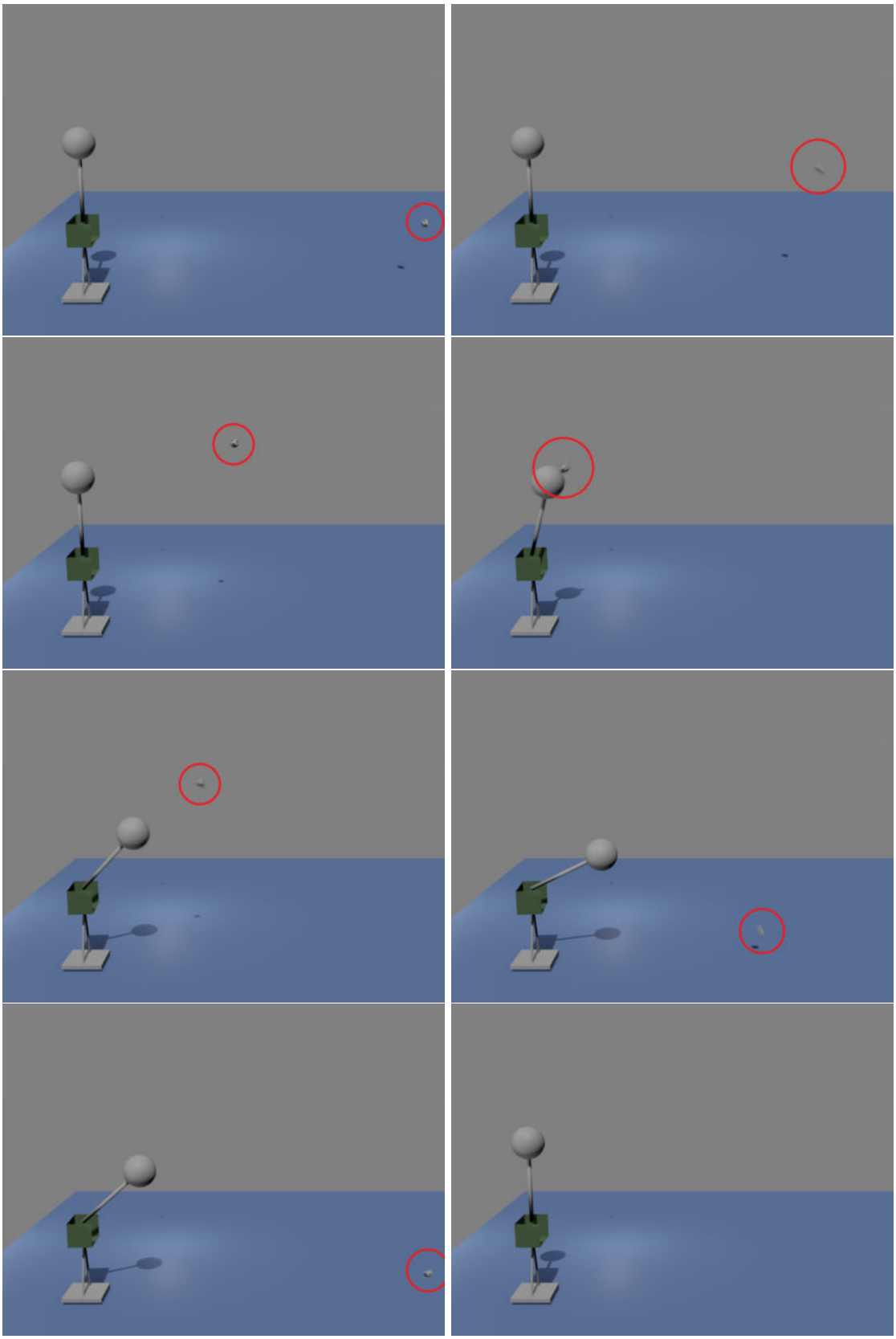

Abbildung 4.5: Simulation des ersten Experiments in Blender. In Reihenfolge von Links-nach-Rechts und Oben-nach-Unten. Bälle sind zur besseren Visualisierung rot eingekreist. Erstes Bild zeigt Startpunkt. Zweites Bild zeigt Start der Roboterbewegung. Bild 4 zeigt Moment vor Kollision. Bild 6 zeigt nächsten Punkt zum Startpunkt. Bild 7 und 8 zeigen Rückkehr des Roboters zur Startposition. Ein Video des Wurfes ist unter folgendem Link zu finden: `https://github.com/ararnice/nn-robotic-task`

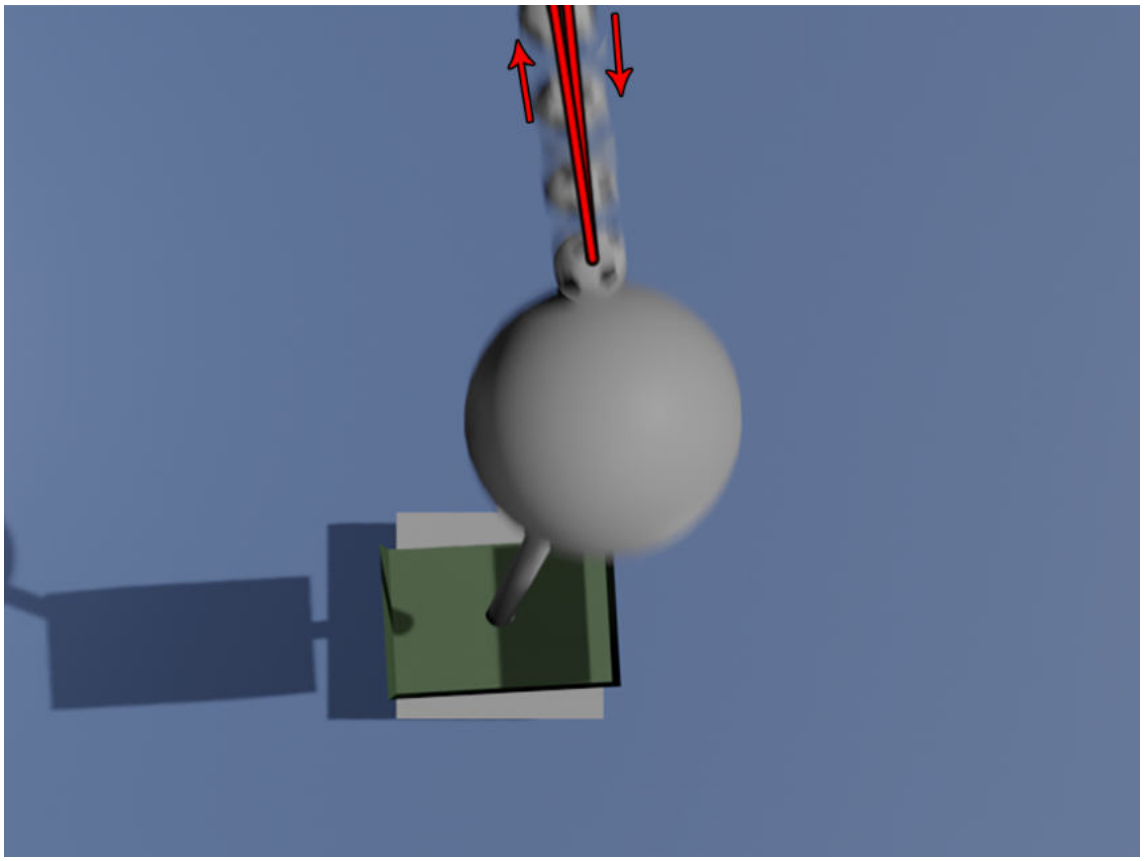

Abbildung 4.6: Aufprall und Abprall des Balls am Endeffektor in Experiment 1. Der Winkel wird durch die Bewegung des Schlägers sehr ähnlich zurückgespielt. In Rot ist die Flugbahn des Balls eingezeichnet, der Schläger ist nur zum Schlagzeitpunkt dargestellt.

Experiment 2: Wurf an den Rand des Aktionsradius

Das zweite Experiment stellt einen Wurf an den Rand des Aktionsradius des Roboters dar. Der Ball wird seitlich am Roboter vorbei geworfen, sodass nahezu der gesamte Bewegungsradius für einen erfolgreichen Schlag stark genutzt werden muss. Der Ball wird gerade vor dem Roboter in einer Distanz von 3,4m in einer Höhe von 0,8m mit einer leichten Richtung nach links außen geworfen. Nach 0,57 Sekunden findet ein Kontakt auf einer Höhe von 1,8m statt. Das Roll-Gelenk befindet sich beim Schlag nur 12,89 Grad von der Bewegungsgrenze entfernt, während das Gier-Gelenk zusammen mit den Nick-Gelenk Geschwindigkeit zum Zurückschlagen aufgebaut hat. Die Flugbahn des Balles ermöglicht an dieser Position kein perfektes Zurückspielen, da der Schläger durch die Schlägerlänge zusammen mit dem kugelförmigen Endeffektor keine Fläche mehr bietet, die einen besseren Abprallwinkel zur Folge hätte. Auch ein früherer Schlag wäre nicht möglich gewesen, da der Ball lediglich mit der oberen Hälfte des Endeffektors erreicht worden wäre. Damit wird nur eine Berührung des Balles erreicht und somit eine Weiterleitung, aber kein Rückspiel, initiiert. 0,38 Sekunden nach dem Schlag befindet sich der Ball 1,86m vom Startpunkt entfernt auf einer Höhe von 1,1m als nächster Punkt zur Startposition und trifft 0,71 Sekunden nach dem Schlag auf den Boden auf. Die Roll-Achse wird genutzt, um den Ball zu erreichen. Dadurch kann hier nur wenig Geschwindigkeit für das Zurückschlagen genutzt werden. In Abbildung 4.7 ist deutlich zu sehen, dass die Roll-Achse beim Schlagzeitpunkt weit ausgestreckt fast bis zur Grenze genutzt wird und sich nach dem Schlag langsam wieder zur Startposition zurückbewegt. Dabei wird kein großer Nachschwinger erzeugt.

Die anderen Achsen haben eine größeren Nachschwinger, befinden sich allerdings während der gesamten Bewegungstrajektorie innerhalb der Bewegungsgrenzen. In Abbildung 4.8 ist hier ebenfalls zu sehen, dass die Roll-Achse zu Beginn schnell beschleunigt wird und beim Schlagzeitpunkt kaum Geschwindigkeit mehr hat. Die anderen Achsen hingegen haben beim Schlagzeitpunkt eine größere Geschwindigkeit, die nach dem Schlag mit Blick auf Abbildung 4.9 mit einer leichten Bremswirkung abgebaut wird.

In Abbildung 4.9 ist außerdem zu sehen, dass während des Zusammenpralls 39 Nm an Drehmoment genutzt werden. Um den Ball am Rande des Bewegungsradius rechtzeitig zu erreichen, muss eine starke Kraft für die Bewegung zur Position und eine antagonistische Kraft zum Bremsen wirken, um innerhalb der Bewegungsgrenzen zu bleiben. Die Roll-Achse benutzt 14 Nm in der Spitze, um die Bewegung zum Ball zu initiieren und bis zu 17 Nm um die Bewegungsgrenzen dieser Achse einzuhalten. Die größere Kraft beim Abbremsen wird durch den größeren wirkenden Hebel durch die Bauweise benötigt. Nick- und Gier-Achse wenden für die Beschleunigung zum Ball jeweils bis zu 14,5 Nm und 7,5 Nm auf. Um die stark beschleunigte Nick-Achse sanft zu stoppen, sodass diese innerhalb der Bewegungsgrenze bleibt, wird nach dem Schlag eine leichte, konstante bremsende Kraft ausgewirkt.

In der Bildserie in Abbildung 4.10 ist die gesamte Flugbahn des Balles in der Seitenansicht zu sehen. Das erste Bild zeigt den Startpunkt des geworfenen Balles und die Startposition des Roboters. Im zweiten Bild ist der Ball geworfen und schon 100 Millisekunden in der Luft. Zu diesem Zeitpunkt beginnt der Roboter seine berechnete Bewegungstrajektorie. Im dritten Bild ist der Ball fast im Bewegungsradius des Schlägers angekommen und der Schläger setzt nach dem Ausholen zum Schlag an. Im 4. Bild ist der Kontakt zwischen Schläger und Ball zu sehen. Im 5. Bild ist der Kontakt abgeschlossen und der Ball folgt der von Blender physiksimulierten Trajektorie. Im 6. Bild befindet sich der Ball am nächsten zum Startpunkt ist dort 1,86m vom Startpunkt entfernt. Der Schlagarm wird langsam gebremst und kehrt im 7. Bild langsam zur Startposition zurück. Das letzte Bild zeigt den abgeschlossenen Vorgang mit der Endposition des Roboters.

In Abbildung 4.11 ist der Abprallwinkel des Balles genauer dargestellt. Die Visualisierung besteht aus der Überlagerung mehrerer Zeitpunkte. Der Endeffektor des Schlägers ist nur zum Schlagzeitpunkt dargestellt. Die kreisförmige Bewegung des Schlägers führt zu einem Schlag des Balles weg von der Startposition. Hierbei ist zu bemerken, dass sich der Ball zu einem früheren Zeitpunkt auf der Trajektorie nicht im vertikalen Aktionsradius des Roboters befindet und somit kein effektiver Schlag durchgeführt werden kann.

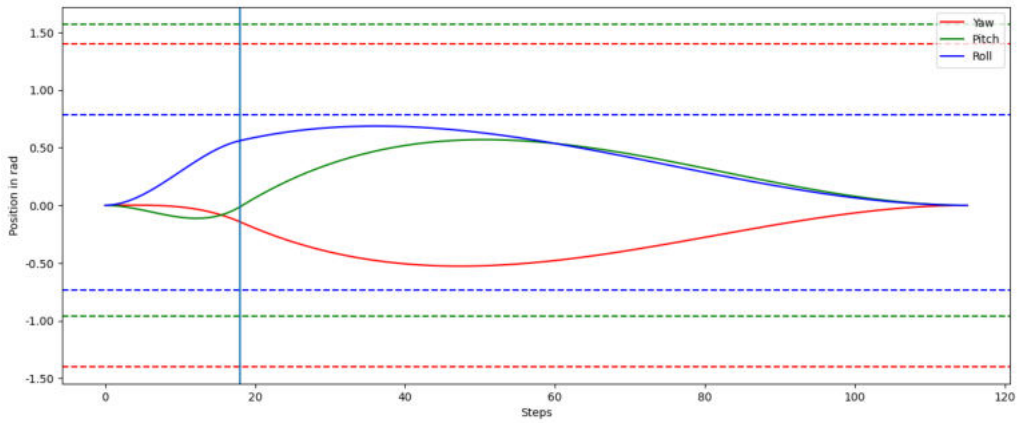

Abbildung 4.7: Gelenkpositionen während der Schlagtrajektorie im Experiment 2. Farblich gestrichelte Linien stellen Bewegungsgrenzen der zugehörigen Achse dar. Vertikaler Strich markiert Schlagzeitpunkt.

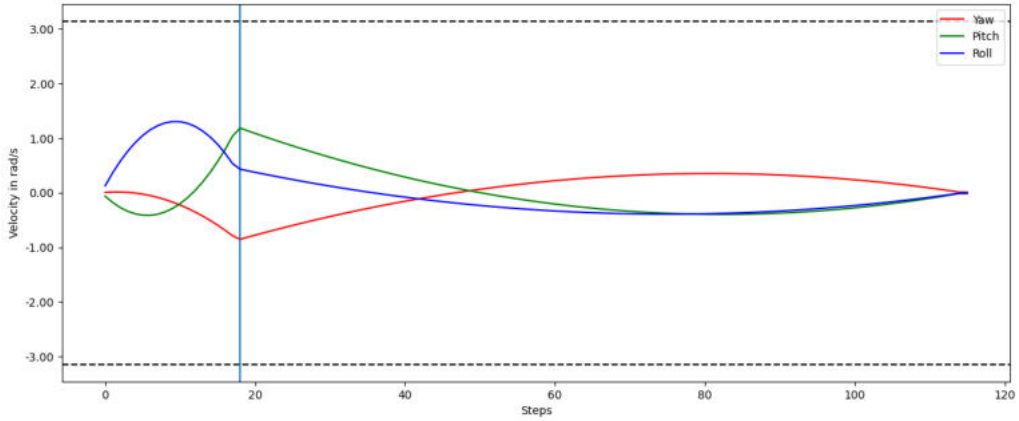

Abbildung 4.8: Gelenkgeschwindigkeiten während der Schlagtrajektorie im Experiment 2. Gestrichelte Linien stellen Geschwindigkeitsgrenzen dar. Vertikaler Strich markiert Schlagzeitpunkt.

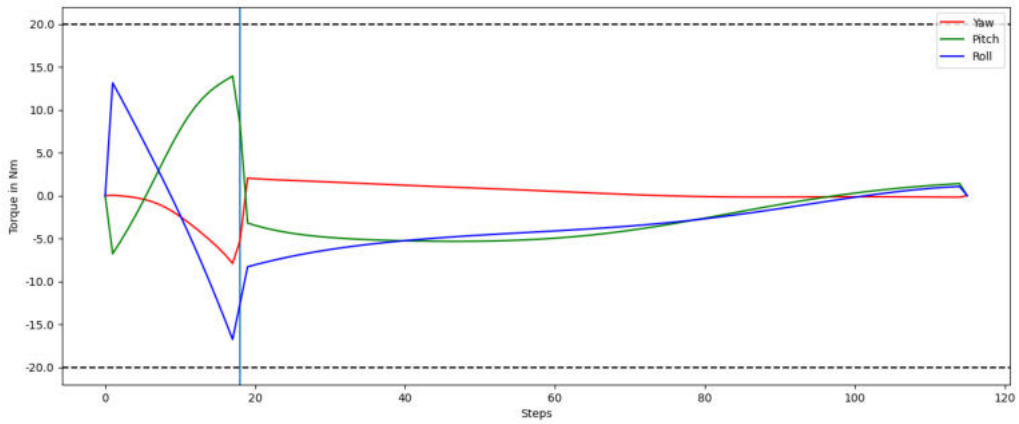

Abbildung 4.9: Wirkende Drehmomente während der Schlagtrajektorie im Experiment 2. Gestrichelte Linien stellen Drehmomentgrenzen dar. Vertikaler Strich markiert Schlagzeitpunkt.

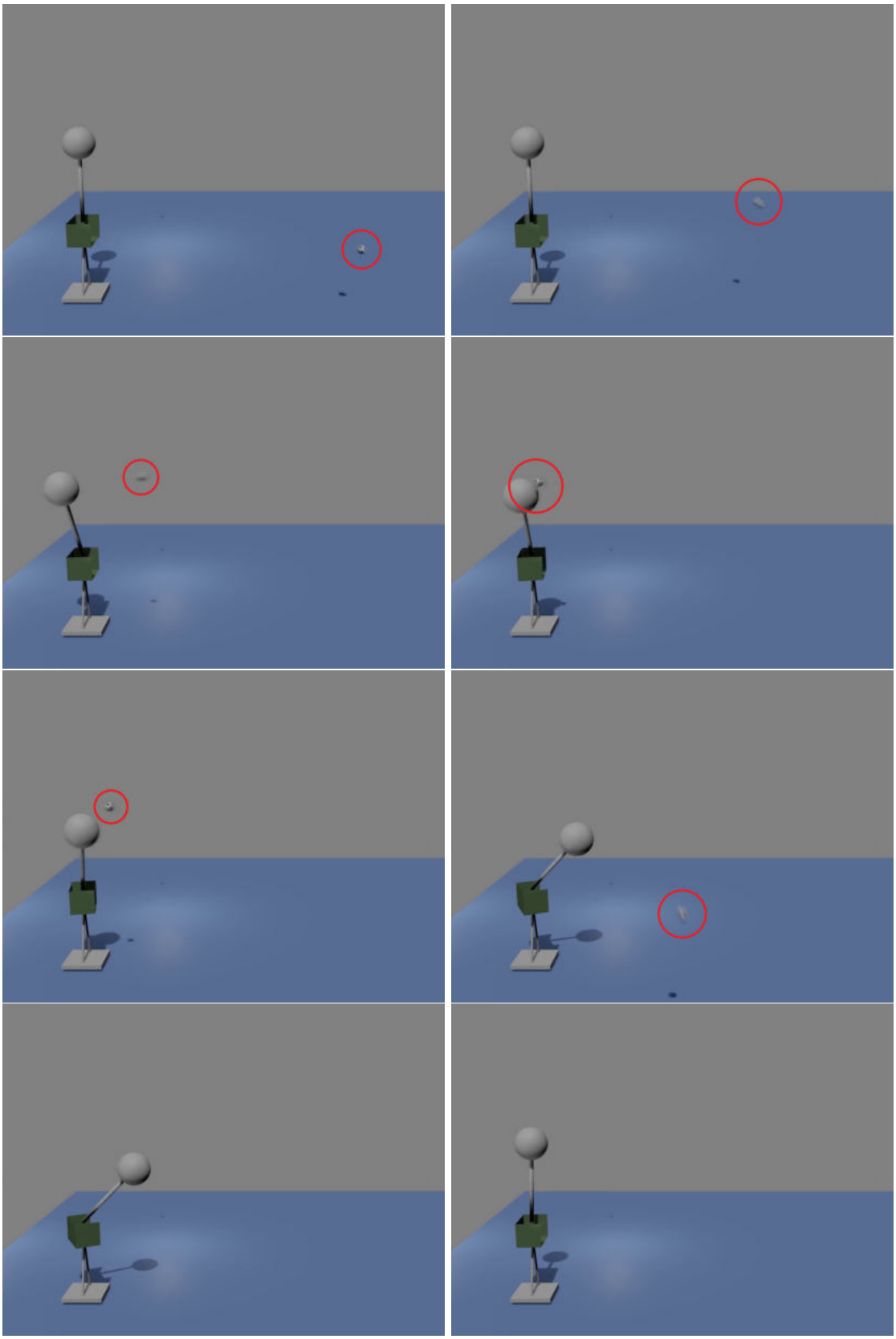

Abbildung 4.10: Simulation des zweiten Experiments in Blender. In Reihenfolge von Links-nach-Rechts und Oben-nach-Unten. Bälle sind zur besseren Visualisierung rot eingekreist. Erstes Bild zeigt Startpunkt. Zweites Bild zeigt Start der Roboterbewegung. Bild 4 zeigt Moment der Kollision. Bild 6 zeigt nächsten Punkt zum Startpunkt. Bild 7 und 8 zeigen Rückkehr des Roboters zur Startposition. Ein Video des Wurfes ist unter folgendem Link zu finden: `https://github.com/ararnice/nn-robotic-task`

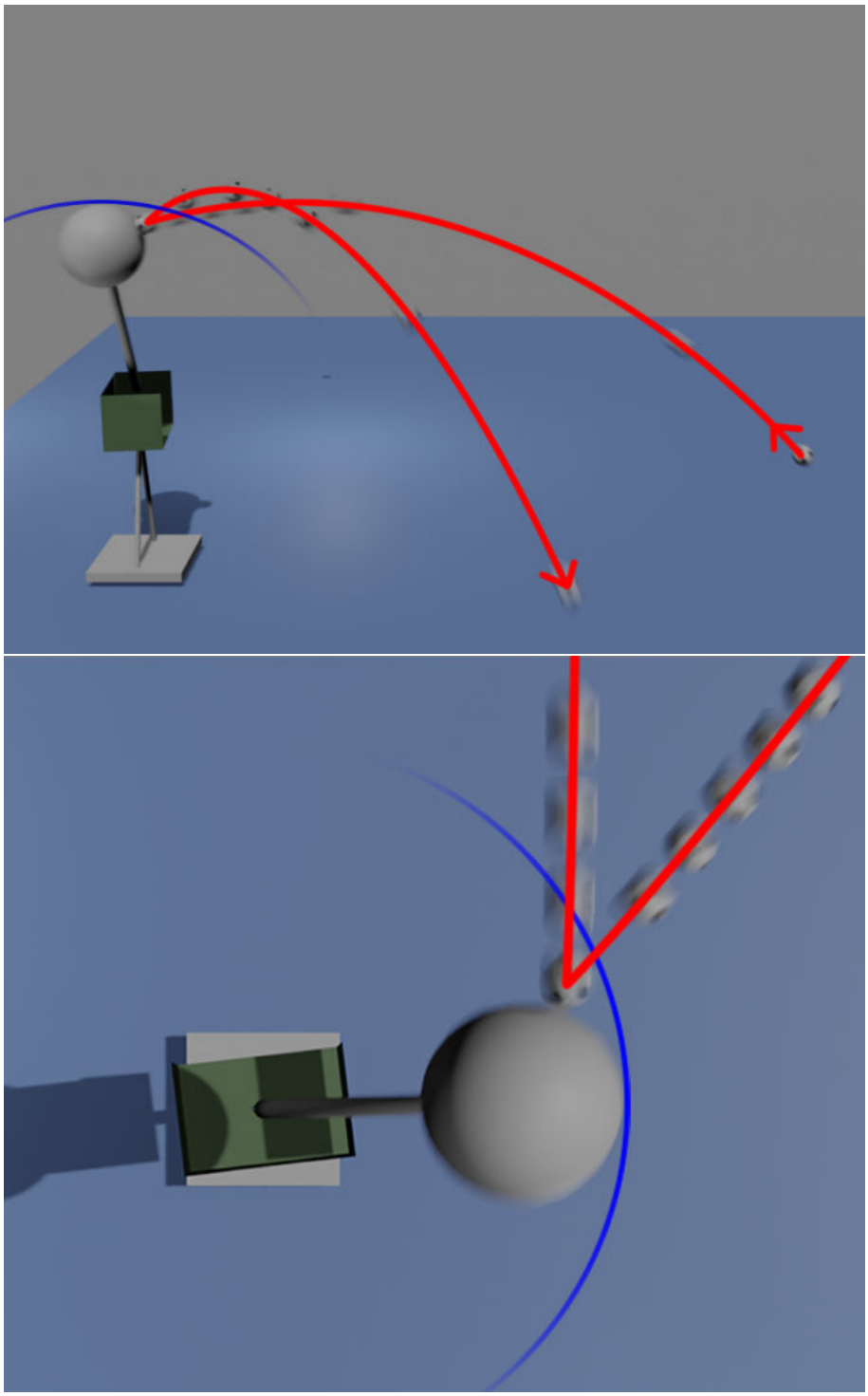

Abbildung 4.11: Einschlags- und Abprallwinkel im 2. Experiment. Überlagerung einiger Ballpositionen, um den Winkel zu verdeutlichen. Rote Linie zeigt Flugbahn des Balles, blaue Linie zeigt möglichen Bewegungsradius des Schlägers.

4.3.2 Quantitative Auswertung

Um eine generelle Aussage über die Leistung des Schlagsystems treffen zu können, wurde eine quantitative Auswertung mittels der Physiksimulation von Blender vorgenommen. Durch eine Generierung von zufälligen Würfen auf den Aktionsradius im Sinne des Zielszenarios können verschiedenste Szenarien erzeugt und ausgewertet werden. Mittels

der Blender Python API können neben einer genauen Kontrolle der Blender Szene auch weitere Bibliotheken eingebunden werden. Durch die Einbindung von TensorFlow in das Steuerungsskript von der Blender Szene kann eine komplette Simulation mit Inferenz des neuronalen Netzes automatisch durchgeführt und ausgewertet werden. Pro simulierten Wurf wird die kürzeste Distanz nach dem Schlag zwischen Ball und Startpunkt des Wurfes ermittelt, bevor der Ball auf den Boden aufgeschlagen ist.

Durch die Auswertung von 25000 simulierten Würfen auf den Aktionsradius des Roboters, mit Berechnung der geringsten Distanz zur Startposition des Wurfes und der genauen Kontrolle über alle Parameter kann eine Aussage über die durchschnittliche Leistung des entwickelten Systems getätigt werden. Die durchschnittliche kürzeste Distanz im Wurf zwischen Startposition des Balles und geschlagenem Ball in der Luft beträgt 2,14 Meter, mit einer Standardabweichung von 0,83 Metern und damit einer Varianz von 0,91 Metern. In Abbildung 4.12 sind alle 25000 Würfe an ihrem Schnittpunkt mit dem Bewegungsradius des Schlägers aufgetragen und in Felder eingeteilt. Die Ansicht zeigt den Roboter von oben, platziert in der Mitte des Bildes, mit eingezeichneten Bewegungsradius des Roboters und der Wurfrichtung der Bälle auf den Roboter. Pro Feld im Bewegungsradius zeigt die Intensität der Farbe die durchschnittliche simulierte Distanz zum Startpunkt der geworfenen Bälle an. Ein Feld mit stärkerer Intensität zeigt eine größere Distanz zum Startpunkt an und kann in eine schlechtere Rückspielleistung des Roboters übersetzt werden. Trotz der Simulation von 25000 Würfen sind einige Felder *unbeworfen* und lassen die Bodenfarbe durchscheinen. Durch die Vogelperspektive des Bildes decken die Felder außen, im Gegensatz zu den Feldern in der Mitte des Bewegungsradius, jeweils einen größeren Bereich ab. Diese Verzerrung liegt an dem halbkugelförmigen Bewegungsradius des Roboters und führt dazu, dass die Werte der äußeren Felder durch eine größere Stichprobenmenge berechnet wurden. Durchschnittlich sind alle Felder, die mindestens einen simulierten Wurf widerspiegeln, im Durchschnitt von 12,7 Bällen beworfen, mit einer Standardabweichung von 2,25 und einer Varianz von 5,1. Der Median der Wurfzahl pro Feld ist ebenfalls 12, die größte Anzahl an Würfen pro Feld ist 39 und die geringste Anzahl beträgt 1 Ball pro Feld. Bei alleiniger Betrachtung von Feldern, bei denen mindestens 6 Würfe pro Feld ausgewertet wurden, steigt die durchschnittlich kürzeste Distanz zwischen Startpunkt eines geworfenen Balles und der nächsten Fangposition des geschlagenen Balles auf 2,22 Meter.

Eine weitere Auswertung mit Blick auf die absoluten Distanzen zur Startposition des Balles gibt Aufschluss, an welchen Positionen im Bewegungsradius definierte Grenzen eingehalten werden können. Abbildung 4.13 zeigt die Felder, in denen eine geschlagene Distanz, damit ein Fangradius in Bezug zur Startposition des Balles, von unter 2 Metern erreicht wird. Dunkler gefärbte Felder zeigen eine höhere Anzahl an Würfen auf dieses Feld, die in diesem Radius bleiben. In absoluten Zahlen sprechen wir hier von 11953 Bällen, die in diesem Radius um ihre Startposition retourniert wurden. Damit wurden 47,8% aller zufällig geworfener Bälle auf dem gesamten Bewegungsradius des Roboters in den definierten Radius retourniert. Abbildung 4.14 zeigt selbige Informationen mit einem definierten Radius von 1,5 Metern um die Startposition. Insgesamt fallen 6458 Bälle, oder 25,83% aller simulierter Würfe in diesen Radius. In Abbildung 4.15 werden die Positionen der Bereiche gezeigt, in denen der retournierte Ball in weniger als 1 Meter um die Startposition gespielt wurde. Insgesamt zählen dazu noch 2352 simulierte Würfe und damit 9,41%. Abbildung 4.16 und 4.17 zeigen jeweils die Felder, in denen unter 0,5 Meter und 0,1 Meter nah an die Startposition heran geschlagen wurde. Für den Radius von 0,5 Meter sind das noch 353 Bälle (1,41%) und für 0,1 Meter Radius 13 Bälle (0,05%).

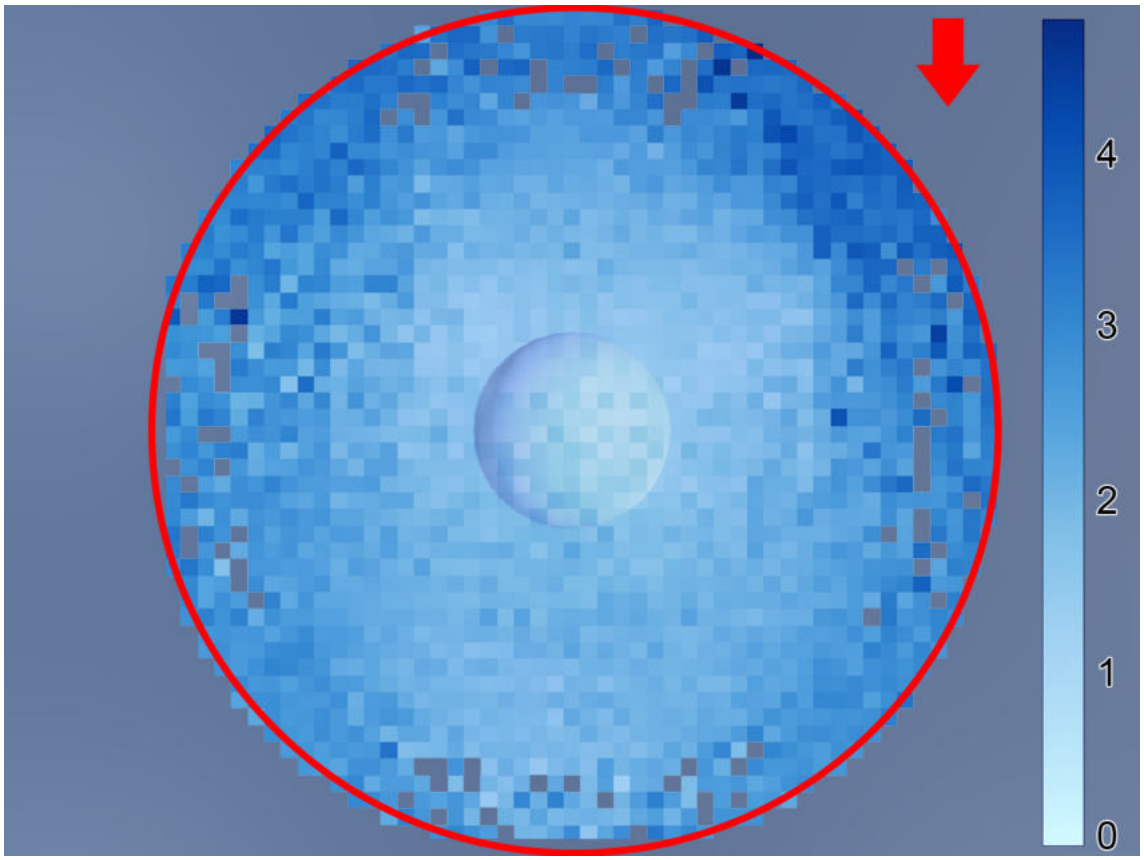

Abbildung 4.12: Durchschnittliche Distanz zum Startpunkt eines geworfenen Balles nach simuliertem Schlag. Mittel nach 25000 simulierten Würfe in den Bewegungsradius des Roboters. Skala zeigt durchschnittliche Distanz in Metern. Bereich in 50x50 Felder eingeteilt. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

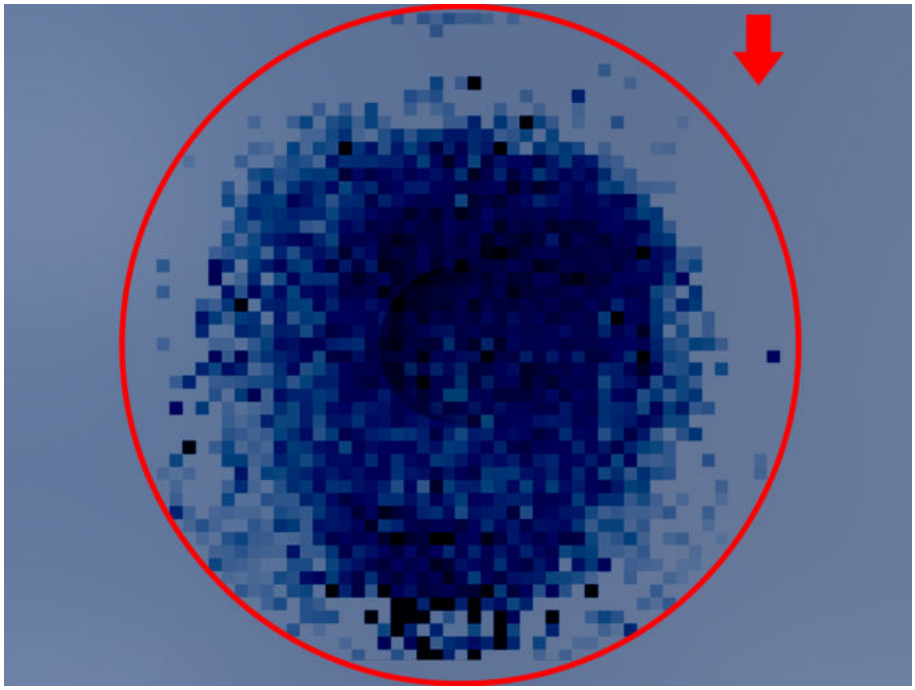

Abbildung 4.13: Simulierte Würfe mit geschlagener Distanz zum Startpunkt eines geworfenen Balles unter 2 Meter. Dunklere Farbe zeigt höhere Anzahl an simulierten Würfen unter 2 Meter retournierter Distanz in dieses Feld. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

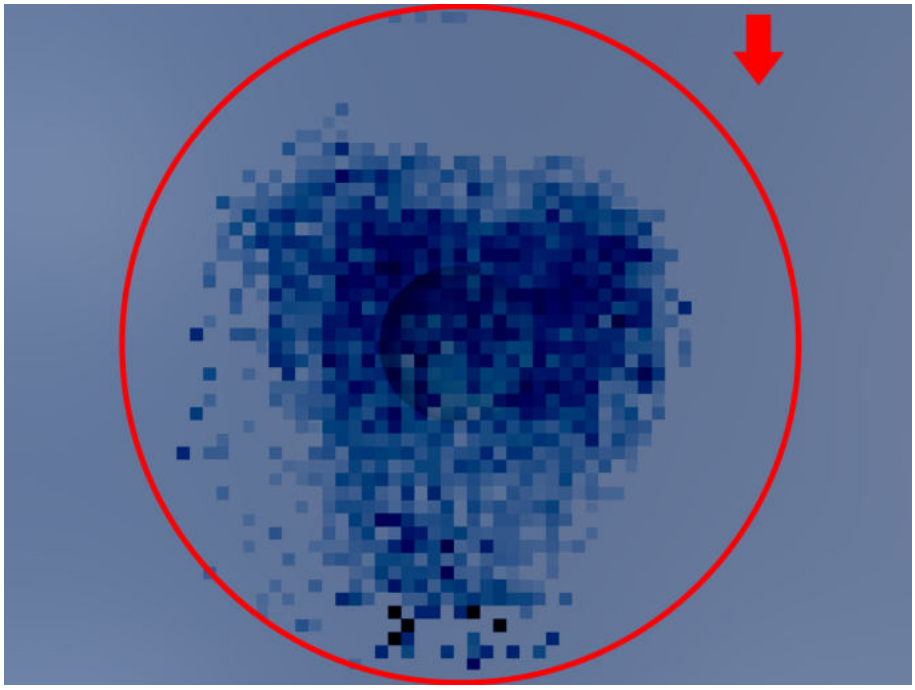

Abbildung 4.14: Simulierte Würfe mit geschlagener Distanz zum Startpunkt eines geworfenen Balles unter 1,5 Meter. Dunklere Farbe zeigt höhere Anzahl an simulierten Würfen unter 1,5 Meter retournierter Distanz in dieses Feld. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

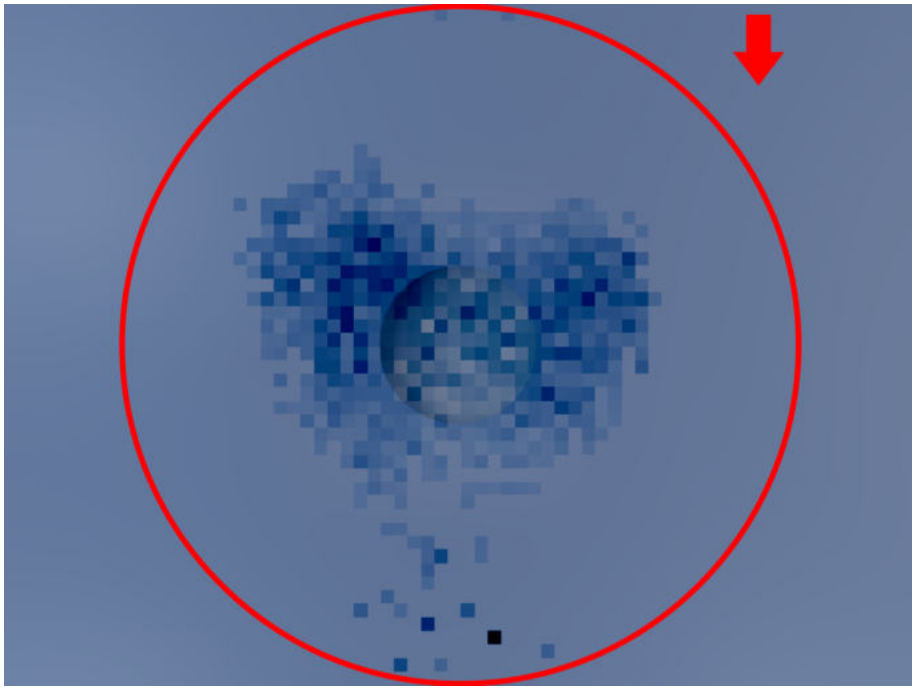

Abbildung 4.15: Simulierte Würfe mit geschlagener Distanz zum Startpunkt eines geworfenen Balles unter 1 Meter. Dunklere Farbe zeigt höhere Anzahl an simulierten Würfen unter 1 Meter retournierter Distanz in dieses Feld. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

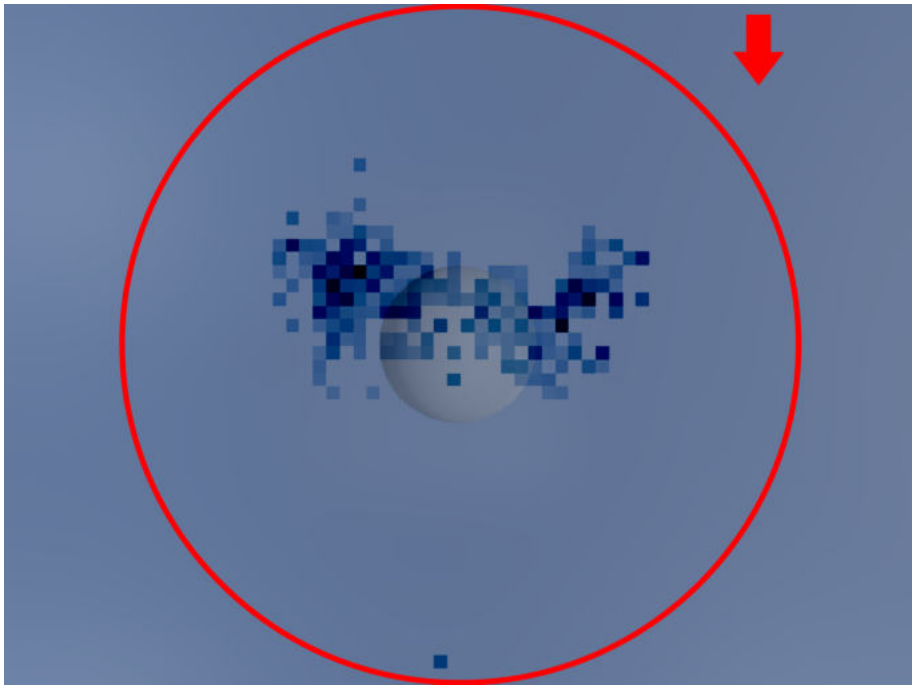

Abbildung 4.16: Simulierte Würfe mit geschlagener Distanz zum Startpunkt eines geworfenen Balles unter 0,5 Meter. Dunklere Farbe zeigt höhere Anzahl an simulierten Würfen unter 0,5 Meter retournierter Distanz in dieses Feld. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

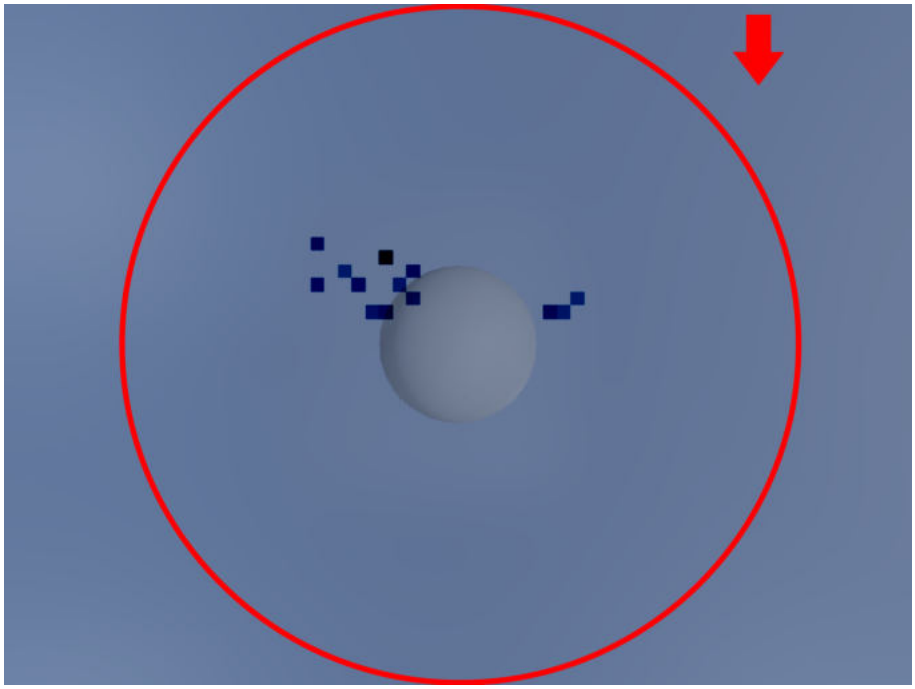

Abbildung 4.17: Simulierte Würfe mit geschlagener Distanz zum Startpunkt eines geworfenen Balles unter 0,1 Meter. Dunklere Farbe zeigt höhere Anzahl an simulierten Würfen unter 0,1 Meter retournierter Distanz in dieses Feld. Roter Kreis zeigt Bewegungsradius des Roboters, Pfeil zeigt Wurfrichtung auf den Roboter.

4.4 Anwendung im Robotersystem

Eine Anwendung des erarbeiteten Systems im Roboter *Doggy* ist, im Gegensatz zum System der Ballerkennung, durchaus mit den aktuellen Hardwarespezifikationen denkbar. Mit der Prozessorleistung des Evaluationssystems, beschrieben in Kapitel 3.4, dauert eine Inferenz ohne besonderer Hardwarebeschleuniger 1,4 Millisekunden und ein gesamter Prozess mit Trajektoriengenerierung für den Schlag 2,3 Millisekunden. Diese Ausführungszeiten lassen mit den aktuellen Spezifikationen des Steuerungssystems des Roboters auf eine nutzbare Geschwindigkeit der Berechnungen schließen. Des Weiteren ist diese Berechnung nicht zeitkritisch, da diese, in aktueller Form, nicht mehrere Male pro Wurf stattfinden muss. Mit der Einschränkung des Start- und Endzustandes des Roboters in der aktuellen Form, ist eine Optimierung der Bewegungstrajektorie mit genaueren Messungen der Flugbahn des Balles während des Wurfes nur mit zusätzlichen Rückrechnungen und Überschreibungen der angestrebten Bewegungstrajektorie möglich. Um dieses System in den Ablauf des Roboters zu integrieren ist, trotz modular gewählten Übergabestellen der Daten zwischen bestehenden System und neuer Komponenten, ein erheblicher Integrationsaufwand nötig.

5 Diskussion der Ergebnisse

Wurde ein Ball im Sinne des Zielszenarios in die Richtung des bestehenden Roboters *Doggy* geworfen, so wurde dieser bereits im bestehenden System erkannt und die Flugbahn hinreichend korrekt vorhergesagt, um einen Kontakt mit dem Ball zu erzeugen. Dies zeigen die Experimente von Schüthe [19]. Das bestehende System nutzt einen Kreiserkenner, um fliegende Bälle auf Bildern zu erkennen. Auf den aufgenommenen Testdaten ist damit eine erfolgreiche Erkennung von ungefähr der Hälfte aller Bälle möglich. Das entwickelte neuronale Netz für die Perzeption kann, gegenüber dem bestehenden System, Bälle in ihrem Kontext und ihrer visuellen Erscheinung zuverlässig erkennen. Anstelle der Erkennung von radialen Gradienten auf visuell transformierten Bildern kann das verwendete neuronale Netz richtige Zusammenhänge lernen. Ein Beispiel dafür ist die Bewegungsunschärfe, bei der sich die Informationen über die Existenz eines Balles außerhalb des eigentlich zu segmentierenden Gebiets befinden. Diese komplexen, extrahierten Informationen zurück auf einen segmentierten Bildausschnitt zu projizieren, kommen jedoch mit dem Preis eines viel höheren Berechnungsaufwands. Das neuronale Netz erreicht damit eine Verbesserung von über 50%. Trotz der vielfältigen Möglichkeiten der Erkennung komplexer Merkmale von geworfenen Bällen können nur Merkmale erkannt werden, die sich in irgendeiner Form in den aufgenommenen Daten befinden. So können komplett überblendete Bälle mit extrem hohen Kontrast vor einem ebenfalls überblendeten Hintergrund so mit dem Hintergrund verschmelzen, dass diese selbst mit Annotation nicht von anderen Objekten oder Artefakten unterschieden werden können. Das Zielszenario definiert den mobilen und variablen Einsatz des Systems, der durch die Nutzung mehrerer Merkmale zur Erkennung von Bällen durchaus erweitert wird. Im Zuge der Evaluation in dieser Thesis ist dies jedoch weniger betrachtet wurde. Der Fokus der Entwicklung lag dabei mehr auf der variablen und mächtigen Datengenerierung. Innerhalb der Evaluation ist dieser Teil nur durch ein ungesehenen, realen Testdatensatz, nicht jedoch durch viele verschiedene Szenarien überprüft wurden. Die bestehende Rechenkomponente des Systems ist nicht auf die Nutzung von neuronalen Netzen ausgelegt worden. Bei der Entwicklung der bestehenden Komponenten wurde der Fokus auf die Anschaffung solider und kostengünstiger Hardware gelegt. Eine Modernisierung des Robotersystems kann allerdings modular, oder in einer weiteren Revision, einfach erfolgen.

Die erarbeitete Schlagbewegung durch die Nutzung von Splines mit Parametern aus der Ausgabe eines neuronalen Netzes scheint effektiv und einfach zu funktionieren. Wird ein Ball grob in die Richtung des Schlägers geworfen, so kann man von einer Retoure des Balls ausgehen und sich auf ein Fangen im Radius von unter 2 Meter bereit machen. Beim Wurf in den Randbereich des Aktionsradius ist dabei eher mit einem schlechteren Rückspiel zu rechnen. In der Evaluation der retournierten Würfe werden die Angaben stark durch die Auswertung des gesamten Aktionsradius gedrückt. Durch ein genaueres Spiel nah an den Endeffektor des Roboters ist auch eine genauere Retoure des Balles zu erwarten. Die visualisierten Bereiche des Zurückspielens zeigen auch, dass bei einem Schlag in der Nähe der Ausgangsposition mehr Zeit und Bewegung in den Aufbau von Schwung gesteckt werden kann, als bei einem Schlag weiter von der neutralen Position entfernt. Bei vielen Würfen an die Grenze des Bewegungsradius wird eine Kollision erreicht, nur ist eine effektive Schlagbewegung ohne das Überschreiten der physischen Drehmomentgrenzen dort nicht möglich. Eine simulierte Erhöhung während der Entwicklung der Drehmomentgrenzen ließ

Schläge viel effektiver und präziser aussehen und könnte die Leistung der Schlagbewegung stark erhöhen. Gleichzeitig wird bei höheren Drehmomentgrenzen der Möglichkeitsraum eines Schlags erweitert und eine präzisere oder spektakuläre Bewegung wäre möglich.

6 Fazit und Ausblick

Im Rahmen dieser Masterarbeit wurde der Einsatz von neuronalen Netzen in zwei entwickelten Systemen am Unterhaltungsroboter *Doggy* evaluiert. Dabei wurde ein System für die Perzeptionskomponente und ein System für die Erzeugung einer geeigneten Schlagbewegung entwickelt. Dabei wurden nur synthetische Daten für das Training dieser Systeme verwendet, sodass eine Entwicklung auf unbekanntem Terrain simuliert wurde. Für die Perzeptionskomponente wurde zuerst eine geeignete Erkennungsmethode entwickelt, die zum bestehenden System passt und sich leicht integrieren lässt. Für die Generierung von Trainingsdaten wurde ein hoch variabler Prozess entwickelt, der mit eigens erzeugten Bällen und KI generierten Hintergrundbildern durch die Nutzung verschiedenster Techniken sehr realistische Trainingsdaten mit perfekter Annotation generiert. Der Generierungsprozess der Daten wurde direkt in das Training eines neuronalen Netzes integriert und ein Netz für eine Auswertung der Methode trainiert. Durch eine Datensammlung von Bildern zu Beginn der Analyse des bestehenden Systems konnte eine Auswertung des entwickelten Systems an echten Kamerabildern vorgenommen werden. Sowohl eine qualitative, als auch eine quantitative Auswertung zeigt eine sehr gute Erkennungsleistung zwischen 86% und 94%, basierend auf den herrschenden Extrembelichtungen. Gegenüber dem bestehenden Erkennungssystem wird dabei eine Verbesserung von 51% erreicht. Eine Integration des erarbeiteten Systems in das bestehende Robotersystem ist derzeit nicht möglich, da in aktueller Form die Rechenleistung nicht ausreicht, um eine echtzeitfähige Erkennung zu gewährleisten.

Für die Entwicklung eines Systems zur Trajektoriengenerierung einer Schlagbewegung wurde zu Beginn das Verhalten des bestehenden Systems analysiert, um eine spätere Integration zu ermöglichen. Basierend auf ermittelten Daten wurde ein Trainingsprozess entwickelt, der insbesondere ohne Grundwahrheit eine Ausgabe von Parametern zur Generierung von Schlagtrajektorien lernen lässt. Dabei werden physische Grenzen und zu erledigende Aufgaben als Verlustfunktionen definiert, um ein Lernverfahren mit automatischer Differenzierung und gewöhnlichem Gradientenabstieg zu ermöglichen. Eine Integration und Auswertung am realen Roboter wurde durch eine Simulation des Szenarios ersetzt, sodass eine Auswertung der durchschnittlichen Rückspielleistung über 25000 simulierten Würfen erstellt werden konnte. In über 47% aller simulierter Würfe kann der Ball nach dem Schlag in einer Distanz von unter 2 Metern gefangen werden, in knapp 10% der Würfe ist ein Zurückspielen mit Radius von 1 Meter an die Abwurfposition möglich.

Durch die Definition des Zielszenarios ist eine sportlich angehauchte Mensch-Roboter Interaktion mit *Doggy* angestrebt. Die erreichten Radien des Zurückspielens laden zur sportlichen Aktion des Spielenden auch außerhalb der Armspannweite ein. Bei genaueren Würfen des Balls in die Nähe des Roboters wird die Genauigkeit des Zurückschlagens besser. Durch die Einordnung des Systems in den Bereich der Unterhaltungsrobotik stimmen die Ergebnisse jedoch mit der Idee und der Geschichte des Roboters überein.

Durch die fehlende Integration der entwickelten Systeme in den Roboter bleiben mindestens zwei Fragen offen: Kann die aufwändigere Ballerkennung dem bestehenden System zu einer sichereren und konstanteren Flugkurven-Vorhersage verhelfen und damit den Anwendungsbereich des Roboters vielseitiger gestalten? Ist durch die Integration

der gezielten Schlagbewegung ein unterhaltsames Zusammenspiel zwischen Mensch und Maschine möglich? Abseits der Kompatibilität von Roboter und entwickelter Systeme ist bei der Ballerkennung eine Einordnung verschiedener neuronaler Netze in Größe und Form mit Training auf den gleichen synthetisierten Daten sehr interessant. Dabei ist die Erkennungsleistung im Zusammenhang mit der Ausführungszeit für diesen Anwendungsfall zu evaluieren. Gleichzeitig bietet die entwickelte Methode der synthetischen Generierung ein gutes Rahmenwerk für Anwendungen, in denen sehr wenig Daten des Szenarios vorhanden sind. Auch die Nutzung der neuronalen Netze als Optimierungsframework für robotische Trajektorienplanung ist grundsätzlich entwickelt, jedoch ohne viel Variation in Netzstruktur oder Art implementiert. Mit den Verlustfunktionen für die Roboterdynamik und der aufgabenspezifischen Verlustfunktion in Bezug auf das Zurückspielen und das Erfüllen des Zielszenarios kann die Methodik durchaus für das Lernen von geeigneten Trajektorien verwendet werden. Jedoch steckt noch viel Verbesserungspotenzial in der Verlustberechnung der Aufgabenerfüllung durch wirklich optimales Zurückspielen des Balles oder der Erweiterung durch die Nutzung fortlaufender Sensorinformationen während des Schlages.

Literaturverzeichnis

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] O. Birbach. Tracking and calibration for a ball catching humanoid robot. 2015.

[3] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *International Symposium on System Integration (SII)*, 2019.

[4] E. E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.

[5] F. Chollet et al. Keras, 2015.

[6] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[7] C. de Boor. *A Practical Guide to Spline*, volume Volume 27. 01 1978.

[8] M. Elsisi, K. Mahmoud, M. Lehtonen, and M. M. F. Darwish. An improved neural network algorithm to efficiently track various trajectories of robot manipulator arms. *IEEE Access*, 9:11911–11920, 2021.

[9] S. Enderes. The impact of annotation errors on neural networks, Jun 2021.

[10] W. J. Gordon and R. F. Riesenfeld. Bernstein-bézier methods for the computer-aided design of free-form curves and surfaces. *J. ACM*, 21(2):293–310, apr 1974.

[11] T. Hammer. Aufbau, Ansteuerung und Simulation eines interaktiven Ballspielroboters. Master's thesis, Universität Bremen, Bremen, 2011.

[12] A. Hasselbring, U. Frese, and T. Röfer. Learning optimal robot ball catching trajectories directly from the model-based trajectory loss. In G. Gini, H. Nijmeijer, W. Burgard, and D. P. Filev, editors, *Proceedings of the 19th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2022, Lisbon, Portugal, July 14-16, 2022*, pages 201–208. SCITEPRESS, 2022.

[13] J. Kim, S. Kim, and M. Lee. Convolutional neural network with biologically inspired on/off relu. In *Neural Information Processing: 22nd International Conference, ICONIP 2015, November 9-12, 2015, Proceedings, Part IV 22*, pages 316–323. Springer, 2015.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[15] A. Kupferberg, S. Glasauer, M. Huber, M. Rickert, A. Knoll, and T. Brandt. Biological movement increases acceptance of humanoid robots as human partners in motor interaction. *AI Society*, 26:339–345, 11 2011.

[16] T. Laue, O. Birbach, T. Hammer, and U. Frese. An entertainment robot for playing interactive ball games. pages 171–182, 01 2014.

[17] P. R. Lipow and I. Schoenberg. Cardinal interpolation and spline functions. iii. cardinal hermite interpolation. *Linear Algebra and its Applications*, 6:273–304, 1973.

[18] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[19] D. Schüthe. *Dynamic Bat-Control of a Redundant Ball Playing Robot*. PhD thesis, 02 2017.

[20] F. sheikhnezhad fard, P. Hollensen, S. Mcilory, and T. Trappenberg. Impact of biased mislabeling on learning with deep networks. 05 2017.

[21] G. Van Zandycke and C. De Vleeschouwer. 3d ball localization from a single calibrated image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3472–3480, June 2022.

[22] M. W. Walker and D. E. Orin. Efficient Dynamic Computer Simulation of Robotic Mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211, 09 1982.

[23] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.

[24] G. V. Zandycke and C. D. Vleeschouwer. Real-time cnn-based segmentation architecture for ball detection in a single view setup. *CoRR*, abs/2007.11876, 2020.