Masterarbeit

Gelenkwinkelvorhersage für Laufbewegungen humanoider Roboter mittels neuronaler Netze

Jan Fiedler

Matrikelnummer: 4235072

12. Mai 2023

1. Gutachter: Dr. Tim Laue
2. Gutachter: Prof. Dr.-Ing. Tanja Schultz

Fachbereich 3: Mathematik und Informatik
Studiengang Informatik

Universität Bremen

Jan Fiedler

Gelenkwinkelvorhersage für Laufbewegungen humanoider Roboter mittels neuronaler Netze

Joint Angle Prediction for Walking Motions of Humanoid Robots Using Neural Networks

Masterarbeit, Fachbereich 3: Mathematik und Informatik
Universität Bremen, 12. Mai 2023

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt und nicht anderweitig zu Prüfungszwecken vorgelegt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 12. Mai 2023

.....

(Jan Fiedler)

Zusammenfassung

Ein häufiges Problem bei der Bewegungssteuerung von Robotern im geschlossenen Regelkreis ist die Verzögerung der Motorreaktion. Zum Beispiel bei der Berechnung eines Gehschritts für einen humanoiden Roboter stammt der verfügbare Zustand der Sensorgelenke aus der Vergangenheit und die neu berechneten Motorpositionen werden nicht sofort umgesetzt. Dies kann zu einer suboptimalen Bewegungsplanung führen, insbesondere bei schnellen und dynamischen Bewegungen.

Der humanoide Roboter NAO hat, wie in anderen Arbeiten nachgewiesen, eine solche Verzögerung. In dieser Arbeit wird untersucht, ob und wie mittels neuronaler Netze die Gelenkwinkel auf dem Roboter vorhergesagt werden können, um der Verzögerung entgegenzuwirken. Als Grundlage dienen Daten, welche auf Wettbewerben des Teams B-Human aufgenommen, in der Arbeit aufbereitet und zu Datensätzen verarbeitet wurden.

Dazu wird das Problem in mehrere Teilprobleme unterteilt. So wird zuerst geklärt, ob und wie die Gelenkwinkel vorhergesagt werden können. Anschließend wird überprüft, ob ein gemeinsames Modell für alle Roboter verwendet werden kann und welche Auswirkungen leicht unterschiedliche Untergründe auf die Vorhersage haben. Außerdem wird die Auswirkung der Abnutzung der Roboter untersucht. Abschließend wird untersucht, wie sich die Ergebnisse auf Robotern verwenden lassen.

Inhaltsverzeichnis

1 Einleitung

Für autonome humanoide Roboter ist eine sichere Fortbewegung von zentraler Bedeutung. Dazu müssen zunächst die Umgebung und der aktuelle Zustand des Roboters möglichst zeitnah und akkurat wahrgenommen werden. Diese Wahrnehmung muss dann in Echtzeit verarbeitet und in eine Ansteuerung für die Motoren umgesetzt werden, um ein stabiles Laufen zu ermöglichen, da jeder Schritt permanent angepasst werden muss. Gibt es Verzögerungen im System, führt dies zu einer Verschlechterung der Laufstabilität. Weiterhin leidet die Reaktionsfähigkeit, sodass auf auftretende Unebenheiten, Hindernisse oder andere äußere Einflüsse möglicherweise nicht rechtzeitig reagiert werden kann.

Der humanoide Roboter *NAO* hat eine solche nachgewiesene Verzögerung vom Anfragen der Gelenkwinkel bis zum Messen der Auswirkungen (Böckmann 2015, S. 7–10; Reichenberg 2021, S. 19–20). Deshalb soll im Kontext dieser Arbeit eine Vorhersage der Gelenkwinkel mit neuronalen Netzen untersucht werden, um somit einigen Effekten der Verzögerung entgegenwirken zu können und zukünftig ein stabileres Laufen zu ermöglichen.

1.1 Hintergrund

Die Arbeit ist im Kontext des studentischen Projektes *B-Human* der *Universität Bremen* in Kooperation mit dem *Deutschen Forschungszentrum für Künstliche Intelligenz* (DFKI) entstanden. B-Human tritt als Team seit 2009 regelmäßig bei den internationalen Forschungswettbewerben der *RoboCup Federation* an (B-Human 2023).

1.1.1 Der RoboCup

Die Wettbewerbe wurden Mitte der neunziger Jahre zur Förderung der Forschung in den Bereichen Robotik und künstliche Intelligenz ins Leben gerufen (Kitano u. a. 1995). Seit 1997 finden diese – mit Ausnahme von 2020 und teilweise 2021 – jährlich statt, um durch die Bereitstellung eines Standardproblems Theorien, Algorithmen und Architekturen über die Jahre zu vergleichen und evaluieren (The RoboCup Federation 2023a,c). Die Wettbewerbe sind dabei in verschiedene Ligen unterteilt, die sich mit Fußball, aber unter anderem auch mit Logistik oder Mensch-Maschine-Interaktion beschäftigen.

B-Human nimmt in der *Soccer Standard Platform League* (SPL) teil. In dieser Liga spielen baugleiche Roboter – welche nicht modifiziert werden dürfen – gegeneinander Fußball. Sie ist somit ein reiner Softwarewettbewerb, der aber gleichzeitig mit den Herausforderungen echter Hardware umgehen muss. Wie in Abbildung 1.1 zu sehen, treten nach bisherigen Regeln (RoboCup Technical Committee 2022) zwei Teams von je fünf Robotern vom Typ NAO (siehe Abschnitt 3.1) auf einem 9 m × 6 m großen Kunstrasenfeld in zwei zehnminütigen Halbzeiten gegeneinander an.

1.1.2 Die Datengrundlage

Die Roboter von B-Human schreiben während des Spielens eingehende Sensordaten und größere Teile ihres internen Zustandes in *Logdateien* (Logs). Ein Log ist eine Binärdatei in einem B-Human spezifischen Format. Dieses kann im eigenen Simulator abgespielt werden (B-Human 2022). Zu Beginn der Arbeit standen für den aktuellen Roboter die Daten von Wettbewerben aus dem Jahr 2019 zur Verfügung, da in den Jahren 2020 und 2021 – aufgrund der Corona Pandemie – keine offiziellen Wettbewerbe mit Fußballspielen stattfanden. Im Verlauf der Arbeit kamen die Daten aus dem Jahr 2022 hinzu. In beiden Jahren hat B-Human jeweils an zwei Wettbewerben teilgenommen, der deutschen Meisterschaft (German Open) und der Weltmeisterschaft (RoboCup) (siehe The RoboCup Federation 2023b). Es gibt somit Daten mit einem Entwicklungszeitraum von 3 Jahren. Außerdem ist der Zustand der Roboter unterschiedlich. Während 2019 weitestgehend neuwertige Roboter spielten, sind diese 2022

Abbildung 1.1: Beispiel eines Fußballspiels in der SPL aus dem Jahr 2019. Bild von Tim Laue (CC BY-SA 4.0).

deutlich abgenutzter. Eine genaue Beschreibung der Daten und den daraus resultierenden Datensätzen ist in Abschnitt 3.2 zu finden.

1.1.3 Die B-Human-Software

Das Team B-Human benutzt seine eigene Software, welche Jahr für Jahr weiterentwickelt und veröffentlicht wird (B-Human Team 2022). Aktuell gibt es in dieser keine Vorhersage der Gelenkwinkel. Es werden die aktuellen Messungen der mitgelieferten Motoransteuerung verwendet. Allerdings gibt es bereits einige Schätzungen, welche Teile der Auswirkungen der Verzögerung abfedern. So wird beim Laufen auf Basis der Gyrometer-Daten balanciert, um dieses zu stabilisieren. Seit 2022 gibt es zusätzlich eine Fußwechselvorhersage von einem Zeitschritt, welche versucht den Zeitpunkt des Schrittwechsels besser

vorherzusagen. Außerdem werden unter anderem Kalmanfilter verwendet, um den Zustand des Roboters zu modellieren und beispielsweise die Lage des Roboters zu schätzen oder ein Umfallen zu erkennen (Röfer, Laue, Ewers u. a. 2022). Ein detaillierter Überblick über die Unterschiede zwischen 2019 und 2022 ist in Abschnitt 5.4 zu finden.

1.2 Motivation

Damit ein humanoider Roboter effektiv Fußball spielen kann, ist ein schnelles, stabiles und präzises Laufen von hoher Bedeutung. Dazu müssen die Umgebung und der aktuelle Zustand des Roboters möglichst zeitnah und akkurat wahrgenommen, in Echtzeit verarbeitet und in eine neue Ansteuerung für die Motoren umgesetzt werden. Die aktuelle Generation des verwendeten Roboters hat mit einer 83 Hz Taktung einen Zeitschritt alle 12 ms (SoftBank Robotics 2023c). Für eine frühere Generation des NAO hat Böckmann (2015, S. 7–10) nachgewiesen, dass es eine Verzögerung von mehreren Zeitschritten, vom Anfragen der Gelenkwinkel bis zum Messen der Auswirkungen gibt, die sogenannte Totzeit. Außerdem werden die Eingaben von der Herstellersoftware im Roboter noch weiter verarbeitet, gefiltert und sind von der Motortemperatur abhängig, wodurch die gesetzten Gelenkwinkel nicht immer präzise erreicht werden. Die Verzögerung wurde von Reichenberg (2021, S. 19–20) auch für den aktuellen NAO[6] untersucht. Dazu wurden in einem Experiment exemplarisch sechs Beingelenke eines in der Luft gehaltenen Roboters einzeln aus einer ruhenden Position angesteuert. Sie wurden jeweils zweimal mit einer kurzen Pause bewegt und die Verzögerung vom Anfragen der Gelenkwinkel bis zur Messung der Auswirkungen gemessen. Dabei wurde eine Verzögerung von bis zu 36 ms, also bis zu drei Zeitschritten festgestellt.

Es ist somit wünschenswert, die Verzögerung zu reduzieren, um damit das Laufen verbessern zu können. Da es sich beim Laufen um eine grundsätzlich gleichmäßige periodische Bewegung handelt, kann diese vorhergesagt werden. Eine Möglichkeit dazu stellt die Vorhersage von zu erwartenden Gelenkwinkeln auf Grundlage der letzten Sensorwerte und der angefragten Gelenkwinkel dar, um bereits zum aktuellen Zeitpunkt auf erwartbare Ereignisse reagieren zu

können und diesen somit vorzubeugen. Die Vorhersage kann zwar keine äußeren Einflüsse vorhersagen, kann aber aus der kürzeren Vergangenheit Schlüsse darauf zulassen, was passiert, bis die aktuelle Anfrage umgesetzt ist, damit diese direkt angepasst werden kann. Somit könnten geplante Gelenkwinkel auch tatsächlich erreicht und somit das Laufen verbessert werden.

Außerdem steht mit den in Abschnitt 1.1.2 beschriebenen Daten bereits eine größere Datengrundlage zur Verfügung, welche nicht extra gelabelt werden muss, da die zukünftigen Gelenkwinkel bereits Teil der Daten sind. Diese müssen somit aufbereitet werden und können dann verwendet werden.

1.3 Zielsetzung

Basierend auf der bekannten Verzögerung der Roboter soll mit bestehenden Daten eine Vorhersage für den NAO^6 untersucht werden. Daraus ergeben sich die grundlegenden Fragen, ob und wie die Gelenkwinkel für einen Roboter am effizientesten vorhergesagt werden können und falls es funktioniert, wie gut es ist. Bei der Beantwortung dieser Fragen ergeben sich weitere Fragestellungen:

- Lassen sich die Vorhersagen auf verschiedene NAO^6 Roboter übertragen?
- Wie stark ist der Einfluss des Untergrundes auf das Laufen und somit auf die Vorhersage?
- Wie stark beeinflusst die Abnutzung der Roboter das Laufen und somit die Vorhersage?
- Ist die Vorhersage schnell genug, um auf Robotern in Spielen eingesetzt werden zu können?

Als Methode sollen dazu neuronale Netze verwendet und mit diesen Fragen unter anderem geklärt werden, ob ein Netz für alle Roboter und Untergründe verwendet werden kann oder ob Unterscheidungen notwendig sind.

Hierzu wird eine Reihe an Experimenten durchgeführt, welche unterschiedliche Ansätze für die Vorhersage untersuchen und gegen Referenzimplementierungen verglichen werden. Anschließend wird analysiert, wie gut der Ansatz auf unterschiedliche Gegebenheiten generalisiert. Abschließend soll dies auf dem Roboter angewendet werden, um eine Verwendung der Ergebnisse zu ermöglichen.

1.4 Aufbau der Arbeit

Im folgenden Kapitel 2 werden verwandte Arbeiten und klassische Ansätze vorgestellt. Anschließend wird in Kapitel 3 der Roboter und der Datensatz erläutert, bevor in Kapitel 4 die Implementierung beschrieben ist. Darauf folgen in Kapitel 5 die Experimente, die im Rahmen dieser Arbeit durchgeführt wurden und in Kapitel 6 ist dann die Anwendung der Erkenntnisse vorgestellt. Abschließend wird in Kapitel 7 ein Fazit gezogen und es werden Anregungen für Verbesserungen sowie die Weiterentwicklung gegeben.

2 Verwandte Arbeiten

In diesem Kapitel werden in Abschnitt 2.1 Arbeiten vorgestellt, die sich mit verwandten Problemen beschäftigen. Anschließend werden in Abschnitt 2.2 klassische Methoden zur Vorhersage von Zeitreihen vorgestellt. Hierbei handelt es sich jeweils um eine kleine Auswahl relevanter Arbeiten.

2.1 Verwandte Probleme

Böckmann und Laue (2017) haben sich im Kontext des Ballschusses mit einer älteren Version des NAO auch mit der Verzögerung beschäftigt. Sie benutzen ein mathematisches Modell, genauer ein dynamisches System zweiter Ordnung, basierend auf einem Masse-Feder-Dämpfer, um die Motorposition vorherzusagen. Dieses wurde anschließend durch ein auf der Stelle Laufen evaluiert. Für diese vereinfachte Bewegung konnte ein Fehler von ungefähr 0,181° erzielt werden. Es wurde somit das Motorverhalten besser modelliert, indem die Verzögerung mit eingerechnet wird. Ein Problem dieses Ansatzes ist, dass das eigentliche Laufen in Spielen und somit das tatsächliche Verhalten der einzelnen Gelenke nicht einzeln betrachtet wurde. Es wurde also lediglich die generelle Trägheit des Systems für die Motoren modelliert.

Ein leicht verwandtes Thema wird zum Beispiel auch von **Seekircher und Visser (2017)** behandelt, welche Laufen für den NAO mit *Covariance Matrix Adaption-Evolutionary Strategy* (CMA-ES) (Hansen, Müller und Koumoutsakos 2003) optimieren. Hier wird eine Laufbewegung vorgestellt, die auf einem linearen Inverses-Pendel-Modell basiert. Dessen Parameter werden online an das Verhalten des Roboters angepasst und somit implizit die Verzögerung behandelt.

Behnke u. a. (2004) und Gloye (2005) haben sich in der *Small Size League* (SSL), einer anderen Fußball Liga im RoboCup, auch mit einem ähnlichen Problem beschäftigt. In dieser spielen fahrende Roboter gegeneinander Fußball, wobei das Feld von einer externen Kamera aufgenommen wird und ein externer Computer dieses auswertet und die Roboter steuert, was zu Verzögerungen führt. Dort ist die Verzögerung zwischen dem Abfragen einer Bewegung und dem dazugehörigen Kamerabild etwa vier Zeitschritte (ungefähr 132 ms). Es wurde ein neuronales Netz trainiert, welches den Unterschied zwischen den aktuellen Daten und denen in vier Zeitschritten vorhersagt. Dazu werden weiterverarbeitete, gemessene und gesendete Daten aus den letzten sechs Zeitschritten benutzt. Bei den Daten handelt es sich jeweils um die Differenzen der Koordinaten und Rotationen. Das neuronale Netz ist mit 42 Eingabeneuronen, 10 verdeckten Neuronen und 4 Ausgabeneuronen recht einfach gehalten. In den Trainingsdaten gibt es keine Kollisionen, da diese Informationen nicht in das neuronale Netz übergeben und somit nicht modelliert werden können. Mit dem trainierten neuronalen Netz konnte die Verzögerung kompensiert und die angesteuerten Zielpositionen präziser erreicht werden. Im Vergleich zu dieser Arbeit, in der alle am Laufen beteiligten Gelenke vorhergesagt werden, wird hier allerdings nur eine Koordinate mit Rotation verwendet und vorhergesagt.

Ommer, Stumpf und Stryk (2018) beschreiben einen anderen Ansatz, der adaptive Kompensations-Vorwärtsregler auf der Basis eines Kompensationsmodells benutzt, um die Verzögerung auszugleichen. Dies kann in Verbindung mit einem beliebigen existierenden Bewegungsmodell verwendet werden. Die Idee ist, nicht die Aktion zwischen zwei Zuständen zu lernen, sondern nur die erforderliche kompensierende Aktion. Dies reduziert die Modellkomplexität und ermöglicht ein Online-Lernen. Statt das gesamte Bewegungsmodell des Roboters trainieren zu müssen, kann der vorgestellte Kompensationsregler somit ohne jegliches Vortraining eingesetzt werden. Es werden verschiedene Ansätze ausprobiert, welche nach wenigen Iterationen sichtbare Verbesserungen erzielen. Am besten hat *Locally Weighted Projection Regression* (LWPR) (Vijayakumar, D'Souza und Schaal 2005; Vijayakumar und Schaal 2000) funktioniert, welches sich schnell an neue Gegebenheiten wie ausfallende Motoren oder Umgebungsänderungen anpassen kann und gute Ergebnisse liefert. Dieser Ansatz lässt

sich zusätzlich zu anderen Reglern verwenden und wurde unter anderem in der SSL getestet.

Außerhalb des RoboCups gibt es verschiedenste Anwendungsbereiche, in denen sich mit verwandten Themen beschäftigt wird. So haben sich zum Beispiel **Huang u. a. (2019)** mit der Gelenkwinkelvorhersage für motorisierte Gliedmaßen beschäftigt. Hierbei besteht ein aktuelles Problem darin, dass Steuerbefehle verzögert sind. Dabei sollen aus einer Kombination von *Oberflächen-Elektromyografie* (sEMG) Signalen und Trägheitsmessungen der Ober- und Unterschenkel in Echtzeit bei limitierter Rechenleistung Kniegelenkwinkel für unterschiedliche Gangarten vorhergesagt werden. sEMG Signale können Bewegungsabsichten von Personen im Voraus anzeigen und somit einen zeitlichen Vorteil von 30 ms bis 100 ms bringen. Dabei werden tiefe rekurrente neuronale Netze mit einer kleinen Parameterzahl pro Schicht verwendet. Es konnte das Kniegelenk in Echtzeit auf einem Mikrocontroller 50 ms in die Zukunft mit einem Vorhersagefehler von 2,93° vorhergesagt werden. Das Netz funktioniert dabei auch auf Daten von Personen, die nicht im Training verwendet wurden.

2.2 Klassische Ansätze

In diesem Abschnitt werden kurz einige klassische Ansätze für das vorliegende Problem vorgestellt. Alle Ansätze sind dabei, wie vieles was Zeitreihenvorhersagen betrifft, darauf ausgelegt, einzelne Daten vorherzusagen und nicht mehrere Daten auf einmal, wie es in dieser Arbeit der Fall ist. Einer ist der Smith Predictor (Smith 1959). Dieser benutzt ein festes Modell eines Systems ohne Verzögerung und verwendet die Messwerte des tatsächlichen Systems zur Korrektur. Dazu ist es allerdings nötig, ein sehr gutes Modell des Systems zu finden, was sich in der Realität meistens als schwierig herausstellt.

Eine weit verbreitete Methode für kurzfristige Vorhersagen sind zum Beispiel *Autoregressive Integrated Moving Average* (ARIMA) Modelle (Box u. a. 1976). Hier werden verschiedene Techniken kombiniert, um aus der Kombination Vorhersagen zu erstellen. Autoregressive beschreibt dabei die Regression gegen sich selbst. Das heißt, es werden frühere Werte der Zielvariable als Eingangs-

variable verwendet, um Werte für die Zukunft vorherzusagen. Das Integral wird verwendet, um eine Zeitreihe stationär zu machen, also dass sich ihre statistischen Eigenschaften im Laufe der Zeit nicht ändern und somit keine Trends mehr vorhanden sind. Der Moving Average betrachtet die Vorhersagefehler der Vergangenheit, um zukünftige Werte vorherzusagen. Kombiniert können diese Techniken bestimmte Zeitreihen gut beschreiben und erlauben damit kurzfristige Vorhersagen. Eine Erweiterung ist SARIMA (Brockwell und Davis 1991), wobei zusätzlich die Saisonalität mit betrachtet wird und somit auch saisonale Effekte explizit mit modelliert werden können.

Eine alternative Methode sind *Exponential Smoothing* (ETS) Modelle (Brown und Meyer 1961). ETS bezieht sich dabei auf Fehler, Trend und Saisonalität. Im Gegensatz zur ARIMA Familie, bei der die Vorhersage eine gewichtete lineare Summe der jüngsten Beobachtungen ist, wird zwar auch die gewichtete Summe vergangener Beobachtungen verwendet, allerdings werden diese mit exponentiell abnehmender Gewichtung betrachtet. Dabei gibt es drei grundlegende Typen. Single Exponential Smoothing für die Vorhersage von einfachen Daten ohne Trend und Saisonalität. Double Exponential Smoothing, welches zusätzlich Trends modellieren kann und Triple Exponential Smoothing, wo auch die Saisonalität explizit betrachtet wird. Diese ermöglichen ebenfalls eine kurzfristige Vorhersage.

3 Datengrundlage

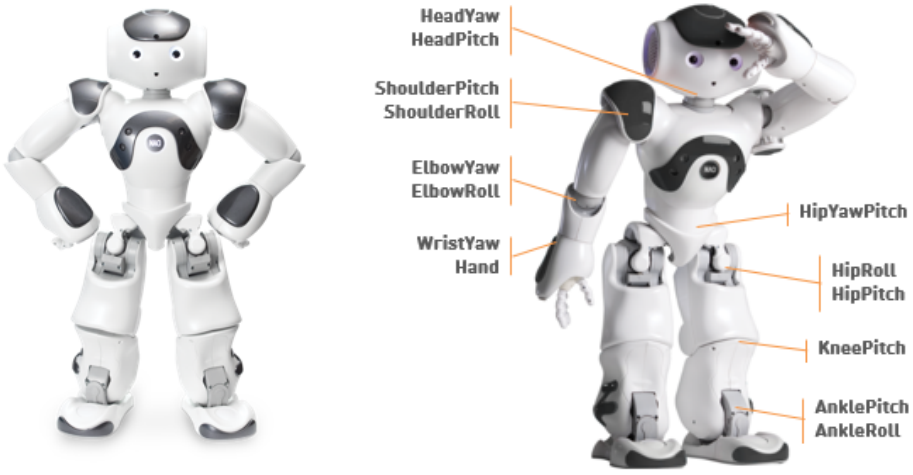
Daten sind die Basis, um neuronale Netze trainieren zu können. Im Folgenden wird sich zunächst in Abschnitt 3.1 die Roboterplattform grundlegend angeschaut, mit welcher die Daten aufgenommen wurden und für die entwickelt wird. Anschließend wird in Abschnitt 3.2 der verwendete Datensatz beschrieben und erläutert, wie die Ausgangsdaten verarbeitet wurden, um auf diesen trainieren zu können.

3.1 NAO

Der NAO ist ein humanoider Roboter des europäischen Roboterherstellers *Aldebaran* (2023). In Abschnitt 3.1.1 wird die aktuelle Generation beschrieben und anschließend in Abschnitt 3.1.2 genauer darauf eingegangen, woher die Sensordaten kommen. Abschließend wird in Abschnitt 3.1.3 das Laufen des Roboters bei B-Human genauer beschrieben.

3.1.1 Der NAO[6]

Das aktuelle Modell ist in Abbildung 3.1a gezeigt. Der Roboter ist 57,4 cm groß und 5,48 kg schwer. Er hat 25 steuerbare Freiheitsgerade, deren Motoren alle 12 ms angesteuert werden können. Die Freiheitsgerade sind in Abbildung 3.1b dargestellt und benannt. Die Positionssensoren der Motoren haben dabei eine Genauigkeit von etwa 0,1° (SoftBank Robotics 2023b). Eine Besonderheit ist die Hüfte, bei der zur Rotation der Beine nur ein Gelenk (HipYawPitch) verwendet wird und sich die Beine somit nicht unabhängig voneinander öffnen können. Weiterhin ist dieses um 45° im Raum gedreht, sodass dieses Gelenk gleichzeitig für eine Rotation in allen Achsen sorgt. Im Torso ist eine *Inertial Measurement*



(a) Der NAO[6]. (b) Die Gelenke des NAO[6].

Abbildung 3.1: Der NAO[6]. Bilder von SoftBank Robotics (2023a,b).

Unit (IMU) verbaut, welche sowohl ein Gyrometer als auch einen Beschleunigungssensor für Messungen im dreidimensionalen Raum besitzt. Außerdem hat der Roboter Drucksensoren unter den Füßen, sowie diverse weitere Sensoren – wie zum Beispiel Kameras – um seine Umgebung wahrzunehmen. Im Roboter ist ein Intel Atom E3845 Quad-Core-Prozessor mit einer Taktrate von 1,91 GHz verbaut, welchem 4 GB DDR3 RAM zur Verfügung stehen (SoftBank Robotics 2023b,c). Für eine Anwendung der neuronalen Netze auf dem Roboter müssen diese somit auf dieser Hardware laufen können.

3.1.2 Hardwareansteuerung

Die Kommunikation zwischen der Software von B-Human und den Motoren und Sensoren vom NAO[6] erfolgt über die *Low Level Abstraction* (LoLA) von Aldebaran. B-Human empfängt die Daten, verarbeitet diese, sendet neue Befehle und wartet anschließend, bis wieder neue Daten von LoLA empfangen werden können. Dies wird als ein Zeitschritt bezeichnet. LoLA ist ein Echtzeitprozess, der mit 83 Hz (ein Zeitschritt alle 12 ms) Daten mit dem *Hardware Abstraction Layer* (HAL) austauscht. HAL kommuniziert dann mit den elektronischen

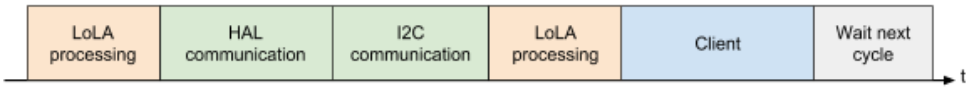


Abbildung 3.2: Der interne Ablauf der Kommunikation von LoLA. Bild von SoftBank Robotics (2023c).

Platinen des Roboters, welche die Motoren und sonstige steuerbare Elemente steuern und die Daten der Sensoren empfangen. Gouaillier u. a. (2009) beschreiben diesen Aufbau detailliert für die erste Version des NAO. Der Kreislauf für die aktuelle Version ist in Abbildung 3.2 dargestellt. Für den RoboCup gibt es dabei einen eigenen LoLA Client, mit dem über ein bereitgestelltes Protokoll kommuniziert wird (B-Human 2022; SoftBank Robotics 2023c). Innerhalb dieses Prozesses entsteht die in Abschnitt 1.2 beschriebene Verzögerung von bis zu drei Zeitschritten. Außerdem werden die Daten von der Herstellersoftware weiterverarbeitet. So wird unter anderem auch die Kraft der Motoren reduziert, wenn die Motoren heißer werden (SoftBank Robotics 2023b).

3.1.3 Humanoides Laufen bei B-Human

Das aktuelle Laufen wird seit 2017 verwendet und seitdem ständig weiterentwickelt. Es handelt sich bei dem Laufen grundsätzlich um eine inverse Pendelbewegung, in der die Füße neu platziert werden. Der aktuelle Stand ist größtenteils im *B-Human Team Report and Code Release 2021* (Röfer, Laue, Bahr u. a. 2021, S. 35–42) beschrieben. Kleinere Neuerungen aus 2022 sind in Röfer, Laue, Ewers u. a. (2022, S. 21–27) zu finden. Es handelt sich dabei nicht um eine vollständige Eigenentwicklung, sondern basiert auf dem Laufen vom Team *rUNSWift*, welches 2014 von Hengst (2014) vorgestellt wurde. Es erlaubt dem Roboter sowohl gleichzeitig vorwärts und seitwärts zu laufen, als auch sich zu drehen. Die Größe der Bewegungen ist dabei durch die physikalischen Gegebenheiten des Roboters begrenzt. Am Laufen sind alle Gelenke, die auf der rechten Seite in Abbildung 3.1b aufgelistet sind, beteiligt.

Für einen Schritt werden die Füße dabei als stets parallel zur gedachten Bodenebene betrachtet. Ein Schritt wird dann als eine Translation mit einer

Tabelle 3.1: Erreichbare und beispielhaft erreichte Gelenkwinkel in Grad eines Roboters beim Laufen. Bei Gelenken, die sich seitlich bewegen, beschreibt Min die Rotation nach innen und Max entsprechend nach außen. Erreichbar entnommen aus SoftBank Robotics (2023b).

	Erreichbar		Erreicht		
Gelenk	Min	Max	Min	Mean	Max
HipYawPitch	−65,62	42,44	−56,07	−2,57	14,59
HipRoll	−21,74	45,29	−4,83	2,99	24,26
HipPitch	−88,00	27,73	−47,99	−21,42	16,35
KneePitch	−5,29	121,04	31,47	50,24	72,95
AnklePitch	−68,15	52,86	−45,70	−28,52	−12,57
AnkleRoll	−22,79	44,06	−7,91	2,87	19,43

Rotation in der gedachten Bodenebene beschrieben. Die Translation nach vorne, sowie die Rotation, werden jeweils zu 50 % auf beide Füße angewendet. Die Translation zur Seite wird dagegen jeweils vollständig angewendet. Dies soll dafür sorgen, dass der Schwerpunkt des Roboters stets genau mittig zwischen beiden Füßen ist. Der Schritt wird dann ausgeführt, indem über die Länge eines Schrittes (aktuell 250 ms) von der Ausgangs- zur Zielposition interpoliert wird. Der Fuß, der den Boden berührt (Standfuß), wird dabei in der Höhe nicht verändert, während bei dem Fuß, der in der Luft ist (Schwingfuß), in der ersten Hälfte die Höhe reduziert und anschließend wieder zurück interpoliert wird (Röfer, Laue, Bahr u. a. 2021, S. 35–42). Die Winkel der einzelnen Gelenke werden über eine inverse Kinematik berechnet. Dabei werden, wie beispielhaft für einen Roboter aus einem Spiel in Abbildung 3.3 dargestellt, für unterschiedliche Gelenke unterschiedliche Geschwindigkeiten erreicht. In Tabelle 3.1 ist zusätzlich dargestellt, welche Gelenkwinkel der NAO[6] erreichen kann und beispielhaft, welche beim Laufen tatsächlich erreicht werden.

Um das Laufen zu balancieren, werden zwei Arten von Sensoren einbezogen, welche die Bewegung modifizieren. Zum einen die Fußdrucksensoren, welche einen Schrittwechsel und somit einen neuen Schritt auslösen, sobald das Gewicht auf dem Schwingfuß größer wird als das auf dem Standfuß. Zum anderen wird die Gyrometermessung der Neigung nach vorne und hinten über einen Tiefpass-

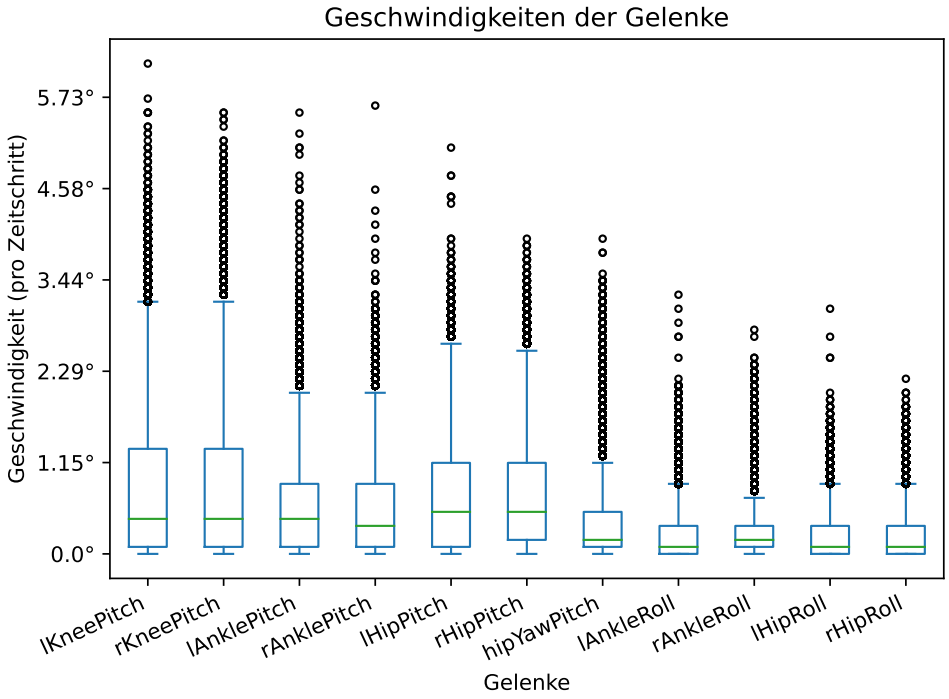


Abbildung 3.3: Beispiel der Verteilung der Geschwindigkeiten, von einem Roboter aus einem Spiel in 2019, die ein Gelenk pro Zeitschritt erreicht.

Filter gefiltert. Dies wird verwendet, um den Standfuß vorne in den Boden zu drücken, wenn der Roboter weiter nach vorne geneigt ist und umgekehrt. Somit wird verhindert, dass der Roboter sich beim Laufen aufschaukelt und umfällt (Reichenberg 2021, S. 21–22).

Die Implementierung, die 2019 verwendet wurde, ist Open Source (B-Human Team 2019) veröffentlicht und in Röfer, Laue, Baude u. a. (2019) beschrieben. Für 2022 wurde eine unter anderem von Reichenberg und Röfer (2022) erweiterte Version verwendet. Diese ist ebenfalls Open Source (B-Human Team 2022) veröffentlicht und in Röfer, Laue, Bahr u. a. (2021, S. 35–42) und Röfer, Laue, Ewers u. a. (2022, S. 21–27) beschrieben.

3.2 Datensatz

Wie in Abschnitt 1.1.2 bereits erwähnt, stehen aus den Jahren 2019 und 2022 jeweils Daten von zwei Wettbewerben zur Verfügung. Die Daten von beiden Wettbewerben eines Jahres bilden dabei zusammen jeweils eine Einheit für das Jahr. Diese werden im Folgenden beschrieben, miteinander verglichen und diskutiert.

Insgesamt wurden mit elf Robotern auf elf Feldern 47 verwertbare Spiele ausgetragen. In Tabelle 3.2 ist dargestellt, wie sich die Felder, Roboter und Spiele auf die Jahre verteilen. Dabei wurde ein Roboter nur in 2019 und zwei nur in 2022 verwendet. Die Roboter mit den Namen *Krapotke* und *Sarah* sind ein Jahr älter als die anderen 2019 verwendeten Roboter, da sie zuerst gekauft wurden, um den NAO[6] zu testen und die Software für die Robotergeneration anzupassen. Sie haben aber gleich viele Wettbewerbe gespielt wie die anderen. Die Roboter *Gott* und *FriedrichWilhelm* sind dagegen erst nach den Wettbewerben 2019 gekauft worden. Der genaue Zustand der Roboter ist aber insbesondere in 2022 nicht bestimmbar, da viele bereits an unterschiedlichen Stellen repariert wurden und teilweise nicht bekannt ist, was genau gemacht wurde. Eine genaue Übersicht, wie viel welcher Roboter auf welchem Feld gelaufen ist, ist am Ende dieses Abschnitts in Abschnitt 3.2.3 zu finden.

Die Roboter haben in den Spielen Logs von eingehenden Sensordaten und größeren Teilen ihres internen Zustandes aufgezeichnet. Einige Logs, aber

Tabelle 3.2: Eckdaten der Jahre, wobei die Roboter größtenteils die gleichen sind.

Jahr	Felder	Roboter	Spiele
2019	6	9	25
2022	5	10	22
Gesamt	11	11	47

auch ganze Spiele wurden aus verschiedenen Gründen im Laufe der Arbeit aussortiert, da sie nicht verwendet werden konnten. Einige sind vermutlich beim Kopieren oder Schreiben beschädigt worden und waren deshalb nicht lesbar. Andere konnten nicht eindeutig Feldern oder Spielen zugeordnet werden, sind von Tests, in denen kein normales Fußballspiel stattfand oder waren aus anderen Gründen ungeeignet. Es wird im Weiteren immer von den insgesamt am Ende verwendeten 682 Logs ausgegangen.

Auf den 682 Logs wurden in 23 847 263 Zeitschritten (4788,61 min) insgesamt 800,71 GiB an Daten aufgezeichnet. Ein Großteil davon sind Bilder und andere Daten, die für diese Arbeit nicht benötigt werden. Aus diesem Grund und um mit den Daten besser umgehen zu können, wurden folgende Daten in eine *JavaScript Object Notation* (JSON) extrahiert:

`FallDownState`: Informationen, ob der Roboter hingefallen ist.
`FrameInfo`: Der Zeitpunkt zu der der Zeitschritt aufgenommen wurde.
`JointRequest`: Die angefragten Gelenkwinkel für die Beine.
`JointSensorData`: Die gemessenen Gelenkwinkel für die Beine.
`MotionInfo`: Die gerade ausgeführte Bewegung.

Dabei gab es in den 2019-er Daten 55 defekte Zeitschritte, welche nicht gelesen werden konnten und übersprungen wurden. Die Zwischenrepräsentation ist mit 26,15 GiB deutlich kleiner und vereinfacht die Handhabung für die weitere Verarbeitung, zumal sie gut menschen- und maschinenlesbar ist. Eine JSON-Datei enthält dabei die Daten aller Logs eines Roboters aus einem Spiel. Die Implementierung dazu ist in Abschnitt 4.3.1 zu finden.

In einem zweiten Schritt wurden diese Daten dann zu den eigentlichen Datensätzen, die als Eingabe zum Lernen dienen, verarbeitet. So wurden aus den 2019-er Daten Zeitschritte entfernt, die aufgrund eines Fehlers in der Software kurzzeitig falsche Gelenkwinkelanfragen enthielten. Außerdem werden nur die Daten von laufenden Robotern verwendet. Bei stehenden Robotern sollten sich die Gelenke nicht großartig verändern und eine Vorhersage bringt somit keinen Mehrwert. In speziellen Situationen wie dem Aufstehen oder während Schüssen wäre eine Vorhersage möglicherweise auch hilfreich. In dieser Arbeit wird sich jedoch auf das Laufen konzentriert. Für diese Situationen würden sich voraussichtlich auch eher eigene Vorhersagen anbieten, da es sich um zeitlich begrenzte Ereignisse handelt, dessen Eintreten unmittelbar bekannt ist. Die resultierenden Abschnitte, in denen gelaufen wird, sind in Abschnitt 3.2.1 genauer beschrieben.

Weiterhin wurden diverse weitere Anpassungen für das Training vorgenommen. Es wurden die angefragten Gelenkwinkel um einen Zeitschritt verschoben. Somit wird zu einem Zeitpunkt davon ausgegangen, dass die aktuellen Sensorwerte und die letzte Anfrage zur Verfügung stehen für eine neue Anfrage. Da im Training die vergangene Zeit zwischen zwei Zeitschritten als konstant betrachtet wird, muss sichergestellt sein, dass dies auch zutrifft. Dies ist, wie in Abschnitt 3.2.2 beschrieben, jedoch nicht immer der Fall und musste entsprechend aufgelöst werden, indem die Abschnitte an diesen Stellen geteilt wurden. Auch wurden zu kleine Abschnitte entfernt, die nicht zum Trainieren benutzt werden können. Die resultierenden Datensätze enthalten 12 112 153 Zeitschritte (2432,16 min) an jeweils elf angefragten und gemessenen Gelenkwinkeln, was 5,11 GiB an *CSV*-Dateien entspricht. Die Datenmenge ist somit 99,36 % kleiner als die Ausgangsdaten. Eine CSV-Datei wird dabei aus einer JSON-Datei erzeugt. Diese können direkt zum Training verwendet werden.

Die Implementierung ist in Abschnitt 4.3.2 zu finden und eine grobe Übersicht über die Dimensionen der Datensätze ist in Tabelle 3.3 dargestellt. Eine Beobachtung ist hier, dass Roboter 2019 53,67 % der Zeit laufen und 2022 nur 47,86 %. Die Ursache lässt sich aus den im Rahmen der Arbeit ausgewerteten Daten nicht erschließen. Diese und diverse weitere Unterschiede und Auffälligkeiten, die innerhalb und zwischen den Daten aus 2019 und 2022

Tabelle 3.3: Übersicht über die Datensätze.

Jahr	Log-Dateien	Größe (in GiB)			Spielzeit (in min)		CSV-Dateien
		Log	JSON	CSV	Gesamt	Laufen	
2019	346	344,57	13,04	2,72	2413,30	1295,24	145
2022	336	456,14	13,11	2,39	2375,31	1136,92	121
Summe	682	800,71	26,15	5,11	4788,61	2432,16	266

festgestellt wurden, könnten weiter untersucht werden, welche an dieser Stelle aber den Themenbereich der Arbeit verlassen. Dazu könnten zum Beispiel zu den Abschnitten Spielernummern und Spielzustände assoziiert werden.

3.2.1 Anzahl der Laufabschnitte

Da aus den Daten nur Abschnitte verwendet werden, in denen gelaufen wird, wurden diese genauer untersucht. Das Laufen wird dabei regelmäßig durch Stehen oder andere Bewegungen unterbrochen. In Abbildung 3.4 ist für 2019 dargestellt, wie sich Zeiträume, in denen durchgehend gelaufen wird, auf die Spiele der Roboter verteilt haben. Insgesamt gibt es 6041 Abschnitte von durchgehendem Laufen.

Links ist zu sehen, dass die mittleren 50 % der Spiele der Roboter 25 bis 52 Abschnitte enthalten. Es gibt aber auch Spiele, in denen ein Roboter 100 oder mehr Abschnitte pro Spiel hat. Dies spricht möglicherweise für viele kurze Läufe. Der Median mit 32 zeigt, dass es im Normalfall nicht so viele Abschnitte gibt.

In der Mitte ist die Länge der einzelnen Abschnitte dargestellt. Dabei ist alles über 1 min abgeschnitten (da die Abbildung sonst sehr gestaucht und somit wenig zu erkennen ist), was 181 Abschnitten entspricht. Von diesen sind 15 Abschnitte auch länger als 2 min. Die kürzesten Abschnitte sind 0,16 s. Im Schnitt läuft ein Roboter 2,7 s bis 16,6 s am Stück, wobei ein Ausreißer mit 318 s auffällt.

Auf der rechten Seite ist die Zeit dargestellt, die einzelne Roboter pro Spiel gelaufen sind. Hier ist ersichtlich, dass Roboter sehr unterschiedlich viel laufen.

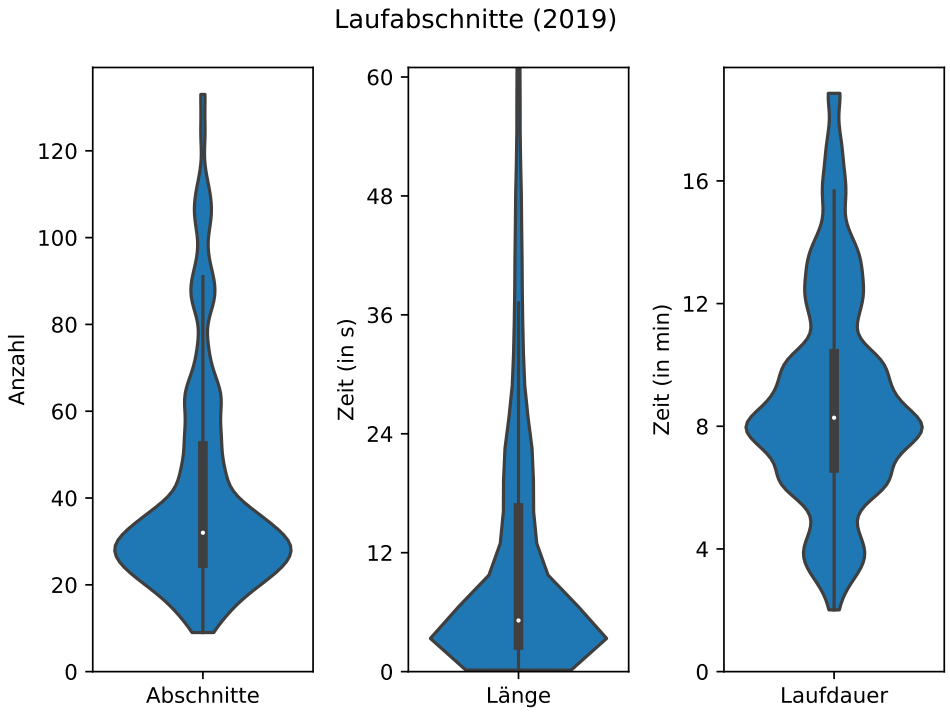


Abbildung 3.4: Beschreibung der Abschnitte der Spiele der Roboter. Abschnitte sind dabei Zeiträume in denen durchgehend gelaufen wird.
Links: Anzahl der Abschnitte pro Spiel der Roboter.
Mitte: Länge der einzelnen Abschnitte.
Rechts: Zeit die ein Roboter pro Spiel läuft.

Wenig läuft dabei vermutlich zum Beispiel der Torwart. Im Schnitt laufen Roboter 6,7 min bis 10,4 min pro Spiel, mit einem Durchschnitt von 8,93 min. Zu beachten ist, dass es sich hierbei nicht um die durchschnittlich gelaufene Zeit pro Spiel handelt, da nicht jeder Roboter ganze Spiele gespielt hat, sondern Roboter während eines Spiels teilweise auch ausgetauscht oder herausgenommen wurden.

In Abbildung 3.5 ist für 2022 dargestellt, wie sich Zeiträume auf die Spiele der Roboter verteilen, in denen durchgehend gelaufen wurde. Dabei handelt es sich mit 6032 Abschnitten um ähnlich viele wie 2019, wobei es drei Spiele weniger gab. Im Gegensatz zu 2019 gibt es hier im Schnitt somit mehr Abschnitte.

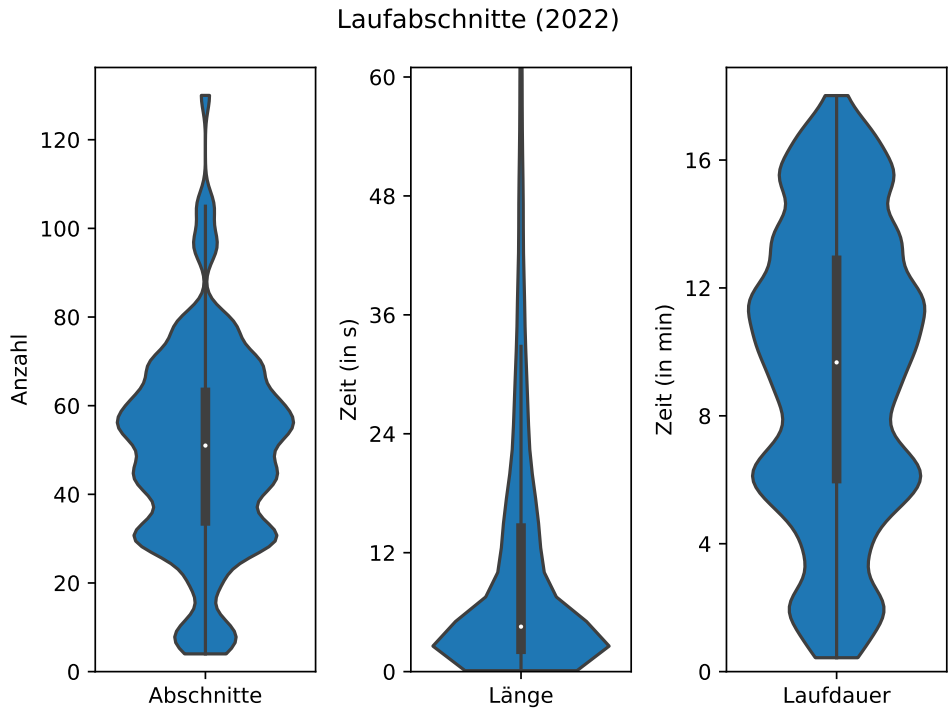


Abbildung 3.5: Beschreibung der Abschnitte der Spiele der Roboter. Abschnitte sind dabei Zeiträume in denen durchgehend gelaufen wird.
Links: Anzahl der Abschnitte pro Spiel der Roboter.
Mitte: Länge der einzelnen Abschnitte.
Rechts: Zeit die ein Roboter pro Spiel läuft.

So sind in der mittleren Hälfte der Spiele der Roboter 34 bis 63 Abschnitte enthalten und der Median ist mit 51 deutlich höher. Dafür gibt es mit 130 nur einen Ausreißer deutlich über 100.

Die Länge der einzelnen Abschnitte hat dagegen mit 2,3 s bis 14,5 s abgenommen. Mit 119 Abschnitten über 1 min, von denen 7 Abschnitte auch länger als 2 min sind, sind hier weniger Abschnitte abgeschnitten. In 2022 gibt es also mehr Läufe, diese sind dafür aber kürzer. Dies hängt möglicherweise mit Veränderungen in der Spieltaktik von B-Human zusammen, in der die Roboter durch häufigeres Passen mehr miteinander spielen als noch 2019.

Auf die gesamte Laufdauer über ein Spiel gesehen lässt sich feststellen, dass

die Spannweite, wie viel gelaufen wurde, 2019 relativ klein war im Vergleich zu 2022 mit 6,1 min bis 12,9 min. Ein Unterschied, der sich implizit auf die Laufdauer auswirkt, ist, dass 2019 durchschnittlich 5,8 Roboter an einem Spiel teilgenommen haben und diese auch untereinander mehr ausgewechselt wurden, in dem zum Beispiel der Torwart in der Halbzeit oft mit einem Feldspieler getauscht hat. 2022 wurden dagegen nur 5,3 Roboter pro Spiel verwendet. Zusätzlich müssen Spiele in der SPL auch nicht immer genau 20 min dauern. Dies hat somit einen zusätzlichen Einfluss, welcher im Rahmen der Arbeit aber nicht näher betrachtet wurde.

3.2.2 Dauer eines Zeitschritts

Die Zeit zwischen zwei Zeitschritten sollte immer 12 ms betragen (SoftBank Robotics 2023c). Bei der Analyse ist aufgefallen, dass dies nicht immer der Fall ist. Aus diesem Grund wurde eine Toleranz von einer Millisekunde festgelegt, da die Zeitstempel in Millisekunden ohne Nachkommastelle angegeben sind und kleine Schwankungen durch Scheduling im Prozessor oder der Übertragung im Roboter auftreten können und somit akzeptabel sind. Dies entspricht einer maximalen Abweichung von ungefähr 10 %. Der Bereich von 11 ms bis 13 ms wird im Weiteren als normal und somit einfach als 12 ms Zeitschritte verstanden. In 2019 haben 99,984 % der Zeitschritte eine normale Dauer. In Abbildung 3.6 sind die Häufigkeiten einer unerwarteten Zeitschrittdauer von 2019 dargestellt. Dabei fällt auf, dass es eine Häufung bei 14 ms und 15 ms gibt. Hier wird eine Verzögerung im System vermutet. Eine weitere Häufung gibt es bei 21 ms bis 23 ms. Hier fehlt ein Zeitschritt. Einen kleinen Teil von 55 Zeitschritten machen dabei bereits erwähnte defekte Zeitschritte im Logs aus, welche bei der Verarbeitung übersprungenen wurden. Die Häufigkeit lässt vermuten, dass der Roboter ab und zu einen Zeitschritt überspringt. Die Häufungen machen zusammen 95 % der Abweichungen aus. Eine weitere Auffälligkeit ist, dass es einzelne Ausreißer gibt, die kleiner als 12 ms sind. Diese treten meistens in Verbindung mit langen Zeitschritten auf. Hier ist die Vermutung, dass es zu einer Verzögerung im System kommt und dann zwei Zeitschritte kürzer hintereinander gesendet werden.

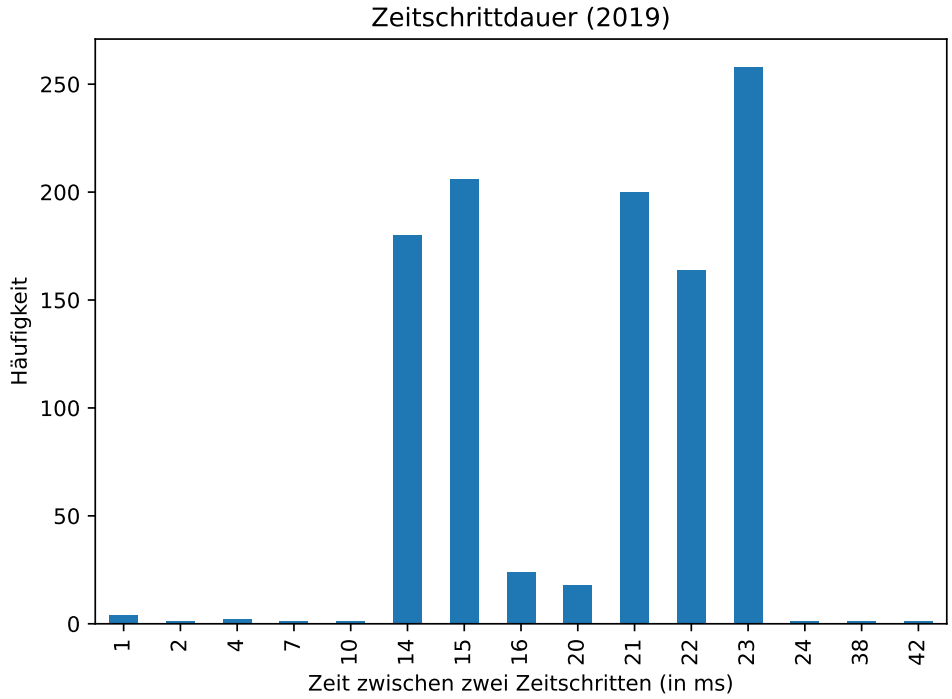


Abbildung 3.6: Darstellung der Häufigkeiten, in der die Zeit zwischen zwei Zeitschritten nicht im erwarteten Bereich von 11 ms bis 13 ms ist.

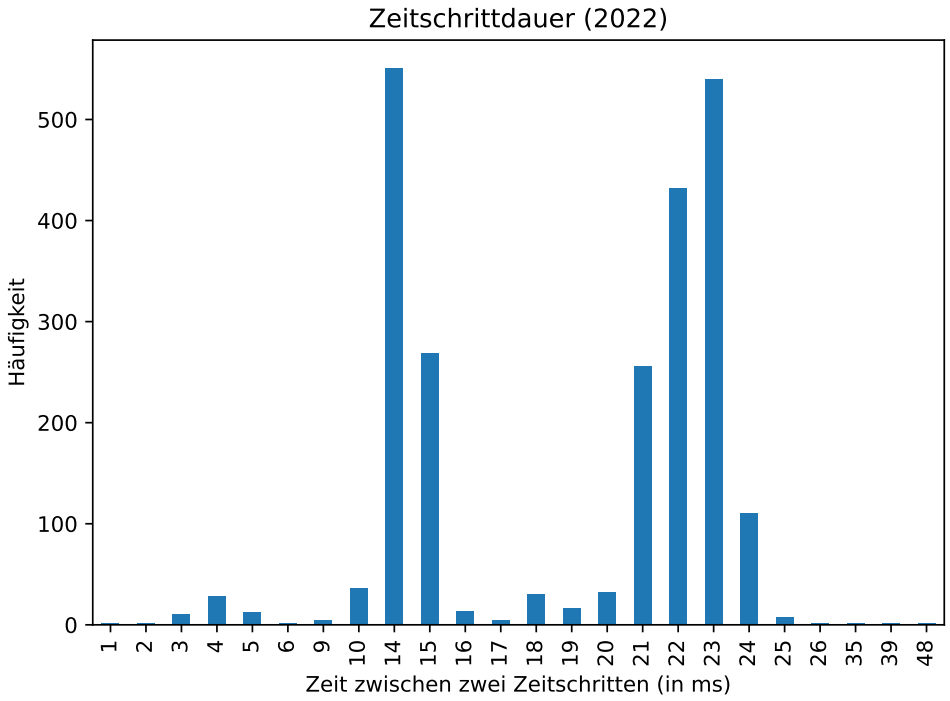


Abbildung 3.7: Darstellung der Häufigkeiten, in der die Zeit zwischen zwei Zeitschritten nicht im erwarteten Bereich von 11 ms bis 13 ms ist.

In 2022 haben 99,958 % der Zeitschritte eine normale Dauer. Somit gibt es prozentual mehr als doppelt so viele Ausreißer wie 2019. Diese sind in Abbildung 3.7 dargestellt. Es gibt auch hier eine Häufung bei 14 ms und 15 ms. Die zweite Häufung geht dieses Mal bis 24 ms. Im Gegensatz zu 2019 sind allerdings keine Aussetzer in den Logs enthalten. Die Häufungen machen zusammen 91,56 % der Abweichungen aus. Somit gibt es deutlich mehr Abweichungen. Diese beschränken sich gleichzeitig aber auch weniger auf die Häufung. Die Ursache dafür ist zum aktuellen Zeitpunkt unbekannt.

3.2.3 Die Laufzeiten auf den Feldern

In Tabelle 3.4 ist für 2019 dargestellt, wie viel welcher Roboter auf welchem Feld gelaufen ist. Dies ermöglicht sinnvoll auszuwählen, welche Daten für welche Experimente genutzt werden sollen. In Tabelle 3.5 ist dies für 2022 dargestellt. 2019 sticht das Feld GO mit 526,67 min heraus. Dies liegt daran, dass der ganze Wettbewerb auf diesem Feld ausgetragen wurde, während es auf den anderen Wettbewerben mehrere Felder gab. Dabei gehören Felder mit gleichem Präfix zum gleichen Event.

Tabelle 3.4: Spielzeit (in min) die B-Human-Roboter 2019 auf Wettbewerbsfeldern liefen. Dies umfasst die German Open (GO) und den RoboCup (RC). Das Maximum ist dabei jeweils hervorgehoben.

	Feld						**Summe**
Roboter	GO	RC A	RC B	RC C	RC D	RC E	
ElseKling	0,00	18,41	14,52	0,00	21,97	**46,59**	101,49
OttoVon	41,86	8,56	0,00	0,00	**25,18**	32,09	107,69
DasKaenguru	58,89	9,17	10,35	0,00	0,00	37,36	115,77
JuliaMueller	46,78	12,54	22,89	0,00	23,96	21,80	127,97
DerPinguin	87,23	18,80	21,45	0,00	11,43	9,36	148,27
Krapotke	56,34	20,40	18,09	**16,19**	16,63	31,80	159,44
Sarah	67,82	23,15	10,17	15,34	14,21	41,99	172,68
Herta	68,93	**27,78**	23,43	0,00	22,77	33,81	176,71
MarcUwe	**98,84**	24,43	**23,80**	7,41	17,08	13,74	**185,29**
Summe	**526,68**	163,25	144,69	38,94	153,22	268,55	

Tabelle 3.5: Spielzeit (in min) die B-Human-Roboter 2022 auf Wettbewerbsfeldern liefen. Dies umfasst das German Open Replacement Event (GORE) und den RoboCup (RC). Das Maximum ist dabei jeweils hervorgehoben.

	Feld					**Summe**
Roboter	GORE A	GORE B	RC A	RC B	RC C	
JuliaMueller	0,00	0,00	9,86	0,00	44,63	54,49
DasKaenguru	6,52	13,94	22,43	**20,24**	16,68	79,80
MarcUwe	0,00	0,00	37,46	6,92	36,63	81,01
Herta	5,98	8,25	23,00	8,84	**47,20**	93,27
OttoVon	16,53	17,28	36,37	17,67	21,69	109,53
Krapotke	37,38	37,83	28,97	14,84	6,88	125,90
Sarah	36,51	42,69	30,57	6,64	12,63	129,03
Gott	43,01	**48,15**	**43,83**	15,55	0,00	150,54
DerPinguin	43,79	36,31	24,93	15,78	34,61	155,42
FriedrichWilhelm	**47,33**	41,09	33,30	19,44	16,96	**158,12**
Summe	237,04	245,54	**290,69**	125,92	237,90	

4 Implementierung

Um mit den beschriebenen Daten arbeiten zu können, wird in diesem Kapitel die geschriebene Software beschrieben, welche die Daten verarbeitet und es ermöglicht Vorhersagen auf Basis dieser zu machen. Dazu werden zunächst in Abschnitt 4.1 auf die Vorarbeiten und in Abschnitt 4.2 auf die verwendete Software eingegangen. In Abschnitt 4.3 wird dann die Implementierung der Erstellung der Datensätze beschrieben und in Abschnitt 4.4 die Aufbereitung für das Training. Anschließend wird in Abschnitt 4.5 das Training eines Modells beschrieben. Abschließend werden in Abschnitt 4.6 Möglichkeiten für eine Reihe von Trainings erläutert und in Abschnitt 4.7 die Evaluation. Für Teile der Implementierung wurden Tests geschrieben, um sicherzustellen, dass die Implementierung korrekt ist und Anpassungen die Ergebnisse nicht verändern. Hierbei geht es insbesondere um das Verarbeiten und Aufbereiten der Daten. Die Implementierung für den Roboter ist gesondert in Abschnitt 6.2 zu finden.

4.1 Vorarbeiten

Die Arbeit baut auf verschiedenen Softwarekomponenten auf, die für diese Arbeit vom Autor geschrieben oder erweitert wurden. Ein von *Arne Hasselbring* geschriebener Python Wrapper, um B-Human Logs in Python verarbeiten zu können, wurde leicht erweitert und für das Team B-Human nutzbar gemacht. Die Ablage von Logs wurde ab 2019 vereinheitlicht und ist somit leichter zu verarbeiten. Darauf aufbauend wurde eine Python Bibliothek geschrieben, um Logs auf dem Datenserver zu durchsuchen und herunterzuladen. Weiterhin können Informationen zu lokalen Logs bereitgestellt werden. Außerdem wurde eine Projektstruktur inklusive Generator für maschinelles Lernen Projekte aufgesetzt, die auf die Bedürfnisse von B-Human angepasst ist. Sie soll

Projekte vereinheitlichen und somit den Einstieg für die regelmäßigen neuen Projektmitglieder erleichtern, versucht dabei aber möglichst wenig Vorgaben zu machen. Sie wird in dieser Arbeit und mittlerweile in diversen anderen Projekten genutzt.

4.2 Verwendete Software

Für die Entwicklung wird auf diverse Bibliotheken zurückgegriffen. Die wichtigsten werden im Folgenden kurz vorgestellt:

Pandas ist eine schnelle, leistungsstarke und flexible Open-Source Bibliothek zur Datenanalyse und -manipulation (McKinney 2010). Den Kern bildet der sogenannte `DataFrame`, welcher das effiziente Verarbeiten von tabellenartigen Daten erlaubt, wie sie in dieser Arbeit vorliegen. Sie wurde verwendet, um die Daten vorzubereiten und zu analysieren. Es wurde die Version 1.5 verwendet (The Pandas Development Team 2023).

TensorFlow 2 ist eine Open-Source end-to-end Plattform für maschinelles Lernen (Abadi u. a. 2015). Sie wurde zum Trainieren der Modelle benutzt, wobei hauptsächlich auf die High-Level-API *Keras* zum Definieren von Modellen und Trainingsprozessen zurückgegriffen wurde. Es wurde die Version 2.10 verwendet (TensorFlow Developers 2022).

Matplotlib ist eine umfassende Open-Source Bibliothek zur Erstellung statischer, animierter und interaktiver Visualisierungen (Hunter 2007). Sie wurde für viele der Abbildungen in dieser Arbeit verwendet. Einige Abbildungen wurden auch mit der High-Level-Schnittstelle *seaborn* angefertigt, welche sich zum Visualisieren von Datensätzen und für statistische Grafiken eignet. Es wurde die Version 3.6 verwendet (Matplotlib Developers 2023)

4.3 Datensatzerstellung

Um Modelle trainieren zu können, müssen die Daten, wie bereits in Abschnitt 3.2 beschrieben, zunächst aus den Logs zu einem zum Trainieren sinnvollem Format

verarbeitet werden. Dazu wurden zunächst automatisch alle verfügbaren Logs gesucht und zu Datensätzen zusammengefasst. Ein Datensatz ist dabei jeweils ein Event. Diese wurden im Anschluss manuell nach verarbeitet, wobei das Spielfeld hinzugefügt und einzelne Logs, die nicht benutzt werden sollen, entfernt wurden. Die Datensätze bilden die Grundlage für die automatische Verarbeitung, die im Folgenden beschrieben wird. Diese ist in zwei Teile aufgeteilt. In einem ersten Schritt werden in Abschnitt 4.3.1 die Logs nacheinander verarbeitet und alle definitiv nicht genutzten Daten weggeworfen. Dies hat zum einen den Grund, dass die Datenmenge mit ungefähr 800 GiB zu groß ist, um sie auf einem Gerät problemlos vorhalten zu können, zum anderen dauert das Verarbeiten einige Stunden. Die daraus entstandene JSON-Zwischenrepräsentation wird im zweiten Schritt in Abschnitt 4.3.2 verarbeitet und in eine CSV-Repräsentation gebracht, mit der trainiert werden kann.

4.3.1 Datengewinnung

Vor der Verarbeitung eines Datensatzes wird geprüft, ob Annahmen stimmen, die während der Verarbeitung gemacht werden. Um den Datensatz auch mit weniger Speicher verarbeiten zu können, werden immer nur die Logs von einem Spiel heruntergeladen, diese Roboter für Roboter verarbeitet und anschließend wieder gelöscht. Ein Roboter kann dabei mehrere Logs haben. Beim Verarbeiten wird für jedes Log zunächst geprüft, ob alle Daten vorkommen, die gesammelt werden sollen. Dann wird das Log verarbeitet und dabei alle Daten mitgeschrieben. Dazu können in einer Konfigurationsdatei Objekte angegeben und Attribute spezifiziert werden, die von diesen gesammelt werden sollen. Diese werden dann rekursiv mitgeschrieben, wobei bestimmte Attribute ignoriert werden können. Ist ein Zeitschritt defekt, wird er einfach übersprungen. Für die Logs werden zusätzlich verschiedene Metadaten mitgeschrieben, für eine Analyse der Datensätze und um rekonstruieren zu können, welche Daten woher kommen. Die Daten vom Spiel eines Roboters werden abschließend jeweils in JSON-Dateien gespeichert.

4.3.2 Datenverarbeitung

Die Datenverarbeitung nimmt einen Datensatz und verarbeitet jeweils die JSON Zwischenrepräsentation eines Spiels eines Roboters zur Zeit. Zuerst werden die Daten dazu in ein `DataFrame` überführt, auf getroffene Annahmen überprüft und vorverarbeitet, um sie effizient verarbeiten zu können. Ein wichtiges Element ist dabei der `Index`, welcher die Zeilen indiziert. Er wird dazu genutzt, kontinuierliche Abschnitte in den Daten zu markieren. Ist die Differenz im Index zwischen zwei Zeilen größer als eins, beginnt ein neuer Abschnitt. Dies wird im ersten Schritt genutzt, um mehrere Logs voneinander zu trennen, indem der Index an den Loggrenzen jeweils um eins verschoben wird. Anschließend werden Daten aussortiert, in denen angefragte Gelenkwinkel nicht valide sind. Dies war 2019 aufgrund eines Fehlers in der B-Human-Software der Fall. Nun werden die angefragten Gelenkwinkel um einen Zeitschritt in die Vergangenheit geschoben, damit zu einem aktuellen Sensorwert die letzte Anfrage zugeordnet ist. Eine Anfrage für diesen Zeitschritt zu berechnen, stellt ein Ziel dar und steht somit nicht zur Verfügung. Im nächsten Schritt werden alle Daten entfernt, in denen nicht gelaufen wird. Anschließend werden Abschnitte, in denen Zeitschritte vorkommen, die nicht 12 ms dauern, an diesen Stellen in weitere Abschnitte unterteilt. So ist sichergestellt, dass in Abschnitten immer eine konstante Zeit zwischen zwei Zeitschritten vergeht. Abschließend werden Abschnitte entfernt, die zu kurz sind, um auf diesen zu trainieren. Zusätzlich werden hier die Metadaten über den Datensatz verarbeitet und aufbereitet.

4.4 Vorverarbeitung

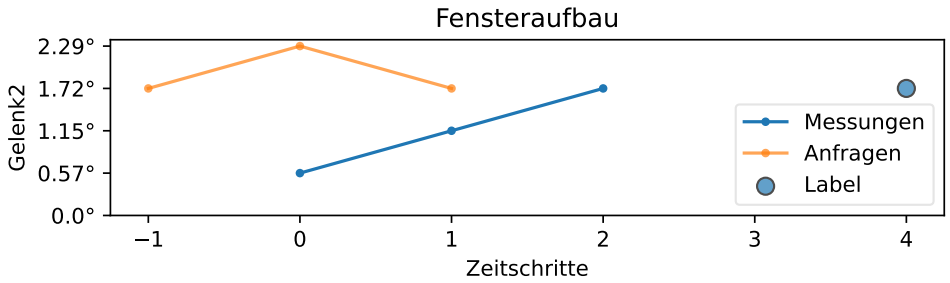
Um auf den CSV-Daten wirklich trainieren zu können, müssen diese noch aufbereitet werden, damit aus einzelnen Zeitpunkten Trainingsdaten werden. Dazu wird aus einer Liste an `DataFrame`s ein TensorFlow-`Dataset` erstellt, welches dynamisch ein sogenanntes Fenster aus Ein- und Ausgaben erstellt, das über die Daten geschoben wird und dabei Abschnittsgrenzen in den Daten beachtet. Ein solches `Dataset` erlaubt es, komplexe Vorverarbeitungen zu erstellen und die Daten dabei dynamisch zu laden und zu verarbeiten.

Dazu wird zunächst das Fenster beschrieben. Ein Fenster besteht aus einer Menge an Eingabe-Zeitschritten, einem Offset zwischen dem Ende der Eingabe und dem Anfang der Labels und der Menge an Label-Zeitschritten. In Abbildung 4.1a ist ein solches Fenster dargestellt. Dieses Fenster hat beispielhaft drei Eingabe-Zeitschritte, einen Offset von zwei und ein Label. Die Gesamtgröße des Fensters beträgt somit fünf. Es zeigt die Eingaben und das vorherzusagende Label. Die Eingabe sind entsprechend die letzten drei Anfragen und Messungen und das Label, also die Messung in zwei Zeitschritten. Die Anfragen sind einen Zeitschritt in die Vergangenheit dargestellt, da dies, wie bereits beschrieben, dem tatsächlichen Zeitpunkt der Daten entspricht. Dies beschreibt den implementierten Aufbau des Fensters. Da der aktuelle Zeitpunkt allerdings Zeitpunkt zwei ist, wird die Zeitachse im Weiteren in der Abbildung verschoben, um wie in Abbildung 4.1b dargestellt, die Zeitpunkte korrekt darzustellen.

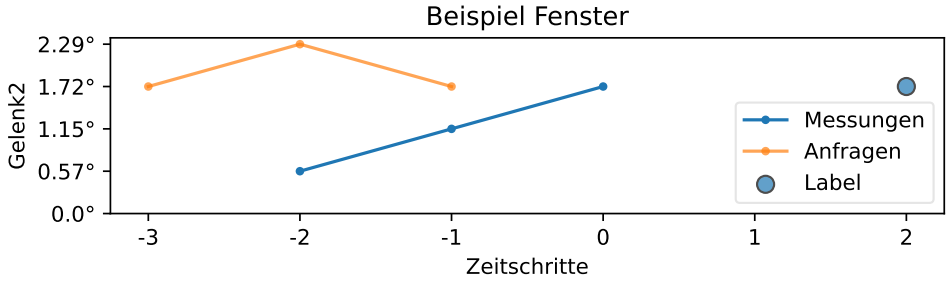
Das Erstellen des `Dataset`s aus Fenstern ist über einen Fenstergenerator realisiert, welcher auf einem Tutorial basiert (TensorFlow Developers 2023). Außerdem werden über diesen die Eingaben und Ausgaben eines Modells definiert und verschiedene Abbildungen bereitgestellt, welche auch die Vorhersagen von Modellen darstellen können. Dieser nimmt eine `DataFrame`-Liste und entfernt alle Daten, die nicht Ein- oder Ausgabe des Modells sind und teilt diese, sodass jedes `DataFrame` nur kontinuierliche Daten enthält. Diese werden dann einzeln zu `Dataset`s verarbeitet, welche jeweils ein gleitendes Fenster mit kontinuierlichen Daten bereitstellen. Anschließend werden sie zu einem `Dataset` kombiniert und durchmischt. Abschließend werden aus den kontinuierlichen Daten jedes Fensters die Eingaben und Labels erstellt. Diese können dann direkt zum Trainieren und Evaluieren von Modellen verwendet werden.

4.5 Training

Um Modelle für die verschiedenen Experimente trainieren zu können, wurde eine modulare Struktur entwickelt. Der Grundgedanke ist, verschiedene Experimente jeweils als eigene Teilprojekte zu realisieren, welche aber möglichst viel Software teilen. Jedes Projekt besteht dazu aus Funktionen um Trainings durchzuführen,



(a) Beispiel Fenster, welches den tatsächlichen Aufbau zeigt.



(b) Beispielfenster mit verständlicheren Zeitschritten. Zeitschritt 0 beschreibt hier den aktuellen Zeitpunkt.

Abbildung 4.1: Aufbau des Fensters und der Grafik. Das Fenster besteht aus drei Eingabe-Zeitschritten, einem Offset von zwei und einem Label. Es sind jeweils die zur Verfügung stehenden Anfragen, Messungen und das vorherzusagende Label dargestellt. Die Anfragen sind dabei einen Zeitschritt in die Vergangenheit dargestellt, da dies dem tatsächlichen Zeitpunkt der Daten entspricht.

einer Konfiguration für dieses Projekt, einer Datei, welche die Modelle definiert und – sofern nötig – einer eigenen Möglichkeit, das Experiment auszuwerten. Grundsätzlich teilen sich die Projekte dabei die gleiche Konfiguration, welche jedes Projekt aber erweitern kann und dann dynamisch diese geladen wird. Die Konfiguration beschreibt dabei zum Beispiel, wie das Fenster definiert ist, die Parameter fürs Training, die Trainingsdaten, welches Modell verwendet werden soll und dessen Parameter.

Im Folgenden ist das Training eines Modells eines Experiments beschrieben. Das Training eines Modells wird dabei mehrfach durchgeführt, um die Auswirkungen der zufälligen Initialisierung des Modells auf das Ergebnis zu reduzieren. Anschließend werden Trainings, die in allen verwendeten Metriken schlechter sind, direkt automatisch entfernt. Vor dem Start der Trainingsläufe werden die Daten, die verwendet werden sollen, zunächst als `DataFrame`-Liste geladen. Diese werden dann in Trainings- und Validierungsdaten unterteilt. Naheliegend wäre, bei Zeitreihen die letzten Daten für die Validierung zu verwenden, da es im Normalfall Korrelationen zwischen Daten gibt, die nahe beieinander liegen und es sonst zu impliziten Korrelationen zwischen Trainings- und Validierungsdaten kommen kann. Da sich die Gelenke des NAO[6] aber im Laufe des Spiels erhitzen und dies zu anderem Verhalten führt (siehe Abschnitt 3.1.2), eignet sich dies hier nicht, zumal meistens mehrere Spiele zum Trainieren verwendet werden und somit das zuletzt gewählte Spiel die Validierung bestimmen würde. Außerdem ist durch die Abschnitte bereits eine Trennung innerhalb der Daten vorhanden. Diese kann bei fehlenden Zeitschritten zwar klein sein, was eine Korrelation nicht ausschließt, allerdings gibt es auch die natürlichen Grenzen, wenn der Roboter andere Aktionen als Laufen ausführt, welche fast 80 % der Abschnitte aus machen. Aus diesen Gründen wurde sich als Kompromiss dazu entschieden, einzelne Abschnitte für die Validierung zu verwenden. Dabei werden aus jedem Spiel eines Roboters Teile für die Validierung verwendet, um einen möglichst gute Verteilung der Validierungsdaten zu erhalten. Für jedes Spiel eines Roboters werden dazu alle Abschnitte durchmischt und anschließend die Kombination gesucht, welche am nächsten an den gewünschten Teiler für die Validierungsdaten kommt. Hierbei handelt es sich um das klassische Rucksackproblem (Dantzig und Mazur 2007; Salkin und De Kluyver 1975),

welches mit dynamischer Programmierung gelöst wurde. Das Ergebnis wird nachverarbeitet und kann anschließend jeweils im Fenstergenerator verwendet werden.

Beim Training werden zunächst die beiden Fenstergeneratoren für Trainings- und Validierungsdaten erstellt. Dann wird dynamisch das definierte Modell aus dem entsprechenden Experiment geladen. Für jedes Modell werden dabei der Fenstergenerator und die Konfiguration übergeben und aus diesen das jeweilige Modell erstellt. Anschließend wird dies mit Keras trainiert. Dabei wird EarlyStopping verwendet, um das Training frühzeitig zu beenden, wenn das Modell sich länger nicht verbessert hat. Als Optimierer wird *Adam* (Kingma und Ba 2014) verwendet und *Mean Squared Error* (MSE) für den Loss (Brownlee 2018). MSE gibt den quadratischen Fehler zwischen Vorhersage und tatsächlichem Wert an. Dies führt dazu, dass Ausreißer stärker bewertet werden. Somit bietet sich diese Metrik als Loss an, da so auch seltenere Phänomene, die sich leicht anders verhalten, wie zum Beispiel ein Straucheln des Roboters, stärker gewichtet werden.

Für jedes Training wird ein eigener Ordner angelegt, welcher alle nötigen Informationen speichert, um ein Training evaluieren oder wiederholen zu können. Dazu zählen unter anderem die Konfiguration, die initialen Gewichte und ein Speicherpunkt des aktuell besten Modells, aber auch die Validation-Metriken und weitere Metadaten für Evaluationen.

Das Training kann jederzeit unterbrochen werden. Dazu wird stets der aktuelle Trainingszustand gespeichert. Zusätzlich werden bei Abbruch die Pfade der durchgeführten und des aktuellen Trainings gespeichert. Beim nächsten Start werden diese automatisch geladen und das Training an der pausierten Stelle fortgesetzt.

4.6 Trainer

Um eine Reihe an bestimmten Trainings auszuführen, wurden zwei Trainer implementiert. Zum einen ein Trainer für Rastersuchen, welche in Abschnitt 5.6 durchgeführt werden und zum anderen einen Trainer, der die Kreuzvalidierungen in Abschnitt 5.8 durchführen kann. Alle Trainer erwarten eine initiale

Konfiguration und eine Trainingsfunktion, welche das eigentliche Training ausführen kann. Außerdem kann angegeben werden, wie oft jedes Training durchgeführt werden soll. Wird ein Training pausiert, sichern die Trainer jeweils ihren Zustand und setzen automatisch wieder auf diesem auf beim nächsten Start.

Für eine Rastersuche werden zusätzlich Listen für einen oder mehrere Parameter der Konfiguration erwartet. Es werden dann nacheinander alle Kombinationen durchgeführt. Eine Parallelisierung ist nicht möglich, da die verwendete und in Abschnitt 5.3 beschriebene Trainingshardware dafür nicht ausreicht.

Für die Kreuzvalidierung können Gruppen von Trainingsdaten angegeben werden. Diese werden dann jeweils einmal zur Evaluation verwendet, während die restlichen Daten für das Training benutzt werden. Dabei wird die Evaluation für jedes Training jeweils mit abgespeichert.

Die Trainer hätten voraussichtlich auch mit *Scikit-learn* (Pedregosa u. a. 2011) implementiert werden können. Da die Integration mit Keras aber komplizierter schien als eine eigene Implementierung, wurde dies letztendlich nicht umgesetzt. Außerdem wurde mit *KerasTuner* (O'Malley u. a. 2019) ein Parameter Optimierer implementiert. Dieser wurde schlussendlich aber nicht verwendet.

Um mehrere Trainer, Trainings und Evaluationen auf einmal durchführen zu können, wurde eine Möglichkeit geschaffen, diese gesammelt definieren und durchlaufen lassen zu können. Hier wird der Zustand nicht gespeichert, wenn das Training pausiert wird, sodass der Aufrufer dafür selber Sorge tragen muss, bereits vollendete Trainings zu entfernen. Allerdings wird erkannt, wenn das Training pausiert wurde und entsprechend darauf hingewiesen sowie passende Optionen zum Wiederaufsetzen angeboten.

4.7 Evaluation

Die Idee für die Evaluation ist, diverse Trainings gegenüberzustellen, um diese dann manuell auswerten zu können. Dazu wurde eine Evaluation implementiert, welche durch diverse Bausteine erweitert werden kann. Die Evaluation nimmt dabei eine Liste von Testdateien, welche für diese genutzt werden sollen, sowie die Trainings, die ausgewertet werden sollen. Sofern eine Kreuzvalidierung

vorliegt, wird diese verwendet und die Testdateien, auf denen eigentlich ausgewertet werden soll, ignoriert. Ist eine explizite Auswertung auf den Testdateien gewünscht, ist dies aber auch möglich. Ansonsten wird die Evaluation verwendet, sofern das Projekt, mit dem das Training trainiert wurde, keine eigene Evaluation hat. In dem Fall wird diese verwendet. Alle Evaluationen nutzen dabei die Auswertung auf Testdaten, die Keras bietet. Da die Evaluation einiges an Zeit benötigt und das wiederholte auswerten somit viel Zeit braucht, werden die Ergebnisse für eine Liste von CSV-Daten, welche für die Evaluation genutzt werden sollen, für jedes Netz gespeichert, sodass die gleiche Evaluation nur einmal durchgeführt wird und anschließend das Ergebnis direkt abgerufen werden kann. Um die Ergebnisse auszuwerten, stehen verschiedene Visualisierungen zur Verfügung. Dabei gibt es sowohl Textform als auch diverse Abbildungen, welche teilweise auch in dieser Arbeit verwendet werden.

Um aus allen Trainings diejenigen zu finden, die gerade betrachtet werden sollen, wurden verschiedene Bausteine implementiert, um die Trainings filtern und sortieren zu können. Diese lassen sich dann beliebig miteinander kombinieren. Eine Möglichkeit sind dabei die Metriken und die Ergebnisse von Auswertungen auf Testdaten. Eine zweite Möglichkeit sind die Anzahl der Parameter, die ein Modell hat. Außerdem kann nach bestimmten Konfigurationen und Konfigurationsmengen sortiert und gefiltert werden. Eine weitere Möglichkeit bietet das Gruppieren von gleichen Trainings. Dabei kann definiert werden, welche Parameter einer Konfiguration für den Vergleich verwendet werden sollen und es können somit unterschiedliche Gruppen für Vergleiche erstellt werden.

5 Experimente

In diesem Kapitel werden die Experimente vorgestellt, die durchgeführt wurden, um herauszufinden, ob sich Gelenkwinkel vorhersagen lassen und wie gut dies generalisiert. Dazu wird in Abschnitt 5.1 zuerst darauf eingegangen, wie die Vorhersage genau aufgebaut ist, bevor in Abschnitt 5.2 die Referenzimplementierungen vorgestellt werden. Anschließend wird in Abschnitt 5.3 die Trainingshardware und in Abschnitt 5.4 die Verwendung der Daten beschrieben. In Abschnitt 5.5 werden die Architekturen vorgestellt und in Abschnitt 5.6 dann die Experimente, die durchgeführt wurden. Danach wird in Abschnitt 5.7 ein Überblick über die Experimente und deren Ergebnisse gegeben. Abschließend wird sich in Abschnitt 5.8 angeschaut, wie gut die Vorhersage in unterschiedlichen Bereichen generalisiert.

Der Begriff Modell wird im Folgenden für eine bestimmte Konfiguration einer neuronale Netze Architektur verwendet, während der Begriff Netz für eine trainierte Instanz eines Modells verwendet wird.

5.1 Ausgangslage

Um zu verstehen, wie die Vorhersage aufgebaut ist, wird sich dies hier einmal genau angeschaut. Der Zeitpunkt, zu dem die Vorhersage gemacht werden soll, ist, wenn neue Gelenkwinkel zur Verfügung stehen und bevor neue Gelenkwinkel angefragt werden, damit die Vorhersage als Eingabe für eine neue Anfrage verwendet werden kann. Dazu stehen die aktuelle und die letzten Messungen sowie die letzten angefragten Gelenkwinkel zur Verfügung. Wie bereits in der Motivation beschrieben, gibt es eine Verzögerung von bis zu drei Zeitschritten von der Anfrage von Gelenkwinkeln bis zur Messung der Auswirkungen. Somit kann die Messung für in zwei Zeitschritten vorhergesagt werden, da

die dritte Messung von den in diesem Zeitschritt angefragten Gelenkwinkeln beeinflusst wird und somit nicht in den Trainingsdaten ist. Um diesen Wert vorhersagen zu können, werden in den meisten Experimenten die letzten drei Messungen und Anfragen verwendet, da aus diesen eine aktuelle Geschwindigkeit und Beschleunigung berechnet werden kann und diese teilweise auch für die Referenzimplementierung benötigt werden. Wie bereits in Abschnitt 4.4 beschrieben, kann mit diesen Informationen ein Generator erstellt werden, der ein entsprechendes Fenster aufbereitet. Das Fenster hat in diesem Fall eine Größe von fünf, wobei die Zeitpunkte 0 bis 2 als Eingabe und 4 als Label verwendet werden. Mit der bereits erklärten Verschiebung der Anfragen in der Eingabe und der Verschiebung der Zeitachse, um den Nullpunkt in diesem Kontext korrekt darzustellen, führt dies zu dem in Abbildung 5.1 dargestellten Fenster.

5.2 Referenzimplementierung

Um eine Aussage darüber zu treffen, ob ein gelerntes Modell besser ist als der aktuelle Zustand ohne Vorhersage und ob sich die Inferenz eines neuronalen Netzes überhaupt lohnt, wurden zunächst unterschiedliche Referenzimplementierungen in Keras geschrieben. Die einfachste ist ein Modell ohne Vorhersage, welches den aktuellen Zustand wieder spiegelt. Dazu wird einfach die aktuelle Messung als Vorhersage verwendet, welche im Folgenden als *Messung* bezeichnet wird. Das zweite Modell, im Folgenden als *Anfrage* bezeichnet, geht davon aus, dass die letzte Anfrage in drei Zeitschritten erreicht wird und sagt entsprechend diese vorher. Das dritte Modell geht davon aus, dass die Änderung der Anfragen umgesetzt werden und addiert entsprechend die Änderung zwischen den Anfragen, die bis dahin umgesetzt werden, auf die Messung. Dieses wird nachfolgend als *Änderung* bezeichnet. Eine beispielhafte Vorhersage der drei Referenzimplementierungen ist in Abbildung 5.2 zu sehen. Diese decken somit sowohl den Istzustand als auch einfache Vorhersagen ab, die davon ausgehen, dass angefragte Gelenkwinkel ganz oder zumindest die Änderung erreicht wird. Erwartet wird dabei, dass die tatsächliche Messung im Normalfall irgendwo zwischen den letzten beiden Referenzimplementierungen liegt.

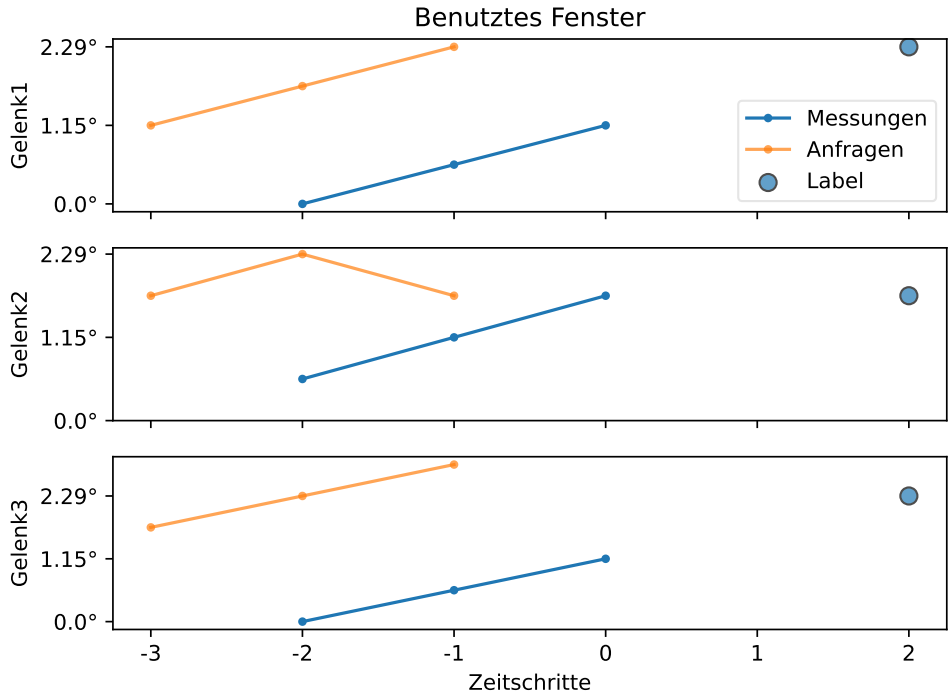


Abbildung 5.1: Beispiele für Eingaben. Zeitschritt 0 beschreibt den aktuellen Zeitpunkt. Es sind die verwendeten letzten Anfragen und Messungen sowie die vorherzusagende Messung als Label dargestellt.

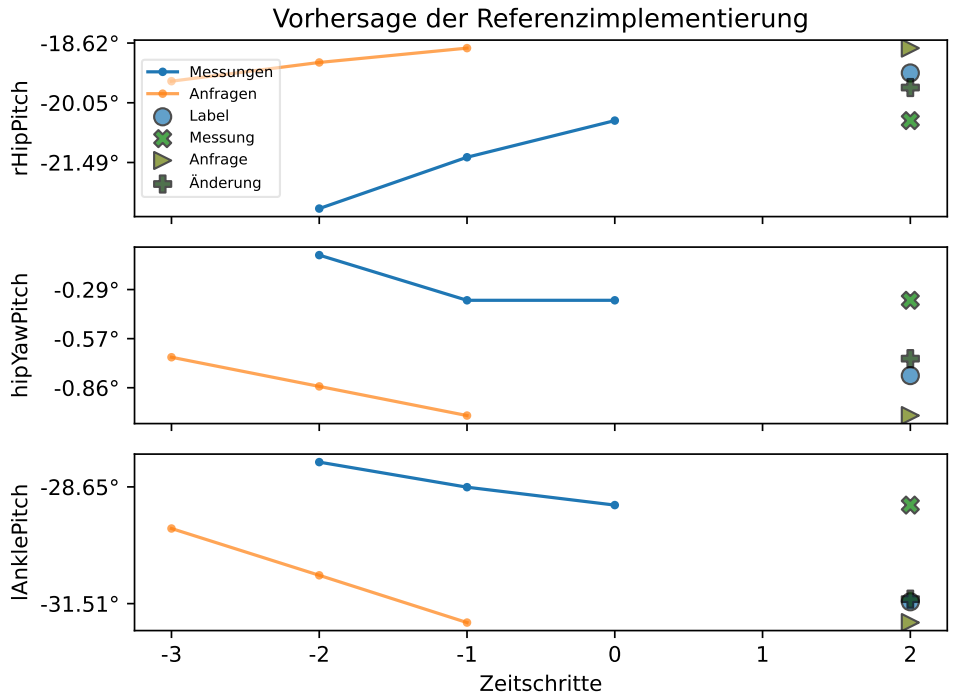


Abbildung 5.2: Beispielhafte Vorhersage für einzelne Gelenke der verschiedenen Referenzimplementierungen (siehe Abschnitt 5.2). Die Vorhersagen sind als Kreuze in verschiedenen Grüntönen dargestellt.

Für die Evaluation der Modelle werden verschiedene Metriken verwendet, um unterschiedliche Aspekte zu betrachten. Die erste Metrik ist der *Mean Absolute Error* (MAE). Dieser gibt den mittleren absoluten Fehler zwischen Vorhersage und tatsächlichem Wert in der gleichen Einheit an und ist somit leicht zu interpretieren. Die zweite Metrik ist der *Mean Absolute Percentage Error* (MAPE), welcher den prozentualen Fehler zwischen Vorhersage und tatsächlichem Wert angibt. Welcher Wert ein guter Wert ist, ist hierbei domänenabhängig. Zusätzlich ist der Wert generell größer, je näher die Eingabe an null liegt. Aus Tabelle 3.1, welche erreichte Gelenkwinkel während des Laufens zeigt, lässt sich daher bereits ableiten, dass der zu erwartende Wert in diesem Fall größer sein könnte, da die Roll-Gelenke einen Durchschnitt nahe

null haben und somit kleinere Änderungen bereits einen größeren prozentualen Einfluss haben werden. Dies führt möglicherweise auch dazu, dass der MAPE für die Pitch-Gelenke im Vergleich so klein ist, dass er auf Abbildungen nicht erkennbar ist.

In Abbildung 5.3 wurden die Referenzimplementierungen exemplarisch evaluiert. Die Evaluation auf unterschiedlichen Daten führt dabei stets zu ähnlichen Ergebnissen. Beim MAE ist zu sehen, dass dieser für die Pitch-Gelenke deutlich größer ist, als für die Roll-Gelenke. Dies hängt damit zusammen, dass sie sich beim Laufen schneller und über größere Distanzen bewegen, als die Roll-Gelenke. Weiterhin hat die Referenzimplementierung, welche die Änderung der Anfragen verwendet, mit Abstand den kleinsten Fehler. Beim MAPE fällt auf, dass der prozentuale Fehler bei der Hüfte und den Roll-Gelenken sehr groß ist. Der durchschnittliche Fehler beträgt dabei ein vielfaches des Durchschnittswertes. Dies hängt damit zusammen, dass diese sich in der Nähe von null bewegen und somit kleine Änderungen bereits einen großen prozentualen Einfluss haben. Dabei ist auch zu sehen, dass die Referenzimplementierung, welche die Anfrage verwendet, für die Roll-Gelenke den größten Fehler aufweist und sogar größer ist, als wenn die aktuelle Messung verwendet wird. Dies lässt darauf schließen, dass die Anfrage im Durchschnitt weiter entfernt ist, als sich das Gelenk in der Zeit überhaupt bewegt. Wie vermutet, lassen sich für die Pitch-Gelenke in der Abbildung aus dem MAPE keine Erkenntnisse gewinnen, da sie zu klein sind im Vergleich zu den Roll-Gelenken.

Keras-Metriken können nur den Durchschnitt und keine Verteilung des Fehlers berechnen. Im Folgenden wird beispielhaft die Verteilung für die Referenzimplementierung der Messung angeschaut. Dies soll Aufschluss darüber geben, ob es eine oder mehrere Häufungen in den Daten gibt, wie diese verteilt sind und wie die Metriken – also der Durchschnitt jeweils – entsprechend zu interpretieren sind. Der Fehler dieser Referenzimplementierung ist auch gleichzeitig die Änderung der Gelenke zwischen Messung und Vorhersage, da der Fehler der Unterschied zwischen einer Messung und der in zwei Zeitschritten ist. Dazu sind in Abbildung 5.4 der MAE, sowie die Verteilung dargestellt. Zu sehen ist, dass jeweils ein Großteil der Daten eine sehr kleine Änderung hat, es aber starke Ausreißer nach oben gibt. Auch lässt sich gut erkennen,

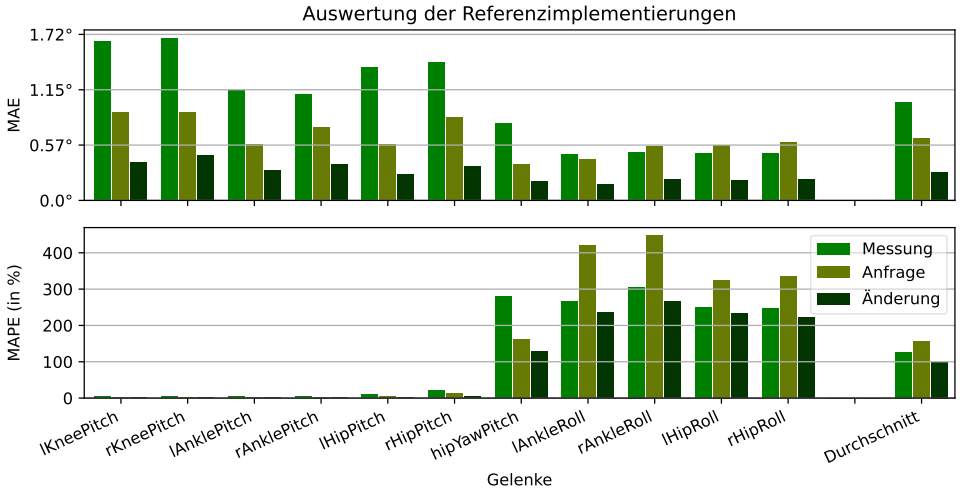


Abbildung 5.3: Beispielhafte Evaluation der Referenzimplementierungen (siehe Abschnitt 5.2) von einem Spiel von einem Roboter.

dass es nur eine Häufung gibt, welche mit steigendem Fehler je nach Gelenk unterschiedlich stark abnimmt. Weiterhin fällt auf, dass der Median stets unter dem Mittelwert liegt und der Median auch stets in der unteren Hälfte der mittleren 50 % der Daten liegt.

5.3 Trainingshardware

Für das Training steht ein Rechner vom Projekt B-Human zur Verfügung, auf welchem Modelle trainiert werden können. Der Rechner ist mit folgender Hardware ausgestattet:

- Intel® Core™ **i5-7500** Prozessor, **4-core @ 3,40 GHz**, 6 MB Smart-Cache
- **32 GB (2x 16 GB) DDR4 / PC2400 UDIMM** Arbeitsspeicher
- **NVIDIA® Gigabyte GeForce RTX 2060 Windforce OC 6G** Grafikkarte, 6 GB GDDR6

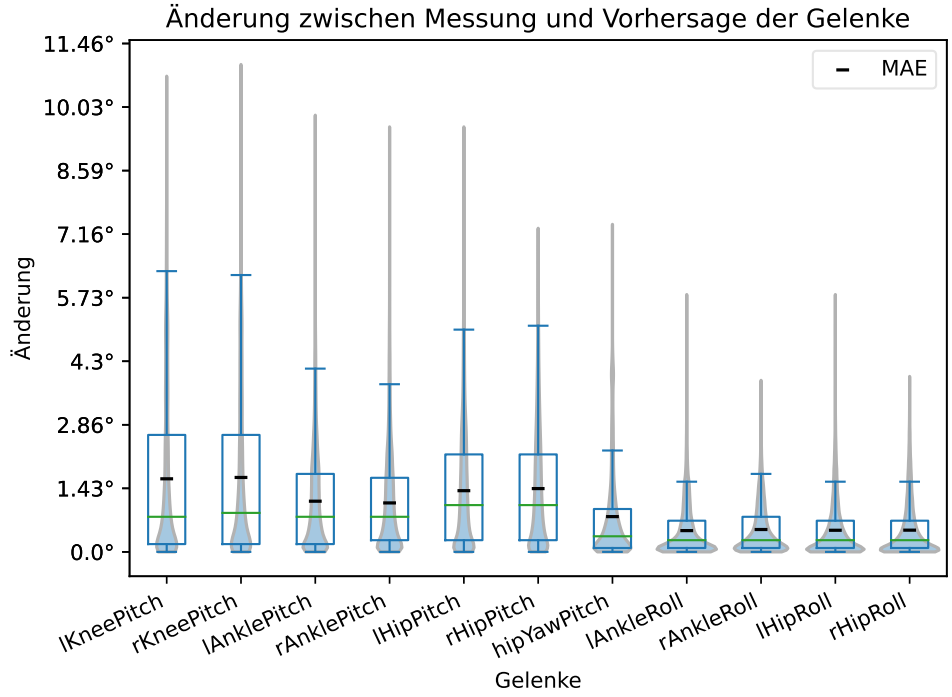


Abbildung 5.4: Beispiel für die Verteilung der Änderung der Winkel in zwei Zeitschritten in einem Spiel von einem Roboter in 2019. Dies entspricht gleichzeitig der Referenzimplementierung der Messung. Dargestellt ist dabei der MAE und die Verteilung, wobei eine Darstellung gewählt wurde, die möglichst wenig verdeckt.

5.4 Verwendung der Daten

Zunächst wurden nur einige Daten aus dem Jahr 2019 verwendet, um eine Pipeline für die Daten zu erstellen und Modelle zu trainieren. Die Daten aus dem Jahr 2022 standen einerseits zu Beginn noch nicht zur Verfügung und anderseits schien es sinnvoll, zunächst nur Daten aus 2019 für Experimente zu verwenden, da die Roboter damals noch weitestgehend neuwertig waren. Somit kann minimiert werden, dass Verschleißerscheinungen das Ergebnis beeinflussen, wie gut sich die Gelenkwinkel überhaupt vorhersagen lassen und ob Roboter miteinander vergleichbar sind, oder ob sie sich generell zu unterschiedlich verhalten. Im späteren Verlauf der Arbeit werden dann große Teile der zur Verfügung stehenden Daten verwendet.

Zusätzlich hat sich die Steuerungssoftware der Roboter von B-Human seitdem weiterentwickelt. So wurde unter anderem eine einfache Fußwechselvorhersage von einem Zeitschritt hinzugefügt. Außerdem wurde das Balancieren beim Laufen erweitert (Reichenberg und Röfer 2022). Da diese und weitere Elemente bereits in das Laufen eingreifen und versuchen es zu stabilisieren, verändert dies die zur Verfügung stehenden Daten. Auch um eine möglichst gute Vorhersage für das eigentliche Laufen zu bekommen, in das möglichst wenig eingegriffen wird, eignen sich Daten aus 2019 somit besser. Wie stark diese Unterschiede sind, ist allerdings nicht Teil der Arbeit.

5.5 Architekturen

Die Experimente werden jeweils mit verschiedenen parametrisierbaren Architekturen durchgeführt, welche im Folgenden vorgestellt werden. Die grundlegende Idee für die Architekturen ist dabei aus einem Tutorial (TensorFlow Developers 2023). Weiterhin wurde sich mit externen Bibliotheken beschäftigt, welche am Ende aber nicht benutzt wurden, da sie sich entweder als nicht passend für die Daten oder das zu lösende Problem herausgestellt haben. Ein Beispiel ist *PyTorchTS* (Rasul 2021), welche verschiedene aktuelle wissenschaftliche Arbeiten implementiert, sich aber eher für die Vorhersage von Zeiträumen als Zeitpunkten zu eignen scheint.

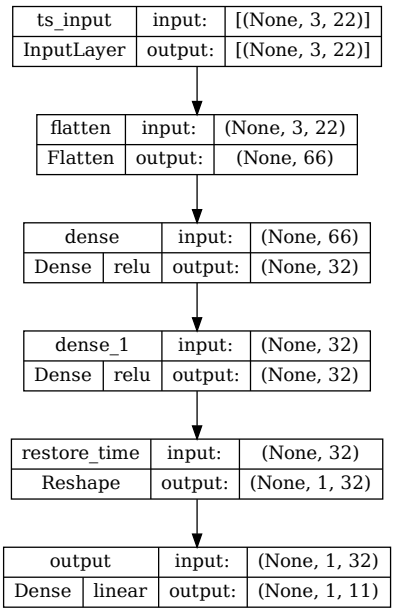


Abbildung 5.5: Beispiel eines Dense-Modells, welches zwei Dense-Schichten verwendet. Es können die Parameter der Dense-Schichten und die Anzahl der Schichten variiert werden.

5.5.1 Dense

Hier wird eine einfache Architektur aus Dense-Schichten, im Folgenden Dense-Modell genannt, verwendet. Die Zeitreihe wird dazu flach gedrückt, wodurch die explizite zeitliche Komponente verloren geht und ein Modell den Zusammenhang lernen muss. Diese wird manuell wieder hergestellt, bevor auf die Ausgabe reduziert wird. Ein Beispiel ist in Abbildung 5.5 dargestellt. Hier lassen sich die Anzahl der Dense-Schichten zwischen Flatten und Reshape, sowie deren Parameterzahl konfigurieren.

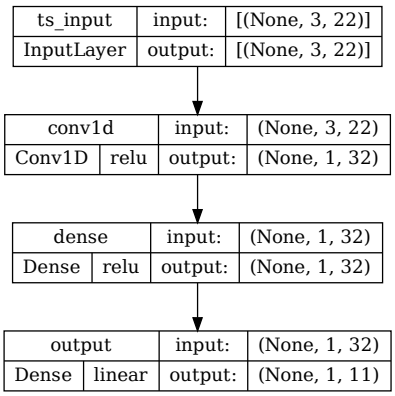


Abbildung 5.6: Beispiel eines Conv-Modells, welches eine Dense-Schicht verwendet. Es können die Parameter der Conv- und Dense-Schichten, sowie die Anzahl der Dense-Schichten variiert werden.

5.5.2 Convolution

Darauf aufbauend wird eine ähnliche Architektur verwendet, im Folgenden Conv-Modell genannt, welche den zeitlichen Aspekt über eine Convolution modelliert. Dies würde es theoretisch auch erlauben, Eingaben anderer zeitlicher Länge zu verwenden, da die Convolution wie ein Fenster über die Eingabe verwendet werden kann. Somit könnten Anwendungsfälle abgedeckt werden, in denen im Nachhinein mehrere aufeinanderfolgende Vorhersagen auf einmal gemacht werden sollen. Ein Beispiel ist in Abbildung 5.6 dargestellt. Hier lassen sich die Anzahl der Filter in der Convolution, sowie wieder die zwischenliegenden Dense-Schichten und deren Parameterzahl konfigurieren.

5.5.3 LSTM

Bevor die verwendete Architektur erklärt wird, folgt zunächst eine Erklärung, was ein LSTM Netz überhaupt ist und wie es funktioniert. Ein LSTM beschreibt allgemein eine spezielle Form rekurrenter neuronaler Netze (RNN), eine Klasse von Netzen mit einem internen Zustand. Ein RNN ist grundsätz-

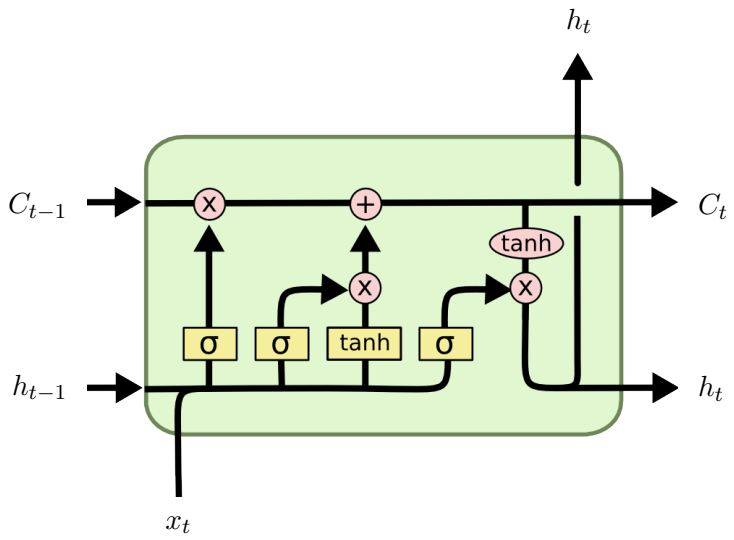


Abbildung 5.7: Darstellung einer Schicht einer *Long Short-Term Memory* (LSTM)-Zelle. Grafik übernommen aus Kuball (2020, S. 71),

lich ein Netz mit Schichten, welche Schleifen enthalten und darüber Zustand speichern können. Wenn man die Schleife abrollt, kann man sich eine Schicht als mehrere Kopien desselben Netzes vorstellen, die jeweils eine Nachricht an einen Nachfolger weitergeben. Die von Hochreiter und Schmidhuber (1997) eingeführten LSTMs bestehen aus einem oder mehreren gleichnamigen Schichten. Diese sind gut darin, Informationen sowohl über kurze, als auch über lange Zeit zu speichern. Dazu besteht eine Schicht, im Folgenden als Zelle bezeichnet, selbst aus mehreren Schichten und Operationen. Eine LSTM-Schicht ist in Abbildung 5.7 dargestellt. Unten ist hierbei die Eingabe der vorherigen Schicht x_t und oben die Ausgabe in die nächste Schicht h_t. Von links nach rechts sind Eingaben und Ausgaben anderer gleicher Schichten der selben Zelle. C ist dabei die Erinnerung, also der Zustand, welche durch die Zelle läuft. Auf dieser kann gelöscht ($\times$) oder geschrieben ($+$) werden. Was gelöscht oder geschrieben wird, wird von vier inneren Schichten bestimmt. Dabei wird aus der Ausgabe der letzten Schicht dieser Zelle und der Eingabe von links nach rechts zuerst bestimmt, was gelöscht werden soll, dann was geschrieben werden soll und anschließend wird aus C und den Eingaben die Ausgabe der Schicht bestimmt

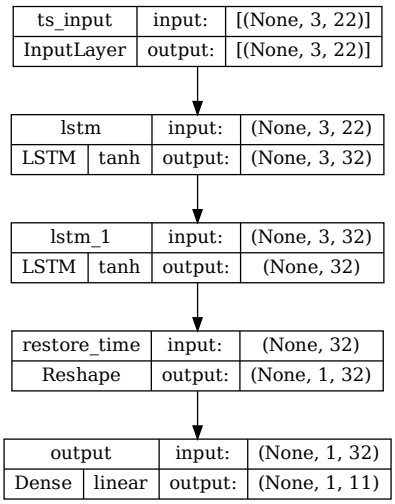


Abbildung 5.8: Beispiel LSTM-Modell, welches eine extra LSTM-Schicht verwendet. Es können die Parameter der LSTM-Schichten und die Anzahl der Schichten variiert werden.

(Kuball 2020, S. 71; Olah 2015).

Die in dieser Arbeit verwendete Architektur, im Folgenden LSTM-Modell genannt, verwendet eine LSTM-Schicht, welche einen Zeitschritt vorhersagt, gefolgt von einer Dense-Schicht, welche auf die Ausgabe reduziert. An den Anfang können zusätzliche LSTM-Schichten hinzugefügt werden, welche jeweils die gesamte Sequenz weitergeben. Ein Beispiel ist in Abbildung 5.8 dargestellt. Es lassen sich die Anzahl der LSTM-Schichten am Anfang, sowie deren Parameterzahl konfigurieren.

5.5.4 Änderungsvorhersage

Wie in Abbildung 5.4 bereits dargestellt, ist die zu lernende Änderung relativ klein im Vergleich zum Wertebereich (siehe Tabelle 3.1). Aus diesem Grund bietet es sich möglicherweise an, dass Modelle nur die Änderung vorhersagen. Damit diese Modelle kompatibel zu den bisherigen sind, wird dies in der

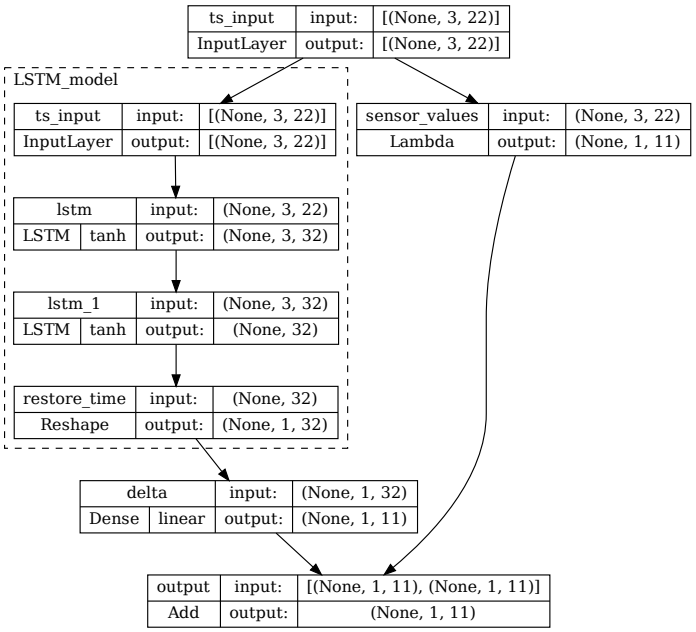


Abbildung 5.9: Beispiel Modell, welches die anderen Modelle kapselt und dabei die letzte Schicht ersetzt.

Architektur gelöst und die Modelle geben weiterhin die Winkel der Gelenke zurück. Dies ist als Dekorator implementiert. Die letzte Schicht ist bei allen Modellen eine Dense-Schicht, welche auf die Ausgabeparameter reduziert. Diese wird abgeschnitten und durch eine Dense-Schicht ersetzt, welche die Änderung bereitstellt und mit Nullen initialisiert wird, was dem Modell hilft, besser und schneller zu lernen, da die Änderungen in der Regel eher klein sind und somit bereits einen guten Ausgangswert liefert. Um Gelenkwinkel vorherzusagen, wird eine Skip-Verbindung genutzt, welche die letzten gemessenen Gelenkwinkel bereitstellt. Auf diese werden dann die Änderungen addiert. Ein Beispiel mit dem Modell aus Abbildung 5.8 ist in Abbildung 5.9 dargestellt. Bei Modellen mit dieser Kapselung wird zur besseren Unterscheidung im Weiteren von Modellen gesprochen, welche die Änderung vorhersagen, obwohl das Netz weiterhin Winkel ausgibt.

5.6 Experimente

Da nicht klar ist, ob und wie sich die Gelenke am besten vorhersagen lassen, werden in Abschnitt 5.6.1 zunächst verschiedene Modelle, welche alle Gelenke auf einmal vorhersagen, ausprobiert, ohne dabei die Laufzeit auf dem NAO zu beachten. Um den Suchraum zu durchsuchen, werden mehrere Rastersuchen durchgeführt, um anschließend mit den gewonnenen Erkenntnissen zielgerichtet weitere Experimente durchführen zu können. Dabei wird sich in Abschnitt 5.6.2 angeschaut, wie gut das für jedes einzelne Gelenk funktioniert. Außerdem wird in Abschnitt 5.6.3 MAPE als Loss ausprobiert und abschließend werden in Abschnitt 5.6.4 andere Anzahlen an Zeitschritten als Eingabe evaluiert.

Für jedes Modell wurden dabei mehrere Netze trainiert, um den Einfluss der zufälligen Initialisierung der Gewichte zu reduzieren. Dennoch bleibt eine gewisse Schwankung in den Ergebnissen enthalten. Die Modelle wurden alle auf dem gleichen kleineren Datensatz von ungefähr 100 min trainiert und entsprechend auf einem kleinen Testset evaluiert, um die Ergebnisse vergleichen zu können. Dies ist nicht speziell balanciert, sollte aber dennoch sinnvolle Ergebnisse liefern, um die verschiedenen Trainings miteinander vergleichen zu können.

Beim Auswählen des besten Netzes pro Modell ist aufgefallen, dass das Netz mit dem minimal schlechteren MAE häufig den besseren MAPE hat. Hier wurde sich bei deutlich kleinerem MAPE für dieses Netz entschieden. Ansonsten wurde sich für das Netz mit dem kleineren MAE entschieden, da andere, während des Trainings aufgezeichnete Metriken, für diese Netze im Normalfall auch besser waren.

5.6.1 Alle Gelenke

Um ein Gefühl dafür zu bekommen, wie die verschiedenen Architekturen für den Anwendungsfall funktionieren, wurde für jede Architektur eine Rastersuche durchgeführt, welche anschließend genauer betrachtet werden. Wie in der Übersicht in Abbildung 5.10 dargestellt, wurden größtenteils relativ kleine Netze verwendet. Modelle, die deutlich mehr als 100 000 Parameter haben, wurden nicht betrachtet. Dabei fällt im kleinen Parameterbereich eine große

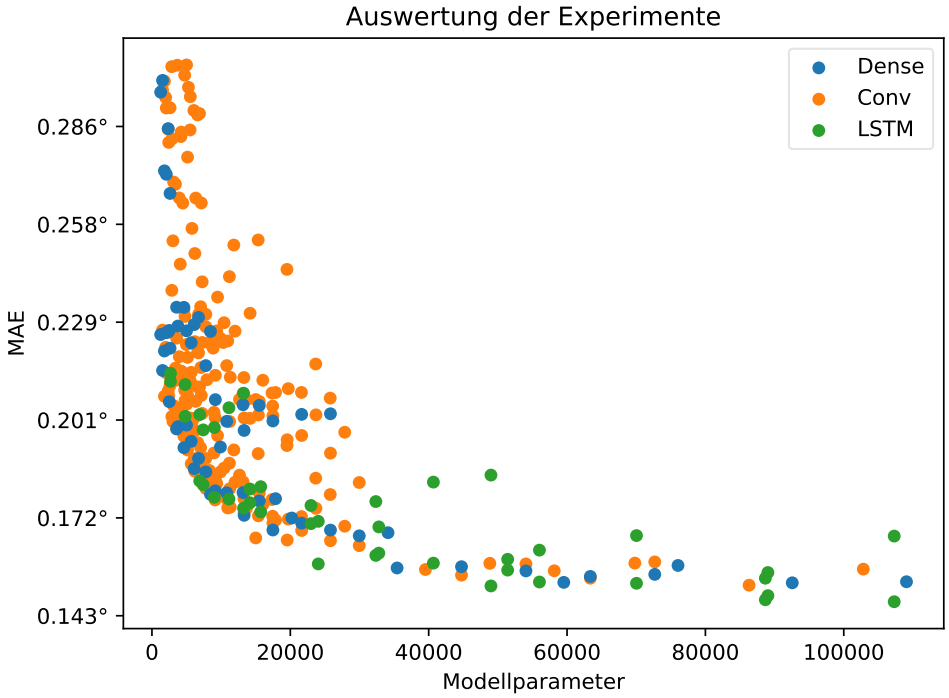


Abbildung 5.10: Visualisierung des MAE für Experimente, die alle Gelenke als Ein- und Ausgabe verwenden, wobei zwischen den verschiedenen Architekturen unterschieden wird. Es sind sowohl die Vorhersage der Gelenkwinkel, als auf die Vorhersage der Änderung enthalten.

Streuung von Dense- und Conv-Netzen nach oben auf, welche im Folgenden noch genauer untersucht wird. Außerdem wurden im Vergleich mehr Conv-Netze trainiert, da diese mit dem Filtern der Convolution einen Freiheitsgrad mehr haben. Während Modelle im kleinen Parameterbereich schnell bessere Ergebnisse liefern, flacht dies bei 15 000 bis 40 000 Parametern ab. Anschließend sind nur noch geringe Verbesserungen erkennbar. In diesem Bereich liegt somit der beste Kompromiss aus Parametern und möglichst geringem Fehler.

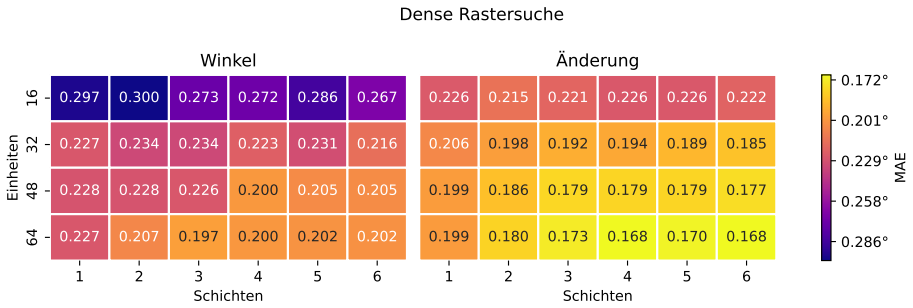


Abbildung 5.11: Übersicht der initialen Rastersuche von Dense-Modellen.

Rastersuchen

In Abbildung 5.11 sind zunächst ein Großteil der Dense-Netze dargestellt. Hier ist zu sehen, dass 16 Dense-Einheiten zu wenig sind, da sie deutlich schlechtere Ergebnisse liefern und somit die bereits erwähnte Streuung verursachen. Weiterhin ist eindeutig zu sehen, dass Netze, welche die Änderung vorhersagen, bei sonst gleichem Modell besser abschneiden. Die besten Ergebnisse werden dabei von den größten Netzen erzielt.

Als Nächstes ist ein Großteil der Rastersuche der Conv-Netze dargestellt. Dazu sind in Abbildung 5.12 die Ergebnisse für die Winkel und in Abbildung 5.13 die Ergebnisse für die Änderungen dargestellt. Da es mit den Filtern für die Convolution einen zusätzlichen Freiheitsgrad gibt, wurden die Raster nach diesem gruppiert. Dabei fällt wieder auf, dass 16 Dense-Einheiten schlechtere Ergebnisse liefern und die angesprochene Streuung verursachen. Ebenso sind die Ergebnisse für 16 Filter im Schnitt sichtbar schlechter als die mit mehr Filtern. Ansonsten liefern größere Filter im Schnitt leicht bessere Ergebnisse. Weiterhin ist an der Skala des MAE zu sehen, dass die Änderung der Vorhersage auch hier bessere Ergebnisse liefert. Auch hier werden die besten Ergebnisse wieder von den größten Netzen erzielt.

Abschließend ist in Abbildung 5.14 die LSTM-Rastersuche dargestellt. Aufgrund der gesetzten Parameterbegrenzung von ungefähr 100 000 ist dieses Raster nicht vollständig. Hier lassen sich die gleichen Schlüsse ziehen, wie bereits bei den Dense-Netzen.

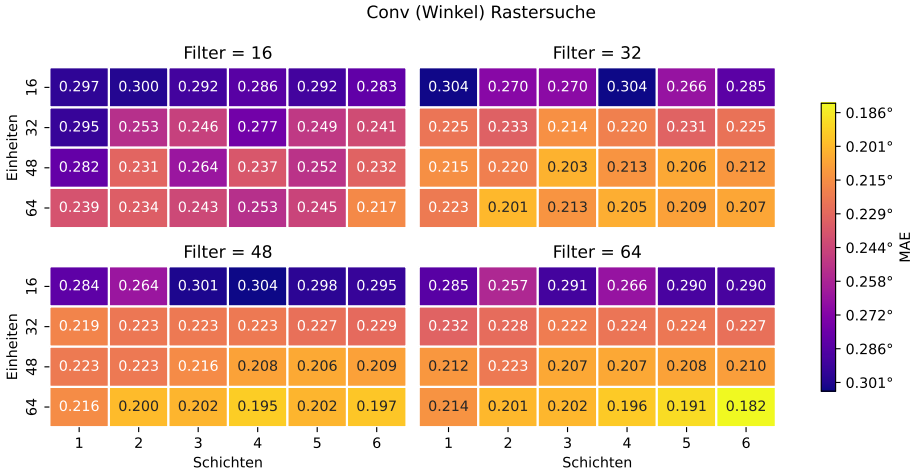


Abbildung 5.12: Übersicht der initialen Rastersuche von Conv-Modellen. Diese sind dabei nach den Filtern der Convolution gruppiert.

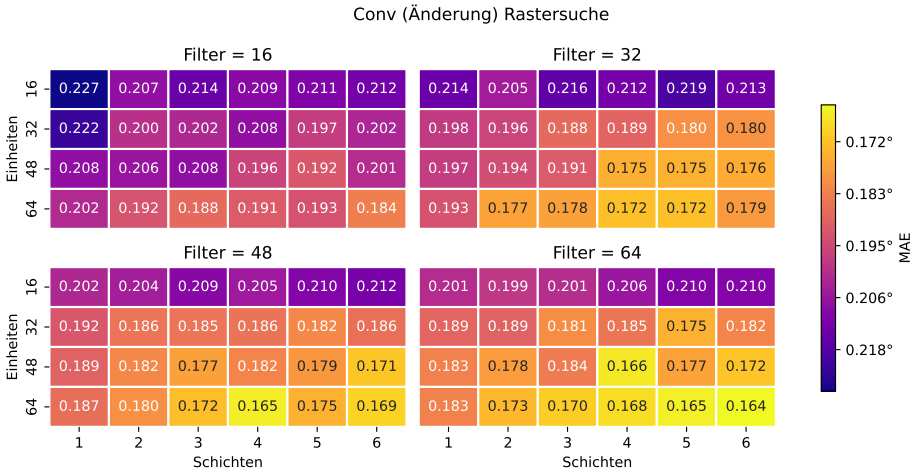


Abbildung 5.13: Übersicht der initialen Rastersuche von Conv-Modellen, welche die Änderung vorhersagen. Diese sind dabei nach den Filtern der Convolution gruppiert.

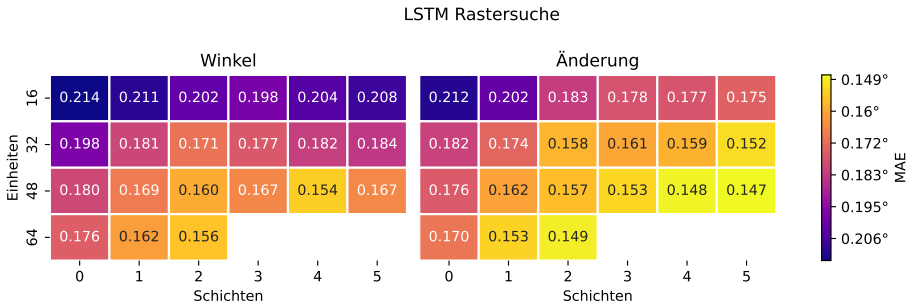


Abbildung 5.14: Übersicht der initialen Rastersuche von LSTM-Modellen.

Wie aus den Rastersuchen bereits hervorgegangen ist und in Abbildung 5.15 nochmal deutlich gezeigt wird, liefern solche Modelle bessere Ergebnisse, welche die Änderung der Winkel vorhersagen. Dabei ist es größtenteils irrelevant, welche Architektur benutzt wird. Aus diesem Grund werden im Weiteren stets Modelle verwendet, welche die Änderung der Winkel vorhersagen.

Architekturvergleich

Da die Anzahl der Parameter bei den LSTM-Modellen größer ist, wurden im Nachhinein noch größere Dense- und Conv-Modelle trainiert, um eine bessere Vergleichbarkeit zu erhalten. Diese sind bereits in den Abbildungen 5.10 und 5.15 enthalten. In Abbildung 5.16 sind nochmal nur die Netze dargestellt, welche die Änderung vorhersagen. Dabei ergibt sich aus dem Verlauf anhand der Parametermenge kein eindeutiges Ergebnis, welche Architektur die Beste ist. Dazu müssten sowohl die Laufzeit als auch die Kombinationen von Tiefe und Parametern pro Schicht mitbetrachtet werden. Allerdings lässt sich erkennen, dass LSTM-Modelle bei ähnlichen Parametern häufig mit die besten Ergebnisse liefern.

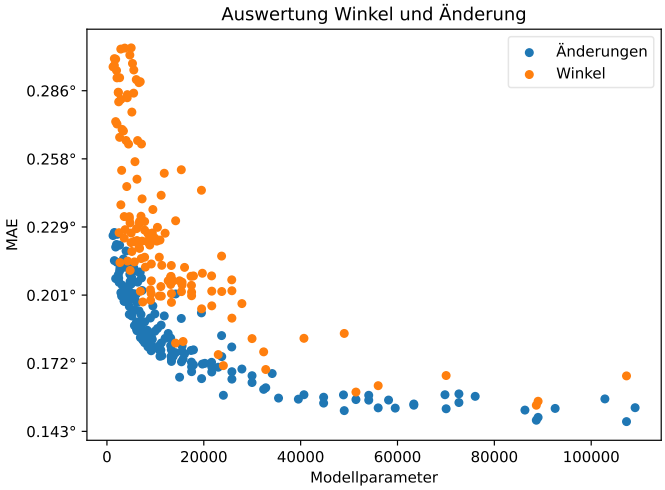


Abbildung 5.15: Visualisierung des MAE für Experimente, die alle Gelenke als Ein- und Ausgabe verwenden, wobei zwischen Modellen, welche die Winkel und die Änderung der Winkel vorhersagen unterschieden wird.

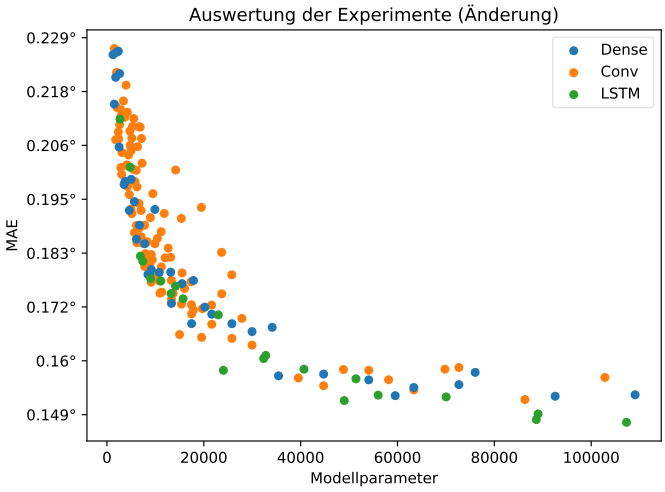


Abbildung 5.16: Visualisierung des MAE für Experimente, die alle Gelenke als Ein- und Ausgabe verwenden, wobei zwischen den verschiedenen Architekturen unterschieden wird. Im Gegensatz zu Abbildung 5.10 sind hier nur die Netze dargestellt, welche die Änderung vorhersagen.

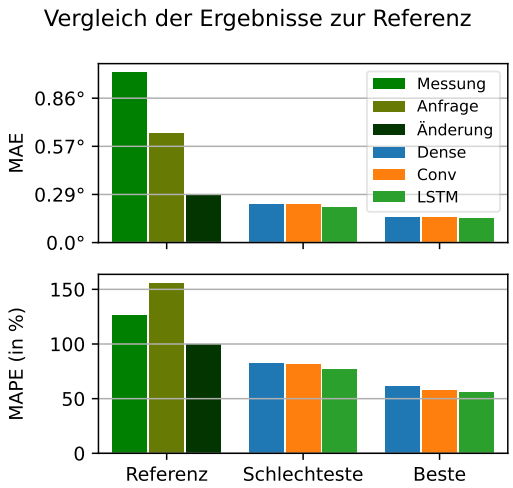


Abbildung 5.17: Gegenüberstellung der Referenzimplementierungen (siehe Abschnitt 5.2) zu den schlechtesten und besten Netzen pro Architektur, welche die Änderung vorhersagen.

Vergleich zur Referenzimplementierung

Um einen Überblick zu bekommen, wie diese Experimente im Vergleich zur Referenzimplementierung abschneiden, werden sie im Folgenden miteinander verglichen. Dazu sind in Abbildung 5.17 die Referenzimplementierungen zusammen mit den besten und schlechtesten Netzen pro Architektur dargestellt. Bereits die schlechtesten Netze für die Vorhersage der Änderung sind besser als alle Referenzimplementierungen. Diese haben im Schnitt allerdings nur etwas über 2000 Parameter und sind somit recht klein. Die besten Netze sind in beiden Metriken ungefähr doppelt so gut wie die beste Referenzimplementierung (Änderung). Im Vergleich zum aktuellen Zustand, den die aktuelle Messung darstellt, sind diese um ein Vielfaches besser. So ist der MAE um den Faktor sieben besser und der MAPE hat sich mehr als halbiert. Da die Sensoren der Motoren, wie in Abschnitt 3.1.1 beschrieben eine Genauigkeit von ungefähr 0,1° haben, wird dieser Wert als untere Schranke für die Vorhersagegenauigkeit angenommen. Mit einem MAE von ungefähr 0,15° sind diese besten Netze somit bereit relativ nah an diesem Optimum im Vergleich zum aktuellen Zustand.

Zusammenfassung

In diesem Abschnitt wurden für die verschiedenen Architekturen in Rastersuchen diverse Modelle trainiert, um zu ermitteln, wie diese auf dem Anwendungsfall funktionieren. Dabei wurden jeweils alle Gelenke als Ein- und Ausgabe verwendet. Es wurde in diesem Abschnitt stets der Fehler in Abhängigkeit zur Parameterzahl betrachtet und die Laufzeit der Modelle außer Acht gelassen. Es wurden zusätzlich weitere Conv- und Dense-Modelle mit größeren Parameterzahlen trainiert, um eine gewisse Vergleichbarkeit zu den LSTM-Modellen zu gewährleisten, welche größere Anzahl an Parametern aufweisen. Mit den Ergebnissen aus Abbildung 5.17 lässt sich die Frage, ob sich die Gelenke vorhersagen lassen, bereits positiv beantworten. Weiterhin lässt sich sagen, dass die Vorhersage voraussichtlich schnell genug für den NAO[6] sein wird, da bereits kleine Netze deutlich bessere Ergebnisse liefern als der aktuelle Zustand (Messung). Bei den unterschiedlichen Architekturen lässt sich festhalten, dass Modelle, welche die Änderung der Gelenkwinkel vorhersagen und die Winkel der Gelenke zurückgeben, bessere Ergebnisse liefern und somit nur diese im Weiteren verwendet werden sollen. Allerdings lässt sich nicht klar sagen, welche Architektur die Beste ist. Deshalb werden im Weiteren alle Architekturen verwendet.

5.6.2 Jedes Gelenk einzeln

Nachdem bereits in Abbildung 5.3 für die Referenzimplementierung jedes Gelenk einzeln evaluiert wurde, soll dies nun auch für die Architekturen gemacht werden. Dies soll zum einen einen genaueren Überblick darüber geben, wie gut sich die einzelnen Gelenke vorhersagen lassen und zum anderen feststellen, ob es sinnvoller ist, ein Modell pro Gelenk zu nutzen oder ein Modell für alle Gelenke, wie im vorherigen Abschnitt.

Dazu werden im Folgenden die Experimente beschrieben, bei denen jedes Gelenk einzeln trainiert wurde. Dabei gibt es zwei Möglichkeiten an Eingaben in ein Modell. Entweder können die Daten aller Gelenke oder nur die des zu vorhersagenden Gelenkes verwendet werden. In der ersten Variante, im Folgenden *Eingabe alle* genannt, stehen die Informationen von allen Gelenken

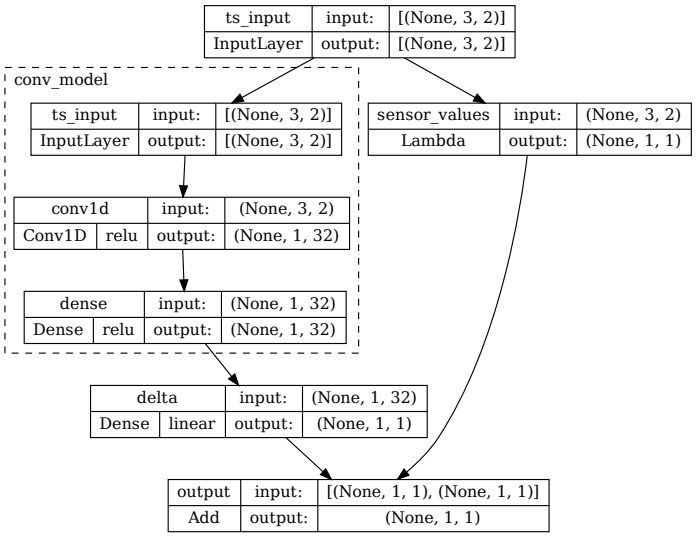


Abbildung 5.18: Beispiel Modell, welches das gleiche Gelenk als Eingabe verwendet. Bei allen Gelenken als Eingabe unterscheidet sich diese nicht von der in Abbildung 5.9, während die Ausgabe wie in diesem Beispiel ist.

zur Verfügung, sodass mögliche Zusammenhänge gelernt werden können und der einzige Unterschied zum vorherigen Abschnitt ist, dass es ein Modell für jedes Gelenk gibt, welches jeweils den Wert für das jeweilige Gelenk vorhersagt. In der anderen Variante, im Folgenden *Eingabe einzeln* genannt, werden die Gelenke dagegen als unabhängig voneinander betrachtet, da die jeweilige Vorhersage nur die Informationen von sich selbst verwendet. Für beide Varianten können, wie in Abbildung 5.18 dargestellt, die gleichen Architekturen wie bisher verwendet werden und nur die Anzahl der Ein- und Ausgabe muss angepasst werden. Somit besteht hier ein Training genau genommen aus elf Trainings. Im Folgenden sind mit den Begriffen jeweils alle elf Netze gemeint, sofern nicht explizit erwähnt, dass es sich um die einzelnen handelt. Da sich die Trainingszeit vervielfacht, wurde ein mehrfaches Trainieren eines Modells hier deutlich reduziert.

Um eine kleine Anzahl von Netzen miteinander zu vergleichen, ist es sinnvoll,

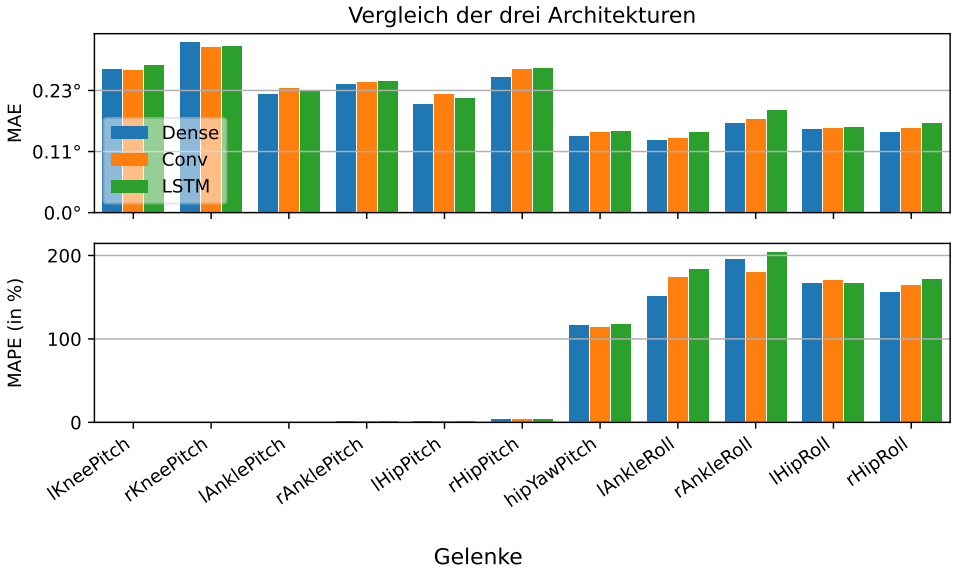


Abbildung 5.19: Evaluation von Beispielnetzen mit einer ähnlichen Menge an Parametern.

sich jedes Gelenk einzeln anzuschauen. Dies ist Beispielhaft in Abbildung 5.19 zu sehen, wo jeweils ein Netz pro Architektur mit ähnlicher Anzahl an Parametern dargestellt ist. Hier ist zu sehen, dass sich die Architekturen in ihrer Vorhersage kaum unterscheiden und für die einzelnen Gelenke ähnliche Ergebnisse liefern. Die Metriken für die einzelnen Gelenke verhalten sich dabei grundsätzlich ähnliche, wie bei der Referenzimplementierung (siehe Abbildung 5.3). Allerdings ist der Unterschied zwischen Pitch- und Roll-Gelenken für den MAE augenscheinlich etwas geringer. Ein Vergleich zu den Referenzimplementierung folgt später in diesem Abschnitt.

Um eine größere Anzahl an Modellen miteinander vergleichen zu können, wird jeweils der Durchschnitt der Metriken über die Netze der Gelenke gebildet. Dies hat den Vorteil, dass sich die Netze wie im vorherigen Abschnitt miteinander vergleichen lassen und auch mit diesen vergleichbar sind. Die Vergleichbarkeit ist gegeben, da es sich in beiden Fällen um den Durchschnitt über die Ausgabe handelt, welche jeweils die einzelnen Gelenke sind.

Experimente

Um die Modelle zu analysieren, wurden jeweils kleinere Rastersuchen sowie Stichproben durchgeführt. Dabei wurden zunächst deutlich kleinere Modelle verwendet, aber auch einige Modelle, wie sie im vorherigen Abschnitt verwendet wurden. In Abbildung 5.20 sind ein Großteil der Netze dargestellt, die nur Werte des Gelenkes selbst als Eingabe bekommen. Hier fällt auf, dass die LSTM- Netze eine schlechtere Performance haben. Dies ist unerwartet, da diese bisher sehr gut funktioniert haben. In der Analyse ist aufgefallen, dass dies unter anderem damit zusammenhängt, dass diese hier mehr Epochen benötigen, um zu einer guten Performance kommen. Es wurden fast immer die maximalen Epochen benötigt. Mit einer Erhöhung der Epochen in einem einzelnen Test, kommen diese dann auf ähnliche Ergebnisse, wie die besten Netze der anderen Architekturen. Für die fünf Modelle mit mehr als 20 000 Parametern wurden mehrere Trainings durchgeführt, weshalb diese weniger raus fallen. Außerdem ist allgemein eine etwas größere Streuung zu sehen. Dies hängt damit zusammen, dass die zufällige Initialisierung einen größeren Einfluss hat, da nur teilweise mehrere Netze pro Modell trainiert wurden. Allgemein ist in dieser Grafik keine genaue Tendenz zu erkennen, wie sich der Fehler bei mehr Parametern entwickelt. Deutlich größere Netze mit über 140 000 Parametern haben einen MAE von ungefähr 0,205°, was darauf schließen lässt, dass die Ergebnisse bei größeren Modellen nicht mehr deutlich besser werden.

In Abbildung 5.21 sind die Netze dargestellt, welche alle Gelenke als Eingabe verwenden, aber nur ein Gelenk vorhersagen. Auch hier lässt sich aufgrund der geringen Menge und der zufälligen Initialisierung keine eindeutige Tendenz erkennen. Es wird aber angenommen, dass eine deutlich größere Anzahl an Parametern keine wesentliche Verbesserung liefert. Wenn man den MAE mit dem aus Abbildung 5.20 vergleicht, ist zu sehen, dass Modelle mit allen Gelenken als Eingabe im Vergleich deutlich bessere Ergebnisse liefern. So ist der Maximalwert nur minimal größer als der Minimalwert für ein Gelenk. Dies lässt darauf schließen, dass der Kontext für die Modelle relevant ist und aus den anderen Gelenken Informationen gewonnen werden, welche die Vorhersage verbessern.

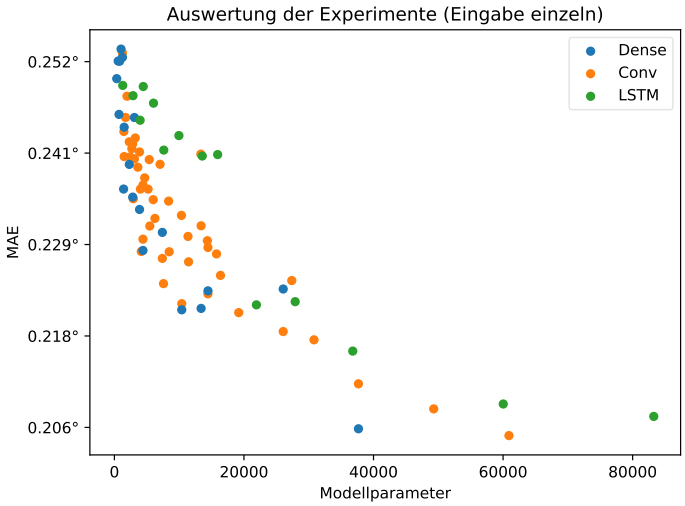


Abbildung 5.20: Visualisierung des Durchschnitts-MAE für einen Großteil der Experimente, bei denen ein Netz pro Gelenk verwendet wird, wobei nur dieses Gelenk als Eingabe dient. Dabei wird zwischen den verschiedenen Architekturen unterschieden.

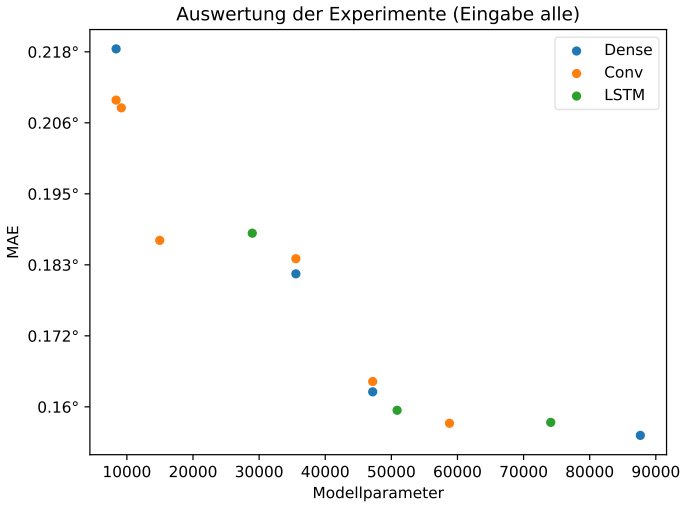


Abbildung 5.21: Visualisierung des Durchschnitts-MAE für Experimente, bei denen ein Netz pro Gelenk verwendet wird, wobei alle Gelenke als Eingabe dienen. Dabei wird zwischen den verschiedenen Architekturen unterschieden.

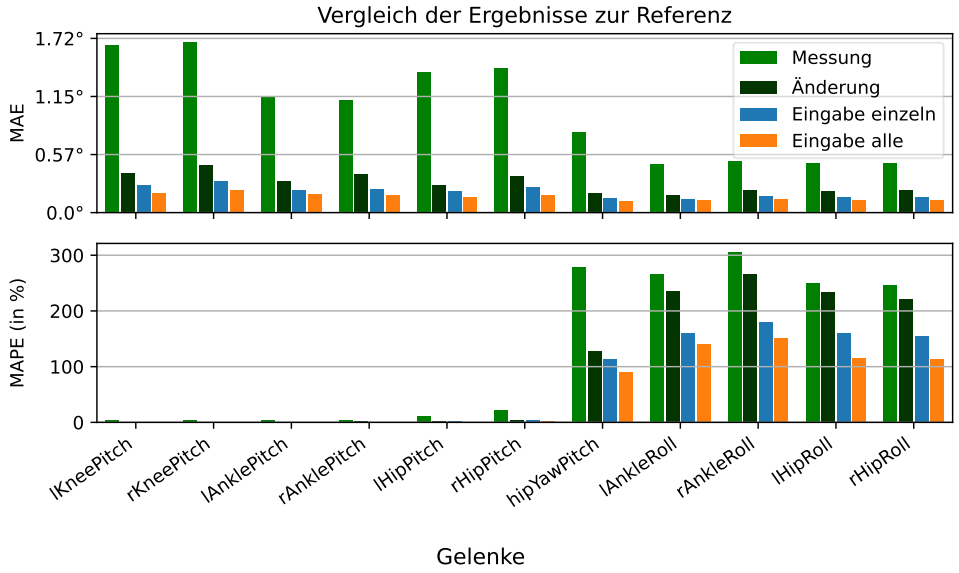


Abbildung 5.22: Gegenüberstellung von zwei Referenzimplementierungen (siehe Abschnitt 5.2) zu den besten Netzen für beide Varianten.

Vergleich zur Referenzimplementierung

Um auch hier einen Überblick zu bekommen, wie die Experimente im Vergleich zur Referenzimplementierung abschneiden, werden diese im Folgenden wieder miteinander verglichen. Dazu sind in Abbildung 5.22 der aktuelle Zustand (Messung), sowie die beste Referenzimplementierung (Änderung) zusammen mit den besten Netzen für die beiden Eingabevarianten dargestellt. Im Vergleich zum aktuellen Zustand haben die Netze dabei wieder einen um ein vielfaches besseren MAE. Auch im Vergleich zur besten Referenzimplementierung sind die Netze stets besser. Das Netz, welches alle Gelenke als Eingabe verwendet, ist für den MAE ungefähr doppelt so gut. Außerdem wird hier nochmal der bereits erwähnte Unterschied zwischen den beiden Varianten deutlich.

Vergleich zu einem einzelnen Modell

Um entscheiden zu können, welches der drei bisherigen Experimente die besten Ergebnisse liefert, werden diese im Folgenden miteinander verglichen. Dazu

sind in Abbildung 5.23 Beispiele dargestellt, bei denen jeweils die gleichen Modelle gruppiert sind. Hier ist zu sehen, dass die Vorhersagen, bei denen jedes Gelenk einzeln vorhergesagt und nur dieses Gelenk als Eingabe verwendet wird, schlechtere Ergebnisse liefern als die anderen Experimente. Allgemein lässt sich bei diesen bei steigender Modellkomplexität kaum eine Verbesserung erkennen. Modelle, die alle Gelenke als Eingabe nutzen und jeweils ein Gelenk vorhersagen, sind besser, als wenn alles in einem einzelnen Modell vorhergesagt wird. Dies war auch zu erwarten bei gleichen Modellen, bei denen einmal elf und einmal nur ein Gelenk optimiert werden müssen. Bei beiden ist außerdem jeweils eine Verbesserung pro Architektur bei steigender Modellkomplexität zu erkennen.

Ein Vergleich der Experimente, bei dem die gleichen Modelle verwendet werden, gibt einen Überblick, wie gut einzelne Modelle funktionieren. Allerdings lassen sich daraus keine Schlüsse ziehen, welche Art von Modellen am besten ist, da Modelle aus diesem Abschnitt elf mal vorhanden sind und somit eine deutliche größere Anzahl an Parametern aufweisen. Deshalb sind in Abbildung 5.24 Modelle gegenübergestellt, welche eine ähnliche Gesamtanzahl an Parametern aufweisen. Hier ist zu sehen, dass Modelle, welche jedes Gelenk einzeln vorhersagen und nur dieses Gelenk als Eingabe verwenden, schlechtere Ergebnisse liefern als die anderen Experimente. Insbesondere bei einer kleineren Anzahl an Parametern sind die Netze am besten, welche nur aus einem Modell bestehen. Bei einer größeren Anzahl an Parametern scheinen die Experimente, bei denen alle Gelenke als Eingabe und ein Gelenk als Ausgabe verwendet werden, ähnlich gute Ergebnisse zu liefern. In Abwägung mit dem Trainingsaufwand und dass kleinere Modelle bessere Ergebnisse liefern, sind die Modelle aus dem vorherigen Abschnitt, welche nur aus einem Modell bestehen, somit die bessere Wahl für eine spätere Anwendung auf dem Roboter.

Zusammenfassung

In diesem Abschnitt wurden zwei Varianten exploriert, welche ein Modell für jedes Gelenk trainieren und dabei entweder nur das Gelenk oder alle Gelenke als Eingabe verwenden. Hiermit sollte untersucht werden, wie sich die Vorhersagen

Abbildung 5.23: Auswahl an Trainings aus den drei bisherigen Experimenten, bei denen das gleiche Modell verwendet wird, wobei diese sich in der Anzahl der Ein- und Ausgaben unterscheiden. Weiterhin ist zu beachten, dass es bei den Varianten welche jedes Gelenk mit einem einzelnen Modell vorhersagen, entsprechend elf Netze gibt. Die Modellnamen bestehen aus dem Namen der Architektur und der Konfiguration des Modells. **l** steht für die Schichten, **u** für die Einheiten pro Schicht und **f** für die Conv-Filter, die sich für die verschiedenen Architekturen konfigurieren lassen.

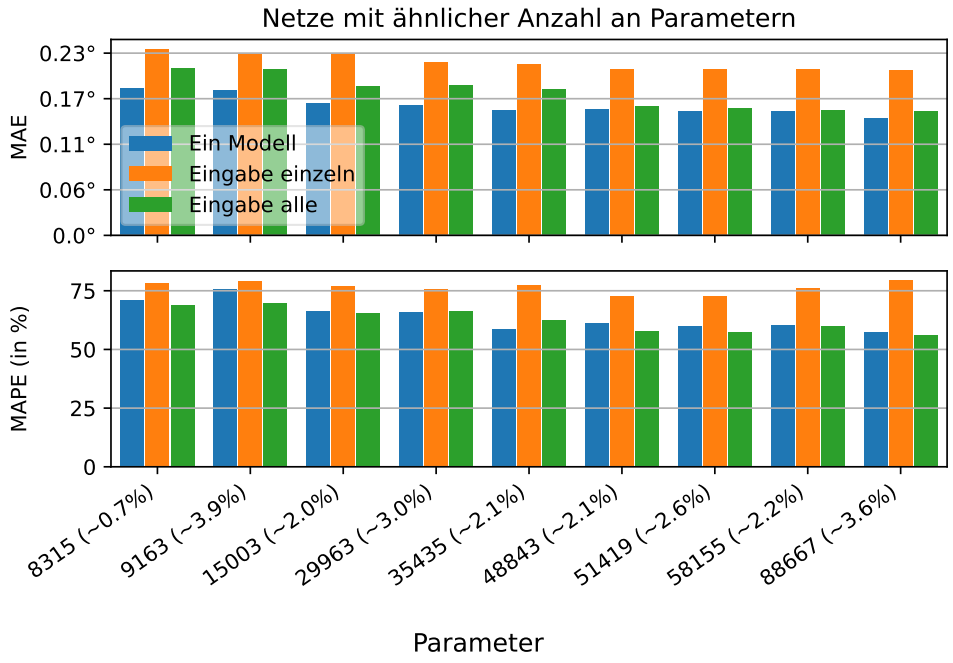


Abbildung 5.24: Auswahl an Trainings aus den drei bisherigen Experimenten, bei denen die Parameterzahl ungefähr gleich groß ist. In Klammern ist hier die prozentuale Abweichung der Parameteranzahl der Modelle aus diesem Abschnitt angegeben.

für einzelne Gelenke verhalten und ob dies bessere Ergebnisse liefert als die Netze aus dem vorherigen Abschnitt. Dabei wurden aufgrund der Trainingsdauer weniger Trainingsdurchläufe pro Modell vorgenommen, was zu einer möglicherweise größeren Streuung in den Ergebnissen führt. Die Ergebnisse wurden dabei wieder mit der Referenzimplementierung verglichen. Beide Varianten liefern für alle Gelenke deutlich bessere Ergebnisse als der aktuelle Zustand (Messung) und sind auch besser als die beste Referenzimplementierung (Änderung). Die Variante, welche alle Gelenke als Eingabe verwendet, liefert dabei die besseren Ergebnisse. Somit lässt sich schlussfolgern, dass der Kontext der anderen Gelenke relevant ist, um ein Gelenk vorherzusagen. Anschließend wurden die Varianten mit Modellen aus dem vorherigen Abschnitt verglichen. Dazu wurde sich zunächst angeschaut, wie sich gleiche Modelle verhalten, wobei ein Gelenk als Eingabe die schlechtesten Ergebnisse erzielt und alle Gelenke als Eingabe und ein Gelenk als Ausgabe, wie zu erwarten, die besten Ergebnisse erzielt. Abschließend wurden Modelle verglichen, die insgesamt eine ähnliche Anzahl an Parametern aufweisen. Hier zeigt sich deutlich, dass Modelle, welche ein Gelenk als Ein- und Ausgabe verwenden, sich nicht eignen. Für die anderen beiden Varianten ist es alleine aufgrund des MAE nicht ganz so eindeutig. Allerdings sind Modelle, die alle Gelenke auf einmal vorhersagen, bei einer kleineren Anzahl an Parametern besser. In zusätzlicher Abwägung mit dem Trainingsaufwand lässt sich folgern, dass es sinnvoller ist, Modelle wie aus dem vorherigen Abschnitt zu verwenden. Varianten aus diesem Abschnitt werden somit im Folgenden nur noch verwendet, um Unterschiede zwischen den Gelenken zu zeigen, sofern relevant.

5.6.3 Loss Funktionen

Wie in Abbildung 5.3 und vorherigem Abschnitt zu sehen, ist der MAPE für die Roll-Gelenke recht hoch. Aus diesem Grund soll untersucht werden, ob es sinnvoll ist, diesen als Loss zu verwenden. Dazu soll dieser sowohl für ein Modell als auch für Modelle, die alle Gelenkwinkel als Eingabe und ein Gelenk als Ausgabe verwenden, angewendet und mit dem bisher verwendeten MSE als Loss verglichen werden. Es wurden exemplarisch eine Reihe von Modellen

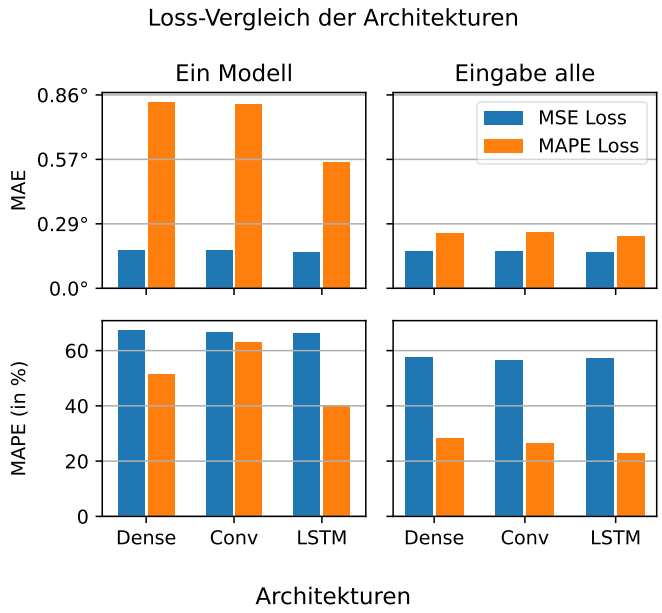


Abbildung 5.25: Beispielhafte Evaluation der Architekturen, wobei jeweils einmal MSE und MAPE als Loss verwendet wurden. Dargestellt ist jeweils ein repräsentatives Beispiel.

unterschiedlicher Größen trainiert, um einen Überblick zu bekommen, wie die Ergebnisse ausfallen.

In Abbildung 5.25 sind jeweils ein Beispiel jeder Architektur dargestellt. Dabei fällt zum einen auf, dass der MAPE-Loss viel bessere Ergebnisse liefert, wenn jedes Gelenk einzeln trainiert wird, als wenn es nur ein Modell gibt. Zum anderen fällt auf, dass der MAPE-Loss allgemein stets deutlich schlechtere Ergebnisse für den MAE liefert. Für den MAPE sind die Ergebnisse besser, allerdings für ein Modell nicht so deutlich, während er sich bei einem Modell pro Gelenk mehr als halbiert. Außerdem ist zu sehen, dass der MAPE-Loss, bei einem Modell, für die LSTM-Architektur bessere Ergebnisse liefert als die anderen beiden. Die Ursache konnte in weiteren Untersuchungen nicht ermittelt werden und ist somit möglicherweise auf die Architektur zurückzuführen. Dass ein Modell pro Gelenk deutlich bessere Ergebnisse liefert, ist gegensätzlich zum vorherigen Abschnitt, wobei der Unterschied hier um ein Vielfaches größer ist.

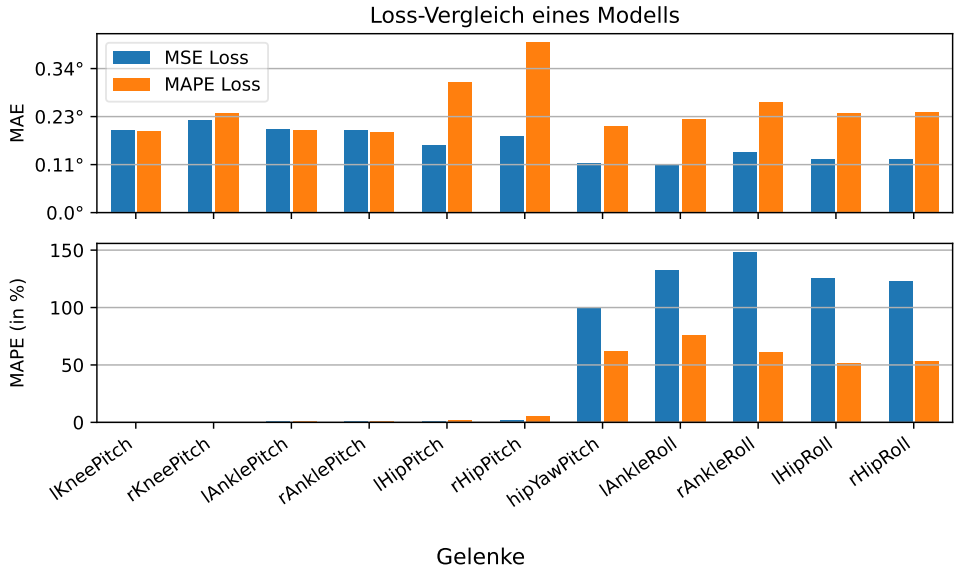


Abbildung 5.26: Beispielhafte Evaluation eines Modells, wobei jeweils einmal MSE und MAPE als Loss verwendet wurden.

Die Vermutung ist, dass eher für die Roll-Gelenke optimiert wird und somit die Gelenke, die größeren Einfluss auf den MAE haben, schlechter optimiert werden. Die tatsächliche Ursache ist allerdings unklar und die These könnte noch weiter untersucht werden, ist für die Arbeit aber nicht relevant, da die Ergebnisse allgemein schlechter sind.

In Abbildung 5.26 ist exemplarisch eines der Modelle dargestellt, welches mit beiden Loss-Funktionen trainiert wurde. Es ist zu sehen, dass die Ergebnisse sich für die KneePitch und AnklePitch-Gelenke kaum unterscheiden, während die HipPitch-Gelenke deutlich schlechtere Ergebnisse liefern. Die Ursache davon konnte in weiteren Untersuchungen nicht ermittelt werden. Für die Roll-Gelenke ist der MAPE zwar wie erwartet besser, allerdings ist der MAE ungefähr doppelt so groß. Zur weiteren Untersuchung wurden die Vorhersagen in Abbildung 5.28 direkt angeschaut. Dabei ist klar zu sehen, dass die Vorhersage des Modells, welches mit MAPE als Loss trainiert wurde, viel schlechtere Ergebnisse liefert. Auf den ersten Blick sieht es hier so aus, dass das Modell, welches mit MSE als Loss trainiert wurde, auch einen besseren MAPE haben

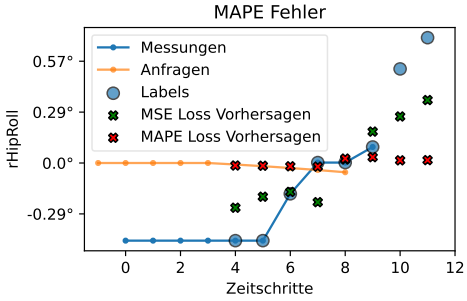


Abbildung 5.27: Ausschnitt aus Abbildung 5.28. Für Zeitschritt sieben ist der tatsächliche Wert ziemlich nah an null, während die Vorhersage deutlich abweicht, was zu einem sehr großen MAPE führt.

müsste. Bei genauerer Betrachtung (siehe Abbildung 5.27) fällt auf, dass es vereinzelte Vorhersagen gibt, bei denen der tatsächliche Wert sehr nah an null ist, während die Vorhersage deutlich abweicht, was zu einem sehr großen MAPE führt. So ist der Fehler für das Modell, welche mit MAPE als Loss trainiert wurde, bereits bei 955 % und für das Modell, welche MAPE als Loss trainiert wurde, sogar bei 9268 %. Durch diese Ausreißer wird die Metrik viel schlechter dargestellt, als sie für die meisten Vorhersagen ist. Somit wird hier deutlich, dass sich der MAPE insbesondere für die Roll-Gelenke nicht als Loss eignet.

Zusammenfassend wurde in diesem Abschnitt untersucht, ob sich der MAPE insbesondere für die Roll-Gelenke als Loss eignet. Dazu wurde eine Reihe von Modellen unterschiedlicher Größen trainiert und evaluiert. Dabei wurde festgestellt, dass dies für ein Modell, welches alle Gelenke vorhersagt überhaupt nicht funktioniert. Für ein Modell pro Gelenk ist zwar der MAPE besser, allerdings sind die Vorhersagen wie in Abbildung 5.28 dargestellt deutlich schlechter. Weiterhin wurde festgestellt, dass der MAPE für die Roll-Gelenke durch einzelne Ausreißer, bei denen das Label sehr nah an null ist und die Vorhersage um ein Vielfaches davon entfernt ist, verzerrt wird. Somit ist der MAPE für die Roll-Gelenke als Loss unbrauchbar und muss als Metrik mit Vorsicht betrachtet werden, da einzelne Werte nahe null einen großen Einfluss auf das Ergebnis haben.

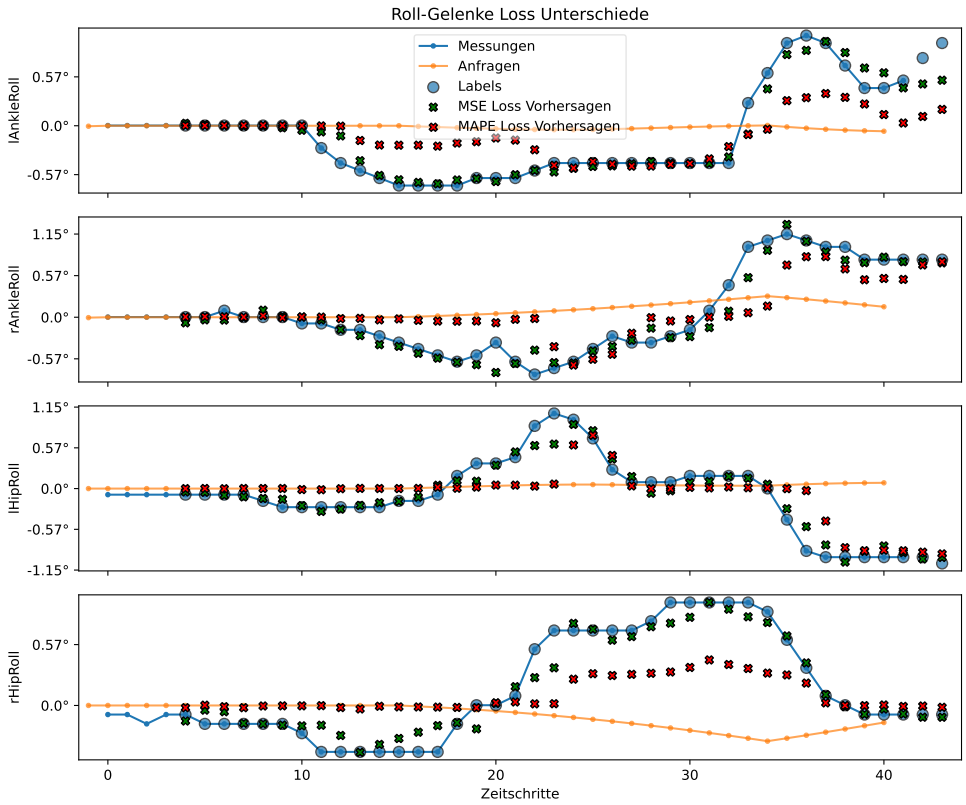


Abbildung 5.28: Beispiel Vorhersagen für die Roll-Gelenke aus Abbildung 5.26. Es sind die Anfragen, Messungen, Labels und die Vorhersagen der beiden Netze jeweils dargestellt. Jede Vorhersage bezieht sich dabei auf ein Fenster (siehe Abbildung 5.1) und es sind somit alle Fenster auf einmal dargestellt.

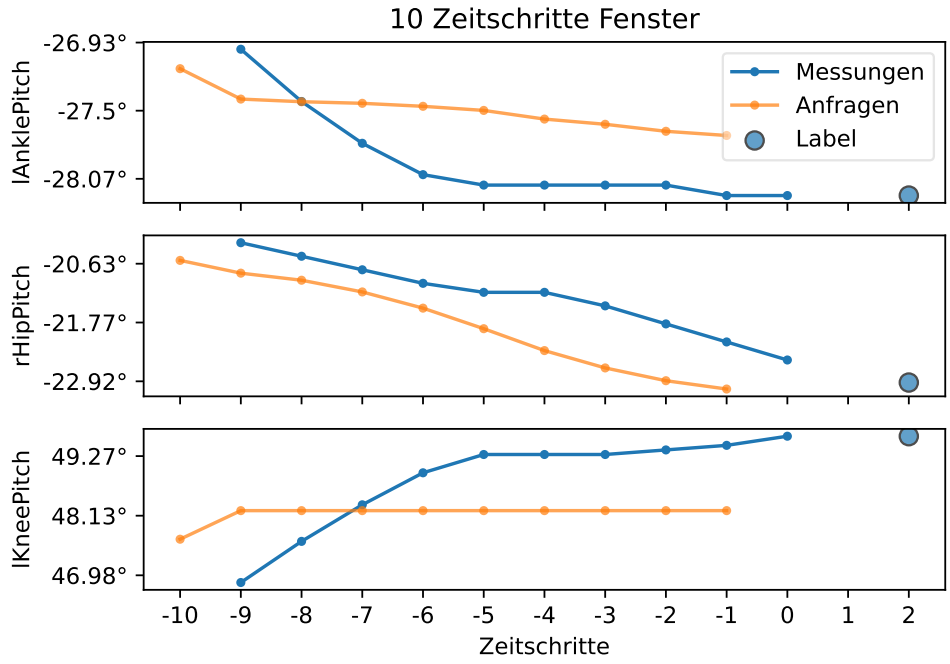


Abbildung 5.29: Beispiele für Eingaben eines Fenster, welches die letzten zehn Zeitschritte verwendet. Zeitschritt 0 beschreibt den aktuellen Zeitpunkt. Es sind die verwendeten letzten Anfragen und Messungen sowie die vorherzusagende Messung dargestellt.

5.6.4 Verschiedene Eingabegrößen

Wie in Abschnitt 5.1 beschrieben wurden bisher stets die letzten drei Messungen und Anfragen als Eingabe verwendet. Dies stellt aber nur eine Möglichkeit dar. Aus diesem Grund soll in diesem Abschnitt ausprobiert werden, mehr Zeitschritte als Eingabe zu verwenden. Die Erwartung ist, dass eine größere Eingabe bessere Ergebnisse liefert, da mehr zeitlicher Kontext zur Verfügung steht. Dazu sollen als Test sechs und zehn Zeitschritte als Eingabe verwendet werden. Ein resultierendes Fenster für zehn Zeitschritte ist beispielhaft in Abbildung 5.29 dargestellt. Es wurden exemplarisch eine Reihe von Modellen unterschiedlicher Größen trainiert, um einen Überblick zu bekommen, wie die Ergebnisse ausfallen.

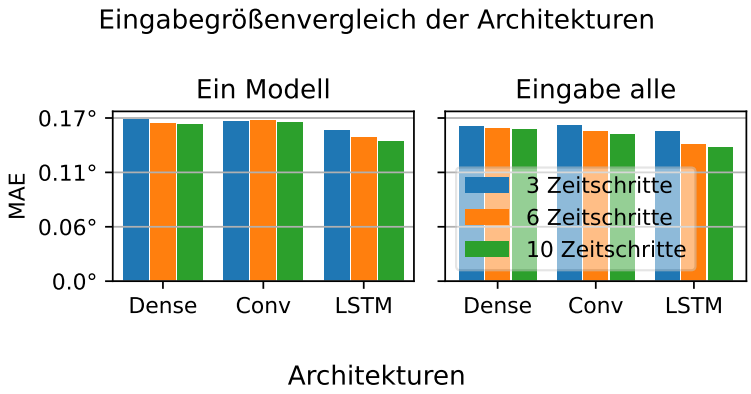


Abbildung 5.30: Beispielhafte Evaluation der Architekturen, wobei jeweils einmal drei, sechs und zehn Zeitschritte als Eingabe verwendet wurden. Dargestellt ist jeweils ein repräsentatives Beispiel.

In Abbildung 5.30 ist jeweils ein Beispiel jeder Architektur dargestellt. Dabei ist zu sehen, dass größere Eingaben im Schnitt gleichwertige oder leicht bessere Ergebnisse erzielen. So erzielen LSTM-Modelle bei einer größeren Menge an Zeitschritten stets bessere Ergebnisse. Bei den anderen Architekturen sind die Unterschiede nur minimal. Weiterhin ist zu beachten, dass die Anzahl der Parameter sich bei LSTM-Modellen bei mehr Zeitschritten nicht ändert. Bei den anderen Architekturen können sich diese je nach Modell mehr als verdoppeln. Somit sind LSTM-Modelle für eine größere Anzahl an Zeitschritten als Eingabe besser geeignet. In Abbildung 5.31 ist exemplarisch das LSTM-Modell dargestellt, welches mit unterschiedlichen Zeitschritten als Eingabe trainiert wurde. Dabei fallen unter Berücksichtigung der Zufallsinitialisierung keine weiteren Besonderheiten auf.

Zusammenfassend wurde in diesem Abschnitt untersucht, ob sich eine größere Menge an Zeitschritten besser für eine Vorhersage eignet. Dazu wurden eine Reihe von Modellen unterschiedlicher Größen trainiert und evaluiert. Es wurde festgestellt, dass sich ausschließlich die LSTM-Architektur eignet und diese auch bessere Ergebnisse liefert. Die anderen Architekturen liefern dagegen bei größerer Anzahl an Parametern nur minimal bessere Ergebnisse. Für eine optimale Anzahl an Zeitschritten sind weitere Untersuchungen notwendig.

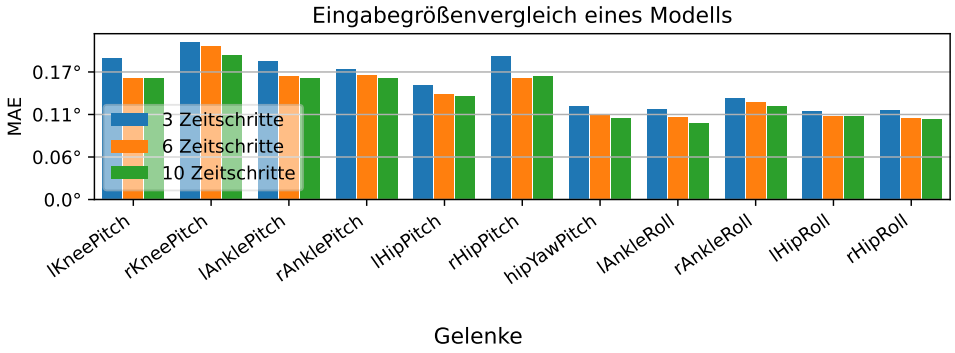


Abbildung 5.31: Beispielhafte Evaluation eines Modells, wobei jeweils einmal drei, sechs und zehn Zeitschritte als Eingabe verwendet wurden.

5.7 Zusammenfassung

In diesem Kapitel wurden die Referenzimplementierungen und verwendeten Architekturen vorgestellt. Gegen diese wurden dann in den folgenden Abschnitten diverse Experimente durchgeführt, um geeignete Modelle zum Vorhersagen von Gelenkwinkeln zu finden. Dabei wurden die verschiedenen Architekturen und Ansätze verwendet und alle mit einem kleinen Datensatz und Testset entsprechend trainiert und evaluiert.

Gestartet wurde mit Modellen, welche alle Gelenke auf einmal vorhersagen. Dabei hat sich herausgestellt, dass solche Modelle bessere Ergebnisse erzielen, welche die Änderung der Gelenke lernen, anstatt die Winkel selbst. Beim Vergleich mit der Referenzimplementierung war zu sehen, dass bereits Netze mit etwas über 2000 Parametern bessere Ergebnisse liefern als die beste Referenz und neuronale Netze allgemein deutlich bessere Ergebnisse liefern als der aktuelle Zustand (Messung). Dabei liefern Modelle im kleinen Parameterbereich schnell bessere Ergebnisse. Dies flacht bei 20 000 bis 50 000 Parametern ab. Sollte es Probleme bei der Laufzeit auf dem NAO[6] geben, ist in diesem Bereich vermutlich eine Abwägung zwischen Laufzeit und Fehler zu treffen. Zwischen den vorgestellten Architekturen konnte allerdings kein klares Ergebnis gefunden werden, weshalb weiterhin alle benutzt wurden.

Anschließend wurden zwei Varianten untersucht, welche ein Modell für jedes Gelenk trainieren und dabei entweder nur das Gelenk oder alle Gelenke als Eingabe verwenden. Auch hier hat sich gezeigt, dass beide Varianten bessere Ergebnisse liefern, als die Referenzimplementierung. Alle Gelenke als Eingabe hat sich dabei als die bessere Variante herausgestellt, woraus sich schlussfolgern lässt, dass der Kontext der anderen Gelenke benötigt wird, um diese präzise vorherzusagen. Weiterhin wurden diese mit dem vorherigen Abschnitt verglichen, wobei zunächst die gleichen Modelle miteinander verglichen wurden. Dabei ist nochmal klar zu sehen, dass ein Gelenk als Eingabe, die schlechtesten, und Modelle, die alle Gelenke als Eingabe und ein Gelenk als Ausgabe haben, die besten Ergebnisse liefern. Vergleicht man allerdings Modelle mit ähnlicher Anzahl an Parametern, wird klar, dass alle Gelenke in einem Modell vorhersagen, die besten Ergebnisse liefert. Somit ist dieses die präferierte Wahl.

Da in diversen Abbildungen zu sehen ist, dass der MAPE für die Roll-Gelenke viel schlechtere Ergebnisse liefert, wurde dieser als Loss ausprobiert. Dabei hat sich herausgestellt, dass dies insgesamt deutlich schlechtere Ergebnisse liefert, da diese Netze insbesondere für die Roll-Gelenke lernen, möglichst keine Ausreißer zu erzeugen und weniger das Verhalten der Gelenke. Das Problem hierbei ist, dass sich diese Gelenke im Durchschnitt einen Winkel nahe null haben. Aus diesem Grund haben kleiner Fehler sowieso einen erwarteten größeren prozentualen Einfluss. Allerdings gibt es einige Fälle, in denen das Label sehr nah an null ist und die Vorhersage um ein Vielfaches davon entfernt ist, was zu einer Verzerrung der Metrik führt. Somit wird weiterhin MSE als Loss verwendet und MAPE als Metrik mit Vorsicht betrachtet.

Bisher wurde stets mit einer gewählten Eingabe von drei Zeitschritten gearbeitet. Deshalb wurden sechs und zehn Zeitschritte als weitere Möglichkeiten ausprobiert. Dabei wurde festgestellt, dass sich ausschließlich die LSTM-Architektur dafür eignet, da sie bei einer größeren Eingabe bessere Ergebnisse liefert und sich die Anzahl der Parameter nicht verändert. Um eine optimale Anzahl an Zeitschritten zu finden, ist an dieser Stelle noch eine weitere Untersuchung notwendig.

In Abbildung 5.32 sind die besten Ergebnisse aus den Experimenten dargestellt. Das insgesamt beste Netz ist dabei ein einzelnes LSTM-Modell mit zehn

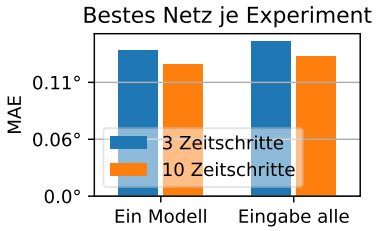


Abbildung 5.32: Ergebnisse der besten Netze für drei und zehn Zeitschritte als Eingabe für ein Modell und ein Modell pro Gelenk mit allen Gelenken als Eingabe.

Zeitschritten als Eingabe und vier extra Schichten mit jeweils 48 Einheiten. Es ist mit einem MAE von 0,13° bereits recht nah an der Genauigkeit der Sensoren, welche bei ungefähr 0,1° liegt (SoftBank Robotics 2023b). Zur Einordnung ist nochmals zu beachten, dass bisher nur einige Daten, von ungefähr 100 min zum Training und 15 min zum Evaluieren, von einem Wettbewerb aus 2019 verwendet wurden. Dabei waren die Roboter weitestgehend neuwertig und somit spielen Effekte, die durch Verschleiß auftreten, noch kaum eine Rolle.

Außerdem ist in Abbildung 5.33 eine Reihe beispielhafter Vorhersagen für einige Gelenke des besten einzelnen Modells mit drei Zeitschritten als Eingabe dargestellt. Jede Vorhersage wurde dabei unabhängig aus dem jeweiligen Fenster getroffen. Hier ist gut zu sehen, dass die Pitch-Gelenke größere Winkeldifferenzen überwinden, während diese bei den Roll-Gelenken recht klein sind. Weiterhin ist für die Roll-Gelenke zu sehen, dass sich die Gelenke bewegen, obwohl eine quasi statische Position angefragt wird. Das Netz ist in der Lage, auch dies einigermaßen korrekt vorherzusagen.

Abschließend lässt sich sagen, dass neuronale Netze deutlich besser Ergebnisse liefern als der aktuelle Zustand (Messung) und die Referenzimplementierungen. Es wurden Modelle gefunden, welche auf neuwertigen Robotern sehr gute Ergebnisse erzielen und verschiedene Grundlagen geschaffen, auf denen im nächsten Abschnitt weitere Fragen dieser Arbeit beantwortet werden können und die als Orientierung für eine Anwendung dienen.

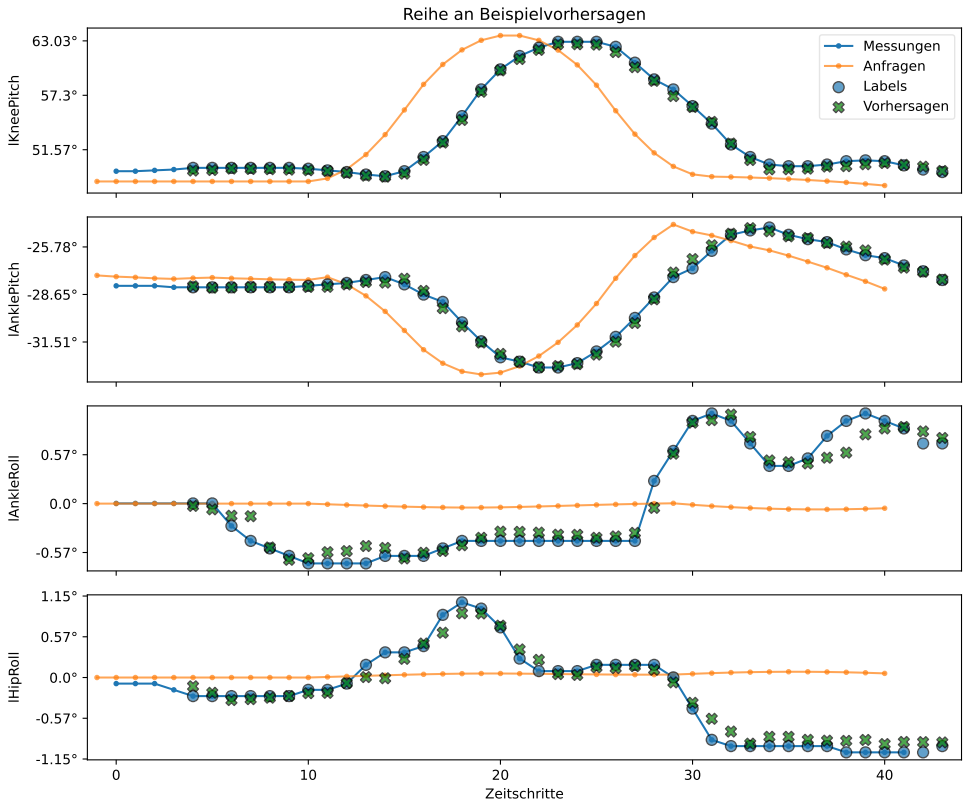


Abbildung 5.33: Beispielvorhersagen für die linken Gelenke, des besten Netzes mit drei Zeitschritten als Eingabe. Es sind die Anfragen, Messungen, Labels und die Vorhersagen des Netzes dargestellt. Jede Vorhersage bezieht sich dabei auf ein Fenster (siehe Abbildung 5.1) und es sind somit alle Fenster auf einmal dargestellt.

5.8 Kreuzvalidierung

In diesem Abschnitt wird vorgestellt, wie gut der Ansatz generalisiert und welche Schlüsse daraus gezogen werden können. Dazu wurden verschiedene Kreuzvalidierungen (Mosier 1951) durchgeführt. Diese eigenen sich auch für Zeitreihen, wobei es verschiedene Ansätze gibt, um diese anzuwenden (Bergmeir und Benítez 2012). In dieser Arbeit wird eine angepasste Version verwendet. Dies hat zum einen damit zu tun, dass es sich bei Zeitreihen nicht um voneinander unabhängige Daten handelt. Zum anderen, sollen hier bestimmte Eigenschaften in den Daten gezeigt und zwischen Daten verglichen werden. Aus diesem Grund, werden jeweils Daten zu Gruppen zusammengefasst, welche teilweise auch stark unterschiedlich groß sind. Die Gruppen werden dann jeweils einmal für die Evaluation verwendet, während die anderen Gruppen als Trainingsdaten dienen. Bei den Abbildungen ist jeweils die Gruppe angegeben, die zur Evaluation verwendet wurde. Dabei sind sowohl die Trainings-Validierung am Ende des Trainings auf den Validierungsdaten als auch die Ergebnisse der Kreuzvalidierung mit den angegebenen Daten dargestellt.

Dazu werden in Abschnitt 5.8.1 verschiedene Roboter auf dem gleichen Feld miteinander verglichen und anschließend in Abschnitt 5.8.2 untersucht, wie sich einzelne Roboter auf unterschiedlichen Feldern verhalten. Abschließend wird in Abschnitt 5.8.3 untersucht, welchen Einfluss das Alter der Roboter auf die Vorhersage hat und jeweils entsprechende Schlüsse gezogen.

Bisher wurde stets nur ein kleiner Teil der Daten verwendet. Im Folgenden wird jetzt teilweise ein Großteil des Datensatzes verwendet, um unterschiedliche Fragen beantworten zu können und die Generalisierung zu klären. Dies führt zu anderen Ergebnissen und ist somit nicht mit den vorherigen Abschnitten vergleichbar, in denen stets die gleichen Daten verwendet wurden. Für diese wurde in Abbildung 5.34 eine Kreuzvalidierung durchgeführt. Dabei wurden jeweils die Daten eines Spiels als Testdaten verwendet und die restlichen Daten fürs Training. Wie zu sehen ist, unterscheiden sich weder die Validierungsdaten vom Training, noch die der Kreuzvalidierung deutlich. Es ist auch keine Tendenz in den späteren Spielen zu erkennen. Somit kann möglicherweise auch auf einzelne Spiele eines Roboters auf einem Feld verzichtet werden, ohne das

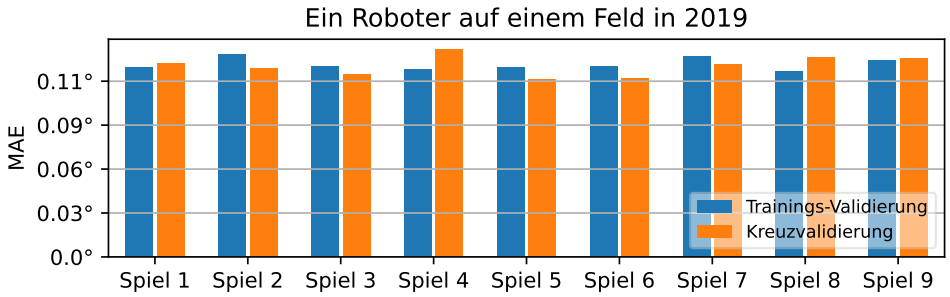


Abbildung 5.34: Kreuzvalidierung, bei der unterschiedliche Spiele eines Roboters auf einem Feld in 2019 jeweils gegenübergestellt sind. Die Spiele sind dabei in der Reihenfolge sortiert, in der sie gespielt wurden.

Ergebnis negativ zu beeinflussen. Für eine allgemeingültige Aussage müsste dies allerdings für mehrere Roboter und Felder gezeigt werden.

5.8.1 Übertragbarkeit auf unbekannte Roboter

Um zu untersuchen, wie unterschiedlich sich einzelne Roboter verhalten und wie gut die Übertragbarkeit auf unbekannte Roboter funktioniert, wurden verschiedene Kreuzvalidierungen trainiert. Somit kann ermittelt werden, ob ein neuronales Netz für alle Roboter verwendet werden kann oder ob neuronale Netze für unterschiedliche Roboter nach trainiert werden sollten.

In Abbildung 5.35 ist dargestellt, wie gut ein neuronales Netz funktioniert, wenn einmal genau auf dem Roboter trainiert (33 min Spielzeit) wird, auf dem evaluiert (14 min Spielzeit) wird und einmal auf allen anderen Robotern auf dem gleichen Feld (ungefähr 200 min Spielzeit). Dabei ist zu sehen, dass die Ergebnisse besser sind, wenn für den gleichen Roboter trainiert und evaluiert wird. Allerdings sind die Ergebnisse nicht wesentlich schlechter, wenn nur auf den anderen Robotern trainiert wird. Für neuere Roboter sind die Unterschiede wie für 2019 zu sehen nur sehr klein. Insbesondere sind die Ergebnisse auf den Testdaten kaum unterschiedlich zur Trainings-Validierung, was dafür spricht, dass das neuronale Netz gute Ergebnisse für den unbekannten Roboter liefert.

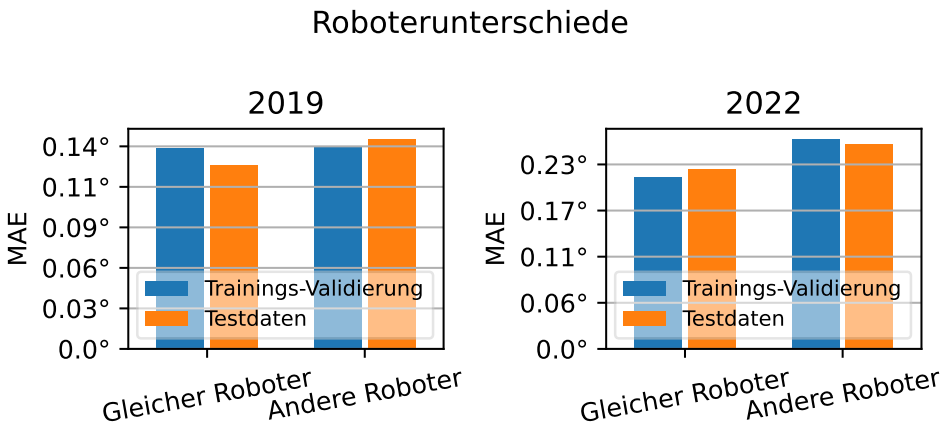
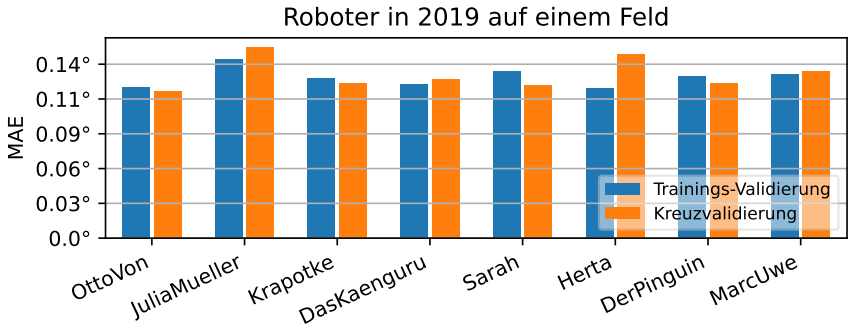


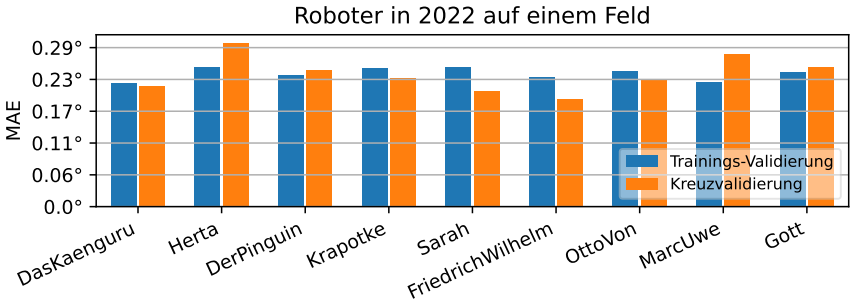
Abbildung 5.35: Es wurde jeweils einmal auf dem gleichen Roboter trainiert, auf dem auch evaluiert wird und einmal auf allen anderen Robotern von einem Feld, wobei jeweils die gleichen Daten als Testdaten verwendet wurden.

Um einen allgemeineren Überblick zu bekommen, sind in Abbildung 5.36a Roboter aus 2019 von einem Feld gegenübergestellt. Diese sind von links nach rechts in aufsteigender Spielzeit sortiert. Die genaue Zeit jeweils kann aus Tabelle 3.4 in der Spalte *GO* entnommen werden. Insgesamt handelt es sich dabei um ungefähr 525 min Spielzeit. Wie zu sehen ist, gibt es leichte Abweichungen sowohl bei der Trainings-Validierung, als auch bei den Ergebnissen der Kreuzvalidierung. Dabei fallen *JuliaMueller* und *Herta* auf, welche einen etwas größeren Fehler haben, wobei erstere auch schon eine etwas schlechtere Trainings-Validierung aufweist. Insgesamt liegen die Fehler aber nahe beieinander, was darauf hinweist, dass sich neuronale Netze auch gut für ungesehene Roboter eignen, da es nur geringe Unterschiede zwischen den Robotern gibt.

In Abbildung 5.36b ist das ganze für 2022 dargestellt. Hierbei handelt es sich um ungefähr 290 min Spielzeit, welche in Tabelle 3.5 in der Spalte *RC A* dargestellt ist. Während die Trainings-Validierungen einigermaßen gleichmäßig sind, gibt es bei der Kreuzvalidierung teilweise etwas größere Unterschiede. Hier fällt wieder *Herta* auf, welche sich somit möglicherweise allgemein etwas anders verhält als die anderen Roboter. Da die Differenz innerhalb der Kreuzvalidierung



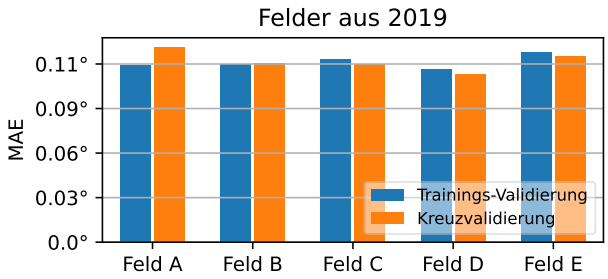
(a) Roboter 2019 auf einem Feld.



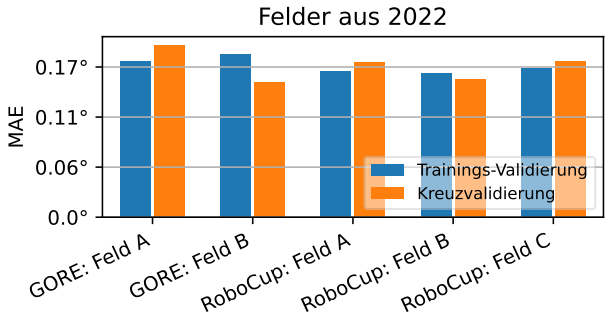
(b) Roboter 2022 auf einem Feld.

Abbildung 5.36: Kreuzvalidierungen, bei der Roboter jeweils auf einem Feld gegenübergestellt sind. Die Roboter sind dabei von links nach rechts in aufsteigender Spielzeit sortiert.

ohne *Herta* mit 0,08° unter der Genauigkeit der Sensoren mit 0,1° (SoftBank Robotics 2023b) liegt und die meisten Ergebnisse relativ ähnlich sind, wird angenommen, dass ein neuronales Netz grundsätzlich auch auf älteren Robotern weiterhin für alle Roboter funktioniert. Wenn man die beiden Abbildungen miteinander vergleicht, fällt auf, dass der MAE auf älteren Robotern ungefähr doppelt so groß ist. Somit lässt sich darauf schließen, dass sich ein neuronales Netz auf dem gleichen Feld auf Roboter des gleichen Alters gut übertragen lässt, ohne dieses nach trainieren zu müssen. Für Erkenntnisse, wie unterschiedliche Felder sich verhalten und welche Auswirkungen das Alter der Roboter hat, werden im Folgenden weitere Untersuchungen durchgeführt.



(a) Ein Roboter aus 2019.



(b) Ein Roboter aus 2022.

Abbildung 5.37: Kreuzvalidierungen, bei denen einzelne Roboter auf verschiedenen Feldern gegenübergestellt sind. Es ist jeweils angegeben, was zur Evaluation verwendet wurde.

5.8.2 Übertragbarkeit auf unbekannte Felder

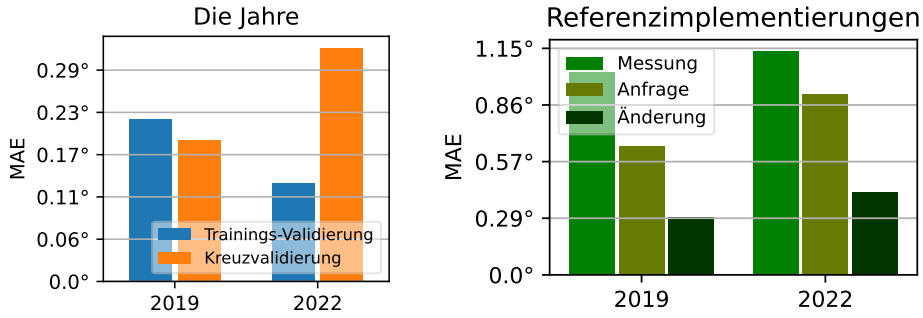
Im Folgenden wird die Übertragbarkeit auf unterschiedliche Felder angeschaut. Die Felder sind dabei grundsätzlich ähnlich beschaffen, allerdings ist unbekannt, wie sich die Unterschiede auf das Laufen auswirken. Dazu wird jeweils ein Roboter verwendet, der auf mehreren Feldern gespielt hat. Wie in Abbildung 5.37 zu sehen, gibt es dabei nur geringfügige Unterschiede. Insgesamt werden dabei ungefähr 100 min und 160 min Spielzeit betrachtet. Somit kann das gleiche neuronale Netz mindestens für Felder verwendet werden, die ähnlich zu denen sind, die auf RoboCups bisher verwendet werden, da kleinere Unterschiede kaum Auswirkungen auf das Laufen und entsprechend die Vorhersage haben.

5.8.3 Anwendung auf Robotern unterschiedlichen Alters

Nachdem bisher einmal mehrere Roboter auf gleichen Feldern und einmal einzelne Roboter auf unterschiedlichen Feldern betrachtet wurden, wird sich dies jetzt in einem größeren Kontext angeschaut. Dabei geht es zum einen darum, ob unterschiedliche neuronale Netze verwendet werden sollten für Roboter unterschiedlichen Alters und zum anderen, wie gut die Ergebnisse allgemein sind, wenn auf größeren Datenmengen trainiert wird. Es werden jeweils zufällig 70 % der vorhandenen Daten jeder Gruppe verwendet, was ungefähr 1700 min Spielzeit entspricht, da die Trainigshardware (siehe Abschnitt 5.3) mit der aktuellen Implementierung nicht mehr zulässt, weil der Arbeitsspeicher ausgelastet wird.

In Abbildung 5.38a wurde jeweils mit den Daten des einen Jahres trainiert und mit denen des anderen Jahres evaluiert. Dabei ist, wie bereits in den vorherigen Abbildungen zu beobachten, bei der Trainings-Validierung zu sehen, dass die Daten aus 2019 einen deutlich kleineren Fehler erreichen als die Daten aus 2022. Umgekehrt ist bei der Kreuzvalidierung zu sehen, dass die Daten aus 2022, die Daten für 2019 gut vorhersagen können, während es umgekehrt nicht gut funktioniert. Hieraus lässt sich folgern, dass Daten von älteren Robotern zum Training verwendet werden können und das resultierende neuronale Netz auch für neuere Roboter funktioniert, wenn auch schlechter, als wenn nur neuere Roboter verwendet würden. Umgekehrt funktioniert dies allerdings nicht so gut. Mögliche Änderungen, welche durch Softwareänderungen zustande kommen, werden hierbei erst mal außenvorgelassen.

Wenn man dies mit der Evaluation der Referenzimplementierungen auf den gleichen Daten in Abbildung 5.38b vergleicht, ist zu sehen, dass die Ergebnisse trotzdem viel besser sind als der aktuelle Zustand (Messung). Die beste Referenzimplementierung (Änderung) ist auf den Daten aus 2019 0,1° schlechter, als wenn auf den Daten aus 2022 trainiert wird. Auf den Daten aus 2022 hat sie mit 0,42° im Vergleich zur 2022 Trainings-Validierung (2019 in Abbildung 5.38a) und auch zur Kreuzvaliduerung einen etwa doppelt so großen Fehler. Somit ist ein neuronales Netz insbesondere für ältere Roboter deutlich besser als die Referenzimplementierungen.



(a) Kreuzvalidierung, bei der die beiden Jahre miteinander verglichen werden.

(b) Evaluation der Referenzimplementierungen auf den beiden Jahren.

Abbildung 5.38: Evaluation der Jahre, welche neue und ältere Roboter repräsentieren. Es ist jeweils das Jahr auf dem evaluiert wurde angegeben.

In Abbildung 5.39 sind die vier Wettbewerbe gegenübergestellt, wobei jeweils auf den anderen dreien trainiert wurde. Während sich die Wettbewerbe in 2022 annähernd gleich verhalten, sind für 2019 deutliche Unterschiede zu erkennen. Die genaue Ursache dafür konnte nicht ermittelt werden, allerdings ist eine Vermutung, dass es möglicherweise mit unterschiedlichen Laufeinstellungen zu tun hat. Für die 2019 GermanOpen wurde eine recht aggressive Einstellung gewählt, während auf dem RoboCup vorsichtiger gelaufen wurde. Dass beim Training Daten aus beiden Jahren verwendet werden, gleicht die Ergebnisse der Trainings-Validierung aneinander an, auch wenn der 2019 RoboCup etwas heraussticht. Außerdem ist im Vergleich mit Abbildung 5.38a zu erkennen, dass einige Daten auf älteren Robotern den Fehler eines neuronalen Netzes, welches überwiegend auf neuen Robotern trainiert wurde, bereits deutlich reduzieren. Ob umgekehrt einige Daten von neuen Robotern das Ergebnis für neue Roboter verbessern ist aufgrund der Unterschiede nicht klar ersichtlich, allerdings wird das Ergebnis auch nicht schlechter. Somit scheint es sinnvoll, Daten von neueren und älteren Robotern zum Training zu verwenden, wobei die Daten von älteren Robotern deutlich wichtiger sind, da die Ergebnisse auf auch neuen Robotern gut funktionieren.

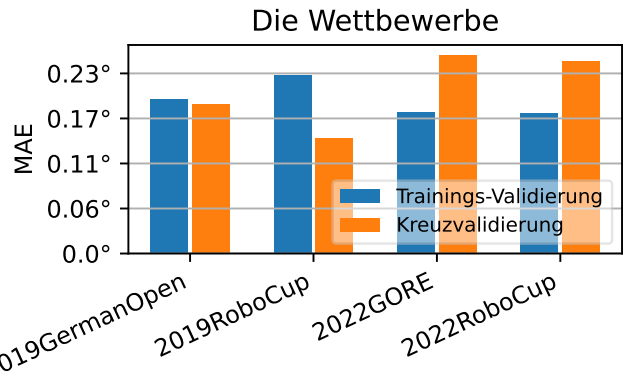


Abbildung 5.39: Kreuzvalidierung, bei der die einzelnen Wettbewerbe gegenübergestellt sind.

5.8.4 Zusammenfassung

In diesem Abschnitt wurden unterschiedliche Kreuzvalidierungen auf den Daten vorgestellt, um herauszufinden, ob ein neuronales Netz für beliebige NAO[6] mit gleichem Laufen verwendet werden kann, oder ob unterschiedliche nötig sind. Dazu wurde dies zunächst für unterschiedliche Roboter auf einem Feld und anschließend für die gleichen Roboter auf unterschiedlichen Feldern betrachtet. Die Ergebnisse sind dabei, dass ein neuronales Netz für das gleiche Feld meistens auch auf unbekannten Robotern gute Ergebnisse liefert, es allerdings Ausnahmen zu geben scheint, in denen die Ergebnisse etwas schlechter sind. Hier konnte nicht ermittelt werden, ob dies an anders verbauter Hardware, der Roboter kaputt war oder einem anderen unbekannten Grund liegt. Auf unterschiedlichen Feldern gab es kaum Abweichungen, womit dies nicht relevant zu sein scheint.

Anschließend wurden alte und neue Roboter miteinander verglichen. Dabei liefert das Training auf neuen Robotern, wie erwartet, deutlich bessere Ergebnisse, als wenn nur alte Roboter verwendet werden. Allerdings liefert ein neuronales Netz, welches auf älteren Robotern trainiert wurde, auch gute Ergebnisse für neuere Roboter, während dies umgekehrt nicht der Fall ist. Die Ergebnisse sind allerdings schlechter, als wenn nur neue Daten verwen-

det würden. Es ist somit besser, Daten von älteren Robotern zum Training zu verwenden. Es konnte nicht nachgewiesen werden, dass zusätzliche Daten von neueren Robotern das Ergebnis verbessern, sie haben aber auch keinen negativen Einfluss. Dabei ist zu beachten, dass der Einfluss der Software auf das Laufen und somit die Ergebnisse unbekannt sind. Es wird aber davon ausgegangen, dass der Verschleiß der Roboter einen größeren Einfluss als die Softwareänderungen hat.

Abschließend lässt sich somit festhalten, dass ein neuronales Netz für alle Roboter verwendet werden kann und auch nicht alle Roboter im Training vorhanden sein müssen. Dabei eignen sich Daten von älteren Robotern und in diesem Fall neuere Daten besser zum Training als umgekehrt.

6 Anwendung der Erkenntnisse

Bisher wurden die Zielplattform und die Anwendung auf dieser nicht explizit betrachtet. Nachdem im vorherigen Kapitel vielversprechende Experimente durchgeführt wurden, werden in diesem Kapitel die Anwendung auf dem NAO[6] und die Verwendung der Ergebnisse in der B-Human-Software beschrieben.

Dazu wird in Abschnitt 6.1 zunächst beschrieben, wie neuronale Netze in der B-Human-Software überhaupt auf dem Roboter ausgeführt werden können, bevor in Abschnitt 6.2 darauf aufbauend die Implementierung für diese Arbeit und deren Ergebnisse vorgestellt werden. Abschließend sind in Abschnitt 6.3 die Anwendung der Ergebnisse dieser Arbeit beschrieben.

6.1 Inferenz auf dem NAO[6]

Um bei B-Human neuronale Netze auf dem Roboter ausführen zu können, wurde von Thielke und Hasselbring (2019) mit *CompiledNN* eine eigene Laufzeitumgebung entwickelt, da bestehende Bibliotheken primär für GPUs optimiert sind, hier aber ausschließlich eine CPU vorhanden ist. Diese übersetzt neuronale Netze zur Laufzeit direkt in optimierten Maschinencode, im Gegensatz zu vielen anderen Ansätzen, die das Netz interpretieren. Dadurch wird eine schnelle Ausführung ermöglicht, die auf kleinen Modellen schneller als andere Implementierungen ist. Die Umgebung ist dabei hauptsächlich auf Bildverarbeitung ausgelegt und wurde zunächst für die Ball- (Röfer, Laue, Felsch u. a. 2019) und Robotererkennung (Poppinga und Laue 2019) eingesetzt. Mittlerweile gibt es diverse weitere Erkennungen, für die diese genutzt wird (Hasselbring und Baude 2022; Röfer, Laue, Ewers u. a. 2022). Die Ballerkennung beispielsweise hat dabei ungefähr 22 000 Parameter bei einer Laufzeit von knapp 200 µs.

Seit 2022 ist zusätzlich eine Inferenz mit der *ONNX Runtime* (ONNX

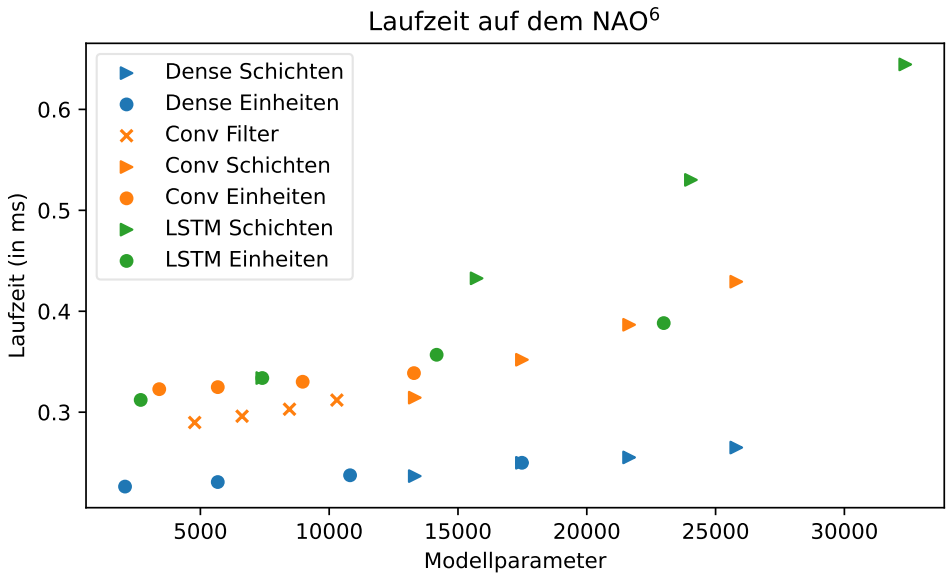


Abbildung 6.1: Visualisierung der Laufzeitunterschiede auf dem NAO[6] für die unterschiedlichen Parameter. Die Parameter wurden jeweils linear erhöht.

Runtime developers 2021) möglich, da diese für ein komplexes Modell benötigt wurde, welches CompiledNN nicht ausführen kann (Röfer, Laue, Ewers u. a. 2022, S. 39). Dies stellte im Laufe der Arbeit somit eine weitere Möglichkeit dar, Modelle auf dem Roboter auszuführen.

Für diese Arbeit wird auch die ONNX Runtime verwendet, da zum Beispiel LSTMs und benutzerdefinierte Schichten aktuell nicht von CompiledNN unterstützt werden. Da bisher die Laufzeit nicht betrachtet wurde und um einen Überblick zu bekommen, wie die Laufzeit der unterschiedlichen Architekturen und deren Parameter (siehe Abschnitt 5.5) auf dem NAO[6] ist, ist dies in Abbildung 6.1 dargestellt. Es wurden jeweils kleinere Modelle gewählt, bei denen ein Parameter linear erhöht wird. Dabei fällt auf, dass Dense-Modelle deutlich schneller sind bei ähnlicher Anzahl an Parametern. LSTM-Modelle benötigen dagegen vergleichsweise lange, insbesondere mehrere Schichten erhöhen die Laufzeit merklich.

6.2 Implementierung

Um diese Arbeit auf dem Roboter ausführen zu können, wurde eine Komponente zur B-Human-Software hinzugefügt, welche die Vorhersage bereitstellt, die dann von anderen Komponenten verwendet werden kann. Diese merkt sich die letzten angefragten Gelenkwinkel und Messungen und hält eine Instanz eines Netzes vor. Dieses wird dann, sofern Daten zur Verfügung stehen, jeden Zeitschritt ausgeführt und die Ergebnisse bereitgestellt.

Um sicherzustellen, dass die Implementierung korrekt ist, wurde ein Roboter laufen gelassen und dieser hat die bisher verwendeten Daten und zusätzlich die Vorhersage eines Netzes aufgenommen. Anschließend wurde auf den Daten das Netz in der Entwicklungsumgebung ausgeführt und die Vorhersage mit der vom Roboter verglichen. Diese waren bis auf bekannte Fließkomma-Ungenauigkeiten identisch, womit die Implementierung korrekt ist.

Bisher wurden keine Netze speziell für den tatsächlichen Einsatz bei B-Human trainiert. Um einen genauen Überblick über die Laufzeiten bestimmter Modelle zu bekommen, wurden in Abbildung 6.2 verschiedene Modelle auf dem NAO[6] ausgeführt und deren Laufzeit gemessen. Es ist gut zu sehen, das einzelne Dense-Modelle, wie bereits im vorherigen Abschnitt vermutet, am schnellsten sind. LSTM-Modelle benötigen dagegen am längsten. Es sind auch bereits die im Folgenden getesteten Netze für die Anwendung mit dargestellt. Dabei wurde jeweils das gleiche Modell, allerdings mit drei, sechs und zehn Zeitschritten als Eingabe, verwendet. Eine größere Eingabe führt dabei auch zu einer größeren Laufzeit. Außerdem sind zusätzlich noch beispielhaft Experimente dargestellt, bei denen ein Netz pro Gelenk verwendet wird, wobei alle Gelenke als Eingabe dienen. Diese benötigen teilweise deutlich länger als alle anderen Netze. Dies zeigt nochmal, abgesehen von den Argumenten aus Abschnitt 5.6.2, dass sich diese Art nicht eignet für dieses Problem, da die Laufzeit viel größer ist und Netze mit ungefähr 50 000 Parametern bereits bis zu fast einem Drittel der gesamten Rechenzeit benötigen würden. Denn auf dem Roboter stehen bei B-Human aktuell bis zu 12 ms Rechenzeit zur Verfügung, wenn davon ausgegangen wird, dass durchgehend auf einem Kern gerechnet werden kann (siehe Abschnitt 3.1.1). Davon werden im Schnitt bisher nur ungefähr 0,45 ms

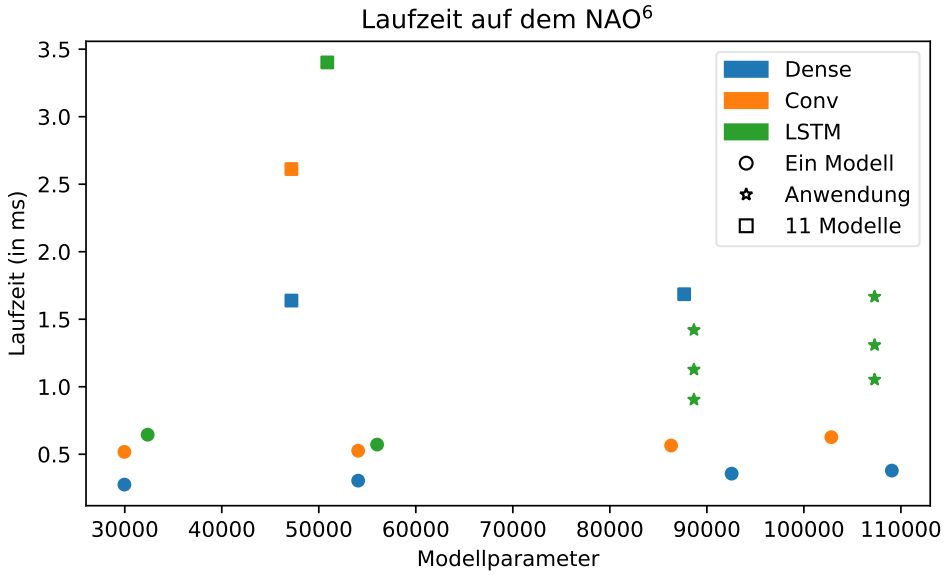


Abbildung 6.2: Laufzeiten unterschiedlicher Modelle auf dem NAO[6]. Die Farbe gibt dabei die Architektur und die Form das Modell an. Bei der Anwendung wurden drei, sechs und zehn Zeitschritte als Eingabe verwendet. Die anderen Modelle verwenden jeweils drei Zeitschritte.

verwendet. Somit ist die Laufzeit aktuell kein direkt begrenzendes Kriterium, wobei eine möglichst kurze Laufzeit trotzdem zu bevorzugen ist, damit andere und zukünftige Komponenten möglichst viel Rechenzeit zur Verfügung haben. Somit wurden die besten Modelle (siehe Abbildung 5.32) gewählt und diese jeweils mit drei, sechs und zehn Zeitschritten als Eingabe trainiert. Dabei handelt es sich um LSTM-Modelle mit vier bis fünf zusätzlichen Schichten mit jeweils 48 Einheiten (siehe Abschnitt 5.5.3). Fürs Training wurden dazu zufällig jeweils zehn Spiele eines Roboters von jedem Wettbewerb gewählt. Dabei handelt es sich um ungefähr 365 min Spielzeit.

Tests dazu wurden auf nicht zum Training verwendeten Daten durchgeführt, welche allerdings nicht auf bestimmte Kriterien testen. Zwar kann über die allgemeine Performance der Netze in speziellen Situationen ohne ein spezielles Testset keine direkte Aussage getroffen werden, allerdings liefert die Arbeit

bisher Ergebnisse, die auf eine generelle gute Funktionalität hinweisen. Wie gut die einzelnen Modelle zum Beispiel in Situationen sind, in denen der Roboter besonders instabil läuft, lässt sich somit aber nicht beantworten. Für ein Netz zur dauerhaften Verwendung bei B-Human würde sich zukünftig zumindest eine Kreuzvalidierung anbieten, um eine gute Auswahl an Daten für das Training zu finden, solange kein Testset zur Verfügung steht.

Die trainierten Netze wurden dann auf extra aufgenommenen Testdaten ausgewertet. Dazu ist in Abbildung 6.3 beispielhaft der Fehler zwischen Messung und Vorhersage für die Netze mit fünf zusätzlichen Schichten dargestellt. Als Vergleich ist zusätzlich der Unterschied zwischen der aktuellen Messung und der in zwei Zeitschritten angegeben. Wie zu sehen ist, liefern die Netze deutlich bessere Ergebnisse. Deren MAE ist dabei ähnlich zu denen der Kreuzvalidierung. Auch die Fehler der Ausreißer sind mit unter 3,5° Fehler deutlich kleiner. Mehr Zeitschritte als Eingabe führen dabei zu leicht besseren Ergebnissen.

Diese eignen sich somit sowohl von der Laufzeit als auch von der Performance für den vorläufigen Einsatz in der B-Human-Software. Es wurde das Netz mit drei Zeitschritten als Eingabe und fünf extra Schichten gewählt, da es mit ungefähr 1 ms Laufzeit die besten Ergebnisse für diese Eingabe liefert und aktuell unbekannt ist, ob mehr Zeitschritte als Eingabe zum Beispiel die Genauigkeit in Sondersituationen verbessern würden, sodass die zusätzliche Laufzeit einen Vorteil bringen würde.

6.3 Anwendung der Ergebnisse

Die vorhergesagten Gelenkwinkel können nun aber nicht direkt die Messung ersetzen. Der Grund dafür ist, dass zwar die Verzögerung bekannt ist, aber nicht wann die Messung aufgenommen wurde. In der B-Human-Software wird deshalb angenommen, dass die gemessenen Gelenkwinkel der aktuelle Zustand sind und somit die Anfragen verzögert umgesetzt werden. Die Verzögerung wird dann implizit an einigen Stellen mitbetrachtet und würde somit sonst doppelt betrachtet werden. Die Komponente kann aber verwendet werden, um eine neue Anfrage zu berechnen.

Die entwickelte Komponente wird dazu auch bereits von Reichenberg und

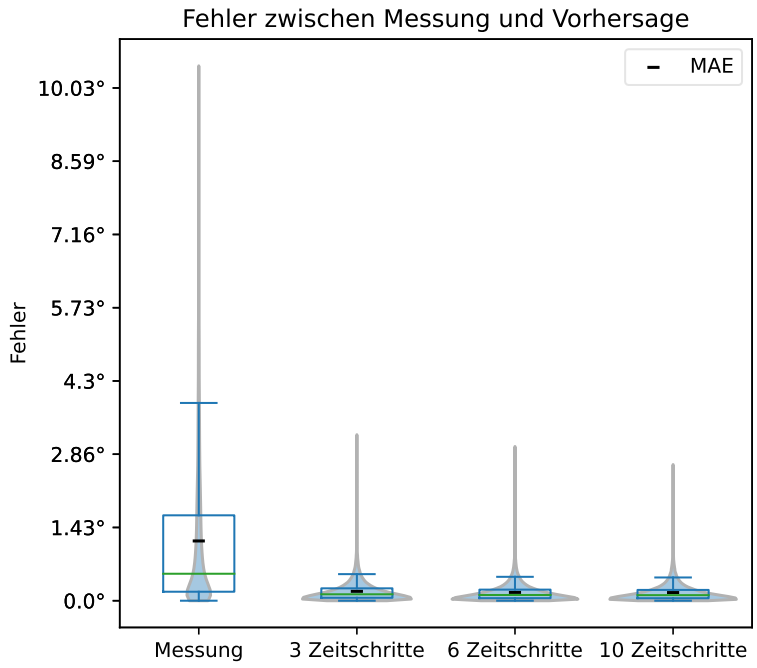


Abbildung 6.3: Fehlerverteilung für einen Test. *Messung* beschreibt den Unterschied zwischen der aktuellen Messung und der in zwei Zeitschritten. Die anderen sind die Vorhersagefehler der unterschiedlichen Netze.

Röfer (2023) verwendet. Sie beschreiben Verbesserungen am Laufen von B-Human, insbesondere für verschlissene Roboter. Dabei werden unter anderem aufgrund von Belastungen festsitzende Gelenke behandelt, also Gelenke, die ihren Zielwinkel nicht erreichen können, da sie nicht stark genug für die aktuelle Belastung sind. Auch die Kompensierung von Gelenkspiel ist beschrieben.

Typische Ansätze, um einen Roboter zu balancieren, beachten nicht, dass ein Gelenk möglicherweise Bewegungen gar nicht ausführen kann, da es nicht stark genug ist. Aus diesem Grund wird eine Anpassung der Gelenkgeschwindigkeit vorgestellt, die eine weitere Destabilisierung verhindert. Dazu wird die Vorhersage des neuronalen Netzes für den Standfuß verwendet, um den

Rotationsfehler von diesem zu bestimmen. Damit werden dann Bewegungen skaliert verlangsamt, die den Roboter instabiler machen würden. Dies sorgt dann dafür, dass der Roboter nicht instabiler wird.

Außerdem werden die Vorhersagen des neuronalen Netzes benutzt, um den Unterschied aus angefragten und gemessenen Gelenkwinkeln zu verringern. Dies ist zum Beispiel der Fall, wenn ein Gelenk festsitzt oder Gelenkspiel hat. Dazu wird der Unterschied aus angefragten und vorhergesagten Gelenkwinkeln verwendet, um die Anfrage so anzupassen, dass das Gelenk näher an die eigentliche Anfrage kommt und somit der Unterschied reduziert wird. Dies hat bei guten Robotern einen kleineren Einfluss, da die Anfragen besser umgesetzt werden, als bei abgenutzten Robotern.

6.4 Zusammenfassung

Zusammenfassend wurde eine Komponente implementiert, die es ermöglicht, Modelle dieser Arbeit in der B-Human-Software auszuführen. Dabei wurde festgestellt, dass die in den Experimenten verwendeten Modelle schnell genug sind, um bei B-Human auf dem Roboter ausgeführt zu werden. Aus diesem Grund wurden dann die Modelle mit den besten Ergebnissen mit mehr Daten trainiert und eines davon wird in der B-Human-Software als vorläufiges Netz eingesetzt. Dazu wurde ein Kompromiss aus Laufzeit und Performance gewählt, indem das beste Netz mit drei Zeitschritten als Eingabe gewählt wurde, da aktuell unbekannt ist, ob mehr Zeitschritte als Eingabe zum Beispiel die Genauigkeit in Sondersituationen verbessern würden. Falls das so wäre, würde sich die extra Laufzeit eines Netzes mit einer größeren Eingabe voraussichtlich lohnen. Außerdem könnte mit speziellen Testdaten auch geprüft werden, wie groß der Unterschied in speziellen Situationen ist, wenn Netze mit weniger Parametern verwendet würden, welches nochmals die Laufzeit senken würde. Diese Komponente wird auch bereits aktiv verwendet, um das Laufen zu stabilisieren.

7 Fazit und Ausblick

In der vorliegenden Arbeit wurde eine Gelenkwinkelvorhersage mit neuronalen Netzen für Laufbewegungen auf dem humanoiden Roboter NAO[6] auf Grundlage einer bekannten Verzögerung untersucht. Dazu wurde ein Datensatz aufbereitet. Auf einem kleinen Teil dieses Datensatzes wurden dann Experimente durchgeführt, um mögliche Architekturen zu finden. Anschließend wurden weitere Experimente durchgeführt, um zu untersuchen, wie gut der Ansatz generalisiert. Basierend auf den Ergebnissen wurde eine Komponente in der B-Human-Software entwickelt, welche Modelle ausführen kann und die Ergebnisse bereitstellt. Für diese wurden vorläufige Modelle trainiert, von denen eines aktiv verwendet wird, um das Laufen zu verbessern. Im Folgenden werden in Abschnitt 7.1 die erhaltenen Ergebnisse in Bezug auf die ursprünglichen Fragestellungen ausgewertet und ein Fazit gezogen. Abschließend wird in Abschnitt 7.2 ein Ausblick gegeben, was noch untersucht werden könnte und mögliche Erweiterungen und Verbesserungen vorgestellt.

7.1 Fazit

Die grundlegenden Fragen dieser Arbeit waren, ob und wie mit neuronalen Netzen die Gelenkwinkel für eine bekannte Verzögerung eines Roboters vorhergesagt werden können und wie gut dies funktioniert. Dazu wurden zunächst verschiedene Referenzimplementierungen erstellt, da keine andere Referenz, zu der verglichen werden könnte, bekannt ist. Die schlechteste stellt dabei den aktuellen Zustand in der B-Human-Software dar, in dem es keine Vorhersage gibt und die aktuelle Messung verwendet wird. Außerdem wurden verschiedene Architekturen ausgewählt, welche für Experimente verwendet wurden. Auf diesen Architekturen wurden dann verschiedene Experimente durchgeführt.

Dabei wurden zunächst Modelle verwendet, welche alle Gelenke auf einmal vorhersagen. Es wurde aber auch ausprobiert, ein Modell pro Gelenk zu verwenden, wobei nur das Gelenk, oder alle Gelenke als Eingabe verwendet wurden. Zusätzlich wurden verschiedene Loss-Funktionen und Mengen an Zeitschritten als Eingabe ausprobiert. Dabei wurde festgestellt, dass Modelle, welche alle Gelenke vorhersagen, am besten funktionieren. Diese liefern für die verwendeten Daten gute Ergebnisse. Sie sind um ein Vielfaches besser als alle Referenzimplementierungen und bereits relativ nah an der unteren Schranke, der Genauigkeit der Sensoren. Somit können die grundlegenden Fragen positiv beantwortet werden. Mehrere Modelle und andere Loss-Funktionen haben sich dagegen als nicht zielführend erwiesen.

Weitere Fragestellungen zur Generalisierung konnten anschließend in verschiedenen Kreuzvalidierungen ebenfalls positiv beantwortet werden. So kann ein Netz auch auf ungesehenen Robotern verwendet werden, auch wenn leichte Unterschiede bei einem Roboter festgestellt wurden. Außerdem beeinflussen die in den Daten vorhandenen, leicht unterschiedlichen Untergründe die Vorhersage nicht und diese spielen somit keine Rolle. Allerdings wurden deutliche Unterschiede in der Vorhersage festgestellt, wenn auf Daten von neueren oder älteren Robotern trainiert wird. Dabei liefern Daten älterer Roboter gute Ergebnisse für neue und ältere Roboter, umgekehrt gilt dies jedoch nicht. Dies hat voraussichtlich mit der Abnutzung der Gelenke zu tun, also dem Abreiben von Material von den Zahnrädern, was dazu führt, dass Zielpositionen nicht mehr präzise angesteuert werden können, da die Gelenke Spiel haben. Somit es sinnvoll, Daten von abgenutzten Robotern zum Training zu verwenden.

Um die Frage klären zu können, ob die Vorhersage schnell genug ist, um auf den Robotern eingesetzt zu werden, wurde eine Komponente in der B-Human-Software entwickelt und die Laufzeiten verschiedener Modelle gemessen. Dabei hat sich herausgestellt, dass die gewählten Modelle die Anforderung erfüllen. Anschließend wurden die besten Modelle der Experimente mit mehr Daten trainiert und eines davon wird jetzt vorläufig eingesetzt und auch aktiv verwendet, um das Laufen zu verbessern. Es wurde bereits auf einem offiziellen Wettbewerb in 2023 eingesetzt, bei dem B-Human mit 67:0 Toren in acht Spielen das Turnier gewann. Dabei sind die Roboter 50 % bis 75 % weniger

hingefallen als alle anderen Teams, wozu unter anderem die Ergebnisse dieser Arbeit beigetragen haben

Es gibt auch einiges, was im Nachhinein besser hätte gemacht werden können. Im Folgenden sind einige Beispiele aufgeführt. So ist der Durchschnitt nicht aussagekräftig genug, um im Detail sagen zu können, welche Netze wie gut funktionieren und welches das Beste für die Anwendung ist. Hier wäre eine eigene Implementierung der Metriken, welche zum Beispiel die Varianz mit betrachten, sinnvoll. Aktuell ist deshalb auch unklar, wie viele Trainingsdaten notwendig sind, um ein gutes Netz zu trainieren. Dies wird deutlich, da Netze mit einer sehr unterschiedlichen Menge an Trainingsdaten teilweise ähnliche durchschnittliche Ergebnisse liefern, aber die Varianz unbekannt ist. Auch eine frühere Betrachtung der tatsächlichen Anwendung auf dem Roboter und damit zusammenhängend der Laufzeit, wäre sinnvoll gewesen, da somit der Suchraum deutlich hätte reduziert werden können. Weiterhin könnte es zum Beispiel sinnvoll sein, insbesondere für die Kreuzvalidierungen die gleichen initialen Gewichte zu verwenden, damit alle Netze den gleichen Ausgangspunkt haben.

Insgesamt konnte in dieser Arbeit aber gezeigt werden, dass eine Vorhersage mit neuronalen Netzen möglich ist, gute Ergebnisse liefert und dabei schnell genug für den Einsatz auf dem Roboter ist.

7.2 Ausblick

Im Folgenden wird ein abschließender Ausblick gegeben, welche Erweiterungen oder Verbesserungen unter anderem noch möglich wären. Dabei ist zunächst zu erwähnen, dass die Erkenntnisse dieser Arbeit als Konferenzbeitrag beim RoboCup Symposium 2023 eingereicht wurden (Fiedler und Laue 2023). Sofern dieser akzeptiert wird, soll der im Rahmen der Arbeit erstellte Datensatz mit veröffentlicht werden. Außerdem soll die entwickelte Komponente überarbeitet werden und diese wird dann in der nächsten Veröffentlichung der B-Human-Software enthalten sein.

Zunächst sind hier einige mögliche Verbesserungen der Software aufgeführt. Das erste ist dabei die Erweiterung der Metriken, wie bereits im vorherigen Abschnitt beschrieben. Zusätzlich besteht noch Potenzial in der aktuellen Ver-

arbeitung der Daten beim Training. Dies könnte vollständig in TensorFlow implementiert werden und somit vermutlich die initiale Startzeit deutlich verringern. Dabei könnten die Daten auch nur noch einfach im Speicher vorgehalten werden und nicht mehrfach, wie aktuell. Weiterhin würden sich die Dense und Conv Architekturen mit einigen Anpassungen auch mit CompiledNN ausführen lassen. Dazu müsste unter anderem die Eingabe in mehrere Teile unterteilt werden und einige weitere Anpassungen vorgenommen werden. Dies würde wahrscheinlich die Laufzeit auf dem Roboter verbessern.

Eine allgemeine Verbesserungsmöglichkeit stellt ein expliziter Testdatensatz dar, welcher insbesondere Sondersituationen enthält, um den besten Kompromiss aus Laufzeit und Performance finden zu können. Dieser sollte Daten von neuen und abgenutzten Robotern und explizit unterschiedliches Laufverhalten, wie Straucheln oder Zweikämpfe, enthalten. Dies ermöglicht dann eine objektivere Metrik, welche genauen Parameter die besten Ergebnisse liefern, um gezielt ein gutes Netz für den langfristigen Einsatz auf dem Roboter trainieren zu können. Eine weitere Hilfe für die Analyse wäre der Fehler pro Ausgabe. Außerdem wäre es interessant zu untersuchen, welche Gelenke welchen Einfluss auf die ausgegebenen Gelenke haben. In der Bildverarbeitung werden dafür zum Beispiel *Saliency Maps* verwendet (Simonyan, Vedaldi und Zisserman 2014). Dies würde insbesondere bei der Version, die alle Gelenke als Eingabe und ein Gelenk als Ausgabe hat, zeigen, welche Gelenke voneinander abhängen. Auch könnten noch weitere Werte einbezogen werden, wie zum Beispiel die IMU, was zu noch besseren Ergebnissen führen könnte.

Die geschaffene Infrastruktur liefert zudem diverse Grundlagen, auf denen aufgebaut werden kann. So wurde in Abschnitt 3.2 bereits häufiger erwähnt, dass weitere Untersuchungen des Datensatzes möglich wären, wobei auf der geschaffenen Aufbereitung für Daten aufgebaut werden kann. Zudem könnten Gelenkwinkel für die beiden Kopfgelenke vorhergesagt werden, da diese häufig Spiel aufweisen und somit möglicherweise eine korrektere Kameraposition bestimmt werden könnte. Auch der Schrittwechsel könnte vorhergesagt werden. Die Daten dafür sind zwar nicht direkt in den Logs gespeichert, könnten aber aus den Daten generiert und dann verwendet werden.

Außerdem kann eine solche Vorhersage grundsätzlich in diversen weiteren

Gebieten Anwendung finden. So ist ein häufiges Problem von schnellen Bewegungen in der Robotik, dass es kurze Verzögerungen in der Regelschleife gibt, welche die Bewegungsplanung negativ beeinflussen. Auch wurde in Abschnitt 2.1 bereits ein Beispiel vorgestellt, in dem eine ähnliche Anwendung für Prothesen bei Menschen verwendet wird.

Literaturverzeichnis

Literatur

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin u. a. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. PDF: 45166.pdf (siehe S. 28).

B-Human Team (Okt. 2019). *B-Human Code Release*. Version 2019 (siehe S. 16).

B-Human Team (Okt. 2022). *B-Human Code Release*. Version 2022 (siehe S. 3, 16).

Behnke, Sven, Anna Egorova, Alexander Gloye, Raúl Rojas und Mark Simon (2004). „Predicting Away Robot Control Latency“. In: *RoboCup 2003: Robot Soccer World Cup VII*. Hrsg. von Daniel Polani, Brett Browning, Andrea Bonarini und Kazuo Yoshida. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 712–719. DOI: 10.1007/978-3-540-25940-4_70 (siehe S. 8).

Bergmeir, Christoph und José M. Benítez (2012). „On the use of cross-validation for time series predictor evaluation“. In: *Information Sciences* 191. Data Mining for Software Trustworthiness, S. 192–213. DOI: https://doi.org/10.1016/j.ins.2011.12.028 (siehe S. 77).

Böckmann, Arne (2015). „Entwicklung einer dynamischen Schussbewegung mit dem humanoiden Roboter NAO“. Masterarbeit. Universität Bremen, S. 7–10 (siehe S. 1, 4).

Böckmann, Arne und Tim Laue (2017). „Kick Motions for the NAO Robot Using Dynamic Movement Primitives“. In: *RoboCup 2016: Robot World Cup XX*. Hrsg. von Sven Behnke, Raymond Sheh, Sanem Sarıel und Daniel D. Lee. Cham: Springer International Publishing, S. 33–44. DOI: 10.1007/978-3-319-68792-6_3 (siehe S. 7).

Box, George EP, Gwilym M Jenkins, Gregory C Reinsel und Greta M Ljung (1976). *Time series analysis: forecasting and control*. John Wiley & Sons (siehe S. 9).

Brockwell, Peter J und Richard A Davis (1991). *Time series: theory and methods*. Springer (siehe S. 10).

Brown, Robert G und Richard F Meyer (1961). „The fundamental theorem of exponential smoothing“. In: *Operations Research* 9.5, S. 673–685 (siehe S. 10).

Brownlee, Jason (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery (siehe S. 34).

Dantzig, T. und J. Mazur (2007). *Number: The Language of Science*. Penguin Publishing Group (siehe S. 33).

Fiedler, Jan und Tim Laue (2023). „Neural Network-based Joint Angle Prediction for the NAO Robot“. In: *RoboCup 2023: Robot World Cup XXVI*. Eingereicht (siehe S. 97).

Gloye, Alexander (2005). „Lernmethoden für autonome mobile Roboter“. Diss. Freie Universität Berlin. Kap. 5. DOI: 10.17169/refubium-17024 (siehe S. 8).

Gouaillier, David, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre und Bruno Maisonnier (2009). „Mechatronic design of NAO humanoid“. In: *2009 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, S. 769–774. DOI: 10.1109/ROBOT.2009.5152516 (siehe S. 13).

Hansen, Nikolaus, Sibylle D. Müller und Petros Koumoutsakos (März 2003). „Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)“. In: *Evolutionary Computation* 11.1, S. 1–18. DOI: 10.1162/106365603321828970 (siehe S. 7).

Hasselbring, Arne und Andreas Baude (2022). „Soccer Field Boundary Detection Using Convolutional Neural Networks“. In: *RoboCup 2021: Robot World Cup XXIV*. Hrsg. von Rachid Alami, Joydeep Biswas, Maya Cakmak und Oliver Obst. Bd. 13132. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing, S. 202–213. DOI: 10.1007/978-3-030-98682-7_17 (siehe S. 87).

Hengst, Bernhard (2014). *rUNSWift Walk2014 Report*. Techn. Ber. Sydney 2052, Australien: University of New South Wales. PDF: Walk2014Report.pdf (siehe S. 13).

Hochreiter, Sepp und Jürgen Schmidhuber (Nov. 1997). „Long Short-Term Memory“. In: *Neural Computation* 9.8, S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (siehe S. 47).

Huang, Yongchuang, Zexia He, Yuxuan Liu, Ruiyuan Yang, Xiufeng Zhang, Guang Cheng, Jingang Yi, João Paulo Ferreira und Tao Liu (Dez. 2019). „Real-Time Intended Knee Joint Motion Prediction by Deep-Recurrent Neural

Networks“. In: *IEEE Sensors Journal* 19.23, S. 11503–11509. DOI: 10.1109/JSEN.2019.2933603 (siehe S. 9).

Hunter, J. D. (2007). „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3, S. 90–95. DOI: 10.1109/MCSE.2007.55 (siehe S. 28).

Kingma, Diederik P. und Jimmy Ba (2014). „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980*. DOI: 10.48550/arXiv.1412.6980 (siehe S. 34).

Kitano, Hiroaki, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda und Eiichi Osawa (1995). „RoboCup: The Robot World Cup Initiative“. In: *IJCAI-95 Workshop on Entertainment and AI/Alife*. DOI: 10.1145/267658.267738 (siehe S. 2).

Kuball, Jonas (2020). „Deep Reinforcement Learning für kooperatives Fußballspiel in einer Multi-Agenten Simulation“. Masterarbeit. Universität Bremen. PDF: master-thesis-jkuball.pdf (siehe S. 47, 48).

Matplotlib Developers (Jan. 2023). *matplotlib/matplotlib*. Version 3.6.3. DOI: 10.5281/zenodo.7527665 (siehe S. 28).

McKinney, Wes (2010). „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman, S. 56–61. DOI: 10.25080/Majora-92bf1922-00a (siehe S. 28).

Mosier, Charles I. (1951). „I. Problems and designs of cross-validation. Symposium: The need and means of cross-validation“. In: *Educational and Psychological Measurement* 11, S. 5–11. DOI: 10.1177/001316445101100101 (siehe S. 77).

O'Malley, Tom, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi u. a. (2019). *KerasTuner* (siehe S. 35).

Ommer, Nicolai, Alexander Stumpf und Oskar von Stryk (2018). „Real-Time Online Adaptive Feedforward Velocity Control for Unmanned Ground Vehicles“. In: *RoboCup 2017: Robot World Cup XXI*. Hrsg. von Hidehisa Akiyama, Oliver Obst, Claude Sammut und Flavio Tonidandel. Cham: Springer International Publishing, S. 3–16. DOI: 10.1007/978-3-030-00308-1_1 (siehe S. 8).

ONNX Runtime developers (Dez. 2021). *ONNX Runtime*. Version 1.10.0 (siehe S. 87).

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg u. a. (2011). „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12.85, S. 2825–2830 (siehe S. 35).

Poppinga, Bernd und Tim Laue (2019). „JET-Net: Real-Time Object Detection for Mobile Robots“. In: *RoboCup 2019: Robot World Cup XXIII*. Hrsg. von Stephan Chalup, Tim Niemueller, Jackrit Suthakorn und Mary-Anne Williams. Bd. 11531. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing, S. 227–240. DOI: 10.1007/978-3-030-35699-6_18 (siehe S. 87).

Rasul, Kashif (2021). *PyTorchTS*. Version 0.6.0 (siehe S. 44).

Reichenberg, Philip (2021). „Sturzverminderung von humanoiden Fußballrobotern unter Entwicklung eines passiven Zweikampfes“. Masterarbeit. Universität Bremen. PDF: master-thesis-preichenberg.pdf (siehe S. 1, 4, 16).

Reichenberg, Philip und Thomas Röfer (2022). „Step Adjustment for a Robust Humanoid Walk“. In: *RoboCup 2021: Robot World Cup XXIV*. Hrsg. von Rachid Alami, Joydeep Biswas, Maya Cakmak und Oliver Obst. Bd. 13132. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing, S. 28–39. DOI: 10.1007/978-3-030-98682-7_3 (siehe S. 16, 44).

Reichenberg, Philip und Thomas Röfer (2023). „Dynamic Joint Control For A Humanoid Walk“. In: *RoboCup 2023: Robot World Cup XXVI*. Eingereicht (siehe S. 91).

RoboCup Technical Committee (2022). *RoboCup Standard Platform League (NAO) Rule Book*. PDF: SPL-Rules-2022.pdf (siehe S. 2).

Röfer, Thomas, Tim Laue, Nikolai Bahr, Jonah Jaeger, Jannes Knychalla, Thorben Lorenzen, Nele Matschull, Yannik Meinken, Lukas Malte Monnerjahn, Lukas Plecher und Philip Reichenberg (2021). *B-Human Team Report and Code Release 2021*. Techn. Ber. 28359 Bremen, Deutschland: Universität Bremen. PDF: CodeRelease2021.pdf (siehe S. 13, 14, 16).

Röfer, Thomas, Tim Laue, Andreas Baude, Jan Blumenkamp, Gerrit Felsch, Jan Fiedler, Arne Hasselbring, Tim Haß, Jan Oppermann, Philip Reichenberg, Nicole Schrader und Dennis Weiß (2019). *B-Human Team Report and Code Release 2019*. Techn. Ber. 28359 Bremen, Deutschland: Universität Bremen. PDF: CodeRelease2019.pdf (siehe S. 16).

Röfer, Thomas, Tim Laue, Finn Marvin Ewers, Enrico Göhrs, Michelle Gusev, Arne Hasselbring, Jo Lienhoop, Ayleen Lührsen, Yannik Meinken, Philip Reichenberg, Laurens Schiefelbein, Florian Scholz, Sina Schreiber und Simon Werner (2022). *B-Human Team Report and Code Release 2022*. Techn. Ber. 28359 Bremen, Deutschland: Universität Bremen. PDF: CodeRelease2022.pdf (siehe S. 4, 13, 16, 87, 88).

Röfer, Thomas, Tim Laue, Gerrit Felsch, Arne Hasselbring, Tim Haß, Jan Oppermann, Philip Reichenberg und Nicole Schrader (2019). „B-Human 2019

– Complex Team Play Under Natural Lighting Conditions“. In: *RoboCup 2019: Robot World Cup XXIII*. Hrsg. von Stephan Chalup, Tim Niemueller, Jackrit Suthakorn und Mary-Anne Williams. Bd. 11531. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing, S. 646–657. DOI: 10.1007/978-3-030-35699-6_52 (siehe S. 87).

Salkin, Harvey M. und Cornelis A. De Kluyver (1975). „The knapsack problem: A survey“. In: *Naval Research Logistics Quarterly* 22.1, S. 127–144. DOI: 10.1002/nav.3800220110 (siehe S. 33).

Seekircher, Andreas und Ubbo Visser (2017). „A Closed-Loop Gait for Humanoid Robots Combining LIPM with Parameter Optimization“. In: *RoboCup 2016: Robot World Cup XX*. Hrsg. von Sven Behnke, Raymond Sheh, Sanem Sarıel und Daniel D. Lee. Cham: Springer International Publishing, S. 71–83. DOI: 10.1007/978-3-319-68792-6_6 (siehe S. 7).

Simonyan, Karen, Andrea Vedaldi und Andrew Zisserman (2014). „Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps“. In: DOI: 10.48550/arXiv.1312.6034 (siehe S. 98).

Smith, Otto JM (1959). „A controller to overcome dead time“. In: *ISA J.* 6, S. 28–33 (siehe S. 9).

TensorFlow Developers (Sep. 2022). *TensorFlow*. Version 2.10.0. DOI: 10.5281/zenodo.7604243 (siehe S. 28).

The Pandas Development Team (Jan. 2023). *pandas-dev/pandas: Pandas*. Version 1.5.3. DOI: 10.5281/zenodo.7549438 (siehe S. 28).

Thielke, Felix und Arne Hasselbring (2019). „A JIT Compiler for Neural Network Inference“. In: *RoboCup 2019: Robot World Cup XXIII*. Hrsg. von Stephan Chalup, Tim Niemueller, Jackrit Suthakorn und Mary-Anne Williams. Bd. 11531. Lecture Notes in Artificial Intelligence. Cham: Springer International Publishing, S. 448–456. DOI: 10.1007/978-3-030-35699-6_36 (siehe S. 87).

Vijayakumar, Sethu, Aaron D'Souza und Stefan Schaal (Dez. 2005). „Incremental Online Learning in High Dimensions“. In: *Neural Computation* 17.12, S. 2602–2634. DOI: 10.1162/089976605774320557 (siehe S. 8).

Vijayakumar, Sethu und Stefan Schaal (2000). „Locally Weighted Projection Regression : An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space“. In: *Proceedings of the seventeenth international conference on machine learning (ICML 2000)*. Bd. 1. Morgan Kaufmann, S. 288–293 (siehe S. 8).

Internetquellen

Aldebaran (2023). *About Aldebaran, a part of United Robotics Group*. URL: https://www.aldebaran.com/en/aldebaran (besucht am 10. 04. 2023) (siehe S. 11).

B-Human (2022). *Architecture*. Wiki des Teams B-Human. URL: https://wiki.b-human.de/coderelease2022/architecture (besucht am 10. 04. 2023) (siehe S. 2, 13).

B-Human (2023). *B-Human. RoboCup Standard Platform League*. Offizielle Webseite des Teams B-Human. URL: https://www.b-human.de (besucht am 10. 04. 2023) (siehe S. 1).

Olah, Christopher (2015). *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 10. 04. 2023) (siehe S. 48).

SoftBank Robotics (2023a). *NAO^6 - Developer Guide*. URL: http://doc.aldebaran.com/2-8/family/nao_technical/index_naov6.html (besucht am 10. 04. 2023) (siehe S. 12).

SoftBank Robotics (2023b). *NAO^6 - Technical overview*. URL: http://doc.aldebaran.com/2-8/family/nao_technical/index_dev_naov6.html (besucht am 10. 04. 2023) (siehe S. 11–14, 75, 80).

SoftBank Robotics (2023c). *NAOqi - LoLA*. URL: http://doc.aldebaran.com/2-8/naoqi/lola/lola.html (besucht am 10. 04. 2023) (siehe S. 4, 12, 13, 22).

TensorFlow Developers (2023). *Time series forecasting*. URL: https://www.tensorflow.org/tutorials/structured_data/time_series (besucht am 10. 04. 2023) (siehe S. 31, 44).

The RoboCup Federation (2023a). *A Brief History of RoboCup*. URL: https://www.robocup.org/a_brief_history_of_robocup (besucht am 10. 04. 2023) (siehe S. 2).

The RoboCup Federation (2023b). *Past Events*. URL: https://www.robocup.org/events/past_events (besucht am 02. 02. 2023) (siehe S. 2).

The RoboCup Federation (2023c). *RoboCup Objective*. URL: https://www.robocup.org/objective (besucht am 10. 04. 2023) (siehe S. 2).