Masterthesis

Entwurf und Implementierung eines Controllers für einen tennisspielenden Roboter

Nils Drebing

2 9 0 9 3 0 5

ndrebing@uni-bremen.de

16. August 2020

Erstgutachter	Prof. Dr. Udo Frese
Zweitgutachter	Prof. Dr. Rolf Drechsler

Universität Bremen

Fachbereich 03: Informatik/Mathematik

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel verfasst zu haben. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Ich bestätige außerdem, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Ort, Datum	Nils Drebing

Zusammenfassung

Robotische Manipulatoren führen in der Industrie seit Jahrzehnten allerlei simple, komplexe, sich wiederholende oder gar für Menschen gefährliche Arbeiten aus. Jedoch gibt es auch ein großes Potenzial und Interesse in der Verwendung dieser Systeme in Sport- und Unterhaltungsanwendungen.
Im Kontext von Ballsportarten ist Tischtennis ein Anwendungsbereich, der bereits seit den achtziger Jahren intensiv untersucht wird. Tennis als Sport fasziniert Menschen auf der ganzen Welt, da er leicht verständlich und leicht auszuüben ist; menschlichen Spielern auf Expertenniveau jedoch Leistungen an der Grenze ihrer körperlichen Belastbarkeit abfordert.
In dieser Arbeit wird ein pragmatisches Gesamtkonzept für das Spielen von Softbällen mit einem Hartplastikschläger auf einem robotischen Manipulator entworfen. Als Referenzsystem wird der *Panda*-Manipulator der Firma *Franka Emika* verwendet. Als Anwendungsfall ist das Zuwerfen von Bällen auf Messen angedacht, sodass die Bälle volley geschlagen werden. Zu diesem Konzept gehört die Adaption eines bereits vorhandenen, probabilistischen Trackingverfahrens auf Basis von Stereokamerabildern. Im Fokus steht die Bewegungsplanung und -steuerung auf Motorebene für sinnvolle Schlagbewegungen. Die Entwicklung und Evaluation des erstellten Softwarepakets erfolgt in einer selbsterstellten Simulationsumgebung.

Abstract

Robotic manipulators have been used for years in industrial fields to execute simple, complex, repetitive or even dangerous tasks that a human may not perform unharmed. Yet there is an astounding potential and interest for the usage of robots in entertainment and sports applications.
In the context of ball sports, table tennis is a task that has been researched in depth since the early eighties for robotic systems of different shapes and sizes. Tennis as a sport is fascinating people all around the world since it is easy to understand and simple to pick up; yet it drives professional players to their physical limits.
In this thesis a pragmatic approach for playing tennis with a softball and a plastic racket is proposed. The *Panda*-manipulator, designed and manufactured by *Franka Emika*, is used as the robotic reference platform. As a use case it is considered that visitors of an exhibition throw balls at the robot so the ball has to be volleyed. As part of the approach, an existing ball tracking application based on stereo vision was adapted. The main focus lies on the design of a motor control component for executing useful ball hitting motions. For development and testing purposes, a simulation environment is implemented.

Genderklausel

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

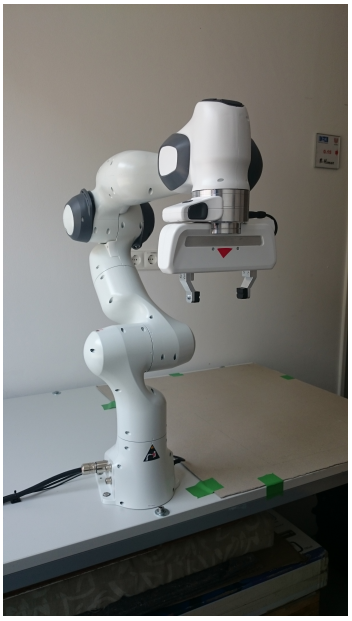


Abbildung 1.1: Manipulator *Panda* der Firma *Franka Emika*, hier mit Greiferwerkzeug.

Während simple Automaten bereits den Alltag der meisten Menschen bereichern, könnten komplexe Robotersysteme mit multiplen Freiheitsgraden in Zukunft auch einen Platz im Sport- und Unterhaltungssektor finden. Solche Systeme werden für Forschung und Lehre immer erschwinglicher. Ein mögliches Anwendungsszenario für solche Manipulatoren ist das Zurückspielen von Bällen in einem Tennismatch.
Ob man sich nun gegen einen Menschen oder ein Roboter behaupten muss: Tennis ist ein Wettbewerb, deren Ausübung jeder Mensch leicht erlernen und verstehen kann; dessen Perfektionierung allerdings jahrzehntelanges Training erfordert. Der daraus resultierende intensive Wettbewerbscharakter dieser Sportart fasziniert Menschen auf der ganzen Welt. Zum Spielen von Tennis ist es notwendig, die Flugbahn des Balls zu erkennen und auf Basis dieser Beobachtung eine passende Armbewegung auszuführen, sodass der Ball zum Gegner zurückfliegt. Die Abbildung der komplexen Denk- und Bewegungsprozesse eines Menschen auf ein automatisiertes, mechanisches System stellt dabei eine besondere Herausforderung dar.

Das Ziel dieser Arbeit ist es, ein Gesamtkonzept für einen robotischen Manipulator mit sieben Freiheitsgraden für das Zurückspielen von Softtennisbällen zu entwerfen und in einem Softwarestack zu implementieren. Dazu gehört die Auswahl eines passenden Verfahrens sowohl für die Balldetektion als auch die anschließende Ansteuerung des robotischen Systems.
Als Anwendungsszenario in dieser Thesis ist ein Messe- oder Festauftritt vorgesehen, bei dem Besucher Bälle werfen, welche daraufhin vom Roboter volley zurückgeschlagen werden. Bis dato beschäftigen sich die meisten Arbeiten in diesem Bereich mit Spielen von Tischtennis. Der entscheidende Unterschied zwischen dem Anwendungsszenario und Tischtennis liegt in der Distanz, die der Ball geschlagen werden muss und dem dazu notwendigen Kraftaufwand. Darüber hinaus wird in dieser Arbeit ein größerer Hartplastikschläger verwendet, dessen Masse einen entscheidenden Einfluss auf die mechanische Belastung der Gelenke hat.

1.2 Aufbau der Arbeit

Zunächst werden die Grundkonzepte, die zum Verständnis der einzelnen Berechnungen erforderlich sind, erläutert. Dazu gehört die Untersuchung der biomechanischen Prozesse, die der Mensch beim Spielen von Tennis durchläuft sowie die Erklärung kinematischer Prinzipien und der Kamerakalibrierung. Aus dem aktuellen Stand der Technik werden Ansätze für die Ballverfolgung und Ansteuerung des Roboters untersucht. Die ausgewählten Verfahren werden daraufhin in passenden Softwarekomponenten modelliert. Anschließend wird die ausgewählte Hardware, die zur Evaluation der Kontrollarchitektur verwendet wird, beschrieben. In der Implementierung wird die Kalibrierung eines Stereokamerasystems sowie die Entwicklung einer Simulationsumgebung näher beschrieben. Zum Schluss wird das Gesamtsystem in der entwickelten Simulation getestet und evaluiert.

1.3 Beiträge der Arbeit

- **Adaption eines Trackingverfahrens**: Es wurde ein Programm für die Datenübertragung der gewählten Kamerahardware implementiert. Darüber hinaus wurde ein, in der Arbeitsgruppe bereits verwendeter Trackingalgorithmus adaptiert.
- **Entwicklung eines Tools zur halbautomatischen Stereokalibrierung**: Zur Kalibrierung des neuen Kamerasystems wurde ein simples, grafisches Tool entwickelt.
- **Pragmatischer Ansatz zur Berechnung und Ansteuerung von Abfangtrajektorien als Startpunkt für zukünftige Arbeiten**

2 Stand der Technik

2.1 Ballverfolgung

Ein Balltracker hat die Aufgabe, eine Schätzung über die Position sowie Linear- und Winkelgeschwindigkeiten möglichst genau zu messen und auf dieser Basis eine Vorhersage über die Flugbahn des Objekts zu machen. Für diese Aufgabe gibt es verschiedene Lösungsansätze.
Beim Golf kommen Radarsysteme zum Einsatz. Die Genauigkeit bei diesen Verfahren liegt in Bezug auf die Rotations- und Bewegungsgeschwindigkeit bei 0.05% (vgl. [Li+15], S. 832). Allerdings bringt diese Technologie entscheidende Nachteile mit sich. Zum einen ist die Einrichtung mit enormem Aufwand und Kosten verbunden. Zum anderen sind diese Systeme lediglich für Sportarten nutzbar, die auf großen, offenen Flächen ausgetragen werden, da Radarwellen in geschlossenen Räumen miteinander interferieren und Messungen folglich verrauschen.
Für Ballverfolgung in einem kleineren Szenario sind bildbasierte, zweistufige Verfahren ein populärer Ansatz. Aus den Bilddaten werden projektilspezifische Features extrahiert; für Tennisbälle liegt die Abbildung auf dem Bild durch einen annähernd idealen Kreis nahe. Durch Stereorekonstruktion werden Ballpose und das daraus resultierende Dynamikmodell approximiert und auf Basis zukünftiger Messungen angepasst. Solch ein Aufbau hat den Vorteil, dass er auch in geschlossenen Räumen verwendet werden kann.
Entscheidend für die Präzision solcher Ansätze ist die Qualität der extrahierten Ballmessungen auf Bildebene in Bezug auf seinen Durchmesser und dem Sphärenzentrum. Ronkainen und Harland [RH10] projizieren dazu mittels eines externen Lasers ein Muster auf den Ball, welches aufgrund eines Wellenlängenfilters vor der Linse eindeutig im Kamerabild identifizierbar ist (siehe Abbildung 2.1). Dadurch ist das Zentrum des Balls präzise auf Pixelebene bestimmbar. Der Nachteil bei diesem Ansatz ist die zusätzliche Verwendung

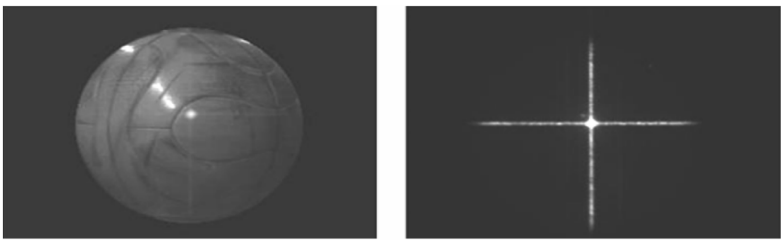


Abbildung 2.1: Extrahierte Ballinformation mittels Laserprojektion. links: Abbildung ohne Wellenlängenfilter. Rechts: mit Filter. ([RH10], S. 222)

eines Laseremitters, der entlang der Blickachse der Kamera ausgerichtet ist. Birbach et al. [BFB11] benutzen einen beleuchtungsinvarianten Gradientenfilter zur Bestimmung von Kreisen in einem Stereokamerasystem und schätzen auf Basis eines Ballmodells die Position und Geschwindigkeit in einem *Unscented Kalman Filter (UKF)*. Der Vorteil bei diesem Ansatz sind die geringen Hardwareanforderung in Form von mindestens zwei Kameras.
Eine große Herausforderung bei bildbasierten Algorithmen ist die Bestimmung der Winkelgeschwindigkeit. Bei den bisher vorgestellten Verfahren wird diese nicht modelliert. Tebbe et al. [TKZ19] trainierten zur Bestimmung der Rotation ein *Convolutional Neural Network (CNN)*, welches ein Firmenlogo auf den Aufnahmen einer Highspeedkamera erkennt und durch die Positionsänderung über die Zeit ein Rotationsmodell approximiert. Das daraus resultierende dynamische Modell, welches den Magnuseffekt mit einbezieht, liefert eine Genauigkeit der Trajektorienvorhersage von durchschnittlich 88.2% (vgl. [TKZ19]). Ein entscheidender Nachteil bei diesem Ansatz ist die enorme Menge an benötigten Trainingsdaten für das neuronale Netz.

2.2 Bewegungsgenerierung

Spielerische Anwendungen von robotischen Systemen sind nicht erst in den letzten Jahren populär geworden, sondern reichen weit zurück die achtziger Jahre [And88]. In diesem Kapitel soll grob auf den Stand der Technik bezüglich Steuerungsverfahren zum Spielen eines Balls eingegangen werden. Dazu werden einige Systeme gezeigt, die auf die ein oder andere Weise in einem Ballspielszenario agieren.

2.2.1 *Virtual Hitting Plane (VHP)*

Beim Ballspielen ist eine Manipulation des Spielzustandes erst dann möglich, wenn der Spieler ihn mit dem Schläger erreichen kann. Sind Geschwindigkeits- und Positionswerte des zurückzuspielenden Objekts zu einem Zeitpunkt bekannt, so lassen sich zukünftige Schnittpunkte mit dem Arbeitsraum der Manipulators approximieren. In der Literatur sind diese Klasse von Ansätzen auch als *Virtual Hitting Plane (VHP)*-Methoden bekannt. Diese Verfahren basieren auf der Theorie, dass Menschen fest definiterte, virtuelle Schlagpositionen auswählen (siehe Kapitel 3.1).

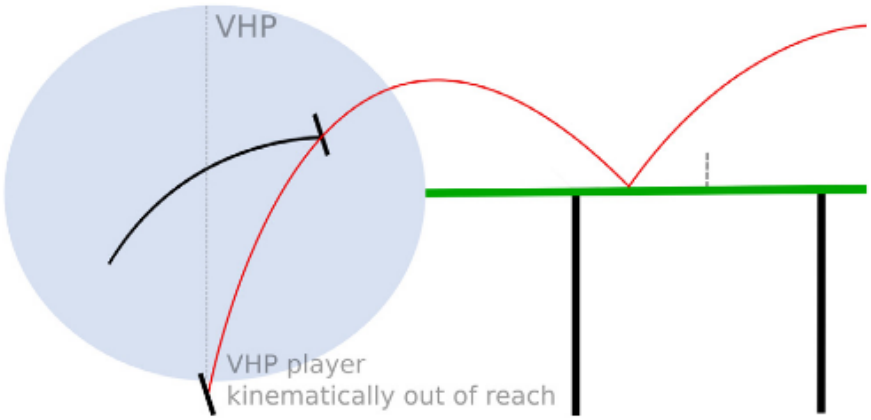


Abbildung 2.2: Darstellung der *Virtual Hitting Plane*-Methodik. Es wird ein konkreter Abschlagszeitpunkt mittels Heuristik ausgewählt, der im Arbeitsraum des Manipulators liegt. Rot: Ballflugbahn; Schwarz: Schlägerbewegung. [KMP18]

Dabei wird der Abschlagszeitpunkt durch die Wahl einer Ebene entlang der Ballflugbahn diskretisiert. Die Bestimmung der Rückspielgeschwindigkeit geschieht durch eine Rückwärtsberechnung des aerodynamischen Modells des Balls und der Invertierung des Kontaktmodells zwischen Ball und Endeffektor [KMP18]. Die nötige Gelenkkonfiguration wird über die inverse Kinematik gelöst. In [Mat+05] wurde dieser Ansatz für einen Vier-*Degrees of Freedom (DOF)*-Manipulator verwendet. Bedingt durch die Konstruktion ist das Planungsproblem für diesen Manipulator allerdings einfacher, da der Zustandsraum der Gelenkkonfiguration kleiner und das kinematische Problem wesentlich simpler ausfällt. Mülling et al. [Mül+09] übertrugen diesen Ansatz erfolgreich auf einen Sieben-*DOF*-Manipulator mit einer durchschnittlichen Rückspielrate von 78% bei zufälligen Ballwürfe in einem Tischtennisszenario.
Bei diesen Ansätzen ist es möglich, sie in Echtzeit auf dem System anzuwenden, da das Kernproblem in Subprobleme unterteilt und in der jeweiligen Domäne optimiert wird. Aus der getrennten Optimierung resultiert jedoch auch in wesentlicher Nachteil dieser Ansätze. Die Diskretisierung von Schlagzeitpunkt in Korrelation mit der Schlagpose und der Bewegungsplanung führt dazu, dass die jeweiligen Optimierungen nur ein lokales, einzelnes Optimum für das jeweilige Problem abbildet. Zudem muss eine passende Heuristik für die Wahl der Posenrotation entlang der Schlägernormalen ermittelt werden.

2.2.2 Reinforcement-Learning

Im Rahmen von Ballspielen ist der spielrelevante Zustandsraum oft sehr groß. Maschinelle Lernverfahren sind in diesem Fall ideal für die Optimierung von Modellen mit vielen Eingabeparametern. In [Hac18] wurde ein Deep-Reinforcement-Learning-Verfahren auf dem dreiachsigen Unterhaltungsroboter *Doggy* evaluiert. Dieser Ansatz eignet sich besonders

bei Systemen, über die keine Informationen zu relevanten Dynamikparametern vorliegen, da sowohl Schlagpolicies als auch die dynamischen Modellparameter in einer Simulationsumgebung gelernt werden. Nach Aussage des Autors sei eine sehr gute Generalisierbarkeit für das gezielte Spielen von Bällen mit zweiachsigen Policies möglich. Allerdings sei die Trainingszeit von zehn Stunden in der Simulation bereits ein großer Zeitfaktor gewesen. Ein siebenachsiges System, wie es in dieser Arbeit verwendet wird, bietet noch einen wesentlich größeren, trainierbaren Parametersatz, weshalb dieses Verfahren nicht direkt in diesem Szenario ohne Simplifizierung anwendbar wäre. Darüber hinaus besteht die Gefahr, dass der Transfer auf das reale System eine größere Herausforderung darstellt als das eigentliche Lernproblem, da bestimmte Dynamiken ausgelassen wurden.
In [Mah+18] wird Reinforcement-Learning eingesetzt, um einer hierarchische Kontrollarchitektur ein Modell für die High-Level-Strategiewahl zu trainieren. Für die darunterliegenden Bewegunsprimitive wurden Daten menschlicher Spieler über ein VR-System aufgezeichnet. Die eigentliche Policy aus diesem Primitiven wurde durch simulierte Spiele zweier robotischer Agenten gelernt.

2.2.3 Dynamic Motor Primitives

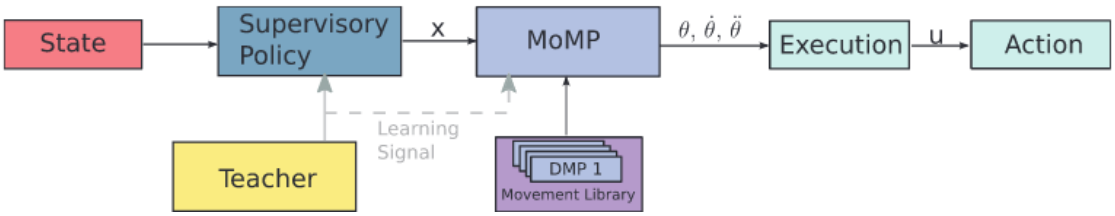


Abbildung 2.3: Aufbau des MoMP-Ansatzes. [KMP18]

Beim Erlernen neuer Verhaltensmuster für wohldefinierte Umgebungen werden die Aufgaben aus der Lösung kleinerer Subprobleme zusammengesetzt. Im Kontext der Bewegung bedeutet das, dass komplexe Bewegungen eine Abfolge kleinerer sogenannter Bewegungsprimitive sind, die entsprechend des neu beobachteten Zustands parametrisiert werden. Solche Ansätze werden auch als *Dynamic Motion Primitive (DMP)*-Verfahren bezeichnet. Mülling et al. [Mül+13] entwickelten dazu ein Framework namens *Mixture of Motor Primitives (MoMP)* (vgl. Abbildung 2.3). Dazu generierten sie mittels Imitations- und kinästhetischem Lernen eine Bibliothek aus Bewegunsprimitiven für einen Sieben-*DOF*-Manipulator, wobei zu jedem Bewegungsprimitiv der beobachtete Zustand x gespeichert wird. Auf Basis dieser Primitive wurde ein Gateway-Netz trainiert, das für eingehende Beobachtungen x der Ball- und Gelenkszustandsmessungen eine lokaloptimale Policy aus Motorprimitiven liefert.
In einer Simulationsumgebung wurde das System über 100 Episoden trainiert, bevor es auf dem echten Manipulator in Wurfexperimenten mittels Ballwurfmaschine getestet wurde.

Auf Basis des Simulationstrainings wurde eine Returnrate von 67% erreicht. Nach weiteren 150 Trainingsepisoden auf dem realen System wurde diese Rate auf 97% gesteigert.

2.2.4 Optimal Control

Alternativ wird in [KMP18] das gesamte Tennisproblem als nichtlineares Optimierungsproblem mit Nebenbedingungen konstruiert. Dabei wird die Steuerungsfunktion als parametrisiertes Polynom dritten Grades über Gelenkposition und -geschwindigkeit sowie den Abschlagszeitpunkt T optimiert, gegeben der mechanischen Nebenbedingungen der Hardware und des Dynamikmodells des Balls über die Zeit.
Der Vorteil dieses Optimal Control-Ansatzes liegt darin, dass alle Freiheitsgrade mit in die Schlagbewegung einbezogen werden, sodass zum einen die mechanische Belastung auf den gesamten Manipulator verteilt wird und zum anderen natürlichere Bewegungsmuster bei der Ausführung entstehen. Die durchschnittliche Returnrate beträgt in [KMP18] etwa 80%. Doch dieser Vorteil wird mit Rechenzeit erkauft: die Optimierung mittels *Sequential Quadratic Programming* dauert etwa 2 Sekunden auf einem aktuellen Computersystem. Um diesen Nachteil auszugleichen, wurde eine Lookup-Tabelle propagiert, welche mit offline optimierten Policies befüllt wird. Damit lässt sich die Optimierungszeit (bis zu einer gewissen Größe der Tabelle) auf wenige Millisekunden beziffern, allerdings sinkt die Returnrate auf etwa 60%, da untrainierte Ballzustände nicht spielbar sind.
In [Den15] wird Optimal-Control für die Schlagbewegung auf dem, ebenfalls in [Hac18] verwendeten dreiachsigen *Doggy* benutzt. Für den komplexen humanoiden Roboter *Rollin' Justin* [Bor+09] wird ebenfalls ein Optimal-Control-Verfahren für das Fangen von Bällen [BWH10] angewandt.

3 Theoretische Grundlagen

In diesem Kapitel sollen die Grundlagen erörtert werden, die zum weiteren Verständnis des vorgestellten Ansatzes nötig sind. Um ein Gefühl für die Anforderungen beim Tennis zu bekommen, werden physikalische Grundlagen erläutert, die beim Spielen relevant sind. In Bezug auf die mechanische Komponente wird anschließend die Definition und Berechnung der Vorwärtskinematik eines Manipulators sowie das Problem der inversen Kinematik beschrieben, welches für die Wegplanung des Roboters entscheidend ist. Darüber hinaus wird die Jacobimatrix zur Bestimmung der Endeffektorgeschwindigkeit erläutert. Zuletzt werden die Grundlagen des Kameramodells näher gebracht, die für die Ballverfolgung von Bedeutung sind.

3.1 Das Tennisspiel beim Vorbild Mensch

Moderne Computersysteme sind in der Lage, eine enorme Anzahl von Rechenoperationen in kurzer Zeit durchzuführen und sind in diesem Aspekt wesentlich effizienter als Menschen. Nichtsdestotrotz übertrifft ein menschlicher Spieler ein robotisches System im Kontext eines Ballspiels in jeder Hinsicht. Deshalb liegt es nahe, die motorischen und gedanklichen Prozesse eines Tennisspielers zu analysieren.
Bei der Wahl der Schlagbewegung stehen drei essenzielle Voraussetzungen im Vordergrund, die erfüllt werden müssen, damit diese im Kontext des Tennisspiels sinnvoll ist.

1. Die Trajektorie des Schlägers muss entlang der Flugbahn des Balls liegen.
2. Sobald Ball und Schläger kollidieren, muss durch Ausrichtung und Endgeschwindigkeit des Schlägers im Moment der Kollision die Geschwindigkeit und Richtung des Balls verändern.
3. Der Ball muss über das Netz auf der Seite des Gegners landen.

Der Tennisschläger fungiert dabei als Schnittstelle zwischen dem Spieler und der Spielumgebung, mit welcher der Spielzustand verändert werden kann, indem die oben genannten Voraussetzungen erfüllt werden.

Ramanantsoa et al. [RDD94] untersuchten 1994 das Tischtennisspiel im Bezug auf die biomechanischen Prozesse, die professionelle Spieler während eines Matches durchlaufen. Dazu wurden Tischtennismatches der Spieler mit einer Highspeedkamera aufgezeichnet und analysiert.

Quelle: Wikimedia Commons [Wik06]

Quelle: John Beaufort [Bea]

Abbildung 3.1: Darstellung der Wartephase. Sowohl beim Tischtennis als auch beim regulären Tennis wird der Schläger vor dem Körper gehalten.

Bei der Videoanalyse der Profispieler ließen sich vier Spielzustände identifizieren, die zyklisch bei jedem Zurückschlagen eines Balls durchlaufen werden:

Wartephase:
In dieser Phase wartet der Spieler darauf, dass der Ball vom Kontrahenten seine Richtung geschlagen wird. Als sensorischer Input dienen hauptsächlich die Augen, welche sowohl die Ballzustand als auch die Position seines Gegners verfolgen. Der Körper des Spielers befindet sich in einer Warteposition: der Schläger wird vor dem Körper gehalten, die Beine sind angewinkelt (vgl. Abbildung 3.1). Die Schlägerposition ist aus zwei Gründen häufig zu beobachten. Zum einen ist die Haltung nicht anstrengend, da die Armmuskulatur kaum belastet wird. Der Hauptaspekt liegt jedoch in der Möglichkeit, dass beim Richtungsspiel des Gegners schnell zwischen Vor- und Rückhand gewechselt werden kann. Bei der Beinposition wird der Körperschwerpunkt des Spielers näher zum Boden bewegt. Auf diese Weise kann eine laterale Positionsanpassung schneller erfolgen.
Diese Phase dauert an, bis der Ball nach dem Schlag des Kontrahenten kurz vor der Netz ist. Zu diesem Zeitpunkt ist eine Approximation der Ballflugbahn gegeben. Die Autoren postulieren, dass zu diesem Zeitpunkt ein virtueller Schlagpunkt vom Spielenden gewählt werde, den es nun anzusteuern gilt.

Vorbereitungsphase:
Um die Balltrajektorie mit dem Schläger zu kreuzen, werden in dieser Phase Positionsänderungen vorgenommen. Der Spieler bewegt sich zum einen mit seinen Beinen, um seine laterale Position anzupassen und grob in die Richtung des Abfangpunktes zu gelangen. Gleichzeitig wird zum Schwung ausgeholt, damit der Schläger in der nächsten Phase die erforderliche Geschwindigkeit erreicht.

Schlagphase:
Die kritischste Phase des Spiels ist die Ausführung der Schlagbewegung. Aus der Videoanalyse ging hervor, dass durchschnittlich $80ms$ vom Beginn der Schlagbewegung bis zum

Abfangen des Ball vergehen. Hier wird der Schläger aus der Ausholposition Richtung Abfangpunkt bewegt. Dabei ist die Schlagbewegung nicht linear, sondern kreisförmig. Beobachtet man die Geschwindigkeitswerte in Abhängigkeit von der Zeit, so ist festzustellen, das der Schläger sein maximale Geschwindigkeit zu dem Zeitpunkt erreicht, an dem der Schläger den Ball trifft.

Abschlussphase:
Die Abschlussphase beginnt, sobald der Ball mit dem Schläger getroffen und auf dem Rückweg zum Gegenspieler ist. Aus dieser Phase lassen sich zwei Teilphasen abstrahieren. Zum einen wird der Schläger in einer Nachschwungbewegung zum Stillstand gebracht. Zum anderen richtet sich der Spieler lateral am Spielfeld aus, um anschließend wieder die Ausgangsposition einzunehmen. Je nach Intensität des Spiels dauern die Phasen, in denen sich der Spieler am Spielfeld ausrichtet, länger oder kürzer.

Besonders auffällig bei der Ausführung ist, dass professionelle Spieler nur wenige Freiheitsgrades des Arms nutzen. Zudem sehen die Schlagbewegungen trotz unterschiedlicher Ballflugbahnen sehr ähnlich aus. Die Autoren stellen aus ihren Beobachtung die Hypothese auf, dass jede Schlagbewegung nur eine leicht angepasste Form einer Grundbewegung sei, die durch Training in ihrem Muskelgedächtnis verankert seien. Diese Reihe an Basisbewegungen werden auch als *Motor Primitives* oder *Motion Primitives* bezeichnet.

3.2 Stossdynamik

Der Kernbestandteil des Tennisspiels ist das Rückspielen eines Balles, indem es durch Kraftzugabe eine Geschwindigkeits- und Richtungsänderung erfährt, sodass er durch die Luft gleitet und in der anderen Spielhälfte wieder auf den Boden auftrumpft. Dabei wird zwischen der linearen Geschwindigkeit $\vec{v} \in \mathbb{R}^3$ in $\frac{m}{s}$ und der Winkelgeschwindigkeit $\omega \in \mathbb{R}^3$ in $\frac{rad}{s}$ unterschieden. $\vec{v}$ beschreibt die Positionsveränderung des Balls in einem Zeitintervall in dem vorgegebenen Koordinatensystem, in dem sie angegeben wird. Die Winkelgeschwindigkeit ist ein Maß für die Rotation innerhalb eines Zeitfensters um die Koordinatenachsen des grundlegenden Koordinatensystems.

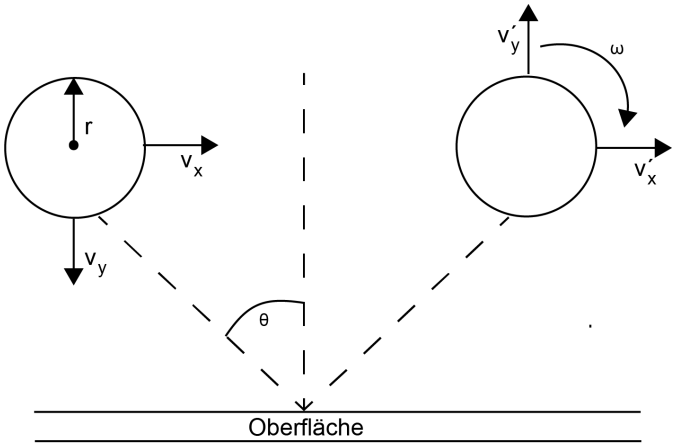


Abbildung 3.2: Visualisierung des Abpralls eines Balls von einer Oberfläche. (Eigene Abbildung nach Cross [Cro10])

Durch den Schlag wird mittels des Schlägers eine Kraft auf einen Tennisball ausgeübt, die in einer Impulsänderung des Balls resultiert. In der Physik wird dieser Vorgang als Stoß bezeichnen. Dabei sind mindestens zwei Körper mit den Massen m_1, m_2 und Geschwindigkeit v_1, v_2 in Bewegung und kollidieren. Am Punkt der Kollision lässt sich eine Tangentialebene konstruieren, die *Berührebene* genannt wird. Die dazugehörige Ebenennormale wird als *Stoßlinie* bezeichnet. Treffen sich die Körper nicht entlang der gleichen Stoßlinie, so spricht man von einem *schiefen Stoß* .
Nach dem Impulserhaltungssatz bleibt der Impuls $p = m \cdot v$ in einem geschlossenen System konstant. Für zwei Körper gilt:

$$m_1 \cdot v_1 + m_2 \cdot v_2 = m_1 \cdot v_1' + m_2 \cdot v_2' \tag{3.1}$$

Es wird zwischen zwei Grenzfällen unterschieden. Im Falle eines *ideal elastischen Stoßes* bleibt die kinetische Energie zwischen den kollidierenden Körpern erhalten. Sie stoßen sich voneinander ab. Im Gegensatz dazu wird bei einem *ideal plastischen Stoß* die gesamte kinetische Energie in andere Energieformen umgewandelt. In diesem Fall stoßen sich die Objekte nicht voneinander ab und bleiben zusammen.
Beim Tennisspiel sind zwei Stoßvorgänge von entscheidender Bedeutung: die Kollision zwischen Ball und Spieluntergrund (beispielsweise der Tisch beim Tischtennis oder der Platz beim regulären Tennis) und die Reflexion des Balls am Tennisschläger. Bei diesen Prozessen geht ein Teil der Bewegungsenergie des Balls in Form von innerer Energie verloren. Somit lässt sich diese Kollision als eine Mischform von elastischem und plastischem Stoß betrachten, welche auch als *Realstoß* bezeichnet wird.
Zunächst wird die Krafteinwirkung entlang der Normalen der Berührebene, auch *Normalkraft* genannt, betrachtet. Ein wichtiger Wert bildet dabei die sogenannte Stoßzahl e, auch *Restitutionskoeffizient* genannt. Dieser Wert kann zu Vereinfachung von Kollisionsmodellen verwendet werden, indem der Verlust von kinetischer Energie des Balls aufgrund von Reib- und Haftungseffekten oder Kompressionsvorgänge mittels eines numerischen Faktors

quantifiziert wird. e beschreibt das Verhältnis der Annäherungsgeschwindigkeit $v_1,\ v_2$ zu der Reflexionsgeschwindigkeit $v_1',\ v_2'$ zweier Objekte [HI12].

$$e = \frac{v_2' - v_1'}{v_1 - v_2} \tag{3.2}$$

Der Restitutionskoeffizient kann durch einen Falltest experimentell bestimmt werden. Sei angenommen, dass ein Ball mit Masse m_b in Kilogramm aus Höhe h in Metern auf eine ruhende, ebene Fläche fallengelassen wird. Die Geschwindigkeit beim Aufprall sei v_b. Die maximale Höhe nach der Reflexion sei h', die mit der Reflexionsgeschwindigkeit v_b' erreicht wurde. Es wird angenommen, dass die Masse der Fläche gegen unendlich geht und die Geschwindigkeitsdifferenz des Fläche vor und nach dem Aufprall somit gegen Null geht.

$$e = \frac{v_b' - 0}{0 - v_b} = \frac{v_b'}{v_b} \tag{3.3}$$

Mithilfe des Energieerhaltungssatzes lässt sich die Annäherungs- und Reflexionsgeschwindigkeit in Relation zur Fallhöhe bestimmen (vgl. [HI12], S. 3):

$$\begin{aligned} v_b &= \sqrt{2gh} \\ v_b' &= \sqrt{2gh'} \end{aligned} \tag{3.4}$$

wobei g die Erdbeschleunigung repräsentiert. Die Änderung der potentiellen Energie ΔE_{pot} ergibt sich durch

$$\Delta E_{pot} = mg(h' - h) \tag{3.5}$$

Dadurch kann der Restitutionskoeffizient e wie folgt in Beziehung zur Abwurf- und Relexionshöhe berechnet werden:

$$e = \sqrt{\frac{h'}{h}} \tag{3.6}$$

Ein entscheidender Faktor bei der Größe des Koeffizienten bildet das Material der kollidierenden Körper. Tendenziell gilt, dass Körper aus starren Materialien wie Stein oder Stahl einen geringeren Restitutionskoeffizienten aufweisen als elastische Materialien wie Gummi. Beim Tennis gibt es aus diesem Grund verschiedene Spieluntergründe (Hartgummi, Kies, etc.), die den *Rebound* des Balls beeinflussen.

Darüber hinaus erfährt der Ball im Falle eines schiefen Stoßes eine Veränderung der Winkelgeschwindigkeit ω. Diese entsteht, wenn eine Kraft nicht auf den Masseschwerpunkt des Balls wirkt. Solche eine Kraft wirkt im Tennisspiel sowohl beim *Rebound* des Balls auf dem Spielfeld als auch bei der Reflexion mit dem Schläger, wenn dieser eine Geschwindigkeit entlang der Oberfläche hält (vgl. 3.2). Bei diesem Prozess wirkt die Reibungskraft, die abhängig vom *Reibungskoeffizient* μ ist, auf die Außenseite des Balls und lässt ihn rotieren. Wirkt auf den Ball vor der Kollision bereits eine Rotationsgeschwindigkeit, so verändert sich die lineare Geschwindigkeit des Balls nach der Kollision lotrecht zur Rotationsachse.

3.3 Ballflugdynamik

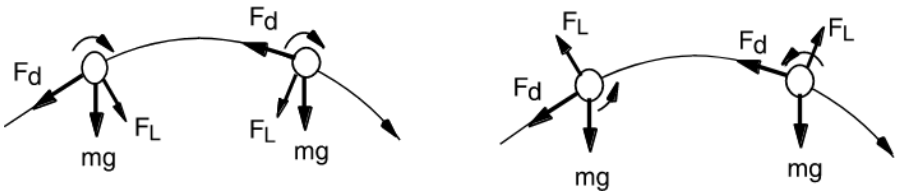


Abbildung 3.3: Visualisierung der Kräfte, die während des Fluges durch die Luft auf einen Ball wirken. links: durch eine Vorwärtsrotation wirkt die Auftriebskraft nach unten. rechts: durch die Rückwärtsrotation wirkt die Antriebskraft nach oben. [BCL02]

Während des Fluges wirken stetig Kräfte auf den Ball, die Einfluss auf seine Flugbahn haben [BCL02]. Zunächst wirkt Gravitationskraft in Form einer stetigen Beschleunigung von $g \approx 9.81 \frac{m}{s^2}$ des Balls lotrecht Richtung Spielfeld. Diese macht das Ballspiel überhaupt erst möglich, da der Ball ohne sie niemals wieder Richtung Erde fliegen würde. Für die wirkende Kraft auf einen Körper mit Masse m_b gilt nach Newton'schem Axiom:

$$F_g = m_b \cdot g \tag{3.7}$$

Wenn ein Projektil durch die Luft gleitet, wirkt eine Kraft entgegen der Flugbahn, da der Luftdruck vor dem Projektil höher ist als dahinter. Aufgrund dessen wird das Projektil verlangsamt. Diese Kraft wird auch als Zugkraft (engl. *drag force*) bezeichnet. Sie ist sowohl abhängig von der Dichte des Mediums, in dem sich der Projektil befindet, als auch der Geschwindigkeit des Projektils. Für die Berechnung der Zugkraft F_D für einen gespielten Tennisball durch Luft gilt:

$$F_D = C_D \cdot A \cdot d \cdot \frac{v^2}{2} \tag{3.8}$$

wobei C_D der Strömungswiderstandskoeffizient $d \approx 1.31 \frac{kg}{m^3}$ die Dichte von Luft, $A = \pi \cdot r^2$ der Durchmesser des Balls mit Radius r und der lineare Geschwindigkeit v ist. Da sich die Zugkraft proportional zur quadrierten Geschwindigkeit verhält, bremsen schnelle Bälle auch rasanter ab als langsame Bälle.

Wie in Kapitel 3.2 erläutert, ist neben Richtungs- und Geschwindigkeitsänderung des Balls auch die Änderung der Winkelgeschwindigkeit ω durch Stoßvorgänge eine spielentscheidende, physikalische Mechanik des Tennisspiels. Sollte der Ball sich drehen, so spielt die Auftriebskraft F_L (engl. *lift force*) eine wichtige Rolle im Flugverhalten. Der Ball wird, je nach Richtung der Rotation, entweder Richtung Boden gedrückt, angehoben oder in eine Links- beziehungsweise Rechtskurve gelenkt. Dieser Effekt wird als *Magnuseffekt* bezeichnet. Dabei wirkt die Kraft stets orthogonal zur Zugkraft und der Rotationsachse

des Balls. F_L wird bestimmt durch

$$\begin{aligned} F_L &= C_L \cdot A \cdot d \cdot \frac{v^2}{2} \\ C_L &= \frac{1}{2 + v/v_{spin}} \\ \text{mit } v_{spin} &= r \cdot \omega \end{aligned} \tag{3.9}$$

wobei C_L der Auftriebskoeffizient ist, welcher maßgeblich von der Rotationsgeschwindigkeit beeinflusst wird. Tennisspieler machen sich den Magnuseffekt zunutze, um die Ballflugbahn zu beeinflussen (vgl. Abbildung 3.3). Dazu wird der Ball mit dem Schläger entweder oben (*Topspin*), unten (*Backspin*) oder seitlich mit der Schlägerfläche angeschnitten. Ein Topspin führt dazu, dass der Ball schneller Richtung Spielfeld fällt. Ein Backspin hat eine Verlängerung des Ballflugs zur Folge, da der Ball während des Fluges angehoben wird. Beim seitlichen Anschneiden fliegt der Ball eine Kurve.

3.4 Vorwärtskinematik

Ein robotischer Manipulator besteht aus mehreren, im Idealfall starren Achsen (engl. *links*), welche mithilfe von Gelenken (engl. *joints*) miteinander verbunden sind. Als Endeffektor bezeichnet man dabei das Ende der letzten Achse, an dem sich ein Werkzeug (Greifer, Schweißgerät, etc.) befindet. Das Werkzeug übt letztendlich einen „Effekt“ auf die Umwelt aus.
Pro enthaltenem Gelenk einer manipulatorischen Kette spricht man von einem Freiheitsgrad (engl. *DOF*). Der Freiheitsgrad einer solchen Kette wird von der Art der Gelenke beeinflusst: Schubgelenke (engl. *prismatic joint*) verändern die Position des Endeffektors translatorisch, während Drehgelenke (engl. *revolute joint*) einen rotatorischen Einfluss haben. Da alle Achsen miteinander verkettet sind, ist es naheliegend, dass die Position einer Achse maßgeblich von der Stellung der vorherigen Gelenke beeinflusst wird.

Um diesen Zusammenhang darzustellen, ist es notwendig, die Achsen- und Gelenkpositionen mithilfe von geometrischen Transformationen innerhalb eines festgelegten Koordinatensystems zu beschreiben. Bei Manipulatoren sind zwei Arten von Koordinatensystemen zu unterscheiden. Zum einen gibt es ein Raumkoordinatensystem, welches häufig in Form eines kartesischen Koordinatensystems in Erscheinung tritt. Der Ursprung dieses Koordinatensystems wird häufig auf an die Stelle gesetzt, an der das erste Glied des Manipulators mit der Welt verbunden ist (beispielsweise Befestigung am Tisch). Oft spricht man dabei auch vom *Baselink*.
Auf der anderen Seite gibt es ein Gelenkkoordinatensystem. Dieses gibt den Rotations- bzw. Schubwert eines Gelenks relativ zu seiner Nullposition an. Jedes Gelenk wird dabei durch ein eigenes Gelenkkoordinatensystem beschrieben. Jedes Gelenk ist generell mechanisch limitiert, sodass es für die Gelenkkoordinaten Ober- und Untergrenzen (engl. *lower bounds* bzw. *upper bounds*) gibt.

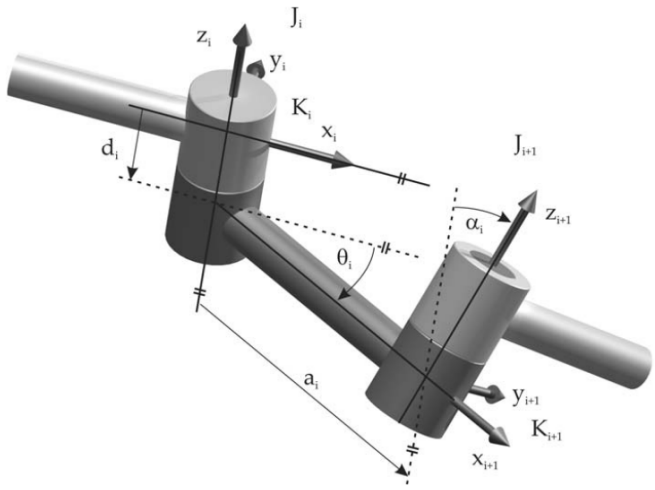


Abbildung 3.4: Visualisierung der Denavit-Hartenberg-Parameter an zwei Rotationsgelenken. [BB09]

Um nun eine Beziehung zwischen der Gelenkkoordinaten und der Pose des Endeffektors im kartesischen Raum herstellen zu können, ist es notwendig, die Achsen- und Gelenkposen in Abhängigkeit der einzelnen Gelenkpositionen zu beschreiben. Dies kann durch eine Aneinanderreihung von homogenen Transformationsmatrizen geschehen. Diese mathematische Beschreibung ist als kinematische Funktion oder auch *Vorwärtskinematik (VK)* bekannt. Eine weit verbreitete Notation ist die Denavit-Hartenberg-Konvention [DH55]. Um eine Achse kinematisch zu definieren, werden vier Parameter benötigt (Notation nach [BB09], S.159-160):

- θ_i: Rotation im Uhrzeigersinn um Winkel zwischen Achse X_i und X_{i+1} um Achse Z_i
- a_i: Abstand zwischen Z_i und Z_{i+1} entlang X_i
- d_i: Abstand zwischen X_i und X_{i+1} entlang Z_i
- α_i: Rotation im Uhrzeigersinn um Winkel zwischen Achse Z_i und Z_{i+1} um Achse X_{i+1}

Mithilfe dieser vier Parameter lässt sich Gelenk $j+1$ relativ zu Gelenk j durch jeweils zwei Translationen sowie Rotationen in einer homogenen Transformationsmatrix ${}^{i-1}T_i \in \mathbb{R}^{3\times4}$ beschreiben:

$$\begin{aligned} {}^{i-1}_{i}T &= Rot(Z_i, \theta_i) \cdot Trans(0,0,d_i) \cdot Trans(a_i,0,0) \cdot Rot(X_{i+1}, \alpha_i) \\ &= \begin{bmatrix} \cos\theta_1 & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.10}$$

Werden diese Parameter nun für jedes Gelenk eines *n-DOF*-Manipulators bestimmt, so

lässt sich die Pose des Endeffektors ${}^{0}_{EE}T$ im Basiskoordinatensysten durch die Multiplikation der einzelnen Transformationsmatrizen ermitteln:

$$ {}^{0}_{EE}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot ... \cdot {}^{n-1}_{n}T \cdot {}^{n}_{EE}T \tag{3.11} $$

Diese Berechnung soll künftig als parametrisierte Funktion $FK(q)$ bezeichnet werden, wobei $q \in \mathbb{R}^n$ eine gesamte Gelenkkonfiguration des Manipulators darstellt.

3.5 Jacobimatrix

Während die *VK* eine Beziehung zwischen Gelenk- und Endeffektorpose herstellt, kann auch der Einfluss der Gelenkgeschwindigkeit auf die Endeffektorgeschwindigkeit abgebildet werden. Dieser Zusammenhang wird durch die *Jacobimatrix* dargestellt.
Sei $\dot{P} = (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z)^T \in \mathbb{R}^6$ ein Vektor, der die Linear- und Winkelgeschwindigkeit des Endeffektors im Weltkoordinaten darstellt und $\dot{q} = (q_1, ..., q_n)^T \in \mathbb{R}^n$ die Geschwindigkeit der Gelenke eines *n-DOF*-Manipulators. Dann besteht folgender Zusammenhang:

$$ \dot{P} = J(q) \cdot \dot{q} \tag{3.12} $$

wobei $J(q) : \mathbb{R}^n \mapsto \mathbb{R}^{6 \times n}$ die dazugehörige Jacobimatrix ist.
Geschwindigkeit ist die erste Ableitung der Positionsfunktion nach der Zeit. Somit lässt sich die Jacobimatrix zu einer Hälfte als Matrix der partielle Ableitungen der Vorwärtskinematikfunktion nach den Positionvariablen auffassen [Kri].

$$ \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & ... & ... & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & ... & ... & \frac{\partial y}{\partial q_n} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & ... & ... & \frac{\partial z}{\partial q_n} \end{pmatrix} \tag{3.13} $$

In q_1 ist die Rotation des Gelenks in dessen Koordinatensystem um die lokale Rotationsachse angegeben. Zur Umrechnung ist es nun von Vorteil, wenn die *Denavit-Hartenberg*-Konvention (siehe Kapitel 3.4) zur Beschreibung der kinematischen Kette verwendet wurde, da Rotationen dort immer um die z-Achse erfolgen. Die rotatorische Geschwindigkeitsveränderung $\hat{\omega}_i = (\omega_x, \omega_y, \omega_z)^T \in R^3,\ i \in 1, ..., n$ ergibt sich aus der Rotationsmatrix ${}^{0}R_n$ des i-ten Gelenks in Gelenkkoordinaten in Relation zum Basiskoordinatensystem 0.

$$ \hat{\omega}_i = {}^{0}R_i \cdot (0, 0, 1)^T \tag{3.14} $$

Werden beide Matrizenteile zusammengefügt, so ergibt sich der Einfluss auf die lineare Geschwindigkeit bei minimaler Gelenkpositionsänderung aus den oberen drei Zeilen der Matrix. Die restlichen Zeilen spiegeln den Einfluss auf die rotatorische Geschwindigkeit

wider. Pro Gelenk enthält die Jacobimatrix eine Spalte.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \dots & \dots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \dots & \dots & \frac{\partial y}{\partial q_n} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \dots & \dots & \frac{\partial z}{\partial q_n} \\ \hat{\omega}_1 & \hat{\omega}_2 & \dots & \dots & \hat{\omega}_n \end{pmatrix} \cdot \begin{pmatrix} \dot{q_1} \\ \dot{q_2} \\ \dots \\ \dot{q_n} \end{pmatrix} \tag{3.15}$$

Da die Jacobimatrix die partiellen Ableitung bei einer vorgegebenen Gelenkposition darstellt, ist diese Matrix auch nur für diesen Zustand gültig. Im Falle einer Veränderung der Position muss auch die Jacobimatrix neu aufgestellt werden.

3.6 Inverse Kinematik

Eine *Inverse Kinematik (IK)* bildet eine Endeffektorpose x auf einen Gelenkwinkelzustand q^* ab und bildet somit das Gegenstück der Funktion der Vorwärtskinematik.

$$q^* = FK^{-1}(P_{EE})$$

Sinn und Zweck der Bestimmung der *IK* ist es, eine Zielkonfiguration q^* der Gelenke zu finden, sodass sich der Endeffektor an einer vorgegebenen Pose $P_{EE} \in \mathbb{R}^{4\times4}$ befindet.
Gerade bei einfachen, kinematischen Systemen mit wenigen Freiheitsgraden kann dieses Problem analytisch gelöst werden. Bei analytischen Verfahren werden geometrische Besonderheiten des robotischen Systems ausgenutzt, um eine trivial lösbare Funktion für die Vorwärtskinematik zu erhalten. So lässt sich beispielsweise die Position eines Plotters, der mit drei prismatisch Gelenken konstruiert wurde, durch simple Verschiebungen auf der X-, Y- und Z-Achse beschreiben.
Für Manipulatoren mit höheren *DOF* werden analytische Verfahren unpraktikabel, sodass auf numerische Methoden zurückgeriffen wird. Numerische Ansätze approximieren eine Lösung iterativ so lange, bis eine ausreichende Näherung an das Zielergebnis erreicht oder aber beispielsweise ein Zeitlimit überschritten wurde. Je nach Komplexität des Manipulators kann die Laufzeit der Algorithmen rasant ansteigen. Ein populärer Ansatz zur Lösung des Problems ist die Nutzung der inversen Jacobimatrix. In diesen Ansätzen werden die Veränderungen der Endeffektorpose relativ zu akuten Änderungen in der Gelenkkonfiguration linear modelliert [AL09].
Bei redundanten, robotischen Manipulatoren müssen bei der Berechnung einige Faktoren beachtet werden. So ist es möglich, dass es für eine angestrebte Pose unendlich viele Lösungen gibt. Auf der anderen Seite ist es auch möglich, dass es keine einzige valide Lösung gibt, da der Manipulator mechanisch limitiert, die Zielpose nicht im Arbeitsbereich des Roboters liegt oder eine Singularität vorliegt. Singularitäten treten dann auf, wenn ein Manipulator in seiner aktuellen Konfigurationen einen Freiheitsgrad verliert und den Endeffektor somit nicht entlang einer Achse bewegen oder keine Rotation um diese

ausführen kann. Solch ein Fall liegt beispielsweise vor, wenn die Achsen der Schubgelenke eines Roboter mit zwei *DOF* kollinear verlaufen.

3.7 Kameramodell

Eine Kamera bildet eine dreidimensionale Szene auf eine zweidimensionale Bildebene durch eine Punktprojektion ab. Zur Modellierung dieses Konzeptes wird häufig das Modell der Lochkamera herangezogen. Im simpelsten Fall fällt ein Lichtstrahl geradlinig durch ein, an der Vorderseite eines Volumens befindliches Loch auf einen lichtempfindlichen Film und wird so abgebildet. Die Projektion ist dabei verkleinert und steht auf dem Kopf. Geometrisch gesprochen wird ein Punkt $P = (X, Y, Z)^T$ in kartesischen Koordinaten auf einen Punkt $p = (u, v)^T$ in Bildkoordinaten abgebildet. Z spiegelt hierbei den Abstand des abzubildenden Punkts vom Koordinatenursprung der Kamera. X und Y bilden analog dazu den vertikalen und horizontalen Abstand zur optischen Achse der Kamera. Die zweidimensionale Bildfläche befindet sich in einem Abstand f hinter dem Loch. Diese Konstante f wird als Brennweite bezeichnet. Wird die Projektionsfläche im Abstand f vor den Fokuspunkt gesetzt (vgl. 3.5), dann ist dieser Vorgang leichter vorzustellen, da die Projektion nun aus Sicht des Fokuspunkts betrachtet wird.

Aus dem Strahlensatz ergeben sich für die Abbildung folgende geometrische Zusammenhänge:

$$(u, v)^T = (f \cdot \frac{X}{Z}, f \cdot \frac{Y}{Z})^T \tag{3.16}$$

In diesem Fall werden u und v nach einem idealen Kameramodell projiziert. Eine wichtige Eigenschaft dieser Abbildung ist es, dass die Tiefeninformation des ursprünglichen Punkts verloren gehen.

In realen Kamerasystemen ist es aufgrund produktionsbedingter Umstände möglich, dass die optische Achse nicht durch das Zentrum des Bildsensors verläuft. Deshalb wird dem Modell die Verschiebungsfaktoren c_x und c_y hinzugefügt:

$$(u, v)^T = (f \cdot \frac{X}{Z} + c_x, f \cdot \frac{Y}{Z} + c_y)^T \tag{3.17}$$

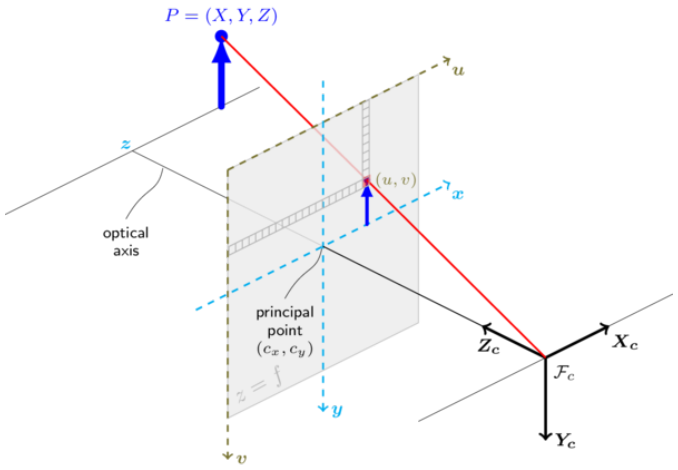


Abbildung 3.5: Visualisierung des erweiterten Lochkameramodells. [Ope19]

Das Lochkameramodell ist recht simpel und bildet die Komplexität heutiger Kameras nicht gänzlich ab. Moderne Geräte haben vor allem Linsen, die die Abbildungen optisch verzerren. Diese Verzerrung wird als *Verzeichnung* bezeichnet und tritt rotatorisch in Bezug zur optischen Achse auf [Sas94]. Des Weiteren kann sich f im Bezug auf die x- und y-Achse des Bildkoordinatensystems unterscheiden. Aus diesem Grund wird das Lochkameramodell um radiale (k_m) und tangentiale (ρ_m) Verzerrungskoeffizienten sowie einer Unterscheidung zwischen f_x und f_y nach [Ope19] erweitert:

$$\begin{aligned} \begin{pmatrix} u \\ v \end{pmatrix} &= \begin{pmatrix} c_x + f_x \cdot \hat{x} \cdot \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2\rho_1\hat{x}\hat{y} + \rho_2(r^2+2\hat{x}^2) \\ c_y + f_y \cdot \hat{y} \cdot \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + \rho_1(r^2+2\hat{y}^2) + 2\rho_2\hat{x}\hat{y} \end{pmatrix} \\ \text{mit } \hat{x} &= \frac{X}{Z},\ \hat{y} = \frac{Y}{Z},\ r^2 = \hat{x}^2 + \hat{y}^2 \\ k_{1..6} &\in \mathbb{R},\ \rho_1,\ \rho_2 \in \mathbb{R} \end{aligned} \tag{3.18}$$

Das in Gleichung 3.18 dargestellte Kameramodell setzt voraus, dass der abzubildende Punkt P in einem Referenzkoordinatensystem der Kamera vorliegt. Oft werden Messpunkte allerdings durch andere Messverfahren oder Sensorik in Relation zu einem anderen Weltkoordinatensystem zur Verfügung gestellt. Deshalb bietet es sich an, $p = (u, v, 1)^T$ durch eine Reihe von homogenen Transformationsmatrizen zu berechnen.
Sei $P_W \in \mathbb{R}^4$ ein beliebiger Punkt in einem arbiträr festgelegten, kartesischen Koordinatensystem W. Dann ergibt sich der gleiche Punkt P_C im Kamerakoordinatensystem durch die Multiplikation mit der Transformationmatrix, die Kamerakoordinaten relativ zu W beschreibt:

$$
\begin{aligned}
P_C &= {}^{W}T_{C}^{-1} \cdot P_W \\
&= \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \cdot P_W \\
&= M \cdot P_W
\end{aligned}
\tag{3.19}
$$

Die Transformationsmatrix M bildet die sogenannten *extrinsischen Parameter* einer Kamera ab.
Durch Zusammenfassen der Gleichungen 3.18 sowie 3.19 und der Überführung in homogene Koordinaten ergibt sich die sogenannte *Kameragleichung* mit der Verzeichnungsmatrix d:

$$
\begin{aligned}
\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} &= \begin{pmatrix} \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2\rho_1\hat{x}\hat{y} + \rho_2(r^2+2\hat{x}^2 \\ \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + \rho_1(r^2+2\hat{y}^2) + 2\rho_2\hat{x}\hat{y} \\ 1 \end{pmatrix} \cdot \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot M \cdot P_W \\
&= d \cdot K \cdot M \cdot P_W
\end{aligned}
\tag{3.20}
$$

Die Matrix $K \in \mathbb{R}^{4\times4}$ wird als *Kameramatrix* oder alternativ *intrinsische Kameraparameter* bezeichnet.

4 Ansatz

In diesem Kapitel wird der Gesamtablauf des Systems auf Programmebene entworfen und beschrieben. Dabei sollen zunächst die Anforderungen an das System untersucht und anschließend eine theoretische Gesamtlösung präsentiert werden.

4.1 Herausforderungen

Tennisspielen besteht aus einer Abfolge hochkomplexer Prozesse, welche mechanische und visuell-analytische Bestandteile enthält. Daraus ergeben sich einige Subprobleme, die es bei der Übertragung auf ein robotisches System zu lösen gilt:

1. **Genauigkeit:** das Gesamtsystem setzt sich aus Lösungen kleinerer Probleme zusammen. Somit ist die Korrektheit der Ausgabe eines Submoduls sehr stark abhängig von möglicherweise fehlerbehafteten Eingabewerte des vorherigen Verarbeitungsschrittes. Aus diesem Grund ist es wichtig, dass jedes Modul mit entsprechender Genauigkeit arbeitet. Kritisch ist in diesem Fall insbesondere die Bestimmung des Ballflugbahn, da eine Planung zu einer fehlerhaften Pose direkt im Verfehlen des Balls resultiert.
2. **Mechanische Limitierungen:** menschliche Spieler bewegen sich in einem kompetitiven Tennismatch stets nah an ihrer mechanischen Belastungsgrenze. Dasselbe Prinzip ist auch für die Anwendung auf einem Roboter gültig. Der Unterschied zum Menschen besteht vor allem darin, dass die Gelenke des Roboters strenge, numerisch vorgegebene Limitierungen haben, um Beschädigung an Motoren und Achsen zu verhindern. Diese Grenzen (Position, Geschwindigkeit, Beschleunigung und Ruck) müssen bei der Bewegungsplanung beachtet werden.
3. **Echtzeitfähigkeit:** Tennis ist ein sehr schnelles Spiel, in dem Entscheidungen innerhalb von wenigen Millisekunden getroffen werden müssen. Dies gilt sowohl für die Erkennung der Ballflugbahn als auch für die Bestimmung eines Abfangpunkts und Bewegungsplanung. Aus diesem Grund steht für die Lösung des jeweiligen Problems nur sehr wenig Zeit zur Verfügung. Dies führt dazu, dass Lösungen schnell berechnet und weitergegeben werden müssen. Besonders kritisch ist dabei die Berechnung der Zielkonfiguration der Gelenke, da diese zusätzlich an die jeweiligen Limitierungen der Gelenke gebunden ist.
4. **Großer Zustandsraum:** Posen von Ball und Endeffektor im kartesischen Raum sowie Gelenkwinkel und Zeit sind kontinuierliche Zustandsräume, die das Pro-

blem sehr dynamisch machen. Um gültige Lösungen für die Gelenkposition und -geschwindigkeit des Manipulators zu finden, muss entweder mit einem kontinuierlichen Zustandsraum gearbeitet werden oder aber eine sinnvolle Diskretisierung stattfinden, die den Lösungsbereich entsprechend eingrenzt.

4.2 Tracking des Balls

In Kapitel 2 wurden einige Systeme vorgestellt, die im Bereich des Ballspiels agieren. Auch wenn sie konzeptuell verschieden funktionieren mögen, gibt es eine essenzielle Komponente, die alle gemeinsam haben: die Detektion und Lokalisierung der Spielballgeschwindigkeit und -position.
Aus der Literaturrecherche hat sich ergeben, dass zweistufige bildgebende Verfahren am sinnvollsten in einer Tennisapplikation sind. Sie funktionieren robust in geschlossenen und offenen Räumen und sind preisgünstig, da für eine dreidimensionale Rekonstruktion einer Szene bereits zwei Kameras ausreichen. Darüber hinaus ist der Montageaufwand gering.

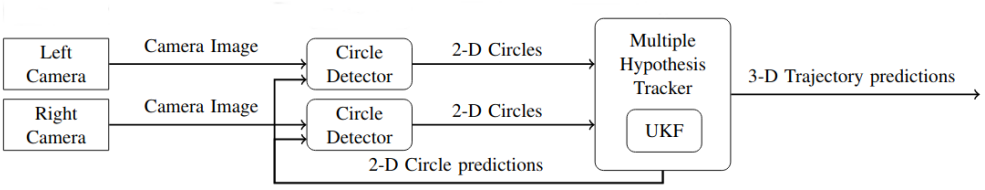


Abbildung 4.1: Aufbau der Balltracking-Komponente des Systems. (nach [BFB11])

Der verwendete Ansatz ist der von Birbach et al. [BFB11] vorgestellt *Multiple Hypothesis Tracker (MHT)*. Ursprünglich entwickelt und getestet wurde er für den humanoiden Roboter *Rollin' Justin*. *Rollin' Justin* hat die Aufgabe, einen oder mehrere Bälle zu detektieren und mit seinen beiden Greifern zu fangen. In seinem Kopf ist zu diesem Zweck ein Stereokamerasystem verbaut, welches Bilddaten für den *MHT* liefert. Da der Kopf drehbar ist, werden die Messdaten einer *Inertial Measurement Unit (IMU)* in die Vorhersage integriert. Somit eignet sich dieser Ansatz auch für bewegliche Kameraaufbauten.

Das Verfahren lässt sich in zwei Phasen einteilen. Als Eingabe für die Detektion werden zeitsynchronisierte 8-bit Graustufenbilder eines Stereokamerasystems verwendet. Da Bälle in Form eines beinah perfekten Kreises auf einem Bild abgebildet werden, werden kreisförmige Objekte aus den Bilddaten extrahiert. Anstelle der, für Kreisdetektion oft angewandten Houghtransformation wird in diesem Verfahren ein *Contrast Normalized Sobel Filter (CNS)* – ein gradientenbasierter Filter – verwendet. Der *CNS* ist in Hinsicht auf Performanz optimiert, da je nach Bildformat und Betriebsfrequenz eine große Menge Daten verarbeitet werden müssen. Aufgrund der Tatsache, dass mit hoher Wahrscheinlichkeit mehrere kreisförmige Objekte abgelichtet wurden, werden a priori Informationen zum Aussehen des Balls übergeben.

Im Sinne der Performanzoptimierung erfolgt die Suche auf verschiedenen *Stufen*. Dort werden der minimale und maximale Radius vorgegeben, die der Ball im Bild, gemessen in Pixeln, einnimmt. Zu Beginn der Messungen wird der Ball aufgrund der größeren Entfernung kleiner abgebildet, weshalb auf dem ersten Stufe nur nach Kreisen mit kleinem Radius innerhalb eines Toleranzbereichs gesucht wird. Je näher das Wurfobjekt kommt, desto größer wird es abgebildet, weshalb die Suchgröße des ersten Levels nicht mehr relevant ist und auf die nächste Stufe zurückgegriffen wird.

Wurden die Kreise in Form von Position und Radius aus dem Bild extrahiert, werden diese im nächsten Schritt an den *MHT* übergeben. Dieser entwirft – gegeben der Masse, Maße und dem daraus resultierenden, aerodynamischen Modell – eine Zustandsschätzung mit einem *UKF*. Darin enthalten sind Geschwindigkeit und -position des Balles im Koordinatensystem der linken Kamera sowie den Mittelwert und die Kovarianz der zugrundeliegenden Wahrscheinlichkeitsverteilung. Die initiale Schätzung wird zusätzlich durch eine manuell übergebene Anfangsschätzung unterstützt. Es ist beispielsweise wahrscheinlich, dass der Ball stets in Richtung der Kamera geworfen wird. Mit dieser Information können Kreismessungen, die vermutlich nicht zu dem Ball gehören, aussortiert werden. Das Modell wird mit jeder zusätzlichen Messung genauer, indem aus dem probabilistischen Modell eine Vorhersage getroffen wird, wo der Ball in der nächsten Messung abgebildet wird und darauf basierend die Zustandsschätzung aktualisiert wird.
Der Ballzustand $S \in R^{12}$ ist definiert als

$$S = (b, \dot{b}, \sigma_b, \sigma_{\dot{b}})^T$$

wobei $b \in \mathbb{R}^3$ die Ballposition in Metern entlang der x-, y- und z-Achse im Weltkoordinatensystem angibt. $\dot{b}$ ist die Lineargeschwindigkeit in $\frac{m}{s}$ pro Achse in Weltkoordinaten. $\sigma_b \in \mathbb{R}^3$ und $\sigma_{\dot{b}} \in \mathbb{R}^3$ repräsentieren die dazugehörigen Standartabweichungen der probabilistischen Verteilungen, die der Schätzung des *UKF* zugrunde liegen. Resultierend aus dem Modell kann eine Schätzung über den zukünftigen Zustand zu einem bestimmten Zeitpunkt abgegeben werden, die dann in der Bewegungsplanungskomponente verwendet werden kann.

Ein Nachteil dieses Ansatzes liegt in der fehlenden Modellierung der Winkelgeschwindigkeit ω. Um dieses Verhalten darstellen zu können, wäre es nötig, weitere Features auf dem Ball zu detektieren. Da im Rahmen dieser Arbeit aber mit einem glatten, einfarbigen Ball gearbeitet wird, sind diese Verfahren nicht nutzbar. Zudem benötigen solche Ansätze Bildaufnahmen von Hochgeschwindigkeitskameras mit hoher Verschlussgeschwindigkeit und Auflösung, um robust zu funktionieren.

4.3 Schlagsteuerung

Aufgrund der Herausforderungsanalyse und der Literaturrecherche wird ein pragmatischer Ansatz auf Basis des Verfahren von Mülling et al. [Mül+09] für die Bewegungsplanung

gewählt. In erster Linie kann dieser Ansatz online zur Berechnung von Bewegungsplänen verwendet werden und benötigt keine offline generierte Datenstruktur. Darüber hinaus ist er sehr intuitiv zu verstehen. Der enorme Nachteil dieses Verfahrens besteht darin, dass es unter Umständen nur eine lokaloptimale Lösung liefert und zu sehr unnatürlichen Bewegungsmustern führen kann.
Für die Berechnung der inversen Kinematik wird *TRAC-IK* als State-of-the-Art-Algorithmus für generische Manipulatoren verwendet. Die Optimierung der Gelenkgeschwindigkeit wird basierend auf dem Optimal-Control-Ansatz aus [KMP18] über die Lösung eines definierten Optimalitätsproblems mit Nebenbedingungen vorgenommen.

4.3.1 Bestimmung der Abfangpose und Schlaggeschwindigkeit

Aus den in Kapitel 3.1 identifizierten Bedingungen geht hervor, dass die Schlagtrajektorie die Flugbahn des Balls kreuzen muss. Daraus resultiert, dass die Abschlagspose des Endeffektors auf dieser Bahn gewählt werden muss. Eine gültige Pose beinhaltet eine Position und eine Rotation in einem festen Referenzkoordinatensystem (im Folgenden auch Weltkoordinatensystem genannt), an die der Endeffektor gesteuert werden muss. Zur Berechnung dieser Pose findet die aus dem Trackingverfahren stammende Vorhersage des Ballflugbahn Anwendung.
Die Ausgabe eines Balltrackingverfahrens ist eine parametrisierte Funktion $B(t) : \mathbb{R}^+ \mapsto \mathbb{R}^6$. Sie bildet eine Zeit in Sekunden auf den Ballzustand B_t zum Zeitpunkt t ab.

$$B_t = (b, \dot{b})^T \tag{4.1}$$

Aufgrund der Tatsache, dass die Funktion zeitabhängig ist, entsteht ein kontinuierlicher Suchraum, aus dem eine Abfangpose ausgewählt werden muss. Deshalb wird in diesem Ansatz der Zustandsraum durch *Sampling* entlang der Zeitdimension diskretisiert. Die gewählte Samplingrate i muss empirisch bestimmt werden, da eine größere Menge an Kandidaten mehr Rechenzeit in der Bestimmung der Zielkonfiguration benötigt. Darüber hinaus ist es im Hinblick auf die Realanwendung sinnvoll, ein diskretes Maximum t_{max} für die Definitionsmenge vorzugeben, da ansonsten Positionen berechnet werden, die im Boden liegen und somit nicht erreichbar sind. Die Menge I der vorläufigen Kandidaten definiert sich daraus wie folgt:

$$I := \{B(t) \mid \forall t \in \{i \cdot 0,\ i \cdot 2, ..., i \cdot n, t_{max}\}\ :\ i \cdot n \leq t_{max},\ n \in \mathbb{N}\}$$

Durch das Balltrackingverfahren können auch Ballpositionen ermittelt werden, die noch weit vom robotischen System sind. Eine weiterer Diskretisierungsschritt kann durch die Vorgabe eines Arbeitsraums (engl. *workspace*) erfolgen. Arbeitsbereiche bilden den Raum ab, in dem ein Manipulator effektiv agieren kann. Er ist somit eine Gesamtmenge der Posen, die durch die kinematische Kette beschrieben werden können. Die Bestimmung kann durch analytische Methoden [GS10] systematisch approximiert werden. Allerdings

reicht eine Annäherung des Arbeitsraum durch dreidimensionale geometrische Figuren wie Sphären oder Würfel aus, wenn diese mit einer gewissen Fehlertoleranz konstruiert werden. Daraus ergibt sich für die finale Kandidatenmenge I_f:

$$I_f := \{x \in I | \text{ Kartesische Koordinaten von x liegen im Arbeitsraum}\}$$

Nun muss aus der Menge der Ballzustände I_f eine erreichbare Pose $P_i = (R_I|T_I)$ extrahiert werden. Diese setzt sich aus einem translatorischen Teil $T_I \in \mathbb{R}^3$ und einer Rotation R_I zusammen. R_I kann dabei eine Rotationsmatrix oder aber ein Quaternion sein. Quaternionen haben den Vorteil, dass sie zur Darstellung nur vier Fließkommazahlen benötigen. Allerdings sind Rotationsmatrizen intuitiver zu benutzen.
Zunächst erfolgt die Ermittlung der Position. Die Abschlagsposition im Weltkoordinatensystem ergibt sich aus dem Ortsvektor des Balls. Allerdings muss auch der Radius der Balls bedacht werden, da der Ortsvektor die Position des Ballzentrums definiert. Somit verschiebt sich der Abschlagspunkt entlang von $\dot{b}$ um Radius r. Für T_I ergibt sich

$$T_I = b - r \cdot \frac{\dot{b}}{|\dot{b}|} \tag{4.2}$$

Für den Stoß wird das Berechnungsmodell des geraden Stoßes verwendet. Dies bedeutet, dass die Schlägernormale kollinear zur Flugbahn des Balls liegen muss. Auf diese Weise kann der Restitutionskoeffizient e zwischen Schläger und Ball zur Berechnung der nötigen Schlaggeschwindigkeit herangezogen werden. Um eine Zielrotation im Weltkoordinatensystem für den Endeffektor zu finden, kann der Ortsvektor $-\dot{b}$ als Rotation ${}^{W}_{\dot{b}}R$ dargestellt werden. Eine wichtige Eigenschaft bei der Berechnung des Rotationsanteils ist die Tatsache, dass die Rotation entlang der Schlägernormalen frei gewählt werden kann. Um sich dieses Phänomen zunutze zu machen, sollte dieser Winkel abhängig vom Aufbau des Manipulators gewählt werden. Denkbar wäre beispielsweise eine Minimalrotation des Schlägergelenks. Um eine Zielpose für das Rückhandspiel zu generieren kann die Pose entlang der Achse, die durch den Griff des Schlägers beschrieben wird, um π *rad* rotiert werden.

Zuletzt wird die benötigte Endgeschwindigkeit des Endeffektors ermittelt. In diesem Ansatz soll der Ball die ursprünglich Bahn des Balls zurückfliegen, sodass ein Richtungsspiel nicht betrachtet wird.
Für die Bestimmung der nötigen Endgeschwindigkeit sei angenommen, dass die Geschwindigkeit des Schlägers vor der Kollision identisch mit der Geschwindigkeit nach der Kollision ist. Zwar ist dies aufgrund des Energieerhaltungssatzes nicht möglich, jedoch ist dieser Umstand durch das Massenverhältnis von Ball und Schläger annähernd gegeben. Sei $\dot{b}$ die lineare Ballgeschwindigkeit vor der Kollision in $\frac{m}{s}$ und e der Resitutionskoeffizient, dann ergibt sich die benötigte, lineare Geschwindigkeit des Endeffektors $V_i \in \mathbb{R}^3$ in $\frac{m}{s}$ aus:

$$V_i = -\dot{b} \cdot (1 - e) \tag{4.3}$$

Da der gewählte Trackingansatz die Winkelgeschwindigkeit ω nicht liefern kann, wird bei diesem Ansatz auf die Berechnung einer entsprechenden Gegenrotation verzichtet.

Sind für alle Kandidaten aus eine Pose p_i und eine Schlägergeschwindigkeit v_i bestimmt, werden diese samt des dazugehörigen diskretisierten Abschlagszeitpunktes T_i für die Auswahl der besten Abfangpose verwendet.

4.3.2 Bestimmung der Zielkonfiguration

Sei $C := \{C_i \mid C_i = (p_i, v_i, T_i)^T\}$ die Menge aller Kandidaten C_i, die eine Zielpose $p_i \in \mathbb{R}^4$, eine Zielgeschwindigkeit $v_i \in \mathbb{R}^3$ und eine Ankunftszeit $T_i \in \mathbb{R}^+$ darstellen. Dann wird nun eine Gelenkkonfiguration $q^* \in \mathbb{R}^n$ für einen n-Achsen-Manipulator gesucht, sodass

$$FK(q^*) = p_i$$

Wie in Kapitel 3.6 bereits erwähnt wird mittels der Lösung des Problems der inversen Kinematik eine Pose im Weltkoordinatensystem auf Gelenkwinkel abgebildet. Aufgrund des hohen Freiheitsgrads des Manipulators bietet sich an dieser Stelle keine analytische Lösung an.
Eine populäre numerische Lösung ist die Verwendung der Pseudoinverse der Jacobimatrix, um eine Fehlerfunktion zu minimieren.

$$q_{post} = q_{pre} + J^+ \cdot (p - p^*)$$

q_{pre} ist dabei initial eine beliebige Gelenkkonfiguration, J^+ die transponierte Jacobimatrix und p, p^* die Ist- und Soll-Pose des Endeffektors. Die Anpassung der Gelenke erfolgt iterativ so lange weiter, bis die Fehlerfunktion innerhalb einer bestimmten Fehlertoleranz liegt. Aufgrund der Tatsache, dass die initiale Konfiguration einmalig zufällig gewählt wird, endet dieser Ansatz häufig in lokale Minima oder gar Singularitäten. Darüber hinaus wird keine Lösung gefunden, sobald die Konfiguration an eine mechanische Grenze stößt. Eine naheliegende Änderung zur Umgehung der lokalen ist die Einführung von zufälligen Neustarts bei trivialer Verbesserung der Fehlerfunktion.
Auf Basis der Literaturrecherche wurde *TRAC-IK* [BA15] als Lösungsalgorithmus gewählt. *TRAC-IK* verwendet den oben erläuterten Ansatz in Kombination mit einem *Sequential Quadratic Programming*-Ansatz. Dort wird das *IK*-Problem als nichtlineares Optimierungsproblem formuliert und gelöst. Durch die parallele Ausführung beider Algorithmen mittels Multithreading kann in einer durchschnittlichen Rechenzeit von $52ms$ eine optimale Lösung in über 99% bei 180.000 zufälligen Posen (vgl. [BA15]) für Manipulatoren unterschiedlichster Freiheitsgrade ermittelt werden. Optional kann *TRAC-IK* nach minimal-verändernden Gelenkkonfigurationen optimieren.
Nun kann für jeden der Posen p_i eine gültige Lösung $q^| = (q_0, q_1, ..., q_n)^T$ gesucht werden. Als Abbruchkriterium verwendet *TRAC-IK* vorrangig das Erreichen einer minimalen Fehlertoleranz zwischen Soll- und Ist-Pose des Endeffektors. Zusätzlich kann ein hartes Zeitkriterium oder Menge an benötigten Lösungen definiert werden. In Bezug auf die benötigte Echtzeittauglichkeit ist eine fest eingestellte Bedenkzeit sinnvoll.

Sollte für eine Pose p_i keine gültige Konfiguration $q^|$ gefunden werden, gilt diese als nicht verwendbar und gehört somit nicht zur Lösungsmenge Q^* der anzufahrenden Gelenkpositionen. Um aus der Zielmenge den finalen Kandidaten q^* zu bestimmen, wird als Optimalitätskriterium der Durchschnitt der Gelenkwinkeländerung zur Ausgangskonfiguration q^0 verwendet.

$$q^* = \arg\min_{q \in Q^*} \frac{1}{n} \sum_{i=0}^{n} (||q_i^0 + q_i||) \tag{4.4}$$

4.3.3 Optimierung der Gelenkgeschwindigkeit zum Abschlagszeitpunkt

Gegeben sind nun die (lokal-)optimale Gelenkkonfiguration $q^* \in \mathbb{R}^n$, welche in T Sekunden angefahren werden muss, sowie die aktuelle Gelenkposition $q^0 \in \mathbb{R}^n$. Darüber hinaus ist die zu erreichende Endeffektorgeschwindigkeit $\dot{p}$ bekannt. Allerdings ist die Gelenkgeschwindigkeit $\dot{q}^*$ noch unbekannt, die in $\dot{p}$ resultiert. Dazu findet die in Kapitel 3.5 erläuterte Jacobimatrix Anwendung. Die Beziehung von $\dot{q}^*$ und $\dot{p}$ ergibt sich durch:

$$\dot{p} = J_{q^*} \cdot \dot{q}^*$$

wobei J_{q^*} die, für die Zielkonfiguration spezifische Jacobimatrix darstellt. Aufgrund der Tatsache, dass ω in diesem Ansatz nicht beachtet wird, werden nur die ersten drei Zeilen der Jacobimatrix benötigt, die den Einfluss auf die Lineargeschwindigkeit im kartesischen Raum abbilden. Jedoch müssen die Geschwindigkeitslimitierungen der Gelenke als Nebenbedingung betrachtet werden.
Inspiriert durch Koç et al. [KMP18] kann dieser Zusammenhang als nichtlineares Optimierungsproblem mit Nebenbedingungen formuliert werden.

$$\begin{aligned} &arg\min_{\dot{q}}(\dot{p} - J_q \cdot \dot{q}) \\ &\text{so, dass } \dot{q}_{ll} < \dot{q} < \dot{q}_{ul} \end{aligned} \tag{4.5}$$

wobei $\dot{q}_{ll},\ \dot{q}_{ul} \in \mathbb{R}^n$ die Geschwindigkeitslimitierung sind.

4.3.4 Bewegungsplanung

Zum Schluss sind Zeitpunkt $T \in \mathbb{R}^+$, Zielposition $q^* \in \mathbb{R}^n$ für die Gelenke sowie die dazugehörige Geschwindigkeit $\dot{q^*} \in \mathbb{R}^n$ bekannt. Damit der Manipulator Informationen darüber erhält, in welcher Konfiguration er sich zum welchem Zeitpunkt befinden soll bzw. mit welcher Geschwindigkeit eine Positionsänderung vorgenommen werden muss, soll jedes $q_{0..n} \in \mathbb{R}$ in Abhängigkeit der Zeit t in einer parametrisierten Funktion $q_i(t) : \mathbb{R} \mapsto \mathbb{R}$ berechnet werden.

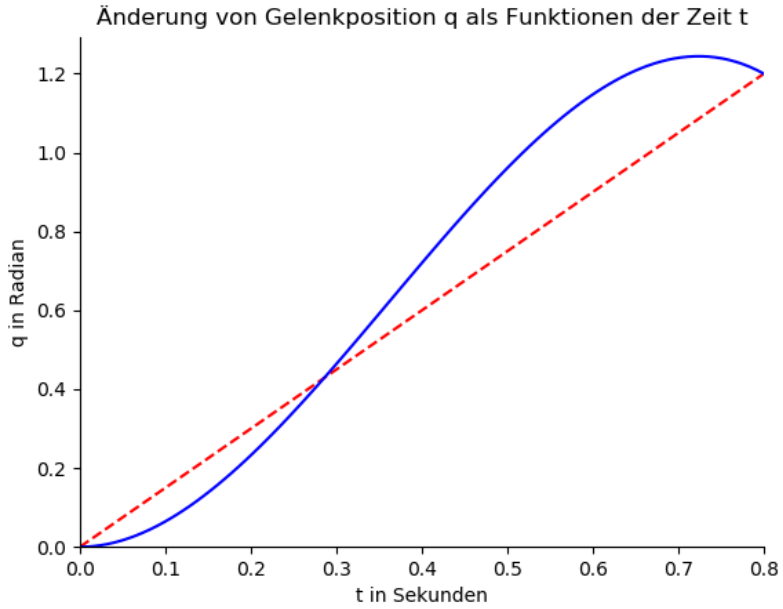


Abbildung 4.2: Visueller Vergleich der Planungsfunktionen für die Bewegungsplanung in Gelenkkoordinaten. Geplant wird zur Position $q_i^* = 1.2\ rad$ und Geschwindigkeit $\dot{q}_i^* = -1.2\ \frac{rad}{sec}$. Rot-gestrichelt: lineare Funktion (Gleichung 4.6). Blau: Polynom dritten Grades (Gleichung 4.7).

Ein naiver Ansatz wäre die Darstellung als einfach lineare Funktion:

$$f(x) = mx + b \Rightarrow q_i(t) = \frac{q_i^* - q_i}{t} + q_i \tag{4.6}$$

In diesem Fall würde der Endeffektor die Abfangpose zwar rechtzeitig erreichen und den Ball somit treffen, allerdings mit der minimal benötigten Geschwindigkeit m. Der Ball prallt folglich undefiniert vom Schläger ab. Darüber hinaus sind die starken Steigungsänderungen an $q(0)$ und $q(T)$ (vgl. Abbildung 4.2) gleichbedeutend mit einer rapiden Geschwindigkeitsänderung und der damit korrelierenden hohen Beschleunigung. In der Praxis äußern sich diese Werte in einem Ruckverhalten beim Anfahren und Abbremsen des Roboters. Im Idealfall sind solche Werte durch Sicherheitsmechanismen in der Hardware gar nicht erst möglich. Darüber hinaus bedeuten solche Beschleunigungs- und Bremsvorgänge eine enorme, mechanische Belastung für die Gelenke, welche in Beschädigungen der Hardware enden können.
Aus physikalischer Sicht ist $\dot{q}^*$ die Ableitung des Funktionswerts $q(T)$. Die Initialgeschwindigkeit des Roboters in der Ruhephase ist für alle Gelenke $q_i = 0$. Als Funktionstyp eignet sich ein Polynom dritten Grades als gute Annäherung, welches mit $q_i, q_i^*, \dot{q}_i, \dot{q}_i^*$ und T

parametrisiert wird (vgl. [KMP18], S. 123).

$$\begin{aligned} q(t) &= x_1 \cdot t^3 + x_2 \cdot t^2 + \dot{q}_i \cdot t + q_i \\ \text{mit } x_1 &= \frac{2}{T^3} \cdot (q_i^* - q_i) + \frac{1}{T^2} \cdot (\dot{q}_i + \dot{q}^*) \\ x_2 &= \frac{3}{T^2} \cdot (q^* - q_i) - \frac{1}{T} \cdot (\dot{q}^* + 2\dot{q}_i) \end{aligned} \tag{4.7}$$

Bei der Planung des Rückwegs kann eine ähnliche Funktionsapproximation $q_r(t)$ verwendet werden. Dabei sind die Anfangsgeschwindigkeit $q_r = \dot{q}(T)$ und Gelenkposition $q_r = q(T)$ gegeben. Als Endbedingung für die Ausführung muss die Startposition $q_r^* = 0\ rad$ mit der Endgeschwindigkeit $\dot{q}_r^* = 0 \frac{rad}{s}$ erreicht werden. Die Ausführungszeit T_r ist bei der Rückkehr in die Ausgangsstellung nicht mehr relevant. Deshalb soll sie in der Zeit erreicht werden, die es mit der mittleren Maximalgeschwindigkeit $\dot{q}_{ul}$ des Gelenks benötigt, sodass

$$T_r = \frac{2q_r}{\dot{q}_{ul}}$$

4.3.5 Zustandsmaschine

Der Ablauf eines Tennisschlags setzt sich aus den in Kapitel 3.1 identifizierten Phasen zusammen. Daraus kann für die Kontrollarchitektur eine Zustandsmaschine (vgl. Abbildung 4.3) zur Ausführung der Trajektorie konstruiert werden.

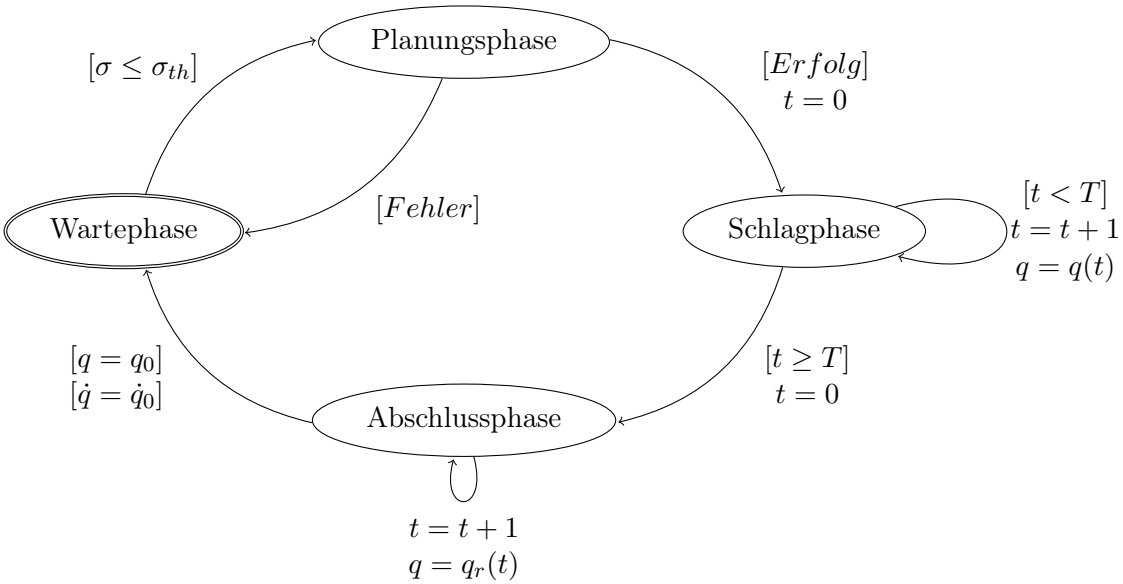


Abbildung 4.3: Zustandsmaschine für den Positionscontroller.

Beim Start soll in der Wartephase begonnen werden. Sobald die Standardabweichung σ des Balltracks kleiner als ein fest gewählter Toleranzwert σ_{th} ist, kann mit der Planung begonnen werden. Liefert eine Planungskomponente keine gültige Lösung, findet eine Transition zurück in den Wartezustand statt. Bei gültiger Planung kann der Plan auf dem Controller mittels der gelieferten Planungsfunktion $q(t)$ ausgeführt werden. Als Abbruchkriterium für diesen Zustand dient das Überschreiten des Abschlagszeitpunktes T. In diesem Fall wird die Rückkehrfunktion $q_r(t)$ ausgeführt, bis q und $\dot{q}$ den Ausgangswert $\dot{q}_0$ und q_0 erreichen. Unter dieser Bedingung findet ein Übergang in den Wartezustand statt und der Zyklus beginnt von Neuem.

5 Hardwarebeschreibung

In diesem Kapitel sollen die Hardwarekomponenten präsentiert werden, die im Rahmen dieser Arbeit verwendet werden.

5.1 Robotersystem *Panda*

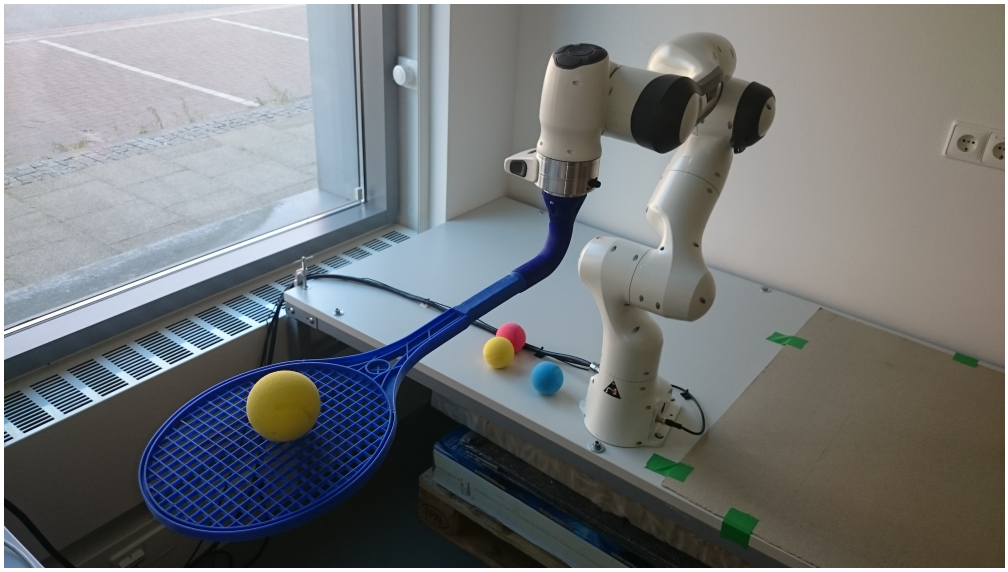


Abbildung 5.1: Manipulator *Panda* der Firma *Franka Emika* mit montierter Halterung samt Schläger.

Der *Panda* ist ein robotischer Manipulator der Firma *Franka Emika*. Mit seinen sieben *DOF* in Form von Rotationsgelenken gehört er zu der Klasse der Knickarmroboter. Konzipiert wurde er für Instrustrie-4.0-Anwendungen, in denen die Mensch-Maschine-Kooperation im Vordergrund steht. Aufgrund von TeachIn-Funktionalitäten und einer grafischen Oberfläche zur Programmierung einfacher Automatisierungsanwendungen erfreut sich der *Panda* ebenfalls großer Beliebtheit im Bereich der Lehre. Für *MoveIt!* – einem Plug-and-Play-Framework zur Ansteuerung von robotischen Manipulatoren – wird der *Panda* als Referenzsystem verwendet[1]. Doch auch in der Fachliteratur ist diese Roboterplattform zur Evaluation von Anwendungen vorzufinden.

[1] Das Quickstart-Tutorial orientiert sich vollständig am Modell des Panda. Quelle: `http://docs.ros.org/melodic/api/moveit_tutorials/html/index.html`, aufgerufen am 20.06.20

Als Standardwerkzeug bietet die Firma einen Zwei-Finger-Greifsystem an, das eine Nutzlast mit einem Gewicht von bis zu drei Kilogramm heben kann. Auf diese Weise sind beispielsweise Pick-and-Place-Routinen denkbar. Der Greifer ist modular entworfen, sodass er jederzeit an- und abmontiert werden kann. So können eigene Werkzeuge an den Flange des Roboters befestigt werden.
Zur Ansteuerung verwendet der *Panda* einen Drehmomentregler. Jedoch ist die softwareseitige Ansteuerung auch über Positions- oder Geschwindigkeitswerte für die Gelenke möglich, da mit diesen intuitiver zu arbeiten ist. Darüber hinaus ist es möglich, kartesische Bewegungsmuster vorgegeben, die dann von einer internen Planungkomponente in Drehmomentwerten umgesetzt werden. Dazu wird ein Softwarepaket – das *Franka Control Interface (FCI)* – bereitgestellt. Dieses Paket bietet unter Anderem eine Bibliothek namens *libfranka* für Steuerungsfunktionalitäten in C++-Programmen sowie eine Schnittstelle zu den gängigsten *Robot Operating System (ROS)*-Hardwareinterfaces an.
Verbunden wird das System mittels Ethernet und erlaubt darüber via *UDP* die Übertragung von Befehlen in Echtzeit. Dies setzt allerdings die Verwendung eines präemptiven Realtimekernels voraus, um die erforderliche Übertragungsrate der Befehle von 1000 Hz zu garantieren. Als Feedback für Nutzeranwendungen werden die Ist-Position, Geschwindigkeit, Drehmoment, Inertialmoment und Jacobimatrix des Ist-Zustands sowie den Zeitpunkt $t + 1$, für den eine Soll-Konfiguration erwartet wird, zurückgegeben.
Um Schäden an Mensch und Maschine zu verhindern, sind einige Sicherheitsmechanismen im *FCI* implementiert. So ist die Endeffektorgeschwindigkeit am Flange auf 2 $\frac{m}{s}$ limitiert. Für die Gelenke sind sowohl Positions-, Geschwindigkeits- als auch Beschleunigungs- und Rucklimitierungen definiert. Darüber hinaus sollen über ein internes Kollisionsmodell Eigenkollisionen verhindert werden.

Gelenk	a (m)	d (m)	α (rad)	θ (rad)
J_1	0	0.333	0	θ_1
J_2	0	0	$-\frac{\pi}{2}$	θ_2
J_3	0	0.316	$\frac{\pi}{2}$	θ_3
J_4	0.0825	0	$\frac{\pi}{2}$	θ_4
J_5	-0.0825	0.384	$-\frac{\pi}{2}$	θ_5
J_6	0	0	$\frac{\pi}{2}$	θ_6
J_7	0.088	0	$\frac{\pi}{2}$	θ_7
Flange	0	0.211	0	0
Schläger	0.508	0	0	0

Tabelle 5.1: erweiterte Denavit-Hartenberg-Parameter des Panda (vgl. [Fra17]).

Für das Tennisspiel wurde eine Erweiterung aus Kunststoff gefertigt, in die ein *Familytennis*-Hartplastikschläger passt (vgl. Abbildung 5.1). Die Halterung kann am Flange befestigt werden. Daraus resultiert eine Veränderung der Vorwärtskinematik, weshalb die vom Hersteller vorgegebenen Transformationsmatrizen der Gelenke um eine Achse erweitert werden muss (vgl. 5.1). Als Wurfobjekte werden werden farbige Schaumstoffbälle

verwendet. Sie haben den Vorteil, dass sie besonders leicht und weich sind, was in einer geringeren Kraftauswirkung auf das Schlaggelenk resultiert.

5.2 Kamera *DMK33UX265*

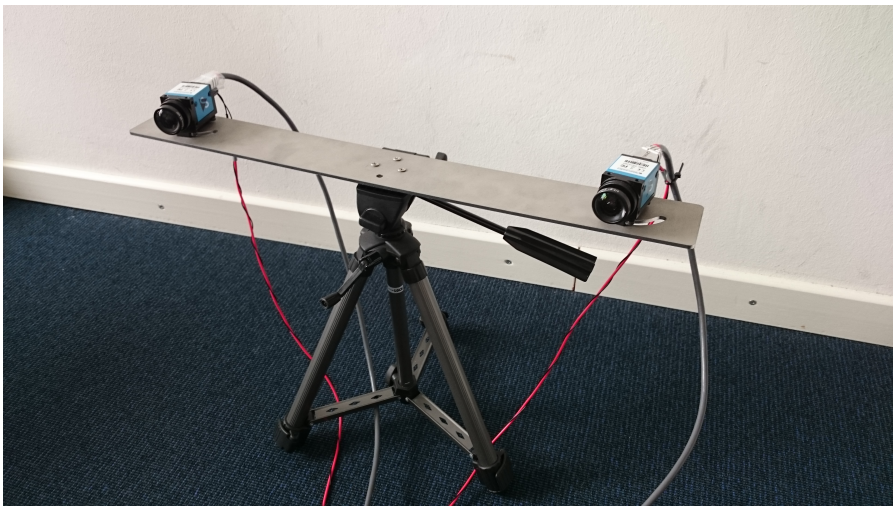


Abbildung 5.2: Die Industriekamera *DMK33UX265* in der entworfenen Stereokonfiguration.

Als Kamerahardware für diese Arbeit wurde die *DMK33UX265* der Firma *The Imaging Source* vorgegeben. Die Aufnahmen werden als 8-bit- oder 16-bit-Monochrombilder bereitgestellt. Der integrierte Sony IMX265LLR-Bildsensor arbeitet mit einem *Global Shutter*, weshalb sich das System sehr gut für industrielle Anwendungen eignet, in denen visuelle Daten schnell beweglicher Teile benötigt werden. Durch die Verwendung des *USB3-Vision*-Standards ist es möglich, große Datenmengen per USB3.0-Verbindung zu übertragen. Dadurch ergeben sich für diese Kamera Betriebsfrequenzen von bis zu 60 Bildern pro Sekunde bei einer Auflösung von maximal 2048×1536 Pixeln [The19]. Die möglichen Auflösungen und Aufnahmeraten sind abhängig von der verwendeten Treibertechnologie. Bei der Verwendung von *UVC* – der generischen Spezifikation für USB-Geräte unter Linux – muss aus einer festen Liste gewählt werden (vgl. Tabelle 5.2).

Auflösungen	mögliche Bilder pro Sekunde
2048×1536	$1, 5, 15, 30, 60$
1920×1080	$1, 5, 15, 30, 60, 90$
640×480	$1, 5, 15, 30, 60, 90, 180, 370$

Tabelle 5.2: Mögliche Betriebsfrequenzen und Auflösungen für die Kamerahardware bei einer Ansteuerung über *UVC* (vgl. [The19]).

Über einen 12-Pin-Ein- und Ausgang können externe Geräte wie Beleuchtung oder andere Kamerasysteme mittels Schaltsignal synchronisiert werden.
Die Brennweiteneinstellung muss manuell vorgenommen werden, da das externe Objektiv per Gewinde vor den Lichtsensor geschraubt wird. Damit die Brennweite nicht unkontrolliert verstellt und die intrinsische Kamerakalibrierung damit unbrauchbar wird, wurde das Objektiv mit Schraubensicherung befestigt.
Um als Stereosystem zu fungieren, wurden zwei dieser Kameras in einem Abstand von $30cm$ auf eine lasergeschnittene Edelstahlplatte montiert, die an einem handelsüblichen Kamerastativ befestigt ist (vgl. Abbildung 5.2). Über das Stativ lässt sich der Aufbau drehen und entlang der Querachse neigen und darüber hinaus leicht transportieren. Auf der Platte können die Kameras entlang der Hochachse rotiert werden, sodass der Blickwinkel verändert werden kann. Diese Funktionalität ist sinnvoll, da eine Überlappung der Sichtbereiche der Kameras nötig ist, um Ballabbildungen aus beiden Perspektiven zu erhalten.

6 Umsetzung

In diesem Kapitel wird die Umsetzung des Gesamtkonzeptes in Form von Software beschrieben. Zusätzlich wird die im Rahmen dieser Arbeit entstandene Simulationsumgebung sowie ein Kalibrierungstool für Stereokamerasysteme erläutert.

6.1 verwendete Bibliotheken und Frameworks

***ROS*:**

Das *Robot Operating System (ROS)* ist ein Framework zur Implementierung von Software zur Steuerung verteilter Robotersysteme. Da im *ROS*-Ökosystem bereits viele Module für bekannte Probleme existieren, wird es heutzutage sehr erfolgreich in Forschung und Wirtschaft eingesetzt. Die Grundidee besteht darin, dass Funktionalitäten in Form von Modulen realisiert, diese in einzelnen Prozessen gestartet und Daten über die von *ROS* bereitgestellten Kommunikationsmechanismen ausgetauscht werden.
Als Entwicklungssprache kann sowohl C++ als auch Python verwendet werden. In dieser Arbeit wird vorrangig C++ verwendet, da zum einen alle Module echtzeitfähig sein müssen. Zum anderen sollte C++ verwendet werden, um den Treiber der Kameras nativ ansprechen zu können.
Im Zentrum des *ROS*-Designs stehen die eigenständigen Anwendungen, auch *Nodes* genannt. Nodes erfüllen im Idealfall nur eine einzige Aufgabe und kommuniziert die Resultate an andere Nodes weiter, sodass ein Gesamtprozess durch einen modularen Aufbau entsteht. Die Verwaltung der Nodes obliegt dem *ROS-Master*, welcher Informationen über laufende Nodes und ihre bereitsgestelleten Kommunikationswege liefert. Zur Kommunikation stellt *ROS* diverse Mechanismen bereit.

- **Topics:** Die Kommunikation über Topics ist auch als *Subscriber-Publisher-Modell* bekannt. Dabei fungiert eine Node als Publisher und stellt zyklisch Daten auf einem sogenannten Topic bereit, deren Identifikation mittels symbolischer Bezeichner über den *Master* stattfindet. Subscriber-Nodes haben die Möglichkeit, sich bei diesen Topics als Client anzumelden, sodass sie neu gesendete Daten erhalten. Die Daten werden im Subscriber in einer Callback-Methode verarbeitet. Die Bereitstellung der Daten erfolgt asynchron über TCP-Verbindungen.
- **Services:** Durch einen *Remote Procedure Calls (RPC)* einer Node auf eine Andere wird ein Datum als Ergebnis synchron zurückgeliefert. Eine Liste der zur Verfügung stehenden Services wird im Master verwaltet.

Ein entscheidender Nachteil bei *ROS* liegt in dem schlecht überschaubaren Laufzeitver-

halten von Nodes. Raceconditions, die durch asynchrone Datenübertragungen entstehen, sind schwer zu identifizieren, weshalb sich das Basiskommunikationsprotokoll nicht für Echtzeitanwendungen eignet. Eine Lösung für diese Problem sind *Nodelets*. Nodelets teilen sich einen Speicherbereich über Shared-Memory, sodass Daten ohne Kopiervorgänge und TCP-Verbindungen publiziert werden können.

OpenCV:

OpenCV ist eine quelloffene Bibliothek, die viele State-of-the-Art-Algorithmen der Bildverarbeitung implementiert. Mithilfe der bereitgestellten *Qt*-Bindings ist es möglich, *OpenCV* problemlos in Applikationen mit grafischen Benutzerschnittstellen einzubinden. In dieser Arbeit wird diese Bibliothek für die intrinsische und extrinsische Kalibrierung der Kameras verwendet.

V4L2:

Video for Linux Version 2 (V4L2) ist eine Programmierschnittstelle[1] für den Zugriff auf Videogeräte wie USB-Webcams. *V4L2* bietet eine *Application Programming Interface (API)* zur Konfiguration von Kameraparametern sowie Aufnahme von Bildern und Videos via ioctl()-Routinen. Kamerahersteller haben die Möglichkeit, diese standardisierte Schnittstelle in ihren Kameratreibern zu implementieren, sodass Entwickler diese Geräte leichter in Software integrieren können. Außerdem ist es möglich, dem Nutzer individuelle Funktionen in sogenannten *User Control*-Registern zur Verfügung zu stellen.

Google Ceres:

Ceres ist eine Bibliothek der Firma Google, welche für die Lösung (nicht-)linearer Optimierungsprobleme genutzt wird. Dazu gehören beispielsweise Kalibrierungsroutinen oder Lokalisationsprobleme. Ein großer Vorteil dieser Bibliothek besteht darin, dass die Kostenfunktion automatisch abgeleitet werden kann, sodass diese lediglich korrekt entworfen werden muss. *Ceres* ist multithreadingfähig, sodass die Mehrkernarchitektur moderner Prozessoren ausgenutzt werden kann. In dieser Arbeit wird sie zur Optimierung der Gelenkgeschwindigkeiten zum Abschlagzeitpunkt verwendet.

***Kinematics and Dynamics Library (KDL)*:**

Das Problem der *VK* und *IK* ist so alt wie die Robotik selbst, weshalb hierfür bereits viele implementierte Lösungen existieren. Eine populäre quelloffene Bibliothek, welche die meisten populären Ansätze bereitstellt, ist die *KDL*. Mit *KDL* ist es möglich, sowohl Jacobimatrizen, Vorwärtskinematiken als auch inverse Kinematiken für generische Manipulatoren zu berechnen. Dazu können kinematische Ketten aus einer *Unified Robot Description Format (URDF)*-Beschreibung des Systems extrahiert und auf deren Basis mittels definierter Schnittstelle eigene *IK*-Algorithmen getestet werden. Zur Sammlung gehört auch eine Implementierung des im Ansatz ausgewählten *TRAC-IK*-Algorithmus.

[1] online unter `https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/v4l2.html`, abgerufen am 12.06.20

6.2 Schematischer Aufbau

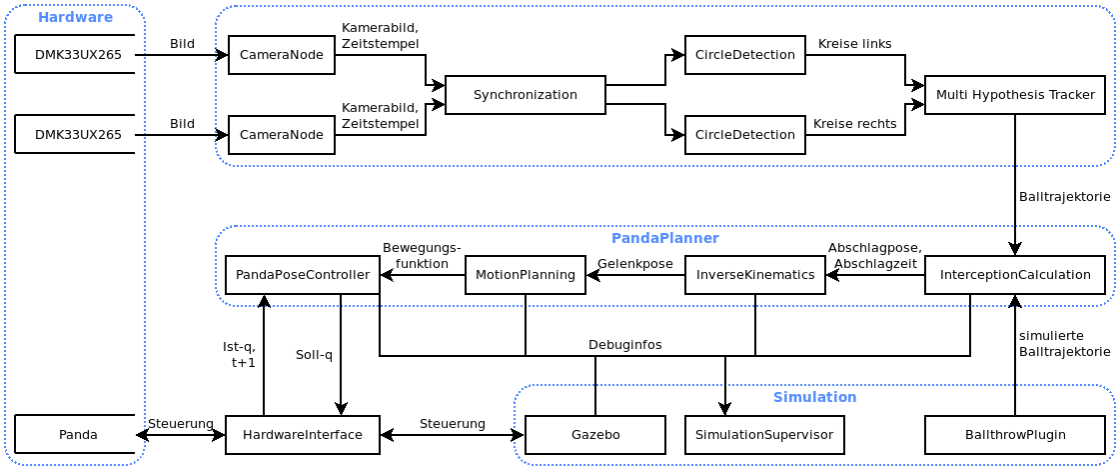


Abbildung 6.1: Überblick über die Komponenten des Gesamtsystems und den dazugehörigen Datenfluss.

Das Gesamtsystem setzt sich aus mehreren dedizierten Softwaremodulen zusammen (siehe Abbildung 6.2). Im Bereich des Balltrackings wurden *ROS*-Nodes für den Kamerabetrieb sowie eine softwareseitige Synchronisierung entwickelt (Kapitel 6.5.1 und 6.5.2). Für die Kamerakalibrierung wurde ein Programm mit Benutzeroberfläche implementiert (Kapitel 6.5.3). Der *MHT* basiert auf einer bereits vorhandenen Implementierung (6.5.4), deren Parametrisierung angepasst werden musste. Der softwareseitige Neubeitrag dieser Arbeit liegt in der Entwicklung der Simulation (Kapitel 6.3) sowie der Planungsbibliothek *PandaPlanner*, die den präsentierten Ansatz umsetzt (Kapitel 6.4).

6.3 Entwurf einer Simulationsumgebung

Ein erster, wichtiger Entwicklungsschritt ist das Erstellen einer Simulationsumgebung. Eine solche Simulation hat den Vorteil, dass zum einen die Nutzungsdauer des realen Systems minimiert und es zum anderen Schäden bei fehlerhaftem Verhalten während des iterativen Entwicklungszyklus der Module verhindert werden. Aus der großen Auswahl an Simulationsumgebungen, die für Robotikanwendungen existieren, wurde *Gazebo* [KH04] als Simulationsframework ausgewählt. *Gazebo* bietet den großen Vorteil, dass es bereits Schnittstellen für *ROS* zur Verfügung stellt und es somit einfacher in den gesamten Softwarestack integriert werden kann. Die standartmäßig verwendete Physikengine in Gazebo ist die *Open Dynamics Engine (ODE)* [Dru+10]. Sie unterstützt diverse Arten von Gelenken sowie Reibungsmodelle und Kollisionserkennung zwischen Objekten.
Als grundlegendes Modell stellt der Hersteller den Manipulator im *URDF* bereit. Es

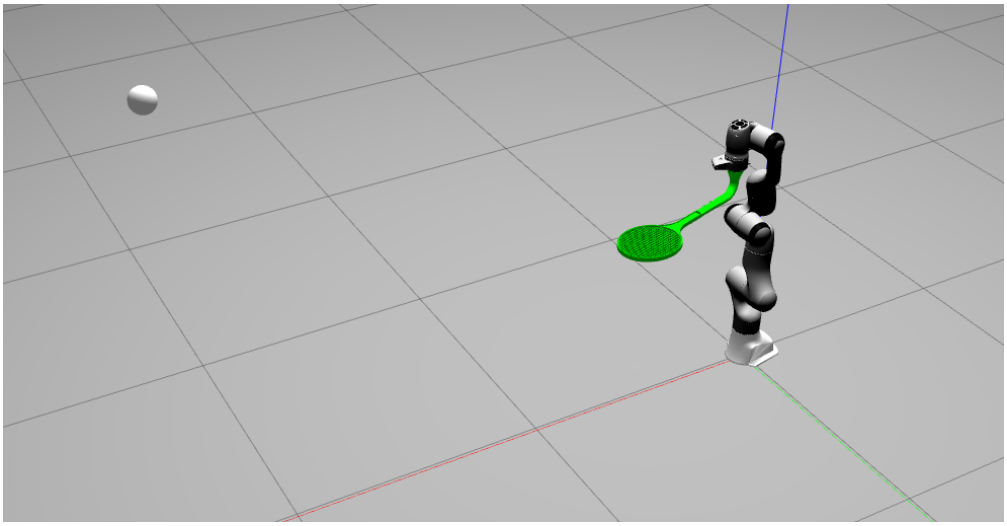


Abbildung 6.2: Der *Panda* mit Tennisschlägererweiterung in der Simulationsumgebung.

beinhaltet bereits einfache, geometrische Körper für das Kollisionsverhalten. Das vorgegebene Modell wurde um eine visuelle Komponente und einer Kollisionsgeometrie für den Schläger erweitert, um das Kollisionsverhalten zwischen Ball und Schläger zu beschreiben. Dazu wird das *Computer-aided design (CAD)*-Modell der Schlägererweiterung für den realen Manipulator in *Meshlab*[2] importiert und dort vereinfacht, um die Performanz der Kollisionsberechnung in der Simulation zu verbessern. Darüber hinaus wurde das Inertialmoment des Schlägers darüber approximiert.
Für die Durchführung von Experimenten in der Simulationsumgebung wurden zudem einige Plugins erstellt, um Ground-Truth-Daten extrahieren zu können. Dazu gehört zum einen ein Kontaktplugin für den Schläger, welches Kollisionen zwischen Ball und Schlagfläche registriert. Ein Kernaspekt der Experimentalumgebung ist das Werfen eines Balls. Da sich der Tracker nur sehr schwer in *Gazebo* simulieren lässt, werden als Eingabe für die Kontrollarchitektur die Vektoren der Ballposition $b \in \mathbb{R}^3$ und der Ballgeschwindigkeit $\dot{b} \in \mathbb{R}^3$ verwendet, die auf Basis eines vereinfachten, aerodynamischen Modells ermittelt werden.

$$\begin{aligned} b_{t+1} &= b_t + v_t \cdot \Delta t + \frac{1}{2} g \cdot \Delta t^2 \\ \dot{b}_{t+1} &= \dot{b} + g \cdot \Delta t \end{aligned} \tag{6.1}$$

$g = (0, 0, -9.81)^T$ repräsentiert dabei den Beschleunigungsvektor der Gravitationskraft. Die Zeitdifferenz zwischen den Simulationsschritten wird durch Δt dargestellt. Um einen möglichst großen Zustandsraum bei den Ballwürfen abbilden zu können, wurde ein Plugin für randomisierte Würfe entwickelt. Die Anfangsposition ergibt sich aus einer festgelegten Distanz sowie eines mittels Normalverteilung realisierten Offsets nach links und rechts.

[2] online unter `https://www.meshlab.net/`, abgerufen am 24.5.2020

Der Geschwindigkeitsvektor ergibt sich aus einer, mit der Flugzeit t parametrisierten Funktion in Abhängigkeit des Arbeitsraums, des Distanz zum Manipulator und dem zufälligen Offset.

Eine besondere Herausforderung stellt die möglichst realitätsnahe Abbildung von physikalischen Gegebenheiten und dynamischen Parametern dar. Dazu gehören beispielsweise Gelenkreibungen sowie Massen und Massenträgheitsmomente. Das *URDF*-Modell beinhaltet lediglich eine visuelle Beschreibung des Roboters. Leider stellt der Hersteller keine Informationen bezüglich der Massenträgheitsmomente sowie der Gewichte der einzelnen Achsen des Manipulators zur Verfügung. Daher werden diese Parameter von Gaz et al. [Gaz+19] übernommen, die dort durch ein Optimierungsverfahren approximiert wurden.

6.4 PandaPlanner

Zur Ansteuerung der Gelenke wird der ab Kapitel 4.3.1 vorgestellte Ansatz in einer Gesamtbibliothek namens *PandaPlanner* implementiert. Eine Klassenübersicht ist in Kapitel 6.2 zu finden. Notwendig für alle Komponenten ist die Ermittlung der benötigten Rechenzeit, damit der approximierte Abschlagszeitpunkt, der zu Beginn des Gesamtprozesses bestimmt wird, zum Steuerungszeitpunkt korrekt ist.

InterceptionCalculation:
In diesem Modul beginnt die Pipeline zur Bewegungsplanung des *Panda*. Diese Komponente übernimmt die Aufgabe des Samplings und der Berechnung der benötigten Endeffektorgeschwindigkeit. Dabei beginnt die Berechnung erst, sobald der probabilistische Fehler $sigma_b$, $sigma_{\dot{b}}$ des Balltrackings (vgl. Kapitel 4.2) unter einen konfigurierten Toleranzwert fällt. Der Threshold ist dabei abhängig von der Qualität der Balltracks des *MHT*.
Für die Samplingauflösung ist auf Basis von Softwaretests ein Wert von $25ms$ als idealer Richtwert gewählt worden. Als Arbeitsraum wurde ein $2m \times 2m \times 2.6m$ großer Würfel angenommen.
Die Rotation der Schlägerpose um die Normalenachse sollte so gewählt werden, dass das Flangegelenk nur minimal um θ gedreht werden muss. Dazu wurde die geometrische Beziehung zwischen der Nullstellung des Endeffektors R und der potenziellen Abschlagspose b ausgenutzt, sodass

$$\theta = atan2(z_b - z_R,\ y_b - y_R).$$

InverseKinematics:
Hier werden die potentiellen Abfangposen nach Erreichbarkeit gefiltert und bewertet (vgl. Kapitel 4.3.2). Dabei werden pro Sample 3 Lösungen berechnet bei einer Berechnungszeit von maximal $5ms$. Für den idealen Kandidaten q^* mit der minimalen Gelenkwinkeländerung relativ zur Startkonfiguration q_0 wird eine Gelenkgeschwindigkeit $\dot{q}^*$ zum Zeitpunkt T nach Gleichung 4.5 optimiert. Als Optimierungsbibliothek wird dafür *Ceres* eingesetzt.

MotionPlanning:
Die gegebenen Parameter T, q_0, $\dot{q_0}$, q^* und $\dot{q}^*$ werden zur Berechnung der Parameter eines Polynoms dritten Grades verwendet. Die Funktionsparameter werden an den Controller übergeben, um die benötigte Rechenzeit in der Steuerungskomponente zu minimieren.

PandaController:
In dieser Komponente findet die eigentliche Übertragung der Steuerungsbefehle statt. Es ist essenziell, dass in diesem Prozess keine komplexe Programmroutine ausgeführt wird, da eine Operationsfrequenz von 1000Hz erreicht werden muss. Sowohl für die Simulation als auch die reale Hardware können Hardwareinterfaces aus *ROS* verwendet werden. Im Rahmen der Experimente wird ein Positioncontroller-Interface verwendet. Diese Schnittstelle empfängt die Soll-Gelenkstellungen in Radian zum Zeitpunkt t und liefert die Ist-Position sowie den Zeitpunkt $t + 1$ in Millisekunden. Die Zustandsmaschine aus Kapitel 4.3.5 dient als Ablaufroutine für den Controller.

6.5 Inbetriebnahme des Kamerasystems

Zur Inbetriebnahme der *DMK33UX265*-Kameras als Stereosystem sind einige Arbeitsschritte erforderlich: Auslesen der Bilddaten, Kalibrierung sowie zeitliche Synchronisation. Ein wesentlicher Teil der Entwicklungsarbeit und -zeit steckt in diesem Teilbereich der Implementierung, da für die gewählte Kamerahardware keine passende Bibliothek in *ROS* zur Verfügung steht.

6.5.1 Bilddatenakquise

Im *ROS*-Stack existiert bereits ein Softwarepaket[3] zur Verwendung von Kameras in Programmanwendungen auf Basis von *V4L2*, allerdings lassen sich damit nur grundlegende Einstellungen wie Auflösung und Bildrate verändern. Zur Konfiguration von gerätespezifischen Einstellungen von Signalpolarität und Triggermodi der *DMK33UX265* wurden allerdings die, für Hersteller freigehaltenen Usercontrol-Register im *V4L2*-Standard verwendet. Dementsprechend wurde eine eigenständige *ROS*-Node für das Auslesen und Konfigurieren der Kameras implementiert.
Die Kommunikation von Ein- und Ausgabegeräten erfolgt in Unixsystem über den Aufruf von *ioctrl()*-Funktionen auf Filedeskriptoren ([BM17], Kapitel 2). Dabei werden durch gezielte Änderungen im Speicherbereich des Geräts Einstellungen angepasst. Die entsprechenden Register und möglichen Einstellungen sind dabei durch den Gerätetreiber definiert. Um die Befehle und Datenstrukturen für verschiedenste Kamerasysteme zu vereinheitlichen, wurde *V4L2* als Standartschnittstelle für Videosysteme in den Unixkernel integriert. Die über *ioctl()* zu manipulierenden Speicherbereiche für allgemeine Einstellun-

[3] online unter `http://wiki.ros.org/usb_cam`, abgerufen am 12.06.20

gen sind aus der Dokumentation von *V4L2* selbst zu entnehmen. Für kameraspezifische Funktionalitäten stellt der Hersteller eine Treiberdatei zur Verfügung.

Um Bilddaten verwenden zu können, werden beim Kameratreiber physikalischer Speicher als Buffer angefragt. Diese werden durch *mmap()* in den virtuellen Adressraum der im Usermode laufenden Node abgebildet, sodass dieser ausgelesen werden kann. Ist ein Buffer leer, wird dieser mittels *enqueue()* für den Treiber freigegeben, sodass er überschrieben werden kann. Volle Buffer können durch *dequeue()* gesperrt und ausgelesen werden. Je nach Bildauflösung, Bildfrequenz und Kodierung werden unterschiedlich große Datenmengen übertragen, weshalb sich die Verwendung von Multibuffering anbietet.

6.5.2 Kamerasynchronisierung

Bildbasierte Trackingalgorithmen wie der hier verwendete *MHT* arbeiten mit visuellen Markern, die mithilfe von Bildaufnahmen aus mehreren Perspektiven zu einem bestimmten Zeitpunkt erkannt werden. Anschließend wird aus den detektierten Features eine dreidimensionale Rekonstruktion des Markers vorgenommen. Bei beweglichen Markern verändert sich die Position in Abhängigkeit der Zeit, sodass die Bildpunkte bei Aufnahmen zu unterschiedlichen Zeitpunkten nicht den selben Punkt im Weltkoordinatensystem abbilden. In unbewegten Szenen hat die zeitliche Differenz zwischen Aufnahmen unterschiedlicher Kameras keinen Einfluss. Allerdings ist das Resultat bei der Verwendung von Stereoverfahren eine fehlerhafte Projektion und letztendlich eine inkorrekte Vorhersage. Deshalb ist es bei der Arbeit mit stereobasierten Trackingsystemen essenziell, das die am PC angeschlossenen Kameras Bilder zeitsynchronisiert bereitstellen. Werden mehrere Kamerasysteme über unterschiedliche Eingänge an einen PC angeschlossen und die Aufnahme anschließend nicht über ein Steuerungssignal geschaltet, so ist die Wahrscheinlichkeit groß, dass diese Kameras Daten zu unterschiedlichen Zeitpunkten liefern.
Aus diesem Grund ist es notwendig, ein externes Synchonisierungssignal zu etablieren. Dieses Synchronisierungssignal kann sowohl software- als auch hardwareseitig implementiert werden. Im Rahmen dieser Arbeit wurde dieses Signal zunächst in Form von Hardware implementiert. Dazu wurde ein Synchronisierungskabel angefertigt, welches die zwei *DMK 33UX265*-Kameras über den 12-Pin-I/O-Connector verbindet. Leider wird die erforderliche Stromspannung für das Triggersignal nicht über die USB-Datenverbindung der Kameras bereitgestellt, weshalb der erforderliche Strom über einen weiteren USB-Port geliefert wird.
Während des Betriebs der Kameras wird bei jeweils einer Kamera das *StrobeOut*-Bit, bei der anderen das *TriggerIn*-Bit via *V4L2-API* gesetzt. Solange die Blende der auslösenden Kamera geöffnet ist, liegt ein High-Signal auf der Schaltleitung. Dies führt dazu, dass die andere Kamera ebenfalls ihre Blende öffnet und erst wieder schließt, sobald das Signal auf Low schaltet. Beim Schreiben in die jeweiligen Buffer der Kameras wird ein Zeitstempel auf Basis der Systemzeit geliefert. Dieser gibt Aufschluss über den Zeitpunkt, an dem das Bild fertig geschrieben wurde. Auf diese Weise werden Bilder im Abstand von durchschnittlich $36\mu s$ aufgezeichnet.

Da für den *MHT* die aktuellsten Messungen am relevantesten sind, erfolgt zusätzlich eine softwareseitige Synchronisierung. Diese ist auch bei der Verwendung von Multibuffering sinnvoll, da die Position des Buffers in der Queuestruktur keine Aussage über die Aktualität liefert.

6.5.3 Kalibrierung

Die Funktion der intrinsischen und extrinsischen Parameter einer Kamera wurden in Kapitel 3.7 bereits beschrieben. Sind diese Werte bekannt, ist es möglich, einen Rückschluss über die den Aufbau einer Szene zu approximieren. Dazu muss die Genauigkeit der Parameter gewährleistet sein. Die Bestimmung der intrinsischen und extrinsischen Kameraparameter ist auch als Kamerakalibrierung bekannt.
Die intrinsischen Parameter repräsentiert die Kameramatrix K (vgl. Kapitel 3.7, Gleichung 3.20). Der extrinsische Teil ${}^{C_R}_{C_L}T$ in Form einer homogenen Transoformationsmatrix beschreibt die Lage der Kameras zueinander. Dabei dient das linke Kamerakoordinatensystem als Basiskoordinatensystem.
Zur Berechnung von K und ${}^{C_R}_{C_L}T$ wird die Implementierung von *OpenCV* verwendet. Das Verfahren zur intrinsischen Kalibrierung basiert auf dem von Zhang [Zha00] vorgestellten Ansatz. Dafür werden Welt-Bild-Korrespondenzen in Form von Messpunkten in einem bekannten Objektkoordinatensystem benötigt und das daraus entstehende Gleichungssystem im Kameramodell durch lineare Ausgleichsrechnung aufgelöst. Wichtig dabei ist, dass alle Messpunkte im Objektkoordinatensystem auf einer Ebene ($z = 0$) liegen, sodass das zu lösende Gleichungssystem weniger Unbekannte enthält. Ein beliebtes Kalibrierungsobjekt für diesen Zweck ist ein planes Schachbrettmuster mit definierter Kantenlänge der Kacheln, welches auch in diesem Rahmen der Arbeit verwendet wurde. Die Ecken der Kacheln sind eindeutig identifizierbare Features, die über mehrere Aufnahmen hinweg wiedererkennbar sind.
Um ausreichend Objektpunkte zu generieren, wurden sowohl für die linke als auch die rechte Kamera jeweils 20 Full-HD-Aufnahmen des Kalibrierungsmusters in verschiedenen Ausrichtungen gemacht (vgl. 6.3). *OpenCV* stellt für Schachbrettmuster einen Bildverar-

Abbildung 6.3: Beispiel einer Kameraaufnahme, die zur Stereokalibrierung des Kamerasystems verwendet wurde. Die Perspektive stammt jeweils aus Sicht der linken bzw. rechten Kamera.

beitungsansatz, um die Ecken vollautomatisch extrahieren zu können. Für die Stereokalibrierung über *OpenCV* ist es allerdings notwendig, dass die *n-zu-n*-Punktekorrelation gegeben ist. Das heißt, dass jeder Objektpunkt P sowohl im linken als auch rechten Bild paarweise identifizierbar ist. Doch im Falle der Randbilder kommt es vor, dass einige Objektpunkte nicht auf beiden Bildern abgebildet werden. Die paarweise Zuordnung ist durch das vollautomatische Verfahren nicht durchführbar, da der oberste linke Punkt stets als Ursprung des Koordinatensystems gewertet wird.
Aus diesem Grund wurde mittels *Qt* eine Benutzeroberfläche mit *OpenCV* im Backend entwickelt, die eine Zuordnung von Pixeln zu identifizierbaren Objektpunkten per Mausklick ermöglicht. Darüber hinaus wird das Ergebnis der intrinsischen und extrinsischen Kameraparameter in Matrixdarstellung ausgegeben. Die Qualität der Gesamtkalibrierung

	Kamera_L	Kamera_R
f_x	1315.7024	1310.2935
f_y	1315.0389	1309.6806
c_x	973.0113	1002.6722
c_y	549.343	543.0997
k_1	−0.1084	−0.1102
k_2	0.102	0.1025
k_3	−0.018	−0.0154
ρ_1	0	0
ρ_2	0	0

Tabelle 6.1: Ergebnisse der intrinsischen Kalibrierung des Kamerasystems.

lässt sich über den *Root-Mean-Squared (RMS)*-Fehler E_{RMS} quantifizieren. Darin wird abgebildet, wie weit alle, über die bestimmten Kameraparameter projizierten Punkte p_p von den erwarteten Pixelkoordinaten p der Kalibrierungsaufnahmen aus n Bildaufnahmen durchschnittlich abweichen.

$$E_{RMS} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(p_p - p)^2} \mid p, p_p \in \mathbb{N}^2$$

Die extrinsische Kalibrierung kann zwar gleichzeitig mit der intrinsischen Optimierung stattfinden; allerdings hat sich nach eigenen Versuchen herausgestellt, das der Parameterraum, der optimiert werden muss, bei einer gleichzeitigen Kalibrierung zu groß wird und zu einem großen $E_{RMS} \approx 34.5$ führt. Deshalb wurden zunächst die intrinsischen Werte berechnet und anschließend für die extrinsische Kalibrierung als ideal angenommen. Für die intrinsischen Werte in Tabelle 6.1 und der extrinsischen Transformationsmatrix

$$\begin{matrix} C_R \\ C_L \end{matrix} T = \begin{bmatrix} 0.94465 & -0.0108 & 0.32789 & -378.4374 \\ 0.00918 & 0.99994 & 0.00641 & -1.8546 \\ -0.32795 & -0.00304 & 0.94469 & 61.20544 \end{bmatrix}$$

ergibt sich ein $E_{RMS} < 0.8$.

Die Transformationsmatrix zwischen linker Kamera C_L und *Baselink* W des Roboters wird in dieser Software nicht bestimmt. Dieser Vorgang wird als Hand-Auge-Kalibrierung bezeichnet. Da der Kameraaufbau aktuell nicht fixiert ist, muss diese Kalibrierung im Fall einer Verschiebung neu durchgeführt werden. Da die manuelle Bestimmung der Messpunkte mit entsprechendem Aufwand verbunden ist, wird auf eine genaue Bestimmung verzichtet. Deshalb wird die Transformationsmatrix ${}^{W}_{C_L}T$ händisch approximiert.

6.5.4 Adaption des *Multi Hypothesis Tracker*

Der in Kapitel 4.2 beschriebene *MHT* war in der Arbeitsgruppe bereits in Form von *ROS*-Nodes vorhanden. Jedoch fand die Entwicklung in einer alten Version von *ROS* statt, die mit der für die *FCI* benötigte Mindestversion aufgrund der strukturellen Veränderung im Buildsystem von *ROS* nicht direkt kompatibel ist. Deshalb wurde der Softwarestack in das neue Buildsystem überführt. Auf funktionaler Ebene muss das Domänenwissen auf das neue Kamerasystem und das neue Ballmodell angepasst werden. Das Ballmodell setzt

Abbildung 6.4: Erkannte Kreise (grün) in einem Bild der linken Kamera.

sich aus dem Radius, der Masse, dem spezifischen Luftwiderstandskoeffizienten α sowie der Abbildungsgröße auf der Bildebene in Pixeln zusammen.
Für einen standardisierten Tennisball liegt α bei 0.576 [MAS08]. Für die dreidimensionale Rekonstruktion wird die extrinsische Transformation ${}^{C_L}_{C_R}T$, Hand-zu-Auge-Transformation ${}^{W}_{C_L}T$ sowie die jeweilige intrinsische Kameramatrix übergeben. Die Werte und deren Ermittlung werden in Kapitel 6.5.3 erläutert.
Die Abbildungsgröße des Balls wird in vier Stufen eingeteilt (vgl. Kapitel 4.2). Der Kreiserkenner liefert in der gewählten Konfiguration robuste Ergebnisse (vgl. Abbildung 6.4).

6.5.5 Probleme

Bei der Anpassung des *MHT* traten allerdings einige Schwierigkeiten auf. Das erste Problem hängt mit der Kamerahardware zusammen. Die möglichen Bildauflösungen und dazugehörigen Betriebsfrequenzen beschränken sich bei der Nutzung von *V4L2* auf eine feste Auswahl an Kombinationen (vgl. Kapitel 5.2, Tabelle 5.2). Birbach et al. [BFB11] erzielten robuste Ergebnisse bei einer Auflösung von 1600×1200 [BFB11], weshalb als Annäherung die verfügbare Full HD-Auflösung gewählt wurde.
Allerdings ist es aufgrund eines Fehlers in der Treiberkomponente des Herstellers bei dieser Auflösung nicht möglich, die Bildrate anzupassen. Somit muss die Standardrate von 90 Bildern pro Sekunde als obligatorische Betriebsfrequenz verwendet werden[4]. Daraus resultiert eine Komplettauslastung zweier Rechenkerne (Referenz: AMD FX-8350 @4.2GHz) für die Bereitstellung von Bilddaten. Darüber hinaus beläuft sich die Verarbeitungszeit von der Aufnahme bis zur Weitergabe auf $\sim 15ms$ in getrennten Prozessen und auf $\sim 30ms$ bei der zusätzlichen Verwendung der softwareseitigen Synchronisation. Da neue Bilddaten aufgrund der Bildrate nach spätestens $11.1ms$ zur Verfügung stehen, wird unnötig Rechenzeit durch das Auslesen ungenutzter Bilddaten verschwendet.

Das zweite Problem ergibt sich durch das asynchrone Laufzeitverhalten von *ROS*. Für jede Kamera wird ein Kreisdetektor in einem eigenen Prozess gestartet, in dem wiederum mit Multithreading die Detektion der Kreismessungen stattfindet. Während der Laufzeit ist es möglich, dass einer der Prozesse bereits zwei Suchzyklen auf zwei verschiedenen Messungen durchgeführt hat, während der andere Detektor noch auf veralteten Daten arbeitet. Dadurch werden im *MHT* zeitasynchrone Messungen miteinander verglichen und sinnvollerweise verworfen. Daraus resultiert jedoch eine stark reduzierte Menge an nötigen Messungen, die zusätzlich unnötigerweise berechnet werden. Auch eine Drosselung der Bildrate durch *ROS*-Mechanismen konnte diesen Fehler nicht beheben.

Unabhängig von den Problemen der Synchronisation besteht die Möglichkeit einer fehlerhafter Parametrisierung der Komponenten:

- **Fehler in der Kalibrierung:** Gegen einen Fehler der Kalibrierung spricht der niedrige *RMS*-Fehler (vgl Kapitel 6.5.3).
- **Transformationen:** Die Transformation ${}^{W}_{C_L}T$ wurde zur Vereinfachung als Identitätsmatrix angenommen. Dies stellte allerdings auch keine Lösung dar.
- **Parametrisierung des *UKF***: Das Debugging dieser Parameter erwies sich als schwierig, das keine Dokumentation dazu existiert. Vermutlich liegt das Problem in diesem Bereich.

Aufgrund der Probleme ist es letztendlich nicht gelungen, den *MHT* mit der gegebenen Hardware funktionsfähig umzusetzen.

[4] Der Fehler wurde zum Zeitpunkt der Ausarbeitung (Stand: 24. Juli 2020) seitens des Herstellers noch nicht behoben.

7 Experimente

Das in Kapitel 6 implementierte System soll nun im Rahmen einiger Experimente evaluiert werden. Diese werden in einem vorgegebenen Simulationsszenario durchgeführt. Die daraus resultierenden Ergebnisse werden qualitativ ausgewertet und auftretende Probleme diskutiert.

7.1 Simulationsszenario

Um die Funktionstüchtigkeit des implementierten Controllers und seiner Subkomponenten zu testen, wurde ein Testaufbau in der Simulation integriert, welches den realen Anwendungsfall abbildet. In diesem Szenario werden dem System beispielsweise auf einem Messeauftritt Bälle zugeworfen, die wiederum erkannt und entlang der geworfenen Ballflugbahn zurückgeschlagen werden sollen. Wichtig bei diesem Szenario ist, dass alle Bälle volley geschlagen werden und nicht, wie bei einem echten Tennisspiel, vorher vom Boden abprallen.

In einem realen Ballspiel lässt sich die Schlagtechnik und die daraus resultierende Flugbahn des Balls beinahe unmöglich vorhersagen. Um diese hohe Dynamik des Problems darstellen und eine gute Generalisierung erreichen zu können, sollte die Art des Wurfs und die Ausgangsposition des Werfenden somit stark variieren. Dazu werden dem Manipulator aus einer festgesetzten Entfernung Bälle mit unterschiedlicher Geschwindigkeit und Ausrichtung zugeworfen. Während die Position in Bezug auf die Entfernung konstant bleibt ($5m$ für alle Experimente) wird die horizontale Position entlang der y-Achse des System verändert. Beim Tracking des Balls werden $\sigma_b = (0, 0, 0)^T$ und $\sigma_{\dot{b}=(0,0,0)^T}$ für die Wahrscheinlichkeitsverteilung der Geschwindigkeits- und Positionsvorhersage des Balls angenommen. Somit wird die Planung sofort nach Abwurf des Balls begonnen.

Damit eine Vergleichbarkeit verschiedener Komponenten und Konfigurationen der Module gewährleistet ist, muss das Testszenario wiederholbar sein. Aus diesem Grund wird für die Randomisierung von Winkelgeschwindigkeit, Wurfhöhe und -geschwindigkeit sowie Positionsoffset ein fester Seed gewählt. Auf diese Weise werden in jedem Testszenario dieselben Würfe ausgeführt. Insgesamt werden für jedes Testszenario 100 Würfe evaluiert.

7.1.1 Evaluation der inversen Kinematik

Um überhaupt in der Lage zu sein, den Ball mit dem Schläger zu treffen, ist es essenziell, dass

- gültige Abschlagposen anhand des Balltracks generiert werden.
- gültige Gelenkwinkelpositionen berechnet werden, sodass der Endeffektor eine Abschlagpose erreicht

Diese beiden Aufgaben werden von der *InterceptionCalculation*-Komponente sowie der *InverseKinematics*-Routine im Zusammenspiel übernommen, die nun überprüft werden sollen.

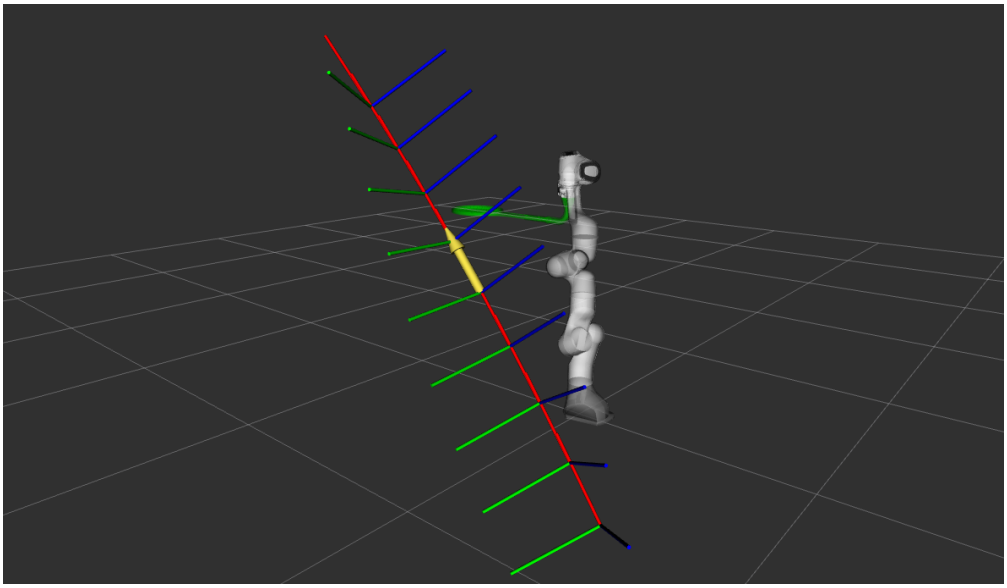


Abbildung 7.1: Visualisierung der möglichen Abschlagposen aus der *InterceptionCalculation*. Koordinatenachsen: mögliche Endeffektorposen entlang der Balltrajektorie. Gelber Pfeil: ideale Pose, ausgewählt durch die *IK*-Komponente.

Erwartetes Verhalten:

1. Die *InterceptionCalculation*-Komponente erzeugt eine gültige anzustrebende Abschlagpose P^* auf Basis eines Ballmodells.
2. Das *IK*-Modul der Software ist in der Lage, einen finalen Gelenkszustand q^* zu berechnen, sodass der Endeffektor die Pose P^* einnimmt.

Durchführung:

Für 100 Würfe werden P^* (vgl. Abbildung 7.1) und q^* mittels des *InterceptionCalculation*- und *IK*-Moduls bestimmt. Anstelle einer Trajektorienplanung wird mithilfe von Gazebo

die Position der Gelenke sofort auf die angegebene Zielkonfiguration gesetzt, sodass der Ball im Idealfall vom Schläger abprallt, unabhängig von der Wurfgeschwindigkeit und des Abschlagzeitpunkts. Auf diese Weise kann zudem die Menge der Würfe bestimmt werden, die auch in späteren Experimenten vermutlich nicht erreicht werden können.

Ergebnisse:

Aus 100 getätigten Würfen wurde in 96 Fällen ein Lösung für die *IK* gefunden. Vier der Würfe lagen folglich außerhalb des Arbeitsbereichs. In jedem der 96 Fälle ist der Ball vom Schläger abgeprallt. Bei einer subjektiven Evaluation während des Experiments ist auffällig, dass die Abschlagsposition in zwei Fällen sehr nah am Basisgelenk des Manipulators bestimmt wurde. In einem realen Szenario besteht hier eine extreme Gefahr einer Eigenkollision, welche Schäden an der Hardware nach sich zieht. Zukünftig wäre es sinnvoll, möglich Abschlagsposen im direkten Umfeld des Roboters zu filtern. Darüber hinaus resultiert aus diesem Experiment, dass der Ansatz zur Bestimmung der Zielposen in seiner jetzigen Form nicht auf dem realen System verwendet werden sollte.

7.1.2 Einfluss des Restitutionskoeffizienten

Der Restitutionskoeffizient e ist eine entscheidender Faktor bei der Berechnung der benötigten Geschwindigkeit des Endeffektors zum Abschlagszeitpunkt. Je größer dieser ist, desto weniger Kraft muss in den Abschlag gelegt werden, um den Ball entlang seiner Flugbahn zurückzuschlagen.

Erwartetes Verhalten:

1. Der Restitutionskoeffizient von Ball und Schläger, welche im realen Szenario verwendet werden sollen, ist zu niedrig, um mit dem implementierten Ansatz sinnvoll zurückspielen zu können.
2. Ein höherer Restitutionskoeffizient, wie er bei einem regulären Tennisschläger und Tennisball vorzufinden ist, vergrößert den Lösungsraum des Optimierungsalgorithmus für $\dot{q}^*$.

Durchführung:

Zunächst wird der Restitutionskoeffizient zwischen dem Hartplastikschläger und dem Softball experimentell approximiert (vgl. 3.2). Dazu wird der Schläger in einen Schraubstock gespannt und aus einer Höhe von einem Meter ein Ball insgesamt zehn Mal auf die Schlagfläche fallen gelassen. Aus der daraus resultierenden Rücksprunghöhe lässt sich der Restitutionskoeffizient bestimmen. Der Schaumstoff der Bälle verhärtet sich mit der Zeit aufgrund längerer Lagerung an der Luft und kann die Abprallhöhe beeinflussen, wenn auch nur minimal. Aus diesem Grund wird dieses Experiment mit insgesamt drei verschiedenen Bällen ausgeführt. Die Ergebnisse zum gemessenen Rebound *Rb* sind in Tabelle 7.1 zu sehen. Aus Gleichung 3.6 ergibt sich e.

Ball	$Rb_{min}(m)$	$Rb_{max}(m)$	$Rb_{mean}(m)$	$Rb_{median}(m)$	e_{median}
B_r	0.10	0.16	0.132	0.13	**0.36**
B_g	0.08	0.15	0.115	0.12	**0.34**
B_b	0.10	0.15	0.133	0.13	**0.36**

Tabelle 7.1: Ergebnisse zur Bestimmung des realen Resitutionskoeffizienten zwischen Schläger und Ball.

Eine wichtige Beobachtung bei diesem Experiment ist, dass sich der Schläger verdreht und nachschwingt, sobald ein Ball an den Rand der Schlagfläche fällt. Dies ist damit zu begründen, dass der Schläger ein offenes Profil hat und es somit an Stabilität zwischen dem Übergang vom Griff zur Schlagfläche mangelt. Der aus der Rotation resultierende Energieverlust wirkt sich zum einen negativ auf die Rücksprunghöhe des Balls aus, was sich jeweils in den kleinst gemessenen Werten widerspiegelt. Zum anderen wird die Flugbahn des Ball durch die hinzugewonnene Winkelbeschleunigung stark von der Eingangsflugbahn abgelenkt, womit selbst eine ideal berechnete Endeffektorgeschwindigkeit zum Abschlagzeitpunkt hinfällig wird. Aus diesem Grund ist es für das gezielte Rückspiel essenziell, dass der Ball mit der Mitte der Schlagfläche getroffen wird.

Die Komponente *Interception Calculation* wird mit zwei verschiedenen Restitutionskoeffizienten konfiguriert, um diesen mit in die Berechnung der nötigen Endeffektorgeschwindigkeit einzubringen. Ein Wert von 1 entspricht dabei der Krafterhaltung in einem idealelastischen Stoß. Nach Wriggers et al. ([Wri+06], S. 352) liegt der, von der *International Tennis Federation (ITF)* vorgegebene Restitutionskoeffizient von Tennisschläger und Ball bei 0.85 . Für den Realvergleich wird $e = 0.36$ aus dem Zwischenexperiment verwendet.
In 100 Würfen werden nur die Anzahl IK_{valide} gewertet, für die das *IK*-Modul auch eine Lösung findet. Im Rahmen der daraus resultierenden, gültigen Episoden wird die durchschnittliche Zahl erreichter Gelenkgeschwindigkeitslimits lim_{mean} alle Episoden bestimmt. Im Idealfall werden die Limits der Gelenke nur selten ausgereizt, um die mechanische Belastung auf der realen Hardware niedrig zu halten. Darüber hinaus werden die durchschnittlichen Restkosten R_{mean} über alle Episoden gebildet. Der Restterm R der Optimierungsfunktion gibt im Falle keiner Konvergenz gegen Null einen Hinweis darauf, dass ein Rückspiel mit dieser Konfiguration in Bezug auf die Geschwindigkeit gar nicht möglich ist. Aus diesem Grund werden zusätzlich die Anzahl $\dot{q}^*_{valide}$ optimierter Gelenkgeschwindigkeiten gemessen, deren Restterm $R \leq 0.1$ ist. Eine Wert für $R = 0.1$ bedeutet, dass ein Fehler von $10\frac{cm}{s}$ bezüglich der Schlägergeschwindigkeit entlang aller Achsen im Referenzkoordinatensystem vorliegt. Der Toleranzwert $R = 0.1$ wurde experimentell bestimmt.

Ergebnisse:

e	IK_{valide}	$\dot{q}^*_{valide}$	lim_{mean}	R_{mean}
0.85	87	87	0.091954	0.000242991
0.36	87	40	2.3	0.784905

Tabelle 7.2: Ergebnisse zum Optimierungsprozess der Schlägergeschwindigkeit mit verschiedenen Restitutionskoeffizienten.

Für den Faktor $e = 0.36$ der tatsächlich verwendeten Schläger-Ball-Kombination wurde für weniger als die Hälfte der Posen eine ausreichend schnelle Gelenkgeschwindigkeit ermittelt, um den Ball zurückspielen zu können. In den Fällen, in denen ein Optimum gefunden wird, müssen etwa drei Gelenke an ihr Geschwindigkeitsmaximum gebracht werden. Auf dem realen System könnte dies unter Umständen ein Auslösen der hardwareseitigen Sicherheitsmechanismen zur Folge haben. Der durchschnittliche Restfehler lässt zudem darauf schließen, dass die benötigte Schlägergeschwindigkeit aufgrund der mechanischen Limitierungen des Manipulators gar nicht erbracht werden kann.
Anders ist es bei einem Koeffizienten $e = 0.85$, der bei regulären Tennisschlägern vorzufinden ist. In diesem Fall wurde für jede Pose eine gültige Endgeschwindigkeit approximiert, welche nur selten eine Steuerung mit maximaler Geschwindigkeit erfordert. R_{mean} lässt darauf schließen, dass der Ball in der Theorie entlang der Flugbahn zurückgeschlagen werden kann.
Aus den Ergebnissen lässt sich ableiten, dass die Verwendung eines regulären Tennisschlägers für die Anwendung auf dem echten System von Vorteil ist. Die Gesamtbelastung der einzelnen Gelenke kann dadurch reduziert werden.

Im Bezug auf die *IK* wird in diesem Experiment das nichtdeterministische Verhalten sehr deutlich. Während bei dem Experiment aus Kapitel 7.1.1 noch 96 Lösungen gefunden wurden, sind in diesem Fall nur 87 Gelenkkonfigurationen für die selben Würfe ermittelt worden. Daraus folgt, dass trotz einer mechanisch erreichbaren Endeffektorpose im Arbeitsraum nicht immer eine Lösung q^* ermittelt wird. Dieser Umstand ist auf die sehr eingeschränkte Bedenkzeit von $5ms$ pro Sample in Kombination mit einem lokalen Optimum im *TRAC-IK*-Algorithmus zurückzuführen.

7.1.3 Evaluation der Simulation

Die Simulation soll das reale Verhalten des Manipulators möglichst originalgetreu nachbilden. Bei der Implementierung der Simulationsumgebung wurden Erkenntnisse aus [Gaz+19] für Massen und Inertialmomente des Achsen verwendet. Daraus resultieren die Größe und Wirkrichtung externer Kräfte, die während des Simulationsbetriebs beim Ausführen von Gelenkbewegungen auf die Achsen wirken.
Für das in der Simulation verwendete Hardwareinterface für einen positionsgesteuerten Controller mussten darüber hinaus *PID*-Werte manuell optimiert werden. Diese haben einen sehr großen Einfluss auf die Fehlerkorrektur des Positionscontrollers und werden direkt von den vorher bestimmten physikalischen Parametern beeinflusst. Beide Teilaspekte zusammen ergeben einen entsprechend großen Dynamikparameterraum.
In diesem Experiment soll evaluiert werden, wie sich die verwendeten Dynamikparameter auf die Steuerung in der Simulationsumgebung auswirken. Darüber hinaus ist es sinnvoll, das Returnverhalten bei diesem Schlag zu untersuchen, da dieses direkt mit Fehlern in der Simulationsumgebung korreliert.

Erwartetes Verhalten:

1. Bei der Ausführung einer Schlagtrajektorie weicht der Ist-Wert der Gelenkpositionen kaum vom berechneten Soll-Wert ab.
2. Beim *Return* wird die Wurfbahn annähernd abgebildet.

Durchführung:

Als Referenzeingabe wird der Ballwurf verwendet, der durch den festgesetzten Seed als Erstes generiert wird. Der Ball wird aus Sicht der Werfenden rechts am Roboter im Abstand von etwa $0.7m$ vorbei geworfen und landet letztendlich im Arbeitsbereich. Rein subjektiv betrachtet ist diese Ballflugbahn gut erreichbar und auch nicht zu nah am Manipulator, sodass in den meisten Fällen ein Trajektorie mit langem Arm ausgeführt wird.
Über das bereitgestellte Hardwareinterface ist es möglich, die tatsächliche Position zum Zeitpunkt t auszulesen. Somit lassen sich die erwartete Position mit der tatsächlichen Position in Abhängigkeit der Zeit für alle Gelenke q_0 bis q_6 darstellen. q_0 ist dabei das unterste Gelenk, welches sich nahe am *Baselink* befindet, während q_6 das Rotationsgelenk am Flange beschreibt.

Ergebnisse:

Für den Beispielschlag folgen alle Gelenke ziemlich genau der vorgegebenen Trajektorie (vgl. Abbildung 7.2). Im Gelenk q_1 lässt sich eine leichte Oszillation um den Soll-Wert erkennen. Daher weicht zum Abschlagzeitpunkt T die Ist-Geschwindigkeit von der geplanten Soll-Geschwindigkeit ab. Ein möglicher Grund dafür ist die Funktionalität von q_2 in der kinematischen Kette. Dieses Gelenk ist für das Vor- und Zurückkippen des Gesamtsystems zuständig. Somit wirken die, durch das Gewicht der Folgegelenke wirkende Kraft direkt entlang der Bewegungsachse. Aus diesem Grund liegt es nahe, dass der Regler den

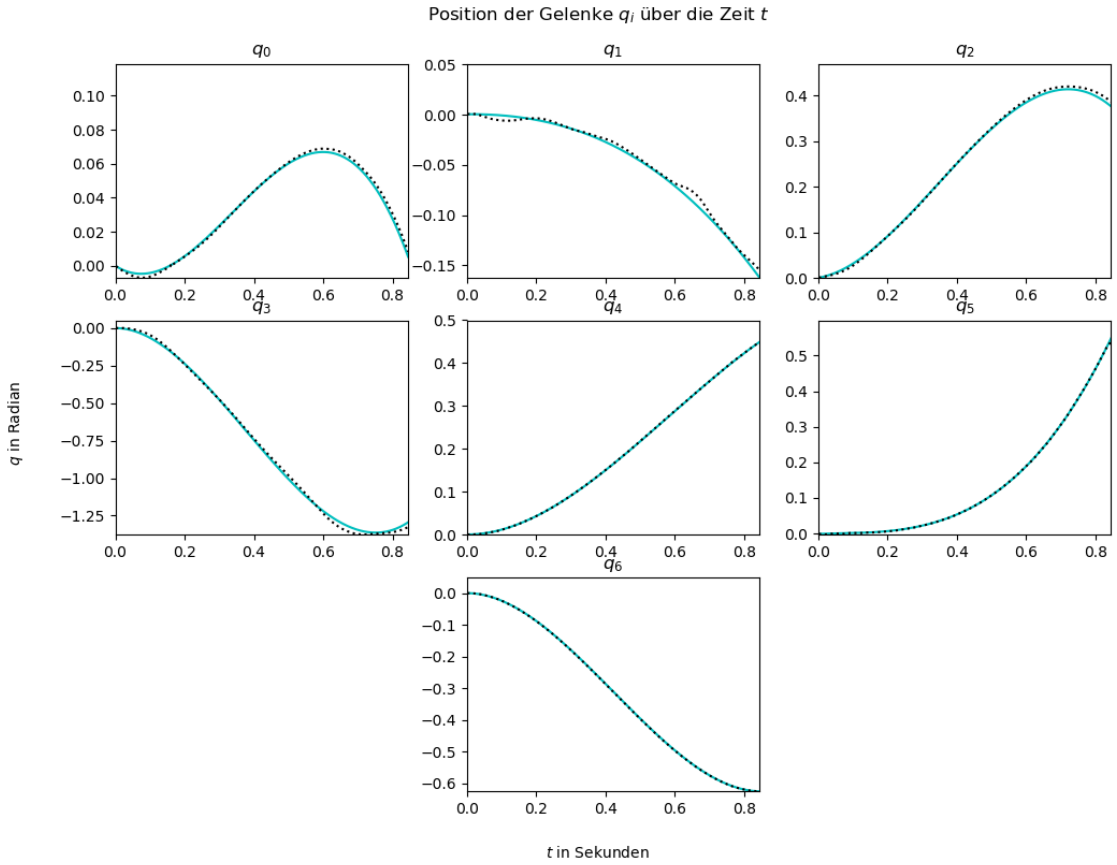


Abbildung 7.2: Vergleich der Soll-Gelenkposition mit der tatsächlichen Position bei dem ersten Beispielwurf in der Simulation. Cyan: Soll-Position aus der Planungsfunktion. Schwarz-gepunktet: Ist-Position der Gelenke.

P-Fehler mit entsprechend großer Gewichtung betrachten sollte. Jedoch hat auch eine weitere manuelle Anpassung der *PID*-Werte des Controllers die Oszillation nicht beseitigt. Möglicherweise wäre eine systematische Bestimmung dieser Werte für das Tennisszenario eine Lösung. Alternativ besteht die Möglichkeit, dass die von Gaz et al. [Gaz+19] ermittelten Werte nicht für dieses Problem angewandt werden können, da sich aufgrund des langen Schlägers die Krafteinwirkung auf die Gelenke maßgeblich ändert.

Im Beispielschlag wird die ursprüngliche Flugbahn nicht in der Returntrajektorie abgebildet (vgl. Abbildung 7.3). Eine kleine Ungenauigkeit in der Schlägerausrichtung resultiert in einer großen Richtungsänderung der Returnflugbahn. Dieser Fehler steht in direktem Zusammenhang mit der nicht erreichten Gelenkposition zum Zeitpunkt T aus Abbildung 7.2. Dieses Phänomen ist im Beispielschlag sehr gut zu erkennen. Im Bezug auf die Ballflughöhe ist zu beobachten, dass die Returnhöhe unter dem erwarteten Wert liegt. Daraus lässt sich ableiten, dass die erwartete Schlaggeschwindigkeit nicht erreicht wurde. Auch dieser Fehler korreliert mit dem Geschwindigkeitsfehler der Gelenksteuerung.

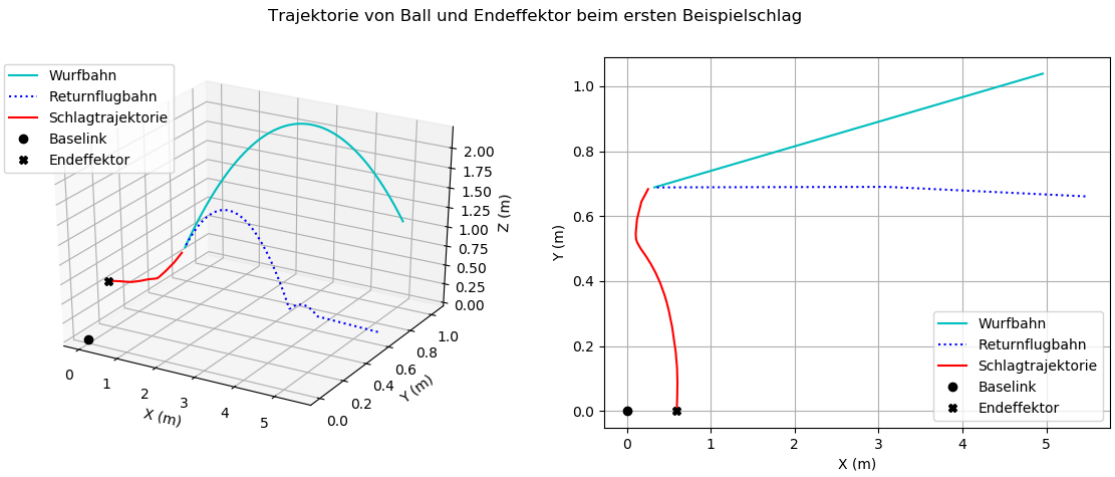


Abbildung 7.3: Visualisierung der Ball- und Schlagbewegung beim ersten Wurf der Testepisoden. Links: dreidimensionale Ansicht, in der sich die Returnhöhe erkennen lässt. Rechts: Vogelperspektive, bei der die Richtungsabweichung zu sehen ist.

Alternativ ist es möglich, dass die Gleichung 4.3 zur Berechnung der nötigen Abschlagsgeschwindigkeit aufgrund der Vereinfachung des physikalischen Modells nicht gültig ist.

7.1.4 Evaluation des Controllers

In den vorherigen Evaluationen wurden die Module des Controllers im Einzelnen getestet. Nun sollen alle Komponenten im Zusammenspiel überprüft werden. Im Fokus dieses Experiments stehen die getroffenen beziehungsweise verfehlten Bälle und das Returnverhalten. Dabei sollen das Spiel mit der Vorhand sowie das kombinierte Spiel von Vor- und Rückhand miteinander verglichen werden.

Erwartetes Verhalten:

1. Der Schläger erreicht in mindestens 50% der gültigen Würfe den Ball.
2. *Returns* finden wenige bis gar keine statt.
3. Es finden keine Kollisionen seitens des Manipulators mit sich selbst oder der Umgebung statt.
4. Das Rückhandspiel in Kombination mit dem Vorhandspiel verbessert die Trefferrate.

Durchführung:

Beim kombinierten Spiel besteht die Besonderheit, dass die doppelte Menge an möglichen Abschlagsposen vorhanden sind. Daraus resultiert bei der Optimierung eine ungefähre Verdopplung der Rechenzeit, die das Vorhandspiel alleine benötigt hat. Um dennoch online

Ergebnisse in Echtzeit zu erzielen, wird die Samplingrate im Gegensatz zum alleinigen Spiel mit der Vorhand halbiert.
Aus den Ergebnissen von Kapitel 7.1.2 geht hervor, dass sich die Verwendung eines regulären bespannten Tennisschlägers positiv auf die Optimierung der Gelenkgeschwindigkeiten und damit auch auf die Gesamtbilanz der mechanischen Auslastung auswirkt. Nichtsdestotrotz wird in diesem Experiment der Restitutionskoeffizient $e = 0.36$ verwendet, um in einer realistische Nachbildung des aktuellen Realszenarios zu evaluieren.
Um von einem erfolgreichen *Return* sprechen zu können, muss dafür ein messbares Merkmal festgelegt werden. Aus den durchgeführten Experimenten in Kapitel 7.1.3 geht hervor, dass Bälle beim Zurückschlagen niemals die Wurfstrecke von 5 Metern zurücklegen, die ursprünglich angestrebt wurden. Auf Basis dieser Ergebnisse ist ein *Return* dann als erfolgreich anzusehen, wenn der Ball mindestens 2 Meter in der Luft zurücklegt, bevor er die Spielfläche berührt.
Für die qualitative Analyse werden diese Experimente per Video festgehalten.

Ergebnisse:

Variante	IK_{valide}	*getroffen*	*Returns*	*verfehlt*	*Kollisionen*
Vorhand	87	62	13	25	6
kombiniert	93	65	10	28	10

Tabelle 7.3: Ergebnisse des Gesamtsystemtests in der Simulation. Getestet wurde das Spiel mit Vorhand sowie mit Vor- und Rückhand (kombiniert).

Bei der Betrachtung des Videomaterials fällt auf, dass der Manipulator stark zittert, wenn Bälle in kurzer Zeit mit gestreckten Arm erreicht werden müssen. Dadurch oszilliert der Endeffektor um seine eigentliche Zielpose und verfehlt daraufhin den Ball. Dieses Verhalten ist auf den Positionsfehler der Gelenke aufgrund hoher mechanischer Belastung zurückzuführen, die durch die große Hebelwirkung durch den langen Arm auftritt. Der P-Fehler macht bei den händisch bestimmten Parametern des Reglers den größten Korrekturwert aus, weshalb bei solchen Trajektorien oft übersteuert wird. Zur Behebung wäre es möglich, die Reglerparameter analytisch zu bestimmen. Allerdings würde dieses Vorgehen den Rahmen dieser Arbeit überschreiten.

In einigen Fällen kollidiert der Schläger mit dem System. Oft treten diese Kollisionen im Rahmen des bereits beschriebenen Zitterns auf. Eine Korrektur der *PID*-Werte wäre auch hier eine mögliche Lösung. Allerdings tritt auch ein Fall auf, in dem der Flange um die Längsachse gedreht und aufgrund dieser Bewegung eindeutig mit der nächsten Achse kollidiert. Dieser Fall kann nur vermieden werden, wenn die Planung vor der Durchführung überprüft wird. Dazu könnte beispielsweise ein pragmatischer Ansatz verwendet werden, bei dem die *VK* des Manipulators entlang der Planung gesampled und mittels einfacher geometrischer Körper eine Achsenkollision ermittelt werden. Solche Überprüfungen benötigen allerdings zusätzliche Rechenkapazitäten. Zudem ist die Genauigkeit dieses Verfahrens abhängig von der gewählten Samplingrate.

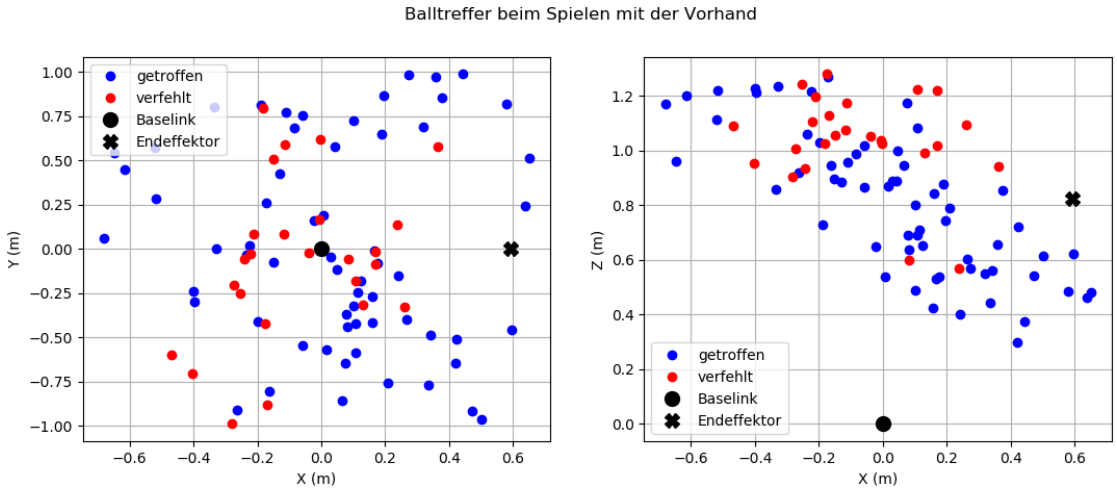


Abbildung 7.4: Visualisierung der getroffenen und verfehlten Ballpositionen beim Vorhandspiel. Links: Vogelperspektive. Rechts: Seitenansicht.

Für das Spiel mit der Vorhand wurden für 100 Würfen in 87 Fällen gültige q^* und $\dot{q}^*$ gefunden. Die Ausführung der dazugehörigen Bewegungspläne führten dazu, dass in 62 Episoden der Ball mit der Schlagfläche Kontakt hat. Die daraus resultierende Trefferquote von 71.3% übersteigt dabei die erwartete Mindestquote von 50%. Die Anzahl der *Returns* beläuft sich auf 13 (vgl. Tabelle 7.3). Bei der Analyse der Verteilung der getroffenen und verpasst Bälle (vgl. Abbildung 7.4) ist zu erkennen, dass sowohl Bälle vor als auch hinter dem Manipulator erfolgreich getroffen werden. Vorrangig werden dabei jene Bälle getroffen, die sich weiter vom Manipulator entfernt befinden. Bälle, die Richtung Mitte des Manipulators fliegen, werden selten getroffen oder resultieren gar in Eigenkollisionen. Dies deckt sich mit Erfahrungen aus dem realen Tennisspiel beim Menschen: Bälle, die in Richtung Körperzentrum gespielt werden, sind schwieriger zu treffen, da die Freiheitsgrade des Arms stark eingeschränkt sind. Zum Ausgleich verändert ein menschlicher Spieler seine Position so, dass mit einem längeren Arm gespielt werden kann. Im Falle dieses Systems ist keine laterale Positionsveränderung vorgesehen, weshalb dieses Problem aktuell nicht gelöst werden kann.
Darüber hinaus ist für einen erfolgreichen Treffer die Abschlagshöhe des Balls in Relation zur Ausgangsposition des Schlägers entscheidend. Wird der Ball unterhalb des Schlaggelenks angesteuert, so wird dieser sehr häufig erfolgreich geschlagen. Soll der Ball über dem Schlaggelenk abgefangen werden, so wird dieser nur sehr selten getroffen (vgl. Abbildung 7.4, rechts).

Dieses Defizit soll durch das Rückhandspiel ausgeglichen werden. Idealerweise wird die Gesamtänderung der Gelenkposition minimiert, indem der Schläger gehoben wird. Im kombinierten Test sind jedoch kaum Verbesserungen zu beobachten (vgl. Abbildung 7.5). Auch hier werden Bälle oberhalb des Schlaggelenks selten getroffen. Insgesamt sinkt die Trefferquote auf 69.9%, was eine Verschlechterung im Gegensatz zum Spiel mit der

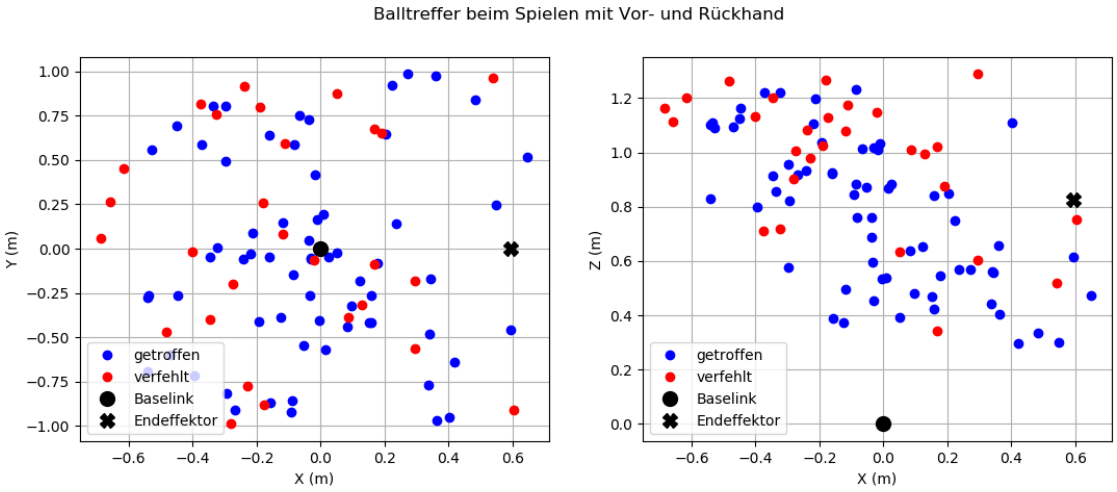


Abbildung 7.5: Visualisierung der getroffenen und verfehlten Ballpositionen beim kombinierten Spiel mit Vor- und Rückhand. Links: Vogelperspektive. Rechts: Seitenansicht.

Vorhand darstellt. Viele Fehlschläge sind allerdings auf die Oszillation des Schlägers in der Simulation zurückzuführen (vgl. Videomaterial). Für eine bessere Vergleichbarkeit erscheint es sinnvoll, die Dynamikparameter der Simulationsumgebung anzupassen. Darüber hinaus besteht durch die halbierte Samplingrate eine höhere Wahrscheinlichkeit dafür, keine optimale Pose zu finden, sodass folglich nur suboptimale Gelenkkonfigurationen zur Verfügung stehen.
Der Vorteil des kombinierten Ansatzes besteht darin, dass Konfigurationen für Ballpositionen gefunden werden, die mit dem Vorhandspiel alleine nicht berechenbar waren. Dies spiegelt sich in einer größeren Lösungsmengengröße $IK_{valid} = 93$ wider.

8 Zusammenfassung

In dieser Arbeit wurde eine Gesamtlösung für das Tennisspiel eines robotischen Manipulators mit sieben *DOF* entworfen und implementiert. Dazu gehört neben der eigentlichen Bewegungssteuerung die Adaption eines Verfahrens zur Erfassung des Ballzustands. Auf Basis der physikalischen Grundlagen des Tennisspiels und des aktuellen Stands der Technik wurde für beide Teilprobleme eine Ansatz ausgewählt.

Für das Balltracking wurde ein bildbasiertes Stereoverfahren, der *Multi Hypothesis Tracker* von Birbach et al. [BFB11], verwendet. Die Kamerahardware wurde in einem Stereoaufbau montiert und über ein eigens angefertigtes Verbindungskabel synchronisiert. Sowohl die intrinsische als auch extrinsische Kamerakalibrierung wurden über ein entwickeltes Tool vorgenommen. Der ausgewählte *MHT* für das Balltracking konnte im Rahmen dieser Arbeit nicht komplett erfolgreich umgesetzt werden. Obwohl die Kreiserkenner robust arbeiten, wird niemals eine Ballvorhersage getätigt. Die Fehleranalyse legt nahe, dass eine fehlerhafte Parametrisierung die Ursache ist.

Als Ansatz zur Bewegungsplanung wurde, inspiriert durch die Arbeiten von Koç et al. [KMP18] und Mülling et al. [Mül+09], ein pragmatisches *Virtual Hitting Plane*-Verfahren entworfen. Die Entwicklung und Evaluation der Bewegungssteuerung fand in einer eigens konstruierten Simulationsumgebung statt.
Das Gesamtergebnis der Controllerkomponente ist eher ernüchternd – 13 Bälle von 100 Würfen werden erfolgreich zurückgeschlagen. Nur 70% der Bälle werden überhaupt getroffen. Die Verwendung des Rückhandspiels brachte entgegen der Erwartung keine Verbesserung der Trefferquote mit sich. Keiner der Bälle wird entlang der ursprünglichen Wurfbahn zurückgeschlagen. Dies führt zu dem Schluss, dass die benötigte Schlägergeschwindigkeit in diesem Szenario aufgrund mechanischer Limitierungen mit diesem System nicht zu erbringen ist.

Nichtsdestotrotz bieten die in dieser Arbeit entwickelten Frameworks eine ideale Grundlage für weiterführende Arbeiten mit dem *Panda* und der Kamerahardware. Die Komplexität des Tennisspiels spiegelt sich in den Ergebnissen der Evaluation wider.

8.1 Ausblick

Im Bezug auf die Hardwarekonfiguration für das Trackingsystem können einige Verbesserungen vorgenommen werden. Es wäre sinnvoll, die Kameras nicht auf einer externen Konstruktion wie in dieser Arbeit zu montieren, sondern sie fest an den Manipulator selbst zu fixieren. Der Vorteil in dieser Konstellation besteht zum einen darin, dass die Informa-

tion über die Rotation des Basisgelenks mit in die Messungen des *MHT* einfließen können. Auf diese Weise wäre es möglich, den Ball beispielsweise mit den Kameras zu verfolgen und mehr Messungen für eine genauere Schätzung der Ballposition und -geschwindigkeit zu erhalten. Der Tracker wird auf seinem ursprünglich entwickelten System (vgl. [BFB11]) in Kombination mit einer *IMU* verwendet. Somit ist die grundlegende Funktion dafür bereits vorhanden.
Ein weiterer Grund für eine Befestigung der Kameras am Roboter selbst ist die robustere Hand-Auge-Kalibrierung des Gesamtsystems. Der Kameraaufbau auf dem Stativ ist in diesem Punkt sehr anfällig, da dieser sehr leicht verschoben werden kann. In diesem Fall ist die Transformation von Kamera zu Roboterkoordinaten fehlerbehaftet und für die Bewegungsplanung nicht zu verwenden Deshalb lohnt sich eine manuelle Hand-zu-Auge-Kalibrierung für diesen Aufbau nicht, sondern es sollte eher geschätzt werden.
Im Falle einer festen Montage der Kameras am Manipulator ist es zudem sinnvoll, die intrinsische, extrinsische sowie Hand-zu-Augen-Kalibrierung halb- oder vollautomatisch vorzunehmen. Dazu kann beispielsweise ein Schachbrettmuster am Schläger des *Panda* befestigt werden. Nun wird der Schläger mit verschiedenen Gelenkpositionen durch das Bild bewegt. Die Objektpunkte des Schachbretts können, wie auch in dieser Arbeit, für die intrinsische sowie extrinsische Kalibrierung der Kameras verwendet werden. Über die aufgezeichneten Gelenkwinkel und die zur Verfügung stehende *VK* kann eine Relation zwischen Schläger- und Kamerakoordinatensystem hergestellt werden.

Der präsentierte Ansatz zur Ansteuerung der Gelenke weißt noch sehr viele Schwächen auf, kann aber als Basis für zukünftige Arbeiten dienen. Ein sehr großer Schwachpunkt liegt in der einmaligen Planung und Optimierung der Trajektorie. Dieses Vorgehen führt dazu, dass auf das Balltracking gewartet werden muss, bis dieser eine Messung bereitstellt, die *genau genug* ist. Daraus resultiert ein enormer Zeitverlust in einem bereits sehr kurzen Zeitfenster, sodass Abschlagspositionen aus rein physikalischer Sicht schon nicht mehr rechtzeitig erreicht werden können. Ein reaktiver Regelkreislauf würde in dieser Hinsicht besser funktionieren.
Um die Robustheit des Ansatzes zu verbessern, wäre es zudem möglich, ideale Policies für möglichst viele Würfe offline zu optimieren und in einer Lookup-Tabelle einzuspeichern. Auf diese Weise wird eine Optimierung während des laufenden Betriebs hinfällig. Dadurch könnte auch Eigenkollisionen vorgebeugt werden.
In dieser Arbeit wurde ausschließlich entlang der Abwurftrajektorie zurückgespielt. Deshalb wäre die Implementierung eines Richtungsspiels eine interessante Erweiterung, die allerdings die Robustheit des Grundansatzes voraussetzt.
Zudem wurde das System vor allem aus Sicherheitsgründen in einer Simulationsumgebung getestet. Während die Komponenten hier lauffähig sind, wäre eine Evaluation mit dem echten Manipulator gerade in Hinblick auf die *Simulation-Reality-Gap* [JHH95] interessant. Solche Tests setzen allerdings die Kollisionsfreiheit des hier propagierten Ansatzes voraus, um Schäden an Mensch und Maschine zu verhindern.

Glossar

Baselink Bezeichnung für das Basiskoordinatensystem eines Roboters. Im Fall des *Panda* wird es durch das Zentrum des untersten Gelenks definiert . 15, 46, 54

Ceres Eine Bibliothek der Firma Google zur Lösung nichtlinearer Optimierungsprobleme. 38, 41

Gazebo Ein Simulator für robotische Systeme mit integrierter Physikengine. 40

OpenCV Eine Bibliothek für State-of-the-Art-Bildverarbeitungsalgorithmen. 38, 44, 45

Panda Ein robotisches Manipulatorsystem mit sieben Freiheitsgraden; produziert von der Firma *Franka Emika*. X, XI, 1, 33, 41, 61, 62

Rebound Das Abprallen eines Tennisballs von der Spielfläche. 13

Return Bezeichnung für den Schlag, der auf den Aufschlag des Gegenspielers folgt. 54, 56–58

TRAC-IK Ein Algorithmus zur Lösung inverser Kinematiken für generische, redundante Manipulatoren. Er kombiniert einen pseudoinversen Jacobiansatz mit einem *Sequential Quadratic Progamming*-Verfahren. 26, 28, 38, 53

Akronyme

API Application Programming Interface. 38, 43

CAD Computer-aided design. 40

CNN Convolutional Neural Network. 4

CNS Contrast Normalized Sobel Filter. 24

DMP Dynamic Motion Primitive. 6

DOF Degrees of Freedom. 5, 6, 15–19, 33, 61

FCI Franka Control Interface. 34, 46

IK Inverse Kinematik. 18, 28, 38, 50–53

IMU Inertial Measurement Unit. 24

ITF International Tennis Federation. 52

KDL Kinematics and Dynamics Library. 38

MHT Multiple Hypothesis Tracker. X, 24, 25, 39, 41, 43, 44, 46, 47, 61, 62

ODE Open Dynamics Engine. 39

RMS Root-Mean-Squared. 45, 47

ROS Robot Operating System. 34, 37, 39, 42, 46, 47

RPC Remote Procedure Calls. 37

UKF Unscented Kalman Filter. 4, 25, 47

URDF Unified Robot Description Format. 38, 39, 41

V4L2 Video for Linux Version 2. 38, 42, 43, 47

VHP Virtual Hitting Plane. IX, 4

VK Vorwärtskinematik. 16, 17, 38, 57, 62

Literatur

[AL09] Aristidou, A. und Lasenby, J. *Inverse Kinematics: a review of existingtechniques and introduction of a new fastiterative solver*. Techn. Ber. University of Cambridge, 2009.

[And88] Anderson, R. L. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control*. MIT Press, Cambridge, MA, USA, 1988. ISBN: 0262011018.

[BA15] Beeson, P. und Ames, B. TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, 928–935.

[BB09] Brutscheck, T. und Bücker, M. Modellierung von Roboterkinematiken nach Denavit und Hartenberg. In: *„Das ist gar kein Modell!“: Unterschiedliche Modelle und Modellierungen in Betriebswirtschaftslehre und Ingenieurwissenschaften*. Hrsg. von Bandow, G. und Holzmüller, H. H. Gabler, Wiesbaden, 2009, 149–165. DOI: 10.1007/978-3-8349-8484-5_7.

[BCL02] Brody, H., Cross, R. und Lindsey, C. The physics and technology of tennis. In: Racquet Tech Pub., Solana Beach, Calif., 2002. Kap. 42. ISBN: 9780972275903.

[Bea] Beaufort, J. *Pro Female Tennis Player*. URL: https://www.publicdomainpictures.net/en/view-image.php?image=284978. veröffentlicht unter CC0 (aufgerufen: 07.07.2020).

[BFB11] Birbach, O., Frese, U. und Bäuml, B. Realtime perception for catching a flying ball with a mobile humanoid. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, 5955–5962.

[BM17] Bertolotti, I. und Manduchi, G. *Real-Time Embedded Systems*. 1st edition. 534. CRC Press, 2017. URL: https://learning.oreilly.com/library/view/-/9781439841617/?ar.

[Bor+09] Borst, C. et al. Rollin' Justin - Mobile platform with variable base. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, 1597–1598.

[BWH10] Bäuml, B., Wimböck, T. und Hirzinger, G. Kinematically optimal catching

a flying ball with a hand-arm-system. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, 2592–2599. DOI: 10.1109/IROS.2010.5651175.

[Cro10] Cross, R. Enhancing the Bounce of a Ball. *The Physics Teacher* 48 (Okt. 2010). DOI: 10.1119/1.3488187.

[Den15] Dennis Schüthe, U. F. Optimal Control with State and Command Limits for a Simulated Ball Batting Task. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, 3988–3994.

[DH55] Denavit, J. und Hartenberg, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics* 22 (Juni 1955), 215–221.

[Dru+10] Drumwright, E. et al. Extending Open Dynamics Engine for Robotics Simulation. In: *Simulation, Modeling, and Programming for Autonomous Robots*. Hrsg. von Ando, N. et al. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, 38–50. ISBN: 978-3-642-17319-6.

[Fra17] Franka Emika GmbH *Robot and interface specifications*. 2017. URL: https://frankaemika.github.io/docs/control_parameters.html. (aufgerufen: 01.05.2020).

[Gaz+19] Gaz, C. et al. Dynamic Identification of the Franka Emika Panda Robot With Retrieval of Feasible Parameters Using Penalty-Based Optimization. *IEEE Robotics and Automation Letters* 4, 4 (Okt. 2019), 4147–4154. DOI: 10.1109/LRA.2019.2931248.

[GS10] Goyal, K. und Sethi, D. An analytical method to find workspace of a robotic manipulator. *Journal of Mechanical Engineering* 41, 1 (Juni 2010), 25–30.

[Hac18] Hackbarth, J. *Simulations-Realitäts-Transfer – Lernen von gezieltem Zurückspielen auf einem Ballspielroboter*. 2018.

[HI12] Haron, A. und Ismail, K. A. Coefficient of restitution of sports balls: A normal drop test. *IOP Conference Series: Materials Science and Engineering* 36 (Sep. 2012), 012038. DOI: 10.1088/1757-899x/36/1/012038.

[JHH95] Jakobi, N., Husbands, P. und Harvey, I. Noise and the reality gap: The use of simulation in evolutionary robotics. In: *Advances in Artificial Life*. Hrsg. von Morán, F. et al. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, 704–720. ISBN: 978-3-540-49286-3.

[KH04] Koenig, N. und Howard, A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2004, 2149–2154.

[KMP18] Koç, O., Maeda, G. und Peters, J. Online optimal trajectory generation for robot table tennis. *Robotics and Autonomous Systems* 105 (2018), 121–137.

ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2018.03.012.

[Kri] Krishna, S. *Jacobian*. URL: https://www.rosroboticslearning.com/jacobian. (aufgerufen: 22.07.2020).

[Li+15] Li, B. et al. Simulation of Golf Realtime Tracking Based on Doppler Radar. *Applied Mechanics and Materials* 743 (März 2015), 828–835. DOI: 10.4028/www.scientific.net/AMM.743.828.

[Mah+18] Mahjourian, R. et al. Hierarchical Policy Design for Sample-Efficient Learning of Robot Table Tennis Through Self-Play. *CoRR* abs/1811.12927 (2018). arXiv: 1811.12927.

[MAS08] Mehta, R., Alam, F. und Subic, A. Review of tennis ball aerodynamics. *Sports Technology* 1, 1 (2008), 7–16. DOI: 10.1002/jst.11.

[Mat+05] Matsushima, M. et al. A Learning Approach to Robotic Table Tennis. *IEEE Transactions on Robotics* 21 (Sep. 2005), 767–771.

[Mül+09] Mülling, K. et al. A Computational Model of Human Table Tennis for Robot Application. *Autonome Mobile Systeme 2009: 21. Fachgespräch, 57-64 (2009)* (Jan. 2009). DOI: 10.1007/978-3-642-10284-4_8.

[Mül+13] Mülling, K. et al. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32, 3 (2013), 263–279. DOI: 10.1177/0278364912472380.

[Ope19] OpenCV *Camera Calibration and 3D Reconstruction*. 2019. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. (aufgerufen: 04.06.2020).

[RDD94] Ramanantsoa, M.-M., Durey, A. und Decret, J. Towards a stroke Construction Model. *International Journal of Table Tennis Sciences* 2 (1994), 97–113.

[RH10] Ronkainen, J. und Harland, A. Laser tracking system for sports ball trajectory measurement. *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology* 224, 3 (2010), 219–228. DOI: 10.1243/17543371JSET67.

[Sas94] Sasse, R. Kameramodell. In: *Bestimmung von Entfernungsbildern durch aktive stereoskopische Verfahren*. Vieweg+Teubner Verlag, Wiesbaden, 1994, 3–24. ISBN: 978-3-322-88814-3. DOI: 10.1007/978-3-322-88814-3_2.

[The19] The Imaging Source *DMK 33UX265 Technical Reference Manual*. 2019.

[TKZ19] Tebbe, J., Klamt, L. und Zell, A. Spin Detection in Robotic Table Tennis. *CoRR* abs/1905.07967 (2019). URL:

http://arxiv.org/abs/1905.07967.

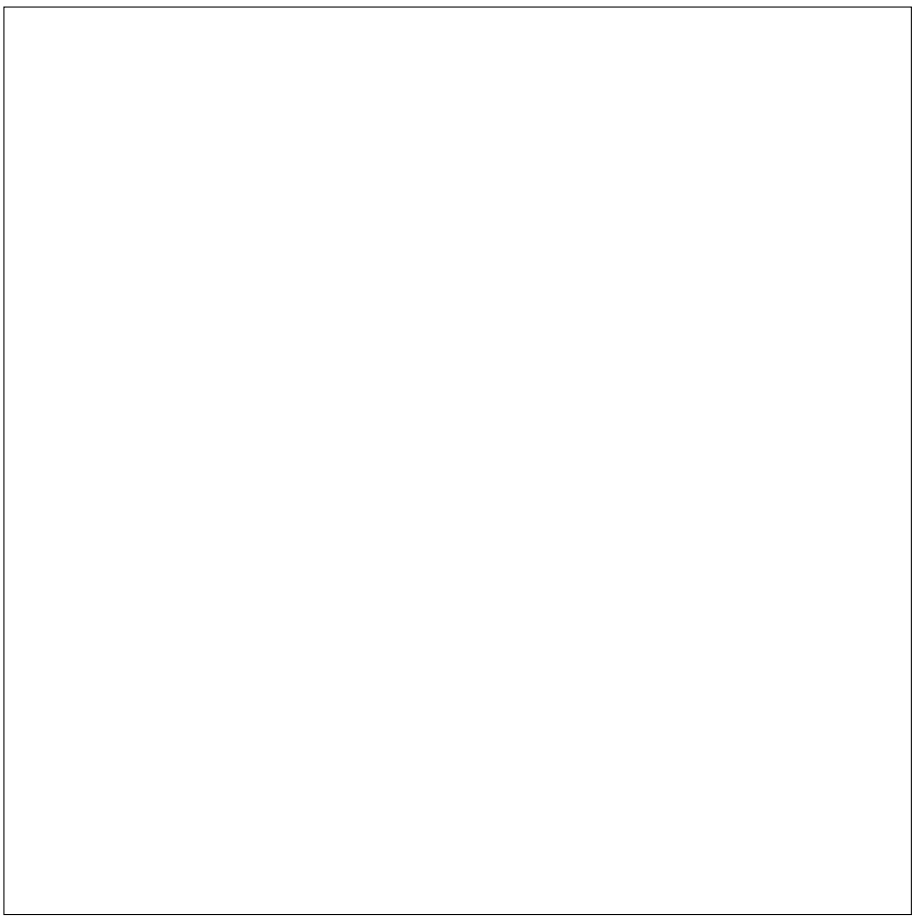
[Wik06] Wikimedia Commons *Timo Boll & Christian Suss*. 2006. URL: https://commons.wikimedia.org/wiki/File:Timo_Boll_%26_Christian_Suss.jpg. veröffentlicht unter CC BY-SA 3.0 (aufgerufen: 07.07.2020).

[Wri+06] Wriggers, P. et al. Kinetik des Massenpunktes. In: *Technische Mechanik kompakt: Starrkörperstatik Elastostatik Kinetik*. Vieweg+Teubner Verlag, Wiesbaden, 2006, 345–373. DOI: 10.1007/978-3-8351-9066-5_19.

[Zha00] Zhang, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334.

Anhang

Datenträger



Auf der Datenträger ist enthalten:

- Dieses Dokument als PDF
- Entwickelte Software
- Logdaten
- Videomaterial