HAW HAMBURG

Bachelorarbeit

Sebastian Bassen

Konstruktion, Simulation und Regelungsimplementierung einer Stewart-Plattform zur Wellenkompensation

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Computer Science and Engineering
Department of Automotive and Aeronautical Engineering

Sebastian Bassen

Konstruktion, Simulation und Regelungsimplementierung einer Stewart-Plattform zur Wellenkompensation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Mechatronik*
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Rasmus Rettig
Zweitgutachter: Prof. Dr.-Ing. Thomas Frischgesell

Eingereicht am: 16. September 2020

Sebastian Bassen

Thema der Arbeit

Konstruktion, Simulation und Regelungsimplementierung einer Stewart-Plattform zur Wellenkompensation

Stichworte

Arbeitsraumberechnung, Kinematik, Abstandsberechnung, Wellenkompensation, Stewart-Plattform

Kurzzusammenfassung

Diese Arbeit umfasst die Bewertung der Konstruktion des Prototyps einer Stewart-Plattform. Dabei wird der Prototyp mathematisch Beschrieben und mit rechnergestützten Verfahren anhand erstellter Anforderungen Bewertet. Die Arbeit beschreibt anschließend die Entwicklung eines Konzepts bis hin zur Implementierung eines Steuer- und Regelungssystem für eine Stewart-Plattform.

Sebastian Bassen

Title of Thesis

Modelling, simulation and control design of a Stewart Platform for wave compensation

Keywords

Workspace calculation, kinematics, distance calculation, wave compensation, stewart-platform

Abstract

This report discribes the prototype evaluation of a stewart-platform design. The prototype is mathematically described and evaluated using computer-aided processes based on the created requirements. The thesis then describes the development of a concept up to the implementation of a control and regulation system for a stewart-platform.

Vorwort

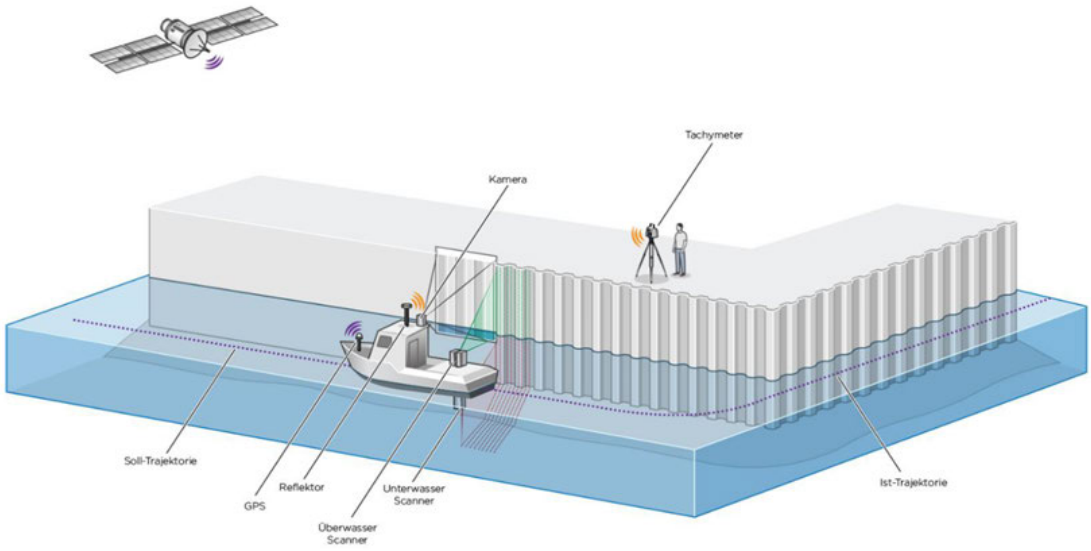
Die vorliegende Arbeit ist im Rahmen meiner Tätigkeit als Werkstudent bei der Firma WKC Hamburg GmbH und der damit verbundenen Mitarbeit am 3D-Hydromapper Projekt entstanden, welches ich im folgenden kurz vorstellen möchte.

Das 3D-Hydromapper Projekt wird seit 2018 vom BMVI durch das Förderprogramm für Innovative Hafentechnologien (IHATEC) unterstützt. Im Rahmen des 3D-Hydromapper Projektes werden neue Mess- und Prüfverfahren für die digitale und automatisierte Bauwerkserfassung und -inspektion entwickelt.

Die beiden Ingenieurbüros WKC Hamburg GmbH und Dr. Hesse und Partner Ingenieure bilden dabei ein Konsortium mit dem Geodätischen Institut der Leibniz Universität Hannover und der Frauenhofer IGP aus Rostock, sowie den zwei großen Infrastrukturbetreibern Niedersachsen Ports und der Wasser- und Schifffahrtsverwaltung des Bundes.

Das von diesem Konsortium entwickelte Multi-Sensorsystem Konzept zur Erfassung von 3D-Punktdaten ist in nachfolgender Abbildung dargestellt. Die integrierte Sensorplattform besteht aus einem hochauflösendem überwasser-Profilscanner, einen schnell messenden hydroakustischen Unterwasserscanner und einer Industriellen 360°-Kamera. Ergänzt wird das System von einer komplementären Positionierungssensorik bestehend aus GNSS, IMU und einer zielverfolgenden Totalstation. Die Totalstation gewährleistet, das auch bei schlechtem oder nicht vorhandenem GPS-Empfang, hervorgerufen durch Verschattung unter Brücken oder im Bereich von hohen Bauwerken, eine präzise Referenzierung des gesamten Messsystems mit einer Genauigkeit von wenigen Millimetern erreicht wird.

Eine besondere Schwierigkeit bei der Aufnahme von Messdaten stellen Querströmungen und Wellen dar. Diese werden durch anderen Schiffsverkehr und Wassereinläufe verursacht und führen dazu, dass die Messplattform in unerwünschte Bewegung versetzt wird. Der in dieser Arbeit untersuchte und weiterentwickelte Prototyp einer Stewart-Plattform, stellt die Basis für eine zukünftige Entkopplung der Sensor- und Schiffsplattform dar.



Konzept 3D-Hydromapper[9]

Ich möchte mich an dieser Stelle bei den beiden Geschäftsführen der 3D-Hydromapper GmbH Herr Karsten Holste und Herr Dr. Christian Hesse für Ihre Unterstützung bei der Entstehung dieser Arbeit bedanken. Besonders hervorzuheben während der Arbeit an diesem Projekt, sind meine beiden Kollegen Pascal und Florian, die jederzeit ein offenes Ohr hatten und für konstruktive Diskussionen gesorgt haben.

Auf Seiten der Lehrenden möchte ich Prof. Dr. Rasmus Rettig für seine Unterstützung und die Betreuung während der Entstehung dieser Arbeit danken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Die Menschen sind schon immer von Robotern fasziniert und der Bedarf an Robotersystemen wächst kontinuierlich. Allein 2017 wurden 381.000 Industrieroboter ausgeliefert. Mit einem Marktanteil von 33% wird von der Automobilindustrie die größte Nachfrage erzeugt [15]. Den überwiegenden Teil stellen hier die industriell eingesetzten seriellen Roboterkonfigurationen, wie sie in Abb. 2.1 dargestellt ist. Die Entwicklung der ersten seriellen Roboter startete im Jahre 1954 mit der Entwicklung des Roboters Unimate. Der Unimate wurde von General Motors nach seiner Entwicklung als erster Schweißroboter eingesetzt [1]. Fast zeitgleich gab es die ersten Entwicklungen von parallel-kinematischen Mechanismen. Diese gehen bereits auf erste Patente aus den 1930er und 1940er Jahre zurück [17]. Die Namensgebung der Stewart-Plattform ist auf Arbeiten von V. E. Gough [6] aus dem Jahre 1956 als Konzept für einen Reifenprüfstand und D. Stewart [18] aus dem Jahre 1965 mit einem Paper zu einer parallel-kinematischen Konstruktion mit sechs Freiheitsgraden zurückzuführen. Heute ergänzen fortschrittliche Roboter unser Leben, sowohl bei der Arbeit als auch zu Hause. Lagerroboter machen die Lieferung an Online-Käufer am nächsten Tag möglich und viele Hausbewohner verlassen sich auf Roboterstaubsauger, die ihre Böden sauber halten. Unter den Industrierobotern stellt die Stewart-Plattform eine häufige Konfiguration von Parallel-Kinematiken dar, auch Hexapod genannt.

1.1 Aufgabenstellung

Konstruktion, Simulation und Regelungsimplementierung einer Stewart-Plattform zur Wellenkompensation.

1.2 Ziel der Arbeit

Im Rahmen dieser Arbeit sollen Anforderungen an den Konstruktionsprototyp einer Stewart-Plattform aufgestellt werden, um seine Konstruktiven Eigenschaften zu beurteilen.

Dabei soll das Arbeitsraumvolumen des Prototyps genauer mit einem iterativen Verfahren bestimmt werden. Die entwickelten mathematischen Zusammenhänge der Stewart-Plattform dienen anschließend als Grundlage für den weiteren Entwurf und die Implementierung eines Steuer- und Regelsystems dienen.

Das System soll dabei Modular aufgebaut werden, um so den leichten Austausch und die einfache Erweiterung des Gesamtsystems zu gewährleisten. Damit wird sichergestellt, dass das System erweiterbar bleibt und künftige Entwicklungen leicht zu integrieren sind.

Zum Abschluss soll das Gesamtsystem basierend auf den entwickelten Anforderung beurteilt und getestet werden.

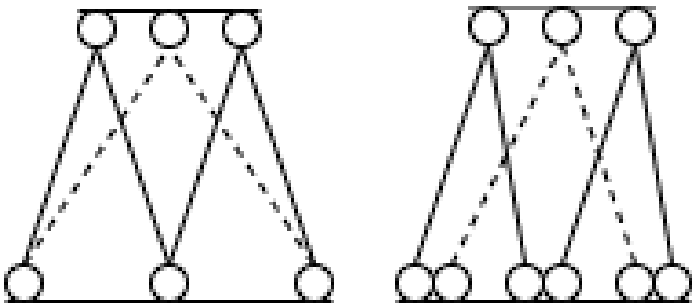
1.3 Stand der Technik

1.3.1 Hardware

Neben den klassischen seriellen Roboterkonfigurationen, wie sie von der Firma Kuka hergestellt werden, sind heute Delta-Roboter (Abb. 1.1) der Hauptanwendungsfall für parallel-kinematische Roboterkonfigurationen. Sie haben ihr Hauptanwendungsgebiet im Bereich Werkzeugmaschinen und in der Verpackungsindustrie[22]. Sie besitzen hier einen Vorteil durch ihre hohe Geschwindigkeit und Steifigkeit und werden dadurch auch immer häufiger in der Medizintechnik eingesetzt[17]. Im privaten Segment finden parallelkinematische Strukturen immer häufiger Anwendung in 3D-Druckern.

Abbildung 1.1: Delta-Roboter der Firma Wittenstein [22]

Die Anordnung, Art und Lagerung der Antriebe stellt auch heute noch ein großes Forschungsgebiet innerhalb der parallel-kinematischen Roboterkonfigurationen dar. So existieren für die Anordnung der Vorschubantriebe zahlreiche Varianten. Diese beziehen sich, wie in Abbildung 1.2 dargestellt, auf die Konstruktion der sechs Gelenkpunkte oder die Art der Antriebsgestaltung Im weiteren Verlauf dieser Arbeit wird eine Stewart-Plattform mit einer 6-6-Konstruktion der Gelenkpunkte untersucht, es existieren aber auch Abwandlungen mit einer 6-3- oder 3-3-Konstruktion. Besonders im Bereich von hochpräzisen Mikro- und Nanopositionierungen sind monolithischen Festkörpergelenke Gegenstand der Forschung [17].



(a) (b)

Abbildung 1.2: Abbildung (a) zeigt den schematischen Aufbau einer 3-3 Gelenk Konfiguration, Abbildung (b) zeigt den schematischen Aufbau einer 6-3 Gelenk Konfiguration

Typischerweise werden bei der Stewart-Plattform sechs Vorschubantriebe parallel, zwischen einer stationären Basis-Plattform und einer Mobilen-Plattform angeordnet. Dadurch wird eine Beweglichkeit in allen sechs Freiheitsgraden (drei translatorische und drei rotatorische) erreicht. Alle Vorschubantriebe arbeiten gleichzeitig um die mobile-Plattform im Raum genau zu positionieren und die Lage einzustellen.

Heute finden Stewart-Plattformen Anwendung in den folgenden Bereichen [19]:

- Aktuatorischer Antrieb von Fahr- und Flugsimulatoren
- Ausrichtung von Teleskopen [4]
- in der Robotik
- in der Medizintechnik als externer Fixateur
- Als Grundelement in Werkzeugmaschinen, besonders beim Zerspanen komplexer Geometrien und Freiformflächen

Ein Vorteil durch die parallele Anordnung der Antriebe bei einer Stewart-Plattformen, verglichen mit einem seriellen Roboter, ist ein besseres Verhältnis von Nutzlast zu Eigengewicht. Weiterhin ist die Reduzierung des Positionierungsfehlers der Mobilen-Plattform von Vorteil, der durch die parallele Anordnung der Antriebe entsteht. Bei seriellen Mechanismen akkumulieren sich die Positionsfehler der einzelnen Antriebe über eine Sequenz von Verbindungen, was diese unbrauchbar für hoch präzise Aufgaben machen kann. Der Positionsfehler einer parallelen Konstruktion ist beschränkt durch den durchschnittlichen Fehler der einzelnen Antriebspositionen. Einzelne Fehler addieren sich nicht auf, wodurch eine sehr hohe Präzision erreicht wird, die selbst Bewegungen im Nanometer-Bereich ermöglicht. Ein Nachteil, der sich aus dem Aufbau ergibt, ist der relativ geringe Arbeitsraum, welcher beschränkt wird durch die minimalen und maximalen Antriebslängen sowie die Dimension und den maximalen Winkel der Kugelgelenke. Die wesentlichen Unterschiede zwischen einer seriellen- und einer parallelen-Kinematik sind in Tabelle 1.1 zusammengefasst.

	Parallele Kinematik	Serielle Kinematik
Abmessungen	Kompakter Aufbau	Relativ großer Aufbau, durch serielles aneinanderreihen von Antriebsachsen
Bewegte Masse	Plattform + Last	Jede Antriebsachse trägt die nachfolgend platzierten + die Last und muss entsprechend ausgelegt werden.
Dynamik /Steifigkeit	Aufgrund der geringen bewegten Masse relativ hoch, für alle Bewegungsachsen gleich	Dynamische Achsen sollten möglichst am Ende der seriellen Kette platziert werden
Genauigkeit	Für alle Bewegungsachsen gleich	Fehleraufsummierung vom Anfang zum Ende, hohe Parallaxenfehler durch Höhe des Aufbaus
Kommandierung	Die Antriebsachsen müssen beim Anfahren einer Pose simultan gesteuert werden	Die Antriebsachsen können Simultan, aber auch nacheinander angesteuert werden
Pivotpunkt	Im Raum frei wählbar	Festgelegt durch die Auswahl und Kombination von Antriebsachsen
Arbeitsraum	Grundsätzlich eingeschränkt	Festgelegt durch die Auswahl und Kombination von Antriebsachsen

Tabelle 1.1: Vergleich einer parallele-n mit einer seriellen-Kinematik[14]

1.3.2 Software

Neben den mechanischen Strukturen findet auch eine kontinuierliche Entwicklung im Bereich der Software für den Betrieb eines Robotersystems statt. Die Entwicklung neuer Roboterplattformen stellt in der Regel einen hohen Aufwand dar. Für einen neu zu entwickelnden Roboter wird häufig versucht standardisierte Hardwarekomponten verwendet. Das soll Wartungs- und Ersatzteilkosten reduzieren. Darüber hinaus kann es die Möglichkeit bieten von einer hohen Verfügbarkeit der Komponenten zu profitieren. Die Integration von Hardware- und Softwarekomponenten kann einen zusätzlichen Kostenfaktor darstellen. Um hier den benötigten Kosten gering zu halten und eine hohe Wiederverwendbarkeit sicherzustellen, kann es sinnvoll sein, auch bei der Software ein standardisiertes System zu verwenden.

Derzeit werden an zahlreichen Standorten sogenannte Metabetriebssysteme zur Steuerung von komplexen und verteilten Systemen entwickelt. Zu diesen Systemen zählen beispielsweise CLARAty und MRDS von der NASA und Microsoft, aber auch zahlreiche weitere wie Marie und ROS [5]. Ein Metabetriebssystem liegt dabei über einem oder mehreren Betriebssystemen. Dabei ergänzt es ein normales Betriebssystem und stellt weitere Werkzeuge und Bibliotheken zum Abrufen, Schreiben, Kompilieren und Ausführen von Code auf mehreren Computern zur Verfügung [16].

Robot Operating System (ROS)

Das in dieser Arbeit verwendete ROS bietet ein Open-Source-Software-Framework für die Entwicklung von Robotern. Ziel ist es die Entwicklung von Robotikanwendungen zu erleichtern. Dabei soll die Möglichkeit geschaffen werden die entwickelten Softwarekomponenten wiederzuverwenden oder auf eine Vielzahl vorhandener Komponenten zurückzugreifen. Dies wird durch eine Abstraktion der Software-seitigen Systemfunktionen in eine höhere Ebene (High-Level) erreicht, während die Ansteuerung von Hardwarekomponenten, wie Sensoren und Aktoren, von Gerätetreibern (Low-Level) realisiert wird.

Die Abstraktion von Low- und High-Level Programmierung in ROS bietet zusätzlich den Vorteil der leichten Anwendungstrennung. Das ermöglicht es, die in der High-Level Ebene erstellten Algorithmen unabhängig von den Gerätetreibern in der Low-Level Ebene zu testen. Dies wird durch die Erstellung sogenannter Bag-Files in ROS erreicht, welche die Systemparameter aufzeichnen, um diese später erneut zu verwenden. ROS bietet darüber hinaus die Flexibilität und Freiheit eigene Softwarekomponenten in einer beliebigen Programmiersprache zu entwickeln. ROS ist darauf ausgelegt neutral gegenüber Programmiersprachen zu sein, so unterstützt ROS Sprachen wie C, C++, Python, Octave, LISP und viele mehr. Außerdem ist ROS darauf ausgelegt wenig Speicher zu benötigen, was bedeutet, dass die vorhandene Komplexität alleine in dem verwendeten Quellcode liegt[16].

Eine Übersicht und Beschreibung mit den Grundlegenden Begriffen innerhalb des ROS-Systems befindet sich in der Anlage dieser Arbeit. Diese Beschreibung wurde in einer zuvor verfassten Arbeit erstellt.

2 Grundlagen

Der Begriff Industrieroboter wird im heutigen Sprachgebrauch als Sammelbegriff für industriell eingesetzte Roboter verwendet. In den meisten Fällen wird mit diesem Begriff ein serieller Roboter, wie er häufig im Automobilbau (Abb. 2.1) zu finden ist, assoziiert. Im folgenden Abschnitt soll daher auf die Unterschiede einer seriellen und parallelen Kinematik eingegangen und spezielle Begriffe und Zusammenhänge im Rahmen dieser Arbeit erläutert werden.

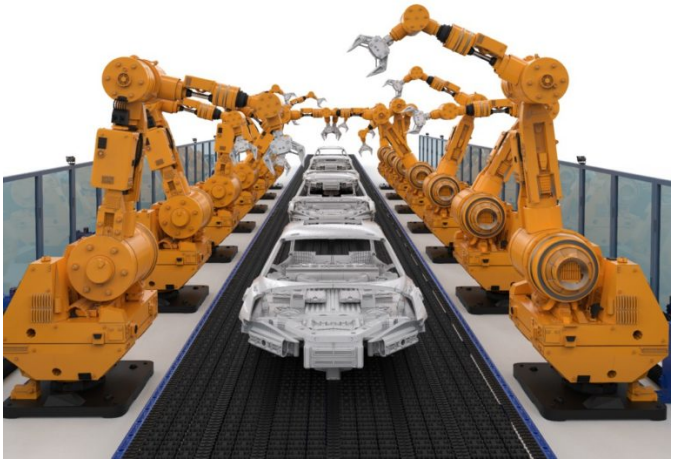


Abbildung 2.1: Fertigungsstrecke im Automobilbau [3]

2.1 Serielle Kinematik

Die Serielle Kinematik beschreibt den klassischen Aufbau einer offenen kinematische Kette. Dies bedeutet, dass jede Baugruppe nur für den Antrieb in einer Bewegungsachse der kinematischen Kette zuständig ist und an nur einer Stelle mit dem nachfolgenden Kettenglied verbunden ist. Das nachfolgende Kettenglied wird dabei immer von der vorherigen

Baugruppe, wie es in Abb. 2.2 dargestellt ist, mitbewegt. Das letzte Glied der Kette ist mit dem Endeffektor verbunden und stellt das Ende der kinematischen Kette dar. Auf diese weise lassen sich beliebige Bewegungsachsen miteinander kombinieren, bis die gewünschte Gesamtbewegung erreicht ist.

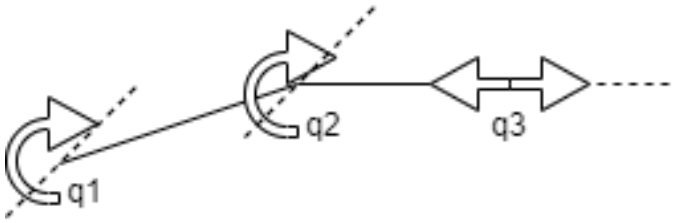


Abbildung 2.2: Schematische Darstellung einer seriellen DDS Konfiguration

2.2 Paralle Kinematik

Die Parallele Kinematik beschreibt eine geschlossene kinematische Kette. Der Endeffektor kann von der Basis aus, auf mehreren Wegen erreicht werden. Die unterschiedlichen Baugruppen sind mit ihren Bewegungsachsen parallel geschaltet. Der Endeffektor ist dadurch, wie in Abb. 2.3 dargestellt, mit mehreren Kettengliedern verbunden.

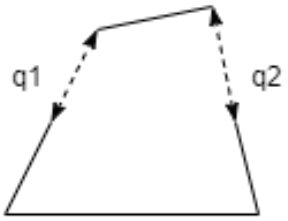


Abbildung 2.3: Schematische Darstellung einer parallelen SS Konfiguration

Ein Vorteil der parallelen Kinematik liegt in der Regel in einer höheren Dynamik. Die einzelnen Baugruppen werden im Vergleich zu seriellen Kinematiken geringer und gleichmäßiger belastet. Dadurch können die einzelnen Baugruppen kleiner ausgeführt werden und weisen eine geringere Masse auf.

2.3 Position und Orientierung im Raum

2.3.1 Pose

Die Pose beschreibt nach DIN EN 8373 in der Robotik die Kombination von Position und Orientierung eines Objektes im 3-dimensionelen Raum. Die Position wird dabei als Punkt entlang der Koordinaten x, y und z definiert und um einen Winkelversatz α, β und γ zum Ursprungskoordinatensystem ergänzt. Die Pose ist damit wie folgt definiert:

$$q := (x, y, z, \alpha, \beta, \gamma)^T \tag{2.1}$$

Bei bekannter Pose kann damit die Postion mit Weglassen der Orientierung direkt aus den Parametern der Pose bestimmt werden, gleiches gilt entsprechend für die Orientierung.

2.3.2 Quaternionen

Das Rechnen mit Quaternionen ist auf William Hamilton aus dem 19. Jahrhundert zurück zu führen. Dieser hat eine Erweiterung der komplexen Zahlen in den 3-dimensionalen Raum gesucht. Das Quaternion Q ist dabei ein 4-Tupel (w, x, y, z) und es gilt:

$$Q := w + x \cdot i + y \cdot j + z \cdot k \tag{2.2}$$

für die imaginären Zahlen i, j und k gilt dabei:

$$i^2 + j^2 + k^2 = ijk = -1 \tag{2.3}$$

Mit dieser Definition kann ein Quaternion als eine komplexe Zahl mit drei unterschiedlichen imaginären Größen i, j und k aufgefasst werden oder als 4-dimensionaler Vektor mit den Komponenten w, x, y, und z. Der imaginäre Teil definiert dabei den Einheitsvektor um der rotiert wird. Der reelle Teil definiert, wie stark die Rotation ist.

Die Berechnung mit Quaternionen ermöglicht die eindeutige Beschreibung einer Orientierung im Raum. Heute werden Quaternionen vorwiegend für interaktive Computergrafiken in Computerspielen verwendet, da ihre Berechnung weniger Rechenzeit erfordert. Eine Rotation durch Quaternionen kann mit vier Elementen beschrieben werden, wohingegen Rotationsmatrizen neun Elemente benötigt.

2.4 Regelungstechnik

Die Regelungstechnik unterscheidet sich von der reinen Steuerung darin, dass sie nicht rein vorwärts gerichtet ist sondern um eine Rückkopplung erweitert ist. Bei einer Rückkopplung wird die Ausgangsgröße überwacht und die Steuerung entsprechend der Differenz zwischen Soll- und Istwert nachgestellt. Der Regelkreis ergibt dabei einen fortlaufenden Kreislauf aus Messen - Vergleichen - Stellen, wie er in Abb. 2.4 dargestellt ist.

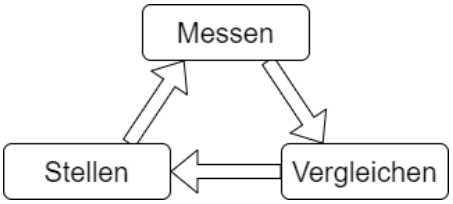


Abbildung 2.4: Prinzip eines Regelkreises

Ein vereinfachter Aufbau eines Regelkreises ist in Abbildung 2.5 dargestellt. Die Zielgröße w wird dem Regelkreis von Außen vorgegeben. Die Regelgröße x wird am Ausgang der Regelstrecke fortlaufend gemessen und an die Regeleinrichtung zurück geführt. Anschließend wird die Differenz aus Zielgröße und Regelgröße gebildet und als Regelabweichung e an den Regler übergeben. Der eigentliche Regler ist dann darauf bestrebt, durch Anpassung der Stellgröße y, die Regelstrecke so zu beeinflussen, das die Regelabweichung verschwindet. Reale Systeme werden darüber hinaus häufig von Störgrößen z beeinflusst. Ihr Einfluss verändert das Verhalten der Regelstrecke und muss vom Regler mit ausgeglichen werden [7].

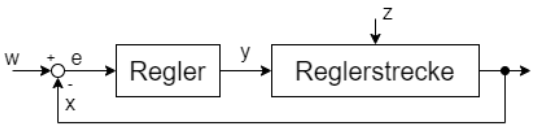


Abbildung 2.5: Aufbau einer Regelkreises

Im Maschinenbau treten dabei häufig die Regelgrößen Kraft, Druck, Drehmoment, Drehzahl und Geschwindigkeit auf.

2.4.1 P-, I- und D-Regler

Die einfachste Art der Regelstrecke stellt in der Regelungstechnik ein sogenannter P-Regler dar. Das P steht hier für Proportionalglied. Bei einem P-Regler wird die Führungsgröße als Eingangsgröße gesetzt und mit der Regelgröße am Ausgang verglichen. Die dabei bestimmte Regeldifferenz wird mit dem vorgegeben Proportionalbeiwert K_p des P-Reglers multipliziert. Das Ergebnis bildet die neue Stellgröße des Reglers. Ein P-Regler hat im Vergleich zu anderen Regelungen einen einfachen Aufbau und sein Ausgangssignal folgt dem Eingangssignal ohne Verzögerung. Ein Problem des P-Regler stellt jedoch seine bleibende stationäre Regelabweichung dar [7].

$$y(t) = K_p \cdot e(t) \tag{2.4}$$

Ein I-Regler stellt einen integrierender Regler dar. Dies bedeutet, er summiert die Regeldifferenz über die Zeit auf. Die Summe der Regeldifferenz wird dann mit dem Integrierbeiwert K_i multipliziert und ergibt damit die neue Stellgröße. Der reine I-Regler ist langsam im Vergleich zu anderen Regelungen, da sich eine Regeldifferenz erst aufsummieren muss. Er bietet dafür jedoch den Vorteil, das eine Regeldifferenz vollständig eliminiert wird.

$$y(t) = K_i \int_0^t e(\tau) d\tau \tag{2.5}$$

Die dritte Grundvariante der Regelkreise stellt der D-Regler dar. Der D-Regler bewertet die Änderungsgeschwindigkeit der Regeldifferenz und multipliziert diese mit dem Differenzierbeiwert K_d. Er differenziert damit die Regeldifferenz. Der D-Regler ist sehr schnell im Vergleich zu anderen Regelungen und kann zur Stabilisierung von anderen Regelkreisen eingesetzt werden.

$$y(t) = K_d \frac{de(t)}{dt} \tag{2.6}$$

Alle drei Regelungsarten können beliebig mit einander kombiniert werden. Geläufige Varianten sind zum Beispiel der PI-, PD- und PID-Regler. Der PID-Regler vereint dabei alle drei Varianten und bietet durch optimale Auslegung der Regelungsparameter das schnelle Ansprechverhalten eines D-Reglers und die eliminierte stationäre Regelabweichung eines I-Reglers. Die Differenzialgleichung der P-, I- und D-Regler ist nachfolgend

als Parallelschaltung gegeben:

$$y(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.7}$$

Das entsprechende Blockschaltbild ist in Abbildung 2.6 dargestellt.

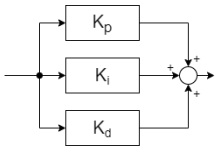


Abbildung 2.6: Prinzip eines PID-Reglers

2.4.2 Kaskadenregelung

Eine komplexe Gesamtregelstrecke kann häufig in kleinere Teilstrecken zerlegt werden. Die gebildeten Teilstrecken können dann einzeln betrachtet und besser optimiert werden. Anschließend werden die Teilstrecken wieder ineinander verschachtelt zusammengeschaltet und bilden dann, wie in Abb. 2.7 dargestellt, eine Kaskadenregelung. Die Stellgröße des äußeren Reglers bildet dabei die Führungsgröße des dahinter liegenden inneren Reglers.

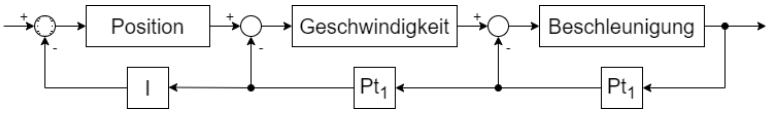


Abbildung 2.7: Prinzip einer Kaskadenregelung

2.4.3 Model Predictive Control (MPC)

Neben den zuvor beschriebenen Regelkreises aus P-, I- und D-Glied zur primären Regelung von single-input, single-output (SISO) Prozessen, existieren weitere Methoden für den Aufbau eines Regelkreises. Bei einer MPC Regelstrecke wird ein zeitdiskretes dynamisches Modell des zu regelnden Prozesses verwendet. Dieses Modell dient dazu, das zukünftige Verhalten in Abhängigkeit zu den Eingangssignalen rechnerisch vorherzusagen. MPC-Regler werden heute häufig in verfahrenstechnischen Prozessen, wie Müllverbrennungsanlagen oder Walzwerken eingesetzt, wo sie eine bessere Regelgüte erreichen[20].

3 Analyse der Anforderungen

3.1 Schiffsbewegung

Es wurden im Rahmen des 3D-Hydromapper Forschungsprojektes Versuchsfahrten mit der Pontonplattform durchgeführt, um diese mit einem herkömmlichen Schiffsrumpf wie beispielsweise bei Sportbooten zu vergleichen. Das hierbei ermittelte Bewegungsprofil ist in nachfolgender Tabelle zusammengefasst:

Bewegung:	Wert:
Roll Geschwindigkeit	$\pm$ 12 °/s
Nick Geschwindigkeit	$\pm$ 12 °/s
Rollwinkel Absolut	$\pm$ 2°
Nickwinkel Absolut	$\pm$ 2°
Heave Beschleunigung	$\pm$ 0,4 m/s^2

3.2 Kinematische Anforderungen an die Stewart-Plattform

Basierend auf dem Bewegungsprofil der Pontonplattform werden für die Stewart-Plattform die zu erreichenden Winkel für den Arbeitsraum mit einem Sicherheitsaufschlag festgelegt. Weiterhin soll das zu stabilisierende Sensorsystem bis zu einem Seegang der Skala 1 eingesetzt werden. Dies entspricht ruhigem Seegang mit einer Wellenhöhe bis 10 cm. Damit ergeben sich folgende geometrische Anforderungen für den Arbeitsraum der Stewart-Plattform:

Bewegung:	Wert:
Wellenhebung	± 10 cm
Rollwinkel	± 5°
Nickwinkel	± 5°
Gierwinkel	± 10°

Zusammen mit den geometrischen Anforderungen an die Stewart-Plattform ergeben sich damit die folgende Gesamtanforderungen:

- Die Mobile-Plattform muss in der Lage sein, den für die Anwendung benötigten Arbeitsraum zu befahren
- Die Linearantriebe müssen in der Lage sein, die benötigten Kräfte und Geschwindigkeiten für die Anwendung zu erzeugen
- Die Ansteuerung der Mobilen-Plattform soll in Echtzeit erfolgen

3.3 Kinematische Modellierung

Um die Bewegung der Stewart-Plattform präzise steuern zu können, wird ein mathematisches Modell benötigt. Das mathematische Modell muss die translatorischen Bewegungen entlang der X-, Y- und Z-Achse sowie die rotatorische Bewegung in Form von Roll, Pitch und Yaw (ϕ θ ψ) als Eingangsdaten der Inertialmesseinheit verarbeiten und in entsprechende Gegenbewegungen der Plattform übersetzen. Die Basis der Stewart-Plattform wird als fest angenommen. Ein für Schiffe häufig genutztes Koordinatensystem ist mit entsprechenden Achsbezeichnungen in Abb. 3.1 dargestellt und wird als Grundlage für die Stewart-Plattform genutzt.

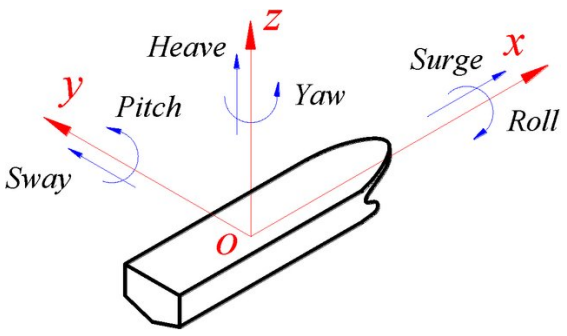


Abbildung 3.1: Koordinatensystem der Messplattform [21]

Eine Stewart-Plattform besteht aus einer festen Basis-Plattform und einer bewegten Mobilen-Plattform, welche von sechs Linearantrieben miteinander verbunden werden. Das Gelenk zwischen Basis-Plattform und Linearantrieb (B_i) ist als Kreuzgelenk ausgeführt. Das Gelenk zwischen Linearantrieb und Mobiler-Plattform (P_i) ist um einen dritten Freiheitsgrad erweitert und als sphärischen Gelenk ausgeführt.

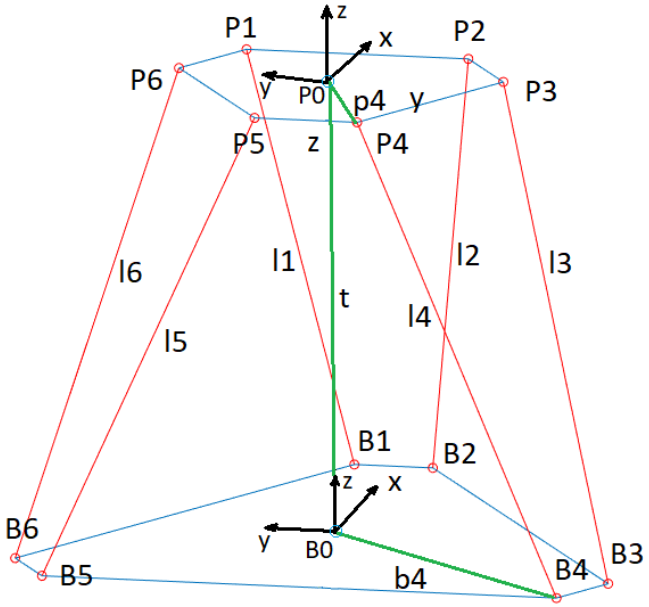


Abbildung 3.2: Prinzipskizze der Stewart-Plattform

Wie in Abb. 3.2 dargestellt, befindet sich das Koordinatensystem B_0 zentral in der Basis-Plattform der Stewart-Plattform. Das Koordinatensystem P_0 der Stewart-Plattform befindet sich zentral in der Mobilen-Plattform. Die Pose der Mobilen-Plattform im Koordinatensystem B_0 wird mit $\mathbf{q} = (\mathbf{t}, \mathbf{R})$ beschrieben. Die Koordinaten $\mathbf{t} = [x, y, z]^T$ beschreiben die kartesische Position der Mobilen-Plattform. $\mathbf{R}$ ist eine 3×3 Rotationsmatrix und beschreibt die Orientierung der Mobilen-Plattform P_0 relativ zur Basis B_0 im Raum .

Der Vektor p_i beschreibt die Position des Mittelpunktes eines sphärishen Gelenks, welches an der Mobilen-Plattform befestigt ist, mit i=1..6. Der Vektor b_i beschreibt die Position eines Kreuzgelenks das an der Basis befestigt ist, mit i=1..6. Der geometrische Zusammenhang zwischen den Befestigungspunkten der Linearantriebe kann dafür verwendet, werden die Längen der Linearantriebe zu berechnen und die inverse Kinematik abzuleiten.

3.3.1 Direkte Kinematik

In der direkten Kinematik wird die Position und Orientierung der Stewart-Plattform aus den Gelenkkoordinaten bestimmt. Diese Berechnung ist für serielle Roboter einfach lösbar, für parallel-kinematische Ketten wie bei einer Stewart-Plattform ist die direkte Kinematik im Allgemeinen nicht geschlossen lösbar. Im Gegensatz zu offenen kinematischen Ketten sind bei parallel-kinematische Ketten nicht mehr alle Eingangsgrößen voneinander unabhängig[10].

Für die Bestimmung der direkten Kinematik muss zuerst die Länge der Linearantriebe als Eingangsgröße bestimmt werden:

$$\| \mathbf{l}_i \| = \sqrt{(\mathbf{t}(\mathbf{q}) + \mathbf{R}(\mathbf{q})\mathbf{p}_i - \mathbf{b}_i)^2} \qquad \text{für i=1..6} \tag{3.1}$$

Zur Bestimmung der direkten Kinematik müssen alle sechs Koordinaten $\mathbf{q}$ eines nichtlinearen Gleichungssystem bestimmt werden, für die es keine geschlossene Lösung gibt. Daher lässt sich die Position und Orientierung nur über einen iterativen Lösungsansatz bestimmen. Erschwert wird die Bestimmung, da bei gegebenen $\mathbf{l}_i$ mehrere $\mathbf{q}$ zur Lösung existieren. Insgesamt sind bis zu 40 Konfigurationen möglich [10].

Ein Beispiele für die zahlreichen Konfigurationen ist die Spiegelung unterhalb der Basis einer Stewart-Plattform. Dieser Bereich ist physikalisch nicht zu erreichen, stellt aber eine mathematische Lösung dar.

3.3.2 Inverse Kinematik

Bei der inversen Kinematik werden die Beinlängen aus der vorgegebenen Pose der Mobilen-Plattform bestimmt. Sie besteht aus sechs nicht linearen Gleichungen, welche eindeutig gelöst werden können. Die Länge eines Linearantriebs l_i entspricht dabei dem Abstand der unteren und oberen Gelenkpunkte und kann mit folgender Gleichung berechnet werden:

$$\mathbf{l}_i = \mathbf{t} + \mathbf{R}\mathbf{p}_i - \mathbf{b}_i \qquad \text{für i=1..6} \tag{3.2}$$

mit i = 1,...,6 wird der jeweilige Linearantrieb gekennzeichnet. Die Position des Mittelpunktes der oberen Mobilen-Plattform wird durch den Vektor $t = [x_t y_t z_t]^T$ beschrieben.

Der Betrag der Vektorgleichung beschreibt die Länge des Linearantriebes:

$$l_i = \| \mathbf{l}_i \| = \| \mathbf{t} + \mathbf{R}\mathbf{p}_i - \mathbf{b}_i \| \qquad \text{für i=1..6} \tag{3.3}$$

Für eine Pose mit vorgegebener Position (x_t, y_t, z_t) und Orientierung (ϕ, θ, ψ) der Plattform stellt diese Gleichung auch die Lösung der inversen Kinematik dar.

3.3.3 Jacobimatrix

Die Jacobi-Matrix wird in der Robotik genutzt, um die benötigten Geschwindigkeiten und Kräfte einer kinematischen Kette zu bestimmen. Darüber hinaus spielt sie eine zentrale Rolle bei der Bestimmung von Singularitäten und soll für weitere Analysen bestimmt werden.

Der Vektor t beschreibt die kartesischen Koordinaten der Mobilen-Plattform relativ zur Basis. Mit dem Term Rp_i werden die Koordinaten von p_i mithilfe der Rotationsmatrix R in das globale Koordinatensystem mit der X-Y-Z Konvention überführt.

Rotation um die X-Achse:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.4}$$

Rotation um die Y-Achse:

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta)]; \end{bmatrix} \tag{3.5}$$

Rotation um die Z-Achse:

$$R_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Berechnung der Gesamtrotationsmatrix nach X-Y-Z Konvention:

$$R = Rx \cdot Ry \cdot Rz \tag{3.7}$$

einsetzen und ausmultiplizieren ergibt:

$$R = \begin{bmatrix} c(\theta) \cdot c(\psi) & -c(\theta) \cdot s(\psi) & s(\theta) \\ c(\phi) \cdot s(\psi) + c(\psi) \cdot s(\phi) \cdot s(\theta) & c(\phi) \cdot c(\psi) - s(\phi) \cdot s(\theta) \cdot s(\psi) & -c(\theta) \cdot s(\phi) \\ s(\phi) \cdot s(\psi) - c(\phi) \cdot c(\psi) \cdot s(\theta) & c(\psi) \cdot s(\phi) + c(\phi) \cdot s(\theta) \cdot s(\psi) & c(\phi) \cdot c(\theta) \end{bmatrix} \tag{3.8}$$

mit s = sinus und c = cosinus.

Die inverse Jacobi Matrix beschreibt den Zusammenhang zwischen Gelenk-Geschwindigkeiten und Arbeitsraum-Geschwindigkeiten. Gleichung 3.2 stellt den Zusammenhang der Beinlänge l_i zur Pose q, wobei t die kartesischen Koordinaten $[x, y, z]$ enthält und die Rotationsmatrix R Orientierungswinkel $[\phi, \theta, \psi]$ der Pose von q enthält.

$$\frac{\delta l}{\delta t} = \frac{\delta l}{\delta q} \frac{\delta q}{\delta t} \tag{3.9}$$

Die inverse Jacobi Matrix J_q wird dann als $J_q = \frac{\delta l}{\delta q}$ eingesetzt, woraus sich folgende Gleichung ergibt:

$$\frac{\delta l}{\delta t} = J_q \frac{\delta q}{\delta t} \quad \Rightarrow \quad \dot{l} = J_q \dot{q} \tag{3.10}$$

$\dot{l}$ enthält die Geschwindigkeiten der Linearantriebe und $\dot{q} = [\dot{t}\, \omega]^T$ enthält die Geschwindigkeiten $\dot{t}$ und Winkelgeschwindigkeiten ω der Mobilen-Plattform. Um das Ableiten der Jacobi Matrix für die Stewart-Plattform weiter zu vereinfachen wird die Gleichung 3.2 um die Einheitsvektoren e_i mit $i = 1..6$ erweitert, daraus folgt:

$$l_i e_i = t + Rp_i - b_i \qquad \text{für i=1..6} \tag{3.11}$$

Differenzierte Jacobi Matrix:

$$\dot{l}_i e_i + l_i \dot{e}_i = \dot{t} + (\omega \times Rp_i) \tag{3.12}$$

Durch das Umstellen von Gleichung 3.12 unter Berücksichtigung der Regeln $e_i * e_i = 1$ und $e_i \dot{e}_i = 0$. ergibt sich:

$$\dot{l}_i = e_i \dot{t} + (Rp_i \times e_i) \cdot \omega \tag{3.13}$$

Die 6×6 Jacobi Matrix ist gegeben mit:

$$J_q(q) = \begin{bmatrix} {e_1}^T & (Rp_1 \times e_1)^T \\ \cdots & \cdots \\ {e_6}^T & (Rp_6 \times e_6)^T \end{bmatrix} \tag{3.14}$$

Die inverse Jacobi Matrix J_q ist eine Funktion der Pose q. Unter der Bedingung, dass die inverse Jacobi Matrix einen vollen Rang besitzt, kann über ihre Dimension abgeleitet werden, das die Stewart-Plattform sechs Freiheitsgerade (DOF) und sechs prismatische Verbindungen besitzt. Die Anzahl der Zeilen beschreibt hierbei die Anzahl der Freiheitsgrade im Raum und die Anzahl der Spalten beschreibt die Anzahl prismatischen Verbindungen. Die ersten drei Spalten der inversen Jacobi Matrix beschreiben außerdem die translatorische Bewegung der Mobilen-Plattforms und die letzten 3 Spalten beschreiben die rotatorische Bewegung der Mobilen-Plattforms.

3.3.4 Statische Kräfte

Das statische Modell der Stewart-Plattform kann umgeschrieben werden, um die Kräfte in den Linearantrieben zu bestimmen:

$$\tau = {J_q}^T(q)g \tag{3.15}$$

Dabei Beschreibt $\tau = [F\,M]^T$ die Gesamtkräfte und -momente, welche auf die Mobile-Plattform einwirken. Das Umstellen der Gleichung liefert:

$$g = {J_q}^T(q)^{-1}\tau \tag{3.16}$$

3.3.5 Isotropie

Ein kinematisches System ist isotrop, wenn in mindestens einem Punkt des beschriebenen Arbeitsraumes kein homogenes Verhalten für eine der kinematischen Eigenschaften vorliegt. Die Isotropie ist eine Eigenschaft der Jacobi Matrix und wird mathematisch nach folgender Gleichung definiert:

$${J_q}^T J_q = \lambda l_{6x6} \tag{3.17}$$

Nach Gleichung 3.17 ist zu erkennen, das die Eigenwerte λ für das System gleich sein müssen um isotrop zu sein. Ein kinematisches System kann isotrop für die Werte Geschwindigkeit, Kraft und Steifigkeit sein.

3.4 Kinematische Performance zur Wellenkompensation

Kein Freiheitsgrad darf bei einer Pose innerhalb des Arbeitsraumes eingeschränkt werden, da die Stewart-Plattform andernfalls ihrer Aufgabe nicht mehr nachkommen kann.

Ein Freiheitsgrad wird eingeschränkt, wenn einer der singulären Werte 0 wird. Ein spezielles kinematisches Design kann singuläre Werte nahe 0 besitzen, welche innerhalb des Arbeitsraumes dazu führen, dass die Geschwindigkeit des Aktuators sehr groß werden muss um die Mobile-Plattform in die gewünschte Richtung zu bewegen. Mit einem anderen kinematischen Design können die singuläre Werte sehr groß werden was dazu führt, dass nur kleine Kräfte in die gewünschte Richtung erzeugt werden können. Dies kann dazu führen, dass die gewünschte Aufgabe nicht wie gefordert ausgeführt werden kann.

Manipulierbarkeit ist das Produkt aller singulären Werte und die singulären Werte sind das Verhältnis zwischen Kraft und Geschwindigkeit des Aktuators zum End-Effektor:

$$J_q{}^{-1}\dot{l} = \dot{q} \tag{3.18}$$

3.5 Analyse der verwendeten Stewart-Plattform

Der im Folgenden näher untersuchte Prototyp einer Stewart-Plattform wurde im Vorfeld entwickelt. Die Geometrien der Stewart-Plattform sollen genau bestimmt werden und das kinematische Verhalten beurteilt werden. Die geometrischen Eigenschaften des Prototyp sind in nachfolgender Tabelle aufgelistet. Die zugehörigen CAD-Zeichnungen sind der Anlage dieser Arbeit angefügt.

Design Parameter:	**Wert**
Base Radius R_b	408 mm
Plattform Radius R_p	210 mm
Öffnungswinkel des Beinpaares an Basis	14 deg
Öffnungswinkel des Beinpaares am End-Effektor	84 deg
Minimale Beinlänge	877 mm
Maximale Beinlänge	1334 mm

Tabelle 3.1: Geometrische Daten

3.6 Analyse des Arbeitsraumes

Der Arbeitsraum einer Stewart-Plattform wird auf mehrere Arten in seinen effektiv nutzbaren Bereichen eingeschränkt. Die wesentlichen Beschränkungen sind nachfolgend aufgelistet:

1. Minimale und maximale Länge der Linearantriebe
2. Winkelgrenzen der Gelenke
3. Kollisionen zwischen den Linearantrieben
4. Singuläre Stellungen der Stewart-Plattform

Um den Arbeitsraum geometrisch zu untersuchen, kann Gleichung 3.2 der inversen Kinematik zur Berechnung der Beinlängen in skalare Form umgeschrieben werden:

$$l_i^2 = (t_x - u_{xi})^2 + (t_y - u_{yi})^2 + (t_z - u_{zi})^2 \tag{3.19}$$

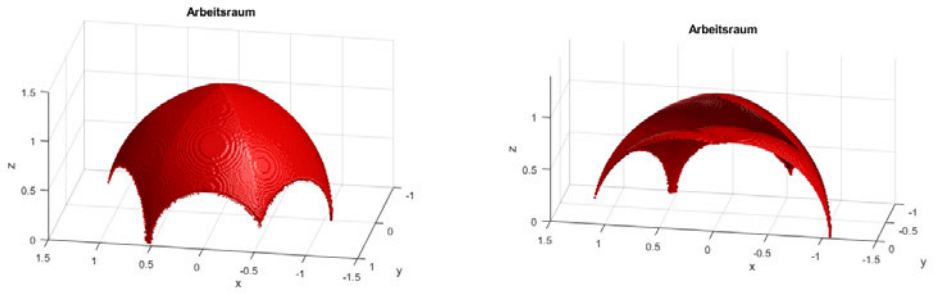
Diese Gleichung entspricht geometrisch betrachtet einer Kugelgleichung. Für die weitere Untersuchung des Arbeitsraumes wurde ein Matlab-Skript erstellt. In diesem Skript werden die möglichen Position der Mobilen-Plattform durch diskrete Punkten innerhalb eines definierten Volumen dargestellt. Anschließend werden die benötigten Beinlängen für die zu untersuchende Position berechnet und geprüft, ob diese Position angesteuert werden kann. Die Gleichung zu Prüfung einer validen Beinlänge, ist wie folgt definiert:

$$l_{min} \leq l_i \leq l_{max} \tag{3.20}$$

Ist diese Gleichung erfüllt, wird keine mechanische Limitierung durch minimale oder maximale Beinlängen verletzt. Befinden sich alle sechs Längen der Linearantriebe innerhalb der Grenzen dieser Gleichung, wird der Punkt als Teil des Arbeitsraumes gewertet. Eine Überprüfung auf Kollisionsfreiheit zwischen den Linearantrieben erfolgt bei dieser Prüfung nicht.

Die Ergebnisse der Arbeitsraum Berechnung sind in Abb. 3.3 dargestellt.

Der Arbeitsraum des zu untersuchenden Prototyps wird nur in begrenztem Maße genutzt. Weiterhin ist Konstruktionsbedingt sichergestellt, das in dem begrenzt genutzten Arbeitsraum keine Kollisionen zwischen den Antrieben stattfinden kann. Konstruktionsbedingt werden auch die Gelenke der Stewart-Plattform nicht in ihren Winkelgrenzen

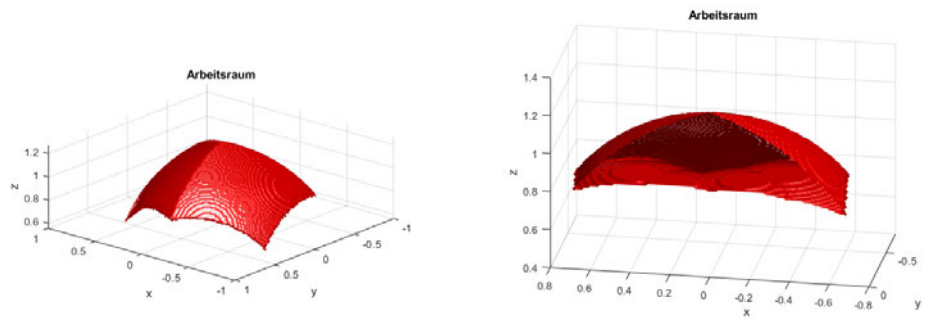


(a) (b)

Abbildung 3.3: Abbildung (a) zeigt das Volumen des Arbeitsraumes der zu untersuchenden Stewart-Plattform, Abbildung (b) zeigt das Volumen des Arbeitsraumes als Schnitt entlang der y-Achse

bewegt. Die einzige verbleibende Limitierung des Arbeitsraumes stellen damit die minimalen und maximalen Längen der Linearantriebe dar. Da die Bewegungsberechnung in Echtzeit erfolgen soll, wird auf eine rechenaufwendige Überprüfung von Singularitäten mithilfe der Jacobi-Matrix verzichtet.

Um die Reduzierung des zur Verfügung stehenden Arbeitsraumes zu visualisieren, ist in nachfolgender Abbildung der Arbeitsraum der Stewart-Plattform bei einem Yaw-Winkel von 45° dargestellt. Diese Konfiguration wird in der Praxis durch Limitierungen in der Software nicht erreicht. Sie zeigt aber, das selbst in dieser Konfiguration noch eine Heave Bewegung entlang der Z-Achse von 43cm möglich ist.



(a) (b)

Abbildung 3.4: Abbildung (a) und (b) zeigen das reduzierte Volumen des Arbeitsraumes der zu untersuchenden Stewart-Plattform bei einem Yaw-Winkel von 45°

4 Systementwurf

4.1 Entwurf des Gesamtsystems

Das System zur Steuerung der Stewart-Plattform soll, wie in Abbildung 4.1 dargestellt, verteilt auf einem Single-Board-Computer (SBC) und mehreren Mikrocontrollern erfolgen.

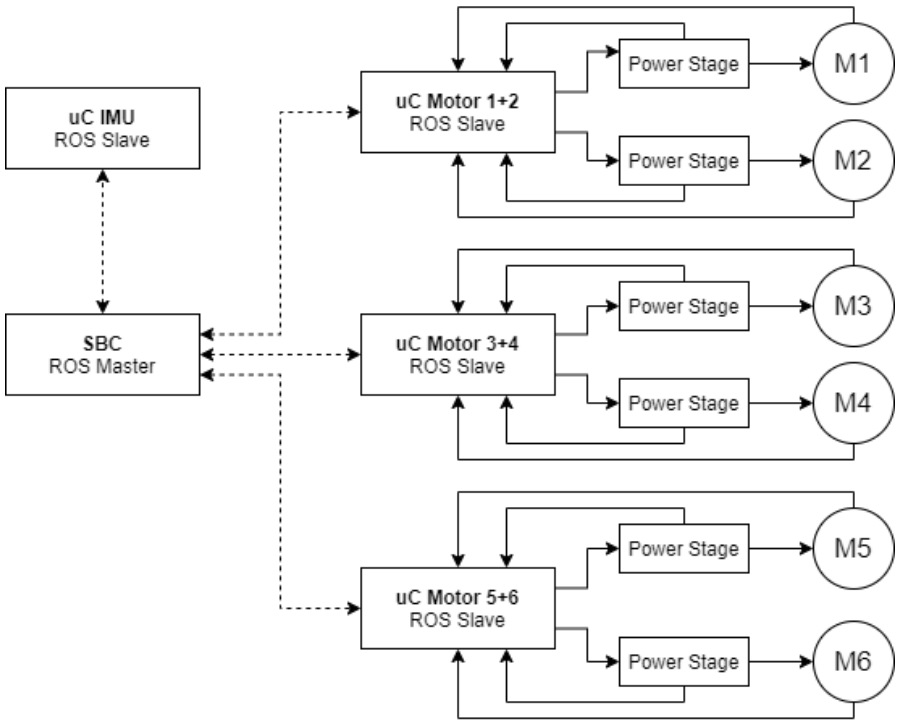


Abbildung 4.1: Blockdiagramm zur Darstellung des Gesamtsystems

Der ROS Master übernimmt dabei die Aufgabe die Systemdaten zusammen zuführen. Basierend auf den ermittelten Daten der IMU wird die Pose der Mobilen-Plattform be-

stimmt. Ist die Pose bekannt, kann mithilfe der Gelenkkoordinaten die benötigte Beinlänge der Linearantriebe bestimmt werden. Die Beinlänge wird anschließend im System veröffentlicht und von den zugehörigen ROS-Slave Mikrocontrollern abgerufen. Die Mikrocontroller übersetzen die empfangenen Nachrichten in ein für die Motorendstufen verständliches Steuersignal.

4.2 Verwendete Hardware

4.2.1 ROS Slave

Für die Ansteuerung und Regelung der Linearantriebe wird ein ESP32 Mikrokontroller der Firma Espressif verwendet. Wie in Abb. 4.2 dargestellt, besitzt der verwendete Mikrocontroller 38 I/O Pins. Davon sind 4 als Masse-Verbindungen und je einer als 3,3 V und 5 V Spannungsein- oder ausgang ausgeführt. Der Mikrocontroller kann mit einem einstellbaren Systemtakt von 160 MHz - 240 MHz betrieben werden. Der interne RAM-Speicher des Mikrocontroller besitzt eine Größe von 160 KiB static allocated und 160 KiB dynamic allocated DRAM. Ergänzt wird der RAM-Speicher von einem internen Festwertspeicher (ROM), welcher zusätzlich den Bootloader enthält. Der verwendete Mikrocontroller besitzt darüber hinaus diverse Peripherieschnittstellen wie zum Beispiel UART-, SPI-, CAN- und I^2C-Schnittstellen. Für die kabellose Anbindung enthält der Mikrocontroller darüber hinaus eine WLAN- und Bluetooth-Schnittstelle.

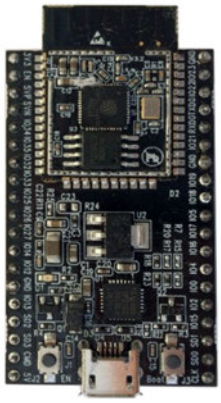


Abbildung 4.2: ESP32 Mikrocontroller zur Ansteuerung der Linearantriebe

4.2.2 ROS Master

Als zentraler Steuerungscomputer für die Ausführung des "Robot Operating System"(ROS) wird ein Rock Pi 4 Singleboard-Computer (SBC) verwendet. Der Rock Pi 4 verfügt über einen RockChip RK3399 Prozessor, welcher 6 Kerne in der Big.Little Architektur vereint. Der RK3399 Prozessor beinhaltet dabei zwei Cortex-A72 Kerne mit 1,8 Ghz Takt in der Big Architektur und vier Cortex-A53 Kerne mit 1,4 Ghz in der Little Architektur. Als Arbeitsspeicher stehen dem System 4 Gb LPDDR4 zur Verfügung sowie eine Mali T860MP4 als Grafikprozessor.

Abbildung 4.3: Rock Pi 4 als zentraler Steuercomputer und ROS Master mit einer Erweiterung für m.2 SSD's

Ähnlich wie das bekanntere Raspberry Pi Board stellt der Rock Pi 4 40 GPIO-Pins zur Verfügung. Die GPIO-Pins bieten dabei folgende erweiterte Funktionen:

- i^2C
- UART
- SPI
- PWM
- SPDIF
- ADC (1,8 V)

Die GPIO-Pins haben drei IO-Spannungspegel, diese liegen bei 1.8 V, 3.0 V und 3.3 V. Wichtig ist dies besonders für den ADC welcher mit einer Spannung von 1.8 V arbeitet.

Die Spannungsversorgung wird über einen USB-C Anschluss sichergestellt. Neben zwei USB 2.0 und zwei USB 3.0 Anschlüssen, verfügt der Rock Pi 4 über einen 1 Gbe Ethernet Port, HDMI Ausgang, ein RTC Modul sowie CSI und DSI und kann neben einem eMMC

Modul auch eine m.2 NVME SSD aufnehmen. Als Funkverbindung stehen WIFI 802.11 ac und Bluetooth 5.0 zur Verfügung.

4.2.3 Linearantriebe

Die verwendeten Linearantriebe stammen von der Firma Moteck und wurden für die Anwendung in der Stewart-Plattform modifiziert. Die Modifizierung betrifft die obere und untere Aufnahme der Linearantriebe. Diese ist nicht wie in Abbildung 4.4 dargestellt mit einer Bohrung versehen, sondern ist als gerades M16x1,5 Gewinde ausgeführt. Damit lassen sich die Linearantriebe direkt mit den oberen und unteren Gelenken verschrauben.

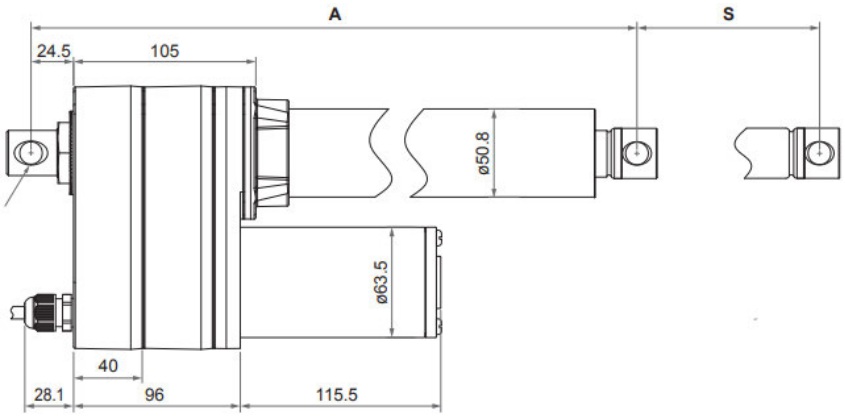


Abbildung 4.4: Konstruktionszeichnung des eingesetzten Moteck ID10 [12]

Dem Datenblatt der Moteck Linearantriebe können folgende Parameter entnommen werden:

- Input voltage: 48 V DC
- Input current: 5.5 A at full load
- Max. rated load: 7,000 N (Ball Screw)
- Max. static load: 13,600 N (Ball Screw)
- Gear ratio: 5:1

- Stroke: 457 mm (18")
- Max force Push/Pull: 2500 N
- Typical speed at no load: 72.1 mm/sec
- Typical speed at full load: 57.5 mm/sec
- IP Protection level: IP54
- Certified: CE Marking, EMC Directive 2014/30/EU

4.2.4 Motorcontroller

Als Leistungsendstufen zur Ansteuerung der Linearantriebe werden 4-Q Servokontroller für DC/EC-Motoren der Firma Escon eingesetzt. Parameter der Leistungsendstufen:

Abbildung 4.5: Leistungsentstufe 4-Q Servokontroller für DC/EC-Motoren der Firma Escon [11]

- Nenn-Betriebsspannung (+V_{cc}): 10-50 VDC
- Ausgangsspannung (max.): 0.98 x +VCC
- Ausgangsstrom I_{cont} / I_{max} (<20 s): 5 A / 15 A

4.2.5 Inertial Measurement Unit (IMU)

IMU's enthalten in der Regel ein ganzes Sensorpaket und sind heutzutage als integrierte Schaltungen (SoC) leicht erhältlich. Dieses Sensorpaket besteht in der Regel aus drei Beschleunigungssensoren und drei Gyroskopen. Häufig wird dieses Sensorpaket wie bei der

Bosch BNO055 noch um ein Magnetometer erweitert. Die translatorische Bewegung kann in allen drei Achsen durch die Beschleunigungssensoren bestimmt werden und mithilfe der Gyroskope kann die rotatorische Bewegung, in Form einer Drehrate, um alle drei Achsen bestimmt werden. Das Magnetometer erfasst die Ausrichtung zum Erdmagnetfeld und kann separat ausgelesen oder zur Referenzierung der Lagebestimmung genutzt werden. Damit können Bewegungen in allen sechs Freiheitsgraden (DoF = Degree of Freedom) erfasst und bestimmt werden, wie es in Abb. 4.6 dargestellt ist.

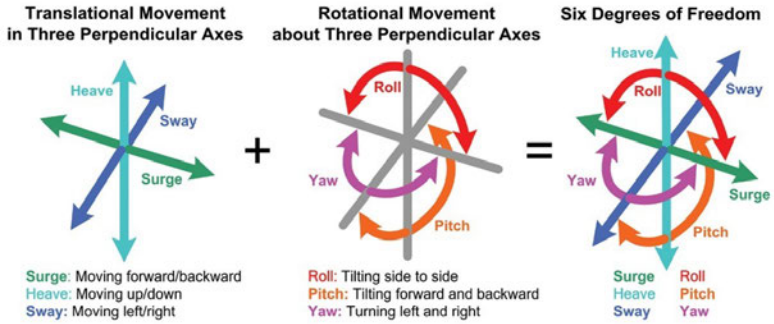


Abbildung 4.6: Darstellung der 3-Achsen und ihrer Freiheitsgrade mit zugehörigen englischen Bezeichnungen [8]

Die BNO055 IMU ist als "System-in-Package" aufgebaut und verfügt neben den drei inertial Sensoren auch über einen integrierten 32-bit Cortex M0+ Microcontroller. Der Microcontroller enthält einen integrierten Fusionsalgorithmus und bietet die Möglichkeit neben den Rohdaten auch die Orientierung in Eulerwinkeln oder Quaternionen auszugeben, den Gravitationvektor zu bestimmen und die gavitationsbereinigte Beschleunigung auszugeben. Die Sensorfusion erfolgt dabei in Echtzeit und kann mit bis zu 100 Hz über eine I2C Schnittstelle ausgelesen werden. Auch die Gyroskop- und Magnetometerdaten werden bei aktiviertem Fusionsmodus bereinigt. Hierfür führt die IMU interne Selbsttests und Echtzeitkalibrierungen durch. Diese sind im Rohdatenmodus allerdings nicht verfügbar [2].

IMUs haben jedoch auch einige Nachteile. Insbesondere die kostengünstigeren MEMS-Bausteine sind zum Teil unpräzise und ihre Daten rauschen mehr oder weniger stark. Problematisch können bereits sehr kleine Messfehler sein. Das Problem liegt hier in der Integration der Messwerte, da der Messfehler für eine Geschwindigkeit einmal integriert wird und zu Bestimmung der Position ein zweites mal. Ein qualitativ hochwertiger Sensor und eine genaue Kalibrierung sind deshalb unverzichtbar.

Beschleunigungssensor

Die BNO055 IMU besitzt einen Beschleunigungssensor mit drei DoF und einer Auflösung von 14 Bit. Der Messbereich kann bei dieser IMU auf ±2 g, ±4 g, ±8 g oder ±16 g konfiguriert werden. Darüber hinaus kann die gemessene Einheit auf m/s^2 oder mg eingestellt werden

Drehratensensor

Der Drehratensensor der BNO055 IMU misst die Drehrate ebenfalls um drei Achsrichtungen. Der Messbereich kann beim Drehratensensor zwischen ±125 °/s und ±2000 °/s eingestellt werden und verdoppelt sich bei jedem nächsthöheren Messbereich. Die Auflösung beträgt beim Drehratensensor 16 Bit und kann auf die Einheit °/s oder rad/s eingestellt werden.

Magnetometer

Ergänzt werden der Beschleunigungs- und Drehratensensor von einem Magnetometer mit drei DoF. Die Messauflösung beträgt 13 Bit in der x- und y-Achse und 15 Bit in der z-Achse. Die Messwertausgabe der magnetischen Flussdichte erfolgt in μT.

4.3 Softwarearchitektur

Die Umsetzung der Steuerungsprogrammierung soll mit dem Robot Operating System realisiert werden. Hierzu sollen entsprechende Nodes programmiert und ihre Daten im System bereit gestellt werden. Die Struktur der geplanten ROS Nodes und die für ihre Kommunikation benötigten Topics sind in Abbildung 4.7 dargestellt.

Die Node Base Broadcaster ist für die Berechnung und Veröffentlichung der Koordinaten der Basis Plattform und den an ihr angebrachten Gelenkkoordinaten zuständig. Die Berechnung der Koordinaten der oberen Mobilen-Plattform mit den an ihr angebrachten Gelenken wird von der Plattform Node durchgeführt. Das Einlesen der Bewegungsdaten, welche von der BNO055 IMU geliefert werden, erfolgt durch die IMU Broadcaster Node. Alle drei Nodes veröffentlichen ihre Daten über die ROS interne tf2 Node.

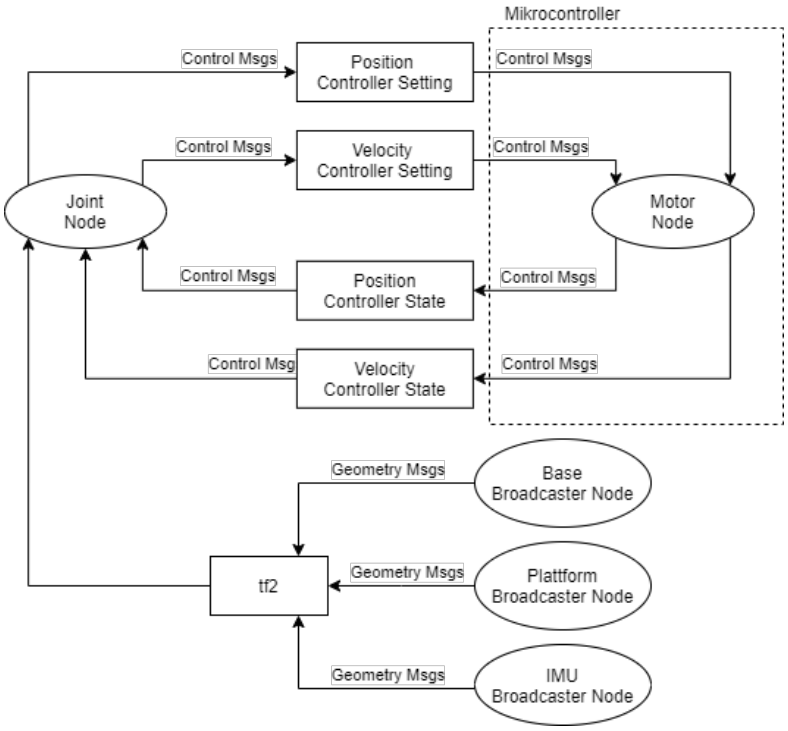


Abbildung 4.7: Blockdiagramm zur Darstellung der internen Kommunikation

Die Joint Node liest die Koordinaten der oberen und unteren Beingelenke ein und berechnet die Koordinatenabstände der Gelenke. Mithilfe der Koordinatenabstände können die Längen der Linearantriebe bestimmt und veröffentlicht werden. Die Joint Node veröffentlicht ihre Nachrichten in den Topics Position Controller Setting und Velocity Controller Setting. Neben den Längen der Linearantriebe sind in diesen Topics auch informationen über die Regelungsparameter der einzelnen Linearantriebe enthalten. Damit ist es möglich die Regelungsparamter auch während des Betriebs anzupassen.

Als Grundlage für die Ansteuerung der Linearantriebe soll in der Motor Node ein PID-Regler implementiert werden. Die in Gleichung 2.7 beschrieben Differenzialgleichung wird dafür in die nachfolgende Differenzengleichung überführt:

$$y_k = K_p \cdot e_k + K_i \cdot T_a \sum_{i=0}^{k} e_i + \frac{K_d}{T_a}(e_k - e_{k-1}) \tag{4.1}$$

Der implementierte PID-Regler soll sich mit Hilfe der Controller Setting Topics während des Betriebes anpassen lassen. Der Zustand des Linearantriebes und die aktuellen Regelungsparameter werden von der Motor Node über die Topics Position Controller State und Velocity Controller State veröffentlicht.

Die Implementierung als PID-Regler bietet zusätzlich den Vorteil, das durch Null setzen der einzelnen Beiwerte zu jederzeit eine beliebige Konfiguration aus P-, I- und D-Regler erstellt werden kann.

5 Realisierung

5.1 Bestimmung der Vektoren

Nachfolgend werden die für die Software Implementierung notwendigen Gleichungen zur Bestimmung der Gelenkkoordinaten aufgeführt

$$\begin{aligned}
\Psi_i &= \frac{\alpha_b}{2} & \Theta_i &= \frac{\alpha_p}{2} && \text{für i=1} \\
\Psi_i &= -\frac{\alpha_b}{2} & \Theta_i &= -\frac{\alpha_p}{2} && \text{für i=2} \\
\Psi_i &= \frac{4\pi}{3} + \frac{\alpha_b}{2} & \Theta_i &= \frac{4\pi}{3} + \frac{\alpha_p}{2} && \text{für i=3} \\
\Psi_i &= \frac{4\pi}{3} - \frac{\alpha_b}{2} & \Theta_i &= \frac{4\pi}{3} - \frac{\alpha_p}{2} && \text{für i=4} \\
\Psi_i &= \frac{2\pi}{3} + \frac{\alpha_b}{2} & \Theta_i &= \frac{2\pi}{3} + \frac{\alpha_p}{2} && \text{für i=5} \\
\Psi_i &= \frac{2\pi}{3} - \frac{\alpha_b}{2} & \Theta_i &= \frac{2\pi}{3} - \frac{\alpha_p}{2} && \text{für i=6}
\end{aligned} \tag{5.1}$$

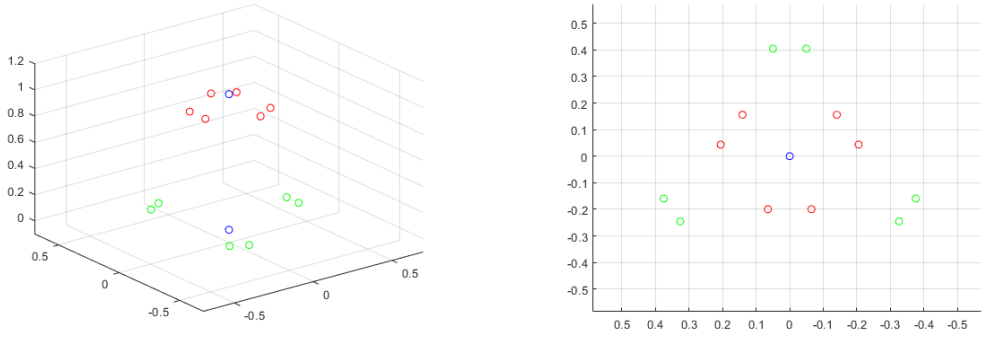
$$B_i = \begin{bmatrix} r_b \cos(\Psi_i) \\ r_b \sin(\Psi_i) \\ r_b \sin(\beta_b) \end{bmatrix} = \begin{bmatrix} b_{ix} \\ b_{iy} \\ b_{iz} \end{bmatrix} \qquad \text{für i=1..6} \tag{5.2}$$

$$P_i = \begin{bmatrix} r_p \cos(\Theta_i) \\ r_p \sin(\Theta_i) \\ r_p \sin(\beta_p) \end{bmatrix} = \begin{bmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{bmatrix} \qquad \text{für i=1..6} \tag{5.3}$$

$$l_i = \sqrt{(p_i - b_i)^2} \qquad \text{für i=1..6} \tag{5.4}$$

5.2 Berechnung in Matlab

Zur Überprüfung der mathematischen Berechnungen der Stewart-Plattform wurde ein Matlab Skript erstellt, bevor die weitere Implementierung auf das ROS-System übertragen wird. Abbildung 5.1 zeigt hierbei die berechnete Lage der oberen (rot) und unteren (unteren) Gelenkkoordinaten sowie die Zentren der Basis- und Mobilen-Plattform (blau).



(a) (b)

Abbildung 5.1: Plot der berechneten Gelenkkoordinaten

Nachdem die Berechnung der richtigen Gelenkkoordinaten abschlossen ist, wird die Gesamtsteuerung auf ein SimMechanics Modell übertragen. Als Grundlage wird ein vorbereitetes Modell verwendet, welches durch Mathworks, dem Hersteller von Matlab, bereitgestellt wird. Eine Übersicht des überarbeiteten Modells ist in Abb. 5.2 dargestellt.

Entsprechend des CAD-Modells der Stewart-Plattform wurden die Koordinatentransformationen und hinterlegen CAD-Modelle angepasst und ersetzt. Um ein realitätsnahes Verhalten des simulierten Modells zu erreichen, wurden die aus dem CAD-Modell ermittelten Bauteilgewichte hinterlegt. Lediglich die Linearantriebe mussten in ihrer Gewichtsverteilung geschätzt werden. Hierfür wurden die Linearantriebe gewogen und eine Gewichtsverteilung von 2/3 für das Gehäuse mit den Elektromotoren und 1/3 für die Kolbenstange angenommen. Der Aufbau der Linearantriebe ist in Abb. 5.3 dargestellt.

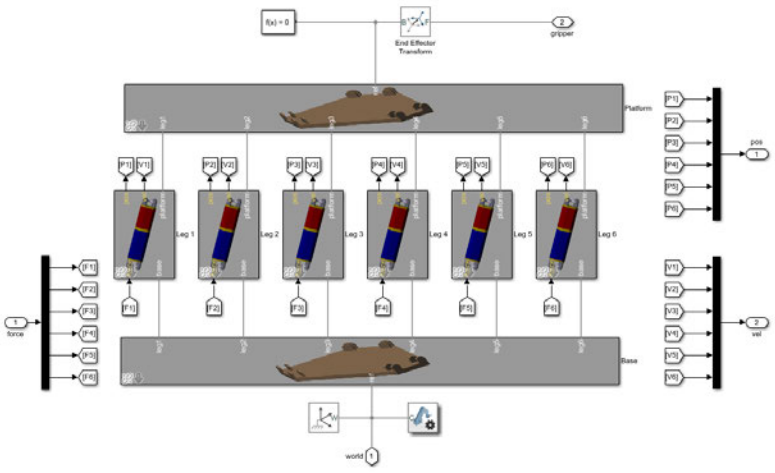


Abbildung 5.2: SimMechanics Übersicht der Verbindung zwischen Beinen und Plattform

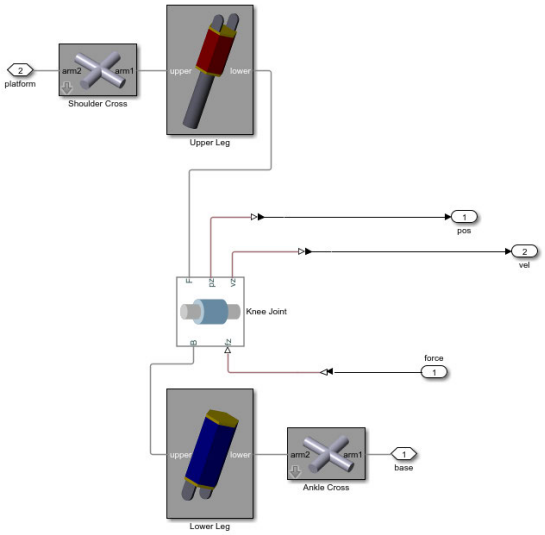


Abbildung 5.3: SimMechanics Modell eines Linearantriebs

Das Ergebnis der SimMechanics Simulation ist in Abb. 5.4 dargestellt. Auf die detaillierte Konstruktion der Gelenkaufnahmen wurde aufgrund der besseren Visualisierung der Verbindungspunkte verzichtet.

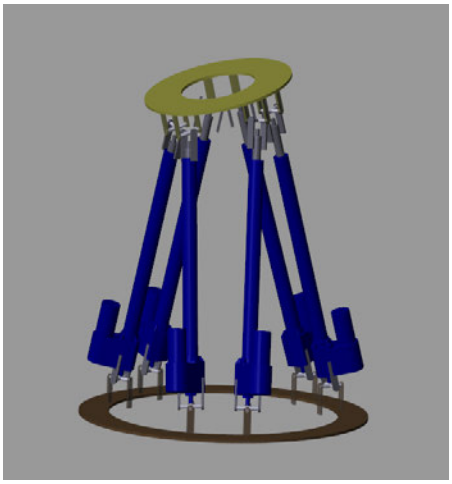


Abbildung 5.4: Simulation der Stewart-Plattform in Matlab SimMechanics

5.2.1 Simulation von reiner Rotationsbewegungen

Die während eines Simulationsdurchlaufs auftretenden Geschwindigkeiten sind in Abb. 5.6 dargestellt. In der Simulation sind nur die Eingangs beschriebenen Massen berücksichtigt und es wurde die maximal zu gewährleistende Winkelgeschwindigkeit von 12 °/s gewählt. Die ermittelten Geschwindigkeiten liegen innerhalb der Spezifikationen, womit hier keine direkte Einschränkung der Stewart-Plattform zu erwarten ist.

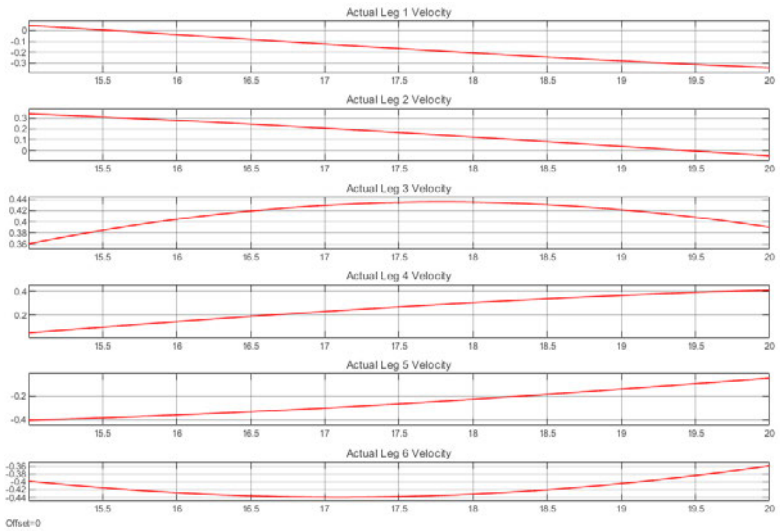


Abbildung 5.5: Verlauf der auftretenden Geschwindigkeiten in den Linearantrieben, Rotation um X- und Y-Achse 12 °/s

In Abb. 5.6 sind die auftretenden Kräfte, welche von den Linearantrieben erzeugt werden müssen, abgebildet. Auch die Kräfte liegen mit 37,2 N innerhalb der Spezifikationen.

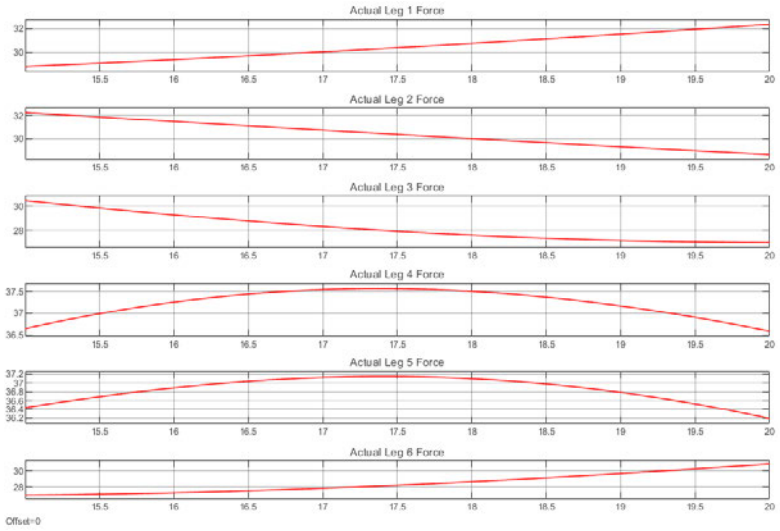


Abbildung 5.6: Verlauf der auftretenden Geschwindigkeiten in den Linearantrieben, Rotation um X- und Y-Achse 12 °/s

5.2.2 Simulation von Rotations- und Translationsbewegung

Im weiteren Verlauf der Simulationen wurde die Bewegungsverlauf der Mobilen-Plattform um eine Bewegung entlang der Z-Achse erweitert. Hier wurde eine Amplitude von 10 cm mit einer Geschwindigkeit von 0,4 m/s gewählt, wie sie als Anforderung zur Kompensierung von Heave-Bewegungen benötigt wird. Die Ergebnisse sind in den Abb. 5.7 und 5.8 dargestellt.

Die auftretenden Kräfte liegen mit 37,4 N nur leicht über den Werten der Simulation ohne translatorische Bewegung entlang der Z-Achse. Anders sieht es hingegen bei den Geschwindigkeiten der Linearantriebe aus. Diese liegen wie in Abb. 5.7 zu erkennen, bei fast 100 mm/s im Maximum. Damit liegt die erforderliche Geschwindigkeit außerhalb der Spezifikationen der verwendeten Linearantriebe.

Basierend auf den Ergebnissen der Simulationen wird bei der Implementierung auf eine Kompensierung der Heave-Bewegung verzichten. Diese Funktion kann aber leicht, basierend auf den Daten der IMU, nachgepflegt werden.

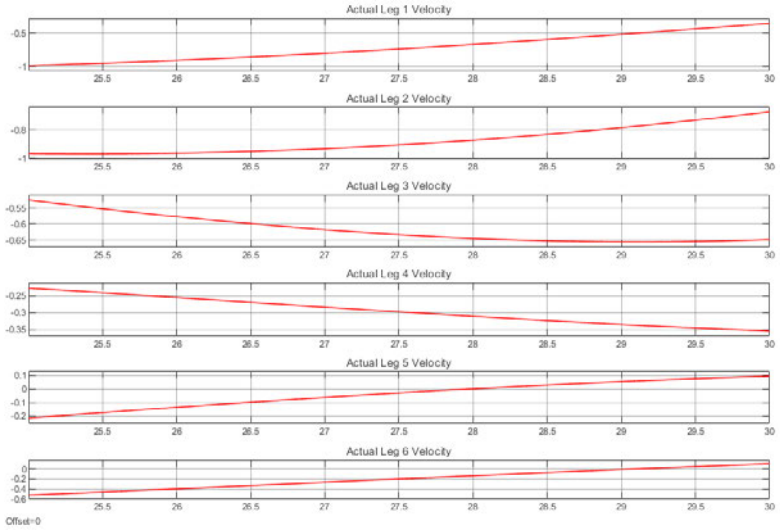


Abbildung 5.7: Verlauf der auftretenden Geschwindigkeiten in den Linearantrieben, Rotation um X- und Y-Achse 12 °/s mit 0,4 m/s entlang der Z-Achse

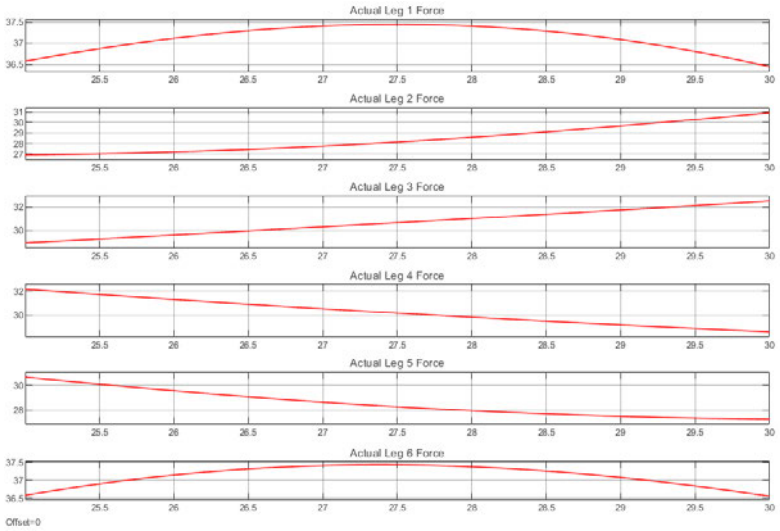


Abbildung 5.8: Verlauf der auftretenden Kräfte in den Linearantrieben, Rotation um X- und Y-Achse 12 °/s mit 0,4 m/s entlang der Z-Achse

5.3 Implementierung in ROS

5.3.1 Einrichten der ROS Umgebung

Im folgenden Abschitt wird die Installation und Einrichtung von ROS Neotic auf dem RockPi 4 mit Ubuntu 20.04 LTS beschrieben. Bei ROS Noetic handelt es sich um die derzeitige LTS Version von ROS mit einem Support bis 2025.

1. Ubuntu 20.04 einrichten:
   In den Einstellung für Repositories müssen restricted, universe und multiverse zugelassen werden. Anschließend kann das ROS Repository mit folgendem Befehl hinzugefügt werden:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(
  lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Schlüssel einrichten:

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-
  key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

   Alternativ kann curl statt apt-key verwendet werden:

```
$ curl -sSL 'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0
  xC1CF6E31E6BADE8868B172B4F42ED6FBAB17C654' | sudo apt-key add -
```

3. Ubuntu Pakete Aktualisieren:

```
$ sudo apt update
```

4. Installation des ROS Noetic Package:

```
$ sudo apt install ros-noetic-desktop-full
```

In dieser Arbeit wurde die Variante desktop-full verwendet, weitere Varianten sind Base und Desktop. Diese stellen einen geringeren Funktionsumfang bereit und benötigen dadurch weniger Speicher.

5. Pakete Suchen:
   Eine Liste mit verfügbaren Paketen kann mit folgendem Befehl abgerufen werden:

```
$ apt search ros-noetic
```

6. Installation von Paketen:
   Zusätliche Pakete können mit dem apt Befehl wie folgt installiert werden:

```
$ sudo apt install ros-noetic-PACKAGE
```

7. Umgebungsvariablen einrichten:
   Um die ROS Befehle dem System bekannt zu machen, muss der Bash Konsole der Pfad zu setup.bash bekannt sein. Dies wird mit folgendem Befehl erreicht:

```
$ source /opt/ros/noetic/setup.bash
```

   Die Einrichtung der Umgebungsvariablen ist immer nur für die aktive Sitzung gültig. Um den vorherigen Befehl nicht bei jedem öffnen der Konsole eingeben zu müssen, kann dieser in die Konsolen Konfiguration eingetragen werden:

```
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

5.3.2 Erstellen eines eigenen ROS-Package

Erstellen eines Workspace

Ist noch kein eigener catkin-Workspace vorhanden, kann dieser mit den aufgeführten Befehlen im Home-Verzeichnis des angemeldeten Users erzeugt werden:

```
$ source /opt/ros/kinetic/setup.bash
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Der erste Befehl macht die ROS-Umgebung im Ubuntu-System bekannt und sorgt dafür, das Befehle der ROS-API aus der Konsole heraus aufgerufen werden können und die installierten Pakete verfügbar sind. Der Befehl mkdir erzeugt unter Ubuntu ein neues Verzeichnis mit dem nachstehenden Namen.

Der Befehl cd öffnet den nachstehenden Dateipfad. Mit dem Befehl catkin_make wird der Workspace mit allen Abhängigkeiten angelegt und kompiliert. Der letzte Befehl macht die im eigenen Workspace erstellten Programme dem ROS- und Ubuntu-System bekannt.

Wird in der vorhandenen ROS-Umgebung auch in der Programmiersprache Python programmiert, ist der Befehl catkin_make wie folgt zu erweitern:

```
$ catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3
```

Nach einmaliger Ausführung kann im Anschluss mit dem regulären catkin_make-Befehl das Kompilieren der Pakete fortgesetzt werden.

Mit dem folgenden Befehl kann geprüft werden, ob der catkin-Workspace richtig angelegt wurde:

```
$ echo $ROS_PACKAGE_PATH
/home/USERNAME/catkin_ws/src:/opt/ros/ROSVERSION/share
```

In der nachfolgenden Antwort, auf den Aufruf $ROS_PACKAGE_PATH, sollten die Platzhalter USERNAME und ROSVERSION durch den angemeldeten Benutzernamen und die installierte ROS-Version ersetzt sein.

Erstellen eines Package

Nachdem ein eigener catkin-Workspace erstellt wurde, kann im Anschluss direkt das eigene ROS-Package erstellt werden. Ein selbst erzeugtes ROS-Package, das von dem Tool catkin erstellt und in das ROS-System integriert wird, muss in der einfachsten Form nur die folgende Struktur aufweisen:

- Paket-Verzeichnis
  - CMakeLists.txt
  - package.xml

Erzeugt wird ein ROS-Package mit den folgenden Befehlen:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg PACKAGE_NAME [depend1] [depend2] [depend3]
$ cd ~/catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.bash
```

Für den Eintrag PACKAGE_NAME ist der gewünschte Name des zu erstellenden Packages einzutragen unter Angabe der benötigten Abhängigkeiten ([depend1]). Hinter den Abhängigkeiten können Bibliotheken zur Programmierung, wie roscpp und rospy stehen, aber auch die im Programm verwendeten Nachrichtentypen, wie zum Beispiel sensor_msgs. Mit Ausführung des Befehls wird das Grundverzeichnis mit dem Package Namen erzeugt und die Dateien package.xml und CMakeList.txt erstellt.

Die package.xml Datei enthält alle Informationen zu den Abhängigkeiten des erzeugten Packages. Diese Informationen sind besonders wichtig, wenn das Package auf ein anderes System übertragen wird, da abhängig von diesen Informationen eventuell fehlende Packages oder Bibliotheken nachinstalliert werden. Ein Kompilieren kann auf dem eigenen System trotz fehlender Einträge möglich sein, sofern die benötigten Packages und Bibliotheken vorhanden sind.

In der CMakeFile.txt Datei sind Informationen für das Kompilieren des Packages hinterlegt. Dies kann zum Beispiel die benötigte Compiler-Version, der Package-Name oder der enthaltene Nachrichtentypen sein.

Das Package-Verzeichnis wird nach Ausführung des Befehls wieder verlassen und mit catkin_make der Workspace kompiliert.

Ein typisches ROS-Package enthält darüber hinaus, in der Regel, folgende Verzeichnisse:

config
Enthält für das Package notwendige Konfigurationsdateien.

include
Enthält die benötigten Header-Dateien.

launch
Enthält die zum Start der Nodes des Packages benötigten Launch-Dateien.

msgs
Enthält die vom Package erzeugten Nachrichtentypen.

scripts
Enthält Skript-Dateien die in den Programmiersprachen Python oder Octave geschrieben sind.

src
Enthält den eigentlichen Programmquellcode in den Programmiersprachen C/C++.

srv
Enthält die von dem Package zur Verfügung gestellten Services.

Base Node und Platform Node

Die Berechnung der Gelenk Positionen in ROS wird exemplarisch für den ersten Linearantrieb näher erläutert und ist für die weiteren fünf Antriebe identisch. Die ROS Base Node hat die Aufgabe die Positionen des Zentrums der unteren Plattform mit den an ihr angebrachten Kreuzgelenken zu berechnen. Der Ursprung der unteren Plattform wird hierfür auf die Koordinaten $[0\ 0\ 0]^T$ gesetzt.

Die Berechnung der Gelenkkoordinaten erfolgt mit den in Gleichung 5.1 und 5.2 aufgestellten zusammenhängen. Die Berechnung ist im folgenden für das Gelenk B_1 aufgeführt:

```
tf_B1.header.frame_id = "base_origin";
tf_B1.child_frame_id = "base_link1";
tf_B1.transform.translation.x = BASE_RADIUS * cos(BASE_ANGLE);
tf_B1.transform.translation.y = BASE_RADIUS * sin(BASE_ANGLE);
tf_B1.transform.translation.z = BASE_HEIGHT;
```

Die Header Frame ID gibt den zugehörigen übergeordneten Parent Frame an. Mit der Child Frame ID wird der zu berechnende Frame bezeichnet. Die korrekte Zuordnung von Parent und Child Frame ist wichtig und dient dem Aufbau von kinematischen Ketten mit zugehöriger Koordinatentransformation.

Entsprechend der Base Node werden in der Platform Node die Transformationen zur Bestimmung der oberen sphärischen Gelenkepositionen an der mobilen Plattform durchgeführt. Diese Transformationen basieren auf den Gleichungen 5.1 und 5.3. Der erstellte Quellcode für diese ROS Node entspricht dem oben beschriebenen Quellcode zur Transformation der unteren Gelenkpositionen.

IMU Node

Die IMU Node dient dem einlesen der Orientierungsdaten, welche von der Bosch BNO055 aufgenommen werden.

Bei Programmstart der IMU Node wird einmalig die Koordinatentransformation zwischen dem World Frame und den beiden Frames Base Footprint und Plattform Origin durchgeführt. Dieses Vorgehen soll dafür sorgen, dass die eingeschaltete und kalibrierte IMU sich in einer beliebigen Orientierung befinden kann. Speziell in der Testphase muss auf diese Weise nicht sichergestellt werden, das sich die Stewart-Plattform und die IMU immer in einer definierten Ausgangslage befinden.

Im regulären Programmablauf werden im Anschluß die Daten der IMU eingelesen und für die Koordinatentransformationen zwischen dem Header Frame imu_bno055 und dem child Frame base_origin verwendet. Auf diese Weise wird des Pivotpunkt der oberen Plattform in das Zentrum der unteren Plattform gelegt. Zusätzlich werden die Winkelorientierungen um die einzelnen Achsrichtungen auf Überschreitung der definierten Grenzen geprüft und begrenzt.

Joint Node

Die ROS Joint Node berechnet die Beinlängen der Stewart-Plattform und stellt die Daten für die Motor Node auf den Microcontrollern bereit.

Darüber hinaus veröffentlicht die Joint Node die aktuelle Parametrierung der Positions- und Geschwindigkeitsregelung. Die Parameter können während der Laufzeit der Joint Node über den folgenden Befehl angepasst werden:

```
rostopic pub /M1/P_Controller_Setting control_msgs/JointControllerState '{p:
    X, i: X, d: X, i_clamp: X, antiwindup: X}'
```

Damit die Nachrichten der Joint Node der Motor Node zur Verfügung stehen, ist es nötig die Nachricht wiederholt zu veröffentlichen. Die Joint Node veröffentlicht dafür ihre Nachrichten mit einer Frequenz von 10 Hz. Das Senden der Nachrichten wird von der Joint Node auch ausgeführt, wenn keine Änderungen der Nachrichten stattfanden.

5.4 Implementierung auf den Microcontrollern

Die verwendeten Mikrocontroller werden über die vorhandene Wlan Schnittstelle mit dem ROS-Master verbunden. Für die Datenübertragung wird das ROS-Serial Package verwendet und als TCP-Server auf dem ROS-Master gestartet. Für jede verbundene Node ist es nötig eine eigene Instanz von ROS-Serial zu starten. Auf den Mikrocontrollern wird die in Abb. 5.9 dargestellte Statemachine implementiert.

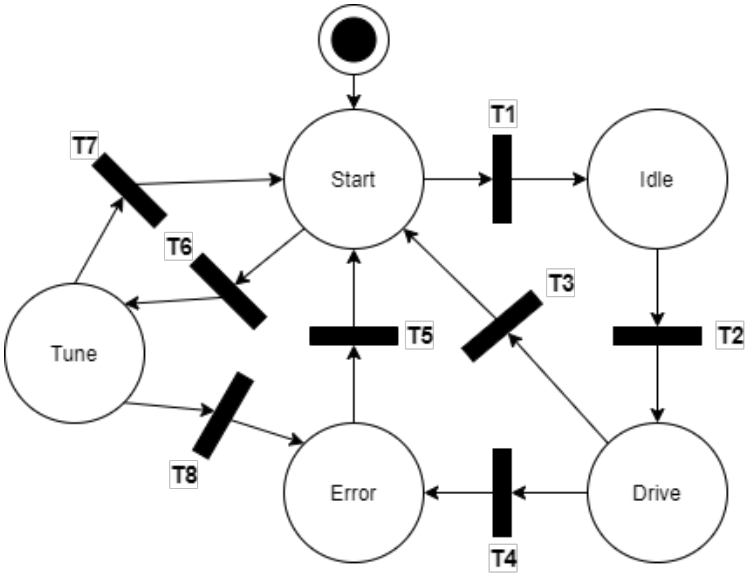


Abbildung 5.9: StateMachine der Motorcontroller

Der Init-State dient dazu die Position der Linearantriebe zu initialisieren da diese über keine Sensoren verfügen welche eine absolute Position nach dem Einschalten der Linearantriebe liefern. Eine kurze Beschreibung der einzelnen Zustände ist nachfolgend aufgeführt:

START Linearantriebe werden mit langsam bis zum Erreichen der Null Position eingezogen und die internen Positionszähler auf Null gesetzt.

IDLE Die Linearantriebe werden in die vorgegebene Parkposition gefahren und gehalten ohne weitere Bewegungen auszuführen.

DRIVE Die Linearantriebe werden aktiv zur Kompensierung von Wellenbewegungen angesteuert.

ERROR Wird ein Fehler in der Steuerung oder Berechnung erkannt, wird der Error-State eingenommen und die Antriebe abgeschaltet.

TUNE Der TUNE-State dient der Regelungsparametrierung. Die Linearantriebe werden mit einem zwischen 10-90 % einstellbaren Sprung bis zu Erreichen ihrer Höchstgeschwindigkeit beaufschlagt.

5.4.1 Ansteuerung der Antriebe

Die Ansteuerung der Linearantriebe erfolgt durch die Motortreiber Escon 50/5. Der Escon 50/5 Motortreiber bietet die Möglichkeit als Strom- oder Drehzahlregler verwendet zu werden. Für die weitere Verwendung wird der Motortreiber als Drehzahlregler konfiguriert und eine Parametrierung der Regelparameter mit dem Escon Autotune Tool für jeden einzelnen Antrieb durchgeführt. Die Vorgabe der der Soll-Geschwindigkeit erfolgt durch ein PWM-Signal mit einem Duty-Cycle zwischen 10% - 90%. Ein Duty-Cycle von 10% steht dabei für Stillstand und ein Duty-Cycle von 90% für die maximale Motordrehzahl. Erzeugt wird das PWM-Signal von dem am Motortreiber angeschlossenen ESP32 Mikrocontroller. Der darauf aufbauende Regelkreises ist in Abb. 5.10 dargestellt.

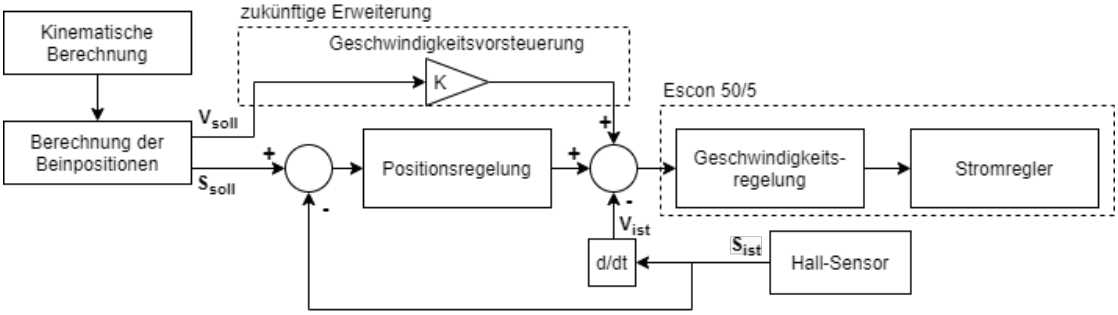


Abbildung 5.10: Regelkreis der Linearantriebe

Die Positionsregelung ist als P-Regler im ESP32 implementiert, gefolgt von einer PI-Regler im Escon 50/5 zur Geschwindigkeitsregelung. Mit Hilfe eines Hall-Sensors im Linearantrieb wird die Ist-Position bestimmt und für die Positionsregelung verwendet. Weiterhin kann über die Positionsänderung die Ist-Geschwindigkeit des Antriebs bestimmt werden. Diese kann zusätzlich vom Escon Motorcontroller direkt ausgegeben werden. Die Reglerausgangsgröße des Positionsreglers wird in ein PWM Signal übersetzt und an den Motorcontroller zur Geschwindigkeitsregelung weitergeleitet.

5.4.2 Implementierung der Regelung

```
void PID_Pos (float process_value, float set_point, PID_setup* PID) {
  PID->e = set_point-process_value;
  PID->iterm += (PID->Ki*PID->Ta*PID->e);
  if (PID->iterm > PID->i_clamp && PID->antiwindup) {
    PID->iterm = PID->i_clamp;
  } else if (PID->iterm < -PID->i_clamp && PID->antiwindup) {
    PID->iterm = -PID->i_clamp;
  }
  PID->y= (PID->Kp*PID->e)+PID->iterm+(PID->Kd*((PID->e-PID->ealt))/PID->Ta);
  PID->ealt = PID->e;
  if (PID->y > PID->y_max) {
    PID->y = PID->y_max;
  } else if (PID->y < -PID->y_max) {
    PID->y = -PID->y_max;
  }
}
```

Die Geschwindigkeit des Regel soll nach dem Nyquist-Shannon-Abtasttheorem bestimmt werden. Der Linearantrieb hat eine maximale Geschwindigkeit von 72,1 mm/s, laut Datenblatt. Der integrierte Hall-Sensor liefert eine Auflösung von 1,27 mm/Puls. Daraus ergibt sich für das Signal des Hall-Sensors

$$f_{max} = \frac{72,1mm/s}{1,27mm/Puls} = 56,77Hz \tag{5.5}$$

als maximale Frequenz.

Das Nyquist-Shannon-Abtasttheorem gibt Auskunft darüber wie oft eine zeit-kontinuierliche Sinusschwingung abgetastet werden muss, um sie aus dem digitalisierten Signal wieder herstellen zu können. [13, S.19]

$$f_{ab} > 2f \tag{5.6}$$

Eine Faustregel zur Bestimmung der Abtastfrequenz besagt, das die Abtastfrequenz um den Faktor 5-10 über der im System höchsten auftretenden Frequenz liegt. [7, S.190]

Die Regelungsfrequenz sollte demnach im Bereich von 5-10 Hz bei Anwendung der Faustformel liegen und bei maximal rund 25,5 Hz bei Zugrundelegung des Abtasttheorem.

Regelungsparametrierung

Die Parameter für die implementierte Regelung wurden nach einer Methode von Ziegler und Nichols bestimmt. Von Ziegler und Nichols wurden dafür zwei unterschiedliche Verfahren entwickelt. Beim ersten Verfahren wird der K_p Faktor solange erhöht, bis die Regelstrecke anfängt in ein dauer schwingen überzugehen. Das ermittelte K_p wird an diesem Punkt als kritische Reglerverstärkung bezeichnet. Für die weitere Parameterbestimmung muss weiterhin noch die Periodendauer der Dauerschwingung bestimmt werden.

Dieses Verfahren wird für die vorliegende Regelstrecke mit ihren Linearantrieben als nicht praktikabel angenommen. Das Risiko einer Beschädigung durch befahren der minimalen oder maximalen Längengrenzen der Linearantriebe kann nicht sicher verhindert werden. Von Ziegler und Nichols wurde aber noch ein weiteres Verfahren für Regelungsparametrierung entwickelt. Bei diesem zweiten Verfahren wird das Verhalten der Regelstrecke aufgezeichnet und bietet die Möglichkeit einer graphischen Auswertung. Das Verfahren hierfür ist im State Tune des Mikrocontroller implementiert. Im State Tune zeichnet der Mikrocontroller die Sprungantwort der Linearantriebe bei aufschalten eines Signalsprungs

auf. Abbildung 5.11 zeigt den idealen Verlauf der Sprungantwort einer Regelstrecke. Zur

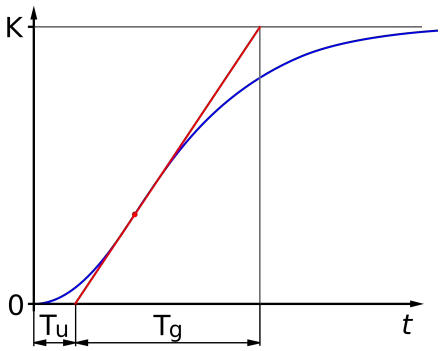


Abbildung 5.11: Sprungantwort einer Regelstrecke [**?**]

Bestimmung der Regelungsparameter wird die Wendetangente (rot) eingezeichnet. Mit Hilfe der Wendetangente kann auf der X-Achse die Verzugszeit T_u und die Ausgleichszeit T_g abgelesen werden.

Der Mikrocontroller misst den zeitlichen Verlauf und bestimmt den Wendepunkt. Die Ausgabe der Parameter erfolgt über die serielle Schnittstellen.

5.4.3 Erzeugung eines PWM-Signales

Die Erzeugung des PWM Signals für die Ansteuerung der Linearantriebe wird mit Hilfe des Motor Control PWM Moduls, kurz MCPWM, des ESP32 durchgeführt. Insgesamt stellt der ESP32 drei Timer (0,1 und 2) für diese Funktion zur Verfügung. Alle drei Timer besitzen erweiterte Eingabefunktionen wie einen Eingang zur Synchronisation, einen Eingang zur Erkennung von Fehlern durch Überspannung und -strom sowie einen Eingang zur Aufnahme eines Feedback Signals wie die Rotor Position.

```
mcpwm_gpio_initialize();
mcpwm_config_t pwm_config;
pwm_config.frequency = 5000;
pwm_config.cmpr_a = 0;
pwm_config.cmpr_b = 0;
pwm_config.counter_mode = MCPWM_UP_COUNTER;
pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
```

```
static void m1_forward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num ,
    float duty_cycle)
{
digitalWrite(M1_DIR, LOW);
mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_A, BIAS + duty_cycle);
mcpwm_set_duty_type(mcpwm_num, timer_num, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);
}
```

Die Funktion m1_backward ist vom Aufbau identisch zur Funktion m1_forward, lediglich das Richtungssignal wird vom Low- in den High-State gewechselt, um dem Escon Motortreiber den geänderten Richtungsbefehl mitzuteilen.

```
static void m1_stop(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num)
{
mcpwm_set_signal_low(mcpwm_num, timer_num, MCPWM_OPR_A);
}
```

5.4.4 Positionsbestimmung der Linearantriebe

Die Positionsbestimmung der Linearantriebe erfolgt mithilfe der integrierten Hall-Sensoren. Der Hall-Sensor des Motors M1 ist dazu an den Pin 21 des zugehörigen ESP32 angeschlossen. Um die korrekte Erfassung neben dem Ablauf des weiteren Programmcodes zu gewährleisten wird ein Interrupt-Handler zur Erfassung des Hall-Sensor Signale implementiert.

Wird an Pin 21 des ESP32 ein Wechsel von einem Low-Pegel zu einem High-Pegel erkannt (Steigende Flanke), wird der Interrupt-Handler ausgeführt. Zu Beginn des Interrupt-Handler wird geprüft, ob die Zeitschwelle zum Entprellen des Eingangssignals überschritten ist. Diese Zeitschelle sorgt dafür, dass eine steigende Flanke nicht fälschlicherweise mehrfach gezählt wird. Anschließend wird ein Zähler für den Interrupt hochgezählt und entsprechend der vorgegebenen Bewegungsrichtung die Geschwindigkeit und Beschleunigung des Linearantriebes berechnet.

Im Hauptprogramm des ESP32 wird geprüft, ob der Interrupt Zähler größer als Null ist. Ist diese Bedingung erfüllt, wird die aktuelle Position des Linearantriebes entsprechend der aktuellen Bewegungsrichtung erhöht oder gesenkt.

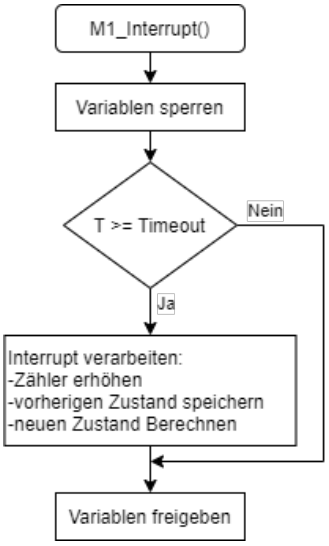


Abbildung 5.12: Programmablaufplan der Funtion M1_Interrupt

5.4.5 Kommunikation mit dem ROS-Master

Die Kommunikation mit dem zwischen dem ROS Master und dem ROS Slave erfolgt über die vorhanden Wlan-Module. Auf diese Weise ist es möglich die Stewart-Plattform räumlich vom ROS Master System zu trennen, da keine direkte Kabelverbindung vorhanden ist.

Für die Kommunikation soll aus den ROS eigenen Control-Msgs der JointControllerState Nachrichtentyp verwendet werden. Der JointControllerState bietet dabei folgenden Nachrichteninhalt:

Der ROS header dieses Nachrichtentyps enthält eine aufsteigende Sequenz ID, einen aktuellen Zeitstempel und eine Frame-ID. Alle weiteren Variablen der Nachricht entsprechen den Zustandsvariablen eines PID-Regelkreises.

5.4.6 Einlesen von IMU-Daten

Zur Ansteuerung der Stewart-Plattform wird ein weiterer ESP32 Mikrocontroller in Kombinationen mit einer Bosch BNO055 IMU verwendet. Diese IMU soll genutzt werden, um die Bewegung der Pontonplattform bzw. der Basis der Stewart-Plattform zu ermitteln und über die mobile Plattform der Stewart-Plattform eine Gegenbewegung auszuführen.

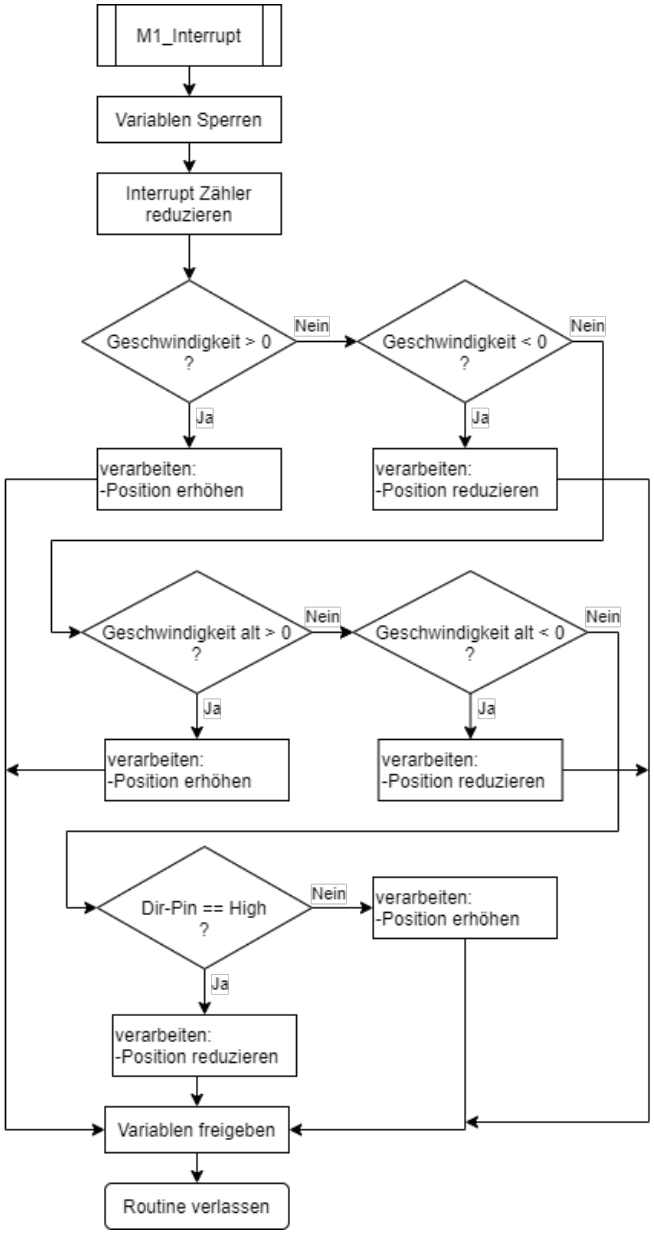


Abbildung 5.13: Programmablaufplan der Interrupt Verarbeitung innerhalb der Main-Funktion

Die in Kapitel 4.2.5 bereits eingehend beschriebene Bosch IMU wird über eine I^2C Verbindung von dem ESP32 Mikrocontroller ausgelesen. Um sicherzustellen das die gemessenen Daten der IMU präzise sind, ist es erforderlich die IMU nach jedem Trennen der Spannungsversorgung neu zu kalibrieren.

Bezeichnung	Typ
header	Header
set_point	Float64
process_value	Float64
process_value_dot	Float64
error	Float64
time_stamp	Float64
command	Float64
p	Float64
i	Float64
d	Float64
i_clamp	Float64

Tabelle 5.1: Dateninhalt des Nachrichtentyps JointControllerState

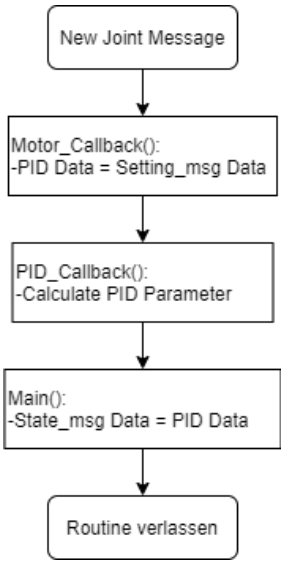


Abbildung 5.14: Programmablaufplan bei Empfang und Sendung neuer Messages

Ablauf der Kalibrierung:

1. Gyroskop Kalibrierung:
   Die IMU muss einige Sekunden ruhig auf einem stabilen Untergrund liegen um die Offsets des Gyroskop zu bestimmen

2. Beschleunigungssensor Kalibrierung:
   Die IMU muss für einige Sekunden ruhig auf jede ihrer sechs Seiten gelegt werden um die Offsets des Beschleunigungssensors zu bestimmen

3. Magnetometer Kalibrierung:
   Mit der IMU müssen in der Luft mehrere acht Figuren durchfahren werden, um die Offsets des Magnetometer und die magnetische Nord Richtung zu bestimmen

Nachdem die IMU erfolgreich kalibriert ist, bleiben die Parameter bis zu deren Abschalten gespeichert. Da die verwendete IMU jedoch über keinen EEPROM zur dauerhaften Speicherung verfügt, ist es nötig nach jedem Neustart eine Kalibrierung durchzuführen. Da die Durchführung der Kalibrierung recht aufwendig ist und bei vollständig verbauter Sensorik zum Teil auch nicht mehr ohne Weiteres möglich ist, wurden die Mittelwerte aus 10 unterschiedlichen Kalibrierungen bestimmt. Die Ergebnisse sind in Tabelle 5.2 aufgeführt.

Weiterhin ist es wichtig die IMU vor dem Start der eigentlichen Steuerung der Stewart-Plattform durchzuführen. Andernfalls kann es zu Unstetigkeiten in den Sensordaten kommen, wenn zum Beispiel der magnetische Nordpol gefunden wird.

Sensorachse	Beschleunigungssensor	Gyroskop	Magnetometer
X	-22	-2	-1820
Y	-13	-2	230
Z	-29	1	70

Tabelle 5.2: Aus 10 Versuchsreihen bestimmte Kalibrierwerte der BNO055 IMU

Neben den Kalibrierwerten für die einzelnen Achsrichtungen werden auch Kalibrierwerte für die Radien des Beschleunigungssensors und Magnetometer ausgegeben. Der Radius des Beschleunigungssensor beträgt 1000 und der Radius des Magnetometer beträgt 816. Die ermittelten Kalibrierwerte werden bei Programmstart der IMU fest in die Kalibrierungsregister geschrieben. Damit ist es möglich den fehlenden EEPROM-Speicher zur dauerhaften Speicherung der Kalibrierwerte zu umgehen.

Die Orientierungsdaten der IMU werden direkt als Quaternionen ausgelesen. Gemeinsam mit den Daten des Beschleunigungssensor werden diese Daten in dem ROS Nachrichtenformat Sensor_msgs/IMU an den ROS-Master gesendet.

Ein Umrechnen der Quaternionen in Eulerwinkel ist nach folgender Gleichung möglich:

$$\begin{bmatrix} \psi \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \arctan(\frac{2(q_0q_1+q_2q_3)}{1-2(q_1^2+q_2^2)}) \\ \arcsin(2(q_0q_2-q_3q_1)) \\ \arctan(\frac{2(q_0q_3+q_1q_2)}{1-2(q_2^2+q_3^2)}) \end{bmatrix} \tag{5.7}$$

6 Funktionstest

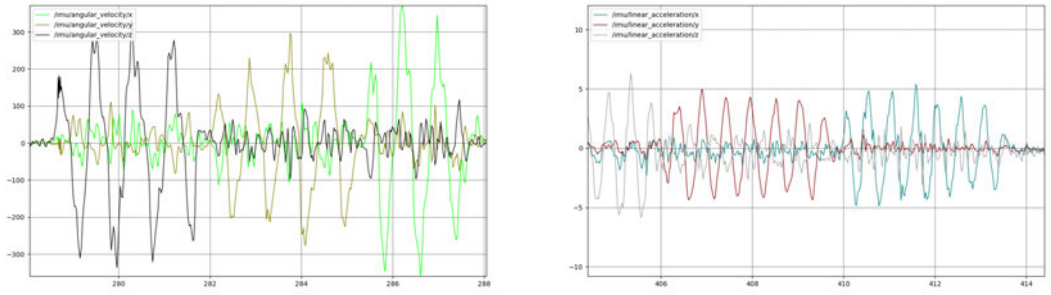
Der Prototyp der Stewart-Plattform wurde um die Leistungsendstufen und die Mikrocontroller erweitert und ist in Abbildung 6.1 abgebildet. Für die zentrale Steuerung ist ein ROCK Pi 4 vorhanden. Die Implementierung wurde basierend auf dem entwickelten Konzept durchgeführt. Alle nachfolgenden Untersuchungen wurden an diesem Prototyp getätigt.

Abbildung 6.1: Prototyp der Stewart-Plattform in seiner Grundstellung

6.1 Bestimmung der Orientierung

Die Orientierungs- und Bewegungsdaten der verwendeten IMU werden mit der maximalen Frequenz des BNO055 ausgelesen. Diese Frequenz liegt bei 100 Hz. Mit der gleichen

Frequenz werden die Daten von dem Mikrocontroller an den ROS-Master gesendet und verarbeitet. Die empfangenen Messdaten des Beschleunigungs- und Drehratensensors sind in Abb. 6.2 dargestellt.



(a) (b)

Abbildung 6.2: Empfangene Messdaten im Topic /imu,
(a) Drehratensensor °/s, (b) Beschleunigungssensor in m/s^2

Das einlesen von Bewegungsdaten ist damit erfolgreich und funktioniert wie geplant.

6.2 Bestimmung der Gelenkkoordinaten

Die Orientierungsdaten der IMU werden vom ROS-Master über ROS-Serial Server empfangen. Wie in Abbildung 6.3 dargestellt, werden die Daten in dem Topic /imu veröffentlicht und stehen im System bereit. Die Koordinatentransformation wird von der Node /imu_tf2_broadcaster definiert und in dem Topic /tf systemweit veröffentlicht.

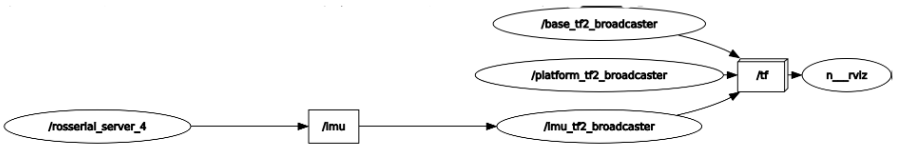


Abbildung 6.3: Datenverlauf zur Berechnung der Gelenktransformationen

ROS stellt mit dem Package Rviz ein Tool zur Visualisierung in Echtzeit bereit. Das Berechnen der Koordinatentransformationen erfolgt von der Joint-Node und ist in Abb. 6.4 mithilfe von Rviz dargestellt. Dabei werden die Transformationsdaten der Broadcaster-Nodes verwendet um die entsprechende Koordinatentransformation zu berechnen.

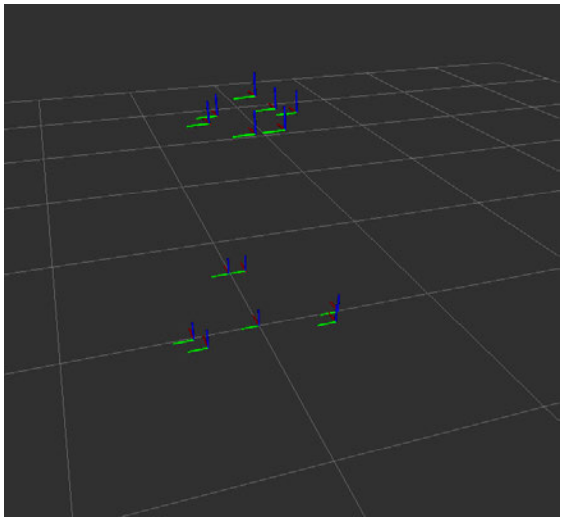


Abbildung 6.4: Visualisierung der Gelenkkoordinaten in ROS-Rviz

Die Berechnung der zugehörigen Beinlängen konnte ebenfalls erfolgreich durchgeführt werden.

6.3 Ansteuerung der Antriebe

Zur Beurteilung der Positionsregelung eines Linearantriebs ist in Abb. 6.5 die Fahrt von Position null auf Position 50 dargestellt. Die Daten wurden von dem Mikrocontroller zur Steuerung der Linearantriebe M1 und M2 an den ROS-Master gesendet. Von der zugehörigen ROS-Serial Node werden diese Daten anschließend in das Topic /P_Controller_State des jeweiligen Antriebes geschrieben.

Die Auflösung der X-Achse ist betragt 200ms und es lässt sich eine Verzögerung von 100ms zwischen der Zielgröße und der Stellgröße liegt. Diese Verzögerung kann mit der Regelungsfrequenz von 10 Hz erklärt werden. Bis zum vollständigen anfahren der Zielposition vergeht ca. 1 Sekunde.

Die Fahrgeschwindigkeit des Beobachteten Linearantriebs ist in Abb. 6.6 abgebildet. Es ist zeigt sich, das die Linearantriebe entgegen dem Datenblatt eine höhere Geschwindigkeit als 72,1 mm/s erreichen. Es konnte reproduzierbar eine Geschwindigkeit von 112 mm/s erreicht werden.

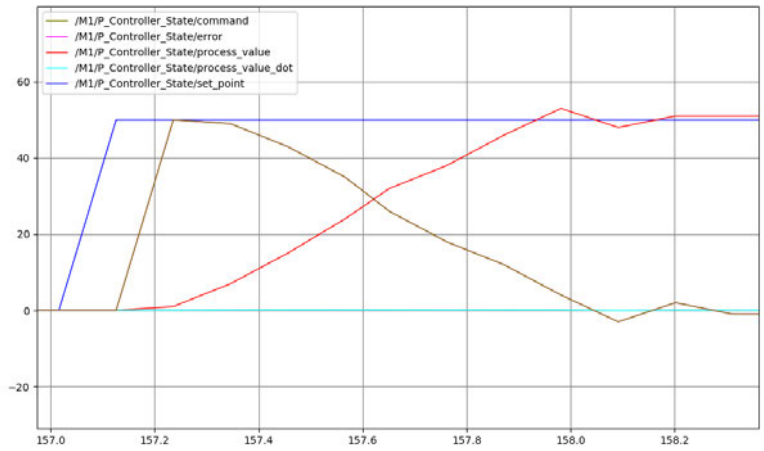


Abbildung 6.5: Visualisierung der Fahrt eines Antriebes von Position 0 auf Position 50

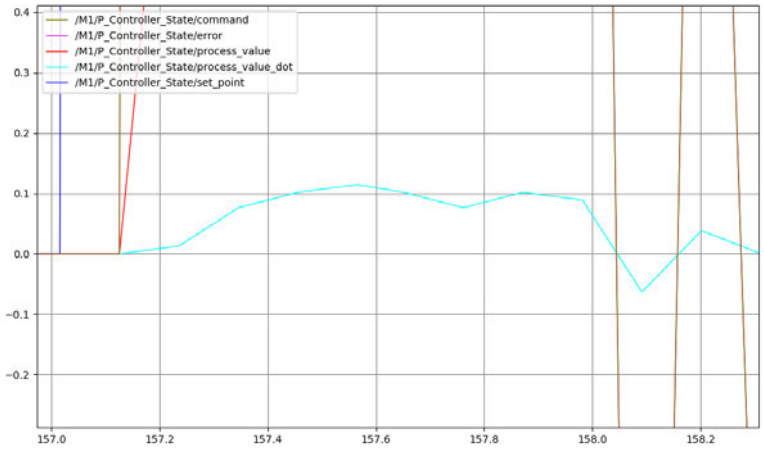


Abbildung 6.6: Visualisierung der Geschwindigkeit eines Antriebes während der Fahrt von Position 0 auf Position 50

Alle weiteren Daten während des Betriebs der Stewart-Plattform sind der Anlage beigefügt. Dazu zählt eine Übersicht des TF-Tree und des Node-Tree sowie der vollständige Quellcode der Implementierung.

Im Gesamtsystem haben sich in den Versuchen noch weitere Verzögerungen neben den Reglern gezeigt. Diese werden aktuell auf ca. 500 ms geschätzt. Diese sind vermutlich auf Latenzen durch die WLAN-Schnittstelle und Verzögerungen innerhalb des Betriebssystems zurückzuführen.

7 Diskussion der Ergebnisse

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der Prototyp einer Stewart-Plattform zur Wellenkompensation untersucht. Dabei wurden Anforderungen an die Geometrien des benötigten Arbeitsraums und die Geschwindigkeiten der Plattform definiert. Mithilfe der mathematischen Zusammenhänge zwischen der Mobilen-Plattform und den sechs Linearantrieben konnte das Arbeitsraumvolumen durch ein numerisches Verfahren bestimmt und graphisch bewertet werden.

Die Verwendung einer Simulation hat darüber hinaus gezeigt, dass die verwendeten Linearantriebe nur in der Lage sind Rotationsbewegungen der Mobilen-Plattform umzusetzen. Wird diese Rotationsbewegung um die angesetzte translatorische Bewegung für eine Heave-Kompensierung erweitert, ist die Geschwindigkeit nicht mehr ausreichend. Die Erzeugten Kräfte der Linearantriebe konnten mithilfe der Simulation jedoch mit ausreichend bewertet werden.

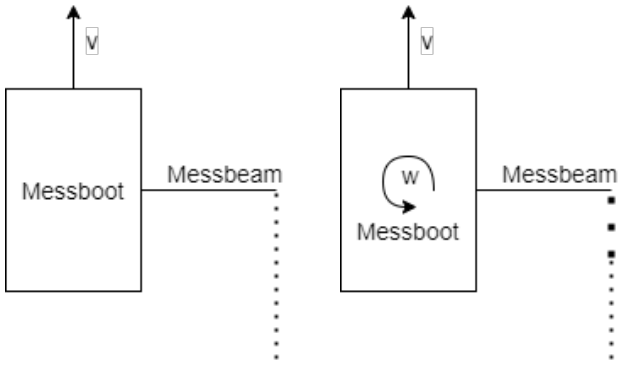
Im weiteren Verlauf wurde das Konzept für eine Steuer- und Regelungseinrichtung entworfen und entsprechende Algorithmen implementiert. Es konnte gezeigt werden, dass das Grundsystem korrekt arbeitet und die Orientierungsdaten in Bewegungen der Mobilen-Plattform übersetzt. Es treten aber schnittstellen- und betriebssystembedingte Verzögerungen auf. In Kombination mit der geringen Positionsauflösung der verwendeten Linearantriebe und daraus resultierenden trägen Regelung, konnte speziell die Anforderung an die Echtzeitfähigkeit nicht erfüllt werden.

Die Anforderung an die Stewart-Plattform können damit wie folgt zusammenfassend Bewertet werden:

Bewegung:	Wert:
Arbeitsraumgeometrie	+
Geschwindigkeit der Linearantriebe	0
Kraft der Linearantriebe	+
Regelung in Echtzeit	-

7.2 Weitere Einflüsse auf das Messsystem

Eine Messung mit dem Multibeam Seabat T50-P wird nach Möglichkeit mit einer reinen geradeaus Fahrt, wie in Abbildung 7.1(a) dargestellt, durchgeführt. Dies ist jedoch bedingt durch äußere Einflüsse nicht immer möglich. Ursachen hierfür sind Wellengang verursacht durch anderen Schiffsverkehr oder Querströmungen an Wassereinläufen. Die Folge der Äußeren Einflüsse sind dann, wie in Abbildung 7.1(b) dargestellt, Rotationen um die Hochachse des Messbootes.



(a) (b)

Abbildung 7.1: Prinzip Darstellung einer typischen Messfahrt
(a) reine geradeaus Fahrt
(b) geradeaus Fahrt mit Rotationsbewegung um die Hochachse

Der Fahrabstand zwischen Sensor und Messobjekt beträgt 3,5m in der kürzesten Strecke. Die Fahrgeschwindigkeit v wird nach eigenen Vorgaben auf Maximal 0,5 m/s begrenzt. Bei der Aussendung von 50 Schallsignalen/s bzw. Messfrequenz von 50 Hz, ergibt sich damit ein Abstand zwischen den einzelnen Beams von

$$\Delta s = \frac{0,5\,m/s}{50\,Hz} = 1\,cm. \tag{7.1}$$

Wird durch äußere Einflüsse eine Rotation um die Hochachse mit einer Winkelgeschwindigkeit $\dot{\varphi}$ von 2 °/s verursacht, erhöht sich der Abstand zwischen den einzelnen Beams wie folgt:

$$\Delta s = \frac{0,5\,m/s}{50\,Hz} + 3,5\,m \cdot \frac{2\,^\circ/s}{50\,Hz} \cdot \frac{\pi}{180\,^\circ} = 1,244\,cm \tag{7.2}$$

Ziel innerhalb einer Messfahrt ist es einen Abstand zwischen den einzelnen Messbeamstreifen von 2 cm nicht zu überschreiten. Bei einer maximalen Fahrgeschwindigkeit von derzeit 0,5 m/s ist damit eine Winkelgeschwindigkeit von

$$\dot{\varphi} = \frac{2\,cm - 1\,cm}{3,5\,m} \cdot \frac{180\,^\circ}{\pi} \cdot 50\,Hz = 8,185\,^\circ/s \tag{7.3}$$

zulässig, ohne den maximalen Messabstand zu überschreiten. Mit dieser Rechnung soll gezeigt werden, wie stark der Einfluss von Rotationsbewegungen um eine Rotationsachse ist. Nicht berücksichtigt wurden Längenänderungen entlang der Messachse infolge von Rotation. Kann dieser Effekt zukünftig durch den Einsatz der Stewart-Plattform reduziert werden, ist eine kontinuierliche Fahrt von 1 m/s möglich, ohne den maximalen Abstand von 2 cm zu überschreiten. Die Fahrgeschwindigkeit kann damit verdoppelt und das Risiko von Fehlmessfahrten deutlich reduziert werden.

8 Ausblick

Im Rahmen dieser Arbeit konnte gezeigt werden, dass die Geschwindigkeit der Positionsregelung noch nicht in Echtzeit stattfindet. Dieser Umstand kann durch Anpassungen der Positionsbestimmung innerhalb der Linearantriebe verbessert werden können. Eine Möglichkeit ist hier die Erweiterung durch Sensoren die den anisotropen magnetoresistiven Effekt (AMR) ausnutzen und damit die Ausrichtung der magnetischen Feldes bestimmen.

Neben der reinen Regelungsgeschwindigkeit begrenzen die Fahrgeschwindigkeit der Linearantriebe die Wellenkompensierung. Es wird daher Empfohlen diese durch Leistungsstärkere Modelle zu ersetzen.

Ergänzend sollte auch die Anpassung des Kommunikationsprotokolls weiter Untersucht werden. Hier besteht die Möglichkeit auf Protokolle wie CAN oder EtherCAT auszuweichen um die Echtzeitfähigkeit weiter zu erhöhen.

Der Arbeitsraum besitzt drei Symmetrieachsen. Diese Eigenschaft kann für Berechnungen ausgenutzt werden, wodurch es prinzipiell möglich ist, eine vollständige Abstandsberechnung mit einem sechstel des Rechenaufwandes abzubilden. Weiterhin sollte der Arbeitsraum auf mögliche Singularitäten untersucht werden. Damit kann der effektiv Nutzbare Arbeitsraum bis an seine durch Singularitäten begrenzten Ränder erweitert werden. Es sollte in zukünftigen Arbeiten genauer betrachtet werden, wo Optimierungen der Rechenverfahren möglich und sinnvoll sind.

Literaturverzeichnis

[1] Association, Robotic I.: *UNIMATE.* – URL https://www.robotics.org/joseph-engelberger/unimate.cfm. – Zugriffsdatum: 10.09.2020

[2] Bosch Sensortec: *Product Data Sheet BNO055*

[3] Ciupek, Martin: *Absatzprognosen im Robotergeschäft zwingen Hersteller wie Kuka zum Umdenken.* – URL https://www.vdi-nachrichten.com/technik/absatzprognosen-im-robotergeschaeft-zwingen-hersteller-wie-kuka-zum-umdenken/. – Zugriffsdatum: 10.09.2020

[4] Düsterlohe, Alexander von: *Kinematik, Geometrie und Mathematik des Hexapod-Teleskops*, Ruhr-Universität Bochum, dissertation, 2003. – URL https://hss-opus.ub.ruhr-uni-bochum.de/opus4/frontdoor/index/index/year/2018/docId/259

[5] Elkady, Ayssam ; Sobh, Tarek: *Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography.* – URL https://www.hindawi.com/journals/jr/2012/959013/. – Zugriffsdatum: 10.09.2020

[6] Gough, V. E.: *Contribution to discussion of papers on research in automobile stability, Proceedings of the Automobile Division of the Institution of Mechanical Engineers.* 1956-1957. – 392–394 S

[7] Heinrich, Berthold ; Linke, Petra ; Glöckler, Michael: *Grundlagen Automatisierung.* Springer Vieweg. – ISBN 978-3-658-17582-5

[8] Honeywell: *6 Degrees of Freedom Inertial Measurement.* – URL https://sensing.honeywell.com/honeywell-sensing-inertial-measurement-unit-6df-applicationnote.pdf

[9] Hydromapper GmbH: *Homepage der Hydromapper GmbH.* – URL https://www.hydromapper.de/de/. – Zugriffsdatum: 10.09.2020

[10] Jadran Lenarcic, Manfred L. H.: *Advances in Robot Kinematics: Analysis and Control*. Springer Science+Business Media Dordrech. – ISBN 978-94-015-9064-8

[11] maxon motor ag: *Product Data Sheet ESCON 50/5*

[12] Moteck.com: *Product Data Sheet Actuator ID10*

[13] Parthier, Rainer: *Messtechnik - Vom SI-Einheitensystem über Bewertung von Messergebnissen zu Anwendungen der elektrischen Messtechnik*. Springer Vieweg. – ISBN 978-3-658-27131-2

[14] Physik Instrumente GmbH & Co. KG: *Parallelkinematik*. – URL https://www.physikinstrumente.de/de/technologie/parallelkinematik/. – Zugriffsdatum: 10.09.2020

[15] Releases, IFR P.: *Global industrial robot sales doubled over the past five years*. – URL https://ifr.org/news/global-industrial-robot-sales-doubled-over-the-past-five-years. – Zugriffsdatum: 10.09.2020

[16] ROS.org: *Robot Operating System*. – URL http://wiki.ros.org/de/ROS/Introduction

[17] Röse, Dipl.-Ing. A.: *Parallelkinematische Mechanismen zum intrakorporalen Einsatz in der laparoskopischen Chirurgie*, Universität Darmstadt, dissertation, 2011. – URL https://tuprints.ulb.tu-darmstadt.de/2493/

[18] Stewart, D.: *A platform with six degrees of freedom, Proceedings of the institution of mechanical engineers*. Bd. 180. 1965-1966. – 371–386 S

[19] Wikipedia.org: *Hexapod*. – URL https://de.wikipedia.org/wiki/Hexapod. – Zugriffsdatum: 10.09.2020

[20] Wikipedia.org: *Model Predictive Control*. – URL https://de.wikipedia.org/wiki/Model_Predictive_Control. – Zugriffsdatum: 10.09.2020

[21] Winter, Roy de: *Designing Ships using Constrained Multi-Objective Efficient Global Optimization*, Dissertation, 05 2018

[22] WITTENSTEIN alpha GmbH: *Planetengetriebe für Delta-Roboter*. – URL https://alpha.wittenstein.de/de-de/produkte/servogetriebe/spielarme-planetengetriebe/planetengetriebe-fuer-delta-roboter/. – Zugriffsdatum: 10.09.2020

A Anhang

A.1 Robot Operating System (ROS)

A.1.1 Packages

Die Packages (Pakete) sind die Hauptelemente für das Organisieren der Software in ROS. Sie können verschiedene Laufzeitprozesse (Nodes), Bibliotheken oder Daten- und Konfigurationsdateien enthalten. Informationen über ein Package und seine Abhängigkeiten zu anderen Packages, werden durch eine Manifest-Datei beschrieben. Packages werden in der Regel mit einem ROS spezifischen System (catkin) erzeugt und kompiliert.

A.1.2 Nodes

ROS-Nodes sind ausführbare Dateien in ROS. Ein ROS-basiertes System besteht in der Regel aus einer Vielzahl parallel ausgeführter und untereinander über ROS-Messages kommunizierender Nodes. Eine ROS Node kann verschiedene Sensoren und Aktoren steuern und ist in der Lage Bibliotheken, welche von anderen Packages bereitgestellt werden, zu laden. ROS-Nodes können in den Programmiersprachen C/C++ und Python implementiert werden.

A.1.3 Nodelet

Eine ROS-Nodelet bietet die Möglichkeit mehrere Algorithmen und Bibliotheken, auf einer Maschinen, in einem einzigen Prozess auszuführen. Dabei spart eine Nodelet Ressourcen ein, indem die Algorithmen nicht extern über das ROS-System Nachrichten austauschen, sondern dieser Austausch prozessintern stattfindet. Eine Nodelet kann darüber hinaus den Programmcode dynamisch laden und in separaten Namespaces ausführen, wodurch eine Nodelet nach außen wie mehrere einzelner Nodes auftreten kann.

A.1.4 Topics

Ein Topic ist das Transportsystem, mit dem die ROS-Messages zwischen zwei Nodes ausgetauscht werden. Jedes Topic besitzt einen Namen über den ein bestimmter Node seine Daten senden (publish) kann. Weitere Nodes können auf diese Weise das jeweilige Topic abonnieren (subscribe) und anschließend die Daten auslesen.

A.1.5 Messages

ROS-Messages (Nachrichten) stellen eine Möglichkeit zur Kommunikation für Nodes untereinander dar und werden in Topics organisiert. Eine Node kann Nachrichten auf einem bestimmten Topic senden (publish) und andere Nodes können Nachrichten vom selben Topic empfangen (subscribe). Die Datenstruktur wird in Message-Diskriptor-Dateien definiert und wird beim kompilieren in mehrere Programmiersprachen übersetzt. Die Daten können dabei vom Typ Integer, Float oder String sein. Zudem unterstützen die ROS-Messages auch Arrays und ähnliche Formate wie der "Strukt" in C.

A.1.6 Services

ROS-Services stellen neben ROS-Topics eine weitere Art der Kommunikation zwischen den ROS-Nodes dar. Jede ROS-Node kann einen ROS-Service unter speziellem Namen anbieten, welcher immer nur einer ROS-Node zugewiesen werden kann. Anders als bei dem Nachrichtenaustausch über ROS-Topics wird ein ROS-Service von der zuständigen ROS-Node nur bearbeitet, wenn eine explizite Anfrage von einer anderen ROS-Node gestellt wird.

A.1.7 Libraries

ROS-Packages können ROS-Libraries (Bibliotheken) für andere ROS-Packages bereitstellen. Um eine ROS Library nutzen zu können, wird diese in der Package-Manifest-Datei mit den dazugehörigen Abhängigkeiten deklariert und die zugehörige Header-Datei im Quellcode der ROS-Node eingebunden. Das ROS spezifische Build System (catkin) linkt automatische die ausführbare ROS-Node-Datei mit der entsprechenden Bibliothek.

A.1.8 Master

Das ROS-System arbeitet nach dem Master-Slave Prinzip und kann immer nur einen ROS-Master im Gesamtsystem haben. Der ROS-Master hat dabei als Zwischenknoten die Aufgabe, die Kommunikation zwischen den ROS-Nodes herzustellen und zu verwalten. Der ROS-Master besitzt dabei alle nötigen Informationen zu allen ROS-Nodes und ihren ROS-Services in der ROS-Umgebung. Ohne eine laufende ROS-Master Instanz können sich ROS-Nodes im System nicht miteinander verbinden, kommunizieren und damit auch keine ROS-Messages untereinander austauschen.

A.1.9 Bags

ROS-Bags ist ein ROS-Dienstprogramm zum Aufzeichnen und Wiedergeben der ROS-Topics und der im System gesendeten ROS-Messages. Auf diese Weise können schwer erfassbare Sensordaten für den späteren Gebrauch gespeichert und archiviert werden. Zusätzlich bieten die ROS-Bags die Möglichkeit diese auf anderen Computern unter anderen Bedingungen zu übertragen und wiederzugeben. Hierdurch ist es möglich neu entwickelte Algorithmen im Labor mit realen Sensordaten zu testen.

A.2 Erstellte Matlab-Skripte zur Auswertung

A.2.1 plot_of_joints.m

```
clear all
close all
clc

%Parameters for the Platform
%Radius of the joints in meters/cm/mm ??
%Offset of the joint in degrees
radiusP=0.21;
offsetP=42;
%Parameters for the Base
%Radius of the joints in meters/cm/mm ??
%Offset of the joint in degrees
radiusB=0.408;
offsetB=7;
%Distanc between Base and Platform
height = 1.036; % +20cm

Px(1) = cosd(0 + offsetP)*radiusP;
Py(1) = sind(0 + offsetP)*radiusP;
Pz(1) = height-0.096;
Px(2) = cosd(0 - offsetP)*radiusP;
Py(2) = sind(0 - offsetP)*radiusP;
Pz(2) = height-0.096;
Px(3) = cosd(240 + offsetP)*radiusP;
Py(3) = sind(240 + offsetP)*radiusP;
Pz(3) = height-0.096;
Px(4) = cosd(240 - offsetP)*radiusP;
Py(4) = sind(240 - offsetP)*radiusP;
Pz(4) = height-0.096;
Px(5) = cosd(120 + offsetP)*radiusP;
Py(5) = sind(120 + offsetP)*radiusP;
Pz(5) = height-0.096;
Px(6) = cosd(120 - offsetP)*radiusP;
Py(6) = sind(120 - offsetP)*radiusP;
Pz(6) = height-0.096;

Bx(1) = cosd(0 + offsetB)*radiusB;
By(1) = sind(0 + offsetB)*radiusB;
Bz(1) = 0.095;
```

```
Bx(2) = cosd(0 - offsetB)*radiusB;
By(2) = sind(0 - offsetB)*radiusB;
Bz(2) = 0.095;
Bx(3) = cosd(240 + offsetB)*radiusB;
By(3) = sind(240 + offsetB)*radiusB;
Bz(3) = 0.095;
Bx(4) = cosd(240 - offsetB)*radiusB;
By(4) = sind(240 - offsetB)*radiusB;
Bz(4) = 0.095;
Bx(5) = cosd(120 + offsetB)*radiusB;
By(5) = sind(120 + offsetB)*radiusB;
Bz(5) = 0.095;
Bx(6) = cosd(120 - offsetB)*radiusB;
By(6) = sind(120 - offsetB)*radiusB;
Bz(6) = 0.095;

%%Plotting
axis([-0.7 0.7 -0.7 0.7 -0.1 1.2])
hold on
%Mobile-Plattform
plot3(Px(:),Py(:),Pz(:),'ro')
plot3(0,0,height,'bo')
%Basis-Plattform
plot3(Bx(:),By(:),Bz(:),'go')
plot3(0,0,0,'bo')
grid on;
```

Listing A.1: Matlab-Skript zur Berechnung der Gelenkkoordinaten

A.2.2 workspace_calculation.m

```
clc;
clear all;

% Minimale und maximale Länge der Linearantriebe
l_min_limit = 0.88;
l_max_limit = 1.33;
% Diskrete Unterteilung des zu untersuchenden Raums [m]:
x = -1.5:0.01:1.5;
y = -1.5:0.005:0;
z = 0:0.01:4;
% Orientierung der mobilen-Plattform [°]
phi = 0;
theta = 0;
psi = 0;
% Leeres Feld zum Speichern der Ergebnisse
valid = zeros(length(x), length(y), length(z), length(phi),length(theta),
    length(psi));

%[a_gelenk,b_gelenk] = eingang_Gelenk_KO;
for i = 1:length(x)
    for j = 1:length(y)
        for k = 1:length(z)
            for l = 1:length(phi)
                for m = 1:length(theta)
                    for n = 1:length(psi)
                        % Berechnung der Länge für vorgegebene Position
                        [leg] = position_mobile(x(i), y(j), z(k), phi, theta,
     psi);
                        for p = 1:length(l)
                            % Prüfung ob der untersuchte Punkt Teil des
                            % Arbeitsraums ist
                            if ((min(leg) > l_min_limit) && (max(leg) <
    l_max_limit))
                            valid(j, i, k) = 1;
                            end
                        end
                    end
                end
            end
        end
    end
end
```

```
% Graphische Darstellung
figure;
xlabel('x'), ylabel('y'),zlabel('z'), title('Arbeitsraum');
hold on;
grid on;
P=valid;
% Berechnung der Oberfläche aus dem Arbeitsraum-Volumen
isosf=isosurface(x,y,z,P,0.0001);
% Farbe und Beleuchtung
p1=patch(isosf,'FaceColor','r','EdgeColor','none');
daspect([1 1 1])
view([-170 20]);
camlight;
camlight(-180,-10);
lighting phong

%%
function output = position_mobile(x, y, z, phi, theta, psi)

radiusP=0.21;
offsetP=42;

radiusB=0.408;
offsetB=7;

Px1 = cosd(0 + offsetP + psi)*radiusP + x;
Py1 = sind(0 + offsetP + psi)*radiusP + y;
Pz1 = z;
Px2 = cosd(0 - offsetP + psi)*radiusP + x;
Py2 = sind(0 - offsetP + psi)*radiusP + y;
Pz2 = z;
Px3 = cosd(240 + offsetP + psi)*radiusP + x;
Py3 = sind(240 + offsetP + psi)*radiusP + y;
Pz3 = z;
Px4 = cosd(240 - offsetP + psi)*radiusP + x;
Py4 = sind(240 - offsetP + psi)*radiusP + y;
Pz4 = z;
Px5 = cosd(120 + offsetP + psi)*radiusP + x;
Py5 = sind(120 + offsetP + psi)*radiusP + y;
Pz5 = z;
Px6 = cosd(120 - offsetP + psi)*radiusP + x;
Py6 = sind(120 - offsetP + psi)*radiusP + y;
Pz6 = z;
```

```
84  Bx1 = cosd(0 + offsetB)*radiusB;
    By1 = sind(0 + offsetB)*radiusB;
86  Bz1 = 0;
    Bx2 = cosd(0 - offsetB)*radiusB;
88  By2 = sind(0 - offsetB)*radiusB;
    Bz2 = 0;
90  Bx3 = cosd(240 + offsetB)*radiusB;
    By3 = sind(240 + offsetB)*radiusB;
92  Bz3 = 0;
    Bx4 = cosd(240 - offsetB)*radiusB;
94  By4 = sind(240 - offsetB)*radiusB;
    Bz4 = 0;
96  Bx5 = cosd(120 + offsetB)*radiusB;
    By5 = sind(120 + offsetB)*radiusB;
98  Bz5 = 0;
    Bx6 = cosd(120 - offsetB)*radiusB;
100 By6 = sind(120 - offsetB)*radiusB;
    Bz6 = 0;
102
    %Calculated Length of each Leg
104 L1=sqrt((Px1-Bx1)^2+(Py1-By1)^2+(Pz1-Bz1)^2);
    L2=sqrt((Px2-Bx2)^2+(Py2-By2)^2+(Pz2-Bz2)^2);
106 L3=sqrt((Px3-Bx3)^2+(Py3-By3)^2+(Pz3-Bz3)^2);
    L4=sqrt((Px4-Bx4)^2+(Py4-By4)^2+(Pz4-Bz4)^2);
108 L5=sqrt((Px5-Bx5)^2+(Py5-By5)^2+(Pz5-Bz5)^2);
    L6=sqrt((Px6-Bx6)^2+(Py6-By6)^2+(Pz6-Bz6)^2);
110
    output = [L1, L2, L3, L4, L5, L6];
112 end
```

Listing A.2: Matlab-Skript zur Berechnung des Arbeitsraumvolumens

A.3 Erstellter Programmcode der Motorcontroller

A.3.1 uC_Motor_Node.ino

```
#include "Param_conf.h"
#include "PID_conf.h"
#include "Counter_conf.h"
#include "WiFi_conf.h"
#include "OTA_conf.h"
#include "PWM_conf.h"
#include "ROS_conf.h"
#include "StateMachine.h"


void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  delay(25);
  PID_Ini();
  PWM_setup();
  delay(25);
  WiFi_setup();
  delay(25);
  OTA_setup();
  delay(25);
  ROS_setup();
  delay(25);
  Counter_setup();
  delay(25);
#if DEBUG
  Serial.println("Booting finished");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
#endif
}

void loop() {
  unsigned long currentMillis = millis();
  ArduinoOTA.handle();

  if (currentMillis - previousMillis >= interval) {     // Loop rate 100Hz
    for speed control
    previousMillis = currentMillis; // save the last time
```

```
40      PID_Vel(m1_vel_msg.process_value, m1_vel_msg.set_point, &M1_pid_vel);
        PID_Vel(m2_vel_msg.process_value, m2_vel_msg.set_point, &M2_pid_vel);
42      if (M1_pid_vel.y > 0) {
          m1_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, M1_pid_vel.y);
44      }
        else if (M1_pid_vel.y < 0) {
46        m1_backward(MCPWM_UNIT_0, MCPWM_TIMER_0, -M1_pid_vel.y);
        } else {
48        m1_stop(MCPWM_UNIT_0, MCPWM_TIMER_0);
        }
50      if (M2_pid_vel.y > 0) {
          m2_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, M2_pid_vel.y);
52      }
        else if (M2_pid_vel.y < 0) {
54        m2_backward(MCPWM_UNIT_0, MCPWM_TIMER_0, -M2_pid_vel.y);
        } else {
56        m2_stop(MCPWM_UNIT_0, MCPWM_TIMER_0);
        }
58

    }
60

    if (currentMillis - previousMillis3 >= interval3) {  // Loop rate 10Hz for
      speed and acceleration to set zero
62      previousMillis3 = currentMillis; // save the last time

64      if ((millis() - M1_debounceTimeout) >= (5 * interval3)) {
          portENTER_CRITICAL_ISR(&mux);
66        M1_Velocity_last = M1_Velocity;
          M1_Acceleration_last = M1_Acceleration;
68        M1_Velocity = (((M1_Position - M1_Position_last) * 0.00127) / (5 *
      interval3));
          M1_Acceleration = ((M1_Velocity - M1_Velocity_last) / (5 * interval3));
70        portEXIT_CRITICAL_ISR(&mux);
          M1_Position_last = M1_Position;
72      }
        if ((millis() - M2_debounceTimeout) >= (5 * interval3)) {
74        portENTER_CRITICAL_ISR(&mux);
          M2_Velocity_last = M1_Velocity;
76        M2_Acceleration_last = M2_Acceleration;
          M2_Velocity = (((M2_Position - M2_Position_last) * 0.00127) / (5 *
      interval3));
78        M2_Acceleration = ((M2_Velocity - M2_Velocity_last) / (5 * interval3));
          portEXIT_CRITICAL_ISR(&mux);
```

```
80        M2_Position_last = M2_Position;
      }
82    }

84    if (currentMillis - previousMillis2 >= interval2) {  // Loop rate 50Hz for
        ros communication
      previousMillis2 = currentMillis; // save the last time
86
      if (nh.connected()) {
88  #if DEBUG
        Serial.println("Connected to ROS Master");
90  #endif

92  #if CONTROLLER_1
        m1_pos_msg.header.frame_id = "/m1_pos_state";
94      m2_pos_msg.header.frame_id = "/m2_pos_state";
        m1_vel_msg.header.frame_id = "/m1_vel_state";
96      m2_vel_msg.header.frame_id = "/m2_vel_state";
  #elif CONTROLLER_2
98      m1_pos_msg.header.frame_id = "/m3_pos_state";
        m2_pos_msg.header.frame_id = "/m4_pos_state";
100     m1_vel_msg.header.frame_id = "/m3_vel_state";
        m2_vel_msg.header.frame_id = "/m4_vel_state";
102 #elif CONTROLLER_3
        m1_pos_msg.header.frame_id = "/m5_pos_state";
104     m2_pos_msg.header.frame_id = "/m6_pos_state";
        m1_vel_msg.header.frame_id = "/m5_vel_state";
106     m2_vel_msg.header.frame_id = "/m6_vel_state";
  #endif
108


110     m1_pos_msg.process_value = (float)M1_Position;
        m1_pos_msg.process_value_dot = M1_Velocity;
112     m1_pos_msg.error = (float)M1_pid_pos.e;
        m1_pos_msg.time_step = (float)M1_pid_pos.Ta;
114     m1_pos_msg.command = (float)M1_pid_pos.y;
        m1_pos_msg.header.seq = seq;
116     m1_pos_msg.header.stamp = nh.now();
        m1_pos_pub.publish( &m1_pos_msg );
118
        m2_pos_msg.process_value = (float)M2_Position;
120     m2_pos_msg.process_value_dot = M2_Velocity;
        m2_pos_msg.error = (float)M2_pid_pos.e;
122     m2_pos_msg.time_step = (float)M2_pid_pos.Ta;
```

```
      m2_pos_msg.command = (float)M2_pid_pos.y;
124   m2_pos_msg.header.seq = seq;
      m2_pos_msg.header.stamp = nh.now();
126   m2_pos_pub.publish( &m2_pos_msg );

128   m1_vel_msg.set_point = m1_pos_msg.command / 1000;
      m1_vel_msg.process_value = M1_Velocity;
130   m1_vel_msg.process_value_dot = M1_Acceleration;
      m1_vel_msg.error = (float)M1_pid_vel.e;
132   m1_vel_msg.time_step = (float)M1_pid_vel.Ta;
      m1_vel_msg.command = (float)M1_pid_vel.y;
134   m1_vel_msg.header.seq = seq;
      m1_vel_msg.header.stamp = nh.now();
136   m1_vel_pub.publish( &m1_vel_msg );

138   m2_vel_msg.set_point = m2_pos_msg.command / 1000;
      m2_vel_msg.process_value = M2_Velocity;
140   m2_vel_msg.process_value_dot = M2_Acceleration;
      m2_vel_msg.error = (float)M2_pid_vel.e;
142   m2_vel_msg.time_step = (float)M2_pid_vel.Ta;
      m2_vel_msg.command = (float)M2_pid_vel.y;
144   m2_vel_msg.header.seq = seq;
      m2_vel_msg.header.stamp = nh.now();
146   m2_vel_pub.publish( &m2_vel_msg );

148   StateMachine();

150   seq++;
      nh.spinOnce();
152  } else {
#if DEBUG
154   Serial.println("Not Connected to ROS Master");
#endif
156   if (i >= 25) { // Try reconnection every 0.5 Seconds
        nh.spinOnce();
158 #if DEBUG
        Serial.println("Connecting to ROS Master");
160 #endif
        i = 0;
162   }
      i++;
164  }
 }
166
```

```
168   if (M1_Counter > 0) {
      portENTER_CRITICAL(&mux);
170   M1_Counter--;
      if (M1_Velocity > 0) {
172     M1_Position++;
      } else if (M1_Velocity < 0) {
174     M1_Position--;
      } else {
176     if (M1_Velocity_last > 0) {
          M1_Position++;
178     } else if (M1_Velocity_last < 0) {
          M1_Position--;
180     } else {
          if (digitalRead(M1_DIR)) {
182         M1_Position--;
          } else {
184         M1_Position++;
          }
186     }
      }
188   portEXIT_CRITICAL(&mux);
#if DEBUG
190   Serial.print("An interrupt has occurred. M1 Total: ");
      Serial.println(M1_Position);
192 #endif
    }
194
    if (M2_Counter > 0) {
196   portENTER_CRITICAL(&mux);
      M2_Counter--;
198   if (M2_Velocity > 0) {
        M2_Position++;
200   } else if (M2_Velocity < 0) {
        M2_Position--;
202   } else {
        if (M2_Velocity_last > 0) {
204       M2_Position++;
        } else if (M2_Velocity_last < 0) {
206       M2_Position--;
        } else {
208       if (digitalRead(M2_DIR)) {
            M2_Position--;
210       } else {
```

```
            M2_Position++;
212         }
          }
214     }
    portEXIT_CRITICAL(&mux);
216 #if DEBUG
    Serial.print("An interrupt has occurred. M2 Total: ");
218 Serial.println(M2_Position);
#endif
220   }
}
```

Listing A.3: Quellcode der Arduino-Projekts der Motor-Controller

A.3.2 Param_conf.h

```
#ifndef PARAM_CONF_H
#define PARAM_CONF_H

//generic properties
#define DEBUG 0                         // Print debug information
#define LED_BUILTIN 1                   // onboard LED pin
#define CONTROLLER_1 1                  // Select controller number 1
#define CONTROLLER_2 0                  // Select controller number 2
#define CONTROLLER_3 0                  // Select controller number 3

unsigned long previousMillis = 0;       // Store the last time (milliseconds)
unsigned long previousMillis2 = 0;      // Store the last time (milliseconds)
unsigned long previousMillis3 = 0;      // Store the last time (milliseconds)
unsigned long deltaMillis = 0;          // Store the delta time (milliseconds)
unsigned long startMillis = 0;          // Store the start time (milliseconds)
unsigned long stopMillis = 0;           // Store the stop time (milliseconds)
const uint8_t interval = 10;            // Delay interval (milliseconds)
const uint8_t interval2 = 20;           // Delay interval (milliseconds)
const uint8_t interval3 = 100;           // Delay interval (milliseconds)
uint16_t interval_number = 0;           // Interval counter
uint8_t i_counter = 0;                  // Interval counter
uint32_t seq = 0;                       // Sequence counter

uint8_t i = 0;
uint8_t counter = 0;
uint8_t var_select = 0;

float MAX_Accel = 0;
float MAX_Vel = 0;
float RISE_Vel = 0;
unsigned long MAXMillis = 0;
unsigned long Tu = 0;
unsigned long Tg = 0;

// Motor 1
int m1_dir = 0;
int m1_pwm = 0;

// Motor 2
int m2_dir = 0;
int m2_pwm = 0;

```

```
// Counter Configuration
#define M1_COUNTER_PIN 21
#define M2_COUNTER_PIN 22
#define DEBOUNCETIME 5

volatile int M1_Counter = 0;
volatile int M2_Counter = 0;
volatile int M1_Position = 0;
volatile int M2_Position = 0;
volatile int M1_Position_last = 0;
volatile int M2_Position_last = 0;
volatile float M1_Velocity = 0;
volatile float M2_Velocity = 0;
volatile float M1_Velocity_last = 0;
volatile float M2_Velocity_last = 0;
volatile float M1_Acceleration = 0;
volatile float M2_Acceleration = 0;
volatile float M1_Acceleration_last = 0;
volatile float M2_Acceleration_last = 0;

volatile uint32_t M1_debounceTimeout = 0;
volatile uint32_t M2_debounceTimeout = 0;

// PID Controller properties
#define P 225
#define I 100
#define D 100
#define I_clamp 15
#define Antiwindup true

// PWM Configuragtion
#define GPIO_PWM0A_OUT 15           // Set GPIO 15 as PWM0A M1
#define GPIO_PWM0B_OUT 16           // Set GPIO 16 as PWM0B M2
#define M1_DIR 2                   // Set GPIO 15 as direction pin M1
#define M2_DIR 33                   // Set GPIO 16 as direction pin M2
#define ACTIVE_DRIVE 19             // Set GPIO 19 as motor activator pin
#define BIAS 10

// Setting PWM properties
const int freq = 5000;
const int pwmChannel = 0;
const int resolution = 8;
int dutyCycle = 0;

```

```
#endif
```

Listing A.4: Quellcode zur Konfiguration allgemeiner Variablen der Motor-Controller

A.3.3 PID_conf.h

```
#ifndef PID_CONF_H
#define PID_CONF_H

typedef struct {
  int Ta;                       // Abtastzeit in ms
  float Ki;                     // Integralanteil
  float Kp;                     // VerstÃ¤rkung
  float Kd;                     // Differenzieller Anteil
  float e;                      // Regelabweichung
  float ealt;                   // Regelabweichung zum ZP z-1
  float iterm;                  // Summe der Regelabweichungen
  float y;                      // Reglerausgabe
  int y_max;                    // Regler Max Ausgang
  float i_clamp;
  bool antiwindup;
} PID_setup;                    // Struktur PID_Einstellungen erzeugen

PID_setup M1_pid_vel;           // Regler M1 velocity
PID_setup M2_pid_vel;           // Regler M2 velocity
PID_setup M1_pid_pos;           // Regler M1 position
PID_setup M2_pid_pos;           // Regler M2 position

void PID_Init(void) {            // In der Init mÃ¼ssen die
    Reglereinstellungen gemacht werden

  M1_pid_vel.y = 0;
  M1_pid_vel.y_max = 100;
  M1_pid_vel.Ta = 10;
  M2_pid_vel.Kp = 225;
  M2_pid_vel.Ki = I;
  M2_pid_vel.Kd = D;
  M1_pid_vel.iterm = 0;
  M1_pid_vel.e = 0;
  M1_pid_vel.ealt = 0;
  M1_pid_vel.i_clamp = I_clamp;
  M1_pid_vel.antiwindup = Antiwindup;

  M2_pid_vel.y = 0;
  M2_pid_vel.y_max = 100;
  M2_pid_vel.Ta = 10;
  M2_pid_vel.Kp = 225;
  M2_pid_vel.Ki = I;
```

```
42    M2_pid_vel.Kd = D;
      M2_pid_vel.iterm = 0;
44    M2_pid_vel.e = 0;
      M2_pid_vel.ealt = 0;
46    M2_pid_vel.i_clamp = I_clamp;
      M2_pid_vel.antiwindup = Antiwindup;
48
      M1_pid_pos.y = 0;
50    M1_pid_pos.y_max = 100;
      M1_pid_pos.Ta = 100;
52    M1_pid_pos.Kp = P;
      M1_pid_pos.Ki = 0;
54    M1_pid_pos.Kd = 0;
      M1_pid_pos.iterm = 0;
56    M1_pid_pos.e = 0;
      M1_pid_pos.ealt = 0;
58    M1_pid_pos.i_clamp = 0;
      M1_pid_pos.antiwindup = false;
60
      M2_pid_pos.y = 0;
62    M2_pid_pos.y_max = 100;
      M2_pid_pos.Ta = 100;
64    M2_pid_pos.Kp = P;
      M2_pid_pos.Ki = 0;
66    M2_pid_pos.Kd = 0;
      M2_pid_pos.iterm = 0;
68    M2_pid_pos.e = 0;
      M2_pid_pos.ealt = 0;
70    M2_pid_pos.i_clamp = 0;
      M2_pid_pos.antiwindup = false;
72  }

74  void PID_Vel (float process_value, float set_point, PID_setup* PID) {

76    PID->e = set_point - process_value;                          // aktuelle
        Regelabweichung bestimmen (setpoint - sensed_output)

78    if (PID->Ki != 0 && PID->Kd != 0) {                    // Deadband
        if (abs(PID->e) <= 0.002) {
80        PID->e = 0;
        }
82    }

84
```

```
  //  if (abs(PID->y) < PID->y_max) {                          // bei
    Uebersteuertem stellglied Integration einfrieren (Anti-Windup)
86 //    PID->esum += PID->e;                                   // Summe der
    Regelabweichung aktualisieren
  //  }
88
  PID->iterm += (PID->Ki * PID->Ta * PID->e);
90
  if (PID->iterm > PID->i_clamp && PID->antiwindup) {
92   PID->iterm = PID->i_clamp;
  } else if (PID->iterm < -PID->i_clamp && PID->antiwindup) {
94   PID->iterm = -PID->i_clamp;
  }
96
  PID->y = (PID->Kp * PID->e) + PID->iterm + (PID->Kd * ((PID->e - PID->ealt)
    ) / PID->Ta); // Reglergleichung
98
  PID->ealt = PID->e;                                  // Regelabweichung fÃ¼r
     nÃ¤chste Abtastung merken
100
  if (set_point > 0) {
102   if (PID->y > PID->y_max) {                          // Stellgroesse auf
    0-Max begrenzen
      PID->y = PID->y_max;
104   }
    if (PID->y < 0) {
106   PID->y = 0;
    }
108 } else if (set_point < 0) {
    if (PID->y > 0) {                                    // Stellgroesse auf
    Min-0 begrenzen
110   PID->y = 0;
    }
112   if (PID->y < -PID->y_max) {
      PID->y = -PID->y_max;
114   }
  } else {
116   PID->y = 0;
  }
118 // Stellgroesse zurÃ¼ckgeben
}
120
void PID_Pos (float process_value, float set_point, PID_setup* PID) {
122
```

```
  PID->e = set_point - process_value;                              // aktuelle
    Regelabweichung bestimmen (setpoint - sensed_output)
124
  PID->iterm += (PID->Ki * PID->Ta * PID->e);
126
  if (PID->iterm > PID->i_clamp && PID->antiwindup) {
128   PID->iterm = PID->i_clamp;
  } else if (PID->iterm < -PID->i_clamp && PID->antiwindup) {
130   PID->iterm = -PID->i_clamp;
  }
132
  PID->y = (PID->Kp * PID->e) + PID->iterm + (PID->Kd * ((PID->e - PID->ealt)
    ) / PID->Ta); // Reglergleichung
134
  PID->ealt = PID->e;                                  // Regelabweichung fÃ¼r
     nÃ¤chste Abtastung merken
136
  if (PID->y > PID->y_max) {                           // Stellgroesse auf 0-
    Max begrenzen
138   PID->y = PID->y_max;
  } else if (PID->y < -PID->y_max) {
140   PID->y = -PID->y_max;
  }
142
  // Stellgroesse zurÃ¼ckgeben
144 }

146 #endif
```

Listing A.5: Quellcode zur PID Konfiguration der Motor-Controller

A.3.4 Counter_conf.h

```
#ifndef COUNTER_CONF_H
#define COUNTER_CONF_H


portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR M1_Interrupt() {
  portENTER_CRITICAL_ISR(&mux);
  if ((millis() - M1_debounceTimeout) >= DEBOUNCETIME) {
    M1_Counter++;
    M1_Velocity_last = M1_Velocity;
    M1_Acceleration_last = M1_Acceleration;
    if (digitalRead(M1_DIR)) {
      M1_Velocity = -(0.00127 / (millis() - M1_debounceTimeout));
      M1_Acceleration = -((M1_Velocity - M1_Velocity_last) / (millis() -
    M1_debounceTimeout));
    } else {
      M1_Velocity = (0.00127 / (millis() - M1_debounceTimeout));
      M1_Acceleration = ((M1_Velocity - M1_Velocity_last) / (millis() -
    M1_debounceTimeout));
    }
    M1_debounceTimeout = millis();
  }
  portEXIT_CRITICAL_ISR(&mux);
}

void IRAM_ATTR M2_Interrupt() {
  portENTER_CRITICAL_ISR(&mux);
  if ((millis() - M2_debounceTimeout) >= DEBOUNCETIME) {
    M2_Counter++;
    M2_Velocity_last = M2_Velocity;
    M2_Acceleration_last = M2_Acceleration;
    if (digitalRead(M2_DIR)) {
      M2_Velocity = -(0.00127 / (millis() - M2_debounceTimeout));
      M2_Acceleration = -((M2_Velocity - M2_Velocity_last) / (millis() -
    M2_debounceTimeout));
    } else {
      M2_Velocity = (0.00127 / (millis() - M2_debounceTimeout));
      M2_Acceleration = ((M2_Velocity - M2_Velocity_last) / (millis() -
    M2_debounceTimeout));
    }
    M2_debounceTimeout = millis();
```

```
40    }
    portEXIT_CRITICAL_ISR(&mux);
42  }

44  void Counter_setup(void) {
    pinMode(M1_COUNTER_PIN, INPUT_PULLUP);
46    pinMode(M2_COUNTER_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(M1_COUNTER_PIN), M1_Interrupt,
      FALLING);
48    attachInterrupt(digitalPinToInterrupt(M2_COUNTER_PIN), M2_Interrupt,
      FALLING);
  #if DEBUG
50    Serial.println("Counter ready");
  #endif
52  }


54

  #endif
```

Listing A.6: Quellcode zur Konfiguration der Hall-Sensor ZÃ¤hler auf dem Motor-Controller

A.3.5 WiFi_conf.h

```
#ifndef WIFI_CONF_H
#define WIFI_CONF_H

#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>



/*********************************/
const char* ssid[] = {
  "FRITZBox-Bassen",
  "dhpi-Gast"
};

const char* password[] = {
  "****************",
  "****************"
};

#define NETWORKS (sizeof(ssid) / sizeof(const char*))

void WiFi_setup(void) {
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();

  if (n == 0) {
#if DEBUG
    Serial.println("no networks found");
#endif
  } else {
#if DEBUG
    Serial.print(n);
    Serial.println(" WiFi networks found");
#endif
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < NETWORKS; ++j) {
        if (WiFi.SSID(i) == ssid[j])
```

```
        {
          WiFi.begin(ssid[j], password[j]);
#if DEBUG
          Serial.print("Connecting to ");
          Serial.print(WiFi.SSID(i));
          Serial.print(", RSSI= ");
          Serial.println(WiFi.RSSI(i));
#endif
        }
      }
    }
  }


  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
#if DEBUG
    Serial.println("Connection Failed! Rebooting...");
#endif
    delay(5000);
    ESP.restart();
  }
#if DEBUG
  Serial.println("WiFi ready");
#endif
}


#endif
```

Listing A.7: Quellcode zur Wifi Konfiguration der Motor-Controller

A.3.6 OTA_conf.h

```
#ifndef OTA_CONF_H
#define OTA_CONF_H

#include <ArduinoOTA.h>

void OTA_setup(void) {
  // Port defaults to 3232
  ArduinoOTA.setPort(3232);

  // Hostname defaults to esp3232-[MAC]
#if CONTROLLER_1
  ArduinoOTA.setHostname("ESP-Control-1");
#elif CONTROLLER_2
  ArduinoOTA.setHostname("ESP-Control-2");
#elif CONTROLLER_3
  ArduinoOTA.setHostname("ESP-Control-3");
#endif

  // Password can be set with it's md5 value
  ArduinoOTA.setPasswordHash("c64c1752006a10e7b7cdbd7f5ea3c512");

  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
#if DEBUG
    Serial.println("Start updating " + type);
#endif
  })
  .onEnd([]() {
#if DEBUG
    Serial.println("\nEnd");
#endif
  })
  .onProgress([](unsigned int progress, unsigned int total) {
#if DEBUG
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
#endif
  })
```

```
  .onError([](ota_error_t error) {
#if DEBUG
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
#endif
  });

  ArduinoOTA.begin();
#if DEBUG
  Serial.println("OTA ready");
#endif
}


#endif
```

Listing A.8: Quellcode zur OTA Konfiguration der Motor-Controller

A.3.7 PWM_conf.h

```
#ifndef PWM_CONF_H
#define PWM_CONF_H

#include <driver/mcpwm.h>

static void mcpwm_gpio_initialize(void)
{
#if DEBUG
  printf("initializing mcpwm gpio...\n");
#endif
  mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, GPIO_PWM0A_OUT);
  mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0B, GPIO_PWM0B_OUT);
}

/**
   brief motor moves in forward direction, with duty cycle
*/
static void m1_forward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num ,
    float duty_cycle)
{
  digitalWrite(M1_DIR, LOW);
  mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_A, BIAS + duty_cycle);
  mcpwm_set_duty_type(mcpwm_num, timer_num, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);
    //call this each time, if operator was previously in low/high state
}

/**
   brief motor moves in backward direction, with duty cycle
*/
static void m1_backward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num ,
    float duty_cycle)
{
  digitalWrite(M1_DIR, HIGH);
  mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_A, BIAS + duty_cycle);
  mcpwm_set_duty_type(mcpwm_num, timer_num, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);
    //call this each time, if operator was previously in low/high state
}
/**
   brief motor moves in forward direction, with duty cycle
*/
static void m2_forward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num ,
    float duty_cycle)
```

```
38 {
     digitalWrite(M2_DIR, LOW);
40   mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_B, BIAS + duty_cycle);
     mcpwm_set_duty_type(mcpwm_num, timer_num, MCPWM_OPR_B, MCPWM_DUTY_MODE_0);
        //call this each time, if operator was previously in low/high state
42 }
   /**
44    brief motor moves in backward direction, with duty cycle
   */
46 static void m2_backward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num ,
       float duty_cycle)
   {
48   digitalWrite(M2_DIR, HIGH);
     mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_B, BIAS + duty_cycle);
50   mcpwm_set_duty_type(mcpwm_num, timer_num, MCPWM_OPR_B, MCPWM_DUTY_MODE_0);
       //call this each time, if operator was previously in low/high state
   }
52
      //brief motor stop
54
   static void m1_stop(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num)
56 {
     mcpwm_set_signal_low(mcpwm_num, timer_num, MCPWM_OPR_A);
58 }

60    //brief motor stop

62 static void m2_stop(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num)
   {
64   mcpwm_set_signal_low(mcpwm_num, timer_num, MCPWM_OPR_B);
   }
66
   void PWM_setup(void) {
68   mcpwm_gpio_initialize();

70 #if DEBUG
     printf("Configuring Initial Parameters of mcpwm...\n");
72 #endif
     mcpwm_config_t pwm_config;
74   pwm_config.frequency = 5000;     //frequency = 5kHz,
     pwm_config.cmpr_a = 0;           //duty cycle of PWMxA = 0
76   pwm_config.cmpr_b = 0;           //duty cycle of PWMxb = 0
     pwm_config.counter_mode = MCPWM_UP_COUNTER;
78   pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
```

```
  mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);    //Configure PWM0A
    & PWM0B with above settings
80
  pinMode(M1_DIR, OUTPUT);
82  pinMode(M2_DIR, OUTPUT);
  pinMode(ACTIVE_DRIVE, OUTPUT);
84

86 #if DEBUG
  Serial.println("PWM ready");
88 #endif
}
90

92 #endif
```

Listing A.9: Quellcode zur PWM Konfiguration der Motor-Controller

A.3.8 ROS_conf.h

```
#ifndef ROS_CONF_H
#define ROS_CONF_H

#include <ros.h>
#include <std_msgs/String.h>
#include <geometry_msgs/WrenchStamped.h>
#include <geometry_msgs/Vector3.h>
#include <control_msgs/JointControllerState.h>
#include <sensor_msgs/JointState.h>

// Set the rosserial socket server IP address
IPAddress server(192, 168, 178, 98);
// Set the rosserial socket server port
#if CONTROLLER_1
const uint16_t serverPort = 11411;
#elif CONTROLLER_2
const uint16_t serverPort = 11412;
#elif CONTROLLER_3
const uint16_t serverPort = 11413;
#endif
/*********************************/
ros::NodeHandle nh;

// Make a joint_controller publisher
control_msgs::JointControllerState m1_pos_msg;
control_msgs::JointControllerState m2_pos_msg;
control_msgs::JointControllerState m1_vel_msg;
control_msgs::JointControllerState m2_vel_msg;

#if CONTROLLER_1
ros::Publisher m1_pos_pub("/M1/P_Controller_State", &m1_pos_msg);
ros::Publisher m1_vel_pub("/M1/V_Controller_State", &m1_vel_msg);
ros::Publisher m2_pos_pub("/M2/P_Controller_State", &m2_pos_msg);
ros::Publisher m2_vel_pub("/M2/V_Controller_State", &m2_vel_msg);
#elif CONTROLLER_2
ros::Publisher m1_pos_pub("/M3/P_Controller_State", &m1_pos_msg);
ros::Publisher m1_vel_pub("/M3/V_Controller_State", &m1_vel_msg);
ros::Publisher m2_pos_pub("/M4/P_Controller_State", &m2_pos_msg);
ros::Publisher m2_vel_pub("/M4/V_Controller_State", &m2_vel_msg);
#elif CONTROLLER_3
ros::Publisher m1_pos_pub("/M5/P_Controller_State", &m1_pos_msg);
ros::Publisher m1_vel_pub("/M5/V_Controller_State", &m1_vel_msg);
```

```
ros::Publisher m2_pos_pub("/M6/P_Controller_State", &m2_pos_msg);
ros::Publisher m2_vel_pub("/M6/V_Controller_State", &m2_vel_msg);
#endif

void M1_pos_Callback(const control_msgs::JointControllerState &msg)
{
#if CONTROLLER_1
  m1_pos_msg.header.frame_id = "/M1_pos_state";
#elif CONTROLLER_2
  m1_pos_msg.header.frame_id = "/M3_pos_state";
#elif CONTROLLER_3
  m1_pos_msg.header.frame_id = "/M5_pos_state";
#endif
  if (msg.set_point < 0) {
    m1_pos_msg.set_point = -10;
  } else {
    m1_pos_msg.set_point = msg.set_point;
  }
  m1_pos_msg.p = msg.p;
  m1_pos_msg.i = msg.i;
  m1_pos_msg.d = msg.d;
  m1_pos_msg.i_clamp = msg.i_clamp;
  m1_pos_msg.antiwindup = msg.antiwindup;
}

void M1_vel_Callback(const control_msgs::JointControllerState &msg)
{
#if CONTROLLER_1
  m1_vel_msg.header.frame_id = "/M1_vel_state";
#elif CONTROLLER_2
  m1_vel_msg.header.frame_id = "/M3_vel_state";
#elif CONTROLLER_3
  m1_vel_msg.header.frame_id = "/M5_vel_state";
#endif
  m1_vel_msg.set_point = msg.set_point;
  m1_vel_msg.p = msg.p;
  m1_vel_msg.i = msg.i;
  m1_vel_msg.d = msg.d;
  m1_vel_msg.i_clamp = msg.i_clamp;
  m1_vel_msg.antiwindup = msg.antiwindup;
}

void M2_pos_Callback(const control_msgs::JointControllerState &msg)
{
```

```
#if CONTROLLER_1
  m2_pos_msg.header.frame_id = "/M2_pos_state";
#elif CONTROLLER_2
  m2_pos_msg.header.frame_id = "/M4_pos_state";
#elif CONTROLLER_3
  m2_pos_msg.header.frame_id = "/M6_pos_state";
#endif
  if (msg.set_point < 0) {
      m2_pos_msg.set_point = -10;
  } else {
      m2_pos_msg.set_point = msg.set_point;
  }
  m2_pos_msg.p = msg.p;
  m2_pos_msg.i = msg.i;
  m2_pos_msg.d = msg.d;
  m2_pos_msg.i_clamp = msg.i_clamp;
  m2_pos_msg.antiwindup = msg.antiwindup;
}

void M2_vel_Callback(const control_msgs::JointControllerState &msg)
{
#if CONTROLLER_1
  m2_vel_msg.header.frame_id = "/M2_vel_state";
#elif CONTROLLER_2
  m2_vel_msg.header.frame_id = "/M4_vel_state";
#elif CONTROLLER_3
  m2_vel_msg.header.frame_id = "/M6_vel_state";
#endif

  m2_vel_msg.set_point = msg.set_point;
  m2_vel_msg.p = msg.p;
  m2_vel_msg.i = msg.i;
  m2_vel_msg.d = msg.d;
  m2_vel_msg.i_clamp = msg.i_clamp;
  m2_vel_msg.antiwindup = msg.antiwindup;
}

#if CONTROLLER_1
ros::Subscriber<control_msgs::JointControllerState> m1_pos_sub("/M1/
    P_Controller_Setting", M1_pos_Callback );
ros::Subscriber<control_msgs::JointControllerState> m1_vel_sub("/M1/
    V_Controller_Setting", M1_vel_Callback );
ros::Subscriber<control_msgs::JointControllerState> m2_pos_sub("/M2/
    P_Controller_Setting", M2_pos_Callback );
```

```
128 ros::Subscriber<control_msgs::JointControllerState> m2_vel_sub("/M2/
        V_Controller_Setting", M2_vel_Callback );
    #elif CONTROLLER_2
130 ros::Subscriber<control_msgs::JointControllerState> m1_pos_sub("/M3/
        P_Controller_Setting", M1_pos_Callback );
    ros::Subscriber<control_msgs::JointControllerState> m1_vel_sub("/M3/
        V_Controller_Setting", M1_vel_Callback );
132 ros::Subscriber<control_msgs::JointControllerState> m2_pos_sub("/M4/
        P_Controller_Setting", M2_pos_Callback );
    ros::Subscriber<control_msgs::JointControllerState> m2_vel_sub("/M4/
        V_Controller_Setting", M2_vel_Callback );
134 #elif CONTROLLER_3
    ros::Subscriber<control_msgs::JointControllerState> m1_pos_sub("/M5/
        P_Controller_Setting", M1_pos_Callback );
136 ros::Subscriber<control_msgs::JointControllerState> m1_vel_sub("/M5/
        V_Controller_Setting", M1_vel_Callback );
    ros::Subscriber<control_msgs::JointControllerState> m2_pos_sub("/M6/
        P_Controller_Setting", M2_pos_Callback );
138 ros::Subscriber<control_msgs::JointControllerState> m2_vel_sub("/M6/
        V_Controller_Setting", M2_vel_Callback );
    #endif
140


142 void ROS_setup(void) {
      // Set the connection to rosserial socket server
144   nh.getHardware()->setConnection(server, serverPort);
      nh.initNode();
146
    #if DEBUG
148   Serial.print("Node-IP = ");
      Serial.println(nh.getHardware()->getLocalIP());
150 #endif

152   // Start ROS nodes
      nh.advertise(m1_pos_pub);
154   nh.advertise(m2_pos_pub);
      nh.advertise(m1_vel_pub);
156   nh.advertise(m2_vel_pub);

158   nh.subscribe(m1_pos_sub);
      nh.subscribe(m2_pos_sub);
160   nh.subscribe(m1_vel_sub);
      nh.subscribe(m2_vel_sub);
162
```

```
  m1_pos_msg.p = M1_pid_pos.Kp;
  m1_pos_msg.i = M1_pid_pos.Ki;
  m1_pos_msg.d = M1_pid_pos.Kd;
  m1_pos_msg.i_clamp = M1_pid_pos.i_clamp;
  m1_pos_msg.antiwindup = Antiwindup;

  m2_pos_msg.p = M2_pid_pos.Kp;
  m2_pos_msg.i = M2_pid_pos.Ki;
  m2_pos_msg.d = M2_pid_pos.Kd;
  m2_pos_msg.i_clamp = M2_pid_pos.i_clamp;
  m2_pos_msg.antiwindup = Antiwindup;

  m1_vel_msg.p = M1_pid_vel.Kp;
  m1_vel_msg.i = M1_pid_vel.Ki;
  m1_vel_msg.d = M1_pid_vel.Kd;
  m1_vel_msg.i_clamp = M1_pid_vel.i_clamp;
  m1_vel_msg.antiwindup = Antiwindup;

  m2_vel_msg.p = M2_pid_vel.Kp;
  m2_vel_msg.i = M2_pid_vel.Ki;
  m2_vel_msg.d = M2_pid_vel.Kd;
  m2_vel_msg.i_clamp = M2_pid_vel.i_clamp;
  m2_vel_msg.antiwindup = Antiwindup;

#if DEBUG
  if (nh.connected()) {
    Serial.println("ROS ready");
  } else {
    Serial.println("ROS not ready");
  }
#endif
}


#endif
```

Listing A.10: Quellcode zur ROS Konfiguration der Motor-Controller

A.3.9 StateMachine.h

```
#ifndef STATEMACHINE_H
#define STATEMACHINE_H

typedef enum {
  start, idle, drive, error, tune
} state;
state State = start;
uint8_t starting = 0;
bool first_move = false;
bool M1_Dir = true;

static void StateMachine(void) {

  switch (State) {
    case start:
#if DEBUG
      Serial.println("start State");
#endif

      if (m1_pos_msg.set_point == (-10) && m2_pos_msg.set_point == (-10)) {
        digitalWrite(ACTIVE_DRIVE, HIGH);
        M1_Position = 0;
        M2_Position = 0;
        PID_Pos(m1_pos_msg.process_value, m1_pos_msg.set_point, &M1_pid_pos);
        PID_Pos(m2_pos_msg.process_value, m2_pos_msg.set_point, &M2_pid_pos);
        first_move = true;
      }
      if (first_move) {
        if (m1_pos_msg.set_point == (-1000) && m2_pos_msg.set_point ==
    (-1000)) {
          State = idle;
          M1_Position = 0;
          M2_Position = 0;
        }
      }


      break;
    case idle:
      PID_Pos(m1_pos_msg.process_value, 100, &M1_pid_pos);
      PID_Pos(m2_pos_msg.process_value, 100, &M2_pid_pos);
```

```
42        if (m1_pos_msg.set_point > 0 && m2_pos_msg.set_point > 0) {
            State = drive;
44        }
          if (m1_pos_msg.set_point == (-10) && m2_pos_msg.set_point == (-10)) {
46          digitalWrite(ACTIVE_DRIVE, LOW);
            first_move = false;
48          State = start;
          }
50

52  #if DEBUG
          Serial.println("idle State");
54  #endif
          break;
56      case drive:
          PID_Pos(m1_pos_msg.process_value, m1_pos_msg.set_point, &M1_pid_pos);
58        PID_Pos(m2_pos_msg.process_value, m2_pos_msg.set_point, &M2_pid_pos);
          if ((M1_pid_pos.y > 0)) {
60          m1_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, M1_pid_pos.y);
          } else if ((M1_pid_pos.y < 0)) {
62          m1_backward(MCPWM_UNIT_0, MCPWM_TIMER_0, -M1_pid_pos.y);
          } else {
64          m1_stop(MCPWM_UNIT_0, MCPWM_TIMER_0);
          }
66        if ((M2_pid_pos.y > 0)) {
            m2_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, M2_pid_pos.y);
68        } else if ((M2_pid_pos.y < 0)) {
            m2_backward(MCPWM_UNIT_0, MCPWM_TIMER_0, -M2_pid_pos.y);
70        } else {
            m2_stop(MCPWM_UNIT_0, MCPWM_TIMER_0);
72        }
          if (m1_pos_msg.set_point == (-1500) && m2_pos_msg.set_point == (-1500))
        {
74          digitalWrite(ACTIVE_DRIVE, LOW);
            first_move = false;
76          State = start;
          }
78

    #if DEBUG
80        Serial.println("drive State");
    #endif
82        break;
        case error:
84        digitalWrite(ACTIVE_DRIVE, LOW);
```

```
      if (m1_pos_msg.set_point == (-1500) && m2_pos_msg.set_point == (-1500))
     {
86      first_move = false;
        State = start;
88     }

90 #if DEBUG
       Serial.println("error State");
92 #endif
       break;
94

    case tune:
96     if (!first_move) {
         startMillis = millis();
98       digitalWrite(ACTIVE_DRIVE, HIGH);
         m1_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, 100);
100      first_move = true;
       }
102    if (interval_number > 20) {
         m1_stop(MCPWM_UNIT_0, MCPWM_TIMER_0);
104      State = idle;
         Serial.println("Start Millis:");
106      Serial.println(startMillis);
         Serial.println("Max_Accel");
108      Serial.println(MAX_Accel);
         Serial.println("RISE_Vel");
110      Serial.println(RISE_Vel);
         Serial.println("Max_Accel time");
112      Serial.println(MAXMillis);
         Serial.println("Max_Vel");
114      Serial.println(MAX_Vel);
         Serial.println("Stop Time:");
116      Serial.println(stopMillis);

118    }

120    ++interval_number;
       break;
122  default:
#if DEBUG
124    Serial.println("default State");
#endif
126    State = start;
       break;
```

```
128     }
    }
130
132 #endif
```

Listing A.11: Quellcode zur StateMachine der Motor-Controller

A.4 Erstellter Programmcode des IMU Controller

A.4.1 uC_IMU_Node.ino

```
#include <Wire.h>
#include <WiFi.h>
#include "OTA_conf.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <ros.h>

#include "BNO055_support.h"

#include <sensor_msgs/Imu.h>
#include <sensor_msgs/MagneticField.h>
#include <std_msgs/String.h>


//This structure contains the details of the BNO055 device that is connected.
struct bno055_t myBNO;
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

struct bno055_accel acc;
struct bno055_mag mag;
struct bno055_gyro gyr;
struct bno055_euler eul;
struct bno055_quaternion qur;
struct bno055_linear_accel lia;
struct bno055_gravity grvt;

// Calibration Parameter.
BNO055_S16 acc_off_x = -22;
BNO055_S16 acc_off_y = -13;
BNO055_S16 acc_off_z = -29;
BNO055_S16 mag_off_x =  -1820;
BNO055_S16 mag_off_y =  230;
BNO055_S16 mag_off_z =   70;
BNO055_S16 gyr_off_x =  -2;
BNO055_S16 gyr_off_y =  -2;
BNO055_S16 gyr_off_z =   1;
BNO055_S16 accel_radius = 1000;
BNO055_S16 mag_radius = 816;
```

```
40
unsigned char remap_axis;
42 unsigned char remap_sign_x;
unsigned char remap_sign_y;
44 unsigned char remap_sign_z;

46 //Variables and Flags
unsigned long lastTime = 0;
48
const char* ssid     = "*****";
50 const char* password = "*****";

52 /**** Set the rosserial socket server IP address ****/
IPAddress server(192, 168, 178, 98);
54 // Set the rosserial socket server port
const uint16_t serverPort = 11414;
56 /********************************/

58 ros::NodeHandle nh;

60 sensor_msgs::Imu imu_msg;
sensor_msgs::MagneticField mag_msg;
62
ros::Publisher pub_imu("imu", &imu_msg);
64 ros::Publisher pub_mag("mag", &mag_msg);

66 uint32_t seq = 0;

68 void setup() {
  /**** Initialize I2C communication ****/
70   Wire.begin();
  /**** Initialize ESP32 serial to monitor the process ****/
72   Serial.begin(115200);
  Serial.println();
74   Serial.print("Connecting to ");
  Serial.println(ssid);
76
  /**** Initialization of the BNO055 ****/
78   //Assigning the structure to hold information about the device
  BNO_Init(&myBNO);
80
  // Change BNO055 Mode to write calibration
82   bno055_set_operation_mode(OPERATION_MODE_CONFIG);
  delay(1);
```

```
84  // Write calibration values
    bno055_write_accel_offset_x_axis(acc_off_x);
86  bno055_write_accel_offset_y_axis(acc_off_y);
    bno055_write_accel_offset_z_axis(acc_off_z);
88  bno055_write_mag_offset_x_axis(mag_off_x);
    bno055_write_mag_offset_y_axis(mag_off_y);
90  bno055_write_mag_offset_z_axis(mag_off_z);
    bno055_write_gyro_offset_x_axis(gyr_off_x);
92  bno055_write_gyro_offset_y_axis(gyr_off_y);
    bno055_write_gyro_offset_z_axis(gyr_off_z);
94  bno055_write_accel_radius(accel_radius);
    bno055_write_mag_radius(mag_radius);
96  // output calibration values
    Serial.println("written calibration:");
98  Serial.println("Accel:");
    Serial.println(acc_off_x);
100 Serial.println(acc_off_y);
    Serial.println(acc_off_z);
102 Serial.println("Gyro:");
    Serial.println(gyr_off_x);
104 Serial.println(gyr_off_y);
    Serial.println(gyr_off_z);
106 Serial.println("Magnetic:");
    Serial.println(mag_off_x);
108 Serial.println(mag_off_y);
    Serial.println(mag_off_z);
110 Serial.println("Radius:");
    Serial.println(accel_radius);
112 Serial.println(mag_radius);
    Serial.println();
114
    /**** Axis remap of BNO055 ****/
116 bno055_set_axis_remap_value(DEFAULT_AXIS);
    bno055_get_axis_remap_value(&remap_axis);
118 Serial.println("Axis Config:");
    Serial.println(remap_axis, HEX);
120 Serial.println();

122 bno055_set_x_remap_sign(false);
    bno055_get_x_remap_sign(&remap_sign_x);
124 Serial.println("X Axis sign: 0=+ , 1=-");
    Serial.println(remap_sign_x);
126 Serial.println();
    bno055_set_y_remap_sign(false);
```

```
128   bno055_get_y_remap_sign(&remap_sign_y);
      Serial.println("Y Axis sign: 0=+ , 1=-");
130   Serial.println(remap_sign_y);
      Serial.println();
132   bno055_set_z_remap_sign(false);
      bno055_get_z_remap_sign(&remap_sign_z);
134   Serial.println("Z Axis sign: 0=+ , 1=-");
      Serial.println(remap_sign_z);
136   Serial.println();

138   //Configuration to NDoF mode
      bno055_set_operation_mode(OPERATION_MODE_NDOF);
140   delay(1);

142   // Connect the ESP32 to wifi AP
      WiFi.begin(ssid, password);
144   while (WiFi.status() != WL_CONNECTED) {
        delay(500);
146     Serial.print(".");
      }
148   Serial.println("");
      Serial.println("WiFi connected");
150   Serial.println("IP address: ");
      Serial.println(WiFi.localIP());
152   // Another way to get IP
      Serial.print("Node-IP = ");
154   Serial.println(nh.getHardware()->getLocalIP());
      Serial.println("Orientation Sensor Test"); Serial.println("");
156
      /**** Setup over-the-air programming ****/
158   OTA_setup();

160   /**** Set the connection to rosserial socket server ****/
      nh.getHardware()->setConnection(server, serverPort);
162   nh.initNode();

164   nh.advertise(pub_imu);
      nh.advertise(pub_mag);
166
      delay(1000);
168
      //Display on serial monitor the completion of the initialization
170   Serial.println("Initialization Complete");
  }
```

```
172
void loop() {
174   ArduinoOTA.handle();

176   imu::Quaternion quat = bno.getQuat();

178   bno055_read_mag_xyz(&mag);
      bno055_read_gyro_xyz(&gyr);
180   bno055_read_quaternion_wxyz(&qur);
      bno055_read_linear_accel_xyz(&lia);
182
      if (nh.connected()) {
184     imu_msg.orientation.x = quat.x();
        imu_msg.orientation.y = quat.y();
186     imu_msg.orientation.z = quat.z();
        imu_msg.orientation.w = quat.w();
188
        imu_msg.angular_velocity.x = ((double)gyr.x) / 16.0;
190     imu_msg.angular_velocity.y = ((double)gyr.y) / 16.0;
        imu_msg.angular_velocity.z = ((double)gyr.z) / 16.0;
192
        imu_msg.linear_acceleration.x = ((double)lia.x) / 100.0;
194     imu_msg.linear_acceleration.y = ((double)lia.y) / 100.0;
        imu_msg.linear_acceleration.z = ((double)lia.z) / 100.0;
196
        mag_msg.magnetic_field.x = ((double)mag.x) / 16.0;
198     mag_msg.magnetic_field.y = ((double)mag.y) / 16.0;
        mag_msg.magnetic_field.z = ((double)mag.z) / 16.0;
200
        imu_msg.orientation_covariance[0] = -1;
202     imu_msg.angular_velocity_covariance[0] = -1;
        imu_msg.linear_acceleration_covariance[0] = -1;
204     mag_msg.magnetic_field_covariance[0] = -1;

206     imu_msg.header.seq = seq;
        imu_msg.header.frame_id = "imu_BNO055";
208     imu_msg.header.stamp = nh.now();
        pub_imu.publish(&imu_msg);
210
        mag_msg.header.seq = seq;
212     mag_msg.header.frame_id = "imu_BNO055";
        mag_msg.header.stamp = nh.now();
214     pub_mag.publish(&mag_msg);
```

```
216   } else {
        Serial.println("Not Connected");
218   }

220   nh.spinOnce();
      seq++;
222   /* Wait the specified delay before requesting nex data */
      delay(5);
224
}
```

Listing A.12: Quellcode des IMU Mikrocontrollen zum Auslesen und Senden der IMU-Daten

A.4.2 OTA_conf.h

```
#ifndef OTA_CONF_H
#define OTA_CONF_H


#include <ArduinoOTA.h>

void OTA_setup(void) {
  // Port defaults to 3232
  ArduinoOTA.setPort(3232);

  // Hostname defaults to esp3232-[MAC]
  ArduinoOTA.setHostname("ESP-IMU-1");

  // Password can be set with it's md5 value as well
  ArduinoOTA.setPasswordHash("c64c1752006a10e7b7cdbd7f5ea3c512");

  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";
#if DEBUG
    Serial.println("Start updating " + type);
#endif
  })
  .onEnd([]() {
#if DEBUG
    Serial.println("\nEnd");
#endif
  })
  .onProgress([](unsigned int progress, unsigned int total) {
#if DEBUG
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
#endif
  })
  .onError([](ota_error_t error) {
#if DEBUG
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
```

```
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
#endif
  });

  ArduinoOTA.begin();
#if DEBUG
  Serial.println("OTA ready");
#endif
}


#endif
```

Listing A.13: Quellcode zur OTA Konfiguration der IMU

A.5 Erstellter Programmcode in ROS

A.5.1 stewart_base_node.cpp

```
#include <ros/ros.h>
#include <tf2_ros/transform_broadcaster.h>
#include <tf2/LinearMath/Quaternion.h>

#include <geometry_msgs/TransformStamped.h>
#include "geometry_msgs/PointStamped.h"

#include "tf2_ros/transform_listener.h"
#include "tf2_ros/message_filter.h"
#include "message_filters/subscriber.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"

#define BASE_RADIUS 0.408
#define BASE_HEIGHT 0.105
#define BASE_ANGLE 0.122173 // 7Â°xpi/180
#define PLATFORM_RADIUS 0.21
#define PLATFORM_HEIGHT 0.106
#define PLATFORM_ANGLE 0.733038 // 42Â°xpi/180
#define PI 3.14159265359

int main(int argc, char **argv)
{
    ros::init(argc, argv, "base_tf2_broadcaster");
    ros::NodeHandle node;

    tf2_ros::TransformBroadcaster tfb0;
    tf2_ros::TransformBroadcaster tfb1;
    tf2_ros::TransformBroadcaster tfb2;
    tf2_ros::TransformBroadcaster tfb3;
    tf2_ros::TransformBroadcaster tfb4;
    tf2_ros::TransformBroadcaster tfb5;
    tf2_ros::TransformBroadcaster tfb6;

    geometry_msgs::TransformStamped tf_B0;
    geometry_msgs::TransformStamped tf_B1;
    geometry_msgs::TransformStamped tf_B2;
    geometry_msgs::TransformStamped tf_B3;
    geometry_msgs::TransformStamped tf_B4;
    geometry_msgs::TransformStamped tf_B5;
```

```
40      geometry_msgs::TransformStamped tf_B6;

42

        tf_B0.header.frame_id = "base_footprint";
44      tf_B0.child_frame_id = "base_origin";
        tf_B0.transform.translation.x = 0.0;
46      tf_B0.transform.translation.y = 0.0;
        tf_B0.transform.translation.z = 0.0;
48      tf2::Quaternion q0;
        q0.setRPY(0, 0, 0);
50      tf_B0.transform.rotation.x = q0.x();
        tf_B0.transform.rotation.y = q0.y();
52      tf_B0.transform.rotation.z = q0.z();
        tf_B0.transform.rotation.w = q0.w();
54
        tf_B0.header.stamp = ros::Time::now();
56      tfb0.sendTransform(tf_B0);

58      tf_B1.header.frame_id = "base_origin";
        tf_B1.child_frame_id = "base_link1";
60      tf_B1.transform.translation.x = BASE_RADIUS * cos((0) + (BASE_ANGLE));
        tf_B1.transform.translation.y = BASE_RADIUS * sin((0) + (BASE_ANGLE));
62      tf_B1.transform.translation.z = BASE_HEIGHT;
        tf2::Quaternion q1;
64      q1.setRPY(0, 0, 0);
        tf_B1.transform.rotation.x = q1.x();
66      tf_B1.transform.rotation.y = q1.y();
        tf_B1.transform.rotation.z = q1.z();
68      tf_B1.transform.rotation.w = q1.w();

70      tf_B2.header.frame_id = "base_origin";
        tf_B2.child_frame_id = "base_link2";
72      tf_B2.transform.translation.x = BASE_RADIUS * cos((0) - (BASE_ANGLE));
        tf_B2.transform.translation.y = BASE_RADIUS * sin((0) - (BASE_ANGLE));
74      tf_B2.transform.translation.z = BASE_HEIGHT;
        tf2::Quaternion q2;
76      q2.setRPY(0, 0, 0);
        tf_B2.transform.rotation.x = q2.x();
78      tf_B2.transform.rotation.y = q2.y();
        tf_B2.transform.rotation.z = q2.z();
80      tf_B2.transform.rotation.w = q2.w();

82      tf_B3.header.frame_id = "base_origin";
        tf_B3.child_frame_id = "base_link3";
```

```
84      tf_B3.transform.translation.x = BASE_RADIUS * cos((4 * PI / 3) + (
        BASE_ANGLE));
        tf_B3.transform.translation.y = BASE_RADIUS * sin((4 * PI / 3) + (
        BASE_ANGLE));
86      tf_B3.transform.translation.z = BASE_HEIGHT;
        tf2::Quaternion q3;
88      q3.setRPY(0, 0, 0);
        tf_B3.transform.rotation.x = q3.x();
90      tf_B3.transform.rotation.y = q3.y();
        tf_B3.transform.rotation.z = q3.z();
92      tf_B3.transform.rotation.w = q3.w();

94      tf_B4.header.frame_id = "base_origin";
        tf_B4.child_frame_id = "base_link4";
96      tf_B4.transform.translation.x = BASE_RADIUS * cos((4 * PI / 3) - (
        BASE_ANGLE));
        tf_B4.transform.translation.y = BASE_RADIUS * sin((4 * PI / 3) - (
        BASE_ANGLE));
98      tf_B4.transform.translation.z = BASE_HEIGHT;
        tf2::Quaternion q4;
100     q4.setRPY(0, 0, 0);
        tf_B4.transform.rotation.x = q4.x();
102     tf_B4.transform.rotation.y = q4.y();
        tf_B4.transform.rotation.z = q4.z();
104     tf_B4.transform.rotation.w = q4.w();

106     tf_B5.header.frame_id = "base_origin";
        tf_B5.child_frame_id = "base_link5";
108     tf_B5.transform.translation.x = BASE_RADIUS * cos((2 * PI / 3) + (
        BASE_ANGLE));
        tf_B5.transform.translation.y = BASE_RADIUS * sin((2 * PI / 3) + (
        BASE_ANGLE));
110     tf_B5.transform.translation.z = BASE_HEIGHT;
        tf2::Quaternion q5;
112     q5.setRPY(0, 0, 0);
        tf_B5.transform.rotation.x = q5.x();
114     tf_B5.transform.rotation.y = q5.y();
        tf_B5.transform.rotation.z = q5.z();
116     tf_B5.transform.rotation.w = q5.w();

118     tf_B6.header.frame_id = "base_origin";
        tf_B6.child_frame_id = "base_link6";
120     tf_B6.transform.translation.x = BASE_RADIUS * cos((2 * PI / 3) - (
        BASE_ANGLE));
```

```
    tf_B6.transform.translation.y = BASE_RADIUS * sin((2 * PI / 3) - (
    BASE_ANGLE));
122     tf_B6.transform.translation.z = BASE_HEIGHT;
    tf2::Quaternion q6;
124     q6.setRPY(0, 0, 0);
    tf_B6.transform.rotation.x = q6.x();
126     tf_B6.transform.rotation.y = q6.y();
    tf_B6.transform.rotation.z = q6.z();
128     tf_B6.transform.rotation.w = q6.w();

130     ros::Rate rate(50.0);
    printf("sending Base TF\n");
132
    while (node.ok())
134     {
        tf_B1.header.stamp = ros::Time::now();
136         tf_B2.header.stamp = ros::Time::now();
        tf_B3.header.stamp = ros::Time::now();
138         tf_B4.header.stamp = ros::Time::now();
        tf_B5.header.stamp = ros::Time::now();
140         tf_B6.header.stamp = ros::Time::now();

142         tfb1.sendTransform(tf_B1);
        tfb2.sendTransform(tf_B2);
144         tfb3.sendTransform(tf_B3);
        tfb4.sendTransform(tf_B4);
146         tfb5.sendTransform(tf_B5);
        tfb6.sendTransform(tf_B6);
148
        rate.sleep();
150     }
};
```

Listing A.14: Quellcode zur Erstellung des TF-Tree der Gough-Stewart Base

A.5.2 stewart_platform_node.cpp

```
#include <ros/ros.h>
#include <tf2_ros/transform_broadcaster.h>
#include <tf2/LinearMath/Quaternion.h>
#include "tf2_ros/transform_listener.h"
#include "tf2_ros/message_filter.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"
#include <geometry_msgs/TransformStamped.h>
#include "geometry_msgs/PointStamped.h"

#define BASE_RADIUS 0.408
#define BASE_HEIGHT 0.105
#define BASE_ANGLE 0.122173 // 7Â°xpi/180
#define PLATFORM_RADIUS 0.21
#define PLATFORM_HEIGHT 0.106
#define PLATFORM_ANGLE 0.733038 // 42Â°xpi/180
#define PI 3.14159265359

int main(int argc, char **argv)
{
    ros::init(argc, argv, "platform_tf2_broadcaster");
    ros::NodeHandle n;

    tf2_ros::TransformBroadcaster tfb1;
    tf2_ros::TransformBroadcaster tfb2;
    tf2_ros::TransformBroadcaster tfb3;
    tf2_ros::TransformBroadcaster tfb4;
    tf2_ros::TransformBroadcaster tfb5;
    tf2_ros::TransformBroadcaster tfb6;

    geometry_msgs::TransformStamped tf_P1;
    geometry_msgs::TransformStamped tf_P2;
    geometry_msgs::TransformStamped tf_P3;
    geometry_msgs::TransformStamped tf_P4;
    geometry_msgs::TransformStamped tf_P5;
    geometry_msgs::TransformStamped tf_P6;

    tf_P1.header.frame_id = "platform_origin";
    tf_P1.child_frame_id = "platform_link1";
    tf_P1.transform.translation.x = PLATFORM_RADIUS * cos((0) + (
    PLATFORM_ANGLE));
    tf_P1.transform.translation.y = PLATFORM_RADIUS * sin((0) + (
    PLATFORM_ANGLE));
```

```
    tf_P1.transform.translation.z = -PLATFORM_HEIGHT;
42  tf2::Quaternion q1;
    q1.setRPY(0, 0, 0);
44  tf_P1.transform.rotation.x = q1.x();
    tf_P1.transform.rotation.y = q1.y();
46  tf_P1.transform.rotation.z = q1.z();
    tf_P1.transform.rotation.w = q1.w();
48
    tf_P2.header.frame_id = "platform_origin";
50  tf_P2.child_frame_id = "platform_link2";
    tf_P2.transform.translation.x = PLATFORM_RADIUS * cos((0) - (
    PLATFORM_ANGLE));
52  tf_P2.transform.translation.y = PLATFORM_RADIUS * sin((0) - (
    PLATFORM_ANGLE));
    tf_P2.transform.translation.z = -PLATFORM_HEIGHT;
54  tf2::Quaternion q2;
    q2.setRPY(0, 0, 0);
56  tf_P2.transform.rotation.x = q2.x();
    tf_P2.transform.rotation.y = q2.y();
58  tf_P2.transform.rotation.z = q2.z();
    tf_P2.transform.rotation.w = q2.w();
60
    tf_P3.header.frame_id = "platform_origin";
62  tf_P3.child_frame_id = "platform_link3";
    tf_P3.transform.translation.x = PLATFORM_RADIUS * cos((4 * PI / 3) + (
    PLATFORM_ANGLE));
64  tf_P3.transform.translation.y = PLATFORM_RADIUS * sin((4 * PI / 3) + (
    PLATFORM_ANGLE));
    tf_P3.transform.translation.z = -PLATFORM_HEIGHT;
66  tf2::Quaternion q3;
    q3.setRPY(0, 0, 0);
68  tf_P3.transform.rotation.x = q3.x();
    tf_P3.transform.rotation.y = q3.y();
70  tf_P3.transform.rotation.z = q3.z();
    tf_P3.transform.rotation.w = q3.w();
72
    tf_P4.header.frame_id = "platform_origin";
74  tf_P4.child_frame_id = "platform_link4";
    tf_P4.transform.translation.x = PLATFORM_RADIUS * cos((4 * PI / 3) - (
    PLATFORM_ANGLE));
76  tf_P4.transform.translation.y = PLATFORM_RADIUS * sin((4 * PI / 3) - (
    PLATFORM_ANGLE));
    tf_P4.transform.translation.z = -PLATFORM_HEIGHT;
78  tf2::Quaternion q4;
```

```
    q4.setRPY(0, 0, 0);
80  tf_P4.transform.rotation.x = q4.x();
    tf_P4.transform.rotation.y = q4.y();
82  tf_P4.transform.rotation.z = q4.z();
    tf_P4.transform.rotation.w = q4.w();
84
    tf_P5.header.frame_id = "platform_origin";
86  tf_P5.child_frame_id = "platform_link5";
    tf_P5.transform.translation.x = PLATFORM_RADIUS * cos((2 * PI / 3) + (
    PLATFORM_ANGLE));
88  tf_P5.transform.translation.y = PLATFORM_RADIUS * sin((2 * PI / 3) + (
    PLATFORM_ANGLE));
    tf_P5.transform.translation.z = -PLATFORM_HEIGHT;
90  tf2::Quaternion q5;
    q5.setRPY(0, 0, 0);
92  tf_P5.transform.rotation.x = q5.x();
    tf_P5.transform.rotation.y = q5.y();
94  tf_P5.transform.rotation.z = q5.z();
    tf_P5.transform.rotation.w = q5.w();
96
    tf_P6.header.frame_id = "platform_origin";
98  tf_P6.child_frame_id = "platform_link6";
    tf_P6.transform.translation.x = PLATFORM_RADIUS * cos((2 * PI / 3) - (
    PLATFORM_ANGLE));
100 tf_P6.transform.translation.y = PLATFORM_RADIUS * sin((2 * PI / 3) - (
    PLATFORM_ANGLE));
    tf_P6.transform.translation.z = -PLATFORM_HEIGHT;
102 tf2::Quaternion q6;
    q6.setRPY(0, 0, 0);
104 tf_P6.transform.rotation.x = q6.x();
    tf_P6.transform.rotation.y = q6.y();
106 tf_P6.transform.rotation.z = q6.z();
    tf_P6.transform.rotation.w = q6.w();
108
    ros::Rate rate(50.0);
110 printf("sending Platform TF\n");
    while (n.ok())
112 {
        tf_P1.header.stamp = ros::Time::now();
114     tf_P2.header.stamp = ros::Time::now();
        tf_P3.header.stamp = ros::Time::now();
116     tf_P4.header.stamp = ros::Time::now();
        tf_P5.header.stamp = ros::Time::now();
118     tf_P6.header.stamp = ros::Time::now();
```

```
120         tfb1.sendTransform(tf_P1);
            tfb2.sendTransform(tf_P2);
122         tfb3.sendTransform(tf_P3);
            tfb4.sendTransform(tf_P4);
124         tfb5.sendTransform(tf_P5);
            tfb6.sendTransform(tf_P6);
126
            rate.sleep();
128     }
};
```

Listing A.15: Quellcode zur Erstellung des TF-Tree der Stewart-Plattform

A.5.3 stewart_imu_node.cpp

```
#include <ros/ros.h>
#include <tf2_ros/transform_broadcaster.h>
#include <tf2/LinearMath/Quaternion.h>
#include <sensor_msgs/Imu.h>

#include <geometry_msgs/TransformStamped.h>
#include "geometry_msgs/PointStamped.h"

#include "tf2_ros/transform_listener.h"
#include "tf2_ros/message_filter.h"
#include "message_filters/subscriber.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"

#define BASE_RADIUS 0.408
#define BASE_HEIGHT 0.105
#define BASE_ANGLE 0.122173     // 7Â°*pi/180
#define PLATFORM_RADIUS 0.21
#define PLATFORM_HEIGHT 0.106
#define PLATFORM_ANGLE 0.733038 // 42Â°*pi/180
#define PI 3.14159265359
#define ROLL_LIMIT 0.0523599    // 5Â°
#define PITCH_LIMIT 0.0523599   // 5Â°
#define YAW_LIMIT 0.0872665     // 10Â°

tf2::Quaternion q_world;
tf2::Quaternion iq_in;
geometry_msgs::TransformStamped tf_world;
geometry_msgs::TransformStamped tf_P0;
bool start_up = false;

void imuCallback(const sensor_msgs::Imu::ConstPtr &msg)
{
    //ROS_INFO("Imu Seq: [%d]", msg->header.seq);
    //ROS_INFO("Imu Orientation x: [%f], y: [%f], z: [%f], w: [%f]", msg->
    orientation.x, msg->orientation.y, msg->orientation.z, msg->orientation.w
    );

    static tf2_ros::TransformBroadcaster tfb;
    geometry_msgs::TransformStamped tf_IMU;

    static tf2_ros::TransformBroadcaster tfb_B0;
    geometry_msgs::TransformStamped tf_B0;
```

```
42      tf2::Quaternion q0, q_in, q_out;
        double roll, pitch, yaw;
44
        tf_IMU.header.stamp = ros::Time::now();
46      tf_IMU.header.frame_id = "base_footprint";
        tf_IMU.child_frame_id = "imu_bno055";
48      tf_IMU.transform.translation.x = 0.0;
        tf_IMU.transform.translation.y = 0.0;
50      tf_IMU.transform.translation.z = 0.0;
        q_in[0] = msg->orientation.x;
52      q_in[1] = msg->orientation.y;
        q_in[2] = msg->orientation.z;
54      q_in[3] = msg->orientation.w;
        q_out = iq_in * q_in;
56      tf2::Matrix3x3 m(q_out);
        m.getRPY(roll, pitch, yaw);
58      if(roll > ROLL_LIMIT)        roll  = ROLL_LIMIT;
        if(roll < -ROLL_LIMIT)       roll  = -ROLL_LIMIT;
60      if(pitch > PITCH_LIMIT)      pitch = PITCH_LIMIT;
        if(pitch < -PITCH_LIMIT)     pitch = -PITCH_LIMIT;
62      if(yaw > YAW_LIMIT)          yaw   = YAW_LIMIT;
        if(yaw < -YAW_LIMIT)         yaw   = -YAW_LIMIT;
64      q_out.setRPY(roll, pitch, yaw);
        //ROS_INFO("Roll:%f, Pitch:%f, Yaw:%f", roll, pitch, yaw);
66      q_out.normalize();
        tf_IMU.transform.rotation.x = q_out.x();
68      tf_IMU.transform.rotation.y = q_out.y();
        tf_IMU.transform.rotation.z = q_out.z();
70      tf_IMU.transform.rotation.w = q_out.w();
        tfb.sendTransform(tf_IMU);
72
        tf_B0.header.stamp = ros::Time::now();
74      tf_B0.header.frame_id = "imu_bno055";
        tf_B0.child_frame_id = "base_origin";
76      tf_B0.transform.translation.x = 0.0;
        tf_B0.transform.translation.y = 0.0;
78      tf_B0.transform.translation.z = 0.0;
        q0.setRPY(0, 0, 0);
80      tf_B0.transform.rotation.x = q0.x();
        tf_B0.transform.rotation.y = q0.y();
82      tf_B0.transform.rotation.z = q0.z();
        tf_B0.transform.rotation.w = q0.w();
84
```

```
    tfb_B0.sendTransform(tf_B0);
86
    if (!start_up)
88  {
        start_up = true;
90      q_world = q_in;
        iq_in[0] = q_in[0];
92      iq_in[1] = q_in[1];
        iq_in[2] = q_in[2];
94      iq_in[3] = -q_in[3];

96      //q_world.setRPY(0, 0, 0);
        tf_world.header.frame_id = "world";
98      tf_world.child_frame_id = "base_footprint";
        tf_world.transform.translation.x = 0.0;
100     tf_world.transform.translation.y = 0.0;
        tf_world.transform.translation.z = 0.0;
102     tf_world.transform.rotation.x = q_world.x();
        tf_world.transform.rotation.y = q_world.y();
104     tf_world.transform.rotation.z = q_world.z();
        tf_world.transform.rotation.w = q_world.w();
106
        tf_P0.header.frame_id = "world";
108     tf_P0.child_frame_id = "platform_origin";
        tf_P0.transform.translation.x = 0.0;
110     tf_P0.transform.translation.y = 0.0;
        tf_P0.transform.translation.z = 1.2;
112     //tf2::Quaternion q_rot;
        //q_rot.setRPY(PI, 0, 0);
114     //q_world2 = q_world;
        //q_world2.normalize();
116     tf_P0.transform.rotation.x = q_world.x();
        tf_P0.transform.rotation.y = q_world.y();
118     tf_P0.transform.rotation.z = q_world.z();
        tf_P0.transform.rotation.w = q_world.w();
120 }
}
122
int main(int argc, char **argv)
124 {
    ros::init(argc, argv, "imu_tf2_broadcaster");
126 ros::NodeHandle n;

128 ros::Subscriber sub = n.subscribe("/imu", 1, &imuCallback);
```

```
130     static tf2_ros::TransformBroadcaster tfb_world;
        static tf2_ros::TransformBroadcaster tfb0;
132
        ros::Rate rate(10.0);
134     printf("sending IMU TF\n");

136     while (n.ok())
        {
138         tf_world.header.stamp = ros::Time::now();
            tf_P0.header.stamp = ros::Time::now();
140         tfb_world.sendTransform(tf_world);
            tfb0.sendTransform(tf_P0);
142
            ros::spinOnce();
144         rate.sleep();
        }
146 };
```

Listing A.16: Quellcode zur Transformation der IMU Bewegung auf die Stewart-Plattform

A.5.4 stewart_joint_node.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "control_msgs/JointControllerState.h"
#include <geometry_msgs/PointStamped.h>
#include <tf/transform_listener.h>

// PID Controller properties
#define SETPOINT 0
#define P 225
#define I 100
#define D 100
#define I_CLAMP 15
#define ANTIWINDUP true
#define IDLE_POSITION 100

double leg1_length_ = -1;
double leg2_length_ = -1;
double leg3_length_ = -1;
double leg4_length_ = -1;
double leg5_length_ = -1;
double leg6_length_ = -1;

class SubscribeAndPublish
{
public:
  ros::NodeHandle n_;
  control_msgs::JointControllerState m1_state_msg;
  control_msgs::JointControllerState m2_state_msg;
  control_msgs::JointControllerState m3_state_msg;
  control_msgs::JointControllerState m4_state_msg;
  control_msgs::JointControllerState m5_state_msg;
  control_msgs::JointControllerState m6_state_msg;
  bool init_position = false;
  bool idle_position = false;
  uint8_t loop_break = 0;

  SubscribeAndPublish()
  {
    //Topic you want to publish
    m1_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M1/
    P_Controller_Setting", 1);
```

```
    m1_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M1/
    V_Controller_Setting", 1);
42  m2_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M2/
    P_Controller_Setting", 1);
    m2_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M2/
    V_Controller_Setting", 1);
44  m3_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M3/
    P_Controller_Setting", 1);
    m3_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M3/
    V_Controller_Setting", 1);
46  m4_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M4/
    P_Controller_Setting", 1);
    m4_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M4/
    V_Controller_Setting", 1);
48  m5_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M5/
    P_Controller_Setting", 1);
    m5_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M5/
    V_Controller_Setting", 1);
50  m6_pos_pub_ = n_.advertise<control_msgs::JointControllerState>("/M6/
    P_Controller_Setting", 1);
    m6_vel_pub_ = n_.advertise<control_msgs::JointControllerState>("/M6/
    V_Controller_Setting", 1);
52
    //Topic you want to subscribe
54  m1_pos_sub_ = n_.subscribe("/M1/P_Controller_Setting", 1, &
    SubscribeAndPublish::M1_pos_Callback, this);
    m1_vel_sub_ = n_.subscribe("/M1/V_Controller_Setting", 1, &
    SubscribeAndPublish::M1_vel_Callback, this);
56  m2_pos_sub_ = n_.subscribe("/M2/P_Controller_Setting", 1, &
    SubscribeAndPublish::M2_pos_Callback, this);
    m2_vel_sub_ = n_.subscribe("/M2/V_Controller_Setting", 1, &
    SubscribeAndPublish::M2_vel_Callback, this);
58  m3_pos_sub_ = n_.subscribe("/M3/P_Controller_Setting", 1, &
    SubscribeAndPublish::M3_pos_Callback, this);
    m3_vel_sub_ = n_.subscribe("/M3/V_Controller_Setting", 1, &
    SubscribeAndPublish::M3_vel_Callback, this);
60  m4_pos_sub_ = n_.subscribe("/M4/P_Controller_Setting", 1, &
    SubscribeAndPublish::M4_pos_Callback, this);
    m4_vel_sub_ = n_.subscribe("/M4/V_Controller_Setting", 1, &
    SubscribeAndPublish::M4_vel_Callback, this);
62  m5_pos_sub_ = n_.subscribe("/M5/P_Controller_Setting", 1, &
    SubscribeAndPublish::M5_pos_Callback, this);
    m5_vel_sub_ = n_.subscribe("/M5/V_Controller_Setting", 1, &
    SubscribeAndPublish::M5_vel_Callback, this);
```

```
64    m6_pos_sub_ = n_.subscribe("/M6/P_Controller_Setting", 1, &
      SubscribeAndPublish::M6_pos_Callback, this);
      m6_vel_sub_ = n_.subscribe("/M6/V_Controller_Setting", 1, &
      SubscribeAndPublish::M6_vel_Callback, this);
66
      //Topic you want to subscribe
68    m1_state_sub_ = n_.subscribe("/M1/P_Controller_State", 1, &
      SubscribeAndPublish::M1_state_Callback, this);
      m2_state_sub_ = n_.subscribe("/M2/P_Controller_State", 1, &
      SubscribeAndPublish::M2_state_Callback, this);
70    m3_state_sub_ = n_.subscribe("/M3/P_Controller_State", 1, &
      SubscribeAndPublish::M3_state_Callback, this);
      m4_state_sub_ = n_.subscribe("/M4/P_Controller_State", 1, &
      SubscribeAndPublish::M4_state_Callback, this);
72    m5_state_sub_ = n_.subscribe("/M5/P_Controller_State", 1, &
      SubscribeAndPublish::M5_state_Callback, this);
      m6_state_sub_ = n_.subscribe("/M6/P_Controller_State", 1, &
      SubscribeAndPublish::M6_state_Callback, this);
74  }

76  void M1_state_Callback(const control_msgs::JointControllerState::ConstPtr &
      msg)
    {
78    m1_state_msg.header.frame_id = msg->header.frame_id;
      m1_state_msg.header.stamp = msg->header.stamp;
80    m1_state_msg.set_point = msg->set_point;
      m1_state_msg.process_value = msg->process_value;
82    m1_state_msg.process_value_dot = msg->process_value_dot;
      m1_state_msg.p = msg->p;
84    m1_state_msg.i = msg->i;
      m1_state_msg.d = msg->d;
86    m1_state_msg.i_clamp = msg->i_clamp;
      m1_state_msg.antiwindup = msg->antiwindup;
88  }

90  void M2_state_Callback(const control_msgs::JointControllerState::ConstPtr &
      msg)
    {
92    m2_state_msg.header.frame_id = msg->header.frame_id;
      m2_state_msg.header.stamp = msg->header.stamp;
94    m2_state_msg.set_point = msg->set_point;
      m2_state_msg.process_value = msg->process_value;
96    m2_state_msg.process_value_dot = msg->process_value_dot;
      m2_state_msg.p = msg->p;
```

```
98      m2_state_msg.i = msg->i;
        m2_state_msg.d = msg->d;
100     m2_state_msg.i_clamp = msg->i_clamp;
        m2_state_msg.antiwindup = msg->antiwindup;
102   }

104   void M3_state_Callback(const control_msgs::JointControllerState::ConstPtr &
        msg)
      {
106     m3_state_msg.header.frame_id = msg->header.frame_id;
        m3_state_msg.header.stamp = msg->header.stamp;
108     m3_state_msg.set_point = msg->set_point;
        m3_state_msg.process_value = msg->process_value;
110     m3_state_msg.process_value_dot = msg->process_value_dot;
        m3_state_msg.p = msg->p;
112     m3_state_msg.i = msg->i;
        m3_state_msg.d = msg->d;
114     m3_state_msg.i_clamp = msg->i_clamp;
        m3_state_msg.antiwindup = msg->antiwindup;
116   }

118   void M4_state_Callback(const control_msgs::JointControllerState::ConstPtr &
        msg)
      {
120     m4_state_msg.header.frame_id = msg->header.frame_id;
        m4_state_msg.header.stamp = msg->header.stamp;
122     m4_state_msg.set_point = msg->set_point;
        m4_state_msg.process_value = msg->process_value;
124     m4_state_msg.process_value_dot = msg->process_value_dot;
        m4_state_msg.p = msg->p;
126     m4_state_msg.i = msg->i;
        m4_state_msg.d = msg->d;
128     m4_state_msg.i_clamp = msg->i_clamp;
        m4_state_msg.antiwindup = msg->antiwindup;
130   }

132   void M5_state_Callback(const control_msgs::JointControllerState::ConstPtr &
        msg)
      {
134     m5_state_msg.header.frame_id = msg->header.frame_id;
        m5_state_msg.header.stamp = msg->header.stamp;
136     m5_state_msg.set_point = msg->set_point;
        m5_state_msg.process_value = msg->process_value;
138     m5_state_msg.process_value_dot = msg->process_value_dot;
```

```
    m5_state_msg.p = msg->p;
    m5_state_msg.i = msg->i;
    m5_state_msg.d = msg->d;
    m5_state_msg.i_clamp = msg->i_clamp;
    m5_state_msg.antiwindup = msg->antiwindup;
  }

  void M6_state_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    m6_state_msg.header.frame_id = msg->header.frame_id;
    m6_state_msg.header.stamp = msg->header.stamp;
    m6_state_msg.set_point = msg->set_point;
    m6_state_msg.process_value = msg->process_value;
    m6_state_msg.process_value_dot = msg->process_value_dot;
    m6_state_msg.p = msg->p;
    m6_state_msg.i = msg->i;
    m6_state_msg.d = msg->d;
    m6_state_msg.i_clamp = msg->i_clamp;
    m6_state_msg.antiwindup = msg->antiwindup;
  }

  void M1_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    control_msgs::JointControllerState m1_pos_msg;
    if (init_position && idle_position)
    {
      m1_pos_msg.header.frame_id = "/M1_pos_state";
      if (leg1_length_ >= 0)
      {
        m1_pos_msg.set_point = (leg1_length_ - 0.88) / 0.00127;
      }
      else
      {
        m1_pos_msg.set_point = msg->set_point;
      }
      m1_pos_msg.p = msg->p;
      m1_pos_msg.i = msg->i;
      m1_pos_msg.d = msg->d;
      m1_pos_msg.i_clamp = msg->i_clamp;
      m1_pos_msg.antiwindup = msg->antiwindup;
      m1_pos_msg.header.stamp = ros::Time::now();
      m1_pos_pub_.publish(m1_pos_msg);
```

```
    }
182 }
  void M1_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
184 {
    control_msgs::JointControllerState m1_vel_msg;
186   if (init_position && idle_position)
    {
188   m1_vel_msg.header.frame_id = "/M1_vel_state";
      m1_vel_msg.set_point = msg->set_point;
190   m1_vel_msg.p = msg->p;
      m1_vel_msg.i = msg->i;
192   m1_vel_msg.d = msg->d;
      m1_vel_msg.i_clamp = msg->i_clamp;
194   m1_vel_msg.antiwindup = msg->antiwindup;
      m1_vel_msg.header.stamp = ros::Time::now();
196   m1_vel_pub_.publish(m1_vel_msg);
    }
198 }
  void M2_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
200 {
    control_msgs::JointControllerState m2_pos_msg;
202   if (init_position && idle_position)
    {
204   m2_pos_msg.header.frame_id = "/M2_pos_state";
      if (leg2_length_ >= 0)
206   {
        m2_pos_msg.set_point = (leg2_length_ - 0.88) / 0.00127;
208   }
      else
210   {
        m2_pos_msg.set_point = msg->set_point;
212   }
      m2_pos_msg.p = msg->p;
214   m2_pos_msg.i = msg->i;
      m2_pos_msg.d = msg->d;
216   m2_pos_msg.i_clamp = msg->i_clamp;
      m2_pos_msg.antiwindup = msg->antiwindup;
218   m2_pos_msg.header.stamp = ros::Time::now();
      m2_pos_pub_.publish(m2_pos_msg);
220   }
  }
```

```
222  void M2_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
       msg)
     {
224    control_msgs::JointControllerState m2_vel_msg;
       if (init_position && idle_position)
226    {
         m2_vel_msg.header.frame_id = "/M2_vel_state";
228      m2_vel_msg.set_point = msg->set_point;
         m2_vel_msg.p = msg->p;
230      m2_vel_msg.i = msg->i;
         m2_vel_msg.d = msg->d;
232      m2_vel_msg.i_clamp = msg->i_clamp;
         m2_vel_msg.antiwindup = msg->antiwindup;
234      m2_vel_msg.header.stamp = ros::Time::now();
         m2_vel_pub_.publish(m2_vel_msg);
236    }
     }
238  void M3_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
       msg)
     {
240    control_msgs::JointControllerState m3_pos_msg;
       if (init_position && idle_position)
242    {
         m3_pos_msg.header.frame_id = "/M3_pos_state";
244      if (leg3_length_ >= 0)
         {
246        m3_pos_msg.set_point = (leg3_length_ - 0.88) / 0.00127;
         }
248      else
         {
250        m3_pos_msg.set_point = msg->set_point;
         }
252      m3_pos_msg.p = msg->p;
         m3_pos_msg.i = msg->i;
254      m3_pos_msg.d = msg->d;
         m3_pos_msg.i_clamp = msg->i_clamp;
256      m3_pos_msg.antiwindup = msg->antiwindup;
         m3_pos_msg.header.stamp = ros::Time::now();
258      m3_pos_pub_.publish(m3_pos_msg);
       }
260  }
     void M3_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
       msg)
262  {
```

```
    control_msgs::JointControllerState m3_vel_msg;
    if (init_position && idle_position)
    {
      m3_vel_msg.header.frame_id = "/M3_vel_state";
      m3_vel_msg.set_point = msg->set_point;
      m3_vel_msg.p = msg->p;
      m3_vel_msg.i = msg->i;
      m3_vel_msg.d = msg->d;
      m3_vel_msg.i_clamp = msg->i_clamp;
      m3_vel_msg.antiwindup = msg->antiwindup;
      m3_vel_msg.header.stamp = ros::Time::now();
      m3_vel_pub_.publish(m3_vel_msg);
    }
  }
  void M4_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    control_msgs::JointControllerState m4_pos_msg;
    if (init_position && idle_position)
    {
      m4_pos_msg.header.frame_id = "/M4_pos_state";
      if (leg4_length_ >= 0)
      {
        m4_pos_msg.set_point = (leg4_length_ - 0.88) / 0.00127;
      }
      else
      {
        m4_pos_msg.set_point = msg->set_point;
      }
      m4_pos_msg.p = msg->p;
      m4_pos_msg.i = msg->i;
      m4_pos_msg.d = msg->d;
      m4_pos_msg.i_clamp = msg->i_clamp;
      m4_pos_msg.antiwindup = msg->antiwindup;
      m4_pos_msg.header.stamp = ros::Time::now();
      m4_pos_pub_.publish(m4_pos_msg);
    }
  }
  void M4_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    control_msgs::JointControllerState m4_vel_msg;
    if (init_position && idle_position)
    {
```

```
      m4_vel_msg.header.frame_id = "/M4_vel_state";
306   m4_vel_msg.set_point = msg->set_point;
      m4_vel_msg.p = msg->p;
308   m4_vel_msg.i = msg->i;
      m4_vel_msg.d = msg->d;
310   m4_vel_msg.i_clamp = msg->i_clamp;
      m4_vel_msg.antiwindup = msg->antiwindup;
312   m4_vel_msg.header.stamp = ros::Time::now();
      m4_vel_pub_.publish(m4_vel_msg);
314   }
    }
316 void M5_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
      msg)
    {
318   control_msgs::JointControllerState m5_pos_msg;
      if (init_position && idle_position)
320   {
        m5_pos_msg.header.frame_id = "/M5_pos_state";
322     if (leg5_length_ >= 0)
        {
324       m5_pos_msg.set_point = (leg5_length_ - 0.88) / 0.00127;
        }
326     else
        {
328       m5_pos_msg.set_point = msg->set_point;
        }
330     m5_pos_msg.p = msg->p;
        m5_pos_msg.i = msg->i;
332     m5_pos_msg.d = msg->d;
        m5_pos_msg.i_clamp = msg->i_clamp;
334     m5_pos_msg.antiwindup = msg->antiwindup;
        m5_pos_msg.header.stamp = ros::Time::now();
336     m5_pos_pub_.publish(m5_pos_msg);
      }
338 }
    void M5_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
      msg)
340 {
      control_msgs::JointControllerState m5_vel_msg;
342   if (init_position && idle_position)
      {
344     m5_vel_msg.header.frame_id = "/M5_vel_state";
        m5_vel_msg.set_point = msg->set_point;
346     m5_vel_msg.p = msg->p;
```

```
      m5_vel_msg.i = msg->i;
      m5_vel_msg.d = msg->d;
      m5_vel_msg.i_clamp = msg->i_clamp;
      m5_vel_msg.antiwindup = msg->antiwindup;
      m5_vel_msg.header.stamp = ros::Time::now();
      m5_vel_pub_.publish(m5_vel_msg);
    }
  }
  void M6_pos_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    control_msgs::JointControllerState m6_pos_msg;
    if (init_position && idle_position)
    {
      m6_pos_msg.header.frame_id = "/M6_pos_state";
      if (leg6_length_ >= 0)
      {
        m6_pos_msg.set_point = (leg6_length_ - 0.88) / 0.00127;
      }
      else
      {
        m6_pos_msg.set_point = msg->set_point;
      }
      m6_pos_msg.p = msg->p;
      m6_pos_msg.i = msg->i;
      m6_pos_msg.d = msg->d;
      m6_pos_msg.i_clamp = msg->i_clamp;
      m6_pos_msg.antiwindup = msg->antiwindup;
      m6_pos_msg.header.stamp = ros::Time::now();
      m6_pos_pub_.publish(m6_pos_msg);
    }
  }
  void M6_vel_Callback(const control_msgs::JointControllerState::ConstPtr &
    msg)
  {
    control_msgs::JointControllerState m6_vel_msg;
    if (init_position && idle_position)
    {
      m6_vel_msg.header.frame_id = "/M6_vel_state";
      m6_vel_msg.set_point = msg->set_point;
      m6_vel_msg.p = msg->p;
      m6_vel_msg.i = msg->i;
      m6_vel_msg.d = msg->d;
      m6_vel_msg.i_clamp = msg->i_clamp;
```

```
      m6_vel_msg.antiwindup = msg->antiwindup;
      m6_vel_msg.header.stamp = ros::Time::now();
      m6_vel_pub_.publish(m6_vel_msg);
    }
  }

  void startup_Call(void)
  {
    control_msgs::JointControllerState M_pos_start;
    control_msgs::JointControllerState M_vel_start;

    M_pos_start.set_point = SETPOINT;
    M_pos_start.header.stamp = ros::Time::now();
    M_pos_start.p = 1;
    M_pos_start.i = 0;
    M_pos_start.d = 0;
    M_pos_start.i_clamp = 0;
    M_pos_start.antiwindup = false;

    M_vel_start.set_point = SETPOINT;
    M_vel_start.header.stamp = ros::Time::now();
    M_vel_start.p = P;
    M_vel_start.i = I;
    M_vel_start.d = D;
    M_vel_start.i_clamp = I_CLAMP;
    M_vel_start.antiwindup = ANTIWINDUP;

    m1_pos_pub_.publish(M_pos_start);
    m1_vel_pub_.publish(M_vel_start);
    m2_pos_pub_.publish(M_pos_start);
    m2_vel_pub_.publish(M_vel_start);
    m3_pos_pub_.publish(M_pos_start);
    m3_vel_pub_.publish(M_vel_start);
    m4_pos_pub_.publish(M_pos_start);
    m4_vel_pub_.publish(M_vel_start);
    m5_pos_pub_.publish(M_pos_start);
    m5_vel_pub_.publish(M_vel_start);
    m6_pos_pub_.publish(M_pos_start);
    m6_vel_pub_.publish(M_vel_start);
    ros::spinOnce();
  }

  int init_Call(void)
  {
```

```
    control_msgs::JointControllerState M_pos_start;
434
    M_pos_start.set_point = -1500;
436 M_pos_start.header.stamp = ros::Time::now();
    M_pos_start.p = 1;
438 M_pos_start.i = 0;
    M_pos_start.d = 0;
440 M_pos_start.i_clamp = 0;
    M_pos_start.antiwindup = false;
442
    m1_pos_pub_.publish(M_pos_start);
444 m2_pos_pub_.publish(M_pos_start);
    m3_pos_pub_.publish(M_pos_start);
446 m4_pos_pub_.publish(M_pos_start);
    m5_pos_pub_.publish(M_pos_start);
448 m6_pos_pub_.publish(M_pos_start);

450 ros::spinOnce();

452 if (m1_state_msg.process_value <= 0 &&
      m2_state_msg.process_value <= 0 &&
454   m3_state_msg.process_value <= 0 &&
      m4_state_msg.process_value <= 0 &&
456   m5_state_msg.process_value <= 0 &&
      m6_state_msg.process_value <= 0)
458 {
      if (m1_state_msg.process_value_dot == 0 &&
460     m2_state_msg.process_value_dot == 0 &&
        m3_state_msg.process_value_dot == 0 &&
462     m4_state_msg.process_value_dot == 0 &&
        m5_state_msg.process_value_dot == 0 &&
464     m6_state_msg.process_value_dot == 0)
      {
466     init_position = true;
        return 0;
468   }
    }
470 else
    {
472   loop_break++;
      if (loop_break < 500)
474   {
        return 1;
476   }
```

```
      else
478   {
        loop_break = 0;
480     return 0;
      }
482   }
}
484

int idle_Call(void)
486 {
  control_msgs::JointControllerState M_pos_start;
488

  M_pos_start.set_point = -1000;
490 M_pos_start.header.stamp = ros::Time::now();
  M_pos_start.p = 1;
492 M_pos_start.i = 0;
  M_pos_start.d = 0;
494 M_pos_start.i_clamp = 0;
  M_pos_start.antiwindup = false;
496

  m1_pos_pub_.publish(M_pos_start);
498 m2_pos_pub_.publish(M_pos_start);
  m3_pos_pub_.publish(M_pos_start);
500 m4_pos_pub_.publish(M_pos_start);
  m5_pos_pub_.publish(M_pos_start);
502 m6_pos_pub_.publish(M_pos_start);

504 ros::spinOnce();

506 if ((m1_state_msg.process_value >= IDLE_POSITION-3 &&
      m2_state_msg.process_value >= IDLE_POSITION-3 &&
508   m3_state_msg.process_value >= IDLE_POSITION-3 &&
      m4_state_msg.process_value >= IDLE_POSITION-3 &&
510   m5_state_msg.process_value >= IDLE_POSITION-3 &&
      m6_state_msg.process_value >= IDLE_POSITION-3) &&
512   (m1_state_msg.process_value <= IDLE_POSITION+3 &&
      m2_state_msg.process_value <= IDLE_POSITION+3 &&
514   m3_state_msg.process_value <= IDLE_POSITION+3 &&
      m4_state_msg.process_value <= IDLE_POSITION+3 &&
516   m5_state_msg.process_value <= IDLE_POSITION+3 &&
      m6_state_msg.process_value <= IDLE_POSITION+3))
518 {
      idle_position = true;
520   return 0;
```

```
    }
522   else
    {
524     return 1;
    }
526 }

528 private:
  ros::Publisher m1_pos_pub_;
530   ros::Publisher m1_vel_pub_;
  ros::Publisher m2_pos_pub_;
532   ros::Publisher m2_vel_pub_;
  ros::Publisher m3_pos_pub_;
534   ros::Publisher m3_vel_pub_;
  ros::Publisher m4_pos_pub_;
536   ros::Publisher m4_vel_pub_;
  ros::Publisher m5_pos_pub_;
538   ros::Publisher m5_vel_pub_;
  ros::Publisher m6_pos_pub_;
540   ros::Publisher m6_vel_pub_;

542   ros::Subscriber m1_pos_sub_;
  ros::Subscriber m1_vel_sub_;
544   ros::Subscriber m2_pos_sub_;
  ros::Subscriber m2_vel_sub_;
546   ros::Subscriber m3_pos_sub_;
  ros::Subscriber m3_vel_sub_;
548   ros::Subscriber m4_pos_sub_;
  ros::Subscriber m4_vel_sub_;
550   ros::Subscriber m5_pos_sub_;
  ros::Subscriber m5_vel_sub_;
552   ros::Subscriber m6_pos_sub_;
  ros::Subscriber m6_vel_sub_;
554
  ros::Subscriber m1_state_sub_;
556   ros::Subscriber m2_state_sub_;
  ros::Subscriber m3_state_sub_;
558   ros::Subscriber m4_state_sub_;
  ros::Subscriber m5_state_sub_;
560   ros::Subscriber m6_state_sub_;
}; //End of class SubscribeAndPublish
562
void transformLeg(const tf::TransformListener &listener)
564 {
```

```
  //we'll create a point in the base_laser frame that we'd like to transform
    to the base_link frame
566 geometry_msgs::PointStamped P1;
  geometry_msgs::PointStamped P2;
568 geometry_msgs::PointStamped P3;
  geometry_msgs::PointStamped P4;
570 geometry_msgs::PointStamped P5;
  geometry_msgs::PointStamped P6;
572 P1.header.frame_id = "platform_link1";
  P2.header.frame_id = "platform_link2";
574 P3.header.frame_id = "platform_link3";
  P4.header.frame_id = "platform_link4";
576 P5.header.frame_id = "platform_link5";
  P6.header.frame_id = "platform_link6";
578
  //we'll just use the most recent transform available for our simple example
580 P1.header.stamp = ros::Time();
  P2.header.stamp = ros::Time();
582 P3.header.stamp = ros::Time();
  P4.header.stamp = ros::Time();
584 P5.header.stamp = ros::Time();
  P6.header.stamp = ros::Time();
586
  //just an arbitrary point in space
588 P1.point.x = 0.0;
  P1.point.y = 0.0;
590 P1.point.z = 0.0;
  P2.point.x = 0.0;
592 P2.point.y = 0.0;
  P2.point.z = 0.0;
594 P3.point.x = 0.0;
  P3.point.y = 0.0;
596 P3.point.z = 0.0;
  P4.point.x = 0.0;
598 P4.point.y = 0.0;
  P4.point.z = 0.0;
600 P5.point.x = 0.0;
  P5.point.y = 0.0;
602 P5.point.z = 0.0;
  P6.point.x = 0.0;
604 P6.point.y = 0.0;
  P6.point.z = 0.0;
606
  try
```

```
608   {
        geometry_msgs::PointStamped B1;
610     listener.transformPoint("base_link1", P1, B1);
        leg1_length_ = (sqrt(pow((P1.point.x - B1.point.x), 2) + pow((P1.point.y
        - B1.point.y), 2) + pow((P1.point.z - B1.point.z), 2)));
612   }
      catch (tf::TransformException &ex)
614   {
        ROS_ERROR("Received an exception trying to transform a point from \"
        platform_link1\" to \"base_link1\": %s", ex.what());
616   }
      try
618   {
        geometry_msgs::PointStamped B2;
620     listener.transformPoint("base_link2", P2, B2);
        leg2_length_ = (sqrt(pow((P2.point.x - B2.point.x), 2) + pow((P2.point.y
        - B2.point.y), 2) + pow((P2.point.z - B2.point.z), 2)));
622   }
      catch (tf::TransformException &ex)
624   {
        ROS_ERROR("Received an exception trying to transform a point from \"
        platform_link2\" to \"base_link2\": %s", ex.what());
626   }
      try
628   {
        geometry_msgs::PointStamped B3;
630     listener.transformPoint("base_link3", P3, B3);
        leg3_length_ = (sqrt(pow((P3.point.x - B3.point.x), 2) + pow((P3.point.y
        - B3.point.y), 2) + pow((P3.point.z - B3.point.z), 2)));
632   }
      catch (tf::TransformException &ex)
634   {
        ROS_ERROR("Received an exception trying to transform a point from \"
        platform_link3\" to \"base_link3\": %s", ex.what());
636   }
      try
638   {
        geometry_msgs::PointStamped B4;
640     listener.transformPoint("base_link4", P4, B4);
        leg4_length_ = (sqrt(pow((P4.point.x - B4.point.x), 2) + pow((P4.point.y
        - B4.point.y), 2) + pow((P4.point.z - B4.point.z), 2)));
642   }
      catch (tf::TransformException &ex)
644   {
```

```
    ROS_ERROR("Received an exception trying to transform a point from \"
    platform_link4\" to \"base_link4\": %s", ex.what());
  }
  try
  {
    geometry_msgs::PointStamped B5;
    listener.transformPoint("base_link5", P5, B5);
    leg5_length_ = (sqrt(pow((P5.point.x - B5.point.x), 2) + pow((P5.point.y
    - B5.point.y), 2) + pow((P5.point.z - B5.point.z), 2)));
  }
  catch (tf::TransformException &ex)
  {
    ROS_ERROR("Received an exception trying to transform a point from \"
    platform_link5\" to \"base_link5\": %s", ex.what());
  }
  try
  {
    geometry_msgs::PointStamped B6;
    listener.transformPoint("base_link6", P6, B6);
    leg6_length_ = (sqrt(pow((P6.point.x - B6.point.x), 2) + pow((P6.point.y
    - B6.point.y), 2) + pow((P6.point.z - B6.point.z), 2)));
  }
  catch (tf::TransformException &ex)
  {
    ROS_ERROR("Received an exception trying to transform a point from \"
    platform_link6\" to \"base_link6\": %s", ex.what());
  }
  //ROS_INFO("Leg 1 = %0.4f m, Leg 2 = %0.4f m, Leg 3 = %0.4f m, Leg 4 = %0.4
    f m, Leg 5 = %0.4f m, Leg 6 = %0.4f m, ", leg1_length_, leg2_length_,
    leg3_length_, leg4_length_, leg5_length_, leg6_length_);
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "JointControllerState");

  SubscribeAndPublish SAPObject;

  ros::Rate loop_rate(10);

  SAPObject.startup_Call();

  SAPObject.init_Call();
  ros::Duration(1).sleep();
```

```
682   while (SAPObject.init_Call() && ros::ok())
      {
684     loop_rate.sleep();
      }
686
      while (SAPObject.idle_Call() && ros::ok())
688   {
        loop_rate.sleep();
690   }

692   tf::TransformListener listener(ros::Duration(2));
      ros::Timer timer = SAPObject.n_.createTimer(ros::Duration(0.1), boost::bind
        (&transformLeg, boost::ref(listener)));
694
      while (ros::ok())
696   {

698     ros::spinOnce();

700     loop_rate.sleep();
      }
702
      return 0;
704 }
```

Listing A.17: Quellcode der Stewart-Node zum berechnen und senden der BeinlÃ□ngen

A.5.5 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(stewart)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  control_msgs
  diagnostic_msgs
  geometry_msgs
  roscpp
  rospy
  sensor_msgs
  std_msgs
  tf2
  tf2_bullet
  tf2_eigen
  tf2_geometry_msgs
  tf2_kdl
  tf2_msgs
  tf2_py
  tf2_ros
  tf2_sensor_msgs
  tf2_tools
  tf
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################
```

```
44 ## To declare and build messages, services or actions from within this
   ## package, follow these steps:
46 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
   ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
48 ## * In the file package.xml:
   ##   * add a build_depend tag for "message_generation"
50 ##   * add a build_depend and a exec_depend tag for each package in
       MSG_DEP_SET
   ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
52 ##     but can be declared for certainty nonetheless:
   ##     * add a exec_depend tag for "message_runtime"
54 ## * In this file (CMakeLists.txt):
   ##   * add "message_generation" and every package in MSG_DEP_SET to
56 ##     find_package(catkin REQUIRED COMPONENTS ...)
   ##   * add "message_runtime" and every package in MSG_DEP_SET to
58 ##     catkin_package(CATKIN_DEPENDS ...)
   ##   * uncomment the add_*_files sections below as needed
60 ##     and list every .msg/.srv/.action file to be processed
   ##   * uncomment the generate_messages entry below
62 ##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES
       ...)

64 ## Generate messages in the 'msg' folder
   # add_message_files(
66 #   FILES
   #   Message1.msg
68 #   Message2.msg
   # )
70
   ## Generate services in the 'srv' folder
72 # add_service_files(
   #   FILES
74 #   Service1.srv
   #   Service2.srv
76 # )

78 ## Generate actions in the 'action' folder
   # add_action_files(
80 #   FILES
   #   Action1.action
82 #   Action2.action
   # )
84
```

```
## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   control_msgs#   diagnostic_msgs#   geometry_msgs#   sensor_msgs#
    std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects
    also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also
    need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES stewart
#  CATKIN_DEPENDS control_msgs diagnostic_msgs geometry_msgs roscpp rospy
    sensor_msgs std_msgs
#  DEPENDS system_lib
```

```
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/stewart.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't
    collide
# add_executable(${PROJECT_NAME}_node src/stewart_node.cpp)

add_executable(base_tf2_broadcaster src/stewart_base_node.cpp)
add_executable(platform_tf2_broadcaster src/stewart_platform_node.cpp)
add_executable(stewart_joint_broadcaster src/stewart_joint_node.cpp)
add_executable(imu_tf2_broadcaster src/stewart_imu_node.cpp)
add_executable(stewart_leg_broadcaster src/stewart_leg_node.cpp)


## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following
    renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg
    someones_pkg_node"
```

```
164 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node
        PREFIX "")

166 ## Add cmake target dependencies of the executable
    ## same as for the library above
168 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} $
        {catkin_EXPORTED_TARGETS})

170 ## Specify libraries to link a library or executable target against
    target_link_libraries(base_tf2_broadcaster   ${catkin_LIBRARIES})
172 target_link_libraries(platform_tf2_broadcaster   ${catkin_LIBRARIES})
    target_link_libraries(stewart_joint_broadcaster   ${catkin_LIBRARIES})
174 target_link_libraries(imu_tf2_broadcaster   ${catkin_LIBRARIES})
    target_link_libraries(stewart_leg_broadcaster   ${catkin_LIBRARIES})
176
    # add_executable(${PROJECT_NAME} src/stewart_node.cpp)
178 # target_link_libraries(stewart ${catkin_LIBRARIES})


180
    #############
182 ## Install ##
    #############
184
    # all install targets should use catkin DESTINATION variables
186 # See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

188 ## Mark executable scripts (Python etc.) for installation
    ## in contrast to setup.py, you can choose the destination
190 # catkin_install_python(PROGRAMS
    #   scripts/my_python_script
192 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
    # )
194
    ## Mark executables for installation
196 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/
        building_executables.html
    # install(TARGETS ${PROJECT_NAME}_node
198 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
    # )
200
    ## Mark libraries for installation
202 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/
        building_libraries.html
    # install(TARGETS ${PROJECT_NAME}
```

```
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_stewart.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

Listing A.18: CMakeLists Datei zum Kompilieren des Gough-Stewart Package

A.5.6 stewart.launch

```
<launch>
    <!--node pkg="tf2_ros" type="static_transform_publisher" name="
    world_to_footprint_broadcaster" args="0 0 0 0 0 0 world base_footprint"
    /-->

    <include file="$(find rosserial_server)/launch/stewart.launch"/>

    <node pkg="stewart" type="base_tf2_broadcaster" name="
    base_tf2_broadcaster" output="screen"/>
    <node pkg="stewart" type="platform_tf2_broadcaster" name="
    platform_tf2_broadcaster" output="screen"/>
    <node pkg="stewart" type="imu_tf2_broadcaster" name="imu_tf2_broadcaster"
     output="screen"/>
    <node pkg="stewart" type="stewart_joint_broadcaster" name="
    stewart_joint_broadcaster" output="screen"/>

    <!--node pkg="tf2_ros" type="static_transform_publisher" name="
    imu_platform_broadcaster" args="0 0 0 0 0 0 imu_BNO055 platform_origin"
    /-->

    <node type="rviz" name="rviz" pkg="rviz" args="-d $(find stewart)/stewart
    .rviz" />
</launch>
```

Listing A.19: Launch-File zum starten Gough-Stewart Nodes in ROS

A.5.7 serial_stewart.launch

```
<launch>
  <node pkg="rosserial_server" type="socket_node" name="rosserial_server_1"
    args="_port:=11411"/>
  <node pkg="rosserial_server" type="socket_node" name="rosserial_server_2"
    args="_port:=11412"/>
  <node pkg="rosserial_server" type="socket_node" name="rosserial_server_3"
    args="_port:=11413"/>
  <node pkg="rosserial_server" type="socket_node" name="rosserial_server_4"
    args="_port:=11414"/>
</launch>
```

Listing A.20: Launch-File zum starten des ROS-Serial TCP Server in ROS

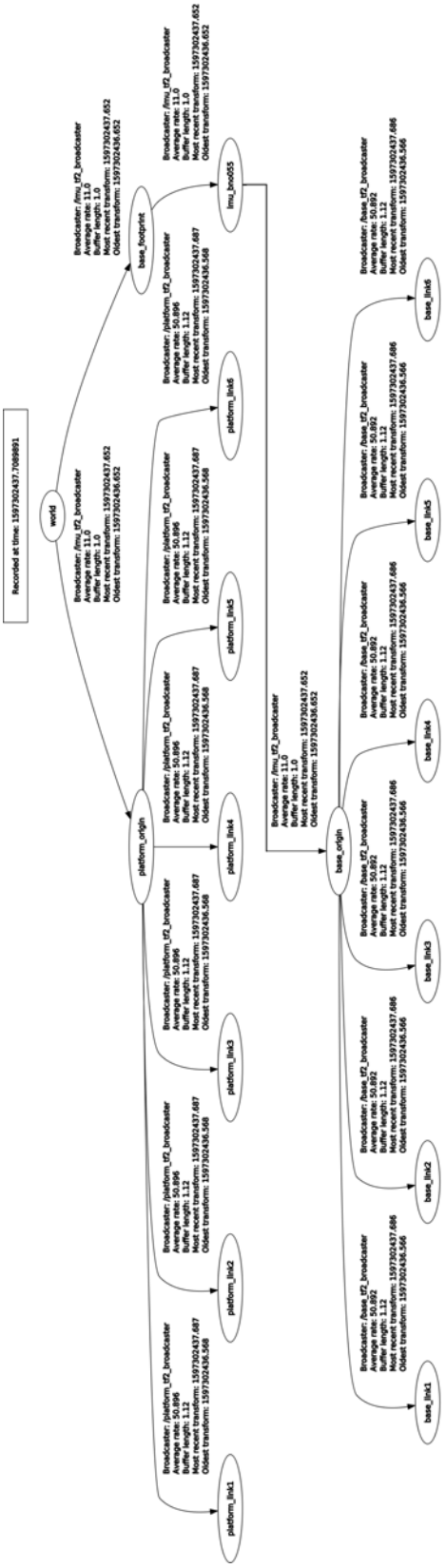


Abbildung A.1: ROS TF-Tree

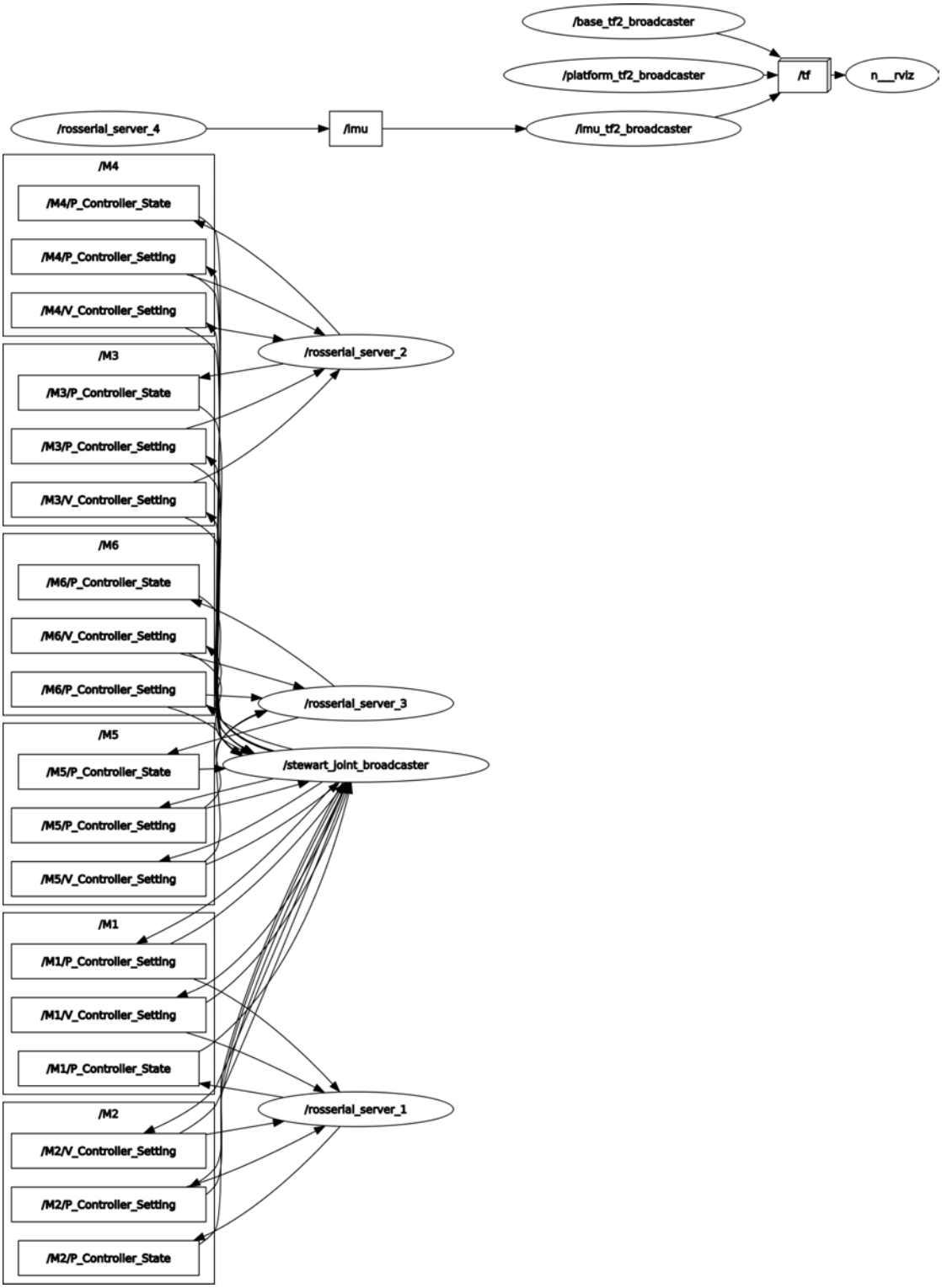
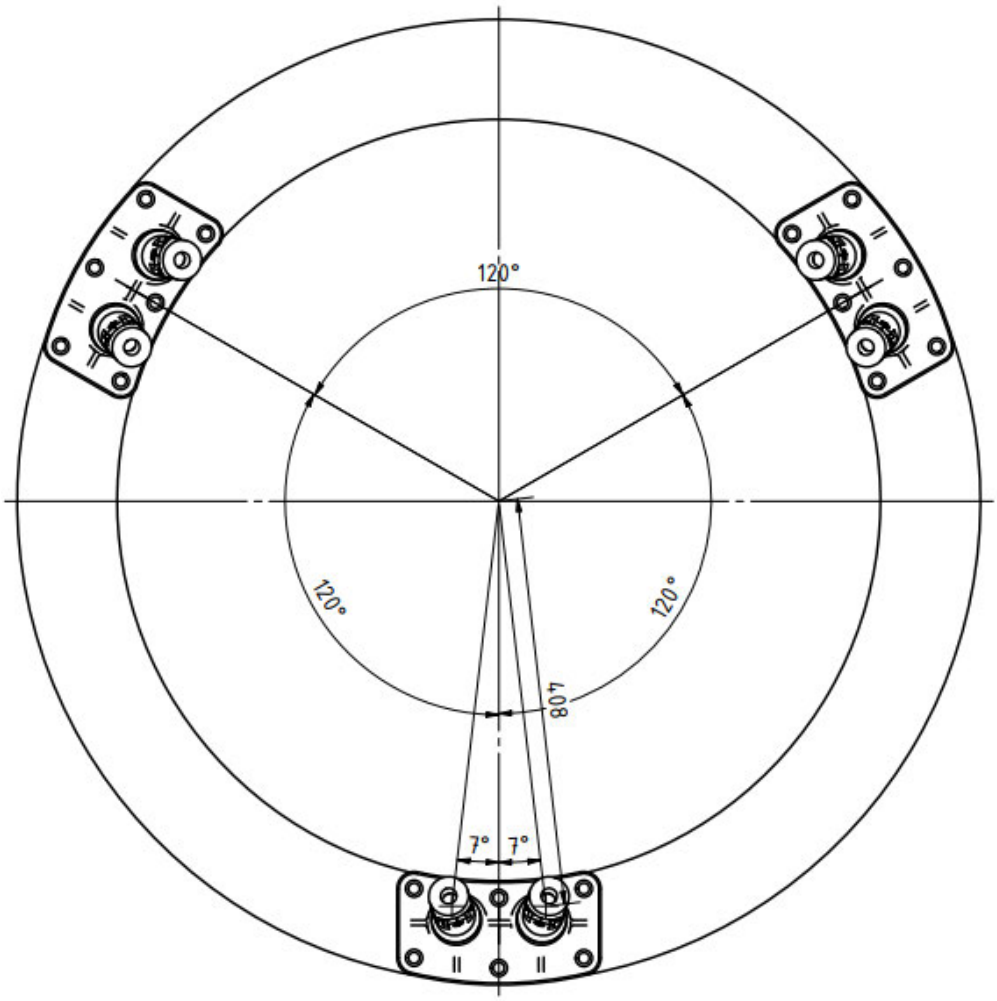
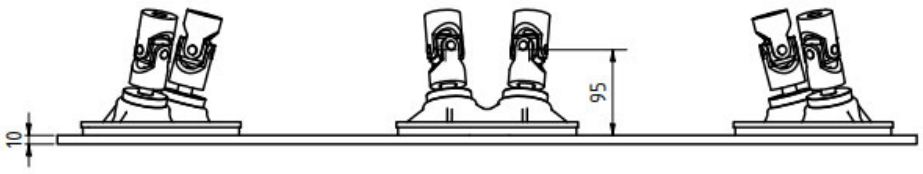


Abbildung A.2: Aktiver ROS Node-Tree

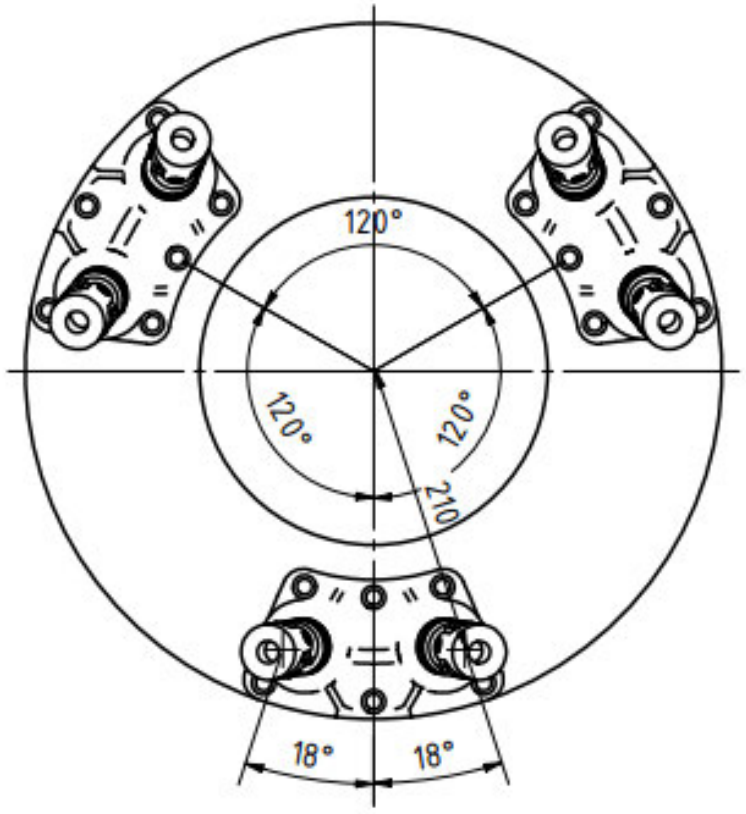


(a)

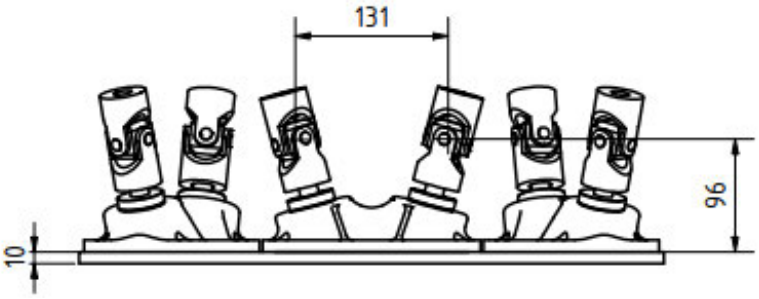


(b)

Abbildung A.3: Bemaßung der Basis



(a)



(b)

Abbildung A.4: Bemaßung der mobilen-Plattform

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konstruktion, Simulation und Regelungsimplementierung einer Stewart-Plattform zur Wellenkompensation

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

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der Bachelorarbeit ist erfolgt durch:

		[redacted]
Ort	Datum	Unterschrift im Original