

# **Konstruktion eines industriellen 6-DOF-Roboterarms**

Maturaarbeit

Vladimir Morozov

*Gymnasium Bäumlighof*

Begleitperson: Herr Dr. Dominik Rohner

13. Oktober 2025

# **Vorwort**

Ich möchte mich herzlich bei allen Mitwirkenden bedanken, die mich bei diesem Projekt unterstützt haben.

# Inhaltsverzeichnis

<b>Vorwort</b>	<b>2</b>
<b>1. Einleitung</b>	<b>5</b>
1.1. Zielsetzung . . . . .	5
1.2. Automatisierung durch Roboter . . . . .	6
<b>2. Aufbau eines industriellen 6-DOF-Roboterarms</b>	<b>7</b>
<b>3. Mechanische Umsetzung</b>	<b>11</b>
3.1. CAD-Software . . . . .	11
3.2. Material . . . . .	13
3.3. Aufbau . . . . .	14
3.4. Oberarm (2. Glied) . . . . .	14
3.5. Unterarm (3. Glied) . . . . .	14
3.6. Schluter und Halterung (1. Glied) . . . . .	14
3.7. Handgelenk Teil 1 (4. Glied) . . . . .	14
3.8. Handgelenk Teil 2 und 3 (5. und 6. Glied) . . . . .	14
3.9. Komplette Montage . . . . .	14
<b>4. Elektrische Aktoren</b>	<b>15</b>
4.1. Motoren . . . . .	15
4.2. Sensoren . . . . .	19
<b>5. Elektronik und Systemsteuerung</b>	<b>20</b>
5.1. Mikrokontroller . . . . .	20
5.2. Motortreiber . . . . .	21
5.3. PCB . . . . .	22
5.4. Code für Mikrokontroller . . . . .	22
5.5. Kontrolle über PC . . . . .	26
<b>6. Robotik</b>	<b>30</b>
6.1. Direkte Kinematik . . . . .	30
6.1.1. FK im 2-dimensionalem Raum . . . . .	30
6.1.2. FK im 3-dimensionalem Raum . . . . .	32

6.2.	Inverse Kinematik . . . . .	35
6.2.1.	1R-Manipulator . . . . .	35
6.2.2.	2R-Manipulator . . . . .	35
6.2.3.	3R-Manipulator . . . . .	38
6.2.4.	6R-Manipulator . . . . .	39
6.3.	Theoretische Umsetzung eines 6R-Manipulatoren mit IK . . . . .	43
6.4.	Praktische Umsetzung eines 6R-Manipulatoren mit IK . . . . .	46
<b>7.</b>	<b>Resultate und Diskussion</b>	<b>48</b>
7.1.	Komplette Konstruktion . . . . .	48
7.2.	Auswertung . . . . .	48
7.2.1.	Geschwindigkeit . . . . .	48
7.2.2.	Genauigkeit und Wiederholbarkeit . . . . .	48
<b>8.</b>	<b>Zusammenfassung und Ausblicke</b>	<b>49</b>
8.1.	Zukünftige Ergänzungen . . . . .	49
8.1.1.	Zusätzliche Funktionalitäten . . . . .	49
8.1.2.	Interpolation . . . . .	49
8.1.3.	FPGA . . . . .	49
8.1.4.	Einbettung von KI . . . . .	49
8.2.	Reflexion . . . . .	49
<b>9.</b>	<b>Quellenverzeichnis</b>	<b>50</b>
<b>10.</b>	<b>Redlichkeitserklärung</b>	<b>51</b>
<b>A.</b>	<b>Anhang</b>	<b>52</b>

# 1. Einleitung

## 1.1. Zielsetzung

Das Ziel dieser Arbeit ist, einen funktionstüchtigen industriellen Roboterarm mit 6 Freiheitsgraden, auf Englisch auf als die "*Degrees of Freedom*" oder auch mit der Abkürzung "*DOF*" bekannt. von Grund aus konstruieren zu können. Dies bedeutet, dass der Roboterarm einen beliebigen Punkt mit einer beliebigen Rotation in einem 3-dimensionalem Raum erreichen kann. Der Grund für die Auswahl stammt daraus, dass heutzutage die Automation von verschiedenste Aufgaben immer mehr wird und für diese jegliche Roboter eingesetzt werden. Dazu werden industrielle Roboterarme eingesetzt. Der konkrete Grund eines industriellen Roboterarms ist, dass heutzutage auch humane Roboterarme im Einsatz sind. Während humane Roboterarme gelähmte oder fehlende Arme ersetzen werden industrielle Roboterarme für **FINDE ETWAS** eingesetzt. Zusätzlich möchte ich unbedingt mich mit Themenbereichen auseinandersetzen, die mich interessieren oder ich in Zukunft gerne weiter erforschen würde.

Das Ziel ist jedoch nicht nur einen Roboterarm graphisch und theoretisch konstruieren zu können, sondern diesen auch physisch zusammensetzen und steuern zu können. Dazu gehören Einsätze von elektrischen Aktoren, hauptsächlich Motoren, eine Systemsteuerung von allen Aktoren und Sensoren mit einer entsprechenden Benutzeroberfläche, mit der man den Roboterarm steuern und bewegen kann. Deshalb werden in dieser Arbeit viele verschiedene Prinzipien aus diversen Themengebieten eingesetzt:

- |                  |                             |
|------------------|-----------------------------|
| - Elektronik     | - Robotik                   |
| - Informatik     | - Konstruktionsmodellierung |
| - Elektrotechnik | - Systemsteuerung           |
| - Mechanik       | - 3D-Druck                  |

Ein industrieller Roboterarm mit 6 Freiheitsgraden besteht dementsprechend aus 6 verschiedenen Glieder. Die Rotationen dieser 6 Glieder bestimmt die Endposition und Endrotation der Spitze des Roboterarms, auch als der Endeffektor genannt. Durch das eine bestimmte Position und Rotation für den Endeffektor gesetzt wird, müssen die Konfigurationen jedes einzelnen Gliedes berechnet werden. Dieser Prozess wird als die "*Inverse Kinematik*", ein

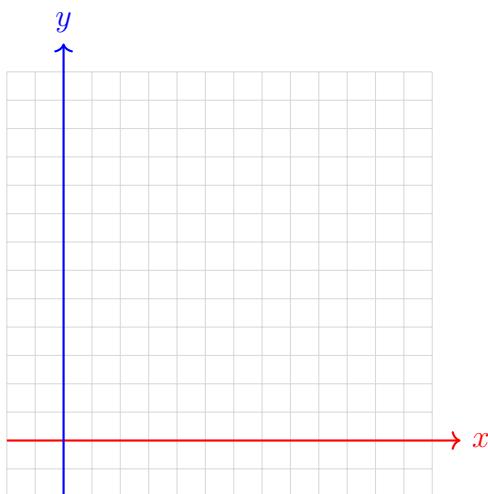
Begriff aus der Robotik, benannt und ist deshalb für das Erreichen einer Position im 3-dimensionalem Raum von entscheidender Bedeutung. Die Inverse Kinematik ist somit auch Teil dieser Arbeit.

## **1.2. Automatisierung durch Roboter**

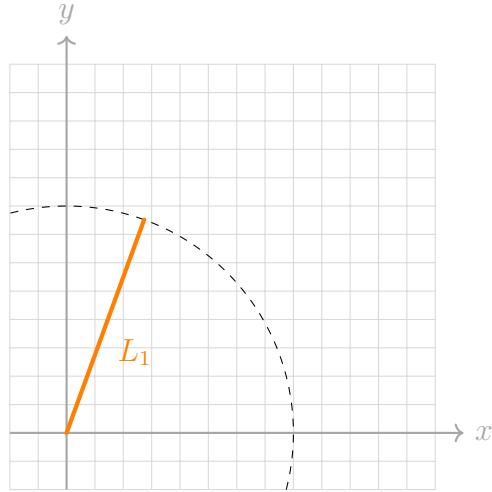
## 2. Aufbau eines industriellen 6-DOF-Roboterarms

Unter einem Roboterarm mit sechs Freiheitsgraden versteht man, dass dessen Endeffektor in einem 3-dimensionalem Raum in jede Richtung bewegen (in  $X$ -,  $Y$ - oder  $Z$ -Richtung) und beliebig rotiert werden kann (auch in  $X$ -,  $Y$ - oder  $Z$ -Richtung). Es können somit drei Positionsargumente ( $X, Y, Z$ ) sowie drei Rotationsargumente ( $\varphi, \theta, \psi$ ) bestimmt werden, die dem Endeffektor entsprechend entsprechen müssen. Für eine Konstruktion eines 6-DOF-Roboterarms muss eine solche Konstruktion so gegliedert sein, dass der Endeffektor frei im 3-dimensionalem Raum bewegt und rotiert werden kann. Damit verständlich wird, welche Glieder gebraucht werden, damit der Endeffektor diese Bedingungen erfüllen kann, wird graphisch schrittweise vorgegangen.

Vorerst wird in einem 2-dimensionalem Koordinatensystem gearbeitet.



Durch das jetzt schrittweise Glied nach Glied vorgegangen wird, wird das erste Glied so in das Koordinatensystem hinzugefügt, dass deren Anfang am Nullpunkt angesetzt wird und dort orthogonal zur  $X$ - und  $Y$ -Achse rotiert wird (an der imaginären  $Z$ -Achse). Anhand dieses Gliedes wird es möglich jeden Punkt auf einem Kreis erreichen zu können, deren Radius der Länge des Glieds entspricht.



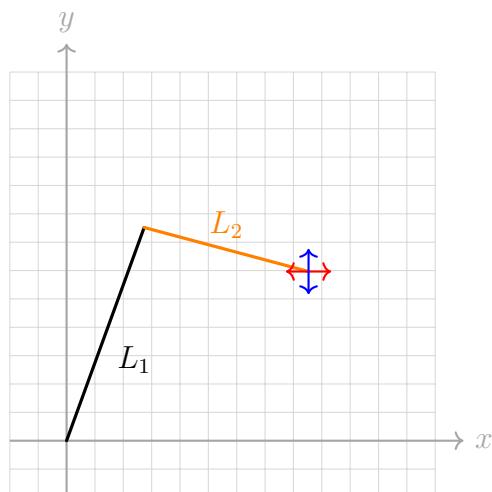
$$X_D = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$|X_D| = \sqrt{x^2 + y^2}$$

$$|X_D| = L_1$$

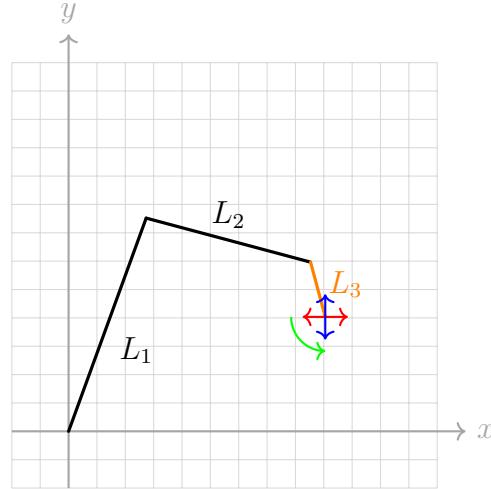
$$x^2 + y^2 = L_1^2$$

Damit ein Punkt erreicht werden kann, der nicht nur auf einem Kreis liegt, muss ein weiteres Glied am Ende des ersten Gliedes hinzugefügt werden, welches der Konstruktion erlaubt den Endeffektor an eine beliebige Position zu stellen, die nicht an einen Kreis gebunden ist. Der Endeffektor kann somit jeden Punkt in einem 2-dimensionalem System erreichen, solange sich dieser Punkt innerhalb eines Kreises mit dem Radius der beiden Längen der Glieder befindet. Gesprochen:



$$|X_D| \leq L_1 + L_2$$

Jedoch ist diese Konstruktion darauf bedingt, dass der Endeffektor eine Position erreicht, deren Rotation von der Position  $X_D$  und der Längen  $L_1$  und  $L_2$  der zwei Glieder entspricht. Damit eine bestimmte Rotation erreicht werden kann, muss ein weiteres Glied hinzugefügt werden, welches einen Punkt mit einer bestimmten Rotation erreichen kann.



Anhand von nur drei Glieder und deren Rotationsachsen kann im einem 2-dimensionalem Raum ein beliebiger Punkt mit einer bestimmten Rotation erreicht werden, bedingt davon, ob dieser Punkt mit der Rotation innerhalb der Greifweite der Konstruktion liegt. Von hier aus wird es möglich das gleiche in einem 3-dimensionalem Raum umsetzen zu können. Um vorerst nur eine beliebige Position in einem 3-dimensionalem Raum erreichen zu können, wird die 2-gliedrige Konstruktion genommen und in die 3-dimensionale gestreckt, indem ein Glied am Anfang des ersten Gliedes befestigt die in Y-Richtung gestreckt wird und in die Y-Richtung rotiert wird. Dadurch wird Endposition des Endeffektors von der 2-dimensionalen in die 3-dimensionale gestreckt. Es kann dadurch jeder Punkt innerhalb einer Kugel erreicht werden, dessen Radius der Summe der Längen der letzten zwei Glieder entsprechen. Gesprochen

$$X_D = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$X_N = \begin{bmatrix} x \\ y - L_1 \\ z \end{bmatrix}$$

$$\begin{aligned} |X_N| &= \sqrt{x^2 + (y - L_1)^2 + z^2} \\ |X_N| &= L_2 + L_3 \end{aligned} \tag{2.1}$$

Durch dass das erste Glied die anderen zwei Glieder in die Y-Richtung versetzt, ist die

$$\text{eigentliche Endposition } X_N = \begin{bmatrix} x \\ y - L_1 \\ z \end{bmatrix}.$$

Damit ebenfalls eine beliebige Rotation erreicht werden kann, werden 3 weitere Glieder am Ende hinzugefügt, die es erlauben, eine Position mit einer bestimmten Rotation erreichen zu können. Dies kann auch graphisch veranschaulicht werden.

Zuerst wird ein Glied aufgebaut, die in die Y-Richtung gestreckt wird und in Y-Richtung rotiert wird. An das Ende des ersten Gliedes wird ein weiteres angesetzt, die auch in Y-Richtung gestreckt wird und in Z-Richtung rotiert wird. Somit wird das zweite Glied durch die Rotation des ersten Gliedes beeinflusst. Schlussendlich wird ein letztes Glied nochmals in Y-Richtung gestreckt und in Y-Richtung rotiert. Die ersten zwei Glieder sind somit dafür zuständig eine bestimmte Rotation zu erreichen und das Letzte nochmals für eine lokale Rotation. Diese drei Glieder werden somit an die ersten drei angesetzt und es wird ein industrieller Roboterarm mit sechs Freiheitgraden (*6DOF*) erzeugt, der aus insgesamt sechs Gliedern besteht.

# **3. Mechanische Umsetzung**

## **3.1. CAD-Software**

Um das physische Produkt verwirklichen zu können, wurden verschiedene CAD-Softwares (Computer-Aided Design) während des Projektes eingesetzt. CAD-Software sind vor allem hilfreich, dass man digitale Konstruktionen fertigen kann, mit denen man zusätzlich Bewegungs- oder Stresssimulationen durchführen kann. Dadurch wird der Arbeitsprozess zur Konstruktion eines Roboterarm rasant beschleunigt. Genauer gesagt wurden 3D-CAD-Software benutzt.

Zum Anfang des Projekts wurde Fusion 360, eine von Autodesk entwickelte Applikation, als die gewählte CAD-Software eingesetzt, da ich mich darin schon gut auskannte und über eine aktive Schülerlizenz verfügte. Fusion 360 ist sehr benutzerfreundlich und übersichtlich und somit sehr gut für Neuanfänger, wird aber auch sehr viel von Professionellen eingesetzt, da auch für extremere Konstruktionen diverse Funktionalitäten vorhanden sind. Jedoch tauchten über die Zeit viele Probleme auf, die den Arbeitsprozess verschlechterten. Ich suchte somit nach einer neuen Alternative für Fusion 360, mit der ich meine Arbeit weiterführen könnte.

SolidWorks, eine von Dassault Systems entwickelte Applikation, wurde schlussendlich der Ersatz für Fusion 360 bis zum Ende des Projektes, mit dem jedes Teil des Roboterarms modelliert worden ist. Durch das ein paar Teile schon in Fusion 360 modelliert worden sind, mussten diese in SolidWorks erneut modelliert werden, wodurch ich mich damit besser auseinandersetzen konnte. Mit dem Umstieg zu SolidWorks war es möglich kompliziertere Teile modellieren zu können und diese in eine grosse und bewegliche Konstruktion stellen zu können. Grundsätzlich unterscheiden sich die zwei CAD-Software nicht viel voneinander, jedoch ist SolidWorks viel optimierter, was beim Arbeitsprozess deutlich spürbar ist.

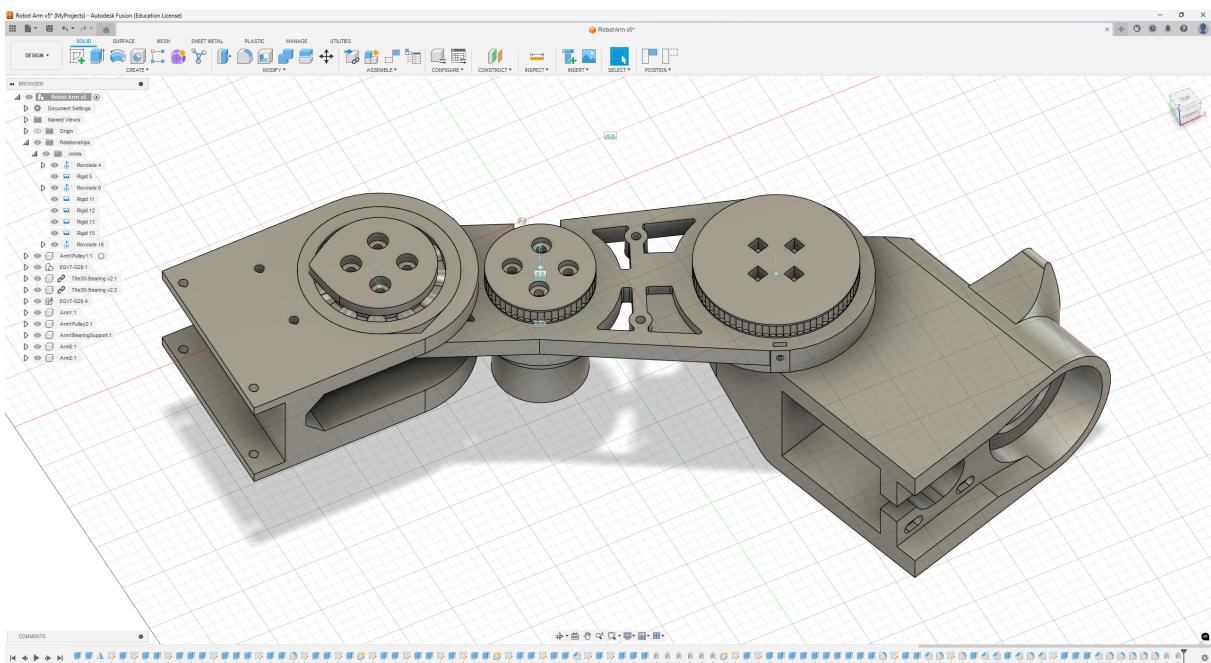


Abbildung 3.1.: Screenshot von einem Workflow in Fusion 360

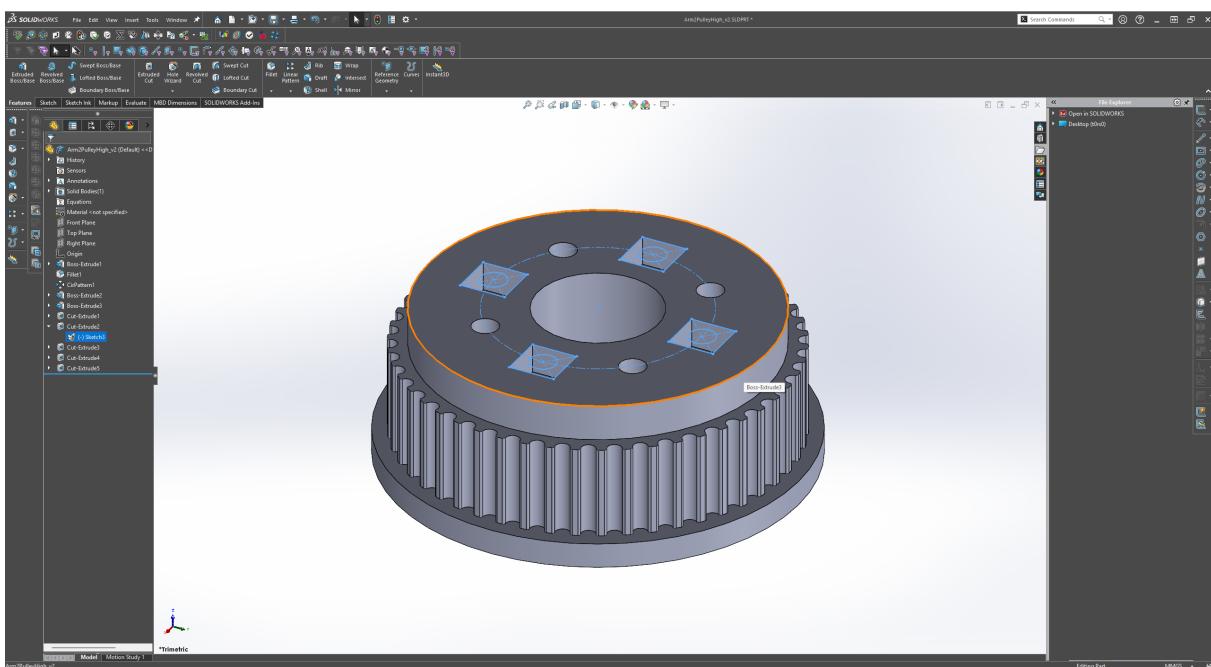


Abbildung 3.2.: Screenshot von einem Workflow in SOLIDWORKS

## 3.2. Material

Um die jegliche Teile des Roboterarms fertigen zu können, habe ich mich auf das 3D-Drucken verlassen. Diese Auswahl war entscheidend, da das Fertigen an Teilen, sowie auch Prototypen, schneller und günstiger wäre, als diese aus Holz oder Metall zu fertigen. Durch das 3D-Drucken entsteht zusätzlich eine viel grössere geometrische Freiheit, wo diese bei traditionell CNC-gefertigten Teilen fehlt. Durch das mit Plastik ausgedruckt wird, was grundsätzlich ziemlich billig ist, fallen keine grossen Kosten auf, vor allem wenn Prototypenteile gefertigt werden, die im Nachhinein in der Endkonstruktion nicht benutzt werden. Als verwendetes Material wurde vor allem PLA eingesetzt, welches sehr einfach zu drucken und zu verarbeiten ist. Jedoch ist PLA nicht sehr Hitzeresistent, was kritisch ist, wenn Motoren daran angehängt werden, die durch deren Stromverbrauch Hitze erzeugen und das Plastik einfach deformieren können. An den Stellen, wo die Motoren angesetzt werden, werden Scheiben oder sogar ganze Teile aus PCCF (Polycarbonate Blend Carbon Fiber) von Prusament eingesetzt, da dieses Plastik viel hitzeresistenter ist. Durch das die Zugfestigkeit des PCCFs auch viel grösser ist als diese des PLAs, werden auch manche mechanisch beanspruchenden Teile damit ausgedruckt. Um das alles auszudrucken, wurde ein von Prusa Research entwickelter 3D-Drucker eingesetzt. Am Anfang des Projekts wurde der Prusa MK4S eingesetzt, wurde aber zum Ende noch aufgewertet auf einen Prusa Core ONE, mit dem es möglich wurde, Teile aus PCCF ausdrucken zu können. Diese 3D-Drucker entsprechen zu einem FDM-Drucker (Fused Deposition Modeling), der Schicht nach Schicht ein Filament erhitzt und verlegt, aus dem ein gedrucktes Teil erfolgt.



(a) Prusa MK4S



(b) Prusa Core ONE

Abbildung 3.3.: Abbildungen von zwei 3D-Druckern, die während des Projektes eingesetzt worden sind.

### **3.3. Aufbau**

Der ganze Roboterarm besteht aus insgesamt 6 verschiedenen Gliedern. Jedes Glied verfügt über eine eigene Rotationsachse, sowie ein Fertigungspunkt, wo das weitere Glied befestigt und rotiert wird. Die verschiedenen Glieder und deren Rotationen sind in der folgenden Abbildung zu sehen:

- 1. Glied: Schulter (und Halterung)
- 2. Glied: Oberarm
- 3. Glied: Unterarm
- 4. Glied: Handgelenk 1
- 5. Glied: Handgelenk 2
- 6. Glied: Handgelenk 3

### **3.4. Oberarm (2. Glied)**

### **3.5. Unterarm (3. Glied)**

### **3.6. Schluter und Halterung (1. Glied)**

### **3.7. Handgelenk Teil 1 (4. Glied)**

### **3.8. Handgelenk Teil 2 und 3 (5. und 6. Glied)**

### **3.9. Komplette Montage**

# 4. Elektrische Aktoren

## 4.1. Motoren

Für die Bewegung eines Roboterarms sind Motoren benötigt. Die traditionellen DC-Motoren, die heutzutage meistverbreitet sind, würden nicht direkt passen, da diese keine Kontrolle über deren Rotation haben, sondern sich mit einer Stromversorgung kontinuierlich drehen. Für eine präzise Kontrolle des Roboterarms werden Motoren gebraucht, deren Bewegungen präzise, wiederholbar und vorhersehbar sind. Dazu könnten Servo-Motoren eingesetzt werden, die auch aus einem DC-Motor bestehen, aber noch zusätzlich über einem Potentiometer verfügen, um deren Rotation zu ermitteln. Mithilfe des Potentiometers und eines eingebauten PID-Controllers wird es ermöglicht eine bestimmte Rotation erreichen zu können, mit der einzigen Limitation, dass diese Rotation auf einen bestimmten Winkel limitiert wird, meist auf  $180^\circ$ . Sobald eine grössere Rotation als diese der Reichweite des Servomotors benötigt wird, können die Servomotoren nicht mehr richtig eingesetzt werden.

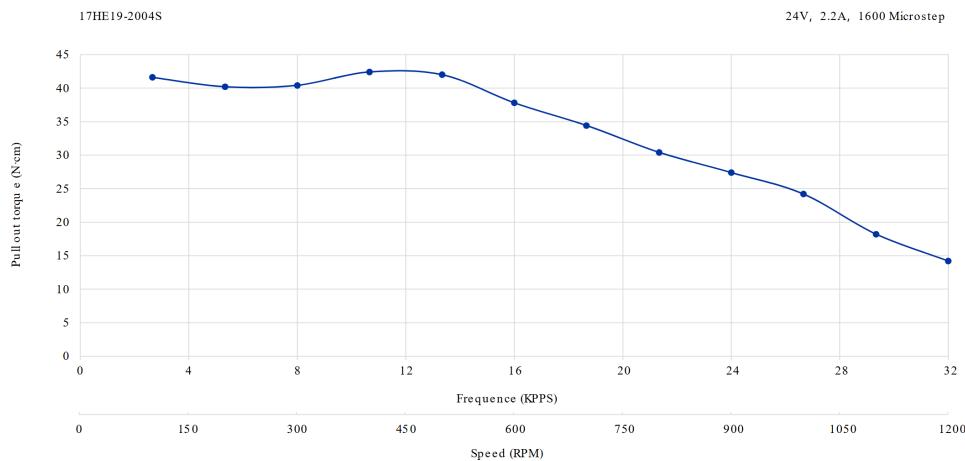
Schrittmotoren vermeiden diese kontinuierliche Bewegung, indem sie schrittweise rotieren können. Somit wird es nicht mehr nötig deren Rotation anhand von aktiven Sensoren bestimmen zu müssen, da es bekannt ist, was für eine Rotation mit einer bestimmten Anzahl Schritte erreicht werden kann. Für die Grösse des Roboterarms orientierte ich mich vor allem an der Standardgrösse NEMA-17 (42mm x 42mm) eines Schrittmotors. Es wurden insgesamt drei verschiedene Längen von Schrittmotoren in diesem Projekt eingesetzt worden. Jedoch braucht es für verschiedene Anwendungen auch verschiedene



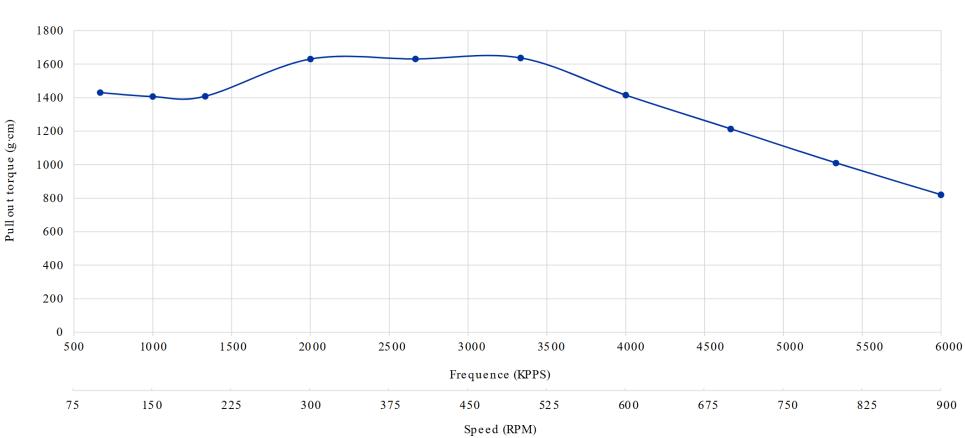
(a) NEMA-17 25mm      (b) NEMA-17 48mm      (c) NEMA-17 60mm

Abbildung 4.1.: Im Roboterarm eingesetzte Schrittmotoren mit verschiedenen Längen

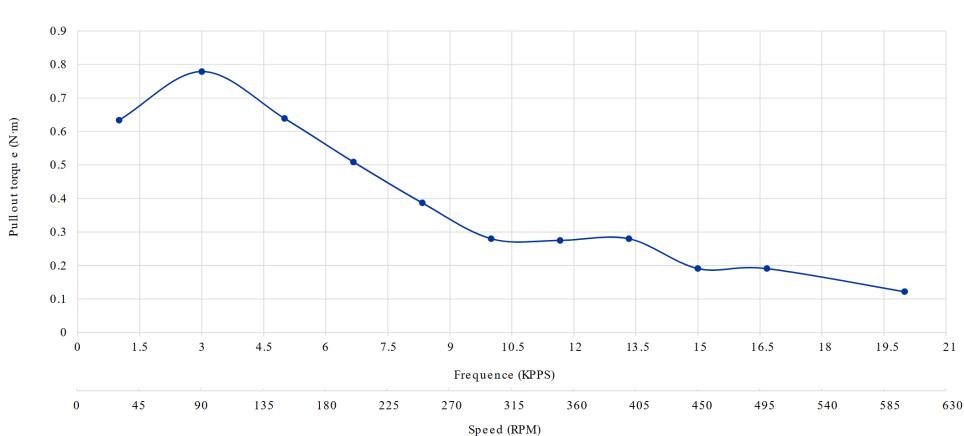
Drehgeschwindigkeiten Drehmomente, die von der Länge eines Schrittmotors abhängig sind. Dementsprechend verfügt jeder Schrittmotor einer anderen Länge über eine Drehmoment-Kurve, bei dem ersichtlich ist, bei was die maximale Geschwindigkeit ist und welches Drehmoment bei einer bestimmter Drehgeschwindigkeit erreicht wird. Somit werden für Einsätze, wo mehr Drehmoment verlangt, wird als Geschwindigkeit, längere Schrittmotoren eingesetzt, wobei bei Einsätzen, wo weniger Drehmoment verlangt wird als Geschwindigkeit, werden kürzere Schrittmotoren eingesetzt. Auf der nächsten Seite sind die Drehmomentkurven der einzelnen Schrittmotoren zu sehen.



(a) Drehmomentkurve von NEMA-17 25mm Schrittmotor



(b) Drehmomentkurve von NEMA-17 48mm Schrittmotor



(c) Drehmomentkurve von NEMA-17 60mm Schrittmotor

Abbildung 4.2.: Drehmomentkurven aller Schrittmotoren, wo  $KPPS$  = "kilopulses per second",  $Ncm$  = "Newton centimeter" und  $RPM$  = "revolutions per minute"

Ein typischer Schrittmotor kann schrittweise funktionieren, indem das Innere aus zwei Spulen besteht. An einen Schrittmotor können somit 4 Kabel verbunden werden, wo jedes Kabelpaar zu einer Spule zugehört. Durch das die Kabelpaare mit Strom versorgt werden wird ein dementsprechend ein Magnetfeld induziert. Im Inneren des Schrittmotors befindet sich ein Rotor, der so verbaut ist, dass diese an ein Magnetfeld gerichtet werden kann. Somit kann durch eine Reihenfolge von Umschaltungen der Polaritäten der Spulen eine Rotation errichtet werden. Ein NEMA-17 Schrittmotor besteht aus der Regel aus 8 solcher Spulen,

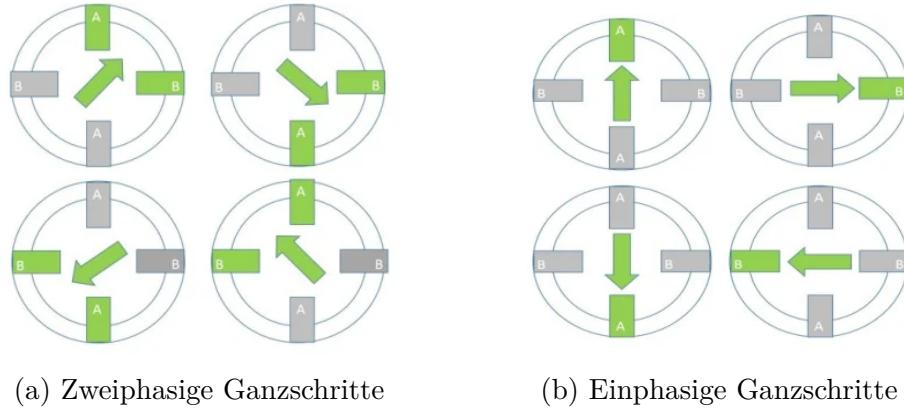


Abbildung 4.3.: Phasenweise Schrittaufteilung eines Schrittmotors

pro Kabelpaar insgesamt 4 Spulen. Diese Spulengruppen werden als Phasen bezeichnet. Der Rotor besteht jedoch nicht nur aus einem Stück gerichtetem Magneten, sondern aus einer runden gezackten Achse, welche von den Zacken an den Spulen angezogen werden, welche feinere Schritte erzeugt. Dadurch kann eine Umpolung einer Spule zu einer kleinen Rotationsbewegung übermittelt werden. Ein NEMA-17 Motor benötigt 200 Schritte, um eine ganze Umdrehung durchzuführen.

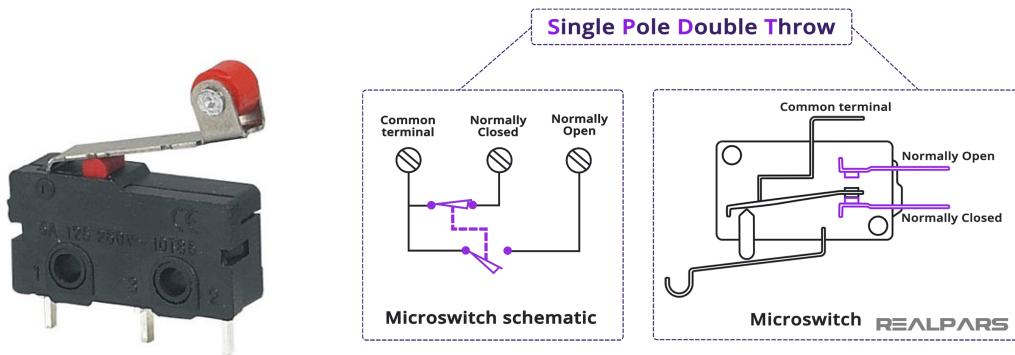


Abbildung 4.4.: Innerer Aufbau eines typischen NEMA-17 Schrittmotoren

## 4.2. Sensoren

Damit die Positionierung des Roboterarms bekannt ist, werden Sensoren eingesetzt, die dazu dienen zu bestimmen, in welcher Orientation jedes Glied sich befindet. Es können entweder aktive Sensoren, die ständig die Orientation eines Gliedes abmessen (Motor-Encoder) oder passive Sensoren, die nur durch deren Aktivierung eine Abmessung geben (Knöpfe, Schalter), verwendet werden.

Für diese Arbeit wurden Grenzschalter eingesetzt, die wie ein Knopf funktionieren, indem sie betätigt werden können. Solche Grenzschalter werden als passive Sensoren bezeichnet. Diese Grenzschalter werden an gefertigten Orten an den modellierten Gliedern befestigt, wo diese durch die Rotation der Glieder betätigt werden. Durch die Betätigung des Grenzschalter bei dem entsprechenden Glied kann die genaue Position bekannt gegeben werden. Dieser Prozess wird auch als "Homing" bezeichnet, indem die Standardposition erreicht wird. Der Grenzschalter verfügt über drei Verbindungen: NC (Normally Closed), NO (Normally Open) und C (Common Terminal) der Verbindungsterminal ist. Aus Sicherheitsgründen wird der Grenzschalter mit C und NC verbunden, da dadurch bekannt ist, wenn der Grenzschalter nicht betätigt wird, dass es einen geschlossenen Schaltkreis gibt.



(a) Eingesetzter Grenzschalter

(b) Elektrisches Schema eines Grenzschalters

Abbildung 4.5.: Abbildung des eingesetzten Grenzschalters mit einem dazu entsprechendem elektrischen Schema.

# 5. Elektronik und Systemsteuerung

## 5.1. Mikrokontroller

Für die Kontrolle eines Roboterarms wird ein Verarbeitungsterminal für die Lesung aller Zustände der Sensoren und die Steuerung der Motortreiber benötigt. Für solche Verarbeitungen werden meist grosse Computer eingesetzt, die solche Aufgaben einfach erledigen können. Jedoch können nicht immer grosse Computer eingesetzt werden, da diese meistens zu Energieaufwendig sind und gross sind. Deshalb werden in solchen Situationen Mikrokontroller eingesetzt, die klein, kompakt und meist sehr energieeffizient sind. Solche Mikrokontroller werden deshalb öfters für kleine DIY-Projekte eingesetzt und sind damit die passende Auswahl für einen kompakten Roboterarm.

Der benutzte Mikrokontroller für diesen Roboterarm ist ein ESP32-S3-DevKitM-1 vom Hersteller Espressif aus der ESP32-Reihe. Insgesamt verfügt dieser Mikrokontroller über 48 I/O-Pins, welche die vielen Verbindungen zwischen den Treibern und Sensoren gewährleisten können. Durch einen schnellen Mikroprozessor, einer grossen Speicherkapazität und einer WiFi- und Bluetooth-Funktionalität, die für die Kommunikation zwischen dem Mikrokontroller und einem PC/Laptop eingesetzt werden können, damit der Roboter entsprechend gesteuert werden kann.

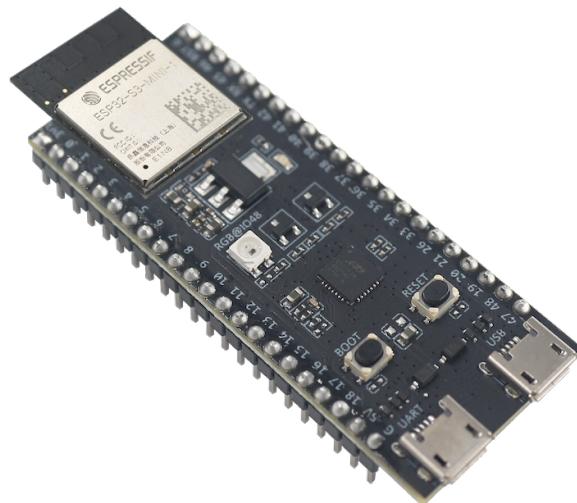
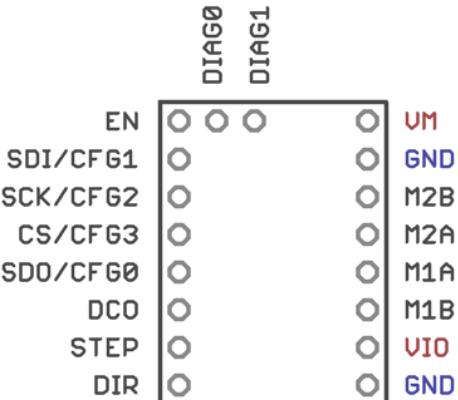
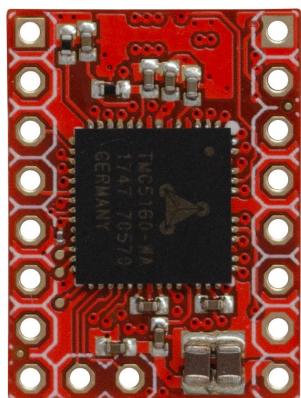


Abbildung 5.1.: Isometrische Darstellung des Mikrocontrollers ESP32-S3-DevKitM-1

## 5.2. Motortreiber

Für die Steuerung der Schrittmotoren werden TMC5160-Treiber von Analog Devices verwendet, um die Motoren phasenweise steuern zu können. Sie dienen nicht nur dazu Motoren zu steuern, sondern auch deren Stromversorgung an die Motore zu regeln. Ein NEMA-17 Schrittmotor verbraucht etwa 800mA (maximal 1500mA) bei 24V, was sich zu etwa 19.2W ergibt (maximal 36W). Da die Treiber über einen ESP32-Mikrokontroller gesteuert werden und dieser Mikrokontroller durch einen USB-Kabel verbunden wird, können nur etwa 500mA bei 5V geliefert werden, was zu 2.5W entspricht. Mit dieser Leistung wäre es nicht richtig möglich die Motore antreiben zu können, noch weniger sechs auf einmal. Deshalb kann eine externe Stromversorgung an die Treiber angeschlossen werden, die die Motore mit einer genügend grossen Leistung angetrieben werden können. Zusätzlich verfügen die TMC5160-Treiber über einem IC (Integrated Circuit), das die ganzen Schrittphasen eines Schrittmotors steuert und dadurch durch zwei Pins kontrolliert werden kann. (STEP: Schritt, DIR: Drehrichtung). Ein weiterer Pin dient dazu den Treiber zu aktivieren oder zu deaktivieren (EN: Aktivieren).



(a) Darstellung des TMC5160-IC integriert auf einem PCB mit Anschlusspins

(b) Pinaufteilung des TMC5160-PCBs

Abbildung 5.2.: Da der TMC5160 ein IC ist, wird es auf eine PCB integriert, die dann als Daughterboard in anderen Applikationen eingesetzt und einfach ausgewechselt werden kann.

## 5.3. PCB

## 5.4. Code für Mikrokontroller

### Parameter- und Variablenaufbau

Code 5.1: Definierung der I/O-Pins am Mikrokontroller für die Treiber

```
1 // PINS FÜR TMC-SPI
2 #define SDO_PIN 41
3 #define SCK_PIN 39
4 #define SDI_PIN 40
5 // PINS FÜR MOTOR 1
6 #define DIRO_PIN 4
7 #define STEPO_PIN 3
8 #define ENO_PIN 1
9 #define CSO_PIN 2
10 #define SWO_PIN 13
11 ...
```

Code 5.2: Definierung der Parameter und Variablen für die Kontroller der Motoren

```
1 //MOTOR VARIABLES
2 int default_steps = 200;
3 int mot_speed[6];
4 int mot_accel[6];
5 float mot_reduction[6];
6 int mot_home_speed[6];
7 bool mot_home_inv[6];
8 int mot_home_accel[6];
9 int mot_shome_offset[6];
10 float mot_home_mult[6];
11 37
12 float mot_home_offset[6];
13 int mot_mcbs[6];
14 int mot_irun[6];
15 int mot_ihold[6];
16 // MOTORO VARIABLES
17 //Positional Arguments
18 float current_deg0 = 0;
19 float move_to_deg0 = 0;
20 int move_to0 = 0;
21 //Homing Arguments
22 bool home_mot0 = false;
```

```
23 bool second_home0 = false;  
24 bool home_slow0 = false;  
25 ...
```

---

## Initialisierung der Treiber- und Motorsteuerung

Code 5.3: Definierung der Treiber durch `TMCStepper.h`

```
1 TMC5160Stepper driver[] = {  
2     TMC5160Stepper(CS0_PIN, R_SENSE),  
3     TMC5160Stepper(CS1_PIN, R_SENSE),  
4     TMC5160Stepper(CS2_PIN, R_SENSE),  
5     TMC5160Stepper(CS3_PIN, R_SENSE),  
6     TMC5160Stepper(CS4_PIN, R_SENSE),  
7     TMC5160Stepper(CS5_PIN, R_SENSE)  
8 };
```

---

Code 5.4: Definierung der Motoren durch `AccelStepper.h`

```
1 AccelStepper stepper[] = {  
2     AccelStepper(1, STEP0_PIN, DIR0_PIN),  
3     AccelStepper(1, STEP1_PIN, DIR1_PIN),  
4     AccelStepper(1, STEP2_PIN, DIR2_PIN),  
5     AccelStepper(1, STEP3_PIN, DIR3_PIN),  
6     AccelStepper(1, STEP4_PIN, DIR4_PIN),  
7     AccelStepper(1, STEP5_PIN, DIR5_PIN)  
8 };
```

---

Code 5.5: Vordefinierte Kontrollfunktionen der Bibliotheken `AccelStepper.h` und `TMCStepper.h`

```
1 TMC5160Stepper driver[] = {  
2     TMC5160Stepper(CS0_PIN, R_SENSE),  
3     TMC5160Stepper(CS1_PIN, R_SENSE),  
4     TMC5160Stepper(CS2_PIN, R_SENSE),  
5     TMC5160Stepper(CS3_PIN, R_SENSE),  
6     TMC5160Stepper(CS4_PIN, R_SENSE),  
7     TMC5160Stepper(CS5_PIN, R_SENSE)  
8 };
```

---

## Kommunikationsstruktur

Durch die eingebaute WiFi-Funktionalität, durch die eine kabellose Verbindung zu einem Gerät hergestellt werden kann, könnte jedes Gerät sich mit dem ESP32 verbinden und es kontrollieren. Beim Starten des Programms erstellt der ESP32 ein eigenes WLAN-Netzwerk, einen Access Point oder ÄP", mit dem sich ein Computer oder Smartphone verbinden kann. Über dieses Netzwerk läuft eine Kommunikation mithilfe des WebSocket-Protokolls, das eine schnelle und bidirektionale Übertragung von Daten erlaubt.

Die Kommunikation zwischen dem Computer und dem Mikrokontroller erfolgt über Textnachrichten (Strings), wo jede Nachricht über eine definierte Struktur besitzt:

{MOTOR}{AKTION}{WERT}

- **MOTOR:** Ein Integer von 0 bis 5, wo jede Zahl einer Rotationsachse entspricht.
- **AKTION:** Eine zweistellige Zahl, die beschreibt, welcher Parameter verändert oder was ausgeführt werden sollte.
- **WERT:** Ein Zahlenwert, entweder ein Integer oder Float, z.B. ein Winkel in Grad oder ein Parameterwert.

Beispielsweise entspricht der Befehl 10190 für den Motor 1, Aktion 01 und dem Wert 90, also "Drehe Motor 1 auf 90 Grad". Damit diese Nachrichten vom ESP32 verarbeitet werden können, werden zwei Abschnitte der Textnachricht aufgenommen

---

```
1 String mot = msg.substring(0, 1);
2 String act = msg.substring(1, 3);
```

---

wo **msg** die erhaltene Textnachricht ist. Durch ein Switch-Case-Block wird entschieden, welche Aktion bei welchem Motor durchgeführt werden sollte

Code 5.6: Switch-Case-Block für die Kontrolle des entsprechenden Motors

---

```
1 switch (mot.toInt()) {
2     case 0: // Motor 0
3         if (act == "01") stepper[0].moveTo(msg.substring(3).ToFloat());
4         break;
5     ...
6 }
```

---

## Bewegungssteuerung

Code 5.7: for-Loop für das Antrieben jedes Motors

---

```
1 if(can_move){  
2     for(int i = 0; i < 6; i++){  
3         stepper[i].run();  
4     }  
5 }
```

---

## 5.5. Kontrolle über PC

### Benutzeroberfläche

Die Benutzeroberfläche, die auf dem PC läuft und dadurch mit dem Mikrokontroller kommuniziert, mit dem der Roboterarm kontrolliert werden kann, wurde in Python geschrieben, da diese am besten ist, um Applikationen zu schreiben, egal ob mit oder ohne UI. Für die Kontrolle des Roboterarms wurde jedoch wichtig, einen Überblick von allen Parametern zu haben, sowie den Roboterarm steuern zu können. Für die Benutzeroberfläche benützte ich die `gradio.py` Bibliothek, mit der einfach eine lokale Website gehostet werden kann und UI-Elemente einfach hinzugefügt werden können. Dadurch wurde möglich eine UI zu designen, die einem einen guten Überblick von allen Einstellungen schaffen kann.

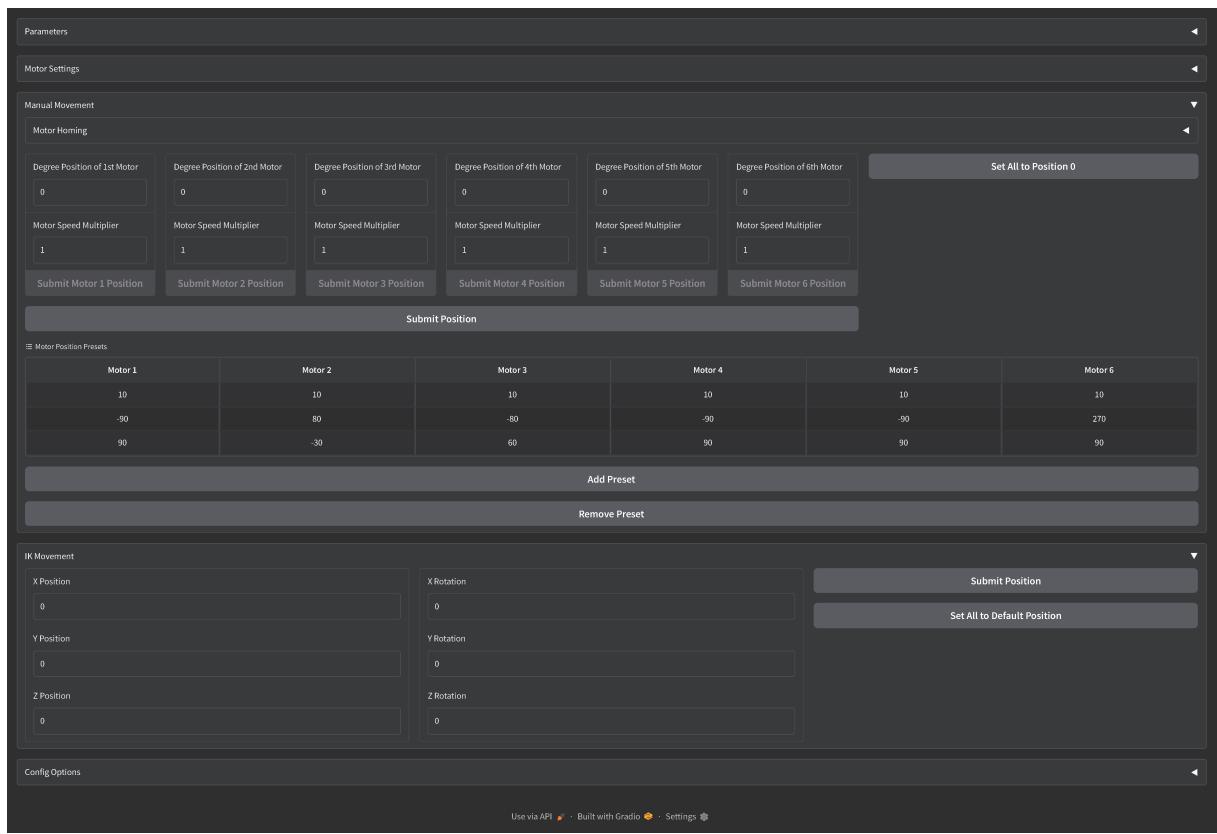


Abbildung 5.3.: Screenshot von der Benutzeroberfläche mithilfe der Bibliothek `gradio.py`

## Kommunikation mit dem Mikrokontroller

Die Kommunikation mit dem ESP32 erfolgt über einer `websockets.py` Bibliothek für die WebSockets Kommunikation, sowie `asyncio.py`, welches erlaubt, eine Aufgabe ausserhalb der normalen Programmausführung asynchron ausführen zu können. Es wird von einer asynchroner Aufgabenerledigung. Es wird somit zu der lokalen IP des ESP32 verbunden und die entsprechenden Textnachrichten verschickt.

Code 5.8: Senden von Textnachrichten durch WebSockets (*Python*)

```
1  async def con(data: list) -> None:
2      async with websockets.connect('ws://192.168.4.1/ws') as websocket:
3          for i in data: # Durchgang aller Textnachrichten in einer Liste
4              await websocket.send(i) # Verschicken von den Textnachrichten
```

Sobald Daten vom Mikrokontroller benötigt werden, werden diese auch anhand von WebSockets angefragt.

Code 5.9: Erhalten von Textnachrichten durch WebSockets (*Python*)

```
1  async def con_get(data: list) -> list:
2      end_data = []
3      async with websockets.connect('ws://192.168.4.1/ws') as websocket:
4          for i in data:
5              await websocket.send(i)
6              end_data.append(str(await websocket.recv()))
7      return end_data
```

## JSON-Konfigurationen

Dadurch, dass alle Parameter durch den PC angegeben werden und nicht lokal auf dem Mikrokontroller gespeichert werden und diese immer verändert werden müssen, wird lokal auf dem PC eine JSON-Datei erstellt, auf der alle Daten für die Einstellungen und Parameter gespeichert werden und bei dem Starten der Website aufgerufen werden.

Code 5.10: JSON-Loading der Konfigurationen (*Python*)

---

```
1 import json
2 def load_json_config() -> dict:
3     with open('../data/user_config.json') as f:
4         d = json.load(f)
5         f.close()
6     return d
```

---

## Geschwindigkeitsplanung der Motoren

Es wird erwartet, dass die Bewegung aller Motoren gleichzeitig anfangen und gleichzeitig enden, eine synchrone Bewegung. Dabei muss um diese erreichen zu können, müssen die Limitationen aller Motoren festgesetzt werden. Jeder Motor hat eine andere mechanische Reduktion, und somit auch eine eigene maximale Geschwindigkeit, bei der es operieren kann. Diese werden bei den Parametern der Motoren festgestellt werden. Danach wird berechnet, welcher Motor die längste Rotation abschliessen sollte, da dieser schlussendlich am schnellsten bewegt werden sollte. Die maximale Geschwindigkeit von diesen Motoren wird übernommen, wo jeder Motor diese maximale Geschwindigkeit übernimmt und diese mit dem Verhältnis der Rotation von dem Motor mit der längsten Rotation, zu der der eigenen multipliziert. Sobald die vorgegebene maximale Geschwindigkeit diese der eigenen überschreitet, wird dessen maximale Geschwindigkeit übernommen und der Prozess wird erneut auf alle Motoren angewandt, bis alle Geschwindigkeiten befriedigend sind.

Code 5.11: Geschwindigkeitsplanung der Motoren (*Python*)

---

```
1 import numpy as np
2 def mots(rotation, max_speeds, reduction):
3     rot = np.asarray(rotation, dtype=float)
4     vmax = np.asarray(max_speeds, dtype=float)
5     red = np.asarray(reduction, dtype=float)
6
7     speeds = np.zeros(6, dtype=float)
8
9     eff = np.abs(rot) * red
10    if not np.any(eff > 0.0):
11        return speeds.tolist()
12
13    m = int(np.argmax(eff))
14
15    satisfied = False
16    while not satisfied:
17        satisfied = True
18
19        base = vmax[m] / eff[m]
20
21        for i in range(6):
22            if eff[i] == 0.0:
23                speeds[i] = 0.0
24                continue
25            speed = base * eff[i]
26            if speed > vmax[i]:
27                m = i
28                satisfied = False
29                break
30            speeds[i] = speed
31
32    return speeds.tolist()
```

---

# 6. Robotik

## 6.1. Direkte Kinematik

Der Begriff "Direkte Kinematik", auch als "Forward Kinematics" oder "FK" bekannt, stammt aus der Robotik und ist dafür da, um aus Gelenkwinkeln der einzelnen Glieder einer gegliederten Konstruktion die Endposition in einem Koordinatensystem bestimmen zu können. Man spricht bei der Endposition des letzten Glieds von einem "Endeffektor".

### 6.1.1. FK im 2-dimensionalem Raum

Um zu verstehen, was man unter der direkten Kinematik erreichen möchte, wird ein 2-dimensionales XY-Koordinatensystem dargestellt. In dem Koordinatensystem befindet sich eine Konstruktion aus 2 Gliedern. Das erste Glied  $L_1$  dreht sich mit dem Anfang um den Nullpunkt, während das zweite Glied  $L_2$  sich am Schluss des ersten Glieds dreht.

Jedes von diesen Gliedern liegt unter einem Winkel relativ zur X-Achse. Der Winkel des ersten Gliedes wird als  $\theta'_1$  bezeichnet und des zweiten Gliedes mit  $\theta'_2$ . Es muss die Position des Endeffektors berechnet werden, wo

$$\theta'_1, \theta'_2 \longrightarrow x_{EE}, y_{EE}$$

das verallgemeinert werden kann zu

$$f(\theta'_1, \theta'_2) = X_D = \begin{pmatrix} x_{EE} \\ y_{EE} \end{pmatrix}$$

wo  $X_D$  die Endposition der Endeffektors entspricht.

Anstatt alles auf einmal zu berechnen, wird zuerst die Endposition des ersten Gliedes berechnet. Es wird eine Vektorrotation  $R(\theta) = \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix}$  angewendet für die Rotation der einzelnen Glieder verwendet.

$$X_{D_1} = R(\theta'_1) L_1 = \begin{pmatrix} \cos \theta'_1 \\ \sin \theta'_1 \end{pmatrix} L_1 = \begin{pmatrix} \cos \theta'_1 \cdot L_1 \\ \sin \theta'_1 \cdot L_1 \end{pmatrix} = \begin{pmatrix} c_1 L_1 \\ s_1 L_1 \end{pmatrix}$$

Von diesem Punkt aus werden  $\cos \theta_n$  und  $\sin \theta_n$  als  $c_n$  und  $s_n$  vereinfacht.

$$\begin{aligned}\cos \theta_n &= c_n \\ \sin \theta_n &= s_n\end{aligned}$$

Durch das die Endposition des Endes vom ersten Glied bekannt ist, wird diese eingesetzt, um die Endposition des Endes vom zweiten Glied herauszufinden:

$$X_D = R(\theta'_1) L_1 + R(\theta'_2) L_2 = \begin{pmatrix} c_1 L_1 \\ s_1 L_1 \end{pmatrix} + \begin{pmatrix} c_2 L_2 \\ s_2 L_2 \end{pmatrix}$$

Jedoch ist der Fall nicht so, dass bewusst ist, unter welchem Winkel das zweite Glied relativ zur  $X$ -Achse steht, sondern nur relativ zu dem ersten Glied.

Dies ist so, da sich das Koordinatensystem für ein anhängendes Glied relativ zum treibenden Glied verändert. Die Rotation  $R(\theta_1)$  des ersten Gliedes entspricht somit zum neuen Koordinatensystem des zweiten Gliedes. Dadurch werden die beiden Rotationen  $R(\theta_1)$  und  $R(\theta_2)$  multipliziert. Aufgrund der Multiplikationsregeln von Rotationen kann  $R(\theta_1) \cdot R(\theta_2)$  als eine Addition von beiden Winkeln  $\theta_1$  und  $\theta_2$  umgeformt werden.

$$\begin{aligned}X_D &= R(\theta_1) L_1 + R(\theta_1) \cdot R(\theta_2) L_2 \\ X_D &= R(\theta_1) L_1 + R(\theta_1 + \theta_2) L_2\end{aligned}$$

Diese Multiplikation kann auch als eine rekursive Multiplikation-Formel umgeformt werden:

$$\begin{aligned}X_D &= R(\theta_1) L_1 + R(\theta_1) \cdot R(\theta_2) L_2 \\ X_D &= R(\theta_1) (L_1 + R(\theta_2) L_2)\end{aligned}$$

Durch diese Umformung kann für eine beliebige Anzahl Gliedern die Endposition des Endeffektors im 2-dimensionalen Raum berechnet werden. Hier noch ein Beispiel mit 3 Gliedern.

$$\theta_1, \theta_2, \theta_3 \longrightarrow x_{EE}, y_{EE}$$

Es kann jetzt entweder die Verkettungsformel benutzt werden, oder die Reihenformel benutzen.

Mit der Reihenformel (explizit):

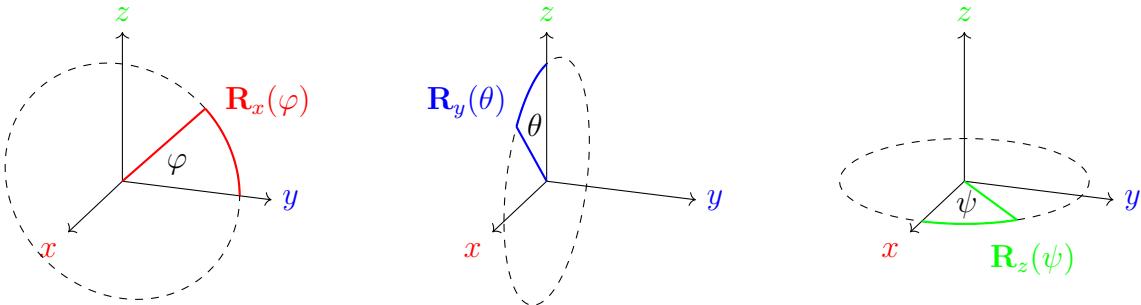
$$X_D = R(\theta_1) L_1 + R(\theta_1 + \theta_2) L_2 + R(\theta_1 + \theta_2 + \theta_3) L_3 \quad (6.1)$$

Oder mit der Verkettungsformel (rekursiv):

$$X_D = R(\theta_1)(L_1 + R(\theta_2)(L_2 + R(\theta_3)L_3)) \quad (6.2)$$

### 6.1.2. FK im 3-dimensionalem Raum

Dieses Vorgehen in der direkten Kinematik kann man auch in den 3-dimensionalen Raum umsetzen. Jedoch erscheinen durch das Hinzufügen von einer weiteren Achse zwei weitere Rotationsachsen. Somit sind neben den 3 verschiedenen Achsen entsprechend auch drei verschiedene Rotationsachsen vorhanden.



Jede von diesen Rotationsachsen verfügt über einer eigenen Winkelrotation.

X-Rotation:  $\varphi$

Y-Rotation:  $\theta$

Z-Rotation:  $\psi$

Dementsprechend verfügt jede Rotationsachse über eine Rotationsmatrix im 3-dimensionalem Raum.

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi \\ 0 & \sin \varphi & \cos \varphi \end{bmatrix} \quad (6.3)$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \quad (6.4)$$

$$R_z(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.5)$$

Mit den vereinfachten Begriffen:

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{bmatrix} \quad (6.6)$$

$$R_y(\theta) = \begin{bmatrix} c_y & 0 & s_y \\ 0 & 1 & 0 \\ -s_y & 0 & c_y \end{bmatrix} \quad (6.7)$$

$$R_z(\psi) = \begin{bmatrix} c_z & -s_z & 0 \\ s_z & c_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.8)$$

Diese drei Rotationsmatrizen können auch zu einer allgemeinen Rotationsmatrix zusammengesetzt werden.

$$R(\varphi, \theta, \psi) = R_z(\psi) \times R_y(\theta) \times R_x(\varphi) \quad (6.9)$$

$$= \begin{bmatrix} c_z & -s_z & 0 \\ s_z & c_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_y & 0 & s_y \\ 0 & 1 & 0 \\ -s_y & 0 & c_y \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{bmatrix} \quad (6.10)$$

$$R(\varphi, \theta, \psi) = \begin{bmatrix} c_z c_y & c_z s_y s_x - s_z c_x & c_z s_y c_x + s_z s_x \\ s_z c_y & s_z s_y s_x + c_z c_x & s_z s_y c_x - c_z s_x \\ -s_y & c_y s_x & c_y c_x \end{bmatrix} \quad (6.11)$$

Wichtig zu beachten ist, dass eine andere Multiplikationsreihenfolge der Rotationsmatrizen einer anderen Rotation entsprechen wird. Es können Phänomene wie «Gimble-Locking» entstehen.

Dementsprechend wirkt die Rotationsmatze auch anders, wenn eine andere Reihenfolge gewählt wird. Es wurde nur als Beispiel eine *ZYX*-Matrize geformt.

Diese kann somit gleich angewendet werden wie bei einer 2-dimensionellen Konstruktion. In diesem Fall werden die Glieder nur an bestimmten Rotationsachsen gedreht. Als Beispiel wird ein Glied ausgewählt, das parallel zur *Y*-Achse vom Nullpunkt ausgestreckt wird und um die Rotationsachse *Y* rotiert wird:

$$X_D = R(0, \theta_1, 0) \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix} = \begin{bmatrix} c_{y_1} & 0 & s_{y_1} \\ 0 & 1 & 0 \\ -s_{y_1} & 0 & c_{y_1} \end{bmatrix} \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix} \quad (6.12)$$

Da jetzt dieses Glied vom Nullpunkt auf der *Y*-Achse gestreckt wird, wirkt die Rotation in *Y*-Richtung nichts aus. Jedoch hat diese Rotation einen Einfluss auf die darauffolgenden

angehängten Glieder. Um das besser zu visualisieren, wird ein zweites Glied genommen, das an der  $X$ -Achse angelegt wird und an das Ende des ersten Gliedes angesetzt wird. Die Rotation des zweiten Gliedes erfolgt anhand der  $Z$ -Achse.

$$X_D = R(0, \theta_1, 0) \left( \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix} + R(0, 0, \psi_2) \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix} \right) \quad (6.13)$$

$$X_D = \begin{bmatrix} c_{y_1} & 0 & s_{y_1} \\ 0 & 1 & 0 \\ -s_{y_1} & 0 & c_{y_1} \end{bmatrix} \left( \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix} + \begin{bmatrix} c_{z_2} & -s_{z_2} & 0 \\ s_{z_2} & c_{z_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ L_2 \\ 0 \end{bmatrix} \right) \quad (6.14)$$

$$X_D = \begin{bmatrix} c_{y_1} & 0 & s_{y_1} \\ 0 & 1 & 0 \\ -s_{y_1} & 0 & c_{y_1} \end{bmatrix} \begin{bmatrix} -s_{z_2}L_2 \\ c_{z_2}L_2 + L_1 \\ 0 \end{bmatrix} \quad (6.15)$$

$$X_D = \begin{bmatrix} -c_{y_1}s_{z_2}L_2 \\ c_{z_2}L_2 + L_1 \\ s_{y_1}s_{z_2}L_2 \end{bmatrix} \quad (6.16)$$

Somit beeinflusst wie bei einer 2-dimensionalen Konstruktion jede Rotation des vorherigen Gliedes diese der Folgenden. In der folgenden Darstellung wird eine Bestimmung von  $X_D$  anhand des Aufbaus eines 6-Achsen Roboterarms, wo alle Glieder in die X-Richtung ausgestreckt werden.

$$X_D = R_x(\theta_1) \left( \begin{bmatrix} L_1 \\ 0 \\ 0 \end{bmatrix} + R_z(\theta_2) \left( \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix} + R_z(\theta_3) \left( \begin{bmatrix} L_3 \\ 0 \\ 0 \end{bmatrix} + R_z(\theta_4) \left( \begin{bmatrix} L_4 \\ 0 \\ 0 \end{bmatrix} + R_z(\theta_5) \left( \begin{bmatrix} L_5 \\ 0 \\ 0 \end{bmatrix} + R_z(\theta_6) \begin{bmatrix} L_6 \\ 0 \\ 0 \end{bmatrix} \right) \right) \right) \right) \quad (6.17)$$

Wichtig zu beachten ist, dass in dieser Darstellung ein Glied immer in  $X$ -Richtung ausgespannt wird. Grundsätzlich kann ein Glied nicht nur in  $X$ -Richtung ausgespannt werden, sondern auch in  $Y$ - oder  $Z$ -Richtung, sowie aus Kombination aus allen drei. Deshalb kann für eine Darstellung eines Gliedes für die Berechnung der Endposition  $X_D$  auch folgendeweise definiert werden

$$\begin{bmatrix} L_{n,x} \\ L_{n,y} \\ L_{n,z} \end{bmatrix}$$

wo  $n$  das entsprechende Glied ist.

## 6.2. Inverse Kinematik

Das Grundprinzip der inversen Kinematik, im Englischen auch bekannt als "*Inverse Kinematics*" oder "*IK*", besteht darin, dass wird eine Position und Rotation vorgegeben wird die vom Endeffektor erreicht werden muss, wodurch die verschiedenen Gelenkwinkel der einzelnen Glieder berechnet werden. Die inverse Kinematik ist deshalb sehr hilfreich, vor allem wenn man anhand einer Konstruktion einen Punkt  $A$  erreichen möchte und von diesem Punkt zu einem Punkt  $B$  bewegen möchte. Die Inverse Kinematik kann jedoch nicht nur auf Roboterarme limitiert werden, sondern werden auch in CAD-Softwares für die Simulation von verschiedenen Konstruktionen häufig benutzt.

### 6.2.1. 1R-Manipulator

Um die Funktionsweise der inversen Kinematik richtig verstehen zu können, ist es am besten zuerst zu verstehen, wenn es nur ein Glied zu Verfügung hat, die um eine Rotationsachse rotiert wird. Bei einem 1R-Manipulatoren wird von einer Konstruktion aus einem Glied gesprochen, dass auf einer Achse rotiert wird. Für solch eine Konstruktion aus einem Glied in einem 2-dimensionalem, ist dies ziemlich einfach zu berechnen, indem die Rotation relativ zur Endposition  $x_D = (x_D, y_D)$  zum Nullpunkt anhand des Pythagoras-Theorems berechnet wird.

$$\theta = \text{atan2}(y_D, x_D) = \tan^{-1} \frac{y_D}{x_D}$$

Dies führt zu einer einzigen Limitation, dass der Betrag der Endposition gleich zur Länge des Gliedes ist:

$$L_1 = |x_D| = \sqrt{x_D^2 + y_D^2}$$

Grundsätzlich gehört dieser Prozess nicht direkt zur inversen Kinematik, jedoch ist dies ein Grundsatz für jedes Problem in der inversen Kinematik.

### 6.2.2. 2R-Manipulator

Bei einem 2R-Manipulator wird über einer Konstruktion gesprochen, die aus 2 Gliedern besteht. Das erste Glied wird am Nullpunkt rotiert, während das zweite Glied um die Länge des zweiten Gliedes  $L_1$  versetzt wird, wodurch der Endeffektor um die Länge des zweiten Gliedes  $L_2$  versetzt wird. Es wird über einem 2R-Manipulator im 2-dimensionalem Raum gesprochen. Hingegen zum 1R-Manipulator sind 2 verschiedene Glieder vorhanden, die sich unabhängig voneinander drehen können. Somit verfügt das erste Glied über eine Rotation  $\theta_1$  und das zweite Glied über eine Rotation  $\theta_2$ . Durch das 2 einzelne Winkel vorhanden sind, ist deren Berechnung anhand einer gewünschten Position nicht mehr übersichtlich. Deshalb wird der Prozess der inversen Kinematik angewandt. Um die Winkel  $\theta_1$  und  $\theta_2$  berechnen zu können, wird zuerst von der Gleichung aus der direkten Kinematik

ausgegangen:

$$X_D = R(\theta_1) \left( \begin{bmatrix} L_1 \\ 0 \end{bmatrix} + R(\theta_2) \begin{bmatrix} L_2 \\ 0 \end{bmatrix} \right).$$

Anstatt den Winkel der einzelnen Glieder auf einmal zu berechnen, wird schrittweise vorgegangen. Um dies zu verwirklichen, stellt man das erste Glied unter keinem Winkel, also  $\theta_1 = 0$ . Dadurch dass das erste Glied stationär ist, wird davon ausgegangen, dass das zweite Glied so positioniert werden sollte, dass sich der Endeffektor auf einem Kreis mit dem Radius der Länge des Betrags der Endposition befinden müsste. Sobald dies erfüllt wäre, könnte man das erste Glied entsprechend so rotieren, dass der Endeffektor die Endposition erreicht.

Somit entsteht eine neue Gleichung:

$$\begin{aligned} X_D &= R(0) \left( \begin{bmatrix} L_1 \\ 0 \end{bmatrix} + R(\theta_2) \begin{bmatrix} L_2 \\ 0 \end{bmatrix} \right) \\ X_D &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \left( \begin{bmatrix} L_1 \\ 0 \end{bmatrix} + \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \begin{bmatrix} L_2 \\ 0 \end{bmatrix} \right) \\ X_D &= \begin{bmatrix} c_2 L_2 + L_1 \\ s_2 L_2 \end{bmatrix} \end{aligned}$$

Durch das jetzt den Winkel  $\theta_1$  berechnet werden muss, wird der Betrag von  $X_D$  genommen und nach  $\theta_1$  aufgelöst:

$$\begin{aligned} |X_D| &= \sqrt{(c_2 L_2 + L_1)^2 + (s_2 L_2)^2} \\ |X_D|^2 &= (c_2 L_2 + L_1)^2 + (s_2 L_2)^2 \\ |X_D|^2 &= c_2^2 L_2^2 + 2c_2 L_2 L_1 + L_1^2 + s_2^2 L_2^2 \end{aligned}$$

$$\begin{aligned} \cos \alpha \cdot \cos \beta \mp \sin \alpha \cdot \sin \beta &= \cos(\alpha \pm \beta) \\ \Rightarrow \cos \alpha \cdot \cos \alpha + \sin \alpha \cdot \sin \alpha &= \cos(\alpha - \alpha) = 1 \end{aligned}$$

$$\begin{aligned}
|X_D|^2 &= L_2^2 (c_2^2 + s_2^2) + L_1^2 + 2c_2L_2L_1 \\
|X_D|^2 &= L_2^2 + L_1^2 + 2c_2L_2L_1 \\
2c_2L_2L_1 &= |X_D|^2 - L_2^2 - L_1^2 \\
c_2 &= \frac{|X_D|^2 - L_2^2 - L_1^2}{2L_2L_1} \\
\theta_2 &= \pm \cos^{-1} \left( \frac{|X_D|^2 - L_2^2 - L_1^2}{2L_2L_1} \right)
\end{aligned}$$

Da durch die umgekehrte Kosinus-Funktion an zwei verschiedenen Stellen den gleichen Wert herausgeben wird, werden auch zwei Werte für  $\theta_2$  erhalten

$$\begin{aligned}
\theta_{2,1} &= \cos^{-1} \left( \frac{|X_D|^2 - L_2^2 - L_1^2}{2L_2L_1} \right) \\
\theta_{2,2} &= -\cos^{-1} \left( \frac{|X_D|^2 - L_2^2 - L_1^2}{2L_2L_1} \right)
\end{aligned}$$

Durch das  $\theta_2$  bekannt ist, kann  $\theta_1$  dementsprechend auch berechnet werden, indem zuerst der Winkel vom Ursprung zum jetzigen Endeffektor berechnet wird und dieser vom Winkel vom Ursprung zur Endposition subtrahiert wird.

$$\begin{aligned}
\theta &= \text{atan2}(X_y, X_x) = \tan^{-1} \left( \frac{X_y}{X_x} \right) \\
\theta_k &= \text{atan2}(s_2L_2, c_2L_2 + L_1) = \tan^{-1} \left( \frac{s_2L_2}{c_2L_2 + L_1} \right) \\
\theta_{1,k} &= \theta - \theta_k
\end{aligned}$$

Da es zwei Lösungen für  $\theta_2$  gibt, entstehen zwei weitere Lösungen für  $\theta_1$ . Somit entstehen die Lösungspaare  $\{\theta_{1,1}, \theta_{2,1}\}$  und  $\{\theta_{1,2}, \theta_{2,2}\}$

Da es zwei verschiedene Lösungen gibt, kann die eine oder andere für den Lösungsansatz verwendet werden. Jedoch ist für den Prozess der inversen Kinematik die Berechnung aller möglichen Lösungen kritisch. Entsprechend ergeben sich Limitation für die Berechnung der Winkel  $\theta_1$  und  $\theta_2$ . Der erste Fall, in dem keine Lösung auffallen würde, wäre, wenn in der Berechnung von  $\theta_2$  die Berechnung  $\frac{|X_D|^2 - L_2^2 - L_1^2}{2L_2L_1} > 1$  wäre, somit  $|X_D|^2 - L_2^2 - L_1^2 > 2L_2L_1$  oder anders gesagt:

$$|X_D|^2 > L_1^2 + 2L_2L_1 + L_2^2 = (L_1 + L_2)^2$$

was bestimmt, dass der zu erreichende Punkt ausserhalb der Reichweite des Manipulators ist. Der zweite Fall wäre, wo die Berechnung von  $\theta_2 < -1$  wäre, somit  $|X_D|^2 - L_2^2 - L_1^2 < 2L_2L_1$ ,

oder anders gesagt:

$$|X_D|^2 > L_1^2 - 2L_2L_1 + L_2^2 = (L_1 - L_2)^2$$

Damit dies möglich sein kann, müssen die Längen der beiden Glieder ungleich sein, wo der Endpunkt innerhalb eines Kreises mit einem Radius von  $|L_1 - L_2|$  am Ursprungspunkt wäre.

### 6.2.3. 3R-Manipulator

Wenn man diesen Vorgang in die 3-dimensionale umsetzen würde, wäre der Prozess der indirekten Kinematik ähnlich zu der 2-dimensionalen. Um einen Punkt im 3-dimensionalen Raum erreichen zu können, bräuchte es mindestens ein weiteres Glied, das sich um die  $Y$ -Achse rotieren würde. Somit würde die optimale Konstruktion aus 3 Gliedern bestehen. Bei einem 3R Manipulatoren wird das erste Glied um die  $Z$ -Achse rotiert, das zweite Glied um die  $Y$ -Achse rotiert mit einer Versetzung von  $L_1$  in die  $Z$ -Richtung und das dritte Glied nochmals um die  $Y$ -Achse mit einer Versetzung von  $L_2$  in die  $X$ -Richtung. Der Endeffektor wird vom dritten Glied über eine Distanz von  $L_3$  in die  $X$ -Richtung versetzt. Mit dieser Definition können die Koordinaten des Endeffektors festgestellt werden:

$$x(\theta) = R_z(\theta_1) \left( \begin{bmatrix} 0 \\ 0 \\ L_1 \end{bmatrix} + R_y(\theta_2) \left( \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix} + R_y(\theta_3) \begin{bmatrix} L_3 \\ 0 \\ 0 \end{bmatrix} \right) \right)$$

$$x(\theta) = \begin{bmatrix} c_1(c_2(L_2 + c_3l_3) - s_2s_3L_3) \\ s_1(c_2(L_2 + c_3l_3) - s_2s_3L_3) \\ L_1 - s_2(L_2 + c_3l_3) - c_2s_3L_3 \end{bmatrix}$$

Ziel ist es diesen Endeffektor zu der Koordinate  $X_D = \begin{bmatrix} x_D \\ y_D \\ z_D \end{bmatrix}$  zu bringen.

Das Vorgehen bei einem Arm mit drei Rotationselementen ist sehr ähnlich, ausser, dass vorerst die Rotation des ersten Gliedes bestimmt werden muss. Anhand der Rotation des ersten Gliedes wird es möglich die Endposition auf eine Fläche projektieren zu können, die entlang zu der Rotation der Konstruktion verläuft. Somit ist die Rotation des ersten Gliedes

$$\theta_1 = \tan^{-1}\left(\frac{y_D}{x_D}\right)$$

Dementsprechend verfügt  $\theta_1$  über eine weitere Rotation, die gegenüber des ersten Punktes liegt:

$$\theta_1 = \tan^{-1}\left(\frac{y_D}{x_D}\right) + \pi$$

Durch die Rotation des ersten Gliedes werden die zwei weiteren Glieder mitrotiert, wodurch die Endposition  $X_D$  relativ zum ersten Glied auch mitrotiert wird. Somit erfolgt eine neue Endposition

$$R(0, 0, \theta_1) x_D = \begin{bmatrix} c_1 & -s_2 & 0 \\ s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_D \\ y_D \\ z_D \end{bmatrix} = \begin{bmatrix} c_1 x_D \\ s_1 y_D \\ z_D \end{bmatrix}.$$

Anhand dieser neuen Endposition wird der Winkel der letzten zwei Glieder berechnen, genauso wie bei einem 2R Manipulator. Somit ist die Endposition  $X_G$  für die zwei Glieder auf einer 2-dimensionalen Ebene

$$X_G = \begin{bmatrix} \sqrt{(c_1 x_N)^2 + (s_1 y_N)^2} \\ z_N \end{bmatrix}$$

Diese neue Endposition wird in die folgende Formel einsetzen:

$$\theta_3 = \cos^{-1} \left( \frac{|X_G|^2 - L_2^2 - L_1^2}{2L_2 L_1} \right)$$

Dadurch wird dann auch der Winkel  $\theta_2$  berechnet:

$$\begin{aligned} \theta_D &= \tan^{-1} \left( \frac{y_D - L_1}{\sqrt{x_D^2 + z_D^2}} \right) \\ \theta_k &= \tan^{-1} \left( \frac{s_2 L_2}{c_2 L_2 + L_1} \right) \\ \theta_{2,k} &= \theta_D - \theta_k \end{aligned}$$

#### 6.2.4. 6R-Manipulator

Die meisten industriellen 6R Manipulatoren sind aus einem Ellbogengelenk (Rotation um die  $Z$ -Achse), einer Schulter bestehend aus 2 sich orthogonal auftreffenden Achsen und einem Handgelenk mit weiteren 3 sich orthogonal auftreffenden Achsen. In dieser Ordnung ist die Konstruktion des Roboterarms in dieser Arbeit gegliedert

Der Endeffektor für einen industriellen Manipulatoren wird auch als ein Werkzeugmittelpunkt oder "Tool Center Point" genannt. Da der Ursprung des letzten Gliedes innerhalb des Handgelenks ist, wird von der Werkzeugmittelpunkt durch eine lokale Koordinate  $X_6$  mit der entsprechenden Verstezung zum letzten Glied definiert. Dadurch wird es möglich die Weltkoordinaten vom Werkzeugmittelpunkt zu ermitteln

$$X_{tcp}(q) = T_6(q) X_6$$

wo  $X_{tcp}$  der Werkzeugmittelpunkt in den Weltkoordinaten ist und  $T_6$  die entsprechende Rotation Versetzungen der verschiedenen Glieder sind. In diesem Fall wird verlangt, dass

der Werkzeugmittelpunkt  $X_{tcp}$  zu der verlangten Position  $X_D$  gestellt wird und das letzte Glied über eine Rotation  $R_D$  verfügen sollte.

Von hier aus kann klargestellt werden, dass man mit der Rotation von allen sechs Gliedern schlussendlich die entsprechende Rotation erreichen muss

$$R_6(q) = R_D$$

Dadurch entsteht die Gleichung

$$X_D = X_{tcp}(q) = R_6(q) X_6 + T_6(q)$$

wo  $T_6(q)$  die Position des Handgelenks ist. Somit also  $T_6(q) = X_D - R_D X_6$ .

Schlussendlich kann festgestellt werden, dass das Handgelenk nur durch die ersten drei Glieder und deren Rotationen beeinflusst wird:  $\theta_1$ ,  $\theta_2$  und  $\theta_3$ . Dadurch wird ersichtlich, dass grundsätzlich nur ein 3R-IK problem gelöst wird und dann schlussendlich nach  $x_D - R_D x_6$  gesucht wird. Es werden somit 4 Lösungen für  $(\theta_1, \theta_2, \theta_3)$  zusammengestellt. Entsprechend zu wie der Roboterarm aufgebaut wird, wird nur eine Lösungsvariante angewandt. Durch das jetzt die 3 Rotationen für die ersten drei Glieder vorhanden sind, wird die Rotation vom dritten Glied bestimmt

$$\begin{aligned} R_3 &= R_y(\theta_1) R_z(\theta_2) R_z(\theta_3) \\ &= \begin{bmatrix} c_1 & 0 & -s_1 \\ 0 & 1 & 0 \\ s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 \\ s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & 0 & -s_3 \\ 0 & 1 & 0 \\ s_3 & 0 & c_3 \end{bmatrix} \\ R_3 &= \begin{bmatrix} c_1 c_2 c_3 - c_1 s_2 s_3 & -c_1 c_2 s_3 - c_1 s_2 c_3 & -s_1 \\ s_2 c_3 + c_2 s_3 & -s_2 s_3 + c_2 c_3 & 0 \\ s_1 c_2 c_3 - s_1 s_2 s_3 & -s_1 c_2 s_3 - s_1 s_2 c_3 & c_1 \end{bmatrix} \\ R_3^T &= R_z(-\theta_3) R_z(-\theta_2) R_y(-\theta_1) \end{aligned}$$

$$R_6 = R_3(\theta_1, \theta_2, \theta_3) R_x(\theta_4) R_y(\theta_5) R_x(\theta_6) = R_D$$

Durch das, dass die Rotation von  $R_3$  vorhanden, kann festgestellt werden, dass die Endrotation  $R_D$  durch das transponierte  $R_3^T$  zurückgedreht werden kann um dann die Rotation  $R_3^6$  der letzten drei Glieder herausfinden zu können.

$$R_3^6 = R_x(\theta_4) R_y(\theta_5) R_x(\theta_6) = R_3^T(\theta_1, \theta_2, \theta_3) R_D$$

Somit entsteht eine  $XYX$ -Konversionsproblem der Eulerschen Winkel, da die Rotationen der letzten drei Winkel anhand der  $X$ -Achse,  $Y$ -Achse und nochmals der  $X$ -Achse erfolgen

$$R_3^6 = R_x(\theta_4) R_y(\theta_5) R_x(\theta_6)$$

Transponierte Matrizen

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \in K^{m \times n}$$

$$A^T = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix} \in K^{n \times m}$$

Transponierte Rotationsmatrizen

$$R_z(\theta) = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z^T(\theta) = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z^T(\theta) R_z(\theta) = \begin{bmatrix} c^2 + s^2 & cs - cs & 0 \\ cs - cs & c^2 + s^2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\cos^2(\theta) + \sin^2(\theta) = 1$$

Durch die Vormultiplikation von  $R_3^6$  wird eine folgende Rotationsmatrize geformt

$$R_3^6 = R_x(\theta_4) R_y(\theta_5) R_x(\theta_6)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_4 & -s_4 \\ 0 & s_4 & c_4 \end{bmatrix} \begin{bmatrix} c_5 & 0 & s_5 \\ 0 & 1 & 0 \\ -s_5 & 0 & c_5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_6 & -s_6 \\ 0 & s_6 & c_6 \end{bmatrix}$$

$$R_3^6 = \begin{bmatrix} c_5 & s_5 s_6 & s_5 c_6 \\ s_4 s_5 & c_4 c_6 - s_4 c_5 s_6 & -c_4 s_6 - s_4 c_5 c_6 \\ -c_4 s_5 & s_4 c_6 + c_4 c_5 s_6 & -s_4 s_6 + c_4 c_5 c_6 \end{bmatrix}$$

Zusätzlich wird die Rotationsmatrix der Endrotation folgenderweise bezeichnet

$$R_3^T(\theta_1, \theta_2, \theta_3) R_D = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Durch das jetzt bewusst ist, wie die Rotation der letzten 3 Glieder aufgebaut werden sollte und wie die Rotationsmatrix der Endrotation mit der transponierten Rotationsmatrix der ersten drei Glieder aussieht, können diese gleichgesetzt werden

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c_5 & s_5s_6 & s_5c_6 \\ s_4s_5 & c_4c_6 - s_4c_5s_6 & -c_4s_6 - s_4c_5c_6 \\ -c_4s_5 & s_4c_6 + c_4c_5s_6 & -s_4s_6 + c_4c_5c_6 \end{bmatrix}$$

Somit stellt sich heraus, dass der Winkel  $\theta_5$  durch  $r_{11} = c_5$  bestimmt werden kann

$$\theta_5 = \pm \cos^{-1}(r_{11})$$

Mit einem bekannten Wert für  $\theta_5$  können zusätzlich  $\theta_4$  und  $\theta_6$  festgestellt werden

$$\begin{aligned} \theta_4 &= \text{atan2}(r_{21}, -r_{31}) = \tan^{-1}\left(\frac{r_{21}}{-r_{31}}\right) \\ \theta_6 &= \text{atan2}(r_{12}, r_{13}) = \tan^{-1}\left(\frac{r_{12}}{r_{13}}\right) \end{aligned}$$

Es gibt eine einzige Ausnahme in dieser Situation, die zu einer Singularität entspricht. Diese entsteht, wenn  $r_{11} = 1$  entspricht. Da

$$r_{11} = 1 \Rightarrow \cos^{-1} 1 = 0 \Rightarrow \theta_5 = 0$$

Somit ist der Fall, dass die Winkel  $\theta_4$  und  $\theta_6$  nicht berechenbar sind, da

$$\sin \theta_5 = 0$$

Dadurch ergibt sich eine unendliche Anzahl Lösungen für

$$\theta_4 + \theta_6 = \tan^{-1} \frac{r_{23}}{r_{22}}$$

Durch das aber die Rotation  $\theta_4$  und  $\theta_6$  auf der  $X$ -Achse erfolgt, kann bestimmt werden, dass nur das letzte Glied rotiert wird, somit also  $\theta_4 = 0$ . Dadurch ergibt sich:

$$\begin{aligned}\theta_4 &= 0 \\ \theta_5 &= 0 \\ \theta_6 &= \cos^{-1} r_{22}\end{aligned}$$

### 6.3. Theoretische Umsetzung eines 6R-Manipulatoren mit IK

Durch das die Funktionsweise der Inversen Kinematik bekannt ist, wird es möglich den ganzen Prozess zur Berechnung der Gelenkweinkel eines 6R-Manipulatoren zu verallgemeinern. Für die Berechnung von  $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$  werden die Eingaben  $\{x, y, z, \varphi, \theta, \psi, L_1, L_2, L_3, L_4\}$  wo  $L_1$  die Länge der Schulter,  $L_2$  die Länge des Oberarms,  $L_3$  die gesamte Länge vom Anstzpunkt des Oberarms bis zu Handgelenk und  $L_4$  die Länge des Handgelenks ist. Somit ist definiert:

Eingaben:

$$x, y, z, \varphi, \theta, \psi, L_1, L_2, L_3, L_4$$

Ausgaben:

$$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$$

Durch die Eingaben wird zuerst eine zu erreichende Rotationsmatrix  $R_D$  geformt

$$R_D = R_z(\psi) R_y(\theta) R_x(\varphi)$$

Danach wird festgesetzt, was die Endposition  $X_D$  ist

$$X_D = \begin{bmatrix} x_D \\ y_D \\ z_D \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} - R_D \begin{bmatrix} L_4 \\ 0 \\ 0 \end{bmatrix}.$$

Dadurch können die jeglichen Berechnungen der Gelenkwinkel folgende Werte berechnet werden:

$$\begin{aligned} \theta_1 &= \tan^{-1} \left( \frac{y_D}{x_D} \right) = \text{atan2}(y_D, x_D) \\ \theta_2 &= \tan^{-1} \left( \frac{y_D - L_1}{\sqrt{x_D^2 + z_D^2}} \right) - \tan^{-1} \left( \frac{L_2 \sin \theta_2}{L_1 + L_2 \cos \theta_2} \right) \\ &= \text{atan2} \left( y_D - L_1, \sqrt{x_D^2 + z_D^2} \right) - \text{atan2}(L_2 \sin \theta_2, L_1 + L_2 \cos \theta_2) \\ \theta_3 &= \pm \cos^{-1} \left( \frac{(x_D^2 + z_D^2 + (z_D - L_1)^2) - L_1^2 - L_2^2}{2L_1 L_2} \right) \end{aligned}$$

$$R_3 = R_y(\theta_1) R_z(\theta_2) R_z(\theta_3)$$

$$R_3^T = R_z(-\theta_3) R_z(-\theta_2) R_y(-\theta_1)$$

$$R_3^T R_D = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\theta_4 = \text{atan2}(r_{21}, -r_{31}) = \tan^{-1} \frac{r_{21}}{-r_{31}}$$

$$\theta_5 = \pm \cos^{-1} r_{11}$$

$$\theta_6 = \text{atan2}(r_{12}, r_{13}) = \tan^{-1} \frac{r_{12}}{r_{13}}$$

Im Fall einer Singularität mit  $r_{11} = 1$  bzw.  $\theta_5 = 0$ , dann

$$\theta_4 = 0$$

$$\theta_5 = 0$$

$$\theta_6 = \cos^{-1} r_{22}$$

## 6.4. Praktische Umsetzung eines 6R-Manipulatoren mit IK

Code 6.1: Berechnung anhand Ik von einem 6R-Manipulatoren (*Python*)

```
1 import numpy as np
2 from numpy import cos as c, sin as s
3 def roll_rotation(roll): # X-Rotation
4     return np.matrix([[1, 0, 0], [0, c(roll), -s(roll)], [0, s(roll), c(roll)]])
5
6 def pitch_rotation(pitch): # Y-Rotation
7     return np.matrix([[c(pitch), 0, s(pitch)], [0, 1, 0], [-s(pitch), 0, c(pitch)]]]
8
9 def yaw_rotation(yaw) -> np.matrix: # Z-Rotation
10    return np.matrix([[c(yaw), -s(yaw), 0], [s(yaw), c(yaw), 0], [0, 0, 1]])
11
12 def inverse_kinematics(x, y, z, rx, ry, rz, l1, l2, l3, l4):
13     R_D = roll_rotation(rx) * pitch_rotation(ry) * yaw_rotation(rz)
14     X_D = np.matrix([x, y, z]) - np.matrix([l4, 0, 0]) * R_D
15     xd, yd, zd = X_D.tolist()[0]
16
17
18     n_X_D = np.sqrt(np.pow(xd, 2) + np.pow(yd - l2, 2) + np.pow(zd, 2))
19     q1 = np.arctan2(yd, xd)
20     solutions_1 = [[q1], [q1 + np.pi]]
21     c3 = (np.pow(xd, 2) + np.pow(zd, 2) + np.pow(yd - l1, 2) - np.pow(l2, 2) - np.
22     pow(l3, 2))/(2 * l2 * l3)
23     if c3 > 0:
24         return None
25     elif c3 == 1:
26         q2 = np.atan2(yd - l1, np.hypot(xd, zd))
27         q3 = 0
28         solutions_2 = [[q2, q3]]
29     elif c3 == -1 and n_X_D != 0:
30         q2 = np.atan2(yd - l1, np.hypot(xd, zd))
31         q3 = np.pi
32         solutions_2 = [[q2, q3]]
33     elif c3 == -1 and n_X_D == 0:
34         q2 = 0
35         q3 = np.pi
36         solutions_2 = [[q2, q3]]
37     else:
38         q3_1 = np.arccos(c3)
39         q2_1 = np.atan2(yd - l1, np.hypot(xd, zd)) - np.atan2(l3 * np.sin(q3_1), l2
+ l3 * np.cos(q3_1))
```

```

39     q3_2 = -np.arccos(c3)
40     q2_2 = np.atan2(yd - l1, np.hypot(xd, zd)) - np.atan2(l3 * np.sin(q3_2), l2
+ l3 * np.cos(q3_2))
41     solutions_2 = [[q2_1, q3_1], [q2_2, q3_2]]
42     solutions_3 = []
43     for i in solutions_1:
44         for j in solutions_2:
45             R_3 = pitch_rotation(i[0]) * yaw_rotation(j[0]) * yaw_rotation(j[1])
46             R_N = R_D * R_3.transpose()
47             if R_N.item(0, 0) == 1:
48                 q4 = 0
49                 q5 = 0
50                 q6 = np.arccos(R_N.item(1, 1))
51                 solutions_3 = [[q4, q5, q6], [q4, q5, -q6]]
52             else:
53                 q4 = np.atan2(R_N.item(1, 0), -R_N.item(2, 0))
54                 q5 = np.arccos(R_N.item(0, 0))
55                 q6 = np.atan2(R_N.item(0, 1), R_N.item(0, 2))
56                 solutions_3.append([q4, -q5, q6])
57                 solutions_3.append([q4, q5, q6])
58
59     solutions = []
60     for i in solutions_1:
61         for j in solutions_2:
62             for k in solutions_3:
63                 solutions.append(i + j + k)
64
65 return solutions

```

---

# **7. Resultate und Diskussion**

## **7.1. Komplette Konstruktion**

## **7.2. Auswertung**

### **7.2.1. Geschwindigkeit**

### **7.2.2. Genauigkeit und Wiederholbarkeit**

# **8. Zusammenfassung und Ausblicke**

## **8.1. Zukünftige Ergänzungen**

### **8.1.1. Zusätzliche Funktionalitäten**

### **8.1.2. Interpolation**

### **8.1.3. FPGA**

### **8.1.4. Einbettung von KI**

## **8.2. Reflexion**

## **9. Quellenverzeichnis**

# **10. Redlichkeitserklärung**

Ich, Vladimir Morozov, 4A,

bestätige mit meiner Unterschrift, dass die eingereichte Arbeit selbstständig und ohne unerlaubte Hilfe Dritter verfasst wurde. Die Auseinandersetzung mit dem Thema erfolgte ausschliesslich durch meine persönliche Arbeit und Recherche. Es wurden keine unerlaubten Hilfsmittel benutzt.

Ich bestätige, dass ich sämtliche verwendeten Quellen sowie Informanten/-innen im Quellenverzeichnis bzw. an anderer da-für vorgesehener Stelle vollständig aufgeführt habe. Alle Zitate und Paraphrasen (indirekte Zitate) wurden gekennzeichnet und belegt. Sofern ich Informationen von einem KI-System wie bspw. ChatGPT verwendet habe, habe ich diese in meiner Maturaarbeit gemäss den Vorgaben im Leitfaden zur Maturaarbeit korrekt als solche gekennzeichnet, einschliesslich der Art und Weise, wie und mit welchen Fragen die KI verwendet wurde.

Ich bestätige, dass das ausgedruckte Exemplar der Maturaarbeit identisch mit der digitalen Version ist.

Ich bin mir bewusst, dass die ganze Arbeit oder Teile davon mittels geeigneter Software zur Erkennung von Plagiaten oder KI-Textstellen einer Kontrolle unterzogen werden können.

## **A. Anhang**