

Hochschule Furtwangen University

Dynamische Auslegung von Werkzeugmaschinen, Robotern und Bewegungsachsen

Manuel Caipo

M.Sc. Advanced Precision Engineering
Matrikelnummer: 286577

Dozent: Prof. Dr.-Ing. Gunter Ketterer

26. Juni 2025 - 78056 Villingen-Schwenningen

Inhaltsverzeichnis

1 Einleitung	1
2 Aufgabe 1: Arbeitsraum, Kinematik und Inverse Kinematik	2
2.1 1-a Arbeitsraum und Kollisionsvermeidung	2
2.1.1 Arbeitsraum	2
2.1.2 Kollisionsvermeidung	6
2.2 1-b Schematisches Ersatzbild des Roboters mit DH Koordinatensystemen	8
2.3 1-c Transformationsmatrizen der Gelenke	9
2.3.1 Transformationsmatrizen der Gelenke	10
2.4 1-d Gesamte Transformation	12
2.4.1 Flussdiagramm und MATLAB-Code für die Transformation	13
2.4.2 Matlab Code für die Transformation	13
2.5 1-e Berechnung der inversen Dynamik des SCARA-Roboters	15
2.5.1 Algebraische Methode	15
2.5.2 Geometrische Methode	17
2.5.3 Ergebnisse des Kinematischen Modells durch die Kombination von geometrische und algebraische Methode	18
2.5.4 Flussdiagramm für die Inverse Kinematik in Matlab	23
2.5.5 Matlab Code Inverse Kinematics	25
2.6 1-f Einfluss thermischer Längenänderungen auf die Endeffektorstellung und notwendige Achskorrekturen	38
2.6.1 Matlab Code Einfluss thermischer Längenänderungen auf die Endeffektorstellung	40
3 Aufgabe 2: Jacobi-Matrix und Kräfteanalyse	44
3.1 Ableitung der Jacobi-Matrix	44
3.1.1 Winkelgeschwindigkeit	45
3.1.2 Lineare Geschwindigkeit	45
3.1.3 Kombination des linearen und des Winkelgeschwindigkeits-Jacobians	47
3.2 Berechnung der Jacobi-Matrix	49
3.2.1 Transformationsmatrizen und Gelenkvariablen	50
3.2.2 Berechnung pro Gelenk	57
3.2.3 Jacobi-Matrix des Roboters	63
3.3 Singularitäten	65
3.3.1 Singulärwertzerlegung (SVD)	65
3.3.2 Ermittlung von Singularitäten	66
3.3.3 Berechnung von Singularitäten	67
3.3.4 Zusammenfassung der Singularitätsanalyse mit Singulärwertzerlegung (SVD)	71
3.4 Berechnung der Gelenkkräfte und -momente	72
3.5 Matlab Code Jacobi:	74

4 Aufgabe 3: Bestimmung der Lagrange-Dynamik	84
4.1 Theoretische Grundlagen des Lagrange-Verfahrens	84
4.1.1 Implementierungshinweise	86
4.2 Berechnung der Lagrange-Gleichung	86
4.2.1 Berechnung der Jacobi-Matrizen bezogen auf den Massenmittelpunkt jedes Körpers	86
4.2.2 Trägheitsmatrizen und deren Transformation durch Rotation	92
4.2.3 Gesamte Berechnung der Massenmatrix $M(q)$	100
4.2.4 Coriolis- und Zentrifugal-Matrix	101
4.2.5 Berechnung des Gravitationsvektors $G(q)$	103
4.2.6 Numerische Auswertung der Lagrange-Gleichungen während der ersten zwei Sekunden	105
4.3 Bestimmung der Bewegungsprofile (Position, Geschwindigkeit, Beschleunigung und Ruck)	108
4.3.1 Ergebnisdarstellung der Bewegungsprofile	109
4.3.2 Auswertung von $\tau(t)$ über die Zeit – Lagrange ausgewertet mit Bewegungsprofilen	115
4.4 Python script für Bewegungsprofile	121
4.5 Numerische Auswertung der Lagrange-Gleichungen mit MATLAB	127
4.6 Symbolische Auswertung der Lagrange-Gleichungen mit MATLAB	136
5 Fazit	151

Abbildungsverzeichnis

2.1	Isometrische Ansichten der Modelle mit Autodesk Inventor	2
2.2	Zx-Ansicht und Xy-Ansicht mit Robot Operating System (ROS2)	3
2.3	Die spezifischen Maße des SCARA-Roboters	3
2.4	3D Arbeitsraum durch Inverse Kinematik	4
2.5	XY Arbeitsraum durch Inverse Kinematik	5
2.6	YZ Arbeitsraum durch Inverse Kinematik	5
2.7	Arbeitsraum und generierte Punkte für die Oberflächengenerierung	5
2.8	Kollision 1	6
2.9	Kollisionen 2-3	7
2.10	Kollision 4	7
2.11	Schematisches Ersatzbild des Roboters	8
2.12	Resultierende Grafik des Systems mit der Robotics-Bibliothek von MATLAB	8
2.13	Resultierende Grafik des Systems mit der Robotics-Bibliothek von MATLAB 2	9
2.14	Matlab Flowchart des Erzeugens DH-Matrix	13
2.15	Geometrische Analyse der inversen Roboterdynamik.	17
2.16	Geometrische Analyse der inversen Roboterdynamik.	18
2.17	Lösung Nummer 1: 3D-Ansicht und Projektionen	21
2.18	Lösung Nummer 2: 3D-Ansicht und Projektionen	22
2.19	Flussdiagramm für die Berechnung mit einer Zielvariable	23
2.20	Flussdiagramm für die Berechnung mit zwei Zielvariablen	23
2.21	Flussdiagramm für die Berechnung mit drei Zielvariablen	24
3.1	Bewegung des Endeffektors durch prismatisches Gelenk	46
3.2	Bewegung des Endeffektors durch rotatorisches Gelenk	47
3.3	Singularitätsprüfung für Konfiguration 1	67
3.4	Singularitätsprüfung für Konfiguration 2	69
3.5	Singularitätsprüfung für Konfiguration 3	70
4.1	Glied 2 CAD	93
4.2	Glied 2 CAD Eigenschaften	93
4.3	Glied 4 CAD	94
4.4	Glied 4 CAD Eigenschaften	94
4.5	Entscheidungsdiagramm	109
4.6	Bewegungsprofile φ_0	110
4.7	Bewegungsprofile φ_1	111
4.8	Bewegungsprofile φ_2	112
4.9	Bewegungsprofile dh_3	113
4.10	Bewegungsprofile φ_4	114
4.11	Generalisierten Gelenkmomente $\varphi_0 - \tau$	115
4.12	Generalisierten Gelenkmomente $\varphi_1 - \tau$	116
4.13	Generalisierten Gelenkmomente $\varphi_2 - \tau$	116

ABBILDUNGSVERZEICHNIS

4.14 Generalisierten Gelenkkraft $dh_3 - \tau$	117
4.15 Generalisierten Gelenkmomente $\varphi_4 - \tau$	117
4.16 Generalisierten Gelenkmomente $\varphi_0 - \tau$	118
4.17 Generalisierten Gelenkmomente $\varphi_1 - \tau$	118
4.18 Generalisierten Gelenkmomente $\varphi_2 - \tau$	119
4.19 Generalisierten Gelenkkraft $dh_3 - \tau$	119
4.20 Generalisierten Gelenkmomente $\varphi_4 - \tau$	120

Tabellenverzeichnis

2.1	Theoretische Bewegungsgrenzen der einzelnen Achsen (vor Kollisionserkennung)	6
2.2	Denavit–Hartenberg-Konvention	10
2.3	Struktur des Roboters mit 8 Körpern und ihren Gelenken	21
2.4	Berechnete Gelenkwerte für Lösung 1 (Fehler: 0,000022)	21
2.5	Berechnete Gelenkwerte für Lösung 2 (Fehler: 0,000022)	22
2.6	Längenänderungen durch Temperaturoausdehnung	38
2.7	Notwendige Korrekturen der Gelenkparameter zur Erhaltung der TCP-Position (Lösung 1)	38
2.8	Notwendige Korrekturen der Gelenkparameter zur Erhaltung der TCP-Position (Lösung 2)	39
3.1	Denavit–Hartenberg-Konvention	49
3.2	Analyse der Singularitäten mithilfe der Singulärwertzerlegung (SVD) für drei verschiedene Konfigurationen	71
3.3	Berechnete Gelenkmomente, -kräfte und Stellungen für Konfiguration 3	73
4.1	Segmentweise Masseverteilung des Robotersystems	100
4.2	Zusammenfassung der Trajektorienstatistik für Position, Geschwindigkeit, Beschleunigung und Ruck	109

Kapitel 1

Einleitung

Die dynamische Auslegung robotischer Systeme ist entscheidend für Präzision, Zuverlässigkeit und Anpassungsfähigkeit in der industriellen Automatisierung. In dieser Hausarbeit wird ein vierachsiger SCARA-Roboter analysiert, um sein kinematisches und dynamisches Verhalten unter realistischen Bedingungen zu verstehen und zu optimieren. Grundlage bilden die Aufgabenstellungen des Moduls „*Dynamische Auslegung von Werkzeugmaschinen, Robotern und Bewegungsachsen*“, mit Fokus auf Kinematik, Singularitäten, Kraftübertragung und dynamische Simulationen.

Aufgabe 1 umfasst die kinematische Modellierung des Roboters:

- Arbeitsraum- und Kollisionsanalyse mit Autodesk Inventor und ROS2.
- Definition der DH-Parameter und symbolische Transformationen in MATLAB.
- Inverse Kinematik zur Bestimmung der Gelenkvariablen $\varphi_0, \varphi_1, \varphi_2, \varphi_4$ und dh_3 .
- Berücksichtigung thermischer Ausdehnung und Kompensationsansätze.

Aufgabe 2 behandelt die statische Kraftübertragung:

- Berechnung der Jacobi-Matrix zur Abbildung von Kräften und Momenten.
- Bestimmung der resultierenden Gelenkkkräfte bei definierten TCP-Belastungen.
- Erkennung von Singularitäten im mechanischen System.

Aufgabe 3 erweitert die Analyse auf dynamische Aspekte:

- Lagrange-Formulierung der Bewegungsgleichungen unter Einbeziehung von Energiegrößen.
- Dynamische Simulation rückbegrenzter Bewegungsprofile in MATLAB.
- Analyse zeitabhängiger Gelenkbelastungen bei periodischen Abläufen.

Überblick zum GitHub-Repository Das GitHub-Repository enthält zentrale MATLAB-Skripte zur Berechnung der inversen Matrix, des Jacobians und der Lagrange-Gleichungen. Ebenfalls enthalten: komplette ROS 2-Konfiguration des Roboters (URDF/XACRO, Setup, Simulation, Bewegungsplanung).



GitHub Repository

Kapitel 2

Aufgabe 1: Arbeitsraum, Kinematik und Inverse Kinematik

2.1 1-a Arbeitsraum und Kollisionsvermeidung

2.1.1 Arbeitsraum

Der 3D-Arbeitsraum des Roboters wurde zunächst in Autodesk Inventor dargestellt, wobei das CAD-Modell zur geometrischen Referenz und Visualisierung der mechanischen Grenzen diente. Anschließend wurde der gleiche Arbeitsbereich in ROS2 (Robot Operating System) simuliert, um ein besseres didaktisches Verständnis der möglichen Bewegungsabläufe und potenziellen Kollisionen zu gewinnen. ROS2, ausgeführt in einem Linux-Container über Docker, wurde hierbei jedoch ausschließlich zur Visualisierung und zur kollisionsfreien Darstellung verwendet.

Die eigentliche Berechnung der inversen Kinematik zur Interaktion mit der Umgebung erfolgte in MATLAB. Dazu wurden 200.000 Zielpositionen im Raum iterativ ausgewertet, um den tatsächlich erreichbaren Arbeitsraum des Roboters zu rekonstruieren. Diese Vorgehensweise ermöglichte eine präzise Analyse der Bewegungskapazitäten unter realistischen Bedingungen.

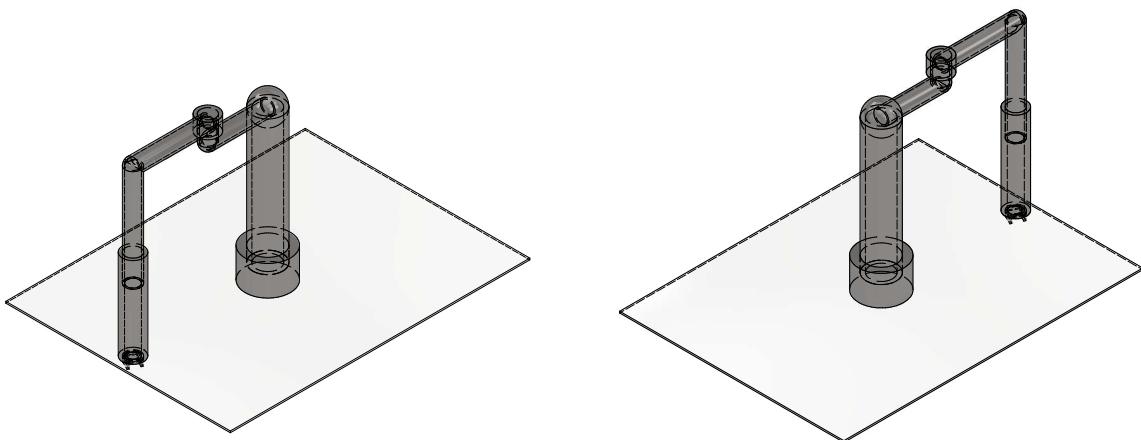


Abbildung 2.1: Isometrische Ansichten der Modelle mit Autodesk Inventor

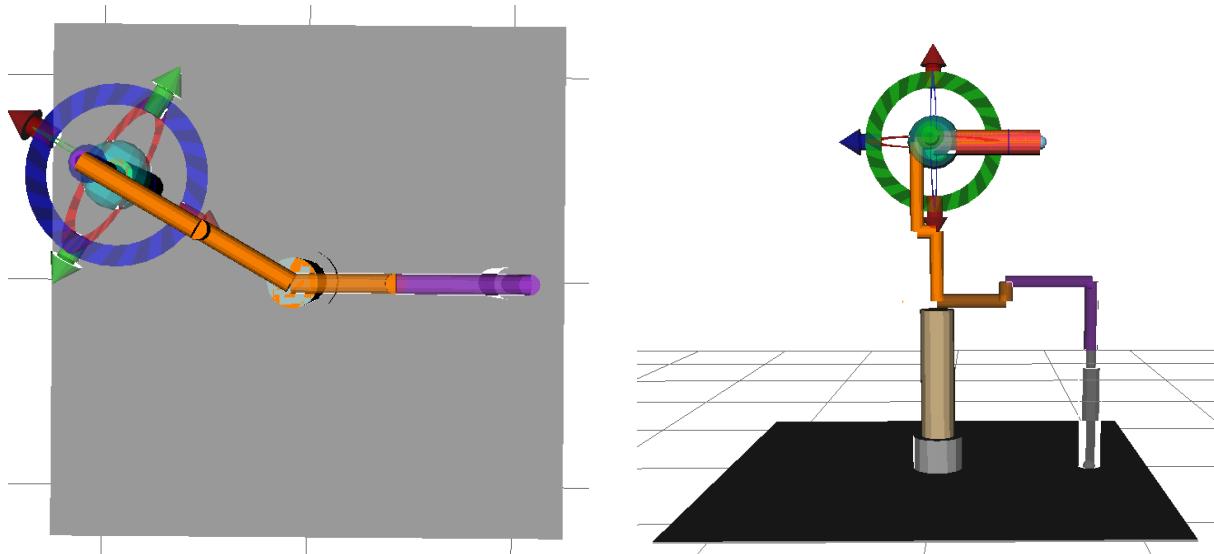


Abbildung 2.2: Zx-Ansicht und Xy-Ansicht mit Robot Operating System (ROS2)

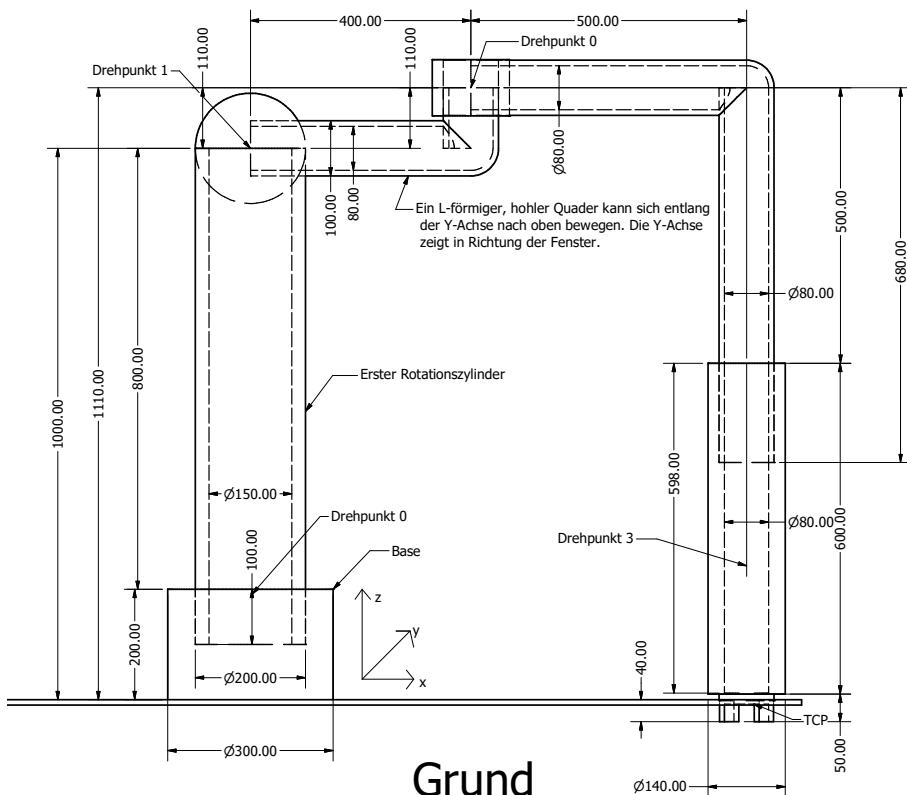


Abbildung 2.3: Die spezifischen Maße des SCARA-Roboters

- **Ergebnis der Interaktion mit der inversen Dynamik:** Durch die Anwendung der in MATLAB implementierten inversen Kinematik konnten 200.000 Zielpunkte ausgewertet werden. Daraus resultierte eine präzise dreidimensionale Rekonstruktion des tatsächlichen Arbeitsraums, der sämtliche erreichbaren Positionen unter Berücksichtigung der mechanischen Beschränkungen und Gelenkwinkelgrenzen abbildet.

3D Workspace with Physical Limits

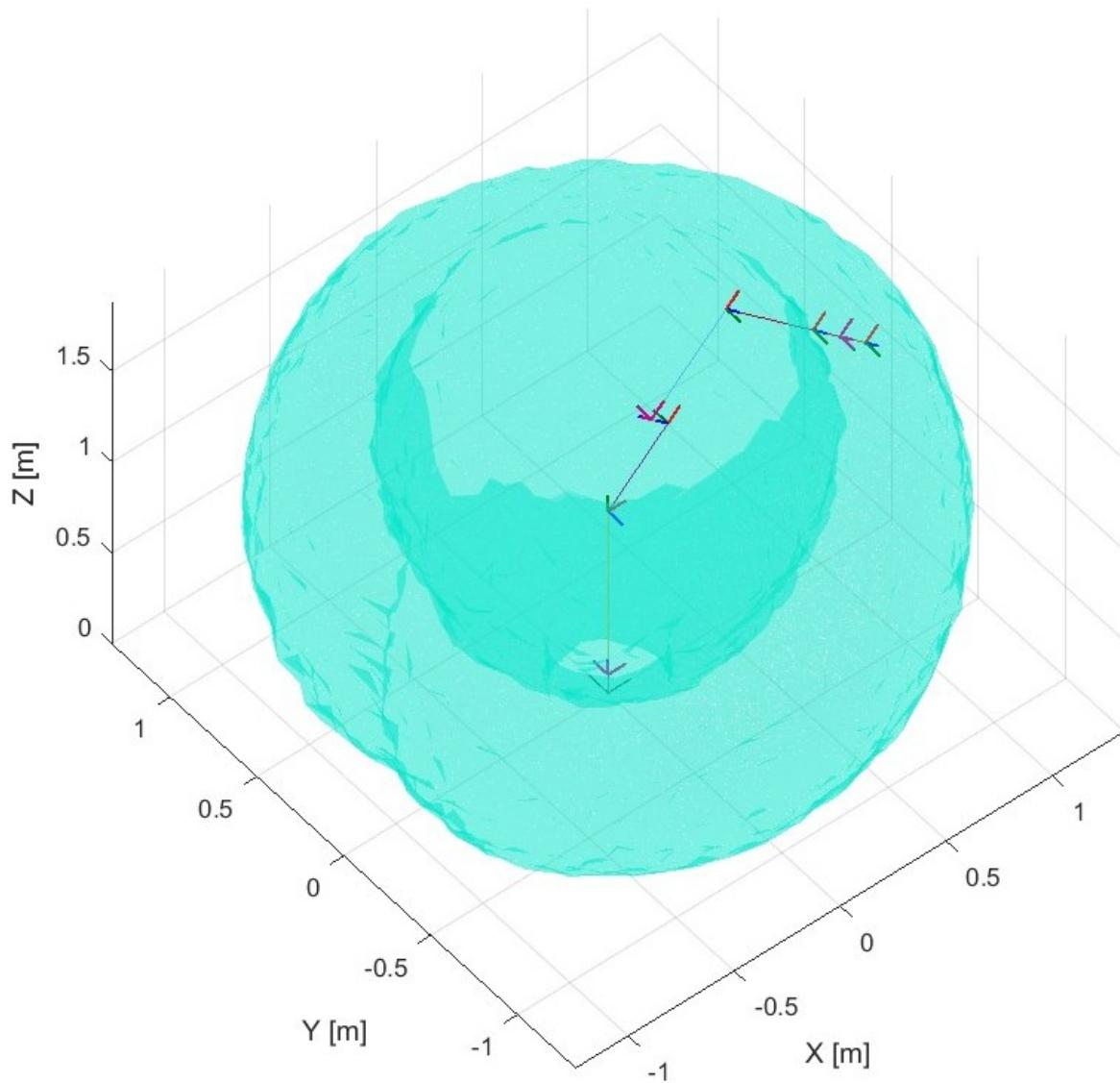


Abbildung 2.4: 3D Arbeitsraum durch Inverse Kinematik

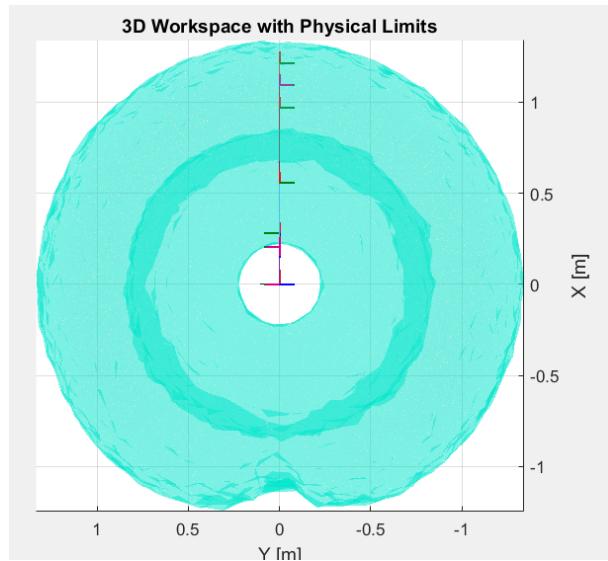


Abbildung 2.5: XY Arbeitsraum durch Inverse Kinematik

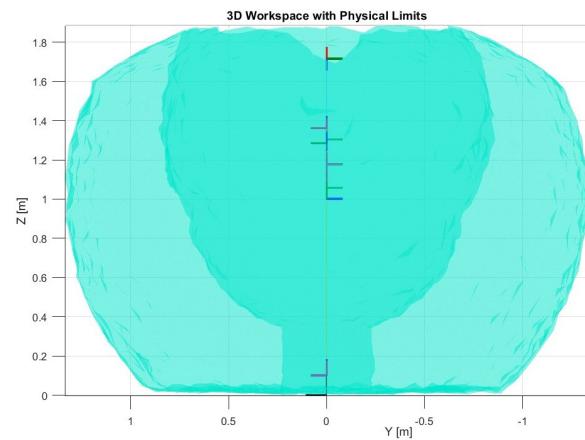


Abbildung 2.6: YZ Arbeitsraum durch Inverse Kinematik

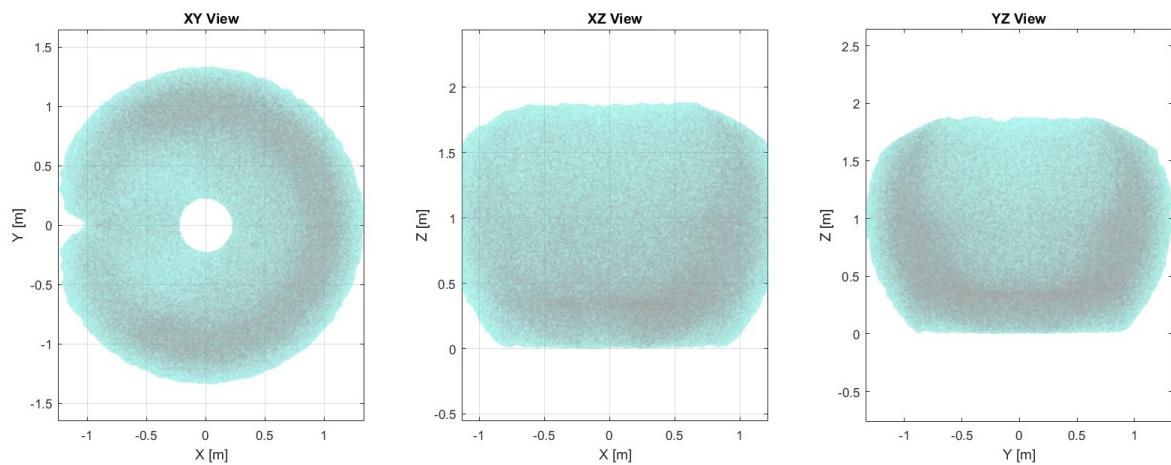


Abbildung 2.7: Arbeitsraum und generierte Punkte für die Oberflächengenerierung

2.1.2 Kollisionsvermeidung

Von ROS2 erkannte Kollisionen mussten in jedem Szenario analysiert werden, da ROS immer noch unmögliche Kollisionen erkennen kann. Auf diese Weise können wir den Arbeitsraum und die möglichen Winkel für $\varphi_0, \varphi_1, \varphi_2, \varphi_4$ und die Extension von dh_3 überprüfen.

Achse / Gelenk	Theoretischer Bewegungsbereich
φ_0	$-150^\circ < \varphi_0 < 150^\circ$
φ_1	$0^\circ < \varphi_1 < 90^\circ$
φ_2	$-170^\circ < \varphi_2 < 170^\circ$
dh_3	$0 \text{ mm} \leq dh_3 \leq 400 \text{ mm}$
φ_4	$-180^\circ < \varphi_4 < 180^\circ$

Tabelle 2.1: Theoretische Bewegungsgrenzen der einzelnen Achsen (vor Kollisionserkennung)

- **Kollision 1:** Kollision mit dem Boden, dies führt zu einer Reduzierung von dh_3 auf 360mm.

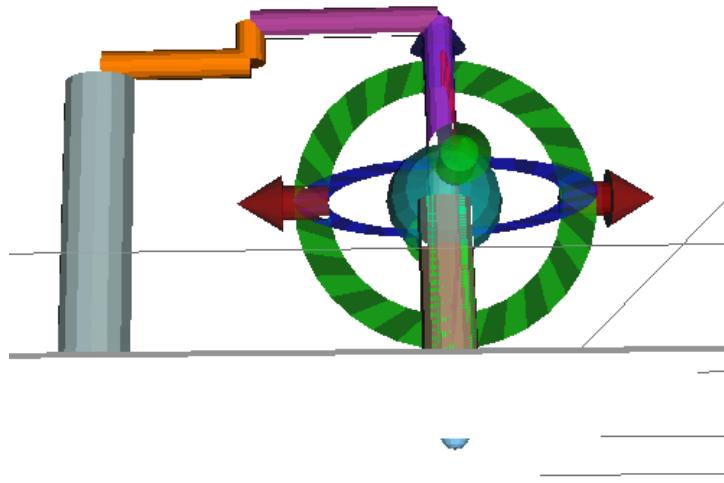


Abbildung 2.8: Kollision 1

- **Kollision 2-3:** Die Kollisionen 2 und 3 entstehen durch eine Einschränkung des Winkels φ_2 , der ursprünglich im Bereich

$$-170^\circ \leq \varphi_2 \leq 170^\circ$$

definiert war. Um die Kollision zu vermeiden, muss φ_2 auf

$$-154^\circ \leq \varphi_2 \leq 154^\circ$$

begrenzt werden.

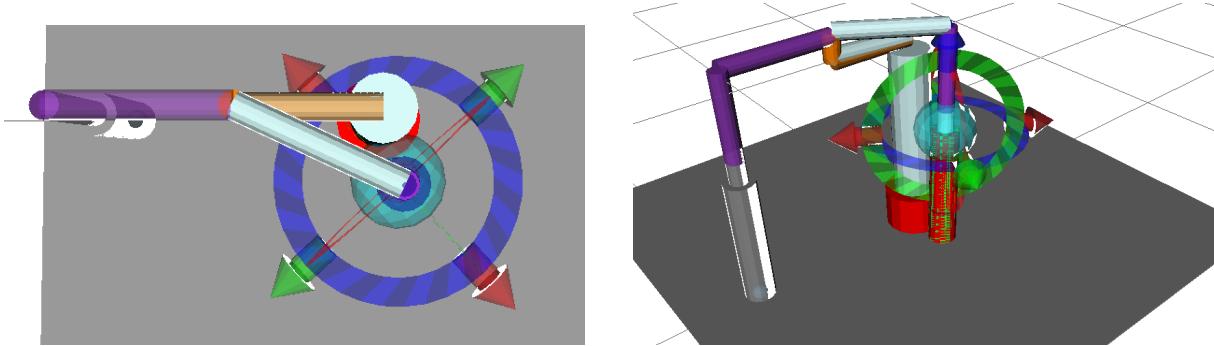


Abbildung 2.9: Kollisionen 2-3

- **Kollision 4:** Diese Kollision ist etwas komplexer, da sie durch die translatorische Bewegung von φ_1 entsteht. Beim Erreichen der Position tritt jedoch keine direkte Kollision auf. Daher kann diese Kollision nur über eine dynamische Simulation erkannt werden. Zur Veranschaulichung wird ein 3D-Modell sowie ein QR-Code zum Video bereitgestellt.

Um diese Kollision zu vermeiden, muss φ_2 weiter eingeschränkt werden:

$$-153^\circ \leq \varphi_2 \leq 153^\circ$$

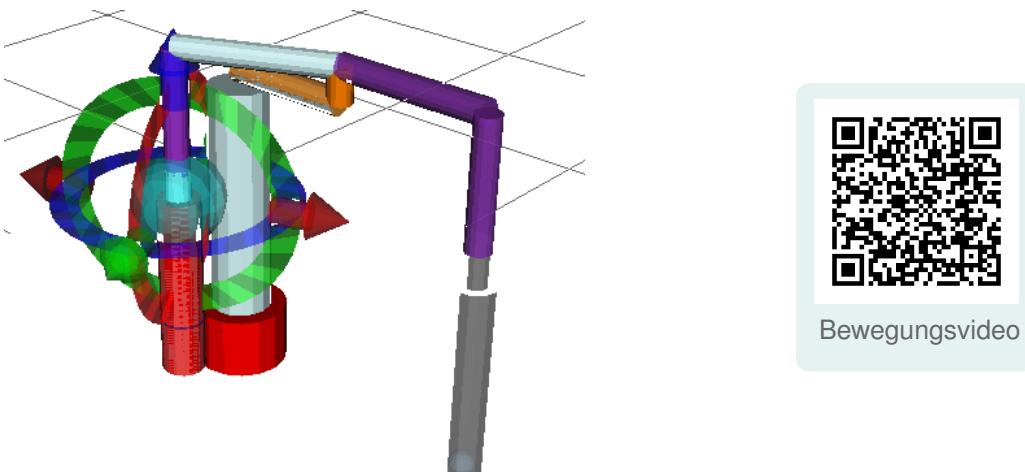


Abbildung 2.10: Kollision 4

2.2 1-b Schematisches Ersatzbild des Roboters mit DH Koordinatensystemen

Zu Beginn wurden die in der Vorlesung vorgestellten Techniken angewendet, bei denen sieben parallele und kollineare Achsen verwendet werden. Eine dieser Achsen sollte dabei entgegengesetzt zur ersten ausgerichtet sein.

Stattdessen wurde eine MATLAB-Bibliothek verwendet, die Koordinatensysteme anhand einer Denavit-Hartenberg-Tabelle grafisch darstellt. Durch iterative Anpassung konnte so die gewünschte Struktur erreicht werden. Dabei wurde insbesondere darauf geachtet, dass alle parallelen Achsen den x -Vektor in dieselbe Richtung zeigen, um den Rechenaufwand zu reduzieren und die Gleichungen zu vereinfachen. Dies führt zu einer geringeren mathematischen Komplexität und erleichtert sowohl die Analyse als auch die anschließenden symbolischen Berechnungen.

Auf diese Weise wurde eine Konfiguration gefunden, die realistische und kohärente Ergebnisse bei der Berechnung der inversen Transformationsmatrix liefert. Diese Konfiguration und die Verwendung der Robotics-Bibliothek in MATLAB ermöglichen zudem teilweise die Berechnung der Lagrange-Funktion sowie der Jacobi-Matrix, wie in den folgenden Kapiteln gezeigt wird.

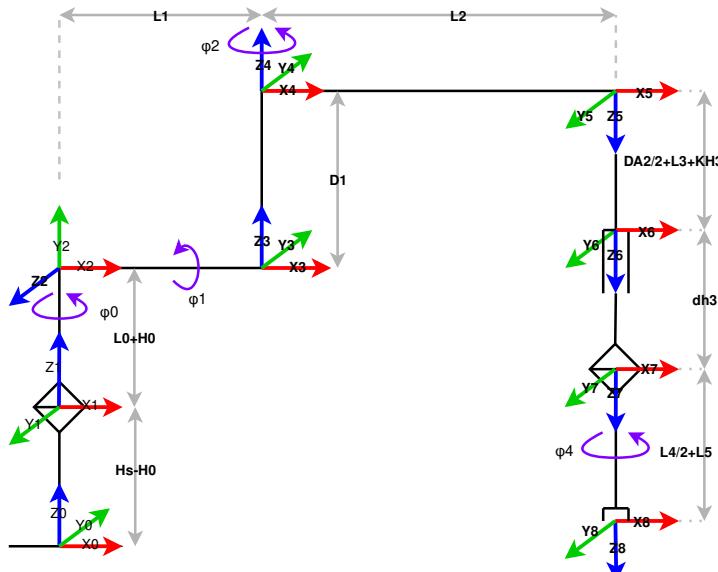


Abbildung 2.11: Schematisches Ersatzbild des Roboters

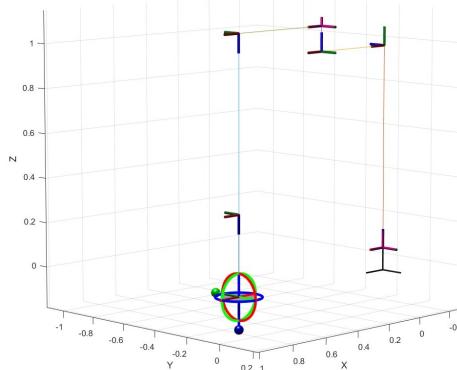


Abbildung 2.12: Resultierende Grafik des Systems mit der Robotics-Bibliothek von MATLAB

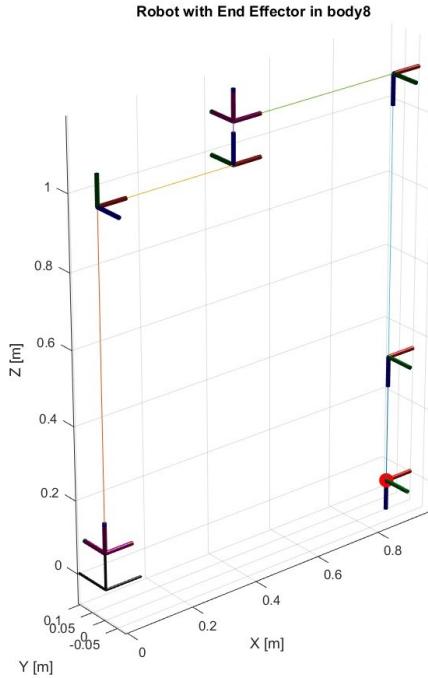


Abbildung 2.13: Resultierende Grafik des Systems mit der Robotics-Bibliothek von MATLAB 2

2.3 1-c Transformationsmatrizen der Gelenke

Werden die AKS nach der beschriebenen DH-Konvention platziert, so lässt sich das nachfolgende AKS_n aus dem Vorgänger AKS_{n-1} stets durch eine Folge von maximal vier Einzelschritten (zwei Drehungen und zwei Verschiebungen) herleiten.

Resultierende DH-Matrix: Die homogene Transformationsmatrix nach Denavit-Hartenberg ergibt sich aus der Multiplikation der Einzelmatrizen:

$$DH = \text{rot}(\varphi_z) \cdot \text{trans}(D_z) \cdot \text{trans}(L_x) \cdot \text{rot}(\alpha_x)$$

$$DH = \begin{bmatrix} \cos \varphi_z & -\sin \varphi_z \cos \alpha_x & \sin \varphi_z \sin \alpha_x & L \cos \varphi_z \\ \sin \varphi_z & \cos \varphi_z \cos \alpha_x & -\cos \varphi_z \sin \alpha_x & L \sin \varphi_z \\ 0 & \sin \alpha_x & \cos \alpha_x & D \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Anmerkungen zur DH-Konvention

1. Die Bewegung einer Achse n wird an der Drehung des zugehörigen AKS sichtbar. Das AKS liegt am **Anfang** der Achse (nicht am Ende).
2. Bei nicht-schneidenden Achsen kann der Koordinatenursprung außerhalb des Körpers liegen. Korrektur durch zusätzliche Verschiebung:

$$T = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \text{Gesamtmatrix: } {}^n[A]_{n-1} = DH \cdot T$$

3. Bei fluchtenden Achsen ($z_{n-1} \parallel z_n$) wird der Ursprung von AKS_n im Verbindungsgelenk platziert.
4. Die z-Achsenrichtung ist bei Schubachsen die positive Bewegungsrichtung, bei Drehachsen gemäß dem positiven Quadranten des WKS.

AKS Nr. i	α_i	L_i	D_i	φ_i
1	0	0	dl_1	0
2	$\pi/2$	0	dl_2	φ_0
3	$-\pi/2$	L_1	0	φ_1
4	0	0	D_1	0
5	π	L_2	0	φ_2
6	0	0	dl_3	0
7	0	0	dh_3	0
8 / TCP	0	0	dl_4	φ_4

Tabelle 2.2: Denavit–Hartenberg-Konvention

Die folgenden Gleichungen beschreiben symbolisch die vier Distanzen dl_1 bis dl_4 , wie sie im Modell verwendet werden. Diese ergeben sich aus den geometrischen Parametern des Roboters:

$$\begin{aligned} dl_1 &= H_s - H_0 \\ dl_2 &= L_0 + H_0 \\ dl_3 &= \frac{DA_2}{2} + L_3 + KH_3 \\ dl_4 &= \frac{L_4}{2} + L_5 \end{aligned}$$

Dabei ist KH_3 eine geometrische Konstante, die den Abstand vom Ende des Glieds L_3 bis zum Schwerpunkt des Bauteils mit der Distanz L_4 bestimmt.

2.3.1 Transformationsmatrizen der Gelenke

1. Joint 1

$${}^0 A_1 = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & dl1 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

2. Joint 2

$${}^1 A_2 = \begin{bmatrix} \cos(\varphi_0) & 0 & \sin(\varphi_0) & 0 \\ \sin(\varphi_0) & 0 & -\cos(\varphi_0) & 0 \\ 0 & 1.0 & 0 & dl2 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

3. Joint 3

$${}^2 A_3 = \begin{bmatrix} \cos(\varphi_1) & 0 & -\sin(\varphi_1) & L_1 \cos(\varphi_1) \\ \sin(\varphi_1) & 0 & \cos(\varphi_1) & L_1 \sin(\varphi_1) \\ 0 & -1.0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

4. Joint 4

$${}^3A_4 = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & D_1 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

5. Joint 5

$${}^4A_5 = \begin{bmatrix} \cos(\varphi_2) & \sin(\varphi_2) & 0 & L_2 \cos(\varphi_2) \\ \sin(\varphi_2) & -\cos(\varphi_2) & 0 & L_2 \sin(\varphi_2) \\ 0 & 0 & -1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

6. Joint 6

$${}^5A_6 = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & dl3 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

7. Joint 7

$${}^6A_7 = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & dh3 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

8. Joint 8

$${}^7A_8 = \begin{bmatrix} \cos(\varphi_4) & -\sin(\varphi_4) & 0 & 0 \\ \sin(\varphi_4) & \cos(\varphi_4) & 0 & 0 \\ 0 & 0 & 1.0 & dl4 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

2.4 1-d Gesamte Transformation

Die Matrix $T_{\text{Basis}}^{\text{TCP}}$ kombiniert sämtliche Gelenktransformationen in einem einzigen Bezugssrahmen. Ihre Elemente ermitteln die Position (t_{14}, t_{24}, t_{34}) und Orientierung (t_{11} bis t_{33}) des TCP in Abhängigkeit der Winkel φ_i und geometrischen Parameter L_i, D_i :

$$T_{\text{Base}}^{\text{TCP}} = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6 \cdot {}^6A_7 \cdot {}^7A_8$$

$$T_{\text{Base}}^{\text{TCP}} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (2.1)$$

$$\begin{aligned} t_{11} &= \sin(\varphi_4)(\cos(\varphi_2)\sin(\varphi_0) + \cos(\varphi_0)\cos(\varphi_1)\sin(\varphi_2)) \\ &\quad - \cos(\varphi_4)(\sin(\varphi_0)\sin(\varphi_2) - \cos(\varphi_0)\cos(\varphi_1)\cos(\varphi_2)) \\ t_{12} &= \cos(\varphi_4)(\cos(\varphi_2)\sin(\varphi_0) + \cos(\varphi_0)\cos(\varphi_1)\sin(\varphi_2)) \\ &\quad + \sin(\varphi_4)(\sin(\varphi_0)\sin(\varphi_2) - \cos(\varphi_0)\cos(\varphi_1)\cos(\varphi_2)) \\ t_{13} &= \cos(\varphi_0)\sin(\varphi_1) \\ t_{14} &= L_1 \cos(\varphi_0)\cos(\varphi_1) - D_1 \cos(\varphi_0)\sin(\varphi_1) - L_2 \sin(\varphi_0)\sin(\varphi_2) \\ &\quad + dh3 \cos(\varphi_0)\sin(\varphi_1) + dl3 \cos(\varphi_0)\sin(\varphi_1) + dl4 \cos(\varphi_0)\sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_0)\cos(\varphi_1)\cos(\varphi_2) \\ \\ t_{21} &= \cos(\varphi_4)(\cos(\varphi_0)\sin(\varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\sin(\varphi_0)) \\ &\quad - \sin(\varphi_4)(\cos(\varphi_0)\cos(\varphi_2) - \cos(\varphi_1)\sin(\varphi_0)\sin(\varphi_2)) \\ t_{22} &= -\cos(\varphi_4)(\cos(\varphi_0)\cos(\varphi_2) - \cos(\varphi_1)\sin(\varphi_0)\sin(\varphi_2)) \\ &\quad - \sin(\varphi_4)(\cos(\varphi_0)\sin(\varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\sin(\varphi_0)) \\ t_{23} &= \sin(\varphi_0)\sin(\varphi_1) \\ t_{24} &= L_1 \cos(\varphi_1)\sin(\varphi_0) + L_2 \cos(\varphi_0)\sin(\varphi_2) - D_1 \sin(\varphi_0)\sin(\varphi_1) \\ &\quad + dh3 \sin(\varphi_0)\sin(\varphi_1) + dl3 \sin(\varphi_0)\sin(\varphi_1) + dl4 \sin(\varphi_0)\sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_1)\cos(\varphi_2)\sin(\varphi_0) \\ \\ t_{31} &= \cos(\varphi_2 - \varphi_4)\sin(\varphi_1) \\ t_{32} &= \sin(\varphi_2 - \varphi_4)\sin(\varphi_1) \\ t_{33} &= -\cos(\varphi_1) \\ t_{34} &= dl1 + dl2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh3 \cos(\varphi_1) \\ &\quad - dl3 \cos(\varphi_1) - dl4 \cos(\varphi_1) + L_2 \cos(\varphi_2)\sin(\varphi_1) \end{aligned}$$

2.4.1 Flussdiagramm und MATLAB-Code für die Transformation

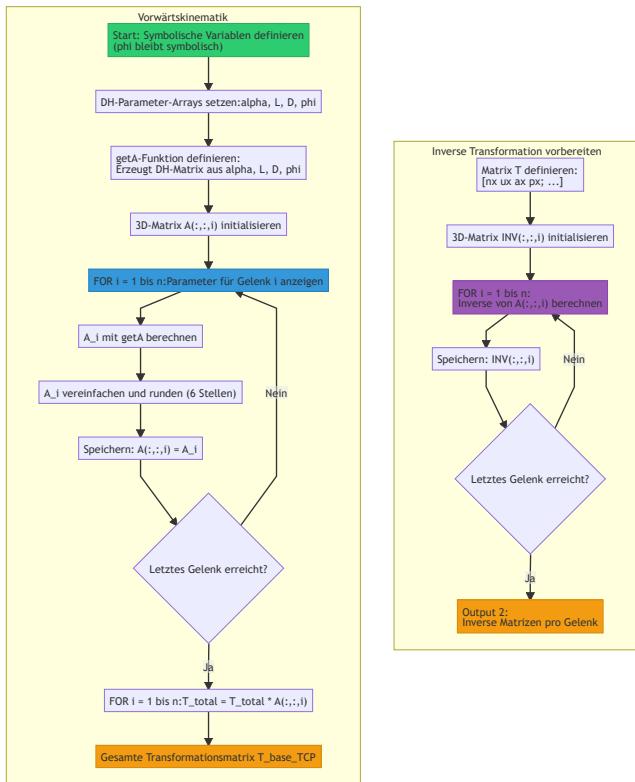


Abbildung 2.14: Matlab Flowchart des Erzeugens DH-Matrix

2.4.2 Matlab Code für die Transformation

```

1 clc; clear;
2
3 % ----- Symbolic parameters -----
4 syms Hs H0 L0 L1 D1 L2 DA2 L3 H3 L5 L4
5 syms phi0 phil phi2 phi4 % all in radians now
6 syms nx ny nz ux uy uz ax ay az px py pz real
7 syms dl1 dl2 dl3 dl4 dh3
8 pi = sym(pi);
9
10
11 % Joint parameters (angles now in radians!)
12 alpha = [ 0, pi/2, -pi/2, 0, pi, 0, 0, 0];
13 L = [ 0, 0, L1, 0, L2, 0, 0, 0];
14 D = [dl1, dl2, 0, D1, 0, dl3, dh3, dl4];
15 phi = [0, phi0, phil, 0, phi2, 0, 0, phi4]; % all in radians
16
17 % Number of joints
18 n = length(alpha);
19
20 % ----- Homogeneous transformation matrix function (radians) -----
21 getA = @(alpha, L, D, phi) ...
22     [ cos(phi), -cos(alpha)*sin(phi), sin(alpha)*sin(phi), L*cos(phi);
23      sin(phi), cos(alpha)*cos(phi), -sin(alpha)*cos(phi), L*sin(phi);
24      0, sin(alpha), cos(alpha), D;
25      0, 0, 0, 1];

```

```
27 % ----- Step-by-step transformation matrices -----
28 A = sym(zeros(4,4,n)); % Symbolic 3D array
29 for i = 1:n
30     fprintf('\n----- Joint %d -----', i);
31     % Show parameter values
32     fprintf('alpha_%d = %.4f rad\n', i, double(alpha(i)));
33     fprintf('L_%d = %s\n', i, string(L(i)));
34     fprintf('D_%d = %s\n', i, string(D(i)));
35     fprintf('phi_%d = %s rad\n', i, string(phi(i)));
36
37     % Build and simplify matrix
38     A_i = getA(alpha(i), L(i), D(i), phi(i));
39     A_i_simple = simplify(A_i, 'Steps', 100);
40     A_i_clean = vpa(A_i_simple, 6); % Round to 6 decimal places
41
42     A(:,:,i) = A_i_clean;
43     % Show matrix
44     disp('Simplified & rounded matrix:');
45     disp(A_i_clean);
46 end
47
48 % ----- Final total transformation -----
49 T_total = eye(4);
50 for i = 1:n
51     T_total = T_total * A(:,:,i);
52 end
53 disp('===== Final Total Transformation T_base_TCP =====');
54 disp(vpa(simplify(T_total, 'Steps', 100), 6)); % Also rounded
55
56 % ----- Inverse transformations -----
57 disp('===== Inverse Transformation =====');
58 T = [nx ux ax px;
59      ny uy ay py;
60      nz uz az pz;
61      0   0   0   1];
62
63 % Inverse transformation matrix
64 % n is already defined as length(alpha)
65 INV= sym(zeros(4,4,n)); % Symbolic 3D array
66 for i =1:n
67     fprintf("Inverse joint : %d \n",i)
68     INV(:,:,i)= simplify(inv(A(:,:,i)));
69     disp(INV(:,:,i));
70
71 end
```

2.5 1-e Berechnung der inversen Dynamik des SCARA-Roboters

Zunächst versuchte ich es mit der algebraischen Methode, wobei ich unendlich viele und physikalisch imaginäre Lösungen fand. Daher kann ich gemäß der Referenz von [1] meinen algebraischen Ansatz mit der geometrischen Einschränkung des Roboters kombinieren, um reelle und systemkonsistente Lösungen zu erhalten. Aus diesem Grund wird nach der Ableitung aller möglichen Gleichungen eine geometrische Analyse durchgeführt, um das Verhalten der Winkel φ_0 , und φ_2 zu verstehen.

Durch einen einfachen Vergleich, nachdem die Gesamttransformationsgleichung erhalten wurde, wurde visuell ersichtlich, dass φ_1 eine direkte Lösung hat. Daher bleibt nur die Berechnung von d_{h3} und φ_4 . Die Ergebnisse finden Sie am Ende dieses Abschnitts.

2.5.1 Algebraische Methode.

Um die unbekannten Variablen $\varphi_0, \varphi_1, \varphi_2, \varphi_4$ und d_{h3} zu berechnen, wird ein vergleichender Ansatz zwischen der gewünschten Endeffektor-Transformation und der tatsächlichen Kettenstruktur des Roboters verwendet.

Die Zieltransformation T beschreibt die gewünschte Position und Orientierung des TCP (Tool Center Point). Diese Transformation wird mit Hilfe der inversen kinematischen Kette Schritt für Schritt in das Basiskoordinatensystem zurückgerechnet, indem sukzessive die inversen Matrizen der Gelenktransformationen A_i von links auf T angewendet werden.

$$T_{\text{umgerechnet}} = A_1^{-1} A_2^{-1} \cdots A_k^{-1} T$$

Dabei wird jede dieser rückgerechneten Transformationen mit dem Produkt der verbleibenden direkten Transformationen $A_{k+1} \cdots A_7$ verglichen:

$$A_1^{-1} A_2^{-1} \cdots A_k^{-1} T \stackrel{!}{=} A_{k+1} A_{k+2} \cdots A_7$$

Dieser Vergleich erfolgt elementweise für die 3×4 -Teilmatrix (Rotationsmatrix und Translationsvektor), um symbolische Gleichungen zu erzeugen. Diese Gleichungen werden anschließend nach der Anzahl der enthaltenen Zielvariablen kategorisiert:

- Gleichungen mit genau einer Zielvariable werden separat gespeichert.
- Gleichungen mit zwei oder drei Zielvariablen werden ebenfalls klassifiziert.

Dieses Verfahren ermöglicht eine schrittweise Lösung der Gleichungen, beginnend mit den einfachen Fällen (eine Variable), um die übrigen Variablen systematisch zu berechnen.

Gleichungen mit einer Zielvariablen

$$\text{Gleichung 1 für } \varphi_1 : a_z + \cos(\varphi_1) = 0 \quad (2.2)$$

$$\text{Gleichung 2 für } \varphi_0 : a_x \sin(\varphi_0) - a_y \cos(\varphi_0) = 0 \quad (2.3)$$

Gleichungen mit zwei Zielvariablen

$$\text{Gleichung 1 für } \varphi_0, \varphi_1 : a_x - \cos(\varphi_0) \sin(\varphi_1) = 0 \quad (2.4)$$

$$\text{Gleichung 2 für } \varphi_0, \varphi_1 : a_y - \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.5)$$

$$\text{Gleichung 3 für } \varphi_0, \varphi_1 : a_x \cos(\varphi_0) - \sin(\varphi_1) + a_y \sin(\varphi_0) = 0 \quad (2.6)$$

$$\text{Gleichung 4 für } \varphi_0, \varphi_2 : L_2 \sin(\varphi_2) - p_y \cos(\varphi_0) + p_x \sin(\varphi_0) = 0 \quad (2.7)$$

$$\text{Gleichung 5 für } \varphi_0, \varphi_1 : a_z \sin(\varphi_1) + a_x \cos(\varphi_0) \cos(\varphi_1) + a_y \cos(\varphi_1) \sin(\varphi_0) = 0 \quad (2.8)$$

$$\text{Gleichung 6 für } \varphi_0, \varphi_2 : p_y \cos(\varphi_0) - L_2 \sin(\varphi_2) - p_x \sin(\varphi_0) = 0 \quad (2.9)$$

$$\text{Gleichung 7 für } \varphi_0, \varphi_1 : n_z \cos(\varphi_1) - n_x \cos(\varphi_0) \sin(\varphi_1) - n_y \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.10)$$

$$\text{Gleichung 8 für } \varphi_0, \varphi_1 : u_z \cos(\varphi_1) - u_x \cos(\varphi_0) \sin(\varphi_1) - u_y \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.11)$$

$$\text{Gleichung 9 für } \varphi_0, \varphi_1 : a_z \cos(\varphi_1) - a_x \cos(\varphi_0) \sin(\varphi_1) - a_y \sin(\varphi_0) \sin(\varphi_1) + 1 = 0 \quad (2.12)$$

$$\text{Gleichung 10 für } \varphi_0, \varphi_1 : n_x \cos(\varphi_0) \sin(\varphi_1) - n_z \cos(\varphi_1) + n_y \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.13)$$

$$\text{Gleichung 11 für } \varphi_0, \varphi_1 : u_x \cos(\varphi_0) \sin(\varphi_1) - u_z \cos(\varphi_1) + u_y \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.14)$$

$$\text{Gleichung 12 für } \varphi_0, \varphi_1 : a_x \cos(\varphi_0) \sin(\varphi_1) - a_z \cos(\varphi_1) + a_y \sin(\varphi_0) \sin(\varphi_1) - 1 = 0 \quad (2.15)$$

Gleichungen mit drei Zielvariablen

$$\text{Gleichung 1 für } \varphi_1, \varphi_2, \varphi_4 : n_z - \sin(\varphi_1) \sin(\varphi_2) \sin(\varphi_4) - \cos(\varphi_2) \cos(\varphi_4) \sin(\varphi_1) = 0 \quad (2.16)$$

$$\text{Gleichung 2 für } \varphi_1, \varphi_2, \varphi_4 : u_z + \cos(\varphi_2) \sin(\varphi_1) \sin(\varphi_4) - \cos(\varphi_4) \sin(\varphi_1) \sin(\varphi_2) = 0 \quad (2.17)$$

$$\text{Gleichung 3 für } dh_3, \varphi_1, \varphi_2 : p_z - dl_2 - dl_1 - D_1 \cos(\varphi_1) - L_1 \sin(\varphi_1) + dh_3 \cos(\varphi_1) \quad (2.18)$$

$$+ dl_3 \cos(\varphi_1) + dl_4 \cos(\varphi_1) - L_2 \cos(\varphi_2) \sin(\varphi_1) = 0 \quad (2.19)$$

$$\text{Gleichung 4 für } \varphi_0, \varphi_2, \varphi_4 : \sin(\varphi_2 - \varphi_4) - n_y \cos(\varphi_0) + n_x \sin(\varphi_0) = 0 \quad (2.20)$$

$$\text{Gleichung 5 für } \varphi_0, \varphi_2, \varphi_4 : u_x \sin(\varphi_0) - u_y \cos(\varphi_0) - \cos(\varphi_2 - \varphi_4) = 0 \quad (2.21)$$

$$\text{Gleichung 6 für } \varphi_0, \varphi_1, \varphi_2 : p_z \sin(\varphi_1) - L_2 \cos(\varphi_2) - dl_1 \sin(\varphi_1) - dl_2 \sin(\varphi_1) \quad (2.22)$$

$$- L_1 + p_x \cos(\varphi_0) \cos(\varphi_1) + p_y \cos(\varphi_1) \sin(\varphi_0) = 0 \quad (2.23)$$

$$\text{Gleichung 7 für } \varphi_0, \varphi_2, \varphi_4 : n_y \cos(\varphi_0) - \sin(\varphi_2 - \varphi_4) - n_x \sin(\varphi_0) = 0 \quad (2.24)$$

$$\text{Gleichung 8 für } \varphi_0, \varphi_2, \varphi_4 : \cos(\varphi_2 - \varphi_4) + u_y \cos(\varphi_0) - u_x \sin(\varphi_0) = 0 \quad (2.25)$$

$$\text{Gleichung 9 für } dh_3, \varphi_0, \varphi_1 : dh_3 - D_1 + dl_3 + dl_4 - dl_1 \cos(\varphi_1) - dl_2 \cos(\varphi_1) \quad (2.26)$$

$$+ p_z \cos(\varphi_1) - p_x \cos(\varphi_0) \sin(\varphi_1) - p_y \sin(\varphi_0) \sin(\varphi_1) = 0 \quad (2.27)$$

$$\text{Gleichung 10 für } \varphi_0, \varphi_1, \varphi_2 : a_y \cos(\varphi_0) \sin(\varphi_2) + a_z \cos(\varphi_2) \sin(\varphi_1) - a_x \sin(\varphi_0) \sin(\varphi_2) \quad (2.28)$$

$$+ a_x \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) + a_y \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) = 0 \quad (2.29)$$

$$\text{Gleichung 11 für } \varphi_0, \varphi_1, \varphi_2 : p_y \cos(\varphi_0) \sin(\varphi_2) - L_1 \cos(\varphi_2) - dl_1 \cos(\varphi_2) \sin(\varphi_1) \quad (2.30)$$

$$- dl_2 \cos(\varphi_2) \sin(\varphi_1) - L_2 + p_z \cos(\varphi_2) \sin(\varphi_1) \quad (2.31)$$

$$- p_x \sin(\varphi_0) \sin(\varphi_2) + p_x \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \quad (2.32)$$

$$+ p_y \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) = 0 \quad (2.33)$$

$$\text{Gleichung 12 für } \varphi_0, \varphi_1, \varphi_2 : a_x \cos(\varphi_2) \sin(\varphi_0) - a_y \cos(\varphi_0) \cos(\varphi_2) + a_z \sin(\varphi_1) \sin(\varphi_2) \quad (2.34)$$

$$+ a_x \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) + a_y \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) = 0 \quad (2.35)$$

$$\text{Gleichung 13 für } \varphi_0, \varphi_1, \varphi_2 : p_x \cos(\varphi_2) \sin(\varphi_0) - p_y \cos(\varphi_0) \cos(\varphi_2) - L_1 \sin(\varphi_2) \quad (2.36)$$

$$- dl_1 \sin(\varphi_1) \sin(\varphi_2) - dl_2 \sin(\varphi_1) \sin(\varphi_2) \quad (2.37)$$

$$+ p_z \sin(\varphi_1) \sin(\varphi_2) + p_x \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \quad (2.38)$$

$$+ p_y \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) = 0 \quad (2.39)$$

2.5.2 Geometrische Methode.

Wie von [2] und [1] dargestellt, wurde die Theorie dieser beiden Autoren grundlegend überarbeitet, was zu folgenden Ergebnissen in Bezug auf die strukturellen Bedingungen des Roboters führte:

- Die kinematische Modellierung wurde an die Mehrkörperdynamik angepasst
- Die Stabilitätsanalyse berücksichtigt nun nichtlineare Effekte

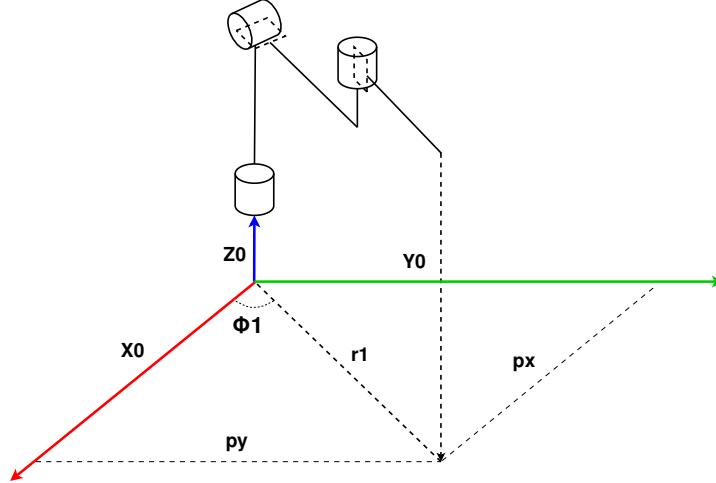


Abbildung 2.15: Geometrische Analyse der inversen Roboterdynamik.

$$\tan(\phi_1) = \frac{P_y}{P_x}$$

$$\phi_1 = \arctan\left(\frac{P_y}{P_x}\right)$$

Und:

$$r_1 = \sqrt{P_x^2 + P_y^2}$$

$$r_1^2 = P_x^2 + P_y^2$$

Es wird eine Hauptverschiebung in P_x und P_y berücksichtigt, um spezifische Lösungen für unseren Roboter zu finden. Anschließend wird gemäß den in [2] und [1] dargestellten Methoden eine der Lösungen unter Berücksichtigung der *Left- und Right-Arm-Bewegung* ausgewertet. Im Folgenden wird unter den unendlich vielen Lösungen nach einer Lösung gesucht, die mit dieser Geometrie übereinstimmt.

Kosinussatz: Für die Berechnung der Winkel φ_0 und φ_{02} in einer möglichen Position wurde eine positive Bewegung von φ_0 angenommen.

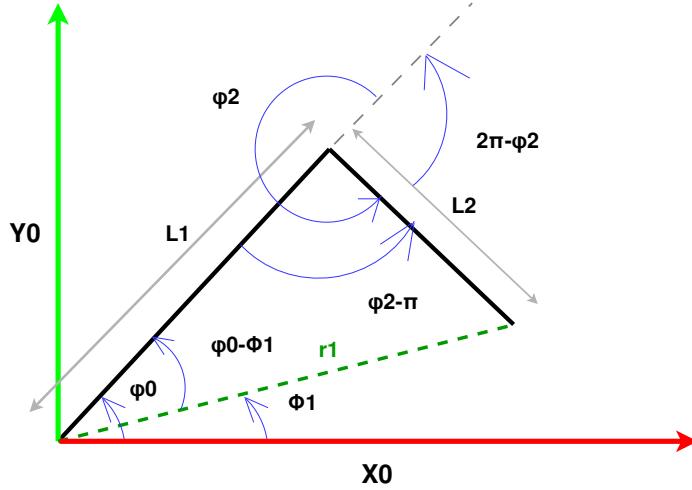


Abbildung 2.16: Geometrische Analyse der inversen Roboterdynamik.

$$\begin{aligned}
 L_2^2 &= L_1^2 + r_1^2 - 2L_1r_1 \cos(\varphi_0 - \phi_1) \\
 2L_1r_1 \cos(\varphi_0 - \phi_1) &= L_1^2 + r_1^2 - L_2^2 \\
 \cos(\varphi_0 - \phi_1) &= \frac{L_1^2 + P_x^2 + P_y^2 - L_2^2}{2L_1\sqrt{P_x^2 + P_y^2}} \\
 \varphi_0 &= \arccos\left(\frac{L_1^2 + P_x^2 + P_y^2 - L_2^2}{2L_1\sqrt{P_x^2 + P_y^2}}\right) + \arctan\left(\frac{P_y}{P_x}\right)
 \end{aligned}$$

$$\begin{aligned}
 r_1^2 &= L_1^2 + L_2^2 - 2L_1L_2 \cos(\varphi_2 - \pi) \\
 \cos(\varphi_2 - \pi) &= \frac{L_1^2 + L_2^2 - (P_x^2 + P_y^2)}{2L_1L_2} \\
 \varphi_2 &= \arccos\left(\frac{L_1^2 + L_2^2 - P_x^2 - P_y^2}{2L_1L_2}\right) + \pi
 \end{aligned}$$

2.5.3 Ergebnisse des Kinematischen Modells durch die Kombination von geometrische und algebraische Methode

Es wurde der in MATLAB programmierte Lösungsalgorithmus verwendet, der alle bereits vorhandenen Gleichungen einfügt und nach Lösungen innerhalb der zuvor erzeugten Gleichungssysteme des algebraischen Verfahrens sucht. Dabei ergab sich folgende Lösung:

Geometrische Parameter

- $H_s = 200.0 \text{ mm}$
- $H_0 = 100.0 \text{ mm}$
- $L_0 = 800.0 \text{ mm}$
- $L_1 = 400.0 \text{ mm}$

- $L_2 = 500.0 \text{ mm}$
- $L_3 = 630.0 \text{ mm}$
- $L_4 = 600.0 \text{ mm}$
- $L_5 = 50.0 \text{ mm}$
- $D_1 = 110.0 \text{ mm}$
- $DA_2 = 100.0 \text{ mm}$
- $KH_3 = 80.0 \text{ mm}$

Auxiliary Definitions

$$\begin{aligned} dl_1 &= H_s - H_0 = 100.0 \text{ mm} \\ dl_2 &= L_0 + H_0 = 900.0 \text{ mm} \\ dl_3 &= \frac{DA_2}{2} + L_3 + KH_3 = 760.0 \text{ mm} \\ dl_4 &= \frac{L_4}{2} + L_5 = 350.0 \text{ mm} \end{aligned}$$

Homogene Transformationsmatrix T_1 Die Transformationsmatrix des WZW in ihrer anfänglichen symbolischen Form ist:

$$T_1^{\text{symb}} = \begin{bmatrix} n_x & u_x & a_x & p_x \\ n_y & u_y & a_y & p_y \\ n_z & u_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Mit den folgenden eingesetzten Werten:

$$\begin{array}{ll} n_x = \cos(\pi/6) \approx 0.8660 & u_x = \cos(\pi/3) = 0.5000 \\ n_y = \sin(\pi/6) = 0.5000 & u_y = -\sin(\pi/3) \approx -0.8660 \\ n_z = 0 & u_z = 0 \\ a_x = 0 \text{ (ersetzt durch } 10^{-6}) & p_x = 400 \\ a_y = 0 \text{ (ersetzt durch } 10^{-6}) & p_y = 100 \\ a_z = -1 & p_z = 50 \end{array}$$

Die resultierende numerische Matrix ist:

$$T_1^{\text{num}} = \begin{bmatrix} 0.8660 & 0.5000 & 0 & 400.0000 \\ 0.5000 & -0.8660 & 0 & 100.0000 \\ 0 & 0 & -1.0000 & 50.0000 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Lösung der Gelenkwinkel Die für jede Variable erhaltenen Gleichungen sind:

$$\varphi_0 = \text{atan2}(p_y, p_x) + \cos^{-1} \left(\frac{L_1^2 - L_2^2 + p_x^2 + p_y^2}{2L_1\sqrt{p_x^2 + p_y^2}} \right)$$

$$\varphi_1 = \cos^{-1}(a_z) + \pi$$

$$\varphi_2 = \cos^{-1} \left(\frac{L_1^2 + L_2^2 - p_x^2 - p_y^2}{2L_1L_2} \right) + \pi$$

$$dh_3 = \frac{1}{\cos(\cos^{-1}(a_z) + \pi)} \cdot \left(dl_1 + dl_2 - p_z + D_1 \cos(\cos^{-1}(a_z) + \pi) \right.$$

$$+ L_1 \sin(\cos^{-1}(a_z) + \pi) - dl_3 \cos(\cos^{-1}(a_z) + \pi) - dl_4 \cos(\cos^{-1}(a_z) + \pi)$$

$$\left. + L_2 \cos \left(\cos^{-1} \left(\frac{L_1^2 + L_2^2 - p_x^2 - p_y^2}{2L_1L_2} \right) + \pi \right) \cdot \sin(\cos^{-1}(a_z) + \pi) \right)$$

$$\varphi_4 = 2\pi k_2 - \sin^{-1} \left(n_y \cos \left(\cos^{-1} \left(\frac{L_1^2 - L_2^2 + p_x^2 + p_y^2}{2L_1\sqrt{p_x^2 + p_y^2}} \right) + \text{atan2}(p_y, p_x) \right) \right.$$

$$- n_x \sin \left(\cos^{-1} \left(\frac{L_1^2 - L_2^2 + p_x^2 + p_y^2}{2L_1\sqrt{p_x^2 + p_y^2}} \right) + \text{atan2}(p_y, p_x) \right) \left. \right)$$

$$+ \cos^{-1} \left(\frac{L_1^2 + L_2^2 - p_x^2 - p_y^2}{2L_1L_2} \right) + \pi$$

Resultierende Numerische Werte Die berechneten Werte für die Gelenkwinkel sind:

- $\varphi_0 = 1.570\,796 \text{ rad} = 90.00^\circ$
- $\varphi_1 = 6.283\,183 \text{ rad} \equiv 0 \text{ rad} = 360.00^\circ \equiv 0.00^\circ$
- $\varphi_2 = 4.068\,885 \text{ rad} = 233.13^\circ \equiv -126.87^\circ$
- $\varphi_4 = 5.116\,083 \text{ rad} = 293.13^\circ \equiv -66.87^\circ$
- $dh_3 = -50.00 \text{ mm}$

Hinweis 1: Die symbolischen Lösungen, die MATLAB für die Gelenkwinkel lieferte, enthielten zusätzlich ganzzahlige Parameter (k, l, m, n, k_2), die aufgrund der periodischen Natur trigonometrischer Funktionen (*Sinus* und *Kosinus*) auftreten. Zur Vereinfachung und um eine eindeutige Lösung zu erhalten, wurden diese Parameter mit Null angesetzt ($k = l = m = n = k_2 = 0$). Diese Annahme entspricht der Auswahl der Hauptlösung innerhalb des Definitionsbereichs.

Grafische Darstellung der möglichen Lösungen Um unsere Koordinatenachsen an der gewünschten Position darzustellen, verwenden wir in MATLAB eine Referenzstruktur von Bodies gemäß der folgenden Tabelle

Idx	Body Name	Joint Name	Joint Type	Parent (Idx)	Children
1	body1	jnt1	fixed	base (0)	body2 (2)
2	body2	jnt2	revolute	body1 (1)	body3 (3)
3	body3	jnt3	revolute	body2 (2)	body4 (4)
4	body4	jnt4	fixed	body3 (3)	body5 (5)
5	body5	jnt5	revolute	body4 (4)	body6 (6)
6	body6	jnt6	fixed	body5 (5)	body7 (7)
7	body7	jnt7	prismatic	body6 (6)	body8 (8)
8	body8	jnt8	revolute	body7 (7)	-

Tabelle 2.3: Struktur des Roboters mit 8 Körpern und ihren Gelenken

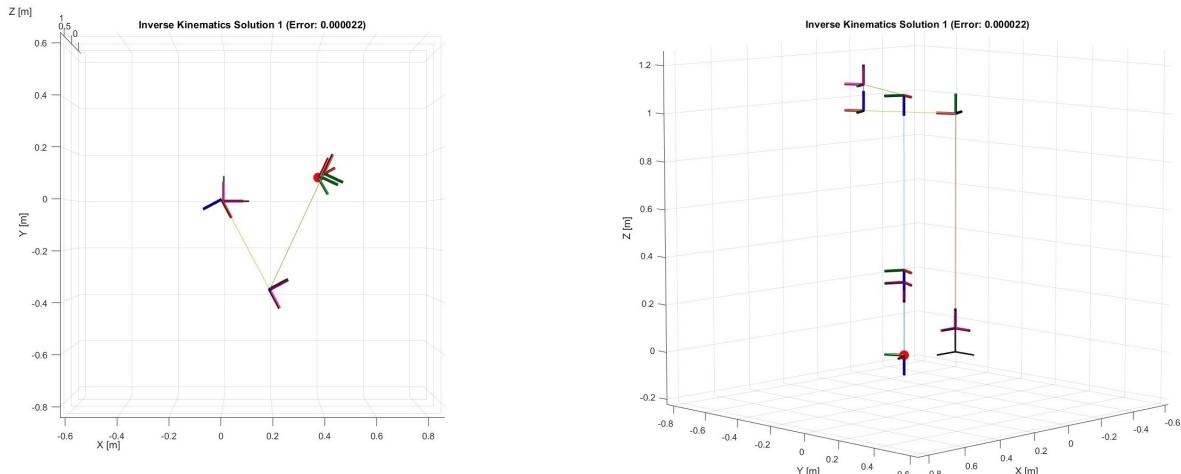


Abbildung 2.17: Lösung Nummer 1: 3D-Ansicht und Projektionen

Variable	Wert (rad / m)	Wert (Grad / mm)
φ_0	-1,0808 rad	-61,93°
φ_1	0,0000 rad	0,00°
φ_2	2,2143 rad	126,87°
dh_3	-0,0500 m	-50,00 mm
φ_4	0,6098 rad	34,94°

Tabelle 2.4: Berechnete Gelenkwerte für Lösung 1 (Fehler: 0,000022)

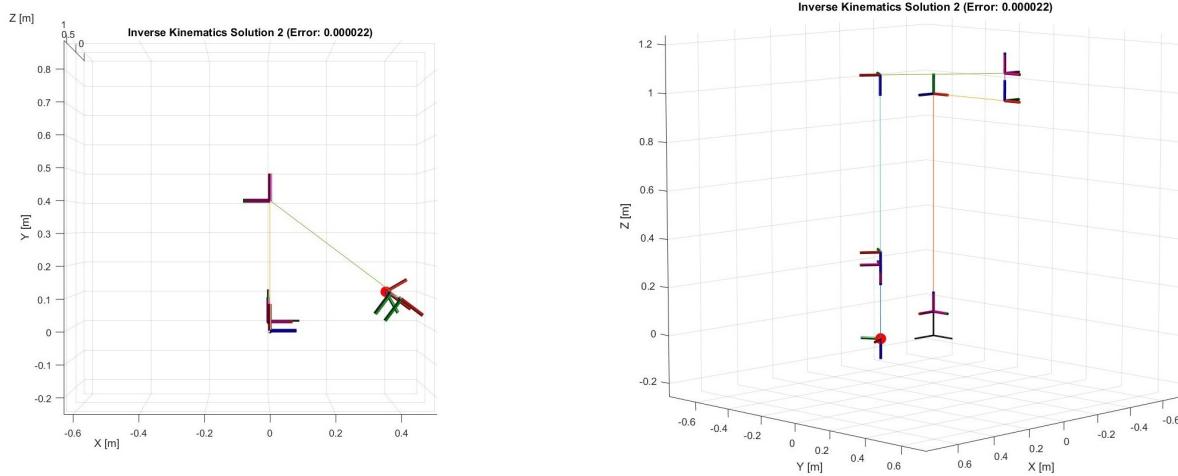


Abbildung 2.18: Lösung Nummer 2: 3D-Ansicht und Projektionen

Variable	Wert (rad / m)	Wert (Grad / mm)
φ_0	1,5708 rad	90,00°
φ_1	0,0000 rad	0,00°
φ_2	-2,2143 rad	-126,87°
dh_3	-0,0500 m	-50,00 mm
φ_4	-1,1671 rad	-66,87°

Tabelle 2.5: Berechnete Gelenkwerte für Lösung 2 (Fehler: 0,000022)

Die Tabelle 2.5 zeigt die gefundene Lösung der inversen Kinematik des SCARA-Roboters, die mit Hilfe eines algebraischen und numerischen Verfahrens ermittelt wurde. Es ist jedoch wichtig zu beachten, dass es mehrere mögliche Lösungen gibt.

2.5.4 Flussdiagramm für die Inverse Kinematik in Matlab

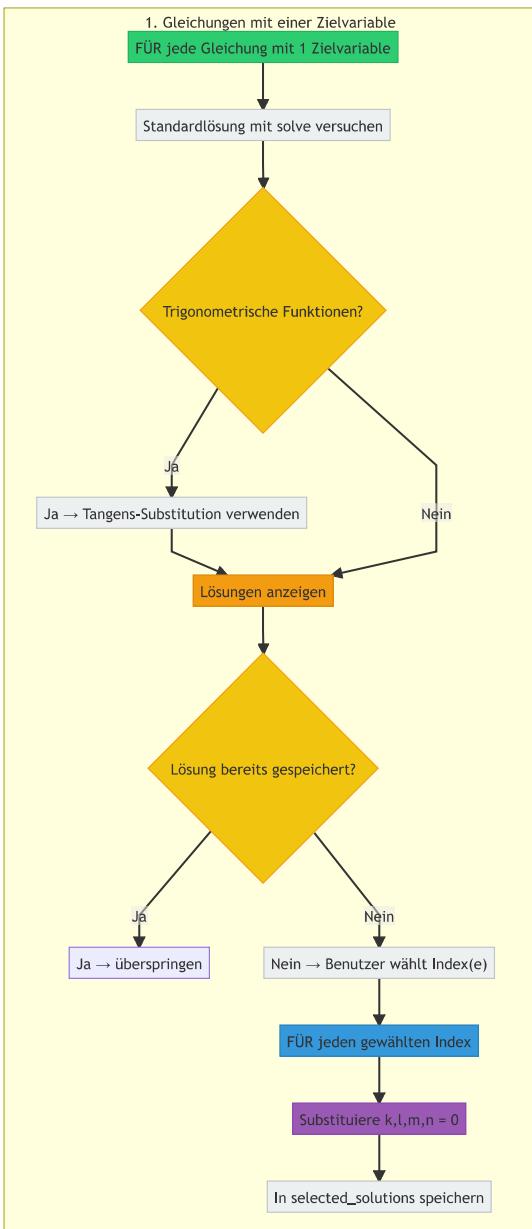


Abbildung 2.19: Flussdiagramm für die Berechnung mit einer Zielvariable

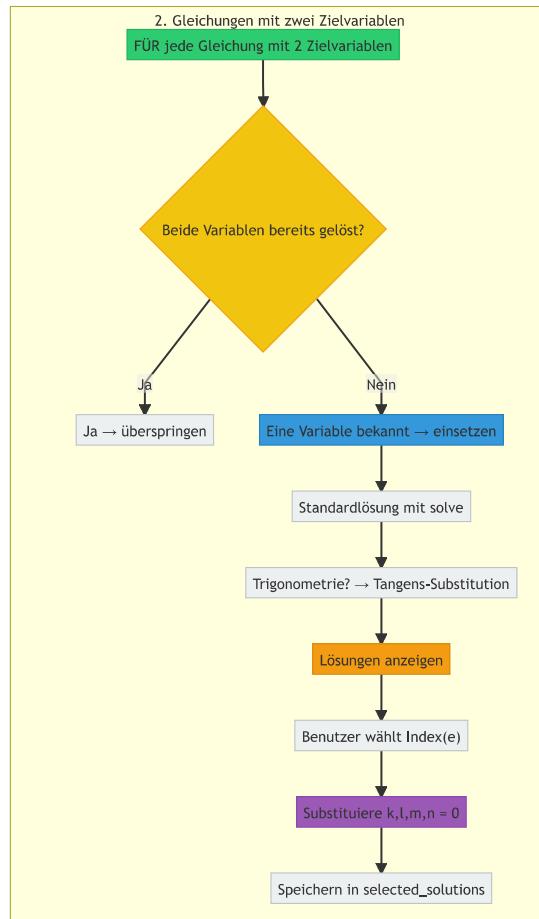


Abbildung 2.20: Flussdiagramm für die Berechnung mit zwei Zielvariablen

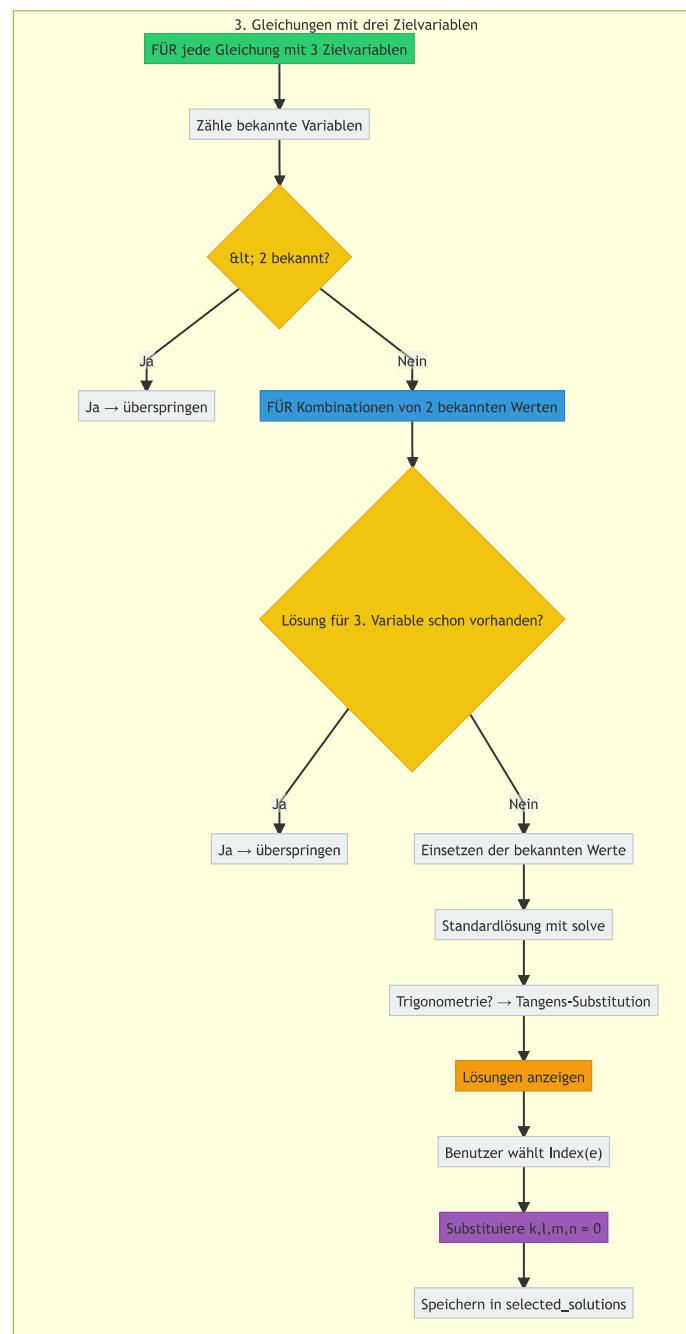


Abbildung 2.21: Flussdiagramm für die Berechnung mit drei Zielvariablen

2.5.5 Matlab Code Inverse Kinematics

Der folgende MATLAB-Code löst das Problem der inversen Kinematik:

```

1 clc; clear;
2 warning('off','all');
3 selected_solutions_file = 'selected_solutions.mat';
4 if isfile(selected_solutions_file)
5     load(selected_solutions_file, 'selected_solutions');
6     skip_solution_selection = true;
7     fprintf('Previous solutions loaded from %s\n', selected_solutions_file);
8 else
9     selected_solutions = struct();
10    skip_solution_selection = false;
11 end
12
13
14 global eqs_one_var eqs_two_vars eqs_three_vars;
15 eqs_one_var = {};% Cell array for 1-variable equations
16 eqs_two_vars = {};% Cell array for 2-variable equations
17 eqs_three_vars = {};
18 % ----- Symbolic parameters -----
19 syms Hs H0 L0 L1 D1 L2 DA2 L3 H3 L5 L4
20 syms phi0 phi1 phi2 phi4 % all in radians now
21 syms d11 d12 d13 d14 dh3
22 syms nx ny nz ux uy uz ax ay az px py pz real
23 pi = sym(pi);
24 % Define target variables
25 % target_vars = [phi0, phi1, phi2, phi4, H3];
26 % target_var_names = {'phi0', 'phi1', 'phi2', 'phi4', 'H3'};
27
28 % Define target variables
29 target_vars = [phi0, phi1, phi2, phi4, dh3];
30 target_var_names = {'phi0', 'phi1', 'phi2', 'phi4', 'dh3'};
31
32 alpha = [ 0, pi/2, -pi/2, 0, pi, 0, 0, 0];
33 L = [ 0, 0, L1, 0, L2, 0, 0, 0];
34
35 D = [d11, d12, 0, D1, 0, d13, dh3, d14];
36 phi = [0, phi0, phi1, 0, phi2, 0, 0, phi4]; % in radians
37
38 % Number of joints
39 n = length(alpha);
40 % ----- Homogeneous transformation matrix function -----
41 getA = @(alpha, L, D, phi) ...
42     [ cos(phi), -cos(alpha)*sin(phi), sin(alpha)*sin(phi), L*cos(phi);
43      sin(phi), cos(alpha)*cos(phi), -sin(alpha)*cos(phi), L*sin(phi);
44      0, sin(alpha), cos(alpha), D;
45      0, 0, 0, 1];
46 % ----- Step-by-step transformation matrices -----
47 A = sym(zeros(4,4,n)); % Symbolic 3D array
48 for i = 1:n
49     A_i = getA(alpha(i), L(i), D(i), phi(i));
50     A_i_simple = simplify(A_i, 'Steps', 100);
51     A_i_clean = vpa(A_i_simple, 6); % Round to 6 decimal places
52     A(:,:,i) = A_i_clean;
53 end
54 % ----- Final total transformation -----
55 T_total = eye(4);
56 for i = 1:n
57     T_total = T_total * A(:,:,i);
58 end
59 % ----- Inverse transformations -----
60 T = [nx ux ax px;

```

```
61     ny uy ay py;
62     nz uz az pz;
63     0 0 0 1];
64 INV = sym(zeros(4,4,n)); % Symbolic 3D array
65 for i =1:n
66     INV(:,:,i)= simplify(inv(A(:,:,i)));
67 end
68
69
70 atemp_pairs = {
71     {T, T_total},
72
73     {inv(A(:,:,1)) * T, A(:,:,2)*A(:,:,3)*A(:,:,4)*A(:,:,5)*A(:,:,6)*A(:,:,7)*A(:,:,8)},
74     {inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,3)*A(:,:,4)*A(:,:,5)*A(:,:,6)*A(:,:,7)*A(:,:,8)},
75
76     {inv(A(:,:,3)) * inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,4)*A(:,:,5)*A(:,:,6)*A(:,:,7)*A(:,:,8)},
77     % Paso 4:
78     {inv(A(:,:,4)) * inv(A(:,:,3)) * inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,5)*A(:,:,6)*A(:,:,7)*A(:,:,8)},
79     % Paso 5:
80     {inv(A(:,:,5)) * inv(A(:,:,4)) * inv(A(:,:,3)) * inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,6)*A(:,:,7)*A(:,:,8)},
81     % Paso 6:
82     {inv(A(:,:,6)) * inv(A(:,:,5)) * inv(A(:,:,4)) * inv(A(:,:,3)) * inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,7)*A(:,:,8)},
83     % Paso 7:
84     {inv(A(:,:,7)) * inv(A(:,:,6)) * inv(A(:,:,5)) * inv(A(:,:,4)) * inv(A(:,:,3)) * inv(A(:,:,2)) * inv(A(:,:,1)) * T, A(:,:,8)}
85 };
86
87
88
89 % ----- Improved analysis function -----
90 function analyze_equation_pair(atemp_left, atemp_right, target_vars,
91     target_var_names)
92     global eqs_one_var eqs_two_vars eqs_three_vars;
93
94     for i = 1:3
95         for j = 1:4
96             eq = simplify(atemp_left(i,j) - atemp_right(i,j));
97             eq = vpa(eq, 6);
98
99             % Identify target variables in equation
100            all_vars = symvar(eq);
101            target_vars_present = intersect(all_vars, target_vars);
102
103            % Create equation info structure
104            eq_info = struct();
105            eq_info.equation = eq == 0;
106            eq_info.position = [i, j];
107            eq_info.target_vars = target_vars_present;
108
109            % Classify and store equations
110            if length(target_vars_present) == 1
111                var_name = target_var_names(target_vars == target_vars_present(1));
112                eq_info.var_name = var_name{1};
113                eqs_one_var{end+1} = eq_info;
114
115            elseif length(target_vars_present) == 2
116                var1_name = target_var_names(target_vars == target_vars_present(1));
```

```

116         var2_name = target_var_names(target_vars == target_vars_present(2));
117         eq_info.var_names = {var1_name{1}, var2_name{1}};
118         eqs_two_vars{end+1} = eq_info;
119     elseif length(target_vars_present) == 3
120         var_names = arrayfun(@(v) target_var_names{target_vars == v},
121             target_vars_present, 'UniformOutput', false);
122         eq_info.var_names = var_names;
123         eqs_three_vars{end+1} = eq_info;
124     end
125 end
126 end
127 % ----- Execute analysis for all pairs -----
128 for pair_idx = 1:length(atemp_pairs)
129     current_pair = atemp_pairs{pair_idx};
130     analyze_equation_pair(current_pair{1}, current_pair{2}, target_vars,
131     target_var_names);
132 end
133 % ----- Display equations in organized matrices -----
134 fprintf('\n\n==== EQUATIONS WITH ONE TARGET VARIABLE ===\n');
135 for k = 1:length(eqs_one_var)
136     eq_info = eqs_one_var{k};
137     fprintf('Equation %d (position %d,%d) for %s:\n', k, eq_info.position(1),
138     eq_info.position(2), eq_info.var_name);
139     disp(eq_info.equation);
140 end
141 fprintf('\n\n==== EQUATIONS WITH TWO TARGET VARIABLES ===\n');
142 for k = 1:length(eqs_two_vars)
143     eq_info = eqs_two_vars{k};
144     fprintf('Equation %d (position %d,%d) variables: %s , %s:\n', k, eq_info.
145     position(1), eq_info.position(2), ...
146         eq_info.var_names{1}, eq_info.var_names{2});
147     disp(eq_info.equation);
148 end
149 fprintf('\n\n==== EQUATIONS WITH THREE TARGET VARIABLES ===\n');
150 for k = 1:length(eqs_three_vars)
151     eq_info = eqs_three_vars{k};
152     fprintf('Equation %d (position %d,%d) variables: %s , %s ,%s :\n', k, eq_info.
153     position(1), eq_info.position(2), ...
154         eq_info.var_names{1}, eq_info.var_names{2}, eq_info.var_names{3});
155     disp(eq_info.equation);
156 end
157 % ----- Solve equations systematically -----
158 fprintf('\n..... SOLVING
159 EQUATIONS ..... \n');
160 solutions = struct();
161 solutions_tan = struct(); % New struct for tangent-based solutions
162 fprintf('===== Equations with 1 Target Variable
163 ======\n');
164 fprintf('===== \n');
165 % First solve single-variable equations
166 selected_solutions = struct(); % Will store the selected solutions
167 for k = 1:length(eqs_one_var)
168     fprintf('===== [Equation %d/%d with 1 variable] =====\n', k, length(eqs_one_var))
169     );
170     fprintf('If no real solutions are found, please enter 0 and proceed to the next
171 equation\n');
172     eq_info = eqs_one_var{k};
173     eq = eq_info.equation;
174     var = eq_info.target_vars;
175     var_name = eq_info.var_name;
176

```

```
170 fprintf('\nSolving equation for %s:\n', var_name);
171 disp(eq);
172
173 % Standard solution
174 sol = [];
175 sol_struct = [];
176 try
177     sol_struct = solve(eq, var, 'ReturnConditions', true, 'PrincipalValue', true
178 );
179     if ~isempty(sol_struct)
180         sol = sol_struct.(char(var));
181     end
182 catch ME
183     fprintf('Error in standard solution for %s: %s\n', var_name, ME.message);
184 end
185
186 % Tangent substitution solution
187 sol_tan = [];
188 if has(eq, 'sin') || has(eq, 'cos')
189     try
190         syms u real;
191         eq_tan = eq;
192         if has(eq, 'sin')
193             sin_expr = sin(var);
194             eq_tan = subs(eq_tan, sin_expr, (2*u)/(1+u^2));
195         end
196         if has(eq, 'cos')
197             cos_expr = cos(var);
198             eq_tan = subs(eq_tan, cos_expr, (1-u^2)/(1+u^2));
199         end
200         sol_u = solve(eq_tan, u);
201         if ~isempty(sol_u)
202             sol_tan = atan(sol_u);
203             sol_tan = simplify(sol_tan);
204         end
205     catch ME
206         fprintf('Error in tangent substitution solution for %s: %s\n', var_name,
207 ME.message);
208     end
209 end
210
211 % Display all found solutions
212 fprintf('== FOUND SOLUTIONS FOR %s ==\n', var_name);
213
214 if ~isempty(sol)
215     fprintf('Standard solutions:\n');
216     for i = 1:length(sol)
217         fprintf('[%d] ', i);
218         disp(vpa(sol(i), 6));
219     end
220
221 if ~isempty(sol_tan)
222     fprintf('Solutions by tangent substitution:\n');
223     for i = 1:length(sol_tan)
224         fprintf('[%d] ', length(sol)+i);
225         disp(vpa(sol_tan(i), 6));
226     end
227
228 if isempty(sol) && isempty(sol_tan)
229     fprintf('No solutions found for %s\n', var_name);
230     continue;
```

```

231 end
232
233 % Allow user to select solutions
234 if isfield(selected_solutions, var_name) && ~isempty(selected_solutions.(var_name))
235     fprintf('Previously selected solutions for %s:\n', var_name);
236     disp(vpa(selected_solutions.(var_name), 6));
237     continue; % Do nothing further if a solution is already saved
238 else
239     % Allow user to select solutions
240     fprintf('Select the solutions you consider reasonable, preferably choose
241 only one, check is not imaginary (e.g., [1 3] or 0 for none):\n');
242     selected_indices = input('Enter solution indices: ');
243
244     all_solutions = [sol; sol_tan];
245     if ~isequal(selected_indices, 0)
246         selected_solutions.(var_name) = all_solutions(selected_indices);
247         fprintf('Selected solutions for %s:\n', var_name);
248         disp(vpa(selected_solutions.(var_name), 6));
249     else
250         fprintf('No solutions selected for %s\n', var_name);
251     end
252 end
253 % Store the selected solutions
254 all_solutions = [sol; sol_tan];
255
256 if ~isequal(selected_indices, 0)
257     for idx = selected_indices
258         if idx <= length(all_solutions)
259             if ~isfield(selected_solutions, var_name) || isempty(
selected_solutions.(var_name))
260                 selected_solutions.(var_name) = all_solutions(idx);
261             else
262                 selected_solutions.(var_name) = [selected_solutions.(var_name);
all_solutions(idx)];
263             end
264         end
265     end
266 end
267
268 fprintf('Selected solutions for %s:\n', var_name);
269 if isfield(selected_solutions, var_name) && ~isempty(selected_solutions.(var_name))
270     disp(vpa(selected_solutions.(var_name), 6));
271 else
272     fprintf('No solutions selected for %s\n', var_name);
273 end
274
275 if ~skip_solution_selection
276     save(selected_solutions_file, 'selected_solutions');
277     % ===== Symbolic replacement k=0, l=0, m=0 in the selected solutions =====
278     % syms k l m n real; % Ensure these symbols exist
279     %
280     % fprintf('\nEvaluating k=0, l=0, m=0 in all selected solutions...\n');
281     % fields = fieldnames(selected_solutions);
282     % for i = 1:length(fields)
283     %     name = fields{i};
284     %     sols = selected_solutions.(name);
285     %     if ~isempty(sols)
286     %         sols = subs(sols, [k, l, m,n], [0, 0, 0,0]);
287     %         selected_solutions.(name) = simplify(sols); % Or use vpa(sols,6) if
you prefer numbers

```

```
288 %      end
289 % end
290 fprintf('Selected solutions saved in %s\n', selected_solutions_file);
291 end
292 fprintf('=====Taking static values from the first iteration
293 =====\n');
294 selected_solutions.phi0 = -acos((-L2^2 + L1^2 + px^2 +py^2) / (2 * ((px^2+py^2)
295 ^ (1/2)) * L1))+atan2(py, px);
296 selected_solutions.phil = acos(az) + 3.14159;
297 selected_solutions.phi2 = acos((px^2 + py^2 - L1^2 - L2^2) / (2 * L1 * L2));
298 disp(selected_solutions);
299
300 % selected_solutions.dh3 = -(1.0*(1.0*d11 + 1.0*d12 - 1.0*pz - 1.0*D1*cos(0+acos(az)
301 % ) - 1.0*L1*sin(0+acos(az)) + d13*cos(0+acos(az)) + d14*cos(0+acos(az)) -
302 % (0.5*sin(0+acos(az))*(-1.0*L1^2 - 1.0*L2^2 + px^2 + py^2 + pz^2))/L1))/cos(0+
303 % acos(az))
304 disp(selected_solutions);
305
306 disp(selected_solutions);
307 fprintf('===== Equations with 2 Target Variable
308 =====\n');
309 for k = 1:length(eqs_two_vars)
310     fprintf('==== [Equation %d/%d with 2 variables] =====\n', k, length(
311     eqs_two_vars));
312     fprintf('If no real solutions are found, please enter 0 and proceed to the next
313     equation\n');
314     eq_info = eqs_two_vars{k};
315     eq = eq_info.equation;
316     vars = eq_info.target_vars;
317     var1_name = eq_info.var_names{1};
318     var2_name = eq_info.var_names{2};
319     if isfield(selected_solutions, var1_name) && ~isempty(selected_solutions.(
320     var1_name)) && ...
321         isfield(selected_solutions, var2_name) && ~isempty(selected_solutions.(
322     var2_name))
323         fprintf('Skipping equation between %s and %s: solutions already exist for
324         both variables.\n', var1_name, var2_name);
325         continue;
326     end
327     fprintf('Solving equation between %s and %s:\n', var1_name, var2_name);
328     disp(eq);
329     has_var1 = isfield(selected_solutions, var1_name) && ~isempty(selected_solutions.
330     (var1_name));
331     has_var2 = isfield(selected_solutions, var2_name) && ~isempty(selected_solutions.
332     (var2_name));
333     if has_var1 && ~has_var2
334         known_var = vars(1);
335         unknown_var = vars(2);
336         known_name = var1_name;
337         unknown_name = var2_name;
338     elseif has_var2 && ~has_var1
339         known_var = vars(2);
340         unknown_var = vars(1);
341         known_name = var2_name;
342         unknown_name = var1_name;
343     else
344         fprintf('No solutions available yet for either of the two variables.\n');
345         continue;
346     end
347     sols = selected_solutions.(known_name);
```

```

338 for idx = 1:length(sols)
339     current_val = sols(idx);
340     % CHECK IF SOLUTIONS ALREADY EXIST
341     if isfield(selected_solutions, unknown_name) &&~isempty(selected_solutions.(unknown_name))
342         fprintf('\nSolutions already exist for %s, skipping combination %d/%d\n',
343             ...
344             unknown_name, idx, length(sols));
345         continue;
346     end
347     fprintf('\n--- Step %d of %d for %s ---\n', ...
348         idx, length(sols), known_name);
349     fprintf('    Value used for %s: %s\n', known_name, char(vpa(current_val, 6)));
350 ;
351     eq_sub = subs(eq, known_var, current_val);
352     % Standard solution
353     fprintf('Standard solutions:\n');
354     try
355         sol_struct = solve(eq_sub, unknown_var, 'ReturnConditions', true);
356         sol_std = vpa(sol_struct.(char(unknown_var)), 6);
357     catch
358         sol_std = [];
359     end
360     fprintf('Solutions by tangent substitution:\n');
361     % Tangent solution
362     sol_tan = [];
363     try
364         if has(eq_sub, sin(unknown_var)) || has(eq_sub, cos(unknown_var))
365             syms u real;
366             eq_tan = eq_sub;
367             if has(eq_sub, sin(unknown_var))
368                 eq_tan = subs(eq_tan, sin(unknown_var), (2*u)/(1 + u^2));
369             end
370             if has(eq_sub, cos(unknown_var))
371                 eq_tan = subs(eq_tan, cos(unknown_var), (1 - u^2)/(1 + u^2));
372             end
373             sol_u = solve(eq_tan, u);
374             sol_u = sol_u(imag(sol_u) == 0); % Filter complex solutions
375             for i = 1:length(sol_u)
376                 phi_candidate = 2 * atan(sol_u(i));
377                 sol_tan = [sol_tan; simplify(vpa(phi_candidate, 6))];
378             end
379         end
380     catch ME
381         fprintf('Error in tangent substitution for %s: %s\n', unknown_name, ME.message);
382     end
383     % Display standard solutions
384     if isempty(sol_std)
385         fprintf('No standard solutions found for %s\n', unknown_name);
386     else
387         fprintf('Standard solutions for %s:\n', unknown_name);
388         for i = 1:length(sol_std)
389             fprintf('[STD %d] %s\n', i, char(sol_std(i)));
390         end
391     end
392     % Display tangent solutions
393     if isempty(sol_tan)
394         fprintf('No solutions by tangent substitution found for %s\n',
395             unknown_name);
396     else

```

```
396     fprintf('Tangent solutions for %s:\n', unknown_name);
397     for i = 1:length(sol_tan)
398         fprintf('[TAN %d] %s\n', i, char(sol_tan(i)));
399     end
400 end
401
402 % Display solutions
403 all_sols = [sol_std; sol_tan];
404 if isempty(all_sols)
405     fprintf('No solutions found for %s\n', unknown_name);
406     continue;
407 end
408
409 fprintf('Possible solutions for %s:\n', unknown_name);
410 for i = 1:length(all_sols)
411     fprintf('[%d] %s\n', i, char(all_sols(i)));
412 end
413 fprintf('\nSelect the valid indices for %s, , preferably choose only one,
check is not imaginary (e.g., [1 2] or 0 for none):\n', unknown_name);
414 sel = input('Indices: ');
415 if ~isequal(sel, 0)
416     % Immediate replacement of k, l, m with 0
417     syms k l m n k1 real;
418     selected = all_sols(sel);
419     selected = subs(selected, [k, l, m,n,k1], [0, 0, 0,0,0]);
420     selected = simplify(selected); % or vpa(selected, 6) if you prefer
numbers
421
422 if ~isfield(selected_solutions, unknown_name) || isempty(
selected_solutions.(unknown_name))
423     selected_solutions.(unknown_name) = selected;
424 else
425     selected_solutions.(unknown_name) = [selected_solutions.(
unknown_name); selected];
426 end
427
428 fprintf('Selected solutions (k=0, l=0, m=0) for %s:\n', unknown_name);
429 disp(vpa(selected_solutions.(unknown_name), 6));
430 else
431     fprintf('No solutions selected for %s\n', unknown_name);
432 end
433 end
434 end
435 fprintf('\n..... values entering the calculation with 3
variables ..... \n');
436 disp(selected_solutions);
437
438 fprintf('=====Equations with 3 Target Variable
===== \n');
439 fprintf('===== \n');
440 for k = 1:length(eqs_three_vars)
441     fprintf('==== [Equation %d/%d with 3 variables] =====\n', k, length(
eqs_three_vars));
442     fprintf('If no real solutions are found, please enter 0 and proceed to the next
equation\n');
443     eq_info = eqs_three_vars{k};
444     eq = eq_info.equation;
445     vars = eq_info.target_vars;
446     var_names = eq_info.var_names;
447     fprintf('\nEquation with three variables: %s, %s and %s\n', var_names{1},
var_names{2}, var_names{3});
448     disp(eq);
449     % Count how many variables already have a solution
```

```

450 known = sum([ ...
451     isfield(selected_solutions, var_names{1}) && ~isempty(selected_solutions.( ...
452         var_names{1})), ...
453     isfield(selected_solutions, var_names{2}) && ~isempty(selected_solutions.( ...
454         var_names{2})), ...
455     isfield(selected_solutions, var_names{3}) && ~isempty(selected_solutions.( ...
456         var_names{3})) ...
457 ]);
458 if known < 2
459     fprintf('At least 2 known variables are needed to solve.\n');
460     continue;
461 end
462 % Try with combinations of 2 known variables
463 for i = 1:3
464     for j = i+1:3
465         name_i = var_names{i};
466         name_j = var_names{j};
467         name_k = setdiff(var_names, {name_i, name_j});
468         var_unknown = sym(name_k{1});
469         if isfield(selected_solutions, name_i) && isfield(selected_solutions,
470             name_j)
471             sols_i = selected_solutions.(name_i);
472             sols_j = selected_solutions.(name_j);
473             for si = 1:length(sols_i)
474                 for sj = 1:length(sols_j)
475                     if isfield(selected_solutions, name_k{1}) && ~isempty(
476                         selected_solutions.(name_k{1}))
477                         fprintf('Skipping combination si=%d, sj=%d: solution
478 already exists for %s\n', ...
479                             si, sj, name_k{1});
480                         continue;
481                     end
482                     eq_sub = subs(eq, [sym(name_i), sym(name_j)], [sols_i(si),
483                         sols_j(sj)]);
484                     % Standard solution
485                     try
486                         sol_struct = solve(eq_sub, var_unknown, '
487 ReturnConditions', true);
488                         sol_std = vpa(sol_struct.(char(var_unknown)), 6);
489                         catch
490                             sol_std = [];
491                         end
492                     % Tangent solution
493                     sol_tan = [];
494                     try
495                         if has(eq_sub, sin(var_unknown)) || has(eq_sub, cos(
496                             var_unknown))
497                             syms u real;
498                             eq_tan = eq_sub;
499                             if has(eq_sub, sin(var_unknown))
500                                 eq_tan = subs(eq_tan, sin(var_unknown), (2*u)/(1
501                                     + u^2));
502                             end
503                             if has(eq_sub, cos(var_unknown))
504                                 eq_tan = subs(eq_tan, cos(var_unknown), (1 - u
505                                     ^2)/(1 + u^2));
506                             end
507                             sol_u = solve(eq_tan, u);
508                             sol_u = sol_u(imag(sol_u) == 0); % Only real
509                         solutions
510                         for r = 1:length(sol_u)
511                             phi_candidate = 2 * atan(sol_u(r));
512                         end
513                     end
514                 end
515             end
516         end
517     end
518 
```

```
500         sol_tan = [sol_tan; simplify(vpa(phi_candidate,
501                                         6))];
502         end
503     end
504     catch ME
505         fprintf('Error in tangent substitution for %s: %s\n',
506                 name_k{1}, ME.message);
507         end
508     % Display solutions
509     all_sols = [sol_std; sol_tan];
510
511     %test
512     if isempty(all_sols)
513         fprintf('No solutions found for %s\n', name_k{1});
514         continue;
515     end
516     fprintf('--- Combination (%d/%d) x (%d/%d) ---\n', ...
517             si, length(sols_i), sj, length(sols_j));
518     fprintf('%s = %s, %s = %s\n', ...
519             name_i, char(vpa(sols_i(si),6)), ...
520             name_j, char(vpa(sols_j(sj),6)));
521
522     fprintf('Looking for solutions for %s...\n', name_k{1});
523     for idx = 1:length(all_sols)
524         fprintf('[%d] ', idx);
525         disp(all_sols(idx));
526     end
527     fprintf('Select the valid indices for %s, preferably choose
528 only one, check is not imaginary (e.g., [1 2] or 0 for none):\n', name_k{1});
529     sel = input('Indices: ');
530     % Add this at the beginning of the block that solves each
531     % three-variable equation
532     % If a previous solution exists, do not ask again or
533     % overwrite
534     if isfield(selected_solutions, name_k{1}) && ~isempty(
535         selected_solutions.(name_k{1}))
536         fprintf('Solution already exists for %s. Skipping
537 assignment.\n', name_k{1});
538         continue;
539     end
540
541     % Evaluate what the user just entered
542     if isequal(sel, 0)
543         fprintf('No solutions selected for %s in this
544 combination.\n', name_k{1});
545         continue;
546     end
547
548     % Save the new solutions
549     syms k l m n k1 real;
550     selected = all_sols(sel);
551     selected = subs(selected, [k, l, m, n, k1], [0, 0, 0, 0, 0]);
552     selected = simplify(selected); % or vpa if you prefer
553
554     selected_solutions.(name_k{1}) = selected;
555     fprintf('Selected solutions for %s:\n', name_k{1});
556     disp(vpa(selected, 6));
557
558     end
559 end
560 end
561 end
562 end
563 end
564 end
```

```

555 disp(selected_solutions);
556 fprintf('\n\n==== FINAL SOLUTIONS SAVED IN selected_solutions.mat ===\n');
557 for k = 1:length(target_var_names)
558     var_name = target_var_names{k};
559     if isfield(selected_solutions, var_name)
560         fprintf('%s:\n', var_name);
561         disp(vpa(selected_solutions.(var_name), 6));
562     else
563         fprintf('%s: no solution found.\n', var_name);
564     end
565 end
566
567 save('final_solutions.mat', 'selected_solutions');
568 fprintf('Final solutions successfully saved in final_solutions.mat\n');

```

Test mit numerischen Werten MATLAB-Code für den Test mit numerischen Werten

```

1 clc; clear;
2 % Load the file with symbolic solutions
3 selected_solutions_file = 'final_solutions.mat';
4 if isfile(selected_solutions_file)
5     load(selected_solutions_file, 'selected_solutions');
6     fprintf('Earlier solutions were loaded from %s\n', selected_solutions_file);
7 else
8     error('The file final_solutions.mat was not found.');
9 end
10 % Define all necessary symbolic variables
11 syms Hs H0 L0 L1 D1 L2 DA2 L3 H3 L5 L4 KH3
12 syms phi0 phi1 phi2 phi4 dh3 % Target variables
13 syms nx ny nz ux uy uz ax ay az px py pz
14 syms k l m n k1 k2 k3 k4 % Possible auxiliary variables in the solutions
15 syms d11 d12 d13 d14 % New symbolic variables to be defined via substitution
16 % Symbolic definitions of auxiliary variables
17 subs_relations = [
18     d11 == Hs - H0;
19     d12 == L0 + H0;
20     d13 == (DA2/2) + L3 + KH3; % New definition for d13
21     d14 == L4/2 + L5
22 ];
23 % =====
24 % FIRST PART: DEFINITION OF CONSTANTS
25 % =====
26 % Numerical values in radians and millimeters
27 values = {
28     H3, 1060;
29     Hs, 200;
30     H0, 100;
31     L0, 800;
32     L1, 400;
33     L2, 500;
34     L3, 630;
35     L4, 600;
36     L5, 50;
37     D1, 110;
38     DA2, 100;
39     KH3, 80; % Updated value for KH3
40     nx, cos(pi/6);
41     ny, sin(pi/6);
42     nz, 0;
43     ux, cos(pi/3);
44     uy, -sin(pi/3);
45     uz, 0;
46     ax, 0;

```

```
47 ay, 0;
48 az, -1;
49 px, 400;
50 py, 100;
51 pz, 50;
52 k, 0;
53 l, 0;
54 m, 0;
55 n, 0;
56 k1, 0;
57 k2, 0;
58 k3, 0;
59 k4, 0
60 };
61 % Get numerical values for the necessary variables
62 Hs_val = values{2,2}; % Hs is the second entry in values
63 H0_val = values{3,2}; % H0 is the third entry
64 L0_val = values{4,2}; % L0 is the fourth entry
65 DA2_val = values{11,2}; % DA2 is the eleventh entry
66 L3_val = values{7,2}; % L3 is the seventh entry
67 L4_val = values{8,2}; % L4 is the eighth entry
68 L5_val = values{9,2}; % L5 is the ninth entry
69 KH3_val = values{12,2}; % KH3 is the twelfth entry
70 % Calculate dl1 to dl4 according to the updated symbolic definitions
71 dl1_val = Hs_val - H0_val;
72 dl2_val = L0_val + H0_val;
73 dl3_val = (DA2_val/2) + L3_val + KH3_val; % Updated calculation
74 dl4_val = (L4_val/2) + L5_val;
75 % Add dl1 to dl4 with correct numerical values
76 values = [values;
77     {dl1, dl1_val};
78     {dl2, dl2_val};
79     {dl3, dl3_val};
80     {dl4, dl4_val}];
81 ];
82 % Display equations and values of dl1 to dl4
83 fprintf('\n==== Definitions and values of dl1 to dl4 ====\n');
84 fprintf('dl1 = Hs - H0 = %.0f - %.0f = %.0f mm\n', Hs_val, H0_val, dl1_val);
85 fprintf('dl2 = L0 + H0 = %.0f + %.0f = %.0f mm\n', L0_val, H0_val, dl2_val);
86 fprintf('dl3 = (DA2/2) + L3 + KH3 = (%.0f/2) + %.0f + %.0f = %.0f mm\n', ...
87     DA2_val, L3_val, KH3_val, dl3_val);
88 fprintf('dl4 = L4/2 + L5 = %.0f/2 + %.0f = %.0f mm\n', L4_val, L5_val, dl4_val);
89 % Display geometric values
90 fprintf('\n==== Geometric Values ====\n');
91 for i = 1:12 % Display up to KH3
92     fprintf('%s = %.1f mm\n', char(values{i,1}), values{i,2});
93 end
94 =====
95 % SECOND PART: EQUATION EVALUATION
96 =====
97 % Create homogeneous transformation matrix T1
98 T1 = [
99     nx, ux, ax, px;
100    ny, uy, ay, py;
101    nz, uz, az, pz;
102    0, 0, 0, 1
103];
104 fprintf('\nMatrix T1 (symbolic):\n');
105 disp(T1)
106 % Evaluate the matrix T1 with the defined values
107 T1_eval = double(subs(T1, values(:,1), values(:,2)));
108 fprintf('\nMatrix T1 with substituted values:\n');
109 disp(T1_eval);
```

```

110 % Names of the target variables
111 vars = {'phi0', 'phi1', 'phi2', 'phi4', 'dh3'};
112 % Evaluation of each target variable
113 for i = 1:length(vars)
114     current_var = vars{i};
115     if isfield(selected_solutions, current_var)
116         expr = selected_solutions.(current_var);
117         fprintf('\n==== Evaluating variable: %s ===%n', current_var);
118         fprintf('Original loaded expression:\n%s\n', char(expr));
119         % First substitute the dl variables with their symbolic definitions
120         expr_subs_dl = subs(expr, [dl1, dl2, dl3, dl4], [Hs - H0, L0 + H0, (DA2/2) +
121             L3 + KH3, L4/2 + L5]);
122         % Then substitute all numerical values
123         try
124             % Create safe values (replace exact zeros with small numbers for ax, ay)
125             values_safe = values;
126             for j = 1:size(values_safe,1)
127                 name = char(values_safe{j,1});
128                 if ismember(name, {'ax', 'ay'}) && values_safe{j,2} == 0
129                     values_safe{j,2} = 1e-6;
130                 end
131             end
132             expr_subs = subs(expr_subs_dl, values_safe(:,1), values_safe(:,2));
133             value = double(expr_subs);
134
135             if startsWith(current_var, 'phi')
136                 fprintf('%s = %.6f rad\n', current_var, value);
137                 fprintf('%s = %.2f\n', current_var, rad2deg(value));
138             else
139                 fprintf('%s = %.2f mm\n', current_var, value);
140             end
141
142             % Store the evaluated value
143             eval([current_var '_val = value;']);
144
145             catch ME
146                 fprintf('\nError during the evaluation of %s:\n%s\n', current_var, ME.
147 message);
148                 fprintf('Expression after substitution:\n%s\n', char(expr_subs_dl));
149                 remaining_vars = symvar(expr_subs_dl);
150                 if ~isempty(remaining_vars)
151                     fprintf('Unsubstituted variables:\n');
152                     disp(remaining_vars);
153                 else
154                     fprintf('No remaining symbolic variables detected, but conversion
155 failed.\n');
156                 end
157             end
158         end
159     end
160 end

```

2.6 1-f Einfluss thermischer Längenänderungen auf die Endeffektorstellung und notwendige Achskorrekturen

Für den betrachteten Roboter wurde die thermische Ausdehnung aufgrund eines Temperaturanstiegs berücksichtigt. Der verwendete lineare Ausdehnungskoeffizient für Stahl beträgt:

- **Ausdehnungskoeffizient:** $\alpha = 1,25 \cdot 10^{-5} \text{ K}^{-1}$
- **Anfangstemperatur:** $T_0 = 18^\circ\text{C}$
- **Endtemperatur:** $T_1 = 30^\circ\text{C}$
- **Temperaturänderung:** $\Delta T = 12 \text{ K}$
- **Ausdehnungsfaktor:** $1 + \alpha \cdot \Delta T = 1,000150$

Die ursprünglichen Längenmaße wurden entsprechend skaliert. Die folgende Tabelle zeigt die ursprünglichen Werte (in Millimetern) und die resultierenden Längen nach thermischer Ausdehnung (in Metern):

Variable	Original [mm]	Expandiert [m]
Hs	200,0	0,200030
H0	100,0	0,100015
L0	800,0	0,800120
L1	400,0	0,400060
D1	110,0	0,110017
L2	500,0	0,500075
L3	630,0	0,630095
DH3	360,0	0,360054
L4	600,0	0,600090
L5	50,0	0,050008
DA2	100,0	0,100015
KH3	80,0	0,080012

Tabelle 2.6: Längenänderungen durch Temperaturausdehnung

Variable	Alt ($T = 18^\circ\text{C}$)	Neu ($T = 30^\circ\text{C}$)	Korrektur
φ_0	$-1,0808 \text{ rad} / -61,93^\circ$	$-1,0810 \text{ rad} / -61,9345^\circ$	$-0,0002 \text{ rad} / -0,0045^\circ$
φ_1	$0,0000 \text{ rad} / 0,00^\circ$	$0,0000 \text{ rad} / 0,0000^\circ$	$0,0000 \text{ rad} / 0,0000^\circ$
φ_2	$2,2143 \text{ rad} / 126,87^\circ$	$2,2145 \text{ rad} / 126,8790^\circ$	$+0,0002 \text{ rad} / +0,0090^\circ$
dh_3	$-0,0500 \text{ m} / -50,00 \text{ mm}$	$-0,0500 \text{ m} / -50,0000 \text{ mm}$	$0,0000 \text{ m} / 0,0000 \text{ mm}$
φ_4	$0,6098 \text{ rad} / 34,94^\circ$	$0,6099 \text{ rad} / 34,9438^\circ$	$+0,0001 \text{ rad} / +0,0038^\circ$

Tabelle 2.7: Notwendige Korrekturen der Gelenkparameter zur Erhaltung der TCP-Position (Lösung 1)

Variable	Alt ($T = 18^\circ\text{C}$)	Neu ($T = 30^\circ\text{C}$)	Korrektur
φ_0	1,5708 rad / 90,00°	1,5709 rad / 90,0070°	+0,0001 rad / +0,0070°
φ_1	0,0000 rad / 0,00°	-0,0000 rad / -0,0000°	0,0000 rad / 0,0000°
φ_2	-2,2143 rad / -126,87°	-2,2145 rad / -126,8790°	-0,0002 rad / -0,0090°
dh_3	-0,0500 m / -50,00 mm	-0,0500 m / -49,9998 mm	+0,0000 m / +0,0002 mm
φ_4	-1,1671 rad / -66,87°	-1,1671 rad / -66,8727°	+0,0000 rad / +0,0023°

Tabelle 2.8: Notwendige Korrekturen der Gelenkparameter zur Erhaltung der TCP-Position (Lösung 2)

2.6.1 Matlab Code Einfluss thermischer Längenänderungen auf die Endeffektorstellung

```
1
2 clc;
3 clear;
4
5 % =====
6 % Thermal Expansion Parameters
7 % =====
8 alpha_St = 12.5e-6; % Thermal expansion coefficient [K^-1]
9 T0 = 18;
10 T1 = 30;
11 delta_T = T1 - T0; % Temperature change [K]
12 expansion_factor = 1 + alpha_St * delta_T;
13
14 % =====
15 % UNIT CONVERSION (mm to m) with thermal expansion
16 % =====
17 % Original dimensions in mm (before expansion)
18 Hs_mm = 200; H0_mm = 100; L0_mm = 800;
19 L1_mm = 400; D1_mm = 110; L2_mm = 500;
20 L3_mm = 630; DH3_mm = 360; L4_mm = 600;
21 L5_mm = 50; H3_mm = 1060; DA2_mm = 100;
22 KH3_mm = 80;
23
24 % Apply thermal expansion to all linear dimensions
25 Hs = Hs_mm * expansion_factor / 1000;
26 H0 = H0_mm * expansion_factor / 1000;
27 L0 = L0_mm * expansion_factor / 1000;
28 L1 = L1_mm * expansion_factor / 1000;
29 D1 = D1_mm * expansion_factor / 1000;
30 L2 = L2_mm * expansion_factor / 1000;
31 L3 = L3_mm * expansion_factor / 1000;
32 DH3 = DH3_mm * expansion_factor / 1000;
33 L4 = L4_mm * expansion_factor / 1000;
34 L5 = L5_mm * expansion_factor / 1000;
35 DA2 = DA2_mm * expansion_factor / 1000;
36 KH3 = KH3_mm * expansion_factor / 1000;
37
38 % Print all thermal expansion information at the beginning
39 disp('=====');
40 disp(' Thermal Expansion Information');
41 disp('=====');
42
43 fprintf('Temperature change: %.1f K\n', delta_T);
44 fprintf('Thermal expansion factor: %.6f\n', expansion_factor);
45 disp(' ');
46 disp('Original dimensions (mm) and expanded dimensions (m):');
47 fprintf('Hs: %.1f mm -> %.6f m\n', Hs_mm, Hs);
48 fprintf('H0: %.1f mm -> %.6f m\n', H0_mm, H0);
49 fprintf('L0: %.1f mm -> %.6f m\n', L0_mm, L0);
50 fprintf('L1: %.1f mm -> %.6f m\n', L1_mm, L1);
51 fprintf('D1: %.1f mm -> %.6f m\n', D1_mm, D1);
52 fprintf('L2: %.1f mm -> %.6f m\n', L2_mm, L2);
53 fprintf('L3: %.1f mm -> %.6f m\n', L3_mm, L3);
54 fprintf('DH3: %.1f mm -> %.6f m\n', DH3_mm, DH3);
55 fprintf('L4: %.1f mm -> %.6f m\n', L4_mm, L4);
56 fprintf('L5: %.1f mm -> %.6f m\n', L5_mm, L5);
57 fprintf('DA2: %.1f mm -> %.6f m\n', DA2_mm, DA2);
58 fprintf('KH3: %.1f mm -> %.6f m\n', KH3_mm, KH3);
59 disp(' ');
60
61 % =====
```

```

62 % Robot Construction (with expanded dimensions)
63 % =====
64 robot = rigidBodyTree('DataFormat','row','MaxNumBodies',8);
65
66 % DH table with expanded dimensions
67 dhparams = [
68     0, 0, Hs-H0, 0;           % 1. base to Z1
69     0, pi/2, L0+H0, 0;       % 2. phi0 (revolute)
70     L1, -pi/2, 0, 0;         % 3. phil (revolute)
71     0, 0, D1, 0;            % 4. fixed
72     L2, pi, 0, 0;           % 5. phi2 (revolute)
73     0, 0, (DA2/2)+L3+KH3, 0; % 6. fixed
74     0, 0, DH3, 0;           % 7. prismatic
75     0, 0, (L4/2)+L5, 0      % 8. phi4 (revolute)
76 ];
77
78 % Rest of your robot construction code remains the same...
79 joint_types = {'fixed', 'revolute', 'revolute', 'fixed', 'revolute', 'fixed', 'prismatic', 'revolute'};
80 activeJointBodies = [2, 3, 5, 7, 8];
81
82 for i = 1:8
83     bodies{i} = rigidBody(['body' num2str(i)]);
84     joints{i} = rigidBodyJoint(['jnt' num2str(i)], joint_types{i});
85     setFixedTransform(joints{i}, dhparams(i,:), 'dh');
86
87     if strcmp(joint_types{i}, 'prismatic')
88         joints{i}.JointAxis = [0 0 1];
89         joints{i}.PositionLimits = [-DH3, 0];
90     elseif strcmp(joint_types{i}, 'revolute')
91         switch i
92             case 3 % phi0
93                 joints{i}.JointAxis = [0 0 1];
94                 joints{i}.PositionLimits = deg2rad([-150, 150]);
95             case 4 % phil
96                 joints{i}.JointAxis = [0 1 0];
97                 joints{i}.PositionLimits = deg2rad([0, 90]);
98             case 5 % phi2
99                 joints{i}.JointAxis = [0 0 1];
100                joints{i}.PositionLimits = deg2rad([-153, 153]);
101            case 8 % phi4
102                joints{i}.JointAxis = [0 0 1];
103                joints{i}.PositionLimits = deg2rad([-180, 180]);
104            end
105        end
106
107     bodies{i}.Joint = joints{i};
108
109    if i == 1
110        addBody(robot, bodies{i}, 'base');
111    else
112        addBody(robot, bodies{i}, bodies{i-1}.Name);
113    end
114 end
115
116 endEffectorFrame = 'body8';
117 showdetails(robot);
118
119 % =====
120 % Inverse Kinematics with Expanded Robot
121 % =====
122 % Target pose remains the same (we want to maintain same end-effector position)
123 T_goal = [0.8660 0.5000 0 400/1000];

```

```
124      0.5000 -0.8660  0       100/1000;
125      0         0     -1.0000  50/1000;
126      0         0       0      1.0000];
127
128 ik = inverseKinematics('RigidBodyTree', robot);
129 ik.SolverParameters.MaxIterations = 1000;
130 weights = [1 1 1 1 1 1];
131
132 % Find solutions with the expanded robot
133 numDesiredSolutions = 2;
134 solutionCount = 0;
135 solutions = {};
136 errors = [];
137
138 while solutionCount < numDesiredSolutions
139     initialGuess = [
140         deg2rad(randi([-150, 150])),    % phi0
141         deg2rad(randi([0, 90])),        % phi1
142         deg2rad(randi([-153, 153])),    % phi2
143         -DH3*rand(),                  % dh3
144         deg2rad(randi([-180, 180]))]; % phi4
145 ];
146
147 [configSoln, solnInfo] = ik(endEffectorFrame, T_goal, weights, initialGuess);
148 configSoln = configSoln(:)';
149
150 % Check solution validity
151 valid = true;
152 for i = 1:length(configSoln)
153     jointBodyIdx = activeJointBodies(i);
154     jointLimits = robot.Bodies{jointBodyIdx}.Joint.PositionLimits;
155     if configSoln(i) < jointLimits(1) || configSoln(i) > jointLimits(2)
156         valid = false;
157         break;
158     end
159 end
160
161 if valid
162     T_solved = getTransform(robot, configSoln, endEffectorFrame);
163     pos_error = norm(T_goal(1:3,4) - T_solved(1:3,4));
164     rot_error = norm(T_goal(1:3,1:3) - T_solved(1:3,1:3));
165     total_error = pos_error + rot_error;
166
167     if total_error < 0.01
168         isDuplicate = false;
169         for j = 1:solutionCount
170             if max(abs(solutions{j} - configSoln)) < 0.01
171                 isDuplicate = true;
172                 break;
173             end
174         end
175
176         if ~isDuplicate
177             solutionCount = solutionCount + 1;
178             solutions{solutionCount} = configSoln;
179             errors(solutionCount) = total_error;
180             fprintf('Found solution %d of %d\n', solutionCount,
181             numDesiredSolutions);
182         end
183     end
184 end
185
```

```
186 % Display thermal compensation results
187 disp('=====');
188 disp(' Thermal Expansion Compensation Results');
189 disp('=====');
190
191 fprintf('Expansion factor: %.6f\n', expansion_factor);
192
193 for i = 1:min(2, length(solutions))
194     fprintf('\nSolution %d (Error: %.6f):\n', i, errors(i));
195     fprintf('phi0: %9.4f rad (%9.4f)\n', solutions{i}(1), rad2deg(solutions{i}(1)));
196     fprintf('phil: %9.4f rad (%9.4f)\n', solutions{i}(2), rad2deg(solutions{i}(2)));
197     fprintf('phi2: %9.4f rad (%9.4f)\n', solutions{i}(3), rad2deg(solutions{i}(3)));
198     fprintf('dh3: %9.4f m (%9.4f mm)\n', solutions{i}(4), solutions{i}(4)*1000);
199     fprintf('phi4: %9.4f rad (%9.4f)\n', solutions{i}(5), rad2deg(solutions{i}(5)));
200 end
201
202 % Visualize solutions
203 for i = 1:min(4, length(solutions))
204     figure;
205     show(robot, solutions{i});
206     hold on;
207     plot3(T_goal(1,4), T_goal(2,4), T_goal(3,4), 'ro', 'MarkerSize', 10, ...
208           'MarkerFaceColor', 'r');
209     title(sprintf('Thermal Compensation Solution %d (Error: %.6f)', i, errors(i)));
210     xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]');
211     grid on;
212 end
```

Kapitel 3

Aufgabe 2: Jacobi-Matrix und Kräfteanalyse

Theoretische Grundlage nach Spong et al. (2006)

Der folgende Abschnitt basiert auf der theoretischen Darstellung in *Robot Modeling and Control* von Spong, Hutchinson und Vidyasagar [3]. Die Inhalte wurden für die vorliegende Hausarbeit übernommen und angepasst, um die Ableitung der Jacobi-Matrix sowie die Interpretation von Geschwindigkeiten und Singularitäten zu erläutern.

3.1 Ableitung der Jacobi-Matrix

Betrachten Sie einen n -gliedrigen Manipulator mit Gelenksvariablen q_1, \dots, q_n . Sei

$${}^0_n T(q) = \begin{bmatrix} {}^0_n R(q) & {}^0_n o(q) \\ 0 & 1 \end{bmatrix} \quad (3.1)$$

die Transformation vom Endeffektorkoordinatensystem zum Basiskoordinatensystem, wobei $q = (q_1, \dots, q_n)$ der Vektor der Gelenksvariablen ist. Während sich der Roboter bewegt, sind sowohl die Gelenksvariablen q_i als auch die Endeffektorposition ${}^0_n o$ und -orientierung ${}^0_n R$ Funktionen der Zeit. Das Ziel dieses Abschnitts ist es, die lineare und Winkelgeschwindigkeit des Endeffektors mit dem Vektor der Gelenksgeschwindigkeiten $\dot{q}(t)$ zu verknüpfen. Sei

$$S({}^0_n \omega) = {}^0_n \dot{R} {}^0_n R^T \quad (3.2)$$

definiert als der Winkelgeschwindigkeitsvektor ${}^0_n \omega$ des Endeffektors, und sei

$${}^0_n v = {}^0_n \dot{o} \quad (3.3)$$

die lineare Geschwindigkeit des Endeffektors. Wir suchen Ausdrücke der Form

$${}^0_n v = J_v \dot{q} \quad (3.4)$$

$${}^0_n \omega = J_\omega \dot{q} \quad (3.5)$$

wobei J_v und J_ω $3 \times n$ Matrizen sind. Wir können die Gleichungen (3.4) und (3.5) zusammen schreiben als

$$\xi = J \dot{q} \quad (3.6)$$

wobei ξ und J gegeben sind durch

$$\xi = \begin{bmatrix} {}^0_n v \\ {}^0_n \omega \end{bmatrix}, \quad J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \quad (3.7)$$

Der Vektor ξ wird manchmal als Körpergeschwindigkeit bezeichnet. Beachten Sie, dass dieser Geschwindigkeitsvektor nicht die Ableitung einer Positionsvariablen ist, da der Winkelgeschwindigkeitsvektor nicht die Ableitung einer bestimmten zeitveränderlichen Größe ist. Die Matrix J wird als *Manipulator-Jacobi-Matrix* oder kurz *Jacobi-Matrix* bezeichnet. Beachten Sie, dass J eine $6 \times n$ Matrix ist, wobei n die Anzahl der Glieder ist. Als Nächstes leiten wir einen einfachen Ausdruck für die Jacobi-Matrix eines beliebigen Manipulators ab.

3.1.1 Winkelgeschwindigkeit

Erinnern Sie sich aus Gleichung (4.29), dass Winkelgeschwindigkeiten als freie Vektoren addiert werden können, vorausgesetzt, sie sind relativ zu einem gemeinsamen Koordinatensystem ausgedrückt. Daher können wir die Winkelgeschwindigkeit des Endeffektors relativ zur Basis bestimmen, indem wir die von jedem Gelenk beigetragene Winkelgeschwindigkeit in der Orientierung des Basiskoordinatensystems ausdrücken und diese dann summieren.

Wenn das i -te Gelenk ein Drehgelenk ist, dann entspricht die i -te Gelenksvariable q_i dem Winkel θ_i und die Drehachse ist \hat{z}_{i-1} . Mit einer leichten Notationsmissbrauch sei ${}^{i-1}\omega_i$ die Winkelgeschwindigkeit des Glieds i , die durch die Rotation des Gelenks i verursacht wird, ausgedrückt im Koordinatensystem $i - 1$ durch

$${}^{i-1}\omega_i = \dot{q}_i \hat{z}_{i-1} = \dot{q}_i \mathbf{k} \quad (3.8)$$

wobei, wie oben, \mathbf{k} der Einheitskoordinatenvektor $(0, 0, 1)^T$ ist.

Wenn das i -te Gelenk ein Schubgelenk ist, dann ist die Bewegung des Koordinatensystems i relativ zum Koordinatensystem $i - 1$ eine Translation und

$${}^{i-1}\omega_i = 0 \quad (3.9)$$

Daher hängt, wenn Gelenk i ein Schubgelenk ist, die Winkelgeschwindigkeit des Endeffektors nicht von q_i ab, das nun d_i entspricht.

Daher wird die Gesamtwinkelgeschwindigkeit des Endeffektors, ${}_n^0\omega$, im Basiskoordinatensystem durch Gleichung (4.29) bestimmt als

$${}_n^0\omega = \rho_1 \dot{q}_1 \mathbf{k} + \rho_2 \dot{q}_2 {}_1^0R\mathbf{k} + \cdots + \rho_n \dot{q}_n {}_{n-1}^0R\mathbf{k} = \sum_{i=1}^n \rho_i \dot{q}_i {}_{i-1}^0R\mathbf{k} \quad (3.10)$$

wobei ρ_i gleich 1 ist, wenn Gelenk i ein Drehgelenk ist, und 0, wenn Gelenk i ein Schubgelenk ist, da

$$\hat{z}_i^0 = \mathbf{k} = {}_i^0R\mathbf{k} \quad (3.11)$$

Natürlich ist ${}_0^0\omega = \mathbf{k} = [0, 0, 1]^T$.

Die untere Hälfte der Jacobi-Matrix J_ω in Gleichung (3.7) ist somit gegeben als

$$J_\omega = [\rho_1 \mathbf{k} \quad \rho_2 {}_1^0R\mathbf{k} \quad \cdots \quad \rho_n {}_{n-1}^0R\mathbf{k}] \quad (3.12)$$

Beachten Sie, dass wir in dieser Gleichung die Hochindizes für die Einheitsvektoren entlang der z -Achsen weggelassen haben, da diese alle auf das Weltkoordinatensystem bezogen sind. Im Rest des Kapitels werden wir dieser Konvention folgen, wenn es keine Mehrdeutigkeit bezüglich des Referenzkoordinatensystems gibt.

3.1.2 Lineare Geschwindigkeit

Die lineare Geschwindigkeit des Endeffektors ist einfach \dot{o}_n^0 . Nach der Kettenregel für die Differentiation gilt:

$$\dot{o}_n^0 = \sum_{i=1}^n \frac{\partial o_n^0}{\partial q_i} \dot{q}_i \quad (4.48)$$

Somit ist die i -te Spalte von J_v , die wir als J_{v_i} bezeichnen, gegeben durch:

$$J_{v_i} = \frac{\partial o_n^0}{\partial q_i} \quad (4.49)$$

Des Weiteren ist dieser Ausdruck genau die lineare Geschwindigkeit des Endeffektors, die sich ergeben würde, wenn q_i gleich eins und die anderen q_j null wären. Mit anderen Worten, die i -te Spalte des Jakobian kann erzeugt werden, indem alle Gelenke außer dem i -ten Gelenk fixiert und das i -te Gelenk mit Einheitsgeschwindigkeit betätigt wird. Diese Beobachtung führt zu einer einfachen und intuitiven Herleitung des Jakobian der linearen Geschwindigkeit, wie wir nun zeigen werden. Wir betrachten nun die beiden Fälle von prismatischen und rotatorischen Gelenken getrennt.

Fall 1: Prismatische Gelenke

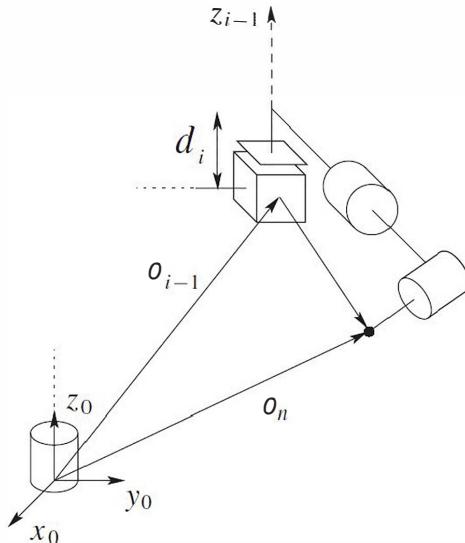


Abbildung 3.1: Bewegung des Endeffektors durch prismatisches Gelenk

Abbildung 3.1 veranschaulicht den Fall, wenn alle Gelenke außer einem einzelnen prismatischen Gelenk fixiert sind. Da Gelenk i prismatisch ist, bewirkt es eine reine Translation des Endeffektors. Die Translationsrichtung ist parallel zur Achse z_{i-1} und die Größe der Translation ist \dot{d}_i , wobei d_i die DH-Gelenkvariable ist. Im Bezug zum Basiskoordinatensystem haben wir somit:

$$\dot{o}_n^0 = \dot{d}_i R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \dot{d}_i z_{i-1}^0 \quad (4.50)$$

wobei d_i die Gelenkvariable für das prismatische Gelenk i ist. Für den Fall prismatischer Gelenke gilt somit, nach Weglassen der Hochstellungen:

$$J_{v_i} = z_{i-1} \quad (4.51)$$

Fall 2: Rotatorische Gelenke

Abbildung 3.2 veranschaulicht den Fall, wenn alle Gelenke außer einem einzelnen rotatorischen Gelenk fixiert sind. Da Gelenk i rotatorisch ist, haben wir $q_i = \theta_i$. Bezuglich Abbildung 4.2 und unter der Annahme, dass alle Gelenke außer Gelenk i fixiert sind, sehen wir, dass die lineare Geschwindigkeit des Endeffektors einfach die Form $\omega \times r$ hat, wobei

$$\omega = \dot{\theta}_i z_{i-1} \quad (4.52)$$

und

$$r = o_n - o_{i-1} \quad (4.53)$$

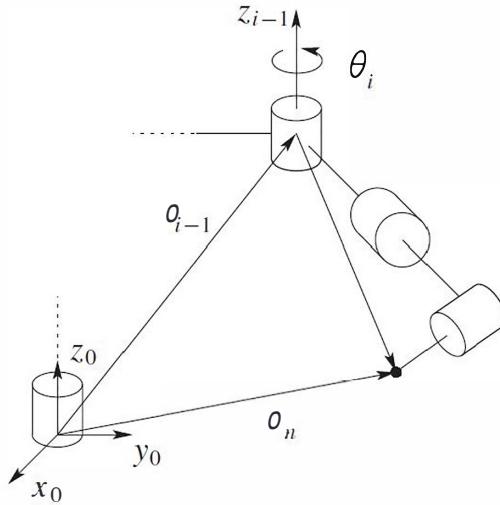


Abbildung 3.2: Bewegung des Endeffektors durch rotatorisches Gelenk

3.1.3 Kombination des linearen und des Winkelgeschwindigkeits-Jacobians

Wie wir im vorherigen Abschnitt gesehen haben, ist die obere Hälfte des Jacobians J_v gegeben durch

$$J_v = [J_{v1} \cdots J_{vn}] \quad (4.55)$$

wobei die i -te Spalte J_{vi} ist:

$$J_{vi} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{für ein rotatorisches Gelenk } i \\ z_{i-1} & \text{für ein prismatisches Gelenk } i \end{cases} \quad (4.56)$$

Die untere Hälfte des Jacobians ist gegeben durch

$$J_\omega = [J_{\omega 1} \cdots J_{\omega n}] \quad (4.57)$$

wobei die i -te Spalte $J_{\omega i}$ ist:

$$J_{\omega i} = \begin{cases} z_{i-1} & \text{für ein rotatorisches Gelenk } i \\ 0 & \text{für ein prismatisches Gelenk } i \end{cases} \quad (4.58)$$

Die obigen Formeln ermöglichen eine einfache Bestimmung des Jacobians eines beliebigen Manipulators, sobald alle erforderlichen Größen nach der Lösung der Vorwärtstransformation verfügbar sind. Tatsächlich sind nur die Einheitsvektoren z_i und die Koordinaten der Ursprünge o_1, \dots, o_n notwendig,

um den Jacobian zu berechnen. Bei genauerer Betrachtung erkennt man, dass die Koordinaten von z_i relativ zum Basis-Koordinatensystem durch die ersten drei Elemente der dritten Spalte von T_0^i gegeben sind, während o_i durch die ersten drei Elemente der vierten Spalte von T_0^i bestimmt wird. Es werden also nur die dritte und vierte Spalte der T -Matrizen benötigt, um den Jacobian gemäß den obigen Formeln zu berechnen.

Die Jacobimatrix dient somit nicht nur zur Berechnung der Geschwindigkeit des Endeffektors, sondern auch zur Bestimmung der Geschwindigkeit eines beliebigen Punktes am Manipulator. Dies wird insbesondere in Kapitel 6 von Bedeutung sein, wenn die Geschwindigkeit des Schwerpunkts der einzelnen Glieder zur Herleitung der Bewegungsgleichungen benötigt wird.

3.2 Berechnung der Jacobi-Matrix

Basierend auf der zuvor dargestellten Theorie wissen wir, dass in unserer Denavit-Hartenberg-Tabelle die Gelenke an folgenden Positionen liegen:

AKS Nr. i	α_i	L_i	D_i	φ_i
1	0	0	dl_1	0
2	$\pi/2$	0	dl_2	φ_0
3	$-\pi/2$	L_1	0	φ_1
4	0	0	D_1	0
5	π	L_2	0	φ_2
6	0	0	dl_3	0
7	0	0	dh_3	0
8 / TCP	0	0	dl_4	φ_4

Tabelle 3.1: Denavit–Hartenberg-Konvention

Diese ergeben sich aus den geometrischen Parametern des Roboters:

$$\begin{aligned} dl_1 &= H_s - H_0 \\ dl_2 &= L_0 + H_0 \\ dl_3 &= \frac{DA_2}{2} + L_3 + KH_3 \\ dl_4 &= \frac{L_4}{2} + L_5 \end{aligned}$$

Die Position der Gelenke wird durch die folgenden Indizes repräsentiert

$$\text{indices_vars} = [2, 3, 4, 7, 8]$$

Variablen für die Jacobi-Matrix Die folgenden Variablen werden für die Berechnung der Jacobi-Matrix berücksichtigt:

$$\{\varphi_0\}, \{\varphi_1\}, \{\varphi_2\}, \{dh3\}, \{\varphi_4\}$$

Die vollständige Form der geometrischen Jacobi-Matrix J , bestehend aus den linearen und rotatorischen Geschwindigkeitsanteilen der Gelenke 2, 3, 4, 7 und 8, ergibt sich wie folgt:

$$J = \begin{bmatrix} J_{v2} & J_{v3} & J_{v5} & J_{v7} & J_{v8} \\ J_{\omega 2} & J_{\omega 3} & J_{\omega 5} & J_{\omega 7} & J_{\omega 8} \end{bmatrix} \quad (3.13)$$

$$J = \begin{bmatrix} z_1 \times (o_8 - o_1) & z_2 \times (o_8 - o_2) & z_4 \times (o_8 - o_4) & z_6 & z_7 \times (o_8 - o_7) \\ z_1 & z_2 & z_4 & \mathbf{0} & z_7 \end{bmatrix} \quad (3.14)$$

In diesem Abschnitt wird die Berechnung der Jacobi-Matrix für ein Robotersystem mit 8 Gelenken beschrieben. Für jedes Gelenk werden die Transformationsmatrizen T_i , die Z-Achsen z_i und die Positionen O_i berechnet.

3.2.1 Transformationsmatrizen und Gelenkvariablen

Für jedes Gelenk i wird die Transformationsmatrix T_i als Produkt aller vorherigen Transformationsmatrizen A_1 bis A_i berechnet:

$$T_i = A_1 \cdot A_2 \cdot \dots \cdot A_i \quad (3.15)$$

Zusätzlich werden für jedes Gelenk folgende Größen bestimmt:

- z_i : Die Z-Achse des Gelenks i im globalen Koordinatensystem
- O_i : Die Position des Gelenks i im globalen Koordinatensystem

Berechnung für jedes Gelenk

Gelenk 1

$$T_1 = A_1 = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & dl1 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.16)$$

$$z_1 = \begin{bmatrix} 0 \\ 0 \\ 1.0 \end{bmatrix}, \quad O_1 = \begin{bmatrix} 0 \\ 0 \\ dl1 \end{bmatrix} \quad (3.17)$$

Gelenk 2

$$T_2 = T_1 \cdot A_2 = \begin{bmatrix} \cos(\varphi_0) & 0 & \sin(\varphi_0) & 0 \\ \sin(\varphi_0) & 0 & -\cos(\varphi_0) & 0 \\ 0 & 1.0 & 0 & dl1 + dl2 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.18)$$

$$z_2 = \begin{bmatrix} \sin(\varphi_0) \\ -\cos(\varphi_0) \\ 0 \end{bmatrix}, \quad O_2 = \begin{bmatrix} 0 \\ 0 \\ dl1 + dl2 \end{bmatrix} \quad (3.19)$$

Gelenk 3

$$T_3 = T_2 \cdot A_3 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.20)$$

$$t_{11} = \cos(\varphi_0) \cos(\varphi_1)$$

$$t_{12} = -\sin(\varphi_0)$$

$$t_{13} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$t_{14} = L_1 \cos(\varphi_0) \cos(\varphi_1)$$

$$\begin{aligned}
 t_{21} &= \cos(\varphi_1) \sin(\varphi_0) \\
 t_{22} &= \cos(\varphi_0) \\
 t_{23} &= -\sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) \\
 t_{31} &= \sin(\varphi_1) \\
 t_{32} &= 0 \\
 t_{33} &= \cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + L_1 \sin(\varphi_1)
 \end{aligned}$$

$$z_3 = \begin{bmatrix} -\cos(\varphi_0) \sin(\varphi_1) \\ -\sin(\varphi_0) \sin(\varphi_1) \\ \cos(\varphi_1) \end{bmatrix}, \quad O_3 = \begin{bmatrix} L_1 \cos(\varphi_0) \cos(\varphi_1) \\ L_1 \cos(\varphi_1) \sin(\varphi_0) \\ dl_1 + dl_2 + L_1 \sin(\varphi_1) \end{bmatrix} \quad (3.21)$$

Gelenk 4

$$T_4 = T_3 \cdot A_4 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.22)$$

$$\begin{aligned}
 t_{11} &= \cos(\varphi_0) \cos(\varphi_1) \\
 t_{12} &= -\sin(\varphi_0) \\
 t_{13} &= -\cos(\varphi_0) \sin(\varphi_1) \\
 t_{14} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) \\
 t_{21} &= \cos(\varphi_1) \sin(\varphi_0) \\
 t_{22} &= \cos(\varphi_0) \\
 t_{23} &= -\sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 t_{31} &= \sin(\varphi_1) \\
 t_{32} &= 0 \\
 t_{33} &= \cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1)
 \end{aligned}$$

$$z_4 = \begin{bmatrix} -1.0 \cos(\varphi_0) \sin(\varphi_1) \\ -1.0 \sin(\varphi_0) \sin(\varphi_1) \\ 1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.23)$$

$$O_4 = \begin{bmatrix} L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) \\ L_1 \cos(\varphi_1) \sin(\varphi_0) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\ dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) \end{bmatrix} \quad (3.24)$$

Gelenk 5

$$T_5 = T_4 \cdot A_5 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.25)$$

$$\begin{aligned}
 t_{11} &= \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2) \\
 t_{12} &= \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \\
 t_{13} &= \cos(\varphi_0) \sin(\varphi_1) \\
 t_{14} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 t_{21} &= \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{22} &= \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2) \\
 t_{23} &= \sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{31} &= \cos(\varphi_2) \sin(\varphi_1) \\
 t_{32} &= \sin(\varphi_1) \sin(\varphi_2) \\
 t_{33} &= -\cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

Positionsvektor O_5

$$O_5 = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \quad (3.26)$$

$$\begin{aligned}
 o_x &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 o_y &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 o_z &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$z_5 = \begin{bmatrix} 1.0 \cos(\varphi_0) \sin(\varphi_1) \\ 1.0 \sin(\varphi_0) \sin(\varphi_1) \\ -1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.27)$$

Gelenk 6

$$T_6 = T_5 \cdot A_6 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.28)$$

$$\begin{aligned}
 t_{11} &= \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2) \\
 t_{12} &= \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \\
 t_{13} &= \cos(\varphi_0) \sin(\varphi_1) \\
 t_{14} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dl_3 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 t_{21} &= \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{22} &= \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2) \\
 t_{23} &= \sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dl_3 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{31} &= \cos(\varphi_2) \sin(\varphi_1) \\
 t_{32} &= \sin(\varphi_1) \sin(\varphi_2) \\
 t_{33} &= -\cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) \\
 &\quad - dl_3 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$O_6 = \begin{bmatrix} o_{x6} \\ o_{y6} \\ o_{z6} \end{bmatrix} \quad (3.29)$$

$$\begin{aligned}
 o_{x6} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dl_3 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 o_{y6} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dl_3 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)
 \end{aligned}$$

$$o_{z6} = dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dl_3 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)$$

$$z_6 = \begin{bmatrix} 1.0 \cos(\varphi_0) \sin(\varphi_1) \\ 1.0 \sin(\varphi_0) \sin(\varphi_1) \\ -1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.30)$$

Gelenk 7

$$T_7 = T_6 \cdot A_7 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.31)$$

$$\begin{aligned}
 t_{11} &= \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2) \\
 t_{12} &= \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \\
 t_{13} &= \cos(\varphi_0) \sin(\varphi_1) \\
 t_{14} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 t_{21} &= \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{22} &= \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2) \\
 t_{23} &= \sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{31} &= \cos(\varphi_2) \sin(\varphi_1) \\
 t_{32} &= \sin(\varphi_1) \sin(\varphi_2) \\
 t_{33} &= -\cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \\
 &\quad - dl_3 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$O_7 = \begin{bmatrix} o_{x7} \\ o_{y7} \\ o_{z7} \end{bmatrix} \quad (3.32)$$

$$\begin{aligned}
 o_{x7} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)
 \end{aligned}$$

$$\begin{aligned}
 o_{y7} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 o_{z7} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) - dl_3 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$z_7 = \begin{bmatrix} \cos(\varphi_0) \sin(\varphi_1) \\ \sin(\varphi_0) \sin(\varphi_1) \\ -\cos(\varphi_1) \end{bmatrix} \quad (3.33)$$

Gelenk 8

$$T_8 = T_7 \cdot A_8 = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \quad (3.34)$$

$$\begin{aligned}
 t_{11} &= \sin(\varphi_4) (\cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)) \\
 &\quad - \cos(\varphi_4) (\sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)) \\
 t_{12} &= \cos(\varphi_4) (\cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)) \\
 &\quad + \sin(\varphi_4) (\sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)) \\
 t_{13} &= \cos(\varphi_0) \sin(\varphi_1) \\
 t_{14} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 t_{21} &= \cos(\varphi_4) (\cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)) \\
 &\quad - \sin(\varphi_4) (\cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)) \\
 t_{22} &= -\cos(\varphi_4) (\cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)) \\
 &\quad - \sin(\varphi_4) (\cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)) \\
 t_{23} &= \sin(\varphi_0) \sin(\varphi_1) \\
 t_{24} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + dl_4 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 t_{31} &= \sin(\varphi_1) \sin(\varphi_2) \sin(\varphi_4) + \cos(\varphi_2) \cos(\varphi_4) \sin(\varphi_1) \\
 t_{32} &= \cos(\varphi_4) \sin(\varphi_1) \sin(\varphi_2) - \cos(\varphi_2) \sin(\varphi_1) \sin(\varphi_4) \\
 t_{33} &= -\cos(\varphi_1) \\
 t_{34} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \\
 &\quad - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$O_8 = \begin{bmatrix} o_{x8} \\ o_{y8} \\ o_{z8} \end{bmatrix} \quad (3.35)$$

$$\begin{aligned}
 o_{x8} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 o_{y8} &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + dl_4 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 o_{z8} &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_2) \sin(\varphi_1)
 \end{aligned}$$

$$z_8 = \begin{bmatrix} 1.0 \cos(\varphi_0) \sin(\varphi_1) \\ 1.0 \sin(\varphi_0) \sin(\varphi_1) \\ -1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.36)$$

Endeffektor-Position Die Position des Endeffektors O_{end} entspricht O_8 :

$$O_{\text{end}} = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \quad (3.37)$$

$$\begin{aligned} o_x &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\ &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\ o_y &= L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + dl_4 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\ o_z &= dl_1 + dl_2 + D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \\ &\quad - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \end{aligned}$$

3.2.2 Berechnung pro Gelenk

a sich das erste Gelenk im Ursprung des Basis-Koordinatensystems befindet, werden die folgenden anfänglichen Parameter definiert:

$$Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (3.38)$$

$$O_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.39)$$

Wobei:

- Z_0 repräsentiert den Einheitsvektor in Richtung der Achse des Anfangsgelenks (Z-Achse)
- O_0 bezeichnet die Position des Ursprungs des Basis-Koordinatensystems

Berechnung des Gelenks Nummer: 1

$$J_1 = \begin{bmatrix} z_1 \times (O_{\text{End}} - O_1) \\ z_1 \end{bmatrix} \quad (3.40)$$

$$z_1 = \begin{bmatrix} 0 \\ 0 \\ 1.0 \end{bmatrix} \quad (3.41)$$

- Kreuzprodukt $z_1 \times (O_{\text{End}} - O_1)$:

$$z_1 \times (O_{\text{End}} - O_1) = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \quad (3.42)$$

$$\begin{aligned} c_x &= D_1 \sin(\varphi_0) \sin(\varphi_1) - L_2 \cos(\varphi_0) \sin(\varphi_2) - L_1 \cos(\varphi_1) \sin(\varphi_0) \\ &\quad - dh_3 \sin(\varphi_0) \sin(\varphi_1) - dl_3 \sin(\varphi_0) \sin(\varphi_1) - dl_4 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad - L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \end{aligned}$$

$$\begin{aligned} c_y &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\ &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$c_z = 0$$

Rotatorisches Gelenk φ_0 :

$$J_{\varphi_0} = \begin{bmatrix} j_{11} \\ j_{21} \\ j_{31} \\ j_{41} \\ j_{51} \\ j_{61} \end{bmatrix} \quad (3.43)$$

$$\begin{aligned}
 j_{11} &= D_1 \sin(\varphi_0) \sin(\varphi_1) - L_2 \cos(\varphi_0) \sin(\varphi_2) - L_1 \cos(\varphi_1) \sin(\varphi_0) \\
 &\quad - dh_3 \sin(\varphi_0) \sin(\varphi_1) - dl_3 \sin(\varphi_0) \sin(\varphi_1) - dl_4 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad - L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \\
 j_{21} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\
 &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \\
 j_{31} &= 0 \\
 j_{41} &= 0 \\
 j_{51} &= 0 \\
 j_{61} &= 1
 \end{aligned}$$

Berechnung des Gelenks Nummer: 2

$$J_2 = \begin{bmatrix} z_2 \times (o_8 - o_2) \\ z_2 \end{bmatrix} \quad (3.44)$$

$$z_2 = \begin{bmatrix} \sin(\varphi_0) \\ -\cos(\varphi_0) \\ 0 \end{bmatrix} \quad (3.45)$$

- Kreuzprodukt $z_2 \times (O_{\text{End}} - O_2)$:

$$z_2 \times (O_{\text{End}} - O_2) = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \quad (3.46)$$

$$\begin{aligned}
 c_x &= -\cos(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \right. \\
 &\quad \left. - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\
 c_y &= -\sin(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \right. \\
 &\quad \left. - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\
 c_z &= \cos(\varphi_0) \cdot \left(L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \right. \\
 &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\
 &\quad \left. + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\
 &\quad + \sin(\varphi_0) \cdot \left(L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + dl_4 \sin(\varphi_0) \sin(\varphi_1) \\
 &\quad \left. + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right)
 \end{aligned}$$

Rotatorisches Gelenk φ_1 :

$$J_{\varphi_1} = \begin{bmatrix} j_{12} \\ j_{22} \\ j_{32} \\ j_{42} \\ j_{52} \\ j_{62} \end{bmatrix} \quad (3.47)$$

$$\begin{aligned} j_{12} &= -\cos(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \right. \\ &\quad \left. - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\ j_{22} &= -\sin(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \right. \\ &\quad \left. - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\ j_{32} &= \cos(\varphi_0) \cdot \left(L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \right. \\ &\quad \left. + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \right. \\ &\quad \left. + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\ &\quad + \sin(\varphi_0) \cdot \left(L_1 \cos(\varphi_1) \sin(\varphi_0) + L_2 \cos(\varphi_0) \sin(\varphi_2) - D_1 \sin(\varphi_0) \sin(\varphi_1) \right. \\ &\quad \left. + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) + dl_4 \sin(\varphi_0) \sin(\varphi_1) \right. \\ &\quad \left. + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right) \\ j_{42} &= \sin(\varphi_0) \\ j_{52} &= -\cos(\varphi_0) \\ j_{62} &= 0 \end{aligned}$$

Berechnung des Gelenks Nummer: 3

$$J_3 = \begin{bmatrix} z_4 \times (o_8 - o_4) \\ z_4 \end{bmatrix} \quad (3.48)$$

$$z_4 = \begin{bmatrix} -1.0 \cos(\varphi_0) \sin(\varphi_1) \\ -1.0 \sin(\varphi_0) \sin(\varphi_1) \\ 1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.49)$$

- Kreuzprodukt $z_4 \times (O_{\text{End}} - O_4)$:

$$z_4 \times (O_{\text{End}} - O_4) = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \quad (3.50)$$

$$\begin{aligned}
 d_x &= \sin(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_1) + dl_3 \cos(\varphi_1) + dl_4 \cos(\varphi_1) - L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\
 &\quad - \cos(\varphi_1) \cdot \left(L_2 \cos(\varphi_0) \sin(\varphi_2) + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right) \\
 d_y &= \cos(\varphi_1) \cdot \left(dh_3 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) + dl_3 \cos(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\
 &\quad - \cos(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_1) + dl_3 \cos(\varphi_1) + dl_4 \cos(\varphi_1) - L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\
 d_z &= \sin(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) + dl_3 \cos(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\
 &\quad - \cos(\varphi_0) \sin(\varphi_1) \cdot \left(L_2 \cos(\varphi_0) \sin(\varphi_2) + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right)
 \end{aligned}$$

Rotatorisches Gelenk φ_2 :

$$J_{\varphi_2} = \begin{bmatrix} j_{13} \\ j_{23} \\ j_{33} \\ j_{43} \\ j_{53} \\ j_{63} \end{bmatrix} \quad (3.51)$$

$$\begin{aligned}
 j_{13} &= \sin(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_1) + dl_3 \cos(\varphi_1) + dl_4 \cos(\varphi_1) - L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \\
 &\quad - \cos(\varphi_1) \cdot \left(L_2 \cos(\varphi_0) \sin(\varphi_2) + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right)
 \end{aligned}$$

$$\begin{aligned}
 j_{23} &= \cos(\varphi_1) \cdot \left(dh_3 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) + dl_3 \cos(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\
 &\quad - \cos(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_1) + dl_3 \cos(\varphi_1) + dl_4 \cos(\varphi_1) - L_2 \cos(\varphi_2) \sin(\varphi_1) \right)
 \end{aligned}$$

$$\begin{aligned}
 j_{33} &= \sin(\varphi_0) \sin(\varphi_1) \cdot \left(dh_3 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) + dl_3 \cos(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \cos(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \right) \\
 &\quad - \cos(\varphi_0) \sin(\varphi_1) \cdot \left(L_2 \cos(\varphi_0) \sin(\varphi_2) + dh_3 \sin(\varphi_0) \sin(\varphi_1) + dl_3 \sin(\varphi_0) \sin(\varphi_1) \right. \\
 &\quad \left. + dl_4 \sin(\varphi_0) \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \right)
 \end{aligned}$$

$$\begin{aligned}
 j_{43} &= -\cos(\varphi_0) \sin(\varphi_1) \\
 j_{53} &= -\sin(\varphi_0) \sin(\varphi_1) \\
 j_{63} &= \cos(\varphi_1)
 \end{aligned}$$

Berechnung des Gelenks Nummer: 4

$$J_4 = \begin{bmatrix} z_6 \\ \mathbf{0} \end{bmatrix} \quad (3.52)$$

$$z_6 = \begin{bmatrix} 1.0 \cos(\varphi_0) \sin(\varphi_1) \\ 1.0 \sin(\varphi_0) \sin(\varphi_1) \\ -1.0 \cos(\varphi_1) \end{bmatrix} \quad (3.53)$$

Prismatisches Gelenk d_3 :

$$J_{d_3} = \begin{bmatrix} j_{14} \\ j_{24} \\ j_{34} \\ j_{44} \\ j_{54} \\ j_{64} \end{bmatrix} \quad (3.54)$$

$$j_{14} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{24} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{34} = -\cos(\varphi_1)$$

$$j_{44} = 0$$

$$j_{54} = 0$$

$$j_{64} = 0$$

Berechnung des Gelenks Nummer: 5

$$J_5 = \begin{bmatrix} z_7 \times (o_8 - o_7) \\ z_7 \end{bmatrix} \quad (3.55)$$

$$z_7 = \begin{bmatrix} \cos(\varphi_0) \sin(\varphi_1) \\ \sin(\varphi_0) \sin(\varphi_1) \\ -\cos(\varphi_1) \end{bmatrix} \quad (3.56)$$

- Kreuzprodukt $z_7 \times (O_{\text{End}} - O_7)$:

$$z_7 \times (O_{\text{End}} - O_7) = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \quad (3.57)$$

$$e_x = 0$$

$$e_y = 0$$

$$e_z = 0$$

Rotatorisches Gelenk φ_4 :

$$J_{\varphi_4} = \begin{bmatrix} j_{15} \\ j_{25} \\ j_{35} \\ j_{45} \\ j_{55} \\ j_{65} \end{bmatrix} \quad (3.58)$$

$$\begin{aligned}j_{15} &= 0 \\j_{25} &= 0 \\j_{35} &= 0 \\j_{45} &= \cos(\varphi_0) \sin(\varphi_1) \\j_{55} &= \sin(\varphi_0) \sin(\varphi_1) \\j_{65} &= -\cos(\varphi_1)\end{aligned}$$

3.2.3 Jacobi-Matrix des Roboters

Die vollständige geometrische Jacobi-Matrix ergibt sich durch die Kombination aller Beiträge der rotatorischen und prismatischen Gelenke bis zum Endeffektor. Sie hat die folgende Form:

$$J = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (3.59)$$

$$\begin{aligned} j_{11} &= D_1 \sin(\varphi_0) \sin(\varphi_1) - L_2 \cos(\varphi_0) \sin(\varphi_2) - L_1 \cos(\varphi_1) \sin(\varphi_0) \\ &\quad - dh_3 \sin(\varphi_0) \sin(\varphi_1) - dl_3 \sin(\varphi_0) \sin(\varphi_1) - dl_4 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad - L_2 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \end{aligned}$$

$$\begin{aligned} j_{12} &= -\cos(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) \right. \\ &\quad \left. - dl_3 \cos(\varphi_1) - dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \end{aligned}$$

$$j_{13} = -L_2 \cdot \left(\cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \right)$$

$$j_{14} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{15} = 0$$

$$\begin{aligned} j_{21} &= L_1 \cos(\varphi_0) \cos(\varphi_1) - D_1 \cos(\varphi_0) \sin(\varphi_1) - L_2 \sin(\varphi_0) \sin(\varphi_2) \\ &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + dl_3 \cos(\varphi_0) \sin(\varphi_1) + dl_4 \cos(\varphi_0) \sin(\varphi_1) \\ &\quad + L_2 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$\begin{aligned} j_{22} &= -\sin(\varphi_0) \cdot \left(D_1 \cos(\varphi_1) + L_1 \sin(\varphi_1) - dh_3 \cos(\varphi_1) - dl_3 \cos(\varphi_1) - \right. \\ &\quad \left. dl_4 \cos(\varphi_1) + L_2 \cos(\varphi_2) \sin(\varphi_1) \right) \end{aligned}$$

$$j_{23} = L_2 \cdot \left(\cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) \right)$$

$$j_{24} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{25} = 0$$

$$j_{31} = 0$$

$$\begin{aligned} j_{32} &= L_1 \cos(\varphi_1) - D_1 \sin(\varphi_1) + dh_3 \sin(\varphi_1) + dl_3 \sin(\varphi_1) + \\ &\quad dl_4 \sin(\varphi_1) + L_2 \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$j_{33} = -L_2 \sin(\varphi_1) \sin(\varphi_2)$$

$$j_{34} = -\cos(\varphi_1)$$

$$j_{35} = 0$$

$$j_{41} = 0$$

$$j_{42} = \sin(\varphi_0)$$

$$j_{43} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$j_{44} = 0$$

$$j_{45} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{61} = 1.0$$

$$j_{63} = \cos(\varphi_1)$$

$$j_{51} = 0$$

$$j_{52} = -\cos(\varphi_0)$$

$$j_{53} = -\sin(\varphi_0) \sin(\varphi_1)$$

$$j_{54} = 0$$

$$j_{55} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{62} = 0$$

$$j_{64} = 0$$

$$j_{65} = -\cos(\varphi_1)$$

Jacobi-Matrix am TCP mit eingesetzten geometrischen Konstanten Die nachfolgende Jacobi-Matrix beschreibt die geometrische Beziehung zwischen Gelenkwinkeln und der Bewegung des Endeffektors (TCP). Alle geometrischen Konstanten wurden bereits ersetzt. Die Matrix ergibt sich als:

$$J_{\text{eval}} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (3.60)$$

Einträge der Matrix:

$$\begin{aligned} j_{11} &= -0.4 \cos(\varphi_1) \sin(\varphi_0) - 0.5 \cos(\varphi_0) \sin(\varphi_2) - 1.0 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad - 1.0 dh_3 \sin(\varphi_0) \sin(\varphi_1) - 0.5 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \end{aligned}$$

$$j_{12} = 1.0 \cos(\varphi_0) (1.0 \cos(\varphi_1) - 0.4 \sin(\varphi_1) - 0.5 \cos(\varphi_2) \sin(\varphi_1) + 1.0 dh_3 \cos(\varphi_1))$$

$$j_{13} = -0.5 \cos(\varphi_2) \sin(\varphi_0) - 0.5 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$j_{14} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{15} = 0$$

$$\begin{aligned} j_{21} &= 0.4 \cos(\varphi_0) \cos(\varphi_1) + \cos(\varphi_0) \sin(\varphi_1) - 0.5 \sin(\varphi_0) \sin(\varphi_2) \\ &\quad + dh_3 \cos(\varphi_0) \sin(\varphi_1) + 0.5 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$j_{22} = 1.0 \sin(\varphi_0) (1.0 \cos(\varphi_1) - 0.4 \sin(\varphi_1) - 0.5 \cos(\varphi_2) \sin(\varphi_1) + 1.0 dh_3 \cos(\varphi_1))$$

$$j_{23} = 0.5 \cos(\varphi_0) \cos(\varphi_2) - 0.5 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)$$

$$j_{24} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{25} = 0$$

$$j_{31} = 0$$

$$j_{32} = 0.4 \cos(\varphi_1) + \sin(\varphi_1) + 0.5 \cos(\varphi_1) \cos(\varphi_2) + dh_3 \sin(\varphi_1)$$

$$j_{33} = -0.5 \sin(\varphi_1) \sin(\varphi_2)$$

$$j_{34} = -\cos(\varphi_1)$$

$$j_{35} = 0$$

$$j_{41} = 0$$

$$j_{42} = \sin(\varphi_0)$$

$$j_{43} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$j_{44} = 0$$

$$j_{45} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{61} = 1.0$$

$$j_{63} = \cos(\varphi_1)$$

$$j_{51} = 0$$

$$j_{52} = -\cos(\varphi_0)$$

$$j_{53} = -\sin(\varphi_0) \sin(\varphi_1)$$

$$j_{54} = 0$$

$$j_{55} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{62} = 0$$

$$j_{64} = 0$$

$$j_{65} = -\cos(\varphi_1)$$

3.3 Singularitäten

3.3.1 Singulärwertzerlegung (SVD)

Für Roboter, die redundant sind, ist die Jacobimatrix \mathbf{J} nicht quadratisch. In solchen Fällen sind klassische Methoden wie Determinanten, Eigenwerte oder Eigenvektoren zur Untersuchung von Singularitäten nicht direkt anwendbar. Die Singulärwertzerlegung (SVD) hingegen stellt eine robuste und allgemeine Methode dar, um auch für nicht-quadratische Matrizen sinnvolle Aussagen über das Verhalten des Roboters in der Nähe von Singularitäten zu machen. Sie erlaubt insbesondere die Bestimmung von Rangverlust, Redundanzen und die Analyse der Bewegungsfreiheit durch die Untersuchung der Singularwerte. Daher ist die SVD der geeignetste und stabilste Ansatz zur Analyse von Singularitäten bei nicht-quadratischen Jacobimatrizen.

Wie zuvor beschrieben, gilt für $\mathbf{J} \in \mathbb{R}^{m \times n}$, dass $\mathbf{J}\mathbf{J}^T \in \mathbb{R}^{m \times m}$. Diese quadratische Matrix hat Eigenwerte und Eigenvektoren, die die folgende Gleichung erfüllen:

$$\mathbf{J}\mathbf{J}^T u_i = \lambda_i u_i \quad (3.61)$$

Dabei sind λ_i und u_i die zugehörigen Eigenwerte und Eigenvektoren von $\mathbf{J}\mathbf{J}^T$. Diese Gleichung lässt sich umformen zu:

$$\mathbf{J}\mathbf{J}^T u_i - \lambda_i u_i = 0 \quad (3.62)$$

$$(\mathbf{J}\mathbf{J}^T - \lambda_i \mathbf{I}) u_i = 0 \quad (3.63)$$

Dies bedeutet, dass die Matrix $(\mathbf{J}\mathbf{J}^T - \lambda_i \mathbf{I})$ singulär ist, und ihre Determinante daher verschwindet:

$$\det(\mathbf{J}\mathbf{J}^T - \lambda_i \mathbf{I}) = 0 \quad (3.64)$$

Wir können Gleichung (3.64) verwenden, um die Eigenwerte $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m \geq 0$ von $\mathbf{J}\mathbf{J}^T$ zu berechnen. Die Singularwerte σ_i der Jacobimatrix \mathbf{J} ergeben sich aus den Quadratwurzeln der Eigenwerte:

$$\sigma_i = \sqrt{\lambda_i} \quad (3.65)$$

Die Singulärwertzerlegung der Matrix \mathbf{J} lautet:

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T \quad (3.66)$$

Dabei sind $\mathbf{U} = [u_1 \ u_2 \ \dots \ u_m]$ und $\mathbf{V} = [v_1 \ v_2 \ \dots \ v_n]$ orthogonale Matrizen und $\Sigma \in \mathbb{R}^{m \times n}$ eine Diagonalmatrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & \\ & & \ddots & \\ 0 & & & \sigma_m \end{bmatrix} \quad (3.67)$$

Die Singularwerte σ_i werden aus (3.64) und (3.65) bestimmt. Diese Werte erlauben es, die Eigenvektoren u_i zu berechnen, die wiederum die Matrix $\mathbf{U} = [u_1 \ \dots \ u_m]$ bilden. Daraus ergibt sich:

$$\mathbf{J}\mathbf{J}^T u_i = \sigma_i^2 u_i \quad (3.68)$$

und die Matrixgleichung:

$$\mathbf{J}\mathbf{J}^T \mathbf{U} = \mathbf{U}\Sigma^2 \quad (3.69)$$

mit:

$$\Sigma_m = \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad (3.70)$$

Nun definieren wir

$$V_m = \mathbf{J}^T \mathbf{U} \Sigma_m^{-1} \quad (3.71)$$

und lassen \mathbf{V} eine orthogonale Matrix sein, die folgende Form erfüllt:

$$\mathbf{V} = [V_m \ V_{n-m}]$$

(wobei V_{n-m} gerade so viele Spalten enthält, dass \mathbf{V} eine $n \times n$ -Matrix ist).

Diese Form erlaubt die Kombination aller Komponenten zur Verifikation von Gleichung (3.66):

$$\mathbf{U} \Sigma \mathbf{V}^T = \mathbf{U}_{[2m]} [\Sigma_m \ 0] \begin{bmatrix} \mathbf{V}_m^T \\ \mathbf{V}_{n-m}^T \end{bmatrix} \quad (3.72)$$

$$= \mathbf{U}_{2m} \mathbf{V}^T \quad (3.73)$$

$$= \Sigma_m (\mathbf{J}^T \mathbf{U}_m) \quad (3.74)$$

$$= \mathbf{U}_m \Sigma_m^{-1} \mathbf{U}_m^T \mathbf{J} \quad (3.75)$$

$$= \mathbf{U} \mathbf{U}^T \mathbf{J} \quad (3.76)$$

$$= \mathbf{J} \quad (3.77)$$

3.3.2 Ermittlung von Singularitäten

Da die symbolische Berechnung der Singulärwertzerlegung (SVD) einer nicht-quadratischen Jacobi-matrix sehr aufwendig und speicherintensiv ist, verfolgen wir alternativ die Methode der numerischen Konvergenz auf Singularitäten.

Nach der Referenz [3] treten Hauptsingularitäten häufig in bestimmten Gelenkkonfigurationen auf. Insbesondere werden die folgenden Konfigurationen zur Bewertung der Singulärität herangezogen:

- **Konfiguration 1 – Voll ausgestreckter Arm (empfohlen in der Literatur):**

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 0.00^\circ, \quad \varphi_2 = 0.00^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = 0.00^\circ$$

- **Konfiguration 2 – Alternative empfohlene Singularitätsstellung:**

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 90.00^\circ, \quad \varphi_2 = 0.00^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = 0.00^\circ$$

- **Konfiguration 3 – Arbeitskonfiguration zur Kraft-/Momentenübertragung:**

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 0.00^\circ, \quad \varphi_2 = -126.87^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = -66.87^\circ$$

Diese drei Stellungen werden verwendet, um die Positionen numerisch in die Jacobimatrix $\mathbf{J}(\varphi_0, \dots, \varphi_4)$ einzusetzen und deren Singulärwerte zu analysieren.

Um die Singularitäten zu bestätigen und deren Auswirkungen auf die Bewegungs- und Übertragungsfähigkeiten des Roboters zu verstehen, gehen wir im nächsten Abschnitt mathematisch wie folgt vor:

1. **Numerischer Ersatz in die Jacobimatrix:** Setzen der obigen Gelenkwinkel in die Jacobimatrix $\mathbf{J}(\varphi_0, \dots, \varphi_4)$, um die Matrix an diesem Punkt zu evaluieren.

2. **Bildung des Produkts $\mathbf{J}\mathbf{J}^T$:** Da \mathbf{J} nicht quadratisch ist, berechnen wir die quadratische Matrix:

$$\mathbf{A} = \mathbf{J}\mathbf{J}^T$$

3. **Bestimmung der Eigenwerte von \mathbf{A} :** Löse:

$$\det(\mathbf{J}\mathbf{J}^T - \lambda\mathbf{I}) = 0$$

um die Eigenwerte λ_i zu erhalten.

4. **Berechnung der Singulärwerte:** Die Singulärwerte σ_i der Jacobimatrix sind die Quadratwurzeln der Eigenwerte:

$$\sigma_i = \sqrt{\lambda_i}$$

5. **Bewertung der Bewegungsfreiheit:** Eine Singulärität liegt dann vor, wenn mindestens ein $\sigma_i = 0$ ist, d. h. wenn die Jacobimatrix Rang verliert. Dies bedeutet, dass der Roboter sich in bestimmten Richtungen nicht mehr bewegen kann.

6. **Zusammensetzung der SVD:** Falls gewünscht, kann nun die vollständige Zerlegung berechnet werden:

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$$

wobei Σ eine Diagonalmatrix mit σ_i ist und \mathbf{U}, \mathbf{V} orthogonale Matrizen darstellen.

7. **Interpretation:** Kleine oder verschwindende Singulärwerte zeigen Richtungen an, in denen keine Kraft oder Geschwindigkeit übertragen werden kann – ein typisches Merkmal für eine kinematische Singularität.

3.3.3 Berechnung von Singularitäten

Analyse der Singularität – Konfiguration 1

Diese Analyse basiert auf der in [3] empfohlenen Konfiguration mit vollständig ausgestrecktem Arm:

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 0.00^\circ, \quad \varphi_2 = 0.00^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = 0.00^\circ$$

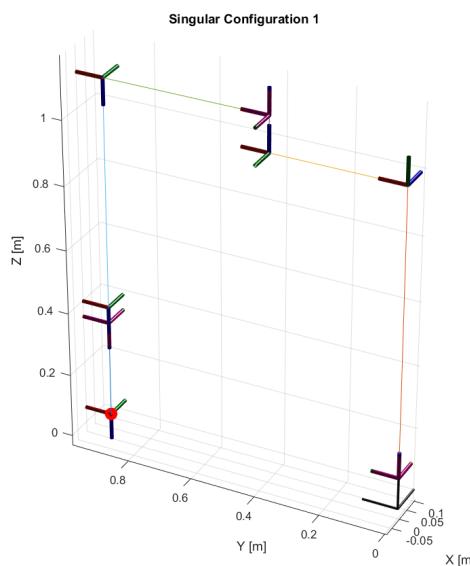


Abbildung 3.3: Singularitätsprüfung für Konfiguration 1

Evaluerte Jacobimatrix \mathbf{J}

Nach dem Einsetzen der obigen Winkel ergibt sich die numerische Form der Jacobimatrix wie folgt:

$$\mathbf{J} = \begin{bmatrix} -0,9000 & 0 & -0,5000 & 0 & 0 \\ 0 & 0,9500 & 0 & 0 & 0 \\ 0 & 0,9000 & 0 & -1,0000 & 0 \\ 0 & 1,0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1,0000 & 0 & 1,0000 & 0 & -1,0000 \end{bmatrix}$$

Singulärwertzerlegung (SVD):

Da \mathbf{J} eine nicht-quadratische Matrix ist, verwenden wir die SVD:

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$$

Die Singulärwerte σ_i der Matrix \mathbf{J} lauten:

$$\sigma_1 = 1,9321, \quad \sigma_2 = 1,7603, \quad \sigma_3 = 0,7836, \quad \sigma_4 = 0,5717, \quad \sigma_5 = 0,0000$$

Interpretation der Singulärwerte:

- *Kinematische Singularität erkannt:* Der kleinste Singulärwert ist exakt null: $\sigma_5 = 0.0000$. Die Jacobimatrix \mathbf{J} hat daher nicht vollen Rang.
- In physikalischer Hinsicht bedeutet dies, dass es mindestens eine Richtung gibt, in der der Endeffektor keine Bewegung erzeugen kann — eine klassische kinematische Singularität.
- Die Konditionszahl der Matrix ist sehr groß ($\kappa = 9,85 \cdot 10^{16}$), was auf eine schlechte numerische Stabilität und nahezu lineare Abhängigkeit der Spaltenvektoren hinweist.

Richtung maximaler Einschränkung:

Die Richtung im Arbeitsraum, in der keine Geschwindigkeit erzeugt werden kann (nulles Singulärwert), ergibt sich aus dem letzten Spaltenvektor von \mathbf{U} :

$$u_5 = [0,00 \ 0,72 \ -0,00 \ -0,69 \ 0,00 \ 0,00]^T$$

Diese Richtung beschreibt den Bewegungsraum, der durch die Roboterstruktur blockiert wird. Dies ist besonders kritisch bei Aufgaben, bei denen Kräfte oder Bewegungen exakt in dieser Richtung erforderlich wären.

Kinematische Singularität: Gestreckte Konfiguration Eine charakteristische Singularität des SCARA-Roboters tritt auf, wenn die beiden rotatorischen Gelenke φ_0 und φ_2 so ausgerichtet sind, dass die Glieder gestreckt in einer Linie liegen, z. B. bei $\varphi_2 = -\varphi_0$ oder $\varphi_2 = \pi + \varphi_0$. In dieser Konfiguration sind die Achsen der ersten beiden Gelenke kollinear, wodurch ein Freiheitsgrad verloren geht.

- Der Arbeitsbereich kollabiert lokal; bestimmte Richtungen der Endeffektorbewegung sind nicht mehr unabhängig steuerbar.
- Besonders betroffen ist die translatorische Bewegung in der XY-Ebene orthogonal zur gestreckten Linie.
- Mathematisch liegt eine lineare Abhängigkeit der Spalten der Jacobi-Matrix \mathbf{J} vor; ein Singulärwert $\sigma_i = 0$ bestätigt dies.
- Die Zustandszahl $\kappa(\mathbf{J})$ steigt stark an ($\kappa \gg 10^{10}$), was numerische Instabilität bei Inversion und Steuerung bedeutet.

Analyse der Singularität – Konfiguration 2

Diese Analyse basiert auf der in [3] beschriebenen zweiten Konfiguration mit angewinkeltem ersten Gelenk:

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 90.00^\circ, \quad \varphi_2 = 0.00^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = 0.00^\circ$$

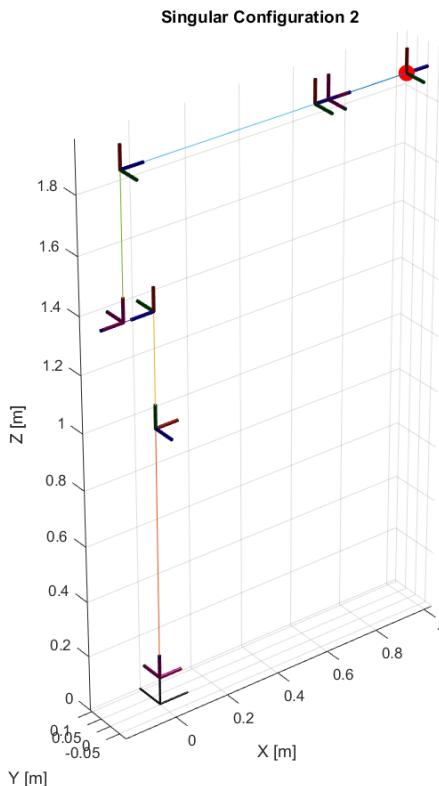


Abbildung 3.4: Singularitätsprüfung für Konfiguration 2

Evaluerte Jacobimatrix \mathbf{J} :

Nach dem Einsetzen der obigen Gelenkwinkel ergibt sich die Jacobimatrix numerisch wie folgt:

$$\mathbf{J} = \begin{bmatrix} -0,9500 & 0 & -0,5000 & 0 & 0 \\ 0 & -0,9000 & 0 & 1,0000 & 0 \\ 0 & 0,9500 & 0 & 0 & 0 \\ 0 & 1,0000 & 0 & 0 & 0 \\ 0 & 0 & -1,0000 & 0 & 1,0000 \\ 1,0000 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Singulärwertzerlegung (SVD):

Da \mathbf{J} eine nicht-quadratische Matrix ist, verwenden wir die Singulärwertzerlegung:

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$$

Die berechneten Singulärwerte σ_i lauten:

$$\sigma_1 = 1,7603, \quad \sigma_2 = 1,5516, \quad \sigma_3 = 1,2975, \quad \sigma_4 = 0,7836, \quad \sigma_5 = 0,2484$$

Interpretation der Singulärwerte:

- *Keine Singularität erkannt:* Alle Singulärwerte sind ungleich null, d. h. die Jacobimatrix \mathbf{J} hat vollen Rang.
- Aus Sicht der Kinematik bedeutet dies, dass der Endeffektor in alle Richtungen im Arbeitsraum eine Geschwindigkeit erzeugen kann.
- Die Konditionszahl der Matrix beträgt $\kappa = 7,09$, was auf eine gute numerische Stabilität hinweist.

Richtung maximaler Einschränkung:

Da kein Singulärwert null ist, existiert keine vollständig blockierte Richtung im Geschwindigkeitsraum. Dennoch ist die Beweglichkeit in Richtung des kleinsten Singulärwerts $\sigma_5 = 0,2484$ am stärksten eingeschränkt.

Analyse der Singularität – Konfiguration 3

Diese Analyse basiert auf einer komplexeren Haltung mit negativer zweiten und vierten Gelenkstellung:

$$\varphi_0 = 90.00^\circ, \quad \varphi_1 = 0.00^\circ, \quad \varphi_2 = -126.87^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = -66.87^\circ$$

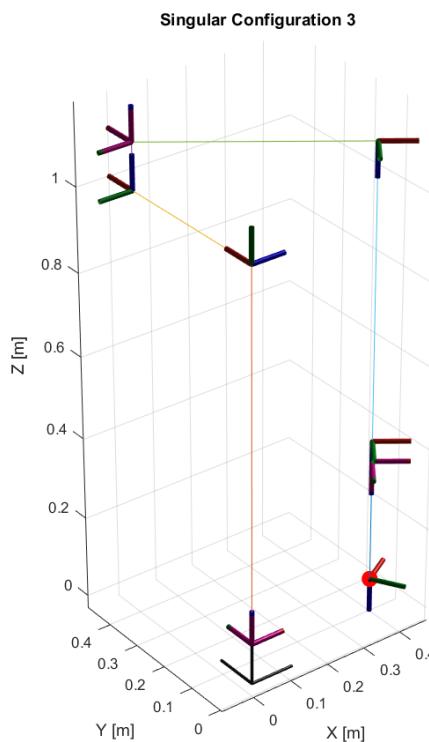


Abbildung 3.5: Singularitätsprüfung für Konfiguration 3

Evaluerte Jacobimatrix \mathbf{J} :

Nach dem Einsetzen der obigen Gelenkwinkel ergibt sich die Jacobimatrix numerisch wie folgt:

$$\mathbf{J} = \begin{bmatrix} -0,1000 & 0 & 0,3000 & 0 & 0 \\ 0,4000 & 0,9500 & 0,4000 & 0 & 0 \\ 0 & 0,1000 & 0 & -1,0000 & 0 \\ 0 & 1,0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1,0000 & 0 & 1,0000 & 0 & -1,0000 \end{bmatrix}$$

Singulärwertzerlegung (SVD):

Da \mathbf{J} eine nicht-quadratische Matrix ist, verwenden wir die Singulärwertzerlegung:

$$\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$$

Die berechneten Singulärwerte σ_i lauten:

$$\sigma_1 = 1,8381, \quad \sigma_2 = 1,3514, \quad \sigma_3 = 0,9946, \quad \sigma_4 = 0,3056, \quad \sigma_5 = 0,2119$$

Interpretation der Singulärwerte:

- *Keine Singularität erkannt:* Alle Singulärwerte sind ungleich null, d. h. die Jacobimatrix \mathbf{J} hat vollen Rang.
- Die Konfiguration ermöglicht dem Endeffektor Bewegungen in allen Richtungen des Arbeitsraums.
- Die Konditionszahl beträgt $\kappa = 8,68$, was auf eine akzeptable numerische Stabilität hinweist.

Richtung maximaler Einschränkung:

Obwohl kein Singulärwert null ist, weist der kleinste Wert $\sigma_5 = 0,2119$ auf eine Richtung mit eingeschränkter Beweglichkeit hin.

3.3.4 Zusammenfassung der Singularitätsanalyse mit Singulärwertzerlegung (SVD)

Konfiguration	Gelenkwinkel	Singulärwerte (SVD)	Singularität
1	$\varphi_0 = 90.00^\circ$ $\varphi_1 = 0.00^\circ$ $\varphi_2 = 0.00^\circ$ $dh_3 = -0,0500 \text{ m}$ $\varphi_4 = 0.00^\circ$	1.9321 1.7603 0.7836 0.5717 0.0000	Ja (Singularität)
2	$\varphi_0 = 90.00^\circ$ $\varphi_1 = 90.00^\circ$ $\varphi_2 = 0.00^\circ$ $dh_3 = -0,0500 \text{ m}$ $\varphi_4 = 0.00^\circ$	1.7603 1.5516 1.2975 0.7836 0.2484	Nein
3	$\varphi_0 = 90.00^\circ$ $\varphi_1 = 0.00^\circ$ $\varphi_2 = -126.87^\circ$ $dh_3 = -0,0500 \text{ m}$ $\varphi_4 = -66.87^\circ$	1.8381 1.3514 0.9946 0.3056 0.2119	Nein

Tabelle 3.2: Analyse der Singularitäten mithilfe der Singulärwertzerlegung (SVD) für drei verschiedene Konfigurationen

3.4 Berechnung der Gelenkkräfte und -momente

In dieser Sektion wird die Berechnung der resultierenden Gelenkgrößen τ anhand der transponierten Jacobi-Matrix beschrieben. Dabei wird davon ausgegangen, dass eine Kraft- und Momentenlast \mathbf{F}_{TCP} am Endeffektor wirkt. Die Jacobi-Matrix \mathbf{J} beschreibt die kinematische Beziehung zwischen den Gelenkgeschwindigkeiten und der Endeffektorbewegung. Die Gelenkmomente und -kräfte ergeben sich über:

$$\boldsymbol{\tau} = \mathbf{J}^\top \cdot \mathbf{F}_{TCP, \text{ordered}} \quad (3.78)$$

Dabei ist $\mathbf{F}_{TCP, \text{ordered}}$ die umgeordnete Kraft-Momenten-Vektor in der Form:

$$\mathbf{F}_{TCP} = \begin{bmatrix} M_x \\ M_y \\ M_z \\ F_x \\ F_y \\ F_z \end{bmatrix} \Rightarrow \mathbf{F}_{TCP, \text{ordered}} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{bmatrix}$$

Konfiguration 3

$$\varphi_0 = 90,00^\circ, \quad \varphi_1 = 0,00^\circ, \quad \varphi_2 = -126,87^\circ, \quad dh_3 = -0,0500 \text{ m}, \quad \varphi_4 = -66,87^\circ$$

Jacobi-Matrix \mathbf{J} :

$$\mathbf{J} = \begin{bmatrix} -0,1000 & 0,0000 & 0,3000 & 0,0000 & 0,0000 \\ 0,4000 & 0,9500 & 0,4000 & 0,0000 & 0,0000 \\ 0,0000 & 0,1000 & 0,0000 & -1,0000 & 0,0000 \\ 0,0000 & 1,0000 & 0,0000 & 0,0000 & 0,0000 \\ 0,0000 & 0,0000 & 0,0000 & 0,0000 & 0,0000 \\ 1,0000 & 0,0000 & 1,0000 & 0,0000 & -1,0000 \end{bmatrix}$$

Aufgebrachte Kraft und Momente am TCP:

$$\mathbf{F}_{TCP} = \begin{bmatrix} 30 \\ 0 \\ 100 \\ -50 \\ 25 \\ 75 \end{bmatrix} \Rightarrow \mathbf{F}_{TCP, \text{ordered}} = \begin{bmatrix} -50 \\ 25 \\ 75 \\ 30 \\ 0 \\ 100 \end{bmatrix}$$

Berechnung von $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \mathbf{J}^\top \cdot \mathbf{F}_{TCP, \text{ordered}} = \begin{bmatrix} 115,00 \\ 61,25 \\ 95,00 \\ -75,00 \\ -100,00 \end{bmatrix}$$

Gelenk Nr.	Typ	Symbol	τ_i	Einheit	Wert
1	Rotatorisch	φ_0	115,00	Nm	90,00°
2	Rotatorisch	φ_1	61,25	Nm	0,00°
3	Rotatorisch	φ_2	95,00	Nm	-126,87°
4	Prismatisch	dh_3	-75,00	N	-0,0500 m
5	Rotatorisch	φ_4	-100,00	Nm	-66,87°

Tabelle 3.3: Berechnete Gelenkmomente, -kräfte und Stellungen für Konfiguration 3

Interpretation der Ergebnisse: Diese Werte stellen die erforderlichen Gelenkkräfte und -momente dar, um die am Endeffektor angreifende Kraft-Momenten-Kombination im statischen Gleichgewicht zu halten.

3.5 Matlab Code Jacobi:

Der folgende MATLAB-Code löst das Problem der Jacobi:

```
1 clc; clear;
2 warning('off','all');
3 syms Hs H0 L0 L1 D1 L2 DA2 L3 H3 L5 L4 KH3
4 syms phi0 phi1 phi2 phi3 phi4 phi5 phi6 % all in radians now
5 syms d11 d12 d13 d14 dh3
6 syms d2 dh d6 real
7 syms nx ny nz ux uy uz ax ay az px py pz real
8 syms a1 a2 a3 real
9 pi = sym(pi);
10 % Define target variables
11 target_vars = [phi0, phi1, phi2, dh3, phi4];
12 target_var_names = {'phi0', 'phi1', 'phi2', 'dh3', 'phi4'};
13 alpha = [ 0, pi/2, -pi/2, 0, pi, 0, 0, 0];
14 L = [ 0, 0, L1, 0, L2, 0, 0, 0];
15 D = [d11, d12, 0, D1, 0, d13, dh3, d14];
16 phi = [0, phi0, phi1, 0, phi2, 0, 0, phi4]; % in radians
17 % Number of joints
18 n = length(alpha);
19 % ----- Homogeneous transformation matrix function -----
20 getA = @(alpha, L, D, phi) ...
21     [ cos(phi), -cos(alpha)*sin(phi), sin(alpha)*sin(phi), L*cos(phi);
22      sin(phi), cos(alpha)*cos(phi), -sin(alpha)*cos(phi), L*sin(phi);
23      0, sin(alpha), cos(alpha), D;
24      0, 0, 0, 1];
25 % ----- Step-by-step transformation matrices -----
26 A = sym(zeros(4,4,n)); % Symbolic 3D array
27 for i = 1:n
28     A_i = getA(alpha(i), L(i), D(i), phi(i));
29     A_i_simple = simplify(A_i, 'Steps', 100);
30     A_i_clean = vpa(A_i_simple, 6); % Round to 6 decimal places
31     A(:,:,i) = A_i_clean;
32 end
33 % ----- Final total transformation -----
34 T_total = eye(4);
35 for i = 1:n
36     T_total = T_total * A(:,:,i);
37 end
38 % ----- Accumulated z_i and o_i calculation -----
39 z_list = sym(zeros(3, n));
40 o_list = sym(zeros(3, n));
41 T = eye(4);
42 for i = 1:n
43     T = T * A(:,:,i);
44     fprintf('\n>> Accumulated Transformation T%d (%d)\n', i, i);
45     disp(T); % <<< here you see the accumulated transformed matrix
46     z_list(:,i) = T(1:3, 3); % third rotation column (z-axis)
47     o_list(:,i) = T(1:3, 4); % position
48
49 % Display z_i and o_i
50 fprintf('\n== Transformation Matrix A%d ==\n', i);
51 disp(A(:,:,i));
52 fprintf('z%d (z-axis after %d)\n', i, i);
53 disp(z_list(:,i));
54 fprintf('o%d (position after %d)\n', i, i);
55 disp(o_list(:,i));
56
57 end
58 % End-effector final position
59 o_end = T_total(1:3, 4);
60 fprintf('\n== Final End-effector Position: o_end ==\n');
```

```

61 disp(o_end);
62 % Indices of joints with mobile variables
63 indices_vars = [2, 3, 4, 7, 8];
64 fprintf('\n==== Variables considered for the Jacobian ====\n');
65 disp(target_var_names);
66 % Initialize Jacobian
67 Jg = sym(zeros(6, length(target_vars))); % 6 rows (3 linear + 3 angular)
68 % Add base frame z and p (needed for first joint calculation)
69 z_prev = [0; 0; 1];
70 o_prev = [0; 0; 0];
71 fprintf("\n Z_0:");
72 disp(z_prev);
73 fprintf("\n O_0:");
74 disp(o_prev)
75 for idx = 1:length(target_vars)
76     var_name = target_var_names{idx};
77
78     if strcmp(var_name, 'dh3')
79         i = 7;
80         z = z_list(:, i-1); % z6
81         Jg(1:3, idx) = z;
82         Jg(4:6, idx) = [0; 0; 0];
83         fprintf('\n==== Calculation for variable %s (prismatic joint %d) ====\n',
84             var_name, i);
85         fprintf('z%d:\n', i-1);
86         disp(z);
87         fprintf('Column %d of the Jacobian (prismatic):\n', idx);
88         disp(Jg(:, idx));
89     else
90         i = find(phi == target_vars(idx));
91         if isempty(i)
92             continue;
93         end
94
95         if i == 1
96             z = z_prev;
97             o = o_prev;
98         else
99             z = z_list(:, i-1);
100            o = o_list(:, i-1);
101        end
102
103        fprintf('\n==== Calculation for variable %s (rotational joint %d) ====\n',
104             var_name, i);
105        fprintf('z%d:\n', i-1);
106        disp(z);
107
108        fprintf('O%d:\n', i-1);
109        disp(o);
110
111        fprintf('O_end - O%d:\n', i-1);
112        disp(o_end - o);
113
114        Jg(1:3, idx) = cross(z, (o_end - o));
115        Jg(4:6, idx) = z;
116
117        fprintf('cross(z%d, (O_end - O%d)):\n', i-1, i-1);
118        disp(cross(z, (o_end - o)));
119
120        fprintf('Column %d of the Jacobian (rotational):\n', idx);
121        disp(Jg(:, idx));
122    end
123 end

```

```
122 % Simplify the Jacobian
123 Jg = simplify(Jg);
124 fprintf('\n==== Final Geometric Jacobian ====\n');
125 disp(Jg);
126 % =====
127 % NUMERICAL EVALUATION
128 % =====
129 % Numerical values in radians and meters
130 geo_subs = {
131     Hs, 200/1000;
132     H0, 100/1000;
133     L0, 800/1000;
134     L1, 400/1000;
135     L2, 500/1000;
136     L3, 630/1000;
137     L4, 600/1000;
138     L5, 50/1000;
139     D1, 110/1000;
140     DA2, 100/1000;
141     KH3, 80/1000;
142     d11, 100/1000;    % Hs_val - H0_val = 200 - 100 = 100
143     d12, 900/1000;    % L0_val + H0_val = 800 + 100 = 900
144     d13, 760/1000;    % (DA2_val/2) + L3_val + KH3_val = 50 + 630 + 80 = 760
145     d14, 350/1000    % (L4_val/2) + L5_val = 300 + 50 = 350
146 };
147 % Joint configurations to evaluate (angles in degrees)
148 configurations = [
149     90,0,0,-50/1000,0;
150     90,90,0,-50/1000,0;
151     90, 0.0000, -126.87, -50/1000, -66.87
152 ];
153 % Convert angle columns (1, 2, 3, 5) from degrees to radians
154 configurations_rad = configurations;
155 configurations_rad(:, [1, 2, 3, 5]) = deg2rad(configurations(:, [1, 2, 3, 5]));
156 % Extract configuration values for each variable
157 phi0_vals = configurations_rad(:, 1);
158 phi1_vals = configurations_rad(:, 2);
159 phi2_vals = configurations_rad(:, 3);
160 dh3_vals = configurations(:, 4); % dh3 is in meters, no conversion needed
161 phi4_vals = configurations_rad(:, 5);
162 % Create an array to store numerical Jacobians
163 num_Jg = zeros(6, 5, size(configurations, 1));
164 for k = 1:size(configurations, 1)
165     % Substitute geometric values (geo_subs)
166     Jg_subs = subs(Jg, geo_subs(:, 1), geo_subs(:, 2));
167
168     % Substitute joint values for configuration k
169     Jg_num = subs(Jg_subs, ...
170         [phi0, phi1, phi2, dh3, phi4], ...
171         [phi0_vals(k), phi1_vals(k), phi2_vals(k), dh3_vals(k), phi4_vals(k)]);
172
173     % Convert to numeric
174     num_Jg(:, :, k) = double(Jg_num);
175 end
176 % =====
177 % ENHANCED SINGULARITY ANALYSIS
178 % =====
179 for k = 1:size(configurations, 1)
180     fprintf('\n===== Configuration %d =====\n', k)
181     ;
182     fprintf('Angles: phi0=% .2f°, phi1=% .2f°, phi2=% .2f°, dh3=% .4f m, phi4=% .2f°\n',
183             ...
184             configurations(k,1), configurations(k,2), configurations(k,3), ...
```

```

183     configurations(k,4), configurations(k,5));
184 J = num_Jg(:,:,k);
185 fprintf("The Jacobian is:\n");
186 disp(J);
187 if k==3
188     % My Jacobian has linear velocity first, then angular
189     F_TCP = [30; 0; 100; -50; 25; 75]; % Original (Mx, My, Mz, Fx, Fy, Fz)
190     F_TCP_ordered = [F_TCP(4:6); F_TCP(1:3)]; % Now: Fx, Fy, Fz, Mx, My, Mz
191     fprintf('tau is:\n');
192     tau = J' * F_TCP_ordered;
193     disp(tau);
194     % The first, second, third, and fifth values will be moments [Nm]
195     % The fourth value is the required linear force on the prismatic axis 3 [N]
196     fprintf('\nTau (joint efforts):\n');
197     fprintf('Tau(1): %.2f Nm (Joint 1 - Moment)\n', tau(1));
198     fprintf('Tau(2): %.2f Nm (Joint 2 - Moment)\n', tau(2));
199     fprintf('Tau(3): %.2f Nm (Joint 3 - Moment)\n', tau(3));
200     fprintf('Tau(4): %.2f N (Joint 4 - Linear force prismatic axis)\n', tau
201 (4));
202     fprintf('Tau(5): %.2f Nm (Joint 5 - Moment)\n', tau(5));
203 end
204 %
205 % -----%
206 % 1. ENHANCED SVD ANALYSIS (FOR RECTANGULAR MATRICES)
207 % -----
208 [U,S,V] = svd(J);
209 sv = diag(S);
210 cond_number = cond(J);
211
212 fprintf('\n1. SVD Analysis (Rectangular Matrix):\n');
213 fprintf(' - Singular values: '); fprintf('%.4f ', sv); fprintf('\n');
214 fprintf(' - Condition number: %.2e\n', cond_number);
215
216 % Enhanced singularity criteria
217 sv_threshold = 1e-6; % Threshold for "zero" singular values
217 if any(sv < sv_threshold)
218     fprintf('SINGULARITY DETECTED: ');
219     num_zero_sv = sum(sv < sv_threshold);
220     fprintf('%d singular value(s) below threshold (%.1e)\n', num_zero_sv,
221 sv_threshold);
222
223     % Show problematic rows/columns
224     [min_sv, idx] = min(sv);
225     problematic_dir = U(:,idx);
226     fprintf(' - Most constrained direction: [');
227     fprintf('.2f ', problematic_dir);
228     fprintf(']\n');
229 else
230     fprintf('No singularity detected via SVD\n');
231 end
232 %
233 % -----%
234 % 2. ZERO-ROW HANDLING & SQUARE SUBMATRICES
235 % -----
236
237 fprintf('\n2. Zero-Row Handling & Square Submatrices:\n');
238
239 % Automatically remove zero rows
240 zero_rows = all(abs(J) < 1e-10, 2); % Numerical tolerance
241 J_clean = J(~zero_rows, :);
242
243 if any(zero_rows)
244     fprintf(' - Removed %d zero row(s): ', sum(zero_rows));
245     fprintf('rows %s\n', mat2str(find(zero_rows')));

```

```
244     else
245         fprintf(' - No zero rows found\n');
246     end
247
248 % Clean matrix analysis
249 if size(J_clean, 1) == size(J_clean, 2)
250     % Square matrix after cleaning
251     det_clean = det(J_clean);
252     fprintf(' - Clean matrix is square (size %dx%d)\n', size(J_clean,1), size(
J_clean,2));
253     fprintf(' - Determinant: %.2e\n', det_clean);
254
255     if abs(det_clean) < sv_threshold
256         fprintf('SINGULAR SUBMATRIX: Near-zero determinant\n');
257     else
258         fprintf('Full-rank square submatrix\n');
259     end
260 else
261     fprintf(' - Clean matrix is still rectangular (%dx%d)\n', size(J_clean,1),
size(J_clean,2));
262
263     % Analysis of all possible square submatrices
264     possible_rows = nchoosek(1:size(J,1), size(J,2));
265     subdets = zeros(size(possible_rows,1),1);
266
267     for i = 1:size(possible_rows,1)
268         subJ = J(possible_rows(i,:),:);
269         subdets(i) = det(subJ);
270     end
271
272     fprintf(' - Found %d square submatrices\n', length(subdets));
273     fprintf(' - Min determinant: %.2e, Max: %.2e\n', min(abs(subdets)), max(
abs(subdets)));
274
275     if any(abs(subdets) < sv_threshold)
276         fprintf(' SINGULAR SUBMATRICES: %d/%d have near-zero det\n',...
sum(abs(subdets) < sv_threshold), length(subdets));
277     end
278 end
279
280 % -----
281 % 3. UNIFIED CONCLUSION
282 % -----
283 fprintf('\n3. Final Singularity Conclusion:\n');
284
285 % Combined criteria
286 is_singular = any(sv < sv_threshold) || ... % SVD
287             (exist('subdets','var') && any(abs(subdets) < sv_threshold)) || ...
288 % Submatrices
289             cond_number > 1e3;
290
291 if is_singular
292     fprintf('    ROBOT IS IN SINGULAR CONFIGURATION\n');
293     fprintf('    Reasons:\n');
294
295     if any(sv < sv_threshold)
296         fprintf('        - %d singular value(s) below threshold\n', sum(sv <
sv_threshold));
297     end
298
299     if exist('subdets','var') && any(abs(subdets) < sv_threshold)
300         fprintf('        - %d singular submatrix(es) found\n', sum(abs(subdets) <
sv_threshold));
301     end
```

```

301     end
302
303     if cond_number > 1e3
304         fprintf('      - High condition number (%.2e > 1e3)\n', cond_number);
305     end
306 else
307     fprintf('Configuration is NOT singular\n');
308 end
309
310 % -----
311 % 4. ADDITIONAL VISUALIZATION (OPTIONAL)
312 %
313 fprintf('\n4. Additional Info:\n');
314 fprintf('      - Matrix rank: %d (out of %d)\n', rank(J), min(size(J)));
315 fprintf('      - Clean matrix rank: %d\n', rank(J_clean));
316 end
317 % ----- Substitute all geometric parameters with their numerical values
318 % (This removes D1, L1, L2, dl3, dl4, etc., leaving only phi0, phil, phi2, dh3, phi4
319 )
319 Jg_sym = subs(Jg, geo_subs(:, 1), geo_subs(:, 2));
320 % Simplify the Jacobian (optional, but recommended for symbolic analysis)
321 Jg_sym = simplify(Jg_sym);
322 % Display the simplified Jacobian
323 fprintf('\n==== Geometric Jacobian (joint variables only) ====\n');
324 disp(Jg_sym);
325 % ----- Symbolic Singularity Analysis -----
326 % Option 1: Calculate the determinant of J^T * J (for rectangular matrices)
327 JtJ = Jg_sym.' * Jg_sym;
328 det_JtJ = det(JtJ);
329 det_JtJ = simplify(det_JtJ);
330 fprintf('\n==== Determinant of J^T * J ====\n');
331 disp(det_JtJ);
332 % Option 2: Analysis of square submatrices (if J is 6x5, we take combinations of 5
333 % rows)
333 fprintf('\n==== Analysis of 5x5 Square Submatrices ====\n');
334 rows = 1:6;
335 combinations = nchoosek(rows, 5);
336 for i = 1:size(combinations, 1)
337     J_sub = Jg_sym(combinations(i, :), :);
338     det_sub = det(J_sub.' * J_sub); % J^T * J for submatrix
339     det_sub = simplify(det_sub);
340
341     fprintf('\nSubmatrix with rows %s:\n', mat2str(combinations(i, :)));
342     fprintf('det(J^T * J) = \n');
343     disp(det_sub);
344
345 end
346 % ----- Check ranks -----
347 fprintf('\n==== Symbolic Rank of the Jacobian ====\n');
348 rank_J = rank(Jg_sym);
349 fprintf('Symbolic rank: %d\n', rank_J);
350 if rank_J < min(size(Jg_sym))
351     fprintf(' Jacobian is singular (rank < %d)\n', min(size(Jg_sym)));
352 else
353     fprintf(' Jacobian has full rank (rank = %d)\n', min(size(Jg_sym)));
354 end

```

- Interaktive Visualisierung des Roboters

```
1 clc;
2 clear;
3
4
5 robot = rigidBodyTree('DataFormat','row','MaxNumBodies',8);
6
7 Hs = 200/1000; H0 = 100/1000; L0 = 800/1000;
8 L1 = 400/1000; D1 = 110/1000; L2 = 500/1000;
9 L3 = 630/1000; DH3 = 360/1000; L4 = 600/1000;
10 L5 = 50/1000; H3 = 1060/1000; DA2 = 100/1000;
11 KH3 = 80/1000;
12
13 densidad_acero = 7850; % kg/m^3
14
15 geometrias = {
16     [Hs, 1, 300/1000, 300/1000, 200/1000, 200/1000]; % Nota: cuerpo fijo
17
18     [L0, 1, 200/1000, 200/1000, 150/1000, 150/1000];
19
20     [L1, 2, 100/1000, 100/1000, 90/1000, 90/1000];
21
22     [D1, 2, 100/1000, 100/1000, 90/1000, 90/1000];
23
24     [L2, 1, 80/1000, 80/1000, 70/1000, 70/1000];
25
26     [(DA2/2)+L3+KH3, 1, 70/1000, 70/1000, 60/1000, 60/1000];
27
28     [DH3, 1, 100/1000, 100/1000, 90/1000, 90/1000];
29
30     [(L4/2)+L5, 1, 60/1000, 60/1000, 50/1000, 50/1000];
31 };
32
33 dhparams = [
34     0, 0, Hs-H0, 0; % 1. base to Z1
35     0, pi/2, L0+H0, 0; % 2. phi0 (revolute)
36     L1, -pi/2, 0, 0; % 3. phi1 (revolute)
37     0, 0, D1, 0; % 4. fixed
38     L2, pi, 0, 0; % 5. phi2 (revolute)
39     0, 0, (DA2/2)+L3+KH3, 0; % 6. fixed
40     0, 0, DH3, 0; % 7. prismatic
41     0, 0, (L4/2)+L5, 0 % 8. phi4 (revolute)
42 ];
43 % Tipos de juntas
44 joint_types = {'fixed', 'revolute', 'revolute', 'fixed', 'revolute', 'fixed', 'prismatic', 'revolute'};
45
46 % Correct joint types configuration
47 joint_types = {'fixed', 'revolute', 'revolute', 'fixed', 'revolute', 'fixed', 'prismatic', 'revolute'};
48
49 % Active (movable) joints:
50 activeJointBodies = [2, 3, 5, 7, 8]; % Indices of revolute/prismatic joints
51
52 % =====
53 % Robot construction with realistic dynamic properties
54 % =====
55 for i = 1:8
56     % Create body and joint
57     bodies{i} = rigidBody(['body' num2str(i)]);
58     joints{i} = rigidBodyJoint(['jnt' num2str(i)], joint_types{i});
```

```

60 % Configure DH transformation
61 setFixedTransform(joints{i}, dhparams(i,:), 'dh');
62
63 % Configure axis and limits based on joint type
64 if strcmp(joint_types{i}, 'prismatic')
65     joints{i}.JointAxis = [0 0 1]; % Movement in Z
66     joints{i}.PositionLimits = [-DH3, 0];
67
68 elseif strcmp(joint_types{i}, 'revolute')
69     % Assign limits based on the physical joint
70     switch i
71         case 3 % phi0
72             joints{i}.JointAxis = [0 0 1]; % Z-axis
73             joints{i}.PositionLimits = deg2rad([-150, 150]);
74         case 4 % phi1 - IMPORTANT: Now rotates around Y
75             joints{i}.JointAxis = [0 1 0]; % Y-axis
76             joints{i}.PositionLimits = deg2rad([0, 90]);
77         case 5 % phi2
78             joints{i}.JointAxis = [0 0 1]; % Z-axis
79             joints{i}.PositionLimits = deg2rad([-153, 153]);
80         case 8 % phi4
81             joints{i}.JointAxis = [0 0 1]; % Z-axis
82             joints{i}.PositionLimits = deg2rad([-180, 180]);
83     end
84 end
85
86 % =====
87 % DYNAMIC PROPERTIES BASED ON CUSTOM GEOMETRY
88 % =====
89 if ~strcmp(joint_types{i}, 'fixed')
90     % Get geometry parameters for this body
91     geom = geometrias{i};
92     longitud = geom(1);
93     tipo = geom(2);
94     dim1 = geom(3); % Radio externo o lado externo
95     dim2 = geom(4); % Para cilindro: igual a dim1, para cuadrado: otro lado
96     dim_int1 = geom(5); % Radio interno o lado interno
97     dim_int2 = geom(6); % Para cilindro: igual a dim_int1, para cuadrado: otro
98     lado
99
100    if tipo == 1 % Cilindro hueco
101        radio_externo = dim1;
102        radio_interno = dim_int1;
103        area_seccion = pi*(radio_externo^2 - radio_interno^2);
104    else % Cuadrado hueco
105        lado_externo = dim1;
106        lado_interno = dim_int1;
107        area_seccion = lado_externo^2 - lado_interno^2;
108    end
109
110    volumen = area_seccion * longitud;
111    masa = densidad_acero * volumen;
112
113    centro_masa = [longitud/2, 0, 0];
114
115    if tipo == 1 % Cilindro hueco
116        Ixx = (1/2)*masa*(radio_externo^2 + radio_interno^2);
117        Iyy = (1/12)*masa*(3*(radio_externo^2 + radio_interno^2) + longitud^2);
118        Izz = Iyy;
119    else
120        a_ext = dim1; b_ext = dim2; % Dimensiones externas

```

```

122     a_int = dim_int1; b_int = dim_int2; % Dimensiones internas
123
124     Ixx = (1/12)*masa*(a_ext^2 + b_ext^2 + a_int^2 + b_int^2);
125     Iyy = (1/12)*masa*(3*(b_ext^2 + b_int^2) + longitud^2);
126     Izz = (1/12)*masa*(3*(a_ext^2 + a_int^2) + longitud^2);
127     end
128
129     bodies{i}.Mass = masa;
130     bodies{i}.CenterOfMass = centro_masa;
131     bodies{i}.Inertia = [Ixx, Iyy, Izz, 0, 0, 0];
132 end
133
134 bodies{i}.Joint = joints{i};
135
136
137 if i == 1
138     addBody(robot, bodies{i}, 'base');
139 else
140     addBody(robot, bodies{i}, bodies{i-1}.Name);
141 end
142 end
143
144 robot.Gravity = [0, 0, -9.81];
145
146 endEffectorFrame = 'body8';
147
148 showdetails(robot);
149
150 configurations_deg = [
151     90,0,0,-50/1000,0;
152     0,90,0,-360/1000,0;
153     90, 0.0000, -126.87, -50/1000, -66.87
154 ];
155
156 configurations_rad = [
157     deg2rad(90), deg2rad(0), deg2rad(0), -50/1000, deg2rad(0);
158     deg2rad(0), deg2rad(90), deg2rad(90), 360/1000, deg2rad(0);
159     deg2rad(90), deg2rad(0.0000), deg2rad(-126.87), -50/1000, deg2rad(-66.87)
160 ];
161
162 for i = 1:3
163     q = configurations_rad(i,:);
164
165     fprintf('\n==== Analyzing Configuration %d ====\n', i);
166     fprintf('Joint angles (rad): [% .4f, %.4f, %.4f, %.4f, %.4f]\n', q);
167     fprintf('Joint angles (deg): [% .2f, %.2f, %.2f, %.2f, %.2f]\n', ...
168             rad2deg(q(1)), rad2deg(q(2)), rad2deg(q(3)), q(4)*1000, rad2deg(q(5)));
169
170
171 J = geometricJacobian(robot, q, endEffectorFrame);
172 fprintf('Jacobi is:\n')
173 disp(J);
174 J_reduced = J(:, [1,2,3,4,5]); % Only active joints (phi0, phi1, phi2, phi4)
175
176
177 [U,S,V] = svd(J_reduced);
178 singular_values = diag(S);
179
180 fprintf('\nSingular values:\n');
181 disp(singular_values');
182
183 tol = max(size(J_reduced)) * eps(max(singular_values));

```

```

185 fprintf('\nAdaptive threshold: %e\n', tol);
186
187 rank_defect = sum(singular_values < tol);
188
189 figure('Name', sprintf('Singular Configuration %d', i));
190 show(robot, q, 'Frames', 'on', 'PreservePlot', false);
191 hold on;
192
193 eePos = getTransform(robot, q, endEffectorFrame);
194 plot3(eePos(1,4), eePos(2,4), eePos(3,4), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
195
196 title(sprintf('Singular Configuration %d', i));
197 xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]');
198 view(3);
199 axis equal;
200 grid on;
201 legend('Robot', 'TCP');
202
203 if rank_defect > 0
204     fprintf('WARNING! Robot is in or near a singular configuration.\n');
205     fprintf('Rank deficiency: %d\n', rank_defect);
206
207 lost_motions = U(:,end-rank_defect+1:end);
208 fprintf('\nLost motion directions (null space vectors):\n');
209 disp(lost_motions);
210
211 fprintf('\nPhysical interpretation:\n');
212 for j = 1:rank_defect
213     linear_part = lost_motions(1:3,j)';
214     angular_part = lost_motions(4:6,j)';
215
216     fprintf('Lost direction %d:\n', j);
217     fprintf(' - Linear component: [% .3f, %.3f, %.3f]\n', linear_part);
218     fprintf(' - Angular component: [% .3f, %.3f, %.3f]\n', angular_part);
219
220     % Determine type of loss
221     if norm(linear_part) > 0.9
222         fprintf(' - Dominant loss: linear motion\n');
223     elseif norm(angular_part) > 0.9
224         fprintf(' - Dominant loss: angular motion\n');
225     else
226         fprintf(' - Combined linear/angular loss\n');
227     end
228 end
229
230
231 else
232     fprintf('Robot is NOT in a singular configuration.\n');
233 end
234
235 if i==3
236     F_TCP = [30; 0; 100; -50; 25; 75]; % [Nm; Nm; Nm; N; N; N]
237     tau = J' * F_TCP;
238     fprintf('tau is:\n');
239     disp(tau);
240 end
241
242 end

```

Kapitel 4

Aufgabe 3: Bestimmung der Lagrange-Dynamik

4.1 Theoretische Grundlagen des Lagrange-Verfahrens

Kombination von Schwerpunkt-Jacobi-Matrix und Trägheitsmatrizen:

I. Symbolische Parameter und DH-Kinematik

Die kinematische Struktur wird durch die Denavit-Hartenberg-Parameter ($\alpha_i, L_i, D_i, \varphi_i$) beschrieben. Die homogene Transformationsmatrix jedes Gelenks ist:

$$A_i = \begin{bmatrix} \cos \varphi_i & -\cos \alpha_i \sin \varphi_i & \sin \alpha_i \sin \varphi_i & L_i \cos \varphi_i \\ \sin \varphi_i & \cos \alpha_i \cos \varphi_i & -\sin \alpha_i \cos \varphi_i & L_i \sin \varphi_i \\ 0 & \sin \alpha_i & \cos \alpha_i & D_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

II. Jacobi-Matrix am Schwerpunkt jedes Körpers

Für die dynamische Modellierung wird für jedes Glied k der **Jacobi am Schwerpunkt** $J_{cm,k}$ berechnet:

$$J_{cm,k} = \begin{bmatrix} J_{lin,k} \\ J_{ang,k} \end{bmatrix} \in \mathbb{R}^{6 \times n} \quad (4.2)$$

$$J_{lin,i} = \mathbf{z}_i \times (\mathbf{r}_k - \mathbf{o}_i) \quad (4.3)$$

$$J_{ang,i} = \mathbf{z}_i \quad \text{für Rotationsgelenke} \quad (4.4)$$

$$J_{lin,i} = \mathbf{z}_i, \quad J_{ang,i} = 0 \quad \text{für prismatische Gelenke} \quad (4.5)$$

Dabei ist \mathbf{r}_k der Vektor vom Koordinatenursprung zum Schwerpunkt des k -ten Körpers im Basis-Koordinatensystem.

III. Trägheitstensoren der Glieder

Jeder Trägheitstensor $I_{body,k}$ wird analytisch aus der Geometrie berechnet. Falls CAD-basierte Modelle verwendet werden, werden die Trägheitswerte direkt übernommen. Die Umrechnung in das Basiskoordinatensystem erfolgt über:

$$I_{global,k} = R_k I_{body,k} R_k^\top \quad (4.6)$$

mit $R_k \in SO(3)$, der Rotationsmatrix des Körpers k relativ zur Basis.

IV. Massenmatrix $M(q)$

Die generalisierte Massenmatrix ergibt sich zu:

$$M(q) = \sum_{k=1}^n \left[m_k J_{\text{lin},k}^\top J_{\text{lin},k} + J_{\text{ang},k}^\top I_{\text{global},k} J_{\text{ang},k} \right] \quad (4.7)$$

Dies berücksichtigt sowohl translatorische als auch rotatorische Beiträge aller Körper zum Gesamtsystem.

V. Coriolis- und Zentrifugalterme $C(q, \dot{q})$

Zur Berechnung der Coriolis-Matrix wird das erste Christoffel-Symbol verwendet:

$$c_{ijk} = \frac{1}{2} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right) \quad (4.8)$$

Die Matrix $C(q, \dot{q})$ entsteht durch:

$$C_{ij}(q, \dot{q}) = \sum_{k=1}^n c_{ijk} \dot{q}_k \quad (4.9)$$

VI. Gravitationsvektor $G(q)$

Die potenzielle Energie ergibt sich aus der Höhe der Schwerpunkte im Gravitationsfeld:

$$V = \sum_{k=1}^n m_k \cdot \mathbf{g}^\top \cdot \mathbf{com}_k(q), \quad G(q) = \frac{\partial V}{\partial q} \quad (4.10)$$

Die Schwerpunktpositionen \mathbf{com}_k werden über $T_k \cdot \mathbf{r}_{\text{local}}$ berechnet, wobei $\mathbf{r}_{\text{local}}$ die Lage des Schwerpunkts im lokalen Körperkoordinatensystem beschreibt (analytisch oder aus CAD-Daten).

VII. Vollständige Bewegungsgleichung

Die Gleichung der Bewegung des Roboters im Lagrange-Formalismus lautet:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau \quad (4.11)$$

womit:

$$q = \begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \varphi_2 \\ d_{h3} \\ \varphi_4 \end{bmatrix} \quad \dot{q} = \begin{bmatrix} \dot{\varphi}_0 \\ \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{d}_{h3} \\ \dot{\varphi}_4 \end{bmatrix} \quad \ddot{q} = \begin{bmatrix} \ddot{\varphi}_0 \\ \ddot{\varphi}_1 \\ \ddot{\varphi}_2 \\ \ddot{d}_{h3} \\ \ddot{\varphi}_4 \end{bmatrix}$$

4.1.1 Implementierungshinweise

- **Jacobi-Matrix am Schwerpunkt:** Der Schwerpunkt jedes Körpers wird im Basis-Koordinatensystem berechnet und zur Bestimmung der Jacobi-Matrix verwendet.
- **Trägheitsmatrix-Transformation:** Die lokalen Trägheitstensoren werden mit Hilfe von Rotationsmatrizen R_k in das globale System transformiert.
- **Numerische Vereinfachung:** Symbole mit Beträgen $< 10^{-6}$ werden genullt, um numerische Stabilität zu gewährleisten.
- **Rechengenauigkeit:** Vereinfachungen mit bis zu 6 signifikanten Stellen, dargestellt durch `vpa(. . . , 6)`.

4.2 Berechnung der Lagrange-Gleichung

4.2.1 Berechnung der Jacobi-Matrizen bezogen auf den Massenmittelpunkt jedes Körpers

In dieser Unterabschnitt wird die Berechnung der Jacobi-Matrizen in Bezug auf den Massenmittelpunkt jedes Körpers behandelt. Die Jacobi-Matrix \mathbf{J}_i für den i -ten Körper ist definiert als:

$$\mathbf{J}_i = \frac{\partial \mathbf{v}_i}{\partial \dot{\mathbf{q}}}$$

wobei \mathbf{v}_i die Geschwindigkeit des Massenmittelpunkts des i -ten Körpers und $\dot{\mathbf{q}}$ die verallgemeinerten Geschwindigkeiten sind.

Für ein Mehrkörpersystem mit n Körpern ergibt sich die Gesamt-Jacobi-Matrix \mathbf{J} durch die vertikale Konkatenation der einzelnen Jacobi-Matrizen:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \vdots \\ \mathbf{J}_n \end{bmatrix}$$

Diese Matrix ist entscheidend für die Berechnung der kinetischen Energie und der Dynamik des Systems.

Für die Berechnung der Jacobi-Matrix wurde das gleiche Verfahren wie in Aufgabe 2 angewendet, mit dem Unterschied, dass als Endpunkt der Schwerpunkt des Körpers verwendet wurde. Zusätzlich wurden konstante Variablen ersetzt, um die Herleitung der Lagrange-Gleichung zu erleichtern. Als einzige veränderliche Größen wurden die rotatorischen und prismatischen Gelenkvariationen beibehalten.

Jacobi-Matrix für den Schwerpunkt von Körper 1

$$J_{\text{com},1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.12)$$

Jacobi-Matrix für den Schwerpunkt von Körper 2

$$J_{\text{com},2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.13)$$

Jacobi-Matrix für den Schwerpunkt von Körper 3

$$J_{\text{com},3} = \begin{bmatrix} -0.2 \cos(\varphi_1) \sin(\varphi_0) & \cos(\varphi_0)(dl_1 + dl_2 - 0.2 \sin(\varphi_1) - 1.0) & 0 & 0 & 0 \\ 0.2 \cos(\varphi_0) \cos(\varphi_1) & \sin(\varphi_0)(dl_1 + dl_2 - 0.2 \sin(\varphi_1) - 1.0) & 0 & 0 & 0 \\ 0 & 0.2 \cos(\varphi_1) & 0 & 0 & 0 \\ 0 & \sin(\varphi_0) & 0 & 0 & 0 \\ 0 & -\cos(\varphi_0) & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.14)$$

Jacobi-Matrix für den Schwerpunkt von Körper 4

$$J_{\text{com},4} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (4.15)$$

$$j_{11} = -0.404 \cos(\varphi_1 + 0.137) \sin(\varphi_0)$$

$$j_{12} = 0.005 \cos(\varphi_0)(200.0dl_1 + 200.0dl_2 - 80.753 \cos(\varphi_1 - 1.434) - 200.0)$$

$$j_{13} = 0$$

$$j_{14} = 0$$

$$j_{15} = 0$$

$$j_{21} = 0.404 \cos(\varphi_1 + 0.137) \cos(\varphi_0)$$

$$j_{22} = 0.005 \sin(\varphi_0)(200.0dl_1 + 200.0dl_2 - 80.753 \cos(\varphi_1 - 1.434) - 200.0)$$

$$j_{23} = 0$$

$$j_{24} = 0$$

$$j_{25} = 0$$

$$j_{31} = 0 \qquad \qquad \qquad j_{41} = 0$$

$$j_{32} = 0.404 \cos(\varphi_1 + 0.137) \qquad j_{42} = \sin(\varphi_0)$$

$$j_{33} = 0 \qquad \qquad \qquad j_{43} = 0$$

$$j_{34} = 0 \qquad \qquad \qquad j_{44} = 0$$

$$j_{35} = 0 \qquad \qquad \qquad j_{45} = 0$$

$$j_{51} = 0 \qquad \qquad \qquad j_{61} = 1.0$$

$$j_{52} = -\cos(\varphi_0) \qquad \qquad j_{62} = 0$$

$$j_{53} = 0 \qquad \qquad \qquad j_{63} = 0$$

$$j_{54} = 0 \qquad \qquad \qquad j_{64} = 0$$

$$j_{55} = 0 \qquad \qquad \qquad j_{65} = 0$$

Jacobi-Matrix für den Schwerpunkt von Körper 5

$$J_{\text{com},5} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (4.16)$$

$$\begin{aligned} j_{11} &= 0.11 \sin(\varphi_0) \sin(\varphi_1) - 0.25 \cos(\varphi_0) \sin(\varphi_2) - 0.4 \cos(\varphi_1) \sin(\varphi_0) \\ &\quad - 0.25 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \end{aligned}$$

$$\begin{aligned} j_{12} &= -0.01 \cos(\varphi_0) (11.0 \cos(\varphi_1) - 100.0 d l_2 - 100.0 d l_1 + 40.0 \sin(\varphi_1) \\ &\quad + 25.0 \cos(\varphi_2) \sin(\varphi_1) + 100.0) \end{aligned}$$

$$\begin{aligned} j_{13} &= L_1 \sin(\varphi_0) - 0.25 \cos(\varphi_2) \sin(\varphi_0) - \sin(\varphi_0) \sin(\varphi_1) - 0.4 \sin(\varphi_0) \\ &\quad + d l_1 \sin(\varphi_0) \sin(\varphi_1) + d l_2 \sin(\varphi_0) \sin(\varphi_1) \\ &\quad - 0.25 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \end{aligned}$$

$$j_{14} = 0$$

$$j_{15} = 0$$

$$\begin{aligned} j_{21} &= 0.4 \cos(\varphi_0) \cos(\varphi_1) - 0.11 \cos(\varphi_0) \sin(\varphi_1) - 0.25 \sin(\varphi_0) \sin(\varphi_2) \\ &\quad + 0.25 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$\begin{aligned} j_{22} &= -0.01 \sin(\varphi_0) (11.0 \cos(\varphi_1) - 100.0 d l_2 - 100.0 d l_1 + 40.0 \sin(\varphi_1) \\ &\quad + 25.0 \cos(\varphi_2) \sin(\varphi_1) + 100.0) \end{aligned}$$

$$\begin{aligned} j_{23} &= 0.4 \cos(\varphi_0) + 0.25 \cos(\varphi_0) \cos(\varphi_2) + \cos(\varphi_0) \sin(\varphi_1) - L_1 \cos(\varphi_0) \\ &\quad - d l_1 \cos(\varphi_0) \sin(\varphi_1) - d l_2 \cos(\varphi_0) \sin(\varphi_1) \\ &\quad - 0.25 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) \end{aligned}$$

$$j_{24} = 0$$

$$j_{25} = 0$$

$$j_{31} = 0$$

$$j_{41} = 0$$

$$j_{51} = 0$$

$$j_{61} = 1.0$$

$$j_{32} = 0.4 \cos(\varphi_1) - 0.11 \sin(\varphi_1) + 0.25 \cos(\varphi_1) \cos(\varphi_2) \quad j_{42} = \sin(\varphi_0)$$

$$j_{52} = -\cos(\varphi_0)$$

$$j_{62} = 0$$

$$j_{33} = -\frac{1}{4} \sin(\varphi_1) \sin(\varphi_2)$$

$$j_{43} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$j_{53} = -\sin(\varphi_0) \sin(\varphi_1)$$

$$j_{63} = \cos(\varphi_1)$$

$$j_{34} = 0$$

$$j_{44} = 0$$

$$j_{54} = 0$$

$$j_{64} = 0$$

$$j_{35} = 0$$

$$j_{45} = 0$$

$$j_{55} = 0$$

$$j_{65} = 0$$

Jacobi-Matrix für den Schwerpunkt von Körper 6

$$J_{\text{com},6} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (4.17)$$

$$j_{11} = -0.4 \cos(\varphi_1) \sin(\varphi_0) - 0.5 \cos(\varphi_0) \sin(\varphi_2) - 0.27 \sin(\varphi_0) \sin(\varphi_1) \\ - 0.5 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)$$

$$j_{12} = \frac{\cos(\varphi_0)(100.0dl_1 + 100.0dl_2 + 27.0 \cos(\varphi_1) - 40.0 \sin(\varphi_1))}{100} \\ - \frac{50.0 \cos(\varphi_2) \sin(\varphi_1) - 100.0}{100}$$

$$j_{13} = L_1 \sin(\varphi_0) - 0.5 \cos(\varphi_2) \sin(\varphi_0) - \sin(\varphi_0) \sin(\varphi_1) - 0.4 \sin(\varphi_0) \\ + dl_1 \sin(\varphi_0) \sin(\varphi_1) + dl_2 \sin(\varphi_0) \sin(\varphi_1) \\ - 0.5 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$j_{14} = 0$$

$$j_{15} = 0$$

$$j_{21} = 0.4 \cos(\varphi_0) \cos(\varphi_1) + 0.27 \cos(\varphi_0) \sin(\varphi_1) - 0.5 \sin(\varphi_0) \sin(\varphi_2) \\ + 0.5 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)$$

$$j_{22} = \frac{\sin(\varphi_0)(100.0dl_1 + 100.0dl_2 + 27.0 \cos(\varphi_1) - 40.0 \sin(\varphi_1))}{100} \\ - \frac{50.0 \cos(\varphi_2) \sin(\varphi_1) - 100.0}{100}$$

$$j_{23} = 0.4 \cos(\varphi_0) + 0.5 \cos(\varphi_0) \cos(\varphi_2) + \cos(\varphi_0) \sin(\varphi_1) - L_1 \cos(\varphi_0) \\ - dl_1 \cos(\varphi_0) \sin(\varphi_1) - dl_2 \cos(\varphi_0) \sin(\varphi_1) \\ - 0.5 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)$$

$$j_{24} = 0$$

$$j_{25} = 0$$

$$j_{31} = 0$$

$$j_{41} = 0$$

$$j_{51} = 0$$

$$j_{61} = 1.0$$

$$j_{32} = 0.4 \cos(\varphi_1) + 0.27 \sin(\varphi_1) + 0.5 \cos(\varphi_1) \cos(\varphi_2) \quad j_{42} = \sin(\varphi_0)$$

$$j_{52} = -\cos(\varphi_0) \quad j_{62} = 0$$

$$j_{33} = -\frac{1}{2} \sin(\varphi_1) \sin(\varphi_2)$$

$$j_{43} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$j_{53} = -\sin(\varphi_0) \sin(\varphi_1)$$

$$j_{63} = \cos(\varphi_1)$$

$$j_{34} = 0$$

$$j_{44} = 0$$

$$j_{54} = 0$$

$$j_{64} = 0$$

$$j_{35} = 0$$

$$j_{45} = 0$$

$$j_{55} = 0$$

$$j_{65} = 0$$

Jacobi-Matrix für den Schwerpunkt von Körper 7

$$J_{\text{com},7} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (4.18)$$

$$\begin{aligned} j_{11} = & -0.4 \cos(\varphi_1) \sin(\varphi_0) - 0.5 \cos(\varphi_0) \sin(\varphi_2) - 0.65 \sin(\varphi_0) \sin(\varphi_1) \\ & - 0.5 d h_3 \sin(\varphi_0) \sin(\varphi_1) - 0.5 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0) \end{aligned}$$

$$j_{12} = \frac{\cos(\varphi_0)(20d l_1 + 20d l_2 + 13 \cos(\varphi_1) - 8 \sin(\varphi_1))}{20} - \frac{10 \cos(\varphi_2) \sin(\varphi_1) + 10d h_3 \cos(\varphi_1) - 20}{20}$$

$$\begin{aligned} j_{13} = & L_1 \sin(\varphi_0) - 0.5 \cos(\varphi_2) \sin(\varphi_0) - \sin(\varphi_0) \sin(\varphi_1) - 0.4 \sin(\varphi_0) \\ & + d l_1 \sin(\varphi_0) \sin(\varphi_1) + d l_2 \sin(\varphi_0) \sin(\varphi_1) \\ & - 0.5 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) \end{aligned}$$

$$j_{14} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{15} = 0$$

$$\begin{aligned} j_{21} = & 0.4 \cos(\varphi_0) \cos(\varphi_1) + 0.65 \cos(\varphi_0) \sin(\varphi_1) - 0.5 \sin(\varphi_0) \sin(\varphi_2) \\ & + 0.5 d h_3 \cos(\varphi_0) \sin(\varphi_1) + 0.5 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) \end{aligned}$$

$$j_{22} = \frac{\sin(\varphi_0)(20d l_1 + 20d l_2 + 13 \cos(\varphi_1) - 8 \sin(\varphi_1))}{20} - \frac{10 \cos(\varphi_2) \sin(\varphi_1) + 10d h_3 \cos(\varphi_1) - 20}{20}$$

$$\begin{aligned} j_{23} = & 0.4 \cos(\varphi_0) + 0.5 \cos(\varphi_0) \cos(\varphi_2) + \cos(\varphi_0) \sin(\varphi_1) - L_1 \cos(\varphi_0) \\ & - d l_1 \cos(\varphi_0) \sin(\varphi_1) - d l_2 \cos(\varphi_0) \sin(\varphi_1) \\ & - 0.5 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) \end{aligned}$$

$$j_{24} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{25} = 0$$

$$j_{31} = 0$$

$$j_{41} = 0$$

$$j_{51} = 0$$

$$j_{61} = 1.0$$

$$\begin{aligned} j_{32} = & 0.4 \cos(\varphi_1) + 0.65 \sin(\varphi_1) + 0.5 \cos(\varphi_1) \cos(\varphi_2) & j_{42} = \sin(\varphi_0) \\ & + 0.5 d h_3 \sin(\varphi_1) & j_{52} = -\cos(\varphi_0) \end{aligned}$$

$$j_{62} = 0$$

$$j_{33} = -\frac{1}{2} \sin(\varphi_1) \sin(\varphi_2)$$

$$j_{43} = -\cos(\varphi_0) \sin(\varphi_1)$$

$$j_{53} = -\sin(\varphi_0) \sin(\varphi_1)$$

$$j_{63} = \cos(\varphi_1)$$

$$j_{34} = -\cos(\varphi_1)$$

$$j_{44} = 0$$

$$j_{54} = 0$$

$$j_{64} = 0$$

$$j_{35} = 0$$

$$j_{45} = 0$$

$$j_{55} = 0$$

$$j_{65} = 0$$

Jacobi-Matrix für den Schwerpunkt von Körper 8

$$J_{\text{com},8} = \begin{bmatrix} j_{11} & j_{12} & j_{13} & j_{14} & j_{15} \\ j_{21} & j_{22} & j_{23} & j_{24} & j_{25} \\ j_{31} & j_{32} & j_{33} & j_{34} & j_{35} \\ j_{41} & j_{42} & j_{43} & j_{44} & j_{45} \\ j_{51} & j_{52} & j_{53} & j_{54} & j_{55} \\ j_{61} & j_{62} & j_{63} & j_{64} & j_{65} \end{bmatrix} \quad (4.19)$$

$$j_{11} = -0.4 \cos(\varphi_1) \sin(\varphi_0) - 0.5 \cos(\varphi_0) \sin(\varphi_2) - 0.825 \sin(\varphi_0) \sin(\varphi_1) \\ - dh_3 \sin(\varphi_0) \sin(\varphi_1) - 0.5 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)$$

$$j_{12} = \frac{\cos(\varphi_0)(40dl_1 + 40dl_2 + 33 \cos(\varphi_1) - 16 \sin(\varphi_1))}{40} \\ - \frac{20 \cos(\varphi_2) \sin(\varphi_1) + 40dh_3 \cos(\varphi_1) - 40}{40}$$

$$j_{13} = L_1 \sin(\varphi_0) - 0.5 \cos(\varphi_2) \sin(\varphi_0) - \sin(\varphi_0) \sin(\varphi_1) - 0.4 \sin(\varphi_0) \\ + dl_1 \sin(\varphi_0) \sin(\varphi_1) + dl_2 \sin(\varphi_0) \sin(\varphi_1) \\ - 0.5 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$j_{14} = \cos(\varphi_0) \sin(\varphi_1)$$

$$j_{15} = 0.4 \sin(\varphi_0) + 0.5 \cos(\varphi_2) \sin(\varphi_0) + \sin(\varphi_0) \sin(\varphi_1) - L_1 \sin(\varphi_0) \\ - L_2 \cos(\varphi_2) \sin(\varphi_0) - dl_1 \sin(\varphi_0) \sin(\varphi_1) - dl_2 \sin(\varphi_0) \sin(\varphi_1) \\ + 0.5 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2) - L_2 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$j_{21} = 0.4 \cos(\varphi_0) \cos(\varphi_1) + 0.825 \cos(\varphi_0) \sin(\varphi_1) - 0.5 \sin(\varphi_0) \sin(\varphi_2) \\ + dh_3 \cos(\varphi_0) \sin(\varphi_1) + 0.5 \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)$$

$$j_{22} = \frac{\sin(\varphi_0)(40dl_1 + 40dl_2 + 33 \cos(\varphi_1) - 16 \sin(\varphi_1))}{40} \\ - \frac{20 \cos(\varphi_2) \sin(\varphi_1) + 40dh_3 \cos(\varphi_1) - 40}{40}$$

$$j_{23} = 0.4 \cos(\varphi_0) + 0.5 \cos(\varphi_0) \cos(\varphi_2) + \cos(\varphi_0) \sin(\varphi_1) - L_1 \cos(\varphi_0) \\ - dl_1 \cos(\varphi_0) \sin(\varphi_1) - dl_2 \cos(\varphi_0) \sin(\varphi_1) \\ - 0.5 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)$$

$$j_{24} = \sin(\varphi_0) \sin(\varphi_1)$$

$$j_{25} = L_1 \cos(\varphi_0) - 0.5 \cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_0) \sin(\varphi_1) - 0.4 \cos(\varphi_0) \\ + L_2 \cos(\varphi_0) \cos(\varphi_2) + dl_1 \cos(\varphi_0) \sin(\varphi_1) + dl_2 \cos(\varphi_0) \sin(\varphi_1) \\ + 0.5 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - L_2 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)$$

$$\begin{aligned}
 j_{31} &= 0 & j_{41} &= 0 \\
 j_{51} &= 0 & j_{61} &= 1.0 \\
 j_{32} &= 0.4 \cos(\varphi_1) + 0.825 \sin(\varphi_1) + 0.5 \cos(\varphi_1) \cos(\varphi_2) & j_{42} &= \sin(\varphi_0) \\
 &+ dh_3 \sin(\varphi_1) & j_{52} &= -\cos(\varphi_0) \\
 j_{62} &= 0 & & \\
 j_{33} &= -\frac{1}{2} \sin(\varphi_1) \sin(\varphi_2) & j_{43} &= -\cos(\varphi_0) \sin(\varphi_1) \\
 j_{53} &= -\sin(\varphi_0) \sin(\varphi_1) & j_{63} &= \cos(\varphi_1) \\
 j_{34} &= -\cos(\varphi_1) & j_{44} &= 0 \\
 j_{54} &= 0 & j_{64} &= 0 \\
 j_{35} &= -\sin(\varphi_1) \sin(\varphi_2) (L_2 - 0.5) & j_{45} &= \cos(\varphi_0) \sin(\varphi_1) \\
 j_{55} &= \sin(\varphi_0) \sin(\varphi_1) & j_{65} &= 0
 \end{aligned}$$

4.2.2 Trägheitsmatrizen und deren Transformation durch Rotation

Glied 1 Geometrie: Hohlzylinder

Länge $L = 0.200 \text{ m}$

Dichte $\rho = 7750 \text{ kg m}^{-3}$

$r_{\text{ext}} = 0.150 \text{ m}$

Innenradius $r_{\text{int}} = 0.100 \text{ m}$

Formeln:

$$\begin{aligned}
 I_{xx} = I_{yy} &= \frac{1}{12} \rho \pi (r_e^2 - r_i^2) L [3(r_e^2 + r_i^2) + L^2] = 0.697450 \text{ kg m}^2 \\
 I_{zz} &= \frac{1}{2} \rho \pi (r_e^2 - r_i^2) L (r_e^2 + r_i^2) = 0.989111 \text{ kg m}^2
 \end{aligned}$$

Masse: 60.868 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.6974 & 0 & 0 \\ 0 & 0.6974 & 0 \\ 0 & 0 & 0.9891 \end{bmatrix}$$

Glied 2 Geometrie: Hohlzylinder + Kugel (aus CAD)

Länge $L = 0.800 \text{ m}$

CAD-Werte verwendet:

Masse: 100.498 kg

Trägheitsmatrix:

$$\begin{bmatrix} 7.749 & 0 & 0 \\ 0 & 0.7763 & 0 \\ 0 & 0 & 7.749 \end{bmatrix}$$

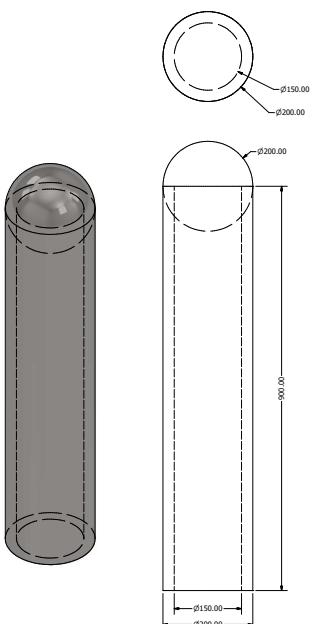


Abbildung 4.1: Glied 2 CAD

General		Summary		Project		Status	Custom	Save	Physical
Material									
								<input type="button" value="Update"/>	
Density		Requested Accuracy						<input type="button" value="Clipboard"/>	
7.730 g/cm ³		Low							
General Properties									
<input type="checkbox"/> Include Cosmetic Welds		<input type="checkbox"/> Include QTY Overrides							
				Center of Gravity					
Mass	100.238 kg (Relative)	<input type="button" value="..."/>	X	0.000 mm (Relative)					
Area	1256165.823 mm ²		Y	470.732 mm (Relative)					
Volume	12967447.276 mm ³	<input type="button" value="..."/>	Z	0.000 mm (Relative)					
Inertial Properties									
<input checked="" type="button" value="Principal"/>		<input type="button" value="Global"/>		<input type="button" value="Center of Gravity"/>					
Principal Moments									
I ₁₁	7749265.619 kg	I ₂₂	776332.607 kg	I ₃₃	7749265.619 kg				
Rotation to Principal									
R _x	0.00 deg (Relative)	R _y	0.00 deg (Relative)	R _z	0.00 deg (Relative)				

Abbildung 4.2: Glied 2 CAD Eigenschaften

Glied 3 Geometrie: Hohler quadratisches Prisma

Länge $L = 0.400 \text{ m}$

$a_{\text{ext}} = 0.100 \text{ m}$

Innenkante $a_{\text{int}} = 0.080 \text{ m}$

Formeln:

$$I_{xx} = I_{yy} = \frac{1}{12} \rho (a_e^2 - a_i^2) L \left[L^2 + \frac{1}{3} (a_e^2 + a_i^2) \right] = 0.154\,000 \text{ kg m}^2$$

$$I_{zz} = \frac{1}{6} \rho (a_e^2 - a_i^2) L (a_e^2 + a_i^2) = 0.030\,504 \text{ kg m}^2$$

Masse: 11.160 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.154 & 0 & 0 \\ 0 & 0.154 & 0 \\ 0 & 0 & 0.0305 \end{bmatrix}$$

Glied 4 Geometrie: Hohler Quader + oberer Zylinder (aus CAD)

Länge $L = 0.110 \text{ m}$

CAD-Werte verwendet:

Masse: 5.741 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.0361 & 0 & 0 \\ 0 & 0.018 & 0 \\ 0 & 0 & 0.0361 \end{bmatrix}$$

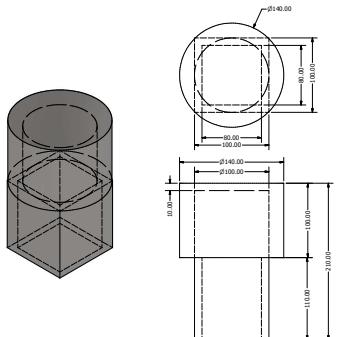


Abbildung 4.3: Glied 4 CAD

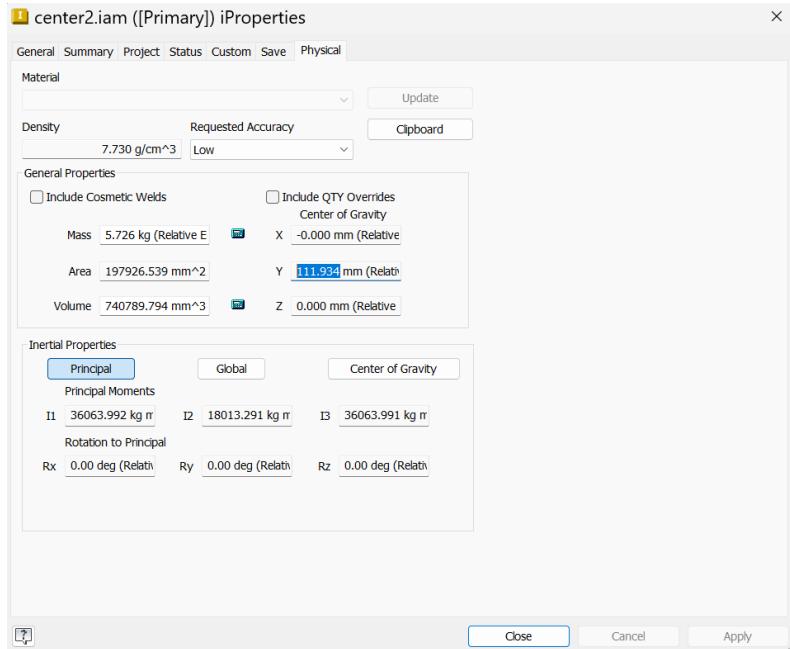


Abbildung 4.4: Glied 4 CAD Eigenschaften

Glied 5 Geometrie: Hohlzylinder

Länge $L = 0.500 \text{ m}$

$r_{\text{ext}} = 0.050 \text{ m}$, $r_{\text{int}} = 0.040 \text{ m}$

Berechnete Trägheitsmomente:

$$I_{xx} = I_{yy} = 0.239\,487 \text{ kg m}^2, \\ I_{zz} = 0.022\,460 \text{ kg m}^2$$

Masse: 10.956 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.2395 & 0 & 0 \\ 0 & 0.2395 & 0 \\ 0 & 0 & 0.0225 \end{bmatrix}$$

Glied 6 Geometrie: Hohlzylinder

Länge $L = 0.760 \text{ m}$

$r_{\text{ext}} = 0.050 \text{ m}$, $r_{\text{int}} = 0.040 \text{ m}$

Berechnete Trägheitsmomente:

$$I_{xx} = I_{yy} = 0.818\,662 \text{ kg m}^2, \\ I_{zz} = 0.034\,140 \text{ kg m}^2$$

Masse: 16.654 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.8187 & 0 & 0 \\ 0 & 0.8187 & 0 \\ 0 & 0 & 0.0341 \end{bmatrix}$$

Glied 7 Geometrie: Hohlzylinder

Länge $L = 0.360 \text{ m}$

$r_{\text{ext}} = 0.070 \text{ m}$, $r_{\text{int}} = 0.040 \text{ m}$

Berechnete Trägheitsmomente:

$$I_{xx} = I_{yy} = 0.359\,389 \text{ kg m}^2,$$

$$I_{zz} = 0.094\,005 \text{ kg m}^2$$

Masse: 28.925 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.3594 & 0 & 0 \\ 0 & 0.3594 & 0 \\ 0 & 0 & 0.094 \end{bmatrix}$$

Glied 8 Geometrie: Hohlzylinder

Länge $L = 0.350 \text{ m}$

$r_{\text{ext}} = 0.070 \text{ m}$, $r_{\text{int}} = 0.040 \text{ m}$

Berechnete Trägheitsmomente:

$$I_{xx} = I_{yy} = 0.332\,767 \text{ kg m}^2,$$

$$I_{zz} = 0.091\,394 \text{ kg m}^2$$

Masse: 28.121 kg

Trägheitsmatrix:

$$\begin{bmatrix} 0.3328 & 0 & 0 \\ 0 & 0.3328 & 0 \\ 0 & 0 & 0.0914 \end{bmatrix}$$

Transformation der Trägheitsmatrizen der einzelnen Glieder

Glied 1

$$R_1 = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0 \\ 0 & 0 & 1.0 \end{bmatrix} \quad (4.20)$$

$$I_{\text{lokal},1} = \begin{bmatrix} 0.697 & 0 & 0 \\ 0 & 0.697 & 0 \\ 0 & 0 & 0.989 \end{bmatrix} \quad (4.21)$$

$$I_{\text{global},1} = R_1 I_{\text{lokal},1} R_1^T = \begin{bmatrix} 0.697 & 0 & 0 \\ 0 & 0.697 & 0 \\ 0 & 0 & 0.989 \end{bmatrix} \quad (4.22)$$

Glied 2

$$R_2 = \begin{bmatrix} \cos(\varphi_0) & 0 & \sin(\varphi_0) \\ \sin(\varphi_0) & 0 & -\cos(\varphi_0) \\ 0 & 1.0 & 0 \end{bmatrix} \quad (4.23)$$

$$I_{\text{lokal},2} = \begin{bmatrix} 7.749 & 0 & 0 \\ 0 & 0.776 & 0 \\ 0 & 0 & 7.749 \end{bmatrix} \quad (4.24)$$

$$I_{\text{global},2} = R_2 I_{\text{lokal},2} R_2^T = \begin{bmatrix} 7.749 & 0 & 0 \\ 0 & 7.749 & 0 \\ 0 & 0 & 0.776 \end{bmatrix} \quad (4.25)$$

Glied 3

$$R_3 = \begin{bmatrix} \cos(\varphi_0) \cos(\varphi_1) & -\sin(\varphi_0) & -\cos(\varphi_0) \sin(\varphi_1) \\ \cos(\varphi_1) \sin(\varphi_0) & \cos(\varphi_0) & -\sin(\varphi_0) \sin(\varphi_1) \\ \sin(\varphi_1) & 0 & \cos(\varphi_1) \end{bmatrix} \quad (4.26)$$

$$I_{\text{lokal},3} = \begin{bmatrix} 0.154 & 0 & 0 \\ 0 & 0.154 & 0 \\ 0 & 0 & 0.031 \end{bmatrix} \quad (4.27)$$

$$I_{\text{global},3} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} \quad (4.28)$$

$$i_{11} = 0.123 \sin^2(\varphi_0) \sin^2(\varphi_1) - 0.123 \sin^2(\varphi_1) + 0.154$$

$$i_{12} = 0.031 \sin(2\varphi_0)(\cos(2\varphi_1) - 1.0)$$

$$i_{13} = 0.123 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_1)$$

$$i_{21} = i_{12}$$

$$i_{22} = 0.154 - 0.123 \sin^2(\varphi_0) \sin^2(\varphi_1)$$

$$i_{23} = 0.123 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_1)$$

$$i_{31} = i_{13}$$

$$i_{32} = i_{23}$$

$$i_{33} = 0.123 \sin^2(\varphi_1) + 0.031$$

Glied 4

$$R_4 = \begin{bmatrix} \cos(\varphi_0) \cos(\varphi_1) & -\sin(\varphi_0) & -\cos(\varphi_0) \sin(\varphi_1) \\ \cos(\varphi_1) \sin(\varphi_0) & \cos(\varphi_0) & -\sin(\varphi_0) \sin(\varphi_1) \\ \sin(\varphi_1) & 0 & \cos(\varphi_1) \end{bmatrix} \quad (4.29)$$

$$I_{\text{lokal},4} = \begin{bmatrix} 0.036 & 0 & 0 \\ 0 & 0.018 & 0 \\ 0 & 0 & 0.036 \end{bmatrix} \quad (4.30)$$

$$I_{\text{global},4} = \begin{bmatrix} 0.036 - 0.018 \sin^2(\varphi_0) & 0.009 \sin(2\varphi_0) & 0 \\ 0.009 \sin(2\varphi_0) & 0.018 \sin^2(\varphi_0) + 0.018 & 0 \\ 0 & 0 & 0.036 \end{bmatrix} \quad (4.31)$$

Glied 5

$$R_5 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (4.32)$$

$$r_{11} = \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2)$$

$$r_{12} = \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$r_{13} = \cos(\varphi_0) \sin(\varphi_1)$$

$$r_{21} = \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)$$

$$r_{22} = \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2)$$

$$r_{23} = \sin(\varphi_0) \sin(\varphi_1)$$

$$r_{31} = \cos(\varphi_2) \sin(\varphi_1)$$

$$r_{32} = \sin(\varphi_1) \sin(\varphi_2)$$

$$r_{33} = -\cos(\varphi_1)$$

$$I_{\text{lokal},5} = \begin{bmatrix} 0.240 & 0 & 0 \\ 0 & 0.240 & 0 \\ 0 & 0 & 0.023 \end{bmatrix} \quad (4.33)$$

$$I_{\text{global},5} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} \quad (4.34)$$

$$i_{11} = 0.054 \cos(2\varphi_1) - 0.054 \cos(2\varphi_0) + 0.054 \cos(2\varphi_0) \cos(2\varphi_1) + 0.185$$

$$i_{12} = 0.054 \sin(2\varphi_0) (\cos(2\varphi_1) - 1.0)$$

$$i_{13} = 0.217 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_1)$$

$$i_{21} = i_{12}$$

$$i_{22} = 0.054 \cos(2\varphi_0) + 0.054 \cos(2\varphi_1) - 0.054 \cos(2\varphi_0) \cos(2\varphi_1) + 0.185$$

$$i_{23} = 0.217 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_1)$$

$$i_{31} = i_{13}$$

$$i_{32} = i_{23}$$

$$i_{33} = 0.217 \sin^2(\varphi_1) + 0.023$$

Glied 6

$$R_6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (4.35)$$

$$r_{11} = \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2)$$

$$r_{12} = \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$r_{13} = \cos(\varphi_0) \sin(\varphi_1)$$

$$r_{21} = \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)$$

$$r_{22} = \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2)$$

$$r_{23} = \sin(\varphi_0) \sin(\varphi_1)$$

$$r_{31} = \cos(\varphi_2) \sin(\varphi_1)$$

$$r_{32} = \sin(\varphi_1) \sin(\varphi_2)$$

$$r_{33} = -\cos(\varphi_1)$$

$$I_{\text{lokal},6} = \begin{bmatrix} 0.819 & 0 & 0 \\ 0 & 0.819 & 0 \\ 0 & 0 & 0.034 \end{bmatrix} \quad (4.36)$$

$$I_{\text{global},6} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} \quad (4.37)$$

$$i_{11} = 0.196 \cos(2\varphi_1) - 0.196 \cos(2\varphi_0) + 0.196 \cos(2\varphi_0) \cos(2\varphi_1) + 0.623$$

$$i_{12} = 0.196 \sin(2\varphi_0) (\cos(2\varphi_1) - 1.0)$$

$$i_{13} = 0.785 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_1)$$

$$i_{21} = i_{12}$$

$$i_{22} = 0.196 \cos(2\varphi_0) + 0.196 \cos(2\varphi_1) - 0.196 \cos(2\varphi_0) \cos(2\varphi_1) + 0.623$$

$$i_{23} = 0.785 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_1)$$

$$i_{31} = i_{13}$$

$$i_{32} = i_{23}$$

$$i_{33} = 0.785 \sin^2(\varphi_1) + 0.034$$

Glied 7

$$R_7 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (4.38)$$

$$r_{11} = \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2) - \sin(\varphi_0) \sin(\varphi_2)$$

$$r_{12} = \cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)$$

$$r_{13} = \cos(\varphi_0) \sin(\varphi_1)$$

$$r_{21} = \cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)$$

$$r_{22} = \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_2)$$

$$r_{23} = \sin(\varphi_0) \sin(\varphi_1)$$

$$r_{31} = \cos(\varphi_2) \sin(\varphi_1)$$

$$r_{32} = \sin(\varphi_1) \sin(\varphi_2)$$

$$r_{33} = -\cos(\varphi_1)$$

$$I_{\text{lokal},7} = \begin{bmatrix} 0.359 & 0 & 0 \\ 0 & 0.359 & 0 \\ 0 & 0 & 0.094 \end{bmatrix} \quad (4.39)$$

$$I_{\text{global},7} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} \quad (4.40)$$

$$i_{11} = 0.066 \cos(2\varphi_1) - 0.066 \cos(2\varphi_0) + 0.066 \cos(2\varphi_0) \cos(2\varphi_1) + 0.293$$

$$i_{12} = 0.066 \sin(2\varphi_0) (\cos(2\varphi_1) - 1.0)$$

$$i_{13} = 0.265 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_1)$$

$$i_{21} = i_{12}$$

$$i_{22} = 0.066 \cos(2\varphi_0) + 0.066 \cos(2\varphi_1) - 0.066 \cos(2\varphi_0) \cos(2\varphi_1) + 0.293$$

$$i_{23} = 0.265 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_1)$$

$$i_{31} = i_{13}$$

$$i_{32} = i_{23}$$

$$i_{33} = 0.265 \sin^2(\varphi_1) + 0.094$$

Glied 8

$$R_8 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (4.41)$$

$$\begin{aligned} r_{11} &= \sin(\varphi_4) [\cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)] \\ &\quad - \cos(\varphi_4) [\sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)] \end{aligned}$$

$$\begin{aligned} r_{21} &= \cos(\varphi_4) [\cos(\varphi_2) \sin(\varphi_0) + \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_2)] \\ &\quad + \sin(\varphi_4) [\sin(\varphi_0) \sin(\varphi_2) - \cos(\varphi_0) \cos(\varphi_1) \cos(\varphi_2)] \end{aligned}$$

$$r_{31} = \cos(\varphi_0) \sin(\varphi_1)$$

$$\begin{aligned} r_{12} &= \cos(\varphi_4) [\cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)] \\ &\quad - \sin(\varphi_4) [\cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)] \end{aligned}$$

$$\begin{aligned} r_{22} &= -\cos(\varphi_4) [\cos(\varphi_0) \cos(\varphi_2) - \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_2)] \\ &\quad - \sin(\varphi_4) [\cos(\varphi_0) \sin(\varphi_2) + \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_0)] \end{aligned}$$

$$r_{32} = \sin(\varphi_0) \sin(\varphi_1)$$

$$r_{13} = \sin(\varphi_1) \sin(\varphi_2) \sin(\varphi_4) + \cos(\varphi_2) \cos(\varphi_4) \sin(\varphi_1)$$

$$r_{23} = \cos(\varphi_4) \sin(\varphi_1) \sin(\varphi_2) - \cos(\varphi_2) \sin(\varphi_1) \sin(\varphi_4)$$

$$r_{33} = -\cos(\varphi_1)$$

$$I_{\text{lokal},8} = \begin{bmatrix} 0.333 & 0 & 0 \\ 0 & 0.333 & 0 \\ 0 & 0 & 0.091 \end{bmatrix} \quad (4.42)$$

$$I_{\text{global},8} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} \quad (4.43)$$

$$i_{11} = 0.060 \cos(2\varphi_1) - 0.060 \cos(2\varphi_0) + 0.060 \cos(2\varphi_0) \cos(2\varphi_1) + 0.273$$

$$i_{12} = 0.060 \sin(2\varphi_0) (\cos(2\varphi_1) - 1.0)$$

$$i_{13} = 0.241 \cos(\varphi_0) \cos(\varphi_1) \sin(\varphi_1)$$

$$i_{21} = i_{12}$$

$$i_{22} = 0.060 \cos(2\varphi_0) + 0.060 \cos(2\varphi_1) - 0.060 \cos(2\varphi_0) \cos(2\varphi_1) + 0.273$$

$$i_{23} = 0.241 \cos(\varphi_1) \sin(\varphi_0) \sin(\varphi_1)$$

$$i_{31} = i_{13}$$

$$i_{32} = i_{23}$$

$$i_{33} = 0.241 \sin^2(\varphi_1) + 0.091$$

Glied	Masse (kg)
1	60.868
2	100.498
3	11.160
4	5.741
5	10.956
6	16.654
7	28.925
8	28.121
Gesamtmasse	262.923

Tabelle 4.1: Segmentweise Masseverteilung des Robotersystems

4.2.3 Gesamte Berechnung der Massenmatrix $M(\mathbf{q})$

Die Trägheitsmatrix $M(q) \in \mathbb{R}^{n \times n}$ eines Roboters mit n Freiheitsgraden wird basierend auf der kinetischen Energie des Systems berechnet. Die vollständige Formel lautet:

$$M(q) = \sum_{k=1}^n \left(m_k J_{v_k}^\top J_{v_k} + J_{\omega_k}^\top R_k I_k R_k^\top J_{\omega_k} \right) \quad (4.44)$$

Dabei bedeuten:

- m_k : Masse des k -ten Gliedes
- J_{v_k} : linearer Jacobi der Schwerpunktposition des k -ten Gliedes
- J_{ω_k} : rotatorischer Jacobi des Schwerpunkts des k -ten Gliedes
- R_k : Rotationsmatrix vom lokalen Koordinatensystem des Gliedes k zum Basis-Koordinatensystem
- I_k : Trägheitstensor des Gliedes k im lokalen Koordinatensystem

Diese Formel berücksichtigt sowohl die translatorische als auch die rotatorische kinetische Energie jedes Gliedes in Bezug auf den Schwerpunkt und ermöglicht so eine präzise Bestimmung der Massenmatrix $M(q)$ in jeder Gelenkkonfiguration des Roboters.

Berechnete Trägheitsmatrix $M(q)$ Die Trägheitsmatrix $M(q)$ ist definiert als:

$$M(q) = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} & M_{15} \\ M_{21} & M_{22} & M_{23} & M_{24} & M_{25} \\ M_{31} & M_{32} & M_{33} & M_{34} & M_{35} \\ M_{41} & M_{42} & M_{43} & M_{44} & M_{45} \\ M_{51} & M_{52} & M_{53} & M_{54} & M_{55} \end{bmatrix} \quad (4.45)$$

mit den Elementen:

$$\begin{aligned}
 M_{11} &= 65.201 dh_3 + 17.991 \sin(2\varphi_1) + 17.033 dh_3 \sin(2\varphi_1) \\
 &\quad - 19.447 \cos^2(\varphi_1) - 19.110 \cos^2(\varphi_2) \\
 &\quad - 35.352 dh_3^2 \cos^2(\varphi_1) + 31.671 \cos^2(\varphi_1) \cos(\varphi_2) \\
 &\quad + 35.352 dh_3^2 + 19.110 \cos^2(\varphi_1) \cos^2(\varphi_2) \\
 &\quad - 65.201 dh_3 \cos^2(\varphi_1) + 45.895 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_1) \\
 &\quad + 42.584 dh_3 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_1) + 54.551, \\
 M_{13} &= 19.352 \cos(\varphi_1) + 15.836 \cos(\varphi_1) \cos(\varphi_2) \\
 &\quad + 22.947 \cos(\varphi_2) \sin(\varphi_1) + 21.292 dh_3 \cos(\varphi_2) \sin(\varphi_1), \\
 M_{14} &= -28.523 \sin(\varphi_1) \sin(\varphi_2), \\
 M_{15} &= -0.091 \cos(\varphi_1), \\
 M_{22} &= 65.201 dh_3 + 31.671 \cos(\varphi_2) + 19.110 \cos^2(\varphi_2) \\
 &\quad + 35.352 dh_3^2 + 49.557, \\
 M_{23} &= -22.947 \sin(\varphi_2) - 21.292 dh_3 \sin(\varphi_2), \\
 M_{24} &= -28.523 \cos(\varphi_2) - 22.818, \\
 M_{33} &= 19.352, \\
 M_{35} &= -0.091, \\
 M_{44} &= 57.046, \\
 M_{55} &= 0.091.
 \end{aligned}$$

4.2.4 Coriolis- und Zentrifugal-Matrix

Die Matrix $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ wird auf Basis der Christoffel-Symbole zweiter Art definiert. Die Komponenten c_{ij} der Matrix \mathbf{C} werden wie folgt berechnet:

$$C_{ij}(q, \dot{q}) = \sum_{k=1}^n \Gamma_{ijk}(q) \cdot \dot{q}_k \quad (4.46)$$

wobei die Christoffel-Symbole durch folgende Formel gegeben sind:

$$\Gamma_{ijk}(q) = \frac{1}{2} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right) \quad (4.47)$$

Zusammenfassend ergibt sich die Matrix:

$$C_{ij}(q, \dot{q}) = \sum_{k=1}^n \frac{1}{2} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right) \cdot \dot{q}_k \quad (4.48)$$

- q_i, q_j, q_k sind die generalisierten Koordinaten.
- \dot{q}_k ist die zeitliche Ableitung der k -ten Koordinate (also die Gelenkgeschwindigkeit).
- M_{ij} ist das ij -te Element der Trägheitsmatrix.

Bemerkung

Die endgültige Matrix $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ enthält Terme sowohl der Coriolis- als auch der Zentrifugalkräfte, welche später in der Lagrange-Gleichung genutzt werden:

$$\mathbf{M}(\mathbf{q}) \cdot \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \cdot \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \quad (4.49)$$

wobei $\mathbf{G}(\mathbf{q})$ die Gravitationskraft und $\boldsymbol{\tau}$ die Gelenkmomente bzw. -kräfte darstellen.
Die Coriolis-Matrix C ist:

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} \end{bmatrix}$$

$$\begin{aligned}
 C_{11} &= 16.3\ddot{d}h_3 + 17.676\ddot{d}h_3dh_3 - 16.3\ddot{d}h_3 \cos(2\varphi_1) \\
 &\quad + 17.991\dot{\varphi}_1 \cos(2\varphi_1) + 8.517\ddot{d}h_3 \sin(2\varphi_1) \\
 &\quad + 4.946\dot{\varphi}_1 \sin(2\varphi_1) + 17.676dh_3^2\dot{\varphi}_1 \sin(2\varphi_1) \\
 &\quad + 22.947\dot{\varphi}_1 \cos(2\varphi_1) \cos(\varphi_2) \\
 &\quad + 10.646\ddot{d}h_3 \sin(2\varphi_1) \cos(\varphi_2) \\
 &\quad - 15.835\dot{\varphi}_1 \sin(2\varphi_1) \cos(\varphi_2) \\
 &\quad - 17.676\ddot{d}h_3dh_3 \cos(2\varphi_1) + 17.033dh_3\dot{\varphi}_1 \cos(2\varphi_1) \\
 &\quad + 32.601dh_3\dot{\varphi}_1 \sin(2\varphi_1) \\
 &\quad - 4.777\dot{\varphi}_1 \cos(2\varphi_2) \sin(2\varphi_1) \\
 &\quad + 21.292dh_3\dot{\varphi}_1 \cos(2\varphi_1) \cos(\varphi_2) \\
 C_{12} &= 9.723\dot{\varphi}_0 \sin(2\varphi_1) - 17.033dh_3\dot{\varphi}_0 - 17.991\dot{\varphi}_0 \\
 &\quad - 22.947\dot{\varphi}_0 \cos(\varphi_2) - 19.231\dot{\varphi}_2 \sin(\varphi_1) \\
 &\quad + 0.0457\dot{\varphi}_4 \sin(\varphi_1) + 35.981\dot{\varphi}_0 \cos^2(\varphi_1) \\
 &\quad + 17.676dh_3^2\dot{\varphi}_0 \sin(2\varphi_1) \\
 &\quad + 34.067dh_3\dot{\varphi}_0 \cos^2(\varphi_1) \\
 &\quad - 24.907\ddot{d}h_3 \cos(\varphi_1) \sin(\varphi_2) \\
 &\quad + 15.835\dot{\varphi}_1 \cos(\varphi_1) \sin(\varphi_2) \\
 &\quad + 22.947\dot{\varphi}_1 \sin(\varphi_1) \sin(\varphi_2) \\
 &\quad + 32.601dh_3\dot{\varphi}_0 \sin(2\varphi_1) \\
 &\quad + 45.895\dot{\varphi}_0 \cos^2(\varphi_1) \cos(\varphi_2) \\
 C_{13} &= 9.555\dot{\varphi}_0 \sin(2\varphi_2) - 19.231\dot{\varphi}_1 \sin(\varphi_1) \\
 &\quad - 3.616\ddot{d}h_3 \cos(\varphi_2) \sin(\varphi_1) \\
 &\quad - 15.835\dot{\varphi}_2 \cos(\varphi_1) \sin(\varphi_2) \\
 &\quad - 22.947\dot{\varphi}_2 \sin(\varphi_1) \sin(\varphi_2) \\
 &\quad - 15.835\dot{\varphi}_0 \cos^2(\varphi_1) \sin(\varphi_2) \\
 C_{14} &= \dot{\varphi}_0(35.352dh_3 + 8.517 \sin(2\varphi_1) - 32.601 \cos^2(\varphi_1)) \\
 &\quad - 24.907\dot{\varphi}_1 \cos(\varphi_1) \sin(\varphi_2) \\
 &\quad - 3.616\dot{\varphi}_2 \cos(\varphi_2) \sin(\varphi_1) \\
 C_{15} &= 0.0457\dot{\varphi}_1 \sin(\varphi_1)
 \end{aligned}$$

$$\begin{aligned}
 C_{21} &= 17.991\dot{\varphi}_0 + 17.033dh_3\dot{\varphi}_0 - 9.723\dot{\varphi}_0 \sin(2\varphi_1) \\
 &\quad + 22.947\dot{\varphi}_0 \cos(\varphi_2) + 0.121\dot{\varphi}_2 \sin(\varphi_1) \\
 &\quad - 0.0457\dot{\varphi}_4 \sin(\varphi_1) - 35.981\dot{\varphi}_0 \cos^2(\varphi_1) \\
 C_{22} &= \ddot{d}h_3(35.352dh_3 + 32.601) \\
 &\quad - \dot{\varphi}_2(9.555 \sin(2\varphi_2) + 15.835 \sin(\varphi_2)) \\
 C_{23} &= 3.616\ddot{d}h_3 \sin(\varphi_2) - 22.947\dot{\varphi}_2 \cos(\varphi_2) \\
 &\quad - 9.555\dot{\varphi}_1 \sin(2\varphi_2) + 0.121\dot{\varphi}_0 \sin(\varphi_1) \\
 C_{24} &= \dot{\varphi}_1(35.352dh_3 + 32.601) + 14.261\dot{\varphi}_2 \sin(\varphi_2) \\
 C_{25} &= -0.0457\dot{\varphi}_0 \sin(\varphi_1) \\
 C_{31} &= 22.947\dot{\varphi}_1 \cos(\varphi_1) \cos(\varphi_2) - 0.121\dot{\varphi}_1 \sin(\varphi_1) \\
 &\quad - 9.555\dot{\varphi}_0 \sin(2\varphi_2) \\
 C_{32} &= 15.835\dot{\varphi}_1 \sin(\varphi_2) - 0.121\dot{\varphi}_0 \sin(\varphi_1) \\
 &\quad - 24.907\ddot{d}h_3 \sin(\varphi_2) \\
 C_{33} &= 0, \quad C_{34} = 24.907\dot{\varphi}_0 \cos(\varphi_2) \sin(\varphi_1) \\
 &\quad - 24.907\dot{\varphi}_1 \sin(\varphi_2), \quad C_{35} = 0 \\
 C_{41} &= -\dot{\varphi}_0(35.352dh_3 + 8.517 \sin(2\varphi_1) \\
 &\quad + \dot{\varphi}_0(32.601 \cos^2(\varphi_1) - 21.292 \cos(\varphi_1) \cos(\varphi_2) \sin(\varphi_1)) \\
 C_{42} &= 24.907\dot{\varphi}_2 \sin(\varphi_2) - 35.352dh_3\dot{\varphi}_1 - 32.601\dot{\varphi}_1 \\
 C_{43} &= 24.907\dot{\varphi}_1 \sin(\varphi_2) - 24.907\dot{\varphi}_0 \cos(\varphi_2) \sin(\varphi_1) \\
 C_{44} &= 0, \quad C_{45} = 0 \\
 C_{51} &= 0.0457\dot{\varphi}_1 \sin(\varphi_1), \quad C_{52} = 0.0457\dot{\varphi}_0 \sin(\varphi_1) \\
 C_{53} &= 0, \quad C_{54} = 0, \quad C_{55} = 0
 \end{aligned}$$

4.2.5 Berechnung des Gravitationsvektors $\mathbf{G}(\mathbf{q})$

Die Gravitationskräfte wirken auf jeden Körper des Roboters entsprechend seiner Masse und der Position seines Schwerpunkts relativ zum Basiskoordinatensystem. Der Gravitationsvektor $\mathbf{G}(\mathbf{q})$ wird aus der potentiellen Energie V abgeleitet.

Gesamtpotentielle Energie V

Die Gesamtpotentielle Energie ergibt sich durch die Summe der potentiellen Energie jeder Körpermasse:

$$V = \sum_{k=1}^n m_k \cdot \mathbf{g}^\top \cdot \mathbf{r}_k \tag{4.50}$$

Dabei ist:

- m_k die Masse des k -ten Körpers,
- $\mathbf{g} = [0 \quad 0 \quad -g]^\top$ der Gravitationsvektor im Basis-KS,
- \mathbf{r}_k die Position des Schwerpunkts des k -ten Körpers im Basis-Koordinatensystem.

Schwerpunktberechnung

Die Schwerpunktposition \mathbf{r}_k jedes Körpers wird wie folgt bestimmt:

1. Die Transformationsmatrix \mathbf{T}_k von der Basis bis zum k -ten Körper wird berechnet durch:

$$\mathbf{T}_k = \prod_{m=1}^k \mathbf{A}_m \quad (4.51)$$

wobei \mathbf{A}_m die homogene Transformationsmatrix des m -ten Gelenks ist.

2. Der lokale Schwerpunkt wird entlang der Hauptachse des Körpers bestimmt:

$$\mathbf{r}_{\text{lokal}} = \begin{cases} \begin{bmatrix} \frac{L_k}{2} \\ 0 \\ 0 \end{bmatrix}, & \text{falls Richtung X} \\ \begin{bmatrix} 0 \\ \frac{L_k}{2} \\ 0 \end{bmatrix}, & \text{falls Richtung Y} \\ \begin{bmatrix} 0 \\ 0 \\ \frac{L_k}{2} \end{bmatrix}, & \text{falls Richtung Z} \end{cases}$$

Bei CAD-basierten Geometrien (Typ 3 oder 4) wird der Schwerpunkt mit festen Werten überschrieben.

3. Transformation des Schwerpunkts in das Basiskoordinatensystem:

$$\mathbf{r}_k = \mathbf{R}_k \cdot \mathbf{r}_{\text{lokal}} + \mathbf{p}_k \quad (4.52)$$

wobei \mathbf{R}_k die Rotationsmatrix und \mathbf{p}_k der Positionsvektor aus \mathbf{T}_k sind.

Berechnung des Gravitationsvektors $\mathbf{G}(\mathbf{q})$

Der Gravitationsvektor ergibt sich aus den partiellen Ableitungen der potentiellen Energie V nach den generalisierten Koordinaten q_i :

$$G_i(q) = \frac{\partial V}{\partial q_i}, \quad \text{für } i = 1, 2, \dots, n \quad (4.53)$$

Daraus ergibt sich der vollständige Gravitationsvektor:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \frac{\partial V}{\partial q_1} \\ \frac{\partial V}{\partial q_2} \\ \vdots \\ \frac{\partial V}{\partial q_n} \end{bmatrix} \quad (4.54)$$

Berechnete Gravitationsmatrix $\mathbf{G}(\mathbf{q})$ (gerundet)

Die berechnete und gerundete Gravitationsmatrix ergibt sich zu:

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} 0 \\ -420,41 \cdot \cos(\varphi_1) - 703,61 \cdot \sin(\varphi_1) - 442,11 \cdot \cos(\varphi_1) \cdot \cos(\varphi_2) - 559,62 \cdot dh_3 \cdot \sin(\varphi_1) \\ 442,11 \cdot \sin(\varphi_1) \cdot \sin(\varphi_2) \\ 559,62 \cdot \cos(\varphi_1) \\ 0 \end{bmatrix} \quad (4.55)$$

4.2.6 Numerische Auswertung der Lagrange-Gleichungen während der ersten zwei Sekunden

Die numerische Auswertung basiert auf der Gleichung:

$$\tau_{\text{eval}}(:, i) = M_i \cdot \ddot{\mathbf{q}}_{\text{num}}^T + C_i \cdot \dot{\mathbf{q}}_{\text{num}}^T + G_i^T$$

Zeitpunkt: 0.000 s

Zustände:

$$\begin{aligned} q &= [0.0002 \quad 0.0001 \quad 0.0001 \quad 0.0000 \quad 0.0006] \\ \dot{q} &= [0.0464 \quad 0.0265 \quad 0.0250 \quad 0.0074 \quad 0.1417] \\ \ddot{q} &= [6.2832 \quad 3.1416 \quad 5.0000 \quad 0.8000 \quad 15.7080] \end{aligned}$$

Ergebnis:

$$\tau = \begin{bmatrix} 594.2270 \\ -588.5931 \\ 316.4820 \\ 443.9385 \\ 0.4044 \end{bmatrix}$$

$\mathbf{M}_i \cdot \ddot{\mathbf{q}}$:

$$\begin{bmatrix} 594.1264 \\ 274.1412 \\ 316.4256 \\ -115.6566 \\ 0.4044 \end{bmatrix}$$

\mathbf{M}_i :

$$\begin{bmatrix} 66.7845 & -0.0019 & 35.1897 & -0.0000 & -0.0914 \\ -0.0019 & 100.3396 & 0 & -51.3412 & 0 \\ 35.1897 & 0 & 19.3516 & 0 & -0.0914 \\ -0.0000 & -51.3412 & 0 & 57.0456 & 0 \\ -0.0914 & 0 & -0.0914 & 0 & 0.0914 \end{bmatrix}$$

$\mathbf{C}_i \cdot \dot{\mathbf{q}}$:

$$\begin{bmatrix} 0.1006 \\ -0.1429 \\ 0.0564 \\ -0.0229 \\ 0.0000 \end{bmatrix}$$

\mathbf{C}_i :

$$\begin{bmatrix} 1.0841 & 1.8988 & -0.0001 & 0.0001 & 0.0000 \\ -2.4723 & 0.2399 & -1.6380 & 0.8634 & -0.0000 \\ 0.6077 & 1.0643 & 0 & 0.0001 & 0 \\ -0.0003 & -0.8633 & -0.0001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

\mathbf{G}_i :

$$\begin{bmatrix} 0 \\ -862.5914 \\ 0.0000 \\ 559.6180 \\ 0 \end{bmatrix}$$

Zeitpunkt: 1.000 s

$$\begin{aligned} q &= [1.3860 \ 1.1912 \ 2.2545 \ 0.3158 \ 2.5939] \\ \dot{q} &= [1.5708 \ 1.5443 \ 0.5182 \ 0.2660 \ -2.4979] \\ \ddot{q} &= [0.0000 \ -3.1416 \ -9.4248 \ -0.8000 \ -15.7080] \end{aligned}$$

$$\tau = \begin{bmatrix} 285.6724 \\ -1050.5847 \\ 136.3562 \\ 25.8723 \\ -0.3683 \end{bmatrix}$$

$$M_i \cdot \ddot{q} = \begin{bmatrix} 166.6556 \\ -188.7121 \\ -180.9488 \\ -30.5560 \\ -0.5743 \end{bmatrix}$$

\mathbf{M}_i :

$$\begin{bmatrix} 66.5447 & -5.8124 & -13.9455 & -20.5374 & -0.0339 \\ -5.8124 & 61.2913 & 0 & -4.8003 & 0 \\ -13.9455 & 0 & 19.3516 & 0 & -0.0914 \\ -20.5374 & -4.8003 & 0 & 57.0458 & 0 \\ -0.0339 & 0 & -0.0914 & 0 & 0.0914 \end{bmatrix}$$

$$C_i \cdot \dot{q} = \begin{bmatrix} 119.0168 \\ 8.0927 \\ -1.0264 \\ -150.9227 \\ 0.2059 \end{bmatrix}$$

\mathbf{C}_i :

$$\begin{bmatrix} 34.9190 & 52.0392 & -57.2724 & 51.2954 & 0.0656 \\ -22.0759 & 10.1274 & 13.5682 & 74.9463 & -0.0667 \\ 16.3884 & -8.2456 & 0 & -52.7741 & 0 \\ -55.2755 & -59.2123 & 52.7741 & 0 & 0 \\ 0.0656 & 0.0667 & 0 & 0 & 0 \end{bmatrix}$$

$$G_i = \begin{bmatrix} 0 \\ -869.9652 \\ 318.3314 \\ 207.3510 \\ 0 \end{bmatrix}$$

Zeitpunkt: 2.000 s

$$\begin{aligned} q &= [2.5992 \quad 1.1806 \quad -0.0995 \quad 0.0000 \quad -2.6601] \\ \dot{q} &= [-0.4862 \quad -1.5659 \quad -3.1416 \quad 0.0000 \quad -2.0400] \\ \ddot{q} &= [-6.2832 \quad -3.1416 \quad 0.0000 \quad 0.0000 \quad 15.7080] \end{aligned}$$

$$\tau = \begin{bmatrix} -471.6292 \\ -1387.5987 \\ -302.7991 \\ 173.3905 \\ 1.7185 \end{bmatrix}$$

$$M_i \cdot \ddot{q} = \begin{bmatrix} -425.7371 \\ -299.4590 \\ -218.0343 \\ 144.3780 \\ 1.6542 \end{bmatrix}$$

M_i:

$$\begin{bmatrix} 68.8390 & -2.3356 & 34.4727 & 2.6215 & -0.0348 \\ -2.3356 & 99.9919 & 0 & -51.2000 & 0 \\ 34.4727 & 0 & 19.3516 & 0 & -0.0914 \\ 2.6215 & -51.2000 & 0 & 57.0458 & 0 \\ -0.0348 & 0 & -0.0914 & 0 & 0.0914 \end{bmatrix}$$

$$C_i \cdot \dot{q} = \begin{bmatrix} -45.8920 \\ -110.1535 \\ -44.1310 \\ -183.8704 \\ 0.0644 \end{bmatrix}$$

C_i:

$$\begin{bmatrix} 62.4952 & 25.5843 & -7.7731 & -11.1162 & -0.0662 \\ -93.1355 & -10.8809 & 54.8870 & -46.5288 & 0.0205 \\ 36.5074 & 16.8475 & 0 & -15.0202 & 0 \\ 91.8878 & 58.7579 & 15.0202 & 0 & 0 \\ -0.0662 & -0.0205 & 0 & 0 & 0 \end{bmatrix}$$

$$G_i = \begin{bmatrix} 0 \\ -977.9862 \\ -40.6338 \\ 212.8829 \\ 0 \end{bmatrix}$$

4.3 Bestimmung der Bewegungsprofile (Position, Geschwindigkeit, Beschleunigung und Ruck)

Um eine präzise und effiziente Bewegungssteuerung zu gewährleisten, müssen die zeitlichen Verläufe von Position $q(t)$, Geschwindigkeit $\dot{q}(t)$, Beschleunigung $\ddot{q}(t)$ und Ruck $\dddot{q}(t)$ für jeden Gelenk- oder Achsenantrieb bestimmt werden. Diese Bewegungsprofile sind entscheidend für:

- Minimierung mechanischer Belastungen (durch Begrenzung von Beschleunigung und Ruck)
- Vermeidung unerwünschter Schwingungen (durch glatte Übergänge)
- Einhaltung von Geschwindigkeits- und Positionsgrenzen

Mathematische Beschreibung Die Trajektorien werden typischerweise durch Polynome (z. B. kubische oder quintische Splines) oder trigonometrische Funktionen modelliert. Ein häufiger Ansatz ist die Verwendung von S-Kurven-Profilen (Trapez- oder Sinusrampen), um den Ruck zu begrenzen.

Für einen Bewegungsabschnitt von q_0 nach q_f in der Zeit t_f können die Profile wie folgt definiert werden:

Position

$$q(t) = q_0 + (q_f - q_0) \cdot s(t) \quad (4.56)$$

wobei $s(t)$ eine normierte Übergangsfunktion ist (z. B. ein Polynom 5. Grades für glatte Übergänge).

Geschwindigkeit

$$\dot{q}(t) = (q_f - q_0) \cdot \dot{s}(t) \quad (4.57)$$

Beschleunigung

$$\ddot{q}(t) = (q_f - q_0) \cdot \ddot{s}(t) \quad (4.58)$$

Ruck (Jerk)

$$\dddot{q}(t) = (q_f - q_0) \cdot \dddot{s}(t) \quad (4.59)$$

Praktische Umsetzung mit Ruckig Zur numerischen Erzeugung dieser Bewegungsprofile verwende ich den Ruckig-Trajektoriengenerator in Kombination mit ROS (Robot Operating System). Dieses Tool ermöglicht die effiziente Berechnung von ruckbegrenzten Trajektorien in Echtzeit, basierend auf den aktuellen kinematischen Zuständen und Begrenzungen.

Evaluierung der Profile Zur Evaluierung und Überprüfung der erzeugten Profile orientiere ich mich am Entscheidungsdiagramm aus der Vorlesung von Prof. Ketterer, welches die Auswahl des geeigneten Bewegungsfalls (Fall I, II, III etc.) auf Basis von Weglänge, Maximalgeschwindigkeit, Beschleunigung und Ruck systematisch darstellt.

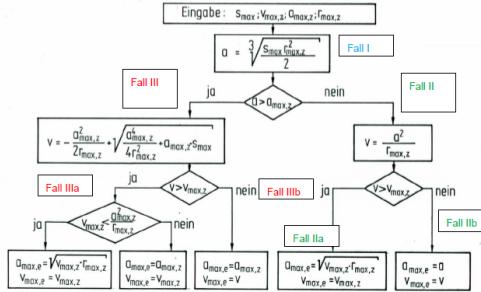


Abbildung 4.5: Entscheidungsdiagramm

4.3.1 Ergebnisdarstellung der Bewegungsprofile

Variable	Minimum	Maximum	Mittelwert
φ_0			
Position (rad)	-2.618	2.618	0.002
Geschwindigkeit (rad/s)	-1.571	1.571	0.000
Beschleunigung (rad/s ²)	-6.283	6.283	0.000
Ruck (rad/s ³)	-628.319	628.319	-0.843
φ_1			
Position (rad)	0.000	1.571	0.785
Geschwindigkeit (rad/s)	-1.571	1.571	0.000
Beschleunigung (rad/s ²)	-3.142	3.142	0.000
Ruck (rad/s ³)	-314.159	314.159	-1.040
φ_2			
Position (rad)	-2.269	2.269	0.001
Geschwindigkeit (rad/s)	-3.142	3.142	0.000
Beschleunigung (rad/s ²)	-9.425	9.425	-0.001
Ruck (rad/s ³)	-500.000	500.000	-1.263
dh_3			
Position (m)	0.00003	0.360	0.182
Geschwindigkeit (m/s)	0.000	0.535	0.267
Beschleunigung (m/s ²)	-0.800	0.800	0.000
Ruck (m/s ³)	-160.000	80.000	-0.593
φ_4			
Position (rad)	-2.793	2.793	-0.005
Geschwindigkeit (rad/s)	-6.283	6.283	0.000
Beschleunigung (rad/s ²)	-15.708	15.708	0.000
Ruck (rad/s ³)	-1570.796	1570.796	-5.236

Tabelle 4.2: Zusammenfassung der Trajektorienstatistik für Position, Geschwindigkeit, Beschleunigung und Ruck

Trajectory Analysis: φ_0

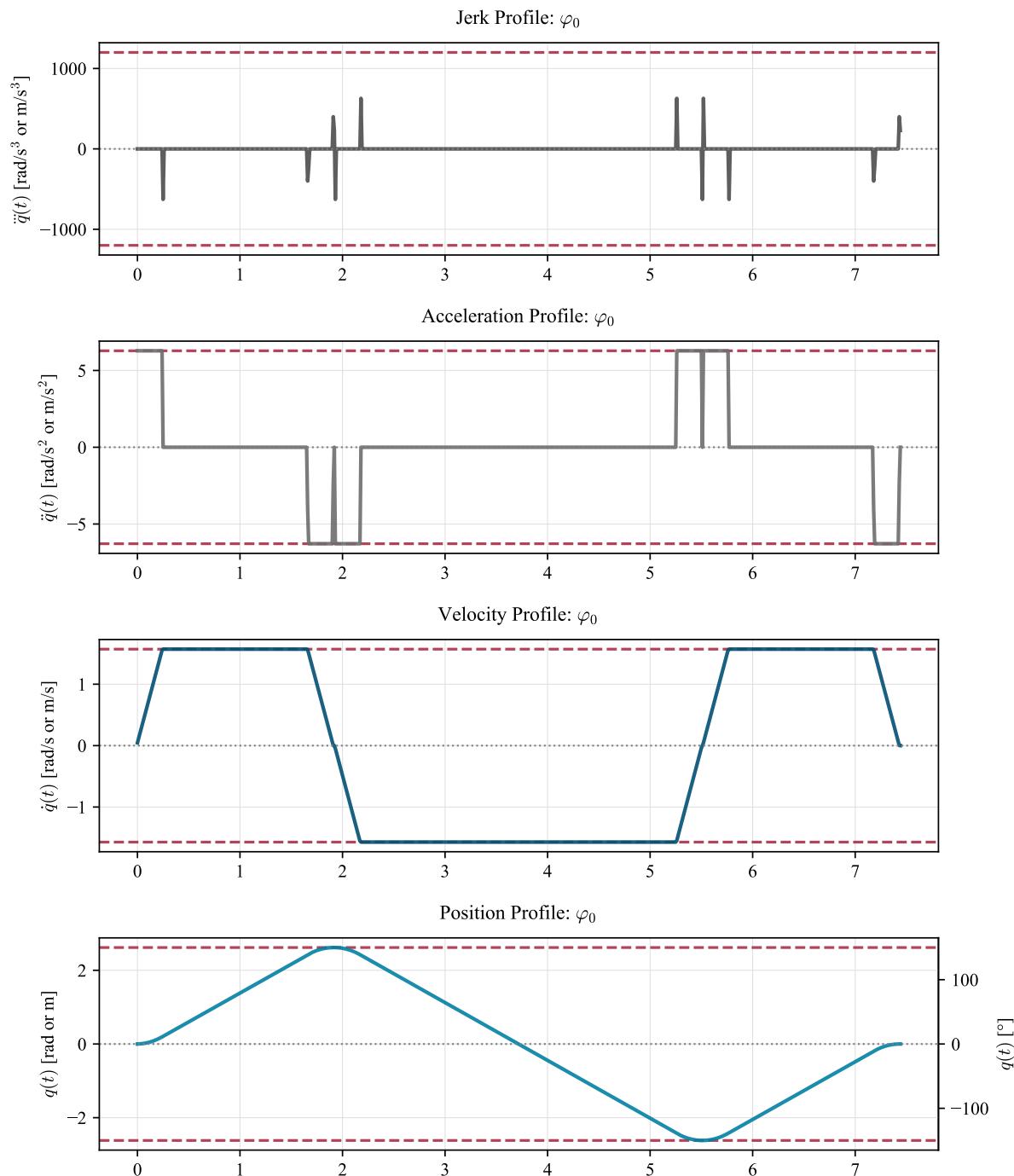


Abbildung 4.6: Bewegungsprofile φ_0

Trajectory Analysis: φ_1

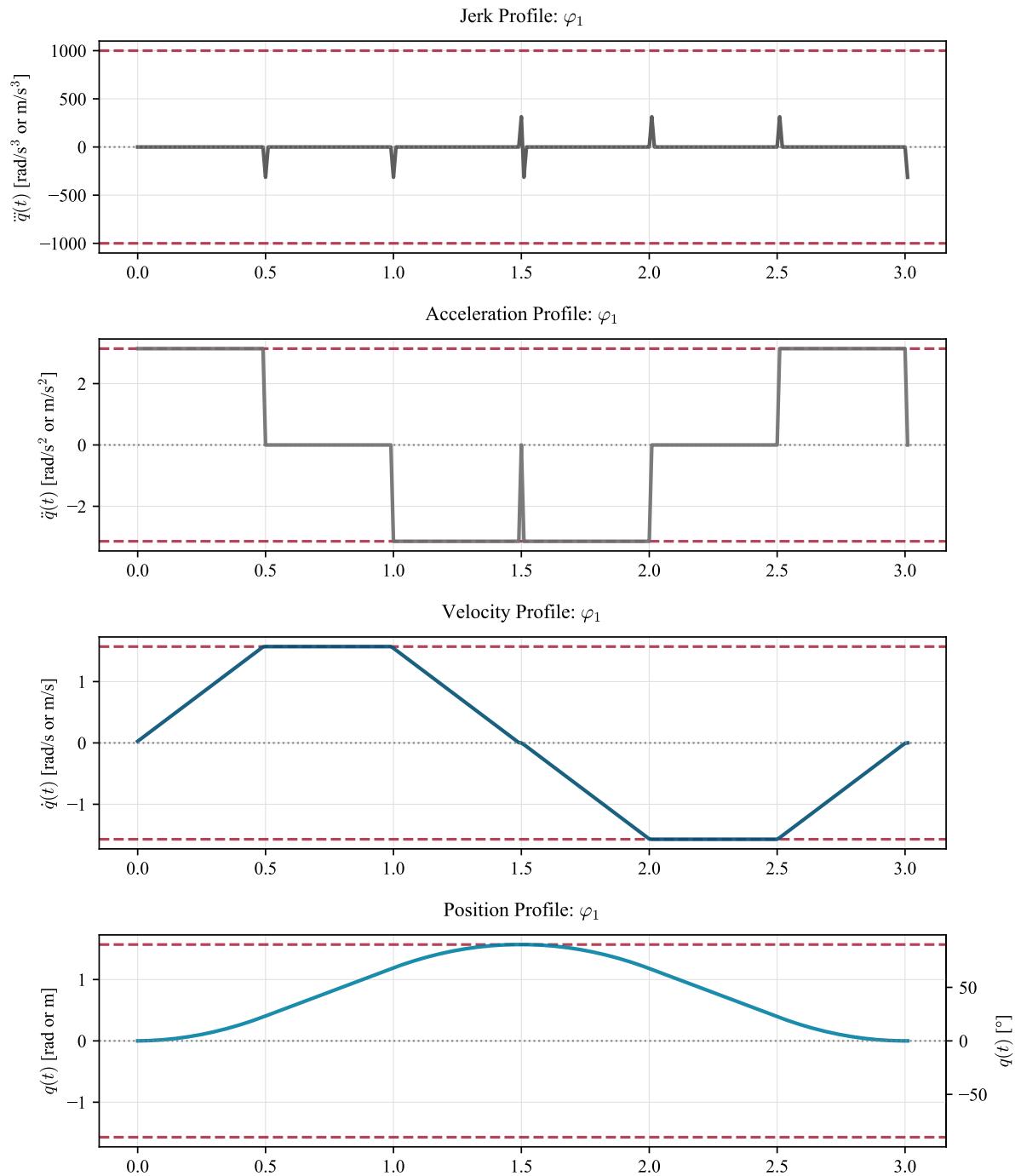


Abbildung 4.7: Bewegungsprofile φ_1

Trajectory Analysis: φ_2

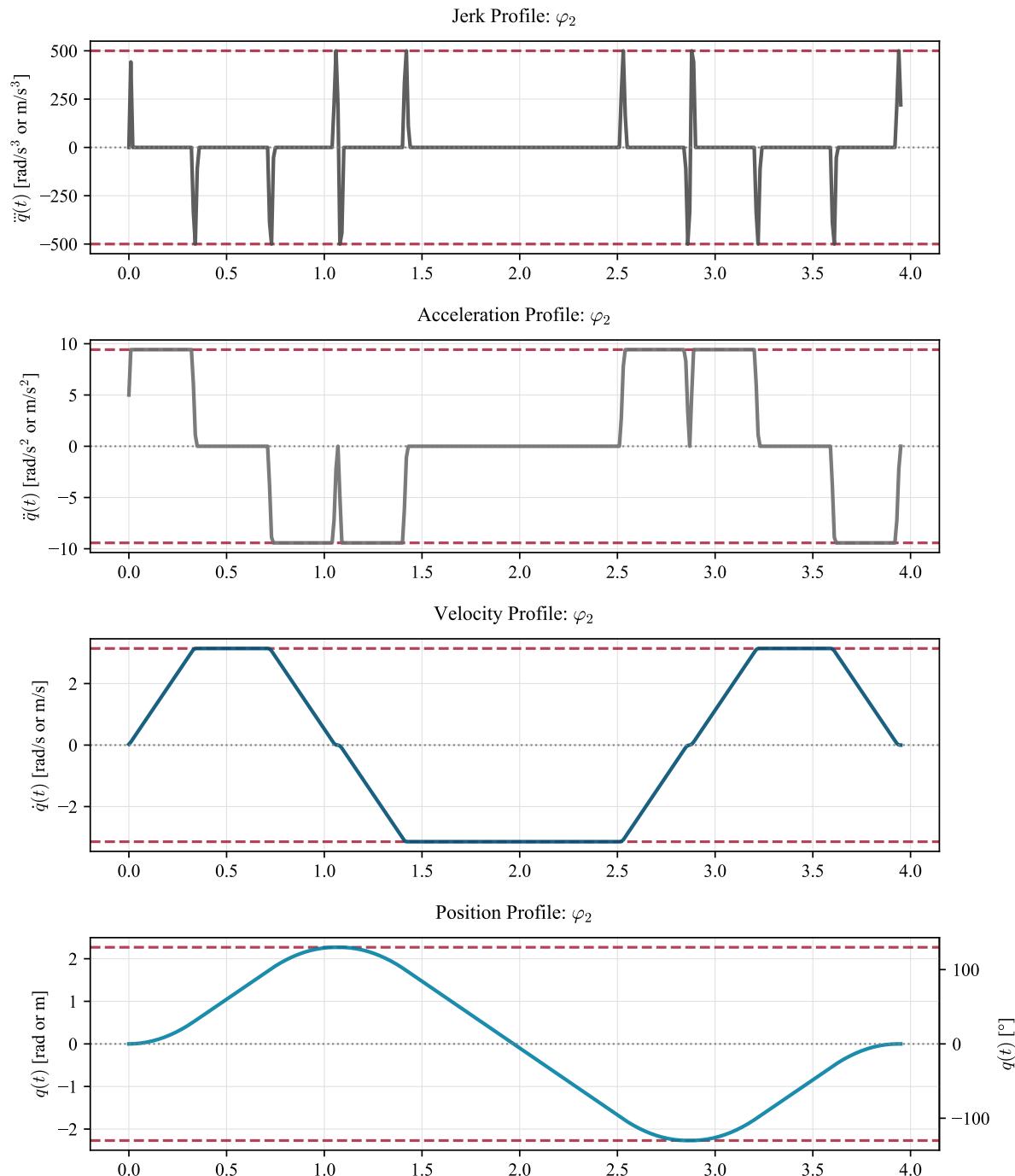


Abbildung 4.8: Bewegungsprofile φ_2

Trajectory Analysis: d_{h3}

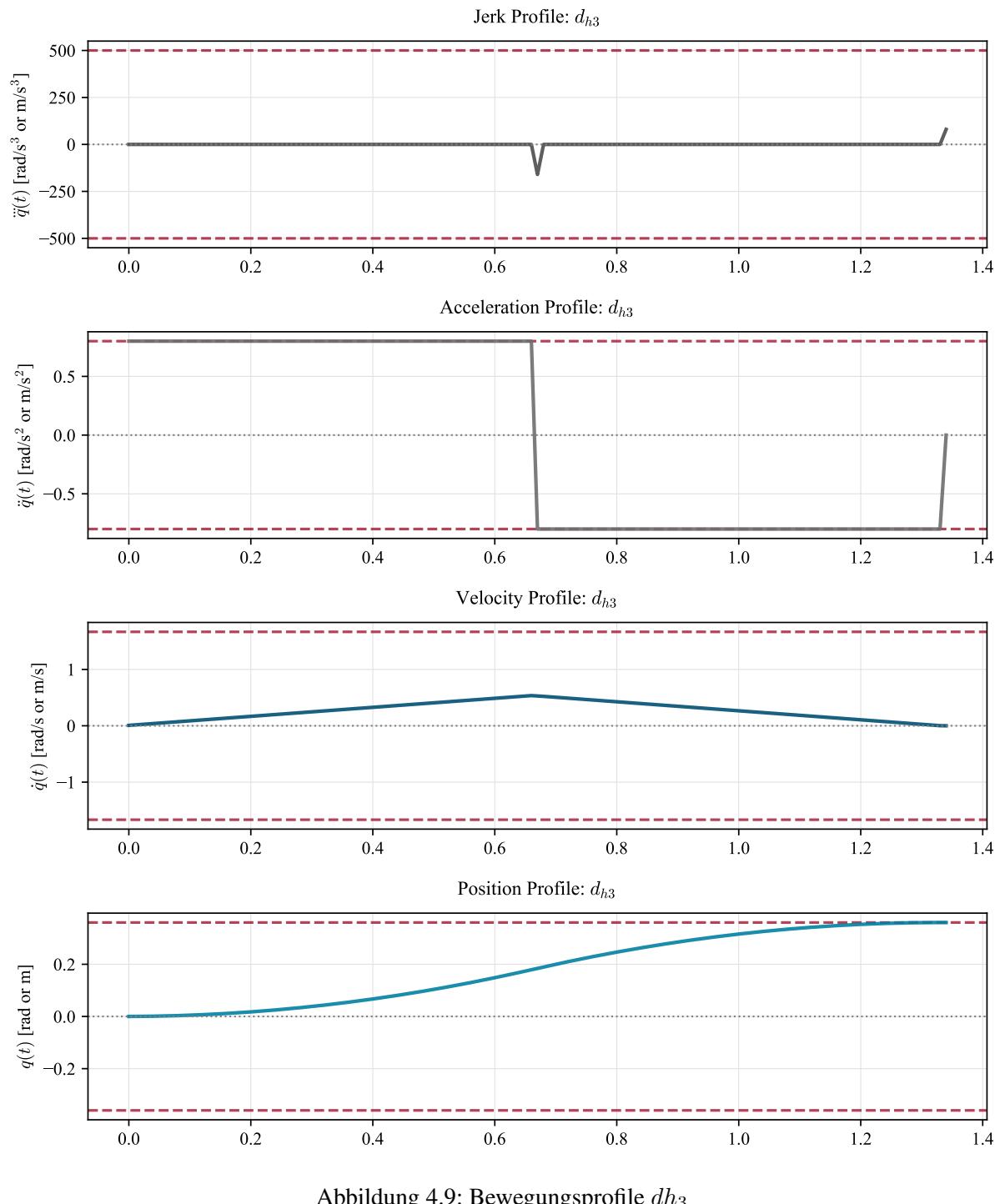


Abbildung 4.9: Bewegungsprofile d_{h3}

Trajectory Analysis: φ_4

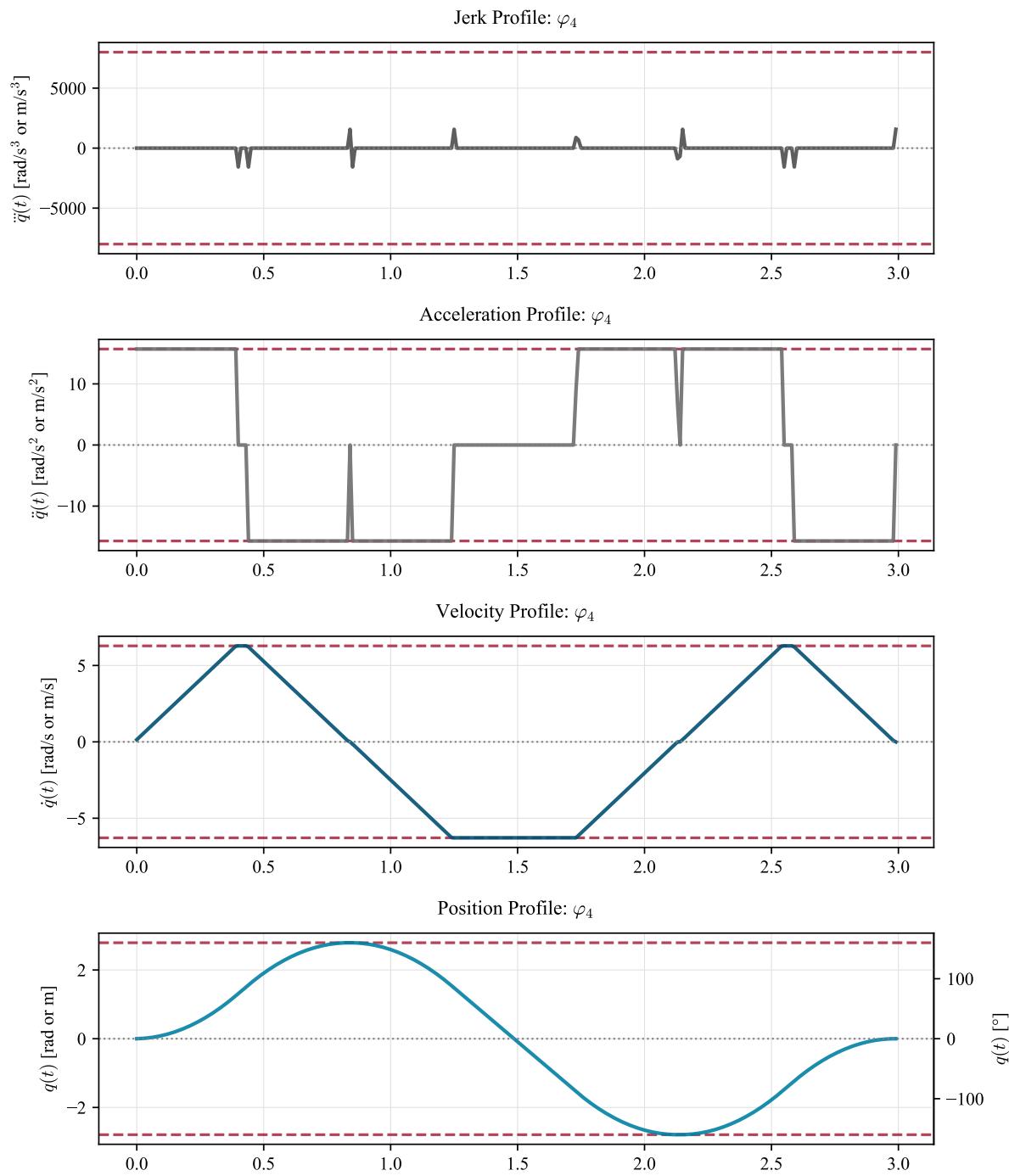


Abbildung 4.10: Bewegungsprofile φ_4

4.3.2 Auswertung von $\tau(t)$ über die Zeit – Lagrange ausgewertet mit Bewegungsprofilen

In diesem Abschnitt wird das zeitabhängige Verhalten der verallgemeinerten Gelenkmomente $\tau(t)$ untersucht, welche durch die Lagrange-Gleichung zweiter Art berechnet wurden. Die Auswertung erfolgt auf Basis vorgegebener Bewegungsprofile für Position, Geschwindigkeit und Beschleunigung jeder Achse des Roboters. Die verwendete Gleichung lautet:

$$\tau(t) = \mathbf{M}(\mathbf{q}(t)) \cdot \ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t)) \cdot \dot{\mathbf{q}}(t) + \mathbf{G}(\mathbf{q}(t)) \quad (4.60)$$

Dabei stellen $\mathbf{M}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ und $\mathbf{G}(\mathbf{q})$ die Trägheitsmatrix, die Coriolis-Zentrifugal-Matrix und den Gravitationsvektor dar, welche zu jedem Zeitpunkt anhand der aktuellen Gelenkstellungen $q(t)$ und ihrer Ableitungen berechnet werden.

Die Auswertung erlaubt die numerische Bestimmung der erforderlichen Antriebsmomente $\tau_i(t)$ für jede Achse im Zeitverlauf.

Die Auswertung basiert auf einer näherungsweisen Berechnung mit der abgeleiteten Lagrange-Gleichung. Aufgrund der symbolischen Ableitungen wurden vereinfachte Ausdrücke verwendet, bei denen Variablen mit sehr kleinen numerischen Koeffizienten während der symbolischen Berechnung vernachlässigt wurden. Dies führt zu einer Differenz im Vergleich zur direkten numerischen Berechnung, bei der die Werte frühzeitig substituiert und ohne symbolische Vereinfachungen ausgewertet werden.

Optimalerweise sollte die Berechnung direkt numerisch erfolgen, um präzise und konsistente Ergebnisse zu erhalten, ohne Einflüsse symbolischer Vereinfachungen. Um jedoch die Ergebnisse visuell darstellen zu können, wurden symbolische Vereinfachungen durchgeführt, was zu einer Abweichung zwischen den beiden Ansätzen führte. Im Folgenden werden die berechneten Gelenkmomente τ sowohl mit als auch ohne symbolische Vereinfachung für jede verallgemeinerte Koordinate dargestellt.

Annäherungsergebnisse aufgrund symbolischer Vereinfachungen Aufgrund der in MATLAB durchgeföhrten Vereinfachungen zur besseren Visualisierung und Analyse wurden die Ergebnisse von τ symbolisch vereinfacht. Dies führt zu leicht abweichenden, aber gut interpretierbaren Näherungsergebnissen.

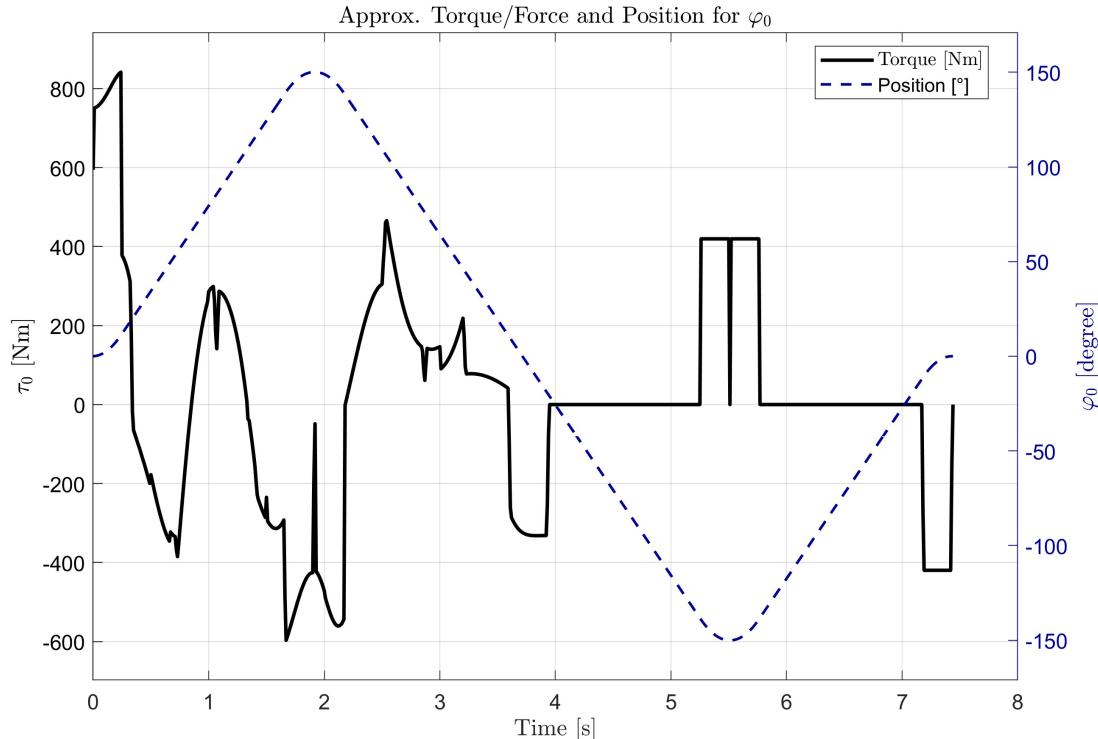


Abbildung 4.11: Generalisierten Gelenkmomente $\varphi_0 - \tau$

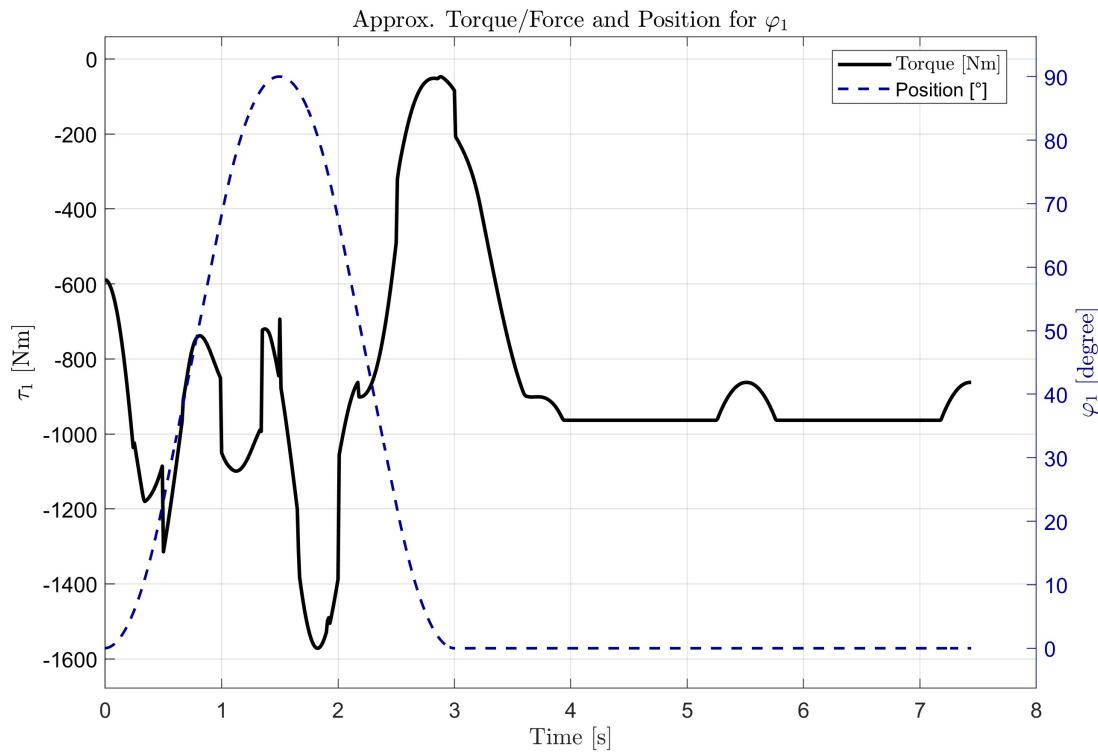


Abbildung 4.12: Generalisierten Gelenkmomente $\varphi_1 - \tau$

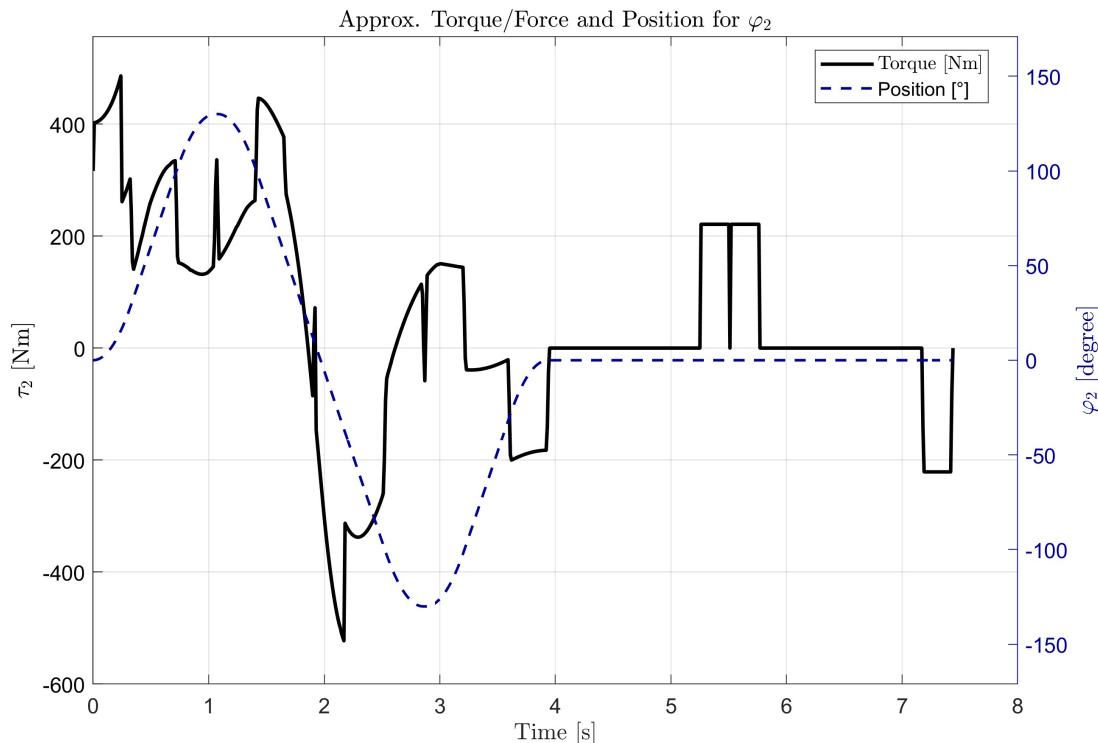
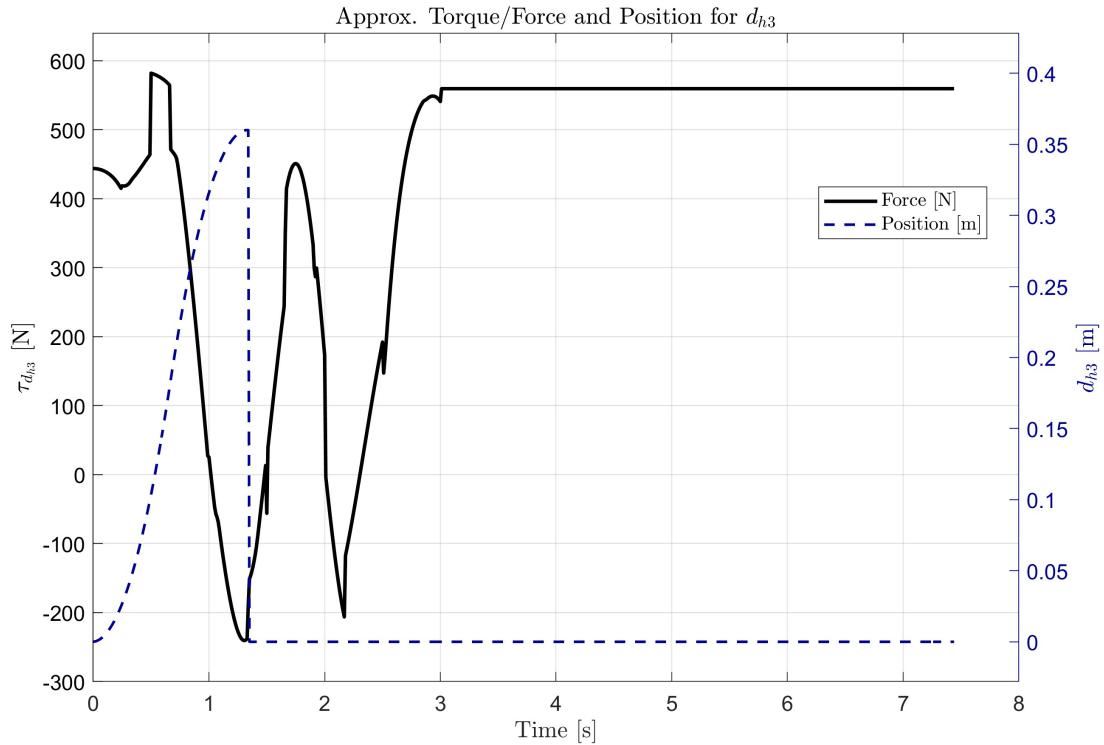
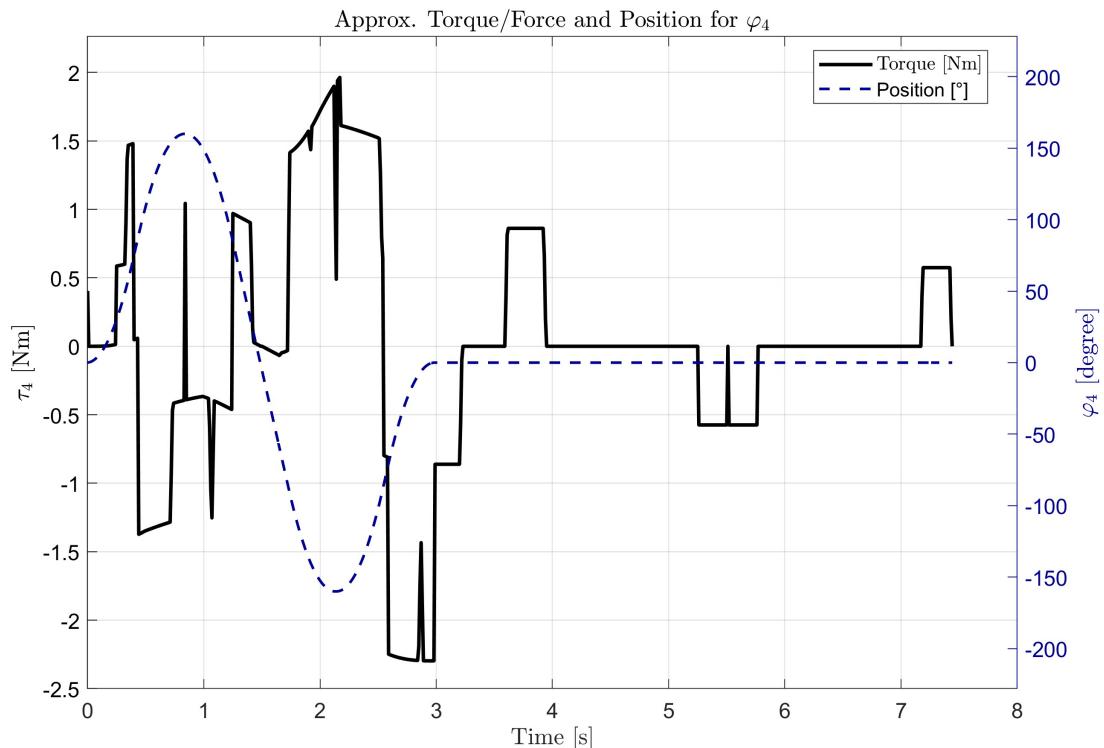


Abbildung 4.13: Generalisierten Gelenkmomente $\varphi_2 - \tau$


 Abbildung 4.14: Generalisierten Gelenkkraft $d_{h3} - \tau$

 Abbildung 4.15: Generalisierten Gelenkmomente $\varphi_4 - \tau$

Präzise Ergebnisse durch direkte numerische Berechnung Die Gelenkmomente τ wurden direkt numerisch berechnet, wobei alle Werte von Anfang an vollständig eingesetzt wurden. Dadurch werden Rundungsfehler und symbolische Vereinfachungsverluste vermieden, was eine robuste und genaue Auswertung der Lagrange-Gleichung ermöglicht.

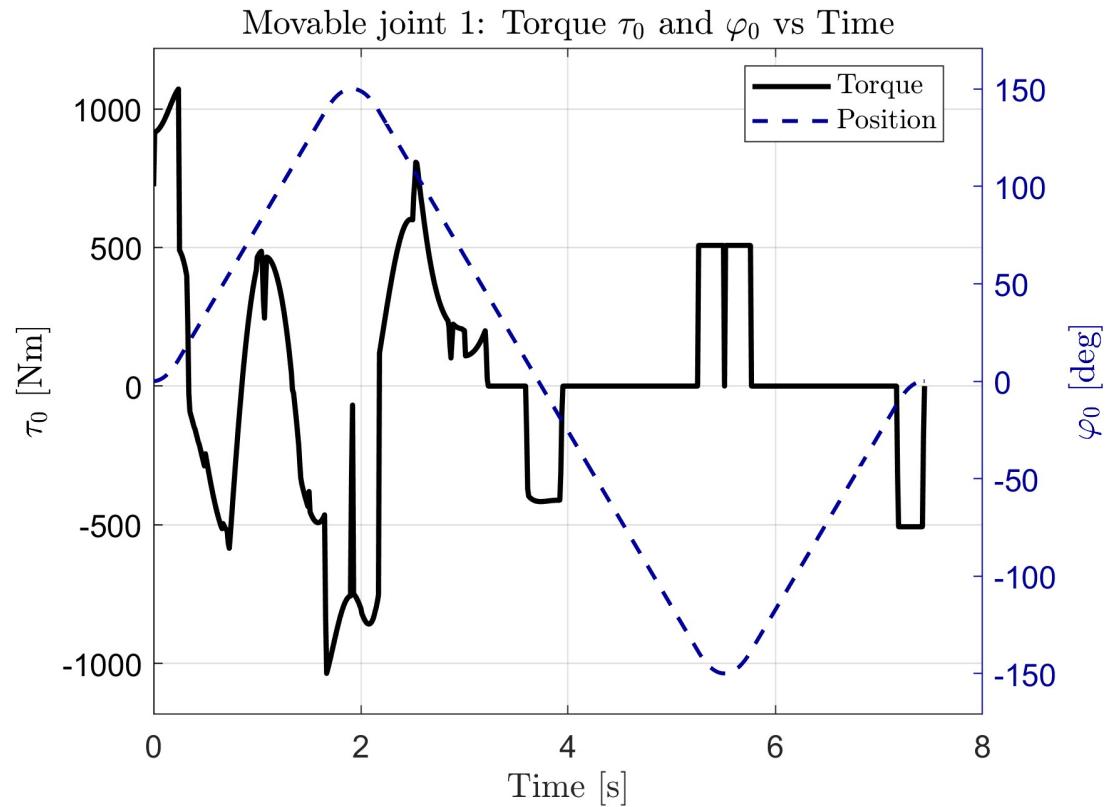


Abbildung 4.16: Generalisierten Gelenkmomente $\varphi_0 - \tau$

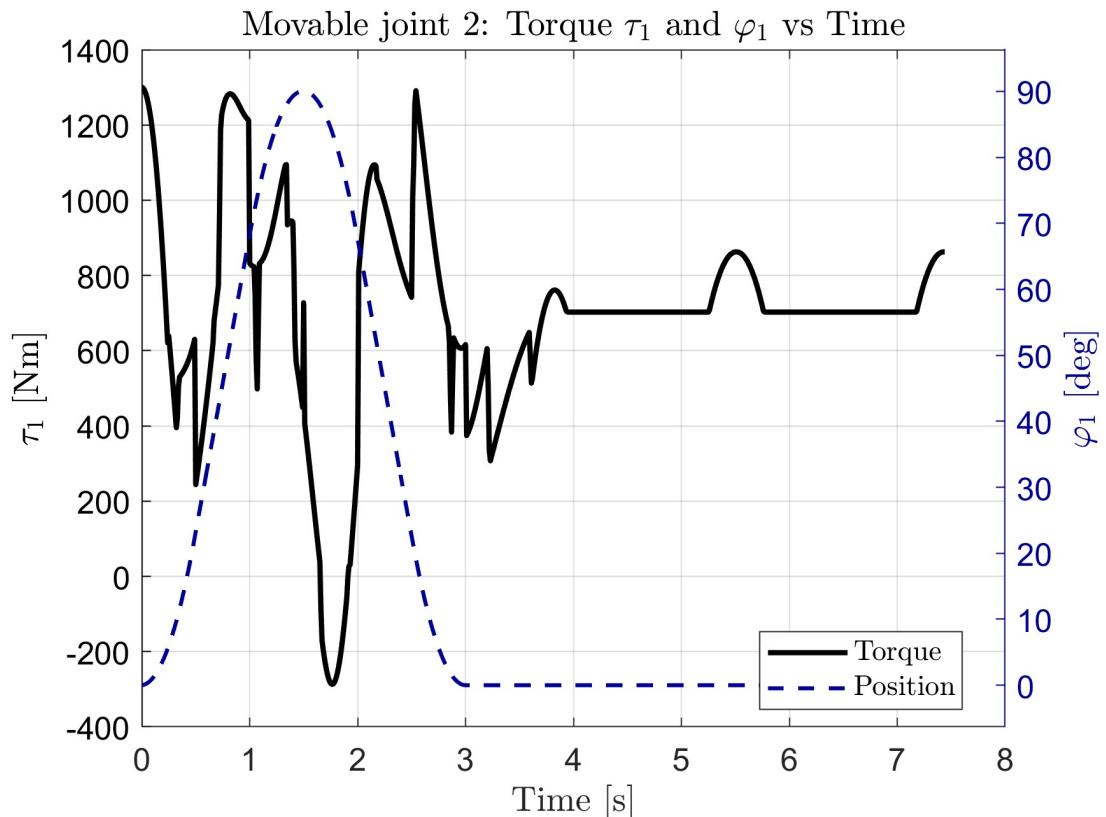


Abbildung 4.17: Generalisierten Gelenkmomente $\varphi_1 - \tau$

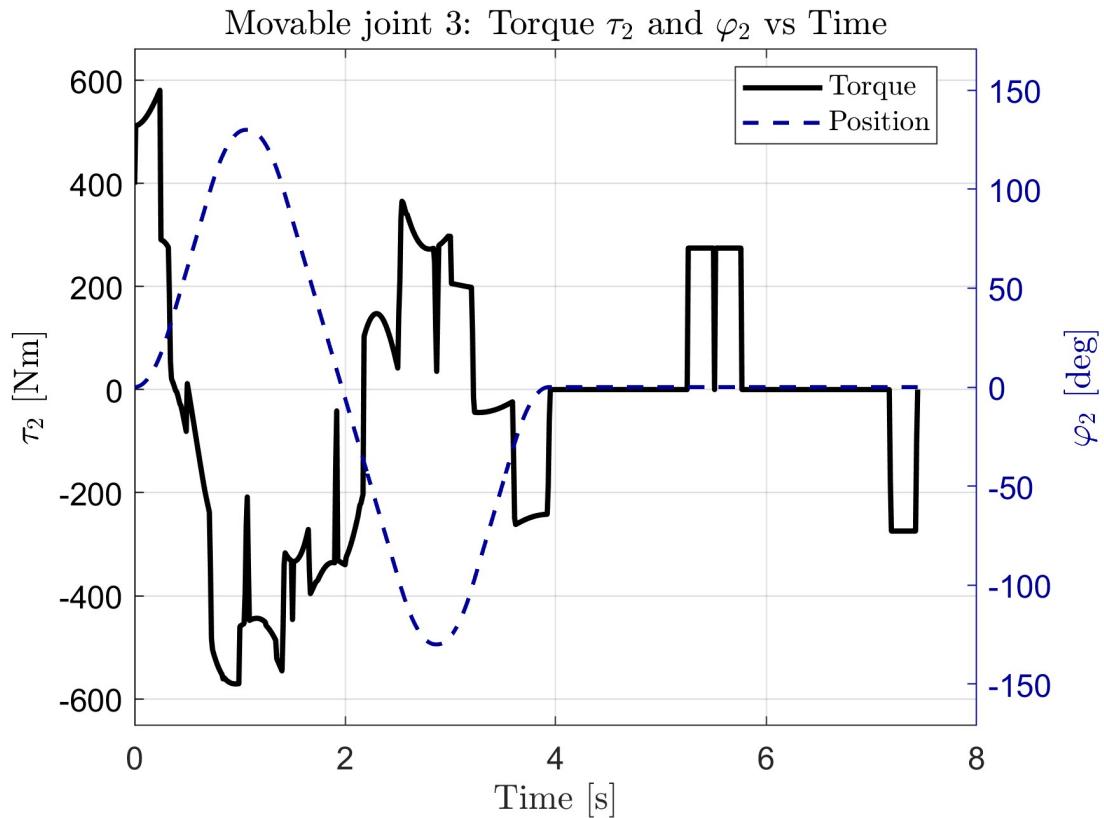


Abbildung 4.18: Generalisierten Gelenkmomente $\varphi_2 - \tau$

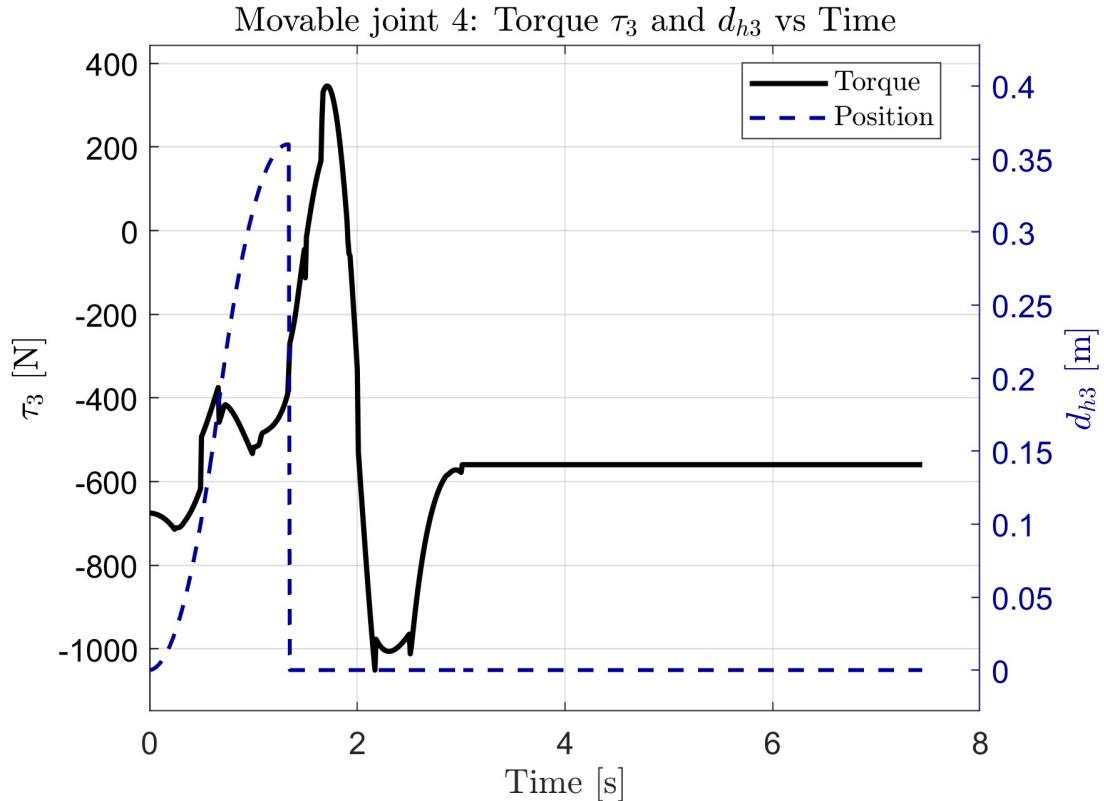


Abbildung 4.19: Generalisierten Gelenkkraft $d_{h3} - \tau$

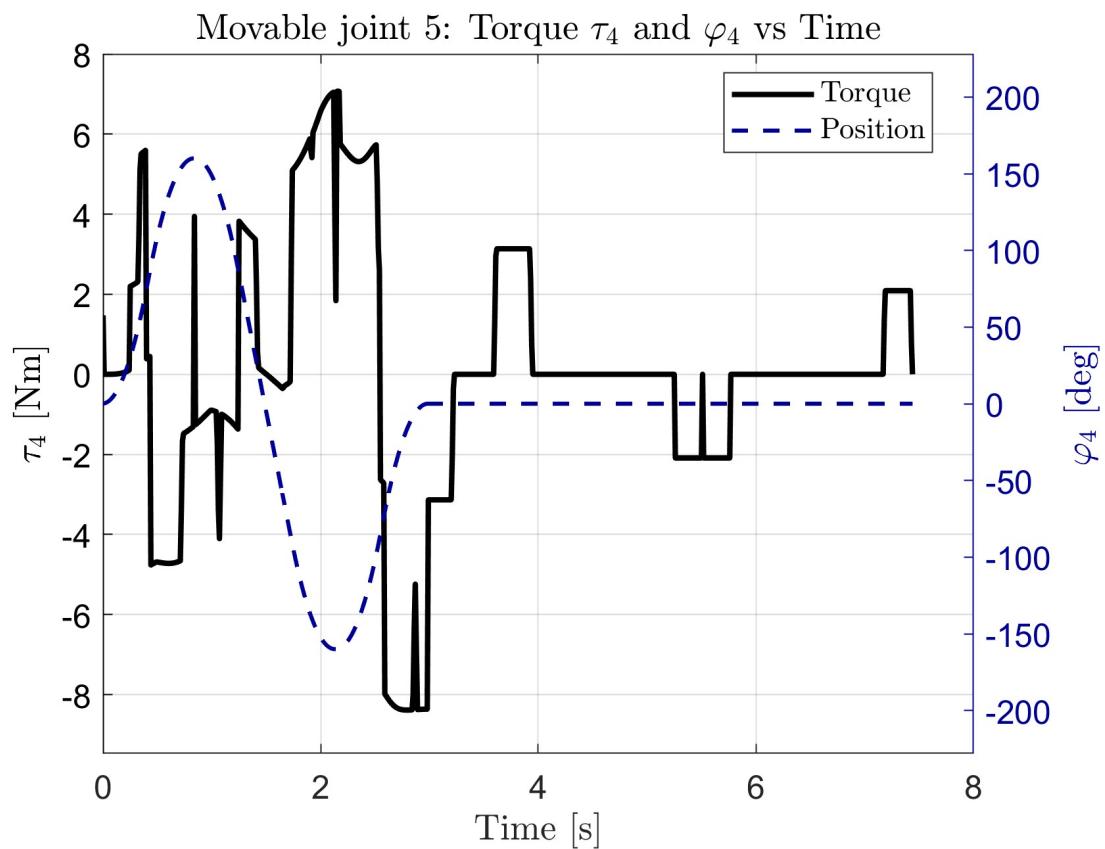


Abbildung 4.20: Generalisierten Gelenkmomente $\varphi_4 - \tau$



Bewegungsplanung (Video)

4.4 Python script für Bewegungsprofile

```

1
2 from ruckig import Ruckig, InputParameter, OutputParameter, Result
3 import numpy as np
4 from scipy.io import savemat
5 import matplotlib.pyplot as plt
6 import matplotlib as mpl
7 from matplotlib import rcParams
8 from pathlib import Path
9 import os
10
11 # ===== Professional figure styling =====
12 mpl.rcParams.update({
13     'font.size': 12,
14     'font.family': 'serif',
15     'axes.labelsize': 14,
16     'axes.titlesize': 16,
17     'xtick.labelsize': 12,
18     'ytick.labelsize': 12,
19     'legend.fontsize': 12,
20     'figure.titlesize': 18,
21     'grid.linestyle': '--',
22     'grid.alpha': 0.7,
23     'lines.linewidth': 2,
24 })
25
26 # ===== Joint parameters =====
27 axes = {
28     'phi0': {'s': np.deg2rad(150), 'v': 0.5 * np.pi, 'a': 2 * np.pi, 'r': 1200},
29     'phi1': {'s': np.deg2rad(90), 'v': 0.5 * np.pi, 'a': np.pi, 'r': 1000},
30     'phi2': {'s': np.deg2rad(130), 'v': np.pi, 'a': 3 * np.pi, 'r': 500},
31     'dh_3': {'s': 0.36, 'v': 100 / 60, 'a': 0.8, 'r': 500},
32     'phi4': {'s': np.deg2rad(160), 'v': 2 * np.pi, 'a': 5 * np.pi, 'r': 8000}
33 }
34
35 waypoints = {
36     'phi0': [0, axes['phi0']['s'], -axes['phi0']['s'], 0],
37     'phi1': [0, axes['phi1']['s'], 0],
38     'phi2': [0, axes['phi2']['s'], -axes['phi2']['s'], 0],
39     'dh_3': [0, axes['dh_3']['s']],
40     'phi4': [0, axes['phi4']['s'], -axes['phi4']['s'], 0],
41 }
42
43 # ===== Simulation configuration =====
44 time_step = 0.01
45 otg = Ruckig(1, time_step)
46 output_folder = Path("plots")
47 output_folder.mkdir(exist_ok=True)
48
49 # ===== Function to prepare InputParameter =====
50 def create_input(current, target, limits):
51     """
52         Creates an InputParameter object for Ruckig.
53
54     Args:
55         current (float): The current position.
56         target (float): The target position.
57         limits (dict): A dictionary containing 'v' (max velocity),
58                         'a' (max acceleration), and 'r' (max jerk).
59
60     Returns:
61         InputParameter: The configured Ruckig InputParameter object.

```

```
62 """
63     inp = InputParameter(1)
64     inp.current_position = [current]
65     inp.current_velocity = [0.0]
66     inp.current_acceleration = [0.0]
67     inp.target_position = [target]
68     inp.target_velocity = [0.0]
69     inp.target_acceleration = [0.0]
70     inp.max_velocity = [limits['v']]
71     inp.max_acceleration = [limits['a']]
72     inp.max_jerk = [limits['r']]
73     return inp
74
75 # ===== Function to plot profiles =====
76 def plot_trajectory(name, data, time_step, limits_dict):
77     """
78         Plots the position, velocity, acceleration, and jerk profiles for a given joint.
79
80     Args:
81         name (str): The name of the joint (e.g., 'phi0', 'dh_3').
82         data (dict): A dictionary containing lists of 'pos', 'vel', and 'acc' data.
83         time_step (float): The time step used in the simulation.
84         limits_dict (dict): A dictionary containing the limits ('s', 'v', 'a', 'r')
85                         for the current joint.
86     """
87     # Professional style configuration
88     limits_dict = axes[name]
89
90     plt.style.use('seaborn-v0_8-whitegrid')
91
92     # Configure fonts (optional, requires LaTeX installed)
93     rcParams['font.family'] = 'serif'
94     rcParams['font.serif'] = ['Times New Roman']
95     rcParams['mathtext.fontset'] = 'cm'
96
97     # Data
98     pos = np.array(data['pos'])
99     vel = np.array(data['vel'])
100    acc = np.array(data['acc'])
101    t = np.arange(len(pos)) * time_step
102    jerk = np.diff(acc, prepend=acc[0]) / time_step
103
104    # Professional color palette (gray/blue tones)
105    colors = {
106        'jerk': '#4E5B6C',    # Dark bluish gray
107        'acc': '#6C7B8B',    # Slate gray
108        'vel': '#3A5FCD',    # Medium blue
109        'pos': '#1E3F66',    # Dark blue
110        'grid': '#E0E0E0',    # Very light gray
111        'zero': '#606060'    # Medium gray
112    }
113
114    labels = [
115        ('Jerk', jerk, r'$\ddot{q}(t) \text{ [rad/s}^3\text{ or m/s}^3\text{]}$', colors['jerk']),
116        ('Acceleration', acc, r'$\dot{q}(t) \text{ [rad/s}^2\text{ or m/s}^2\text{]}$', colors['acc']),
117        ('Velocity', vel, r'$\dot{q}(t) \text{ [rad/s or m/s]}$', colors['vel']),
118        ('Position', pos, r'$q(t) \text{ [rad or m]}$', colors['pos']),
119    ]
120
121    fig = plt.figure(figsize=(8, 10))
122    plt.suptitle(f'Trajectory Analysis: {name}', y=0.98,
123                 fontsize=14, fontweight='bold')
```

```

124
125     for i, (title, data_i, ylabel, color) in enumerate(labels, 1):
126         ax = plt.subplot(4, 1, i)
127
128         # === Upper/lower limits based on type ===
129         lim_key = {'Jerk': 'r', 'Acceleration': 'a', 'Velocity': 'v', 'Position': 's'
130         }[title]
131         if lim_key in limits_dict:
132             limit = limits_dict[lim_key]
133             ax.axhline(+limit, color='gray', linestyle='--', linewidth=1.2, alpha
134             =0.8, label='Upper limit')
135             ax.axhline(-limit, color='#5B7DB1', linestyle='--', linewidth=1.2, alpha
136             =0.8, label='Lower limit')
137
138         # Add right axis in degrees only for position and if it's 'phi'
139         if title == 'Position' and 'phi' in name:
140             def rad2deg(y): return np.rad2deg(y)
141             def deg2rad(y): return np.deg2rad(y)
142             ax_deg = ax.secondary_yaxis('right', functions=(rad2deg, deg2rad))
143             ax_deg.set_ylabel(r'$q(t)$ [°], fontsize=10)
144             ax_deg.tick_params(axis='y', labelsize=9)
145
146             # Force visible limits and ticks in degrees
147             deg_min = np.floor(np.rad2deg(np.min(data_i)) / 30) * 30
148             deg_max = np.ceil(np.rad2deg(np.max(data_i)) / 30) * 30
149             ax_deg.set_yticks(np.arange(deg_min, deg_max + 1, 30))
150
151             print(f"Adding secondary axis for {name}")
152
153         # Thicker main line
154         ax.plot(t, data_i, label=title, color=color,
155                 linewidth=1.8, alpha=0.9)
156         ax.legend(loc='lower left', bbox_to_anchor=(0.4, 0.08), fontsize=6, frameon=
157         False, ncol=2)
158
159         # More subtle grid
160         ax.grid(True, color=colors['grid'], linestyle='--',
161                 linewidth=0.5, alpha=0.7)
162
163         # Axis style
164         for spine in ax.spines.values():
165             spine.set_edgecolor(colors['grid'])
166             spine.set_linewidth(0.8)
167
168         if i == 4:
169             ax.set_xlabel('Time [s]', fontsize=10)
170
171         # Adjust ticks
172         ax.tick_params(axis='both', which='both', labelsize=9,
173                         direction='in', width=0.8)
174
175         # Add title to each subplot (optional)
176         ax.text(0.98, 0.92, title, transform=ax.transAxes,
177                 fontsize=10, ha='right', va='top',
178                 bbox=dict(facecolor='white', alpha=0.7,
179                             edgecolor='none', pad=2))
180
181         plt.tight_layout(rect=[0, 0, 1, 0.96])
182

```

```
183     # Save in high quality
184     plt.savefig(output_folder / f"{name}_trajectory.png",
185                 dpi=600, bbox_inches='tight')
186     plt.savefig(output_folder / f"{name}_trajectory.pdf",
187                 bbox_inches='tight')
188     plt.close()
189
190 def plot_trajectory(name, data, time_step, limits_dict):
191     """
192         Plots the position, velocity, acceleration, and jerk profiles for a given joint.
193     """
194     # Professional configuration
195     plt.style.use('default')
196
197     # Configure fonts and text
198     rcParams['font.family'] = 'serif'
199     rcParams['font.serif'] = ['Times New Roman']
200     rcParams['mathtext.fontset'] = 'cm'
201     rcParams['axes.titlesize'] = 14
202     rcParams['axes.labelsize'] = 12
203
204     # Prepare data
205     pos = np.array(data['pos'])
206     vel = np.array(data['vel'])
207     acc = np.array(data['acc'])
208     t = np.arange(len(pos)) * time_step
209     jerk = np.diff(acc, prepend=acc[0]) / time_step
210
211     # Dictionary to convert names to LaTeX
212     name_to_latex = {
213         'phi0': r'$\varphi_0$',
214         'phi1': r'$\varphi_1$',
215         'phi2': r'$\varphi_2$',
216         'dh_3': r'$d_{h3}$',
217         'phi4': r'$\varphi_4$',
218         # Add more mappings as needed
219     }
220
221     # Get LaTeX representation or use default
222     latex_name = name_to_latex.get(name, f'${name}$')
223
224     # Professional subplot titles in English
225     subplot_titles = {
226         'Position': f'Position Profile: {latex_name}',
227         'Velocity': f'Velocity Profile: {latex_name}',
228         'Acceleration': f'Acceleration Profile: {latex_name}',
229         'Jerk': f'Jerk Profile: {latex_name}'
230     }
231
232     # Professional color scheme (blues, grays, dark colors)
233     colors = {
234         'jerk': '#4C4C4C',      # Dark gray
235         'acc': '#6D6D6D',       # Medium gray
236         'vel': '#005073',       # Dark blue
237         'pos': '#0080A3',       # Medium blue
238         'limit': '#A2142F',       # Dark red for limits
239         'grid': '#E0E0E0',       # Light gray for grid
240         'zero': '#808080'        # Medium gray for zero line
241     }
242
243     # Create figure
244     fig = plt.figure(figsize=(8, 10), dpi=100)
245     plt.suptitle(f'Trajectory Analysis: {latex_name}', y=0.98,
```

```

246         fontsize=14, fontweight='bold')
247
248     # Plot configurations for each subplot
249     plot_configs = [
250         ('Jerk', jerk, r'$\dddot{q}(t) [rad/s^3] or m/s^3$', colors['jerk']),
251         ('Acceleration', acc, r'$\ddot{q}(t) [rad/s^2] or m/s^2$', colors['acc'])
252     ],
253         ('Velocity', vel, r'$\dot{q}(t) [rad/s or m/s]', colors['vel']),
254         ('Position', pos, r'$q(t) [rad or m]', colors['pos'])
255     ]
256
257     for i, (title, data_i, ylabel, color) in enumerate(plot_configs, 1):
258         ax = plt.subplot(4, 1, i)
259
260         # Configure limits
261         lim_key = {'Jerk': 'r', 'Acceleration': 'a', 'Velocity': 'v', 'Position': 's'}[title]
262         if lim_key in limits_dict:
263             limit = limits_dict[lim_key]
264             ax.axhline(+limit, color=colors['limit'], linestyle='--',
265                         linewidth=1.5, alpha=0.8, label='Limit')
266             ax.axhline(-limit, color=colors['limit'], linestyle='--',
267                         linewidth=1.5, alpha=0.8)
268
269         # Plot data
270         ax.plot(t, data_i, color=color, linewidth=2.0, alpha=0.9)
271
272         # Configure secondary axis for angles
273         if title == 'Position' and 'phi' in name:
274             ax_deg = ax.secondary_yaxis('right', functions=(np.rad2deg, np.deg2rad))
275             ax_deg.set_ylabel(r'$q(t) [^\circ]', fontsize=10)
276
277         # Styling
278         ax.set_ylabel(ylabel, fontsize=10)
279         ax.axhline(0, color=colors['zero'], linestyle=':', linewidth=1.0)
280         ax.grid(True, color=colors['grid'], linestyle='-', linewidth=0.5)
281
282         # Subplot title
283         ax.set_title(subplot_titles[title], fontsize=11, pad=10,
284                     bbox=dict(facecolor='white', alpha=0.8, edgecolor='none'))
285
286         plt.tight_layout(rect=[0, 0, 1, 0.96])
287
288         # Save in high quality
289         output_folder.mkdir(exist_ok=True)
290         plt.savefig(output_folder / f"{name}_trajectory.png",
291                     dpi=600, bbox_inches='tight', facecolor='white')
292         plt.savefig(output_folder / f"{name}_trajectory.pdf",
293                     bbox_inches='tight', facecolor='white')
294         plt.close()
295
296     # ===== Joint trajectory simulation =====
297     trajectories = {name: {'pos': [], 'vel': [], 'acc': []} for name in axes}
298     timestamps = {name: [] for name in axes}
299
300     for name, points in waypoints.items():
301         current = 0.0
302         for target in points[1:]:
303             inp = create_input(current, target, axes[name])
304             out = OutputParameter(1)
305             time = 0.0
306             while True:
307                 res = otg.update(inp, out)

```

```
307     out.pass_to_input(inp)
308
309     trajectories[name]['pos'].append(out.new_position[0])
310     trajectories[name]['vel'].append(out.new_velocity[0])
311     trajectories[name]['acc'].append(out.new_acceleration[0])
312     timestamps[name].append(time)
313     time += time_step
314
315     if res == Result.Finished:
316         break
317     current = target
318     plot_trajectory(name, trajectories[name], time_step, axes[name])
319
320 # ===== Save to .mat file =====
321 savemat("ruckig_trajectories.mat", {'trajectories': trajectories})
322 print("Trajectories exported to 'ruckig_trajectories.mat' and plots saved in /plots")
)
```

Listing 4.1: Python script für Lagesollprofile

4.5 Numerische Auswertung der Lagrange-Gleichungen mit MATLAB

Der folgende MATLAB-Code dient der numerischen Auswertung der Lagrange-Gleichungen. Dabei wird nicht die symbolische Form, sondern eine direkt eingesetzte numerische Berechnung verwendet. Grundlage der Berechnung ist der vollständige Jacobian sowie die Profile der Gelenkgeschwindigkeiten und -beschleunigungen. Als Ergebnis wird das Momentenvektor $\tau(t)$ in Abhängigkeit von den Bewegungsprofilen bestimmt.

```

1
2 clc;
3 clear;
4
5 % Create the robot with correct configuration
6 robot = rigidBodyTree('DataFormat','row','MaxNumBodies',8);
7
8 % =====
9 % UNIT CONVERSION (mm to m) and parameters
10 % =====
11 Hs = 200/1000; H0 = 100/1000; L0 = 800/1000;
12 L1 = 400/1000; D1 = 110/1000; L2 = 500/1000;
13 L3 = 630/1000; DH3 = 360/1000; L4 = 600/1000;
14 L5 = 50/1000; H3 = 1060/1000; DA2 = 100/1000;
15 KH3 = 80/1000;
16
17 % Material properties (steel)
18 densidad_acero = 7750; % kg/m^3
19
20 % Custom geometry for each body (external/internal diameters/sides)
21 % Format: [length, type, dimension1, dimension2, internal_dimension1,
22 %           internal_dimension2]
22 % type: 1 = hollow cylinder, 2 = hollow square
23 % Extended format: [length, type, dim1, dim2, int1, int2, axis]
24 % axis = 1 (X), 2 (Y), 3 (Z)
25 geometrias = {
26     [Hs, 1, 0.3/2, 0.3/2, 0.2/2, 0.2/2, 3]; % body1: vertical Z
27     [L0, 3, 0.2/2, 0.2/2, 0.15/2, 0.15/2, 3]; % body2: vertical Z
28     [L1, 2, 0.1, 0.1, 0.08, 0.08, 1]; % body3: horizontal X
29     [D1, 4, 0.1, 0.1, 0.08, 0.08, 3]; % body4: vertical Z
30     [L2, 1, 0.1/2, 0.1/2, 0.08/2, 0.08/2, 1]; % body5: horizontal X
31     [(DA2/2)+L3+KH3, 1, 0.1/2, 0.1/2, 0.08/2, 0.08/2, 3]; % body6: vertical Z
32     [DH3, 1, 0.14/2, 0.14/2, 0.08/2, 0.08/2, 3];
33     [(L4/2)+L5, 1, 0.14/2, 0.14/2, 0.08/2, 0.08/2, 3]; % body8: vertical Z
34 };
35
36
37 % DH table: [a L alpha d theta phi] (now in meters)
38 dhparams = [
39     0, 0, Hs-H0, 0; % 1. base to Z1
40     0, pi/2, L0+H0, 0; % 2. phi0 (revolute)
41     L1, -pi/2, 0, 0; % 3. phi1 (revolute)
42     0, 0, D1, 0; % 4. fixed
43     L2, pi, 0, 0; % 5. phi2 (revolute)
44     0, 0, (DA2/2)+L3+KH3, 0; % 6. fixed
45     0, 0, DH3, 0; % 7. prismatic
46     0, 0, (L4/2)+L5, 0 % 8. phi4 (revolute)
47 ];
48
49 % Correct joint types configuration
50 joint_types = {'fixed', 'revolute', 'revolute', 'fixed', 'revolute', 'fixed', 'prismatic', 'revolute'};
51
52 % Active (movable) joints:
53 activeJointBodies = [2, 3, 5, 7, 8]; % Indices of revolute/prismatic joints

```

```
54 % =====
55 % Robot construction with realistic dynamic properties
56 % =====
57 % =====
58 for i = 1:8
59     % Create body and joint
60     bodies{i} = rigidBody(['body' num2str(i)]);
61     joints{i} = rigidBodyJoint(['jnt' num2str(i)], joint_types{i});
62
63     % Configure DH transformation
64     setFixedTransform(joints{i}, dhparams(i,:), 'dh');
65
66     % Configure axis and limits based on joint type
67     if strcmp(joint_types{i}, 'prismatic')
68         joints{i}.JointAxis = [0 0 1]; % Movement in Z
69         joints{i}.PositionLimits = [-DH3, 0];
70
71     elseif strcmp(joint_types{i}, 'revolute')
72         % Assign limits based on the physical joint
73         switch i
74             case 3 % phi0
75                 joints{i}.JointAxis = [0 0 1]; % Z-axis
76                 joints{i}.PositionLimits = deg2rad([-150, 150]);
77             case 4 % phi1 - IMPORTANT: Now rotates around Y
78                 joints{i}.JointAxis = [0 1 0]; % Y-axis
79                 joints{i}.PositionLimits = deg2rad([0, 90]);
80             case 5 % phi2
81                 joints{i}.JointAxis = [0 0 1]; % Z-axis
82                 joints{i}.PositionLimits = deg2rad([-153, 153]);
83             case 8 % phi4
84                 joints{i}.JointAxis = [0 0 1]; % Z-axis
85                 joints{i}.PositionLimits = deg2rad([-180, 180]);
86         end
87     end
88
89 % =====
90 % DYNAMIC PROPERTIES BASED ON CUSTOM GEOMETRY
91 % =====
92 if ismember(joint_types{i}, {'fixed', 'revolute', 'prismatic'})
93     % Get geometry parameters for this body
94     geom = geometrias{i};
95     longitud = geom(1);
96     tipo = geom(2);
97     dim1 = geom(3);
98     dim2 = geom(4); otro lado
99     dim_int1 = geom(5);
100    dim_int2 = geom(6); o
101
102    % Mass calculation based on geometry type
103    if tipo == 1
104        radio_externo = dim1;
105        radio_interno = dim_int1;
106        area_seccion = pi*(radio_externo^2 - radio_interno^2);
107        volumen = area_seccion * longitud;
108
109    elseif tipo ==
110        lado_externo = dim1;
111        lado_interno = dim_int1;
112        area_seccion = lado_externo^2 - lado_interno^2;
113        volumen = area_seccion * longitud;
114
115    elseif tipo == 3
116        volumen = 0.012967447276; CAD
```

```

117
118 elseif tipo == 4
119     volumen = 0.000740789794;
120
121 else
122     error('type of body not defined %d', i);
123 end
124
125
126
127 masa = densidad_acero * volumen;
128 masses(i) = masa;
129
130 % Center of mass (at link center)
131 % direction (1=X, 2=Y, 3=Z)
132 eje_principal = geom(7);
133 centro_masa = [0, 0, 0]; % Valor por defecto
134
135 % === Si el tipo es 3 o 4 (CAD), se usa valor corregido ===
136 if tipo == 3
137
138     distancia = 0.470732;
139     centro_masa(eje_principal) = distancia;
140 elseif tipo == 4
141     distancia = 0.111934;
142     centro_masa(eje_principal) = distancia;
143 else
144
145     switch eje_principal
146         case 1
147             centro_masa = [longitud/2, 0, 0];
148         case 2
149             centro_masa = [0, longitud/2, 0];
150         case 3
151             centro_masa = [0, 0, longitud/2];
152         otherwise
153             error('Axis not defined %d', i);
154     end
155 end
156
157
158 % Inertia tensor calculation based on geometry type
159 if tipo == 1
160     Ixx = (1/2)*masa*(radio_externo^2 + radio_interno^2);
161     Iyy = (1/12)*masa*(3*(radio_externo^2 + radio_interno^2) + longitud^2);
162     Izz = Iyy;
163 elseif tipo==2
164
165     a_ext = dim1; b_ext = dim2; % Dimensiones externas
166     a_int = dim_int1; b_int = dim_int2; % Dimensiones internas
167
168     Ixx = (1/12)*masa*(a_ext^2 + b_ext^2 + a_int^2 + b_int^2);
169     Iyy = (1/12)*masa*(3*(b_ext^2 + b_int^2) + longitud^2);
170     Izz = (1/12)*masa*(3*(a_ext^2 + a_int^2) + longitud^2);
171 elseif tipo == 3 % tipo CAD - cuerpo largo
172     Ixx = 7749265.619e-6;
173     Iyy = 776332.607e-6;
174     Izz = 7749265.619e-6;
175
176 elseif tipo == 4
177     Ixx = 36063.992e-6;
178     Iyy = 18013.291e-6;
179     Izz = 36063.991e-6;

```

```
180
181     end
182
183     % Assign properties
184     bodies{i}.Mass = masa;
185     bodies{i}.CenterOfMass = centro_masa;
186     bodies{i}.Inertia = [Ix, Iy, Iz, 0, 0, 0];
187 end
188
189 bodies{i}.Joint = joints{i};
190
191 % Add to rigid body tree
192 if i == 1
193     addBody(robot, bodies{i}, 'base');
194 else
195     addBody(robot, bodies{i}, bodies{i-1}.Name);
196 end
197 end
198
199 % =====
200 % GRAVITY CONFIGURATION (IMPORTANT!)
201 % =====
202 robot.Gravity = [0, 0, -9.81];
203
204 %% End Effector configuration (body8 as end effector)
205 endEffectorFrame = 'body8';
206
207 % Show robot details
208 showdetails(robot);
209
210 %% Enhanced visualization
211 figure('Name','Robot with End Effector in body8');
212 show(robot, 'Frames', 'on', 'PreservePlot', false);
213 hold on;
214
215 % Mark end effector position
216 config = homeConfiguration(robot);
217 eePos = getTransform(robot, config, endEffectorFrame);
218 plot3(eePos(1,4), eePos(2,4), eePos(3,4), 'ro', 'MarkerSize', 10, 'MarkerFaceColor',
219      'r');
220
221 title('Robot with End Effector in body8');
222 xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]'); % Now in meters
223 view(3);
224 axis equal;
225 grid on;
226
227 %% Interactive interface
228 figure("Name","Interactive GUI test");
229 gui = interactiveRigidBodyTree(robot, 'MarkerScaleFactor', 0.5, 'MarkerBodyName',
230      endEffectorFrame);
231
232 %% Dynamic calculations
233 % 1. Calculate Jacobian
234 config = homeConfiguration(robot);
235 jacobian = geometricJacobian(robot, config, endEffectorFrame);
236
237 fprintf('\n--- Jacobian in home configuration ---\n');
238 disp(jacobian);
239
240 load('ruckig_trajectories.mat');
```

```

241 time_phi0 = trajectories.phi0.time;
242 time_phi1 = trajectories.phi1.time;
243 time_phi2 = trajectories.phi2.time;
244 time_dh3 = trajectories.dh3.time;
245 time_phi4 = trajectories.phi4.time;
246
247 phi0_values = trajectories.phi0.position;
248 phi1_values = trajectories.phi1.position;
249 phi2_values = trajectories.phi2.position;
250 dh3_values = trajectories.dh3.position;
251 phi4_values = trajectories.phi4.position;
252
253 dphi0_values = trajectories.phi0.velocity;
254 dphi1_values = trajectories.phi1.velocity;
255 dphi2_values = trajectories.phi2.velocity;
256 ddh3_values = trajectories.dh3.velocity;
257 dphi4_values = trajectories.phi4.velocity;
258
259 ddphi0_values = trajectories.phi0.acceleration;
260 ddphi1_values = trajectories.phi1.acceleration;
261 ddphi2_values = trajectories.phi2.acceleration;
262 dddh3_values = trajectories.dh3.acceleration;
263 ddphi4_values = trajectories.phi4.acceleration;
264
265 n_points = length(phi0_values);
266
267
268 tau_eval = zeros(5, n_points);
269
270
271 phi0_values = phi0_values(:)';
272 phi1_values = phi1_values(:)';
273 phi2_values = phi2_values(:)';
274 dh3_values = dh3_values(:)';
275 phi4_values = phi4_values(:)';
276
277 dphi0_values = dphi0_values(:)';
278 dphi1_values = dphi1_values(:)';
279 dphi2_values = dphi2_values(:)';
280 ddh3_values = ddh3_values(:)';
281 dphi4_values = dphi4_values(:)';
282
283 ddphi0_values = ddphi0_values(:)';
284 ddphi1_values = ddphi1_values(:)';
285 ddphi2_values = ddphi2_values(:)';
286 dddh3_values = dddh3_values(:)';
287 ddphi4_values = ddphi4_values(:)';
288
289 whos phi0 phi1 phi2 dh3 phi4
290
291 fprintf(' \n===== ESTADISTICAS ORIGINALES DE LAS TRAYECTORIAS =====\n');
292
293 % PHI0
294 print_stats(trajectories.phi0.position,      'phi0.position');
295 print_stats(trajectories.phi0.velocity,       'phi0.velocity');
296 print_stats(trajectories.phi0.acceleration,  'phi0.acceleration');
297 print_stats(trajectories.phi0.jerk,          'phi0.jerk');
298
299 % PHI1
300 print_stats(trajectories.phi1.position,       'phi1.position');
301 print_stats(trajectories.phi1.velocity,        'phi1.velocity');
302 print_stats(trajectories.phi1.acceleration,   'phi1.acceleration');
303 print_stats(trajectories.phi1.jerk,           'phi1.jerk');

```

```
304 % PHI2
305 print_stats(trajectories.phi2.position,      'phi2.position');
306 print_stats(trajectories.phi2.velocity,       'phi2.velocity');
307 print_stats(trajectories.phi2.acceleration,  'phi2.acceleration');
308 print_stats(trajectories.phi2.jerk,           'phi2.jerk');
309
310
311 % DH3
312 print_stats(trajectories.dh3.position,        'dh3.position');
313 print_stats(trajectories.dh3.velocity,         'dh3.velocity');
314 print_stats(trajectories.dh3.acceleration,    'dh3.acceleration');
315 print_stats(trajectories.dh3.jerk,             'dh3.jerk');
316
317 % PHI4
318 print_stats(trajectories.phi4.position,        'phi4.position');
319 print_stats(trajectories.phi4.velocity,         'phi4.velocity');
320 print_stats(trajectories.phi4.acceleration,    'phi4.acceleration');
321 print_stats(trajectories.phi4.jerk,             'phi4.jerk');
322
323
324 fprintf('\n== shape position ===\n');
325 disp(['phi0: ', mat2str(size(phi0_values))]);
326 disp(['phi1: ', mat2str(size(phi1_values))]);
327 disp(['phi2: ', mat2str(size(phi2_values))]);
328 disp(['dh3: ', mat2str(size(dh3_values))]);
329 disp(['phi4: ', mat2str(size(phi4_values))]);
330
331
332 final_time = max([
333     trajectories.phi0.time(end);
334     trajectories.phi1.time(end);
335     trajectories.phi2.time(end);
336     trajectories.dh3.time(end);
337     trajectories.phi4.time(end)
338 ]);
339
340
341 n_points = max([length(trajectories.phi0.time); length(trajectories.phi1.time);
342                  length(trajectories.phi2.time); length(trajectories.dh3.time);
343                  length(trajectories.phi4.time)]);
344
345 common_time = linspace(0, final_time, n_points);
346
347 pad_with_zeros = @(data, target_len) [data, zeros(1, target_len - length(data))];
348
349 phi0_values      = pad_with_zeros(trajectories.phi0.position, n_points);
350 dphi0_values     = pad_with_zeros(trajectories.phi0.velocity, n_points);
351 ddphi0_values   = pad_with_zeros(trajectories.phi0.acceleration, n_points);
352
353 phi1_values      = pad_with_zeros(trajectories.phi1.position, n_points);
354 dphi1_values     = pad_with_zeros(trajectories.phi1.velocity, n_points);
355 ddphi1_values   = pad_with_zeros(trajectories.phi1.acceleration, n_points);
356
357 phi2_values      = pad_with_zeros(trajectories.phi2.position, n_points);
358 dphi2_values     = pad_with_zeros(trajectories.phi2.velocity, n_points);
359 ddphi2_values   = pad_with_zeros(trajectories.phi2.acceleration, n_points);
360
361 dh3_values       = pad_with_zeros(trajectories.dh3.position, n_points);
362 ddh3_values      = pad_with_zeros(trajectories.dh3.velocity, n_points);
363 dddh3_values     = pad_with_zeros(trajectories.dh3.acceleration, n_points);
364
365 phi4_values      = pad_with_zeros(trajectories.phi4.position, n_points);
366 dphi4_values     = pad_with_zeros(trajectories.phi4.velocity, n_points);
```

```

367 ddphi4_values = pad_with_zeros(trajectories.phi4.acceleration, n_points);
368
369
370 % PHI0
371 print_stats(phi0_values,      'phi0.position (interp)');
372 print_stats(dphi0_values,     'phi0.velocity (interp)');
373 print_stats(ddphi0_values,   'phi0.acceleration (interp)');
374
375 % PHI1
376 print_stats(phi1_values,      'phi1.position (interp)');
377 print_stats(dphi1_values,     'phi1.velocity (interp)');
378 print_stats(ddphi1_values,   'phi1.acceleration (interp)');
379
380 % PHI2
381 print_stats(phi2_values,      'phi2.position (interp)');
382 print_stats(dphi2_values,     'phi2.velocity (interp)');
383 print_stats(ddphi2_values,   'phi2.acceleration (interp)');
384
385 % DH3
386 print_stats(dh3_values,       'dh3.position (interp)');
387 print_stats(ddh3_values,      'dh3.velocity (interp)');
388 print_stats(dddh3_values,    'dh3.acceleration (interp)');
389
390 % PHI4
391 print_stats(phi4_values,      'phi4.position (interp)');
392 print_stats(dphi4_values,     'phi4.velocity (interp)');
393 print_stats(ddphi4_values,   'phi4.acceleration (interp)');
394
395 time = common_time;
396 n_points = length(time);
397
398 fprintf('\n==== Shape de variables de posici on ====\n');
399 disp(['phi0: ', mat2str(size(phi0_values))]);
400 disp(['phi1: ', mat2str(size(phi1_values))]);
401 disp(['phi2: ', mat2str(size(phi2_values))]);
402 disp(['dh3: ', mat2str(size(dh3_values))]);
403 disp(['phi4: ', mat2str(size(phi4_values))]);
404
405
406
407 tau_eval = zeros(5, n_points);
408
409
410 for i = 1:n_points
411     % Configuraci on actual de las articulaciones
412     q_current = [phi0_values(i), phi1_values(i), phi2_values(i), dh3_values(i),
413                  phi4_values(i)];
414     qd_current = [dphi0_values(i), dphi1_values(i), dphi2_values(i), ddh3_values(i),
415                   dphi4_values(i)];
416     qdd_current = [ddphi0_values(i), ddphi1_values(i), ddphi2_values(i),
417                    dddh3_values(i), ddphi4_values(i)];
418
419     tau_eval(:, i) = inverseDynamics(robot, q_current, qd_current, qdd_current);
420
421     if mod(i, 100) == 0 || i == n_points
422         fprintf('Paso %d/%d - Tiempo %.3f s\n', i, n_points, time(i));
423         fprintf('Tau evaluated');
424         disp(tau_eval(:, i));
425     end
426 end
427
428 fprintf('\nGenerating Torque vs Time and Position vs Time plots...\n');

```

```
427
428 joint_names = {'$\varphi_0$', '$\varphi_1$', '$\varphi_2$', '$d_{h3}$', '$\varphi_4$'};
429 units_tau = {'Nm', 'Nm', 'Nm', 'N', 'Nm'};
430 positions = {phi0_values, phil_values, phi2_values, dh3_values, phi4_values};
431 is_angle = [true, true, true, false, true]; % dh3 is linear
432
433 % Define colors
434 color_torque = [0 0 0]; % black
435 color_position = [0 0 0.6]; % dark blue
436
437 for i = 1:5
438     figure('Name', ['Torque and Position - Joint ', joint_names{i}], 'Color', 'w');
439     file_names = {'tau_phi0.png', 'tau_phil.png', 'tau_phi2.png', 'tau_dh3.png',
440                 'tau_phi4.png'};
441     % LEFT Y-AXIS: Torque
442     yyaxis left
443     ax = gca;
444     ax.YColor = color_torque; % Match y-axis color
445     plot(time, tau_eval(i, :), '--', 'LineWidth', 2, 'Color', color_torque);
446     ylabel(sprintf('$\tau_{%d} [%s]', i-1, units_tau{i}), ...
447             'FontSize', 12, 'Interpreter', 'latex');
448     ylim padded
449     grid on;
450
451     % RIGHT Y-AXIS: Position
452     yyaxis right
453     ax.YColor = color_position; % Match y-axis color
454     if is_angle(i)
455         plot(time, rad2deg(positions{i}), '--', 'LineWidth', 1.5, 'Color',
456              color_position);
457         ylabel(sprintf('%s [deg]', joint_names{i}), ...
458               'FontSize', 12, 'Interpreter', 'latex');
459     else
460         plot(time, positions{i}, '--', 'LineWidth', 1.5, 'Color', color_position);
461         ylabel(sprintf('%s [m]', joint_names{i}), ...
462               'FontSize', 12, 'Interpreter', 'latex');
463     end
464     ylim padded
465
466     % X-Axis
467     xlabel('Time [s]', 'FontSize', 12, 'Interpreter', 'latex');
468
469     % Title
470     title(sprintf('Movable joint %d: Torque $\tau_{%d}$ and %s vs Time', ...
471                 i, i-1, joint_names{i}), ...
472             'FontSize', 14, 'Interpreter', 'latex');
473
474     % Legend
475     legend({'Torque', 'Position'}, 'Location', 'best', 'Interpreter', 'latex');
476
477     set(gca, 'FontSize', 12);
478     grid on;
479     box on;
480
481     jpg_name = strrep(file_names{i}, '.png', '.jpg');
482     exportgraphics(gcf, jpg_name, 'Resolution', 300);
483
484 %% === Funci on auxiliar para estad isticas ===
485 function print_stats(signal, name)
486     fprintf('\n--- Estad isticas para %s ---\n', name);
```

```
487 fprintf(' minimum: %.6f\n', min(signal));  
488 fprintf(' max: %.6f\n', max(signal));  
489 fprintf(' mean: %.6f\n', mean(signal));  
490 fprintf(' Desv. Std: %.6f\n', std(signal));  
491 end  
492  
493  
494 %% === 8. Calculation and visualization of total robot mass ===  
495 total_mass = sum(masses);  
496 fprintf('\n==== Total Robot Mass ====\n');  
497 fprintf('Total mass: %.3f kg\n', total_mass);  
498 fprintf('Breakdown by links:\n');  
499 for i = 1:numel(masses)  
500     fprintf(' Link %d: %.3f kg\n', i, masses(i));  
501 end
```

4.6 Symbolische Auswertung der Lagrange-Gleichungen mit MATLAB

Der folgende MATLAB-Code dient der symbolischen Auswertung der Lagrange-Gleichungen. Dabei werden die Ausdrücke für die Massenmatrix $\mathbf{M}(\mathbf{q})$, die Coriolis- und Zentrifugalkräfte $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ sowie der Gravitationsvektor $\mathbf{G}(\mathbf{q})$ vollständig symbolisch hergeleitet. Grundlage der Berechnung sind die symbolisch bestimmten Jacobi-Matrizen sowie die generalisierten Koordinaten \mathbf{q} und deren Ableitungen. Als Ergebnis wird der Momentenvektor $\boldsymbol{\tau}(t)$ in symbolischer Form als Funktion der Gelenkbewegungen bestimmt.

```
1
2
3 % Clear workspace and command window, disable all warnings
4 clc;
5 clear;
6 % warning('off','all');
7
8
9 %% 1. Symbolic Parameters and Jacobian Calculation (Original Code)
10
11 % Define symbolic variables for geometric parameters
12 syms Hs H0 L0 L1 D1 L2 DA2 L3 H3 L5 L4 KH3 real;
13 % Define symbolic variables for joint angles/displacements (all in radians for
14 % angles)
14 syms phi0 phi1 phi2 phi3 phi4 phi5 phi6 dh3 real;
15 % Define symbolic variables for joint velocities
16 syms dphi0 dphi1 dphi2 ddh3 dphi4 real;
17 % Define symbolic variables for joint accelerations
18 syms ddphi0 ddphi1 ddphi2 dddh3 ddphi4 real;
19 % Define symbolic variables for specific link lengths/displacements
20 syms dl1 dl2 dl3 dl4 real;
21 % Unused symbolic variables (can be removed if not needed elsewhere)
22 syms d2 dh d6 real;
23 % Unused symbolic variables for pose (can be removed if not needed elsewhere)
24 syms nx ny nz ux uy uz ax ay az px py pz real;
25 % Unused symbolic variables (can be removed if not needed elsewhere)
26 syms a1 a2 a3 real;
27
28 % Define pi as a symbolic constant
29 pi = sym(pi);
30
31 % Define target variables (active joints) for dynamics calculation
32 target_vars = [phi0, phi1, phi2, dh3, phi4];
33 target_var_names = {'phi0', 'phi1', 'phi2', 'dh3', 'phi4'};
34
35 % Denavit-Hartenberg (DH) parameters: alpha, L, D, phi
36 % alpha: Twist angle
37 % L: Link length
38 % D: Link offset
39 % phi: Joint angle (theta in standard DH)
40 alpha = [ 0, pi/2, -pi/2, 0, pi, 0, 0, 0];
41 L = [ 0, 0, L1, 0, L2, 0, 0, 0];
42 D = [dl1, dl2, 0, D1, 0, dl3, dh3, dl4];
43 phi = [0, phi0, phi1, 0, phi2, 0, 0, phi4]; % All angles in radians
44
45 % Number of joints
46 n = length(alpha);
47
48 % Anonymous function for homogeneous transformation matrix (standard DH parameters)
49 getA = @(alpha, L, D, phi) ...
50     [ cos(phi), -cos(alpha)*sin(phi), sin(alpha)*sin(phi), L*cos(phi);
51      sin(phi), cos(alpha)*cos(phi), -sin(alpha)*cos(phi), L*sin(phi);
52      0, sin(alpha), cos(alpha), D;
53      0, 0, 0, 1];
```

```

54 % Symbolic transformation matrices for each link
55 A = sym(zeros(4,4,n)); % 3D symbolic array
56 for i = 1:n
57     A_i = getA(alpha(i), L(i), D(i), phi(i));
58     A_i_simple = simplify(A_i, 'Steps', 100); % Simplify expression
59     A_i_clean = vpa(A_i_simple, 6); % Round to 6 decimal places for cleaner output
60     A(:,:,i) = A_i_clean;
61 end
62
63
64 % A_eval will store transformation matrices with numerical geometric constants
65 A_eval = sym(zeros(4, 4, n));
66
67 % === Replace Geometric Constants with Numerical Values ===
68 L1_val = 0.4;
69 L2_val = 0.5;
70 D1_val = 0.11;
71 d11_val = 0.1;
72 d12_val = 0.9;
73 d13_val = 0.76;
74 d14_val = 0.35;
75
76 % Create a struct for substitutions
77 subs_vals = struct( ...
78     'L1', L1_val, ...
79     'L2', L2_val, ...
80     'D1', D1_val, ...
81     'd11', d11_val, ...
82     'd12', d12_val, ...
83     'd13', d13_val, ...
84     'd14', d14_val ...
85 );
86
87 % Populate A_eval by applying numerical geometric values
88 for i = 1:n
89     A_i_eval = getA(alpha(i), L(i), D(i), phi(i));
90     A_i_eval = subs(A_i_eval, subs_vals); % Apply geometric values
91     A_i_eval = simplify(A_i_eval, 'Steps', 100); % Simplify again
92     A_i_eval = vpa(A_i_eval, 6); % Round visually
93     A_eval(:,:,i) = A_i_eval;
94 end
95
96 % Total transformation matrix from base to end-effector
97 T_total = eye(4);
98 for i = 1:n
99     T_total = T_total * A(:,:,i);
100 end
101
102 % Calculate accumulated z-axes and origin positions for each joint frame
103 z_list = sym(zeros(3, n)); % List of z-axes of each frame
104 o_list = sym(zeros(3, n)); % List of origins of each frame
105
106 T = eye(4); % Initialize cumulative transformation
107 for i = 1:n
108     T = T * A(:,:,i);
109     z_list(:,i) = T(1:3, 3); % Third column of rotation matrix is z-axis
110     o_list(:,i) = T(1:3, 4); % Fourth column (position vector) is origin
111 end
112
113 % Final end-effector position
114 o_end = T_total(1:3, 4);
115
116 % Initialize Geometric Jacobian (6 rows: 3 linear, 3 angular)

```

```

117 Jg = sym(zeros(6, length(target_vars)));
118
119 % Add base frame z and p (needed for first joint calculation)
120 z_prev = [0; 0; 1]; % Z-axis of the base frame
121 o_prev = [0; 0; 0]; % Origin of the base frame
122
123 % Populate the Geometric Jacobian columns
124 for idx = 1:length(target_vars)
125     var_name = target_var_names{idx}; % Get the name of the current target variable
126
127     if strcmp(var_name, 'dh3') % Special case for prismatic joint dh3 (linear
128         displacement)
129         i = 7; % Corresponds to the 7th frame in the DH table for dh3
130         z = z_list(:, i-1); % Z-axis of the frame BEFORE the prismatic joint (z6)
131         Jg(1:3, idx) = z; % Linear velocity component is simply the z-axis
132             direction
133         Jg(4:6, idx) = [0; 0; 0]; % Angular velocity component is zero for a
134             prismatic joint
135     else % Revolute joints (phi0, phi1, phi2, phi4)
136         i = find(phi == target_vars(idx)); % Find which DH frame corresponds to this
137             joint variable
138         if isempty(i)
139             continue; % Skip if the variable is not found in the phi list
140         end
141
142         if i == 1 % For the first joint (phi0)
143             z = z_prev; % Use base frame z-axis
144             o = o_prev; % Use base frame origin
145         else % For subsequent joints
146             z = z_list(:, i-1); % Z-axis of the frame BEFORE the current joint
147             o = o_list(:, i-1); % Origin of the frame BEFORE the current joint
148         end
149
150         Jg(1:3, idx) = cross(z, (o_end - o)); % Linear velocity component (cross
151             product)
152         Jg(4:6, idx) = z; % Angular velocity component (z-axis of the joint)
153     end
154 end
155
156 % Simplify the Geometric Jacobian
157 Jg = simplify(Jg);
158
159 fprintf('\n==== Final Geometric Jacobian ====\n');
160 disp(Jg);
161
162 % Apply numerical geometric constant substitutions to the Jacobian
163 Jg_num = subs(Jg, subs_vals);
164
165 % Round visually for cleaner numerical output
166 Jg_num = vpa(Jg_num, 4); % 4 decimal places
167
168 fprintf('\n==== Geometric Jacobian with evaluated geometric constants ====\n');
169 disp(Jg_num);
170
171 % Calculation of Jacobians for Centers of Mass
172
173 % 1. First, we need the positions of the centers of mass for each link.
174 % We assume the center of mass is at the midpoint of each link,
175 % unless specified otherwise for CAD-derived bodies.
176
177 % Transformation matrices to each center of mass
178 T_cm = sym(zeros(4, 4, n));
179 J_cm = cell(1, n); % Cell array to store Jacobians for each center of mass
180

```

```

175 for i = 1:n
176     % Accumulator for transformation up to the previous link (i-1)
177     T_prev = eye(4);
178     for j = 1:i-1
179         T_j = getA(alpha(j), L(j), D(j), phi(j));
180         T_j = subs(T_j, subs_vals); % Substitute geometric constants
181         T_prev = T_prev * T_j;
182     end
183
184     % Lengths to midpoint for CM (these values are already defined in subs_vals)
185     % For links with length, this will be L(i)/2, for prismatic, D(i)/2 etc.
186     % This part assumes L(i) and D(i) are symbolic, so we need to substitute.
187     L_half = subs(L(i), subs_vals) / 2;
188     D_half = subs(D(i), subs_vals) / 2;
189
190     % Advance to the center of mass of link i
191     A_half = getA(alpha(i), L_half, D_half, phi(i));
192     A_half = subs(A_half, subs_vals); % Substitution applied here as well
193
194     % Total transformation to the center of mass of link i
195     T_cm(:,:,i) = simplify(T_prev * A_half, 'Steps', 50);
196 end
197
198 % 2. Now, we calculate the Jacobian for each center of mass.
199 for cm_idx = 1:n
200     % Initialize Jacobian for the current center of mass
201     J_current = sym(zeros(6, length(target_vars)));
202     o_cm = T_cm(1:3, 4, cm_idx); % Position of the current center of mass
203
204     for var_idx = 1:length(target_vars)
205         var_name = target_var_names{var_idx};
206
207         if strcmp(var_name, 'dh3') % Prismatic joint case
208             i = 7; % Corresponds to the 7th DH frame for dh3
209             if cm_idx >= i % dh3 affects subsequent bodies
210                 z = z_list(:, i-1); % z-axis of the frame BEFORE the prismatic
joint (z6)
211                 J_current(1:3, var_idx) = z; % Linear velocity component
212                 J_current(4:6, var_idx) = [0; 0; 0]; % Angular velocity component is
zero
213             end
214         else % Revolute joint case
215             i = find(phi == target_vars(var_idx));
216             if isempty(i)
217                 continue;
218             end
219
220             if i == 1 % First joint
221                 z = [0; 0; 1]; % Z-axis of the base frame
222                 o = [0; 0; 0]; % Origin of the base frame
223             else
224                 z = z_list(:, i-1); % Z-axis of the frame BEFORE the current joint
225                 o = o_list(:, i-1); % Origin of the frame BEFORE the current joint
226             end
227
228             if i <= cm_idx % Only previous joints influence the current link's CM
velocity
229                 J_current(1:3, var_idx) = cross(z, (o_cm - o)); % Linear velocity
component
230                 J_current(4:6, var_idx) = z; % Angular velocity component
231             end
232         end
233     end

```

```
234 J_cm{cm_idx} = simplify(J_current); % Simplify the Jacobian for the current CM
235 fprintf('\n==== Jacobian for link %d center of mass ===\n', cm_idx);
236 disp(J_cm{cm_idx});
237 end
238
239 % 3. Apply numerical substitutions to the CM Jacobians.
240 J_cm_num = cell(1, n);
241 for cm_idx = 1:n
242 if ~isempty(J_cm{cm_idx})
243 J_cm_num{cm_idx} = subs(J_cm{cm_idx}, subs_vals);
244 J_cm_num{cm_idx} = vpa(J_cm_num{cm_idx}, 4); % Round to 4 decimal places
245 fprintf('\n==== Numerical Jacobian for link %d center of mass ===\n', cm_idx)
246 ;
247 disp(J_cm_num{cm_idx});
248 end
249
250 %2. Calculation of Inertia Matrices for Each Link
251
252 % Density of aluminum in kg/m^3
253 density = 7750;
254
255 % Define geometries for each link:
256 % [Length, Type, Outer Radius/Side (r_ext), Inner Radius/Side (r_int),
257 % Outer Side (a_ext), Inner Side (a_int), Principal Axis Direction (1=X, 2=Y, 3=Z)]
258 % Type: 1 = Hollow Cylinder, 2 = Hollow Square Prism, 3 = CAD-derived (Cylinder +
259 % Sphere), 4 = CAD-derived (Square + Cylinder)
260 geometrias = {
261 [0.200, 1, 0.300/2, 0.200/2, 0, 0, 3]; % Link 1: Hollow Cylinder
262 [0.800, 3, 0.200/2, 0.150/2, 0, 0, 3]; % Link 2: CAD-derived (Cylinder +
263 % Sphere) - Note: L is probably unused for type 3/4
264 [0.400, 2, 0, 0, 0.100, 0.080, 1]; % Link 3: Hollow Square Prism
265 [0.110, 4, 0, 0, 0.100, 0.080, 3]; % Link 4: CAD-derived (Square +
266 % Cylinder)
267 [0.500, 1, 0.100/2, 0.080/2, 0, 0, 1]; % Link 5: Hollow Cylinder
268 [0.050+0.630+0.08, 1, 0.100/2, 0.080/2, 0, 0, 3]; % Link 6: Hollow Cylinder
269 [0.360, 1, 0.140/2, 0.080/2, 0, 0, 3]; % Link 7: Hollow Cylinder
270 [0.300 + 0.050, 1, 0.140/2, 0.080/2, 0, 0, 3]; % Link 8: Hollow Cylinder
271 };
272
273 % Calculate inertia matrices for each link
274 I_bodies = cell(1, n); % Cell array to store inertia matrices (in local frame)
275 masses = sym(zeros(1, n)); % Symbolic array to store mass of each link
276
277 fprintf('\n==== Link Inertia Matrices ===\n');
278 for i = 1:n
279 fprintf('=====Link ===== %d =====\n', i);
280 params = geometrias{i};
281 L_ = params(1);
282 tipo = params(2);
283 r_ext = params(3);
284 r_int = params(4);
285 a_ext = params(5);
286 a_int = params(6);
287
288 % Calculate inertia matrix for the current link
289 I_bodies{i} = computeInertiaMatrix(tipo, L_, r_ext, r_int, a_ext, a_int, density);
290
291 % Calculate mass based on geometry type
292 V = 0; % Initialize volume
293 if tipo == 1 % Hollow cylinder
```

```

292     V = pi * (r_ext^2 - r_int^2) * L_;
293 elseif tipo == 2 % Hollow square prism
294     V = (a_ext^2 - a_int^2) * L_;
295 elseif tipo == 3 % CAD-derived (Cylinder + Sphere) - Volume derived from CAD
296     V = 0.012967447276;
297 elseif tipo == 4 % CAD-derived (Square + Cylinder) - Volume derived from CAD
298     V = 0.000740789794;
299 end
300 masses(i) = density * V; % Calculate mass
301
302 fprintf('Link %d - Mass: %.3f kg\n', i, masses(i));
303 disp(I_bodies{i});
304 end
305
306 %3. Calculation of the Mass Matrix using the Jacobian
307
308 % Initialize the inertial mass matrix
309 M = sym(zeros(length(target_vars)));
310
311 % Iterate through each movable link (assuming all links contribute)
312 for k = 1:n
313     % Get the linear and angular Jacobian for the center of mass of link k
314     % (We use the pre-calculated Jacobians for the center of mass)
315     if isempty(J_cm_num{k})
316         continue; % Skip if there's no Jacobian for this center of mass
317     end
318     Jk_lin = J_cm_num{k}(1:3,:); % Linear Jacobian for CM
319     Jk_ang = J_cm_num{k}(4:6,:); % Angular Jacobian for CM
320
321     % Get the accumulated transformation T_k for body k
322     % T_k = eye(4);
323     % for m = 1:k
324     %     T_k = T_k * A_eval(:,:,m); % Use A_eval which has numerical geometric
325     % constants
326     % end
327
328     % Get the rotation matrix R_k for body k
329     %R_k = T_k(1:3, 1:3);
330
331     % Get the transformation matrix to the center of mass of body k
332     T_k = T_cm(:,:,k); % More accurate for rotating the inertia matrix
333     R_k = T_k(1:3, 1:3);
334
335     % Transform the inertia matrix from the local frame to the base frame
336     I_global_k = vpa(simplify(R_k * I_bodies{k} * R_k.'), 4);
337
338     fprintf('\n--- Inertia Transformation for Link %d ---\n', k);
339     fprintf('R_k:\n');
340     disp(R_k);
341     fprintf('I_body (local):\n');
342     disp(I_bodies{k});
343     fprintf('I_global = R_k * I_local * R_k^T:\n');
344     disp(I_global_k);
345
346     % Calculate contribution to the mass matrix (using the generalized inertia
347     % equation)
348     M = M + masses(k)*(Jk_lin')*Jk_lin + (Jk_ang')*I_global_k*Jk_ang;
349 end
350 M = simplify(M); % Simplify the final mass matrix
351 fprintf('\n==== Inertial Mass Matrix M(q) ====\n');
352 vpa(M, 3); % Display with 3 significant figures
353 disp(M);

```

```

353 % 4. Calculation of Coriolis and Centrifugal Terms
355
356 % Initialize the Christoffel symbol matrix (which forms C(q, dq)*dq)
357 C = sym(zeros(length(target_vars)));
358
359 % Define symbolic time for differentiation
360 syms t real;
361
362 % Define symbolic joint position (q) and velocity (dq) vectors
363 q = target_vars;
364 dq = sym(zeros(size(q)));
365 for i = 1:length(q)
366     dq(i) = sym(['d' char(q(i))], 'real'); % e.g., dphi0, dphil
367 end
368
369 % Calculate Christoffel symbols (and thus the Coriolis/Centrifugal matrix C)
370 % C_ijk = 0.5 * (dM_ij/dq_k + dM_ik/dq_j - dM_jk/dq_i)
371 % C_matrix(i,j) = sum(C_ijk * dq_k)
372 for i = 1:length(q)
373     for j = 1:length(q)
374         for k = 1:length(q)
375             % Christoffel symbol component
376             c = 0.5*(diff(M(i,j), q(k)) + diff(M(i,k), q(j)) - diff(M(j,k), q(i)));
377             C(i,j) = C(i,j) + c*dq(k); % Accumulate into C matrix
378         end
379     end
380 end
381
382 C = simplify(C); % Simplify the C matrix
383 fprintf('\n==> Coriolis and Centrifugal Terms C(q,dq) ==\n');
384 vpa(C, 3); % Display with 3 significant figures
385 disp(C);
386
387 %5. Calculation of Gravity Terms
388
389 % Assume gravity acts in the -Z direction
390 g = [0; 0; -9.81];
391
392 % Initialize gravity vector
393 G = sym(zeros(length(q),1));
394
395 % Calculate total potential energy V
396 V = 0;
397 for k = 1:n
398     % Get the transformation matrix T_k for link k (to its end, or to its CM if more
399     % precise)
400     T_k = eye(4);
401     for m = 1:k
402         T_k = T_k * A_eval(:,:,m); % Use A_eval for numerical geometric constants
403     end
404
405     % Determine the local center of mass (COM) position based on the geometry type
406     % and principal axis.
407     params = geometrias{k};
408     L_k = params(1); % Length of the body
409     tipo_k = params(2); % Type of body
410     direccion = 3; % Default principal axis is Z
411
412     if size(params, 2) >= 7
413         direccion = params(7); % If specified, use the given principal axis
414     end
415
416 end

```

```

414 % === Local Center of Mass based on principal direction ===
415 com_local = zeros(3,1); % Initialize local COM vector
416 switch direccion
417     case 1 % X-axis
418         com_local(1) = L_k/2;
419     case 2 % Y-axis
420         com_local(2) = L_k/2;
421     case 3 % Z-axis
422         com_local(3) = L_k/2;
423     otherwise
424         error('Unrecognized direction for body %d', k);
425 end
426
427 % === If the body is CAD-derived, override com_local with CAD values ===
428 if tipo_k == 3 % CAD type 3 (Cylinder + Sphere)
429     com_local = zeros(3,1); % Reset to zero
430     com_local(direccion) = 0.470732; % Specific CAD-derived COM distance for
this type
431 elseif tipo_k == 4 % CAD type 4 (Square + Cylinder)
432     com_local = zeros(3,1); % Reset to zero
433     com_local(direccion) = 0.111934; % Specific CAD-derived COM distance for
this type
434 end
435
436 % === Transform the center of mass to the base frame ===
437 R_k = T_k(1:3, 1:3); % Rotation matrix from T_k
438 com_k = R_k * com_local + T_k(1:3, 4); % Transform local COM to base frame
439
440 % Accumulate potential energy: V = sum(mass_k * g' * com_k)
441 V = V + masses(k)*g'*com_k;
442 end
443
444 % Calculate partial derivatives of V with respect to each joint variable (q_i)
445 % G(i) = dV/dq_i
446 for i = 1:length(q)
447     G(i) = diff(V, q(i));
448 end
449
450 G = simplify(G); % Simplify the gravity vector
451 fprintf('\n==== Gravity Terms G(q) ===\n');
452 vpa(G, 3); % Display with 3 significant figures
453 disp(G);
454
455 % 6. Final Equations of Motion
456
457 fprintf('\n==== Complete Equations of Motion ===\n');
458 fprintf('M(q)*ddq + C(q,dq)*dq + G(q) = tau\n');
459
460 tol = 1e-6; % Threshold to consider a number as zero
461
462 % === Cleaning and Rounding M Matrix ===
463 M_clean = vpa(M, 3); % Step 1: Round to 3 significant figures
464 M_clean = mapSymType(M_clean, 'constant', ...
465     @(x) piecewise(abs(x) < tol, 0, x)); % Step 2: Set small terms to zero
466 M_clean = simplify(M_clean, 'Steps', 100); % Step 3: Further simplification
467
468 tol = 1e-4;
469 digits(5);
470
471 M_clean = simplify(M, 'Steps', 50); % Opcional: simplifica primero
472
473 M_clean = arrayfun(@(x) vpa(x, 3), M_clean);
474

```

```
475
476
477 M_clean = mapSymType(M_clean, 'constant', ...
478     @(x) piecewise(abs(x) < tol, 0, x));
479
480 M_clean = simplify(M_clean, 'Steps', 100);
481
482
483 fprintf('\n==== Cleaned and Rounded M Matrix test 2====\n');
484 disp(M_clean)
485
486
487 % === Cleaning and Rounding C Matrix ===
488 C_clean = vpa(C, 3); % Step 1: Round to 3 significant figures
489 C_clean = mapSymType(C_clean, 'constant', ...
490     @(x) piecewise(abs(x) < tol, 0, x)); % Step 2: Set small terms to zero
491 C_clean = simplify(C_clean, 'Steps', 100); % Step 3: Further simplification
492
493 tol = 1e-4;
494 digits(5);
495
496 C_clean = simplify(C, 'Steps', 50); % Opcional: simplifica primero
497
498 C_clean = arrayfun(@(x) vpa(x, 3), C_clean);
499
500
501 C_clean = mapSymType(C_clean, 'constant', ...
502     @(x) piecewise(abs(x) < tol, 0, x));
503
504 % === Paso 4: Simplificar final ===
505 C_clean = simplify(C_clean, 'Steps', 100);
506
507 % === Mostrar resultado ===
508 fprintf('\n==== Cleaned and Rounded C Matrix test 2====\n');
509 disp(C_clean)
510
511
512 % === Cleaning and Rounding G Vector ===
513 G_clean = vpa(G, 3); % Step 1: Round
514 G_clean = mapSymType(G_clean, 'constant', ...
515     @(x) piecewise(abs(x) < tol, 0, x)); % Step 2: Set small terms to zero
516 G_clean = simplify(G_clean, 'Steps', 100); % Step 3: Further simplification
517
518
519 tol = 1e-4; % Umbral de tolerancia para considerar como cero
520 digits(5); % de vpa
521
522 G_clean = simplify(G, 'Steps', 50); % Opcional: simplifica primero
523 G_clean = arrayfun(@(x) vpa(x, 3), G_clean);
524 G_clean = mapSymType(G_clean, 'constant', ...
525     @(x) piecewise(abs(x) < tol, 0, x));
526
527 G_clean = simplify(G_clean, 'Steps', 100);
528
529 % === Mostrar resultado ===
530 fprintf('\n==== Cleaned and Rounded G Matrix test 2====\n');
531 disp(G_clean)
532
533
534 % Save the cleaned matrices to a .mat file
535 save('lagrange_matrices.mat', 'M_clean', 'C_clean', 'G_clean', 'target_vars');
536
537 % Helper function to compute inertia matrix for various geometries
```

```

538 function I_body = computeInertiaMatrix(type, L, r_ext, r_int, a_ext, a_int, density)
539     fprintf('\n===== Inertia Calculation=====\\n');
540     fprintf('Length (L): %.3f m\\n', L);
541
542
543     I_xx = 0; I_yy = 0; I_zz = 0; % Initialize inertia components
544
545     if type == 1 % Hollow cylinder
546         fprintf('Geometry: Hollow Cylinder\\n');
547         fprintf('Outer radius (r_ext): %.3f m\\n', r_ext);
548         fprintf('Inner radius (r_int): %.3f m\\n', r_int);
549         % Formulas for hollow cylinder inertia about its principal axes (assuming Z-
550         % axis along length)
551         I_xx = (1/12) * density * pi * (r_ext^2 - r_int^2) * L * (3*(r_ext^2 + r_int
552         ^2) + L^2);
553         I_yy = I_xx; % Symmetric for cylinders
554         I_zz = (1/2) * density * pi * (r_ext^2 - r_int^2) * L * (r_ext^2 + r_int^2);
555
556     elseif type == 2 % Hollow square prism
557         fprintf('Geometry: Hollow Square Prism\\n');
558         fprintf('Outer side length (a_ext): %.3f m\\n', a_ext);
559         fprintf('Inner side length (a_int): %.3f m\\n', a_int);
560         % Formulas for hollow square prism inertia about its principal axes (
561         % assuming Z-axis along length)
562         I_xx = (1/12) * density * (a_ext^2 - a_int^2) * L * (L^2 + (1/3)*(a_ext^2 +
563         a_int^2));
564         I_yy = I_xx; % Symmetric for square prisms
565         I_zz = (1/6) * density * (a_ext^2 - a_int^2) * L * (a_ext^2 + a_int^2);
566
567     elseif type == 3 % CAD-derived values for a specific part (Cylinder + Sphere)
568         fprintf('Geometry: Hollow Cylinder + Sphere (from CAD)\\n');
569         I_xx = 7749265.619e-6; % From CAD
570         I_yy = 776332.607e-6; % From CAD
571         I_zz = 7749265.619e-6; % From CAD
572         % Note: Center of gravity (Schwerpunkt) at 563.834 mm along the principal
573         % direction
574     elseif type == 4 % CAD-derived values for another specific part (Square + Top
575     % Cylinder)
576         fprintf('Geometry: Hollow Square + Top Cylinder (from CAD)\\n');
577         fprintf('Using CAD-derived inertia values :\\n');
578         I_xx = 36063.992e-6; % From CAD
579         I_yy = 18013.291e-6; % From CAD
580         I_zz = 36063.991e-6; % From CAD
581         % Note: Center of gravity (Schwerpunkt) at 129.031 mm along the principal
582         % axis
583     else
584         error('Invalid geometry type. Must be 1 (cylinder), 2 (square), 3 (CAD type
585         1), or 4 (CAD type 2).');
586     end
587
588     % Round inertia components and form the diagonal inertia matrix
589     I_xx = round(I_xx, 4);
590     I_yy = round(I_yy, 4);
591     I_zz = round(I_zz, 4);
592     I_body = diag([I_xx, I_yy, I_zz]);
593     I_body = vpa(I_body, 4); % Display with 4 significant figures
594
595 end
596
597
598 %7. Numerical Evaluation of Torques/Forces using Velocity Profiles
599
600 % Load pre-generated trajectories (ensure 'ruckig_trajectories.mat' exists)
601 load('ruckig_trajectories.mat'); % Loads 'trajectories' struct

```

```
593 % Extract time vectors (assuming phi0 has the longest/most complete time vector)
594 time_phi0 = trajectories.phi0.time;
595 time_phi1 = trajectories.phi1.time;
596 time_phi2 = trajectories.phi2.time;
597 time_dh3 = trajectories.dh3.time;
598 time_phi4 = trajectories.phi4.time;
599
600
601 % Extract position, velocity, and acceleration data for each joint
602 phi0_values = trajectories.phi0.position;
603 phi1_values = trajectories.phi1.position;
604 phi2_values = trajectories.phi2.position;
605 dh3_values = trajectories.dh3.position;
606 phi4_values = trajectories.phi4.position;
607
608 dphi0_values = trajectories.phi0.velocity;
609 dphi1_values = trajectories.phi1.velocity;
610 dphi2_values = trajectories.phi2.velocity;
611 ddh3_values = trajectories.dh3.velocity;
612 dphi4_values = trajectories.phi4.velocity;
613
614 ddphi0_values = trajectories.phi0.acceleration;
615 ddphi1_values = trajectories.phi1.acceleration;
616 ddphi2_values = trajectories.phi2.acceleration;
617 dddh3_values = trajectories.dh3.acceleration;
618 ddphi4_values = trajectories.phi4.acceleration;
619
620 % Number of time points (before interpolation/padding)
621 n_points_orig = length(phi0_values);
622
623 % Preallocate space for calculated torques/forces
624 tau_eval = zeros(5, n_points_orig); % 5 joints, n_points_orig time steps
625
626 % Force all trajectory data to be row vectors for consistent indexing
627 phi0_values = phi0_values(:)';
628 phi1_values = phi1_values(:)';
629 phi2_values = phi2_values(:)';
630 dh3_values = dh3_values(:)';
631 phi4_values = phi4_values(:)';
632
633 dphi0_values = dphi0_values(:)';
634 dphi1_values = dphi1_values(:)';
635 dphi2_values = dphi2_values(:)';
636 ddh3_values = ddh3_values(:)';
637 dphi4_values = dphi4_values(:)';
638
639 ddphi0_values = ddphi0_values(:)';
640 ddphi1_values = ddphi1_values(:)';
641 ddphi2_values = ddphi2_values(:)';
642 dddh3_values = dddh3_values(:)';
643 ddphi4_values = ddphi4_values(:)';
644
645 % Display original trajectory statistics
646 fprintf('\n===== ORIGINAL TRAJECTORY STATISTICS =====\n');
647 % Helper function to print stats (defined at the end of the script)
648 print_stats(trajectories.phi0.position,      'phi0.position');
649 print_stats(trajectories.phi0.velocity,       'phi0.velocity');
650 print_stats(trajectories.phi0.acceleration, 'phi0.acceleration');
651 print_stats(trajectories.phi0.jerk,          'phi0.jerk');
652
653 print_stats(trajectories.phi1.position,       'phi1.position');
654 print_stats(trajectories.phi1.velocity,        'phi1.velocity');
655 print_stats(trajectories.phi1.acceleration, 'phi1.acceleration');
```

```

656 print_stats(trajectories.phi1.jerk,           'phi1.jerk');
657
658 print_stats(trajectories.phi2.position,       'phi2.position');
659 print_stats(trajectories.phi2.velocity,        'phi2.velocity');
660 print_stats(trajectories.phi2.acceleration,   'phi2.acceleration');
661 print_stats(trajectories.phi2.jerk,            'phi2.jerk');
662
663 print_stats(trajectories.dh3.position,         'dh3.position');
664 print_stats(trajectories.dh3.velocity,          'dh3.velocity');
665 print_stats(trajectories.dh3.acceleration,     'dh3.acceleration');
666 print_stats(trajectories.dh3.jerk,              'dh3.jerk');
667
668 print_stats(trajectories.phi4.position,         'phi4.position');
669 print_stats(trajectories.phi4.velocity,          'phi4.velocity');
670 print_stats(trajectories.phi4.acceleration,     'phi4.acceleration');
671 print_stats(trajectories.phi4.jerk,              'phi4.jerk');
672
673 fprintf('\n==== Shape of position variables (before common time axis) ====\n');
674 disp(['phi0: ', mat2str(size(phi0_values))]);
675 disp(['phi1: ', mat2str(size(phi1_values))]);
676 disp(['phi2: ', mat2str(size(phi2_values))]);
677 disp(['dh3: ', mat2str(size(dh3_values))]);
678 disp(['phi4: ', mat2str(size(phi4_values))]);
679
680 % Create a common time axis and interpolate/pad trajectory data.
681 % 1. Find the maximum final time across all trajectories
682 final_time = max([
683     trajectories.phi0.time(end);
684     trajectories.phi1.time(end);
685     trajectories.phi2.time(end);
686     trajectories.dh3.time(end);
687     trajectories.phi4.time(end)
688 ]);
689
690 % 2. Create a new common time vector with the maximum number of points
691 n_points = max([length(trajectories.phi0.time); length(trajectories.phi1.time);
692                  length(trajectories.phi2.time); length(trajectories.dh3.time);
693                  length(trajectories.phi4.time)]);
694 common_time = linspace(0, final_time, n_points);
695
696 % Helper function to pad data with zeros if its length is less than target_len
697 pad_with_zeros = @(data, target_len) [data, zeros(1, target_len - length(data))];
698
699 % Pad all trajectory data to the common length (n_points)
700 phi0_values    = pad_with_zeros(trajectories.phi0.position, n_points);
701 dphi0_values   = pad_with_zeros(trajectories.phi0.velocity, n_points);
702 ddphi0_values = pad_with_zeros(trajectories.phi0.acceleration, n_points);
703
704 phi1_values    = pad_with_zeros(trajectories.phi1.position, n_points);
705 dphi1_values   = pad_with_zeros(trajectories.phi1.velocity, n_points);
706 ddphi1_values = pad_with_zeros(trajectories.phi1.acceleration, n_points);
707
708 phi2_values    = pad_with_zeros(trajectories.phi2.position, n_points);
709 dphi2_values   = pad_with_zeros(trajectories.phi2.velocity, n_points);
710 ddphi2_values = pad_with_zeros(trajectories.phi2.acceleration, n_points);
711
712 dh3_values     = pad_with_zeros(trajectories.dh3.position, n_points);
713 ddh3_values    = pad_with_zeros(trajectories.dh3.velocity, n_points);
714 dddh3_values   = pad_with_zeros(trajectories.dh3.acceleration, n_points); % Note:
715 % dddh3_values is velocity here, dddh3_values is acceleration
716
717 phi4_values    = pad_with_zeros(trajectories.phi4.position, n_points);
718 dphi4_values   = pad_with_zeros(trajectories.phi4.velocity, n_points);

```

```

718 ddphi4_values = pad_with_zeros(trajectories.phi4.acceleration, n_points);
719
720 % Display trajectory statistics after padding/interpolation
721 fprintf('\n===== STATISTICS AFTER INTERPOLATION/PADDING =====\n');
722 print_stats(phi0_values,      'phi0.position (interp)');
723 print_stats(dphi0_values,    'phi0.velocity (interp)');
724 print_stats(ddphi0_values,   'phi0.acceleration (interp)');
725
726 print_stats(phi1_values,      'phi1.position (interp)');
727 print_stats(dphi1_values,    'phi1.velocity (interp)');
728 print_stats(ddphi1_values,   'phi1.acceleration (interp)');
729
730 print_stats(phi2_values,      'phi2.position (interp)');
731 print_stats(dphi2_values,    'phi2.velocity (interp)');
732 print_stats(ddphi2_values,   'phi2.acceleration (interp)');
733
734 print_stats(dh3_values,       'dh3.position (interp)');
735 print_stats(ddh3_values,     'dh3.velocity (interp)');
736 print_stats(dddh3_values,    'dh3.acceleration (interp)');
737
738 print_stats(phi4_values,      'phi4.position (interp)');
739 print_stats(dphi4_values,    'phi4.velocity (interp)');
740 print_stats(ddphi4_values,   'phi4.acceleration (interp)');
741
742 time = common_time; % Use the common time vector
743 n_points = length(time); % Update n_points to the new length
744
745 fprintf('\n== Shape of position variables (after common time axis) ==\n');
746 disp(['phi0: ', mat2str(size(phi0_values))]);
747 disp(['phi1: ', mat2str(size(phi1_values))]);
748 disp(['phi2: ', mat2str(size(phi2_values))]);
749 disp(['dh3: ', mat2str(size(dh3_values))]);
750 disp(['phi4: ', mat2str(size(phi4_values))]);
751
752 % Group symbolic variables
753 q_syms = [phi0, phi1, phi2, dh3, phi4];
754 dq_syms = [dphi0, dphi1, dphi2, ddh3, dphi4];
755 ddq_syms = [ddphi0, ddphi1, ddphi2, dddh3, ddphi4];
756
757 % Preallocate torque array again with updated n_points
758 tau_eval = zeros(5, n_points);
759
760 % === Evaluate torques/forces using the interpolated trajectory data ===
761 for i = 1:n_points
762     % Numerical values for position, velocity, acceleration at current time step
763     q_num = [phi0_values(i), phi1_values(i), phi2_values(i), dh3_values(i),
764              phi4_values(i)];
765     dq_num = [dphi0_values(i), dphi1_values(i), dphi2_values(i), ddh3_values(i),
766               dphi4_values(i)];
767     ddq_num = [ddphi0_values(i), ddphi1_values(i), ddphi2_values(i), dddh3_values(i),
768                ddphi4_values(i)];
769
770     try
771         % Substitute numerical values into the symbolic M, C, G matrices
772         M_i = double(subs(M_clean, [q_syms, dq_syms, ddq_syms], [q_num, dq_num,
773                     ddq_num]));
774         C_i = double(subs(C_clean, [q_syms, dq_syms], [q_num, dq_num]));
775         G_i = double(subs(G_clean, q_syms, q_num));
776
777         % Calculate torque/force using the dynamic equation: tau = M*ddq + C*dq + G
778         tau_eval(:, i) = M_i * ddq_num' + C_i * dq_num' + G_i;
779
780         % Display progress and calculated values every 100 steps or at the end
781     end
782 end

```

```

777     if mod(i, 100) == 1 || i == n_points
778         fprintf("\ntau_eval(:, i) = M_i * ddq_num' + C_i * dq_num' + G_i'\n");
779         fprintf('\n== Step %d / %d ==\n', i, n_points);
780         fprintf('Time = %.3f s\n', time(i));
781         fprintf('q      = [%8.4f, %8.4f, %8.4f, %8.4f, %8.4f]\n', q_num);
782         fprintf('dq     = [%8.4f, %8.4f, %8.4f, %8.4f, %8.4f]\n', dq_num);
783         fprintf('ddq    = [%8.4f, %8.4f, %8.4f, %8.4f, %8.4f]\n', ddq_num);
784         fprintf('tau   = [%8.4f, %8.4f, %8.4f, %8.4f, %8.4f]\n', tau_eval(:,i));
785
786         fprintf("Tau_i:\n");
787         disp(tau_eval(:, i))
788         fprintf("M_i * ddq_num':\n");
789         disp(M_i * ddq_num')
790         fprintf("M_i:\n");
791         disp(M_i)
792         fprintf("C_i * dq_num':\n");
793         disp(C_i * dq_num')
794         fprintf("C_i:\n");
795         disp(C_i)
796         fprintf("G_i:\n");
797         disp(G_i)
798         fprintf('\n==== Numerical torque/force values per time instant ===\n');
799         fprintf('%8s | %10s | %10s | %10s | %10s | %10s\n', 'Time', 'Tau_0', '
Tau_1', 'Tau_2', 'Tau_dh3', 'Tau_4');
800         fprintf('%8.3f | %10.3f | %10.3f | %10.3f | %10.3f | %10.3f\n', ...
time(i), tau_eval(1,i), tau_eval(2,i), tau_eval(3,i), tau_eval(4,i),
tau_eval(5,i));
801     end
802
803     catch ME
804         fprintf("Error at step %d: %s\n", i, ME.message);
805         disp("Values used:");
806         disp([q_num, dq_num, ddq_num]);
807         break; % Exit loop on error
808     end
809 end
810
811 fprintf('\nGenerating Torque vs Time and Position vs Time plots (approx.)...\n');
812
813 % === Symbolic Labels and Plot Configuration ===
814 joint_labels = {'$\\tau_0$ [Nm]', '$\\tau_1$ [Nm]', '$\\tau_2$ [Nm]', '$\\tau_{d_3}$ [N]',
'$\\tau_4$ [Nm]'};
815 symbol_names = {'$\\varphi_0$', '$\\varphi_1$', '$\\varphi_2$', '$d_{h3}$', '$\\varphi_4$'};
816 units_tau = {'Nm', 'Nm', 'Nm', 'N', 'Nm'};
817 positions = {phi0_values, phi1_values, phi2_values, dh3_values, phi4_values};
818 file_names = {'Aprox_tau_phi0.png', 'Aprox_tau_phi1.png', 'Aprox_tau_phi2.png',
'Aprox_tau_dh3.png', 'Aprox_tau_phi4.png'};
819 is_angle = [true, true, true, false, true]; % Identify which joints are
angular (for degrees conversion)
820
821 % === Matching Colors for Y-axes ===
822 color_torque = [0 0 0]; % Black
823 color_position = [0 0 0.6]; % Dark Blue
824
825
826 % Generate plots for each joint
827 for i = 1:5
828     figure('Name', ['Torque and Position - Joint ', symbol_names{i}], ...
        'Color', 'w', 'Position', [100 100 900 600]);
829
830     % === Left Y-axis: Torque or Force ===
831     yyaxis left;
832     ax = gca;
833

```

```
834 ax.YColor = color_torque; % Set color for left Y-axis
835 plot(time, tau_eval(i,:), '--', 'LineWidth', 2, 'Color', color_torque);
836 ylabel(joint_labels{i}, 'FontSize', 12, 'Interpreter', 'latex');
837 ylim padded; % Adjust y-axis limits automatically
838 grid on;
839
840 % === Right Y-axis: Angular or Linear Position ===
841 yyaxis right;
842 ax.YColor = color_position; % Set color for right Y-axis
843 if is_angle(i)
844     pos = rad2deg(positions{i}); % Convert radians to degrees for angular joints
845     plot(time, pos, '--', 'LineWidth', 1.5, 'Color', color_position);
846     ylabel(sprintf('%s [degree]', symbol_names{i}), 'Interpreter', 'latex', 'FontSize', 12);
847 else
848     plot(time, positions{i}, '--', 'LineWidth', 1.5, 'Color', color_position);
849     ylabel(sprintf('%s [m]', symbol_names{i}), 'Interpreter', 'latex', 'FontSize', 12);
850 end
851 ylim padded; % Adjust y-axis limits automatically
852
853 % === X-axis: Time ===
854 xlabel('Time [s]', 'FontSize', 12, 'Interpreter', 'latex');
855
856 % === Title and Legend ===
857 title(sprintf('Approx. Torque/Force and Position for %s', symbol_names{i}), ...
858       'Interpreter', 'latex', 'FontSize', 14);
859 if is_angle(i)
860     legend({'Torque [Nm]', 'Position [°]'}, 'Location', 'best', 'Interpreter', 'latex');
861 else
862     legend({'Force [N]', 'Position [m]'}, 'Location', 'best', 'Interpreter', 'latex');
863 end
864 set(gca, 'FontSize', 12); % Set font size for current axes
865 box on; % Display box around the plot
866
867 % Save plot as JPG with high resolution
868
869 jpg_name = sprintf(file_names{i}, '.png', '.jpg');
870 exportgraphics(gcf, jpg_name, 'Resolution', 300);
871 end
872
873 % Helper function to print signal statistics
874 function print_stats(signal, name)
875     fprintf('\n--- Statistics for %s ---\n', name);
876     fprintf(' Minimum: %.6f\n', min(signal));
877     fprintf(' Maximum: %.6f\n', max(signal));
878     fprintf(' Average: %.6f\n', mean(signal));
879     fprintf(' Std. Dev: %.6f\n', std(signal));
880 end
881
882 % Calculation and Visualization of Total Robot Mass
883
884 % Sum all individual link masses
885 total_mass = sum(masses);
886
887 fprintf('\n--- Total Robot Mass ===\n');
888 fprintf('Total mass: %.3f kg\n', total_mass);
889 fprintf('Breakdown by links:\n');
890 for i = 1:n
891     fprintf(' Link %d: %.3f kg\n', i, masses(i));
892 end
```

Kapitel 5

Fazit

- Zu Beginn des Projekts wurde die inverse Kinematik des SCARA-Roboters durch eine Kombination aus algebraischen und geometrischen Methoden erfolgreich gelöst. Diese hybride Herangehensweise ermöglichte eine robuste Bestimmung der Gelenkvariablen für komplexe Zielpositionen und diente als Grundlage für die spätere Dynamikanalyse.
- Aufbauend auf dem kinematischen Modell wurde die Jacobi-Matrix detailliert hergeleitet, wobei sowohl lineare als auch rotatorische Komponenten betrachtet wurden. Diese Matrix war entscheidend für die Analyse von Singularitäten und für die spätere Ableitung der Lagrange-Gleichungen, da sie die Verbindung zwischen Gelenk- und Endeffektorbewegung beschreibt.
- Ziel dieses Projekts war es, ein SCARA-Robotersystem mit vier Freiheitsgraden vollständig zu modellieren, zu analysieren und zu simulieren. Dies wurde erfolgreich durch die Erstellung eines vollständigen kinematischen Modells mit Hilfe der Denavit-Hartenberg-Parameter sowie durch die Definition und Berechnung von Trajektorien unter Berücksichtigung von Ruckbegrenzungen erreicht.
- Die Verwendung von *Ruckbegrenzungsprofilen* ermöglichte eine realitätsnahe Bewegungsplanung, welche nicht nur Beschleunigungs- und Geschwindigkeitsgrenzen einhält, sondern auch mechanische Belastungen reduziert. Diese Profile wurden erfolgreich auf alle Gelenke angewendet.
- Auf Grundlage dieser Bewegungsprofile wurden die verallgemeinerten Koordinaten sowie deren zeitliche Ableitungen für die Anwendung in der Lagrange-Dynamik berechnet. Die resultierenden Gelenkmomente wurden sowohl symbolisch vereinfacht als auch numerisch exakt ausgewertet, wodurch die Unterschiede zwischen beiden Methoden aufgezeigt werden konnten.
- Der entwickelte MATLAB-Code bietet eine strukturierte und automatisierbare Umgebung zur Generierung von Transformationsmatrizen, zur Berechnung der kinetischen und potenziellen Energie sowie zur Ermittlung der Dynamikgleichungen über das Lagrange-Formalismus. Besonders hervorzuheben ist die Möglichkeit, die gesamte Trajektorie systematisch zu evaluieren und dabei sowohl symbolische als auch numerische Methoden miteinander zu vergleichen.
- Die im Projekt entwickelten Methoden sind universell auf andere Robotertypen übertragbar, sofern deren kinematische Struktur entsprechend definiert ist. Somit bietet dieses Projekt nicht nur eine Lösung für den gegebenen Roboter, sondern stellt auch eine wertvolle Vorlage für zukünftige robotische Entwurfsprozesse dar.

Literaturverzeichnis

- [1] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 3rd edition, 2005.
- [2] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing. Springer, 2000.
- [3] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2 edition, 2006.