Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Hendrik Thönnißen

Lageregelung eines redundanten Roboters mit elastischen Elementen

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Hendrik Thönnißen

Lageregelung eines redundanten Roboters mit elastischen Elementen

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Maschinenbau/Berechnung und Simulation
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr.-Ing. Thomas Frischgesell
Zweitprüfer: Prof. Dr.-Ing. Wolfgang Schulz

Abgabedatum: 31. Januar 2014

Zusammenfassung

Hendrik Thönnißen

Thema der Masterarbeit

Lageregelung eines redundanten Roboters mit elastischen Elementen

Stichworte

Redundanz, Roboter, Lageregelung, Bachmann, Faulhaber

Kurzzusammenfassung

In der Masterthesis wird das Verhalten eines redundanten 3-RRR Roboters untersucht. Hierzu wurde der Roboter mit einer Bachmann Steuerung und Faulhaber Servomotoren in Betrieb genommen. Für den Roboter wurden elastische Armsegmente konstruiert und gefertigt, um die Steifigkeit des Roboters zu verringern. Dieser Roboter wurde in einem SimMechanics Modell mit elastischen Elementen modelliert. Eine Positionsmessung des Roboters mit einer Kamera wurde in Matlab realisiert. Für die Lageregelung des Roboters wurde eine Korrekturmethode entwickelt, welche in dem SimMechanics Modell und an dem Roboter umgesetzt wurde. Abschließend wurden die Messungen am Roboter mit den Simulationsdaten des SimMechanics Modells verglichen und bewertet.

Hendrik Thönnißen

Master Thesis title

Position control of a redundant robot with flexible elements

Keywords

position control, robot, redundancy, Bachmann, Faulhaber

Abstract

In this master thesis the behavior of a redundant 3-RRR robot is investigated. Therefore the robot was put into opertation with a Bachmann control and Faulhaber servo motors. For the robot fexible arm segments were constructed and produced to reduce the stiffness of the robot. This robot was modeled in a SimMechanics model with flexible elements. A position measurement of the robot with a camera was implemented in Matlab. For the position control of the robot, a correction method has been developed which was implemented at the SimMechanics model and the robot. Finally, the measurements were compared and evaluated on the robot with the simulation data of the SimMechanics model.

Thema der Masterarbeit

Lageregelung eines redundanten Roboters mit elastischen Elementen

Aufgabenstellung:

In dieser Masterthesis soll das Verhalten von parallelen redundanten Robotern mit flexiblen Elementen untersucht werden. Dies wird am Beispiel des planaren, parallelen und redundanten 3-RRR Roboters durchgeführt. Als flexible Elemente sollen für den Roboter elastische Arme konstruiert werden. Durch den elastischen Anteil ergibt sich bei einem Krafteinfluss eine Abweichung des Endeffektors. Um diese Abweichung am Roboter zu messen, ist eine geeignete Messtechnik auszuwählen und in Betrieb zu nehmen. Um an verschiedenen Punkten im Arbeitsraum zu messen, muss eine Steuerung ausgewählt werden, mit der der Roboter im Arbeitsraum in vorgegebenen Positionen gesteuert und dort dann arretiert werden kann. Die elastischen Arme sollen auch in einem Simulink/SimMechanics Modell für den Roboter nachgebildet werden. Anschließend sollen die Ergebnisse der Simulation mit den gemessenen Werten am Roboter verglichen werden.

Da die Regelung im dynamischen Bereich durch eine Vielzahl von dynamischen Effekten beeinflusst wird, soll im Rahmen dieser Arbeit eine Regelung für den statischen Fall entworfen werden. Diese Regelung soll die Position des Endeffektors korrigieren, die durch die elastischen Roboterarme beeinflusst wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelzeichenverzeichnis

Formelzeichen	Einheit	Beschreibung
a_0	mm	0. Koeffizient der Abbildungsfunktion f
a_1	mm/Pixel	1. Koeffizient der Abbildungsfunktion f
a_2	mm/Pixel	2. Koeffizient der Abbildungsfunktion f
a_3	mm/Pixel^2	3. Koeffizient der Abbildungsfunktion f
B	m	Bildgröße
b	m	Bildweite
b_0	mm	0. Koeffizient der Abbildungsfunktion g
b_1	mm/Pixel	1. Koeffizient der Abbildungsfunktion g
b_2	mm/Pixel	2. Koeffizient der Abbildungsfunktion g
b_3	mm/Pixel^2	3. Koeffizient der Abbildungsfunktion g
c	m	Länge des Aluminiumprofils
d	m	Kugeldurchmesser
E	N/mm^2	Elastizitätsmodul
e	%	prozentualer Fehler
G	m	Gegenstandsgröße
g	m	Gegenstandsweite
h	m	Länge der fiktiv verformten Armsegmente
I	m^4	Flächenträgheitsmoment
J	kgm^2	Massenträgheitsmoment
k	Nm/rad	Torsionssteifigkeit
L	m	Armlänge
l	m	Elementlänge
l_B	m	Länge des Bodys
M	kg	Armmasse
m	kg	Elementmasse
m_B	kg	Masse des Bodys
n	−	Anzahl der Elemente
R	m	Abstand der Kamera
r	m	Radius
w	mm	Durchbiegung
$\overrightarrow{x}$	[mmm]	Ortsvektor
x_E	m	x-Koordinate des Endeffektors
x_G	m	x Koordinate im globalen Koordinaten System
x_K	Pixel	x Koordinate im Koordinatensystem der Kamera

x_M	m	x-Koordinate des Kugelmittelpunktes
x_P	m	x-Koordinate des Projektionsmittelpunktes
y_E	m	y-Koordinate des Endeffektors
y_G	m	y Koordinate im globalen Koordinaten System
y_K	Pixel	y Koordinate im Koordinatensystem der Kamera
z	m	Projektionsfehler
Δ	m	relative Auslenkung
λ	mm/Pixel	Umrechnungsfaktor

1 Einleitung

Als Alternative zu seriellen Robotern erlangen parallele Kinematiken eine immer größere Bedeutung in der Robotertechnik. Durch die parallele Anordnung der Gelenke und Achsen kann gegenüber einer seriellen Bauweise eine kompaktere und steifere Struktur geschaffen werden. Als Vorteile einer parallelen Anordnung sind die höhere Genauigkeit, ein besseres Verhältnis zwischen Nutzlast und Eigengewicht und eine höhere Dynamik durch geringere Eigenmassen, die bewegt werden müssen, zu nennen, da bei einem seriellen Roboter jeder Antrieb alle nachfolgenden Glieder und Gelenke mit bewegen muss. Als Nachteil einer parallelen Kinematik ist der geringere Arbeitsraum bezogen auf das Bauvolumen zu erwähnen. Die Nutzbarkeit des Arbeitsraumes kann durch auftretende Kraftsingularitäten weiter verkleinert werden. Eine Möglichkeit die Kraftsingularitäten teilweise zu kompensieren bietet eine redundante Kinematik. Als Redundant bezeichnet man eine Kinematik, wenn sie mehr Bewegungsfreiheitsgrade besitzt als notwendig sind, um Bewegungsvorgaben ausführen zu können. Durch eine Redundanz ist es unter anderem möglich das System mit den Antrieben vorzuspannen und damit so zu versteifen, dass eine höhere Genauigkeit erzielt werden kann.
In dem Institut für Mechanik und Mechatronik an der Hochschule für Angewandte Wissenschaften Hamburg sollen in dem Themenbereich „Redundante Roboter“ Untersuchungen durchgeführt werden. Zu diesem Thema wurde bereits in einem Projekt ein 3-RRR Roboter konstruiert. Für diesen Roboter wird in dieser Masterthesis eine Steuerung und Messmethode ausgewählt und konfiguriert und damit erste Untersuchungen an dem Roboter vorgenommen.

1.1 Aufgabenstellung

In dieser Masterthesis soll das Verhalten von parallelen redundanten Robotern mit flexiblen Elementen untersucht werden. Dies wird am Beispiel des planaren, parallelen und redundanten 3-RRR Roboters durchgeführt. Als flexible Elemente sollen für den Roboter elastische Arme konstruiert werden. Durch den elastischen Anteil ergibt sich bei einem Krafteinfluss eine Abweichung des Endeffektors. Um diese Abweichung am Roboter zu messen, ist eine geeignete Messtechnik auszuwählen und in Betrieb zu nehmen. Um an verschiedenen Punkten im Arbeitsraum zu messen, muss eine Steuerung ausgewählt werden, mit der der Roboter im Arbeitsraum in vorgegebenen Positionen gesteuert und dort dann arretiert werden kann. Die elastischen Arme sollen auch in einem Simulink/SimMechanics Modell für den Roboter nachgebildet werden. Anschließend sollen die Ergebnisse der Simulation mit den gemessenen Werten am Roboter verglichen werden.

Da die Regelung im dynamischen Bereich durch eine Vielzahl von dynamischen Effekten beeinflusst wird, soll im Rahmen dieser Arbeit eine Regelung für den statischen Fall entworfen werden. Diese Regelung soll die Position des Endeffektors korrigieren, die durch die elastischen Roboterarme beeinflusst wird.

1.2 Strukturierung der Arbeit

Dieser Abschnitt soll ein Überblick über den Inhalt und den Aufbau der Masterthesis geben. Zu Beginn wird im Kapitel 2 der Aufbau des 3-RRR Roboters mit den verwendeten Komponenten vorgestellt. Anschließend wird in Kapitel 3 die Dimensionierung des Armsegmentes mit einer elastischen Komponente erklärt. Als Referenz zum Roboter wird in Kapitel 4 ein Simulink/SimMechanics Modell vorgestellt mit dem das statische Verhalten des Roboters simuliert wird. Außerdem wird eine Korrekturmethode für eine statische Belastung vorgestellt. Die Inbetriebnahme und Konfiguration, sowie das Programm zur Steuerung des Roboters wird in Kapitel 5 beschrieben. Hierbei wurde das Kapitel 5.1 in Form einer Bedienungsanleitung geschrieben, damit eine spätere Inbetriebnahme anhand dieses Kapitels möglich ist. Die Messung der Position des Roboters mit Hilfe einer Kamera und die Umsetzung in Matlab wird in Kapitel 6 erklärt. Der Vergleich zwischen dem Verhalten in der Simulation und am Roboter wird im Kapitel 7 vorgenommen. Abschließend ist in Kapitel 8 ein Fazit und Ausblick formuliert.

2 Grundlagen

Im Folgenden werden die für den Betrieb des 3-RRR Roboters benötigten Komponenten und der Aufbau des Roboters mit den Notationen erklärt. Es werden die in der Masterthesis verwendete Bachmann Steuerung, die zu der Steuerung gehörige Software und das CANopen Kommunikationsprotokoll vorgestellt. Abschließend werden die eingesetzten FAULHABER Servomotoren und der verwendete FAULHABER Befehlssatz beschrieben.

2.1 3-RRR Roboter

Zuerst wird als Basis für die Arbeit der 3-RRR Roboter, der in dem Masterprojekt „Konstruktion, Entwicklung und Simulation eines parallelen, planaren und redundanten 3-RRR und 4-RRR Roboters“ [5] entwickelt und konstruiert wurde, vorgestellt.

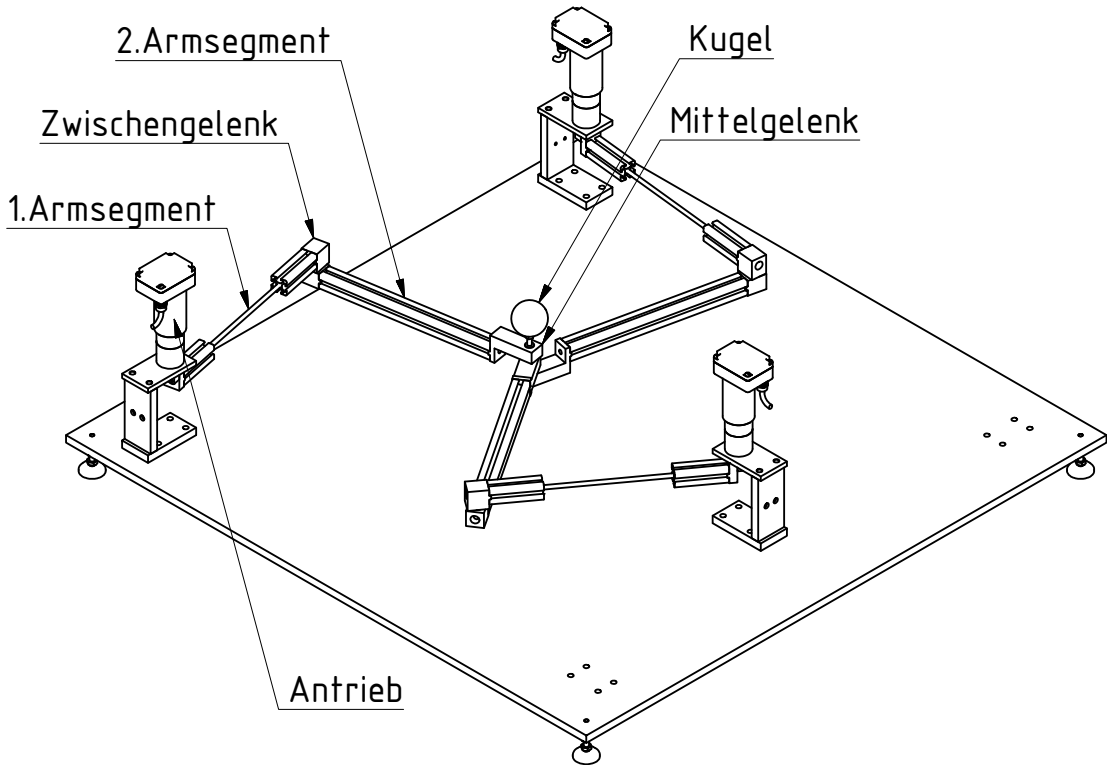


Abbildung 2.1: Aufbau des 3-RRR Roboters

In Abbildung 2.1 ist der Aufbau des 3-RRR Roboters abgebildet. Die Bezeichnung 3-RRR bedeutet, dass der Roboter aus drei Armen besteht mit jeweils drei Rotationsgelenken. Durch den Unterstrich wird gekennzeichnet, dass jeweils das erste Gelenk von einem Motor angetrieben wird. Das dritte Rotationsgelenk von jedem Arm ist das Mittelgelenk über den die drei Arme miteinander verbunden sind. Auf dem Mittelgelenk befindet sich auf dem Gelenkbolzen eine rote Kugel, diese dient der Bestimmung der Position mit einer Kamera. Für die Masterthesis wird an dieser Stelle die Notation der Winkel und Gelenke eingeführt. Dies wird in Abbildung 2.2 veranschaulicht. Die Punkte A_1, A_2 und A_3 zeigen die Positionen der drei Antriebe. Die Koordinaten der Antriebe sind in der Tabelle 2.1 aufgeführt. An den Positionen B_1, B_2 und B_3 befinden sich die Zwischengelenke. Die Position des Endeffektors ist mit EE gekennzeichnet. Die Winkel q_1, q_2 und q_3 der Motoren werden wie in der Abbildung 2.2 gegen den Uhrzeigersinn von der x-Achse aus angegeben. Bei den Winkeln ϑ_1, ϑ_2 und ϑ_3 der Zwischengelenke beginnen die Winkel an der Verlängerung des ersten Armsegmentes und werden ebenfalls gegen den Uhrzeigersinn angegeben. Die Länge der Armsegmente beträgt $L = 288\,\text{mm}$. Des Weiteren ist das globale Koordinatensystem KS *0* abgebildet.

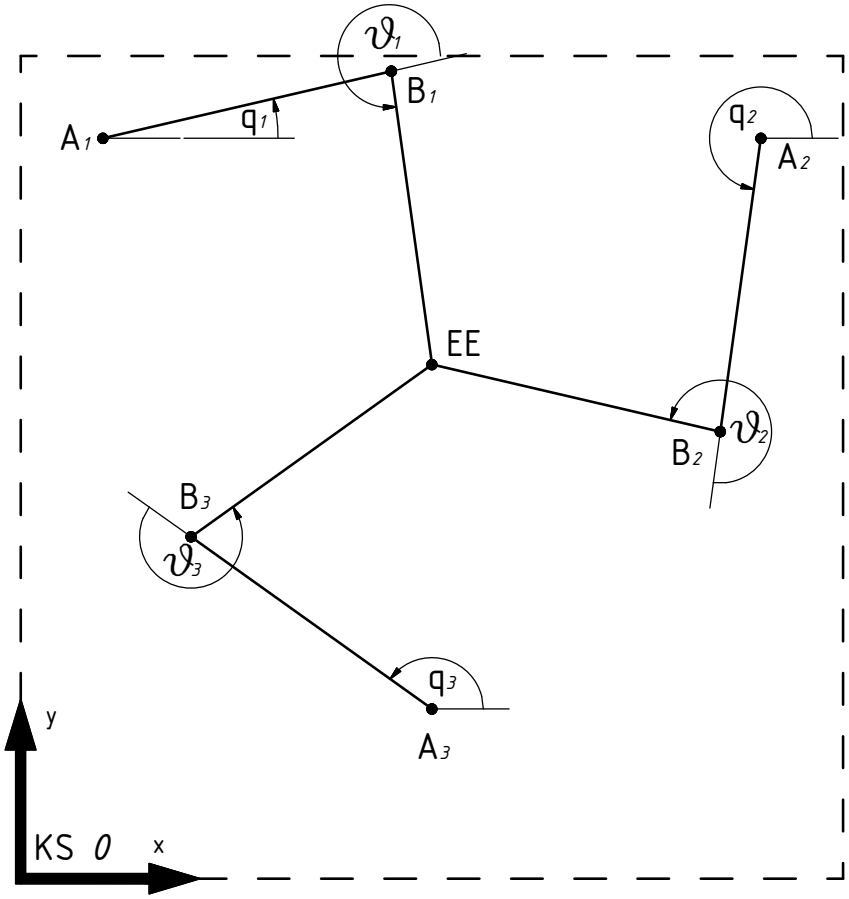


Abbildung 2.2: Notation der Winkel und Gelenke des 3-RRR Roboters

Tabelle 2.1: Position der Antriebe

Antrieb	Position	
	x [mm]	y [mm]
A_1	80	720
A_2	720	720
A_3	400	165,74

2.2 Bachmann Steuerung

Die Bachmann MX 220 wird zur Steuerung und Regelung der Motoren verwendet. Die Bachmann MX 220 ist eine speicherprogrammierbare Steuerung (SPS).

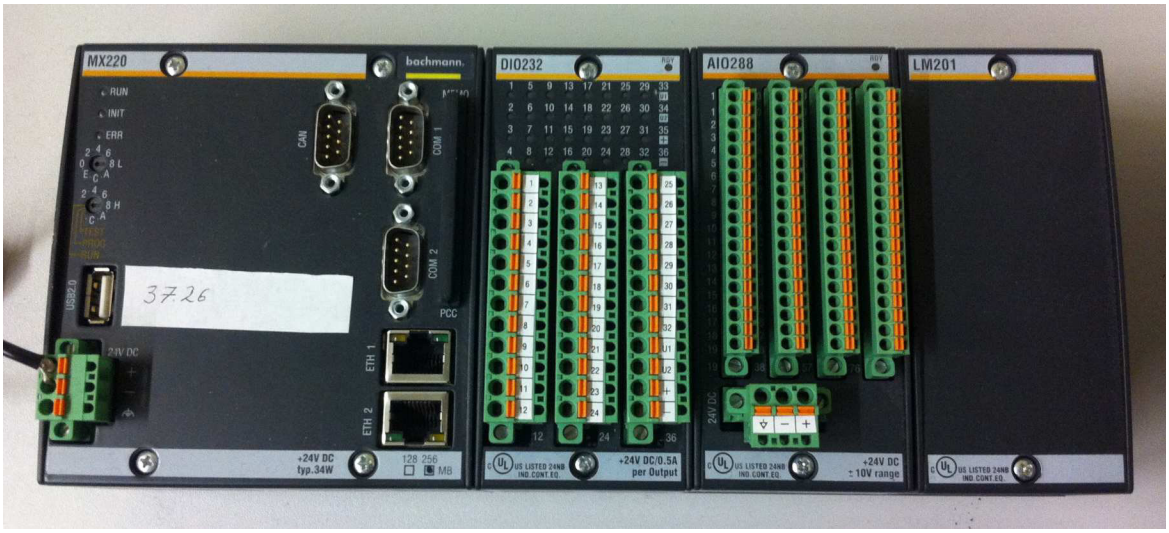


Abbildung 2.3: Bachmann MX 220 mit Erweiterungsmodulen
(v.l.n.r MX 220, AIO 288, DIO 232 und LM 201)

Für die MX 220 stehen verschiedene Erweiterungsmodule zur Verfügung mit denen die Steuerung um verschiedene Ein- und Ausgänge erweitert werden kann. Das MX 220 Prozessormodul verfügt über zwei Ethernet Anschlüsse, einen USB-Port, zwei serielle RS232 und einen CAN-Bus Anschluss. Als Erweiterungsmodule sind auf der Steuerung das AIO 288, DIO 232 und das LM 201 montiert. Durch das Erweiterungsmodul AIO 288 wurde die Steuerung um analoge Ein- und Ausgänge erweitert. Das Erweiterungsmodul DIO 232 bietet digitale Ein- und Ausgänge. Abschließend ist das LM 201 auf der Steuerung montiert, hierbei handelt es sich um ein Leermodul, welches keine weiteren Funktionen besitzt. In der

Masterthesis werden nur ein Ethernet Anschluss und der CAN-Bus Anschluss verwendet. Die Bachmann MX 220 lässt sich mit der „Solution Center Software“ von Bachmann konfigurieren und programmieren. Die Software bietet auch die Möglichkeit während des Betriebes die Prozessvariablen zu überwachen und die Priorität verschiedener Programme auf der Steuerung einzustellen. Die Einrichtung eines CANopen Netzwerkes ist ebenfalls mit der „Solution Center Software“ möglich.
Mit der MATLAB/Simulink Erweiterung „M-Target for Simulink“ bietet Bachmann die Möglichkeit Simulink Modelle, die mit einer festen Zykluszeit arbeiten, auf die Steuerung zu übertragen und dort eigenständig auszuführen. Die Erweiterung bietet auch die Möglichkeit sich in Echtzeit mit dem auf der Steuerung laufenden Programm zu verbinden, wodurch direkte Veränderungen am Programm vorgenommen werden können, wie z.B. das Verändern von Variablen.

2.3 CANopen

CANopen ist ein Kommunikationsprotokoll, welches auf CAN (Controller Area Network) basiert. Dieses benutzt die CAN-Bus Schnittstelle und ermöglicht einem Anwender die Inbetriebnahme eines CAN-Bus Systems ohne nähere Kenntnisse. An so einem CANopen-Netzwerk können bis zu 127 verschiedene Teilnehmer angeschlossen werden. Eine besondere Eigenschaft von CANopen sind die elektronischen Datenblätter, die sogenannten EDS-Dateien (Electronic Datasheets), in diesen stehen die Eigenschaften der Geräte, die an einem CANopen-Netzwerk teilnehmen. Die EDS-Datei legt die Struktur der Kommunikation eines Teilnehmers fest. Zur Kommunikation der Teilnehmer untereinander dienen die Prozessdatenobjekte (PDOs). Hierbei unterscheidet man zwischen Empfangs-PDOs (RxPDO) und Sende-PDOS (TxPDO). Die Kennzeichnung der PDOs erfolgt aus der Funktion des Gerätes. So wird z.B. ein Sende-PDO eines Motors einem Empfangs-PDO der Steuerung zugeordnet. In Kapitel 2.4 werden die Prozessdatenobjekte der FAULHABER Servomotoren beschrieben.

2.4 FAULHABER DC-Servomotor

Der FAULHABER 3564K024B CC ist ein bürstenloser Servomotor mit integrierter Antriebssteuerung. An den Servomotoren ist jeweils ein Planetengetriebe mit einer Übersetzung von 66:1 angebaut. Die Konfiguration der Servomotoren erfolgt über die Motion Manager 4 Software der Firma FAULHABER. Eine Möglichkeit die Motoren zu steuern besteht in der Nutzung des FAULHABER-Befehlssatzes. Dabei erfolgt die Echtzeitkommunikation der Servomotoren mit der Steuerung über sechs Prozessdatenobjekte (PDO), wobei drei PDOs zum Empfangen und drei PDOs zum Senden von Daten dienen. Der Inhalt der PDOs ist von der eingestellten Betriebsart abhängig. In dieser Masterthesis wird der FAULHABER-Betriebsmodus gewählt. Der Inhalt der PDOs ergibt sich dadurch wie folgt:

- **Empfangs PDO 1 Controlword**
  Das Controlword dient der Steuerung des Motorstatus. Für die Kommunikation im FAULHABER-Betriebsmodus ist dieses PDO nicht erforderlich und wird daher im Rahmen dieser Mastherthesis nicht verwendet.

- **Sende PDO 1 Statusword**
  Über das Statusword werden Informationen über den aktuellen Motorstatus gesendet. Die Zuordnungen der jeweiligen Statusbits im Statusword sind in Tabelle 2.2 zu sehen. Das Statusword ist für diese Masterthesis nicht erforderlich.

Tabelle 2.2: FAULHABER-Statusword[4]

Bit	Funktion
0	Ready to Switch On
1	Switched On
2	Operation Enabled
3	Fault
4	Voltage Enabled
5	Quick Stop
6	Switch On Disabled
7	Warning
8	0
9	Remote
10	Target Reached
11	Internal limit active
12	Set-point acknowledge/ Speed / Homing attained
13	Homing Error
14	Hard Notify
15	0

- **Empfangs PDO 2 FAULHABER Kommando**
  Mit diesem PDO werden die FAULHABER-Befehle an den Motor gesendet. Die in dieser Masterthesis verwendeten Befehle sind in Tabelle 2.3 aufgeführt. Ein FAULHABER-Befehlssatz setzt sich aus einem Befehl und einem Argument zusammen.

Tabelle 2.3: FAULHABER-Befehle[4]

Befehl	Dezimal	Argument	Beschreibung
EN	15	-	Antrieb aktivieren
DI	8	-	Antrieb deaktivieren
M	60	-	Lageregelung aktivieren und Positionierung starten
LA	180	Positionswert	Neue absolute Sollposition laden
V	147	Motordrehzahl	Drehzahlregelung mit angegebenem Wert starten
HO	184	Positionswert	Definiert die aktuelle Positon mit dem angegeben Wert
GRC	52	-	Aktueller Motorstrom in mA

- **Sende PDO 2 FAULHABER Abfragedaten**
  Über diese PDO antwortet der Motor auf die über das Empfangs PDO 2 empfangenen Befehle. Der Motor antwortet mit dem empfangenen Befehl, einem Argument und einem Fehlercode. Das Argument stellt den angeforderten Wert da, wie z.B der aktuelle Motorstrom. Wenn kein Wert angefordert wird, enthält das Argument den Wert des empfangenen Argumentes. Über den Fehlercode sendet der Motor eine Bestätigung über die Ausführung des empfangenen Befehls.

- **Empfangs PDO 3 Trace Konfiguration**
  Über das Empfangs PDO 3 wird die Konfiguration des Trace eingestellt. Über das Trace Daten PDO können zwei Motorvariablen an die Steuerung übertragen werden. Das Trace Konfigurations PDO besitzt eine Länge von fünf Bytes. Die Einstellungsmöglichkeiten der fünf Bytes sind:
  - **Byte „0“**
    Über das „0“te Byte wird die erste Motorvariable festgelegt, die über das Trace gesendet werden soll. Die Einstellmöglichkeiten sind in Tabelle 2.4 dargestellt. Als erste Variable wird in der Masterthesis die Ist-Position gewählt.

Tabelle 2.4: FAULHABER-Tracevariablen[4]

Parameter	Motorvariable	Datentyp	Einheit
0	Ist-Drehzahl	Integer 16	rpm
1	Soll-Drehzahl	Integer 16	rpm
4	Motorstrom	Integer 16	mA
44	Gehäusetemperatur	Unsigned 16	°C
46	Spulentemperatur	Unsigned 16	°C
200	Ist-Position	Integer 32	Inkremente
201	Soll-Position	Integer 32	Inkremente

  - **Byte „1“**
    In dem „1“ten Byte wird die zweite Motorvariable festgelegt. Die Einstellung erfolgt ebenfalls über die Parameter in der Tabelle 2.4. Der Datentyp auf der Steuerung ist auf „Integer 32“ festgelegt, deshalb konnte der Motorstrom mit dem Datentyp „Integer 16“ nicht gewählt werden. Zur Auffüllung des Datensatzes wurde hier stattdessen die Soll-Position gewählt. Der Motorstrom wir später über eine Befehlsabfrage erfasst.
  - **Byte „2“**
    Das „2“te Byte legt fest ob ein „Timecode“ mit übertragen werden soll. In der Masterthesis wird das Byte auf „0“ gesetzt, wodurch kein „Timecode“ gesendet wird.
  - **Byte „3“**
    Das „3“te Byte legt die Anzahl der zu übertragenden Trace Daten im PDO 3 nach einem Request oder SYNC-Telegram fest. In dieser Masterthesis soll ein Trace Daten PDO nach dem SYNC Telegramm gesendet werden, also wird in dem Byte der Wert „1“ eingetragen.
  - **Byte „4“**
    In dem „4“ten Byte wird die Wartezeit zwischen zwei aufeinander folgenden Trace Daten PDOs festgelegt. Bei einer Übertragung von nur einem Trace PDO pro SYNC-Telegramm ist diese Einstellung wirkungslos.

- **Sende PDO 3 Trace Daten**
  In dem Trace Daten PDO werden je nach Konfiguration über das Trace Konfigurations PDO zwei Motorvariablen und gegebenenfalls ein „Timecode“ gesendet.

3 Konstruktion des elastischen Armsegmentes

Gegen über dem Originalroboter aus dem Masterprojekt [5] wurden die ersten Armsegmente neu konstruiert. Die Steifigkeit der ersten Armsegmente sollte verringert werden. Dies wurde dadurch erreicht, dass in dem mittleren Abschnitt des Armsegmentes ein Teil des Aluminiumprofiles durch einen Rundstab aus Stahl mit 5 mm Durchmesser ersetzt wurde. Die Abmessungen der Armsegmente, die in Abbildung 3.1 zu sehen sind, wurden über eine Vordimensionierung bestimmt. Die Fertigungszeichnungen der neuen Komponenten befinden sich im Anhang A.14.

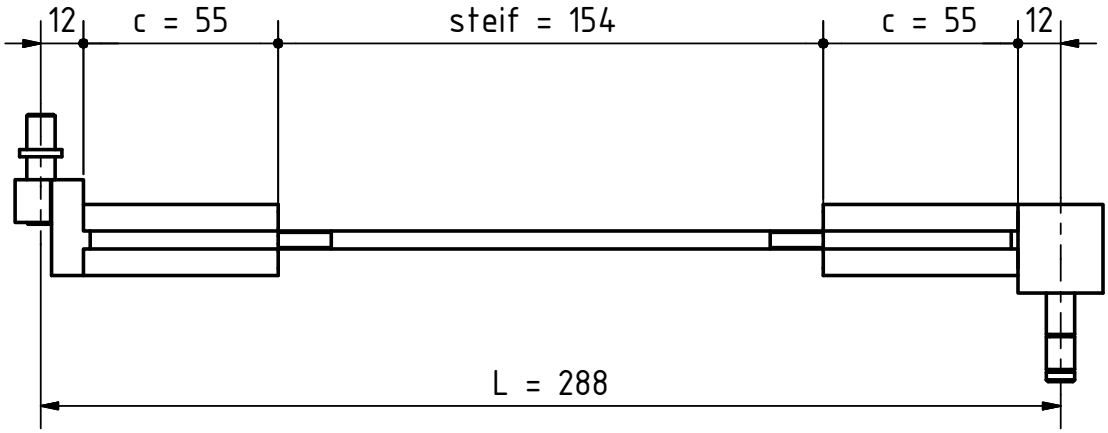


Abbildung 3.1: Armsegment mit elastischem Anteil

Die Vordimensionierung erfolgt über das Verhalten des 3-RRR Roboters mit Elastomerkupplungen. Mit dem m-file „Kinematikanalyse_3RRR_oM“ aus dem Masterprojekt [5] kann die Auslenkung des Endeffektors unter Krafteinfluss berechnet werden. In diesem Modell besitzen die Antriebe Elastomerkupplungen mit einer Torsionssteifigkeit k. Die maximale Auslenkung soll ungefähr 20 mm bei einer Kraft von 19,62 N entsprechen. Aus dem m-file ergibt sich diese Auslenkung bei einer Steifigkeit von $k = 150\,\mathrm{Nm/rad}$. Zur Bestimmung der Abmaße der Armsegmente dienen die jeweiligen Auslenkungen der beiden Modelle. Beim ersten Modell wird das neue Armsegment als Kragarm (Abb. 3.2 oben) modelliert. Das zweite Modell besteht aus einem starren Armsegment, das auf der einen Seite mit einer Drehfeder mit einer Torsionssteifigkeit von $k = 150\,\mathrm{Nm/rad}$ drehbar gelagert ist (Abb. 3.2 unten). Beide Modelle werden an der Spitze mit der Kraft F belastet.

Nun werden die Abmaße des Armsegmentes so gewählt, dass die Auslenkungen w_1 und w_2 der Modelle ungefähr gleich groß sind.

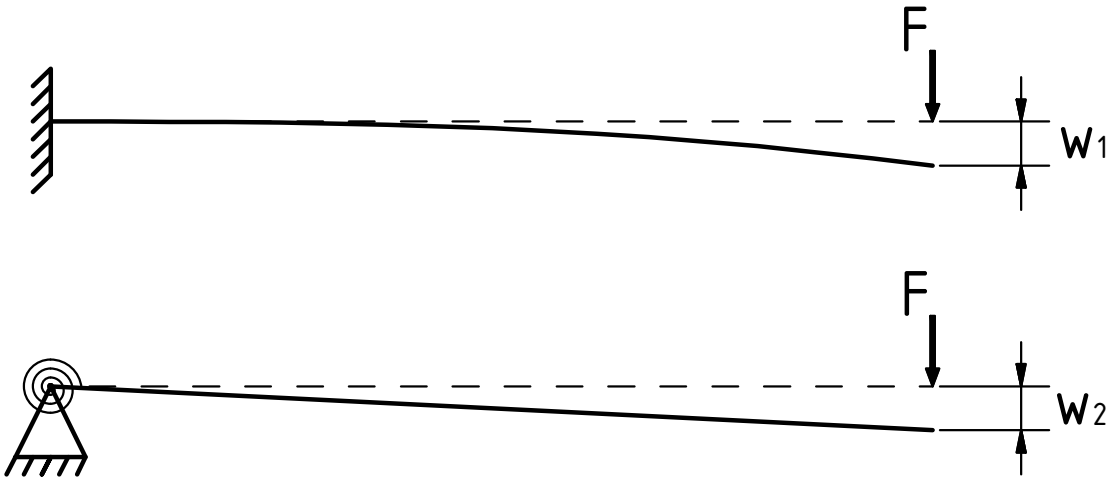


Abbildung 3.2: Kragarm und Arm mit Drehfeder

Für das Kragarmmodell wurde ein FE-Modell mit den entsprechenden Steifigkeiten in Matlab erstellt. Das FE-Modell steht in dem m-file „FEModell_Kragarm.m“. Der Quellcode befindet sich im Anhang A.6. Der Aufbau wird im Folgenden erklärt.
Zu Beginn werden die Parameter des Modells angegeben. Die Parameter bestehen aus der Länge L des Armsegmentes, die Länge c des Aluminiumprofils, der Radius r des Rundstabes und die Anzahl n der Elemente. In Abbildung 3.1 ist die Geometrie des Armsegmentes dargestellt. Anschließend wird die Länge l der Elemente und die Anzahl der Freiheitsgrade dof berechnet. Mit der Kraft F wird die Belastung des Kragarmes festgelegt.

```
% Parameter:
L  = 288 ;          % Länge des Armsegmentes [mm]
c  = 55 ;           % Länge des Aluminium Profils [mm]
r = 2.5;            % Radius des Rundstabes [mm]
n = 288;            % Anzahl der Elemente
l = L/n;            % Länge der Elemente [mm]
dof = 2*(n+1);      % Anzahl Freiheitsgrade global
F = 1;              % Kraft an der Spitze [N]
```

Danach werden die Übergänge der Abschnitte des Armsegmentes festgelegt. Sie werden einmal in der Einheit mm und einmal nach der Anzahl der Elemente angegeben.

```
Abschnitte_mm = [12 12+c L-c-12 L-12 L];
Abschnitte_element = Abschnitte_mm/l;
```

In den nächsten Zeilen werden den Abschnitten der jeweilige Elastizitätsmodul zugewiesen. Die Abschnitte eins, drei und fünf bestehen aus Stahl mit einem Elastizitätsmodul von 210 GPa. Der Elastizitätsmodul des Aluminiumprofils beträgt 70 GPa.

```
% Elastizitatsmodule
E(1) = 210e3;        %[MPa]
E(2) = 70e3;         %[MPa]
E(3) = 210e3;        %[MPa]
E(4) = E(2);
E(5) = E(1);
```

In dem nächsten Abschnitt werden die Flächenträgheitsmomente der Abschnitte nach [6] berechnet. Das Flächenträgheitsmoment für den ersten und letzten Abschnitt wird nach der Gleichung 3.1 mit $h = 20\,\text{mm}$ berechnet.

$$I_{1/2} = \frac{h^4}{12} \tag{3.1}$$

Für die Abschnitte zwei und vier wird das Flächenträgheitsmoment des Aluminiumprofils mit $I_{2/3} = 7200\,\text{mm}^4$ dem Datenblatt des Herstellers entnommen. Dies befindet sich im Anhang A.2. Der dritte Abschnitt ist der Rundstab, dessen Flächenträgheitsmoment sich nach der Gleichung 3.2 mit einem Radius $r = 2{,}5\,\text{mm}$ berechnet.

$$I_{2/3} = \frac{\pi}{4} \cdot r^4 \tag{3.2}$$

```
% Flächenträgheitsmomente
I(1) = 20*20^3/12;  %[mm^4]
I(2) = 0.72*10^4;   %[mm^4]
I(3) = pi/4*r^4;    %[mm^4]
I(4) = I(2);
I(5) = I(1);
```

Als nächstes werden die Gesamtsteifigkeitsmatrix K, der Vektor der Knotenverformung u und der Vektor der Knotenkräfte f initialisiert.

```
% Initialisieren
K = zeros(dof,dof);    % Gesamtsteifigkeitsmatrix
u = zeros(dof,1);      % Vektor der Knotenverformungen
f = zeros(dof,1);      % Vektor der Knotenkraefte
```

In dem nächsten Abschnitt werden die Randbedingungen des Modells festgelegt. Bei dem Kragarm sind die ersten beiden Freiheitsgrade blockiert.

```
% Randbedingungen
ufree =  (3 : dof) ;   % Welche Freiheitsgrade frei sind
```

Die Belastung des Modells wird in dem Vektor f festgelegt. Da das Modell an der Spitze mit der Kraft F belastet wird, wird an der vorletzten Stelle des Vektors die Kraft eingetragen.

```
% Vektor der Knotenkraefte (f-Vektor)
f(dof-1) = F;
```

Mittels der „for“-Schleife wird die Gesamtsteifigkeitsmatrix K mit den Steifigkeitsmatrizen der einzelnen Balkenelemente erstellt. Die Steifigkeitsmatrizen der Balkenelemente werden durch die Funktion *esm* mit der Steifigkeit EI und der Länge l der Elemente berechnet. Die Zusammensetzung der Gesamtsteifigkeitsmatrix K und der Steifigkeitsmatrizen der Balken werden, wie in [7] beschrieben, erstellt.

```
% Aufstellen der Gesamtsteifigkeitsmatrix K
for i= 1   : n
    if i ≤ Abschnitte_element(5)
        EI(i) = E(5)*I(5);
    end
    if i ≤ Abschnitte_element(4)
        EI(i) = E(4)*I(4);
    end
    if i ≤ Abschnitte_element(3)
        EI(i) = E(3)*I(3);
    end
    if i ≤ Abschnitte_element(2)
        EI(i) = E(2)*I(2);
    end
    if i ≤ Abschnitte_element(1)
        EI(i) = E(1)*I(1);
    end
    inz = (i*2-1:i*2-1+3);
    K(inz,inz) = K(inz,inz) + esm(EI(i),l);
end
```

Zur Lösung des Gleichungssystems $Ku = f$ werden die Gesamtsteifigkeitsmatrix K und der Vektor der Knotenkräfte f reduziert und anschließend von links mit der Inversen der Gesamtsteifigkeitsmatrix multipliziert und somit die freien Verformungen berechnet.

```
% Loesung des reduzierten Gleichungssystems  Ku = f
Kred = K(ufree,ufree);
fred = f(ufree);
ured = inv(Kred)*fred;
```

Daraufhin werden die berechneten Verformungen in den Vektor der Verformungen u eingetragen und die Durchbiegung w an der Spitze des Kragarmes ausgegeben.

```
% Berechnung der unbekannten Knotenkraefte aus  f = Ku
u(ufree) = ured; % Vektor u wieder auf volle Groesse setzen
% Ausgabe der Durchbiegung
w = u(end-1)
```

Bei einer Belastung mit der Kraft $F = 1\,\mathrm{N}$ ergibt sich eine Durchbiegung von $w_1 = 0{,}5502\,\mathrm{mm}$. Die Durchbiegung w_2 wird nach Gleichung 3.3 mit einer Armsegmentlänge von $L = 288\,\mathrm{mm}$ berechnet.

$$w_2 = L \cdot tan\left(\frac{F \cdot L}{k}\right) = 0{,}553\,\mathrm{mm} \approx w_1 \tag{3.3}$$

Somit besitzt das konstruierte Armsegment die angestrebte Steifigkeit. Nach der Fertigung der Armsegmente wurde die Steifigkeit überprüft. Hierfür wurde ein Armsegment einseitig eingespannt und auf der anderen Seite mit einem 2 kg Gewicht belastet. Anschließend wurde die Auslenkung an der belasteten Stelle gemessen. Die Auslenkung des Armsegmentes beträgt $w_A = 21{,}1\,\mathrm{mm}$. In dem FE-Modell wurde bei einer Kraft $F = 19{,}62\,\mathrm{N}$ eine Auslenkung von $w_{S1} = 10{,}795\,\mathrm{mm}$ berechnet. Hieraus ist zu erkennen, dass das FE-Modell 1,96 mal steifer ist als das gefertigte Armsegment. Die Abweichung zwischen FE-Modell und der Messung kann verschiedene Ursachen haben. Eine Ursache kann darin liegen, dass für den Rundstab ein Material mit geringerem Elastizitätsmodul verwendet wurde. Um dies zu überprüfen, wurde nur der Rundstab belastet und die Auslenkung mit der analytischen Lösung verglichen. Diese Messung ergab eine größere Auslenkung als in der Berechnung. Daraus lässt sich schließen, dass das Material ein geringerer Elastizitätsmodul besitzt. Des Weiteren besteht das Armsegment aus mehreren verschraubten Elementen. Die Übergänge zu den einzelnen Elementen haben auch einen Einfluss auf die Steifigkeit des Armsegmentes. Der Unterschied zwischen dem FE-Modell und dem gefertigten Armsegment scheint eine Kombination aus den genannten Gründen. Um nun das FE-Modell dem gefertigtem Armsegment anzupassen, wird der verwendete Elastizitätsmodul des Rundstabes durch 1,96 geteilt. Dadurch ergibt sich eine Auslenkung von $w_{S2} = 21{,}02\,\mathrm{mm}$. Damit wurde das FE-Modell dem gefertigten Armsegment angepasst. Diese Anpassung wird in Kapitel 4.3 verwendet. Da die maximale Auslenkung des Roboters bei einer Kraft von 19,62 N nun größer ist als 20 mm wird die maximale Kraft auf 9,81 N reduziert.

4 Simulink/SimMechanics

Für den Vergleich der an dem Roboter gemessenen Auslenkungen wurde in „Simulink/SimMechanics“ ein Modell des redundanten 3-RRR Roboters erstellt. „Simulink/SimMechanics“ bietet die Möglichkeiten Modelle von starren Mehrkörpersystemen zu erstellen. Für den Roboter wurde ein solches Modell bereits in dem Masterprojekt [5] erstellt. In dieser Masterthesis wurden die ersten Armsegmente des Modells durch Armsegmente mit elastischen Anteilen ersetzt. Das Modell des Roboters wird im Kapitel 4.1 und die Einarbeitung des elastischen Anteils wird im Kapitel 4.2 erklärt.

4.1 SimMechanics Modell

Zu dem Simulationsmodell des 3-RRR Roboters gehören das in „Simulink/SimMechanics“ erstellte Modell, welches in Abbildung 4.1 zu sehen ist und eine Initialisierungsdatei „Sim_Dreiarm_oM_kor_INIT.m“. Zunächst wird der Aufbau des „Simulink/SimMechanics“ Modells beschrieben.

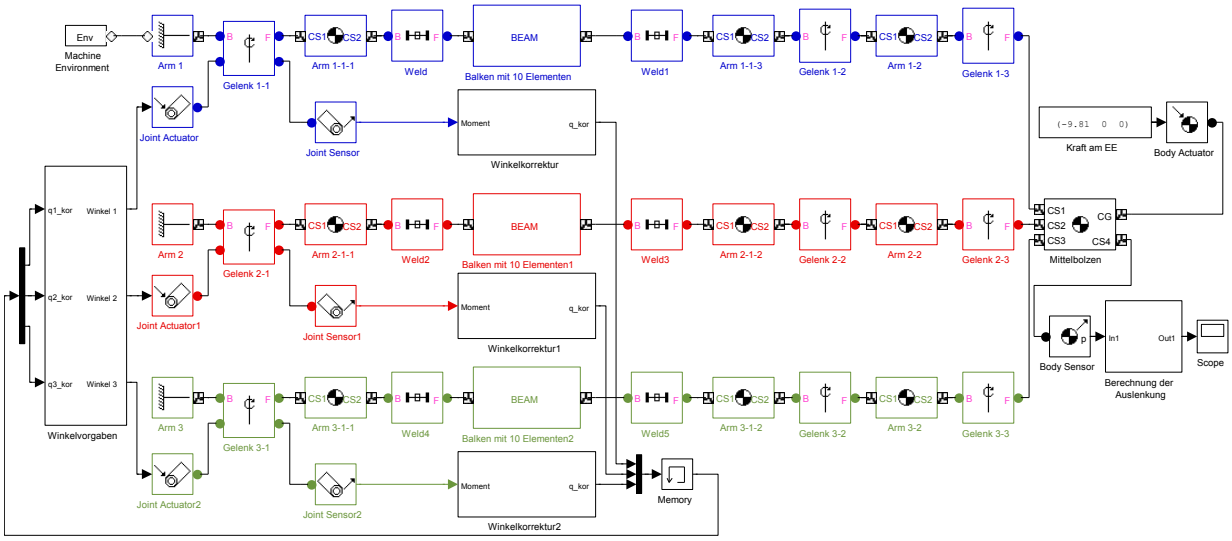


Abbildung 4.1: SimMechanics Modell des 3-RRR Roboters

Die drei Arme des Roboters sind jeweils farblich gekennzeichnet. Jeder Arm besteht aus zwei Armsegmenten. Die ersten Armsegmente sind jeweils über einen „Revolute“-Joint,

welche die Antriebsmotoren darstellen, mit einem „Ground“-Element fixiert. Des Weiteren bestehen die ersten Armsegmente aus einer Kombination von zwei steifen „Body“ Elementen und einem elastischen Element, welches in Kapitel 4.2 genauer beschrieben wird. Die zweiten Armsegmente sind mit einem „Revolute“ Joint an den ersten Armsegmenten angeschlossen und bestehen aus steifen „Body“ Elementen. Abschließend sind alle drei Arme mit „Revolute“-Joints über ein „Body“ Element miteinander verbunden, welches den Bolzen des Mittelgelenkes und damit die Position des Endeffektors darstellt. An diesem Bolzen wird über einen „Body Actuator“ die Kraft aufgebracht mit der der Endeffektor belastet wird, über einen „Body Sensor“ wird die Position des Endeffektors ausgelesen. In dem Subsystem „Berechnung der Auslenkung“ wird die normierte Differenz zwischen der Soll- und Ist-Position des Endeffektors berechnet.

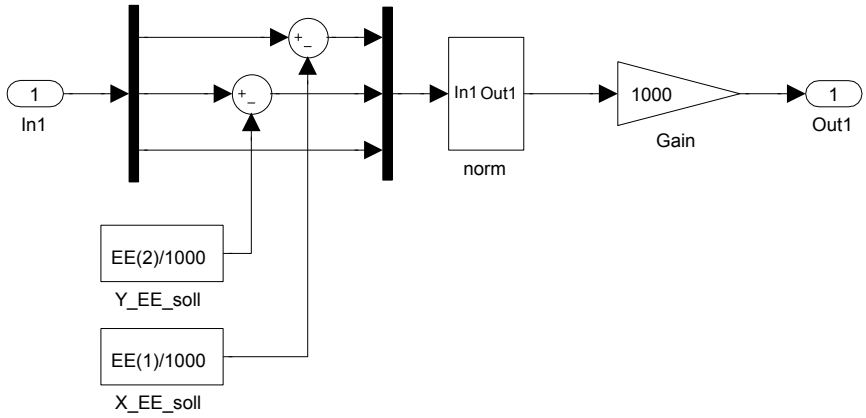


Abbildung 4.2: Subsystem: Berechnung der Auslenkung

Die Berechnung des in Abbildung 4.2 gezeigten Subsystems entspricht folgender Gleichung:

$$\Delta \;=\; |\overrightarrow{x}_{Ist} - \overrightarrow{x}_{Soll}| \cdot 1000\,\frac{\mathrm{mm}}{\mathrm{m}} \tag{4.1}$$

Um später die Abweichung des Endeffektors, die durch die Krafteinwirkung entsteht, von der Soll-Position zu korrigieren, wird an den Antriebsmotoren mit einem „Joint Sensor“ das Moment gemessen. Mit Hilfe des Momentes wird in dem Subsystem „Winkelkorrektur“ die nachzustellende Winkeldifferenz errechnet und über das Subsystem „Winkelvorgabe“ und einem „Joint Actuator“ an die Motoren weiter geleitet. Dieser Vorgang wird in Kapitel 4.3 näher beschrieben.
Als nächstes wird die Initialisierungsdatei „Sim_Dreiarm_oM_kor_INIT.m“ beschrieben, die sich im Anhang A.5 befindet. In dieser werden die Parameter, die für das Simulationsmodell benötigt werden, initialisiert.

Als erstes werden der Zeitschritt *dt*, sowie die Dauer *tend* der Simulation festgelegt. Der Wert *tkor* legt den Zeitpunkt der Korrektur fest und wird im Abschnitt 4.3 beschrieben. Mit dem Wert EE wird die Position des Endeffektors angegeben.

```
%% Eingangswerte
dt = 0.01 ;                          % [s]
tend = 100 ;                         % [s]
tkor = 50 ;                          % [s]
EE = [300 650];                      % x[mm] y[mm]
```

In dem nächsten Abschnitt werden die Befestigungsorte A_1, A_2 und A_3 der drei Aktoren anhand der Geometrie der Grundplatte berechnet. Diese befinden sich in der Tabelle 2.1.

```
% Bohrungsabstände Grundplatte
gp = 800;                            % [mm] Abmaße der Grundplatte
a = 640;                             % [mm] Abstand der Aktoren 1u2
a_aussen = (gp-a)/2;                 % [mm] Abstand der Aktoren 1u2 zur Kante
a_ver = a*sind(60);                  % [mm] y Abstand Aktor 3 zu 1u2

% Befestigungsorte der Aktoren
A1 = [a_aussen gp-a_aussen]';        % x[mm] y[mm]
A2 = [gp-a_aussen gp-a_aussen]';     % x[mm] y[mm]
A3 = [gp/2 gp-a_aussen-a_ver]';      % x[mm] y[mm]
```

Als nächstes wird die Länge der Arme *L_arm* und die Länge des steifen Bereichs *steif* der ersten Armsegmente festgelegt. Diese sind in Abbildung 3.1 dargestellt.

```
% Parameter für die Arme
L_arm = 288 ;                        % [mm]
c  = 55 ;                            % [mm]
steif = c + 12;                      % [mm]
```

Mit der Funktion *inv_Kin_3RRR_oM* werden die Winkel der Gelenke berechnet. Dies erfolgt nach den folgenden Gleichungen 4.2 und 4.3 aus dem Masterprojekt [5].

$$\vartheta_i = \arccos\left(\frac{x_E^2 + y_E^2 - 2L^2}{2L^2}\right) \tag{4.2}$$

$$q_i = \arccos\left(\frac{y_E\left(1+\cos\left(\vartheta_i\right)\right) - x_E \sin\left(\vartheta_i\right)}{x_E\left(1+\cos\left(\vartheta_i\right)\right) + y_E \sin\left(\vartheta_i\right)}\right) \tag{4.3}$$

Der Quellcode der Funktion befindet sich im Anhang A.7.

```
% Berechnung der Gelenkwinkel
[q theta] = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,EE(1),EE(2));
```

Zum Schluss wird für die Korrekturmethode bei Krafteinfluss der benötigte Faktor *kk* mit der Funktion *FEModell* berechnet. Die Funktion *FEModell* und der Faktor *kk* werden in dem Kapitel 4.3 beschrieben.

```
%% FEM-Modell des elastischen Arms
kk = FEModell();
```

4.2 Erstellung des elastischen Anteils

Für das „Simulink/SimMechanics“ Modell soll ein elastisches Element in die ersten Armsegmente integriert werden. Hierfür wurde das Armsegment durch Einfügen eines Rundstabes in der Mitte als elastischer Anteil hinzugefügt (Abb. 3.1). In „Simulink/SimMechanics“ wird nur der Rundstab elastisch modelliert, da die Steifigkeit EI hier deutlich kleiner ist als die Steifigkeit im übrigen Armsegment.
Eine Methode elastische Elemente zu modellieren ist in dem Artikel „Modeling Flexible Bodies in SimMechanics“ [2] beschrieben. Dabei wird mit Hilfe von Gelenk-, Feder- und Dämpfungselementen ein elastisches Verhalten nach der Balkentheorie erzeugt. Hierfür wird der elastische Bereich mit der Länge L und Masse M in n gleiche Elemente der Länge $l = L/n$ und Masse $m = M/n$ unterteilt. Dies ist in der Abbildung 4.3 dargestellt.

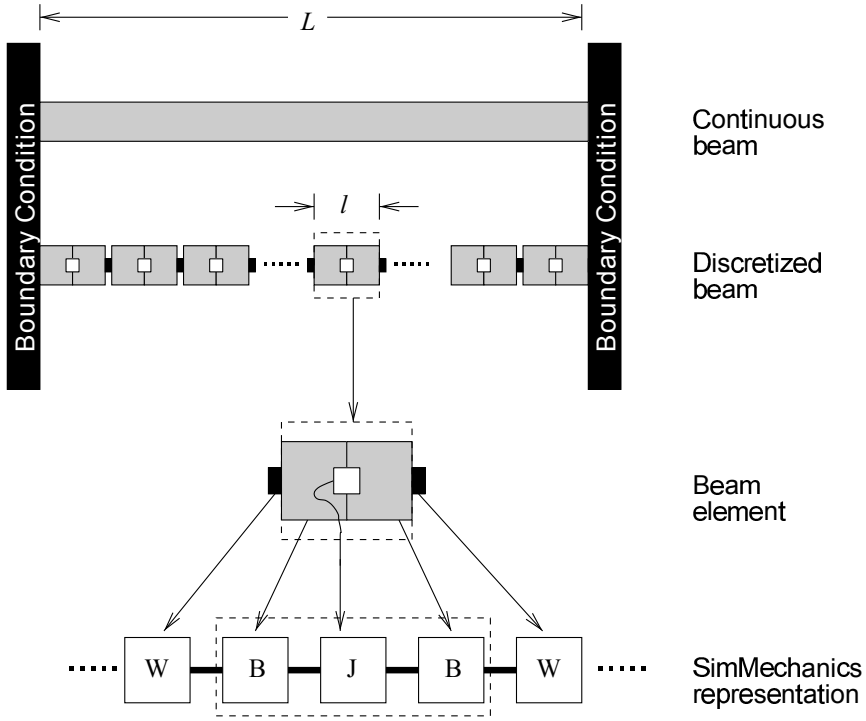


Abbildung 4.3: Aufbau elastischer Balken mit Welds (W), Bodies (B) und Joints (J) [2]

Jedes dieser Elemente besteht aus einer Kombination von „Body-Joint-Body“ Elementen, welche untereinander mit „Weld“ Elementen verbunden sind. Die elastischen Eigenschaften werden durch Feder- und Dämpfungselemente am „Joint“ eingebracht. Da jedes Element

aus zwei „Bodys“ besteht, ergeben sich für die Länge der „Bodys“ $l_B = l/2$ und für die Masse $m_B = m/2$. Die Massenträgheitsmomente J werden anhand der Geometrie und der Masse der „Bodys“ berechnet. In diesem Fall handelt es sich um einen Rundstab mit einem Radius $r = 2{,}5\,\text{mm}$. Daraus folgt nach [6] für das Massenträgheitsmoment J_1 um die Symmetrieachse:

$$J_1 = \frac{1}{2} m_B \cdot r^2 \tag{4.4}$$

Die Massenträgheitsmomente $J_{2/3}$ um die Querachsen lauten:

$$J_{2/3} = \frac{1}{4} m_B \cdot r^2 + \frac{1}{12} m_B \cdot l_B^2 \tag{4.5}$$

Durch die Steifigkeit k des „Joints“ entsteht die Elastizität in diesem Bereich, diese ergibt sich nach [2] zu:

$$k = \frac{EI}{l_B} \tag{4.6}$$

Hierfür wird der Elastizitätsmodul E des Materials und das Flächenträgheitsmoment I benötigt, welches nach [6] für einen Rundstabes wie folgt berechnet wird:

$$I = \frac{\pi}{4} \cdot r^4 \tag{4.7}$$

Für die Umsetzung in „Simulink/SimMechanics“ wurde ein Subsystem erstellt, mit dem es möglich ist durch Parametrisierung einen beliebigen elastischen Balken zu modellieren. In diesem Subsystem sind $n = 10$ Elemente verbaut, siehe Abbildung 4.4.

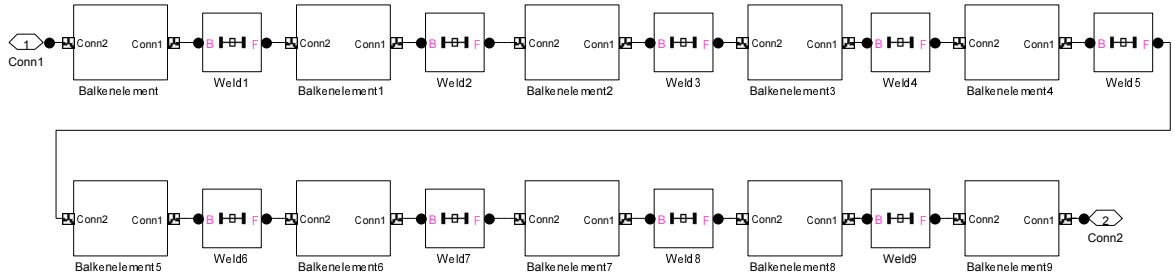


Abbildung 4.4: Aufbau des Balken-Subsystems

In dem Subsystem sind 10 Balkenelemente-Subsysteme enthalten, deren Aufbau wie oben beschrieben aus einer Kombination von „Body-Joint-Body“ Elementen besteht, siehe Abbildung 4.5. Als „Joint“ wird hier ein „Revolute“ Joint verwendet. An diesem wird ein „Joint Spring & Damper“ angebracht, mit dem die elastischen Eigenschaften an den „Joint“ übertragen werden.

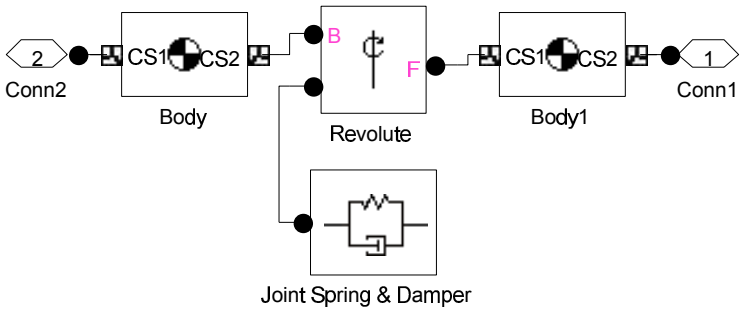


Abbildung 4.5: Aufbau des Balkenelement-Subsystems

Die für die Beschreibung der Eigenschaften der in dem Subsystem verwendeten Elemente benötigten Parameter werden über eine Maske eingegeben. Diese Maske unterteilt sich in drei Reiter: „Material“, „Geometrie“ und „Position“. In dem ersten Reiter „Material“ (Abb. 4.6) werden der Elastizitätsmodul, die Dichte und der Dämpfungsparameter eingegeben. Der Elastizitätsmodul wird durch den Korrekturfaktor 1,96 aus dem Kapitel 3 geteilt, damit das SimMechanics Modell den statischen Eigenschaften der gefertigten Armsegmente entspricht.

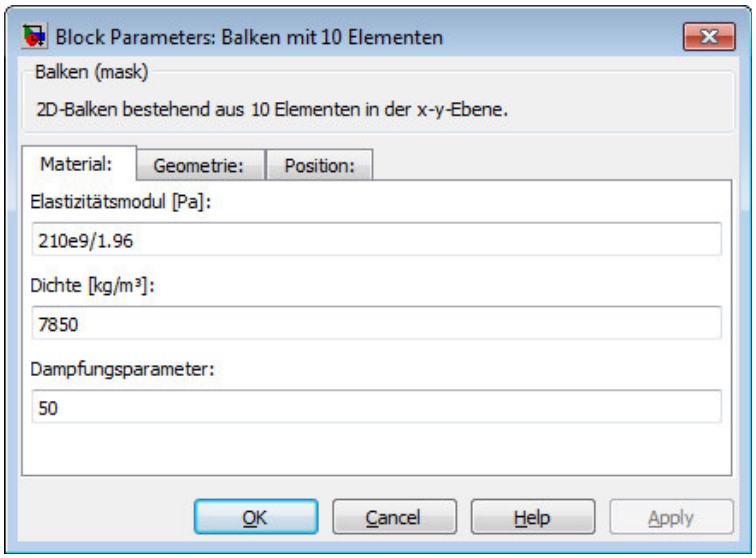


Abbildung 4.6: Maske des Subsystems: Material

In dem Reiter „Geometrie“ (Abb. 4.7) werden die Länge, die Fläche und das Flächenträgheitsmoment des Balkens angegeben.

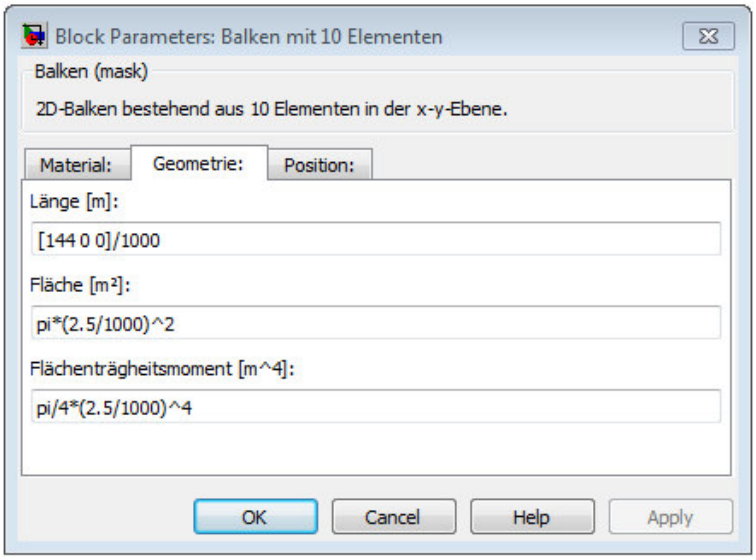


Abbildung 4.7: Maske des Subsystems: Geometrie

Der letzte Reiter „Position“ ist in Abbildung 4.8 dargestellt. Hier kann die Orientierung des letzten Koordinatensystems in Bezug auf das erste anhand der Eulerwinkel X-Y-Z verdreht werden.

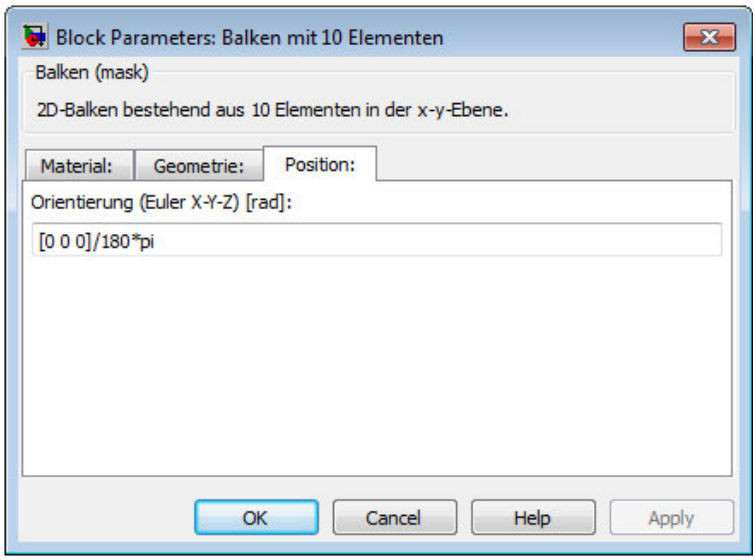


Abbildung 4.8: Maske des Subsystems: Position

Mit Hilfe dieser Angaben werden dann die Parameter für die Balkenelemente berechnet. Die Massenträgheitsmomente J werden jedoch nicht nach den Gleichungen 4.4 und 4.5 berechnet. Da in dieser Arbeit nur das statische Verhalten des Roboters untersucht werden

soll, wird hier jeweils ein deutlich höherer Wert von $J_{1/2/3} = 1\,\mathrm{kgm}^2$ eingesetzt, um die Eigenkreisfrequenzen $\omega_0 = \frac{k}{J}$ zu verringern. Damit kann in der Simulation mit größeren Zeitschritten gerechnet werden. Hierdurch wird die Zeit, die für eine Simulation benötigt wird, verkürzt. Diese Einstellung hat keine Auswirkung auf das statische Verhalten des Robotermodells.

4.3 Korrekturmethode

Für das Simulationsmodell soll eine Korrekturmethode entwickelt werden, mit der die Abweichung des Endeffektors zur Soll-Position korrigiert wird, die durch einen Krafteinfluss am Endeffektor entsteht. Diese Methode soll auch mit der Steuerung an dem Roboter angewendet werden. Hierfür sollen an dem Roboter aber keine neuen Sensoren angebracht werden. Die Abweichung entsteht durch die eingebrachten Elastizitäten in den ersten Armsegmenten des Roboters. Durch die Krafteinwirkung werden die Enden der Armsegmente aus der Soll-Position bewegt. Es werden die Winkel berechnet, um die die Armsegmente gedreht werden müssen, damit die Soll-Positionen an den Enden der Armsegmente erreicht werden. Die Längenänderungen der Balken aus den elastischen Verformungen werden dabei vernachlässigt, da diese sehr viel kleiner sind als die Fehler aus der Durchbiegung. Die Antriebsmomente der Motoren entsprechen den statischen Momenten um die Roboterarme im Gleichgewicht zu halten. Mit den einzelnen Momenten können die Winkel berechnet werden die zur Korrektur notwendig sind.

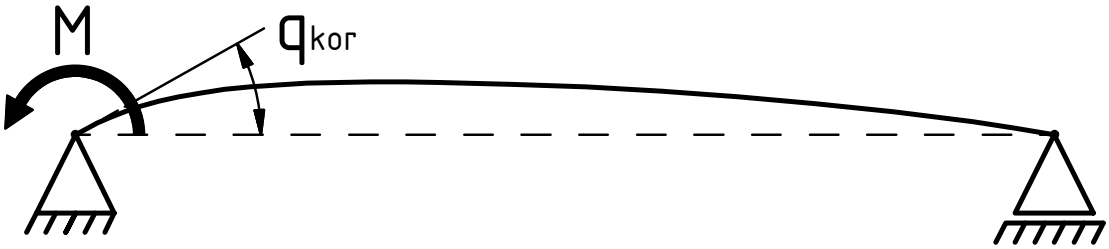


Abbildung 4.9: Skizze des Balkenmodells

Um den Winkel für einen Motor zu berechnen, wird das erste Armsegment als vereinfachtes Balkenmodell, mit den entsprechenden Elastizitäten der einzelnen Abschnitte, modelliert. Hierbei wird ein körperfestes Koordinatensystem gewählt, bei dem die beiden Enden des Balkens auf der x-Achse liegen. Das Motorende wird als Festlager modelliert und das Zwischengelenk als Loslager. An dem Motorende wird das Motormoment aufgebracht, wodurch eine Biegung im Armsegment entsteht. Dies ist in der Skizze in Abbildung 4.9 dargestellt. Für die Korrektur ist hierbei der Winkel q_{kor} der am Motorende entsteht relevant. Die Berechnung erfolgt über ein FE-Modell des Armsegmentes. Dies wird in der Funktion „FEModell“ erstellt, welches auf dem FE-Modell aus Kapitel 3 aufbaut.
In der ersten Zeile der Funktion „FEModell“ ist zu sehen, dass die Funktion keine Einga-

bewerte benötigt, sondern nur die Variable *kk* ausgibt. Die Berechnung der Variable wird im Folgenden erklärt.

```
function [kk] = FEModell()
```

In den ersten Zeilen stehen die Parameter des Modells. Hierzu gehört die Länge L des Armsegmentes, die Länge c des Aluminiumprofils, der Radius r des Rundstabes, die Anzahl n der Elemente. Die Geometrie des Armsegmentes ist in Abbildung 3.1 dargestellt. Anschließend wird die Anzahl der Freiheitsgrade dof berechnet.

```
% Parameter:
L  = 288 ;          % Länge des Armsegmentes [mm]
c  = 55 ;           % Länge des Aluminium Profils [mm]
r = 2.5;            % Radius des Rundstabes [mm]
n = 288;            % Anzahl der Elemente
l = L/n;            % Länge der Elemente [mm]
dof = 2*(n+1) ;     % Anzahl Freiheitsgrade global
```

Des Weiteren werden die Übergänge der unterschiedlichen Abschnitte des Armsegmentes bestimmt. Dies geschieht einmal in der Einheit „mm“ und einmal nach der Anzahl der Elemente.

```
Abschnitte_mm = [12 12+c L-c-12 L-12 L];
Abschnitte_element = Abschnitte_mm/l;
```

Als nächstes wird den jeweiligen Abschnitten der Elastizitätsmodul zugewiesen. Die Abschnitte eins, drei und fünf bestehen aus Stahl mit einem Elastizitätsmodul von 210 GPa. Der Elastizitätsmodul des dritten Abschnittes wird durch den Korrekturfaktor 1,96 geteilt, um dem statischen Verhalten des gefertigten Armsegmentes aus Kapitel 3 zu entsprechen. Die Abschnitte zwei und vier bestehen aus einer Aluminiumlegierung mit einem Elastizitätsmodul von 70 GPa. Da im SimMechanics Modell nur der dritte Abschnitt elastisch modelliert wurde, werden die andern Abschnitte versteift, indem die Elastizitätsmoduln der Abschnitte um den Faktor 10^3 erhöht werden.

```
% Elastizitäten
E(1) = 210e3*1e3;   % 210e3 MPa für die Korrektur am Roboter
E(2) = 70e3*1e3;    %  70e3 MPa für die Korrektur am Roboter
E(3) = 210e3/1.96;  % 1.96 : Angleichung an die gefertigten Armsegmente
E(4) = E(2);
E(5) = E(1);
```

In dem nächsten Abschnitt werden die Flächenträgheitmomente der Abschnitte berechnet. Das Flächenträgheitsmoment für den ersten und letzten Abschnitt wird nach der Gleichung 3.1 mit $h = 20\,\text{mm}$ berechnet. Für die Abschnitte zwei und vier wird das Flächenträgheitsmoment des Aluminiumprofils mit $I_{2/3} = 7200\,\text{mm}^4$ aus dem Datenblatt entnommen. Dies

befindet sich im Anhang A.2. Für den dritten Abschnitt, wird das Flächenträgheitsmoment des Rundstabes nach der Gleichung 3.2 mit einem Radius $r = 2{,}5\,\text{mm}$ berechnet.

```
% Flächenträgheitsmomente
I(1) = 20*20^3/12;  % [mm^4]
I(2) = 0.72*10^4;   % [mm^4]
I(3) = pi/4*r^4;    % [mm^4]
I(4) = I(2);
I(5) = I(1);
```

Danach werden die Gesamtsteifigkeitsmatrix und der Vektor der Knotenverformung initialisiert.

```
% Initialisieren
K = zeros(dof,dof);    % Gesamtsteifigkeitsmatrix
u = zeros(dof,1);      % Vektor der Knotenverformungen
```

Um das Gleichungssystem zu lösen, müssen die freien Freiheitsgrade bekannt sein, damit die Inverse der Gesamtsteifigkeitsmatrix gebildet werden kann. In diesem Modell sind durch das Festlager der erste und durch das Loslager der vorletzte Freiheitsgrad blockiert.

```
% Randbedingungen
ufree =  [(2 : dof-2) dof] ;   % Welche Freiheitsgrade frei sind
```

Mit der „for“-Schleife wird die Gesamtsteifigkeitsmatrix K mit den Steifigkeitsmatrizen der einzelnen Balkenelemente gefüllt. Die Steifigkeitsmatrizen der Balkenelemente werden in der Funktion *esm* mit der Steifigkeit EI und der Länge l der Elemente berechnet. Die Zusammensetzung der Steifigkeitsmatrix K und der Steifigkeitsmatrizen der Balkenelemente werden, wie in [7] beschrieben, erstellt.

```
% Aufstellen der Gesamtsteifigkeitsmatrix K
for i = 1 : n
    if i ≤ Abschnitte_element(5)
        EI(i) = E(5)*I(5);
    end
    if i ≤ Abschnitte_element(4)
        EI(i) = E(4)*I(4);
    end
    if i ≤ Abschnitte_element(3)
        EI(i) = E(3)*I(3);
    end
    if i ≤ Abschnitte_element(2)
        EI(i) = E(2)*I(2);
    end
    if i ≤ Abschnitte_element(1)
        EI(i) = E(1)*I(1);
```

```
    end
    inz = (i*2-1:i*2-1+3);
    K(inz,inz) = K(inz,inz) + esm(EI(i),l);
end
```

Zur Lösung des Gleichungssystem $f = Ku$, muss die Steifigkeitsmatrix reduziert werden und anschließend von links mit der Inversen $Kred\_inv$ multipliziert werden.

```
% Bildung der inversen des reduzierten Systems
Kred = K(ufree,ufree);
Kred_inv = inv(Kred);
```

Für die Korrektur ist die Verdrehung des Balkens am Festlager zu berechnen, da das System nur mit einem Moment an dem ersten Knoten belastet wird, reduziert sich die Gleichung wie folgt:

$$ured \quad = \quad inv(Kred) \begin{bmatrix} M \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.8}$$

Die Verdrehung steht in dem Vektor $ured$ an der ersten Stelle. In dem Kraftvektor f ist nur an der ersten Stelle ein Eintrag ungleich null, somit vereinfacht sich die Gleichung 4.8 wie folgt:

$$q_{kor} = ured_1 = inv(Kred)_{11} \cdot M \tag{4.9}$$

Für die Berechnung der Verdrehung q_{kor} wird nur der Wert $kk = inv(Kred)_{11}$ benötigt, mit dem das Moment multipliziert wird.

```
kk = Kred_inv(1,1);
```

Der Quellcode der Funktion „FEModell“ befindet sich im Anhang A.4.

Für die Umsetzung der Korrektur in dem Simulationsmodell wurde das Subsysteme „Winkelkorrektur“ erstellt, dieses wird im Folgenden beschrieben.

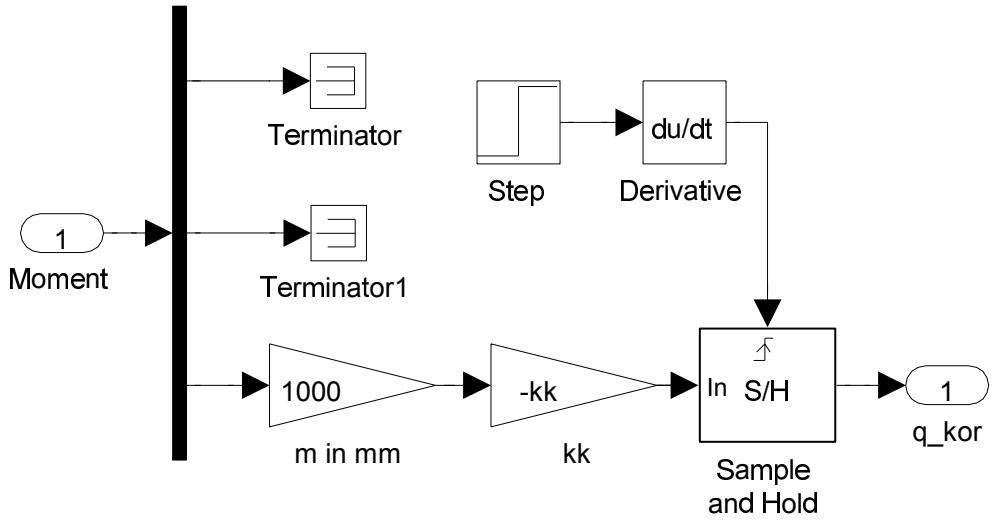


Abbildung 4.10: Subsystem: Winkelkorrektur

Das Subsystem „Winkelkorrektur“ ist in Abbildung 4.10 dargestellt. In diesem Subsystem wird die Gleichung 4.9 umgesetzt. Hierfür wird das Motormoment um die z-Achse in die Einheit „Nmm“ umgerechnet und mit dem Wert $-kk$ multipliziert. Das negative Vorzeichen entsteht dadurch, dass das Nachstellen entgegen der Auslenkung wirken soll. Die Korrektur soll zum Zeitpunkt *tkor* vorgenommen werden und dann bis zum Ende gehalten werden. Hierfür wurde der Block „Sample and Hold“ verwendet. Dieser Block gibt den Wert der am Eingang anliegt an den Ausgang weiter, solange an dem oberen „trigger“ Eingang ein Signalwert von „1“ anliegt. Liegt ein Signalwert von „0“ an, gibt er den letzten Wert aus, bei dem der Signalwert am „trigger“ Eingang „1“ war. Zu Beginn der Simulation ist der Ausgabewert auf „0“ gesetzt.

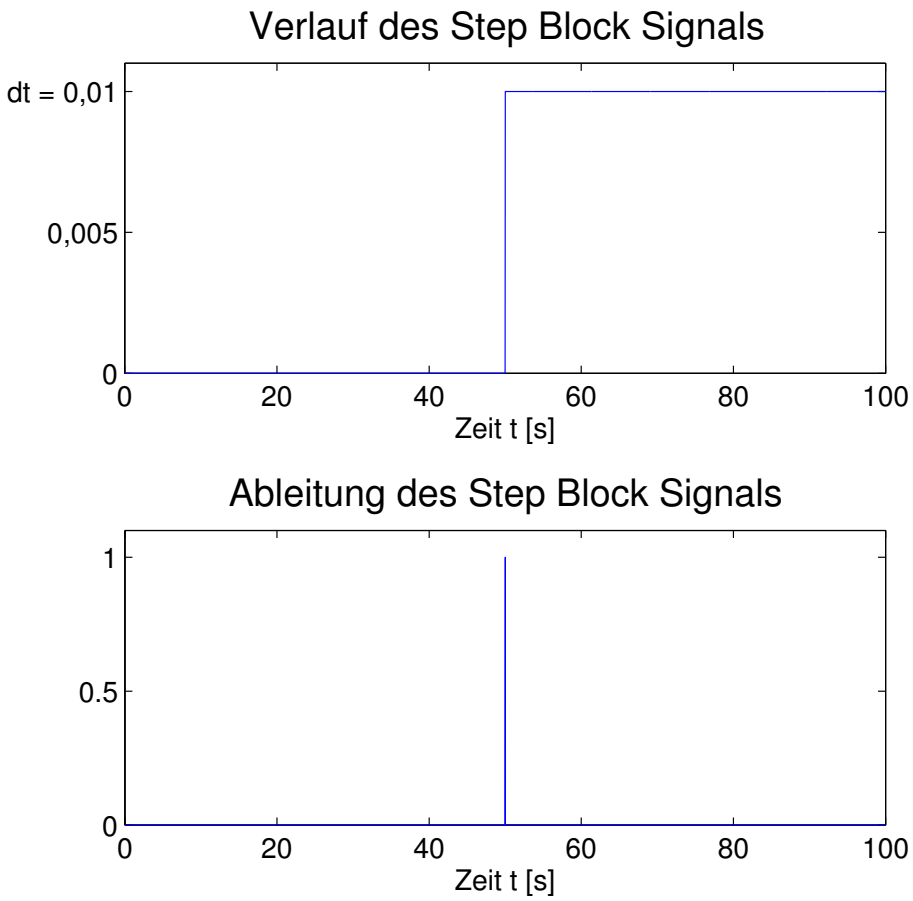


Abbildung 4.11: Verlauf des Step Block Signals und der Ableitung

Um ein „trigger“ Signal zu erzeugen, welches nur zum Zeitpunkt *tkor* den Wert „1“ annimmt, wird ein „Step“ Block verwendet, der zum Zeitpunkt *tkor* sein Signal von „0“ auf „*dt*“ wechselt. Dieses Signal wird mit dem Block „Derivate“ abgeleitet. Dadurch, dass der Zeitschritt der Simulation *dt* beträgt, ergibt sich bei der Ableitung des Sprunges der Wert „1“. Der Verlauf des „Step“ Block Signals und der Ableitung ist in Abbildung 4.11 dargestellt.

5 3-RRR Roboter

Als erstes wird in diesem Kapitel die Inbetriebnahme und Konfiguration des 3-RRR Roboters mit den dazugehörigen Komponenten beschrieben. Anschließend wird die in Simulink erstellte Steuerung erklärt.

5.1 Inbetriebnahme und Konfiguration des 3-RRR Roboters

In diesem Abschnitt wird die Inbetriebnahme und Konfiguration der FAULHABER DC Servomotoren, der Bachmann MX 220 SPS und des Simulink Modells in Form einer Bedienungsanleitung beschrieben. Die Einstellungen wurden mit Hilfe der Hausarbeit „SCARA-Robotersteuerung mit MX 220 SPS und CANopen“ [1] durchgeführt, in der dieselben Komponenten verwendet werden.

5.1.1 Konfiguration der FAULHABER Motoren

Bevor die FAULHABER Motoren in Betrieb genommen werden können, müssen diese konfiguriert werden. Für die Konfiguration und Inbetriebnahme der Motoren sind folgende Komponenten erforderlich:

- ein USB zu CAN Interface („USB-to-CAN compact“ von der Firma IXXAT)
- FAULHABER 3564K024B DC - Servomotor
- eine Adapterplatine pro Motor, Artikel Nr. 6501.00065
- ein vorkonfektioniertes Verbindungskabel
- ein 24 V Netzteil
- Software: FAULHABER Motion Manager 4

Jeder Motor wird mit einer Adapterplatine verbunden. Die Verkabelung und Einstellung für einen Betrieb über einen CAN-Anschluss erfolgt nach dem Gerätehandbuch von Faulhaber [4]. Danach kann der Computer über das „USB-to-CAN compact“ Interface von der Firma IXXAT mit einem vorkonfektioniertem Verbindungskabel an die Adapterplatine angeschlossen werden. Nun können die Motoren nacheinander mit der „FAULHABER Motion Manager 4“ Software konfiguriert werden.

Die Konfiguration der Motoren erfolgt nach dem Kommunikations- und Funktionshandbuch [8]. Beim Starten der Software „FAULHABER Motion Manager 4“ wird nach angeschlossenen Antriebsknoten gesucht. Ist diese Suche erfolglos, öffnet sich der Verbindungsassistent (Abb. 5.1).

Abbildung 5.1: Verbindungsassistent (Schritt 1: Auswahl der Steuerung)

In dem ersten Schritt ist die Steuerung auszuwählen, hier wird die Option „Motion Controller mit CAN-Schnittstelle“ gewählt. Nach der Vollendung des ersten Schrittes, folgt der zweite Schritt (Abb. 5.2) mit der Auswahl der Verbindung.

Abbildung 5.2: Verbindungsassistent (Schritt 2: Auswahl der Verbindung)

Im zweiten Schritt ist das CAN-Interface mit „IXXAT - VCI3“ als Schnittstelle zu wählen und des Weiteren muss der Punkt „Angeschlossene Geräte müssen noch konfiguriert werden“ ausgewählt werden. Nach der Bestätigung der Einstellung öffnet sich ein weiteres Fenster (Abb. 5.3).

Abbildung 5.3: Verbindungsassistent (Select VCI Device)

In diesem Fenster ist das VCI Device „USB-to-CAN compact“ in der Liste auszuwählen und mit „OK“ zu bestätigen. Nach der Auswahl des Interfaces folgt der dritte Schritt 5.4.

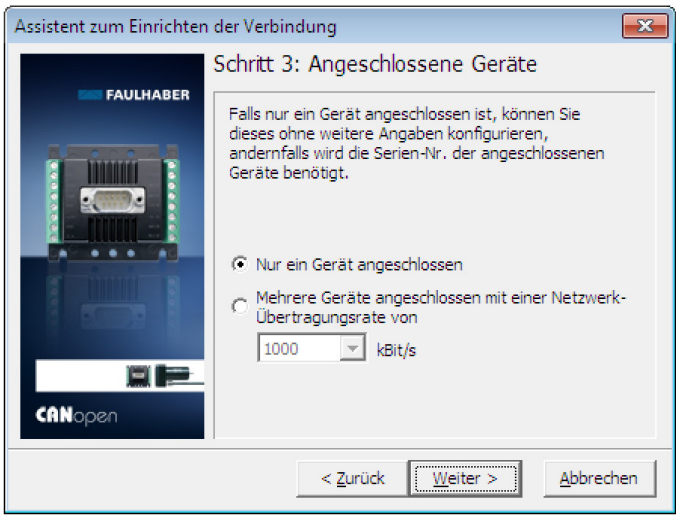


Abbildung 5.4: Verbindungsassistent (Schritt 3: angeschlossene Geräte)

Da jeder Motor einzeln konfiguriert wird, ist bei Schritt 3 „Nur ein Gerät angeschlossen“ zu wählen und mit „Weiter“ zu bestätigen.

Abbildung 5.5: Verbindungsassistent (Schritt 4: Geräte-Konfiguration)

In dem letzten Schritt werden für die Motoren die Übertragungsrate (Baudrate) und die Knotennummer eingestellt, wie in Abbildung 5.5 zu sehen. Für die Übertragungsrate werden „1000 kBit/s“ ausgewählt. Für die drei Motoren werden die Knotennummern 1 bis 3 gewählt.

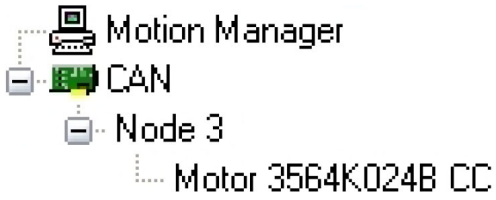


Abbildung 5.6: Netzwerkteilnehmer

Nach Abschluss der Gerätekonfiguration wird der Motor mit seiner Knotennummer angezeigt. Durch Doppelklick auf den Motor kann dieser ausgewählt werden und unter dem Startleistenmenüpunkt „Konfiguration>Antriebskonfiguration“ können die Einstellungen für den Motor vorgenommen werden. Für die Ansteuerung der Motoren mit FAULHABER-Befehlen über CAN muss in der Antriebskonfiguration im Reiter „Modus“ der Punkt „FAULHABER Mode(-1)“ gewählt werden, wie in Abbildung 5.7 zu sehen.

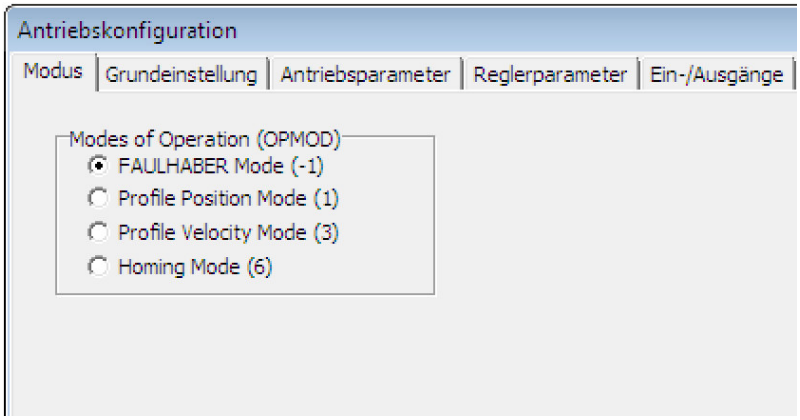


Abbildung 5.7: Antriebskonfiguration: Modus

Unter dem Reiter „Reglerparameter“ werden die Parameter der Antriebsregelung eingestellt. Diese werden wie in der Abbildung 5.8 zu sehen gewählt.

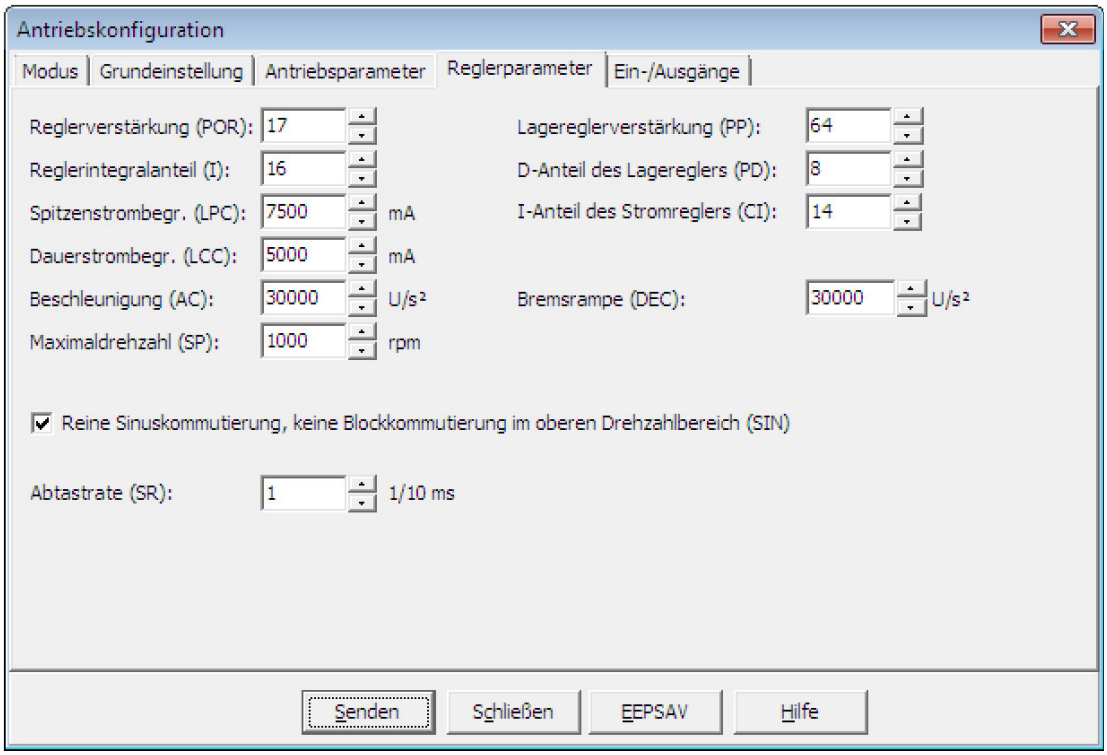


Abbildung 5.8: Antriebskonfiguration: Reglerparameter

Weitere Einstellungen sind für die einzelnen Motoren nicht vorzunehmen. Nachdem dies für alle Motoren durchgeführt wurde, sind diese für den Betrieb in einem CANopen Netzwerk bereit.

5.1.2 Konfiguration der Bachmann Steuerung

Bevor die Steuerung das erste Mal eingeschaltet werden kann, muss die Steuerungshardware montiert und verkabelt werden. Die Erstinbetriebnahme ist in der Bachmann Anleitung [3] beschrieben. Die Steuerung besteht aus einem digitalem Ein- und Ausgangsmodul DIO232, einem analogen Ein- und Ausgangsmodul AIO288 sowie einem Leermodul LM201. Diese werden zusammen mit dem MX 220 Prozessormodul auf der BS 205 Busschiene montiert. Nachdem die Steuerung verkabelt und die einzelnen Module mit Strom versorgt wurden, kann die Steuerung eingeschaltet werden.
Die Steuerung wird mit der „Solution Center Software“ von Bachmann konfiguriert. Diese muss nach der Anleitung im Referenzhandbuch [9] installiert werden. Anschließend müssen der Rechner und die Steuerung miteinander verbunden werden. Hierfür wurde eine Verbindung mit Ethernet über ein vorhandenes LAN gewählt. Somit kann die Position des Rechners und der Steuerung an jeder Anschlussmöglichkeit zum LAN gewählt werden. Nachdem die Geräte verbunden sind, kann in der „Solution Center Software“ die Steuerung über das Kontextmenü „Datei>NEU>Device“ mit der IP-Adresse 141.22.37.26 hinzugefügt werden. Nach erfolgreicher Konfiguration erscheint die Steuerung unter dem Punkt „Devices“ und ist nun betriebsbereit.

5.1.3 Konfiguration von Matlab Simulink

Um mit Matlab Simulink Programme für die Steuerung zu schreiben und diese auf die Steuerung zu laden, muss das Plug-In „M-Target for Simulink Plug-In“ nach der technischen Beschreibung von Bachmann [10] installiert werden. Hierbei ist besonders auf die Reihenfolge der Installationen zu achten, wie sie im folgenden aufgeführt ist:

- Bachmann Solution Center
- MATLAB/Simulink inklusive Realtime Workshop
- M-Target for Simulink

Eine andere Reihenfolge der Installationen ist möglich, hierbei muss jedoch das „M-Target for Simulink Plug-In“ nach der Installation manuell konfiguriert werden. Dies ist in der technischen Beschreibung von Bachmann [10] näher erläutert. Des Weiteren muss das Simulink Modell konfiguriert werden. Hierfür wird in der Menüleiste Simulation>Configuration Parameters ausgewählt, worauf sich das Konfigurationsfenster (Abb. 5.9) öffnet.

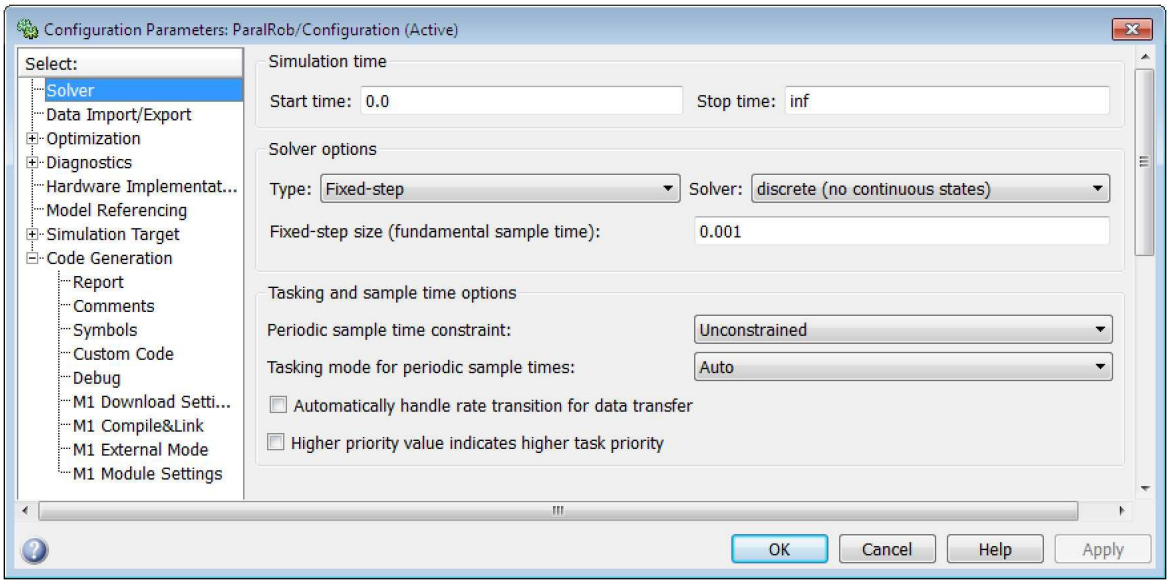


Abbildung 5.9: Simulink Parameter/Solver

In dem Menüpunkt „Solver“ werden folgende Einstellungen vorgenommen:

- **Simulation time**
  Bei „Start time“ wird die Zeit eingetragen nach der das Modell startet. Hier wird der Wert „0.0“ eingetragen, damit das Modell direkt nach der Übertragung auf die Steuerung gestartet wird. Als „Stop time“ wird „inf“ eingetragen, wodurch das Modell läuft bis es manuell gestoppt wird.

- **Solver options**
  Damit das Modell auf der Bachmann MX 220 ausgeführt wird, muss als „Type“ „Fixed-step“ gewählt werden. Als „Fixed-step size (fundamantal sample time)“ wird „0.001“ gewählt, damit es der Zykluszeit des CAN-Bus von 1 ms entspricht.

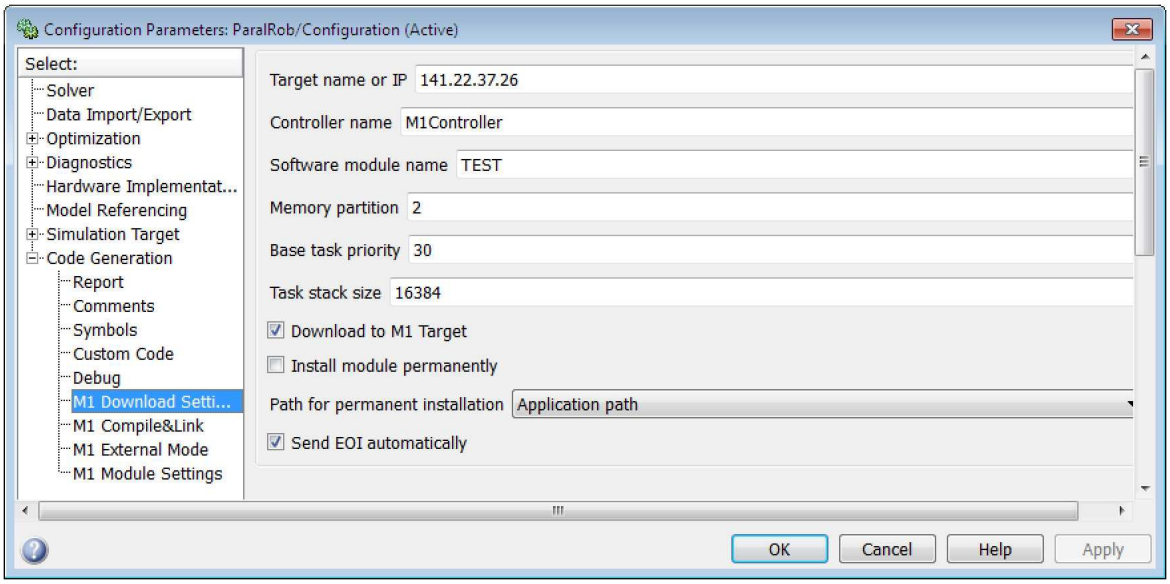


Abbildung 5.10: Simulink Parameter/M1 Download Settings

Die Verbindungsparameter zur MX 220 Steuerung werden unter dem Menüpunkt „M1 Download Settings“ (Abb. 5.10) eingestellt:

- **Target name or IP**
  Hier muss die IP-Adresse der Bachmann Steuerung eingetragen werden.
- **Software module name**
  Unter dem hier eingetragenen Namen wird das Modell im Solution Center nach der Übertragung angezeigt.
- **Download to M1 Target**
  Dieser Punkt wird ausgewählt, damit das Modell nach der Übersetzung auf die Steuerung übertragen wird. Wird dieser Punkt nicht ausgewählt, wird ein Programmcode erzeugt, der nachträglich übertragen werden kann.
- **Send EOI automatically**
  Dieser Punkt wird ausgewählt, damit das Modell direkt nach der Übertragung auf die Steuerung automatisch gestartet wird.

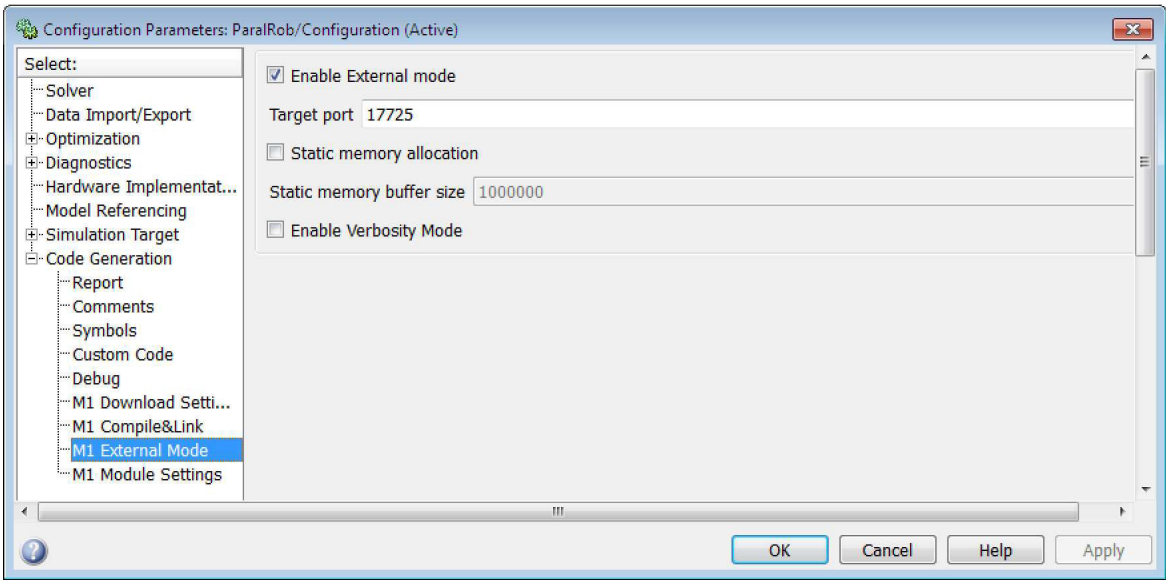


Abbildung 5.11: Simulink Parameter/M1 External Mode

In dem Menüpunkt „M1 External Mode“ (Abb. 5.11) gibt es eine besondere Einstellungsmöglichkeit:

- **Enable External mode**
  Die Besonderheit dieser Einstellungsmöglichkeit liegt darin, dass es möglich ist sich mit dem auf der Steuerung laufenden Simulink Model zu verbinden. Somit besteht die Möglichkeit direkt auf das laufende Modell zuzugreifen und z.B. Variablen zu ändern.

5.1.4 CANopen Netzwerk

Nachdem alle Komponenten für das CANopen Netzwerk konfiguriert wurden, können nun die Motoren und die Steuerung verbunden werden. In einem CAN Bussystem werden alle Knoten, wie in Abbildung 5.12 dargestellt, parallel angeschlossen. Hierzu wird jeder Motor mit einer Adapterplatine verbunden und die Bachmann Steuerung mit einem Buskabel an eine zusätzliche Adapterplatine angeschlossen. Die Adapterplatinen werden dann, wie in Abbildung 5.12 dargestellt, parallel geschaltet und an jedem Ende wird zwischen der CAN_H und CAN_L Leitung ein Abschlusswiderstand von 120 Ω angeschlossen.

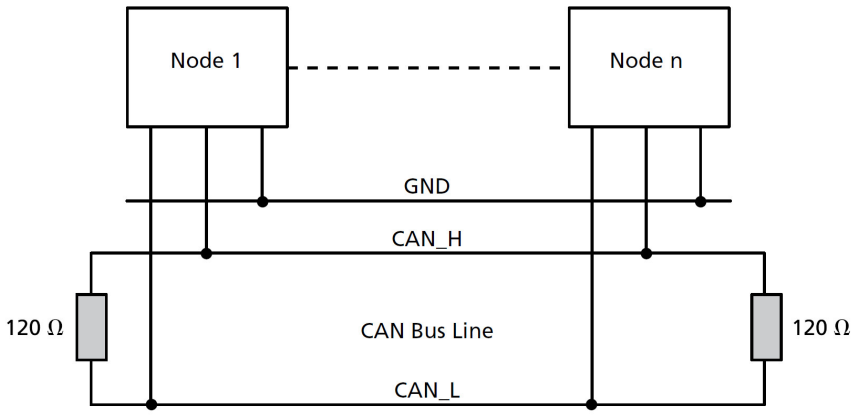


Abbildung 5.12: CAN Netzwerk[3]

Konfiguration des CANopen-Netzwerks im Solution Center

Mit der „Solution Center Software“ von Bachmann kann ein CANopen-Netzwerk eingerichtet werden. Hierfür sind die in den vorherigen Kapiteln beschriebenen Konfigurationen und Verkabelungen der Komponenten erfolgreich durchzuführen. Des Weiteren wird eine EDS-Datei für die Motoren benötigt, in der die Geräteeigenschaften festgelegt sind. Nachdem alle notwendigen Schritte erledigt sind, kann ein CANopen-Netzwerk eingerichtet werden. Dies erfolgt im Kontextmenü des Menüpunktes „CAN“ der Steuerung, wie in Abbildung 5.13 zu sehen.

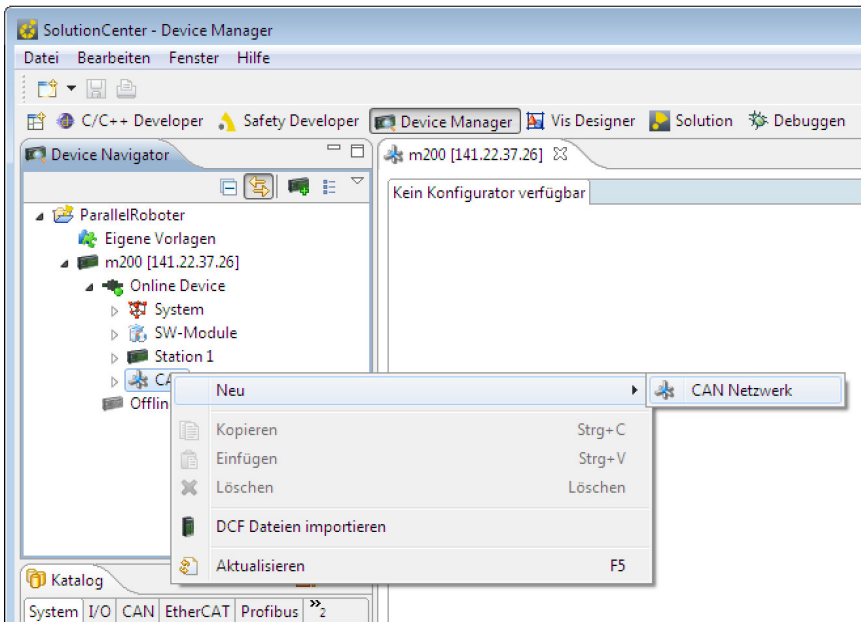


Abbildung 5.13: Einrichtung eines CAN Netzwerkes

In dem neuen Fenster „Online CAN Netzwerk Konfiguration“ (Abb. 5.14) erfolgt die Auswahl eines freien Ports. Hier wird der CAN-Bus-Anschluss der MX 220 Steuerung angezeigt und ausgewählt.

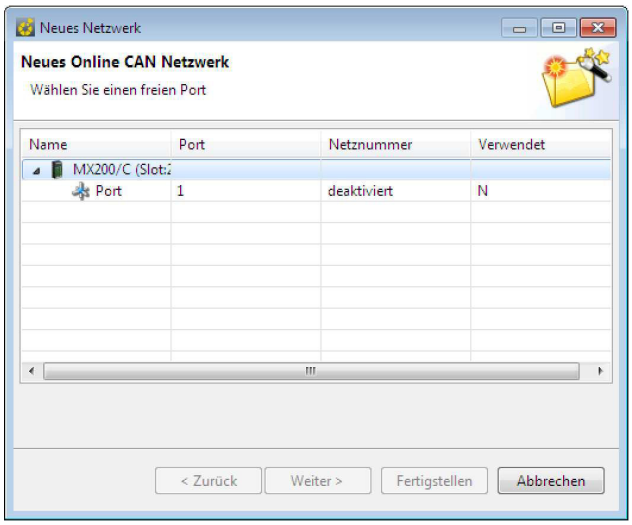


Abbildung 5.14: Auswahl CAN Netzwerk

Nach der Bestätigung der Auswahl erfolgt die Einstellung für den Netzwerknamen, die Netzwerknummer und der Baudrate (Abb. 5.15). Der Netzwerkname wird hier mit „NET1“ gewählt und für die Netzwerknummer wird die „1“ gewählt. Die Baudrate wird wie in Kapitel 5.1.1 bei den Motoren mit 1000 kBit/s gewählt. Nach Bestätigung der Einstellung muss die Steuerung neu gebootet werden.

Abbildung 5.15: Konfiguration eines CAN-Netzwerkes

Nachdem die Steuerung erfolgreich neu gebootet wurde, ist ein neuer Eintrag im CAN-Menüpunkt vorhanden. Dies ist das neu eingerichtete CAN-Netzwerk, welches die Bachmann Steuerung mit der Netzwerknummer „1“ enthält. In dem Kontextmenü des Netzwerkes kann das CAN-Netzwerk nach neuen Teilnehmern durchsucht werden, wie in Abbildung 5.16 zu sehen ist.

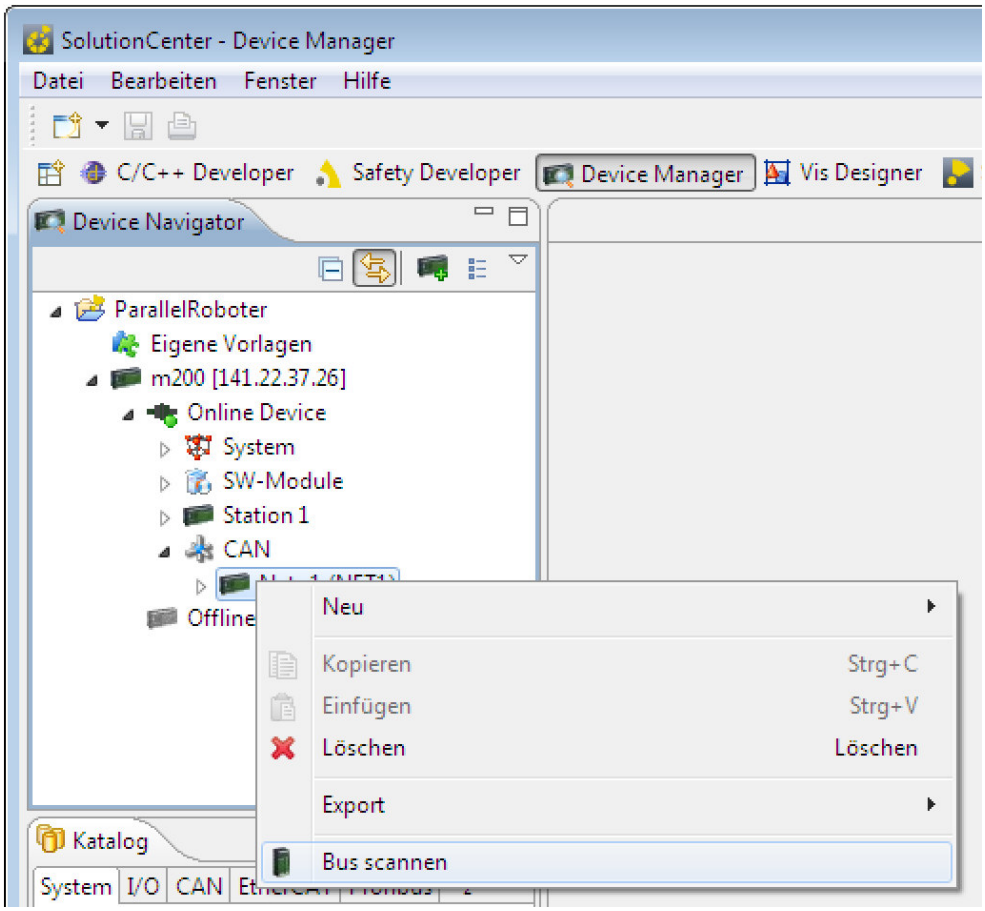


Abbildung 5.16: CAN Bus Scannen

Nach der Auswahl des Menüpunktes „Bus Scannen“ erscheint ein Fenster (Abb. 5.17), in dem der Bereich angegeben wird, in dem gesucht werden soll. Hier wird der Bereich von „1“ bis „3“ angegeben, da sich die Knotennummern der Motoren in diesem Bereich befinden.

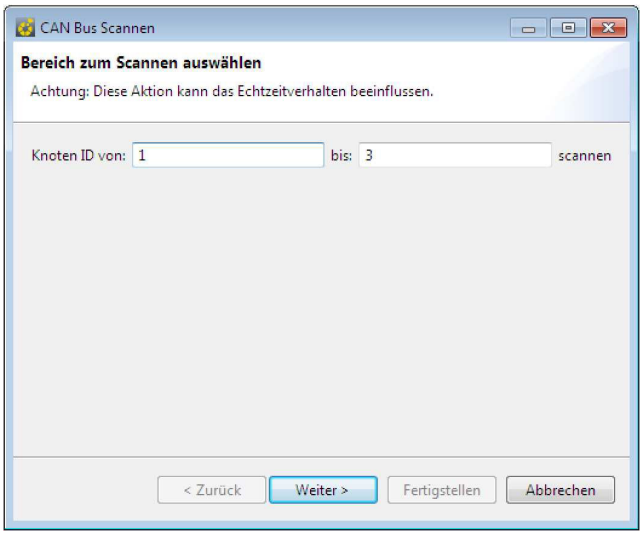


Abbildung 5.17: Auswahl des Suchbereiches

Nachdem die Suche beendet ist, werden in dem Fenster (Abb. 5.18) die gefundenen Busteilnehmer angezeigt. Nun werden die drei FAULHABER Motoren in der Liste angezeigt. Für diese werden nun die EDS-Dateien durch klicken auf das EDS-Feld hinzugefügt. Nach dem Klicken auf das jeweilige EDS-Feld öffnet sich ein Fenster in dem die entsprechende EDS-Datei ausgewählt wird. Nachdem die EDS-Dateien für die Motoren hinzugefügt wurden, werden die Motoren durch Drücken des Knopfes „Fertigstellen“ dem CANopen Netwerk hinzugefügt.

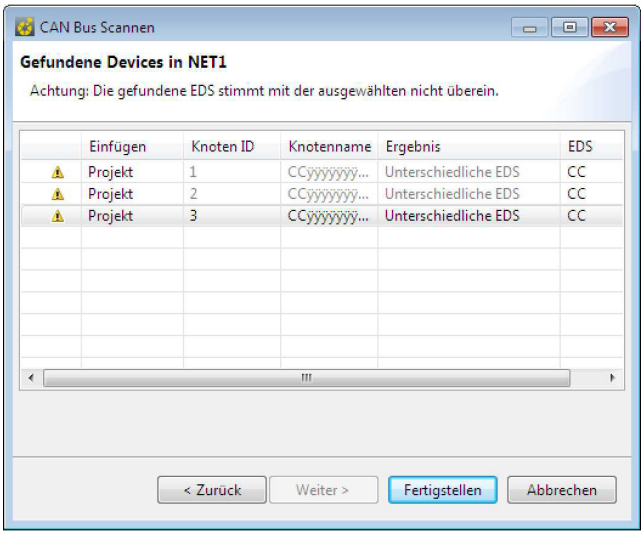


Abbildung 5.18: Teilnehmer hinzufügen

Die Motoren können nun von der Steuerung mit den FAULHABER Befehlen angesteuert werden. Durch das Hinzufügen der Motoren wurde anhand der EDS-Dateien ein PDO-Netzwerk erstellt, das man sich unter dem Reiter „Konfiguration>PDO Verbindungen“ ansehen kann (Abb. 5.19).

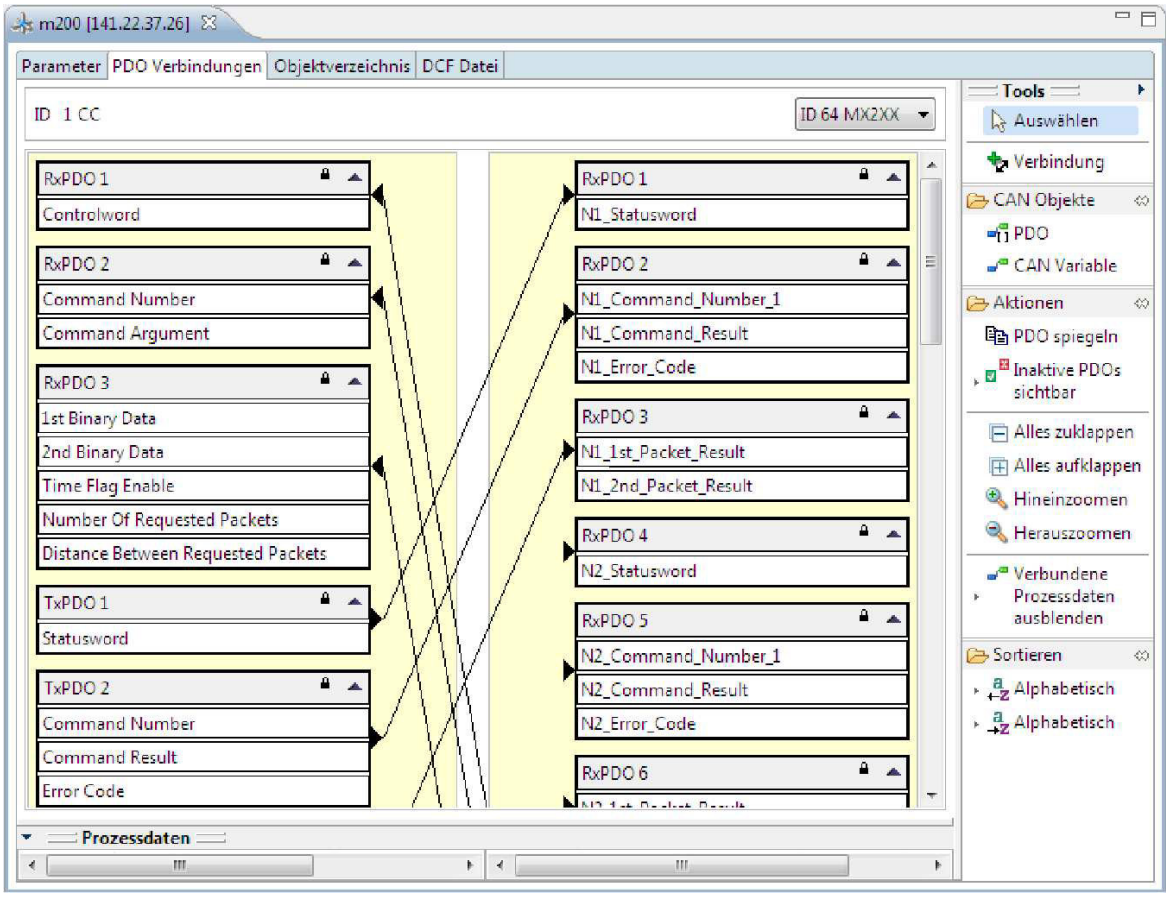


Abbildung 5.19: PDO Verbindungen

Bei den PDO Verbindungen kann man die einzelnen Verknüpfungen zwischen der Steuerung und den Motoren erkennen. Anhand der Pfeilrichtung ist zu erkennen in welche Richtung ein Datenpaket gesendet wird. Innerhalb der PDOs stehen die Namen der Variablen in denen die empfangenen Daten geschrieben werden.

Einrichtung einer synchronen Datenübertragung

Bei einer synchronen Datenübertragung werden von den Netzwerkteilnehmern in einem vorgegebenen Zeitabstand ein SYNC-Telegramm von einem Erzeuger gesendet und von den Teilnehmern empfangen. Diese Einstellung wird nur bei bestimmten PDOs verwendet. Hierzu zählen die PDOs, die für die Übertragung der „FAULHABAER Kommandos“ und der „Trace Daten“ verantwortlich sind. Die Einstellung der PDOs wird in Abbildung 5.20 beispielhaft an einem PDO dargestellt. Durch Doppelklick auf das gewünschte PDO wird die Dialogbox „PDO Objekt“ geöffnet. Um eine synchrone Datenübertragung zu erreichen, muss dort der „Transmission Type“ auf „cyclic synchronous“ gestellt werden.

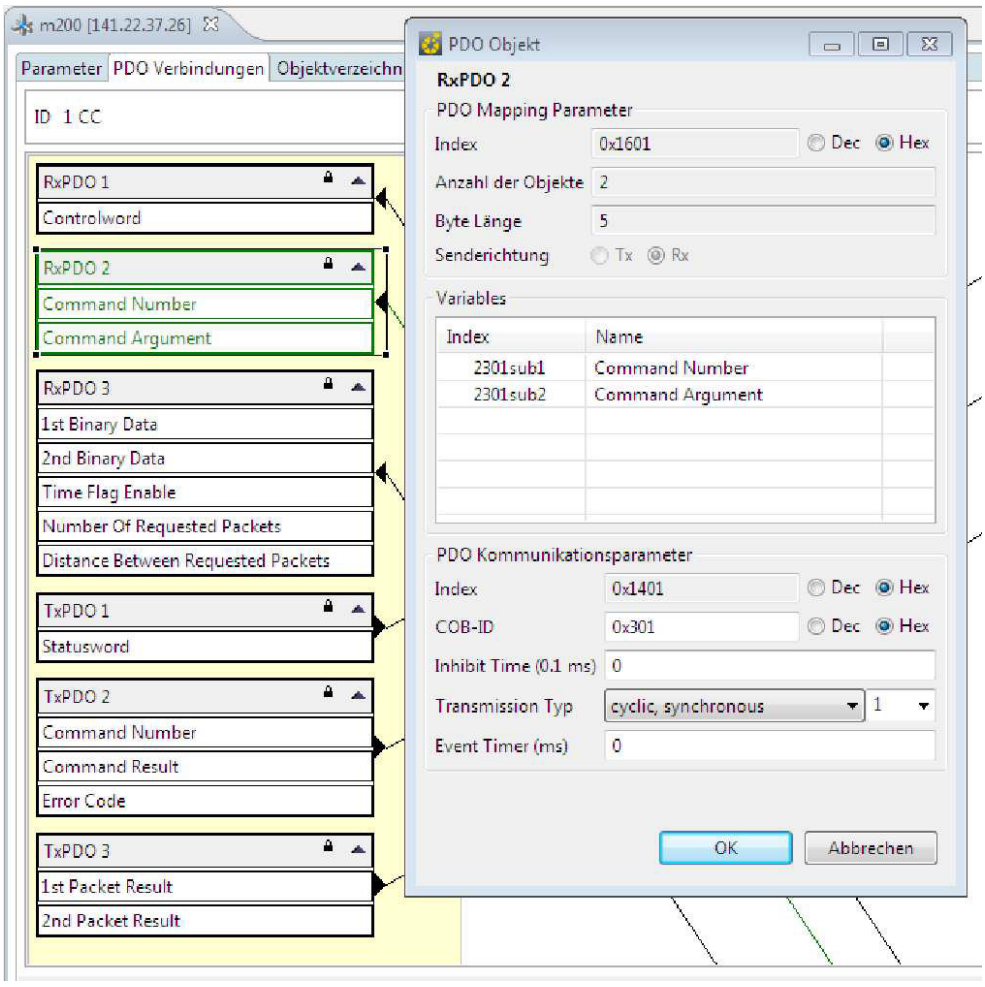


Abbildung 5.20: PDO Einstellung

Abschließend muss die Bachmann Steuerung als SYNC-Telegramm Erzeuger konfiguriert werden. Dies erfolgt in der DCF-Datei, die unter dem Menüpunkt „Konfigurator>DCF Datei>DCF Datei bearbeiten“ geöffnet wird. In dieser Datei müssen zwei Einstellungen

vorgenommen werden, welche in der Abbildung 5.21 rot umrandet sind. Mit dem ersten Eintrag wird die Bachmann Steuerung zu einem SYNC-Telegramm Erzeuger konfiguriert und der zweite Eintrag stellt die Zykluszeit auf 1000 μs .

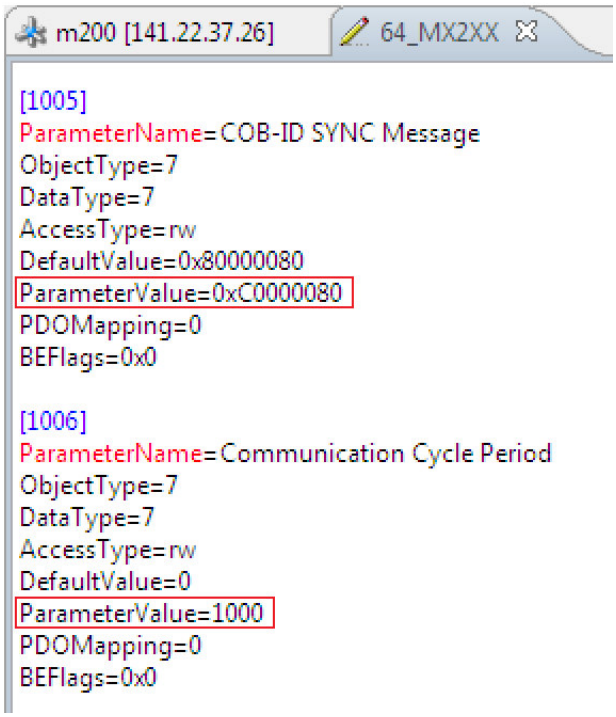


Abbildung 5.21: Erstellung eines SYNC-Telegramm Erzeugers

5.2 Simulink Steuerung

Für die Steuerung des Roboters wurde in Simulink ein Modell erstellt, mit dem die Motoren des Roboters über die Bachmann Steuerung angesteuert werden. Im folgenden wird der Aufbau und die Funktionsweise des Modells beschrieben.

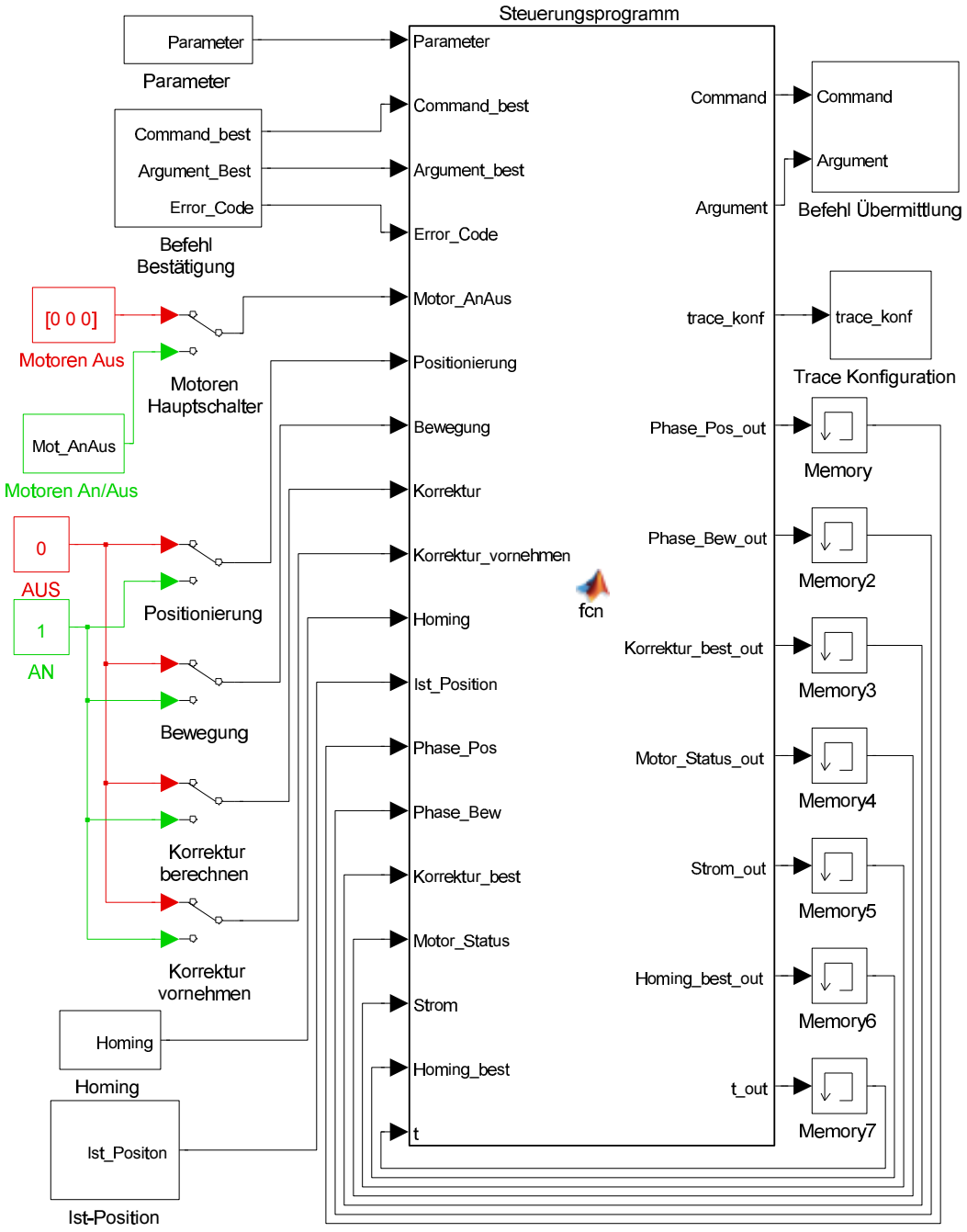


Abbildung 5.22: Simulink Steuerungsmodell

In der Abbildung 5.22 ist das Simulink Steuerungsmodell abgebildet. Im Zentrum des Modells ist der „Matlab Function“ Block „Steuerungsprogramm“. Mit diesem Block kann eine Matlab Funktion in ein Simulinkmodell eingebunden werden. Der Ablauf des Steuerungsprogramms wird in dem Kapitel 5.2.1 beschrieben. Diese Matlab Funktion wird in jedem Zeitschritt mit den auf der linken Seite anliegenden Variablen ausgeführt und gibt die berechneten Ergebnisse auf der rechten Seite aus. Da der Matlab Function Block keine globalen Variablen zulässt, werden folgende Variablen über „Memory“ Blöcke, mit denen die Werte aus dem letzten Zeitschritt gespeichert werden, auf die linke Seite der Funktion zurückgeführt:

- Phase_Pos
- Phase_Bew
- Korrektur_best
- Motor_Status
- Strom
- Homing_best
- t

In dem Subsystem Parameter (Abb. 5.23) wird in den Blöcken „X“ und „Y“ die Position, auf die der Endeffekor gefahren werden soll, eingegeben und über den Block „kk“ wird der Wert für die Korrektur an die Matlab Funktion weiter geleitet. Dieser Wert wird beim Übertragen des Simulinkmodells an die Steuerung durch die Funktion „FEModell“ berechnet, welches in Kapitel 4.3 beschrieben wird.

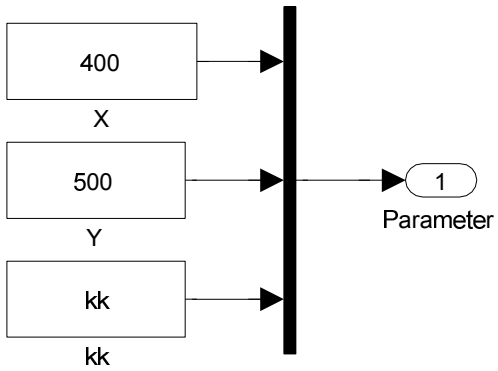


Abbildung 5.23: Subsystem: Parameter

Des Weiteren benötigt die Matlab Funktion Werte, die von den Motoren über den CAN Bus an die Steuerung übertragen werden. Diese Werte werden in den Subsystemen „Befehl

Bestätigung“ und „Ist-Position“ verarbeitet und an die Matlab Funktion weitergeleitet. Der Empfang der Variablen über den CAN erfolgt über die „CAN-PDO-Read“ Funktionsblöcke. Die Einstellungen der Funktionsblöcke erfolgt über die Maske der Blöcke. Eine solche Maske ist in Abbildung 5.24 dargestellt.

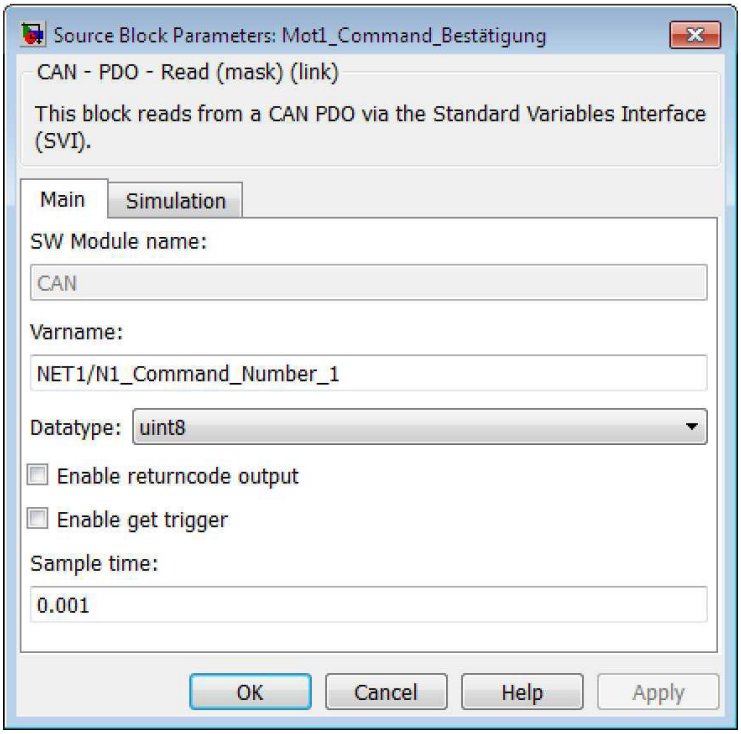


Abbildung 5.24: Maske eines CAN-PDO-Read Funktionsblocks

In der Maske wird unter „Varname“ der Name des Netzwerkes „NET1“ und der Name der Variablen angegeben, die mit diesem Block empfangen werden soll. Diese werden mit einem „/“ verbunden. Außerdem wird der Datentyp der Variablen festgelegt, sowie die „Sample time“ mit „0,001“ auf die Zykluszeit gestellt.

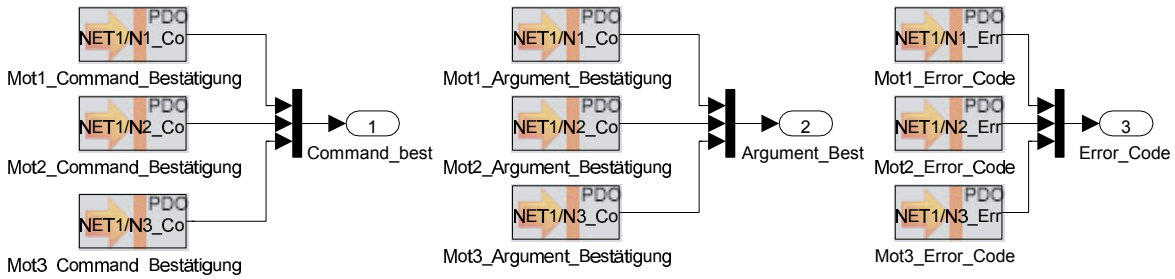


Abbildung 5.25: Subsystem: Befehl Bestätigung

Über das Subsystem „Befehl Bestätigung“ (Abb. 5.25) werden die Bestätigung der empfangenen Befehle, die von den Motoren gesendet werden, an die Matlab Funktion weiter geleitet. Zur Bestätigung sendet jeder Motor den empfangenen Befehl, das Argument und einen Error Code. Nimmt der Error Code den Wert „1“ an, wurde der Befehl mit dem Argument ausgeführt, andernfalls wird der Wert „0“ gesendet.

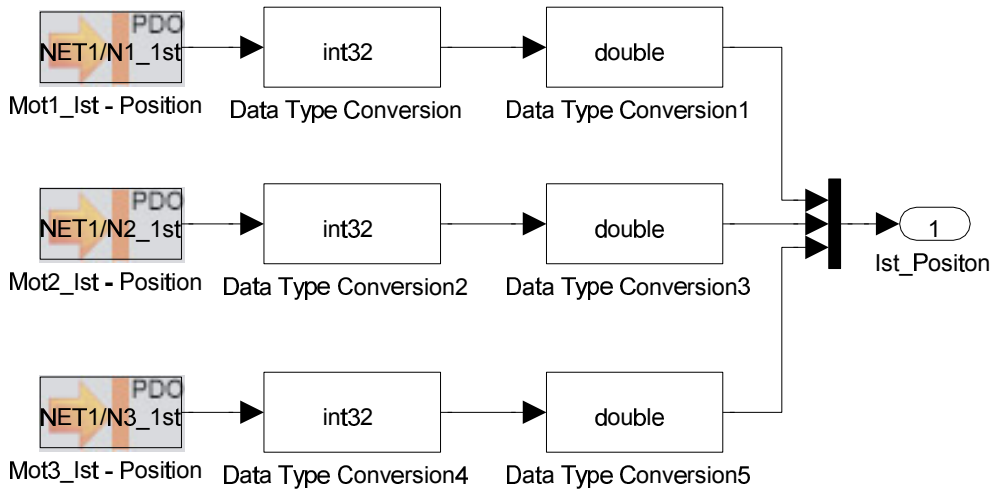


Abbildung 5.26: Subsystem: Ist-Position

Das Subsystem „Ist-Position“ (Abb. 5.26) leitet die Ist-Position der Motoren aus dem Trace an die Funktion weiter. Der Wert der Ist-Position, die der Motor sendet, hat den Datentype „uint32“. Bei diesem Datentyp sind Werte von 0 bis 4 294 967 295 zulässig. Für die Verarbeitung der Ist-Position in der Matlab Funktion entsteht bei dem Übergang an der 0-Position in den negativen Bereich ein Problem bei der Berechnung. Um dieses Problem zu umgehen, wird die Ist-Position mit dem Block „Data Type Conversion“ in den Datentyp „int32“ umgewandelt. Dadurch wird der Wertebereich nun von $-2\,147\,483\,648$ bis $2\,147\,483\,647$ gesetzt. Diese Umwandlung verschiebt das Problem auf den Übergang an den beiden neuen Enden. Dieser Übergang liegt aber nicht im Arbeitsraum des Roboters und führt daher zu keinen Problemen. Anschließend muss der Wert mit einem weiteren „Data Type Conversion“ Block in den Datentype „double“ konvertiert werden, damit die Matlab Funktion den Wert verarbeiten kann.
Durch das Verbinden des Simulinkmodells mit der Bachmann Steuerung ist es möglich in Echtzeit auf die Steuerung zuzugreifen. Mit dem Betätigen der Schalter „Motoren Hauptschalter“, „Positionierung“, „Bewegung“ „Korrektur berechnen“ und „Korrektur vornehmen“ werden verschiedene Routinen in der Matlab Funktion aktiviert und deaktiviert. Diese Routinen werden in dem Kapitel 5.22 beschrieben. Mit dem Schalter „Motoren Hauptschalter“ werden die Motoren des Roboters ein- und ausgeschaltet, wobei über das Subsystem „Motoren An/Aus“ (Abb.5.27) die Motoren einzeln ein- und ausgeschaltet werden können.

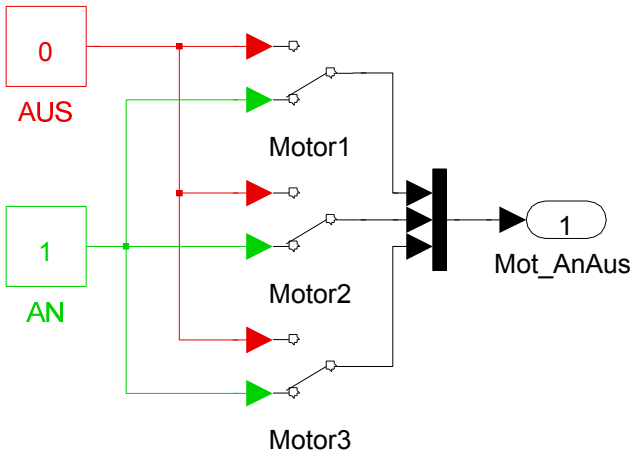


Abbildung 5.27: Subsystem: Motoren An/Aus

Durch Betätigen des Schalters „Positionierung“ wird der Endeffektor auf die Position, die im Subsystem „Parameter“ eingegeben wurden, gefahren. Anschließend kann mit dem Schalter „Bewegung“ eine Kreisbahn mit dem Roboter ausgeführt werden. Alternativ dazu kann man die in 4.2 beschriebene Korrektur durchführen, indem zuerst der Schalter „Korrektur berechnen“ und danach der Schalter „Korrektur vornehmen“ betätigt werden.

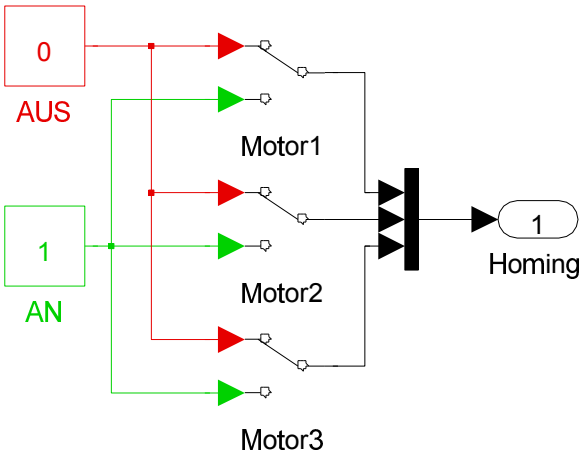


Abbildung 5.28: Subsystem: Homing

Das Subsystem „Homing“ dient der Kalibrierung der Motoren. Bei eingeschalteten Motoren wird durch Betätigen der Schalter im Subsystem (Abb. 5.28) die Postion der jeweiligen Motoren auf einen vordefinierten Wert gesetzt. Hierzu müssen die ausgeschalteten Motoren in die dafür vorgesehen Positionen gebracht werden.
Die beiden letzten Subsysteme „Befehl Übermittlung“ (Abb. 5.30) und „Trace Konfiguration“ (Abb. 5.31) sind für die Übermittlung der Befehle an die Motoren verantwortlich. Die

Übermittlungen der Variablen über den CAN erfolgt über die Funktionsblöcke „CAN-PDO-Write“. Die Einstellungen der Funktionsblöcke erfolgt, wie die Einstellung Funktionsblöcke „CAN-PDO-Read“, über die Masken. Die Maske eines „CAN-PDO-Write“ Funktionsblocks ist in Abbildung 5.29 abgebildet.

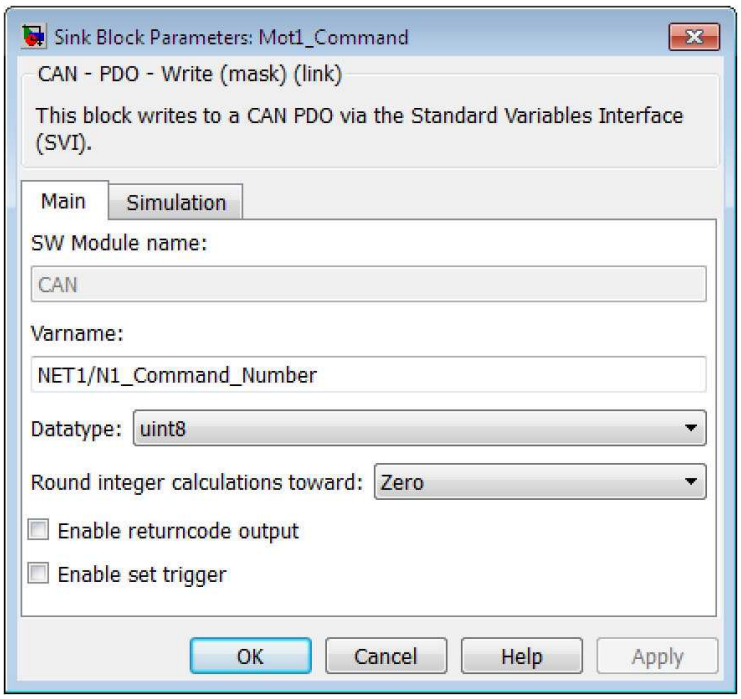


Abbildung 5.29: Maske eines CAN-PDO-Write Funktionsblocks

Die Einstellung des Variablennamen in „Varname“ erfolgt analog zu den Funktionsblöcken „CAN-PDO-Read“. Des Weiteren muss hier der Datentyp der Variablen festgelegt werden und die „Sample time“ auf „0,001“ gesetzt werden.

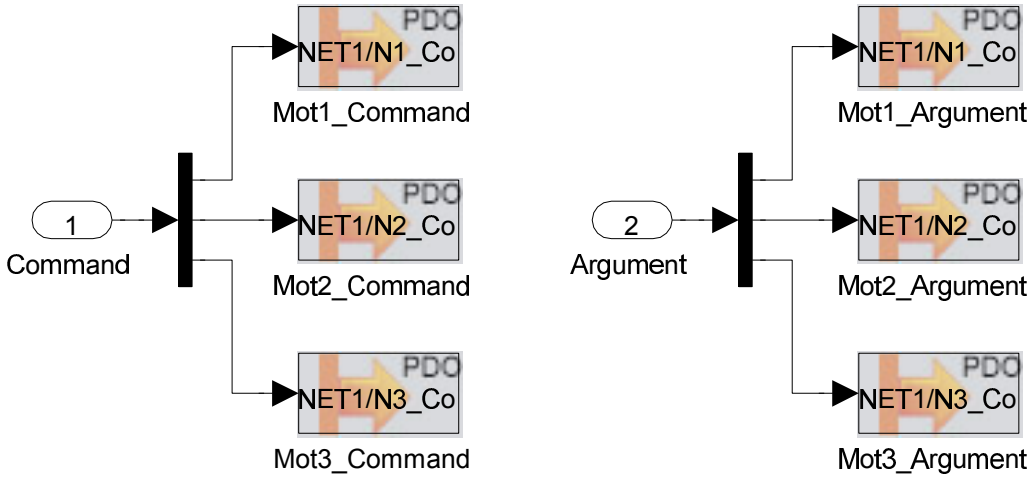


Abbildung 5.30: Subsystem: Befehl Übertragung

In dem Subsystem „Befehl Übermittlung“ werden die in den Routinen erzeugten Befehle und Argumente an die Motoren gesendet.

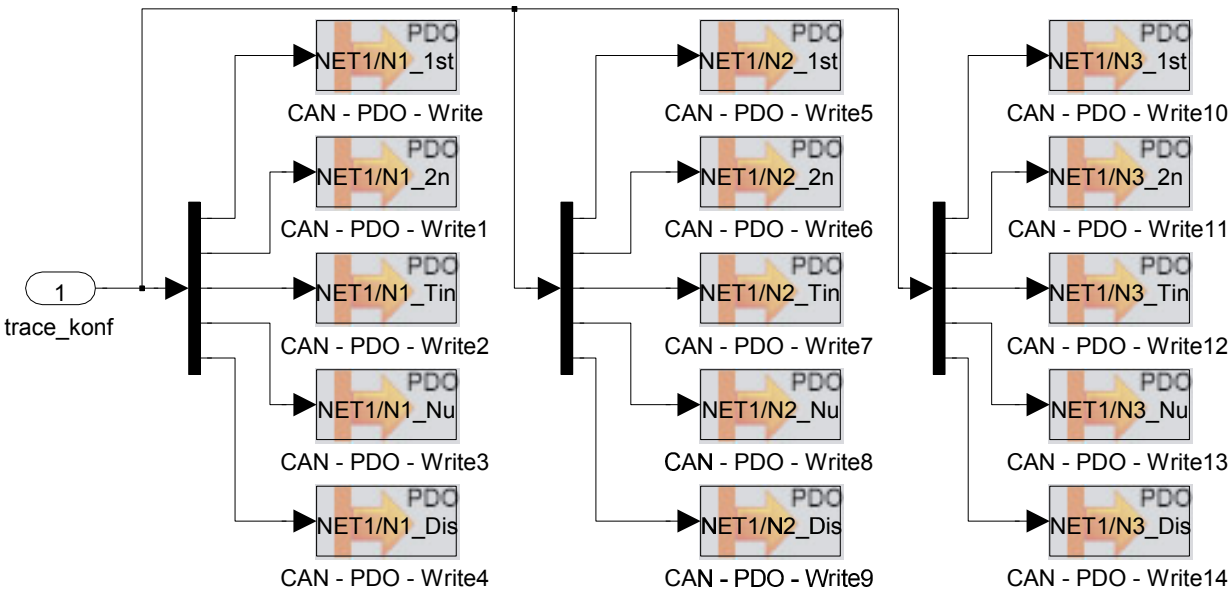


Abbildung 5.31: Subsystem: Trace Konfiguration

Über das Subsystem „Trace Konfiguration“ wird das Trace der Motoren konfiguriert. Die Konfiguration des Traces erfolgt in der Matlab Funktion und ist in Kapitel 5.22 beschrieben.

5.2.1 Steuerungsprogramm

In diesem Kapitel wird der Ablauf des Steuerungsprogramms beschrieben. Nachdem das Simulinkmodell auf die Steuerung geladen wurde, startet dieses automatisch. Anschließend muss Simulink mit der Steuerung verbunden werden, damit eine Bedienung des Steuerungsprogramms über die „Manual Switches“ möglich ist. Nach dem Starten des Simulinkmodells auf der Steuerung, wird das Trace Konfigurations PDO an die Motoren, mit den in Kapitel 2.4 beschrieben Inhalt, gesendet. Somit erhält das Steuerungsprogramm über das Trace PDO in jedem Zeitschritt die Ist-Position des Roboters.
Um den Roboter in Betrieb zu nehmen, muss nach jedem Neustart der Steuerung die Motorenpositionen kalibriert werden. Hierzu müssen die ausgeschalteten Motoren nacheinander auf eine vordefinierte Position gebracht werden und dort wieder mit den Schaltern im Subsystem „Motor An/Aus“ eingeschaltet werden. Das Einschalten der Motoren geschieht mit dem Befehl „EN“ und das Ausschalten mit dem Befehl „DI“. Nach dem Einschalten eines Motors muss die Ist-Position, die von dem Motor gesendet wird, auf die bekannte Position gestellt werden. Dies geschieht über den Homing Befehl „HO“. Als Argument wird die eingestellte Position des Motors in Inkrementen (Ink) gesendet. Die Umrechnung des Winkels erfolgt nach folgender Gleichung:

$$Argument \quad = \quad q \cdot 66 \cdot \frac{3000\,\mathrm{Ink}}{360\,^\circ} \tag{5.1}$$

Da an dem Motor ein Getriebe mit einer Übersetzung ü = 66 montiert ist, wird der Winkel mit dieser Übersetzung multipliziert. Da eine Umdrehung von 360 ° des Motors 3000 Ink entspricht, wird anschließend mit 3000 Ink multipliziert und durch 360 ° geteilt. Der Befehl und das Argument werden an die Motoren gesendet, wenn der jeweilige „Manual Switch“ im Subsystem „Homing“ betätigt wird. Die Homing Positionen der drei Motoren sind in Tabelle 5.1 dargestellt.

Tabelle 5.1: Homing Positionen der Motoren

Motor	Winkel q[°]	Argument[Ink]
1	0	0
2	−90	−49500
3	90	49500

Nachdem die Kalibrierung der Motoren abgeschlossen ist, können alle Motoren eingeschaltet werden. Anschließend kann der Roboter mit dem Schalter „Positionierung“ auf die in dem Subsystem „Parameter“ definierte Position gefahren werden. Um den Roboter auf die Position zu fahren, werden nur die ersten beiden Motoren verwendet. Aus diesem Grund wird der dritte Motor durch das Steuerungsprogramm ausgeschaltet. Anschließend werden die beiden anderen Motoren durch den Befehl „V“ mit einer konstanten Geschwindigkeit

auf ihre Positionen gefahren. Nachdem die beiden Motoren ihre Positionen erreicht haben, wird der dritte Motor wieder eingeschaltet. Anschließend kann entweder mit dem Schalter „Bewegung“ eine Kreisbahn abgefahren werden oder mit den Schaltern „Korrektur berechnen“ und „Korrektur vornehmen“ die statische Position des Roboter korrigiert werden.
Zunächst einmal wird der Ablauf des Steuerungsprogramms beim Betätigen des Schalters „Bewegung“ beschrieben. Nach dem Betätigen des Schalters fährt der Roboter mit konstanter Geschwindigkeit vom Mittelpunkt des Kreises 30 mm auf einer Geraden in Richtung der x-Achse. Hierfür wird in jedem Zyklus der Steuerung die neue Winkelstellung der Motoren berechnet und als Argument mit dem Befehl „LA“ an die Motoren gesendet. Nach dem Erhalten des Befehls wird im nächsten Zyklus mit dem Befehl „M“ die Lageregelung aktiviert und die Positionierung gestartet. Nachdem die Gerade abgefahren wurde, befindet sich der Roboter auf der Kreisbahn und beginnt sich mit konstanter Geschwindigkeit gegen den Uhrzeigersinn auf einer Kreisbahn mit dem Radius $R = 30\,\text{mm}$ zu bewegen. Hierbei werden, wie bei der Geraden, die neuen Winkelstellungen der Motoren berechnet und an die Motoren gesendet. Anschließend wird die Lageregelung aktiviert und die Positionierung gestartet. Nachdem die Kreisbahn abgefahren wurde, verbleibt der Roboter auf der letzten Position.
Nach der Positionierung des Roboter kann auch die statische Position korrigiert werden. Hierbei wird die Korrekturmethode aus dem Simulink/SimMechanics Modell verwendet. Hierfür wird, wie in Kapitel 4.3, der Wert kk benötigt, der über das Subsystem „Parameter“ an das Steuerungsprogramm weitergeleitet wird. Nach dem Betätigen des Schalters „Korrektur berechnen“ werden die Winkel für die Korrektur berechnet. Für die Berechnung der Winkel werden die Motormomente benötigt, welche über die Motorströme berechnet werden. Der Motorstrom (I in mA) eines Motors wird über den Befehl „GRC“ abgefragt. Mit der Drehmomentkonstante $k_M = 20{,}2\,\text{mNm/A}$ der Motoren lässt sich das Motorenmoment M_M nach folgender Gleichung berechnen.

$$M_M = I \cdot k_M \tag{5.2}$$

Durch das Getriebe mit der Übersetzung ü = 66 ergibt sich das Moment M_A am Arm mit:

$$M_A = M_M \cdot 66 \tag{5.3}$$

Mit dem Moment M_A lässt sich der Winkel q_{kor} wie folgt berechnen.

$$q_{kor} = M_A \cdot kk \cdot \frac{360\,^\circ}{2 \cdot \pi\,\text{rad}} \tag{5.4}$$

Da der Motorstrom immer positiv ist, lässt dieser keinen Rückschluss auf die Richtung der Auslenkung zu. Dies wird mit der Funktion *ZugDruck* bestimmt. In dieser Funktion wird berechnet, ob die zweiten Armsegmente der Arme auf Zug oder Druck belastet werden. Hierfür benötigt die Funktion die Richtung der Belastung des Endeffektors und seine Position. In der Funktion werden die Winkel zwischen der Kraft und den zweiten Armsegmenten berechnet. Die Kraftwirkung erfolgt in negativer x-Richtung. Sind diese Winkel kleiner 90 ° werden die Armsegmente auf Druck belastet, bei Winkeln größer 90 ° handelt

es sich um eine Zugbelastung. Bei einer Zugbelastung der zweiten Armsegmente muss der Winkel q_{kor} zu der Position q_{soll} addiert werden und bei einer Druckbelastung subtrahiert werden.
Bei der Betätigung des Schalters „Korrektur vornehmen“ werden die neuen Positionen mit dem Befehl „LA“ gesendet. Die Argumente zu dem Befehlen werden bei einer Zugbelastung nach Gleichung 5.5 und bei einer Druckbelastung nach Gleichung 5.6 berechnet.

$$Argument = (q_{soll} + q_{kor}) \cdot 66 \cdot \frac{3000\,\mathrm{Ink}}{360\,^\circ} \tag{5.5}$$

$$Argument = (q_{soll} - q_{kor}) \cdot 66 \cdot \frac{3000\,\mathrm{Ink}}{360\,^\circ} \tag{5.6}$$

Nach der erfolgreichen Übermittlung der Befehle werden mit dem Befehl „M“ die Lageregelung aktiviert und die Positionierung gestartet.
Der gesamte Matlab Code des Steuerungsprogramms befindet sich im Anhang A.3.

5.2.2 Matlab Funktion zur Richtungsbestimmung der Korrektur

Mit der Funktion *ZugDruck* wird berechnet, ob die zweiten Armsegmente des Roboters auf Zug oder Druck belastet werden. Dies bestimmt die Richtungen der Auslenkungen der ersten Armsegmente. Für die Bestimmung der Belastungen wird das in Abbildung 5.32 dargestellte Modell verwendet.

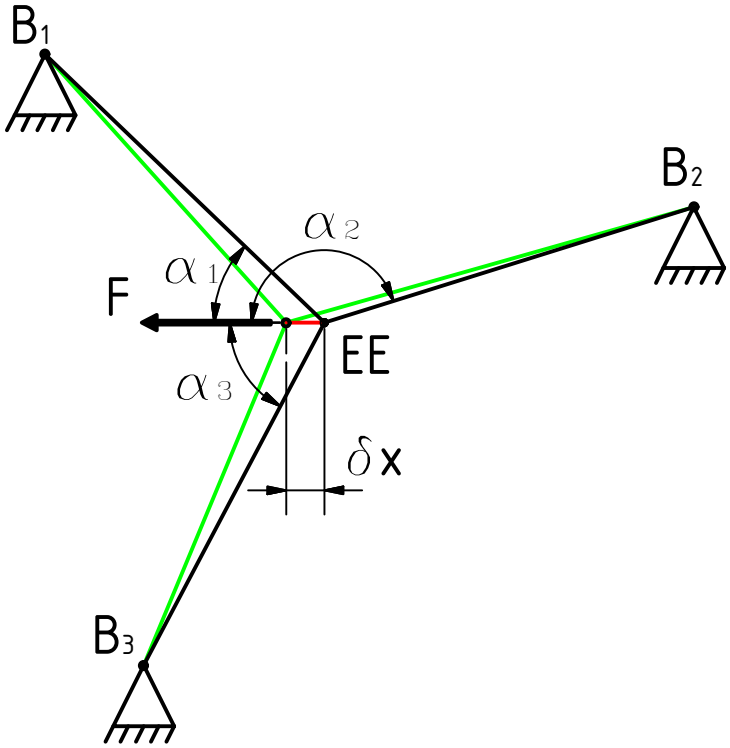


Abbildung 5.32: Modell zur Bestimmung der Kraftrichtungen

In diesem Modell sind die zweiten Armsegmente mit dem Endeffektor abgebildet. Die Mittelgelenke B_i sind als drehbare Festlager modelliert. Die Postionen der Mittelgelenke und die des Endeffektors sind bekannt. Nun werden mit Hilfe einer fiktiven infinitesimalen Verschiebung δx in Kraftrichtung die Belastungsrichtungen der Armsegmente bestimmt. Die Verschiebung δx ist in der Skizze rot gekennzeichnet. Die aus der Verschiebung resultierenden Armsegmente sind grün dargestellt. Die Belastungsrichtung der Armsegmente wird darüber bestimmt, ob die resultierende Armsegmentlänge h kleiner oder größer ist als die Ausgangslänge L_{Arm}. Ist die Länge kleiner, wird das Armsegment auf Druck belastet, anderenfalls auf Zug. Die resultierenden Armlängen h_i lassen sich mit dem Kosinussatz wie folgt berechnen:

$$h_i^2 = L_{Arm}^2 + \delta x^2 - 2 \cdot \delta x \cdot L_{Arm} \cdot cos\,(\alpha_i) \tag{5.7}$$

Die Winkel α_i werden über die Geometrie nach der Gleichung 5.8 bestimmt. Diese Gleichung liefert einen Wert zwischen $0\,^\circ$ und $180\,^\circ$.

$$\alpha_i = \arccos\left(\frac{\overrightarrow{EEB_i} \cdot \overrightarrow{F}}{\left|\overrightarrow{EEB_i}\right| \cdot \left|\overrightarrow{F}\right|}\right) \tag{5.8}$$

Die Bedingung für eine Zugbelastung lautet:

$$h_i > L_{Arm} \tag{5.9}$$

Aus der Bedingung 5.9 und der Gleichung 5.7 folgt:

$$\sqrt{L_{Arm}^2 + \delta x^2 - 2 \cdot \delta x \cdot L_{Arm} \cdot cos\,(\alpha_i)} > L_{Arm} \tag{5.10}$$

Nach dem Quadrieren der Ungleichung 5.10 erhält man die Ungleichung 5.11, die wie folgt umgeformt wird.

$$L_{Arm}^2 + \delta x^2 - 2 \cdot \delta x \cdot L_{Arm} \cdot cos\,(\alpha_i) > L_{Arm}^2 \tag{5.11}$$

$$\delta x^2 - 2 \cdot \delta x \cdot L_{Arm} \cdot cos\,(\alpha_i) > 0 \tag{5.12}$$

$$\delta x^2 > 2 \cdot \delta x \cdot L_{Arm} \cdot cos\,(\alpha_i) \tag{5.13}$$

$$\frac{\delta x^2}{2 \cdot \delta x \cdot L_{Arm}} > cos\,(\alpha_i) \tag{5.14}$$

Da δx eine infinitesimal kleine Verschiebung ist, folgt aus der Ungleichung 5.14:

$$\lim_{\delta x \to 0} \frac{\delta x^2}{2 \cdot \delta x \cdot L_{Arm}} = 0 > cos\,(\alpha_i) \tag{5.15}$$

Da α_i nach Gleichung 5.8 nur Werte zwischen $0\,^\circ$ und $180\,^\circ$ annehmen kann, liegt die Lösung der Ungleichung 5.15 für α_i zwischen $90\,^\circ$ und $180\,^\circ$. Bei einem Winkel von α_i zwischen $0\,^\circ$ und $90\,^\circ$ handelt es sich um eine Druckbelastung. Diese Berechnung gilt nur, solange durch

die Auslenkung des Endeffektors die Winkel den Zug- bzw. Druckbereich nicht wechseln. Hierauf ist bei der Anwendung am Roboter zu achten. Im folgenden wird die Funktion *ZugDruck* beschrieben. Der Quellcode der Funktion befindet sich im Anhang A.10.
Als Eingabewerte werden für die Funktion die Armlänge L_arm, die Kraft F als Vektor, die Soll-Koordinaten XEE und YEE des Endeffektors, sowie die Position der Antriebe A_1, A_2 und A_3. Die Funktion gibt den Wert *ZugDruck* zurück.

```
function [ZugDruck] = ZugDruck(F,XEE,YEE,A1,A2,A3,L_arm)
```

Zuerst wird die Ausgabevariable *ZugDruck* initialisiert und die Soll-Koordinaten in einen Vektor EE geschrieben.

```
ZugDruck = zeros(3,1)
EE = [XEE YEE 0];
```

Anschließend werden die Winkelpositionen der Arme mit der Funktion *inv_Kin_3RRR_oM* berechnet.

```
[q] = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,XEE,YEE);
```

Danach werden die Positionsvektoren der drei Zwischengelenke initialisiert und berechnet.

```
B1 = zeros(1,3);
B2 = zeros(1,3);
B3 = zeros(1,3);

B1(1) = A1(1)+L_arm*cosd(q(1));
B1(2) = A1(2)+L_arm*sind(q(1));
B1(3) = 0;

B2(1) = A2(1)+L_arm*cosd(q(2));
B2(2) = A2(2)+L_arm*sind(q(2));
B2(3) = 0;

B3(1) = A3(1)+L_arm*cosd(q(3));
B3(2) = A3(2)+L_arm*sind(q(3));
B3(3) = 0;
```

Daraufhin werden die Winkel *alpha* zwischen dem Kraftvektor F und den Armsegmenten berechnet.

```
alpha=[acosd((B1-EE)*F'/(norm(B1-EE)+norm(F)))
       acosd((B2-EE)*F'/(norm(B2-EE)+norm(F)))
       acosd((B3-EE)*F'/(norm(B3-EE)+norm(F)))];
```

Zuletzt werden für Winkel größer gleich 90 ° der jeweilige Wert in der Variablen $ZugDruck$ gleich „1“ gesetzt, für Winkel kleiner 90 ° wird der Wert „-1“ eingetragen. Der Wert „1“ steht für eine Zugbelastung, bei dem Wert „-1“ handelt es sich um eine Druckbelastung.

```
ZugDruck(find(alpha≥90)) = 1;
ZugDruck(find(alpha<90)) = −1;
```

6 Positionsbestimmung des Roboters

Für die Bestimmung der Position wird die Logitech HD Webcam C615 verwendet. Prinzipiell wir mit dieser Kamera ein Bild aufgenommen und die Position einer roten Kugel, die sich auf dem Endeffektor befindet, anhand der Pixelmatrix bestimmt. Die Kamerakoordinaten [Pixel] werden mit dem Abbildungsgesetz, welches in Kapitel 6.1 näher beschrieben wird, in globale Koordinaten [mm] überführt. Hierzu gehört eine entsprechende Kalibrierung des Systems, zur Bestimmung der Koeffizienten des Abbildungsgesetzes. Die Grundvoraussetzung für die Anwendung des Abbildungsgesetzes ist die Positionierung der Kamera. Diese muss so hoch oberhalb des Roboters angebracht werden, dass sie den gesamten Arbeitsraum des Roboters erfasst und sie muss parallel zu der Grundplatte ausgerichtet sein. Anschließend wird in Kapitel 6.2 die Genauigkeit der Messmethode überprüft. Des Weiteren wird die Funktionsweise von drei Graphical User Interfaces (GUIs) in Kapitel 6.4 beschrieben, welche zur Nutzung der Kamera als Positionsmesser benötigt werden.

6.1 Berechnungen zur Bestimmung der Position

Um die Position des Endeffektors aus den Kamerabildern zu bestimmen, werden die Funktionen f und g in den Gleichungen 6.1 und 6.2 gesucht, mit denen die globalen Koordinaten x_G und y_G aus den Kamerakoordinaten x_K und y_K berechnet werden.

$$x_G = f(x_K, y_K) \tag{6.1}$$

$$y_G = g(x_K, y_K) \tag{6.2}$$

Als Grundlage der Positionsbestimmung dient das Abbildungsgesetz, welches im Folgenden erklärt wird.

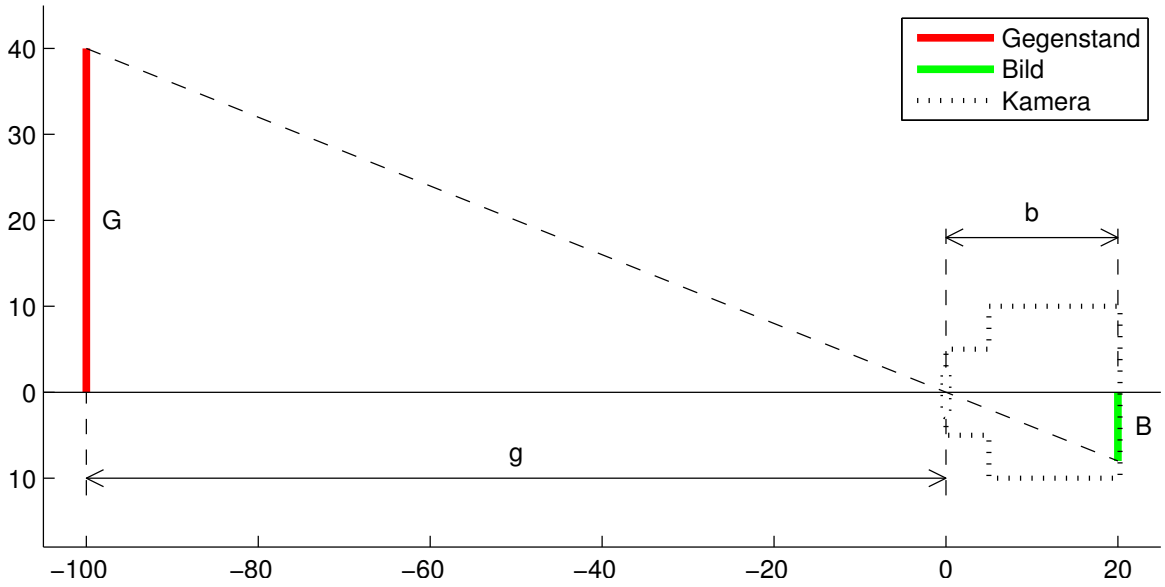


Abbildung 6.1: Abbildungsgesetz

Wie in Abbildung 6.1 zu erkennen ist, lautet das Abbildungsgesetz:

$$\frac{G}{g} = \frac{B}{b} \tag{6.3}$$

Daraus folgt:

$$G = \frac{g}{b} \cdot B \tag{6.4}$$

Da für die Positionsbestimmung eine rote Kugel mit einem Durchmesser $d = 40\,\text{mm}$ verwendet wird, ergibt sich bei der Projektion ein Fehler. Dies wird in der Abbildung 6.2 verdeutlicht. Da die Kugel ein dreidimensionaler Körper ist, stimmt der Mittelpunkt x_p der projizierten Fläche (blauer Bereich) der Kugel nicht mit dem Mittelpunkt x_m der Kugel überein. Hierdurch entsteht der Fehler z.

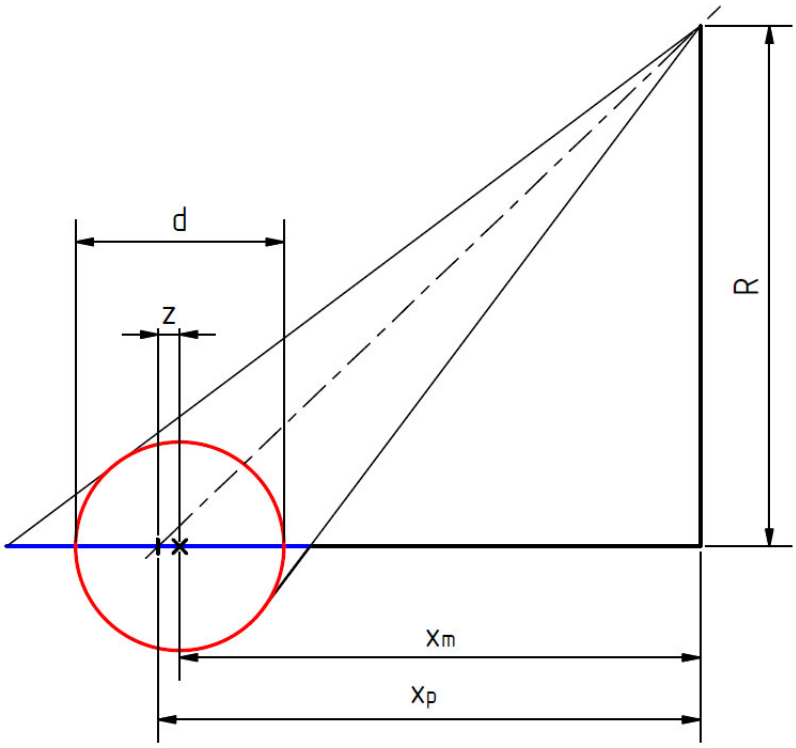


Abbildung 6.2: Skizze zur Bestimmung des Fehlers

Um den Grad des Fehlers zu bestimmen, wird der Verlauf des Fehlers z über den Abstand des Mittelpunktes x_m der Kugel bestimmt. Der Abstand der Kamera zur Arbeitsebene beträgt hierbei $R = 80\,\text{cm}$. Hierzu wurden die Punkte in der Tabelle 6.1 mit Hilfe einer Skizze in *Autodesk Inventor Professional 2011* bestimmt und in Abbildung 6.3 dargestellt.

Tabelle 6.1: Daten des Projektionsfehlers

x_m[mm]	z[mm]	x_m/z
0	0	-
100	0,06254	1598,98
200	0,12508	1598,98
300	0,18762	1598,98
400	0,25016	1598,98
500	0,3127	1598,98

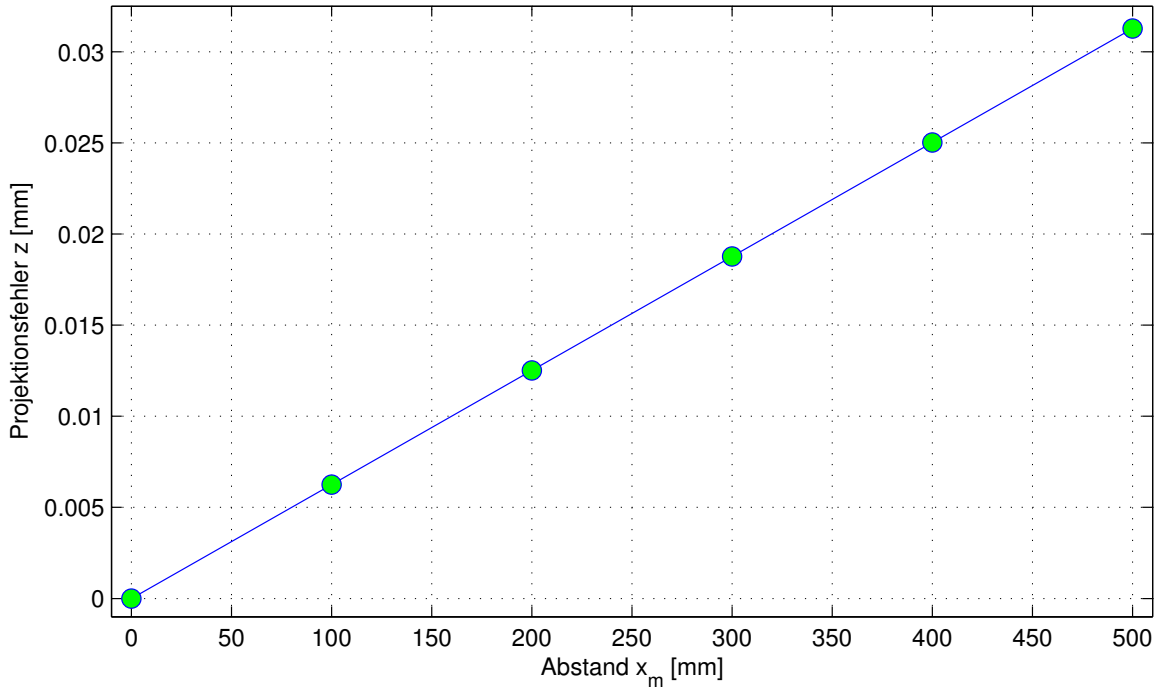


Abbildung 6.3: Verlauf des Projektionsfehlers z über den Abstand des Projektionsmittelpunktes x_m

Wie in der Tabelle 6.1 und der Abbildung 6.3 zu erkennen ist, ist der Fehler z linear zum Mittelpunkt x_m der Kugel. Somit muss die Gleichung 6.4 um einen Fehlerteil erweitert werden.

$$G = \frac{g}{b} \cdot B + z \tag{6.5}$$

Da der Fehler z linear zum Abstand ist, kann man den Fehler durch einen linearen Ansatz $z = a \cdot B$ ersetzen, daraus folgt:

$$G = \frac{g}{b} \cdot B + a \cdot B \tag{6.6}$$

$$G = \frac{g + a \cdot b}{b} \cdot B \tag{6.7}$$

Wie in Gleichung 6.7 zu sehen ist, ist die Umrechnung mit dem Projektionsfehler linear. Um die Kamera-Koordinaten in globale Koordinaten umzurechnen, wird sowohl für die x_G-Koordinate als auch für die y_G-Koordinate ein linearer Ansatz gewählt. Dieser besteht aus einer Konstanten, einem linearen x_K und y_K Anteil und einem gemischten Term aus x_K und y_K.

$$x_G = a_0 + a_1 x_K + a_2 y_K + a_3 x_K y_K \tag{6.8}$$

$$y_G = b_0 + b_1 x_K + b_2 y_K + b_3 x_K y_K \tag{6.9}$$

Um die Koeffizienten aus den Gleichungen 6.8 und 6.9 zu bestimmten, werden vier Punkte im Messgebiet, bei denen die globalen Koordinaten bekannt sind, gemessen. Dadurch erhält man folgende zwei Gleichungssysteme mit je vier Gleichungen und vier Unbekannten.

$$\begin{bmatrix} x_{1G} \\ x_{2G} \\ x_{3G} \\ x_{4G} \end{bmatrix} = \begin{bmatrix} 1 & x_{1K} & y_{1K} & x_{1K}y_{1K} \\ 1 & x_{2K} & y_{2K} & x_{2K}y_{2K} \\ 1 & x_{3K} & y_{3K} & x_{3K}y_{3K} \\ 1 & x_{4K} & y_{4K} & x_{4K}y_{4K} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{6.10}$$

$$\begin{bmatrix} y_{1G} \\ y_{2G} \\ y_{3G} \\ y_{4G} \end{bmatrix} = \begin{bmatrix} 1 & x_{1K} & y_{1K} & x_{1K}y_{1K} \\ 1 & x_{2K} & y_{2K} & x_{2K}y_{2K} \\ 1 & x_{3K} & y_{3K} & x_{3K}y_{3K} \\ 1 & x_{4K} & y_{4K} & x_{4K}y_{4K} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{6.11}$$

Durch Multiplikation mit den Inversen der Matrizen von links erhält man die Koeffizienten der Abbildungsfunktionen nach folgenden Gleichungssystemen.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = inv \begin{bmatrix} 1 & x_{1K} & y_{1K} & x_{1K}y_{1K} \\ 1 & x_{2K} & y_{2K} & x_{2K}y_{2K} \\ 1 & x_{3K} & y_{3K} & x_{3K}y_{3K} \\ 1 & x_{4K} & y_{4K} & x_{4K}y_{4K} \end{bmatrix} \cdot \begin{bmatrix} x_{1G} \\ x_{2G} \\ x_{3G} \\ x_{4G} \end{bmatrix} \tag{6.12}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = inv \begin{bmatrix} 1 & x_{1K} & y_{1K} & x_{1K}y_{1K} \\ 1 & x_{2K} & y_{2K} & x_{2K}y_{2K} \\ 1 & x_{3K} & y_{3K} & x_{3K}y_{3K} \\ 1 & x_{4K} & y_{4K} & x_{4K}y_{4K} \end{bmatrix} \cdot \begin{bmatrix} y_{1G} \\ y_{2G} \\ y_{3G} \\ y_{4G} \end{bmatrix} \tag{6.13}$$

Mit Hilfe einer linearen Regression können zusätzliche Messpunkte zur Bestimmung der Koeffizienten verwendet werden. Durch diese Methode wird die Genauigkeit der Abbildungsfunktionen erhöht. Mit der linearen Regression wird nicht das Gleichungssystem nach den Koeffizienten aufgelöst, sondern unter Berücksichtigung der Methode der kleinsten Quadrate werden die Fehler zu den Messpunkten minimiert. Die lineare Regression mit vier Messpunkten würde dem oben genannten Berechnungsweg als Spezialfall entsprechen.
Nachdem die Kalibrierung durch die Berechnung der Koeffizienten erfolgt ist, können mit den Gleichungen 6.8 und 6.9 die globalen Koordinaten des Endeffektors berechnet werden. Dies geschieht in der Matlab Funktion *Cam2Glob*, die im Kapitel 6.3.2 erklärt wird.

6.2 Bestimmung der Genauigkeit

Um die Genauigkeit der Positionsbestimmung zu ermitteln, müssen von verschiedenen Punkten im Arbeitsraum die Kamerakoordinaten aus den Bildern berechnet werden. Zu diesen Positionen müssen die globalen Koordinaten der Ist-Positionen in der Ebene des Endeffektors gemessen werden. Hierzu wurden 16 Messungen im Arbeitsraum vorgenommen. Anhand dieser Messungen wurden die Koeffizienten für die Berechnung der Koordinaten ermittelt, diese stehen in der Tabelle 6.2.

Tabelle 6.2: Koeffizienten

	Koeffizienten der Abbildungsfunktionen			
	0.[mm]	1.[mm/Pixel]	2.[mm/Pixel]	3.[mm/Pixel²]
a_i	$663,68$	$-0,0140$	$-0,8263$	$9,107 \cdot 10^{-6}$
b_i	$744,65$	$-0,8307$	$-0,0017$	$3,032 \cdot 10^{-5}$

Außerdem wird die berechnete Position mit der Ist-Position verglichen und die Abweichung Δ berechnet. Die Messwerte und Abweichungen sind in Tabelle 6.3 zusammengefasst.

Tabelle 6.3: Messungen

Ist-Positon		Kamera Position		Messung		Abweichung
x[mm]	y[mm]	x[Pixel]	y[Pixel]	x[mm]	y[mm]	Δ[mm]
240	339	495.98	507.57	239.6	339.43	0.59
240	403	417.87	508.37	239.68	403.13	0.34
240	467	339.24	509	239.9	467.24	0.26
240	531	261.12	509.7	240.06	530.93	0.09
352	339	494.88	370.59	352.18	338.5	0.53
352	403	416.74	371.65	352.15	402.54	0.48
352	467	338.72	373.18	351.71	466.49	0.58
352	531	259.87	374.19	351.72	531.1	0.30
464	339	492.13	233.56	464.83	338.94	0.83
464	403	414.05	234.71	464.81	403.26	0.85
464	467	336.59	235.93	464.73	467.07	0.73
464	531	258.67	237.45	464.4	531.25	0.47
576	339	490.1	99.36	575.14	338.84	0.87
576	403	412.03	100.01	575.63	403.46	0.59
576	467	334.91	100.99	575.84	467.3	0.34
576	531	258.54	102.49	575.61	530.51	0.63

Wie in der Tabelle zu sehen ist, liegt die Abweichung Δ bei der Messung unter 0,9 mm. Der Mittelwert mit Standartabweichung beträgt $S = 0{,}52\,\text{mm} \pm 0{,}25\,\text{mm}$.
Anhand der Abbildungsfunktionen f und g kann der Umrechnungsfaktor λ von Pixel in mm berechnet werden. Um den Umrechnungsfaktor zu bestimmen werden die Abstände von zwei unterschiedlichen Punkten, sowohl in den Kamerakoordinaten als auch in globalen

Koordinaten, berechnet. Für die Abstände in Kamerakoordinaten gilt:

$$\Delta x_K = x_{K2} - x_{K1} \tag{6.14}$$
$$\Delta y_K = y_{K2} - y_{K1} \tag{6.15}$$

Aus den Kamerakoordinaten mit den Abbildungsfunktionen f und g folgt für die Abstände in den globalen Koordinaten:

$$\Delta x_G = f(x_{K1}, y_{K1}) - f(x_{K2}, y_{K2}) \tag{6.16}$$
$$\Delta y_G = g(x_{K1}, y_{K1}) - g(x_{K2}, y_{K2}) \tag{6.17}$$

Mit den Abständen aus den Gleichungen 6.14, 6.15, 6.16 und 6.17 lässt sich der Umrechnungsfaktor λ berechnen.

$$\lambda = \sqrt{\frac{(\Delta x_G)^2 + (\Delta y_G)^2}{(\Delta x_K)^2 + (\Delta y_K)^2}} = 0{,}8363\,\text{mm/Pixel} \tag{6.18}$$

Wie in Gleichung 6.18 zu sehen beträgt der Umrechnungsfaktor $\lambda = 0{,}8363\,\text{mm/Pixel}$. Da der Umrechnungsfaktor λ größer ist als der berechnete Mittelwert mit Standartabweichung $S = 0{,}52\,\text{mm} \pm 0{,}25\,\text{mm}$, wird die Genauigkeit der Positionsbestimmung durch die Auflösung der Kamera eingeschränkt, da nur ganze Pixel gemessen werden können. Deshalb liegt der Messfehler mindestens bei der Pixelgenauigkeit von $S_P = 0{,}8363\,\text{mm}$ und wird im Folgenden als Messfehler angenommen.

6.3 Matlab Funktionen zur Berechnung der Postion

Für die Positionsbestimmung mit einer Kamera wurden in Matlab zwei Funktionen geschrieben. Mit der ersten Funktion *fncBildPos_HSV* (Kapitel 6.3.1) wird die Position der roten Kugel auf dem Endeffektor in einem Bild bestimmt. Mit den Koordinaten dieser Postion werden dann mit der Funktion *Cam2Glob* (Kapitel 6.3.2) die globalen Koordinaten des Endeffektors berechnet.

6.3.1 Matlab Funktion zur Bestimmung der Kamerakoordinaten

Die Funktion *fncBildPos_HSV* dient zur Bestimmung der Position einer roten Kugel in einem von der Kamera aufgenommenen Bild.
Wie in der ersten Zeile zu sehen ist, benötigt die Funktion als Eingabe die Variable *Data*, diese enthält ein Bild im RGB Format. Als Rückgabe der Funktion erhält man die Position der Mitte in der Variablen *Mitte*. Die Variable *z* gibt an, ob sich eine Kugel im Bild befindet und die Variable *d* ist der Durchmesser der Kugel in Pixel.

```
function [Mitte,z,d] = fncBildPos_HSV(data)
```

In den nächsten Zeilen werden die Ausgangsvariablen initialisiert und die Breite und Höhe des Bildformates ermittelt.

```
z=0;
d=0;
Mitte = zeros(1,2);
[Hoehe Breite temp] = size(data);
```

In dem nächsten Block wir zunächst das Format des Bildes von RGB in HSV geändert. HSV steht für **H**UE, **S**ATURATION und **V**ALUE, was für Farbton, Sättigung und Wert steht. Der Farbton wird in der *hPlane* gespeichert mit einem Bereich von 0 ° bis 360 °. Der Farbton wird in einem Farbkreis angegeben, wobei 0 ° und 360 ° reinem Rot, 120 ° reinem Grün und 240 ° reinem Blau entspricht. Die Sättigung sagt aus wie kräftig eine Farbe dargestellt wird. Diese Werte werden in der *sPlane* gepeichert und gehen von 0 bis 1, wobei 0 Weiß und 1 die reine Farbe bedeutet. Der Wert einer Farbe entspricht der Helligkeit der Farbe, dieser Wert wird in dieser Funktion nicht verwendet. Man kann sich das HSV-Farbmodell als einen Zylinder vorstellen, bei dem der Winkel den Farbton angibt, der radiale Abstand die Sättigung und die Höhe die Helligkeit, wie in Abbildung 6.4 zu sehen.

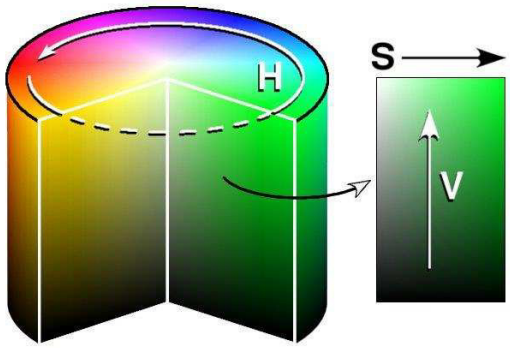


Abbildung 6.4: HSV-Farbmodell als Zylinder[1]

Anhand des Farbtons in der *hPlane* werden von Rot verschiedene Pixel gesucht, die im Bereich zwischen 20 ° und 350 ° liegen. Für die gefundenen Pixel wird die Sättigung in der *sPlane* auf 0 gesetzt. Dadurch besitzen nur noch die roten Pixel eine Sättigung die ungleich 0 ist. Anschließend werden mit der Funktion *bwareaopen* alle Objekte die kleiner *pixdel* sind entfernt. Der Wert *pixdel* ist ein Erfahrungswert und liegt bei 1/300 der Bildfläche.

[1]Quelle: http://commons.wikimedia.org/wiki/File:HSV_cylinder.jpg

```
hsvImage = rgb2hsv(data);           % Konvertieren in HSV Raum
hPlane = 360.*hsvImage(:,:,1);      % Skalieren von 0 bis 360
sPlane = hsvImage(:,:,2);           % Sättigung
nonRedIndex = (hPlane > 20) & ...   % Auswahl der nicht roten Pixel
              (hPlane < 350);
sPlane(nonRedIndex) = 0;            % Sättigung für nicht rote Pixel auf 0
pixdel = round(Hoehe*Breite/300);
bw = bwareaopen(sPlane,pixdel);     % alle Objekte kleiner pixdel entfernen
```

In dem letzten Abschnitt wird die Position der Kugel bestimmt. Hierfür werden die Koordinaten der roten Pixel mit dem Befehl *find(bw)* in x und y gespeichert und anhand der Länge dieser Vektoren wird die Pixelfläche A des Kreises berechnet. Ist die Pixelfläche A ungleich 0 befindet sich die rote Kugel im Bild und es kann durch Bildung der Mittelwerte der Koordinaten der roten Pixel die Position der Kugel berechnet und in der Variablen *Mitte* gespeichert werden. Da eine Kugel gefunden wurde, wird z gleich 1 gesetzt und der Durchmesser d anhand der Fläche A berechnet.

```
% Mittelpunkt bestimmen
[y x] = find(bw);
A = length(x);
if A ≠ 0
    xx = mean(x);
    yy = mean(y);
    z = 1;
    Mitte = [xx yy];
    d = sqrt(4*A/pi);
end
```

Der Quellcode der Funktion befindet sich im Anhang A.8

6.3.2 Matlab Funktion zur Berechnung der globalen Koordinaten

Mit der Funktion *Cam2Glob* werden die Kamerakoordinaten in globale Koordinaten umgewandelt.

```
function [X Y] = Cam2Glob(x,y,Dateiname)
Dateiname = [Dateiname '.mat'];
load(Dateiname)
c = ones(length(x),1);
X = [c, x, y, x.*y]*a;
Y = [c, x, y, x.*y]*b;
```

Als Eingabe benötigt die Funktion die x und y Koordinaten in Kamerakoordinaten und den Namen der Kalibrierungsdatei, in der sich die Koeffizienten a und b befinden. Anhand der Gleichungen 6.8 und 6.9 werden die globalen Koordinaten X und Y berechnet. Es

können mit dieser Funktion auch ganze Datensätze auf einmal umgewandelt werden. Der Quellcode der Funktion befindet sich im Anhang A.9.

6.4 Graphical User Interfaces

Um die Kamera für die Positionsmessung zu nutzen, wurden drei Graphical User Interface (GUIs) in Matlab programmiert. Das erste GUI dient der Kalibrierung der Kamera, mit der die Koeffizienten der linearen Regression berechnet werden. Die zweite GUI dient der Positionsmessung einzelner Punkte. Die dritte GUI bietet die Möglichkeit für ein bestimmtes Zeitintervall den Weg des Endeffektors aufzunehmen.

6.4.1 GUI: Kalibrierung

Mit der GUI Kalibrierung werden Messpunkte aufgenommen, mit denen die Koeffizienten durch eine lineare Regression berechnet werden. Abschließend werden die Koeffizienten in einer Datei für die Funktion *Cam2Glob* gespeichert.

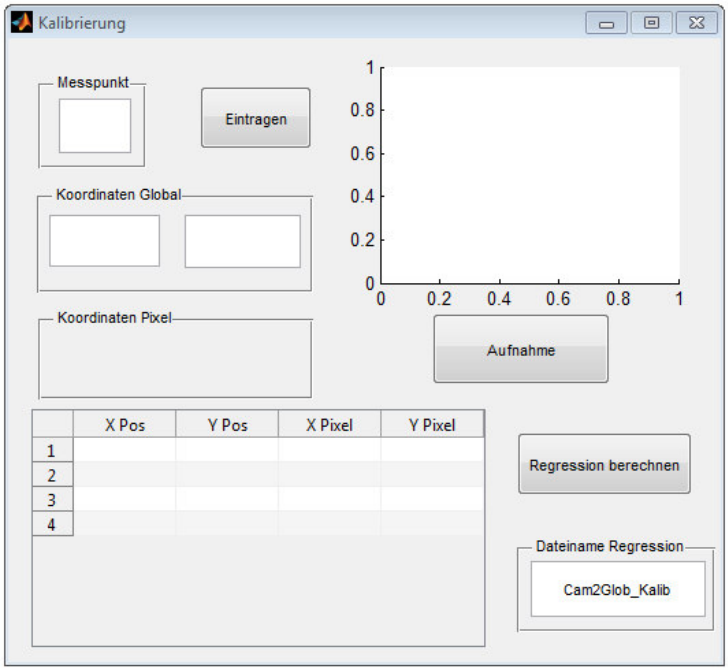


Abbildung 6.5: GUI Kalibrierung

Durch das Betätigen des Knopfes „Aufnahme“, werden mit der Kamera 50 Bilder mit einer Auflösung von 800x600 Pixel aufgenommen. In diesen Bildern wird die Position des Mittelpunktes der roten Kugel mit der Funktion *fncBildPos_HSV* berechnet und aus diesen

50 Positionen der Mittelwert gebildet. Der Mittelwert wird dann in dem Panel „Koordinaten Pixel“ angezeigt. Des Weiteren wird in der Grafik das letzte der 50 Bilder angezeigt und ein blaues Kreuz an der Stelle des ermittelten Kugelmittelpunktes gezeichnet. Der Benutzer kann somit die berechnete Position visuell überprüfen und gegebenenfalls eine neue Aufnahme starten. Ist die Position korrekt, müssen in den Feldern „Koordinaten Global“ die bekannten globalen Koordinaten der Kugel eingetragen werden. Nachdem dies geschehen ist, werden die Werte der Messung mit dem Knopf „Eintragen“ in die Tabelle eingetragen, woraufhin der Zähler „Messpunkt“ sich um eins erhöht. Diese Vorgehensweise muss mindestens viermal durchgeführt werden. Durch manuelle Veränderung des Feldes „Messpunkt“ können nachträglich Messungen wiederholt werden.
Nachdem genügend Messpunkte für die Kalibrierung erfasst wurden, wird durch Betätigung des Knopfes „Regression berechnen“ mit den in der Tabelle stehenden Werte eine lineare Regression durchgeführt und die damit berechneten Koeffizienten in der Datei mit dem Namen, des in dem Feld „Dateiname Regression“ befindlichen Textes, als .mat gespeichert. Mit dieser Kalibrierungsdatei kann die Funktion *Cam2Glob* Kamerakoordinaten in globale Koordinaten umwandeln.
Der Quellcode der GUI Kalibrierung befindet sich im Anhang A.11.

6.4.2 GUI: Positionsmessung

Die GUI Positionsmessung dient der Erfassung von einzelnen statischen Messpunkten des Endeffektors. Die Bedien- und Funktionsweise wird anhand der Abbildung 6.6 verdeutlicht.

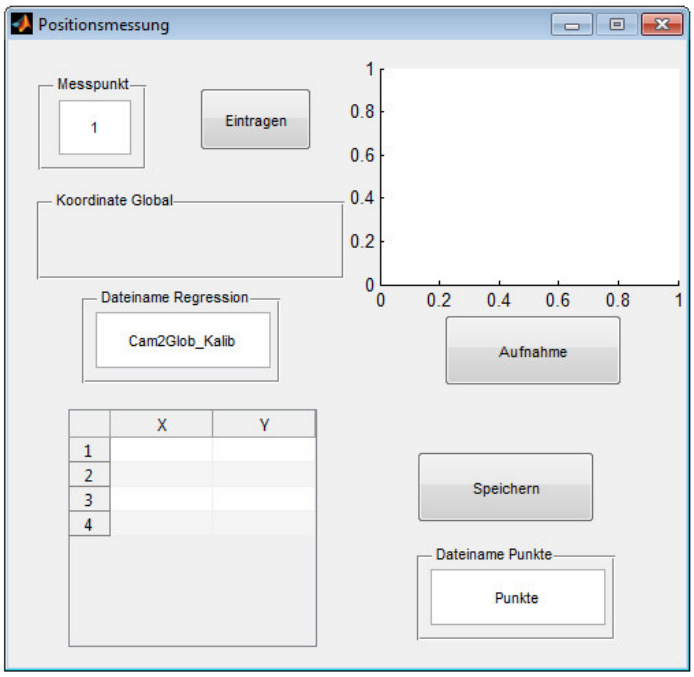


Abbildung 6.6: GUI Positionsmessung

Um eine einzelne Position zu ermitteln, wird der Knopf „Aufnahme“ betätigt, woraufhin mit der Kamera 10 Bilder aufgenommen werden. Anhand dieser Bilder werden, wie bei der Kalibrierung, mit der Funktion *fcnBildPos_HSV* die Mittelpunkte der Kugel in jedem Bild bestimmt und durch Mittelwertbildung die Kamerakoordinaten der Kugel berechnet. Mit den in der Datei mit dem Namen „Dateiname Regression“ gespeicherten Koeffizienten, die mit der GUI Kalibrierung 6.4.1 erzeugt wurden, werden unter Berücksichtigung der Abbildungsfunktionen (6.8 und 6.9) die globalen Koordinaten berechnet. Dies geschieht in der Funktion *Cam2Glob*. Die globalen Koordinaten werden dann in dem Panel „Koordinaten Global“ angezeigt. Außerdem wird das letzte der 10 Bilder in der Grafik angezeigt und der Mittelpunkt der Kugel mit einem blauen Kreuz markiert. Dies dient dem Benutzer zur visuellen Bestätigung der Berechnung. Anschließend kann der Benutzer den Messpunkt mit dem Knopf „Eintragen“ in die Tabelle übernehmen. Sind alle gewünschten Messungen vorgenommen, können mit dem Knopf „Speichern“ die in der Tabelle befindlichen Messungen in einer .mat Datei gespeichert werden. Der Name der Datei kann in dem Textfeld „Dateiname Punkte“ eingetragen werden.
Im Anhang A.12 befindet sich der Quellcode der GUI Positionsmessung.

6.4.3 GUI: Wegmessung

Durch die GUI Wegmessung kann für eine bestimmte Dauer die Position des Endeffektors aufgezeichnet werden.

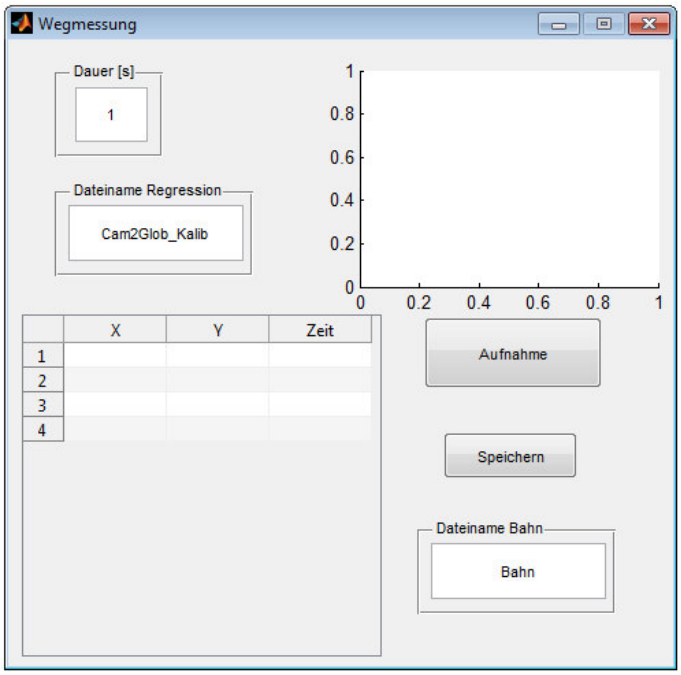


Abbildung 6.7: GUI Wegmessung

Der Benutzer kann in dem Textfeld „Dauer [s]“ die Aufnahmedauer der Messung einstellen. Die Aufnahme des Weges wird mit dem Betätigen des Knopfes „Aufnahme“ gestartet. Nachdem der Knopf betätigt wurde, läuft ein Countdown von 5 herunter, bevor die Kamera die Aufnahme startet. Die Auflösung beträgt 800x600 Pixel und es werden 15 Bilder pro Sekunde aufgenommen. Nachdem die Bilder aufgenommen wurden, wird auf jedem Bild die Position der Kugel auf dem Endeffektor mit den Funktionen *fcbBildPos_HSV* und *Cam2Glob* bestimmt. Der Name der Kalibrierungsdatei steht in dem Textfeld „Dateiname Regression“. Anschließend werden die globalen Koordinaten und der Zeitstempel der Aufnahme in die Tabelle eingetragen und in der Grafik dargestellt. Der Inhalt der Tabelle kann mit dem Knopf „Speichern“ in einer .mat Datei gespeichert werden. Der Name der Datei steht in dem Textfeld „Dateiname Bahn“.
Der Quellcode der GUI Wegmessung befindet sich im Anhang A.13.

7 Auswertung

Die Auswertung ist in zwei Abschnitte unterteilt. Zum einen wird in Kapitel 7.1 der Verlauf einer Kreisbahn gezeigt, die mit dem Steuerungsprogramm durchgeführt wurde. Da dies ein dynamischer Prozess ist, dient dieses Kapitel nur als Überprüfung der Routine und als grundsätzlicher Überblick. Um das statische Verhalten des Roboters zu überprüfen, wird abschließend in Kapitel 7.2 der Vergleich zwischen dem Roboter und dem Simulationsmodell gezogen. In diesem Kapitel wird außerdem die Korrekturmethode ausgewertet, welche sowohl am Roboter als auch im Simulationsmodell verwendet wird.

7.1 Kreisbahn des Roboters

In diesem Abschnitt wird die Funktion der Routine „Bewegung“ des Steuerungsprogramms dargestellt. Da es sich bei dieser Routine um eine Bewegung handelt, kann hier die entwickelte Korrekturmethode nicht angewandt werden. In Abbildung 7.1 sind die Ist- und Soll-Position des Endeffektors abgebildet. Die Bewegung startet im Mittelpunkt des Kreises bei $M = (400\,\text{mm}/500\,\text{mm})$. Von dort bewegt sich der Roboter 30 mm mit konstanter Geschwindigkeit in x-Richtung. Nachdem die Gerade abgefahren ist, bewegt sich der Roboter entgegen dem Uhrzeigersinn um den Mittelpunkt M.

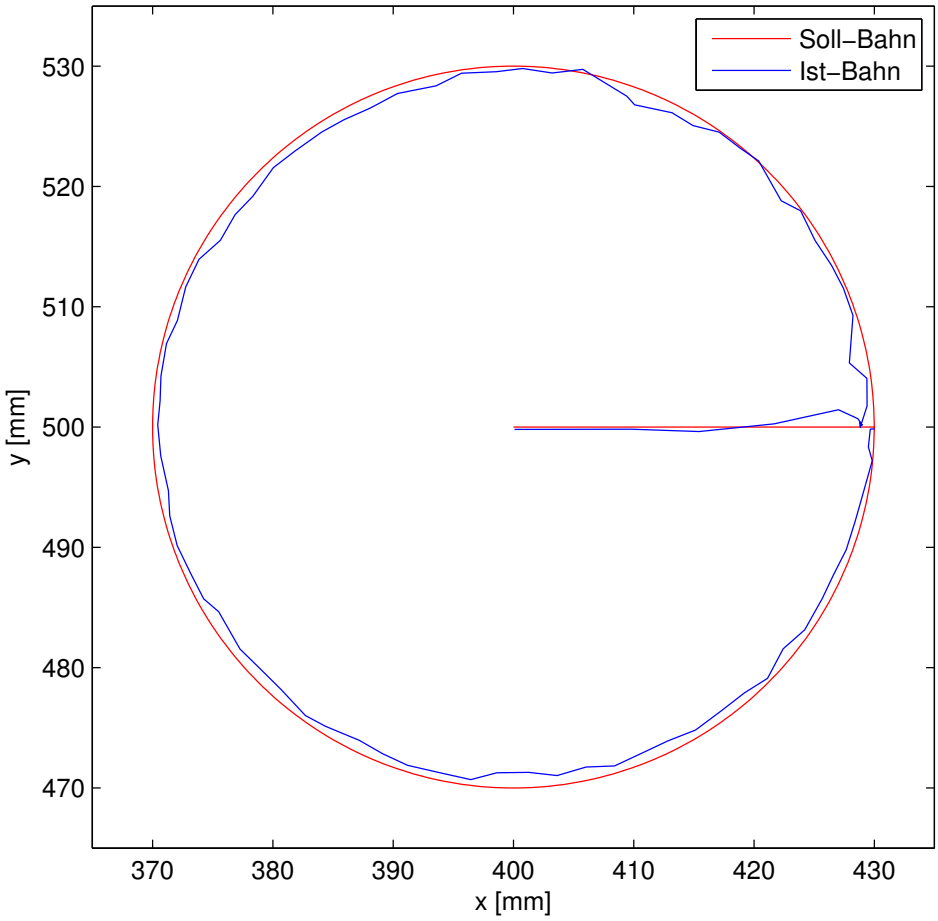


Abbildung 7.1: Kreisbahn

In der Abbildung 7.1 ist zu erkennen, dass die Kreisbahn des Endeffektors nur gering von der Soll-Bahn abweicht. Abschließend ist hierzu anzumerken, dass die Routine ihre Aufgabe erfüllt. Somit kann später mit dieser Routine z.B. das dynamische Verhalten des Roboters untersucht werden.

7.2 Vergleich zwischen Roboter und Simulationsmodell

Für den Vergleich des Simulationsmodells mit dem Roboter werden an sechs verschiedenen Punkten Messungen vorgenommen. An jedem Messpunkt werden je drei Einzelmessungen vorgenommen. Als erstes wird die Ausgangsposition gemessen. Danach wird der Endeffektor mit einer Kraft von 9,81 N in negativer x-Richtung belastet. Die Belastung am Roboter erfolgt mit einem Gewicht von 1 kg über eine Umlenkrolle, wie in Abbildung 7.2 zu sehen. Nach der Belastung mit dem Gewicht muss die Umlenkrolle in Kraftrichtung gedreht

werden, um das Reibungsmoment der Umlenkrolle zu überwinden. Damit wird erst die gewünschte Belastung erreicht. Anschließend wird die neue Position gemessen.

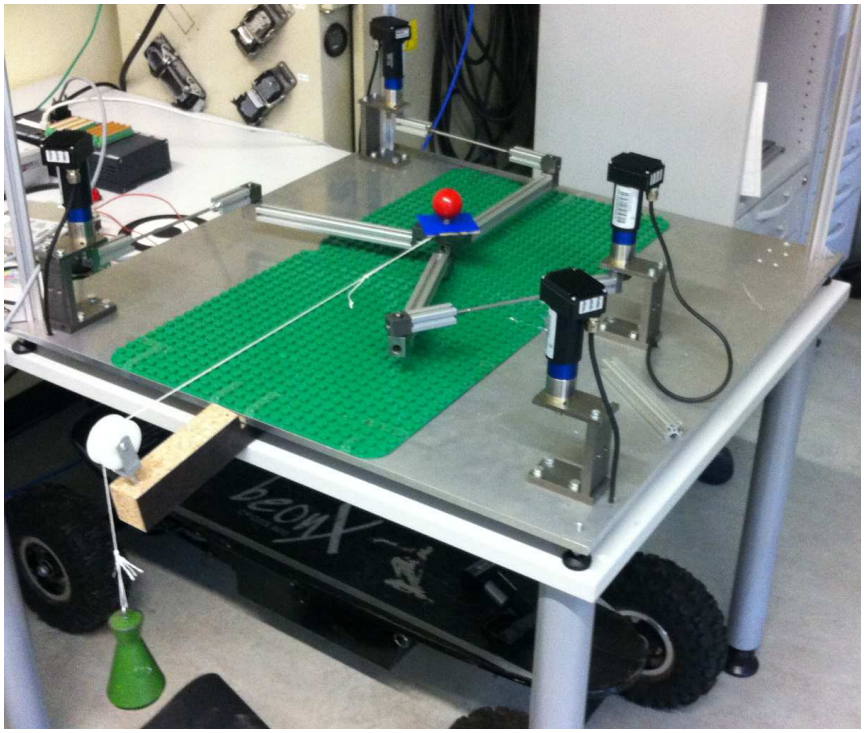


Abbildung 7.2: Aufbau der Messung

Danach wird die Korrekturberechnung eingeschaltet, um die Winkelkorrektur zu berechnen. Bevor die Korrektur durchgeführt wird, muss der Roboter entlastet werden, da das Gewicht für die Ausführung der Korrekturbewegung zu schwer ist. Nachdem die Korrektur durchgeführt wurde, wird der Roboter wieder mit dem Gewicht belastet und das Reibungsmoment der Umlenkrolle durch Drehung in Kraftrichtung überwunden. Abschließend wird die korrigierte Position gemessen.
Mit diesen Daten werden dann die Auslenkungen Δ_1 und Δ_2 berechnet. Die Auslenkung Δ_1 ist die Differenz zwischen der Ausgangspostion und der belasteten Position. Die Differenz zwischen der Ausgangsposition und der korrigierten Position ergibt die Auslenkung Δ_2. Außerdem wird der Wert $e = {}^{\Delta_2}/_{\Delta_1} \cdot 100\,\%$ berechnet, der den prozentualen Restfehler nach der Korrektur angibt.

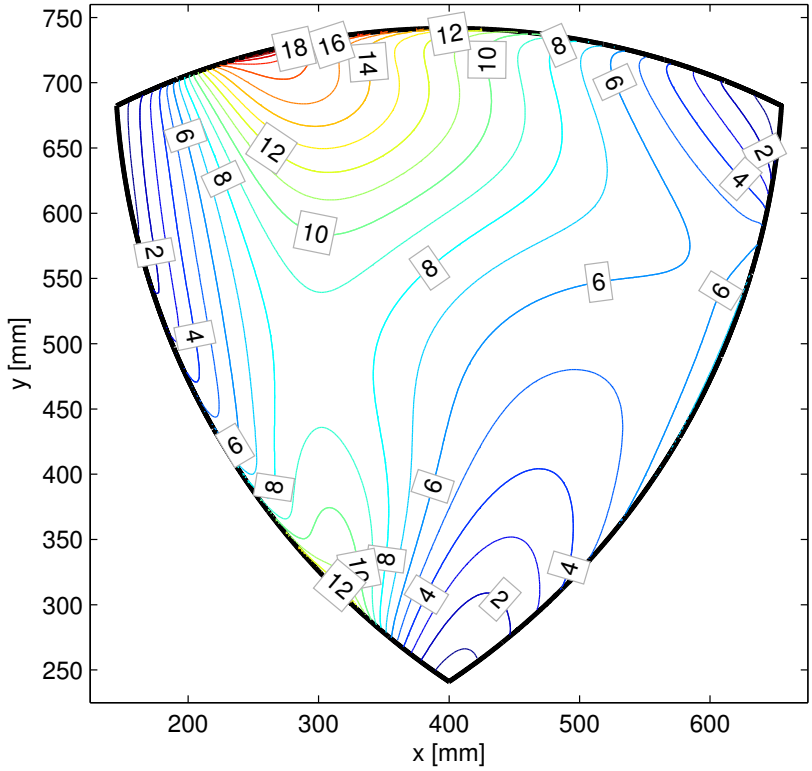


Abbildung 7.3: Kontur der Auslenkung Δ[mm] im Arbeitsraum eines 3-RRR Roboters mit Elastomerkupplungen ($k = 75\,\mathrm{Nm/rad}$) unter einer Kraft $F = 9{,}81\,\mathrm{N}$ in negativer x-Richtung

Für die Auswahl der Messpunkte wurde ein Kontur Diagramm (Abb. 7.3) der Auslenkung Δ des 3-RRR Roboters mit Elastomerkupplungen ($k = 75\,\mathrm{Nm/rad}$) erzeugt. Damit der Messfehler deutlich kleiner ist als die Auslenkung Δ_1 des Roboters, kann mit dem Diagramm eine Auswahl der Messpunkte getroffen werden. Die Messpunkte wurden daher im oberen Bereich des Arbeitsraumes gewählt, da dort die Auslenkungen relative hoch sind. Die Messpunkte und Auslenkungen am Roboter sind in der Tabelle 7.1 zusehen.

Tabelle 7.1: Messdaten am Roboter bei einer Belastung mit 1 kg in negativer x-Richtung

unbelastet		Auslenkungen		Restfehler
x[mm]	y[mm]	Δ_1[mm]	Δ_2[mm]	e[%]
300, 48	600, 10	11, 24	0, 26	2, 31
400, 29	600, 65	9, 85	0, 17	1, 73
500, 33	600, 65	7, 72	0, 16	2, 07
300, 20	650, 81	13, 86	0, 41	2, 89
399, 85	650, 95	11, 86	0, 20	1, 69
499, 41	650, 63	8, 46	0, 14	1, 65

Aus den Daten in der Tabelle 7.1 ist zu erkennen, dass durch die Korrekturmethode die Position des Endeffektors korrigiert wird. Der Restfehler e liegt dabei unter 3 %.

Für die Bestimmung der Ergebnisse in der Simulation wird durch das Subsystem „Berechnung der Auslenkung“ die Auslenkung Δ berechnet. Da die Ergebnisse der Simulation zeitabhängig sind, erreicht das System erst nach einer gewissen Zeit das statische Gleichgewicht. Eine solche Messung ist in Abbildung 7.4 zu sehen.

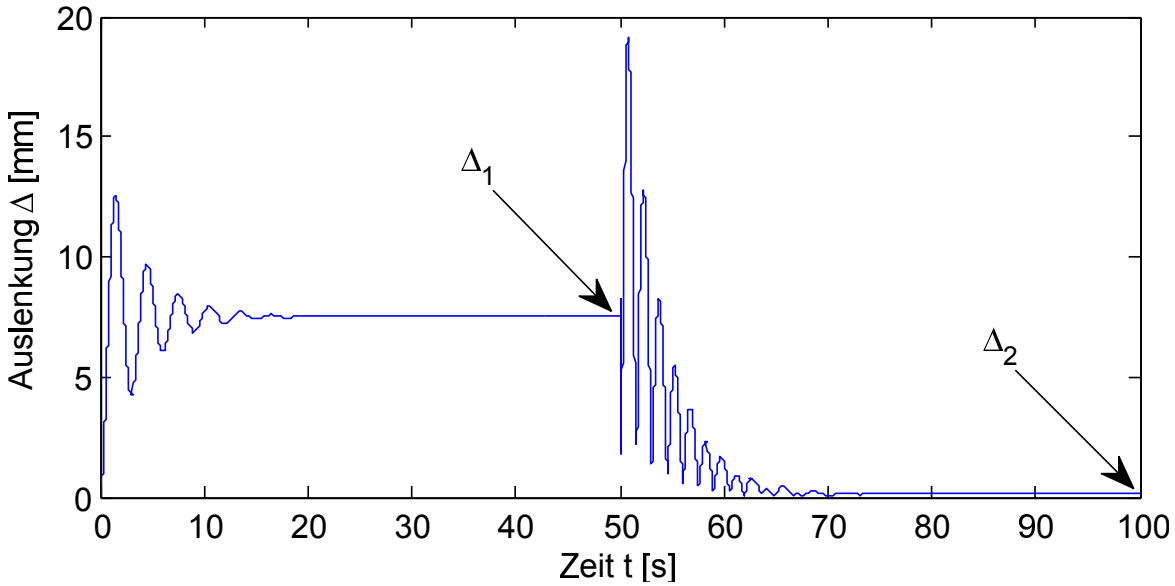


Abbildung 7.4: Simulationsergebnis der Auslenkung Δ

In der Abbildung ist zwischen den Zeitpunkten 0 s und 50 s das Ausschwingen des Systems nach der Belastung zu erkennen. Nach dem Ausschwingen kann der Wert Δ_1 bestimmt werden. Zum Zeitpunk $t = 50$ s wird die Korrektur vorgenommen, die daraus resultierende

Schwingung ist nach 100 s beendet und es kann der Wert Δ_2 abgelesen werden. Die so bestimmten Auslenkungen sind in Tabelle 7.2 zusammengefasst.

Tabelle 7.2: Ergebnisse aus der Simulation bei einer Belastung von $F = 9{,}81\,\mathrm{N}$ in negativer x-Richtung

Ausgangsposition		Abweichungen		Restfehler
x[mm]	y[mm]	Δ_1[mm]	Δ_2[mm]	e[%]
300	600	9,846	0,081	0,82
400	600	8,584	0,092	1,07
500	600	6,740	0,043	0,64
300	650	12,143	0,129	1,07
400	650	10,328	0,097	0,94
500	650	7,360	0,038	0,51

Wie in der Tabelle 7.2 zu sehen wird die Auslenkung des Endeffektors korrigiert. Der Restfehler e ist nach der Korrektur kleiner als 1,1 %.

Um die Daten aus der Simulation mit den Messungen vergleichen zu können, muss der Messfehler $S_P = 0{,}8363\,\mathrm{mm}$ aus Kapitel 6.2 berücksichtigt werden. Wird dieser berücksichtigt, liegen die Auslenkungen Δ_1 des Modells außerhalb der gemessenen Auslenkungen des Roboters. Daraus folgt, dass die Steifigkeit des Modells und die des Roboters nicht übereinstimmen. Da alle gemessenen Auslenkungen am Roboter größer sind als in der Simulation, ist die Steifigkeit des Roboters geringer als im SimMechanics Modell. Dies kann daran liegen, dass der Roboter weitere Elastizitäten besitzt, die nicht in dem Simulationsmodell modelliert wurden. Hierzu gehören z.B. mögliche Elastizitäten, die durch die Verschraubung der Elemente entstehen, sowie Elastizitäten in den Motoren und Planetengetrieben.
Das SimMechanics Modell wird durch die Korrekturmethode auf unter 1,1 % und der Roboter auf unter 3 % zurückgestellt. Betrachtet man nun die Auslenkungen Δ_2, so liegen die Auslenkungen des Modells innerhalb der Auslenkung des Roboters unter Berücksichtigung des Messfehlers. Daher wird der Roboter innerhalb des Fehlerbereiches genauso gut zurückgestellt wie das Modell. Um dies zu verifizieren ist eine genauere Messmethode erforderlich.

8 Fazit und Ausblick

Fazit

In dieser Masterthesis wurde für den in dem Masterprojekt [5] konstruierten 3-RRR Roboter erfolgreich eine Steuerung von Bachmann konfiguriert, implementiert und in Betrieb genommen. Die einzelnen Komponenten des 3-RRR Roboters wurden für den Betrieb mit der Bachmann Steuerung über ein CANOpen Netzwerk konfiguriert. Die dazugehörige Beschreibung erfolgte in Form einer Bedienungsanleitung.
Die ersten Armsegmente wurden neu konstruiert und mit elastischen Elementen versehen, um die Steifigkeit des Roboters zu verringern. Aus diesem Grund wurde das Simulink/SimMechanics Modell durch elastische Elemente erweitert, um es dem geänderten statischen Verhalten des Roboters anzupassen.
Für die Bachmann Steuerung wurde ein Steuerungsprogramm entwickelt, mit dem es möglich ist den Roboter auf eine gewünschte Position zu fahren, um dort Versuche vorzunehmen oder eine Referenzfahrt durchzuführen.
Mit der Positionsbestimmung über eine Kamera wurde eine kostengünstige Methode geschaffen, die Position des Endeffektors mit einem Messfehler von $S_P = 0{,}8363\,\text{mm}$ zu bestimmen. Diese Methode ist jedoch stark von der Beleuchtung abhängig und benötigt daher eine konstante und gleichmäßige Ausleuchtung des Roboters. Bei den Messungen der Auslenkungen nach der Korrektur stößt die Messmethode an ihre Grenzen, da diese kleiner sind als der Messfehler.
Die Korrekturmethode liefert die Korrekturwinkel aus den gemessenen Motorströmen zur Erreichung der Ausgangsposition nach einer Belastung. Mit diesen Korrekturwinkeln werden am Roboter die Auslenkungen durch die Belastungen auf unter 3 % korrigiert. Bei der Simulation werden die Korrekturwinkel anhand der Momente der Aktoren berechnet. Der Fehler nach der Korrektur liegt hierbei unter 1,1 %. Damit liegen die Messungen und Berechnungen nur 2 Prozentpunkte auseinander und zeigen damit die prinzipielle Anwendbarkeit der Korrekturmethode innerhalb des Messfehlers.

Ausblick

Für die Untersuchungen in dem Themenbereich redundanter Roboter wurde in dieser Masterthesis durch die programmierte Steuerung und entwickelte Messtechnik eine Grundlage erarbeitet auf der weiterführende Projekte in diesem Bereich aufbauen können.
Das neu konstruierte Armsegment weist nicht die in dem FE-Modell berechnete Steifigkeit auf. Dies liegt zum einen daran, dass der Elastizitätsmodul des Rundstabes geringer ist als angenommen. Zum anderen daran, dass das Armsegment aus mehreren verschraubten Elementen besteht. Aus diesem Grund sollten die Armsegmente aus einem Element neu konstruiert und gefertigt werden. Des Weiteren ist die Steifigkeit des Roboters geringer als im Modell. Da im Modell nicht alle Elastizitäten Berücksichtigt wurden, sollte das Modell dem Roboter weiter angenähert werden. Hierzu zählen z.B. die Elastizitäten aus den Motoren und Planetengetrieben.
Bei zukünftigen Untersuchungen sollte für die Positionsbestimmung des Roboters eine unempfindlichere Messmethode mit höherer Genauigkeit ausgewählt werden, um präzisere Ergebnisse und ein störungsfreies Arbeiten zu gewährleisten. Eine höhere Genauigkeit kann z.B. durch eine Kamera mit einer größeren Pixeldichte erreicht werden.
Die Kalibrierung der Postionen der Antriebe erfolgte bisher manuell, zukünftig sollte dies automatisiert werden, um Ungenauigkeiten bei der manuellen Konfiguration zu vermeiden.
Die Steifigkeit des Roboters, die aus den Elastizitäten der Armsegmente resultiert, kann durch eine Vorspannung, die über die Redundanz aufgebracht wird, erhöht werden. Dieser Effekt sollte eingehend in weiteren Projekten untersucht werden.
Die programmierte Referenzfahrt kann später dazu verwendet werden, das dynamische Verhalten des Roboter zu untersuchen.
Ebenfalls ist es denkbar den Roboter für Lehrzwecke einzusetzen. Hierbei können die Theorien zu redundanten Kinematiken im Rahmen eines Praktikums anschaulich erklärt werden.

9 Literaturverzeichnis

[1] Henrik Göhrs. *SCARA-Robotersteuerung mit MX 220 SPS und CANopen.* Hochschule für Angewandte Wissenschaften in Hamburg, 2011.

[2] Jeff Wendlandt und Dallas Kennedy Victor Chudnovsky, Arnav Mukherjee. *Modeling Flexible Bodies in SimMechanics.* MathWorks, 2006.

[3] *Bachmann: M1-Anwenderhandbuch.*

[4] *FAULHABER Motion Control Systeme Gerätehandbuch.*

[5] Sebastian Teuscher. *Konstruktion, Entwicklung und Simulation eines parallelen, planaren und redundanten 3-RRR und 4-RRR Roboters.* Hochschule für Angewandte Wissenschaften in Hamburg, 2012.

[6] Sascha Beuermann Holger Spiess und Stefan Löhnert Peter Wriggers, Udo Nackenhorst. *Technische Mechanik kompakt.* Teubner, 2005.

[7] Jürgen Dankert und Helga Dankert. *Technische Mechanik.* Vieweg+Teubner Verlag, 2011.

[8] *FAULHABER Kommunikations-/Funktionshandbuch.*

[9] *Bachmann: M1-Referenzhandbuch.*

[10] *Bachmann: Technical Description M-Target for Simulink.*

Anhang

A.1 Datenblatt des Motors (Auszug)

FAULHABER

Bürstenlose DC-Servomotoren 53 mNm

mit integriertem Motion Controller
und RS232 oder CAN-Schnittstelle

Kombinierbar mit
Getriebe:
30/1, 32A, 32ALN, 32/3 (S), 38/1 (S), 38/2 (S)

3564 ... B Cx

		3564 K		024 B CS/CC/CO	
1	Nennspannung	U_N		24	Volt
2	Anschlusswiderstand, Phase-Phase	R		1,16	Ω
3	Abgabeleistung [1]	$P_{2\,max.}$		51	W
4	Wirkungsgrad	$\eta_{max.}$		82	%
5	Leerlaufdrehzahl	n_0		10 500	rpm
6	Leerlaufstrom [3]	I_0		0,225	A
7	Anhaltemoment bei 8A	M_H		160	mNm
8	Reibungsdrehmoment, statisch	C_0		1,10	mNm
9	Reibungsdrehmoment, dynamisch	C_V		$2{,}4 \cdot 10^{-4}$	mNm/rpm
10	Drehzahlkonstante	k_n		473	rpm/V
11	Generator-Spannungskonstante	k_E		2,114	mV/rpm
12	Drehmomentkonstante	k_M		20,2	mNm/A
13	Stromkonstante	k_I		0,05	A/mNm
14	Steigung der n-M-Kennlinie	$\Delta n/\Delta M$		26,2	rpm/mNm
15	Anschlussinduktivität, Phase-Phase	L		194	µH
16	Mechanische Anlaufzeitkonstante	τ_m		9,3	ms
17	Rotorträgheitsmoment	J		34	gcm²
18	Winkelbeschleunigung	$\alpha_{max.}$		47	$\cdot 10^3$rad/s²
19	Wärmewiderstände	$R_{th\,1}$ / $R_{th\,2}$	2,5 / 6,3		K/W
20	Thermische Zeitkonstante	τ_{w1} / τ_{w2}	23 / 1 175		s
21	Betriebstemperaturbereich		– 30 ... +85		°C
22	Wellenlagerung		Kugellager, vorgespannt		
23	Wellenbelastung, max. zulässig:				
	– radial bei 3 000 rpm (4,5 mm vom Befestigungsflansch)		108		N
	– axial bei 3 000 rpm		50		N
	– axial im Stillstand		131		N
24	Wellenspiel:				
	– radial	≤	0,015		mm
	– axial	=	0		mm
25	Gehäusematerial		Motor: Aluminium, schwarz eloxiert; Anbau: Zink		
26	Gewicht		510		g
27	Drehrichtung		ansteuerungsbedingt		
Empfohlene Werte - diese gelten unabhängig voneinander					
28	Drehzahl von-bis	$n_{e\,max.}$		5 - 12 000	rpm
29	Dauerdrehmoment bis [1] [2]	$M_{e\,max.}$		39 / 53	mNm
30	Thermisch zulässiger Dauerstrom [1] [2] [3]	$I_{e\,max.}$		2,1 / 2,8	A

[1] bei 8 400 rpm [2] Wärmewiderstand $R_{th\,2}$ nicht reduziert / Wärmewiderstand $R_{th\,2}$ um 55% reduziert
[3] zuzüglich 0,055 A Elektronikstrom bei U_B = 24V

A.2 Technische Daten zu Profil 5 20x20, natur

Eigenschaften			
Baureihe		=	Baureihe 5
Material		=	Al, eloxiert
Eigenschaft		=	natur
Liefereinheit		=	Zuschnitt max. 6000 mm
Höhe	h	=	20 mm
Breite	b	=	20 mm
Fläche	A	=	1.8 cm^2
Flächenträgheitsmoment um x-Achse	I_x	=	0.72 cm^4
Flächenträgheitsmoment um y-Achse	I_y	=	0.72 cm^4
Flächenträgheitsmoment Torsion	I_t	=	0.07 cm^4
Widerstandsmoment um x-Achse	W_x	=	0.72 cm^3
Widerstandsmoment um y-Achse	W_y	=	0.72 cm^3
Gewicht spez. Länge [kg/m]	m	=	0.48 kg/m

A.3 Steuerungsprogramm

```
function [Command,Argument,trace_konf,Pos_Messung,q_Fehler,...
          Phase_Pos_out,Phase_Bew_out,Korrektur_best_out,...
          Motor_Status_out,Strom_out,Homing_best_out,t_out]...
          = fcn(Parameter,Command_best,Argument_best,Error_Code,...
          Motor_AnAus,Positionierung,Bewegung,Korrektur,...
          Korrektur_vornehmen,Homing,Ist_Position,Phase_Pos,...
          Phase_Bew,Korrektur_best,Motor_Status,Strom,Homing_best,t)
%%%%%%%%%%%%%%%%
% Variablen INIT
%%%%%%%%%%%%%%%%
Command = zeros(3,1);
Argument = zeros(3,1);
q_Fehler = zeros(3,1);
Strom_Mittelwert = zeros(3,1);
Pos_Messung = zeros(3,1);
Gesch = 100;
R = 30;
dt = 0.001;
X_Messung = Parameter(1);
Y_Messung = Parameter(2);
kk = Parameter(3);

% Dreiarmroboter Geometrie ohne Mittelplatte
% Bohrungsabstände Grundplatte
gp = 800;                              % [mm]
a = 640;                               % [mm]
a_aussen = (gp-a)/2;                   % [mm]
a_ver = a*sind(60);                    % [mm]

% Armlänge
L_arm = 288 ;                          % [mm]

% Befestigungsorte der Aktoren
A1 = [a_aussen gp-a_aussen]';
A2 = [gp-a_aussen gp-a_aussen]';
A3 = [gp/2 gp+-a_aussen-a_ver]';

% Berechnung der Motorwinkel
Pos_Messung = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,X_Messung,Y_Messung);
% Umwandlung von Grad in Inkremente
Pos_Messung = round(Pos_Messung*3000*66/360);

%%%%%%%%%%%%%%%%%%%%%
% Trace Konfiguration
%%%%%%%%%%%%%%%%%%%%%
trace_konf = zeros(5,1);
trace_konf(1) = 200;
```

```
trace_konf(2) = 201;
trace_konf(3) = 0;
trace_konf(4) = 1;
trace_konf(5) = 0;

%%%%%%%%%%%%%%%%%%%%%
% Homing
%%%%%%%%%%%%%%%%%%%%%
for i = 1 : 3
    if Homing(i) == 1
        if Command_best(i) == 184 && Error_Code(i) == 1
            Homing_best(i) = 1;
        end
        if Homing_best(i) == 0;
           Command(i) = 184;
           if i == 1
                Argument(i) = 0;
           end
           if i == 2
                Argument(2) = -49500;
           end
           if i == 3
                Argument(3) = 49500;
           end
        end
    end
    if Homing(i) == 0
        Homing_best(i) = 0;
    end
end

%%%%%%%%%%%%%%%%%%%%%
% Bewegung
%%%%%%%%%%%%%%%%%%%%%
% Bewegung starten
if Bewegung == 1 && Positionierung == 1 && Phase_Pos(4) == 2 &&...
    Motor_Status(1) == 1 && Motor_Status(2) == 1 && Motor_Status(3) == 1
    for i = 1 : 3
        if Phase_Bew(i) == 0
           Phase_Bew(i) = 1;
        end
    end
    if Phase_Bew(1) == 1 && Phase_Bew(2) == 1 && Phase_Bew(3) == 1
        Phase_Bew(4) = 1;
    end
    % Gerade in x-Richtung vom Kreismittelpunkt auf den Kreis
    if Phase_Bew(4) == 1
        EE_start = [X_Messung Y_Messung];
        EE_ziel = [X_Messung+R Y_Messung];
        Richtung = [EE_ziel(1)-EE_start(1) EE_ziel(2)-EE_start(2)];
        Gesch_XY = Richtung/norm(Richtung)*Gesch*dt;
```

```
        Fahrzeit = norm(Richtung)/Gesch;
        steps = Fahrzeit/dt;
        if t < steps
            t=t+1;
        end
        XEE = EE_start(1) + Gesch_XY(1)*t;
        YEE = EE_start(2) + Gesch_XY(2)*t;

        if t ≥ steps;
            XEE = EE_ziel(1);
            YEE = EE_ziel(2);
        end
        Command(1) = 180;
        Command(2) = 180;
        Command(3) = 180;
        q = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,XEE,YEE);
        % Umwandlung von Grad in Inkremente
        Argument = round(q*3000*66/360);

        for i = 1 : 3
            if Command_best(i) == 60 && Error_Code(i) == 1
                Phase_Bew(i) = 2;
            end
        end
        if Phase_Bew(1) == 2 && Phase_Bew(2) == 2 && Phase_Bew(3) == 2
            Phase_Bew(4) = 2;
            Command(1) = 0;
            Command(2) = 0;
            Command(3) = 0;
            t = 0;
        end
    end
    % Kreisbahn entgegen dem Uhrzeugersinn
    if Phase_Bew(4) == 2
        Phi_Start = 0;
        Phi_Ziel = 360;
        T = 5;
        Omega = 360/T;
        %%%%%%%%%%
        if Phi_Ziel-Phi_Start<0
        Omega=-Omega;
        end
        Fahrzeit = (Phi_Ziel-Phi_Start)/Omega;
        steps = Fahrzeit/dt;
        if t ≤ steps
            t=t+1;
        end
        Phi = Phi_Start+Omega*t*dt;
        if t ≥ steps;
            Phi=Phi_Ziel;
        end
```

```
        Command(1) = 180;
        Command(2) = 180;
        Command(3) = 180;
        XEE = X_Messung + cosd(Phi)*R;
        YEE = Y_Messung + sind(Phi)*R;
        q = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,XEE,YEE);
        % Umwandlung von Grad in Inkremente
        Argument = round(q*3000*66/360);

        for i = 1 : 3
            if Command_best(i) == 60 && Error_Code(i) == 1
                Phase_Bew(i) = 3;
            end
        end
        if Phase_Bew(1) == 3 && Phase_Bew(2) == 3 && Phase_Bew(3) == 3
            Phase_Bew(4) = 3;
            Command(1) = 0;
            Command(2) = 0;
            Command(3) = 0;
            t = 0;
        end
    end
end

% Bewegung Beenden
if Bewegung == 0 && Phase_Bew(4) ≠ 0
    for i = 1 : 3
        if Phase_Bew(i) ≠ 0
            Command(i) = 147;
            Argument(i) = 0;
        end
        if Command_best(i) == 147 && Argument_best(i) == 0 &&...
                Error_Code(i) == 1
            Phase_Bew(i) = 0;
        end
    end
    if Phase_Bew(1) == 0 && Phase_Bew(2) == 0 && Phase_Bew(3) == 0;
        Phase_Bew(4) = 0;
        t=0;
    end
end

%%%%%%%%%%%%%%%%%%%%%%
% Positionierung
%%%%%%%%%%%%%%%%%%%%%%
% Positionierung Starten %%%%%%
if Positionierung == 1 && Motor_Status(1) == 1 &&...
        Motor_Status(2) == 1 && Motor_Status(3) == 1
    for i = 1 : 2
        if Phase_Pos(i) == 0
           Phase_Pos(i) = 1;
```

```
    end
end
% Ausschalten von Motor 4
if Phase_Pos(3) == 0
    if Command_best(3) == 8 && Error_Code(3) == 1
        Phase_Pos(3) = 1;
    else
        Command(3) = 8;
        Argument(3) = 0;
    end
end
if Phase_Pos(1) == 1 && Phase_Pos(2) == 1 && Phase_Pos(3) == 1
    Phase_Pos(4) = 1;
end
% Bewegungung des Robotors mit Motor 1 und 2
if Phase_Pos(4) == 1
    for i = 1 : 2
        if Phase_Pos(i) == 1
            Command(i) = 147;
            if Ist_Position(i) > Pos_Messung(i)
                Argument(i) = -Gesch;
            else
                Argument(i) = Gesch;
            end
            if Ist_Position(i) < Pos_Messung(i)+100 &&...
                    Ist_Position(i) > Pos_Messung(i)-100
                Command(i) = 180;
                Argument(i) = Pos_Messung(i);
                if Command_best(i) == 60 && Error_Code(i) == 1
                    Phase_Pos(i) = 2;
                end
            end
        end
    end
end
% Anschalten von Motor 3
if Phase_Pos(1) == 2 && Phase_Pos(2) == 2
    if Command_best(3) == 15 && Error_Code(3) == 1
        Phase_Pos(3) = 1.5;
    else
        Command(3) = 15;
        Argument(3) = 0;
    end
end
% Positionieren von Motor 3
if Phase_Pos(4) == 1 && Phase_Pos(3) == 1.5
    Command(3) = 180;
    Argument(3) = Pos_Messung(3);
    if Command_best(3) == 60 && Error_Code(3) == 1
        Phase_Pos(3) = 2;
        Phase_Pos(4) = 2;
```

```
        end
    end
end

% Positionierung Beenden
if Positionierung == 0 && Phase_Pos(4) ≠ 0
    for i = 1 : 3
        if Phase_Pos(i) ≠ 2
            Command(i) = 147;
            Argument(i) = 0;
            if Command_best(i) == 147 && Argument_best(i) == 0 &&...
                    Error_Code(i) == 1
                Phase_Pos(i) = 0;
            end
        else
            Phase_Pos(i) = 0;
        end
    end
    if Phase_Pos(1) == 0 && Phase_Pos(2) == 0 && Phase_Pos(3) == 0;
        Phase_Pos(4) = 0;
        t=0;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Korrektur
%%%%%%%%%%%%%%%%%%%%%%%%%
if Korrektur == 1 && Phase_Pos(4) == 2
    for i = 1 : 3
        Command(i) = 52;
        Argument(i) = 0;
        if Command_best(i) == 52 && Error_Code(i) == 1
            Strom(i,:) = [Argument_best(i) Strom(i,1:end-1)];
        end
    end
    % kk [rad/Nmm]
    Strom_Mittelwert = mean(Strom,2);           % mA
    k_M = 20.2;                                 % mNm/A
    M_Motor = k_M * Strom_Mittelwert / 1000;    % Nmm
    M_Arm = M_Motor * 66;                       % Nmm
    q_kor = M_Arm*kk *66*3000/(2*pi);           % Ink
    q_Fehler = round(q_kor);
    ZD = ZugDruck([-1 0 0],X_Messung,Y_Messung,A1,A2,A3,L_arm);
    q_Fehler = q_Fehler .* ZD;

    for i = 1 : 3
        if Korrektur_vornehmen == 1 && Korrektur_best(i) == 0
                Command(i) = 180;
                Argument(i) = Pos_Messung(i) + q_Fehler(i);
                if Command_best(i) == 60 && Error_Code(i) == 1
                    Korrektur_best(i) = 1;
```

```
                end
        end
    end
end
if Korrektur_vornehmen == 0
    Korrektur_best = zeros(3,1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Motor An und Ausschalte
%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1 : 3
    if Command_best(i) == 15 && Error_Code(i) == 1
        Motor_Status(i) = 1;
    end
    if Command_best(i) == 8 && Error_Code(i) == 1
        if i == 3
            if Phase_Pos(3) ≠ 1
                Motor_Status(i) = 0;
            end
        else
            Motor_Status(i) = 0;
        end
    end
    if Motor_AnAus(i) == 1 && Motor_Status(i) == 0
        Command(i) = 15;
        Argument(i) = 0;
    end
    if Motor_AnAus(i) == 0
        Command(i) = 8;
        Argument(i) = 0;
    end
    if Command_best(i) == 180 && Error_Code(i) == 1
        Command(i) = 60;
        Argument(i) = 0;
    end
end

Motor_Status_out = Motor_Status;
Homing_best_out = Homing_best;
t_out = t;
Strom_out = Strom;
Phase_Pos_out = Phase_Pos;
Phase_Bew_out = Phase_Bew;
Korrektur_best_out = Korrektur_best;
```

A.4 Matlab m-file: FEModell

```
function [kk] = FEModell()
% Parameter:
L  = 288 ;          % Länge des Armsegmentes [mm]
c  = 55 ;           % Länge des Aluminium Profils [mm]
r = 2.5;            % Radius des Rundstabes [mm]
n = 288;            % Anzahl der Elemente
l = L/n;            % Länge der Elemente [mm]
dof = 2*(n+1) ;     % Anzahl Freiheitsgrade global

Abschnitte_mm = [12 12+c L-c-12 L-12 L];
Abschnitte_element = Abschnitte_mm/l;

% Elastizitäten
E(1) = 210e3*1e3;   % 210e3 MPa für die Korrektur am Roboter
E(2) = 70e3*1e3;    %  70e3 MPa für die Korrektur am Roboter
E(3) = 210e3/1.96;  % 1.96 : Angleichung an die gefertigten Armsegmente
E(4) = E(2);
E(5) = E(1);

% Flächenträgheitsmomente
I(1) = 20*20^3/12;  % [mm^4]
I(2) = 0.72*10^4;   % [mm^4]
I(3) = pi/4*r^4;    % [mm^4]
I(4) = I(2);
I(5) = I(1);

% ========================================

% Initialisieren
K = zeros(dof,dof);   % Gesamtsteifigkeitsmatrix
u = zeros(dof,1);     % Vektor der Knotenverformungen

% Randbedingungen
ufree =  [(2 : dof-2) dof] ;   % Welche Freiheitsgrade frei sind

% ========================================

% Aufstellen der Gesamtsteifigkeitsmatrix K
for i = 1 : n
    if i ≤ Abschnitte_element(5)
        EI(i) = E(5)*I(5);
    end
    if i ≤ Abschnitte_element(4)
        EI(i) = E(4)*I(4);
    end
    if i ≤ Abschnitte_element(3)
        EI(i) = E(3)*I(3);
    end
    if i ≤ Abschnitte_element(2)
        EI(i) = E(2)*I(2);
    end
```

```
    if i ≤ Abschnitte_element(1)
        EI(i) = E(1)*I(1);
    end
    inz = (i*2-1:i*2-1+3);
    K(inz,inz) = K(inz,inz) + esm(EI(i),l);
end

% Bildung der inversen des reduzierten Systems
Kred = K(ufree,ufree);
Kred_inv = inv(Kred);

kk = Kred_inv(1,1);
```

A.5 Matlab m-file: Sim_Dreiarm_oM_kor_INIT

```
%% Masterthesis %%
% Autor: Hendrik Thönnißen
% Eingangswerte für das Simulink Modell
clc
clear all

%% Eingangswerte
dt = 0.01 ;                         % [s]
tend = 100 ;                        % [s]
tkor = 50 ;                         % [s]
EE = [300 650];                     % x[mm] y[mm]

%% Dreiarmroboter oM

% Bohrungsabstände Grundplatte
gp = 800;                           % [mm] Abmaße der Grundplatte
a = 640;                            % [mm] Abstand der Aktoren 1u2
a_aussen = (gp-a)/2;                % [mm] Abstand der Aktoren 1u2 zur Kante
a_ver = a*sind(60);                 % [mm] y Abstand Aktor 3 zu 1u2

% Befestigungsorte der Aktoren
A1 = [a_aussen gp-a_aussen]';       % x[mm] y[mm]
A2 = [gp-a_aussen gp-a_aussen]';    % x[mm] y[mm]
A3 = [gp/2 gp-a_aussen-a_ver]';     % x[mm] y[mm]

% Parameter für die Arme
L_arm = 288 ;                       % [mm]
c  = 55 ;                           % [mm]
steif = c + 12;                     % [mm]

% Berechnung der Gelenkwinkel
[q theta] = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,EE(1),EE(2));
```

```
%% FEM-Modell des elastischen Arms
kk = FEModell();
```

A.6 Matlab Funktion: FEModell_Kragarm

```
% Berechnung der Maße des elastischen Arms
clear all
clc

% Parameter:
L  = 288 ;          % Länge des Armsegmentes [mm]
c  = 55 ;           % Länge des Aluminium Profils [mm]
r = 2.5;            % Radius des Rundstabes [mm]
n = 288;            % Anzahl der Elemente
l = L/n;            % Länge der Elemente [mm]
dof = 2*(n+1);      % Anzahl Freiheitsgrade global
F = 1;              % Kraft an der Spitze [N]

Abschnitte_mm = [12 12+c L-c-12 L-12 L];
Abschnitte_element = Abschnitte_mm/l;

% Elastizitatsmodule
E(1) = 210e3;       %[MPa]
E(2) = 70e3;        %[MPa]
E(3) = 210e3;       %[MPa]
E(4) = E(2);
E(5) = E(1);

% Flächenträgheitsmomente
I(1) = 20*20^3/12;  %[mm^4]
I(2) = 0.72*10^4;   %[mm^4]
I(3) = pi/4*r^4;    %[mm^4]
I(4) = I(2);
I(5) = I(1);

% ========================================

% Initialisieren
K = zeros(dof,dof);   % Gesamtsteifigkeitsmatrix
u = zeros(dof,1);     % Vektor der Knotenverformungen
f = zeros(dof,1);     % Vektor der Knotenkraefte

% Randbedingungen
ufree =  (3 : dof) ;   % Welche Freiheitsgrade frei sind
```

```
% Vektor der Knotenkraefte (f-Vektor)
f(dof-1) = F;


% ========================================

% Aufstellen der Gesamtsteifigkeitsmatrix K
for i= 1  : n
    if i <= Abschnitte_element(5)
        EI(i) = E(5)*I(5);
    end
    if i <= Abschnitte_element(4)
        EI(i) = E(4)*I(4);
    end
    if i <= Abschnitte_element(3)
        EI(i) = E(3)*I(3);
    end
    if i <= Abschnitte_element(2)
        EI(i) = E(2)*I(2);
    end
    if i <= Abschnitte_element(1)
        EI(i) = E(1)*I(1);
    end
    inz = (i*2-1:i*2-1+3);
    K(inz,inz) = K(inz,inz) + esm(EI(i),l);
end


% Loesung des reduzierten Gleichungssystems  Ku = f
Kred = K(ufree,ufree);
fred = f(ufree);
ured = inv(Kred)*fred;

% Berechnung der unbekannten Knotenkraefte aus  f = Ku
u(ufree) = ured; % Vektor u wieder auf volle Groesse setzen
% Ausgabe der Durchbiegung
w = u(end-1)
```

A.7 Matlab Funktion: inv_Kin_3RRR_oM

```
%% Funktion zur Berechnung der inversen Kinematik des 3-RRR o.M.

function [q theta] = inv_Kin_3RRR_oM(A1,A2,A3,L,XEE,YEE)
q = zeros(3,1);
theta = zeros(3,1);
C1 = [XEE YEE]' ;
C2 = [XEE YEE]' ;
```

```
C3 = [XEE YEE]' ;


% Berechnung der Zwischenwinkel teta der einzelnen Arme
theta_1 = -180/pi*acos( ( (C1(1)-A1(1))^2 + (C1(2)-A1(2))^2 - 2*L^2 ) / ...
    (2*L^2 ) ) ;
theta_2 = -180/pi*acos( ( (C2(1)-A2(1))^2 + (C2(2)-A2(2))^2 - 2*L^2 ) / ...
    (2*L^2 ) ) ;
theta_3 = -180/pi*acos( ( (C3(1)-A3(1))^2 + (C3(2)-A3(2))^2 - 2*L^2 ) / ...
    (2*L^2 ) ) ;
theta = [theta_1 theta_2 theta_3];


% Berechnung der inversen Kinematik / Antriebswinkel q

% Aufteilung der inversen Kinematik in Zähler und Nenner
zaehler1 = ( (1+cos(theta_1*pi/180))*(C1(2)-A1(2)) - ...
    (sin(theta_1*pi/180))*(C1(1)-A1(1)) );
nenner1  = ( (1+cos(theta_1*pi/180))*(C1(1)-A1(1)) + ...
    (sin(theta_1*pi/180))*(C1(2)-A1(2)) );
% Fallunterscheidung, in welchem Quadranten der Winkel ist, da der Tangens
% nur von -90 bis +90 definiert ist
if     zaehler1 > 0 && nenner1 > 0 % erster Quadrant
q(1) = 0   + 180/pi*atan ( zaehler1 / nenner1 ) ;
elseif zaehler1 > 0 && nenner1 < 0 % zweiter Quadrant
q(1) = 180 + 180/pi*atan ( zaehler1 / nenner1 ) ;
elseif zaehler1 < 0 && nenner1 < 0 % dritter Quadrant
q(1) = 180 + 180/pi*atan ( zaehler1 / nenner1 ) ;
elseif zaehler1 < 0 && nenner1 > 0 % vierter Quadrant
q(1) = 360 + 180/pi*atan ( zaehler1 / nenner1 ) ;
elseif zaehler1 == 0 && nenner1 < 0
q(1) = 180   + 180/pi*atan ( zaehler1 / nenner1 ) ;
else
q(1) = 0   + 180/pi*atan ( zaehler1 / nenner1 ) ;
end

zaehler2 = ( (1+cos(theta_2*pi/180))*(C2(2)-A2(2)) - ...
    (sin(theta_2*pi/180))*(C2(1)-A2(1)) );
nenner2  = ( (1+cos(theta_2*pi/180))*(C2(1)-A2(1)) + ...
    (sin(theta_2*pi/180))*(C2(2)-A2(2)) );
if     zaehler2 > 0 && nenner2 > 0 % erster Quadrant
q(2) = 0   + 180/pi*atan ( zaehler2 / nenner2 ) ;
elseif zaehler2 > 0 && nenner2 < 0 % zweiter Quadrant
q(2) = 180 + 180/pi*atan ( zaehler2 / nenner2 ) ;
elseif zaehler2 < 0 && nenner2 < 0 % dritter Quadrant
q(2) = 180 + 180/pi*atan ( zaehler2 / nenner2 ) ;
elseif zaehler2 < 0 && nenner2 > 0 % vierter Quadrant
q(2) = 360 + 180/pi*atan ( zaehler2 / nenner2 ) ;
elseif zaehler2 == 0 && nenner2 < 0
q(2) = 180   + 180/pi*atan ( zaehler2 / nenner2 ) ;
else
```

```
q(2) = 0   + 180/pi*atan ( zaehler2 / nenner2 ) ;
end

zaehler3 = ( (1+cos(theta_3*pi/180))*(C3(2)-A3(2)) - ...
    (sin(theta_3*pi/180))*(C3(1)-A3(1)) );
nenner3  = ( (1+cos(theta_3*pi/180))*(C3(1)-A3(1)) + ...
    (sin(theta_3*pi/180))*(C3(2)-A3(2)) );
if     zaehler3 > 0 && nenner3 > 0 % erster Quadrant
q(3) = 0   + 180/pi*atan ( zaehler3 / nenner3 ) ;
elseif zaehler3 > 0 && nenner3 < 0 % zweiter Quadrant
q(3) = 180 + 180/pi*atan ( zaehler3 / nenner3 ) ;
elseif zaehler3 < 0 && nenner3 < 0 % dritter Quadrant
q(3) = 180 + 180/pi*atan ( zaehler3 / nenner3 ) ;
elseif zaehler3 < 0 && nenner3 > 0 % vierter Quadrant
q(3) = 360 + 180/pi*atan ( zaehler3 / nenner3 ) ;
elseif zaehler3 == 0 && nenner3 < 0
q(3) = 180 + 180/pi*atan ( zaehler3 / nenner3 ) ;
else
q(3) = 0   + 180/pi*atan ( zaehler3 / nenner3 ) ;
end

q(find(q(:)>180))=q(find(q(:)>180))-360;
end
```

A.8 Matlab Funktion: fncBildPos_HSV

```
function [Mitte,z,d] = fncBildPos_HSV(data)

% Werte Null setzen
z=0;
d=0;
Mitte = zeros(1,2);
[Hoehe Breite temp] = size(data);
hsvImage = rgb2hsv(data);          % Konvertieren in HSV Raum
hPlane = 360.*hsvImage(:,:,1);     % Skalieren von 0 bis 360
sPlane = hsvImage(:,:,2);          % Sättigung
nonRedIndex = (hPlane > 20) & ...  % Auswahl der nicht roten Pixel
              (hPlane < 350);
sPlane(nonRedIndex) = 0;           % Sättigung für nicht rote Pixel auf 0
pixdel = round(Hoehe*Breite/300);
bw = bwareaopen(sPlane,pixdel);    % alle Objekte kleiner pixdel entfernen

% Mittelpunkt bestimmen
[y x] = find(bw);
A = length(x);
if A ~= 0
    xx = mean(x);
```

```
    yy = mean(y);
    z = 1;
    Mitte = [xx yy];
    d = sqrt(4*A/pi);
end
```

A.9 Matlab Funktion: Cam2Glob

```
function [X Y] = Cam2Glob(x,y,Dateiname)
Dateiname = [Dateiname '.mat'];
load(Dateiname)
c = ones(length(x),1);
X = [c, x, y, x.*y]*a;
Y = [c, x, y, x.*y]*b;
```

A.10 Matlab Funktion: ZugDruck

```
function [ZugDruck] = ZugDruck(F,XEE,YEE,A1,A2,A3,L_arm)
ZugDruck = zeros(3,1)
EE = [XEE YEE 0];
[q] = inv_Kin_3RRR_oM(A1,A2,A3,L_arm,XEE,YEE);

B1 = zeros(1,3);
B2 = zeros(1,3);
B3 = zeros(1,3);

B1(1) = A1(1)+L_arm*cosd(q(1));
B1(2) = A1(2)+L_arm*sind(q(1));
B1(3) = 0;

B2(1) = A2(1)+L_arm*cosd(q(2));
B2(2) = A2(2)+L_arm*sind(q(2));
B2(3) = 0;

B3(1) = A3(1)+L_arm*cosd(q(3));
B3(2) = A3(2)+L_arm*sind(q(3));
B3(3) = 0;

alpha=[acosd((B1-EE)*F'/(norm(B1-EE)+norm(F)))
       acosd((B2-EE)*F'/(norm(B2-EE)+norm(F)))
       acosd((B3-EE)*F'/(norm(B3-EE)+norm(F)))];

ZugDruck(find(alpha>=90)) = 1;
```

```
ZugDruck(find(alpha<90)) = -1;
```

A.11 Matlab GUI: Kalibrierung

```
function varargout = Kalibrierung_gui(varargin)
% KALIBRIERUNG_GUI MATLAB code for Kalibrierung_gui.fig
%      KALIBRIERUNG_GUI, by itself, creates a new KALIBRIERUNG_GUI or ...
   raises the existing
%      singleton*.
%
%      H = KALIBRIERUNG_GUI returns the handle to a new KALIBRIERUNG_GUI ...
   or the handle to
%      the existing singleton*.
%
%      KALIBRIERUNG_GUI('CALLBACK',hObject,eventData,handles,...) calls ...
   the local
%      function named CALLBACK in KALIBRIERUNG_GUI.M with the given ...
   input arguments.
%
%      KALIBRIERUNG_GUI('Property','Value',...) creates a new ...
   KALIBRIERUNG_GUI or raises the
%      existing singleton*.  Starting from the left, property value ...
   pairs are
%      applied to the GUI before Kalibrierung_gui_OpeningFcn gets ...
   called.  An
%      unrecognized property name or invalid value makes property ...
   application
%      stop.  All inputs are passed to Kalibrierung_gui_OpeningFcn via ...
   varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Kalibrierung_gui

% Last Modified by GUIDE v2.5 20-Nov-2013 16:25:29

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Kalibrierung_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @Kalibrierung_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
```

```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Kalibrierung_gui is made visible.
function Kalibrierung_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Kalibrierung_gui (see VARARGIN)

% Choose default command line output for Kalibrierung_gui
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Kalibrierung_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Kalibrierung_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in Aufnahme.
function Aufnahme_Callback(hObject, eventdata, handles)
% hObject    handle to Aufnahme (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
vid = videoinput('winvideo', 1, 'RGB24_800x600');
set(hObject,'String','läuft ...');
set(hObject,'Enable','off')
src = getselectedsource(vid);
frameRates = set(src, 'FrameRate');
src.FrameRate = frameRates{1};
```

```
Bilder = 50;
set(vid, 'FramesPerTrigger', Bilder);
% set(vid.Source,'WhiteBalance',5000)
triggerconfig(vid, 'manual');
start(vid);
pause(2)
trigger(vid);
[frames, timeStamp] = getdata(vid);

for i = 1 : Bilder
    [Mitte_Kugel z d] = fncBildPos_HSV(frames(:,:,:,i));
    Kugel(i,:) = Mitte_Kugel;
end

Mitte_Kugel = mean(Kugel);
axes(handles.axes1);
hold off
imshow(frames(:,:,:,end));
hold on
plot(Mitte_Kugel(1),Mitte_Kugel(2),'+');
stop(vid);
set(handles.text1,'String',num2str(Mitte_Kugel(1)));
set(handles.text2,'String',num2str(Mitte_Kugel(2)));
set(hObject,'String','Aufnahme');
set(hObject,'Enable','on')


% --- Executes on button press in Eintragen.
function Eintragen_Callback(hObject, eventdata, handles)
% hObject    handle to Eintragen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Data = get(handles.uitable1,'Data');
Messpunkt = str2num(get(handles.edit1,'String'));
XPos = str2num(get(handles.edit2,'String'));
YPos = str2num(get(handles.edit3,'String'));
XPixel = str2num(get(handles.text1,'String'));
YPixel = str2num(get(handles.text2,'String'));
Data{Messpunkt,1} = XPos;
Data{Messpunkt,2} = YPos;
Data{Messpunkt,3} = XPixel;
Data{Messpunkt,4} = YPixel;
set(handles.uitable1,'Data',Data);
Messpunkt = Messpunkt + 1;
set(handles.edit1,'String',num2str(Messpunkt))


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes1


% --- Executes on button press in Regression.
function Regression_Callback(hObject, eventdata, handles)
% hObject    handle to Regression (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Data = cell2mat(get(handles.uitable1,'Data'));
% [r c] = size(Data);
XPos = Data(:,1);
YPos = Data(:,2);
XPixel = Data(:,3);
YPixel = Data(:,4);

Dateiname = get(handles.Dateiname,'String');
Dateiname = [Dateiname '.mat'];

c = ones(length(XPixel),1);

Reg = [c, XPixel, YPixel, XPixel.*YPixel];
a = regress(XPos,Reg);
b = regress(YPos,Reg);

save(Dateiname,'a','b','XPos','YPos','XPixel','YPixel')
```

```
function Dateiname_Callback(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Dateiname as text
%        str2double(get(hObject,'String')) returns contents of Dateiname ...
    as a double


% --- Executes during object creation, after setting all properties.
function Dateiname_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end
```

A.12 Matlab GUI: Positionsmessung

```
function varargout = Messsung_Punkt_gui(varargin)
% MESSSUNG_PUNKT_GUI MATLAB code for Messsung_Punkt_gui.fig
%      MESSSUNG_PUNKT_GUI, by itself, creates a new MESSSUNG_PUNKT_GUI ...
    or raises the existing
%      singleton*.
%
%      H = MESSSUNG_PUNKT_GUI returns the handle to a new ...
    MESSSUNG_PUNKT_GUI or the handle to
%      the existing singleton*.
%
%      MESSSUNG_PUNKT_GUI('CALLBACK',hObject,eventData,handles,...) ...
    calls the local
%      function named CALLBACK in MESSSUNG_PUNKT_GUI.M with the given ...
    input arguments.
%
%      MESSSUNG_PUNKT_GUI('Property','Value',...) creates a new ...
    MESSSUNG_PUNKT_GUI or raises the
%      existing singleton*.  Starting from the left, property value ...
    pairs are
%      applied to the GUI before Messsung_Punkt_gui_OpeningFcn gets ...
    called.  An
```

```
%      unrecognized property name or invalid value makes property ...
    application
%      stop.  All inputs are passed to Messsung_Punkt_gui_OpeningFcn via ...
    varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Messsung_Punkt_gui

% Last Modified by GUIDE v2.5 21-Nov-2013 12:08:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Messsung_Punkt_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @Messsung_Punkt_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Messsung_Punkt_gui is made visible.
function Messsung_Punkt_gui_OpeningFcn(hObject, eventdata, handles, ...
    varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Messsung_Punkt_gui (see VARARGIN)

% Choose default command line output for Messsung_Punkt_gui
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Messsung_Punkt_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Messsung_Punkt_gui_OutputFcn(hObject, eventdata, ...
    handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in Aufnahme.
function Aufnahme_Callback(hObject, eventdata, handles)
% hObject    handle to Aufnahme (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(hObject,'String','läuft ...');
set(hObject,'Enable','off')
vid = videoinput('winvideo', 1, 'RGB24_800x600');
src = getselectedsource(vid);
frameRates = set(src, 'FrameRate');
Bilder = 10;
set(vid, 'FramesPerTrigger', Bilder);
src.FrameRate = frameRates{1};
% set(vid.Source,'WhiteBalance',5000);
triggerconfig(vid, 'manual');
start(vid);
pause(2)
trigger(vid);
[frames, timeStamp] = getdata(vid);

for i = 1 : Bilder
    [Mitte_Kugel z d] = fncBildPos_HSV(frames(:,:,:,i));
    Kugel(i,:) = Mitte_Kugel;
end

Kugel_mean = mean(Kugel);
axes(handles.axes1);
hold off
imshow(frames(:,:,:,end));
hold on
plot(Kugel_mean(1),Kugel_mean(2),'+');
stop(vid);
Dateiname = get(handles.Dateiname,'String');
[X Y] = Cam2Glob(Kugel_mean(1),Kugel_mean(2),Dateiname);
set(handles.text1,'String',num2str(X));
set(handles.text2,'String',num2str(Y));
set(hObject,'String','Aufnahme');
```

```
set(hObject,'Enable','on')


% --- Executes on button press in Eintragen.
function Eintragen_Callback(hObject, eventdata, handles)
% hObject    handle to Eintragen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Data = get(handles.uitable1,'Data');
Messpunkt = str2num(get(handles.edit1,'String'));
XPos = str2num(get(handles.text1,'String'));
YPos = str2num(get(handles.text2,'String'));
Data{Messpunkt,1} = XPos;
Data{Messpunkt,2} = YPos;

set(handles.uitable1,'Data',Data);
Messpunkt = Messpunkt + 1;
set(handles.edit1,'String',num2str(Messpunkt))



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes1
```

```
function Dateiname_Callback(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Dateiname as text
%        str2double(get(hObject,'String')) returns contents of Dateiname ...
    as a double


% --- Executes during object creation, after setting all properties.
function Dateiname_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Speichern.
function Speichern_Callback(hObject, eventdata, handles)
% hObject    handle to Speichern (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Data = get(handles.uitable1,'Data');
X = cell2mat(Data(:,1));
Y = cell2mat(Data(:,2));
Dateiname = get(handles.edit6,'String');
Dateiname = [Dateiname '.mat'];
save(Dateiname,'X','Y')



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as ...
    a double


% --- Executes during object creation, after setting all properties.
```

```
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

A.13 Matlab GUI: Wegmessung

```
function varargout = Messsung_Bahn_gui(varargin)
% MESSSUNG_BAHN_GUI MATLAB code for Messsung_Bahn_gui.fig
%      MESSSUNG_BAHN_GUI, by itself, creates a new MESSSUNG_BAHN_GUI or ...
    raises the existing
%      singleton*.
%
%      H = MESSSUNG_BAHN_GUI returns the handle to a new ...
    MESSSUNG_BAHN_GUI or the handle to
%      the existing singleton*.
%
%      MESSSUNG_BAHN_GUI('CALLBACK',hObject,eventData,handles,...) calls ...
    the local
%      function named CALLBACK in MESSSUNG_BAHN_GUI.M with the given ...
    input arguments.
%
%      MESSSUNG_BAHN_GUI('Property','Value',...) creates a new ...
    MESSSUNG_BAHN_GUI or raises the
%      existing singleton*.  Starting from the left, property value ...
    pairs are
%      applied to the GUI before Messsung_Bahn_gui_OpeningFcn gets ...
    called.  An
%      unrecognized property name or invalid value makes property ...
    application
%      stop.  All inputs are passed to Messsung_Bahn_gui_OpeningFcn via ...
    varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Messsung_Bahn_gui
```

```
% Last Modified by GUIDE v2.5 20-Nov-2013 17:59:43

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Messsung_Bahn_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @Messsung_Bahn_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Messsung_Bahn_gui is made visible.
function Messsung_Bahn_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Messsung_Bahn_gui (see VARARGIN)

% Choose default command line output for Messsung_Bahn_gui
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Messsung_Bahn_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Messsung_Bahn_gui_OutputFcn(hObject, eventdata, ...
    handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

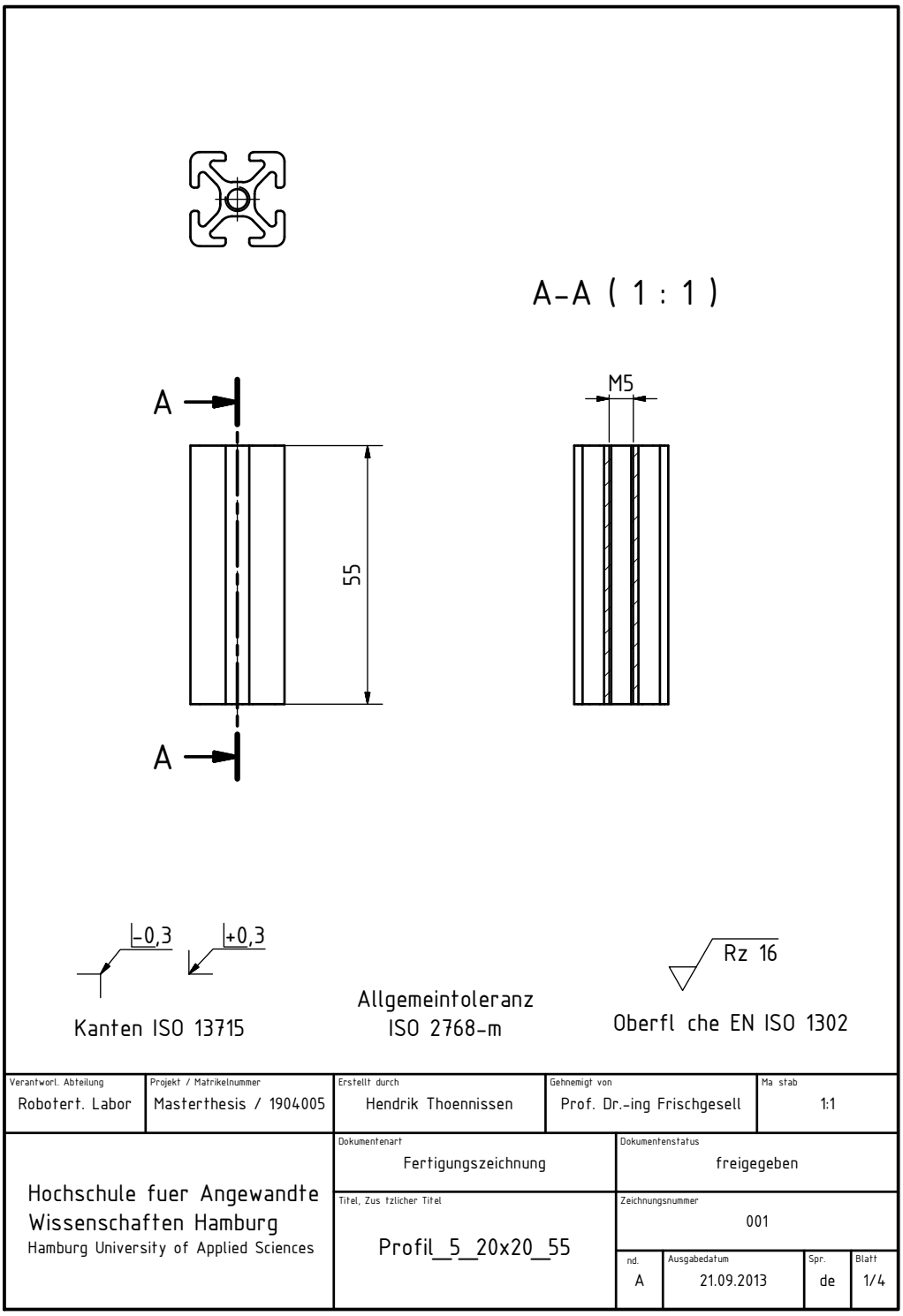
```
% --- Executes on button press in Aufnahme.
function Aufnahme_Callback(hObject, eventdata, handles)
% hObject    handle to Aufnahme (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


set(hObject,'Enable','off')
vid = videoinput('winvideo', 1, 'RGB24_800x600');
src = getselectedsource(vid);
frameRates = set(src, 'FrameRate');
actualRate = str2num( frameRates{4} );
src.FrameRate = frameRates{4};
Dauer = str2num(get(handles.edit1,'String'));
set(vid, 'FramesPerTrigger', Dauer*actualRate);
% set(vid.Source,'WhiteBalance',5000)
triggerconfig(vid, 'manual');
start(vid);

for i = 1 : 5
    set(hObject,'String',num2str(6-i));
    pause(1)
end
trigger(vid);
set(hObject,'String','läuft ...');
[frames, timeStamp] = getdata(vid);
stop(vid);
for i = 1 : Dauer*actualRate
    [Mitte_Kugel z d] = fncBildPos_HSV(frames(:,:,:,i));
    Kugel(i,:) = Mitte_Kugel;
end
Dateiname = get(handles.Dateiname,'String');
[X Y] = Cam2Glob(Kugel(:,1),Kugel(:,2),Dateiname);
axes(handles.axes1);
hold off
plot(X,Y)
hold on
set(handles.uitable1,'Data',[X Y timeStamp]);


set(hObject,'String','Aufnahme');
set(hObject,'Enable','on')


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
```

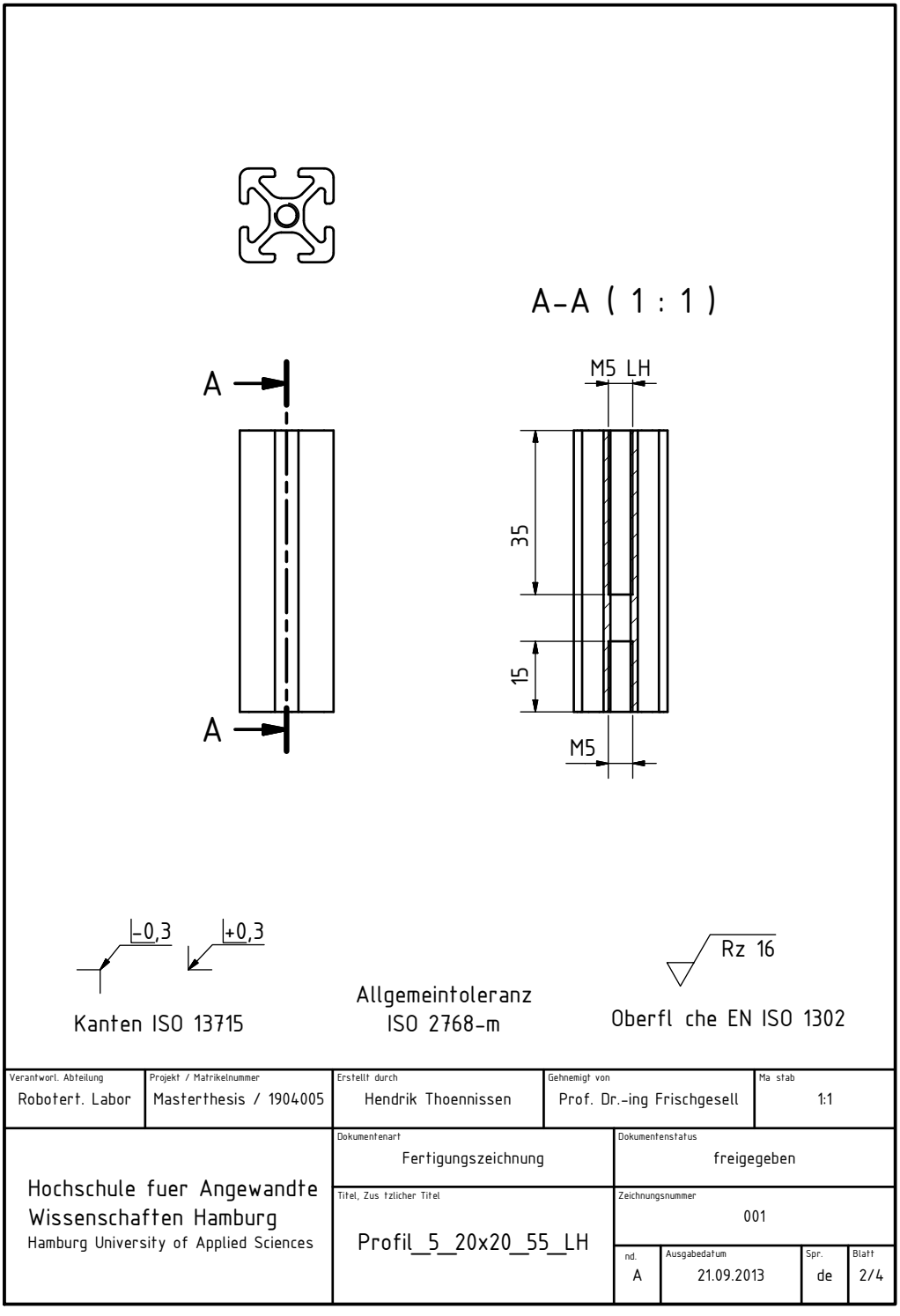
```
%        str2double(get(hObject,'String')) returns contents of edit1 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
```

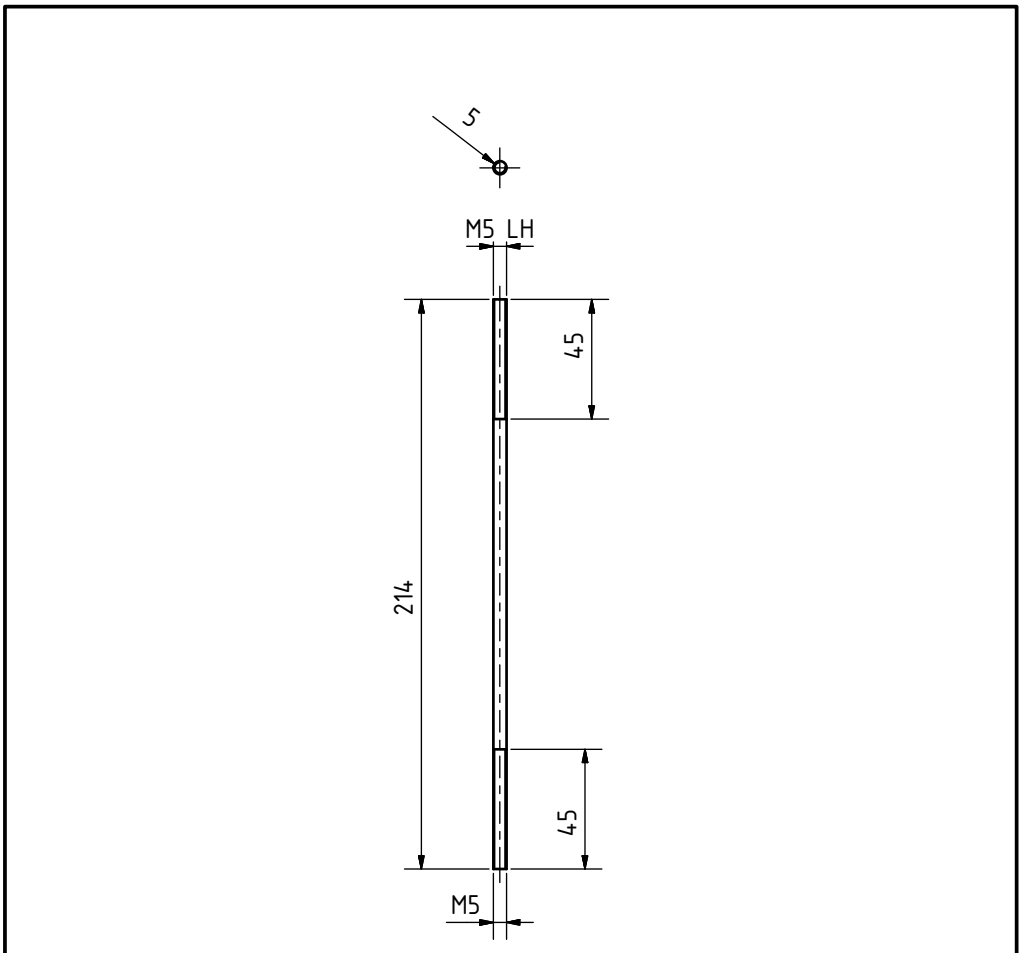
```
%        str2double(get(hObject,'String')) returns contents of edit3 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate axes1


function Dateiname_Callback(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Dateiname as text
%        str2double(get(hObject,'String')) returns contents of Dateiname ...
    as a double


% --- Executes during object creation, after setting all properties.
function Dateiname_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Dateiname (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Save.
```

```
function Save_Callback(hObject, eventdata, handles)
% hObject    handle to Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Data = get(handles.uitable1,'Data');
X = Data(:,1);
Y = Data(:,2);
timeStamp = Data(:,3);
Dateiname = get(handles.edit5,'String');
Dateiname = [Dateiname '.mat'];
save(Dateiname,'X','Y','timeStamp')


function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as ...
    a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end
```

A.14 Fertigungszeichnungen



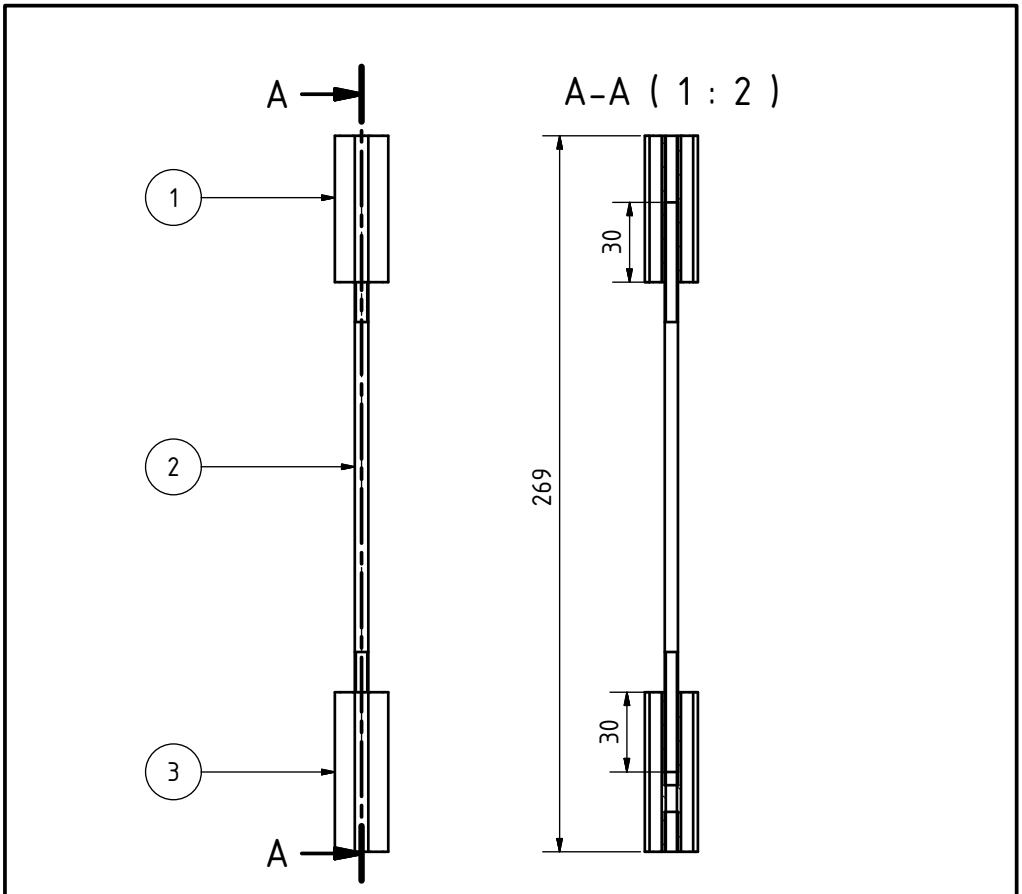

A-A (1 : 1)
A
A
M5 LH
35
15
M5
-0,3
+0,3
Kanten ISO 13715
Allgemeintoleranz
ISO 2768-m
Rz 16
Oberfl che EN ISO 1302
Verantwortl. Abteilung
Robotert. Labor
Projekt / Matrikelnummer
Masterthesis / 1904005
Erstellt durch
Hendrik Thoennissen
Gehnemigt von
Prof. Dr.-ing Frischgesell
Ma stab
1:1
Hochschule fuer Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences
Dokumentenart
Fertigungszeichnung
Dokumentenstatus
freigegeben
Titel, Zus tzlicher Titel
Profil_5_20x20_55_LH
Zeichnungsnummer
001
nd.
A
Ausgabedatum
21.09.2013
Spr.
de
Blatt
2/4



Verantwortl. Abteilung	Projekt / Matrikelnummer	Erstellt durch	Genehmigt von	Ma stab
Robotert. Labor	Masterthesis / 1904005	Hendrik Thoennissen	Prof. Dr.-ing Frischgesell	1:2

	Dokumentenart	Dokumentenstatus
Hochschule fuer Angewandte Wissenschaften Hamburg Hamburg University of Applied Sciences	Fertigungszeichnung	freigegeben
	Titel, Zus tzlicher Titel: Rundstab	Zeichnungsnummer: 003

nd.	Ausgabedatum	Spr.	Blatt
A	21.09.2013	de	3/4



TEILELISTE

OBJEKT	ANZAHL	BAUTEILNUMMER	BESCHREIBUNG
1	1	Profil_5_20x20_55	
2	1	Rundstab	
3	1	Profil_5_20x20_55_LH	

Verantwortl. Abteilung	Projekt / Matrikelnummer	Erstellt durch	Genehmigt von	Ma stab
Robotert. Labor	Masterthesis / 1904005	Hendrik Thoennissen	Prof. Dr.-ing Frischgesell	1:2

Hochschule fuer Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Dokumentenart	Dokumentenstatus
Zusammenbau	freigegeben

Titel, Zus tzlicher Titel: elastischer Arm

Zeichnungsnummer: 004

nd.	Ausgabedatum	Spr.	Blatt
A	21.09.2013	de	4/4

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

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

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

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Thönnißen

Vorname: Hendrik

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

Lageregelung eines redundanten Roboters mit elastischen Elementen

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

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

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Halstenbek	31.01.2014	
Ort	Datum	Unterschrift im Original