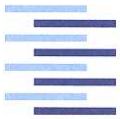


Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Robert Julius

Automatisierung eines Parallelkinematik-Roboters

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Robert Julius

Automatisierung eines Parallelkinematik-Roboters

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing Ulfert Meiners
Zweitgutachter : Prof. Dr. Thomas Holzhüter

Abgegeben am 31.Mai 2012

Robert Julius

Thema der Masterthesis

Automatisierung eines Parallelkinematik-Roboters

Stichworte

Siemens Embedded Automation, WinAC RTX, WinLC RTX, WinAC ODK, S7-mEC, STEP 7, STARTER, Delta-Roboter, S120, Synchronmaschine, Führungsgrößengenerator, Bahninterpolation, Kinematik, Vorsteuerung

Kurzzusammenfassung

Diese Thesis beschreibt die Automatisierung eines Parallelkinematik Roboters unter Verwendung eines S7-mECs und S120 Antriebssysteme von Siemens. Die Regelung des Roboters basiert auf dem Prinzip der kaskadierten Lageregelung mit Vorsteuerung durch einen Führungsgrößengenerator.

Ende des Textes

Robert Julius

Title of the paper

Automation of a Parallelkinematic-Robot

Keywords

Siemens Embedded Automation, WinAC RTX, WinLC RTX, WinAC ODK, S7-mEC, STEP 7, STARTER, Delta-Robot, S120, synchronous machine, reference variable generator, path interpolation, Kinematic, reference-variable feedforward control

Abstract

This Thesis describes the automation of a Parallelkinematic-Robot using a S7-mEC and a S120 drive system manufactured by Siemens. The controling of the robot is based on a cascaded position control with a feedforward control using a generator of reference variables.

End of text

Inhaltverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AWL	Anweisungsliste
CU	Control Unit
CMI	Controller Management Interface
CCX	Custom Code Extension
CSV	Comma-Separated Values
DB	Datenbaustein
DLL	Dynamic Link Library
FI-Schutzschalter	Fehlerstromschutzschalter
FC	Funktion
FB	Funktionsbaustein
FUP	Funktionsplan
GSD	Geräte-Stammdaten-Datei
IO	Input Output
LU	Lastumdrehungen
MFC	Microsoft Foundation Classes
ODK	Open Development Kit
OB	Organisationsbaustein
PC	Personal Computer
PCI	Peripheral Component Interconnect
PN	PROFINET
PEW/PAW	Prozesseingangs- bzw. Ausgangswörter
PTP	Point to Point
RTDLL	Real-Time Dynamic Link Library
SMX	Shared Memory Extension
SFB	Systemfunktionsbaustein
TCP	Tool Central Point
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP/IP	User Datagram Protocol/Internet Protocol
WinAC RTX	Windows Automation Center Real-Time Extension
WinLC RTX	Windows Logic Controller Real-Time Extension

1 Einleitung

1.1 Motivation

In das Verbundprojekt „Autonome Systeme“ des Masterstudiengangs Automatisierungstechnik, soll ein Parallelkinematik-Roboter integriert werden. Grund hierfür ist einerseits der Bedarf eines zusätzlichen Gewerkes für die steigende Zahl Studierender im Studiengang. Andererseits zeigt der Roboter, welche Möglichkeiten und Kenntnisse ein Studium der Ingenieurswissenschaften beinhaltet. Somit stellt das fertige Produkt eine gute Werbung für den Studiengang dar.
Meine persönliche Motivation ist die Durchführung eines komplexen Projektes, von der Konzeptionierung bis zur Inbetriebnahme. Dabei beinhaltet die Aufgabe Themengebiete aus der Antriebstechnik, Steuer- und Regelungstechnik, Echtzeitprogrammierung und Modellierung. Somit ergibt sich die Möglichkeit, einen großen Teil der im Studium behandelten Themengebiete anzuwenden und praktische Erfahrungen zu sammeln. Außerdem spielt das Interesse am Endprodukt eine wichtige Rolle, das mich über alle Phasen der Masterthesis motivierte und so zum Erfolg beigetragen hat.
Parallelkinematik-Roboter sind zwar schon lange und erfolgreich in Industrie und Lehre im Einsatz, wurden aber nach meinen Recherchen noch nie mit der hier verwendeten Hard- und Software realisiert.

1.2 Aufgabenstellung

Ziel der Masterthesis ist es, unter Einsatz eines Siemens Embedded Controllers, die Entwicklung der Software für einen Parallelkinematik-Roboter mit pneumatischen Greifwerkzeug und XYZ-Positionierung. Ebenfalls sind die erforderlichen Schnittstellen im Rahmen des Projektes zu definieren. Der Roboter soll in der Lage sein, innerhalb seines Arbeitsbereiches, Werkstücke dynamisch von einer Position zur anderen zu transportieren. Der mechanische Aufbau des Roboters ist gegeben. Die erforderliche Hard- und Software wird dem Studenten zur Verfügung gestellt.

Folgende Inhalte sollen realisiert werden:

- Mathematisches Modell:
  - Inverse-Kinematik
  - Vorwärts-Kinematik
  - Dynamische Bewegungsgleichungen
  - Bahninterpolation und Führungsgrößengenerator
- Steuerungstechnische Lösung unter Einsatz von SIMATIC STEP 7 und C++ mittels des ODKs[1]
- Dynamische Lageregelung der Antriebe mittels STARTER und STEP 7
- Grafische Bedienoberfläche in C++ (Läuft auf dem Embedded Controller)
- Festlegung eines geeigneten Protokolls zur externen Ansteuerung des Roboters über TCP/IP[2] und UDP/IP[3]
- Betriebsarten:
  - Manuelle Bedienung durch Vorgabe der Koordinaten und Ansteuerung des pneumatischen Saugers
  - Bedienung mittels TCP/IP und UDP/IP von einem Zweitrechner
  - Demonstrationsfahrt
- Elektronischer Aufbau des Roboters

1.3 Vorgehensweise

Nach Vergabe des Themas der Masterthesis folgte zunächst eine von Siemens durchgeführte Schulung in Köln, um sich mit der Hard- und Software von „Siemens Embedded Automation“ vertraut zu machen. Diese Schulung wurde noch vor dem Start der Bearbeitungszeit besucht.

Zu Beginn der Thesis erfolgte zunächst eine Einarbeitung in die vorhandene Hard- und Software, um die Machbarkeit der Automatisierungsaufgabe zu prüfen. Dazu wurde eines der drei Antriebssysteme in Betrieb genommen und über eine C++-Applikation gesteuert.

Nachdem sichergestellt war, dass der Roboter mit den zur Verfügung stehenden Mitteln automatisiert werden kann, folgte die Modellierung und Simulation des Parallelkinematik-

[1] Open Development Kit

[2] Transmission Control Protocol/Internet Protocol

[3] User Datagram Protocol/Internet Protocol

Roboters. Außerdem wurde in dieser Phase beschlossen, den Roboter mit einer kaskadierten Lageregelung mit Vorsteuerung über einen Führungsgrößengenerator zu regeln.
Als das Gesamtsystem in Betrieb genommen werden sollte stellte sich heraus, dass es dazu in einem abgeschlossenen Schaltschrank verbaut werden muss. Grund hierfür war die Gewährleisung der Sicherheit von Studenten und Mitarbeitern.
Nachdem der gesamte elektrotechnische Aufbau in einem Schaltschrank verbaut war, konnten die verbleibenden beiden Antriebssysteme in Betrieb genommen werden. Der mechanische Aufbau des Roboters war zu diesem Zeitpunkt bereits vorhanden.
Als nächstes wurde die Software der Steuerungsapplikation und der Bedienoberfläche fertiggestellt und das Gesamtsystem in Betrieb genommen. Dabei wurde zuerst der Handbetrieb, dann die externe Steuerung über TCP/IP und UDP/IP und zum Schluss die geforderte Demonstrationsfahrt implementiert und getestet.

1.4 Aufbau der Masterthesis

Der Aufbau der Masterthesis entspricht im Wesentlichen der Vorgehensweise bei der Durchführung der Automatisierungsaufgabe. In der Einleitung wurden zunächst die Motivation und die Aufgabenstellung der Bachelorthesis erläutert. Im folgenden Kapitel "Grundlagen" wird das Wissen, das zur Umsetzung der Aufgabe notwendig war vorgestellt. Im Kapitel Versuchsaufbau wird die der Arbeit zugrundeliegende Hardware vorgestellt. Darauf folgt die mathematische Modellierung des Roboters und der Bahninterpolation. Im Anschluss wird die Projektierung der Antriebsysteme mittels der Inbetriebnahme-Software STARTER erklärt. Im Kapitel Simulation wird das Gesamtsystem des Roboters mittels Matlab/Simulink simuliert, um die Modellierung des Systems zu testen. Die folgenden Kapitel befassen sich mit der Programmierung des Steuerungsprogrammes und der Bedienoberfläche. Zum Abschluss wird die Inbetriebnahme der Anlage erläutert, bevor die Arbeit zusammengefast wird und ein Ausblick auf Verbesserungen und Weiterentwicklungen gegeben wird.

2 Grundlagen

2.1 Parallelkinematik-Roboter

Ein Parallelkinematik-Roboter, auch Delta-Roboter genannt, ist ein Anfang der 1980er Jahre von Reymond Clavel entwickelter Roboter, für schnelle Pick-and-Place Aufgaben.

Der Roboter verfügt in der hier verwendeten Ausführung über drei Servo-Antriebe, die auf einer oberen Grundplatte in einer Kreisbahn um 120° versetzt angeordnet sind. Auf der Drehachse jedes Antriebes ist ein Oberarm fest verschraubt, der über Kugelgelenke mit dem Unterarm verbunden ist. Alle drei Enden der Oberarme sind, ebenfalls durch Kugelgelenke, mit einer kleineren Arbeitsplatte verbunden. Die Position der Arbeitsplatte stellt sich in Abhängigkeit der Drehachsen der Antriebe ein.
Da alle Antriebe auf der gleichen Höhe montiert sind und alle drei Ober- und Unterarme dieselben Maße haben, ist die Arbeitsplatte immer parallel zur Grundplatte. Daher der Name Parallelkinematik-Roboter. Abbildung 2.1-1 zeigt den prinzipiellen Aufbau eines Delta-Roboters:

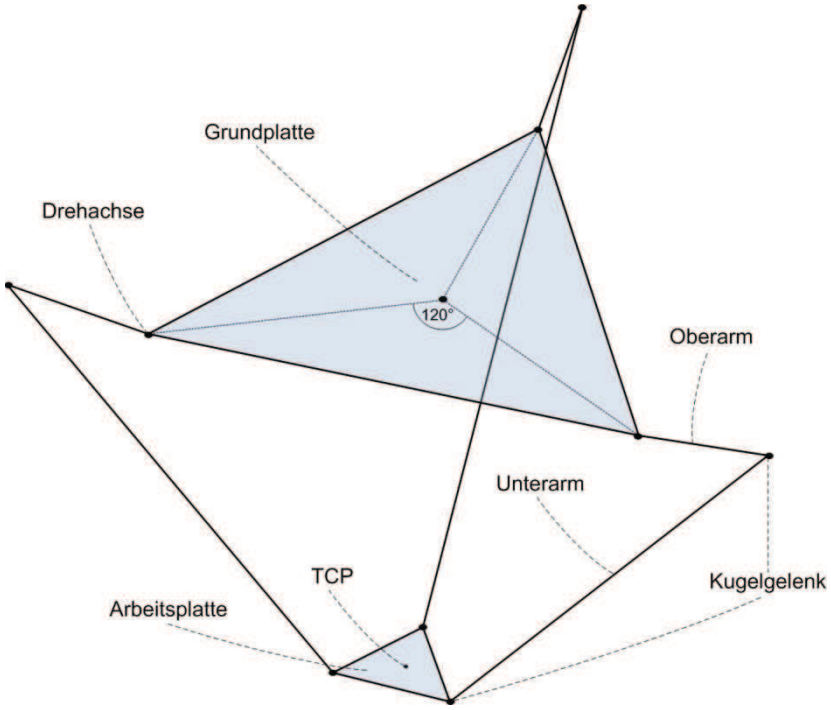


Abbildung 2.1-1:Prinzipieller Aufbau des Delta-Roboters

Da die Massen der Antriebe nicht bewegt werden müssen um die Position des TCPs[4] zu verändern, werden die bewegten Massen gering gehalten. Somit ist eine sehr schnelle Positionierung möglich.

Die folgende Abbildung zeigt den hier verwendeten Roboter:

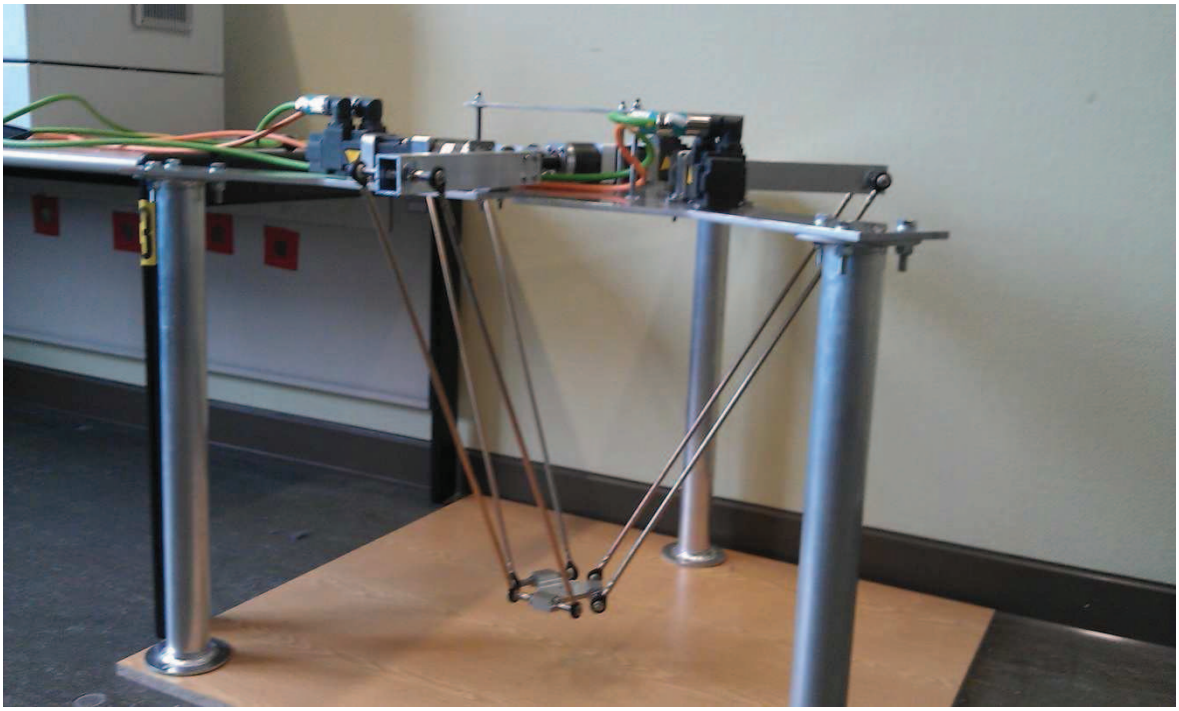


Abbildung 2.1-2:Verwendeter Roboter

Verantwortlich für die mechanische Konstruktion und den Aufbau des Roboters war Herr Eugen Habermann, aus dem Studiengang Mechatronik. Herr Habermann erfüllte diese Aufgabe im Rahmen seiner Studienarbeit. Um das erforderliche Drehmoment der Antriebe, für eine schnelle Positionierung, möglichst gering zu halten, ist zwischen Antrieb und Oberarm ein Getriebe eingebaut. Die Übersetzung beträgt 1:40. [1]

[4] Tool Central Point

2.2 Siemens Embedded Automation

Siemens Automation Embedded beschreibt die Produktpallete der Embedded Controller der Firma Siemens. Sie sind einschaltfertig installierte Embedded PCs mit hoher Industrietauglichkeit und stehen in unterschiedlichen Hard- und Softwarelösungen zur Verfügung. Die Embedded Controller sind in folgende Kategorien unterteilt. [12]

Embedded Panel PC Bundles:
Die Panel PCs mit Touchdisplay eignen sich für einfache Steuerungs-, Kommunikations- und Visualisierungsaufgaben. Sie zeichnen sich durch eine geringe Einbautiefe und maschinennahe Installation aus.

Embedded Box PC-Bundles:
Hierbei handelt es sich um Hutschienen PCs die sich für einfache und komplexe Steuerung-, Kommunikations- und Visualisierungsaufgaben eignen.

SIMATIC S7-modular Embedded Controller:
Diese Embedded Controller, auch S7-mEC genannt, basieren auf der S7-300-Aufbautechnik. Sie bestehen aus dem Grundgerät EC31 (CPU) und modularen Erweiterungsmodulen für PC[5] oder PCI[6]-104 Schnittstellen. Die Controller werden vor allem im Sonder- und Serienmaschinenbau eingesetzt, die neben der Steuerungsaufgabe zusätzlich weitere Automatisierungsaufgaben auf einer Hardware ausführen.

In dieser Automatisierungslösung wird ein S7-mEC in fehlersicherer Ausführung eingesetzt. Er verfügt über ein Windows XP Embedded Betriebssystem. Ergänzt wird das Betriebssystem um das WinAC RTX[7]. Das Automation Center erweitert das Betriebssystem des Embedded PCs um die Funktionalität einer Soft-SPS[8]. Die sogenannte WinLC RTX[9] stellt eine Software-Version einer S7-Steuerung dar, bei der die Echtzeitsteuerung über ein Echtzeitsystem für das Windows-Betriebssystem zur Verfügung gestellt wird. Die Soft-SPS lässt sich wie eine gewöhnliche SIMATIC Steuerung mittels STEP 7 konfigurieren und programmieren. WinLC RTX kann in zwei Prozesse unterteilt werden:

[5] Personal Computer

[6] Peripheral Component Interconnect

[7] Windows Automation Center Real-Time Extension

[8] Speicherprogrammierbare Steuerung

[9] Windows Logic Controller Real-Time Extension

- Ein Prozess läuft auf dem Echtzeitsystem und führt das WinLC RTX-Steuerungsprogramm aus, wobei die Prozesssteuerung die höchste Priorität erhält.
- Der andere Prozess läuft in der Windows-Umgebung und bearbeitet andere Vorgänge, z.B. Kommunikation und Schnittstellen zu Windows-Systemen und -Anwendungen.

Die folgenden von WinLC RTX unterstützen Funktionen wurden zur Lösung der Automatisierungsaufgabe genutzt.

- Profinet IO[10] zur Anbindung dezentraler Peripherie
- Tuning Panel zum Optimieren des Betriebsverhaltens der Systems
- Schnittstelle von WinAC ODK

2.3 WinAC ODK

Das ODK ist eine Schnittstelle zwischen der Soft-SPS WinLC RTX und vom Entwickler erstellten Hochsprachen-Applikationen. Die Schnittstelle ermöglicht es, komplexe Berechnungen innerhalb der Zykluszeit der Steuerung auszuführen und macht die Realisierung der Automatisierungsaufgabe mit der hier verwendeten Hard- und Software erst möglich. Die Anwendungen können in C/C++, C# und Basic programmiert werden. Das ODK bietet hierbei drei verschiedene Schnittstellen zur Einbindung von Hochsprachenapplikationen.

- CCX[11] bietet die Möglichkeit Hochsprachenprogrammierung direkt in das Steuerungsprogramm einzufügen.
- SMX[12] erlaubt es Applikationen, die im Betriebssystem des Embedded PCs laufen, Daten des Steuerungsprogramms zu lesen und zu schreiben.
- CMI[13] ermöglicht es Windows-Applikationen, auf die Funktion des WinAC-Panel zuzugreifen, um somit z.B. die Steuerung zu starten oder zu stoppen.

In dieser Masterthesis werden sowohl CCX als auch SMX genutzt, die im Folgenden genauer erklärt werden.

[10] Input Output

[11] Custom Code Extension

[12] Shared Memory Extension

[13] Controller Management Interface

2.3.1 CCX

CCX Applikationen werden mit Microsoft Visual Studio programmiert und als DLL[14] oder RTDLL[15] auf dem Embedded PC hinterlegt. Zur Erzeugung einer CCX Anwendung kann das „WinAC ODK AppWizard“ genutzt werden. Dieses, von Siemens bereit gestellte, Programm, stellt Projektschablonen für CCX und SMX Anwendungen zur Verfügung.
In diesen Projekten sind alle benötigten Klassen und Bibliotheken vorhanden, um mit dem Steuerungsprogramm zu kommunizieren. Das Steuerungsproramm kann die erstellten DLLs und RTDLLs anschließend, unter Angabe des entsprechenden Verzeichnisses, aufrufen.

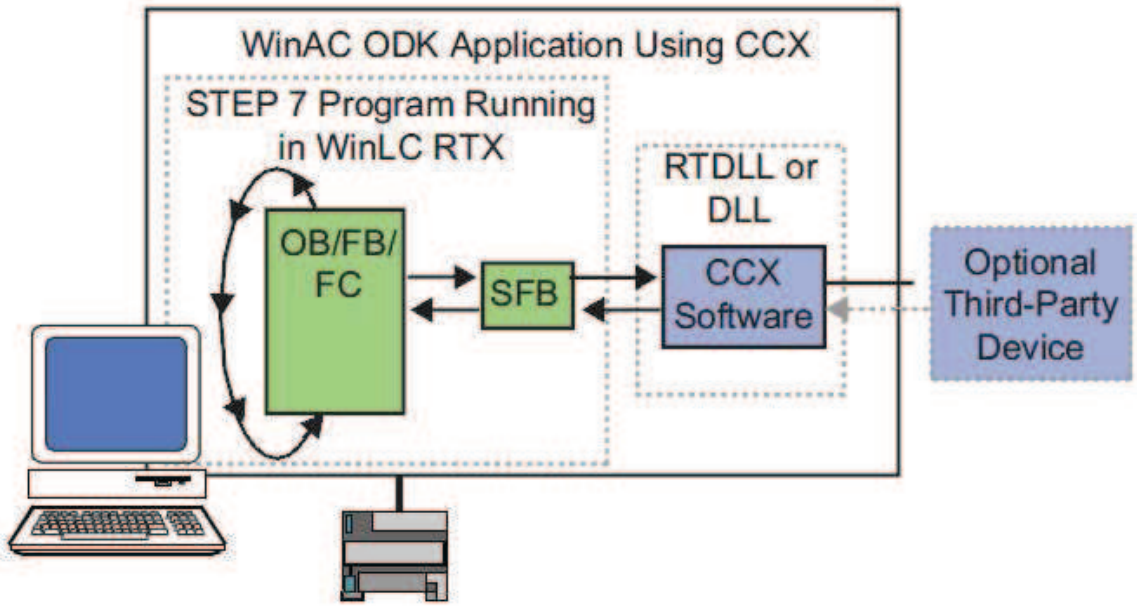


Abbildung 2.3-1:CCX-Aufruf [2]

Zum Aufruf der DLLs werden verschiedene SFBs[16] bereit gestellt die innerhalb des Steuerprogramms aufgerufen werden. Die hier genutzten sind:

SFB65001

Dieser Baustein wird benutzt, um die DLL/RTDLL zu laden und zu initialisieren. Dazu wird diesem Baustein der Verzeichnispfad der CCX-Anwendung übergeben. Der Rückgabewert ist ein Statuswort der Initialisierung. Ist dieses negativ, d.h. das höchste Bit ist gesetzt, liegt ein Fehler bei der Initialisierung vor und die DLL/RTDLL kann anschließend

[14] Dynamic Link Library

[15] Real-Time Dynamic Link Library

[16] Systemfunktionsbaustein

nicht ausgeführt werden. Dieser Baustein soll vorzugsweise im OB100 aufgerufen werden, der zum Start der Steuerung ausgeführt wird.

SFB65002

Der SFB65002 wird für den synchronen Aufruf einer CCX-Applikation aus dem Steuerungsprogramm benutzt. Synchron bedeutet hierbei, zur Zykluszeit der Steuerung. Das Steuerungsprogramm wird unterbrochen solange die Applikation arbeitet. Die Eingänge des Bausteins sind.

- Statuswort des SFB65001 um zu verhindern, dass bei fehlerhafter Initialisierung das Steuerungsprogramm unterbrochen wird.
- Nummer des Subkommandos, das ausgeführt werden soll. Somit ist es möglich verschiedene Funktionen in einer Applikation zu realisieren.
- Pointer auf die Eingangsdatenstruktur
- Pointer auf die Ausgangsdatenstruktur

Die Ein- und Ausgangdatenstrukturen können aus verschiedenen Datentypen bestehen. Der Baustein gibt wieder ein Statuswort zurück, dass die Information über die korrekte Ausführung des SFBs enthält.
Der Programmaufrufe selber sieht wie folgt aus:

- Auswahl des angeforderten Subkommandos
- Lesen der Steuerungsdaten mittels ReadInput-Funktionen
- Bearbeitung der auszuführenden Aktivitäten
- Schreiben der Steuerungsdaten mittels WriteOutput-Funktionen

Das Lesen und Schreiben von Steuerungsdaten kann wie folgt aussehen.

rInput.ODK_ReadS7REAL(int Index ,float value);
rOutput.ODK_WriteS7WORD(int Index ,ODK_Bit16 value);

„rInput“ und „rOutput“ sind dabei Instanzen der entsprechenden ODK-Klassen zum Lesen und Schreiben von Daten. Den Methoden wird der Index in der Struktur übergeben, aus dem gelesen oder geschrieben werden soll. Außerdem wird die zu lesende oder zu schreibende Variable übergeben. Es stehen Methoden für alle in STEP 7 verwendeten Datentypen zu Verfügung.

2.3.2 SMX

SMX Programme sind Applikationen, die im Betriebssystem des S7-mEC ausgeführt werden. Mittels des WinAC ODKs ist es möglich, innerhalb dieser Programme auf Prozesseingangs- und Ausgangswörter der Steuerung zuzugreifen. Da SMX-Projekte auch als MFC[17]-Anwendungen erstellt werden können, ist es mit diesen möglich, Bedienoberflächen in einer Hochsprache wie C/C++ für eine Automatisierungsanwendung zu programmieren.

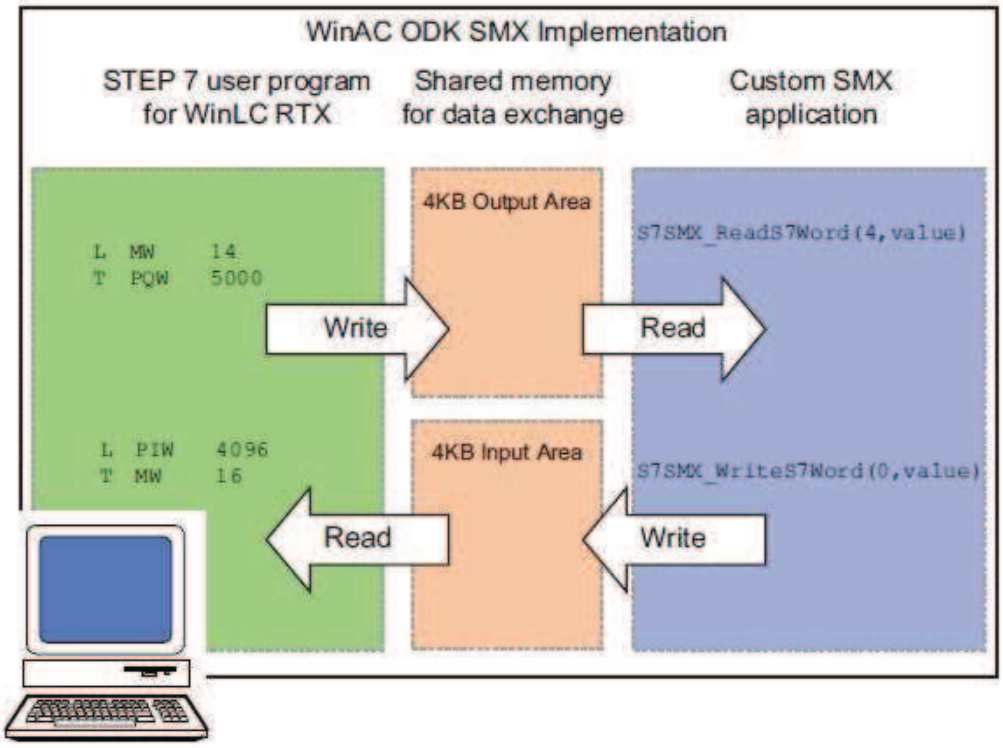


Abbildung 2.3-2:SMX Datenaustausch zwischen S7 und Windowsapplikationen[2]

Zum Datenaustausch sind innerhalb der Soft-SPS 4 KB Input- und Output-Bereiche reserviert. Die Bereiche reichen von PEW/PAW[18] 16384 bis 20478.

Das SMX Programm selbst kann wie eine ganz normale MFC-Anwendung mit Microsoft Visual Studio programmiert werden. Der einzige Unterschied besteht in den mitgelieferten ODK-Klassen, die die oben besprochene Kommunikation realisieren. Dazu muss zuerst die Schnittstelle innerhalb der Anwendung initialisiert werden. Anschließend können ähnlich wie bei CCX, mittels ODK-Methoden, Steuerungsvariablen gelesen und gesteuert werden. Die genaue Implementierung ist im ODK Manual erklärt.[2]

[17] Microsoft Foundation Classes

[18] Prozesseingangs- bzw. Ausgangswörter

2.4 Prinzipieller Programmaufbau

Nach der Einarbeitung in das Thema, der verwendeten Software und der notwendigen Schnittstellen, konnte der prinzipielle Programmablauf bestimmt werden.
Die Bedienoberfläche wird als MFC-Anwendung unter C++ programmiert und auf dem Embedded PC unter Windows gestartet. Mittels der SMX-Schnittstelle kann der Bediener den Roboter steuern. Sie erfüllt folgende Funktionen:

- Kalibrierung der Roboterarme
- Einschalten des Antriebssystems
- Anzeige der aktuellen Position
- Betriebsartenumschaltung
- Handbetrieb zur manuellen Vorgabe einzelner Fahrten
- Demonstrationsfahrt des Roboters zur Vorführung der Dynamik
- TCP- und UDP-Schnittstelle zur externen Steuerung des Roboters
- Messen von Soll- und Istwinkeln für einzelne Fahrten
- Sicherstellung, dass nur Fahrten angetreten werden, die sich innerhalb des Arbeitsbereiches befinden und die Mechanik des Roboters nicht beschädigen

Die Funktionen des Steuerungsprogramms, das auf der Soft-SPS läuft, sind:

- Kommunikation mit der SMX-Schnittstelle der Bedienoberfläche
- Lageregler zur Positionierung der einzelnen Antriebe
- Aufruf des mit Hilfe von CCX in C++ programmierten Führungsgrößengenerators
- Kommunikation mit den Antriebssystemen, Vorgabe von Drehzahl- und Drehmomentsollwerten
- Sicherheitsfunktion zur Verhinderung mechanischer Schäden am Roboter

Die Umrichter der Antriebssysteme selbst erfüllen die Funktion der Drehzahlregelung und die Erfassung der Lageistwerte der einzelnen Antriebe.
Die folgende Abbildung verdeutlicht den Programmablauf.

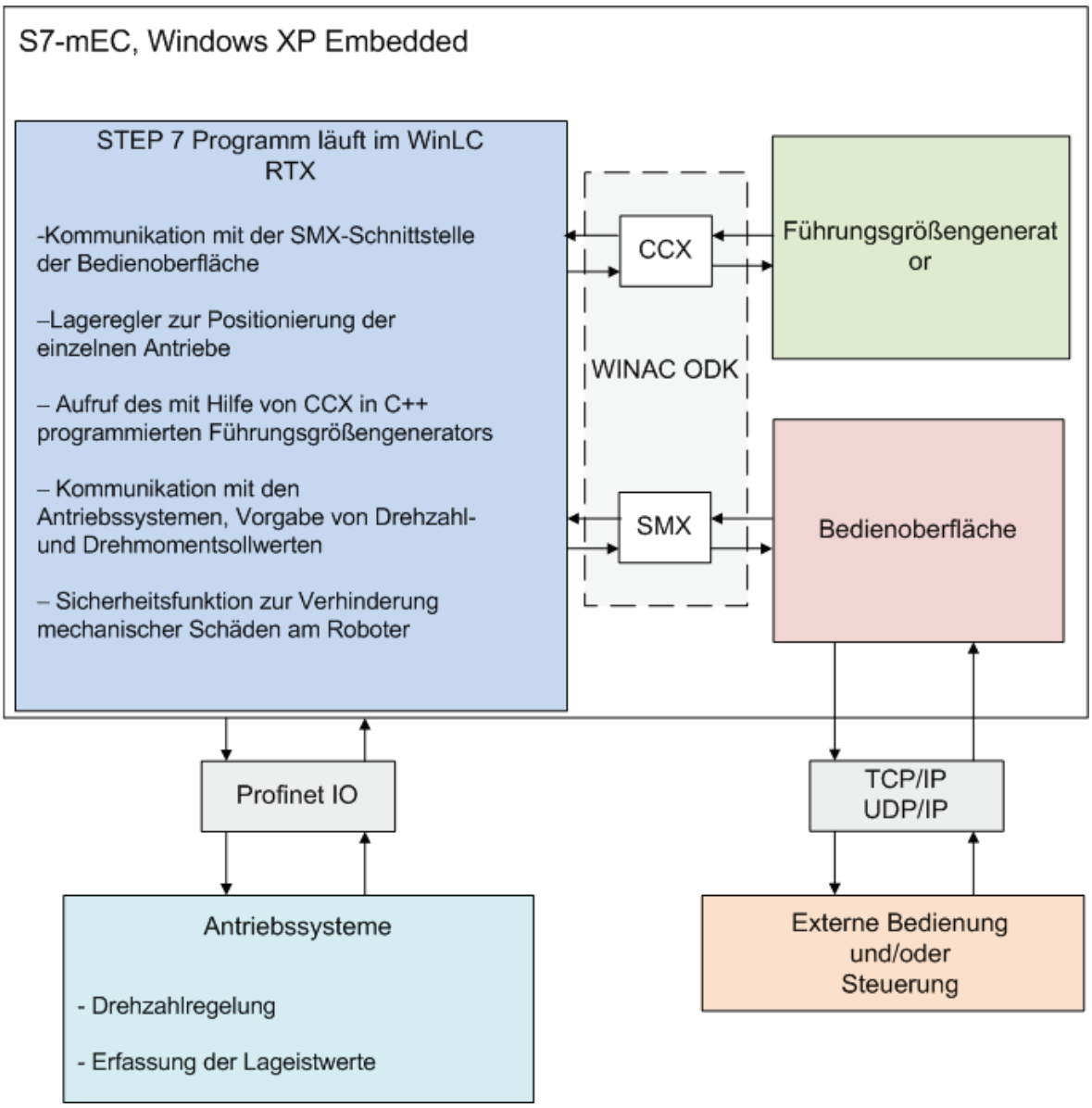


Abbildung 2.4-1:Prinzipieller Programmablauf

3 Technischer Aufbau

In diesem Kapitel werden die eingesetzten technischen Komponenten vorgestellt, die zur Automatisierung des Delta-Roboters eingesetzt werden. Dazu gehören der Embedded PC, das Antriebssystem, der Schaltschrank sowie dessen sicherheitsrelevante Komponenten. Die Komponenten sind entweder Spenden der Firma Siemens oder wurden hinzugekauft. Alle für den technischen Aufbau relevanten Dokumente befinden sich im Anhang A1:.

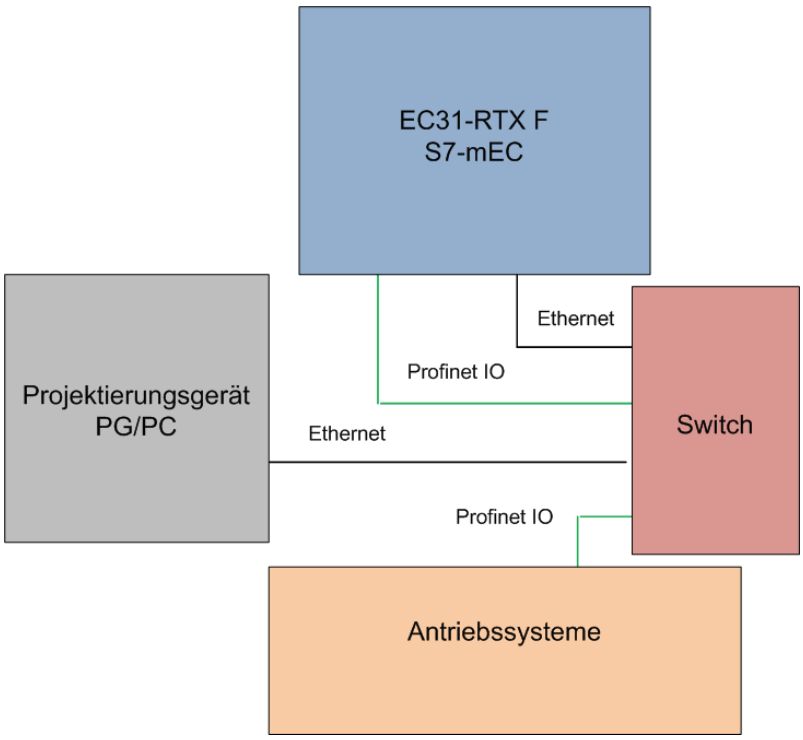


Abbildung 2.4-1:Prinzipieller Aufbau

3.1 EC31-RTX F

Der hier verwendete Embedded Controller EC31-RTX F ist ein PC in S7-300 Bauform mit einem Windows XP Embedded Betriebssystem. Er verfügt außerdem über WinAC RTX F 2009 als Soft-SPS. Die Software ist vorinstalliert und vorkonfiguriert. Der Controller arbeitet ohne Lüfter und verfügt über einen Flash-Speicher. Er ist um die folgenden Baugruppen erweitert.

EM PCI-104

Das Erweiterungsmodul EM PCI-104 bietet die Möglichkeit, die Funktionalität des Embedded PCs um Kommunikations, Speicher, Audio- und schnelle Messtechnik-Karten zu erweitern. Das Erweiterungsmodul erlaubt die Aufnahme von bis zu drei Karten des PCI-104 Standards. In dieser Anwendung wird das Modul nicht benutzt, ist aber für eine mögliche Erweiterung des Systems bereits im Schaltschrank verbaut.

EM PC

Das Erweiterungsmodul EM PC bietet Standard-PC-Schnittstellen und ermöglicht somit den Umgang wie mit einem PC. Neben den Anschlussmöglichkeiten für eine lokale Bedienung und Beobachtung stehen zusätzliche Speicher-Slots und Kommunikations-Schnittstellen zur Verfügung.

Folgende Abbildung zeigt den hier verwendeten S7-mEC und dessen Anschlüsse.

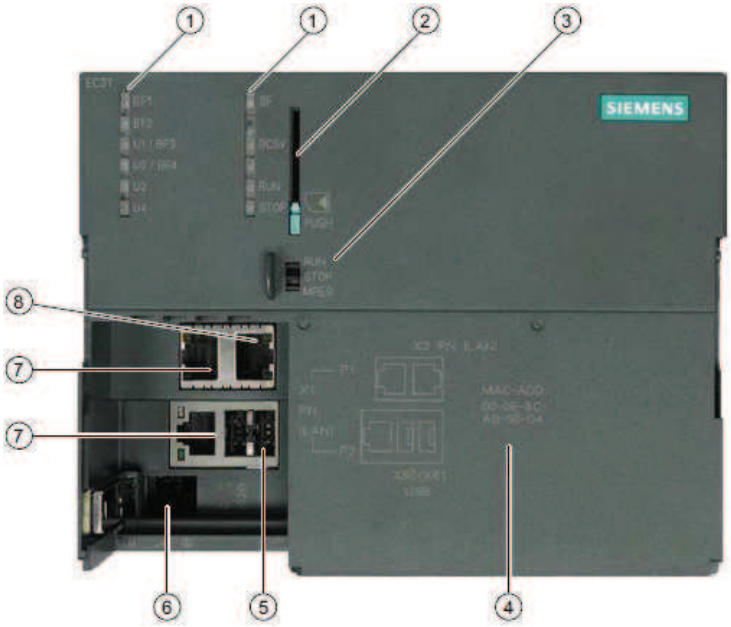


Abbildung 3.1-1:EC31-RTX F [4]

1. LED-Anzeige
2. Schacht für Multi Media Card, inkl. Auswerfer
3. Betriebsartenschalter
4. MAC-Adresse des Standard-Ethernet-Controllers X2 PN(LAM)
5. USB 2.0-Anschluss
6. Anschluss für Spannungsversorgung
7. PROFINET IO-Anschluss (als Submodul der WinAC)
8. Ethernet-Anschluss mit PROFINET-Basisdiensten

3.2 Antriebssystem

In diesem Abschnitt wird das Antriebsystem des Roboters am Beispiel eines der drei identischen Systeme vorgestellt.

Jedes Antriebssystem ist aus der SINAMICS-S120-Baureihe und besteht aus einem Umrichter (PM340) mit untergebauter Netzdrossel und einer CU310-PN[19]. Der Umrichter wird einphasig an das 230V Netz angeschlossen und stellt die dreiphasige Spannungsversorgung der Drehstrommaschine, mit der er über eine Leistungsleitung verbunden ist, bereit. Die CU310 PN ist das Kernstück der Regelung und Steuerung des Antriebes. Die Drehzahlregelung ist nach dem Prinzip der Kaskadenregelung aufgebaut und wird mittels der Inbetriebnahme-Software STARTER von einem externen PC projektiert. In der CU wird außerdem die Kommunikation mit dem S7-mEC realisiert.

Die Geberauswertung des Antriebs erfolgt mit dem Geberauswertungsmodul SMC20. Dieses berechnet aus den analogen Signalen der Geberleitung die Drehzahl, den Lageistwert und die Temperatur des Antriebes. Diese Informationen werden über Ethernet an die CU übertragen.

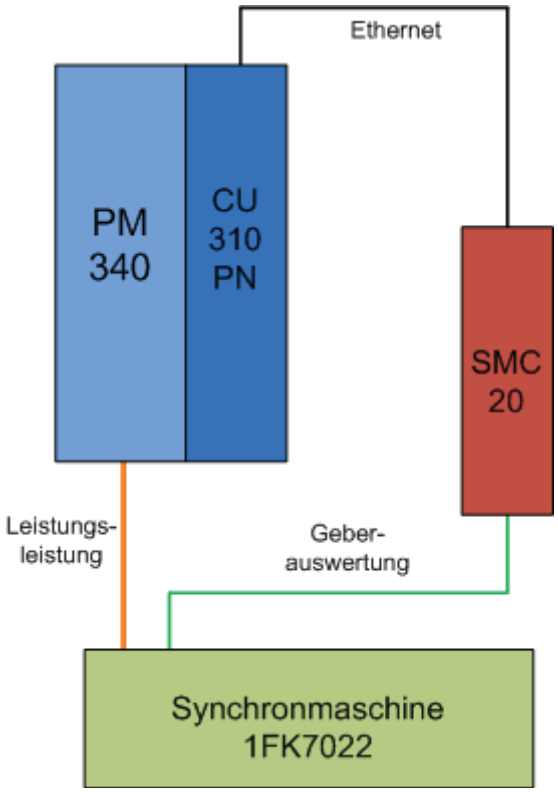


Abbildung 3.2-1:Antriebssystem

[19] PROFINET

Abgesehen von den Antrieben selbst, waren alle Komponenten des Antriebssystems zu Beginn der Masterthesis vorhanden. Die ursprünglichen Antriebe erwiesen sich, mit einem Nenndrehmoment von 0,1Nm, nach der Simulation von Herrn Habermann als zu schwach, um die Arbeitsplatte innerhalb einer Sekunde zu jedem Punkt im Arbeitsbereich des Roboters zu bewegen. Dies wäre nur mit einer Getriebeübersetzung von 1:120 möglich. Da solche Getriebe speziell gefertigt werden müssen und somit mit erheblichen Kosten verbunden sind, hat Herr Habermann einen Servo mit einem Nenndrehmoment von mindestens 0,5Nm vorgeschlagen. Um zu gewährleisten, dass ein anderer Antrieb zur vorhanden Leistungselektronik passt, wurde ein Antrieb aus der gleichen Baureihe mit einem größeren Nenndrehmoment ausgewählt.

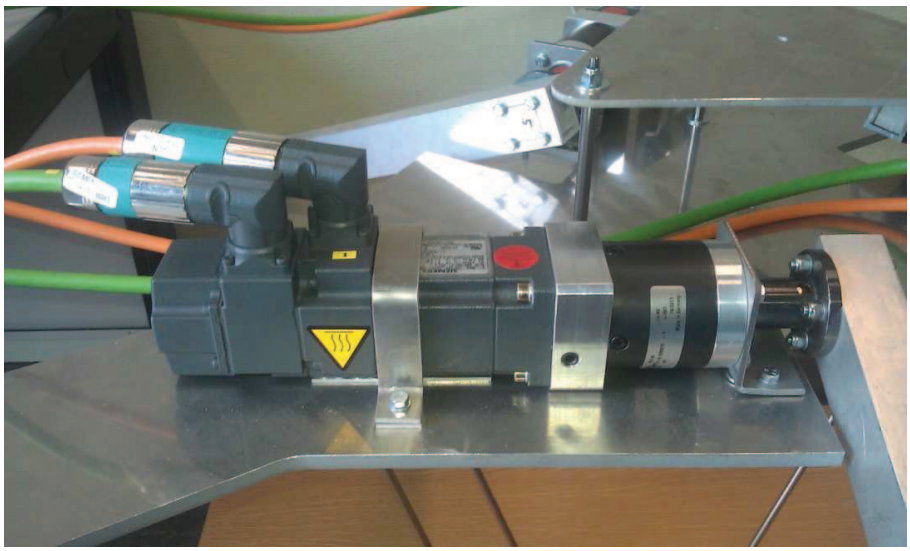


Abbildung 3.2-2:Verwendeter Synchronservo

Der Antrieb selbst ist eine permanenterregte Synchronmaschine der 1FK7-Baureihe von Siemens. Der in den Antrieb integrierte Inkrementalgeber arbeitet nach dem Prinzip der optoelektronischen Abtastung von Teilscheiben im Durchlichtverfahren. Die wichtigsten Kenndaten der Synchronmaschine sind folgende.

Bemessungsdrehzahl	*6000* min^{-1}
Polpaarzahl	*3*
Nenndrehmoment	*0,6Nm*
Nennstrom	*1,4 A*
Trägheitsmoment	*2,8* $\cdot 10^{-4} kgm^2$
Maximaldrehzahl	*10000* min^{-1}
Maximaldrehmoment	*3,68Nm*

Tabelle 1: Kenndaten der Synchronmaschinen

3.3 Schaltschrank

Damit keine Personen zu Schaden kommen und um die Anlage kompakt in das Verbundprojekt des Masterstudiengangs „Automatisierungstechnik“ zu integrieren, wurden alle elektrotechnischen Komponenten in einem Rittal-Wandgehäuse eingebaut. Das Wandgehäuse bietet die Möglichkeit, in zwei Ebenen Komponenten zu verbauen und an diese ohne Demontage des Schaltschrankes heranzukommen. Der Schrank wird über einen Drehstromanschluss versorgt. Jede der drei Phasen ist an einem der drei Umrichter angeschlossen.

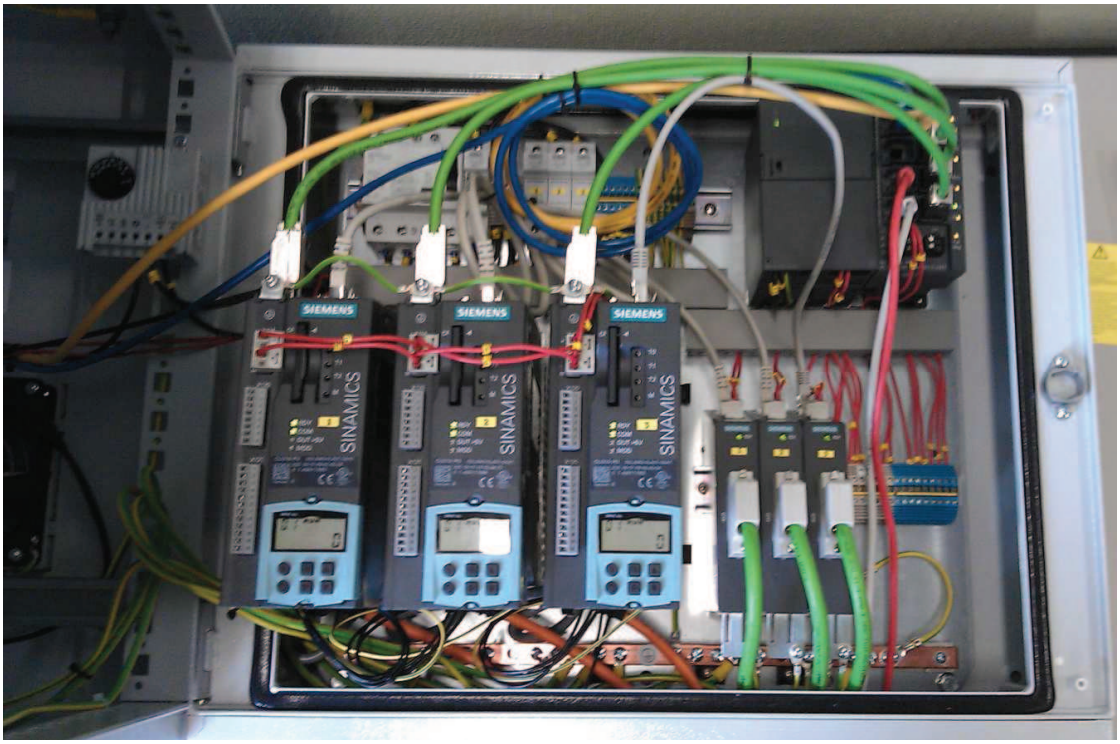


Abbildung 3.3-1:Grundplatte

Die obige Abbildung zeigt die hintere Grundplatte des Schaltschrankes. Auf ihr ist die komplette Leistungselektronik der Anlage montiert. Neben den Antriebssystemen befinden sich hier folgende Komponenten.

FI-Schutzschalter

Der hier eingesetzte FI-Schutzschalter[20] ist ein vierpoliger allstromsensitiver Typ B Schutzschalter der Firma Siemens. Typ B Schutzschalter werden für das Antriebssystem empfohlen, da sie neben Wechselfehlströmen auch glatte Gleichfehlströme erkennen. Der

[20] Fehlerstromschutzschalter

Einsatz dieser allstromsensitiven Schutzschalter ist bei Wechsel- und Frequenzumrichtern, welche im Bereich des Zwischenkreises mit Gleichrichtern arbeiten, üblich. Standard Typ A Schutzschalter würden bei einem Erdschluss hinter der Gleichrichterbrücke durch den dann vorhandenen Gleichfehlerstrom nicht auslösen.

Sicherungen

Als Sicherungen für die Anlage werden zylindrische superflinke gRL-Schmelzsicherungen der Firma SIBA, mit einem Bemessungsstrom von 10A, eingesetzt. Sie werden von Siemens, für die Halbleiterbauelemente des Umrichters empfohlen und sind in den entsprechenden Sicherungshaltern im Schaltschrank verbaut.

Switch

Um die Ethernet- und PROFINET-Schnittstellen der Anlage mit einander zu verbinden, wird ein Switch in S7-300-Bauform von Siemens eingesetzt.

24V DC-Versorgung

Zur Spannungsversorgung des S7-mEC, der CU310-PNs und SMC20 der Antriebssysteme sowie des Switches, ist eine 24V DC-Spannungsversorgung in S7-300-Bauform in den Schaltschrank integriert.

Lüfter

Zur Kühlung des Innenraumes des Schaltschrankes wird ein Lüfter mit Temperaturregler eingesetzt. Der Temperaturregler ist ein Zweipunktregler und schaltet den mit 230V AC versorgten Lüfter ab einer einstellbaren Schwelltemperatur ein. Die eingestellt Schwelltemperatur ist mit 30℃ eingestellt um sicherzustel len, dass sich der Innenraum nicht zu sehr erhitzt.

Hauptschalter

Seitlich am Schaltschrank ist ein Hauptschalter montiert. Er erlaubt es, die Anlage ein- und auszuschalten, ohne den Schrank öffnen zu müssen. Außerdem erfüllt er die Funktion eines Notausschalters.

Gerätesteckdose

Ebenfalls seitlich befindet sich eine Drehstrom-Gerätesteckdose zur Spannungsversorgung des Schaltschrankes.

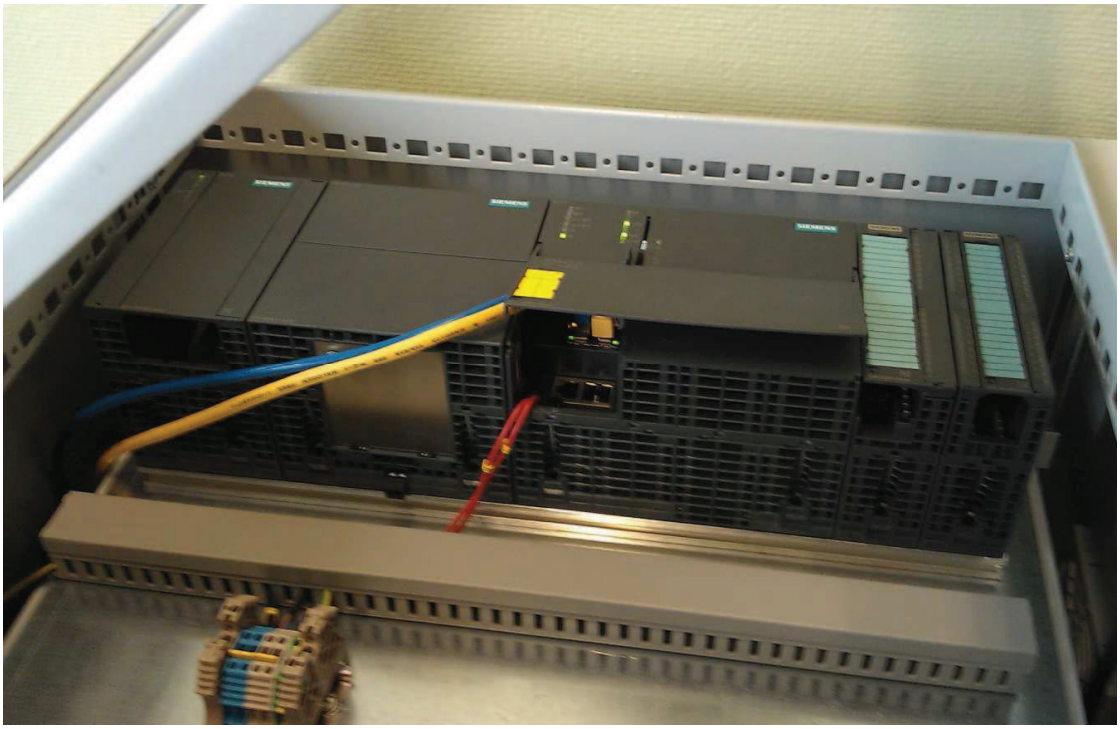


Abbildung 3.3-2:Fronplatte

Auf der Frontplatte ist der unter 3.1 beschriebene S7-mEC eingebaut. Zusätzlich sind, für eine spätere Erweiterung der Anlage um einen pneumatischen Sauger, IO-Baugruppen montiert.

Die Montage des Schaltschrankes wurde in den Räumen der HAW Hamburg durchgeführt. Als Vorbereitung wurden zunächst alle benötigten Komponenten, die noch nicht vorhanden waren, recherchiert. Aus den Angaben der Maße der einzelnen Bauteile konnte die Mindestgröße des Schaltschrankes ermittelt werden. Es wurde das nächstgrößere Model der Rittal-Wandgehäuse ausgewählt, da nur diese Modelreihe die Anforderungen an die Einbautiefe erfüllt.

4 Modellierung

Im folgenden Kapitel wird die Modellierung erläutert, die zur Beschreibung des Roboters notwendig ist. Um die Berechnungen übersichtlicher zu gestalten, sind einige der verwendeten Formelzeichen hier aufgelistet.

$f = 0{,}15m$	*Abstand der Drehachsen vom Koordinatenursprung*
$rf = 0{,}25m$	*Länge der Oberarme*
$re = 0{,}7m$	*Länge der Unterarme*
$e = 0{,}05m$	*Abstand zwischen TCP und den Gelenkpunkten der Arbeitsplatte*
$g = 0{,}81\frac{m}{s^2}$	*Gravitationskonstante*
$m_e = 0{,}4kg$	*Masse der Arbeitsplatte*
$m_{payload}$	*Masse der bewegten Werkstücke*
$m_{re} = 0{,}2kg$	*Masse der Unterarme*
$m_{rf} = 0{,}35kg$	*Masse der Oberarme*
$m_a = 0{,}08kg$	*Masse der Kugelgelenke*
$\ddot{u} = 40$	*Übersetzungsverhältnis von Last- zur Antriebsseite*

Tabelle 2:Roboter Geometrie

4.1 Kinematik

Zunächst müssen die folgenden beiden Probleme gelöst werden. Erstens, ist die Zielposition des TCPs gegeben, müssen aus diesen Angaben im kartesischen Koordinatensystem die erforderlichen Winkel der Roboterarme berechnet werden. Diesen Prozess zur Bestimmung der Winkel nennt man Inverse-Kinematik.
Zweitens, sind die Winkel des Roboters bekannt, muss die Position des TCPs im Koordinatensystem bestimmt werden. Diese Berechnung nennt man Forward-Kinematik. Den Berechnungen liegt die Veröffentlichung von Jon Martínez García zugrunde. [5]

Zur Übersicht zeigt Abbildung 4.1-1, die definierten Bezeichnungen der Robotergeometrie sowie die Definition des Koordinatensystems und dessen Ursprung.

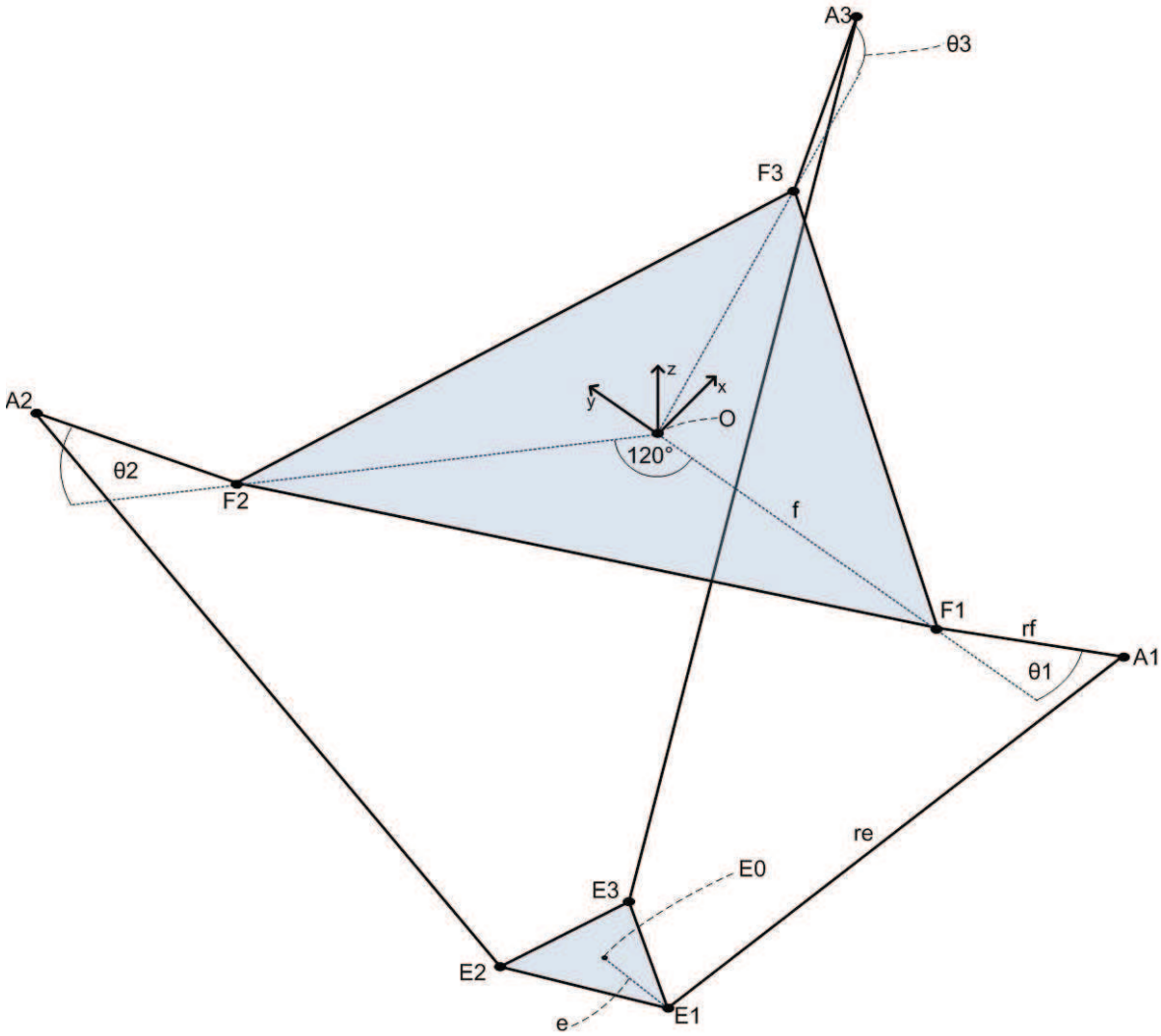


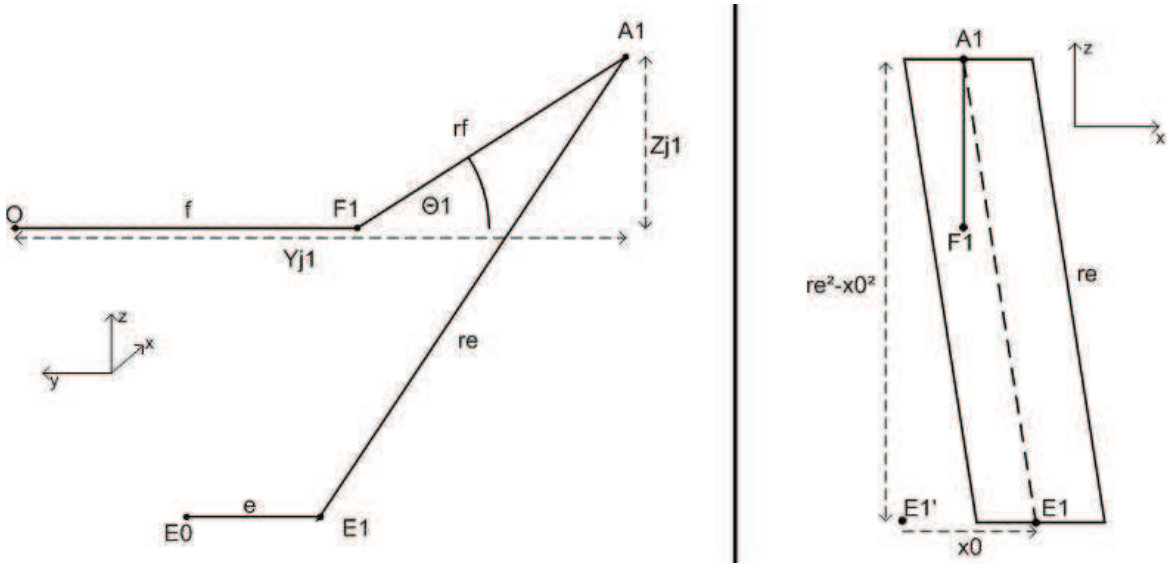
Abbildung 4.1-1:Bezeichnungen der Robotergeometrie

O	*Koordinatenursprung*
F_i	*Gelenkpunkte zw. Grundplatte und Oberarm*
A_i	*Gelenkpunkte zw. Oberarm und Unterarm*
E_i	*Gelenkpunkte zw. Unterarm und Arbeitsplatte*
E_0	*Mittelpunkt der Arbeitsplatte(TCP)*

Tabelle 3: Definition der Gelenkpunkte

4.1.1 Inverse-Kinematik

Der Ansatz zur Berechnung der Oberarmwinkel aus der Position des TCPs heraus ist folgender. Laut Definition liegen die Punkte F_1 und J_1 des ersten Oberarms auf der Y-Achse des Koordinatensystem und liegen somit immer auf dem Nullpunkt der X-Achse. Es sind lediglich die Punkte E_0, also die Position des TCPs und F_1 bekannt.



$$\vec{O} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\ \vec{F}_1 = \begin{pmatrix} x_{F1} \\ y_{F1} \\ z_{F1} \end{pmatrix} = \begin{pmatrix} 0 \\ -f \\ 0 \end{pmatrix};\ \vec{A}_1 = \begin{pmatrix} x_{J1} \\ y_{J1} \\ z_{J1} \end{pmatrix};\ \vec{E}_0 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix};\ \vec{E}_1 = \begin{pmatrix} x_0 \\ y_0 - e \\ z_0 \end{pmatrix};$$

$$\vec{E}_1' = \begin{pmatrix} x_{E1'} \\ y_{E1'} \\ z_{E1'} \end{pmatrix} = \begin{pmatrix} 0 \\ y_0 - e \\ z_0 \end{pmatrix}$$

Abbildung 4.1-2:Ansatz der inversen Kinematik

Aus Abbildung 4.1-2 werden nun die beiden folgenden Gleichungen aufgestellen.

$$rf^2 = \left(y_{J1} - y_{F1}\right)^2 + {z_{J1}}^2 \qquad \textbf{(4-1)}$$

$$re^2 - {x_0}^2 = \left(y_{J1} - y_{E1\prime}\right)^2 + \left(z_{J1} - z_{E1\prime}\right)^2 \qquad \textbf{(4-2)}$$

Da zur Berechnung von θ_1 die Werte y_{J1} als auch z_{J1} bekannt sein müssen, werden sie nun aus Gl.(4-1) und Gl.(4-2) berechnet.

Gl.(4-1) nach z_{J1} aufgelöst:

$$z_{J1} = \sqrt{-\left(y_{J1} - y_{F1}\right)^2 + rf^2} \qquad \textbf{(4-3)}$$

Gl.(4-2) nach z_{J1} aufgelöst:

$$z_{J1} = b \cdot y_{J1} + a \tag{4-4}$$

mit

$$a = \frac{{x_0}^2 + {y_{E1\prime}}^2 + {z_0}^2 + rf^2 - re^2 - {y_{F1}}^2}{2 \cdot z_0} \; ; \; b = \frac{y_{J1} - y_{E1\prime}}{z_0}$$

Um auf eine Gleichung mit nur einer Unbekannten zu kommen, werden Gl.(4-3) und Gl.(4-4) gleich gesetzt und nach y_{J1} aufgelöst.

$${y_{J1}}^2 + \frac{2(a \cdot b - y_{F1})}{b^2 + 1} y_{J1} + \frac{a^2 + {y_{F1}}^2 - rf^2}{b^2 + 1} = 0 \tag{4-5}$$

Mittels p-q-Formel wird nun der kleinere der beiden Werte für y_{J1} berechnet, da nur dieser laut Definition des Koordinatenursprungs gültig sein kann. Im Anschluss wird z_{J1} mit Gl.(4-4) berechnet.

Der Winkel θ_1 kann jetzt wie folgt bestimmt werden.

$$\theta_1 = arctan\left(\frac{z_{J1}}{y_{F1} - y_{J1}}\right) \tag{4-6}$$

Um die anderen beiden gesuchten Winkel θ_2 und θ_3 zu bestimmen, wird der Punkt E_0 mittels der Matrix $\underline{R}(\varphi)$ transformiert.

$$\vec{E}_0{}' = \underline{R}(\varphi) \cdot \vec{E}_0 = \begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{E}_0, \qquad \varphi = 120°, 240° \tag{4-7}$$

Im Anschluss können die Winkel nach dem gleichen Vorgehen wie bei θ_1 berechnet werden.

Somit können aus gegebenem Punkt E_0 des TCPs, die dazu gehörigen Winkel des Roboters bestimmt werden.

4.1.2 Forward-Kinematik

Die Winkel θ_1, θ_2 und θ_3 sind nun gegeben. Gesucht wird die Position E_0 des TCPs. Die Punkte A_1', A_2' und A_3' können wie folgt berechnet werden.

$$\vec{A_i}' = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} \left(-f + e - rf \cdot cos(\theta_i)\right) \cdot \cos(\varphi_i) \\ \left(-f + e - rf \cdot cos(\theta_i)\right) \cdot \sin(\varphi_i) \\ rf \cdot \sin(\theta_i) \end{pmatrix} \quad \textbf{(4-8)}$$

$$i = 1,2,3\ ;\ \varphi_{1-3} = 0°, 120°, 240°$$

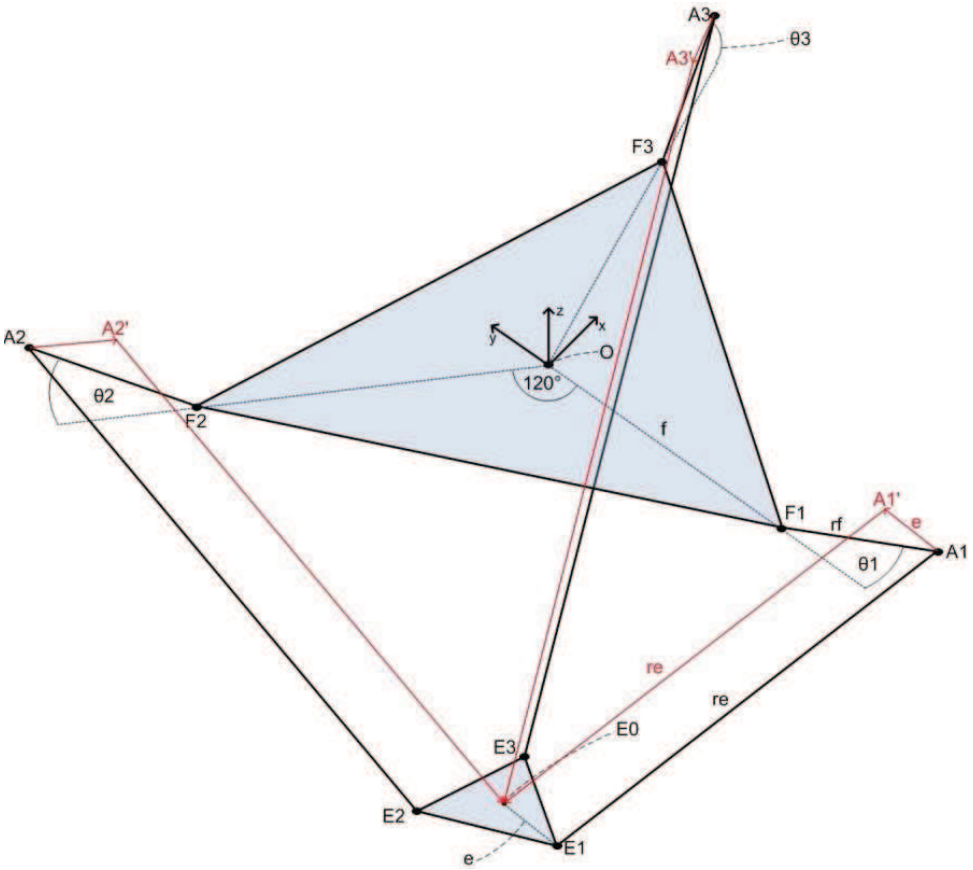


Abbildung 4.1-3:Ansatz der Forward-Kinematik

Werden die Punkte $\vec{A_1}'$, $\vec{A_2}'$ und $\vec{A_3}'$ um die Länge re des Unterarms verlängert, ergeben sich genau zwei Schnittpunkte. Der untere Schnittpunkt ist hierbei die Position des TCPs $\vec{E}_0$. Aus Abbildung 4.1-3 kann jetzt folgende Gleichung aufgestellt werden.

$$re^2 = \left\| \vec{E}_0 \vec{A_i}' \right\|_2^2 = (x_0 - x_i)^2 + (y_0 - y_i)^2 + (z_0 - z_i)^2 \quad \textbf{(4-9)}$$

Daraus folgt:

$$x_0{}^2 + y_0{}^2 + z_0{}^2 - 2y_1y_0 - 2z_1z_0 = re^2 - y_1{}^2 - z_1{}^2 \quad \textbf{(4-10)}$$

$$x_0{}^2 + y_0{}^2 + z_0{}^2 - 2x_2x_0 - 2y_2y_0 - 2z_2z_0 = re^2 - x_2{}^2 - y_2{}^2 - z_2{}^2 \quad \textbf{(4-11)}$$

$$x_0{}^2 + y_0{}^2 + z_0{}^2 - 2x_3x_0 - 2y_3y_0 - 2z_3z_0 = re^2 - x_3{}^2 - y_3{}^2 - z_3{}^2 \quad \textbf{(4-12)}$$

mit

$$w_i = x_i{}^2 + y_i{}^2 + z_i{}^2,\ i = 1,2,3$$

folgt

$$\text{(4-10)-(4-11):}\ x_2x_0 + (y_1 - y_2)y_0 + (z_1 - z_2)z_0 = \frac{(w_1 - w_2)}{2} \quad \textbf{(4-13)}$$

$$\text{(4-10)-(4-12):}\ x_3x_0 + (y_1 - y_3)y_0 + (z_1 - z_3)z_0 = \frac{(w_1 - w_3)}{2} \quad \textbf{(4-14)}$$

$$\text{(4-11)-(4-12):}\ (x_2 - x_3)x_0 + (y_2 - y_3)y_0 + (z_2 - z_3)z_0 = \frac{(w_2 - w_3)}{2} \quad \textbf{(4-15)}$$

Aus Gl.(4-13) und Gl.(4-14) können nun folgende Gleichung hergeleitet werden.

$$x_0 = a_1z_0 + b_1 \quad \textbf{(4-16)}$$

mit

$$a_1 = \frac{1}{d}[(z_2 - z_1)(y_3 - y_1) - (z_3 - z_2)(y_2 - y_1)]$$

$$b_1 = \frac{1}{2d}[(w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1)]$$

$$d = (y_2 - y_1)x_2 - (y_3 - y_1)x_2$$

und

$$y_0 = a_2z_0 + b_2 \quad \textbf{(4-17)}$$

mit

$$a_2 = \frac{-1}{d}[(z_2 - z_1)(x_3) - (z_3 - z_1)(x_2)]$$

$$b_2 = \frac{1}{2d}[(w_2 - w_1)(x_3) - (w_3 - w_1)(x_2)]$$

Durch einsetzten von Gl.(4-16) und Gl.(4-17) in Gl.(4-10) wird anschließend folgende Gleichung aufgestellt werden.

$$z_0{}^2 + p \cdot z_0 + q = 0 \quad \textbf{(4-18)}$$

mit

$$p = \frac{2(a_1 b_1 + a_2(b_2 - y_1) - z_1)}{({a_1}^2 + {a_2}^2 + 1)^2}$$

$$q = \frac{\left({b_1}^2 + (b_2 - y_1)^2 + {z_1}^2 - re^2\right)}{({a_1}^2 + {a_2}^2 + 1)^2}$$

Es wird wieder der kleinere Wert für z_0 ermittelt, da nur dieser laut Definition des Koordinatenursprungs gültig sein kann. Im Anschluss werden x_0 und y_0 mit Gl.(4-16) bzw. Gl.(4-17) bestimmt.

4.2 Dynamisches Modell

Um die Vorsteuerung des Roboters zu realisieren, muss aus den Sollwerten für Geschwindigkeit und Beschleunigung des TCPs, die Winkelgeschwindigkeit und das benötigte Drehmoment berechnet werden. Diese Werte werden anschließend der Vorsteuerung der einzelnen Antriebe vorgegeben.
Zu diesem Zweck wird zunächst die Jacobi-Matrix berechnet. Die Jacobi-Matrix eines Roboterarms beschreibt die Abbildung von Gelenkgeschwindigkeiten auf die Lineargeschwindigkeit des TCPs.

$$\dot{\vec{E}}_0 = \underline{J}(\vec{\theta}) \cdot \dot{\vec{\theta}} \qquad \textbf{(4-19)}$$

mit

$$\vec{\theta} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}, \dot{\vec{\theta}} = \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} \; und \; \dot{\vec{E}}_0 = \begin{pmatrix} \dot{x}_0 \\ \dot{y}_0 \\ \dot{z}_0 \end{pmatrix}$$

Im Anschluss wird durch Ableiten der Jacobi-Matrix der Zusammenhang zwischen Gelenkbeschleunigung und der Linearbeschleunigung hergestellt. Dies ist notwendig um die auftretenden Drehmomente zu berechnen.

$$\ddot{\vec{E}}_0 = \underline{\dot{J}}\left(\vec{\theta}, \dot{\vec{\theta}}\right) \cdot \dot{\vec{\theta}} + \underline{J}(\vec{\theta}) \cdot \ddot{\vec{\theta}} \qquad \textbf{(4-20)}$$

mit

$$\ddot{\vec{\theta}} = \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{pmatrix} \; und \; \ddot{\vec{E}}_0 = \begin{pmatrix} \ddot{x}_0 \\ \ddot{y}_0 \\ \ddot{z}_0 \end{pmatrix}$$

Die folgenden Abschnitte erläutern schrittweise den Ansatz und die Berechnung der benötigten Geschwindigkeiten, Beschleunigungen sowie der Drehmomente. Der Ansatz bezieht sich auf die Veröffentlichung von Alain Codourey. [6]

4.2.1 Geschwindigkeitskinematik

Zur Bestimmung der Jacobi-Matrix wird der gleiche Ansatz wie in Abschnitt 4.1.2 verwendet. Der Vektor ${E_0A_i}'$ aus Gl.(4-9) wird als s_i definiert und folgendermaßen bestimmt.

$$\vec{s}_i = \vec{O}\vec{E}_0 - \left(\vec{O}\vec{F}_i + \vec{F}_i{\vec{A}_i}'\right) = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} - \underline{R}(\varphi_i)\left(\begin{bmatrix} 0 \\ -f \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ e - rf\cos(\theta_i) \\ rf\sin(\theta_i) \end{bmatrix}\right) \quad \textbf{(4-21)}$$

$$i = 1{,}2{,}3\ ;\ \varphi_{1-3} = 0°, 120°, 240°$$

Da rf immer konstant ist gilt:

$$\vec{s}_i{}^T\vec{s}_i - rf^2 = 0 \quad i = 1{,}2{,}3 \quad \textbf{(4-22)}$$

Nach Ableiten von Gl.(4-22) folgt:

$$\vec{s}_i{}^T\dot{\vec{s}}_i + \dot{\vec{s}}_i{}^T\vec{s}_i = 0 \quad i = 1{,}2{,}3 \quad \textbf{(4-23)}$$

Gl.(4-23) kann nach dem Kommutativgesetz vereinfacht werden.

$$\vec{s}_i{}^T\dot{\vec{s}}_i = 0 \quad i = 1{,}2{,}3 \quad \textbf{(4-24)}$$

Wobei für die Ableitung von s_i gilt:

$$\dot{\vec{s}}_i = \dot{\vec{E}}_0 - \vec{b}_i\dot{\vec{\theta}}_i \quad i = 1{,}2{,}3 \quad \textbf{(4-25)}$$

mit

$$\vec{b}_i = \underline{R}(\varphi_i)\begin{pmatrix} 0 \\ rf\sin(\theta_i) \\ rf\cos(\theta_i) \end{pmatrix} \quad i = 1{,}2{,}3\ ;\ \varphi_{1-3} = 0°, 120°, 240°$$

Somit kann Gl.(4-24) für einen Roboterarm wie folgt aufgestellt werden.

$$\vec{s}_i{}^T\dot{\vec{E}}_0 - \vec{s}_i{}^T\vec{b}_i\dot{\vec{\theta}}_i = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad i = 1{,}2{,}3 \quad \textbf{(4-26)}$$

Daraus folgt für alle drei Roboterarme.

$$\begin{pmatrix} \vec{s}_1{}^T \\ \vec{s}_2{}^T \\ \vec{s}_3{}^T \end{pmatrix}\dot{\vec{E}}_0 - \begin{pmatrix} \vec{s}_1{}^T\vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2{}^T\vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3{}^T\vec{b}_3 \end{pmatrix}\dot{\vec{\theta}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \textbf{(4-27)}$$

In Anlehnung an Gl.(4-19) kann die Jacobi-Matrix $J(\theta)$ jetzt folgendermaßen beschrieben werden.

$$\underline{J}(\theta) = \begin{pmatrix} \vec{s}_1{}^T \\ \vec{s}_2{}^T \\ \vec{s}_3{}^T \end{pmatrix}^{-1} \begin{pmatrix} \vec{s}_1{}^T\vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2{}^T\vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3{}^T\vec{b}_3 \end{pmatrix} \tag{4-28}$$

Da bei der in dieser Masterthesis verwendeten Anwendung jedoch nicht die Lineargeschwindigkeit in Abhängigkeit der Winkelgeschwindigkeit ermittelt werden muss, sondern genau anders herum, muss zum Abschluss die Inverse der Jacobi-Matrix bestimmt werden.

$$\dot{\vec{\theta}} = \underline{J}(\theta)^{-1} \cdot \dot{\vec{E}}_0 \tag{4-29}$$

4.2.2 Beschleunigungskinematik

Um die Beschleunigungskinematik, also den Zusammenhang von Linearbeschleunigung des TCPs zur Winkelbeschleunigung der Roboterarme, zu beschreiben, kann der Ansatz aus dem oberen Abschnitt fortgeführt werden. Die folgende Gleichung zeigt die Ableitung von Gl.(4-27).

$$\begin{pmatrix} \vec{s}_1{}^T \\ \vec{s}_2{}^T \\ \vec{s}_3{}^T \end{pmatrix} \ddot{\vec{E}}_0 + \begin{pmatrix} \dot{\vec{s}}_1{}^T \\ \dot{\vec{s}}_2{}^T \\ \dot{\vec{s}}_3{}^T \end{pmatrix} \dot{\vec{E}}_0 \tag{4-30}$$

$$-\left(\begin{bmatrix} \vec{s}_1{}^T\vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2{}^T\vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3{}^T\vec{b}_3 \end{bmatrix} \ddot{\vec{\theta}} + \begin{bmatrix} \dot{\vec{s}}_1{}^T\vec{b}_1 + \vec{s}_1{}^T\dot{\vec{b}}_1 & 0 & 0 \\ 0 & \dot{\vec{s}}_2{}^T\vec{b}_2 + \vec{s}_2{}^T\dot{\vec{b}}_2 & 0 \\ 0 & 0 & \dot{\vec{s}}_3{}^T\vec{b}_3 + \vec{s}_3{}^T\dot{\vec{b}}_3 \end{bmatrix} \dot{\vec{\theta}} \right) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

mit

$$\dot{\vec{b}}_i = \underline{R}(\varphi_i) \begin{pmatrix} 0 \\ rf\cos(\theta_i) \\ -rf\sin(\theta_i) \end{pmatrix} \dot{\theta}_i \quad i = 1,2,3\ ;\ \varphi_{1-3} = 0°, 120°, 240°$$

Nach dem umstellen von Gl.(4-30) nach Gl.(4-20) ergibt sich.

$$\ddot{\vec{E}}_0 = \begin{pmatrix} \vec{s}_1{}^T \\ \vec{s}_2{}^T \\ \vec{s}_3{}^T \end{pmatrix}^{-1} \left(-\begin{bmatrix} \dot{\vec{s}}_1{}^T \\ \dot{\vec{s}}_2{}^T \\ \dot{\vec{s}}_3{}^T \end{bmatrix} \underline{J}(\vec{\theta}) + \begin{bmatrix} \dot{\vec{s}}_1{}^T\vec{b}_1 + \vec{s}_1{}^T\dot{\vec{b}}_1 & 0 & 0 \\ 0 & \dot{\vec{s}}_2{}^T\vec{b}_2 + \vec{s}_2{}^T\dot{\vec{b}}_2 & 0 \\ 0 & 0 & \dot{\vec{s}}_3{}^T\vec{b}_3 + \vec{s}_3{}^T\dot{\vec{b}}_3 \end{bmatrix} \right) \dot{\vec{\theta}} + \underline{J}(\vec{\theta}) \cdot \ddot{\vec{\theta}} \tag{4-31}$$

Die Ableitung der Jacobi-Matrix, auch Hesse-Matrix genannt, kann nun wie folgt bestimmt werden.

$$\underline{\dot{J}}\left(\vec{\theta},\dot{\vec{\theta}}\right)=\begin{pmatrix}\vec{s}_1{}^T\\ \vec{s}_2{}^T\\ \vec{s}_3{}^T\end{pmatrix}^{-1}\left(-\begin{bmatrix}\dot{\vec{s}}_1{}^T\\ \dot{\vec{s}}_2{}^T\\ \dot{\vec{s}}_3{}^T\end{bmatrix}\underline{J}(\vec{\theta})+\begin{bmatrix}\dot{\vec{s}}_1{}^T\vec{b}_1+\vec{s}_1{}^T\dot{\vec{b}}_1 & 0 & 0\\ 0 & \dot{\vec{s}}_2{}^T\vec{b}_2+\vec{s}_2{}^T\dot{\vec{b}}_2 & 0\\ 0 & 0 & \dot{\vec{s}}_3{}^T\vec{b}_3+\vec{s}_3{}^T\dot{\vec{b}}_3\end{bmatrix}\right) \quad \textbf{(4-32)}$$

Um anschließend die benötigten Winkelbeschleunigungen berechnen zu können wird Gl.(4-20) nach $\ddot{\vec{\theta}}$ umgestellt.

$$\ddot{\vec{\theta}}=\underline{J}(\vec{\theta})^{-1}\left[\ddot{\vec{E}}_0-\underline{\dot{J}}\left(\vec{\theta},\dot{\vec{\theta}}\right)\cdot\dot{\vec{\theta}}\right] \quad \textbf{(4-33)}$$

4.2.3 Berechnung der Drehmomente

In diesem Abschnitt wird die Berechnung der auftretenden Drehmomente, die an der Drehachse zwischen den Getrieben und den Oberarmen des Roboters auftreten, erläutert. Dazu müssen die Trägheitsmomente bestimmt werden, die in Abhängigkeit der Position des TCPs vorliegen. Anschließend werden die einzelnen Kräfte berechnet, die auf die Drehachse wirken und in der Summe die benötigte Kraft ergeben.

Die Modellierung des dynamischen Modells beruht auf dem Newton-Euler-Verfahren. Folgende Annahmen wurden aufgestellt:

- Die Anfangswerte für $\vec{\theta}$, $\dot{\vec{\theta}}$ und $\ddot{\vec{\theta}}$ bzw. $\vec{E}_0$, $\dot{\vec{E}}_0$ und $\ddot{\vec{E}}_0$ sind bekannt.
- Die Masse des Unterarms wird zur Vereinfachung auf die Kugelgelenke an seinen Enden aufgeteilt. D.h. $^2/_3$ werden dem Verbindungspunkt mit dem Oberarm $(\vec{A}_i)$ zugewiesen. Das restliche Drittel wird dem Verbindungspunkt mit der Arbeitsplatte $(\vec{E}_i)$ zugeordnet. Dies beruht auf der Tatsache, dass das Trägheitsmoment eines dünnen Stabes mit der Masse *m* und der Länge *L*, der um eine Querachse an einem Ende rotiert, nach der Steiner-Regel ,wie folgt beschrieben wird: $I=\,^1/_3\,mL^2$
- Reibungsverluste, das Trägheitsmoment der Getriebe und Antriebe sowie die Elastizität der mechanischen Konstruktion werden vernachlässigt.

Grund für die Vernachlässigung der Trägheitsmomente der Getriebe und Antriebe ist, dass diese bereits in den Umrichter internen Vorsteuerungen berücksichtigt werden.

4.2.3.1 Definition der Parameter

In diesem Unterabschnitt werden die Parameter definiert, die zur Berechnung der Drehmomente notwendig sind.
Zuerst wird die Ebene der unteren Arbeitsplatte betrachtet.

$$m_{et} = m_e + m_{payload} + 3(1-r)m_{re} \quad r = {}^{2}/_{3} \tag{4-34}$$

Die Masse m_{et} ist die Summe aus der Masse der Arbeitsplatte m_e, der bewegten Werkstücke $m_{payload}$ und der den drei Gelenkpunkten der Arbeitsplatte zugeordneten Massen der Unterarme.
Auf der Ebene der Oberarme wird zuerst der Massenschwerpunkt wie folgt ausgerechnet.

$$r_{Ga} = rf\frac{\frac{1}{2}m_{rf} + m_a + r \cdot m_{re}}{m_b} \quad r = {}^{2}/_{3} \tag{4-35}$$

mit

$$m_b = m_{rf} + m_a + r \cdot m_{re} \quad r = {}^{2}/_{3}$$

m_{rf} ist die Masse der Oberarme, m_a die Masse des Kugelgelenkes und $r \cdot m_{re}$ der Teil der Unterarmmasse die dem Oberarm angerechnet wird.
Als letztes wird das Trägheitsmoment der Oberarme bezogen auf die Drehachse berechnet. Dazu kommt wieder die Steiner-Regel zum Einsatz.

$$I_{rfi} = {}^{1}/_{3}\, m_{rf} rf^2 + (m_a + r \cdot m_{re}) rf^2 = rf^2\left(\frac{m_{rf}}{3} + m_a + r \cdot m_{re}\right) \quad i = 1,2,3 \tag{4-36}$$

4.2.3.2 Drehmomentgleichung

Das auf die Drehachsen wirkende Drehmoment setzt sich aus verschiedenen Kräften zusammen. Dazu gehören die Anziehungskraft, Corioliskraft, Zentrifugalkraft und die benötigte Kraft, um die Arbeitsplatte zu beschleunigen. Mittels der oben beschriebenen Jacobi-Matrix $\underline{J}(\vec{\theta})$ und dessen Ableitung $\underline{\dot{J}}\left(\vec{\theta}, \dot{\vec{\theta}}\right)$, können diese Kräfte in Drehmomente umgerechnet werden. Sie ergeben in der Summe das gesamte Drehmoment.
Zwei Kräfte wirken auf die Arbeitsplatte. Zum einen die Anziehungskraft $\vec{G}_e$.

$$\vec{G}_e = m_{et}\begin{pmatrix}0\\0\\-g\end{pmatrix} \tag{4-37}$$

Zum anderen die Beschleunigungskraft $\vec{F}_e$.

$$\vec{F}_e = m_{et}\ddot{\vec{E}}_0 \tag{4-38}$$

Unter Verwendung der transponierten Jacobi-Matrix können diese beiden translatorischen in rotatorische Kräfte, also Drehmomente, umgerechnet werden.

$$\vec{M}_{Ge} = \underline{J}(\vec{\theta})^T \vec{G}_e = \underline{J}(\vec{\theta})^T m_{et}\begin{pmatrix}0\\0\\-g\end{pmatrix} \tag{4-39}$$

und

$$\vec{M}_e = \underline{J}(\vec{\theta})^T \vec{F}_e = \underline{J}(\vec{\theta})^T m_{et}\ddot{\vec{E}}_0 \tag{4-40}$$

Die auf die Oberarme wirkenden Drehmomente können wie folgt beschrieben werden.

$$\vec{M}_{Grf} = m_b r_{Ga}(-g)\begin{pmatrix}\cos(\theta_1)\\\cos(\theta_2)\\\cos(\theta_3)\end{pmatrix} \tag{4-41}$$

und

$$\vec{M}_{rf} = \underline{I_{rf}}\ddot{\vec{\theta}} \tag{4-42}$$

mit der Trägheitsmatrix

$$\underline{I_{rf}} = \begin{pmatrix}I_{rf1} & 0 & 0\\0 & I_{rf2} & 0\\0 & 0 & I_{rf3}\end{pmatrix}$$

Da nach dem D'Alembert-Prinzip die Summe aller inneren Kräfte gleich der Summe der äußeren Kräfte sein muss, kann folgende Gleichung aufgestellt werden.

$$\vec{M}_\theta + \vec{M}_{Ge} + \vec{M}_{Grf} = \vec{M}_e + \vec{M}_{rf} \tag{4-43}$$

Das von den Roboterarmen erbrachte Drehmoment ist dabei $\vec{M}_\theta$. Aus Gl.(4-43) und Gl.(4-20) folgt.

$$\vec{M}_\theta = \underline{A}(\vec{\theta})\ddot{\vec{\theta}} + \underline{C}\left(\vec{\theta},\dot{\vec{\theta}}\right)\dot{\vec{\theta}} - \vec{G}(\vec{\theta}) \tag{4-44}$$

Mit der Gewichts-Matrix:

$$\underline{A}(\vec{\theta}) = \underline{I_{rf}} + m_{et}\underline{J}(\vec{\theta})^T \underline{J}(\vec{\theta}) \quad \textbf{(4-45)}$$

Der Matrix der Coriolis- und Zentrifugalkräfte:

$$\underline{C}\left(\vec{\theta}, \dot{\vec{\theta}}\right) = m_{et}\underline{J}(\vec{\theta})^T \underline{\dot{J}}\left(\vec{\theta}, \dot{\vec{\theta}}\right) \quad \textbf{(4-46)}$$

Und dem Vektor der Gravitationskräfte:

$$\vec{G}(\vec{\theta}) = \vec{M}_{Ge} + \vec{M}_{Grf} \quad \textbf{(4-47)}$$

Da die Antriebe nicht direkt mit den Oberarmen des Roboters verbunden sind, sondern über Getriebe, müssen diese bei der Berechnung von Drehzahl und Drehmoment berücksichtigt werden.

$$\vec{M}_m = \frac{1}{\ddot{u}} \vec{M}_\theta [Nm] \quad \textbf{(4-48)}$$

$$\vec{n}_m = \frac{60}{2\pi} \cdot \ddot{u} \cdot \dot{\vec{\theta}} \left[\frac{1}{min}\right] \quad \textbf{(4-49)}$$

Somit können mit Hilfe von Gl.(4-44), bei bekannter translatorischer Bewegung und der Inversen-Kinematik aus Abschnitt 4.1.1, die benötigte Drehzahl und die Drehmomente der Antriebe berechnet werden.

4.3 Modell der Synchronantriebe

In diesem Abschnitt werden der Aufbau des elektromechanischen Modells sowie die Regelung der Synchronmaschinen erläutert. Dabei wird nicht auf die Modellierung der Antriebe eingegangen, da dies den Zeitrahmen der Thesis überschreiten würde. Das Modell der Regelung der Anriebe soll lediglich den Zweck erfüllen, das Gesamtsystem des Roboters zu simulieren und den Nutzen einer Vorsteuerung für das Antriebssystem des Roboters aufzuzeigen.
Moderne Servoantriebe sind, wie auch hier, vollständig mikroprozessorgesteuert und kaskadiert geregelt. Der Kaskadenregler wird mit Hilfe der Inbetriebnahme-Software STARTER ausgelegt. Die Kaskadenregelung ist dort mit einer feldorientierten Regelung realisiert. Zur Modellierung des Antriebssystems sind folgende Parameter notwendig.

Motordaten:	
$J = 2{,}8 \cdot 10^{-5} Kgm^2$	*Rotationsträgheitsmoment*
$R = 4{,}2\Omega$	*Wicklungswiderstand*
$L = 9{,}1mH$	*Wicklungsinduktivität*
$T_{el} = \frac{L}{R} = 2{,}2ms$	*Elektrische Zeitkonstante*
$K_M = 0{,}46\,{}^{Nm}/_{A}$	*Drehmomentkonstante*
$K_U = 0{,}277\,{}^{Vs}/_{rad}$	*Spannungskonstante*
$M_R(\omega)$	*Kennlinie des Reibmomentes des Motors*
M_L	*Das auf den Motor wirkende Lastmoment*
Leistungsteil:	
$T_{PWM} = 62{,}5\mu s$	*Stromrichterzeitkonstante*
$T_{abt} = 125\mu s$	*Abtastzeit*
$T_{tot} = T_{PWM} + 2T_{abt}$	*Gesamttotzeit*
$U_Z = 300V$	*Zwischenkreisspannung*
$I_{max} = 4{,}6A$	*Maximalstrombegrenzung*
Regelung:	
K_{pi}	*Proportionalverstärkung Stromregler*
T_{ni}	*Nachstellzeit Stromregler*
K_{pn}	*Proportionalverstärkung Drehzahlregler*
T_{nn}	*Nachstellzeit Drehzahlregler*
K_v	*Proportionalverstärkung Positionsregler*

Tabelle 4:Parameter des Antriebsystems

Das Blockschaltbild der Synchronmaschine sieht wie folgt aus.

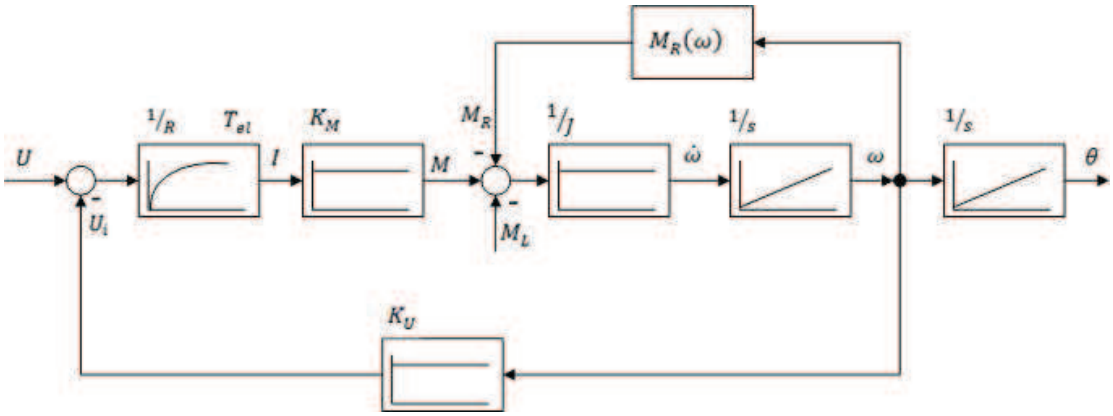


Abbildung 4.3-1:Blockschaltbild der Synchronmaschine

Dabei ist das elektrische Verhalten der Maschine durch ein PT-1-Glied angenähert, dass das Verhältnis von Eingangsspannung zum Iststrom realisiert. Das Verhältnis von Strom zu Drehmoment wird durch ein P-Glied mit der Verstärkung K_M dargestellt. Dem von der Maschine erzeugten Drehmoment wirken das innere Reibmoment M_R des Antriebs sowie das äußere Lastmoment M_L entgegen. Durch Multiplikation des Gesamtdrehmomentes mit der Inversen des Rotationsträgheitsmomentes J^{-1} ergibt die Winkelbeschleunigung $\dot{\omega}$. Die sich aus der Drehzahl ω ergebene induzierte Gegenspannung U_i, wirkt der Eingangsspannung entgegen. Hier realisiert durch ein P-Glied mit der Verstärkung K_U.

Der Stromregelkreis besteht im Wesentlichen aus einem PI-Regler und der durch den Leistungsteil hervorgerufenen Totzeit. Die Stellgröße des Stromreglers ist hierbei auf die Zwischenkreisspannung U_Z des Umrichters begrenzt. Da der Sollwert als Drehmoment vorgegeben wird, muss dieser durch die Drehmomentkonstante K_M geteilt und anschließend auf I_{max} begrenzt werden. Um die durch die Rotation der Maschine induzierte Gegenspannung U_i zu kompensieren, wird sie auf die Stellgröße U addiert.
Der Drehzahlregelkreis ist dem Stromregelkreis in der Kaskadenstruktur übergeordnet und ebenfalls als PI-Regler realisiert. Die Stellgröße ist das erforderliche Drehmoment zur Beschleunigung der Maschine.
Um die Drehachsen des Roboters zu positionieren, ist zusätzlich ein übergeordneter Lageregelkreis vorhanden. Der Lageregler ist als P-Regler realisiert.

Abbildung 4.3-2 stellt die Kaskadenregelung der Synchronmaschine dar.

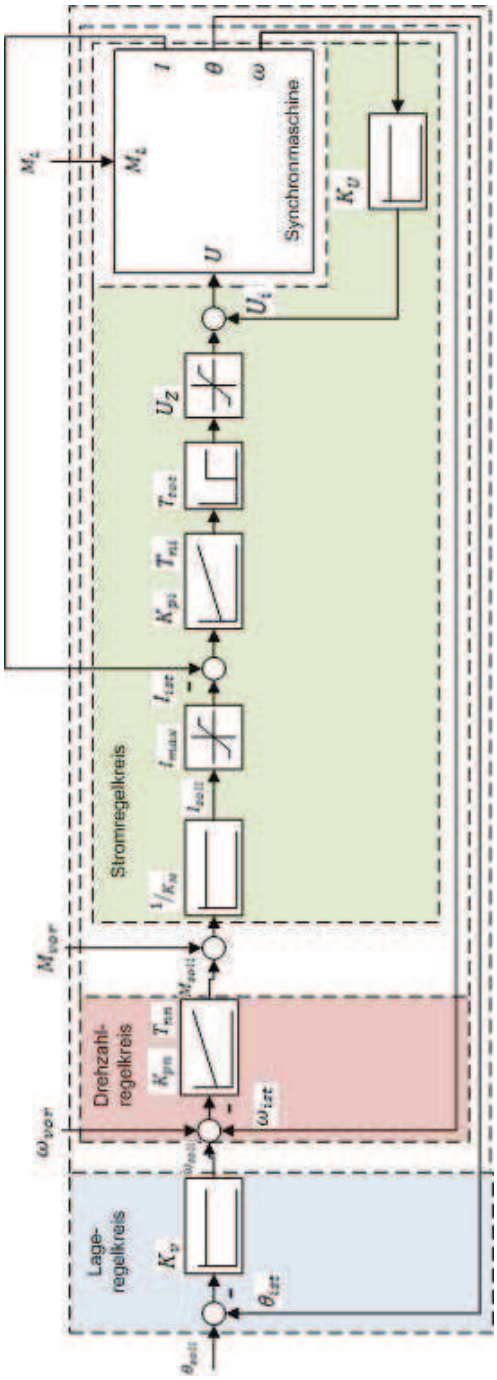


Abbildung 4.3-2:Lageregelkreis der Synchronmaschine

5 Bahninterpolation

In diesem Kapitel wird erläutert, wie sich der Roboter auf einer Kreisbahn oder zwischen zwei Zielstellungen bewegen soll. Dies beinhaltet die Berechnung von Zwischenstellungen auf Grundlage von Geschwindigkeit- und oder Beschleunigungsangaben. Die Berechnung von Bahnzwischenpunkten nach einer festen Rechenregel nennt man Interpolation. Grundlage für die Vorbetrachtung und des Siniodenprofils zur Interpolation ist das Buch von Herrn Wolfgang Weber. [10]

5.1 Vorbetrachtung

Die einfachste Art der Bewegung zwischen Start- und Zielposition ist die PTP[21]-Bahn. Bei dieser Bewegungsart werden lediglich die Armwinkel eines Roboters interpoliert, so dass sich der TCP auf einer für den Anwender nicht vorhersehbaren Bahn bewegt. Man unterscheidet zwischen der asynchronen und synchronen PTP-Bahn. Bei der asynchronen PTP-Bahn verfährt jeder Arm des Roboters vollkommen unabhängig von den anderen zu seiner Zielstellung. Dies bedeutet, dass die Armwinkel zu verschiedenen Zeitpunkten zum Stillstand kommen. Im Gegensatz dazu wird bei der synchronen PTP-Bahn die Geschwindigkeit der Drehachse proportional zur zu bewältigenden Winkeländerung vorgegeben. Somit ist eine Vorgabe der Zeitdauer der Bewegung möglich. Außerdem erreichen alle Armwinkel gleichzeitig ihre Zielposition.
Ein weiterer Ansatz, der hier in Betracht gezogen wurde ist die Vorgabe von Zwischenpunkten. Dabei wird dem Roboter nicht nur die zu erreichende Endposition vorgegeben, sondern zusätzlich eine Reihe von Zwischenhalten. Somit kann zum Teil verhindert werden, dass sich der TCP auf einer für den Anwender nicht vorhersehbaren Bahn bewegt.
Ein dritter Ansatz ist die Linearbahn. Dabei wird die Zielstellung auf einer Geraden angefahren, also auf dem kürzesten Weg zwischen Start- und Zielposition. Die Vorteile der Linearbahn sind, dass die gesamte zu fahrende Stecke bekannt ist und eine Vorgabe für Geschwindigkeit und Beschleunigung möglich ist. Außerdem lässt sich bei dieser Interpolation eine Vorsteuerung für Geschwindigkeit und Drehmoment recht einfach realisieren. Diese Vorteile bedeuten jedoch auch einen größeren Rechenaufwand der Steuerung.

Zum besseren Verständnis sind in Abbildung 5.1-1 die unterschiedlichen Bewegungsarten am Beispiel eines YZ-Graphen dargestellt.

[21] Point to Point

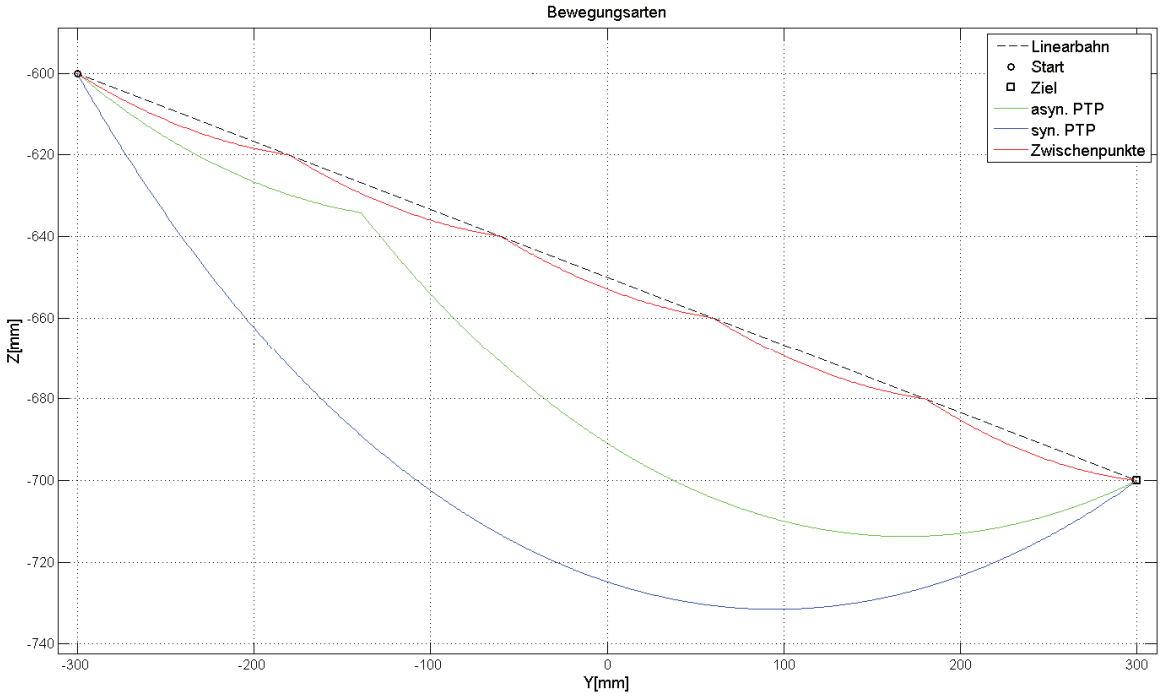


Abbildung 5.1-1:Bewegungsarten zwischen Start- und Zielposition

Da bei den hohen Geschwindigkeiten, die der Roboter fahren soll, eine Kollision mit Hindernissen oder dem Boden unbedingt verhindert werden muss, wurde die Interpolation nach dem Ansatz der Linearbahn ausgewählt. Die genaue Berechnung wird im folgenden Abschnitt erläutert.

5.2 Sinoidenprofil zur Interpolation

Um eine sprungförmige Beschleunigung des TCPs zu vermeiden und somit die Mechanik des Roboters zu schonen, wurde das Sinoidenprofil zur Interpolation ausgewählt. Bei dieser Interpolationsart, wird eine weiche Sollbewegung des TCPs vorgegeben. Das Verfahren des Roboters wird hierbei in zwei Zeitabschnitte unterteilt, in denen die Beschleunigung des TCPs $a(t)$ sowie dessen Integrale für Geschwindigkeit und Strecke unterschiedlich berechnet werden. Neben der Vorgabe der Zielposition wird hier auch die Maximalgeschwindigkeit v_m vorgegeben und die Maximalbeschleunigung b_m berechnet.

$$b_m = \frac{2 \cdot {v_m}^2}{s_e} \qquad \textbf{(5-1)}$$

mit

$b_m : Maximalbeschleunigung \left[{m}/{s^2}\right]$

$v_m : Maximalgeschwindigkeit \left[{m}/{s}\right]$

$s_e : gegebene\ Bahnlänge\ [m] = |s_{Start} - s_{Ziel}|$

Die Beschleunigungszeit t_b und die Gesamtfahrtzeit t_v werden folgendermaßen berechnet.

$$t_b = \frac{2 \cdot v_m}{b_m}, t_e = \frac{s_e}{v_m} + t_b = 2 \cdot t_b \quad \textbf{(5-2)}$$

Die Zeitfunktion der Beschleunigung für $0 \leq t \leq t_b$ wird wie folgt berechnet.

$$a(t) = b_m \cdot \sin^2\left(\frac{\pi}{t_b} t\right) \quad \textbf{(5-3)}$$

Durch Integration von Gl.(5-3) ergeben sich die Zeitfunktionen für Geschwindigkeit $v(t)[{}^{m}/_{s}]$ und Strecke s$(t)[m]$:

$$v(t) = b_m \left(\frac{1}{2} \cdot - \frac{t_b}{4\pi} \cdot \sin\left(\frac{2\pi}{t_b} t\right)\right) \quad \textbf{(5-4)}$$

$$s(t) = s(0) + b_m \left(\frac{1}{4} \cdot t^2 + \frac{{t_b}^2}{8\pi^2} \cdot \left(\cos\left(\frac{2\pi}{t_b} t\right) - 1\right)\right) \quad \textbf{(5-5)}$$

Auf das Beschleunigen des TCPs folgt das Abbremsen für $t_b \leq t \leq t_e$.

$$a(t) = -b_m \cdot \sin^2\left(\frac{\pi}{t_b}(t - t_b)\right) \quad \textbf{(5-6)}$$

Daraus folgt für $v(t)$ und $s(t)$:

$$v(t) = v_m - b_m \left(\frac{1}{2} \cdot - \frac{t_b}{4\pi} \cdot \sin\left(\frac{2\pi}{t_b}(t - t_b)\right)\right) \quad \textbf{(5-7)}$$

$$s(t) = s(t_b) + \frac{b_m}{2}\left[t_e \cdot (t + t_b) - \frac{(t^2 + {t_e}^2 + 2 \cdot {t_b}^2)}{2} + \frac{{t_b}^2}{4\pi^2}\left(1 - \cos\left(\frac{2\pi}{t_b}(t - t_b)\right)\right)\right] \quad \textbf{(5-8)}$$

Da nicht nur eine Translationsachse sondern drei vorgegeben werden müssen, muss zunächst geprüft werden, welche Achse den längsten Weg zurücklegen muss. Dieser Achse wird die Maximalgeschwindigkeit vorgegeben, den anderen eine Geschwindigkeit im Verhältnis zu ihrer Wegstrecke. Somit wird die Linearbahn des TCPs auch im dreidimensionalen Raum sichergestellt.

Im Anschluss kann die Interpolation für die Translationsachsen mittels der Berechnungen aus Kapitel 4, in Soll-Winkel, Winkelgeschwindigkeit und Drehmoment für die Vorsteuerung der Antriebe umgerechnet werden.

Abbildung 5.2-1 zeigt die Interpolation für eine Maximalgeschwindigkeit von $v_m = 1\,{}^{m}/_{s}$.

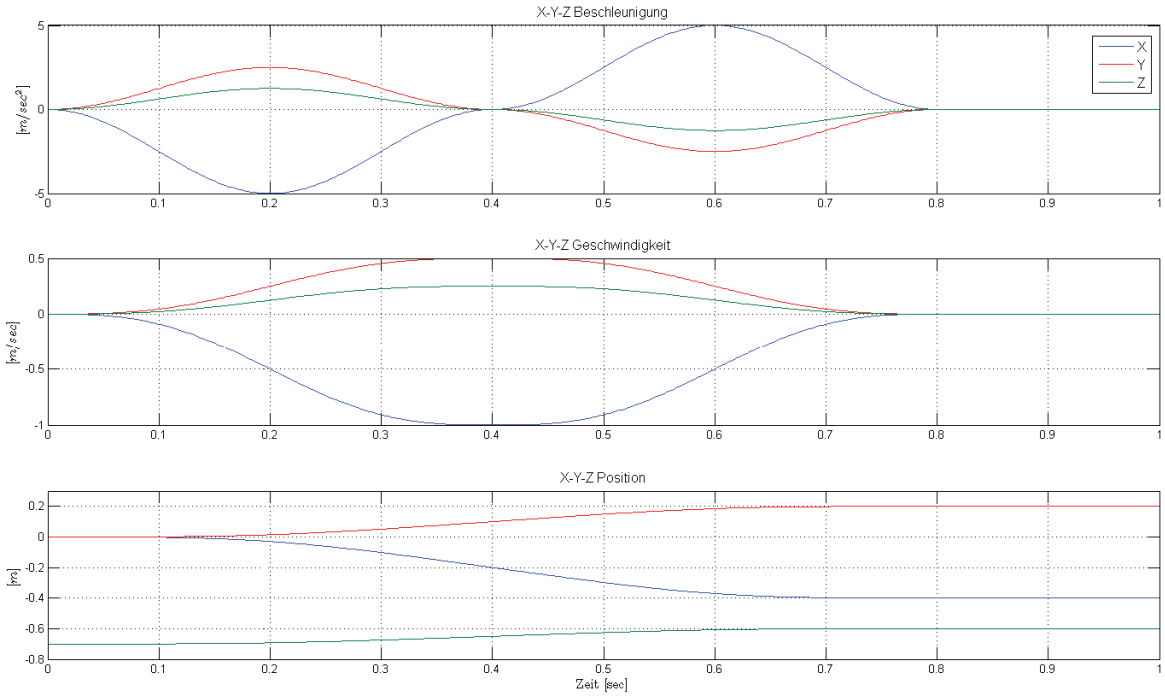


Abbildung 5.2-1:Bahninterpolation Ist(0,0,-0.7) nach Soll(-0.4,0.2,-0.6)

5.3 Interpolation einer Kreisbahn

Zu Demonstrationszwecken wurde neben der Interpolation einer Linearbahn auch noch die Bewegung auf einer Kreisbahn mit konstanter Höhe vorbereitet. Dabei übernimmt der Roboter seinen jetzigen Abstand des TCPs vom Nullpunkt der X- und Y-Achse als Radius und fährt mit vorgegebener Geschwindigkeit auf der Anfangshöhe eine Kreisbahn.
Die Geschwindigkeit wird in Umdrehungen pro Sekunde vorgegeben. Für die Strecke $s(t)$ ergibt sich folgende Zeitfunktion.

$$\vec{s}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right) & \sin\left(\frac{2\pi \cdot t}{p^{-1}}\right) & 0 \\ -\sin\left(\frac{2\pi \cdot t}{p^{-1}}\right) & \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{s}(0) \qquad \textbf{(5-9)}$$

mit

p : $vorgegebene\ Geschwindigkeit\ \left[{}^{1}/_{s}\right]$

$\vec{s}(t)$: $Koordinaten\ des\ TCP's\ zum\ Zeitpunkt\ t\ [m]$

$\vec{s}(0)$: $Koordinaten\ des\ TCP's\ zum\ Zeitpunkt\ t = 0\ [m]$

Durch Ableiten von Gl.(5-9) werden die entsprechenden Geschwindigkeiten $\vec{v}(t)[^m/_s]$ ermittelt.

$$\vec{v}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix} = \begin{pmatrix} \frac{-2\pi \cdot \sin\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-1}} & \frac{2\pi \cdot \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-1}} & 0 \\ \frac{-2\pi \cdot \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-1}} & \frac{-2\pi \cdot \sin\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-1}} & 0 \\ 0 & 0 & 0 \end{pmatrix} \vec{s}(0) \qquad \textbf{(5-10)}$$

Anschließend werden die Beschleunigungen $\vec{a}(t)\left[^m/_{s^2}\right]$ wie folgt berechnet.

$$\vec{a}(t) = \begin{pmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ \ddot{z}(t) \end{pmatrix} = \begin{pmatrix} \frac{-4\pi^2 \cdot \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-2}} & \frac{-4\pi^2 \cdot \sin\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-2}} & 0 \\ \frac{4\pi^2 \cdot \sin\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-2}} & \frac{-4\pi^2 \cdot \cos\left(\frac{2\pi \cdot t}{p^{-1}}\right)}{p^{-2}} & 0 \\ 0 & 0 & 0 \end{pmatrix} \vec{s}(0) \qquad \textbf{(5-11)}$$

Abbildung 5.3-1 zeigt das Ergebnis der Interpolation einer Kreisbahn mit einer Umdrehung pro Sekunde.

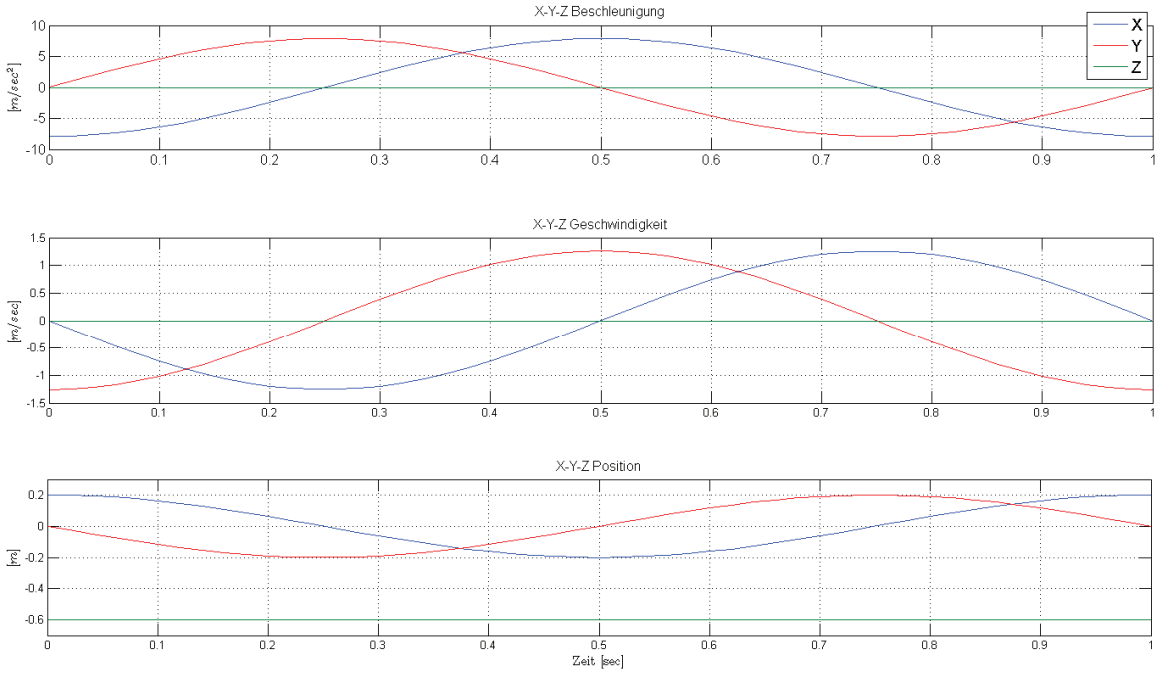


Abbildung 5.3-1:Kreisbahn-Interpolation Start(0.2,0,-0.6)

Wie auch bei der Linearbahn kann im Anschluss die Interpolation für die Translationsachsen mittels der Berechnungen aus Kapitel 4 in Soll-Winkel, Winkelgeschwindigkeit und Drehmoment für die Vorsteuerung der Antriebe umgerechnet werden.

6 Projektierung mittels STARTER

Die Software STARTER der Firma Siemens ist eine grafische Projektierungsoberfläche zur Inbetriebnahme von Antriebssystemen der SINAMICS und MICROMASTER Baureihe. Mit der Software lassen sich alle für diese Automatisierungsaufgabe benötigten Einstellungen der Umrichter vornehmen. Dazu kann einerseits die grafische Oberfläche der einzelnen Funktionen genutzt werden. Andererseits gibt es die Möglichkeit, spezielle Einstellungen in einer Expertenliste vorzunehmen. Folgende Funktionen des Tools wurden genutzt:

- Anlegen von Kommunikationsschnittstellen
- Vermessen des Antriebes
- Automatische Reglereinstellungen
- Messfunktionen
- Einstellung der Winkelerfassung
- Expertenliste
- Aufschalten von Vorsteuerungen für Drehzahl und Drehmoment
- Diagnose der Steuer- und Zustandswörter

Da STARTER ein sehr umfangreiches Werkzeug ist, war zunächst eine gewisse Einarbeitungszeit notwendig, um effektiv arbeiten zu können. Das hier erstellte STARTER-Projekt besteht aus drei gleichen Einzelantriebsgeräten, für jeweils einen Umrichter und Antrieb. Sie unterscheiden sich lediglich in ihren IP-Adressen und den mittels STARTER vorgenommenen automatischen Reglereinstellungen für Strom- und Drehzahlregelkreis. Für jedes Antriebgerät werden zunächst die vorhandenen Hardwarekomponenten, bestehend aus dem verwendeten Umrichter, Control-Unit, Antrieb und dem Inkrementalwertgeber, ausgewählt. Zudem wird die Zusatzfunktion Lageregelung freigeschaltet, um die Einstellungen der Mechanik vornehmen zu können.
In diesem Kapitel werden nun die wichtigsten Einstellungen, die in STARTER gemacht wurden erläutert.

6.1 Kommunikation

Zur Projektierung mittels STARTER und zur Kommunikation mit dem Embedded Controller über Profinet, wurde jedem Antriebssystem eine feste IP-Adresse zugewiesen. Die folgende Tabelle zeigt die eingestellten IP-Adressen.

Antrieb	IP-Adresse
1	192.168.225.10
2	192.168.225.20
3	192.168.225.30

Tabelle 5:IP-Adressen der Antriebssysteme

Die Profinet-Kommunikation der Antriebe mit dem Embedded-Controller erfolgt über freie Telegramme mit jeweils vier Eingangs- und Ausgangswörtern. Zur Kommunikation mit der Control-Unit wird das Siemens-Telegramm 390 eingesetzt. Diese wird jedoch nicht genutzt, sondern dient lediglich als Vorimplementierung für eine mögliche Erweiterung der Automatisierungslösung.

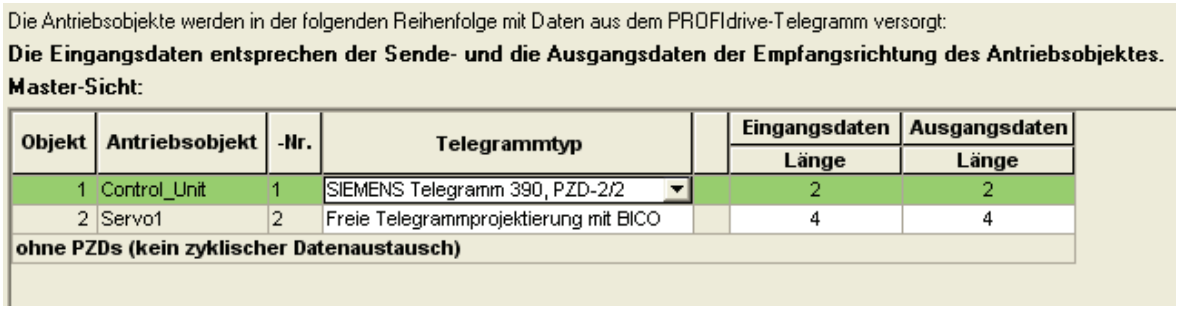


Abbildung 6.1-1:Telegramkonfiguration

Im Folgenden werden nun die Einstellungen für die Empfangs und Sendeeinrichtungen vorgestellt.

6.1.1 Empfangseinrichtung

Die Eingangsdaten jedes einzelnen Antriebs bestehen aus einem Steuerwort für die Umrichter, dem Sollwert für den Drehzahlregelkreis, der Drehzahlvorsteuerung sowie Drehmomentvorsteuerung des Antriebes jeweils als 16 Bit-Wort. Sie sind wie in Abbildung 6.1-2 zu sehen mit den entsprechenden Parametern verschaltet.

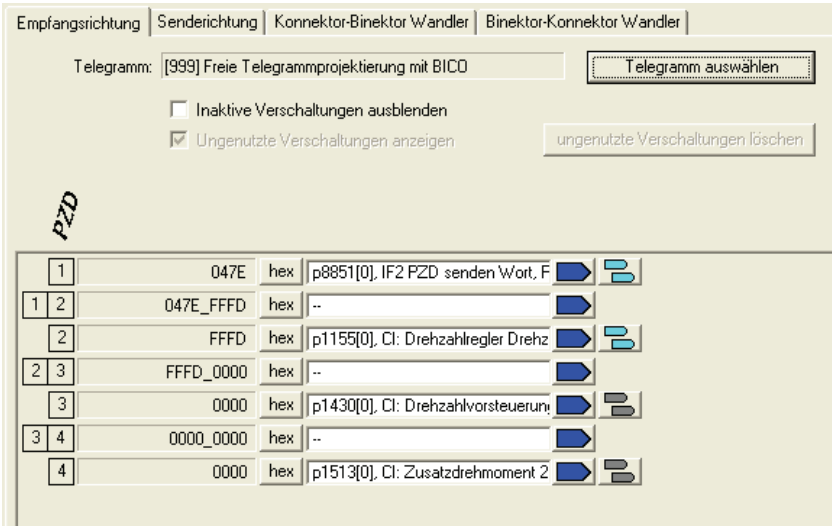


Abbildung 6.1-2:Einstellung der Eingangsdaten

Dabei werden die Werte für Drehzahl und Drehmoment wie in STARTER definiert nicht als absolute Werte vorgegeben, sondern prozentual zum Nennwert der Drehzahl oder zum Maximalmoment. Dabei entspricht ein Dezimalwert von 16384 Hundertprozent des entsprechenden Parameters. Diese Definition muss später bei der Umsetzung von Lageregelung und Vorsteuerung beachtet werden, um diese korrekt umzusetzen. Die folgende Tabelle zeigt die Parametrierung des Steuerwortes. Dieses wurde von Starter übernommen und musste nicht angepasst werden.

Bitnummer	Name	Beschreibung
0	Reserve	-
1	Reserve	-
2	Reserve	-
3	Reserve	-
4	Reserve	-
5	Reserve	-
6	Drem_betr	Momentgeregelter Betrieb
7	Reserve	-
8	Aus1	Aus 1
9	Aus2	Aus 2
10	Aus3	Aus 3
11	Freiwechsel	Freigabe Wechselrichter
12	Hochl_frei	Hochlaufgeber Freigabe
13	HL_start	Hochlaufgeber Starten
14	Frei_sollw	Freigabe Drehzahlsollwert
15	Stoer_ruecks	Störspeicher Rücksetzen / Störung quittieren

Tabelle 6:Steuerwort der Antriebe

6.1.2 Sendeeinrichtung

Jedes Antriebsystem sendet ein Telegramm bestehend aus vier Wörtern an die Steuerung auf dem Embedded Controller. Es besteht aus einem Zustandswort des Antriebes und der aktuellen Drehzahl. Außerdem wird ein Doppelwort mit der aktuellen Lage des Antriebes als Integer-Wert gesendet.

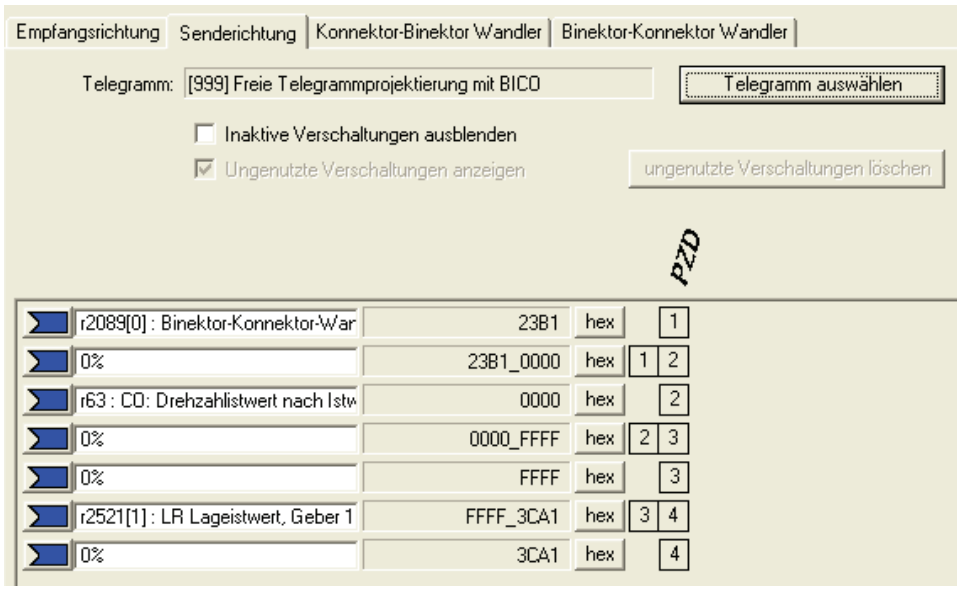


Abbildung 6.1-3:Einstellung der zu sendenden Daten

Das Statuswort des Antriebes ist wie folgt definiert. Es enthält auch Einstellungen zur Umrichter internen Lageregelung, die aber nicht genutzt werden.

Bitnummer	Name	Beschreibung
0	Kein_Schleppfehler	Kein Schleppfehler / Schleppfehler liegt an
1	Reserve	-
2	Sollposition_erreicht	Sollposition ist erreicht
3	Referenzpunkt_gesetzt	Referenzpunkt ist gesetzt
4	Sollwert_Quittierung	Sollwert ist Quittiert
5	Antrieb_steht	Antrieb steht
6	Reserve	-
7	Reserve	-
8	Einschaltbereit	Antrieb ist einschaltbereit
9	Betriebsbereit_keineStoer	Antrieb ist Betriebsbereit
10	Status_Regelfreigabe	Regelfreigbe
11	Stoerung_wirksam	Antrieb ist in Störung
12	Kein_Aus2_steht_an	Kein Aus2 steht an
13	Kein_Aus3_steht_an	Kein Aus3 steht an

14	Einschaltsrperre	Einschaltsperre aktiv
15	Warnung wirksam	Warnung liegt vor

Tabelle 7:Statuswort der Antriebe

6.2 Regler-Einstellungen

Die Regler-Einstellungen von Strom- und Drehzahlregelkreis erfolgt mittels der automatischen Regler-Optimierung von STARTER. Die Identifikation der Regelstecke geschieht mit einer Frequenzganganalyse des Antriebes. Im Anschluss werden nach einem dem Benutzter nicht bekannten Algorithmus, die Verstärkungsfaktoren und Nachstellzeiten der PI-Regler bestimmt. STARTER führt hierfür die folgenden Schritte aus.

- Vermessen der Mechanik Teil 1
  Die Mechanik des Antriebes wird im unteren Frequenzbereich mit einem geringen Drehzahlsollwert vermessen.
- Vermessen der Mechanik Teil 2
  Die Mechanik des Antriebes wird im oberen Frequenzbereich mit einem geringen Drehzahlsollwert vermessen.
- Identifikation des Stromregelkreises
  Durch Vorgabe eines Rauschsignales wird der Stromregelkreis identifiziert.
- Im Anschluss werden die Parameter der Regler berechnet

Die so ermittelten Parameter der drei Antriebssysteme sind in folgender Tabelle dargestellt.

	Antrieb 1	Antrieb 2	Antrieb 3
$K_{pi}[{}^{V}/_{A}]$	21,954	22,292	22,382
$T_{ni}[ms]$	2	2	2
$K_{pn}\left[{}^{Nm}/_{rad}\right]$	0,036	0,05	0,043
$T_{nn}[ms]$	6,19	4,05	5,14

Tabelle 8:Regelparameter der Antriebe

Die unterschiedlichen Regelparameter ergeben sich aus den von Antrieb zu Antrieb unterschiedlichen Ständerwiederständen und Induktivitäten, die sich bei der Vermessung ergeben haben.
Nach der automatischen Regler-Einstellung jedes Antriebes wurden anschließend, mittels der internen Messfunktion von STARTER, die Sprungantworten für Strom- und Drehzahlregelkreis aufgenommen, um so die korrekte Funktion der Regelung zu überprüfen.

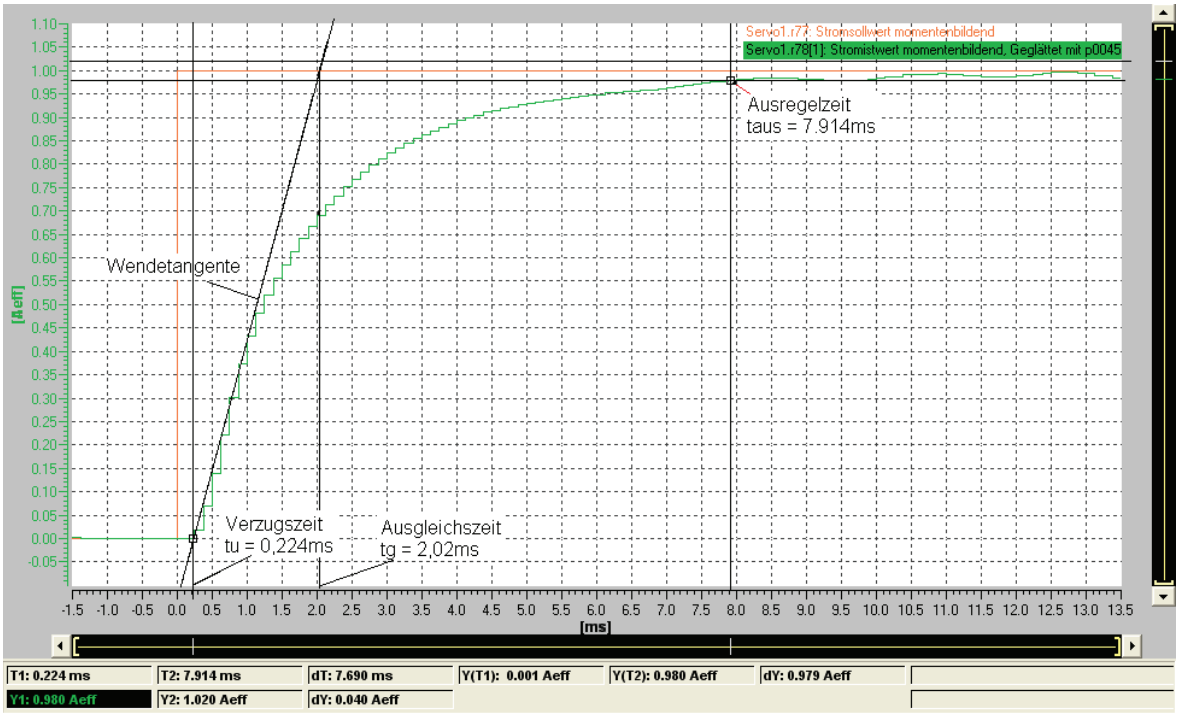


Abbildung 6.2-1:Sprungantwort Stromregelkreis 0 auf 1A

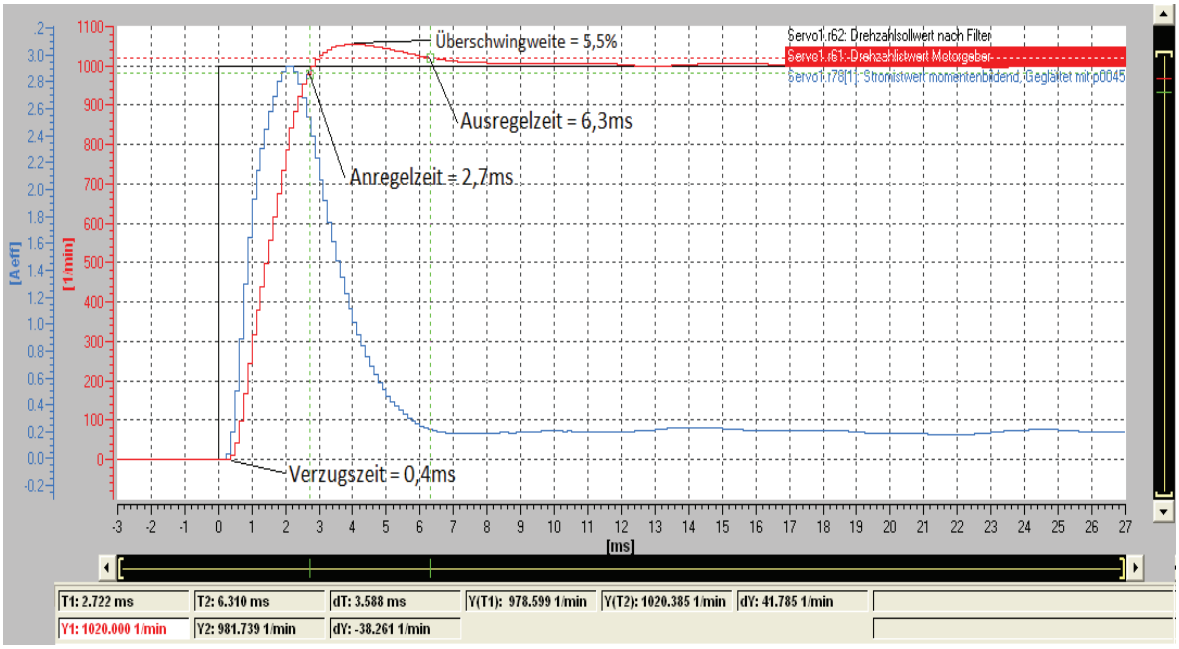


Abbildung 6.2-2:Sprungantwort Drehzahlregelkreis 0 auf 1000min^{-1}

Die Oberen Abbildungen zeigen die Sprungantworten des Strom- und Drehzahlregelkreises. Der Drehzahlregelkreis ist mit einer Ausregelzeit von 6,3ms hochdynamisch und damit für die Anforderungen an das Antriebssystem des Roboters geeignet. Die Abtastzeit der Istwert-Erfassung liegt bei 125µs.

6.3 Weitere Einstellungen

In diesem Abschnitt werden weitere Einstellungen des STARTER-Projektes vorgestellt. Dazu gehören das Implementieren einer Reibkennlinie in die Vorsteuerung des Drehmomentes der einzelnen Antriebe, die Einstellungen der Getriebeübersetzung und Auswertung der Winkelerfassung sowie das Einfügen der Vorsteuerwerte in den Drehzahlregelkreis.

6.3.1 Reibkennlinie

Neben der Drehmomentvorsteuerung des Führungsgrößengenerators, der die Drehmomente berechnet, die zur Bewegung der unteren Werkzeugmasse notwendig sind, gibt es in STARTER die Möglichkeit, das Reibmoment von Motor und Getriebe vorzusteuern. Dazu wird dieses Reibmoment bei verschiedenen Drehzahlen des Antriebs gemessen und in einer Kennlinie gespeichert. Bei der Drehzahlregelung des Antriebes wird diese Reibkennlinie anschießend in Abhängigkeit des Drehzahlistwertes vorgesteuert.

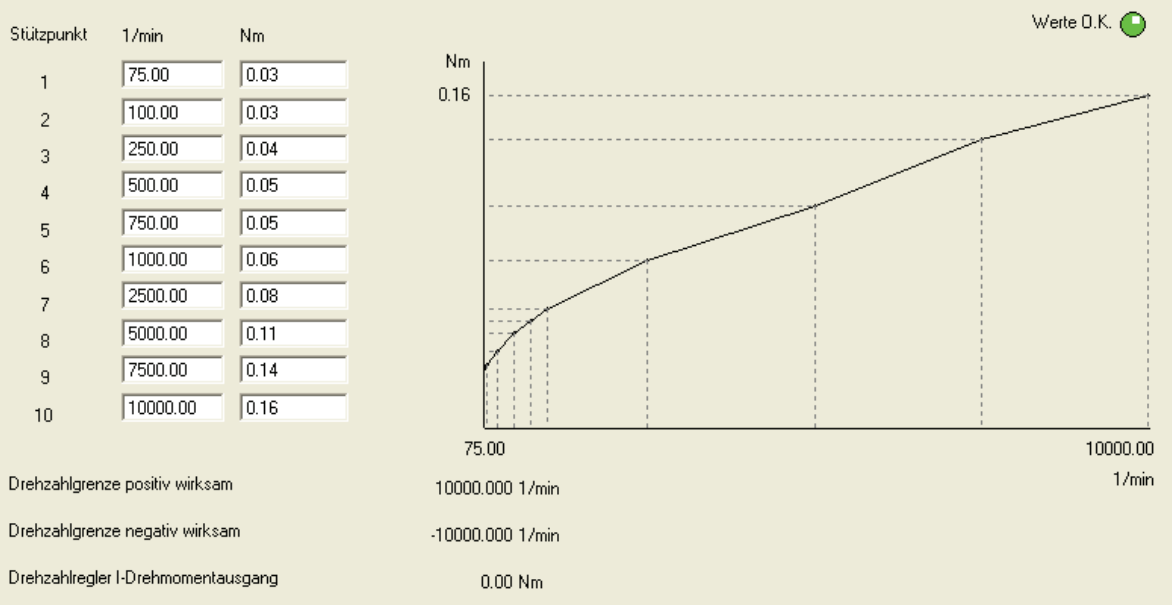


Abbildung 6.3-1:Reibkennlinie

Die Reibmomente wurden in verschiedenen Abständen von 75 bis 10000 min^{-1}, in positiver wie auch in negativer Richtung, gemessen. Diese Reibkennlinie wird auch bei der Modellierung des Drehzahlregelkreises angenähert, um das Antriebssystem möglichst genau zu simulieren. Siehe Gl.(7-1).

6.3.2 Mechanik und Winkelerfassung

Unter der Option Mechanik in der STARTER-Software kann die Geberauswertung des Antriebes konfiguriert werden. Die Getriebeübersetzung ist auf 40 Motorumdrehungen pro Lastumdrehung eingestellt. Der Lageistwert auf Lastseite wird in der Einheit LU[22] ausgegeben. In dieser Anwendung ist die Auflösung auf 360000 LUs pro Lastumdrehung eingestellt, also auf eine tausendstel Grad genau, um somit eine möglichst genaue Positionierung vornehmen zu können. Die Modulokorrektur, die den Lageistwert bei einem bestimmten Wert wieder auf null setzt, ist aufgeschaltet. Grund hierfür ist, dass es nie zu einer ganzen Lastumdrehung des Roboterarmes kommen kann.

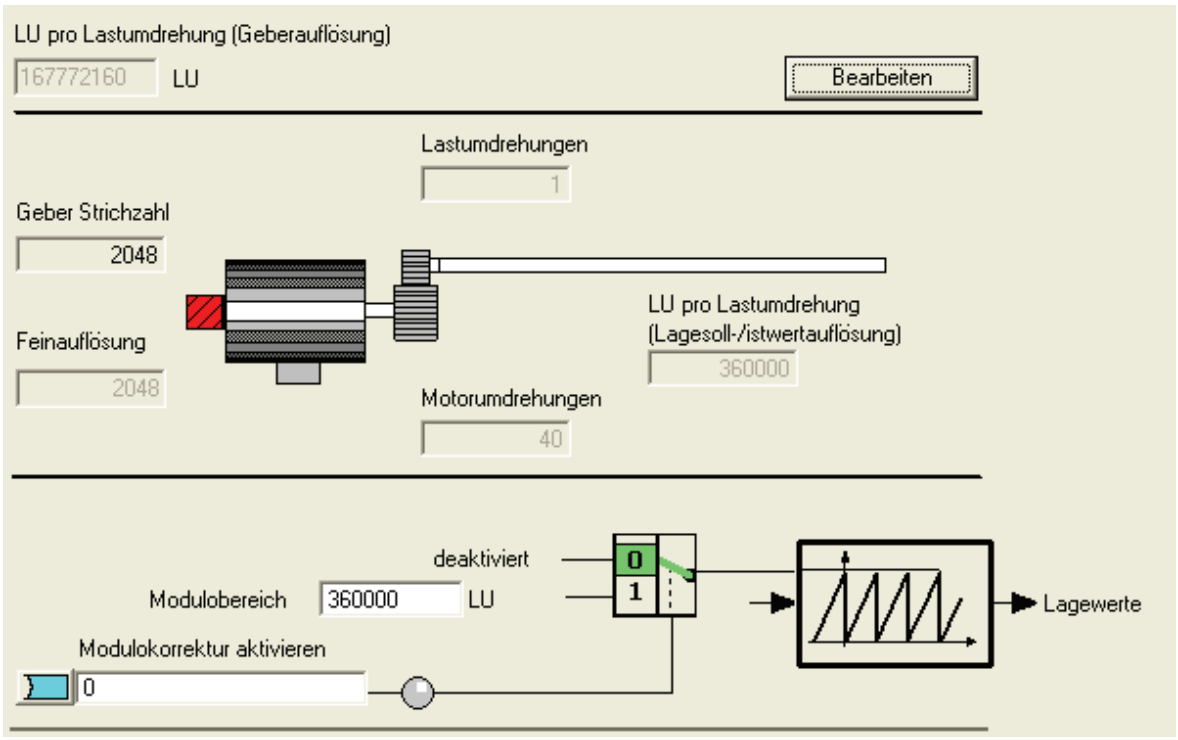


Abbildung 6.3-2:Einstellungen der Mechanik

Die obige Abbildung zeigt die Maske, in der die entsprechenden Einstellungen vorgenommen werden. Zusätzlich wurde innerhalb der Expertenliste der Antriebe eingestellt, dass der aktuelle Lageistwert auf Lastseite im nicht Flüchtigen Speicher der Umrichter-CU[23] gespeichert wird. Somit ist es möglich, die Position des Roboters über das Abschalten der Anlage zu erhalten. Andernfalls müssten die Roboterarme bei jedem Neustart neu kalibriert werden.

[22] Lastumdrehungen

[23] Control Unit

6.3.3 Vorsteuerung

Zur Drehzahl- und Drehmomentvorsteuerung müssen die entsprechenden Parameter, der Eingangsdaten, verschaltet werden. Dies geschieht in den Masken „Drehmomentsollwertfilter“ und „Momentensollwerte“ in STARTER.

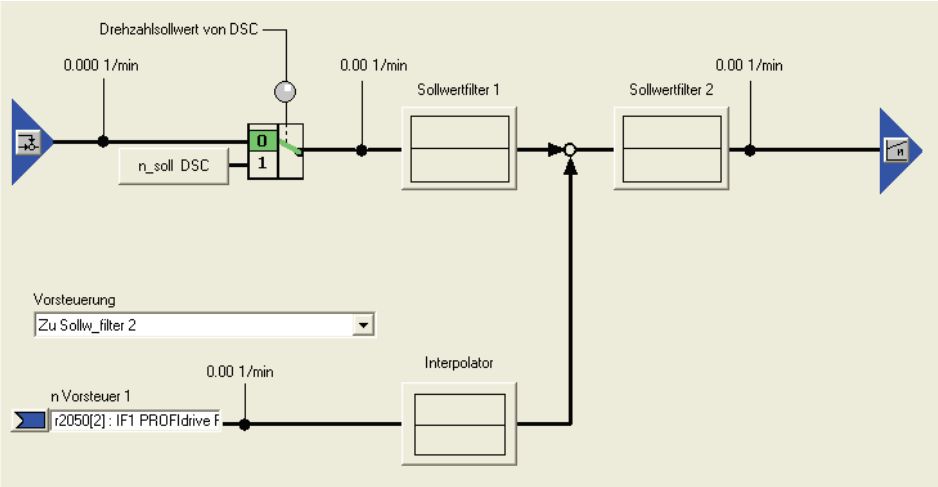


Abbildung 6.3-3:Drehzahlvorsteuerung

In der obigen Abbildung ist die Maske „Drehmomentsollwertfilter“ zu sehen. Der Vorsteuerwert wird mit „n Vorsteuer 1“ verschaltet, der unten in der Abbildung zu sehen ist. Die Sollwertfilter können optional eingestellt werden, was hier nicht geschehen ist. In Abbildung 6.3-4 ist die Maske „Momentensollwerte“ dargestellt. Hier wird sowohl die Reibkennlinie „Zusatzdrehmoment 3“ als auch der Vorsteuerwert des Führungsgrößengenerators „Zusatzdrehmoment 2“ verschaltet.

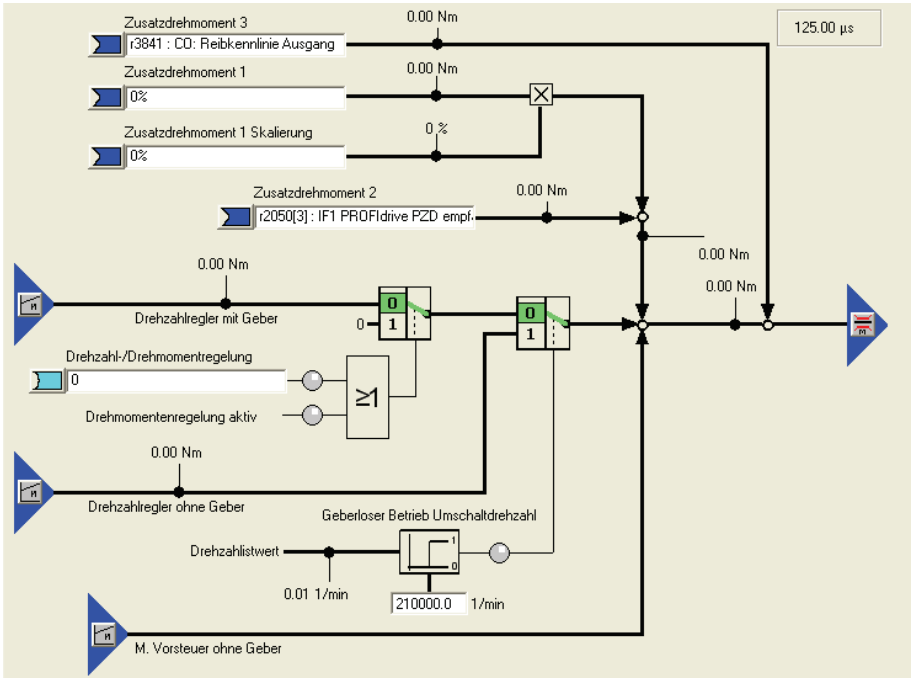


Abbildung 6.3-4:Drehmomentvorsteuerung

7 Simulation

Um den Ansatz der Lageregelung mit Bahninterpolation und Führungsgrößengenerator zu überprüfen, wird das Gesamtsystem des Roboters simuliert. Dabei wird zuerst das Simulationsmodel der Synchronmaschine vorgestellt. Anschließend wird das unter 4.2 ermittelte Model des Delta-Roboters simuliert und gegen das mittels SimMechanics erstellte Simulink-Model von Herrn Habermann getestet. Zum Abschluss des Kapitels wird das Gesamtsystem vorgestellt.

7.1 Synchronantriebe

Dem Model der Synchronmaschinen liegen die in Tabelle 4 dokumentierten Parameter zu Grunde. Es entspricht der unter 4.3 vorgestellte theoretische Betrachtung.

Abbildung 7.1-1 zeigt das verwendete Model der Synchronmaschine.

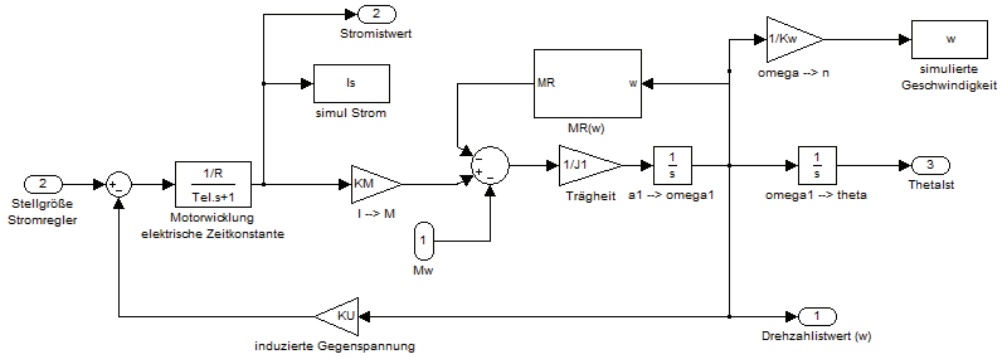


Abbildung 7.1-1:Simulink-Modell der Synchronmaschine

Die Werte für Iststrom und Drehzahlistwert werden mittels der „To Workspace“ Blöcke gespeichert und anschließend ausgewertet.

Das innere Reibmoment ist in Anlehnung an 6.3.1 wie folgt realisiert.

$$M_R(\omega) = \frac{M_R(5000min^{-1}) - M_R(100min^{-1})}{(5000min^{-1} - 100min^{-1})\frac{2\pi}{60}}\omega + M_{Roffset} \quad \textbf{(7-1)}$$

$$= \frac{0{,}11Nm - 0.03Nm}{(4900min^{-1})\frac{2\pi}{60}}\omega + 0{,}03Nm = 1{,}56 \cdot 10^{-3}Nm \cdot \omega + 0.03Nm$$

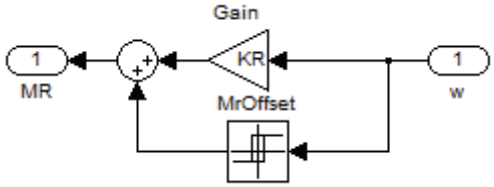


Abbildung 7.1-2:Simulinkmodell von $M_R(\omega)$

7.1.1 Drehzahlregelung

Der Drehzahlregelkreis wurde in Matlab/Simulink wie folgt modelliert.

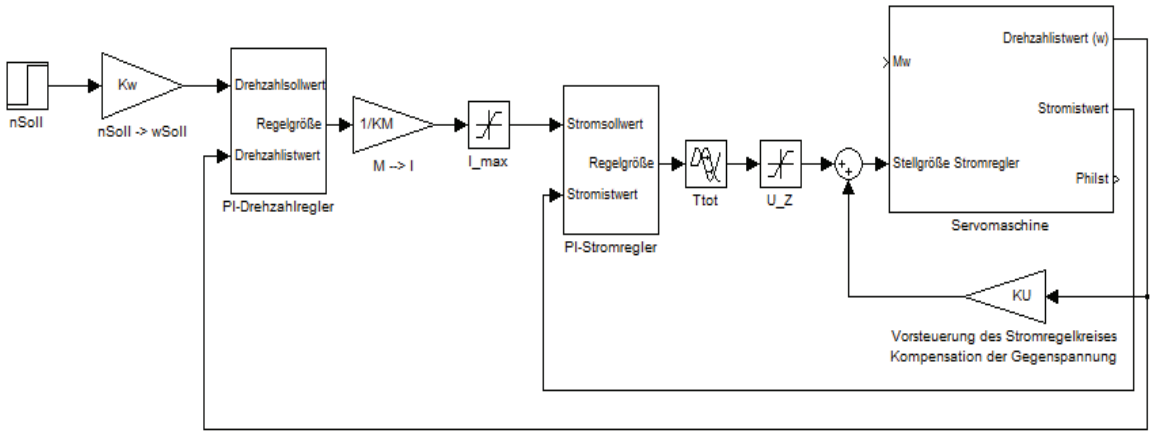


Abbildung 7.1-3:Drehzahlregelkreis der Synchronmaschine

Das Simulationsmodell ist der in Abbildung 4.3-2 dargestellten Kaskadenregelung nachempfunden. Die Parameter für die PI-Regler für Strom- und Drehzahlregelkreis wurden aus STARTER übernommen. Sie lauten wie folgt:

Regelparameter:

$K_{pi} = 21{,}954\,{}^{V}/_{A}$	*Proportionalverstärkung Stromregler*
$T_{ni} = 2ms$	*Nachstellzeit Stromregler*
$K_{pn} = 0{,}036\,{}^{Nm}/_{rad}$	*Proportionalverstärkung Drehzahlregler*
$T_{nn} = 6{,}19ms$	*Nachstellzeit Drehzahlregler*

Die PI-Regler sind folgendermaßen realisiert.

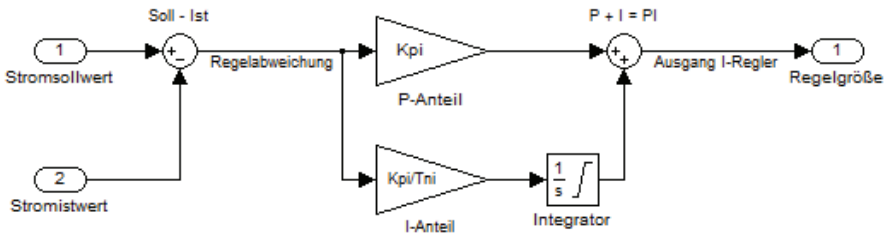


Abbildung 7.1-4:PI-Regler des Stromregelkreises

Um die Korrektheit der Dynamik des Simulationsmodells zu validiert, werden nun die Sprungantworten der Simulation und des realen Systems verglichen. Die Sprungantworten wurden vom Servo Nummer 1 aufgenommen. Abbildung 7.1-5 zeigt die simulierten Sollwertsprungantwort von $n = 0\ min^{-1}$ auf $n = 100\ min^{-1}$.

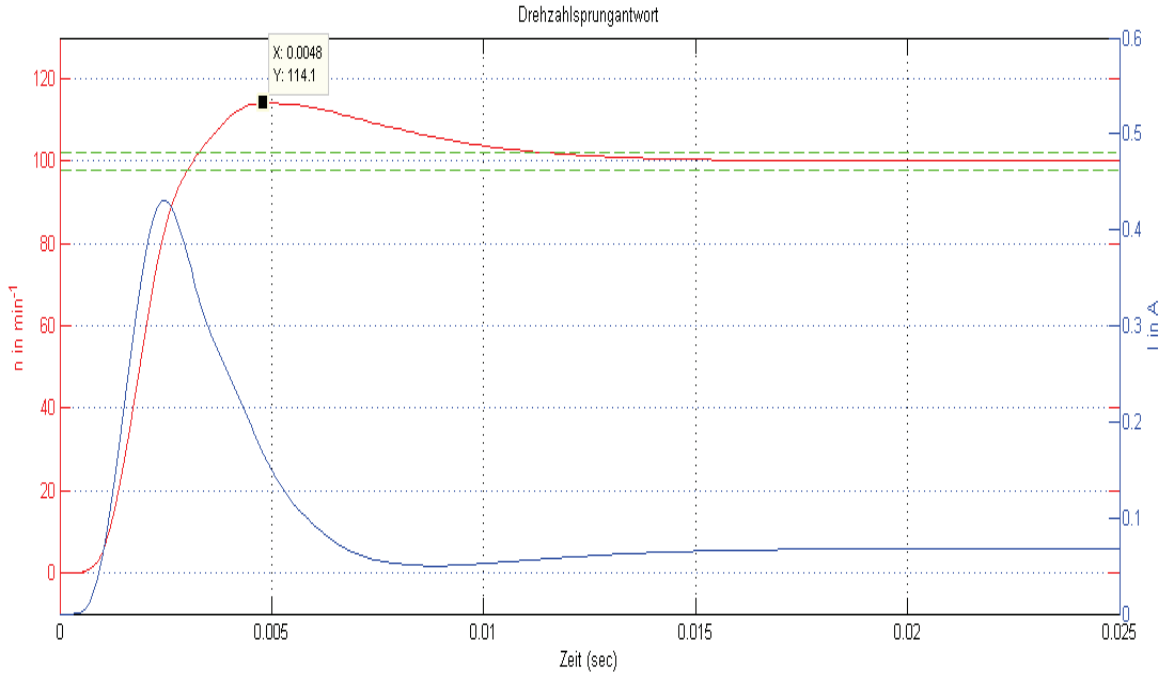


Abbildung 7.1-5:Drehzahlsollwertsprung simuliert 0 auf 100min^{-1}

Im Vergleich dazu zeigt Abbildung 7.1-6 die Sollwertsprungantwort des realen Regelreises.

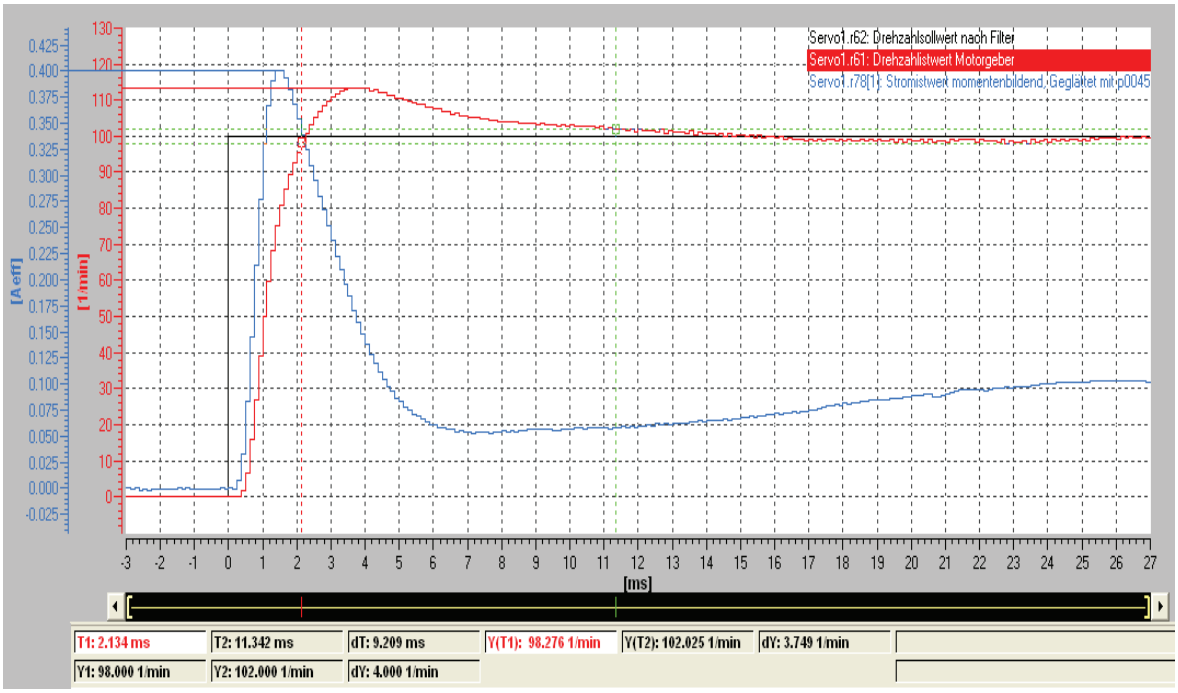


Abbildung 7.1-6:Drehzahlsollwertsprung Realsystem 0 auf 100min^{-1}

Dabei entspricht die rote Kurve dem Drehzahlistwert n des Antriebes und die blaue Kurve dem momentbildenen Stromistwert I. Deutlich zu sehen ist, dass die Stellgröße des momentbildenen Stromes nicht in die Begrenzung I_{max} geht. Der Vergleich mit der Sprungantwort des realen Systems zeigt, dass die Unterschiede sehr gering sind. Die Unterschiede in der Dynamik des Stromistwertes lassen sich damit erklären, dass dieser im realen System geglättet wurde. Die geringere Auflösung der Sprungantwort des realen Systems ergibt sich aus der Abtastzeit des Umrichters $T_{abt} = 125\mu s$. Der Vergleich der An- und Ausregelcharakteristik sowie der Überschwingweite sieht folgendermaßen aus:

	Simulink-Model	Reales System
Anregelzeit:	$3ms$	$2{,}13ms$
Ausregelzeit:	$11{,}5ms$	$11{,}34ms$
Überschwingweite:	$14{,}1\%$	$13{,}5\%$

Ein zweiter Vergleich zeigt den Sollwertsprung von $n = 0\ min^{-1}$ auf die Nenndrehzahl $n_N = 6000\ min^{-1}$.

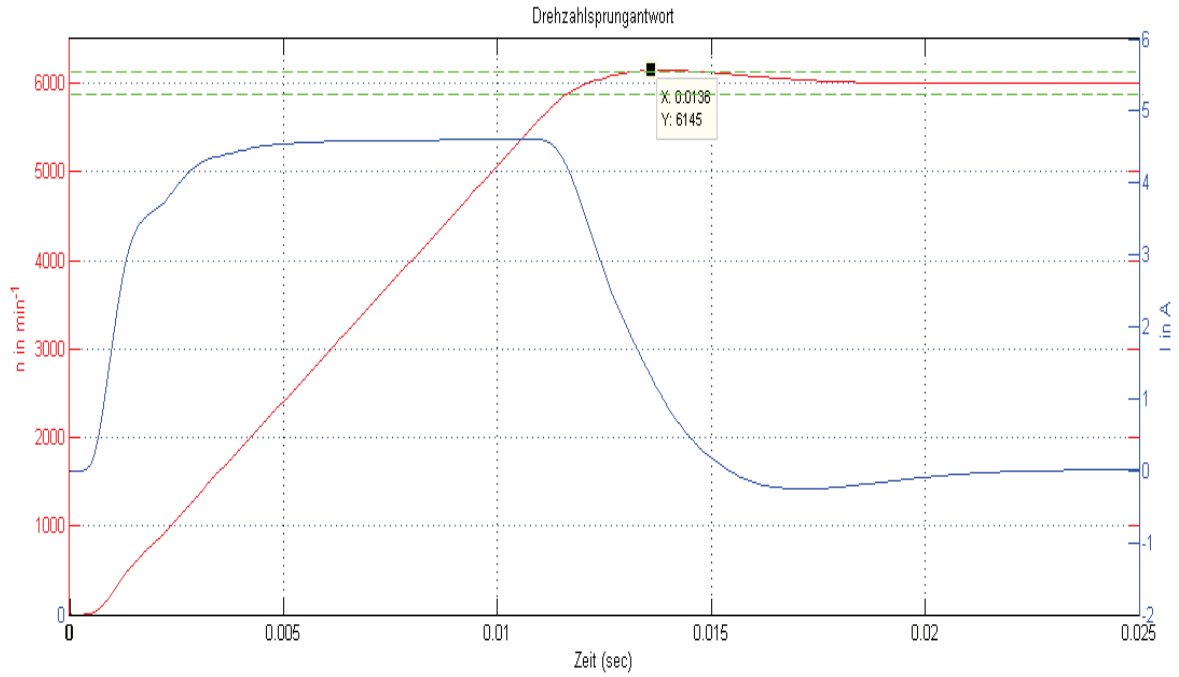


Abbildung 7.1-7:Drehzahlsollwertsprung simuliert 0 auf 6000min^{-1}

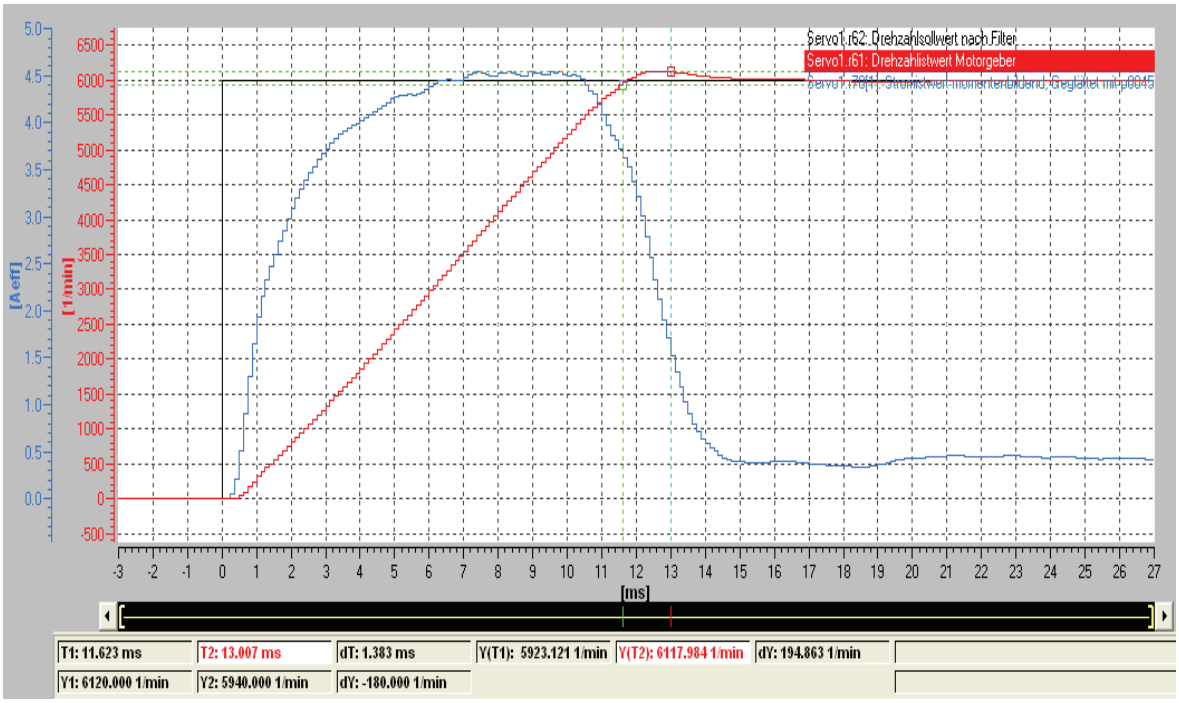


Abbildung 7.1-8:Drehzahlsollwertsprung Realsystem 0 auf 6000min^{-1}

Sowohl in der Simulation als auch im realen System geht der momentbildene Stromistwert in die Begrenzung I_{max}. Dies sorgt für einen linearen Anstieg des Drehzahlistwertes. Beide Sprungantworten zeigen einen sehr ähnlichen Verlauf auf. Der Vergleich der An- und Ausregelcharakteristik sowie der Überschwingweite sieht wie folgt aus:

	Simulink-Model	Reales System
Anregelzeit:	$11{,}6ms$	$11{,}6ms$
Ausregelzeit:	$15ms$	$13ms$
Überschwingweite:	$2{,}4\%$	$2{,}2\%$

Der Vergleich der Simulation mit dem realen Drehzahlregelkreis zeigt, dass das Simulationsmodel ausreichend ist um die Roboterdynamik zu simulieren.

7.1.2 Lageregelung

Der Drehzahlregelkreis aus 7.1.1 wird nun um den in Step7 implementierten Lageregler erweitert. Somit ergibt sich der vollständige Lageregelkreis des Antriebssystems. Abbildung 7.1-9 zeigt dass entsprechende Simulink-Modell.

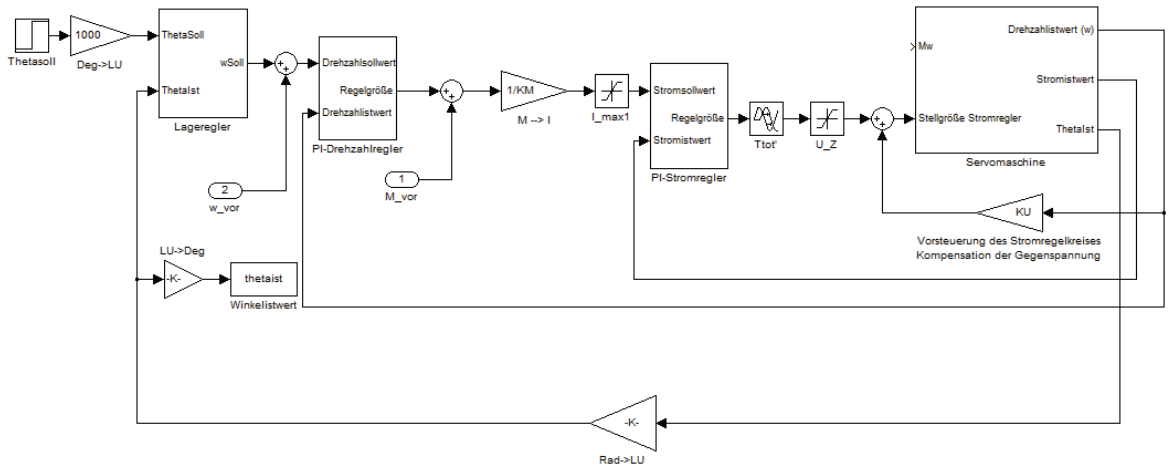


Abbildung 7.1-9:Lageregelkreis der Synchronmaschine

Da die Positionsangaben wie unter 6.3.2 erklärt in LU angegeben werden, wobei eine Umdrehung auf der Lastseite der Getriebe 360000 LU entspricht, wird der Winkelist- und Sollwert in Rad bzw. Grad zuerst in LU umgerechnet, bevor aus ihnen die Regeldifferenz bestimmt wird. Im obigen Modell sind die Aufschaltpunkte der Vorsteuerung von Drehzahl und Drehmoment bereits eingefügt. Der Lageregler ist als P-Regler ausgelegt. Um das Führungsverhalten des Lageregelkreises möglichst genau zu simulieren, ist er als diskreter Regler mit einer Abtastzeit von 2ms implementiert. Dies simuliert die Zykluszeit der Steuerung, zu der der reale Regler ausgeführt wird. Abbildung 7.1-10 das Subsystem des Lagereglers.

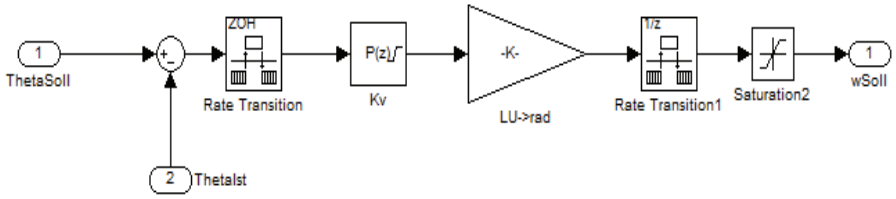


Abbildung 7.1-10:Lageregler

Zur Veranschaulichung der Dynamik des Lageregelkreises wurde ein Sollwertsprung von $\theta = 0\ auf\ 45°$ simuliert und mit der Messung der gleichen Sprungantwort des realen Systems verglichen. Der Verstärkungsfaktor des Lagereglers in auf $K_v = 700$ eingestellt.

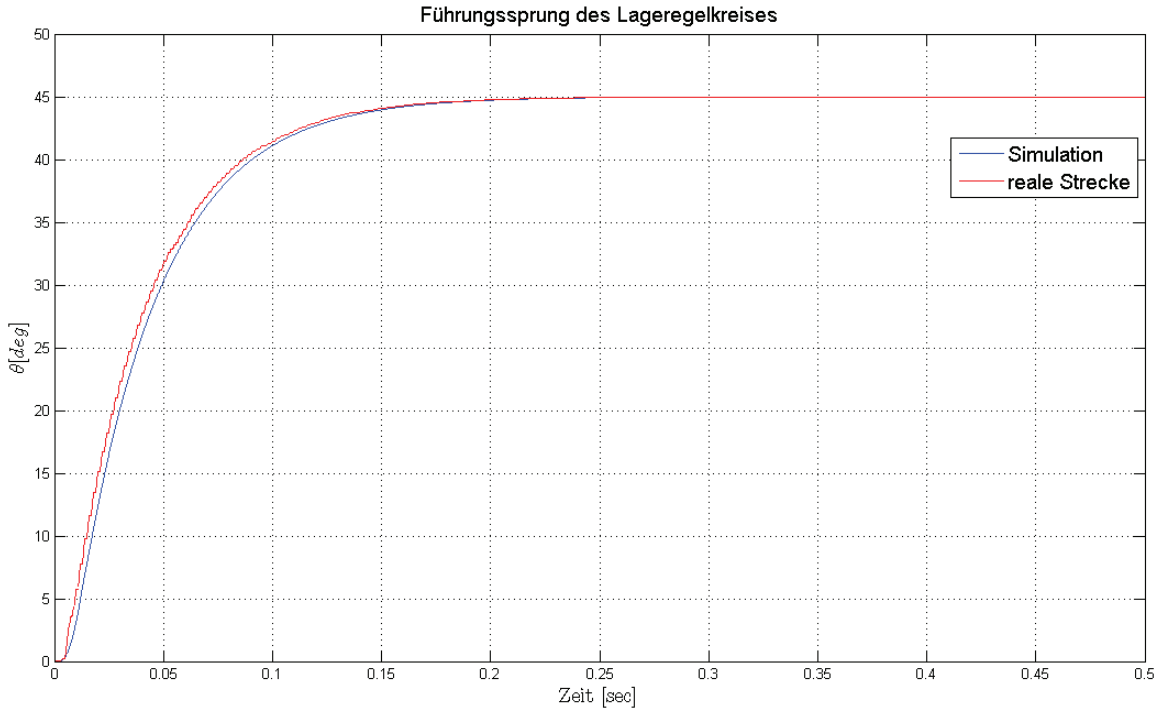


Abbildung 7.1-11:Führungssrungantwort des Lageregelkreises

Der Führungssprung wird zum Zeitpunkt $t = 0$ ausgeführt. Die Verzugszeit liegt bei beiden Kurven bei ca. 3ms. Dabei regelt das reale System etwas schneller ein als das simulierte System. Grund hierfür ist das Vereinfachte Modell der Synchronmaschine sowie die Tatsache, dass im realen Antriebssystem eine Vorsteuerung des Reibmomentes implementiert ist. Wie zu sehen ist, ist das Simulationsmodell des Lageregelkreises eine gute Näherung des realen Regelkreises. Somit kann das Modell des Lageregelkreises zur Simulation des Gesamtsystems des Roboters genutzt werden.

7.2 Roboterdynamik

In diesem Abschnitt wird die Modellierung der Roboterdynamik überprüft. Dazu wird das modellierte Robotermodel gegen das SimMechanics-Modell von Herrn Habermann getestet. Das Simulink-Model sieht hier wie folgt aus.

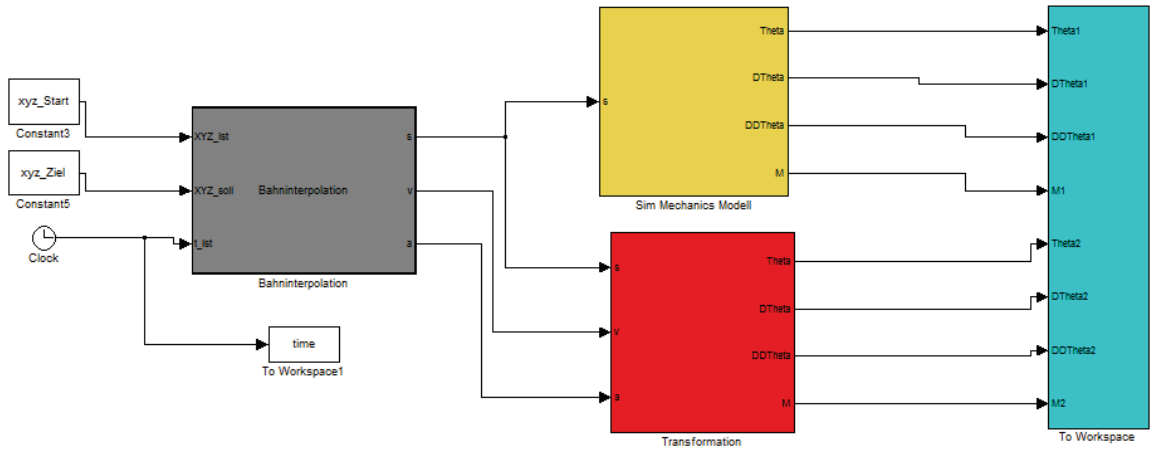


Abbildung 7.2-1:Simulink-Model der Roboterdynamik

In dem grauen Subsystem ist eine „Embedded MATLAB Function“ implementiert, die nach Abschnitt 5.2 die Sollbahn, Geschwindigkeiten und Beschleunigung im translatorischen Koordinatensystem berechnet. Die XYZ-Koordinaten werden anschließend an das gelbe Subsystem, in dem sich das SimMechanics-Model von Herrn Habermann befindet und an das rot markierte eigene Robotermodel weitergegeben. Das eigene Model benötigt zusätzlich die Trajektorien für Geschwindigkeit und Beschleunigung. Beide Robotermodelle geben die Winkelistwerte, Winkelgeschwindigkeiten und Winkelbeschleunigungen im Gradmaß sowie die wirkenden Drehmomente wieder. Diese können anschließend verglichen werden. Der Grund für ein eigenes Modell ist, dass sich aus dem vorhandenen SimMechanics-Model kein für den Führungsgrößengenerator notwendiger C++-Code generieren lässt. Im Gegensatz dazu kann das selbst erstellte Modell einfach in C++ umgeschrieben werden.

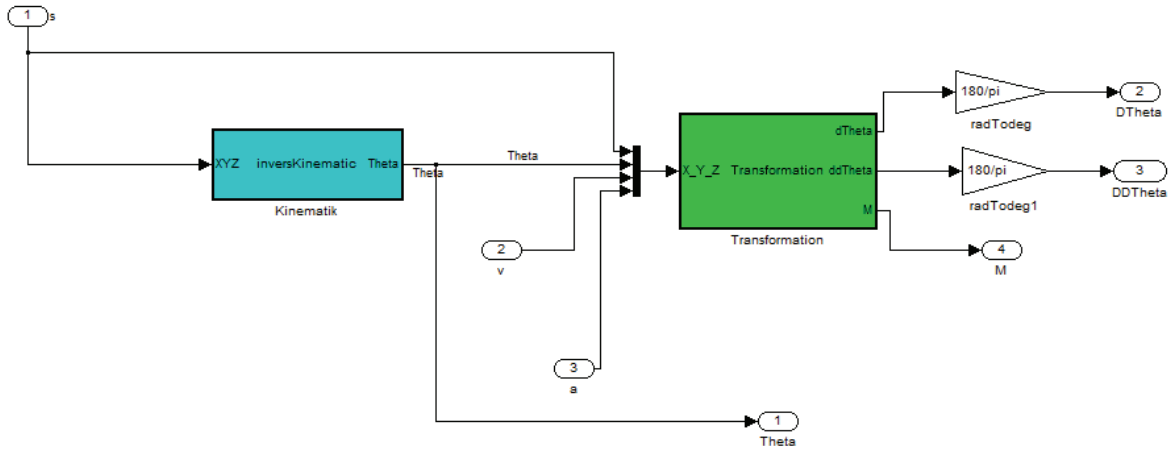


Abbildung 7.2-2:Subsystem der Roboterdynamik

Das Subsystem der selbst erstellten Roboterdynamik besteht im Wesentlichen aus zwei Blöcken. Der erste berechnet aus der aktuellen XYZ-Position die entsprechenden Roboterwinkel mittels der inversen Kinematik. In dem zweiten Block erfolgt die Transformation der Geschwindigkeits- und Beschleunigungsinformation mittels der XYZ-Koordinaten und Winkel, in die Winkelgeschwindigkeiten, Beschleunigungen und Drehmomente der Roboterarme.

Zum Model von Herrn Habermann ist zusätzlich zu sagen, dass hier eine andere Definition des Koordinatenursprungs gewählt wurde. Deshalb ist hier noch eine Koordinatentransformation, in Form einer Transformationsmatrix, notwendig.

Zum Vergleich der beiden Modelle wird nun eine Fahrt von

$$\vec{E}_0 = \begin{pmatrix} 0 \\ 0{,}3 \\ -0.6 \end{pmatrix} [m]$$

nach

$$\vec{E}_0 = \begin{pmatrix} 0 \\ -0{,}3 \\ -0.6 \end{pmatrix} [m]$$

simuliert. Mit einer Geschwindigkeit von $v_{max} = 1\,{}^{m}/_{s}$. Die Auswertung ergibt folgendes:

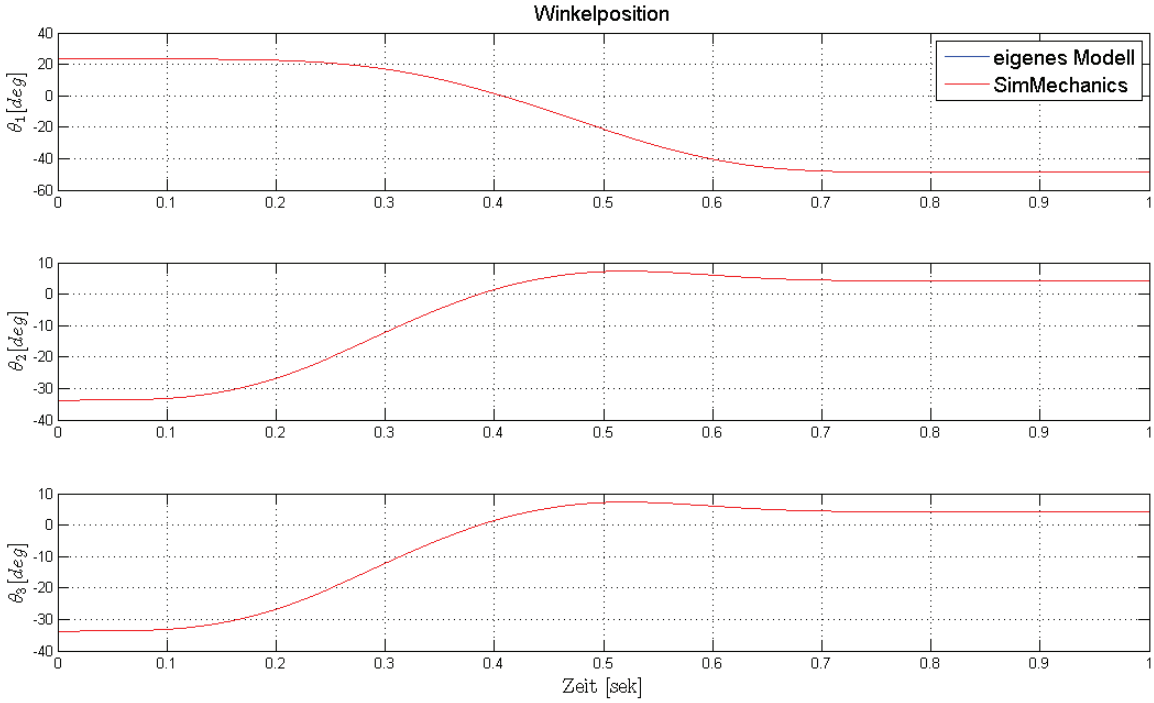


Abbildung 7.2-3:Vergleich der Winkelpositionen

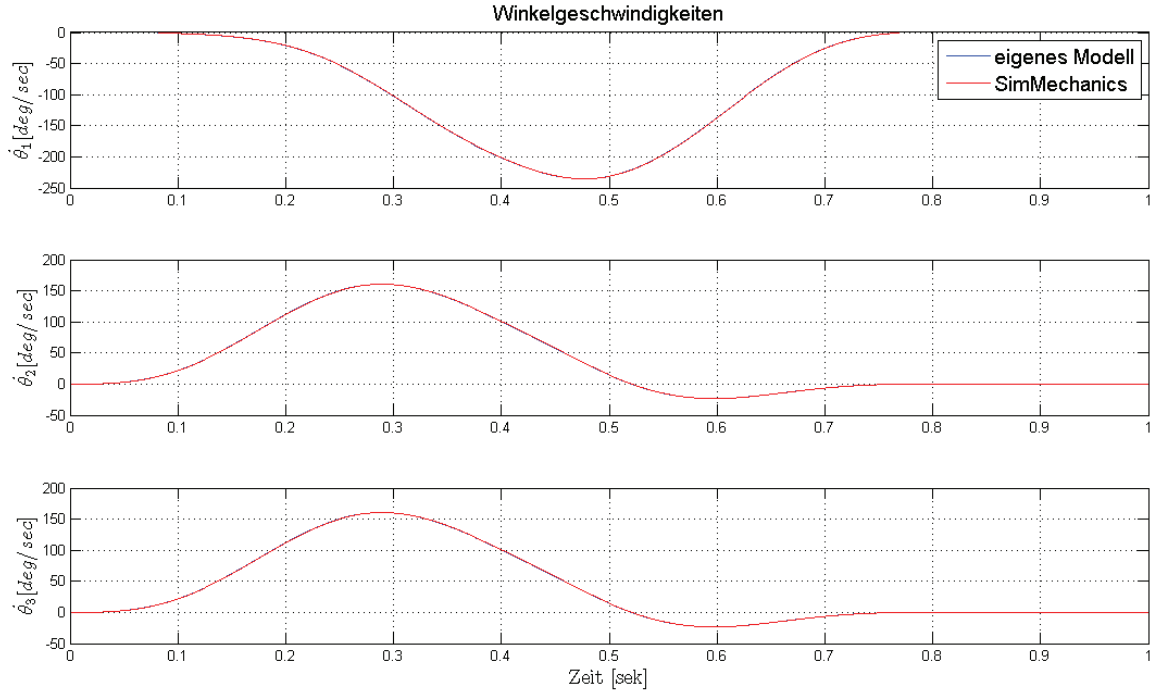


Abbildung 7.2-4:Vergleich der Winkelgeschwindigkeiten

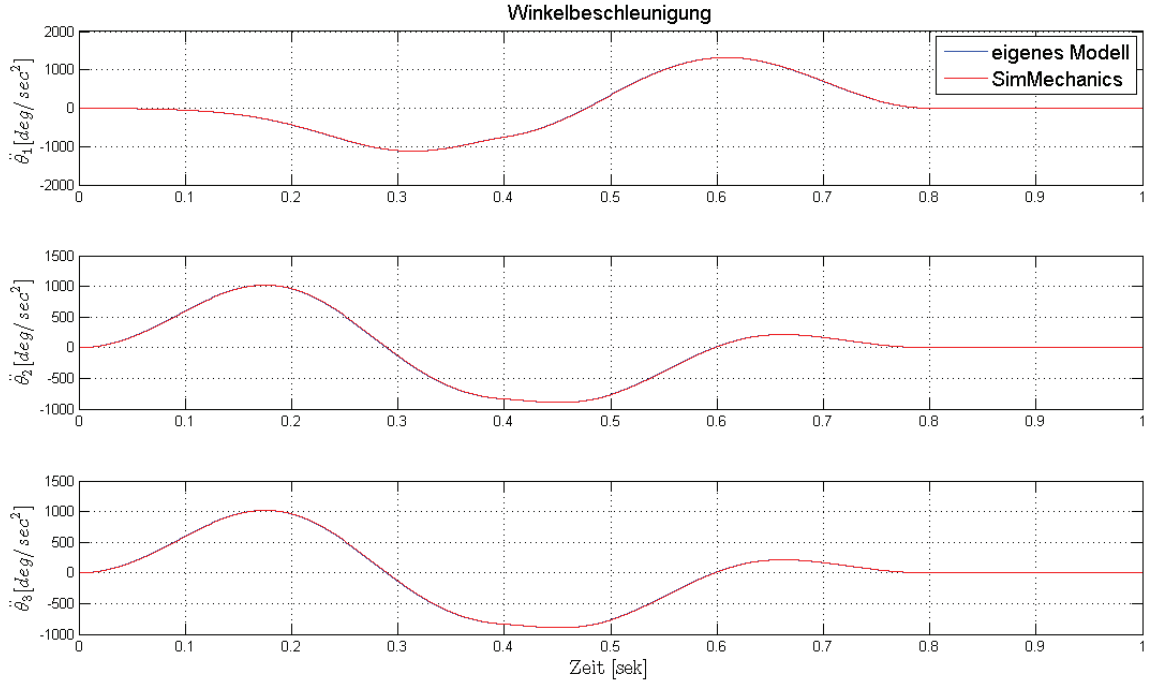


Abbildung 7.2-5:Vergleich der Winkelbeschleunigungen

Der Vergleich der Winkel, Winkelgeschwindigkeiten und Beschleunigungen zeigt, dass die Berechnungen beider Modelle übereinstimmen, da die Kurven deckungsgleich sind.

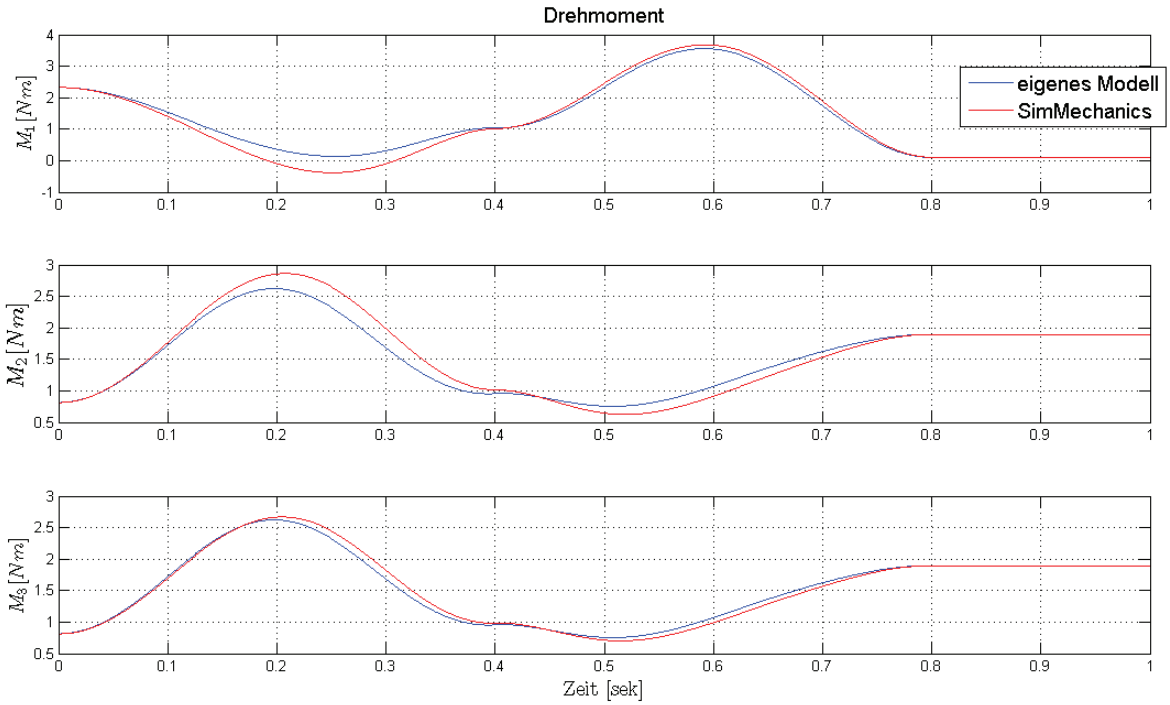


Abbildung 7.2-6:Vergleich der wirkenden Drehmomente

Der Vergleich der Drehmomente offenbart, dass hier die Berechnungen unterschiedlich sind. Nach dem lange nach möglichen Fehlern in der Berechnung gesucht wurde, ohne

solche zu finden, wurde Rücksprache mit Herrn Habermann gehalten. Nach seiner Auffassung kann seine Berechnung der Drehmomente nicht exakt stimmen. Grund hierfür ist, dass nicht jeder dafür notwenige Trägheitstensor ausgerechnet wurde. Für seine Arbeit war eine genaue Berechnung der Drehmomente nicht notwendig, sodass sein Model ausreichend ist. Dies wird auch aus Abbildung 7.2-6 deutlich. Obwohl Arm zwei und drei dieselbe Bewegung mit exakter Geschwindigkeit und Beschleunigung durchführen, verhalten sich die Drehmomente beim SimMechanics-Model unterschiedlich. Aus diesem Grund werden die abweichenden Drehmomente hingenommen, da eine genauere Überprüfung nicht möglich ist.

7.3 Gesamtsystem

In diesem Abschnitt wird das Gesamtsystem des Roboters simuliert. Dazu wurden die Bahninterpolation, die Vorsteuerung, das Antriebssystem und die Roboterdynamik in einem Simulink-Model zusammengefügt. Um den Einfluss einer Vorsteuerung zu verdeutlichen, werden anschließend die Winkel- und Positionsistwerte mit und ohne Vorsteuerung verglichen. Das Model des Gesamtsystems sieht wie folgt aus.

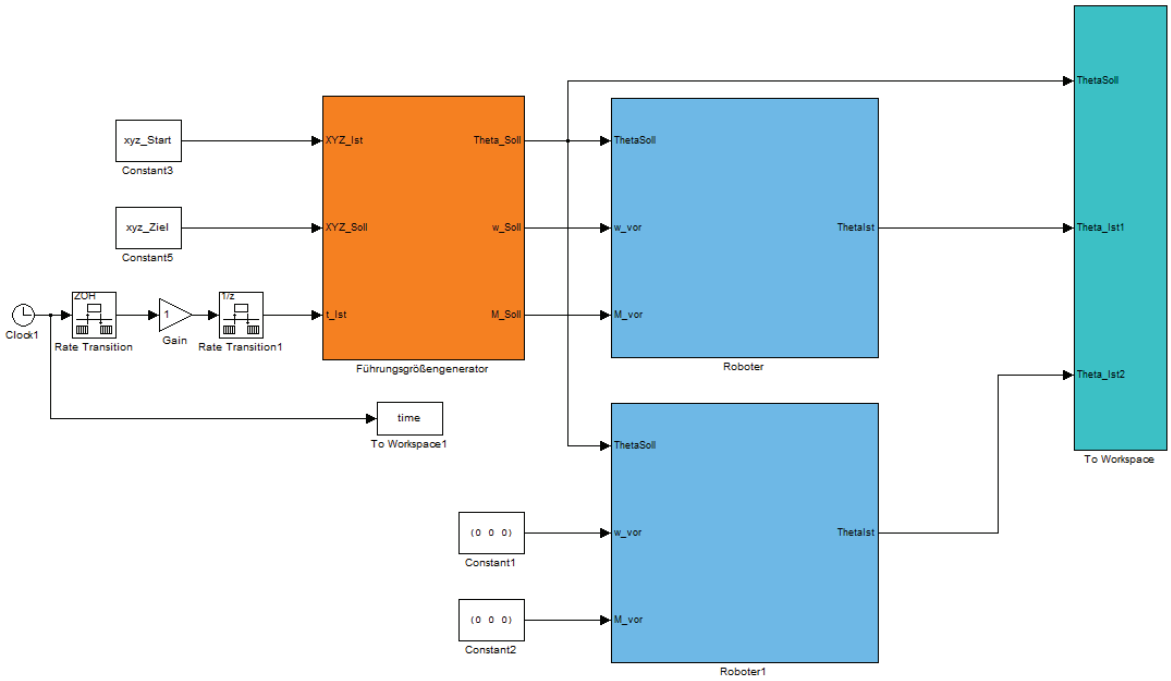


Abbildung 7.3-1:Simulink-Model des Gesamtsystems

Das orangene Subsystem stellt hierbei den Führungsgrößengenerator dar. Er besitzt als Eingänge Ist- und Sollposition sowie die aktuelle Zeit zur Interpolation der Solltrajektorien. Die translatorischen Trajektorien werden anschließend in die benötigten rotatorischen

Trajektorien umgerechnet. Ausgegeben werden die Trajektorien der Winkel-, Winkelgeschwindigkeits und Drehmomentsollwerte an die Robotermodelle.

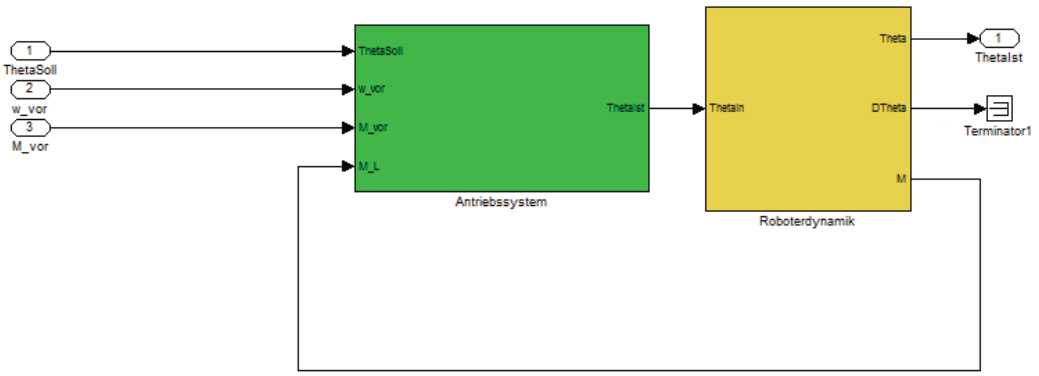


Abbildung 7.3-2:Subsystem Roboter

Das Subsystem der Roboter besteht aus dem Antriebssystem und der Roboterdynamik. Das Antriebsystem ist aus drei gleichen Modellen der Synchronmaschinen aus 7.1 zusammengestellt. Hier gehen die bereits angesprochenen Trajektorien zur Vorsteuerung ein. Das Antriebsystem gibt die Winkelistwerte aus, aus denen die Roboterdynamik die wirkenden Drehmomente berechnet. Diese werden wieder den entsprechenden Antrieb als Lastmoment vorgegeben.

Da sich der dem Führungsgrößengenerator vorgegebene Zeitwert, softwarebedingt im 10ms Takt ändert, wird dieser auch hier diskret vorgegeben. Die Folge ist, dass sich Trajektorien der Winkelsollwerte und der Vorsteuerung nur alle 10ms ändern. Der Grund für diese Diskretisierung wird in Abschnitt 8.4 erläutert.

7.3.1 Linearfahrt

Zunächst wird nun eine lineare Fahrt des Roboters zwischen zwei Punkten simuliert. Die Start und Zielkoordinaten sind hierfür folgendermaßen definiert.

$$xyz_{Start} = \begin{pmatrix} 0{,}25 \\ 0{,}25 \\ -0.6 \end{pmatrix}[m]; xyz_{Ziel} = \begin{pmatrix} -0{,}25 \\ -0{,}25 \\ -0.6 \end{pmatrix}[m]$$

Die Maximalgeschwindigkeit ist mit $v_{max} = 1\,{}^{m}/_{s}$ eingestellt.

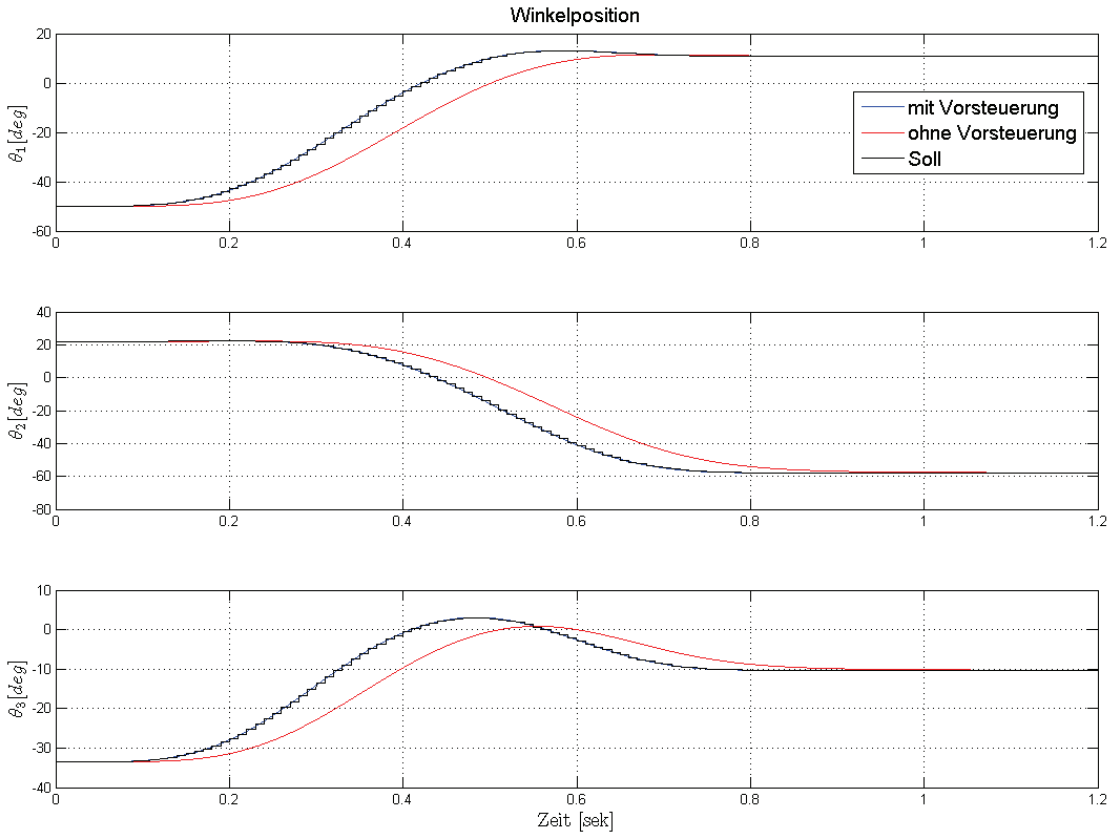


Abbildung 7.3-3:Winkelistwert der simulierten Linearfahrt

Wie erwartet wurde zeigt der Vergleich der Winkelistwerte mit und ohne Vorsteuerung eine signifikante Verbesserung. Mit der Vorsteuerung folgen die Istwerte fast genau den Sollwerttrajektorien. Ohne diese Vorsteuerung eilen die Istwerte den Sollwerten hinterher. Grund hierfür ist, neben den wirkenden Lastmomenten, die Dynamik der Lageregelkreise der Antriebe, die ca. 200ms benötigen um ihren derzeitigen Sollwert zu erreichen.

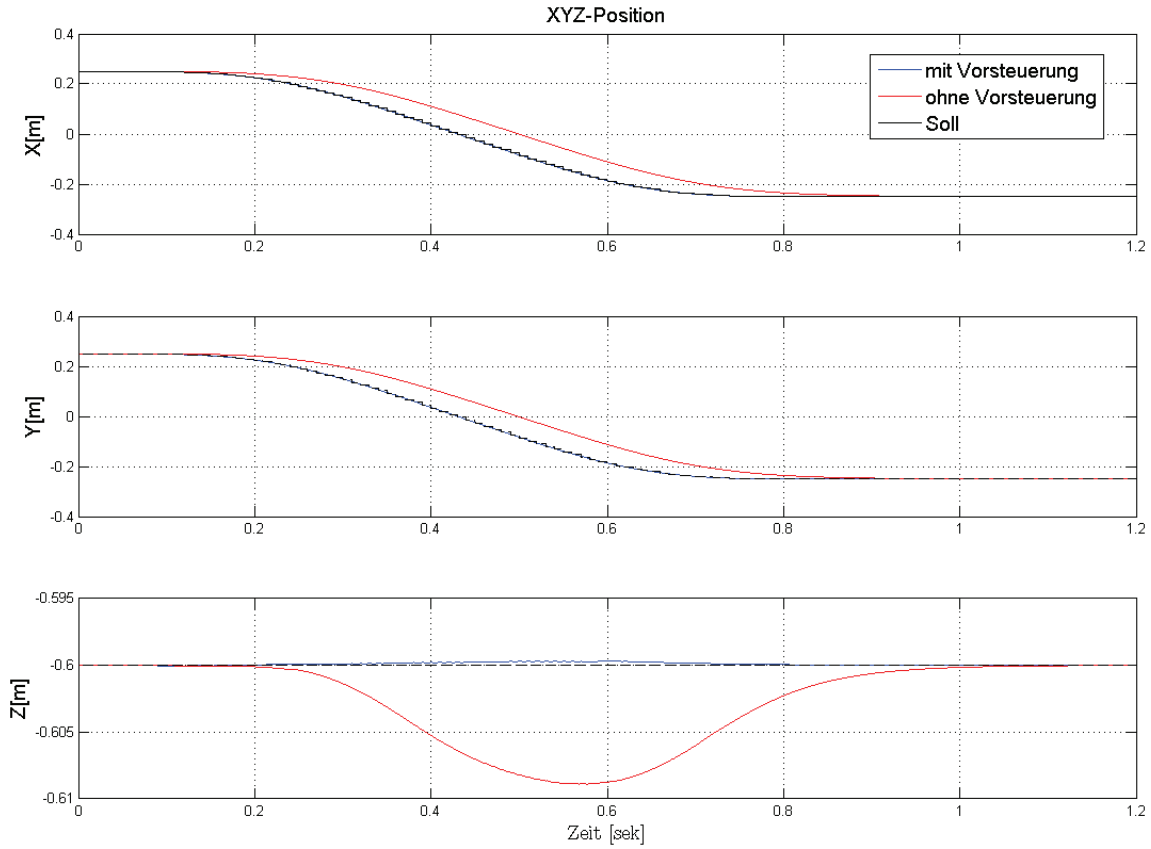


Abbildung 7.3-4: XYZ-Position der simulierten Linearfahrt

Die Auswirkungen der Beobachtungen aus Abbildung 7.3-3 auf die Linearfahrt des TCPs im kartesischen Koordinatensystem zeigen sich in Abbildung 7.3-4. Die Bewegung des vorgesteuerten Roboters gegenüber dem nicht vorgesteuerten ist nicht nur verzögert sondern auch ungleich. Besonders deutlich wird die Abweichung von der Sollbahn bei der Betrachtung der Z-Achse. Die Z-Position des TCPs soll sich zwischen Start- und Zielpunkt nicht ändern und konstant auf -0,6m bleiben. Mit der Vorsteuerung kann diese Vorgabe millimetergenau eingehalten werden. Ohne die Vorsteuerung weicht die Position des TCPs bis zu 9mm von der Sollposition ab. Wird die Maximalgeschwindigkeit erhöht, erhöhen sich auch diese Abweichungen, was in weiteren Simulationen nachgewiesen wurde.
Grund für die Abweichungen ist die Geometrie des Roboters. Bei einem Parallelkinematik-Roboter ist eine Änderung der Armwinkel nicht proportional zur Änderung der Position der unteren Arbeitsplatte.
Um sich den Fahrtverlauf besser vorstellen zu können, zeigt Abbildung 7.3-5 den dreidimensionalen Verlauf.

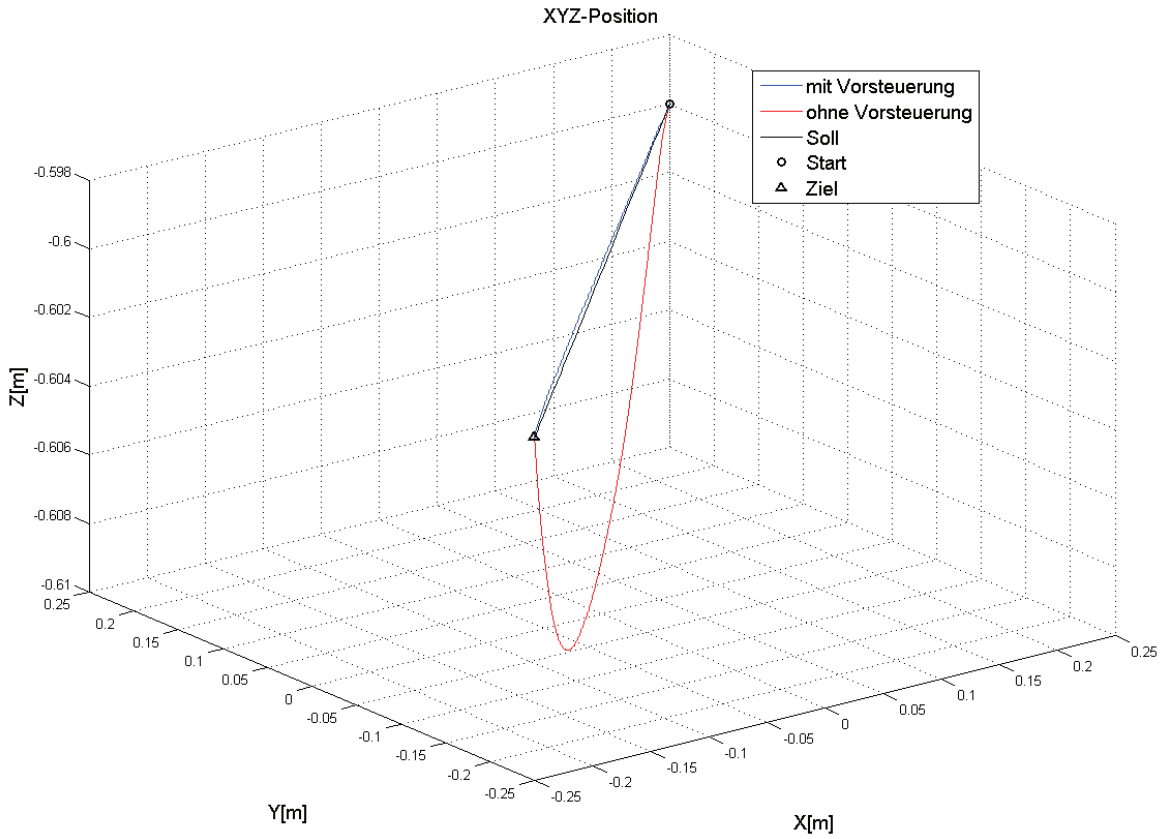


Abbildung 7.3-5:XYZ-Position der simulierten Linearfahrt in 3D

7.3.2 Kreisfahrt

Zu Demonstrationszwecken kann neben der Abfahrt einer Linearbahn auch noch das Fahren auf einer Kreisbahn eingestellt werden. Die Geschwindigkeit ist auf eine Umdrehung pro Sekunde eingestellt. Die Startposition des TCPs ist:

$$xyz_{Start} = \begin{pmatrix} 0 \\ 0{,}2 \\ -0.6 \end{pmatrix} [m]$$

Es ist zu erwarten, dass sich die Beobachtungen die bei der Linearfahrt gemacht wurden auch bei dieser Simulation zeigen.

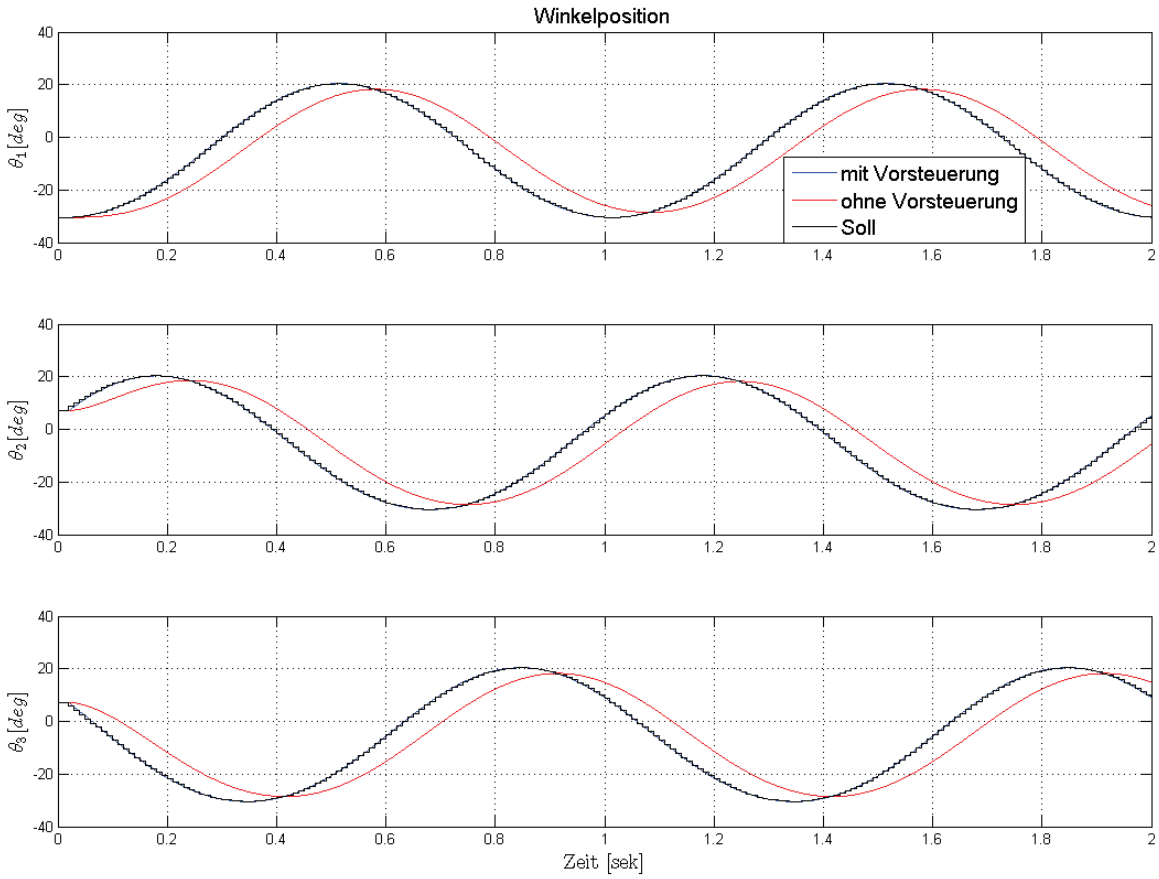


Abbildung 7.3-6:Winkelistwert der simulierten Kreisfahrt

Wie in Abbildung 7.3-6 zu sehen ist, weisen die Armwinkel einen periodischen, um 120° Grad verschobenen Verlauf auf. Auch hier eilen die Winkelistwerte des nicht vorgesteuerten Roboters hinterher. Zudem werden die Spitzenwerte nicht erreicht. Das hat zur Folge, dass die Position des TCPs auch hier nicht die gewünschte Sollbahn abfährt. In Abbildung 7.3-7 ist wieder eine deutliche Abweichung bei der Z-Achse zu erkennen, die in dieser Simulation bei ca. 5mm schwankt. Das vorgesteuerte System kann nach einer Einregelzeit die geforderte Bahn mit einer Abweichung unter einem Millimeter halten.

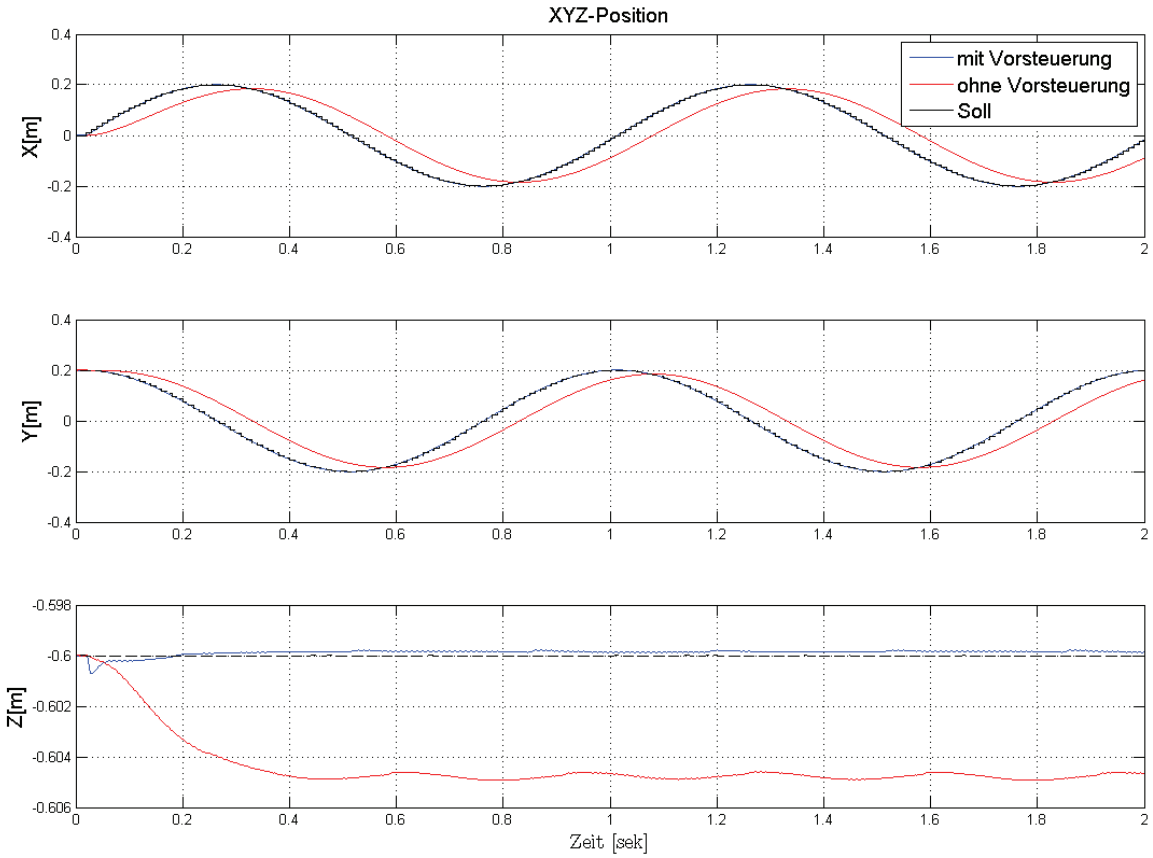


Abbildung 7.3-7:XYZ-Position der simulierten Kreisfahrt

Um sich die Abfahrt der Kreisbahn besser vorstellen zu können, zeigen Abbildung 7.3-8 und Abbildung 7.3-9 die dreidimensionale bzw. die XY-Darstellung der simulierten Fahrt. Hierbei ist deutlich zu sehen, dass sich die Abweichungen von der Sollbahn nicht auf die Z-Achse beschränken. Sie liegen in Bezug auf die X- und Y-Achse sogar bei 1,5 cm.

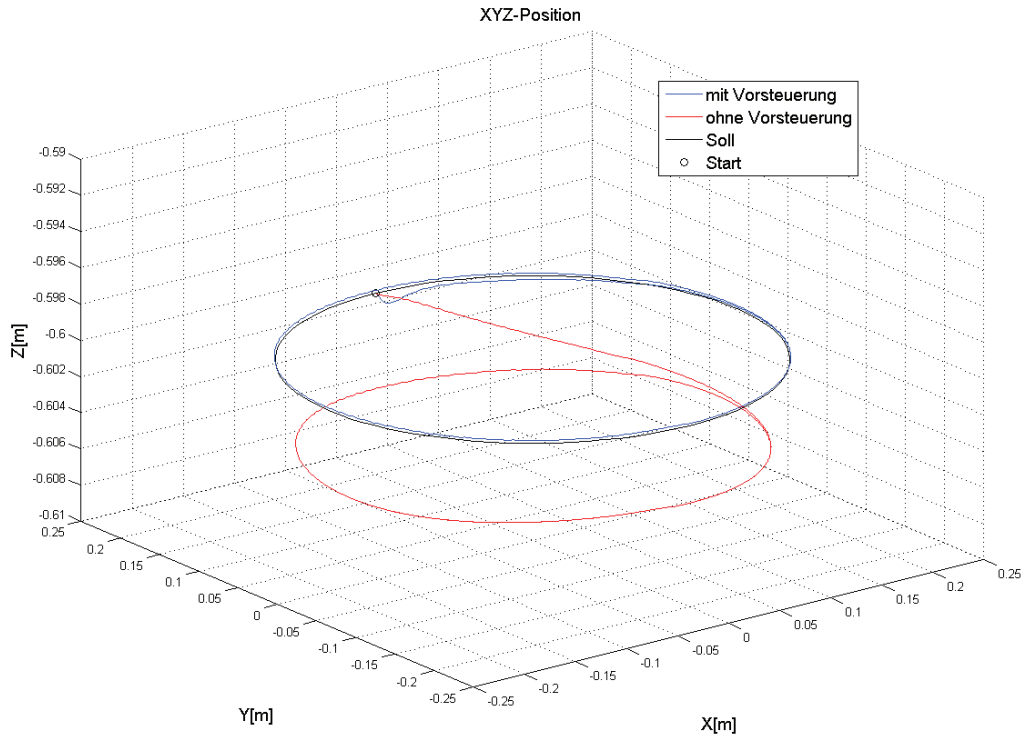


Abbildung 7.3-8:Position der simulierten Kreisfahrt in 3D

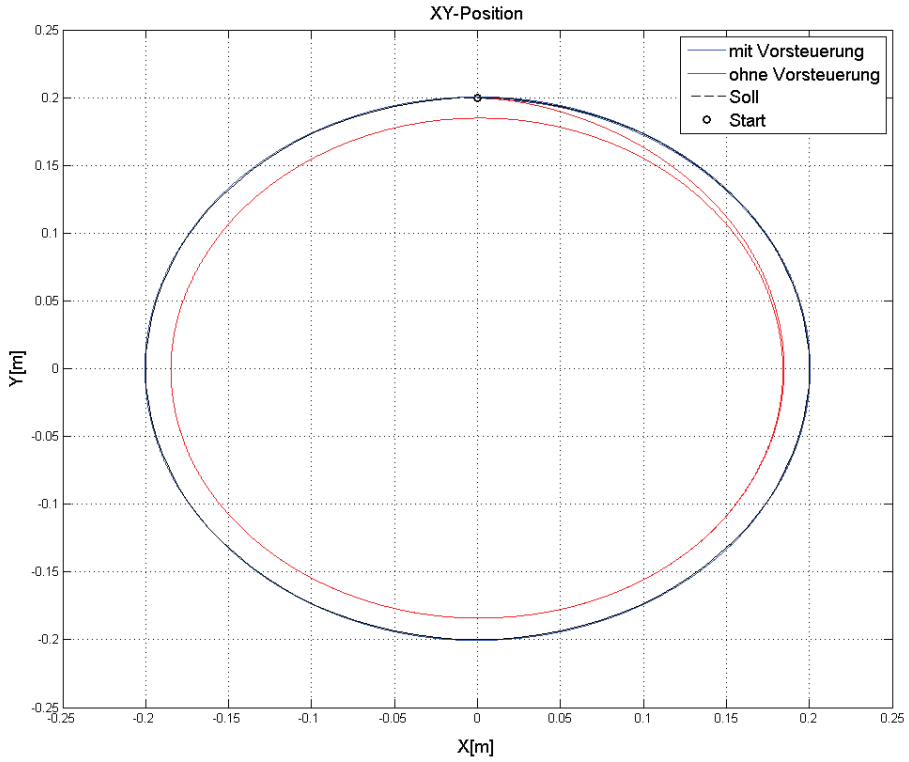


Abbildung 7.3-9:XY-Plot der simulierten Kreisfahrt

8 Projektierung STEP 7

In diesem Kapitel wird die in SIMATIC STEP 7 (V5.4 + SP5 + HF1) programmierte Steuerungsapplikation vorgestellt. Dazu wird auf die Hardware-Konfiguration, den Datenaustausch, die Lageregelung der Antriebe, die Bahninterpolation und die Sicherheitsfunktionen eingegangen.

Die einzelnen Funktionen und Funktionsbausteine werden zyklisch im OB1[24] abgearbeitet.

Zur Übersicht sind in der folgenden Tabelle alle projektierten Bausteine aufgelistet.

Name	**Beschreibung**
OB1	Zyklische Ausführung des Steuerprogrammes
OB100	Initialisierung des Steuerprogrammes
FB1	Führungsgrößengenerator
FB10	Lageregler
FC1	Sicherheitsfunktion
FC120	Kommunikation Steuerung ↔ Antriebssysteme
FC121	Reset aller Sollwerte (Aufruf im OB100)
FC122	Kommunikation Steuerung ↔ Bedienoberfläche
FC123	Lageregelung der Antriebe
DB1	Daten Steuerung ↔ Bedienoberfläche
DB2	Daten Sollwerttrajektorien
DB10	Instanz DB von FB10 des Lageregler Antrieb 1
DB11	Instanz DB von FB1 Führungsgrößengenerator
DB20	Instanz DB von FB10 des Lageregler Antrieb 2
DB30	Instanz DB von FB10 des Lageregler Antrieb 3
DB65001	Instanz DB von SFB65001
DB65002	Instanz DB von SFB65002
SFB65001	Initialisierung CCX-Schnittstelle
SFB65002	Synchroner Aufruf CCX-Programm
SFC14	Lesen der Peripherie-Eingänge
SFC15	Scheiben der Peripherie-Ausgänge

Tabelle 9:Bausteine des STEP 7 Projektes

[24] Organisationsbaustein

8.1 Hardware Konfiguration

In der Hardware-Konfiguration wird die Anbindung der einzelnen verwendeten Baugruppen projektiert. Dazu wurde zunächst der EC31 RTX F als Zentralbaugruppe ausgewählt und anschießend die CU310 PNs der Antriebssysteme an dessen PROFINET-Schnittstelle angefügt.

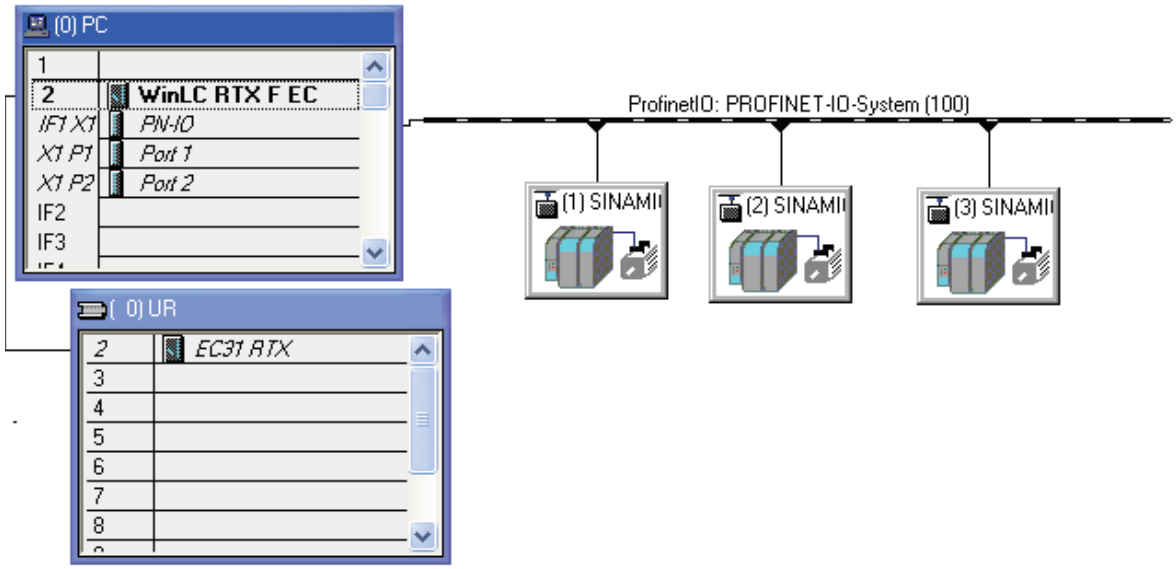


Abbildung 8.1-1:Hardware Konfiguration

Zur Konfiguration gehören die richtige Auswahl der GSD[25]-Dateien der einzelnen Baugruppen sowie die Vergabe der korrekten IP-Adressen für deren Kommunikation. Bei der Projektierung musste darauf geachtet werden, dass der Embedded Controller über zwei IP-Adressen verfügt. Eine für dessen Ethernet-Port und eine für den PROFINET-Port. Sie lauten wie folgt.

Ethernet: 192.168.2.1
PROFINET: 192.168.225.1

Die PROFINET-Adresse befindet sich im gleichen Subsystem wie auch die Adressen der CU310 PNs. Zum Datenaustausch zwischen Steuerung und den Antriebssystemen werden die gleichen Protokolle eingestellt wie schon unter 6.1.

[25] Geräte-Stammdaten-Datei

8.2 Kommunikationsbausteine

Die Kommunikation des Steuerungsprogramms mit dem Antriebssystemen und der Bedienoberfläche wurde möglichst einfach gehalten. Für den Datenaustausch wurde der DB1[26] programmiert. In ihm werden alle relevanten Daten gespeichert. Dazu gehören die Zustands- und Steuerworte der einzelnen Antriebe, Lageist- und Sollwerte, die Begrenzungen der Maximaldrehzahl und der Lageoffsets der Antriebe. Die Kommunikationsbausteine schreiben und lesen in diesem DB. Die Bausteine selbst sind in AWL[27] programmiert.

8.2.1 FC120 Steuerung zum Antriebsystem

Im FC120[28] wird der Datenaustausch zwischen Steuerung und den Antrieben realisiert. Dazu dienen die Standartfunktionen SFC14 und SFC15. Mit ihnen ist es möglich, Daten von Baugruppen, die über PROFINET oder PROFIBUS mit der Steuerung ausgetauscht werden, zu lesen und zu schreiben. Der Vorteil dieser Standartfunktionen ist es, dass es mit ihnen möglich ist, größere Datenmengen zusammenhängend zu bearbeiten. Mit den normalen Ladebefehlen ist dies nur mit bis zu vier Byte möglich. Grund hierfür ist die Größe der Akkumulator-Register.
Die Funktionen benötigen dazu Informationen über die Anfangsadresse des Eingangs bzw. des Ausgangs, die Länge in Byte sowie die Adresse in die geschrieben oder aus der gelesen werden soll.
Da der FC120 zyklisch im OB1 aufgerufen wird, können die Antriebe einfach über den DB1 gesteuert und beobachtet werden.

8.2.2 FC122 Steuerung zur Bedienoberfläche

Der FC122 dient zum Datenaustausch zwischen der Steuerung und der in C++ programmierten Bedienoberfläche. Dank der SMX-Schnittstelle kann dies über einfache Ladebefehle realisiert werden. Der FC ist in vier Netzwerke unterteilt.

[26] Datenbaustein

[27] Anweisungsliste

[28] Funktion

Netzwerk 1-3:

In jedem dieser Netzwerke ist der Datenaustausch für einen der drei Antriebe programmiert. Die Daten werden aus dem DB1 gelesen und geschrieben. Dazu gehören:

- Statuswort
- Steuerwort
- Lageistwert
- Lagesollwert
- Maximalgeschwindigkeit

Außerdem wird in dem entsprechenden Netzwerk das Setzten des Lageoffsets realisiert. Setzt der Bediener den Winkelnullwert des entsprechenden Roboterarmes neu, wird der aktuelle Lageistwert in den Offset geschrieben. Der Offset wird zum Lageistwert dazu addiert. Somit kann der Roboter, wenn er bei ausgeschalteter Anlage bewegt wurde, neu kalibriert werden.

Netzwerk 4:

In diesem Netzwerk ist die Kommunikation zur Steuerung des Roboters realisiert. Dazu dient zum einem ein Steuerwort im DB1, dass die Bewegung des Roboters beschreibt. Zum anderen werden hier Informationen zur Zielposition und Geschwindigkeit von der Bedienoberfläche gelesen. Das Steuerwort ist wie folgt definiert:

Adresse	Name	Beschreibung
72.0	ResetLageS1	Offset neu setzen Servo 1
72.1	ResetLageS2	Offset neu setzen Servo 2
72.2	ResetLageS3	Offset neu setzen Servo 3
72.3	Aus S1	Servo 1 einschalten
72.4	Aus S2	Servo 2 einschalten
72.5	Aus S3	Servo 3 einschalten
72.6	Start_Fahrt	Roboterfahrt starten
72.7	Stop_Fahrt	Roboterfahrt stoppen
73.0	Modus	Linearfahrt/Kreisfahrt
73.1	Regelung_Kalib_Auto	Kalibrierung = False, Roboterfahrt = True
73.2	StopServos	Antriebe anhalten
73.3	Sauger_Ein	Sauger Ein/Aus

73.4	Reserve	
73.5	Reserve	
73.6	Reserve	
73.7	Reserve	

Tabelle 10:Steuerwort des Roboters

Neben diesem Steuerwort werden noch weitere Daten ausgetauscht.

- X-Sollwert des Roboters
- Y-Sollwert des Roboters
- Z-Sollwert des Roboters
- Geschwindigkeit bei linearer Fahrt
- Geschwindigkeit bei der Kreisfahrt
- Winkelsollwerte zur Aufzeichnung
- Winkelistwerte zur Aufzeichnung

Die Aufzeichnung der Winkelsoll- und Istwerte geschieht innerhalb der Bedienoberfläche und wird in Abschnitt 9.3.4 erklärt.

8.3 Lageregelung

Die Lageregelung der einzelnen Antriebe geschieht innerhalb der Steuerapplikation und wurde mittels FUP[29] ebenfalls in STEP 7 programmiert. Eine Lageregelung ist zwar standardmäßig auch mit STARTER direkt im Antriebssystem möglich. Dies hat aber den großen Nachteil, dass ein neuer Lagesollwert erst vorgegeben werden kann, wenn der vorherige erreicht wurde. Wird ein neuer Sollwert vorgegeben, wenn der Antrieb noch in Bewegung ist, stoppt der Antrieb und geht in einen Fehlerzustand. Die Lageregelung wurde in dem FB10[30] programmiert. Der Vorteil eines Funktionsbausteines ist hierbei, dass dieser mehrfach instanziiert werden kann. Somit kann für jeden Antrieb eine Instanz des FBs eingefügt werden, um dessen Lage zu regeln.

Der Baustein hat als Eingänge den Lagesoll- und Istwert sowie eine Drehzahlbegrenzung in Prozent bezogen auf die Nenndrehzahl. Er gibt die Solldrehzahl des entsprechenden

[29] Funktionsplan

[30] Funktionsbaustein

Antriebes aus. Der Lageregler ist als P-Regler ausgelegt und unterliegt der Zykluszeit der Steuerung. Ein P-Regler ist deshalb ausreichend, da die Strecke selbst schon I-Verhalten aufweist. Auf den programmiertechnischen Aufbau des FBs wird an dieser Stelle nicht eingegangen, es wird jedoch der prinzipielle Ablauf erklärt.

1. Ermittlung der Regeldifferenz
2. Ermittlung der maximalen positiven und negativen Drehzahlgrenze
3. Die Regeldifferenz wird mit dem Verstärkungsfaktor K_v des P-Reglers multipliziert
4. Der Drehzahlsollwert wird in Prozent bezogen auf die Nenndrehzahl umgerechnet und mit 16384 multipliziert.

$$n_{soll} = \frac{e[LU] \cdot 16384}{9000[LU] \cdot 6000} \quad \textbf{(8-1)}$$

mit

e : $Regeldifferenz\ (\theta_{ist} - \theta_{soll})$

n_{soll} : $Solldrehzahl$

5. Ist der Drehzahlsollwert kleiner oder größer als die positive oder negative Drehzahlgrenze, wird der Sollwert auf diese begrenzt.
6. Aufgabe der Solldrehzahl

Wie schon bei der Projektierung der Antriebsysteme besprochen entspricht ein Wert von 16384 Hundertprozent bezogen auf die Nenndrehzahl $n_n = 6000min^{-1}$.
Die Ermittlung des optimalen Verstärkungsfaktors K_v des Lagereglers wurde empirisch durchgeführt. Dazu wurde die Sprungantwort des Lageregelkreises bei verschiedenen P-Verstärkungen aufgezeichnet und miteinander verglichen. Abbildung 8.3-1 zeigt die aufgezeichneten Sprungantworten.

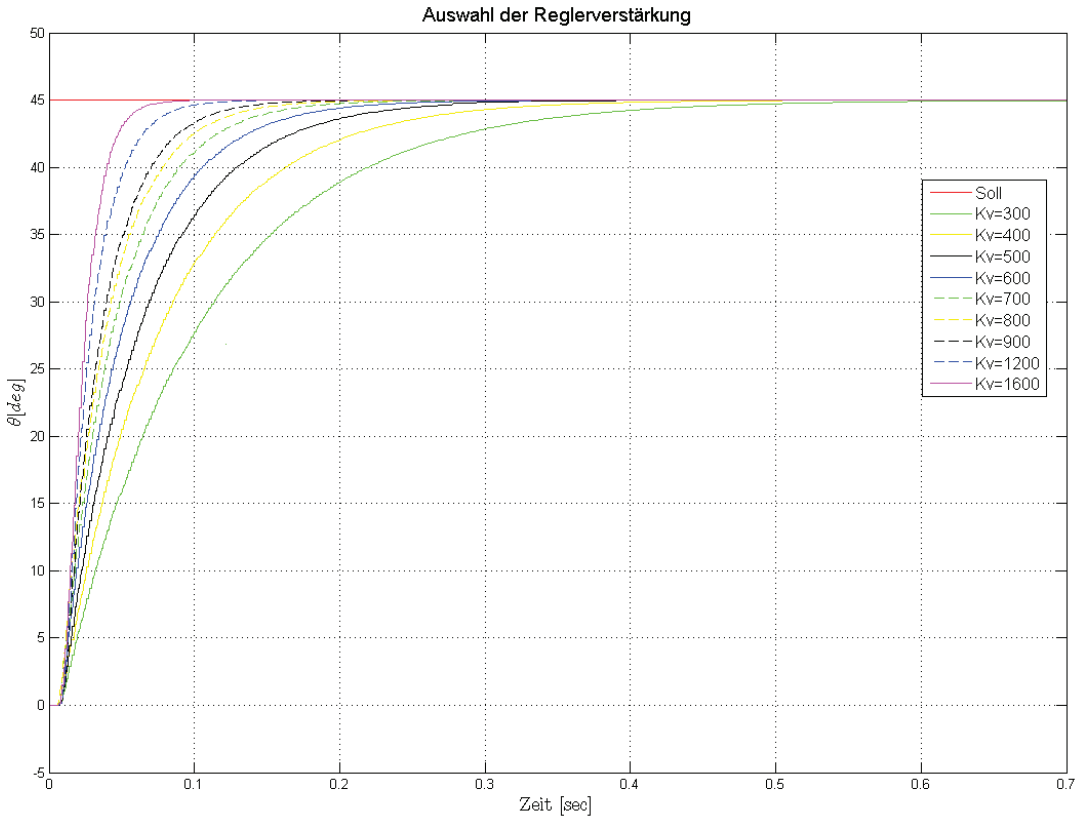


Abbildung 8.3-1:Ermittlung der Reglerverstärkung K_v

Um eine hohe Dynamik des Lageregelkreises zu gewährleisten, die Mechanik des Roboters aber nicht unnötig zu belasten, wurde die Reglerverstärkung mit $K_v = 700$ gewählt.

Die Instanzen der Lageregler werden im FC123 aus dem OB1 heraus aufgerufen. Dabei ist jeder Regler zweifach aufgeführt. Einmal für die Kalibrierung der Antriebe, bei der der Lageistwert direkt vom Benutzer vorgegeben wird. Zum anderen für die Fahrt nach den Sollwerttrajektorien, wobei die Sollwerte vom Führungsgrößengenerator vorgegeben werden. Dieses Vorgehen ist notwendig, da die Sollwerte je nach Betriebsart von einem anderen DB stammen. Für die Kalibrierung werden die Lagesollwerte aus dem DB1 benötigt, die dort über die SMX-Schnittstelle von der Bedienoberfläche eingetragen werden. Die Sollwerte des Führungsgrößengenerators werden im DB2 eingetragen. Die Umschaltung erfolgt über die Enable-Eingänge der Instanzen des FB10s. Dieser ist mit dem Steuerbit 73.1 „Regelung_Kalib_Auto“ des DB1 verschaltet.

8.4 Führungsgrößengenerator mittels CCX

Der Führungsgrößengenerator ist mittels der schon erklärten CCX-Schnittstelle umgesetzt. Dazu ruft die Steuerungsapplikation eine in C++ programmierte DLL-Datei auf. In dieser finden die Berechnungen der Bahninterpolation für die translatorischen Trajektorien und dessen anschließende Transformation in die rotatorischen Trajektorien statt.

8.4.1 Umsetzung in STEP 7

Der Aufruf des CCX-Programmes aus der Steuerapplikation erfolgt mittels des FB1. Er wird sowohl zur Initialisierung der Anwendung im OB100 als auch zyklisch im OB1 aufgerufen. Der Baustein ist in FUP programmiert und besitzt folgende Ein- und Ausgänge.

Eingänge:

- INIT (bool), True = Initialisierung, False = Aufruf
- XYZ-Sollposition (Int)
- Maximalgeschwindigkeit (Int)
- Modus (Bool), True = Linearfahrt False = Kreisfahrt
- Start (Bool), Startet die Fahrt
- Stop (Bool), Stoppt die Fahrt
- Lageistwert der Antriebe (Dint)
- Eingestellter Offset der Lageistwerte (Dint)

Ausgänge:

- Lagesollwerte (Dint)
- Drehzahlsollwerte (Int)
- Drehmomentsollwerte (Int)
- Statuswort (Word) = 0 wenn der Aufruf erfolgreich war

Der Baustein FB1 arbeitet nach Aufruf folgendermaßen.

Netzwerk 1:

Wird der Baustein mit dem Eingang „INIT“ = True aufgerufen, wird der SFB65001 ausgeführt. Der Systemfunktionsbaustein lädt und initialisiert die DLL-Datei der CCX-Applikation unter Angabe des Pfads und Namens.

Netzwerk 2-3:

Sobald der Bediener eine Fahrt über den Eingang „Start“ gestartet hat, wird ein Timer T1 gestartet. Es handelt sich hierbei um einen „S_IMPULST“ Timer, der beginnend mit 9,99 Sekunden in 10 Millisekunden Schritten rückwärts zählt. Eine kleinere Zeitauflösung ist in STEP 7 leider nicht möglich. Es ist ebenfalls nicht möglich, eine größere Zeit als 9,99 Sekunden herunter zählen zu lassen, da der Timer in diesem Fall mit einer größeren Schrittweite zählen würde. Dies ist auch nicht notwendig, da eine Fahrt des Roboters im Normalfall nicht länger als 1-2 Sekunden dauert. Setzt der Benutzer den Eingang „Stop“, wird der Timer zurückgesetzt. Der Zeitwert des Timers wird in die Eingangsstruktur der CCX-Applikation geschrieben.

Netzwerk 4-7:

Bei einer positiven Flanke an dem Eingang „Start“ werden die Sollwerte für die Bahninterpolation, die Lageistwerte der Antriebe und die benötigten Geschwindigkeitsvorgaben in die Eingangsstruktur für die CCX-Applikation geschrieben. Dies geschieht nur beim Start der Roboterfahrt, da der Führungsgrößengenerator ansonsten eine falsche Bahn interpolieren würde.

Netzwerk 9:

In diesem Netzwerk wird der boolsche Eingang „Modus“ in den Datentyp „Word“ umgewandelt. Dies ist notwendig, da der SFB65002 die Vorgabe des auszuführenden Subkommandos in diesem Datentyp benötigt.

Netzwerk10:

Ist der Eingang „INIT“ nicht gesetzt und der Eingang „Start“ ist gesetzt, wird der SFB65002 ausgeführt. Der Baustein führt die in der DLL-Datei gespeicherte CCX-Applikation zur Zykluszeit der Steuerung aus. Dazu wird dem Baustein die Information über das auszuführende Subkommando mittels des Eingangs „Command“ übergeben. Außerdem werden Pointer auf die Ein- und Ausgangsstrukturen übergeben. Diese sind wie folgt definiert.

Eingangsstruktur „ODK_In“:

- XYZ-Sollpositionen (Int)
- Maximalgeschwindigkeiten für Linear- und Kreisfahrt (Int)
- Lageistwert zum Startzeitpunkt (Dint)
- Zeitwert des Timer-Bausteines (Int)

Ausgangsstruktur „ODK_Out“:

- Lagesollwerte (Dint)
- Drehzahlsollwerte (Int)
- Drehmomentsollwerte (Int)

Nachdem das Programm ausgeführt wurde, werden dessen Ergebnisse in die Ausgangsstruktur geschrieben.

Netzwerk 11-16:
In diesen Netzwerken werden, wenn „Start“ aktiv ist, die Ergebnisse des Führungsgrößengenerators in die Ausgänge des FBs geschrieben. Andernfalls werden die derzeitigen Lageistwerte und null für die Drehzahlen und Drehmomente in die Ausgänge geschrieben.

Netzwerk 17-19:
Um sicherzustellen, dass es zu keinen mechanischen Schäden kommt, wenn der Roboter aus irgendwelchen Gründen seinen Arbeitsbereich verlässt, sind in diesen Netzwerken Sicherheitsfunktionen implementiert. Sobald einer der Roboterarme einen Lageistwert $\geq \pm 85°$ hat, werden automatisch die Vorsteuersollwerte für Drehzahl und Drehmoment auf null gesetzt.

8.4.2 Umsetzung in C++

Zur Realisierung des Führungsgrößengenerators wurde in C++ eigens dafür eine Klasse programmiert. Sie trägt den Namen „kinematik“. In ihr sind alle Attribute und Methoden enthalten, die zur Berechnung der Solltrajektorien für die einzelnen Antriebe notwendig sind. Der Vorteil dieser objektorientierten Programmiertechnik ist, dass somit die Implementierung des Führungsgrößengenerators in die CCX-Applikation einfacher und übersichtlicher wird. Der Quellcode der Klasse ist gut kommentiert und unter Zuhilfenahme der Kapitel 4 und 5 verständlich. Somit muss an dieser Stelle nur auf dessen Struktur und die Funktion der einzelnen Methoden eingegangen werden.
Die Attribute der Klasse „kinematik“ sind die gleichen wie die Variablen aus den Kapiteln 4 und 5, in denen die mathematischen Grundlagen des Führungsgrößengenerators bereits vorgestellt wurden. Diese sind lediglich um einige Hilfsvariablen zum Umrechnen von Rad- und Gradmaß erweitert.
Folgende Methoden wurden programmiert:

Get- und Set-Methoden:
Zum Lesen und Schreiben der privaten Attribute wurden verschiedene Get- und Set-Methoden definiert. Mit ihnen ist es möglich, Soll- und Istwerte zu setzen und die berechneten Werte für Lage-, Drehzahl und Drehmomentsollwerte auszulesen.

Inverskinematik(double x, double y, double z):
Diese Methode berechnet aus den Istwerten der Position des TCPs die entsprechenden Winkel der Oberarme. Die Vorgabe der XYZ-Position wird in Metern gemacht. Die berechneten Winkel werden in die entsprechenden privaten Attribute geschrieben.

forwardkinematik(double theta1, double theta2, double theta3):
Innerhalb dieser Methode wird aus den Winkelistwerten der Roboteroberarme die Position des TCPs berechnet. Die Winkel werden im Gradmaß vorgegeben.

linBahn(double t_ist, double x, double y, double z,double v_max):
Zur Interpolation einer zeitabhängigen linearen Fahrt zwischen zwei Punkten innerhalb des Arbeitsbereiches des Roboters wurde diese Methode programmiert. Der Funktion werden dafür die Sollposition des TCPs und dessen maximale Geschwindigkeit vorgegeben. Außerdem muss die Information über die aktuelle Laufzeit der Fahrt vorgegeben werden. Innerhalb der Methode wird aus diesen Informationen und der zuvor eingegeben Istposition des TCPs die Werte der translatorischen Bewegung für Position, Geschwindigkeit und Beschleunigung berechnet.

kreisBahn(double t_ist, double periode):
Zur Demonstration der Roboterdynamik wird in dieser Methode eine Kreisbahn interpoliert. Dazu wird eine Kreisbewegung mit gleichbleibender Höhe berechnet. Der Radius bezieht sich auf den aktuellen Abstand des TCPs vom Koordinatenursprung bezogen auf die X- und Y-Achse. Die Geschwindigkeit wird in Form der Periodendauer dieser Kreisbewegung vorgegeben.

transformation():
Diese Methode wird innerhalb der Interpolationsmethoden „linBahn()“ und „kreisBahn()“ nach der Berechnung der translatorischen Bewegung aufgerufen. Sie berechnet aus dessen Ergebnissen die Lage-, Drehzahl- und Drehmomentsollwerte für die einzelnen Antriebe nach Abschnitt 4.2.

Die Klasse wurde nach Fertigstellung und Überprüfung der Korrektheit in das CCX-Programm eingefügt. Die CCX-Applikation wurde mittels des „WinAc ODK Application Wizzard “ erstellt. Die dadurch erstellte Vorlage verfügt über zwei Subkommandos. Eins für eine Linearfahrt und eins für die Bewegung auf einer Kreisbahn. Da die Programmvorlage selbst schon lauffähig ist und über alle benötigten Klassen und Methoden verfügt, muss nur noch der Inhalt der beiden Subkommandos programmiert werden. Wie unter 8.4.1 beschrieben, wird das entsprechende Subkommando aus der Steuerapplikation aufgerufen.

Der prinzipielle Ablauf innerhalb der beiden Subkommandos ist wie folgt.

1. Die Eingangsstruktur der vorgegebenen Ist- und Sollwert wird ausgelesen
2. Die Integer-Sollwerte werden in die benötigten Doublewerte umgewandelt
3. Winkelistwerte und die entsprechenden Sollwerte werden gesetzt
4. Der Führungsgrößengerator wird mit der aus STEP 7 übergebenen Zeit ausgeführt
5. Normierung der Sollwerte für die Antriebssysteme unter Berücksichtigung der Getriebeübersetzung
6. Die Sollwerte werden in die, an die Steuerungsapplikation zurückgegebene Ausgangsstruktur, geschrieben

Der programmierte Quellcode ist kommentiert und kann in der Klasse „Bahninterpolationfunc.cpp“ innerhalb des CCX-Projektes „Bahninterpolation“ angeschaut werden. Die aus dem Projekt generierte DLL-Datei „Bahninterpolation.dll“ wurde unter dem Pfad „C:\WINDOWS\system32“ auf dem Embedded PC hinterlegt:

8.5 Sicherheitsfunktion

Zur Sicherstellung, dass es zu keinen mechanischen Schäden am Roboter kommt, wird als letzte Funktion im OB1 der FC1 aufgerufen. Dazu überprüft der FC, ob sich der Roboter in seinem Arbeitsbereich befindet. Ist dies nicht der Fall, werden die Antriebe sofort gestoppt. Wie schon unter 8.4.1 erklärt wird dazu überprüft, ob einer der Roboterarme einen Lageistwert von $\geq \pm 85°$ aufweist. Den Antrieben wird dann eine Solldr ehzahl von null vorgegeben, um diese zu stoppen. Würde man sie einfach ausschalten, würden sie sich durch den Schwung und die Massenträgheit der Roboterarme und der Arbeitsplatte unter Umständen noch weiter, auf einer nicht vorhersehbaren Bahn, bewegen. Neben der Überprüfung der Lageistwerte der Antriebe wird an dieser Stelle ebenfalls überprüft, ob der Bediener den AUS-Button innerhalb der Bedienoberfläche betätigt hat. In diesem Fall ist das Steuerbit 73.2 „StopServos“ aktiv und die Antriebe werden ebenfalls gestoppt.

9 Bedienoberfläche

Um den Roboter Bedienen und Beobachten zu können, wurde eine Bedienoberfläche programmiert. Die Bedienapplikation wurde mittels Visual Studio 6.0 erstellt und in C++, unter Verwendung von MFC programmiert. Sie wird aus dem Betriebssystem des S7-mEC heraus gestartet und nutzt die schon beschriebene SMX-Schnittstelle, um auf die Prozessvariablen der Steuerungsapplikation zuzugreifen.

In diesem Kapitel wird nicht im Detail auf den programmierten Quellcode eingegangen. Es wird vorausgesetzt, dass Entwickler, die zu einem späteren Zeitpunkt an der Bedienoberfläche arbeiten, die Grundlagen der objektorientierten Programmierung unter C++ beherrschen. Unter Zuhilfenahme des kommentierten Quellcodes, der Dokumentation in Form dieser Masterthesis und des ODK Manuals [2], ist das Verständnis gewährleistet.

9.1 Grundlegendes

Die Bedienoberfläche setzt sich im Wesentlichen aus zwei Teilen zusammen. Zum einen wurde ein einfacher Handbetrieb in einem separaten Fenster Namens „Servos Kalibrieren“ programmiert. Mit diesem ist es möglich, die einzelnen Antriebe unabhängig von den anderen zu bewegen und den Nullpunkt der Armwinkel zu bestimmen.
Das andere Bedienfenster „Delta Roboter“ realisiert den Automatikbetrieb des Roboters. In diesem Fenster sind alle geforderten Betriebsarten umgesetzt.

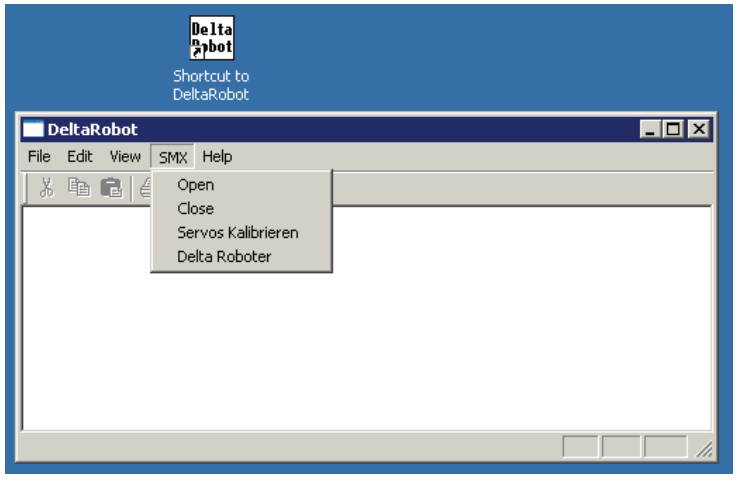


Abbildung 9.1-1:Start der Bedienoberfläche

Die Bedienoberfläche wird über eine entsprechende Verknüpfung auf dem Desktop des Embedded PCs gestartet. Es öffnet sich daraufhin das Standard-Startfenster der SMX-

Applikationen. Hier ist es möglich die SMX-Schnittstelle zu initialisieren oder zu schließen und die bereits angesprochenen Bedienfenster zu öffnen.

9.1.1 MFC

MFC ist eine Sammlung von Klassenbibliotheken, die von Microsoft zur Programmierung grafischer Benutzeroberflächen zur Verfügung gestellt werden. Mit Visual Studio kann das Layout eines Bedienfensters in einem entsprechenden Editor festgelegt werden. Dazu können einzelne Bedienelemente per Maus in das Fenster eingefügt werden. Für jedes dieser Bedienelemente können eine oder mehrere Methoden implementiert werden, die das Verhalten und Aussehen des Elementes definieren. Die Bedienfenster sind von der MFC-Klasse „Dialog“ abgeleitet.

9.1.2 Remote-Desktop Protokoll

Um auf das Betriebssystem des S7-mECs und damit auf die Bedienoberfläche zugreifen zu können, wird das Remote-Desktop-Protokoll eingesetzt. Mit diesem ist es möglich, den Roboter zu steuern, ohne direkt Bildschirm, Tastatur und Maus an dem Embedded PC anzuschließen. Somit kann der Roboter auch bei geschlossenem Schaltschrank bedient werden. Die Einwahl erfolgt über die IP-Adresse des Ethernet-Ports des S7-mECs. Benutzername und Passwort entsprechen den Default-Einstellungen des Embedded PCs und wurden nicht verändert.

IP-Adresse:	**192.168.2.1**
Benutzername:	**Administrator**
Passwort:	**admin**

9.1.3 Threading

Um das Problem zu lösen, den Roboter zu steuern und gleichzeitig die Aktualisierung der Prozessvariablen zum Beobachten zu gewährleisten, war es notwendig, diese beiden Programmteile voneinander zu entkoppeln. Dies wurde mittels Threads, die zyklisch innerhalb des Bedienfensters abgearbeitet werden, umgesetzt.
Wird eines der beiden Bedienfenster geöffnet, wird in diesem ein Thread gestartet, der eine Methode zum Auslesen der Prozessvariablen und deren Visualisierung ausführt. Die Methode beinhaltet zu diesem Zweck eine Schleife. Nach jedem Durchlauf der Schleifen, wartet diese 500ms, bevor sie das nächste Mal durchlaufen wird. Somit wird sicherge-

stellt, dass die Applikation die CPU nicht überlastet. Innerhalb der Schleife werden zunächst die einzelnen benötigten Prozessvariablen mittels der entsprechenden SMX-Methoden ausgelesen. In Abhängigkeit der Variablen werden im Anschluss die dazugehörigen Visualisierungselemente aktualisiert. Die Threads werden beendet, sobald das entsprechende Bedienfenster geschlossen wird.

9.2 Kalibrierung der Antriebe

Um jeden der drei Antriebe einzeln einzustellen und somit eine Kalibrierung der Armwinkel vornehmen zu können, wurde dafür ein eigenes Bedienfenster implementiert. Dazu ist das Fenster in drei Hauptteile unterteilt, in denen der Bediener jeden Antrieb separat steuern kann. Abbildung 9.2-1 zeigt das entsprechende Bedienfenster.

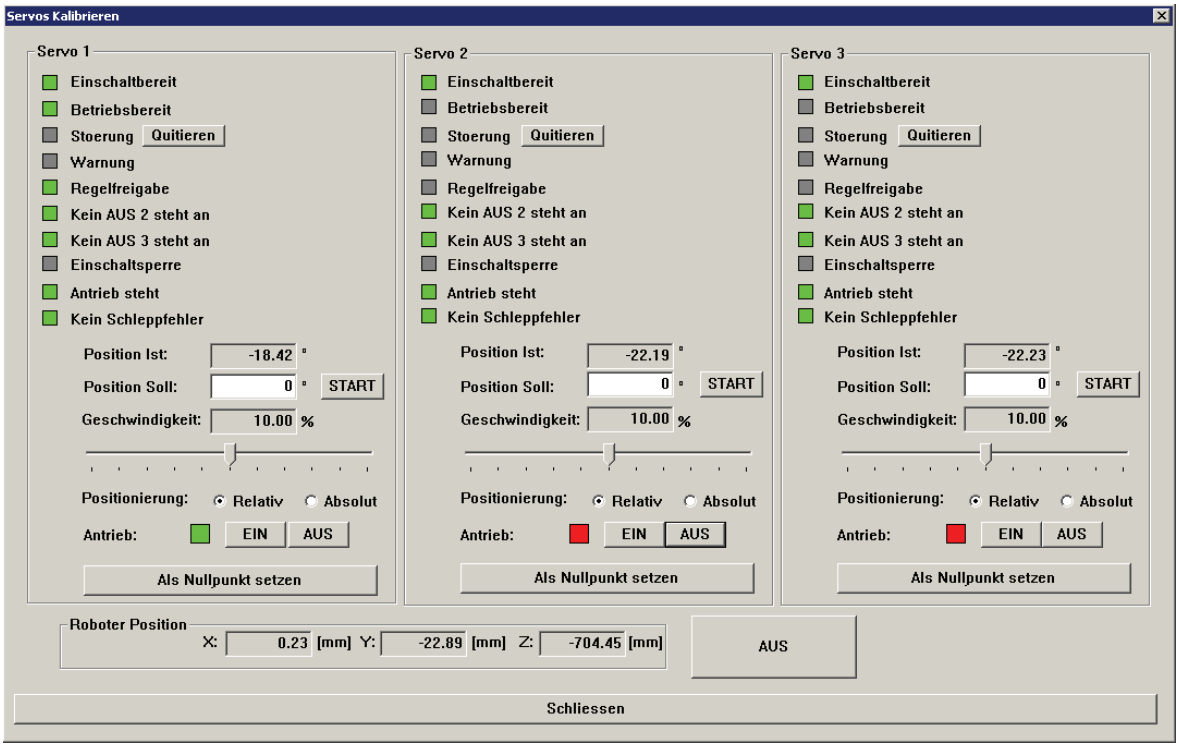


Abbildung 9.2-1:Bedienfenster: Kalibrierung

Für jeden Antrieb sind Statusanzeigen vorhanden, mit denen der Bediener den Zustand des Antriebssystems feststellen kann. Außerdem ist es möglich, den Antrieb Ein- und Auszuschalten und eventuelle Störungen zu quittieren. Die Anzeige des Status der einzelnen Antriebe beinhaltet hierbei mehr Informationen als für die Bedienung notwendig sind. Grund hierfür ist, dass die Anzeige noch innerhalb der Machbarkeitsanalyse der Automatisierungsaufgabe programmiert wurde und seitdem nicht verändert wurde. Für den Bediener sind lediglich die oberen Anzeigen für Störungen, Einschaltbereitschaft und Betriebsbereitschaft wichtig.

Die Lageregelung kann absolut bezogen auf den Winkelnullpunkt oder relativ zum Lageistwert des Antriebes eingestellt werden. Über die zu sehenden Slider kann die Begrenzung der Maximaldrehzahl in Prozent, bezogen auf die Nenndrehzahl $6000\ min^{-1}$, eingestellt werden. Sie ist auf 20% begrenzt, da eine höhere Geschwindigkeit zur Kalibrierung der Antriebe nicht notwendig ist. Über die Start-Buttons wird bei eingeschaltetem Antrieb dessen Lageregelung gestartet. Um den aktuellen Lageistwert als Winkelnullwert festzulegen, wird der Button „Als Nullpunkt setzen“ betätigt.
Der Sollwert der relativen Lageregelung ist auf ±10° begrenzt, um einer falschen Bedienung und somit dem Risiko eines mechanischen Schadens am Roboter vorzubeugen. Bei der absoluten Lageregelung ist der Sollwert auf ±45° begrenzt.
In diesem Fenster wird zur Übersicht außerdem die XYZ-Position des TCPs angezeigt. Dazu ist in einer eigenen Methode die Forward-Kinematik implementiert.
Um alle Antriebe gleichzeitig stoppen zu können, wurde hierfür ein eigener Button „AUS“ eingefügt. Wird dieser betätigt, werden sofort alle Antriebe ausgeschaltet. Die Antriebe werden ebenfalls ausgeschaltet, sobald das Bedienfenster geschlossen wird.

Trotz der implementierten Sollwertbegrenzungen kann bei fehlerhafter Bedienung des Bedienfensters ein mechanischer Schaden am Roboter nicht ausgeschlossen werden. Grund hierfür ist, dass der Winkelnullwert vom Bediener gesetzt wird und somit auch falsch eingestellt werden kann. Aus diesem Grund ist bei der Kalibrierung der Antriebe äußerste Vorsicht geboten. Es hat sich in der Praxis bewährt die Antriebe gar nicht erst einzuschalten, sondern den Arm per Hand in die gewünschte Nullposition zu bringen und dann den Nullpunkt zu setzen.

9.3 Automatikbetrieb

Im Automatikbetrieb können die Fahrten des Roboters in verschiedenen Betriebsarten eingestellt werden. Dazu gehört der Handbetrieb, in dem vom Benutzer einzelne Fahrten eingestellt und gestartet werden können, die externe Steuerung des Roboters über TCP/IP oder UDP/IP oder der Start einer Abfolge von Fahrten zu Demonstrationszwecken. Dabei werden je nach gewählter Betriebsart die entsprechenden Bedienelemente freigeschaltet und alle anderen gesperrt. Ebenfalls ist auch hier die aktuelle XYZ-Position des TCPs visualisiert. Auch die schon im Bedienfenster „Servos Kalibrieren“ vorgestellten Statusanzeigen der Antriebe, dessen Ein- und Ausschalten und die Störungsquittierung sind in diesem Bedienfenster implementiert.
Vor dem Start einer der drei Betriebsarten müssen die Antriebe über den entsprechenden Button eingeschaltet werden.

In der folgenden Abbildung ist das Bedienfester „Delta Robot“ des Automatikbetriebes zu sehen. Die verschiedenen Betriebsarten werden in den folgenden Abschnitten vorgestellt.

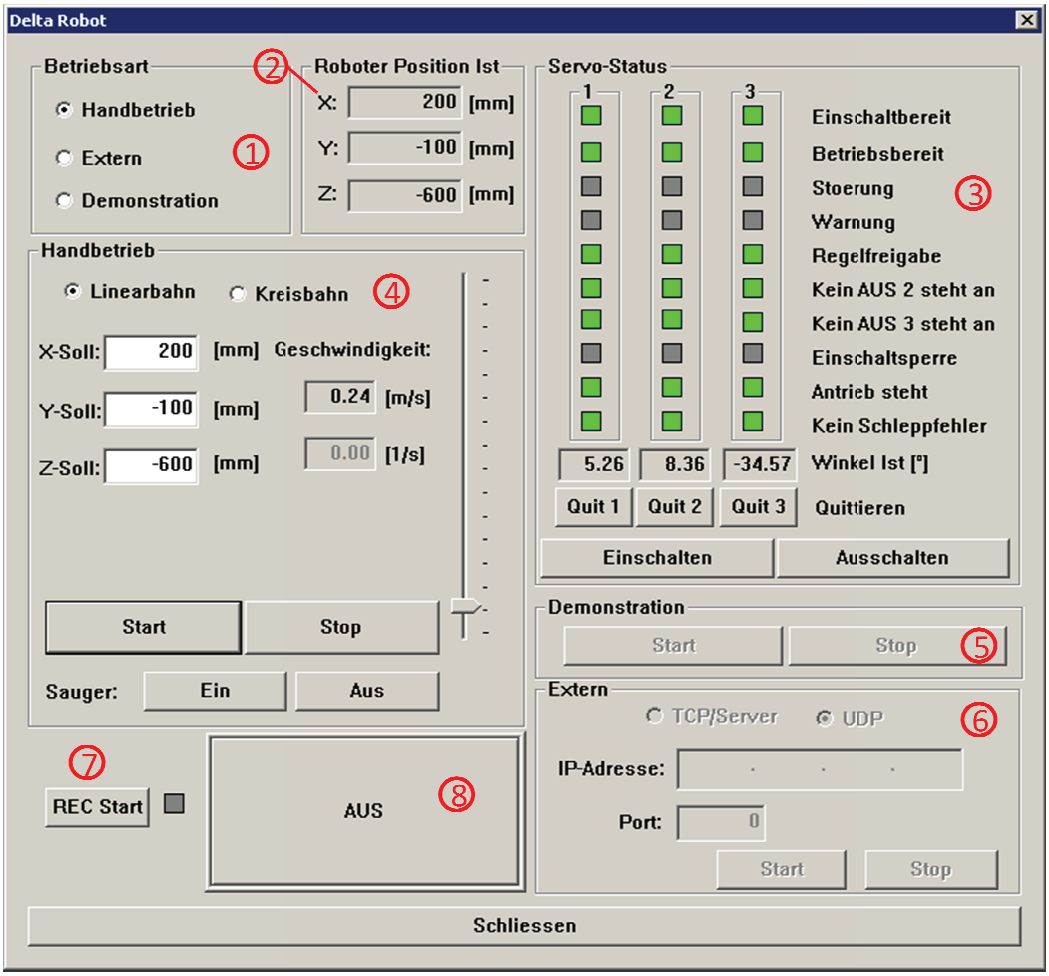


Abbildung 9.3-1:Bedienfenster: Automatikbetrieb

1. Betriebsartenauswahl
2. Anzeige des Positionsistwertes des TCPs
3. Statusanzeige und Steuerung der Antriebe
4. Handbetrieb
5. Demonstrationsfahrt
6. Externe Steuerung
7. Messwert-Aufzeichnung starten
8. Not Aus

Die Not-Aus-Funktion im Automatikbetrieb ist so umgesetzt, dass die Antriebe des Roboters nach der Betätigung des entsprechenden Buttons nicht abgeschaltet werden,

sondern zum Stillstand gebracht werden. Somit ist gewährleistet, dass der Roboter nicht durch den Eigenschwung weiter fährt. Um den Roboter im Anschluss wieder in Betrieb zu nehmen, ist ein erneutes Drücken des Buttons „Einschalten“ im Bereich der Antriebssteuerung notwendig.

Der Roboter sollte in diesem Bedienfenster nur benutzt werden, wenn zuvor die korrekte Kalibrierung der Roboterarme erfolgt ist. Andernfalls kann ein sicherer Betrieb nicht gewährleistet werden.

9.3.1 Handbetrieb

Im Handbetrieb ist es dem Bediener möglich, einzelne Fahrten des Roboters vorzugeben, um so eine Fahrtabfolge für eine externe Steuerung festzulegen.
Dabei fährt der Roboter bei einer Linearfahrt immer von seiner Istposition beim Start der Fahrt zu der vorgegebenen Sollposition auf dem kürzesten Weg. Die Maximalgeschwindigkeit, die der Roboter dabei erreichen soll, wird in Metern pro Sekunde vorgegeben und ist auf $3\,{}^{m}/_{s}$ begrenzt. Grund hierfür ist, dass die Mechanik des Roboters ansonsten zu sehr belastet wird.
Bei der Fahrt auf einer Kreisbahn, die in dieser Anwendung lediglich zu Demonstrationszwecken gedacht ist, wird eine Kreisbahn abgefahren. Der Abstand des TCPs vom Koordinatenursprung bezogen auf X- und Y-Achse dient dabei als Radius. Die Höhe zum Startzeitpunkt wird beibehalten. Die Geschwindigkeit wird in Umdrehungen pro Sekunde vorgegeben und ist auf $2\,{}^{1}/_{s}$ begrenzt.

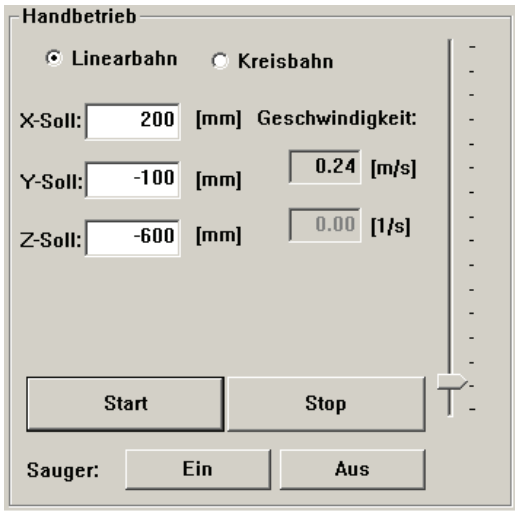


Abbildung 9.3-2:Handbetrieb

Die Fahrt wird, wenn sie gültig ist, über den Start-Button gestartet und kann jederzeit über den Stop-Button gestoppt werden. Da das System noch um einen Pneumatischen Sauger erweitert werden soll, ist dessen Ansteuerung hier schon implementiert. Die Ein- und Ausgänge des Saugers müssen nur noch innerhalb der Steuerungsapplikation mit den entsprechenden Bits in DB1.DBW73.3 bzw. DB1.DBW105.0 verschaltet werden.

Die Minimal- und Maximalwerte der Sollpositionen sind auf den Arbeitsbereich des Roboters begrenzt und können in der Klasse „RobotAuto.cpp“ in den entsprechenden Prüfprozessordefinitionen angepasst werden. Der Arbeitsbereich ist auf einen Radius in Bezug auf den Koordinatenursprung von 400mm für X- und Y-Achse begrenzt. Der gültige Arbeitsbereich der Z-Achse ist zwischen -500mm und -700mm. Diesen Begrenzungen liegt die Studienarbeit von Herrn Habermann zu Grunde. [1]

Beim Handbetrieb ist darauf zu achten, dass bei kleinen zu fahrenden Wegstrecken nicht mit hohen Geschwindigkeiten gefahren werden sollte. Dies ist einerseits nicht sinnvoll, andererseits führt es zu einer unnötigen Belastung der Mechanik. Grund hierfür ist, dass das die Arbeitsplatte immer auf die eingestellte Maximalgeschwindigkeit beschleunigt, was bei kleinen Wegstrecken zu hohen Kräften führt.

9.3.2 Externe Steuerung

Zur Steuerung des Roboters von einer auf einem externen Gerät laufenden Steuerungsapplikation, wurde ein geeignetes Protokoll zur Ansteuerung über TCP/IP oder UDP/IP realisiert. Die Bedienoberfläche kann dabei entweder als UDP-Teilnehmer durch Eingabe der IP-Adresse des externen Gerätes und des entsprechenden Port oder als TCP-Server nur mit Eingabe des Port eingestellt werden. Bei Festlegung als TCP-Server ist dabei zu beachten, dass die externe Steuerung innerhalb der Bedienoberfläche erst aktiviert wird, wenn das als Client ausgelegte Fremdgerät seine TCP/IP-Schnittstelle gestartet hat. Andernfalls findet keine Kommunikation statt.

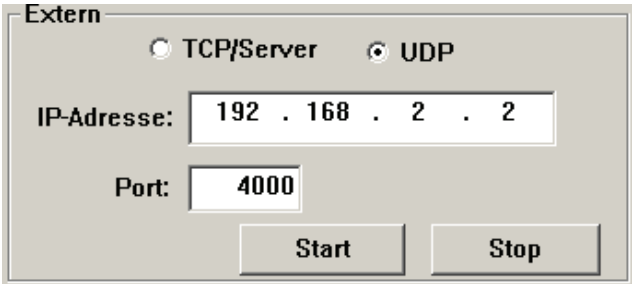


Abbildung 9.3-3:Externe Steuerung

Das Sendeprotokoll des Roboters ist wie folgt definiert.

Adresse	Typ	Name	Beschreibung
0	INT32	X-Ist	Istposition der X-Achse [mm]
4	INT32	Y-Ist	Istlposition der Y-Achse [mm]
8	INT32	Z-Ist	Istposition der Z-Achse [mm]
12	INT32	Handshake_Out	Handshake_In = Handshake_Out
16	DWORD	Status	Status des Roboters
16.0	BOOL	Reserve	-
⋮	⋮	⋮	⋮
19.0	BOOL	Antr_Stoerung	Einer oder mehrere Antriebe sind in Störung
19.1	BOOL	Antr_Bereit	Alle Antriebe sind bereit
19.2	BOOL	PosErreicht	Sollposition ist erreicht
19.3	BOOL	PosOk	Sollposition liegt im Arbeitsbereich
19.4	BOOL	Werk_an	Ein Werkstück ist angesaugt
⋮	⋮	⋮	⋮
19.7	BOOL	Reserve	-

Tabelle 11:Sendeprotokoll des Roboters

Das Protokoll, dass der Roboter empfangen soll wird folgendermaßen definiert.

Adresse	Typ	Name	Beschreibung
0	INT32	X-Soll	Sollposition der X-Achse [mm]
4	INT32	Y-Soll	Sollposition der Y-Achse [mm]
8	INT32	Z-Soll	Sollposition der Z-Achse [mm]
12	INT32	Geschw_Soll	Sollgeschwindigkeit [mm/s]
16	INT32	Handshake_In	Handshake_In = Handshake_Out
24	DWORD	Steuer	Status des Roboters
24.0	BOOL	Reserve	-
⋮	⋮	⋮	⋮
27.0	BOOL	Senden_Ein	Roboter soll das Statusprotokoll zurücksenden
27.1	BOOL	Start	Fahrt starten
27.2	BOOL	Stop	Fahrt stoppen
27.3	BOOL	Reserve	-
27.4	BOOL	Sauger_Ein	Sauger einschalten/ausschalten
⋮	⋮	⋮	⋮
27.7	BOOL	Reserve	-

Tabelle 12:Empfangsprotokoll des Roboters

Neben den Soll- und Istwerten sowie der Status- und Steuerworte, wird eine Handshakevariable versendet. Dabei sendet der Roboter genau den Wert zurück, den er zuvor empfangen hat. Somit lässt sich aus Sicht der externen Applikation, die diesen Wert vorgibt überprüfen, ob die Kommunikation noch läuft und aktuell ist.
Um außerdem die Kommunikation möglichst gering zu halten, sendet der Roboter sein Statusprotokoll nur, wenn dies über das Bit 27.0 von der externen Applikation angefordert wurde.
In dieser Betriebsart ist nur die Fahrt einer Linearbahn zwischen zwei Punkten möglich.

Die Schnittstelle wurde mittels einer eigens dafür programmierten Hilfsapplikation in C++ sowie mit Matlab/Simulink getestet.

9.3.3 Demonstrationsfahrt

Um die Dynamik des Roboters Interessierten vorzuführen, wurde speziell hierfür eine eigene Betriebsart realisiert. Der Roboter fährt nach Start der Demonstrationsfahrt eine definierte Reihenfolge von Bahnen ab. Dazu gehören sowohl Linear- als auch Kreisbahnen. Der Roboter simuliert dabei das Aufheben und Absetzen von Werkstücken mit unterschiedlichen Geschwindigkeiten.

9.3.4 Aufzeichnung von Messwerten

Um das Fahrverhalten des Roboters zu untersuchen, ist hierfür eine Messfunktion implementiert worden. Wird die Funktion über den Button „REC Start“ innerhalb des Bedienfensters aktiviert, werden bei der nächsten Fahrt im Handbetrieb die Lageistwerte der Antriebe sowie die entsprechenden Sollwerte des Führungsgrößengenerators in einer CSV[31]-Datei abgespeichert.
Dazu wird ein neuer Thread gestartet, der alle 5ms die entsprechenden Werte unter Angabe der fortlaufenden Zeit abspeichert. Der Thread zur Aktualisierung der Prozessvariablen wird für diesen Zeitraum unterbrochen, um die Aufzeichnung nicht zu verzögern. Die Aufzeichnung der Messwerte läuft immer für 5 Sekunden.
Im Anschluss kann die CSV-Datei mit Namen „LogData.csv“ im Ordner der Bedienapplikation auf dem Desktop den S7-mECs heraus kopiert werden und mittels Matlab ausgewertet werden. Jede Spalte ist wie folgt aufgebaut:

$t_{ist}[sek]$	$\theta_{1ist}[°]$	$\theta_{2ist}[°]$	$\theta_{3ist}[°]$	$\theta_{1soll}[°]$	$\theta_{2soll}[°]$	$\theta_{3soll}[°]$

[31] Comma-Separated Values

10 Inbetriebnahme

Die Inbetriebnahme der Gesamtanlage wird in diesem Kapitel vorgestellt. Dazu gehören die Vorgehensweise, die Einstellungen die an der WinLC getroffenen wurden sowie die Überprüfung des Fahrverhaltens des Roboters gegenüber der Simulation.

10.1 Vorgehensweise

Vor Beginn der Inbetriebnahme war das bereits beschriebene Konzept zur Automatisierung des Roboters erarbeitet und alle benötigten Schnittstellen vorbereitet. Somit mussten die einzelnen Programm-Teile nur noch implementiert werden und keine Grundlagen mehr erarbeitet werden.
Nach der Montage des Schaltschrankes und dessen Anschluss an die, sich am Roboter befindenden Antriebe, wurden zunächst alle Antriebssysteme projektiert und dessen Dynamik überprüft. Dabei wurden die Roboterarme demontiert, um mechanische Schäden während der Inbetriebnahme auszuschließen.
Im Anschluss wurde die softwaretechnische Lösung der Steuerungsapplikation vorbereitet. Dazu zählen die Kommunikation mit den Antriebssystemen und der Bedienoberfläche, die Lageregelung der Antriebe sowie die Implementierung des bereits vorbereiteten Führungsgrößengenerators.
Im nächsten Schritt wurde das Bedienfenster der Antriebskalibrierung fertiggestellt, um die Antriebe einstellen zu können und die richtigen Nullpunkte der Roboterarme zu definieren. Darauf folgte die Programmierung des Automatikbetriebes. Dabei wurde zuerst der Handbetrieb, zur Vorgabe einzelner Fahrten des Roboters, realisiert. Dieser wurde im Anschluss ausgiebig getestet, ohne dass die Arme des Roboters montiert waren. Hierbei wurden Feinabstimmungen zum Zuschalten der Sollwerte gemacht. Auch die Sicherheitsfunktion innerhalb der Steuerapplikation und die Begrenzung des Arbeitsbereiches wurden in dieser Phase programmiert.
Als die Testphase abgeschlossen war und eine Beschädigung des Roboters ausgeschlossen werden konnte, wurden die Arme des Roboters montiert. Es stellte sich bei den Tests mit montierten Armen schnell heraus, dass die gewünschte Millimeter genaue Bahnverfolgung mit dem bloßen Auge nicht überprüft werden konnte. Aus diesem Grund wurde die bereits angesprochene Messfunktion umgesetzt und an Hand dieser die Bahnverfolgung überprüft.
Nach der Sicherstellung der korrekten Bahnverfolgung wurden die anderen beiden Betriebsarten umgesetzt, beginnend mit der externen Steuerung.

10.2 Einstellungen

Mittels des Tuning Panels, dass aus dem Betriebssystem des Embedded Controllers heraus gestartet werden kann, lassen sich die Parameter zur Leistungsfähigkeit der Soft-SPS einstellen. Dazu gehören die Priorisierung der Echtzeitanwendung und Vorgabe der Mindestruhezeit, Mindestzykluszeit, maximale Ausführungszeit und Ausführungslast. Außerdem lassen sich hier die Zykluszeiten und die CPU-Auslastung beobachten. Die folgende Abbildung zeigt das Tuning Panel und die Einstellungen die an ihm vorgenommen wurden.

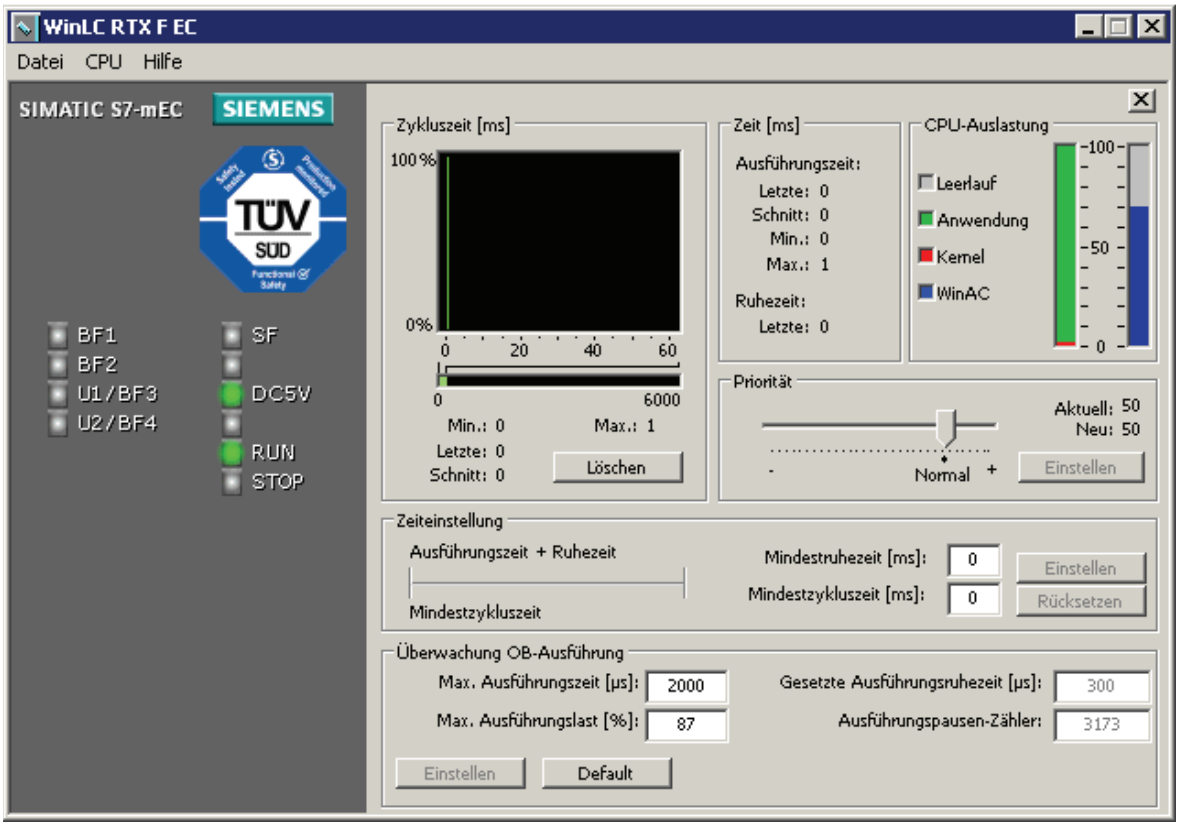


Abbildung 10.2-1:Einstellungen am Tuning Panel

Um die Zykluszeit der Lageregler und damit der Soft-SPS möglichst klein zu halten, wurde eine maximale Zykluszeit von 2ms eingestellt und die Mindestwartezeit ausgeschaltet. Eine kleinere Zykluszeit ist nicht sinnvoll, da die Aktualisierung der PROFINET-Schnittstelle zwischen Steuerung und den Antriebsystemen ebenfalls auf 2ms eingestellt ist und nicht weiter verkleinert werden kann. Außerdem muss darauf geachtet werden, dass anderen Anwendungen, die auf dem Embedded Controller abgearbeitet werden, ebenfalls Rechenzeit zur Verfügung steht. Alle anderen Einstellungen wurden nicht geändert.

10.3 Messungen

In diesem Abschnitt wird das Fahrverhalten des Roboters mit der schon vorgestellten Simulation verglichen. Dazu wurden die gleichen Fahrten des realen Systems gemessen, die auch schon unter 7.3 simuliert wurden.

10.3.1 Linearfahrt

Zunächst wird nun eine lineare Fahrt des Roboters zwischen zwei Punkten gemessen. Die Start und Zielkoordinaten sind hierfür folgendermaßen definiert.

$$xyz_{Start} = \begin{pmatrix} 0{,}25 \\ 0{,}25 \\ -0.6 \end{pmatrix}[m]; xyz_{Ziel} = \begin{pmatrix} -0{,}25 \\ -0{,}25 \\ -0.6 \end{pmatrix}[m]$$

Die Maximalgeschwindigkeit ist mit $v_{max} = 1\,{}^{m}/_{s}$ eingestellt.

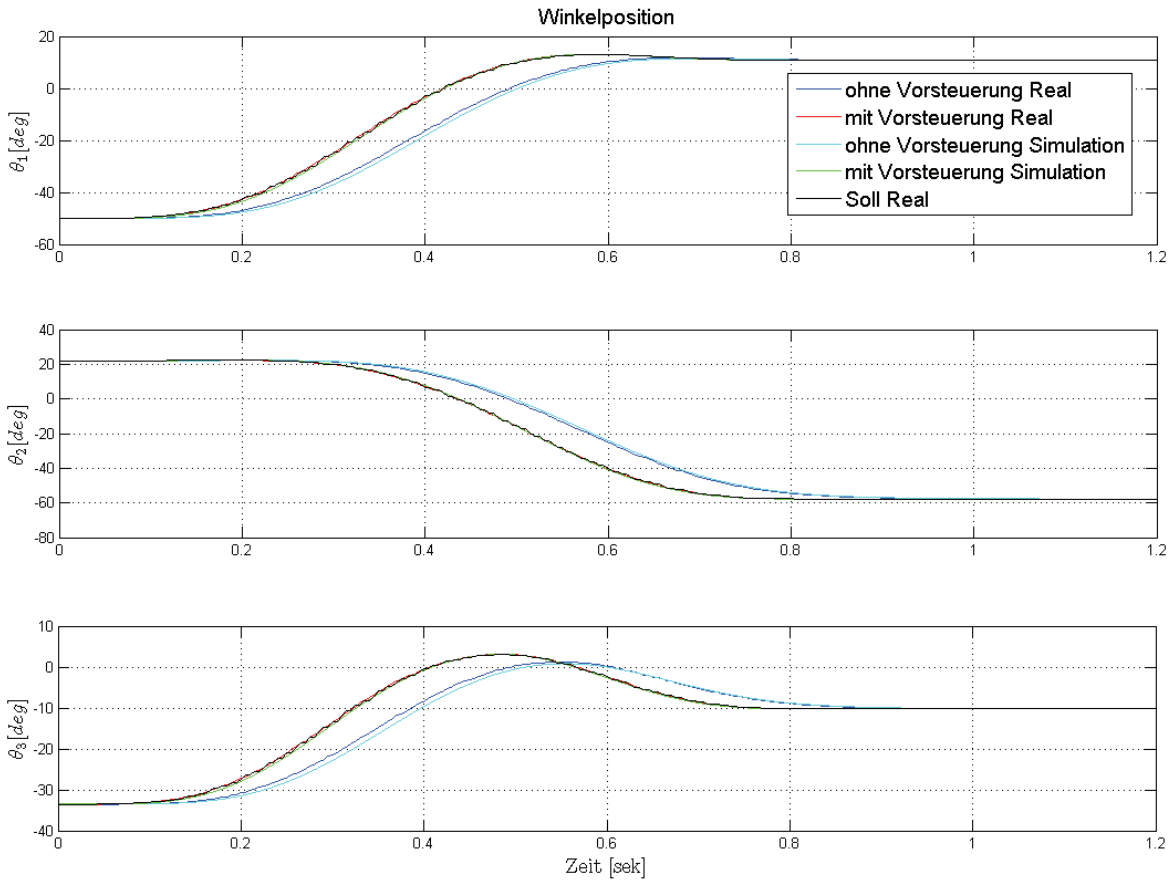


Abbildung 10.3-1:Vergleich Vergleich der Winkelistwerte Linearfahrt

In allen Abbildungen entspricht Schwarz den Sollwerten des realen Führungsgrößengenerators, Rot dem gemessenen Istwert mit Vorsteuerung, Blau dem gemessenen Istwert ohne Vorsteuerung, Grün dem simulierten Istwert mit Vorsteuerung und Cyan dem simulierten Istwert ohne Vorsteuerung.

Die obige Abbildung zeigt die Messung und Simulation der Winkelistwerte bei der Linearfahrt. Deutlich zu sehen ist, dass sich die Plots vom realen und simulierten System mit Vorsteuerung kaum unterscheiden und fast deckungsgleich mit der Sollwerttrajektorien sind. Im Gegensatz dazu zeigt sich beim Vergleich der Winkelistwerte ohne Vorsteuerung ein leichter Unterschied. Die gemessenen Winkelistwerte eilen weniger nach als die Simulierten. Der Grund hierfür ist in der Modellierung der Antriebe zu finden. In der Abbildung 7.1-11 wurde die Sprungantwort des Lageregelkreises von simuliertem und realem Antrieb verglichen. Dabei zeigte sich, dass das reale System etwas schneller auf den Winkelsollwert einregelt als das simulierte Model. Wird der simulierte Lageregelkreis schneller eingestellt fällt dieser Unterschied weg. Dieser Effekt macht sich bei den vorgesteuerten Fahrten nicht bemerkbar, da hier die Stellgröße der Solldrehzahl der Antriebe vom Führungsgrößengenerator stammt und der Lageregler im Idealfall keine Stellgröße beisteuert.

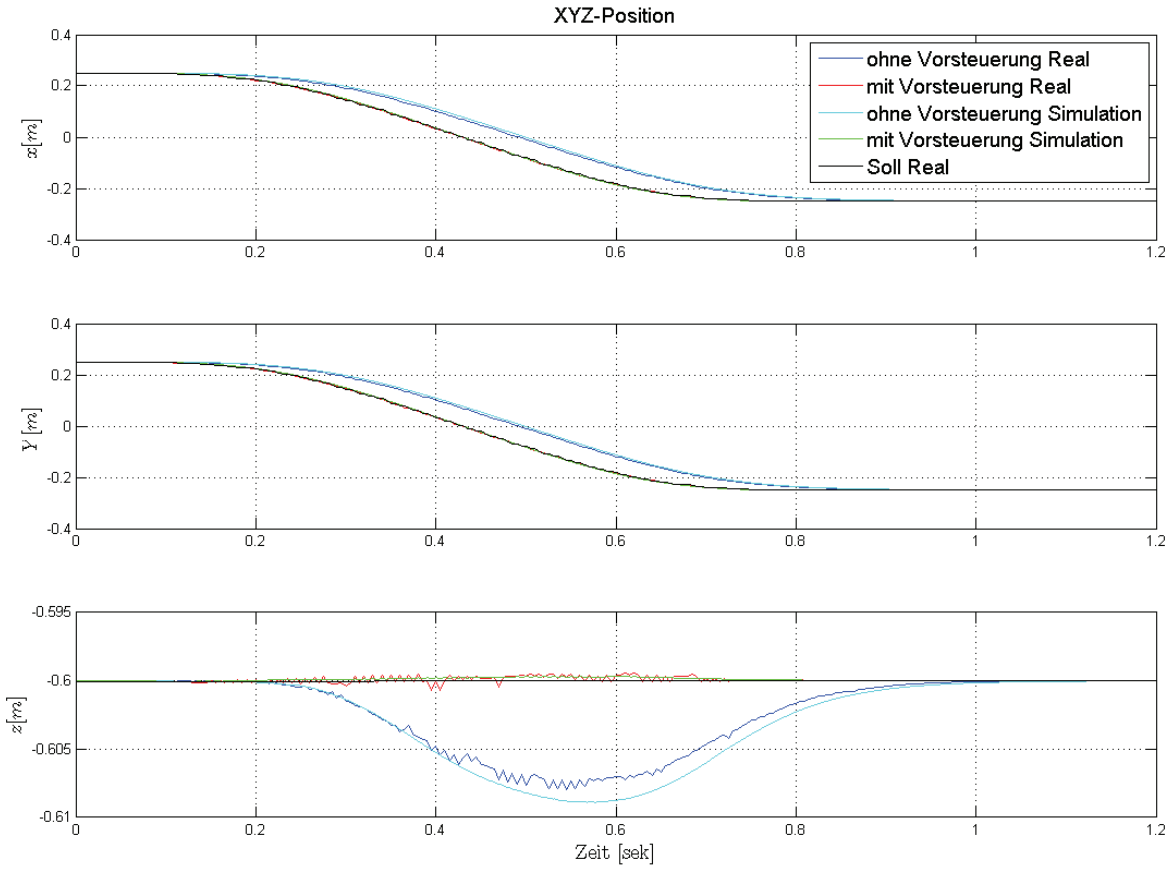


Abbildung 10.3-2:Vergleich der XYZ-Position Linearfahrt

Die Beobachtungen der Winkelistwerte zeigen sich auch beim Vergleich der XYZ-Positionen des TCPs in Abbildung 10.3-2. Dabei wurden die gemessenen Winkelistwerte mittels der Forward-Kinematik mit Matlab in die entsprechenden Positionswerte umge-

rechnet. Während die Positionsistwerte der vorgesteuerten Fahrten sich kaum unterscheiden und die Sollbahn Millimeter genau abfahren, ist ohne Vorsteuerung eine deutliche Abweichung zu erkennen. Dabei ist das reale System schneller und genauer. Dies wird vor allem beim Vergleich der Z-Achsen deutlich. Während der gemessene Positionsistwert um maximal ca. 7mm von der Sollbahn abweicht, sind es bei der Simulation ca. 9mm.

Es ist zu beobachten, dass die gemessenen Positionsistwerte des realen Systems leicht verrauscht sind. Der Grund hierfür ist vermutlich, dass die Steuerungsapplikation und die Umrichter, interne Drehzahlregelung und Lageistwert-Erfassung, unterschiedliche Zykluszeiten haben. Die Zykluszeit der Steuerung liegt bei ca. 2ms, während die Zykluszeit der Umrichter bei 125µs liegt. Somit kann es dazu kommen, dass sich die Lageistwerte der Antriebe während des Auslesens, seitens der Steuerung bereits geändert haben und ein Istwert aktueller als der andere ist. Für die Berechnung der XYZ-Position mittels der Forward-Kinematik bedeutet dies, dass die der Berechnung zu Grunde liegenden Winkel nicht zu einander passen und somit eine falsche Position ergeben. Da die Abweichungen jedoch minimal sind, werden sie hingenommen.

Abbildung 10.3-3 zeigt die Linearfahrt zur Veranschaulichung im dreidimensionalen Koordinatensystem.

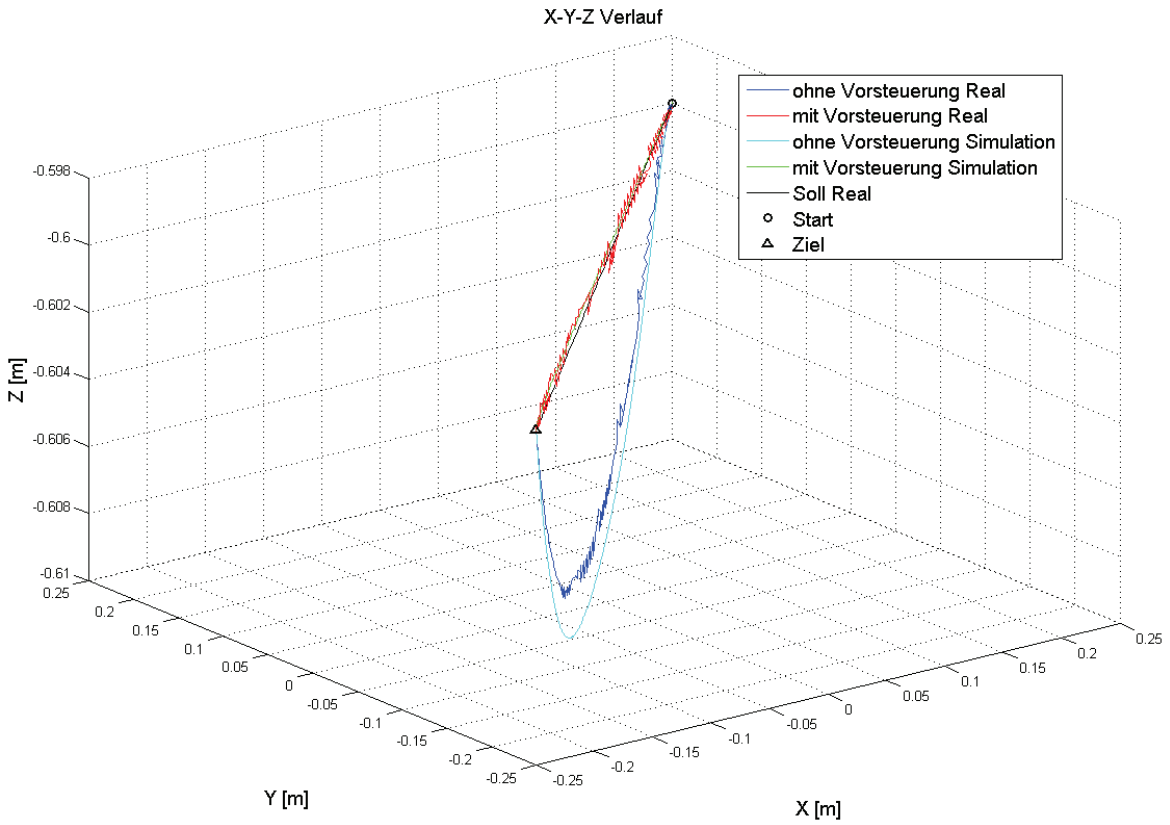


Abbildung 10.3-3:Vergleich der XYZ-Position Linearfahrt 3D

10.3.2 Kreisfahrt

Um die Analyse des Fahrverhaltens abzuschließen, wird nun die bereits beschriebene Abfahrt einer Kreisbahn untersucht. Die Geschwindigkeit ist auf eine Umdrehung pro Sekunde eingestellt. Die Startposition des TCPs ist:

$$xyz_{Start} = \begin{pmatrix} 0 \\ 0{,}2 \\ -0.6 \end{pmatrix} [m]$$

Es ist zu erwarten, dass sich die Beobachtungen, die bei der Linearfahrt gemacht wurden, auch bei dieser Messung zeigen.

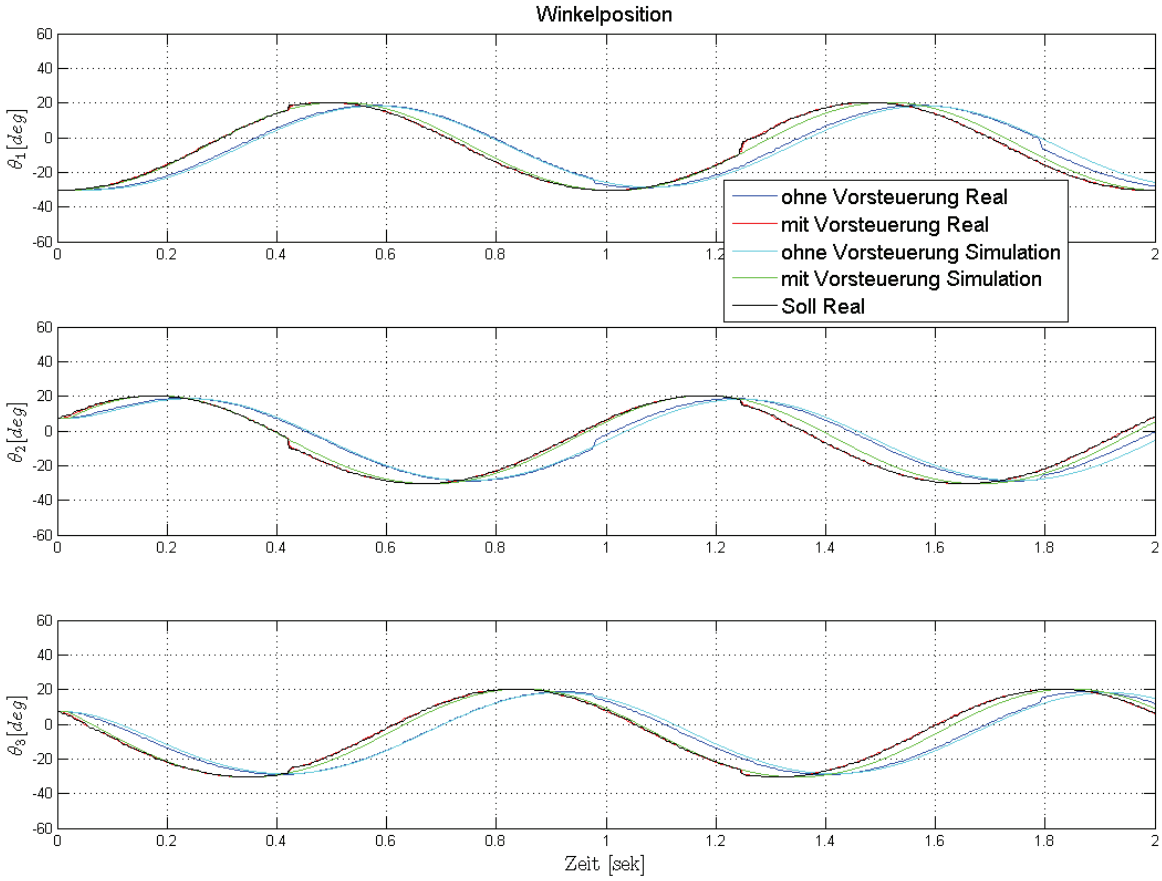


Abbildung 10.3-4:Vergleich der Winkelistwerte Kreisfahrt

In der obigen Abbildung ist die zeitliche Änderung der Winkelistwerte dargestellt. Es fällt deutlich auf, dass die gemessenen Winkelistwerte alle ca. 800ms ein sprunghaftes Verhalten zeigen. Dabei handelt es sich jedoch nicht um eine ungewollte Bewegung die der Roboter ausführt, sondern um eine Unterbrechung der in der Bedienoberfläche implementierten Messfunktion. Da die Bedienoberfläche innerhalb des Betriebssystems ausgeführt wird, unterliegt sie damit auch dessen Resourcenverwaltung. Obwohl der Thread, der die Messungen abspeichert, mit der höchsten in C++ möglichen Priorisierung ausgeführt wird,

wird er vom Betriebssystem unterbrochen, um andere Vorgänge zu bearbeiten. Diese Unterbrechung lässt sich nicht umgehen und muss hingenommen werden.
Trotz der fehlerhaften Messwert-Speicherung lassen sich die Aussagen, die bei der Simulation des Systems getroffen wurden, bestätigen. Auch hier eilen die Winkelistwerte des nicht vorgesteuerten Roboters hinterher. Zudem werden die Spitzenwerte nicht erreicht. Das hat zur Folge, dass die Position des TCPs auch hier nicht die gewünschte Sollbahn abfährt.

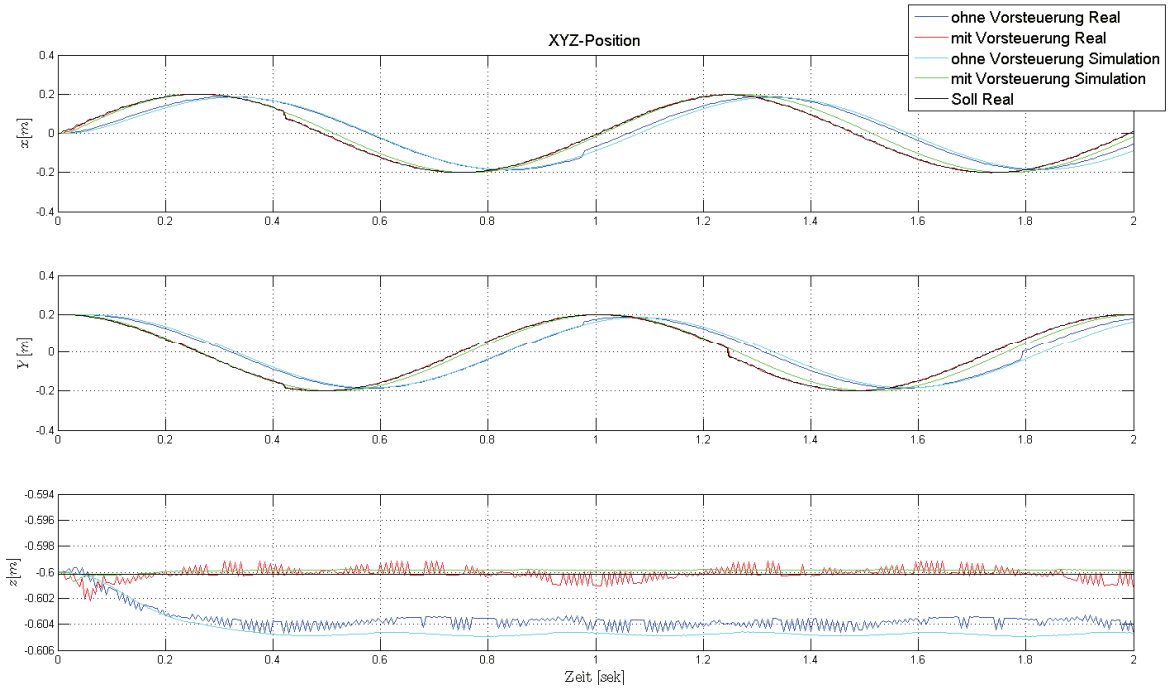


Abbildung 10.3-5:Vergleich der XYZ-Position Kreisfahrt

Die gleichen Beobachtungen lassen sich auch bei XYZ-Position des TCPs treffen. Auffällig ist hierbei der Vergleich der Z-Achse. Die gemessenen Istwerte schwanken deutlich größer als die Simulierten. Da diese Schwankungen immer dann auftreten, wenn einer der Antriebe seine Drehrichtung ändert, wird angenommen, dass das leichte Spiel zwischen Antriebsflansch und Getriebe damit zu tun hat sowie die Einregelzeit der Drehzahlregelkreise. Dieses Spiel sorgt dafür, dass auf die Antriebe kurzzeitig keine mechanische Last seitens des Roboters wirken und die Vorsteuerung der Drehmomente für diesen Zeitraum fehlerhaft ist. Das Spiel zwischen Getriebe und Antriebsflansch war bei der mechanischen Konstruktion nicht vorgesehen und wurde deshalb auch nicht im Führungsgrößengenerator berücksichtigt.
Auch bei dieser Messung zeigt sich beim Vergleich der nicht vorgesteuerten realen und simulierten Messung, dass das reale System etwas genauer verfährt. Während die nicht vorgesteuerte Messung des Roboters zeigt, dass die Abweichung für die Z-Achse bei ca.

4mm liegt, sind es bei der Simulation ca. 5mm. Die Abweichungen bezogen auf X- und Y-Achse liegen beim realen System bei ca. 1,3cm und bei der Simulation bei ca. 1,5cm.

Zur Veranschaulichung der Messergebnisse zeigen die beiden folgenden Abbildungen die dreidimensionale Darstellung der Fahrt sowie die XY-Darstellung.

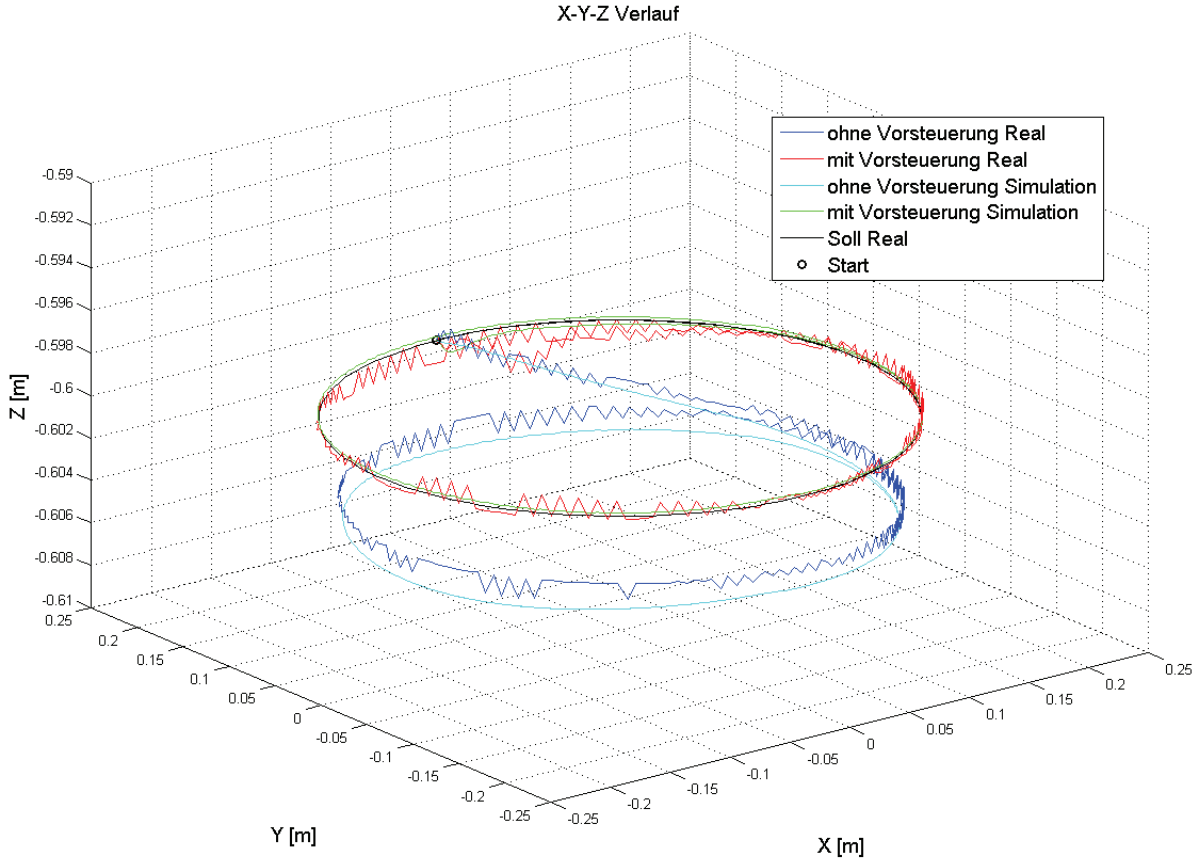


Abbildung 10.3-6:Vergleich der XYZ-Position Kreisfahrt 3D

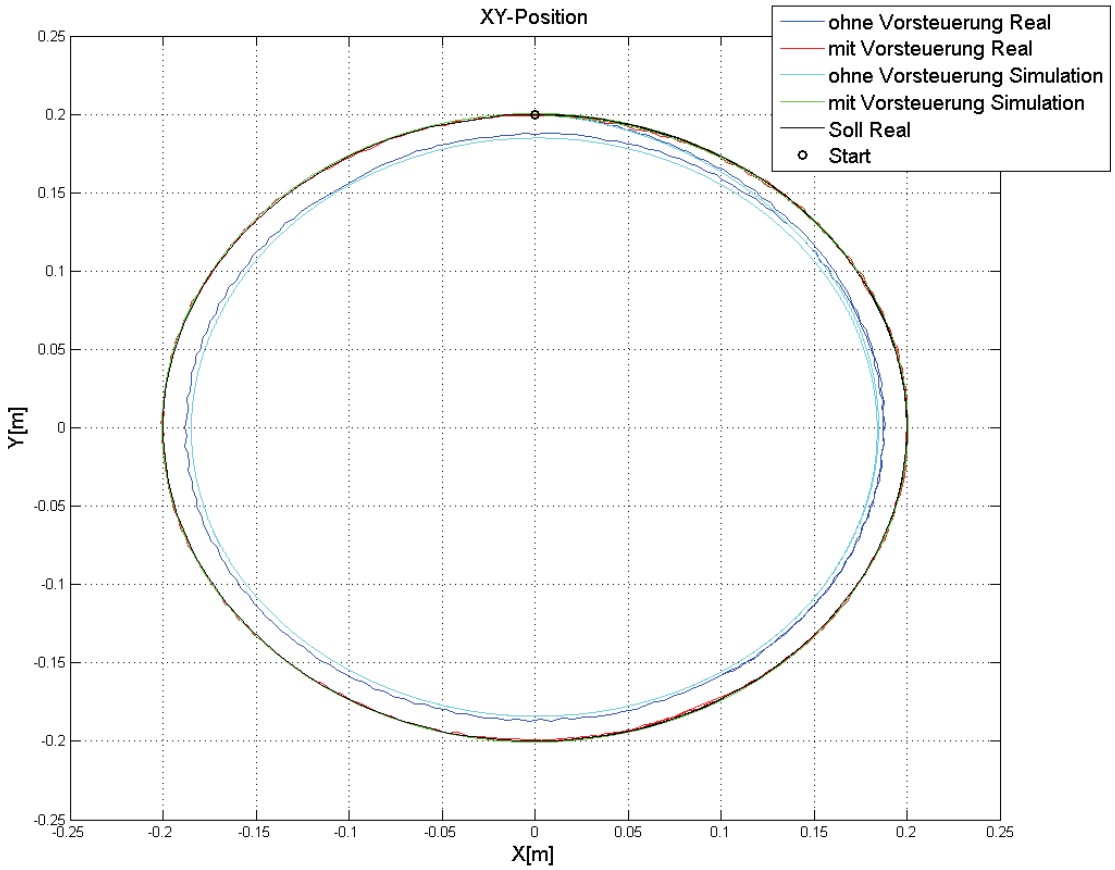


Abbildung 10.3-7:Vergleich der XY-Position Kreisfahrt

11 Zusammenfassung und Ausblick

In dieser Masterthesis wurde ein Parallelkinematik-Roboter mittels eines S7-mEC und S120 Antriebssystemen automatisiert. Die Regelung erfolgt über eine kaskadierte Lageregelung der Synchronantriebe und eines Führungsgrößengenerators, der sowohl die Interpolation des Fahrprofils als auch die Berechnung der benötigten Drehzahlen und Drehmomente beinhaltet.
Zur Automatisierung des Roboters wurde dessen mathematisches Modell hergeleitet und mittels Matlab/Simulink der zu Grunde liegende Regelungsansatz simuliert. Sowohl die Simulation als auch die Messungen am realen System zeigen, dass der Ansatz eines Führungsgrößengenerators zur Berechnung der Vorsteuergrößen eine signifikante Verbesserung des Fahrverhaltens bewirkt. Der Roboter ist so in der Lage, innerhalb einer Sekunde jeden beliebigen Punkt innerhalb seines Arbeitsbereiches anzufahren und dabei weniger als einen Millimeter von seiner Sollbahn abzuweichen.
Die softwaretechnische Lösung der Aufgabe setzt sich aus verschiedenen Komponenten zusammen. Die Umrichter interne Drehzahlregelung wurde mit der Inbetriebnahme-Software STARTER projektiert. Der in C++ programmierte Führungsgrößengenerator wird innerhalb der in STEP 7 projektierten Steuerungsapplikation aufgerufen. In dieser ist außerdem sowohl die Lageregelung der Antriebe, als auch die Kommunikation mit den Antriebssystemen und der Bedienoberfläche realisiert. Die Bedienoberfläche selber ist eine in C++ programmierte Anwendung, die aus dem Betriebssystem des Embedded Controller heraus aufgerufen wird. Die Betriebsarten des Roboters umfassen die Ansteuerung der einzelnen Roboterarme, die manuelle Vorgabe von Fahrten, eine externe Steuerung über TCP/IP oder UPD/IP und das Starten einer Demonstrationsfahrt. Zur Integration der Hochsprachenapplikationen des Führungsgrößengenerator und der Bedienoberfläche in die Steuerungslösung wird die Schnittstellenerweiterung WINAC ODK genutzt.
Des Weiteren wurde der elektrotechnische Aufbau des Roboters erarbeitet und in einem Schaltschrank montiert.

Abgesehen von der Montage eines pneumatischen Saugers wurden alle geforderten Aufgabenteile erfolgreich erfüllt. Die Integration des Saugers konnte aus zeitlichen Gründen nicht abgeschlossen werden, ist aber softwaretechnisch vorbereitet. Im Anschluss an diese Thesis kann der Roboter in das Verbundprojekt „Autonome Systeme“ des Masterstudiengangs Automatisierungstechnik integriert werden. Dazu sollte eine oder

mehrere Andockstationen montiert werden, die es sowohl dem Delta-Roboter als auch den mobilen Robotern erlauben, Werkstücke aufzunehmen und abzulegen.
Im Laufe der Bearbeitungszeit der Thesis haben sich verschiedene Verbesserungs- und Erweiterungsvorschlage ergeben.

Bedienoberfläche:
Die Bedienoberfläche kann mittels MFC um eine graphische Anzeige der Roboterposition erweitert werden. Dazu könnte die Position des Roboters in einem dreidimensionalen Plot dargestellt werden. Desweiteren bietet sich die Möglichkeit, den Roboter um eine weitere Betriebsart zu erweitern. Dabei könnten die im Handbetrieb vorgegebenen Fahrten in einer CSV-Datei gespeichert werden und anschließend wieder automatisch in der gleichen Reihenfolge ausgeführt werden.

Messungen:
Die unter 10.3.2 beschriebene Unterbrechung der Messfunktion könnte behoben werden, indem diese nicht mehr innerhalb der Bedienoberfläche aufgerufen wird, sondern in der Applikation des Führungsgrößengenerators. Die genaue Umsetzung zum Speichern der Messwerte müsste dafür allerdings erst erarbeitet werden.

Mechanik:
Im Laufe der Testphase stellte sich heraus, dass der Roboteraufbau bei schnellen Fahrten wackelt. Um diese ungewollte Bewegung zu verhindern, könnten zusätzliche Stützstreben montiert werden. Ebenfalls sollte geprüft werden, ob das Spiel, das zwischen Getriebe und Antriebsflansch auftritt, behoben werden kann. Außerdem ist es ratsam, über einen Austausch der Unterarmstreben und dessen Sicherungsringe nachzudenken. Die verbauten Streben können bei schnellen Positionierungen an den Grenzen des Arbeitsbereiches brechen. Abhilfe könnten stabilere Streben schaffen, die nur aus Metall bestehen.

Regelung:
Bei einer Folgearbeit könnten weitere Ansätze zur Positionierung des Roboters implementiert werden. Denkbar ist z.B. der Ansatz einer adaptiven Zustandsregelung auf Grundlage des erarbeiteten Robotermodels. Eine solche Regelung wird bereits in der Praxis angewendet. [7] Außerdem könnte die Interpolation der Kreisbahn überarbeitet werden. Zum Start der Kreisfahrt treten ruckartige Beschleunigungen auf, da ein oder mehrere Arme sofort eine hohe Geschwindigkeit erreichen müssen.

Danksagung

An dieser Stelle möchte ich zuerst meiner Familie danken, die mich im Laufe meiner Ausbildung finanziell und moralisch unterstützt hat und so mein Studium erst ermöglicht hat.

Ein weiterer Dank gilt den Mitarbeitern der Hochschule für Angewandte Wissenschaften Hamburg und meinen Kommunionen, die mir mit Rat und Tat zur Seite standen.

Des Weiteren bedanke ich mich bei Herrn Prof. Dr. Ing. Ulfert Meiners und Herrn Prof. Dr. Thomas Holzhüter für die Betreuung der Masterthesis und die Übernahme des Erst- und Zweitgutachtens.

Literaturverzeichnis

[1] „Konstruktion eines Delta-Roboters“, Eugen Habermann, Studienarbeit HAW Hamburg

[2] Siemens SIMATIC WinAC ODK User Manual 04/2009

[3] Siemens SIMATIC Windows Automation Center RTX 05/2004

[4] Betriebsanleitung S7-modular Embedded Controller 07/2009

[5] „Inverse - Forward Kinematics of a Delta Robot”, Jon Martinez Garcia, 2010

[6] „Dynamic Modeling and Mass Matrix Evaluation of the DELTA Parallel Robot for Axes Decoupling Control“, Alain Codourey, Institute of Robotics, ETH-Zürich, 1996

[7] „Modeling and control of a Delta-3 robot“, André Olsson, Masterthesis Lund University, 2009

[8] „Regelungstechnische Untersuchung eines Servoantriebs und Entwurf eines system-theoretischen Simulationsmodells“, Hannes Müller, Bachelorthesis HAW Hamburg, 2012

[9] „Antriebstechnik für mobile Systeme“, Prof. Dr. –Ing. Michael Röther, HAW Hamburg, 2010

[10] „Industrieroboter: Methoden der Steuerung und Regelung“, Wolfgang Weber, 2. Aufl., Fachbuchverl. Leipzig im Carl-Hanser-Verl., 2009

[11] „Visual C++ 6 in 21 Tagen“ David Chapman, Markt und Technik Verlag, 2004

[12] http://www.automation.siemens.com, Stand 24.05.2012

Anhang

A1: Schaltschrank und Antriebe: Datenblätter, Stückliste, Schaltpläne und Montagezeichnungen
A2: Matlab-Dateien
A3: STARTER-Projekt
A4: STEP 7-Projekt
A5: Projekte der Bedienoberfläche und des Führungsgrößengenerators
A6: Bilder und Videos

Die Anhänge A1-A6 sind in elektronischer Form auf der beigelegten CD zu finden.

Versicherung der Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 31.Mai 2012

Ort, Datum Unterschrift