Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit

Entwicklung eines Flächenportals als Teil einer flexiblen Fertigungszelle

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs (Dipl.-Ing)
eingereicht an der TU Wien, Fakultät für Maschinenwesen und Betriebswissenschaften von

Fabian Josef SCHEIDL
Mat.Nr.: 1126160

unter der Leitung von

Univ.Prof. Dipl.-Ing. Dr.-Ing. Detlef GERHARD
Institut für Konstruktionswissenschaften und Technische Logistik, E307

Ich nehme zur Kenntnis, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Diplomarbeit

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen sind als solche kenntlich gemacht.
Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein.

Wien, Januar, 2017

Unterschrift

Zusammenfassung

In Zeiten von Industrie 4.0 wird Produktionsautomatisierung enormer Wert zugeschrieben. Werkzeugmaschinen sollen automatisch, ohne menschliches Eingreifen, bestückt sowie entladen und über Netzwerke ferngesteuert werden können.

Diese Arbeit beschäftigt sich mit dem Aufbau eines Flächenportals als Teil eines Portalroboters mit der Funktion einer Handhabungsanlage zur Bestückung einer Portalfräse. Hierfür werden Alternativsysteme und ein Eigenbau in Betracht gezogen.

Es wird ein Flächenportal analog zur Funktionsweise einer bestehenden Portalfräse entworfen sowie eine integrierte Elektronikeinheit mit einer USB-Schnittstelle zur Kommunikation mit einem Host-PC zur Steuerung des Portalroboters erstellt. Darüberhinaus wird ein einfaches Kommunikationsprotokoll zur Informationsübertragung über USB/UART vorgestellt und eine beispielhafte Implementierung dieses in Form einer Mikrocontroller-Firmware für die Steuerplatine präsentiert.

Im Zuge der Beschreibung der Motorsteuerung wird auf verschiedene Ansteuerungsvarianten und Regelungen für Schrittmotoren eingegangen. Weiters wird die Herleitung einer Näherungsformel für die effiziente Berechnung und Erleichterung einer Mikrocontroller-Implementierung eines vorgegebenen Bewegungsprofils, das beim Anfahren und Bremsen konstante Beschleunigung aufweist, demonstrativ hergeleitet.

Abstract

In times of Industry 4.0 utter importance is placed on the automation of production. Machine tools are to be loaded and unloaded automatically without the necessity of manual human intervention and remote controlled via various networks.

The objective of this paper is the construction of an area gantry as part of a 3-axis handling robot for the purpose of loading a portal milling machine. Various available alternative options and a custom-developed version are considered in their price points.

An area gantry is constructed analogously to the setup of an existing CNC router. Furthermore, an integrated mainboard, featuring a USB-interface enabling communication with a host-PC, for controlling the gantry is designed. A simple communication protocol for transmitting data and control information via USB/UART and a sample implementation of that very protocol as part of a microcontroller firmware is presented.
Within the scope of the description of the motor control unit, various state-of-the-art methods for control of stepper motors are shown. A simple approximation for the efficient calculation and microcontroller-implementation of a given movement profile, comprising phases of constant acceleration and deceleration, is derived.

Inhaltsverzeichnis

[1]Metal–Oxide–Semiconductor Field-Effect Transistor

Liste der Abkürzungen

AC	Alternating Current
BLDC	Brushless Direct Current
CAD	Computer Aided Design
CNC	Computer Numerical Control
CRC	Cyclic Redundancy Check
DC	Direct Current
EMF	Electromotive Force
GPIO	General Purpose In-Out
IC	Integrated Circuit
LDO	Low-dropout
LQFP	Low Profile Quad Flat Package
LSB	Least Significant Bit
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
MSB	Most Significant Bit
PCB	Printed Circuit Board
PIT	Periodic Interrupt Timer

PMSM Permanent Magnet Synchronous Motor

PWM Pulse Width Modulation

SPDT Single Pole Double Throw

SPI Serial Peripheral Interface Bus

SMD Surface Mount Device

SSR Solid State Relais

TI Texas Instruments

TRIAC Triode for Alternating Current

UART Universal Asynchronous Receiver/Transmitter

USB Universal Serial Bus

VAC Volts Alternating Current

Vrms Volts Root Mean Square

Kapitel 1

Problemstellung und Motivation

Im Zuge dieser Arbeit soll eine bestehende CNC[1]-Portalfräse[2] (Siehe Abbildung 1.1) für automatisierten und rechnerfernen Betrieb aufgerüstet und erweitert werden. Konkret wurde hierfür ein Raumportal (Analog zum mehrachsigen, linearen, kartesischen Aufbau der Fräse selbst) angedacht, das über der CNC-Fräse angebracht wird und, ausgestattet mit einer Greifeinheit, zum Be- und Entladen von Werkstücken und Werkzeugen verwendet wird.
In weiterer Folge soll mit diesem Aufbau eine semi-automatisierte, sukzessive Herstellung von mehreren Werkstücken erfolgen, ohne menschliches Eingreifen für die Einspannung und Be-/Entladung zu erfordern. So könnten beispielsweise die Polywürfel eines SOMA-Würfels[3] in Serie hergestellt werden.
Darüberhinaus soll der Aufbau des Portalroboters als Gestell bzw. Rahmen für die Portalfräse selbst dienen und einen Späneschutz (vorzugsweise in Form von automatisch öffenbaren Türmodulen oder eines Rolladensystems) integrieren.
Aufgrund des Ausmaßes dieser Aufgabe wird diese auf zwei parallel auszuarbeitende und ineinandergreifende Diplomarbeiten aufgeteilt.

[1] Computer Numerical Control
[2] High-Z S-1000/T CNC-Router von `https://www.cnc-step.de`
[3] `https://en.wikipedia.org/wiki/Soma_cube`

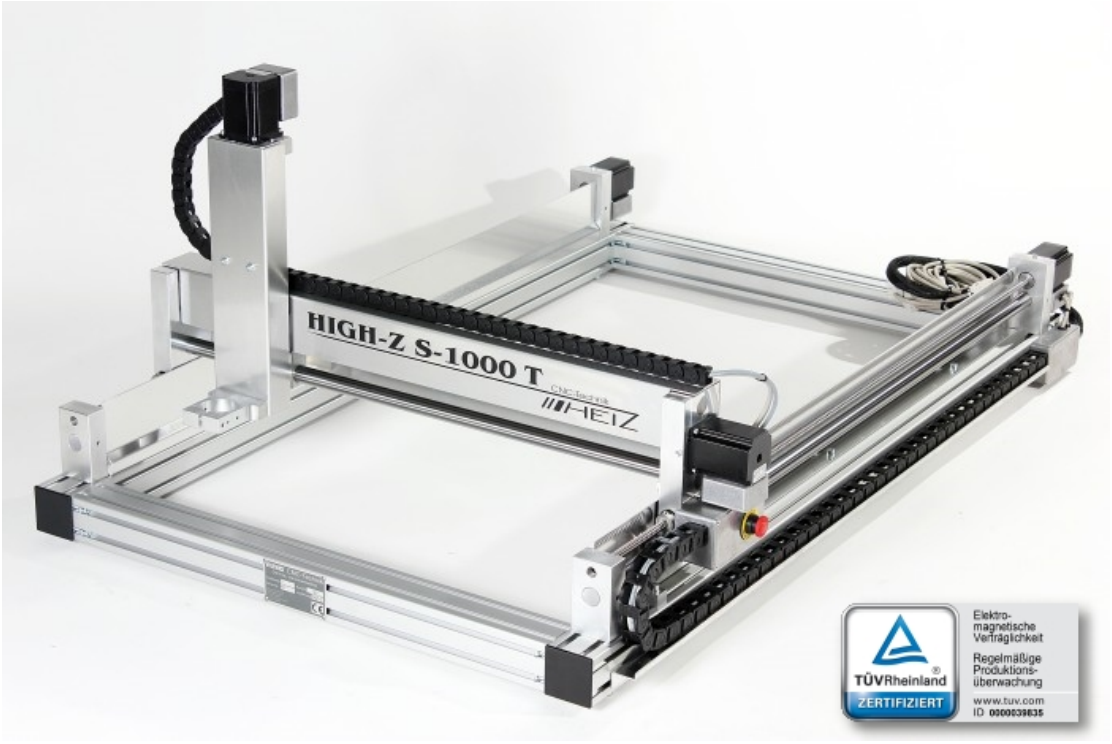


Abbildung 1.1: Portalfräse / CNC Fräse High-Z S-1000/T Kugelgewindetrieb 1000x600mm [1]

Aus dem Grund der Aufteilung des überspannenden Projekts in zwei Diplomarbeiten wird in dieser Arbeit bloß auf die folgenden Arbeitsschritte, die sich auf die Konstruktion eines 2-Achs Flächenportals inklusive Steuerelektronik und entsprechender Mikrocontroller-Firmware beschränken, eingegangen:

- Konstruktion der X-Achse
- Konstruktion der Y-Achse
- Motorauswahl, Antrieb, Sensorik
- Steuerelektronik und Positionsbestimmung
- Mögliche Softwareimplementierung

Der tatsächliche Aufbau des Portalroboters wird erst nach Abschluss der Arbeit starten, somit beruhen die Ausführungen in dieser Arbeit bisher nur auf Auslegungsrechnungen, Simulationen und bisherigen Erfahrungswerten.

Kapitel 2

Konzept und Alternativen

Zur Bewertung der Wirtschaftlichkeit des Vorhabens werden Alternativen zu einem Eigenbau eines Raumportals angedacht. Hierzu gehören folgende Produkte von Fremdfirmen:

- Festo 3D Raumportal
- igus robolink®
- igus Raumportal
- KUKA Roboterarm für niedrige Traglasten

2.1 Alternativsysteme

Im Folgenden werden die eben erwähnten Alternativen zu einer Neukonstruktion der Anlage kurz diskutiert.

2.1.1 Festo 3D Raumportal

Festo bietet als quasi-Baukastensystem ein 3D-Raumportal (kartesisch, 3 lineare Vorschubsachsen) an. Hierfür steht ein Online-Konfigurator[1] zur Verfügung, durch den die meisten der benötigten Funktionen definiert werden können. Laut diesem

[1] `https://www.festo.com/cat/de_de/products_YXCR`

bewegt sich der Preis für die benötigten Funktionen außerhalb des vereinbarten Budgets. Aus diesem Grund wird diese Option somit vorerst verworfen.
Nachteil hierbei ist, einige Funktionen separat entwickeln zu müssen. So zum Beispiel würde dazu noch ein Rahmen, der das Raumportal selbst und wie geplant die CNC-Fräse unterstützt, sowie ein Späneschutz, wie auch geartet, benötigt werden.

2.1.2 igus robolink

Modulare Roboter in Form eines Roboterarmes von igus, angetrieben über Seilzüge durch Schrittmotoren, schienen ursprünglich aufgrund ihres niedrigen Preises für ein 5-Achs-System und ihrer Modularität eine optimale Lösung zu bieten. Bei näherer Betrachtung und nach Rücksprache mit igus liegt die realistische Nutzlast des Roboterarms bei nur ungefähr 1 kg. Darüberhinaus ist die Reichweite des Roboterarms zu gering, um den gesamten Arbeitsraum der CNC-Fräse zu bedienen. Rahmen und Späneschutz müssen auch hier getrennt entworfen werden.

2.1.3 igus Raumportal

Auch igus bietet ein modulares Raumportal an. Nach Anfrage für ein Sonder-Raumportal liegt eine derartige Lösung im vereinbarten Budget. Igus betreibt diese Raumportale über Zahnriemen auf X- und Y-Achse und (selbsthemmende) Trapezgewindespindeln auf der Z-Achse. Durch die Verwendung von Zahnriemen werden hier Einbuße bei der erzielbaren Genauigkeit und in weiterer Folge der Laufruhe gemacht. Darüberhinaus ist die Linearführung direkt in diese *igus Zahnriemenachsen*[1] integriert und ist nur für geringe zu bewegende Lasten ausgelegt.

2.1.4 KUKA Roboter für niedrige Traglasten

Eine Anfrage bei KUKA Robotics hat ergeben, dass ein Roboterarm für Traglasten bis 5 kg (was zugegebenermaßen bereits sehr eng dimensioniert ist) ohne hierfür bereitgestellte Schulung für Universitäten bereits weit außerhalb des vereinbarten

[1] `http://www.igus.de/wpck/2370/drylin_zlw`

Budgets liegt und die Traglast mit 5 kg-6 kg zu eng dimensioniert ist. (Die Traglast des Roboters muss, um die effektive Nutzlast der Gesamteinheit zu errechnen, noch um das Gewicht einer Greifeinheit und Befestigungsplatte vermindert werden) Deshalb wird auch diese Option vorerst verworfen.

2.2 Eigenbau

Für einen Eigenbau wird eine Kostenabschätzung von ungefähr 10000 € angesetzt, wobei ein Großteil hiervon auf den auch bei den anderen Alternativen notwendigen Rahmen sowie einem Rollladen als Späneschutz fallen. Weiters kann hierbei die Steuerelektronik sowie die Funktion des gesamten Systems so konzipiert werden, dass alle Vorgaben hochintegriert erreicht werden.
Im Folgenden wird demnach der Aufbau eines Portalroboters (In dieser Arbeit beschränkt sich dieser auf ein Flächenportal) als Eigenkonstruktion sowie das Design einer proprietären Steuerelektronik verfolgt.

Kapitel 3

Flächenportal

In diesem Kapitel wird die Konstruktion und Auswahl der Komponenten zur Konstruktion eines Flächenportals, das bedeutet in diesem Fall eine Vorrichtung zum Verschub einer Endplatte[1] auf zwei unabhängigen Linearachsen, beschrieben.

3.1 Zwei-Achs-Roboter

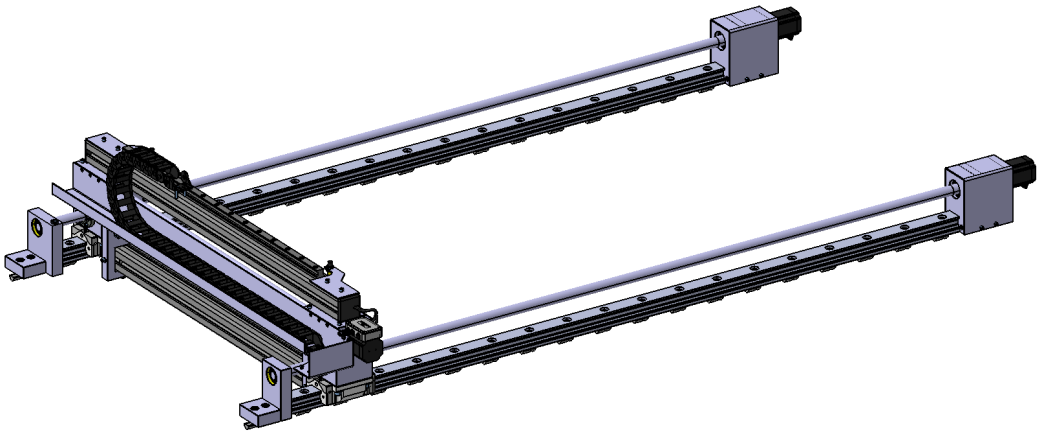


Abbildung 3.1: Fertiges Flächenportal ohne Rahmen

[1]Auf dieser können in weiterer Folge zusätzliche Module wie beispielsweise eine Achse in vertikaler Richtung (Z-Achse) befestigt werden

Das Flächenportal besteht aus zwei voneinander kinematisch unabhängigen Vorschubsachsen, die einen kartesischen Verfahrweg in zwei Richtungen, im Folgenden genannt *X-Achse* und *Y-Achse*, ermöglichen. Abbildung 3.1 zeigt die endgültige Konstruktion des Flächenportals.

Auf diesem Flächenportal, genauer gesagt auf der Y-Achse des Portals, kann zur Erweiterung, und damit den Inhalt des zweiten Teils dieser Diplomarbeit referenzierend, eine weitere Achse angebracht werden, die einen Verschub, beispielsweise einer pneumatischen Greifeinheit, in vertikaler Richtung (Z-Achse) ermöglicht.

Der in Abbildung 3.1 dargestellte Teil des Portalroboters soll in der Folge auf einem Rahmen aus Aluminiumprofilen befestigt werden.

Der Vorschub und somit die winkelgetreue Übersetzung der Rotationsbewegung der Motoren in eine Translationsbewegung wird über Kugelgewindetriebe (Kugelgewindespindeln) realisiert. Hierbei wird eine Gewindespindel angetrieben, auf der eine gegen Verdrehung gesicherte Mutter axial bewegt wird. [2, S. 724] Diese ermöglichen eine präzise Positionierung bei einem verhältnismäßig hohen Wirkungsgrad ($\eta \geq 95\,\%$ [3]) und vertretbaren Kosten.[1]

Die Lagerung der Kugelgewindetriebe erfolgt durch eine Fest-Los-Lagerung analog der von SKF angebotenen Zubehör-Stehlagereinheiten. (PLBU Stehlager und BUF Stehlager[2]). In Lagerblöcken wird hier einerseits die Loslagerung mit einem im Lagerblock axial verschieblichen Rillenkugellager (Loslager), die Festlagerseite andererseits mit einem nicht verschieblichen doppelreihigen Schrägkugellager (Festlager) ausgeführt.

Eine kostengünstige Alternative zu Kugelgewindetrieben sind Trapezgewindetriebe. Diese haben weiters die vorteilhafte Eigenschaft (dies findet beispielsweise bei vertikal stehenden Achsen Anwendung), selbsthemmend zu wirken, haben jedoch den erheblichen Nachteil eines geringeren Wirkunsgrades: Kugelgewindetriebe be-

[1] `http://www.skf.com/de/products/linear-motion/ball-and-roller-screws/ball-screws/index.html`

[2] `http://www.skf.com/de/products/linear-motion/ball-and-roller-screws/accessories/support-bearing-for-precision-rolled-ball-screws/index.html`

sitzen einen Wirkungsgrad von $\eta \geq 98\,\%$. Hier gegenüber besitzen Trapezgewindespindeln nur einen Wirkungsgrad von ungefähr 50 %-60 %. [2, S. 724]
Die Gewindespindel wird mit dem Motorzapfen über eine, der Kompaktheit der Konstruktion und aus Einbaugründen wegen, geteilt ausgeführten Elastomer Klauenkupplung (R+W, Modell EKH10[1]) verbunden.
Die X-Achse wird weiterhin aus konstruktiven Gründen zweiachsig ausgeführt. Hiermit ist gemeint, dass als Antrieb zwei parallele Kugelgewindetriebe und zwei identische Motoreinheiten verwendet werden.
Um zu vermeiden, dass die Kugelgewindespindeln/-triebe Kräfte quer zu ihrer Achse aufnehmen (sei dies durch Trägheitskräfte der zu bewegenden Masse oder die Gewichtskraft), wird weiterhin die Last über die gesamte Achslänge durch kugelgelagerte Linearführungen (*Profilschienenführungen*) unterstützt.

3.1.1 Geometrische Beziehungen von Kugelgewindetrieben

Die geometrischen Zusammenhänge des Kugelgewindetriebes werden im Folgenden erläutert. Hierbei ist s der Translationsweg, φ der entsprechende Motordrehwinkel und p die Steigung des Kugelgewindetriebs.

$$s = p\frac{\varphi}{2\pi} \tag{3.1}$$

und somit durch ein- und zweimaliges Ableiten mit $v = \dot{s}$, $a = \ddot{s}$ und $\omega = \dot{\varphi}$:

$$v = p\frac{\omega}{2\pi} \tag{3.2}$$

$$a = p\frac{\omega}{2\pi} \tag{3.3}$$

Über eine Leistungsbilanz $P_{Antrieb} + P_{Abtrieb} = 0$ können weitere Zusammenhänge hergestellt werden. Die Antriebsleistung eines Motors ergibt sich mit M als momentanes Motordrehmoment zu:

$$P_{Motor} = M \cdot \omega \tag{3.4}$$

[1] http://www.rw-kupplungen.de/produkte/praezisionskupplungen/elastomerkupplungen/ekh.html

Die nötige Leistung zur Bewegung der Last ergibt sich:

$$P_{Last} = -F \cdot v \tag{3.5}$$

Wird angenommen, es wirke nur die Trägheitskraft auf die zu bewegende Masse und somit auf den Kugelgewindetrieb, kann $F = ma$ eingesetzt werden.

$$P_{Last} = -ma \cdot v \tag{3.6}$$

Unter Vernachlässigung weiterer Reibungseffekte in den Lagern ergibt sich mit einem Spindelwirkungsgrad der folgende Zusammenhang:

$$\eta \cdot P_{Motor} + P_{Last} = 0 \tag{3.7}$$

$$\eta \cdot M \cdot \omega = ma \cdot v \tag{3.8}$$

Mit den geometrischen Beziehungen eines Kugelgewindetriebes für die axiale Vorschubgeschwindigkeit:

$$\eta \cdot M \cdot \omega = F \cdot p\frac{\omega}{2\pi} \tag{3.9}$$

Umgeformt:

$$F = \frac{\eta \cdot M \cdot 2\pi}{p} \tag{3.10}$$

Oder anders ausgedrückt

$$M = \frac{F \cdot p}{\eta \cdot 2\pi} \tag{3.11}$$

Mit eingesetzter Trägheitskraft:

$$M = \frac{m \cdot a \cdot p}{\eta 2\pi} \tag{3.12}$$

$$M = \frac{m \cdot p^2}{\eta(2\pi)^2}\dot{\omega} \tag{3.13}$$

Durch Koeffizientenvergleich mit dem Drallsatz in der allgemeinen Form $M = J \cdot \dot{\omega}$ lässt sich für die axial zu bewegende Last ein Ersatz-Massenträgheitsmoment $J_{m,ers}$ angeben. Beaufschlagt man den Motor mit einer gedachten Last mit nicht verschwindendem Massenträgheitsmoment J, statisch und dynamisch ausgewuchtet auf die Drehachse des Motors, erfährt dieser dieselbe Belastung wie mit einer über den Kugelgewindetrieb zu bewegenden Last m:

$$J_{m,ers} = \frac{m \cdot p^2}{\eta\,(2\pi)^2} \tag{3.14}$$

Wird nun, im Gegensatz zu den vorherigen Betrachtungen, die Spindel selbst als nicht-masselos angenommen, erhalten wir, bei Betrachtung dieser als Vollzylinder mit dem effektiven Radius $r_{Sp,e}$ und der Masse m_{Sp} für die Kugelgewindespindel ein Trägheitsmoment

$$J_{Spindel} = \frac{1}{2} r_{Sp,e}^2 m_{Sp} \tag{3.15}$$

Das Massenträgheitsmoment des Rotors des Motors ergibt sich laut Datenblatt des verwendeten Modells (Nanotec ST5918L3008, [4]) zu:

$$J_{Motor} = 3 \times 10^{-5}\,\mathrm{kg\,m^2} \tag{3.16}$$

Das Massenträgheitsmoment der Kugellager wird weiterhin vernachlässigt.
Die Verlustmomente der Lager können nach [5] überschlägig ermittelt werden mit μ als Reibungszahl des Lagers, P als äquivalente dynamische Belastung und d als Lagerbohrungsdurchmesser:

$$M_{Verlust,Lager} = 0.5 \mu P d \tag{3.17}$$

Aufgrund der Tatsache, dass Schrittmotoren, wie der Name schon andeutet, mit Schritten betrieben werden, beginnt sich der Motor mit einer Startdrehzahl und nicht mit einer stetigen Bewegung zu drehen.
Deshalb sollte für eine rasche Beschleunigung das externe Trägheitsmoment maximal 20 fach im Bezug auf das Rotorträgheitsmoment des Motors gewählt werden. [6]
Für die X-Achse gilt pro Motor:

$$J_{ext} = J_{Spindel} + \frac{J_{m,ers}}{2} \tag{3.18}$$

Und damit die Bedingung

$$\frac{J_{ext}}{J_{Motor}} \leq 20 \tag{3.19}$$

Eingesetzt ergeben sich mit einer überschlagsmäßig angenommenen auf der X-Achse zu bewegenden Masse von $m = 100\,\mathrm{kg}$, einer Spindelsteigung von $p = 5\,\mathrm{mm}$, einem mittleren Gewindedurchmesser von $d_{Sp} \approx 23.5\,\mathrm{mm}$, einer Spindellänge von

$l_{Sp} = 2\,\mathrm{m}$, Stahl als Spindelmaterial mit der Dichte $\rho_{Sp} = 7850\,\frac{\mathrm{kg}}{\mathrm{m}^3}$ und einem konservativ angenommenen Wirkungsgrad des Lineargetriebes von $\eta = 90\,\%$:

$$J_{m,ers,X} \approx 7 \times 10^{-5}\,\mathrm{kg\,m^2} \tag{3.20}$$

$$J_{Spindel,X} \approx 47 \times 10^{-5}\,\mathrm{kg\,m^2} \tag{3.21}$$

Und somit für das gesamte (Ersatz-)Massenträgheitsmoment pro X-Achse:

$$J_{ext} \approx 51 \times 10^{-5}\,\mathrm{kg\,m^2} < 20 J_{Motor} \quad \checkmark \tag{3.22}$$

Für eine Spindelsteigung von $p = 10\,\mathrm{mm}$ ergibt sich:

$$J_{ext} \approx 61 \times 10^{-5}\,\mathrm{kg\,m^2} \approx 20 J_{Motor} \quad \checkmark \tag{3.23}$$

Da hierbei laut Motorhersteller die vorliegende Bedingung bloß eine Faustformel darstellt, wird hier jedoch auch eine kleine Überschreitung des Grenzwertes ohne weitere Untersuchungen zugelassen.
Eine Verringerung des Lineargetriebewirkungsgrades würde sich nach Gleichung (3.14) in einer Erhöhung des Ersatzträgheitsmoments der zu bewegenden Masse auswirken.

Wirkende Kräfte

Zur Berechnung der wirkenden Kräfte (Hier wird angenommen, dass alle auftretenden Kräfte in Achsrichtung aus den Trägheitskräften resultieren) wird die maximale Beschleunigung ermittelt. Diese tritt aufgrund der Drehzahlabhängigkeit des Motormoments $M_{Motor} = M_{Motor}(n)$ bei Anfahren aus dem Stillstand auf.
Die folgende Beziehung (Siehe Gleichung (3.3)) gilt weiterhin pro Kugelgewindetrieb:

$$a_{max} = p\frac{\dot{\omega}_{max}}{2\pi} \tag{3.24}$$

Um also die maximale Beschleunigung zu ermitteln, wird die maximale Winkelbeschleunigung $\dot{\omega}$ benötigt. Diese ergibt sich aus dem Drallsatz:

$$M_{Motor,max} = J_{ges}\dot{\omega}_{max} = (J_{ext} + J_{Motor})\dot{\omega}_{max} \tag{3.25}$$

Und somit:

$$a_{max} = p\frac{\frac{M_{Motor,max}}{J_{ext}+J_{Motor}}}{2\pi} \tag{3.26}$$

$$F_{axial,max} = m\, a_{max} \tag{3.27}$$

3.2 X-Achse

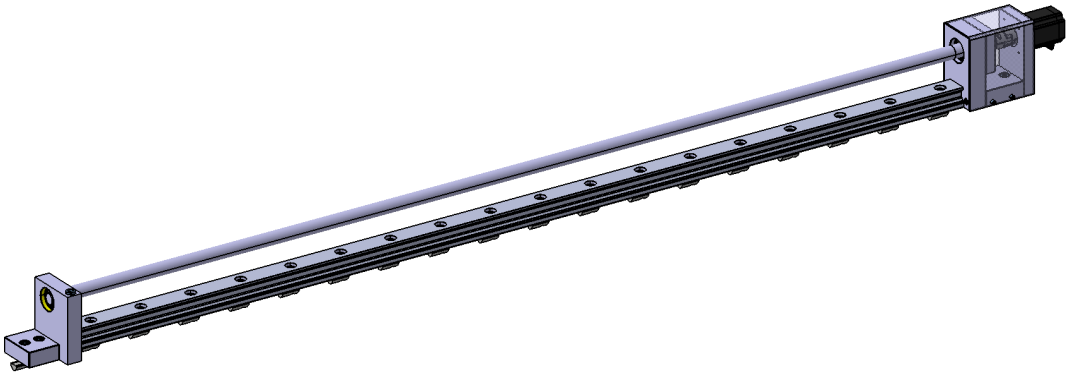


Abbildung 3.2: Ansicht der X-Achse ohne Rahmen und Linearführungsschlitten sowie Spindelmutter. Links: Loslager, Rechts: Festlager, Kupplung und Motor

Nachdem an der X-Achse zwei Motoren, zwei Spindeln und bloß eine gesamte zu bewegende Last gegeben sind, kann der Drallsatz für die X-Achse insgesamt angeschrieben werden als:

$$2M_{Motor} = (2J_{Motor} + 2J_{Spindel} + J_{m,ers})\dot{\omega} \tag{3.28}$$

Für die X-Achse gilt pro Motor:

$$J_{ext,X} = J_{Spindel} + \frac{J_{m,ers}}{2} \tag{3.29}$$

Eingesetzt ergeben sich mit einer überschlagsmäßig angenommenen auf der X-Achse insgesamt zu bewegenden Masse von 100 kg und einer Spindelsteigung von 5 mm nach Gleichung (3.22):

$$J_{ext,X} = J_{Spindel} + \frac{J_{m,ers}}{2} \approx 51 \times 10^{-5}\,\mathrm{kg\,m^2} \tag{3.30}$$

Die X-Achse wird analog zur Konstruktion der gekauften CNC-Portalfräse CNC-Step mit einem Kugelgewindetrieb und einem Schrittmotor ausgeführt.

3.2.1 Wirkende Kräfte

Die maximale Beschleunigung an der X-Achse ergibt sich nach Gleichung (3.26) mit einem maximalen Drehmoment des Motors nach [4] bei Anfahren aus dem Stillstand mit einer Spindelsteigung von $p = 5\,\text{mm}$ zu:

$$a_{max,X} \approx 2.78\,\tfrac{\text{m}}{\text{s}^2} \tag{3.31}$$

und somit die maximal jeweils auf eine Kugelgewindespindel in Achsrichtung wirkende Kraft

$$F_{axial,max,X} = 0.5 m_X a_{max,X} \approx 139\,\text{N} \tag{3.32}$$

Die Lagerstellen sowie die Linearführungen werden insgesamt weiters durch die Gewichtskraft der auf der X-Achse zu bewegenden Gesamtmasse belastet. Diese wird wieder überschlagsmäßig mit 100 kg angenommen. Somit ergibt sich:

$$F_G = mg = 100\,\text{kg} \cdot 9.81\,\tfrac{\text{m}}{\text{s}^2} = 981\,\text{N} \tag{3.33}$$

Die an der Y-Achse durch die Lineargetriebeeinheit (Kugelgewindetrieb) auftretende Axialkraft wird als Radialkraft (bzw. Kraft quer zur Spindelachse) von der X-Achse aufgenommen. (Siehe Abschnitt 3.3.1) Aufgrund der Tatsache, dass diese Kraft im Grunde von einer Fest-Fest-Lagerung aufgenommen wird, (zwei Linearführungen, *starr* gelagert, sowie Radiallager) wird der Einfachheit weiter angenommen, dass die gesamte Axialkraft der Y-Achse von einer Lagerstelle kompensiert wird. Dies entspricht dem schlechtest möglichen Fall und überschätzt die auftretende Belastung.

3.2.2 Berechnung Kugelgewindetrieb

Als Kugelgewindetrieb wird eine, im Vergleich zu hochpräzisen Konkurrenzprodukten, relativ preiswerte Spindel-Mutter-Kombination[1] vom Hersteller *Micron* verwendet. Die Belastung der Spindel wird wie folgt festgelegt:

[1] `http://lineareasy.de/shop/en/ball-screw/kgt-r-2505-rh-t5.html`

- Die maximale Belastung in axialer Richtung wird nach Abschnitt 3.2.1 angenommen.
- Die Belastung quer zur Spindelachse wird vernachlässigt. Diese Kraft wird zur Gänze durch die über die ganze Spindellänge vorhandene Linearführung aufgenommen. Einzig vorhandene Ungenauigkeiten und Toleranzen könnten hier Gegenteiliges bewirken.

Kritische Axiallast

Nach Herstellerangaben in [7] berechnet sich die kritische Axialkraft bei der verwendeten Kugelgewindespindel mit Nenndurchmesser 25 mm zu:

$$P_{KP} = \frac{102 K d^4}{\mu^2 l^2} \tag{3.34}$$

wobei P_{KP} die kritische Axiallast in kN, d der Kerndurchmesser der Spindel, l der Lagerabstand, μ ein Lagerungsabhängiger Faktor laut Angabe und K der reziproke Wert der geforderten Sicherheit $K = \frac{1}{S}$ ist.
Aufgrund fehlender Angaben bezüglich des lagerungsabhängigen Faktors μ wird das beiliegende Diagramm zur Ermittlung der kritischen Axiallast verwendet.
Bei einem Lagerabstand von 2000 mm, und somit im schlechtesten Fall bei Positionierung der Mutter am Anschlag des Loslagers (Bei Positionierung der Spindelmutter in der Mitte der Spindel wird diese durch die Konstruktion noch durch die vorhandene Linearführung unterstützt), ergibt sich bei einem Kugelgewindetrieb der Größe 25:

$$P_{KP} \approx 5.1\,\text{kN} \tag{3.35}$$

Kritische Drehzahl

Die kritische Drehzahl lässt sich nach Herstellerangaben mit

$$n_{kr} = 5 \cdot 10^7 \frac{d}{l^2} V k \qquad \text{und} \qquad n^*_{kr} = \frac{80000}{d} \tag{3.36}$$

ermitteln. [7]
Die maximal zulässige Drehzahl ist dann der geringere Wert von n_{kr} und n^*_{kr}.

Hierbei werden die Bezeichnungen der Parameter analog zur Ermittlung der kritischen Axiallast verwendet. V ist wieder ein lagerungsabhängiger Faktor.
Über ein beiligendes Diagramm wird ermittelt: $n_{kr} \approx 1100\,\frac{1}{\text{min}}$
Mit einem Kerndurchmesser von $d = 21.7\,\text{mm}$ ergibt sich:

$$n^*_{kr} = \frac{80\,000\,\frac{\text{mm}}{\text{min}}}{21.7\,\text{mm}} \approx 3686\,\frac{1}{\text{min}} \tag{3.37}$$

und somit

$$n_{kr,ges} = min(n_{kr},\, n_{kr}*) \approx 1100\,\frac{1}{\text{min}} \tag{3.38}$$

Dies entspricht mit einer Spindelsteigung von $p = 10\,\text{mm}$ einer maximalen Vorschubgeschwindigkeit in X-Richtung von

$$v_{max} = \frac{n \cdot p}{60\,\frac{\text{s}}{\text{min}}} \approx 183.333\,\frac{\text{mm}}{\text{s}} \tag{3.39}$$

Bei einer Spindelsteigung von $p = 10\,\text{mm}$ hingegen einer Vorschubgeschwindigkeit von

$$v_{max} = \frac{n \cdot p}{60\,\frac{\text{s}}{\text{min}}} \approx 91.667\,\frac{\text{mm}}{\text{s}} \tag{3.40}$$

Gleichung (3.38) gilt unter der Annahme, dass sich die Y-Achse gerade an einem Ende der X-Achse befindet und sich die ungestützte Spindellänge l daher zu ungefähr 2 m ergibt. In der Praxis wird die ungestützte Spindellänge jedoch um bis zu 50 % reduziert, da die Spindel auf der Höhe der Y-Achse durch die Linearführung gestützt wird.
Mit einer ungestützten Spindellänge von 1.5 m ergibt sich $n_{kr} = 2300\,\frac{1}{\text{min}}$ und bei einer ungestützten Spindellänge von 1 m (Mutter und Wagen der Linearführung befinden sich am halben Verfahrweg der X-Achse) eine kritische Drehzahl von $n_{kr,ges} = min(4000\,\frac{1}{\text{min}}, 3686\,\frac{1}{\text{min}}) = 3686\,\frac{1}{\text{min}}$. Dies ergibt unter Einbeziehung von Gleichung (3.37) eine maximale Vorschubgeschwindigkeit von $v_{max} \approx 383.333\,\frac{\text{mm}}{\text{s}}$ und $v_{max} \approx 614.333\,\frac{\text{mm}}{\text{s}}$ respektive.

Lebensdauer

Die Lebensdauer wird aufgrund fehlender Daten in der Ausführung von *Micron* basierend auf [8], aber weiterhin mit den von Micron bereitgestellten dynamischen und statischen Tragzahlen [7] berechnet.

Die Lebensdauer in Umdrehungen ergibt sich zu:

$$L = (\frac{C_{dyn}}{f_W \cdot F})^3 \cdot 10^6 \tag{3.41}$$

mit F als Axiallast, f_W einem Belastungsfaktor abhängig vom Anwendungsfall und C_{dyn} als dynamische Tragzahl.
Unter der Annahme, dass die Last durch $i^* = 2$ Gewindegänge getragen wird, (Hierbei wird aufgrund nicht weiter bekannter Bedingungen im Betrieb auf der sichersten Seite gerechnet) ergeben sich die Tragzahlen einer 25x5 *Micron* Gewindespindel zu $C_{dyn} = 9.3\,\text{kN}$ und $C_{0,stat} = 14\,\text{kN}$. [7]
Mit einer maximal zulässigen Vorschubgeschwindigkeit von $0.67\,\frac{\text{m}}{\text{s}}$ wird nach [8] ein Belastungsfaktor von $f_W = 1.4$ verwendet.
Die Lebensdauer in Umdrehungen ergibt sich mit Gleichung (3.32) zu:

$$L = (\frac{9.3\,\text{kN}}{1.4 \cdot F_{axial,max,X}})^3 \cdot 10^6 > 10^{11} \quad \checkmark \tag{3.42}$$

Es wird also Dauerfestigkeit angenommen.

Statischer Sicherheitsfaktor

Zur Verhinderung von plastischer Verformung der Wälzkörper oder deren Lauffläche ergibt sich folgende Sicherheit [7]:

$$f_S = \frac{C_{0,stat}}{F_{axial,max,X}} \approx 100 \quad \checkmark \tag{3.43}$$

3.2.3 Lebensdauer Profilschienenführung

Analog zu Wälzlagern und Kugelgewindetrieben wird auch bei Profilschienenführungen die Lebensdauer berechnet. Hierbei wird die nominelle Lebensdauer L in Vielfachen von 100 km angegeben [9, S. 9]:

$$L = (\frac{C_{dyn}}{P})^p \tag{3.44}$$

mit C als dynamische Tragzahl, P als dynamisch äquivalente Belastung und p als Lebensdauerexponent (Bei kugelgelagerten Profilschienenführungen, wie den hier verwendeten, ist $p = 3$).

Für die Belastung aufgrund der Gewichtskraft gilt näherungsweise $P_g = m_X g = 981\,\mathrm{N}$. Obwohl die Belastung auf zwei Profilschienenführungen aufgeteilt wird, wird zur Berechnung des unsichersten auftretenden Falles angenommen, dass eine Profilschienenführung das gesamte Gewicht trägt. (Schlitten und Mutter der Y-Achse sind an einem der beiden Enden positioniert) Weiters wird die Belastung aufgrund der Gewichtskraft um die Belastung durch Trägheitseffekte der auf der Y-Achse bewegten Masse erhöht. Diese werden näherungsweise (aufgrund überschlagsmäßig verringerter Masse um 30 % und gleicher Spindelsteigung von 5 mm) mit Gleichung (3.70) analog zur Berechnung der Trägheitskräfte in X-Richtung folgend angenommen:

$$P_{dyn} = F_{axial,max,Y} \approx 472\,\mathrm{N} \tag{3.45}$$

$$P = \sqrt{P_g^2 + P_{dyn}^2} \approx 1089\,\mathrm{N} \tag{3.46}$$

Für die hier verwendeten Profilschienenführungen HRC45 von CPC ergibt sich die dynamische Tragzahl zu $C_{dyn} = 71.3\,\mathrm{kN}$ und die statische Tragzahl zu $C_{0,stat} = 122.1\,\mathrm{kN}$. [9, S. 25]
Und somit die Lebensdauer zu:

$$L \cdot 100\,\mathrm{km} = (\frac{71.3\,\mathrm{kN}}{1089\,\mathrm{N}})^3 \cdot 100\,\mathrm{km} \approx 2.8 \times 10^7\,\mathrm{km} \quad \checkmark \tag{3.47}$$

Diese Linearführung ist also mehr als ausreichend dimensioniert.
Die Profilschienenführung HRC45 wurde hier trotz der Überdimensionierung verwendet, da die Breite des Schlittens zur einfachen Montage und Konstruktion notwendig war. Weiters hätte eine kleinere Ausführung einer Profilschienenführung, das bedeutet mit geringerer Tragfähigkeit, keinen nennenswerten Preisvorteil gebracht.
Aufgrund der relativ langsamen Bewegung bei Hebevorgängen der Z-Achse wird quasistatische Belastung angenommen. Würden hier dynamische Effekte auftreten, wäre die Trägheitskraft des Vorschubs einer Masse in Z-Richtung noch zu berücksichtigen.

Die statische Sicherheit gegen nennenswerte bleibende Verformung des Wälzkontaktes ergibt sich somit zu:

$$S_0 = \frac{C_{0,stat}}{P_{0,stat}} = \frac{122.1\,\mathrm{kN}}{1089\,\mathrm{N}} \approx 112.1 \quad \checkmark \tag{3.48}$$

Was ebenfalls weit ausreichend dimensioniert ist.

3.2.4 Lebensdauer Wälzlager

Zur Berechnung der Lebensdauer und Festigkeit der Wälzlager wird eine vereinfachte Berechnung nach [10, S. 64] verwendet. Es werden an der X- und Y-Achse ausschließlich lebensdauergeschmierte, abgedichtete einreihige Kugellager und doppelreihige Schrägkugellager verwendet.
Die nominelle Lebensdauer bei 90 % Erlebenswahrscheinlichkeit ergibt sich somit analog zu vorherigen Berechnungen:

$$L_{10} = (\frac{C}{P})^p \tag{3.49}$$

und die statische Tragsicherheit, also die Sicherheit gegen bleibende Verformung im Wälzkontakt:

$$s_0 = \frac{C_0}{P_0} \tag{3.50}$$

Bestimmung der Lebensdauer

Da die genaue Belastung der Wälzlager bis auf die Axialkräfte nicht bekannt ist, weil die Aufteilung der Gewichtskraft der zu bewegenden Masse auf Wälzlager und Profilschienenführung unbekannt ist, wird, um erneut auf der sicheren Seite zu rechnen, analog der Profilschienenführung, angenommen, ein einziges Lager nähme jeweils die Gesamtheit der Gewichts- und Trägheitskräfte auf.

Loslager Die Belastung des Loslagers ist somit die Resultierende aus der Gewichtskraft der Masse und der Trägheitskraft durch Bewegungen quer zur Wellenachse, also Bewegungen der Y-Achse.

Wieder angenommen mit $m = 100\,\text{kg}$ und nach Gleichung (3.45) für die rein radiale Belastung des Kugellagers:

$$P = P_0 = F_r = \sqrt{F_{axial,max,Y}^2 + F_g^2} = \sqrt{(472\,\text{N})^2 + 981\,\text{N})^2} \approx 1089\,\text{N} \tag{3.51}$$

Hier wird ein Kugellager 6004-2RSH von SKF verwendet: $C = 9.95\,\text{kN}$ und $C_0 = 5\,\text{kN}$. [10, S. 354]

$$L_{10} = (\frac{C}{P})^p = (\frac{9.95\,\text{kN}}{1089\,\text{N}})^3 \approx 763 \quad \checkmark \tag{3.52}$$

$$s_0 = \frac{5\,\text{kN}}{1089\,\text{N}} \approx 4.59 \quad \checkmark \tag{3.53}$$

Sowohl die Lebensdauer als auch die Sicherheit gegen bleibende Verformung sind mit 763 Millionen Umdrehungen und einer statischen Sicherheit von über 4 ausreichend dimensioniert.

Festlager Die Belastung des Festlagers ist somit eine Kombination der gesamten Gewichtskraft, Trägheitskräfte durch Vorschub in Y-Richtung (also der auf das Loslager wirkenden Radialkraft) und der wirkenden maximalen Axialkraft.
Bei einer Kombination aus Axial- und Radialkraft als Belastung bei doppelreihigen Schrägkugellagerung wird die äquivalente dynamische Belastung nach [10, S. 492] berechnet:

$$\frac{F_a}{F_r} \leq e \rightarrow P = F_r + Y_1 F_a \tag{3.54}$$

$$\frac{F_a}{F_r} > e \rightarrow P = X F_r + Y_2 F_a \tag{3.55}$$

Die statisch äquivalente Belastung wird hingegen folgend berechnet:

$$P_0 = F_r + Y_0 F_a \tag{3.56}$$

Für die hier verwendete Serie 33A gilt: $e = 0.8$, $X = 0.63$, $Y_1 = 0.78$, $Y_2 = 1.24$ und $Y_0 = 0.66$. [10, S. 494]
Wieder angenommen mit $m = 100\,\text{kg}$ und einer Spindelsteigung von $p = 5\,\text{mm}$ nach Gleichung (3.32) und Gleichung (3.51):

$$F_r = 1089\,\text{N} \tag{3.57}$$

$$F_a = ma_{max} = F_{max} = 147\,\text{N} \tag{3.58}$$

Hier wird ein Kugellager 3304A-2RS1 von SKF verwendet: $C = 23.4\,\mathrm{kN}$ und $C_0 = 14.6\,\mathrm{kN}$ [10, S. 526]

$$\frac{F_a}{F_r} \approx 0.146 < e \tag{3.59}$$

und daher

$$P = F_r + Y_1 F_a = 1089\,\mathrm{N} + 0.78 \cdot 147\,\mathrm{N} = 1204\,\mathrm{N} \tag{3.60}$$

$$P_0 = F_r * Y_0 F_a = 1089\,\mathrm{N} + 0.66 \cdot 147\,\mathrm{N} = 1186\,\mathrm{N} \tag{3.61}$$

$$L_{10} = (\frac{C}{P})^p = (\frac{23.4\,\mathrm{kN}}{1204\,\mathrm{N}})^3 \approx 7341 \quad \checkmark \tag{3.62}$$

$$s_0 = \frac{14.6\,\mathrm{kN}}{1186\,\mathrm{N}} = 12.3 \quad \checkmark \tag{3.63}$$

Sowohl die Lebensdauer als auch die Sicherheit gegen bleibende Verformung sind mit 7341 Millionen Umdrehungen und einem Wert von über 12 ausreichend dimensioniert.

3.2.5 Kupplung

Zur Anbindung der Kugelgewindespindel an den Zapfen des Motors wurde eine Elastomer-Klauenkupplung (*EKH10A* von *R+W*) verwendet. Diese ist die Kleinstausführung der von *R+W* angebotenen Elastomerkupplungen und bietet ein Nenndrehmoment von $12.6\,\mathrm{N\,m}$ und ist somit für den verwendeten Schrittmotor mit einem Haltemoment (maximalen Drehmoment) von $1.87\,\mathrm{N\,m}$ geeignet. [11]

$$1.87\,\mathrm{N\,m} < 12.6\,\mathrm{N\,m} \quad \checkmark \tag{3.64}$$

3.2.6 Dynamik

Um das (schnellste) dynamische Verhalten der X-Achse mit zu bewegender Last zu berechnen, wird eine Simulation in *MATLAB Simulink* vorgenommen.
Hierzu wird die Kennlinie der verwendeten Motoren benötigt. Diese ist jedoch bloß als Grafik verfügbar (Siehe Abbildung 3.3). Um den Inhalt dieser Grafik nach *MATLAB Simulink* zu übertragen, wird die Funktion im Linear-Logarithmischen

Maßstab abschnittsweise durch im einfach logarithmischen Maßstab lineare Funktionen $y(log(x)) = k * log(x) + d$ angenähert. Dies wird auf diese Weise vorgenommen, da sich, wie in Abbildung 3.3 zu sehen, in der Linear-Logarithmischen Darstellung beinahe-lineare Abschnitte ergeben. Weiters wird an den Übergängen Stetigkeit gefordert.

Dies führt, basierend auf der Motorkennlinie für 4.2 A, 24 V und parallele Beschaltung näherungsweise auf folgenden Zusammenhang:

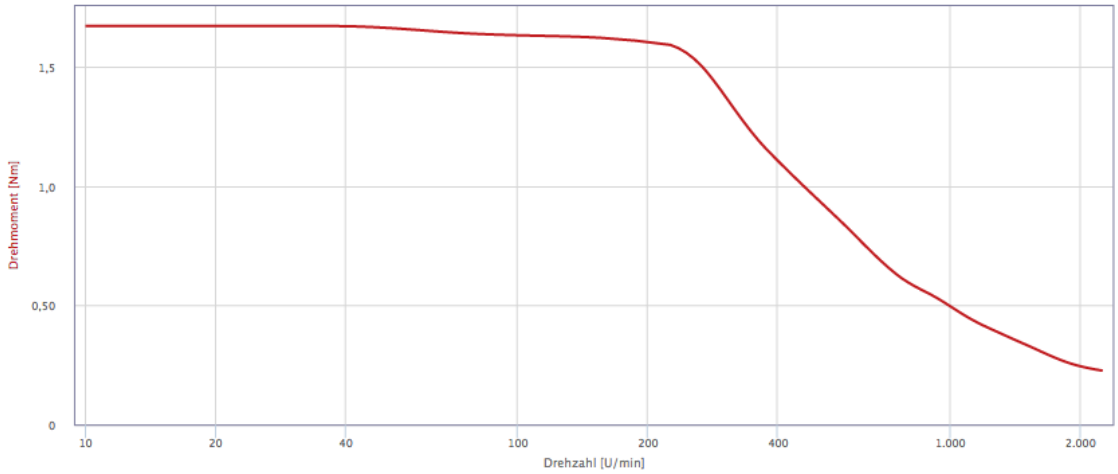


Abbildung 3.3: Motorkennlinie für das Modell ST5918L3008-A [12]

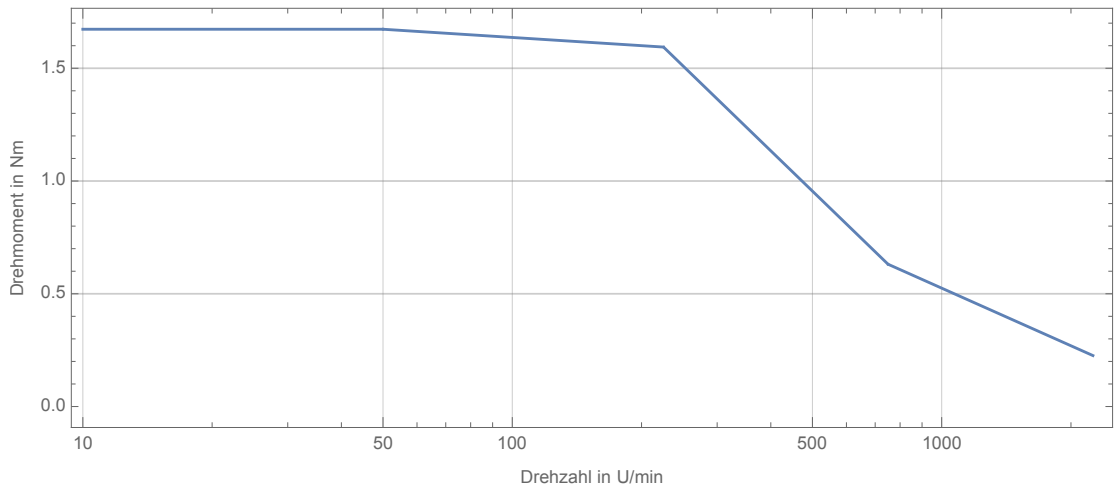


Abbildung 3.4: Angenäherte Motorkennlinie

Diese angenäherte Funktion wird nun als 1-dimensionaler Lookup-Table (Siehe *Motorkennlinie* in Abbildung 3.5) in MATLAB Simulink eingepflegt.

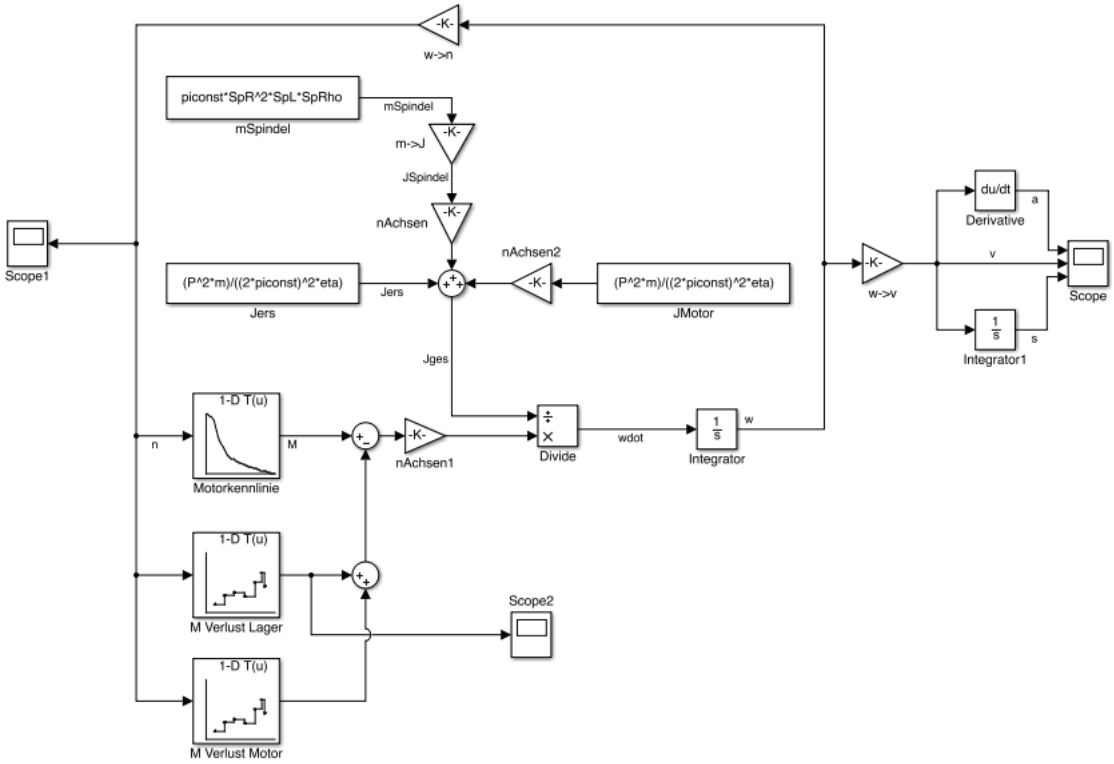


Abbildung 3.5: Graphische Darstellung des MATLAB Simulink Simulationsmodells

Die Last des Motors wird mit der vereinfachten Beziehung über ein Ersatzmassenträgheitsmoment der tatsächlich zu bewegenden Last, der Rotor-Trägheitsmomente des Motors und der Massenträgheitsmomente der Spindeln dargestellt.
Die Reibverluste des Kugelgewindetriebes werden wie in Abschnitt 3.1 über den Wirkungsgrad in einer Leistungsbilanz im Ersatzträgheitsmoment der zu bewegenden Masse berücksichtigt.
Weiters muss nun eine Abschätzung der Reibverluste in den Lagern getroffen werden. (Siehe auch Gleichung (3.17))

$$M_{Verlust,Lager} = 0.5\mu\, P\, d$$

Die Reibungszahl des Lagers μ ist für nicht abgedichtete Rillenkugellager $\mu = 0.0014$ und für doppelreihige Schrägkugellager $\mu = 0.0024$. [5] Aufgrund des Fehlens einer Formel zur überschlägigen Abschätzung des Lagerreibmoments für abgedichtete Lager wird diese Näherung auch im vorliegenden Fall verwendet. P ist

hierbei die äquivalente Belastung des Lagers und d der Bohrungsdurchmesser. Die Reibverluste eines einzelnen hier verwendeten Lagers unter der vorhandenen Belastung bewegen sich hier ungefähr in der Größenordnung von 0.02 N m-0.03 N m. Die Verlustmomente der Lager werden derart modelliert, dass ebenfalls über einen 1D-Lookup-Table eine Abhängigkeit des auftretenden Verlustmoments und der momentanen Drehzahl entsteht. Dies wird so ausgeführt, dass das Verlustmoment bei Drehzahlen größer als $10\,\frac{1}{\mathrm{min}}$ dem berechneten, überschlägigen Verlust-Reibmoment entspricht und bei kleineren Drehzahlen dem doppelten Wert. Hierdurch wird ein erhöhtes Anfahr- bzw. Losdrehmoment des Lagers simuliert.
Das Losdrehmoment des Motors ist im Datenblatt gegeben.
Das drehzahlabhängige Verlustmoment des Motors wird derart realisiert, dass wiederum über einen 1D-Lookup-Table das Losdrehmoment bis zu einer Drehzahl von $10\,\frac{1}{\mathrm{min}}$ dem im Datenblatt angegebenen *Detent Torque* entspricht. Danach wird angenommen, dass keine weiteren Verluste im Motor auftreten.
Durch diese Modellierungen wird nicht nur der Effekt berücksichtigt, dass, sobald ein reibungsbehafteter Körper einmal in Bewegung ist, weniger Kraft benötigt wird, um diesen in Bewegung zu halten, sondern auch die Auswirkungen eines Motors mit Permanentmagneten im unbestromten Zustand.
Bei Anfahren aus dem Stillstand mit Volleistung des Motors stellt sich also ein Drehzahlverlauf nach Abbildung 3.6 ein.
In diesem Modell wird basierend auf dem Drallsatz in der Form

$$\dot{\omega} = \frac{M}{J} \tag{3.65}$$

$$\dot{\omega} = \frac{M_{Motor} - M_{Verlust}}{J_{ges}} \tag{3.66}$$

das Signal $\dot{\omega}$ über einen Integrator mit dem Startwert $\omega(0) = 0$ in ω übergeführt und daraus wiederum über den geometrischen Zusammenhang $\omega = \frac{2\pi n}{60}$ mit n in $\frac{1}{min}$ das aktuelle Motor- und Verlustmoment in Abhängigkeit ebendieser Drehzahl berechnet.
Aus dem Signal ω kann nun direkt über die in Abschnitt 3.1 hergeleiteten geometrischen Beziehungen für Kugelgewindetriebe die Vorschubgeschwindigkeit des

Schlittens berechnet werden. Durch erneute zeitliche Integration und Differentiation kann die Vorschubgeschwindigkeit mit entsprechendem Startwert $s(0) = 0$ zur Darstellung in Position und Momentanbeschleunigung übergeführt werden.

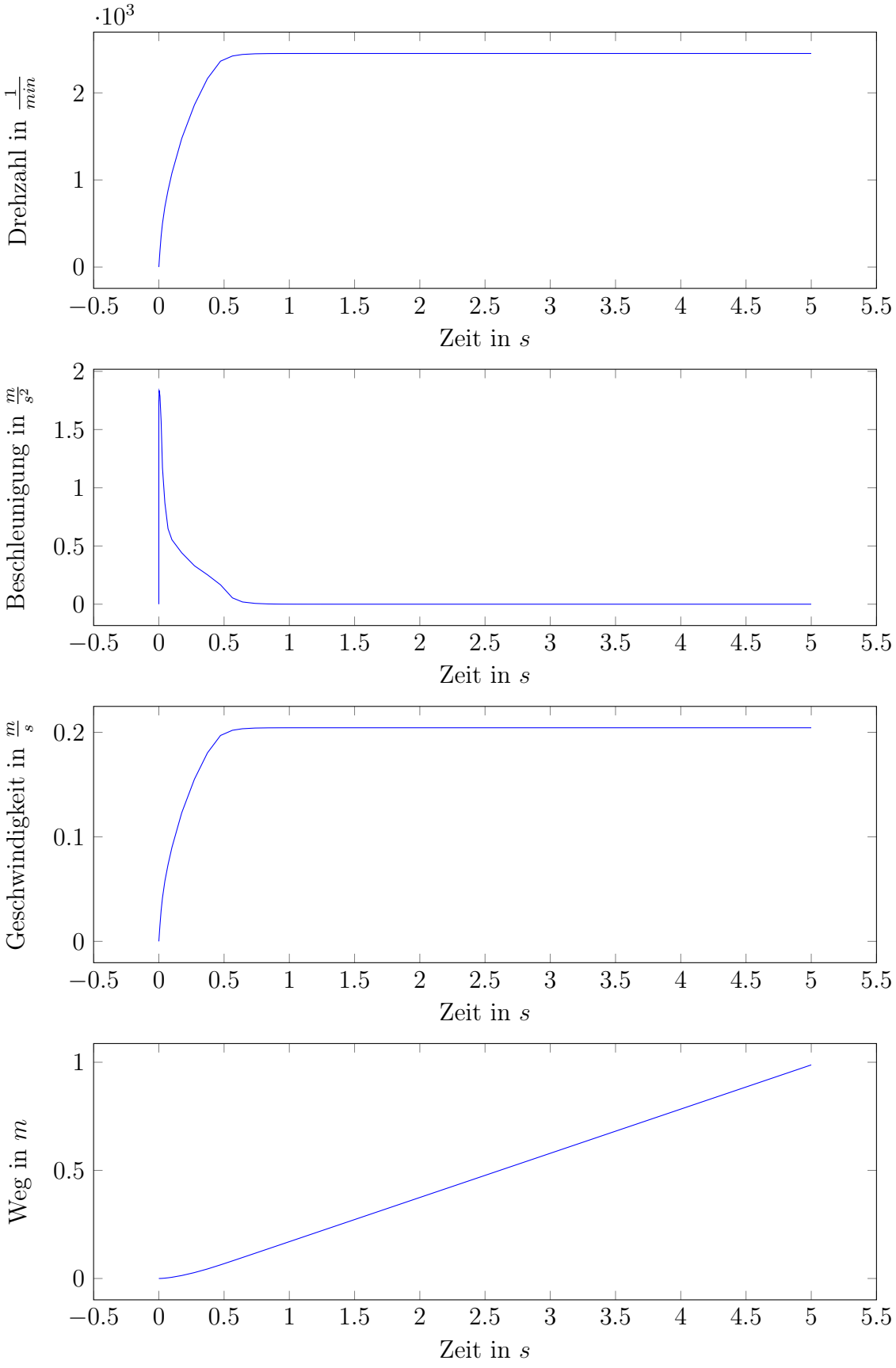


Abbildung 3.6: Verlauf der Bewegungsgrößen an der X-Achse

3.3 Y-Achse

Die Berechnung der Lebensdauer und statischen Sicherheiten der an der Y-Achse verwendeten Komponenten wird entsprechend der Berechnung an der X-Achse durchgeführt.

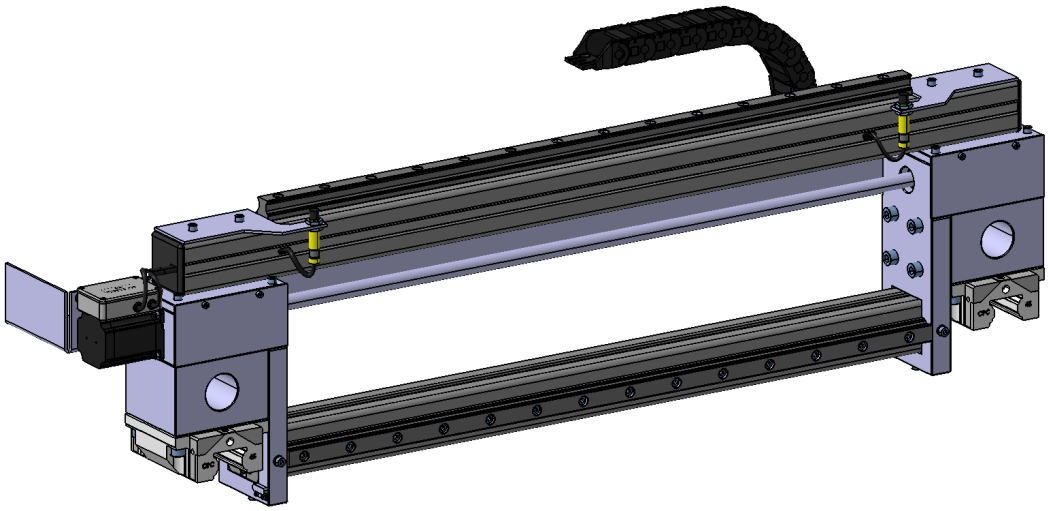


Abbildung 3.7: Konstruktion der Y-Achse

Nachdem an der Y-Achse ein Motor, eine Spindel und eine zu bewegende Last gegeben sind, kann der Drallsatz für die Y-Achse insgesamt angeschrieben werden als:

$$M_{Motor} = (J_{Motor} + J_{Spindel} + J_{m,ers})\dot{\omega} \tag{3.67}$$

Hier gilt insgesamt:

$$J_{ext,Y} = J_{Spindel} + J_{m,ers} \tag{3.68}$$

Eingesetzt ergeben sich mit einer überschlagsmäßig angenommenen auf der Y-Achse insgesamt zu bewegenden Masse von 70 kg und einer Spindelsteigung von 5 mm:

$$J_{ext,Y} = J_{Spindel} + J_{m,ers} \approx 8 \times 10^{-5}\,\mathrm{kg\,m^2} \tag{3.69}$$

3.3.1 Wirkende Kräfte

Zur überschlagsmäßigen Berechnung der auftretenden Belastung an den Lagerungen und Lineargetrieben der Y-Achse wird eine der X-Achse entsprechende Spin-

delsteigung von $p = 5\,\text{mm}$ sowie eine entlang dieser zu bewegenden Masse von $m_Y \approx 0.7 m_X = 70\,\text{kg}$ angenommen.
Mit Gleichung (3.31) ergibt sich für die maximale Axialkraft auf die Kugelgewindespindel und die Festlagereinheit:

$$F_{axial,max,Y} = m_Y a_{max,Y} \approx 472\,\text{N} \tag{3.70}$$

Die Belastung, die auf die Linearführungen an der Y-Achse und die Wälzlager wirkt, errechnet sich aus der zu bewegenden Masse mit der Erdbeschleunigung:

$$F_G = m_Y g = 686.7\,\text{N} \tag{3.71}$$

3.3.2 Berechnung Kugelgewindetrieb

Der Kugelgewindetrieb an der Y-Achse wird wieder unter den Annahmen von Abschnitt 3.2.2 berechnet.

Kritische Axiallast

Die kritische Axiallast ergibt sich nach [13] für den hier verwendeten Kugelgewindetrieb der nominellen Größe 16 und einer Länge von $l_{Sp} \approx 1\,\text{m}$ zu:

$$P_{KP} \approx 4\,\text{kN} \quad \checkmark \tag{3.72}$$

Kritische Drehzahl

Die maximal zulässige Drehzahl ist wiederum der geringere Wert von n_{kr} und n^*_{kr}. [13]
Über ein beiligendes Diagramm wird ermittelt: $n_{kr} \approx 1700\,\frac{1}{\text{min}}$
Mit einem Kerndurchmesser von $d = 12.7\,\text{mm}$ ergibt sich:

$$n^*_{kr} = \frac{80\,000\,\frac{\text{mm}}{\text{min}}}{12.7\,\text{mm}} \approx 3000\,\frac{1}{\text{min}} \tag{3.73}$$

und somit

$$n_{kr,ges} = min(n_{kr},\, n^*_{kr}) \approx 1700\,\frac{1}{\text{min}} \tag{3.74}$$

Bei einer Spindelsteigung von $p = 5\,\text{mm}$ entspricht dies einer Vorschubgeschwindigkeit von

$$v_{max} = \frac{n \cdot p}{60\,\frac{\text{s}}{\text{min}}} \approx 141.667\,\frac{\text{mm}}{\text{s}} \tag{3.75}$$

Gleichung (3.74) gilt unter der Annahme, dass sich die Mutter der Y-Achse gerade an einem Ende des Verfahrweges befindet und sich die ungestützte Spindellänge l daher zu ungefähr 1 m ergibt. In der Praxis wird die ungesützte Spindellänge jedoch um bis zu 50 % reduziert, da die Spindelmutter über einen Schlitten durch die Linearführung unterstüzt wird.
Mit einer ungestützten Spindellänge von 0.5 m ergibt sich $n_{kr} = 2300\,\frac{1}{\text{min}}$.

Lebensdauer

Die Lebensdauer wird analog zu Abschnitt 3.2.2 nach [8] berechnet.
Unter der Annahme, dass die Last wieder durch $i^* = 2$ Gewindegänge getragen wird, (Hierbei wird aufgrund nicht weiter bekannten Bedingungen im Betrieb auf der sichersten Seite gerechnet) ergeben sich die Tragzahlen einer 16x5 *Micron* Gewindespindel zu $C_{dyn} = 6.7\,\text{kN}$ und $C_{0,stat} = 7.2\,\text{kN}$. [13]
Mit einer maximal zulässigen Vorschubgeschwindigkeit von $0.67\,\frac{\text{m}}{\text{s}}$ wird nach [8] ein Belastungsfaktor von $f_W = 1.4$ verwendet.
Die Lebensdauer in Umdrehungen ergibt sich mit Gleichung (3.32) zu:

$$L = (\frac{6.7\,\text{kN}}{1.4 \cdot F_{axial,max,Y}})^3 \cdot 10^6 > 10^9 \quad \checkmark \tag{3.76}$$

Es wird also Dauerfestigkeit angenommen.

Statischer Sicherheitsfaktor

Zur Verhinderung von plastischer Verformung der Wälzkörper oder deren Lauffläche ergibt sich folgende Sicherheit:

$$f_S = \frac{C_{0,stat}}{F_{axial,max,Y}} \approx 15.2 \quad \checkmark \tag{3.77}$$

3.3.3 Lebensdauer Profilschienenführung

Für die Belastung aufgrund der Gewichtskraft gilt näherungsweise $P_g = m_Y g = 686.7\,\mathrm{N}$. Obwohl die Belastung auf zwei Profilschienenführungen aufgeteilt wird, wird zur Berechnung des unsichersten auftretenden Falles angenommen, dass eine Profilschienenführung das gesamte Gewicht trägt.

Für die hier verwendeten Profilschienenführungen HRC25MN von CPC ergibt sich die dynamische Tragzahl zu $C_{dyn} = 24.8\,\mathrm{kN}$ und die statische Tragzahl zu $C_{0,stat} = 42.5\,\mathrm{kN}$. [9, S. 25]

Und somit die Lebensdauer nach Gleichung (3.44) zu:

$$L \cdot 100\,\mathrm{km} = (\frac{24.8\,\mathrm{kN}}{686.7\,\mathrm{N}})^3 \cdot 100\,\mathrm{km} \approx 4.7 \times 10^6\,\mathrm{km} \quad \checkmark \tag{3.78}$$

Diese Linearführung ist also mehr als ausreichend dimensioniert.

3.3.4 Lebensdauer Wälzlager

Siehe auch Abschnitt 3.2.4.

Bestimmung der Lebensdauer

Loslager Die Belastung des Loslagers entspricht der Gewichtskraft der Masse. Wieder angenommen mit $m = 70\,\mathrm{kg}$ für die rein radiale Belastung des Kugellagers:

$$P = P_0 = F_r = F_g = 686.7\,\mathrm{N} \tag{3.79}$$

Hier wird ein Kugellager 6001-2RSH von SKF verwendet: $C = 5.4\,\mathrm{kN}$ und $C_0 = 2.36\,\mathrm{kN}$ [10, S. 350]

$$L_{10} = (\frac{C}{P})^p = (\frac{5.4\,\mathrm{kN}}{686.7\,\mathrm{N}})^3 \approx 486 \quad \checkmark \tag{3.80}$$

$$s_0 = \frac{2.36\,\mathrm{kN}}{686.7\,\mathrm{N}} \approx 3.44 \quad \checkmark \tag{3.81}$$

Sowohl die Lebensdauer als auch die Sicherheit gegen bleibende Verformung sind mit 486 Millionen Umdrehungen und einer statischen Sicherheit von über 3 ausreichend dimensioniert.

Festlager Für die hier verwendete Serie 32A gilt: $e = 0.8$, $X = 0.63$, $Y_1 = 0.78$, $Y_2 = 1.24$ und $Y_0 = 0.66$ [10, S. 494]
Wieder angenommen mit $m = 70\,\text{kg}$ und einer Spindelsteigung von $p = 5\,\text{mm}$:

$$F_r = F_G = 686.7\,\text{N} \tag{3.82}$$

$$F_a = F_{axial,max,Y} = 472\,\text{N} \tag{3.83}$$

Hier wird ein Kugellager 3201A-2RS1 von SKF verwendet: $C = 10.1\,\text{kN}$ und $C_0 = 5.6\,\text{kN}$ [10, S. 526]

$$\frac{F_a}{F_r} \approx 0.3 < e \tag{3.84}$$

und daher

$$P = F_r + Y_1 F_a = 686.7\,\text{N} + 0.78 \cdot 472\,\text{N} = 1054.86\,\text{N} \tag{3.85}$$

$$P_0 = F_r * Y_0 F_a = 686.7\,\text{N} + 0.66 \cdot 472\,\text{N} = 998.22\,\text{N} \tag{3.86}$$

$$L_{10} = (\frac{C}{P})^p = (\frac{10.1\,\text{kN}}{1054.86\,\text{N}})^3 \approx 877.77 \quad \checkmark \tag{3.87}$$

$$s_0 = \frac{5.6\,\text{kN}}{998.22\,\text{N}} \approx 5.61 \quad \checkmark \tag{3.88}$$

Sowohl die Lebensdauer als auch die Sicherheit gegen bleibende Verformung sind mit 877 Millionen Umdrehungen und einem Wert von über 5 ausreichend dimensioniert.

3.3.5 Kupplung

Siehe Abschnitt 3.2.5.

3.3.6 Dynamik

Die Dynamik, also in dem vorliegenden Fall der Verlauf der Größen Drehzahl, Beschleunigung, Geschwindigkeit und Vorschubsweg an der Y-Achse wird analog zu Abschnitt 3.2.6 mit MATLAB Simulink ermittelt. Die Resultate sind in Abbildung 3.8 dargestellt.

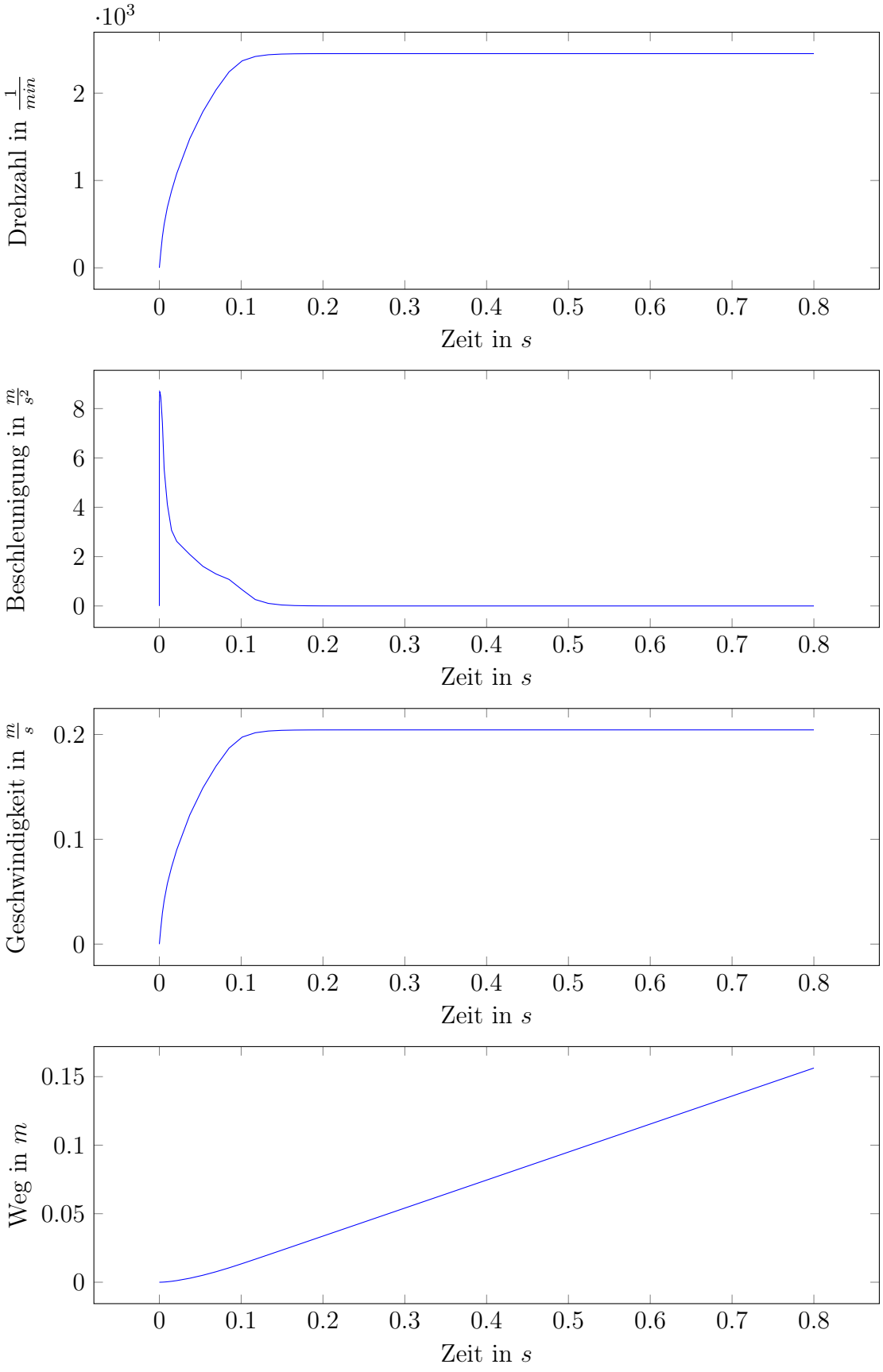


Abbildung 3.8: Verlauf der Bewegungsgrößen an der Y-Achse

Kapitel 4

Steuerelektronik

Zur Steuerung des Portalroboters kommt einerseits die Verwendung fertiger Schrittmotorsteuerungen (Beispielsweise ein PoKeys57CNC[1] in Kombination mit je einem PoStep60-256[2] pro Schrittmotor) und andererseits ein proprietäres Design einer einzigen Platine, die sämtliche Steueraufgaben übernimmt, in Frage.
Die verwendete Elektronik muss folgende Funktionen bieten:

- Steuerung von 5 Schrittmotoren bis zu 4.0 A pro Phase
- Kommunikation mit PC über USB[3]/Ethernet
- Steuerung von 2 Relais für 230VAC[4] für einen *Rollladen* als Späneschutz
- Steuerung von 3 Magnetventilen für Druckluft (Pneumatische Greifeinheit: Auf/Zu sowie Düse zum Entfernen von groben Spänen)
- Input für mindestens 7 Endschalter (Initiatoren)

Möglich ist somit, wie bereits erwähnt, die Verwendung eines PoKeys57CNC. Hierbei müsste jedoch eine zweite Platine (Möglicherweise reicht hier ein Prototyping-Board aus, ist jedoch aus Sicherheitsgründen nicht zu empfehlen) gefertigt werden,

[1] `https://www.poscope.com/product/pokeys57cnc/`
[2] `https://www.poscope.com/product/postep60-256/`
[3] Universal Serial Bus
[4] Volts Alternating Current

die neben den Schrittmotoren auch die Relais und Magnetventile ansteuert und die Signale der Endschalter auf ein mit den GPIO[1]-Pins des PoKeys57CNC verträgliches Spannungssignal umwandelt.
Aufgrund des relativ hohen Aufwands der Adaptierung eines bestehenden PoKeys57CNC, wurde die Entscheidung getroffen, eine einzelne, eigenständige Platine zu entwickeln, die in einer Einheit sämtliche geforderten Funktionen abdeckt.

4.1 Eigenbau Steuerelektronik

In den folgenden Abschnitten wird der Aufbau und die Berechnung sowie die Auswahl der verwendeten Komponenten auf der Steuerplatine näher erläutert.

4.1.1 Aufbau

Mikrocontroller

Das Herz der Steuereinheit bildet ein *NXP MPC5602D* Mikrocontroller im LQFP[2]100 Format aus der Familie NXP MPC56xx. Die Serie MPC560xB von NXP, zu der auch der verwendete *MPC5602D* gehört, beinhaltet *ultra-reliable* 32-bit Mikrocontroller für Automotive und Industrielle Anwendungen wie Motorsteuerungen and andere *General Purpose* Anwendungen. [14]
Dieser Mikrocontroller wurde aufgrund seiner einfachen Programmiermöglichkeit über eine UART[3]-Schnittstelle, seine relativ hohe maximale Taktfrequenz (48 MHz) und vielfältige, dynamisch zuweisbare, Auswahl an Funktionen einzelner Pins, ausgewählt. Dies erleichtert das Platinendesign und die Kommunikation mit der verwendeten Peripherie auf dem Steuerboard.
Das Package (Format) LQFP100 wurde gewählt, da in dieser Version mit 100 externen Pins genügend PWM[4]-Ausgänge und Externe Interrupts vorhanden sind.

[1] General Purpose In-Out
[2] Low Profile Quad Flat Package
[3] Universal Asynchronous Receiver/Transmitter
[4] Pulse Width Modulation

Der Mikrocontroller erlaubt mit geringen Abweichungen grundsätzliche Versorgungsspannungen von 3.3 V oder 5 V. [15, S. 23,25]

Gate-driver für MOSFETs zur Motorsteuerung

Zur direkten Steuerung der Motoren und somit Regelung des Stroms durch die einzelnen Motorspulen wird ein Texas Instruments DRV8711 verwendet. Dieser sogenannte *Gate-driver*-IC[1] lässt sich so konfigurieren, dass im Takt eines PWM-Input-Signals bei jeder steigenden Flanke des Signals der Spulenstrom über extern zugeschaltene MOSFETs durch die Motoren so geregelt wird, dass der Schrittmotor genau einen Schritt (Ein Vollschritt oder je nach Einstellung ein Mikroschritt bis zu $\frac{1}{256}$ eines Vollschrittes) ausführt. Über ein SPI[2] Interface kann über den Mikrocontroller die Konfiguration vorgenommen werden. Darüberhinaus verfügt der TI[3] DRV8711 über automatische Erkennung von mechanischem Blockieren (*Stalling*) und Kurzschluss des Motors. [16]
Derartige Gate-driver sind notwendig, da die MOSFETs in relativ schneller Abfolge an- und ausgeschaltet werden müssen, um den Spulenstrom konstant zu halten. Dies soll erreicht werden um so geringe Schwankungen im Drehmoment des Motors wie möglich zu erhalten. Das Drehmoment eines Schrittmotors ist proportional zum Spulenstrom. Dieser IC hat sogenannte *Charge-Pumps* integriert, um eine Spannung von 10 V über Versorgungsspannung zu generieren, um N-Kanal MOSFETs verlässlich steuern zu können.

Auswahl der MOSFETs

Da in diesem Fall pro Motor 8 MOSFETs benötigt werden, um eine sogenannte *Full-H-Bridge* (Siehe auch Abbildung 4.11; es werden 4 Transistoren pro Phase benötigt) zu bilden, wird Wert darauf gelegt, MOSFETs in einem so kleinen For-

[1] Integrated Circuit
[2] Serial Peripheral Interface Bus
[3] Texas Instruments

mat wie nur möglich auszuwählen, um in weiterer Folge die Größe der Platine so klein wie möglich zu halten.
Die ausgewählten MOSFETs sind Texas Instruments CSD88537ND. [17] Diese bieten jeweils zwei N-Kanal MOSFETs in einem 5 mm$x$6 mm SO-8-Package, sind mit dem ausgewählten Gate-driver IC im Hinblick auf die Eingangs-Kapazität und Gate-Ladung kompatibel und erlauben einen Stromfluss von über 4 A.

USB Kommunikation

Da der ausgewählte Mikrocontroller keinen direkten USB Tranceiver/Controller inkludiert, wird auf den USB-UART Schnittstellenchip FT230X von FTDI zurückgegriffen. Dieser wandelt USB-Signale in ein Basic-UART Signal (RX/TX) um und scheint, die richtigen Treiber vorausgesetzt, angeschlossen an einen handelsüblichen PC als Serial Port (Analog zu RS232) auf. [18] Somit kann auch ohne direkte USB-Programmierung über jede moderne Programmiersprache über eine emulierte serielle Schnittstelle mit dem Steuerboard über ein vordefiniertes Protokoll (Siehe Abschnitt 4.4.1) kommuniziert werden.
Da die Motoren im Regelfall mit 24 V-48 V betrieben werden, kann dies bei Ausfall eines oder mehrer Komponenten auf der Platine zu einer Überspannung am USB-Port (Standard USB Versorgungsspannung: 5 V) und somit Vernichtung dieses oder des gesamten Mainboards des Host-PCs führen. Um dies in jedem Fall zu vermeiden, wird mittels eines *Digital Isolators* von Texas Instruments eine galvanische Isolation für die Anbindung der USB Komponenten zur restlichen Steuerplatine geschaffen, die aber dennoch Übertragungsraten von bis zu 25 $\frac{\text{Mbit}}{\text{s}}$ zulässt. Eine Alternative wäre die Verwendung von Optokopplern. Diese bieten aber meist keine All-In-One Lösung und sind darüberhinaus nicht auf Übertragungsraten von mehreren 100 $\frac{\text{kbit}}{\text{s}}$ ausgelegt. Konkret wird hier der TI ISO7321C verwendet, der galvanische Isolation bis zu 3000 V bietet. [19]

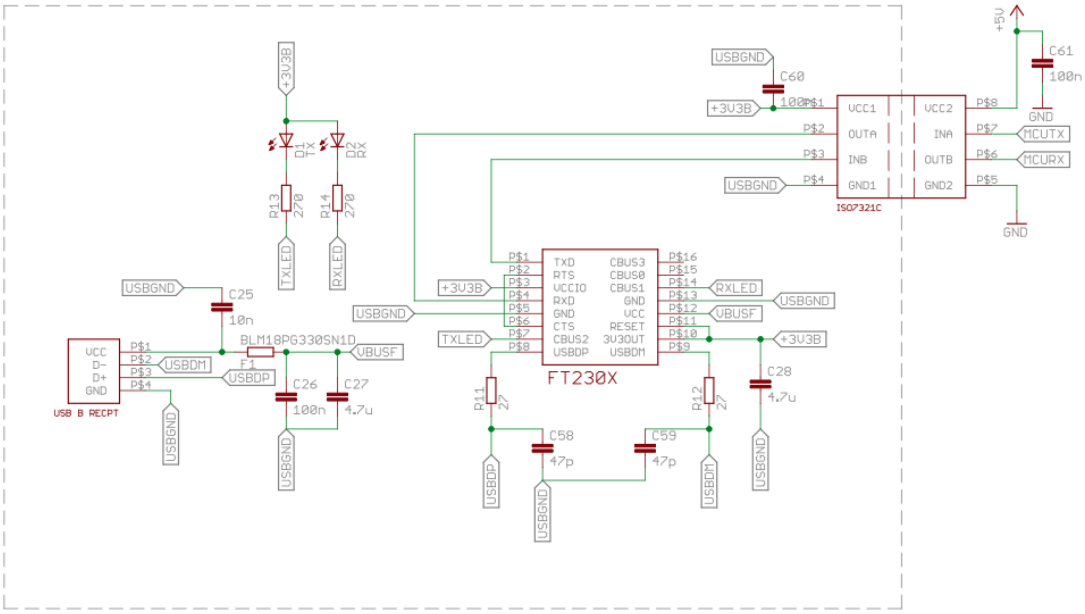


Abbildung 4.1: Schaltplan zur USB-Kommunikation mit einem Host-PC. Grau strichlierte Linien stellen voneinander galvanisch getrennte Bereiche der Steuerung dar

Spannungsversorgung des Mikrocontrollers

Schrittmotoren mit einer Leistung wie die hier verwendeten werden im Regelfall mit einer Spannung $U \geq 24\,\mathrm{V}$ betrieben. Diese Spannung muss zur Verwendung als Energiequelle für den betrachteten Mikrocontroller auf konstant maximal 5 V reduziert werden. Dies geschieht mit einem Schalt-Spannungsregler. Hier wurde aufgrund der Einfachheit der externen Beschaltung, mit so wenigen externen Bauteilen wie nur möglich, auf einen Texas Instruments LMZ14201H zurückgegriffen. Dieser bietet einen Maximal-Ausgangsstrom von 1 A bei einer Ausgangsspannung von 5 V. [20]

Aufgrund des relativ hohen Spannungsunterschiedes von Ausgangs- und Eingangsspannung des Spannungsreglers wird hier, wie bereits erwähnt, auf einen Schaltspannungsregler und nicht auf einen LDO[1] Spannungsregler zurückgegriffen. Bei einer Eingangsssspannung von 24 V und einer Ausgangsspannung von 5 V bietet der

[1] Low-dropout

TI LMZ14201H eine Effizienz von ungefähr 80 %. [20] Im Gegensatz hierzu weist ein LDO Spannungsregler bloß eine Effizienz von $\frac{U_{aus}}{U_{ein}} = \frac{5\,\mathrm{V}}{24\,\mathrm{V}} \approx 20.833\,\%$ auf. Eine dermaßen niedrige Effizienz kann bei der geringen Oberfläche der Bauteile ohne weitere Vorkehrungen oder aktiver Kühlung zu Überhitzen führen. Vorteile von LDOs inkludieren geringere elektromagnetische Emissionen sowie einfachere Beschaltung. (Einige Modelle kommen, je nach Leistung, sogar ohne jegliche äußerlich zugeschaltenen Komponenten aus)

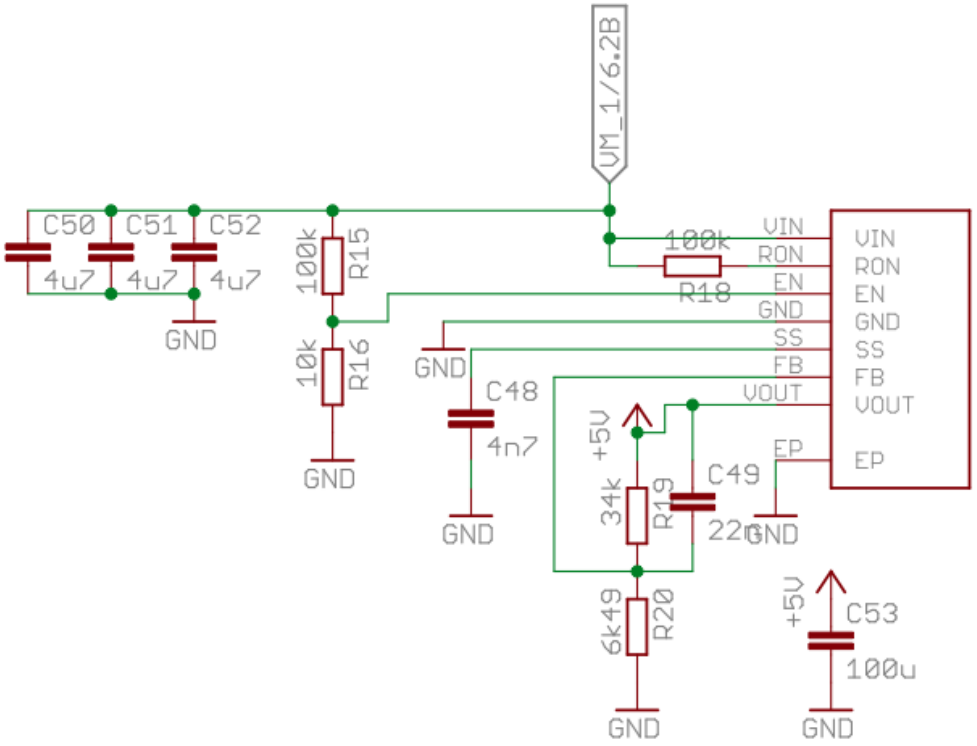


Abbildung 4.2: Beschaltung des verwendeten Schaltspannungsreglers

Steuerung des Rollladen-Rohrmotors

Handelsübliche Rollladen-Rohrmotoren werden, wie in Abbildung 4.3 dargestellt, angeschlossen.

Das bedeutet, um einen Rollladenmotor in zwei Richtungen drehen zu lassen, werden zwei separate Schalter (Relais) oder ein SPDT[1] Schalter benötigt. Da hier

[1] Single Pole Double Throw

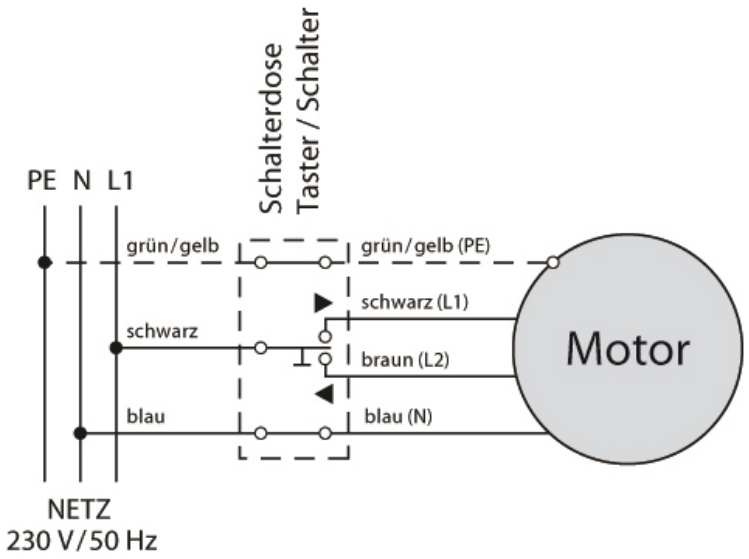


Abbildung 4.3: Anschluss eines 4-adrigen Rohrmotors [21]

Netzspannung mit 230Vrms[1] verwendet wird, müssen beispielsweise zwei antiparallele Thyristoren, ein TRIAC[2], zwei serielle MOSFETs oder ein simples elektromechanisches Relais verwendet werden, um sowohl die positive als auch negative Halbwelle der Wechselspannung schalten und sperren zu können. (Ein einzelner MOSFET würde hier aufgrund der sogenannten *Body-diode* nicht genügen und in eine Richtung unabhängig von der momentanen Ansteuerung leiten)
Hierfür wurden zwei separate SSR[3]s ausgewählt. Gegenüber regulären Relais haben diese den Vorteil, keine beweglichen Teile für den Schaltvorgang aufzuweisen. Weiters bietet bei den verwendeten Relais die standardmäßig bereits integrierte Optokopplung eine sichere galvanische Isolation der restlichen Teile der Steuerlektronik zu der Netzspannung.
Hier werden aufgrund des kleineren *footprints* sowie der größeren Oberfläche und somit besserer Wärmeabfuhr durch höhere Bauweise *Through-Hole Komponenten* anstatt SMD-Komponenten verwendet.

[1]Volts Root Mean Square
[2]Triode for Alternating Current
[3]Solid State Relais

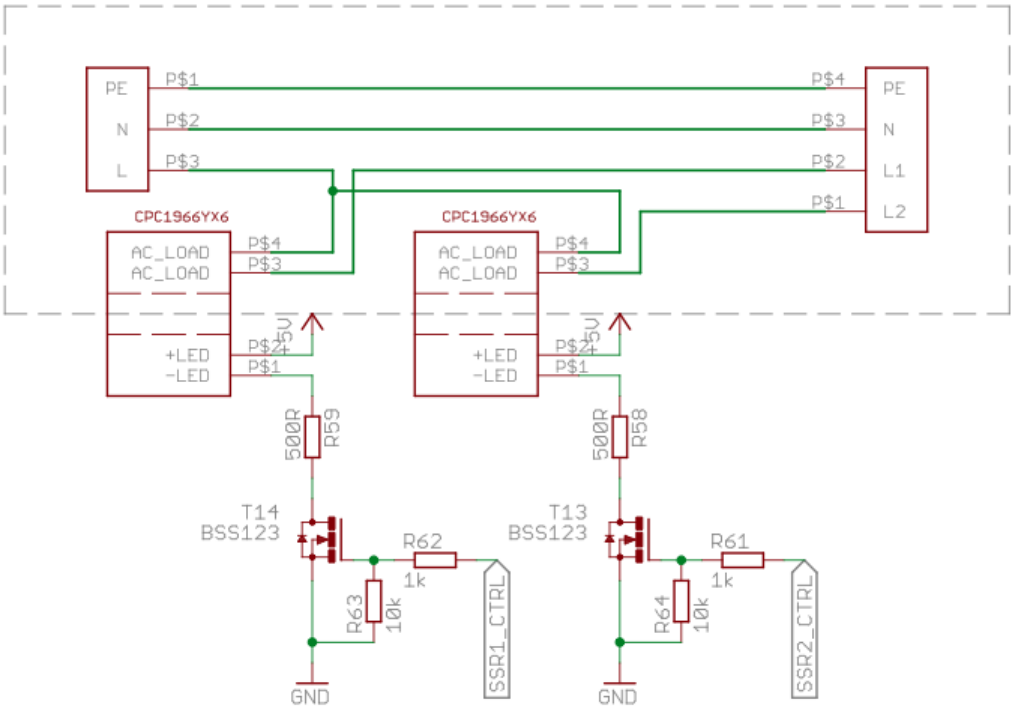


Abbildung 4.4: Ansteuerung und Verschaltung der Relais

Das Datenblatt des Solid State Relais CPC1966YX6 gibt den für den Schaltvorgang maximal benötigten Strom durch die für den Optokoppler integrierte Diode mit 5 mA sowie den maximalen Peak Control Current mit 50 mA an. [22] Um hier auf der sicheren Seite zu arbeiten, wird ein Steuerstrom $I > 5\,\text{mA}$ gewählt. Dies wird durch einen Serienwiderstand und einen N-Kanal MOSFET an der Kathode der in das SSR integrierten Diode realisiert. Die Anode wird direkt mit der 5 V Spannungsversorgung des Mikrocontrollers verbunden.

Eine direkte Ansteuerung der Diode über einen GPIO Pin des Mikrocontrollers wird hier aufgrund der Tatsache, dass ein Steuerstrom von mindestens 5 mA benötigt wird und die GPIO Pins des verwendeten Mikrocontrollers mit ihrer maximalen Belastbarkeit von wenigen mA nicht überlastet werden sollten, vermieden. [15] Deshalb muss hier der Umweg über eine Schaltung mit einem N-Kanal MOSFET gegangen werden. (Siehe Abbildung 4.4)

Liegt keine Spannung am GPIO Pin `SSR1_CTRL` (dies ist ein GPIO Pin des Mikrocontrollers, der per Software als Ausgang deklariert wurde) an, so ist der N-Kanal MOSFET hochohmig. Dies wird darüberhinaus auch bei nicht angeschlossenem `SSR1_CTRL` durch einen *pull-down Widerstand* gewährleistet. Dieser hält die anliegende Spannung am Gate des MOSFETs auch bei fehlender Verbindung oder in Umgebungen mit starken elektromagnetischen Störsignalen *low*. Zwischen dem GPIO Pin und dem Gate des MOSFETS wird weiters ein 1 kΩ Widerstand gesetzt, um den GPIO Pin durch das kapazitive Verhalten des Gates gegenüber Source (*Input Capacitance* [23]) nicht bei Aktivierung kurzfristig zu überlasten. Wird nun eine Spannung an `SSR1_CTRL` gegenüber GND angelegt, also der entsprechende Ausgang des Mikrocontrollers *high* geschaltet, wird der MOSFET niederohmig und es fließt, bei Vernachlässigung des $R_{DS(ON)}$ (also des An-Widerstands) des MOSFETs besagter Strom durch die Diode und das Solid-State-Relais wird über den Optokoppler aktiviert.
Die Berechnung des durch die Diode fließenden Stroms wird im Folgenden erläutert.
Aus der Spannungsbilanz ergibt sich:

$$5\,\mathrm{V} = U_{Diode,forward} + I(500\,\Omega + R_{DS(ON)}) \tag{4.1}$$

und somit mit $U_{Diode,forward}$ und $R_{DS,on}$ laut Datenblättern:

$$I = \frac{5\,\mathrm{V} - 1.2\,\mathrm{V}}{500\,\Omega + 10\,\Omega} \tag{4.2}$$

$$I \approx 7.45\,\mathrm{mA} \tag{4.3}$$

Der Strom durch den MOSFET und somit gleichzeitig durch die Diode (Steuerstrom) im ausgeschalteten Zustand liegt laut Datenblatt bei maximal 100 nA, ist also im vernachlässigbaren Zustand. [23] Deshalb kann hier der Strom einfacherweise als null angenommen werden.

Ansteuerung der Magnetventile

Die Ansteuerung der Magnetventile (Abbildung 4.5) geschieht analog zu den bisher vorgestellten Schaltungen mit N-Kanal MOSFETs. Das positive Ende der Span-

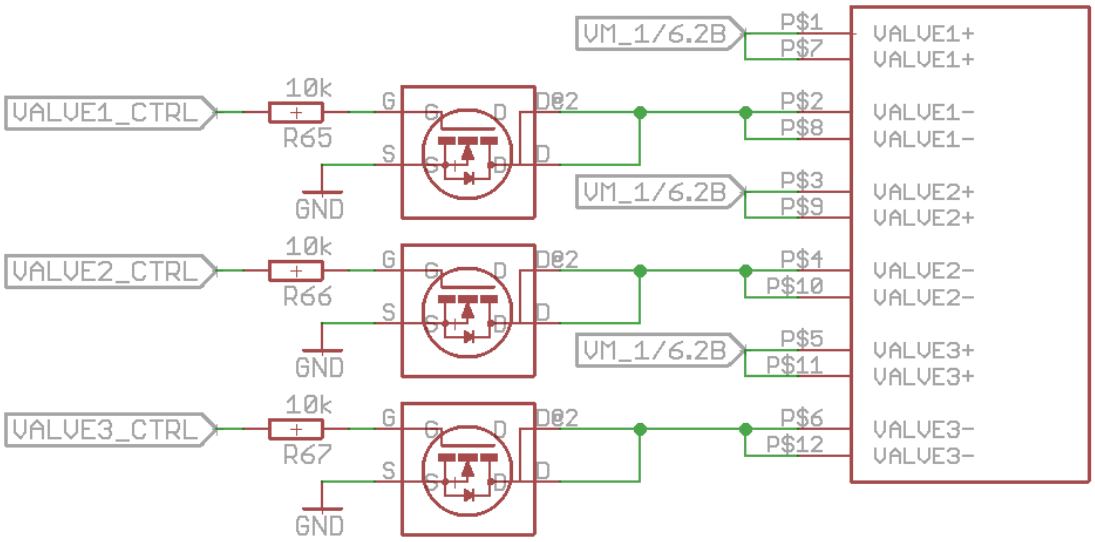


Abbildung 4.5: Ansteuerung der Magnetventile

nungsversorgung wird mit der Spannungsversorgung eines Motors (`VM_1`) verbunden. Das negative Ende der Spannungsversorgung des Magnetventils (Das Steckterminal) wird über einen MOSFET (NDT3055) mit `GND`, also dem negativen Ausgang der Spannungsversorgung des Motors, verbunden. Auch hier wird über einen GPIO Ausgang des Mikrocontrollers via eines *Gate Widerstands* das Gate des MOSFETs gesteuert.

Platinenmaterial

Als Platinenmaterial wird Standard FR-4 Glasfaser-Epoxid benutzt. Aufgrund der Komplexität der Leiterbahnen wird ein *4 Layer Stackup* verwendet, mit 35 µm Kupfer auf den inneren Layern und 70 µm Kupfer auf den äußeren, die größere Ströme führen.

Platinendesign

Das Platinendesign wurde mit der Educational Version von Eagle PCB[1] ausgeführt. Bis auf die bisher bereits angesprochenen Solid-State-Relais und die Schraub/Klemmterminals werden nur SMD[2]s verwendet.

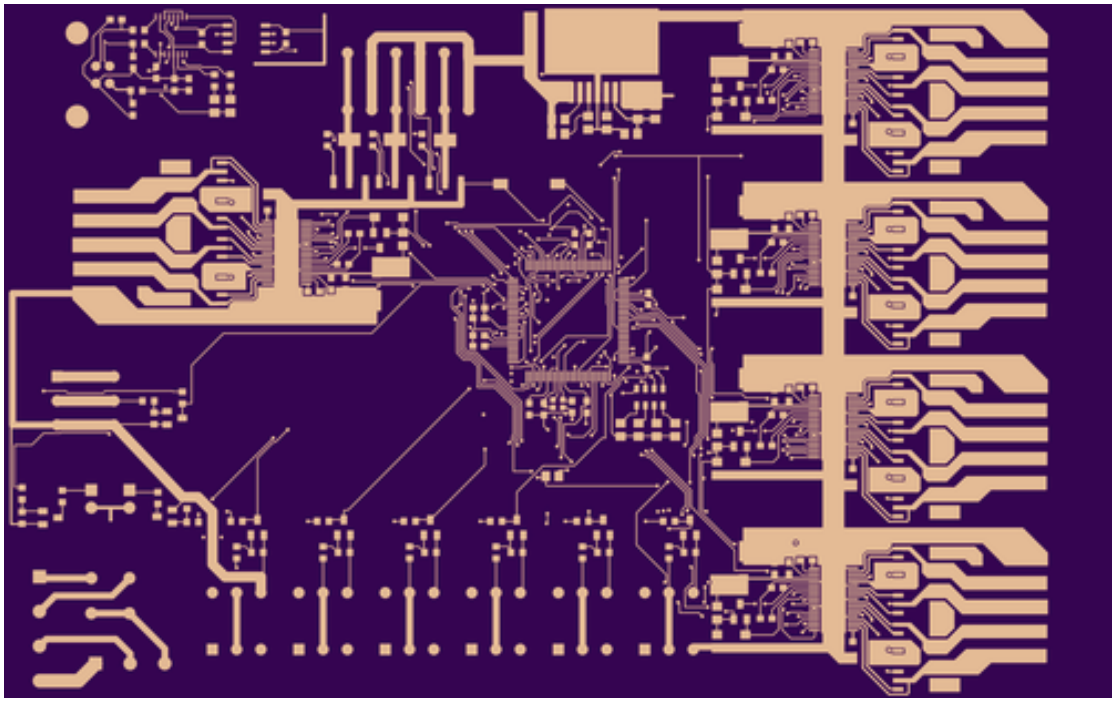


Abbildung 4.6: Äußerster Layer der fertigen Leiterplatte, ohne Lötstopplack

Die hier dargestellte Abbildung 4.6 zeigt den oberen Layer der Platine, hier ohne Lötstopplack. Die effektive Kupferschicht/Kupferpads (Leiter, in Gold) dieses Layers ist auf dem Glasfaser-Platinenmaterial (Violett) sichtbar.
Abbildung 4.7 zeigt dieselbe Platine, zur einfacheren Bestückung, dort mit Lötstopplack, also zinnabweisender Beschichtung, versehen, wo keine Bauelemente angelötet werden müssen. Dies ist die gewohnte Darstellung einer Leiterplatte.

[1]Printed Circuit Board
[2]Surface Mount Device

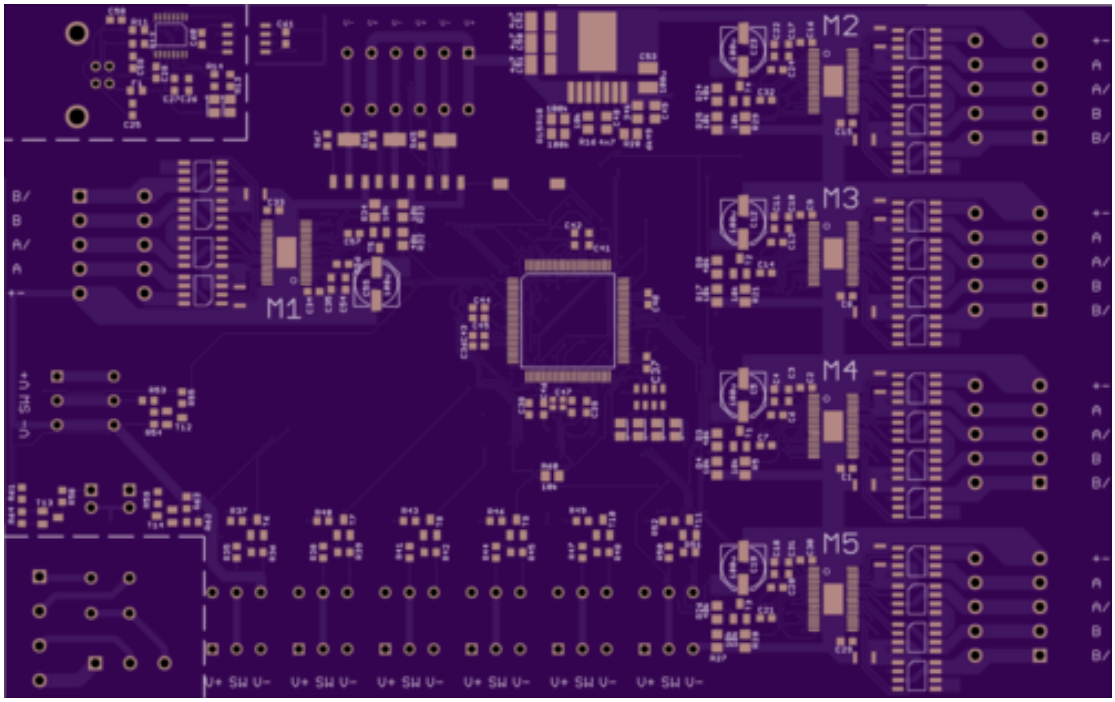


Abbildung 4.7: Fertige Steuerplatine, ohne Lötstopplack

4.1.2 Anbindung der Peripherie

Motoren

Für die Anschlüsse der Motoren werden zweireihige Block-Steckterminals benutzt. Die beiden vertikal übereinanderliegenden Steckverbinder werden durch Kupferleiter auf der Platine verbunden. Jeweils zwei Steckverbinder sind somit Anschlüsse für ein Ende einer Motorphase. So können in Übereinstimmung mit [4] verschiedene Schrittmotor Ansteuerungsverfahren (Bipolar parallel, seriell) durch bloße Änderung der Reihenfolge der Anschlüsse des Motors verwendet werden, ohne die Steuerelektronik modifizieren zu müssen.

Sensoren/Endschalter und Magnetventile

Für die Anbindung der Sensoren (Endschalter) und Magnetventile wurden Schraubterminals mit 3.5 mm Rastermaß verwendet. Bei den Magnetventilen ist jeweils ein Terminal für das positive und ein Terminal für das negative Ende der Versorgungsspannung von direktgesteuerten 3-Wege Magnetventilen vorgesehen.

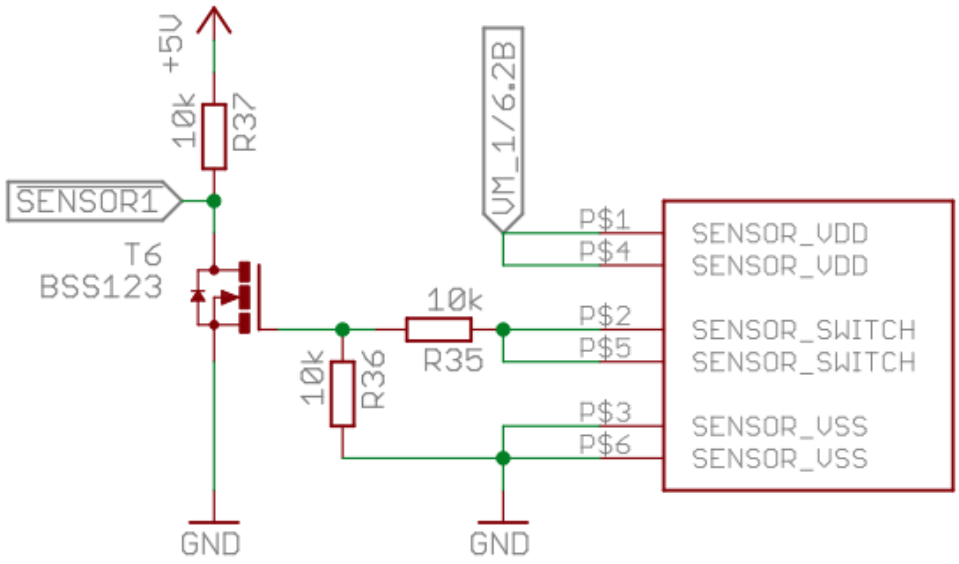


Abbildung 4.8: Schaltung, um 24 V-Signale für einen 5 V-Eingang verträglich zu machen

Die Terminals für die Endschalter wurden so ausgelegt, dass jeweils ein Verbinder für die positive Spannungsversorgung, ein Verbinder für die negative Spannungsversorgung und ein Verbinder für die geschaltene Phase vorhanden ist.
Auf diese Art können sowohl simple 2-Kabel Sensoren wie Hall- und Reedsensoren und Mechanische Mikroschalter als auch komplexere induktive und kapazitive 3-Kabel PNP/NPN Schalter, die eine separate Spannungsversorgung benötigen, verwendet werden. Bei Sensoren, die 3 Verbindungen ($V+$, SW, $V-$) aufweisen, müssen alle drei Kabel verbunden werden. Andererseits, wie beispielsweise bei dem beim Rollladen angedachten Endschalter (Panasonic Mikroschalter, mechanisch) vorgesehen, müssen bei Endschaltern ohne dedizierte Spannungsversorgung bloß $V+$ und SW verbunden werden. Bei Aktivierung wird die Verbindung zwischen $V+$ und SW niedrigohmig und es liegt die Spannung $V+$ an SW an.
Um die Schaltsignale für den verwendeten Mikrocontroller verträglich zu machen (*Recommended Operating Condition 5V* nach [15]: $V_{DD} \pm 0.5V$, also in dem vorliegenden Fall mit Versorgungsspannung $V_{DD} = 5\,\mathrm{V}$ ist hier das Maximum des empfohlenen Betriebsbereichs 5.5 V), wird eine Schaltung mit N-Kanal MOSFETs

verwendet. Diese invertiert in der vorliegenden Form nach Abbildung 4.8 das anliegende Signal zwar, da dieses aber sowieso per Software eingelesen wird, ist es trivial dieses erneut zu invertieren.
Nach [23] ist das *Absolute Maximum Rating* für die Gate-Source-Spannung am N-Kanal-MOSFET $V_{GS} = \pm 20\,\mathrm{V}$. Um nicht direkt die Versorgungsspannung `VM_1` an das Gate des MOSFETs anzulegen, wird vor das Gate ein Spannungsteiler geschalten. Dieser halbiert die vom Sensor kommende Spannung. Die laut [23] maximale *Gate threshold voltage* ist 2.8 V. Die minimale Spannung an `SENSOR_SWITCH`, die ein Aktivieren des MOSFETs und somit eine Signalweiterleitung bewirkt, ist demnach aufgrund des 1:1 Spannungsteilers $V_{SW} = 2 \cdot 2.8\,\mathrm{V} = 5.6\,\mathrm{V}$.
Die Magnetventile und Sensoren werden über die Spannungsversorgung des ersten Motors (M1) mit Spannung versorgt. Aufgrund der verschwindend geringen Leistung der Sensoren/Endschalter und relativ geringen Leistung der Magnetventile (12 W, nicht dauerhaft aktiv) stellt dies kein praktisches Problem dar.

Rollladen-Rohrmotor

Ebenfalls über Schraubklemmen wird die Rollladen-Rohrmotor-Anbindung ausgeführt. Hierbei wird aber auf Schraubklemmen mit einem Rastermaß von 5mm zurückgegriffen, um eine bessere Isolation aufgrund der höheren anliegenden Netzspannung zu erhalten.
Trotz des relativ einfachen Bedienens des Rohrmotors muss in jedem Fall darauf geachtet werden, Schutzleiter, Neutralleiter und Phase richtig anzuschließen. Aus diesem Grund war es leider nicht möglich, einen `IEC` (Kaltgeräte-)Stecker zu verwenden, da diese mit einem Schutzkontaktstecker als anderes Ende nicht verpolungssicher ausgeführt sind. Ein Vertauschen von Phase und Nullleiter ist durch das Datenblatt der meisten Rollladen-Rohrmotoren nicht abgedeckt und könnte somit zur Zerstörung ebendieses und zu Sicherheitsbedenken führen. Die Steuerelektronik des Portalroboter würde dadurch jedoch nicht beschädigt.

Programmierung des Mikrocontrollers

Ein mit NXP Codewarrior in C geschriebenes Programm kann nach dem Kompilieren mittels des frei verfügbaren RappID Bootloaders von NXP über eine serielle Schnittstelle (USB, hier emuliert durch eine USB-UART-Bridge, siehe auch Abbildung 4.1) über UART auf den Mikrocontroller überspielt werden.

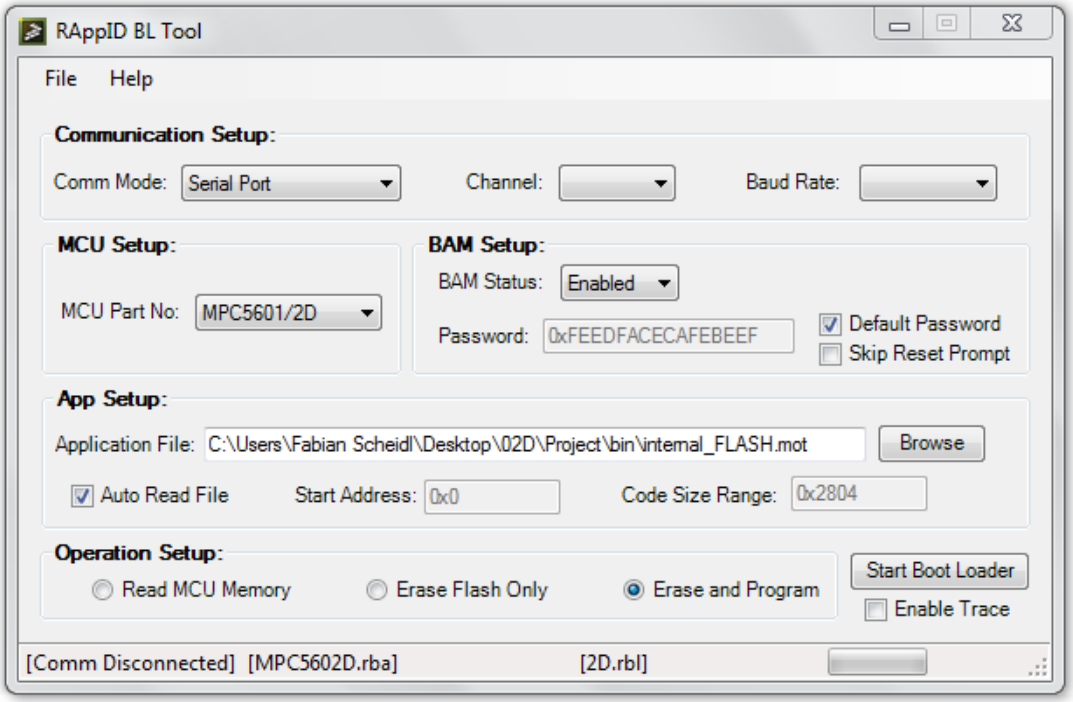


Abbildung 4.9: Graphisches User Interface des NXP RAppID BL Tools

Die Baud Rate ergibt sich nach dem Reference Manual des MPC5602D in Abhängigkeit der Beschaltung der Crystal Resonators-Pins `XTAL` und `EXTAL`. Im Fall der Steuerelektronik wird hier ein 16 MHz Quartz benutzt. Dies ergibt eine Baud Rate zur Übertragung des Bootloaders von $b = 19600$. Eine nähere Erklärung des Boot Assist Modules zur Programmierung des MPC5602D ist [24, 59ff] zu entnehmen.

4.2 Grundlagen der Steuerung von Schrittmotoren

Bei Schrittmotoren wird zwischen unipolar betriebenen und bipolar betriebenen Schrittmotoren unterschieden. Der grundsätzliche Aufbau beider Typen unter-

Polarität	Spule A	+	+		-	-	-		+
	Spule B		+	+	+		-	-	-
Kommutationszyklus	Full-stepping	1		2		3		4	
	Half-stepping	1	2	3	4	5	6	7	8

Tabelle 4.1: Abfolge der Bestromung der Spulen für Full-Stepping und Half-Stepping, adaptiert nach [25]

scheidet sich nur dadurch, dass beim unipolaren Schrittmotor eine Mittelleitung an den Spulen vorhanden ist. (Siehe Abbildung 4.10)

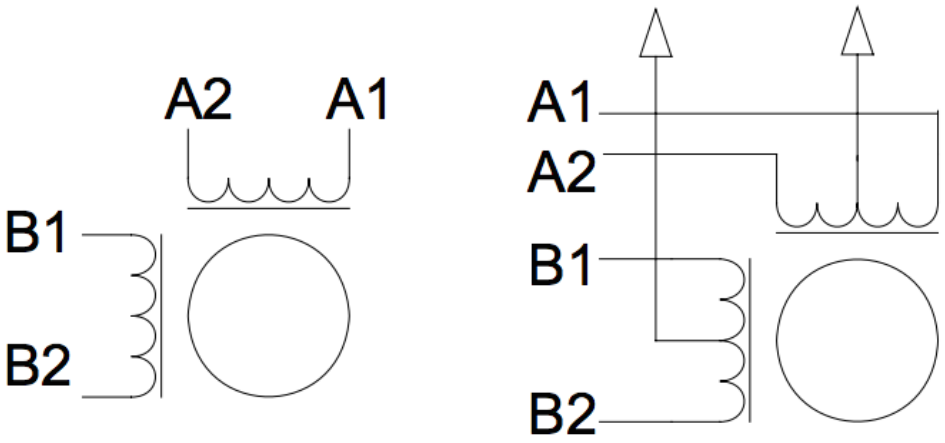


Abbildung 4.10: Schematischer Aufbau eines unipolaren und bipolaren Schrittmotors [25]

Die Folge, in der die Spulen bestromt werden müssen, um den Motor in Bewegung zu setzen wird im Datenblatt des Schrittmotors [4] angeführt. Dies entspricht dem Industriestandard für Schrittmotoren.
Wenn diese Abfolge eingehalten wird, spricht man vom sogenannten *Full Stepping*. Eine Möglichkeit, kleinere Schritte und somit ein kontinuierlicheres Drehmoment zu erzeugen, ist, mehrere Spulen gleichzeitig unter Strom zu setzen. Somit wird der Rotor zwischen zwei *Full-Steps* gehalten. Moderne Schrittmotorsteuerungen können durch genaue Stromregelung sogenannte *Microsteps* bis zu $\frac{1}{256}$ eines Fullsteps ausführen. [16]

Schrittmotoren müssen, analog zu BLDC[1] Motoren (bürstenlosen Gleichstrommotoren) elektronisch kommutiert werden und besitzen keine Kommutatorbürsten. Dies führt zwar zu erhöhtem Implementierungsaufwand in Hard- und Software, bietet aber durch Möglichkeit von Microstepping größere Flexibilität in der Positions- und Bewegungssteuerung, weniger Verluste (durch Kontakt- und reibungsfreien Betrieb bis auf die Lager des Rotors) und einen bis auf die Lagerung verschleißfreien Betrieb.
Um die elektronische Kommutierung der Spulen zu vereinfachen und genaues Timing zu gewährleisten, wird ein dedizierter Stepper Motor-Driver (DRV8711) verwendet. Dieser bietet Ausgänge, um 8 MOSFETs zu steuern. (Siehe auch [16])
Jeweils 4 dieser 8 MOSFETs werden in einer sogenannten *Full H-Bridge* (Siehe Abbildung 4.11) verschalten. Dies ermöglicht, Strom durch eine Spulen in beliebige Richtung fließen zu lassen.

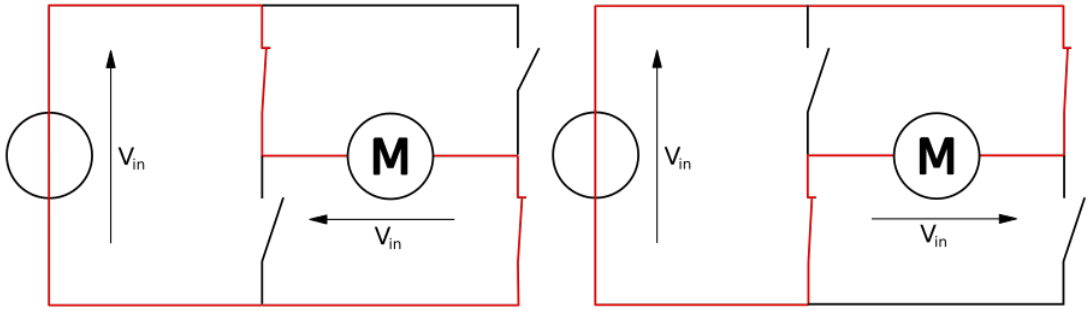


Abbildung 4.11: Die zwei grundsätzlichen aktivierten Zustände einer H-Bridge [26] nach CC BY-SA 3.0

Eine (reale) Spule des Motors kann vereinfachterweise als Reihenschaltung einer idealen Induktivität und eines Ohmschen Widerstands gesehen werden, wobei die jeweiligen Werte dem Datenblatt des Motors entnommen werden.
Wird nun Spannung an die Spule angelegt, steigt analog zur Differentialgleichung einer widerstandsbehafteten Spule der Strom durch die Spule.

$$V_{in} = V_L + V_R \tag{4.4}$$

[1] Brushless Direct Current

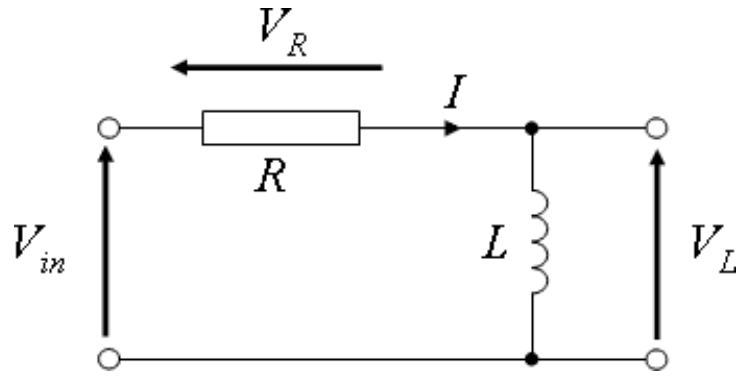


Abbildung 4.12: Ersatzmodell einer widerstandsbehafteten Motorspule [27] nach CC BY-SA 3.0

$$V_{in} = L\frac{di(t)}{dt} + R \cdot i(t) \tag{4.5}$$

Eine Simulation mit MATLAB Simulink (Siehe Abbildung 4.13) mit für Motorsteuerungen übertriebenen, jedoch zur qualitativen Veranschaulichung verwendeten Induktivitätswerten ($L = 2\,\mathrm{H}$, $R = 1\,\Omega$, $V_{in,max} = 10\,\mathrm{V}$) ergibt mit Hystereseschwellen $I_o = 2.5\,\mathrm{A}$ und $I_u = 2\,\mathrm{A}$ bei Verwendung eines *Zweipunktreglers* zur Stromregelung einen zeitlichen Stromverlauf durch Spule und Widerstand nach Abbildung 4.14.

Wie weiters in Abbildung 4.14 zu sehen ist, erhöht bei Zweipunktregelung eine Erhöhung der Versorgungsspannung den quasistationären Spulenstrom (Drehmoment) nicht, verringert jedoch die Zeit, in der dieser erreicht wird. Das bedeutet, dass mit höherer Eingangsspannung das Haltemoment des Motors gleich bleibt, das drehzahlabhängige Drehmoment bei höheren Winkelgeschwindigkeiten aufgrund der schnelleren Kommutierung und des schnelleren Erreichens des maximal möglichen Drehmoments gegenüber geringerer Eingangsspannung steigt.

Um ein konstantes Drehmoment und somit einen konstanten Strom durch die Spule zu erreichen, erfolgt eine Messung des Stroms durch die Spule durch einen Strommesswiderstand im Bereich von wenigen Milliohm in Serie zur Spule. Basierend auf dem momentanen Strom durch die Spule wird mittels PWM entweder die Versorgungsspannung oder 0 V an die Spule angelegt. (Dies entspricht einem sogenannten *Slow Decay Mode*; wird die negative Versorgungsspannung angelegt

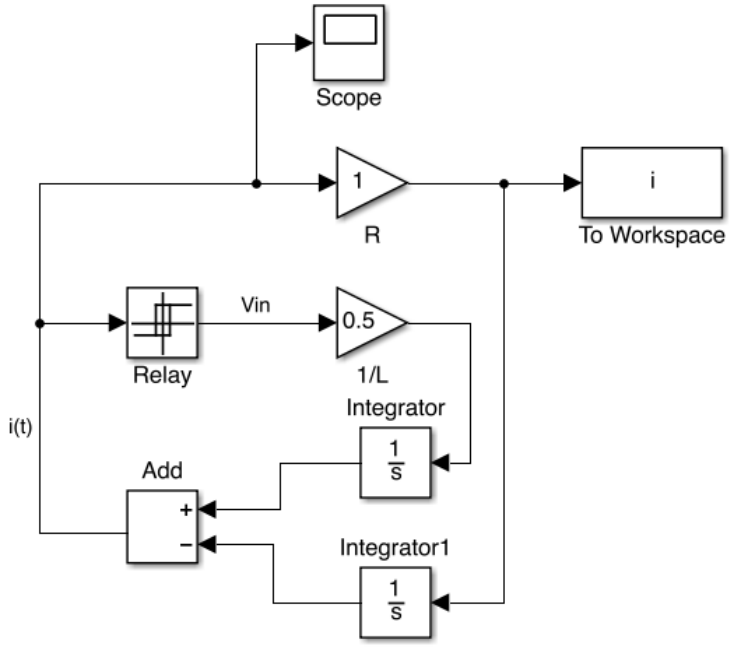


Abbildung 4.13: MATLAB Simulink Modell einer widerstandsbehafteten Motorspule mit Zweipunktregler

spricht man von *Fast Decay*; siehe auch [16, 18f]) Eine mögliche Methode der Regelung ist, beispielsweise eine Hysterese um den Sollwert zu legen. Ist der momentane Strom größer als der Sollwert, wird die Spannung entfernt, ist der momentane Strom geringer als der Sollwert, wird die Versorgungsspannung angelegt. Dies ergibt eine Dreiecksspannung um den Sollwert mit relativ kleiner Amplitude. Je geringer die Schwellwerte der Hysterese auseinanderliegen, desto genauer wird die Regelung, jedoch erhöht sich die Schaltfrequenz. Da bei jedem Schaltvorgang mit MOSFETs durch Durchschreiten eines mittleren Widerstandsbereichs beim Übergang von mehreren Megaohm zu wenigen Milliohm mehr Leistung als im hoch- oder niedrigohmigen Betrieb dissipiert wird, muss hier auf die Schaltdauer, -frequenz und die maximale Temperatur der verwendeten Komponenten Rücksicht genommen werden.

Die Ansteuerung und Regelung des Spulenstroms wird in diesem Fall durch den TI DRV8711 übernommen. Dieser benötigt neben einer Programmierung über eine SPI Schnittstelle nur ein pulsartiges Signal am Eingang, wobei jede steigende

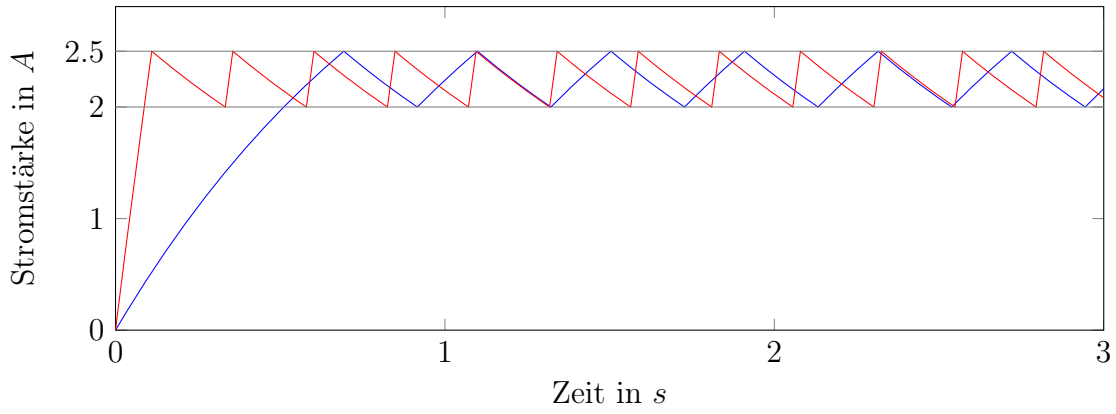


Abbildung 4.14: Verlauf des Spulenstroms mit Anfangsbedingung $i(0) = 0\,\mathrm{A}$ bei einer Zweipunktregelung zwischen $I_u = 2\,\mathrm{A}$ und $I_o = 2.5\,\mathrm{A}$ mit $L = 2\,\mathrm{H}$, $R = 1\,\Omega$ bei $V_{in,max} = 10\,\mathrm{V}$ (blau) und $V_{in,max} = 48\,\mathrm{V}$ (rot)

Flanke des Signals als Trigger, einen Schritt fortzuschreiten, interpretiert wird. [16] Dieses pulsartige PWM-Signal muss nun im Folgenden in der entsprechenden Frequenz durch den Mikrocontroller generiert werden und zur Positionssteuerung verwendet werden.

4.3 Positionssteuerung mit Schrittmotoren

Schrittmotoren bieten gegenüber anderen Motoren (z.B. PMSM[1] oder BLDC) den erheblichen Vorteil, dass diese *Open Loop* betrieben werden können. Durch die Tatsache, dass jeweils ein einzelner Schritt gemacht werden kann und dieser eine (nicht akkumulative) Toleranz von ungefähr 5 % aufweist, kann mit geringer Ungenauigkeit der Drehwinkel des Motors und in weiterer Folge die Position der Last bestimmt werden.[2] [4]

Im Gegensatz hierzu können BLDC und PMSM Motoren im Allgemeinen nur kontinuierlich als Servos betrieben werden. Hierbei wird weiters ein Sensor zur Feedbackgebung wie ein absoluter Axialencoders oder ein relativer Drehencoder

[1] Permanent Magnet Synchronous Motor

[2] Bei Verwendung einer Kugelgewindespindel mit Steigung $p = 5\,\mathrm{mm}$ bedeutet ein Vollschritt mit $1.8° \pm 5\,\%$ einen linearen Vorschub von $s = \frac{5\,\mathrm{mm}\cdot(1.8°\pm5\,\%)}{360\,\mathrm{degree}} = 0.025\,\mathrm{mm} \pm 0.001\,25\,\mathrm{mm}$

(Optisch) verwendet, um die Position des Motors zu bestimmen und in weiterer Folge beispielsweise mittels eines PID-Reglers zu regeln.
Absolute Axialencoder (Absolutwertgeber) bieten hier den Vorteil, dass bereits bei Systemstart die Position der Last bekannt ist und nicht erst eine sogenannte Referenzfahrt ausgeführt werden muss.
Bei einer Referenzfahrt werden die Motoren in eine vorgegebene Richtung bewegt. Vor dem Anschlag aller Achsen befindet sich ein Sensor (bspw. kapazitiv, induktiv, mechanisch oder ein Reed-Switch), der der Steuerung anzeigt, dass ein bekannter Referenzpunkt (Der *Nullpunkt*) erreicht wurde.
Eine Möglichkeit der relativen Positionsbestimmung für BLDC und PMSM Motoren, die jedoch auch bei handelsüblichen Schrittmotoren verwendet werden kann, ist die Messung der sogenannten *Rück-EMF*[1], also der in den nicht bestromten Spulen induzierten Induktionsspannung. Durch den Verlauf dieser kann auf die momentane Ausrichtung des Rotors und die Geschwindigkeit geschlossen werden. Eine Referenzfahrt ist aber aufgrund der relativen Natur dieser Methode auch hier notwendig. Weiters ist die Positionsbestimmung über Messung der *Rück-EMF* aufgrund des Induktionsgesetzes nur bei höheren Drehzahlen aussagekräftig.
Weiters muss bei Positionssteuerung darauf geachtet werden, den Motor nicht zu überlasten (beispielsweise durch zu hohes Lastmoment am Rotor oder aufgrund der Trägheit des Rotors/der bewegten Masse analog durch zu schnelle Beschleunigung). Bei intelligenter Stromregelung wird durch die Begrenzung des Spulenstroms der Motor dadurch im Regelfall nicht zu Schaden kommen, es kann jedoch vorkommen, dass der Rotor dem sich drehenden Magnetfelds durch die Spulenkommutierung nicht folgen kann und somit Schritte übersprungen werden. Das führt dazu, dass die berechnete Position aus Nullpunkt, Anzahl der Schritte und Vorschub pro Schritt nicht mehr mit der tatsächlichen Position der Last übereinstimmt.
Der DRV8711 Stepper Motor Gate Driver Chip integriert eine sogenannte Stall-Detection (Siehe [16]) durch Messung der *Back-EMF* der nicht bestromten Spulen,

[1] Electromotive Force

die eben auf solche Vorkommnisse hinweist. In so einem Fall wird dem Mikrocontroller ein Signal übermittelt und die momentane Fahrt ist abzubrechen, eine erneute Referenzfahrt auszuführen und, mit verringerter Beschleunigung/Last die ursprünglich geplante Fahrt neu aufzunehmen.
Um das Überspringen von Schritten (Sog. *Stalling*) weitgehend zu vermeiden, wird ein Geschwindigkeitsprofil zum Betreiben des Schrittmotors verwendet, das eine näherungsweise konstante Beschleunigung aufweist. (Siehe Abbildung 4.15)
In Anlehnung an [25] sind die allgemeinen Zusammenhänge eines Rotors bei eindimensionaler (einachsiger) Drehung mit konstanter Winkelbeschleunigung $\dot{\omega} = const.$, Winkelgeschwindigkeit ω und Winkelstellung des Rotors φ wie im Folgenden ausgeführt.
Durch zeitliche Integration der Winkelbeschleunigung kann im eindimensionalen Fall einfach die Winkelgeschwindigkeit ermittelt werden:

$$\omega(t) = \int_0^t \dot{\omega} dt = \dot{\omega} \cdot t \tag{4.6}$$

Durch weitere Integration kann auf den zurückgelegten Winkel des Rotors geschlossen werden.

$$\varphi(t) = \int_0^t w(t) dt = \frac{\dot{\omega} \cdot t^2}{2} \tag{4.7}$$

Weiters ergibt sich beim Schrittmotor mit der Anzahl n der bereits getätigten Schritte zur Zeit t und α als Winkel pro Schritt (durch die Abhängigkeit von α und n werden auch Microsteps berücksichtigt):

$$\varphi(t) = n \cdot \alpha \tag{4.8}$$

Durch Gleichsetzen von Gleichung (4.7) und Gleichung (4.8) ergibt sich für die Zeit t beim n-ten Schritt

$$t = \sqrt{\frac{2n\alpha}{\dot{\omega}}} \tag{4.9}$$

Zur Implementierung auf einem Mikrocontroller wird ein Timer benutzt. Dieser basiert im Fall des MPC5602D PIT[1] (Siehe [24, 646ff]) auf einem Free-Running

[1] Periodic Interrupt Timer

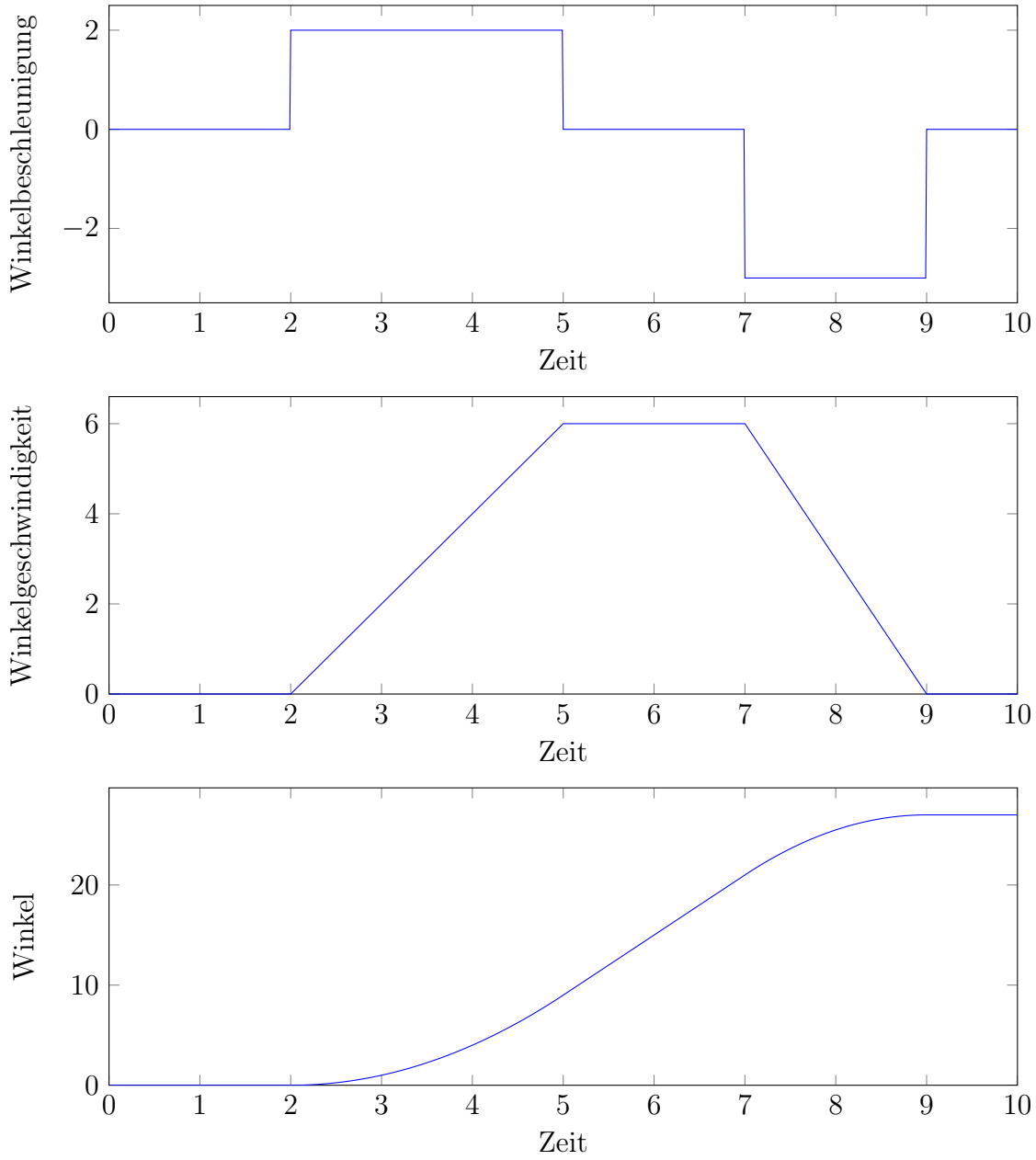


Abbildung 4.15: Beispielhafter Bewegungsverlauf für das Anfahren und Bremsen mit konstanter Beschleunigung

Counter mit Frequenz der Clock-Frequency, der bei Erreichen eines Compare-Values wieder auf den Startwert zurückgesetzt wird.
Deshalb müssen die Zeitabstände zwischen den einzelnen Schritten (Impulse an den Motor-Driver) und nicht die momentane Frequenz der Impulse ermittelt werden. Eine andere Möglichkeit, der Mikrocontroller-Hardware die Frequenz übergeben zu können, wäre ein PWM Signal zu verwenden. Hierbei stellt jedoch das Zählen der tatsächlich ausgesandten Impulse eine Herausforderung dar.
Die Frequenz steigt bei konstanter Beschleunigung nach Gleichung (4.6) linear mit der Zeit an, die Wartezeit zwischen zwei Schritten nimmt jedoch invers proportional ab.
Mit der Beziehung aus Gleichung (4.9) für konstante Beschleunigung kann nun nach [25] die Zeitdifferenz des n-ten und $n-1$-ten Schrittes ermittelt werden.

$$\Delta t_n = t_{n+1} - t_n = \sqrt{\frac{2(n+1)\alpha}{\dot{\omega}}} - \sqrt{\frac{2(n)\alpha}{\dot{\omega}}} \tag{4.10}$$

$$\Delta t_n = \sqrt{\frac{2\alpha}{\dot{\omega}}}(\sqrt{n+1} - \sqrt{n}) \tag{4.11}$$

Für $n = 0$ ergibt sich:

$$\Delta t_0 = \sqrt{\frac{2\alpha}{\dot{\omega}}} \tag{4.12}$$

und somit

$$\Delta t_n = \Delta t_0(\sqrt{n+1} - \sqrt{n}) \tag{4.13}$$

Die Berechnung von Quadratwurzeln ist für einen Mikrocontroller relativ aufwendig. Daher wird versucht, diese (bisher exakte) Beziehung anzunähern:
Der schrittabhängige Verzögerungsfaktor $c_t(n)$ ergibt sich als Quotient zweier aufeinanderfolgender Zeitdifferenzen als:

$$c_t(n) = \frac{\Delta t_n}{\Delta t_{n-1}} = \frac{\Delta t_0(\sqrt{n+1} - \sqrt{n})}{\Delta t_0(\sqrt{n} - \sqrt{n-1})} \tag{4.14}$$

Umformungen analog [25] ergeben:

$$c_t(n) = \frac{\sqrt{n}(\sqrt{1+\frac{1}{n}} - 1)}{\sqrt{n}(1 - \sqrt{1-\frac{1}{n}})} \tag{4.15}$$

Als quadratische Laurent-Reihenentwicklung um $n = \infty$ ergibt sich:

$$\sqrt{1 \pm \frac{1}{n}} \ddot{=} 1 \pm \frac{1}{2n} - \frac{1}{8n^2} + \mathcal{O}(\frac{1}{n^3}) \tag{4.16}$$

Eingesetzt:

$$c_t^{sq}(n) \ddot{=} \frac{1 + \frac{1}{2n} - \frac{1}{8n^2} - 1}{1 - (1 - \frac{1}{2n} - \frac{1}{8n^2})} = \frac{\frac{1}{2n} - \frac{1}{8n^2}}{\frac{1}{2n} + \frac{1}{8n^2}} = \frac{4n-1}{4n+1} \tag{4.17}$$

Und somit eine quadratisch angenäherte Iterationsvorschrift, die bloß Grundrechenarten erfordert:

$$\Delta t_n^{sq} = c_t^{sq}(n) \cdot \Delta t_{n-1}^{sq} \qquad \text{mit } \Delta t_0^{sq} = \Delta t_0 \tag{4.18}$$

Um die Genauigkeit der Annäherung in Bezug auf die exakte Lösung in Gleichung (4.11) zu analysieren, wird der Fehler der Annäherung $e^{sq} = \frac{|\Delta t_n - \Delta t_n^{sq}|}{\Delta t_n}$ mit *Wolfram Mathematica* berechnet.

Im Folgenden werden je für die exakte und für die quadratisch angenäherte Abschätzung eine Funktion definiert, die eine Liste für k Schritte für Δt_n und Δt_n^{sq} generieren.

```
In[1]:= approxSolutionTable[k_]:=Flatten[RecurrenceTable[{c[n]==
        c[n-1] (4 n-1)/(4 n+1),c[0]==c0},{c},{n,0,k}]];
        exactSolutionTable[k_]:=Table[c0(Sqrt[n+1]-Sqrt[n]),{n,0,k}];
```

Die ersten vier Werte in Abhängigkeit von $c_0 = \Delta t_0$ sind somit:

```
In[2]:= N[exactSolutionTable[3]]

        N[approxSolutionTable[3]]

Out[2]= {c0, 0.414214`c0, 0.317837`c0, 0.267949`c0}

Out[3]= {c0, 0.6`c0, 0.466667`c0, 0.394872`c0}
```

Im Folgenden wird der relative Fehler für die ersten vier Werte berechnet, wobei mit `Refine` die Annahme eingebracht wird, dass $c_0 = \Delta t_0 > 0$ gilt:

```
In[4]:= count=3;
        error=
        Abs[exactSolutionTable[count]-approxSolutionTable[count]]/
                        exactSolutionTable[count];

        N[Refine[error, c0>0]]
```

```
Out[4]= {0.`,0.448528`,0.468257`,0.473682`}
```

Es lässt sich erkennen, dass ab $n = 1$ der Wert der Näherung um beinahe 45 % vom exakten Wert abweicht. Darüberhinaus lässt sich erkennen, dass bei weiteren Iteration der entstandene Fehler nicht stark steigt.
Eine Mittelung des relativen Fehlers über die ersten 10000 Iterationen ergibt:

```
In[5]:= count=10000;

        error=N[Total[Refine[
        Abs[ exactSolutionTable[count]-approxSolutionTable[count]]
        ------------------------------------------------------------,
                            exactSolutionTable[count]

        c0>0]]/count]

Out[5]= 0.479331`
```

Unter der Annahme, dass ein Faktor η existiert, der mit $c_0 = \Delta t_0$ multipliziert einen besseren Startwert als c_0 selbst ergibt, lässt sich ein Minimierungsproblem definieren.
Hierfür wird die Funktion um diesen Faktor η erweitert:

```
In[6]:= approxSolutionTableOpt[k_,η_]:=RecurrenceTable[{c[n]==
        c[n-1] (4 n-1)
        --------------,c[0]==η c0},{c},{n,0,k}];
            4 n+1
```

Nun kann mit `NMinimize` der Error in Hinsicht auf η numerisch minimiert werden. Um die Berechnung zu vereinfachen, wird $c_0 = 1$ gesetzt - auf das Ergebnis hat dies aufgrund der Verwendung des *relativen* Fehlers keinerlei Einfluss.

```
In[7]:= c0=1;
        count=100000;
        exactSolution=exactSolutionTable[count];

        NMinimize[Total[Abs[exactSolution-approxSolutionTableOpt[
        count,η]]],η]
```

Hier wurde für die ersten 100000 berechneten Werte die Abweichung von der exakten Lösung minimiert. Dies führt auf $\eta = 0.676702$ und in Folge auf einen relativen Fehler von $e = 0.107\,\%$, was die Anforderungen an Genauigkeit erfüllt.

4.4 Software

Eine Auswahl des Quellcodes zum Betrieb der Steuerelektronik ist Anhang A zu entnehmen.
Dieser grobe Entwurf der Software zur Positionssteuerung der X, Y und Z-Achse wurde, mangels gefertigter Steuerelektronik und mechanischer Konstruktion zum momentanen Stand noch nicht getestet und beruht nur auf theoretischen Berechnungen, wie Abschnitt 4.3 zu entnehmen.

4.4.1 Kommunikation mit einem Host-PC

Die Kommunikation mit einem Host-PC geschieht über einen auf der Platine integrierten USB-UART Bridge-Controller. (Siehe Abschnitt 4.1.1)
Über einen FTDI Virtual Serial Port Treiber[1], der für LINUX, Mac OS und Windows verfügbar ist, kann mit der Steuerelektronik über jede handelsübliche serielle Schnittstelle kommuniziert werden.
Hierfür wurde entwurfsweise ein Protokoll zur Kommunikation entwickelt, um 5 Motoren mit, entwurfsweise noch hartkodierten Geschwindigkeits- und Beschleunigungsprofilen eine über serielle Kommunikation vorgegebene Anzahl an Schritten in eine beliebige Richtung zu bewegen.
Dies wird mit folgendem Message-Aufbau über eine serielle Schnittstelle gesteuert:

0x80 0x80	0x03	0x00 0x06 0x00 0x00	0xE1
Message Start	Motor ID	Schritt-Endzustand	CRC8-Checksumme

Tabelle 4.2: Beispiel einer Serial-Message zur Motorsteuerung

Eine Message besteht in diesem Fall aus 8 Bytes. Die ersten beiden Bytes, diese bilden die Startsequenz einer Message dieses Protokolls, müssen bei Übertragung von Information an die Steuerelektronik 0x80 0x80 sein.
Das nächste Byte beinhaltet die ID des anzusteuernden Motors. (In diesem Fall mit 5 ansteuerbaren Motoren 0x01-0x05, hier verwendet 0x03 für Motor 3)

[1] `http://www.ftdichip.com/Drivers/VCP.htm`

Die nächsten 4 Bytes beinhalten den Soll-Endzustand ausgehend von einem Nullpunkt. Wurde bisher keine Referenzfahrt getätigt (d.h. wurden die entsprechenden Initiatoren/Endschalter noch nicht aktiviert), wird der Einschaltzustand als Nullzustand angenommen. Wurden bereits Endschalter ausgelöst, wurde beim Aktivieren ebender Initiatoren der momentane Schrittcounter auf Null gesetzt. Somit wird bei einer Referenzfahrt ein wiederholbarer Nullpunkt gesetzt. Hierbei gibt das MSB[1] dieser 4 Bytes die anzufahrende Richtung an. Wird beispielsweise das MSB auf 1 gesetzt, kann eine negative Schrittanzahl vom momentanen Nullpunkt (bzw. Einschaltzustand) angefahren werden. Hierdurch ist es auch mit diesem sehr einfachen Protokoll ohne Definition von komplexeren Nachrichten möglich, eine Referenzfahrt in eine vordefinierte Richtung durchzuführen.

0x80 0x80	0x01	0xFF 0xFF 0xFF 0xFF	0x62
Message Start	Motor ID	Schritt-Endzustand	CRC8-Checksumme

Tabelle 4.3: Beispiel einer Serial-Message zur Durchführung einer Referenzfahrt in eine vordefinierte Richtung

In Tabelle 4.3 ist eine Beispielinstruktion für eine Referenzfahrt angeführt. Nachdem der HEX-Code 0xFFFFFFFF das MSB 1 besitzt, ist der Zielzustand $n = -2^{31} - 1$ Mikroschritte. Dies würde bei $n = 256 \cdot 200$ Mikroschritten pro Umdrehung und einer Spindelsteigung von 5 mm einer Position von ungefähr -200 m vom momentanen Nullpunkt aus, in Y-Richtung gemessen, entsprechen. Dies führt also jedenfalls unweigerlich zur automatischen Beendigung des Verfahrvorganges durch Betätigung eines Endschalters.
Der Schritt-Endzustand wird, wie bereits angedeutet, in der Einheit kleinster Mikroschritte vom momentanen Nullzustand aus angegeben. Durch Verwendung des TI DRV8711 werden kleinste Mikroschritte von $\frac{1}{256}$ eines vollen Schrittes möglich. Das bedeutet, dass ein Schritt-Endzustand von 0x60000 folgende Information beinhaltet: Umgerechnet in Dezimaldarstellung ergibt dies einen Wert von 16^4. Das

[1] Most Significant Bit

bedeutet, dass die instruierte Endposition 16^4 kleinste Mikroschritte vom Nullpunkt entfernt ist. Das bedeutet wiederum: Mit einem Schritt-Endzustand von 0x60000 wird direkt nach dem Einschalten mit der momentanen Einstellung ein Äquivalent von $\frac{16^4}{256} = 256$ vollen Schritten getätigt. Dies entspricht (mit 200 Schritten pro Umdrehung nach [4]) einer Drehung von $n = \frac{256}{200} = 1.28$ Umdrehungen von dem Referenzzustand aus. (Sei dies der Einschaltzustand oder ein Referenzpunkt nach einer Referenzfahrt).

Mit dem momentanen Stand der Software sollte darauf geachtet werden, nur Vielfache Werte der momentanen Schritt-Einstellung (diese ist im Moment hartkodiert gespeichert) zu tätigen. Ansonsten kann eine Positionsgenauigkeit nicht garantiert werden.

Das bedeutet, dass bei Vollschritten nur Vielfache von 256 Mikroschritten und bei Viertelschritten nur Vielfache von 64 Mikroschritten getätigt werden sollten.

Das letzte Byte der Message bildet eine CRC[1]8-Checksumme (Analog zum *Ethernet*-Protokoll) zum Überprüfen der Datenintegrität. Die Checksumme basiert auf den Bytes 3-7 (gezählt vom MSB) der Nachricht, also des tatsächlichen Inhalts der Message, beinhaltend die Motor-ID und alle Bytes des Schritt-Endzustands.

Schlägt der Mikrocontroller-interne Abgleich des berechneten und übertragenen CRC8-Wertes fehl und ist somit mit hoher Wahrscheinlichkeit ein Übertragungsfehler aufgetreten, wird die erhaltene Nachricht zurück an den Host Computer (jedoch mit zwei veränderten Message Start Bytes in der Form 0x81 0x81) gesendet. Dieser ist hier somit verantwortlich, eine erneute Übertragung zu starten. Bei fehlerhafter CRC8-Checksumme wird keine Bewegung ausgeführt.

0x81 0x81	0x01	0xFF 0xFF 0xFF 0xFF	0x62
Message Start	Motor ID	Schritt-Endzustand	CRC8-Checksumme

Tabelle 4.4: Beispiel einer Rück-Antwort der Steuerplatine bei fehlerhafter CRC8-Checksumme

[1]Cyclic Redundancy Check

Kapitel 5

Résumé

In dieser Arbeit wurden diverse Alternativsysteme zu einem Eigenbau eines Flächenportals als Teil einer flexiblen Fertigungszelle zur Bestückung einer Portalfräse in Betracht gezogen. Nach sorgfältigem Abwägen ist die Entscheidung zugunsten eines Eigenbaus gefallen. Analog zur bestehenden Portalfräse wurde ein Flächenportal entworfen, angetrieben über Lineargetriebe mit Schrittmotoren.
Zur Steuerung des Flächenportals wurde weiters ein Entwurf einer integrierten Steuerplatine erstellt, die unter anderem Controller für 5 Schrittmotoren, Anbindungen für Initiatoren sowie einen dedizierten Mikrocontroller und eine USB-Schnittstelle zur Kommunikation mit einem Host-PC bietet - und zwar unter Anbetracht galvanischer Isolation, um bestehendes Equipment durch Fehlfunktionen der Steuerung nicht zu gefährden.
Schlussendlich wurde eine Firmware für die Steuerung geschrieben, die als *proof-of-concept* Positionssteuerungen des Flächenportals über eine serielle Schnittstelle (USB) ermöglicht. Zwischen der Steuerplatine und einem Host-PC werden Nachrichten über diese Schnittstelle basierend auf einem in einfachster Weise konstruierten Kommunikationsprotokoll ausgetauscht. Dieses Protokoll unterstützt zur Überprüfung der Datenintegrität weiters eine *Frame Check Sequence* in Form einer CRC8-Checksumme.
Um die notwendige Rechenleistung des Mikrocontrollers minimal zu halten, wurde darüberhinaus eine Approximation nichtlinearer Beziehungen durchgeführt, die

Durchfahren eines vorgegebenen Beschleunigungsprofils mit geringstem Rechenaufwand ermöglicht.

Wie anfangs erwähnt, stellt diese Diplomarbeit nur einen Teil des gesamten Projektes dar. In Kombination mit den Erkenntnissen aus der komplementierenden Arbeit wird als nächster Schritt mit dem Aufbau der Anlage begonnen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *Portalfräse | High-Z T-Serie Portalfräse mit Kugelgewinde.* URL: https://www.cnc-step.de/produkte/graviermaschine-cnc-fraesmaschine-kugelgewinde/76-portalfraese-s1000t (besucht am 06. 09. 2016).

[2] Professor Dr.-Ing. Ulrich Grünhaupt Professor Dr.-Ing. Hans-Jürgen Gevatter. *Handbuch der Mess- und Automatisierungstechnik in der Produktion.* 2. Aufl. 10. The address: Springer-Verlag Berlin Heidelberg 2006, 2006. ISBN: 3540212078.

[3] *Kugelgewindetriebe haben Wirkungsgrad von 98%.* URL: http://www.maschinenmarkt.vogel.de/index.cfm?pid=1610&pk=256343 (besucht am 15. 09. 2016).

[4] *Stepper Motor Datasheet.* ST5918L3008-A. Nanotec Electronic. März 2007.

[5] *Überschlägige Ermittlung des Reibungsmoments.* URL: http://www.skf.com/de/products/bearings-units-housings/ball-bearings/principles/friction/estimating-frictional-moment/index.html (besucht am 01. 09. 2016).

[6] *FAQ - Nanotec, Drehen und Beschleunigen von Schwungmassen.* URL: http://de.nanotec.com/support/faq/ (besucht am 16. 09. 2016).

[7] *Onlinestore for linear technology from lineareasy.com - Ball Screws from lineareasy.com.* URL: http://lineareasy.de/shop/en/ball-screw/kgt-r-2505-rh-t5.html (besucht am 05. 09. 2016).

[8] *Kugelgewindetriebe - Technische Grundlagen.* URL: http://www.nadella.de/fileadmin/nadella/downloads/kgt/KGT_Grundlagen.pdf (besucht am 10. 09. 2016).

[9] *ARC / HRC / ERC Linearführungen*. URL: http://www.cpc-europa.de/images/pdfs/ARC-Standard.pdf.pdf (besucht am 05. 08. 2016).

[10] *Rolling Bearings Catalogue*. PUB BU/P1 10000 EN. SKF. Okt. 2012.

[11] *EKH - R+W Kupplungen*. URL: http://www.rw-kupplungen.de/produkte/praezisionskupplungen/elastomerkupplungen/ekh.html (besucht am 01. 07. 2016).

[12] *Nanotec: ST5918 Schrittmotor - NEMA 23*. URL: http://de.nanotec.com/produkte/497-st5918-schrittmotor-nema-23/ (besucht am 16. 09. 2016).

[13] *Onlinestore for linear technology from lineareasy.com - Ball Screws from lineareasy.com*. URL: http://lineareasy.de/shop/en/ball-screw/kgt-r-1605-rh-t5.html (besucht am 05. 09. 2016).

[14] *MPC560xB/32-bit MCU/Body-Electronic/NXP*. URL: http://www.nxp.com/products/automotive-products/microcontrollers-and-processors/16-bit-s12-s12x-mcus/ultra-reliable-mpc56xb-mcu-for-automotive-industrial-general-purpose:MPC560xB (besucht am 16. 01. 2017).

[15] *Microcontroller Data Sheet*. MPC5602D. Rev. 6. Freescale Semiconductor. Jan. 2013.

[16] *Stepper Motor Gate Driver with On-Chip 1/256 Micro-Stepping Indexer and Stall Detect Datasheet*. DRV8711. Rev. F. Texas Instruments. Juni 2016.

[17] *Dual 60-V N-Channel NexFET Power MOSFET Datasheet*. CSD88537ND. SLPS455A. Texas Instruments. Aug. 2014.

[18] *USB to BASIC UART IC Datasheet*. FT230X. Version 1.4. FTDI. Mai 2016.

[19] *Robust EMC, Low Power, Dual-Channel Digital Isolators Datasheet*. ISO732x. SLLSEK8C. Texas Instruments. Apr. 2015.

[20] *6V to 42V, 1A High Output Voltage Power Module Datasheet*. LMZ14201H. SNVS690H. Texas Instruments. Okt. 2015.

[21] *Enobi - Rohrmotor 4adrig Anschluss*. URL: http://www.enobi.de/bilder/detail/rohrmotor-4adrig-anschluss.jpg (besucht am 10. 12. 2016).

[22] *Rapid Turn-On AC Power Switch Datasheet*. CPC1966YX6. DS-CPC1966YX6-R01. IXYS Integrated Circuits Division. Okt. 2013.

[23] *Logic level FET Data Sheet*. BSS123. Rev. 1.0. Philips Semiconductors. Aug. 2000.

[24] *Microcontroller Reference Manual*. MPC5602D. Rev. 4.2. Freescale Semiconductor. Dez. 2013.

[25] *Application Note: Linear speed control of stepper motor*. AVR446. Rev. 8017A-AVR-06/06. ATMEL. 2006.

[26] Wikimedia Commons. *The two basic states of an H bridge*. File: `H bridge operating.svg`. 2006. URL: `https://commons.wikimedia.org/wiki/File:H_bridge_operating.svg`.

[27] Wikimedia Commons. *Series RL circuit*. File: `Series-RL.png`. 2006. URL: `https://en.wikipedia.org/wiki/RL_circuit#/media/File:Series-RL.png`.

Anhang A

Quellcode Steuerung

Code A.1: main.c

```
1  //#include "MPC5602D.h"
2  #include "typedefs.h"
3  #include "jdp.h"
4
5  #include "sys_init.h"
6  #include "IntcInterrupts.h"
7
8  #include "definitions.h"
9
10 #include <math.h>
11
12 //Interrupt handler definition for endswitches
13 void interrupt_handler_sensor(void);
14
15 //Interrupt handler definitions for making steps
16 void interrupt_handler_PITX(void);
17 void interrupt_handler_PITY(void);
18 void interrupt_handler_PITZ(void);
19 void interrupt_handler_PITR(void);
20
21 void interrupt_handler_motorWatchdog(void);
22
23 position_struct getPosition();
24
25 unsigned char positionHasBeenReferenced = 0;
26
27 //Counting global steps in multiples of the smallest microstep (1/256)
28 int32_t stepCounterX = 0;
29 int32_t stepCounterY = 0;
30 int32_t stepCounterZ = 0;
31 int32_t stepCounterR = 0;
32
33 float mmPerSmallestMicrostepX = SCREWPITCH/(256.*FULLSTEPSPERREVOLUTION);
34 float mmPerSmallestMicrostepY = SCREWPITCH/(256.*FULLSTEPSPERREVOLUTION);
35 float mmPerSmallestMicrostepZ = SCREWPITCH/(256.*FULLSTEPSPERREVOLUTION);
36 float degPerSmallestMicrostepR = 360/(256.*FULLSTEPSPERREVOLUTION*RGEARRATIO);
37
38 //Acceleration slopes and target RPMs for each axis
39 int accelSpeedX = 1000; //deg per s^2
40 int decelSpeedX = 1000; //deg per s^2
41 int maxSpeedX = 1500; //deg per s
42 int accelSpeedY = 1000;
43 int decelSpeedY = 1000;
44 int maxSpeedY = 1500;
45 int accelSpeedZ = 1000;
46 int decelSpeedZ = 1000;
47 int maxSpeedZ = 1500;
48 int accelSpeedR = 1000;
49 int decelSpeedR = 1000;
50 int maxSpeedR = 1500;
51
52 unsigned char incomingByte, lastIncomingByte, CRC1;
53 unsigned char dataLine[17] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
54 unsigned char checkCRCLine[3] = {0, 0, 0};
55
56 motor_instruction createNewMovementInstruction(int32_t currentSteps, int32_t targetSteps,
       float acceleration, float deceleration, float maxSpeed, unsigned int inactive, unsigned
       int referenceInstruction);
57
58 void scheduleNewMotorMovement(int32_t targetGlobalSteps, motorID id);
59
```

```
60  motor_instruction currentInstructionX = 0;
61  motor_instruction currentInstructionY = 0;
62  motor_instruction currentInstructionZ = 0;
63  motor_instruction currentInstructionR = 0;
64
65  unsigned int freeRunningInstructionID = 0;
66
67  uint32_t main(void) {
68          volatile uint32_t i = 0;
69
70          sys_init_fnc();
71
72          asm("wrteei 1"); //Enable external interrupts
73
74          //Install interrupt handlers for end switches with priority 15 (highest)
75          INTC_InstallINTCInterruptHandler(interrupt_handler_sensor, 41, 15); //EIRQ 0-7
76          INTC_InstallINTCInterruptHandler(interrupt_handler_sensor, 42, 15); //EIRQ 8-15
77          INTC_InstallINTCInterruptHandler(interrupt_handler_sensor, 43, 15); //EIRQ 16-23
78
79          //Initialize motor step interrupt with priority 13
80          INTC_InstallINTCInterruptHandler(interrupt_handler_PITX, 59, 14);
81          INTC_InstallINTCInterruptHandler(interrupt_handler_PITY, 60, 14);
82          INTC_InstallINTCInterruptHandler(interrupt_handler_PITZ, 61, 14);
83          INTC_InstallINTCInterruptHandler(interrupt_handler_PITR, 127, 14);
84
85          //Initialize motor watchdog interrupt with priority 13
86          INTC_InstallINTCInterruptHandler(interrupt_handler_motorWatchdog, 30, 13);
87
88          INTC.CPR.R = 0x0; //Enable all interrupts by setting minimum priority to 0
89
90          RESET_DRV = 0x1; //Reset all motor drivers
91          GLOBALSLEEPN = 0x0; //Disable sleep mode for all motors
92          for (i=0; i<48000; i++){ //Wait at least 1ms (48000 clock cycles)
93                  asm("nop"); //Wait
94          }
95          RESET_DRV = 0x0;
96          DRV_Init(); //Initialize motor drivers and enable motors
97
98          MOTOR_WATCHDOG_INTERVAL = 1000; //Microseconds
99          MOTOR_WATCHDOG_ENABLE = 0x1; //Enable motor watchdog
100
101         Uart0BufInit(); //Initialize LINFLEX Module for UART
102
103         for (;;) {
104                 i++;
105                 checkUART();
106         }
107 }
108
109 void checkUART() {
110         unsigned char receivedMotorID = 0;
111         uint32_t receivedTargetSteps = 0;
112
113         Uart0RxFillBuf(); //Fill UART buffer
114         if(Uart0RxBufEmpty() != 1) //Check if message received
115         {
116                 lastIncomingByte = incomingByte;
117                 incomingByte = Uart0RxDataByte();
118
119                 if(incomingByte == 0x80 && lastIncomingByte == 0x80){ //Check if two 0x80
                        bytes were received (Start sequence)
120                     dataLine[0] = 0x80;
121                     countUART = 1;
122                     shouldRecordUART = 1; //Start to record incoming bytes
123                 }
124
125                 if(shouldRecordUART){
126                         dataLine[countUART1] = incomingByte;
127                 }
128
129                 countUART++;
130
131                 if(countUART >= 8){
132                         shouldRecordUART = 0;
133
134                         checkCRCLine[0] = dataLine[2]; //ID
135                         checkCRCLine[1] = dataLine[3]; //Data Byte
136                         checkCRCLine[2] = dataLine[4]; //Data Byte
137                         checkCRCLine[3] = dataLine[5]; //Data Byte
138                         checkCRCLine[4] = dataLine[6]; //Data Byte
139
140                         CRC1 = CRC8(checkCRCLine, 5); //Calculate CRC8 with length 5
141
142                         if(dataLine[7] == CRC1){ //If message valid
143                                 //Valid data
144                                 //Assemble data
145                                 receivedMotorID = dataLine[2];
146                                 receivedTargetSteps = dataLine[6] & (dataLine[5] << 8) & (
                                        dataLine[4] << 16) & (dataLine[3] << 24);
147
148                                 scheduleNewMotorMovement(receivedTargetSteps, receivedMotorID)
                                        ;
149
150                                 //Echo information back
151                                 Uart0TxMsg((unsigned char *)0x81, 1);
152                                 Uart0TxMsg((unsigned char *)0x81, 1);
153                                 Uart0TxMsg((unsigned char *)dataLine[2], 1);
154                                 Uart0TxMsg((unsigned char *)dataLine[3], 1);
155                                 Uart0TxMsg((unsigned char *)dataLine[4], 1);
156                                 Uart0TxMsg((unsigned char *)dataLine[5], 1);
157                                 Uart0TxMsg((unsigned char *)dataLine[6], 1);
```

```
158                                     Uart0TXMsg((unsigned char *)dataLine[7], 1); //Can use old CRC
                                            , because it was already validated
159                             }
160                     }
161             }
162     }
163
164     void scheduleNewMotorMovement(int32_t targetGlobalSteps, motorID id){
165             unsigned char dir = 0;
166
167             switch(id){
168             case MOTOR1:
169             case MOTOR2:
170                     if(targetGlobalSteps != stepCounterX){
171                             if(targetGlobalSteps > stepCounterX){
172                                     dir = 1;
173                             }
174                             DIR1 = dir;
175                             DIR2 = dir;
176                     }
177
178                     currentInstructionX = createNewMovementInstruction(stepCounterX,
                            targetGlobalSteps, accelSpeedX, decelSpeedX, maxSpeedX, 0, 0);
179                     PITX_INTERVAL = currentInstructionX.firstDelay;;
180                     PITX_ENABLED = 1;
181                     break;
182
183             case MOTOR3:
184                     if(targetGlobalSteps != stepCounterY){
185                             if(targetGlobalSteps > stepCounterY){
186                                     dir = 1;
187                             }
188                             DIR3 = dir;
189                     }
190
191                     currentInstructionY = createNewMovementInstruction(stepCounterY,
                            targetGlobalSteps, accelSpeedY, decelSpeedY, maxSpeedY, 0, 0);
192                     PITY_INTERVAL = currentInstructionY.firstDelay;;
193                     PITY_ENABLED = 1;
194                     break;
195
196             case MOTOR4:
197                     if(targetGlobalSteps != stepCounterZ){
198                             if(targetGlobalSteps > stepCounterZ){
199                                     dir = 1;
200                             }
201                             DIR4 = dir;
202                     }
203
204                     currentInstructionZ = createNewMovementInstruction(stepCounterZ,
                            targetGlobalSteps, accelSpeedZ, decelSpeedZ, maxSpeedZ, 0, 0);
205                     PITZ_INTERVAL = currentInstructionZ.firstDelay;;
206                     PITZ_ENABLED = 1;
207                     break;
208
209             case MOTOR5:
210                     if(targetGlobalSteps != stepCounterR){
211                             if(targetGlobalSteps > stepCounterR){
212                                     dir = 1;
213                             }
214                             DIR5 = dir;
215                     }
216
217                     currentInstructionR = createNewMovementInstruction(stepCounterR,
                            targetGlobalSteps, accelSpeedR, decelSpeedR, maxSpeedR, 0, 0);
218                     PITR_INTERVAL = currentInstructionR.firstDelay;;
219                     PITR_ENABLED = 1;
220                     break;
221             }
222     }
223
224     motor_instruction createNewMovementInstruction(int32_t currentSteps, int32_t targetSteps, int
            acceleration, int deceleration, int maxSpeed, unsigned int inactive, unsigned int
            referenceInstruction){
225             motor_instruction newInstruction;
226             int max_accel_steps_speed;
227             int max_accel_steps_decel;
228
229             newInstruction.stepsToTake = targetSteps>currentSteps?targetSteps-currentSteps:
                    currentSteps-targetSteps;
230             newInstruction.id = freeRunningInstructionID++;
231             newInstruction.inactive = inactive;
232             newInstruction.referenceInstruction = referenceInstruction;
233
234             max_accel_steps_speed = pow(maxSpeed, 2)/(2.*ALPHA*acceleration); //Maximum number of
                    acceleration steps to reach set speed
235             max_accel_steps_decel = targetSteps*deceleration/(acceleration+deceleration); //
                    Maximum number of acceleration steps before deceleration has to set in
236             decel_steps_speed = max_accel_steps_speed*acceleration/deceleration;
237
238             if(max_accel_steps_speed > max_accel_steps_decel){
239                     //Max speed not reached
240                     newInstruction.accelSteps = max_accel_steps_decel;
241                     newInstruction.startDecelSteps = max_accel_steps_decel;
242             }else{
243                     //Max speed reached, normal phase
244                     newInstruction.accelSteps = max_accel_steps_speed;
245                     newInstruction.startDecelSteps = decel_steps_speed;
246             }
247
248             newInstruction.firstDelay = (int)(CLOCKFREQ*sqrt(2*ALPHA/acceleration));
```

```
249         return motor_instruction;
250 }
251
252 void interrupt_handler_motorWatchdog(void){
253         STM.CIR0.R = 0x00000001; //Reset interrupt flag at first for more exact timing
254         MOTOR_WATCHDOG_ENABLE = 0x0; //Disable STM
255         MOTOR_WATCHDOG_COUNT = 0x0; //Reset STM Counter
256         MOTOR_WATCHDOG_ENABLE = 0x1; //Enable STM
257 }
258
259 void interrupt_handler_PITX(void){
260         uint32_t i = 0;
261         uint32_t smallestMicroStepsTaken = 1 << (8-stepModeX); //calculate microsteps from
                stepping mode
262         //Create rising edge
263         PWMCNTR1 = 0x1;
264         PWMCNTR2 = 0x1;
265
266         currentInstructionX.stepsTaken += smallestMicroStepsTaken;
267         if((DIR1 & DIR2) == 1){
268                 stepCounterX += smallestMicroStepsTaken;
269         }else{
270                 stepCounterX -= smallestMicroStepsTaken;
271         }
272
273         for (i=0; i<4800; i++){ //Wait at least .1ms (4800 clock cycles)
274                 asm("nop"); //Wait
275         }
276         PWMCNTR1 = 0x0;
277         PWMCNTR2 = 0x0;
278
279         if(currentInstructionX.stepsTaken < currentInstructionX.accelSteps){
280                 //Accelerate
281                 //Calculate new PIT Interval/Delay
282                 PITX_INTERVAL = PITX_INTERVAL*((4*currentInstructionX.stepsTaken-1)/(4*
                        currentInstructionX.stepsTaken+1));
283         }else if(currentInstructionX.stepsTaken > currentInstructionX.startDecelSteps){
284                 //Decelerate
285                 //Calculate new PIT Interval/Delay
286                 PITX_INTERVAL = PITX_INTERVAL*((4*currentInstructionX.stepsTaken+1)/(4*
                        currentInstructionX.stepsTaken-1));
287         }
288
289         if(currentInstructionX.stepsTaken >= currentInstructionX.stepsToTake){
290                 //Disable PIT and remove Instruction
291                 PITX_ENABLED = 0;
292                 currentInstructionX = createMovementInstruction(0, 0, 0, 0, 0, 1, 0);
293         }
294
295         PIT.CH[0].TFLG.R = 0x00000001; //Reset interrupt flag
296 }
297
298 void interrupt_handler_PITY(void){
299         uint32_t i = 0;
300         uint32_t smallestMicroStepsTaken = 1 << (8-stepModeY); //calculate microsteps from
                stepping mode
301         //Create rising edge
302         PWMCNTR3 = 0x1;
303
304         currentInstructionY.stepsTaken += smallestMicroStepsTaken;
305         if(DIR3 == 1){
306                 stepCounterY += smallestMicroStepsTaken;
307         }else{
308                 stepCounterY -= smallestMicroStepsTaken;
309         }
310
311         for (i=0; i<4800; i++){ //Wait at least .1ms (4800 clock cycles)
312                 asm("nop"); //Wait
313         }
314         PWMCNTR3 = 0x0;
315
316         if(currentInstructionY.stepsTaken < currentInstructionY.accelSteps){
317                 //Accelerate
318                 //Calculate new PIT Interval/Delay
319                 PITY_INTERVAL = PITY_INTERVAL*((4*currentInstructionY.stepsTaken-1)/(4*
                        currentInstructionY.stepsTaken+1));
320         }else if(currentInstructionY.stepsTaken > currentInstructionY.startDecelSteps){
321                 //Decelerate
322                 //Calculate new PIT Interval/Delay
323                 PITY_INTERVAL = PITY_INTERVAL*((4*currentInstructionY.stepsTaken+1)/(4*
                        currentInstructionY.stepsTaken-1));
324         }
325
326         if(currentInstructionY.stepsTaken >= currentInstructionY.stepsToTake){
327                 //Disable PIT and remove Instruction
328                 PITY_ENABLED = 0;
329                 currentInstructionY = createMovementInstruction(0, 0, 0, 0, 0, 1, 0);
330         }
331
332         PIT.CH[1].TFLG.R = 0x00000001; //Reset interrupt flag
333 }
334
335 void interrupt_handler_PITZ(void){
336         uint32_t i = 0;
337         uint32_t smallestMicroStepsTaken = 1 << (8-stepModeZ); //calculate microsteps from
                stepping mode
338         //Create rising edge
339         PWMCNTR3 = 0x1;
340
341         currentInstructionZ.stepsTaken += smallestMicroStepsTaken;
342         if(DIR3 == 1){
```

```
343                     stepCounterZ += smallestMicroStepsTaken;
344             }else{
345                     stepCounterZ -= smallestMicroStepsTaken;
346             }
347
348             for (i=0; i<4800; i++){ //Wait at least .1ms (4800 clock cycles)
349                     asm("nop"); //Wait
350             }
351             PWMCNTR3 = 0x0;
352
353             if(currentInstructionZ.stepsTaken < currentInstructionZ.accelSteps){
354                     //Accelerate
355                     //Calculate new PIT Interval/Delay
356                     PITZ_INTERVAL = PITZ_INTERVAL*((4*currentInstructionZ.stepsTaken-1)/(4*
                            currentInstructionZ.stepsTaken+1));
357             }else if(currentInstructionZ.stepsTaken > currentInstructionZ.startDecelSteps){
358                     //Decelerate
359                     //Calculate new PIT Interval/Delay
360                     PITZ_INTERVAL = PITZ_INTERVAL*((4*currentInstructionZ.stepsTaken+1)/(4*
                            currentInstructionZ.stepsTaken-1));
361             }
362
363             if(currentInstructionZ.stepsTaken >= currentInstructionZ.stepsToTake){
364                     //Disable PIT and remove Instruction
365                     PITZ_ENABLED = 0;
366                     currentInstructionZ = createMovementInstruction(0, 0, 0, 0, 0, 1, 0);
367             }
368
369             PIT.CH[2].TFLG.R = 0x00000001; //Reset interrupt flag
370     }
371
372     void interrupt_handler_PITR(void){
373             uint32_t i = 0;
374             uint32_t smallestMicroStepsTaken = 1 << (8-stepModeR); //calculate microsteps from
                    stepping mode
375             //Create rising edge
376             PWMCNTR4 = 0x1;
377
378             currentInstructionR.stepsTaken += smallestMicroStepsTaken;
379             if(DIR5 == 1){
380                     stepCounterR += smallestMicroStepsTaken;
381             }else{
382                     stepCounterR -= smallestMicroStepsTaken;
383             }
384
385             for (i=0; i<4800; i++){ //Wait at least .1ms (4800 clock cycles)
386                     asm("nop"); //Wait
387             }
388             PWMCNTR4 = 0x0;
389
390             if(currentInstructionR.stepsTaken < currentInstructionR.accelSteps){
391                     //Accelerate
392                     //Calculate new PIT Interval/Delay
393                     PITR_INTERVAL = PITR_INTERVAL*((4*currentInstructionR.stepsTaken-1)/(4*
                            currentInstructionR.stepsTaken+1));
394             }else if(currentInstructionR.stepsTaken > currentInstructionR.startDecelSteps){
395                     //Decelerate
396                     //Calculate new PIT Interval/Delay
397                     PITR_INTERVAL = PITR_INTERVAL*((4*currentInstructionR.stepsTaken+1)/(4*
                            currentInstructionR.stepsTaken-1));
398             }
399
400             if(currentInstructionR.stepsTaken >= currentInstructionR.stepsToTake){
401                     //Disable PIT and remove Instruction
402                     PITR_ENABLED = 0;
403                     currentInstructionR = createMovementInstruction(0, 0, 0, 0, 0, 1, 0);
404             }
405
406             PIT.CH[4].TFLG.R = 0x00000001; //Reset interrupt flag
407     }
408
409     void interrupt_handler_sensor(void){
410             //Something happened - Check all corresponding endswitches
411             if(SENSOR1 == 0x1 || SENSOR2 == 0x1){
412                     PITX_ENABLE = 0;
413                     if(SENSOR1 == 0x1){
414                             stepCounterX = 0;
415
416                     }
417             }
418             if(SENSOR3 == 0x1 || SENSOR4 == 0x1){
419                     PITY_ENABLE = 0;
420                     if(SENSOR3 == 0x1){
421                             stepCounterY = 0;
422                     }
423             }
424             if(SENSOR5 == 0x1 || SENSOR6 == 0x1){
425                     PITZ_ENABLE = 0;
426                     if(SENSOR5 == 0x1){
427                             stepCounterZ = 0;
428                     }
429             }
430             if(SENSOR7 == 0x1){
431                     stepCounterR = 0;
432             }
433     }
434
435     position_struct getPosition(void){
436             position_struct pos;
437             pos.x = mmPerSmallestMicrostepX*stepCounterX;
438             pos.y = mmPerSmallestMicrostepY*stepCounterY;
```

```
439             pos.z = mmPerSmallestMicrostepZ*stepCounterZ;
440             pos.r = degPerSmallestMicrostepR*stepCounterR;
441             pos.referenced = positionHasBeenReferenced;
442             return pos;
443     }
```

Code A.2: main.c

```
1   /*
2    * definitions.h
3    *
4    *  Created on: Aug 28, 2016
5    *      Author: Fabian Scheidl
6    */
7
8   #include "jdp.h"
9
10  #ifndef DEFINITIONS_H_
11  #define DEFINITIONS_H_
12
13  #define FULLSTEPSPERREVOLUTION 200
14  #define SCREWPITCH 5
15  #define RGEARRATIO 150
16
17  #define ALPHA 360./(256.*FULLSTEPSPERREVOLUTION)
18
19  #define CLOCKFREQ 48000000
20
21  #define PITX_INTERVAL PIT.CH[0].LDVAL.R
22  #define PITY_INTERVAL PIT.CH[1].LDVAL.R
23  #define PITZ_INTERVAL PIT.CH[2].LDVAL.R
24  #define PITR_INTERVAL PIT.CH[3].LDVAL.R
25
26  #define PITX_ENABLE PIT.CH[0].TCTRL.B.TEN
27  #define PITY_ENABLE PIT.CH[1].TCTRL.B.TEN
28  #define PITZ_ENABLE PIT.CH[2].TCTRL.B.TEN
29  #define PITR_ENABLE PIT.CH[3].TCTRL.B.TEN
30
31  #define MOTOR_WATCHDOG_INTERVAL STM.CMP0.R //Microseconds
32  #define MOTOR_WATCHDOG_ENABLE STM.CCR0.B.CEN
33  #define MOTOR_WATCHDOG_COUNT STM.CNT0.R
34
35  #define RESET_DRV SIU.GPDO[41].R
36
37  #define SENSOR1 SIU.GPDI[76].B.PDI
38  #define SENSOR2 SIU.GPDI[35].B.PDI
39  #define SENSOR3 SIU.GPDI[34].B.PDI
40  #define SENSOR4 SIU.GPDI[11].B.PDI
41  #define SENSOR5 SIU.GPDI[46].B.PDI
42  #define SENSOR6 SIU.GPDI[47].B.PDI
43  #define SENSOR7 SIU.GPDI[66].B.PDI
44
45  #define LED1 SIU.GPDO[32].R
46  #define LED2 SIU.GPDO[21].R
47  #define LED3 SIU.GPDO[22].R
48  #define LED4 SIU.GPDO[23].R
49
50  #define VALVE1 SIU.GPDO[16].R
51  #define VALVE2 SIU.GPDO[42].R
52  #define VALVE3 SIU.GPDO[43].R
53
54  #define SSR1 SIU.GPDO[17].R
55  #define SSR2 SIU.GPDO[38].R
56
57  #define STALLN1 SIU.GPDI[70].B.PDI
58  #define STALLN2 SIU.GPDI[69].B.PDI
59  #define STALLN3 SIU.GPDI[68].B.PDI
60  #define STALLN4 SIU.GPDI[36].B.PDI
61  #define STALLN5 SIU.GPDI[37].B.PDI
62  #define GLOBALSTALLN (STALLN1 | STALLN2 | STALLN3 | STALLN4 | STALLN5)
63
64  #define FAULTN1 SIU.GPDI[67].B.PDI
65  #define FAULTN2 SIU.GPDI[10].B.PDI
66  #define FAULTN3 SIU.GPDI[44].B.PDI
67  #define FAULTN4 SIU.GPDI[45].B.PDI
68  #define FAULTN5 SIU.GPDI[40].B.PDI
69  #define GLOBALFAULTN (STALLN1 | STALLN2 | STALLN3 | STALLN4 | STALLN5)
70
71  #define SLEEPN1 SIU.GPDO[74].R
72  #define SLEEPN2 SIU.GPDO[73].R
73  #define SLEEPN3 SIU.GPDO[72].R
74  #define SLEEPN4 SIU.GPDO[65].R
75  #define SLEEPN5 SIU.GPDO[39].R
76  #define GLOBALSLEEPN SLEEPN1 = SLEEPN2 = SLEEPN3 = SLEEPN4 = SLEEPN5
77
78  #define DIR1 SIU.GPDO[62].R
79  #define DIR2 SIU.GPDO[27].R
80  #define DIR3 SIU.GPDO[60].R
81  #define DIR4 SIU.GPDO[61].R
82  #define DIR5 SIU.GPDO[26].R
83
84  #define PWMCNTR1 SIU.GPDI[0].R
85  #define PWMCNTR2 SIU.GPDI[1].R
```

```
86  #define PWMCNTR3 SIU.GPDI[7].R
87  #define PWMCNTR4 SIU.GPDI[3].R
88  #define PWMCNTR5 SIU.GPDI[71].R
89
90  typedef enum {
91          MOTOR1 = 0x0,
92          MOTOR2 = 0x1,
93          MOTOR3 = 0x2,
94          MOTOR4 = 0x3,
95          MOTOR5 = 0x4
96  } motorID;
97
98  typedef struct {
99      float x; //mm
100     float y; //mm
101     float z; //mm
102     float r; //deg
103     unsigned char referenced;
104 } position_struct;
105
106 typedef struct {
107         unsigned int accelSteps; //Number of acceleration steps to take
108         unsigned int startDecelSteps; //Steps at which to start deceleration
109     unsigned int stepsTaken; //Number of steps taken in this movement so far
110     unsigned int id; //Identification
111     unsigned char inactive; //First bit is set if this movement is not an active movement or
            movement is empty
112     unsigned char referenceInstruction; //First bit is set if movement is set to find
            reference point
113     unsigned int firstDelay;
114     unsigned int stepsToTake;
115 } motor_instruction;
116
117 uint8_t SPI_Enable_Motor(uint8_t enable, motorID id);
118 void DRV_Init(void);
119 unsigned char CRC8(const unsigned char *data, unsigned char len);
120
121 extern unsigned char stepModeX;
122 extern unsigned char stepModeY;
123 extern unsigned char stepModeZ;
124 extern unsigned char stepModeR;
125
126 #endif /* DEFINITIONS_H_ */
```

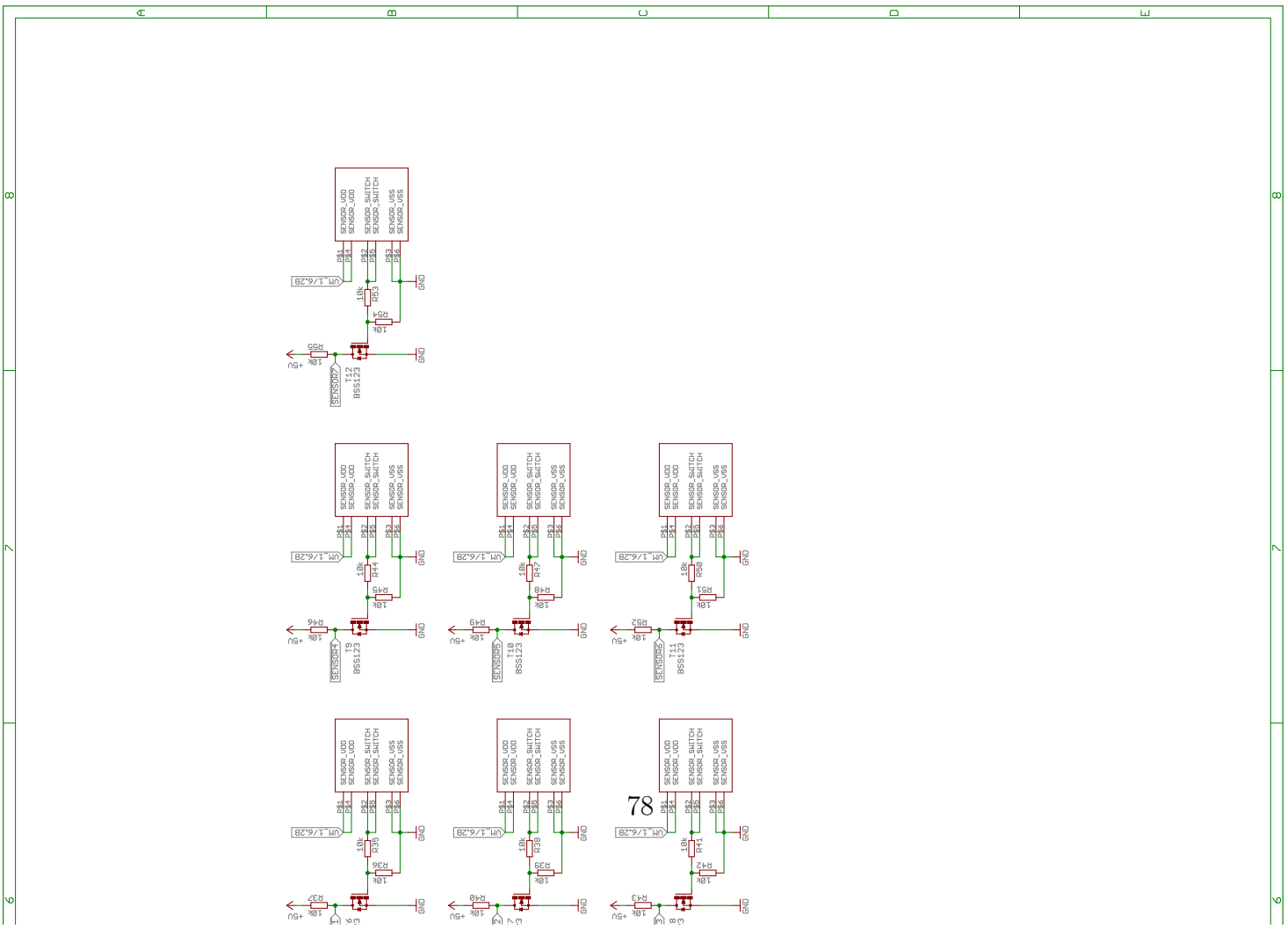
Code A.3: main.c

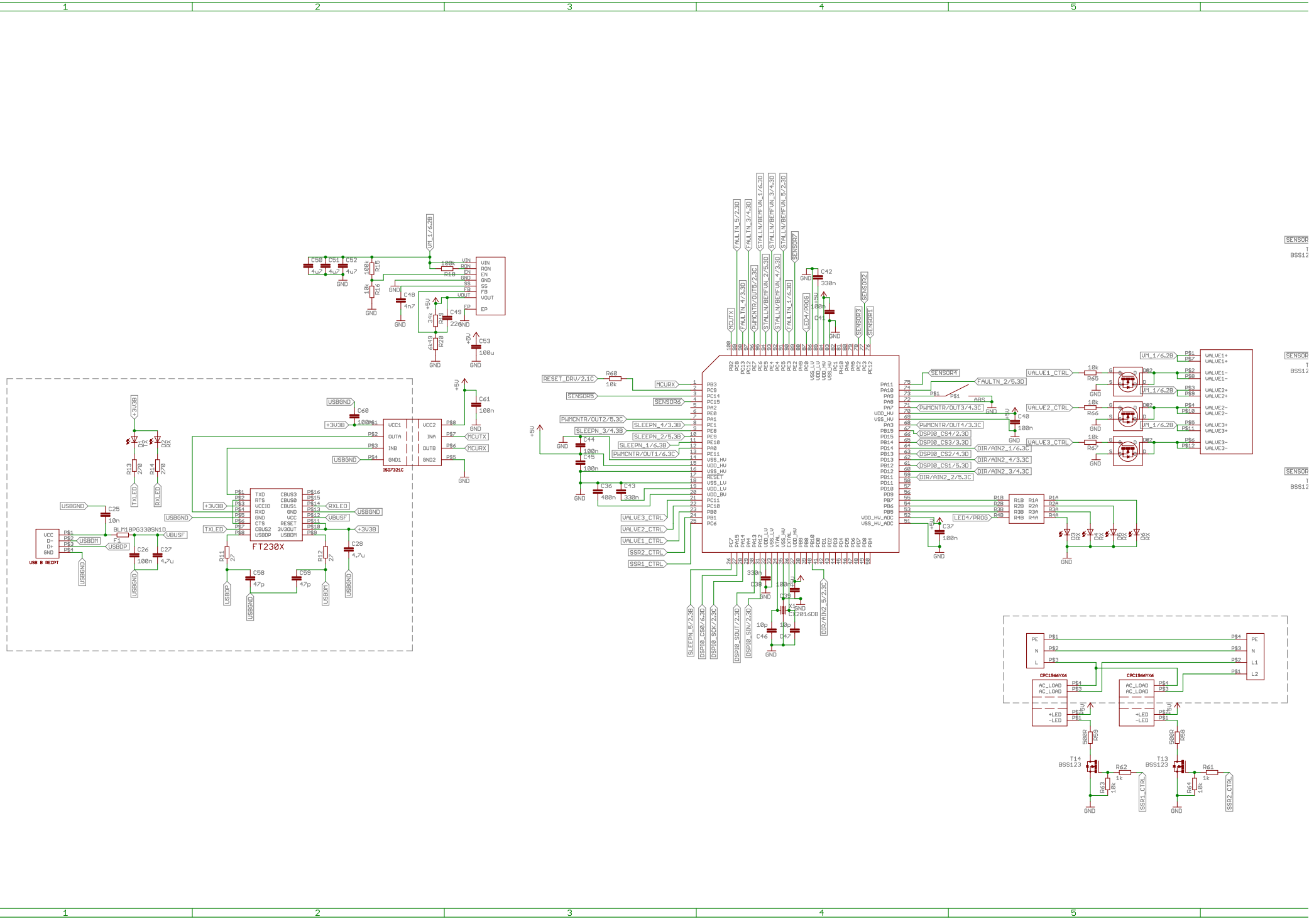
```
1   /*
2    * definitions.c
3    *
4    *  Created on: Aug 28, 2016
5    *      Author: Fabian Scheidl
6    */
7
8   #include "jdp.h"
9
10  #define SPI_DRV_STD_SETUP 0b0000110000010000
11  #define SPI_READ_CTRL     0b1000000000000000
12  #define BIT_RW 15
13  #define BIT_MOTOR_EN 0
14
15  #define BIT_CS0 16
16  #define BIT_CS1 17
17  #define BIT_CS2 18
18  #define BIT_CS3 19
19  #define BIT_CS4 20
20
21  unsigned char stepModeX = 0;
22  unsigned char stepModeY = 0;
23  unsigned char stepModeZ = 0;
24  unsigned char stepModeR = 0;
25
26  uint32_t SPI_Write_Data(uint32_t);
27
28  uint32_t SPI_Write_Data(uint32_t data){
29          uint8_t i;
30          uint32_t read_data = 0; // Data received on master SPI
31
32          DSPI_1.PUSHR.R = data;
33          while (DSPI_1.SR.B.RFDF != 1){
34                  asm("nop"); // Wait for Receive FIFO Drain Flag = 1
35          }
36          read_data = DSPI_1.POPR.R; // Read data received by slave SPI
37          DSPI_1.SR.R = 0x80020000; // Clear TCF, RDRF flags by writing 1 to them
38          for (i=0; i<200; i++){
39                  asm("nop"); // Wait
40          }
41          return read_data;
42  }
43
44  //Function to keep settings for motor as they are, but enable or disable it. MotorID is
        enumerated, enable is 0 or 1
45  uint8_t SPI_Enable_Motor(uint8_t enable, motorID id){
46          uint32_t datain;
```

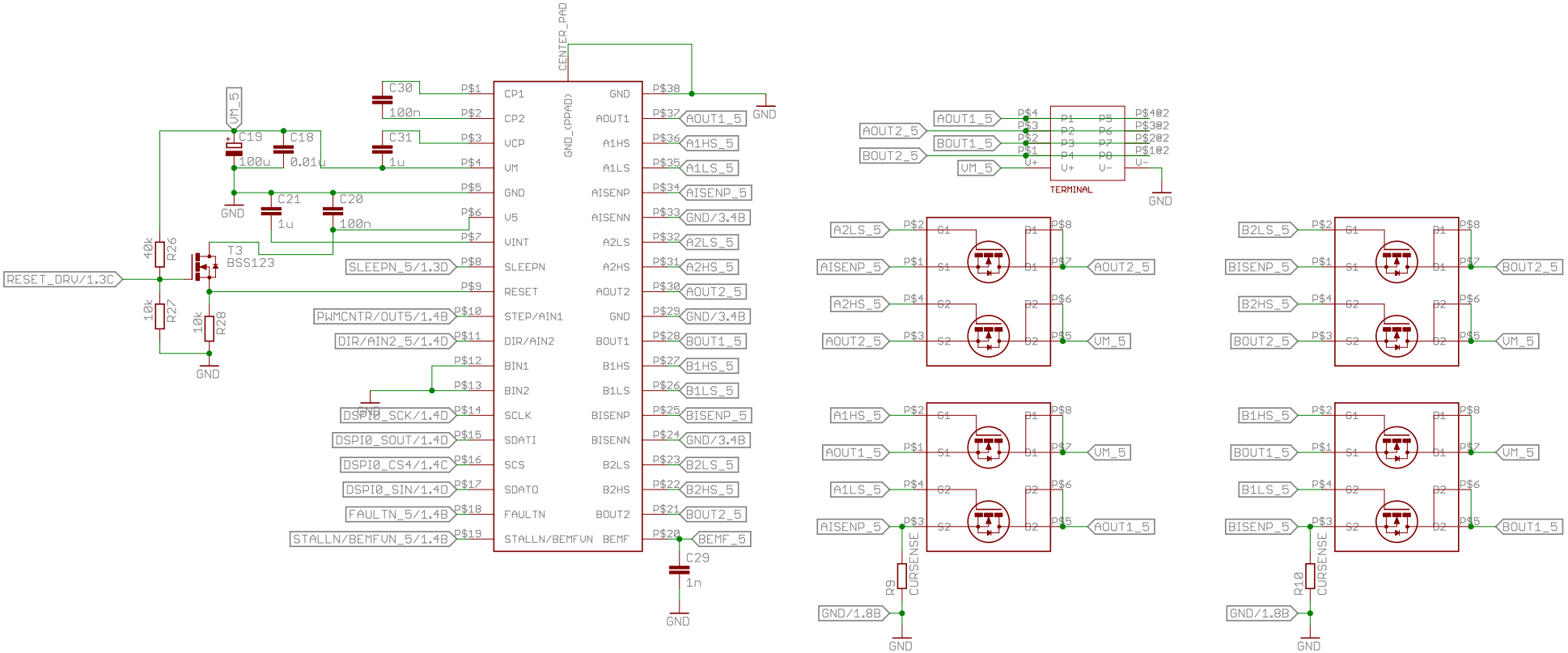
```
47
48          datain = SPI_WRITE_DATA(SPI_READ_CTRL | (1 << (16 + id)));
49
50          if((datain & (1 << BIT_MOTOR_EN)) != enable){
51                  //Change motor state, write new command
52                  SPI_WRITE_DATA(datain ^ (1 << BIT_MOTOR_EN) & (1 << BIT_RW));
53          }
54  }
55
56  void DRV_Init(void){
57          //Program all motor drivers and enable motors
58          stepModeX = (SPI_DRV_STD_SETUP >> 3) & 0b1111;
59          stepModeY = (SPI_DRV_STD_SETUP >> 3) & 0b1111;
60          stepModeZ = (SPI_DRV_STD_SETUP >> 3) & 0b1111;
61          stepModeR = (SPI_DRV_STD_SETUP >> 3) & 0b1111;
62
63          SPI_Write_Data(SPI_DRV_STD_SETUP | (1 << BIT_MOTOR_EN) & (0 << BIT_RW) | (1 << BIT_CS0
                ));
64          SPI_Write_Data(SPI_DRV_STD_SETUP | (1 << BIT_MOTOR_EN) & (0 << BIT_RW) | (1 << BIT_CS1
                ));
65          SPI_Write_Data(SPI_DRV_STD_SETUP | (1 << BIT_MOTOR_EN) & (0 << BIT_RW) | (1 << BIT_CS2
                ));
66          SPI_Write_Data(SPI_DRV_STD_SETUP | (1 << BIT_MOTOR_EN) & (0 << BIT_RW) | (1 << BIT_CS3
                ));
67          SPI_Write_Data(SPI_DRV_STD_SETUP | (1 << BIT_MOTOR_EN) & (0 << BIT_RW) | (1 << BIT_CS4
                ));
68  }
69
70  unsigned char CRC8(const unsigned char *data, unsigned char len) {
71          unsigned char crc = 0x00;
72          unsigned char tempI;
73          while (len--) {
74                  unsigned char extract = *data++;
75                  for (tempI = 8; tempI; tempI--) {
76                          unsigned char sum = (crc ^ extract) & 0x01;
77                          crc >>= 1;
78                          if (sum) {
79                                  crc ^= 0x8C;
80                          }
81                          extract >>= 1;
82                  }
83          }
84          return crc;
85  }
```

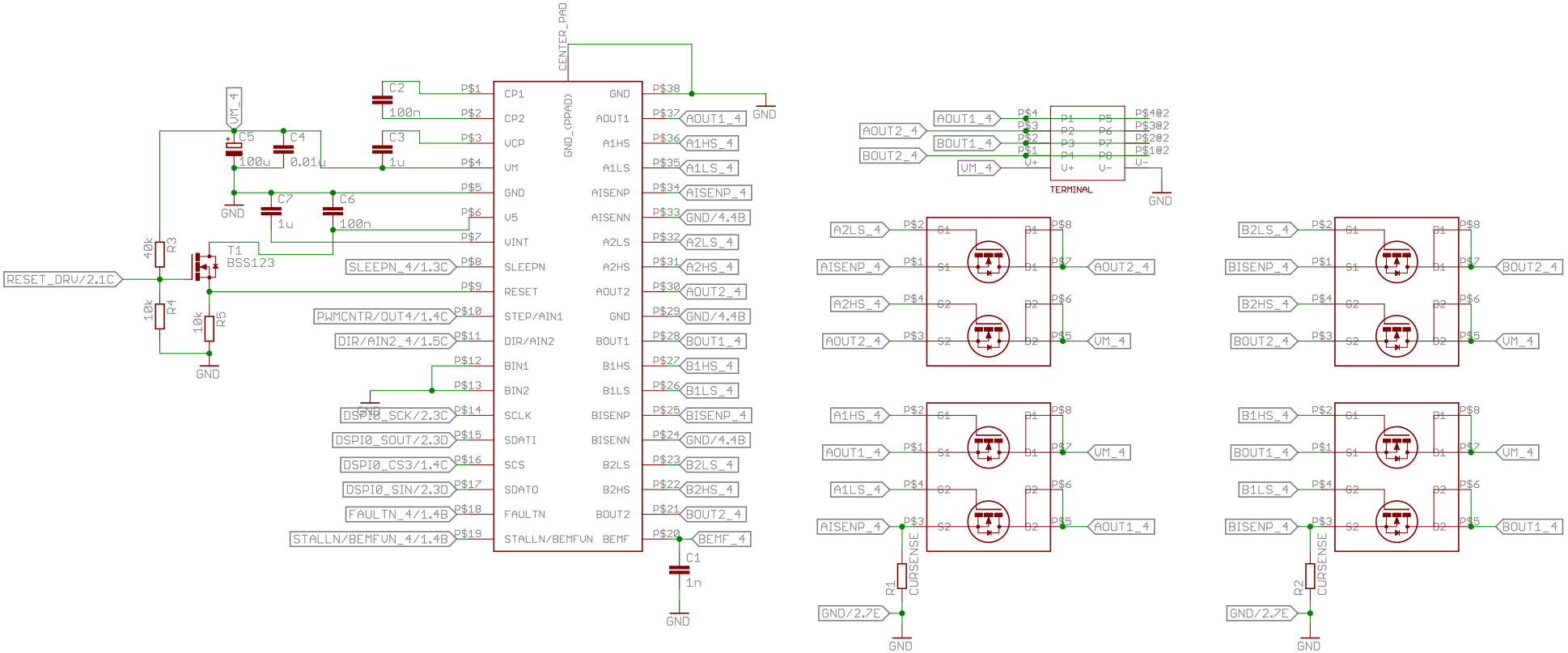
Anhang B

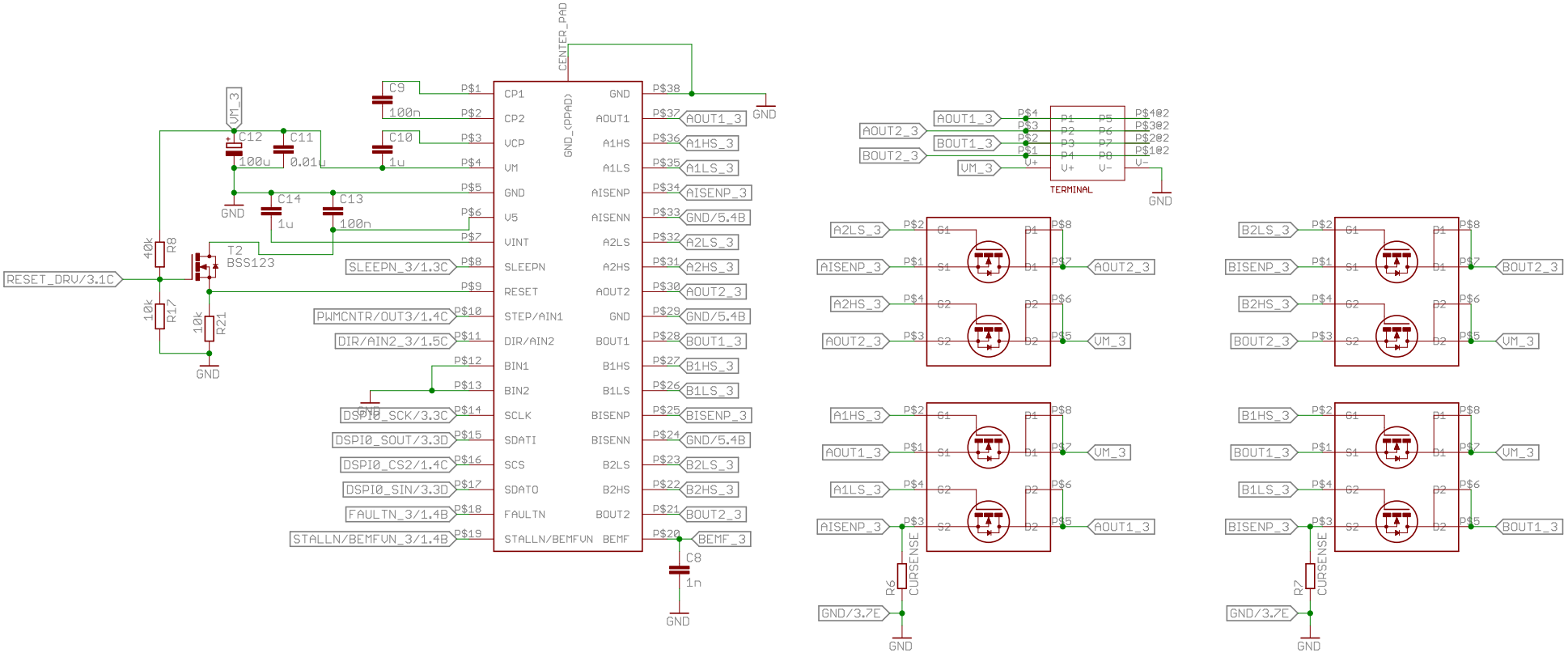
Schaltplan Steuerelektronik

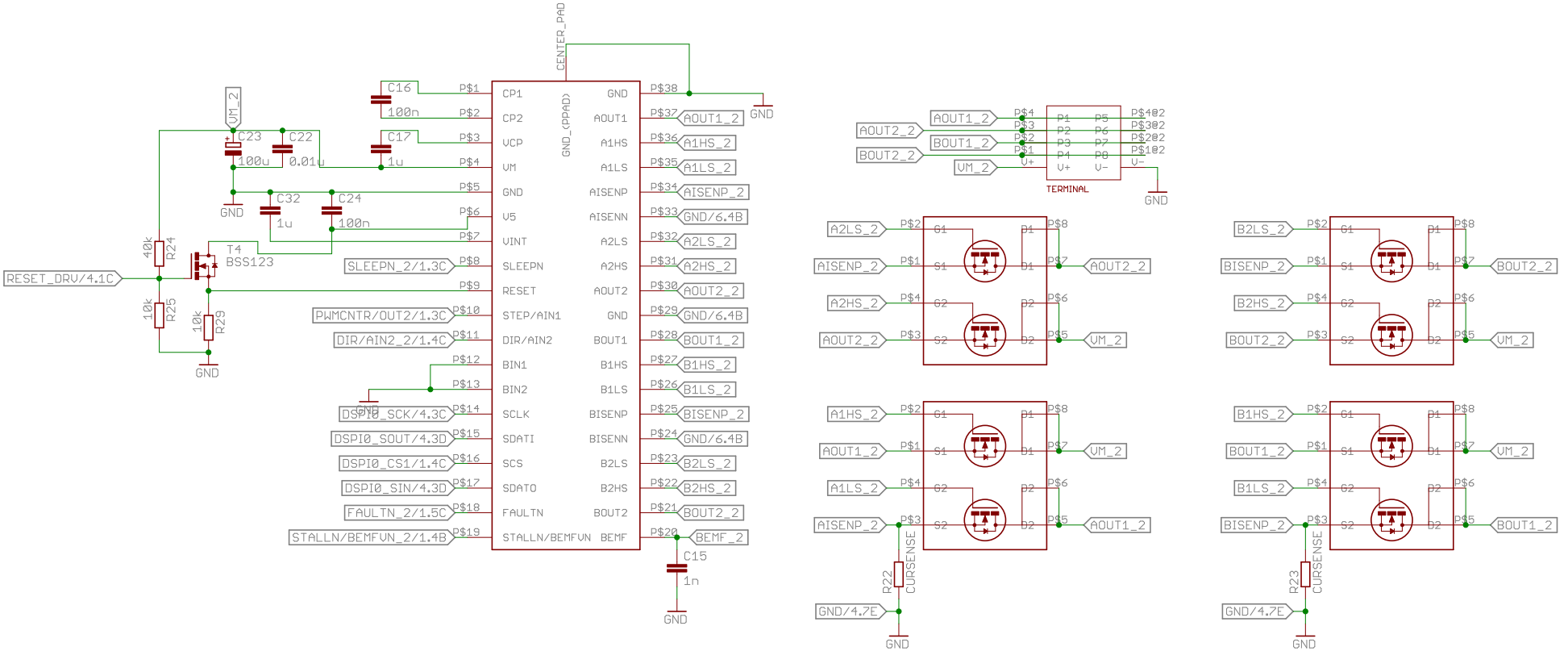


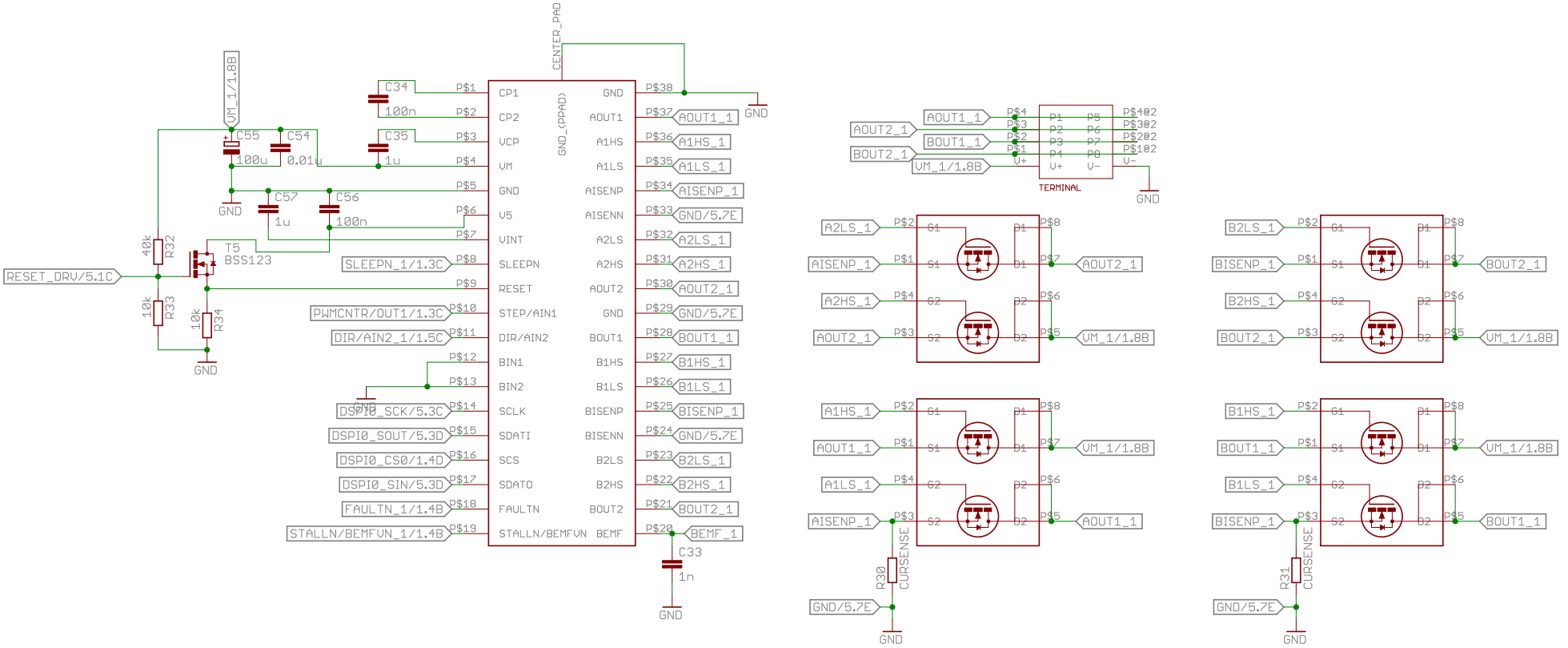

VM_1/6.2B
VIN
RON
EN
GND
SS
FB
VOUT
EP
+5V
+3V3B
USBGND
USBDM
USBDP
VBUSF
TXLED
RXLED
FT230X
ISO7321C
USB B RECPT
MCURX
MCUTX
SENSOR8
SENSOR6
SENSOR4
SENSOR3
SENSOR2
SENSOR1
RESET_DRV/2.1C
PWMCNTR/OUT2/5.3C
SLEEPN_4/3.3B
SLEEPN_2/5.3B
SLEEPN_1/6.3B
PWMCNTR/OUT1/6.3C
VALVE3_CTRL
VALVE2_CTRL
VALVE1_CTRL
SSR2_CTRL
SSR1_CTRL
FAULTN_5/2.3D
FAULTN_3/4.3D
FAULTN_4/3.3D
FAULTN_1/6.3D
FAULTN_2/5.3D
STALLN/BEMFUN_1/6.3D
STALLN/BEMFUN_3/4.3D
STALLN/BEMFUN_2/5.3D
STALLN/BEMFUN_4/3.3D
STALLN/BEMFUN_5/2.3D
PWMCNTR/OUT5/2.3C
PWMCNTR/OUT3/4.3C
PWMCNTR/OUT4/3.3C
LED4/PROG
DSPI0_CS4/2.3D
DSPI0_CS3/3.3D
DSPI0_CS2/4.3D
DSPI0_CS1/5.3D
DIR/AIN2_1/6.3C
DIR/AIN2_4/3.3C
DIR/AIN2_3/4.3C
DIR/AIN2_2/5.3C
DSPI0_CS0/6.3D
DSPI0_SCK/2.3C
DSPI0_SOUT/2.3D
DSPI0_SIN/2.3D
ABS
VDD_HV_ADC
VSS_HV_ADC
CPC1966YX6
AC_LOAD
+LED
-LED
BSS123


CENTER_PAD
GND_(PPAD)
CP1
CP2
VCP
VM
GND
V5
VINT
SLEEPN
RESET
STEP/AIN1
DIR/AIN2
BIN1
BIN2
SCLK
SDATI
SCS
SDATO
FAULTN
STALLN/BEMFVN
GND
AOUT1
A1HS
A1LS
AISENP
AISENN
A2LS
A2HS
AOUT2
GND
BOUT1
B1HS
B1LS
BISENP
BISENN
B2LS
B2HS
BOUT2
BEMF
C30
100n
C31
1u
C19
100u
C18
0.01u
C21
1u
C20
100n
C29
1n
VM_5
T3
BSS123
R26
40k
R27
10k
R28
10k
RESET_DRV/1.3C
SLEEPN_5/1.3D
PWMCNTR/OUT5/1.4B
DIR/AIN2_5/1.4D
DSPI0_SCK/1.4D
DSPI0_SOUT/1.4D
DSPI0_CS4/1.4C
DSPI0_SIN/1.4D
FAULTN_5/1.4B
STALLN/BEMFVN_5/1.4B
AOUT1_5
A1HS_5
A1LS_5
AISENP_5
GND/3.4B
A2LS_5
A2HS_5
AOUT2_5
GND/3.4B
BOUT1_5
B1HS_5
B1LS_5
BISENP_5
GND/3.4B
B2LS_5
B2HS_5
BOUT2_5
BEMF_5
AOUT2_5
BOUT1_5
VM_5
TERMINAL
A2LS_5
AISENP_5
A2HS_5
AOUT2_5
A1HS_5
AOUT1_5
A1LS_5
AISENP_5
R9
CURSENSE
GND/1.8B
B2LS_5
BISENP_5
B2HS_5
BOUT2_5
B1HS_5
BOUT1_5
B1LS_5
BISENP_5
R10
CURSENSE
GND/1.8B


CENTER_PAD
GND_(PPAD)
CP1
CP2
VCP
VM
GND
V5
VINT
SLEEPN
RESET
STEP/AIN1
DIR/AIN2
BIN1
BIN2
SCLK
SDATI
SCS
SDATO
FAULTN
STALLN/BEMFVN
GND
AOUT1
A1HS
A1LS
AISENP
AISENN
A2LS
A2HS
AOUT2
GND
BOUT1
B1HS
B1LS
BISENP
BISENN
B2LS
B2HS
BOUT2
BEMF
VM_4
C1
1n
C2
100n
C3
1u
C4
0.01u
C5
100u
C6
100n
C7
1u
R3
40k
R4
10k
R5
10k
T1
BSS123
RESET_DRV/2.1C
SLEEPN_4/1.3C
PWMCNTR/OUT4/1.4C
DIR/AIN2_4/1.5C
DSPI0_SCK/2.3C
DSPI0_SOUT/2.3D
DSPI0_CS3/1.4C
DSPI0_SIN/2.3D
FAULTN_4/1.4B
STALLN/BEMFVN_4/1.4B
AOUT1_4
A1HS_4
A1LS_4
AISENP_4
GND/4.4B
A2LS_4
A2HS_4
AOUT2_4
BOUT1_4
B1HS_4
B1LS_4
BISENP_4
GND/4.4B
B2LS_4
B2HS_4
BOUT2_4
BEMF_4
AOUT2_4
BOUT1_4
VM_4
TERMINAL
A2LS_4
AISENP_4
A2HS_4
AOUT2_4
B2LS_4
BISENP_4
B2HS_4
BOUT2_4
A1HS_4
AOUT1_4
A1LS_4
AISENP_4
B1HS_4
BOUT1_4
B1LS_4
BISENP_4
R1
CURSENSE
R2
CURSENSE
GND/2.7E


CENTER_PAD
GND_(PPAD)
CP1
CP2
VCP
VM
GND
V5
VINT
SLEEPN
RESET
STEP/AIN1
DIR/AIN2
BIN1
BIN2
SCLK
SDATI
SCS
SDATO
FAULTN
STALLN/BEMFVN
GND
AOUT1
A1HS
A1LS
AISENP
AISENN
A2LS
A2HS
AOUT2
GND
BOUT1
B1HS
B1LS
BISENP
BISENN
B2LS
B2HS
BOUT2
BEMF
C9
100n
C10
1u
C12
100u
C11
0.01u
C14
1u
C13
100n
C8
1n
VM_3
R8
40k
R17
10k
R21
10k
T2
BSS123
RESET_DRV/3.1C
SLEEPN_3/1.3C
PWMCNTR/OUT3/1.4C
DIR/AIN2_3/1.5C
DSPI0_SCK/3.3C
DSPI0_SOUT/3.3D
DSPI0_CS2/1.4C
DSPI0_SIN/3.3D
FAULTN_3/1.4B
STALLN/BEMFVN_3/1.4B
AOUT1_3
A1HS_3
A1LS_3
AISENP_3
GND/5.4B
A2LS_3
A2HS_3
AOUT2_3
GND/5.4B
BOUT1_3
B1HS_3
B1LS_3
BISENP_3
GND/5.4B
B2LS_3
B2HS_3
BOUT2_3
BEMF_3
AOUT2_3
BOUT1_3
BOUT2_3
VM_3
TERMINAL
A2LS_3
AISENP_3
A2HS_3
AOUT2_3
A1HS_3
AOUT1_3
A1LS_3
AISENP_3
R6
CURSENSE
GND/3.7E
B2LS_3
BISENP_3
B2HS_3
BOUT2_3
B1HS_3
BOUT1_3
B1LS_3
BISENP_3
R7
CURSENSE
GND/3.7E


CENTER_PAD
GND_(PPAD)
CP1
CP2
VCP
VM
GND
V5
VINT
SLEEPN
RESET
STEP/AIN1
DIR/AIN2
BIN1
BIN2
SCLK
SDATI
SCS
SDATO
FAULTN
STALLN/BEMFVN
GND
AOUT1
A1HS
A1LS
AISENP
AISENN
A2LS
A2HS
AOUT2
GND
BOUT1
B1HS
B1LS
BISENP
BISENN
B2LS
B2HS
BOUT2
BEMF
C16
100n
C17
1u
C23
100u
C22
0.01u
C32
1u
C24
100n
C15
1n
VM_2
T4
BSS123
R24
40k
R25
10k
R29
10k
RESET_DRV/4.1C
SLEEPN_2/1.3C
PWMCNTR/OUT2/1.3C
DIR/AIN2_2/1.4C
DSPI0_SCK/4.3C
DSPI0_SOUT/4.3D
DSPI0_CS1/1.4C
DSPI0_SIN/4.3D
FAULTN_2/1.5C
STALLN/BEMFVN_2/1.4B
AOUT1_2
A1HS_2
A1LS_2
AISENP_2
GND/6.4B
A2LS_2
A2HS_2
AOUT2_2
GND/6.4B
BOUT1_2
B1HS_2
B1LS_2
BISENP_2
GND/6.4B
B2LS_2
B2HS_2
BOUT2_2
BEMF_2
AOUT2_2
BOUT1_2
BOUT2_2
VM_2
TERMINAL
A2LS_2
AISENP_2
A2HS_2
AOUT2_2
A1HS_2
AOUT1_2
A1LS_2
AISENP_2
R22
CURSENSE
GND/4.7E
B2LS_2
BISENP_2
B2HS_2
BOUT2_2
B1HS_2
BOUT1_2
B1LS_2
BISENP_2
R23
CURSENSE
GND/4.7E


VM_1/1.8B
C55
100u
C54
0.01u
C34
100n
C35
1u
C57
1u
C56
100n
GND
T5
BSS123
40k
R32
10k
R33
10k
R34
RESET_DRV/5.1C
SLEEPN_1/1.3C
PWMCNTR/OUT1/1.3C
DIR/AIN2_1/1.5C
DSPI0_SCK/5.3C
DSPI0_SOUT/5.3D
DSPI0_CS0/1.4D
DSPI0_SIN/5.3D
FAULTN_1/1.4B
STALLN/BEMFVN_1/1.4B
CENTER_PAD
GND_(PPAD)
CP1
CP2
VCP
VM
GND
V5
VINT
SLEEPN
RESET
STEP/AIN1
DIR/AIN2
BIN1
BIN2
SCLK
SDATI
SCS
SDATO
FAULTN
STALLN/BEMFVN
AOUT1
A1HS
A1LS
AISENP
AISENN
A2LS
A2HS
AOUT2
BOUT1
B1HS
B1LS
BISENP
BISENN
B2LS
B2HS
BOUT2
BEMF
AOUT1_1
A1HS_1
A1LS_1
AISENP_1
GND/5.7E
A2LS_1
A2HS_1
AOUT2_1
BOUT1_1
B1HS_1
B1LS_1
BISENP_1
B2LS_1
B2HS_1
BOUT2_1
BEMF_1
C33
1n
AOUT2_1
BOUT1_1
BOUT2_1
VM_1/1.8B
TERMINAL
A2LS_1
AISENP_1
A2HS_1
AOUT2_1
A1HS_1
AOUT1_1
A1LS_1
AISENP_1
R30
CURSENSE
GND/5.7E
B2LS_1
BISENP_1
B2HS_1
BOUT2_1
B1HS_1
BOUT1_1
B1LS_1
BISENP_1
R31
CURSENSE
GND/5.7E

Anhang C

CAD-Renders

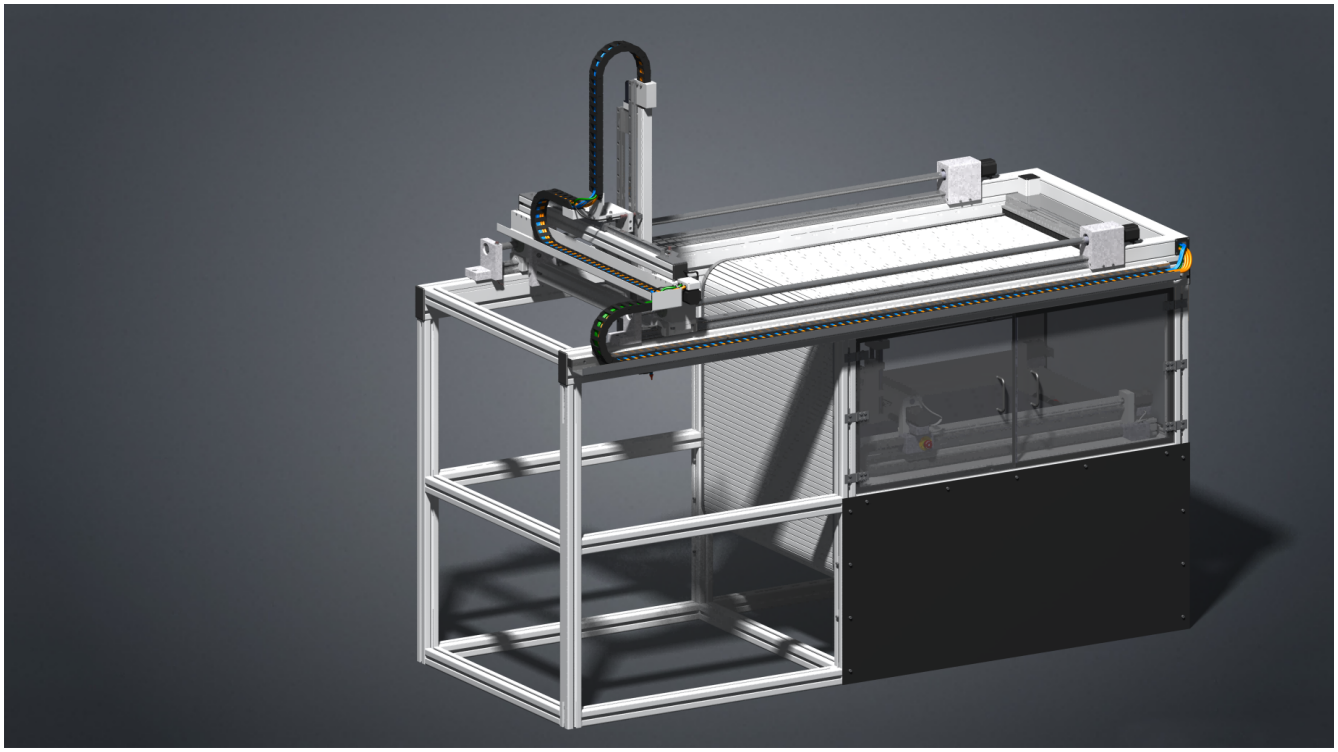


Abbildung C.1: Gesamte Konstruktion des Portalroboters bestehend aus Rahmen, Flächenportal, Z-Achse und Portalfräse inklusive deren Einhausung

Abbildung C.2: Einzelne X-Achse

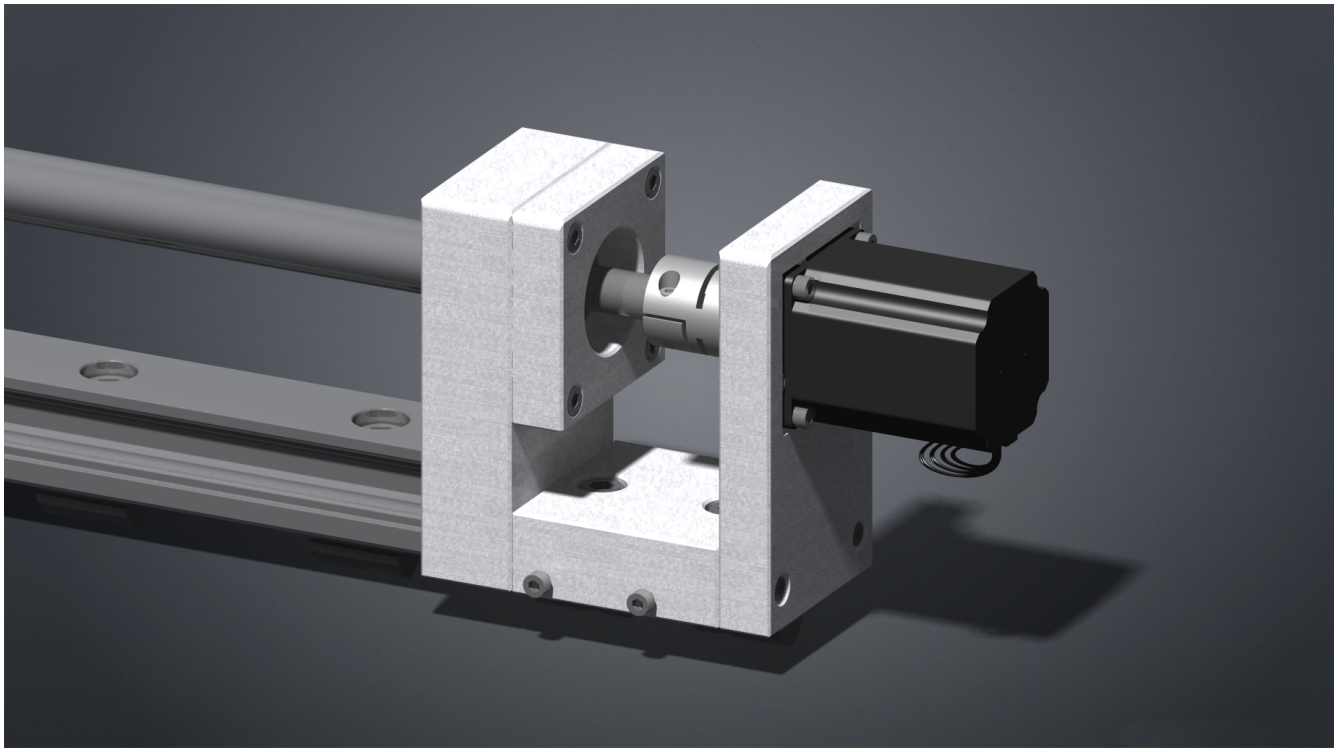


Abbildung C.3: Motoranbindung an der X-Achse

Abbildung C.4: Konstruktion der Y-Achse

Abbildung C.5: Weitere Ansicht der Y-Achse

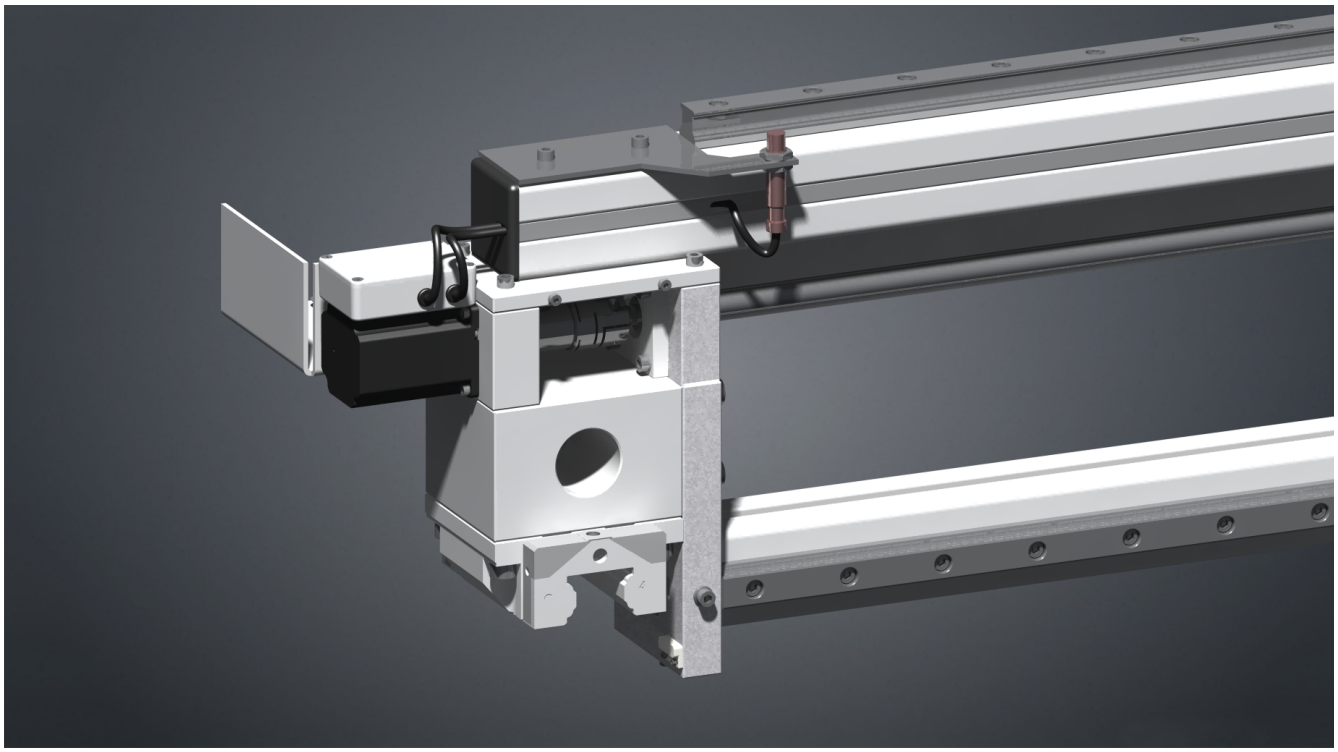


Abbildung C.6: Anbindung der Motoreinheit an der Y-Achse