HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

DIPLOMARBEIT

Herr B.Eng.
Robert Lublow

Konzeption und Entwicklung eines Versuchsaufbaus zur translatorischen und rotatorischen Bewegung von Versuchskörpern

Mittweida, 2023

Fakultät Ingenieurwissenschaften

DIPLOMARBEIT

Konzeption und Entwicklung eines Versuchsaufbaus zur translatorischen und rotatorischen Bewegung von Versuchskörpern

Autor:
Herr B.Eng. Robert Lublow

Studiengang:
Industrial Engineering

Seminargruppe:
IE21w1-D

Erstprüfer:
Prof. Dr.-Ing. Michael Kuhl

Zweitprüfer:
M.Sc. Kevin Blümel

Einreichung:
Mittweida, 31.03.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of Engineering

DIPLOMA THESIS

Conception and development of an experimental setup for the translational and rotational movement of test bodies

author:
Mr. B.Eng. Robert Lublow

course of studies:
Industrial Engineering

seminar group:
IE21-w1-D

first examiner:
Prof. Dr.-Ing. Michael Kuhl

second examiner:
M.Sc. Kevin Blümel

submission:
Mittweida, 31.03.2023

defence/evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Lublow, Robert:
Konzeption und Entwicklung eines Versuchsaufbaus zur translatorischen und rotatorischen Bewegung von Versuchskörpern. – 2023. – 64 Seiten, 37 Abbildungen. Mittweida, Hochschule Mittweida, Fakultät Ingenieurwissenschaften, Diplomarbeit, 2023

Referat:

Die vorliegende Diplomarbeit befasst sich mit der Konzeption und Entwicklung eines Versuchsaufbaus zur translatorischen und rotatorischen Bewegung von Versuchskörpern. Der Schwerpunkte liegen in der Analyse von Hardware-Komponenten, der Entwicklung eines Konzepts zur Umsetzung der Versuchsanlage, der mechanischen und elektrischen Errichtung der Anlage sowie der Entwicklung von Steuer- und Bediensoftware.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Wie dessen Name bereits andeutet, wurde die Entwicklung des Sensorteststands in Auftrag gegeben, um eine bestimmte Klasse von Sensoren testen zu können. Es handelt sich dabei um Sensoren, die Objekte im Raum detektieren können, wie zum Beispiel Radar- und Ultraschallsensoren oder Time of Flight Kameras. Mit Hilfe des Teststands sollen reproduzierbare Versuche durchgeführt werden, in denen das Sichtfeld der Sensoren analysiert sowie deren Erfassung von sich bewegenden Objekten getestet werden kann.

Bei dem vorliegenden Sensorteststand handelt es sich um die Weiterentwicklung einer Anlage, die zuvor im Rahmen einer Masterarbeit aufgebaut wurde. Zu Beginn dieses Projekts befand sich die Anlage in einem demontierten Zustand mit wenig verfügbarer Dokumentation. Die Herausforderung bestand darin, die vorhandenen Hardware-Komponenten zu untersuchen, den Sensorteststand wieder aufzubauen und darüber hinaus Verbesserungen zu implementieren. Die Umsetzung des Projekts erstreckte sich über mehrere ingenieurstechnische Disziplinen in den Bereichen des Maschinenbaus, der Elektrotechnik und der Informatik.

1.1 Aufgabenstellung

Ziel der Arbeit ist der prototypische Aufbau eines Versuchsstandes zur translatorischen und rotatorischen Bewegung von Versuchskörpern. Dabei soll ein plattenartiger Versuchskörper von mind. 50 x 50 cm mit einer Masse von max. 2 kg über einen Verfahrweg von mindestens 2 m gleichförmig bzw. gleichförmig beschleunigt bewegt werden. Die Ansteuerung der Motoren soll bzw. kann über ein µC-Board (z.B. Arduino) mit entsprechender Zusatzbeschaltung erfolgen. Die Einstellung der Parameter (Entfernungen, Winkel, Geschwindigkeitsprofil) soll über eine Software auf einem PC stattfinden und über die serielle Schnittstelle an das Board übertragen werden. Das System soll einen Triggereingang zum Ausgeben von Start und Stopp der Bewegung enthalten. Eine Visualisierung der Einstellungen erfolgt am PC bzw. am Controller über ein Display. Der Teststand soll transportfähig sein, das heißt er muss leicht demontierbar sein und die elektrische Verbindung zwischen den Teilsegmenten durch Steckkontakte realisiert werden.

Teilaufgaben der Diplomarbeit:

- Begriffsbestimmung und Definitionen (Glossar, Schlüsselworte)
- Einarbeitung in die Thematik „Ansteuerung elektrischer Motoren“:
  - → Wie kann eine translatorische und rotatorische Bewegung erzeugt werden?
  - → Welche Arten von Motoren können eingesetzt werden?

  - → Wie können diese Motoren angesteuert werden?
  - → Wie kann der Verfahrweg, die Geschwindigkeit bzw. der Drehwinkel und die Drehgeschwindigkeit gemessen und gesteuert, bzw. geregelt werden?
  - → Die vorhandene Motoren werden gestellt und final verwendet
- Erarbeitung eines Hardware-Systemkonzeptes mit den vorhandenen Hardware-Komponenten
- Erarbeitung eines Software-Ablaufplanes zur Ansteuerung und Auswertung der Aktoren mit einem µC, z.B. Arduino Mega-Mikrocontroller-Board, Eingabe und Übergabe der Parameter über einen PC
- Entwurf und Umsetzung der Schaltung
- Durchführung und Auswertung von Basisversuchen
- Bewertung, Darstellung und Dokumentation der Projektergebnisse

1.2 Zielstellung der Diplomarbeit

Das Diplomprojekt hat zum Ziel, einen Sensorteststand zu entwickeln, der den gestellten Anforderungen entspricht. Dafür wird ein Hard- und Software-Konzept erstellt und umgesetzt. Die geforderte Funktionalität des Sensorteststands ist in der Aufgabenstellung definiert. Die erforderlichen Hardware-Komponenten für den Bau der Anlage sind teilweise vorhanden. Weitere benötigte Komponenten werden ermittelt und beschafft. Der bereits vorhandene Teil des Teststands wird für das Projekt analysiert und reengineered. Im Hardware-Konzept wird berücksichtigt, dass die Anlage transportabel bleibt. Die erforderliche Software für den Steuerungs- und den Bedienteil wird für das Projekt von Grund auf neu erstellt.

2 Stand der Technik

Zu Beginn der Arbeit wird in diesem Kapitel der aktuelle Stand der Technik der für das Projekt verwendeten vorgefertigten Komponenten erläutert. Es wird dargelegt, welche Antriebsmöglichkeiten für die angestrebte Anwendung grundsätzlich bestehen und im Detail die Funktionsweise und Eigenschaften von Schrittmotoren beschrieben. Des Weiteren wird der Aufbau und das Designkonzept der verwendeten Mikrocontrollerplattform Arduino erläutert. Da für die Umsetzung der Bediensoftware für den PC die Wahl auf die Programmiersprache Python fiel, wird hier auf deren Besonderheiten eingegangen.

2.1 Antriebstechnik für mechatronische Systeme

Für Bewegungsaufgaben in einem mechatronischen System werden Aktoren eingesetzt. Es werden meist lagegeregelte Antriebssysteme verwendet, wofür drehzahl- und geschwindigkeitsregelbare Antriebe in Frage kommen. Diese Anforderungen können die meisten elektrischen und fluidischen Antriebe erfüllen (Roddeck 2019, S. 214). Abbildung 2-1 zeigt eine Übersicht von Antrieben, unterteilt nach ihrer jeweiligen Antriebsart. In der Antriebstechnik werden größtenteils sich drehende elektrische Maschinen eingesetzt, in denen elektrische in mechanische Energie umgesetzt wird. Sie werden nach der Art ihrer Spannungsversorgung unterschieden. Erfolgt die Einspeisung der elektrischen Energie in Form von Wechselstrom, liegt die Grundform der Drehfeldmaschine vor, welche als Synchron- und Asynchronmaschine ausgeführt wird. Bei einer Versorgung mit Gleichstrom spricht man von einem Gleichstrommotor. Wird eine permanenterregte Synchronmaschine mit einem Frequenzumrichter schrittweise angesteuert, so erhält man einen Schrittmotor (Roddeck 2019, S. 216f.).

Gleichstrommotoren wurden lange als Antriebe in Werkzeugmaschinen eingesetzt. Ihre Drehzahl kann in einem weiten Bereich geregelt werden, während die Motoren gleichzeitig eine hohe Drehsteifigkeit unter Last aufweisen (Brecher und Weck 2021, S. 27). Weitere Vorteile sind ein guter Gleichlauf und eine hohe Dynamik (Roddeck 2019, S. 218). Aufgrund seiner Versorgung mit Gleichstrom muss das Magnetfeld zum Antrieb des Motors mit Hilfe von Kohlebürsten kommutiert werden. Die Drehzahl kann entweder durch Änderung der Ankerspannung U_a oder des magnetischen Flusses φ erfolgen. Eine Erhöhung der Ankerspannung führt zu einer Erhöhung der Drehzahl, während eine Erhöhung des magnetischen Flusses zu einer Verringerung der Drehzahl führt (Brecher und Weck 2021, S. 28). Neben mechanisch kommutierten Gleichstrommotoren gibt es auch elektronisch kommutierte Bauformen. Hier befindet sich im Rotor ein Dauermagnet und das Drehfeld wird in der Ständerwicklung elektronisch geregelt. Diese bürstenlosen Gleichstrommoto-

ren sind wartungsarm und können nahezu bis zur Hälfte ihrer Nenndrehzahl mit dem maximalen Drehmoment belastet werden (Roddeck 2019, S. 224).

Drehfeldmaschinen werden mit Wechselstrom versorgt, weshalb die Rotoren wegen des resultierenden sich ändernden Magnetfelds auch als Dauermagneten ausgelegt werden können (Roddeck 2019, S. 230). Synchronmaschinen werden bei Werkzeugmaschinen und Industrierobotern ausschließlich als permanenterregte Synchronmaschinen eingesetzt (Brecher und Weck 2021, S. 32). Der Rotor bewegt sich abhängig von der Polpaarzahl synchron zum in der Ständerwicklung angelegten Drehfeld. Zur stufenlosen Einstellung der Drehzahl werden die Motoren mit einem Frequenzumrichter angesteuert. Eine Sonderbauform der Synchronmaschine stellt der Schrittmotor dar, welcher im anschließenden Abschnitt 2.2 ausführlicher erläutert wird. Bei Asynchronmaschinen besteht der Rotor aus einem sogenannten Kurzschlussläufer, in dem durch das von außen angelegte Wechselfeld ein Magnetfeld induziert wird. Auf Grund dieses Wirkungsprinzips dreht sich der Rotor immer in einer etwas niedrigeren Umdrehungsfrequenz als das antreibende Feld. Die Drehzahl kann bei der Asynchronmaschine ebenfalls durch die Verwendung eines Frequenzumrichters eingestellt werden. Die fortschreitende Entwicklung und die gesunkenen Preise der für die Antriebsregelung notwendigen Mikroelektronik haben dazu geführt, dass Drehfeldmotoren die bis vor einigen Jahren vorherrschenden Gleichstrommotoren im Bereich der Servoantriebe größtenteils abgelöst haben (Roddeck 2019, S. 236).

<table>
<tr><td rowspan="8">Antriebe</td><td rowspan="7">rotierend</td><td rowspan="4">elektromagnetisch</td><td>Synchronmotor</td></tr>
<tr><td>Asynchronmotor</td></tr>
<tr><td>Gleichstrom</td></tr>
<tr><td>Schrittmotor</td></tr>
<tr><td rowspan="3">hydraulisch</td><td>Flügelzellen-Motor</td></tr>
<tr><td>Axialkolben-Motor</td></tr>
<tr><td>Radialkolben-Motor</td></tr>
<tr><td rowspan="3">linear</td><td rowspan="2">elektromagnetisch</td><td>Asynchron-Linearmotor</td></tr>
<tr><td>Linear-Schrittmotor</td></tr>
<tr><td>hydraulisch (pneumatisch)</td><td>Hydraulikzylinder</td></tr>
</table>

Abbildung 2-1: Einteilung drehzahl- und geschwindigkeitssteuerbarer Antriebe (Roddeck 2019, S. 215)

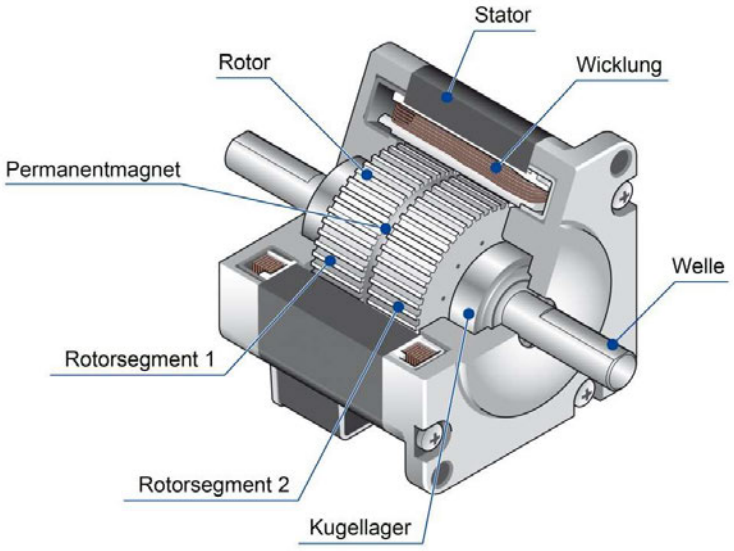
Neben den dominierenden rotierenden Drehfeldmaschinen können als Antriebe auch elektromotorische Linearsysteme oder fluidische Aktoren in mechatronischen Systemen eingesetzt werden. Linearmotoren entsprechen dem Wirkprinzip nach einem Drehstrommotor. Darüber hinaus gibt es auch Linearschrittmotoren (Roddeck 2019, S. 238 f.). Fluidische Aktoren nutzen gasförmige oder flüssige Medien zur Erzeugung von Kräften. Durch ihren einfachen Aufbau haben fluidische Antriebe bei der Erzeugung von Linear-

bewegungen Vorteile gegenüber elektrischen Antrieben. Jedoch ist eine ausreichende Positioniergenauigkeit nur mit einem erhöhtem Aufwand erreichbar (Roddeck 2019, S. 240).

2.2 Schrittmotoren und deren Ansteuerung

Ein Schrittmotor ist dem Funktionsprinzip nach eine Synchronmaschine. Der Rotor bewegt sich demnach synchron zu einem wechselnden Magnetfeld in den Statorwicklungen. Die schrittweise Fortbewegung des Rotors wird durch eine Zahnung in Längsrichtung erreicht (siehe Abbildung 2-2). Es existieren unterschiedliche Bauarten von Schrittmotoren, welche sich hauptsächlich durch die Ausführung des Rotors unterscheiden. Bei Reluktanzschrittmotoren ist der Rotor magnetisch unerregt. Die Bewegung des Läufers wird durch eine unterschiedliche magnetische Leitfähigkeit der einzelnen Zähne hervorgerufen. Permanenterregte Schrittmotoren besitzen einen Rotor, der aus einem Dauermagneten besteht. Diese Bauform entspricht der Vollpolsynchronmaschine mit Permanenterregung. Kombiniert man die Bauformen von Reluktanzschrittmotor und permanenterregtem Schrittmotor, erhält man einen Hybridmotor, welcher die Vorteile beider Funktionsprinzipien vereint (Brecher und Weck 2021, S. 51).

a)



b)

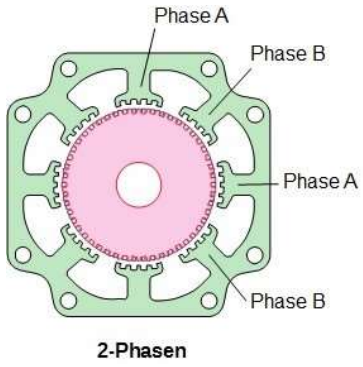


Abbildung 2-2: Aufbau eines Schrittmotors (SPS Magazin 2021): a) Schnittansicht eines Hybridmotors, b) Seitenansicht von Rotor und Ständer eines zweiphasigen Hybridmotors

Der Rotor des Hybridmotors besteht im Kern aus einem in Längsrichtung polarisierten Dauermagneten auf dessen Nord- und Südpol jeweils eine Zahnscheibe mit z.B. 50 Zähnen angebracht ist. Die beiden Zahnscheiben sind um einen halben Zahn zueinander versetzt, so dass sich eine Abfolge vieler kleiner magnetischer Nord- und Südpole entlang des Rotorumfangs ergibt. Ein Rotor mit zwei Zahnscheiben mit jeweils 50 Zähnen besitzt

demnach 50 Polpaare. Die Polschuhe der Ständerwicklungen sind ebenfalls gezahnt, allerdings mit einer geringeren Zähnezahl als der des Rotors. Durch eine gezielte Bestromung der Stränge im Ständer wird ein umlaufendes Magnetfeld erzeugt, welches eine Bewegung des Rotors hervorruft. Durchlaufen in einem Zwei-Phasen-Schrittmotor die Stränge elektrisch eine volle Umdrehung von 360°, so dreht sich das Statorfeld um 180°. Der Rotor bewegt sich dabei um vier Schritte vorwärts, was bei einer Polpaarzahl von 50 einem Winkel von 7,2° entspricht. Diese Betriebsart wird Vollschrittmodus genannt, wobei alle Phasen wechselseitig voll elektrisch beaufschlagt werden. Die Beschaltung der Motorwicklungen erfolgt durch einen Frequenzumrichter mit zwei Vollbrücken. Im Halbschrittmodus wird wechselseitig eine Phase abgeschaltet. Der Rotor führt dadurch Halbschritte aus und benötigt demnach doppelt so viele Schritte für eine Umdrehung wie im Vergleich zum Vollschrittmodus. Im sogenannt Mikroschrittmodus werden je nach Anzahl der Mikroschritte mehrere Zwischenschritte eingefügt, in denen die Stränge des Ständers durch den Frequenzumrichter nur teilweise bestromt werden. Der Strom in den Wicklungen wird dabei einem sinusförmigen Verlauf angenähert. Durch die Verwendung von Mikroschritten steigt das Auflösungsvermögen der Schritte pro Umdrehung. Dies hat zur Folge, dass sich der Motor gleichmäßiger und ruhiger bewegt, wodurch störende Schwingungen vermieden werden. Allerdings sinkt im Mikroschrittbetrieb das Haltemoment erheblich, weshalb die Positioniergenauigkeit negativ beeinflusst wird (Faulhaber 2020).

Die auf in ihrem Funktionsprinzip begründete Schwingungsneigung von Schrittmotoren ist im Allgemeinen als nachteilig anzusehen. Sie bewirkt eine starke Geräuschentwicklung und erhöht den Verschleiß der angeschlossenen mechanischen Komponenten. Ein weiterer Nachteil ist die Möglichkeit des Außerschrittfallens bei zu hohen Lastmomenten. Dem gegenüber stehen Vorteile, die Schrittmotoren im Vergleich zu anderen Antriebsformen aufweisen. Sie können schnell und schrittgenau positioniert werden, ohne dass dafür eine aufwendige Ansteuerung nötig ist. Ihr einfacher Aufbau führt zu einer hohen Robustheit und Wartungsfreiheit (Brecher und Weck 2021, S. 51).

2.3 Die Mikrocontrollerplattform Arduino

Arduino ist eine Open-Source Plattform, die zum ersten Mal im Jahr 2005 vorgestellt wurde. Das Projekt wurde in der Stadt Ivrea, Italien entwickelt, die einst von einem König Arduin regiert wurde. Der Name Arduino steht im Italienischen für „starker Freund“ (Wheat 2011, S. 1f.).

Die Arduino Plattform umfasst die Hardware, bestehend aus einem Mikrocontroller Board mit Peripherie, eine Softwareumgebung sowie ein Entwicklungsteam und einer großen Community, die das Projekt stetig weiterentwickeln. Die Mikrocontroller Boards sind in verschiedenen Varianten erhältlich und eine häufig genutzte Variante ist der Arduino Uno. Alle Boards beruhen auf dem gleichen Grunddesign, wobei sie sich in Größe und Leistungsfähigkeit unterscheiden. Das Kernstück des Arduino Uno ist ein Atmel ATmega328 Mikrocontroller. Weitere Komponenten auf dem Board sind eine USB-Schnittstelle, ein

Netzteil und Anschlussbuchsen für I/O-Konnektivität. Die Spannungsversorgung erfolgt über eine Gleichspannungsquelle mit 7 – 12 V. Die Spannung wird intern auf 5 V geregelt, um den Mikrocontroller zu versorgen. Durch die USB-Konnektivität kann die Spannungsversorgung auch von einem Host-PC oder einem USB-Ladegerät aus erfolgen. Dieser Spannungspfad ist mit 500 mA abgesichert. Ein interner Schalter wählt automatisch die Spannungsquelle mit der höheren Spannung. Zusätzlich enthält das Board noch einen 3,3 V Spannungsregler. Die beiden Betriebsspannungen können auch für externe Schaltungselemente vom Board abgegriffen werden (Wheat 2011, S. 75 ff.). Für die serielle Kommunikation zu einem Host-PC enthält das Board einen zusätzlichen Atmel ATmega8U2 Mikrocontroller, in welchem die USB-Funktionalität integriert ist. Weitere nutzbare Schnittstellen sind USART, I²C und ISP (Wheat 2011, S. 81). Die Arduino Boards verfügen über Buchsenleisten, über welche auf einfache Weise Peripheriegeräte angeschlossen werden können. Die elektrischen Kenngrößen sind durch den Mikrocontroller vorgegeben und betragen 5 V, bzw. 40 mA. Eine Besonderheit der Arduino Plattform ist die standardisierte Anordnung der Buchsenleisten, auf welche Erweiterungsplatinen aufgesteckt werden können. Die sogenannten Shields können z.B. Motortreiber, Ethernet-Controller oder Lochrasterplatinen zur freien Bestückung sein. Eine Ausbaustufe des Arduino Uno ist der Arduino Mega 2560 (Abbildung 2-3). Er enthält einen ATmega2560 als Mikrocontroller und verfügt dadurch über mehr I/O-Pins und über zusätzliche serielle Kanäle.

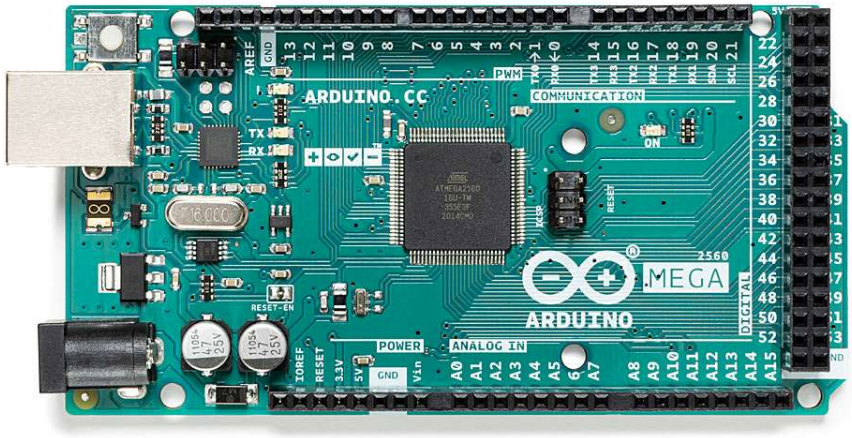


Abbildung 2-3: Arduino Mega 2560 Rev.3 (Arduino Official Store 2023)

Die Arduino Plattform verfügt über eine eigene Programmiersprache, die auf der Basis von C entwickelt wurde. Als Entwicklungsumgebung dient die sogenannte Arduino IDE, welche die Kommunikation zum Arduino gewährleistet und in welcher der Programmcode erstellt wird. Jede Syntax enthält die zwei Hauptfunktionen setup() und loop(). Setup() wird genau einmal beim Start oder Reset des Mikrocontrollers aufgerufen und enthält Elemente zum Konfigurieren der Hardware. Die Funktion loop() enthält den eigentlichen Programmcode und wird in einer Endlosschleife aufgerufen. Um die Programmierung ei-

nes Arduinos möglichst leicht zu gestalten, erfolgt die Kompilierung und der Upload des Codes über einen einzigen Tastendruck. Um dies zu ermöglichen, enthält der Mikrocontroller einen vorinstallierten Bootloader. Das Debugging erfolgt durch Testen der Funktionalität der Hardwareapplikation und durch die Ausgabe von Textnachrichten über die serielle Schnittstelle. Hierfür stellt die Arduino IDE einen Serial Monitor bereit (Wheat 2011, S. 25 ff.).

2.4 Die Programmiersprache Python

Python ist eine interpretierte, interaktive und objektorientierte Programmiersprache. Entwickelt wurde sie von Guido van Rossum und 1991 erstmalig vorgestellt. Zu ihrem Funktionsumfang gehören integrierbare Module, ein Exception Handler, dynamische Typisierung und High-Level Datentypen. Python verfügt über viele Schnittstellen für System Calls und Bibliotheken, was die Einbettung in ein Betriebssystem erleichtert. Die Programmiersprache ist portabel, so dass sie über verschiedene Betriebssysteme, wie Windows, Linux oder MacOS hinweg genutzt werden kann. Im Allgemeinen wird Python als sehr einsteigerfreundlich eingestuft (Python Software Foundation 2023).

Dass Python plattformübergreifend ausführbar ist, liegt daran, dass der Quellcode interpretiert wird. Dies beschreibt die Art und Weise, wie der Quellcode in Maschinencode übersetzt wird. Viele Programmiersprachen nutzen Compiler, das heißt, der gesamte Quellcode wird plattformabhängig in Maschinencode kompiliert. Im Gegensatz dazu übersetzt ein Interpreter den Quellcode zeilenweise während dessen Ausführung. Der Quellcode kann von allen Betriebssystemen gleichermaßen gelesen werden. Python ist nicht auf ein bestimmtes Programmier-Paradigma beschränkt. Es unterstützt sowohl den objektorientierten, den aspektorientierten, den strukturierten, als auch den funktionalen Programmierstil (Steyer 2018, S. 2 ff.).

3 Konzept der Umsetzung der Aufgabenstellung

Das Projekt ist so angelegt, dass es auf einen bereits vorhandenen Prüfstand aufbaut, der im Rahmen einer Masterarbeit entwickelt wurde. Der Prüfstand wurde im Anschluss an die Arbeit demontiert und ließ sich wegen unzureichender technischer Dokumentation in diesem Zustand nicht mehr betreiben. Ziel ist es, den Prüfstand zu rekonstruieren und ihn in seiner Funktionalität zu erweitern. Da der Prüfstand trotz seiner großen Abmaße transportierbar sein soll, muss die Konstruktion so ausgeführt werden, dass sie leicht auf- abbaubar ist. Das folgende Kapitel stellt die Überlegungen dar, welche in den Plan für die Umsetzung des Projekts eingeflossen sind.

3.1 Hardware-Konzept

Der grundlegende Aufbau des Sensorteststands wird aus dem vorausgegangenen Projekt übernommen. Die Konstruktion ist in Abbildung 3-1 schematisch dargestellt. Das Grundgerüst trägt eine rotierbare Sensorplattform und einen Schlitten für Prüfkörper, welcher rotierbar und axial verschiebbar ist. Die elektromechanischen Antriebe werden durch Schrittmotoren bereitgestellt. Im Folgenden wird das erarbeitete Hardware-Konzept für die Umsetzung der Aufgabenstellung erläutert. Dafür wird das Konzept in die Teilbereiche Mechanik, Elektromechanik, Elektronik und Elektroinstallation untergliedert.

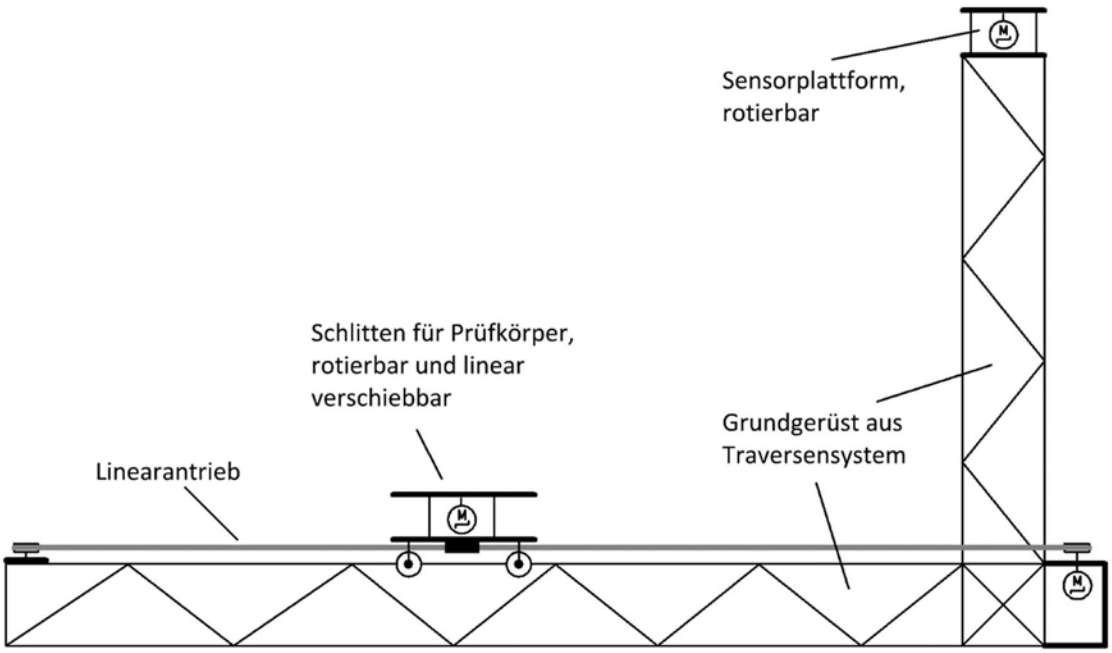


Abbildung 3-1: Schematischer Aufbau des Sensorteststands

3.1.1 Mechanik

Das Grundgerüst des Sensorteststands bilden Elemente des Traversensystems F43 des Herstellers Global Truss. Es setzt sich zusammen aus einer 3 m langen horizontalen Traverse, die als Schiene für den Schlitten dient, und einer 1 m und einer 0,5 m langen vertikalen Traverse, die einen Turm für die Sensorplattform bildet. Ergänzt wird der Aufbau durch diverse Verbindungselemente und Anbauteile aus der Systemserie. Die Trassenteile sind demontierbar, um einen Transport zu erleichtern. Die Sensorplattform besteht aus einer Holzkonstruktion, welche durch einen Schrittmotor und eine Getriebestufe mit einem Drehkranz rotatorisch bewegt werden kann. Die für den Prüfstand verwendeten Schrittmoren werden im Abschnitt 3.1.2 näher erläutert. Der Schlitten für die Prüfkörper ist ebenfalls aus Holz gefertigt. Er besitzt einen drehbaren Aufbau, der ähnlich wie die Sensorplattform konstruiert ist. Der Antrieb erfolgt durch den gleichen Schrittmotortyp und die gleiche Getriebestufe, wie in der Sensorplattform. Für die translatorische Bewegung ist der Schlitten rollbar auf der Traverse gelagert. Die Bewegung wird durch einen weiteren Schrittmotor erzeugt, welcher den Schlitten über einen Zahnriemen antreibt.

Die vorhandene Konstruktion weist Schwachstellen auf, die überarbeitet werden müssen. Erste Versuche haben gezeigt, dass die Verankerungen der Motoren zum Teil den auftretenden Kräften nicht standhalten. Die Sensorplattform ist nicht exakt zentral zum darunterliegenden Turm ausgerichtet, was präzise Messungen mit dem Prüfstand möglicherweise erschwert. Neben konstruktiven Verbesserungen wird der Aufbau durch weitere Anbauten ergänzt. Für eine zusätzliche Positionserfassung werden an den Bewegungselementen Schaltfahnen für Gabellichtschranken angebracht (siehe Abschnitt 3.1.3). Entlang der horizontalen Trasse wird eine Schleppkette angebracht, welche die elektrischen Leitungen für die Spannungsversorgung und Signalübertragung an den Drehantrieb des Schlittens führen wird (siehe Abschnitt 3.1.4). An den beiden Endpunkten des Verfahrwegs des Schlittens werden auf der Trasse mechanische Festanschläge angebracht, welche im Fehlerfall ein sicheres Stoppen des Schlittens gewährleisten.

3.1.2 Elektromechanik

Die drei Bewegungsachsen des Sensorteststands werden durch Schrittmotoren angetrieben. Diese wurden bereits im vorangegangenen Projekt dimensioniert und implementiert (Wang 2021, S. 18f) und werden für dieses Projekt übernommen. Es handelt sich bei den Motoren um 2-Phasen Hybrid-Schrittmotoren des Herstellers ACT-Motor. Für die beiden rotatorischen Achsen kommt jeweils ein Schrittmotor vom Typ 23SSM6440-EC1000 mit der Baugröße Nema 23 und einem Haltemoment von 1,1 Nm zum Einsatz. Der Linearantrieb enthält einen leistungsstärkeren Schrittmotor vom Typ 34SSM8460-EC1000 mit der Baugröße Nema 34 und einem Haltemoment von 5 Nm. Die Kenngrößen der verwendeten Motoren sind in Tabelle 1 aufgeführt. Für die Auslegung des Prüfstands sind insbesondere die mechanischen Kennwerte wie Schrittwinkel, Haltemoment, Nenndrehzahl und Trägheitsmoment von Bedeutung. Die elektrischen Kennwerte, wie die Stromaufnahme,

finden weniger Berücksichtigung, da die Motoren mit vorkonfektionierten Kabeln ausgeliefert und durch Leistungstreiber angetrieben werden, welche auf die Motoren zugeschnitten sind (siehe Abschnitt 3.1.3). Die Motoren sind mit Encodern ausgestattet, deren technische Eigenschaften allerdings in den Datenblättern nicht näher benannt werden.

	23SSM6440-EC1000	34SSM8460-EC1000
Baugröße	Nema 23	Nema 34
Schrittwinkel	1,8 ±5% °/Schritt	1,8 ±5% °/Schritt
Nennspannung	1,6 V	1,56 V
Nennstrom	4 A/Phase	6 A/Phase
Widerstand	4 ±10% Ω/Phase	0,26 ±10% Ω/Phase
Induktivität	1,2 ±20% mH/Phase	2 ±20% mH/Phase
Haltemoment	1,1 Nm	5 Nm
Rastmoment	4 Ncm	6,5 Ncm
Trägheitsmoment	300 gcm²	1400 gcm²
Nenndrehzahl	1000 min^{-1}	1000 min^{-1}

Tabelle 3-1: Kennwerte der verwendeten Schrittmotoren (ACT-Motor 2017c, 2017d)

3.1.3 Elektronik

Für die Ansteuerung der Schrittmotoren und die Bedienung des Prüfstandes werden Elektronikkomponenten benötigt. Abbildung 3-2 zeigt das Konzept für die Struktur der elektronischen Komponenten als schematischen Aufbau.

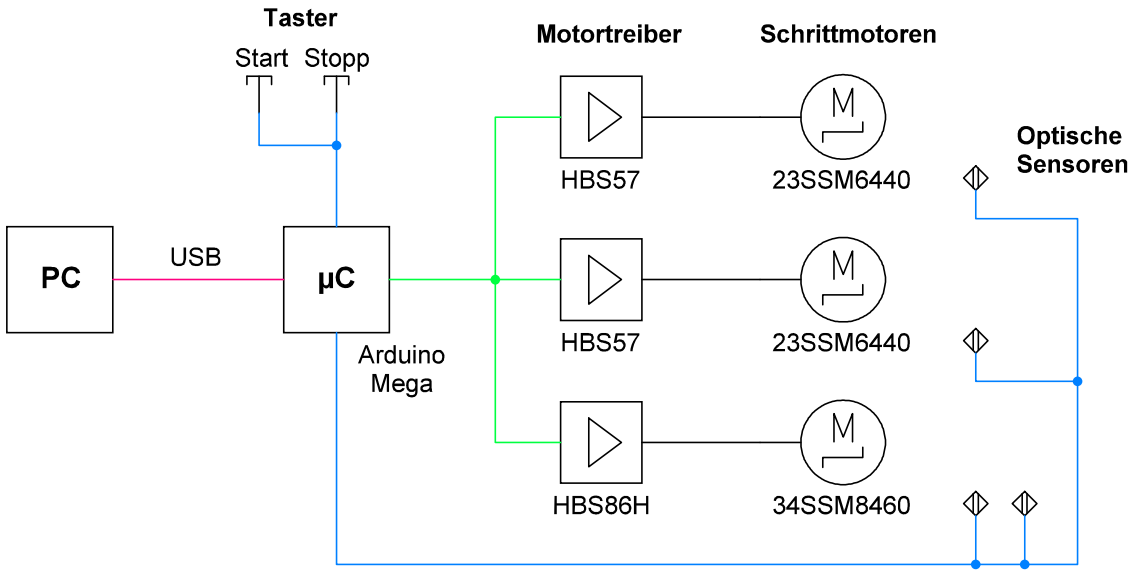


Abbildung 3-2: Schematische Darstellung des Elektronik-Konzepts

Die Motortreiber für die Schrittmotoren können aus dem vorangegangenen Projekt übernommen werden. Sie stammen von ACT-Motor, demselben Hersteller wie für die Motoren. Die kleineren Nema 23 werden jeweils von einem HBS57 angetrieben, mit einem maximalen Ausgangsstrom von 4,5 A (ACT Motor 2017a). Die Treiber benötigen für die Spannungsversorgung eine Gleichspannungsquelle mit einer wählbaren Spannung von 24 – 60 V. Der größere Nema 34 Motor wird von einem HBS86H angesteuert, welcher bis zu 8 A Ausgangsstrom liefern kann (ACT Motor 2017b). Für die Spannungsversorgung wird eine Spannungsquelle benötigt, die 24 – 80 VAC oder 30 – 100 VDC liefert. Abgesehen von den, mit der unterschiedlichen bereitgestellten Ausgangsleistung einhergehenden verschiedenen Abmessungen, unterscheiden sich die Geräte nicht voneinander. Es gibt Ausgangsklemmen für die zwei Phasen des Schrittmotors und Eingangsklemmen für die zwei Phasen des Encoders. Über Optokoppler sind die Steuereingänge „Pulse“, „Direction“ und „Enable“ realisiert. Hierüber werden vom Controller Richtungs- und Schrittsignale gesendet. Über die Frequenz eines Rechtecksignals werden am Eingang „Pulse“ die einzelnen Schritte des Motors vorgegeben. Die Motortreiber können im Mikroschrittbetrieb von 400 – 51 200 Schritten pro Umdrehung arbeiten. Für den Prüfstand wird die niedrigste Schrittzahl von 400 gewählt, da eine höhere Auflösung nicht erforderlich ist und die damit einhergehenden Nachteile daher nicht in Kauf genommen werden müssen (siehe Abschnitt 2.2).

Die zentrale Steuerung des Prüfstandes erfolgt durch einen Mikrocontroller. Dafür wird aus dem vorangegangenen Projekt das Mikrocontrollerboard Arduino Mega Rev3 übernommen, da dieses für die Aufgabe gut geeignet ist (siehe Abschnitt 3.2). Das Board besteht im Kern aus einem ATmega2560 des Herstellers Microchip. Es handelt sich hierbei um einen Mikrocontroller mit einer 8-Bit RISC Architektur, die mit 16 MHz getaktet wird. Der interne Speicher besteht aus 4 KB EEPROM, 256 KB Flash und 8 KB SRAM (Microchip Technology Inc 2005). Das Board verfügt über weitere Komponenten, wie einem Step-Down Converter für die Spannungsversorgung, einem USB Controller und Pfostenstecker für eine einfache Kontaktierung der I/O-Pins. Die Hauptaufgabe des Mikrocontrollers ist es, die korrekten Steuerimpulse an die Motortreiber zu senden, damit die Motoren die gewünschten Bewegungen ausführen. Dafür liefert er über drei Ausgänge jeweils ein Rechtecksignal mit einer berechneten Frequenz an die einzelnen Motortreiber. An seinen Eingängen verarbeitet der Controller die Signale von vier Positionssensoren und zwei Tastern. Die Positionssensoren bestehen aus Gabellichtschranken und detektieren die Endlagen der Achsen des Prüfstands. Die beiden rotatorischen Achsen besitzen jeweils einen Sensor, der den Nullpunkt markiert für die definierte Positionierung der Drehscheiben. Die translatorische Achse erhält jeweils einen Sensor zur Erfassung der Endlagen. Die Verwendung der Signale soll verhindern, dass der Schlitten an seine mechanischen Endlagen anstößt. Einer der beiden Sensoren wird gleichzeitig als definierter Nullpunkt zur Ausrichtung des Schlittens verwendet. Die Taster werden jeweils für ein Start- und Stoppsignal für die Bewegung des Schlittens verwendet. Eine Beleuchtung der Taster soll Anzeigen, zu welchem Zeitpunkt die Taster aktiv sind.

Die Steuerung des Prüfstandes durch einen Bediener erfolgt im Wesentlichen von einem PC aus. Dieser verfügt über eine grafische Benutzeroberfläche und wird per USB mit dem Controller verbunden (siehe Abschnitt 3.2). Der PC als Hardware-Komponente ist nicht integraler Bestandteil des Projekts, was bedeutet, dass ein beliebiges Gerät verwendbar sein sollte. Da es vorgesehen ist, die Benutzeroberfläche mit Python als Windows-Anwendung zu realisieren, muss der PC mindestens über das Betriebssystem Windows 10 verfügen.

3.1.4 Elektrik

Im vorgegangenen Projekt wurde eine feste Installation der erforderlichen elektrischen Verbindungen nicht umgesetzt. Der Prüfstand soll nun mit einer robusten Elektroinstallation versehen werden, die dem Einsatz über mehrere Jahre standhalten kann. Es gilt dabei zu berücksichtigen, dass die Verbindung zum Schlitten beweglich und somit zusätzlichen mechanischen Belastungen ausgesetzt ist. Da der Prüfstand demontierbar sein soll, müssen an definierten Trennpunkten Steckverbindungen zwischen den elektrischen Leitungen vorgesehen werden. Am Turm wird eine Hauptverteilung in Form eines Schaltkastens angebracht. Dieser enthält die Motorsteuerung, Bedienelemente und Anschlüsse für die Spannungsversorgung und die Kommunikation zum PC. Abbildung 3-3 zeigt das Konzept für die Elektrik als schematische Übersicht.

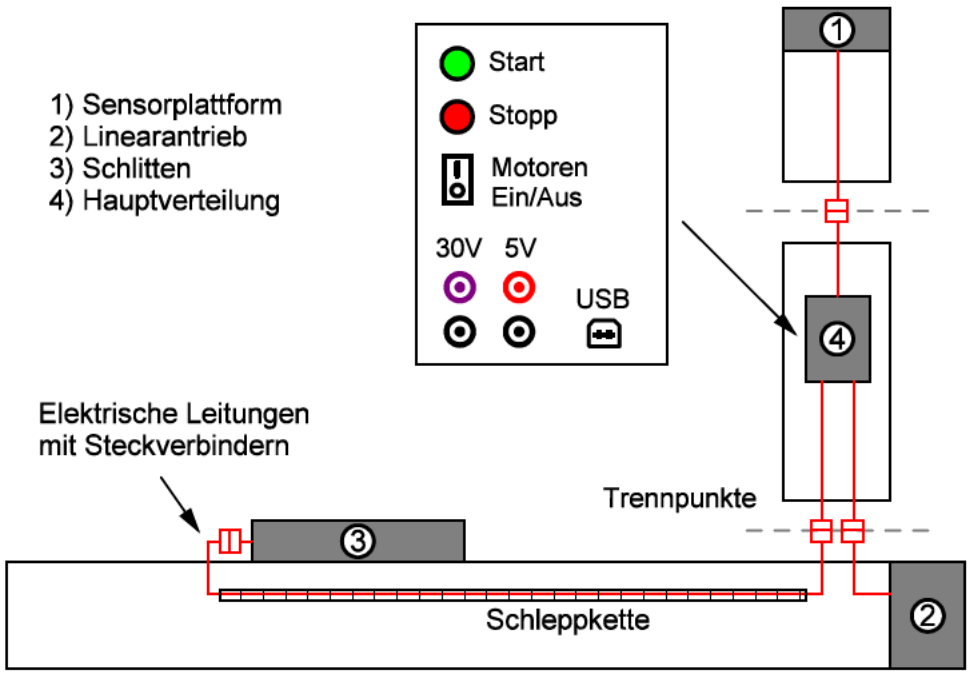


Abbildung 3-3: Konzept für die Elektrik

Die Motortreiber werden unmittelbar in der Nähe der jeweiligen Schrittmotoren platziert. Von den drei Achsen müssen Leitungen zur Hauptverteilung geführt werden, welche die Spannungsversorgung der Treiber gewährleisten sowie die Steuersignale vom Controller zu den Treibern und die Sensorsignale der Positionssensoren zum Controller übertragen. Zugunsten der Materialkosten wird für alle Achsen derselbe Kabeltyp verwendet. Die erforderlichen Eigenschaften des Kabels ergeben sich daher aus der Summe von verschiedenen Anforderungen der jeweiligen Achsen. Da die elektrische Zuführung des linear beweglichen Schlittens mechanisch stärker beansprucht wird, muss das Kabel schleppkettentauglich sein. Die Anzahl der benötigten Leitungen setzt sich aus der Spannungsversorgung und den Signalleitungen zusammen. Die Motortreiber werden an allen Achsen mit jeweils einer Leitung für die Betriebsspannung und die Steuerspannung sowie zwei Leitungen für die Steuersignale (Richtung und Geschwindigkeit) verbunden. Hinzu kommen Signalleitungen für die Endlagenschalter und eine gemeinsame Masseleitung. Der Schlitten benötigt mit seinen zwei Endlagenschaltern die größte Aderzahl von sieben. Der Linearantrieb verfügt über die größte Leistung und gibt damit den Querschnitt der Leitungen vor. Die Leistungsaufnahme des Motortreibers wird im Datenblatt nicht angegeben, weshalb die interne Absicherung des Geräts in Form einer 10 A Schmelzsicherung als Bemessungsgröße herangezogen wird. Für die Einzeladern wird demnach ein Querschnitt von 0,75 mm² gewählt. Gemäß DIN VDE 0298-4 darf dieser Querschnitt bei 30 °C Umgebungstemperatur mit bis zu 12 A belastet werden (LAPP Kabel, S. 1021). Unter Berücksichtigung der aufgeführten Anforderungen wurde der Kabeltyp ÖLFLEX CHAIN 809 7G 0,75 der Firma LAPP gewählt. Für die Demontierbarkeit des Prüfstands werden die Kabel mit Steckverbindern versehen, welche große Ströme und Signale übertragen können. Hierfür wurden Sub-D Steckverbinder mit einer gemischten Kontaktbelegung gewählt, wie sie in Abbildung 3-4 dargestellt sind.

Abbildung 3-4: Sub-D Mischkontakte (RS Components 2022)

Die Spannungsversorgung des Prüfstandes soll mit Hilfe eines Labornetzteils erfolgen. Dieses muss über zwei Kanäle verfügen, welche die Spannungsebenen von 30 V für die Motoren und 5 V für die Steuerung bereitstellen. Zur Erhöhung der Sicherheit soll der zentrale Schaltkasten über einen Hauptschalter verfügen, über den die Spannungsversorgung für alle Motoren abgeschaltet werden kann.

3.2 Software-Konzept

Das Software-Konzept erstreckt sich auf zwei Hardwareumgebungen in Form eines PCs für die Bedienung und eines Mikrocontrollers für die Steuerungsaufgaben. Der PC enthält eine Benutzeroberfläche, von welcher der Prüfstand bedient werden kann. Die Programmierung soll in der Programmiersprache Python erfolgen. Der Mikrocontroller übernimmt die direkte Ansteuerung der Schrittmotoren und verarbeitet die Sensorsignale des Prüfstands. Da es sich bei dem verwendeten Controller um einen Arduino handelt, wird die zugehörige Programmierumgebung Arduino IDE verwendet, welche auf der Programmiersprache „Processing“ basiert. Die Kommunikation zwischen Controller und PC erfolgt über eine serielle Verbindung und bedarf der Festlegung eines Kommunikationsprotokolls. Im Folgenden werden die Software-Konzepte für die zwei Bereiche PC und Controller vorgestellt.

3.2.1 PC-Software

Für die Programmierung der PC-Software fällt die Wahl auf die Programmiersprache Python, da diese leicht zu erlernen ist und über eine übersichtliche Syntax verfügt (Theis 2019, S. 17). Die Entwicklung des Programms wird auf das Betriebssystem Windows 10 ausgelegt. Es wird eine Benutzeroberfläche programmiert, mit der gemäß der Anforderungen aus der Aufgabenstellung die folgenden Funktionen ausgeführt werden können:

- Verbindungsaufbau zum Controller
- Initialisierung der Bewegungsachsen
- Positionierung der Drehachsen per Winkelangabe
- Translatorische Achse: Einstellung von Beschleunigung, Geschwindigkeit und Position mit Prüfung der Verfahrbarkeit
- Anzeige der Positionen aller Achsen
- Manuelles Stoppen einer Achsenbewegung zu jedem Zeitpunkt
- Ausgabe von Statusmeldungen

Die Funktionen wurden zusammengefasst und in einen Entwurf einer Benutzeroberfläche übertragen (siehe Abbildung 3-5). Im Hintergrund der Oberfläche werden Programmabläufe ausgeführt, welche im Folgenden allgemein beschreiben werden. Nach dem Starten des Programms muss zunächst eine Verbindung zum Controller hergestellt werden. Hierfür kann der COM-Port ausgewählt werden, unter dem der Controller in das Betriebssystem eingebunden wurde. Nach erfolgreicher Herstellung der Verbindung werden über entsprechende Schaltflächen die einzelnen Achsen initialisiert und so der Prüfstand in einen definierten Zustand überführt. Von da an befindet sich das System im normalen Betriebszustand und die einzelnen Achsen können nach Wunsch verfahren werden. Das Programm rechnet die eingegebenen Positions- und Geschwindigkeitsangaben in die entsprechenden Schritte der Schrittmotoren um und übermittelt diese an den Controller. Dabei wird auch überprüft, ob das eingegebene Geschwindigkeitsprofil am Prüfstand rea-

lisierbar ist. Um die zeitkritische Ansteuerung der Schrittmotoren durch den Controller nicht zu kompromittieren, kann während einer laufenden Achsbewegung kein weiterer Befehl an den Controller gesendet werden. Davon ausgenommen ist der Stopp-Befehl. Die gegenseitige Verriegelung der Befehle wird durch die zeitweise Deaktivierung der jeweiligen Schaltflächen in der Benutzeroberfläche erreicht. Ein Meldungsfenster gibt Rückmeldungen, bspw. über den Verbindungsstatus, abgeschlossene Bewegungen oder Fehlermeldungen, aus.

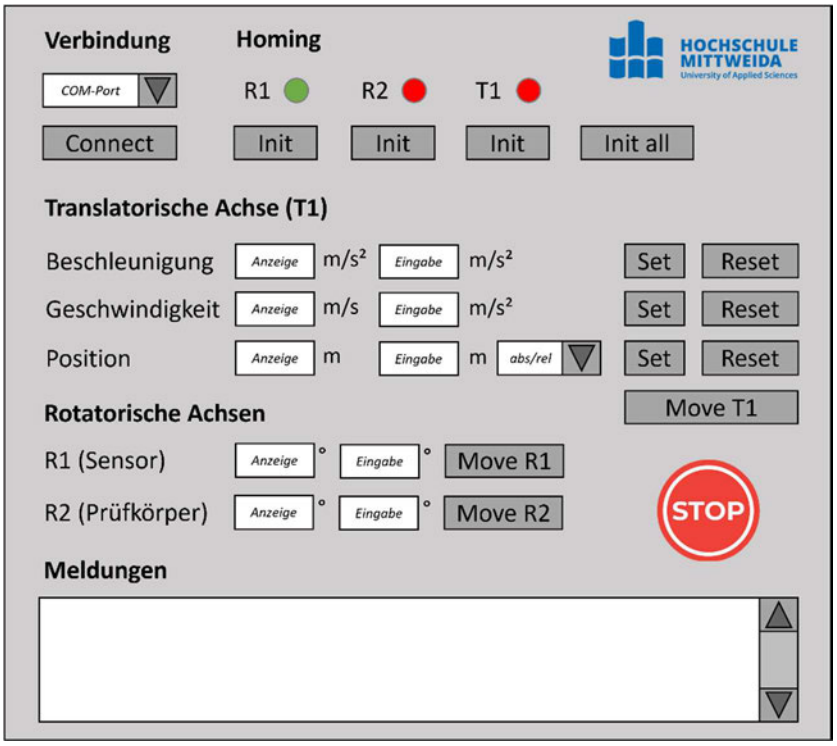


Abbildung 3-5: Entwurf der Benutzeroberfläche

3.2.2 Controller-Software

Der Controller in Form eines Arduino Mega 2560 hat die Hauptaufgabe, die Steuerungssignale für die Schrittmotoren zu generieren. Außerdem verarbeitet er die Signale der Positionssensoren und Taster und liefert Ausgangssignale zur Visualisierung des Status des Prüfstands. Der Controller steht dabei in ständiger Verbindung zur PC-Software. Abbildung 3-6 zeigt den Ablaufplan für die Hauptfunktionen, welche in einer Endlosschleife aufgerufen werden. Es werden nacheinander neue Nachrichten von der PC-Software empfangen und verarbeitet, die Sensor- und Tastersignale eingelesen, die Outputs für die Visualisierung gesetzt und die Steuersignale für die Schrittmotoren aktualisiert.

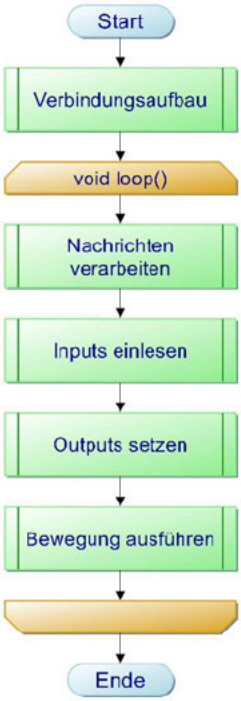


Abbildung 3-6: Ablauf der Hauptschleife im Mikrocontroller.

Für die Umsetzung der Programmieraufgaben wird auf die zahlreichen Bibliotheken aus der Arduino-Gemeinschaft zurückgegriffen. Das Kernstück der Steuerung der Schrittmotoren bildet die Programmbibliothek AccelStepper von Mike McCauley (AirSpayce 2022). Die Bibliothek erfüllt alle Anforderungen für die Umsetzung in diesem Projekt. So können mehrere Motoren gleichzeitig und unabhängig voneinander mit einer einstallbaren Beschleunigung und Verzögerung angesteuert werden. Die Objekte für die Ansteuerung der Motoren sind „nonblocking“, d.h. es kann während einer Motorbewegung weiterer zeitkritischer Programmcode ausgeführt werden. Dies ist z.B. wichtig für das Empfangen und Ausführen eines Stopp-Befehls. Die Programmbibliothek nutzt zur Geschwindigkeitsberechnung Formeln aus einer Publikation von David Austin (2005). Dort wird beschrieben, wie die Geschwindigkeitswerte des Schrittmotors durch Approximation berechnet werden, so dass sie auch auf einem einfachen Mikrocontrollersystem wie dem Arduino schnell genug ausgeführt werden können. Die Genauigkeit der Approximation steigt mit der Anzahl der auszuführenden Motorschritte. Die Abweichung zum theoretischen exakten Geschwindigkeitswert beträgt im zweiten Schritt 1 % und bereits im fünften Schritt nur noch 0,1 %.

Für die Einrichtung der seriellen Kommunikation zwischen Mikrocontroller und PC wird ebenfalls eine Programmbibliothek verwendet. Die Bibliothek CmdMessenger von Thijs Elenbaas und Valeriy Kucherenko ermöglicht es, eine Liste von Befehlen zu definieren, welche auf einfache Weise im Programmcode eingebunden, gesendet und empfangen werden können (Arduino 2022). Eine entsprechende Bibliothek für die Einbindung in Python ist ebenfalls verfügbar.

4 Vorbereitungen für die Umsetzung des Projekts

Nachdem ein Konzept für die Lösung der Aufgabenstellung erarbeitet wurde, werden im nächsten Schritt Vorbereitungen dafür getroffen, um das Projekt zu realisieren zu können. Dazu gehörten zunächst die Entwicklung einer elektrischen Schaltung und die Erstellung entsprechender Schaltungsunterlagen sowie die Beschaffung zusätzlich benötigter Bauteile. Weiterhin wurden die für die Verwendung vorgesehenen Hard- und Software-Komponenten erprobt, bevor sie in dem Sensorteststand implementiert wurden. Dazu zählen vor allem die Schrittmotoren mit ihren Motortreibern und die fertigen Softwarebibliotheken für die Mikrocontrollersteuerung.

4.1 Entwicklung der elektrischen Schaltung

Für die Umsetzung des Projekts wurde eine elektrische Schaltung entworfen, welche sich auf die Planungen aus Abschnitt 3.1.3 und 3.1.4 stützt. Der fertige Schaltplan befindet sich im Anhang dieser Arbeit. Die Schaltung gliedert sich in vier Hauptelemente, bestehend aus der zentralen Verteilung, dem Schlitten, der Sensorplattform und dem Linearantrieb. Jedes der Module verfügt über Reihenklemmen, an denen die Verbindungsleitungen angeschlossen werden. Die Leitungen für die Spannungsversorgung und Signalübertragung der Motortreiber und Sensoren wurden auf die Reihenklemmen geführt und gekennzeichnet. Die Verdrahtung der Schrittmotoren zu ihren Motortreibern wurde den Datenblättern entnommen (ACT Motor 2017a, 2017b). Für die Verbindungsleitungen wurde ein separater Stromlaufplan erstellt, aus welchem die Kontaktierungen zu den Steckverbindern ersichtlich sind. Der Schaltplan der Hauptverteilung legt die Zuordnung der Signalleitungen zu den I/O-Pins des Arduino Boards fest und definiert die Verdrahtung der Bedienelemente und Anschlussbuchsen.

Die Vorwiderstände für die LEDs der Leuchttaster wurden zunächst für einen Betriebsstrom von rund 20 mA berechnet und nach einem Test der Schaltung angepasst (siehe Abschnitt 4.3.2). Während der Belegung der Kontakte der Schlitten-Einheit fiel auf, dass die vorgesehene Schleppkettenleitung auf Grund eines Planungsfehlers über zu wenig Einzeladern verfügte. Nach Abwägung von Änderungsmöglichkeiten und der Verifizierung mittels eines Versuchs, wurde die Entscheidung getroffen, die Motortreiber als 5 V Spannungsquelle zu nutzen (siehe Abschnitt 4.3.3). Daraus ergab sich der Vorzug, dass auf eine externe 5 V Einspeisung verzichtet werden kann. Die Spannungsversorgung des Mikrocontrollers und dessen Peripherie kann über die ohnehin notwendige USB-Verbindung zum PC erfolgen. Die Strombelastbarkeit eines USB-Ports von üblicherweise 500 mA bietet genügend Marge für die konzipierte Anwendung.

4.2 Materialbeschaffung

Für die Erfüllung der Aufgabenstellung war es notwendig, den Sensorteststand an einigen Stellen zu verbessern, bzw. zu erweitern. Hierfür wurden zusätzliche Bauelemente und Materialien recherchiert und beschafft. Das zugewiesene Budget für die Umsetzung des Projekts betrug 500 €. Im Anhang der Arbeit ist eine detaillierte Auflistung des gekauften Materials einsehbar. Tabelle 4-1 zeigt die beschafften Teile, untergliedert in die Kategorien Elektroinstallation, Mechanik und Elektronik. Insbesondere zur Sicherstellung der Feldtauglichkeit des Sensorteststands bedurfte es einer Vielzahl zusätzlicher Teilen im Bereich der Elektroinstallation. In Kombination mit Verbesserungen im Bereich der Mechanik konnte so die Robustheit und Wiederverwendbarkeit des Teststands erhöht werden. Durch die Ergänzung von Leuchttastern wurde die Bedienbarkeit verbessert. Die Verwendung von Anschlussbuchsen und Steckverbindern erleichtert die Inbetriebnahme des Teststands erheblich.

Elektroinstallation	**Mechanik**	**Elektronik**
Schleppkette	Motorhalter	Gabellichtschranken
Schleppkettenleitung	Sortiment Zylinderschrauben M3/M4/M5	Widerstände
D-Sub Stecker	Flanschlager UCFL	Hutschinenmontage Arduino
Reihenklemmen	Stahlwelle	
4 mm Einbaubuchsen		
Drehschalter		
Drucktaster		
USB Einbauadapter		
Hutschienen		
Verteilerdosen		
Schaltschrank		
Aderendhülsen		
Kabelbinderhalter		

Tabelle 4-1: Zusätzlich beschafftes Material für das Projekt

4.3 Test von Hardware-Komponenten

4.3.1 Inbetriebnahme der Schrittmotoren

Zu Beginn des Projekts wurden die bereitgestellten Schrittmotoren versuchsweise in Betrieb genommen, um Erkenntnisse über deren Handhabung und Betriebsverhalten zu gewinnen. Die in den Datenblättern dokumentierten elektrischen und mechanischen Eigenschaften der Motoren und den dazugehörigen Motortreibern wurden bereits in den Abschnitten 3.1.2 und 3.1.3 beschrieben. Für die versuchsweise Inbetriebnahme wurde die kleinere Bauform der Motoren vom Typ 23SSM6440-EC1000 herangezogen. Da die Steuerung beim größeren Typ 34SSM8460-EC1000 identisch ist, ließen sich die gewonnen Erkenntnisse auf diesen Typ übertragen. In einem ersten Versuch, den Motor in Bewegung zu setzen, wurde ein Funktionsgenerator verwendet, um die Schrittfrequenz des Motors vorzugeben. Abbildung 4-1 zeigt den Aufbau der Versuchsschaltung. Die Betriebsspannung des Motortreibers wurde durch ein Labornetzteil bereitgestellt. Um den Motor in eine Richtung zu bewegen, genügt es, ein Rechtecksignal mit einer Amplitude von 5 V an den Anschluss PUL+ des Motortreibers anzulegen. Die Frequenz des Signals entspricht dabei der Frequenz der vollzogenen Schritte des Motors. Konstruktiv benötigt der Motor 200 Schritte für eine volle Umdrehung im Vollschrittmodus (siehe Abschnitt 2.2). Die Motortreiber bieten keinen Vollschrittmodus an, sondern nur den Halb- und Mikroschrittmodus. Die kleinste Schrittzahl für eine volle Umdrehung ist demnach 400. Der Test mit dem Funktionsgenerator hatte zum Ergebnis, dass sich die Motoren problemlos auf diese Weise ansteuern lassen. Durch Stoppen der Zeit und Zählen der Umdrehungen der Motorwelle konnte das theoretische Verhältnis von den vorgegebenen Schritten zur Schrittfrequenz verifiziert werden. So vollzieht die Motorwelle z.B. bei einer Ansteuerung mit 400 Hz eine volle Umdrehung pro Sekunde. Ebenfalls wurde nachgewiesen, dass der Motor bei seiner Nenndrehzahl von 1 000 min^{-1} und der entsprechenden Eingangsfrequenz von 6 667 Hz im Leerlauf ruhig läuft. Die maximale Stromaufnahme bei den Tests betrug 0,75 A.

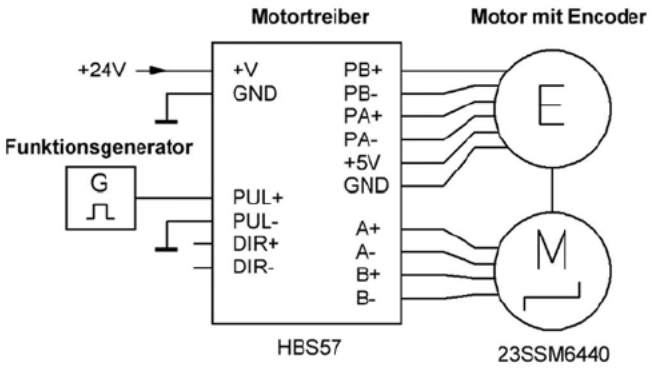


Abbildung 4-1: Ansteuerung des Schrittmotors mit einem Funktionsgenerator

In einem weiteren Versuch wurde getestet, wie sich grundsätzlich die Ansteuerung der Schrittmotoren mittels eines Mikrocontrollers realisieren lässt. Hierfür wurde ein Arduino Uno mit einem Taster und einem Potentiometer als Bedienelemente verwendet (siehe Abbildung 4-2). Für den Versuch wurde ein einfaches Programm in der Arduino IDE erstellt. Bei Betätigen des Tasters wird mittels Hardwareinterrupt im Mikrocontroller die Spannung am Eingang DIR+ des Motortreibers zwischen 0 und 5 V umgeschaltet und so die Drehrichtung des Motors gesteuert. Das Potentiometer liefert einen Analogwert zwischen 0 und 5 V, welcher vom Mikrocontroller auf einen Integer zwischen 50 und 2 000 gemappt wird. Der Wert definiert in Mikrosekunden die Pausen zwischen Pulsen, welche an PUL+ gesendet werden. Dieser einfache Versuch hat gezeigt, dass die Ansteuerung der Schrittmotoren mittels Arduino auf einfache Weise möglich ist und zunächst keine unerwarteten Probleme auftreten. Dennoch wurden Schwachstellen in dem für den Versuch erstellten Programmcode sichtbar. Durch Messen der Steuerpulse des Arduino an einem Oszilloskop wurde sichtbar, dass die Map-Funktion für die Analogwerte sehr rechenlastig ist und das Timing der Steuerung negativ beeinflusst. Das Signal des Tasters wurde nicht digital entprellt, was zu einem teils unkontrollierten Richtungswechsel des Motors führte.

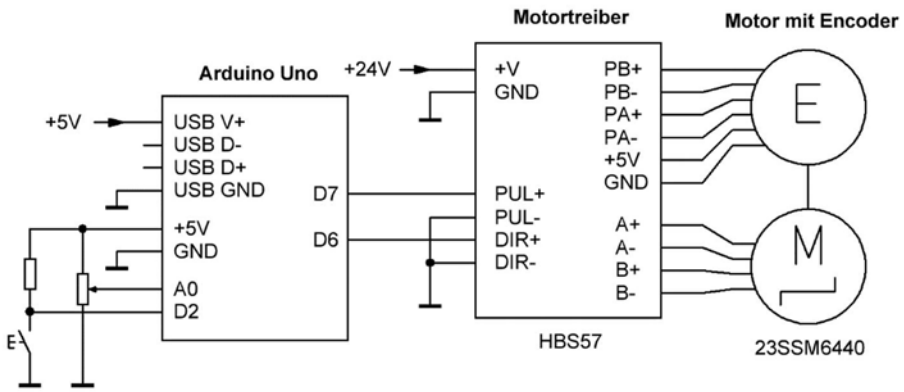


Abbildung 4-2: Ansteuerung des Schrittmotors mit einem Mikrocontroller

4.3.2 Test weiterer Hardware-Komponenten

Neben den Schrittmotoren und ihren Motortreibern wurden weitere elektrische und elektronische Komponenten für die Umsetzung des Projekts benötigt (siehe Abschnitt 3.1, 4.1 und 4.2). Bevor alle Teile endgültig im Sensorteststand verbaut wurden, sollten einige Komponenten vorab auf ihre Funktionsweise und Eignung geprüft werden. Zu den zu testenden Komponenten zählten der Drehschalter, die Drucktaster und deren LEDs sowie die Gabellichtschranken. Ziel der Tests war es zu überprüfen, ob die gewählten Komponenten im Zusammenspiel mit dem verwendeten Arduino Mega 2560 geeignet sind. Abbildung 4-3 zeigt die entwickelte Schaltung, in der die zu testenden Komponenten als Ein- und Ausgabegeräte mit dem Mikrocontroller verbunden wurden. In diesem Zusammen-

hang wurde auch die USB-Buchse für die Frontblende des Schaltschranks in die Signalkette eingebunden. Für den Versuch wurde ein Programm in der Arduino IDE erstellt, welches die LEDs ansteuert und die Eingaben der Signalgeber erfasst. Um eine Aussage über das Prellverhalten der mechanischen Signalgeber zu erhalten, wurde ein Zähler implementiert, welcher die erfassten Low-High-Transitionen über den seriellen Monitor ausgibt.

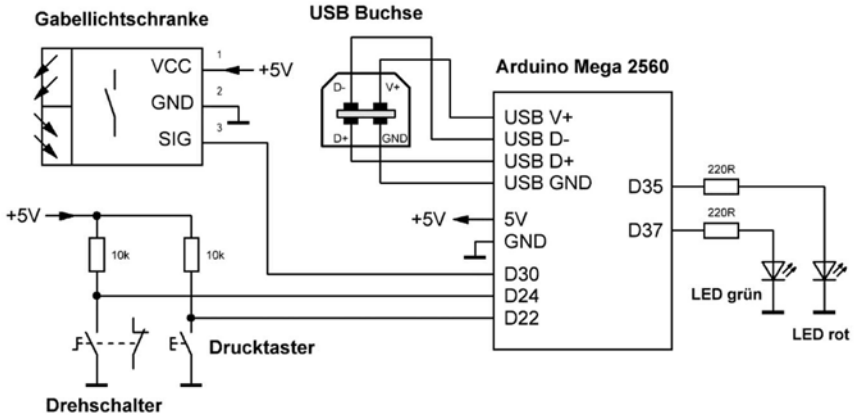


Abbildung 4-3: Testschaltung für die weiteren Hardware-Komponenten

Der Test des Drucktasters ergab, dass der Mikrocontroller bis zu drei Pegelwechsel bei einer Betätigung erfasste. Beim Test des Drehschalters lag der Wert mit bis zu sieben erfassten Pegelwechseln noch höher. Das auftretende Prellverhalten kann mit geeigneter Programmierung umgangen werden. Davon abgesehen haben sich die mechanischen Signalgeber als tauglich erwiesen. Beim Auslösen der Gabellichtschranke, durch das Einbringen eines Gegenstands in den Lichtstrahl, wurde vom Mikrocontroller ein sauberer Pegelwechsel erfasst. Im späteren Verlauf des Projekts stellt sich heraus, dass die Lichtschranken Störimpulse an ihrem Signalausgang ausgeben (siehe Abschnitt 6.2.3). Dieses Verhalten wurde in dem hier beschriebenen Versuch nicht beobachtet. Der Test der LEDs der Drucktaster hat gezeigt, dass die rote LED mit dem errechneten Vorwiderstand von 220 Ω (siehe Abschnitt 4.1) ein gut sichtbares Licht abgibt. Die grüne LED hingegen leuchtete nur sehr schwach. Eine Messung ergab, dass durch die LED mit dem dimensionierten Widerstand von 220 Ω ein Strom von 14 mA fließt. Um die Helligkeit zu erhöhen, wurde der Vorwiderstand halbiert, in dem ein baugleicher Widerstand parallelgeschaltet wurde. Die Sichtbarkeit der grünen LED hat sich dadurch verbessert, wenngleich sie trotzdem nicht so ausgeprägt war wie die, der roten LED. Der gemessene Strom mit dem geänderten Vorwiderstand betrug nun 25 mA und sollte nicht weiter erhöht werden. Insgesamt hat der Versuchsaufbau gezeigt, dass die gewählten Bauelemente mit kleineren Anpassungen für das Projekt geeignet und nutzbar sind.

4.3.3 Nutzung der Motortreiber als 5 V Spannungsquelle

Im Verlauf der Planung des Projekts erwies es sich als vorteilhaft, wenn der 5 V Spannungsausgang der Motortreiber als lokale Spannungsversorgung für die optischen Sensoren genutzt werden könnte (siehe Abschnitt 4.1). Der Spannungsausgang ist lediglich dafür vorgesehen, die Betriebsspannung des angeschlossenen Encoders bereitzustellen. Durch einen Versuch wurde ermittelt, ob das interne Netzteil in der Lage ist, zusätzliche Leistung abzugeben, ohne dabei den sicheren Betrieb Encoders und damit des Schrittmotors zu gefährden. Abbildung 4-4 zeigt den Schaltungsaufbau für den Lastversuch.

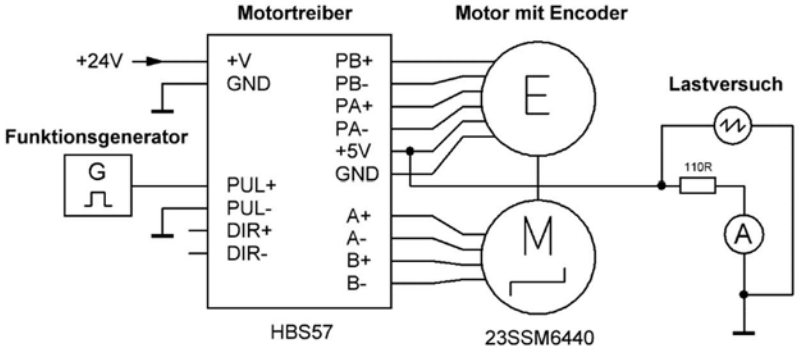


Abbildung 4-4: Lastversuch am 5 V Ausgang des Motortreibers

Eine Gabellichtschranke benötigt einen maximalen Strom von 10 mA. Bei bis zu drei angeschlossenen Lichtschranken und einer zusätzlichen Marge wurde der Laststrom für den Versuch auf 50 mA festgelegt. Als Lastwiderstand wurden zwei 220 Ω Widerstände parallelgeschaltet, wodurch die aufgenommene Leistung pro Widerstand halbiert werden konnte. Ein in Reihe geschaltetes Multimeter diente der Stromüberwachung und mit einem Oszilloskop wurde das Verhalten der Spannungsquelle untersucht. Für möglichst betriebsnahe Testbedingungen wurde der Motortreiber so angesteuert, dass sich der angeschlossene Schrittmotor mit der Maximaldrehzahl von 1000 min^{-1} dreht. Im ersten Teilversuch wurden die Spannung und Stromaufnahme ohne Zusatzlast gemessen. Diese betrugen 5,04 V und 94,4 mA. Im zweiten Teilversuch mit Zusatzlast stieg der Laststrom erwartungsgemäß auf 140 mA. Die Ausgangsspannung änderte sich dabei leicht auf 4,99 V. Die Messung mit dem Oszilloskop ergab, dass die Restwelligkeit der Spannungsquelle mit der zusätzlichen Last sogar noch sank (siehe Abbildung 4-5 und Abbildung 4-6). Dies lässt sich dadurch erklären, dass das verbaute Netzteil möglicherweise die Spannung bei höherem Laststrom besser einregeln kann. Die Ergebnisse des Versuchs haben gezeigt, dass die Motortreiber durchaus als 5 V Spannungsquelle für den projektierten Anwendungsfall genutzt werden können.

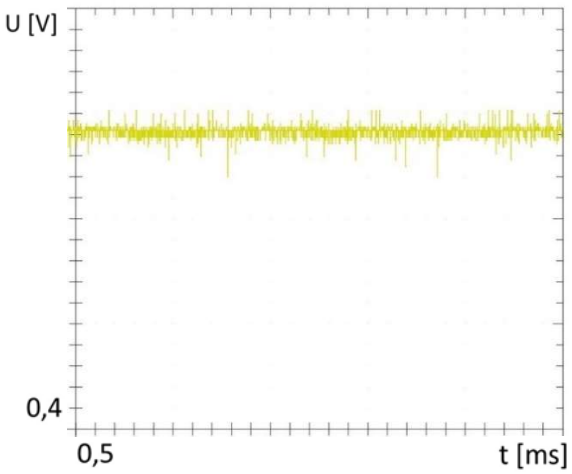


Abbildung 4-5: Spannungsverlauf ohne Zusatzlast

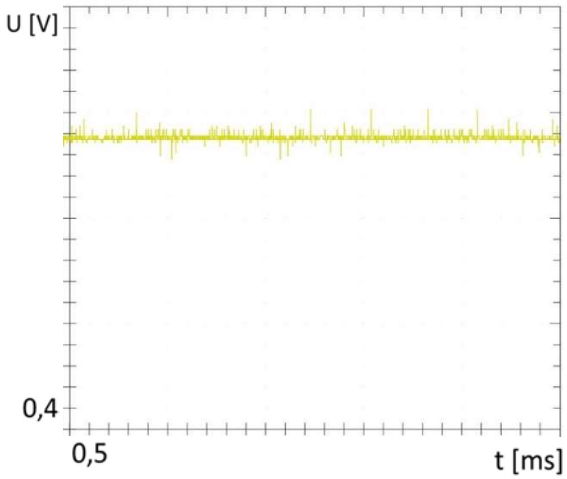


Abbildung 4-6: Spannungsverlauf mit Zusatzlast

4.4 Test von Software-Komponenten

4.4.1 Test der Arduino Bibliothek AccelStepper

Während der Konzipierung des Projekts wurde die Arduino Bibliothek AccelStepper für die Steuerung der Schrittmotoren ausgewählt. Bevor der Sensorteststand als Ganzes programmiert wurde, sollte zunächst die Eignung und Handhabung der Programmbibliothek untersucht werden. Für einen ersten Einstieg wurde ein fertiger Programmcode verwendet, der zur freien Verfügung ins Internet gestellt wurde (Curious Scientist 2019). Der Code nutzt die AccelStepper Bibliothek und wurde in einem Projekt mit einem ähnlichen Schrittmotor verwendet. Für die Tests wurde der gleiche Schaltungsaufbau wie in Abschnitt 4.3.1 verwendet, jedoch erfolgte von nun an die Befehlseingabe per Serial Monitor über die Arduino IDE. Das Testprogramm ermöglicht es, einen einzelnen Schrittmotor mit gewählter Beschleunigung, Endgeschwindigkeit und Schrittzahl zu bewegen. Die Durchführung des Tests hat gezeigt, dass die AccelStepper Bibliothek gut mit den vorhandenen Schrittmotoren zusammenspielt. Die kommandierten Bewegungen des Schrittmotors wirkten gleichmäßig und verstärkten den Eindruck, dass die Bibliothek für das Projekt geeignet ist. Durch eine Messung der Pulsfrequenz am Motortreibereingang wurde iterativ die maximal mögliche Schrittfrequenz ermittelt, welche der Mikrocontroller ausgeben kann. Diese lag bei etwa 5 000 Steps/s, was einer Motordrehzahl von 750 min^{-1} entspricht. In der Dokumentation der Programmbibliothek wird die maximale empfohlene Schrittfrequenz hingegen mit 4 000 Steps/s angegeben.

Nachdem dieser erste Einstieg gezeigt hat, dass es sinnvoll ist, mit der Bibliothek weiterzuarbeiten, wurde in einem weiteren Vorversuch ein eigenes Programm für die Schrittmotorensteuerung erstellt. Als Hilfestellung zur Einarbeitung in die Verwendung der Bibliothek wurde die offizielle Dokumentation von AccelStepper (AirSpayce 2022) sowie ein

Blogeintrag mit Erläuterungen genutzt (doctek 2022). Für die Befehlsübergabe an den Mikrocontroller per Serial Monitor wurde die Programmbibliothek CmdMessenger verwendet, welche im folgenden Abschnitt 4.4.2 näher erläutert wird. Ziel des zu erstellenden Programms sollte sein, einen einzelnen Schrittmotor mit definierter Beschleunigung, Endgeschwindigkeit und Schrittzahl zu bewegen sowie den Start und Stopp der Bewegung des Motors zu steuern. Der Funktionsumfang ähnelt dem des vorangegangenen Versuchs, jedoch sollte nun ein Programm nach eigenen Vorgaben mit der Möglichkeit der späteren Skalierung für die Steuerung aller Achsen des Sensorteststands entstehen.

Nach der Entwicklung des Testprogramms wurde es auf seine Funktionalität überprüft. Durch Rückmeldung der vollzogenen Schritte nach jeder Motorbewegung über den Serial Monitor wurde ersichtlich, dass die Bibliothek einen absoluten Wert des zurückgelegten Weges über die gesamte Dauer der Ausführung des Programms aktualisiert und speichert. Dies war nicht erwünscht, da das Speichern der Achspositionen später im Hauptprogramm am PC erfolgen sollte. Durch das Verwenden des Befehls move() und das Rücksetzen der Motorposition nach jeder Bewegung wurde das gewünschte Verhalten erreicht. In einem weiteren Test wurde geprüft, wie der Motor abrupt angehalten werden kann. Es hat sich gezeigt, dass der Befehl stop() der Programmbibliothek den Motor kontrolliert mit der eingestellten Beschleunigung abbremst. Gesucht war aber eine Möglichkeit, den Motor sofort zum Stehen zu bringen, etwa für einen Nothalt oder beim Anfahren einer Home-Position. Die Lösung lag in dem Prinzip begründet, wie die Bewegung des Motors von der Programmbibliothek berechnet wird. Hierfür wird die Funktion run() so oft aufgerufen, wie es im Programm möglich ist. Bei der Ausführung der Funktion wird überprüft, ob der Zeitpunkt erreicht ist, an dem der Motor einen neuen Schritt ausführen muss. Stoppt man per Eingabe den Aufruf von run(), wird auch die Bewegung des Motors augenblicklich beendet. Die Häufigkeit des Aufrufs von run() ist kritisch für das Timing der Motorbewegung. Der Befehl muss mindestens einmal pro durchzuführenden Schritt aufgerufen werden. Ist das Programm zu langsam, ist keine Bewegung der Motoren entsprechend den Beschleunigungs- und Geschwindigkeitsvorgaben möglich. Um einen Eindruck zu gewinnen, wie häufig der Befehl in dem Testprogramm aufgerufen wird, wurde ein Zähler implementiert. Der Test hat ergeben, dass der Befehl bei 10 000 durchzuführenden Schritten ca. 260 000 mal aufgerufen wurde, also 26 mal häufiger als minimal notwendig. Um die Programmausführung testweise zu verlangsamen, wurde während der Motorbewegung der Wert der Beschleunigung geändert. Dies erfordert die mathematische Berechnung von Wurzeln und es wird in der Dokumentation davon abgeraten, dies während einer Bewegung durchzuführen. Der Aufruf des run() Befehls hat sich durch die herbeigeführte Störung lediglich um etwa 300 verringert, allerdings war beim Aufruf des Beschleunigungsbefehls deutlich ein Stocken in der Motorbewegung wahrnehmbar. Die Schlussfolgerung aus dem Test war, dass der Mikrocontroller grundsätzlich schnell genug arbeitet, um die Motorbewegung ordentlich auszuführen, jedoch sollte sehr darauf geachtet werden, den Programmfluss nicht zu verlangsamen oder zu stören.

4.4.2 Test der Arduino Bibliothek CmdMessenger

Für die Übertragung von Befehlen und Daten zwischen Arduino und PC wurde die Programm Bibliothek CmdMessenger ausgewählt. Sie wurde für die Arduino Plattform entwickelt, um auf komfortable Weise umfangreiche Kommunikationsprotokolle zu erstellen und ist ebenfalls für Python verfügbar (Arduino 2022). Der Aufbau einer Nachricht besteht darin, dass ein Befehlsname definiert wird, dem beliebig viele Parameter angehangen werden. Zur Steuerung des Mikrocontrollers ist es möglich, die Befehle direkt über der Serial Monitor einzugeben. Für den Versuch in Abschnitt 4.4.1 wurde eine Befehlsliste erstellet, wie sie in Tabelle 4-2 abgebildet ist. Damit durch die Befehle vom Mikrocontroller Aktionen ausgeführt werden, müssen sogenannte Callbacks definiert werden. Der Test der Bibliothek war erfolgreich und es hat sich gezeigt, dass sie sich auch für die Entwicklungsphase sehr gut eignet. Es ist auf schnelle und unkomplizierte Weise möglich die Befehlsliste zu erweitern und Funktionen anzupassen. Die Anwendung ist gut skalierbar und die Nutzung des Protokolls zur Steuerung über den Serial Monitor hat sich während der gesamten Entwicklungszeit des Sensorteststands als praktisch erwiesen.

ID	Name	Parameter
0	mStopp	-
1	mStart	-
2	mSteps	[Steps]
3	mAcc	[Steps/s²]
4	mSpeed	[Steps/s]

Tabelle 4-2: Kommunikationsprotokoll für die Vorversuche

5 Errichtung des Sensorteststands

In dem folgenden Kapitel wird beschrieben, wie die erfolgten Planungen und Vorbereitungen in den mechanischen und elektrischen Aufbau des Sensorteststands umgesetzt wurden. Der Teil der Mechanik erstreckte sich über Arbeiten an den Baugruppen der Sensorplattform, des Linearantriebs und des Schlittens. Die Umsetzung der Elektrik und Elektronik untergliederte sich in die Teilgebiete des Aufbaus der Hauptverteilung, der Verdrahtung der Bewegungsachsen und der Herstellung der Verbindungsleitungen zur Hauptverteilung. Im letzten Abschnitt des Kapitels wird erläutert, wie die Ergebnisse der Arbeiten auf ihre Funktionstüchtigkeit überprüft wurden.

5.1 Mechanische Arbeiten

5.1.1 Mechanische Bearbeitung der Sensorplattform

Auf der senkrecht aufgestellten Trasse des Prüfstands befindet sich am oberen Ende die drehbare Plattform, auf der später die zu prüfenden Sensoren platziert werden sollen. Mit Hilfe des Schrittmotorantriebs kann diese Plattform axial verdreht werden. Für das Projekt konnte zu großen Teilen der bereits vorhandene Aufbau in seiner grundlegenden Konstruktion übernommen werden. Der Schrittmotor bewegt mit einem 20 zahnigen Ritzel einen Zahnkranz mit 139 Zähnen, auf dem eine Platte als Auflagefläche befestigt ist. Die senkrechten Stehbolzen zum Tragen des Zahnkranzes werden durch lange Schrauben realisiert. Als Optimierung zum vorangegangenen Projekt wurde die gesamte Konstruktion nun mit der darunterliegenden Aluminiumplatte fest verschraubt. Der Motor wurde mit einem speziell dafür vorgesehenen Befestigungswinkel an der Grundplatte angebracht. In dem verbliebenen Freiraum hinter dem Motor wurde der Motortreiber und an der Unterseite der Aluminiumplatte die Verteilerdose für den elektrischen Anschluss angebracht. Als eine Erweiterung der Konstruktion kam ein Positionssensor in Form einer Gabellichtschranke hinzu. Der Sensor musste knapp unterhalb der sich drehenden Platte angebracht werden, um dessen Bewegung erfassen zu können. Hierfür wurde eine Erhöhung aus Holz konstruiert, welche mit der Grundplatte verschraubt und auf deren Oberseite der Sensor angebracht wurde. Auf der Gegenseite wurde eine Schaltfahne angebracht, welche den Schaltvorgang der Lichtschranke auslöst. Für die Schaltfahne wurde ein herkömmlicher Befestigungswinkel verwendet, an dem ein Schenkel auf ein minimales Maß reduziert wurde. Abbildung 5-1 und Abbildung 5-2 zeigen das Ergebnis der Konstruktion der Sensorplattform. Die Grundplatte ist Teil des Trassensystems und durch lösbare Bolzen mit dem Rest des Teststands verbunden. Es ist somit möglich, die Sensorplattform bei Bedarf separat zu transportieren.

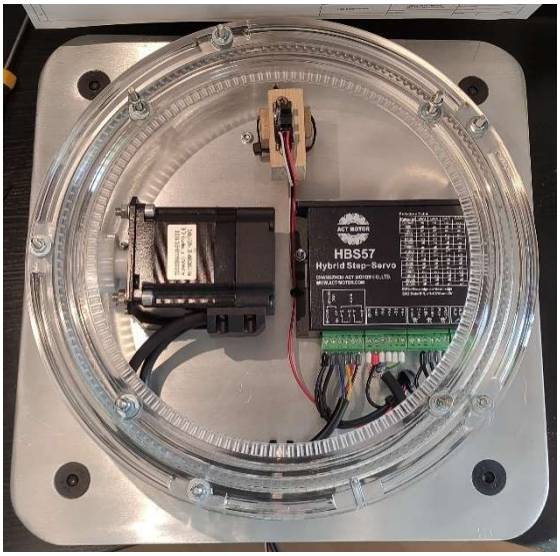


Abbildung 5-1: Anordnung der Hardware-Elemente der Sensorplattform

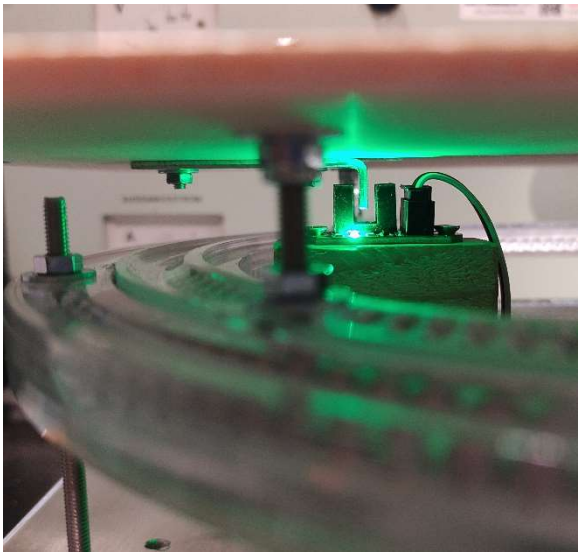


Abbildung 5-2: Konstruktion der Positionserfassung

5.1.2 Mechanische Bearbeitung des Linearantriebs

Für die translatorische Bewegung des Schlittens verfügt der Sensorteststand über einen Linearantrieb in Form eines Riementriebs. Die Einheit besteht aus der Antriebsseite mit dem Schrittmotor, dem Riemen für die Kraftübertragung und dem Gegenlager zum Spannen des Riemens. Die Konstruktion des Antriebs wurde weitestgehend aus dem vorangegangenen Projekt übernommen. Ein Rahmen aus Multiplexplatten ist mit dem Traversensystem verbunden und trägt den Schrittmotor. Auf der Motorwelle ist die Antriebsscheibe für den Zahnriemen verschraubt. An dem Rahmenwerk mussten Höhenanpassungen vorgenommen werden, damit der Zahnriemen über seine gesamte Länge waagerecht verläuft. Als Ergänzung wurden der Motortreiber und die Verteilerdose für die elektrischen Anschlüsse an der Rahmenkonstruktion montiert. Abbildung 5-3 zeigt die fertiggestellte Antriebseinheit. Der Zahnriemen konnte unverändert aus dem vorherigen Projekt weiterverwendet werden. Die konfektionierte Länge bietet eine gute Riemenspannung, wobei die Zugkräfte nicht so groß sind, dass sie die Lagerung des Antriebssystems zu stark belasten. An der Gegenseite mit der Umlenkrolle wurde die Lagerung vollständig überarbeitet (siehe Abbildung 5-4). Für die Umsetzung wurde ein Wälzlagersystem gesucht, dass wenig zusätzlichen Konstruktionsaufwand erforderte. Die Wahl viel hierbei auf UCFL 201 Flanschlager für einen Wellendurchmesser von 12 mm. Es wurden zwei Flanschlager gegenüberliegend positioniert und mit M8 Maschinenschrauben und selbstsichernden Muttern befestigt. In den Lagern wurde eine Welle mit einem Durchmesser von 12 mm fixiert. Die bereits vorhandene Riemenscheibe hat einen Bohrungsdurchmesser von 14 mm, weshalb die Verbindung zur Welle über eine Buchse mit einer Wandstärke von 1 mm realisiert wurde. Die Länge der Welle wurde so gewählt, dass bei eventuellen zukünftigen Modifizierungen andere Höhen für die Riemenscheibe eingestellt werden können. Für einen Transport des Sensorteststands kann der Zahnriemen zusammen mit dem Schlitten

leicht demontiert werden. Die Antriebseinheit ist mit dem Eckstück des Traversensystems, der sogenannten Multi Box verbunden. Diese kann im Transportfall vom Rest des Prüfstands getrennt werden und bildet zusammen mit dem Linearantrieb eine kompakte Einheit.

Abbildung 5-3: Antriebsseite des Linearantriebs

Abbildung 5-4: Gegenlager mit Umlenkrolle

5.1.3 Mechanische Bearbeitung des Schlittens

Wie bei den anderen Mechanik-Komponenten des Sensorteststands wurde auch der vorhandene Schlitten in seiner grundlegenden Konstruktion für das Projekt übernommen. Es wurden auch hier Anpassungen und Erweiterungen vorgenommen, um das angestrebte Betriebsverhalten zu erreichen. Der Schlitten verfügt über ein Rollensystem, mit welchem er während der translatorischen Bewegung auf den Schienen der Traverse geführt wird. Die Anordnung der Rollen war für eine reibungsarme Bewegung nicht optimal und wurde korrigiert. Weiterhin musste die vertikale Lage des Schlittens erhöht werden, damit sich der darunter befestigte Zahnriemen über den gesamten Verfahrweg frei bewegen kann. In Abbildung 5-6 ist die Konstruktion des Rollensystems zu sehen, welches sich an allen vier Ecken des Schlittens wiederholt. Für den Fall von einem Ansteuerungsfehler benötigte der Prüfstand Endanschläge, damit der Schlitten sicher gestoppt werden kann, ohne dass Anlagenteile beschädigt werden. Auf der einen Seite bildet der senkrechte Traversenteil einen festen Anschlag. Am Schlitten wurden Holzblöcke angebracht, über die im Falle einer Kollision die auftretenden Kräfte eingeleitet werden können. Am anderen Ende des Linearantriebs vor der Umlenkrolle wurden auf den Schienen Rohrschellen befestigt (erkennbar in Abbildung 5-4). In diese wurden Maschinenschrauben eingeschraubt, welche im Fehlerfall einen Anschlag für den Schlitten bilden. Die einfache Lösung mit den Rohr-

schellen wurde gewählt, da sich deren Position bei Bedarf leicht ändern lässt. Um die Bewegung des Schlittens bereits vor Erreichen der Endlagen stoppen zu können, wurde das System mit Endlagesensoren ausgestattet. Diese sind vom gleichen Typ wie jene zur Erfassung der rotatorischen Bewegungen der Drehachsen. Es wurde an der Unterseite des Schlittens jeweils ein Sensor zur Detektion der vorderen und hinteren Endlage angebracht. Als Schaltfahnen dienen handelsübliche Befestigungswinkel, deren Höhe auf die der Sensoren angepasst und an den Enden der Traverse befestigt wurden. Abbildung 5-6 zeigt einen der beiden Sensoren mit der zugehörigen Schaltfahne.

Auf der fahrbaren Plattform des Schlittens sind die Komponenten für den Drehtisch montiert (siehe Abbildung 5-5). Die verwendeten Hardware-Elemente sind identisch zu denen, die für die Sensorplattform verwendet wurden und deren Anordnung ist ähnlich. Aufgrund anderer Platzverhältnisse wurden die Verteilerbox und der Sensor für die Drehbewegung jeweils außen neben dem Zahnkranz platziert.

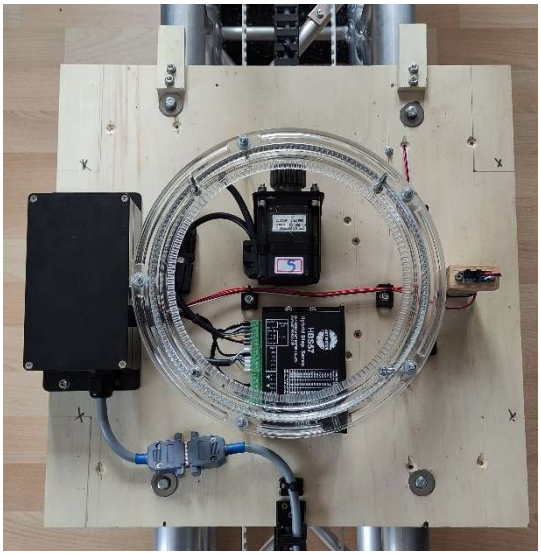


Abbildung 5-5: Anordnung der Hardware-Elemente auf dem Schlitten

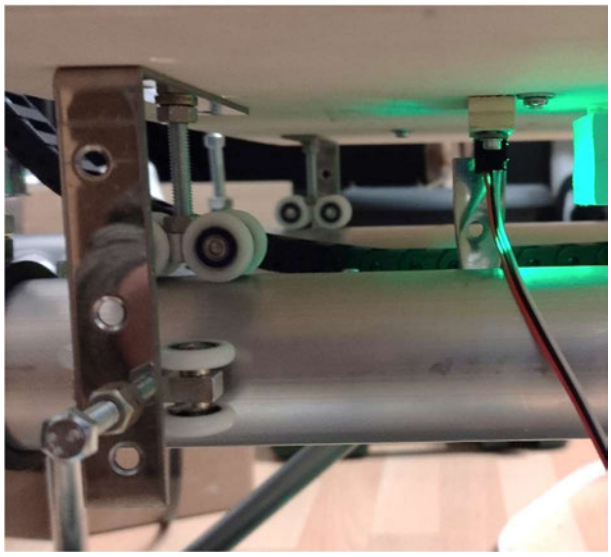


Abbildung 5-6: Unterkonstruktion des Schlittens

5.2 Elektrische Arbeiten

5.2.1 Aufbau der Hauptverteilung

Die Hauptverteilung des Sensorteststands ist in Form eines Schaltkastens am senkrechten Teil des Traversensystems angebracht. Das Modul erfüllt mehrere Aufgaben: die Beherbergung der zentralen Motorsteuerung, die Verteilung der elektrischen Energie und die Schnittstelle zur Bedienung. Im Inneren des Schaltkastens wurden Hutschienen für die Befestigung der Betriebsmittel angebracht und die Vorderseite wurde für die Anordnung von Bedien- und Anschlusselementen genutzt.

Abbildung 5-8 zeigt die Innenansicht der Hauptverteilung. Das Arduino Controllerboard für die Motorsteuerung wurde auf ein Prototyping-Shield mit Schraubklemmen und Klemmvorrichtungen an der oberen Hutschiene montiert. Das Shield enthält die notwendigen Pull-Up-Widerstände für den Schalter und die Taster sowie die Vorwiderstände für die LEDs der Taster. Die untere Hutschiene der Hauptverteilung enthält die Reihenklemmen, über welche die elektrische Verbindung zwischen Motorsteuerung, Energieversorgung und der Frontblende hergestellt wurden. Für die interne Verdrahtung wurde das noch in großen Mengen vorhandene Leitungsmaterial in Form von ummantelter 0,75 mm² Kupferleitung aus dem vorangegangenen Projekt genutzt. In Abbildung 5-7 ist zu sehen, wie die Bedienelemente und Buchsen an der Frontblende angeordnet sind. Der positive Eingang der Betriebsspannungsversorgung wird über den Hauptschalter auf die Reihenklemme geführt. Der negative Eingang wird auf den Reihenklemmen mit allen Massepotentialen der anderen Betriebsmittel verbunden. Die Taster mit ihren Leuchtmitteln werden ebenfalls auf die Reihenklemmen geführt und weiter mit dem Controllerboard verbunden. Die USB-Buchse für die Schnittstelle zum Host-PC wird über ein internes USB-Kabel direkt mit dem Arduino Mega verbunden. An der Ober- und Unterseite des Schaltkastens wurden Kabelverschraubungen für die Aufnahme der Verbindungsleitungen zu den Bewegungsachsen des Prüfstands montiert.

Abbildung 5-7: Frontblende der Hauptverteilung

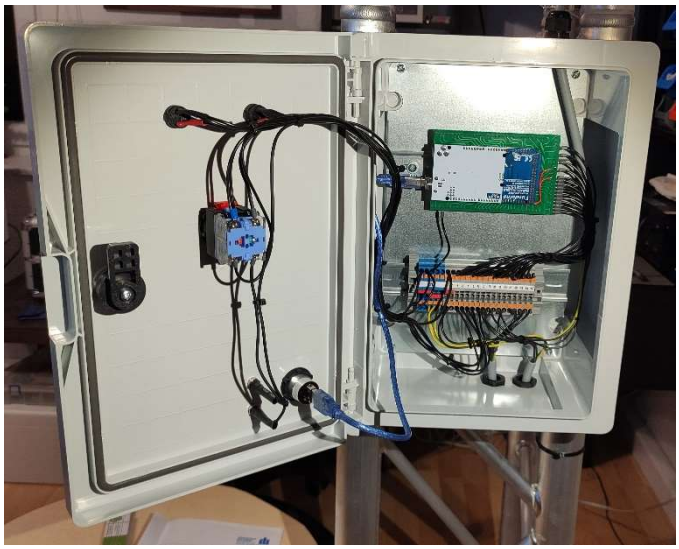


Abbildung 5-8: Innenansicht der Hauptverteilung

5.2.2 Elektrische Verdrahtung der Bewegungsachsen

Das Prinzip der elektrischen Verdrahtung ist bei allen drei Bewegungsachsen des Sensorteststands gleich. Die Schrittmotoren und die mit ihnen verbundenen Encoder verfügen jeweils über eine vorkonfektionierte Verbindungsleitung zum Motortreiber. Aufgrund der räumlichen Nähe der Motortreiber zu den Motoren wurden die Kabel eingekürzt. Die Einzeladern der Kabel wurden mit Aderendhülsen verpresst und an den Anschlussklemmen

der Motortreiber verschraubt. An den Achsen der Sensorplattform und des Schlittens wurde jeweils eine zusätzliche Leitung vom 5 V Ausgang der Motortreiber für die Spannungsversorgung der Positionssensoren abgeführt. Die Sensoren in Form der Gabellichtschranken wurden ebenfalls durch vorkonfektionierte Leitungen verbunden, welche für die Anwendung eingekürzt wurden. Die Verbindung zur Hauptverteilung für die Spannungsversorgung und Signalübertragung wird für jede Achse gebündelt über eine Anschlussdose mit Reihenklemmen geführt. Abbildung 5-9 zeigt exemplarisch die Anschlussdose des Schlittens. Als Anschluss zu den Verbindungsleitungen wurde jeweils ein Kabel mit einem Steckverbinder nach außen geführt.

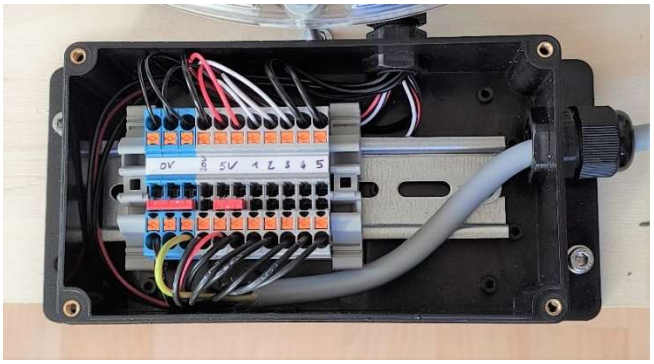


Abbildung 5-9: Anschlussdose für die elektrische Verbindung der Bewegungsachsen

5.2.3 Herstellung der Verbindungsleitungen

Die elektrischen Verbindungen zwischen den Bewegungsachsen und der Hauptverteilung wurden mit der in Abschnitt 3.1.4 beschriebenen siebenadrigen Schleppkettenleitung und den D-Sub Steckverbindern hergestellt. Die Leitungen wurden entlang der Traversen verlegt und an den vorgesehen Trennpunkten mit den Steckverbindern versehen. Am Eckpunkt des Sensorteststands verlaufen die Verbindungen des Linearantriebs und des Schlittens gemeinsam. Um ein Vertauschen der jeweiligen Steckverbinder zu verhindern, wurde die Polarität der Stecker entgegengesetzt gewählt. Abbildung 5-10 zeigt, wie die Kontaktierung zur Verbindungsleitung im Innern des Steckers durch Lötverbindungen hergestellt wurde. Die beiden Kontakte mit dem größeren Querschnitt werden für die Spannungsversorgung der Motortreiber genutzt. Die dünneren Kontakte dienen der Signalübertragung. Die Leitung zum Schlitten wurde für eine sichere Bewegbarkeit über eine Schleppkette geführt (siehe Abbildung 5-11).

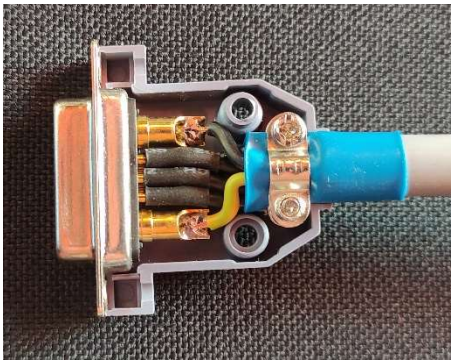


Abbildung 5-10: Kontaktierung der Steckverbinder

Abbildung 5-11: Teilansicht der Schleppkette

5.3 Kontrolle und Test der aufgebauten Anlage

Nach der Beendigung aller mechanischen und elektrischen Arbeiten war der Sensorteststand vollständig errichtet (siehe Abbildung 5-12). Bevor zum nächsten Arbeitsabschnitt in Form der Programmierung übergegangen werden konnte, wurde der Teststand auf seine korrekte mechanische und elektrische Funktionsfähigkeit überprüft.

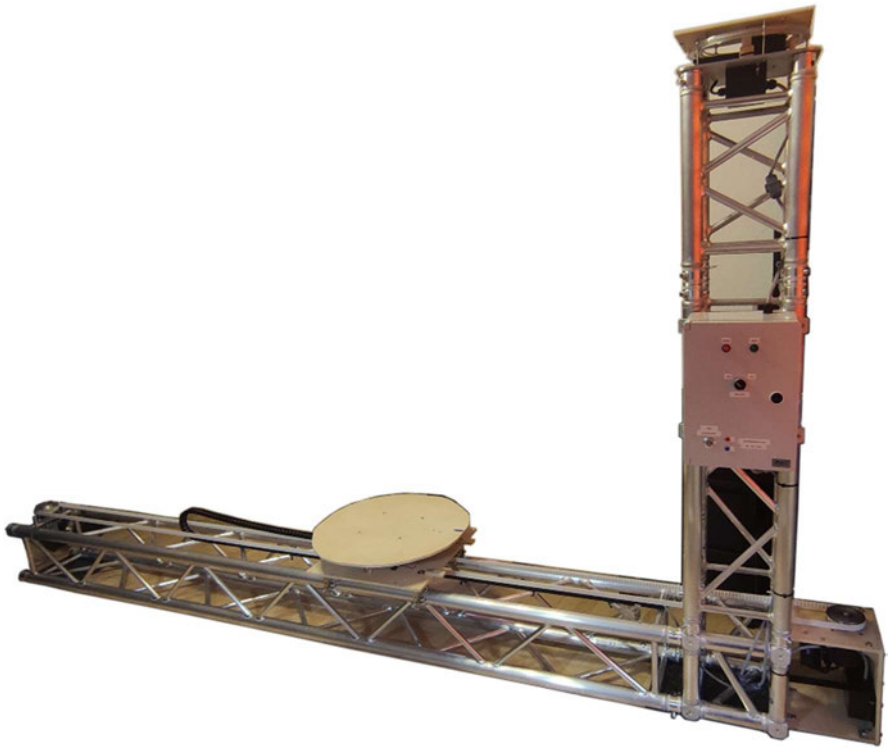


Abbildung 5-12: Vollständig aufgebauter Sensorteststand

Schon während der Aufbauphase wurden die Hauptverteilung und die Sensorplattform ersten Tests unterzogen. Zur Verifizierung der elektrischen Verdrahtung der Hauptverteilung wurde das Testprogramm für die I/O-Schnittstellen auf den Arduino geladen (siehe Abschnitt 4.3.2). Dabei wurden keine Fehler festgestellt. Zur Überprüfung der Konstruktion der Sensorplattform wurde diese provisorisch mit der Hauptverteilung verkabelt und das Programm für den Test der Schrittmotoren hochgeladen (siehe Abschnitt 4.3.1). Bei der rotatorischen Bewegung der Plattform gab es keine Probleme. Das Schalten der Gabellichtschranke konnte anhand ihrer Statusleuchte bestätigt werden. In einer iterativen Vorgehensweise wurden passende Beschleunigungs- und Geschwindigkeitswerte für die Motorbewegung ermittelt. Da die Drehbewegung der Plattform nicht zeitkritisch ist, wurden Werte gesucht, die subjektiv einen harmonischen Bewegungsablauf abbildeten. So fiel die Wahl auf eine Endgeschwindigkeit von 1 000 Steps/s mit einer Beschleunigung von 2 000 Steps/s². Im Anschluss wurde die Sensorplattform mit diesen Werten im Dauerlauf für eine halbe Stunde betrieben. Das Resultat aus diesem Test war, dass sich keine Schraubverbindungen gelockert haben und die Temperatur des Motors und Motortreibers nicht höher als handwarm wurde. Ebenfalls schon während des Aufbaus wurden die drei Lagesensoren am Schlitten in Betrieb genommen. Dabei zeigte sich, dass die Schaltfahnen für die Erfassung der Endlagen nicht hoch genug waren, um die Sensoren auszulösen. Die Höhe der Schaltfahnen wurde daraufhin korrigiert. Zum Ende der Aufbauphase wurden die Verbindungsleitungen zwischen der Hauptverteilung und den Bewegungsachsen angeschlossen. Dabei wurden alle elektrischen Pfade zwischen ihrem Anfangs- und Endpunkt durch eine Niederohmmessung überprüft. Nachdem die korrekte elektrische Verbindung aller Baugruppen sichergestellt war, konnte der Sensorteststand als Ganzes in Betrieb genommen und getestet werden.

Für einen gemeinsamen Test aller Bewegungsachsen wurde das Programm zum Test der Schrittmotoren aus Abschnitt 4.3.1 so erweitert, dass jede Achse mit beliebigen Beschleunigungs- und Geschwindigkeitswerten mittels Befehlseingabe über den Serial Monitor der Arduino IDE angesteuert werden konnte. Die Drehachse des Schlittens wurde mit Bewegungswerten angesteuert, welche bereits für die Sensorplattform ermittelt wurden. Diese erwiesen sich auch hier als brauchbar, was aufgrund der ähnlichen Konstruktion zu erwarten war. Für den Test des Linearantriebs wurde zunächst mit einer niedrigen Geschwindigkeit der Verfahrweg des Schlittens abgefahren. So wurde ermittelt, dass sich die zurückgelegte Stecke bei 2 000 Motorschritten noch in einem deutlichen Abstand zu den Endlagen befindet. Innerhalb dieser sicheren Fahrstrecke wurden durch schrittweise Steigerung mögliche Beschleunigungs- und Geschwindigkeitswerte getestet. Die Werte wurden gesteigert bis zu einer Motorbeschleunigung von 3 000 Steps/s² und einer Maximalgeschwindigkeit von 2 000 Steps/s angehoben, was der Hälfte der maximal möglichen Motordrehzahl entspricht. Die gesamte Konstruktion machte hierbei einen fähigen Eindruck. Zum Abschluss wurde der gesamte Teststand einem Dauerlauf unterzogen. Dabei wurden in einer Endlosschleife die drei Bewegungsachsen nacheinander vor und zurück bewegt. Für die Beschleunigung und Geschwindigkeit wurden die vorher beschriebenen Werte verwendet. Der Dauerlauf wurde für eine Stunde durchgeführt, wobei keine Prob-

leme auftraten. Zu Versuchszwecken wurde getestet, wie sich die Anlage verhält, wenn mehrere Achsen gleichzeitig angesteuert werden. Es hat sich gezeigt, dass die Bewegungen der einzelnen Achsen im simultanen Betrieb nicht mehr gleichmäßig vollzogen wurden. Dies bekräftigte die in der Konzeptionsphase getroffene Entscheidung, die einzelnen Motorbewegungen nur sequenziell durchzuführen.

Im weiteren Verlauf der Projektumsetzung hatte sich gezeigt, dass die Welle-Nabe-Verbindungen an den Motorwellen kritische Punkte sind. Sowohl das Ritzel an der Drehachse des Schlittens als auch die Riemenscheibe am Linearantrieb hatten sich in der Phase der Software-Entwicklung gelockert. Um die Dauerfestigkeit der Welle-Nabe-Verbindungen zu erhöhen, wurden Schraubensicherungen in Form eines anaeroben Klebers appliziert.

6 Programmierung der Software

Das Konzept für den Sensorteststand sieht vor, dass sich die Steuerung der Hardware auf eine PC-Software und auf einen lokalen Mikrocontroller verteilt. Zwischen den beiden Plattformen werden Daten über ein festgelegtes Kommunikationsprotokoll ausgetauscht. Der Controller übernimmt die Aufgaben der unmittelbaren Ansteuerung der Schrittmotoren und der Erfassung von Sensordaten und Eingaben über das Bedienpult. Die Entwicklung der Controller-Software erfolgte mit der Programmierumgebung Arduino IDE. Das PC-Programm wurde zu dem Zweck erstellt, eine einfache Bedienung des Sensorteststands per grafischer Benutzeroberfläche zu ermöglichen. Die Software wurde in der Programmiersprache Python mit der Entwicklungsumgebung Visual Studio Code erstellt. Im folgenden Kapitel wird die Entwicklung und Funktionalität der Software-Komponenten des Sensorteststands vorgestellt.

6.1 Definition eines Kommunikationsprotokolls

Die Datenübertragung zwischen PC-Software und Mikrocontrollersteuerung findet mittels einer seriellen Verbindung über einen USB-Anschluss statt. Für den Austausch von Befehlen und Informationen zwischen den beiden Endpunkten wurde ein Kommunikationsprotokoll festgelegt. Der Aufbau einer Nachricht wird durch die Programmbibliothek CmdMessenger vorgegeben und besteht aus einem Befehlsnamen, gefolgt von Parametern in Form von unterschiedlichen Datentypen (siehe Abschnitt 4.4.2). Tabelle 6-1 zeigt die endgültige Version des Protokolls, welches für den Sensorteststand verwendet wurde. Die farbliche Hervorhebung der Befehle zeigt an, in welche Richtung sie gesendet werden. Demnach werden die meisten Befehle nur in eine Richtung vom PC zum Controller oder umgekehrt verwendet.

ID	Name	Parameter 1	Parameter 2
0	mStopp	-	-
1	mStart	Axis [int]	-
2	mMoveCompleted	-	-
3	mMoveAborted	Input [int]	-
4	mSetSteps	Axis [int]	Steps [int]
5	mSetAcc	Axis [int]	Acceleration [float]
6	mSetSpeed	Axis [int]	max Speed [float]
7	mInputState	Input [int]	-
8	mSetInputOperation	Input [int]	Operation [int]

Legende
PC -> Controller
Controller -> PC
Beide Richtungen

Tabelle 6-1: Protokoll für die Kommunikation zwischen PC und Controller

Das Kommunikationsprotokoll hatte sich im Laufe des Projektfortschritts mehrmals geändert. Es sind Befehle hinzugekommen und wurden teilweise auch wieder entfernt, wenn sich herausstellte, dass sie nicht benötigt werden. Die Befehle für den Start und Stopp einer Motorbewegung (ID 1 und ID 0) und die Set-Befehle für Schrittanzahl (ID 4), Beschleunigung (ID 5) und Geschwindigkeit (ID 6) eines Motors wurden bereits früh benötigt und implementiert (siehe Abschnitt 4.4.2). Mit dem Parameter „Axis“ wird angegeben, für welchen Schrittmotor der Befehl gilt. Im zweiten Parameter der Set-Befehle wird der einzustellende Wert übertragen. Der Stopp-Befehl benötigt keine Angabe eines Motors, da bei Ausführung des Befehls grundsätzlich jede Motorbewegung gestoppt wird. Mit den Nachrichten der IDs 2 und 3 meldet der Mikrocontroller an die PC-Software, dass eine Motorbewegung vollständig ausgeführt bzw. vorzeitig abgebrochen wurde. Der zugehörige Parameter Input gibt an, ob ein Sensor oder der Stopp-Taster den Abbruch ausgelöst hat. Während der Entwicklungsphase wurden in den Nachrichten mit einem weiteren Parameter auch die zurück gelegten Schritte der Motoren zurückgemeldet. Dieser Parameter wurde in der Endversion entfernt, da die Information im Programm des Host-PCs nicht benötigt wird. Die Befehle mit den IDs 7 und 8 wurden für die Verwendung der Sensoren und Taster hinzugefügt. Mit dem Inputstate-Befehl kann der Status der Inputs abgefragt werden. Vom PC kommend gibt der erste Parameter des Befehls an, welcher Input abgefragt wird, während in die andere Richtung der Status des Inputs zurückgegeben wird. Mit dem SetInputOperation-Befehl können alle Sensoren und Taster aktiviert bzw. deaktiviert werden.

Weitere Nachrichten waren zunächst vorgesehen, sind aber im weiteren Verlauf wieder entfernt worden. Eine Nachricht diente dazu, zu signalisieren, dass der Mikrocontroller nach einem Verbindungsaufbau bereitsteht. Dies hat sich als nicht notwendig erwiesen, da die Python-Funktion für den Verbindungsaufbau zum Mikrocontroller dessen Status abfragt und die Programmausführung bei einer fehlgeschlagenen Verbindung unterbricht. Ein weiterer Befehl war dafür vorgesehen, dass der Mikrocontroller zurückmeldet, dass eine Nachricht empfangen wurde. In der Praxis zeigte sich, dass die Datenübertragung störungsfrei verläuft und es für die Programmausführung nicht notwendig ist, eine erfolgreiche Datenübertragung durch eine zusätzliche Nachricht zu bestätigen.

6.2 Entwicklung der Controller-Software

6.2.1 Hauptprogramm – setup() und loop()

Die Grundstruktur eines Programms für einen Arduino-Mikrocontroller besteht immer aus den zwei Hauptfunktionen setup() und loop(). Der Teil in der Funktion setup() wird nur einmal nach dem Einschalten des Mikrocontrollers durchlaufen, während die Funktion loop() in einer Endlosschleife aufgerufen wird. Abbildung 6-1 zeigt den Ablaufplan des Hauptprogramms der Mikrocontroller-Software. Die ersten vier Schritte im Ablauf sind der Teil des Setups. Für die verwendeten I/O-Pins des Mikrocontrollers wird definiert, ob es

sich um Ein- oder Ausgänge handelt. Die Objekte der AccelStepper Bibliothek für die drei Schrittmotoren erhalten Initialwerte für die Beschleunigung und Maximalgeschwindigkeit. Da diese Werte ausschließlich von der PC-Software verwaltet werden, sind die Initialwerte nach Einschalten des Controllers auf null gesetzt. Der letzte Teil des Setups ist der Kommunikation zum PC gewidmet. Die serielle Schnittstelle des Mikrocontrollers wird mit einer Baud-Rate von 115 200 initialisiert. Für die Bibliothek CmdMessenger werden die Callbacks verknüpft, welche beim Empfangen von Befehlen aufgerufen werden.

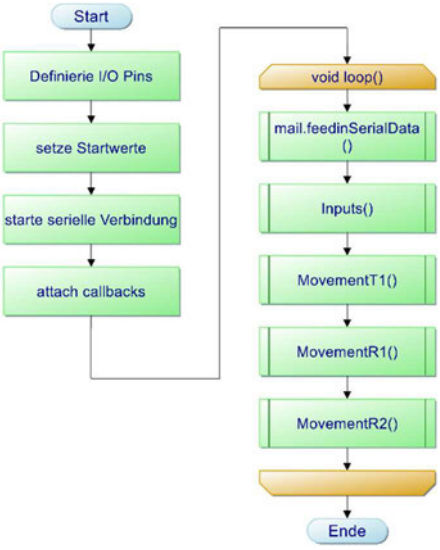


Abbildung 6-1: Setup und Hauptschleife der Controller-Software

In der darauffolgenden Endlosschleife werden die Hauptfunktionen aufgerufen, die für den Betrieb des Sensorteststands notwendig sind. Die Funktion feedinSerialData() ist Teil der CmdMessenger Bibliothek. Bei ihrem Aufruf wird überprüft, ob sich neue Daten im Buffer der seriellen Schnittstelle befinden. Ist dies der Fall, werden die Daten decodiert und entsprechende Callbacks aufgerufen (siehe Abschnitt 6.2.2). Mit der Funktion Inputs() werden die Signaleingänge des Mikrocontrollers abgefragt. Dies umfasst die Taster und Positionssensoren des Sensorteststands. Die genauere Funktionsweise dieses Programmabschnitts wird in Abschnitt 6.2.3 erläutert. Die drei Funktionen MovementT1(), MovementR1() und MovementR2() dienen dazu die Bewegungen der drei Schrittmotoren auszuführen und werden in Abschnitt 6.2.4 näher erläutert.

6.2.2 Die Funktion FeedinSerialData()

Wie bereits im vorangegangen Abschnitt beschrieben, wird die Funktion FeedinSerialData() kontinuierlich aufgerufen, um zu prüfen, ob neue Nachrichten vom Host-PC gesendet wurden. Ist ein neues Datenpaket eingetroffen, wird entsprechend dessen Nachrichtennummer (siehe Tabelle 6-1) ein Callback aufgerufen. Dies ist ein Unterprogramm, in dem Operationen ausgeführt werden, die den Nachrichten zugeordnet sind. Abbildung 6-2 zeigt den Programmablaufplan für die Funktion FeedinSerialData(). Wenn eine Nachricht

empfangen wurde, wird der Callback aufgerufen, der der Nachrichtennummer entspricht. Wurde keine Nachricht empfangen, wird die Funktion verlassen und die Hauptschleife läuft weiter.

Der Callback mDoStopp() ruft eine weitere Funktion zum Stoppen der Motoren auf (siehe Abschnitt 6.2.4) und überträgt eine Variable, die anzeigt, dass der Stopp-Befehl vom Host-PC ausgelöst wurde. Der Callback mDoStart() ruft eine Funktion zum Starten der Motoren auf und überträgt die als Parameter mitgesendete Nummer des zu bewegenden Motors (siehe hierzu ebenfalls Abschnitt 6.2.4). Die Callbacks mDoSetSteps(), mDoSetAcc() und mDoSetSpeed() sind ähnlich aufgebaut. Der erste übertragene Parameter der zugehörigen Nachricht gibt an, von wessen Motor der Wert geändert werden soll. Der zweite Parameter enthält den Wert der Schrittzahl, der Beschleunigung oder der Geschwindigkeit, welcher dann in der AccelStepper Bibliothek aktualisiert wird. Die Callbacks mDoInputState und mDoSetInputOperation beziehen sich auf die Sensoren und Taster des Teststands. Beim Aufruf von mDoInputState wird der gespeicherte Pegelwert des angefragten Inputs an den Host-PC gesendet. Im Parameter der empfangenen Nachricht wird angegeben, von welchem Input der Status gesendet werden soll. Auch bei mDoSetInputOperation gibt der erste Parameter die Nummer des Inputs an. Der Zweite Parameter signalisiert mit dem Wert 0 oder 1, ob der Input ein- oder ausgeschaltet werden soll. Der Callback setzt einen entsprechenden Wert für den jeweiligen Input, der definiert, ob dessen Status beim Aufruf der Funktion Inputs() abgefragt werden soll. Bei der Aktivierung des Start- und des Stopptasters wird zusätzlich deren LED eingeschaltet.

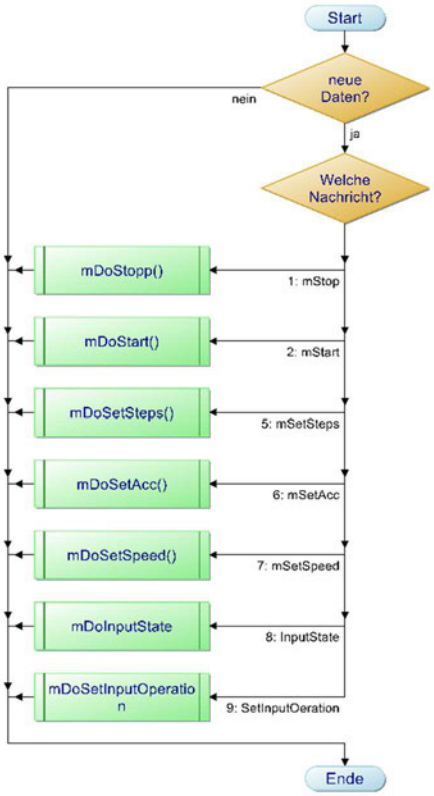


Abbildung 6-2: Ablaufplan der Funktion feedinSerialData()

6.2.3 Die Funktion Inputs()

Mit der Funktion Inputs() wird kontinuierlich der Status der Positionssensoren und Taster abgefragt. Abbildung 6-3 zeigt den Ablaufplan der Funktion. Für jeden einzelnen Input wird zunächst geprüft, ob dieser aktiv geschaltet ist. Wenn dies nicht der Fall ist, wird zum nächsten Input übergegangen. Durch die Deaktivierung der Abfrage von nicht genutzten Eingangssignalen kann Prozessorzeit eingespart werden. Des Weiteren wird so verhindert, dass das Auftreten eines Signals an einem unerwünschten Zeitpunkt den Programmablauf stört. Die Abfrage des Schalters für die Spannungsversorgung der Motortreiber ist permanent aktiv und kann nicht abgeschaltet werden. Ist ein Input aktiv, wird ein Unterprogramm aufgerufen, dass bei allen Tastern und Sensoren ähnlich aufgebaut ist.

Bei der Implementierung der Abfrage der Gabellichtschranken kam es zunächst zu Problemen. Aus den Vorversuchen war bekannt, dass ein einfaches Einlesen des Signalausgangs möglich ist (siehe Abschnitt 4.3.2). Bei der Umsetzung der Controller-Software für den vollständig aufgebauten Sensorteststand stellte sich jedoch heraus, dass der Mikrocontroller Signale von den Lichtschranken erfasst, obwohl jene nicht ausgelöst wurden. Elektrische Messungen ergaben, dass auf den Signalleitungen der Sensoren kurze Impulse auftreten, welche vom Mikrocontroller als Pegelwechsel interpretiert werden (siehe Abbildung 6-4). Die Ursache für diese Störsignale wurde nicht gefunden. Die Spannungsspitzen sind nur sehr kurz, weshalb als Problemlösung eine Zeiterfassung bei der Sensorabfrage hinzugefügt wurde. So werden nur Signale als valide bewertet, die länger als eine Mindestzeit am Eingang anliegen. Um die Abfrage der Sensoren nicht zu träge zu gestalten, wurde empirisch versucht, diese Mindestzeit möglichst kurz zu halten. Die Versuche ergaben, dass bei einer Verzögerungszeit von 2 ms über einen Zeitraum von 4 h kein Störsignal von der Software erfasst wurde. Für die Abfrage der Taster und des Motorschalters wurde der für die Gabellichtschranken entwickelte Programmcode übernommen. Die Verzögerungszeit zum Ausblenden der Störsignale eignete sich ebenso zum Entprellen der mechanischen Signalgeber. Basierend auf den Messungen in den Vorversuchen wurde eine Verzögerungszeit von 50 ms für die Abfrage gewählt.

Für den Fall, dass erkannt wird, dass das Signal eines Tasters oder Positionssensors aktiv geworden ist, wird ein weiteres Unterprogramm aufgerufen. Beim Start-Taster ist dies die Funktion Start(). Diese Funktion speichert bei ihrem Aufruf in einer Variable die Information, welcher Motor aktuell bewegt werden soll, und aktiviert die Funktion des Stopp-Tasters. Die Positionssensoren und der Stopp-Taster lösen bei ihrer Betätigung den Aufruf der Funktion Stopp() aus. Diese Funktion dient dazu, einen Motor augenblicklich anzuhalten. Hierfür wird die Variable zum Bewegen der Motoren zurückgesetzt. Des Weiteren werden die verblieben Motorschritte in der AccelStepper-Bibliothek gelöscht, der Stopp-Taster deaktiviert und eine Nachricht über den Abbruch der Bewegung an den Host-PC gesendet.

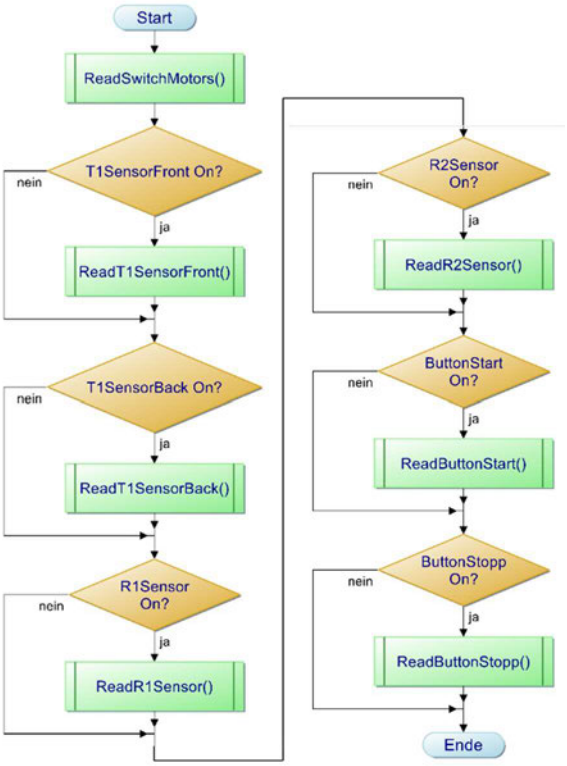


Abbildung 6-3: Ablaufplan der Funktion Inputs()

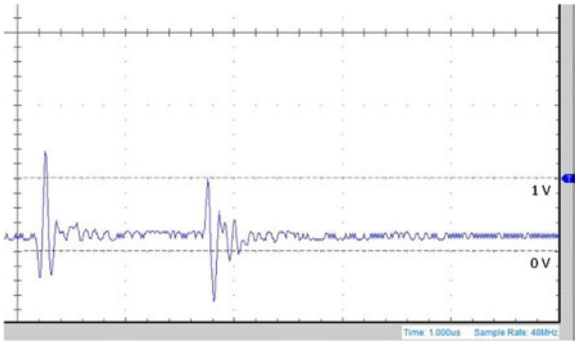


Abbildung 6-4: Störimpulse im Signal der Gabellichtschranken

6.2.4 Die Movement-Funktionen

Im dritten Teil der Hauptschleife werden die Funktionen MovementT1(), MovementR1() und MovementR2() aufgerufen. Die Funktionen sind der AccelStepper Bibliothek zugeordnet und dienen der Ansteuerung der Schrittmotoren des Sensorprüfstands. In Abbildung 6-5 ist der Ablaufplan der Funktion MovementT1() dargestellt, welche den Schrittmotor für den Linearantrieb steuert. Die anderen beiden Funktionen für die rotatorischen Achsen sind in ihrem Aufbau identisch zu MovementT1().

Zu Beginn der Funktion wird geprüft, ob der jeweilige Schrittmotor momentan eine Bewegung ausführen soll. Die Variable hierfür wird über den Start-Befehl gesetzt (siehe Abschnitt 6.2.3). Steht die Variable auf null, wird die Funktion wieder verlassen. Andernfalls wird der Befehl run() der AccelStepper-Bibliothek aufgerufen, welcher einen Motorschritt auslöst, sofern der Zeitpunkt dafür erreicht ist. Die Bibliothek errechnet die Zeitpunkte für die fälligen Schritte anhand des eingestellten Beschleunigungs- und Geschwindigkeitswerts, sowie der zu vollführenden Gesamtschrittzahl (siehe Abschnitt 3.2.2). Nach seiner Ausführung zeigt die Funktion run() mit ihrem Rückgabewert an, ob noch weitere Motorschritte folgen (Rückgabewert „wahr“), oder ob der letzte Schritt ausgeführt wurde (Rückgabewert „falsch“). Folgen noch weitere Schritte, wird die Funktion an dieser Stelle bis zum nächsten Aufruf verlassen. Wurde die Motorbewegung vollständig ausgeführt und beendet, werden weitere Programmschritte in der Funktion ausgeführt. Es wird die Vari-

able zurückgesetzt, durch welche zu Beginn abgefragt wird, ob sich der Motor in Bewegung befindet. An den Host-PC wird via CmdMessenger die Nachricht gesendet, dass die Achsbewegung vollständig abgeschlossen wurde. Die Funktion des Stopp-Tasters wird deaktiviert. Der in der AccelStepper-Bibliothek gespeicherte Positionswert des Motors wird zurückgesetzt, da die Speicherung der Position in der PC-Software erfolgt.

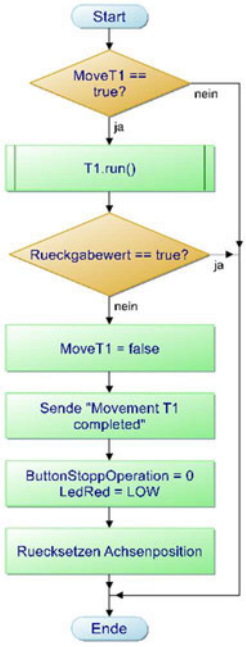


Abbildung 6-5: Ablaufplan für die Movement-Funktionen am Beispiel der Linearachse T1

6.2.5 Zusammenfassung zur Entwicklung der Controller-Software

Die zuvor beschriebenen Funktionsblöcke wurden als einzelne Einheiten entwickelt und danach in einem Programm zusammengefasst. Das Ziel war es, eine Steuerung für den Sensorprüfstand zu entwickeln, die die Ein- und Ausgangssignale der Sensoren und Taster verwaltet, die unmittelbare Ansteuerung der Schrittmotoren übernimmt und per serieller Kommunikationsschnittstelle Daten mit einem Host-PC austauscht. Die Controller-Software bildete die Grundlage für die darauffolgende Entwicklung der Software für den Host-PC. Um sicherzustellen, dass die Controller-Software vollumfänglich über die serielle Verbindung bedienbar ist, wurden alle möglichen Nachrichten des Kommunikationsprotokolls per Eingabe über den Serial Monitor getestet und verifiziert. Ab diesem Punkt war der Sensorprüfstand voll steuerbar durch die direkte Eingabe von Befehlen über die serielle Schnittstelle. Voraussetzung hierfür war die Kenntnis über die Verwendung des Kommunikationsprotokolls und Umrechnung der Motorschritte in Längenmaße, welche den tatsächlichen Bewegungen der Achsen entsprechen. Die folgenden Kapitel beschreiben die Entwicklung der PC-Software, mit deren Hilfe eine einfache Bedienung des Sensorprüfstandes per grafischer Benutzeroberfläche ermöglicht werden sollte. Für die Um-

setzung der PC-Software war es erforderlich, kleinere Änderungen an der Controller-Software vorzunehmen. Diese werden an den entsprechenden Stellen in den folgenden Abschnitten beschrieben. Der abschließend verwendete Programmcode für die Controller-Software ist im Anhang dieser Arbeit hinterlegt.

6.3 Entwicklung der PC-Software

6.3.1 Erprobung von pySerial, pyCmdMessenger und tkinter

Zu Beginn der Entwicklung der PC-Software wurden grundlegende Funktionalitäten getestet, die für die Umsetzung des Programms notwendig waren. Für Python sind fertige Module verfügbar, die die hier benötigten Funktionen bereitstellen. Im ersten Schritt der Umsetzung wurde getestet, wie sich ein Verbindungsaufbau zum Mikrocontroller realisieren lässt. Hierfür wurde das Python-Modul pySerial verwendet. Das Modul ermöglicht es, von Python aus einen Serial Port des PCs zu verwenden und Schreib- und Lesebefehle darüber auszuführen (Liechti 2015). Voraussetzung für einen Verbindungsaufbau zum Arduino Mikrocontroller ist es, dass dessen USB-Treiber im Betriebssystem des Computers installiert sind. Der Test des pySerial-Moduls ergab, dass ein Verbindungsaufbau problemlos möglich ist. In dem Modul wird der verwendete COM-Port und die Baud-Rate definiert, woraufhin über ein Terminal auf die gleiche Weise mit dem Mikrocontroller kommuniziert werden kann, wie über den Serial Monitor in der Arduino IDE.

Im nächsten Schritt wurde die Verwendung von pyCmdMessenger getestet. Es handelt sich hierbei um eine Python Klasse, die das Gegenstück zur CmdMessenger-Bibliothek im Arduino bildet. Die Klasse sendet die Datenpakete als binäre Argumente, weshalb die entsprechenden Datentypen aus dem Kommunikationsprotokoll im Arduino-Programmcode geändert werden mussten (Harms 2017). Nach der Anpassung der Datentypen war es möglich, Befehle per CmdMessenger zwischen Arduino und PC auszutauschen. Die Liste der Befehle aus dem Kommunikationsprotokoll muss auf beiden Plattformen stets identisch gehalten werden. Entgegen der Annahme während der Konzipierung des Projekts, verfügt die Klasse pyCmdMessenger über einen geringeren Funktionsumfang als die Arduino-Bibliothek. So fehlt z.B. die Möglichkeit der Verwendung von Callbacks in der Python Umgebung, was den Programmieraufwand erhöhte.

Ein weiterer wesentlicher Bestandteil des PC-Programms ist die grafische Benutzeroberfläche. Um eine solche erstellen zu können, sollte in diesem Projekt das Paket Tk interface, kurz tkinter verwendet werden. Tkinter ist das Standard Python Interface zum Tcl/Tk GUI Toolkit (Python Software Foundation 2021). Die Erstellung einer Benutzeroberfläche mit tkinter umfasst drei Hauptschritte: die Erzeugung eines Hauptfensters, die Anordnung von Schaltflächen und Eingabefeldern, genannt Widgets, und der Aufruf einer Endlosschleife, in der die Ereignisse in der Benutzeroberfläche erfasst und zugehörige Programmteile gestartet werden (Theis 2019, S. 371). Abbildung 6-6 zeigt eine erste Benut-

zeroberfläche, die zu Testzwecken mit tkinter erstellt wurde. Sie kann verwendet werden, um den Linearantrieb des Sensorteststands zu steuern, indem eine Schrittanzahl für den Motor und ein Start- bzw. Stoppbefehl gesendet wird. Das Testprogramm vereint somit alle Grundfunktionen, die für das finale Programm benötigt werden: eine Benutzeroberfläche mit Eingabefeldern und Buttons und die Funktionalität, die eingegebenen Befehle an die Motorsteuerung zu senden. Es war an dieser Stelle also bereits möglich, die Schrittmotoren vom Host-PC aus über eine Benutzeroberfläche zu steuern.

Abbildung 6-6: Testweise Erstellung einer Benutzeroberfläche mit tkinter

6.3.2 Berechnungen für die Bewegung der Achsen

Für eine anwenderfreundliche Steuerung des Sensorteststands ist es notwendig, innerhalb der Software Umrechnungen zwischen den Daten durchzuführen, die in der Benutzeroberfläche eingegeben und denen, die an die Motorsteuerung gesendet werden. Die Steuerung erhält Befehle vom Host-PC ausschließlich in der Maßeinheit von Motorschritten. In der Benutzeroberfläche werden die Positionier- und Bewegungsbefehle aber durch den Anwender in der Einheit Grad bzw. Meter angegeben.

Der Schlitten des Sensorteststands wird unter Angabe eines Absolutmaßes in Metern auf dem Verfahrweg der Trasse positioniert. Der zurückgelegte Weg des Schlittens pro Motorschritt des Linearantriebs errechnet sich aus dem wirksamen Durchmesser des Riemenantriebs und der Anzahl von Motorschritten pro Umdrehung.

$$Zurückgelegter\ Weg\ pro\ Motorschritt = \frac{Durchmesser\ Riemenantrieb \cdot \pi}{Schritte\ pro\ Motorumdrehung}$$

Der wirksame Durchmesser des Riemenantriebs wird an der neutralen Faser des Zahnriemens beim Umlauf um die Riemenscheibe gemessen. Die Zahlenwerte der Berechnungen des Linearantriebs sind in Tabelle 6-2 dargestellt. Die Höchstgeschwindigkeit des Schlittens wird durch die höchstmögliche Schrittzahl pro Zeit der Motorsteuerung limitiert (siehe Abschnitt 4.4.1).

$$Höchstgeschwindigkeit = Zurückgelegter\ Weg\ pro\ Motorschritt \cdot Höchstschrittzahl$$

Die maximal mögliche Beschleunigung des Schlittens hängt im Wesentlichen vom erzeugbaren Drehmoment des Schrittmotors und der zu bewegenden Last ab. Der Schlitten

selbst wiegt mit samt seinen Anbauten 5,8 kg. Hinzu kommt das Gewicht eines Prüfkörpers, welches mit bis zu 2 kg in der Aufgabenstellung spezifiziert wurde. Die für die Berechnung herangezogene Hebellänge entspricht der Hälfte des wirksamen Durchmesser des Riemenantriebs.

$$max.Beschleunigung = \frac{Drehmoment}{Hebellänge \cdot (Gewicht\ Schlitten + Gewicht\ Prüfkörper)}$$

Während der späteren Erprobung der Software wurde deutlich, dass der errechnete Wert von über 13 m/s² in der Realität nicht erreicht werden kann (siehe Abschnitt 6.3.4). Gründe hierfür liegen möglicherweise in einem abweichenden tatsächlichen Drehmoment des Schrittmotors und der Vernachlässigung von auftretenden Reibungskräften.

Die Positionierung der beiden rotatorischen Achsen wird aufgrund der identischen verwendeten Bauteile gleich berechnet. Die Umrechnung von Motorschritten auf ein Winkelmaß der Plattformen ergibt sich aus der verwendeten Getriebeübersetzung.

$$Winkel\ pro\ Motorschritt = \frac{360°}{Schritte\ pro\ Motorumdreheung \cdot Übersetzungsverhältnis}$$

Die Zahlenwerte der Berechnung der Drehachsen sind in Tabelle 6-3 zusammengefasst. Im Gegensatz zum Linearantrieb wurden bei den Drehachsen 1600 Mikroschritte für eine Motorumdrehung gewählt, um eine sanftere Achsbewegung zu erreichen (siehe Abschnitt 6.3.4). Für die Beschleunigung und die Geschwindigkeit sind keine Berechnungen notwendig, da hier konstante Werte im Programmablauf festgelegt wurden.

Wirksamer Durchmesser Riemenantrieb	0,09550 m
Schritte pro Motorumdrehung	400
Zurückgelegter Weg pro Motorschritt	**0,00075 m**
Höchstschrittzahl	4000 Steps/s
Höchstgeschwindigkeit	**3 m/s**
Drehmoment Motor	5 Nm
Hebellänge	0,04775 m
Masse Schlitten	5,8 kg
Masse Prüfkörper	2,0 kg
max. Beschleunigung	**13,42 m/s²**

Tabelle 6-2: Berechnungen für den Linearantrieb

Zähnezahl Ritzel	20
Zähnezahl Zahnkranz	139
Übersetzungsverhältnis	6,95
Schritte pro Motorumdrehung	1600
Winkel pro Motorschritt	**0,03237 °**

Tabelle 6-3: Berechnung für die Drehachsen

6.3.3 Initialisierung der Bewegungsachsen

Für eine Positionierung der Bewegungsachsen über die Benutzeroberfläche vom Host-PC muss in der Software bekannt sein, in welcher Stellung sich die Achsen befinden. Hierfür werden die Achsen initialisiert, indem sie einen bekannten Punkt anfahren, welcher durch die Schaltfahnen der Positionssensoren verkörpert wird. In Abbildung 6-7 ist dargestellt, wie die Bewegungsachsen des Sensorteststands in der Software definiert wurden. Der Schlitten bewegt sich translatorisch auf der Achse T1. Die Drehachse des Schlittens trägt Bezeichnung R1 und die der Sensorplattform R2. Die absolute Position von T1 ist durch den Abstand der Mittelpunkte der beiden Drehachsen definiert. Die Drehachsen liegen in ihrer Grundposition einander zugewandt jeweils auf dem Winkel von 0°. Sie können sich nach links auf -90° und nach rechts auf +90° drehen.

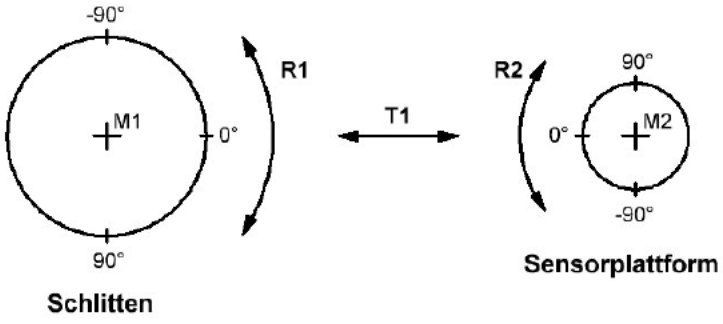


Abbildung 6-7: Definition der Bewegungsachsen

Die Software ist so konzipiert, dass die Eingabe von Positionierungsbefehlen nur möglich ist, wenn die entsprechende Achse initialisiert ist, also die tatsächliche Position einmal erfasst wurde. In der Benutzeroberfläche wird durch eine farbliche Kennzeichnung signalisiert, ob eine Bewegungsachse initialisiert ist und bereitsteht. Solange eine Achse nicht initialisiert ist, sind die Eingabefelder für die Steuerung der Achse deaktiviert. Beim Starten der Benutzeroberfläche muss grundsätzlich eine Initialisierung durchgeführt werden. Danach kann eine Initialisierung jederzeit bei Bedarf wiederholt werden.

Zur Initialisierung der Drehachsen werden diese so lange nach links bewegt, bis vom Controller das Signal gesendet wird, dass der Positionssensor ausgelöst wurde. Von dieser Position aus werden die Drehachsen auf ihre Home-Position gefahren, welche per Definition 0° entspricht. Der Programmablauf zur Steuerung von Bewegungsbefehlen zwischen Host-PC und Controller wird im anschließenden Abschnitt 6.3.4 beschrieben. Der Winkel zwischen dem Positionssensor und der Home-Position wird empirisch ermittelt und kann als Parameter in einer separaten Parameter-Datei hinterlegt werden. Nach Erreichen der Home-Position werden die Eingabefelder für die Positionierung der Achse aktiv geschaltet. Die Positionssensoren der Drehachsen werden im Normalbetrieb nicht benötigt und daher nach einer erfolgreichen Initialisierung deaktiviert.

Die Initialisierung der translatorischen Achse verläuft ähnlich, wie die der Drehachsen. Der Schlitten wird in Richtung der vorderen Schaltfahne bewegt, bis der zugehörige Sensor auslöst und der Controller das Ereignis zurückmeldet. Befindet sich der Schlitten zu Beginn der Initialisierung bereits so weit vorn, dass der Sensor schon ausgelöst ist, wird der Schlitten zunächst 40 mm zurück und danach wieder auf die Schaltfahne zubewegt. Nachdem der Sensor auslöst, wird der Schlitten auf seine Home-Position gefahren. Die Position ist als Abstand zur Schaltfahne definiert und wird in der Parameter-Datei hinterlegt. Die Home-Position verkörpert den geringsten Abstand zur Sensorplattform, den der Schlitten im Normalbetrieb anfahren kann. In Abbildung 6-8 ist nachvollziehbar, aus welchen Maßen sich die Absolutposition des Schlittens berechnet. Die Home-Position wurde mit einem Abstand von 40 mm zur vorderen Schaltfahne (C) gewählt und der Parameter-Datei hinterlegt. Dort wird außerdem angegeben, wie lang der mögliche Fahrweg des Schlittens sein darf. Er wurde für das Projekt auf 2 200 mm festgelegt, wodurch sich rechnerisch ein Abstand von 47 mm zur hinteren Schaltfahne (B) ergibt. Das Konzept für den Sensorteststand sieht vor, dass die Endlagensensoren des Linearantriebs im Normalbetrieb nie erreicht werden. Wird durch eine Fehlfunktion dennoch einer der Sensoren ausgelöst, wird der Motor des Linearantriebs gestoppt und die Achse muss neu initialisiert werden. Das Maß der Absolutposition des Schlittens auf der Trasse ist als Abstand zwischen den Mittelpunkten der beiden Drehachsen definiert. Die Angabe der Position des Schlittens kann in der Parameter-Datei geändert werden. Dort wird über einen Parameter angeben, welchem Absolutwert die Home-Position entspricht.

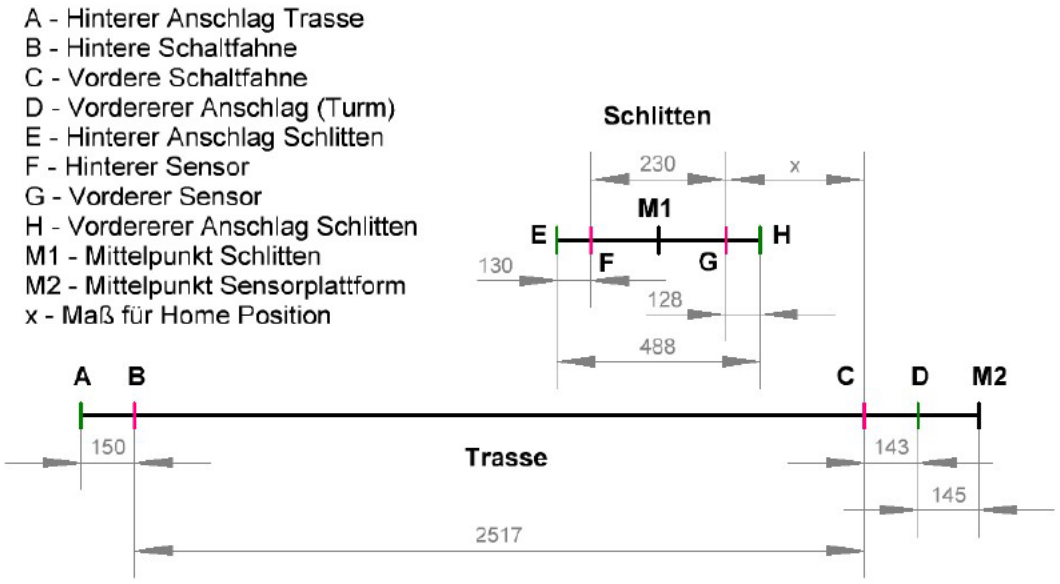


Abbildung 6-8: Maße für die Positionierung des Schlittens in Millimeter

6.3.4 Steuerung der Bewegungsachsen über die Benutzeroberfläche

Die Bewegungen der Achsen des Sensorteststands werden mittels der Benutzeroberfläche gesteuert, indem von der PC-Software Positionseingaben in erforderliche Motorschritte umgerechnet und an den Controller übertragen werden. Die Drehachsen können auf einen Winkel von -90° bis +90° gedreht und die translatorische Achse auf einer Fahrstrecke von 2,2 m bewegt werden. Für die translatorische Achse ist es zusätzlich möglich, die Geschwindigkeit und Beschleunigung über die Benutzeroberfläche vorzugeben. Solange sich eine Achse in Bewegung befindet, kann diese durch eine Betätigung des Stopp-Tasters am Bedienpult des Sensorteststands zu einem Halt gebracht werden.

Für die Positionierung der Drehachsen wird in der Benutzeroberfläche der gewünschte Zielwinkel eingetragen und mit der nebenstehenden Schaltfläche bestätigt. Liegt der Zielwinkel außerhalb des definierten Bereichs oder stimmt er mit dem aktuellen Winkel überein, wird dies in einer Meldung auf der Benutzeroberfläche mitgeteilt. Ist der Zielwinkel valide, wird die Differenz zum aktuellen Winkel gebildet und diese in Motorschritte umgerechnet (siehe Abschnitt 6.3.2). Die berechneten Motorschritte werden an den Controller übertragen und die Achsbewegung gestartet. Nach dem Start der Bewegung wartet das Programm auf eine Rückmeldung des Controllers über die Beendigung der Bewegung. Wurde die Bewegung wegen einer Betätigung des Stopp-Tasters am Bedienpult beendet, wird ein Abbruch der Bewegung gemeldet und die Achse muss neu initialisiert werden (siehe Abschnitt 6.3.3). Sendet der Controller die Rückmeldung, dass die Bewegung vollständig durchgeführt wurde, wird dies in der Benutzeroberfläche gemeldet. Der Winkel der neuen Position wird gespeichert und neben dem Eingabefeld für die Drehachse angezeigt. Während der Erprobung der Software viel auf, dass sich die Drehachse auf dem Schlitten teilweise ungleichmäßig bewegt. Es wurde versucht dem entgegenzuwirken, indem die Teilschritte für eine Motorumdrehung erhöht wurden. Eine Erhöhung von 400 auf 1 600 Mikroschritte erzielte eine leichte Verbesserung. Die Drehachsen werden grundsätzlich mit einer Geschwindigkeit von 800 Steps/s und einer Beschleunigung von 2 000 Steps/s² sowohl bei der Initialisierung als auch im Normalbetrieb bewegt. Die Werte sind in der Parameter-Datei hinterlegt und werden während der Initialisierung der Drehachse an den Controller übertragen.

Die Steuerungsmöglichkeiten für die Bewegung des Schlittens auf der translatorischen Achse sind unter Berücksichtigung der Aufgabenstellung umfangreicher. Es kann sowohl die Position als auch die gewünschte Geschwindigkeit und Beschleunigung in der Benutzeroberfläche eingestellt werden. Die Maximalwerte für die Geschwindigkeit und die Beschleunigung sind in der Parameter-Datei hinterlegt (siehe Abschnitt 6.3.2). Im Gegensatz zum errechneten Wert hat sich in der Praxis gezeigt, dass nur ein maximaler Beschleunigungswert von 4 m/s² sinnvoll ist. Bei hören Werten weist der Linearantrieb ein fehlerhaftes Abbremsverhalten auf. Um eine hohe Genauigkeit zu erreichen, wurde für die Initialisierungsfahrt der translatorischen Achse eine niedrige Geschwindigkeit in Höhe von 0,022 m/s gewählt. Dieser Wert kann in der Parameter-Datei geändert werden. Im Normalbetrieb werden die gewünschte Geschwindigkeit und Beschleunigung in der Benut-

zeroberfläche eingeben, woraufhin diese in Steps/s bzw. Steps/s² umgerechnet und an den Controller gesendet werden. Der Schlitten wird bewegt, indem in der Benutzeroberfläche eine neue Zielposition eingegeben wird. Die Software überprüft auf Grundlage des Weg-Zeit- und des Weg-Geschwindigkeits-Gesetzes, ob bei der resultierenden Fahrstrecke von der aktuellen Position zur Zielposition und der gewählten Beschleunigung die eingestellte Endgeschwindigkeit erreicht werden kann. Ist dies nicht der Fall, wird über die Benutzeroberfläche ein Hinweis ausgegeben mit der maximal erreichbaren Geschwindigkeit und der nötigen Beschleunigung zur Erreichung der eingestellten Geschwindigkeit. Der Anwender kann darüber entscheiden, ob er die Fahrt trotzdem durchführen oder andere Werte verwenden möchte. Die Schlittenfahrt kann wahlweise direkt von der Benutzeroberfläche aus oder vom Bedienpult gestartet werden. Für Zweiteres wird eine entsprechende Schaltfläche angewählt, woraufhin der Host-PC den Start-Taster am Bedienpult aktiv schalten lässt. Nach dem Start der Bewegung erfolgt, wie bei den Drehachsen vom Controller, die Rückmeldung, ob die Bewegung vollständig abgeschlossen oder abgebrochen wurde. Je nach Ausgang wird in der Oberfläche die neue Position gespeichert und angezeigt oder eine erneute Initialisierung angefordert.

Das Design der Software mit der gleichzeitigen Verwendung einer Endlosschleife zum Aktualisieren der Benutzeroberfläche und dem Warten auf eine Rückmeldung des Controllers während der Bewegung einer Achse führt zu einer Einschränkung bei der Bedienbarkeit. Solange eine Motorbewegung durchgeführt wird und eine Rückmeldung vom Controller noch nicht erfolgt ist, reagiert die Benutzeroberfläche auf keine neuen Eingaben. Das Zeit-Budget für die Umsetzung des Projekts reichte nicht aus, um diese Einschränkung in der Bedienbarkeit abzustellen. Aus diesem Grund musste die geplante Implementierung einer Schaltfläche in der Benutzeroberfläche zum Stoppen von Achsbewegungen verworfen werden. Die Stopp-Funktion ist weiterhin jederzeit durch die Betätigung des Stopp-Tasters am Bedienpult des Sensorteststands ausführbar.

6.3.5 Zusammenfassung zum Funktionsumfang der PC-Software

Die PC-Software wurde mit dem Ziel entwickelt, eine einfache Bedienung des Sensorteststands zu ermöglichen. In Abbildung 6-9 ist die fertige Benutzeroberfläche dargestellt. Die Hauptfunktion des Programms ist es, Benutzereingaben zur Bewegung der Antriebsachsen in die korrekten Motorbewegungen umzurechnen, diese an den Controller zu übertragen und die Position der Achsen zu speichern und anzuzeigen. Zusätzlich zu dem Programm existiert eine Parameter-Datei, in der Änderungen eingetragen werden können, die die mechanischen Eigenschaften des Sensorteststands betreffen. Dies dient hauptsächlich der Möglichkeit, die Anlage zu einem späteren Zeitpunkt modifizieren zu können.

Beim Start der Benutzeroberfläche wird automatisch eine Verbindung zum Controller des Sensorprüfstands aufgebaut. Der vom Betriebssystem verwendete COM-Port kann in der Parameterdatei angegeben werden. Bevor beliebige Achsbewegungen durchgeführt werden können, muss jede Achse initialisiert werden, um in der PC-Software eine Ausgangs-

position zu erfassen. Von da an werden die Achspositionen nicht mehr direkt gemessen, sondern durch Zählen der Motorschritte umgerechnet. In Abschnitt 7.3 wird überprüft, welche Genauigkeit der Positionierung der Achsen mit diesem Verfahren erreicht werden kann. Die Drehachsen können unter Angabe eines Winkels mit einer fest vorgegebenen Geschwindigkeit und Beschleunigung bewegt werden. Die translatorische Achse wird unter Angabe von Zielposition, Beschleunigung und Geschwindigkeit in einem definierten Bereich bewegt. Das Programm berechnet, ob bei den angeforderten Fahrwegen die eingestellten Parameter eingehalten werden können. Ein Meldungsfeld liefert Informationen zur Richtigkeit von Eingaben und erfolgten Bewegungen von Achsen. Der Programmcode für die PC-Software ist im Anhang dieser Arbeit hinterlegt.

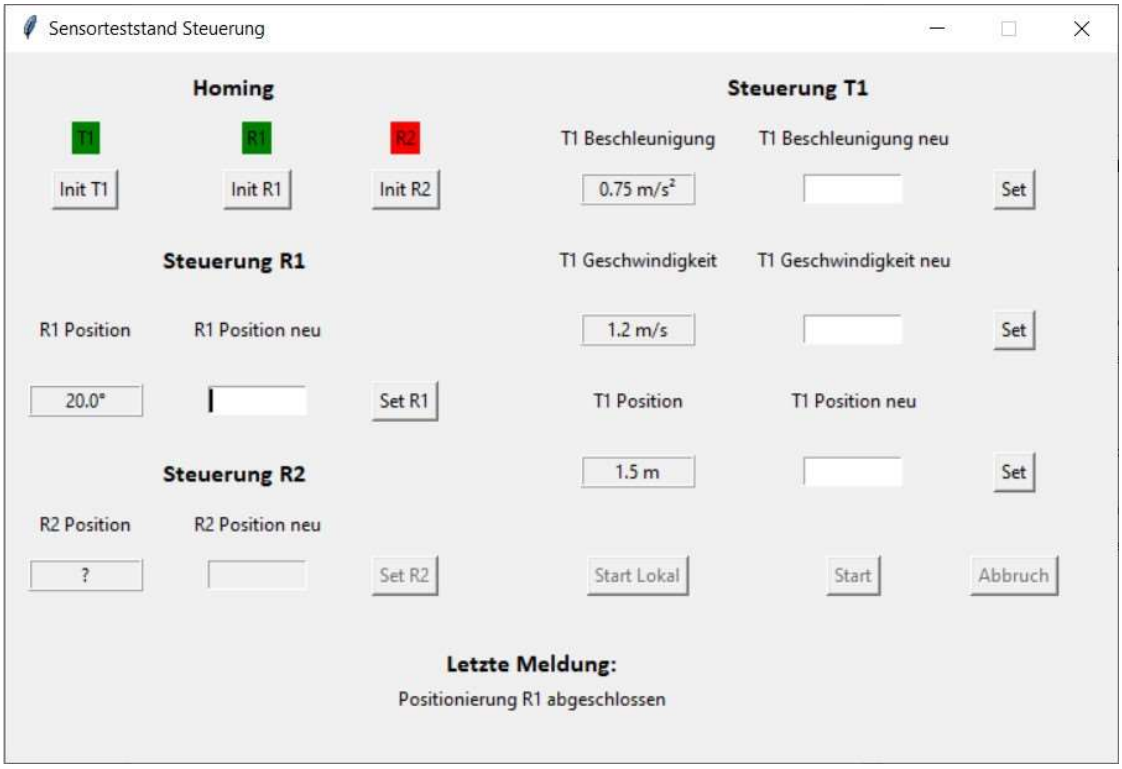


Abbildung 6-9: Ansicht der fertigen Benutzeroberfläche

7 Überprüfung des Sensorteststands

Dieses Kapitel befasst sich mit der Untersuchung, wie sich der Sensorteststand in der Praxis verhält. Zu Beginn wird dargestellt welche Arten von Messungen dafür in Frage kommen. Im weiteren Verlauf des Kapitels wird erörtert, welche dieser Messungen durchgeführt werden sollen, da nicht alle Messungen gleich sinnvoll oder möglich sind. Da der Sensorteststand über kein Wegemesssystem verfügt, war von besonderem Interesse, inwieweit die Eingaben in der Benutzeroberfläche mit den tatsächlichen Bewegungen übereinstimmen. Im letzten Abschnitt werden die Ergebnisse der durchgeführten Untersuchungen vorgestellt.

7.1 Möglichkeiten der Überprüfung

Es gibt eine Vielzahl von Eigenschaften, die im Hinblick auf die Verwendung des Sensorteststands von Interesse sein können. In Tabelle 7-1 wurde zusammengetragen, welche Eigenschaften und zugehörige Messgrößen in Frage kommen und welche Messmittel zur Ermittlung der Eigenschaften verwendet werden könnten. Die möglichen Messgrößen unterscheiden sich in ihrer Relevanz und nicht alle ermittelten Mess- und Prüfmittel standen für das Projekt zur Verfügung. Im nächsten Abschnitt wird anhand von Entscheidungskriterien eine Auswahl getroffen, welche Messungen und Prüfungen durchgeführt werden.

Eigenschaft	**Messgrößen**	**Mess- und Prüfmittel**
Lage	Entfernung, Winkel, Positioniergenauigkeit, Wiederholgenauigkeit	Messschieber, Anschlagwinkel, Winkelmesser, Gliedermaßstab, Bandmaß, Lineal, Winkelschablone, Ultraschallsensor
Bewegung	Geschwindigkeit, Beschleunigung	Beschleunigungssensor, Tachometer, Ultraschallsensor
Elektrische Eigenschaften	Strom, Spannung, Leistung	Multimeter, Oszilloskop
Temperatur	Erwärmung von Motor und Motortreiber	Thermometer

Tabelle 7-1: Messbare physikalische Eigenschaften des Sensorteststands

7.2 Auswahl der Mess- und Prüfmethoden

Die messbaren Eigenschaften des Sensorteststands lassen sich in zwei grobe Kategorien einteilen. Die Messung der Lage und Bewegung der Achsen gibt Auskunft über die Genauigkeit von deren Ansteuerung und weist nach, ob die in der Benutzeroberfläche eingegebenen Werte mit den tatsächlich ausgeführten Bewegungen übereinstimmen. Die elektrischen Eigenschaften sowie das Temperaturverhalten der elektronischen Komponenten und der Schrittmotoren sind für den Betreib der Anlage von Bedeutung.

Bei den Lageeigenschaften der Bewegungsachsen sind die Positionier- und die Wiederholgenauigkeit ausschlaggebend. Diese lassen sich mit einfachen Mitteln überprüfen, z.B. mit einem Längenmessgerät, wie einem Lineal oder einem Gliedermaßstab. Der Vergleich der tatsächlichen Beschleunigung und Geschwindigkeit des Schlittens gegenüber den eingegebenen Werten in der Benutzeroberfläche ist ebenfalls ein sinnvolles Prüfkriterium. Allerdings musste auf die dafür erforderlichen Messungen verzichtet werden, da die notwendigen Messgeräte in dem Projekt nicht zur Verfügung standen.

Die zweite Kategorie zur Messung des Betriebsverhaltens der Anlage wurde von geringerer Relevanz gegenüber der der ersten Kategorie bewertet. Bereits während der Erprobung des Sensorteststands wurden Eigenschaften wie Stromaufnahme und Temperaturverhalten beobachtet und diese verhielten sich unauffällig. Auf Grund ihrer einfachen Durchführbarkeit wurden dennoch Messungen in diesem Bereich durchgeführt.

7.3 Durchführung und Auswertung der Überprüfungen

Die Überprüfung der Positionier- und Wiederholgenauigkeit der Drehachsen wurde stellvertretend für beide Achsen an der Achse R1 durchgeführt. In den Tests wurden die Drehwinkel, welche in der Benutzeroberfläche eingestellt wurden, mit den tatsächlich angefahrenen Winkeln verglichen. Für die Kontrolle der Winkel wurde eine Schablone angefertigt, auf der der Winkel in 45° Schritten von -90° bis +90° abgetragen wurden. Ein exaktes Messen von Winkeln war so nicht möglich, aber der große Radius der Drehachse von 28 cm bietet die Möglichkeit einer guten näherungsweisen Abschätzung. Ein Winkel von 1° entspricht bei diesem Radius einer Bogenlänge von 0,5 cm. So konnte eine qualitative Aussage darüber getroffen werden, ob die Achsposition mit der Markierung der Schablone übereinstimmt, oder ob eine Abweichung von kleiner oder größer 1 ° vorliegt. Tabelle 7-2 zeigt die Ergebnisse der Untersuchungen. Zwischen den einzelnen Versuchen wurde die Achse jedes Mal initialisiert. Das einmalige und wiederholte Anfahren bestimmter Positionen wies eine hohe Genauigkeit auf. Im Rahmen der Messgenauigkeit wurden alle Positionen exakt angefahren. Bei dem Test mit dem Verfahren kleiner Winkel hintereinander über einen großen Bereich trat am Ende eine deutliche Abweichung von schätzungsweise 1° auf. Dies könnte in Ungenauigkeiten durch Rundungsfehler begründet liegen, die bei der Umrechnung von Grad in Motorschritte entstehen und sich bei dieser Art der Verfahrweise aufsummieren.

Test	Ergebnis
Anfahren der Positionen -90°; -45°; 0°; 45° und 90°	Positionen Deckungsgleich mit Winkelschablone
Fortbewegung in 5° Schritten von der Position -90° nach 90°	Positionsabweichung von ca. 1° nach Fahrweg von 180°
Wiederholtes Anfahren der Positionen -90° und 90°	Endposition Deckungsgleich mit Winkelschablone nach 10 Wiederholungen

Tabelle 7-2: Überprüfung der Positionier- und Wiederholgenauigkeit der Achse R1

Die Überprüfung der Positionier- und Wiederholgenauigkeit der translatorischen Achse T1 erfolgte mit Hilfe eines Lineals für kürzere und einem Gliedermaßstab für längere Abstände. Beide Längenmessgeräte besitzen eine Skalenteilung von 1 mm, wodurch sich die Messgenauigkeit definiert. Für die Versuche wurden zwei unterschiedliche Geschwindigkeitsprofile definiert. Ein Langsames mit einer Geschwindigkeit von 0,1 m/s und einer Beschleunigung von 0,2 m/s² und ein Schnelles mit ein er Geschwindigkeit von 2 m/s und einer Beschleunigung von 3 m/s². Die Positioniergenauigkeit wurde ausschließlich mit der niedrigen Geschwindigkeit überprüft, während die Wiederholgenauigkeit sowohl mit der langsamen als auch mit der hohen Geschwindigkeit überprüft wurde. In Tabelle 7-3 sind die Ergebnisse der Versuche mit der translatorischen Achse aufgelistet. Es zeigte sich hier ein ähnliches Verhalten wie bei der Drehachse. Die Positionier- und Wiederholgenauigkeit ist in Hinblick auf die Anforderungen sehr hoch in einem Toleranzbereich von 1 mm. Eine hohe Geschwindigkeit des Schlittens beeinträchtigt die Genauigkeit nicht negativ. Die Positioniergenauigkeit nach dem Aneinanderreihen vieler kurzer Einzelfahrten nimmt auch hier ab, da sich die Einzelfehler bei der Positionierung aufsummieren.

Test	Ergebnis
Anfahren der Positionen 0,5 m; 1 m; 1,5 m und 2 mit niedriger Geschwindigkeit	Abweichung max. 1 mm
Fortbewegung in 5 cm Schritten von der Position 0,05 m auf 1 m mit niedriger Geschwindigkeit	Abweichung von 3 mm an Endposition
Wiederholtes Anfahren der Positionen 0,1 m und 1 m mit niedriger Geschwindigkeit	Keine Abweichung messbar nach 10 Wiederholungen
Wiederholtes Anfahren der Positionen 0,1 m und 2 m mit hoher Geschwindigkeit	Abweichung von 1 mm nach 10 Wiederholungen

Tabelle 7-3: Überprüfung der Positionier- und Wiederholgenauigkeit der Achse T1

Wie im letzten Abschnitt erwähnt, wurden auch Messungen zum Betriebsverhalten des Sensorteststands durchgeführt. Die Steuerung der Anlage, welche sich in der Hauptverteilung befindet, wird über den USB-Port des Host-PCs mit elektrischer Energie versorgt. Zur Messung der Stromaufnahme wurde ein USB-Tester verwendet (siehe Abbildung 7-1). Die maximale Stromaufnahme des Controllers ist in dem Szenario zu erwarten, wenn die LEDs des Start- und des Stopp-Tasters leuchten. Die Messung ergab hier einen Stromfluss von 110 mA bei einer Spannung von 4,99 V. Somit ist die Energieversorgung über einen USB-Port problemlos möglich. Zusätzlich wurde im Inneren des Schaltschranks die Erwärmung gemessen, welche durch den Betrieb des Controllers hervorgerufen wird. Diese betrug 21,2 °C gegenüber einer Raumtemperatur von 20,2 °C. Die Stromaufnahme der Schrittmotoren verhält sich sehr dynamisch. Bei einer Versorgungsspannung von 30 V wurden Höchstwerte von rund 2 A gemessen. Dies entspricht einem Leistungsbereich, der von einem Labornetzteil bereitgestellt werden kann.

Abbildung 7-1: USB-Tester zum Messen von Strom und Spannung (UM25C Messgerät | Joy-IT 2023)

8 Ergebnisse und Ausblick

Im abschließenden Kapitel werden die wesentlichen Ergebnisse der Arbeit zusammengefasst und ein Ausblick darauf gegeben, auf welche Weise der Sensorteststand verbessert und erweitert werden kann. Die Anforderungen der Aufgabenstellung wurden erfüllt, dennoch sind während der Bearbeitung weitere Ideen entstanden, deren Umsetzung über den gesetzten Rahmen des Projekts hinausgingen.

8.1 Ergebnisse der Arbeit

Zur Bearbeitung der Aufgabenstellung wurde das Projekt in mehrere Schritte unterteilt. Zu Beginn wurden theoretische Kenntnisse zum Themengebiet erworben und ein Konzept für die Umsetzung der Aufgabenstellung entworfen. Für das Konzept wurden die Anforderungen an den Sensorteststand in konkrete Arbeitsschritte und Teilgebiete übertragen. Das vorhandene Material wurde gesichtet und es wurden weitere benötigte Komponenten ermittelt. Nach der Konzepterstellung wurden vorbereitende Maßnahmen in Form von Vorversuchen und Materialbeschaffung durchgeführt. Die Arbeiten für die Umsetzung des Konzepts unterteilten sich in mechanische und elektrische Arbeiten am Sensorteststand sowie die Programmierung der Controller- und der PC-Software. Zum Abschluss wurden Versuche durchgeführt, in denen die erreichten Eigenschaften der Anlage überprüft wurden.

Die Einschätzung der Ergebnisse der Arbeit lautet, dass alle Anforderungen aus der Aufgabenstellung umgesetzt und erfüllt wurden. Der Sensorteststand wurde vollständig errichtet und lässt sich bei Bedarf mit einfachen Mitteln ab- und wieder aufbauen. Die Anlage verfügt über drei voll funktionsfähige Bewegungsachsen. Eine Achse ermöglicht die Drehung des zu testenden Sensors, die zwei weiteren Achsen bewirken die translatorische und rotatorische Bewegung von Prüfkörpern. Die Bedienung des Sensorteststands erfolgt von einer Benutzeroberfläche aus über einen angeschlossenen PC. Von dort können Winkel für die Drehachsen und Entfernungen mit Bewegungsprofilen für die translatorische Achse eingeben werden. Die Benutzeroberfläche visualisiert getätigte Eingaben und die Positionen der Bewegungsachsen. Eingegebene Bewegungsprofile werden vom Programm auf ihre Verfahrbarkeit überprüft. Ist das eingebebene Bewegungsprofil nicht umsetzbar, werden Vorschläge für mögliche Beschleunigungs- oder Geschwindigkeitswerte angezeigt. Über Taster am vorhandenen Bedienpult des Sensorteststands können sämtliche Achsbewegung gestoppt und bei Bedarf die Bewegung der translatorischen Achse gestartet werden. Die mit dem Bedienpult verbundene Hauptverteilung verfügt über Anschlussbuchsen für die Spannungsversorgung der Motoren und der USB-Verbindung

zum Controller. Durch eine Designänderung konnte auf die Notwendigkeit einer externen Spannungsversorgung mit 5 V verzichtet werden.

Die Benutzeroberfläche verfügt über einen geringeren Funktionsumfang als ursprünglich im Konzept angedacht. Auf einige Zusatzfunktionen wurde verzichtet, da sie im vorhandenen Zeitrahmen nicht umgesetzt werden konnten (siehe hierzu Abschnitt 8.2). Die Bediensoftware ist parametrierbar, wodurch es möglich ist, die Anlage hardwareseitig zu modifizieren, ohne dass neue Software geschrieben werden muss.

In der abschließenden Überprüfung des Sensorteststands wurde nachgewiesen, dass die Positionierbarkeit der Bewegungsachsen hinreichend genau ist. Ein Nachweis über die realen Beschleunigungs- und Geschwindigkeitswerte der translatorischen Achse erfolgte nicht. Die thermischen und elektrischen Eigenschaften der Anlage genügen den Anforderungen für den Einsatz über einen längeren Zeitraum. Die Schaltunterlagen der Anlage sowie der Programmcode für die Controller- und PC-Software sind im Anhang dieser Arbeit hinterlegt.

8.2 Ausblick

Der Umfang der Benutzeroberfläche könnte noch um einige Funktionen erweitert werden. Zunächst wäre es sinnvoll, die Programmierung dahingehend zu verbessern, dass die Software auch auf Eingaben reagiert, während eine Bewegungsachse verfährt. Die folgenden Funktionen werden als sinnvolle Erweiterungen zum jetzigen Versionsstand erachtet:

- Auswahlmöglichkeit des COM-Ports über ein Drop-Down Menü
- Anzeige möglicher Wertebereiche an den Eingabefeldern
- Jog-Mode zum Verfahren der translatorischen Achse
- Zähler von kurzen Achsenbewegungen zum Triggern einer erneuten Initialisierung
- Grafische Visualisierung der Achsstellungen

Bei der Bedienung des Sensorteststands hat sich gezeigt, dass eine Betriebsleuchte am Bedienpult zur Anzeige der Bereitschaft des Controllers wünschenswert wäre. Außerdem sollte eine Schaltung implementiert werden anhand derer der Controller erkennt, dass die Spannungsversorgung der Motoren unterbrochen wurde. So könnte über die Benutzeroberfläche eine erneute Initialisierung der Achsen veranlasst werden. Die Messung der Beschleunigung und Geschwindigkeit der Linearachse sollte nachgeholt werden und Korrekturfaktor für ggf. notwendige Anpassungen in der Software implementiert werden. Wie schon in der Aufgabenstellung erwähnt, kann der Verfahrweg der Linearachse noch verlängert werden. Der Antriebsmotor könnte dann durch ein Modell mit mehr Drehmoment ersetzt werden. Es wäre auch zu überlegen, ob ein leistungsstärkerer Controller statt des eingesetzten Arduino Mega verwendet werden sollte, um steuerungsseitig höhere Motor-

geschwindigkeiten zu erreichen. Dies würde aber auch tiefgreifende Änderungen in der Programmierung mit sich ziehen. Die Konstruktion des Schlittens könnte grundlegend überarbeitet werden, um eine höhere Robustheit zu erzielen. Nachdem der Sensorteststand im Anschluss an das Projekt zum Einsatz kommt, wird es möglich, weitere Erkenntnisse zu Verbesserungsmöglichkeiten zu sammeln.

Literaturverzeichnis

ACT Motor (2017a): HBS57. Datenblatt. Online verfügbar unter https://www.reichelt.de/index.html?ACTION=7&LA=3&OPEN=0&INDEX=0&FILENAME=X200%2FACT-HBS57INSTRUCTIONS.pdf, zuletzt geprüft am 23.07.2022.

ACT Motor (2017b): HBS86H. Datenblatt. Online verfügbar unter https://cdn-reichelt.de/documents/datenblatt/X200/THEINSTRUCTIONOFHBS86HACT_201909231013531.DOCX.pdf, zuletzt geprüft am 23.07.2022.

AirSpayce (2022): AccelStepper library for Arduino. Unter Mitarbeit von Mike McCauley. Online verfügbar unter https://www.airspayce.com/mikem/arduino/AccelStepper/, zuletzt geprüft am 22.08.2022.

Arduino (2022): CmdMessenger. Online verfügbar unter https://www.arduino.cc/reference/en/libraries/cmdmessenger/, zuletzt geprüft am 27.08.2022.

Austin, David (2005): Generate stepper-motor speed profiles in real time. Online verfügbar unter http://web.archive.org/web/20140705143928/http://fab.cba.mit.edu/classes/MIT/961.09/projects/i0/Stepper_Motor_Speed_Profile.pdf, zuletzt geprüft am 23.08.2022.

Brecher, Christian; Weck, Manfred (2021): Werkzeugmaschinen Fertigungssysteme 3. Berlin, Heidelberg: Springer Berlin Heidelberg.

Curious Scientist (2019): Arduino with TB6600 using AccelStepper library. Online verfügbar unter https://curiousscientist.tech/blog/arduino-accelstepper-tb6600-walkthrough, zuletzt geprüft am 28.01.2023.

doctek (2022): Using the Arduino AccelStepper Library. Online verfügbar unter https://hackaday.io/project/183713-using-the-arduino-accelstepper-library, zuletzt geprüft am 28.01.2023.

Faulhaber (2020): Technische Mitteilung zu Schrittmotoren: Fakten und Mythen zum Mikroschrittbetrieb. Online verfügbar unter https://www.faulhaber.com/fileadmin/user_upload_global/support/MC_Support/Motors/Tutorials_Motors/dff_200276_whitepaper_microstepping_de_4cb.pdf, zuletzt geprüft am 16.07.2022.

Harms, Mike (2017): PyCmdMessenger. Online verfügbar unter https://github.com/harmsm/PyCmdMessenger, zuletzt geprüft am 26.02.2023.

LAPP Kabel: Auswahl Technische Tabellen. Online verfügbar unter https://asset.conrad.com/media10/add/160267/c1/-/de/000608864IN01/information-608864-lapp-1026712-1-schleppkettenleitung-oelflex-chain-809-7-g-075-mm-grau-meterware.pdf, zuletzt geprüft am 02.08.2022.

Liechti, Chris (2015): pySerial 3.0 documentation. Online verfügbar unter https://pythonhosted.org/pyserial/, zuletzt geprüft am 25.02.2023.

Microchip Technology Inc (2005): ATmega640/1280/1281/2560/2561 Datasheet. Online verfügbar unter http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf, zuletzt geprüft am 30.07.2022.

Python Software Foundation (2021): tkinter — Python interface to Tcl/Tk, zuletzt geprüft am 26.02.2023.

Python Software Foundation (2023): General Python FAQ. Online verfügbar unter https://docs.python.org/3/faq/general.html#what-is-python-good-for, zuletzt geprüft am 23.01.2023.

Roddeck, Werner (2019): Einführung in die Mechatronik. 6., überarb. u. erw. Auflage 2019. Wiesbaden: Springer Fachmedien Wiesbaden (Springer eBook Collection).

SPS Magazin (2021): Spezifikationen von 2-Phasen- und 5-Phasen-Schrittmotoren im Vergleich. Online verfügbar unter https://www.sps-magazin.de/antriebstechnik/detaillierte-betrachtung-empfohlen/3/?onepage=1, zuletzt geprüft am 14.07.2022.

Steyer, Ralph (2018): Programmierung in Python. Ein Kompakter Einstieg Für Die Praxis. Wiesbaden: Vieweg. Online verfügbar unter https://link.springer.com/content/pdf/10.1007/978-3-658-20705-2.pdf?pdf=button.

Theis, Thomas (2019): Einstieg in Python. 5., aktualisierte Auflage 2017, 3., korrigierter Nachruck 2019. Bonn: Rheinwerk Verlag (Rheinwerk Computing).

UM25C Messgerät | Joy-IT (2023). Online verfügbar unter https://joy-it.net/de/products/JT-UM25C, zuletzt aktualisiert am 20.03.2023, zuletzt geprüft am 20.03.2023.

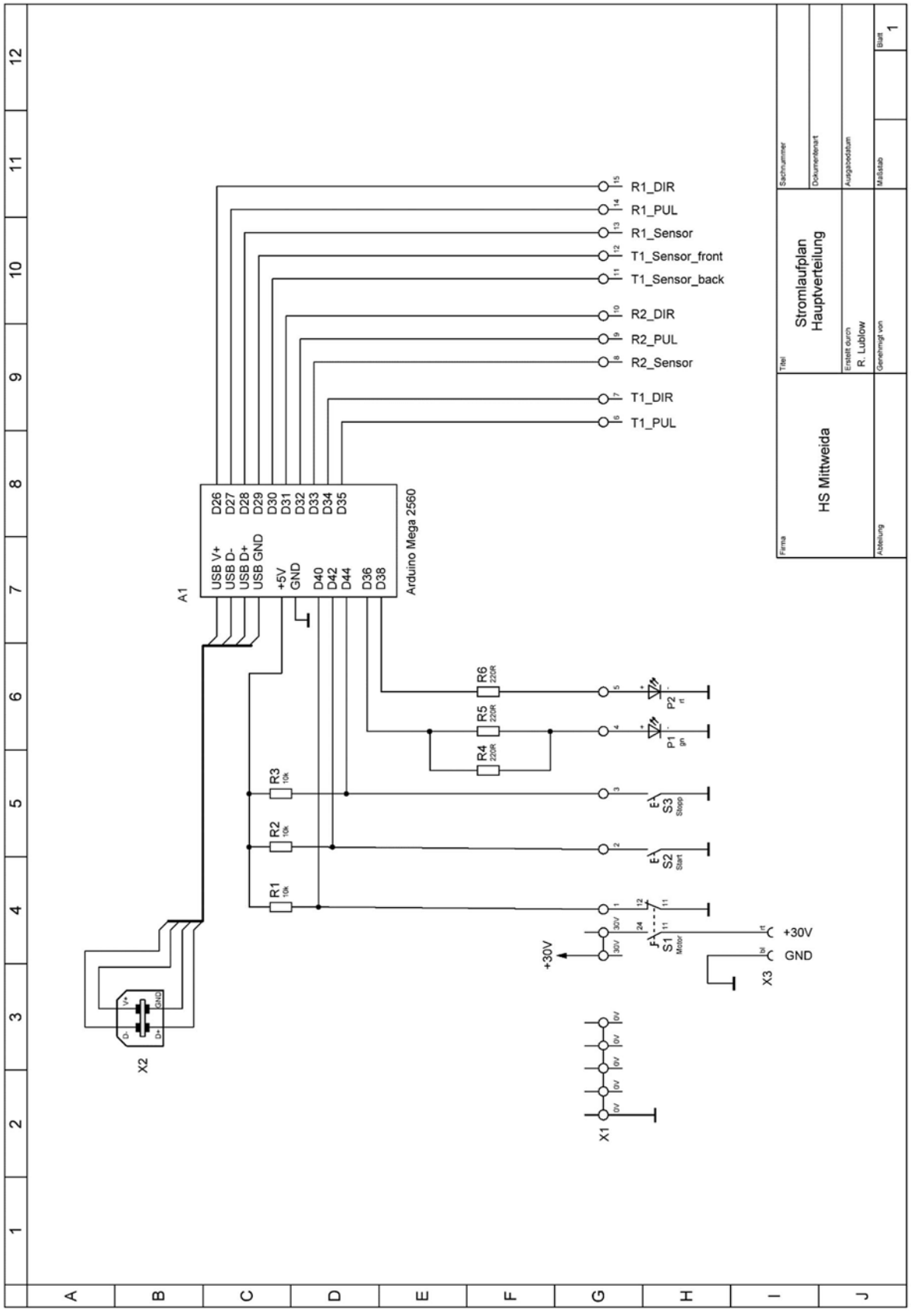
Wang, Yujing (2021): Konzeptionierung, Entwurf und Umsetzung eines automatisierten Sensor-Prüfstands. Masterarbeit. Hochschule Mittweida, Mittweida. Fakultät Ingenieurwissenschaften.

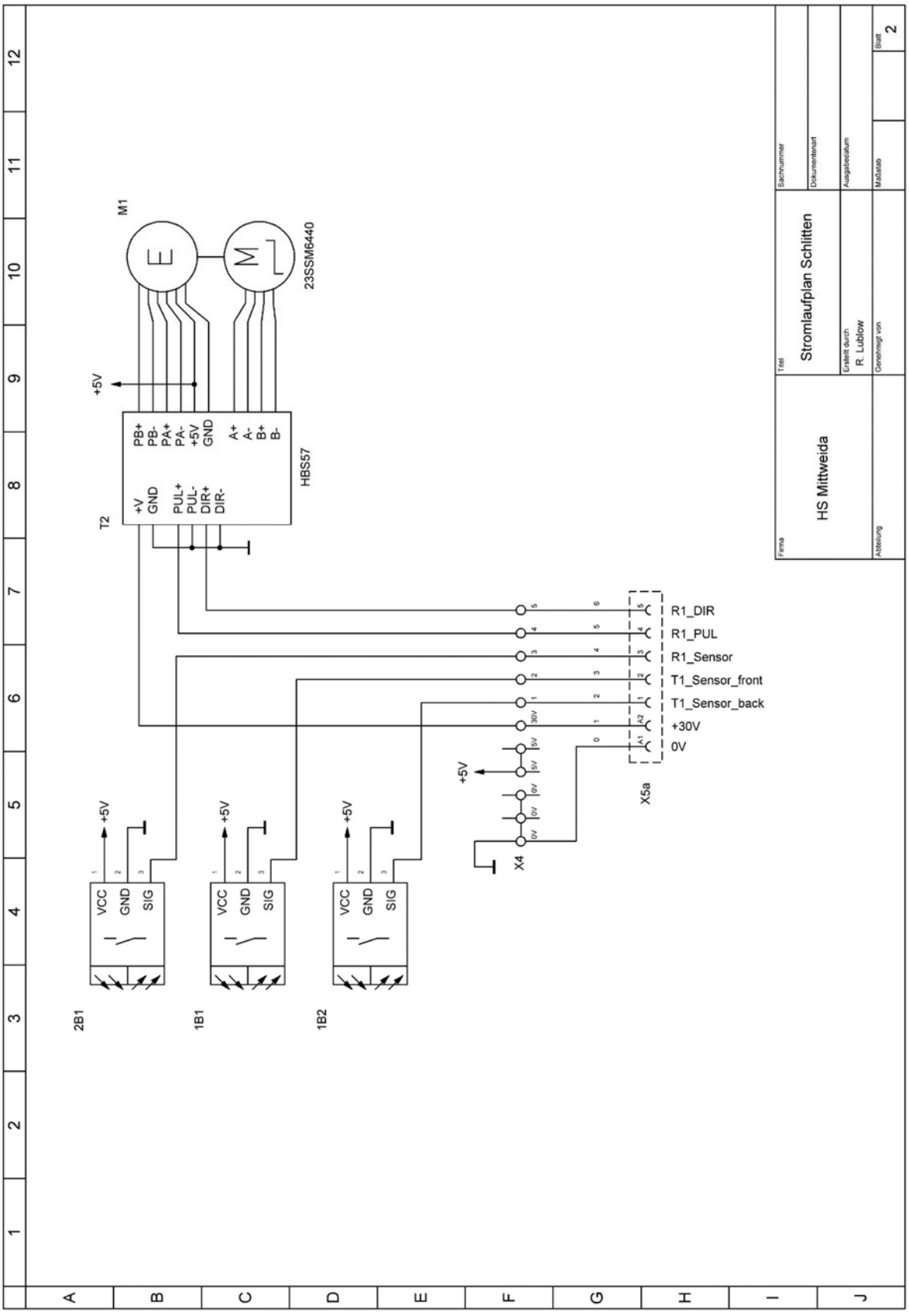
Wheat, Dale (2011): Arduino Internals. Berkeley, CA: Apress L. P. Online verfügbar unter https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=883804.

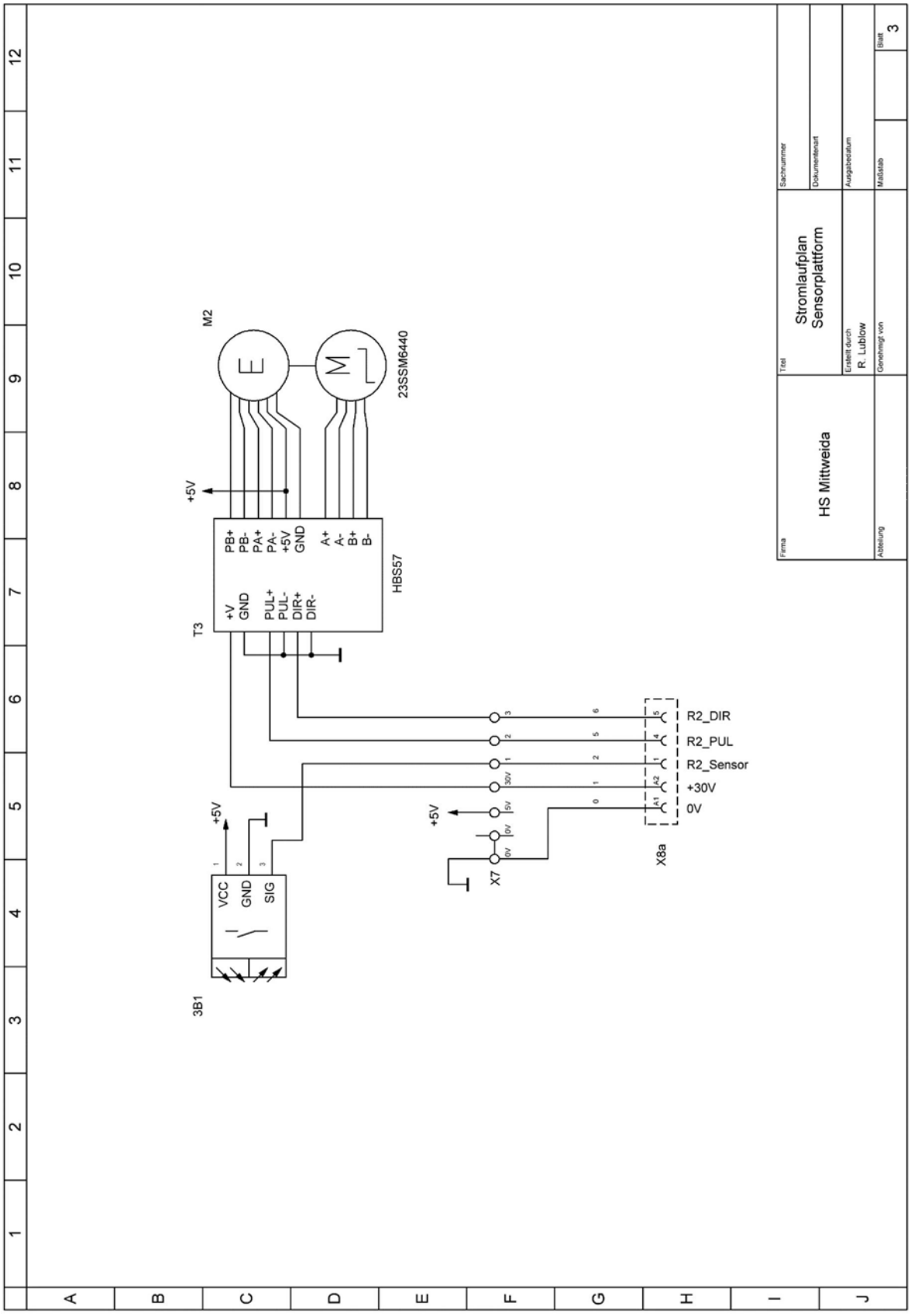
Anhang

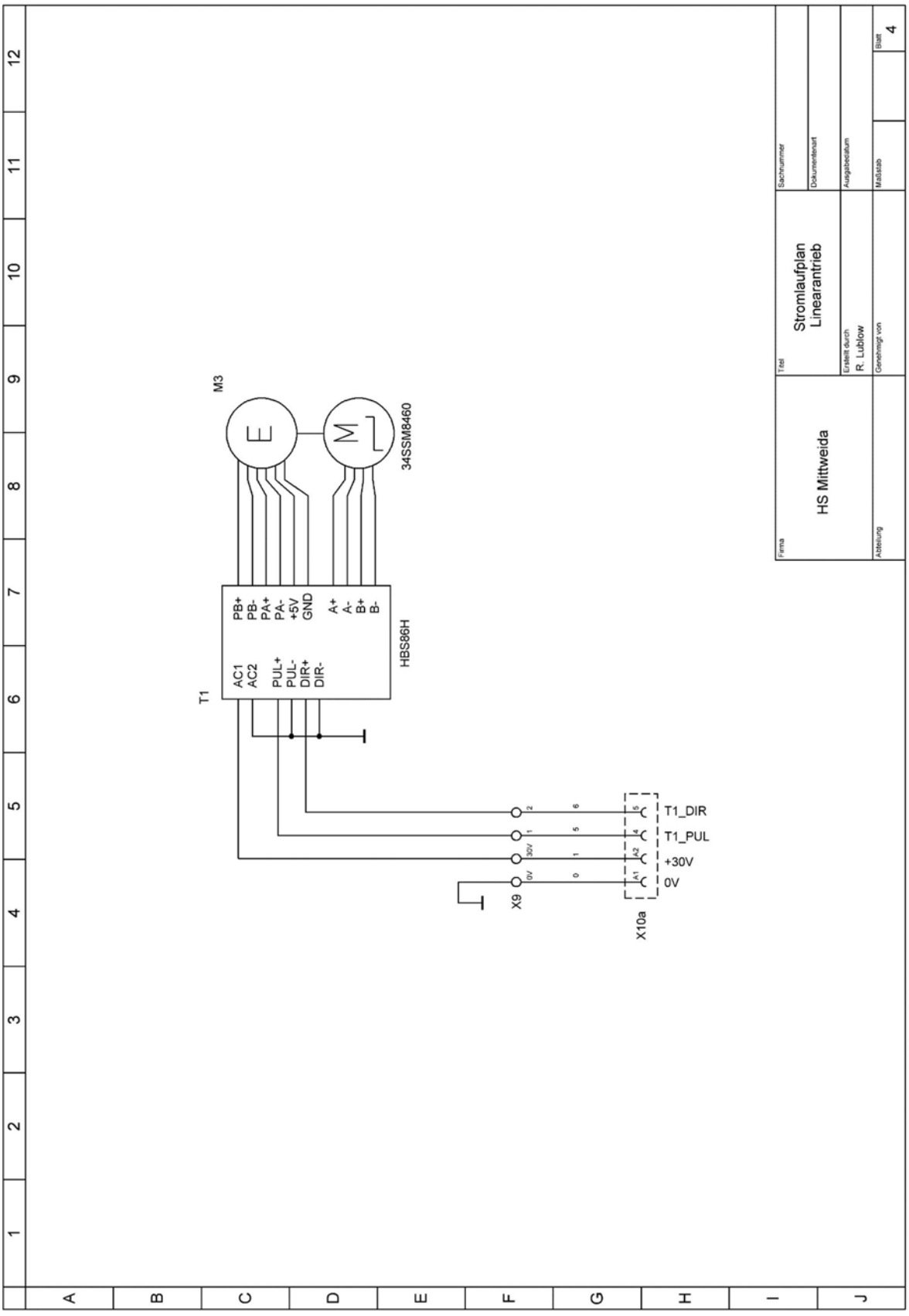
Anhang, Teil 1

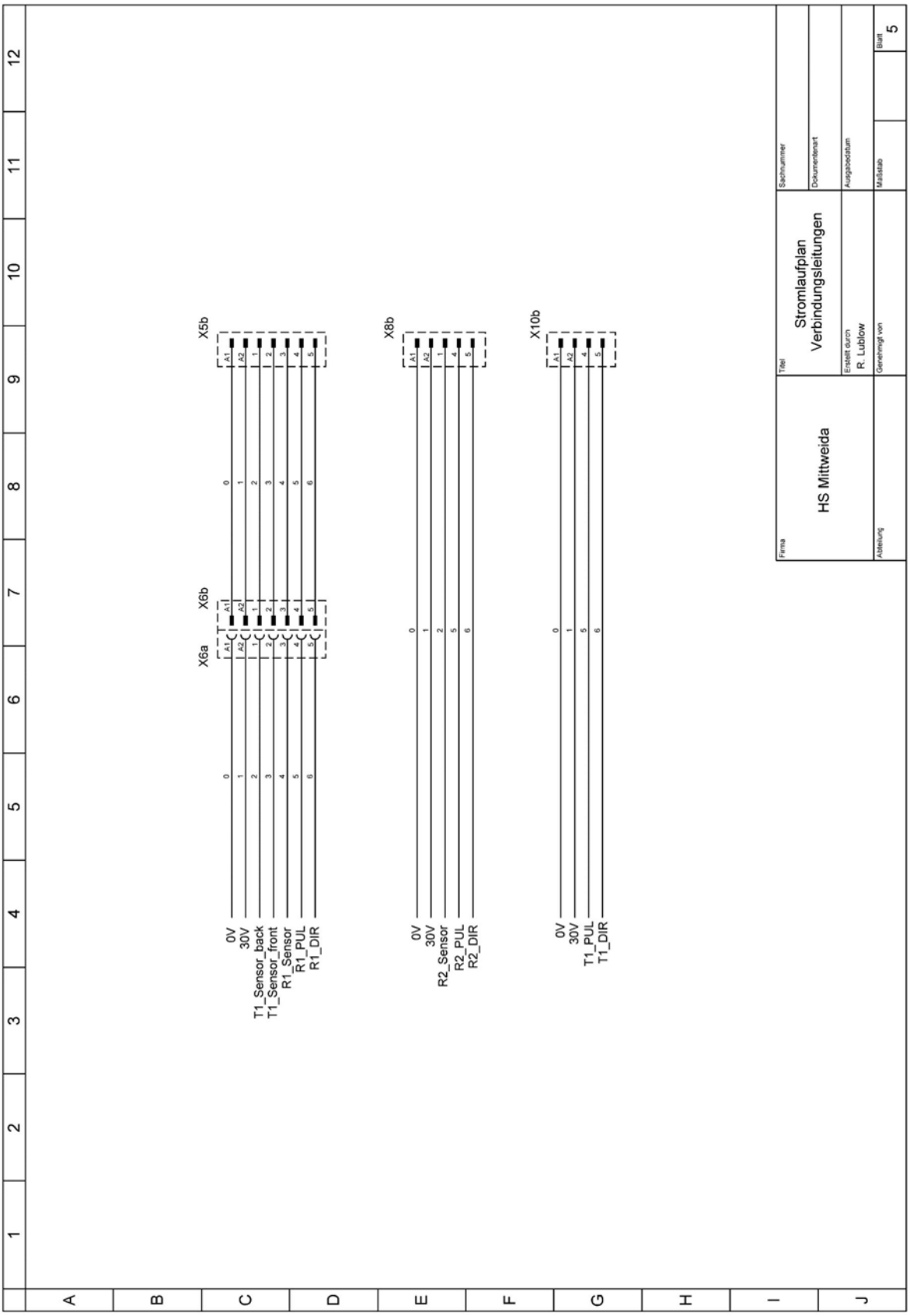
Stromlaufpläne des Sensorteststands




M1
23SSM6440
+5V
PB+
PB-
PA+
PA-
+5V
GND
A+
A-
B+
B-
HBS57
T2
+V
GND
PUL+
PUL-
DIR+
DIR-
R1_DIR
R1_PUL
R1_Sensor
T1_Sensor_front
T1_Sensor_back
+30V
0V
X5a
X4
VCC
GND
SIG
2B1
1B1
1B2
Firma
HS Mittweida
Titel
Stromlaufplan Schlitten
Erstellt durch
R. Lublow
Abteilung
Genehmigt von
Sachnummer
Dokumentenart
Ausgabedatum
Maßstab
Blatt
2


M2
23SSM6440
+5V
PB+
PB-
PA+
PA-
+5V
GND
A+
A-
B+
B-
HBS57
T3
+V
GND
PUL+
PUL-
DIR+
DIR-
R2_DIR
R2_PUL
R2_Sensor
+30V
0V
X8a
X7
VCC
GND
SIG
3B1
Firma
HS Mittweida
Titel
Stromlaufplan
Sensorplattform
Erstellt durch
R. Lublow
Abteilung
Genehmigt von
Sachnummer
Dokumentenart
Ausgabedatum
Maßstab
Blatt
3


M3
34SSM8460
HBS86H
T1
PB+
PB-
PA+
PA-
+5V
GND
A+
A-
B+
B-
AC1
AC2
PUL+
PUL-
DIR+
DIR-
T1_DIR
T1_PUL
+30V
0V
X9
X10a
HS Mittweida
Stromlaufplan
Linearantrieb
R. Lublow
Blatt
4


X5b
X6a
X6b
X8b
X10b
0V
30V
T1_Sensor_back
T1_Sensor_front
R1_Sensor
R1_PUL
R1_DIR
0V
30V
R2_Sensor
R2_PUL
R2_DIR
0V
30V
T1_PUL
T1_DIR
Firma
HS Mittweida
Titel
Stromlaufplan
Verbindungsleitungen
Erstellt durch
R. Lublow
Abteilung
Genehmigt von
Sachnummer
Dokumentenart
Ausgabedatum
Maßstab
Blatt
5

Anhang, Teil 2

Stückliste des zusätzlich beschafften Materials

Pos.	Lieferant	Bezeichnung + Menge	Betrag
1	Roboter-Bausatz.de	2x Motorhalter 5x Schleppkette 1m	34,25 €
2	Schrauben-Niro	1x Sortiment Zylinderschrauben M3/M4	24,01 €
3	Amazon	6x Gabellichtschranke	11,99 €
4	Conrad	10m Ölflex Schleppkettenleitung 7G, 0,75mm²	44,15 €
5	Aliexpress	5x D-Sub male 5x D-Sub female	22,24 €
6	Schrauben-guenstiger.de	1x Sortiment Zylinderschrauben M5	24,85 €
7	Conrad	8x SCHNELLMONTAGE-ENDHALTER 35x DURCHGANGSREIHENKLEMME PT 2,5 15x DURCHGANGSREIHENKLEMME PT 2,5 BU 3x FBS 3-5 5x BESCHRIFTUNGSSTREIFEN ZBF 6:UNBEDRUCKT 1x DRUCKTASTER R13-523AL-05 RT LED TC 1x 4 MM EINBAUBUCHSEN LB-I4R-A ROT 1x DREHSCHALTER LAS0-A3Y-11X/21 1x 4 MM EINBAUBUCHSEN LB-I4R-A BLAU 1x USB-EINBAUADAPTER TYP B AUF TYP A 1x DRUCKTASTER R13-523AL-05 GN LED TC	92,07 €
8	Conrad	1x TRAGSCHIENE DELTA-3F/BV 2M 1x FBS 2-5	20,98 €
9	Amazon	3x Therlan Verteilerdose Außen Wasserdicht	41,97 €
10	Amazon	PCB Prototyp Solder Board	12,99 €
11	Amazon	Dabixx PieceB 35-mm-Adapter für die Hutschienenmontage	4,99 €
12	Amazon	Schaltschrank IP65 Industriegehäuse 250 x 350 x 150 mm	22,49 €
13	Hydrauliktechnik24	2x Flanschlager UCFL	16,33 €
14	Monkeypower	1x Präzisionsstahlrohr Hülse Buchse 12 /14mm	10,80 €
15	Zahnriemen24	1x Präzisions-Wellenstahl CF53 - 12 mm (h6) x 150 mm	15,50 €
16	Amazon	50x Kabelbinder Halter Schraubbar	12,38 €
17	Amazon	10x Linsenkopf Schrauben M4x30	3,43 €
18	Amazon	10x Maschinenschrauben M4x80	18,98 €
19	Amazon	100x 0,5mm², weiss, Aderendhülse	3,80 €
20	Amazon	100x 0,75mm², grau, Aderendhülse	3,80 €
21	Amazon	Kabelverschraubung Set	9,99 €
		Summe:	451,99 €

Anhang, Teil 3

Quelltext der Datei Sensorteststand_Arduino_v1.ino für die Mikrocontroller-Steuerung

```
/* Arduino Code fuer das Projekt Sensorteststand

  Autor: Robert Lublow

  v1: Alle Grundfunktionen enthalten
*/

#include <AccelStepper.h>
#include <CmdMessenger.h>

//----------------Definition Anschluesse-----------------------

//Anschlusspins Stepper
const int PulsePinT1 = 35;
const int DirPinT1 = 34;
const int PulsePinR1 = 27;
const int DirPinR1 = 26;
const int PulsePinR2 = 32;
const int DirPinR2 = 31;

//Anschlusspins fuer Inputs
const int T1SensorFront = 29;
const int T1SensorBack = 30;
const int R1Sensor = 28;
const int R2Sensor = 33;
const int ButtonStart = 42;
const int ButtonStopp = 44;
const int SwitchMotors = 40;

//Anschlusspins LEDs
const int LedGreen = 36;
const int LedRed = 38;

//-----Definition Bewegungsvariablen fuer Steppermotoren-------

//T1
int SetStpT1 = 0;         // Steps
float SetSpdT1 = 0;       // Speed in Steps/s
float SetAccT1 = 0;       // Accelaration in Steps/s²
bool MoveT1 = false;      // Flag for movement

//R1
int SetStpR1 = 0;
float SetSpdR1 = 0;
float SetAccR1 = 0;
bool MoveR1 = false;

//R2
int SetStpR2 = 0;
float SetSpdR2 = 0;
float SetAccR2 = 0;
bool MoveR2 = false;

//------------------------AccelStepper--------------------------

//Erzeugung Objekte
AccelStepper T1(1, PulsePinT1, DirPinT1); // Liniearantrieb
AccelStepper R1(1, PulsePinR1, DirPinR1); // Schlitten
AccelStepper R2(1, PulsePinR2, DirPinR2); // Sensorplattform

//------------Definition Variablen fur Inputhandling------------

//T1SensorFront
int T1SensorFrontOperation = 1;            //Set Operation
bool T1SensorFrontVal = 0;                 //Current Input Reading
bool T1SensorFrontOldVal = 0;              //Last Input Reading
int T1SensorFrontState = 0;                //Sensor State
unsigned long T1SensorFrontTimeOn = 0;     //Time when Sensor turned On
unsigned long T1SensorFrontTimeOff = 0;    //Time when Sensor turned Off
bool T1SensorFrontTriggerOn = 0;           //Flag Sensor turned On
bool T1SensorFrontTriggerOff = 0;          //Time when Sensor turned Off

//T1SensorBack
int T1SensorBackOperation = 1;
bool T1SensorBackVal = 0;
bool T1SensorBackOldVal = 0;
int T1SensorBackState = 0;
unsigned long T1SensorBackTimeOn = 0;
unsigned long T1SensorBackTimeOff = 0;
bool T1SensorBackTriggerOn = 0;
bool T1SensorBackTriggerOff = 0;

//R1Sensor
int R1SensorOperation = 1;
bool R1SensorVal = 0;
bool R1SensorOldVal = 0;
int R1SensorState = 0;
unsigned long R1SensorTimeOn = 0;
unsigned long R1SensorTimeOff = 0;
bool R1SensorTriggerOn  = 0;
bool R1SensorTriggerOff = 0;

//R2Sensor
int R2SensorOperation = 1;
bool R2SensorVal = 0;
bool R2SensorOldVal = 0;
int R2SensorState = 0;
unsigned long R2SensorTimeOn = 0;
unsigned long R2SensorTimeOff = 0;
bool R2SensorTriggerOn = 0;
bool R2SensorTriggerOff = 0;

//ButtonStart
int ButtonStartOperation = 0;
bool ButtonStartVal = 0;
bool ButtonStartOldVal = 0;
int ButtonStartState = 0;
unsigned long ButtonStartTimeOn = 0;
unsigned long ButtonStartTimeOff = 0;
bool ButtonStartTriggerOn = 0;
bool ButtonStartTriggerOff = 0;
```

```
//ButtonStopp
int ButtonStoppOperation = 0;
bool ButtonStoppVal = 0;
bool ButtonStoppOldVal = 0;
int ButtonStoppState = 0;
unsigned long ButtonStoppTimeOn = 0;
unsigned long ButtonStoppTimeOff = 0;
bool ButtonStoppTriggerOn;
bool ButtonStoppTriggerOff;

//SwitchMotors
bool SwitchMotorsVal = 0;
bool SwitchMotorsOldVal = 0;
int SwitchMotorsState = 0;
unsigned long SwitchMotorsTimeOn = 0;
unsigned long SwitchMotorsTimeOff = 0;
bool SwitchMotorsTriggerOn;
bool SwitchMotorsTriggerOff;

//Delays
int DelayOptical = 2;      //Totzeit gegen Glitche der optischen Sensoren
int DelayMechanical = 50;  //Totzeit fuer Entprellen

//-------------------CmdMessenger--------------------------

//Erzeugung CmdMessenger Objekt
CmdMessenger mail = CmdMessenger(Serial, ',', ';', '/');

//Serial Command List
enum
{
  mStopp,
  mStart,
  mMoveCompleted,
  mMoveAborted,
  mSetSteps,
  mSetAcc,
  mSetSpeed,
  mInputState,
  mSetInputOperation,
};

// Attach Callback Methods
void attachCommandCallbacks()
{
  mail.attach(mStopp, mDoStopp);
  mail.attach(mStart, mDoStart);
  mail.attach(mSetSteps, mDoSetSteps);
  mail.attach(mSetAcc, mDoSetAcc);
  mail.attach(mSetSpeed, mDoSetSpeed);
  mail.attach(mInputState, mDoInputState);
  mail.attach(mSetInputOperation, mDoSetInputOperation);
}

//Serial Command Parameter: Input
int mInput = 0;
const int mT1SensorFront = 1;
const int mT1SensorBack = 2;
const int mR1Sensor = 3;
const int mR2Sensor = 4;
const int mButtonStart = 5;
const int mButtonStopp = 6;
const int mSwitchMotors = 7;
const int mSerial = 8;

//Serial Command Parameter: Axis
int mAxis = 0;
const int mT1 = 1;
const int mR1 = 2;
const int mR2 = 3;

//Callbacks

void mDoStopp()
{
  Stopp(mSerial);       // call Stopp()
}

void mDoStart()
{
  mAxis = mail.readBinArg<int>();      // read in Axis number
  Start(mAxis);                       // call Start()
  ButtonStartOperation = 0;           // Deaktiviere Start Taster
  digitalWrite(LedGreen, LOW);
}

void mDoSetSteps()
{
  mAxis = mail.readBinArg<int>();            // read in Axis number

  switch (mAxis) {
    case 1:                                  // T1
      SetStpT1 = mail.readBinArg<int>();     // read in Steps
      T1.move(SetStpT1);                     // write steps into move()
      break;
    case 2:                                  // R1
      SetStpR1 = mail.readBinArg<int>();     // read in Steps
      R1.move(SetStpR1);                     // write steps into move()
      break;
    case 3:                                  // R2
      SetStpR2 = mail.readBinArg<int>();     // read in steps
      R2.move(SetStpR2);                     // write steps into move()
      break;
  }
}

void mDoSetAcc()
{
  mAxis = mail.readBinArg<int>();             // read in Axis number

  switch (mAxis) {
    case 1:                                   // T1
      SetAccT1 = mail.readBinArg<float>();    // read in Acceleration
      T1.setAcceleration(SetAccT1);           // set acceleration
      break;
    case 2:                                   // R1
      SetAccR1 = mail.readBinArg<float>();    // read in Acceleration
      R1.setAcceleration(SetAccR1);           // set acceleration
      break;
    case 3:                                   // R2
      SetAccR2 = mail.readBinArg<float>();    // read in Acceleration
      R2.setAcceleration(SetAccR2);           // set acceleration
```

```
      break;
  }
}

void mDoSetSpeed()
{
  mAxis = mail.readBinArg<int>();           // read in Axis number

  switch (mAxis) {
    case 1:                                 // T1
      SetSpdT1 = mail.readBinArg<float>();  // read in argument for max speed
      T1.setMaxSpeed(SetSpdT1);             // set max speed
      break;
    case 2:                                 // R1
      SetSpdR1 = mail.readBinArg<float>();  // read in argument for max speed
      R1.setMaxSpeed(SetSpdR1);             // set max speed
      break;
    case 3:                                 // R2
      SetSpdR2 = mail.readBinArg<float>();  // read in argument for max speed
      R2.setMaxSpeed(SetSpdR2);             // set max speed
      break;
  }
}

void mDoInputState() {
  mInput = mail.readBinArg<int>();                      //Read in Input

  switch (mInput) {
    case 1:
      mail.sendBinCmd(mInputState, T1SensorFrontState);  //Send State of T1SensorFront
      break;
    case 2:
      mail.sendBinCmd(mInputState, T1SensorBackState);   //Send State of T1SensorBack
      break;
    case 3:
      mail.sendBinCmd(mInputState, R1SensorState);       //Send State of mR1Sensor
      break;
    case 4:
      mail.sendBinCmd(mInputState, R2SensorState);       //Send State of R2SensorFront
      break;
    case 7:
      mail.sendBinCmd(mInputState, R1SensorState);       //Send State of SwitchMotors
      break;
  }
}

void mDoSetInputOperation() {

  mInput = mail.readBinArg<int>();                      //Read in Input

  switch (mInput) {
    case 1:
      T1SensorFrontOperation = mail.readBinArg<int>();   //Setze T1SensorFront On/Off
      break;
    case 2:
      T1SensorBackOperation = mail.readBinArg<int>();    //Setze T1SensorBack On/Off
      break;
    case 3:
      R1SensorOperation = mail.readBinArg<int>();        //Setze R1Sensor On/Off
      break;
    case 4:
      R2SensorOperation = mail.readBinArg<int>();        //Setze R2Sensor On/Off
      break;
    case 5:
      ButtonStartOperation = mail.readBinArg<int>();     //Setze ButtonStart On/Off
      digitalWrite(LedGreen, HIGH);                      //Aktiviere Start Led
      break;
    case 6:
      ButtonStoppOperation = mail.readBinArg<int>();     //Setze ButtonStopp On/Off
      digitalWrite(LedRed, HIGH);                        //Aktiviere Stopp Led
      break;
  }
}

//------------------------Setup----------------------------

void setup() {

  // Setting up I/O Pins
  pinMode(LedGreen, OUTPUT);
  pinMode(LedRed, OUTPUT);
  pinMode(T1SensorFront, INPUT);
  pinMode(T1SensorBack, INPUT);
  pinMode(R1Sensor, INPUT);
  pinMode(R2Sensor, INPUT);
  pinMode(ButtonStart, INPUT);
  pinMode(ButtonStopp, INPUT);
  pinMode(SwitchMotors, INPUT);

  // Setting up default Motor values
  T1.setMaxSpeed(SetSpdT1);
  T1.setAcceleration(SetAccT1);
  R1.setMaxSpeed(SetSpdR1);
  R1.setAcceleration(SetAccR1);
  R2.setMaxSpeed(SetSpdR2);
  R2.setAcceleration(SetAccR2);

  //Setting up Serial Connection
  Serial.begin(115200);       // Open Connection, Baud rate 115200
  mail.printLfCr();           // Adds newline to every command
  attachCommandCallbacks();   // Attach callback methods for CmdMessenger

}

//----------------------------Mainloop-----------------------------

void loop() {

  mail.feedinSerialData();  // Nachrichten verarbeiten
  Inputs();                 // Inputs verarbeiten
  MovementT1();             // Bewegung T1 ausführen
  MovementR1();             // Bewegung R1 ausführen
  MovementR2();             // Bewegung R2 ausführen
}

//----------------------------Functions----------------------------

void Inputs() {
  ReadSwitchMotors();                                   //Motorschalter einlesen
  if (T1SensorFrontOperation == 1)  ReadT1SensorFront();  //T1SensorFront einlesen
  if (T1SensorBackOperation == 1)  ReadT1SensorBack();    //T1SensorBack einlesen
  if (R1SensorOperation == 1)  ReadR1Sensor();            //R1Sensor einlesen
```

```
  if (R2SensorOperation == 1)  ReadR2Sensor();            //R2Sensor einlesen
  if (ButtonStartOperation == 1)  ReadButtonStart();      //ButtonStart einlesen
  if (ButtonStoppOperation == 1)   ReadButtonStopp();     //ButtonStopp einlesen
}

void MovementT1() {
  if (MoveT1 == true) {                                 // Start triggered
    T1.run();                                           // check and do next step
    if (!T1.run()) {                                    // movement completed
      MoveT1 = false;                                   // reset start trigger
      mail.sendCmd(mMoveCompleted);                     // report move completed
      ButtonStoppOperation = 0;                         // Deaktiviere Stopp Taster
      digitalWrite(LedRed, LOW);
      T1.setCurrentPosition(0);                         // reset current position
    }
  }
}

void MovementR1() {
  if (MoveR1 == true) {                                 // Start triggered
    R1.run();                                           // check and do next step
    if (!R1.run()) {                                    // movement completed
      MoveR1 = false;                                   // reset start trigger
      mail.sendCmd(mMoveCompleted);                     // report move completed
      ButtonStoppOperation = 0;                         // Deaktiviere Stopp Taster
      digitalWrite(LedRed, LOW);
      R1.setCurrentPosition(0);                         // reset current position
    }
  }
}

void MovementR2() {
  if (MoveR2 == true) {                                 // Start triggered
    R2.run();                                           // check and do next step
    if (!R2.run()) {                                    // movement completed
      MoveR2 = false;                                   // reset start trigger
      mail.sendCmd(mMoveCompleted);                     // report move completed
      ButtonStoppOperation = 0;                         // Deaktiviere Stopp Taster
      digitalWrite(LedRed, LOW);
      R2.setCurrentPosition(0);                         // reset current position
    }
  }
}

void Stopp(int var) {

  switch (var) {
    case 1:                                             // mT1SensorFront triggered
      if (MoveT1 == true) {
        MoveT1 = false;                                 // Stopp T1
        mail.sendBinCmd(mMoveAborted, mT1SensorFront); // report Stopp + Input
        T1.setCurrentPosition(0);                       // reset current position
      }
      break;
    case 2:                                             // mT1SensorBack triggered
      if (MoveT1 == true) {
        MoveT1 = false;                                 // Stopp T1
        mail.sendBinCmd(mMoveAborted, mT1SensorBack);  // report Stopp + Input
        T1.setCurrentPosition(0);                       // reset current position
      }
      break;
    case 3:                                             // mR1Sensor triggered
      if (MoveR1 == true) {
        MoveR1 = false;                                 // Stopp R1
        mail.sendBinCmd(mMoveAborted, mR1Sensor);       // report Stopp + Input
        R1.setCurrentPosition(0);                       // reset current position
      }
      break;
    case 4:                                             // mR2Sensor triggered
      if (MoveR2 == true) {
        MoveR2 = false;                                 // Stopp R2
        mail.sendBinCmd(mMoveAborted, mR2Sensor);       // report Stopp + Input
        R2.setCurrentPosition(0);                       // reset current position
      }
      break;
    case 6:                                             // Stopp Button pushed
      if (MoveT1 == true) {                             // T1 moving
        MoveT1 = false;                                 // Stopp T1
        T1.setCurrentPosition(0);                       // reset current position
      }
      if (MoveR1 == true) {                             // R1 moving
        MoveR1 = false;                                 // Stopp R1
        R1.setCurrentPosition(0);                       // reset current position
      }
      if (MoveR2 == true) {                             // R2 moving
        MoveR2 = false;                                 // Stopp R2
        R2.setCurrentPosition(0);                       // reset current position
      }
      if (ButtonStartOperation == 1) {                  // Abort Start via Button Stopp
        ButtonStartOperation = 0;
        digitalWrite(LedGreen, LOW);
      }
      mail.sendBinCmd(mMoveAborted, mButtonStopp);      // report Stopp + Input
      break;
    case 8:                                             // Stopp sent via Serial
      if (MoveT1 == true) {                             // T1 moving
        MoveT1 = false;                                 // Stopp T1
        T1.setCurrentPosition(0);                       // reset current position
      }
      if (MoveR1 == true) {                             // R1 moving
        MoveR1 = false;                                 // Stopp R1
        R1.setCurrentPosition(0);                       // reset current position
      }
      if (MoveR2 == true) {                             // R2 moving
        MoveR2 = false;                                 // Stopp R2
        R2.setCurrentPosition(0);                       // reset current position
      }
      mail.sendBinCmd(mMoveAborted, mSerial);           // report Stopp + Input
      break;
  }
  ButtonStoppOperation = 0;                             // Deaktiviere Stopp Taster
  digitalWrite(LedRed, LOW);
}

void Start(int var) {

  switch (var) {
    case 1:
      MoveT1 = true;                // set trigger for movement T1
      mail.sendCmd(mStart);         // report start of movement T1
      break;
    case 2:
```

```
      MoveR1 = true;               // set trigger for movement R1
      break;
    case 3:
      MoveR2 = true;               // set trigger for movement R2
      break;
  }
  ButtonStoppOperation = 1;        // Aktiviere Stopp Taster
  digitalWrite(LedRed, HIGH);
}

void ReadT1SensorFront() {

  T1SensorFrontVal = digitalRead(T1SensorFront);                    //Sensor einlesen

  if ((T1SensorFrontVal == HIGH) && (T1SensorFrontOldVal == LOW)) {  //Transition 0>1
    T1SensorFrontTimeOn = millis();                                  //Einschaltzeitpunkt speichern
    T1SensorFrontTriggerOn = 1;                                      //Flag setzen fuer Ueberpruefung Einschaltzeit
    T1SensorFrontTriggerOff = 0;                                     //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((T1SensorFrontTriggerOn == 1) && (T1SensorFrontState == 0)) {  //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - T1SensorFrontTimeOn) > DelayOptical) {           //laenger als Delay eingeschaltet
      T1SensorFrontState = 1;                                        //Setze Sensorstatus und uebermittle
      Stopp(mT1SensorFront);                                         //Rufe Funktion Stopp auf
      T1SensorFrontTriggerOn = 0;                                    //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((T1SensorFrontVal == LOW) && (T1SensorFrontOldVal == HIGH)) {  //Transition 1>0
    T1SensorFrontTimeOff = millis();                                 //Ausschaltzeitpunkt speichern
    T1SensorFrontTriggerOff = 1;                                     //Flag setzen fuer Ueberpruefung Ausschaltzeit
    T1SensorFrontTriggerOn = 0;                                      //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((T1SensorFrontTriggerOff == 1) && (T1SensorFrontState == 1)) { //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - T1SensorFrontTimeOff) > DelayOptical) {          //laenger als Delay ausgeschaltet
      T1SensorFrontState = 0;                                        //Setze Sensorstatus und uebermittle
      T1SensorFrontTriggerOff = 0;                                   //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  T1SensorFrontOldVal = T1SensorFrontVal;                            //Ablegen des Inputwerts
}

void ReadT1SensorBack() {

  T1SensorBackVal = digitalRead(T1SensorBack);                      //Sensor einlesen

  if ((T1SensorBackVal == HIGH) && (T1SensorBackOldVal == LOW)) {   //Transition 0>1
    T1SensorBackTimeOn = millis();                                  //Einschaltzeitpunkt speichern
    T1SensorBackTriggerOn = 1;                                      //Flag setzen fuer Ueberpruefung Einschaltzeit
    T1SensorBackTriggerOff = 0;                                     //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((T1SensorBackTriggerOn == 1) && (T1SensorBackState == 0)) {   //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - T1SensorBackTimeOn) > DelayOptical) {           //laenger als Delay eingeschaltet
      T1SensorBackState = 1;                                        //Setze Sensorstatus und uebermittle
      Stopp(mT1SensorBack);                                         //Rufe Funktion Stopp auf
      T1SensorBackTriggerOn = 0;                                    //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((T1SensorBackVal == LOW) && (T1SensorBackOldVal == HIGH)) {   //Transition 1>0
    T1SensorBackTimeOff = millis();                                 //Ausschaltzeitpunkt speichern
    T1SensorBackTriggerOff = 1;                                     //Flag setzen fuer Ueberpruefung Ausschaltzeit
    T1SensorBackTriggerOn = 0;                                      //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((T1SensorBackTriggerOff == 1) && (T1SensorBackState == 1)) {  //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - T1SensorBackTimeOff) > DelayOptical) {          //laenger als Delay ausgeschaltet
      T1SensorBackState = 0;                                        //Setze Sensorstatus und uebermittle
      T1SensorBackTriggerOff = 0;                                   //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  T1SensorBackOldVal = T1SensorBackVal;                             //Ablegen des Inputwerts
}

void ReadR1Sensor() {

  R1SensorVal = digitalRead(R1Sensor);                      //Sensor einlesen

  if ((R1SensorVal == HIGH) && (R1SensorOldVal == LOW)) {   //Transition 0>1
    R1SensorTimeOn = millis();                              //Einschaltzeitpunkt speichern
    R1SensorTriggerOn = 1;                                  //Flag setzen fuer Ueberpruefung Einschaltzeit
    R1SensorTriggerOff = 0;                                 //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((R1SensorTriggerOn == 1) && (R1SensorState == 0)) {   //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - R1SensorTimeOn) > DelayOptical) {       //laenger als Delay eingeschaltet
      R1SensorState = 1;                                    //Setze Sensorstatus und uebermittle
      Stopp(mR1Sensor);                                     //Rufe Funktion Stopp auf
      R1SensorTriggerOn = 0;                                //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((R1SensorVal == LOW) && (R1SensorOldVal == HIGH)) {   //Transition 1>0
    R1SensorTimeOff = millis();                             //Ausschaltzeitpunkt speichern
    R1SensorTriggerOff = 1;                                 //Flag setzen fuer Ueberpruefung Ausschaltzeit
    R1SensorTriggerOn = 0;                                  //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((R1SensorTriggerOff == 1) && (R1SensorState == 1)) {  //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - R1SensorTimeOff) > DelayOptical) {      //laenger als Delay ausgeschaltet
      R1SensorState = 0;                                    //Setze Sensorstatus und uebermittle
      R1SensorTriggerOff = 0;                               //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  R1SensorOldVal = R1SensorVal;                             //Ablegen des Inputwerts
}

void ReadR2Sensor() {
  R2SensorVal = digitalRead(R2Sensor);                      //Sensor einlesen

  if ((R2SensorVal == HIGH) && (R2SensorOldVal == LOW)) {   //Transition 0>1
    R2SensorTimeOn = millis();                              //Einschaltzeitpunkt speichern
    R2SensorTriggerOn = 1;                                  //Flag setzen fuer Ueberpruefung Einschaltzeit
    R2SensorTriggerOff = 0;                                 //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((R2SensorTriggerOn == 1) && (R2SensorState == 0)) {    //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - R2SensorTimeOn) > DelayOptical) {       //laenger als Delay eingeschaltet
      R2SensorState = 1;                                    //Setze Sensorstatus und uebermittle
      Stopp(mR2Sensor);                                     //Rufe Funktion Stopp auf
      R2SensorTriggerOn = 0;                                //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((R2SensorVal == LOW) && (R2SensorOldVal == HIGH)) {   //Transition 1>0
    R2SensorTimeOff = millis();                             //Ausschaltzeitpunkt speichern
    R2SensorTriggerOff = 1;                                 //Flag setzen fuer Ueberpruefung Ausschaltzeit
    R2SensorTriggerOn = 0;                                  //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((R2SensorTriggerOff == 1) && (R2SensorState == 1)) {  //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - R2SensorTimeOff) > DelayOptical) {      //laenger als Delay ausgeschaltet
```

```
      R2SensorState = 0;                                  //Setze Sensorstatus und uebermittle
      R2SensorTriggerOff = 0;                             //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  R2SensorOldVal = R2SensorVal;                           //Ablegen des Inputwerts
}

void ReadButtonStart() {

  ButtonStartVal = !digitalRead(ButtonStart);                     //Sensor einlesen (active LOW)

  if ((ButtonStartVal == HIGH) && (ButtonStartOldVal == LOW)) {   //Transition 0>1
    ButtonStartTimeOn = millis();                                 //Einschaltzeitpunkt speichern
    ButtonStartTriggerOn = 1;                                     //Flag setzen fuer Ueberpruefung Einschaltzeit
    ButtonStartTriggerOff = 0;                                    //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((ButtonStartTriggerOn == 1) && (ButtonStartState == 0)) {   //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - ButtonStartTimeOn) > DelayMechanical) {       //laenger als Delay eingeschaltet
      ButtonStartState = 1;                                       //Setze Sensorstatus und uebermittle
      Start(mT1);                                                  //rufe Start() auf
      ButtonStartTriggerOn = 0;                                   //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((ButtonStartVal == LOW) && (ButtonStartOldVal == HIGH)) {   //Transition 1>0
    ButtonStartTimeOff = millis();                                //Ausschaltzeitpunkt speichern
    ButtonStartTriggerOff = 1;                                    //Flag setzen fuer Ueberpruefung Ausschaltzeit
    ButtonStartTriggerOn = 0;                                     //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((ButtonStartTriggerOff == 1) && (ButtonStartState == 1)) {  //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - ButtonStartTimeOff) > DelayMechanical) {      //laenger als Delay ausgeschaltet
      ButtonStartState = 0;                                       //Setze Sensorstatus und uebermittle
      ButtonStartOperation = 0;                                   // Deaktiviere Start Taster
      digitalWrite(LedGreen, LOW);
      ButtonStartTriggerOff = 0;                                  //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  ButtonStartOldVal = ButtonStartVal;                             //Ablegen des Inputwerts
}

void ReadButtonStopp() {

  ButtonStoppVal = !digitalRead(ButtonStopp);                     //Sensor einlesen (active LOW)

  if ((ButtonStoppVal == HIGH) && (ButtonStoppOldVal == LOW)) {   //Transition 0>1
    ButtonStoppTimeOn = millis();                                 //Einschaltzeitpunkt speichern
    ButtonStoppTriggerOn = 1;                                     //Flag setzen fuer Ueberpruefung Einschaltzeit
    ButtonStoppTriggerOff = 0;                                    //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((ButtonStoppTriggerOn == 1) && (ButtonStoppState == 0)) {   //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - ButtonStoppTimeOn) > DelayMechanical) {       //laenger als Delay eingeschaltet
      ButtonStoppState = 1;                                       //Setze Sensorstatus und uebermittle
      Stopp(mButtonStopp);                                        //Rufe Funktion Stopp auf
      ButtonStoppTriggerOn = 0;                                   //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((ButtonStoppVal == LOW) && (ButtonStoppOldVal == HIGH)) {   //Transition 1>0
    ButtonStoppTimeOff = millis();                                //Ausschaltzeitpunkt speichern
    ButtonStoppTriggerOff = 1;                                    //Flag setzen fuer Ueberpruefung Ausschaltzeit
    ButtonStoppTriggerOn = 0;                                     //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((ButtonStoppTriggerOff == 1) && (ButtonStoppState == 1)) {  //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - ButtonStoppTimeOff) > DelayMechanical) {      //laenger als Delay ausgeschaltet
      ButtonStoppState = 0;                                       //Setze Sensorstatus und uebermittle
      ButtonStoppTriggerOff = 0;                                  //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  ButtonStoppOldVal = ButtonStoppVal;                             //Ablegen des Inputwerts
}

void ReadSwitchMotors() {

  SwitchMotorsVal = digitalRead(SwitchMotors);                    //Sensor einlesen

  if ((SwitchMotorsVal == HIGH) && (SwitchMotorsOldVal == LOW)) {  //Transition 0>1
    SwitchMotorsTimeOn = millis();                                //Einschaltzeitpunkt speichern
    SwitchMotorsTriggerOn = 1;                                    //Flag setzen fuer Ueberpruefung Einschaltzeit
    SwitchMotorsTriggerOff = 0;                                   //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
  }
  if ((SwitchMotorsTriggerOn == 1) && (SwitchMotorsState == 0)) {  //Ueberpruefung Einschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - SwitchMotorsTimeOn) > DelayMechanical) {      //laenger als Delay eingeschaltet
      SwitchMotorsState = 1;                                      //Setze Sensorstatus und uebermittle
      SwitchMotorsTriggerOn = 0;                                  //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
    }
  }
  if ((SwitchMotorsVal == LOW) && (SwitchMotorsOldVal == HIGH)) {  //Transition 1>0
    SwitchMotorsTimeOff = millis();                               //Ausschaltzeitpunkt speichern
    SwitchMotorsTriggerOff = 1;                                   //Flag setzen fuer Ueberpruefung Ausschaltzeit
    SwitchMotorsTriggerOn = 0;                                    //Flag ruecksetzen fuer Ueberpruefung Einschaltzeit
  }
  if ((SwitchMotorsTriggerOff == 1) && (SwitchMotorsState == 1)) { //Ueberpruefung Ausschaltzeit falls sich Sensorstatus geaendert hat
    if ((millis() - SwitchMotorsTimeOff) > DelayMechanical) {     //laenger als Delay ausgeschaltet
      SwitchMotorsState = 0;                                      //Setze Sensorstatus und uebermittle
      SwitchMotorsTriggerOff = 0;                                 //Flag ruecksetzen fuer Ueberpruefung Ausschaltzeit
    }
  }
  SwitchMotorsOldVal = SwitchMotorsVal;                           //Ablegen des Inputwerts
}
```

Anhang, Teil 4

Quelltext der Datei Sensorteststand_Python_v1_01.py für das Programm des Host-PCs

```
# Python Code fuer das Projekt Sensorteststand
# Autor: Robert Lublow
# v1: Alle Grundfunktionen enthalten
# v1_01 Referenzmaß für Home Position von T1 hinzugefügt
# kompatibel mit Arduino Code v1

import PyCmdMessenger
import tkinter as tk
import param
import time

# Set up Serial Connection
arduino = PyCmdMessenger.ArduinoBoard(param.COM, baud_rate=115200)

# Command list identisch zu Arduino; Definition der Datentypen: i=int, f=float, ?=bool
commands = [["mStopp", ""],
            ["mStart", "i"],
            ["mMoveCompleted", ""],
            ["mMoveAborted", "i"],
            ["mSetSteps", "ii"],
            ["mSetAcc", "if"],
            ["mSetSpeed", "if"],
            ["mInputState", "i"],
            ["mSetInputOperation", "ii"]]

# initialize CmdMessenger
mail = PyCmdMessenger.CmdMessenger(arduino, commands)

# global variables
R1Pos = 0  # Position R1
R2Pos = 0  # Position R2
T1Pos = 0  # Position T1
T1newPos = 0  # Neue Position T1
T1Acc = param.T1DEFACC  # Beschleunigung T1 in m/s²
T1Speed = param.T1DEFSPD  # Geschwindigkeit T1 in m/s

# -------------------Functions-----------------------------

def T1Init():

    global T1Pos

    # Set speed and acc for homing
    mail.send('mSetAcc', 1, param.T1DEFACC/param.P1)
    mail.send('mSetSpeed', 1, param.T1DEFSPD/param.P1)
    mail.send('mSetInputOperation', 2, 0)  # Turn off T1SensorBack
    lT1Init['bg'] = 'red'  # Turn indicator for homing red
    lT1Acc['text'] = str(param.T1DEFACC) + ' m/s²'
    lT1Speed['text'] = str(param.T1DEFSPD) + ' m/s'
    lT1Pos['text'] = '?'  # Clear Output for Position T1
    bT1SetAcc['state'] = 'disabled'  # Disable Set Acc Button
    bT1SetSpeed['state'] = 'disabled'  # Disable Set Speed Button
    bT1SetPos['state'] = 'disabled'  # Disable Set Position Button
    eT1Acc['state'] = 'disabled'  # Disable Acc Entry
    eT1Speed['state'] = 'disabled'  # Disable Speed Entry
    eT1Pos['state'] = 'disabled'  # Disable Pos Entry

    mail.send('mInputState', 1)  # Check if T1SensorFront is triggered
    msg = mail.receive()

    if msg[1] == [1]:  # If T1 is already triggering T1SensorFront move away 40mm
        Steps = StepsT(0.04)
        mail.send("mSetSteps", 1, -Steps)
        mail.send('mStart', 1)
        msg = mail.receive()  # Read out unused feedback for movement startet

        msg = 'wait'
        while msg == 'wait':  # Wait for movement to finish
            msg = mail.receive()
            if msg == None:
                msg = 'wait'
        pass
        if msg[0] == 'mMoveCompleted':      # Movement finished
            time.sleep(0.4)
        if msg[0] == 'mMoveAborted':
            # Report Homing aborted
            lMessageBox['text'] = 'Initialisierung T1 abgebrochen'
            mail.send('mSetInputOperation', 2, 1)  # Turn on T1SensorBack
            return

    # do home return with distance longer than any real distance
    Steps = StepsT(10)
    mail.send("mSetSteps", 1, Steps)
    mail.send('mStart', 1)

    msg = 'wait'  # Wait for end of movement
    while not msg[0] == 'mMoveAborted':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
```

```
    pass

    if msg[1] == [1]:  # Check if sensor was reached
        time.sleep(0.4)  # Wait
        # Move to home position relative to sensor
        Steps = StepsT(param.HOMET1)
        mail.send('mSetSteps', 1, -Steps)
        mail.send('mStart', 1)

        msg = 'wait'  # Wait till position is reached
        while not (msg[0] == 'mMoveCompleted' or msg[0] == 'mMoveAborted'):
            msg = mail.receive()
            if msg == None:
                msg = 'wait'
        pass
        if msg[0] == 'mMoveCompleted':  # Homing finished
            lT1Init['bg'] = 'green'  # Turn indicator for homing green
            bT1SetAcc['state'] = 'normal'  # Enable Set Acc Button
            bT1SetSpeed['state'] = 'normal'  # Enable Set Speed Button
            bT1SetPos['state'] = 'normal'  # Enable Set Position Button
            eT1Acc['state'] = 'normal'  # Enable Acc Entry
            eT1Speed['state'] = 'normal'  # Enable Speed Entry
            eT1Pos['state'] = 'normal'  # Enable Pos Entry
            T1Pos = param.HOMET1_REF  # Safe initial Position T1
            lT1Pos['text'] = str(T1Pos) + " m"  # Output Position T1
            # Report homing finished
            lMessageBox['text'] = 'T1 auf Home Position'

    if msg[1] == [6]:  # Homing aborted
        # Report Homing aborted
        lMessageBox['text'] = 'Initialisierung T1 abgebrochen'

    mail.send('mSetInputOperation', 2, 1)  # Turn on T1SensorBack

def R1Init():

    global R1Pos

    mail.send('mSetAcc', 2, param.R1DEFACC)  # Set speed and acc for homing
    mail.send('mSetSpeed', 2, param.R1DEFSPD)
    mail.send('mSetInputOperation', 3, 1)  # Turn on Sensor
    lR1Init['bg'] = 'red'  # Turn indicator for homing red
    lR1Pos['text'] = '?'  # Clear Output for Position R1
    bR1Move['state'] = 'disabled'  # Disable Button Move R1
    eR1Pos['state'] = 'disabled'  # Disable Entry Pos R1

    # Set distance to search for sensor (>360) and start Movement
    Steps = StepsR(365)

    mail.send('mSetSteps', 2, Steps)
    mail.send('mStart', 2)

    msg = 'wait'  # Wait for end of movement
    while not msg[0] == 'mMoveAborted':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass

    if msg[1] == [3]:  # Check if sensor was reached
        time.sleep(0.4)  # Wait
        # Move to home position relative to sensor
        Steps = StepsR(param.HOMER1)
        mail.send('mSetSteps', 2, -Steps)
        mail.send('mStart', 2)

        msg = 'wait'  # Wait till position is reached or Stopp button pushed
        while not (msg[0] == 'mMoveCompleted' or msg[0] == 'mMoveAborted'):
            msg = mail.receive()
            if msg == None:
                msg = 'wait'
        pass
        if msg[0] == 'mMoveCompleted':  # Homing finished
            mail.send('mSetInputOperation', 3, 0)  # Turn off sensor
            lR1Init['bg'] = 'green'  # Turn indicator for homing green
            bR1Move['state'] = 'normal'  # Enable Button Move R1
            eR1Pos['state'] = 'normal'  # Enable Entry Pos R1
            R1Pos = 0.0  # Zero Psition R1
            lR1Pos['text'] = str(R1Pos) + "°"  # Output Position R1
            # Report homing finished
            lMessageBox['text'] = 'R1 auf Home Position'

    if msg[1] == [6]:  # Homing aborted
        # Report Homing aborted
        lMessageBox['text'] = 'Initialisierung R1 abgebrochen'

def R2Init():

    global R2Pos

    mail.send('mSetAcc', 3, param.R2DEFACC)  # Set speed and acc for homing
    mail.send('mSetSpeed', 3, param.R2DEFSPD)
    mail.send('mSetInputOperation', 4, 1)  # Turn on Sensor
    lR2Init['bg'] = 'red'  # Turn indicator for homing red
    lR2Pos['text'] = '?'  # Clear Output for Position R2
    bR2Move['state'] = 'disabled'  # Disable Button Move R2
    eR2Pos['state'] = 'disabled'  # Disable Entry Pos R2
```

```
    # Set distance for search for sensor (>360) and start Movement
    Steps = StepsR(365)

    mail.send('mSetSteps', 3, -Steps)
    mail.send('mStart', 3)

    msg = 'wait'  # Wait for end of movement
    while not msg[0] == 'mMoveAborted':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass

    if msg[1] == [4]:  # Check if sensor was reached
        time.sleep(0.4)  # Wait
        # Move to home position relative to sensor
        Steps = StepsR(param.HOMER2)
        mail.send('mSetSteps', 3, Steps)
        mail.send('mStart', 3)

        msg = 'wait'  # Wait till position is reached
        while not (msg[0] == 'mMoveCompleted' or msg[0] == 'mMoveAborted'):
            msg = mail.receive()
            if msg == None:
                msg = 'wait'
        pass
        if msg[0] == 'mMoveCompleted':  # Homing finished
            mail.send('mSetInputOperation', 4, 0)  # Turn off sensor
            lR2Init['bg'] = 'green'  # Turn indicator for homing green
            bR2Move['state'] = 'normal'  # Enable Button Move R2
            eR2Pos['state'] = 'normal'  # Enable Entry Pos R2
            R2Pos = 0.0  # Zero Psition R2
            lR2Pos['text'] = str(R2Pos) + "°"  # Output Position R2
            # Report homing finished
            lMessageBox['text'] = 'R2 auf Home Position'

    if msg[1] == [6]:  # Homing aborted
        # Report Homing aborted
        lMessageBox['text'] = 'Initialisierung R2 abgebrochen'

def T1Move():
    global T1Pos
    global T1newPos

    bT1Move['state'] = 'disabled'  # Disable button Start
    bT1MoveLocal['state'] = 'disabled'  # Disable button Start lokal
    bT1DontMove['state'] = 'disabled'  # Disable button Abbruch
    mail.send('mStart', 1)  # Start movement

    msg = mail.receive()  # Read out unused feedback for movement startet
    msg = 'wait'  # Wait till position is reached
    while msg == 'wait':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass

    if msg[0] == 'mMoveAborted':  # Movement aborted
        lT1Init['bg'] = 'red'  # Turn indicator for homing red
        lT1Pos['text'] = '?'  # Clear Output for Position T1
        lMessageBox['text'] = 'Bewegung T1 abgebrochen'  # print message

    if msg[0] == 'mMoveCompleted':  # Movement completed
        T1Pos = T1newPos  # Safe new position
        lMessageBox['text'] = 'Bewegung T1 abgeschlossen'  # Print message
        lT1Pos['text'] = str(T1Pos) + " m"  # Update label for position

        bT1SetAcc['state'] = 'normal'  # Enabel buttons for T1 control
        bT1SetSpeed['state'] = 'normal'
        bT1SetPos['state'] = 'normal'
        eT1Acc['state'] = 'normal'
        eT1Speed['state'] = 'normal'
        eT1Pos['state'] = 'normal'

def T1MoveLocal():

    global T1Pos
    global T1newPos

    bT1Move['state'] = 'disabled'  # Disable button Start
    bT1MoveLocal['state'] = 'disabled'  # Disable button Start lokal
    bT1DontMove['state'] = 'disabled'  # Disable button Abbruch
    # Enable local start Button at controller
    mail.send('mSetInputOperation', 5, 1)
    mail.send('mSetInputOperation', 6, 1)

    msg = 'wait'  # Wait for next event
    while msg == 'wait':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass

    if msg[0] == 'mMoveAborted':  # Start of movement canceled through local stopp button
        bT1SetAcc['state'] = 'normal'  # Enable buttons for T1 control
        bT1SetSpeed['state'] = 'normal'
        bT1SetPos['state'] = 'normal'
        eT1Acc['state'] = 'normal'
```

```
        eT1Speed['state'] = 'normal'
        eT1Pos['state'] = 'normal'
        lMessageBox['text'] = ' '  # Clear message output
        return  # leave function

    if msg[0] == 'mStart':  # Movement startet through local start button
        msg = 'wait'  # Wait till position is reached
        while msg == 'wait':
            msg = mail.receive()
            if msg == None:
                msg = 'wait'
        pass

    if msg[0] == 'mMoveAborted':  # Movement aborted
        lT1Init['bg'] = 'red'  # Turn indicator for homing red
        lT1Pos['text'] = '?'  # Clear Output for Position T1
        lMessageBox['text'] = 'Bewegung T1 abgebrochen'  # print message

    if msg[0] == 'mMoveCompleted':  # Movement finished
        T1Pos = T1newPos  # Safe new position of T1
        lMessageBox['text'] = 'Bewegung T1 abgeschlossen'  # print message
        lT1Pos['text'] = str(T1Pos) + " m"  # Update label for position

        bT1SetAcc['state'] = 'normal'  # Enabel buttons for T1 control
        bT1SetSpeed['state'] = 'normal'
        bT1SetPos['state'] = 'normal'
        eT1Acc['state'] = 'normal'
        eT1Speed['state'] = 'normal'
        eT1Pos['state'] = 'normal'

def T1DontMove():

    bT1Move['state'] = 'disabled'  # Reset Buttons for T1 control
    bT1MoveLocal['state'] = 'disabled'
    bT1DontMove['state'] = 'disabled'
    bT1SetAcc['state'] = 'normal'
    bT1SetSpeed['state'] = 'normal'
    bT1SetPos['state'] = 'normal'
    eT1Acc['state'] = 'normal'
    eT1Speed['state'] = 'normal'
    eT1Pos['state'] = 'normal'
    lMessageBox['text'] = ' '  # Clear message output

def T1SetPos():

    if eT1Pos.get() == '':  # leave function if nothing was written in entry
        return

    newPos = round(float(eT1Pos.get()), 3)  # Get entry

    if newPos == T1Pos:  # Leave function if new position is identical to old one
        eT1Pos.delete(0, 'end')  # Clear entry
        return

    if not (param.HOMET1_REF <= newPos <= (param.HOMET1_REF + param.T1LENGTH)):  # Leave function if new position is invalid
        lMessageBox['text'] = ('Fehler: Position ausserhalb von ' + str(param.HOMET1_REF) + ' - ' +
                               str(round((param.HOMET1_REF + param.T1LENGTH), 3)) + ' m')  # Print valid postions
        return

    eT1Pos.delete(0, 'end')  # Clear entry
    MoveCalc(newPos)  # Hand over new position to MoveCalc()

def T1SetAcc():

    global T1Acc

    if eT1Acc.get() == '':  # leave function if nothing was written in entry
        return

    newAcc = round(float(eT1Acc.get()), 3)  # Get entry

    if not (0 < newAcc <= param.T1MAXACC):  # Leave function if new acceleration is invalid
        lMessageBox['text'] = ('Fehler: Beschleunigung ausserhalb von 0.001 und ' +
                               str(param.T1MAXACC) + ' m/s²')  # Print valid acceleration
        return

    T1Acc = newAcc  # Safe new acceleration and transmit to controller
    mail.send('mSetAcc', 1, T1Acc/param.P1)
    lT1Acc['text'] = str(T1Acc) + ' m/s²'  # Update label for acceleration
    eT1Acc.delete(0, 'end')  # Clear entry

def T1SetSpeed():

    global T1Speed

    if eT1Speed.get() == '':  # leave function if nothing was written in entry
        return

    newSpeed = round(float(eT1Speed.get()), 3)  # Get entry

    # Leave function if new speed is invalid
    if not (param.T1DEFSPD <= newSpeed <= param.T1MAXSPD):
        lMessageBox['text'] = ('Fehler: Geschwindigkeit ausserhalb von ' + str(
            param.T1DEFSPD) + ' - ' + str(param.T1MAXSPD) + ' m/s')  # Print valid speed
        return

    T1Speed = newSpeed  # Safe new speed and transmit to controller
    mail.send('mSetSpeed', 1, T1Speed/param.P1)
```

```
    lT1Speed['text'] = str(T1Speed) + ' m/s'  # Update label for speed
    eT1Speed.delete(0, 'end')  # Clear entry

def R1Move():

    global R1Pos

    if eR1Pos.get() == '':  # leave function if nothing was written in entry
        return

    newPos = round(float(eR1Pos.get()), 1)  # Get and clear entry
    eR1Pos.delete(0, 'end')

    if newPos > abs(90):  # Check if new position is valid
        # print out valid postitions
        lMessageBox['text'] = 'Fehler: Positionseingabe nur von -90° bis +90°'
        return

    Steps = StepsR(newPos-R1Pos)  # Calculate steps for movement
    # Transmit steps to controller and start movement
    mail.send('mSetSteps', 2, Steps)
    mail.send('mStart', 2)

    msg = 'wait'  # Wait till position is reached
    while msg == 'wait':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass
    if msg[0] == 'mMoveAborted':  # Movement aborted
        bR1Move['state'] = 'disabled'  # Disable button for R1 movement
        eR1Pos['state'] = 'disabled'  # Disable Entry for R1 position
        lR1Pos['text'] = '?'  # Clear Output for Position R1
        lR1Init['bg'] = 'red'  # Turn indicator for homing red
        # Print out message
        lMessageBox['text'] = 'Positionierung R1 abgebrochen'

    if msg[0] == 'mMoveCompleted':  # Movement finished
        R1Pos = newPos  # Safe new position
        # Print out message
        lMessageBox['text'] = 'Positionierung R1 abgeschlossen'
        lR1Pos['text'] = str(R1Pos) + "°"  # Update label for R1 position

def R2Move():

    global R2Pos

    if eR2Pos.get() == '':  # leave function if nothing was written in entry
        return

    newPos = round(float(eR2Pos.get()), 1)  # Get and clear entry
    eR2Pos.delete(0, 'end')

    if newPos > abs(90):  # Check if new position is valid
        # Print out valid postitions
        lMessageBox['text'] = 'Fehler: Positionseingabe nur von -90° bis +90°'
        return

    Steps = StepsR(newPos-R2Pos)  # Calculate steps for movement
    # Transmit steps to controller and start movement
    mail.send('mSetSteps', 3, Steps)
    mail.send('mStart', 3)

    msg = 'wait'  # Wait till position is reached

    while msg == 'wait':
        msg = mail.receive()
        if msg == None:
            msg = 'wait'
    pass
    if msg[0] == 'mMoveAborted':  # Movement aborted
        bR2Move['state'] = 'disabled'  # Disable button for R2 movement
        eR2Pos['state'] = 'disabled'  # Disable Entry for R2 position
        lR2Pos['text'] = '?'  # Clear Output for Position R2
        lR2Init['bg'] = 'red'  # Turn indicator for homing red
        # Print out message
        lMessageBox['text'] = 'Positionierung R2 abgebrochen'

    if msg[0] == 'mMoveCompleted':  # Movement finished
        R2Pos = newPos  # Safe new position
        # Print out message
        lMessageBox['text'] = 'Positionierung R2 abgeschlossen'
        lR2Pos['text'] = str(R2Pos) + "°"  # Update label for R1 position

def StepsR(degree):

    Steps = round(degree/param.P2)  # Calculate steps for rotation
    return Steps

def StepsT(distance):

    Steps = round(distance/param.P1)  # Calculate steps for linear movement
    return Steps

def MoveCalc(Pos):

    global T1Acc
    global T1Speed
```

```
    global T1Pos
    global T1newPos

    T1newPos = Pos  # Get new position
    Steps = StepsT(T1newPos-T1Pos)  # Calculate steps for movement
    # Check if set speed will be reached
    StepsMin = pow((T1Speed/param.P1), 2)/(T1Acc/param.P1)      # s = V²/a

    if StepsMin >= abs(Steps):  # Set speed will not be reached
        # Calculate possible speed or required acceleration and print them out
        SpeedMax = round(pow((abs(Steps*param.P1)*(T1Acc)), 0.5), 2) # V = √(s/a)
        AccMin = round(((pow(T1Speed, 2))/abs(Steps*param.P1)), 2) # a = V²/s
        if AccMin <= param.T1MAXACC:
          lMessageBox['text'] = 'Warnung: Geschwindigkeit wird bei dieser Distanz nicht erreicht. Vmax = ' + \
          str(SpeedMax) + ' m/s oder amin = ' + str(AccMin) + ' m/s²'
        else:
           lMessageBox['text'] = 'Warnung: Geschwindigkeit wird bei dieser Distanz nicht erreicht. Vmax = ' + \
          str(SpeedMax) + ' m/s'
    else:
        # If set speed can be reached print out set new position
        lMessageBox['text'] = 'Position ' + str(T1newPos) + ' m gesetzt'

    # Transmit steps for T1 movement to controller
    mail.send('mSetSteps', 1, -Steps)
    bT1Move['state'] = 'normal'  # Enable buttons for start of T1 movement
    bT1MoveLocal['state'] = 'normal'
    bT1DontMove['state'] = 'normal'
    bT1SetAcc['state'] = 'disabled'  # Disable buttons for T1 control
    bT1SetSpeed['state'] = 'disabled'
    bT1SetPos['state'] = 'disabled'
    eT1Acc['state'] = 'disabled'
    eT1Speed['state'] = 'disabled'
    eT1Pos['state'] = 'disabled'

# ----------------------GUI Construction-------------------------------

# Hauptfenster erzeugen
main = tk.Tk()

# Optik Haupfenster
main.title('Sensorteststand Steuerung')
main.geometry('720x460')
main.resizable(width=False, height=False)

# Labels erzeugen
lInitHead = tk.Label(main, text='Homing', font=('Calibri', 12, 'bold'))
lT1Init = tk.Label(main, text='T1', bg='red')
lR1Init = tk.Label(main, text='R1', bg='red')
lR2Init = tk.Label(main, text='R2', bg='red')

lT1Head = tk.Label(main, text='Steuerung T1', font=('Calibri', 12, 'bold'))
lT1AccHead = tk.Label(main, text='T1 Beschleunigung')
lT1NewAccHead = tk.Label(main, text='T1 Beschleunigung neu')
lT1SpeedHead = tk.Label(main, text='T1 Geschwindigkeit')
lT1NewSpeedHead = tk.Label(main, text='T1 Geschwindigkeit neu')
lT1PosHead = tk.Label(main, text='T1 Position')
lT1NewPosHead = tk.Label(main, text='T1 Position neu')
lT1Acc = tk.Label(main, relief='ridge', text=str(T1Acc) + ' m/s²', width=4)
lT1Speed = tk.Label(main, relief='ridge', text=str(T1Speed) + ' m/s', width=4)
lT1Pos = tk.Label(main, relief='ridge', text='?', width=4)

lR1Head = tk.Label(main, text='Steuerung R1', font=('Calibri', 12, 'bold'))
lR1PosHead = tk.Label(main, text='R1 Position')
lR1NewPosHead = tk.Label(main, text='R1 Position neu')
lR1Pos = tk.Label(main, text='?', relief='ridge', width=4)

lR2Head = tk.Label(main, text='Steuerung R2', font=('Calibri', 12, 'bold'))
lR2PosHead = tk.Label(main, text='R2 Position')
lR2NewPosHead = tk.Label(main, text='R2 Position neu')
lR2Pos = tk.Label(main, text='?', relief='ridge', width=4)

lEmpty = tk.Label(main, text=' ')
lEmpty2 = tk.Label(main, text=' ')
lMessageBoxHead = tk.Label(
    main, text='Letzte Meldung:', font=('Calibri', 12, 'bold'))
lMessageBox = tk.Label(main)

# Buttons erzeugen
bT1Init = tk.Button(main, text='Init T1', command=T1Init)
bR1Init = tk.Button(main, text='Init R1', command=R1Init)
bR2Init = tk.Button(main, text='Init R2', command=R2Init)

bT1SetAcc = tk.Button(main, text='Set', command=T1SetAcc, state='disabled')
bT1SetSpeed = tk.Button(main, text='Set', command=T1SetSpeed, state='disabled')
bT1SetPos = tk.Button(main, text='Set', command=T1SetPos, state='disabled')
bT1Move = tk.Button(main, text='Start', command=T1Move, state='disabled')
bT1DontMove = tk.Button(main, text='Abbruch',
                        command=T1DontMove, state='disabled')
bT1MoveLocal = tk.Button(main, text='Start Lokal',
                         command=T1MoveLocal, state='disabled')

bR1Move = tk.Button(main, text='Set R1', command=R1Move, state='disabled')
bR2Move = tk.Button(main, text='Set R2', command=R2Move, state='disabled')

# Entries erzeugen
eT1Acc = tk.Entry(main, width=10, state='disable')
eT1Speed = tk.Entry(main, width=10, state='disable')
eT1Pos = tk.Entry(main, width=10, state='disable')
```

```
eR1Pos = tk.Entry(main, width=10, state='disable')
eR2Pos = tk.Entry(main, width=10, state='disable')

# Widgets anordnen
# Row0
lInitHead.grid(row=0, column=0, padx=0, pady=10, columnspan=3)
lEmpty2.grid(row=0, column=3, padx=20, pady=10)
lT1Head.grid(row=0, column=4, padx=0, pady=10, columnspan=3)

# Row1
lT1Init.grid(row=1, column=0, padx=15, pady=0)
lR1Init.grid(row=1, column=1, padx=15, pady=0)
lR2Init.grid(row=1, column=2, padx=15, pady=0)
lT1AccHead.grid(row=1, column=4, padx=10, pady=0)
lT1NewAccHead.grid(row=1, column=5, padx=10, pady=0)

# Row2
bT1Init.grid(row=2, column=0, padx=15, pady=10)
bR1Init.grid(row=2, column=1, padx=15, pady=10)
bR2Init.grid(row=2, column=2, padx=15, pady=10)
lT1Acc.grid(row=2, column=4, padx=15, pady=10, ipadx=20)
eT1Acc.grid(row=2, column=5, padx=15, pady=10)
bT1SetAcc.grid(row=2, column=6, padx=0, pady=10)

# Row3
lR1Head.grid(row=3, column=0, padx=15, pady=10, columnspan=3, sticky='s')
lT1SpeedHead.grid(row=3, column=4, padx=10, pady=10)
lT1NewSpeedHead.grid(row=3, column=5, padx=10, pady=10)

# row4
lR1PosHead.grid(row=4, column=0, padx=15, pady=0)
lR1NewPosHead.grid(row=4, column=1, padx=15, pady=0)
lT1Speed.grid(row=4, column=4, padx=15, pady=10, ipadx=20)
eT1Speed.grid(row=4, column=5, padx=15, pady=10)
bT1SetSpeed.grid(row=4, column=6, padx=0, pady=10)

# Row5
lR1Pos.grid(row=5, column=0, padx=15, pady=10, ipadx=20)
eR1Pos.grid(row=5, column=1, padx=15, pady=10,)
bR1Move.grid(row=5, column=2, padx=15, pady=10)
lT1PosHead.grid(row=5, column=4, padx=10, pady=10)
lT1NewPosHead.grid(row=5, column=5, padx=10, pady=10)

# Row6
lR2Head.grid(row=6, column=0, padx=15, pady=10, columnspan=3, sticky='s')
lT1Pos.grid(row=6, column=4, padx=15, pady=10, ipadx=20)
eT1Pos.grid(row=6, column=5, padx=15, pady=10)
bT1SetPos.grid(row=6, column=6, padx=0, pady=10)

# Row7
lR2PosHead.grid(row=7, column=0, padx=15, pady=0)
lR2NewPosHead.grid(row=7, column=1, padx=15, pady=0)

# Row8
lR2Pos.grid(row=8, column=0, padx=15, pady=10, ipadx=20)
eR2Pos.grid(row=8, column=1, padx=15, pady=10,)
bR2Move.grid(row=8, column=2, padx=15, pady=10)
bT1MoveLocal.grid(row=8, column=4, padx=15, pady=10)
bT1Move.grid(row=8, column=5, padx=15, pady=10)
bT1DontMove.grid(row=8, column=6, padx=0, pady=10)

# RowLast-2
lEmpty.grid(row=48, column=0)

# RowLast-1
lMessageBoxHead.grid(row=49, column=0, columnspan=7)

# RowLast
lMessageBox.grid(row=50, column=0, columnspan=7)

# ------------------------Loop-----------------------

main.mainloop()
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Dresden, den 31.03.2023

Robert Lublow