Hochschule Aalen

University of Aalen

Mechatronic Project - Electronic Lead Screw

February 2022 - Lukas Schwörer

Preface

This project is part of the master's degree program "System engineering" of Aalen University and is scheduled to be performed during the first two semesters of the program. This report covers the work realized from April 2021 to February 2022.
The practical work and the writing for this project was performed from home due to the Covid-19 Pandemic.

Dieses Projekt ist Teil des Masterstudiengangs "System Engineering" der Hochschule Aalen und muss während der ersten beiden Semester absolviert werden. Die in diesem Bericht beschriebene praktische Arbeit wurde von April 2021 bis zum Februar 2022 realisiert.
Die praktische Arbeit, wie auch das Schreiben des Berichts wurde Zuhause ausgeführt aufgrund der Covid-19 Pandemie.

Abstract

This Project is about the development, setup, testing and qualification of an Electronic Lead screw (ELS). This project was proposed to the university by myself because of the limited possibilities of practical projects due to the Covid-19 pandemic. Its aim is to develop a system to replace the gearbox inside a conventional lathe, which will synchronize the rotation of the Lead screw to the rotation of the main spindle. The ELS need to be able to keep up with the spindle rotation during conventional turning with different feeds and speeds. In addition to this, it should be possible to cut precise metric and imperial threads.

The electromechanical system of the ELS is build around an encoder to read the rotational position of the main spindle and a servo motor to control the position of the Lead screw. A microcontroller computes the information gathered by the encoder and commands the servo-motor to the correct positions.

To be able to easily change and add features as well as to predict the behavior of the system, the development of the ELS needs to be model based. This model needs to incorporate all aspects of the system including the spindle, the encoder, the microcontroller and the servo motor. As comparison, the conventional gearbox should also be modeled.

Kurzfassung

Dieses Projekt beschäftigt sich mit der Entwicklung, dem Aufbau, dem Testen und Qualifizieren einer Elektronischen Leitspindel (ELS). Dieses Projekt wurde der Universität von mir vorgeschlagen, da es aufgrund der Covid-19 Pandemie nur begrenzt möglich war praktische Projekte durchzuführen. Sein Ziel ist es ein System zu entwickeln, dass das Getriebe in einer konventionellen Drehbank ersetzt und die Rotation der Leitspindel zu der Rotation der Hauptspindel synchronisiert. Die ELS muss fähig sein, mit der Rotation der Hauptspindel mitzuhalten während einer konventionellen Drehbearbeitung mit unterschiedlichen Drehzahlen und Vorschüben. Zusätzlich muss es möglich sein, präzise metrische und imperische Gewinde herzustellen.

Das elektromechanische System der ELS besteht aus einem Encoder, der die Position der Hauptspindel ausliest und einem Servomotor, der die Position der Leitspindel kontrolliert. Ein Mikrocontroller verarbeitet die vom Encoder gesammelten Informationen und bestimmt die korrekte Position des Servomotors.
Um ein einfaches Ändern und Entfernen von Features zu ermöglichen und das Verhalten des Systems vorherzusagen, muss die Entwicklung der ELS Modellbasiert durchgeführt werden. Dieses Modell muss alle Komponenten des reellen Systems beinhalten, eingeschlossen der Spindel, des Encoders, dem Mikrocontroller und dem Servomotor. Zum Vergleich sollte auch ein System mit einem konventionellen Getriebe modelliert werden.

Acknowledgement

At this point I would like to thank the following people who made this project possible:

- **Prof. Dr. Markus Glaser** For supervising and supporting my project.
- **James Clough (Clough42)** For helping me to find hardware matching my specifications and supplying me with the great documentation about his own ELS project.

Table of Contents

1. Introduction

This chapter will highlight the working environment, the project background as well as its aim.

1.1. Working environment

1.1.1. University of Aalen

The university of Aalen was founded in 1962 and by now is one of the leading university's of applied sciences in Germany. It has a focus in technical and economic research and has approximately 6000 students. The University is located in Aalen, a smaller city in the south of Germany. The University has ongoing cooperations with many companies in the region such as Zeiss, Mapal and Trumpf. Further, Aalen University has international cooperations with 136 other Universities.[1]

1.1.2. Workshop

The practical part of this project was carried out in my home workshop. The workshop has tools for basic metal work, wood work, a 3D-Printing as well as some tools for the development of electronics. Further, equipment for welding, conventional milling and turning is available.

1.2. Project background

The mechatronic project is fixed part of the masters program "System Engineering" at Aalen University. It is supposed to cover both practical and theoretical work for solving a mechatronic problem.
In the theoretical part of the project, the problem needs to be split in smaller individual issues. In the next step, solutions for each of the issues need to be found by thoroughly analyzing the requirements of the projects.
In the practical part of the project, the previously discovered issues and the solutions for it are supposed to be translated and tested to their functionality in practice.
This development process can be summarized in the so called V-Modell. The V-Modell is shown in 1.1.
The motivation and idea of this project were founded by my interest in metal work as well as online sources about similar projects.[2]

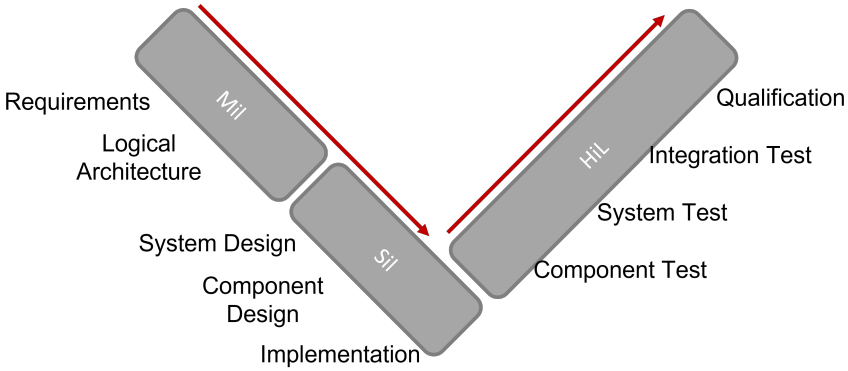


Figure 1.1.: V-Model

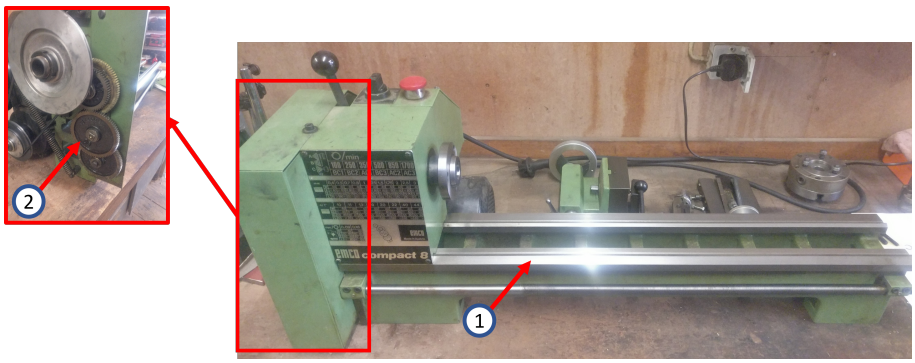


Figure 1.2.: Image of the Emco Compact 8 and the manual gearbox; 1 - Emco Compact 8, 2 - Manual gearbox

1.2.1. Aim of study

The aim of this project is to add the Electronic Lead screw system to the Emco Compact 8 mini Lathe (see Appendix A) in a way that is reversible. This is important in order to remain the value of the lathe. Further, the ELS system is supposed to be an addition that speeds up and simplifies the manual operation of the lathe by replacing the manual gearbox with electronics. For this reason, it is not planned to add servos to the other axis of the lathe in the future. Both, the Lathe as well as the Gearbox that is going to be replaced are shown in Figure 1.2.

2. Theoretical Background

This chapter will discuss the theoretical background knowledge that was part of the decision-making during the development process.

2.1. Numerical Mathematics

The science of the numerical mathematics is a branch of mathematics that concentrates on the research of solving continuous problems with discrete systems, such as a computer. One of the main issues in numerical mathematics is the representation of fractional numbers. This is because computers can only represent a finite subset of decimals, due to memory and processing limitations. This can lead to rounding errors that can have a significant impact on the precision of any calculation with fractional numbers. Further, calculations with numbers that have a large amount of decimal places, require more calculation steps by the processor and therefore slow down the computation.[3]

2.1.1. Floating-Point Arithmetic

One way to represent fractional number is the floating point number system. With this system are stored in memory in two parts: the significant and its base. The significant represents the number as an integer, without the decimal point. The base (10^x) describes the position of the decimal point in the original number.[3]

2.1.2. Fixed-Point Arithmetic

The fixed point representation of numbers on a numerical system, defines the fractional number with a fixed amount of rational numbers. Further, the amount of significant numbers and decimal places is fixed. The intention behind this number representation is to simplify and therefore speed up numerical calculations.[3]

2.2. Manufacturing Methods

Nowadays, engineers have a broad spectrum of different manufacturing methods available when it comes to producing a part. The decision which to choose should incorporate component specific properties such as the desired amount to be produced, the general shape and material of the part, as well as the environment the part will be used in.

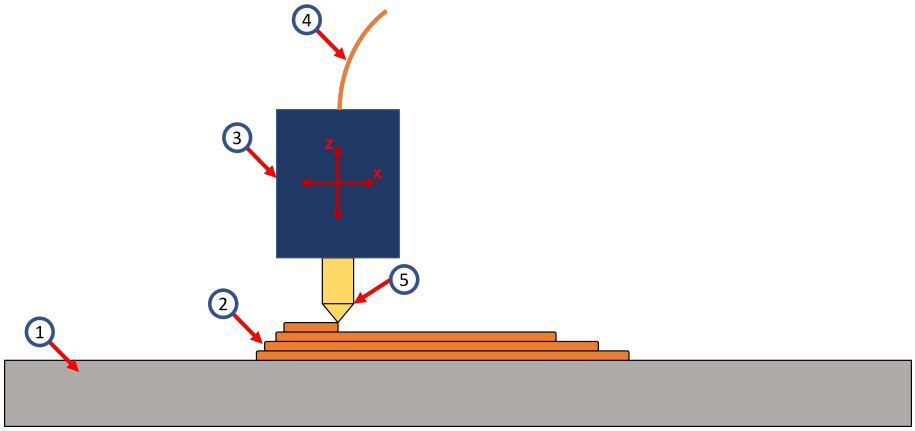


Figure 2.1.: 2D Drawing of the 3D-Printing Process; 1 - Print bed, 2 - Plastic layers, 3 - Print head, 4 - Plastic filament, 5 - Nozzle

2.2.1. Additive Manufacturing

Additive manufacturing (AM) is a process, where material is build up in order to create the desired shape. AM has first appeared in 1981, with the first commercial available machine in 1988. Additive manufacturing incorporates a wide variety of different manufacturing methods. The most well know Additive manufacturing method today probably is Fused Deposition Modeling (FDM), also known as 3D-Printing. With FDM, a thin thermoplastic filament is molten in a "Print head" and then squeezed through a nozzle. The molten plastic is then laid down in layers onto a print bed, in order to create a 3D shape. This process is shown in Figure 2.1. The core advantages of FDM are its speed and flexibility. This makes prototyping and small production batches very cost-effective and easy. Further, because 3D-Printing is a layer by layer process, almost any shape can be created, this is especially important for shapes with undercuts and included geometry's.[4]

2.2.2. Subtractive Manufacturing

Subtractive manufacturing is a process, where material is removed from a stock material in order to create a 2D or 3D shape. Two of the most common subtractive manufacturing processes are turning and milling.
Turning is a process where the stock material is rotated in a lathe and cut with a non-rotating tool. The tool itself only moves in X and Z direction. This process is shown in Figure 2.2. For this reason, turning is especially well suited for the production of round parts.

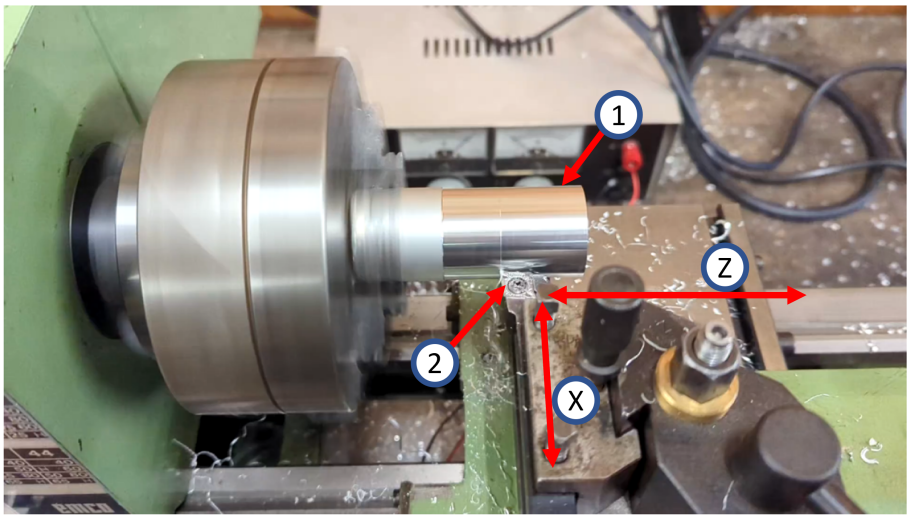


Figure 2.2.: Picture of a turning process; 1 - Rotating work pice, 2 - Cutting tool, X - X Axis, Z - Z axis

3. Hardware and Software

This chapter will give an in-depth look into the hardware and software used to complete the project. This knowledge is necessary to understand the design decisions made.

3.1. Hardware

3.1.1. Rotary Encoder

A rotary encoder is an electromechanical transducer to convert a rotary movement into an electrical signal. One of the most commonly used rotary encoders is the optical rotary encoder. Its working principle is based on a Light emitting diode and a photo sensitive device, such as a photo-transistor or photo-resistor. The diode and the photo-sensitive device are located on opposite sides of a light blocking disk. This is illustrated in Figure 3.1.
The light blocking disk is perforated with small slits in a rotational symmetric and regular pattern. As soon as the disc is rotated between the diode and the photo-sensitive device, the light is transmitted and blocked in an alternating pattern. This creates a square wave on the output of the photo-sensitive device, which frequency is dependent on the rotational speed of the disk.[5]

3.1.2. Stepper Motors

A stepper motor is an electrical, synchronous motor. Its rotor follows the magnetic field created in the stator exactly. In contrast to commonly known DC or AC motors, a stepper motor has a minimum angle that it needs to be rotated. This has the big advantage, that the position is always a multiple of this angle, the so-called step angle. Therefor, a sensor is not necessary to rotate the motor by a very precise amount. A stepper motor is a cheap alternative to closed loop controlled motors. Here, an additional sensor has to determine the positional error of the motor in order to move the motor precisely.
The disadvantage of stepper motors usually is their low rpm limit. At a certain speed, the rotor of the motor is no longer able to follow the magnetic field created by the stator. This can also happen when the motor is under high load. When this happens, the motor will skip steps. Therefore, stepper motors are often combined with rotational sensors in order to detect skipped steps and compensate for it.[6]

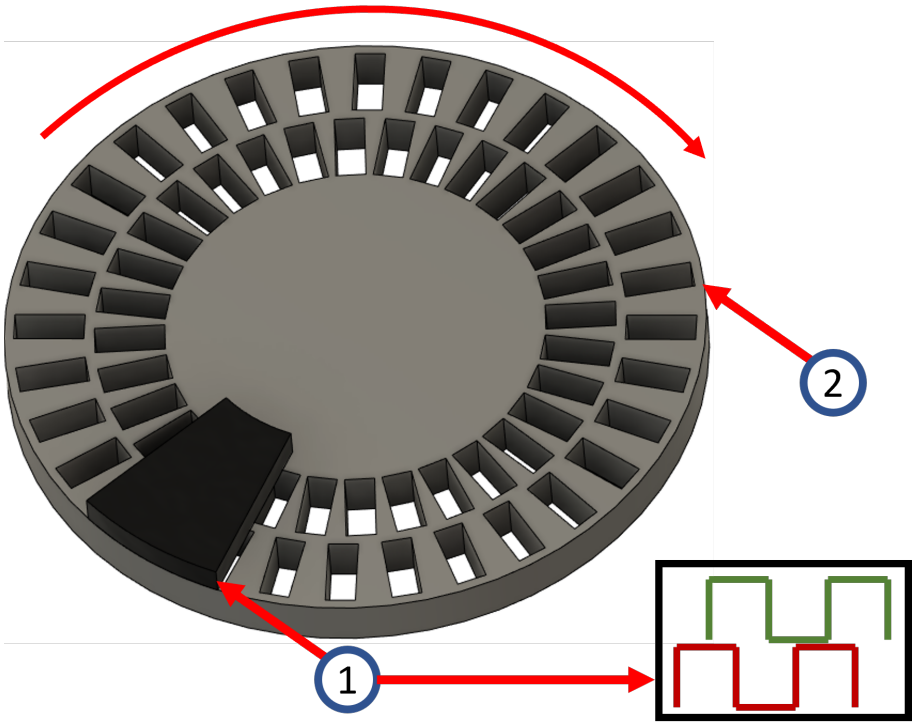


Figure 3.1.: Illustration of an optical encoder; 1 - Detector unit and detector unit signal, 2 - Encoder disk

3.1.3. Closed Loop Servos

Closed Loop Servos are an electromechanical actuator with a continuous feedback loop between its output (speed, torque, position, etc...) and input (Voltage, Current). Servos require additional electronics to compute the difference between its desired output and its current state. These electronics are typically referred to as "controller" or "driver". Servos are used in a wide variety of mechatronic applications when high precision, reliability or dynamics are required.[7]

3.1.4. Microcontroller

A microcontroller is an integrated circuit designed to serve a specific task in a system. Microcontroller consist of a processing unit, memory as well as In- and outputs. Within a system, a microcontroller can be used for a wide variety of tasks such as communication, data acquisition or actuator control. For mechatronic tasks, microcontrollers are commonly used as a so called embedded system. An embedded system is designed around a microcontroller and a specific task within a project.[8]

TI LaunchXL F280049C

The Texas Instruments Launchpad F280049C is a prototyping board designed around the Texas Instruments Picolo F280049C Real-Time microcontroller as an embedded system. Its design specifically targets highly dynamic control tasks while maintaining a

low unit cost. The Picolo F280049C is a 32bit floating point microcontroller with 256 KB Flash memory, 100 KB RAM and an operation frequency of 100MHz. Further, the microcontroller has hardware support for up to two rotary encoders, a wide variety of communication protocol's as well as general purpose I/O. In addition, the LaunchPad development board build around the microcontroller offers a debugging probe. This probe can be used to flash the microcontroller with software or observe and direct the operation of the microcontroller. This way, it is for example possible to stop the execution of the program on the microcontroller to access and change the value of every bit in every register of the microcontroller.[9]

Logic Level Shifter

A logic level shifter is an electronic circuit designed to convert one logic voltage level into another one, while maintaining the signal integrity. In modern mechatronic systems, it is often necessary that different electronic systems can communicate with each other. In some cases, however, it is not possible that all components work on the same voltage level. This makes it inevitable to use a logic level shifting circuit.[10]

3.1.5. Raspberry Pi

A Raspberry Pi is a Single board computer that features a full graphic operating system based on Linux. Further, it offers wireless connectivity as well as general purpose I/O's. The Raspberry Pi is mostly open source and very cost-effective. Because it has a very wide customer base, the Raspberry Pi has a wide variety of hardware and software add-ons available.[11]

3.2. Software

3.2.1. Matlab and Matlab-Simulink

MATrix LABoratory (MATLAB) is an Integrated Development Environment (IDE) and programming language. MATLAB was developed in the need of a numerical algebraic system at the university of New Mexico. On this base, the company MathWorks (see A) was created. Since 1984 MathWorks is further developing MATLAB and additional software for numerical algebraic computing. To support different hardware packages MathWorks offers so-called toolboxes. These toolboxes contain functions and scripts for communication, data acquisition, motion control etc. MATLAB is mainly used in technical development and research.[12]

3.2.2. Programing Language C

C is a high level programming language. It is most commonly used to write applications for embedded systems. It is chosen over low level programming languages such as Assembly because of the increasing complexity of programs for embedded systems. C is

a compiled programming language. This means that a program needs to be translated (compiled) to machine code before it can be executed. This translation is done with a program called "compiler". Compared to lower level languages such as Assembly, the code reusability of C has proven to be a step forward in the development of embedded systems.[13]

3.2.3. Programing Language Python

Python is a high level object-oriented programming language. In contrast to C, Python is an interpreted language. This means, an Interpreter translates source code into machine commands in real time, while the program is executed. Python features a good combination of capability and ease of uses. This makes the language easy to learn for beginners while remaining very powerful. Python further features a huge variety of proprietary and community build support packages and is based on the one of the most used programming languages used today.[14]

Kivy

Kivy is an open source Python library for the rapid development of graphical user interfaces and apps. It is script based and its scripting language can be directly implemented in Python. Further, Kivy supports use with touch screens. During this project, Kivy was used in order to develop the software for the Human Machine Interface.[15]

3.2.4. Code Composer Studio

Code Composer Studio is an Integrated Development Environment produced by Texas Instruments. It presents an interface to the Hardware produced by Texas Instruments and offers optimizing compilers for C and C++. Further, Code Composer Studio features support packages for the development and deployment of software to Texas Instruments Hardware, such as the TI LaunchXL F280049C. These support packages, in combination with a hardware debugging probe, enable the programmer to interact with the processor in real time.[16]

3.2.5. Git

Git is a software tool for source code management and versioning, as well as for parallel development. Git was developed by Linus Torvald in 2005. Git was developed in the need of source code management software for the development of the operating system Linux. Git allows development in so-called branches. These are independent copies of an already existing and probably used software. This ensures that modifications or enhancements are not affecting already used software. If a feature is finished, it can be merged back into the higher-level branch. Merging is the process of bringing two files, directories or branches together. Different developers, working on different

features of the same project, is called parallel development. Further, Git is a tool for software versioning. Software versioning is a tracking of changes in a project. In case of a mistake, the project can be recovered to every tracked point.
During this project, Git was very heavily used for software versioning as well as backup tool. This way, it was possible to revert changes to the software with very little work.

4. Experimental

This chapter will show the complete development process of the Electronic Lead screw and how it was executed based on the V-Modell.

4.1. Requirements and Logical Architecture

In this section, the development process in the first part of the V-Modell will be described. As shown in figure 4.1 this part is split into two sections. The requirements describe the fundamental properties of the system in order to function. The system Logical Architecture describes how the different system components are supposed to interact with each other.

4.1.1. Requirements

The development of the System requirements was the first step in the development process. This first step is one of the most important steps in the development process, because it defines the functions and properties of all System components. A selection of the requirements for the Human Machine Interface (HMI), the microcontroller (μC), the motor and the motor controller are shown in Table 4.1. The full list of requirements can be found in Appendix B.

4.1.2. Logical Architecture

The next step in the development process of the ELS was to create the Logical Architecture of the System. This was done as a Top level block diagram as shown in Figure 4.2.

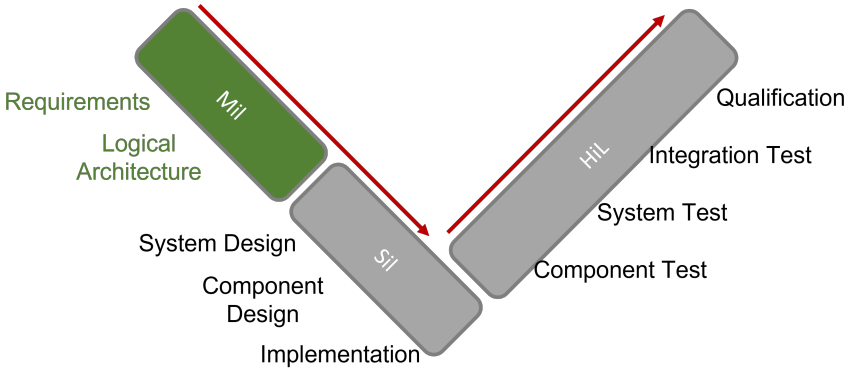


Figure 4.1.: V Model Requirements Stage; Green - Development status

Req. Nr.	HMI	μC	Motor	Motor Controller
1	Touch	Real Time	Max. Torque	Closed Loop
2	Modular	Matlab Code Gen.	Min. Torque	Supply Voltage
3	UART Com.	UART Com.	Max. Speed	Programmable
4	Separate PS	Quad. Encoder	Min. Speed	Resolution
5	Mount	GPIO	Space Claim	Communication

Table 4.1.: Selection of Key Requirements for the ELS

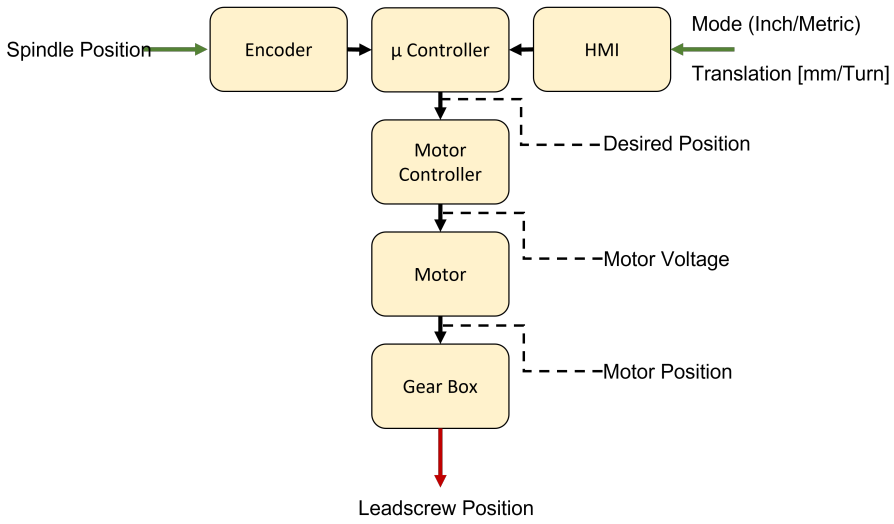


Figure 4.2.: Logical Architecture

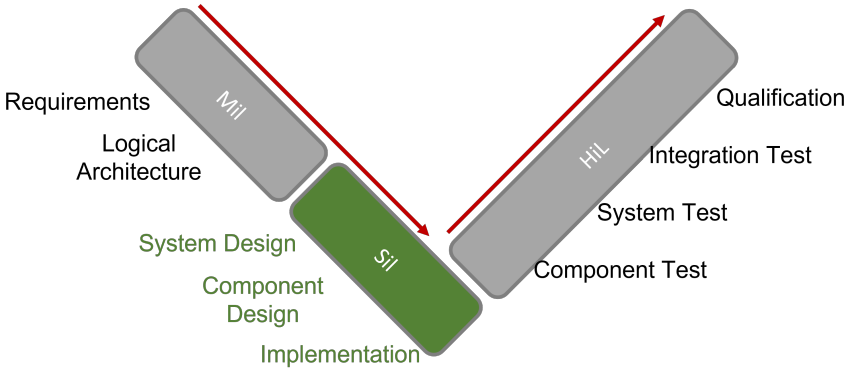


Figure 4.3.: V Model SIL Stage; green - Developement steps SIL stage

4.2. System Design and Implementation

This section describes the second half of the development process, divided into system and component design, as well as the integration of the components. This is the most time-consuming process of the development. For the system and component design, every function of each system and subsystem must be specified in detail. During the implementation phase, these functions are then transferred into the real system.
This development stage is represented by the Software in the Loop (SIL) stage and is shown in Figure 4.3.

4.2.1. System Design

The system design is executed based on the previously developed system requirements and its logical architecture. The system design was approached by filling the requirements list as well as the block diagram of the logical architecture with component specific parameters. These parameters either need to be defined or found during the system design process.
A good example for this process is the required torque of the Lead screw servo during a cutting process. This parameter can be found by measuring the power consumption of the AC motor for different cutting parameters with the old Lead screw drive train. However, while disassembling the original Lead screw drive train in order to find a good place to mount the servo motor, an easier way to determine the required motor torque was found. In order to protect the Lead screw, the manufacturer secured the gear, that is, driving the Lead screw with a brass pin (see Figure 4.4). According to the manufacturer, this pin will shear off when more than 1.2Nm is acting on the Lead screw. After this, all other parameters were determined similarly. The resulting logical structure is shown in Figure 4.5.

4.2.2. Component Design

During the component design phase, the function of each component is developed based on the previous development steps. However, the components are not yet put together

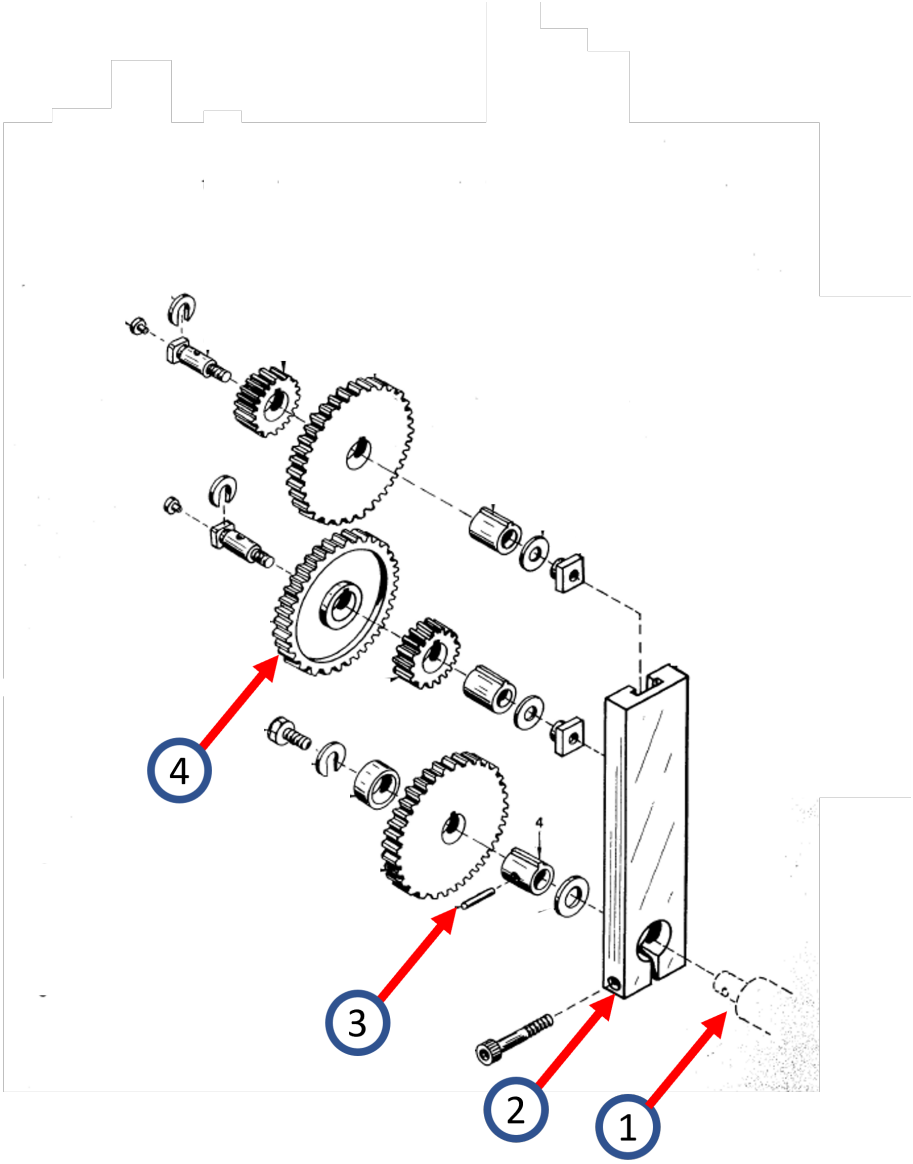


Figure 4.4.: Exploded view of the old drive train; 1 - Lead Screw, 2 - Gear Holder, 3 - Security Pin, 4 - Change Gears

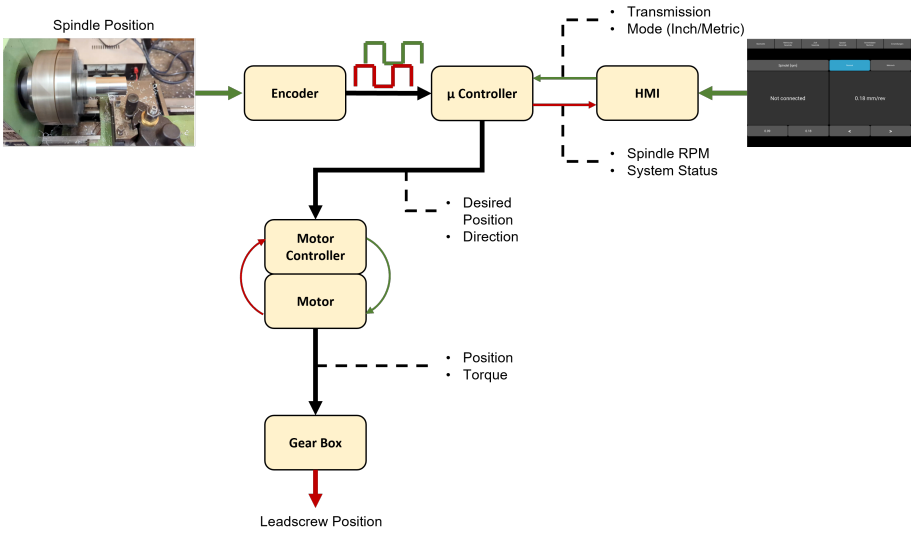


Figure 4.5.: Block diagram of the system design

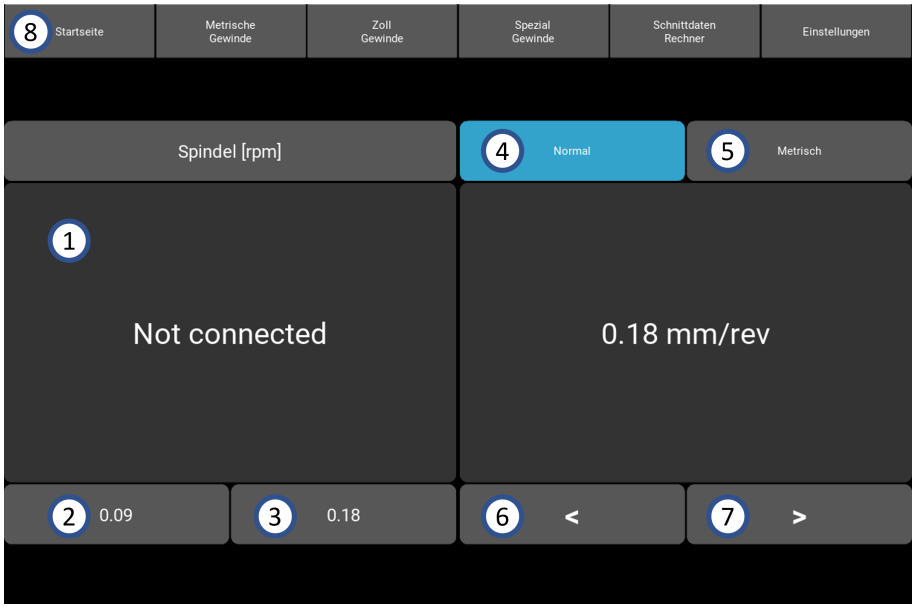


Figure 4.6.: Picture of the main HMI screen; 1 - Main spindle rpm display, 2+3 - Shortcuts to most common feed rates, 4+5 - Mode selection, 6 - Step back, 7 - Step forward, 8 - Submenus

as a system.

HMI

Because the HMI was required to be as modular as possible, a Raspberry Pi 4 (see Appendix A) was chosen as the main component. It was combined with a 7" capacitive Touchscreen that can be bought as an accessory for the Raspberry Pi.
The software for the HMI was developed a main screen that shows the current RPM of the main spindle as well as the currently selected feed. Further, the main screen incorporates buttons to change the current feed rate or mode. The mode dictates how the ELS is working. Three options are available:

- **Normal Mode** Mode for normal turning. The feed rate (mm/revolution) is selectable in 0.01 mm/rev increments.
- **Metric Mode** Mode for metric thread cutting. The step buttons allow to step through a list of predefined thread pitches.
- **Imperial Mode** Mode for imperial thread cutting. The step buttons allow to step through a list of predefined thread pitches (threads/inch).

In addition to that, the HMI offers submenus for all available thread pitches and various other functions. The programming of the graphical user interface was done with Kivy and Python (see Appendix A) The main screen of the HMI is shown in Figure 4.6.

Micro Controller

The first step of the design of the microcontroller was its selection. This was done based on the requirements and the work of James Clough. As a result, the microcontroller

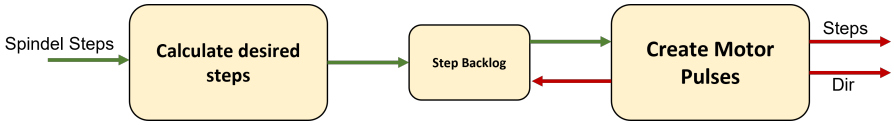


Figure 4.7.: Block diagram of the functional software on the TI Launchpad

platform Texas Instruments LaunchXL F280049C was selected (see Appendix A). Because of the Matlab support and the hardware ports for encoder, this was a perfect match for this project.
The software for the Texas Instruments microcontroller was developed in two parts. First, a detailed block diagram of the intended function of the software was created. This was done to keep the structure of the software as simple as possible. As shown in Figure 4.7, the functional software, running on the microcontroller, is build around two main functions. The first function calculates the number of desired steps based on the steps generated by the encoder on the main spindle. The second function generates the motor control signals. These are represented by a logical signal for the intended motor direction and a logical puls with a width of 1μs and a duty cycle of 50%. These actions need to be performed in real-time. This means that those functions need to be executed cyclic within a certain time limit, the so-called "real time requirement". In the case of the ELS, the real time requirement is the 1μs pulse length that is needed for the motor signal. However, this time frame is so short that the microcontroller could not execute both functions within the given real-time requirement. For this reason, two real time loops were created, one with a 100ms cycle time to calculate the Desired Steps, the other with a 1μs cycle time to generate the motor pulses. These functions work with a shared memory, the "step backlog". Every time the calculation of the desired steps is done, the steps in the backlog are incremented. Whenever the pulse function is finished with one pulse, the step backlog is decremented.

In the second part of the software development the previously discussed steps were translated into a Matlab-Simulink Model. This had two big advantages: First, in addition to the software model, it is possible to model the complete electromechanical system of the ELS. This is very helpful while testing and debugging the developed code. Second, it is possible to generate C code for the Texas Instruments microcontroller directly from the Simulink model. This function code is then guaranteed to work in the exact same way it did in the model environment inside of Simulink when implemented correctly. In Figure 4.8 the complete Simulink Model is shown.

Encoder

The design or in this case selection of the encoder was purely based on the previously defined requirements. The most significant requirements are shown in Table 4.2.
This led to the selection of an Encoder from the manufacturer Opkon (see Appendix A). The Encoder creates 4096 pulses/revolution, with half of them phase shifted by 90°in

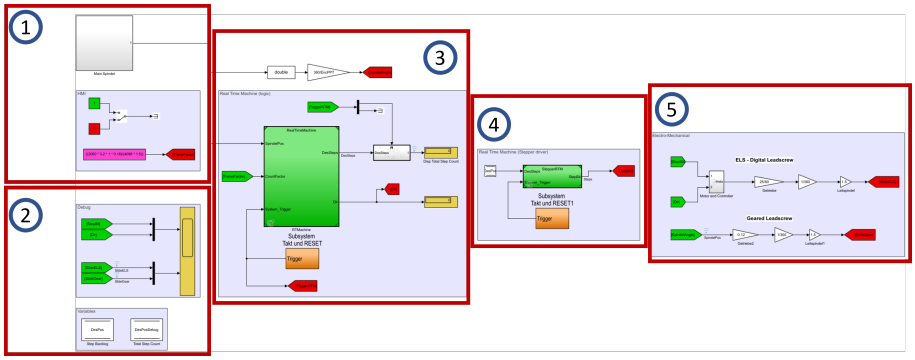


Figure 4.8.: Picture of the created Simulink Model; 1 - Model of the Spindle and the Encoder, 2 - Outputs of the Model, 3 - Model of the function calculating the desired Steps, 4 - Model of the function generating the motor pulses, 5 - Model of the Leadscrew with ELS and with the original drive train

Req. Nr.	Requirement	Value
1	Physical Size	60 mm
2	Supply Voltage	5V
3	Resolution	4096 ppt
4	Directional	Yes

Table 4.2.: Selection of Key Requirements for the ELS

order to determine the turning direction. Further, the encoder covers an input voltage range from 5V - 30V, which is compatible with the Encoder Pins on the Launchpad XL. With a high of 57 mm, the Encoder just fits into the previously determined specifications.

Motor and Motor Controller

The motor was selected based on the requirements list as well as recommendations from James Clough [2]. The selected motor is from the company Steppers Online (see Appendix A). It is a so-called "integrated servo motor". This means, the motor consist of a brushless DC-motor, an encoder as well as the corresponding motor controller. All three of these parts are packaged into one unit as shown in Figure 4.9.

The motor features an RPM range from 0 to 4000 rpm and a very constant maximum torque of 0.4 nm. Further, it has the right dimensions in order to fit the space claim requirements.
The motor controller in combination with the required software represent an easy programming interface. Via this interface, control parameters as well as parameter for the dynamics and precision of the motor can be set. As a starting value, the motor stiffness (responsiveness) gain was set to its maximum. Based on the software design of the microcontroller, the precision of the motor was set to 2000 steps/revolution.
The interface to the motor is provided by a simple digital protocol. It consists of a direction and a step pin. The direction reacts to a logical 5V signal, where a logical 1 (5V)

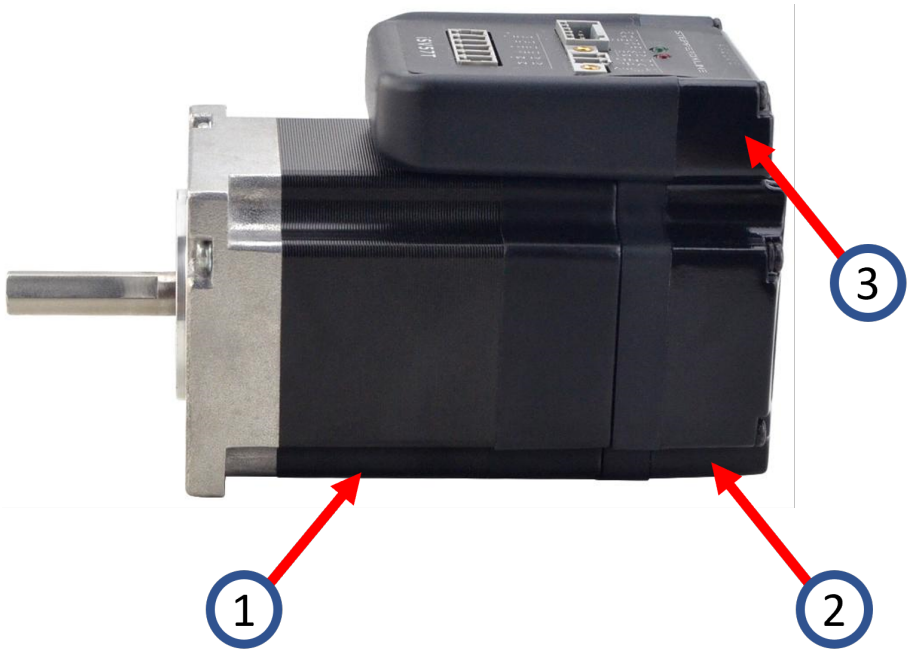


Figure 4.9.: Integrated Servo Motor; 1 - BLDC Motor, 2 - Encoder Unit, 3 - Control Unit

corresponds to clockwise rotation and a logical 0 (0V) corresponds to counterclockwise rotation. The step input reacts to logical pulses with a minimum pulse width of 1μs. Each pulse will then move the motor by $\frac{1}{motor-resulution}$.

Gearbox

Because the motor is mounted underneath the Lead screw, a coupling between both elements was needed. This was done with the already existing gears from the old gearbox of the lathe. This decision was made because it required the least modification (only a coupler from the motor to the gear had to be manufactured) and was the cheapest option. The gear ratio of 25/80 was selected to optimize the feed range of the lathe with the available RPM of the motor. This is shown in Figure 4.10.
The transmission of the gearbox also increases the torque of the motor by a factor of 3.2. This way, the motor meets the specified torque of 1.2 Nm.

4.2.3. Software Implementation

The task during the implementation of the software was to deploy the developed code to the according hardware. Further, the communication between the HMI and the real time controller needed to be established.

Micro Controller

The implementation of the software design for the microcontroller was one of the biggest tasks during the development of the Electronic Lead screw System. It was done in multiple steps.
First, the functional part of the software was created using Matlab and Matlab Simulink.

Leitspindel Drehzahlen							
Steigung	Einheit	Übersetzung	100	250	350	500	1700
0,4	mm	0,266666667	26,6666667	66,6666667	93,3333333	133,333333	453,333333
0,5	mm	0,333333333	33,3333333	83,3333333	116,666667	166,666667	566,666667
0,7	mm	0,466666667	46,6666667	116,666667	163,333333	233,333333	793,333333
0,8	mm	0,533333333	53,3333333	133,333333	186,666667	266,666667	906,666667
1	mm	0,666666667	66,6666667	166,666667	233,333333	333,333333	1133,33333
1,25	mm	0,833333333	83,3333333	208,333333	291,666667	416,666667	1416,66667
1,5	mm	1	100	250	350	500	1700
1,75	mm	1,166666667	116,666667	291,666667	408,333333	583,333333	1983,33333
2	mm	1,333333333	133,333333	333,333333	466,666667	666,666667	2266,66667
2,5	mm	1,666666667	166,666667	416,666667	583,333333	833,333333	2833,33333
3	mm	2	200	500	700	1000	3400
10	TPI	1,692307692	169,230769	423,076923	592,307692	846,153846	2876,92308
11	TPI	1,538461538	153,846154	384,615385	538,461538	769,230769	2615,38462
13	TPI	1,3	130	325	455	650	2210
19	TPI	0,888888889	88,8888889	222,222222	311,111111	444,444444	1511,11111
20	TPI	0,846153846	84,6153846	211,538462	296,153846	423,076923	1438,46154
22	TPI	0,769230769	76,9230769	192,307692	269,230769	384,615385	1307,69231
40	TPI	0,423076923	42,3076923	105,769231	148,076923	211,538462	719,230769
44	TPI	0,384615385	38,4615385	96,1538462	134,615385	192,307692	653,846154
0,09	mm/REV	0,05859375	5,859375	14,6484375	20,5078125	29,296875	99,609375
0,018	mm/REV	0,1171875	11,71875	29,296875	41,015625	58,59375	199,21875

Figure 4.10.: Chart of the Lead screw dynamics; green - reachable Lead screw RPM with a 25/80 transmission, red - not reachable Lead screw RPM with a 25/80 transmission

In the functional software, a series of logical actions were needed in order to compute the required motor signal based on the encoder position. The task of the first software part was it to process the position data coming from the encoder and calculate the desired motor position. This process is shown in form of a block diagram in figure 4.11.

Because the encoder is constantly incrementing the value of a 16bit register, it is possible that the value reaches the maximum or minimum possible value. In this case, the register value will "overflow" and jump from its maximum to its minimum or from its minimum to its maximum value. This would result in a very big but wrong movement of the motor. The first step of the functional software is to detect such an overflow and correct for it. This way, the calculation of the desired steps is always performed with the absolute movement of the encoder. Next, the direction of rotation of the spindle needs to be detected and the direction output to the motor need to be set accordingly. In the last step of the functional software, the desired steps are calculated. This is done by multiplying the steps of the encoder by a previously determined translation factor.
The second part of the functional software generates the logical pulses in order to move the motor. Therefore, it checks the register value of the desired steps on a cyclic basis. If this value is not zero, the software will create a one micro second logical pulse and decrement the register value afterwards by one. These steps are also shown in figure 4.11.
During the second part of the software implementation, the functional code was translated to C Code using Matlabs Code generation toolbox and deployed to the hardware. To be able to use the functional code on the microcontroller, an implementation routine needed to be created. This routine was used in order to read all inputs, such as the

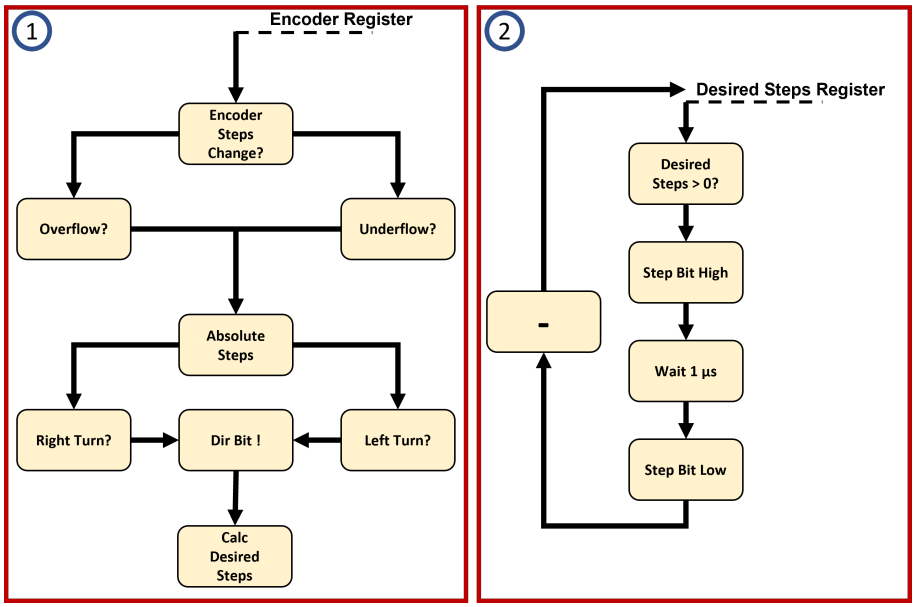


Figure 4.11.: Block diagram of the first part of the functional software; 1 - Functional Software first part, 2 - Functional Software second part

encoder position, and the communication interface relay this information to the functional code. Further, the implementation routine enabled the functional code to use the GPIO module of the microcontroller and send out the current RPM of the spindle via the communication interface.

HMI

After the creation of the graphical interface during the design phase, the HMI needed to be connected to the microcontroller. For this purpose, a UART communication interface was used. To transmit Mode and translation factor and receive the spindle RPM simultaneously, a custom communication protocol was created. This protocol consist of 5 bytes of data. The first and last byte are so-called start and stop bytes, used in order to determine between different data packages. They have the value 0xFF. The second byte is an integer with a value between 0 and 2 in order to communicate normal, metric or imperial mode to the microcontroller. The third and fourth bytes represent the determined translation factor. Because it is not possible to transmit fractional numbers via a UART communication interface, the integer part of the translation factor is transmitted in the third byte. The fractional value of the translation factor is then converted to an integer and transmitted in the fourth byte. This process has then to be reversed on the microcontroller.

4.2.4. Hardware Implementation

For the implementation of the Hardware, the main task was to solve mechanical problems. This was done with the extensive use of a CAD Software, a 3-D Printer as well as the Lathe.

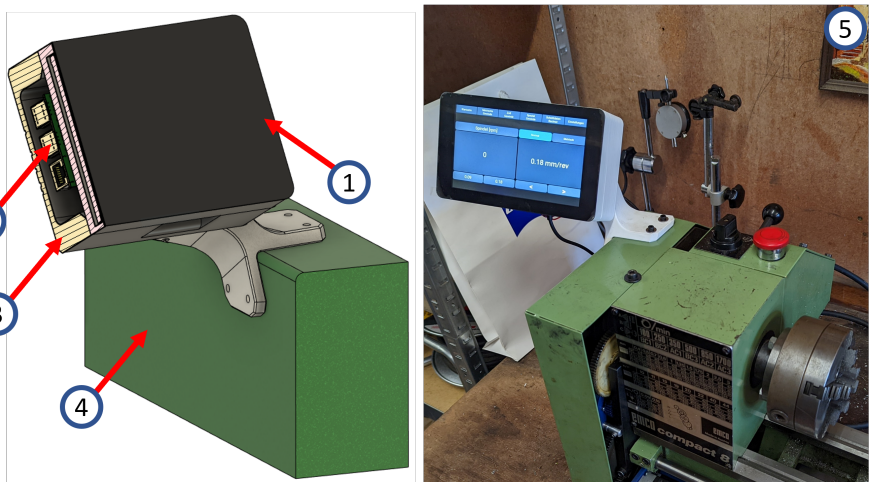


Figure 4.12.: Cross-section of the Touchscreen mount; 1 - Touchscreen, 2 - Raspberry Pi, 3 - Enclosure and mount, 4 - Lathe model, 5 - Picture of the finished implementation

Touchscreen

In order to mount the touchscreen as well as the Raspberry Pi to the lathe, a housing needed to be created. This was done by first modeling each of the components as well as a section of the lathe in the CAD Software Autodesk Fusion 360. After this, a house was created the positioned the touchscreen in a way that made it convenient to read and operate while working on the lathe. Further, the housing featured an enclosure for the Raspberry Pi that was mounted to the back of the touchscreen.
After the 3-D Model of the touchscreen mount was complete, it was 3-D printed in PETG plastic. Finally, the touchscreen could be mounted to the lathe. The 3-D model and the final result are shown in figure 4.12.

Encoder

For the implementation of the encoder to, problems needed to be solved. First, a method needed to be developed in order to mount the encoder to the lathe. Second, the RPM of the main Spindle needed to be transferred to the shaft of the rotary encoder. In order to mount the encoder to the lathe, all components were modeled in Fusion 360. As a good mounting point on the lathe, the old gear holder stood out. This way, the encoder position could be changed easily. In the second step, the mount for the encoder was modeled as well as a gear in order to transfer the spindle RPM to the encoder with a 1:1 transmission factor. Both the holder and the gear were 3-D printed. Only the standoffs for the mount were manufactured from Aluminum on the lathe. This was done in order to insure a rigid mounting of the encoder. Figure 4.13 shows the complete mounting of the encoder.

Motor

Similar to the mount of the encoder and the touchscreen, the motor mount was designed with the use of Fusion 360. Because of the very limited space on the machine, this was

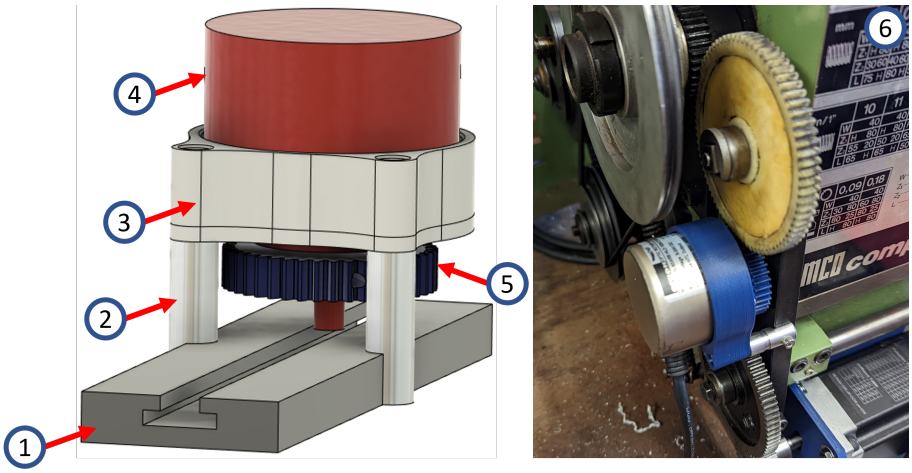


Figure 4.13.: Image of the encoder mount; 1 - Old gear holder, 2 - Machined standoffs, 3 - 3-D printed Encoder mount, 4 - Encoder model, 5 - 3-D printed gear, 6 - Picture of the finished implementation

very important. First, the parts of the Lathe that could possibly interfere with the motor or its mount were modeled. Then, the motor and the motor could be designed. The motor mount was designed, such that the play of the gears can be adjusted by sliding the motor in its mount up and down. This is shown in Figure 4.14. After the model of the motor mount was finished, it was 3-D printed in order to test the fit of the mount. Initially, the motor mount was supposed to be manufactured from steel afterward but during testing the 3-D printed motor mount proved to be rigid enough. In order to connect the motor to the microcontroller a logic level shifter was needed. This was caused, because the motor controller was operating on a 5V logic level, while the microcontroller was operating on a 3.3V logic level.

4.3. Testing

During the testing phase of the development process, every part of the system is tested to see if it is functioning as defined in the requirements and the System design. This is done on component level, system level and finally on the integrated system. As shown in Figure 4.15, this is the second part of the V Model development cycle.

4.3.1. Component Test

First, the correct function of each of the components needs to be ensured. This can be done on the finished system, but is often easier to do before the implementation. Because the motor, motor controller and encoder are bought from a third party, they are not included in the component testing.

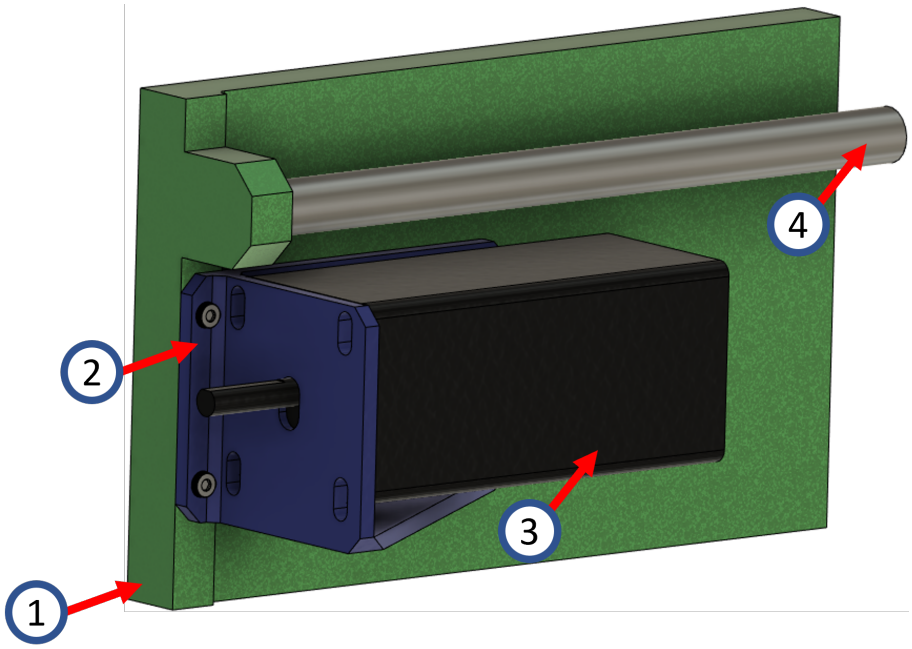


Figure 4.14.: Image of the motor mount; 1 - Old gear holder, 2 - Machined standoffs, 3 - 3-D printed Encoder mount, 4 - Encoder model, 5 - 3-D printed gear, 6 - Picture of the finished implementation

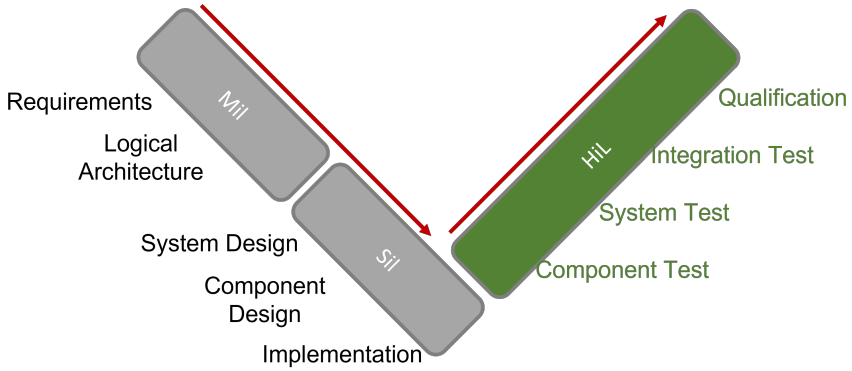


Figure 4.15.: V Model Component Test Stage

HMI

In order to test the functionality of the HMI and especially the communication, a special test program was created. This program allowed to monitor the data packages send by the HMI. Further, it was possible to send RPM data to the HMI in order to test the RPM display.

Micro Controller

During the component testing phase, only the implementation softer on the microcontroller was tested. This step was tremendously simplified by the onboard debugging probe of the TI LAUNCHXL development board. This way, it was able to track every register value on the microcontroller to ensure a correct working behavior. The correct timing of both real time loops were checked by toggling one of the GPIO Pins of the microcontroller and reading its value with the help of an Oscilloscope.

4.3.2. System Test

The system testing was done in two stages, the first testing series was done within Matlab Simulink. Secondly, the on a test bench assembled system was tested. Later, the result of both test could be compared.

Matlab Simulink

In order to test the system in Matlab Simulink, every component needed to be modeled. The most complex part to model in the Simulink environment was the encoder and the drive train for the Lead screw.
The model of the encoder needed to be able to simulate events like a register overflow, different trajectories as well as constant speeds. The model of the encoder is shown in Figure 4.16.
Every test function of the encoder was then routed over a case-sensitive switch. This switch can be controlled by a variable and connects based on this the according input to the output. This makes switching the test scenario easy and fast.

For the simulation of the drive train, both the electronic and the original gear drive train were modeled. The motor and motor controller were modeled with a library of Simulink called "Simscape". Simscape is designed in order to model complex electromechanical with minimum effort. The model of the electronic drivetrain is shown in Figure 4.17.
It features four sections, first the step and direction signal from Simulink are converted to signal that are compatible with the Simscape system. Second, the motor controller is modeled. The third block represents the BLDC motor. However, in this case, a Stepper motor block was used in order to model the stepwise control of the motor. In the last block of the model, the output signal of the motor (angle Θ) is converted back to a numeric Simulink signal.

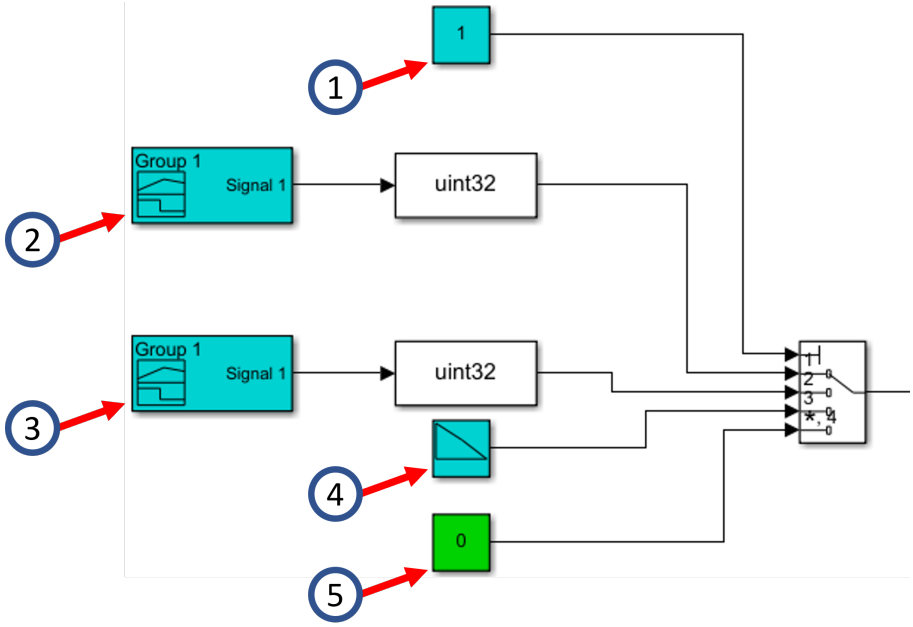


Figure 4.16.: Simulink Model of the encoder; 1 - Constant position, 2 - Trajectory 1, 3 - Trajectory 2, 4 - Register overflow, 5 - Case selector

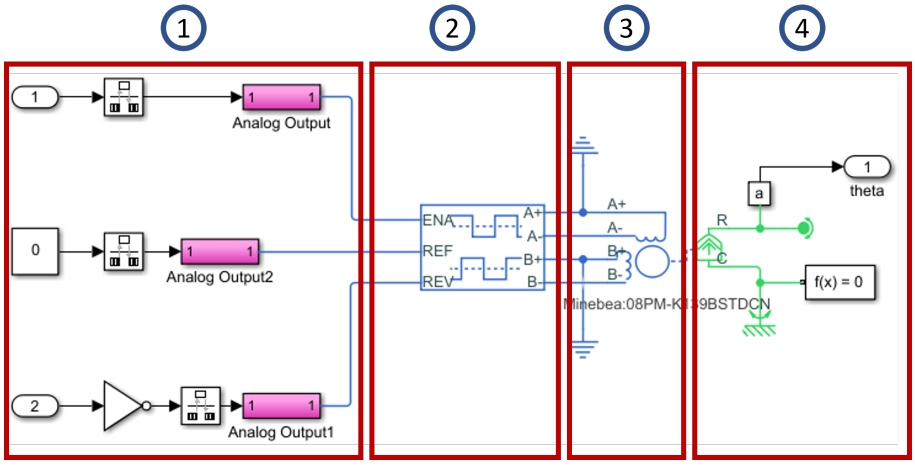


Figure 4.17.: Simulink Model of the electronic drive train; 1 - Input processing, 2 - Motor Controller, 3 - Motor, 4 - Output processing

Test Bench

In order to test both hardware and software in a realistic environment, the system was assembled in a "Test Bench" configuration. This means, that the inputs into the system can be tightly controlled, and every output can be observed. For this reason, the encoder was not connected to the system during this test. It was replaced by another microcontroller, which simulated the signal of the encoder. This way, the amount of encoder steps could be precisely determined. The outputs of the system were connected to both, the motor and an Oscilloscope. This way both, the motor movement and the logical signal of the system could be observed.

Because the basic functionality of the system was already tested in previous steps, the test performed on the test bench were focused on system performance. System performance was mainly determined by the floating point division that was needed in order to calculate the desired amount of steps. To optimize the processing time of this calculation, two different methods were tested:

For the first method that was tested, the translation factor was calculated beforehand. This way, the predefined translation factor has just to be multiplied with the encoder steps. However, this has the big disadvantage, that it will introduce a cumulating rounding error into the calculation. If this method is later used in the final software version, this rounding error has to be investigated.

For the second method, the translation factor was not calculated beforehand. Everything the encoder position changes, the complete translation calculation is done. The equation to calculate the desired Steps is given by:

$$EncoderSteps * \frac{MotorSteps * MotorTransmission * EncoderTransmission * FeedSetting}{EncoderResolution * LeadscrewSlope} \tag{4.1}$$

With this calculation method, the rounding error of the calculation will not cumulate because the calculation is done every time a new Desired Step value is calculated.

4.3.3. Integration Test

The last step in the testing phase is the integration test. This test determines if the behavior of the fully integrated system is matching the specifications. In case of the Electronic Lead screw, tow different integration tests were performed. First regular turning was tested, second metric and imperial thread cutting was investigated.

Turning

The testing of normal turning operation was performed with brass, aluminum and stainless steel. First, the maximum possible feed for each of the metal was investigated.

This was done with a 1 mm depth of cut. In the second testing phase, the best possible surface quality was investigated. This was done with a depth of cut of 0.05 mm. The feed of the lathe was then adjusted in order to create the best possible surface finish.

Threading

The testing of metric and imperial thread cutting was performed with aluminum and a hand ground HSS threading tool. The main focus during this testing series was the consistency of the thread pitch. For the imperial test, a 3/8" thread with 19 threads per inch was chosen. For the metric test, a standard M8x1.25 thread was selected. Both of these threads are very common and could later be checked with thread gauges.

5. Results, Discussion and Conclusion

In this chapter, the results of the testing phase will be shown. Further, the design decisions that were made based on these results will be discussed.

5.1. Arithmetic

During the system testing phase, extensive testing was done on the performance of different arithmetic algorithms. The performance of each algorithm was measured by toggling one of the GPIO pins to a HIGH state at the beginning of the calculation and resetting it back to LOW as soon as the calculation was done. This way, the processing time could be measured with an Oscilloscope. It is important to note that the time limit for the real time loop was set to 100μs. All steps prior to the floating point calculation were measured to use 40μs of time.

5.1.1. Floating point arithmetic

First, the floating point algorithm was tested. This algorithm introduces a cumulating rounding error into the calculation, but had the advantage that the calculation done within the real-time loop is as small as possible. The time measurement with the Oscilloscope showed a calculation time of 40.2μs. This leaves 59.8μs time for other steps. When considering the previously measured time of 40μs for the steps needed prior to the floating point calculation, this leaves a time buffer of 19.8μs. The measurement result is shown in Figure 5.1.

5.1.2. Fractional arithmetic

Second, an algorithm was tested using fractional arithmetic. This algorithm had the big advantage that it did not introduce a cumulating rounding error into the calculation of the desired steps. The measurement of this calculation, as shown in Figure 5.1, resulted in a calculation time of 55μs. With the additional 40μs, while still being within the specification, this algorithm only leaves 5μs of buffer to the real time criteria.

5.1.3. Discussion

Based on the previously done measurements, both algorithms will be calculated within the time limit of 100μs. However, the fractional arithmetic only leaves 5μs of buffer, bevor breaking the real time limitation. This would render the solution of the algorithm invalid and one complete calculation cycle would be missed by the system. For this

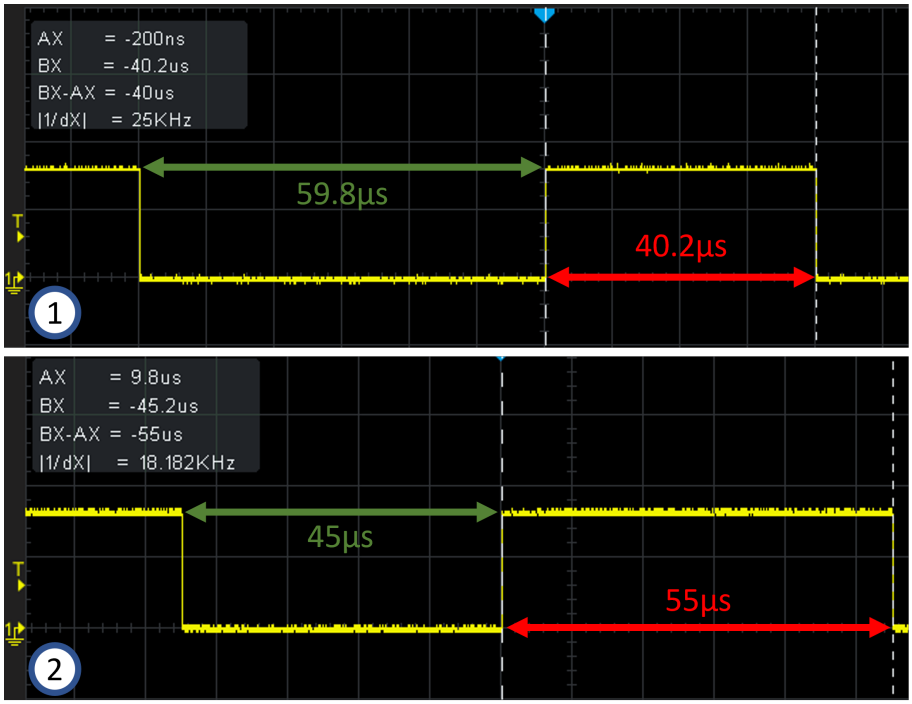


Figure 5.1.: Results of the algorithm testing; 1 - Floating point algorithm, 2 - Fractional arithmetic, red - Time needed for the calculation, green - Time needed for the calculation, blue - Time available for other calculation steps

Test. Nr.	Depth of cut	Feed	Success
1	1 mm	0.09 mm/rev	good
2	1 mm	0.11 mm/rev	good
3	1 mm	0.13 mm/rev	good
4	1 mm	0.15 mm/rev	good
5	1 mm	0.17 mm/rev	good
6	1 mm	0.19 mm/rev	canceled

Table 5.1.: Results material removal performance

reason, a real time violation has to be avoided. Based on this, the floating point algorithm was chosen to be implemented into the system. However, further tests have to show if the accumulation of the rounding error of this algorithm has a significant impact on the performance of the Electronic Lead screw.

5.2. Turning

The test of the normal turning performance of the system was part of the integration test. It was aimed to test both material removal performance and surface quality. As shown in Table 5.1 the system performed good up to a feed rate of 0.17 mm/rev. At 0.19 mm/rev the test was canceled because the RPM of the main spindle decreased by ~200rpm.

As shown in table 5.2 the test for surface quality was also performed with a fixed depth

Test. Nr.	Depth of cut	Feed	Success
1	0.05 mm	0.09 mm/rev	ok
2	0.05 mm	0.08 mm/rev	ok
3	0.05 mm	0.07 mm/rev	ok
4	0.05 mm	0.06 mm/rev	ok
5	0.05 mm	0.05 mm/rev	good
6	0.05 mm	0.04 mm/rev	blunt

Table 5.2.: Results of the surface quality testing

of cut and a variable feed rate. With decreasing feed rate, the surface quality was improving consistently, up to a point of 0.05 mm/rev. As soon as the feed was lowered below this point, the surface of the part became blunt. However, it is important to note that the surface quality was determined by eye. This was done because no tool was available to measure the surface quality after each cut.

5.3. Threading

The testing of the threading performance of the system was done on two example threads. These threads were chosen, because tools were available to measure the thread precision objectively. Further, it was essential to test both the metric mode of the machine and the transformation into the imperial unit system. In figure 5.2 the final result of both threading tests and the previous surface quality test is shown. Both, the imperial thread and the metric thread, were within the specifications of corresponding thread gauges. This proofed the capability of the machine to transform its motion calculation into another unit system. Further, it was shown that the impact of the cumulating rounding error introduced by the floating point algorithm on normal threading operations is insignificant.

5.4. Conclusion

During the testing, the Electronic Lead screw system has proven to meet the initial defined requirements. Further, this system shows to be a very valuable addition to this lathe. It enables the operator to use a wider range of feeds while reducing the setup time of the machine significant. This increases the capability of a small lathe like the Emco Compact 8 to a point where it is a good addition for every home metal workshop. While reflecting the development process of the Electronic Lead screw and comparing the needed time for each of the work items, to the planned time, two things stand out. The first was the time needed for the selection and sourcing of the components. During the last year, a big shortage in electronic components has occurred. This delayed the delivery of some components by up to 6 months. Further, the complexity of the selection of the components was underestimated. The main problem here was the physical size of

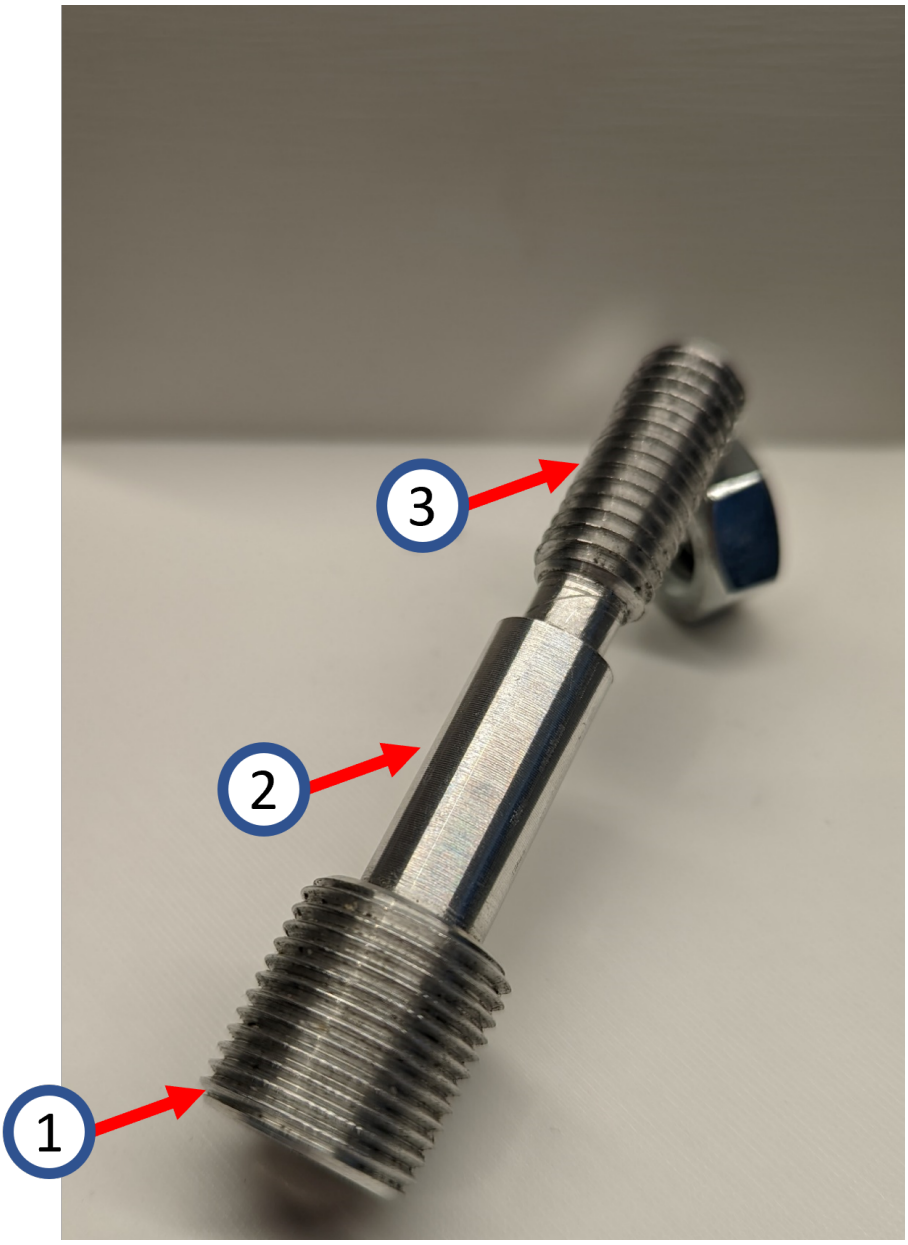


Figure 5.2.: Results of the threading and surface quality test; 1 - 3/8" 19G thread, 2 - best surface quality, 3 - M8x1.25 thread

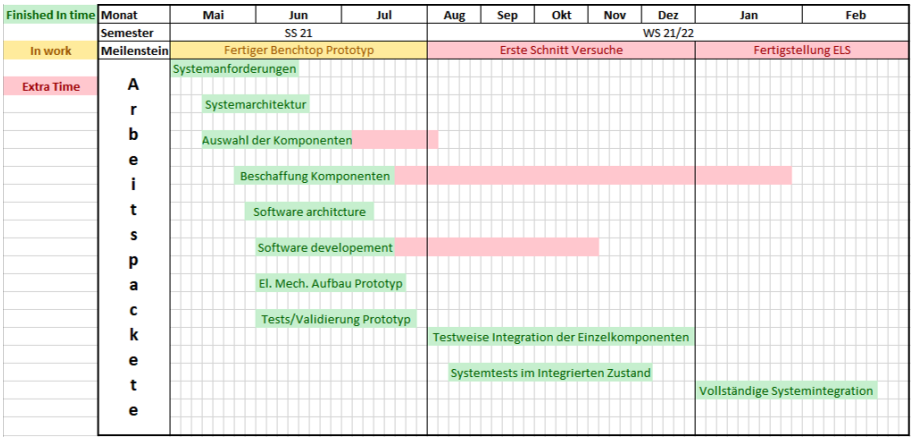


Figure 5.3.: Chart of the planned time vs the need time per task

each of the components and how they should be integrated into the system.
The second thing that stands out from the timetable shown in Figure 5.3 is the time required for the Software development. The task that had been underestimated here was the software development for the Texas Instruments microcontroller. It took a lot of time to get familiar with its complex working principle.

6. Outlook

The Electronic Lead screw system offers due to the modularity of the control software and the interface the opportunity to add further functions to the system. In the future, modes for circular grinding as well as partially automated threading are planned. However, more important additions to the Electronic Lead screw system will be a cabinet for the electronics. This cabinet and its components were already planned and ordered during the project, but did not arrive in time due to a global electronics shortage.
Further, a Variable Speed Drive (VSD) will be added to the main spindle in the future. A VSD enables the operator to control the speed of the main spindle step-less. In addition to that, the VSD could be controlled connected to the ELS. This would enable features such as automatic threading or turning.

7. List of Figures

8. List of Tables

9. References

[1] Hochschule Aalen. *Ueber uns*. https://www.hs-aalen.de/pages/ueber-uns.

[2] James Clough. *Electronic Leadscrew Controller*. https://github.com/clough42/electronic-leadscrew.

[3] Alfio Quarteroni. *Numerical Mathematics*. New York, NY: Springer, 2007. ISBN: 9781475773941.

[4] Fraunhofer-Institut IGCV. *Additive Manufacturing*. https://www.igcv.fraunhofer.de/de/forschung/kompetenzen/additive_fertigung_am.html.

[5] Celera Motion. *Was ist ein optischer Encoder?* https://www.celeramotion.com/microe/de/optischer-encoder/.

[6] Faulhaber Group. *Schrittmotoren für anspruchsvolle Positionieraufgaben*. https://www.faulhaber.com/de/produkte/schrittmotoren/. Jan. 2022.

[7] Suk-Hwan Suh. *Theory and Design of CNC Systems*. Springer-Verlag GmbH, Aug. 2008. 456 pp. ISBN: 978-1-84800-336-1. URL: https://www.ebook.de/de/product/12469020/dae_hyuk_chung_seong_kyoon_kang_ian_stroud_suk_hwan_suh_theory_and_design_of_cnc_systems.html.

[8] Cem Ünsalan, Hüseyin Deniz Gürhan, and Mehmet Erkin Yücel. *Embedded System Design with ARM Cortex-M Microcontrollers*. Springer International Publishing, Jan. 2022. 569 pp. ISBN: 978-3-030-88439-0. URL: https://www.ebook.de/de/product/42038562/cem_uensalan_hueseyin_deniz_guerhan_mehmet_erkin_yuecel_embedded_system_design_with_arm_cortex_m_microcontrollers.html.

[9] Texas Instruments. *LAUNCHXL-F280049C*. https://www.ti.com/tool/LAUNCHXL-F280049C.

[10] Gregor berle. *Tabellenbuch Elektrotechnik Tabellen, Formeln, Normenanwendungen*. Haan-Gruiten: Verlag Europa-Lehrmittel Nourney, Vollmer GmbH & Co. KG, 2018. ISBN: 9783808534908.

[11] Raspberry Pi Foundation. *What is a Raspberry Pi?* https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[12] Mathworks. *Matlab - Overview*. https://de.mathworks.com/products/matlab.html. Accessed: 07.08.2021.

[13] ISO and IEC. *ISO IEC JTC1 SC22 WG14 N1169 Programming languages - C*. Standart. ISO, Jan. 2022.

[14] Python Software Foundation. *What is Python*. https://docs.python.org/3/faq/general.html#what-is-python. Jan. 2022.

[15] Kivy.org. https://kivy.org/doc/stable/gettingstarted/intro.html. Feb. 2022.

[16] Texas Instruments. *Code Composer Studio integrated development environment (IDE)*. https://www.ti.com/tool/CCSTUDIO. Jan. 2022.

Appendix

A. List of Companies

Company: EMCO GmbH
Website: `https://www.emco-world.com/`

Company: Kivy.org
Website: `https://kivy.org/`

Company: The MathWorks, Inc.
Website: `https://www.mathworks.com/`

Company: OPKON Optic Electronic A.Ş.
Website: `https://www.opkon.com.tr/`

Company: Raspberry Pi Foundation

Website: `https://www.raspberrypi.org/`

Company: STEPPERONLINE, Inc.

Website: `https://www.omc-stepperonline.com/`

Company: Texas Instruments, Inc.

Website: `https://www.ti.com/`

B. Requirements ELS

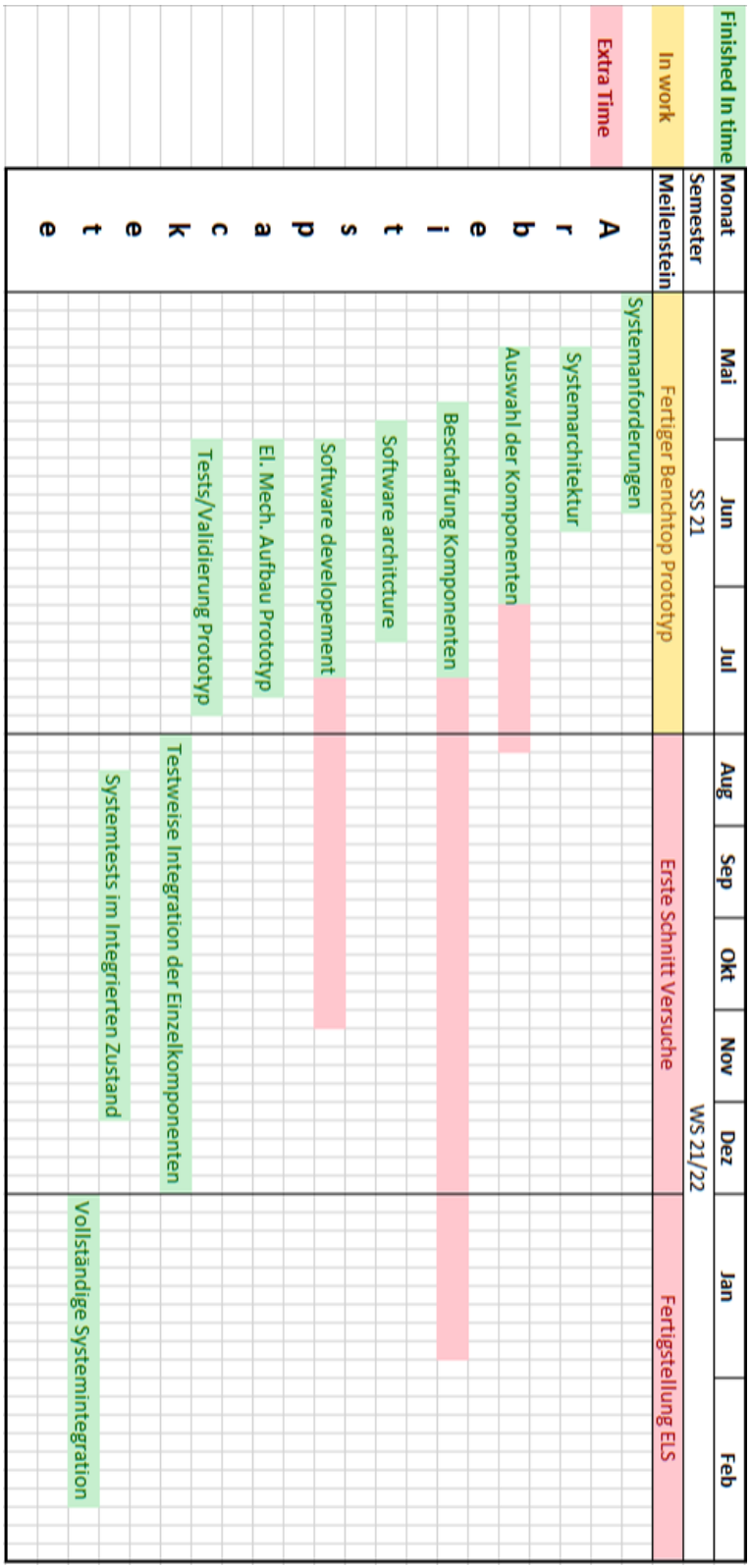
Req.Nr/Komponente	Requirement	Design
uC		
1	Takt	100MHz
2	IO-Ports	> 10
3	Flash	> 200kB
4	RAM	> 200kB
5	Harware Com.	Encoder
6	ADC	> 1
7	Timer	> 2
8	Dokumentation	Ja
9	Kosten	< 100€
Motor		
1	Drehzahl	> 3000rpm
2	Drehmoment	< 0.3Nm
3	Motor Kosten	< 150€
4	Min drehzahl	< 1rpm
5	Motor Größe	
Motor Controller		
1	Closed Loop	Ja
2	Versorgungs Spannung	> 24V < 50V
3	Programmierbar	Ja
4	Ansteuerung	logisch
5	Auflösung	> 1000 ppt
6	Kosten	> 100€
7	Montage	DIN Rail

Figure B.1.: System requirements table 1

Encoder		
1	Pulses per Turn	> 2000 ppt
2	Max RPM	> 2000 rpm
3	Größe	< 60mm
4	Richtungs erkennung	Ja
5	Montage	Schrauben
6	Kosten	< 100€
7	Spannungsversorgung	5V
HMI		
1	Touch	Ja
2	Modular	Raspberry Pi
3	UART	Ja
4	Eigene Spannungsversorgung	5V
5	Anbringung	3-D Druck
6	Kosten	< 150€

Figure B.2.: System requirements table 2

C. Time Plan


Finished In time
In work
Extra Time
Monat
Semester
Meilenstein
Mai
Jun
Jul
Aug
Sep
Okt
Nov
Dez
Jan
Feb
SS 21
WS 21/22
Arbeitspaket
Systemanforderungen
Systemarchitektur
Auswahl der Komponenten
Beschaffung Komponenten
Software architcture
Software developement
El. Mech. Aufbau Prototyp
Tests/Validierung Prototyp
Fertiger Benchtop Prototyp
Erste Schnitt Versuche
Fertigstellung ELS
Testweise Integration der Einzelkomponenten
Systemtests im Integrierten Zustand
Vollständige Systemintegration

D. Source Code

D.1. main.c

```
1  //
2  // Included Files
3  //
4  #include "F28x_Project.h"
5  #include "Configuration.h"
6
7  #include "..\Matlab\RealTimeMachine_ert_rtw\RealTimeMachine.h"
8  #include "..\Matlab\RealTimeMachine_ert_rtw\rtwtypes.h"
9  #include "..\Matlab\RealTimeMachine_ert_rtw\
       zero_crossing_types.h"
10 #include "..\Matlab\RealTimeMachine_ert_rtw\
       RealTimeMachine_data.c"
11
12 #include "..\Matlab\StepperRTM_ert_rtw\StepperRTM.h"
13 #include "..\Matlab\StepperRTM_ert_rtw\rtwtypes.h"
14 #include "..\Matlab\StepperRTM_ert_rtw\zero_crossing_types.h"
15
16 #define _FLASH
17
18 //
19 // Global Variables Inputs
20 //
21 static uint32_T arg_SpindelPos = 0U;
22 volatile  real32_T arg_CountFactor = 0;
23 volatile uint16_T var_StepBacklog;
24
25 //
26 // Global Variables Outputs
27 //
28 static uint16_T arg_StepBit;
29 static uint16_T arg_Dir;
30 static uint16_T arg_DesSteps;
31
32 //
33 // Global Variables Statemachine Clock
34 //
35 static uint16_T System_Trigger[2] = { 0U, 0U };
36 static boolean_T System_Takt = 0;
37 static uint16_T Stepper_Trigger[2] = { 0U, 0U };
```

```
38 static boolean_T Stepper_Takt = 0;
39
40 //
41 // Global Variables Helpers
42 //
43 volatile uint32_T current = 0;
44 volatile uint32_T count = 0;
45 volatile uint32_T previous = 0;
46 volatile uint16_T RPM;
47
48 volatile int msg[] = {0, 0, 0, 0, 0};
49 volatile int i = 0;
50 volatile float feed = 0.0;
51 volatile int Mode = 1;
52 volatile int TransferComplete = 0;
53
54 void main(void)
55 {
56
57     #ifdef _FLASH
58         // Copy time critical code and Flash setup code to RAM
59         // The RamfuncsLoadStart, RamfuncsLoadEnd, and
               RamfuncsRunStart
60         // symbols are created by the linker. Refer to the
               linker files.
61         memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)
               &RamfuncsLoadSize);
62
63         // Initialize the flash instruction fetch pipeline
64         // This configures the MCU to pre-fetch instructions
               from flash.
65         InitFlash();
66     #endif
67
68
69     //
70     // Initialize Autocode
71     //
72     StepperRTM_initialize();
73     RealTimeMachine_initialize();
74     //
```

```
75      // Initialize device clock and peripherals
76      //
77      InitSysCtrl();
78
79      //
80      // Initialize GPIO, Timer and EQEP
81      //
82      InitGpio();
83      setupGPIO();
84      setupTimer();
85      setupEQEP();
86
87
88
89
90      //
91      // Initialize UART
92      //
93      initSCIAFIFO();
94      initSCIAEchoback();
95
96
97
98
99
100     while(1)
101     {
102         //
103         // Send RPM via UART
104         //
105         if(EQep1Regs.QFLG.bit.UTO==1)
106         {
107
108             Uint32 current = EQep1Regs.QPOSLAT;
109             Uint32 count = (current > previous) ? current -
                    previous : previous - current;
110
111             // deal with over/underflow
112             if( count > _ENCODER_MAX_COUNT/2 )
113              {
114                  count = _ENCODER_MAX_COUNT - count; // just
```

```
                subtract from max value
115             }
116
117             RPM = count * 60 * RPMSampleTime / EncoderRes;
118
119             int highbyte = RPM >> 8;
120             int lowbyte = RPM & 0x00ff;
121
122             previous = current;
123             transmitSCIAChar(lowbyte);            // Send RPM
                    out via UART
124             transmitSCIAChar(highbyte);           // Send RPM
                    out via UART
125             EQep1Regs.QCLR.bit.UTO=1;         // Clear interrupt
                     flag
126         }
127
128         //
129         // Check for new Massages from the Raspberry Pi
130         //
131          if(SciaRegs.SCIFFRX.bit.RXFFST != 0)
132         {
133            msg[i] = SciaRegs.SCIRXBUF.bit.SAR;
134
135            if(i == 4)
136            {
137               i = 0;
138               TransferComplete = 1;
139               Mode = msg[1];
140               feed = msg[2] + (msg[3] * 0.01);
141            }
142            else
143            {
144                i++;
145            }
146         }
147
148         //
149         // Calculate Step-factor based on information received
                via UART
150         //
```

```
151
152         if(TransferComplete && (msg[0] == 0xff) && (msg[4] ==
                0xff))
153         {
154             TransferComplete = 0;
155              // Normal Feed and metric thread cutting
156             if(Mode == 1 || Mode == 2)
157             {
158                 arg_CountFactor = ((Steps * MotorTransmission *
                        EncoderTransmission * feed)/(EncoderRes *
                        LeadscrewSlope));
159             }
160
161              // Imperial thread cutting
162             else if(Mode == 3)
163             {
164                 arg_CountFactor = ((Steps * MotorTransmission *
                        EncoderTransmission * OneInch)/(EncoderRes
                        * LeadscrewSlope * feed ));
165             }
166         }
167     }
168 }
169
170 //
171 // cpuTimer0ISR - CPU Timer0 ISR
172 //
173 __interrupt void cpuTimer0ISR(void)
174 {
175
176     Stepper_Takt = !Stepper_Takt; // Toggle System Clock
177     Stepper_Trigger[0] = (uint16_T)Stepper_Takt;
178
179     if(Stepper_Takt == 0)
180     {
181         Stepper_Trigger[1] = 1; //Power on Reset
182     }
183
184     StepperRTM_step(var_StepBacklog, (uint16_t*)&
            Stepper_Trigger, &arg_StepBit);
185
```

```
186     //
187     // Stepper Clock for Debugging
188     //
189
190     GpioDataRegs.GPASET.bit.GPIO6 = arg_StepBit;
191     GpioDataRegs.GPADAT.bit.GPIO6 = arg_StepBit;
192
193     //
194     // Acknowledge this interrupt to receive more
195     //
196     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
197 }
198
199
200
201 //
202 // cpuTimer0ISR - CPU Timer2 ISR
203 //
204 __interrupt void cpuTimer2ISR(void)
205 {
206     System_Takt = !System_Takt; // Toggle System Clock
207     System_Trigger[0] = (uint16_T)System_Takt;
208
209     if(System_Takt == 0)
210     {
211         System_Trigger[1] = 1; //Power on Reset
212     }
213
214     arg_SpindelPos = EQep1Regs.QPOSCNT;
215
216     RealTimeMachine_step(arg_SpindelPos, arg_CountFactor, (
        uint16_t*)&System_Trigger,
217                          &arg_DesSteps, &arg_Dir);
218
219     var_StepBacklog = var_StepBacklog + arg_DesSteps;
220
221     GpioDataRegs.GPBSET.bit.GPIO39 = !arg_Dir;
222     GpioDataRegs.GPBDAT.bit.GPIO39 = !arg_Dir;
223
224     //
225     // Acknowledge this interrupt to receive more
```

```
226     //
227     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
228 }
```

D.2. MainUI.py

```
#!/usr/bin/python

## Libraries Import
from logging import Manager
from time import sleep, time
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.core.window import Window
from kivy.clock import Clock
import serial
import RPi.GPIO as GPIO

class CommunicationClass(object):

        def __init__(self):
                self.Mode = int(1)
                self.Feed = 0.09
                self.serialIndicator = 0
                self.RPM = 0
                self.Metric_BTN = 0
                self.Imperial_BTN = 0
                self.FeedFeed = 0.09

        def SetBTN(self, Screen, BTN, State):

                if Screen == 1:
                        self.Metric_BTN = BTN

                elif Screen == 2:
                        self.Imperial_BTN = BTN

                if State:
                        kv.screens[Screen].ids[str(BTN)].
                            enabled = 1

                else:
                        for n in range(0,4):
```

```
39                                  for m in range(0,14):
40                                          try:
41                                                  kv.screens[n].
                                                        ids[str(m)
                                                        ].enabled =
                                                         0
42                                          except:
43                                                  pass
44
45          def getStatus(self):
46                  return self.Mode, self.Feed, self.
                        serialIndicator, self.Metric_BTN, self.
                        Imperial_BTN, self.FeedFeed
47
48          def initCom(self):
49
50                  if self.serialIndicator == 0:
51                          try:
52                                  self.ser = serial.Serial('/dev
                                        /ttyACM0', 9600, timeout =
                                        0.2)
53                                  sleep(1)
54                                  self.serialIndicator = 1
55                                  self.Mode = 'Normal'
56
57                          except:
58                                  self.serialIndicator = 0
59                          pass
60
61          def TX(self, Mode, Feed):
62                  if Mode == 1:
63                          self.FeedFeed = Feed
64
65                  self.Mode = int(Mode)
66                  self.Feed = round(Feed, 2)
67                  self.FeedInt = int(Feed)
68                  self.FeedDez = int((Feed - int(Feed))*100)
69
70                  if self.serialIndicator:
71                          self.ser.write(b'\xff')
72                          self.ser.write(self.Mode.to_bytes(1,
```

```
                        byteorder='big'))
73                      self.ser.write(self.FeedInt.to_bytes
                        (1, byteorder='big'))
74                      self.ser.write(self.FeedDez.to_bytes
                        (1, byteorder='big'))
75                      self.ser.write(b'\xff')
76                      print('Transmission completed')
77
78      def RX(self):
79
80              lowbyte = 0
81              highbyte = 0
82              TransferComplete = 0
83
84              if self.serialIndicator:
85                      while not TransferComplete:
86                              if self.ser.in_waiting == 2:
87                                      lowbyte = int.
                                        from_bytes(self.ser
                                        .read(1), 'big',
                                        signed=False)
88                                      highbyte = int.
                                        from_bytes(self.ser
                                        .read(1), 'big',
                                        signed=False)
89                                      highbyte = highbyte <<
                                         8
90                                      self.RPM = lowbyte +
                                        highbyte
91                                      self.ser.flushInput()
92                                      TransferComplete = 1
93                                      return str(self.RPM)
94
95
96                              elif self.ser.in_waiting > 2:
97                                      self.ser.flushInput()
98
99                              else:
100                                     return str(self.RPM)
101             else:
102                     return 'Not connected'
```

```
103
104 class Startseite(Screen):
105         def btn_defone(self):
106                 MainApp.MainCom.SetBTN(0,0,0)
107                 MainApp.MainCom.TX(1, 0.09)
108
109         def btn_deftwo(self):
110                 MainApp.MainCom.SetBTN(0,0,0)
111                 MainApp.MainCom.TX(1, 0.18)
112
113         def btn_normal(self):
114                 feed = MainApp.MainCom.getStatus()[5]
115                 MainApp.MainCom.TX(1, feed)
116                 MainApp.MainCom.SetBTN(0,0,0)
117
118         def btn_gewinde(self):
119                 if MainApp.MainCom.getStatus()[0] == 1:
120                         kv.screens[1].ids[str(MainApp.MainCom.
                                getStatus()[3])].dispatch('on_press
                                ')
121
122                 elif MainApp.MainCom.getStatus()[0] == 2:
123                         kv.screens[2].ids[str(MainApp.MainCom.
                                getStatus()[4])].dispatch('on_press
                                ')
124
125                 elif MainApp.MainCom.getStatus()[0] == 3:
126                         kv.screens[1].ids[str(MainApp.MainCom.
                                getStatus()[3])].dispatch('on_press
                                ')
127
128         def btn_prev(self):
129                 if MainApp.MainCom.getStatus()[0] == 1:
130                         MainApp.MainCom.TX(1,MainApp.MainCom.
                                getStatus()[1] - 0.01)
131                         global release_event
132                         release_event = Clock.
                                schedule_interval(self.Decrement,
                                0.2)
133
134                 elif MainApp.MainCom.getStatus()[0] == 2:
```

```
135                         try:
136                                 kv.screens[1].ids[str(MainApp.
                                        MainCom.getStatus()[3] - 1)
                                        ].dispatch('on_press')
137                         except:
138                                 pass
139
140                 elif MainApp.MainCom.getStatus()[0] == 3:
141                         try:
142                                 kv.screens[2].ids[str(MainApp.
                                        MainCom.getStatus()[4] - 1)
                                        ].dispatch('on_press')
143                         except:
144                                 pass
145
146         def Decrement(self, *args):
147                 MainApp.MainCom.TX(1,MainApp.MainCom.getStatus
                        ()[1] - 0.01)
148
149         def cancelDec(self):
150                 if MainApp.MainCom.getStatus()[0] == 1:
151                         release_event.cancel()
152
153         def btn_next(self):
154                 if MainApp.MainCom.getStatus()[0] == 1:
155                         MainApp.MainCom.TX(1,MainApp.MainCom.
                                getStatus()[1] + 0.01)
156                         global down_event
157                         down_event = Clock.schedule_interval(
                                self.Increment, 0.2)
158
159                 elif MainApp.MainCom.getStatus()[0] == 2:
160                         try:
161                                 kv.screens[1].ids[str(MainApp.
                                        MainCom.getStatus()[3] + 1)
                                        ].dispatch('on_press')
162                         except:
163                                 pass
164
165                 elif MainApp.MainCom.getStatus()[0] == 3:
166                         try:
```

```
167                                     kv.screens[2].ids[str(MainApp.
                                            MainCom.getStatus()[4] + 1)
                                            ].dispatch('on_press')
168                             except:
169                                     pass
170
171         def Increment(self, *args):
172                 MainApp.MainCom.TX(1,MainApp.MainCom.getStatus
                        ()[1] + 0.01)
173
174         def cancelInc(self):
175                 if MainApp.MainCom.getStatus()[0] == 1:
176                         down_event.cancel()
177
178 class MetrischeGewinde(Screen):
179         def btn_zero_four(self):
180                 MainApp.MainCom.TX(2,0.4)
181                 MainApp.MainCom.SetBTN(1, 0, 0)
182                 MainApp.MainCom.SetBTN(1, 0, 1)
183                 pass
184
185         def btn_zero_five(self):
186                 MainApp.MainCom.TX(2,0.5)
187                 MainApp.MainCom.SetBTN(1, 1, 0)
188                 MainApp.MainCom.SetBTN(1, 1, 1)
189                 pass
190
191         def btn_zero_seven(self):
192                 MainApp.MainCom.TX(2,0.7)
193                 MainApp.MainCom.SetBTN(1, 2, 0)
194                 MainApp.MainCom.SetBTN(1, 2, 1)
195                 pass
196
197         def btn_zero_eight(self):
198                 MainApp.MainCom.TX(2,0.8)
199                 MainApp.MainCom.SetBTN(1, 3, 0)
200                 MainApp.MainCom.SetBTN(1, 3, 1)
201                 pass
202
203         def btn_one(self):
204                 MainApp.MainCom.TX(2,1.0)
```

```
205                 MainApp.MainCom.SetBTN(1, 4, 0)
206                 MainApp.MainCom.SetBTN(1, 4, 1)
207                 pass
208
209         def btn_one_two_five(self):
210                 MainApp.MainCom.TX(2,1.25)
211                 MainApp.MainCom.SetBTN(1, 5, 0)
212                 MainApp.MainCom.SetBTN(1, 5, 1)
213                 pass
214
215         def btn_one_five(self):
216                 MainApp.MainCom.TX(2,1.5)
217                 MainApp.MainCom.SetBTN(1, 6, 0)
218                 MainApp.MainCom.SetBTN(1, 6, 1)
219                 pass
220
221         def btn_one_seven_five(self):
222                 MainApp.MainCom.TX(2,1.75)
223                 MainApp.MainCom.SetBTN(1, 7, 0)
224                 MainApp.MainCom.SetBTN(1, 7, 1)
225                 pass
226
227         def btn_two(self):
228                 MainApp.MainCom.TX(2,2.0)
229                 MainApp.MainCom.SetBTN(1, 8, 0)
230                 MainApp.MainCom.SetBTN(1, 8, 1)
231                 pass
232
233         def btn_two_five(self):
234                 MainApp.MainCom.TX(2,2.5)
235                 MainApp.MainCom.SetBTN(1, 9, 0)
236                 MainApp.MainCom.SetBTN(1, 9, 1)
237                 pass
238
239         def btn_three(self):
240                 MainApp.MainCom.TX(2,3.0)
241                 MainApp.MainCom.SetBTN(1, 10, 0)
242                 MainApp.MainCom.SetBTN(1, 10, 1)
243                 pass
244         pass
245
```

```
246 class ZollGewinde(Screen):
247         def btn_ten(self):
248                 MainApp.MainCom.TX(3,10.0)
249                 MainApp.MainCom.SetBTN(2, 0, 0)
250                 MainApp.MainCom.SetBTN(2, 0, 1)
251                 pass
252 
253         def btn_eleven(self):
254                 MainApp.MainCom.TX(3,11.0)
255                 MainApp.MainCom.SetBTN(2, 1, 0)
256                 MainApp.MainCom.SetBTN(2, 1, 1)
257                 pass
258 
259         def btn_therteen(self):
260                 MainApp.MainCom.TX(3,13.0)
261                 MainApp.MainCom.SetBTN(2, 2, 0)
262                 MainApp.MainCom.SetBTN(2, 2, 1)
263                 pass
264 
265         def btn_nineteen(self):
266                 MainApp.MainCom.TX(3,19.0)
267                 MainApp.MainCom.SetBTN(2, 3, 0)
268                 MainApp.MainCom.SetBTN(2, 3, 1)
269                 pass
270 
271         def btn_twenty(self):
272                 MainApp.MainCom.TX(3,20.0)
273                 MainApp.MainCom.SetBTN(2, 4, 0)
274                 MainApp.MainCom.SetBTN(2, 4, 1)
275                 pass
276 
277         def btn_twentytwo(self):
278                 MainApp.MainCom.TX(3,22.0)
279                 MainApp.MainCom.SetBTN(2, 5, 0)
280                 MainApp.MainCom.SetBTN(2, 5, 1)
281                 pass
282 
283         def btn_fourty(self):
284                 MainApp.MainCom.TX(3,40.0)
285                 MainApp.MainCom.SetBTN(2, 6, 0)
286                 MainApp.MainCom.SetBTN(2, 6, 1)
```

```
287                 pass
288
289         def btn_fourtyfour(self):
290                 MainApp.MainCom.TX(3,44.0)
291                 MainApp.MainCom.SetBTN(2, 7, 0)
292                 MainApp.MainCom.SetBTN(2, 7, 1)
293                 pass
294         pass
295
296 class SpezialGewinde(Screen):
297         pass
298
299 class SchnittdatenRechner(Screen):
300         pass
301
302 class Einstellungen(Screen):
303         pass
304
305 class WindowManager(ScreenManager):
306         pass
307
308 kv = Builder.load_file("kvroot.kv")
309
310 class MainApp(App):
311
312         MainCom = CommunicationClass()
313
314         def on_start(self):
315                 Clock.schedule_interval(self.Cyclic, 0.1)
316
317         def Cyclic(self, *args):
318                 MainApp.MainCom.initCom()
319                 self.root.screens[0].ids.rpm_lable.text =
                        MainApp.MainCom.RX()
320
321                 if MainApp.MainCom.getStatus()[0] == 1:
322                         self.root.screens[0].ids.btn_gewinde.
                                enabled = 0
323                         self.root.screens[0].ids.btn_normal.
                                enabled = 1
324                         self.root.screens[0].ids.lable_feed.
```

```
                                text = str(MainApp.MainCom.
                                getStatus()[1])+' mm/rev'
325
326             elif MainApp.MainCom.getStatus()[0] == 2:
327                     self.root.screens[0].ids.btn_normal.
                            enabled = 0
328                     self.root.screens[0].ids.btn_gewinde.
                            enabled = 1
329                     self.root.screens[0].ids.btn_gewinde.
                            text = 'Metrisch'
330                     self.root.screens[0].ids.lable_feed.
                            text = str(MainApp.MainCom.
                            getStatus()[1])+' mm'
331
332             elif MainApp.MainCom.getStatus()[0] == 3:
333                     self.root.screens[0].ids.btn_normal.
                            enabled = 0
334                     self.root.screens[0].ids.btn_gewinde.
                            enabled = 1
335                     self.root.screens[0].ids.btn_gewinde.
                            text = 'Zoll'
336                     self.root.screens[0].ids.lable_feed.
                            text = str(MainApp.MainCom.
                            getStatus()[1])+' TPI'
337
338     def build(self):
339             return kv
340
341 if __name__ == "__main__":
342     Raspi = True
343     if Raspi == True:
344             GPIO.setmode(GPIO.BCM)
345             GPIO.setup(2, GPIO.OUT, initial=GPIO.HIGH)
346             sleep(1)
347             GPIO.output(2, GPIO.LOW)
348             sleep(1)
349             GPIO.output(2, GPIO.HIGH)
350     MainApp().run()
```

Eidesstattliche Erklärung

Name: Schwörer	**Vorname:** Lukas
Matrikel-Nr.: 65283	**Studiengang:** Mechatronik

Hiermit versichere ich, **Lukas Schwörer**, an Eides statt, dass ich die vorliegende Bachelorarbeit
an der **Hochschule Aalen**
mit dem Titel **„Developement of an electronic Leadscrew“**
selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht.

Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§23 Abs. 3 des allg. Teils der Bachelor-SPO der Hochschule Aalen) sowie die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Auszug aus dem Strafgesetzbuch (StGB)

§156 StGB Falsche Versicherung an Eides Statt Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Abtsgmünd, 23.02.22
Ort, Datum

Lukas Schwörer
Unterschrift