Hochschule Aalen

University of Aalen

Mechatronic Project - Electronic Lead Screw

January 2022 - Lukas Schwörer

Preface

This project is part of the masters degree program "System engineering" of Aalen University and is scheduled to be performed during the first two semesters. 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.

Dieses Projekt ist Teil des Masterstudiengangs SSystem Engeneering"der Hochschule Aalen und muss während der ersten beiden Semester absolviert werden. Die in diesem Bericht beschriebene praktische Arbeit wurde von Aprill 2021 bis zum Februar 2022 realisiert.
Die praktische Arbeit, wie auch das Schreiben des Berichts wurde Zuhause ausgeführt.

Abstract

This Project is about the development, setup, testing and qualification of an Electronic Leadscrew (ELS). This project was proposed to the university by myself. Its aim is to develop a system to replace the gearbox inside a conventional lathe which will synchronize the rotation of the Leadscrew to the rotation of the spindle. The ELS needs 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 electro-mechanical system of the ELS is build around an encoder to read rotational position of the main spindle and a servo motor to control the position of the Leadscrew. A micro-controller 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 micro-controller 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). Dises Projekt wurde der Universität von mir vorgeschlagen. 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 zu drehen.

Das elektro-mechanische System der ELS besteht aus einem Encoder der die Position der Haupspindel ausliest und einem Servo-Motor, der die Position der Leitspindel kontrolliert. Ein Microcontroller verarbeitet die vom Encoder gesammelten Informationen und bestimmt die korrekte Position des Servo-Motors.

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 Spindel, Encoder, Mikrocontroller und Servo-Motor. 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 an approximate of 6000 students. The University is located in Aalen, a smaller city in the south of Germany.

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.

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 as well as 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 discoverd issues and the solutions for it are supposed to be translated and tested to their functionality in practice.
This developement process can be summarized in the so called V-Modell. The V-Modell is shown in 1.1.

1.2.1. Aim of study

The goal of this project is to add .

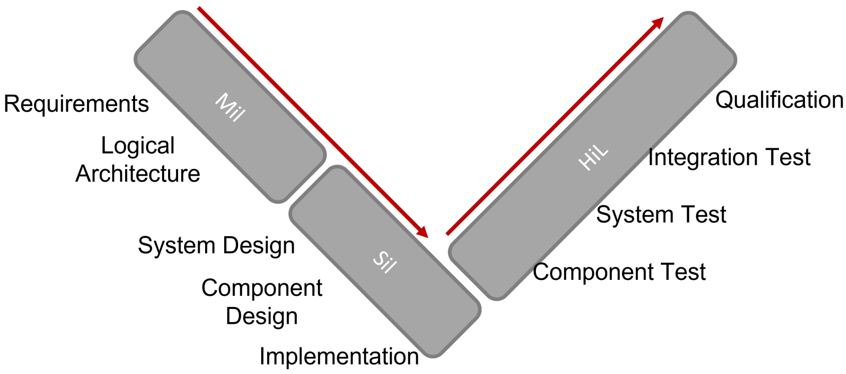

Figure 1.1.: V-Model

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 problems in numerical mathematics is the representation of fractional numbers. This is because computers can only represent a finite subset decimal, due to memory and processing limitations. This can lead to rounding errors that can have an 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.[1]

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.[1]

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.[1]

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.

2.2.1. Additive Manufacturing

Additive manufacturing (AM) is a process, wehre material is build up in order to crate 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 and than squeezed through a nozzle. The molten plastic is than layed down in layers, in order to create a 3D shape. 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.[2]

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 and is cut with a fixed tool. The tool itself only moves in X and Y direction. For this reason, turning is especially well suited for the production of round parts.

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 a 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. The light blocking disk is is perforated with small slits in a rotations 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 squarewave on the output of the photo-sensitive device which frequency is dependant on the rotationary speed of the disk.

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 the 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, where 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 and is very hard to detect. Therefore, stepper motors are often combined with rotational sensors in order to compensate the positional error, when the motor comes out of synchronization.[3]

3.1.3. Closed Loop Servos

Closed Loop Servos is an electromechanical actuator with an continuous feedback loop between its output (speed, torque, position, etc..) and input. Servos require additional electronics to compute the difference between its desired output of the motor and its current output. These electronics are usually referred to as "controller" or "driver". Servos are used in a wide variety of mechatronic applications when high precision, reliability or dynamics are required.[4]

3.1.4. Microcontroller

A Mikrocontroller is an integrated circuit designed to serve a specific task in a system. Mikrocontroller consist of a processing unit, memory and In- and Outputs. Within the system Mikrocontrollers can be used for a wide variety of tasks such as communication, data acquisition or motor control. For mechatronic tasks, microcontrollers are commonly used in an so called embedded system. An embedded system is usually designed around a microcontroller and a specific task within a project.[5]

TI LaunchXL F280049C

The Texas Instruments Launchpad F280049C is a prototyping board designed around their Picolo F280049C Real-Time microcontroller as an embedded system. Its design specificallay targets highly dynamic controll tasks while maintaining a low unit cost. The Picolo F280049C is a 32bit floating point microcontroller with 256KB Flash memory, 100KB RAM and a operation frequency of 100MHz. Further, the microcontroller has hardware support for up to two rotary encoder, a wide variety of communication protocolls and general purpose I/O. In addition, the LaunchPad developement board build around the microcontroller offers a debugging probe. This probe can be used to flash the microcontroller with maschinecode or observe and direct the operation of the microcontroller. This way it is for excample possible to stop the execution of the program on the microcontroller and access the value of every bit in every register of the microcontroller.[6]

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 with the same voltage level. This makes it inevitable to use a logic level shifting circuit.

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 conectivety 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 wide variety of hardware and software add-ons available.[7]

Touchscreen

A touchscreen is a Human Maschine Interface (HMI) that can display information and react to touch inputs. The commonly used touchscreen combines a regular LED screen with touch sensitive layer. Highly sensitive touchscreens that can detect multiple touch inputs at once, usually work on a capacitive principle.

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 B) 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.[8]

3.2.2. Programing Language C

C is a high level programing language. It is most commonly used to write applications for embedded systems. It is chosen over low level programing 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.[9]

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.[10]

Kivy

Kivy is an open source Python library for the rapid development of grafical user interfaces and apps. It is script based and its scripting language can bei directly implemented in Python. Further, kivy supports use with touch screens.[11]

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.[12]

3.2.5. Git

Git is an 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 an mistake the project can be recovered to every tracked point. Git was very heavily used for software development after and during our testing.

3.3. Assembly

In this section the assembly and programming of the electronic Leadscrew will be discussed. This will be split into two parts, covering Hardware and Software.

3.3.1. Hardware Assembly

3.3.2. Programming

4. Experimental

This chapter will show the complete developement process of the Electronic Leadscrew based on the V-Modell.

4.1. Requirements and Logical Architecture

In this section the developement process in the first part of the V-Modell will be described. As shown in figure 4.1 this part is split into two. 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 developement of the System requirements was the first step in the developement process. This first step is one of the most important steps in the developement process, because it defines the functions and properties of all Systemcomponents.A selectoin of the requirements for the Human Machine Interface (HMI), the micro controller (μC), the motor and the motor controller are shown in Table 4.1. The full list of requirements can be found in Appendix C.

4.1.2. Logical Architecture

The next step in the developement process of the ELS was to create the Logical Architecture of the System. This was done as a Toplevel Blockdiagramm as shown in Figure 4.2.

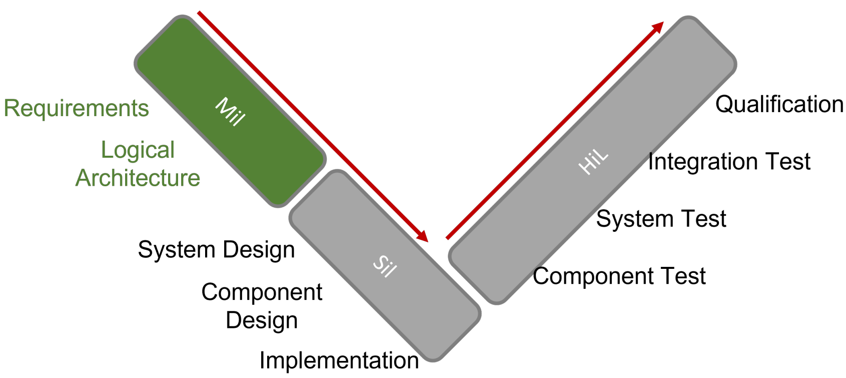

Figure 4.1.: V Model Requirements Stage; Green - Developement status

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

Table 4.1.: Selection of Key Requirements for the ELS

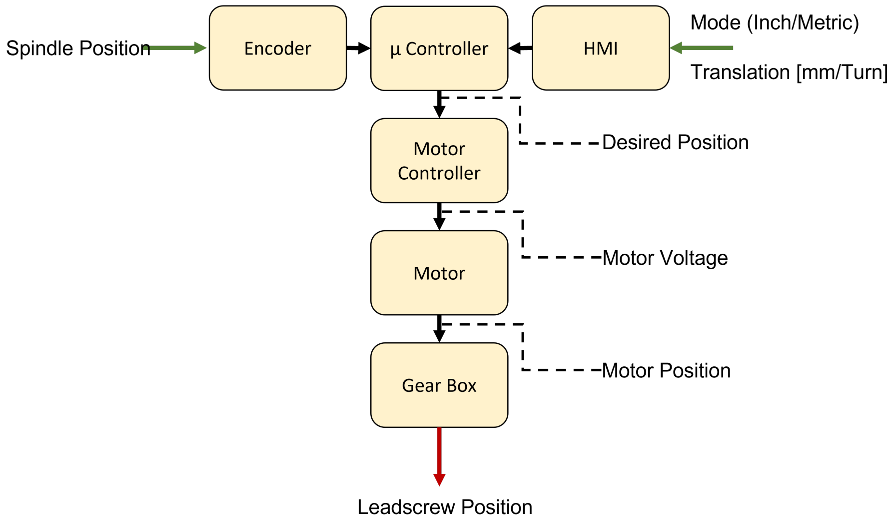

Figure 4.2.: Logical Architecture

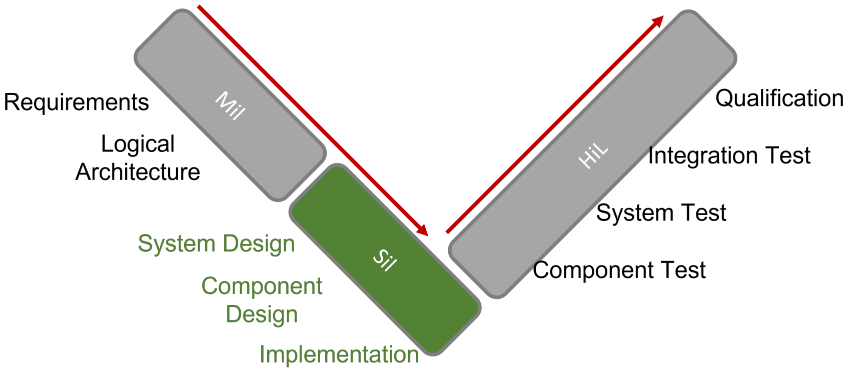

Figure 4.3.: V Model SIL Stage; green - Developement steps SIL stage

4.2. System Design and Implementation

This section describes the second half of the developement process, divided into system and component design as well as the integration of the components. This is the most time consuming process of the developement.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 than transferred into the real system.
This developement 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 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 Leadscrew 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 Leadscrew drive train. However while disassembling the original Leadscrew 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 Leadscrew the manufacturer secured the gear, that is driving the leadscrew with a brass pin. According to the manufacturer, this pin will shear off when more than 1.2Nm is acting on the leadscrew.
After this, all other parameter were determined in a similar matter. The resulting logical structure is shown in Figure 4.4.

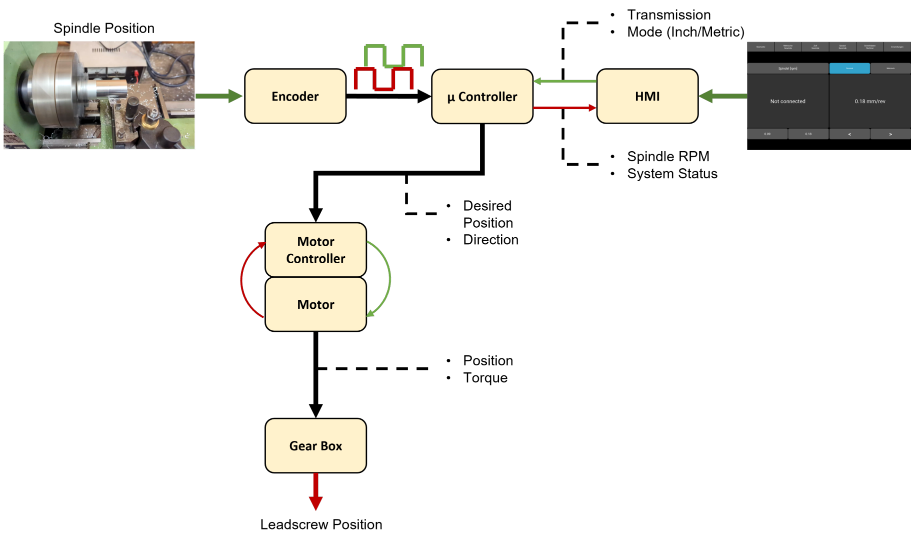

Figure 4.4.: Block diagram of the system design

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

Table 4.2.: Selection of Key Requirements for the ELS

4.2.2. Component Design

HMI

Micro Controller

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 lead to the selection of a Encoder from the manufacturer Opkon (see Appendix B). The Encoder creates 4096 pulses/revolution, with half of them phase shifted by 90°in 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 hight of 57mm, the Encoder just fits into the previously determined specifications.

Motor and Motor Controller

The motor was selected based the requirements list as well as recommendations from James Clough [13]. The selected motor is from the company Steppers Online (see Appendix B). It is a so called "integrated servo moter". 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 4.5.

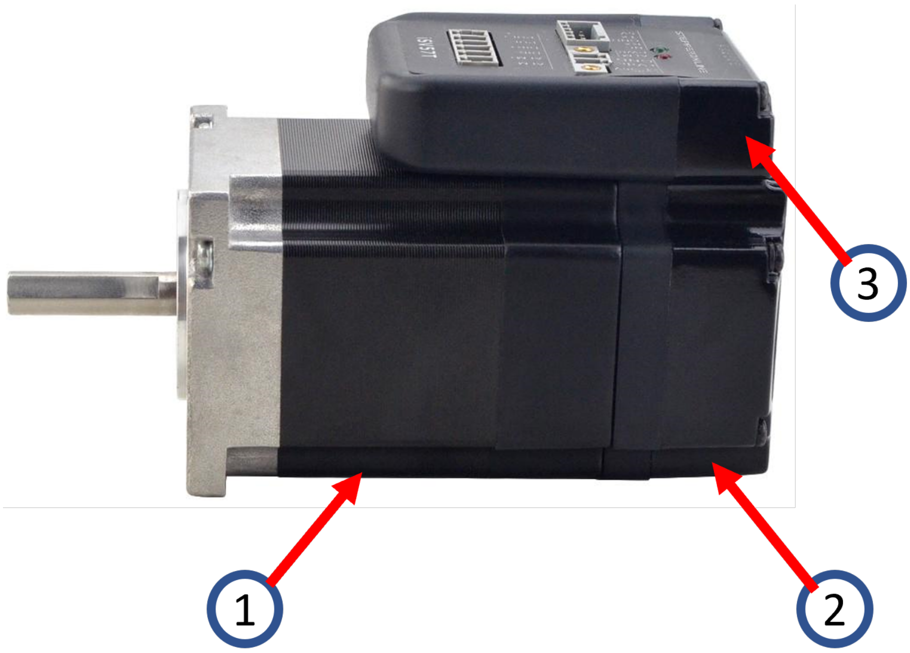

Figure 4.5.: Integrated Servo Motor; 1 - BLDC Motor, 2 - Encoder Unit, 3 - Control Unit

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 micro controller, 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) 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 puls will than move the motor by $\frac{1}{motor-resulution}$.

Gearbox

Because the motor is mounted underneath the Leadscrew, a 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 needed 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 4.6.
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.

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 4.6.: Chart of the Leadscrew dynamics; green - reachable Leadscrew RPM with a 25/80 transmission, red - not reachable Leadscrew RPM with a 25/80 transmission

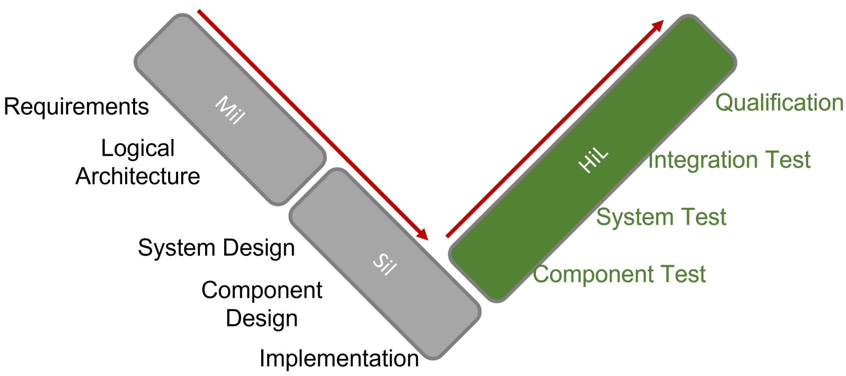

Figure 4.7.: V Model Component Test Stage

4.3. System Testing

4.3.1. Component Test

Encoder

HMI

Micro Controller

Motor

Motor Controller

Gearbox

4.3.2. System Test

4.3.3. Integration Test

Turning

Threading

5. Results and Discussion

5.1. Arithmetic

5.2. Turning

5.3. Threading

6. System Validation

7. Outlook

7.1. Features

8. List of Figures

9. List of Tables

10. References

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

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

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

[4] 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.

[5] 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.

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

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

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

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

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

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

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

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

Appendix

A. Additional Topics

A.1. Pin Out

A.2. External Reset

B. List of Companies

Company: The MathWorks, Inc.

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

C. Requirements ELS

D. Organisation Chart

E. Source Code

E.1. main.c

```
//
// Included Files
//
#include "F28x_Project.h"
#include "Configuration.h"

#include "..\Matlab\RealTimeMachine_ert_rtw\RealTimeMachine.h"
#include "..\Matlab\RealTimeMachine_ert_rtw\rtwtypes.h"
#include "..\Matlab\RealTimeMachine_ert_rtw\zero_crossing_types.h"
#include "..\Matlab\RealTimeMachine_ert_rtw\RealTimeMachine_data.c"

#include "..\Matlab\StepperRTM_ert_rtw\StepperRTM.h"
#include "..\Matlab\StepperRTM_ert_rtw\rtwtypes.h"
#include "..\Matlab\StepperRTM_ert_rtw\zero_crossing_types.h"

#define _FLASH

//
// Global Variables Inputs
//
static uint32_T arg_SpindelPos = 0U;
volatile  real32_T arg_CountFactor = 0;
volatile uint16_T var_StepBacklog;

//
// Global Variables Outputs
//
static uint16_T arg_StepBit;
static uint16_T arg_Dir;
static uint16_T arg_DesSteps;

//
// Global Variables Statemachine Clock
//
static uint16_T System_Trigger[2] = { 0U, 0U };
static boolean_T System_Takt = 0;
static uint16_T Stepper_Trigger[2] = { 0U, 0U };
```

```
static boolean_T Stepper_Takt = 0;

//
// Global Variables Helpers
//
volatile uint32_T current = 0;
volatile uint32_T count = 0;
volatile uint32_T previous = 0;
volatile uint16_T RPM;

volatile int msg[] = {0, 0, 0, 0, 0};
volatile int i = 0;
volatile float feed = 0.0;
volatile int Mode = 1;
volatile int TransferComplete = 0;

void main(void)
{

    #ifdef _FLASH
        // Copy time critical code and Flash setup code to RAM
        // The RamfuncsLoadStart, RamfuncsLoadEnd, and
            RamfuncsRunStart
        // symbols are created by the linker. Refer to the
            linker files.
        memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)
            &RamfuncsLoadSize);

        // Initialize the flash instruction fetch pipeline
        // This configures the MCU to pre-fetch instructions
            from flash.
        InitFlash();
    #endif


    //
    // Initialize Autocode
    //
    StepperRTM_initialize();
    RealTimeMachine_initialize();
    //
```

```
    // Initialize device clock and peripherals
    //
    InitSysCtrl();

    //
    // Initialize GPIO, Timer and EQEP
    //
    InitGpio();
    setupGPIO();
    setupTimer();
    setupEQEP();




    //
    // Initialize UART
    //
    initSCIAFIFO();
    initSCIAEchoback();





    while(1)
    {
        //
        // Send RPM via UART
        //
        if(EQep1Regs.QFLG.bit.UTO==1)
        {

            Uint32 current = EQep1Regs.QPOSLAT;
            Uint32 count = (current > previous) ? current -
                previous : previous - current;

            // deal with over/underflow
            if( count > _ENCODER_MAX_COUNT/2 )
             {
                count = _ENCODER_MAX_COUNT - count; // just
```

```
                subtract from max value
        }

        RPM = count * 60 * RPMSampleTime / EncoderRes;

        int highbyte = RPM >> 8;
        int lowbyte = RPM & 0x00ff;

        previous = current;
        transmitSCIAChar(lowbyte);          // Send RPM
            out via UART
        transmitSCIAChar(highbyte);         // Send RPM
            out via UART
        EQep1Regs.QCLR.bit.UTO=1;           // Clear interrupt
            flag
    }

    //
    // Check for new Massages from the Raspberry Pi
    //
    if(SciaRegs.SCIFFRX.bit.RXFFST != 0)
    {
        msg[i] = SciaRegs.SCIRXBUF.bit.SAR;

        if(i == 4)
        {
            i = 0;
            TransferComplete = 1;
            Mode = msg[1];
            feed = msg[2] + (msg[3] * 0.01);
        }
        else
        {
            i++;
        }
    }

    //
    // Calculate Step-factor based on information received
        via UART
    //
```

```
        if(TransferComplete && (msg[0] == 0xff) && (msg[4] ==
            0xff))
        {
            TransferComplete = 0;
             // Normal Feed and metric thread cutting
            if(Mode == 1 || Mode == 2)
            {
                arg_CountFactor = ((Steps * MotorTransmission *
                    EncoderTransmission * feed)/(EncoderRes *
                    LeadscrewSlope));
            }

             // Imperial thread cutting
            else if(Mode == 3)
            {
                arg_CountFactor = ((Steps * MotorTransmission *
                    EncoderTransmission * OneInch)/(EncoderRes
                    * LeadscrewSlope * feed ));
            }
        }
    }
}

//
// cpuTimer0ISR - CPU Timer0 ISR
//
__interrupt void cpuTimer0ISR(void)
{

    Stepper_Takt = !Stepper_Takt; // Toggle System Clock
    Stepper_Trigger[0] = (uint16_T)Stepper_Takt;

    if(Stepper_Takt == 0)
    {
        Stepper_Trigger[1] = 1; //Power on Reset
    }

    StepperRTM_step(var_StepBacklog, (uint16_t*)&
        Stepper_Trigger, &arg_StepBit);

```

```
    //
    // Stepper Clock for Debugging
    //

    GpioDataRegs.GPASET.bit.GPIO6 = arg_StepBit;
    GpioDataRegs.GPADAT.bit.GPIO6 = arg_StepBit;

    //
    // Acknowledge this interrupt to receive more
    //
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}



//
// cpuTimer0ISR - CPU Timer2 ISR
//
__interrupt void cpuTimer2ISR(void)
{
    System_Takt = !System_Takt; // Toggle System Clock
    System_Trigger[0] = (uint16_T)System_Takt;

    if(System_Takt == 0)
    {
        System_Trigger[1] = 1; //Power on Reset
    }

    arg_SpindelPos = EQep1Regs.QPOSCNT;

    RealTimeMachine_step(arg_SpindelPos, arg_CountFactor, (
        uint16_t*)&System_Trigger,
                      &arg_DesSteps, &arg_Dir);

    var_StepBacklog = var_StepBacklog + arg_DesSteps;

    GpioDataRegs.GPBSET.bit.GPIO39 = !arg_Dir;
    GpioDataRegs.GPBDAT.bit.GPIO39 = !arg_Dir;

    //
    // Acknowledge this interrupt to receive more
```

```
    //
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

E.2. Configuration.h

```
/*
 * Configuration.h
 *
 *  Created on: 26 Oct 2021
 *      Author: Lukas Schwoerer
 */
#include "F28x_Project.h"

//
// Predefine Functions
//

void setupGPIO(void);
void setupTimer(void);
void setupEQEP(void);

void initSCIAEchoback(void);
void transmitSCIAChar(int msg);
void initSCIAFIFO(void);

__interrupt void cpuTimer0ISR(void);
__interrupt void cpuTimer2ISR(void);

#ifndef CONFIGURATION_H_
#define CONFIGURATION_H_

//
// Statemachine cycle times
//
#define Stepper_Clock 5
#define System_Clock 100

//
// Refreshrate RPM (DO NOT EDIT)
//
#define RefreshRate 100

//
// Hardware constants
```

```
//
#define _ENCODER_MAX_COUNT        0x00ffffff
#define MotorTransmission         3.2
#define EncoderTransmission       1
#define Steps                     2000
#define EncoderRes                4096
#define LeadscrewSlope            1.5
#define OneInch                   25.4
#define RPMSampleTime             5            //Sample Rate RPM in Hz
#endif /* CONFIGURATION_H_ */
```

E.3. Configuration.c

```
/*
 * Configuration.c
 *
 *  Created on: 26 Oct 2021
 *      Author: Lukas Schwoerer
 */

#include "Configuration.h"
#include "F28x_Project.h"


void setupTimer(void)
{

    // Disable CPU interrupts
    //
    DINT;

    //
    // Initialize the PIE control registers to their default
        state.
    // The default state is all PIE interrupts disabled and
        flags
    // are cleared.
    //
    InitPieCtrl();

    //
    // Disable CPU interrupts and clear all CPU interrupt
        flags
    //
    IER = 0x0000;
    IFR = 0x0000;

    //
    // Initialize the PIE vector table with pointers to the
        shell Interrupt
    // Service Routines (ISR)
    //
```

```
    InitPieVectTable();

    //
    // Map ISR functions
    //
    EALLOW;
    PieVectTable.TIMER0_INT = &cpuTimer0ISR;
    PieVectTable.TIMER2_INT = &cpuTimer2ISR;
    EDIS;

    //
    // Initialize the Device Peripheral. For this example,
        only initialize the
    // Cpu Timers.
    //
    InitCpuTimers();

    //
    // Configure CPU-Timer 0 and 2
    // 100MHz CPU Freq, Clock in uSeconds
    //
    ConfigCpuTimer(&CpuTimer0, 100, Stepper_Clock);
    ConfigCpuTimer(&CpuTimer2, 100, System_Clock);

    //
    // To ensure precise timing, use write-only instructions
        to write to the
    // entire register. Therefore, if any of the configuration
         bits are changed
    // in ConfigCpuTimer and InitCpuTimers, the below settings
         must also be
    // be updated.
    //
    CpuTimer0Regs.TCR.all = 0x4000;
    CpuTimer2Regs.TCR.all = 0x4000;

    //
    // Enable CPU int1 which is connected to CPU-Timer 0, CPU
        int13
    // which is connected to CPU-Timer 1, and CPU int 14,
        which is connected
```

```
    // to CPU-Timer 2
    //
    IER |= M_INT1;
    IER |= M_INT14;

    //
    // Enable TINT0 in the PIE: Group 1 interrupt 7
    //
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

    //
    // Enable global Interrupts and higher priority real-time
        debug events
    //
    EINT;
    ERTM;

}

void setupGPIO(void)
{
    EALLOW;
    //
    // Setup Port A
    //
    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;        // Enable pull-up
        on GPIO6 (DirPin)
    GpioCtrlRegs.GPAQSEL1.bit.GPIO6 = 0;      // Sync to
        SYSCLKOUT GPIO6 (DirPin)
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;       // Configure GPIO6
          as GPIO
    GpioCtrlRegs.GPAGMUX1.bit.GPIO6 = 0;
    GpioDataRegs.GPASET.bit.GPIO6 = 0;        // Configure GPIO6
          as Output
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;

    GpioCtrlRegs.GPAPUD.bit.GPIO23 = 0;       // Enable pull-up
        on GPIO23 (Step Pin)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO23 = 0;     // Sync to
        SYSCLKOUT GPIO23 (Step Pin)
    GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;      // Configure
```

```
    GPIO23 as GPIO
    GpioCtrlRegs.GPAGMUX2.bit.GPIO23 = 0;
    GpioDataRegs.GPASET.bit.GPIO23 = 0;      // Configure
        GPIO23 as Output
    GpioCtrlRegs.GPADIR.bit.GPIO23 = 1;

    //
    // Setup Port B for EQEP1
    //
    GpioCtrlRegs.GPBPUD.bit.GPIO35 = 0;      // Enable pull-up
        on GPIO35 (EQEP1A)
    GpioCtrlRegs.GPBPUD.bit.GPIO37 = 0;      // Enable pull-up
        on GPIO371 (EQEP1B)
    GpioCtrlRegs.GPBPUD.bit.GPIO59 = 0;      // Enable pull-up
        on GPIO59 (EQEP1I)

    GpioCtrlRegs.GPBQSEL1.bit.GPIO35 = 0;    // Sync to
        SYSCLKOUT GPIO35 (EQEP1A)
    GpioCtrlRegs.GPBQSEL1.bit.GPIO37 = 0;    // Sync to
        SYSCLKOUT GPIO37 (EQEP1B)
    GpioCtrlRegs.GPBQSEL2.bit.GPIO59 = 0;    // Sync to
        SYSCLKOUT GPIO59 (EQEP1I)

    GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 1;     // Configure
        GPIO35 as EQEP1A
    GpioCtrlRegs.GPBGMUX1.bit.GPIO35 = 2;
    GpioCtrlRegs.GPBMUX1.bit.GPIO37 = 1;     // Configure
        GPIO37 as EQEP1B
    GpioCtrlRegs.GPBGMUX1.bit.GPIO37 = 2;
    GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 3;     // Configure
        GPIO59 as EQEP1I
    GpioCtrlRegs.GPBGMUX2.bit.GPIO59 = 2;

    GpioCtrlRegs.GPBPUD.bit.GPIO39 = 0;      // Enable pull-up
        on GPIO39 (EnablePin)
    GpioCtrlRegs.GPBQSEL1.bit.GPIO39 = 0;    // Sync to
        SYSCLKOUT GPIO39 (EnablePin)
    GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0;     // Configure
        GPIO39 as GPIO
    GpioCtrlRegs.GPBGMUX1.bit.GPIO39 = 0;
    GpioDataRegs.GPBSET.bit.GPIO39 = 0;      // Configure
```

```
        GPIO39 as Output
    GpioCtrlRegs.GPBDIR.bit.GPIO39 = 1;

    EDIS;
}

void setupEQEP(void)
{


    EQep1Regs.QDECCTL.bit.QSRC = 0;            // QEP quadrature
        count mode
    EQep1Regs.QDECCTL.bit.IGATE = 1;           // gate the index
        pin
    EQep1Regs.QDECCTL.bit.QAP = 1;             // invert A input
    EQep1Regs.QDECCTL.bit.QBP = 1;             // invert B input
    EQep1Regs.QDECCTL.bit.QIP = 1;             // invert index
        input
    EQep1Regs.QEPCTL.bit.FREE_SOFT = 2;        // unaffected by
        emulation suspend
    EQep1Regs.QEPCTL.bit.PCRM = 1;             // position count
        reset on maximum position
    EQep1Regs.QPOSMAX = 0x00ffffff;

    EQep1Regs.QUPRD = 100000000/RPMSampleTime;// Unit Timer
        latch at RPM_CALC_RATE_HZ Hz
    EQep1Regs.QEPCTL.bit.UTE=1;                // Unit Timeout
        Enable
    EQep1Regs.QEPCTL.bit.QCLM=1;               // Latch on unit
        time out
    EQep1Regs.QEPCTL.bit.QPEN=1;               // QEP enable
}

void initSCIAFIFO(void)
{
    GPIO_SetupPinMux(28, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinOptions(28, GPIO_INPUT, GPIO_PUSHPULL);
    GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_ASYNC);

    SciaRegs.SCIFFTX.all = 0xE040;
```

```
    SciaRegs.SCIFFRX.all = 0x2044;
    SciaRegs.SCIFFCT.all = 0x0;
}

void initSCIAEchoback(void)
{
    //
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function
    //
    SciaRegs.SCICCR.all = 0x0007;           // 1 stop bit,  No loopback
                                            // No parity, 8 char bits,
                                            // async mode, idle-line protocol
    SciaRegs.SCICTL1.all = 0x0003;          // enable TX, RX, internal SCICLK,
                                            // Disable RX ERR, SLEEP, TXWAKE
    SciaRegs.SCICTL2.all = 0x0003;
    SciaRegs.SCICTL2.bit.TXINTENA = 1;
    SciaRegs.SCICTL2.bit.RXBKINTENA = 1;

    //
    // SCIA at 9600 baud
    // @LSPCLK = 25 MHz (100 MHz SYSCLK) HBAUD = 0x01  and LBAUD = 0x44.
    //
    SciaRegs.SCIHBAUD.all = 0x0001;
    SciaRegs.SCILBAUD.all = 0x0044;

    SciaRegs.SCICTL1.all = 0x0023;          // Relinquish SCI from Reset
}

//
// transmitSCIAChar - Transmit a character from the SCI
//
void transmitSCIAChar(int msg)
```

```
{
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0)
    {
    }
    SciaRegs.SCITXBUF.all = msg;
}
```

E.4. 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):
```

```
                for m in range(0,14):
                    try:
                        kv.screens[n].ids[str(m)].enabled = 0
                    except:
                        pass

    def getStatus(self):
        return self.Mode, self.Feed, self.serialIndicator, self.Metric_BTN, self.Imperial_BTN, self.FeedFeed

    def initCom(self):

        if self.serialIndicator == 0:
            try:
                self.ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 0.2)
                sleep(1)
                self.serialIndicator = 1
                self.Mode = 'Normal'

            except:
                self.serialIndicator = 0
            pass

    def TX(self, Mode, Feed):
        if Mode == 1:
            self.FeedFeed = Feed

        self.Mode = int(Mode)
        self.Feed = round(Feed, 2)
        self.FeedInt = int(Feed)
        self.FeedDez = int((Feed - int(Feed))*100)

        if self.serialIndicator:
            self.ser.write(b'\xff')
            self.ser.write(self.Mode.to_bytes(1,
```

```
                        byteorder='big'))
                self.ser.write(self.FeedInt.to_bytes(1, byteorder='big'))
                self.ser.write(self.FeedDez.to_bytes(1, byteorder='big'))
                self.ser.write(b'\xff')
                print('Transmission completed')

    def RX(self):

        lowbyte = 0
        highbyte = 0
        TransferComplete = 0

        if self.serialIndicator:
            while not TransferComplete:
                if self.ser.in_waiting == 2:
                    lowbyte = int.from_bytes(self.ser.read(1), 'big', signed=False)
                    highbyte = int.from_bytes(self.ser.read(1), 'big', signed=False)
                    highbyte = highbyte << 8
                    self.RPM = lowbyte + highbyte
                    self.ser.flushInput()
                    TransferComplete = 1
                    return str(self.RPM)


                elif self.ser.in_waiting > 2:
                    self.ser.flushInput()

                else:
                    return str(self.RPM)
        else:
            return 'Not connected'
```

```python

class Startseite(Screen):
        def btn_defone(self):
                MainApp.MainCom.SetBTN(0,0,0)
                MainApp.MainCom.TX(1, 0.09)

        def btn_deftwo(self):
                MainApp.MainCom.SetBTN(0,0,0)
                MainApp.MainCom.TX(1, 0.18)

        def btn_normal(self):
                feed = MainApp.MainCom.getStatus()[5]
                MainApp.MainCom.TX(1, feed)
                MainApp.MainCom.SetBTN(0,0,0)

        def btn_gewinde(self):
                if MainApp.MainCom.getStatus()[0] == 1:
                        kv.screens[1].ids[str(MainApp.MainCom.
                            getStatus()[3])].dispatch('on_press
                            ')

                elif MainApp.MainCom.getStatus()[0] == 2:
                        kv.screens[2].ids[str(MainApp.MainCom.
                            getStatus()[4])].dispatch('on_press
                            ')

                elif MainApp.MainCom.getStatus()[0] == 3:
                        kv.screens[1].ids[str(MainApp.MainCom.
                            getStatus()[3])].dispatch('on_press
                            ')

        def btn_prev(self):
                if MainApp.MainCom.getStatus()[0] == 1:
                        MainApp.MainCom.TX(1,MainApp.MainCom.
                            getStatus()[1] - 0.01)
                        global release_event
                        release_event = Clock.
                            schedule_interval(self.Decrement,
                            0.2)

                elif MainApp.MainCom.getStatus()[0] == 2:
```

```
                try:
                    kv.screens[1].ids[str(MainApp.MainCom.getStatus()[3] - 1)].dispatch('on_press')
                except:
                    pass

            elif MainApp.MainCom.getStatus()[0] == 3:
                try:
                    kv.screens[2].ids[str(MainApp.MainCom.getStatus()[4] - 1)].dispatch('on_press')
                except:
                    pass

    def Decrement(self, *args):
        MainApp.MainCom.TX(1,MainApp.MainCom.getStatus()[1] - 0.01)

    def cancelDec(self):
        if MainApp.MainCom.getStatus()[0] == 1:
            release_event.cancel()

    def btn_next(self):
        if MainApp.MainCom.getStatus()[0] == 1:
            MainApp.MainCom.TX(1,MainApp.MainCom.getStatus()[1] + 0.01)
            global down_event
            down_event = Clock.schedule_interval(self.Increment, 0.2)

        elif MainApp.MainCom.getStatus()[0] == 2:
            try:
                kv.screens[1].ids[str(MainApp.MainCom.getStatus()[3] + 1)].dispatch('on_press')
            except:
                pass

        elif MainApp.MainCom.getStatus()[0] == 3:
            try:
```

```
                    kv.screens[2].ids[str(MainApp.
                        MainCom.getStatus()[4] + 1)
                        ].dispatch('on_press')
                except:
                    pass

    def Increment(self, *args):
        MainApp.MainCom.TX(1,MainApp.MainCom.getStatus
            ()[1] + 0.01)

    def cancelInc(self):
        if MainApp.MainCom.getStatus()[0] == 1:
            down_event.cancel()

class MetrischeGewinde(Screen):
    def btn_zero_four(self):
        MainApp.MainCom.TX(2,0.4)
        MainApp.MainCom.SetBTN(1, 0, 0)
        MainApp.MainCom.SetBTN(1, 0, 1)
        pass

    def btn_zero_five(self):
        MainApp.MainCom.TX(2,0.5)
        MainApp.MainCom.SetBTN(1, 1, 0)
        MainApp.MainCom.SetBTN(1, 1, 1)
        pass

    def btn_zero_seven(self):
        MainApp.MainCom.TX(2,0.7)
        MainApp.MainCom.SetBTN(1, 2, 0)
        MainApp.MainCom.SetBTN(1, 2, 1)
        pass

    def btn_zero_eight(self):
        MainApp.MainCom.TX(2,0.8)
        MainApp.MainCom.SetBTN(1, 3, 0)
        MainApp.MainCom.SetBTN(1, 3, 1)
        pass

    def btn_one(self):
        MainApp.MainCom.TX(2,1.0)
```

```
            MainApp.MainCom.SetBTN(1, 4, 0)
            MainApp.MainCom.SetBTN(1, 4, 1)
            pass

    def btn_one_two_five(self):
            MainApp.MainCom.TX(2,1.25)
            MainApp.MainCom.SetBTN(1, 5, 0)
            MainApp.MainCom.SetBTN(1, 5, 1)
            pass

    def btn_one_five(self):
            MainApp.MainCom.TX(2,1.5)
            MainApp.MainCom.SetBTN(1, 6, 0)
            MainApp.MainCom.SetBTN(1, 6, 1)
            pass

    def btn_one_seven_five(self):
            MainApp.MainCom.TX(2,1.75)
            MainApp.MainCom.SetBTN(1, 7, 0)
            MainApp.MainCom.SetBTN(1, 7, 1)
            pass

    def btn_two(self):
            MainApp.MainCom.TX(2,2.0)
            MainApp.MainCom.SetBTN(1, 8, 0)
            MainApp.MainCom.SetBTN(1, 8, 1)
            pass

    def btn_two_five(self):
            MainApp.MainCom.TX(2,2.5)
            MainApp.MainCom.SetBTN(1, 9, 0)
            MainApp.MainCom.SetBTN(1, 9, 1)
            pass

    def btn_three(self):
            MainApp.MainCom.TX(2,3.0)
            MainApp.MainCom.SetBTN(1, 10, 0)
            MainApp.MainCom.SetBTN(1, 10, 1)
            pass
    pass
```

```
class ZollGewinde(Screen):
        def btn_ten(self):
                MainApp.MainCom.TX(3,10.0)
                MainApp.MainCom.SetBTN(2, 0, 0)
                MainApp.MainCom.SetBTN(2, 0, 1)
                pass

        def btn_eleven(self):
                MainApp.MainCom.TX(3,11.0)
                MainApp.MainCom.SetBTN(2, 1, 0)
                MainApp.MainCom.SetBTN(2, 1, 1)
                pass

        def btn_therteen(self):
                MainApp.MainCom.TX(3,13.0)
                MainApp.MainCom.SetBTN(2, 2, 0)
                MainApp.MainCom.SetBTN(2, 2, 1)
                pass

        def btn_nineteen(self):
                MainApp.MainCom.TX(3,19.0)
                MainApp.MainCom.SetBTN(2, 3, 0)
                MainApp.MainCom.SetBTN(2, 3, 1)
                pass

        def btn_twenty(self):
                MainApp.MainCom.TX(3,20.0)
                MainApp.MainCom.SetBTN(2, 4, 0)
                MainApp.MainCom.SetBTN(2, 4, 1)
                pass

        def btn_twentytwo(self):
                MainApp.MainCom.TX(3,22.0)
                MainApp.MainCom.SetBTN(2, 5, 0)
                MainApp.MainCom.SetBTN(2, 5, 1)
                pass

        def btn_fourty(self):
                MainApp.MainCom.TX(3,40.0)
                MainApp.MainCom.SetBTN(2, 6, 0)
                MainApp.MainCom.SetBTN(2, 6, 1)
```

```
                        pass

        def btn_fourtyfour(self):
                MainApp.MainCom.TX(3,44.0)
                MainApp.MainCom.SetBTN(2, 7, 0)
                MainApp.MainCom.SetBTN(2, 7, 1)
                pass
        pass

class SpezialGewinde(Screen):
        pass

class SchnittdatenRechner(Screen):
        pass

class Einstellungen(Screen):
        pass

class WindowManager(ScreenManager):
        pass

kv = Builder.load_file("kvroot.kv")

class MainApp(App):

        MainCom = CommunicationClass()

        def on_start(self):
                Clock.schedule_interval(self.Cyclic, 0.1)

        def Cyclic(self, *args):
                MainApp.MainCom.initCom()
                self.root.screens[0].ids.rpm_lable.text = MainApp.MainCom.RX()

                if MainApp.MainCom.getStatus()[0] == 1:
                        self.root.screens[0].ids.btn_gewinde.enabled = 0
                        self.root.screens[0].ids.btn_normal.enabled = 1
                        self.root.screens[0].ids.lable_feed.
```

```
text = str(MainApp.MainCom.getStatus()[1])+' mm/rev'

            elif MainApp.MainCom.getStatus()[0] == 2:
                self.root.screens[0].ids.btn_normal.enabled = 0
                self.root.screens[0].ids.btn_gewinde.enabled = 1
                self.root.screens[0].ids.btn_gewinde.text = 'Metrisch'
                self.root.screens[0].ids.lable_feed.text = str(MainApp.MainCom.getStatus()[1])+' mm'

            elif MainApp.MainCom.getStatus()[0] == 3:
                self.root.screens[0].ids.btn_normal.enabled = 0
                self.root.screens[0].ids.btn_gewinde.enabled = 1
                self.root.screens[0].ids.btn_gewinde.text = 'Zoll'
                self.root.screens[0].ids.lable_feed.text = str(MainApp.MainCom.getStatus()[1])+' TPI'

    def build(self):
        return kv

if __name__ == "__main__":
    Raspi = True
    if Raspi == True:
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(2, GPIO.OUT, initial=GPIO.HIGH)
        sleep(1)
        GPIO.output(2, GPIO.LOW)
        sleep(1)
        GPIO.output(2, GPIO.HIGH)
    MainApp().run()
```