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:

- **Person** Reason.

Table of Contents

1. Introduction

In this chapter the working environment, project background and all involved companies will be introduced. Furthermore the aim and limitations of the project will be described.

1.1. Working environment

1.1.1. University of Aalen

1.1.2. Home office

Metall workshop

1.2. Project background

1.2.1. Aim of study

1.2.2. Limitations

2. Theoretical Background

2.1. Numerical Mathmatic

2.1.1. Fractional Mathmatic

2.1.2. Floatingpoint Mathmatic

2.2. Manufactoring

2.2.1. Additive Manufactoring

2.2.2. Subtractive Manufactoring

3. Hardware and Software

In this Chapter the needed background information for the used hardware and software will be described.

3.1. Hardware

3.1.1. Electro mechanical actuators

Stepper Motors

Closed Loop Servos

3.1.2. Microcontroller

TI LaunchXL F280049C

Logic Level Shifter

3.1.3. Raspberry Pi

A Raspberry Pi is a Single board computer.

Touchscreen

Touchscreens can be either resistive or capacitive. The react to them being touched and are used to interact with electronic devices.

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 Appendix 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.

3.2.2. Code Composer Studio

C Code

C is a programming language which is used when quick code execution and a small program size is needed.

3.2.3. Python

Python is a object oriented programming language.

Kivy

Kivy is a Python software Package for the creation of graphical user interfaces.

3.2.4. 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. [25] 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. At some time 4 developers where working in parallel on software tooling for data recording and data analysis. The structure used for this development was the “git-flow” Workflow [26]. This structure is shown in Figure 13. It features a master branch, an development branch and separate branches for new features. To be able to fix Bugs on the Master-branch quickly, this is done as “Hotfix” on a separate branch.

4. Experimental

4.1. Model in the Loop

4.1.1. Initial Considerations

4.1.2. Parameter Identification

4.1.3. Simulink Model

Encoder and Spindel

Microcontroler

Stepper Motor

Gearbox

4.1.4. Code Generation

4.1.5. Mechanical Implementation

4.2. Software in the Loop

4.2.1. Logic

4.2.2. Arithmetic

4.3. Hardware in the Loop

4.3.1. Component Test

4.3.2. System Test

4.3.3. Integration Test

Turning

Threading

5. Results and Discussion

5.0.1. Arythmatic

5.0.2. Threading

5.0.3. Turning

6. System Validation

7. Outlook

7.1. Features

8. List of Figures

9. List of Tables

Appendix

A. Additional Topics

A.1. Pin Out

A.2. External Reset

B. List of Companies

Company: Volvo Cars

Website: `https://www.volvocars.com/se`

Company: The MathWorks, Inc.

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

Company: National Instruments

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

Company: Tamron

Website: https://www.tamron.com/

Company: LUCID Vision Labs

Website: https://thinklucid.com/

Company: Thorlabs, Inc.

Website: https://www.thorlabs.com/

Company: DIGI International, Inc.

Website: https://www.digi.com/

Company: MikroTik

Website: https://mikrotik.com/

C. Network setup and configuration

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
```

```c
    // 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

```python
#!/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,
```

```python
                    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()
```