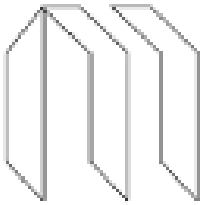


Wydział Mechatroniki

POLITECHNIKA WARSZAWSKA

course
FMDII

Spring-O-Meter 2000

Spring Measurement Device

Konrad Kosuge, Pawel Seroka
student record book number

supervisor
Dr. Ksawery Sziekedans

Contents

1. Introduction

Presented here is the device Spring-O-Meter 1000, created for the class FMDII at the faculty of Mechatronics. The aim of this project was to measure the spring constant in both tension and compression springs by building some device. Being a Mechatronic device it contains Electrical, Software and Mechanical Components that work in synchrony. The following documentation will contain a description of the 3 components as well as the usage of the device. For the the Step and Program Files go to
https://github.com/Soundwave0/Spring-o-meter.git

2. Specifications

Spec	Value	Unit
Range Comp.	90	mm
Range Ten.	115	mm
MAX Comp pos	110	mm
MIN Comp. pos	20	mm
MAX Ten. pos	135	mm
MIN Ten. pos	10	mm
Max Comp. Di	19	mm
MIN Comp. Di	3	mm
MAX force	9.806	N
MAX cal. weight	950	g
MIN cal. weight	3	g
Disp per Rotation	2	mm
Gear Ratio(lead:enc.)	1:3	
Encoder Disp. Resolution	0.022	mm
Force sensor Accuracy	≈0.1	%
K Accuracy	≈0.015	N/mm

3. Usage

1. Download the Spring-O-Meter GUI code either from the git-hub link or install from the executable (if you face any issues in the git-hub version try turning off signing)
2. Plug in the Micro-USB cable into the Spring-O-Meter and then into the USB port in your computer, see if the blue led begins to shine through the support on the bottom. Remember if your Micro-USB cable does not have data transfer capabilities it will turn on but no serial communication will be possible
3. Go to Device Manager-> Ports and check to see if you can see the Silicon Labs CP210x, remember the COM port it is connected to
4. Launch the GUI and click on "COM Configuration" and select the COM that you found in the previous step
5. Press Check COM if it informs you that everything is good then continue if not restart the application and recheck the COM in device manager
6. If you would like to Calibrate your sensor then turn on Calibration , wait until the button settles then place your Spring-O-Meter on its head then press "Calibrate Zero", after this enter the weight of your test weight then place it, then press "Complete Calibration". Once this button is changed to "Calibrated" continue
7. If you don't want to Calibrate but instead want to use a previous value of Calibration taken from EEPROM, then press Use Prev Cal instead of doing Step 6
8. Next place your Spring-O-Meter in the testing Position, install your spring and press tare, then reset
9. If you would like to measure the point at which the spring begins to exert a force on the sensor to set your zero better, turn on the kickpoint by pressing the button next to kickpoint label turning the OFF to ON. Move the Stage to tense or compress the spring until the kickpoint is no longer null. Once this happens, move your stage to kick point, tare and reset
10. Now you can begin sampling, move your stage a bit, wait for the measurement to settle then press sample, repeat for desired number of samples then press Graph Spring
11. To reset everything and repeat another trial click New Trial

4. Parts

4.1. Stock-Parts

NAME	SPEC	QTY
LSU		1
ESP8266	NodeMCU 12-E	1
Encoder 30IMP	KY-040	1
Load-Cell 1Kg	NA27	1
Load-Cell Amp	HX711	1
M2.5x40		2
M6x12	ISO4762	4
M2.5 pin		1
micro-USB cable	W/ Data Bus	1
Breadboard cables		12

4.2. 3D printed parts

NAME	SPEC	QTY
Spur Gear1	3d printed	1
Spur Gear Clamp	3d printed	1
Spur Gear 2	3d printed	1
Base 1	3d printed	1
Base 2	3d printed	1
Base Lid	3d printed	1
Cable Lid	3d printed	1
Clamp	3d printed	1
Plate Big	3d printed	1
Plate Small Gear	3d printed	1
Plate Small	3d printed	1

4.2.1. Spur Gears

Parts (Spur Gear 1 and Spur Gear 2)used in gearing mechanism.

4.2.2. Spur Gear Clamp

Clamp is used to fix the Spur Gear 2 onto the screw shaft and prevent it from sliding on the thread by tightening with screws.

4.2.3. Supports

Base 1 with Base Lid is used to hide the electronics (amplifier,esp controller). Combined with Base 2 it lifts the device so that there is clearance between ground and the moving gears.

4.2.4. Plates

Plate Big is used to provide mounting point for the force sensor and to hide cables in the space inside it, closed by the Base Lid.
Plate Small is used as a mounting point for the springs.
Plate Small Gear is used to mount angle sensor with a Spur Gear 1 attached to it.

4.2.5. Clamp

Clamp is used to provide attachment point for the springs to the force sensor.

5. Mechanical Implementation

5.1. Gearing

To measure the rotation of the screw shaft we need to be connect it to the sensor. This can be obtained by using belts, chains or gears. For this project we chose the gear system, due to ease of obtaining gears by 3D printing. Additionally we introduced the gear ratio to improve sensor accuracy. Sensor has 30 impulses per rotation and this causes uncertainty of 12°. By introducing a gear ratio equal to 3, we triple the number of pulses per screw shaft rotation. This changes the uncertainty to 4°.

5.2. Force Sensor

Force sensor is fixed by screws on to the plate and its purpose is to measure the force applied to the spring.

5.3. LSU

Linear stage unit is used to deflect springs with good precision. It acts as a basis in our device to which all the components are mounted. Additionally it enables to maintain the deflection of the spring without additional mechanism.

5.4. Mechanism

The idea behind the design is that by turning the threaded shaft the gears are turned with it encoding the rotated angle on the encoder that multiples it by constant (around 0.00185 mm/degree of turn). In this way we obtain the length by which spring is deflected. When the threaded shaft is turned the table is moved and with it the Big Plate causing the offset of the force sensor to which the spring is attached to. This enables the deflection force to be measured. Deflection distance and force information is then sent to the esp controller where it can be processed to provide the spring constant.

6. Electrical Implementation

6.1. Encoder

KY-040 model, Connecting the Digital Data pins to the ESP and using the CLK to generate the timing pulses to determine the direction of rotation of position for further detail look at the code in the esp8266.

KY-040

Parameter	**Specification**	**Unit**
General		
Type	Mechanical Incremental Rotary Encoder	-
Resolution	20 pulses per revolution (PPR)	-
Operating Voltage	5V (3.3V compatible)	V
Current Consumption	<10	mA
Mechanical		
Shaft Type	6mm D-shaped with push button	-
Rotation Life	100,000	cycles
Switch Life	50,000	cycles
Electrical		
Output Type	Quadrature (A/B phase) + Switch	-
Output Signal	Digital (TTL compatible)	-
Debounce Circuit	Included (RC filter)	-
Physical		
Module Dimensions	30 × 19 × 30	mm
Shaft Length	15	mm
Performance		
Maximum RPM	100	RPM
Contact Resistance	<50	m
Environmental		
Operating Temp	-25 to +70	°C

Pin Configuration:

1. **CLK** (A phase output)
2. **DT** (B phase output)
3. **SW** (Push button switch)
4. **VCC** (3.3-5V power supply)
5. **GND** (Ground)

6.2. Force Sensor

Load cell NA27 is connected to the HX711 load cell amplifier. The amplifier is connected to the ESP8266 and the code reads the output of the amplifier and determines the force on the sensor. The HX711 interfaces the ESP8266 through the HX711 ADC library. Ensure that the pin that supplies the voltage for the force sensor is the Vin pin since this is the only one that can power 5 Volts. (Force Sensor takes in between 5-12V for proper functionality)

HX711

Parameter	**Specification**	**Unit**
General		
ADC Type	24-bit Sigma-Delta	-
Sampling Rate	10 or 80 (selectable)	SPS
Supply Voltage	2.6 - 5.5	V
Operating Current	< 1.5	mA
Input		
Differential Input	±20mV (PGA=128), ±80mV (PGA=64)	mV
PGA Gain	128 (default), 64 (selectable)	-
Output		
Interface	Serial (2-wire)	-
Data Rate	10Hz (80Hz noise) or 80Hz (10Hz noise)	Hz
Load Cell Support		
Excitation Voltage	5	V
Recommended Load Cells	1kg, 5kg, 10kg, 20kg	-
Physical		
Dimensions	34 × 17	mm
Pin Count	16	-
Performance		
Resolution	1 in 16,777,216	counts
Noise (PGA=128)	0̃.5	V RMS
Non-linearity	±0.01	% FS
Temperature		
Operating Range	-40 to +85	°C

NA27	
Rated Capacities	0.61235610kg
Rated Output	1.0 ±0.15mV/V
Zero Balance	±0.1000 mV/V
Non-linearity	0.03 %R.O.
Hysteresis	0.03 %R.O.
Repeatability	0.03 %R.O.
30mins Creep	0.03 %R.O.
30mins Return	0.03 %R.O.
Safe Overload	150 %R.O.
Ultimate Overload	200 %R.O.
Temperature Effect On Output	0.01 %R.O./°C
Temperature Effect On Zero	0.05 %R.O./°C
Input Impedance	1115±10%
Output Impedance	1000±10%
Insulation Impedance	2000 M/(50VDC)
Recommended Excitation	5~12 VDC
Maximum Excitation	15 VDC
OperatingTemperatureRange	-20~60 °C
Construction	Aluminum Alloy
Protection Class	IP65
Cable	0.8×0.25m
Recommended Platform Size	150×150 mm
Mode of Connection	RedEXC+,BlackEXC-,GreenSIG+,WhiteSIG-

7. Software Implementation

7.1. Serial COM Protocol

A protocol specifically programmed for this device, the Application can send requests or data over the serial port to the device, and the device can send requests or data over the serial port to the Application. The table features how the Application and the ESP communicate with each other.

Command	Description	Expected Response
ONN	Turn on calibration, LED turns on	-
OFF	Turn off calibration, LED turns off	-
WGH	Get the force signal	Force value
POS	Get the position	Position value
IMP	Get the impulse counter	Impulse counter value
COM	Test COM port connection	"OK"
RST	Reset the position	"SUC"
TRE	Tare the scale	"SUC"
CAL	Calibrate the weight	-
TREF	Tare fast during measurement	-
ERR	Error value received when request is invalid	Error message
CALVAL	Change the calibration value in the ESP program	-
APUL	Application pulls from microcontroller EEPROM	EEPROM data
MPUL	Microcontroller gets value from application	Value from application

7.2. ESP 8266

```
#include<HX711_ADC.h>
#include <EEPROM.h>

//global variables for position encoder
const int pinA = 5;// Connected to CLK on KY040 D1
const int pinB = 4; // Connected to DT on KY040 D2
//global variables for the force sensor
const int pinSCK = 14;//D5
const int pinDT = 12;//D6
//global variables for position encoder functionality
const double impulses = 30;
const double distance_per_rotation = 3.2;//change after measurement
const double gear_ratio = 4;
int encoderPosCount = 0;
int pinALast;
int aVal;
boolean bCW;
const double distance_per_impulse = (1/impulses)*(distance_per_rotation)/gear_ratio;
float plat_pos = 0;
//global variables for force sensor
HX711_ADC LoadCell(pinDT,pinSCK);
boolean _tare = true;
unsigned long t =0;
float calibrationValue =100;
unsigned long stabilizingtime = 3000;//increasing the accuracy of the tare operation by in
float weight_value = 0;
const int samples = 10;//sampling during filtering and averaging for returning the signal
//global varaibles for communication protocol
String com_data;
// for the EEPROM
double calibrationmap10X5 = 1;   //Variable to store data read from EEPROM.
int eepromAddress = 0; //address in EEPROM to read and write
int EEPROM_SIZE = 512;//EERPROM SIZE
void encoder_position_reader();//function to read the position of the encoder
void com_functions();//function for the communication protocol
void HX711_Loop();//Function for force sensor loop


void setup()
{
pinMode (pinA,INPUT);//Defining pin modes
pinMode (pinB,INPUT);
```

```
pinMode(LED_BUILTIN, OUTPUT);
pinALast = digitalRead(pinA);
Serial.begin (9600);//Serial begin the Baud rate
delay(10);
LoadCell.begin();//Starting the load cell
LoadCell.start(stabilizingtime, _tare);
LoadCell.setSamplesInUse(samples);
// EEPROM calibration storage
EEPROM.begin(EEPROM_SIZE);
}
void loop()
{
  com_functions();//Check for communication between application and ESP in serial and respond
  encoder_position_reader();//Read encoder position
  HX711_Loop();//Read Force
}
void HX711_Loop()
{
  LoadCell.update();
  weight_value =  LoadCell.getData();
}
//Helper Functions
void com_functions()
{
    if(Serial.available())
   {
    com_data = Serial.readStringUntil('\n');
    if(com_data=="ONN")
    {
      digitalWrite(LED_BUILTIN,LOW);
    }
    else if(com_data == "OFF")
    {
      digitalWrite(LED_BUILTIN, HIGH);
    }
    else if(com_data == "COM")
    {
      Serial.write("OK\n");
    }
    else if(com_data == "XRC")
    {

    }
    else if(com_data == "POS") // function to get the position of the platform
```

```
{
  String myString = String(plat_pos);
  //Serial.println(plat_pos);
  char* buf1 = (char*) malloc(sizeof(char)*myString.length()+1);
  myString.toCharArray(buf1, myString.length()+1);
  Serial.write(buf1);
  free(buf1);
  Serial.write('\n');
}
else if(com_data == "IMP")
{
  String myString = String(encoderPosCount);
  char* buf2 = (char*) malloc(sizeof(char)*myString.length()+1);
  myString.toCharArray(buf2, myString.length()+1);
  Serial.write(buf2);
  free(buf2);
  Serial.write('\n');
}
else if(com_data == "RST")//resets the position on the encoder to 0
{
  plat_pos = 0;
  encoderPosCount = 0;
  Serial.write("SUC");
  Serial.write('\n');
}
else if(com_data == "CALVAL")//pulls the calibration constant from the gui and sets it
{
    Serial.write("CALRQS\n");//implement in c# part
    //Serial.write('\n');
    String calibration_value_string = Serial.readStringUntil('\n');
    float calibration_value = calibration_value_string.toFloat();
    calibrationValue = calibration_value;
}
else if(com_data == "TRE")//more accuarate slower tare
{
  LoadCell.tare();
  Serial.write("SUC\n");
}
else if(com_data == "CAL")//preforms the calibration operation
{
  LoadCell.setCalFactor(calibrationValue);
  Serial.write("SUC");
  Serial.write('\n');
}
```

```
    else if (com_data == "TREF")//tare fast
    {
      LoadCell.tareNoDelay();
    }
    else if(com_data == "WGH")//send the weight
    {
      String myString = String(weight_value);
      //Serial.println(weight_value);
      char* buf3 = (char*) malloc(sizeof(char)*myString.length()+1);
      myString.toCharArray(buf3, myString.length()+1);
      Serial.write(buf3);
      free(buf3);
      Serial.write('\n');
    }
    else if(com_data == "MPUL")//Micro-controller pulls data from application application dat
    {
      //send request to application
      Serial.write("MAPREQ");//send calibrationmap request to application
      Serial.write('\n');
      //receive value
      String calibration_value_string = Serial.readStringUntil('\n');
      calibrationmap10X5 = calibration_value_string.toDouble();
      //write value to memory
      EEPROM.put(eepromAddress,calibrationmap10X5);
      EEPROM.commit();
    }
    else if(com_data == "APUL")//Application pulls data from microcontroller Micro-controller
    {
      double calmap;//write to eeprom when power is on but when off will chnage vlaue to 1671
      EEPROM.get(eepromAddress,calmap);
      String myString = String(calmap);
      char* buf = (char*) malloc(sizeof(char)*myString.length()+1);
      myString.toCharArray(buf, myString.length()+1);
      Serial.write(buf);
      free(buf);
      Serial.write('\n');
    }
    else
    {
      Serial.write("ERR\n");
    }
   }
}
void encoder_position_reader()
```

```
{
  aVal = digitalRead(pinA);
if (aVal != pinALast )
{
if (digitalRead(pinB) != aVal)
  {
    encoderPosCount ++;
    bCW = true;
  }
  else
  {
  bCW = false;
  encoderPosCount--;
  }
  if (bCW){}
  else{}
  plat_pos = distance_per_impulse*encoderPosCount;
  }
  pinALast = aVal;
}
```

Above is the code that is flashed to the ESP8266
It contains

1. Setup of the pins
2. Setup of the HX711 Force reading
3. Variables to keep track of the data
4. Encoder position reading
5. Communication Protocol Implementation

7.3. GUI

7.4. GUI Backend

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Text;
using System.Linq;
using System.Reflection;
using System.Security.Cryptography.X509Certificates;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;



namespace Spring_o_meter
{

    public partial class Form1 : Form
    {
        //initializing values for calibration
        private double signal_zero_state_calibration = 0;
        private double signal_test_weight_calibration = 0;
        private double to_force_map = 0;
        Boolean calibrated = false;


      //initalizing values for reading
      //idea of using structs for this purpose
        public struct  data_trial
        {

            public data_trial(double pos, double force)
            {
                Pos = pos;
                Force = force;
            }
            public double Pos { get; set; }
            public double Force { get; set; }
        }
        data_trial current_data = new data_trial();//made null just in case
        data_trial prev_data = new data_trial();
        Stack<data_trial> data_trial_stack = new Stack<data_trial>();// will store the data in stack format
        private double spring_k = 0;
        private double mean_k = 0;
        private double sum = 0;
        // for position component
        private const double GEAR_RATIO = 3;//between driver and gear
        private const double DISTANCE_PER_ROTATION = 2; // distance in mm of distance per rotation
        private  const double IMPULSES_PER_ROTATION = 30;// impulses in the encoder
        //for the graph
        private const int k_stack_reading_depth = 5;// stack reading depth for graphing trials
        private Stack<double> stack_k = new Stack<double>(); // stack that stores the spring constants

        private int sample_count = 0;
        //for the port
        private string port = "NULL";
        Boolean com_connected = false;
        private Boolean kickpoint = false;
        private Boolean kickpoint_found = false;
        private double kickpoint_value = 0.15;
        private double sigweight;
       private double Impulse_to_position(double encoder_count)

        {
            return (1 / IMPULSES_PER_ROTATION) * (DISTANCE_PER_ROTATION / GEAR_RATIO) * encoder_count;
        }
        void reset()// implement rest function
        {
            Get_STM("STM");
        }
        private void Force_Calibration()
        {
            if (calibration_Toggle.Text == "ON" && com_connected)
```

```
    {
        serialPort1.WriteLine("WGH");
        signal_test_weight_calibration = Convert.ToDouble(serialPort1.ReadLine());
        double sigdif = signal_test_weight_calibration - signal_zero_state_calibration;
        if (double.TryParse(textBox1.Text, out double testweight))//checks if is a double
        {
            sigweight = testweight / (sigdif);
            to_force_map = sigweight * 9.806 / 1000;//now in newtons, gives newtons per sig unit
            enter_Weight_Button.Text = "Calibrated";
        }
        else
        {
            to_force_map = -1;
            MessageBox.Show("please enter a valid value", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
}
void Graph_K(Chart chartx, String series_name,Stack<data_trial> stack)
{
    data_trial[] data_trial_array = stack.ToArray();
    SortedDictionary<double,double> keyValuePairs = new SortedDictionary<double,double>();
    for (int i =0; i < data_trial_array.Length;i++)
    {
        double key = Math.Round(Math.Abs(data_trial_array[i].Pos),2);
        double value = Math.Round(Math.Abs(data_trial_array[i].Force),2);
        keyValuePairs.Add(key, value);

    }
    foreach(var j in  keyValuePairs)
    {
        chartx.Series[series_name].Points.AddXY(j.Key, j.Value);
    }
}
private double Signal_To_Force(double Signal)
{
    return Signal*to_force_map;
}
private String Get_STM(String code)
{
    if (com_connected)
    {
        serialPort1.WriteLine(code);
        String stm_output = serialPort1.ReadLine();
        return stm_output;
    }
    else
    {
        MessageBox.Show("COM is not activated", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return "ERR";
    }

}

private double Get_Position()
{
    String imp = (Get_STM("IMP"));
    int current_impulse_count = Int32.Parse(imp);
    double current_position = (Impulse_to_position(current_impulse_count));
    return Math.Round(current_position,5);
}
private double Get_Force()
{
    String output = Get_STM("WGH");
   double cur_sig =  Convert.ToDouble(output);
    double current_force = Math.Round(Signal_To_Force(cur_sig), 5);
    return current_force;
}
private void start_COM(String port_method)
{
    serialPort1.PortName = port_method; //make sure to get the right COM
    serialPort1.Open();//opening the serial port
}
private Boolean Check_COM()
{
    if(serialPort1.IsOpen)
    {
        serialPort1.Close();
    }
    try
    {
        start_COM(port);
        serialPort1.Write("COM");
        //System.Threading.Thread.Sleep(200);
        string com_status = serialPort1.ReadLine();
```

```
        if (com_status == "OK")
        {
            MessageBox.Show("Serial Connected successfully", "PORT NOTIFICATION", MessageBoxButtons.OK, MessageBoxIcon.Information);
            com_connected = true;
            return true;
        }
        else
        {
            MessageBox.Show("Correct the COM#", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return false;
        }
    }
    catch
    {
        MessageBox.Show("Correct the COM# big boy error", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return false;
    }
}
private String COM_dialog()
{
    Form2 port_dialog = new Form2();
    port_dialog.ShowDialog();
    while (port_dialog.finished == false) { }
    port = port_dialog.port;
    return port;
}
private void Graph_Stack(Chart chartx,int stack_reading_depth, Stack<double> stack, String series_name) //  fix this function
{

    Double[] stack_array = stack.ToArray();
    double max = -1;
    if (chart1.Series.Count != 0)
    {
        chart1.Series.Clear();
        chart1.Series.Add(series_name);

    }

    for (int i = 0; i < stack_reading_depth; i++)
    {
        if (i<stack_array.Length)
        {
            double sample_spring_constant = stack_array[i];

            if (max < sample_spring_constant) max = sample_spring_constant;

            chart1.Series[series_name].Points.AddXY(i, sample_spring_constant);
            chart1.ChartAreas[0].AxisY.Maximum = max + 0.1 * max;
            chart1.ChartAreas[0].AxisY.Minimum = 0;
        }

        else
        {
            chart1.Series[series_name].Points.AddXY(i, 0);
        }
    }
}
static double standardDeviation(IEnumerable<double> sequence)
{
    double result = 0;
    if (sequence.Any())
    {
        double average = sequence.Average();
        double sum = sequence.Sum(d => Math.Pow(d - average, 2));
        result = Math.Sqrt((sum) / sequence.Count());
    }
    return result;
}
static double Mean(IEnumerable<double> sequence)
{
    if (sequence.Any())
    {
        double average = sequence.Average();
        return average;
    }
    return -1;
}

public Form1()
{
    InitializeComponent();
    timer1.Start();
}

private void Form1_Load(object sender, EventArgs e)//upon loading the form
```

```
{

}
private void button1_Click(object sender, EventArgs e)// toggling the
{

    try
    {
        if (calibration_Toggle.Text == "OFF")
        {
            calibration_Toggle.Text = "ON";
            calibration_Toggle.BackColor = Color.Green;
            serialPort1.Write("ONN\n");//turns led on stm on
            serialPort1.WriteLine("TRE");
            if (serialPort1.ReadLine() == "SUC") { }
            else
            {
                MessageBox.Show("STMERROR", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
        else
        {
            serialPort1.WriteLine("TRE");
            calibration_Toggle.Text = "OFF";
            calibration_Toggle.BackColor = Color.Red;
            serialPort1.Write("OFF\n");
        }
    }
    catch
    {
        MessageBox.Show("PORT is not configured", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
private void button1_Click_1(object sender, EventArgs e)//complete calibration
{
    if (calibration_Toggle.Text == "ON" )
    {
        Force_Calibration();
        String response = Get_STM("MPUL");
        if (response == "MAPREQ")
        {
            String to_force_map_string = (to_force_map * 100000).ToString();
            serialPort1.WriteLine(to_force_map_string);

        }
        calibrated = true;
    }
    else if(calibration_Toggle.Text == "OFF")
    {
        MessageBox.Show("Turn On calibration", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
private void button2_Click(object sender, EventArgs e)//calibrating the self_weight;
{

    if (calibration_Toggle.Text == "ON" && !calibrated)
    {
        serialPort1.WriteLine("CAL");
        String output = Get_STM("WGH");
        signal_zero_state_calibration =  Convert.ToDouble(serialPort1.ReadLine());
        button2.Text = "Calibrated";
        label23.Text = "LOAD TEST";
        label23.BackColor = Color.Green;
    }
    else if(calibrated)
    {
        MessageBox.Show("Restart to calibrate", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else
    {
        MessageBox.Show("Turn On calibration", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
private void button1_Click_2(object sender, EventArgs e)//sampling button add to chart also
{
    if (button4.BackColor != Color.Green)
    {
        MessageBox.Show("calibrate", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
    prev_data = current_data;
    sample_count++;
    current_data = new data_trial(Get_Position(), Get_Force());
    data_trial_stack.Push(current_data);
    label12.Text = current_data.Pos.ToString();
```

```
    label13.Text = current_data.Force.ToString();
    label14.Text = prev_data.Pos.ToString();
    label15.Text = prev_data.Force.ToString();
    if (prev_data.Pos == current_data.Pos)
    {
        MessageBox.Show("change position to attain measurement", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
    if (sample_count >= 2)
    {
        spring_k = Math.Round(Math.Abs((current_data.Force - prev_data.Force) / (current_data.Pos - prev_data.Pos)), 3);
        sum += spring_k;
        mean_k = Mean(stack_k);
        label17.Text = spring_k.ToString();
        stack_k.Push(spring_k);
        try
        {
            Graph_Stack(chart1, k_stack_reading_depth, stack_k, "K");// to graph the whole thing by displaying
        }
        catch
        {
            stack_k.Pop();
            stack_k.Push(0);
            MessageBox.Show("Graph error", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
    //last 5 measurements from the stack in the form of a line graph
}
private void button3_Click(object sender, EventArgs e)
{
    String port_output  = COM_dialog();
    label4.Text = port_output;
}
private void button4_Click(object sender, EventArgs e)
{
    if(Check_COM())
    {
        button4.BackColor = Color.Green;
    }
    else button4.BackColor = Color.Red;
}
private void timer1_Tick(object sender, EventArgs e)
{
   if (com_connected&&calibrated)
     {
        double force = Get_Force();
        double position = Get_Position();
        label20.Text = Math.Round(position,5).ToString()+" mm";
        label22.Text = Math.Round(force, 5).ToString()+" N";
        if(!kickpoint_found && kickpoint)
        {
            if(Math.Abs(force) >=kickpoint_value)
            {
                kickpoint_found = true;
                label30.Text = Math.Round(position, 5).ToString() + " mm";
            }
        }
    }
}
private void button5_Click(object sender, EventArgs e)
{
    serialPort1.WriteLine("TRE");
    if (serialPort1.ReadLine() == "SUC") { }
    else
    {
        MessageBox.Show("STMERROR", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
private void button6_Click(object sender, EventArgs e)
{
    if (Get_STM("RST") == "SUC") { }
    else
    {
        MessageBox.Show("STMERROR", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
private void button7_Click(object sender, EventArgs e)
{
    Graph_K(chart2,"KC", data_trial_stack);
    label25.Text = Math.Abs(Math.Round(sum/(sample_count-1),3)).ToString();
    double stand_div = standardDeviation(stack_k.ToArray());
    label27.Text= Math.Round(stand_div,5).ToString();
}

private void button8_Click(object sender, EventArgs e)
```

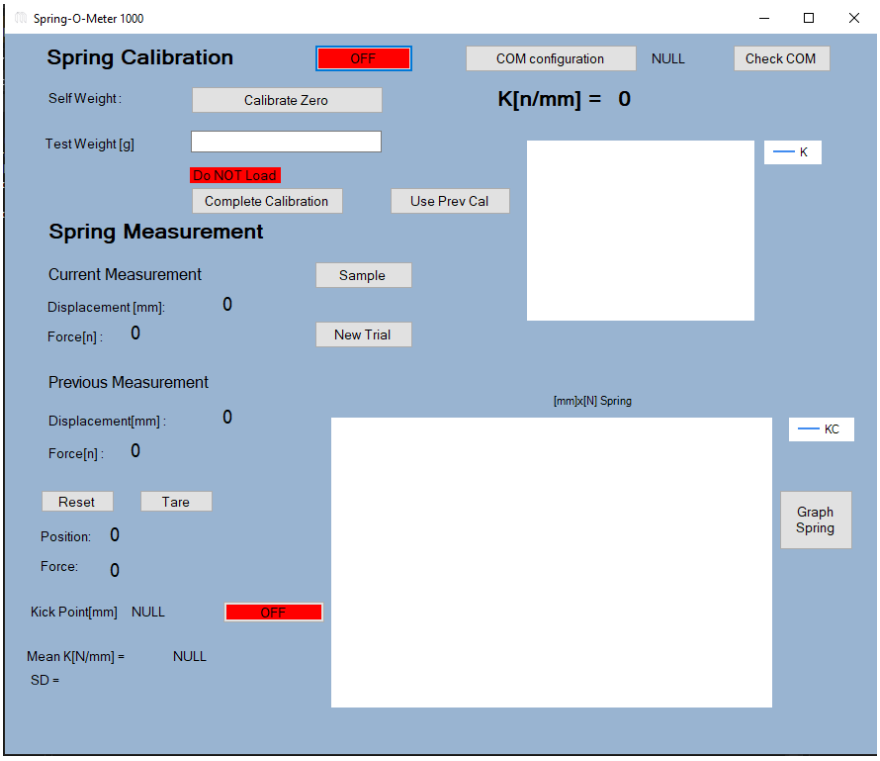
```
        {
            if(button8.Text == "OFF")
            {
                button8.Text = "ON";
                button8.BackColor = Color.Green;
                kickpoint = true;
            }
            else
            {
                button8.Text = "OFF";
                button8.BackColor = Color.Red;
                kickpoint = false;
            }
        }
        private void button9_Click(object sender, EventArgs e)//new trial . resets all values
        {
            stack_k.Clear();
            data_trial_stack.Clear();
            sum = 0;
            sample_count = 0;
            label12.Text = "0";
            label13.Text = "0";
            label14.Text = "0";
            label15.Text = "0";
            label25.Text = "NULL";
            label30.Text = "NULL";
            chart1.Series.Clear();
            chart1.Series.Add("K");
            chart2.Series.Clear();
            chart2.Series.Add("KC");

        }

        private void button10_Click(object sender, EventArgs e)
        {
            String output = Get_STM("APUL");
            double output_normal = Convert.ToDouble(output)/10000000;
            to_force_map = output_normal;
            calibrated = true;
        }
    }
}
```

7.5. GUI Frontend

Here is how the GUI should look like upon launch



8. Test

	K_last	K_mean	SD	
Tension Spring 1	0.206	0.208	0.0074	[N/mm]
Tension Spring 2	0.197	0.195	0.0038	[N/mm]
Compression Spring 1	0.891	0.899	0.0124	[N/mm]
Compression Spring 2	0.446	0.44	0.0088	[N/mm]

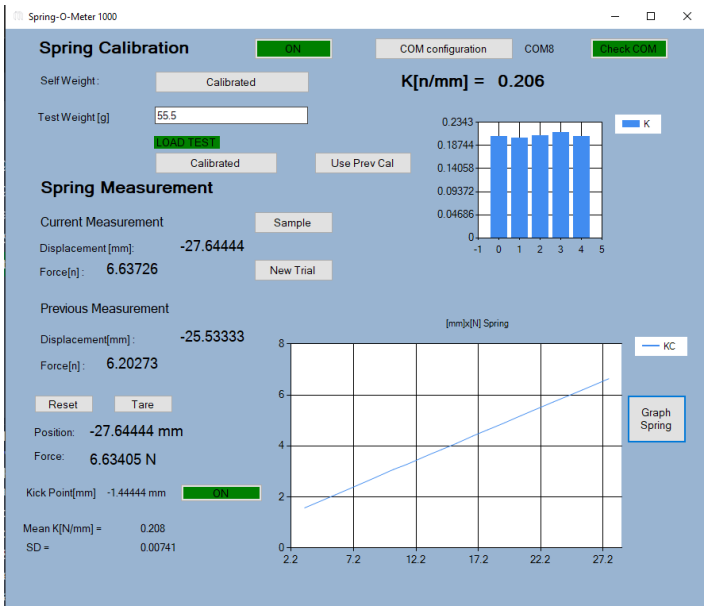


Figure 8.1. Result for Tension Spring 1 (Shorter)

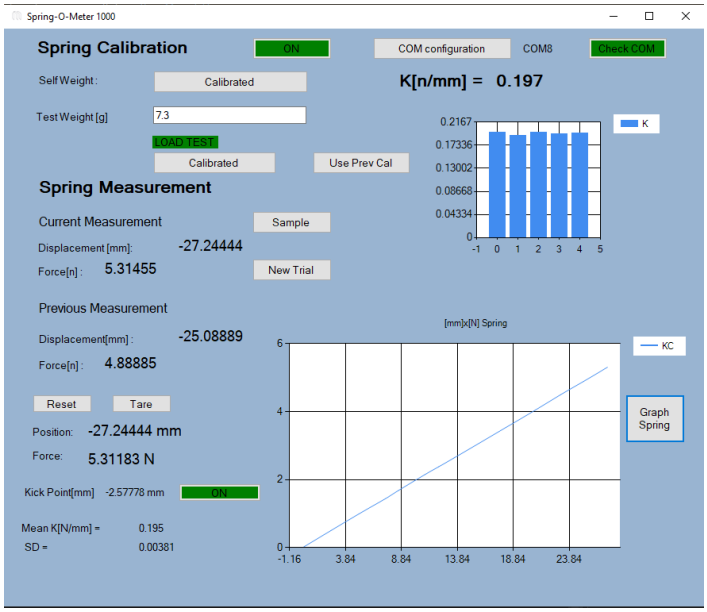


Figure 8.2. Result for Tension Spring 2

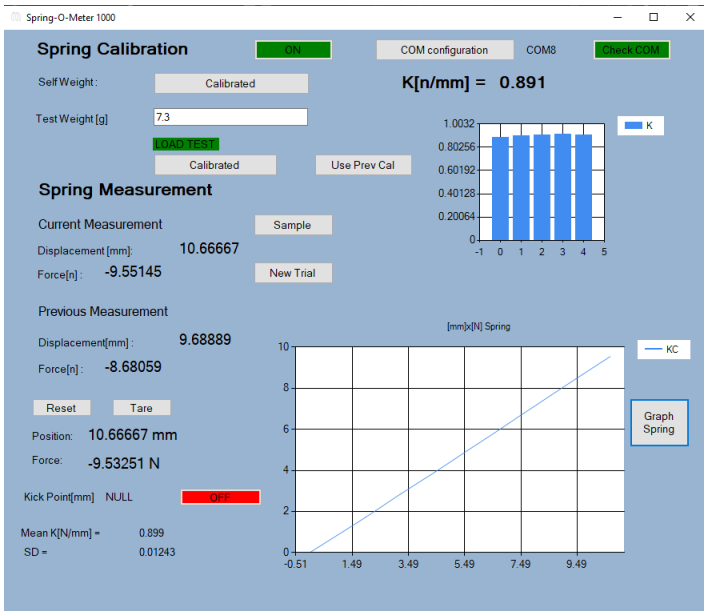


Figure 8.3. Result for Compression Spring 1

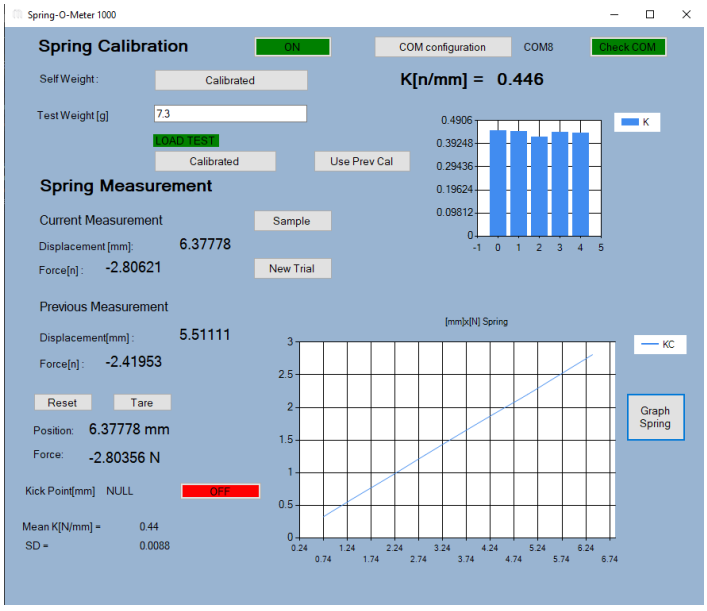


Figure 8.4. Result for Compression Spring 2 (smaller one)