

Sensory Feedback in Prosthetic Arms

Nibras Abir Ahad

2019

Master's Thesis in
Biomedical Engineering

Supervisors: Nebojsa Malesevic, Pamela Svensson



Department of Biomedical Engineering

1.1 Abstract

The sense of touch in our body is something that many of us take for granted. It is after all hard to imagine how we would move and utilize our arms and legs without any sensory feedback.

For people who need to use hand prosthetics it is however very clear how they could improve their quality of life if they could “feel” with their hands once again. This could be realized using an artificial sense of touch, a sensory feedback system.

There are different ways to accomplish the above. A simple solution would be to pair force sensors on the hand prosthesis with a device which provides mechanical feedback to the user. Such an approach is investigated in this report.

The initial step was to build a prototype device which would allow for controlling a DC motor with force input from the user. This system was then used as a test bench to compare results from using two different sensor types, the FSR and SingleTact sensors.

With the main findings from this report it can be concluded that a feedback system using both sensor types can work. While the error was reduced to some extent using PI control, further work will be necessary until the system can be used in practice.

1.2 Sammanfattning

Det är många som tar för givet kroppens förmåga att kunna känna. Det är trots allt svårt att kunna föreställa oss hur vi skulle kunna röra och använda våra armar och ben utan någon som helst återkoppling.

För människor som behöver använda handproteser så är det väldigt tydligt att de skulle kunna förbättra kvalitén på vardagen och livet om de skulle kunna "känna" med deras proteser. Detta är något som kan realiseras med artificiell känselåterkoppling.

Det finns olika sätt att åstadkomma ovanstående. Ett simpelt sätt är att para ihop en handprotes som använder trycksensorer med en enhet som utför mekanisk återkoppling till användaren. En sådan metod undersöks i denna rapport.

Det första steget var att bygga ihop en prototyp som möjliggjorde för styrning av en DC motor med manuell inmatning av kraft värde från användaren. Detta användes sedan som ett testsystem för att mäta och jämföra resultat från två olika sensortyper.

Med de viktigaste fynden från denna rapport så går det att dra slutsatsen att känselåterkoppling med båda typer av sensorer är möjligt. Medans det gick att reducera felvärden med hjälp av PI regulator till en viss grad, så finns det en del arbete kvar tills systemet får tillräckligt bra prestanda för att praktisk tillämpning kan ske.

1.3 Acknowledgments

This thesis was completed during autumn 2019 at the department of Biomedical Engineering at Lund University.

This project has been very exciting yet challenging, and finishing it would not be possible without support. So I would like to thank all involved people who have helped with this project.

I would like to thank Christian Antfolk for help with setting up the thesis and offering helpful advice with sensors and equipment. And thank Pamela Svensson for providing help by 3D printing material used to attach sensors for this project as well as assistance with lab equipment when needed.

And last but not least, a very special thanks to Nebojsa Malesevic who has offered tremendous help with support and guidance throughout the project, specially towards the end.

Table of Contents

Sensory Feedback in	1
Prosthetic Arms	1
1.1 Abstract.....	3
1.2 Sammanfattning	4
1.3 Acknowledgments	5
Table of Contents.....	6
1.4 List of abbreviations	10
2 Introduction & Theory.....	11
2.1 Background	11
2.1.1 Amputation.....	11
2.1.2 Transplantation	11
2.1.3 Prosthetics	11
2.1.4 Types of Prosthetic Arms	12
2.1.5 Sensory Feedback System	14
2.2 Objectives and Scope.....	15
2.3 Components and Equipment	16
2.3.1 Force Sensitive Resistor (FSR).....	16
2.3.2 Capacitive Sensors	18
2.3.3 Arduino Microcontroller	20
2.3.4 Micro Linear Actuator	20
2.3.5 Linear Actuator Control Board.....	21
2.4 Motor Control.....	21
2.4.1 Proportional control	21
2.4.2 PID control	22
2.5 Measurements and Graphs	25
2.5.1 Box Plots	25
3 Method	26
3.1 Electronic circuit	26

3.1.1	Arduino Board	26
3.1.2	Sensor attachment	27
3.1.3	FSR sensor.....	27
3.1.4	SingleTact sensor.....	28
3.1.5	Motor and LAC board configuration	29
3.2	Mounting and setup	30
3.2.1	Mounting	30
3.2.2	Target plate.....	31
3.3	Measurement and System overview	32
3.3.1	Measurement, Rise Time and RMS Error	32
3.3.2	System and data transfer overview	34
3.3.3	Software Flow Chart	35
3.3.4	Graphical User Interface	36
4	Results	38
4.1	FSR Measurements with Mark 10 Force Gauge	39
4.2	Proportional control.....	40
4.2.1	Proportional control - Rise time - FSR	41
4.2.2	Proportional control - RMSE.....	42
4.3	Integrative and Derivative Control	43
4.3.1	Integrative control.....	44
4.3.2	Derivative control	45
5	Discussion.....	46
5.1	Measurements and PID Parameters	46
5.1.1	RMS Error	46
5.1.2	Comparison of FSR and SingleTact sensors	46
5.2	Sources of error	46
5.2.1	SingleTact offset error	46
5.2.2	FSR and calibration accuracy	47
5.2.3	Motor and Encoder limitation.....	47
5.2.4	Human factor	47

6	Conclusion.....	48
6.1	Final system	48
6.2	Further Work.....	48
6.2.1	Other sensor types, Strain Gauge.....	48
6.2.2	System identification	49
6.2.3	Automated test bench.....	49
	Appendix	50
6.3	Appendix - FSR measurements: Box Plots	50
6.3.1	Proportional control RMSE error: FSR.....	50
6.3.2	Proportional control Rise time: FSR	52
6.3.3	Integrative Control RMSE: FSR.....	54
6.3.4	Integrative control Rise time: FSR	56
6.3.5	Derivative control RMS Error: FSR	58
6.3.6	Derivative control Rise Time: FSR.....	60
6.4	Appendix -SingleTact sensor measurements	62
6.4.1	Proportional control RMSE: SingleTact	62
6.4.2	Proportional control Rise time: SingleTact	64
6.4.3	Integrative control RMSE: SingleTact.....	66
6.4.4	Integrative control Rise time: SingleTact.....	68
6.4.5	Derivative control RMSE: SingleTact	71
6.4.6	Derivative control Rise time: SingleTact	73
6.5	Appendix - Arduino Code.....	75
6.6	Appendix - Java Code.....	82
6.6.1	ReaderLine Class	82
6.6.2	Main Class.....	89
6.7	Appendix – FSR Calibration Data.....	99
7	References	101

1.4 List of abbreviations

- FSR – Force Sensitive Resistor
- SingleTact – SingleTact Capacitive Force sensor
- Linear actuator – a longitudinal, DC motor from Actonix Inc
- LAC board – Linear Actuator Control board from Auctonic Inc
- Kp value – Proportional constant term of PID controller
- Ki value – Integrative constant term of PID controller
- Kd value – Derivative constant term of PID controller
- GUI – Graphical user interface
- I/O – Input and Output
- RMS value – Root Mean Square value
- RT – Rise Time
- IQR – Interquartile Range
- Set point – In Control Theory: desired output value
- PNS – Peripheral Nervous System

2 Introduction & Theory

2.1 Background

The importance of the sense of touch in our hands cannot be understated. It is essential for supporting the mechanical functionality of our hands and arms. Attempting to hold a cup using a common prosthetic arm which does not have any sensory features, turns an otherwise trivial action into a daily challenge.

2.1.1 *Amputation*

Amputations can be performed for a variety of reasons some of which can be due to an accident (a traumatic injury) or due to an infection or disease, if there is a risk of it spreading to other parts of the body.

Most amputees, estimated to over 80%, has experienced the phantom limb syndrome. This sensation can be described as awareness of a non-existent part of the body. For some these can be sensations of movement in their non-existent limbs while others may experience pain. [1] [2]

2.1.2 *Transplantation*

The ideal solution for patients with amputations, would be to replace the hand/arm (or other limb) using transplantation. This approach could be the best in terms of physical functionality since the replacement limb would be similar to the original. The level of restoration of the functionality would depend on the type and extent of amputation for each patient. There are many drawbacks however: first there is the difficulty of finding a suitable hand/arm to transplant, and also the cost and availability of surgeons to perform the transplantation. Furthermore, there is the possibility that the immune system of the patient can reject the transplant. Hence, transplantation alone cannot be the solution for all patients. [1] [3]

2.1.3 *Prosthetics*

Prosthetics have long been a way to aid people with a missing limb. While originally this was used mainly as way to improve the appearance of the user, in modern prosthetics with more functionality are being researched and developed.

While there are nowadays commercially available prosthetics there are always challenges in the development for better devices. For example, limitations with size, weight and battery life (for electronic prosthetics). Designers and researchers will have to balance the above limitations along with the cost and actual functionality. Actually only around 50-60% of hand amputees wear a prosthetic device, likely because the gained functionality does not offset the discomfort and other previously mentioned issues. There is definitely a lot of room for improvement before the development of an ideal prosthetic can be finalized. [4]

Before looking into the different types of prosthetic arms, it is worth summarizing the functionality of our natural arms which we want the devices to restore.

- **Aesthetics**, the visual appearance of our limbs (and other body parts) affect our feelings of wellbeing and normality
- **Control**, the ability to physically maneuver our arms and fingers
- **Sensory Feedback**, the sense of touch which is not only important on its own, but also a necessity to assist with the above point

2.1.4 Types of Prosthetic Arms



Figure 1. A passive prosthetic arm [5]

Prosthetic arms and hands can be classified into two groups depending on its functionality, as discussed by C. Antfolk [2] in his doctoral thesis:

- Passive prosthetics
- Active prosthetics

The latter can be further divided the second type into subclasses: electric, body operated.

Passive prosthetic

The passive versions do not contain any electronics for providing sensory feedback to the user. Mainly, they are used to mimic the appearance of the lost hand, and is therefore useful from aesthetic point of view, see Fig 1. The main problem with this type of prosthetic arises when the user needs to perform actions which require more complex movements, in which case an “active” device will be required. [6]

Active prosthetic

The active prosthetic can be controlled by using the body or electronically, containing electronic components for added functionality. The main purpose of the added complexity in these active devices is for the physical movement and operation of the device. Example. being able to grip or being able to retract and detract the arm.

The “body powered” version, works by linking the device to another part of the body, for example the shoulder. The Bowden Cable System is an example of this and works through attachment of a flexible cable from the prosthetic to a harness on the shoulder of the user. The user can then operate the device by moving his or her shoulder muscles. A big advantage of this system is the sense of control an amputee can obtain because they are controlling the prosthetic using a part of their own body. Other benefits of the Bowden Cable System, is the low cost and durability. A major drawback is the fact that only a limited range of motion can be obtained with this system, and also there is a requirement to wear a harness which can cause discomfort and may prevent the user from using the prosthetic arm over extended periods. [5] [7] [8]



Figure 2 A motorized prosthetic arm (Shirley Ryan Abilitylab) [9]

Active prosthetics: myoelectric sensors

For a more complex range of motions the best choice is to use electronically controlled (i.e. motorized) prosthetics, as illustrated in Fig 2. The motors can be controlled through myoelectric sensors which in turn can be used to read the electric activity of muscles in a residual limb. [6]

With these devices there is however, still a big challenge due to the lack of response towards the user when moving the arm. In this case, it is therefore very important to create the means of a sensory feedback system to allow the user to naturally operate the device in a similar manner to how they would use their own arm. [6]

2.1.5 Sensory Feedback System

There are different approaches for providing feedback to users of prosthetics, these can mainly be grouped into two categories: invasive and non-invasive methods.

Invasive method

The invasive feedback methods mainly involve using electrodes to send stimuli to nerves in the body, such as the peripheral nerves. This can be done through extra and intra-neural electrodes. The latter use electrodes which penetrates the nerves, and can provide more natural stimuli at the cost of increased chance of nerve damage after inserting the transplant. Another problem is that the implants can interfere with previously mentioned myoelectric sensors used to control the device. [10]

Non-Invasive methods

Various invasive feedback options are mentioned in Antfolk [6] and Svensson et al [10]: some of these are mechanical, for example utilizing pressure and vibration. A non-mechanical feedback system is to use temperature for feedback, however due to long reaction time as well as high energy consumption it has not been considered to be a viable option. In addition, for hand prosthetics, the location of the actuators should be in close relation to the receptors of the hand. This placement can affect the learning curve for the user to understand which stimuli represent which part of the prosthetic is being touched. [6] [10]

Vibrotactile feedback involves the use of devices which provide vibrations. The combination of time duration and strength of the vibrations allows the prosthetic arm where to receive feedback in more detail (compared to just using visual

cues). Some benefits are the small size of the devices as well as low power consumption. On the other hand, the sound produced by the devices may irritate the user. For example, if the user needs to perform a longer action with the prosthetic, then the vibration will be felt throughout the entire duration. [6] [10]

Electrotactile sensory feedback system works by stimulating the skin with low-level electric current. This is done by depolarizing afferents neurons of the skin by sending pulses of electrical current. The stimulation can be altered by modulating the parameters of the pulses, for example pulse-width, intensity etc. Unlike the case with the invasive feedback systems, the electrotactile stimulation device can be placed on top of the skin and be used without interfering with myoelectric electrodes. [6] [11]

Mechano-tactile feedback is a way to provide feedback by mechanically pushing towards the skin. An advantage of this is the possibility to match the strength of the feedback with the pressure measured by sensors on the prosthetic device. As a result of this, the user of the arm will have a decreased cognitive burden as they will be able to feel in detail with how much force they are holding a cup or pushing a door etc. Overall because this system is non-invasive, simple and durable, it has been chosen for this report. See next section for objectives and scope [6] [10]

2.2 Objectives and Scope

The main objective in this project was to study a mechano-tactile feedback, which would work by pushing towards a material with specified force. The physical action of pressing towards the skin was accomplished using a servo motor as the actuator.

The secondary goal was to find adequate sensors and motor for this task. In this project two different force sensors were tested.

While a lot of measurements were taken to compare the different components, the core of this project was to build the system itself, essentially test bench for testing the sensors and motor.

2.3 Components and Equipment

The following electrical components and objects were used in this project:

- Force Sensitive Resistor (FSR), Interlink 402 model.
- Capacitive Force sensor, SingleTact
- Arduino Micro (programmable micro controller)
- Auctonix Linear Actuator (DC motor)
- Auctonix Linear actuator (motor control board)
- Custom made target plate, using plastic foam and metal

The following tools were used:

- El 183 Power Supply (DC voltage generator)
- Bench Vise (to fix motor + target setup)
- 3D printer
- Mark 10 Force Gauge

2.3.1 Force Sensitive Resistor (FSR)

The Force Sensitive Resistor (FSR) sensor (fig 3) is a component which changes its resistivity after force has been applied to its sensing area. It was invented and patented by Franklin Eventoff in 1977. [12]



Figure 3. FSR sensor, Interlink 402 model [43]

It consists mainly of two layers separated by a plastic spacer. As the top area (the sensing film), which could be explained as a grid of conductors, is pressed, it will

push towards the lower conductive film. As the space between the layers is reduced the result is that the active area will become increasingly conductive. [13]

Figure 4. illustrates the different layers of the Interlink 402 FSR. As the force is applied to the surface of the sensor, more of the dots on the active area will touch the conductive film and the sensor will become more conductive, and thus reduced resistance.

A simple way of connecting a FSR sensor is through a voltage divider. This is used to get a voltage measurement as a result of applied force, which is usually more convenient than measuring the FSR resistivity directly. [14]

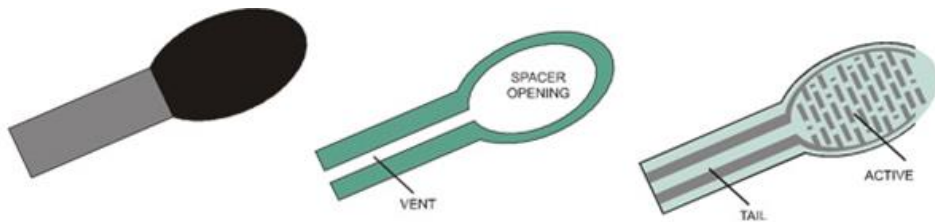


Figure 4. FSR sensor layers from the left: flexible substrate with conductive area, spacer, and flexible active film, [44]

Non-linearity and calibration

The relation between the resistivity and applied force on an FSR sensor is nonlinear, see figure 5.

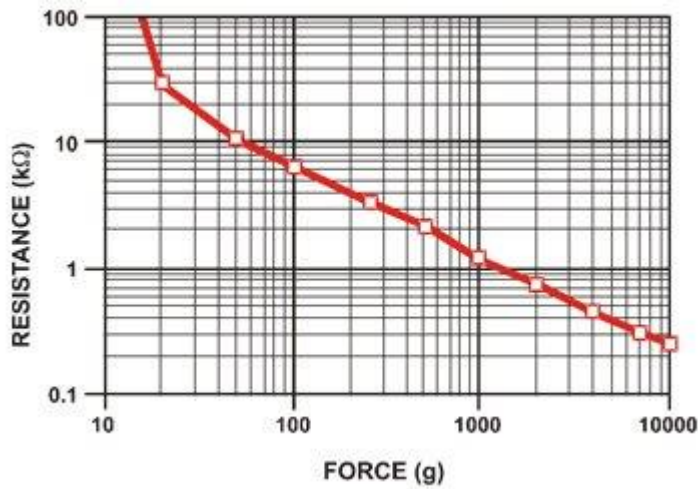


Figure 5 Conductance due to force in an FSR sensor [15] [16]

Because of the nonlinear relation it is therefore a necessity to calibrate the sensor, and the calibration method itself may affect the final accuracy.

Although the sensor does not have ideal accuracy, with a measurement error around 10%, it is thin (less than half a millimeter) and cheap to produce. [12] [15] Besides the advantage of the sensor being low in cost and being thin and small in size makes it very suitable for robotic and medical applications [17]

2.3.2 Capacitive Sensors

The capacitive force sensor works by measuring changes in the capacitance. This can be realized by creating a thin silicon plate attached on a substrate. The silicon plate along the underlying substrate forms two electrodes which acts as a parallel plate capacitor. High sensitivity is a benefit for this type of sensor, however to read the output it is essential to be able to detect very small changes in capacitance. [13] [18] [19]



Figure 6. SingleTact sensor with digital interface board [7]

A commercially made capacitive sensor, the SingleTact (Pressure Profile Systems, Inc.) [20] was used in this project, see figure 6.

The sensor is attached to a digital interface boards (also provided by the manufacturer) to read the sensor output with 16-bit precision, and has an internal capacitance to digital converter (CPC) which converts the capacitance input to a digital force output. [20]

The SingleTact sensor along with the interface board can be configured to communicate digitally with an external microcontroller using the I2C bus, which is how it was set up in this project, see the method section for further details.

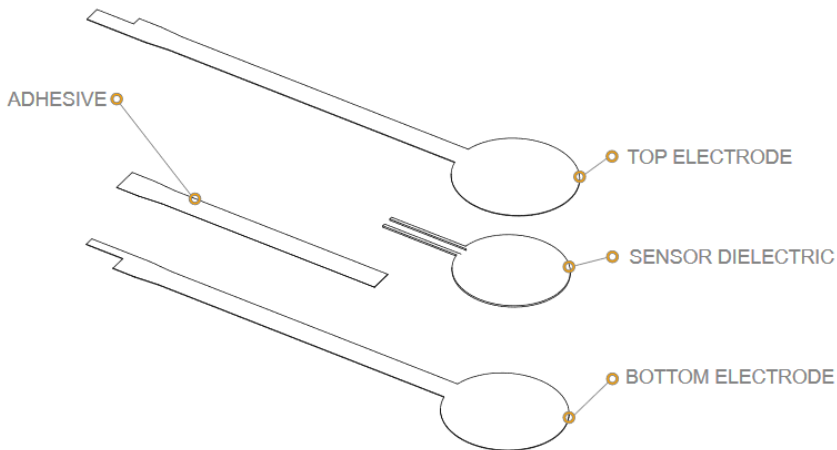


Figure 7 SingleTact Sensor, internal view [21]

The internal layers of the SingleTact sensor are illustrated in figure 7. It shows how the sensor is separated by the top and bottom electrodes by a dielectric layer.

2.3.3 *Arduino Microcontroller*

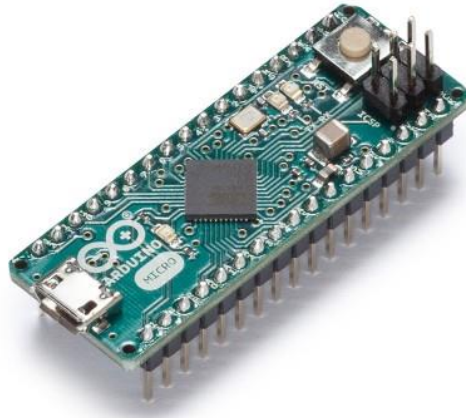


Figure 8. Arduino Micro Board [10]

The project utilized an Arduino micro board (see figure 8) to connect sensors and motor. The board has analog and digital pins which can both provide and read voltage. Being both easy to program and connect to a computer, the Arduino micro was the core of the electronic circuit in this project. Sensor data was sent from the micro controller to the computer, using the Serial interface. The data was then viewable using a graphical user interface, this has been described more in detail in the method section.

2.3.4 *Micro Linear Actuator*



Figure 9. PQ12 Auctonix linear actuator micro [11]

The mechanical actuator used in this project consists of a PQ12 Auctonix linear actuator (see figure 9). This is a servo motor which operates by extending and retracting an arm in a single direction.

Notice that the motor used in this project uses PQ12-30-6-P configuration, meaning that it is rated for 6 Volt DC and uses the potentiometer values from the LAC board. See Method section for connections and pins.

2.3.5 Linear Actuator Control Board



Figure 10. Auctonix LAC [12]

The Auctonix Linear Actuator Control (LAC), is a motor control board connected with external power, it was used as an intermediate link between the Arduino board and motor. This was done since the micro controller itself cannot reliably supply sufficient power by itself to drive the actuator.

The LAC board features an internal encoder to control the linear actuator. It has four rotary-switches for speed, accuracy and min/max positions. Tweaking these using a screwdriver allowed the attached motor to reach selected position at faster speed or with more precision, the tradeoff being stability. This is discussed in further sections.

2.4 Motor Control

2.4.1 Proportional control

As the actuator pushes towards an elastic material, such as the skin, an expected problem is that the control of the motor cannot be static. If the motor is set to push with a predetermined force value, some errors are to be expected. The skin will

have varying elasticity depending on the area of the body, and thus the motor will encounter varying resistance depending on where it is pushing.

Furthermore, we cannot expect the device to handle the same value of resistance when used on individuals with different skin and body characteristics. In essence controlling the motor with predetermined values will not allow us to obtain desired force values since it would only be optimized for one person.

A solution to this problem would be to keep driving the motor with sufficient power until desired force value have been confirmed by the sensor, after which it is turned off. This method called *on/off switching* would have some disadvantages. The problem is that this approach will never allow precise control since the actual force value will always be above or below desired level, constantly moving.

It is evident that there is a need for proportional control where the power of the motor can be adjusted from 0-100%.

2.4.2 *PID control*

To maintain the motor at a desired position with stability, a PID controller was used. The controller works by constantly measuring the output from the process and comparing it with the desired value (the set point). In this case it was used to compare the measured force value in Newton with the desired value. [22]

Furthermore, the difference between the mentioned values (the error) is applied to the output signal being sent to the process, in our case the motor. How the error alters the output signal depends on the proportional, integrative and derivative action of the controller, illustrated in figure 11. The summation symbol in the left hand side of the illustration represents the output being subtracted from the set point, the resulting term is the “error”.

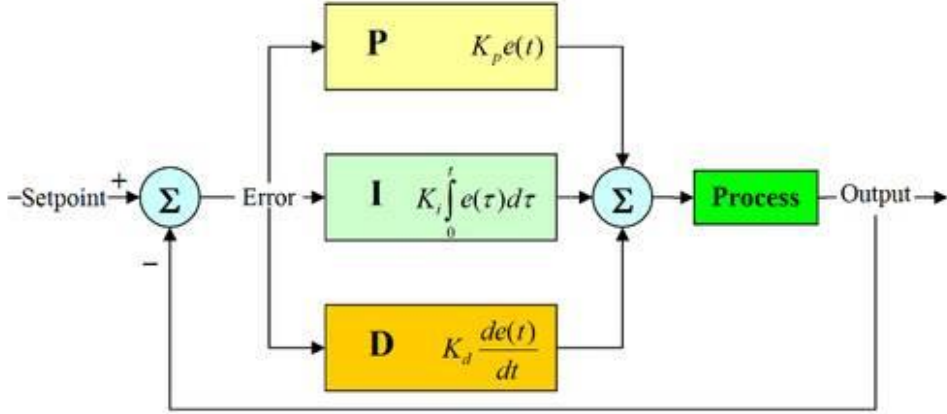


Figure 11 PID controller [23]

The blocks of the controller:

Proportional action, P, is calculated by multiplying the error “e”, with a constant term, K_p .

Integrative action, I, is the sum of the total error measured over a specified time t , multiplied by integral term K_i .

Derivative action, D, is added to the output by derivation of the error and multiplied by differential term K_d .

The PID control output signal can thus be tweaked by changing the parameters K_p , K_i and K_d . The continuous time formula for the output can be seen in equation 1 below. [22] [23]

$$u = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (eq1)$$

The P, I and D terms are added to the output, u , which can also be seen in Figure 11, right hand side, represented by the summation symbol, Σ .

Note that equation 1 shows the PID output formula in continuous time.

Discrete Time controller

Because the measurements are not continuously taken, the Arduino controller takes finite samples per second from the sensors, eq1 has to be adapted for discrete time control so that it can be implemented into the code.

Arduino Code used for the controller:

```
setPoint = targetPressure;
currentTimeRegul = millis();
elapsedTimeRegul = (double) (currentTimeRegul - previousTimeRegul);
error = setPoint - inp;

cumError += error * elapsedTimeRegul;
rateError = (error - lastError) / elapsedTimeRegul;
lastError = error;
previousTimeRegul = currentTimeRegul;

if (ki == 0)
    cumError = 0;

double out = kp * error + ki * cumError + kd * rateError;
```

Description of parameters:

- setPoint: desired value (e.g. force or pressure)
- elapsedTimeRegul: the difference of time between this and previous calculation
- cumError: the total summation of error over time, used by integrative part
- rateError: the rate of change of the error, used by derivative part
- out: the PID output value

The full code can be found at the end of the report: Appendix – Arduino Code.

2.5 Measurements and Graphs

2.5.1 Box Plots

For presenting the measurement data in this project, “box plots” were used. The advantage of using box plots, also called whisker plots, is to obtain an overview when observing the spread of the data points in a set of data, and comparing to another set. [24]

This type of plot is a summary of a set of data presented in 5 parts (+ outliers), see figure 12.

These parts or components of the box plot is calculated in the following way:

Median

First the median is found in the set of data points.

Quartiles

Using the median, the set is split into two parts: values over the median (“upper set”) and values under the median (the “lower set”).

The mean value of the “lower” set of data, is defined as our first quartile. In similar way, the third quartile is found by taking the median of the “upper” set of data. [25]

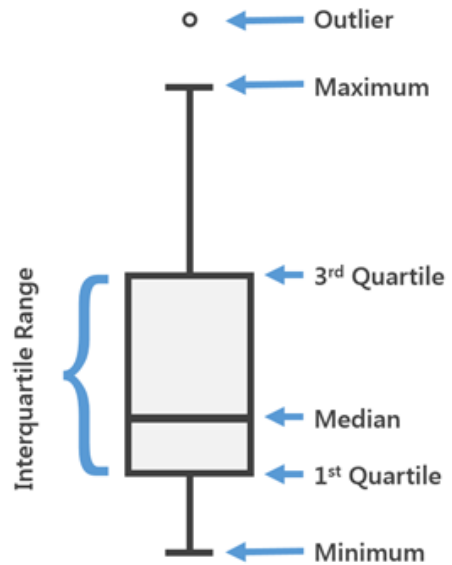


Figure 12 Box Plot

The interquartile range is the difference between both quartiles. [25] [26]

Outlier

An outlier is a data point which is outside a range of 1,5 times the interquartile range, IQR:

The range is 1,5 times IQR above the 3rd quartile and 1,5 IQR below the 1st quartile. All values outside this range is illustrated as an outlier in the box plot, see the small circle of figure 12. [26]

Minimum and maximum values

To complement the above, the smallest and largest values from the set is also presented. This is illustrated in figure 9 as the end point of the “whiskers”, the long lines that stretch up and down from the box.

3 Method

3.1 Electronic circuit

3.1.1 Arduino Board

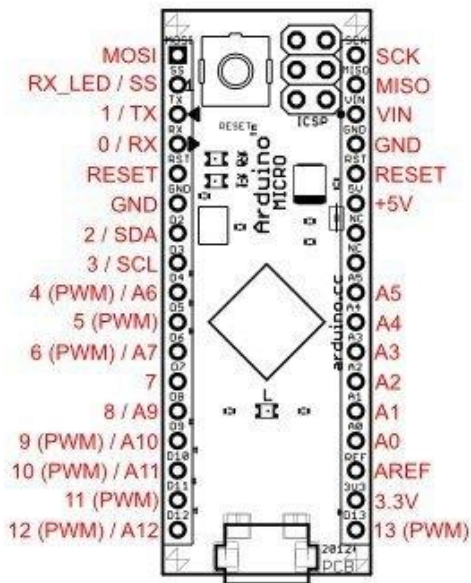


Figure 13. Arduino Micro Pin Schematics [10]

The pin layout of the Arduino micro board is shown in figure 13, and was used as a reference for pin names in this project.

Notice how only pins 2 and 3 can be used for SDA and SCL interfaces respectively.

3.1.2 Sensor attachment

A 3D printed plastic attachment was created to fit the sensors to the linear motor. It had on its top side, a circular flat surface (16 mm diameter). On the bottom end it had rectangular slot to fit the extending head of the linear actuator. Both the actuator extension and the mouth piece had a cutout in which a screw (3 mm diameter) could pass through with a counter nut attached to on the other end.

Double sided tape was used as an adhesive to fit each sensor to the plat surface of the circular shape. See figure 14 for an illustration of the 3D printed part.



Figure 14 Plastic attachment for linear actuator in black (3D printed), and 3 mm screw with counter nut

The attachment had a surface diameter of 18 mm, while the sensors had active sensing areas of diameters, FSR: 15 mm and SingleTact: 14 mm.

3.1.3 FSR sensor

To read measurements from the sensor a voltage divider setup was used, see figure 15. The FSR sensor was connected to the Arduino in the following way:

- Red wire to the 5V pin of the Arduino
- Black wire to the A0 pin

- A 10 K Ohm resistor was connected from the A0 pin to ground

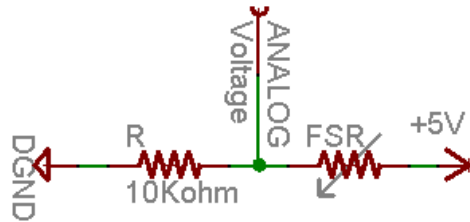


Figure 15 FSR circuit with 10 K Ohm “pull down” resistor [27]

Essentially measurements were taken by using the A0 pin as an analog input and reading the voltage over the resistor.

To use the FSR sensor, due to its nonlinearity, it is necessary to map the resistivity value with the actual force value. This can be done by calibration: by taking simultaneous measurements of the resistance and force. [27]

Force measurements were taken using a Mark 10 Force Gauge, see figure 16. A circular plastic attachment of the same diameter as shown in figure 14, was attached to the Mark 10 to fit the sensors. This was done to ensure that the calibration and the actual tests used mouth-pieces of similar shape.



Figure 16. Mark 10 Force Gauge [42]

3.1.4 SingleTact sensor

The SingleTact sensor, through its connected interface board, had 6 pins which were connected as:

- 1: VCC, was connected to 5V pin of the Arduino board
- 2: Analog Out, was left unused
- 3: I2C Interface (SCL), was connected to Arduino pin 3 (SCL enabled)
- 6: I2C Interface data (SDA), connected to Arduino pin 2 (SDA enabled)
- 7: Frame Synchronization, was not used

- 8: Ground, connected to ground on the full circuit

Note pin number 4 and 5 were reserved by the manufacturer.

This setup did not use the analog reading of the sensor and instead utilized the I2C interface to communicate with the Arduino.

The frame sync pin was not used; it functions by sending a “high” signal whenever a new measurement is available. [20]

3.1.5 Motor and LAC board configuration

The Auctonix linear motor was connected by its built in 5 position ribbon cable to a port on the motor board.

The 5 pins of the ribbon connector carry the following signals in the PQ12- P configuration [28]:

- Pin 1 - Potentiometer reference
- Pin 2 – Actuator Power
- Pin 3 – Actuator Power
- Pin 4 – Potentiometer reference
- Pin 5 – Potentiometer wiper
-

The LAC board were connected at the following pins/connections:

“-“, connected to ground

“+“, connected to 6V DC power source (using the voltage generator)

“RC“, connected to an analog (PWM enabled) port on the Arduino Micro

Potentiometers on the LAC control boards were configured in the following setup, (where maximum input means turning the potentiometer as fully clockwise as possible):

- Speed: 25-degree counter clockwise from maximum speed
- Accuracy: 15-degree counter clockwise from maximum accuracy
- Minimum range: 100% clockwise
- Maximum range: 100% clockwise

Note the min and max settings at 100% clockwise, yields the largest longitudinal movement area possible for the linear actuator.

The above settings were found using the Auctonix motor control software (for Microsoft Windows) available on the manufacturer website. While the highest possible accuracy was preferred, turning the potentiometer to the maximum output for accuracy, caused instability. This is why the corresponding potentiometer was turned back 15-degree counter-clockwise. In a similar manner, the optimum potentiometer setting for speed (while maintaining stability) was found to be a 25-degree counter clockwise twist from the maximum speed.

3.2 Mounting and setup

3.2.1 *Mounting*

The following setup was used for the measurements:

- The linear actuator mounted on a bench vise
- A “target plate” held using a vertical desk mount
- Two black clamps to hold both in place

The clamps were used to hold the bench vise and the desk mount together in place, even when the motor would run with maximum output. The clamps were also used to ensure that the motor position was always at a similar starting distance from the target whenever beginning a new test.

See figure 17 for an illustration of the setup, with a couple of minor details missing:

- One of the black clamps has been removed to show the motor
- The motor is to be positioned 2 mm away from the target plate when the motor is fully detracted, which is done at the start of each measurement

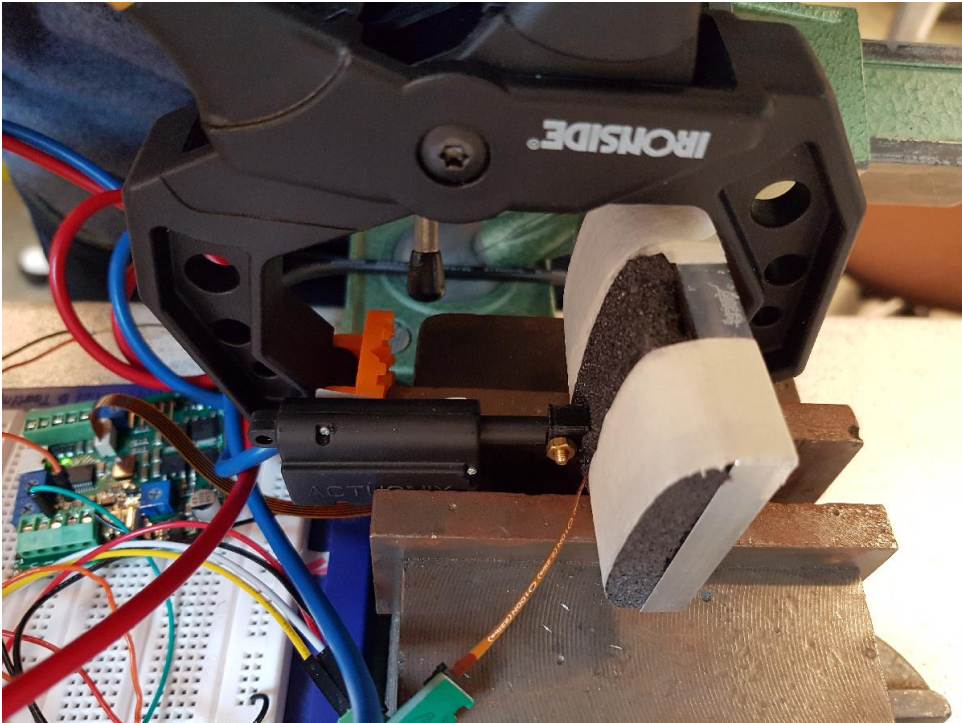


Figure 17 Linear actuator and the target plate. The SingleTact sensor (with its orange flex cable) can be seen attached to the extending end of the motor, touching the black foam of the target plate.

3.2.2 Target plate

A target plate was made as the receiving end for the linear actuator, which is the material the sensors are pushed against. This soft interface was used to stimulate the mechanical properties of the human tissue. Notice however that the human body has different characteristics depending on area for example areas with more fat will provide lesser resistance to areas with more muscle or bone mass. Furthermore, this will also vary from person to person. The material used in this project was made to resemble the soft areas of the upper arm, because this area is often a good option for hand and arm amputees (for receiving mechanical feedback).

This target material consisted of:

- Metal plate
- Soft plastic foam
- Thin layer hard plastic cover (not shown in the figure)

While the metal plate is used as the base, it is coated with the plastic foam to provide a flexible material. In addition to this, a thin hard plastic cover was used as the final layer to balance out the load on the surface of the foam.

Through testing the motor, it was found that because the metal piece alone would not flex: very small changes in the actuator position would result in high changes in pressure as measured by the sensor. Because of the limitation of accuracy in the Auctonix control board, the internal encoder is only able to process changes in position larger than a minimum value.

Because of this fact, using a more flexible material compared to metal was essential to allow the motor have a more suitable retraction/extension distance. The flexible soft plastic foam was used to allow this.

Note that in the start of each measurement, referred here as the “disengaged” position, the motor is not touching the target plate and is positioned 2 mm away.

3.3 Measurement and System overview

In this project the primary coding was written using Arduino as well as Java code.

The Arduino board which was connected to sensor and dc motor, was also programmed to implement PID control of the motor.

Java was used for the graphical interface, which allowed the user to control the motor and collect readings from sensor.

3.3.1 *Measurement, Rise Time and RMS Error*

In this project, measurements were taken using the following process:

- First a force value is sent to the motor, whilst at disengaged state, 2 mm away from the target material.
- The motor then receives an 8 second pulse and attempts to press with enough force to so that the force is registered by the sensor
- At the end of the pulse RMS error and the rise time is calculated.
- Set points: 2N, 3N and 4N were sent to the motor

Ideally further measurements for further set points would provide more data for analysis of the results, however due to time constraints the above 3 constraints were chosen.

8 Second pulse

When the motor received a signal to move in response to a specified force input, there was an 8 second window for the system to move the motor to the desired position. After this duration the motor would disengage and return to the starting position, and the total Rise Time as well as RMS error were calculated.

8 seconds specifically was chosen as a sufficient time length for the motor to reach the desired target force and stay around this point for at least 5 seconds. This number was considered to be a reasonable time for many practical situations for example when the user of a prosthetic arm would only need to feel for a short moment (e.g. opening a door). The trade off is that if the motor was allowed to set for a longer duration the error would be reduced further as the system would go towards a “steady state”.

Rise Time

The rise time for each 8 second pulse, is calculated by recording the time for the actuator to reach 70% of desired position from a fully disengaged state, where the motor is not touching the target material.

RMS Error

The java program calculates the RMS error by collecting 10 different measurement data from the sensor and by computing the following steps:

- The 10 latest error values are saved into a list. These errors are the difference of force value between the force sensor measurement and the actual input from the user.
- The square root of each error value is summed
- The total sum is then divided by 10
- The RMS error is then calculated by taking the square root of above number
- At the end of the 8 second pulse, the RMS error is collected

3.3.2 System and data transfer overview

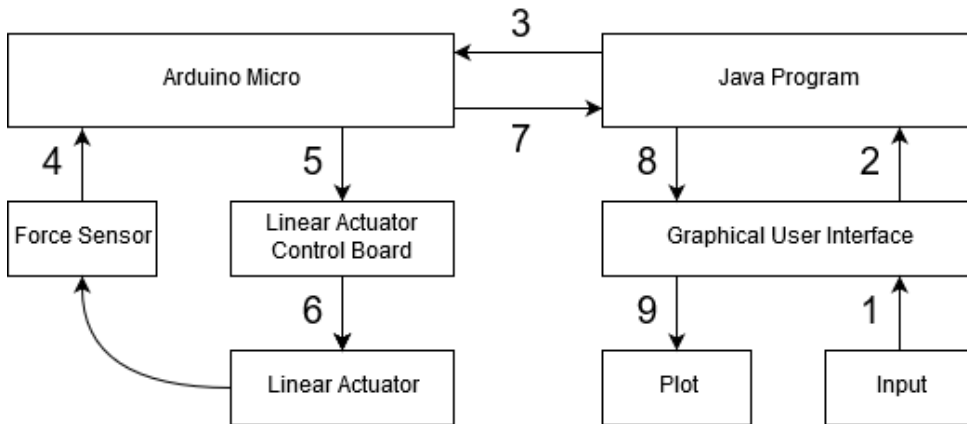


Figure 18 Overview of the system

The system in figure 18 shows the overview of how data is transferred as a measurement is taken, it can be simplified into the following steps:

1. The user selects an input force on the GUI
2. The “pulse button is pressed”: and the Java program reads the entered force value, and sends an 8 second pulse
3. Through the COM port, the force value is sent to the Arduino Micro
4. Force measurement is taken by the sensor
5. Using the PID controller of the Arduino code, a motor output signal is calculated and sent to the LAC board
6. The LAC board sets the linear actuator to the desired position, which in turn changes the pressure on the force sensor (the latter is represented by a curved arrow in the figure)
7. Both force measurement data and current motor position are sent to the Java program using the COM port
8. At the end of the pulse: the JAVA program calculates the RMS error as well as Rise time in milliseconds
9. Sensor measurements and motor position data are placed into each of the respective graph windows of the GUI

3.3.3 Software Flow Chart

To further explain the above steps and how the Java and Arduino programs work together, figure 19 shows a flow chart of the process.

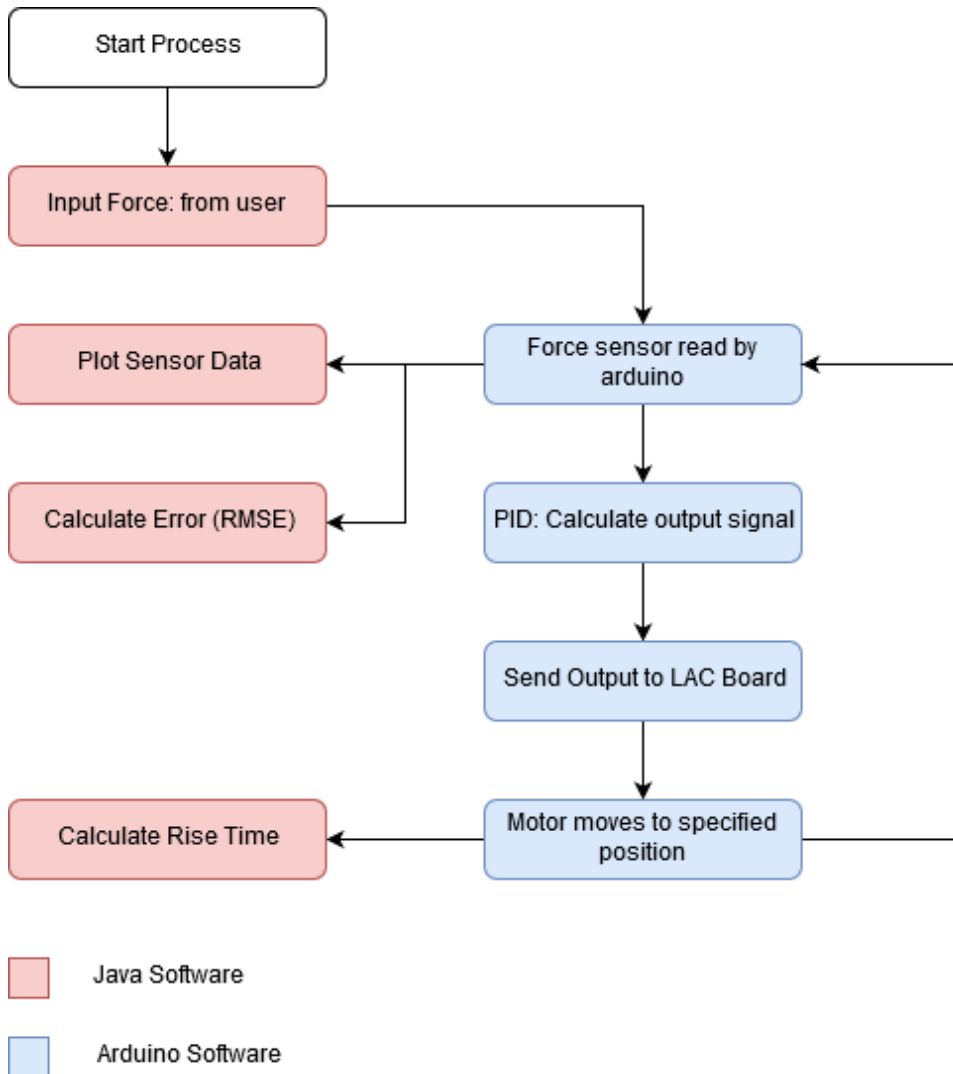


Figure 19. Flow chart of the measurement process. The red and blue colors are used to illustrate which in which program the step is executed, in Java or Arduino respectively

3.3.4 *Graphical User Interface*

To allow for input and easy view of sensor data, a graphical interface was created using Java Swing.

The GUI consists mainly of the following elements:

- Connect/disconnect buttons (attempts to connect to Arduino on specified com port)
- Drop down menu to select com port to connect
- Sensor data plot
- Motor position plot
- A scroll bar and a text field to select motor output signal
- Text fields to view the RMSE and Rise time values.
- Buttons to pause plots, toggle settings/regulator parameters

The GUI functions by reading and sending data to the Arduino micro-processor using the Com Port. The user is able to get an overview of current and past sensor values as well as motor control values. For input the target pressure can be changed using a scroll wheel or typing a letter into the textbox.

See figure 20 for a full page illustration of the GUI.

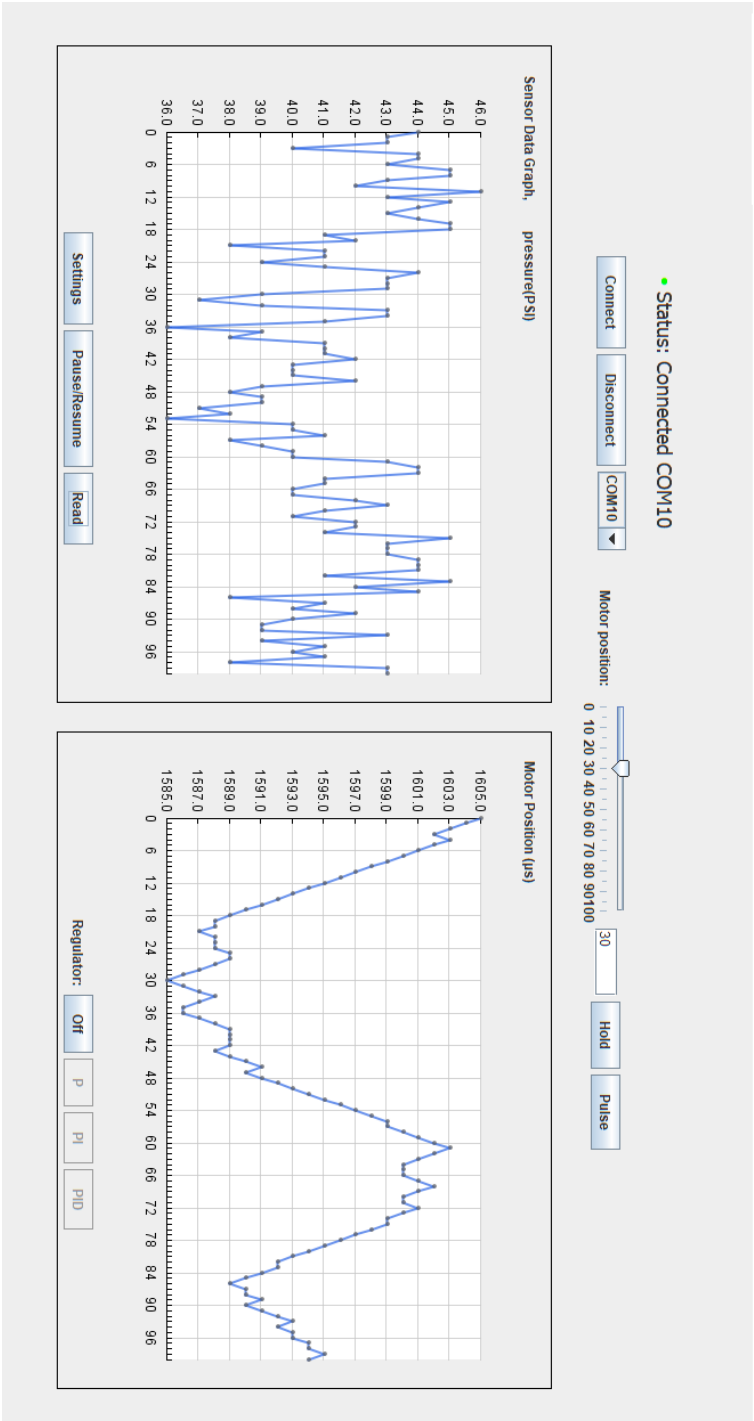


Figure 20 Graphical User Interface

4 Results

In this section only a limited portion of the full measurements are presented to facilitate ease of reading. The graphs in this section shows resulting RMSE and Rise times for target 2 N. Please note that diagrams for targets 3N and 4N can be found in appendix section.

The complete measurement data for FSR, SingleTact, RMS error and rise times, can be found in the appendix section at the end of this report.

Each box plot consists of a set of 10 measurements.

The measurements are analyzed later in the Discussion section.

Small note: the figures in this section, illustrating the box plots are marked Target=200 (same as 2N), Target = 300 (same as 3N) and Target = 400 (same as 4N).

4.1 FSR Measurements with Mark 10 Force Gauge

The table below shows in the left column: the force value on the FSR sensor (read by the Mark 10 Force Gauge), whereas the right column shows the voltage of the resistor in the voltage divider circuit.

A plot using the measurement data in the above table can be found in figure 21. Note that this voltage actually refers to the voltage over the pull down resistor (monitored by the Arduino board) and not the voltage of the sensor itself.

The full table of measurements used for the plot can be found in Appendix – FSR Calibration Data, at the end of the report.

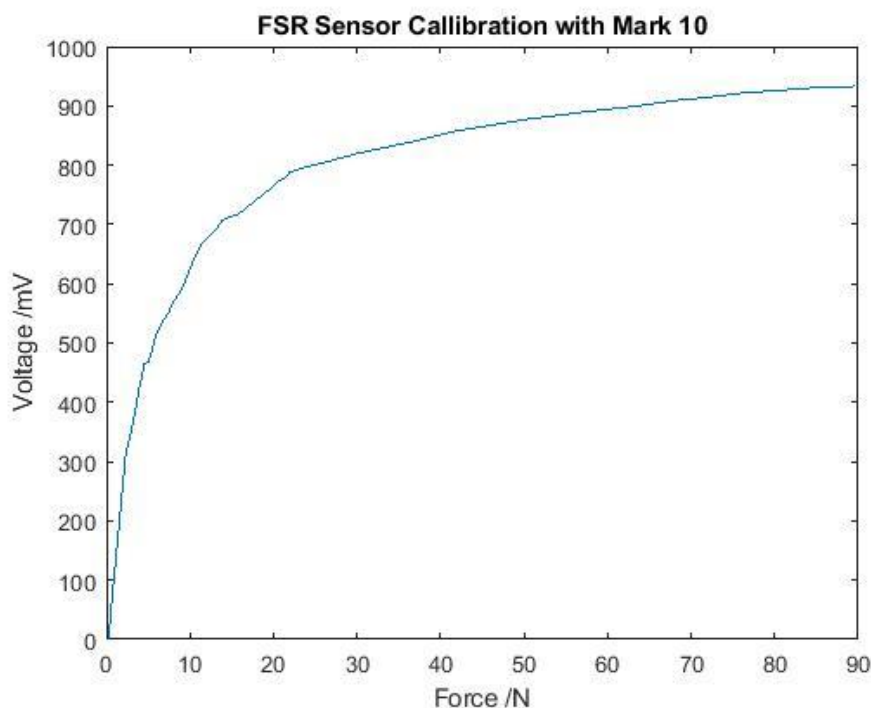


Figure 21 FSR calibration - voltage/force curve

4.2 Proportional control

Rise times - Proportional Control

Figure 22 Shows how the rise time for the system with the FSR sensor is affected by changing the proportional constant term K_p . Figure 23 Shows how the rise time for the system with the SingleTact sensor is affected by changing the proportional constant term K_p .

For plots of both sensors, notice the downward slope which eventually starts to flatten at increased constant values.

RMSE - Proportional Control

Figure 24, top, shows the RMS error for the system with FSR sensor, when changing the proportional constant. Figure 24, bottom, shows the RMS error for the system with SingleTact sensor, when changing the proportional constant.

For both plots it can be observed a noticeable decrease in error at the $K_p = 0.12$ - 0.14 range.

The median error in $K_p=0.12$, showed around $0.45N$ for the FSR and $0.8N$ for the SingleTact which is roughly 25% error, considering target of $2N$. Furthermore, at this value rise times were recorded at 750 ms for FSR and 520 ms for SingleTact.

4.2.1 Proportional control - Rise time - FSR

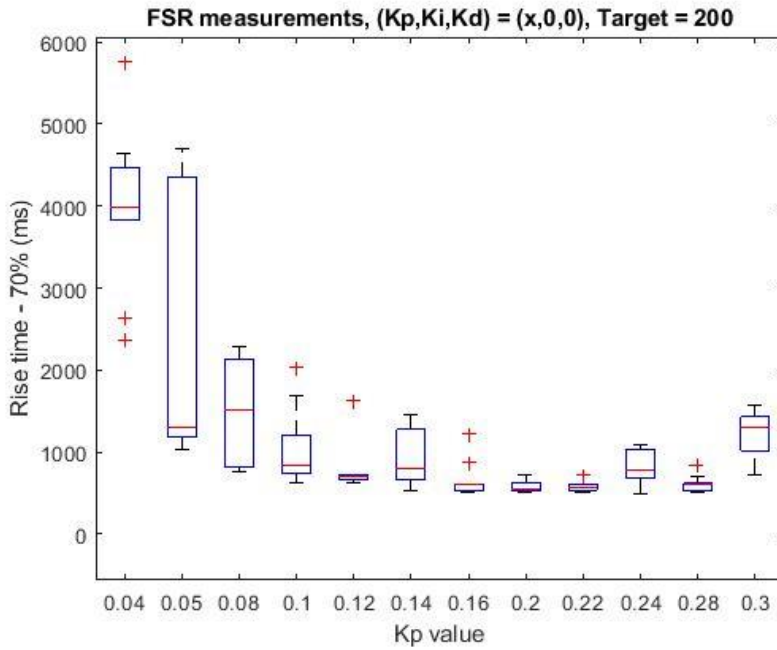


Figure 22 Rise Times with FSR sensor and proportional control

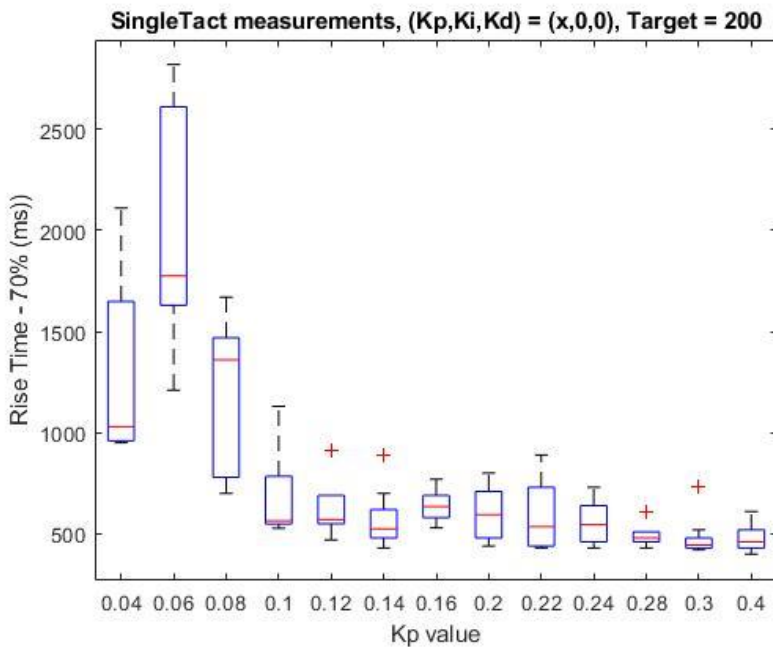


Figure 23 Rise Times with SingleTact sensor and proportional control

4.2.2 Proportional control - RMSE

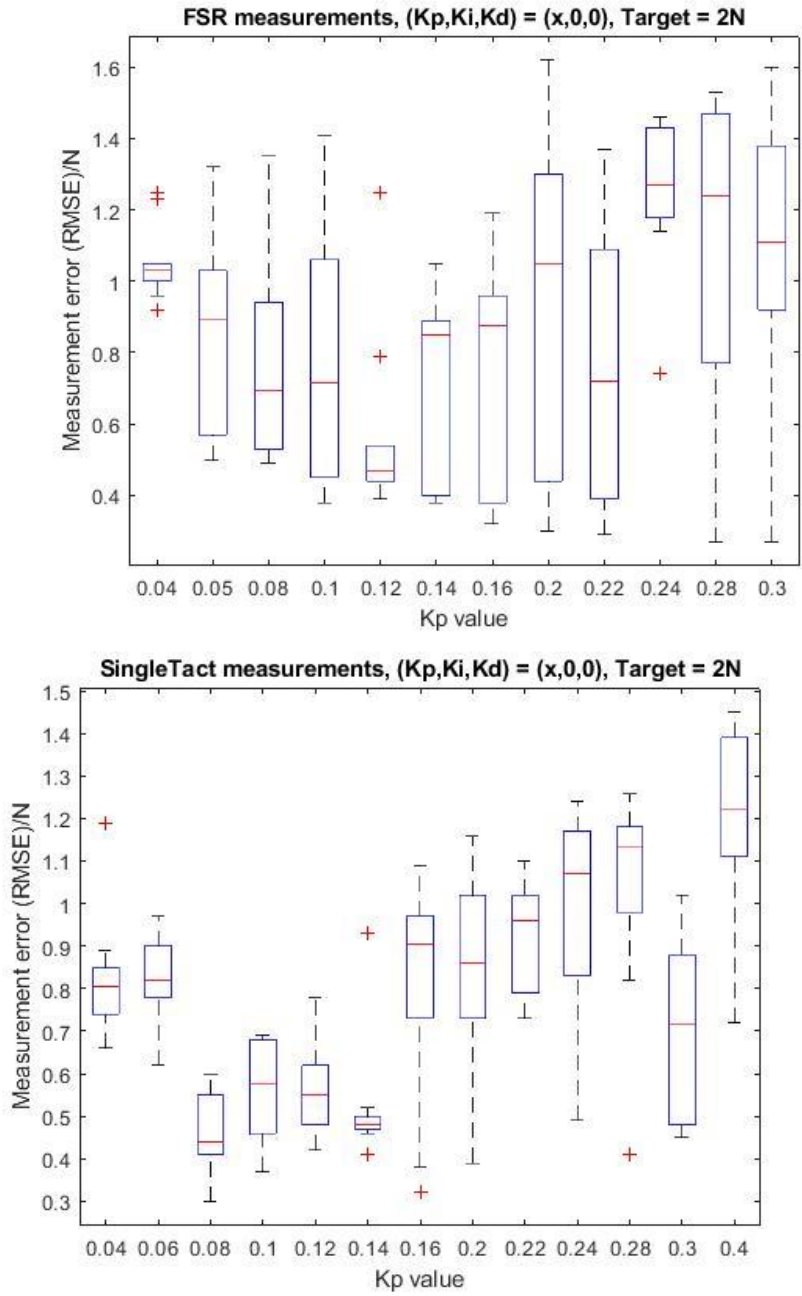


Figure 24 Proportional constant measurements, FSR and SingleTact

4.3 Integrative and Derivative Control

RMSE – Integrative Control

Figure 24, top, shows the RMS error for the system with FSR sensor, when changing the integrative constant. Figure 24, bottom, shows the RMS error for the system with SingleTact sensor, when changing the integrative constant.

For the top, FSR diagram, there is a lack of noticeable reduction when decreasing changing the integrative constant. For the lower SingleTact diagram, it is possible to observe a small reduction at $5e-06$ and $1e-05$ values, 0.000005 and 0.00001 respectively.

RMSE – Derivative Control

Figure 25, top, shows the RMS error for the system with FSR sensor, when changing the derivative constant. Figure 24, bottom, shows the RMS error for the system with SingleTact sensor, when changing the derivative constant.

For both diagrams no improvement in error can be observed when changing derivative constants.

4.3.1 Integrative control

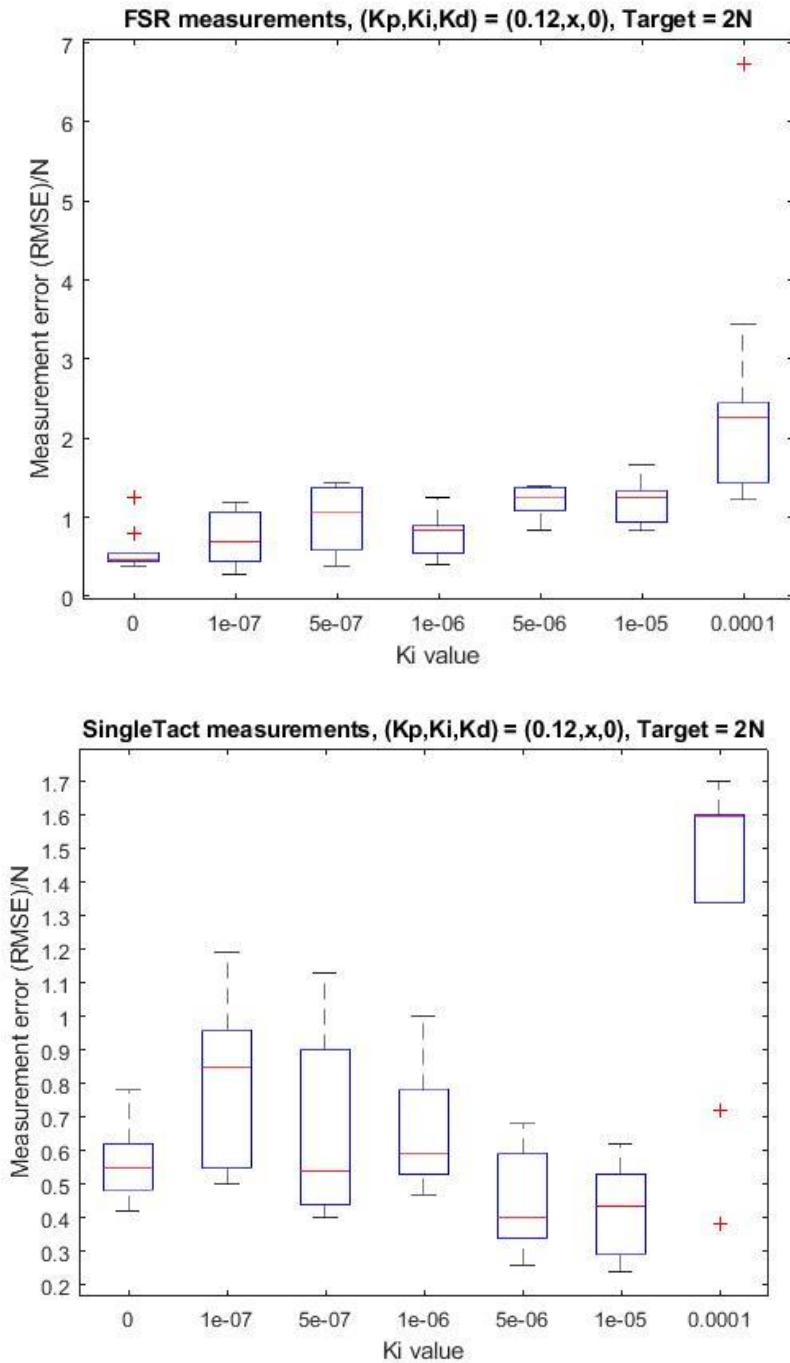


Figure 25 RMS Errors with Integrative control, FSR and SingleTact

4.3.2 Derivative control

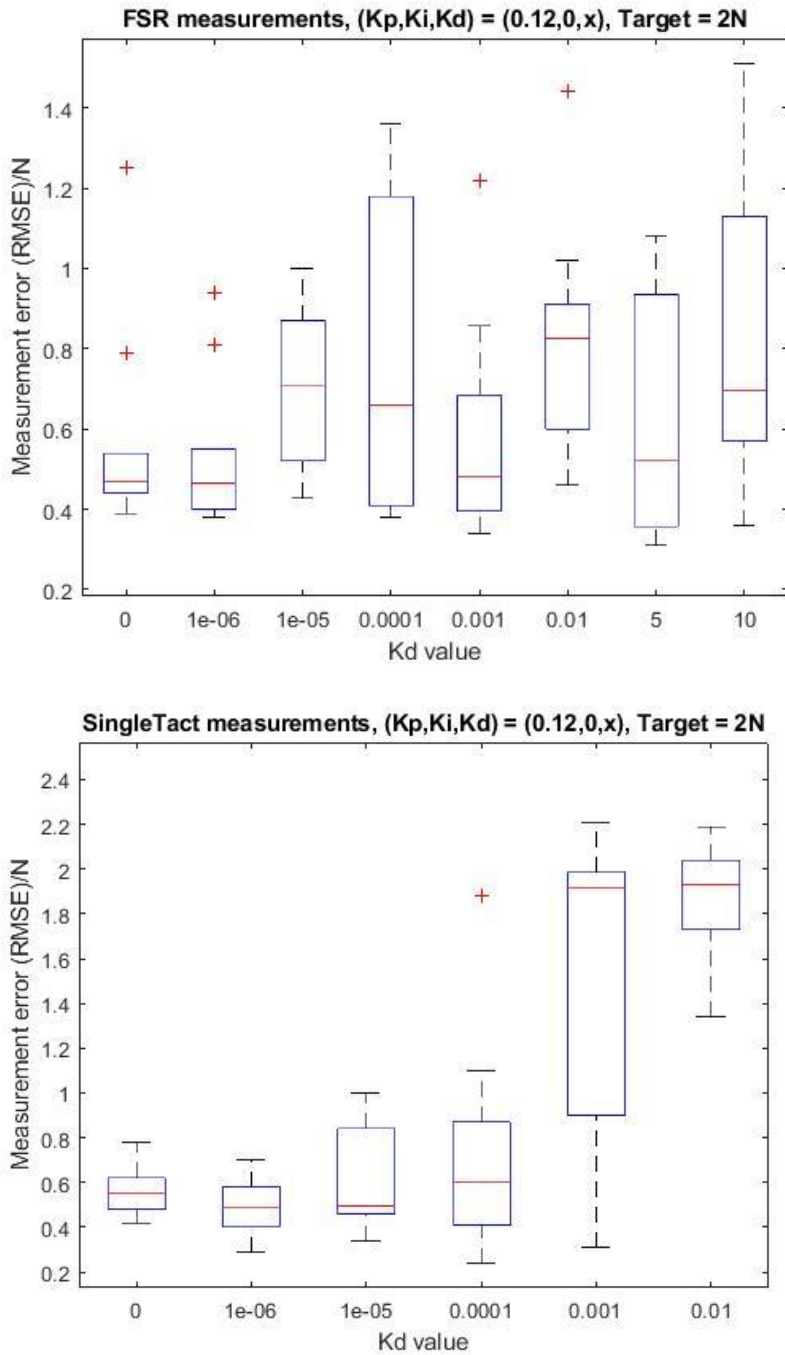


Figure 26 RMS Errors with Derivative control

5 Discussion

5.1 Measurements and PID Parameters

5.1.1 *RMS Error*

The results show that the smallest RMSE error is attained by using K_p value at 0.12 for both sensor types. This can be considered to be the optimal proportional gain parameter. The lowest RMS error for both sensor is at 0.5 N, which is as mentioned 25% for target of 2 N. Notice that this error percentage is a function of 8 second pulse. Increasing the pulse duration would allow the system to stabilize closer to the target value (steady state) and thus the percentage would decrease further.

5.1.2 *Comparison of FSR and SingleTact sensors*

Looking at figure 17, around the $K_p = 0.12$ range, by comparing the RMSE for both sensors it may be possible to conclude that by using the SingleTact sensors yielded in less error and variance (see heights of the box plots), compared to the case with FSR.

It could however be argued that many more measurements are needed to support the above statement, see the section about “automated test bench”, in chapter 6, Conclusion.

5.2 Sources of error

5.2.1 *SingleTact offset error*

While the SingleTact interface board reduces offset error, the presence of offset error was still observed when reading force values. When not under load the Sensor could fluctuate between 0-0.1 N.

5.2.2 *FSR and calibration accuracy*

During the calibration around 100 measurements were taken by manually pushing with the force gauge against the sensor. Whether enough data points were taken to achieve accuracy in the band of force used in this project, can be argued.

In a report (Basel 2017), the authors have found that the use of voltage dividers are inadequate for reading the resistance of an FSR. They have concluded that because voltage dividers as well as Wheatstone bridges and multi-meters operate on basis of changing the voltage across an unknown resistance, a modulation effect is created as the applied force and voltage across the sensor changes simultaneously. The use of an amplifier in “inverting configuration” to collect sensor data has been suggested instead. [29]

5.2.3 *Motor and Encoder limitation*

From analyzing the measurement results it can be concluded that there is room for improvements in terms of motor performance.

The internal encoder used for the PQ12 DC motor had a limitation in terms of detected accuracy. When sending a signal to the motor board to retract or extend its arm, the difference in position will need to be large enough for the encoder to process this value. Meaning that too small changes in position values would not be registered by the encoder. As an effect of this, overshoot or instability could be observed as the controller attempted to control to a certain set point (i.e. the force value).

5.2.4 *Human factor*

Because a lot of parameters as well as measurement data were manually written down or typed in, the possibility of human errors should not be disregarded.

6 Conclusion

6.1 Final system

In this project a system was created which could allow control of a motor with an input of specified force from the user.

Both sensor types have shown to provide comparative performance, and measurement results from this report was insufficient to decide whether or not one should be preferred over the other for use in a prosthetic arm.

Some of the findings from the measurements were that the proportional constant value K_p , affected the RMS error and thus the accuracy of the system. Finding the optimal value for this parameter resulted in reduced error. The integrative and derivative actions did not seem to have any significant positive effects on the system.

Overall it can be argued whether or not the final system is accurate enough for use in a real world product. At 25% RMS error for the 8 second pulse simple actions could be done while more intricate action will be a bit challenging to perform. The user of the system may need to perform action for durations longer than 8 seconds to allow the feedback system to stabilize at lower errors.

A rise time of 500-700 ms can be considered quick enough if the prosthetic user is expected to perform actions at minimum 5-8 seconds (e.g. holding a cup), but may not be sufficiently quick for much shorter actions (e.g. knocking on a door).

6.2 Further Work

6.2.1 *Other sensor types, Strain Gauge*

Due to time constraints only data from FSR sensors and SingleTact sensors were analyzed, ideally further sensor types could also be assessed. This could as an example be the strain gauge, see figure 22

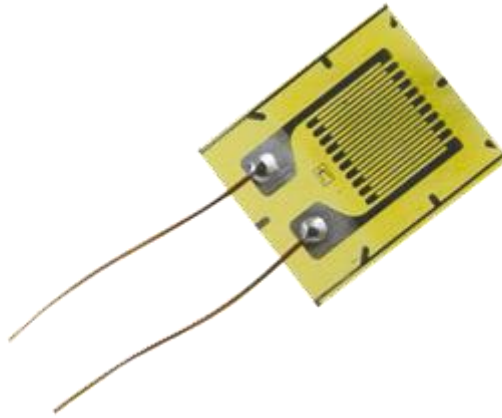


Figure 27 A strain gauge from Kyowa Electronics [30]

6.2.2 System identification

The use of system identification could be used to mathematically find optimal PID controller parameters. This step was excluded in this project as it was deemed to be out of scope.

6.2.3 Automated test bench

In addition to above or as an alternative, further improvements could be found by further automation of the measurement process. An automatic test bench could be designed for collecting a lot more measurements. These would allow for analyzing PID parameters over a much larger span than what was assessed in this project.

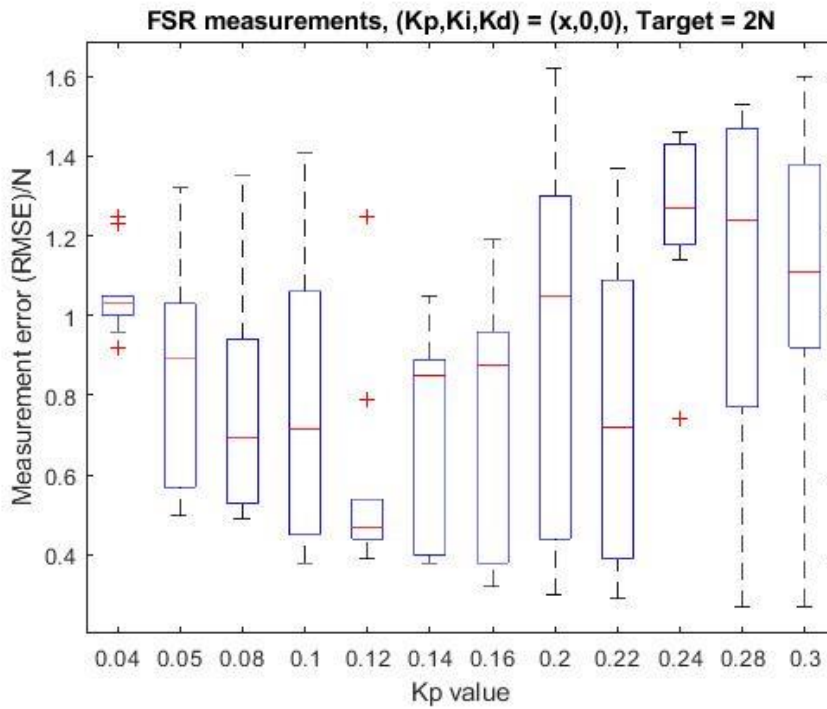
The software used here to take the measurements, the Java program, would have to implement the following principles to achieve above:

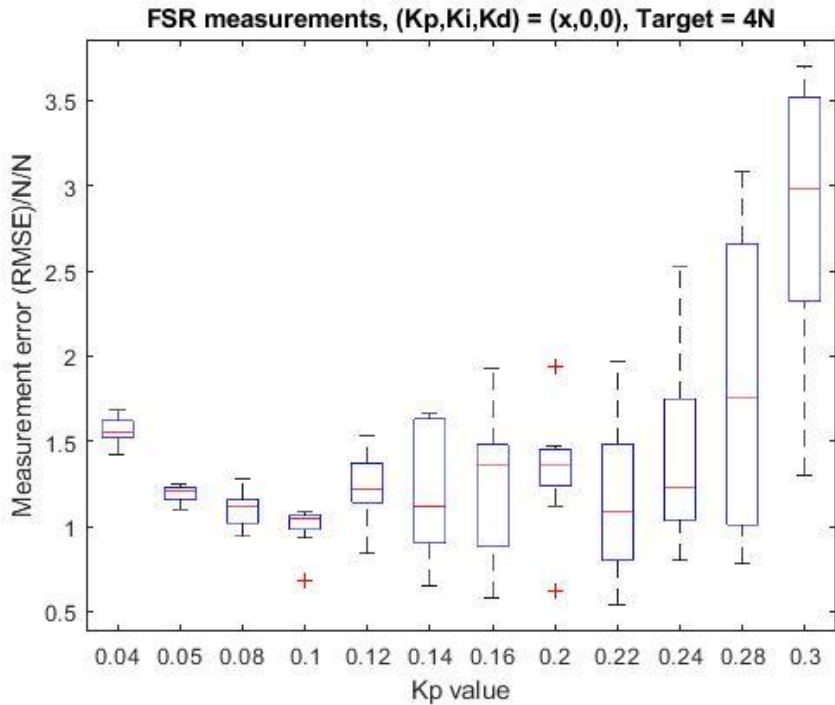
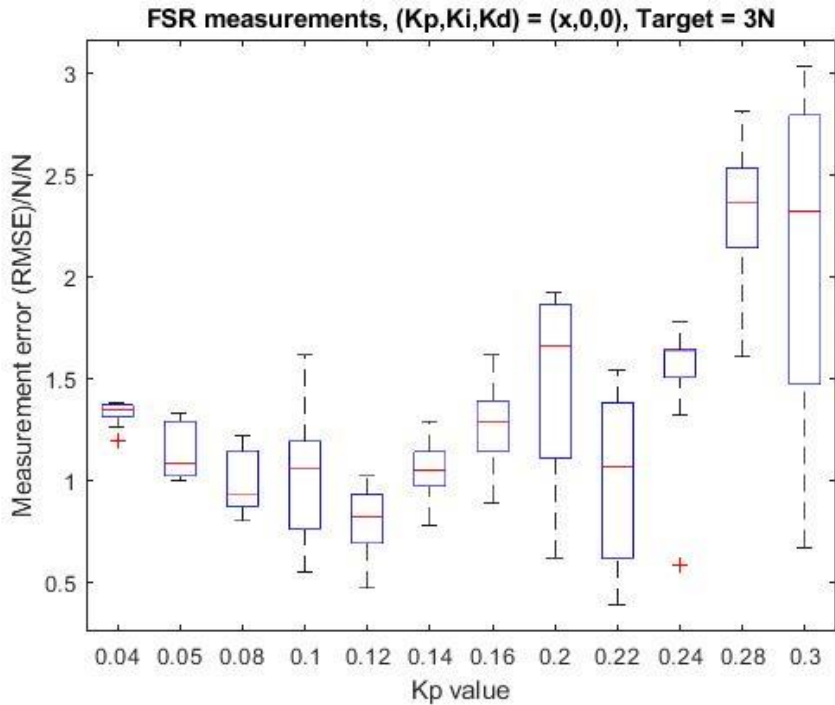
- A software loop needs to be present which would take a measurement, incrementally increase PID parameters and take another measurement.
- The program would need to be able to write all of the above measurements into an external document, such as an excel sheet, for further export to a graphing tool (e.g. Matlab)

Appendix

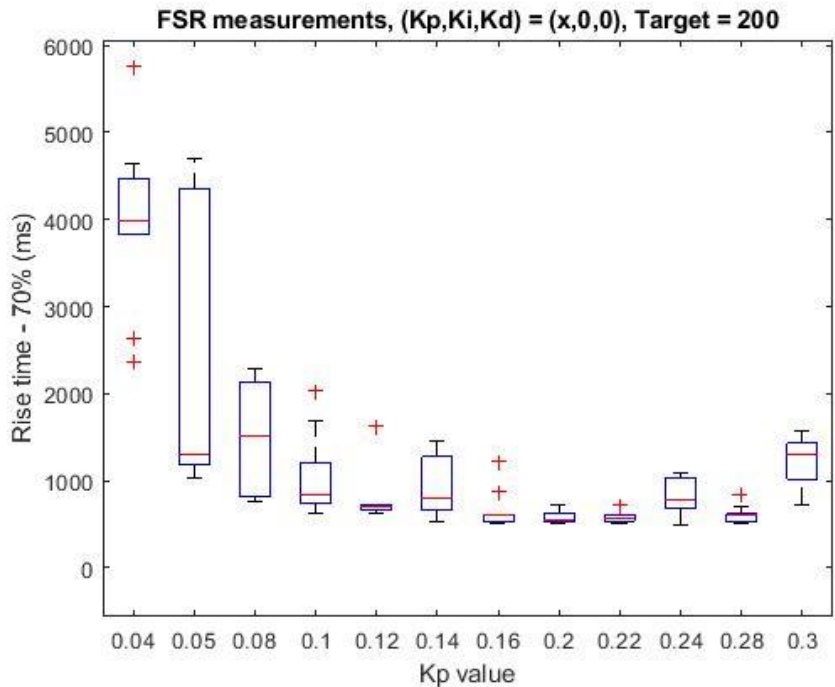
6.3 Appendix - FSR measurements: Box Plots

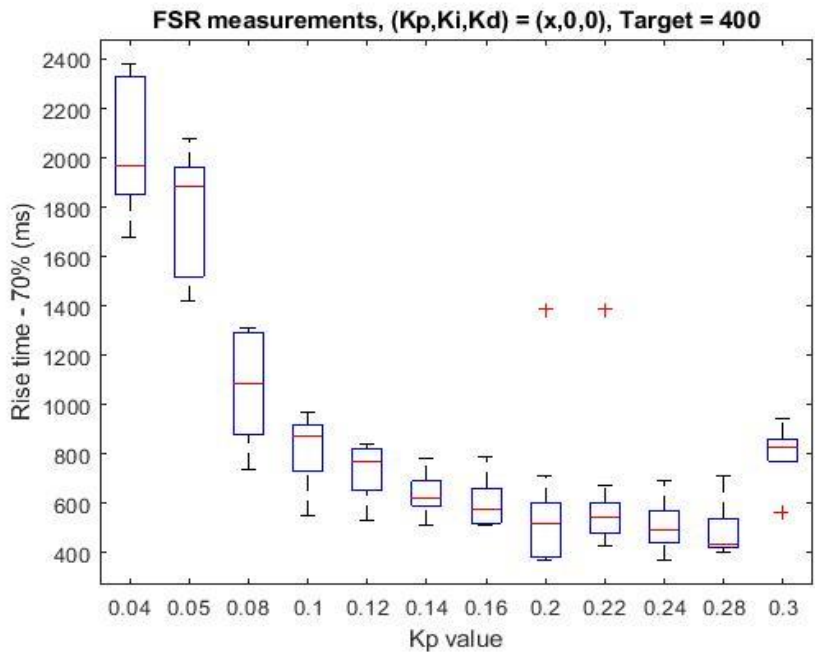
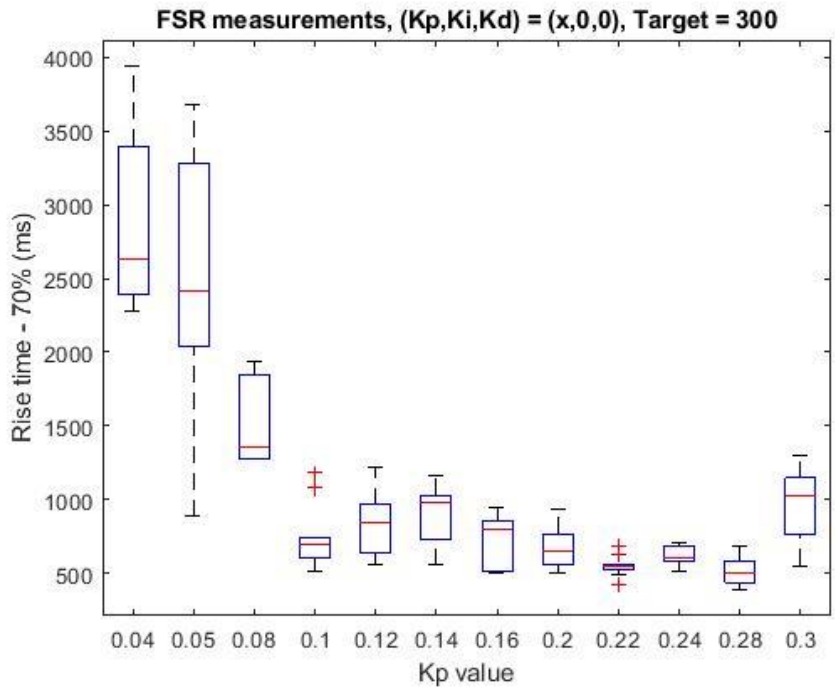
6.3.1 Proportional control RMSE error: FSR



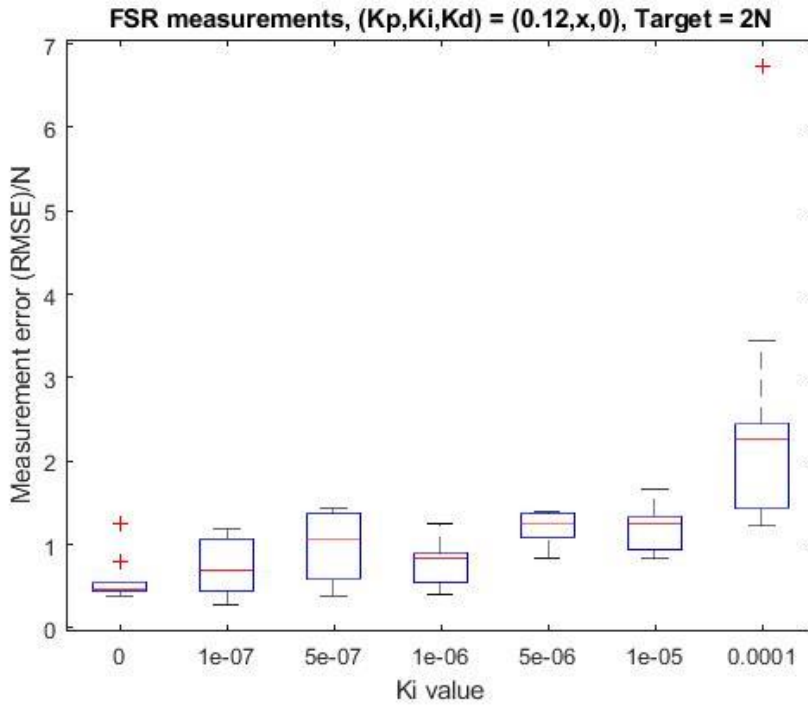


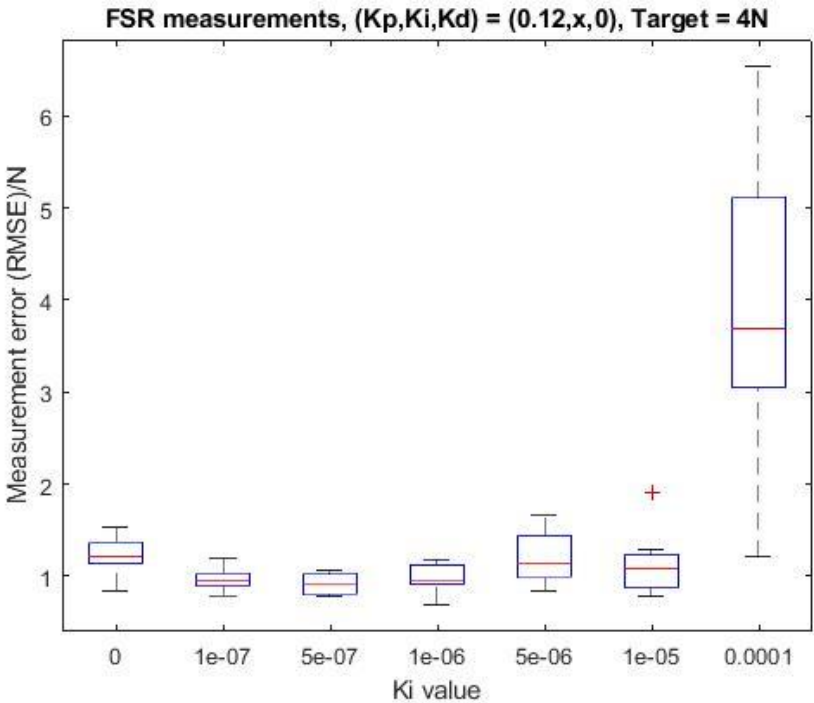
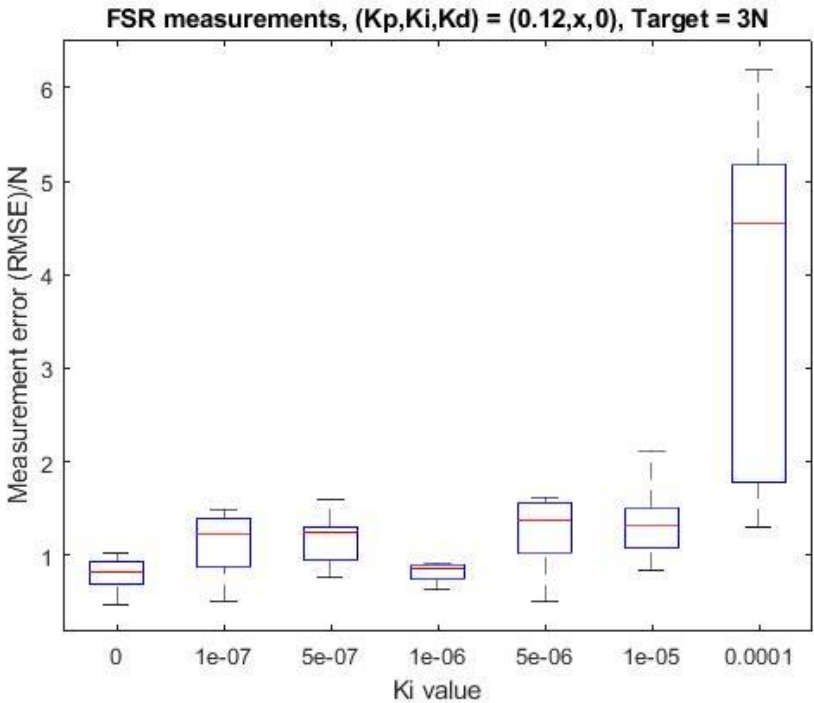
6.3.2 Proportional control Rise time: FSR



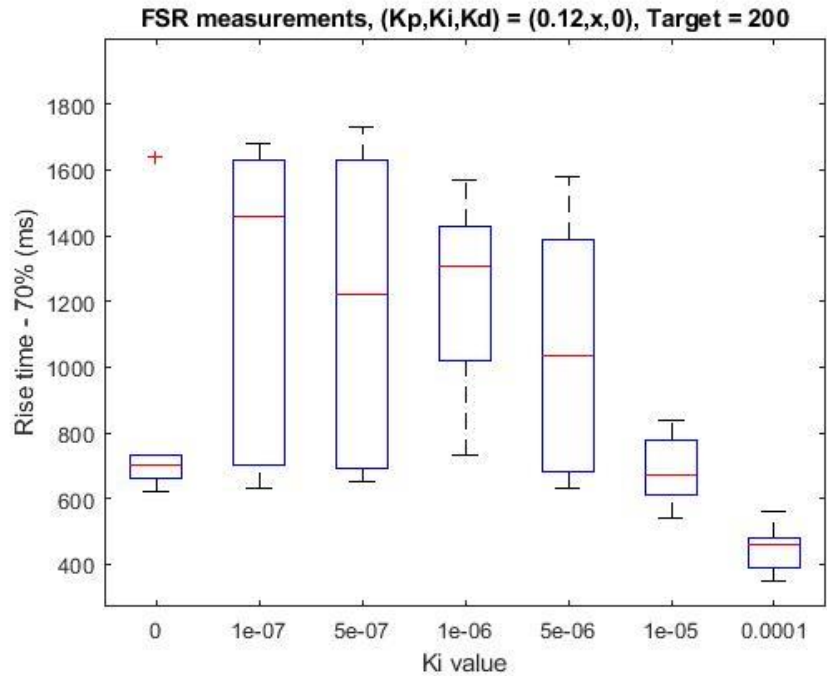


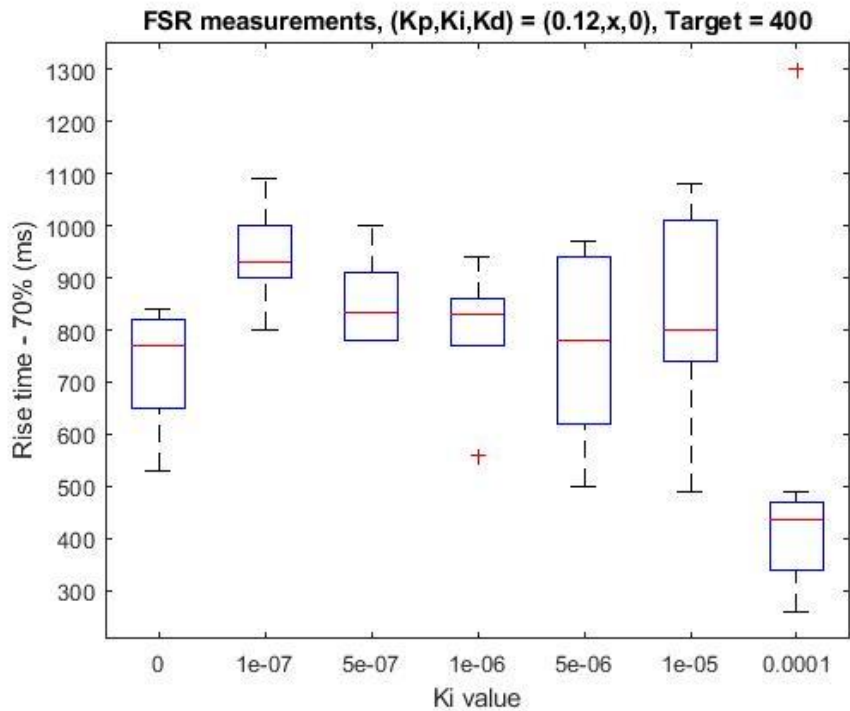
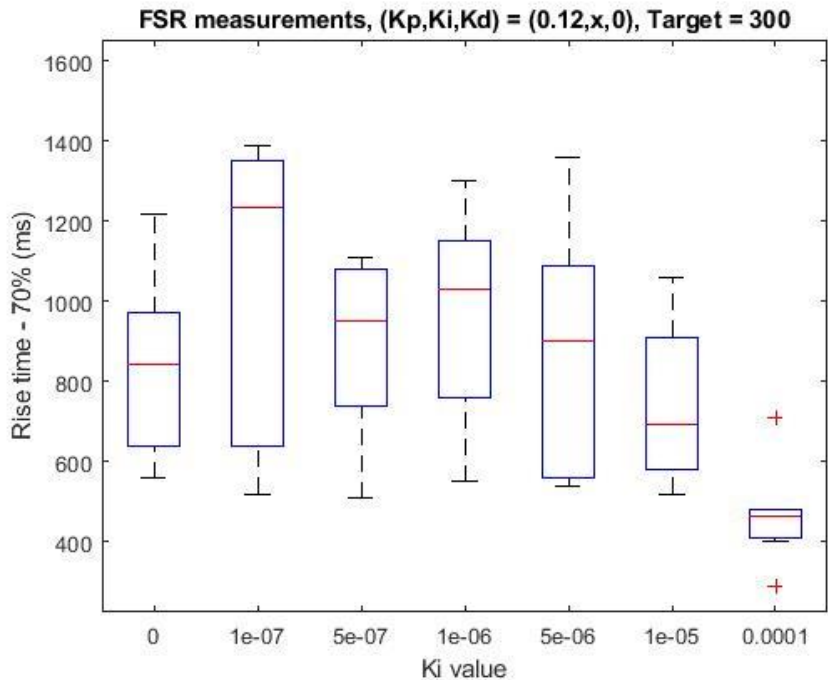
6.3.3 Integrative Control RMSE: FSR



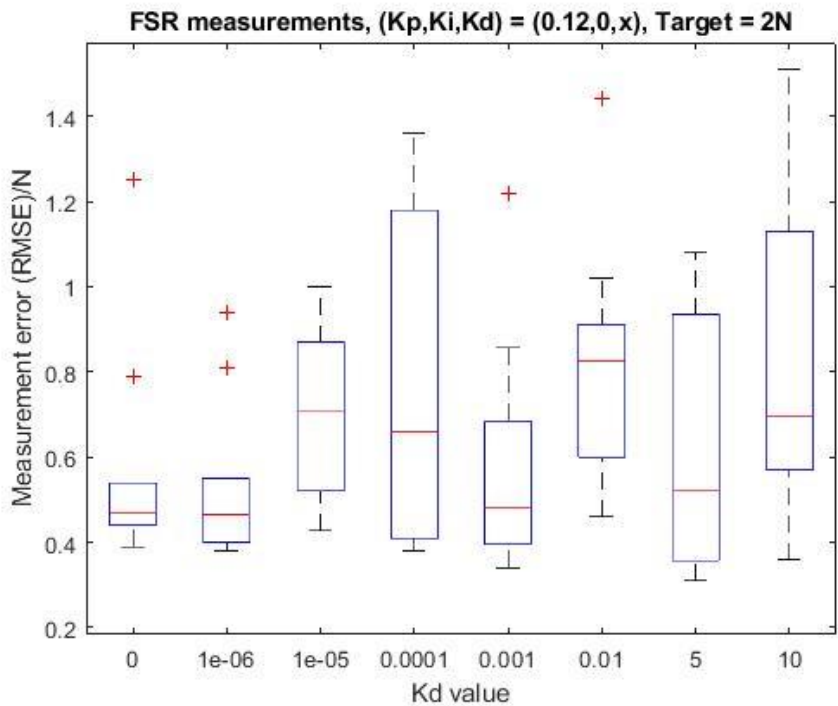


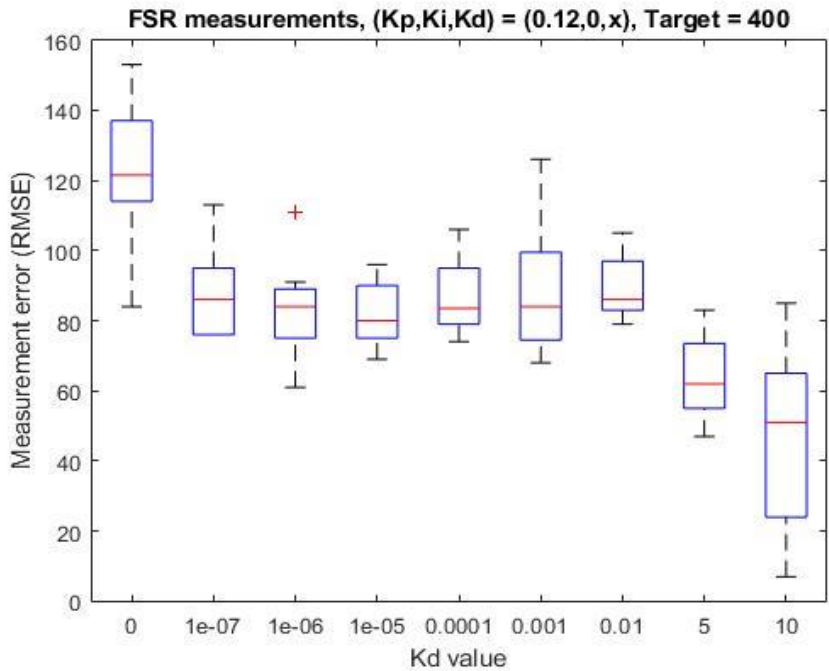
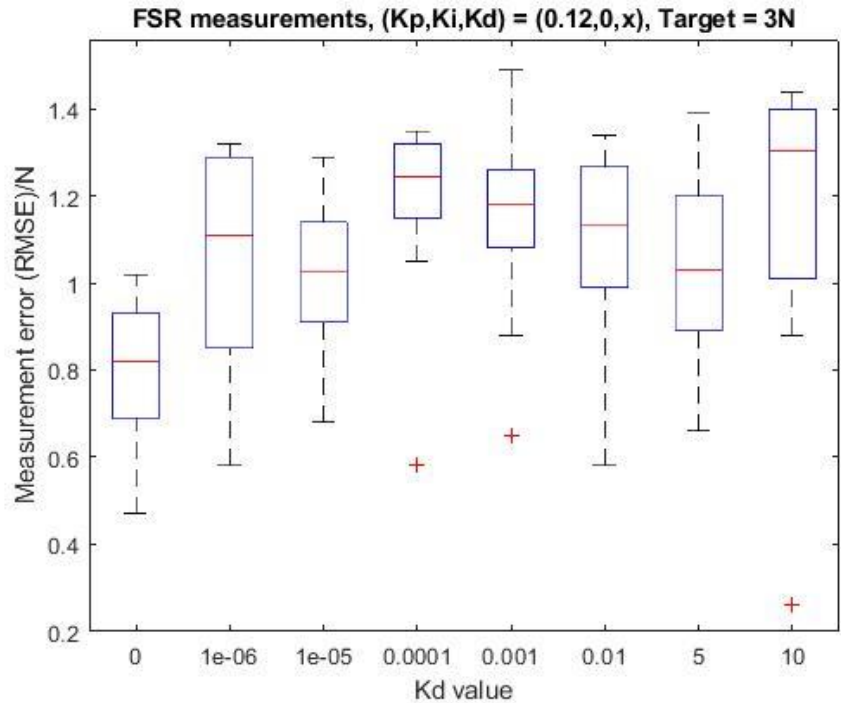
6.3.4 Integrative control Rise time: FSR



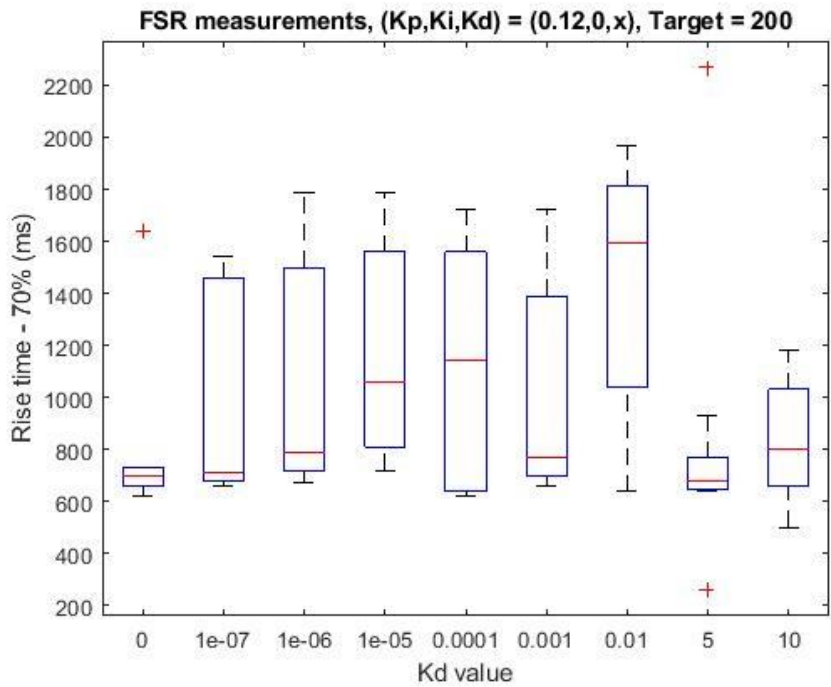


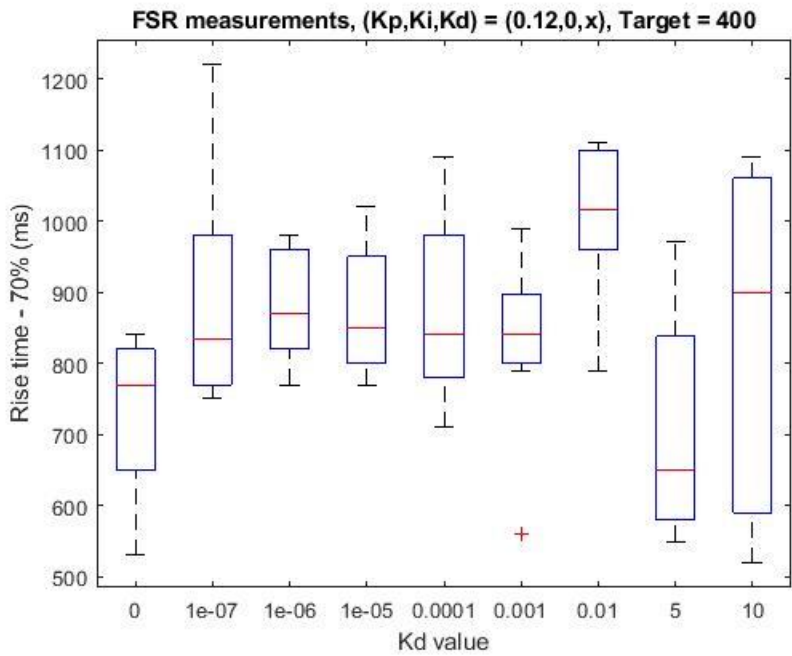
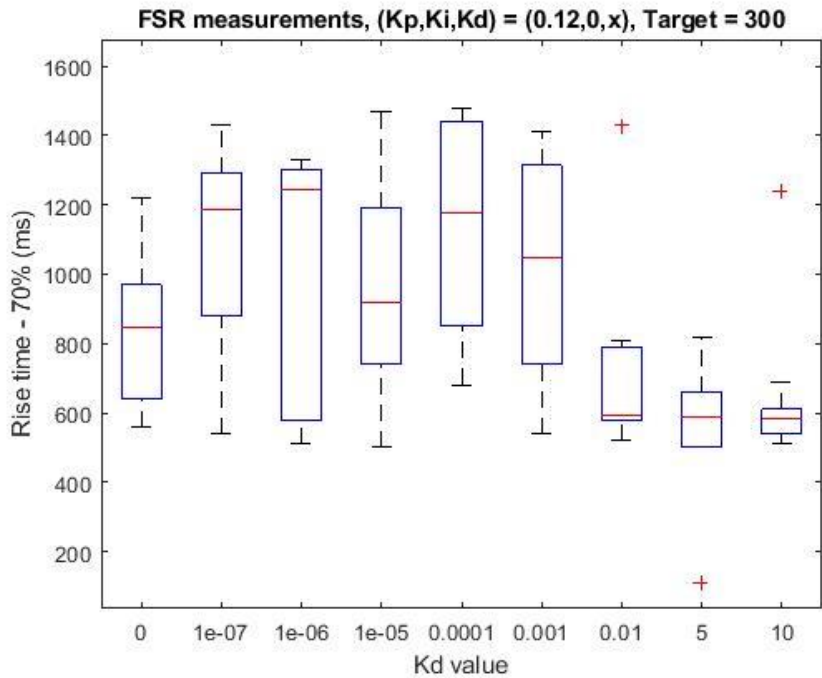
6.3.5 Derivative control RMS Error: FSR





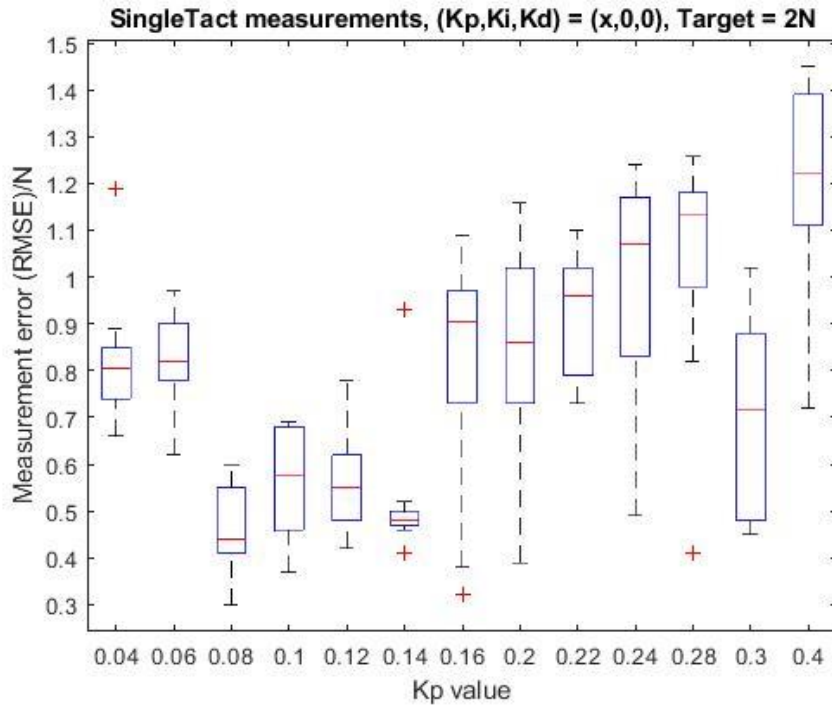
6.3.6 Derivative control Rise Time: FSR

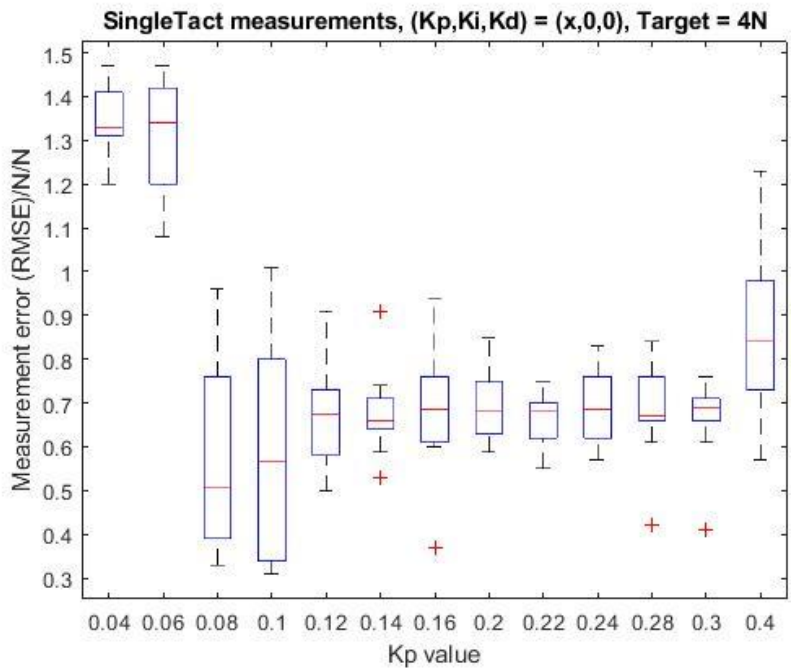
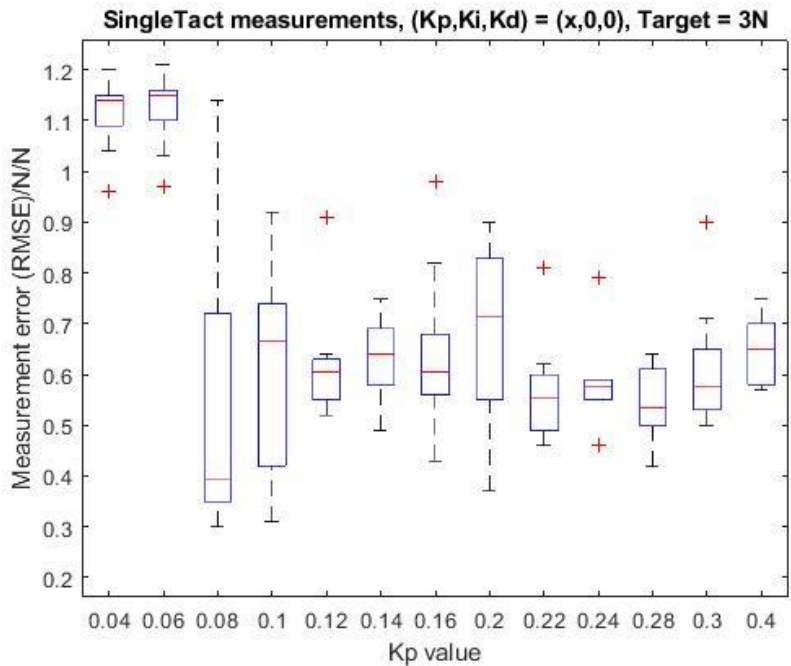




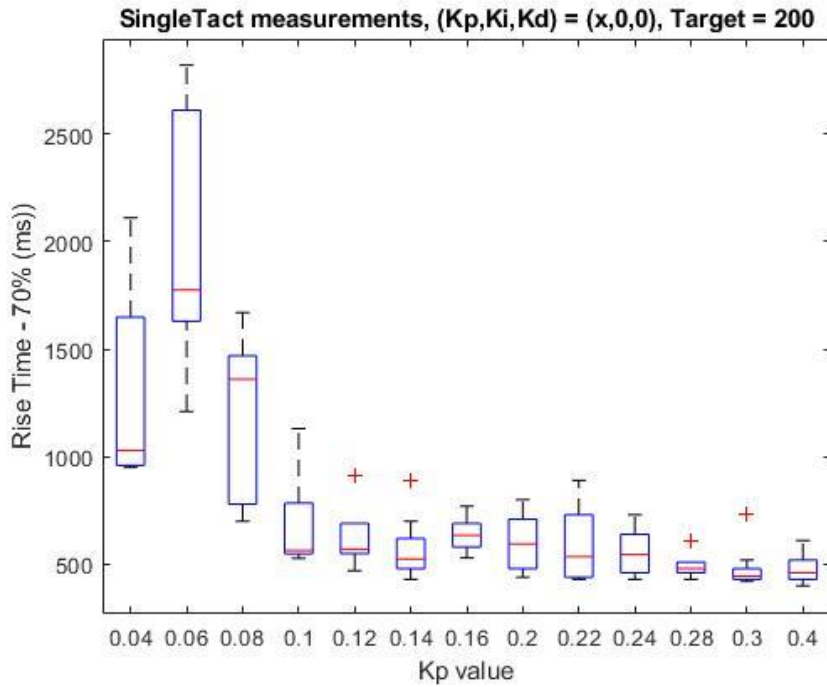
6.4 Appendix -SingleTact sensor measurements

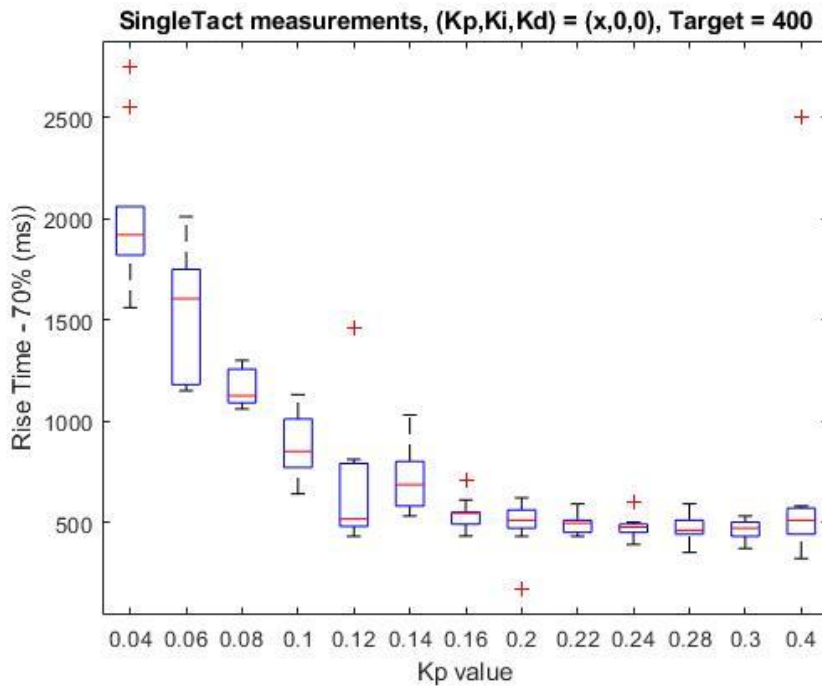
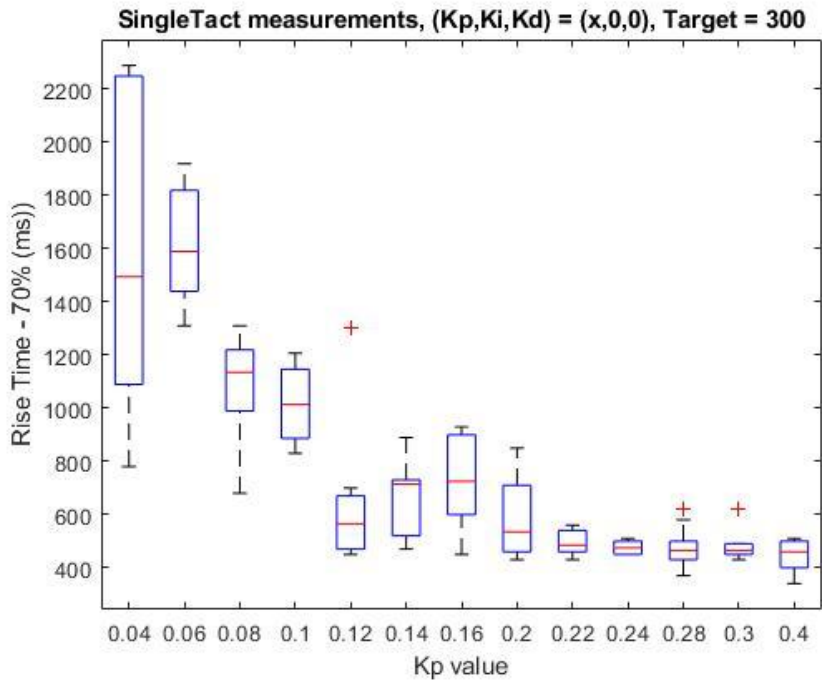
6.4.1 Proportional control RMSE: SingleTact



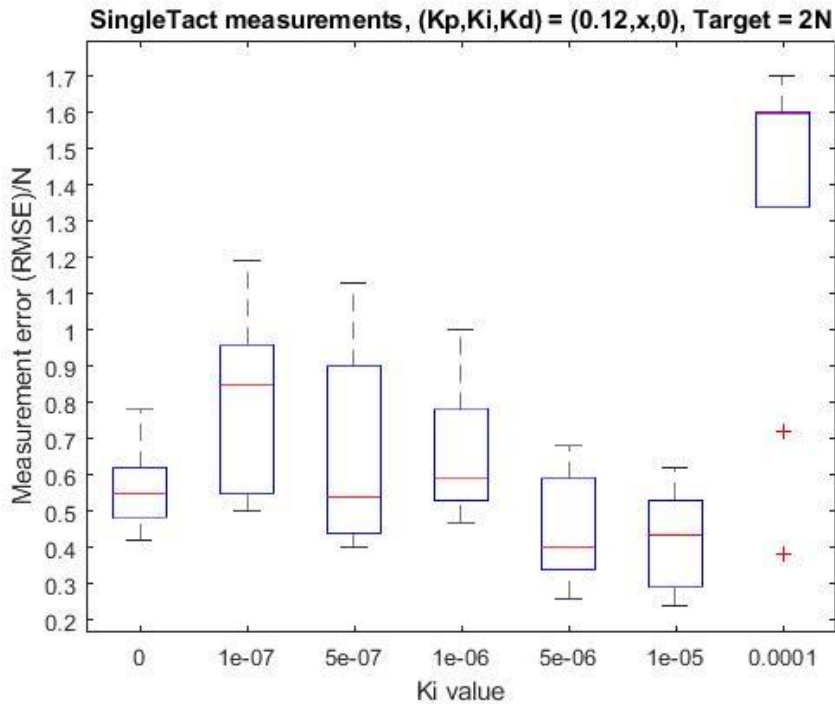


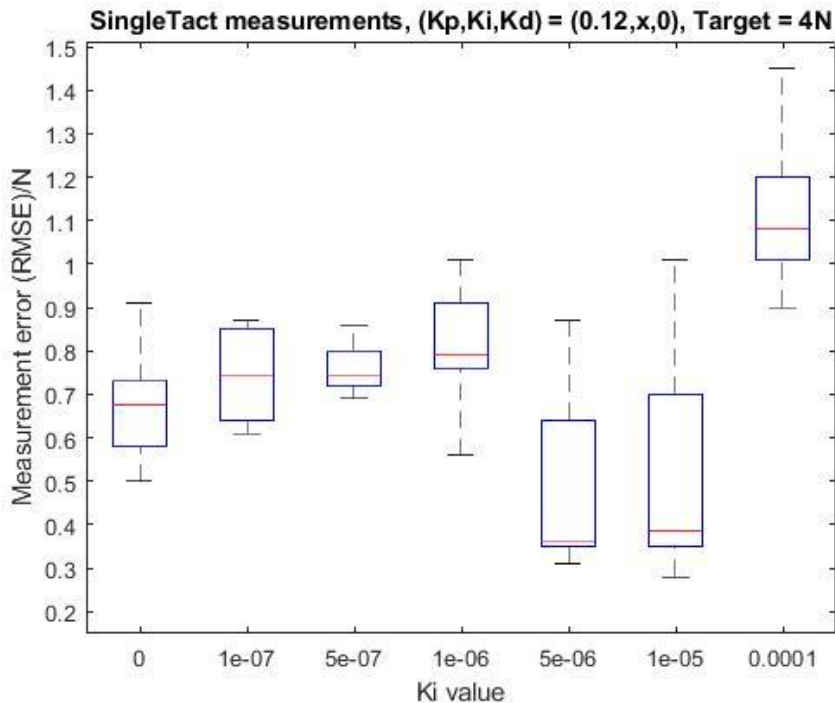
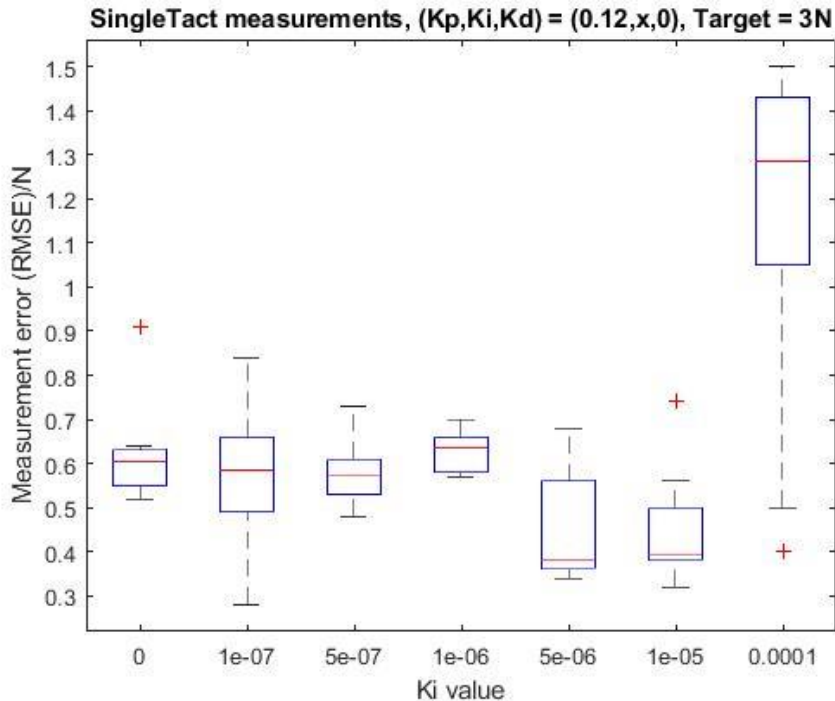
6.4.2 Proportional control Rise time: SingleTact



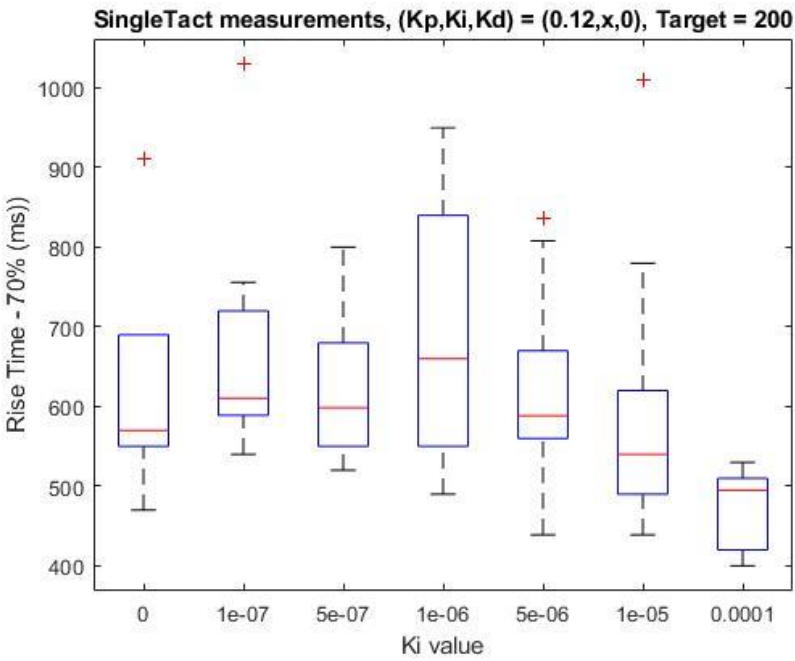


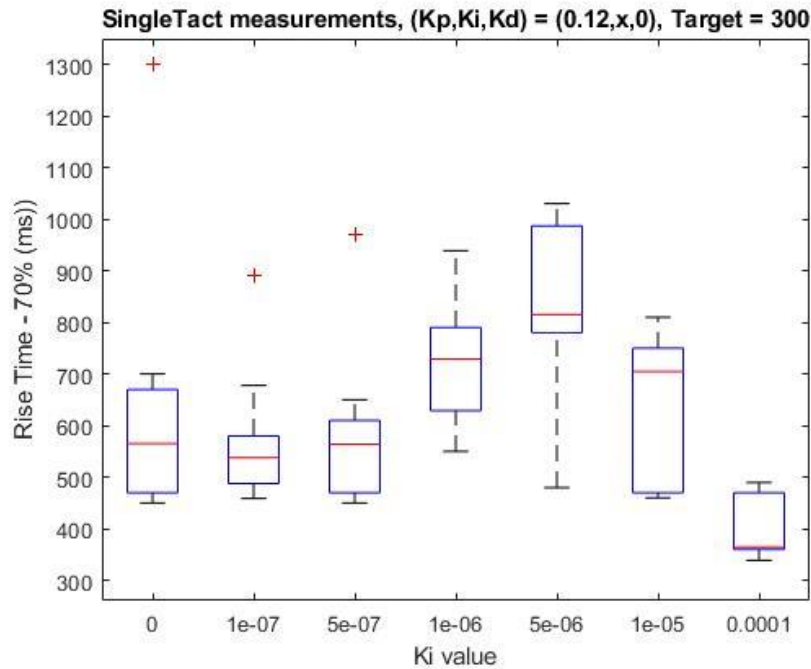
6.4.3 Integrative control RMSE: SingleTact

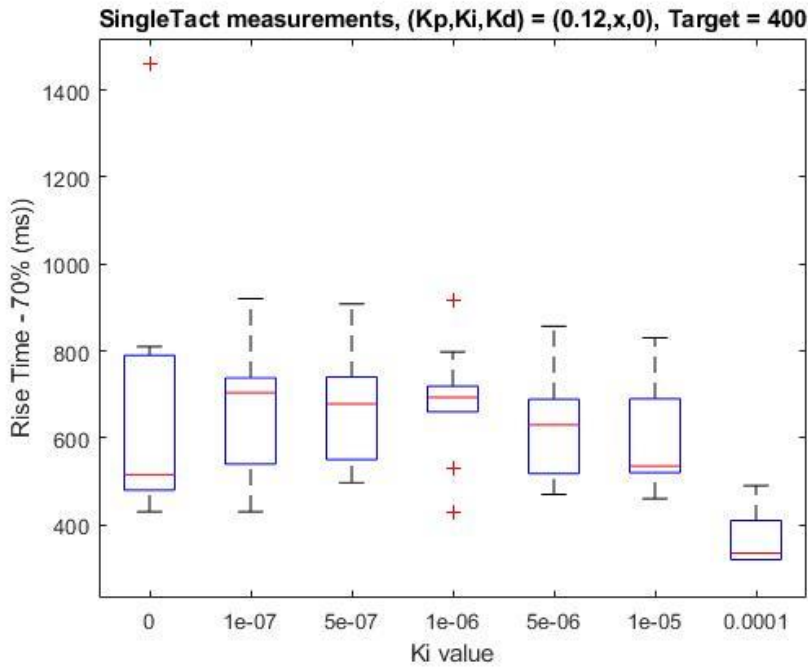




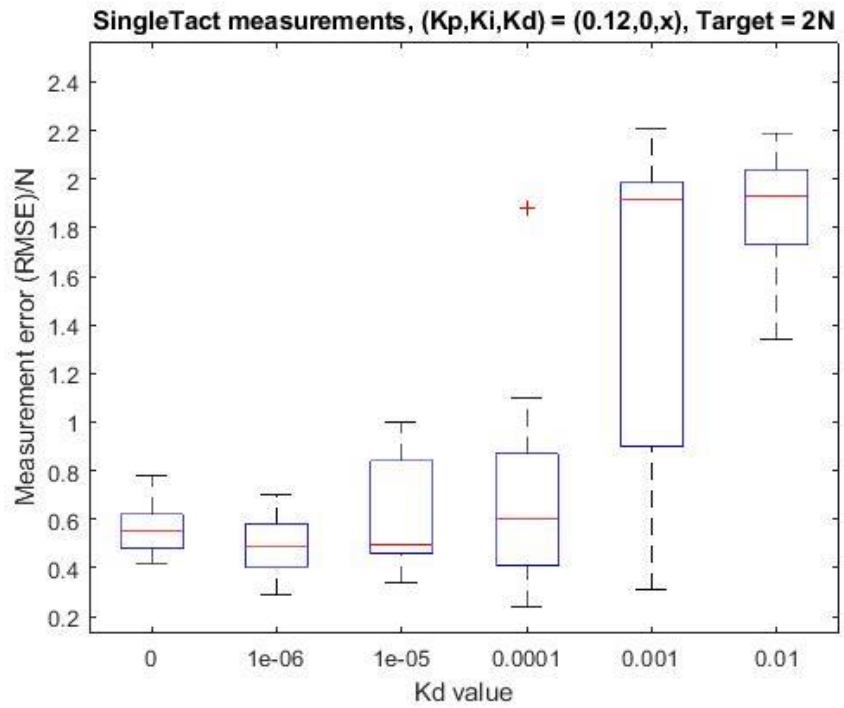
6.4.4 Integrative control Rise time: SingleTact

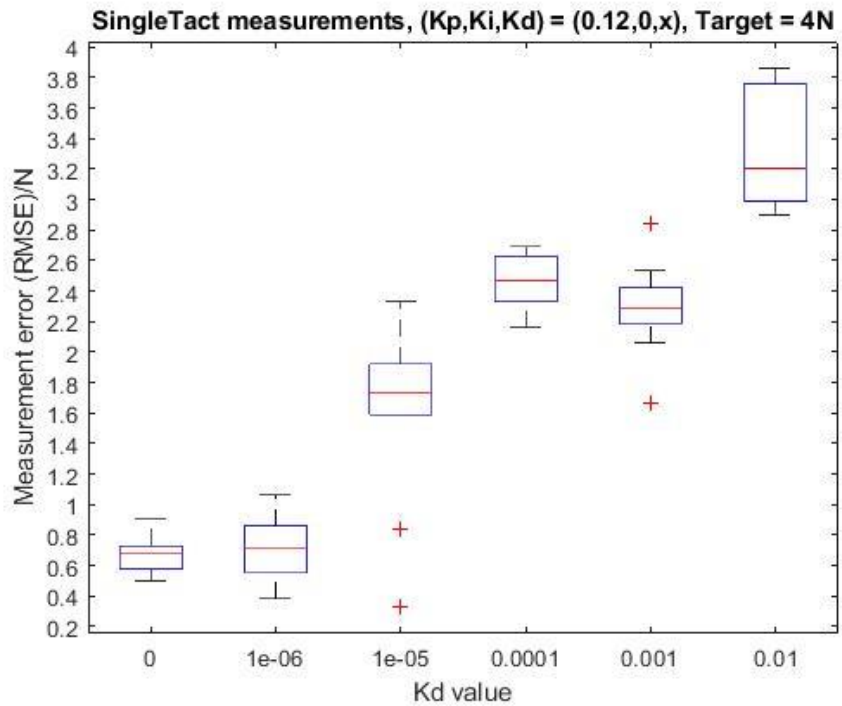
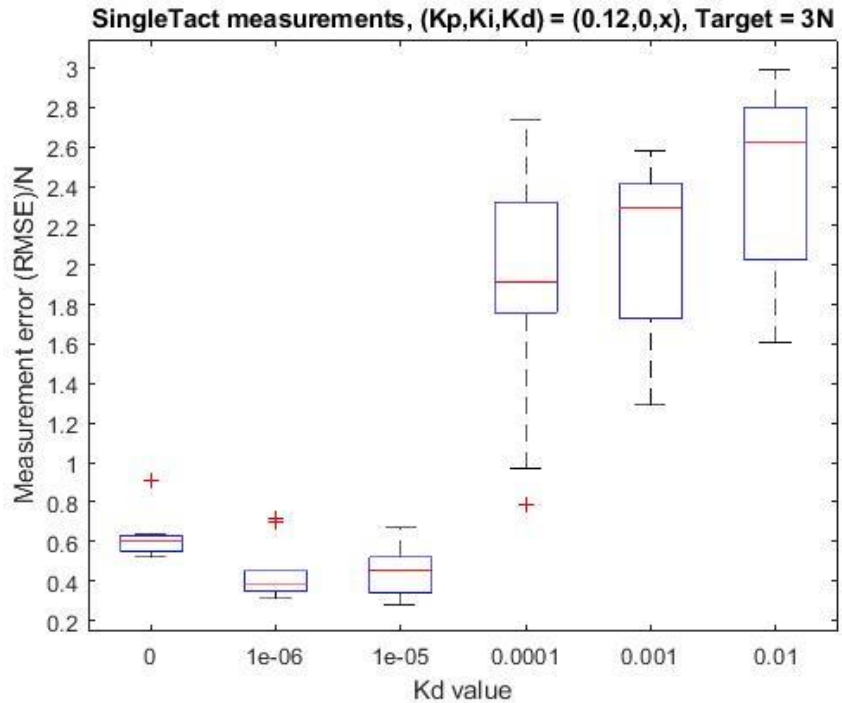




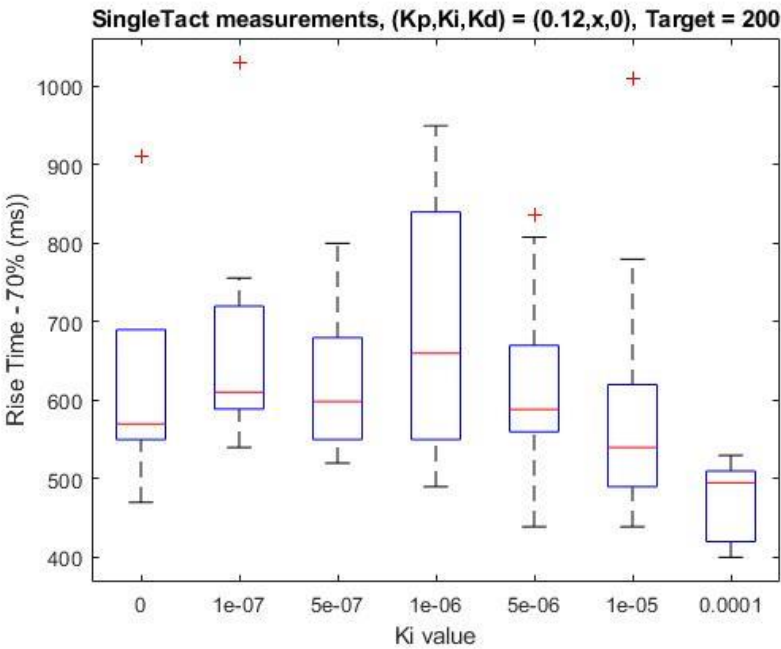


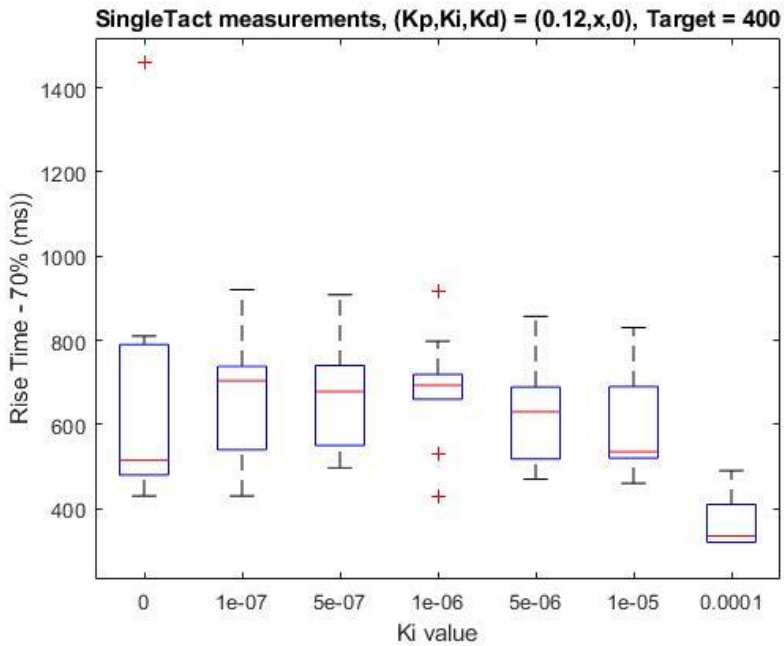
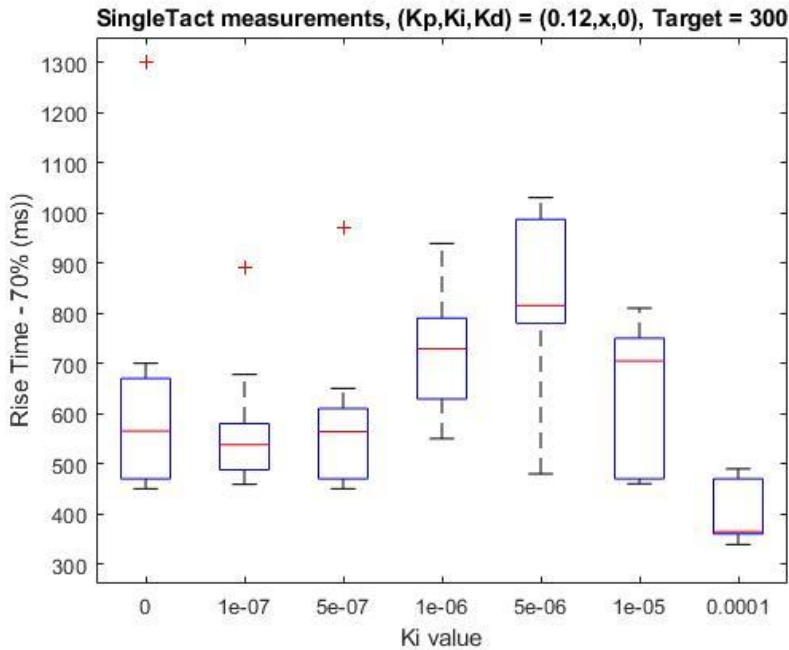
6.4.5 Derivative control RMSE: SingleTact





6.4.6 Derivative control Rise time: SingleTact





6.5 Appendix - Arduino Code

```

////////////////////Main modes////////////////////
// 0 = waiting for input, 1 = singletact, 2 = FSR, 3 = Strain gauge
int mode = 2;
//parameters for PID
//note max value for double is 32000
double output = 1500; //Starting value for control output
/*double kp = 0.03;
double ki = 0.00002; //0.0005
double kd = 0.0003;*/
double kp = 0.3; ///0.024
double ki = 0.0000000; //0.0000001
double kd = 0.0000000; //0.0000001
////////////////////Servo variables////////////////////
//Includes
#include <Servo.h>
//Defines
#define LINEARACTUATORPIN 9 //Linear Actuator Digital Pin
Servo LINEARACTUATOR; // create servo objects to control the linear
actuator
int MINVAL = 1050; //Lower range limit of value sent to servo (ms), 1050
int STARTVAL = 1050;
int MAXVAL = 2000; //upper range limit of value sent to servo (ms)
int MOTORSTEP = 3; //rate of change of motor for each increase "tick",
tested step1 and delay 20
boolean once = true;
int motorVal; //Current motorPositionVal (starts from STARTVAL)
long motorTime = 0; //when the motor last updated
long motorTimeInterval = 200; //millis, how often the motor should update
int targetPressure;
String motorCommandString; //received command from gui
String testString; //print helpful commands
////////////////////PID Parameters////////////////////
unsigned long currentTimeRegul, previousTimeRegul;
double elapsedTimeRegul;
double error;
double lastError;
double input, setPoint;
double cumError, rateError;
boolean disengaged = false;
int paramMax = 8; //max amount of values for parameters, ex: kp = 0.0000009
////////////////////FSR variables////////////////////
int fsrAnalogPin = 0; // FSR is connected to analog 0
int LEDpin = 9; // connect Red LED to pin 9 (PWM pin), LED used to
show FSR is on, remove //this?
int fsrReading; // the analog reading from the FSR resistor divider
int LEDbrightness;
////////////////////SingleTact variables////////////////////
#include <Wire.h> //For I2C/SMBus
////////////////////Main Setup////////////////////
void setup() {
    Serial.println("Start...");
    Serial.begin(9600); // We'll send debugging information via the Serial
monitor

```

```
//pinMode(LEDpin, OUTPUT); //Used to show FSR is on. Not important,
remove this?
initiateServo();
setupSingleTact();
// targetPressure = 30; //singletact test
//targetPressure = 300; //FSR test initial targetpressure
disengage(); //disengaged default
delay(500);
}
////////////////////Main
Loop////////////////////////////////////
void loop() {
    loopSingleTact();
    //sensorTest();
    //testLoop1();
    //testLoop2();
    Serial.print("Test String: ");
    Serial.println(testString);
    delay(100); // Change this if you are getting values too quickly
}
////////////////////Main help methods////////////////////////////////
void loopSingleTact() {
    readCommand();
    if (once)
        Serial.println("Unconnected"); //Just a boolean variable to test if GUI
can connect (false = connectedf)
    else
        Serial.println("Connected - str: " + motorCommandString);
    int data = readPressure();
    //send all parameters
    Serial.print("I2C Sensor Data:");
    Serial.print(data);
    Serial.print("\n");
    Serial.print("motorVal: ");
    Serial.println(motorVal);
    Serial.print("targetPressure: ");
    Serial.println(targetPressure);
    //updateMotorDynamic(data, 5);
    updateMotorPID(data, 2);
}
void updateMotor(int pressure, int marginZone) {
    if (pressure < targetPressure && motorVal < MAXVAL)
        motorVal += MOTORSTEP;
    if (pressure > targetPressure && motorVal > MINVAL)
        motorVal -= MOTORSTEP;
}
void updateMotorDynamic(int pressure, int marginZone) {
    if (disengaged)
        return;
    int data = pressure;
    int mStep;
    if (abs(data - targetPressure) > 33)
        mStep = 35;
    else if (abs(data - targetPressure) > 23)
        mStep = 25;
    else if (abs(data - targetPressure) > 13)
        mStep = 10;
    else mStep = 1;

    if (pressure < targetPressure && motorVal < MAXVAL)
        motorVal += mStep;
```

```
    if (pressure > targetPressure && motorVal > MINVAL)
        motorVal -= mStep;
    LINEARACTUATOR.writeMicroseconds(motorVal);
}

void updateMotorPID(int pressure, int marginZone) {
    int data = pressure;
    int mStep;
    if (disengaged) {
        LINEARACTUATOR.writeMicroseconds(STARTVAL);
        Serial.print("Disengaged, writing motor value: ");
        Serial.println(STARTVAL);
        return;
    }
    mStep = computePID(pressure);
    //send all parameters
    Serial.print("computePID: ");
    Serial.println(mStep);
    Serial.print("Parameters:");
    Serial.print(kp, paramMax);
    Serial.print(":");
    Serial.print(ki, paramMax);
    Serial.print(":");
    Serial.println(kd, paramMax);
    Serial.print("Target:");
    Serial.println(targetPressure);
    Serial.print("Mode:");
    Serial.print(mode);
    Serial.println("");
    Serial.println("-----");
    motorVal += mStep;
    if (motorVal > MAXVAL)
        motorVal = MAXVAL;
    if (motorVal < MINVAL)
        motorVal = MINVAL;

    LINEARACTUATOR.writeMicroseconds(motorVal);
}

//////////
//Read Serial, example string from java code
void readCommand() {
    char data[2];
    char dataKp[4];
    char dataKi[4];
    char dataKd[4];
    while (Serial.available() > 0 ) {
        char input = Serial.read();
        //Serial.print(input);
        //$ Reading motor command value with HOLD
        if (input == '$') {
            int i = 0;
            while (Serial.available()) {
                input = Serial.read();
                data[i] = input;
                i++;
            }
            if (data[0] == '0') {
                disengage();
                return;
            }
            disengaged = false;
            motorCommandString = String(data);
        }
    }
}
```

```
        targetPressure = motorCommandString.toInt();
    }
}
}
void disengage() {
    motorVal = (MINVAL+400);
    LINEARACTUATOR.writeMicroseconds(motorVal);
    delay(100);*/
    targetPressure = 0;
    disengaged = true;
    cumError = 0;
    motorVal = STARTVAL;
    LINEARACTUATOR.writeMicroseconds(motorVal);
}
//////////////////////////////// code////////////////////////////////////////
void changeMode(int i) {
    //Used for changing between sensor modes
    if (i < 3 && i > -1) {
        mode = i;
    }
}
int readPressure() {
    //returns pressure value depending on which sensor is selected
    //for now only FSR
    int pressure = 0;
    int singleTactCompensation = 255; //last save 238
    boolean singleTactCompOn = true;
    switch (mode) {
        case 1: //Singletact
            pressure = (int)readSingleTact();
            if (!singleTactCompOn)
                break;
            if (pressure > singleTactCompensation) //SingleTact has offset Error
                pressure -= singleTactCompensation;
            else
                pressure = 0;
            break;
        case 2: //FSR
            pressure = fsrRead();
            break;
    }
    return pressure;
}
////////////////////////////////Regulator Control
Code////////////////////////////////////////
//Original Source: https://www.teachmemicro.com/arduino-pid-controlfs-tutorial/
double computePID(double inp) {
    setPoint = targetPressure;
    currentTimeRegul = millis();
    elapsedTimeRegul = (double)(currentTimeRegul - previousTimeRegul);
    //time elapsed from previous computation
    error = setPoint - inp;
    // find error
    cumError += error * elapsedTimeRegul; // compute
    // integrative part
    rateError = (error - lastError) / elapsedTimeRegul; // compute
    // derivative part
    lastError = error;
    //save current error
}
```

```
    previousTimeRegul = currentTimeRegul;                                //save
current time
    if (ki == 0)
        cumError = 0;
    double out = kp * error + ki * cumError + kd * rateError;
//OUTPUT for PID regulator
    return out;
}
////////////////////////////////////Servo control
code////////////////////////////////////
void initiateServo1() {
    //initialize servo/linear actuator objects
    LINEARACTUATOR.attach(LINEARACTUATORPIN, MINVAL, MAXVAL);          //
attaches/activates //the linear actuator as a servo object
    //use the writeMicroseconds to set the linear actuators to their default
positions
    motorVal = STARTVAL;
    LINEARACTUATOR.writeMicroseconds(motorVal);
}
void testLoop1() {
    //moves the servo back and forth
    Serial.println("-----");
    int time1 = 4000;
    int linearValue = 1300;
    Serial.println("Low");
    LINEARACTUATOR.writeMicroseconds(linearValue);
    delay(time1);
    linearValue = 1800;
    Serial.println("High");
    LINEARACTUATOR.writeMicroseconds(linearValue);
    delay(time1);
}
void testLoop2() {
    motorVal = MINVAL;
    while (motorVal < MAXVAL) {
        motorVal += 1;
        LINEARACTUATOR.writeMicroseconds(motorVal);
        Serial.print("motorVal: ");
        Serial.println(motorVal);
        delay(50);
    }
}
////////////////////////////////////Sensors////////////////////////////////////
////////////////////////////////////
////////////////////////////////////FSR
Code////////////////////////////////////
int fsrRead() {
    //FSR reading functionality
    //Note that "Sensor Data:" is a key word for gui
    fsrReading = analogRead(fsrAnalogPin);
    Serial.print("FSR Sensor1 Data: ");
    Serial.print(fsrReading);
    Serial.print("\n");
    Serial.print("Sensor Data: ");
    int mappedVal = convertToNewton1(fsrReading);
    Serial.print(mappedVal);
    Serial.print("\n");

    delay(20); // Change this if you are getting values too quickly orig=50
    return mappedVal;
}
```

```
void sensorTest() {
  int data = readPressure();;
  Serial.print("Sensor Data:");
  Serial.print(data);
  Serial.print("\n");
  delay(200);
}

int convertToNewton1(int reading) {
  //(FSR circuit) Maps measured voltage from voltage divider to Newton
  value obtained from //callibration with Mark-10 force gauge
  //Note: out has been multiplied by 100 to solve double integer problem,
  divide by 100 to receive newton
  int in[] = {0, 67, 228, 311, 370, 423, 439, 465, 467, 469, 512, 537, 548,
573, 585, 600, 627, 651, 667, 676, 692, 706, 715, 718, 761, 771, 780, 788,
796, 802, 820, 839, 858, 877, 889, 900, 909, 914, 921, 928, 931, 933};
  int out[] = {26, 66, 174, 230, 326, 390, 416, 454, 492, 508, 592, 674,
732, 814, 882, 934, 1002, 1084, 1140, 1202, 1312, 1378, 1518, 1584, 1970,
2034, 2156, 2119, 2370, 2540, 3010, 3642, 4200, 5010, 5678, 6376, 6794,
7184, 7556, 8230, 8512, 8956};
  //Comment: multimap is not made for doubles, below code shows original
  values which wont //work
  //int in[] = {0, 67, 228, 311, 370, 423, 439, 465, 467, 469, 512, 537,
548, 573, 585, 600, 627, 651, 667, 676, 692, 706, 715, 718, 761, 771, 780,
788, 796, 802, 820, 839, 858, 877, 889, 900, 909, 914, 921, 928, 931, 933};
  //int out[] = {0.26, 0.66, 1.74, 2.3, 3.26, 3.9, 4.16, 4.54, 4.92, 5.08,
5.92, 6.74, 7.32, 8.14, 8.82, 9.34, 10.02, 10.84, 11.4, 12.02, 13.12,
13.78, 15.18, 15.84, 19.7, 20.34, 21.56, 21.19, 23.7, 25.4, 30.1, 36.42,
42, 50.1, 56.78, 63.76, 67.94, 71.84, 75.56, 82.3, 85.12, 89.56};
  return multiMap(reading, in, out, 42);
}

int multiMap(int val, int* _in, int* _out, uint8_t size)
//This code received from internet: for multimap
// note: the _in array should have increasing values
//Source: http://playground.arduino.cc/Main/MultiMap
{
  // take care the value is within range
  // val = constrain(val, _in[0], _in[size-1]);
  if (val <= _in[0]) return _out[0];
  if (val >= _in[size - 1]) return _out[size - 1];

  // search right interval
  uint8_t pos = 1; // _in[0] allready tested
  while (val > _in[pos]) pos++;
  // this will handle all exact "points" in the _in array
  if (val == _in[pos]) return _out[pos];
  // interpolate in the right segment for the rest
  return (val - _in[pos - 1]) * (_out[pos] - _out[pos - 1]) / (_in[pos] -
_in[pos - 1]) + _out[pos - 1];
}
////////////////////////////////////
////
////////////////////////////////////SingleTact////////////////////////////////////
void setupSingleTact() {

  Wire.begin(); // join i2c bus (address optional for master)
  //TWBR = 12; //Increase i2c speed if you have Arduino MEGA2560, not
suitable for Arduino UNO
  Serial.begin(57600); // start serial for output
  Serial.flush();
  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
```



```
    }
    Serial.println("PPS UK: SingleTact sensor value in PSI. \n(resembles PC
executable display)");
    Serial.println("Refer manual for any other calculation.");
    Serial.println("-----");
}
short readSingleTact() {
    byte i2cAddress = 0x04; // Slave address (SingleTact), default 0x04
    short data = readDataFromSensor(i2cAddress);
    return data;
}
short readDataFromSensor(short address)
{
    byte i2cPacketLength = 6; // i2c packet length. Just need 6 bytes from each
slave
    byte outgoingI2CBuffer[3]; // outgoing array buffer
    byte incomingI2CBuffer[6]; // incoming array buffer

    outgoingI2CBuffer[0] = 0x01; // I2c read command
    outgoingI2CBuffer[1] = 128; // Slave data offset
    outgoingI2CBuffer[2] = i2cPacketLength; // require 6 bytes

    Wire.beginTransmission(address); // transmit to device
    Wire.write(outgoingI2CBuffer, 3); // send out command
    byte error = Wire.endTransmission(); // stop transmitting and check slave
status
    if (error != 0) return -1; // if slave not exists or has error, return -1
    Wire.requestFrom(address, i2cPacketLength); // require 6 bytes from slave
    byte incomeCount = 0;
    while (incomeCount < i2cPacketLength) // slave may send less than
requested
    {
        if (Wire.available())
        {
            incomingI2CBuffer[incomeCount] = Wire.read(); // receive a byte as
character
            incomeCount++;
        } else
        {
            delayMicroseconds(10); // Wait 10us
        }
    }
    short rawData = (incomingI2CBuffer[4] << 8) + incomingI2CBuffer[5]; // get
the raw data
    return rawData;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

6.6 Appendix - Java Code

6.6.1 *ReaderLine Class*

```

/*
 * To change this license header, choose License Headers in Project
 * Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javaguarduino;
import com.fazecast.jSerialComm.SerialPort;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import static java.lang.Math.abs;
import static java.lang.Math.sqrt;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JSlider;
import javax.swing.JTextField;
public class ReaderLine implements Runnable {
    //////////////////////////////////////////////////Main modes//////////////////////////////////////
    // 0 = waiting for input, 1 = singletact, 2 = FSR, 3 = Strain gauge
    public static int mode;
    private static boolean isOn = false;

    private SerialPort comPort;
    private InputStream in;
    private List<Double> scores;

    private List<Double> savedVals; //for sensor data
    private List<Double> savedValsError; //for measuring suceesrate
    private static final int errorSize = 20; //how many vals to save for
    RMSE calculation

    private List<Double> savedMotorVals;
    private GraphPanel gp, gp2;
    private BufferedReader bf;
    private JTextField j1, j2, j3, jtjf, jtjf_success, jtjf_riseTime;
    //Jtextfields for pid parameters, target pressure and success rate
    respectively
    private JSlider slider;
    private JButton btnHold;
    private double kp, ki, kd;
    private int target = -1;
    private boolean sliderInitiated = false;
    public static boolean sliderTextEnabled = true;
    private static boolean clearError = false;
    private List<Integer> input;
    public static boolean targetInitiated = false; //check if target
    pressure slider should be updated
    private static long lastTimePressed = 0;
    public static boolean iniParameters = false; //Check if obtained pid
    parameters from arduinn when init of program
    public final int FSRMAX = 600, SINGLETACTMAX = 100, MINIMUMSENSORVAL =
    30; //minimum reliable value from sensor;

```

```
//////////
private int listSize = 100; //Amount of lines read before update gui.
//standard value 100
//Pulse feature
private static long pulseStart = 0;
private static int pulseLength = 8000;
private static boolean pulseOn = false;
private static boolean disengaged = false; //If the servo should be
disengaged
//private static boolean pulseOnLast = false; //this is used to
calculate last two seconds "when pulse returns to zero"
//private static long pulseLastBegin = 0;
private final boolean doDisengageAfterPulse = true;
//rise time feature
public static boolean riseOn = false;
private static long riseBegin;
public static boolean RMSenabled = false; //updating the rms stats
public ReaderLine(SerialPort cp, GraphPanel gp, GraphPanel gp2,
    JTextField j1, JTextField j2, JTextField j3, JTextField jtf,
    JSlider slider, JTextField jtf_success, JTextField
jtf_riseTime, JButton btnHold) throws IOException {
    System.out.println("Reader started....");
    this.comPort = cp;
    this.gp = gp;
    this.gp2 = gp2;
    comPort.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING,
100, 0);
    in = comPort.getInputStream();
    scores = new ArrayList();
    toggle(); //start reader first time
    savedVals = new ArrayList();
    savedValsError = new ArrayList();
    savedMotorVals = new ArrayList();
    bf = new BufferedReader(new InputStreamReader(in));
    this.j1 = j1;
    this.j2 = j2;
    this.j3 = j3;
    this.jtf = jtf;
    this.slider = slider;
    this.jtf_success = jtf_success;
    this.jtf_riseTime = jtf_riseTime;
    this.btnHold = btnHold;
}
@Override
public void run() {
    while (true) {
        updateSliderText();
        if (checkIfPulseEnd()) {
            endPulseAction();
        }
        if (clearError) {
            savedValsError.clear();
            clearError = false;
        }
        try {
            read2();
        } catch (IOException ex) {
            //Logger.getLogger(ReaderLine.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```
}
private void read2() throws IOException {
    String line = null;
    while ((line = bf.readLine()) != null) {
        appendLine(line);
    }
}
public void close() throws IOException {
    in.close();
}
public static void toggle() {
    if (isOn) {
        toggleOff();
    } else {
        toggleOn();
    }
}
public static void toggleOn() {
    ReaderLine.isOn = true;
    System.out.println("Reader is on");
}
public static void toggleOff() {
    ReaderLine.isOn = false;
    System.out.println("Reader is off");
}
public static boolean checkIsOn() {
    return ReaderLine.isOn;
}
public List<Double> getScores() {
    return scores;
}
private void appendLine(String line) {
    /*if (!line.contains("I2C Sensor Data")) {
        return;
    }*/
    if (line.contains("Sensor Data")) {
        appendLineSensor(line);
    }
    if (line.contains("Parameters")) {
        appendLineParameterData(line);
    }
    if (line.contains("motorVal")) {
        appendLineMotor(line);
    }
    if (line.contains("Target")) {
        appendLineTarget(line);
    }
    if (line.contains("Mode")) {
        appendLineMode(line);
    }
}
private void appendLineMotor(String line) {
    try {
        String[] str = line.split(":");
        Double nbr = Double.parseDouble(str[1]);
        if (savedMotorVals.size() > listSize) {
            //savedVals = new ArrayList();
            savedMotorVals.remove(0);
        }
        if (checkIsOn()) {
            System.out.println("Saved motor value: " + nbr);
        }
    }
}
```

```
    }
    if (ReaderLine.checkIsOn()) {
        savedMotorVals.add(nbr);
        updateGP2();
    }
} catch (Exception e) {
    System.err.print("Motor not attached");
}
}
private void appendLineSensor(String line) {
    try {
        String[] str = line.split(":");
        Double nbr = Double.parseDouble(str[1]);
        if (savedVals.size() > listSize) {
            //savedVals = new ArrayList();
            savedVals.remove(0);
        }
        if (checkIsOn()) {
            System.out.println("Saved value: " + nbr);
        }
        //unit conversion
        /*if (Arduino.CONVERSION) {
            nbr = nbr * 0.689476; //Conversion from PSI to Newton/cm^2
            System.out.println("Conversion from PSI to Newton/cm^2 is
on");
        }*/
        if (ReaderLine.checkIsOn()) {
            savedVals.add(nbr);
            updateSuccessRate(nbr);
            updateGP();
            if (checkIfRiseTimeReached(nbr) && riseOn) {
                saveRiseTime();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void appendLineParameterData(String line) {
    try {
        if (iniParameters) {
            return;
        }
        String[] str = line.split(":");
        System.out.println("Read value: " + str[1]);
        if (str[1].equals("")) {
            return;
        }
        kp = Double.parseDouble(str[1]);
        ki = Double.parseDouble(str[2]);
        kd = Double.parseDouble(str[3]);
        updateParameters();
        iniParameters = true;

        //System.out.println("Initialized control parameters: " + "kp:
" + kp + ", ki: " + ki + ", kd: " + kd);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void updateGP() {
```

```
        gp.setScores(savedVals);
    }
    private void updateGP2() {
        gp2.setScores(savedMotorVals);
    }
    private void updateParameters() {
        System.out.println("set parameters(kp,ki,kd) to: " + kp + ", " + ki
+ ", " + kd);
        String skp = String.format("%.7f", kp);
        String ski = String.format("%.7f", ki);
        String skd = String.format("%.7f", kd);
        j1.setText(skp);
        j2.setText(ski);
        j3.setText(skd);
        //Note double is more accurate than float
    }
    private void appendLineTarget(String line) {
        if (targetInitiated) {
            return;
        }
        if (System.currentTimeMillis() - lastTimePressed < 100) {
            //Gives some time for the arduino to respond before updating
target force value
            return;
        }
        String[] str = line.split(":");
        String str2 = str[1];
        target = Integer.parseInt(str2);
        jtf.setText(str2);
        System.out.println("set target pressure in textfield to: " + str2);
        System.out.println("-----");
        targetInitiated = true;
    }

    private void appendLineMode(String line) {
        String[] str = line.split(":");
        mode = Integer.parseInt(str[1]);

        if (sliderInitiated) {
            return;
        }
        switch (mode) {
            case 0:
                toggleOff();
                break;
            case 1:
                //updateSlider(SINGLETRACTMAX, 0, target, 1, 10);
                //Arduino.maxAcceptedVal = SINGLETRACTMAX;
                //toggleOn();
                // break;
            case 2:
                //updateSlider(FSRMAX, 0, target, 50, 100);
                updateSlider(FSRMAX, 0, target, 50, 100);
                Arduino.maxAcceptedVal = FSRMAX;
                toggleOn();
                break;
        }
    }
    private void updateSlider(int max, int min, int currentValue, int
minTick, int maxTick) {
```

```
        slider.setMaximum(max);
        slider.setMinimum(min);
        slider.setValue(currentValue);
        slider.setMajorTickSpacing(maxTick);
        slider.setMinorTickSpacing(minTick);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        slider.setSnapToTicks(true);
        sliderInitiated = true;
    }
    //This will calculate the RMS error
    private void updateSuccessRate(double data) {
        if (!RMSenabled) {
            return;
        }
        savedValsError.add(data - target);
        double success = 0, meanErr = 0;
        double cum = 0;
        double temp;
        for (double val : savedValsError) {
            temp = val * val;
            cum += temp;
        }
        meanErr = cum / savedValsError.size();
        success = sqrt(meanErr);
        jtf_success.setText(success + "");
        System.out.println("Target value: " + target);
        System.out.println("Observed value: " + data);
    }
    public static void targetInitiate() {
        lastTimePressed = System.currentTimeMillis();
        targetInitiated = false;
    }

    private void setForceZero() {
        jtf.setText("0");
        target = 0;
        btnHold.doClick();
    }

    public static void startPulse() {

        pulseStart = System.currentTimeMillis();
        pulseOn = true;
        riseTimerBegin();
        System.err.println("pulse began");

        if (!RMSenabled) {
            RMSenabled = true;
            clearError = true;
        }
    }

    private boolean checkIfPulseEnd() {
        return System.currentTimeMillis() - pulseStart > pulseLength &&
        pulseOn;
    }
    private void endPulseAction() {
        if (!pulseOn) {
            return;
        }
    }
}
```

```
String old = target + "";
target = 0;

//disengage action
if (doDisengageAfterPulse) {
    disengaged = true;
    jtf.setText("0");
    btnHold.doClick();
    jtf.setText(old);
    disengaged = false;
}
if (riseOn) {
    saveRiseTime();
}
System.err.println("Triggered checkpulsend");
pulseStart = 0;
pulseOn = false;
savedValsError.clear();
RMSEnabled = false;
}

private static void riseTimerBegin() {
    riseOn = true;
    riseBegin = System.currentTimeMillis();
}
private boolean checkIfRiseTimeReached(double data) {
    double rise = 0.7 * target;
    if (data > rise && target > 0 && data > MINIMUMSENSORVAL) {
        System.out.println("-----");
        System.out.println("Triggered riseTime, data: " + data + ",
rise number: " + rise);
        return true;
    }
    if (System.currentTimeMillis() - riseBegin > pulseLength) {
        return true;
    }
    return false;
}
private void saveRiseTime() {
    if (riseOn) {
        long time = System.currentTimeMillis() - riseBegin;
        jtf_riseTime.setText(time + "");
        riseOn = false;
    }
}
private void updateSliderText() {
    if (sliderTextEnabled) {
        jtf.setText(slider.getValue() + "");
    }
}
}
```


6.6.2 Main Class

```
package javaguiarduino;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import com.fazecast.jSerialComm.SerialPort;
import java.awt.Color;
import java.awt.Dimension;
import java.io.PrintWriter;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.Insets;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.io.InputStream;
import java.text.NumberFormat;

import java.util.ArrayList;

import java.util.List;
import java.util.Random;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JSlider;
import javax.swing.JTextField;

import javax.swing.border.LineBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.NumberFormatter;
import javax.swing.text.PlainDocument;

/**
 * CLASS:      This class will launch the GUI.
 */
public class Arduino extends JFrame {
    private String comPortName; //starting comport
    private String[] comPortNames = new String[]{"COM3", "COM5", "COM9",
"COM10"};
    private int startComIndex = 2;
    private int comIndex = 0;
    private JPanel contentPane;
    static final int S_MIN = 0;
    static final int S_MAX = 100;
    static final int S_INIT = 30;    //initial slider value
    final int resetTime = 1000; //ms, 1000 lowest confirmed value that
works
    private JTextField jtf_success, jtf_riseTime;
```

```
private static final int XPIX = 600, YPIX = 600; //Minimum
size of main frame
private GraphPanel gp, gp2;
private boolean readOn = false; //If read button is pressed
private boolean connected = false;
private boolean ReaderIsOn = false;
private boolean autoRunReader = true;
//Input panel
public static int maxAcceptedVal = 100; //only 100 for singletact, 500
for FSR
public static int minAcceptedVal = 0;
private final String inputErrorString = "NaN", inputOutOfBounds =
">Bounds";
private final int defaultInputVal = 30;
private final int defaultInputCharLimit = 5;
private JButton btnHold, btnReset;
//Graphs
private String graph2Description = "Motor Position ( $\mu$ s)";
//jtextfields
JTextField jt_kp, jt_ki, jt_kd, jt_f;
JSlider slider;
String kp, ki, kd;
int PIDdecimals = 7; //decimals for above parameters
public static boolean holdOn = false;
/**
 * convert from PSI to Newton/cm^2
 */
public static final boolean CONVERSION = false;
/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Arduino frame = new Arduino();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
/**
 * Create the GUI FRAME
 */
public Arduino() {
    setVisible(true);
    setTitle("Arduino - Servo Control");
    //Choose what happens when the 'x' is clicked.
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //Set the size of the GUI.
    //setBounds(100, 100, 246, 162);
    setSize(1300, 700);

    //Create a new contentPane and set the borders
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();
```

```
setMinimumSize(new Dimension(XPIX, YPIX));
setResizable(true);
setAlwaysOnTop(true);

/////Init logo///
Image frameImage = new ImageIcon("pic.png").getImage();
setIconImage(frameImage);
//////////
//Top Panel: Status + motorpanel
JPanel topPanel = new JPanel();
topPanel.setLayout(new GridBagLayout());
//topPanel.setBorder(new LineBorder(Color.BLACK));
//////////
// Status Panel
//Add the status JLabel
JPanel pnlStatus = new JPanel();
JLabel ledStatus = new JLabel("•");
ledStatus.setForeground(Color.RED);
pnlStatus.add(ledStatus);
JLabel lblStatus = new JLabel("Status: Unconnected");
lblStatus.setFont(new Font("Tahoma", Font.PLAIN, 18));
pnlStatus.add(lblStatus);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
topPanel.add(pnlStatus, c);
//////////
//////////Connection panel//////////
JPanel panel = new JPanel();
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 1;
topPanel.add(panel, c);

//Add connect button.
JButton btnConnect = new JButton("Connect");
panel.add(btnConnect);

//Add disconnect button.
JButton btnDisconnect = new JButton("Disconnect");
panel.add(btnDisconnect);

//Add JLabel to describe COM port
//JButton btnComPort = new JButton(comPortName);
JComboBox btnComPort = new JComboBox(comPortNames);
btnComPort.setSelectedIndex(startComIndex); //starting index
comPortName = comPortNames[startComIndex];
panel.add(btnComPort);

//Second Panel.
/////Motor Panel -----Slider-----////////
JPanel panel_2 = new JPanel();
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 2;
c.gridy = 1;
c.insets = new Insets(10, 25, 0, 0);
topPanel.add(panel_2, c);
JLabel jl2 = new JLabel("Motor position: ");
panel_2.add(jl2, c);
slider = new JSlider(JSlider.HORIZONTAL,
    S_MIN, S_MAX, S_INIT);
```

```
/* //Turn on labels at major tick marks.
slider.setMajorTickSpacing(10);
slider.setMinorTickSpacing(5);
slider.setPaintTicks(true);
slider.setPaintLabels(true);*/
panel_2.add(slider);

jtf = new JTextField(defaultInputVal + "", defaultInputCharLimit);
panel_2.add(jtf);
btnHold = new JButton("Hold");
JButton btnPulse = new JButton("Pulse");
btnReset = new JButton("Reset");

panel_2.add(btnHold);
panel_2.add(btnPulse);
panel_2.add(btnReset);

JLabel jbl_success = new JLabel("RMSE: ");
jtf_success = new JTextField();
jtf_success.setColumns(6);

panel_2.add(jbl_success);
panel_2.add(jtf_success);

JLabel jbl_riseTime = new JLabel("Rise Time (ms): ");
jtf_riseTime = new JTextField();
jtf_riseTime.setColumns(6);

panel_2.add(jbl_riseTime);
panel_2.add(jtf_riseTime);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
contentPane.add(topPanel, c);

////////////////////
//Graph window 1 - force diagram
JPanel gPanel = new JPanel();
gPanel.setLayout(new GridBagLayout());

//Panel 1
JPanel gPanel1 = new JPanel();
gPanel1.setLayout(new GridBagLayout());
gPanel1.setBorder(new LineBorder(Color.BLACK));
gp = new GraphPanel(getZeroes());
gp.setPreferredSize(new Dimension(XPIX - 50, YPIX - 250));
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
JLabel jlblGraph = new JLabel("Sensor Data Graph,          Newton x
100");

if (ReaderLine.mode == 1) {
    jlblGraph.setText("Sensor Data Graph,          Newton");
}
gPanel1.add(jlblGraph, c);

c.gridy = 1;
gPanel1.add(gp, c);

//Graph Command panel
```

```
JPanel pnlGraph = new JPanel();

//Graph radio button00
JButton btn_settings = new JButton("Settings");
c.fill = GridBagConstraints.HORIZONTAL;
pnlGraph.add(btn_settings);

//Add pause/resume button.
JButton btnPauseReader = new JButton("Pause/Resume");
btnPauseReader.setEnabled(false);
pnlGraph.add(btnPauseReader);

//Add Read button.
//JButton btnClearGraph = new JButton("Clear");
JButton btnRunReader = new JButton("Read");
btnRunReader.setEnabled(false);
pnlGraph.add(btnRunReader);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 2;
gPanel1.add(pnlGraph, c);

///////// Panel 2
gp2 = new GraphPanel(getZeroes());
gp2.setPreferredSize(new Dimension(XPIX - 50, YPIX - 250));
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
JPanel gPanel2 = new JPanel();
gPanel2.setLayout(new GridBagLayout());
gPanel2.setBorder(new LineBorder(Color.BLACK));

JLabel jlblGraph2 = new JLabel(graph2Description);
gPanel2.add(jlblGraph2, c);
c.gridy = 1;
gPanel2.add(gp2, c);
/////Regulator parameter control buttons
JPanel pnlGraph2 = new JPanel();
JLabel jl_gmo = new JLabel("Regulator: ");
pnlGraph2.add(jl_gmo);

JLabel lbl_kp = new JLabel("Kp");
JLabel lbl_ki = new JLabel("Ki");
JLabel lbl_kd = new JLabel("Kd");

jt_kp = new JTextField("", PIDdecimals); //need to fit number of
decimals + "0." characters
jt_ki = new JTextField("", PIDdecimals);
jt_kd = new JTextField("", PIDdecimals);

pnlGraph2.add(lbl_kp);
pnlGraph2.add(jt_kp);
pnlGraph2.add(lbl_ki);
pnlGraph2.add(jt_ki);
pnlGraph2.add(lbl_kd);
pnlGraph2.add(jt_kd);
//Add pause/resume button.
JButton btn_r_apply = new JButton("Apply");
btn_r_apply.setVisible(false);
//btn_r_apply.setEnabled(false);
pnlGraph2.add(btn_r_apply);
```

```
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 2;
gPanel2.add(pnlGraph2, c);
//////////
//Add to gPanel and finialize
c.gridx = 0;
c.gridy = 0;
gPanel.add(gPanel1, c);
c.gridx = 1;
c.gridy = 0;
gPanel.add(gPanel2, c);
c.gridx = 0;
c.gridy = 1;
contentPane.add(gPanel, c);
//////////
//Listener for connect button.
btnConnect.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        if (connected) {
            return; //check if already connected
        }
        //Set up the values for the USB port.
        comPort = SerialPort.getCommPort(comPortName);
        comPort.setBaudRate(baudRate);
        //If the port is not closed, open the USB port.
        if (comPort.isOpen() == false) {
            try {
                //Open the USB port and initialize the PrintWriter.
                comPort.openPort();
                //Thread.sleep(1000);
                Thread.sleep(500);
                outPut = new
PrintWriter(comPort.getOutputStream());
                //Input reader thread
            } catch (Exception c) {
                System.err.println("Error connecting...");
                System.out.println("Error opening port.");
                lblStatus.setText("Status: Opening USB failed");
                ledStatus.setForeground(Color.BLUE);
                return;
            }
            //Update the console and status.
            System.out.println("Connection to Arduino successful. "
+ comPortName);

            lblStatus.setText("Status: Connected " + comPortName);
            ledStatus.setForeground(Color.GREEN);
            connected = true;
            btnRunReader.setEnabled(true);
            if (autoRunReader) {
                btnRunReader.doClick();
            }
        } else {
            //If the port couldn't be opened print out to the
console.

            System.out.println("Error opening port.");
            lblStatus.setText("Status: Opening USB failed");
            ledStatus.setForeground(Color.BLUE);
        }
    }
}
```

```
});

//Listener for the disconnect button.
btnDisconnect.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent arg0) {
        //Close the USB port if it's open.
        if (comPort.isOpen() == true) {
            //Close the port and update the console/status.
            comPort.closePort();
            System.out.println("Disconnected from Arduino.");
            lblStatus.setText("Status: Disconnected");
            ledStatus.setForeground(Color.RED);
            connected = false;
        }
        //reset items
        btnRunReader.setEnabled(false);
        btnPauseReader.setEnabled(false);
    }
});

// Update Graph Button
btnPauseReader.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //gp.setScores(getScores());
        ReaderLine.toggleOn();
    }
});

// read button
btnRunReader.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        startReader();
        btnPauseReader.setEnabled(true);
    }
});

// Switch comport button
btnComPort.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        comIndex++;
        if (comIndex >= comPortNames.length) {
            comIndex = 0;
        }
        JComboBox cb = (JComboBox) e.getSource();
        String itemName = (String) cb.getSelectedItem();
        System.out.println("Selected: " + itemName);
        comPortName = itemName;

        //btnComPort.setText(comPortNames[comIndex]);
        //comPortName = comPortNames[comIndex];
    }
});

// motor force HOLD button, example code: $12 which is 12 Newton
btnHold.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String nbr = jtf.getText();
        if (isNumeric(nbr)) {
            int nbr2 = Integer.parseInt(nbr);

```

```
        if (nbr2 <= maxAcceptedVal && nbr2 >= minAcceptedVal) {
            //Hold button action
            outPut.write("$" + nbr);
            outPut.flush();
            System.err.println("Button pressed: " + nbr);
            ReaderLine.iniParameters = false;
            ReaderLine.targetInitiate();
            ReaderLine.toggleOn();
            //ReaderLine.RMSenabled = true;
            if (!nbr.equals("0")) {
                jtf_riseTime.setText("-");
            } else {
                ReaderLine.riseOn = false;
            }
            holdOn = true;
        } else {
            jtf.setText(inputOutOfBounds);
            holdOn = false;
        }
    }
} else {
    jtf.setText(inputErrorString);
    holdOn = false;
}
});
// motor force PULSE apply button, example code: M12#, which is 12
Newton
btnPulse.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        btnHold.doClick();
        ReaderLine.startPulse();
    }
});
btnReset.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ReaderLine.sliderTextEnabled = false;
        ReaderLine.RMSenabled = false;
        ReaderLine.disengaged = true;
        ReaderLine.riseOn = false;
        outPut.write("$500");
        outPut.flush();
        Long time = System.currentTimeMillis();
        while (System.currentTimeMillis() - time < resetTime) {
        }
        //Hold button action
        outPut.write("$0");
        outPut.flush();
        while (System.currentTimeMillis() - time < 100) {
        }
        ReaderLine.sliderTextEnabled = true;
    }
}
});
btn_r_apply.addActionListener(
    new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e
        ) {
            /*kp = jt_kp.getText();
```



```
        ki = jt_ki.getText();
        kd = jt_kd.getText();*/

        kp = formatTextDecimals(PIDdecimals, jt_kp.getText());
        ki = formatTextDecimals(PIDdecimals, jt_ki.getText());
        kd = formatTextDecimals(PIDdecimals, jt_kd.getText());

        if (holdOn = true) {
            btnHold.doClick();
        }
    }
};
slider.addChangeListener(
    (ChangeEvent e) -> {
        jtf.setText(slider.getValue() + "");
    }
);
//Window Listener
addWindowListener(
    new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e
        ) {
            try {
                resetPosition();
            } catch (Exception ex) {
            }
            System.exit(0);
        }
    }
);
}
private void startReader() {
    if (ReaderIsOn) //check if already on
    {
        return;
    }
    try {
        //first graph
        final Runnable runnable = new ReaderLine(comPort, gp, gp2,
jt_kp, jt_ki, jt_kd, jtf, slider, jtf_success, jtf_riseTime, btnHold);
        final Thread thread = new Thread(runnable);
        thread.start();
        ReaderIsOn = true;
    } catch (IOException ex) {
        System.err.println("Couldn't start reader (inputstream)");
    }
}

private List<Double> getSampleScores() {
    List<Double> scores = new ArrayList<>();
    Random random = new Random();
    int maxDataPoints = 40;
    int maxScore = 10;
    for (int i = 0; i < maxDataPoints; i++) {
        scores.add((double) random.nextDouble() * maxScore);
//        scores.add((double) i);
    }
    return scores;
}
```

```
private List<Double> getZeroes() {
    List<Double> scores = new ArrayList<>();
    int maxDataPoints = 40;
    for (int i = 0; i < maxDataPoints; i++) {
        scores.add(0.0);
    }
    return scores;
}
//Help method, use to check if a string is a number
public static boolean isNumeric(String str) {
    try {
        Double.parseDouble(str);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}
private void openOutput() {
    try {
        outPut = new PrintWriter(comPort.getOutputStream());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
private void resetPosition() {
    jtf.setText("0");
    btnHold.doClick();
}
private String formatTextDecimals(int decimals, String s) {
    double d = Double.parseDouble(s);
    String s2 = String.format("%.7f", d);
    return s2;
}
}
```

6.7 Appendix – FSR Calibration Data

Force Gauge(Newton)	Analog read (mV from FSR resistor divider)
0,26	0
0,66	67
1,74	228
2,3	311
3,26	370
3,9	423
4,16	439
4,54	465
4,92	467
5,08	469
5,92	512
6,74	537
7,32	548
8,14	573
8,82	585
9,34	600
10,02	627
10,84	651
11,4	667
12,02	676
13,12	692
13,78	706
15,18	715
15,84	718
19,7	761
20,34	771
21,56	780
21,9	788
23,7	796
25,4	802
30,1	820
36,42	839
42	858

Sensory Feedback in Prosthetic Arms

50,1	877
56,78	889
63,76	900
67,94	909
71,84	914
75,56	921
82,3	928
85,12	931
89,56	933

7 References

- [1] Brigham and Women's Hospital, "Hand/Arm Transplant Surgery: Benefits and Risks," [Online]. Available: <https://www.brighamandwomens.org/surgery/plastic-surgery/restorative-surgery/benefits-and-risks-hand-arm-transplant>. [Accessed November 2019].
- [2] L. Chahine and G. Kanazi, "345M.E.J. ANESTH 19 (2), 2007LAMA CHAHINEPHANTOM LIMB SYNDROME:A REVIEW," American Univeristy of Lebanon (AUB), 2007.
- [3] L. Dahlin, M. Arner, P. Fredrikson, T. Laurell, H. Lindblom, P. Sassu, J. Wadström and M. Wiberg, "Handtransplantation snart verklighet i Sverige," Läkartidningen, [Online]. Available: <https://lakartidningen.se/Klinik-och-vetenskap/Klinisk-oversikt/2017/09/Handtransplantation-snart-verklighet-i-Sverige/>. [Accessed November 2019].
- [4] B. Stephens-Fripp, G. Alici and R. Mutlu, "A Review of Non-Invasive Sensory FeedbackMethods for Transradial Prosthetic Hands," University of Wollongong, 2018.
- [5] S. Arjunan, D. Kumar, L. Bueno and J. Mayor, "Upper Limb Prosthesis Devices," CRC Press, 2014.
- [6] C. Antfolk, "On sensory feedback in hand prostheses," Lund University, 2012.
- [7] New Zealand Artificial Limb Service (NZALZ), "Bowden Cable system," [Online]. Available: <https://www.nzals.co.nz/products/categories/upper-extremity/bowden-cable-system>. [Accessed November 2019].
- [8] J. Segil, G. Bertos and P. E, "Upper-Limb Prosthetic Devices," in *Handbook of Biomechatronics*, Academic Press, pp. 177-230.
- [9] "Ric Arm: Summary," Shirley Ryan AbilityLab, [Online]. Available: <https://www.sralab.org/cbm/ricarm>. [Accessed November 2019].

- [10] P. Svensson, U. Wijk, A. Björkman and C. Antfolk, "A review of invasive and non-invasive sensory feedback in upper limb prostheses," Department of Biomedical Engineering, Lund University, 2017.
- [11] M. Isaković, M. Belić, M. Štrbac, I. Popović, S. Došen, D. Farina and T. Keller, "Electrotactile Feedback Improves Performance and Facilitates Learning in the Routine Grasping Task".
- [12] ELPROCUS, "Force Sensing Resistor Technology," [Online]. Available: <https://www.elprocus.com/force-sensing-resistor-technology/>. [Accessed June 2019].
- [13] J. Dobrzynska and M. Gijs, "Capacitive flexible force sensor," Lausanne, 2010.
- [14] ELPROCUS, "Voltage Divider," [Online]. Available: <https://www.elprocus.com/voltage-divider-rule-with-examples/>. [Accessed June 2019].
- [15] W. Putnam and R. B. Knapp, "Input/Data Acquisition System Design for Human Computer," 1996.
- [16] Ada, L, "Adafruit - assets/429," Adafruit, [Online]. Available: <https://learn.adafruit.com/assets/429>. [Accessed September 2019].
- [17] A. Sadun, J. Sukor and J. Jalani, "Force Sensing Resistor (FSR): a brief overview and the low-cost sensor for active compliance control," Universiti Tun Hussein Onn Malaysia, 2017.
- [18] A. DeAngelis, D. Wilson and B. Mazzeo, "Capactive Sensor, US Patent: US7578195B2," 2006.
- [19] M. Hopstadius, "TRYCKGIVARE FÖR MÄTNING I INDUSTRIELLA PROCESSER," [Online]. Available: http://www8.tfe.umu.se/courses/elektro/tillmat/v45_98/projektrap/MagnusH-Tryck.htm.
- [20] Pressure Profile Systems, Inc., "SingleTact User Manual V3," 2017. [Online]. Available: https://www.singletact.com/SingleTact_Manual.pdf. [Accessed June 2019].

- [21] Pressure Profile Systems, Inc., "SingleTact Data Sheet," 2016. [Online]. Available: https://www.singletact.com/SingleTact_Datasheet.pdf.
- [22] Microcontrollers Lab, "PID controller implementation using arduino," Februari 2019. [Online]. Available: <https://microcontrollerslab.com/pid-controller-implementation-using-arduino/>. [Accessed June 2019].
- [23] C. Foster, "Modern updates in PID control tuning," Control Engineering, [Online]. Available: <https://www.controleng.com/articles/modern-updates-in-pid-control-tuning/>. [Accessed October 2019].
- [24] Minitab, "Interpret the key results for Boxplot," [Online]. Available: <https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/graphs/boxplot/interpret-the-results/key-results/>. [Accessed September 2019].
- [25] Khan Academy, "Box plot review," [Online]. Available: <https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/box-plot-review>. [Accessed September 2019].
- [26] Flinders University, "Box and Whisker Plots," Adelaide, 2013.
- [27] L. Ada, "Using an FSR," Adafruit, July 2012. [Online]. Available: <https://learn.adafruit.com/force-sensitive-resistor-fsr/using-an-fsr>. [Accessed June 2019].
- [28] Auctonix Motion Drives Inc, "Miniature Linear Motion Series - PQ12, Data Sheet," 2016.
- [29] L. Paredes-Madrid, C. Palacio, A. Matute and C. Parra Vargas, "Underlying Physics of Conductive Polymer Composites and Force Sensing Resistors (FSRs) under Static Loading Conditions," Sensors (Basel), 2017.
- [30] KYOWA Electronics, "Foil Strain Gage for Hydrogen Gas Environment," [Online]. Available: <https://www.kyowa-ei.com/eng/index.html>. [Accessed September 2019].
- [31] Actuonix, "PQ12-P Linear Actuator with Feedback," [Online]. Available: <https://www.actuonix.com/Actuonix-PQ-12-P-Linear-Actuator-p/pq12-p.html>. [Accessed June 2019].

- [32] Actuonix, "Linear Actuator Control Board," [Online]. Available: <https://www.actuonix.com/LAC-Board-p/lac.html>.
- [33] Tech Briefs, "Capacitive Force Sensor," February 2016. [Online]. Available: <https://www.techbriefs.com/component/content/article/tb/features/products-of-the-month/24008>. [Accessed June 2019].
- [34] P. Khatri, "Wheatstone Bridge," Circuit Digest, [Online]. Available: <https://circuitdigest.com/article/what-is-wheatstone-bridge>. [Accessed 2019 June].
- [35] National Instruments, "PID Theory Explained," 5 March 2019. [Online]. Available: <https://www.ni.com/en-us/innovations/white-papers/06/pid-theory-explained.html>. [Accessed June 2019].
- [36] R. Kansagara, Circuit Digest, 7 January 2018. [Online]. Available: <https://circuitdigest.com/article/what-is-pid-controller-working-structure-applications>. [Accessed June 2019].
- [37] Arduino, "Arduino Micro (Store)," [Online]. Available: <https://store.arduino.cc/arduino-micro>.
- [38] esri, "Box plot," [Online]. Available: <https://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/charts/box-plot.htm>. [Accessed September 2019].
- [39] Circuit Digest, "What is a PID controller," [Online]. Available: <https://circuitdigest.com/article/what-is-pid-controller-working-structure-applications>. [Accessed June 2019].
- [40] N. Fallahian, H. Saeedi, M. H and F. T. Ghomshe, "Sensory feedback add-on for upper-limb prostheses," University of Social Welfare and Rehabilitation Sciences, Tehran, Iran, 2016.
- [41] C. Antfolk, M. D'Alonzo, B. Rosén, G. Lundborg, F. Sebelius and C. Cipriani, "Sensory feedback in upper limb prosthetics," 2014.
- [42] Mark-10, "Advanced Digital Force Gauges Series 4," [Online]. Available: <http://www.mark-10.com/instruments/force/series4.html>. [Accessed October 2019].

- [43] SOLARBOTICS, "Force Sensitive Resistor," [Online]. Available: <https://solarbotics.com/product/50803/>. [Accessed October 2019].
- [44] L. Ada, "Force Sensitive Resistor (FSR) - Overview," [Online]. Available: <https://learn.adafruit.com/force-sensitive-resistor-fsr/overview>.
- [45] A. Ciancio, F. Cordella, R. Barone, R. R, A. Bellingegni, S. R, D. A, G. Pino, F. Ranieri, V. Lazzaro, G. E and L. Zollo, "Control of Prosthetic Hands via the Peripheral Nervous System," 2016.
- [46] C. Antfolk, C. Cipriani, M. Carozza, C. Balkenius, G. Lundborg, B. Rosén and F. Sebelius, "Transfer of tactile input from an artificial hand to the forearm: experiments in amputees and able-bodied volunteers," Department of Measurement Technology and Industrial Electrical Engineering, Lund University, 2018.
- [47] S. Arhunan, D. Kumar, L. Bueno and J. Mayor, "Upper Limb Prosthesis Devices," CRC Press, 2014.
- [48] J. Dobrzynska and M. Gijs, "Capacitive flexible force sensor," 2010.
- [49] M. Carrozza, G. Cappiello, S. Micera, B. Edin, L. Beccai and C. Cipriani, "Design of a cybernetic hand for perception and action.," 2006.