A Stroke Therapy Brace Design

By

Evan Kirkbride

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

June 2016

Table of Contents

List of Tables and Figures

Abstract

Victims of stroke often have difficulty with rehabilitation. With limited movement on their affected arm, patients often do not want to move much for physical therapy. In this project, we design a robotic brace that helps stroke patients move their arm more effectively in a reaching or pulling motion. By giving patients more movement in their affected arm than they would have otherwise, patients gain more from rehabilitation. The brace also adapts to the patient's needs, providing more inclination or resistance as needed for their physical therapy. This kind of therapy engages patients rather than relying on their likely dwindled motivation. This project includes both software coding and hardware implementation. A video of the project demonstration is available upon request.

Chapter 1 - Introduction

Stroke claims the second most amount of lives in the world, and leaves 75% of its survivors with enough disabilities to decrease their employability. Most stroke victims use physical therapy for treatment. Recent research in stroke therapy has included elements such as virtual reality, video games, and robotic therapy for improvement [1]. Different research groups have developed and tested these methods. S.-C. Yeh *et al* showed that repetitious actions that simulated reality in a “skinner-box” virtual reality set-up proved effective for stroke treatment [2]. And Holden *et al* demonstrated how virtual reality allowed for the treatment to occur remotely with similar results [3].

The iMove laboratory at University of California, Irvine researches these concepts. The lab treats and improves the lives of those affected with neural injuries by developing robotics and devices that assist with their movement during treatment. These devices provide the patient with more movement than they would have otherwise for physical therapy, allowing for more effective treatment. Past projects include the robotic exoskeleton T-WREX [4], and the interactive web-based telerehabilitation [5].

I received the opportunity to intern at this lab and worked on a rat training box. This box allowed a rat with a neural injury to train their affected arm by pulling on an apparatus to dispense a treat. An electric motor and microcontroller helped the rat move the apparatus, adapting to their needs but still increasing in difficulty. The system also collected and exported data during the trial to allow for more analysis. I left iMove with the project unfinished, thus for my senior project I hoped to complete and expand upon the task to create a functional product, while moving up the evolutionary ladder and updating the machine to work with humans.

For this report, the next chapter will outline more of the literature and past projects surrounding these concepts, and how this project relates to those. Chapter 3 gives more background on the previous project I worked on for the internship and how it applies to this project, while outlining other considerations for this project such as discussing the customer needs research. Chapter 4 summaries the design method and results of the project. Chapter 5 offers conclusions for the project and gives insight on future works. The appendices include items such as the code used, the cost accumulated, and the ABET senior project analysis.

Chapter 2 – Literature Review

Devices for stroke therapy prove effective at addressing upper extremity impairment of stoke patients [2] - [6]. The devices demonstrate an opportunity for repeatable actions that simulate or correlate to useful everyday behavior. Systems of this variety also allow for precision measurements that give clearer pictures to the physical therapists to guide treatment. They also allow for more engaging therapy as instructing patients with a simple and interactive goal helps with retention through iteration [2, 3, 5]. Though these projects grow in number, the amount of actual commercial or practical patents on these findings are still limited, and mostly consist of individual tools or subsystems for robotics [7].

S.-C. Yeh *et al* demonstrates a virtual reality (VR) simulation and motion capture for a similar affect. While more computer science orientated, the paper illustrates an easy to understand metric for measuring progress: the amount of time used to complete a task. Plots of this progress show that whether controlled or automated, repetitious treatment can lead to better performance in stroke patients [2]. M. Holden *et al* illustrates a similar process in a VR environment, but uses different metrics such as the Fugl-Meyer Motor score and the Wolf Motor test to track progress, and the results are similar [3].

D. Reinkensmeyer *et al* highlights a therapy process using a Java software and more mechanical input device, such as a joystick. While using similar data collection methods as in S.-C. Yeh *et al* [2] and M. Holden *et al* [3], Reinkensmeyer *et al* dabbles in this robot centered therapy in conjunction with software techniques. The software techniques also incorporate game theory concepts through the design of a trial, which can help keep a patient engaged [5]. A more involved robotic approach was introduced with the T-WREX. This robot arm brace gave patients more movement and control over their arm movements. While the results were not too different from previous methods, such as in [5], the user satisfaction saw a massive increase in the patient confidence in their actions and the use of the treatment [4].

This project centers on the hardware control and interfacing those theories through a training algorithm. The project takes a more engineering centered approach, rather than using human trials to prove the effectiveness of this device through trials and statistics. The project allows for an exploration how such devices are made, and what should be considered during the design process.

Chapter 3 – Background

Based on previous projects, the training algorithm needs to offer stable control of the motor process while offering the benefits of robotic medical training. In order for this to happen, the program needs to increase difficulty in order to ensure that the patient will improve progress, but still allow room to decrease difficulty of a task if the patient has trouble completing it, effectively adapting to the needs of the patient. Past robotic-centered treatments have caused the patients to become lazy in their treatment, letting the machine do the work for them [3], [4]. As such, the device needs to ensure that the patient is in the most amount of control.

For the physical therapists and researchers, the system also collects data during a trial, and compiles a data file at the end of a trial. This data may include: the force the motor or patient exerts, the motor's velocity, and the state or difficulty level of the system over time. This allows the physical therapists to interpret the progress of the patient. The therapist can also alter the state of the system and adjust its difficulty for the patients based on their judgement. These states and desired variables base themselves on S.-C. Yeh *et al*, though on a smaller scope [2].

The past version of the project, the rat box, used a state machine to accomplish this task. The microcontroller measured parameters such as velocity and position of the motor via a linear piston potentiometer, as well as the error of the current position versus the desired position to switch between states and collect data with. The motor also somewhat functioned like a spring, with the "spring constant" being determined by the voltage outputted to the motor via the microcontroller and motor driver. If the patient pulled on the motor, they would experience a resistance similar to that of a spring. A picture of the motor can be seen in Figure 1 below. The challenge of the previous project was to pull on the motor past certain distances, which contributed resistance similar to a spring, and the tension of the "spring" would tighten or loosen depending on how well the patient preformed. This device worked for a rat with a neural injury, and the device would dispense a food pellet to reward the rat for accomplishing the task and incentivize the rat to engage with it more. The states are outlined in the Figure 2 below.

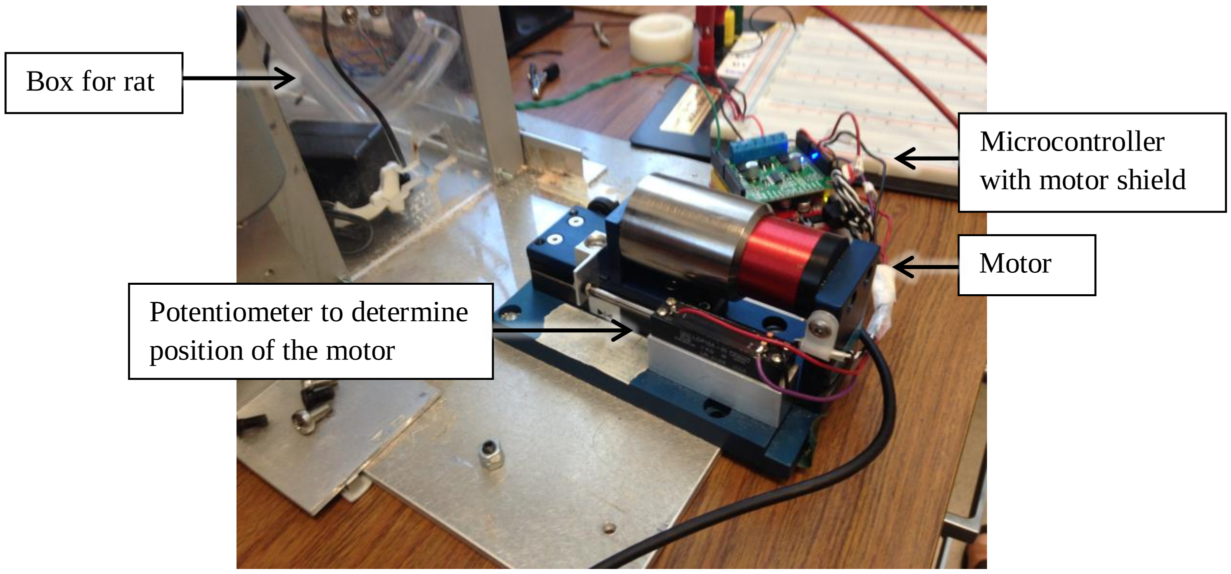

Box for rat
Microcontroller with motor shield
Motor
Potentiometer to determine position of the motor

Figure 1: Original Project Setup

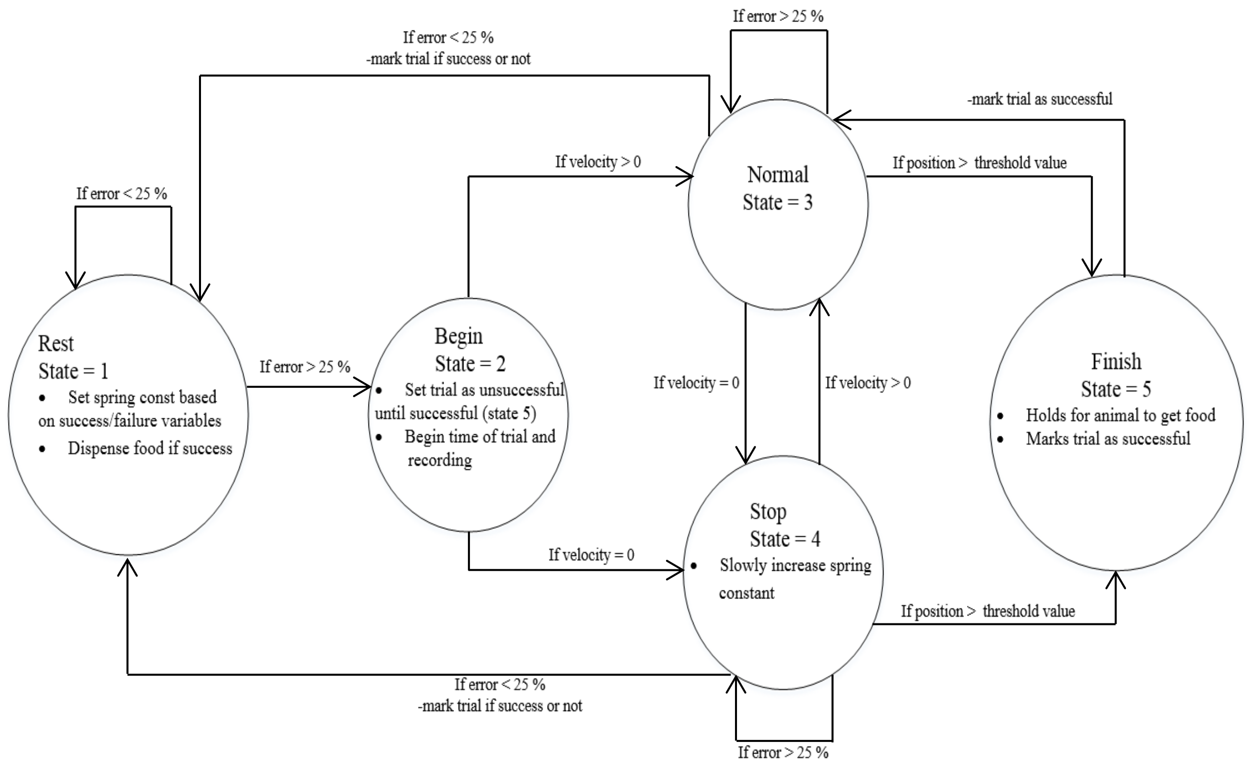

If error > 25 %
If error < 25 %
-mark trial if success or not
-mark trial as successful
If velocity > 0
Normal
State = 3
If position > threshold value
If error < 25 %
Rest
State = 1
Set spring const based on success/failure variables
Dispense food if success
If error > 25 %
Begin
State = 2
Set trial as unsuccessful until successful (state 5)
Begin time of trial and recording
If velocity = 0
If velocity > 0
Finish
State = 5
Holds for animal to get food
Marks trial as successful
If velocity = 0
Stop
State = 4
Slowly increase spring constant
If position > threshold value
If error < 25 %
-mark trial if success or not
If error > 25 %

Figure 2: Original Project's State Machine

State 1 in Figure 2 acts as a rest state, with the machine staying in that state while the rat did not move the motor, and once the rat started to move the motor to initiate a trial, the error signal would change and the machine would move into State 2 after it was above 25 %. State 2 would only be entered once to flag that a trial was starting, and while the patient pulled on the motor, the state would shift between states 3 and 4. If the motor didn't move (i.e. the velocity was zero), the "spring constant" would increase in each iteration of the *void loop* in the Arduino code by increasing the voltage sent to the motor via the motor shield, and the motor would begin to pull against the patient. This was done in order to incentivize the rat to constantly move the motor. Once the rat moved the motor past a certain threshold value, the machine would enter State 5 and mark the trial was successful. Trials would be marked as unsuccessful until the patient reached State 5, meaning if they never reached State 5 after initiating a trial, and returned to State 1, the machine will mark the trial as unsuccessful. State 5 is only entered once, similar to State 2, and enters State 3 next, effectively pulling the motor back to the starting position and State 1, if the patient did not continue to the pull on the machine. If the trial was successfully finished by the time the machine entered State 1 again, the "spring constant" of the motor would increase, and effectively increase the difficulty of the next trial. The machine would also dispense a food pellet as reward for the rat if the trial was successful, incentivizing them to go again. If the trial was unsuccessful, then the spring constant would decrease in order to lessen the difficulty of the next trial.

There were additional features of the internship machine, however the methodology explained above summarizes and acts as the main influence on the project discussed in this paper. This state machine did not do an accurate job of determining an adaptive difficulty for a patient, only increasing or decreasing the “spring constant” or effective difficulty for a trial when certain criteria were met. There were also stability issues for the machine: if the user or microcontroller wanted to motor to reach a certain position, the motor would often oscillate around that position or become unstable. Ideally, the motor would help the patient move using a dictated path for the motor, but with the instability issues of the motor, the motor had difficulty following any such path, and the feature was left unfinished. A new state machine was required for this project. Certain ideas, such as having a rest state and finished state, are carried over, but the criteria and methods for determining difficulty and the status of the patient were updated. Different parts are also be used for this project, which affected the design process and planning around these parts.

To prepare, a customer needs assessment was required to determine who this project is aimed at, and from there different marketing requirements and engineering specifications take shape. After assessing different literature: stroke patients, their families, doctors, physical therapists, and researchers make up this product’s major customers. While the intention of the system relates to stroke patients, anyone who could benefit from the device can also serve as a potential customer. Customers require treatment that offers more movement and control of their affected arm. Allowing their arm more flexibility can help with the treatment process. Researchers and physical therapists can also use the system to control the therapy more effectively. The ability to adequately control the system and receive data gives the therapists more flexibility. Past research assessed and determined these needs [2-5].

The system allows for easy use, giving patients an incentive to use it over typical physical therapy. Improved muscle movement satisfies the customer’s main concern. The system also adapts to the needs of the patient. Past treatments of this caliber have caused the patients to become lazy in their treatment, letting the machine do the work for them [3], [4]. As such, the system adapts and become harder, but still lessens the difficulty if the patient has trouble.

Table 1 summarizes the engineering specifications and marketing requirements found from these needs. Table 4 showcases the specifications of the finished product, and can be used to compare with Table 1. The format of the table derives from Ford and Coulston [6].

Table 1: Stroke Therapy Brace Initial Requirements and Specifications

Marketing Requirements	Engineering Specifications	Justification
1, 5	Should take less than 5 minutes to begin a trial on average	Guarantees easy use for the patient.
2, 3	Increases the reaching motion length of the average patient during a trial by at least 10%	Ensures effective product treatment. Previous experiments indicate that 10% improvement is desirable.
5, 6	Powers supplied by Microcontroller power source and/or 9 Volt battery	Helps reduce system size and lowers power limits.
5, 6	Entire system weighs less than 20 lbs.	Helps reduce size and feasibility of the system.
1, 5, 6	The apparatus the patient moves weighs less than 5 lbs.	Allows the affected arm to properly move and work the device.
1, 3, 4, 7	Reads distance of brace (in) and force (N) applied or exerted and compiles data within 1 min of a trial	Gives instant feedback and ensures data collection.
5, 7	Can turn off the system within 1second of initiating stop	Gives safety measure for unforeseen events during a trial.
1, 7	Can alter initial conditions (max/min variables, time of trial, difficulty level) within 5 % accuracy	Allows therapist and patient control on the treatment progress and allows for multiple patients to use system.
6	At most 1' X 3' X 2'(when fully extended)	Within specified dimensions to limit space and give portability to the system.
Marketing Requirements: 1. Easy to use 2. Offers better treatment and use of arm 3. Adapts to needs 4. Able to collect data 5. Safe to use 6. Small and portable 7. Offers control for the therapist and patient		

An Arduino Uno with an ATMega328p microcontroller was used for this project [7], as the designer was most familiar with this microcontroller. The microcontroller could be used either with the Arduino licensed GUI and the use of open source code, or by using Atmel studio to set the bits of registers within the ATMega328p microcontroller to allow for more nuanced control and capabilities from the microcontroller. For this project, the Arduino licensed GUI was used for simplicity. LabView was also used late in the project's design as a user interface, communicating to the Arduino via serial and USB. Initially a python script tested a user interface, but LabView's potential created a more approachable face to the project. The inductive motor found in Figure 1 could not be found for this project, so a Frigelli L12 linear actuator was used instead.

Chapter 4 – Design and Results

The motor used in the original internship project and seen in Figure 1 could not be found in preparation for this design. As a result, a Frigelli L12 linear actuator was used for this project. This actuator moves in a linear motion and can be operated using a pulse width modulation (PWM) signal to control the position of the actuator. This allows for a nice interface with a microcontroller, which can easily generate a PWM signal while executing different processes. The dimensions of the actuator can be seen in Figure 3 below.

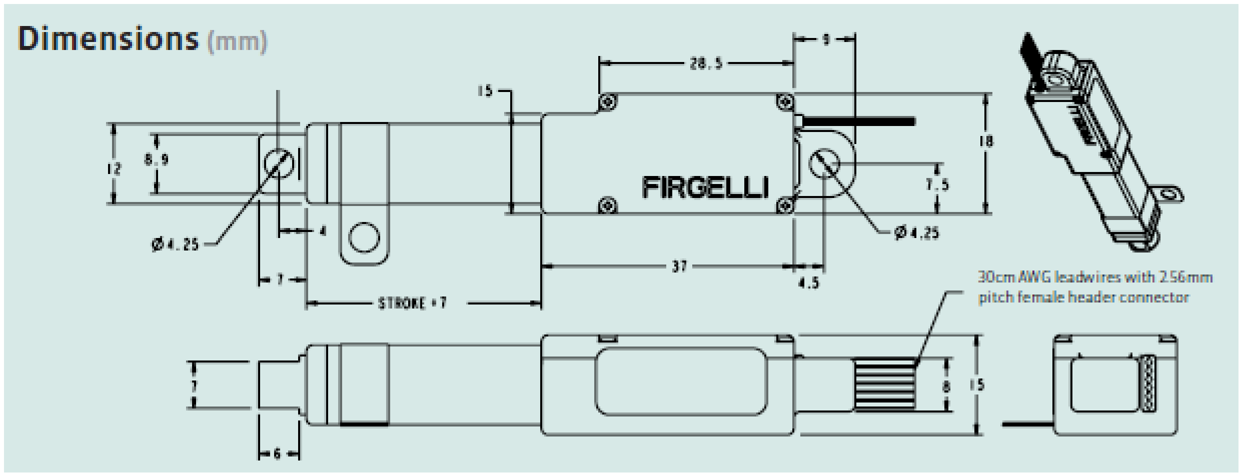
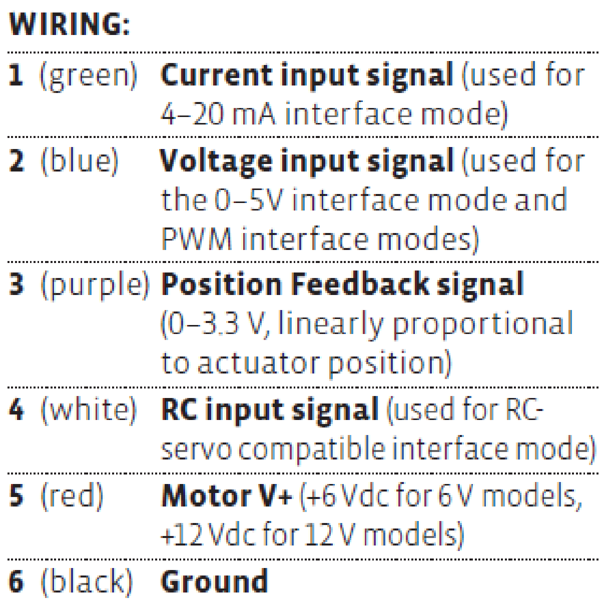


Figure 3: Firgelli L12 Linear Actuator Dimensions [8]

The Actuator in Figure 3 requires a PWM signal of 0 V - 5 V, 1 kHz square wave, where the duty cycle of the waveform dictates the position of the actuator. The wiring of the actuator also allows for a current input or RC input, however these control methods were not used. The product also features a position feedback signal, which is useful for giving feedback to the microcontroller on the state of the actuator. The Actuator also needed a 12 V supply. The table in Figure 4 below outlines this wiring scheme with a picture of the actuator wires as reference.



WIRING:

1	(green)	**Current input signal** (used for 4–20 mA interface mode)
2	(blue)	**Voltage input signal** (used for the 0–5V interface mode and PWM interface modes)
3	(purple)	**Position Feedback signal** (0–3.3 V, linearly proportional to actuator position)
4	(white)	**RC input signal** (used for RC-servo compatible interface mode)
5	(red)	**Motor V+** (+6 Vdc for 6 V models, +12 Vdc for 12 V models)
6	(black)	**Ground**

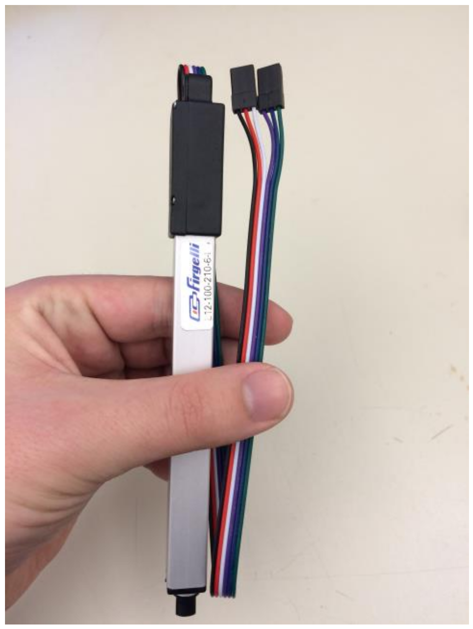
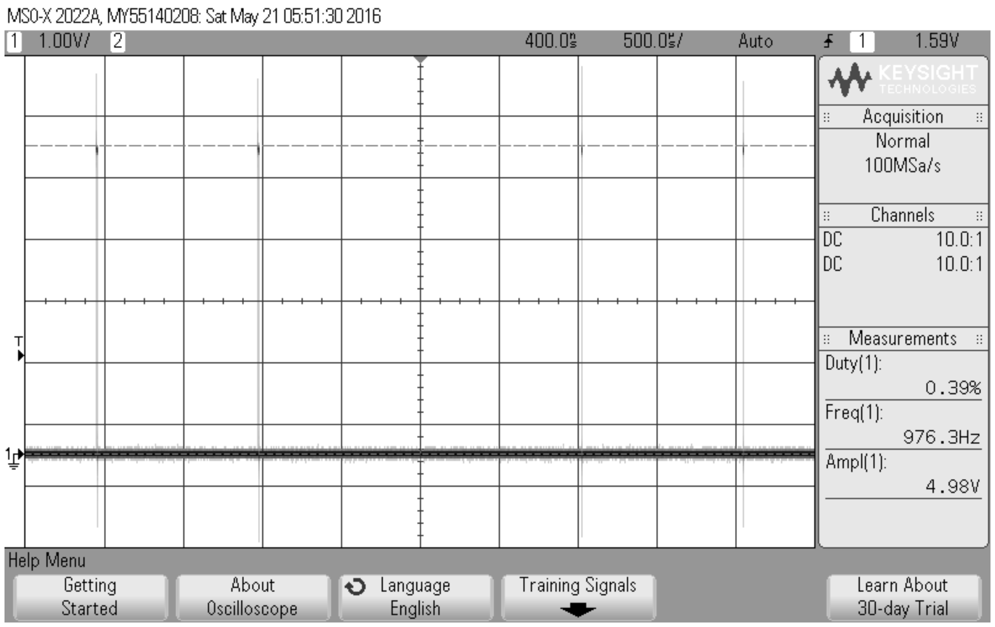


Figure 4: Firgelli L12 Linear Actuator Wiring [8]

Different duty cycles of a 0-5 V, 1 kHz square wave were inputted to characterize the control of the actuator, and what length or extension of the actuator correlates to what duty cycle. Roughly half of the reach of the actuator is achieved when a 50% duty cycle square wave is inputted. A near 0% duty cycle fully retracts the actuator, and a near 100 % duty cycle fully extends the actuator. The actual PWM signal generated by the microcontroller uses a built in 8-bit timer, and as such the microcontroller does not generate a waveform at exactly 1 kHz, or at an exact duty cycle, but the actuator still operates adequately, implying some room for error that the actuator can sense. Some examples of the PWM signal that drives the actuator can be seen in Figures 5 - 7 below.



MSO-X 2022A, MY55140208: Sat May 21 05:51:30 2016

1 1.00V/ 2 400.0s 500.0s/ Auto 1 1.59V

Acquisition
Normal
100MSa/s

Channels
DC 10.0:1
DC 10.0:1

Measurements
Duty(1): 0.39%
Freq(1): 976.3Hz
Ampl(1): 4.98V

Figure 5: 0 % Duty Cycle Output

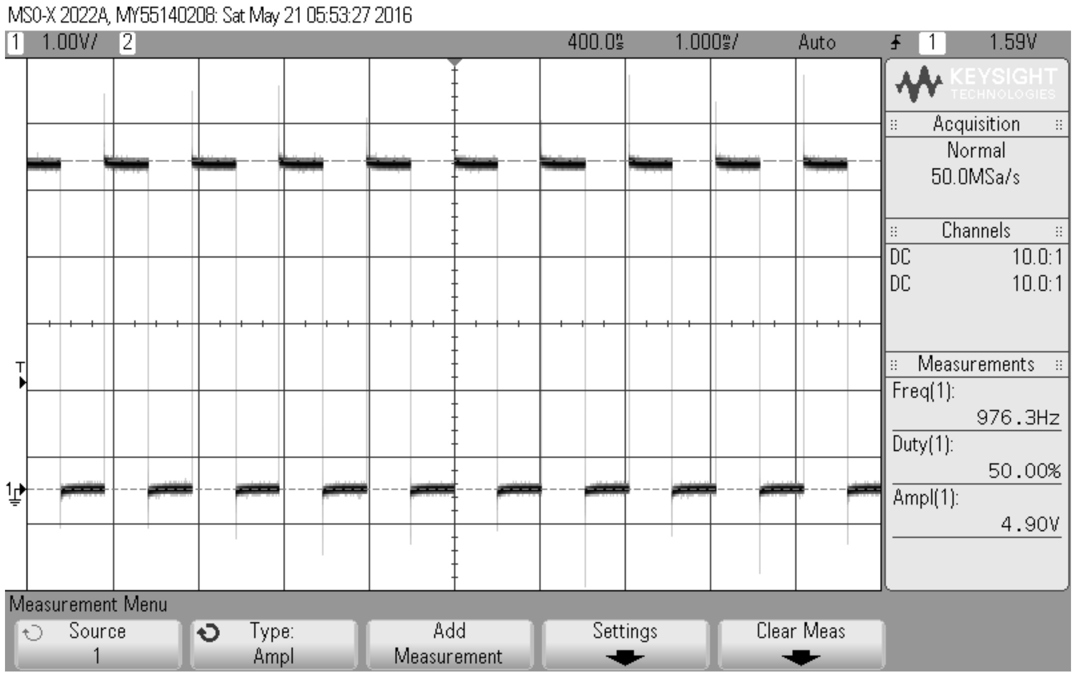


Figure 6: 50 % Duty Cycle Output

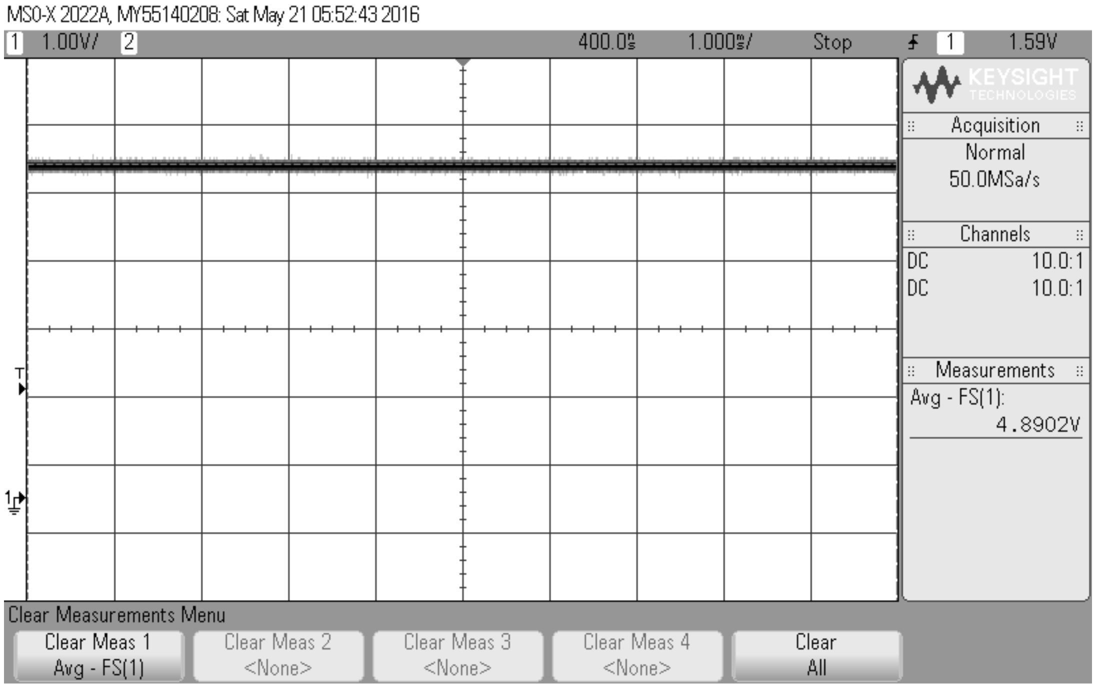


Figure 7: 100 % Duty Cycle Output

Each waveform in Figures 5, 6, and 7 operate around 5 V_{pp}, with a maximum variance of 2.24 %. While the 0 % duty cycle output in Figure 5 actually outputs a 0.39% duty cycle from the microcontroller; however this still fully retracts the linear actuator. The duty cycles in Figures 6 and 7 are much more accurate; however the frequency of the signal is at 976.32 Hz, 2.425 % off from the desired 1 kHz. Again, the error discussed is due to the limited capabilities of the microcontroller, but the actuator still operates under these conditions.

While the actuator moves and reaches its ending position, the actuator often adjusts itself a bit at the end, overshooting and undershooting a bit before completely stopping, somewhat similar to a response found in an underdamped or PID controller step response. Such phenomena can be sporadic and random depending on how the actuator moves, and the only way to avoid it totally is to have the actuator constantly move. The full reach of the actuator is 4.25", outlined with a ruler below in Figure 8 for reference.

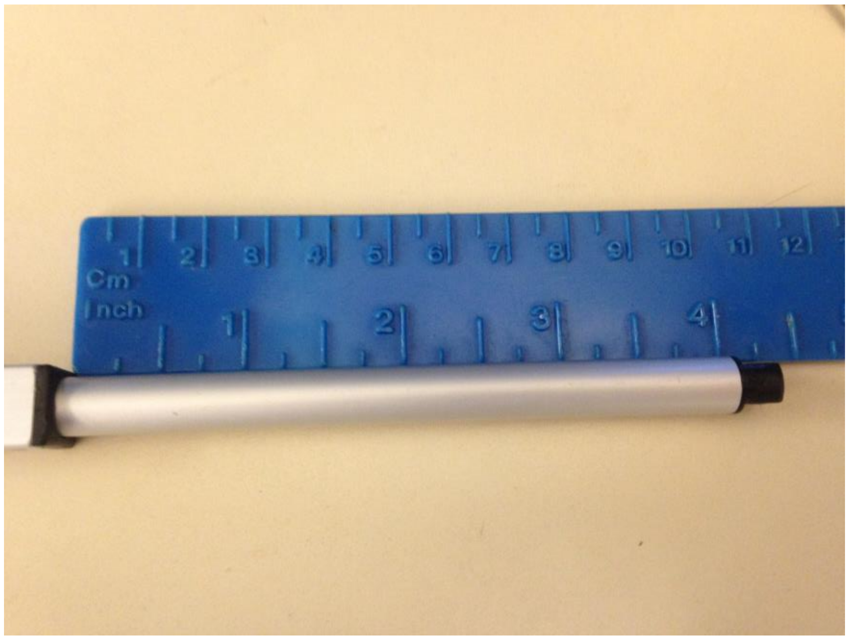


Figure 8: Actuator Length

The position feedback signal (wire 3 in Figure 4) was then measured and compared with the PWM input to outline the functionality of the actuator. This data can be seen in Table 2 and Figure 9 below.

Table 2: Linear Actuator Position Feedback

PWM input (%duty cycle)	Position feedback (V)
1	3.2784
5	3.1422
10	2.9766
15	2.8371
20	2.6648
25	2.4938
30	2.3253
35	2.17
40	1.9975
45	1.8207
50	1.663
55	1.4859
60	1.3511
65	1.1833
70	1.0209
75	0.83583
80	0.66558
85	0.51167
90	0.35525
95	0.17196

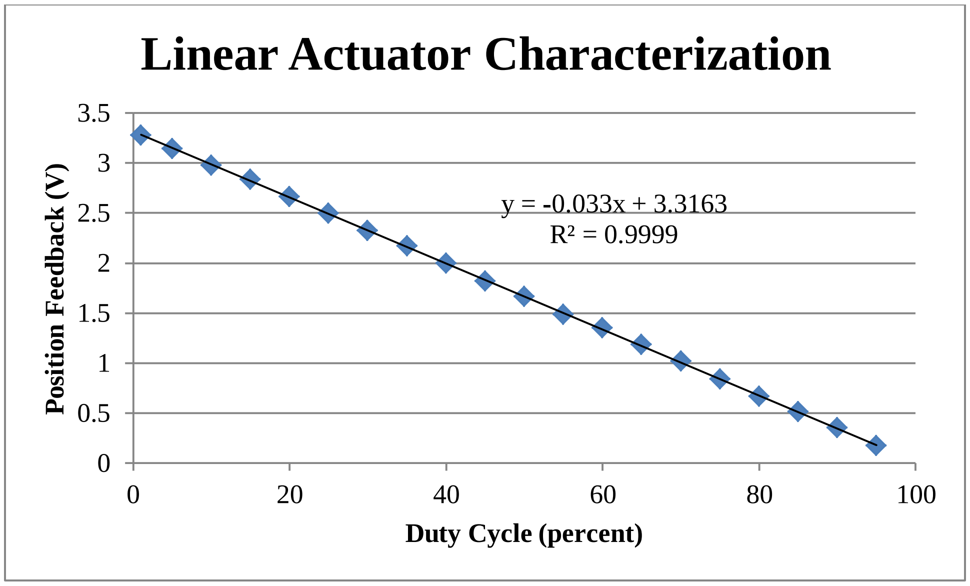


Figure 9: Firgelli L12 Linear Actuator Characterization

As seen in Figure 9, the position feedback signal has an inverse, near linear relationship with the PWM duty cycle. This feedback also works at a maximum of 3.3 V, which is convenient for interfacing with a microcontroller. The actuator requires a 12 V supply to operate, however this does not have an easy battery equivalent, which was an initial specification in Table 1. The actuator also has a peak current draw of about 150 mA, with an average of 100 mA current draw when the actuator moves. While the 12 V requirement can be achieved with two 9 V batteries in series (such as a MN1604 Duracell battery), stepped down to 12 V (via a resistor divider or buck converter), the required current draw is a more prominent limitation of battery capabilities. As such, the linear actuator was found to only work with a separate DC power supply that was able to supply the required current, with typical batteries unable to adequately power the device.

One drawback about using a linear actuator is that a patient cannot grab on the arm of the actuator and pull to make it move. Not only does the actuator not have an apparatus to grab onto, but the actuator arm itself is stiff and won't move until it is dictated to do so by the PWM signal. As such, the device needs an apparatus for the patient to grab onto. This can be done by attaching a sliding potentiometer or linear transducer to the actuator. If a patient pulls on this, the signal generated can be used to tell the microcontroller that the patient is initiating a movement, and that the linear actuator should start moving. For this project, the sliding potentiometer was also attached to a piece of wood so it could move across a table or flat surface. Because the actuator has to push against a piece of wood and a human hand, it needs a heavier material to push back against, such that the potentiometer and actuator arm adequately moves. Thus a larger piece of wood was used for behind the actuator. This kind of potentiometer setup can be seen in Figure 10 below.

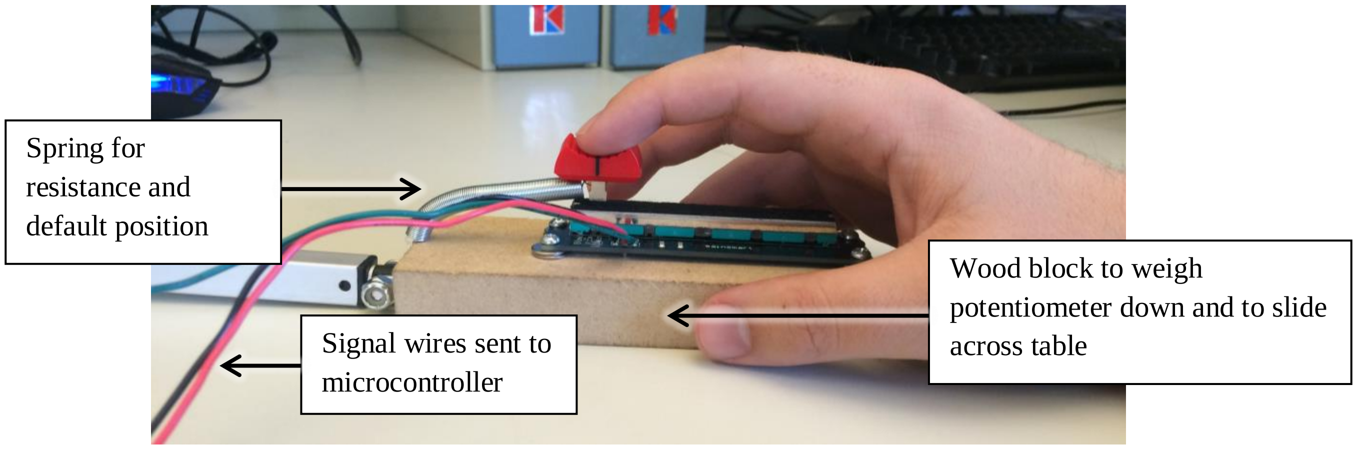

Spring for resistance and default position
Wood block to weigh potentiometer down and to slide across table
Signal wires sent to microcontroller

Figure 10: Linear Actuator with Potentiometer Setup.

With the linear actuator, potentiometer, and microcontroller usage outlined, a block diagram of the whole device setup can be seen in the block diagram in Figure 11 below.

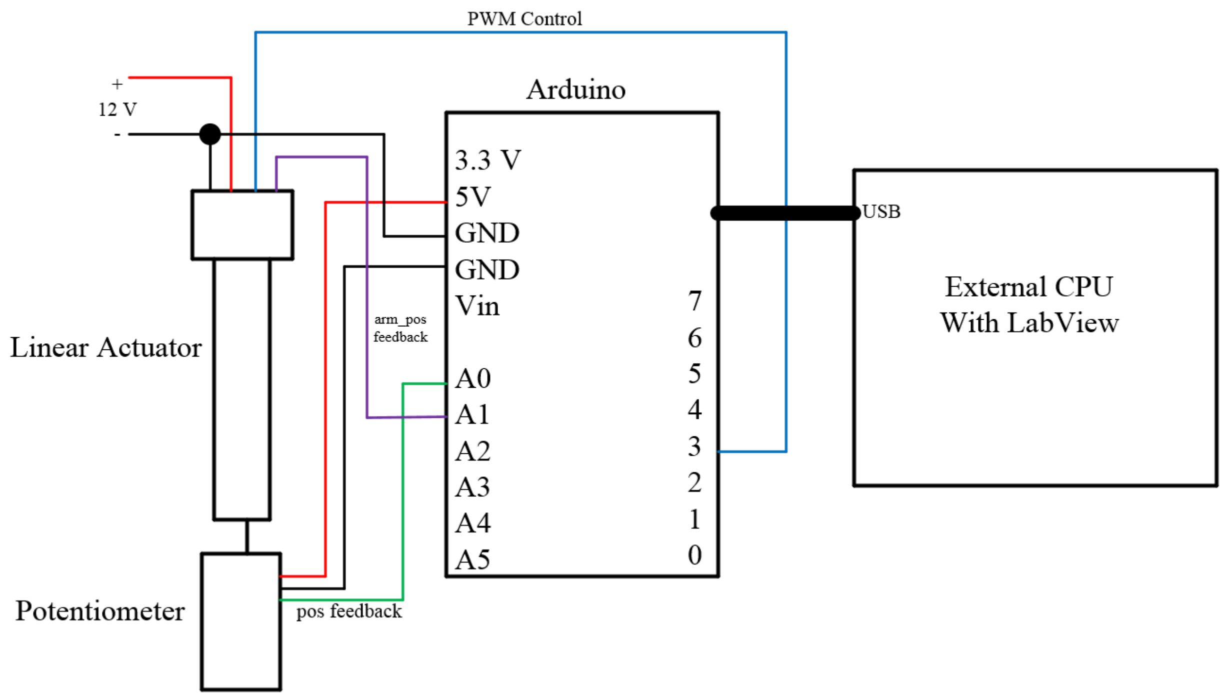


Figure 11: Device Block Diagram

Figure 11 showcases the wiring diagram of the device. The Arduino microcontroller acts as the centerpiece of the design, as it does most of the measuring and processing, with most inputs and outputs of each subsystem leading to or from the Arduino. The linear actuator requires a 12 V supply, with its ground pin attached to the ground pin of the Arduino, such that it can interface with the internal voltage levels of the Arduino. The potentiometer uses the 5 V supply and other ground pin of the Arduino to power it, and its position feedback signal is driven back to the A0-analog input pin of the Arduino. Similarly, the position feedback of the linear actuator is attached to the A1-analog input pin of the Arduino. These are labeled *pos* feedback and *arm_pos* feedback on Figure 7, which correlates with the variables in the Arduino code they represent. The digital pin 3 of the Arduino acts as the PWM control for the linear actuator. Lastly, the Arduino is powered via a USB cord, which connects to an external CPU. This USB also communicates to the CPU via serial, with the CPU running LabView as a user interface for the whole device and as a way to collect data. A picture of the testing setup similar to the diagram in Figure 7 can be seen in Figure 12 below.

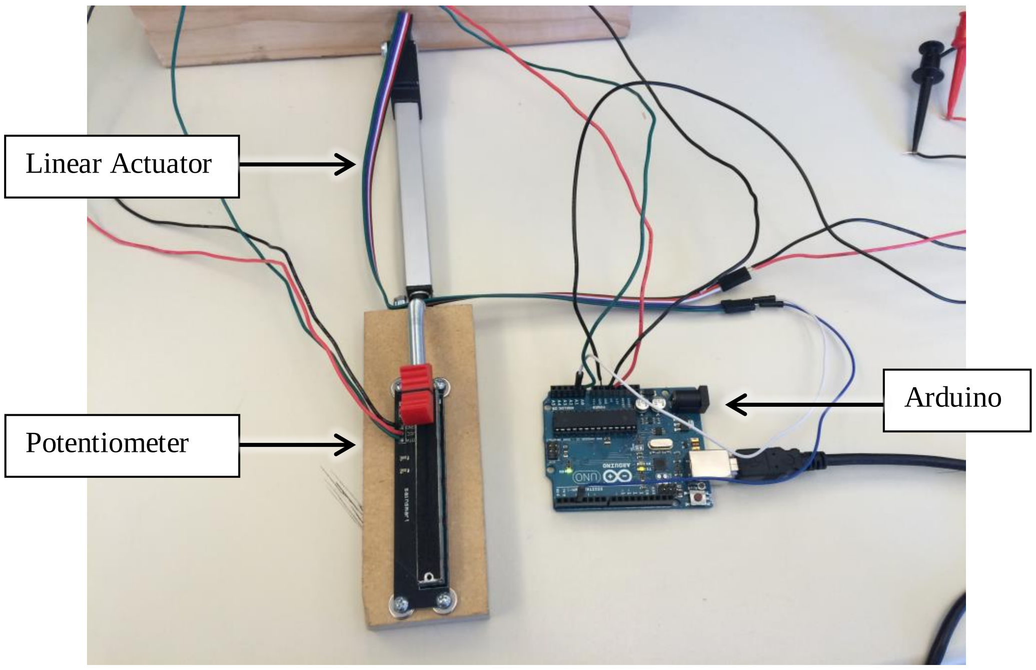


Figure 12: Device Wiring

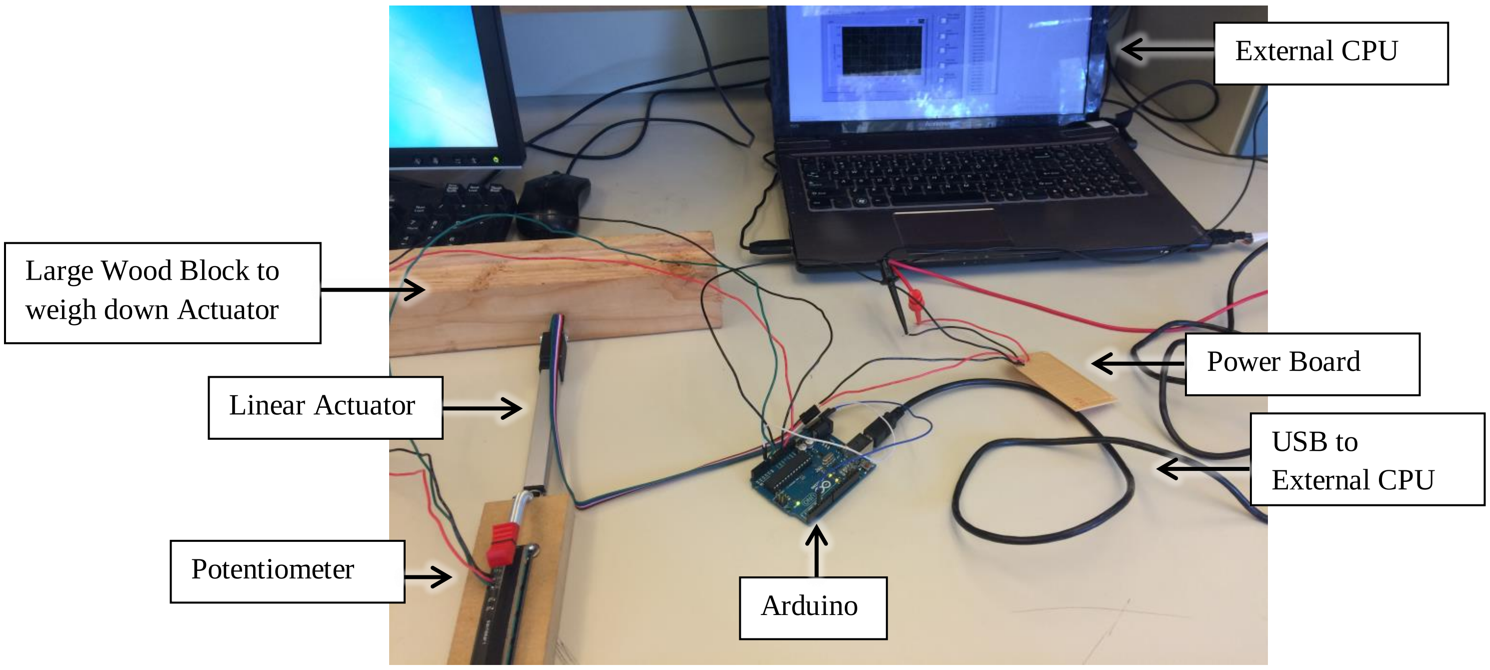


Figure 13: Full Device

While Figure 12 showcases the wiring similar to Figure 11, Figure 13 shows the computer connected via USB, and also includes a small PCB board used for the 12 V source. This PCB can be seen in more detail in Figure 14 below.

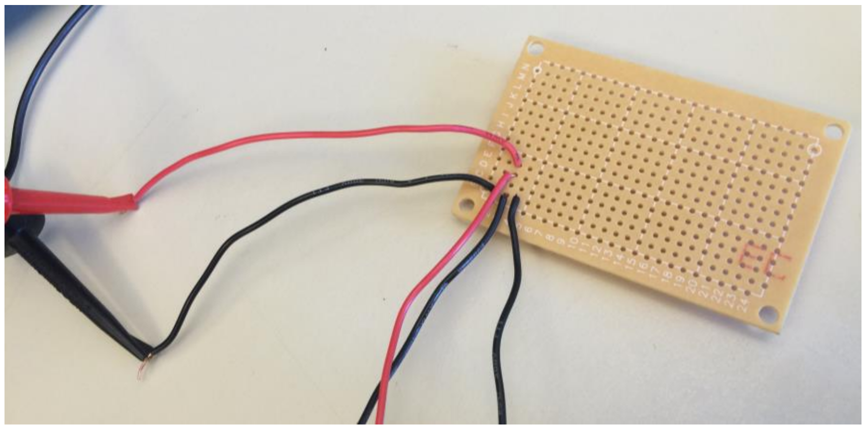
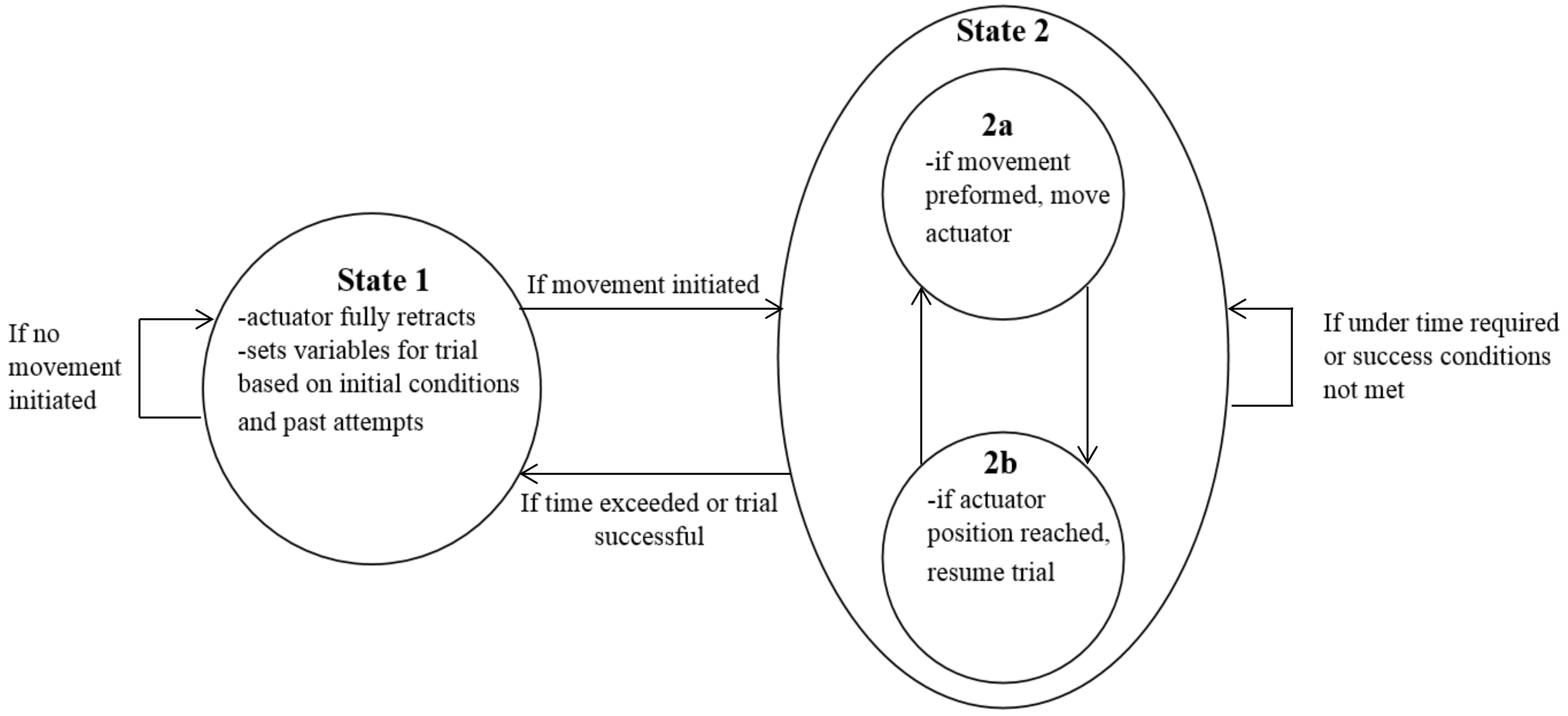


Figure 14: Power Board

The board in Figure 14 takes the 12 V supply and splits up the ground node, with one wire going to the ground wire of the linear actuator (wire 6 in Figure 4), and the other going to the ground pin of the Arduino. This is done to ensure both devices use the same ground node, so they can properly interface. The positive rail also goes to the linear actuator (wire 5 in Figure 4).

With the linear actuator characterized, and the actuator movement planned, the state machine that runs the training algorithm was created. Through a series of testing the Arduino's capabilities in driving the linear actuator, and building of the original state machine in Figure 2, the resulting state machine can be seen below in Figure 15.



State 2
2a
-if movement preformed, move actuator
State 1
-actuator fully retracts
-sets variables for trial based on initial conditions and past attempts
If movement initiated
If no movement initiated
If under time required or success conditions not met
2b
-if actuator position reached, resume trial
If time exceeded or trial successful

Figure 15: Device State Machine

Similar to the original state machine in Figure 2, State 1 in Figure 15 represents a rest state, insuring the linear actuator does not move until the patient initiates a movement. Once a movement is initiated, the machine moves into State 2, which has two sub-states. State 2 represents the machine during a trial, where the patient needs to continuously move the potentiometer in order to move the actuator. If the maximum time set for a trial passes, or if a trial is deemed successful, then the machine will return to State 1 and the linear actuator will return to its initial position. Within State 2, the machine will alternate between two states, 2a and 2b. 2a initiates a movement of the linear actuator if the patient adequately preforms a movement, and 2b waits until the linear actuator reaches its desired position before it moves again, to prevent the actuator from stalling. This effectively creates steps of movement in the actuator until a trial is successful. These steps can help dictate the difficulty of a trial, if more steps are required before the actuator fully extends, the task is more difficult.

To illustrate the function of the state machine above, and how it relates to the actual movement of the device, Figure 16 below outlines the process the state machine in Figure 15, with arrows indicating the movement of the potentiometer or actuator.

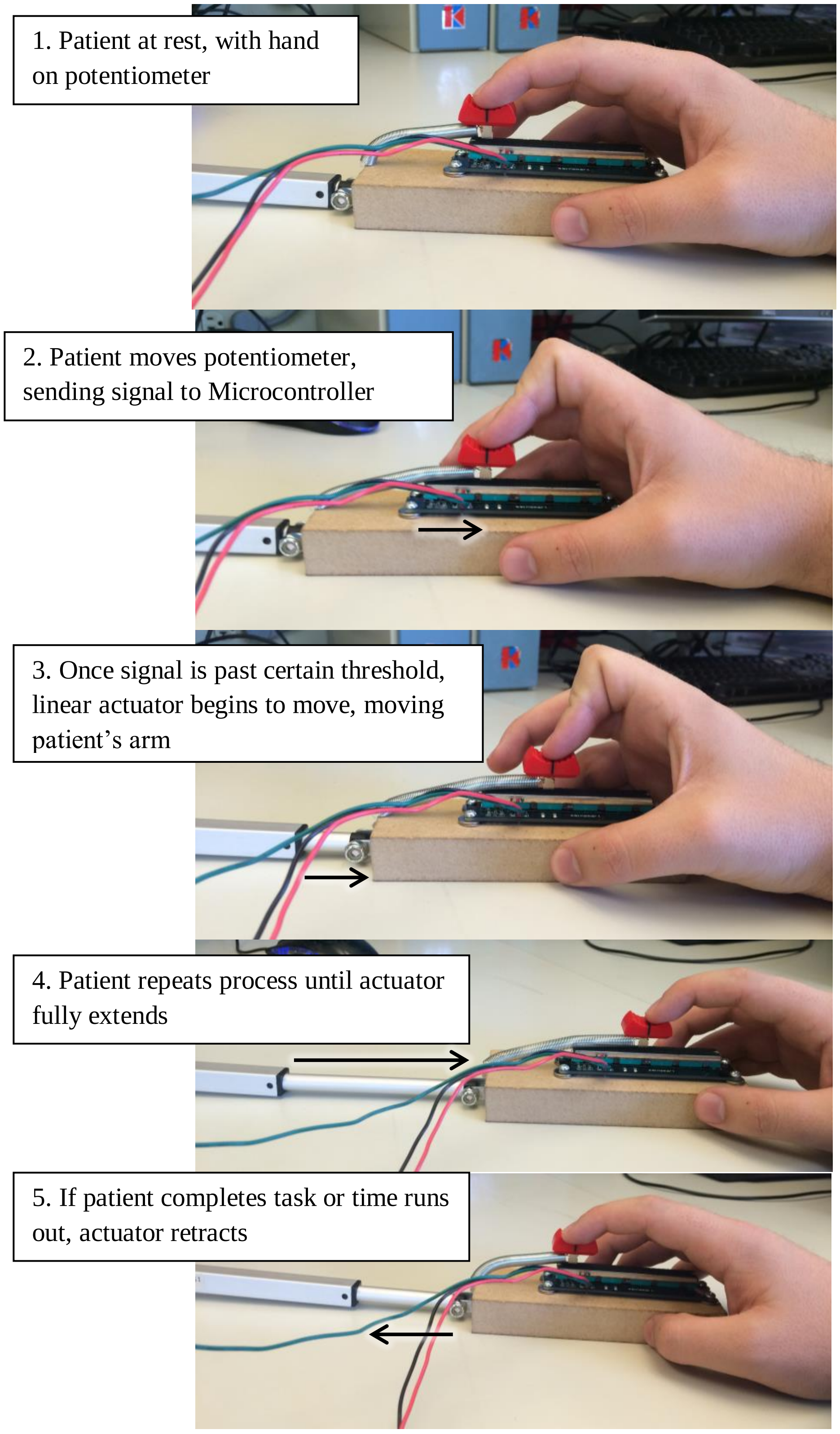

1. Patient at rest, with hand on potentiometer
2. Patient moves potentiometer, sending signal to Microcontroller
3. Once signal is past certain threshold, linear actuator begins to move, moving patient’s arm
4. Patient repeats process until actuator fully extends
5. If patient completes task or time runs out, actuator retracts

Figure 16: Trial process

In conjunction with the State machine detailed above in Figure 13 and 14, different difficulty levels can be introduced in order to give some utility to the device. The patient or therapist gives initial inputs that affect certain variables within the state machine that alters its function. Difficulty levels alter the threshold that the potentiometer needs to move to move the actuator, such as in step 2 and 3. They also alter how far the linear actuator moves whenever a potentiometer threshold is reached, such as in steps 3 and 4. These values also adapt depending on how well the patient does throughout a trial. More detail into how much difficulty changes and what its effect will be discussed while explaining the microcontroller code.

One might notice that the patient only has to move their thumb on the potentiometer in order to move the actuator.

Table 3 below outlines the variables used in the Arduino code to execute the State machine and data collection. This Table can be used as a reference for the discussion on the programming of the state machine, in order to keep track of the amount of variables used when discussing the code.

Table 3: Variables in Arduino Code

Variable	**Type**	**Function**
pos	int	Reads patient input position. Fed into A0 pin of on-board ADC. Possible values 0-1023; 5 V max.
arm_pos	int	Linear Actuator position feedback. Fed into A1 pin of on-board ADC. Possible values: 0-680; 3.3 V max. Has inverse relationship as seen in Figure 5.
pull_dist	double	Holds minimum pull distance that the patient has to pull in order to move the actuator. Compared to *pos* variable; 1023 max.
pull_dist_max	double	Holds the maximum pull distance allowed, based on difficulty level. Compared to *pull_dist*, 1023 max.
distance	int	Holds of the amount that the linear actuator moves per successful pull. OCR2B uses this to act as the timer for the PWM; 255 max.
state	int	Holds the state number in order to which States as needed.
time	unsigned long	Uses the *millis*() function of the Arduino library to keep track of time in milliseconds. Needs to be an unsigned long in order to hold a large time value.
time_state	int	Holds the start time once a trial is initiated and State 1 → State 2.
current_time	int	Compares *time* to *time_start* to track how long a trial is taking.
max_time	int	Compared to *current_time* to see if trial is taking too long.
difficulty	int	Has an inverse relationship with *distance*, where distance = 255/difficulty; minimum value of 1.
diff_max	int	Compared to *difficulty*, holds the maximum value *difficulty* can be, determined by the overall difficulty level (*diff_level*).
diff_level	int	Holds difficulty level initiated by patient or therapist, values 1-5.
stateflag	int	Shifts between States 2a and 2b within State 2. Either 0 or 1.
doitonce	int	Flag to make sure initial conditions are only instigated once.
ser	int	Decodes serial inputs dictated by the patient or therapist.
attempt	int	Tracks the amount of unsuccessful attempts a patient has with a trial. Alters *difficulty* and *pull_dist*. Is reset once a successful trial is completed.

With the major variables of the Arduino code cataloged in Table 3, the code itself will be discussed in order to describe its function, and how it executes the state machine in Figure 15. The complete code can be found in the appendix at the end of this paper. The python test code can also be found in the appendix section. The first part of the Arduino code declares the variables outlined in Table 3 above, with their respective variable types. After this, the initial *void setup*() function of the Arduino code is called, as outlined in Figure 17 below.

```
void setup() {

  pinMode(3, OUTPUT);
  pinMode(11, OUTPUT);
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS22);

  Serial.begin(9600);


  OCR2B = 0; // pin 3 PWM
  difficulty = 0;    //"difficulty level", distance = 255/difficulty
  pull_dist = 205; //1023 = max 5 V, used to compare with pos
  state = 1; //state variable
  flag = 0; //flag to ensure actuator only moves once at a time
  attempt = 1; //tracks # of attempts
  max_time = 8000; //max time one can take w/ o doing anything in a trial (in milliseconds)
  pull_dist_max = 1023;
  diff_max = 30;

}
```

Figure 17: void setup() Function

The first four lines within the *void setup*() of Figure 17 format the PWM output used for controlling the linear actuator, on pin 3 of the Arduino. This is done by using one of the 8-bit timers on the ATMega328p, and using the *TCCR2A* timer, and setting the *OCR2B* compare register, where its max value of 255 corresponds with a 100% duty cycle [8]. *OCR2B* is initially set to 0 to fully retract the linear actuator. The *Serial.begin(9600)* function initiates serial communication through the USB port at a rate of 9600 baud. The rest of the *void setup*() function sets some of the variables outlined in Table 3 to their respective initial values for a trial, with commented explanations for each variable.

The *void loop*() function is then called and is the primary function used to run the device, as it repeats itself over and over until the device is reset. The first step in this function is to measure some key values that need to be constantly updated, and placing them at the beginning of the loop allows for that to occur and to be used later on. This is outlined in Figure 18 below.

```
void loop() {

  pos = analogRead(A0); //A0 pin = patient position input , 1023 = max 5 V

  arm_pos = analogRead(A1); //A1 pin = actuator position feedback, 678 = max 3.3 V

  time = millis(); //measures time in milliseconds
```

Figure 18: Initial Variables in void loop()

The *pos*, *arm_pos*, and *time* variables are the first to be updated in the *void loop()* as seen in Figure 18. From here, the code decodes any serial inputs that dictate the initial conditions of a trial. This process will be discussed later in the paper, however.

State 1, as discussed in Figure 18 above, acts to hold the actuator still until the patient initiates a movement. Once this happens, State 1 also sets the values that dictate the behavior of State 2. This process and the code for State 1 can be seen in Figure 19 below.

```
//*****STATE 1

  if(state == 1){
    OCR2B = 0;
    flag = 0;
    if( pos > pull_dist && arm_pos >= 678){
      state = 2;
      time_start = time;
      pull_dist =  pull_dist + 10;
      if(pull_dist > pull_dist_max){
        pull_dist = pull_dist_max;
      }
      difficulty = difficulty + 2;
      if(difficulty > diff_max){
        difficulty = diff_max;
      }
      distance = 255/difficulty; //how far actuator will move with each step
    }
  }
```

Figure 19: State 1 Code

While the *state* variable in Figure 19 equals 1, the device is in State 1. During this state, the output compare register OCR2B equals 0, resulting in a 0 % duty cycle PWM driving the linear actuator, fully retracting it. Once the patient moves their arm to generate a voltage with the potentiometer (measured with *pos*) at a large enough value (dictated by *pull_dist*), the trial effectively begins. The movement also cannot initiate the actuator until it is fully retracted and *arm_pos* (showing the position of the actuator) is equal or greater than 678 (at the 3.3 V of the actuator feedback signal). When the trial starts, the *pull_dist* and *difficulty* values are increased, which increases the effective difficulty of the trial, as the distance a patient needs to pull on the potentiometer increases, and the distance the linear actuator moves with each pull decreases. This is also done at the beginning of a trial to increase the difficulty over the course of a training session, as each time the patient begins a trial, it becomes harder. The difficulty can also be diminished if the patient has difficulty, which will be covered later in the report. The *time_start* variable is also set, keeping track of the starting time of the upcoming trial.

Once State 2 has been reached, it alternates between States 2a and 2b. State 2a increases the actuator position once the patient preforms a successful movement, and State 2b insures the actuator reaches that position to ensure the actuator does not stall. This is controlled with the *flag* variable within State 2. Also, if the patient does not move within the *max_time* allowed while in State 2, the trial fails and the device revers back to State 1. This process is outlined in the code in Figure 20 below.

```
//*****STATE 2 (plus 3&4)

  else if (state == 2 || state == 3 || state == 4){
    current_time = time - time_start;

    if(current_time > max_time ){ //unsuccessful trial
      state = 1;
      if(OCR2B < 250){
        difficulty = difficulty - 15*attempt;
        if(difficulty < 0){
          difficulty = 0;
        }
        pull_dist = pull_dist - 15*attempt;
        if(pull_dist < 50){
          pull_dist = 50;
        }
        attempt++;
      }
```

Figure 20: State 2 Unsuccessful Trial Code

When a trial is unsuccessful (when the *max_time* or time limit of a trial has been reached), an *attempt* variable is increased to keep track of the number of times a patient unsuccessfully completes a trial. The *attempt* variable is used to decrease the *difficulty* and *pull_dist* variables, effectively decreasing the overall difficulty of future trials, depending on how long the patient has trouble with them. If the patient fails many times in a row, this will significantly decrease the difficulty of the trials. The *attempt* variable is also reset if the patient successfully completes a trial (shown later in the report). The code in Figure 21 below outlines the process behind States 2a and 2b.

```
    if( (pos > pull_dist) && (flag == 0) ){
      if(OCR2B + distance > 255){
        OCR2B = 255;
      }
      else{
        OCR2B = OCR2B + distance; //increase actuator length is pos met
      }
      time_start = time;
      flag = 1;
      state = 3;
    }
    else if( (flag == 1) ){
      if(difficulty >= 20 && arm_pos >= 670 && (710 - (679/255)*OCR2B) >= arm_pos ){
           flag = 0;
      }
      else if( (679 - (679/255)*OCR2B) >= arm_pos ){
           flag = 0;
      }
       state=4;
    }

    if( (OCR2B >= 250) && (arm_pos < 10) ){ //successful trial
```

Figure 21: State 2a and 2b Code

While in State 2, State 2a is triggered when the patient pulls the potentiometer past the distance threshold (*pos > pull_dist*), as seen in Figure 21. Doing so increases the *OCR2B* (increasing the linear actuator position) and resets the *time_start* variable, resetting the amount of time the patient has to successfully pull. The *flag* variable is also set, which moves the device to State 2b. While in State 2b, the device predicts where the linear actuator will go based on the equation: 679 – (679/255)*OCR2B, and compares this to its actual actuator position: *arm_pos*. Once *arm_pos* has reached its predicted position, *flag* is reset and the device returns to State 2a. There is a potential glitch in this process, which occurs when the *distance* value is too low (i.e. when *difficulty* is greater than or equal to 20) to make a movement occur when the arm is fully retracted, effectively trapping the device in this state. This is curbed by altering the equation in this situation (when the actuator is fully retracted, *arm_pos* > 678 and *difficulty* is >= 20), which insures that the device does not get stuck in this state and will increase *OCR2B* until the device moves. Once the device moves beyond its rest state, this glitch does not occur and the device behaves as normal.

The last amount of code in State 2 determines if a trial is successful. This occurs when the position of the actuator, *arm_pos*, fully extends (approaches 0), and the *OCR2B* register approaches 255 for a 100% duty cycle. Once this happens, the device returns to State 1, and the attempt variable is reset as previously mentioned. This process is outlined in Figure 22 below.

```
if( (OCR2B >= 250) && (arm_pos < 10) ){
//successful trial

      state = 1;
      attempt = 1;
    }
```

Figure 22: State 2 Successful Trial Code

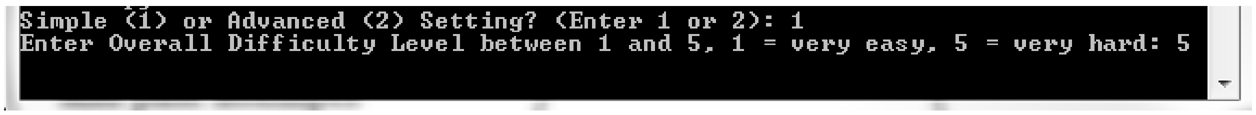
The remaining Arduino code to be discussed details the data that is sent to and from the device via serial. At the end of the *void loop*(), a series of *Serial.print*() functions are executed to send the noteworthy data variables over the course of a trial. This is also useful for testing and observing how certain variables change over a trial for debugging purposes. This process is outlined in the code in Figure 23 below.

```
     Serial.print(time);
     Serial.print('\t');
     Serial.print(pos);
     Serial.print('\t');
     Serial.print(arm_pos);
     Serial.print('\t');
     Serial.print(difficulty);
     Serial.print('\n');

 }
```

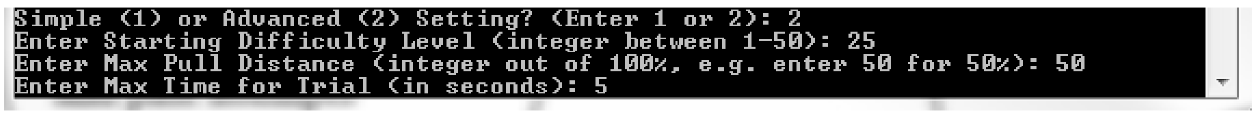
Figure 23: Serial Transmitting Code

As previously stated, a python script initially read these variables and acted as a user interface. The script prompted the user to enter the difficulty levels and general settings, before the script continuously read the variables from serial, with the intention of compiling the data once a trial finished. This continuous reading and printing of the variables gathered also allowed for effective debugging during the design process. Eventually, LabView was used instead for a cleaner user interface and more options. Nonetheless the test python script prompts and code can be seen in Figure 24 below for reference.

```
import serial


ser = serial.Serial('COM6', 9600,timeout=1)
ser.readline()

setup = input('Simple (1) or Advanced (2) Setting? (Enter 1 or 2): ')
if(setup == '1'):
    diff_level = input('Enter Overall Difficulty Level between 1 and 5, 1 = very easy, 5 = very 
hard: ')
    ser.write(bytes([50]))
    ser.write(bytes([int(diff_level)]))

else:
    difficulty = input('Enter Starting Difficulty Level (integer between 1-50): ')
    ser.write(bytes([10]))
    ser.write(bytes([int(difficulty)]))

    pull_dist = input('Enter Max Pull Distance (integer out of 100%, e.g. enter 50 for 50%): ')
    ser.write(bytes([20]))
    ser.write(bytes([int(pull_dist)]))

    max_time = input('Enter Max Time for Trial (in seconds): ')
    ser.write(bytes([30]))
    ser.write(bytes([int(max_time)]))


try:
  while True:
    line = ser.readline()
    print(line)


except KeyboardInterrupt:
    ser.close()
```



```
Simple (1) or Advanced (2) Setting? (Enter 1 or 2): 1
Enter Overall Difficulty Level between 1 and 5, 1 = very easy, 5 = very hard: 5
```



```
Simple (1) or Advanced (2) Setting? (Enter 1 or 2): 2
Enter Starting Difficulty Level (integer between 1-50): 25
Enter Max Pull Distance (integer out of 100%, e.g. enter 50 for 50%): 50
Enter Max Time for Trial (in seconds): 5
```

Figure 24: Python Test Code (Above) and User Prompts (Below)

Beyond receiving data from the Arduino from Figure 23, the LabView virtual instrument (vi) also acts as the interface with the patient or therapist before initiating a trial. The front panel of the vi displays a general and advanced tab. The general tab outlines basic controls and data collection for the potential patient or anyone wanting a quick trial. This includes setting the simple difficulty levels outlined below and in Table 4, and whether or not to generate a text file. The general settings tab also displays the graphs for the various variables collected throughout the previously run trial, and an LED labeled "program running," to alert the user when the program is on and running. These settings on the front panel can be seen on Figure 25 below.

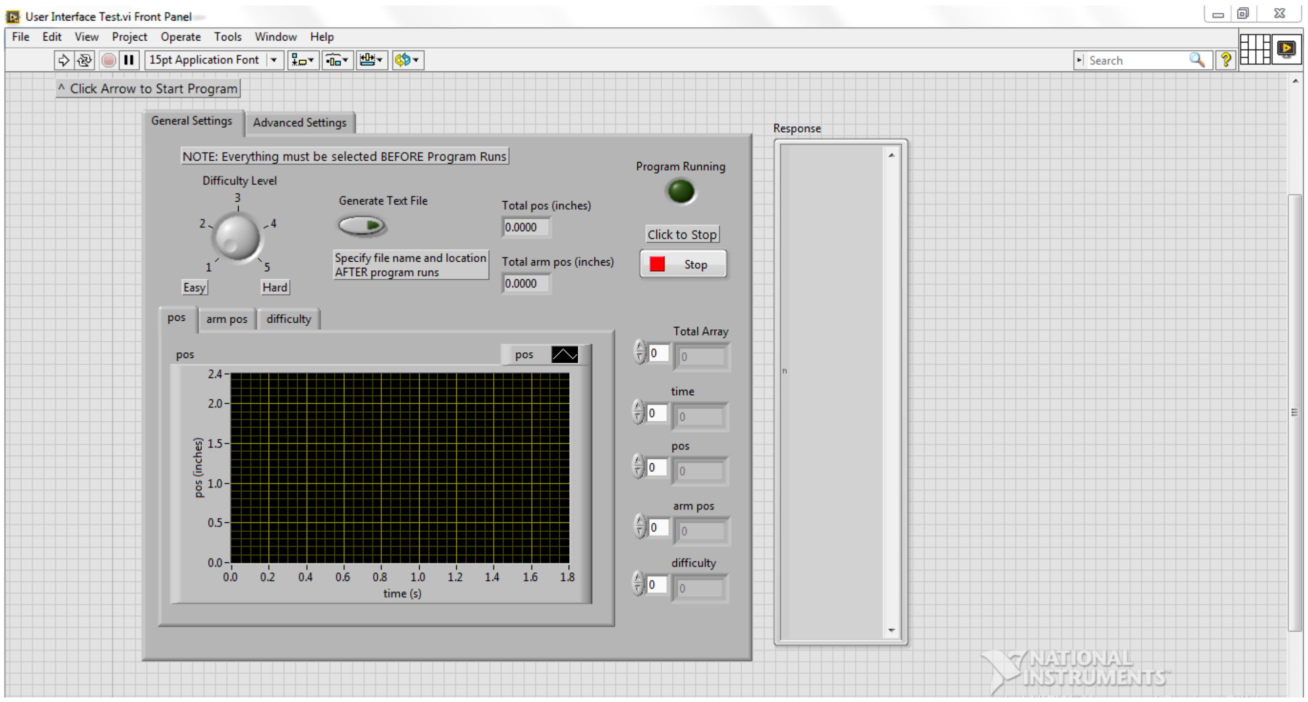


Figure 25: LabView Front Panel – General Settings

The simple difficulty levels vary from 1-5, with differences outlined in Table 4 below. Each level changes the minimum and maximum values of the *pull_dist* (possible values 0-1023), *difficulty* (needs to be at least 1), and *max_time* variables for a trial.

Table 4: Difficulty Levels

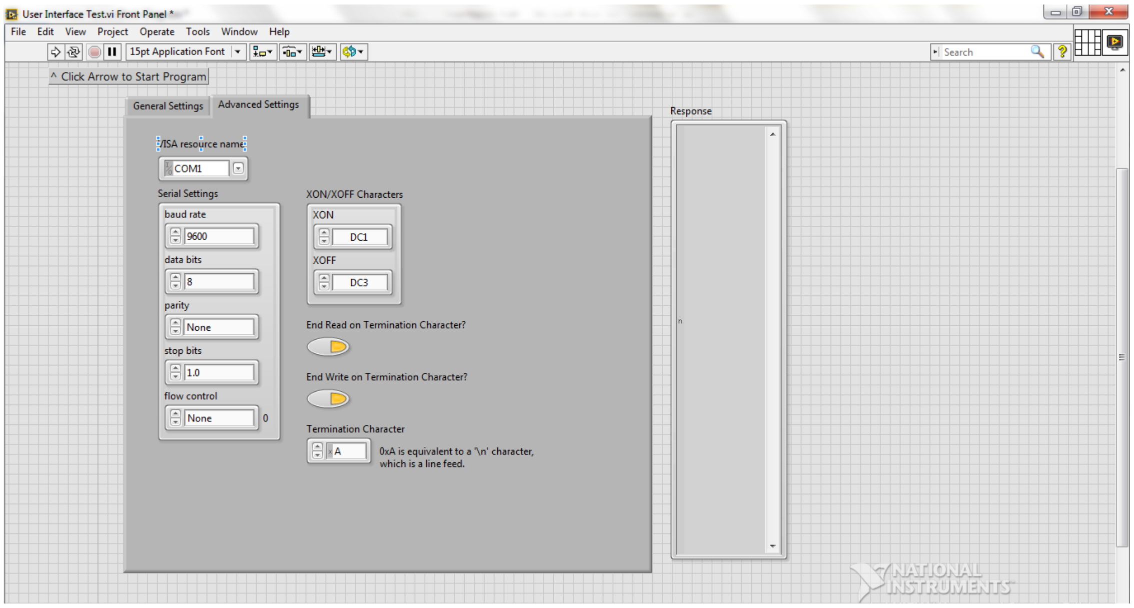
Difficulty Level	**Serial Letter**	**pull_dist values**	**difficulty values**	**max_time value**
1-"very easy"	A	50-205	1-10	8 seconds
2-"easy"	B	200-410	5-15	6 seconds
3-"normal"	C	410-610	10-20	4 seconds
4-"hard"	D	610-820	15-25	3 seconds
5-"very hard"	E	820-1020	20-30	2 seconds

In order to coordinate the difficulty levels illustrated in Table 4 above, the LabView vi continuously writes a letter to the Arduino via serial (as seen in the Serial Letter column in Table 4) that dictates the desired difficulty level. The Arduino reads this letter in ASCII, so "A" appears as 65, and so on. The reason for writing letters as a string instead of numbers is that the Arduino would often misinterpret numbers or instead use the new line character and alter the variable's value instead of keeping it constant. Since the variables in Table 4 need to only be altered once: when the trial starts, and since the vi continuously writes, the variable *doitonce* checks for the initial update of the variables required in Table 4, and ensures the variables do not update anymore and deviate from the changes made throughout the trial. The Arduino code for such reading and interpreting the difficulty levels can be seen in Figure 26 below.

```
ser = Serial.read();

  //Overall Difficulty Level
  if(ser == 65 && doitonce == 0){      //Level 1, A
    pull_dist = 50;
    pull_dist_max = 205;
    difficulty = 2;
    diff_max = 10;
    max_time = 8000;
    doitonce = 1;
  }
  else if(ser == 66 && doitonce == 0){ //Level 2, B
    pull_dist = 200;
    pull_dist_max = 410;
    difficulty = 5;
    diff_max = 15;
    max_time = 6000;
    doitonce = 1;
  }
  else if(ser == 67 && doitonce == 0){ //Level 3, C
    pull_dist = 410;
    pull_dist_max = 610;
    difficulty = 10;
    diff_max = 20;
    max_time = 4000;
    doitonce == 1;
  }
  else if(ser == 68 && doitonce == 0){ //Level 4, D
    pull_dist = 610;
    pull_dist_max = 820;
    difficulty = 15;
    diff_max = 25;
    max_time = 3000;
    doitonce = 1;
  }
  else if (ser == 69 && doitonce == 0){ //Level 5, E
    pull_dist = 820;
    pull_dist_max = 1020;
    difficulty = 20;
    diff_max = 30;
    max_time = 2000;
    doitonce = 1;
  }
```

Figure 26: Arduino Serial Transmitting Code

The LabView wiring diagram in Figures 28 and 29 below alters the LabView example “Continuous Serial Write and Read.” The visa port settings are specified to open the serial port, and a while loop with two case statements allowing for a continuous writing and reading of the serial port. The settings to open the serial port are under the advanced settings tab in the vi’s front panel. The default values for these settings usually work with the Arduino; however the USB port where the Arduino is attached to the computer often needs to be specified. The advanced settings tab of the front panel and the serial communication section of the vi’s wiring diagram described can be seen in Figures 27 - 29 below.


User Interface Test.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Click Arrow to Start Program
General Settings
Advanced Settings
Response
NATIONAL INSTRUMENTS

VISA resource name: COM1

Serial Settings
baud rate: 9600
data bits: 8
parity: None
stop bits: 1.0
flow control: None 0

XON/XOFF Characters
XON: DC1
XOFF: DC3

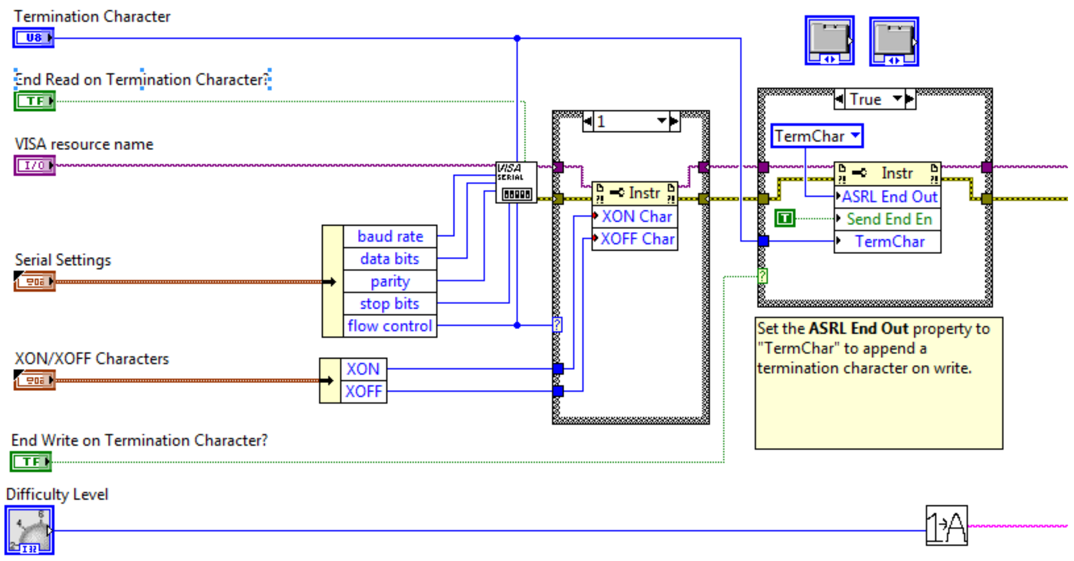
End Read on Termination Character?

End Write on Termination Character?

Termination Character: A

0xA is equivalent to a '\n' character, which is a line feed.

Figure 27: LabView Front Panel Advanced Settings Tab



Set the **ASRL End Out** property to "TermChar" to append a termination character on write.

Figure 28: LabView Wiring Diagram – Serial Port Initiation

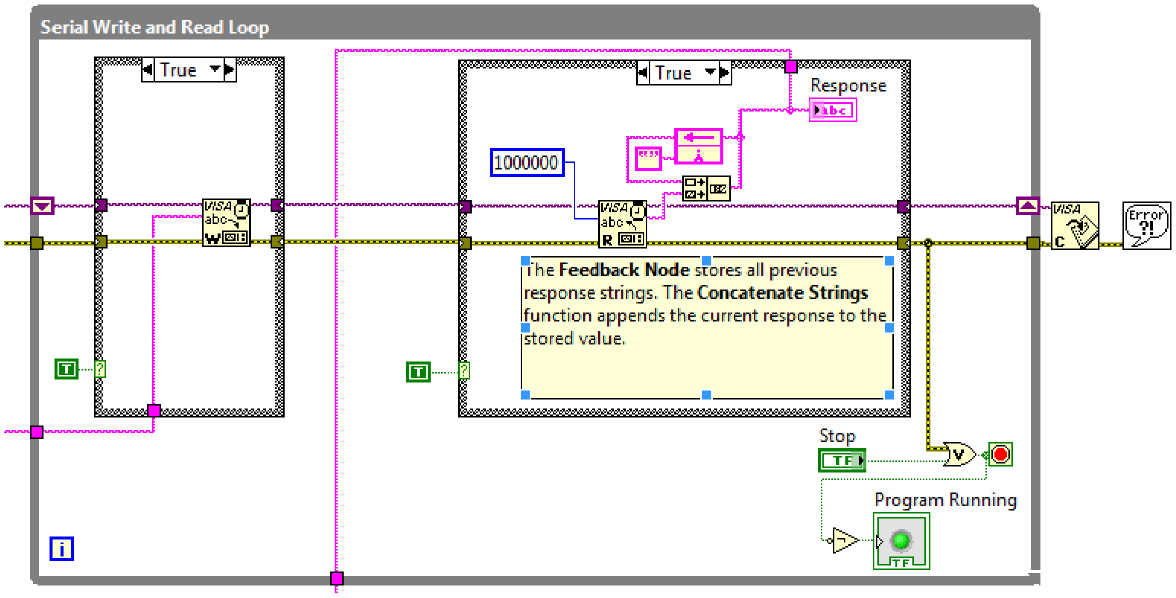

Serial Write and Read Loop
True
True
1000000
Response
The Feedback Node stores all previous response strings. The Concatenate Strings function appends the current response to the stored value.
Stop
Program Running

Figure 29: LabView Wiring Diagram – Serial Write and Read

After the continuous serial write and read initiated from the LabView vi, and once the operator clicks the STOP button on the front panel, the vi complies the data collected. The STOP button is also connected to the LED on the front panel labeled "program running," meaning when the STOP button is pressed, the LED is turned off and the user is alerted that the program is no longer running. Of course the program actually does continue to run to analyze the data, but this happens so quick the user will likely not notice when the program actually stops. To compile the data, the vi first converts the long string response into a workable array for data processing. The LabView example "Search String for Numbers" was altered to search the long string response from the continuous write and read, find the numbers, and organize them as an array, removing extra characters such as tabs and the 'new line.' This generates a 1D array of values. This altered LabView example in the wiring diagram can be seen in Figure 30 below.

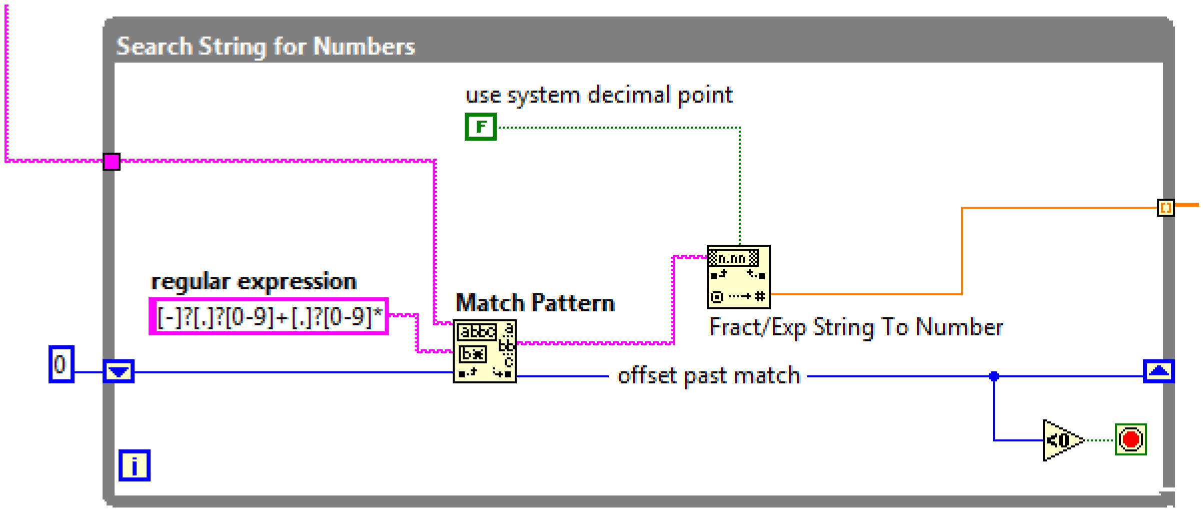
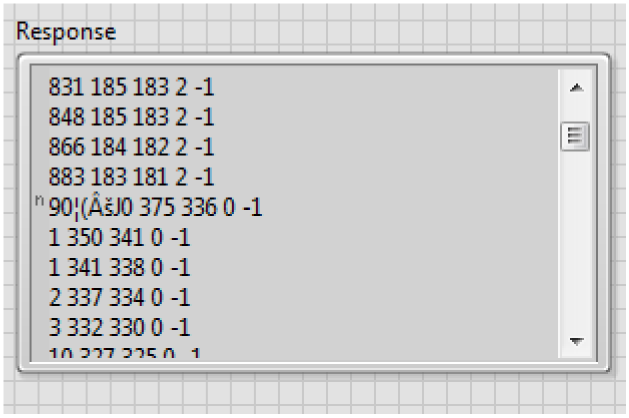


Figure 30: LabView Wiring Diagram – Search String for Numbers

Once the data is in a presentable array, the variables gathered need to be separated. Upon starting up, the Arduino serial values often show random, unworkable values in the beginning, and these values need to be filtered out. This is done by splitting the array where ‘1’ appears, which occurs once the Arduino’s *millis()* function begins counting the time. If a ‘1’ appears when the Arduino starts up in the “random junk” section, then this method will not work, however this event is unlikely. Figure 31 below outlines the initial junk that appears in the serial communication and where the ‘1’ representing the start of the trial is.



```
831 185 183 2 -1
848 185 183 2 -1
866 184 182 2 -1
883 183 181 2 -1
n 90¦(ÅšJ0 375 336 0 -1
1 350 341 0 -1
1 341 338 0 -1
2 337 334 0 -1
3 332 330 0 -1
10 327 325 0 -1
```

Figure 31: Initial Serial Data

Once the array is separated at the ‘1’, each variable is separated using the “Decimate Array” sub vi. This sub vi separates every four variables, which coordinates with how the data was written to serial. Some of the variables are then converted to workable units. The *time* variable is divided by 1000 to convert the milliseconds to seconds. The *pos* is multiplied by $\frac{19}{8184}$, in order to convert the ADC analog reading integer value to the distance the potentiometer moves in inches. This was found by noting that the potentiometer moves 2 3/8” total, and that the ADC value varies 0-1023, thus dividing 2.375/1023 for a conversion factor. The *arm_pos* variable is divided by -160 and added to 4.25, converting its ADC read value to the amount of inches the actuator moves. The *arm_pos* is divided by a negative number as the *arm_pos* as an inverse relationship between the distance moved and *arm_pos* value. And 4.25 is added as the actuator moves 4.25” at most.

From here the data of each individual variable is bundled with time and graphed, with each variable given its separate graph. These graphs are organized with a tab in the general settings on the front panel of the vi. These variables could not appear on the same graph as the units and typical range of values are all too different for them to be presentable on the same graph. While the tabs might be cumbersome to click through in order to compare different variables at different times, separating them creates less clutter than having four graphs next to each other.

The total distance moved for both the *pos* and *arm_pos* are calculated by integrating the data of each value over time. This is done by first finding the mean of the differences in the *time* variable first. A for loop uses a shift register to compile an array of the differences between each time data point, and the mean is found from that array. This mean serves as the ‘dt’ input for the integration function, with both the *pos* and *arm_pos* given their own function. The data analysis process can be seen in Figure 32 below, including the separation of variables, conversion into workable units, graphing, and integrating.

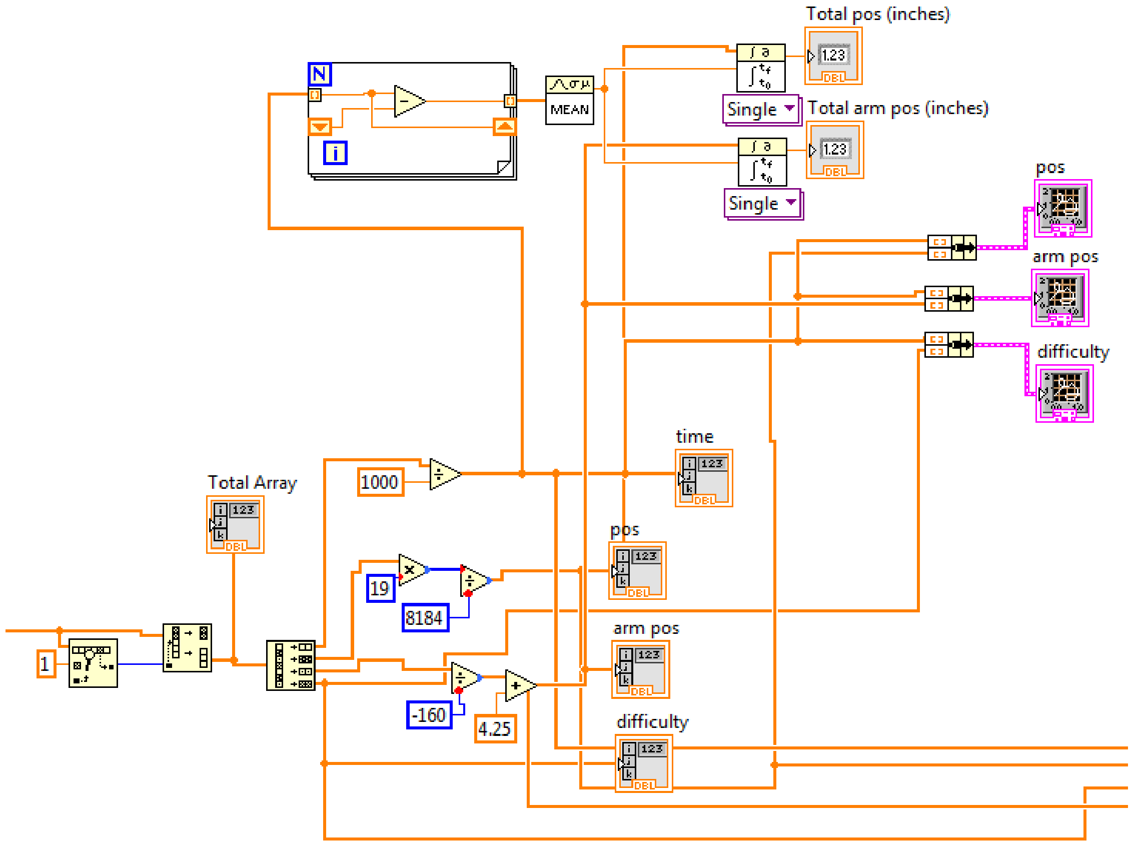


Figure 32: LabView Wiring Diagram – Data Analysis

Generating a text file of the data collected requires the user to push a button on the general settings tab on the front panel of the vi. Doing so activates the case structure in the wiring diagram that combines the different variables back into one array, converts it into a string, and generates a text file from it. The file is also given a title row with the name of each variable in the order they appear in each row. The case statement of this process is outlined in the section of the wiring diagram in Figure 33 below, and an example text file can be seen in Figure 34.

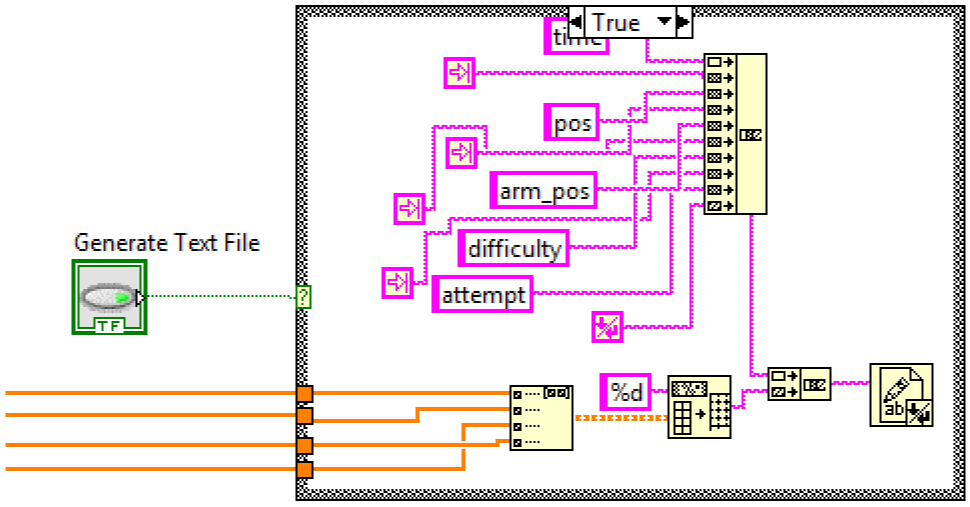
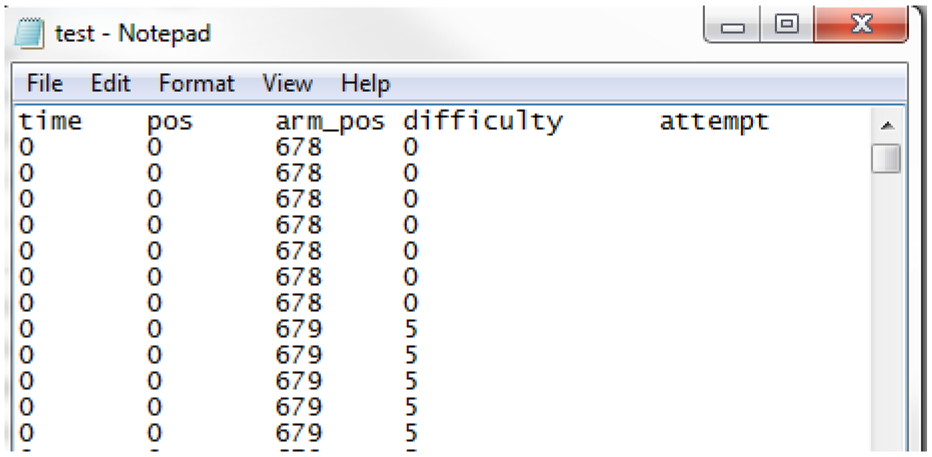


Figure 33: LabView Wiring Diagram –Text File Generation



```
time      pos       arm_pos difficulty        attempt
0         0         678     0
0         0         678     0
0         0         678     0
0         0         678     0
0         0         678     0
0         0         678     0
0         0         678     0
0         0         679     5
0         0         679     5
0         0         679     5
0         0         679     5
0         0         679     5
```

Figure 34: Example Text File

Chapter 5 – Conclusions

The project has researched and identified the properties of past works, in order to properly identify the potential of this project. The linear actuator used has been characterized in order to accurately design around the new part. The state machine has been designed and detailed, with the stability of the linear actuator guaranteed. A sliding potentiometer adds a way for the user to move the actuator, and having the potentiometer mounted on a wood block gives the setup some stability. LabView replaced python as a user interface, and adds more functionality and options to the user while keeping an approachable visual presence. Table 5 below details the specifications of the final product, comparing it with the initial specifications found in Table 1. The reasoning behind these specifications can also be found in Table 1.

Table 5: Stroke Therapy Device Final Results

Marketing Requirements	Initial Engineering Specifications	Final Result
1, 5	Should take less than 5 minutes to begin a trial on average	Can take less than 5 minutes on average, only depending on the speed of the software (or CPU hardware) and find comfortable position of device
2, 3	Increases the reaching motion length of the average patient during a trial by at least 10%	Has not been extensively studied, but can increase movement by at least 10 %
5, 6	Powers supplied by Microcontroller power source and/or 9 Volt battery	Voltage and Current draw too high for typical 9 V battery, requires external 12 V source
5, 6	Entire system weighs less than 20 lbs.	Entire system (other than external CPU) weighs around 5 lbs. maximum
1, 5, 6	The apparatus the patient moves weighs less than 5 lbs.	The apparatus the patient moves weighs around 1 lbs
1, 3, 4, 7	Reads distance of brace (in) and force (N) applied or exerted and compiles data within 1 min of a trial	Device reads distance of brace in inches, force exerted is not read. Complies data in on average less than a 1min of trial (time depending on the length of the trial and amount collected)
5, 7	Can turn off the system within 1second of initiating stop	Gives safety measure for unforeseen events during a trial
1, 7	Can alter initial conditions (max/min variables, time of trial, difficulty level) within 5 % accuracy	Each initial condition specified in Table 3 is altered within 5 % accuracy
6	At most 1’ X 3’ X 2’(when fully extended)	When fully extended, the system is 10” X 18” X 2” (excluding wires and external CPU)
Marketing Requirements: 1. Easy to use 2. Offers better treatment and use of arm 3. Adapts to needs 4. Able to collect data 5. Safe to use 6. Small and portable 7. Offers control for the therapist and patient		

Comparing Table 5 to Table 1 demonstrates that the final product meets most of the initial requirements, with the exception of the power consumption and measuring the force exerted by the patient and applied to the patient via the actuator. These areas allow for the most amount of improvement.

Other Improvements on the project include adding more options through LabView, such as altering individual variables, or generating an excel spreadsheet with data. An indication for when the linear actuator retracts, such as an LED, can tell the patient when the device reverts to State 1 and the patient can't move the device. Bugs in the LabView data collection also needs to be solved. Above all else, the effectiveness of this device via human trials should occur to test the validity of the device. Even if the device is not as successful as it could be, the concepts behind this paper may prove useful as a discussion around the design considerations of robotically assisted therapy.

References

[1] Wikipedia, “Stroke”, 2015. [Online]. Available: https://en.wikipedia.org/wiki/Stroke. [Accessed: 19-Oct- 2015].

[2] S.-C. Yeh et al, “Evaluation Approach for Post-stroke Rehabilitation Via Virtual reality Aided Motor Training”, M.J. Dainoff , *Ergonomics and Health Aspects*, pp. 378–387, 2007.

[3] M. Holden et al, “Telerehabilitation Using a Virtual Environment Improves Upper Extremity Function in Patients With Stroke”, *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 15, no. 1, pp. 36-42, 2007.

[4] D. Reinkensmeyer et al, "A Pneumatic Robot for Re-Training Arm Movement after Stroke: Rationale and Mechanical Design", *Proc. IEEE 9th Int. Conference on Rehabilitation Robotics*. pp. 500-504, 2005.

[5] D. Reinkensmeyer et al, “Web-based telerehabilitation for the upper extremity after stroke”, *IEEE Trans. Neural Syst. Rehabil*. Eng., vol. 10, no. 2, pp. 102-108, 2002.

[6] R. Ford and C. Coulston, Design for Electrical and Computer Engineers, McGraw-Hill, 2007, p. 37

[7] *8-bit Atmel Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash,* Atmel Co., San Jose, CA, 2011.

[8] Linear Motion Series: L12, Firgelli Technologies Inc., Victoria, BC, Canada, 2008.

[9] “Wobbling interface e.g. Steward platform or Hexapod,” U.S. Patent A61H 2001/0207, January 2013.

[10] Griffith University, "Average computer energy usage." [Online]. Available: https://www.griffith.edu.au/sustainability/sustainable-campuses/sustainable-initiatives/energy/average-computer-energy-usage [Accessed: 25- May- 2016].

[11] IEEE, “IEEE Code of Ethics”, 2015. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: 31- Oct- 2015].

[12] J. Rawls, “A Theory of Justice,” Rev. ed. Cambridge, MA: Harvard University Press, 1999.

Appendix A. Programming Code

Arduino Code:

```
int pos;
int arm_pos;
double pull_dist;
double pull_dist_max;
int state;
unsigned long time;
int time_start;
int current_time;
int distance;
int difficulty;
int diff_max;
int diff_level;
int stateflag;
int attempt;
int ser;
int max_time;
int doitonce;

void setup() {
  pinMode(3, OUTPUT);
  pinMode(11, OUTPUT);
  TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
  TCCR2B = _BV(CS22);
  Serial.begin(9600);


  OCR2B = 0; // pin 3 PWM
  difficulty = 0;    //"difficulty level", distance = 255/difficulty
  pull_dist = 205; //1023 = max 5 V, used to compare with pos
  state = 1; //state variable
  stateflag = 0; //flag to ensure actuator only moves once at a time
  attempt = 1; //tracks # of attempts
  max_time = 8000; //max time one can take w/ o doing anything in a trial (in
milliseconds)
  pull_dist_max = 1023;
  diff_max = 30;
  doitonce = 0;

}

void loop() {
  pos = analogRead(A0); //A0 pin = patient position input , 1023 = max 5 V
  arm_pos = analogRead(A1); //A1 pin = actuator position feedback, 670 = max
3.3 V
  time = abs(millis()); //measures time in milliseconds

  ser = Serial.read();

  //Overall Difficulty Level
  if(ser == 65 && doitonce == 0){      //Level 1, A
    pull_dist = 50;
    pull_dist_max = 205;
    difficulty = 2;
    diff_max = 10;
    max_time = 8000;
    doitonce = 1;
  }
  else if(ser == 66 && doitonce == 0){ //Level 2, B
    pull_dist = 200;
    pull_dist_max = 410;
    difficulty = 5;
    diff_max = 15;
    max_time = 6000;
    doitonce = 1;
  }
  else if(ser == 67 && doitonce == 0){ //Level 3, C
    pull_dist = 410;
    pull_dist_max = 610;
    difficulty = 10;
    diff_max = 20;
    max_time = 4000;
    doitonce == 1;
  }
  else if(ser == 68 && doitonce == 0){ //Level 4, D
    pull_dist = 610;
    pull_dist_max = 820;
    difficulty = 15;
    diff_max = 25;
    max_time = 3000;
    doitonce = 1;
  }
  else if (ser == 69 && doitonce == 0){ //Level 5, E
    pull_dist = 820;
    pull_dist_max = 1020;
    difficulty = 20;
    diff_max = 30;
    max_time = 2000;
    doitonce = 1;
  }


  //*****STATE 1
  if(state == 1){
    OCR2B = 0;
    stateflag = 0;
    if( pos > pull_dist && arm_pos >= 678){
      state = 2;
      time_start = time;
      pull_dist =  pull_dist + 10;
      if(pull_dist > pull_dist_max){
        pull_dist = pull_dist_max;
      }
      difficulty = difficulty + 2;
      if(difficulty > diff_max){
        difficulty = diff_max;
      }
      distance = 255/difficulty; //how far actuator will move with each step
    }
  }
  //*****STATE 2 (plus 3&4)
  else if (state == 2 || state == 3 || state == 4){
    current_time = time - time_start;

    if(current_time > max_time ){ //unsuccessful trial
      state = 1;
      if(OCR2B < 250){
        difficulty = difficulty - 15*attempt;
        if(difficulty < 0){
          difficulty = 0;
        }
        pull_dist = pull_dist - 15*attempt;
        if(pull_dist < 50){
          pull_dist = 50;
        }
        attempt++;
      }
    }


    if( (pos > pull_dist) && (stateflag == 0) ){
      if(OCR2B + distance > 255){
        OCR2B = 255;
      }
      else{
        OCR2B = OCR2B + distance; //increase actuator length is pos met
      }
      time_start = time;
      stateflag = 1;
      state = 3;
    }
    else if( (stateflag == 1) ){
      if(difficulty >= 20 && arm_pos >= 670 && (710 - (679/255)*OCR2B) >=
arm_pos ){ //fix for glitch with getting actuator to move at higher
difficulty
          stateflag = 0;
      }
      else if( (679 - (679/255)*OCR2B) >= arm_pos ){ //making sure actuator
reaches destination
          stateflag = 0;
      }
       state=4;
    }

    if( (OCR2B >= 250) && (arm_pos < 10) ){ //successful trial
      state = 1;
      attempt = 1;
    }

  }

    Serial.print(time);
    Serial.print('\t');
    Serial.print(pos);
    Serial.print('\t');
    Serial.print(arm_pos);
    Serial.print('\t');
    Serial.print(difficulty);
    Serial.print('\n');

}
```

Python Code:

```
import serial

ser = serial.Serial('COM6', 9600,timeout=1)
ser.readline()

setup = input('Simple (1) or Advanced (2) Setting? (Enter 1 or 2): ')
if(setup == '1'):
    diff_level = input('Enter Overall Difficulty Level between 1 and 5, 1 =
very easy, 5 = very hard: ')
    ser.write(bytes([50]))
    ser.write(bytes([int(diff_level)]))

else:
    difficulty = input('Enter Starting Difficulty Level (integer between 1-
50): ')
    ser.write(bytes([10]))
    ser.write(bytes([int(difficulty)]))

    pull_dist = input('Enter Max Pull Distance (integer out of 100%, e.g.
enter 50 for 50%): ')
    ser.write(bytes([20]))
    ser.write(bytes([int(pull_dist)]))

    max_time = input('Enter Max Time for Trial (in seconds): ')
    ser.write(bytes([30]))
    ser.write(bytes([int(max_time*1000)]))


try:
  while True:
    line = ser.readline()
    print(line)


except KeyboardInterrupt:
    ser.close()
```

Appendix B. ABET Senior Project Analysis

1. Summary of Functional Requirements

The project allows stroke patients or people who cannot move their arm during physical therapy to move their arm more effectively using an apparatus to train their arm for rehabilitation. The system offers a more full use of the arm than traditional physical therapy, as well as adapting to the needs of the patients. Trials increase in difficultly over time, but still become easier if the patient has trouble. Physical therapists control the general difficulty before starting a trial, and gather data during a trial for future use. The device also offers safety, portability, and easy use.

2. Primary Constraints

Finding a proper motor system that would provide a linear movement for the device provided the first major hurdle of the design process. The motor in the original internship project could not be found in a timely fashion, so a substitute had to be found and characterized to fully understand and design around its performance. Creating a compact system also caused issues in deciding parts and limiting the power and scope of the hardware involved. Buying and shipping these parts within the timeline and time constraints of the Cal Poly quarter system also limited the scope of the project. The design of a training algorithm to effectively address the needs of the patient proved to be the most significant challenge of this project. Not only did this process take the most amount of time to design and test, but it also affected and dictated the function of the rest of the device. For instance scalping the user interface required knowledge of the training algorithm and device capabilities. An initial plan for the user interface was also abandoned during the design process once a preferable substitute was found, but this did not cause much difficulty as there was a good amount of time left.

3. Economic

The human capital of this project includes the design engineers, and those who would benefit most from the system, such as patients, physical therapists, and researchers. Their knowledge, findings, and desires drive the design for this project, and ultimately decide the practicality and longevity of the project into the future. The manufacturing personnel who provided the parts for the project also contribute to the human capital. The various components this project requires make up the main cost of the project, since labor costs do not add to the overall cost. The financially involved also include the companies and manufacturers who supply these parts. Any profitability of the project can lead to an increase in production for the product, which results in profitability for the manufacturers. The shipping companies and their resulting costs also contribute. The microcontroller, linear actuator, wires, wood blocks, and the external computer to run the user interface software for the apparatus make up the manufactured capital. All of these use the Earth's resources in varying degrees. The power requirements of the system also consume Earthly resources. The system uses a DC power supply, which has its own environmental and economic concerns surrounding their fabrication, resources, efficiency, and disposal. The farming and mining of these Earthly resources have an effect on the environment and species where these materials reside. Changes in these areas can affect how much or how sustainable each material is.

Most of the costs of this project accumulate during the design process. Purchasing and shipping parts requires time and money, and additional parts accumulated throughout the design process. The companies involved in manufacturing and shipping the parts have their own costs during this process. The benefits of the project do not occur until the project nears completion. If certain organizations wish to use the project for their own benefit or research, then some income for the project will accumulate.

The inputs of the project include the labor and time costs of the design process, the manufacturing and parts costs, the documentation, and upkeep of the project. The costs estimates can be seen in Table 6 below, with the exclusion of monetary labor costs. Table 6 uses following equation in Ford and Coulston for the Expected cost.

$$Expected\ Time = \frac{Optimistic\ Time + 4 \times Realistic\ Time + Pesimisstic\ Time}{6} \quad [6]$$

Table 6: Initial Cost Estimation

Part	Optimistic Cost ($)	Realistic Cost ($)	Pessimistic Cost ($)	Expected Cost ($)
Electric Motor	10	50	100	51.67
Arm Brace	20	50	80	50
Wheels	5	20	50	22.5
Casings	20	50	150	61.67
Total Cost ($)				185.84

The actual cost accumulated over the project can be seen in Table 7 below.

Table 7: Final Cost

Part	Cost ($)
Linear Actuator	110.54
Potentiometer	9.80
Wire	10.79
PCB	2.69
Wood Blocks	Free (Scrap)
Total Cost	133.82

Comparing Table 6 to Table 7, the initial cost estimate came to $185.84, and the actual cost for the finished product came to $133.82. This ignores the cost of the microcontroller, which the designer already had before the project began, but whose cost is around $25. This make the total cost of the project around $158.82. The engineers designing and testing the project handle the major costs of the project during the initial design process. The design process required various kinds of equipment, such as DC power supplies, multimeters, and oscilloscopes. Since the design process occurred at Cal Poly San Luis Obispo, where these devices are readily available, they did not add any additional costs.

How much the product earns relies on how the product is sold, which is discussed more in the next section. However, if sold at the $250 rate, with the $158.82 production cost, and with an estimated 100 units sold annually, the net profit would yield $9,118. The designer, therapist or research companies that endorse this product would profit.

Products can emerge late into the design process, if human testing takes place during the design process to receive feedback on the project’s functionality. Operation costs would come in the form of power requirements, for both the device and computer required to run the user interface. Maintenance may need to occur for the wood blocks that weigh down the linear actuator and potentiometer, The LabView protocol could potentially be updated, or have later installments. Replacement parts for the actuator, potentiometer, or even microcontroller may also require consideration. The design process for this project was influenced by the Cal Poly quarter system, nonetheless the expected development time and tasks can be seen in the Gantt chart in Figure 35 below, with the actual timeline in Figure 36.

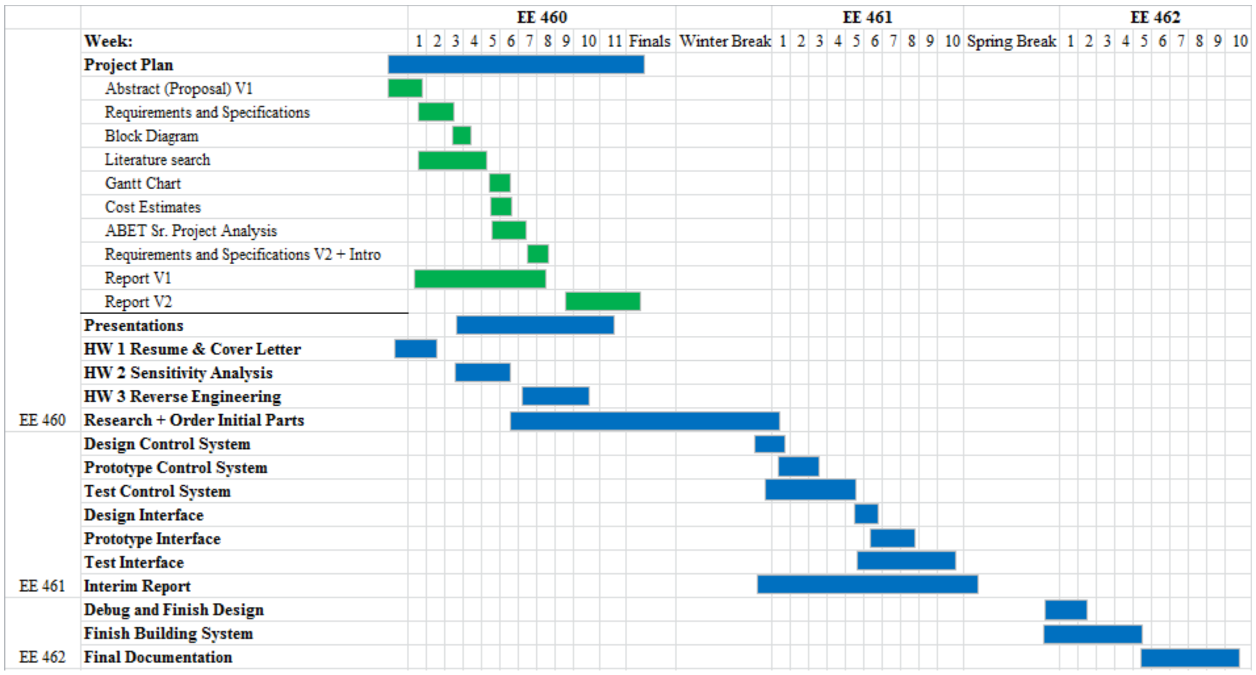


Figure 35: Expected Timeline Gantt Chart

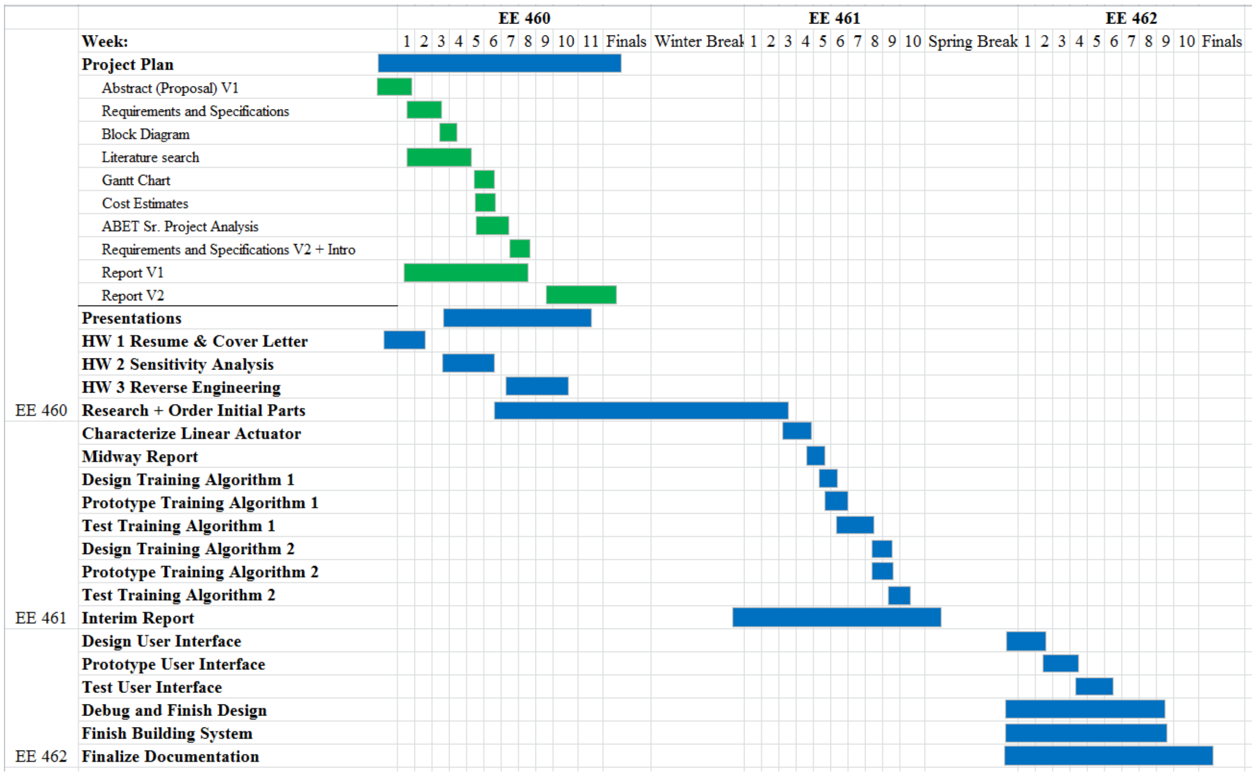


Figure 36: Actual Timeline Gantt Chart

Comparing Figure 35 to Figure 36, the work in EE 461 focused primarily around finding and characterizing the linear actuator, as well as going through two design/build/test cycles to optimize the training algorithm and the programming of the microcontroller. Two reports were also required in EE 461 in Figure 36, not one in Figure 35. EE 462 was mostly spent around the user interface and finishing the loose ends of the project.

After the project ends, the report goes into the digital commons of the Cal Poly library, to be seen by future students for reference or to inspire their future projects. Cal Poly will also have access to the software and programs used for this design. If researchers or therapists find the project and deem it worthy to get a hold of a prototype or seek out and invest in similar projects, then the lifespan of this project or similar projects will persist.

4. If Manufactured on a Commercial Basis

If this project becomes available on a commercial basis, then the main customers of the project as outlined previously are the main beneficiaries. If the project receives support from organizations for research or to further this project or similar projects, then this anticipation may occur sooner. However, only a few organizations may initially consider giving projects like this support, even if the project is considered product ready. As such, the projected number of devices sold per year expects around 100 initially. This project may also inspire future similar projects and products to appear, if successful. Since little specific patents on stroke therapy of this nature currently exist, this stands as a greater potential market [9].

Since the individual components of the project come from their respective manufacturers, the components continued presence in the market allows for the continuation of this project. From these components, the manufactured cost came out to $133.82 at the end of the design process. Because this requires acquiring the rights from the companies who manufactured these parts, designing and reverse engineering these parts for a commercial basis seems like a more viable solution to reduce the cost per unit. With the manufactured cost above, however, and with the few amount of products sold, the cost of each individual device would likely fall in the $250 range, resulting in a net annual profit of:
($250 – $158.82) × 100 units = $9,118.

The power required to work the device determines the cost for the user to operate the device. Both a DC 12 V supply and external computer (which the user is likely to already have) are required. Thus the cost to operate the device comes from the cost used to power these instruments. With the average power consumption of the project estimated at 0.12 W (from 12 V and 100 mA draw) and the average power consumption of a computer being 130 W [10], and for a time interval of an hour a day, three days a week, for a total three hours, the energy usage comes to 0.39036 kWh per week or 1.5624 kWh per month. The actual cost to run, would therefore be determined by the user’s cost of power in their home.

5. Environmental

The environmental manufacturing costs mainly come from the companies who design and manufacture the parts needed for the project. The IC’s and PCB’s used to create the microcontroller and inner workings of the linear actuator require raw materials such as silicon and packaging materials, causing an environmental concern for the manufacturers. The material used for the linear actuator arm also has its own manufacturing environmental concerns based on the eventual material used. The actual resources and specific environments and species affected become difficult to determine as they vary between the companies involved. Shipping costs and transportation fuel also affect the environment based on the companies, locations, and vehicles used for these steps. The wood blocks require the wood from trees, which can affect the environment and various species through deforestation. The power the system requires causes the main environmental concern as the project requires power during and after the design process. Originally power outlets, function generators, computers, and oscilloscopes that need wall outlets for the design and testing process consume power. The power each of the tools use and their environmental concerns from the power companies involved supplement the interests for this project. With most of the testing occurring at Cal Poly San Luis Obispo, both the university and PG&E become responsible for these costs.

After the design process, the project requires similar power needs. The 12 V supply for the linear actuator could come from an external DC supply, or a 12 V battery that would supply adequate amount of current levels. Both cases require more raw materials and chemicals to construct, and each has different energy efficiencies and performance levels which affect the longevity and sustainability of the supply. Each addition of new materials required affects the environments and surrounding species that supply these resources. All of the concerns raised ultimately harm the environment, as most deplete Earthly resources and offer little sustainability. The wood blocks used to mount the device can be a sustainable resource, if farmed properly.

6. Manufacturability

The manufacturing issues derived mainly from the companies who manufacture the components for the project. Since the companies sell these products on a commercial basis, as long as they are on the market, most of the manufacturing planning and sustainability concerns are of secondary. Choosing the right components to fit within the dimensions specified in Table 1 offered some concern. The wood blocks used came from a scrap pile of a local mill, but the criteria to choose them relied on their weight, and potential friction on a table or flat surface.

7. Sustainability

Since the project mostly consists of components from various companies, as long as these products, or similar products, are on the market, the project could continue to be produced and allow for replacement parts. Reverse engineering or designing original components that offer similar functionality may be a viable route to defend against these components going out of sale. It also offers a more thorough understanding on how these components work, and how to properly debug or address issues in the design. Currently the device is fairly durable with its individual components; however wear may affect the wood blocks used for the potentiometer and to weigh down the linear actuator, such that they may require replacements. The sliding potentiometer also sees a fair amount of human interaction, and is thus more likely to break and require a replacement.

Since the project mostly consists of components from various companies, the sustainability of the project depends on the materials that make up these components. If each material used does not offer sustainable production, it hampers the overall sustainability. Currently most of the materials used are raw materials with limited Earthly resources. The power and size concerns of the design process affect the sustainability of the project. The project is portable, and thus fairly small and able to fit on a desk, but the current size may not be an economic use of resources. The power limitations may hamper the sustainability of the project if the sources used are not sustainable. The supply required to run the external CPU may use inefficient or non-sustainable methods, depending on the utility company or sources used.

Improvements can be made in creating more efficient power usage, and determining a 12 V supply that operates on an efficient and sustainable manner. More research can be made into the different components that make up the system, and whether a more sustainable option is present. Reverse engineering the various components required would also allow for preferable upgrades, however doing so may be costly.

8. Ethical

According the IEEE Code of Ethics [11], this system ensures that the health and safety of those using the system falls under the concerns of the main designer. Exaggeration of the findings and effectiveness of the system or mishandling information in order to give the project more notoriety does not occur during the production of this project. No bribery affects the development of the project. No discrimination of different customs, cultures, or orientations inherently exists with the project, and assistance from colleagues occurred when relevant.

Using Rawls' Contractarianism [12], this project upholds equal liberty as its intention does not discriminate against anyone. Stroke patients that the project seeks to aid get the most out of the project capabilities, however. Limitations may occur based on the society this product enters and how well received the project or similar projects become in the medical community. These limits may include the prescription, insurance availability, or limited researchers or therapists who use the device. Individual economic status may also affect who can use the device. In practicality, not everyone has the most immediate access to the project initially, but in theory, the project does not inherently discriminate against anyone in its design. Since the project helps people move their affected arm more effectively, the project gives people a better chance to advance in society. It more accurately addresses the disparaging rights of people affected by stroke or equivalent by offering differing or more effective treatment. If this project (or similar projects) does not succeed, it could distract research and treatment away from more viable solutions, affectively jeopardizing the opportunity of the individuals who need it. The inequalities of the least advantaged in society are not given much more flexibility with this project beyond the groups it directly affects. As stated before, the project can give people more opportunity with effective treatment, which improves the lesser advantaged in society, but it is not an absolute solution to the difference principle. Many of the facets of fairness and inequality this project creates depend on the society the project integrates to. The project does not seek to change any of those tenants, but instead change the lifestyle of those who benefit and engage with the product.

9. Health and Safety

Providing better health defines the essence of this project, and the health and safety implications of the project guide every facet. The project potentially gives the recipients movement they might never have again, and allowing them to do so acts as this project's top priority. As such, the project aims to help the customers' well-being than harm. And since the project has the customers act physically during their participation, the potential harm to their arm becomes a main concern. Economically, the product sells with these factors as marketing principles. Thus the health and safety concerns affect the design, manufacturing, and economic facets of the system and permeates its purpose for existing.

10. Social and Political

The stakeholders of this project include the patients, therapists, researchers, and engineers involved. The medical community and any political party that backs such treatment potentially benefits from the project. More notoriety may also be given to stroke treatment and potentially more compelling research. The project could also negatively affect these institutions if the product offers questionable performance. Public reception also dictates the longevity, lasting impact, and reputation of those involved. Physical therapists might become more obsolete if this kind of treatment becomes effective and more widespread. If the stroke therapy from this device succeeds, then drug and pharmaceutical companies potentially acts as competition or their medicine becomes outdated. Welfare and insurance companies may be affected by the device as they either endorse or deny such treatment. Limited patents for such products currently exist

[6], and if an increase in the number patents for this brand of therapy occurs it could prove more complicated for different products to arise and for other parties to get involved. Patients with certain conditions benefit most from the functionality of the project, and physical therapy may not be as adequate a treatment for some. Depending on potential insurance costs, customers may even be locked out of this kind of treatment. Thus this project favors those with a more steady income. Some knowledge of computers and running software is also required to run the device, thus those living in more First World countries who more readily have that kind of knowledge may benefit more.

11. Development

This project allowed for the exploration of many facets surrounding the design and development of a functional product, while building upon an unfinished project from an internship. Research into the design process and what it entailed began early into the planning of the project [6]. This also required investigation into therapy techniques and data gathering in order to accurately address and identify customer needs [2-5]. Finding and characterizing components also allowed for a more detailed probing and deciphering of datasheets [7], [8]. The project also allowed for the consideration of additional elements of a product, such as aesthetics, which are not often considered in hardware and circuit based design. Thorough documentation proved valuable as a skill during the design process.