Wheelchair Assist:
Systems and Controls

6.1 Overview

The goal of the Systems and Controls team was to work with Systems Integration to be able to incorporate a microcomputer in order to control the actuators. Electronic wiring, power consumption, and data recording were important tasks in optimizing the wheelchair design. By using a Raspberry Pi (RPi) and planning several connections, a Graphical User Interface (GUI) was coded and used to provide user control for the entire operation of the wheelchair. Multiple General Purpose Input/Output Pins (GPIO) were used to send signals/receive signals and essentially control the wheelchair.

6.2 Logic Flow

The flow of logic was largely based on the previous design. After testing the previous wheelchair, it became clear that the ability to press up or down on the actuator at any time caused issues for wiring, programming, and user functionality. The new program involves a case basis, meaning that once a button is pressed by a user, the program determines what actions need to be performed and performs them in order without interruption from the user. Based on the previously recorded mode the system decided which actuators still need to be extended or retracted. The order of operations can be seen in the following chart, where based on the previous mode and input command, the signals are sent in order moving towards the Red Redundant setting.

		Input_Command				
Previous_Mode		Mobile	Horizontal	Paws	Diagonal	Seat Lift
	Mobile	Redundant	"Engage Brakes"	"Engage Brakes"	"Engage Brakes"	"Engage Brakes"
	Horizontal	HRails_Retract	Redundant	HRails_Extend	HRails_Extend	HRails_Extend
	Stabilized	Paws_Retract	Paws_Retract	Redundant	Paws_Extend	Paws_Extend
	Diagonal	DRails_Retract	DRails_Retract	DRails_Retract	Redundant	DRails_Extend
	Seat Lift	Seat_Fall	Seat_Fall	Seat_Fall	Seat_Fall	Redundant

Figure 6.1: Logic flow of the wheelchair program

6.3 Circuit Diagram

A circuit diagram was established to help distinguish pin assignments as well as simple connections. Ten GPIO pins were output to the 8-relay switch, controlling the seven actuators (3 Sets and one lone actuator) installed on the wheelchair. The 12V battery was connected to the 8-relay switch, allowing power and current to flow when the program logic allowed it to. A fuse was added in line with the 12V battery to protect the Relays and Actuators. The actuator sets are introduced by a wire splitter past the relays, meaning that the left and right actuators of any set

can only be operated simultaneously. A smaller battery pack is stepped down to 5V and used to power the controller (Raspberry Pi 3).

The sensor array makes up the remaining wiring. Two contact sensors set in normally open are connected to GPIO input pins on the RPi. Additionally two pressure sensors are set up similarly with voltage only passing through when the paws put sufficient force on the pressure sensors. The signal from the Pressure sensors needed to be filtered as the preload on the paw fixture would raise the baseline voltage. In the final system, there is almost no preload on the paw sensor; however, potentiometers were still used to filter the signal as needed. The Raspberry Pi detects a high input signal at 1.3V, so these potentiometers are a simple way to tune the pressure sensor to different user weights as well as the multiple iterations of paw fixture design. During early stages of development the 5V pins on the RPi were both occupied (one by relays and one by the touch screen). It was for this reason that the reference voltages of each sensor were sent from a GPIO output high pin (~3.3 V).

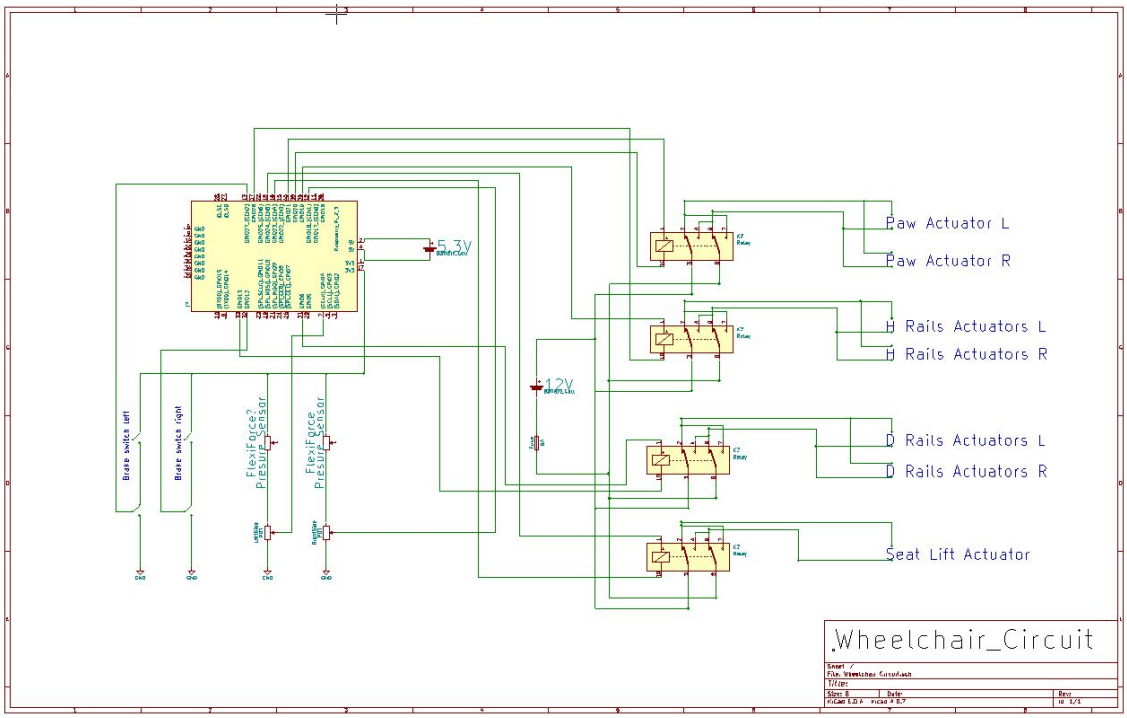

Figure 6.2: Wheelchair Electronics Circuit Diagram

6.4 Wiring Plans and Enclosure Design

The wiring used outside of the electrical enclosure was the TURCK Eurofast Connector. This plug allows for a quick disconnect with multiple channels to be purchased pre-soldered with excess cord length. The five plugs were purchased with product identifier (RKC 4.4T-2-RSC 4.4T). These plugs have four channels each suited for 18 AWG wiring. Before the enclosure was fully designed the arrangement of wiring was proposed as follows. Two Plugs would be located on the left, two located on the right, and one would be located near the center or front. The first of the side plugs would have two channels dedicated to the Diagonal Rail Actuator and two channels dedicated to the Horizontal Rail Actuator. The second side plug would have two

dedicated channels connecting to the Paw actuator, one channel dedicated to the Paw Pressure Sensor, and one channel dedicated to the Brake Sensor. The front plug connected to the Main Seat Lift Actuator with two dedicated channels. Each actuator wire connects to the two inputs on the actuators, and when one is powered with 12-V, it forces the actuator to either retract or extend.

Currently, there are 6 channels running through the telescoping rails, including paw sensor rails and brake sensors. All wires were placed along the tubing to reduce chances of the user catching the wires. The horizontal rails have openings on the bottom channels through which the necessary wires were fed from the circuit enclosure to the Brake and Paw Systems. The position of the Brake system is static so the wires are a fixed length whereas the Paw System depoys twelve inches forward. Adjustability in the wire was achieved by leaving sufficient length to prevent tension in the wire during normal operation.

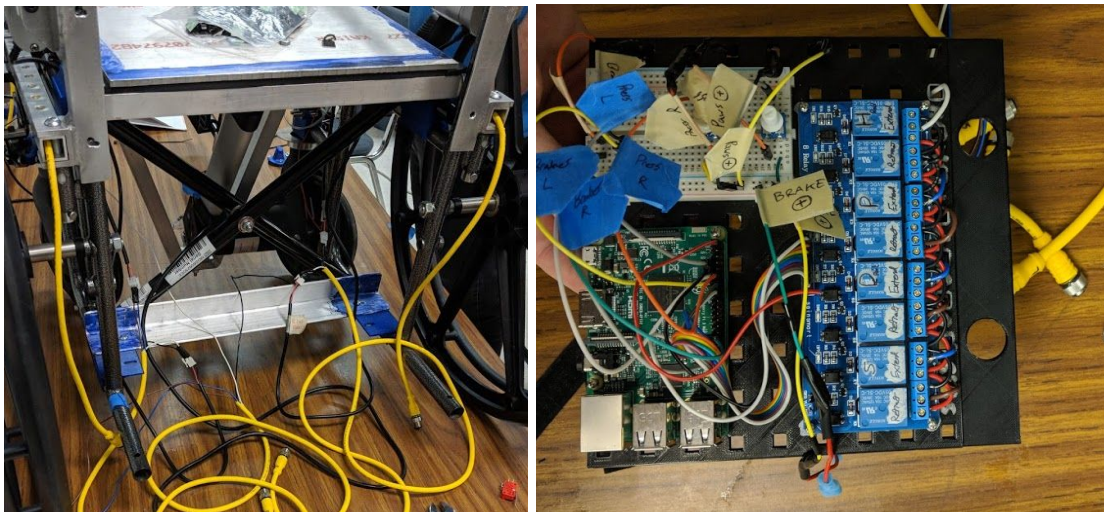

Figure 6.3: (Left) Early wiring of Eurofact Plugs through telescoping rails. (Right) the lattice shelf shown outside of the Electrical Box

After the plugs were decided the dimensions for the plugs, RPi, Relays, Wire Splicers, Power Supplies, and potential Arduino could be factored in to the enclosure design. Based on the dimensions available the ideal enclosure design was determined to involve a separate enclosure for the power supply and for the electronics. The limited space of our design envelope required creation of a very flat design. In order to showcase the electronics while hiding the mess of wires involved, a lattice shelf concept was designed. The lattice shelf supports all of the necessary electronics components and allows for wiring to be tucked underneath the shelf with access through small holes. This design was especially attractive because it gave the wires opportunity to bend more gradually and allowed for easy access to the space when changing arrangements. In the final design the wires that should stay stagnate and do not need to be altered (the Eurofast plug wiring, associated splitters, and step down electronics for the 5V power supply) were tucked underneath the lattice, while the wiring that tended to be changed or needed to be tuned was left atop the lattice shelf. In the final product, as the box is opened the relays, RPi, and sensor array wiring can all be seen.

6.5 User Input Control

The user input was upgraded to a Graphical User Interface (GUI) which is supported through a 7 inch HDMI screen which connects directly to the Raspberry Pi through HDMI. By running the Wheelchair Program using either Thonny or IDLE python IDE, a GUI can provide options for operating the wheelchair. In the previous system five cases were originally written as Mobile, Stationary, Rails Extended, Paws Extended, and Seat Lifted. In the final iteration of this design, these cases were replaced with Mobile, Forward Rails Extended, Paws Extended, Vertical Rails Extended, and Seat Lifted.

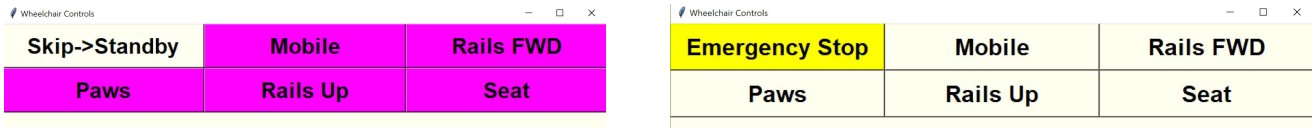

Figure 6.4: (Left) the Initializer Screen and (Right) the Main Screen

In order to run the Wheelchair Program, the RPi must be started by connecting the 7V battery. Then upon loading the Raspbian desktop screen should appear, with a wheelchair program icon. Double tap the icon (if the text label of the icon is tapped it will attempt to rename the program instead of running it) once the Thonny or IDLE IDE appears, tap the drop down menu for Run and select Run Module. It is possible to alter the startup sequence of the RPi in order to have the Wheelchair program load automatically.

When the program is run a minimalistic window appears on screen with buttons for each of these five case modes and one additional button which has two modes. When the screen first appears each of the buttons is purple, this is the initializer screen. It has slightly different logic which assumes that the Previous_Mode value is not accurate. This mode is still capable of achieving each case; however, it takes significantly longer (up to 94 seconds). The sixth button on the initializer screen allows the user to skip to the main screen. This button should only be pressed when the user is confident that the wheelchair is in the same state as the Previous_Mode value. The default mode upon startup is “Mobile”. The design of this logic is meant to save time for users, allow for users to focus on standing/sitting without needed continued input, and limits the current draw in the system by ensuring that only one actuator set is active at any given time.

Once in the Main Screen the button labeled “Skip to Main” is replaced by the Emergency Stop button. Due to limited programming knowledge the Emergency-Stop-Button’s full functionality was not reached. Instead this button simply turns off any actuators and returns the user to the Initializer Screen. This can be useful if a user is unsure if the Wheelchair recognizes its previous mode. Additionally there is a physical emergency stop button located on one of the rails. This button is wired in line with the 12V battery. No actuation is possible while this button is engaged, meaning it should be checked before attempting to actuate. Attempting to actuate while this button engages will often cause the program to have erroneous previous mode values, therefore a program was written to force the user to return to the Initializer screen if this button is pressed.

The HDMI screen utilizes a HDMI cable as well as a USB-A cable the connects to the Raspberry Pi through micro USB-B. This is necessary to enable the touch sensitivity and input for the touch screen.

The GUI was also designed to be compatible with a smartphone or computer through a program called VNC Viewer. VNC viewer is a free app which can be downloaded for Android, Iphone, Windows, Mac, and many distributions of Linux. VNC Server is installed on the RPi. After inserting the correct IP address associated with the RPi and the RPi's password the system can be completely controlled by a remote phone or desktop. VNC requires that both the server (RPi) and the user control (Phone or Computer) are on the same local network. For more in depth testing the system was tested to run off of Hotspot Wifi, meaning that VNC can function if a user phone generates a hotspot and the RPi connects to that wifi. This was found to be extremely useful for testing purposes but ultimately while using VNC a dedicated screen is required to ensure functionality outside of stable wifi range.

The screen is currently minimal, but can be vastly upgraded given sufficient experience in the tkinter module. Further the capabilities of the VNC Viewer system could be emulated through bluetooth. This would require that serial communications are written into the python code as well as a paired app written to be run on a smartphone. While this process would allow for the most ideal system, the programming resources and skill required were not available during this year. In future iterations of this project, it is highly likely that a bluetooth system could be established to allow for seamless control of the chair by a user's phone.

6.6 Perception

Only a few sensory inputs were designed for this system. First, the Brake Sensor was reused from the previous design iteration, although its mounting was changed significantly. The Brake Sensor is a simple contact sensor resulting in an on/off signal. The configuration was designed to be Normally Open. This signal can be received by a digital GPIO Pin and was therefore directly wired to a GPIO Pin on the RPi. Programming on Python for the Brake Sensor involved writing a small sub program which waits for sensors to show that the Right and Left Brakes are both engaged.

The Paw Pressure Sensor was a new addition to this design. The pressure sensor used was a FlexiForce ESS301 Sensor. The pressure sensor sends an analog signal which an RPi is unable to read properly. Two potential solutions were explored, one a software solution and one a hardware solution. The software solution involved slaving an Arduino to the RPi by connecting the USB-A (RPi) ports to the USB-B (Arduino Uno) or Micro-USB (Arduino Uno). This solution would require some minor Arduino programming and separate additions to the Python programming to allow for communication between the two. When Tuning the sensor the process would require trial and error sessions where the Arduino code would need to be altered and reuploaded in between trials. This solution would also require the additional space for an Arduino to be added to the electrical enclosure as well as additional power to be diverted to the Arduino. The hardware solution was far simpler. This involved filtering the signal with a potentiometer which required tuning once installed. The potentiometer would be physically

tuned to change the signal threshold of the pressure sensor and once set would retain a static threshold.

6.7 Processor, Controller

In the previous design iteration an Arduino Uno was used to control the entire system. In this design a more advanced RPi was chosen to allow for more functionality. This required that the programming language Python were used instead of the Arduino language. As a micro-computer, many programs are able to be ran and configured on a graphical user interface (GUI). In an effort to create a completely touch screen controller, a GUI was created and coded to work with the relays

Figure 6.4: Raspberry Pi 3 Model B

The Raspberry Pi is running off of a Quad Core 1.2GHz 64bit CPU, which is plenty enough processing power to run the program that was created to control the wheelchair. Along with that, a small micro SD card with an operating system installer (NOOBS) is connected to the micro-computer. This is what was used to store the program and is essential to getting the programs to work. The extended 40-pin GPIO was used to send and receive digital readings of up to 3.3-V, making it a little bit more limited than the 5-V Arduino pins.

Ten pins were allocated to operating the eight-relay system. Two pins were needed to provide power and ground, with the other eight pins providing a signal to switch the relays on and off. A full size HDMI cable was utilized to for a regular monitor, as well as two USB-A connections for a keyboard and mouse. The final product would still utilize the HDMI cable, but it would output to a capacitive touch screen utilizing one USB-A cable to provide touch control.

6.8 Battery and Power Analysis

In designing the power arrangement both a 5V and 12V circuit needed to be built. The 12 volt system powered by a single large battery would run the actuators by supplying power to the high side of the relays. The 5V circuit is powered by a six NiMH batteries with simple circuitry to reduce the voltage from 7.2 to 5 volts. The 5-V circuit was required to power the RPi and the 7 inch resistive touch screen.

When deciding which batteries to use to power the screen and Raspberry Pi, the biggest thing that had to be taken into account was the necessary draw of current and a steady 5-V output. A 9-Volt alkaline battery would be the most simple in terms of replacing batteries and space, but the capacity in amp-hours is low. Compared to a single AA battery, the current is about six times less when considering Ah ratings (around 500 mAh). A single AA alkaline battery can supply anywhere from 2000 mAh to 3000 mAh. A pack of four AA batteries were initially used to provide power to the screen and raspberry pi. It worked until the screen was replaced, because the 7 inch screen drew more current than expected. Finally, the battery settled upon was a Venom 3000mAh battery. It provided a steady 7.2-V that was stepped down with the buck converter. The overall current output was higher, and is ideal because of the NiMH battery type, allowing rechargeability.

SEAT LIFT ACTUATION:

When considering the seat lift actuator and the load it will be experiencing, it was decided that the 400 lb actuator was appropriate for the task at hand. The seat actuator was tested with two weights, keeping in mind that the maximum user for this wheelchair is not to exceed 250 lbs.

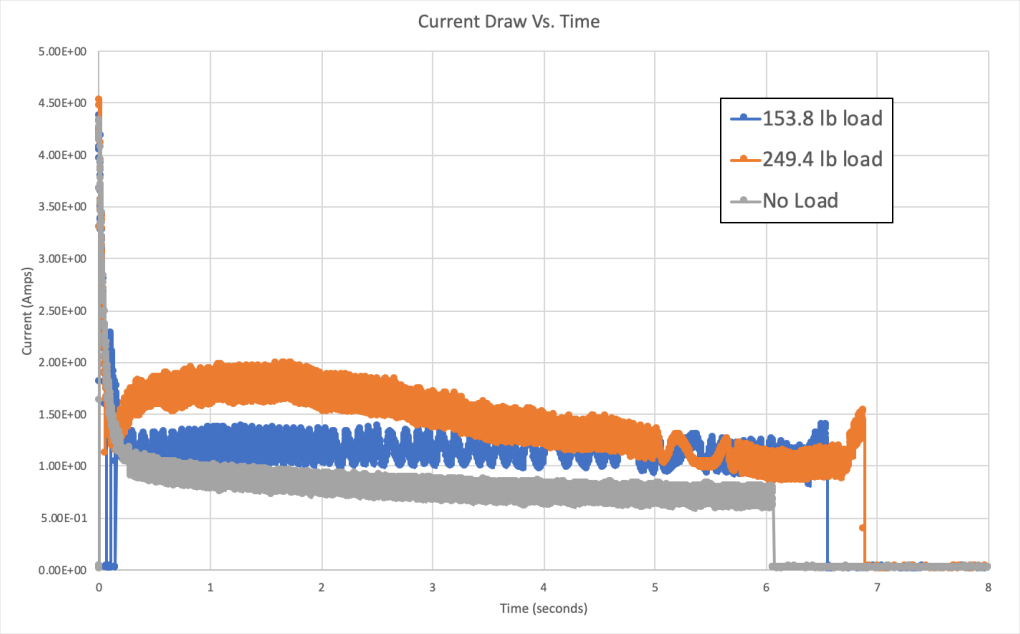

Figure 6.5: Seat Lift Actuator Overall Current Draw

When looking at the graph, the battery was disconnected from the actuator when it was fully actuated. This is shown at the end of the graph when the lines show a steep drop. Each actuator took around 7 seconds to fully extend, with the heavier loaded tests taking a bit longer to fully extend. Because of the nature of current draw, the graph below is needed to show the quick current spike. It is also observable that the 249.4 lb load had a higher average current draw compared to the 153.8 lb load and the loadless tests. The current values for when the seat is retracting are similar to the extension values.

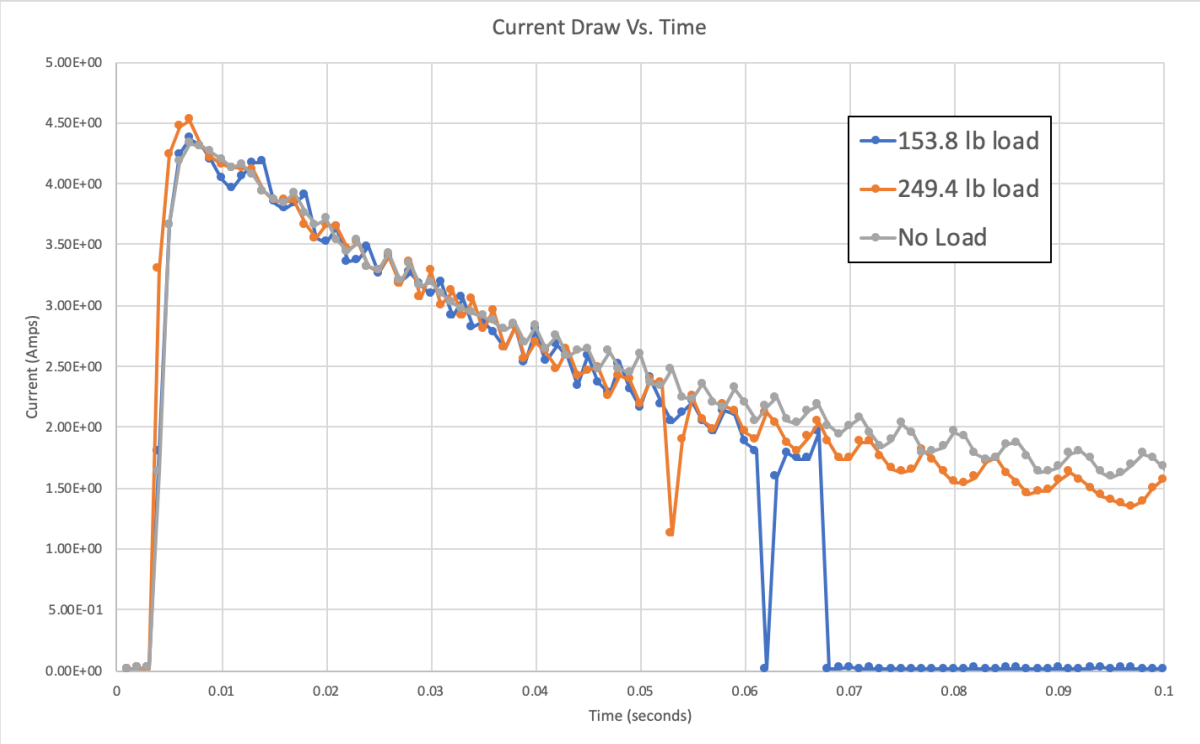

Figure 6.5: Current Spike from Initial Actuation

Seat Lift Actuator	
Load (lbs)	Max Current Draw (A)
0	4.33
153.8	4.37
249.4	4.52

Table 6.1: Current Draw According to Weight Applied

When analyzing the initial draw when the seat is extending, the difference is very minimal, ranging from 4.33-4.52 amps. Although this is important, it was more essential to check the draw over time in order to control current. Too much current could burn out an actuator or create too much heat in the wires.

HORIZONTAL RAILS ACTUATION:

The horizontal rails are not supposed to support any loads when extending. The user should not be resting on them in order to make sure that they are operating correctly and not experiencing any unnecessary forces. Two actuators are running in parallel, and the results below represent both actuators drawing current.

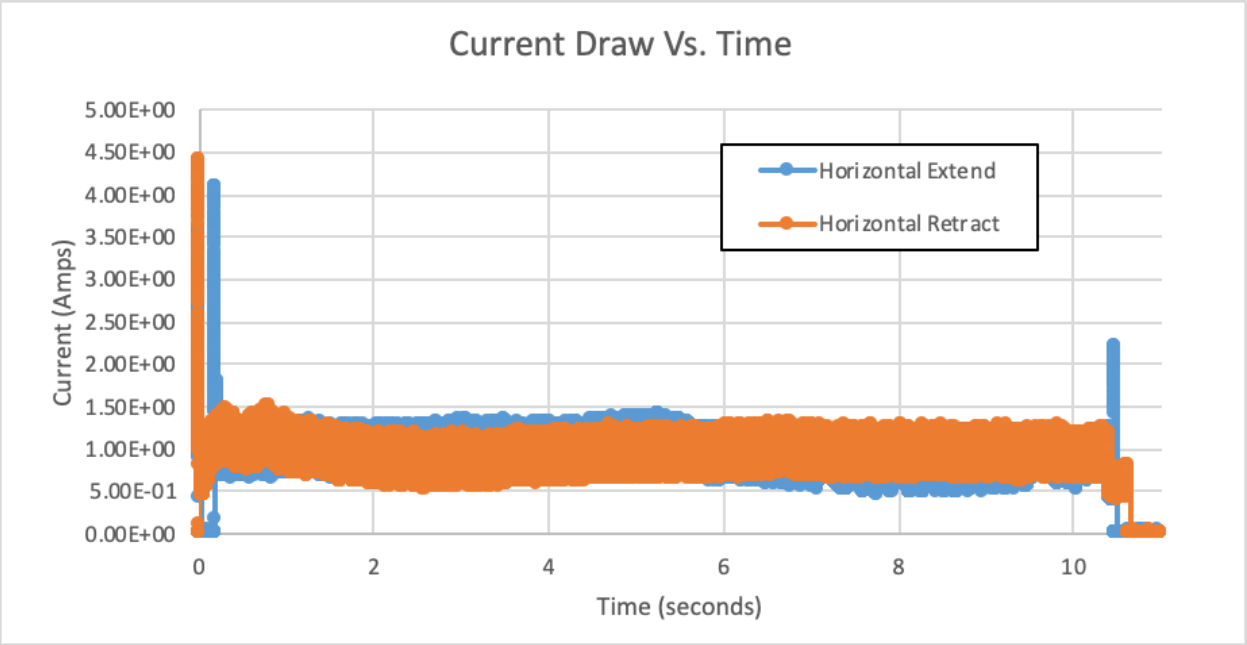

Figure 6.6: Horizontal Actuators Overall Current Draw

When looking at the graph, the battery was disconnected from the actuator when it was fully actuated. This is shown at the end of the graph when the lines show a steep drop. Each actuator took around 11 seconds to fully extend, with the retraction tests taking a bit longer to fully retract. Because of the nature of current draw, the graph below is needed to show the quick current spike. It is also observable that the retraction actuation had a higher initial current draw compared to the extension.

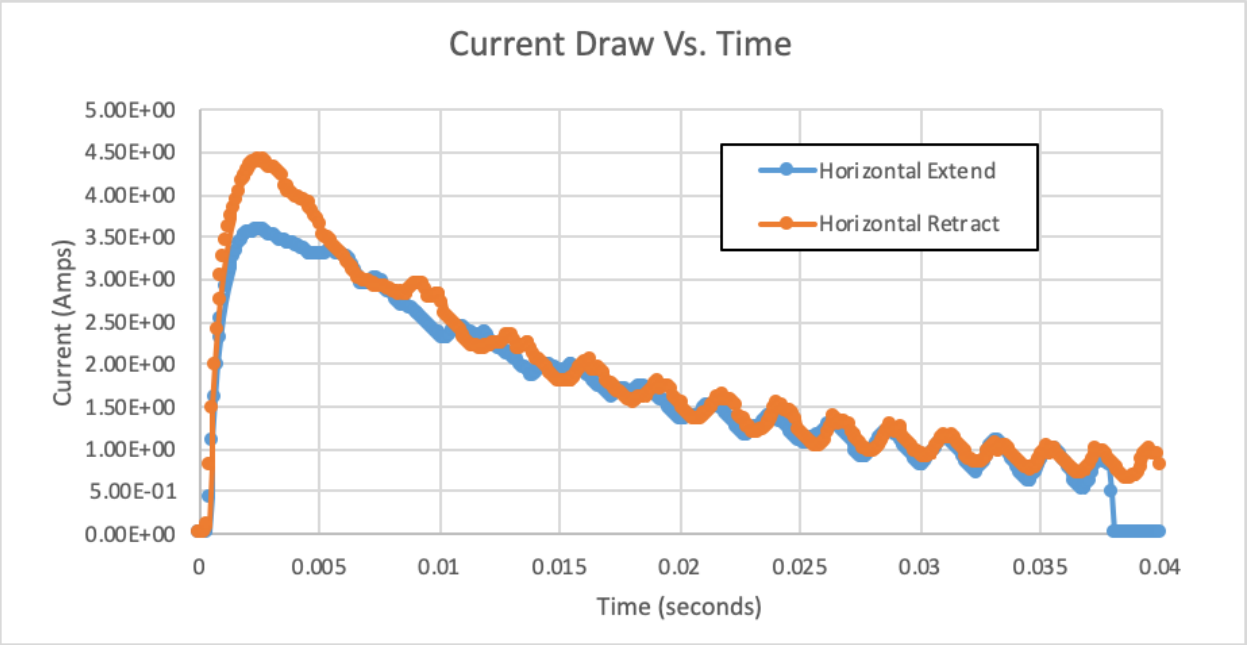

Figure 6.7: Current spike from Initial Actuation

Horizontal Rail Actuators		
Action	**Max Current Draw (A)**	**Time (seconds)**
Extension	4.082	9.5
Retraction	4.392	11

Table 6.2: Current draw and actuation time

When considering the extension time of the horizontal rails, the full extension of the actuators would not be compatible with the telescoping rails. If the left rail reached full extension, it would have trouble telescoping upon return because of the play between the horizontal rails, their brackets, and the slide. At full extension, the sliding assembly droops ever so slightly, due to the U-channel used on the base rail and the thin 3D-printed retaining sleeve. This droop is just enough to push the assembly out-of-square and cause jamming upon retraction. Solving this problem would involve larger, sturdier structures to keep the rails and slides fully aligned at all points in RSS deployment. As a note, the actuators would take a little bit longer to retract, something that needs to be considered since the code relies on manual time inputs.

PAWS ACTUATION:

Two paws were run in parallel and were used to stabilize the wheelchair to prevent it from tipping forward and making the user fall forward/lose balance. As shown in the figure below, the current draw starts high initially and then quickly settles

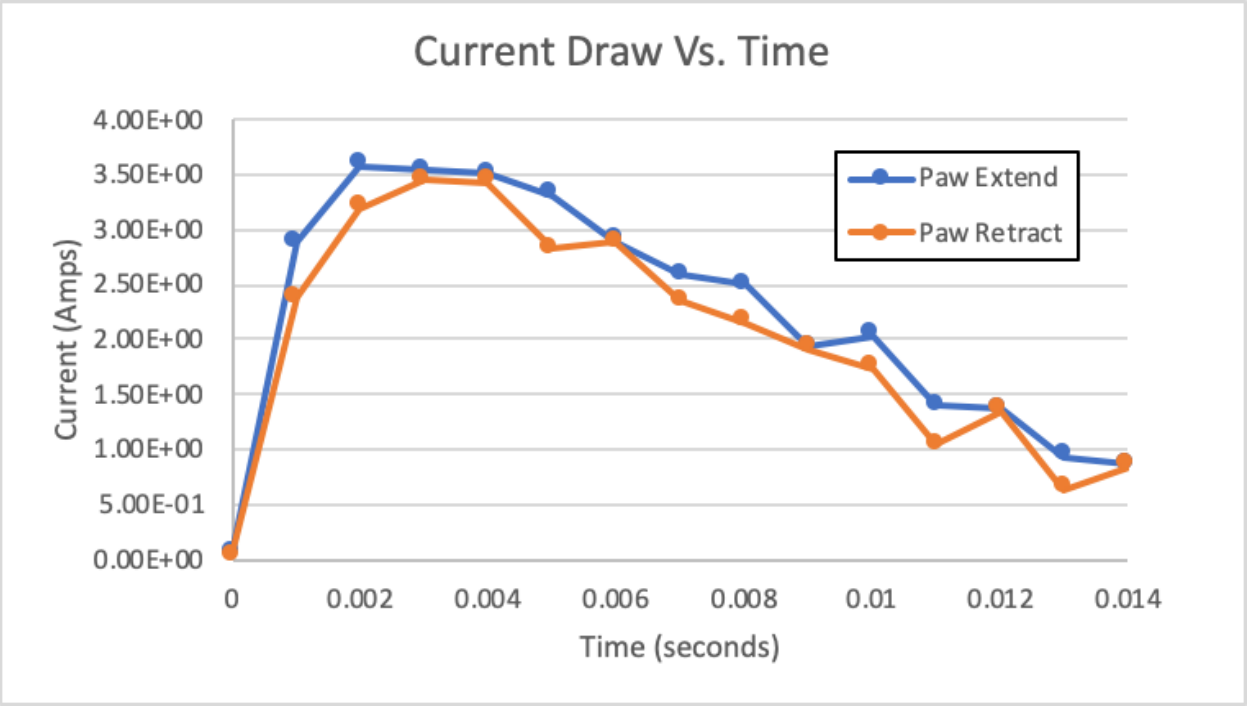

Figure 6.8: Current spike from Initial Actuation

Paw Lift Actuators		
Action	**Max Current Draw (A)**	**Time (seconds)**
Extension	3.585	4
Retraction	3.42	5

Table 6.3: Current draw and actuation time

Actuation time depended on pressure sensors that communicated to the micro-computer that the paws are applying an appropriate amount of pressure to the ground. Because of this, recording times varied significantly during testing.

VERTICAL RAILS ACTUATION:

The vertical rails are also not supposed to support any loads when extending. The user will not be resting on them in order to make sure that they are operating correctly and not experiencing any unnecessary forces. Two actuators are running in parallel, and the results below represent both actuators drawing current.

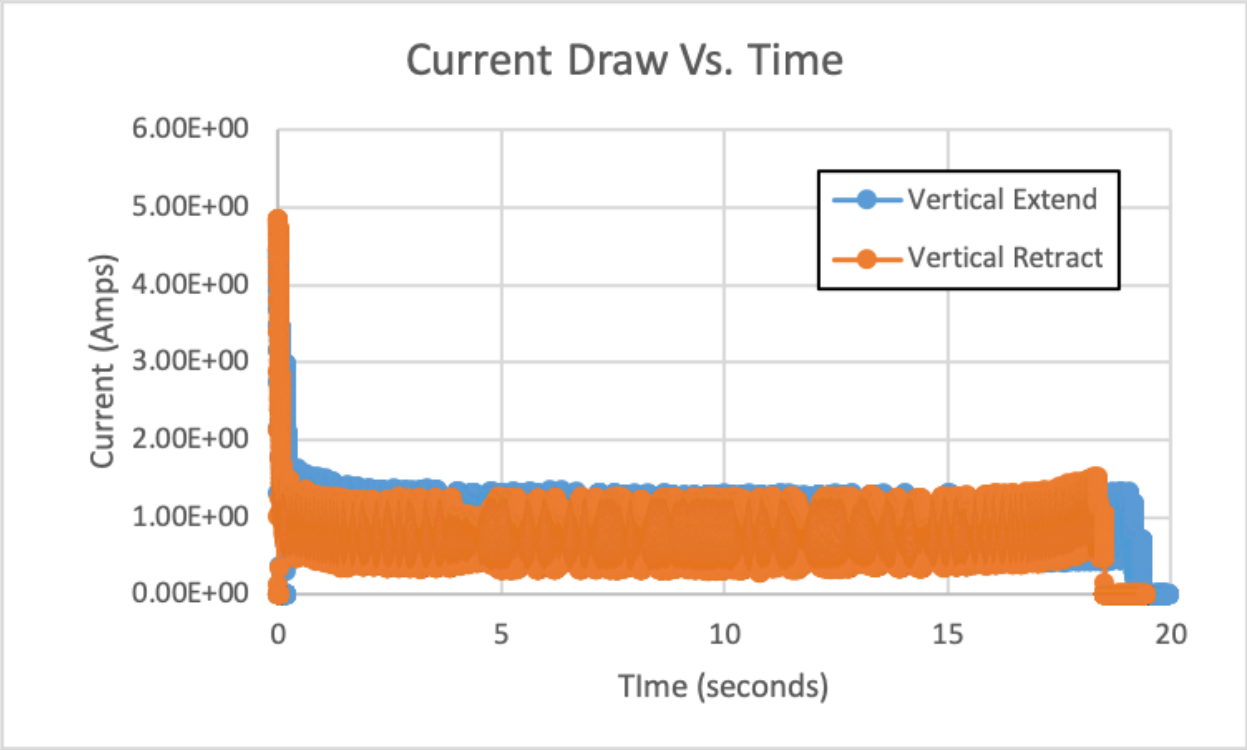

Figure 6.9: Vertical actuators overall current draw

When looking at the graph, the battery was disconnected from the actuator when it was fully actuated. This is shown at the end of the graph when the lines show a steep drop. Each actuator took around 20 seconds to fully extend, with the extension tests taking a bit longer. Because of the nature of current draw, the graph below is needed to show the quick current spike. It is also observable that the retraction actuation had a higher initial current draw compared to the extension.

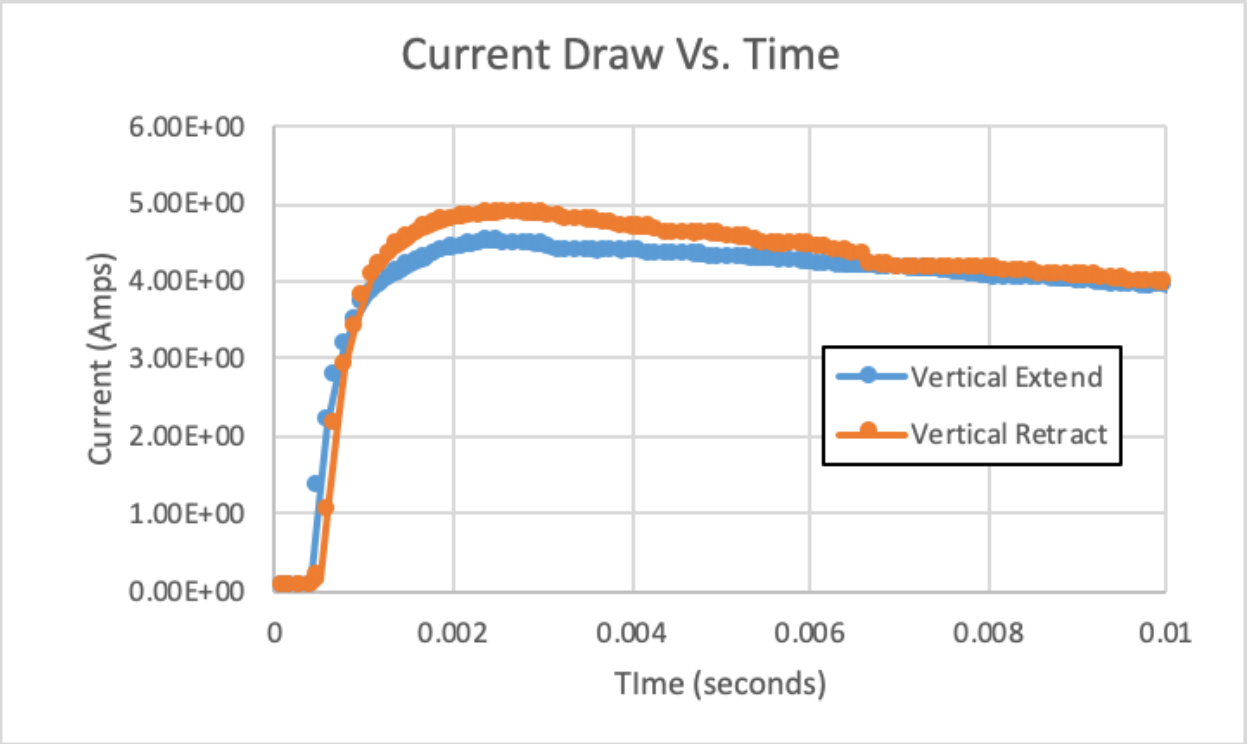

Figure 6.10: Current spike from initial actuation

Vertical Rail Actuators		
Action	**Max Current Draw (A)**	**Time (seconds)**
Extension	4.466	19.7
Retraction	4.850	18.8

Table 6.4: Current draw and actuation time

Again, when considering the extension time of the vertical rails, the full extension of the actuators would not be compatible when considering usage for test subjects.The full extension of the actuators were too high and would prove unusable for stability purposes. The actuators would take a little bit longer to extend, something that needs to be considered since the code relies on manual time inputs.

TOTAL CURRENT DRAW:

Keeping in mind the utilization of a 12-volt battery that has a 10 Ah life, it was decided that the battery would be limited to 50% of its max charge. As such, every actuator was tested and the current draw was measured. For each actuator, a high current draw is necessary for it to start actuating. After a quick draw, an current draw had dropped to an average level of around 2 amps or below. The average current draw was multiplied by the time it took for a full cycle of usage, with each actuator extending and retracting.

Full Cycle Actuation and Current Draw					
	# of Actuators	Full Cycle Time (seconds)	Initial Current Draw (A)	Average Current Draw (A)	Amp-Hours
Seat	1	14	4.52	2	0.0108
Horizontal	2	20.5	4.392	1.5	0.0237
Vertical	2	38.5	4.85	1.6	0.0474
Paws	2	9	3.585	2	0.0138
Total Amps per Complete Cycle (Ah)					0.0957
Total Time per Complete Cycle (seconds)					82

Table 6.5: Total Current Draw

Considering the restrictions placed upon the design, the battery was limited to 60% of its charge cycle in order to best preserve the charge capacity. This resulted with 62 cycles, accounting for the total current draw for a full cycle. This would mean that the battery would be charged once it would reach about 40% left of its charge.