GNG 2101

Design Project User and Product Manual

Team: Z4

Enhanced Walker Braking System

Submitted by:

Walker Braking, Group Z4

[Michael Puma, 300069597]

[Caleb Cronin, 300138147]

[Lauren Healy, 300131780]

Date: July 25th, 2021

University of Ottawa

Table of Contents

List of Figures

List of Tables

List of Acronyms

Table 1. Acronyms

Acronym	Definition
GND	Ground
+	Positive
-	Negative
PCB	Printed Circuit Board
LED	Light Emitting Diode
IDE	Integrated Development Environment

1 Introduction

This User and Product Manual (UPM) provides the information necessary for users to effectively use the one-handed walker braking system and for prototype documentation. This report details the functions and features of the one-handed walker braking system as designed by the Witty Walkers. Most commercial walkers currently available today require the use of two hands and do not accommodate individuals with physical disabilities. In consideration of the client’s partial paralysis, there was need for a walker braking system that will engage both sides of the brake unit at the same time with the use of only one hand. The contents of this report summarize the solution created by team Z4: a new braking system that can be implemented onto any ordinary walker and used with easily with only one hand. This report also provides documentation of the full development process of the one-handed walker braking system to fulfil the previously identified client needs. Firstly, an overview of the product function and a walkthrough of the system is provided to clarify the need for the product for individuals with physical disabilities. Also included in this report is a detailed description of each user function and feature of the walker braking system as well as necessary troubleshooting for future users. A detailed documentation of the developed prototype is provided, and further supported with visual renderings and detailed images to demonstrate its function. In summary, the objective of this document is to provide the reader with full documentation of the development of the walker braking system, its function, and how it can be used to effectively activating the brakes of a walker with one hand. Lastly, this report also includes improvements that can be made to the product based on final client feedback, as well as recommendations for future work on the one-handed walker braking system project.

2 Overview

There are many diverse types of diseases and disorders that can leave one with limited mobility in one or both of their hands, some of these include: carpal tunnel syndrome, arthritis, trigger finger, dupuytren disease, and loss of movement from other injuries. As previously mentioned, most commercial walkers currently available today employ the use of two hands and do not accommodate individuals with physical disabilities. Because of this, there is a need for an inexpensive, reliable, and effortless way of engaging a one-handed braking system on a walker; while also maintaining a lightweight, transportable, and user-friendly design that still maintains all the walker's previous functionality. Based on a benchmarking analysis, each of the products currently available on the market fell short of at least one of the previously identified client needs. Compared to other products that require the users grip strength to apply the brakes, our product allows the user to brake simply with the pull of the brake lever, with minimal grip strength required while additionally costing a fraction of the price. The only user requirement that the product requires is the ability to walk and the mobility of at least one hand. It is also assumed that the user will have the required knowledge and physical capacity to install the product to their existing walker.

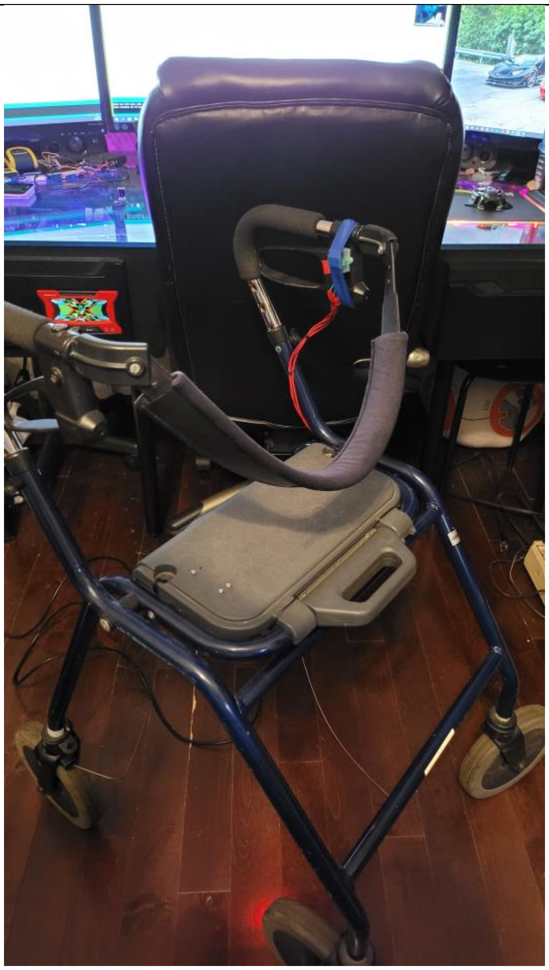


Figure 1.1: Final Prototype

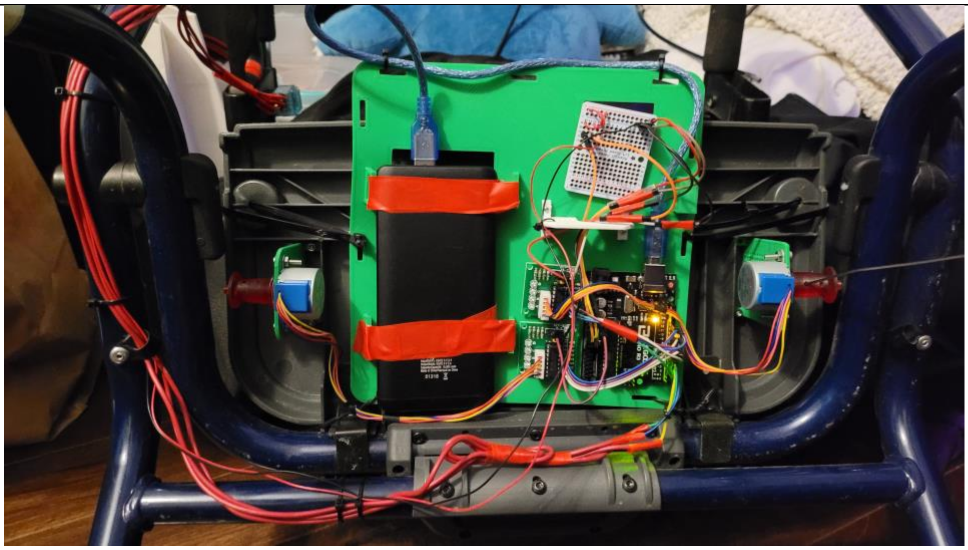


Figure 1.2: Final Prototype Mechanism & Wiring

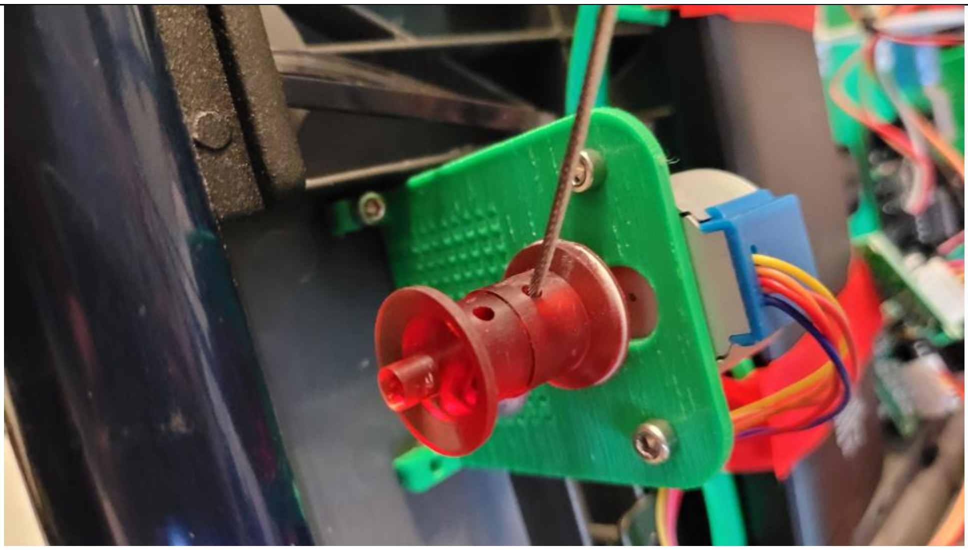


Figure 1.3: Final Prototype Motor & Braking Cable

This system allows users to effectively activated the brakes of a walker with one hand, while using minimal grip strength. While the one-handed braking is the most notable function of the product, it is also equipped with several other features that optimize the user's experience. This includes an E-Brake button that is featured in the design, which allows users to lock the brakes to sit or lean on the walker. Additionally, a mode-selection button is included in the design that allows the user to customize their experience by selecting how sensitive the braking is. The modes are indicated by a RGB LED which changes color from green being the least sensitive, to red being the most sensitive.

The key aspects of the architecture of the system include but are not limited to an Arduino Uno, a rotary encoder, a stepper motor, a USB battery, a mode selection button and "E-brake" button. All mechanical components are housed in 3D-printed enclosures that can be universally mounted to any walker using Velcro and/or zip ties.

2.1 Cautions & Warnings

It is important to note that the final prototype is not waterproof as initially planned. Because of this, further developments need to be made before the product can be used in all weather conditions. It is important that until modifications are made, this product does not come into contact with water. One additional caution is that there is currently no emergency back-up system in place in the event that the battery does die. Because of this, it is the recommendation of Witty Walkers that all users carry an extra USB battery so that the system can function at all times to ensure the safety of the user.

3 Getting started

Firstly, after the customer purchases the braking system, they will have to decide if they want to send their walker in to have the braking system installed in case they are physically limited or, if the customer wishes to install the braking system on their own. Self-installation can be easily done since the system is fitted with tiedown points along with zip ties and Velcro straps that can be easily adjusted for preference. Removing the brake cable and installing the rotary encoder can prove to be a bit difficult but this process is better detailed at 6.1.3. Once the system is installed onto the walker the next step will be to make sure the systems battery is fully charged, this is done by removing it from the case and plugging it into a USB charger. Once the battery is charged the user will have to select their braking intensity level where users with low grip strength will select a higher intensity braking level while others would select a lower intensity. Now the walker is ready to begin operation with the new braking system and a reminder to charge the battery once it is done being used.

3.1 Set-up Considerations

The devices used in our braking system include an Arduino Uno, a portable USB charger (battery), a rotary encoder, a stepper motor, a spool, protoboard, zip ties, Velcro straps, buttons, LEDs, cases for LEDs, and 3D printed enclosers that fit all the devices and keep them weatherproof. The first 3D model is designed to house the Arduino Uno, the stepper motors along with the spool, the protoboard, and the potable USB charger/battery along with many wires. It will be mounted under the seat or on the side of the walker. The other 3D encloser is for the rotary encoder that will mount on the pivoting axis of the brake. The system is configured by the brake cable being detached from the brakes handle and attached around the spool. The rotary encoder which is mounted to the pivoting axis of the brake handle relays data to the Arduino which translate the data to the stepper motor which tells it to engage the brakes by turning the spool tightening the brakes. This is all powered by the USB charger/battery.

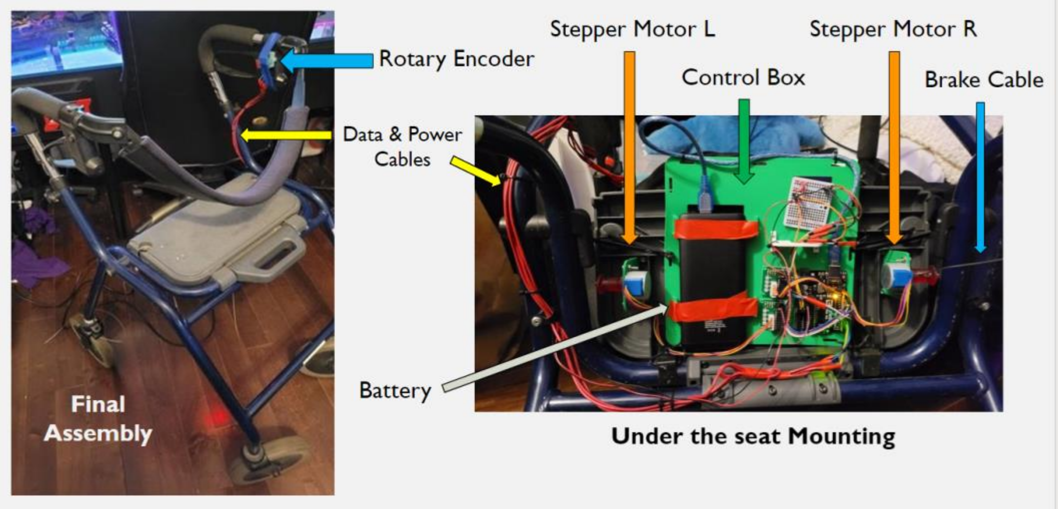


Figure 3.1 Final Assembly

3.2 User Access Considerations

The only user restriction that the device has is the ability to function at least one hand and can also the ability to walk. Other than this there are no other requirements or restrictions on the users for accessing the system.

3.3 Accessing the System

The proper procedure to accessing/turning on the system is to make sure that the Arduino Uno is turned on and properly connected to the USB battery. If the user requires to open the Arduino code file, one must detach the Arduino Uno from the 3D enclosure using a screwdriver and plugging it into a computer that is running the Ardinuo program. This is not recommended to be done unless deemed super necessary

3.4 System Organization & Navigation

To use the walker, the user must operate it in the exact same way the original walker was intended. This means that if the user would like to brake, they simply pull up on the walker's brake handle which will electronically apply the brakes for them. The emergency brake is a button that when pressed locks the brakes in the on position. This button is then pressed again to deactivate. The final prototype excludes this feature, but it is an optional add-on; the user can choose to add on a mode changing button. By pressing this button, the braking sensitivity is increased or decreased, indicated by a color changing LED. The user can fine tune the sensitivity of the brakes to their liking by scrolling through different sensitivities. There are five different modes that all change how far the brake handle needs to be pulled before the brakes are applied. The goal of our system was to mimic the original walkers design, and we believe that this has been achieved by reducing the amount of force required on the brake handle, but still working in the exact same way the walker was originally designed to.

3.5 Exiting the System

To properly turn off the system the user must detach the battery from the system and begin recharging. To conserve battery the USB power cable should be removed when the walker is not in use, this will prevent the walkers control systems from drawing power when the walker is inactive.

4 Using the System

The following sub-sections provide detailed, step-by-step instructions on how to use the various functions or features of the One-Handed Braking System.

4.1 Braking System

The main system is the braking system itself which includes the use of the Arduino Uno, rotary encoder, stepper motor and spool, and the brake cable itself. Its functions by taking data from the rotary encoder, basically how many degrees in a certain direction it is rotated. This data is then transferred and translated from the Arduino Uno to the stepper motor which rotates the spoil in the desired direction which is where the brake cable has been reattached thus tightening the brakes. The only user input that is required is engaging the brakes rotating the rotary encoder.

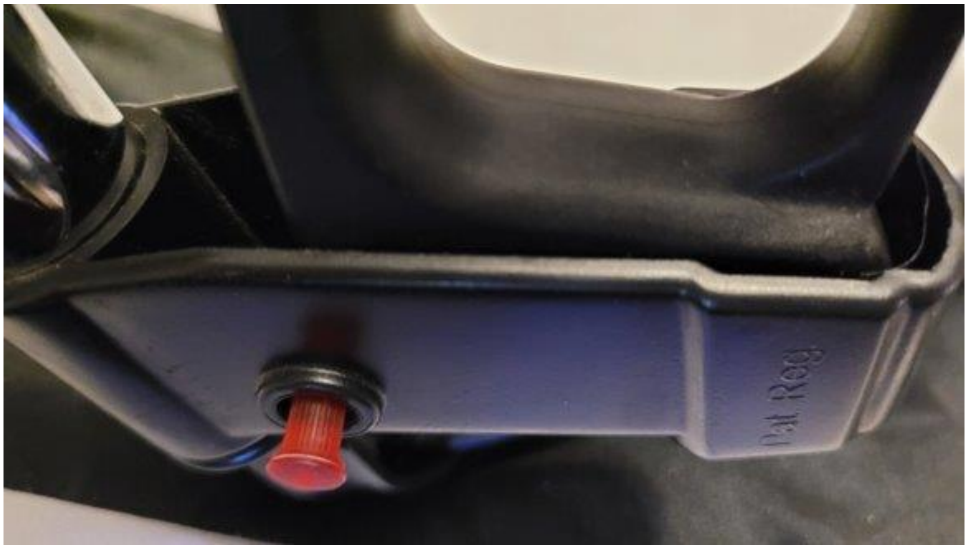


Figure 4.1.1 Rotary Encoder Friction Lock Pin

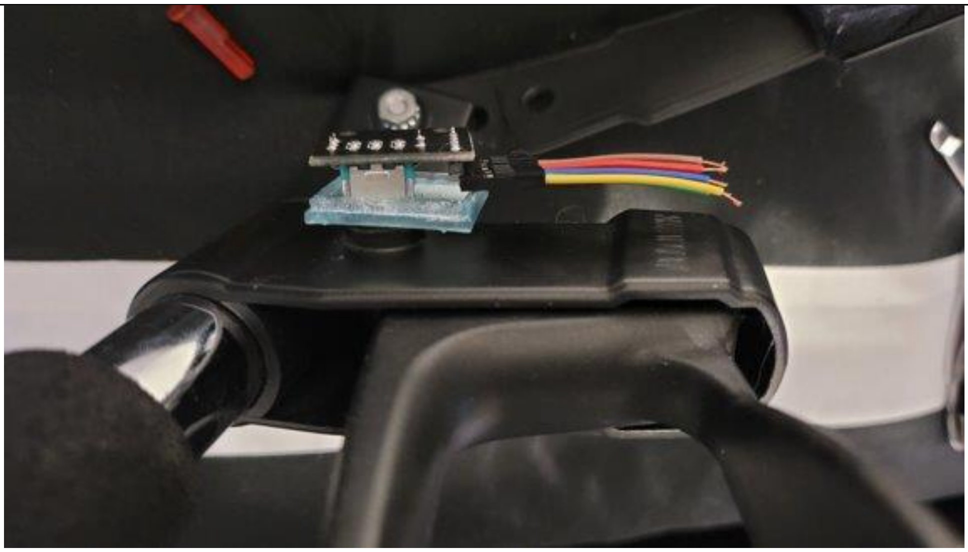


Figure 4.1.2 Rotary Encoder Mounted on Side Handle

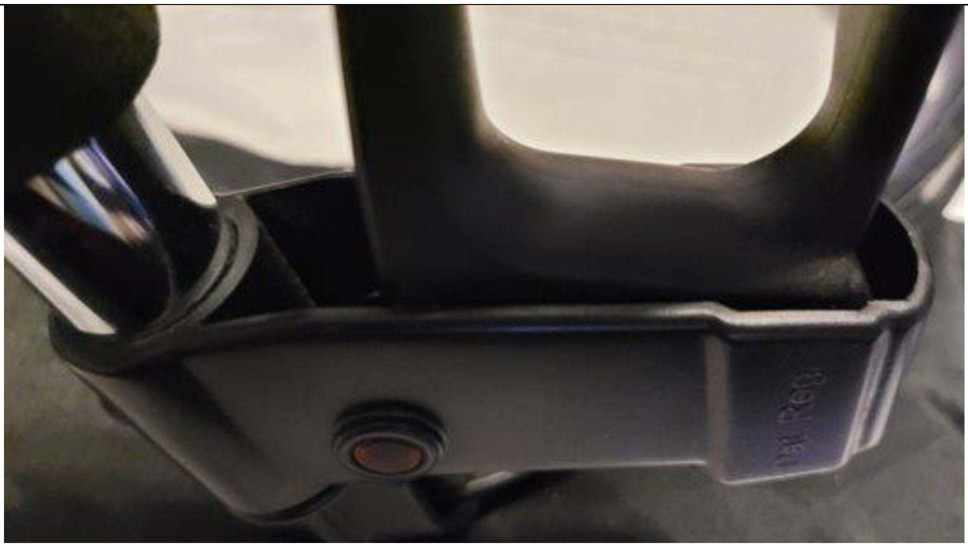


Figure 4.1.3 Pin Allows Handle to Freely Rotate

4.2 Emergency Brake/Parking Brake

The emergency brake is operated by a button that locks the brakes and lets the Arduino tell the stepper motor to fully apply the brakes and to not release them until the user presses the button again. During this time when the Arduino Uno is waiting for the users input it will go into a low battery mode where it will wait for the user to turn off the emergency brake or parking brake.

4.3 Mode Selection

Another feature the braking system offers is the mode selection button that is wired to a 3 colour LED indicator and the Arduino Uno itself. The button changes which braking intensity the user is on which is coloured coded by red being the highest intensity perfect for users with minimal grip strength, yellow for middle intensity, and green for the lowest intensity level used for when the user does not lack grip strength. The intensity levels are programmed into the Arduino Uno already so the only input the user must do is pressing the button to decide their preferred intensity.

5 Troubleshooting & Support

This section will outline all potential errors and their accompanying solutions.

5.1 Error Messages or Behaviors

The braking lever is not entirely accurate. This is due to the cheap rotary encoders used that sometimes register faulty information. If this happens the user must release the brake handle and re-apply it for the brakes to be applied.

Another known issue includes the delay between the brakes lever being pulled and the brakes being applied. This issue can easily be resolved by replacing the motors with a more powerful version or getting more precise rotary encoders that are able to process the users input quicker, applying the brakes faster.

The brakes not being applied extraordinarily strong. This error is due to the weak motors that were used to meet budgetary constraints. If the motors are upgraded to larger more powerful stepper motors, more force will be applied when braking, which would provide a more stable braking experience.

If the brakes are not being applied but the motor is turning, this is due to human error. The user should release the brake handle, allowing the motors to go to the resting position, then tension the brake cable such that the brakes are only slightly not applied. The user can then apply the brake lever and see if the brakes are properly applied, if so, the braking system is calibrated, if not, repeat the above steps of re-tensioning the brake cable.

Nothing happens when the Arduino is powered on:

The Arduino might be faulty, connect it directly to your computer and see if it powers on. If no LEDs are flashing on the Arduino, then the Arduino has been damaged, and needs to be replaced.

Arduino is on, but motors not turning:

This is due to a wiring issue. Check that the wires from the GND and 5V pins are going to the – and + terminals on the proto-board. If they are check that the wiring is correct for all other parts of the system. Confirm this by repeating all steps outlined in the wiring section of the installation guide. If the system still refuses to power on, see section "5.2 Special Considerations". If the solution is not found under the outlined section, please contact customer support to run further tests, and for part replacements if needed.

5.2 Special Considerations

Catastrophic Failure:

If in the unlikely event that a part brakes, or a cable comes loose, the cable should be reattached, or the part should be replaced. These circumstances would lead to catastrophic failure of the braking system and could prevent the walker from braking altogether.

Motor is not turning:

Ensure all cables are correctly connected. The motor controllers LEDs should light up. If the lights are not illuminated, check the wiring. If the lights are on but the motor is still not spinning, then remove the brake cable relieving all tension on the motor, test again. If the motor continues to not spin, the gears are broken, and the motor must be replaced. Contact customer support for replacement parts.

Rotary Encoder not registering hand movements:

Check that the rotary encoder is correctly wired to the Arduino and protoboard. If the rotary encoder is still not registered, and the motors LEDs are on, then the rotary encoder may be faulty. Contact customer support and replace the rotary encoder.

5.3 Maintenance

Maintenance includes routinely checking the cables, and motors for any signs of wear and tear. The 3D printed mounts holding the motors experience a very large amount of force and run the risk of snapping due to the brittle nature of 3D printing. If any cracks form these parts should be replaced as soon as possible.

The portable power bank should be replaced every year to ensure that the battery retains its longevity.

Routinely check that there is not dust or dirt buildup on the electronics as this can reduce the parts lifetime or lead to part failure.

5.4 Support

If any part failure or errors occur that are not already outlined in this document, any of us can be contacted to receive support via email. Please outline in the email your exact issue, providing photographs to help us decipher the problem and determine the best course of action. If the issue is a hardware issue, the most likely solution is to replace the part. If there is a software issue, please reset the Arduino by pressing the red button on the bottom corner. If this does not solve the problem, re-install the software by downloading the Arduino IDE, plugging the USB into a computer, and uploading the provided code. If this does not work, contact customer support.

Table 5.4.1: Customer Support Contact

Name	**Title**	**Contact**
Lauren Healy	Hardware Support & Customer Care	Lheal085@uottawa.ca
Michael Puma	Hardware & Software Support	Mpuma102@uottawa.ca
Caleb Cronin	Electrical & Wiring Support	ccron052@uottawa.ca

6 Product Documentation

Hardware:

The hardware designed for this product include five different 3D printable enclosures. These include the control box, the two motor mounts, and the enclosures for the rotary encoder, LED Mode, the brake Button.

The Motor mounts are simply 3D printed risers for the motors. The mount consists of two horizontally mounted holes to allow the mounting of the stepper motor, and four mounting holes on the bottom to be secured to the control box or directly to the frame of the walker. It should be noted that the mounts are the same on both sides and can be mounted in any orientation.

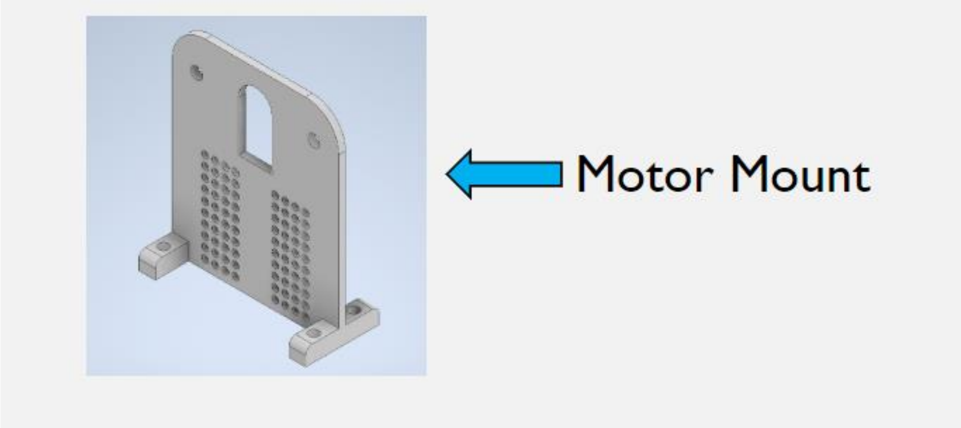


Figure 6.1: Motor Mount 3D Design

First off is the enclosures for the brake and LED. These cases are designed to allow the circuit boards to be placed inside, with a clear design to allow the LED to flash through. There is also an opening on the bottom to allow the wires to pass through. The cases snap together or can be further secured using super glue. The only difference between the two case designs is an opening for the buttons. Both enclosures were printed using clear blue 3D printed resin. The Button enclosures are the same for both the E-Brake button, and the mode selection button.

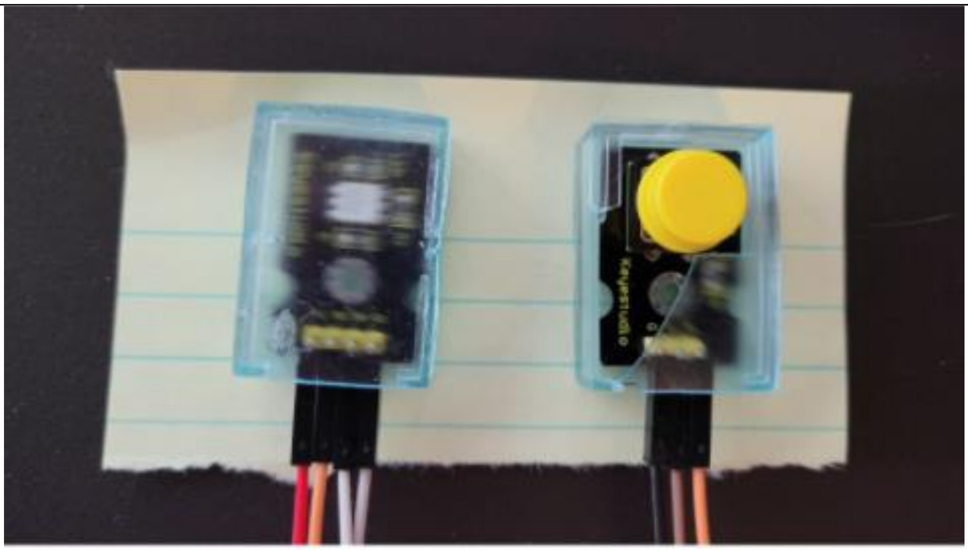


Figure 6.2: Light and Button Enclosures

The control box was designed with modularity in mind. The battery is sandwiched by four ties down points, allowing for the widest range of sizes and shapes of batteries secured using either Velcro, zip-ties, or tape. The enclosure also features standoffs for the Arduino. For ease of use, the standoffs are Snap-On, meaning the Arduino can be push fit to the control box. There are also openings for the protoboard, this allows for easy wire routing of any modifications needed. It is intended for the protoboard to be mounted on the inside, but this opening allows for some wires, potentially some add-ons to be easily added in later. The original design of the control box was based around the use of one motor, this motor is mounted in the center, using the 3D printed motor mounts, and the provided screws. There are also two feed areas on the left side, that allows the brake cables to be threaded through, and screwed down. Surrounding the control box are a series of holes, allowing for the box to be mounted to any point on the walker. These holes are intended for the use of tape, Velcro, or zip-ties, to secure the control box to the bottom of the seat.

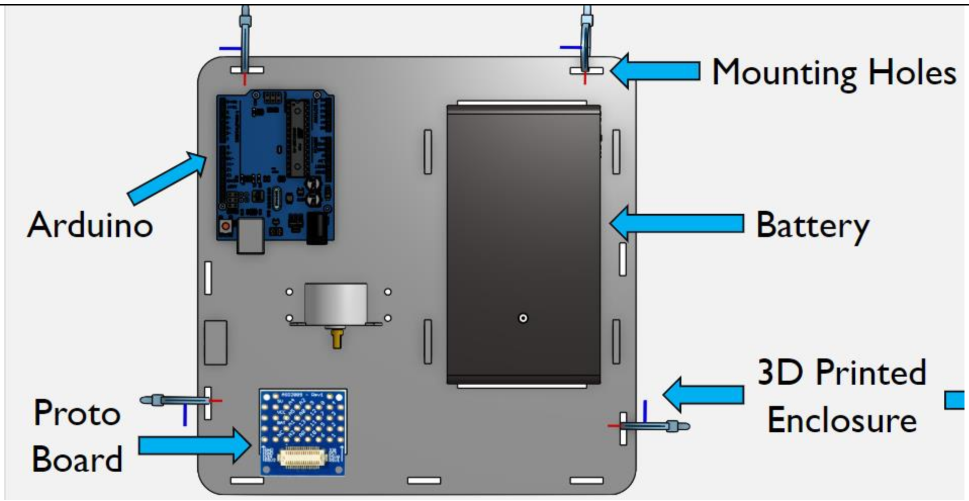


Figure 6.3: Control Box Model

The rotary encoder enclosure works similarly to the LED and Button boxes, with the addition of a screw hole. This hole allows for the lid to be secured to the bottom of the box. The rotary encoder fits snuggly inside, and the dial fits cleanly through the top. The nut shaped raised edge surrounding the dial hole, allows for the rotary encoders PCB as well as enclosure to be locked in rotation, by press fitting the box against the walker handle. The enclosure also has a loop located on the bottom of the box, which faces towards the inside of the walker when mounted. This allows for the use of tape, Velcro, or zip-ties to secure the enclosure to the frame of the walker.

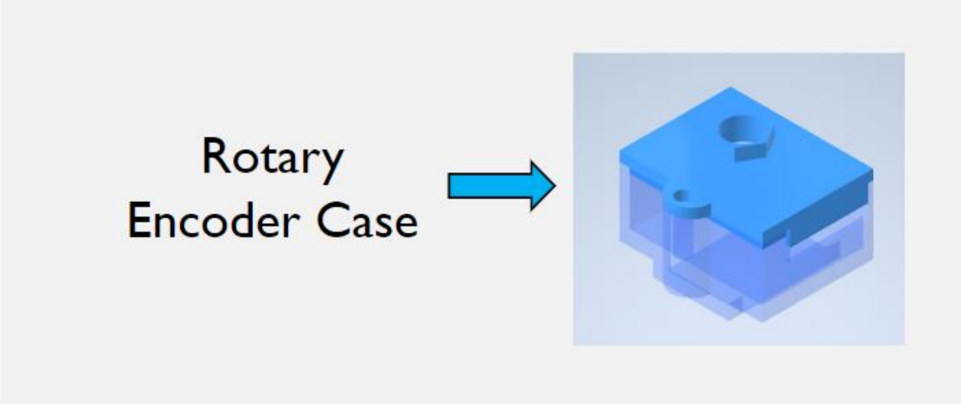
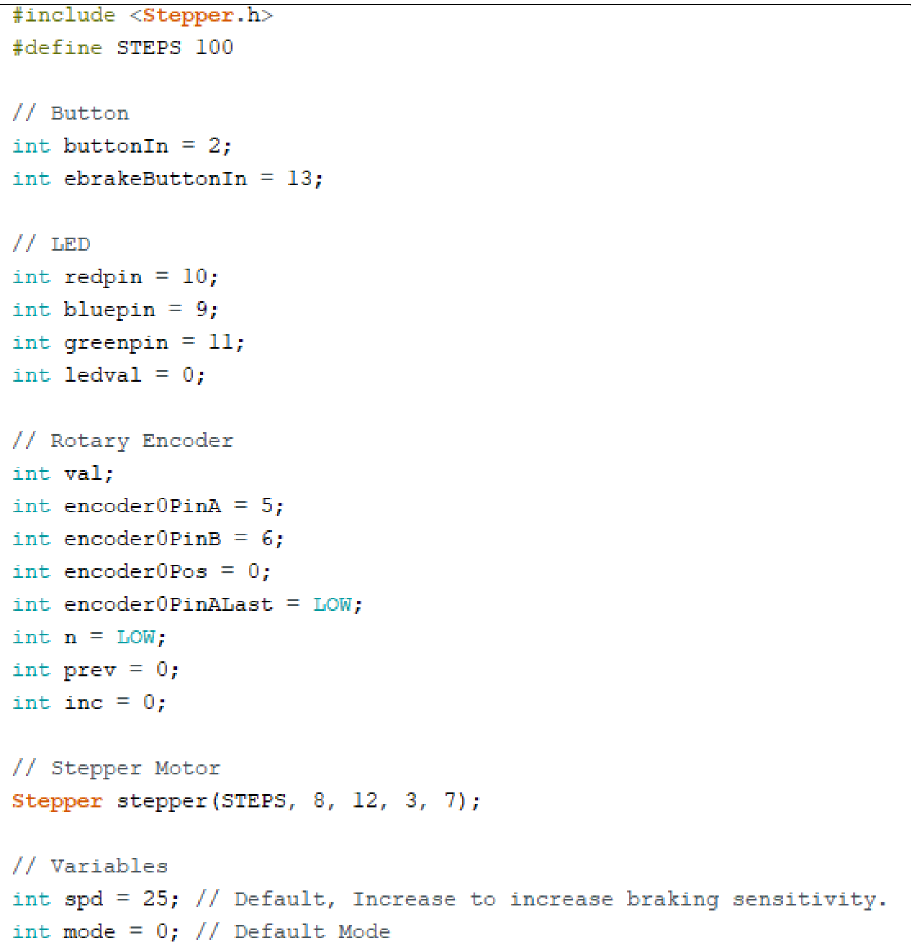


Figure 6.4: Rotary Encoder Enclosure

Software for prototype 1:

A program was written in C++ using the Arduino IDE, as seen below. Two different scripts were made. One that was used for our final prototype and was optimized for the functionality that we were able to complete by design day. The other was a more in-depth code that would allow for the full functionality of the walkers braking system, if the hardware made it onto the final walker.

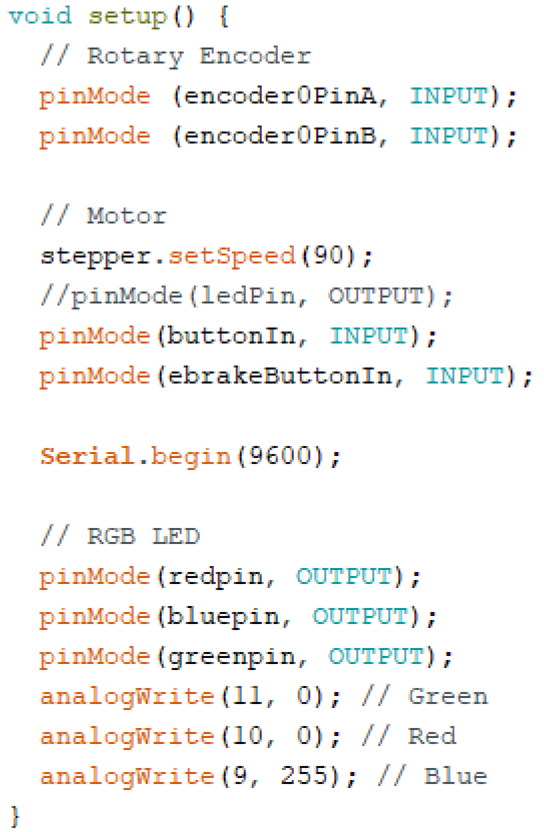
To begin we will focus on the prototype code, that has more functionality. The code begins by initializing all the components, and variables as seen in the figure below. It should be noted that this code was designed around the initial prototype of using only one stepper motor.



```
#include <Stepper.h>
#define STEPS 100

// Button
int buttonIn = 2;
int ebrakeButtonIn = 13;

// LED
int redpin = 10;
int bluepin = 9;
int greenpin = 11;
int ledval = 0;

// Rotary Encoder
int val;
int encoder0PinA = 5;
int encoder0PinB = 6;
int encoder0Pos = 0;
int encoder0PinALast = LOW;
int n = LOW;
int prev = 0;
int inc = 0;

// Stepper Motor
Stepper stepper(STEPS, 8, 12, 3, 7);

// Variables
int spd = 25; // Default, Increase to increase braking sensitivity.
int mode = 0; // Default Mode
```

Figure 6.5: Prototype 1 - Initialization

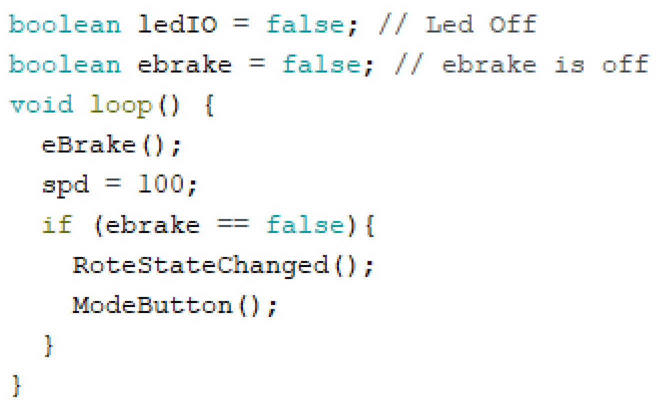
The code then goes on to enter the setup function, which will run once every time the walker is initially booted. The function initializes the Rotary encoder and sets whether the pins are input or output. This is then repeated for the motor setting the default speed. Lastly the RGB Led is initialized and set to the default color (Blue).



```
void setup() {
  // Rotary Encoder
  pinMode (encoder0PinA, INPUT);
  pinMode (encoder0PinB, INPUT);

  // Motor
  stepper.setSpeed(90);
  //pinMode(ledPin, OUTPUT);
  pinMode(buttonIn, INPUT);
  pinMode(ebrakeButtonIn, INPUT);

  Serial.begin(9600);

  // RGB LED
  pinMode(redpin, OUTPUT);
  pinMode(bluepin, OUTPUT);
  pinMode(greenpin, OUTPUT);
  analogWrite(11, 0); // Green
  analogWrite(10, 0); // Red
  analogWrite(9, 255); // Blue
}
```

Figure 6.6: Prototype 1 - Initialization Setup

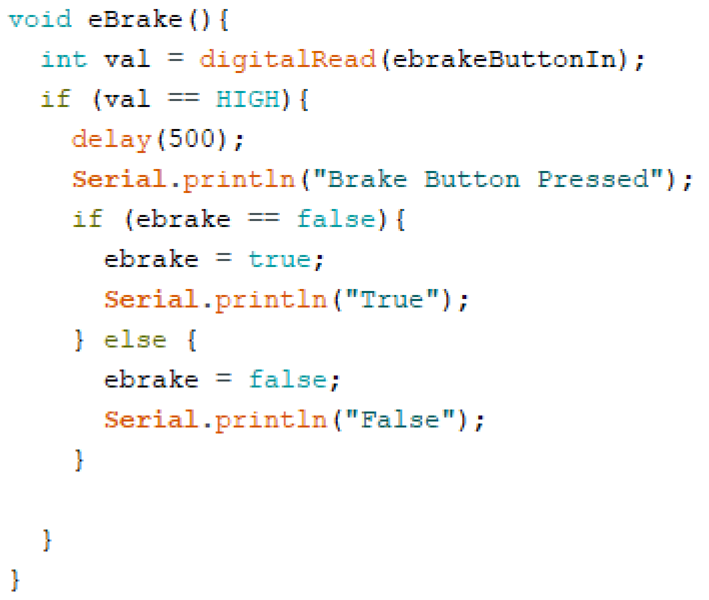
Next, we enter the loop function. This function runs continuously throughout the walkers use. Variables ledIO and e-brake are set to false, implying that the LED indicator is off or in its default mode, and the e-brake button has not been pressed. The initial speed is set to 100, which indicates the default braking sensitivity. The e-Brake function is called to check if the user has initialized the e-brake. If the e-brake is off meaning the user has yet to press the braking button it enters the first if statement which checks to see if the user has pulled the brake lever or changed the mode. Note that if the e-brake button was pressed this if statement would not be entered and the handle, and mode selection button would be disabled.



```
boolean ledIO = false; // Led Off
boolean ebrake = false; // ebrake is off
void loop() {
  eBrake();
  spd = 100;
  if (ebrake == false){
    RoteStateChanged();
    ModeButton();
  }
}
```

Figure 6.7: Prototype 1 - Loop

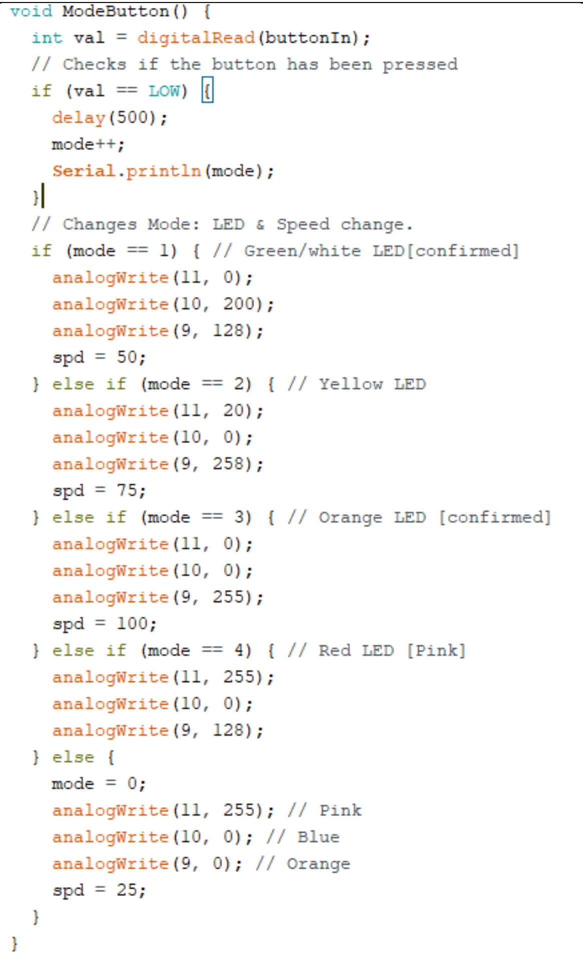
The e-brake function checks to see if the button that activates the brake has been pressed. It checks the last state of the button and then switches it to the next state, meaning that if the e-brake is on, the next press of this button will de-activate the brake, and vice versa.



```
void eBrake(){
  int val = digitalRead(ebrakeButtonIn);
  if (val == HIGH){
    delay(500);
    Serial.println("Brake Button Pressed");
    if (ebrake == false){
      ebrake = true;
      Serial.println("True");
    } else {
      ebrake = false;
      Serial.println("False");
    }

  }
}
```

Figure 6.8: Prototype 1 - e-Brake Function

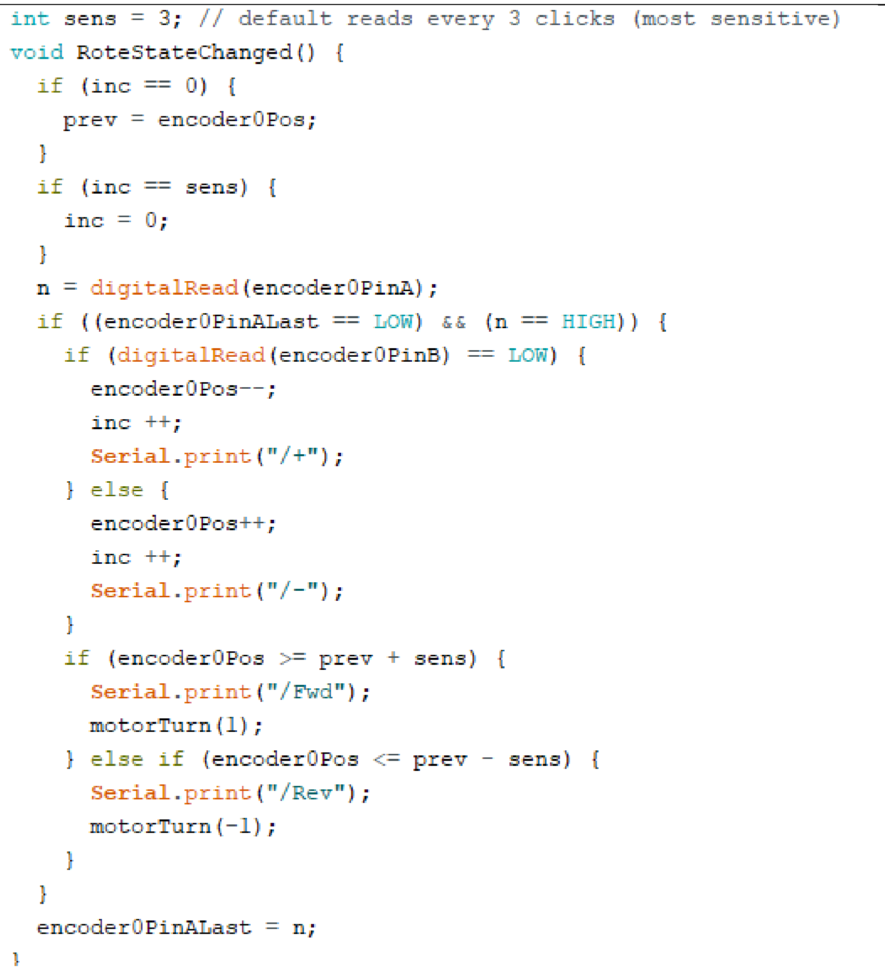
The mode change function checks to see if the mode change button has been pressed by the user. If the button has been pressed the code cycles through preset sensitivities. These sensitivities change how much the motor turns per degree the brake handle rotates. The mode is indicated by an RGB led which changes color from green being the least sensitive, to red being the most sensitive.



```
void ModeButton() {
  int val = digitalRead(buttonIn);
  // Checks if the button has been pressed
  if (val == LOW) {
    delay(500);
    mode++;
    Serial.println(mode);
  }
  // Changes Mode: LED & Speed change.
  if (mode == 1) { // Green/white LED[confirmed]
    analogWrite(11, 0);
    analogWrite(10, 200);
    analogWrite(9, 128);
    spd = 50;
  } else if (mode == 2) { // Yellow LED
    analogWrite(11, 20);
    analogWrite(10, 0);
    analogWrite(9, 258);
    spd = 75;
  } else if (mode == 3) { // Orange LED [confirmed]
    analogWrite(11, 0);
    analogWrite(10, 0);
    analogWrite(9, 255);
    spd = 100;
  } else if (mode == 4) { // Red LED [Pink]
    analogWrite(11, 255);
    analogWrite(10, 0);
    analogWrite(9, 128);
  } else {
    mode = 0;
    analogWrite(11, 255); // Pink
    analogWrite(10, 0); // Blue
    analogWrite(9, 0); // Orange
    spd = 25;
  }
}
```

Figure 6.9: Prototype 1 - Mode Selection function

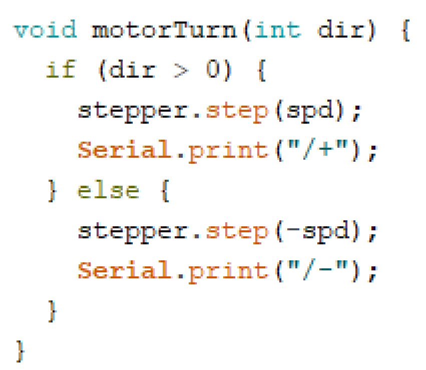
The RotateStateChanged() function checks to see if the rotary encoder has received input. When the user pulls the brake handle, it turns around a pivot pin. This pin is directly connected to the rotary encoder, meaning that with every degree the brake handle turns the rotary encoder also turns. This allows the rotary encoder to find which way the user is moving the handle. If the user is pulling the handle upwards the rotary encoder registers this as a positive value and increases a counter that means the handle is moving upwards, applying the brake. This call the next function motorTurn(1), with a positive value 1, meaning that we are applying the brakes. When the user releases the handle, the code resets this counter to 0, then counts how many degrees backwards the handle has turned, then calls motorTurn(-1) to indicate that the motor should turn the opposite direction releasing the brakes. The importance of the counter instead of turning for every input is to determine that it is only worth applying the change in position to the brakes if the handle has turned more than 3 degrees in any given direction. This is important to combat jitter, or miss inputs caused by the rotary encoder's inaccuracies. This implies that if the handle being jittered up and down only 1 or 2 degrees, the brakes do not change. Only when a 3 degree change in a given direction is found will the brakes be applied.



```
int sens = 3; // default reads every 3 clicks (most sensitive)
void RoteStateChanged() {
  if (inc == 0) {
    prev = encoder0Pos;
  }
  if (inc == sens) {
    inc = 0;
  }
  n = digitalRead(encoder0PinA);
  if ((encoder0PinALast == LOW) && (n == HIGH)) {
    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos--;
      inc ++;
      Serial.print("/+");
    } else {
      encoder0Pos++;
      inc ++;
      Serial.print("/-");
    }
    if (encoder0Pos >= prev + sens) {
      Serial.print("/Fwd");
      motorTurn(1);
    } else if (encoder0Pos <= prev - sens) {
      Serial.print("/Rev");
      motorTurn(-1);
    }
  }
  encoder0PinALast = n;
}
```

Figure 6.10: Prototype 1 - Rotary Encoder Function

The last function motorTurn(int) takes in an integer value. This value will determine which direction the motor should be turned, if positive, then the brake cables will be applied, and negative the motor will turn the opposite direction, unwinding the spool, and releasing the brake cables. It should be noted that in this case the motor turns at a given speed, selected in the mode selection function above. This speed indicates how far the motor turns per turning motion. The higher the speed the more the motor turns, per 3 degrees the handle is pulled.

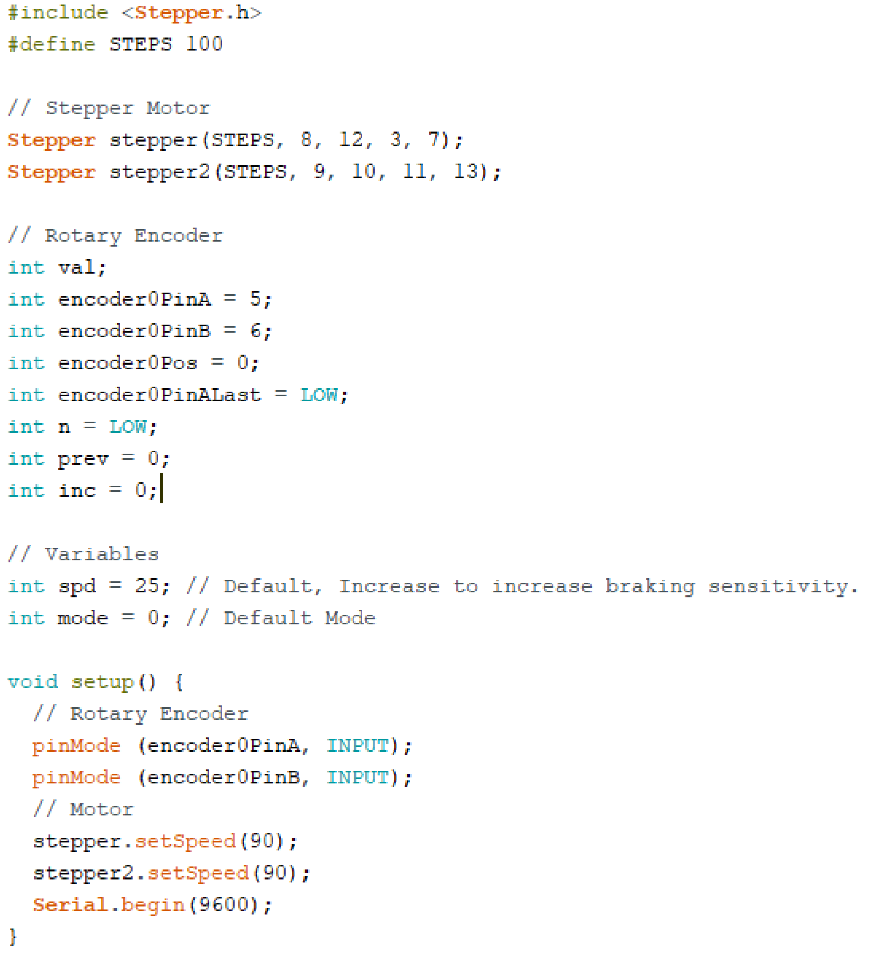


```
void motorTurn(int dir) {
  if (dir > 0) {
    stepper.step(spd);
    Serial.print("/+");
  } else {
    stepper.step(-spd);
    Serial.print("/-");
  }
}
```

Figure 6.11: Prototype 1 - Motor Turning Function

Software for Final Prototype:

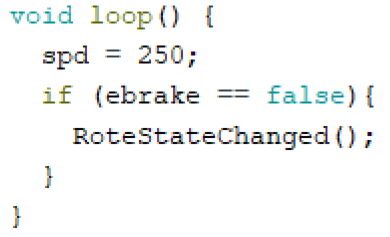
The second code created was a simplified and optimized to our final prototype. This prototype employs two motors and bypasses the jitter function.



```
#include <Stepper.h>
#define STEPS 100

// Stepper Motor
Stepper stepper(STEPS, 8, 12, 3, 7);
Stepper stepper2(STEPS, 9, 10, 11, 13);

// Rotary Encoder
int val;
int encoder0PinA = 5;
int encoder0PinB = 6;
int encoder0Pos = 0;
int encoder0PinALast = LOW;
int n = LOW;
int prev = 0;
int inc = 0;

// Variables
int spd = 25; // Default, Increase to increase braking sensitivity.
int mode = 0; // Default Mode

void setup() {
  // Rotary Encoder
  pinMode (encoder0PinA, INPUT);
  pinMode (encoder0PinB, INPUT);
  // Motor
  stepper.setSpeed(90);
  stepper2.setSpeed(90);
  Serial.begin(9600);
}
```

Figure 6.12: Final Prototype - Initialization

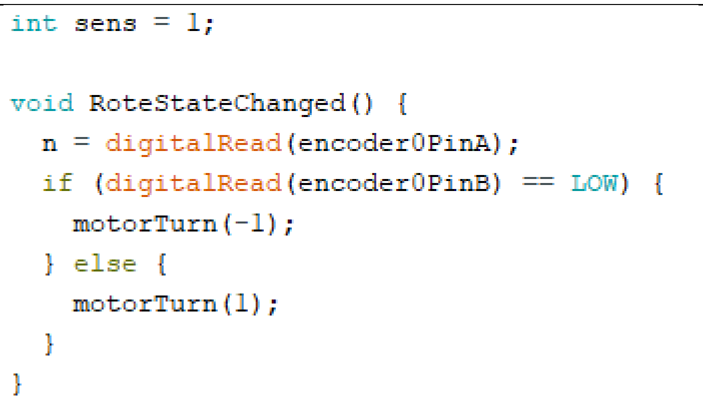
The loop function does not check for mode changes, instead is initialized the highest speed possible. The e-brake is also not included in this code and is always off.



```
void loop() {
  spd = 250;
  if (ebrake == false){
    RoteStateChanged();
  }
}
```

Figure 6.12: Final Prototype - Loop

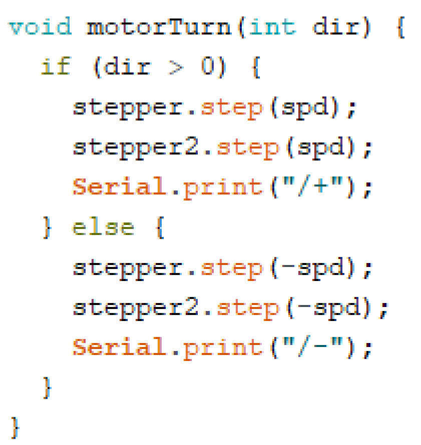
The rotary encoder's function is also simplified and compared to code one ignores the jitter algorithm. Meaning that the motor turns for every degree the handle is pulled, this leads to lots of miss readings, but is better optimized for testing purposes due to the hardware limitations. The rotary encoder used only clicks 3 times throughout the entire pulling of the handle, implying that it is only able to pull 3 data point in either direction. This is not enough to create an algorithm around, and instead turns the motor whenever the handle is pulled.



```
int sens = 1;

void RoteStateChanged() {
  n = digitalRead(encoder0PinA);
  if (digitalRead(encoder0PinB) == LOW) {
    motorTurn(-1);
  } else {
    motorTurn(1);
  }
}
```

Figure 6.13: Final Prototype – Rotary Encoder Function

The motor turning function is the same as before.



```
void motorTurn(int dir) {
  if (dir > 0) {
    stepper.step(spd);
    stepper2.step(spd);
    Serial.print("/+");
  } else {
    stepper.step(-spd);
    stepper2.step(-spd);
    Serial.print("/-");
  }
}
```

Figure 6.14: Final Prototype – Motor Function

Electrical:

The wiring of each output will be shown in the table below. It should be noted that for all parts the + and - terminals are attached to the protoboard on their corresponding ports. For most of the electronics used the – terminal will be labelled as GND. The protoboards + terminal will be attached to the 5V header on the Arduino, and the negative terminal to the GND header on the Arduino.

The protoboard wiring is outline in the figure below. The voltage is required to be constant across all electronics, so the + terminal is in parallel. This allows a constant 5V to be supplied to all the attached electronics. Further images of the wiring can be seen in the instructions portion of the document.

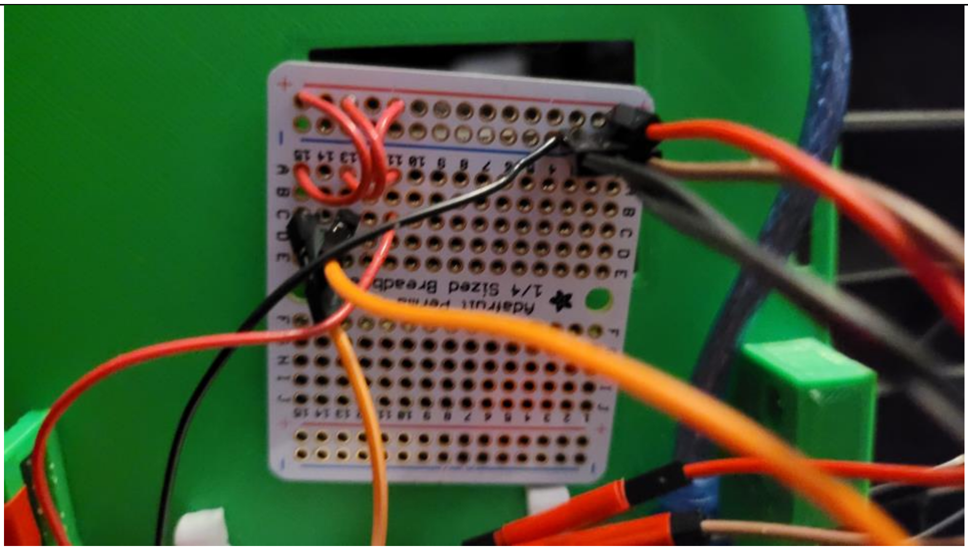


Figure 6.15: Protoboard Wiring

Note: The tables below outline two different prototypes. Prototype 1, employing only one motor. The final prototype employs two motor and excludes the LED and brake button. Prototype 1 has more functionality but requires the use of a stronger motor. The final prototype excludes functionality due to the limited available pins on the Arduino

Table 6.1: Arduino Wiring Prototype 1

Part Name	**Part Header Name**	**Arduino Pin**
Rotary Encoder	CLK	5

	DT	6
	SW	4
LED	R	10
	G	9
	B	11
Mode Button	V	2
E-Brake Button	V	13
Motor Controller	IN1	3
	IN2	7
	IN3	8
	IN4	12

Table 6.2: Arduino Wiring Final Prototype

Part Name	**Part Header Name**	**Arduino Pin**
Rotary Encoder	CLK	5
	DT	6
	SW	4
Motor Controller 1	IN1	3
	IN2	7
	IN3	8

	IN4	12
Motor Controller 2	IN1	9
	IN2	10
	IN3	11
	IN4	13

6.1 <Subsystem 1 of prototype>

6.1.1 BOM (Bill of Materials)

Table 6.1.1 - Bill of Materials

Item	Cost ($)	#	Supplier	Link
Arduino Uno	16.98	1	Amazon	https://www.amazon.ca/s?k=arduino+uno&ref=nb_sb_noss
Velcro Ties (10)	1.89	1	Amazon	https://www.amazon.ca/VELCRO-Brand-Reusable-Fastening-Organizing/dp/B001E1Y5O6/ref=sr_1_5?dchild=1&keywords=Velcro+Ties&qid=1622304742&sr=8-5
Portable Charger	25.45	1	Amazon	https://www.amazon.ca/26800mAh-Ultra-High-Capacity-Universal-LED-Indicator/dp/B0869DMJH7/ref=sr_1_1_sspa?dchild=1&keywords=portable+charger&qid=1622304846&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyV1M5TUhXWDRCWDEmZW5jcnlwdGVkSWQ9QTA5NDI3NzkzMkFZUTdBUE1aVEVQJmVuY3J5cHRlZEFkSWQ9QTA2MzAwMTJCVVdIMktUS0lCN1Umd2lkZ2V0TmFtZT1zcF9hdGYmYWN0aW9uPWNsaWNrUmVkaXJlY3QmZG9Ob3RMb2dDbGljaz10cnVl

Rotary Encoder (5)	11.16	1	Amazon	https://www.amazon.ca/Willwin-KY-040-Encoder-Development-Arduino/dp/B0778MX39R/ref=sr_1_2_sspa?dchild=1&keywords=rotary+encoder&qid=1622304939&sr=8-2-spons&psc=1&smid=AVKVKSQRPUCTR&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyWTJGR0tHSDNNNklFJmVuY3J5cHRlZElkPUEwNDQ5ODU4SUxGNU03VTRZRUI3JmVuY3J5cHRlZEFkSWQ9QTA3Mjg2MDcyM045VVNGUkVTVlk0JndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==
Button	12.98	1	Amazon	https://www.amazon.ca/Gikfun-12x12x7-3-Tactile-Momentary-Arduino/dp/B01E38OS7K/ref=sr_1_12?dchild=1&keywords=arduino+button&qid=1623440038&sr=8-12
Wire Red (5ft)	1.60	2	Makerstore	https://makerstore.ca/shop/ols/products/5ft-hook-up-wire-22awg-red
Wire Black (5ft)	1.60	2	Makerstore	https://makerstore.ca/shop/ols/products/5ft-hook-up-wire-22awg-black
Stepper Motor	10.78	1	Amazon	https://www.amazon.ca/Nema17-Stepper-42BYGH-17HS4401S-Printer/dp/B0787BQ4WH/ref=sr_1_1_sspa?dchild=1&keywords=stepper+motor&qid=1622305199&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExQU1VRFFaTjhCNlFJJmVuY3J5cHRlZElkPUEwNTc4NDYwMlpFVkZXOFVBRjNMOCZlbmNyeXB0ZWRBZElkPUEwNDIyNDc4MzJJSDVUUUtKVFVBRCZ3aWRnZXROYW1lPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU=
Nuts & Bolts .11” Diam, ½" Length	.25	8	Hardware Store	
3D Print (Est)	5.00	1		
Total	**$ 91.64**	**11**		
Total Tax	**$ 103.55**	**11**		

6.1.2 Equipment list

Tools required

- Screwdriver (Philips and flat head)
- Allan key set (Varying sizes deepening on walker brand)
- Drill (Optional)
- Hammer
- Zip-ties, tape or Velcro-ties (Varying sizes)

6.1.3 Instructions

Removing the Brake Cable:

1. Undo all screws holding the brake cable to the walker's handle

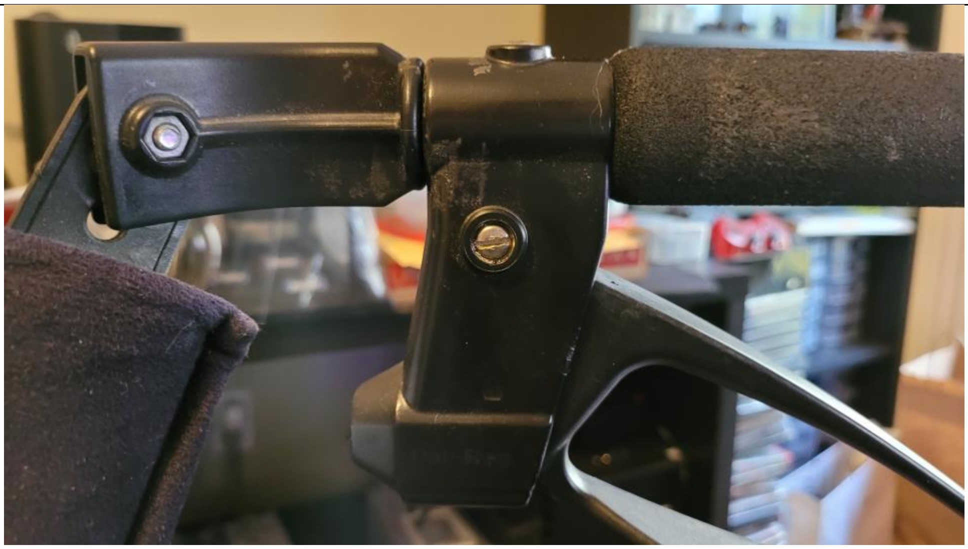


Figure 6.16: Removing the Brake cable from the walker handle

2. Loosen the tensioning screw at the base of the walker over each tire, pull the walkers braking cable through.

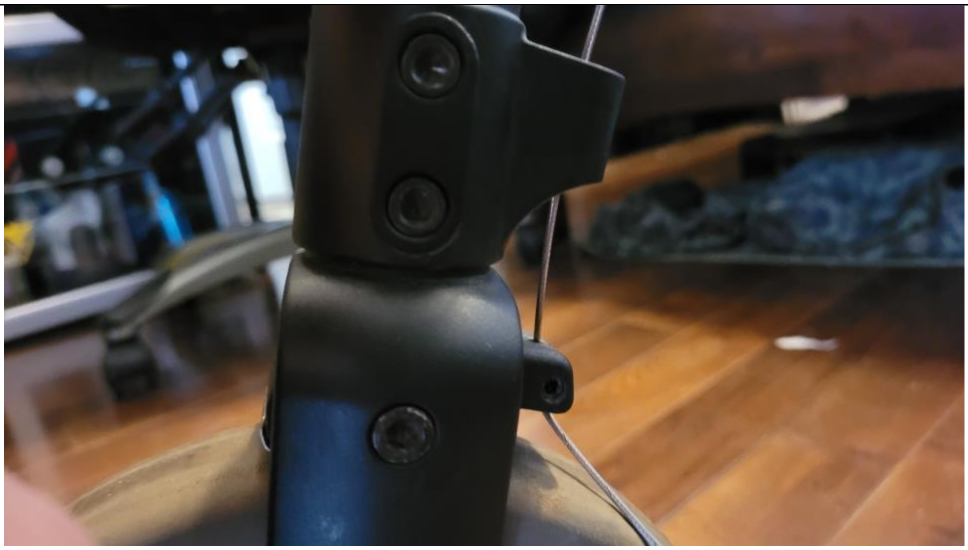


Figure 6.17: Loosening the tensioning screw

Rotary encoder installation:

1. Remove the screw and pin holding the handle on the walker.

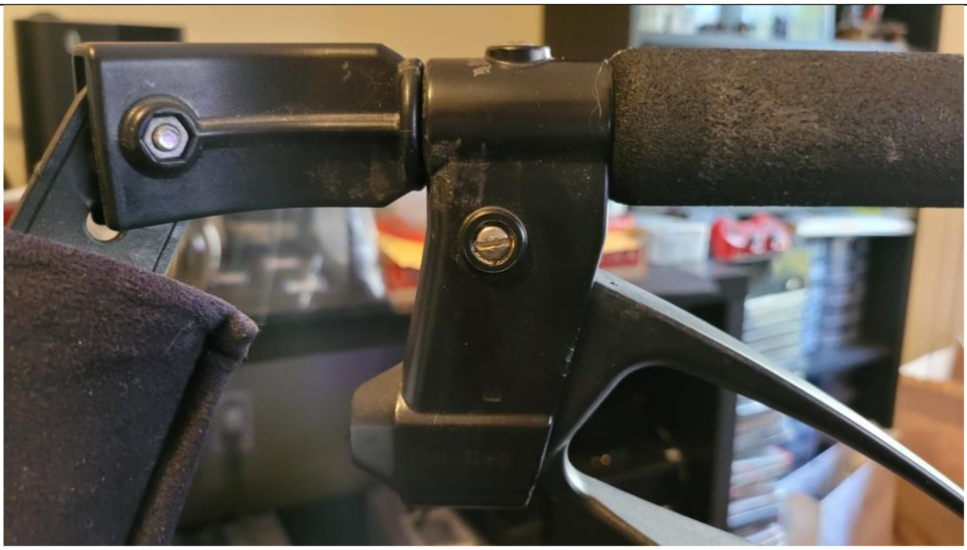


Figure 6.18: Handle Walker Screws

2. Place the rotary encoders dial through the hole in the handle. Ensure the body of the encoder is on the inside of the walker and the output pins are facing downwards.

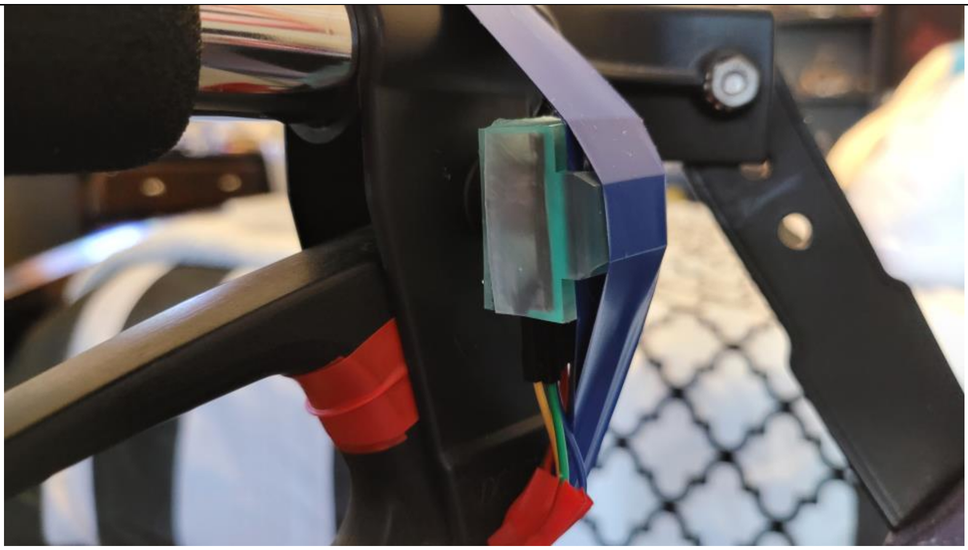


Figure 6.19: Rotary Encoder Mount

3. Using a hammer, lightly tap the friction pin in place, ensuring that the pin is flush with the body of the walker handle. You may have to rotate the pin to get it to line up with the receiving end of the rotary encoder.

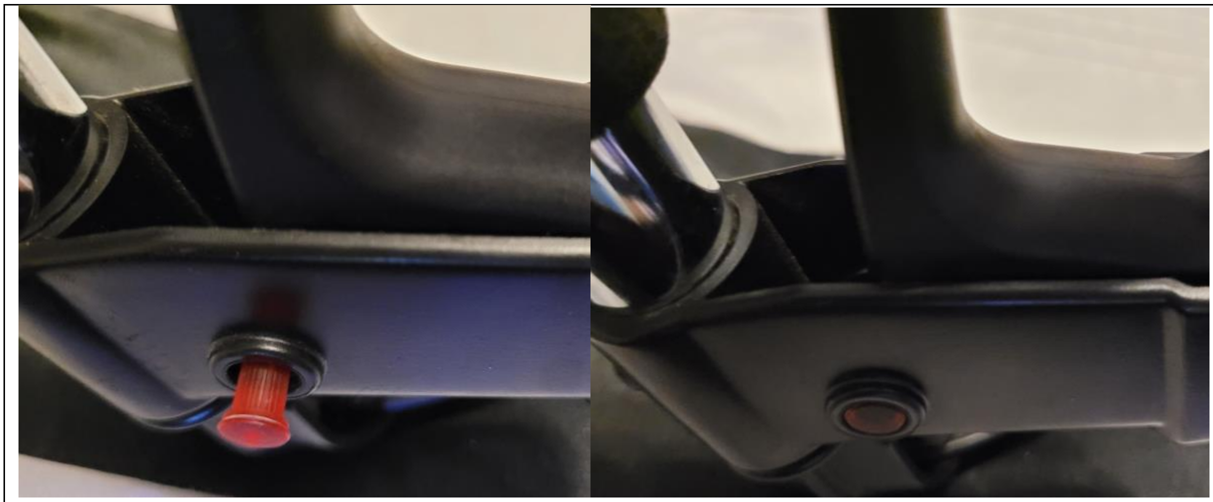


Figure 6.20: Rotary Encoder Pin

4. Using tape, Velcro ties, or zip ties; secure the rotary encoders enclosure to the side of the walker's handle, ensuring that the rotary encoder does not rotate when the handle is pulled. The handle should be able to be pulled with minimal resistance.

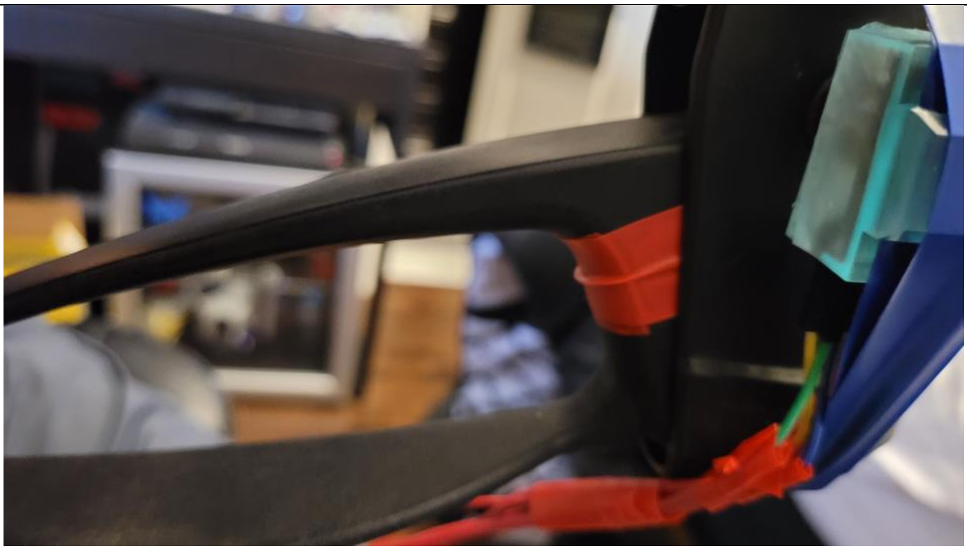


Figure 6.21: Rotary Encoder Securing

Control Box Installation:

1. Mount the control box under the seat using Velcro Ties, zip-ties or tape.

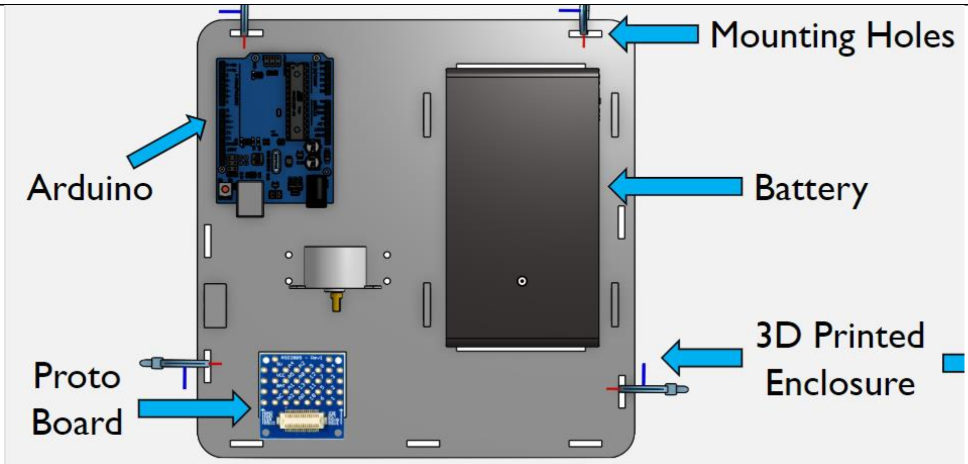


Figure 6.22: Part Layout for mounting

2. Mount he Arduino the control box by pushing the Arduino into the corresponding pin holes. No screws are required as the Arduino holes are designed to be push fit. See Figure 6.7 above for the part layout
3. Mount the battery to the underside of the walker using the tie down points on the control box. The battery can be mounted using zip-ties, Velcro or tape. See Figure 6.7 for part layout.

Motor Mounting:

1. To mount the motors, there should be two 3D printed motor mounts.

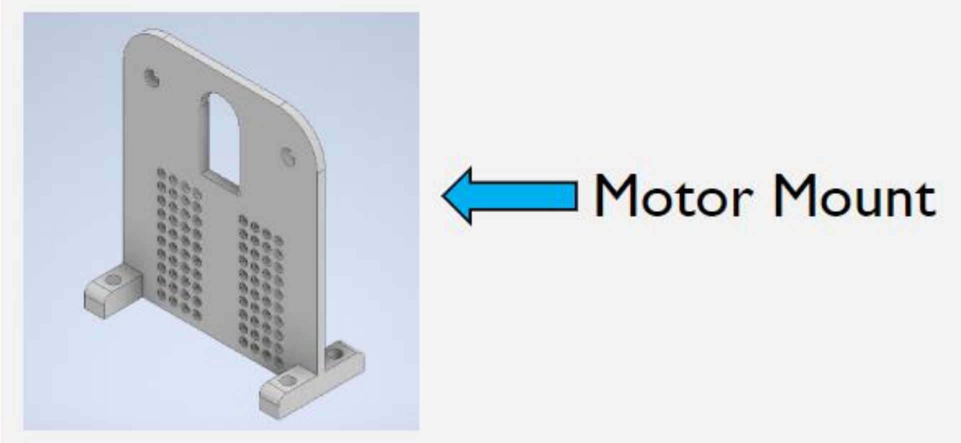


Figure 6.23: Motor Mount

2. Place the mount under the seat as close to the outer wall as possible on the left-hand side. Using a marker mark the layout of the 4 holes.
3. Drill 4 1/4” holes into the seat. Note this does not void the walker's warranty, as the seat is not covered.
4. Place the motor mount underneath the seat
5. Fasten the motor mount using a .11” diameter, ½" length bolt, and its accompanying nut.

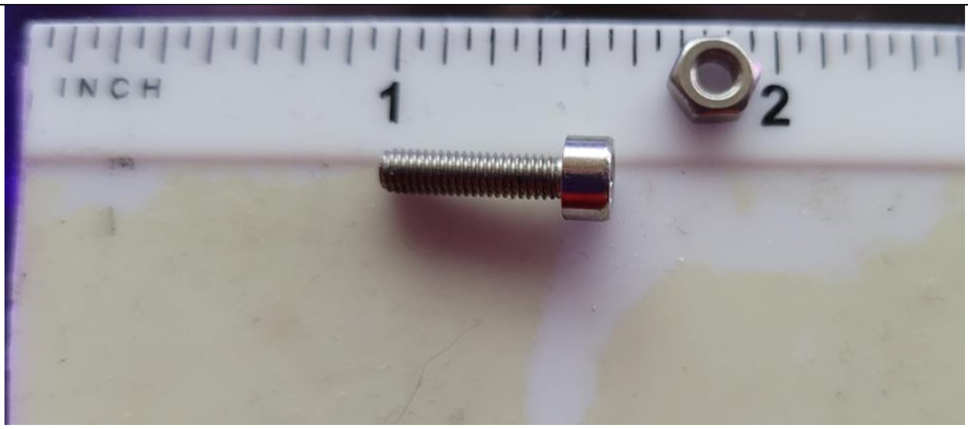


Figure 6.24: Nuts & bolts

6. Place the motor so that the motor body is facing the inside of the walker and the chrome rotating portion is facing the outside. Using the same nut and bolts listed above secure the motor to the motor mounts.

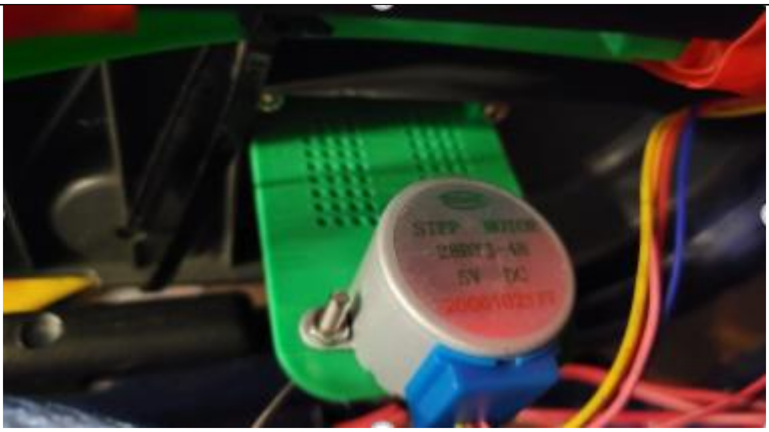


Figure 6.25: Motor Mount

7. Repeat steps 2-6 for the right side of the walker.
8. Push fit the provided spools onto the motor

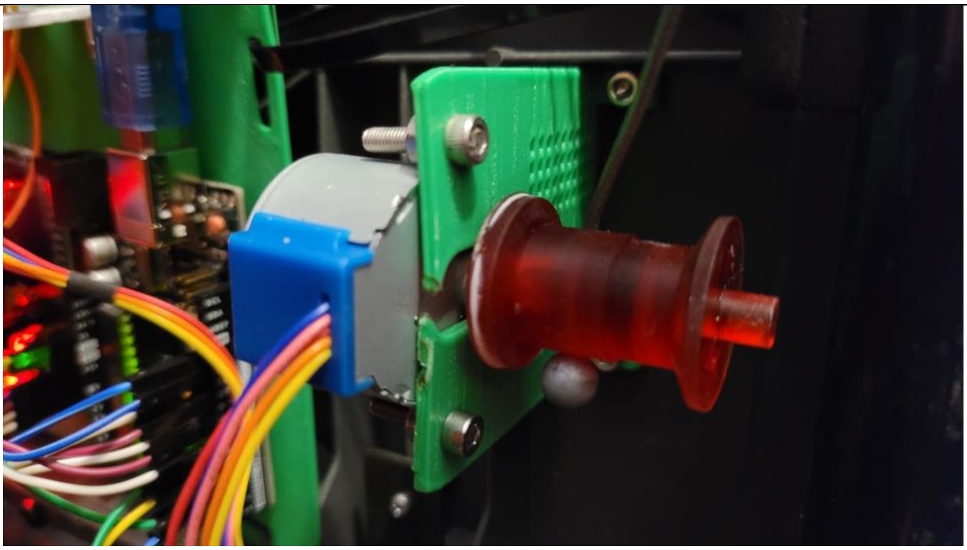


Figure 6.26: Spool Location

9. Thread the brake cable through the spool in any orientation, and then back down to the wheels.

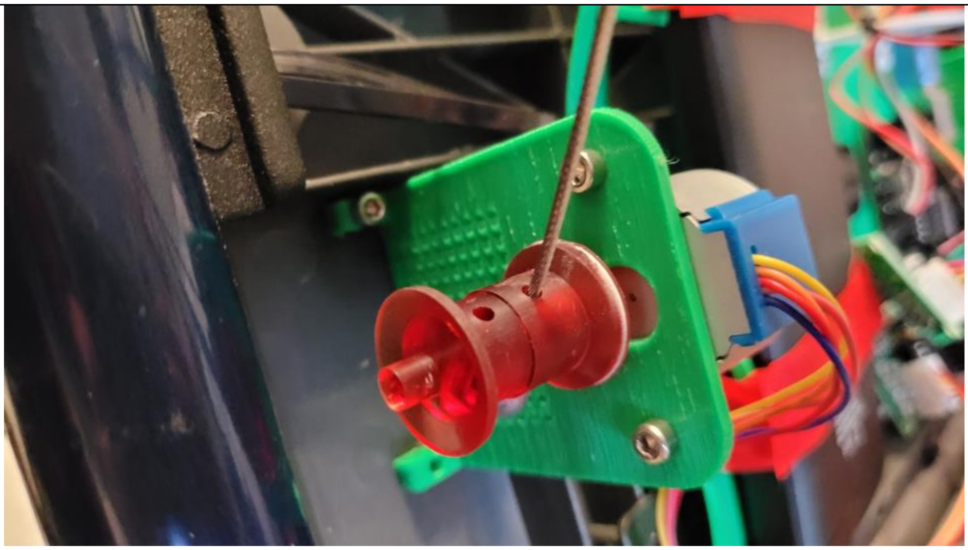


Figure 6.27: Cable Routing

Wiring:

1. To wire the rotary encoder plug in the provided cables or make your own as needed. The cables should have 4 prongs. The 4 prongs are color coded for easy. The yellow prong attaches to the rotary encoder in the CLK pin. The green to the DT. The blue to the SW. The red and black cables are then connected to + and ground respectively.
2. The cables are then routed downwards toward the control box using Velcro, zip-ties, or tape as a guide.

3. The + and Ground cables should be pre attached to a protoboard using the layout provided below. If they are not already applied, or the cables came undone during handling, please soldering the + and GND terminals following the image below.

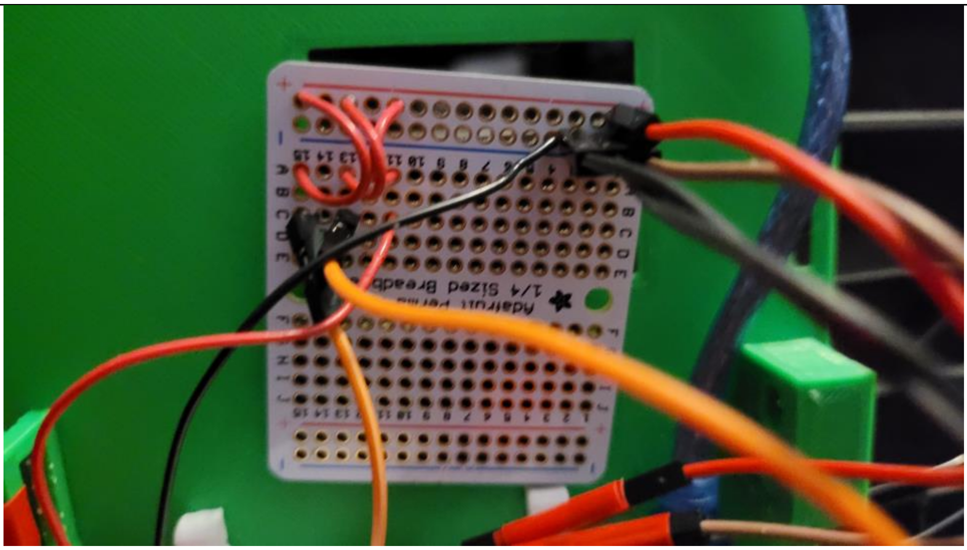


Figure 6.28: Protoboard Wiring

4. Connect the yellow, green, and blue cables to pin 5, 6, and 4 on the Arduino respectively.

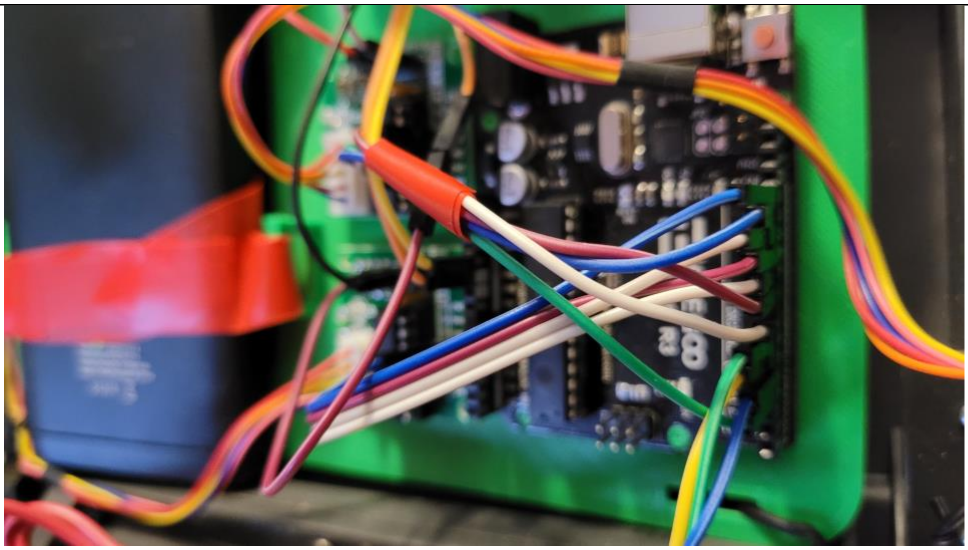


Figure 6.29: Arduino Wiring Layout

5. As for the motors, connect the push plug into the motor control board. Secure the motors control board under the walker using double sided tape or screwing it to one of the mounting holes using the same screws as other steps.

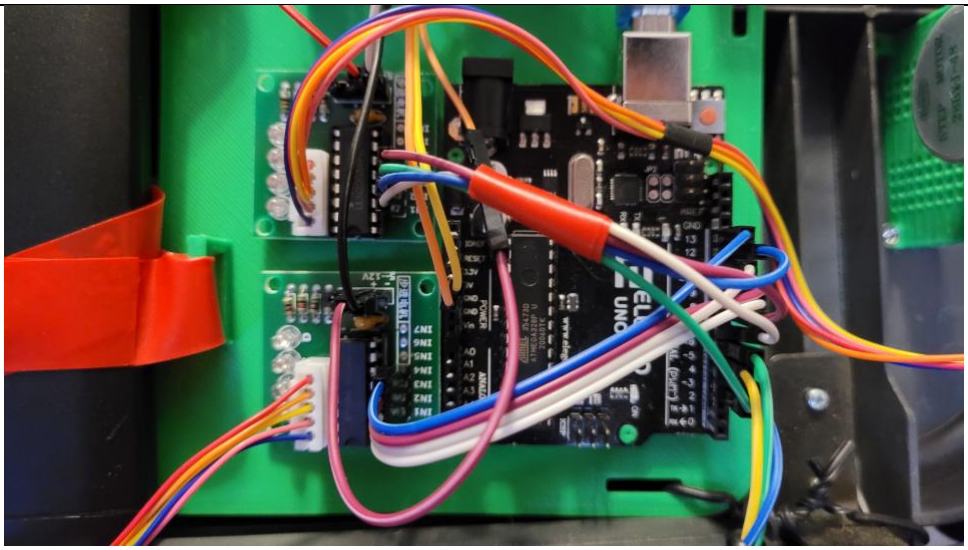


Figure 6.30: Control boards

6. On the motor control board connect the + and GND terminals from the protoboard, these cables should already be applied, if not solder them following the image provided in step 3. The data pins IN1-IN4 should be connected to the Arduino pins 9, 10, 11, 13 for one of the motors, for the second motor connect the IN1-IN4 to pins 3, 7, 8, and 12, respectively.
7. Finally connect the + and - cables coming from the protoboard to the 5V out and GND pins respectively on the Arduino. Once again, these cables should be pre soldered, if not, follow the wiring layout provided in step 3.
8. Secure the protoboard to the bottom of the control box using double sided tape or screwing it to one of the provided mounting holes.

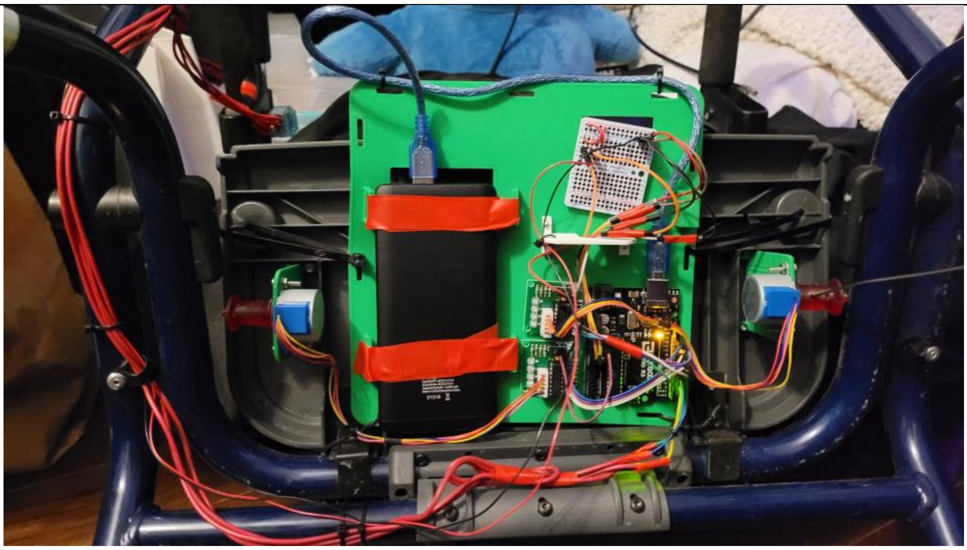


Figure 6.31: Protoboard with respect to control box

9. Finally plug the USB from the Arduino into the USB output on the portable power bank. Note that when plugged in the system is slowly draining the battery. If the walker is being left for extended periods of time it is best to unplug the Arduino, to prevent the battery from being drained.

Tensioning the brake cable:

1. To properly tension the brake cable the system should be powered on and the brake handle in the down or off position.
2. Thread the brake cable back through the original hole on the base of the walker above the tire and tension the cable such that it is taught but the brakes are not yet applied.
3. Tighten the retention screw down so that the cable is in place.

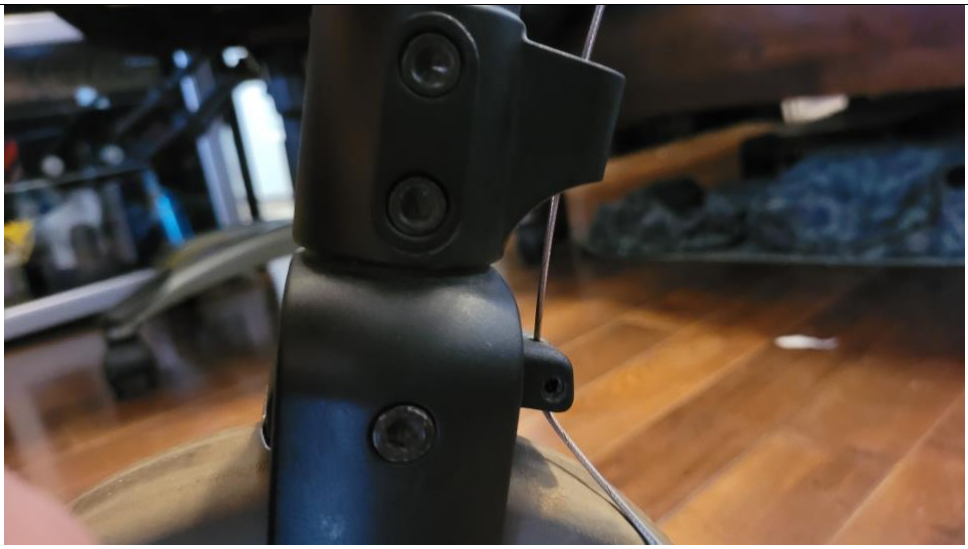


Figure 6.32: Brake Cable Tension

4. Pull the brake handle, the motors should turn winding up the brake cable. The brakes should then be applied. If the brakes are not yet applied, or the braking is weak. Repeat the above steps, tensioning the brake cable.

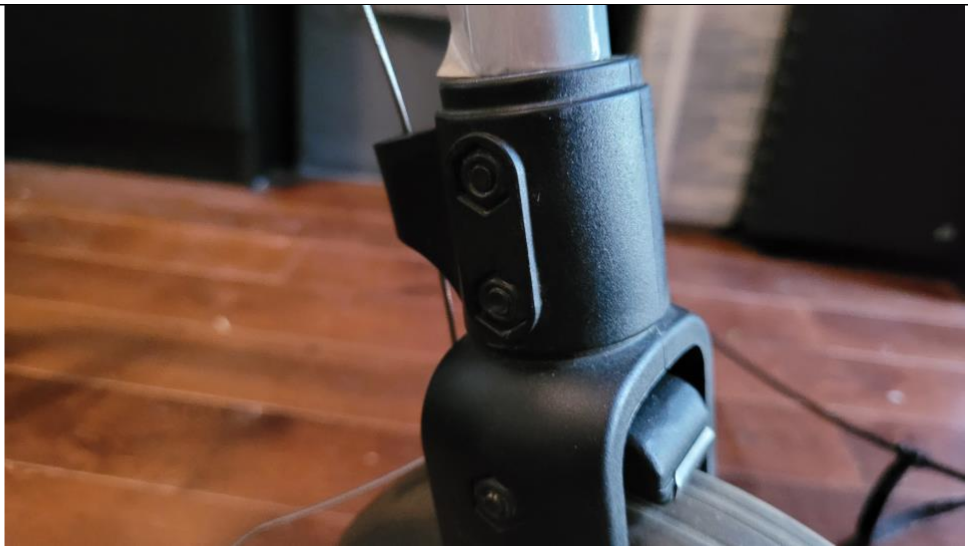


Figure 6.33: Proper Tensioning

6.2 Testing & Validation

Tests that were conducted included testing the accuracy of the braking system. This included applying and releasing the brakes quickly, and different ways the user could apply the brakes. Other tests included the battery life, and optimization of the code to increase the life of the battery. The tests concluded that the battery would be able to keep up with an entire day of use and could possibly be tuned for more.

Known Issues:

- If the brakes are not applied fast enough a miss input may be received, and the brakes may not be applied
- There is a delay with the braking system between the user input and the brakes being applied. This can be fixed by replacing the rotary encoder by a far more precise one.

7 Conclusions and Recommendations for Future Work

In conclusion, the purpose of this report was to provide the reader with full documentation of the development of the walker braking system, its function, and how it can be used to effectively activating the brakes of the walker with one hand. Throughout the development process, Witty Walkers has grown as a team and learned many important lessons. Primarily, as a group of 3 members, we learned the importance of clear communication and feedback with each other to get work done on time. Furthermore, another lesson learned was that there is never a wrong idea and that there are diverse ways to approach problems. Included in this report are aspects of the design that can be improved to optimize the one-handed walker braking system. Specifically, two important safety related improvements to be made include a waterproof enclosure, as well as an emergency brake. It is important to note that the ability to be waterproof was initially included in the design but was not executed in the final protype due to various limitations. Additionally, when the brakes are applied, the delay between the motor being applied was minimized to the best of our abilities. However, to improve upon the work of team Z4, a better rotary encoder or a gear system should be implemented to allow the rotary encoder to collect more data over the limited distance the handle is turned. This will allow the encoder to plot more data points and allow more precise braking, as well as speed up the braking process. To fulfil the goal of Witty Walkers, the aforementioned improvements will provide individuals with physical disabilities with a safe, waterproof, and user-friendly one-handed walker braking system.

8 Bibliography

Current, E. (2016, August 1). Eddy Current Electromagnetic Brakes. Retrieved May 23, 2021, from https://patents.google.com/patent/US5656001A/en

Invacare. (2015), Dolomite Rollators. Retrieved May 16, 2021, from http://www.invacare.ca/doc_files/Dolomite%20Rollators_05_430C.pdf

Invacare. (2021), Dolomite Alpha Basic Walker. Retrieved May 16, 2021, from http://www.invacare.ca/cgi-bin/imhqprd/inv_catalog/prod_cat_detail.jsp?prodID=12230-02-35-87

Mobility-Aids. EVA Electric Adult Support Walker. Retrieved May 16, 2021, from https://www.mobility-aids.com/eva-electric-adult-support-walker-wide.html#:~:text=The%20EVA%20Electric%20Adult%20Support%20Walker%20Wide%20is,wider%20entrance%20for%20wheelchair%20users%20in%20medical%20institutions.

Nova, (2016), Nova Cruiser Series Rolling Walker. Retrieved May 16, 2021, from https://www.novajoy.com/our-products/rolling-walkers/cruiser-series-rolling-walkers/

APPENDICES

9 APPENDIX I: Design Files

Table 2. Referenced Documents

Document Name	Document Location and/or URL	Issuance Date
MakerRepo	https://makerepo.com/laurenhealyyy/909.gng2101z4-witty-walkers-enhanced-walker-braking-system	July 2, 2021

10 APPENDIX II: Other Appendices

All features were not able to be added to the final prototype but was designed to have them.