File: P21311 Joystick Controller Documentation

MSD Team: Mind Controlled Wheelchair

Owner: Sidney Davis - sxd9336

Date: 5/9/2021

Table of Contents

[1. Purpose 3](#_Toc71474686)

[1.1. Project Overview 3](#_Toc71474687)

[1.2. Design Choice 3](#_Toc71474688)

[2. Wheelchair Joystick Specifications 4](#_Toc71474689)

[2.1. Original Specs 4](#_Toc71474690)

[2.2. Observed Specs 4](#_Toc71474691)

[2.3. Observed Design 5](#_Toc71474692)

[3. Joystick Controller V0 6](#_Toc71474693)

[3.1. Schematic 6](#_Toc71474694)

[3.2. Theory of Operation 6](#_Toc71474695)

[3.3. Layout 6](#_Toc71474696)

[3.4. Discovered Issues 7](#_Toc71474697)

[4. Joystick Controller V1 8](#_Toc71474698)

[4.1. Schematic 8](#_Toc71474699)

[4.2. Theory of Operation 8](#_Toc71474700)

[4.3. Layout 9](#_Toc71474701)

[4.4. Discovered Issues 9](#_Toc71474702)

[5. Joystick Controller V2 10](#_Toc71474703)

[5.1. Schematic 10](#_Toc71474704)

[5.2. Theory of Operation 10](#_Toc71474705)

[5.3. Layout 10](#_Toc71474706)

[5.4. Connecting to the Wheelchair 11](#_Toc71474707)

[5.5. Discovered Issues 11](#_Toc71474708)

[5.6. Re-Work 11](#_Toc71474709)

[5.7. Future Investigations 12](#_Toc71474710)

[6. Joystick Controller V3 13](#_Toc71474711)

[6.1. Schematic 13](#_Toc71474712)

[6.2. Theory of Operation 13](#_Toc71474713)

[6.3. Layout 14](#_Toc71474714)

[6.1. Connecting to the Wheelchair 14](#_Toc71474715)

[6.2. Discovered Issues 14](#_Toc71474716)

[6.3. Future Investigations 15](#_Toc71474717)

[7. Appendix 16](#_Toc71474718)

[7.1. Joystick Controller V0 Arduino Code 16](#_Toc71474719)

[7.2. Joystick Controller V1 Arduino Code 18](#_Toc71474720)

[7.3. Joystick Controller V2 Arduino Code 22](#_Toc71474721)

[7.4. Joystick Controller V3 Arduino Code 27](#_Toc71474722)

# Purpose

## Project Overview

For the Mind Controlled Wheelchair project, the goal is to modify an electric wheelchair to drive semi-autonomously with occasional heading instructions from a BCI headset. Part of this modification involves “hacking” the joystick console that comes with the wheelchair to allow our motherboard to move the wheelchair.

## Design Choice

Our team has elected to design an electrical interface that intercepts the analog signals coming out of the joystick sensor before reaching the PCB, and replacing it with our own analog signals.

The unit producing these analog signals at our command will be an Arduino Uno. Since the Uno doesn’t have any analog output, however, the PWM output is being filtered into an analog signal. Further details in the “Joystick Controller Design” section.

Commands are sent to the Arduino via USB. Currently, the master is a Raspberry Pi 4, but could easily be adapted to another “motherboard” (Jetson, Laptop, Desktop, etc.) so long as it has an open USB port on it.

# Wheelchair Joystick Specifications

## Original Specs

A full description of the original wheelchair controller can be found in

[R-net-Electronics-Technical-Manual-v6.pdf](https://permobilus.com/wp-content/uploads/2016/11/R-net-Electronics-Technical-Manual-v6.pdf)

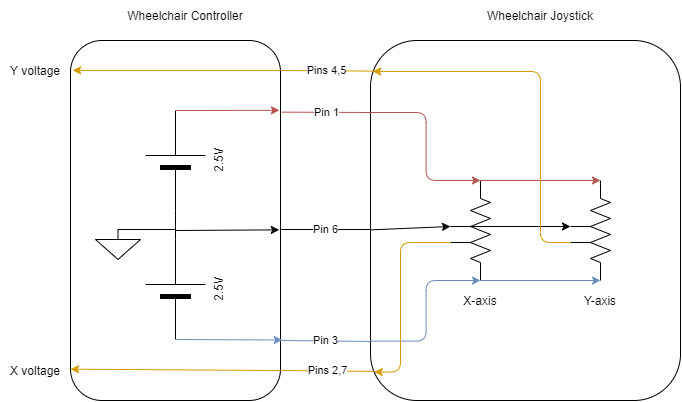
## Observed Specs

These voltages were probed between live wires and ground pad on PCB. fwd, rvs, left and right refer to pushing the joystick to its max position and measuring the resulting voltage. The pin numbering is based on the silk-screen values on the PCB the joystick connects to.

Table : Joystick Observed Voltages and Functionality

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pin | Rest  (V) | Forward  (V) | Reverse  (V) | Left  (V) | Right  (V) | Function |
| 1 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | VDD |
| 2 | 0 | 0 | 0 | -1 | 1 | X1 |
| 3 | -2.5 | -2.5 | -2.5 | -2.5 | -2.5 | VSS |
| 4 | 0 | 1 | -1 | 0 | 0 | Y1 |
| 5 | 0 | 1 | -1 | 0 | 0 | Y2 |
| 6 | 0 | 0 | 0 | 0 | 0 | GND |
| 7 | 0 | 0 | 0 | -1 | 1 | X2 |
| 8 | 0 | 0 | 0 | 0 | 0 | N/C |

## Observed Design



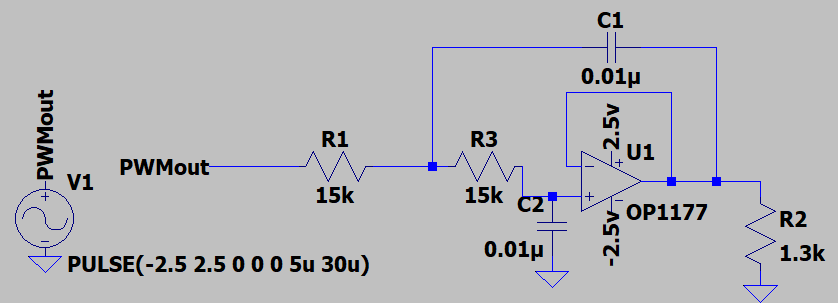
Based on the measurements in Table 1, this schematic was derived. The joystick consists of two center-tapped potentiometers, and the wipers on the potentiometers are determined by the position of the joystick. A corresponding voltage is returned to the controller that is providing the positive and negative rails to both elements.

Using this design, an equivalent circuit to the joystick was implemented on a breadboard using two 10k potentiometers and 2 equal resistors to make a voltage divider between the two rails to represent the center tap.

When connecting this breadboard to the wheelchair controller, the wheelchair screen showed no errors, and upon twisting the potentiometers the chair began to move, validating the schematic in Figure X. The motors turned slower than anticipated, but this was later determined to be due to the position of the seat of the chair, which lowers the max velocity if the user is not reclined back past an internal threshold.

# Joystick Controller V0

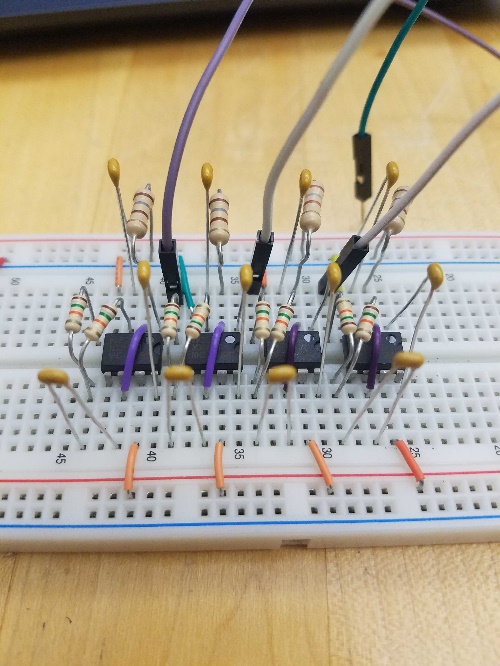
## Schematic



## Theory of Operation

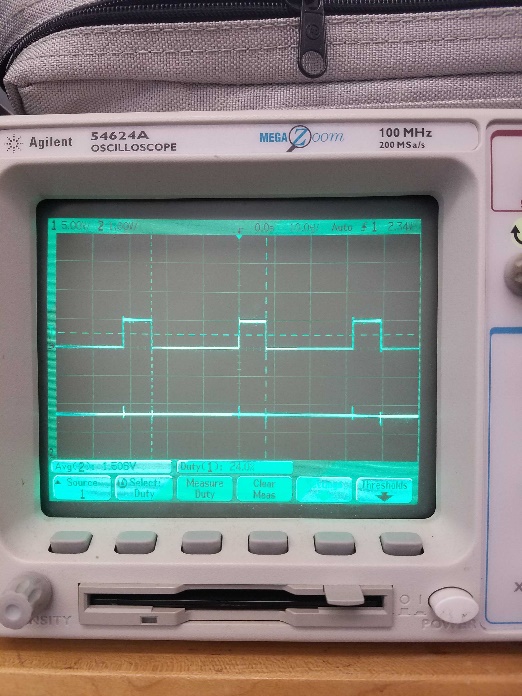
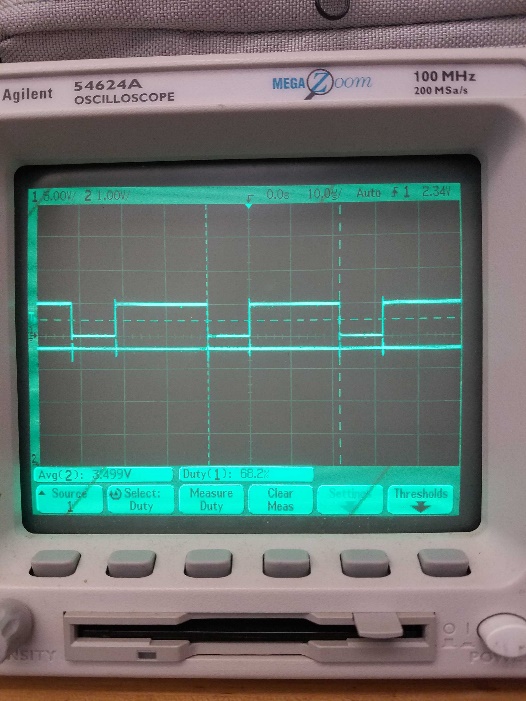
The first attempt at creating a circuit to control the joystick focused on producing an analog voltage given a ~30kHz PWM input, and allowing any needed current flow in that process. This was achieved using a Sallen Key Low Pass Filter tuned to a cutoff frequency of ~1000Hz

## Layout



Nothing more than a Bread-board implementation was performed for testing

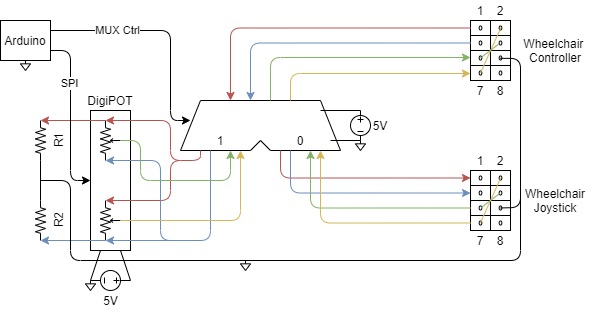
## Discovered Issues

Although the analog circuit could produce a flat DC signal that ranged from 0V to 5V, there was noise on each edge of the input signal that caused the output to jump. We did not want to risk the wheelchair interpreting these jumps, so the analog circuit idea was abandoned before a PCB layout was designed.

# Joystick Controller V1

## Schematic



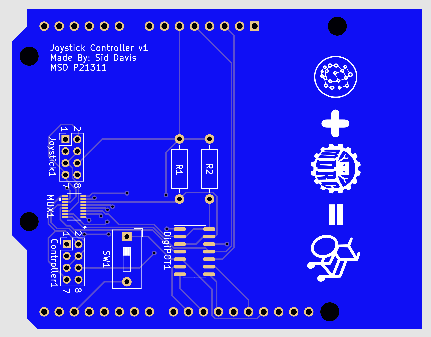
## Theory of Operation

When an analog circuit was abandoned, the possibility of digital ICs was taken advantage of. The Arduino could represent the joystick using a AD8402ARZ1 Digital Potentiometer, and a separate set of resistors between the rails to mimic the center-tapped grounding point.

Furthermore, since one integrated circuit was to be included anyway, the ability to switch between the original joystick and the Arduino was included using the TMUX1574DYYR 4-bit wide dual direction 2-to-1 multiplexer.

Although not shown in the schematic, a switch was also added to the Arduino to allow the user to toggle the MUX control line from 0 (mechanical mode) to 1 (electrical mode).

## Layout



## Discovered Issues

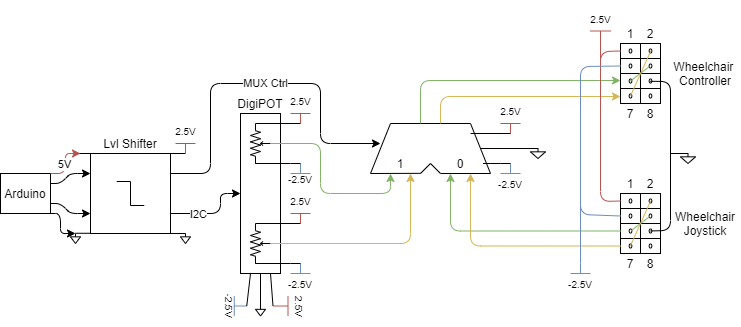
It was observed that the full range of operation for the joystick is 5V peak to peak. However, this is grounded in the center, making the true voltage range from -2.5V to 2.5V. I thought I could get away with IC’s with 5V ranges, and setting ground reference to the -2.5V line.

Unfortunately, since the Arduino can be powered by the chair’s batteries, the Arduino’s ground is already sourced at the chair’s ground, and the joystick controller’s ground is the same as the chair, trying to connect the -2.5V rail to the Arduino’s ground creates a short.

The circuit was re-worked to reference all the same grounds, but the discovery was made that the IC’s cannot handle negative voltages with reference to ground, so there was nothing to be saved from this design.

# Joystick Controller V2

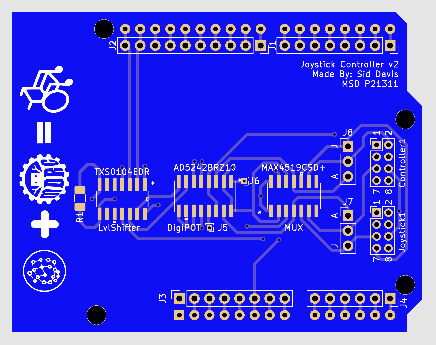
## Schematic



## Theory of Operation

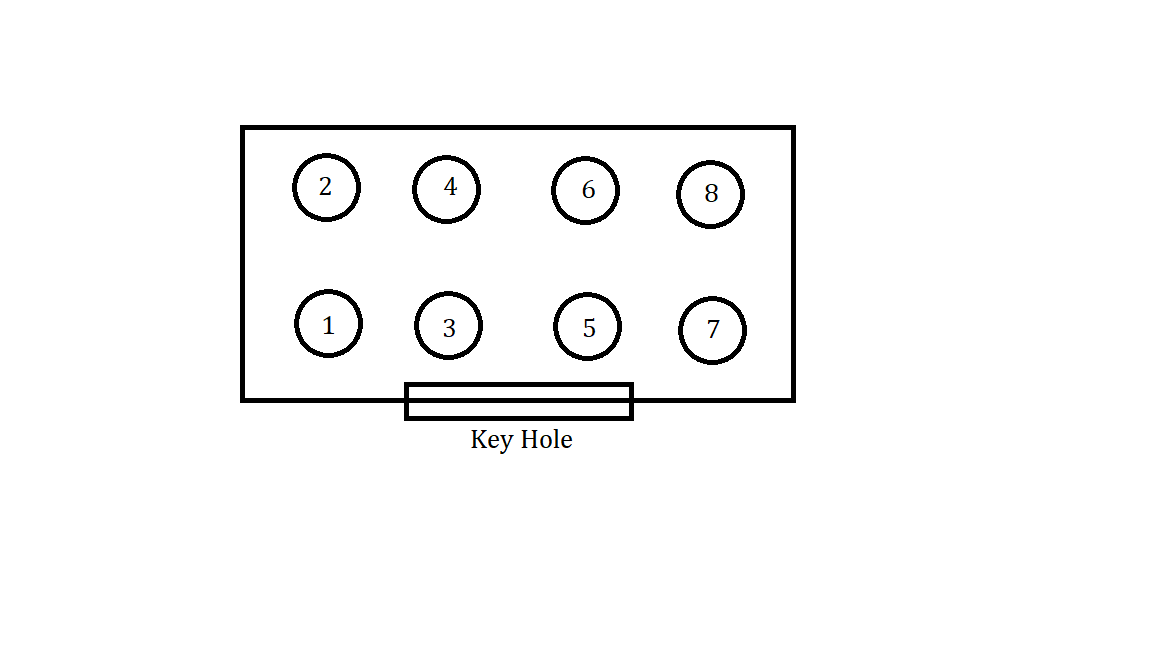
The main difference between V1 and V2 is the selection of IC’s that are capable of handling negative voltages. The rails provided by the wheelchair controller are distributed for use by the IC’s, so each device receives power at the same reference. The MUX was replaced with the MAX4519CSD+, and the DigiPOT was replaced with the AD5242BRZ10. For the Arduino to control these devices, a level shifter TXS0104EDR was also included to handle the difference between the 5V Arduino output and the 2.5V rail.

## Layout



## Connecting to the Wheelchair

The PCB has 2 output headers: one labeled *Controller*, the other labeled *Joystick*. Using an 8-position, 2mm pitch cable about ½ a meter long, connect the *Controller* to the plug found on the PCB within the Wheelchair Joystick Case. There should be digits on the PCB indicating which pin goes where. For reference, it should be similar to this figure:



The same goes for the *Joystick* connection on the Custom PCB, which connects to the original joystick in the Wheelchair Joystick Case. If uninstalling, then the cable goes from the wheelchair’s PCB to the Wheelchair’s joystick, ensuring to line up the pin numbers the same as the custom PCB.

## Discovered Issues

These ones are admittingly embarrassing.

1. I forgot to include pull-up resistors on the I2C lines, which can be included on the bottom of the PCB running from the SDA and SCL pins of the Arduino to the 5V pin, also on the Arduino.
2. The level shifter has a specific direction, with 2.5V on one side and 5V on the other side. I failed to realize this during design, so it is swapped.

## Re-Work

As mentioned before, add 4.7k ohm resistors from the SDA pin on the Arduino to the 5V pin, and from the SCL pin on the Arduino to the 5V pin. This will allow the built-in I2C library to properly communicate with the DigiPOT.

To correct the orientation of the level shifter, the following must be done. Rotate the IC by 180 degrees, and shift it down the pads by 1 pin. Then jumper a connection from the Vcc A pin to the respective N/C pin, and do the same for Vcc B. This connects power from the board to the IC. I got very lucky with the positions of the N/C pins. Then, you need to connect the ground pin of the IC (not on a pad) to the Arduino ground. Lastly, to allow the Arduino to enable the Level Shifter output, connect a voltage divider to the OE pin in the IC (not on a pad), as shown.

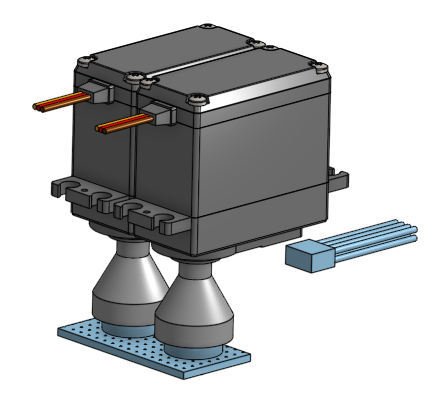
|  |  |  |
| --- | --- | --- |
| Before | After | OE Divider |

## Future Investigations

Even after this rework, which allows the output of the PCB to produce all the necessary voltages and controls, the wheelchair would not accept input from the Digital Potentiometer. To start this investigation, I would purchase a breakout-board that matches the footprint of the DigiPOT to use, then make only the necessary connections between the Arduino, the DigiPOT, and the wheelchair controller to see if a configuration can be created that the wheelchair will accept. Good Luck!

# Joystick Controller V3

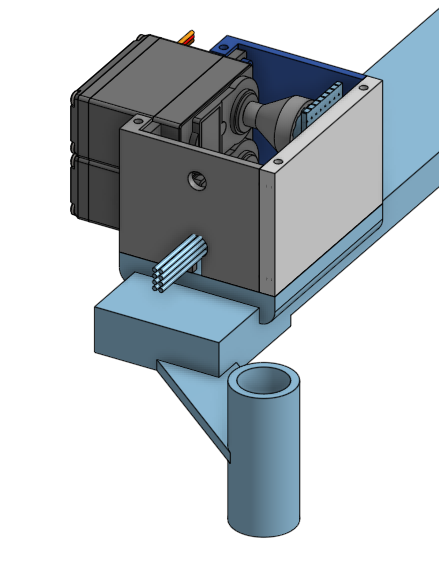
## Schematic



## Theory of Operation

As a last resort, when V2 was producing the correct voltages but the chair refused to accept it as input, I fell back on the physical potentiometers I used to verify the joystick implementation in section 2.3. Connecting these potentiometers to a protoboard with resistors acting as the center-taps mimics the same circuitry as before, so attaching servos to the heads of the potentiometers provides electrical control from the Arduino, making the system go from electrical, to mechanical, back to electrical.

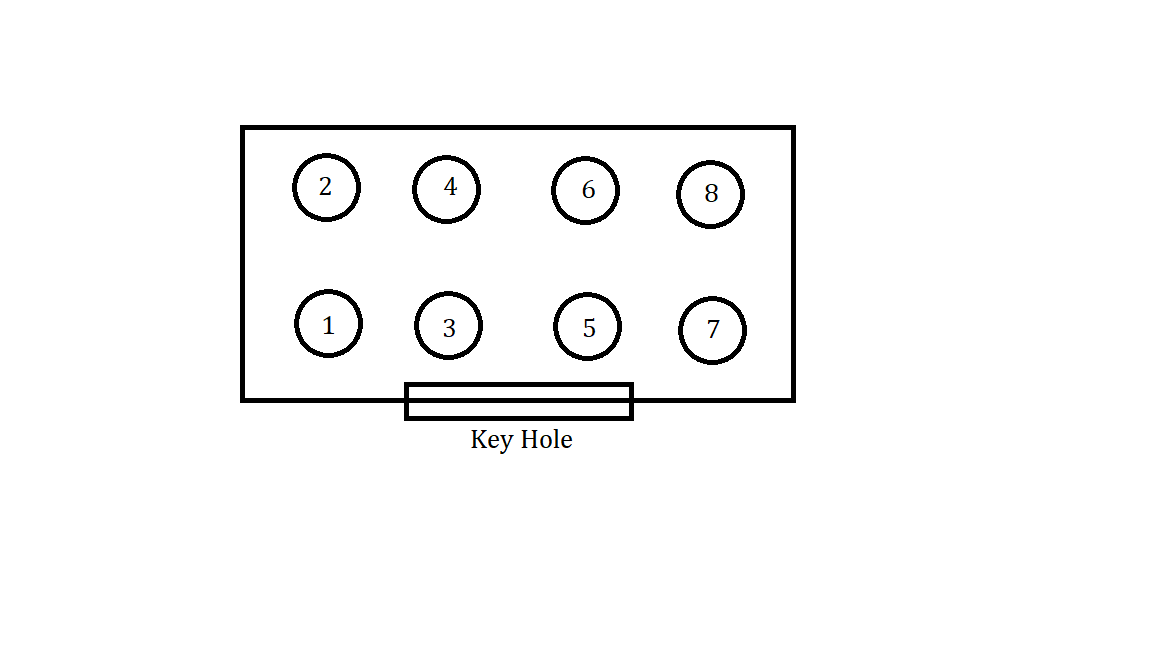
## Layout



The servos, potentiometers, and protoboard are given a case that can screw into the right arm of the wheelchair, midway between the pre-existing Arduino case and the wheelchair joystick.

## Connecting to the Wheelchair

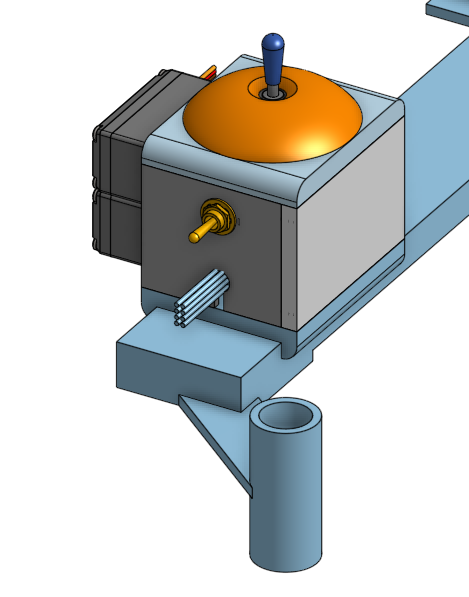
The cable that comes out of the case only connects to the wheelchair’s PCB. No connection to the wheelchair’s joystick is made during this design. The cable is hard-wired to the internal protoboard, so do not pull on it too hard. The orange wire is pin 1, the 2.5V rail, so use this to connect to the Wheelchair’s PCB.



## Discovered Issues

The servos do not always return exactly to the center position, depending on whether it is rotating clockwise towards the center vs rotating counterclockwise towards the center. This causes an error on boot for the wheelchair if not perfectly centered, but a quick fix is to set the potentiometers off center in the Arduino Code on boot, return to center, and then try powering on the wheelchair. If this doesn’t work, set the potentiometers to rotate in the opposite directions on boot, then return to center.

## Future Investigations



A joystick was added to the top of the case since this design has no multiplexer between the original joystick and the potentiometers. This would allow the user to give commands to the Arduino (if the switch on front is in the up position) using this joystick, so the Arduino can relay the command to the servos. If the switch is in the down position, then the Arduino ignores the joystick and only uses commands sent serially.

Furthermore, to fix the non-return-to-zero issue between the servos and the potentiometers, a feedback loop could be created by feeding the potentiometer value back to the Arduino, so if it is expecting a value equivalent to zero, the servo turns to whatever value is necessary to ensure the potentiometer produces that zero.

# Appendix

## Joystick Controller V0 Arduino Code

// Testing DAC from PWM conversions using a RC LP filter

/\*

\* Uno PWM pins:

\* 490 Hz on Pin {3, 9, 10, 11}

\* 980 Hz on Pin {5, 6}

\*

\* Leonardo PWM pins:

\* 490 Hz on Pin {5, 6, 9, 10, 13}

\* 980 Hz on Pin {3, 11}

\*/

#define pwm\_pin\_490 10

#define pwm\_pin\_980 11

int pwm\_val = 0;

#define pot\_pin 0

int pot\_val = 0;

void setup() {

// set PWM freq on UNO pins 3&11 to 31372.55Hz

TCCR2B = TCCR2B & B11111000 | B00000001;

// put your setup code here, to run once:

pinMode(pwm\_pin\_490, OUTPUT);

pinMode(pwm\_pin\_980, OUTPUT);

analogWrite(pwm\_pin\_490, 0);

analogWrite(pwm\_pin\_980, 0);

// while(!Serial);

// Serial.begin(9600);

// Serial.println("Hello Mr. MSD");

}

void loop() {

// put your main code here, to run repeatedly:

pot\_val = analogRead(pot\_pin);

// Serial.print("\npot val: ");

// Serial.println(pot\_val);

pwm\_val = map(pot\_val, 0, 1023, 0, 255);

// Serial.print("PWM val: ");

// Serial.println(pwm\_val);

// analogWrite(pwm\_pin\_490, pwm\_val);

analogWrite(pwm\_pin\_980, pwm\_val);

delay(100);

}

## Joystick Controller V1 Arduino Code

// Communicate with DigiPOT over SPI

#define DEBUG 0

#define CLK\_PIN 13

#define MISO\_PIN 12

#define MOSI\_PIN 11

#define SS\_PIN 10

#define SHDN\_PIN 9

#define EN\_MUX\_PIN 4

#define S\_MUX\_PIN 3

const int FREQ = 20; //kHz for SPI

float T = 1/FREQ \* 1000.0;

const int QUART\_DEL = round(T/4);

const int MECHANICAL = 0;

const int ELECTRICAL = 1;

int mux\_state = 0;

const int XPOT = 0;

const int YPOT = 1;

int x\_pos = 127;

int y\_pos = 127;

void digitalPotWrite(int address, int value);

void SPI\_transfer(int value, int bits);

void setup() {

while(!Serial);

Serial.begin(9600);

if (DEBUG) Serial.println("Joystick Controller in DEBUG mode. Feedback Enabled");

// put your setup code here, to run once:

pinMode(S\_MUX\_PIN, OUTPUT);

pinMode(EN\_MUX\_PIN, OUTPUT);

pinMode(CLK\_PIN, OUTPUT);

pinMode(MISO\_PIN, INPUT);

pinMode(MOSI\_PIN, OUTPUT);

pinMode(SS\_PIN, OUTPUT);

pinMode(SHDN\_PIN, OUTPUT);

// Mux

digitalWrite(EN\_MUX\_PIN, LOW); // enable mux

digitalWrite(S\_MUX\_PIN, mux\_state);

// DigiPOT

digitalWrite(SHDN\_PIN, HIGH);

digitalPotWrite(XPOT, x\_pos);

digitalPotWrite(YPOT, y\_pos);

}

void loop() {

if (Serial.available()) {

// parse command

char cmd = Serial.read();

switch(cmd) {

case 'X':

case 'x':

setX();

Serial.print('0');

break;

case 'Y':

case 'y':

setY();

Serial.print('0');

break;

case 'M':

case 'm':

toggleMux();

Serial.print('1');

break;

case '\n':

case '\r':

case '\t':

case ' ':

case ',':

break;

default:

Serial.print('1');

}

}

}

void setX() {

// get Y value

int input = Serial.parseInt();

if (DEBUG) {

Serial.print("Value received: ");

Serial.println(input);

}

// force within bounds

input = min(max(input, -100), 100);

if (DEBUG) {

Serial.print("After Bounding: ");

Serial.println(input);

}

// map [-100, 100] to [0, 255]

x\_pos = map(input, -100, 100, 0, 255);

if (DEBUG) {

Serial.print("Mapped Position: ");

Serial.println(x\_pos);

}

// commit the value

digitalPotWrite(XPOT, x\_pos);

if (DEBUG) {

Serial.print("The X rail has been set to position ");

Serial.println(x\_pos);

}

}

void setY() {

// get Y value

int input = Serial.parseInt();

if (DEBUG) {

Serial.print("Value received: ");

Serial.println(input);

}

// force within bounds

input = min(max(input, -100), 100);

if (DEBUG) {

Serial.print("After Bounding: ");

Serial.println(input);

}

// map [-100, 100] to [0, 255]

y\_pos = map(input, -100, 100, 0, 255);

if (DEBUG) {

Serial.print("Mapped Position: ");

Serial.println(y\_pos);

}

// commit the value

digitalPotWrite(YPOT, y\_pos);

if (DEBUG) {

Serial.print("The Y rail has been set to position ");

Serial.println(y\_pos);

}

}

void toggleMux() {

if (mux\_state == ELECTRICAL) {

mux\_state = MECHANICAL;

digitalWrite(S\_MUX\_PIN, MECHANICAL);

if (DEBUG) Serial.println("Mux Switch set to MECHANICAL");

} else {

mux\_state = ELECTRICAL;

digitalWrite(S\_MUX\_PIN, ELECTRICAL);

if (DEBUG) Serial.println("Mux Switch set to ELECTRICAL");

}

}

void digitalPotWrite(int address, int value) {

digitalWrite(SS\_PIN, LOW);

delay(1);

SPI\_transfer(address, 2);

SPI\_transfer(value, 8);

delay(1);

digitalWrite(SS\_PIN, HIGH);

}

void SPI\_transfer(int value, int bits) {

for (int i = bits - 1; i >= 0; i--) {

bool tx\_bit = bitRead(value, i);

digitalWrite(MOSI\_PIN, tx\_bit);

delayMicroseconds(QUART\_DEL);

digitalWrite(CLK\_PIN, HIGH);

delayMicroseconds(2 \* QUART\_DEL);

digitalWrite(CLK\_PIN, LOW);

delayMicroseconds(QUART\_DEL);

}

}

## Joystick Controller V2 Arduino Code

// Code created by Sid Davis for MSD P21311

#define DEBUG 1

// pins & addresses

#define MUX 12

#define LVL\_EN 11

#define DIGIPOT 0b00101100

#define X\_POT 0x00

#define Y\_POT 0x80

// libraries

#include <Wire.h>

// consts

#define ELECTRICAL 0

#define MECHANICAL 1

#define RAMP\_DELAY 10

#define MIN\_SET 50

#define MAX\_SET 205

// state variables

bool mux\_state = MECHANICAL;

bool lvl\_enabled = false;

byte x\_pos = 128;

byte y\_pos = 128;

byte x\_goal = 128;

byte y\_goal = 128;

void setup() {

// put your setup code here, to run once:

pinMode(MUX, OUTPUT);

pinMode(LVL\_EN, OUTPUT);

digitalWrite(MUX, mux\_state);

digitalWrite(LVL\_EN, lvl\_enabled);

// prep comms with DigiPOT

Wire.begin();

// Wait for the Serial Connection

while(!Serial);

Serial.begin(9600);

if (DEBUG) Serial.println("Joystick Controller in DEBUG mode. Feedback Enabled");

// Arduino is properly powered, enable the level shifter

toggleLvlOut();

}

void loop() {

// put your main code here, to run repeatedly:

if (Serial.available()) {

// parse command

char cmd = Serial.read();

switch(cmd) {

case 'X':

case 'x':

Serial.print(setX());

break;

case 'Y':

case 'y':

Serial.print(setY());

break;

case 'M':

case 'm':

Serial.print(toggleMux());

break;

case 'O':

case 'o':

//Serial.print(toggleLvlOut());

Serial.print(1);

break;

case '\n':

case '\r':

case '\t':

case ' ':

case ',':

break;

default:

Serial.print(1);

}

}

if (x\_goal > x\_pos) {

x\_pos++;

digitalPotWrite(X\_POT, x\_pos);

delay(RAMP\_DELAY);

} else if (x\_goal < x\_pos) {

x\_pos--;

digitalPotWrite(X\_POT, x\_pos);

delay(RAMP\_DELAY);

}

if (y\_goal > y\_pos) {

y\_pos++;

digitalPotWrite(Y\_POT, y\_pos);

delay(RAMP\_DELAY);

} else if (y\_goal < y\_pos) {

y\_pos--;

digitalPotWrite(Y\_POT, y\_pos);

delay(RAMP\_DELAY);

}

}

void digitalPotWrite(int address, byte value) {

Wire.beginTransmission(DIGIPOT);

Wire.write(byte(address));

Wire.write(value);

Wire.endTransmission();

}

int setX() {

int error = 0;

// grab the value to set

int input = Serial.parseInt();

// if (!length(input)) {

// if (DEBUG) Serial.println("No integer value found for 'SET X' cmd");

// error = 1;

// return error;

// }

// check for bounding

if ((input > 100) || (input < -100)) {

if (DEBUG) Serial.println("Received value for 'SET X' is out of range, but has been clipped");

error = 1;

}

input = min(max(input, -100), 100);

// map to a byte and commit

x\_goal = byte(map(input, -100, 100, MIN\_SET, MAX\_SET));

if (DEBUG) {

Serial.print("X has been set to ");

Serial.println(x\_goal);

}

return error;

}

int setY() {

int error = 0;

// grab the value to set

int input = Serial.parseInt();

// if (length(input) == 0) {

// if (DEBUG) Serial.println("No integer value found for 'SET Y' cmd");

// error = 1;

// return error;

// }

// check for bounding

if ((input > 100) || (input < -100)) {

if (DEBUG) Serial.println("Received value for 'SET Y' is out of range, but has been clipped");

error = 1;

}

input = min(max(input, -100), 100);

// map to a byte and commit

y\_goal = byte(map(input, -100, 100, MIN\_SET, MAX\_SET));

if (DEBUG) {

Serial.print("Y has been set to ");

Serial.println(y\_goal);

}return error;

}

int toggleMux() {

if (mux\_state == ELECTRICAL) {

mux\_state = MECHANICAL;

digitalWrite(MUX, mux\_state);

if (DEBUG) Serial.println("Mux set to MECHANICAL");

} else {

mux\_state = ELECTRICAL;

digitalWrite(MUX, mux\_state);

if (DEBUG) Serial.println("Mux set to ELECTRICAL");

}

return 0;

}

int toggleLvlOut() {

if (lvl\_enabled == 1) {

lvl\_enabled = 0;

digitalWrite(LVL\_EN, lvl\_enabled);

if (DEBUG) Serial.println("Lvl Shft set OFF");

} else {

lvl\_enabled = 1;

digitalWrite(LVL\_EN, lvl\_enabled);

if (DEBUG) Serial.println("Lvl Shft set ON");

}

return 0;

}

## Joystick Controller V3 Arduino Code

// control joystick using servos & potentiometers

#define DEBUG 0

// Libraries

#include <Servo.h>

// pins

#define SERVO\_X 3

#define SERVO\_Y 5

// consts

#define ELECTRICAL 0

#define MECHANICAL 1

#define RAMP\_DELAY 10

#define MIN\_SET 0

#define MAX\_SET 180

// state variables

bool ctrl\_state = MECHANICAL;

byte x\_goal = 90;

byte y\_goal = 90;

byte x\_pos = 90;

byte y\_pos = 90;

// servo objects

Servo servo\_x;

Servo servo\_y;

void setup() {

// put your setup code here, to run once:

// initialize the servos

servo\_x.attach(SERVO\_X);

servo\_y.attach(SERVO\_Y);

servo\_x.write(x\_pos);

servo\_y.write(y\_pos);

// Wait for the Serial Connection

while(!Serial);

Serial.begin(9600);

if (DEBUG) Serial.println("Joystick Controller in DEBUG mode. Feedback Enabled");

}

void loop() {

// put your main code here, to run repeatedly:

if (Serial.available()) {

// parse command

char cmd = Serial.read();

switch(cmd) {

case 'X':

case 'x':

Serial.print(setX());

break;

case 'Y':

case 'y':

Serial.print(setY());

break;

case 'M':

case 'm':

// Serial.print(toggleMux());

Serial.print(1);

break;

case 'O':

case 'o':

//Serial.print(toggleLvlOut());

Serial.print(1);

break;

case '\n':

case '\r':

case '\t':

case ' ':

case ',':

break;

default:

Serial.print(1);

}

}

if (x\_goal > x\_pos) {

x\_pos++;

servo\_x.write(x\_pos);

delay(RAMP\_DELAY);

} else if (x\_goal < x\_pos) {

x\_pos--;

servo\_x.write(x\_pos);

delay(RAMP\_DELAY);

}

if (y\_goal > y\_pos) {

y\_pos++;

servo\_y.write(y\_pos);

delay(RAMP\_DELAY);

} else if (y\_goal < y\_pos) {

y\_pos--;

servo\_y.write(y\_pos);

delay(RAMP\_DELAY);

}

}

int setX() {

int error = 0;

// grab the value to set

int input = Serial.parseInt();

// check for bounding

if ((input > 100) || (input < -100)) {

if (DEBUG) Serial.println("Received value for 'SET X' is out of range, but has been clipped");

error = 1;

}

input = min(max(input, -100), 100);

// map to a byte and commit

x\_goal = byte(map(input, -100, 100, MIN\_SET, MAX\_SET));

if (DEBUG) {

Serial.print("X has been set to ");

Serial.println(x\_goal);

}

return error;

}

int setY() {

int error = 0;

// grab the value to set

int input = Serial.parseInt();

// check for bounding

if ((input > 100) || (input < -100)) {

if (DEBUG) Serial.println("Received value for 'SET Y' is out of range, but has been clipped");

error = 1;

}

input = min(max(input, -100), 100);

// map to a byte and commit

y\_goal = byte(map(input, -100, 100, MIN\_SET, MAX\_SET));

if (DEBUG) {

Serial.print("Y has been set to ");

Serial.println(y\_goal);

}

return error;

}