Project goals.

The goal of this project is to control the skid steer car I built in my previous project with a Raspberry Pi over the internet. As of now my plan is to use SSH to run a program on the Pi that takes constant user input keystrokes to control the PWM output of the GPIO pin. These pins will drive the MOSFETs via some isolating circuit, and the Raspberry Pi will operate on its own battery supply. For a stretch goal I would like to set up a server on the Pi that takes live video feed from a Pi camera.

Grading for this project.

Each Friday there is a set of deliverables set out. If these are not completed by that day, it is 5% off my total grade per day late.

If the project is not fully functioning by the end of the 6 weeks, the project grade is capped at 50% unless it is due to an accidental component failure, at which point it is 5% off per class day.

At the end of the project, the end of lab day 14, the project needs to pass all conformance and functional tests (**outlined in the next section**) as well as all construction standards (**outlined in construction standards section**). A test point challenge will be done on lab day 15 where I will outline specific test points and their expected value, and the instructor will probe my knowledge of the circuit and outlined test points. If any of these standards are not passed, it is -5% per rework.

Completion of the project on time gets me 75%. The additional 25% is broken up into 5 extra deliverables worth 5% each.

- Final documentation completed on a computer.
- Abstract explaining how it works in non-technical terms.
- Bill of materials completed with part numbers, manufacturer, and current list prices for all components.
- User’s guide explaining to a non-expert how to operate the project.
- Impeccable craftsmanship, compared to that of a professional assembly.

Test Criteria

CONFORMANCE - If connection is severed between robot and PC, the robot stops. RP power supplied by separate battery, not the same battery that powers the motors.

FUNCTIONAL - Must be able to control robot speed and direction via commands entered on a PC. **STRETCH:** video feed back to PC from camera mounted on robot.

Deliverables

Due 10/3 (All-Lab Day 07) -- grading/deadline dates, Construction Standards, conformance/functional criteria, prototype design, define deliverables, order parts

Due 10/10 (All-Lab Day 08) -- experiment with writing test programs that scan keystrokes continuously in a loop w/o needing Return keystrokes ; journal entry

Due 10/17 (All-Lab Day 09) -- research GPIO control on RP, experiment with writing test program for GPIO output ; journal entry

Due 10/24 (All-Lab Day 10) -- experiment with PWM control of GPIO pins on RP ; journal entry

Due 10/31 (All-Lab Day 11) -- work day ; journal entry

Due 11/7 (All-Lab Day 12) -- work day ; journal entry

Due 11/14 (All-Lab Day 13) -- work day ; journal entry

Due 11/21 (All-Lab Day 14) -- construction completed ; journal entry

Due 11/21 (All-Lab Day 14) -- Construction Standards inspection passed (-??% for retries)

Due 11/21 (All-Lab Day 14) -- Conformance tests passed (-??% for retries)

Due 11/21 (All-Lab Day 14) -- Functional tests passed (-??% for retries)

Due 11/28 (All-Lab Day 15) -- Submit Project Journal (includes schematic, code, test data, daily work entries, etc.)

Due 11/28 (All-Lab Day 15) -- Test-point/code challenge (-??% for retries)

Due 11/28 (All-Lab Day 15) -- additional = e-docs (+?%), abstract (+?%), BOM (+?%), UserGuide (+?%), impeccable (+?%)

construction standards

Construction Standards

General layout

- All components shall be mounted securely where applicable.
- All electrical components shall be located to avoid accidental exposure to liquids.
- All manual controls (e.g. buttons, handles, knobs) shall be accessible and function without undue effort.
- Fragile components (e.g. heat-sensitive semiconductors) shall be easily accessed for replacement.

Fastening

- All threaded fasteners shall be properly engaged and tightened.
- A minimum of 1-1/2 threads shall extend beyond the threaded hardware (e.g., nut), unless specified otherwise.

- All cable ties shall be trimmed off, flush with the back end of the strap head. (NASA-STD-8739.4, NFPA 79 2007 edition (13.1.5.6))

Thermal considerations

- Power-handling components shall have adequate cooling capacity, usually in the form of a heat sink.
- All components expected to run hot shall be located in such a way that their heat does not affect the function or longevity of any other components.
- Components dissipating heat in quantities of 1 Watt or greater, or in quantities sufficient to damage a PCBshall be mounted with sufficient standoff [> 1.5 mm (0.060 in)] and shall be mechanically restrained.

Power wiring

- All electrical sources capable of generating currents exceeding conductor ampacity ratings shall be overcurrent-protected, regardless of voltage.
- Overcurrent protection shall be on the ungrounded (“hot”) conductor(s) only (NFPA 70 2017 edition (240.15(A))). No grounded conductor shall be overcurrent-protected or switched (NFPA 70 2017 edition (240.22)).
- All power conductors shall be strain-relieved so that tension applied to them will not stress the electrical connections themselves (NFPA 79 2007 edition (13.4.3.1.1)). Permanent conductors not in a raceway shall be securely fastened at least every 6 inches using cable ties or other appropriate means.
- All conductors shall be prevented from chafing against any sharp edges (NFPA 79 2007 edition (13.5.1.2)), and this includes installing bushings on all electrical fittings.
- All power conductors shall be properly colored according to American wiring conventions (e.g. red and black for DC + and −, black and white for AC “hot” and “neutral”, green for earth ground) (NFPA 79 2007 edition (13.2)). Colored tape is permissible at both ends of a wire whose insulation is not of the correct color.

Other wiring and connections *All other conductors shall have sufficient ampacity and insulation voltage ratings for their application (NFPA 79 2007 edition (12.5)).

- Compression terminals– crimping of solid wire, component leads, or stranded wire that has been solder tinned, is prohibited. The conductor shall extend a minimum of flush with, and a maximum of one (1) wire diameter beyond the conductor crimp edge. No protruding wire strands outside the terminal barrel. (NASA-STD-8739.4)
- Compression-style wire splices are prohibited between terminals (NFPA 79 2007 edition (13.1.2.1)).
- Solid wire is prohibited where wire motion is possible, to prevent metal fatigue from vibration and other mechanical stresses. Only stranded wire shall be used for unrestrained wire runs, wires running between components not rigidly mounted to each other or to the same rigid frame, etc. (NFPA 79 2007 edition (12.2.4))
- Only solid wires shall be wrapped around screw terminals, and this direction shall be clockwise. Wrapdistance shall be between 180o and 270o (between 12 and 34 turn).

- Attached wires shall withstand being lightly pulled with fingers.
- All wire insulation shall be intact (i.e. no bare wires anywhere).
- After insulation removal, the remaining conductor insulation shall not exhibit any damage such as nicks, cuts, or charring. Conductors with damaged insulation shall not be used. Scuffing from mechanical stripping or slight discoloration from thermal stripping is acceptable. (NASA-STD-8739.3)
- All exposed wiring shall be generally neat and not messy in appearance.
- Multiple conductors extending beyond an enclosure or panel shall be bundled together as a multi conductor cable wherever possible, unless separation is necessary to avoid undesired signal coupling between conductors.
- Solderless breadboards are prohibited for permanent assemblies; acceptable only for prototyping.
- Tape is prohibited as electrical insulation; heat-shrink tubing shall be used instead.
- Cables shall not be bent below the minimum recommended inside bend radius (6 diameters for flexible coaxial cable, 3 diameters for multi-wire harnesses 10 AWG and smaller). (NASA-STD-8739.4).

Circuit design • Decoupling capacitors– every integrated circuit should have a decoupling capacitor of at least 0.1 μF connected closely to its DC power terminals, for the purpose of stabilizing DC voltage at each IC. The same is true for loads generating noise, such as DC electric motors.

• Separate power sources– when possible, separate power sources should be used to energize high-current loads versus low-current control circuitry, in order to avoid placing noise caused by the high-current loads onto the control circuit's power rails.

• Star-point power distribution– circuits that might interfere with one another through a shared power source should have separate power conductors routing to that source.

• Keep high-current loops small– conductors carrying high currents should run closely parallel to each other in order to minimize the "loop area" and thus minimize magnetic field creation.

• Twisted-pair, shielded cable– sensitive signals should be protected from crosstalk by electrostatic shielding (grounded at one end only) and by twisting conductors (to minimize magnetic coupling).

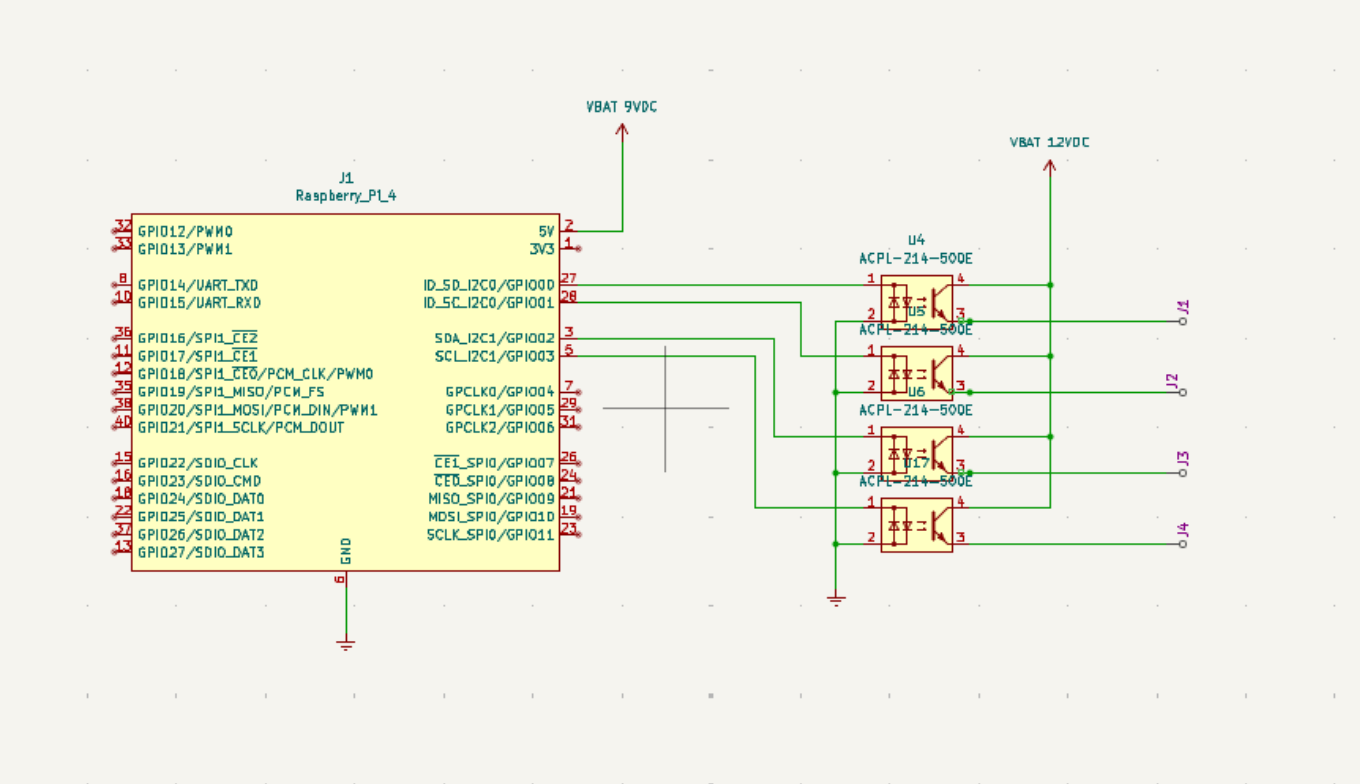
• Conductor fault tolerance– circuits linked by plug-in conductors should be designed to assume safe states if disconnection occurs.

• Consistent control directions– clockwise knob rotation and upwards switch/slider should result in "more" or "on" actions.

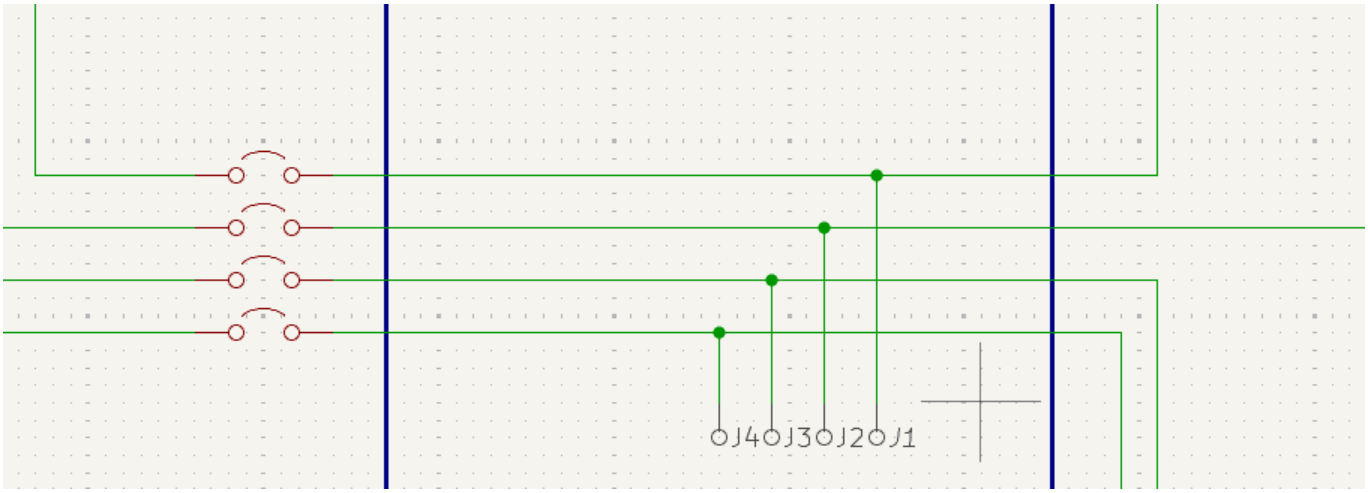
Pototype desgin.

The code for the project will consist of some sort of Python or C coding that constantly checks for user input without having to press Enter. Right now I'm thinking some sort of user input numbers for speed and then the arrow keys adjust direction proportionally to the speed. These inputs will adjust the outputs of the GPIO pins on the Raspberry Pi, which send signals to opto-isolators to drive the MOSFETs of my H-bridge.

Isolation circuitry.



These points labled J1-J4 will connect to my signal condition circuit that has the time delay and inverters on it.



I'm not sure weather or not I need pulldowns on this or not I will have to test to find out.

Parts

Raspberry Pi - already in stock.

Pi camera V3 - Ordering today so that if time premitts I can work on FPV.

all lab day 8

Started off by connecting the pi to the internet, updating then enabling SSH. Then SSHed into the pi via my wsl on my labtop (ssh pi@ipaddress). After this I created a python text file (nano hello.py) wrote a simple hello world program and ran it to make sense of how to write and run python code through the terminal. Ounce this was completed successfully I prompted AI to write my a python code that scans for user input keys constantly without the need to prompt the user or the need for an enter key. Now I am trying to learn so I did reasearch the code that was written for and tried to undstand each function but I did rely on AI pretty heavily in writting this code.

```
import sys #python library that allows for me to interact with the system
import termios #this python library controls terminal settings like turning off
line buffering
import tty # puts the terminal into raw mode (instantously looks for inputs.

speedright = 0
speedleft = 0
x = 0
def get_key(): # function that looks at a single user input key

        fd = sys.stdin.fileno() # used to change the settings of the terminal
        old_settings = termios.tcgetattr(fd) #saves the terminals current setting
so we can revert back to them

        try: # try to run loop and if broken jump to finally
                tty.setraw(fd) # turns the terminal to raw mode "doesn't need the
user to press enter after each key press."
                ch = sys.stdin.read(1) # reads one character from stdin
        finally: # Runs when the while loop is borke
                termios.tcsetattr(fd, termios.TCSADRAIN, old_settings) # takes
terminal back out of raw mode

        return ch

print("Press keys (press 'q' to quit):")

while True:

        key = get_key() #runs the get key function defined above

        if key.isdigit(): # checks if key is a number
                num = int(key) # if key is a number convert it to an interger
                x = num * 10 # converts to a "percentage" for speed command used
to set pmw speed
        elif key == "w": # forward key
                speedleft = x # sets both left and right speeds to the number
above
                speedright = x
                print(f"speed left = {speedright}%, speed right = {speedleft}%") #
prints speed on user screen
        elif key == "s": # reverse key
                speedleft = x
                speedright = x
                print(f"speed left = {speedright}%, speed right = {speedleft}%")
```

```
        elif key == "a": #left key
                speedleft = 0 # sets left motor to 0% speed
                speedright = 90 # sets right motor to 90% speed
                print(f"speed left = {speedright}%, speed right = {speedleft}%")
        elif key == "d": # right key
                speedleft = 90
                speedright = 0
                print(f"speed left = {speedright}%, speed right = {speedleft}%")
        elif key =="q": # breaks out of loop
                print("exit")
                break
```

Code explaintion

- **sys libary** allows for the interaction of the system like scanning keyboard input.
- **termios libary** allows for the control of terminal settings in my case turning off line buffering (no need for enter key).
- **tty libary** allows for you to change the terminal modes. Raw mode is used in this code which is part of what is used to disable the enter key press after a user input
- **get key function** looks at a single key press and stores it, reads the old terminal settings and saves them, changes the terminal mode to "raw", then tries to read one character from stdin and if this isn't possible for some reason it will return the terminal back to it's old settings.
- **fd = sys.stdin.fileno()** This function is a bit confusing, but I think I've got the gist. Python can't directly control the terminal at a low level because it's a high-level language. Libraries like termios and tty act as bridges to interact with the lower level operating system. sys.stdin is Python's high-level input stream that represents keyboard input. When this function wants to read the keyboard input it needs to know what device to read from fd = sys.stdin.fileno() tells python go look for this device in this case it is a number given to the keyboard by the operating system. Functions like termios and tty then use that file descriptor to modify the terminal's settings.
- **rest of code**

This is alot easier to understand. Key is the variable representing the input ascii character. Even numbers are represented as characters hints why I need to convert check if the input was a number and convert it to an intger.

Keys a,s,d, and w will be used for direction control in the future.

User presses a number 1-9 which changes the "speed". When w is pressed (forward) both sides equal that speed same with reverse of course the robot will move in differnent direction. When a or d is press I set the speed to max for one side and 0 for the other.

When the letter q is pressed a it prints exit, breaks out of the loop runs the "finally" portion of the getkey function ie. reverts back to orginal terminal settings and exits the program.

code output.

This is the output following this user input. 2, w, a, d, s, 30, w, a

```
pi@raspberrypi:~ $ python3 hello.py
Press keys (press 'q' to quit):
speed left = 20%, speed right = 20%
speed left = 90%, speed right = 0%
speed left = 0%, speed right = 90%
speed left = 20%, speed right = 20%
speed left = 30%, speed right = 30%
speed left = 90%, speed right = 0%
```

future ideas/improvements

Right now the code scans one key stroke at a time which only allows for numerals 1-9 to be valid speed setting. My code then converts the singal digit to a 0 - 90 scale with out the possibility for 100. So in my mind I have two options. Option 1 plus and minus buttons increase and or decrease speed pros possibility for finer control of speed cons: will take longer to to make big jumps in speed and in my mind is a little more clusmy. Option 2 9 speed control setting 1-9 and scale this to the proper PMW speed so each step is 11 percent pros: more user freindly and is quick cons: less fine control. I will have to play with it ounce I have the robot controlled. I do know that both approaches will take some scaling as I don't think my robot will actually move forward until you hit like a 30% duty cycle.

I need to set up either a SSH username or static IP so that you do not have to find the dynamic ip everytime you want to SSH into it. However right now I am working on this both at home and at school so setting a static IP isn't an option right now.

I can think of two option for steering the robot as well. Option one set one side to 0% speed and other to 100% every time the turn key is pressed. You can still steer fairly accurate and ensures that you don't stall the motor due to friction. Small changes in direction would just have to be short pulses of the key. Option 2 turn rate based on speed setting. For instance if speed is set to 50% turn one side to 50% and the other to 0%. Or some other form of scaling. For this one to work well I do think slicker tires would be needed as the robot already is super grippy and doesn't like to steer well on carpet. I can think of taping the tires or putting some sort of plastic on the outsides or 3d print some. Option 3 let the user choose if steering based on speed changes works well enough I could prompt the user to select what control mode they would like to be in.

PMW signaling

Next I added PMW signaling to the code. https://randomnerdtutorials.com/raspberry-pi-pwm-python/ I used code outlined here and edited it to suit my needs.

```
from gpiozero import PWMLED # GPIO libary and PMW libary
from time import sleep # Sleep is a time delay

mrightforward = PWMLED(18) #Right set of motors PWM signal to move forward GPIO 18
mleftforward = PWMLED(19) #Left set of motors PWM singal to move backwards GPIO 19
mrightreverse = PWMLED(12) #Right set of motors PWM siganl to move forward GPIO 12
mleftreverse = PWMLED(13) # Left set of motors PWM signal to move backwards GPIO
```

```
13
```

First I imported the needed libraries and declared varaibles associated to the desired pins.

```
        elif key == "w": # forward key
                mrightforward.value = speed # sets the duty cycle to user input
speed
                mleftforward.value =  speed # sets the duty cycle to user input
speed
                mrightreverse.value = 0
                mleftreverse.value = 0
                print(f"speed left = {speed}, speed right = {speed}") # prints
speed on user screen
        elif key == "s": # reverse key
                mrightreverse.value = speed
                mleftreverse.value = speed
                mrightforward.value = 0
                mleftforward.value = 0
                print(f"speed left = {speed}, speed right = {speed}")
        elif key == "a": #left key
                mleftreverse.value = speed # sets left set of wheels reverse
signal to user input speed and right forward to user input speed
                                        #which will cuase the robot to turn
left
                mrightforward.value = speed
                print(f"speed left = {speed}, speed right = {speed}")
        elif key == "d": # right key
                mrightreverse.value = speed
                mleftforward.value = speed
                print(f"speed left = {speed}, speed right = {speed}%")
        elif key =="q": # breaks out of loop
                mrightforward.value = 0
                mleftforward.value = 0
                mrightreverse.value = 0
                mleftreverse.value = 0
                print("exit")
                break
```

I then set those values to the user input speed as shown. When forward button is pressed both the left and right forward PMW signals go at that the user input duty cycle.

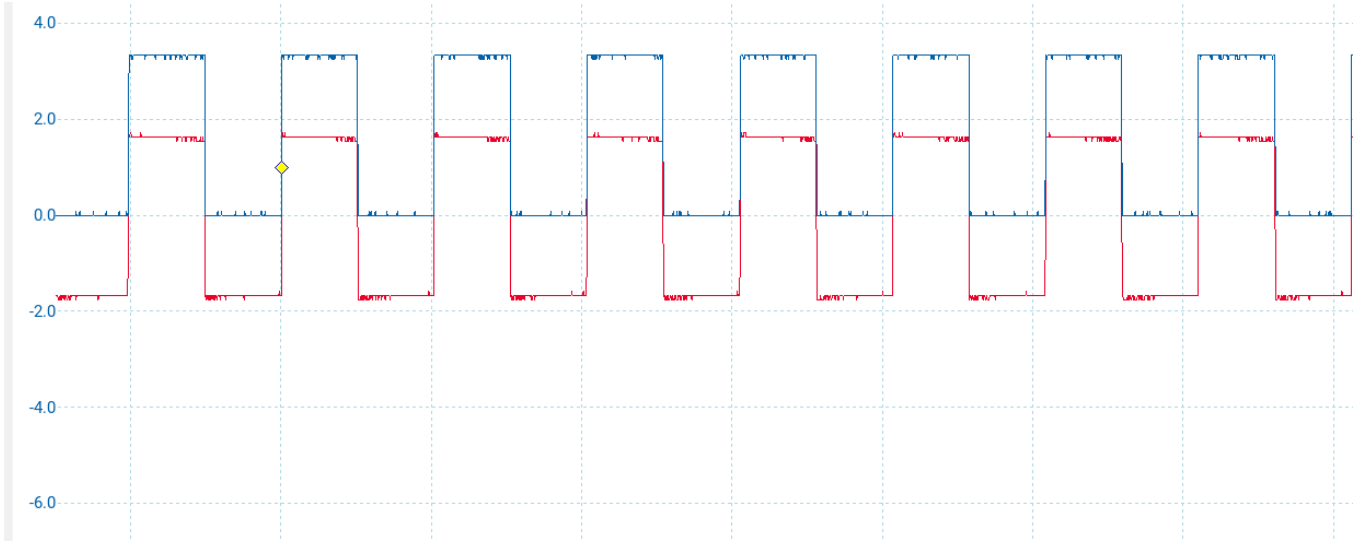
For turning I set the apporiate side to reverse at the user input speed and the other side to forward at the user input speed.

Note - I have not tried controlling the robot like this yet but I do stil assume for the lower rates of speed the motors will stall out when trying to turn. Also how this will control now the user presses a button and it does that action until another button is pressed. ie. breifly press forward button the robot will drive forward indefiently unless another button is pressed or the user hits 0 then another button at which point the robot will stop. I experimented breifly with time delays to make it stop moving it said direction after a blank amount

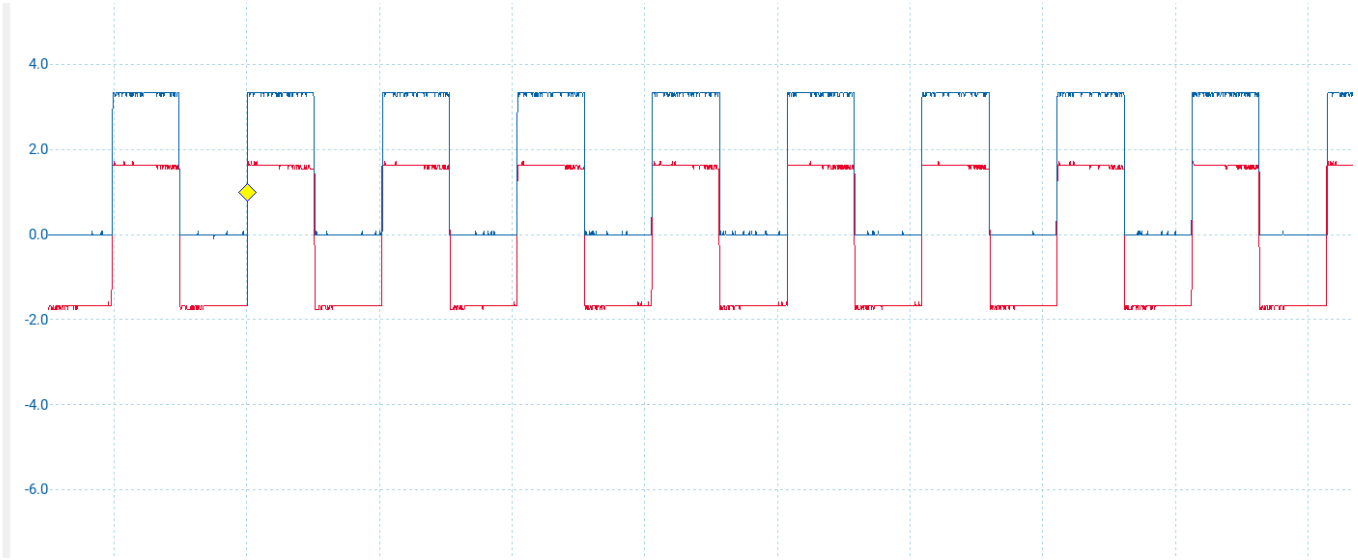
of time like if you press and release the forward button it would move forward for 1 second then stop and move forward for as long as it was pressed but the time delay seemed to accumulate so if you pressed and released the forward button for 10 seconds another 10 seconds would pass before it stopped moving in that direction.

examples of code.

50% duty cycle with the forward key pressed. red = left forward blue = right forward.

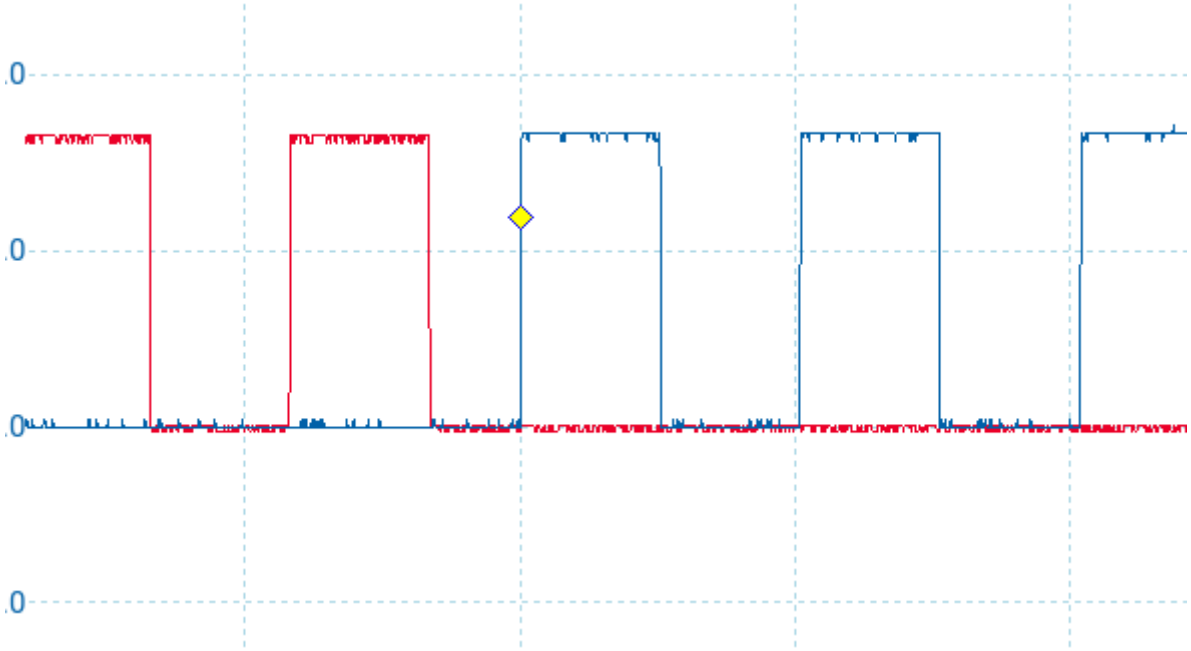


50% duty cycle with the left key pressed red = left motor reverse blue = right motor forward.



In this pictures I set the red probe to AC coupling so you could see the different signals.

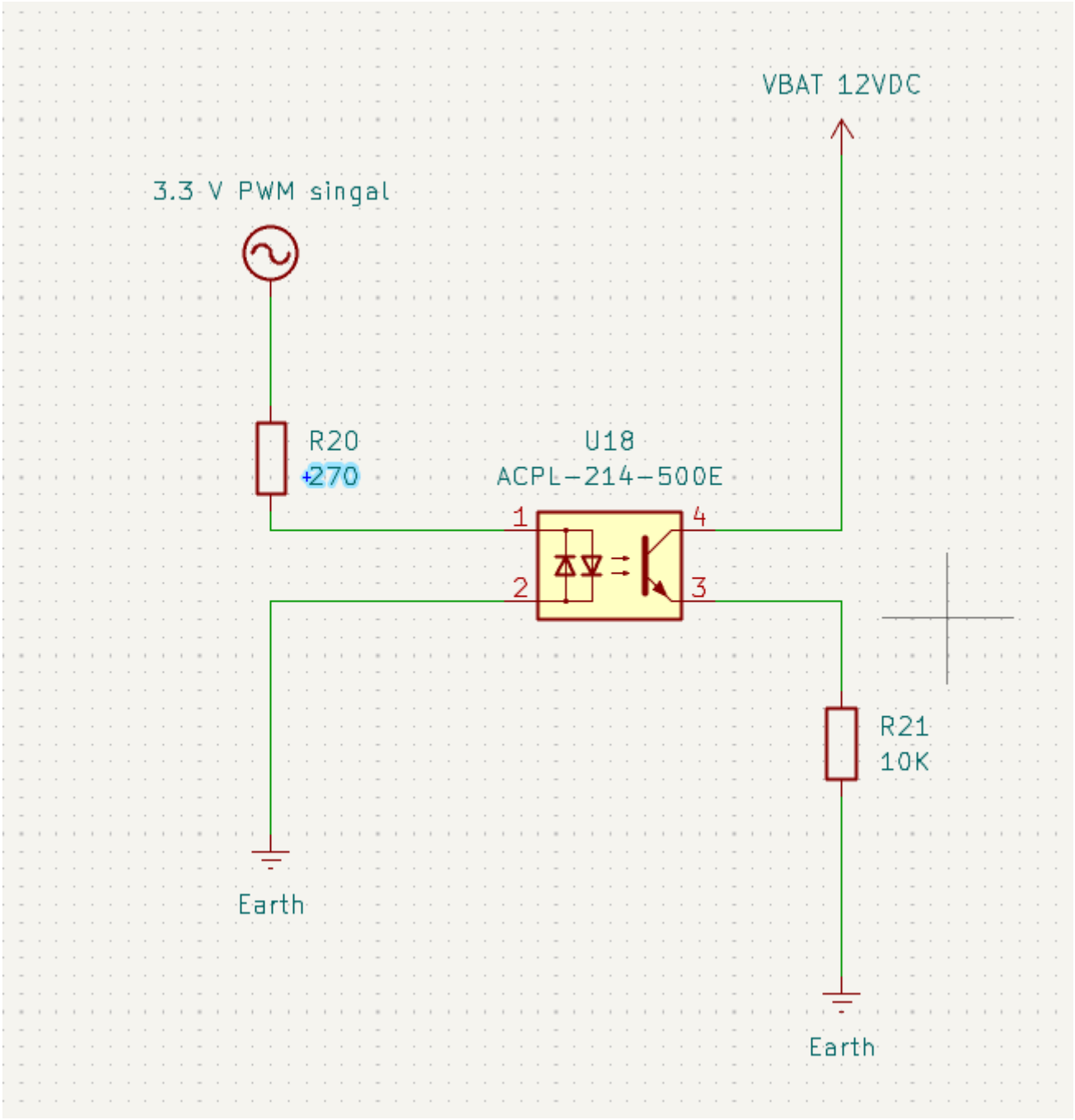
This next picture is hard to see but I am reversing direction. Going from forward to reverse to see if the waveforms are ever on at the same time. The following picture shows the transition from forward to reverse with blue on right forward and red on right reverse. The signals were never on at the same time. It should also be noted that in all that pictures above none of the unused signals were on. This is just not possible to show with my 2 channel scope.

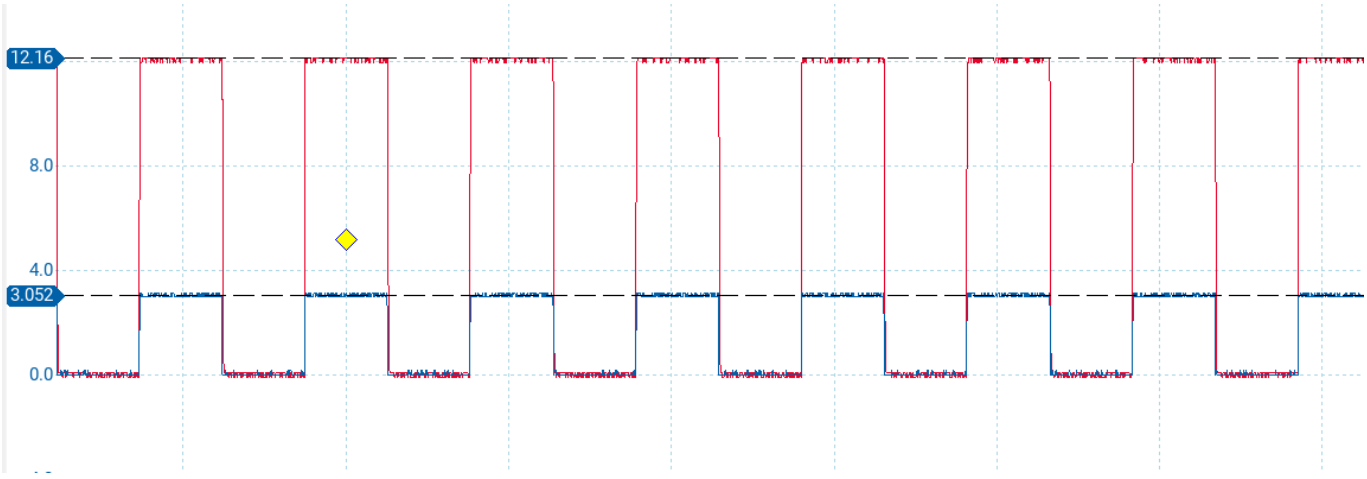


Interface design.

I am using a quad optical isolator chip to isolate the pi from the other circuitry. According to the datasheet it seems to me that they are saying that it takes 20mA to turn on the LED enough to saturate the "BJT" this is a problem as each pin on the pi can only output 16mA and and a total max for all the pins of 50 mA. which limits me to 12 mA per pin. However the ouput of the optoisolator should only draw microamps or a few mA so I might be able to get away with it.

Ok I take that back I set up the following circuit with a current limiting resistor of 270 ohms limiting the current to 8mA. And the outuput connected to a 10k resitor powered with a 12 volt source and the resistor drops the full 12 volts.





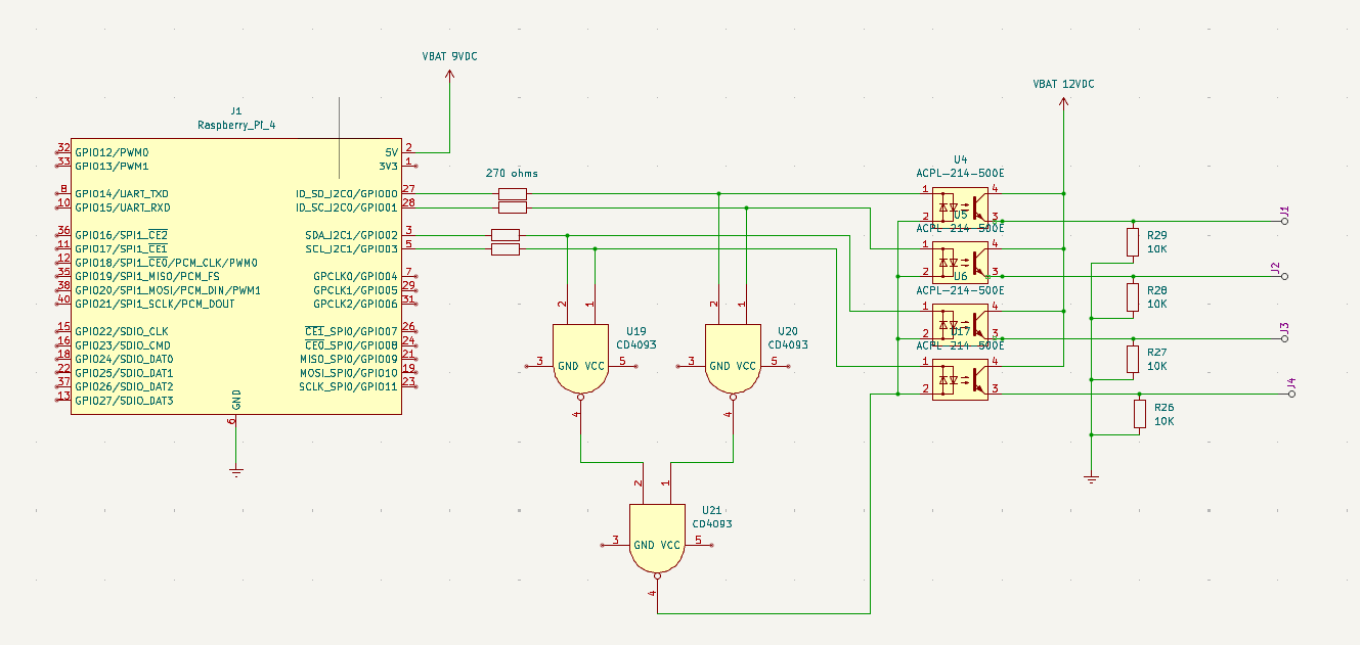
The blue line indicates the 3.3 V PMW input and the red indicates the full 12 volt PWM signal across the 10 k resistor.

all lab day 9

interface design

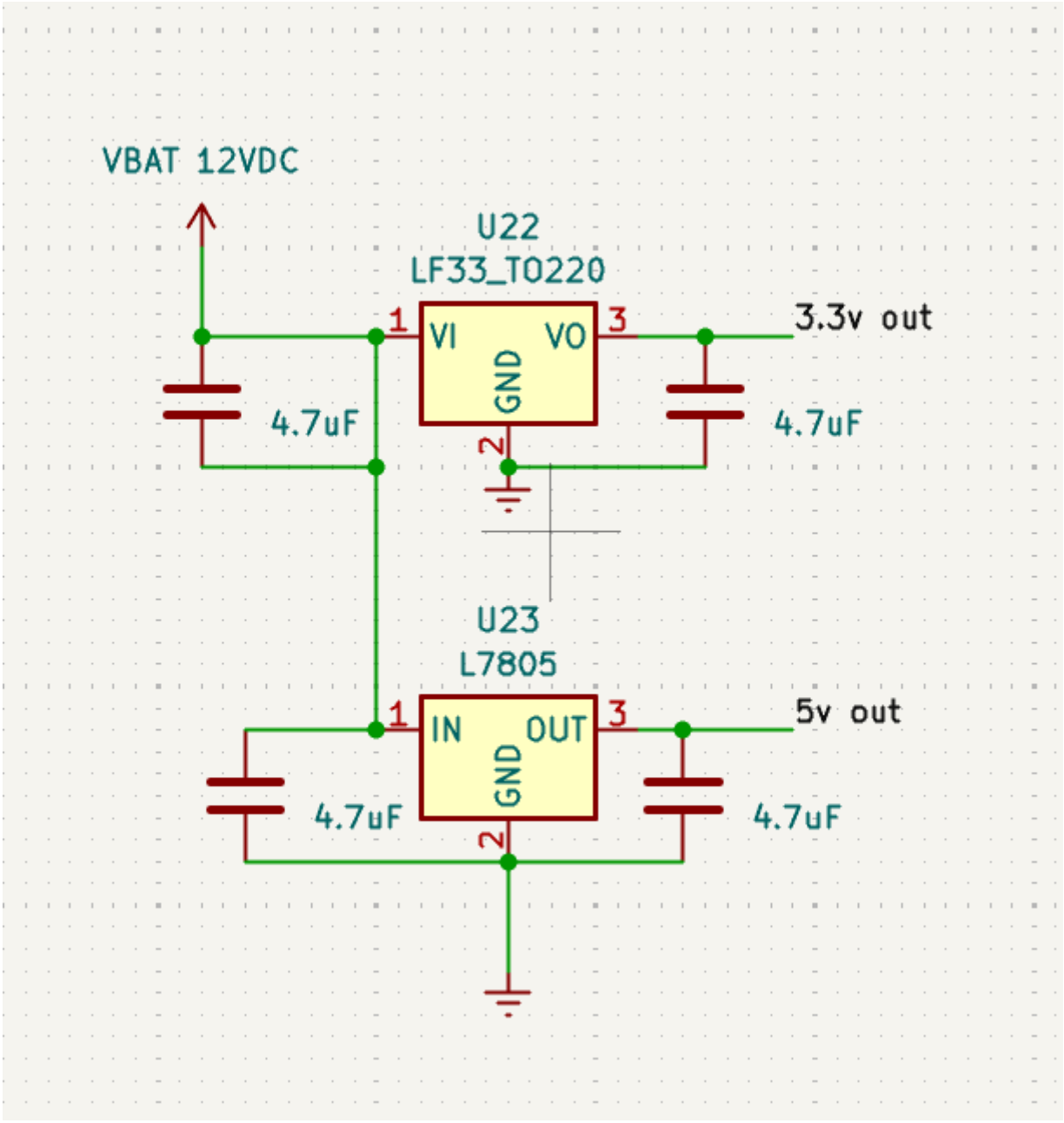
I continued with the 4 input opto-isolater testing and found this should work well. All signals all coupled fine even at full power and I am during less than the max rating for the Pi's output. Each pin draws 8ma with only two outputs ever sending signals at 1 time.

- Over current protection - Tony found what I think is a brillant way of hardware protecting an over current condition within my H bridge. The issue that could happen is if the forward and reverse commands for one set of motors were to ever go high at the same time my H bridge directly short + to -. To solve this he came up with using 3 NAND gates which is equivalent to and in mind easier to think of has two and gates and an OR gate. One and gate for each set of controls and an OR gate used so that if either and gate detects 2 highs at one time it sends a high to the ground point of my optoisolators making it to where they cannot turn on.



This was tested and it worked but the oscillscope graphs were boring and I forgot to try and take pictures that would mean something. Basically with two highs I would get a low ouput.

- power supplies - The Raspiberry PI needs to be on a seprate supply from the motor supply. The Pi can be powered via the GPIO pins and takes 5 volts. The control cricuitry ie. NANDS recieve 3.3 volt signals from the PI so they need to be powered via 3.3 volts. So I will have a seprate 12 volt lead acid battery for the Pi and two regulators one for the pi and one for the NAND gates.



The rest of my day was spent soldering a circuit board for this circuit online above.

all lab day 10/11

Last week (all lab day 10) I talked with a kid on my team who is a software engineer at SEL in R and D. He pointed out to the inefficeincy of my code. Previously the code would save the terminal settings, change the terminal settings and then put the setting back to the orginal settings each time the loop ran. We fixed this so it changes the settings to raw mode at the beginning of the loop keeps and keeps it in raw mode for as long as the loop is running. Then if the loop is broken out of it reverts back to the orginal settings.

Then today, I set a hostname up for the raspiberry pi so I don't have to find the IP address everytime the pi is powered down.

Hostname: robotraspi

And finished soldering my interface board. Ounce this was done I was able to hook it up to the robot and run it at which point I found the issue within my code. If the motors are going one direction and press a key that

cuases them to go the other direction it does so inmetiently blowing my motor fuses. Tony came up with a fix for this. The following code does so.

```
elif key == "w": # forward key
            if mrightreverse.value > 0:
                sleep(0.25)
            if mleftreverse.value > 0:
                sleep(0.25)
            mrightforward.value = speed # sets the duty cycle to user input speed
            mleftforward.value = speed # sets the duty cycle to user input speed
            mrightreverse.value = 0
            mleftreverse.value = 0
            print(f"\rspeed left = {speed}, speed right = {speed}") # prints speed
on user screen
```

If the code goes to write one of the PWM speeds it first checks if the speed in the opposite direction is moving and if so it waits a 1/4 second.

The delay part of this code works I just need to set the previous speed equal to zero first then wait.

Total edited code.

```
import sys #python library that allows for me to interact with the system
import termios #this python library controls terminal settings like turning off
line buffering
import tty # puts the terminal into raw mode (instantously looks for inputs.
from gpiozero import PWMLED # GPIO libary and PMW libary
from time import sleep # Sleep is a time delay

mrightforward = PWMLED(18) #Right set of motors PWM signal to move forward GPIO 18
mleftforward = PWMLED(19) #Left set of motors PWM singal to move backwards GPIO 19
mrightreverse = PWMLED(12) #Right set of motors PWM siganl to move forward GPIO 12
mleftreverse = PWMLED(13) # Left set of motors PWM signal to move backwards GPIO
13

def get_key(): # function that looks at a single user input key
    ch = sys.stdin.read(1) # reads one character from stdin
    return ch

print("Press keys (press 'q' to quit):")

speed = 0 # varaible for user input speed

# --- set terminal into raw mode once ---
fd = sys.stdin.fileno() # used to change the settings of the terminal
old_settings = termios.tcgetattr(fd) #saves the terminals current setting so we
can revert back to them
tty.setraw(fd) # turns the terminal to raw mode "doesn't need the user to press
enter after each key press."
```

```
try:
    while True:
        key = get_key() #runs the get key function defined above
        #oldkey = key
        print(f"\r{key}")

        if key.isdigit(): # checks if key is a number
            num = int(key) # if key is a number convert it to an interger
            speed = num / 10 # converts to a "percentage" for speed command used
to set pmw speed

        elif key == "w": # forward key
            if mrightreverse.value > 0:
                sleep(0.25)
            if mleftreverse.value > 0:
                sleep(0.25)
            mrightforward.value = speed # sets the duty cycle to user input speed
            mleftforward.value = speed # sets the duty cycle to user input speed
            mrightreverse.value = 0
            mleftreverse.value = 0
            print(f"\rspeed left = {speed}, speed right = {speed}") # prints speed
on user screen

        elif key == "s": # reverse key
            if mrightforward.value > 0:
                sleep(0.25)
            if mleftforward.value > 0:
                sleep(0.25)
            mrightreverse.value = speed
            mleftreverse.value = speed
            mrightforward.value = 0
            mleftforward.value = 0
            print(f"\rspeed left = {speed}, speed right = {speed}")

        elif key == "a": # left key
            if mleftforward.value > 0: # which will cause the robot to turn left
                sleep(0.25)
            if mrightreverse.value > 0:
                sleep(0.25)
            mleftreverse.value = speed # sets left set of wheels reverse signal to
user input speed
            mrightforward.value = speed
            mleftforward.value = 0
            mrightreverse.value = 0
            print(f"\rspeed left = {speed}, speed right = {speed}")

        elif key == "d": # right key
            if mrightforward.value > 0:

                sleep(0.25)
            if mleftreverse.value > 0:
                sleep(0.25)
            mrightreverse.value = speed
            mleftforward.value = speed
```

```
            mrightforward.value = 0
            mleftreverse.value = 0
            print(f"\rspeed left = {speed}, speed right = {speed}")

        #elif key != oldkey
        #    mrightforward.value = 0

        elif key == "q": # breaks out of loop
            mrightforward.value = 0
            mleftforward.value = 0
            mrightreverse.value = 0
            mleftreverse.value = 0
            print("exit")
            break

        sleep(0.05) # slight delay to prevent CPU overload

finally:
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings) # takes terminal back
out of raw mode
```

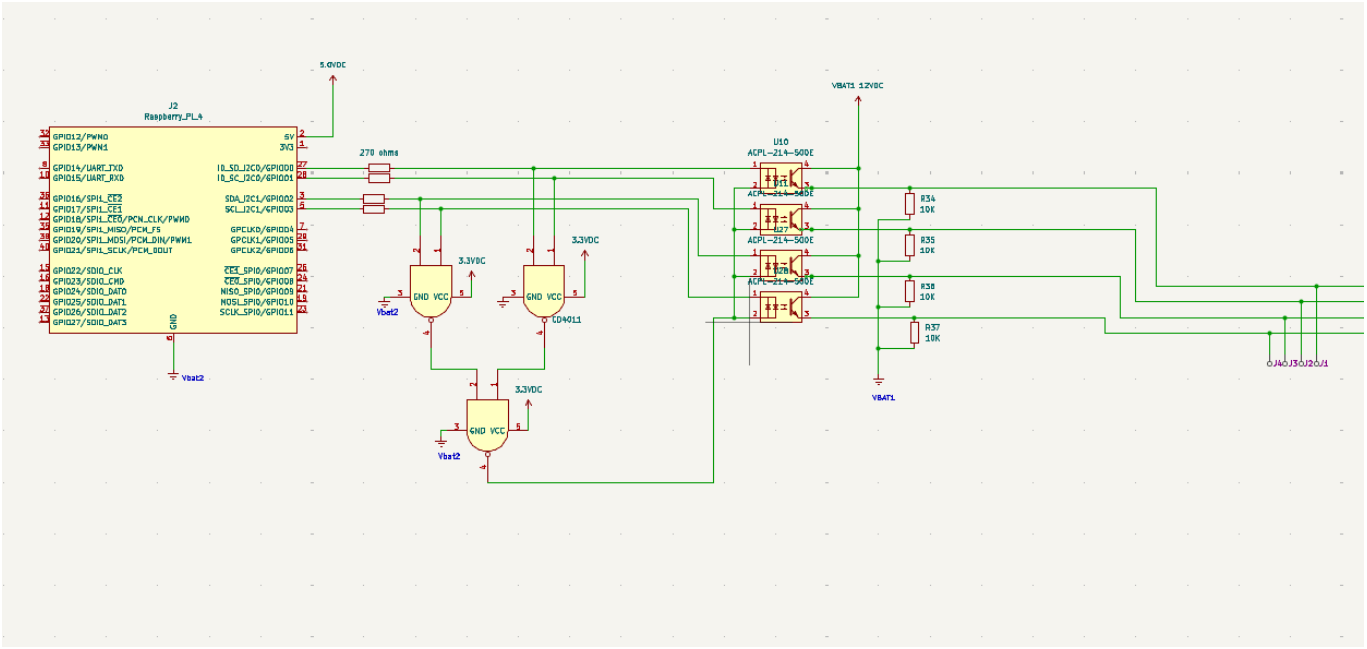
all lab day 13

Started off today by removing the old interface circuitry for the manual controls. This included removing the old circuit board and moving some of the power conductors around to better locations with the new layout. During this I bird caged a wire so that one of the strands in a way that it short to ground past one of my N-channel MOSFET's. This took me a couple of hours to trouble shoot. After this was done I was able to get my circuit to run the motors just fine on the desk. I also found that when switching the direction that a motor spins the .25 second time delay I previously had was not enough so I increased it to 1 second.

Schematic updates.

The first thing I did was remove the time delay capacitor. This was used becuase in my previous circuit I had decoupling capacitors that took some to to charge up. Which could cuase for short circuits to occur on start up. This is no longer needed in this circuit and for a reason unknown to be when the switch was first turned on the RC would joit forward a bit on first startup.

Some other schematic updates were the seprate DC supplies and making sure that there were well marked showing that they were completely seprate.



Vbat2 is the raspiberry pi 5volt reg. and optoisolators.

Vbat1 is everthing else.

Another update is I bought a 5 volt switch mode regulator used to plug USB devices into to power the pi. It will output up to 2.4 amps and gets away from the over heating of the linear 5 volt reg. I previously used.

The last thing I should have to finish up a custom battery mount for the bottom of the car has my circuitry panel will not clear the motor to fit in the bottom and my batteries will. I will work on this over the weekend.

All lab day 14

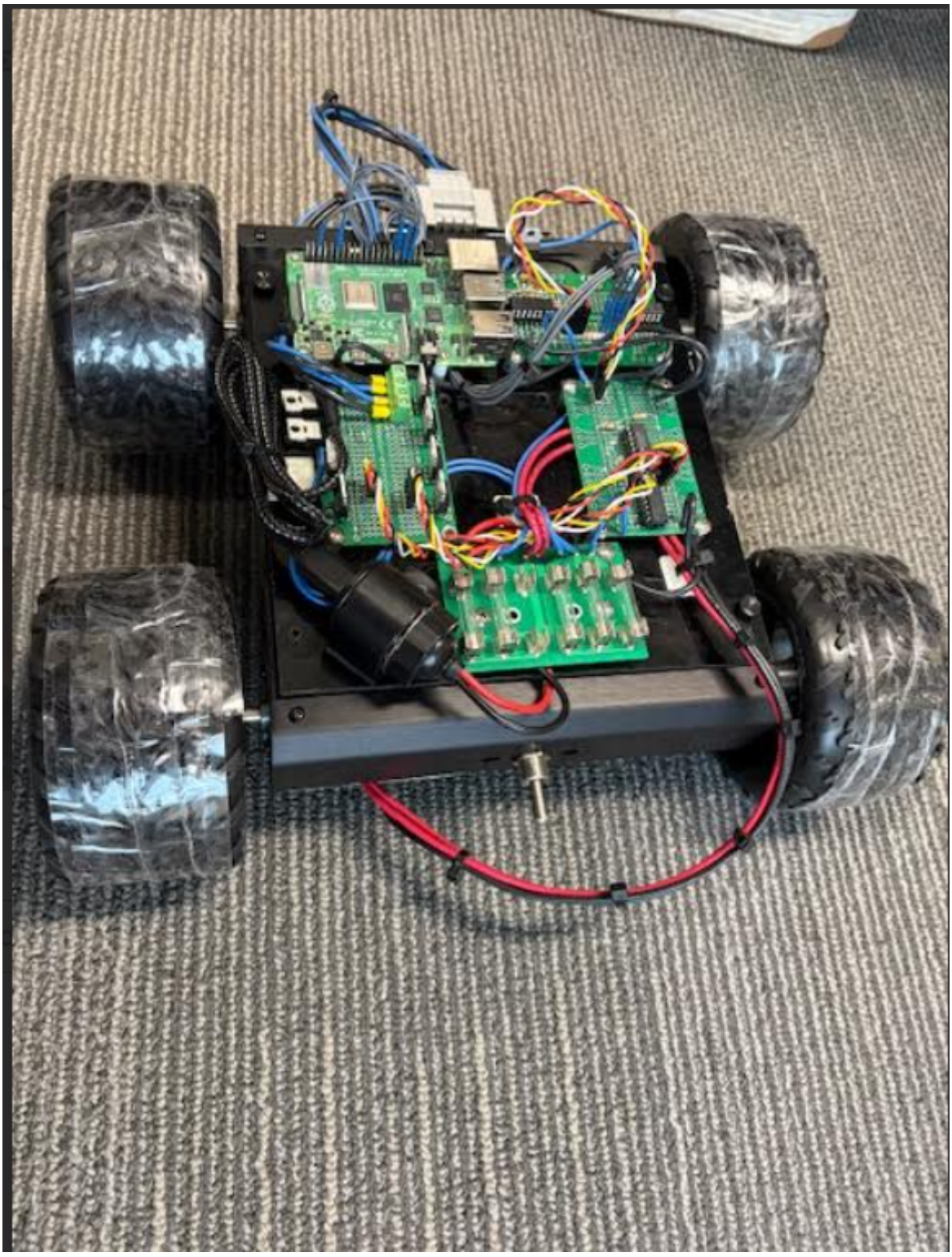
I had built the custom battery tray which allowed for me to put the control circuitry on top. It is a simple piece of aluminum bolted to the bottom with long bolts and double nuts so that the aluminum hung below the bottom of the chassis by 1/2 inch.

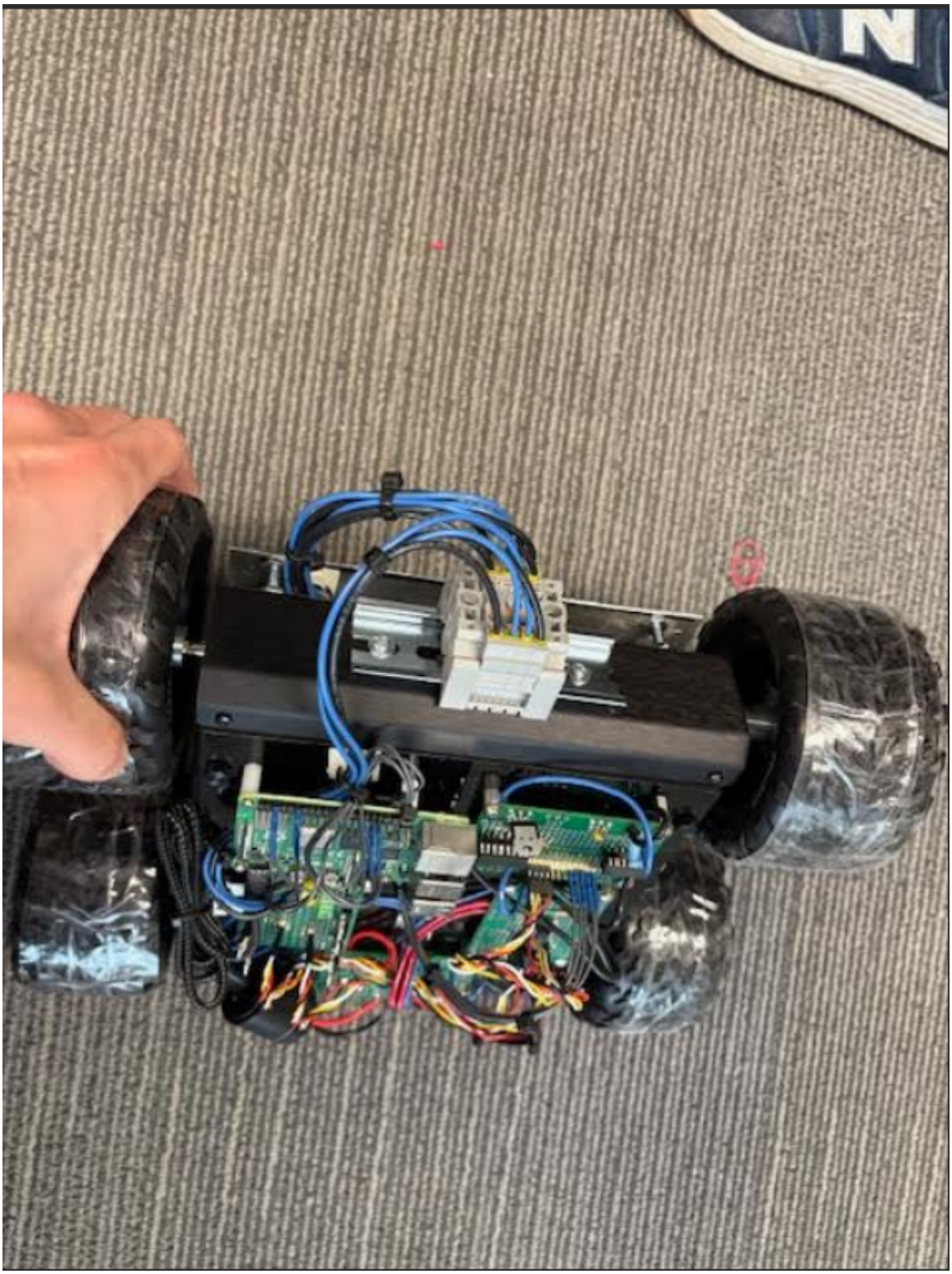
Groves were cut into the aluminum allowing for straps to be ran over the batteries to hold them in place.

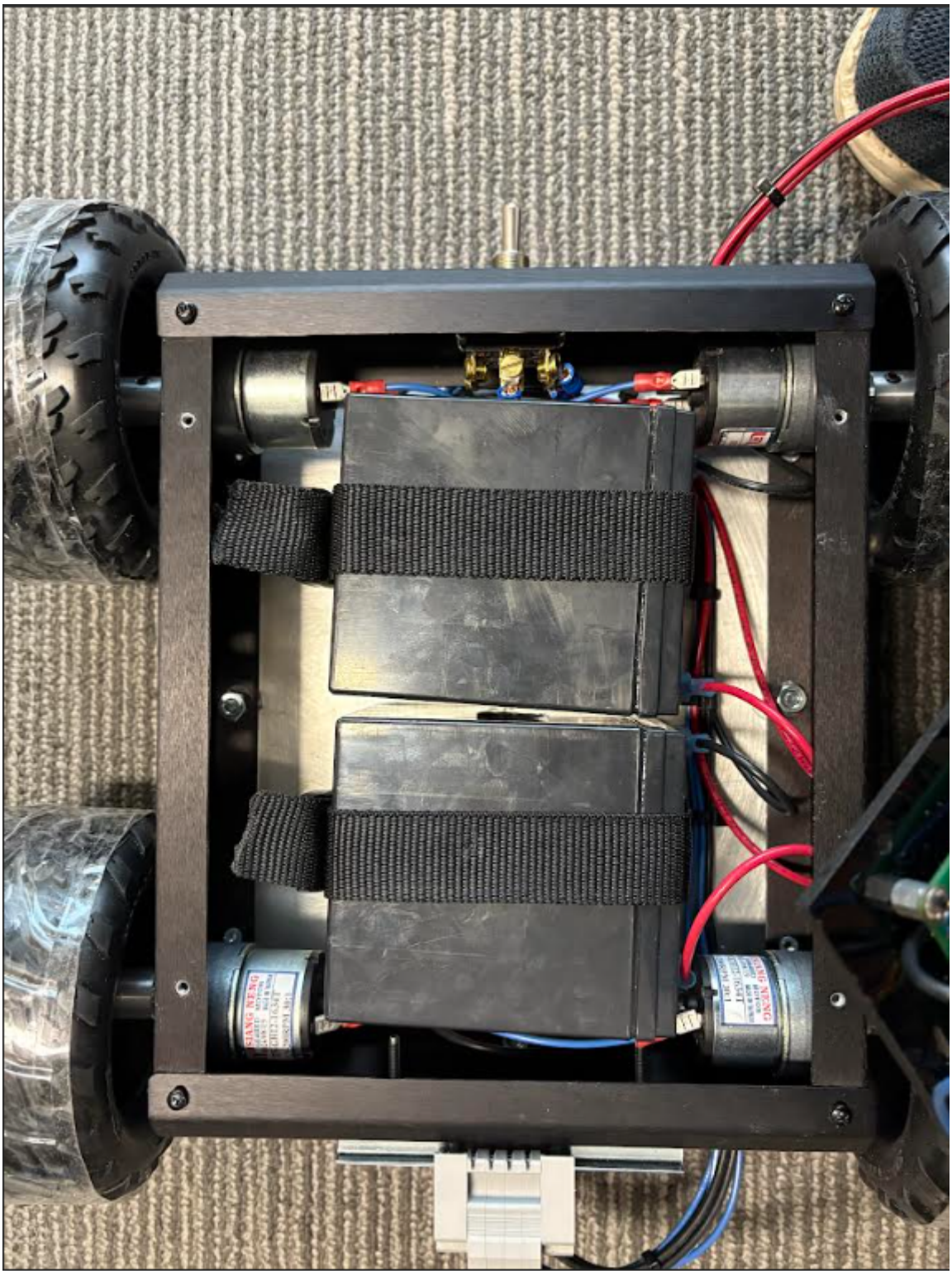
I rembered that I needed power switch turn off the robot when it wasn't in use. This was mounted in the front of the robot.

I used larger motor power wires that did not allow for me to fit the two wires in parallel into the PCB mounted terminal blocks. So I mounted normal terminal blocks on the front of the chassis allowing for me to land the two wires together then run one up to the PCB terminal blocks.

I also speant some more time cleaning up the wiring.







code edits

I found that the lowest PWM duty cycle that the motor would still drive forward with was 40%. I also found that the robot really struggles with turning espically with the added weight of the second battery.

In the forward and reverse direction the user is still able to select speeds I just scaled the user input speed. (user presses keys 1-9 to select a speed) but rather than this being 10% - 100% it is now 40% to 100 %. Which colerates well with 10% robot speed and 100% robot speed rather than PWM duty cycle. I also added in a scale for the user so it prints the percentage of speed that the robot is traveling.

I played around with turning quite a bit and found that setting one side to 100% in the forward direction and the other side to 20% in the other works well. This does not allow for turning speed to be similar to the driving speed however it is the only way I could get the robot to turn relibily.

```
    if key.isdigit(): # checks if key is a number
            num = int(key) # if key is a number convert it to an interger
            speed = ( .6 / 8) * (num-1) + .4 # scales the user input speed to
viable userinput speed
            userspeed = (num * 10) + 10 # Scales the PWM speed to robot speed
            print(f"\r speed set to {userspeed} %")
            if num == 0:
                speed = 0
                userspeed = 0
```

```
        elif key == "a": # left key
            if mleftforward.value > 0: # checks if the motor is turning in the
opposite direction than what it is going to be instructed to turn.
                mleftforward.value = 0 # if moving in opposite direction stops the
motor
                sleep(1) # waits 1 second
            if mrightreverse.value > 0:
                mrightreverse.value = 0
                sleep(1)
            mleftforward.value = 1 # sets left forward to 100% duty cycle
            mrightreverse.value = .2 # sets right reverse to 20% dutycycle
```

I also found an issue when turning. Lets say I was driving forward then wanted to turn left if I hit "a" when driving forward it would continue to drive both sides forward becuase I was only looking at one forward value in the code above. Mright forward was never set to 0 to fix this I set everything all values to 0 wait .5 seconds then change the values to turn.

```
    elif key == "a": # left key
            mrightreverse.value = 0
            mrightforward.value = 0
            mleftforward.value = 0
            mleftreverse.value = 0
            sleep(.5)
            mleftforward.value = 1.0 # sets left forward to 100% duty cycle
            mrightreverse.value = 0.20 # sets right reverse to 20% dutycycle
            print("\r left")
```

Next I tested weather or not the car would stop if the ssh connection was servered. It was not it would turn go forward randomally. I thought about putting in a timer to see look for user input and if no user input has been detected for 5 seconds turn all motors to zero. However I put my code into ai and asked it to come up with a solution for it. Basically from what I understand when the connection is served "stdin" the function that reads user input changes to an "EOF" basically it returns a blank charactor so I set it so if a blank character is

returned it stops the motors and breaks out of the loop. The reason to break out of the loop is so it returns the terminal setting back to the orginal settings

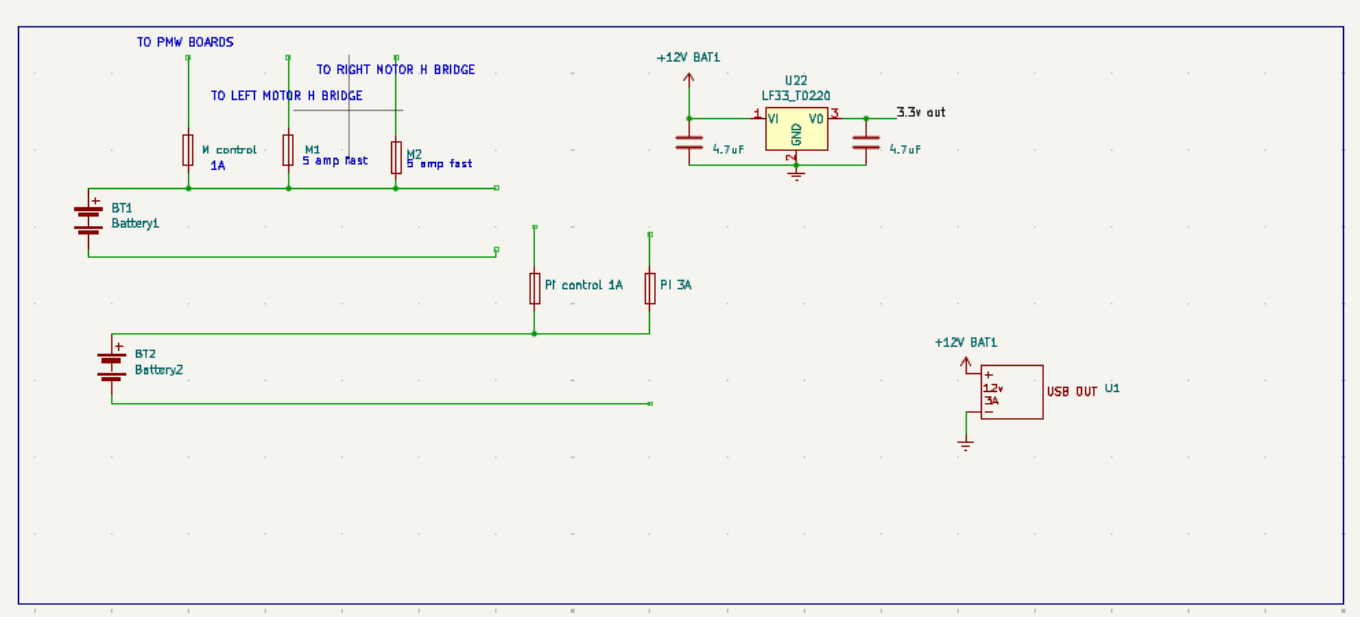
Final Code

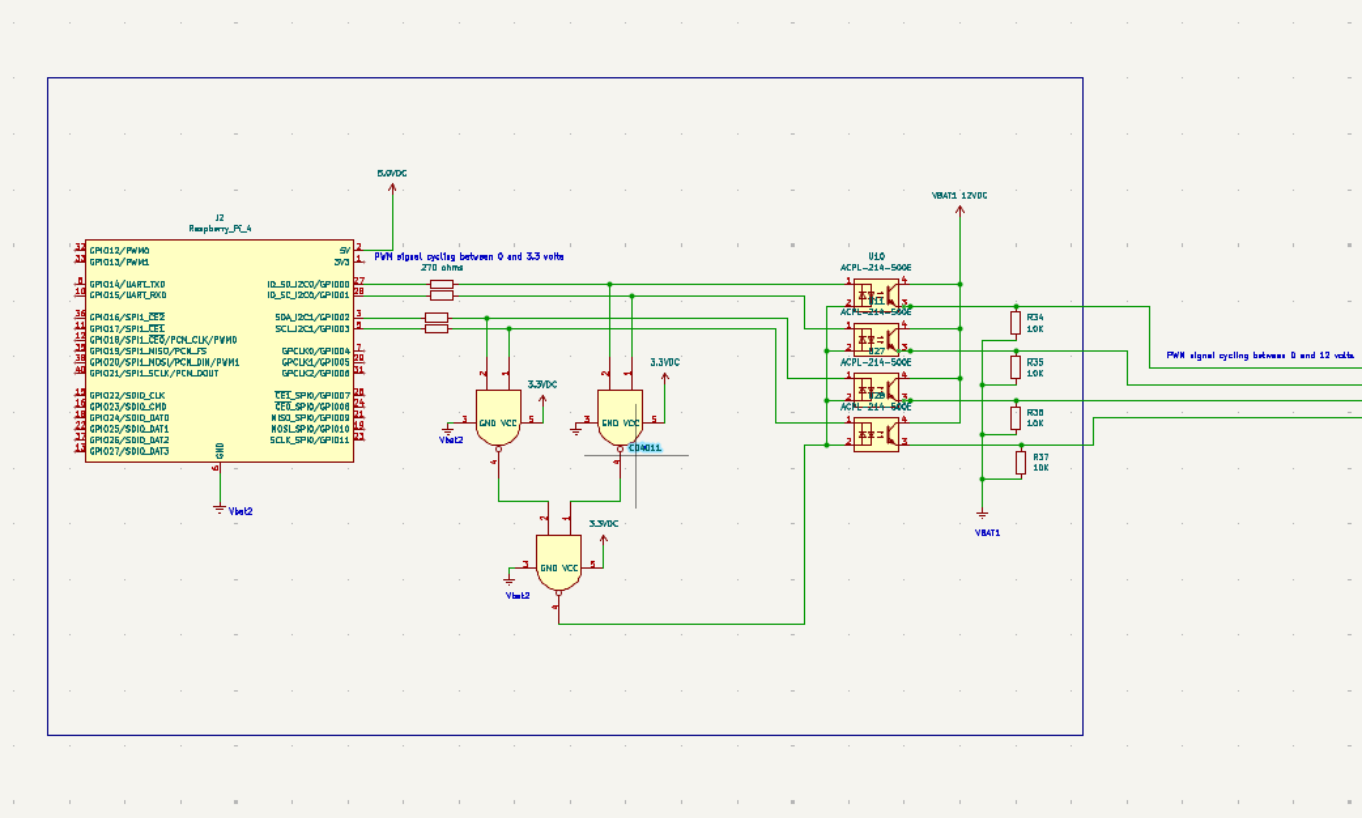
```
import sys #python library that allows for me to interact with the system
import termios #this python library controls terminal settings like turning off
line buffering
import tty # puts the terminal into raw mode (instantaneously looks for inputs)
from gpiozero import PWMLED # GPIO library and PWM library
from time import time,sleep # Sleep is a time delay. Time is a Timer
import select # Libabary that allows for the system to be continually looking for
input even without input
#Need for the deadman switch so loop will run indefinently

mrightforward = PWMLED(19) #Right set of motors PWM signal to move forward GPIO 18
mleftforward = PWMLED(18) #Left set of motors PWM signal to move backwards GPIO 19
mrightreverse = PWMLED(13) #Right set of motors PWM signal to move forward GPIO 12
mleftreverse = PWMLED(12) # Left set of motors PWM signal to move backwards GPIO
13

def get_key_nonblocking(timeout=0.05): # function that looks at a single user
input key this function was created so that it runs if no key has been
#pressed or if a key has been pressed. Previous version only ran if a key was
pressed which would not allow for the deadman switch to work properly.
    rlist, _, _ = select.select([sys.stdin], [], [], timeout) # waits for key
press or timeout
    if rlist: # if a key is pressed
        return sys.stdin.read(1) # reads one character from stdin
    return None # Returns none if no key has been read used for "deadman switch"

print("""\rWelcome to the Amazing Robot control interface. To select a speed hit
numbers 1-9 at any point. To drive press and release the following
keys forward (w), reverse (z), left (a), right (d), and (s) for stop. To exit the
program press (q) to quit:""")

speed = 0 # variable for user input speed
timelastkey = time() # initialize last key pressed timestamp

fd = sys.stdin.fileno() # used to change the settings of the terminal so that any
key input is read imdentially
old_settings = termios.tcgetattr(fd) # saves the terminal's current settings so we
can revert back to them
tty.setraw(fd) # turns the terminal to raw mode "doesn't need the user to press
enter after each key press."


try:
    while True:

        key = get_key_nonblocking(0.05)  # reads both keys and weather or not a
```

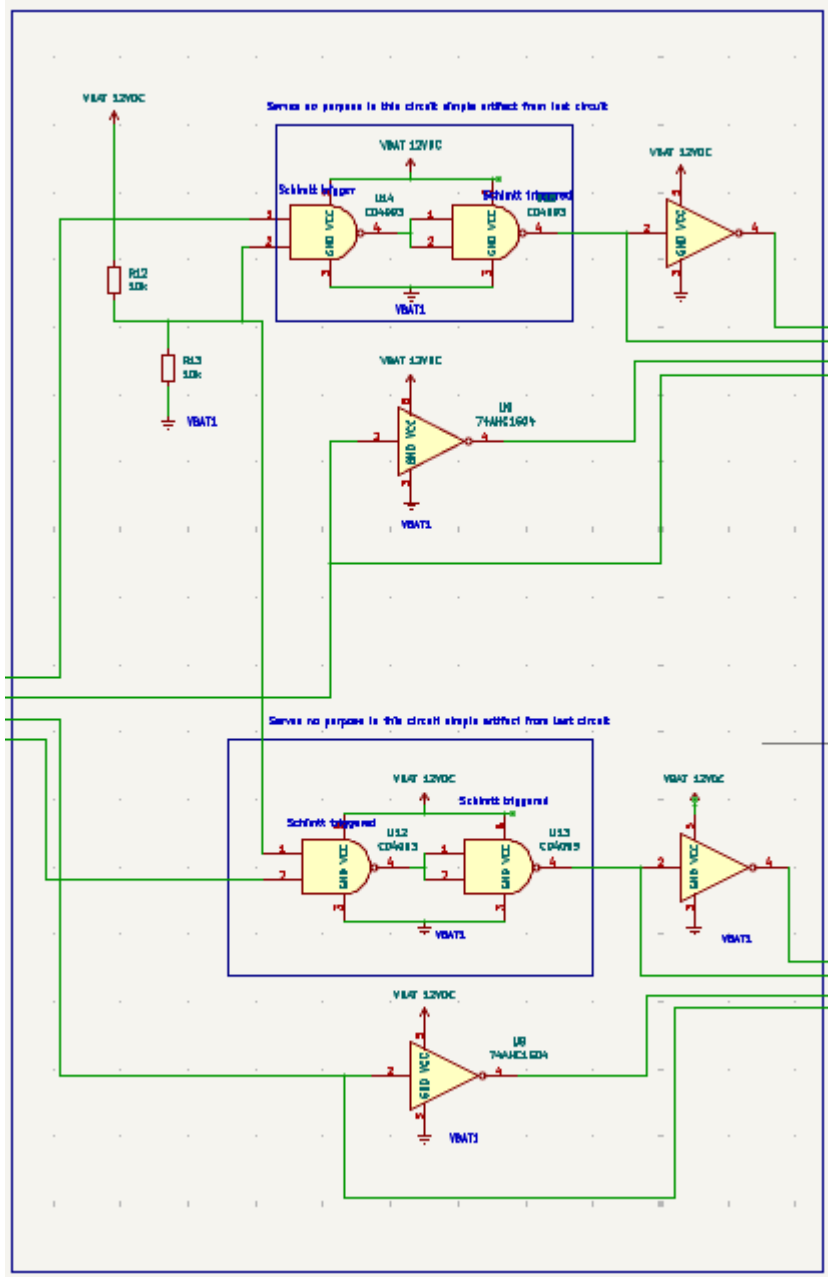
```
key has been pressed

         if key is not None and (key.isdigit() or key.isalpha()): # if any key is
pressed resets timer
            timelastkey = time()

        if time() - timelastkey > 5: # if it has been 5 or more seconds since last
key press stop all motors
            mrightreverse.value = 0
            mrightforward.value = 0
            mleftforward.value = 0
            mleftreverse.value = 0
            print("\rPress key to drive timeout error")

        if key is not None and key.isdigit(): # checks if key is a number
            num = int(key) # if key is a number convert it to an interger
            speed = ( 0.60 / 8) * (num-1) + 0.40 # scales the user input speed to
viable PWM duty cycle (40% duty cycle is the lowest duty cycle that
            # makes the car move
            userspeed = (num * 10) + 10 # Scales the user input speed to robot
speed
            print(f"\r speed set to {userspeed} %")
            if num == 0:
                speed = 0
                userspeed = 0

        elif key == "w": # forward key
            if mrightreverse.value > 0: # checks if the motor is turning in the
opposite direction than what it is going to be instructed to turn.
                mrightreverse.value = 0 # if moving in opposite direction stops
the motor
                sleep(0.5) # wait half second for the car to coast to a stop
            if mleftreverse.value > 0:
                mleftreverse.value = 0
                sleep(0.5)
            mrightforward.value = speed # sets the duty cycle to user input speed
            mleftforward.value = speed # sets the duty cycle to user input speed
            print(f"\r forward, {userspeed}")

             elif key == "z": # reverse key
            if mrightforward.value > 0: # checks if the motor is turning in the
opposite direction than what it is going to be instructed to turn.
                mrightforward.value = 0 # if moving in opposite direction stops
the motor
                sleep(0.5) # wait half a second for the motors to stop
            if mleftforward.value > 0:
                mleftforward.value = 0
                sleep(0.5)
            mrightreverse.value = speed
            mleftreverse.value = speed
            print(f"\r reverse {userspeed}")

            elif key == "a": # left key
            mrightreverse.value = 0 # sets all motors to 0 needed so we can write
```

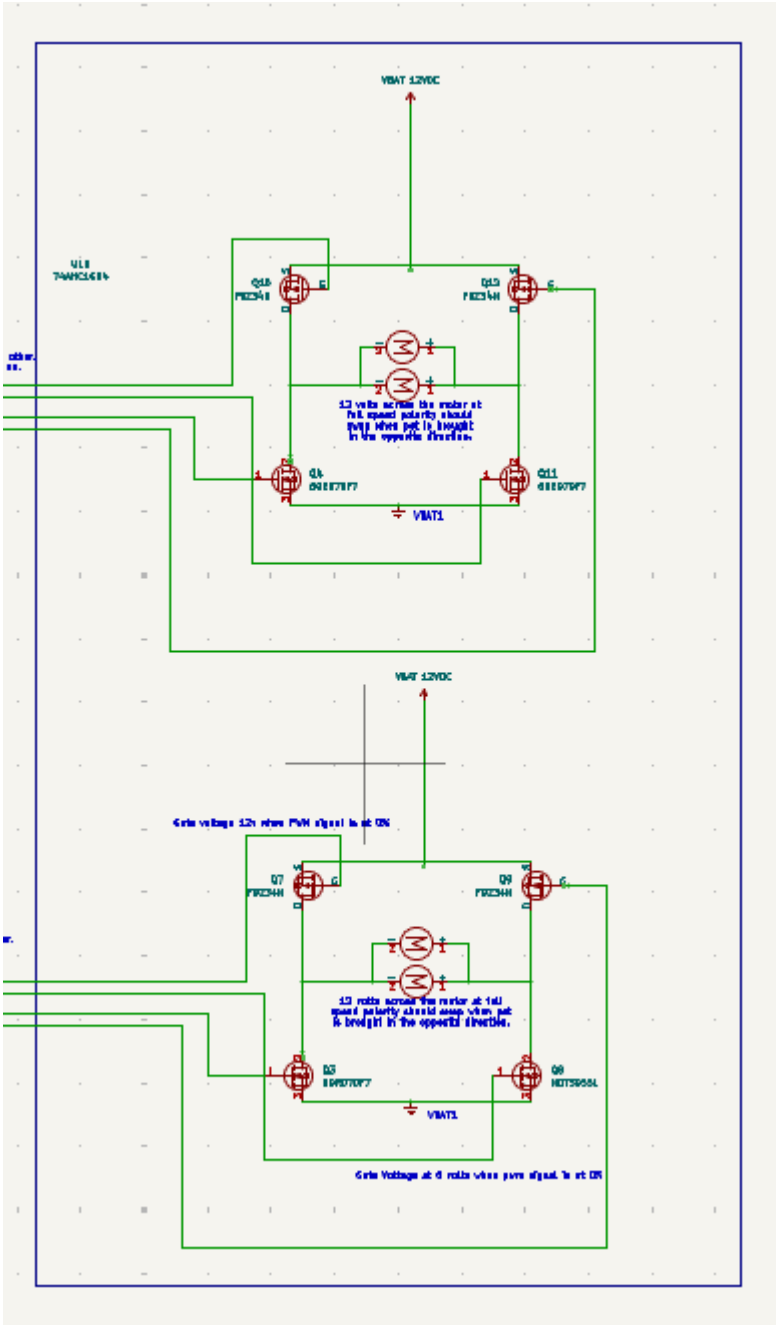
```
the proper motor speeds/direction
            mrightforward.value = 0
            mleftforward.value = 0
            mleftreverse.value = 0
            sleep(0.5)
            mleftforward.value = 1.0 # sets left forward to 100% duty cycle
            mrightreverse.value = 0.20 # sets right reverse to 20% dutycycle
            print("\r left")


        elif key == "d": # right key
            mrightreverse.value = 0 # sets all motors to 0 needed so we can write
the proper motor speeds/direction
            mrightforward.value = 0
            mleftforward.value = 0
            mleftreverse.value = 0
            sleep(0.5)
            mrightforward.value = 1
            mleftreverse.value = .2
            print("\r right")

             elif key == "s": # "break" button stops the car from moving
            mrightforward.value = 0
            mleftforward.value = 0
            mrightreverse.value = 0
            mleftreverse.value = 0
            print("\r stop")

        elif key == "q": # ends program
            mrightforward.value = 0
            mleftforward.value = 0
            mrightreverse.value = 0
            mleftreverse.value = 0
            print("\r exit")
            break

        sleep(0.07) # slight delay to prevent CPU overload

finally:
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings) # takes terminal back
out of raw mode
```

Final Schematic


TO PMW BOARDS
TO RIGHT MOTOR H BRIDGE
TO LEFT MOTOR H BRIDGE
M control
1A
M1
5 amp fast
M2
5 amp fast
BT1
Battery1
BT2
Battery2
PI control 1A
PI 3A
+12V BAT1
U22
LF33_TO220
VI
VO
GND
3.3v out
4.7uF
4.7uF
+12V BAT1
12v
3A
USB OUT U1


5.0VDC
J2
Raspberry_Pi_4
PWM signal cycling between 0 and 3.3 volts
270 ohms
3.3VDC
GND VCC
Vbat2
VBAT1 12VDC
U10
ACPL-214-500E
R34
10K
R35
10K
R36
10K
R37
10K
PWM signal cycling between 0 and 12 volts
VBAT1


VBAT 12VDC
Serves no purpose in this circuit simple artifact from last circuit
VBAT 12VDC
Schmitt triggered
U14
CD4093
Schmitt triggered
CD4093
VCC
GND
VBAT1
VBAT 12VDC
R12
10k
R13
10k
VBAT1
VBAT 12VDC
U9
74AHC1G04
VBAT1
Serves no purpose in this circuit simple artifact from last circuit
VBAT 12VDC
Schmitt triggered
Schmitt triggered
U12
CD4093
U13
CD4093
VBAT1
VBAT 12VDC
VBAT1
VBAT 12VDC
U8
74AHC1G04
VBAT1



Abstract

In this project I built a “remote-controlled” skid steer–type car. The term skid steer refers to the fact that all wheels are fixed and the steering is done by adjusting the speed of the left and right motors independently. For instance, if the right motors are moving forward and the left motors are stopped, the car will turn right. All of the electronics in this car are there to adjust the speed of each set of wheels independently.

On board the car is a Raspberry Pi, which is a very small computer. On that computer I wrote code that sends electrical pulses to the motors. The longer the pulse, the more power the motors get, meaning the faster the motors will drive. These pulses are sent to a set of electronic switches which control the direction the motors spin. If the right motor forward pulses are on, the right set of motors will drive forward. If the right motor reverse pulses are on, the right set of motors will drive in reverse.

The user of the car will remotely log into the Raspberry Pi (the small computer controlling the car). Remotely logging into this computer allows the person to run the Raspberry Pi code on their laptop and adjust

parameters. For instance, press “w” to drive forward, press “5” to go at 50% speed, press “s” to go in reverse, etc.

User’s Guide

Step 1

Turn on the robot using the toggle switch on the back (push to the right).

Step 2

Connect your computer to Warrior Guest.

Step 3

Open WSL on your computer and type the following commands:

```
$ ssh pi@robotraspi
pi@robotraspi's password: raspberry
$ python3 RobotControl.py
```

Step 4

You are now running the program. Press the desired speed you wish to drive at using keys 0–9 (0 = 0% speed, 9 = 100%). Press the following keys to drive.

Note: that after a key has been pressed and released, the car will continue to drive in that direction until another key has been pressed.

Press and release w to drive forward

Press and release z to drive in reverse

Press and release s to stop

Press and release a to turn left

Press and release d to turn right

Press any number 0–9 to change speed

Note: If the user does not press a key for 5 seconds or more, the car will stop moving and will print “Press key to drive timeout error.” If this happens and you wish to continue, press any key to drive again. For instance, if you are driving forward and haven’t pressed another key for 5 seconds, it will stop. If you wish to continue driving forward, press w again and it will start to drive forward.

Step 5 — Exiting the Program

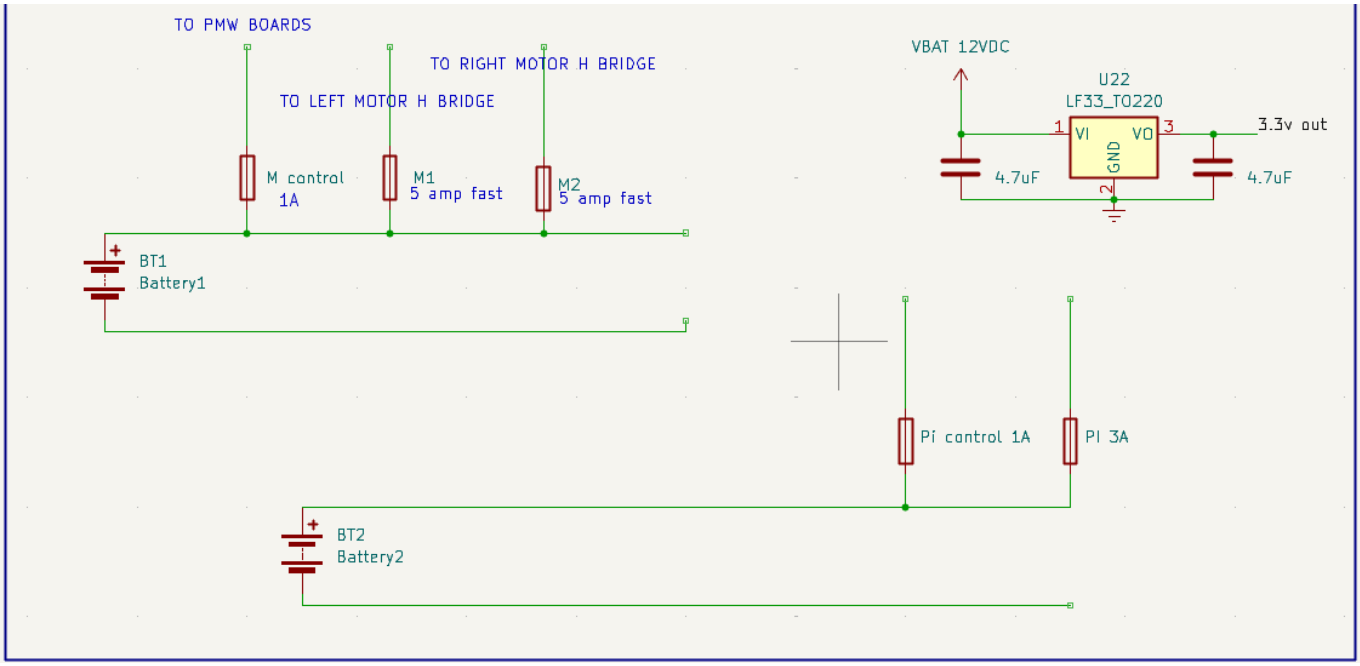
Press q at any time to exit the program. Once you have exited the program, you are still SSHed into the Raspberry Pi. If at this point you wish to power down the car, it is important that you shut down the Pi first. Run the following command:

$ sudo shutdown now

You may now turn off the robot using the toggle switch located on the back.

Maintenance

If a fuse ever needs replaced, follow the schematic shown and the corresponding fuse labels located under each fuse.

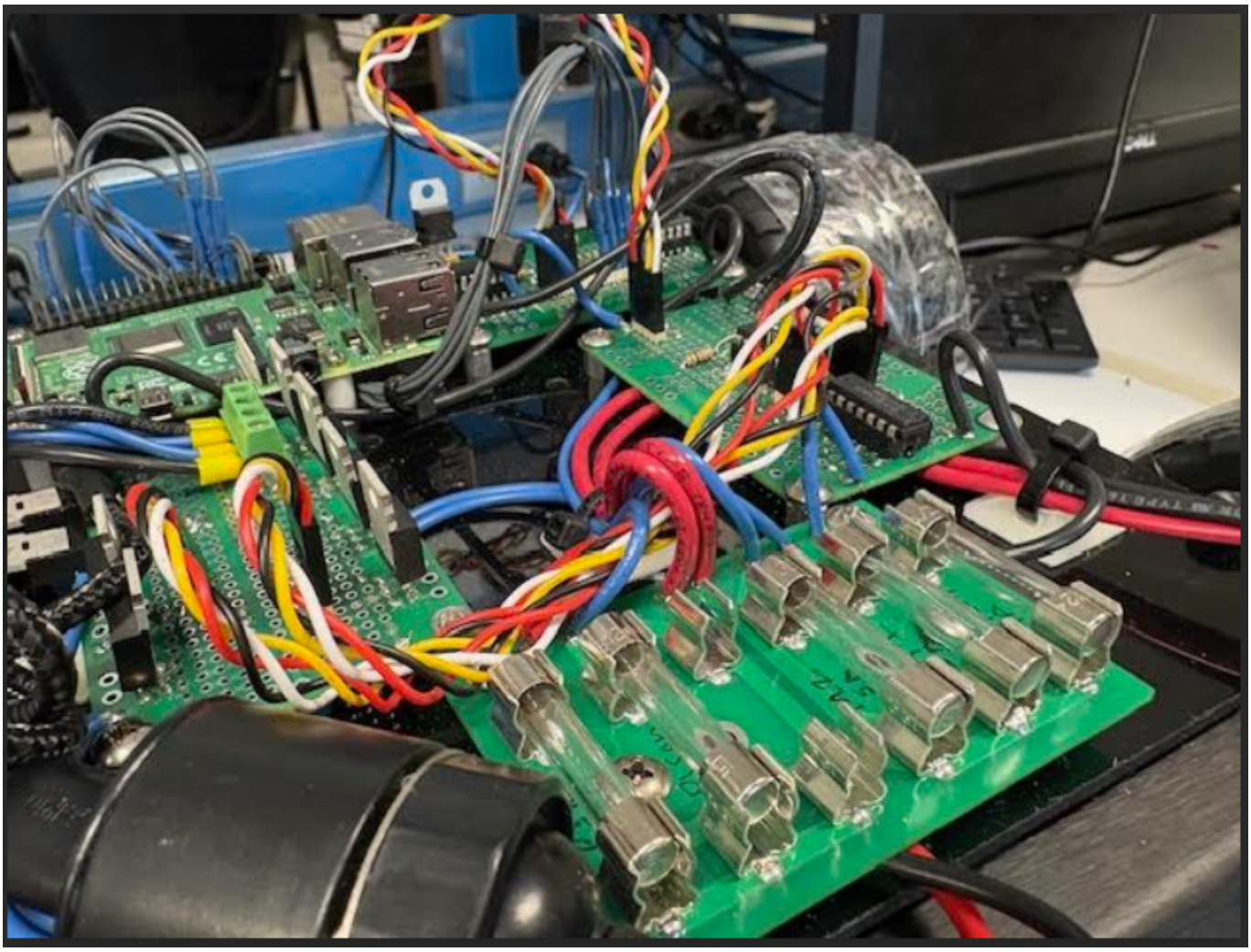


To charge the batteries, remove the top cover using the four thumb screws. Undo the battery straps and remove the spade connectors from the battery. Note: Do not pull the spade connectors off by pulling on the wire—grab the connectors themselves.

Siganl wires:

If the signal wire orientation is swapped, it will short-circuit the H-bridge. The picture below shows the correct orientation. The 4 wire cables that connect to the Hbridge connect the wires so that pins don't cross. Ie. the back pin on board 1 connects to the back pin of board 2.

For the isolated board (the one next to the Raspberry Pi), the red wire should be on the right side when you are looking at the back of the robot (the side with the switch). The other end of the cable connects to the motor driver board behind it, with the red wire facing forward.



Bill of materials

Robot project previously made and outline ----------$347.00

Raspiberry Pi ----------$63.00

https://www.amazon.com/Raspberry-Pi-Model-2GB/dp/B09TTNPB4J/ref=sr_1_6?crid=2ACI4QCR92DLR&dib=eyJ2IjoiMSJ9.I9DWX787q5nRmx2BUn6JKEuRON1rTweWZ8cQXTE3xcwbpl2avqA1cHevwze_9se0Kci_ByyR6syl37Ar9ZQWhgSSxqLngnHcmM8ZewQgGO4lJFhH5d3W-8seF1fduDBtAhxgDjsFAkLFgVT3CxmbNeQU3YeQz6vdyA-7A0rpBiHNQNDVUDolGSiTfU2-_Tm8wKXS0a3J5rzUAN7FOSJoL0WrC2jZUgcW8WiqQlIXJ_uuVEkK4yrJTRpoKRmq6KBocdlxmmZlBuTUczvhiOQDtRQOtfXq1RluAc0XT3RK6CE.8vq9kZdUD4qsASvQxqxwluvfGm5YdPuNv44yiEm36RI&dib_tag=se&keywords=raspberry+pi+4&qid=1764891678&sprefix=rasp%2Caps%2C139&sr=8-6

USB switch mode supply ----------$9.00

https://www.amazon.com/Female-Converter-Adapter-Charging-mounting/dp/B0B8C5MT37/ref=sr_1_2_sspa?crid=3MDVDU4PSTA1F&dib=eyJ2IjoiMSJ9.XHDu0yP2MaWenRQXP1F8UfgrqNyHwwY-XIMiCTy2utd9XUKDvenlc6wfTB1tQz0A8Y-pIKdAv8CaN1Y_4tFoaPwOJnDxcBP0tEbBwsrSErSZ5Z9EZgCY1I5QFpd-

ZjNWXc1_lP279rrXhyHt2Ld9TvwVAW5Gi0dhFWqn7po8-1m-8ZOX8f4FTFbBnMdeWtOoLyehr7v9wEEs6jfsPcuX7Zw_FnDlHDixq6DzDOFtiw0.tJFMbjei6wwwh17vFUlzumdxLPqxniS1DimVzYx_Ld0&dib_tag=se&keywords=12v%2Bto%2Busb%2Bconverter&qid=1764891965&sprefix=12v%2Bto%2Busb%2Bcon%2Caps%2C148&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1

USB to USB C cable ----------$6.00

https://www.amazon.com/Amazon-Basics-Charger-480Mbps-Certified/dp/B01GGKZ2SC/ref=sr_1_7_ffob_sspa?crid=2F0QSROMV69TL&dib=eyJ2IjoiMSJ9.oD_EH9aZtLqkp1r6B_7OhNZP80lRkYZQwcjd36Z6dzEKqPnlTygNlBl2nFLs0WBB1FEJ_cUSGHZkqVbJbIbRZX5knkJQs3NR_eD6YAtR5FAWOwJIqbbNykG4-t0swwj4P5CEpgsjpAaH8zdin3xItfvGtTx7NQBDLzv64C0Qn1adhv_7qlFU6FN19dYoYJbF57LTj4LjBgSMAOfzHo_Cq-JEaD8oKpq1MNWspOGbmRQ.g9Ru7_8yenuoAq8pRQxN7WS4xMw1XsMOuac_J-dY8hg&dib_tag=se&keywords=usbc+cable&qid=1764892004&sprefix=usbc+%2Caps%2C168&sr=8-7-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9tdGY&psc=1

3.3 Volt regulator ----------$1.50

https://www.digikey.com/en/products/detail/stmicroelectronics/LF33CV/1038546

NAND gate ----------$0.70

https://www.digikey.com/en/products/detail/texas-instruments/CD4011BE/67241

Optoisolators ----------$1.00

https://www.digikey.com/en/products/detail/liteon/LTV-846/385834

Toggle Switch ----------$2.20

https://www.digikey.com/en/products/detail/cit-relay-and-switch/ANT21SECQE/12503353

Terminal Blocks ----------$8.00

https://www.digikey.com/en/products/detail/phoenix-contact/3209578/2263912

Din Rail ----------$5.00

https://www.digikey.com/en/products/detail/altech-corporation/2511120-1M/8546913

End Caps ---------- $3.00

https://www.digikey.com/en/products/detail/galco-industrial-electronics/CA202/15658135