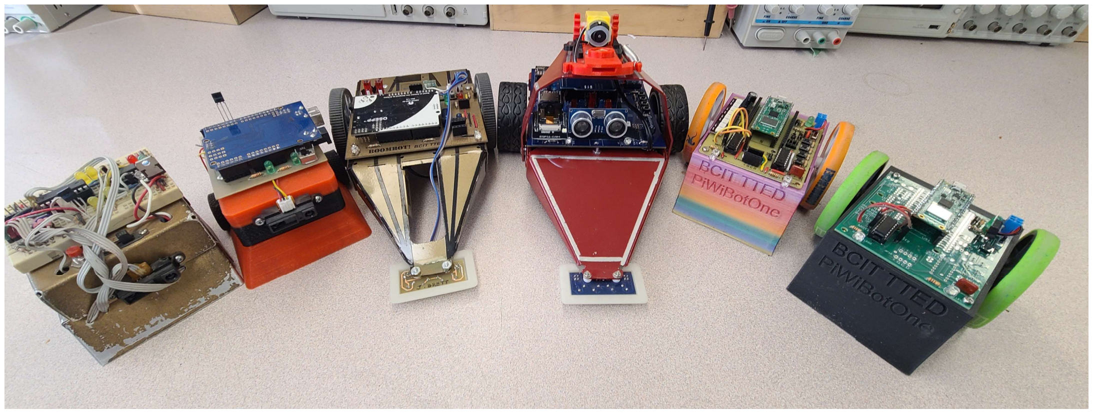
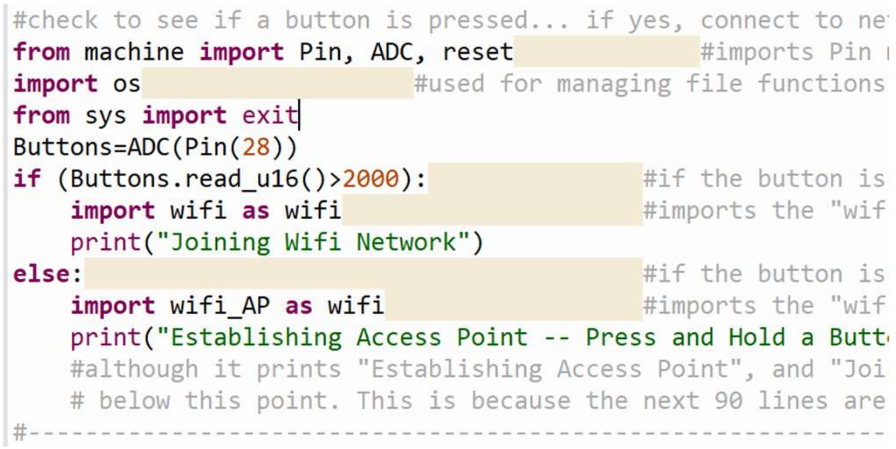


PiWiBot Software

How I think it works.

Jason Brett
BCIT TTED
January 2025

Coded in MicroPython



```
#check to see if a button is pressed... if yes, connect to ne
from machine import Pin, ADC, reset          #imports Pin 
import os                                    #used for managing file functions
from sys import exit
Buttons=ADC(Pin(28))
if (Buttons.read_u16()>2000):                #if the button is
    import wifi as wifi                      #imports the "wif
    print("Joining Wifi Network")
else:                                        #if the button is
    import wifi_AP as wifi                   #imports the "wif
    print("Establishing Access Point -- Press and Hold a Butt
    #although it prints "Establishing Access Point", and "Joi
    # below this point. This is because the next 90 lines are
#----------------------------------------------------------
```

- An Interpreted Language:
  - The PicoW doesn't "run your code"
    - It runs a "MicroPython Interpreter"
    - Your source code is stored on the PicoW and executed one command at a time
  - Use "Thonny" to install the MicroPython interpreter
    - Then to upload your code to the PicoW
- On PicoW boot up the Micropython Interpreter starts to run
  - It looks for a file called "main.py"
    - It executes the first instruction in that file
      - Then the next, etc.

Different from the Arduino IDE

- Arduino IDE:
  - Compiled language (“C++”)
    - Your computer converts your source code to machine language instructions
  - The code on the microcontroller is in machine code
    - You can’t read it or change it
    - You have to re-compile and re-download
  - Benefit: Compiled code runs faster than interpreted
    - (as a general rule)

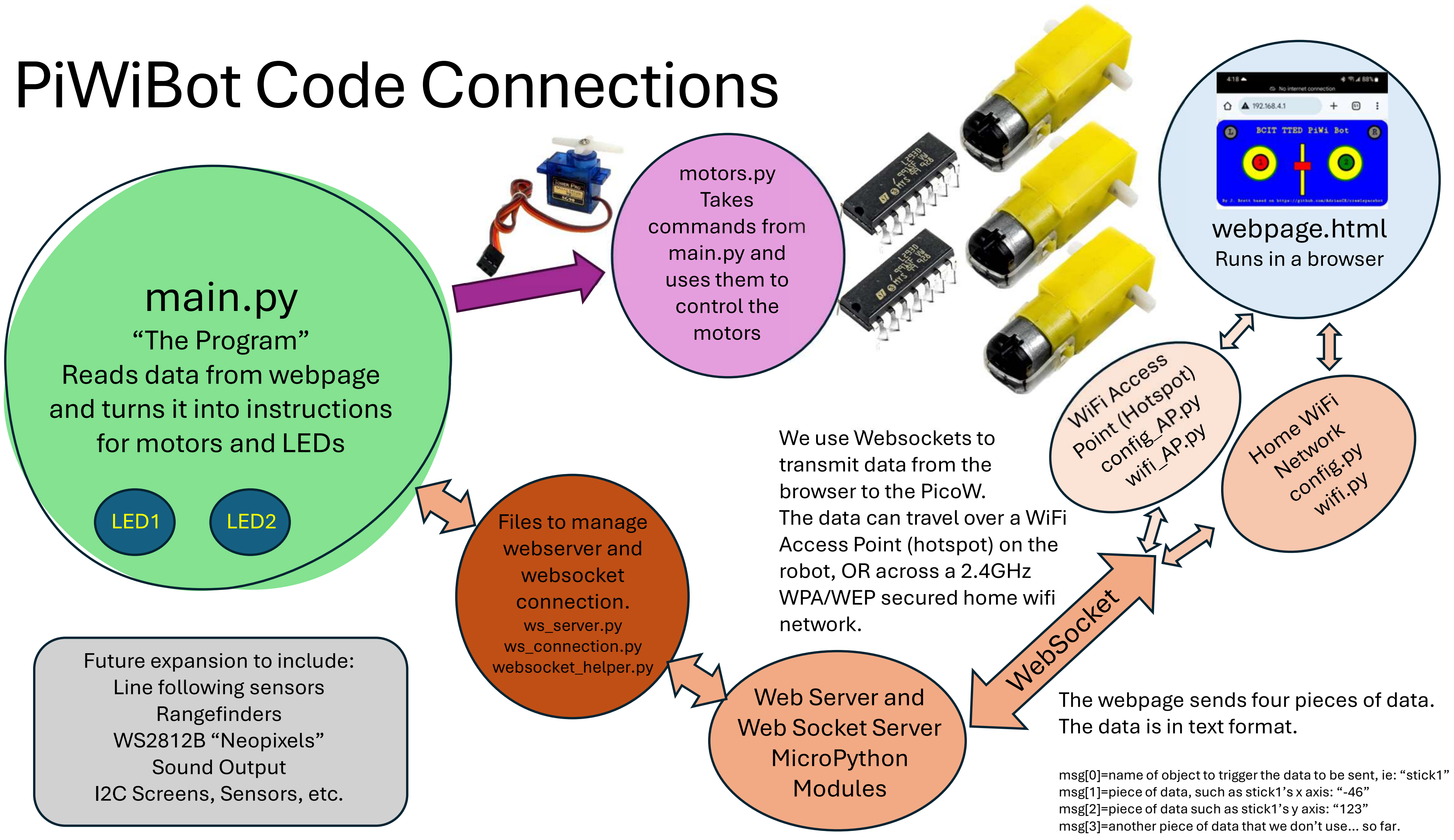
- Micropython:
  - Interpreted language (“Python”)
    - Your microcontroller stores, reads and executes source code
  - The code on the microcontroller is in source code
    - You can edit and change it easily
    - No need to re-compile or re-download
  - Benefit: Interpreted code can be easier to develop
    - Faster chips compensate for slower code

Notable Syntax Differences

"A cartoon image of two computer programmers, one male, one female, fighting with a Python." Generated by Microsoft CoPilot, Feb 2025

- Python does not require semi-colons at end of line
- Python does not use {loop} to manage loops
  - Loops are managed by indentations
    - Indentation really, really matters!
- Python's comment code is "#" rather than "//"
- "print()" includes the line feed and carriage return
  - "println()" not needed
- Python offers several benefits when managing data
  - Not so important in this application but can reduce coding time

PiWiBot Code Connections



main.py
"The Program"
Reads data from webpage and turns it into instructions for motors and LEDs

motors.py
Takes commands from main.py and uses them to control the motors

webpage.html
Runs in a browser

WiFi Access Point (Hotspot)
config_AP.py
wifi_AP.py

Home WiFi Network
config.py
wifi.py

Files to manage webserver and websocket connection.
ws_server.py
ws_connection.py
websocket_helper.py

Web Server and Web Socket Server MicroPython Modules

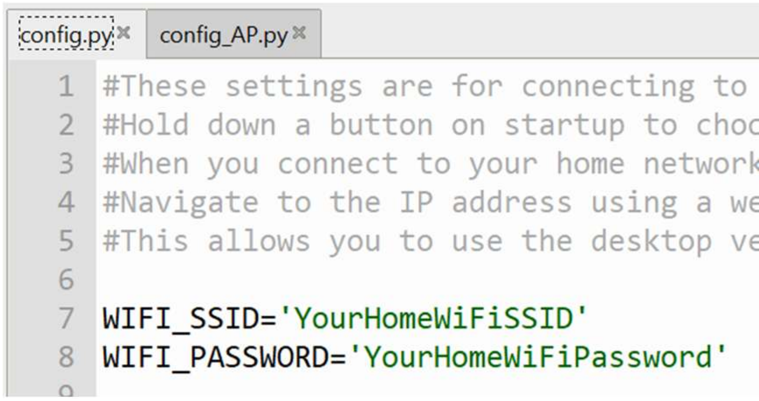
We use Websockets to transmit data from the browser to the PicoW.
The data can travel over a WiFi Access Point (hotspot) on the robot, OR across a 2.4GHz WPA/WEP secured home wifi network.

The webpage sends four pieces of data.
The data is in text format.

msg[0]=name of object to trigger the data to be sent, ie: "stick1"
msg[1]=piece of data, such as stick1's x axis: "-46"
msg[2]=piece of data such as stick1's y axis: "123"
msg[3]=another piece of data that we don't use… so far.

Future expansion to include:
Line following sensors
Rangefinders
WS2812B "Neopixels"
Sound Output
I2C Screens, Sensors, etc.

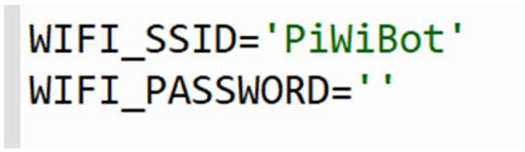
Change WiFi Details



config.py config_AP.py

```
#These settings are for connecting to
#Hold down a button on startup to choo
#When you connect to your home network
#Navigate to the IP address using a we
#This allows you to use the desktop ve

WIFI_SSID='YourHomeWiFiSSID'
WIFI_PASSWORD='YourHomeWiFiPassword'
```

- Config.py
  - The SSID and Password for connecting to your home network
  - Hold down “the button” on the board on startup to connect
- Config_AP.py
  - Change the name of your robot’s network Access Point
    - Seriously, DO this… we don’t want six robot APs all named “PiWiBot”!
    - Add a password if you want.



```
WIFI_SSID='PiWiBot'
WIFI_PASSWORD=''
```

webpage.html

html keywords are enclosed in < >

Page is created in "divisions"
Look for <div> to begin and </div> to end

Graphics are created using svg

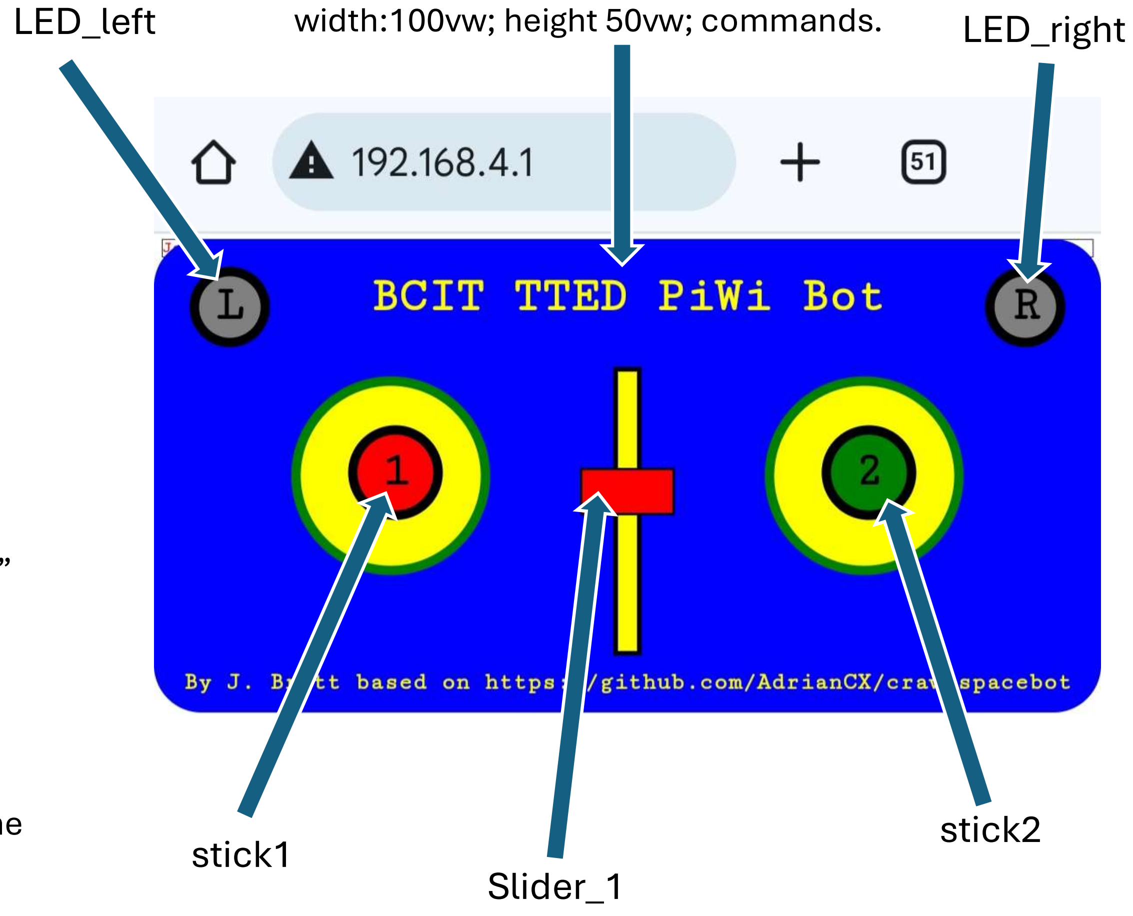
Divisions can be named and manipulated using javascript code

The code watches for "pointerdown" and "pointerup" events… when you touch and release the screen.

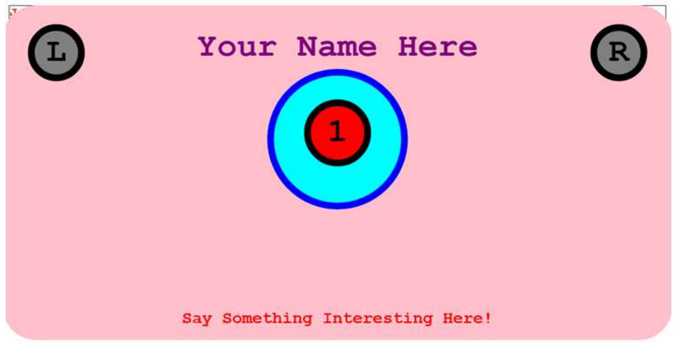
It knows which division you are touching and can animate it appropriately

The code uses the position of your finger to determine what values to return to the server

The background is not a named division because it is not animated. But it is set to scale to the view window using the width:100vw; height 50vw; commands.



What Can I Hack?



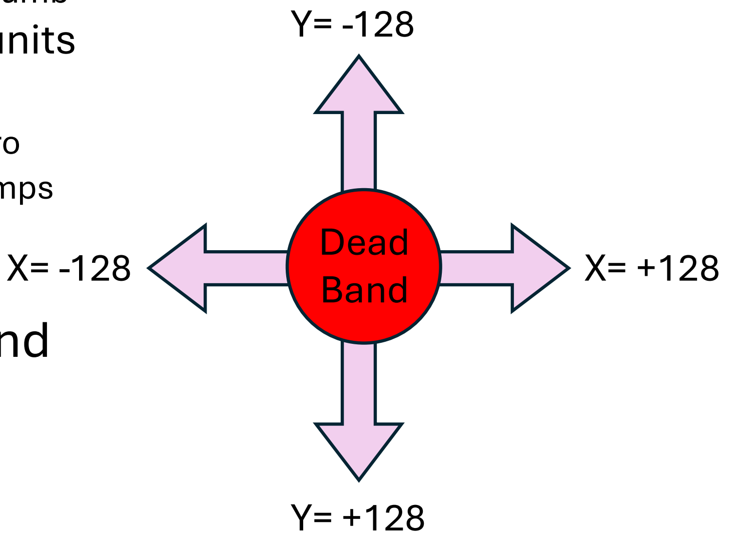
- Lots of easy stuff to change:
  - Look for things between quotes… fill="blue", or r="40"
  - Look for text between <text> and </text> commands
- <text x="200" y="190" font-size="10" text-anchor="middle" fill="Yellow">By J. Brett based on https://github.com/AdrianCX/crawlspacebot</text>

- A bit trickier, but you can remove unused objects
  - Save a "webpage.backup" file in case you need to go back to step one!
  - Delete the section from <div id="stick2" to the next </div>
  - At the end of the file delete the "let joystick2= ….,16);
    - Do the same for Slider1 if you don't need it
      - Now can you re-centre joystick 1 to the middle of the screen?

- Maybe you want one joystick with two sliders?

How Does the Joystick Work?

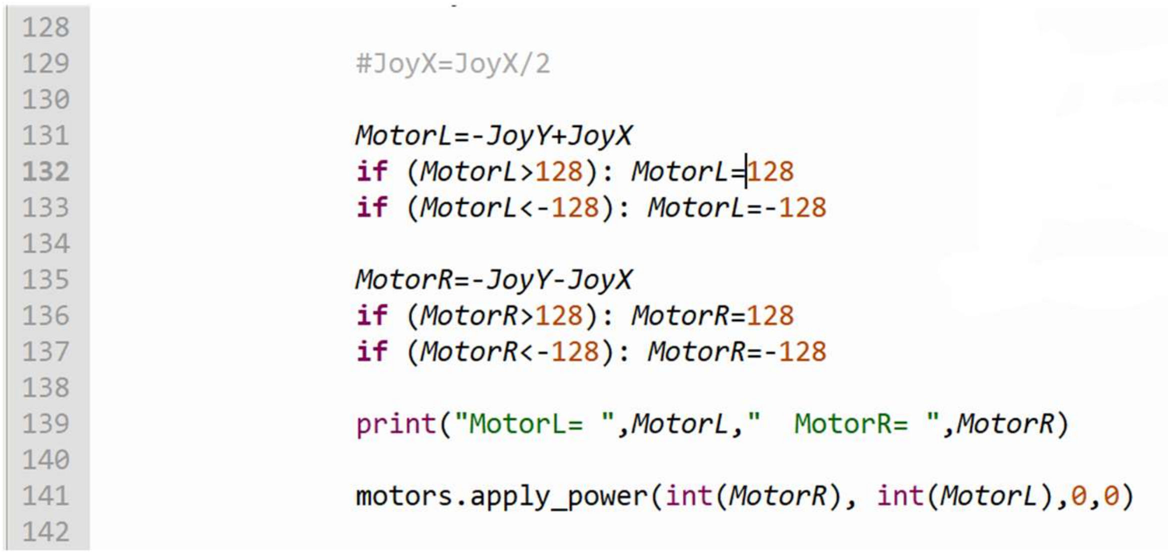
- let joystick1 = new JoystickController("stick1", 128, 16);
  - Creates a new Joystick object
  - Names it “stick1”
  - Associates it with the “stick1” division graphics
    - The round button… it will animate this button as you move your thumb
  - Creates a maximum range of movement of 128 drawing units
  - Creates a “deadband” of 16 drawing units
    - When either X or Y is less than 16 units from centre, it sets it to zero
    - Makes it easier to re-centre the stick, and prevents accidental bumps



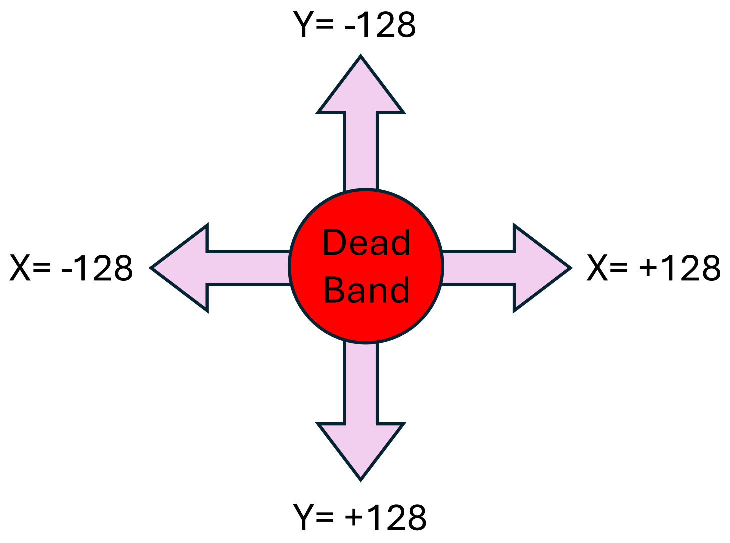
- I was completely stuck on how to do this until I found
  - “CrawlSpaceBot” by Adrian Cruceru on Github
    - I’ve figured most of it out, but not all of it…. yet. ☺

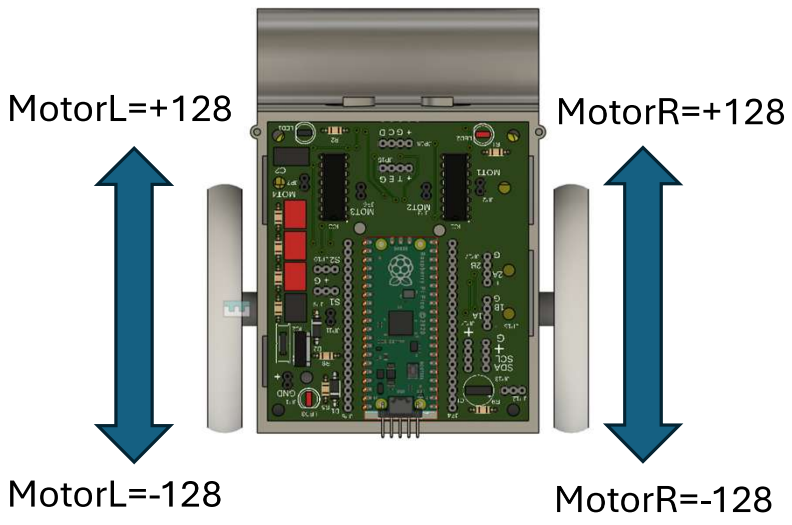
Mixing the Joystick

- You’ve got X,Y coordinates
  - But you need skid steer motor speeds!

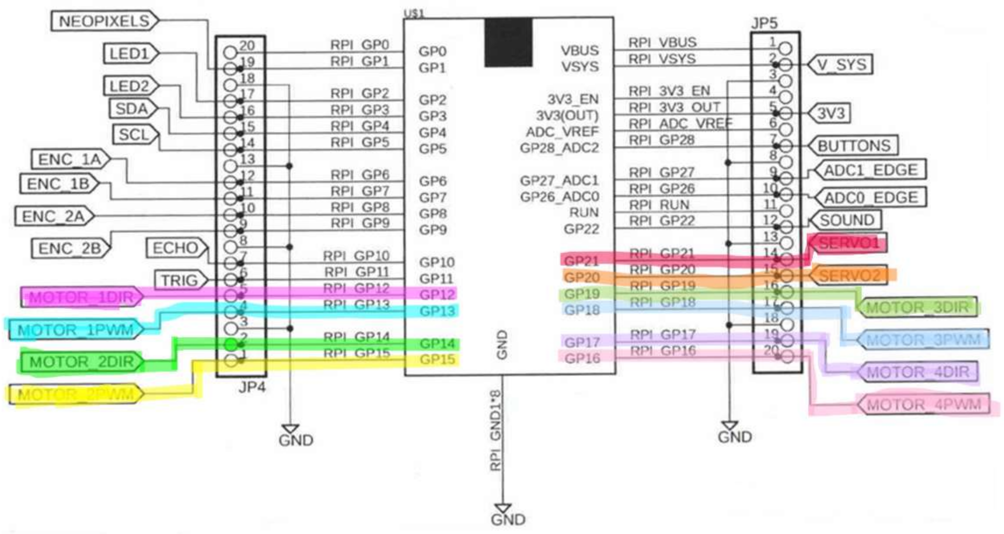


```
            #JoyX=JoyX/2

            MotorL=-JoyY+JoyX
            if (MotorL>128): MotorL=128
            if (MotorL<-128): MotorL=-128

            MotorR=-JoyY-JoyX
            if (MotorR>128): MotorR=128
            if (MotorR<-128): MotorR=-128

            print("MotorL= ",MotorL,"  MotorR= ",MotorR)

            motors.apply_power(int(MotorR), int(MotorL),0,0)
```





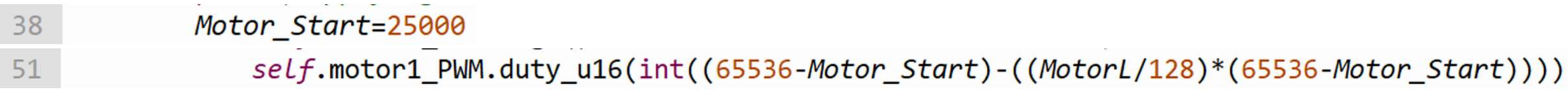
Driving the Motors



- Wiring Determined by Schematic
- Each motor has
  - 1 “DIR” pin to set direction
  - 1 “PWM” pin to set speed
    - If decreasing the duty cycle makes the motor speed up going forward (DIR=HIGH)
    - Then increasing the duty cycle makes the motor speed up going in reverse (DIR=LOW)
- Pico has many PWM options
    - Set the frequency (how often it turns on and off)
      - We use a default of 5000Hz… you can sometimes hear a 5000Hz whining!
    - Set the duty cycle (the fraction of the time that it is turned “on”)
      - This is a fraction of a 16 bit number… 0 is “always off”, 65,535 is “always on”
        - The Arduino only has an 8 bit “analogWrite”… at 980Hz

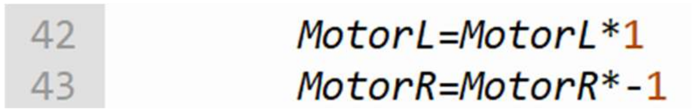
Motors Need a Kickstart

- The friction on the gearbox requires a minimum torque to turn
  - Below this the motor is stalled
  - The code uses a PWM setting of 25,000/65,536 = 38% to start up
    - The code scales the motor response from 38% at low throttle to 100% at full throttle



```
    Motor_Start=25000
        self.motor1_PWM.duty_u16(int((65536-Motor_Start)-((MotorL/128)*(65536-Motor_Start)))
```

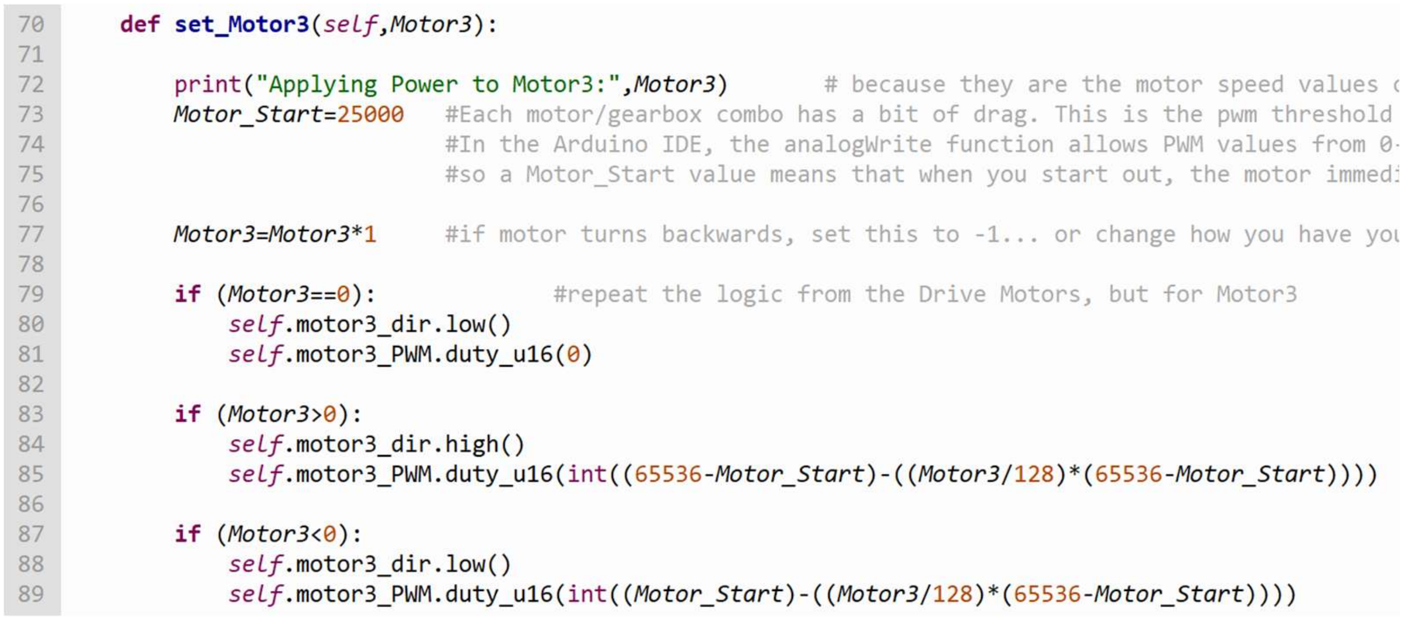
- Often motors are "reversed"
  - Direction can be changed by swapping the wires on the board, or
  - Swapping a "+" to a "-" in code



```
        MotorL=MotorL*1
        MotorR=MotorR*-1
```

Motor3 is On Slider1

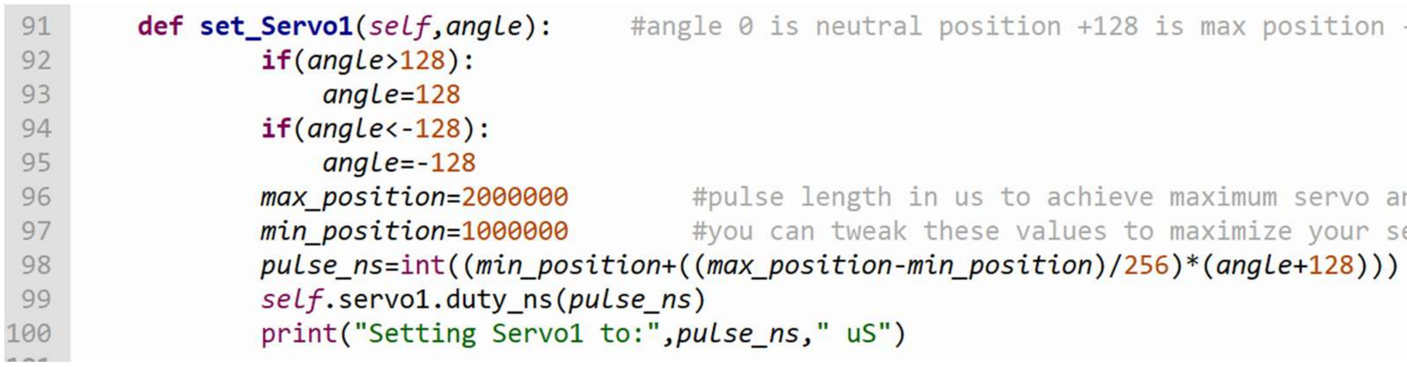
- You can slide the slider to position and leave it there
  - You can take your thumb off the controller and Motor3 keeps moving
    - It’s “return to zero” function has been turned off in the webpage javascript code
  - It won’t stop when you stop the code from Thonny
    - You’ll need to add some lines to shut the motor down automatically
    - Look at lines 168-174 in main.py



```python
    def set_Motor3(self,Motor3):

        print("Applying Power to Motor3:",Motor3)       # because they are the motor speed values 
        Motor_Start=25000   #Each motor/gearbox combo has a bit of drag. This is the pwm threshold
                            #In the Arduino IDE, the analogWrite function allows PWM values from 0
                            #so a Motor_Start value means that when you start out, the motor immed

        Motor3=Motor3*1     #if motor turns backwards, set this to -1... or change how you have yo

        if (Motor3==0):             #repeat the logic from the Drive Motors, but for Motor3
            self.motor3_dir.low()
            self.motor3_PWM.duty_u16(0)

        if (Motor3>0):
            self.motor3_dir.high()
            self.motor3_PWM.duty_u16(int((65536-Motor_Start)-((Motor3/128)*(65536-Motor_Start))))

        if (Motor3<0):
            self.motor3_dir.low()
            self.motor3_PWM.duty_u16(int((Motor_Start)-((Motor3/128)*(65536-Motor_Start))))
```

Servo1 Uses Joystick2’s Y-Axis

- We use a 50Hz PWM signal to send a pulse every 20ms
  - Servos will turn off if they don’t receive regular updates
- Typical pulse lengths for a “hobby servo” are 1000-2000ms
  - Neutral position is 1500ms (angle=0)
    - Positions range from +128 to -128 in our code
  - We set the duty cycle using nanoseconds
    - 1500ms = 1,500,000ns



```
    def set_Servo1(self,angle):     #angle 0 is neutral position +128 is max position 
            if(angle>128):
                angle=128
            if(angle<-128):
                angle=-128
            max_position=2000000        #pulse length in us to achieve maximum servo ar
            min_position=1000000        #you can tweak these values to maximize your se
            pulse_ns=int((min_position+((max_position-min_position)/256)*(angle+128)))
            self.servo1.duty_ns(pulse_ns)
            print("Setting Servo1 to:",pulse_ns," uS")
```