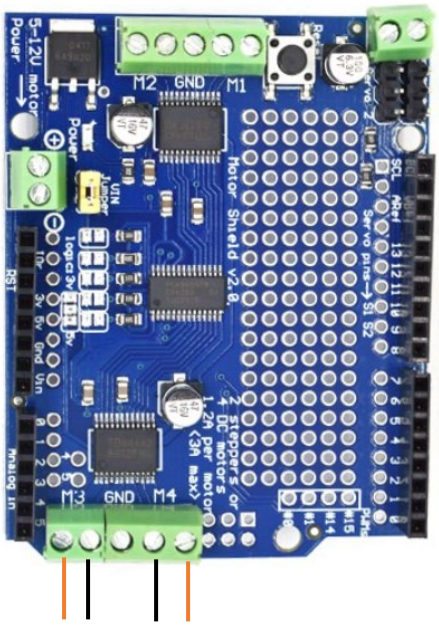
Line Tracker Vehicle

Metro M4 with Motors/Tracker Sensor Board

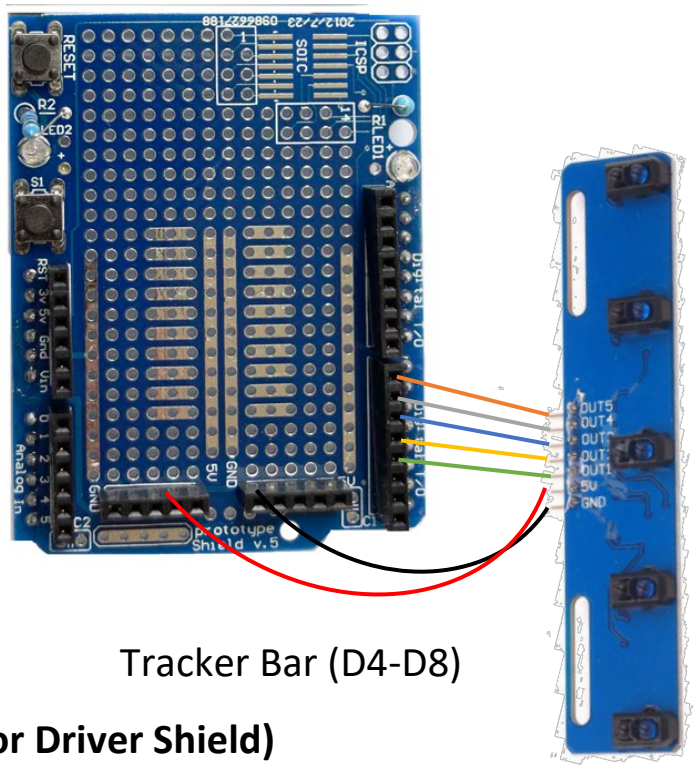
Barry Butler
bbutl58@eq.edu.au

A. Metro M4 Connections

Connect the servo's and ultrasonic sensor to the CPX as shown.

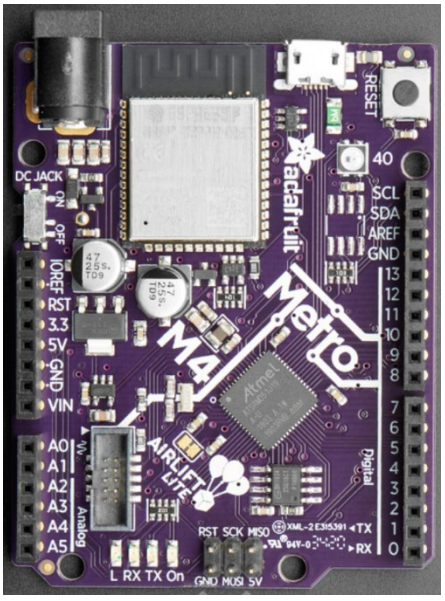


Motors



Tracker Bar (D4-D8)

Metro M4 Connections (with Motor Driver Shield)

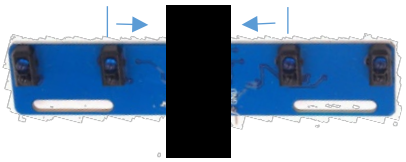
A0				
A1			LED	
A2			L (red)	
A3				
A4 (SCL)			SCL	
A5 (SDA)			SDA	
Motor 1			D13	On/off switch
Motor 2			D12	
Motor 3	TT Left		D11	
Motor 4	TT Right		D10	
			D9	
I2C 1			D8	Tracker Out 5 (Far Right)
I2C 2			D7	Tracker Out 4 (Right)
I2C 3			D6	Tracker Out 3 (Centre)
I2C 4			D5	Tracker Out 2 (Left)
			D4	Tracker Out 1 (Far Left)
			D3	
			D2	
			D1	
			D0	

Coolum State High School
Care • Respect • Excellence

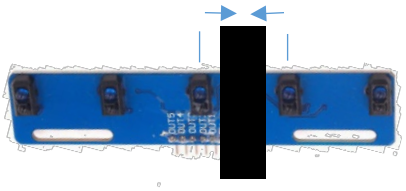
ROBOCOAST
Sunshine Coast Robotics
www.robocoast.tech

The Tracker Bar can be used in at least two ways (black line on white surface). The sensors must be within about 2 mm from the surface.

1. Use the left and right sensors to track the black line. Use the centre sensor to adjust the direction, as a turn brings the vehicle back from the left or right sensors. Probably best for wider black lines.



2. Use the centre and right sensors to track the black line. When both sensors are on white, the line is between them. When the left sensor shows black, turn to the right. When the right sensor shows black, turn to the left.



The two outer sensors are used to check for junctions.

B. CircuitPython Basic Code

The following code is used for all CircuitPython Code. Type this code into the Mu editor (in **CircuitPython mode**).

```
#IMPORTS----------------------------------------


#CONNECTIONS------------------------------------


#GLOBAL VARIABLES-------------------------------


#FUNCTIONS--------------------------------------


#PRESS BUTTON TO START--------------------------


#MAIN LOOP--------------------------------------------
while True:


```

EACH TIME YOU ENTER A SECTION OF CODE, CLICK THE "SAVE" BUTTON TO TEST IT, AND FIX ANY ERRORS.

- **Select the CIRCUITPY Folder**
- **Always Save Using the Filename – code.py**

C. Import the Required Libraries

We need to import the MotorKit, Neopixel, Digital Input libraries.

```
#IMPORTS----------------------------------------
import time
from adafruit_motorkit import MotorKit
import board
import neopixel
from digitalio import DigitalInOut, Direction, Pull
```

D. Setup All the Connections

Now we need to setup the connections between the motors / sensors and the Metro M4 pins. We have two sensors – a switch and an ultrasonic sensor. We can change the colour of the neopixel (led) to tell us things that are going on when we run the vehicle.

```
#CONNECTIONS---------------------------------------
switch = DigitalInOut(board.D13)
switch.direction = Direction.INPUT
switch.pull = Pull.DOWN

kit = MotorKit()

led = neopixel.NeoPixel(board.NEOPIXEL, 1)
led[0] = (255, 0, 0)

T4 = DigitalInOut(board.D4)
T4.direction = Direction.INPUT
T5 = DigitalInOut(board.D5)
T5.direction = Direction.INPUT
T6 = DigitalInOut(board.D6)
T6.direction = Direction.INPUT
T7 = DigitalInOut(board.D7)
T7.direction = Direction.INPUT
T8 = DigitalInOut(board.D8)
T8.direction = Direction.INPUT
```

E. Global Variables

We need a couple of global variables, to store the vehicle motor state (on or off) and the Tracker Bar values.

```
#GLOBAL VARIABLES----------------------------------
vehicle_on = False
TFL = not T4.value
TL = not T5.value
TC = not T6.value
TR = not T7.value
TFR = not T8.value
TAny = TFL or TL or TC or TR or TFR
```

F. Create a Function to Change the Speed of Both Motors

Because we are running 2 motors, it simplifies the code by creating a function to control both motors at the same time.

```
#FUNCTIONS-----------------------------------------
def run_motors(s1, s2):
    kit.motor3.throttle = s1
    kit.motor4.throttle = s2
    if (s1==0) and (s2==0): led[0] = (255, 0, 0)
    else: led[0] = (0, 255, 0)
```

G. Create Functions to Read all the Sensors

The best way to run the vehicle is to read all the sensors for every iteration of the main loop.

This function checks if the button switch has been pressed so we can stop the motors if required. Pressing the button toggles the motors on again.

```
def read_all_sensors(show_all):
    global vehicle_on, TFL, TL, TC, TR, TFR, TAny

    if switch.value:
        vehicle_on = not vehicle_on
        if vehicle_on: print('Running')
        else: print('Stopped')
        time.sleep(0.1)

    TFL = T4.value
    TL = T5.value
    TC = T6.value
    TR = T7.value
    TFR = T8.value
    TAny = TFL or TL or TC or TR or TFR
    if show_all: print(TFL, TL, TC, TR, TFR)
```

H. Wait for the Button to be Pressed to turn the Vehicle On

We don't want the motors to start running as soon as we download the code the Metro M4. So, write the code to check if the button has been pressed.

```
#PRESS BUTTON TO START----------------------------
print('Waiting to start')
while not switch.value:
    time.sleep(0.1)
vehicle_on = True
```

I. Make the Vehicle Move and Avoid Obstacles

The main loop firstly reads all the sensor values. If the vehicle motors are on, the code then checks the ultrasonic sensor distance. If the distance is less than 10 cm the motors stop, reverse and turn to point in a new direction.

```
#MAIN LOOP-----------------------------------------
print('Running')
while True:
    read_all_sensors(True)

    if vehicle_on:
        if TC:
            run_motors(0.6,0.6)    #on line --> straight ahead
        elif TL:
            run_motors(0.6,0.4)    #off to left --> turn right
        elif TR:
            run_motors(0.4,0.6)    #off to right --> turn left
        elif TFL:
            run_motors(0.5,0)     #left junction --> turn hard left
        elif TFR:
            run_motors(0,0.5)     #right junction --> turn hard right
    else:
        run_motors(0,0)
```