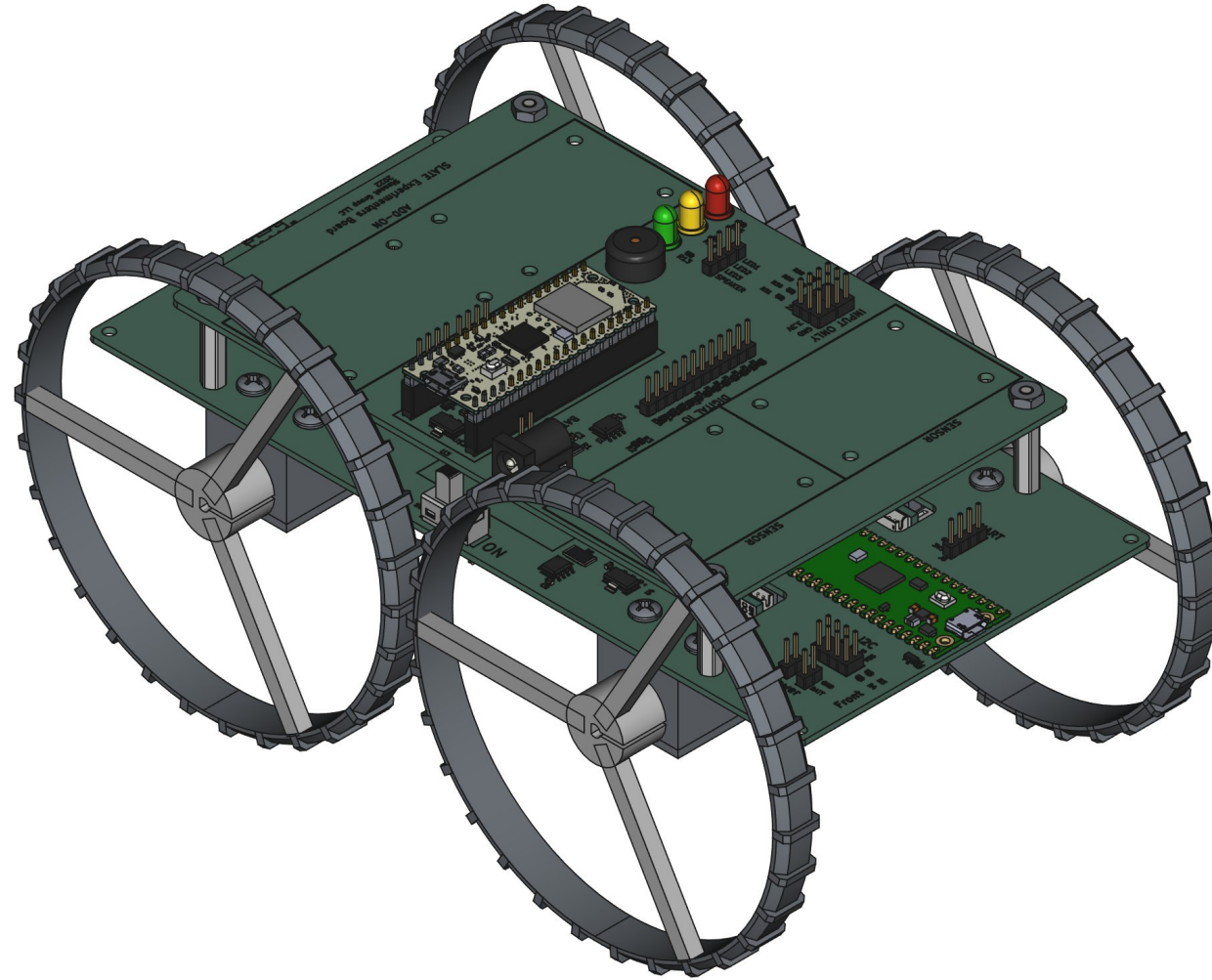
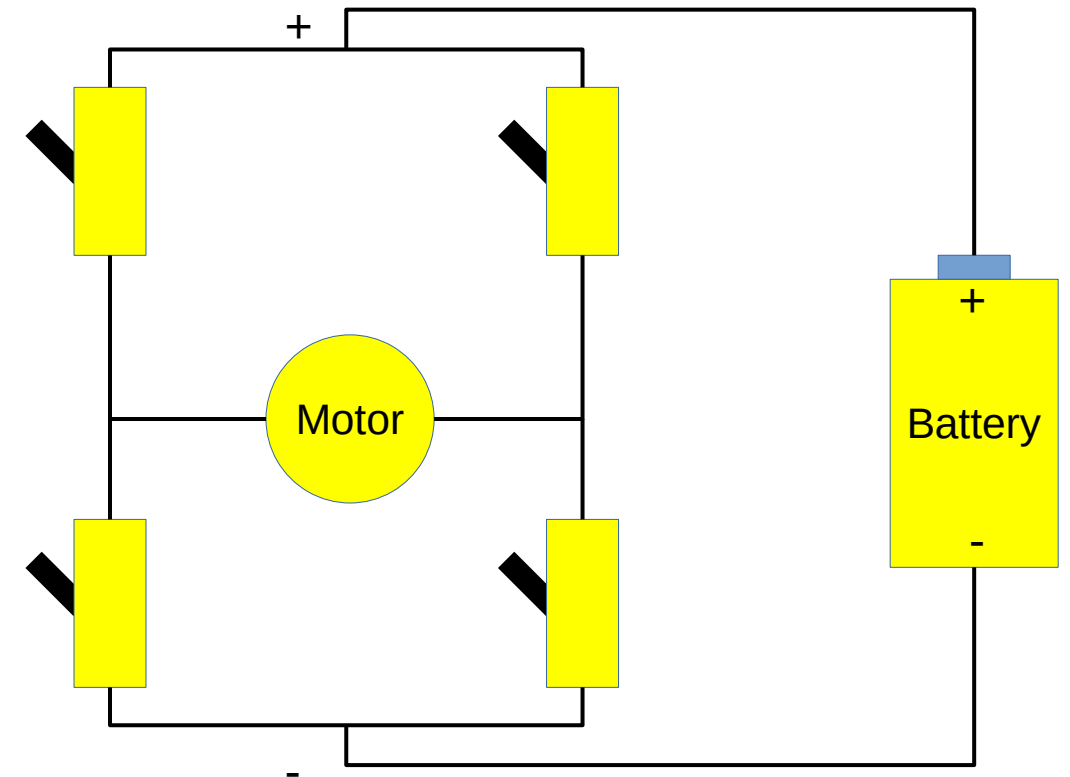
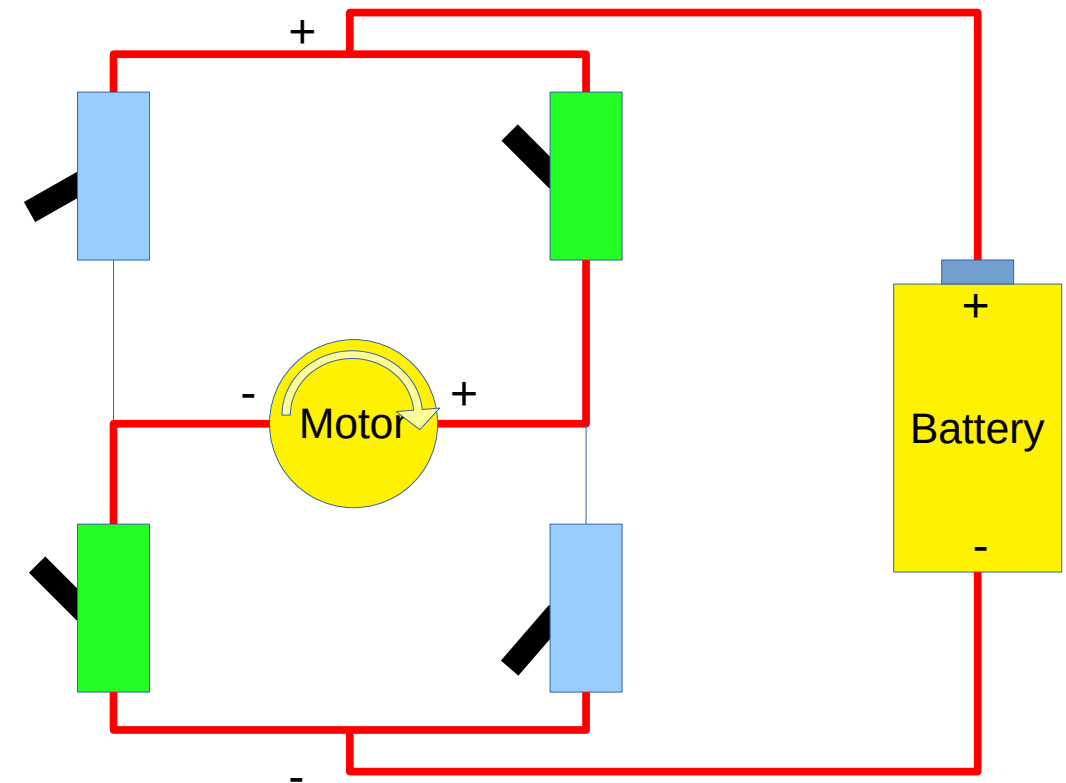




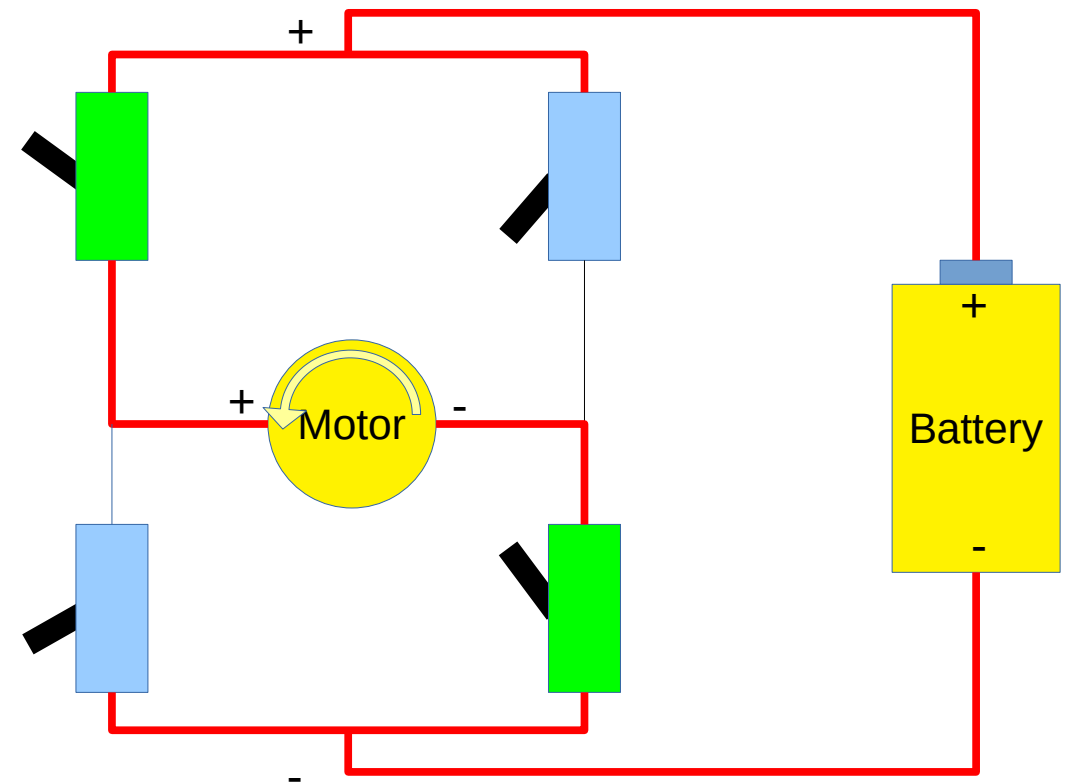
- Controlling the motors is the same as controlling the LED except two signals are needed.
- With two signals, you can control the direction of the motors and turn them on and off.
- The following pages will describe how to hook up the motors.
- A motor driver module is needed. This module allows a computer to control the motors. The motors require more power than the computer signals can provide so the module provides the power.
- The motor driver uses what is called an H-Bridge Driver.
- Dual H-Bridge Driver is used to control the motors. It uses four transistors to control the polarity of the voltage supplied to the motor. The transistors are used as switches turning on and off. Below shows the H-bridge driver circuit and the current flows.



- To make the motor turn on one direction, two switches need to be turned on to let power get to the motor. One switch connects the positive side of the battery to one side of the motor and another switch connects the negative side to the other side of the motor.

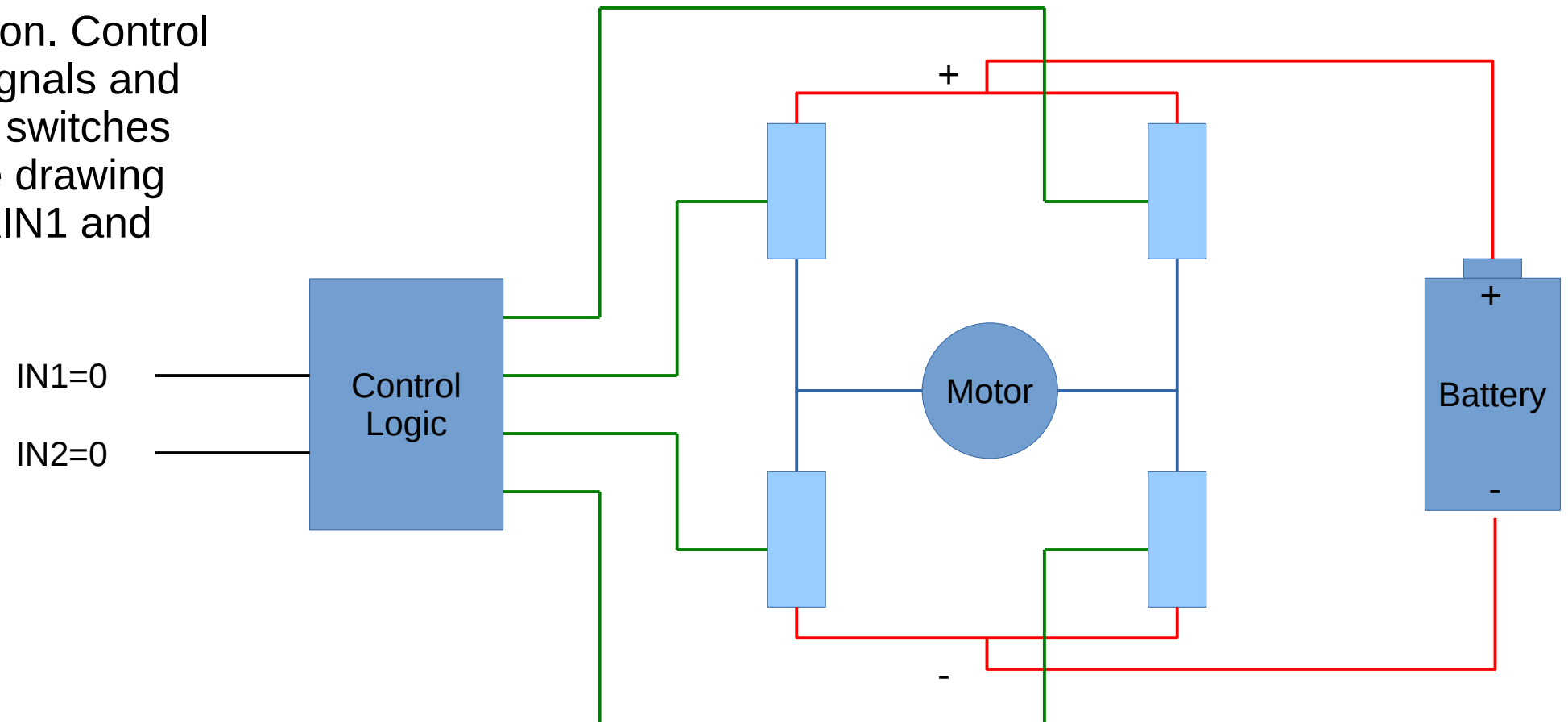


- Flip all the switches to the opposite position and the motor turns in reverse. Notice the polarity signs on the motor switched sides.

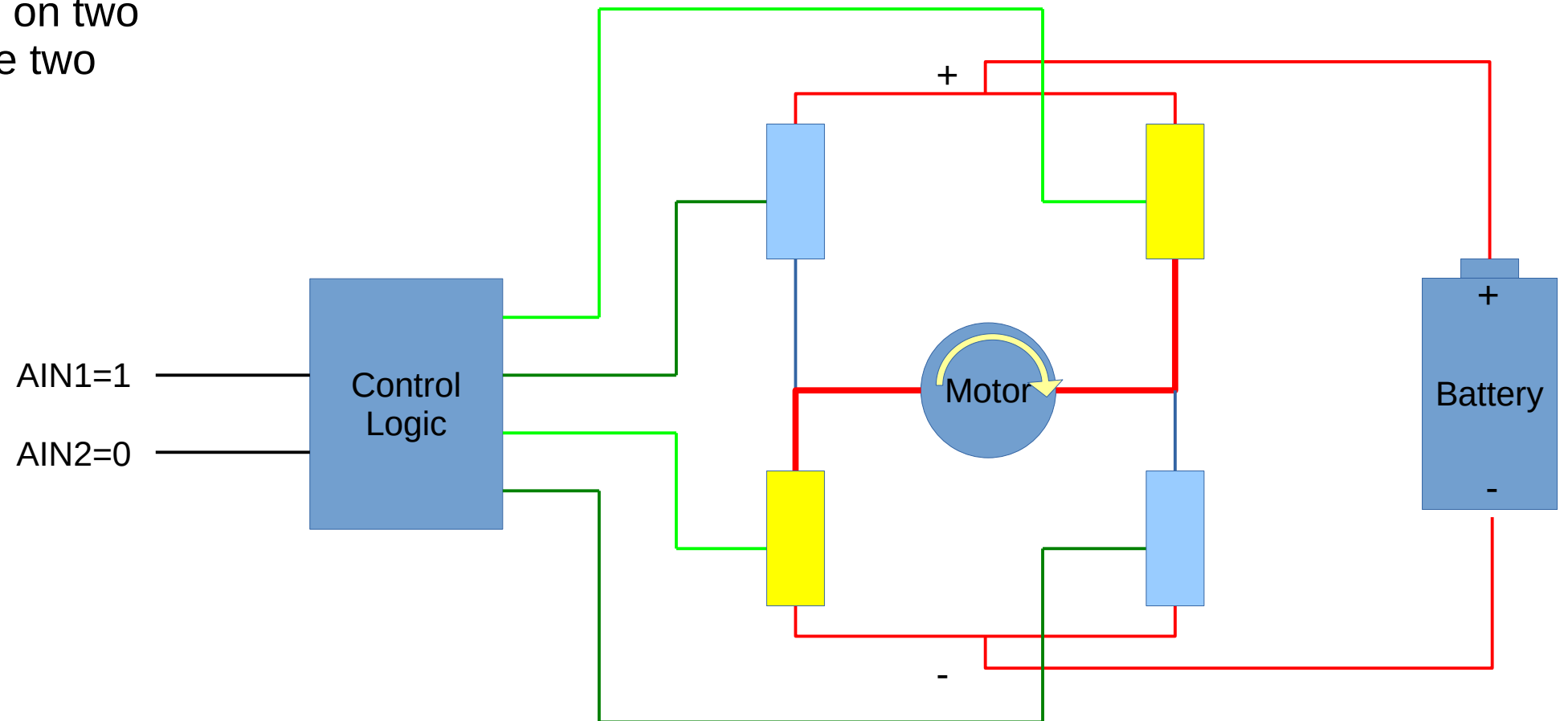


# H-Bridge Driver

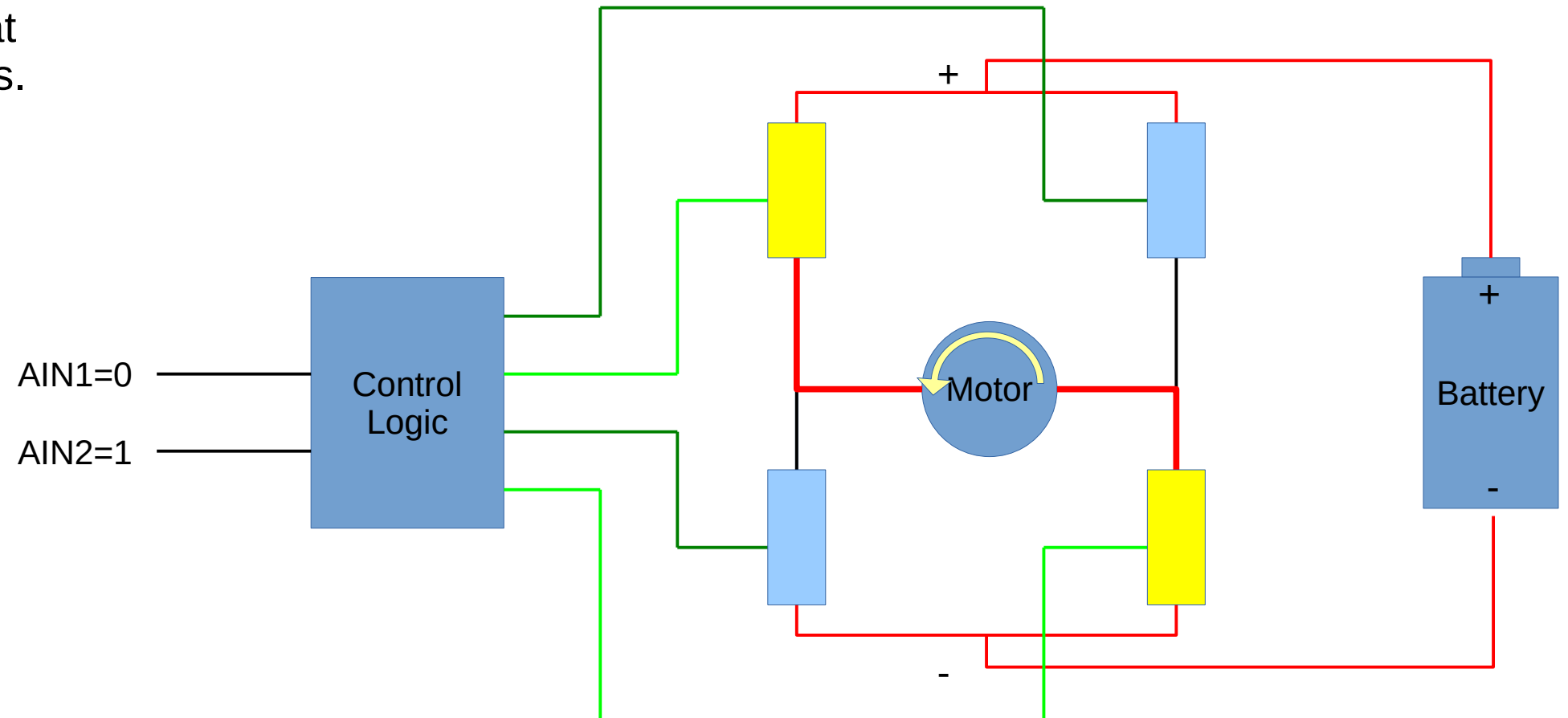
- This drawing shows how the H-Bridge driver works. Only one is shown.
- There are two signals that control the direction and operation. Control logic decodes the two signals and turns on the appropriate switches to control the motor. The drawing shows the condition of AIN1 and AIN2 set to logic zero.



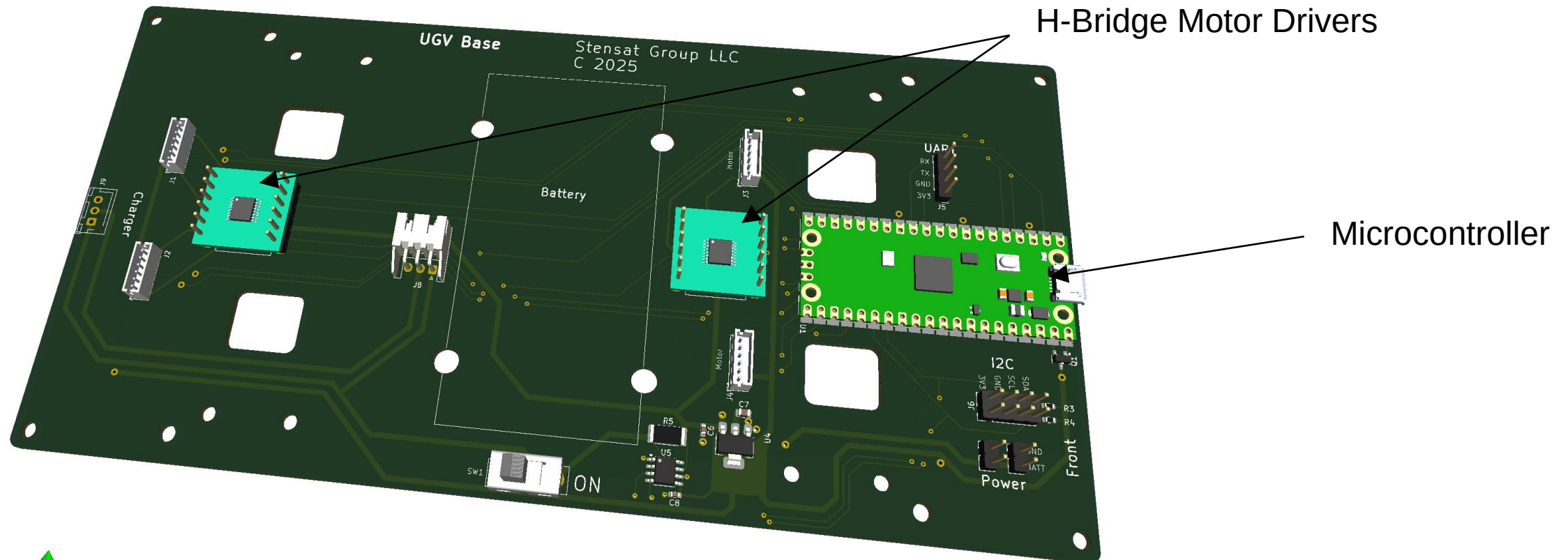
- When AIN1 is set to logic 1, the motor drives in the forward direction.
- You will notice that setting AIN1 = 1, and AIN2=0 turns on two signals that turn on the two switches.



- When AIN2 is set to logic 1, the motor drives in the reverse direction.
- You will notice that setting AIN1 = 0, and AIN2=1 turns on two signals that turn on the two switches.



- The UGV board includes a microcontroller and four motor drivers.
- The microcontroller operates as an I2C device that can accept commands to control the speed and direction of the motors. It also tracks the encoders and provides a count for each wheel to track the rotations of the motors.



# Controlling Motors

- Controlling the motor is done through the UGV board microcontroller which operates like an I2C device. It has an I2C address of 0x19.
- One 4-byte write to the microcontroller is required to set the speed of the four motors.
- Each byte controls the speed and direction. Forward direction has a range of 0 to 127. Reverse is 0 to -127.
- We will create a python module in Thonny to control the motors. Name the program **rover5.py**.
- The code is to the right and uses a python module called **struct**.
- The module will allow the program to pack values into a specific binary format.

```
import struct

class ugv:
    def __init__(self, bus):
        self.i2c = bus

    def get_encoders(self):
        m = self.i2c.readfrom(25, 16)
        me = struct.unpack('<llll', m)
        return(me)

    def drive(self, lf, lr, rf, rr):
        v = struct.pack('bbbb', lf, lr, rf, rr)
        l = self.i2c.writeto_mem(0x19, 0, v)
```

Robot Program

# Controlling Motors

- We will create a class object called `ugv`. The `__init__()` function copies the I2C bus object to **`self.i2c`** so it can be used by the other functions.
- We will ignore the **`get_encoder()`** function for now but do include it in the module.
- The **`drive()`** function will take the speed and direction values for each wheel. The order of the values is important. The order is left front wheel, left rear wheel, right front wheel, and right rear wheel.
- The **`struct.pack()`** function converts the four values into four bytes. The **`'bbbb'`** indicates the format. **`b`** represents byte, an 8-bit value, and four of them will generate four bytes and store them in variable **`v`**.
- The last line writes the four bytes to the microcontroller. The first argument is the I2C address. The second argument is the address location to write the bytes and the last argument is the variable to send.
- Save the code to **`ugv.py`** in on the SLATE board in Thonny.

```
import struct

class ugv:
    def __init__(self, bus):
        self.i2c = bus

    def get_encoders(self):
        m = self.i2c.readfrom(25, 16)
        me = struct.unpack('<llll', m)
        return(me)

    def drive(self, lf, lr, rf, rr):
        v = struct.pack('bbbb', lf, lr, rf, rr)
        l = self.i2c.writeto_mem(0x19, 0, v)
```

Robot Program

# Using Motor Functions

- Create a new program and enter the program.
- Save it as **rover5a.py**
- Turn the power on the UGV base board.
- Run the program on the robot.
- On the laptop, connect to the robot and run the trigger.py program.
- The robot should start moving forward, then reverse, then turn in two directions.

```
from machine import Pin, SoftI2C
import ugv
import time
import connect

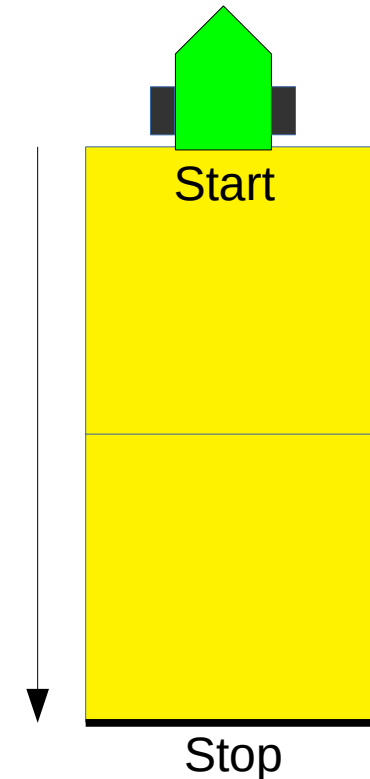
i2c = SoftI2C(sda=Pin(4), scl=Pin(5))
drv = ugv.ugv(i2c)
s, addr = connect.connect()

while 1:
    drv.drive(100, 100, 100, 100)
    time.sleep(2)
    drv.drive(-100, -100, -100, -100)
    time.sleep(.5)
    drv.drive(100, 100, -100, -100)
    time.sleep(.8)
    drv.drive(-100, -100, 100, 100)
    time.sleep(2)
```

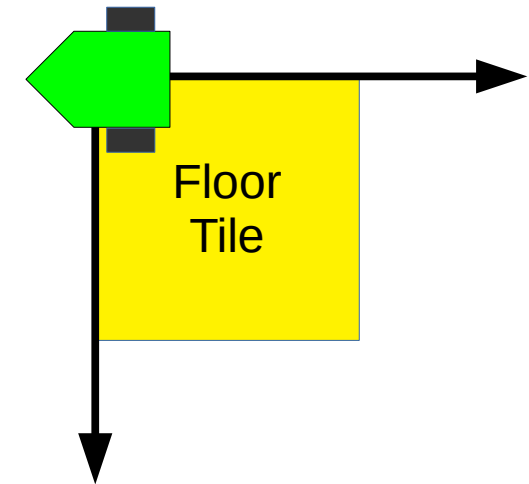
Robot Program

# Calibrating Travel Distance

- Since there is no feedback on the motors to detect distance or wheel rotation, time will be used to specify the distance and the amount of turning.
- Mark off two feet on the floor. Floor tile is usually 1 foot square.
- Write a program to move forward two feet and stop. Start with a delay of 1 second.
- Adjust the delay until the robot travels two feet. Keep this value.
- If necessary, adjust the PWM values to keep the robot as straight as possible and slow it down.

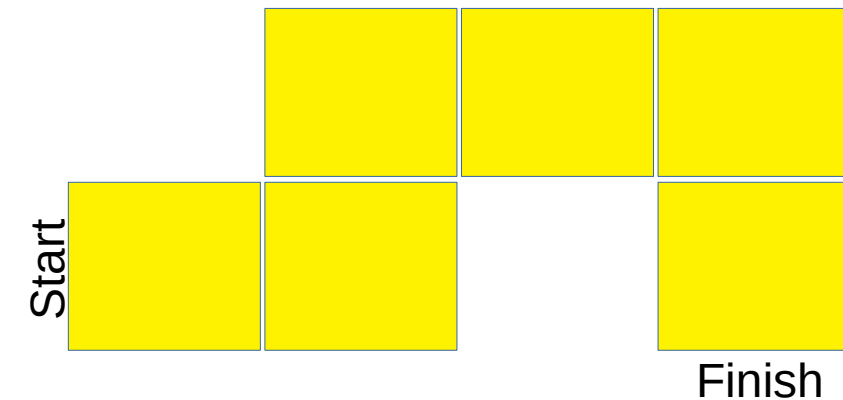


- Now mark on the floor a right angle. If the floor has tiles, use the corner of a tile for your right angle.
- Program the robot to turn right and set the delay to 400 ms and turn off.
- Place the robot on the corner of the right angle facing the left line.
- See how much the robot turns and adjust the delay until it turns 90 degrees.
- Verify the value turning left and adjust if necessary.



# Obstacle Course Time

- Now for the fun part. Modify and expand the program to go through the obstacle course shown. The large square represent 2 foot grids.
- This attempt through the course is called dead reckoning. Write a program to go through a sequence of motions to reach the end of the course. Adjust the delays between each motion so the robot travels to the finish line without going on off the squares.
- Change the course and make it more complex if you want.



- If you noticed, the robot does not drive straight. This is because no two motors are identical. This is true for all types of motors.
- There is a way to compensate. Each motor can be controlled individually.
- Adjust the left and right side motor speeds to try making them go straight.

