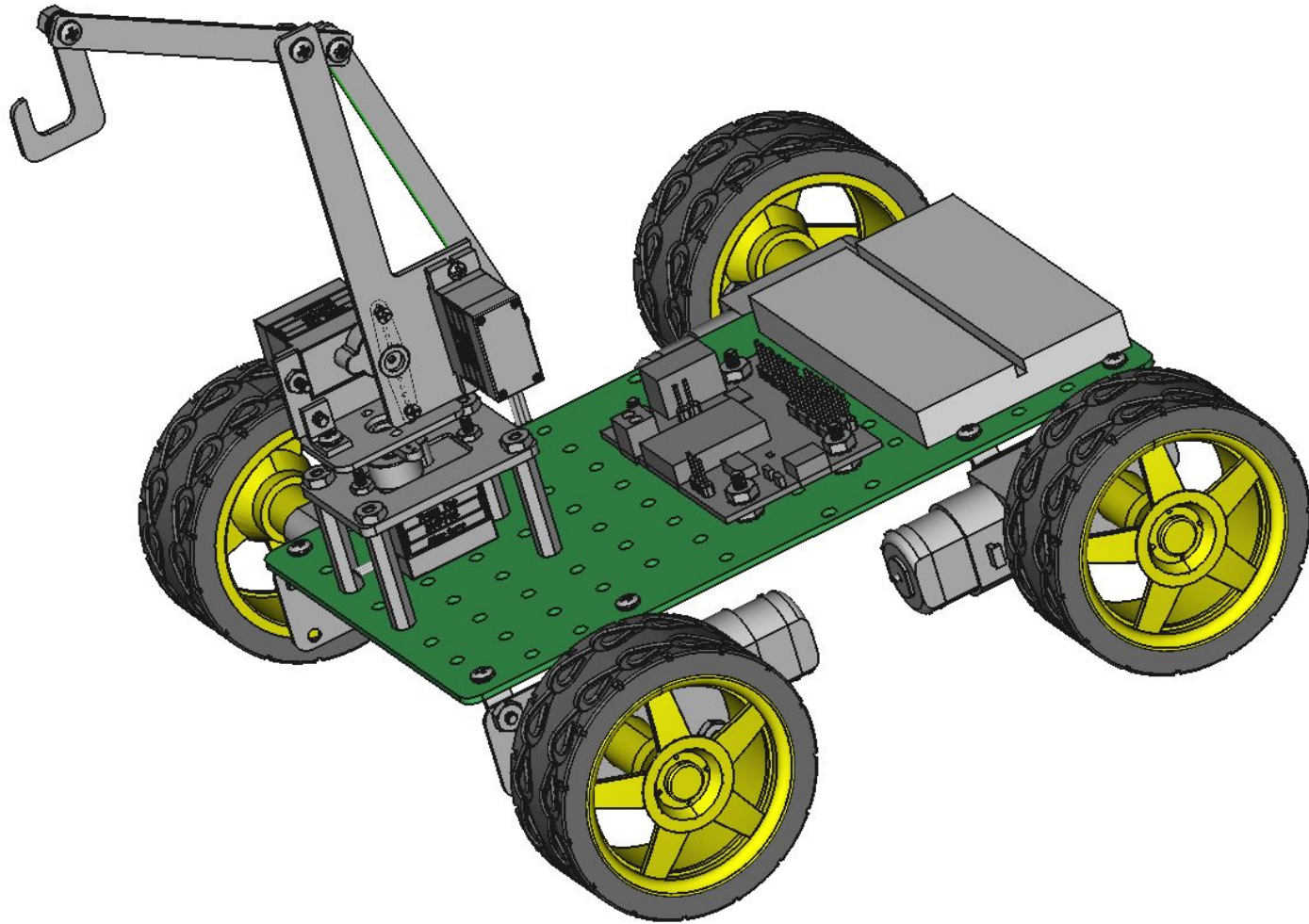


StenBOT Robot Kit



Legal Stuff

- Stensat Group LLC assumes no responsibility and/or liability for the use of the kit and documentation.
- There is a 90 day warranty for the Quad-Bot kit against component defects. Damage caused by the user or owner is not covered.
 - Warranty does not cover such things as over tightening nuts on standoffs to the point of breaking off the standoff threads, breaking wires off the motors, causing shorts to damage components, powering the motor driver backwards, plugging the power input into an AC outlet, applying more than 9 volts to the power input, dropping the kit, kicking the kit, throwing the kit in fits of rage, unforeseen damage caused by the user/owner or any other method of destruction.
- If you do cause damage, we can sell you replacement parts or you can get most replacement parts from online hardware distributors.
- This document can be copied and printed and used by individuals who bought the kit, classroom use, summer camp use, and anywhere the kit is used. Stealing and using this document for profit is not allowed.
- If you need to contact us, go to www.stensat.org and click on contact us.

References

- www.arduino.cc
- <http://esp8266.github.io/Arduino/versions/2.1.0/doc/reference.html>

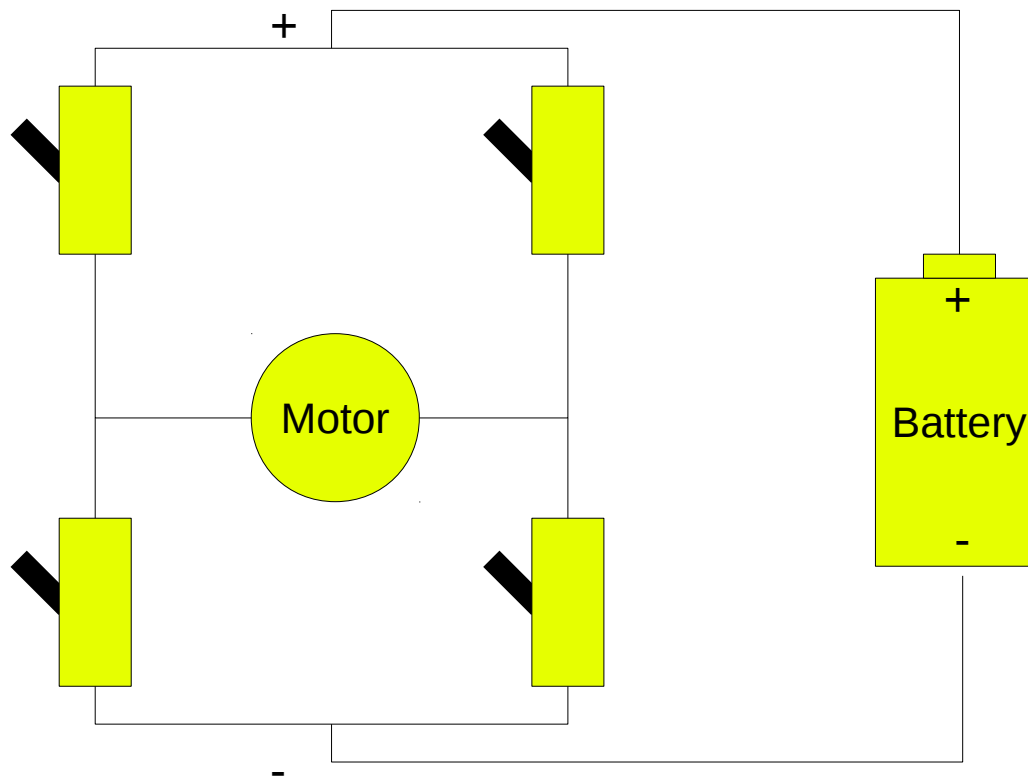


Motor Control

- 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.

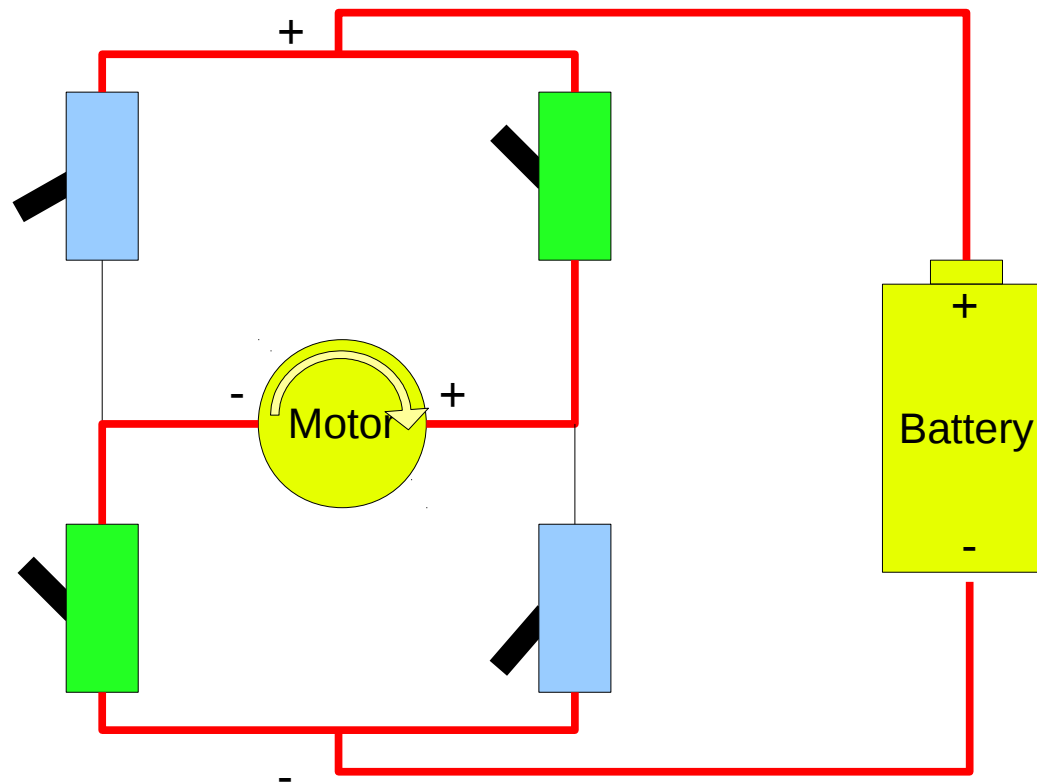
Motor Control

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.



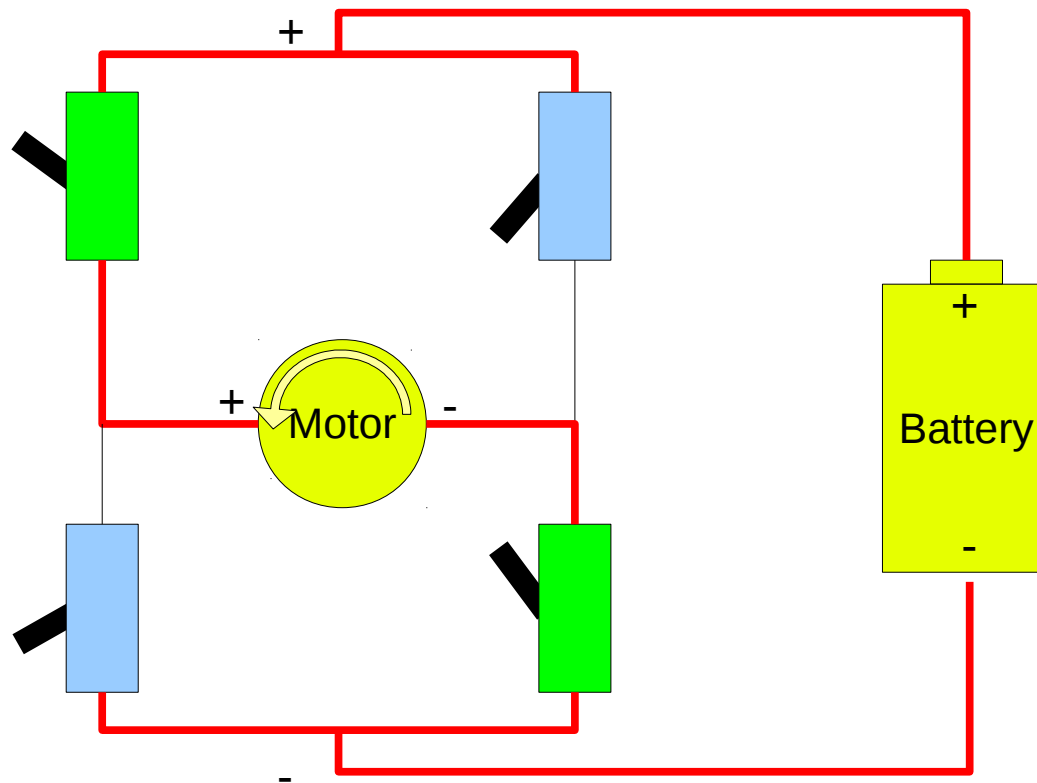
Motor Control

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 to one side of the motor and another switch connects the negative side to the other side of the motor.



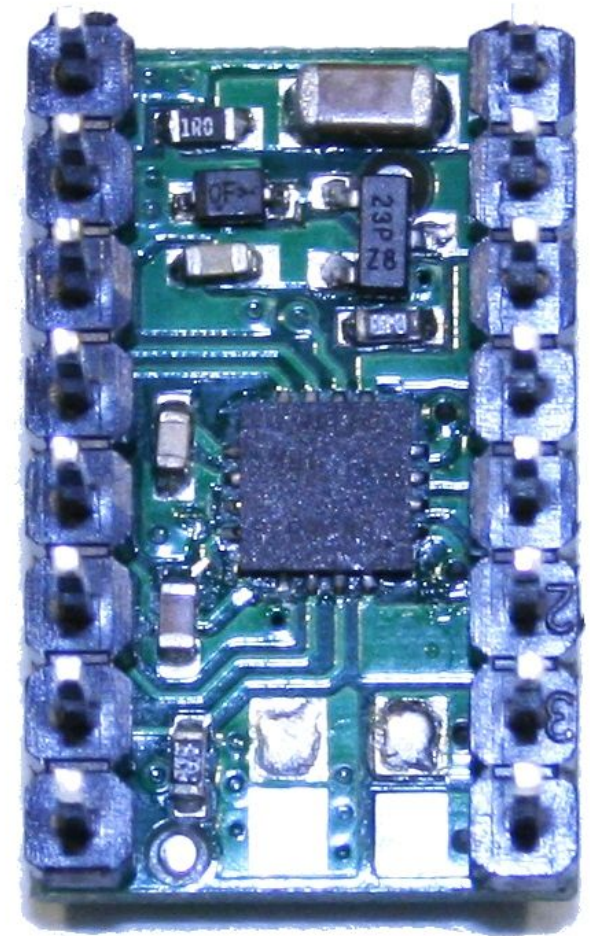
Motor Control

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



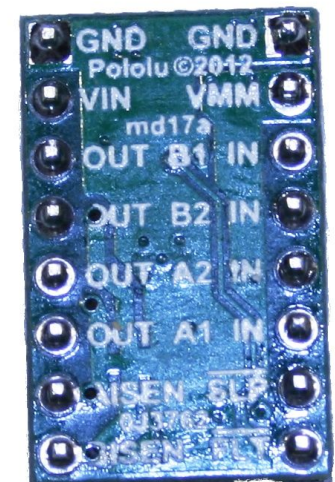
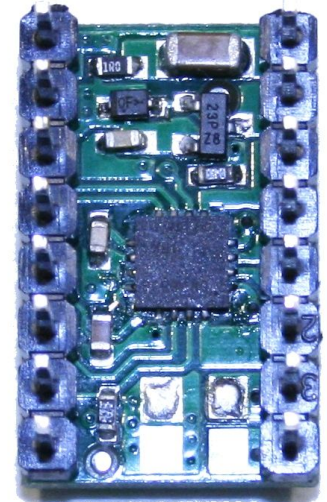
Motor Controller

- The motor controller is the interface between the motors and the processor board. It has circuitry to allow control of the motors and can handle the high currents required to operate the motors. The processor board cannot directly power the motors. The controller is capable of providing the needed current and is used as the interface.



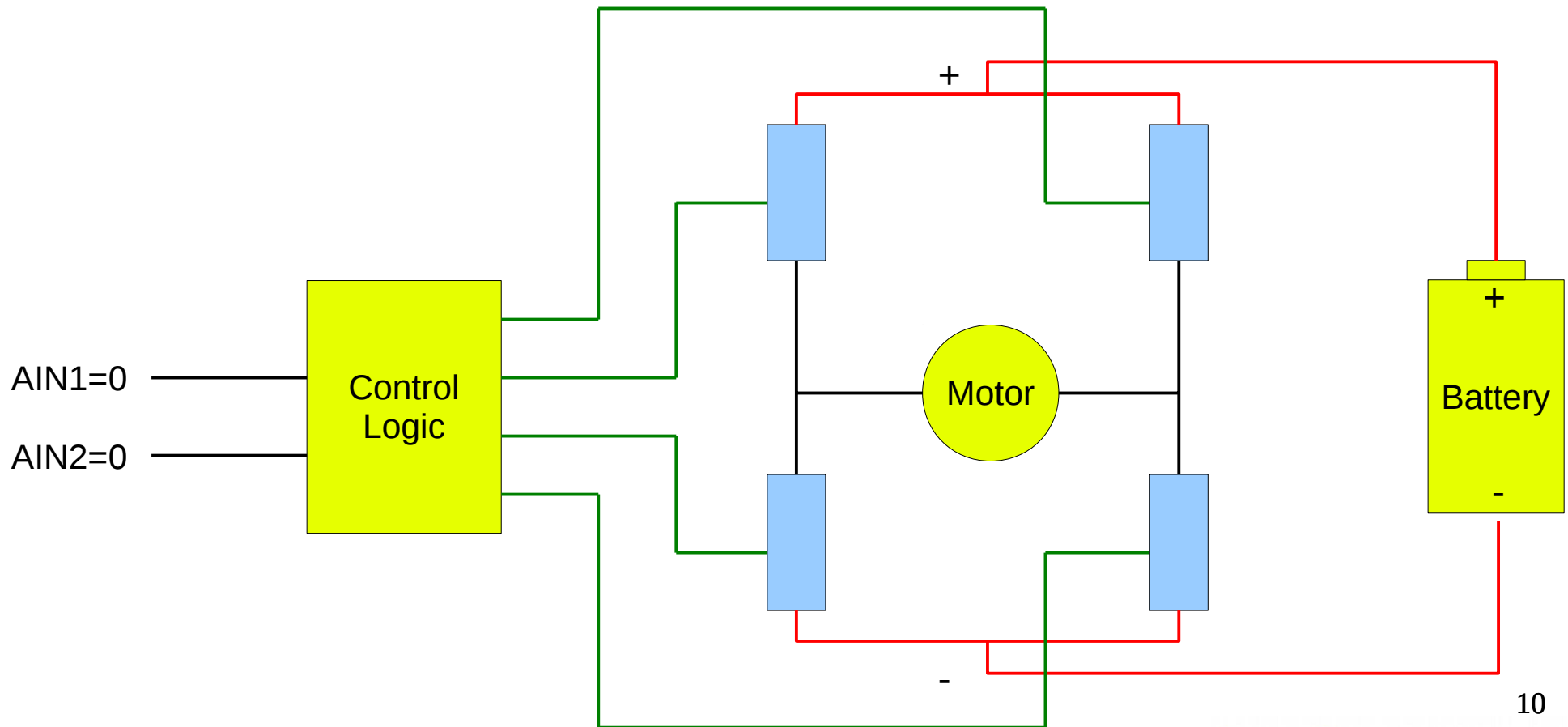
H-Bridge Driver

- The motor controller module consists of two H-bridge drivers to control two motors.
- The circuit side is shown at the top right. The square block in the center contains the two motor drivers.
- The bottom picture shows the signal names next to the pins.
- Power is supplied at pins GND and VIN.
- Control signals for each motor is A1 IN, A2 IN, and B1 IN, B2 IN.
- The motors connect to the pins marked OUT.
- The other pins are not used.



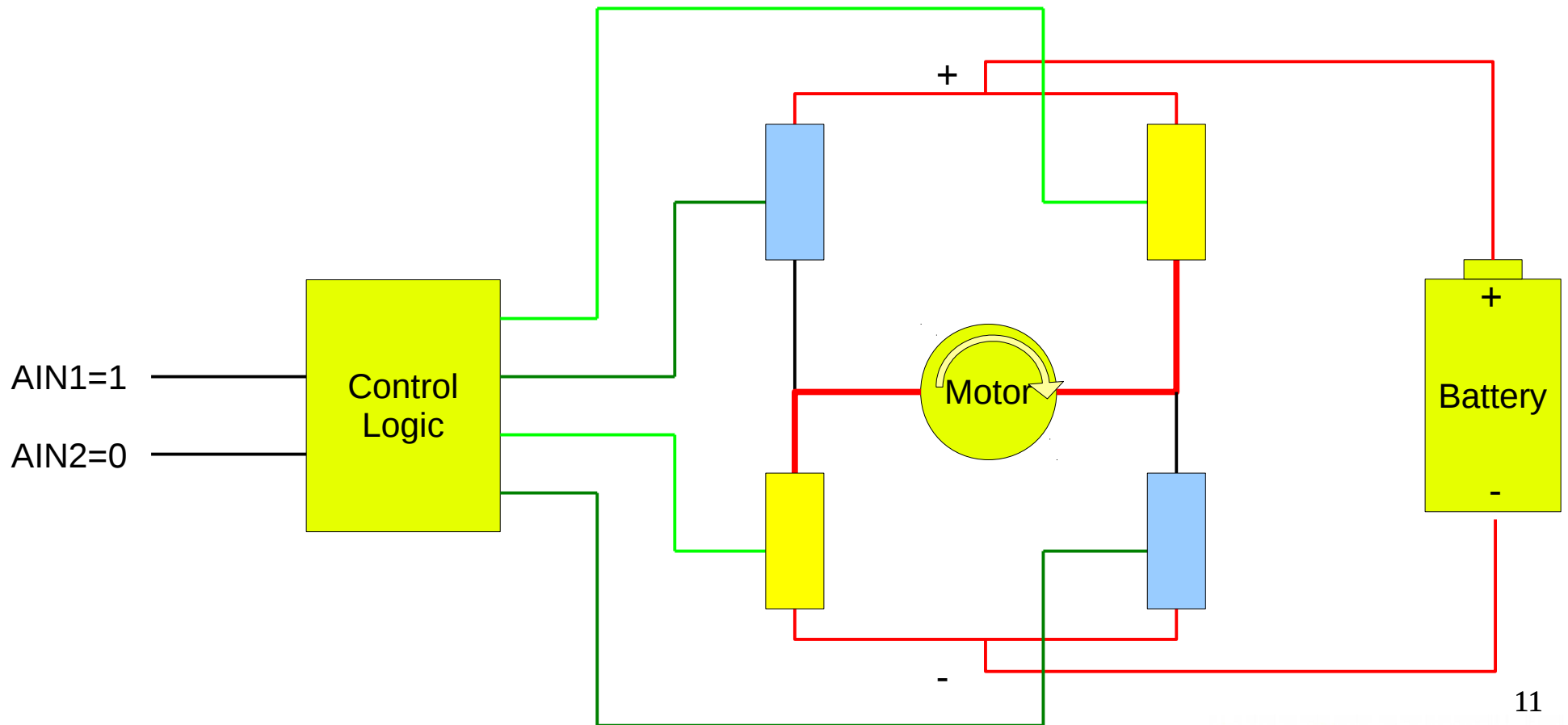
How the H-Bridge Driver Works

- 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.



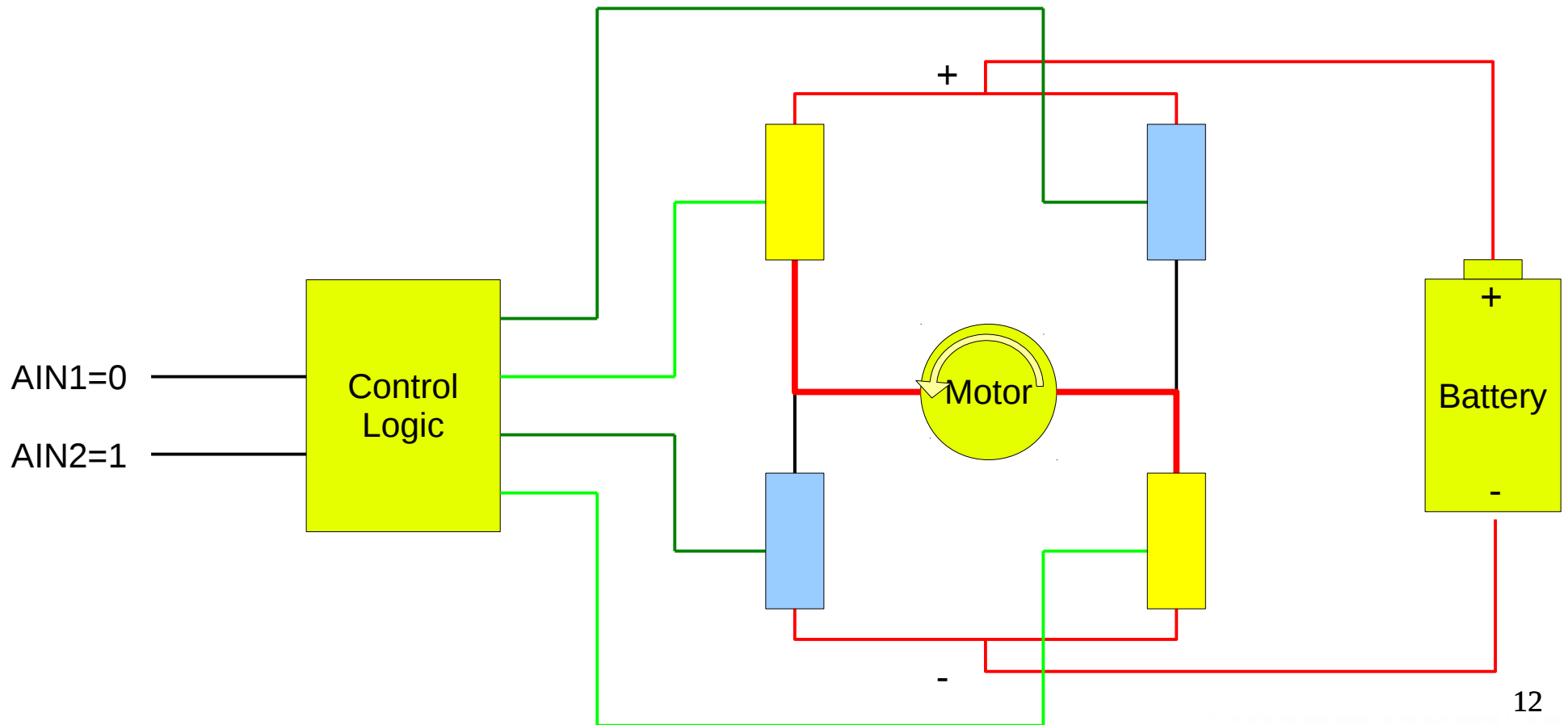
How the H-Bridge Driver Works

- 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.



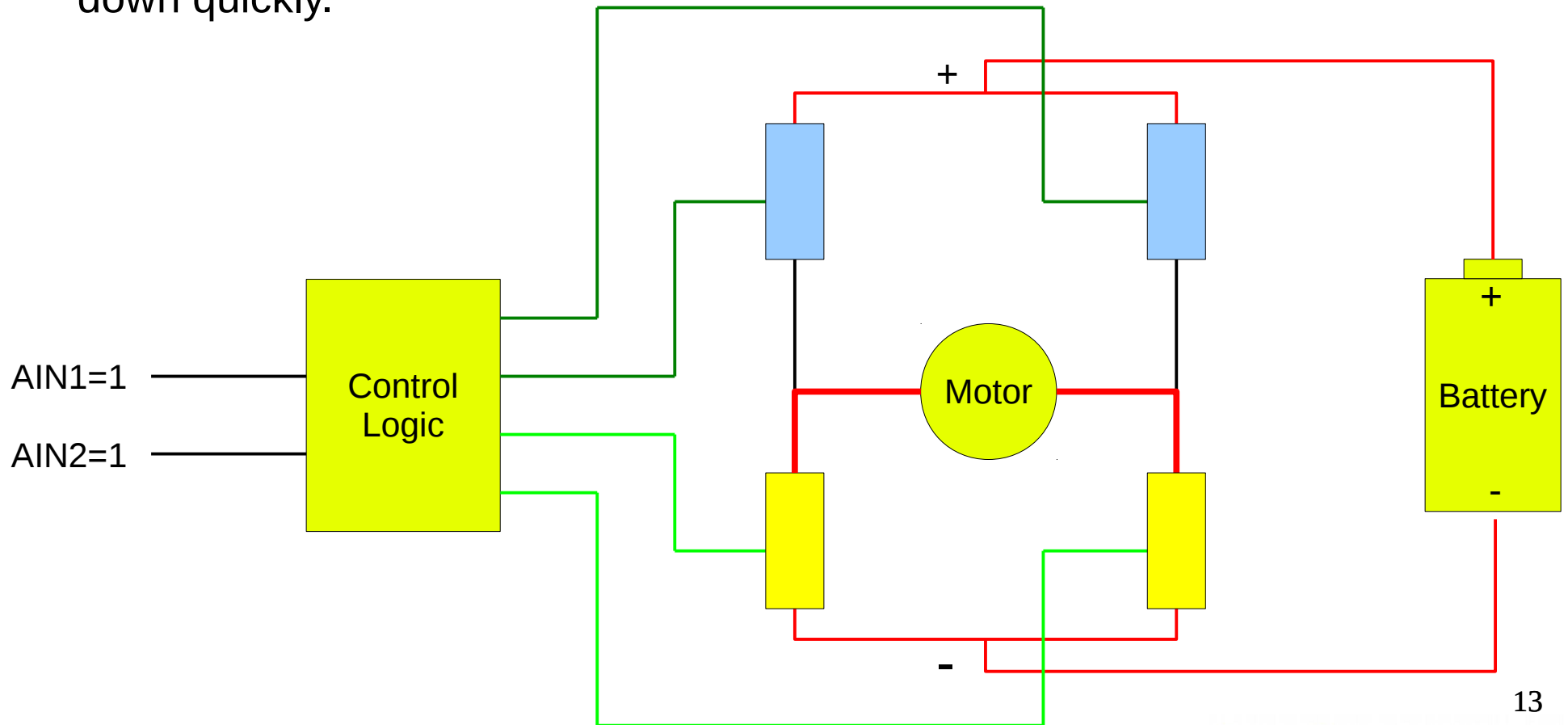
How the H-Bridge Driver Works

- 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.



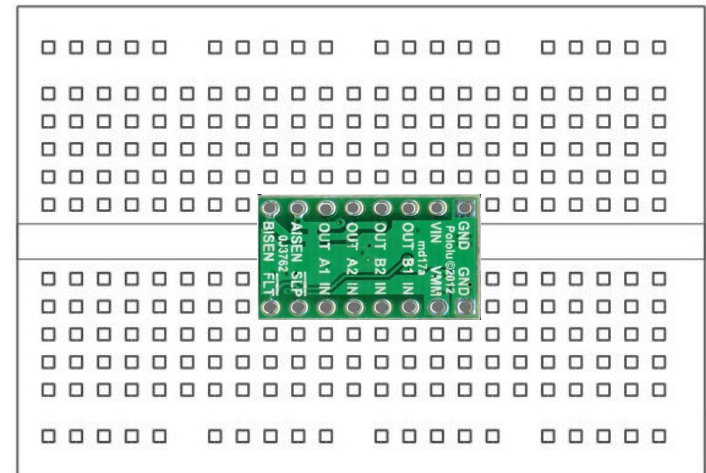
How the H-Bridge Driver Works

- When you set both AIN1 and AIN2 to logic 1, you get a braking action.
- This turns on the two bottom switches which shorts the motor connections together. The inductance created by the motor turning in one direction will power the motor to turn in the opposite direction. It causes the motor to slow down quickly.



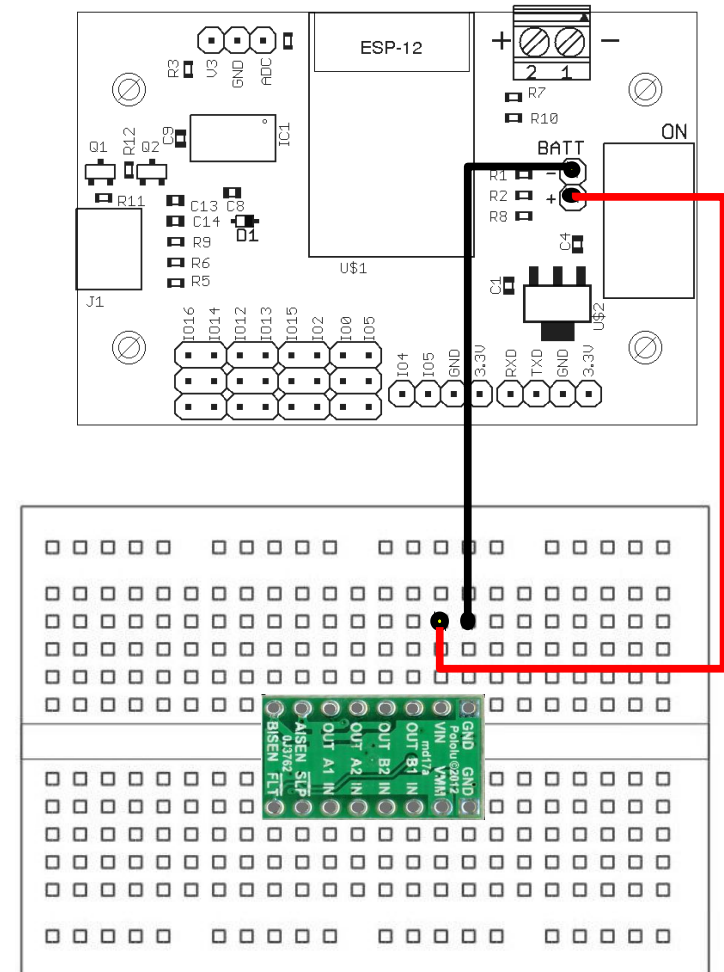
Installing the Motor Driver

- Remove the LED circuits from the solderless bread board.
- Insert the motor driver into the solderless bread board as shown. The driver needs to be mounted in the middle as shown.



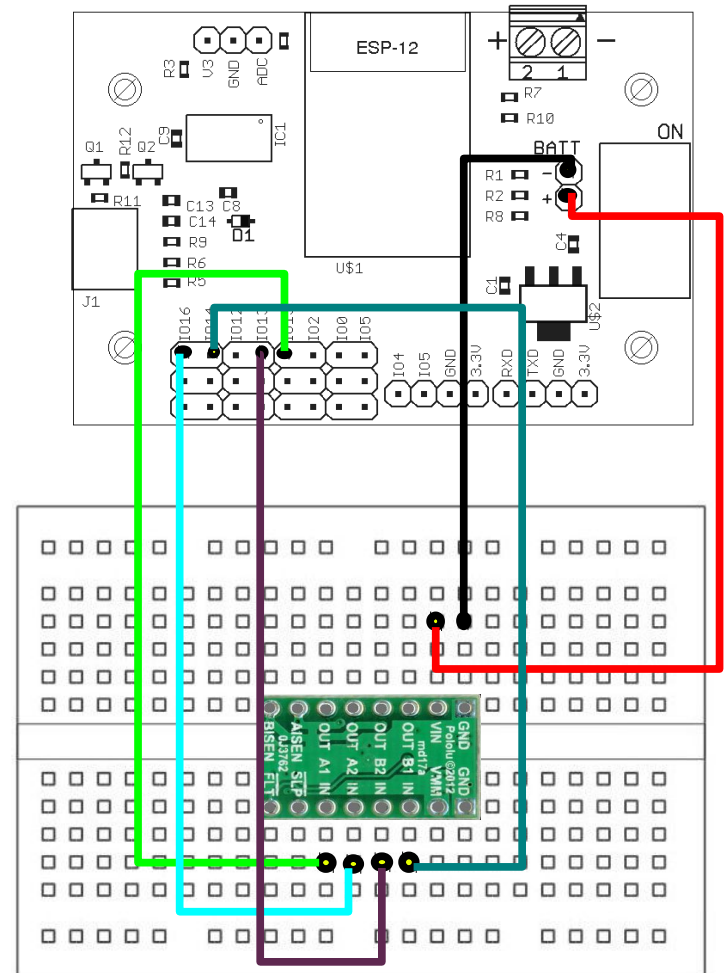
Connecting Power

- Connect a jumper wire from the pin marked **BATT+** to the signal VIN on the motor driver module.
- Connect a jumper wire from the pin next to **BATT -** marked GND to GND on the motor driver module.



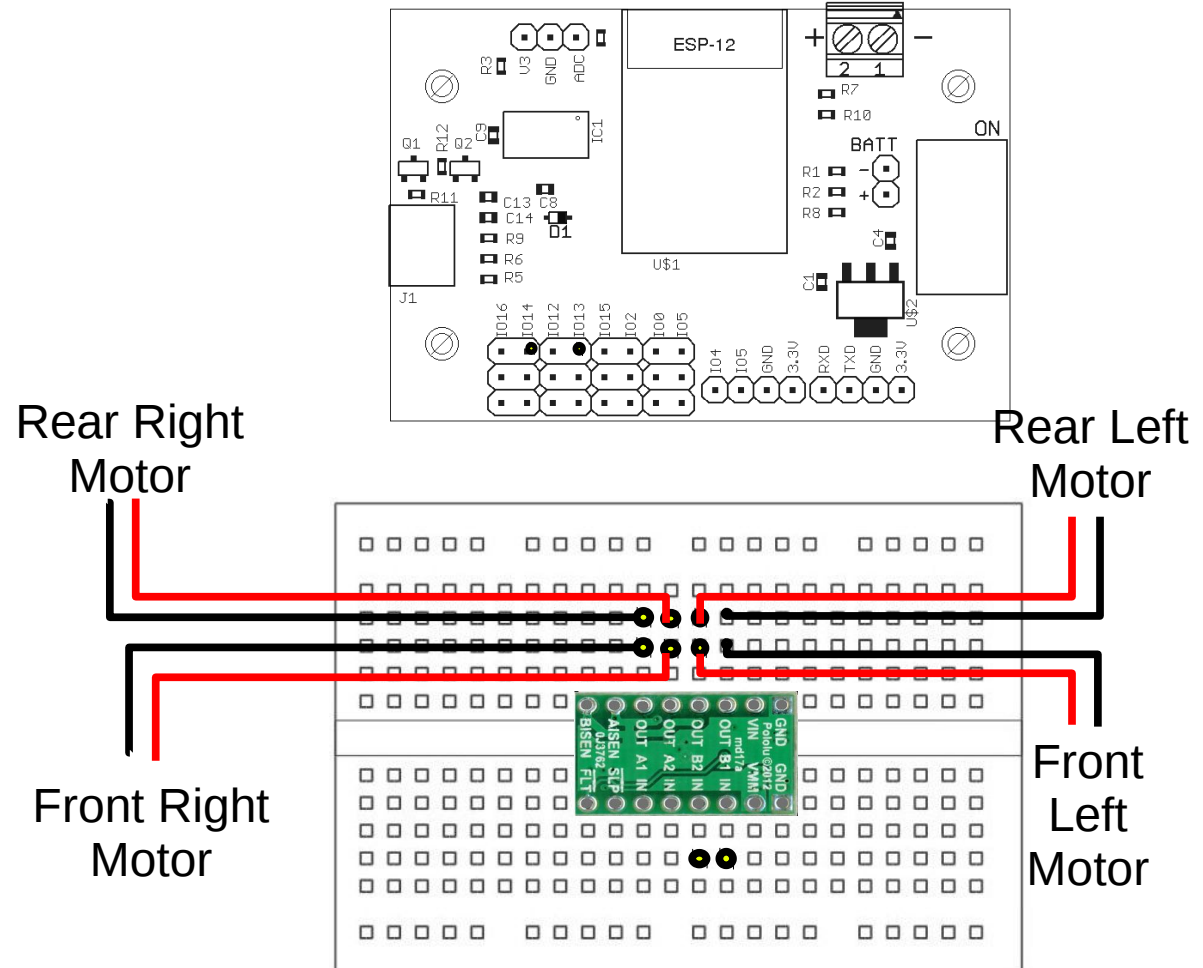
Wiring The Motor Controller

- Use the jumper wires to connect the motor controller.
- Connect A1 IN to D15
- Connect A2 IN to D16
- Connect B2 IN to D13
- Connect B1 IN to D14



Wiring The Motor Controller

- Connect the left motors to the A output of the motor driver.
- Connect the right motors to the B outputs of the motor driver.
- The color of the wires for the motors may vary. Follow the instructions on the next page for making sure the motors spin in the proper direction.



Warning

- After the motors are connected to the SLATE, whenever you upload, you must:
 - Have the battery connected and the switch on.
 - or disconnect the BATT + wire from the SLATE to the motor driver.
 - The reason is the motors begin pulling power from the USB port and causes the **esp comm** error.



Testing the Motors

- To operate the motors, signals A1 or A2 on the motor driver need to be set high or low.
- Operation is simple if A1 and A2 are set off, the motors do not operate.
- If A1 is set high and A2 is low, the motors will turn one direction.
- If A1 is low and A2 is high, the motors will turn in the opposite direction.
- The same applies for B1 and B2.
- Enter the program on the right to turn the motors on.
- See which way the wheels are turning and swap the motor pins if needed to make the wheels spin forward.
- These pin settings will be used for forward motion.

```
void setup()
{
    pinMode(13, OUTPUT);
    pinMode(14, OUTPUT);
    pinMode(15, OUTPUT);
    pinMode(16, OUTPUT);
}

void loop()
{
    digitalWrite(13, HIGH);
    digitalWrite(14, LOW);
    digitalWrite(15, HIGH);
    digitalWrite(16, LOW);
    delay(5000);
    digitalWrite(13, LOW);
    digitalWrite(15, LOW);
    delay(2000);
}
```



Direction Control

- The digital pins D13 and D14 control the right motors. (A = left)
 - Setting D13 high and D14 low makes the right wheels spin forward.
 - Setting D13 low and D14 high makes the right wheels spin reverse.
 - Setting D13 low and D14 low turns off the motors.
- The digital pins D15 and D16 control the left motors. (B = right)
 - Setting D15 high and D16 low makes the left wheels spin forward.
 - Setting D15 low and D16 high makes the left wheels spin reverse.
 - Setting D15 low and D16 low turns off the motors.
- Making the left motors go forward and the right motors go reverse turns the robot right.
- Making the left motors go reverse and the right motors go forward turns the robot left. The next page shows the code for each direction.

Direction Control Code

Forward Motion

```
digitalWrite(13,HIGH);  
digitalWrite(14,LOW);  
digitalWrite(15,HIGH);  
digitalWrite(16,LOW);
```

Reverse Motion

```
digitalWrite(13,LOW);  
digitalWrite(14,HIGH);  
digitalWrite(15,LOW);  
digitalWrite(16,HIGH);
```

Halt

```
digitalWrite(13,LOW);  
digitalWrite(14,LOW);  
digitalWrite(15,LOW);  
digitalWrite(16,LOW);
```

Right Turn

```
digitalWrite(13,LOW);  
digitalWrite(4,HIGH);  
digitalWrite(15,HIGH);  
digitalWrite(16,LOW);
```

Left Turn

```
digitalWrite(13,HIGH);  
digitalWrite(14,LOW);  
digitalWrite(15,LOW);  
digitalWrite(16,HIGH);
```

Creating Functions

- To make programming easier, functions will be created to specify the motions of the robot.
- A **function** is a collection of instructions that are grouped by braces { } and given a name.
- The format is shown to the right with one of the motions.
- The code in the loop function can call the forward function eliminating the need to rewrite the **digitalWrite()** functions every time.
- The motion functions should be inserted at the top of all programs.

```
void forward()  
{  
    digitalWrite(13, HIGH);  
    digitalWrite(14, LOW);  
    digitalWrite(15, HIGH);  
    digitalWrite(16, LOW);  
}
```



Motion Functions

```
void forward()  
{  
    digitalWrite(13, HIGH);  
    digitalWrite(14, LOW);  
    digitalWrite(15, HIGH);  
    digitalWrite(16, LOW);  
}
```

```
void reverse()  
{  
    digitalWrite(13, LOW);  
    digitalWrite(14, HIGH);  
    digitalWrite(15, LOW);  
    digitalWrite(16, HIGH);  
}
```

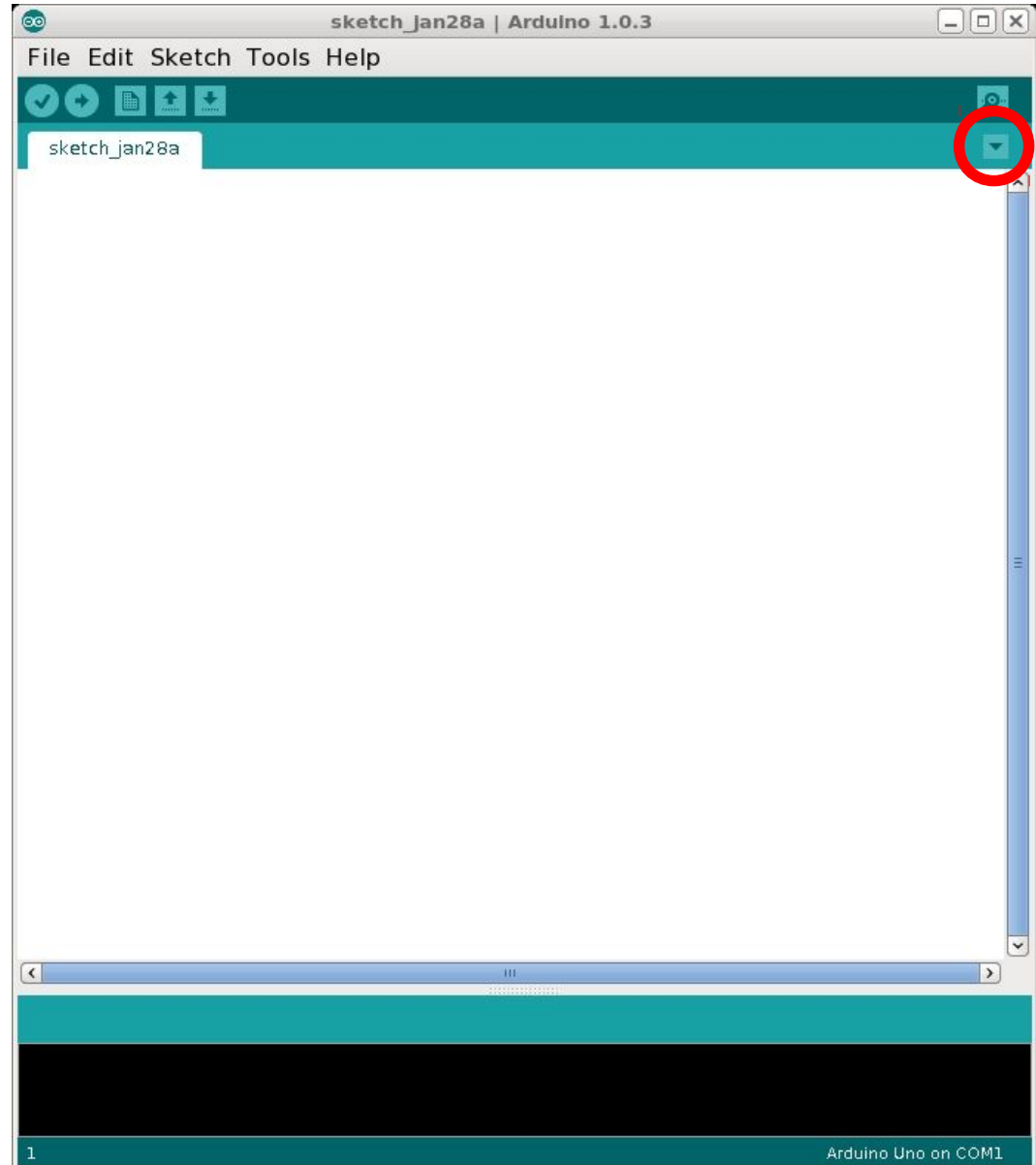
```
void halt()  
{  
    digitalWrite(13, LOW);  
    digitalWrite(14, LOW);  
    digitalWrite(15, LOW);  
    digitalWrite(16, LOW);  
}
```

```
void right()  
{  
    digitalWrite(13, LOW);  
    digitalWrite(14, HIGH);  
    digitalWrite(15, HIGH);  
    digitalWrite(16, LOW);  
}
```

```
void left()  
{  
    digitalWrite(13, HIGH);  
    digitalWrite(14, LOW);  
    digitalWrite(15, LOW);  
    digitalWrite(16, HIGH);  
}
```

Creating a Separate Function File

- Start a new program with the Arduino program.
- Click on the down arrow to the right where circled in red.
 - A menu will open. Select “New Tab”
 - Below, it will ask for a name. Enter 'motion'
 - Click 'OK'
 - A new tab is created called 'motion'
 - You will enter all the movement functions here.
- Enter the functions listed in the previous page.



Driving Around

- Click on the tab to the left of 'move' tab. Enter the code to the right.
- The code to the right is a start.
- Notice the **delay()** function is included after each motion function. This give the robot time to perform that motion. The value included in the delay function is time in milliseconds.
- Add directions to the program and change up the delays. Come up with a complex set of motions. Always remember to include a delay after the function to move the robot.
- When complete save the sketch as “move”. Remember this sketch. The motion file will be reused.

```
void setup()
{
    pinMode(13, OUTPUT);
    pinMode(14, OUTPUT);
    pinMode(15, OUTPUT);
    pinMode(16, OUTPUT);
}

void loop()
{
    forward();
    delay(1000);
    left();
    delay(400);
    reverse();
    delay(1000);
}
```

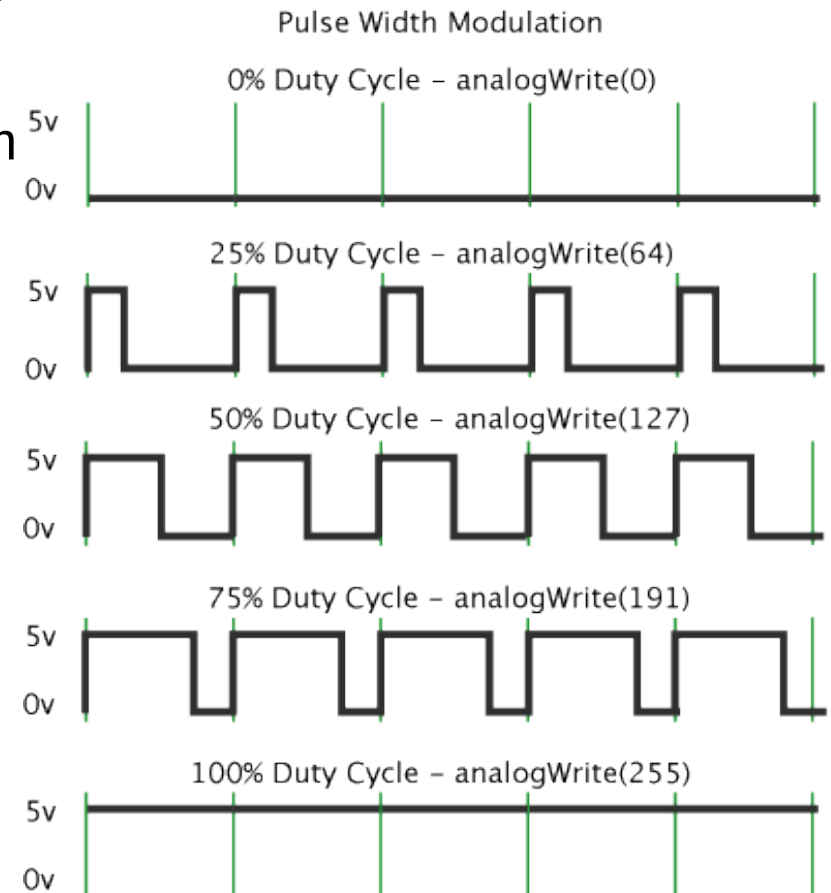


- Now that you know the basics. Write a program to make the robot move in a square pattern.
- Use the **delay()** function to control how long the robot turns and moves in a straight direction.



Speed Control

- It may be noticed that the robot may tend to drift to the left or right. This is due to the motors not being equally powerful.
- There is a way to attempt to equalize them by controlling their speed.
- A simple way to control the speed is to pulse power to the motors. This technique is called pulse width modulation.
- On the arduino, the **analogWrite()** function performs this. It generates a repeating pulse at about 490 Hz.
- The size of each pulse is the duty cycle. The higher the duty cycle the more power the motor gets.
- Adjusting the duty cycle will adjust the motor speed.



analogWrite()

- The function **analogWrite()** function takes two values.
 - First is the pin number.
 - Second is the duty cycle represented as a value from 0 to 1023.
 - 0 is 0% duty cycle.
 - 1023 is 100% duty cycle.
 - 511 is 50% duty cycle.
 - The function is written as
 - **analogWrite(pin, duty);**



Controlling Motor Speed

- Enter the program to the right. This program generates a PWM signal to the motor. Only one side needs a PWM signal. The other is set to 0 so no PWM signal is present.
- The code sets the PWM signal to 1023 which is 100% duty cycle meaning it is on all the time. This is the same as **digitalWrite()** function.
- Run the code and see which direction the robot drifts.
- Reduce the value for the opposite direction by 10 and try again. Keep adjusting until the robot drives relatively straight. It won't be perfect.
- The **analogWrite()** functions can replace the **digitalWrite()** functions in the motion functions.

```
void setup()
{
    pinMode(13, OUTPUT);
    pinMode(14, OUTPUT);
    pinMode(15, OUTPUT);
    pinMode(16, OUTPUT);
}

void loop()
{
    analogWrite(13, 1023);
    analogWrite(14, 0);
    analogWrite(15, 1023);
    analogWrite(16, 0);
    delay(5000);
    analogWrite(13, 0);
    analogWrite(15, 0);
    delay(2000);
}
```



Motor Speed Control Functions

- New functions can be created allowing you to control the speed of the robot.
- This function has two argument or parameters that get passed to it, **l** and **r**. These variables are set to the values passed and used inside the function only.
- Add the function to the right to the move tab.

```
void sforward(int l, int r)
{
    analogWrite(13,l);
    analogWrite(14,0);
    analogWrite(15,r);
    analogWrite(16,0);
}
```



Motor Speed Control Functions

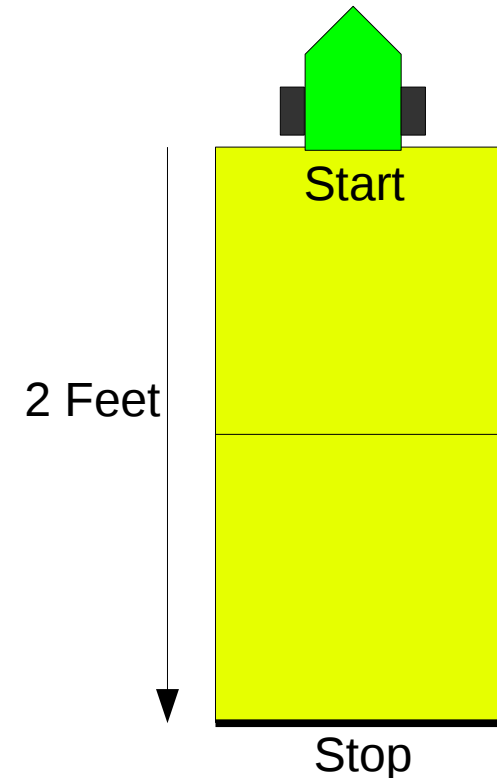
- In the first tab, change the program in the loop() function to what is in the right.
- Upload the code and see what it does.
- Modify the program to make the robot move in large curves and small curves.
- Modify the program to make the robot move in a S pattern and then in a circle.
- Lastly, add a function to make the robot go backwards with adjustable speed.

```
void loop()  
{  
    sforward(1000,1000);  
    delay(1000);  
    sforward(1000,600);  
    delay(1000);  
}
```



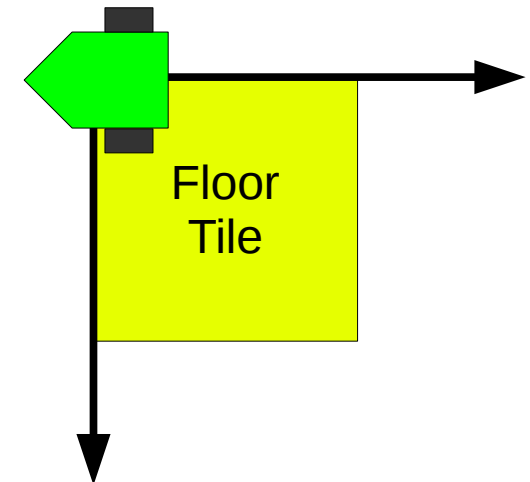
Calibrating Travel Distance

- 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 1000 ms.
- 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.



Calibrating Turns

- 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 below. 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.

