[yellow motor controller](https://www.amazon.com/RioRand-Brushless-Electric-Controller-Hall-Less/dp/B087M2378D?ref_=ast_sto_dp&th=1)

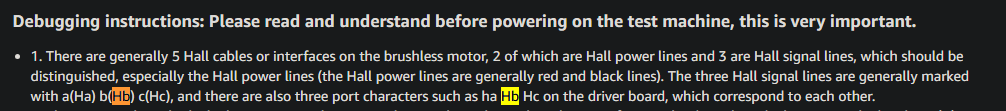
[RED motor controller](https://www.amazon.com/RioRand-Brushless-Electric-Controller-Hall-Less/dp/B087M2378D?ref_=ast_sto_dp&th=1)

[robot shop link to the motors](https://www.robotshop.com/products/e-s-motor-28mm-brushless-dc-planetary-gear-motor-12v-68rpm)

[says what the motor wires are (mostly)](https://cdn.robotshop.com/rbm/a00a7635-653b-4220-aac9-b0c23c5c5e2c/e/e8115577-62ef-460b-929e-61753e141324/3f7f3db6_28pg-2838bl-motor.pdf)

From amazon page





* right, so this is what the motor controllers would want if we had a hall MOTOR. what we have is a bldc single phase motor with hall ENCODER capabilities. if/when we decide to use this, our single motor wire would go somewhere in the ha hb hc terminals, where the giant white plug is (we are only using vcc and gnd as of right now)

**Possible Helpful Links for Arduino stuffs**

<https://mad-ee.com/controlling-a-hoverboard-motor-with-a-simple-arduino/> (pin assignment and analog write, which helps to set a pwm value)

* DIR on the mc might need to go to a pin on the mega instead of the motor pin going to the mc

<https://docs.cirkitdesigner.com/component/4cf35777-c70f-44c9-9828-838f50915cea/zs-x11h-v2-350w-motor-controller>

**Test code for both motors**

**// Pin definitions for RIGHT motor**

**const int rightPwmPin = 9; // PWM pin for speed control (Blue)**

**const int rightDirPin = 8; // Digital pin for direction control (White)**

**const int rightFgPin = 2; // FG signal input for speed measurement (Yellow)**

**// Pin definitions for LEFT motor**

**const int leftPwmPin = 11; // PWM pin for speed control (Blue)**

**const int leftDirPin = 10; // Digital pin for direction control (White)**

**const int leftFgPin = 3; // FG signal input for speed measurement (Yellow)**

**// Variables for speed measurement**

**volatile int rightPulseCount = 0; // Counter for RIGHT motor FG pulses**

**volatile int leftPulseCount = 0; // Counter for LEFT motor FG pulses**

**unsigned long lastTime = 0; // Time of last RPM calculation**

**void setup() {**

**// Set pin modes for RIGHT motor**

**pinMode(rightPwmPin, OUTPUT);**

**pinMode(rightDirPin, OUTPUT);**

**pinMode(rightFgPin, INPUT);**

**// Set pin modes for LEFT motor**

**pinMode(leftPwmPin, OUTPUT);**

**pinMode(leftDirPin, OUTPUT);**

**pinMode(leftFgPin, INPUT);**

**// Attach interrupts to FG pins for pulse counting**

**attachInterrupt(digitalPinToInterrupt(rightFgPin), rightPulse, RISING);**

**attachInterrupt(digitalPinToInterrupt(leftFgPin), leftPulse, RISING);**

**// Initialize serial communication for debugging**

**Serial.begin(9600);**

**// Initialize motor directions (HIGH = CW, LOW = CCW)**

**digitalWrite(rightDirPin, HIGH); // Set initial direction to clockwise**

**digitalWrite(leftDirPin, HIGH); // Set initial direction to clockwise**

**}**

**void loop() {**

**// Example: Move robot forward for 5 seconds**

**moveForward(128); // 50% speed**

**delay(5000);**

**// Example: Move robot backward for 5 seconds**

**moveBackward(128); // 50% speed**

**delay(5000);**

**// Example: Turn robot right for 2 seconds**

**turnRight(128); // 50% speed**

**delay(2000);**

**// Example: Turn robot left for 2 seconds**

**turnLeft(128); // 50% speed**

**delay(2000);**

**// Example: Stop the robot**

**stopMotors();**

**delay(2000);**

**}**

**// Function to move robot forward**

**void moveForward(int speed) {**

**// Set both motors to move forward**

**digitalWrite(rightDirPin, HIGH); // RIGHT motor forward**

**digitalWrite(leftDirPin, HIGH); // LEFT motor forward**

**analogWrite(rightPwmPin, speed); // Set RIGHT motor speed**

**analogWrite(leftPwmPin, speed); // Set LEFT motor speed**

**}**

**// Function to move robot backward**

**void moveBackward(int speed) {**

**// Set both motors to move backward**

**digitalWrite(rightDirPin, LOW); // RIGHT motor backward**

**digitalWrite(leftDirPin, LOW); // LEFT motor backward**

**analogWrite(rightPwmPin, speed); // Set RIGHT motor speed**

**analogWrite(leftPwmPin, speed); // Set LEFT motor speed**

**}**

**// Function to turn robot right**

**void turnRight(int speed) {**

**// RIGHT motor backward, LEFT motor forward**

**digitalWrite(rightDirPin, LOW); // RIGHT motor backward**

**digitalWrite(leftDirPin, HIGH); // LEFT motor forward**

**analogWrite(rightPwmPin, speed); // Set RIGHT motor speed**

**analogWrite(leftPwmPin, speed); // Set LEFT motor speed**

**}**

**// Function to turn robot left**

**void turnLeft(int speed) {**

**// RIGHT motor forward, LEFT motor backward**

**digitalWrite(rightDirPin, HIGH); // RIGHT motor forward**

**digitalWrite(leftDirPin, LOW); // LEFT motor backward**

**analogWrite(rightPwmPin, speed); // Set RIGHT motor speed**

**analogWrite(leftPwmPin, speed); // Set LEFT motor speed**

**}**

**// Function to stop both motors**

**void stopMotors() {**

**analogWrite(rightPwmPin, 0); // Stop RIGHT motor**

**analogWrite(leftPwmPin, 0); // Stop LEFT motor**

**}**

**// Interrupt service routine to count RIGHT motor FG pulses**

**void rightPulse() {**

**rightPulseCount++;**

**}**

**// Interrupt service routine to count LEFT motor FG pulses**

**void leftPulse() {**

**leftPulseCount++;**

**}**

**// Function to calculate RPM for RIGHT motor**

**int calculateRightRPM() {**

**unsigned long currentTime = millis();**

**unsigned long elapsedTime = currentTime - lastTime;**

**// Calculate RPM: (pulses / 9) \* (60000 / elapsedTime)**

**int rpm = (rightPulseCount / 9) \* (60000 / elapsedTime);**

**// Reset pulse count and update lastTime**

**rightPulseCount = 0;**

**lastTime = currentTime;**

**return rpm;**

**}**

**// Function to calculate RPM for LEFT motor**

**int calculateLeftRPM() {**

**unsigned long currentTime = millis();**

**unsigned long elapsedTime = currentTime - lastTime;**

**// Calculate RPM: (pulses / 9) \* (60000 / elapsedTime)**

**int rpm = (leftPulseCount / 9) \* (60000 / elapsedTime);**

**// Reset pulse count and update lastTime**

**leftPulseCount = 0;**

**lastTime = currentTime;**

**return rpm;**

**}**

**Code That We Know Makes It Move**

// Pin definitions

const int pwmPin = 9; // PWM pin for speed control (Blue)

const int dirPin = 8; // Digital pin for direction control (White)

const int fgPin = 2; // FG signal input for speed measurement (Yellow)

// Variables for speed measurement

volatile int pulseCount = 0; // Counter for FG pulses

unsigned long lastTime = 0; // Time of last RPM calculation

void setup() {

// Set pin modes

pinMode(pwmPin, OUTPUT);

pinMode(dirPin, OUTPUT);

pinMode(fgPin, INPUT);

// Attach interrupt to FG pin for pulse counting

attachInterrupt(digitalPinToInterrupt(fgPin), countPulses, RISING);

// Initialize serial communication for debugging

Serial.begin(9600);

// Initialize motor direction (HIGH = CW, LOW = CCW)

digitalWrite(dirPin, HIGH); // Set initial direction to clockwise

}

void loop() {

// Example: Rotate motor clockwise at 50% speed

digitalWrite(dirPin, HIGH); // Set direction to clockwise

analogWrite(pwmPin, 128); // Set speed to 50% (PWM value 128/255)

delay(5000); // Run for 5 seconds

// Measure and print RPM

int rpm = calculateRPM();

Serial.println("Clockwise RPM: " + String(rpm));

// Example: Rotate motor counterclockwise at 75% speed

digitalWrite(dirPin, LOW); // Set direction to counterclockwise

analogWrite(pwmPin, 192); // Set speed to 75% (PWM value 192/255)

delay(5000); // Run for 5 seconds

// Measure and print RPM

rpm = calculateRPM();

Serial.println("Counterclockwise RPM: " + String(rpm));

// Example: Stop the motor

analogWrite(pwmPin, 0); // Set speed to 0% (motor stops)

delay(2000); // Wait for 2 seconds

}

// Interrupt service routine to count FG pulses

void countPulses() {

pulseCount++;

}

// Function to calculate RPM

int calculateRPM() {

unsigned long currentTime = millis();

unsigned long elapsedTime = currentTime - lastTime;

// Calculate RPM: (pulses / 9) \* (60000 / elapsedTime)

int rpm = (pulseCount / 9) \* (60000 / elapsedTime);

// Reset pulse count and update lastTime

pulseCount = 0;

lastTime = currentTime;

return rpm;

}