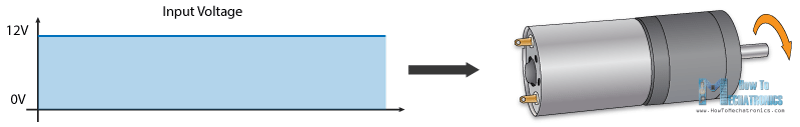
# Arduino DC Motor Control Tutorial – L298N | PWM | H-Bridge

In this Arduino Tutorial we will learn how to control DC motors using Arduino. We well take a look at some basic techniques for controlling DC motors and make two example through which we will learn how to control DC motors using the L298N driver and the Arduino board.

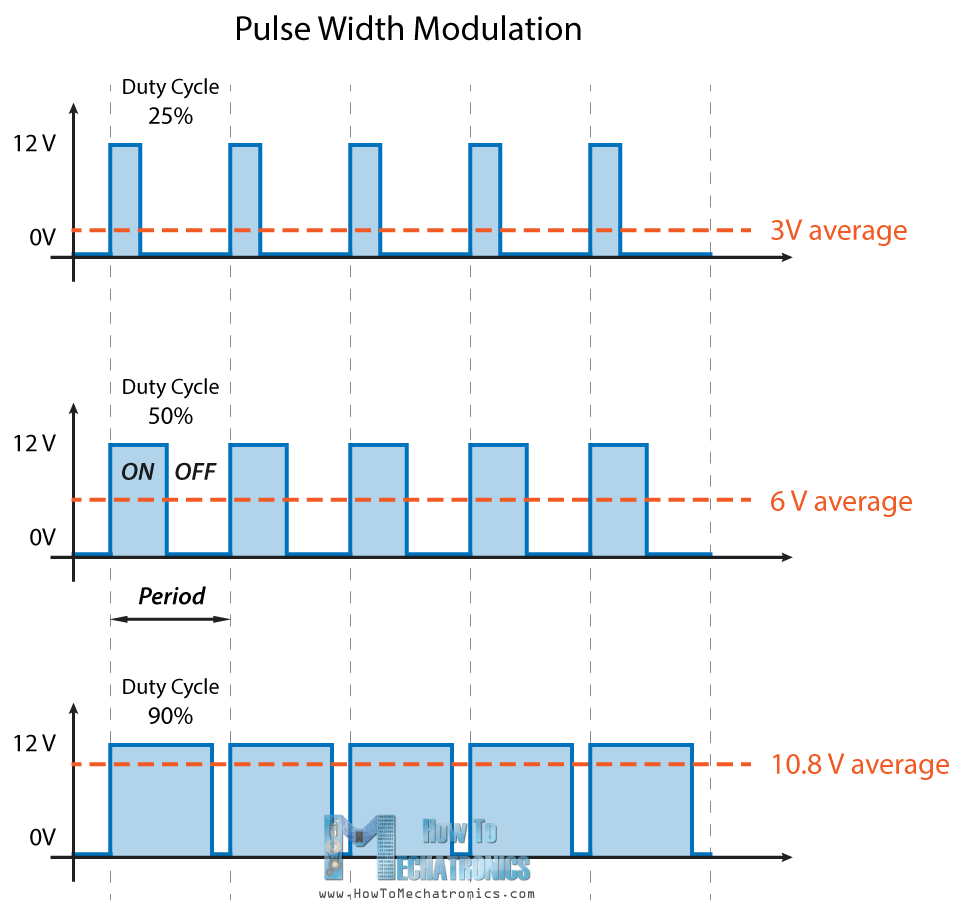
**Overview**

We can control the speed of the DC motor by simply controlling the input voltage to the motor and the most common method of doing that is by using PWM signal.

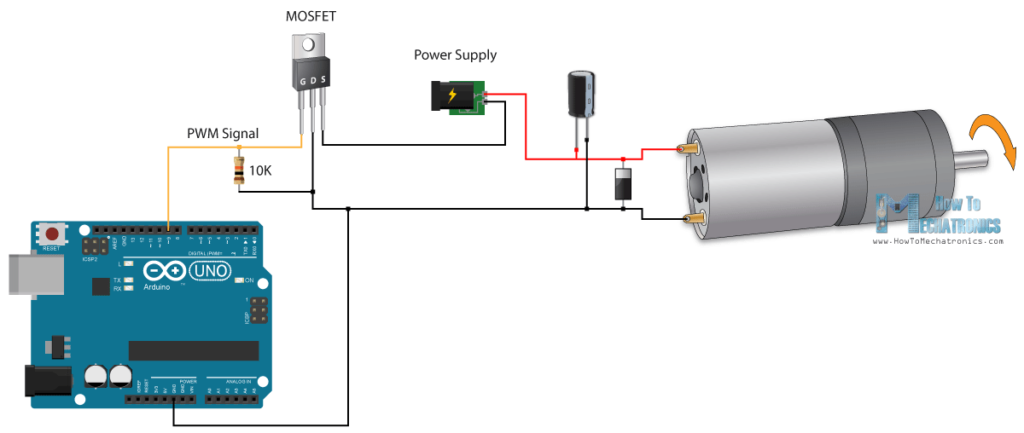


# PWM DC Motor Control

PWM, or pulse width modulation is a technique which allows us to adjust the average value of the voltage that’s going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time.



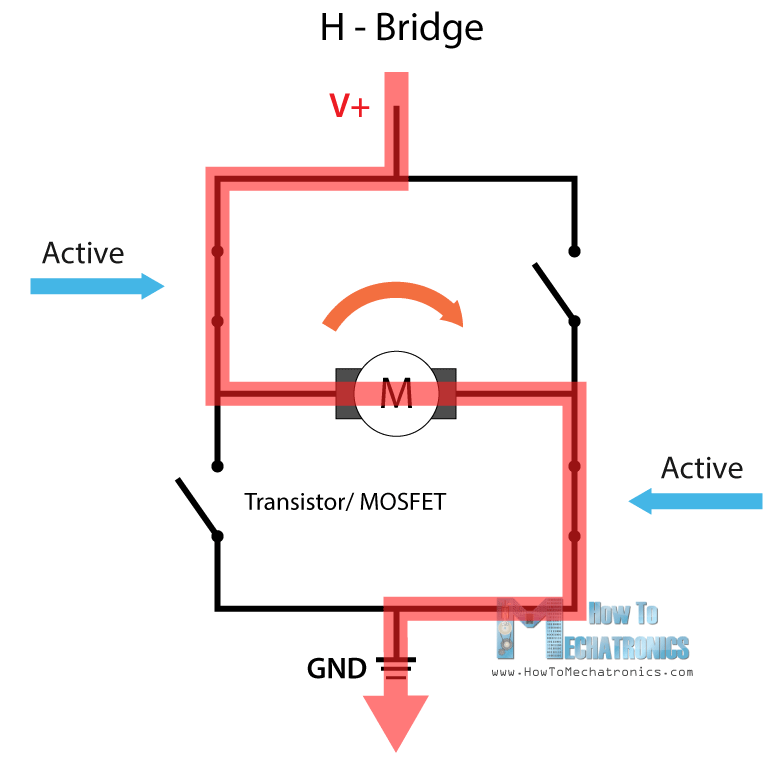
 So depending on the size of the motor, we can simply connect an Arduino PWM output to the base of transistor or the gate of a MOSFET and control the speed of the motor by controlling the PWM output. The low power Arduino PWM signal switches on and off the gate at the MOSFET through which the high power motor is driven.

[](https://howtomechatronics.com/wp-content/uploads/2017/08/Arduino-PWM-DC-Motor-Control-Circuit-Diagram.png)

Note: Arduino GND and the motor power supply GND should be connected together.

# H-Bridge DC Motor Control

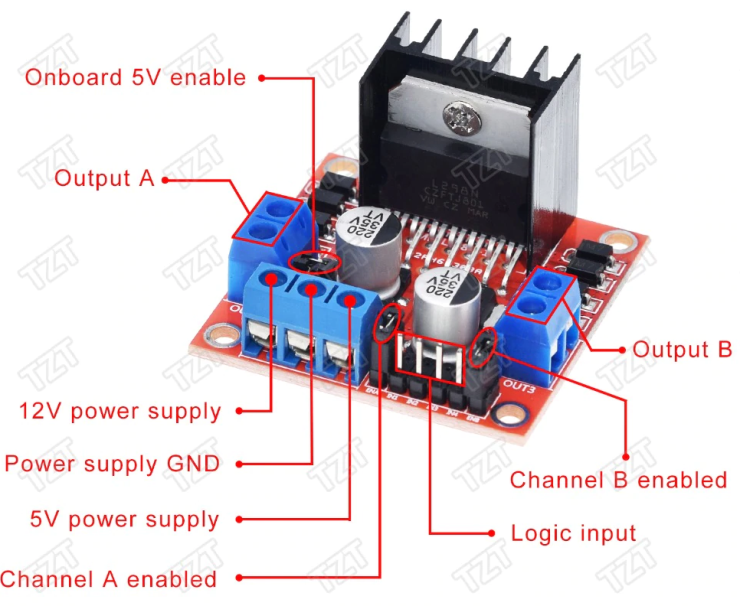
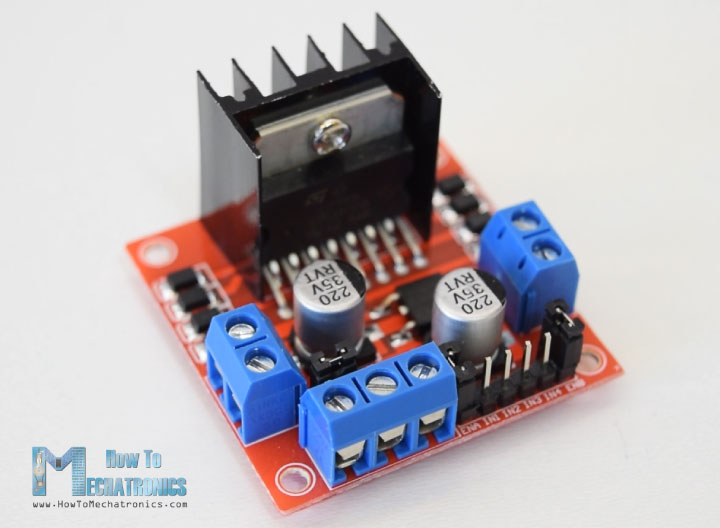
On the other hand, for controlling the rotation direction, we just need to inverse the direction of the current flow through the motor, and the most common method of doing that is by using an H-Bridge. An H-Bridge circuit contains four switching elements, transistors or MOSFETs, with the motor at the center forming an H-like configuration. By activating two particular switches at the same time we can change the direction of the current flow, thus change the rotation direction of the motor.



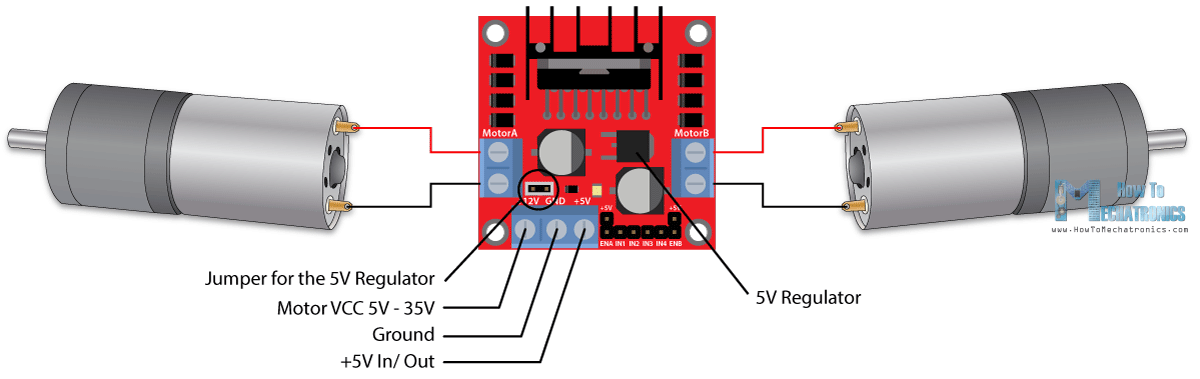
So if we combine these two methods, the PWM and the H-Bridge, we can have a complete control over the DC motor. There are many DC motor drivers that have these features and the L298N is one of them.

# L298N Driver

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

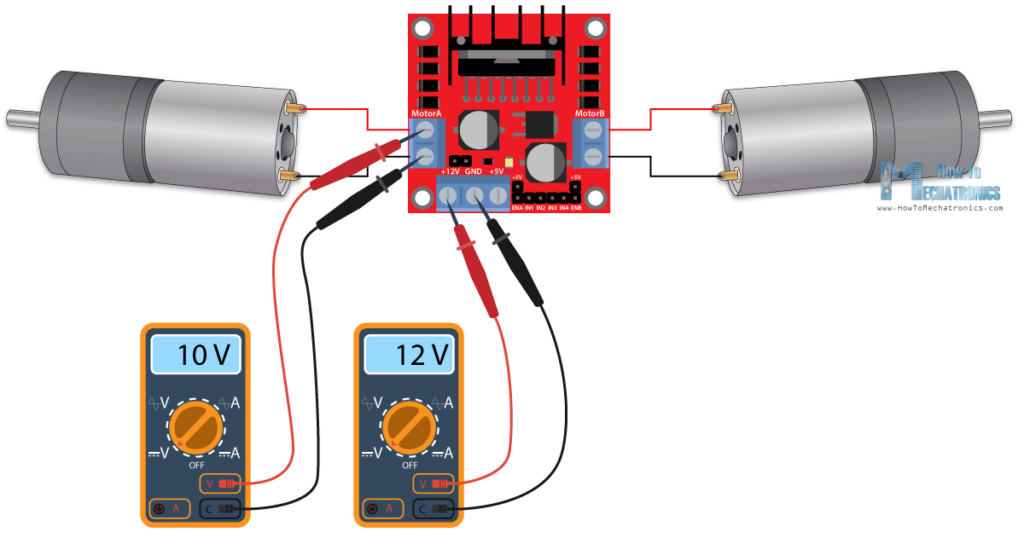


Let’s take a closer look at the pinout of L298N module and explain how it works. The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output.

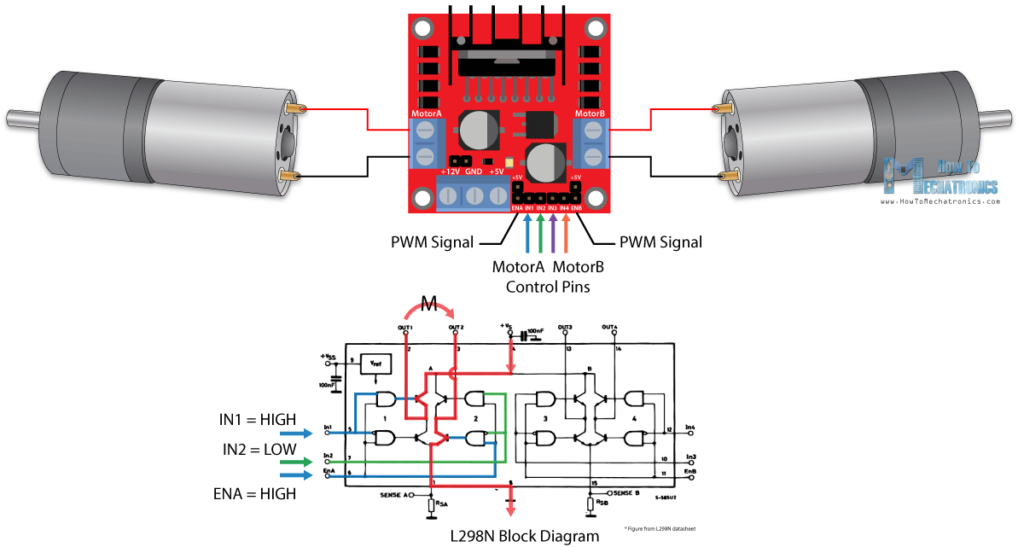


This depends on the voltage used at the motors VCC. The module have an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly.

We can note here that this IC makes a voltage drop of about 2V. So for example, if we use a 12V power supply, the voltage at motors terminals will be about 10V, which means that we won’t be able to get the maximum speed out of our 12V DC motor.



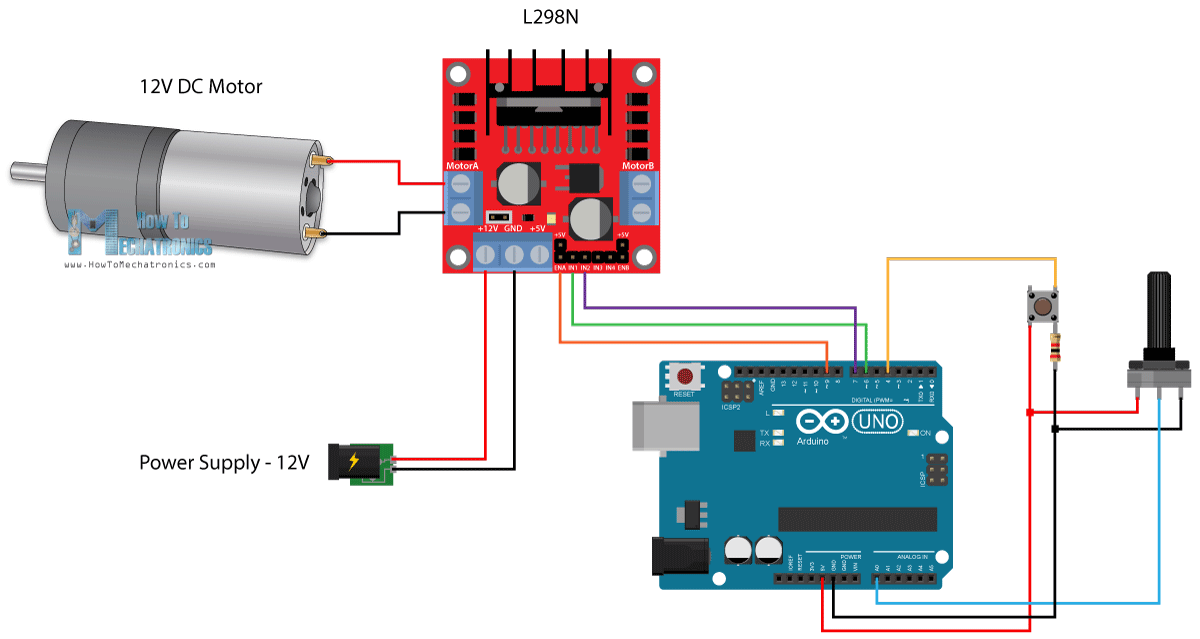
Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.

[](https://howtomechatronics.com/wp-content/uploads/2017/08/L298N-Block-Diagram-Current-Flow-How-It-Works.png)

Next, the Input 1 and Input 2 pins are used for controlling the rotation direction of the motor A, and the inputs 3 and 4 for the motor B. Using these pins we actually control the switches of the H-Bridge inside the L298N IC. If input 1 is LOW and input 2 is HIGH the motor will move forward, and vice versa, if input 1 is HIGH and input 2 is LOW the motor will move backward. In case both inputs are same, either LOW or HIGH the motor will stop. The same applies for the inputs 3 and 4 and the motor B.

# Arduino and L298N

Now let’s make some practical applications. In the first example we will control the speed of the motor using a potentiometer and change the rotation direction using a push button. Here’s the circuit schematics.

[](https://howtomechatronics.com/wp-content/uploads/2017/08/Arduino-and-L298N-Circuit-Diagram-DC-Motor-Control.png)

So we need an L298N driver, a DC motor, a potentiometer, a push button and an Arduino board.

You can get the components needed for this Arduino Tutorial from the links below:

## Arduino Code

Here’s the Arduino code:

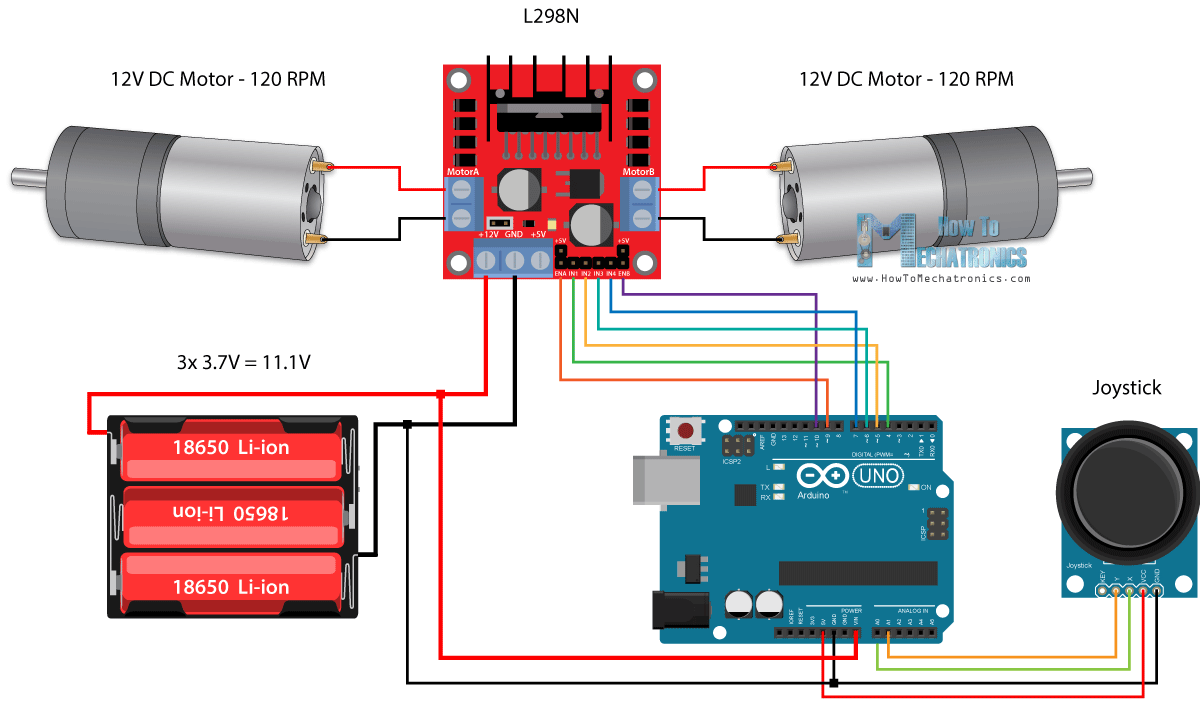
1. /\* Arduino DC Motor Control - PWM | H-Bridge | L298N - Example 01
2. by Dejan Nedelkovski, www.HowToMechatronics.com
3. \*/
4. #define enA 9
5. #define in1 6
6. #define in2 7
7. #define button 4
8. **int** rotDirection = 0;
9. **int** pressed = **false**;
10. **void** setup() {
11. pinMode(enA, OUTPUT);
12. pinMode(in1, OUTPUT);
13. pinMode(in2, OUTPUT);
14. pinMode(button, INPUT);
15. // Set initial rotation direction
16. digitalWrite(in1, LOW);
17. digitalWrite(in2, HIGH);
18. }
19. **void** loop() {
20. **int** potValue = analogRead(A0); // Read potentiometer value
21. **int** pwmOutput = map(potValue, 0, 1023, 0 , 255); // Map the potentiometer value from 0 to 255
22. analogWrite(enA, pwmOutput); // Send PWM signal to L298N Enable pin
23. // Read button - Debounce
24. **if** (digitalRead(button) == **true**) {
25. pressed = !pressed;
26. }
27. **while** (digitalRead(button) == **true**);
28. delay(20);
29. // If button is pressed - change rotation direction
30. **if** (pressed == **true** & rotDirection == 0) {
31. digitalWrite(in1, HIGH);
32. digitalWrite(in2, LOW);
33. rotDirection = 1;
34. delay(20);
35. }
36. // If button is pressed - change rotation direction
37. **if** (pressed == **false** & rotDirection == 1) {
38. digitalWrite(in1, LOW);
39. digitalWrite(in2, HIGH);
40. rotDirection = 0;
41. delay(20);
42. }
43. }

**Description:** So first we need to define the pins and some variables needed for the program. In the setup section we need to set the pin modes and the initial rotation direction of the motor. In the loop section we start by reading the potentiometer value and then map the value that we get from it which is from 0 to 1023, to a value from 0 to 255 for the PWM signal, or that’s 0 to 100% duty cycle of the PWM signal. Then using the analogWrite() function we send the PWM signal to the Enable pin of the L298N board, which actually drives the motor.

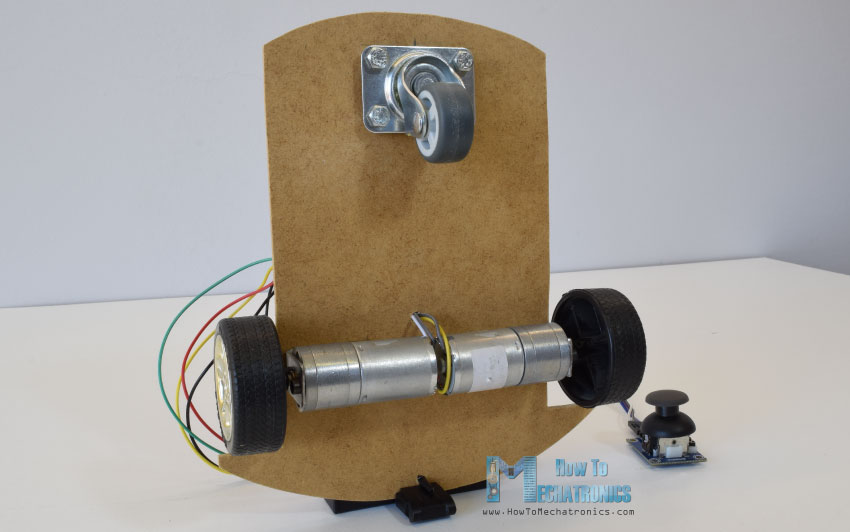
Next, we check whether we have pressed the button, and if that’s true, we will change the rotation direction of the motor by setting the Input 1 and Input 2 states inversely. The push button will work as toggle button and each time we press it, it will change the rotation direction of the motor.

# Arduino Robot Car Control using L298N Driver

So once we have learned this, now we can build our own Arduino robot car. Here’s the circuit schematic:

[](https://howtomechatronics.com/wp-content/uploads/2017/08/Arduino-Robot-Car-Control-using-L298N-Driver-Circuit-Schematic.png)

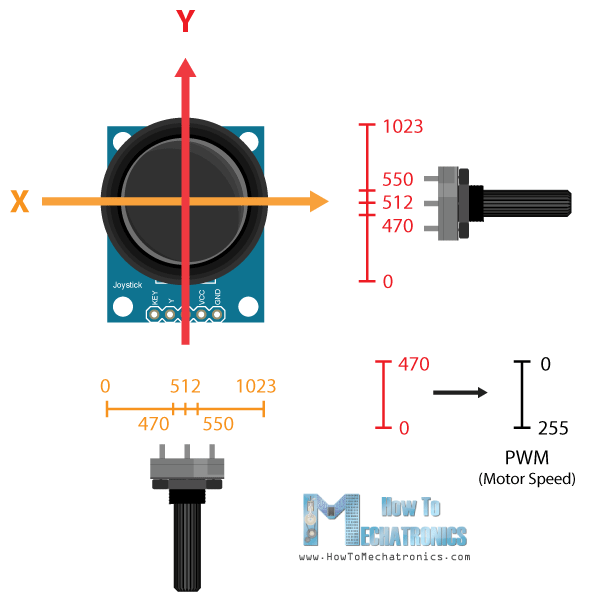
All we need is 2 DC Motors, the L298N driver, an Arduino board and a [joystick](https://howtomechatronics.com/recommends/joystick-banggood/) for the control. As for the power supply, I chose to use three 3.7V Li-ion batteries, providing total of 11V. I made the chassis out of 3 mm tick plywood, attached the motors to it using metal brackets, attached wheels to the motors and in front attached a swivel wheel.



Now let’s take a look at the Arduino code and see how it works. (Down below you can find the complete code)

1. **int** xAxis = analogRead(A0); // Read Joysticks X-axis
2. **int** yAxis = analogRead(A1); // Read Joysticks Y-axis

After defining the pins, in the loop section, we start with reading the joystick X and Y axis values. The joystick is actually made of two potentiometers which are connected to the analog inputs of the Arduino and they have values from 0 to 1023. When the joystick stays in its center position the value of both potentiometers, or axes is around 512.



We will add a little tolerance and consider the values from 470 to 550 as center. So if we move the Y axis of joystick backward and the value goes below 470 we will set the two motors rotation direction to backward using the four input pins. Then, we will convert the declining values from 470 to 0 into increasing PWM values from 0 to 255 which is actually the speed of the motor.

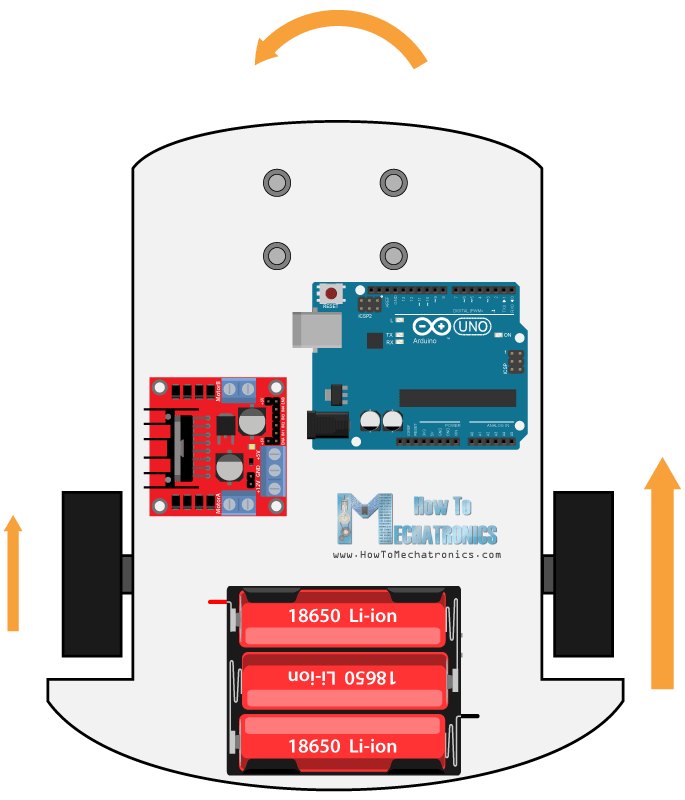
1. // Y-axis used for forward and backward control
2. **if** (yAxis < 470) {
3. // Set Motor A backward
4. digitalWrite(in1, HIGH);
5. digitalWrite(in2, LOW);
6. // Set Motor B backward
7. digitalWrite(in3, HIGH);
8. digitalWrite(in4, LOW);
9. // Convert the declining Y-axis readings for going backward from 470 to 0 into 0 to 255 value for the PWM signal for increasing the motor speed
10. motorSpeedA = map(yAxis, 470, 0, 0, 255);
11. motorSpeedB = map(yAxis, 470, 0, 0, 255);
12. }

Similar, if we move the Y axis of the joystick forward and the value goes above 550 we will set the motors to move forward and convert the readings from 550 to 1023 into PWM values from 0 to 255. If the joystick stays in its center the motors speed will be zero.

Next, let’s see how we use the X axis for the left and right control of the car.

1. // X-axis used for left and right control
2. **if** (xAxis < 470) {
3. // Convert the declining X-axis readings from 470 to 0 into increasing 0 to 255 value
4. **int** xMapped = map(xAxis, 470, 0, 0, 255);
5. // Move to left - decrease left motor speed, increase right motor speed
6. motorSpeedA = motorSpeedA - xMapped;
7. motorSpeedB = motorSpeedB + xMapped;
8. // Confine the range from 0 to 255
9. **if** (motorSpeedA < 0) {
10. motorSpeedA = 0;
11. }
12. **if** (motorSpeedB > 255) {
13. motorSpeedB = 255;
14. }
15. }

So again, first we need to convert the X axis readings into speed values from 0 to 255. For moving left, we use this value to decrease the left motor speed and increase the right motor speed. Here, because of the arithmetic functions we use two additional “if” statements to confine the range of the motor speed from 0 to 255.



The same method is used for moving the car to the right.

Depending on the applied voltage and the motor itself, at lower speeds the motor is not able to start moving and it produces a buzzing sound. In my case, the motors were not able to move if the value of the PWM signal was below 70. Therefore using this two if statements I actually confined to speed range from 70 to 255. At the end we just send the final motor speeds or PWM signal to the enable pins of the L298N driver.

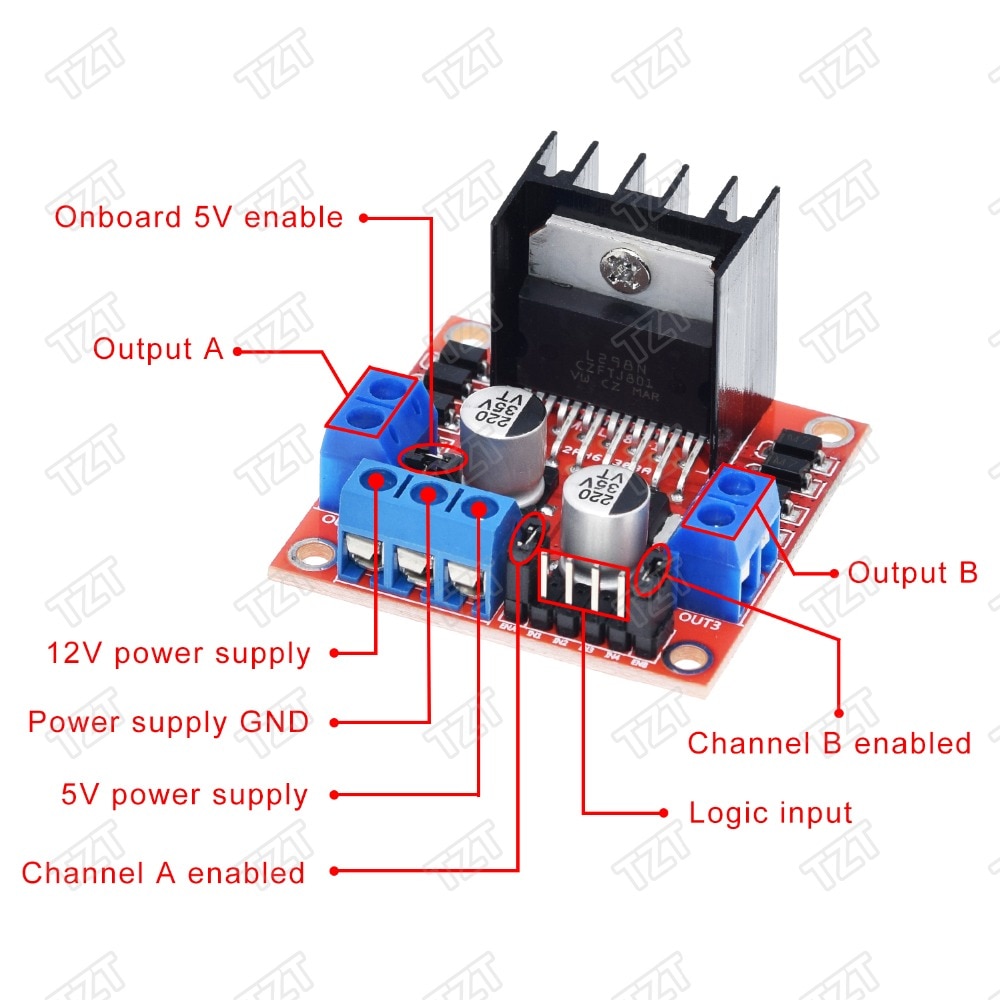
1. // Prevent buzzing at low speeds (Adjust according to your motors. My motors couldn't start moving if PWM value was below value of 70)
2. **if** (motorSpeedA < 70) {
3. motorSpeedA = 0;
4. }
5. **if** (motorSpeedB < 70) {
6. motorSpeedB = 0;
7. }
8. analogWrite(enA, motorSpeedA); // Send PWM signal to motor A
9. analogWrite(enB, motorSpeedB); // Send PWM signal to motor B

Here’s the complete code of the Arduino robot car example:

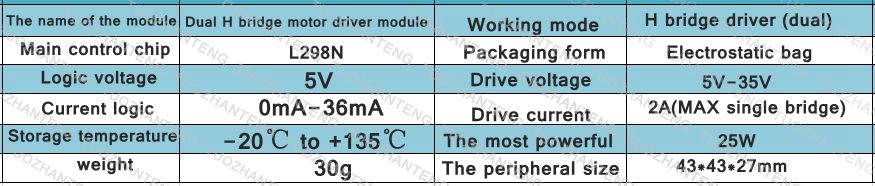
1. /\* Arduino DC Motor Control - PWM | H-Bridge | L298N
2. Example 02 - Arduino Robot Car Control
3. by Dejan Nedelkovski, www.HowToMechatronics.com
4. \*/
5. #define enA 9
6. #define in1 4
7. #define in2 5
8. #define enB 10
9. #define in3 6
10. #define in4 7
11. **int** motorSpeedA = 0;
12. **int** motorSpeedB = 0;
13. **void** setup() {
14. pinMode(enA, OUTPUT);
15. pinMode(enB, OUTPUT);
16. pinMode(in1, OUTPUT);
17. pinMode(in2, OUTPUT);
18. pinMode(in3, OUTPUT);
19. pinMode(in4, OUTPUT);
20. }
21. **void** loop() {
22. **int** xAxis = analogRead(A0); // Read Joysticks X-axis
23. **int** yAxis = analogRead(A1); // Read Joysticks Y-axis
24. // Y-axis used for forward and backward control
25. **if** (yAxis < 470) {
26. // Set Motor A backward
27. digitalWrite(in1, HIGH);
28. digitalWrite(in2, LOW);
29. // Set Motor B backward
30. digitalWrite(in3, HIGH);
31. digitalWrite(in4, LOW);
32. // Convert the declining Y-axis readings for going backward from 470 to 0 into 0 to 255 value for the PWM signal for increasing the motor speed
33. motorSpeedA = map(yAxis, 470, 0, 0, 255);
34. motorSpeedB = map(yAxis, 470, 0, 0, 255);
35. }
36. **else** **if** (yAxis > 550) {
37. // Set Motor A forward
38. digitalWrite(in1, LOW);
39. digitalWrite(in2, HIGH);
40. // Set Motor B forward
41. digitalWrite(in3, LOW);
42. digitalWrite(in4, HIGH);
43. // Convert the increasing Y-axis readings for going forward from 550 to 1023 into 0 to 255 value for the PWM signal for increasing the motor speed
44. motorSpeedA = map(yAxis, 550, 1023, 0, 255);
45. motorSpeedB = map(yAxis, 550, 1023, 0, 255);
46. }
47. // If joystick stays in middle the motors are not moving
48. **else** {
49. motorSpeedA = 0;
50. motorSpeedB = 0;
51. }
52. // X-axis used for left and right control
53. **if** (xAxis < 470) {
54. // Convert the declining X-axis readings from 470 to 0 into increasing 0 to 255 value
55. **int** xMapped = map(xAxis, 470, 0, 0, 255);
56. // Move to left - decrease left motor speed, increase right motor speed
57. motorSpeedA = motorSpeedA - xMapped;
58. motorSpeedB = motorSpeedB + xMapped;
59. // Confine the range from 0 to 255
60. **if** (motorSpeedA < 0) {
61. motorSpeedA = 0;
62. }
63. **if** (motorSpeedB > 255) {
64. motorSpeedB = 255;
65. }
66. }
67. **if** (xAxis > 550) {
68. // Convert the increasing X-axis readings from 550 to 1023 into 0 to 255 value
69. **int** xMapped = map(xAxis, 550, 1023, 0, 255);
70. // Move right - decrease right motor speed, increase left motor speed
71. motorSpeedA = motorSpeedA + xMapped;
72. motorSpeedB = motorSpeedB - xMapped;
73. // Confine the range from 0 to 255
74. **if** (motorSpeedA > 255) {
75. motorSpeedA = 255;
76. }
77. **if** (motorSpeedB < 0) {
78. motorSpeedB = 0;
79. }
80. }
81. // Prevent buzzing at low speeds (Adjust according to your motors. My motors couldn't start moving if PWM value was below value of 70)
82. **if** (motorSpeedA < 70) {
83. motorSpeedA = 0;
84. }
85. **if** (motorSpeedB < 70) {
86. motorSpeedB = 0;
87. }
88. analogWrite(enA, motorSpeedA); // Send PWM signal to motor A
89. analogWrite(enB, motorSpeedB); // Send PWM signal to motor B
90. }

So that would be all for this tutorial, and in my next video we will upgrade this Arduino robot car, by adding a [Bluetooth](https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/) and [Radio](https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-12-long-range-wireless-communication-module/) devices for enabling smartphone and wireless control.

# Arduino STEPPER Motor Control Tutorial – L298N | PWM | H-Bridge



L298N è un ad alta tensione, alta corrente di chip di driver del motore. Il chip utilizza 15-pin package. Le caratteristiche principali sono: ad alta tensione, tensione di funzionamento massima fino a 46 V; corrente di uscita, istantanea corrente di picco fino a 3A, corrente di funzionamento continuo di 2A; potenza nominale 25 W. Contiene due H-bridge ad alta tensione e alta corrente driver full-bridge può essere usato per guidare motori a corrente CONTINUA e motori passo passo, relè, bobine e altri carico induttivo; utilizzando lo standard di livello logico di controllo del segnale; avere due abilitare terminale di controllo, in permette segnale di ingresso senza essere colpiti o disattivare il dispositivo è dotato di una logica di alimentazione di ingresso, La logica interna parte del circuito del lavoro a una bassa tensione; può essere un esterno resistore di rilevamento, la quantità di cambiare di nuovo per il circuito di controllo. Motore di azionamento utilizzando L298N circuito integrato, il chip in grado di pilotare motori passo-passo a due fasi o motore passo a quattro fasi, può pilotare due motori DC.

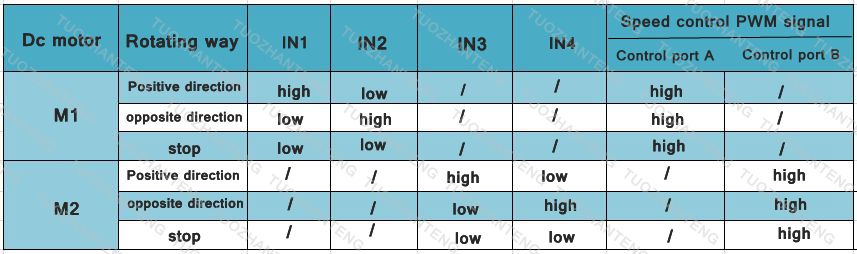


Caratteristiche:

1. il modulo utilizza L298N circuito integrato come l'unità principale, con la capacità di guida, a fuoco basso, funzionalità anti-jamming.

2. può essere utilizzato per prendere il potere attraverso il built-in 78M05 potenza di azionamento parte del lavoro, ma al fine di evitare danni al chip regolatore, quando un drive tensione superiore a 12 V, si prega di utilizzare un esterno 5 V di alimentazione logica.

3. Uses di alta-capacità di condensatori di filtro, a ruota libera diodo di protezione, affidabilità può essere migliorata.

  
  
Applicazione:



Nota:

1. quando si guida di tensione (la figura identificato come 12 V in ingresso, può accettare la gamma di ingresso reale è 7-12 V) a 7 V

-12 V, Si può attivare la on-board 5 V di alimentazione logica, quando si utilizza il bordo 5 V di alimentazione, interfaccia + 5 V

Non inserire la tensione di alimentazione, ma può portare a 5 V di tensione per uso esterno. (Questa è la routine usare!)

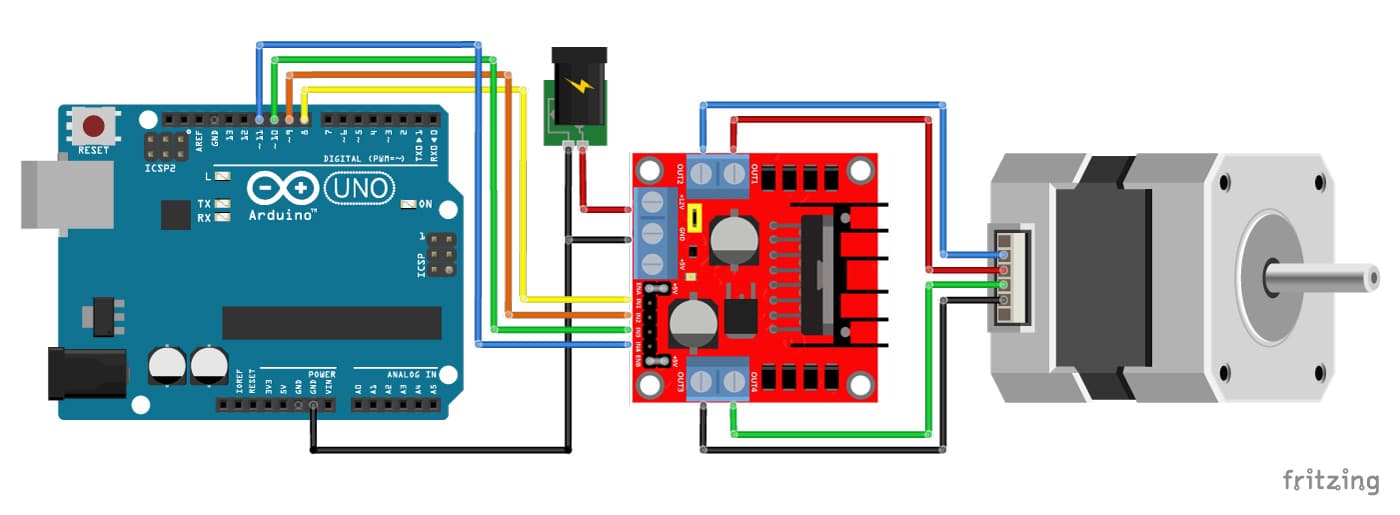
2. quando il drive tensione superiore a 12 V, pari a meno di 24 V (circuito integrato in grado di supportare manuale propone a 35 V, ma in conformità con un

298 applicazioni di controllo a 24 V tensione massima di sostegno Conservatore generale è stato molto grande! Tempo), come ad esempio per guidare la tensione nominale

18 V del motore. In primo luogo, è necessario scollegare il bordo 5 V di uscita consentono ponticello Cappellini. E poi 5 V 5 V porta di uscita di Accesso Esterno

L298N interna tensione logica circuiti. (Questo è un ad alta tensione driver non convenzionale applicazione)

# Control a stepper motor with L298N motor driver and Arduino



In this article you will learn how to control a stepper motor with the L298N Motor Driver. This driver board is usually used to control DC motors, but it is also an inexpensive alternative to control stepper motors! It can control both the speed and the spinning direction of most stepper motors like a NEMA 17.

I have included a wiring diagram and many example codes. In the first example we will look at the **Stepper.h** Arduino library. I highly recommend to also take a look at the example codes for the **AccelStepper** library at the end of this tutorial. This library is fairly easy to use and can greatly improve the performance of your hardware.

After each example, I break down and explain how the code works, so you should have no problems modifying it to suit your needs.

The Arduino Motor Shield Rev3 also uses a L298 driver.

## Information about the L298N Motor Driver

The [L298N Motor Driver Board](https://www.amazon.com/HiLetgo-Controller-Stepper-H-Bridge-Mega2560/dp/B07BK1QL5T/ref=as_li_ss_tl?keywords=l298n&qid=1562745291&s=gateway&sr=8-3&linkCode=ll1&tag=makerguides-20&linkId=3730d8f3d0b471b6cfb1a3acd6e96577&language=en_US) is built around the L298 dual full-bridge driver, made by STMicroelectronics. With this motor driver you can control DC motors, stepper motors, relays and solenoids. It comes with two separate channels, called A and B, that you can use to drive 2 DC motors, or 1 stepper motor when combined.

The L298N is usually mounted on a (red) breakout board, which makes wiring a lot easier. The breakout board also includes a 78M05 5 V power regulator.

### Why is my stepper motor getting HOT?

One thing that is very important to remember is that the L298 **does not have an easy way to set a current limit** unlike [other stepper motor drivers](https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/#how-to-set-the-current-limit). This means that the current draw depends on the relationship between the inductance and resistance (L/R) of the stepper motor that you connect to it. When the motor draws too much current, you can damage the driver and the motor will get hot!

What this means for you, is that you you need to be careful when selecting the stepper motor and power supply to use with this motor driver. **Not all stepper motors will work!** The L298N operating voltage is between 4.8 and 46 volts (max 35 V when mounted on the breakout board). Since the driver can supply a **maximum of 2 amperes per channel**, you need to find a stepper motor that can be used in this voltage range and doesn’t exceed the maximum current rating.

Check the datasheet of your stepper motor and look for the voltage/current draw of the motor. If you can’t find the datasheet, you can measure the resistance of one of the windings and use the following formula to get an estimation of the current draw:

I = U ÷ R or Current draw (A) = Supply voltage (V) ÷ Winding resistance (Ω)

I would try to find a motor that draws less than 2 A at the voltage that you want to use.

The motor I used for this tutorial draws around 1 A at 5 V. I also found [this stepper motor from Adafruit](https://amzn.to/2YjOKkJ) that works great at 12V and only draws 350 mA.

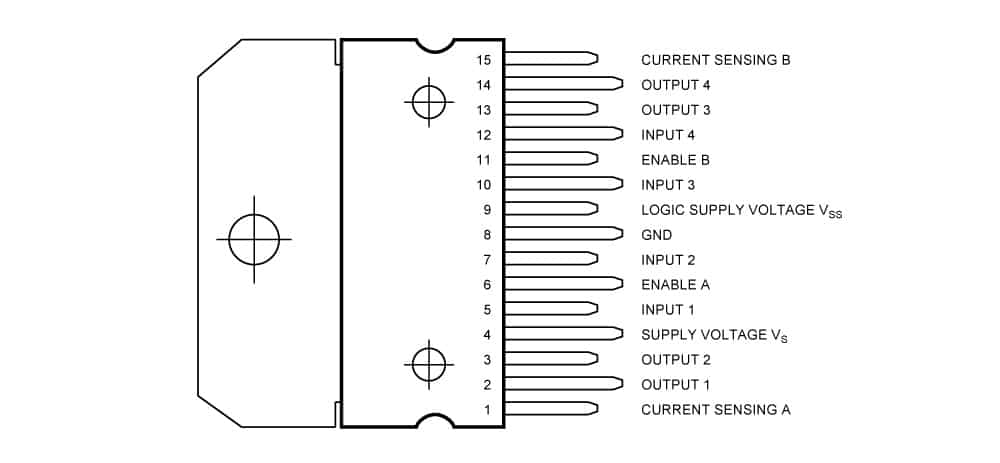
If the motor you want to drive doesn’t work with the L298N motor driver, it is best to use a chopper drive instead. I wrote tutorials for the [A4988](https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/) and [DRV8825](https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/) driver that work great with many stepper motors.

### L298N Motor Driver Specifications

|  |  |
| --- | --- |
| Operating voltage | 5 – 35 V |
| Logic voltage | 4.5 – 7 V |
| Max current | 2 A per channel or 4 A max |
| Motor controller | L298N, drives 2 DC motors or 1 stepper motor |
| Voltage regulator | 78M05 |
| Module dimensions | 43 x 43 x 28 mm |
| Hole dimensions | 3.2 mm, 37 mm spacing |
| Cost | [Check price](https://amzn.to/2ZDdDba) |

### L298N Pinout

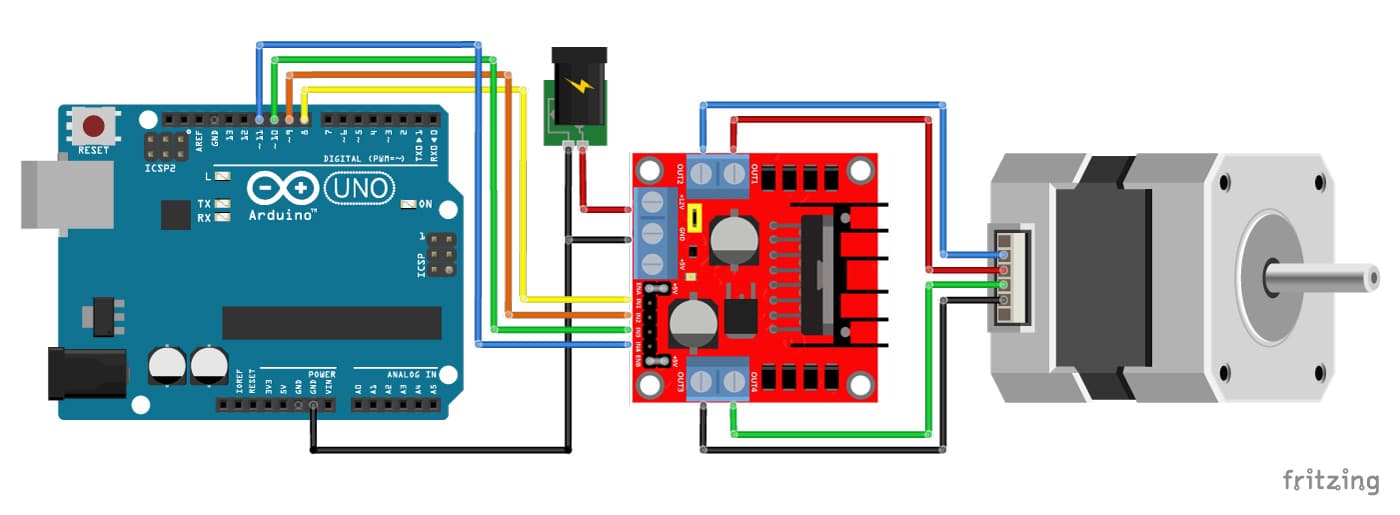
The L298 comes in several different packages, the pinout for the L298N (Multiwatt15) is given below:



|  |  |  |
| --- | --- | --- |
| L298N (Multiwatt15) pinout | | |
| Pin no. | **Name** | **Function** |
| 1, 15 | Sense A, Sense B | The sense resistor needs to be connected between this pin and GND (not used on breakout board). |
| 2, 3 | Out 1, Out2 | Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1. |
| 4 | Vs | Supply Voltage for the Power Output Stages |
| 5, 7 | Input 1, Input 2 | TTL Compatible Inputs of Bridge A |
| 6, 11 | Enable A, Enable B | TTL Compatible Enable Input: the LOW state disables the bridge A (enable A) and/or the bridge B (enable B). |
| 8 | GND | Ground |
| 9 | VSS | Supply Voltage for the Logic Blocks. |
| 10, 12 | Input 3, Input 4 | TTL Compatible Inputs of the Bridge B. |
| 13, 14 | Out 3, Out4 | Outputs of the Bridge B; the current that flows through the load connected between these two pins is monitored at pin 15. |

## Wiring – Connecting L298N to stepper motor and Arduino

The wiring diagram/schematic below shows you how to connect a stepper motor, power supply and Arduino to the L298N breakout board.

L298N motor driver with stepper motor and Arduino wiring diagram.

The connections are also given in the table below:

### L298N Connections

|  |  |
| --- | --- |
| L298N | Connection |
| +12V | 5 – 35 V power supply |
| GND | Power supply and Arduino ground |
| 12 V jumper | Remove if motor power > 12 V! |
| 5V+ (optional) | 5 V Arduino if 12 V jumper is removed |
| IN1 | Pin 8 Arduino |
| IN2 | Pin 9 Arduino |
| IN3 | Pin 10 Arduino |
| IN4 | Pin 11 Arduino |
| ENA and ENB jumper | Leave installed |
| OUT1 + OUT2 | Stepper motor coil A |
| OUT3 + OUT4 | Stepper motor coil B |

**Important note: remove the +12V jumper if you are using a power supply higher than 12 V.**

When the +12V jumper is attached, the on-board voltage regulator is enabled and it will create the 5 V logic voltage.

You also need to keep both the ENA and ENB jumpers in place so the the motor is always enabled.

### How to determine the stepper motor wiring?

If you can’t find the datasheet of your stepper motor, it can be difficult to figure out how to wire your motor correctly. I use the following trick to determine how to connect 4 wire bipolar stepper motors:

The only thing you need to identify is the two pairs of wires which are connected to each of the two coils. One coil gets connected to OUT1 and OUT2 and the other to OUT3 and OUT4, the polarity doesn’t matter.

To find the two wires from one coil, do the following with the motor disconnected:

1. Try to spin the shaft of the stepper motor by hand and notice how hard it is to turn.
2. Now pick a random pair of wires from the motor and touch the bare ends together.
3. Next, try to spin the shaft of the stepper motor again.
4. **If you feel a lot of resistance, you have found a pair of wires from the same coil.** If you can spin the shaft freely, try another pair of wires.

Now connect the two coils to the pins shown in the wiring diagram above.

(If it is still unclear, please leave a comment below, more info can also be found on the [*RepRap.org wiki*](https://reprap.org/wiki/Stepper_wiring))

## Stepper.h library example code for L298N driver with stepper motor and Arduino

You can upload the following example code to your Arduino using the [Arduino IDE](https://www.arduino.cc/en/main/software).

This example uses the **Stepper.h** library, which should come pre-installed with the Arduino IDE. You can find it by going to **Sketch > Include Library > Stepper**.

This sketch turns the stepper motor 1 revolution in one direction, pauses, and then turns 1 revolution in the other direction.

1. /\* Example sketch to control a stepper motor with L298N motor driver, Arduino UNO and Stepper.h library. More info: https://www.makerguides.com \*/
2. // Include the Stepper library:
3. #include <Stepper.h>
4. // Define number of steps per revolution:
5. **const** **int** stepsPerRevolution = 200;
6. // Initialize the stepper library on pins 8 through 11:
7. Stepper myStepper = Stepper(stepsPerRevolution, 8, 9, 10, 11);
8. **void** setup() {
9. // Set the motor speed (RPMs):
10. myStepper.setSpeed(100);
11. }
12. **void** loop() {
13. // Step one revolution in one direction:
14. myStepper.step(200);
15. delay(2000);
16. // Step on revolution in the other direction:
17. myStepper.step(-200);
18. delay(2000);
19. }

### How the code works:

The sketch starts by including the Stepper.h Arduino library. More information about this library can be found on the [Arduino website](https://www.arduino.cc/en/Reference/Stepper).

1. // Include the Stepper library:
2. #include <Stepper.h>

Next we need to define how many steps it takes for the motor to rotate 1 revolution. n this example we will be using the motor in **full step mode**. This means it takes 200 steps to rotate 360 degrees. You can change this value if you want if you are using a different type of stepper motor or setup.

1. // Define number of steps per revolution:
2. **const** **int** stepsPerRevolution = 200;

After this, you need to create a new instance of the Stepper class, which represents a particular stepper motor connected to the Arduino. For this we use the function Stepper(steps, pin1, pin2, pin3, pin4) where steps is the number of steps per revolution and pin1 through pin4 are the pins used to drive the stepper motor. In our case these are pins 8, 9, 10 and 11.

1. // Initialize the stepper library on pins 8 through 11:
2. Stepper myStepper = Stepper(stepsPerRevolution, 8, 9, 10, 11);

In this case I called the stepper motor ‘myStepper’ but you can use other names as well, like ‘z\_motor’ or ‘liftmotor’ etc. Stepper liftmotor = Stepper(stepsPerRevolution, 8, 9, 10, 11);. The name ‘myStepper’ will be used to set the speed and number of steps for this particular motor. Note that you can create multiple stepper objects with different names if you want to control more than one motor.

In the setup() we define the speed of the motor. You can set the speed of the motor in **RPM** with the function setSpeed(rpm). I set it to 100, so we should see around 1.6 revolutions per second.

1. // Set the motor speed (RPMs):
2. myStepper.setSpeed(100);

In the loop section of code, we simply call the step(steps) function which turns the motor a specific number of steps at a speed determined by the setSpeed(rpm) function. Passing a negative number to this function reverses the spinning direction of the motor.

1. **void** loop() {
2. // Step one revolution in one direction:
3. myStepper.step(200);
4. delay(2000);
5. // Step on revolution in the other direction:
6. myStepper.step(-200);
7. delay(2000);
8. }

Note that the **step(steps)** function is **blocking**, this means it will wait until the motor has finished moving to pass control to the next line in your sketch.

## Installing the AccelStepper library

In the following three examples I will show you how you can control both the speed, the direction and the number of steps the stepper motor should take. In this example I will be using the **AccelStepper** library.

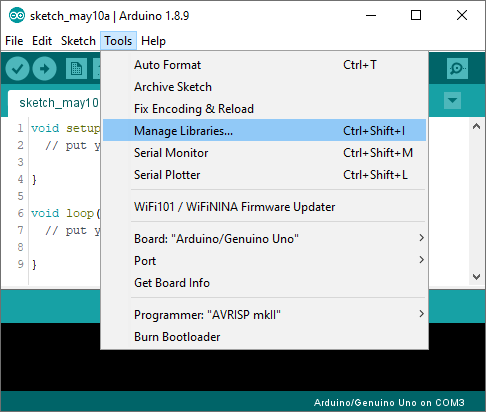
The AccelStepper library written by Mike McCauley is an awesome library to use for your project. One of the advantages is that it supports acceleration and deceleration, but it has a lot of other nice functions too.

You can download the latest version of this library [here](https://www.airspayce.com/mikem/arduino/AccelStepper/files.html) or click the button below.

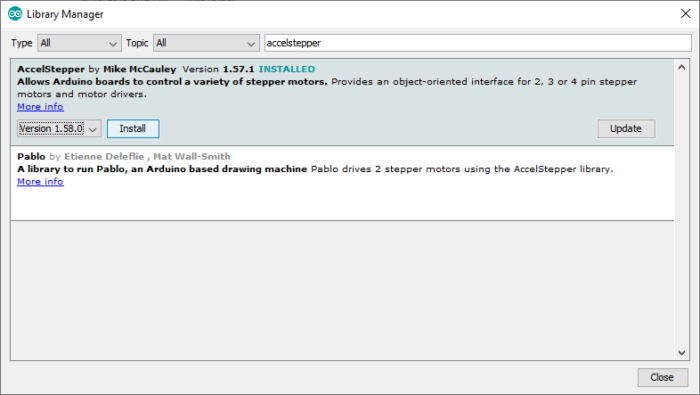
[AccelStepper-1.59.zip](https://www.makerguides.com/wp-content/uploads/2019/02/AccelStepper-1.59.zip)

You can install the library by going to**Sketch > Include Library > Add .ZIP Library…**in the Arduino IDE.

Another option is to navigate to **Tools > Manage Libraries…** or type Ctrl + Shift + I on Windows. The Library Manager will open and update the list of installed libraries.



You can search for ‘**accelstepper**‘ and look for the library by Mike McCauley. Select the latest version and then click Install.



**1. Continuous rotation AccelStepper example code**

The following sketch can be used to run one or more stepper motors continuously at a constant speed. (No acceleration or deceleration is used).

1. /\* Example sketch to control a stepper motor with L298N motor driver, Arduino UNO and AccelStepper.h library. Contiuous rotation. More info: https://www.makerguides.com \*/
2. // Include the AccelStepper library:
3. #include <AccelStepper.h>
4. // Define the AccelStepper interface type:
5. #define MotorInterfaceType 4
6. // Create a new instance of the AccelStepper class:
7. AccelStepper stepper = AccelStepper(MotorInterfaceType, 8, 9, 10, 11);
8. **void** setup() {
9. // Set the maximum speed in steps per second:
10. stepper.setMaxSpeed(1000);
11. }
12. **void** loop() {
13. // Set the speed of the motor in steps per second:
14. stepper.setSpeed(500);
15. // Step the motor with constant speed as set by setSpeed():
16. stepper.runSpeed();
17. }

### How the code works:

The first step is to include the library with #include <AccelStepper.h>.

1. // Include the AccelStepper library:
2. #include <AccelStepper.h>

The next step is to define the motor interface type. The motorinterface type must be set to 4 when using a 4 wire stepper motor in full step mode (200 steps/revolution). You can find the other interface types [here](https://www.airspayce.com/mikem/arduino/AccelStepper/classAccelStepper.html#a608b2395b64ac15451d16d0371fe13ce).

The statement #define is used to give a name to a constant value. The compiler will replace any references to this constant with the defined value when the the program is compiled. So everywhere you mention motorInterfaceType, the compiler will replace it with the value 4 when the program is compiled.

1. // Define the AccelStepper interface type:
2. #define MotorInterfaceType 4

Next, you need to create a new instance of the AccelStepper class with the appropriate motor interface type and connections.

In this case I called the stepper motor ‘stepper’ but you can use other names as well, like ‘z\_motor’ or ‘liftmotor’ etc. AccelStepper liftmotor = AccelStepper(motorInterfaceType, 8, 9, 10, 11);. As you saw in the previous example, the name that you give to the stepper motor will be used later to set the speed, position and acceleration for that particular motor. You can create multiple instances of the AccelStepper class with different names and pins. This allows you to easily control 2 or more stepper motors at the same time.

1. // Create a new instance of the AccelStepper class:
2. AccelStepper stepper = AccelStepper(MotorInterfaceType, 8, 9, 10, 11);

In the setup() section of the code we define the maximum speed in steps/second. Speeds of more than 1000 steps per second can be unreliable, so I set this as the maximum. Note that I specify the name of the stepper motor (‘stepper’), for which I want to define the maximum speed. If you have multiple stepper motors connected, you can specify a different speed for each motor:

1. **void** setup() {
2. // Set the maximum speed in steps per second:
3. stepper.setMaxSpeed(1000);
4. stepper2.setMaxSpeed(300);
5. }

In the loop() we first set the speed that we want the motor to run at. For this we use the function setSpeed(). (you can also place this in the setup section of the code).

stepper.runSpeed() polls the motor and when a step is due, executes 1 step. This depends on the set speed and the time since the last step. If you want to change the direction of the motor, you can set a negative speed: stepper.setSpeed(-400); turns the motor the other way.

1. **void** loop() {
2. // Set the speed of the motor in steps per second:
3. stepper.setSpeed(500);
4. // Step the motor with constant speed as set by setSpeed():
5. stepper.runSpeed();
6. }

**2. Example code to control number of steps or revolutions**

With the following sketch you can control both the speed, direction and the number of steps/revolutions.

In this case, the stepper motor turns 2 revolutions clockwise with 200 steps/sec, then turns 1 revolution counterclockwise at 600 steps/sec, and lastly turns 3 revolutions clockwise at 400 steps/sec.

1. /\* Example sketch to control a stepper motor with L298N motor driver, Arduino UNO and AccelStepper.h library. Number of steps or revolutions. More info: https://www.makerguides.com \*/
2. // Include the AccelStepper library:
3. #include <AccelStepper.h>
4. // Define the AccelStepper interface type:
5. #define MotorInterfaceType 4
6. // Create a new instance of the AccelStepper class:
7. AccelStepper stepper = AccelStepper(MotorInterfaceType, 8, 9, 10, 11);
8. **void** setup() {
9. // Set the maximum steps per second:
10. stepper.setMaxSpeed(1000);
11. }
12. **void** loop() {
13. // Set the current position to 0:
14. stepper.setCurrentPosition(0);
15. // Run the motor forward at 200 steps/second until the motor reaches 400 steps (2 revolutions):
16. **while** (stepper.currentPosition() != 400) {
17. stepper.setSpeed(200);
18. stepper.runSpeed();
19. }
20. delay(1000);
21. // Reset the position to 0:
22. stepper.setCurrentPosition(0);
23. // Run the motor backwards at 600 steps/second until the motor reaches -200 steps (1 revolution):
24. **while** (stepper.currentPosition() != -200) {
25. stepper.setSpeed(-600);
26. stepper.runSpeed();
27. }
28. delay(1000);
29. // Reset the position to 0:
30. stepper.setCurrentPosition(0);
31. // Run the motor forward at 400 steps/second until the motor reaches 600 steps (3 revolutions):
32. **while** (stepper.currentPosition() != 600) {
33. stepper.setSpeed(400);
34. stepper.runSpeed();
35. }
36. delay(3000);
37. }

### Code explanation:

The first part of the code up to the loop() section is exactly the same as in the previous example.

In the loop I make use of a [while loop](https://www.arduino.cc/reference/en/language/structure/control-structure/while/) in combination with the currentPosition() function. First, I set the current position of the stepper motor to zero with stepper.setCurrentPosition(0).

1. // Set the current position to 0:
2. stepper.setCurrentPosition(0);

Next we make use of the while loop. A while loop will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. So in this case I check if the current position of the stepper motor is not equal to 200 steps (!= means: is not equal to). While this is not the case, we run the stepper motor at a constant speed as set by setSpeed().

1. // Run the motor forward at 200 steps/second until the motor reaches 400 steps (2 revolutions):
2. **while** (stepper.currentPosition() != 400) {
3. stepper.setSpeed(200);
4. stepper.runSpeed();
5. }

In the rest of the loop, we do exactly the same, just with a different speed and target position.

**3. Acceleration and deceleration example code**

In this example we will look at one of the main reasons to use the AccelStepper library.

With the following sketch you we can add acceleration and deceleration to the movements of the stepper motor, without any complicated coding. The first section of this sketch is the same as in example 1, but the setup and the loop are different.

The motor will run five revolutions back and forth with a speed of 200 steps per second and an acceleration of 50 steps/second2.

1. /\* Example sketch to control a stepper motor with L298N motor driver, Arduino UNO and AccelStepper.h library. Acceleration and deceleration. More info: https://www.makerguides.com \*/
2. // Include the AccelStepper library:
3. #include <AccelStepper.h>
4. // Define the AccelStepper interface type:
5. #define MotorInterfaceType 4
6. // Create a new instance of the AccelStepper class:
7. AccelStepper stepper = AccelStepper(MotorInterfaceType, 8, 9, 10, 11);
8. **void** setup() {
9. // Set the maximum steps per second:
10. stepper.setMaxSpeed(200);
11. // Set the maximum acceleration in steps per second^2:
12. stepper.setAcceleration(50);
13. }
14. **void** loop() {
15. // Set target position:
16. stepper.moveTo(1000);
17. // Run to position with set speed and acceleration:
18. stepper.runToPosition();
19. delay(1000);
20. // Move back to original position:
21. stepper.moveTo(0);
22. // Run to position with set speed and acceleration:
23. stepper.runToPosition();
24. delay(1000);
25. }

### How the code works:

In the setup(), besides the maximum speed, we need to define the acceleration/deceleration. For this we use the function setAcceleration().

1. // Set the maximum steps per second:
2. stepper.setMaxSpeed(200);
3. // Set the maximum acceleration in steps per second^2:
4. stepper.setAcceleration(50);

In the loop section of the code, I used a different way to let the motor rotate a predefined number of steps. First I set the target position with the function moveTo(). Next, we simply use the function runToPosition() to let the motor run to the target position with the set speed and acceleration. The motor will decelerate before reaching the target position.

1. // Set target position:
2. stepper.moveTo(1000);
3. // Run to position with set speed and acceleration:
4. stepper.runToPosition();

Finally, we set the new target position back to the 0, so that we return to the origin.