

DRIVER PER PILOTARE MOTORI DC CON ARDUINO

- L298N Dual H-Bridge Motor Controller
- L9110S Dual-Channel Driver Module
 - Modulo IRF520 MOSFET Switch
- BTS7960 H-Bridge Motor HIGH POWER

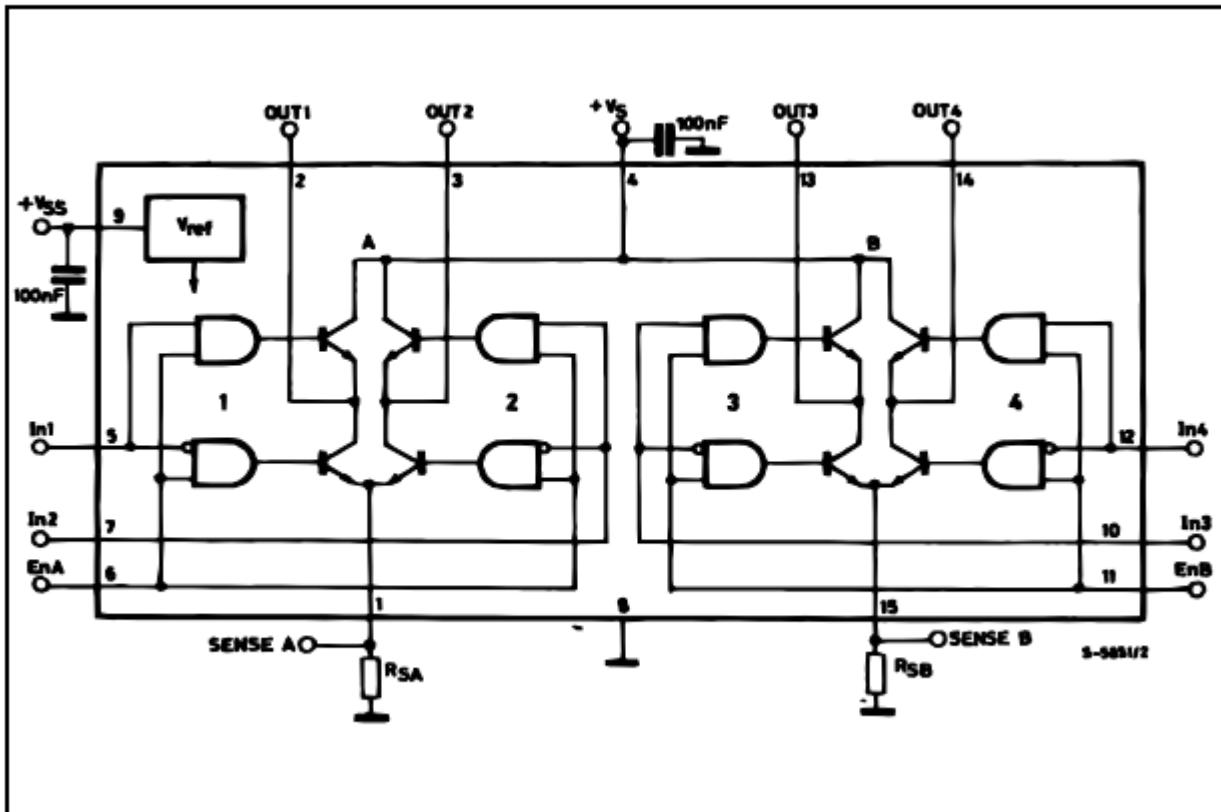
L298N Dual H-Bridge Motor Controller

Tra i tanti moduli ora in commercio per pilotare dei motori elettrici in corrente continua se ne può trovare una denominata **L298N Dual H-Bridge Motor Controller** di piccole dimensioni ed estremamente compatta misura solo 4.3 cm x 4.3 cm x 2.7 cm e dal peso di soli 28 gr.

Si tratta di una scheda driver con finale **L298** della **STMicroelectronics**, al suo interno troviamo due ponti H integrati, che supportano un elevato voltaggio (teoricamente sino a 46V) ed elevate correnti (2A per ponte) e che possono essere pilotati con livelli in logica TTL. Ciascun ponte può essere disabilitato o abilitato tramite il relativo piedino di enable per comandare un motore passo passo o due motori DC direttamente da Arduino o altri microcontrollori .

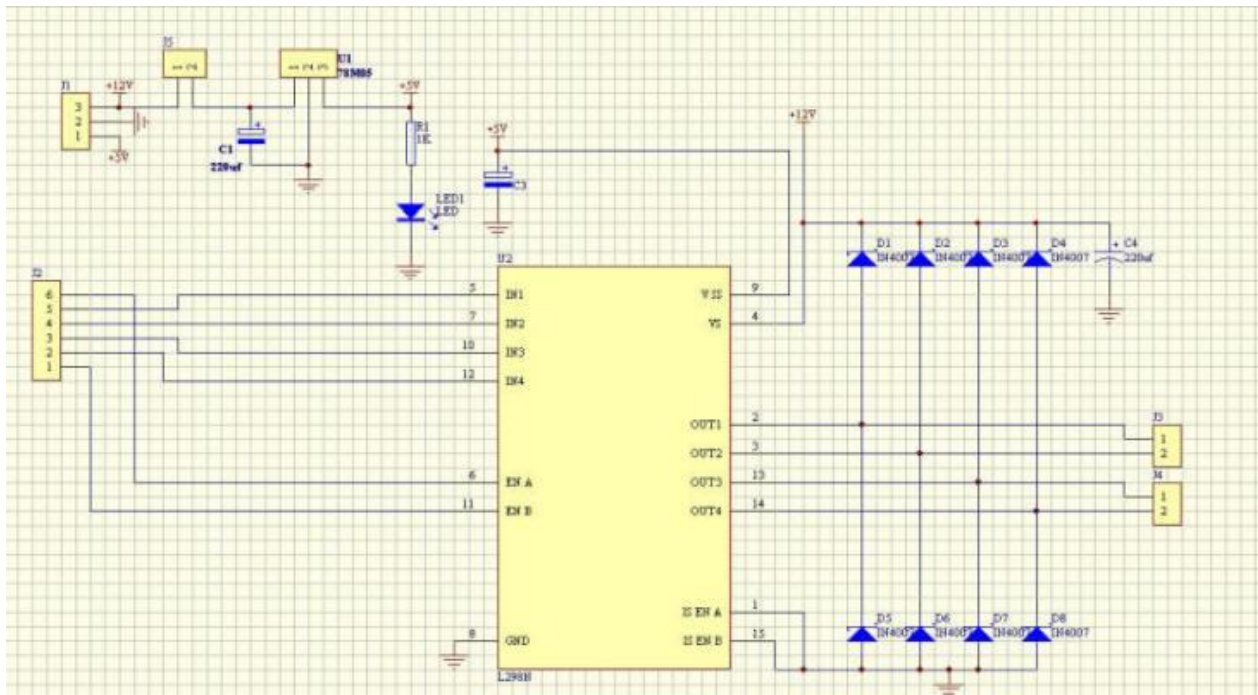


BLOCK DIAGRAM

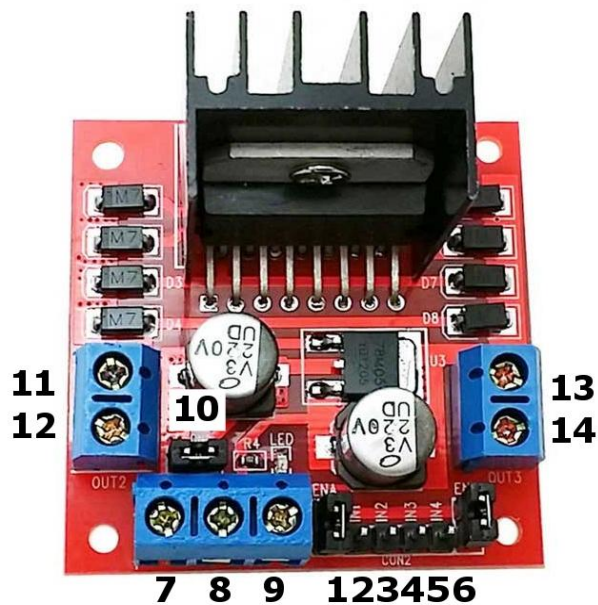


Sulla scheda, oltre all' integrato L298, troviamo i diodi di ricircolo, per la sua alimentazione è richiesta una doppia alimentazione: 5Vdc per la logica e una tensione compresa tra 5 e 35Vdc per l'alimentazione del motore, la corrente in uscita è di ben 2A su ogni canale.

Schema elettrico del modulo L298N Dual H-Bridge Motor Controller



Descrizione pin



N°	Descrizione
1	ENA - ponticello di abilitazione motore a corrente continua A Non rimuovere nel caso si utilizzi un motore passo-passo. Connettersi a un'uscita PWM per il controllo della velocità del motore DC.
2	IN 1
3	IN 2

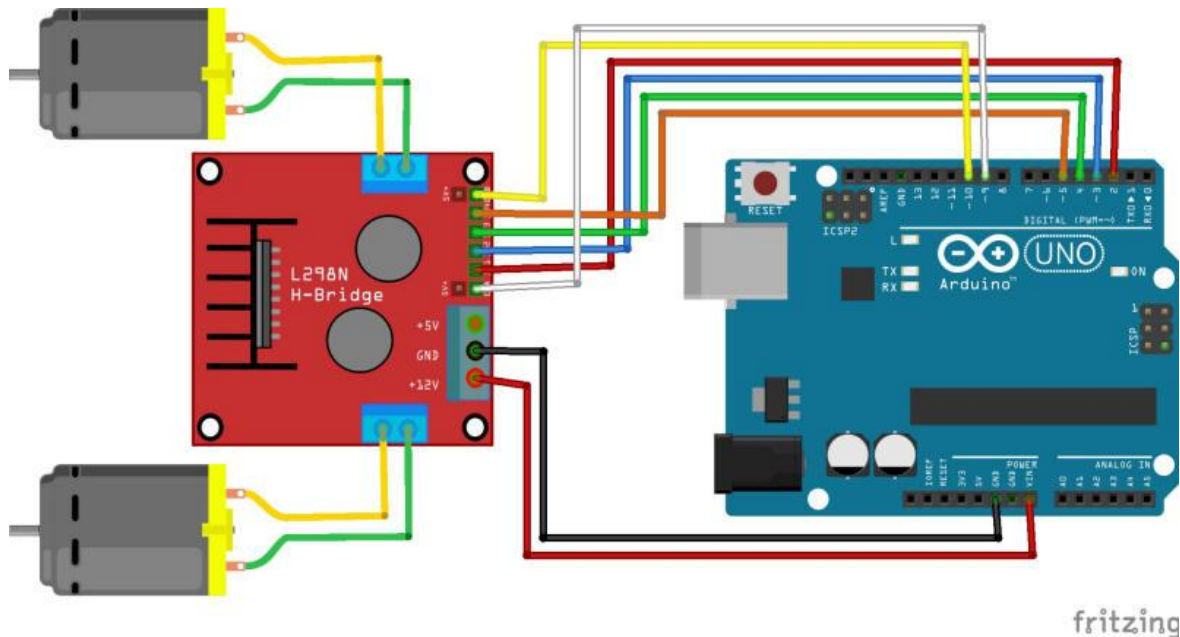
4	IN 3
5	IN 4
6	ENB - ponticello di abilitazione motore a corrente continua B Non rimuovere nel caso si utilizzi un motore passo-passo. Connettersi a un'uscita PWM per il controllo della velocità del motore DC.
7	Collegare la tensione di alimentazione del motore, massima di 35V DC. Rimuovere il ponticello [10] se la tensione è > 12V DC
8	GND
9	uscita 5V se 12V ponticello in luogo, ideale per alimentare il vostro Arduino (etc)
10	jumper 12V - rimuovere questo se si utilizza una tensione di alimentazione superiore a 12V DC. Ciò consente l'alimentazione tramite il regolatore 5V di bordo
11	DC motor 1 "+" o motore passo-passo A +
12	motore DC 1 "-" o motore passo-passo A-
13	motore a corrente continua 2 "+" o motore passo-passo B +
14	motore DC 2 "-" o motore passo-passo B-

Gestione di due motori DC

Per controllare uno o due motori a corrente continua è abbastanza facile con il [L298N Dual H-Bridge Motor Controller](#). Sotto vediamo i principali componenti necessari



Schema di collegamento



- Prima collegare ogni motore alle connessioni A e B sul modulo L298N [pin 11-12/13-14], se si utilizzano due motori per esempio su un robot, assicurarsi che la polarità dei motori sia la stessa su entrambi gli ingressi. In caso contrario, la rotazione potrebbe non essere corretta, per questa sarà sufficiente invertire i cavi
- Collegare l'alimentazione, il positivo al pin [7] sul modulo e il negativo / GND al pin [8]. Se si fornisce è fino a 12 V è possibile lasciare nel ponticello [10] i 5V saranno disponibile al pin [9] del modulo. L'alimentazione sarà in questo caso prelevata dal pin Vin di Arduino, alimentandolo attraverso il plug della scheda.
- Ora sarà necessario collegare i cavi per la gestione dei motore ai sei pin di uscita digitali su Arduino, due dei quali hanno bisogno di essere **PWM (Pulse Width Modulation)** pin. I pin PWM sono indicati con la tilde ("~") accanto al numero di pin, sono pin PWM il 3, 5, 6, 10 e 11.

Collegeremo quindi i pin digitali D3, D4, D5 e D6 che saranno collegati ai pin IN1, IN2, IN3 e IN4, rispettivamente. Quindi collegare D9 al modulo pin 1 ENA (rimuovere prima il ponticello) e D10 a PIN 6 del modulo ENB (di nuovo, rimuovere prima il ponticello).

La direzione del motore è controllata inviando un segnale alto o basso per l'azionamento per ogni motore (o canale). Ad esempio per il motore uno, un HIGH a IN1 e IN2 LOW a causerà la svolta in una direzione, e LOW e HIGH causerà la svolta in direzione opposta.

Tuttavia i motori non si accende fino a quando un segnale HIGH non è inviato per abilitare il pin 1 per il motore, oppure al pin 6 per il motore 2. I motori possono essere disattivati con un segnale LOW allo stesso pin. Tuttavia, se avete bisogno di controllare la velocità dei motori, il segnale PWM dal pin digitale collegata al pin di di abilitazione permette di regolare la velocità.

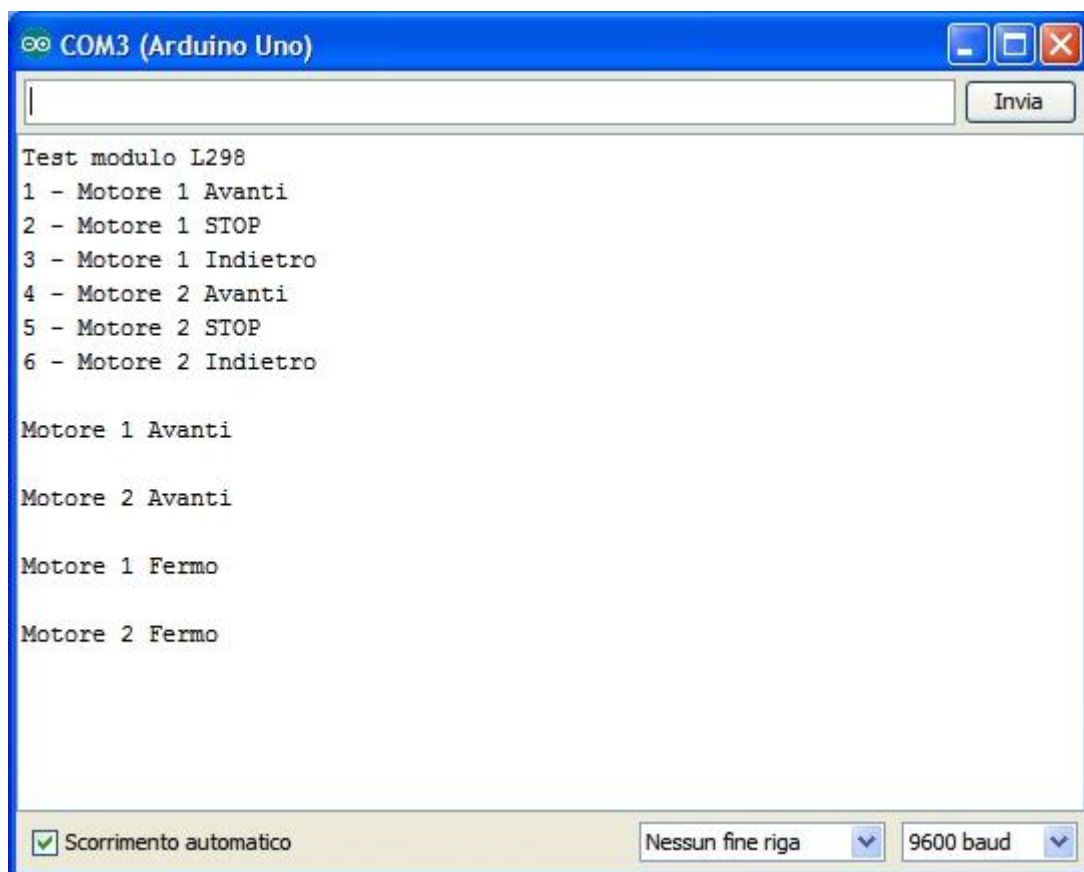
Programma per il test

Il programma di test proposto, permette di comandare la rotazione, cambio di direzione, STOP dei due motori, inviando da tastiera l'opportuno comando.

Modificando il valore della variabile **speed**, di default impostata a 200, si varierà la velocità di rotazione di entrambi i motori.

Tasti di controllo:

- 1 -Motor 1 Avanti
- 2 -Motor 1 FERMO
- 3 -Motor 1 Indietro
- 4 -Motor 2 Avanti
- 5 -Motor 2 FERMO
- 6 -Motor 2 Indietro



```
Test_2_motore-L298 | Arduino 1.7.7
File Modifica Sketch Strumenti Aiuto
Test_2_motore-L298 $
// Motore 1
int dir1PinA = 2;
int dir2PinA = 3;
int speedPinA = 9; // Deve essere un pino PWM per essere in grado di contr

// Motore 2
int dir1PinB = 4;
int dir2PinB = 5;
int speedPinB = 10; // Deve essere un pino PWM per essere in grado di cont

Caricamento completato

Le variabili globali usano 366 byte (17%) di memoria dinamica, lasciando
altri 1.682 byte liberi per le variabili locali. Il massimo è 2.048 byte.

34 Arduino Uno on COM3
```

```
/*
Test_2_motore-L298.ino
Programma per testare il comando di due motori DC tramite un
L298N Dual H-Bridge Motor Controller

Vengono utilizzati i seguenti pin
Pin Vin          -> Alimentazione motori
Pin GND          -> Pin GND
Pin Digital 2    -> Collegato al pin IN1
Pin Digital 3    -> Collegato al pin IN2
Pin Digital 4    -> Collegato al pin IN3
Pin Digital 5    -> Collegato al pin IN4
Pin Digital 9    -> Collegato al pin ENA
Pin Digital 10   -> Collegato al pin ENB

Tasti di controllo:
1 -Motor 1 Avanti
2 -Motor 1 FERMO
3 -Motor 1 Indietro
4 -Motor 2 Avanti
5 -Motor 2 FERMO
6 -Motor 2 Indietro
*/

// Imposta i pin di controllo del L298N Dual H-Bridge Motor Controller

// Motore 1
int dir1PinA = 2;
int dir2PinA = 3;
int speedPinA = 9; // Deve essere un pino PWM per essere
                  //in grado di controllare la velocità del motore

// Motore 2
int dir1PinB = 4;
int dir2PinB = 5;
int speedPinB = 10; // Deve essere un pino PWM per essere in grado
```



```

// di controllare la velocità del motore

void setup() { // Installazione eseguita una volta al ripristino
// initialize serial communication @ 9600 baud:
Serial.begin(9600);

//Definisce il pins del L298N Dual H-Bridge Motor Controller

pinMode(dir1PinA, OUTPUT);
pinMode(dir2PinA, OUTPUT);
pinMode(speedPinA, OUTPUT);
pinMode(dir1PinB, OUTPUT);
pinMode(dir2PinB, OUTPUT);
pinMode(speedPinB, OUTPUT);

Serial.println("Test modulo L298");
Serial.println("1 - Motore 1 Avanti");
Serial.println("2 - Motore 1 STOP");
Serial.println("3 - Motore 1 Indietro");
Serial.println("4 - Motore 2 Avanti");
Serial.println("5 - Motore 2 STOP");
Serial.println("6 - Motore 2 Indietro");
Serial.println(" ");
}

void loop() {

// Inizializza l'interfaccia seriale:

if (Serial.available() > 0) {
int inByte = Serial.read();
//int speed; // Local variable
int speed = 200;
switch (inByte) {

// _____ Motore 1 _____

case '1': // Motor 1 Avanti
analogWrite(speedPinA, speed); //Imposta la velocità via PWM
digitalWrite(dir1PinA, LOW);
digitalWrite(dir2PinA, HIGH);
Serial.println("Motore 1 Avanti"); // Scrive sul monitor seriale
Serial.println(" "); // Crea una linea bianca sul monitor seriale
break;

case '2': // Motor 1 Fermo
analogWrite(speedPinA, 0);
digitalWrite(dir1PinA, LOW);
digitalWrite(dir2PinA, LOW);
Serial.println("Motore 1 Fermo");
Serial.println(" ");
break;

case '3': // Motor 1 Indietro
analogWrite(speedPinA, speed);
digitalWrite(dir1PinA, HIGH);
digitalWrite(dir2PinA, LOW);
Serial.println("Motore 1 Indietro");
Serial.println(" ");
break;

// _____ Motore 2 _____

case '4': // Motore 2 Avanti
analogWrite(speedPinB, speed);
digitalWrite(dir1PinB, LOW);
digitalWrite(dir2PinB, HIGH);

```

```
    Serial.println("Motore 2 Avanti");
    Serial.println("  ");
    break;

case '5': // Motore 1 FERMO
    analogWrite(speedPinB, 0);
    digitalWrite(dir1PinB, LOW);
    digitalWrite(dir2PinB, LOW);
    Serial.println("Motore 2 Fermo");
    Serial.println("  ");
    break;

case '6': // Motore 2 Indietro
    analogWrite(speedPinB, speed);
    digitalWrite(dir1PinB, HIGH);
    digitalWrite(dir2PinB, LOW);
    Serial.println("Motore 2 Indietro");
    Serial.println("  ");
    break;

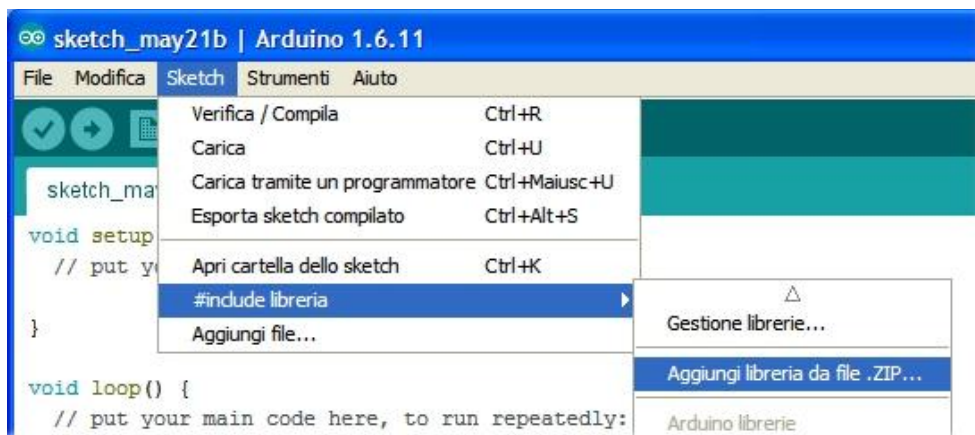
default:
    // Imposta tutti i pin a OFF se si preme un tasto non mappato:
    for (int thisPin = 2; thisPin < 11; thisPin++) {
        digitalWrite(thisPin, LOW);
    }
}
}
```

Libreria di gestione per Arduino

Per semplificare la gestione del modulo, è possibile utilizzare una libreria realizzata da **yohendry**, con questa libreria il codice è molto semplificato.

Nella prima parte sono assegnati i vari pin, viene poi creato l'oggetto controllare un **motore passo-passo** o **stepper** bipolare con l'utilizzo del **L298N Dual H-Bridge Motor Controller**. sono necessari i principali componenti sotto riportati

Prima di utilizzare la libreria, la si dovrà scaricare accedendo [a questo link](#)
Una volta scaricata la libreria in formato zip si dovrà andare al menu Sketch -> #Include Libreria -> Aggiungi libreria da file .ZIP e aggiungere la libreria appena scaricata



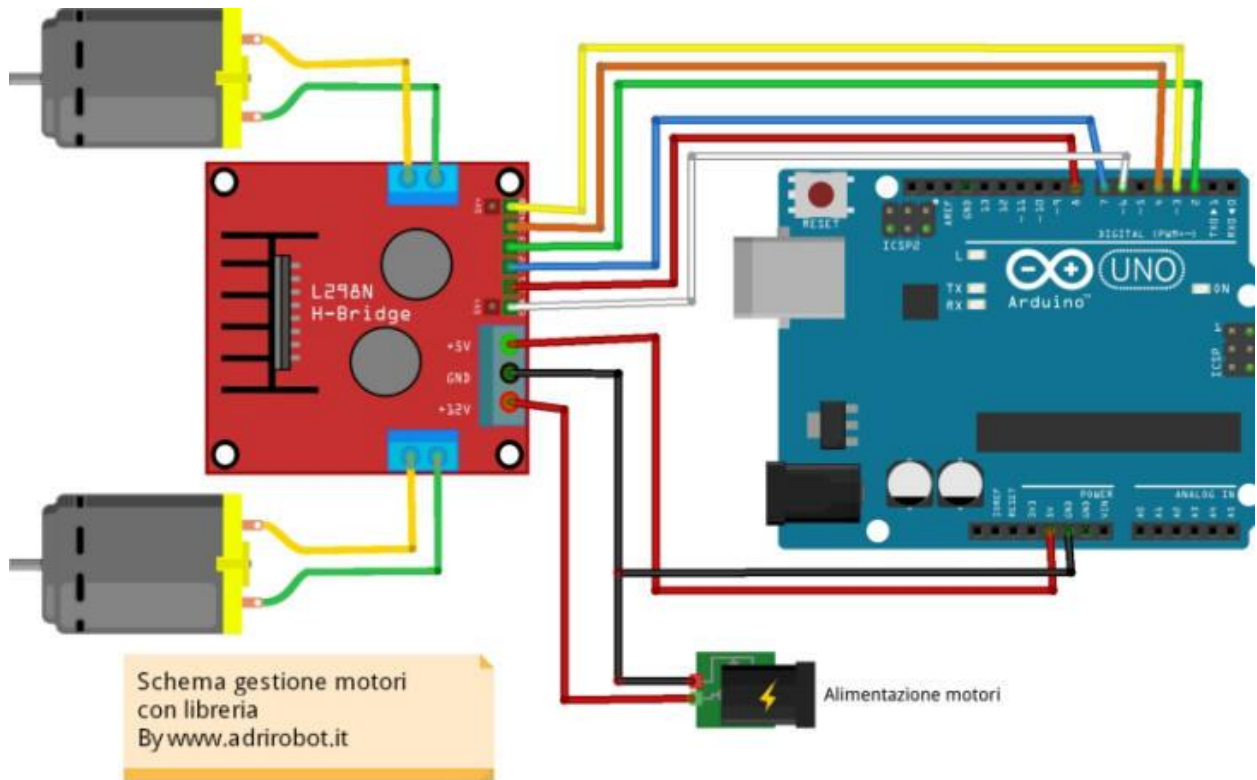
Nella sezione esempi della libreria potrete caricare questo esempio che gestisce direttamente due motori DC

```
#include <L298N.h>
const int ENA = 6;
const int IN1 = 8;
const int IN2 = 7;
const int IN3 = 2;
const int IN4 = 4;
const int ENB = 3;
L298N
driver(ENA, IN1, IN2, IN3, IN4, ENB);
int time_delay = 500;
int speed = 150;
void setup()
{
}

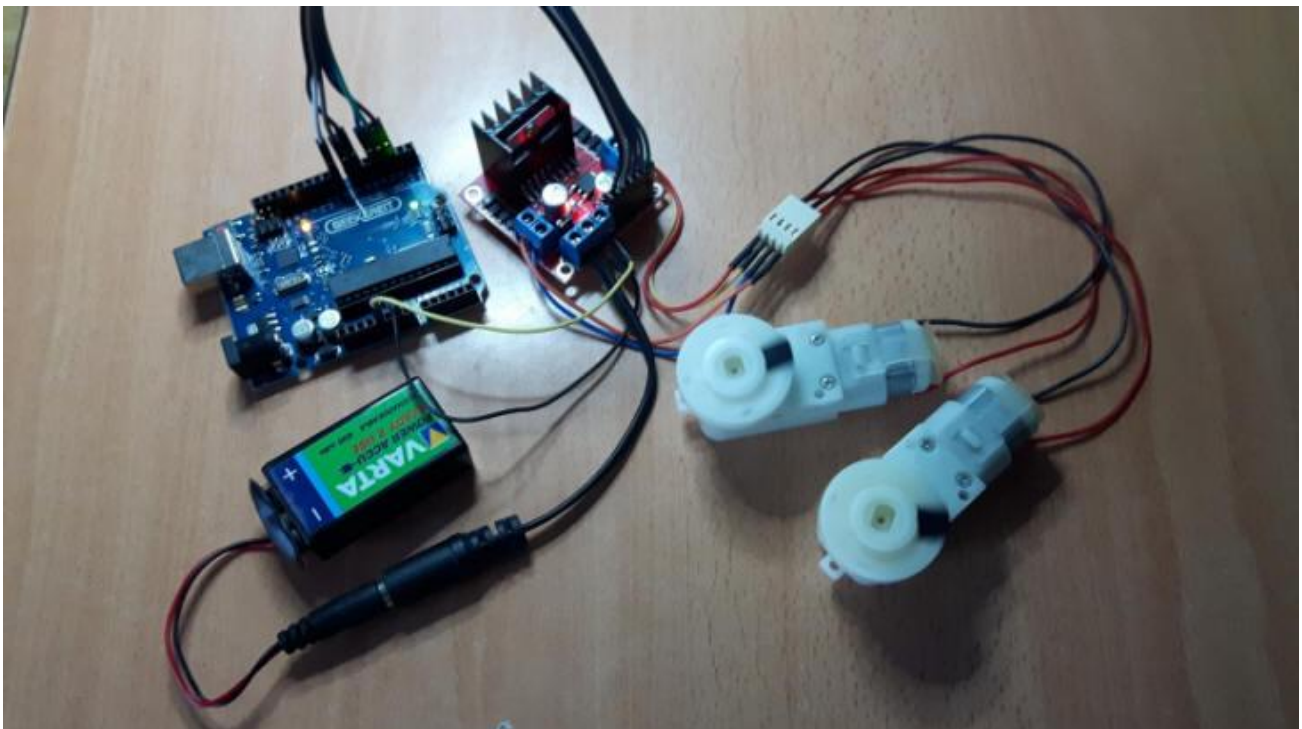
void loop()
{
  driver.forward(speed, time_delay);
  driver.full_stop(time_delay);

  driver.turn_right(speed, time_delay);
  driver.full_stop(time_delay);
  driver.turn_left(speed, time_delay);
  driver.full_stop(time_delay);
  driver.backward(speed, time_delay);
}
```

Schema di collegamento



fritzing



Gestione di un motore stepper

Per controllare un motore passo-passo o stepper bipolare con l'utilizzo del L298N Dual H-Bridge Motor Controller, sono necessari i principali componenti sotto riportati



**Scheda controllo
Arduino UNO R3**



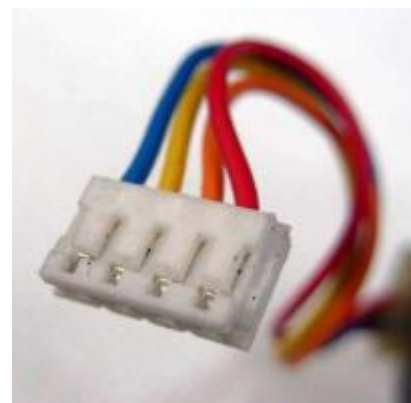
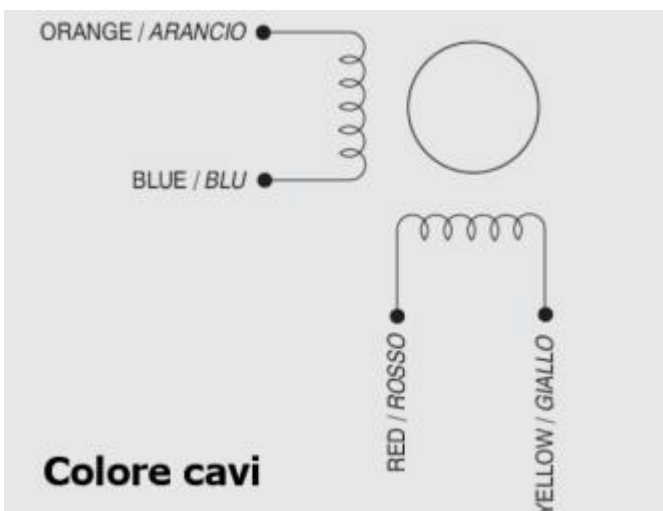
**L298N Dual H-Bridge
Motor Controller**



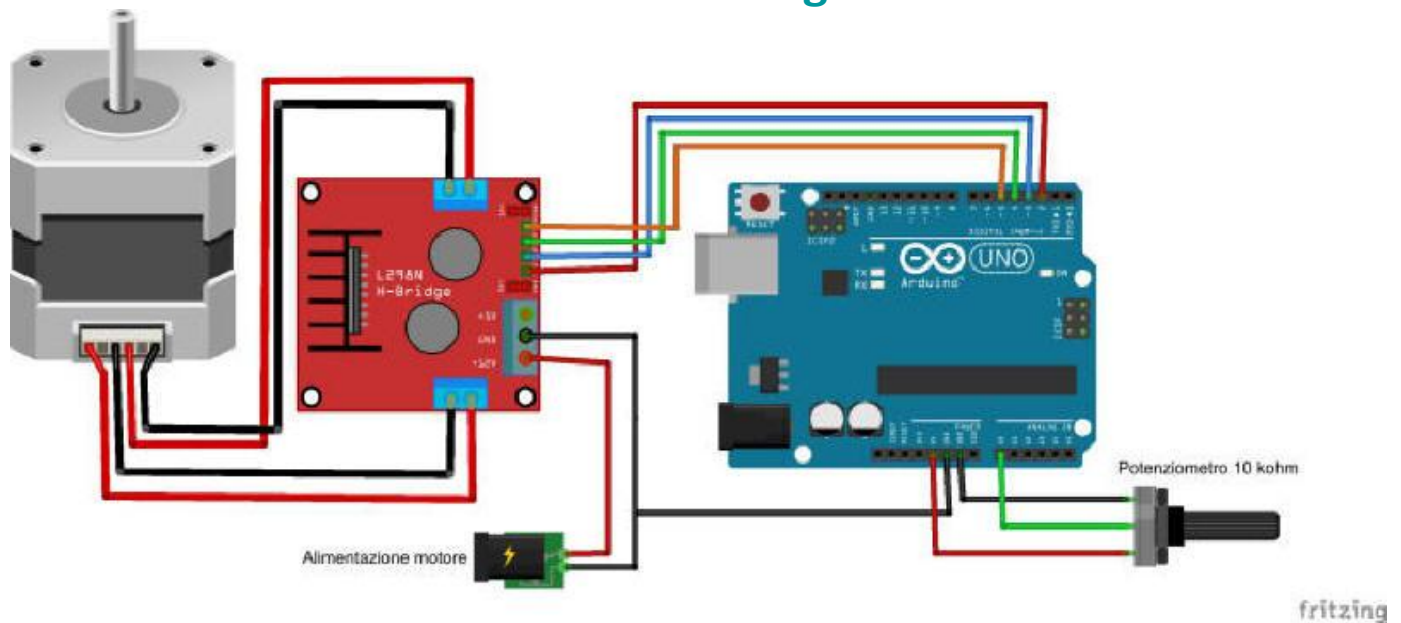
**Stepper Bipolare
SANYO DENKI
Tipo 103-546-6843**



**Potenzionometro rotativo
lineare da 10 Kohm**



Schema di collegamento



- Prima collegare il motore passo-passo alle connessioni A e B sul modulo L298N [pin 11-12/13-14], con i cavi che fanno capo a una bobina alla connessione A e gli altri alla connessione B.
- Collegare l'alimentazione del motore, il positivo al pin [7] sul modulo e il negativo / GND al pin [8].
Se si fornisce è fino a 12 V è possibile lasciare nel ponticello [10]. Collegare il negativo pin [8] anche al relativo pin GND di Arduino.
- L'alimentazione di Arduino sarà fornita attraverso il plug della scheda per esempio tramite una batteria 9V.
- Per quanto riguarda i pin ENA e ENB dovranno essere collegati al +5V tramite i ponticelli, in questo modo le uscite saranno sempre abilitate.
- Collegheremo quindi i pin digitali D3, D4, D5 e D6 che saranno collegati ai pin IN1, IN2, IN3 e IN4, rispettivamente.
- Collegheremo poi il potenziometro da 10 KΩ: il pin centrale all'ingresso Analogico A0 di Arduino, quindi i due pin laterali del potenziometro al pin +5V e GND di Arduino.

Programma per il test

Il programma di test proposto, permette di controllare la velocità di rotazione del motore passo-passo agendo sul potenziometro. Il programma fa uso della libreria [Stepper.h](#) che permette di gestire sia [stepper unipolari](#) che [bipolari](#). Sono disponibili i seguenti comandi/istanze.


Stepper (steps, pin1, pin2, pin3, pin4) crea l'istanza per comandare il motore, **steps** è il numero di passi che deve fare un motore per compiere un giro del motore.

Se è disponibile il numero di gradi per passo, dividere il numero per 360 per ottenere il numero di steps (ad esempio $360 / 1,8$ dà 200 steps). pin1, pin2, pin3, pin4

sono il numero dei pin di Arduino per pilotare il motore

setSpeed (rpm) Imposta la velocità del motore in giri al minuto (RPM).

step(steps) Attiva il motore per un numero specifico di **steps**, ad una velocità determinata dalla chiamata più recente per **SetSpeed ()**.



```
Test_stepper-L298 | Arduino 1.7.7
File Modifica Sketch Strumenti Aiuto
Test_stepper-L298
#include <Stepper.h>

const int stepsPerRevolution = 200;
//cambiare questo per adattare il numero di passi per giro
//per il vostro motore da calcolare dividendo 360° con il
//numero di gradi per passo

// Inizializzare la libreria passo-passo su pin da 2 a 5:
Stepper myStepper(stepsPerRevolution, 2,3,4,5);

Compilazione completata

Le variabili globali usano 41 byte (2%) di memoria dinamica, lasciando altri
2.007 byte liberi per le variabili locali. Il massimo è 2.048 byte.

15 Arduino Uno on COM3
```

```
/*
Test_stepper-L298.ino
Programma per testare il comando di un motore stepper tramite
un L298N Dual H-Bridge Motor Controller

Vengono utilizzati i seguenti pin
Pin +5V          -> Pin laterale del potenziometro
Pin GND          -> Pin GND modulo e pin laterale potenz.
Pin Digital 2    -> Collegato al pin IN1
Pin Digital 3    -> Collegato al pin IN2
Pin Digital 4    -> Collegato al pin IN3
Pin Digital 5    -> Collegato al pin IN4
Pin Analog A0    -> Collagato al centrale del potenziometro

#include <Stepper.h>

const int stepsPerRevolution = 200;
//cambiare questo per adattare il numero di passi per giro
//per il vostro motore da calcolare dividendo 360° con il
//numero di gradi per passo

// Inizializzare la libreria passo-passo su pin da 2 a 5:
Stepper myStepper(stepsPerRevolution, 2,3,4,5);

int stepCount = 0;          // numero di passi del motore

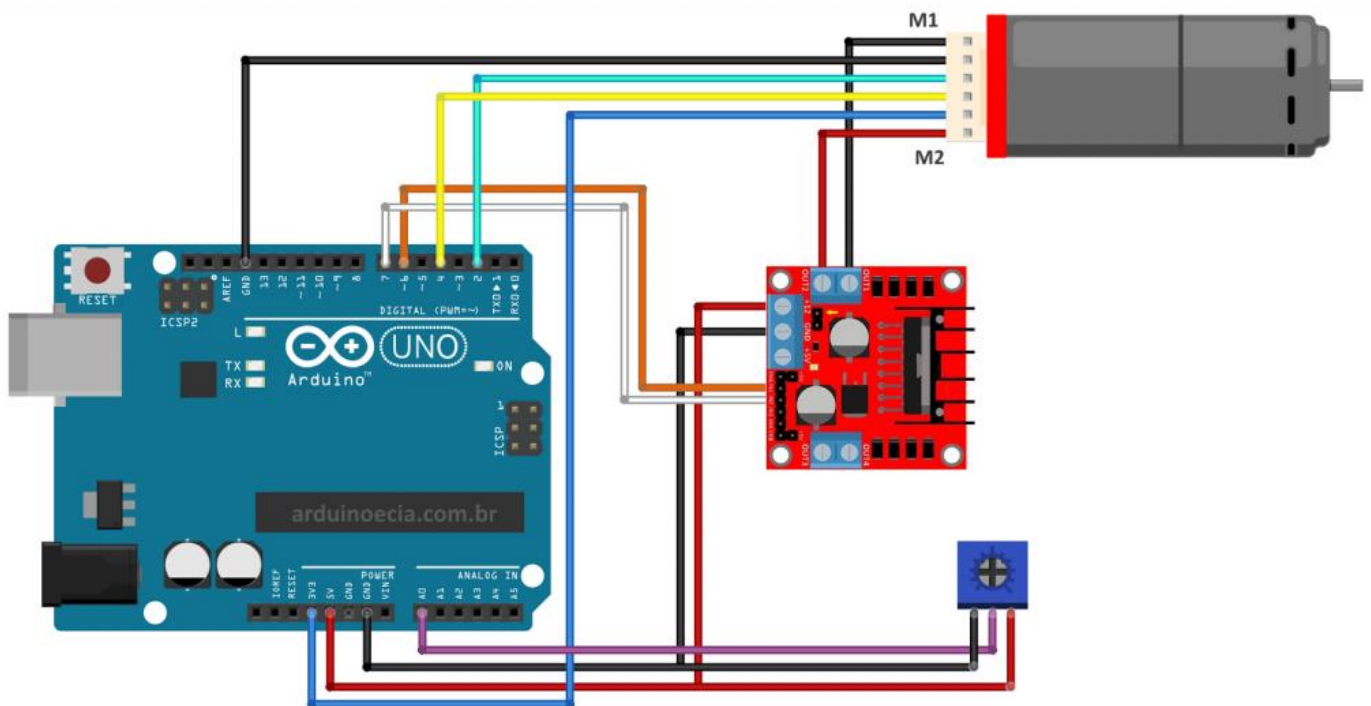
void setup() {
  // niente da fare all'interno della configurazione
}
```

```

void loop() {
  // Legge il valore della tensione fornito dal potenziometro:
  int sensorReading = analogRead(A0);
  // mappa il valore nel range da 0 a 100:
  int motorSpeed = map(sensorReading, 0, 1023, 0, 100);
  // Imposta la velocità del motore:
  if (motorSpeed > 0) {
    myStepper.setSpeed(motorSpeed);
    // step 1/100 of a revolution:
    myStepper.step(stepsPerRevolution/100);
  }
}

```

Motori con encoder



L9110S Dual-Channel Driver Module Features

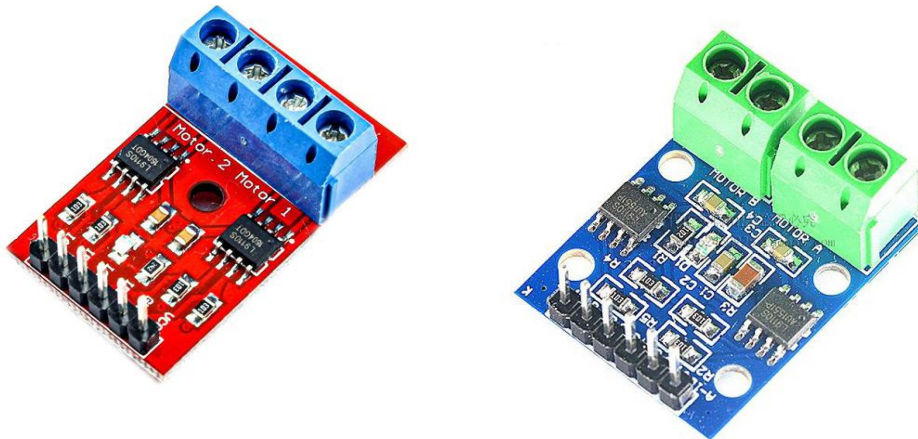
Motor driver modules are very common nowadays and widely used to control the speed and direction of motors. The L9110S dual-channel module is one of them. This module can control two DC motors and one stepper motor. It is based on L9110 IC. The key features are:

- The allowable continuous current for each channel: 800 mA
- The maximum allowable current: 1.5 A
- Power supply: 2.5V to 12V

Note

The two connected pins to each DC motor can be PWM or digital. If defined as digital, it can only control the direction of motors motion.

You can see two similar modules based on L9110 IC in the pictures below.



L9110S Dual-Channel Driver Module Pinout

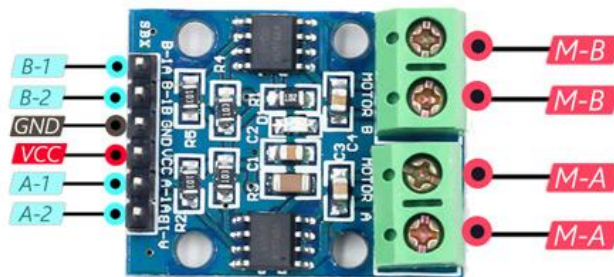
This Module has following pins:

- **VCC:** Module power supply
- **GND:** Ground
- **M-A:** Motor A pin
- **M-B:** Motor B pin
- **A-1:** Control signal for motor A
- **A-2:** Control signal for motor A
- **B-1:** Control signal for motor B
- **B-2:** Control signal for motor B

You can see the pinout of these modules in the image below.



www.Electropeak.com



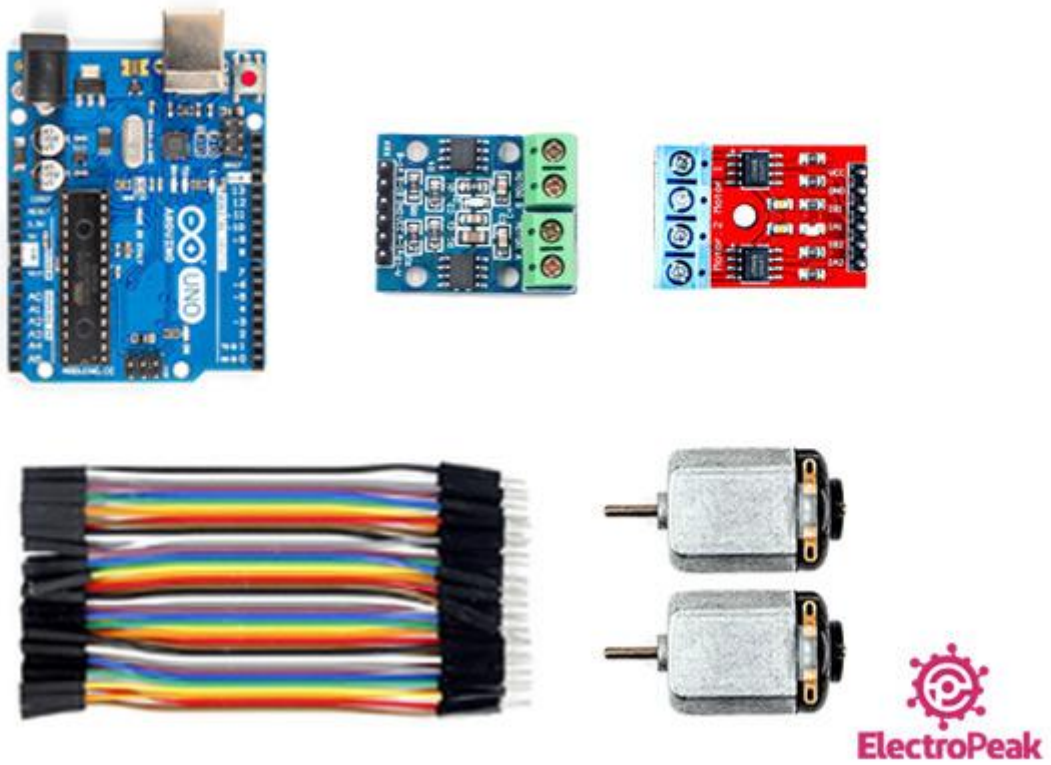
www.Electropeak.com



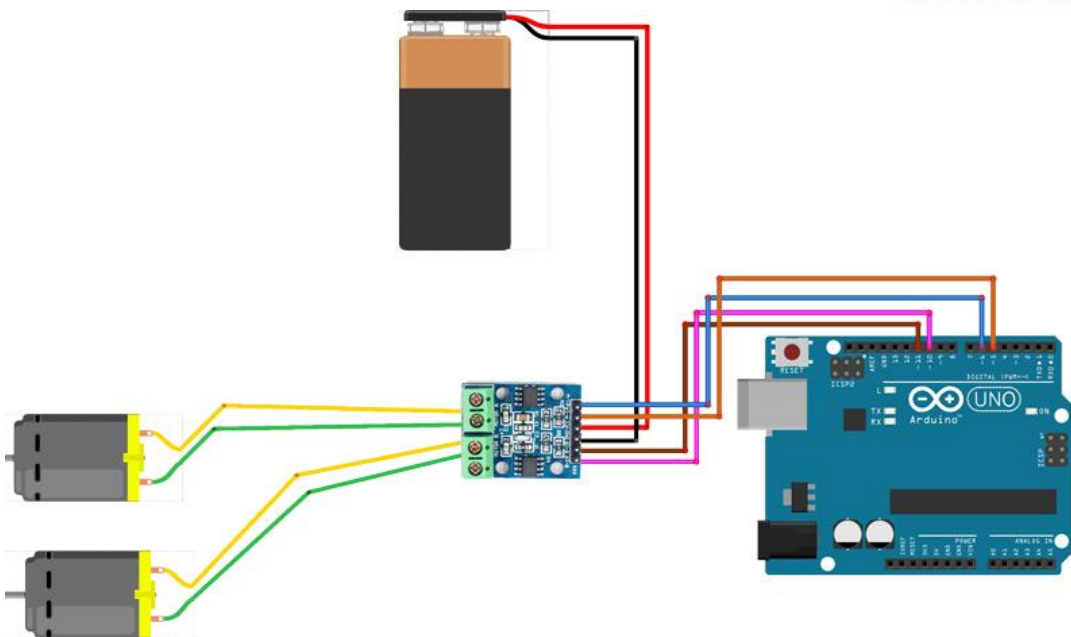
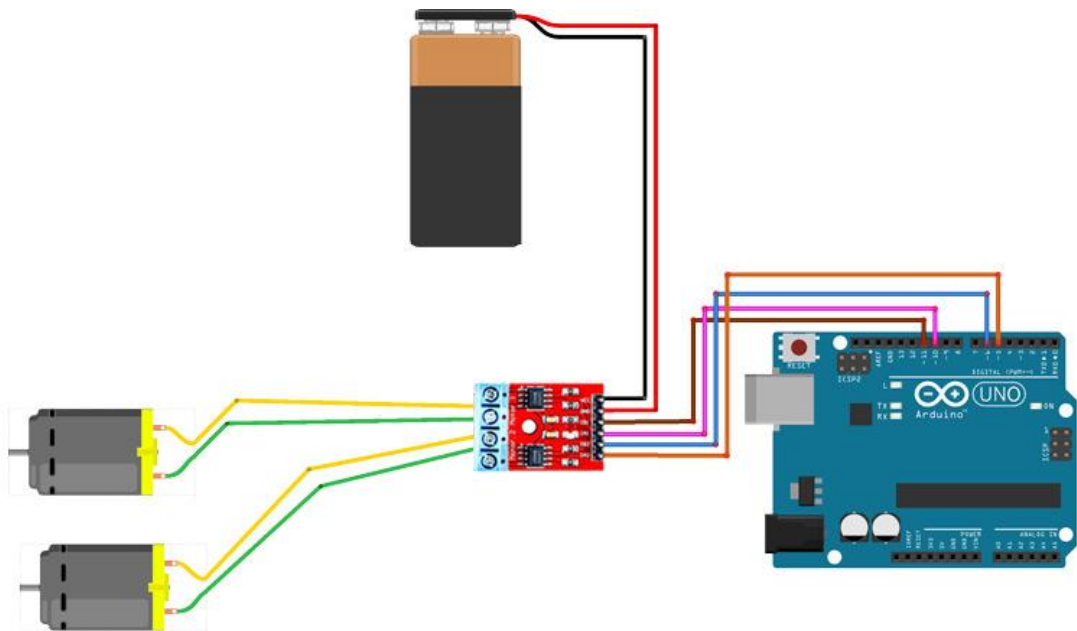
Note that with different modes, motor performance varies. See different modes below.

	AIN1	AIN2
Forward Direction	HIGH	LOW
Reverse Direction	LOW	HIGH
Brake / Stopped	LOW	LOW
Brake / Stopped	HIGH	HIGH

Required Materials



The 2 following images show how you should connect Arduino to these modules. Connect wires accordingly.



Note

Connect Arduino GND pin to power supply GND pin.

Warning

Be careful not to use power supply greater than 6V, because the motors voltage are 6V.

Step 2: Code

Upload the following code to your Arduino.

```
/*
  L9110S-Stepper-DC-motor-Driver-Module
  made on 28 oct 2020
  by Amir Mohammad Shojaee @ Electropeak
  Home
*/

#define A1 5 // Motor A pins
#define A2 6
#define B1 10 // Motor B pins
#define B2 11

int incomingByte = 0; // for incoming serial data

void setup() {

  pinMode(A1, OUTPUT);
  pinMode(A2, OUTPUT);
  pinMode(B1, OUTPUT);
  pinMode(B2, OUTPUT);

  digitalWrite(A1, LOW);
  digitalWrite(A2, LOW);
  digitalWrite(B1, LOW);
  digitalWrite(B2, LOW);

  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps

  Serial.println("select direction of movement");
  Serial.println("1.forward");
  Serial.println("2.backward");
  Serial.println("3.stop");

}

int input = 0;
void loop() {

  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();
    input = incomingByte - 48; //convert ASCII code of numbers to 1,2,3

    switch (input) {
```

```

case 1:      // if input=1 ..... motors turn forward
  forward();
  break;
case 2:      // if input=2 ..... motors turn backward
  backward();
  break;
case 3:      // if input=1 ..... motors turn stop
  Stop();
  break;
}
delay(200);
input=0;
}
}
void forward() {          //function of forward
  analogWrite(A1, 255);
  analogWrite(A2, 0);
  analogWrite(B1, 255);
  analogWrite(B2, 0);
}

void backward() {        //function of backward
  analogWrite(A1, 0);
  analogWrite(A2, 210);
  analogWrite(B1, 0);
  analogWrite(B2, 210);
}

void Stop() {           //function of stop
  digitalWrite(A1, LOW);
  digitalWrite(A2, LOW);
  digitalWrite(B1, LOW);
  digitalWrite(B2, LOW);
}
}

```

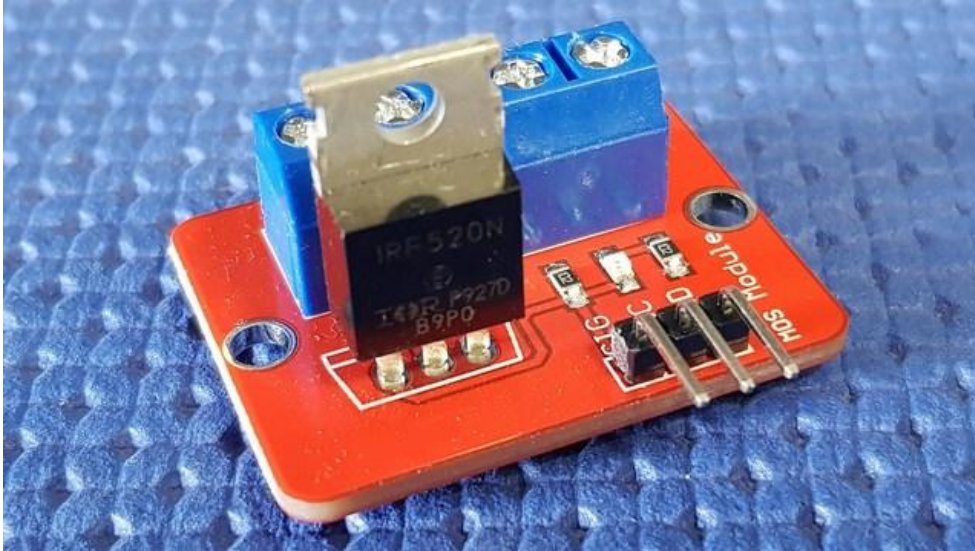
Arduino

Using this code, the motors direction can be controlled through the Serial Monitor. This means if you enter key number 1, the motor will move in the straight direction at maximum speed. If key number 2 is entered, the motor moves in the reverse direction at a speed less than the maximum. Finally, by pressing the number 3 key, the motor will stop.

Modulo IRF520 MOSFET Switch

Questo piccolo modulo è una scheda breakout su cui è presente un **MOSFET IRF520** che funziona come uno Switch cioè interruttore, con esso è possibile sostituire per esempio l'utilizzo di un relè.

Il modulo è progettato per commutare carichi alimentati in corrente continua. La gestione è possibile tramite un singolo pin digitale del microcontrollore. Altri possibili esempi di utilizzo sono pilotare un motore CC per applicazioni robotiche, o controllare altri carichi, sempre in corrente continua.



Descrizione del modulo IRF520 MOSFET

Il **modulo IRF520 MOSFET** è fornito di terminali a vite per interfacciarsi al carico e alla fonte di alimentazione esterna. Un indicatore LED fornisce un'indicazione visiva di quando il carico viene cambiato.

Caratteristiche:

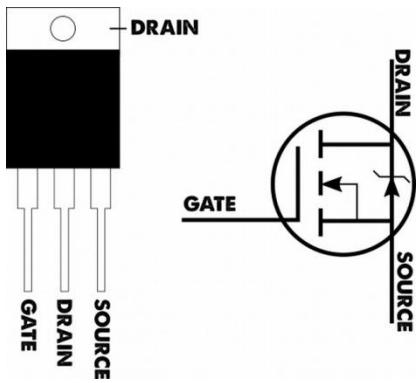
- Dimensioni modulo: 33,4 * 25,6 mm
- Tensione di pilotaggio: 3,3-5V
- Controllo: digitale.
- Tensione di carico in uscita: da 0 a 24V
- Corrente di carico in uscita: 2A con pilotaggio di 5V.

Circuito elettrico

Il componente principale è il **MOSFET IRF520** ([vedere datasheet](#)), la parola **MOSFET** è l'acronimo del termine inglese **Metal-Oxide-Semiconductor Field-Effect Transistor**, ovvero semiconduttore metallo-ossido- transistor ad effetto di campo.

Si tratta di semiconduttori molto utilizzati nelle applicazioni comuni di commutazione azionate da tensione, con alta tensione e corrente elevata.

La caratteristica dei MOSFET è che la corrente che transita dal **DRAIN (D)** I_d è pilotata dalla differenza di tensione tra **GATE (G)** e **SOURCE (D)** V_{gs} .



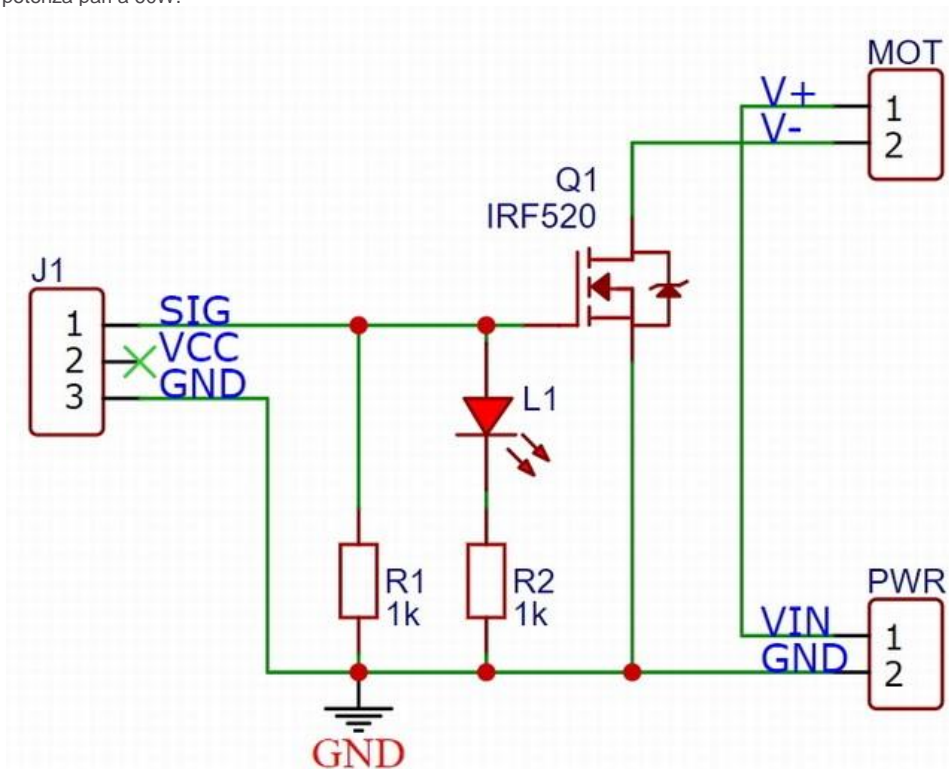
Dal datasheet ricaviamo che il componente è in grado di gestire un'alimentazione (V_{ds}) anche di 100V.

Può arrivare ad impulsi di corrente I_{dm} di 37A o corrente continua a circa 9A.

Il massimo passaggio di corrente (9.2 Ampere a 25°C) potrà avvenire quando applicherò sul GATE una tensione di circa 10V.

Nel caso di pilotaggio del gate direttamente con un'uscita digitale di Arduino (5 Volt) il **MOSFET** ammetterà un passaggio di corrente tra drain e source di massimo 2 Ampere. Nel caso di 1A è meglio aggiungere un dissipatore di calore

La grossa placca metallica opposta ai 3 pin è direttamente connessa al SOURCE ed è utilizzata per dissipare il calore permettendo la dissipazione di una potenza pari a 60W.



Nel circuito oltre al MOSFET IRF520 Q1 con funzione di switch, è poi presente la resistenza di Pull Down R1 da 1k Ω , questa assicura che il gate sia a zero quando non viene pilotato.

La resistenza R2 ha la funzione di limitatrice per la corrente di alimentazione del Led L1 che segnala l'attivazione del MOSFET.

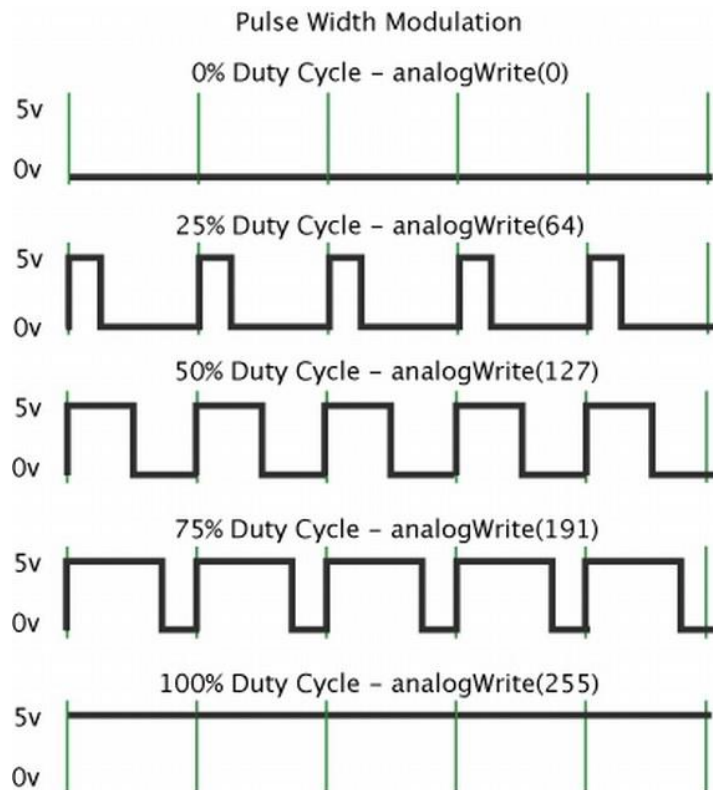
Utilizzo della tecnica PWM

Il **modulo IRF520 MOSFET** può controllare dei motori DC mediante la [tecnica PWM](#) (Pulse Width Modulation).

Con questo sistema si converte una tensione di ingresso costante in una tensione variabile controllando il periodo di tempo in cui l'impulso è ALTO (Duty Cycle – ciclo di lavoro) .

Come visibile nel grafico sotto riportato, con un valore di Duty Cycle pari a 0 % al motore non arriva tensione , mentre con un valore di Duty Cycle pari a 100% al motore sarà fornita la massima tensione.

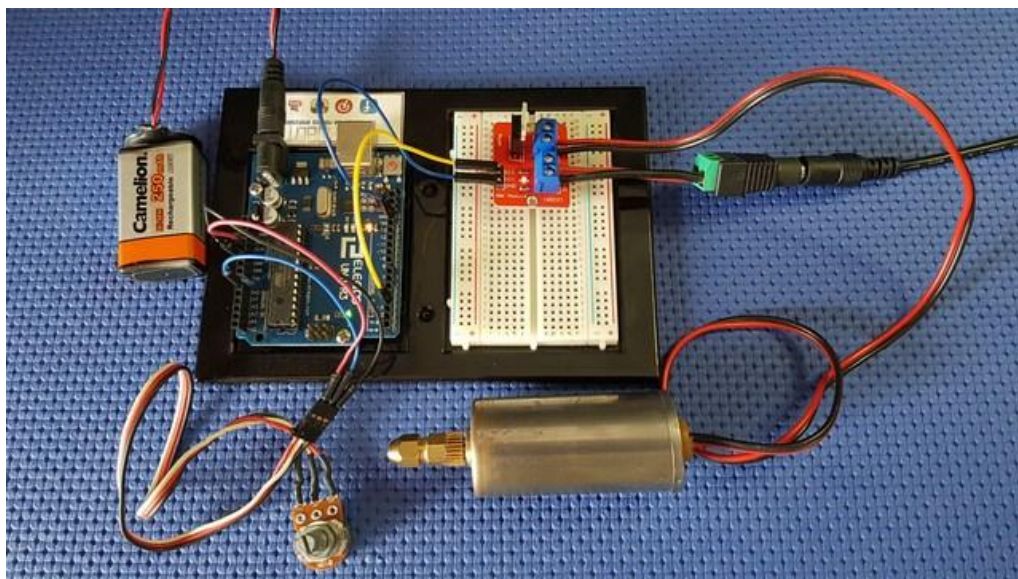
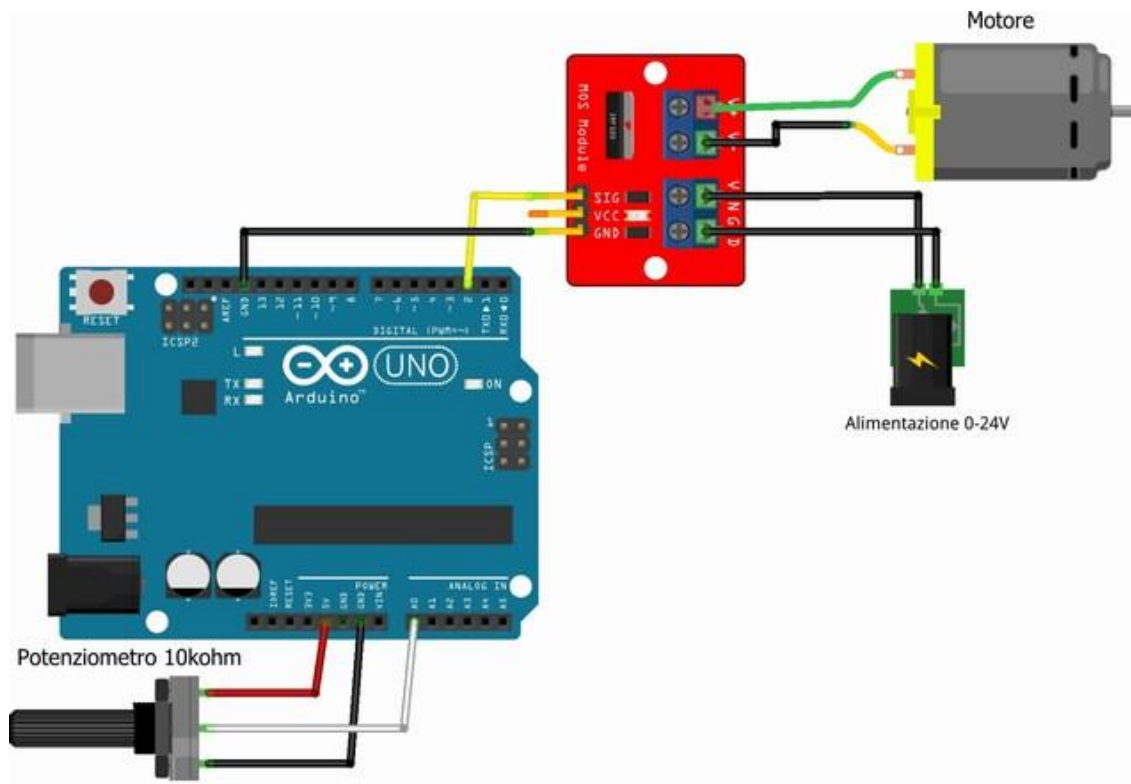
Nei valori intermedi al motore al motore sarà fornito una tensione sempre più elevata.



Circuito per il test

Per il test occorrono:

- Una scheda Arduino UNO
- Modulo IRF3205
- Potenzziometro 10k Ω
- Cavetti di collegamento
- Motore elettrico DC, in questo caso è stato utilizzato un elettro mandrino 9/12 volt 10 watt 800/10000 giri'
- Una fonte di alimentazione per Arduino e una per il motore



Sketch di prova

In questo codice, ruotando il potenziometro, il valore della tensione che giungerà al pin analogico A0, varierà tra 0 e 5V, questo valore sarà trasformato in valore digitale da 0 a 1023.

Con la [funzione map](#), il valore letto sarà convertito poi in un range compreso tra 0 e 255, questo valore sarà utilizzato dal [comando analogWrite](#) per fornire il valore PWM.

Si consideri che sino ad un certo valore di tensione minima il motore non ruoterà, mentre ad un certo punto inizierà a ruotare sempre più velocemente a seconda della rotazione del potenziometro.

La tensione di alimentazione non dovrà essere comunque superiore alla tensione massima ammissibile del motore pena il suo possibile danneggiamento.

```
1 #define PWM 3 //Definisce il pin di uscita tra 3-5-6-9-10-11
```

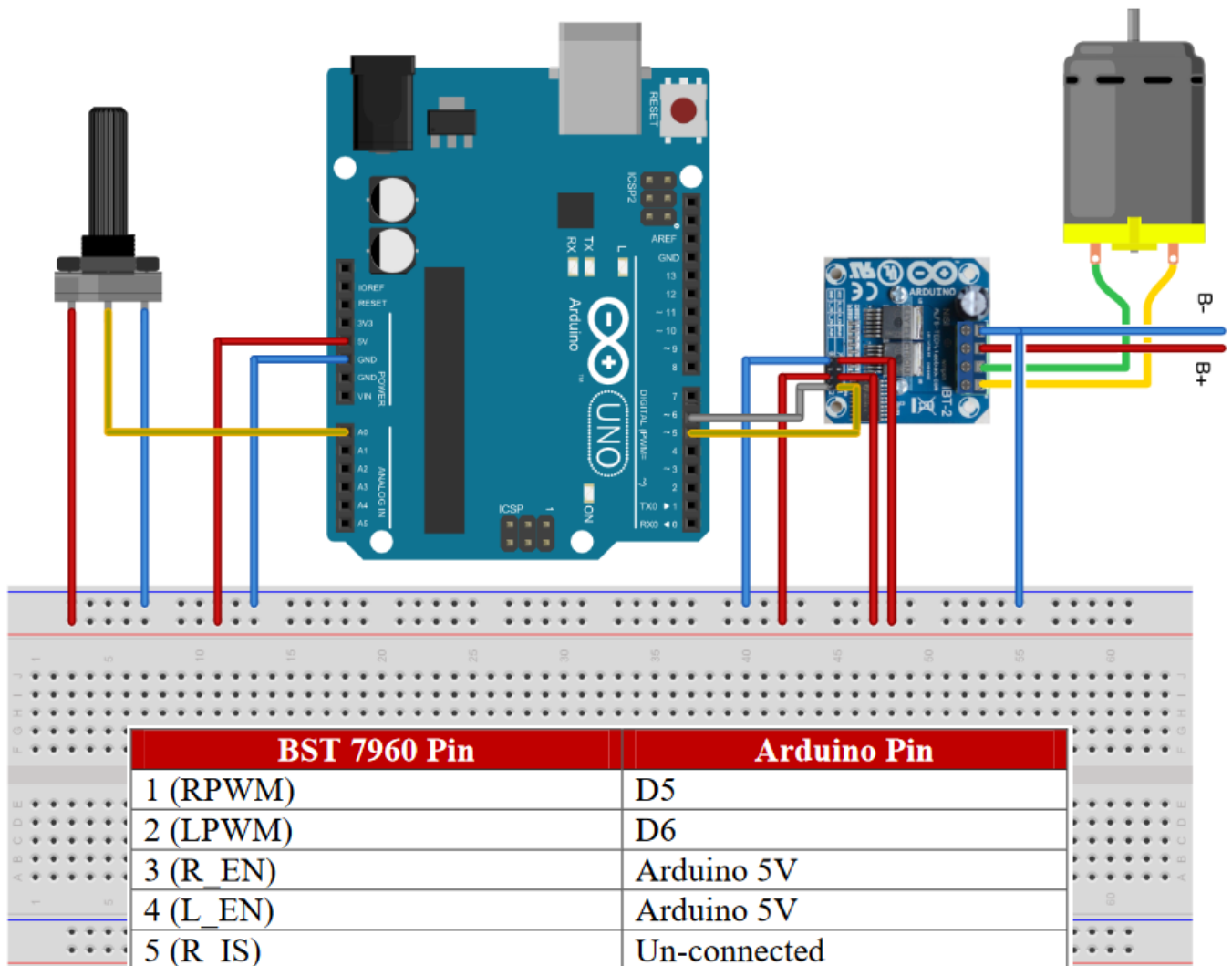
```
2 int pot;
3 int out;
4
5 void setup() {
6   pinMode(PWM,OUTPUT); //Imposta pin PWM come uscita
7 }
8
9 void loop() {
10  pot=analogRead(A0); //Lettura potenziometro
11  out=map(pot,0,1023,0,255);
12  analogWrite(PWM,out); //Uscita valore PWM
13 }
```

BTS7960 43A DUAL H-BRIDGE HIGH-POWER MOTOR DRIVER

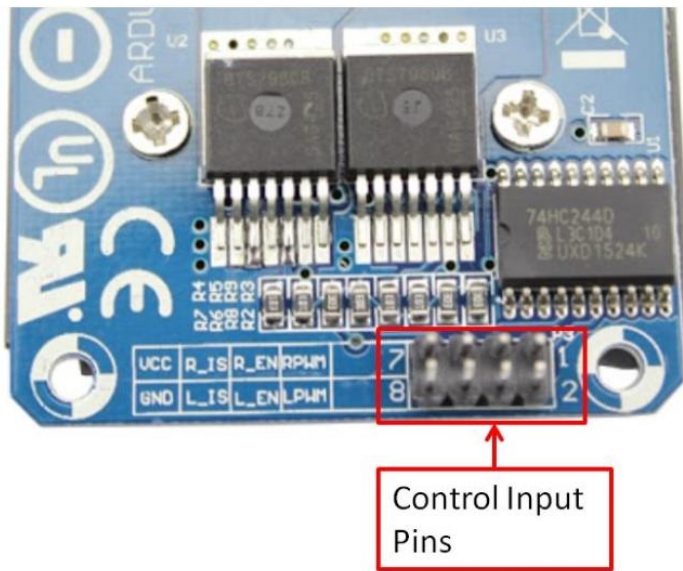
The BTS7960 is a fully integrated high current H bridge module for motor drive applications. Interfacing to a microcontroller is made easy by the integrated driver IC which features logic level inputs, diagnosis with current sense, slew rate adjustment, dead time generation and protection against over temperature, overvoltage, undervoltage, overcurrent and short circuit. The BTS7960 provides a cost optimized solution for protected high current PWM motor drives with very low board space consumption.

Specifications:

- Input Voltage: 6 ~ 27Vdc.
- Driver: Dual BTS7960H Half-Bridge Configuration.
- Peak current: 43-Amp.
- PWM capability of up to 25 kHz.
- Control Input Level: 3.3~5V.
- Control Mode: PWM or level
- Working Duty Cycle: 0~100%.
- Over-voltage Lock Out.Under-voltage Shut Down.
- Board Size(LxWxH): 50mmx50mmx 43mm.
- Weight: ~66g.

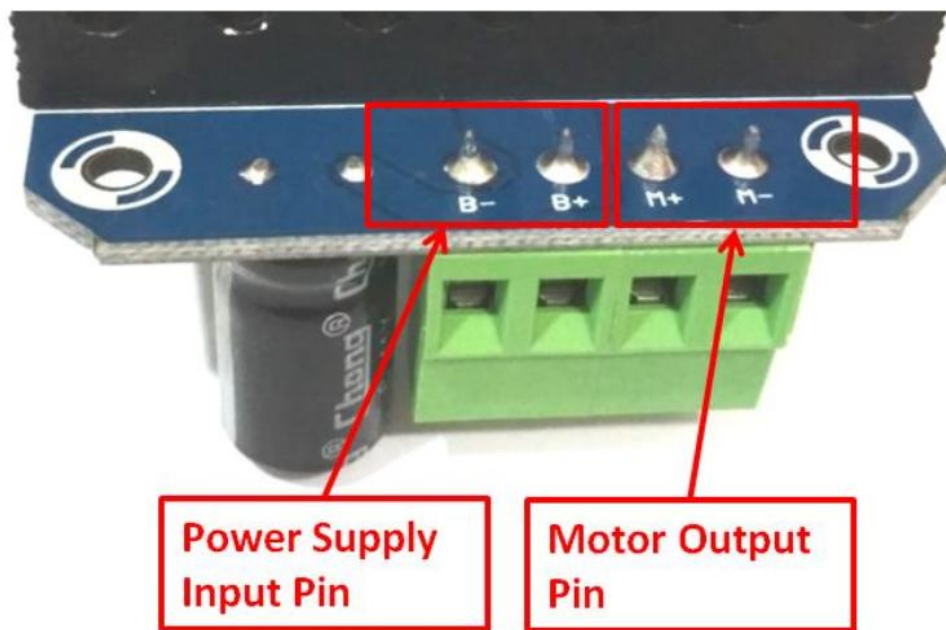


BST 7960 Pin	Arduino Pin
1 (RPWM)	D5
2 (LPWM)	D6
3 (R_EN)	Arduino 5V
4 (L_EN)	Arduino 5V
5 (R_IS)	Un-connected
6 (L_IS)	Un-connected
7 (VCC)	Arduino 5V
8 (GND)	Arduino GND



Control Input Pins

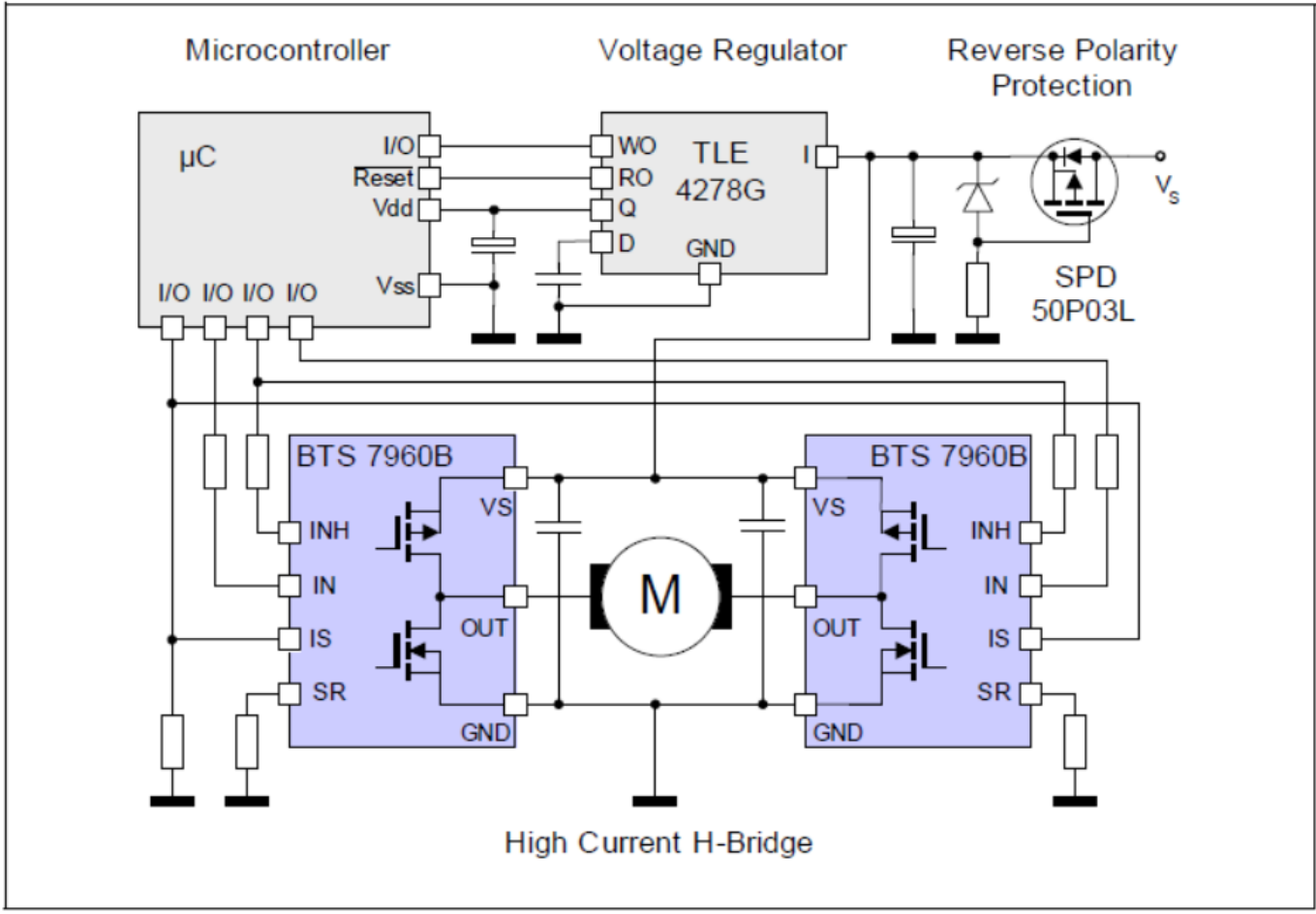
Pin No	Function	Description
1	RPWM	Forward Level or PWM signal, Active High
2	LPWM	Reverse Level or PWM signal, Active High
3	R_EN	Forward Drive Enable Input, Active High/ Low Disable
4	L_EN	Reverse Drive Enable Input, Active High/Low Disable
5	R_IS	Forward Drive, Side current alarm output
6	L_IS	Reverse Drive, Side current alarm output
7	Vcc	+5V Power Supply microcontroller
8	Gnd	Ground Power Supply microcontroller



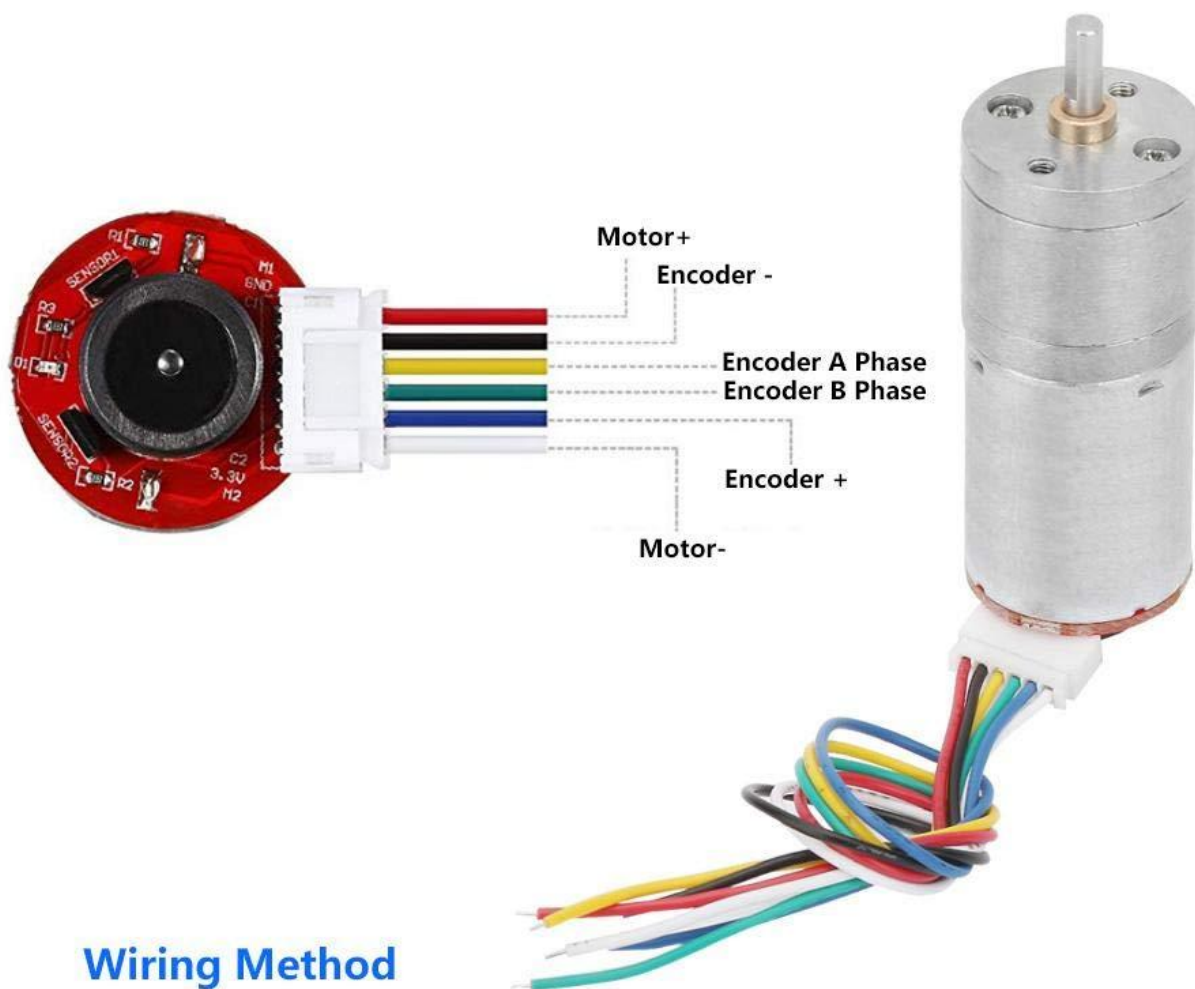
Power Supply Input Pin

Motor Output Pin

Pin No	Function	Description
1	B+	Positive Motor Power Supply. 6 ~ 27VDC
2	B-	Negative Motor Power Supply. Ground
3	M+	Motor Output +
4	M-	Motor Output -

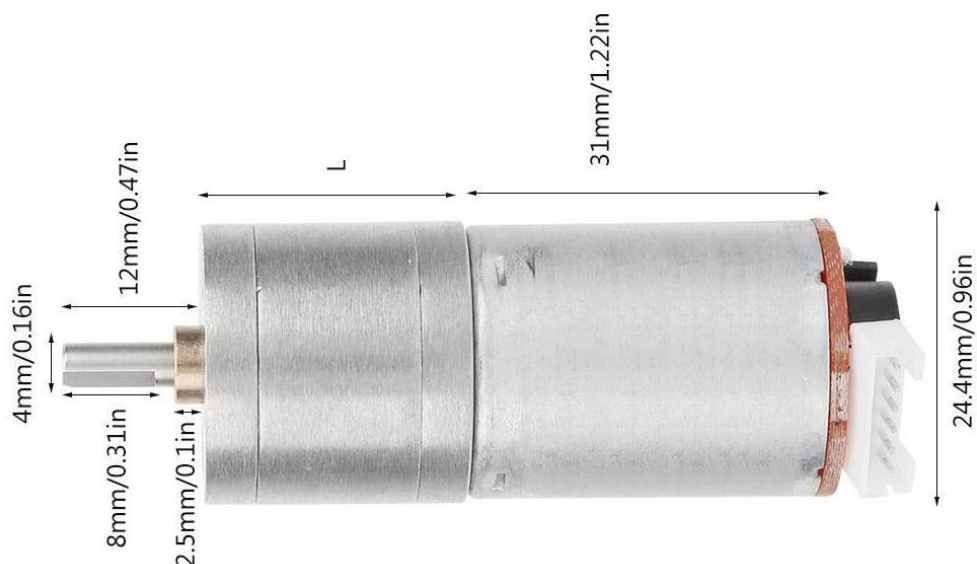


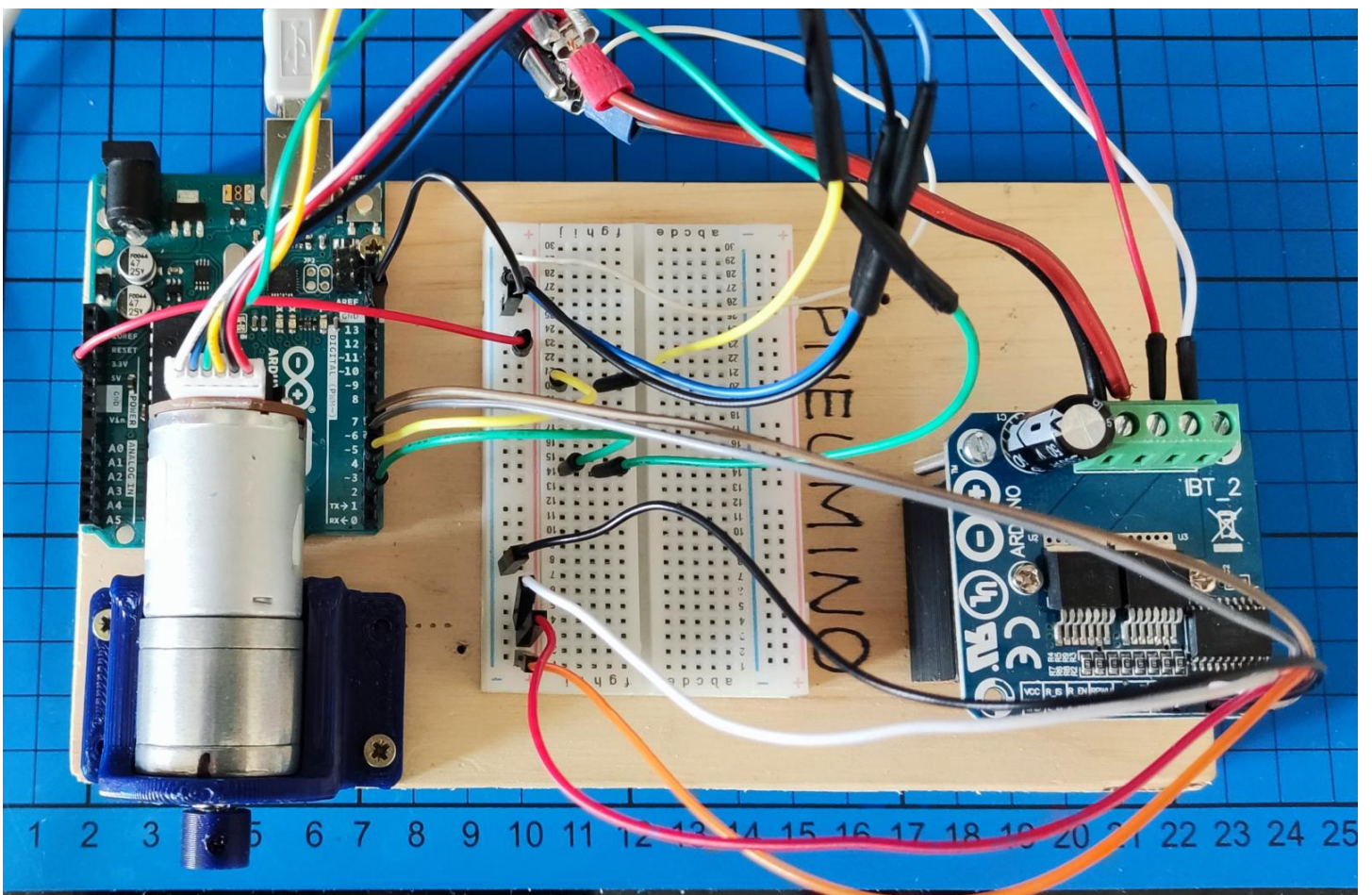
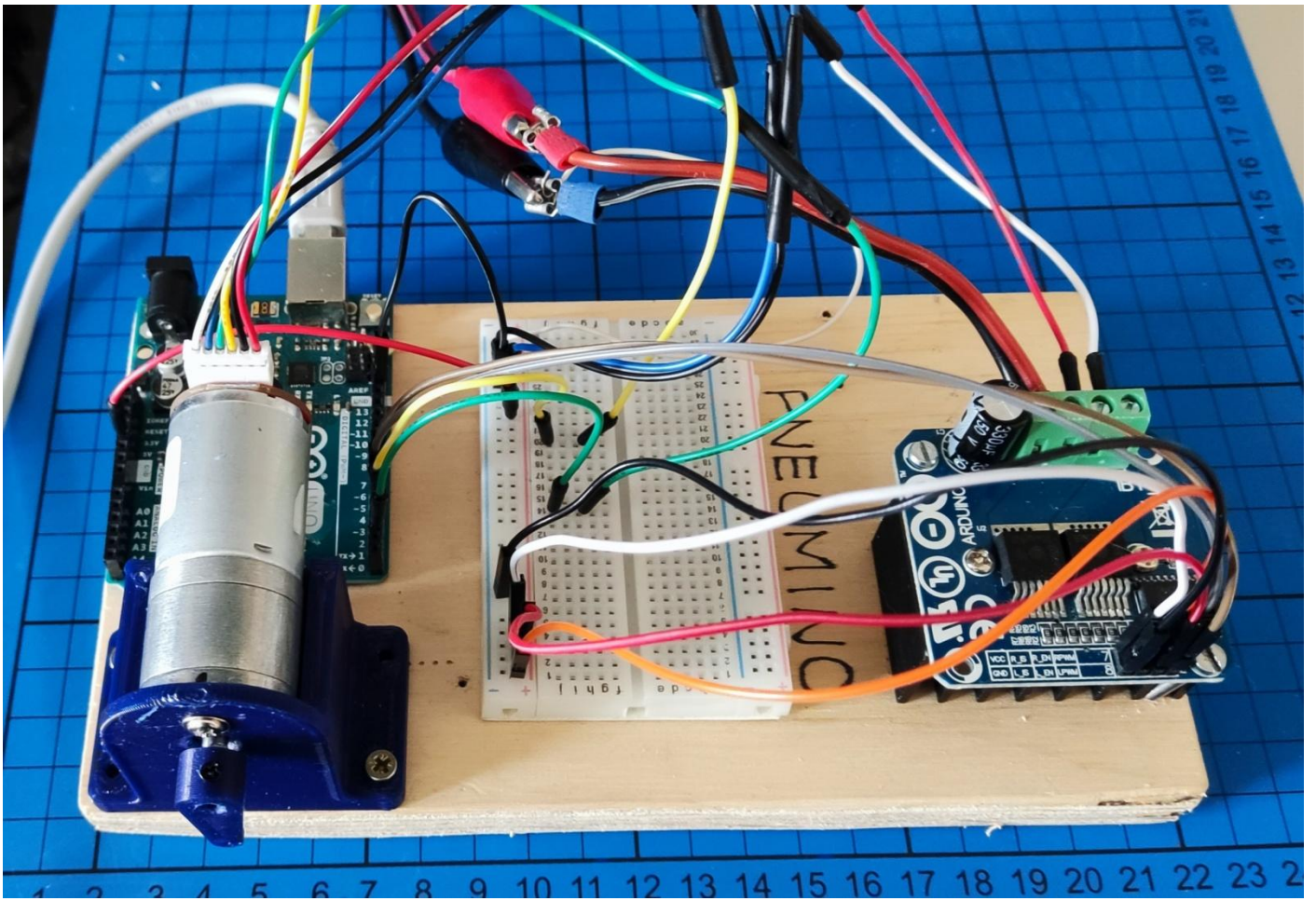
DC motor 12 V 130 o 200 RPM



Wiring Method

- Red - Motor+ (Reverse to control Forward/Reverse of the motor)
- Black - Encoder - (3.3-5V, do not connect positive pole and negative pole wrong)
- Yellow - Encoder A Phase (11 signals when the motor rotate one circle)
- Green - Encoder B Phase (11 signals when the motor rotate one circle)
- Blue - Encoder+ (3.3-5V, do not connect the positive and negative wrong)
- White - Motor- (Reverse to control Forward/Reverse of the motor)





```

#include <util/atomic.h>

// Pins for BTD7960 Motor Driver
#define ENCA 2 // decoder A
#define ENCB 4 // decoder B
#define IN1 5 // PWM 1
#define IN2 6 // PWM 2

// Counters for milliseconds during interval
long previousMillis = 0;
long currentMillis = 0;

// globals time var
int pos = 0;
long prevT = 0;
int posPrev = 0;

// Use the "volatile" directive for variables used in an interrupt
volatile int pos_i = 0;
volatile float velocity_i = 0;
volatile long prevT_i = 0;

// Filtered velocity
float v1Filt = 0;
float v1Prev = 0;

//SERIAL INPUT SETUPS
String inputString = ""; // a string to hold incoming data
String Pin;
int iPin;
String State;
boolean stringComplete = false; // whether the string is complete
long startTime ; // start time for stop watch
long elapsedTime ;

//PID variables
double set_speed = 50; // setpoint to 30 rpm
double v_speed = 0; // actual speed
double e_speed = 0; //error of speed = set_speed - v_speed
double e_speed_pre = 0; //last error of speed
double e_speed_sum = 0; //sum error of speed
double pwm_pulse = 0; //this value is 0~255

double kp = 5;
double ki = 20;
double kd = 0.1;

void setup() {
  Serial.begin(9600);

  // Setup BTD7960 Motor Driver
  pinMode(ENCA,INPUT);
  pinMode(ENCB,INPUT);

```

```

pinMode(IN1,OUTPUT);
pinMode(IN2,OUTPUT);
attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);
}

void loop() {
  // read the position in an atomic block to avoid potential misreads
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE){ pos = pos_i; }

  // Compute velocity DC motor
  long currT = micros();
  float deltaT = ((float) (currT-prevT))/1.0e6;
  float velocity1 = abs((pos - posPrev)/deltaT);
  posPrev = pos;
  prevT = currT;

  // Convert count/s to RPM
  float v1 = velocity1/600.0*60.0;
  // Low-pass filter (25 Hz cutoff)
  v1Filt = 0.854*v1Filt + 0.0728*v1 + 0.0728*v1Prev;
  v1Prev = v1;
  v_speed = v1Filt; // actual speed

  //PID code
  e_speed = set_speed - v_speed; // error speed
  // calculate voltage power for DC motor with P.I.D.
  //   proportional   integral   derivative
  pwm_pulse = kp * e_speed + ki * e_speed_sum + kd * (e_speed - e_speed_pre)/ deltaT;
  e_speed_sum += (e_speed * deltaT); //sum of error --> integral
  e_speed_pre = e_speed; //save last (previous) error

  // set limit to sum of error (integral)
  if (e_speed_sum >100) {e_speed_sum = 100; }
  else if (e_speed_sum <-100) {e_speed_sum = -100; }

  // set PWM limits
  if(pwm_pulse > 255) { pwm_pulse = 255; }
  else if(pwm_pulse < 0) { pwm_pulse = 0; }

  // set DC motor speed
  setMotor(pwm_pulse,IN1,IN2);

  // print data
  Serial.print(set_speed); Serial.print(" "); Serial.print(v1Filt); Serial.print(" "); Serial.print(pwm_pulse); Serial.println();

  // check for new setup rpm non serial -> 1=rpm
  CheckSerial();

  delay(10);
}

// SerialEvent occurs whenever a new data comes in the hardware serial RX.
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:

```

```

inputString += inChar;
// if the incoming character is a newline, set a flag
// so the main loop can do something about it:
if (inChar == '\n') {
    stringComplete = true;
}
}
}

```

```

void CheckSerial(){
// if Newline arrived on SERIAL
if (stringComplete) {
//Serial.println(inputString);

int id = inputString.indexOf("=");
if (id>0) {
    Pin = inputString.substring(0, id) ;
    State= inputString.substring(id+1, inputString.length() - id+1);
    iPin= State.toInt();

// rotation
if (iPin>=0 && iPin < 255) {
    if (Pin== "1") {
        //Serial.println("DC" + Pin + "=" + State);
        //analogWrite(IN1, iPin);
        //analogWrite(IN2, 0);
        set_speed = iPin;
    }
    else if (Pin== "2") {
        //Serial.println("DC" + Pin + "=" + State);
        //analogWrite(IN1, iPin);
        //analogWrite(IN2, 0);
    }
}
else {
    //Serial.println("error " + inputString);
    // STOP DC motor
    analogWrite(IN1, 0);
    analogWrite(IN2, 0);
}
}
// clear the input string:
inputString = "";
stringComplete = false;
}
}

```

```

void setMotor(int pwmVal, int in1, int in2){
    analogWrite(in1,pwmVal);
    analogWrite(in2,LOW);
}

```

```

void readEncoder(){
// Read encoder B when ENCA rises
int b = digitalRead(ENCB);
int increment = 0;

```

```
if(b>0){
  // If B is high, increment forward
  increment = 1;
}
else{
  // Otherwise, increment backward
  increment = -1;
}
pos_i = pos_i + increment;

// Compute velocity with method 2
long currT = micros();
float deltaT = ((float) (currT - prevT_i))/1.0e6;
velocity_i = abs(increment/deltaT);
prevT_i = currT;
}
```