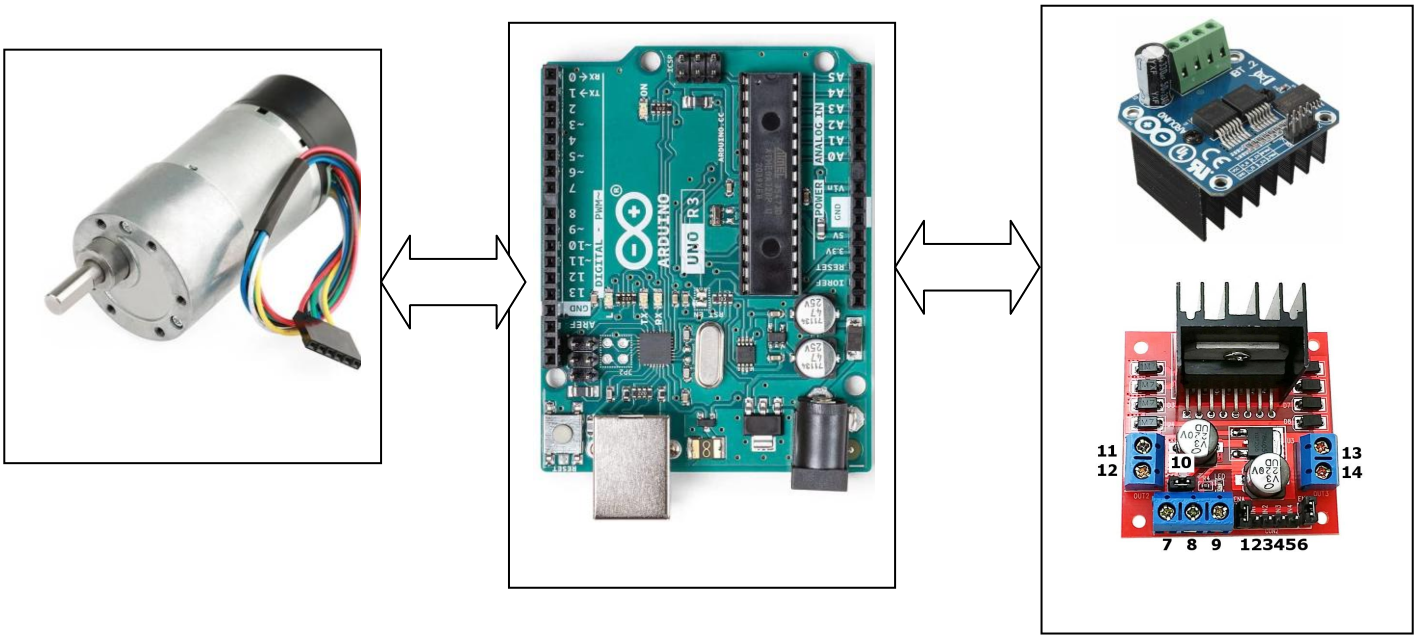
Target

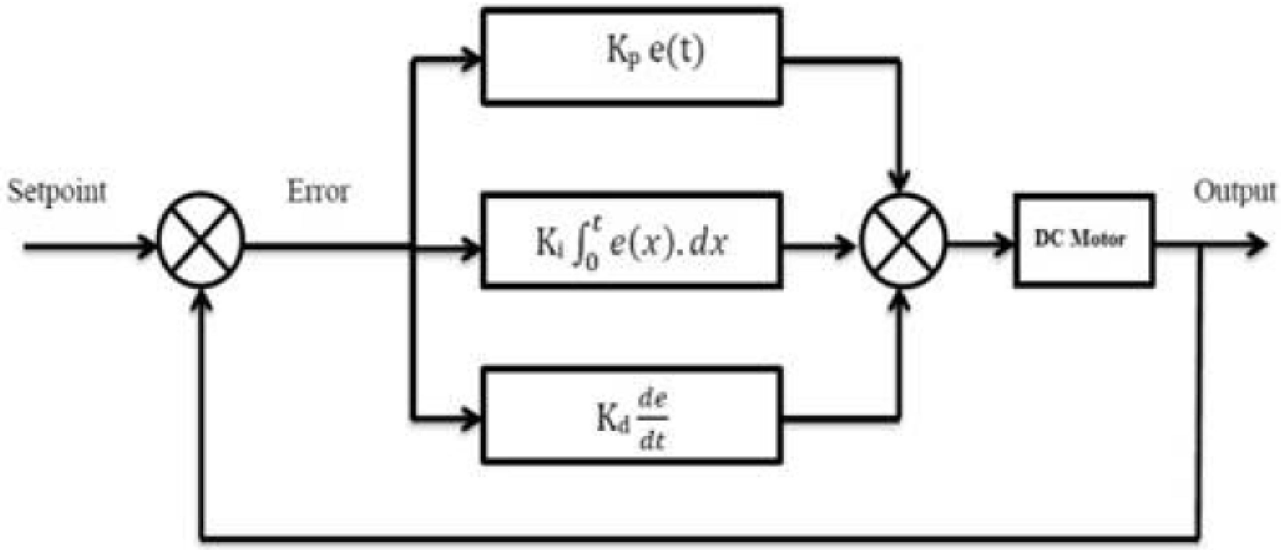
Keep the motor speed constant.

The speed can vary for several reasons, for example:

- reduction of the motor supply voltage (battery)
- increase of the resistant torque to the shaft

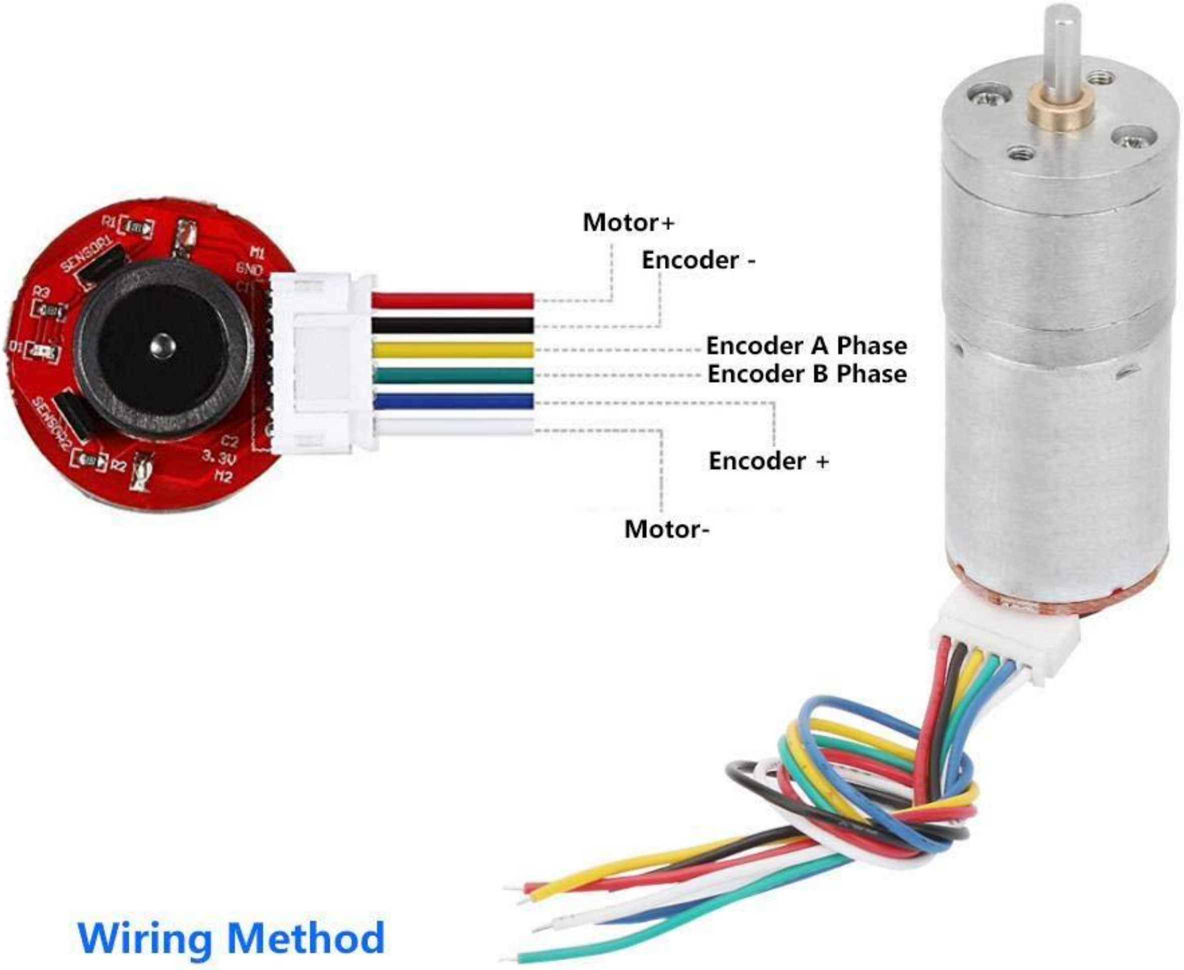


PID Controller Design



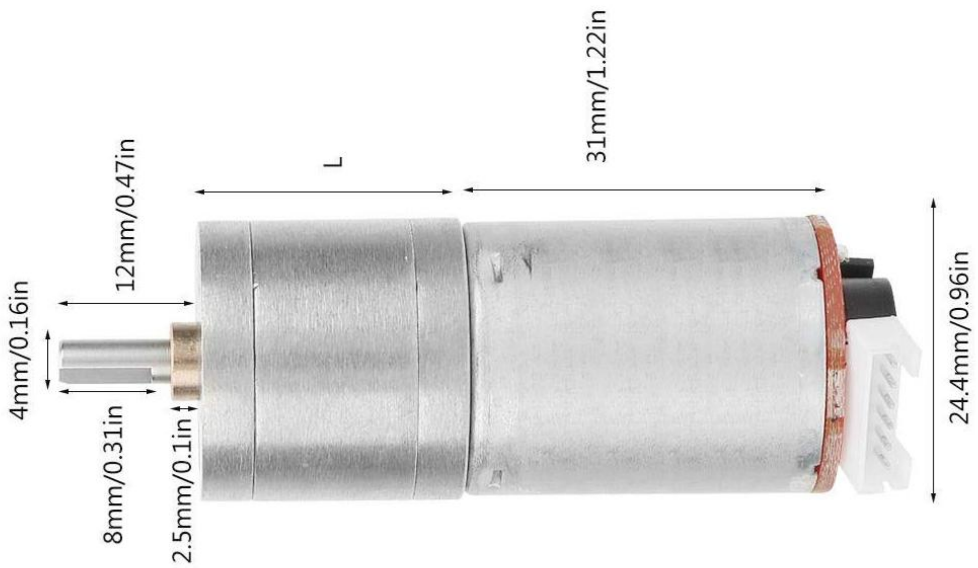
```
//speed error
e_speed = set_speed - v_speed;   // error speeed
// calculate voltage power for DC motor with P.I.D.
//          proportional       integral          derivative
pwm_pulse = kp * e_speed    +  ki * e_speed_sum  + kd * (e_speed - e_speed_pre)/ deltaT;
// integral error
e_speed_sum +=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

DC motor 12 V 130 o 200 RPM

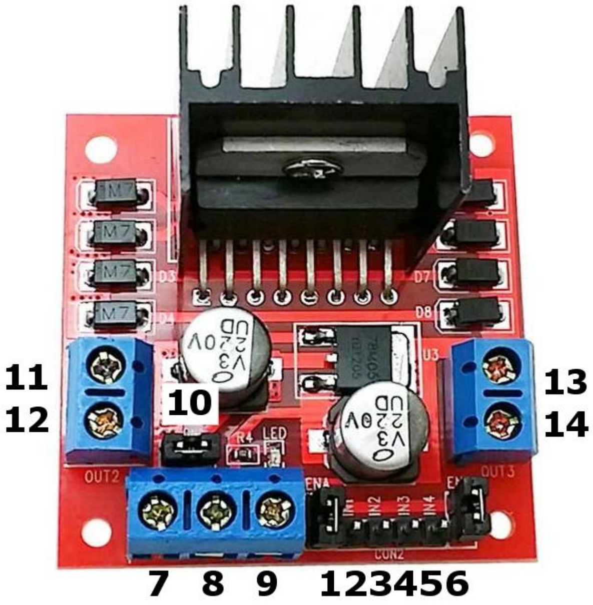


Wiring Method

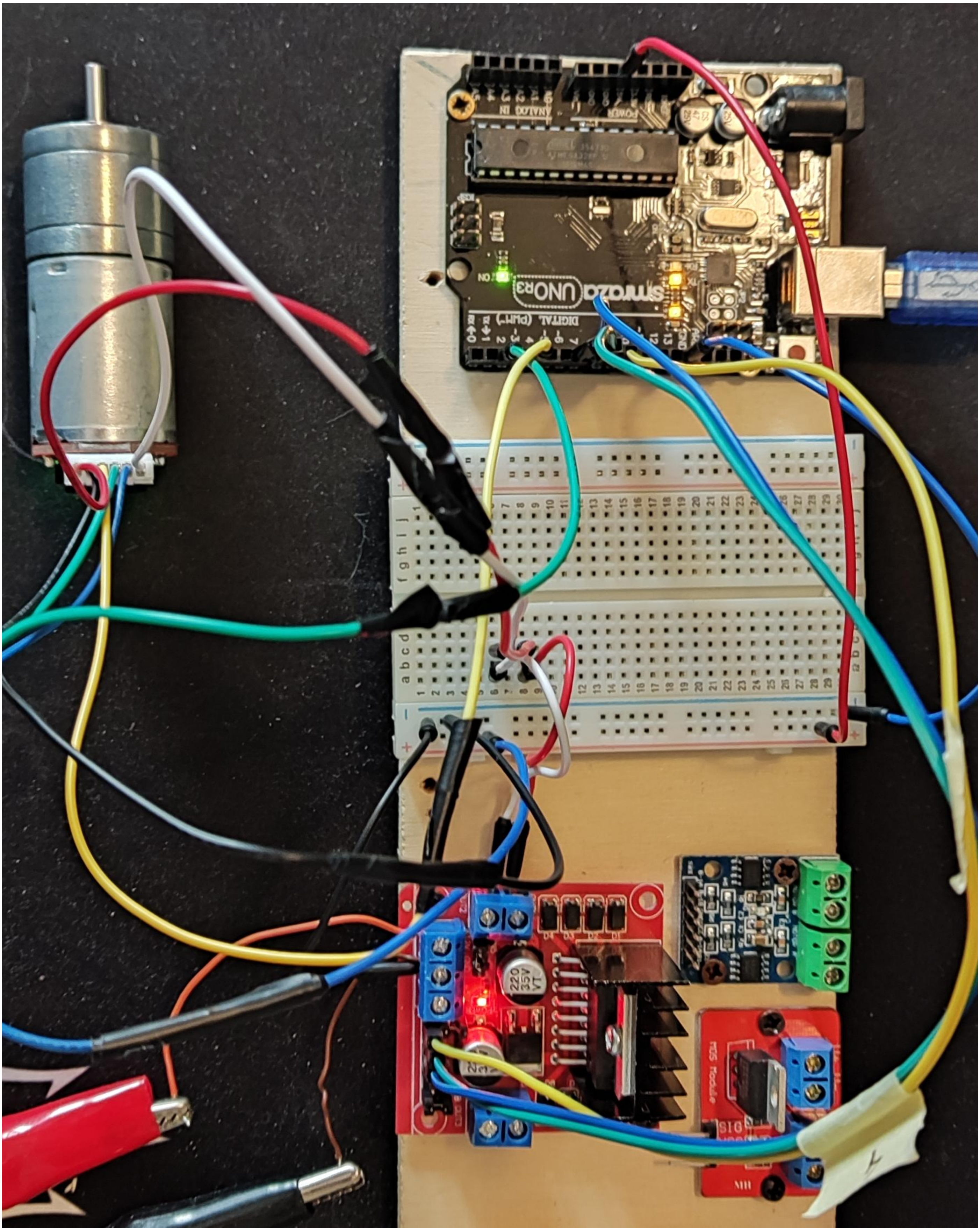
- 🟥 Red - Motor+ (Reverse to control Forward/Reverse of the motor)
- ⬛ Black - Encoder - (3.3-5V, do not connect positive pole and negative pole wrong)
- 🟨 Yellow - Encoder A Phase (11 signals when the motor rotate one circle)
- 🟩 Green - Encoder B Phase (11 signals when the motor rotate one circle)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



L298N driver



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-


smraza UNO R3
ANALOG IN
POWER
DIGITAL (PWM~)
ON
220 35V VT
MOS Module
SIG

CODICE

```
#include <util/atomic.h>

// Encoder signal
#define ENCA 2  // decoder A
#define ENCB 4  // decoder B  --> direction of rotation

// Pins for LN298n Motor Driver
// Motor A connections
int enA = 9;  // PWM signal
            // input1  input2
int in1 = 8;  // High(1) Low(0)  Forward
int in2 = 7;  // Low(0)  High(1) Backward

int pulses_per_revolution= 500;

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

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

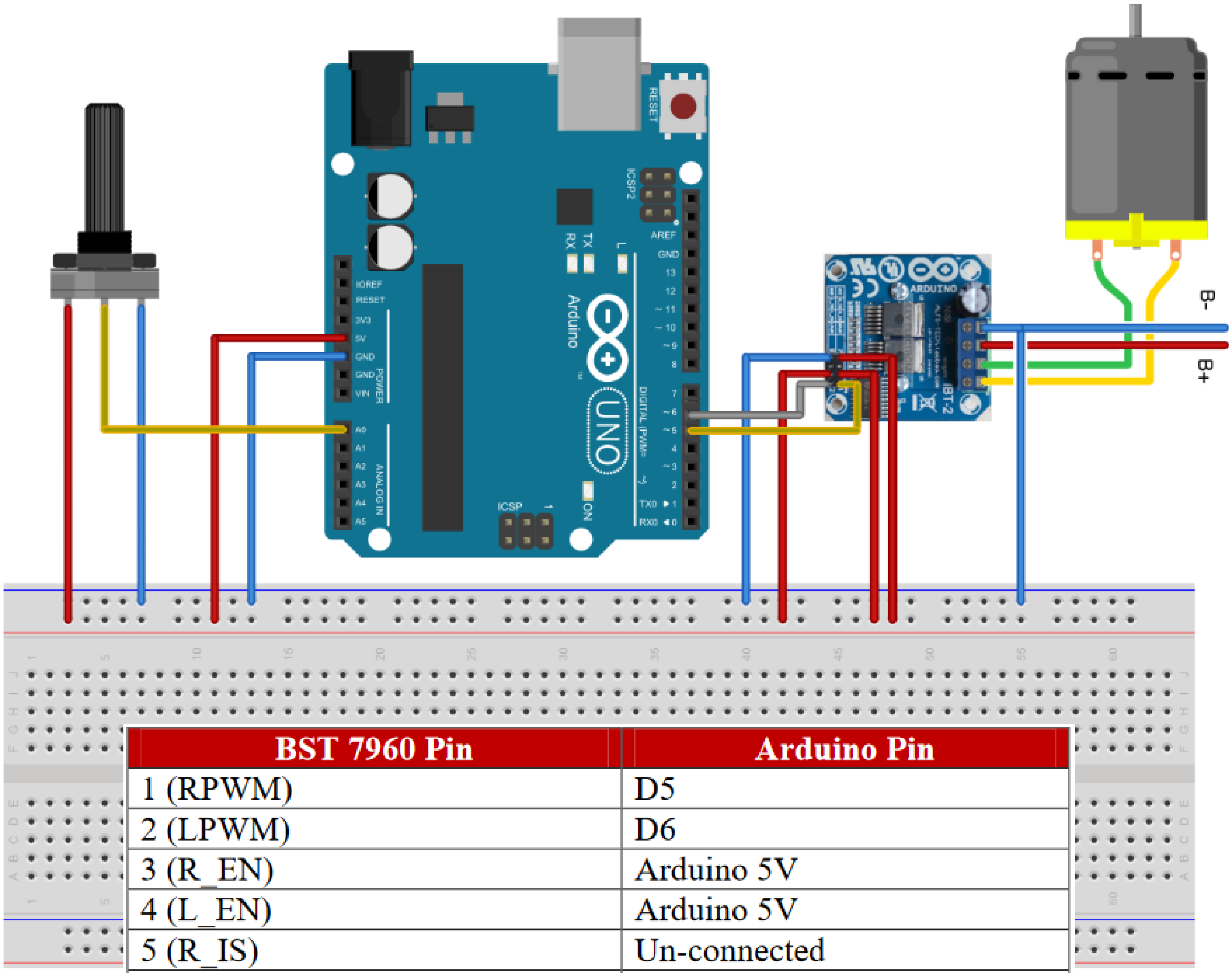
```
// Plotter / serial print
int plotter=1;

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

  // Setup  BTD7960 Motor Driver
  pinMode(ENCA,INPUT);
  pinMode(ENCB,INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);

  // Set all the motor control pins to outputs
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  // Turn off motors - Initial state
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);

  TCCR1B = TCCR1B & 0b11111000 | 1;  // set 31KHz PWM to prevent motor noise
}

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

    // 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/pulses_per_revolution*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
    //v_speed = v1;  // actual speed

    //PID code
    e_speed = set_speed - v_speed;   // error speeed
    // 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 V1filt limits
    if (v1Filt > 150)    {  v1Filt = 150; }
    else if(v1Filt < 0) {  v1Filt = 0;   }
    // set set_speed limits
    if (set_speed > 150)    {  set_speed = 150; }
    else if(set_speed < 0) {  set_speed = 0;   }

    // set DC motor speed
    setMotor(pwm_pulse,enA,in1,in2);

  // print data
  if (plotter==0) {
    if ((currT - prevT_print) >= 0.5e6 ) {
      prevT_print = currT;
      Serial.print(set_speed); Serial.print(" "); Serial.print(v1Filt); Serial.print(" "); Serial.print(pwm_pulse);
Serial.println();
    }
  }
  else
  {
      Serial.print(set_speed); Serial.print(" "); Serial.print(v1Filt); Serial.print(" "); Serial.print(pwm_pulse);
Serial.println();
  }

  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) {

    if (plotter==1) Serial.println(inputString);

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

       // rotation
       if (iPin>=0 && iPin < 255) {
         if (Pin== "1") {
          if (plotter==1 ) Serial.println("DC motor 1 " + Pin + "=" + State);
          //analogWrite(IN1, iPin);
          //analogWrite(IN2, 0);
          set_speed = iPin;
         }
         else if (Pin== "2") {
          if (plotter==1 ) Serial.println("DC motor 2 " + Pin + "=" + State);
          //analogWrite(IN1, iPin);
          //analogWrite(IN2, 0);
         }
         // PID constant
         else if (Pin== "p") {
          if (plotter==1 ) Serial.println("Proportional x 10 " + Pin + "=" + State);
          kp = iPin / 10;
         }
         else if (Pin== "i") {
          if (plotter==1 ) Serial.println("Integral x 10 " + Pin + "=" + State);
          ki = iPin / 10;
         }
         else if (Pin== "d") {
          if (plotter==1 ) Serial.println("Derivative x 10 " + Pin + "=" + State);
          kd = iPin / 10;
         }
         else if (Pin== "plotter") {
          if (plotter==1 ) Serial.println("Print to plotter " + Pin + "=" + State);
          plotter= iPin;
         }

       }
       else {
          if (plotter==1 ) Serial.println("error " + inputString);
          // STOP DC motor
          setMotor(0, enA, in1, in2);
       }
    }
       // clear the input string:
    inputString = "";
    stringComplete = false;
  }
}

void setMotor(int pwmVal, int EN, int IN1, int IN2){
  // For PWM maximum possible values are 0 to 255
  analogWrite(EN, pwmVal);
  // Turn on motor A Forward
  if (pwmVal>0){
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
  }
  else{
    digitalWrite(IN1, LOW);
    digitalWrite(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;
}
```

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

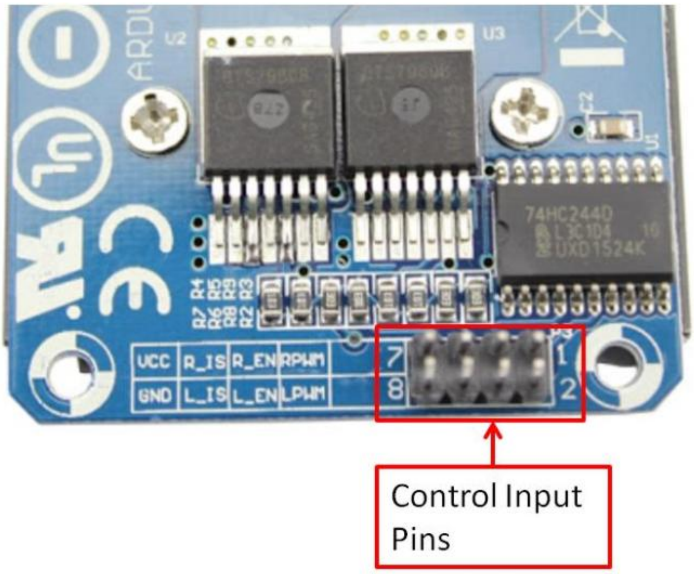
The BTS7960 is a fully integrated high current H bridge module for motor drive applications. Interfacing to a microcontroller is made easy by the integrated driver IC which features logic level inputs, diagnosis with current sense, slew rate adjustment, dead time generation and protection against over 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.

Specifictions:

- 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.

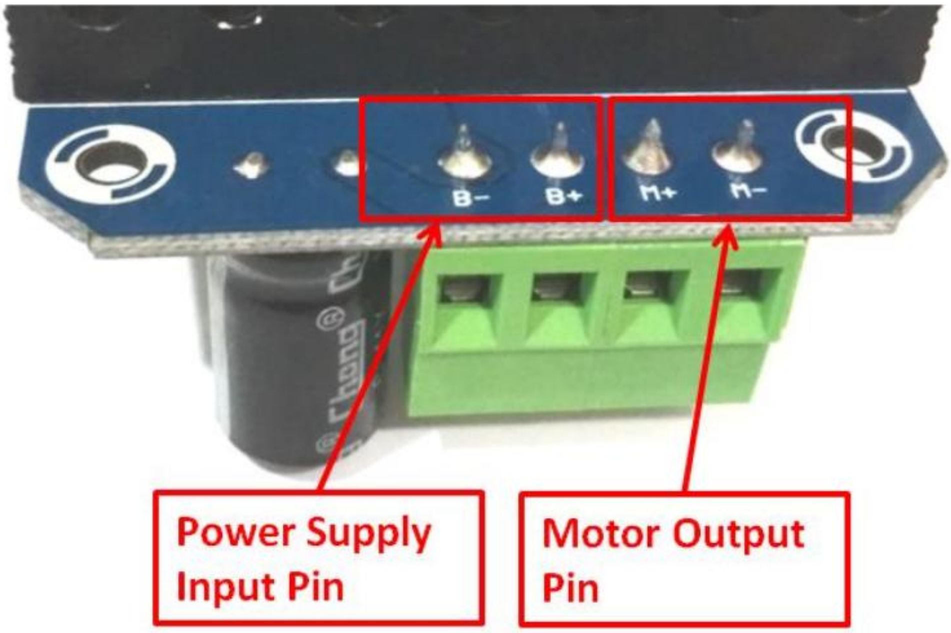


BST 7960 Pin	Arduino Pin
1 (RPWM)	D5
2 (LPWM)	D6
3 (R_EN)	Arduino 5V
4 (L_EN)	Arduino 5V
5 (R_IS)	Un-connected
6 (L_IS)	Un-connected
7 (VCC)	Arduino 5V
8 (GND)	Arduino GND



Control Input Pins

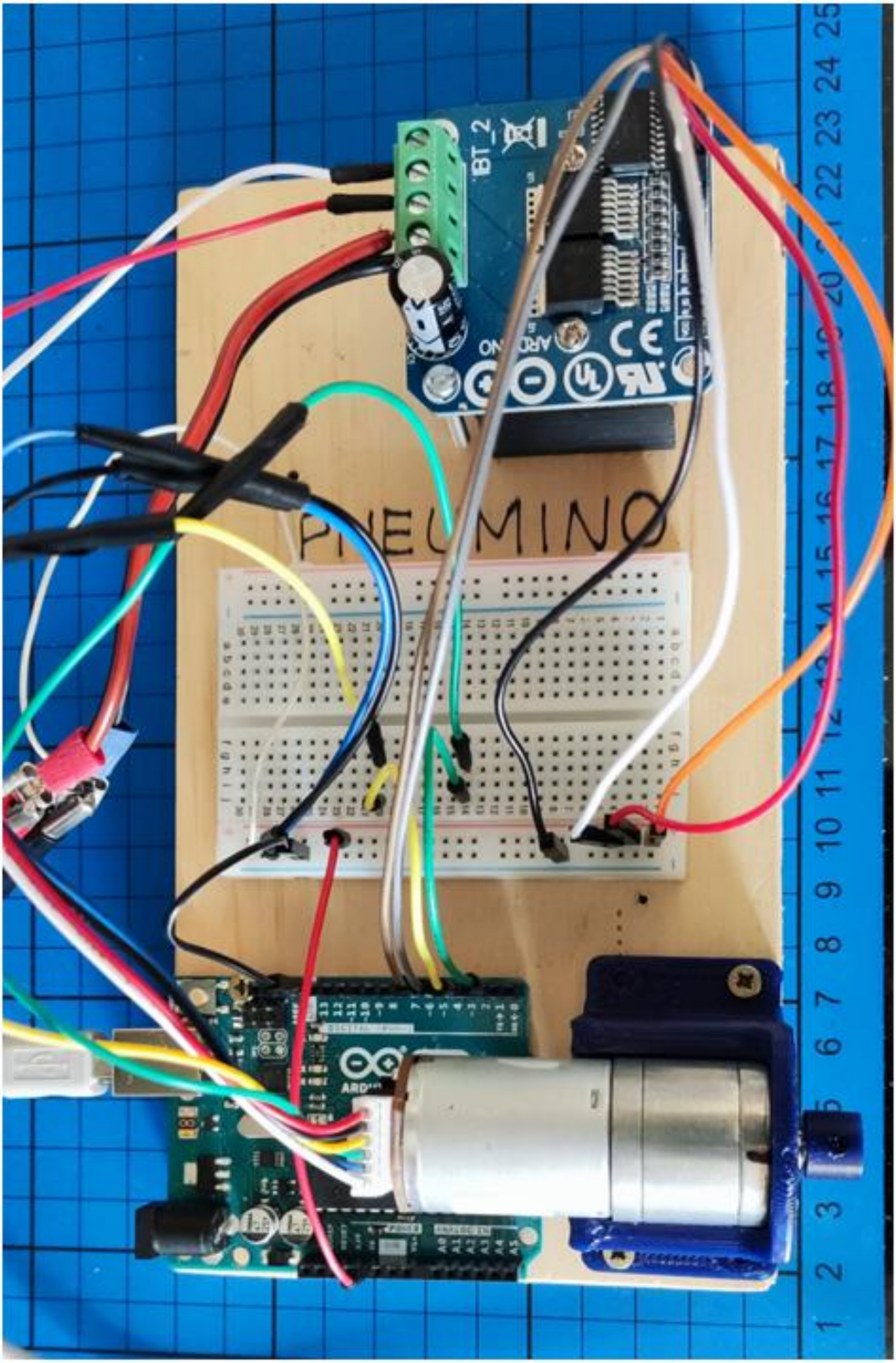
Pin No	Function	Description
1	RPWM	Forward Level or PWM signal, Active High
2	LPWM	Reverse Level or PWM signal, Active High
3	R_EN	Forward Drive Enable Input, Active High/ Low Disable
4	L_EN	Reverse Drive Enable Input, Active High/Low Disable
5	R_IS	Forward Drive, Side current alarm output
6	L_IS	Reverse 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 -


PNEUMINO

CODICE

```
#include <util/atomic.h>


// Pins for BTD7960 Motor Driver
#define ENCA 2  // decoder A
#define ENCB 4  // decoder B  --> direction of rotation
#define IN1 5   // PWM 1  forward
#define IN2 6   // PWM 2  backward


int pulses_per_revolution= 600;


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


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


// Plotter / serial print
int plotter=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);

  TCCR1B = TCCR1B & 0b11111000 | 1;  // set 31KHz PWM to prevent motor noise
}

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

    // 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/pulses_per_revolution*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
    //v_speed = v1;  // actual speed

    //PID code
    e_speed = set_speed - v_speed;   // error speeed
    // 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 V1filt limits
    if (v1Filt > 150)    {  v1Filt = 150; }
    else if(v1Filt < 0) {  v1Filt = 0;   }
    // set set_speed limits
    if (set_speed > 150)    {  set_speed = 150; }
    else if(set_speed < 0) {  set_speed = 0;   }




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

  // print data
  if (plotter==0) {
    if ((currT - prevT_print) >= 0.5e6 ) {
      prevT_print = currT;
```

```
      Serial.print(set_speed); Serial.print(" "); Serial.print(v1Filt); Serial.print(" "); Serial.print(pwm_pulse);
Serial.println();
    }
  }
  else
  {
      Serial.print(set_speed); Serial.print(" "); Serial.print(v1Filt); Serial.print(" "); Serial.print(pwm_pulse);
Serial.println();
  }


  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) {


  if (plotter==1) Serial.println(inputString);


  int id = inputString.indexOf("=");

  if (id>0) {

    //State= inputString.substring(id+1, inputString.length() - id+1);

    State= inputString.substring(id+1, inputString.length());

    Pin = inputString.substring(0, id) ;

    if (plotter==1) Serial.println(id);

    iPin= State.toInt();

    if (plotter==1) Serial.println(Pin + "=" + State);


    // rotation

    if (iPin>=0 && iPin < 255) {

        if (Pin== "1") {

          if (plotter==1 ) Serial.println("DC motor 1 " + Pin + "=" + State);

          //analogWrite(IN1, iPin);

          //analogWrite(IN2, 0);

          set_speed = iPin;

        }

        else if (Pin== "2") {

          if (plotter==1 ) Serial.println("DC motor 2 " + Pin + "=" + State);

          //analogWrite(IN1, iPin);

          //analogWrite(IN2, 0);

        }

        // PID constant

        else if (Pin== "p") {

          if (plotter==1 ) Serial.println("Proportional x 10 " + Pin + "=" + State);

          kp = iPin / 10;
```

```
      }
      else if (Pin== "i") {
       if (plotter==1 ) Serial.println("Integral x 10 " + Pin + "=" + State);
       ki = iPin / 10;
      }
      else if (Pin== "d") {
       if (plotter==1 ) Serial.println("Derivative x 10 " + Pin + "=" + State);
       kd = iPin / 10;
      }
      else if (Pin== "plotter") {
       if (plotter==1 ) Serial.println("Print to plotter " + Pin + "=" + State);
       plotter= iPin;
      }

    }
    else {
       if (plotter==1 ) Serial.println("error " + inputString);
       // STOP DC motor
       analogWrite(IN1, 0);
       analogWrite(IN2, 0);
    }
  }
    // clear the input string:
  inputString = "";
  stringComplete = false;
 }
}

// Drive DC motor
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;

}
```