

La piattaforma hardware e software Arduino: parte III

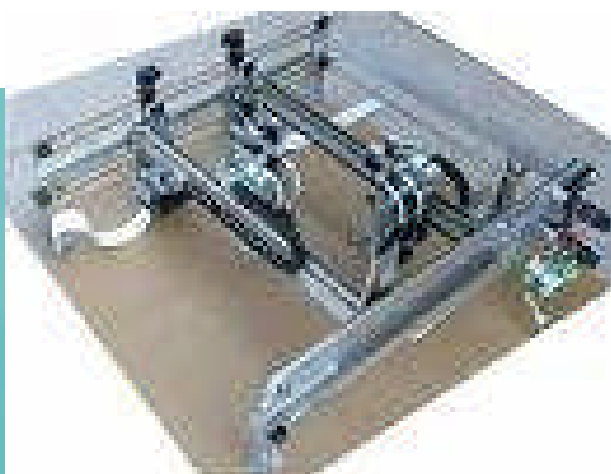
Corso di autoapprendimento

Tempo previsto: 2-3 ore

Prof. Angelo Monfroglio

Questa terza parte del corso tratta il controllo dei motori e dei servomotori con Arduino. Prendiamo in considerazione i motori in continua (DC). Una prima distinzione è fra motori con spazzole (brushed) e senza (brushless). Questi ultimi si stanno diffondendo in robotica e nel modellismo (ad esempio per gli aerei radiocomandati) perché, non avendo spazzole, durano in genere di più. Occorre tenere in conto che motori di questo tipo, anche relativamente piccoli, possono assorbire 50-60 ampere, con tensione di lavoro fra 5 e 12 volt. Vengono usate batterie ricaricabili NiMH (Nichel Metallo Idruro) e sempre più spesso batterie ricaricabili ai polimeri di litio (LiPo). Il termine brushless in realtà è generico e copre svariati motori dalle caratteristiche diverse: sono brushless i motori passo-passo, ad esempio, ma anche altri motori basati su un principio di funzionamento differente. Nel settore che trattiamo, si usano motori a trazione diretta (direct-drive), passo-passo (stepping motor), e demoltiplicati (geared). Il termine servomotore si riferisce in genere a motori in continua che contengono già all'interno dell'involucro un trasduttore di posizione, una demoltiplica e un microchip per il controllo. Hanno sempre 3 fili: rosso per il + dell'alimentazione, nero per la massa, e giallo (o bianco) per il segnale di controllo.

Bisogna sapere che un motore in continua all'avvio assorbe una corrente di spunto anche 10 volte maggiore di quella di regime. Inoltre, anche in stallo assorbe molta corrente. Per questo, a volte, i servomotori, bloccati, si bruciano rapidamente. Nelle applicazioni robotiche, come i bracci a 5 – 6 gradi di libertà, occorre avere servomotori ad alta coppia e robustezza, altrimenti durano pochi giorni.





Le specifiche principali dei motori DC sono:

- tensione di lavoro
- corrente assorbita
- velocità
- coppia in Newton * Metri o Kg * cm
- per i servomotori: fattore di demoltiplica
- dimensioni esterne e diametro dell'asse motore.

Ad esempio:

- tensione fra 9 e 12 volt
- coppia 30 Kg * cm
- velocità: 0.13 secondi per ruotare di 60 gradi.

Consideriamo dapprima un semplice motore in continua. E' una macchina reversibile: se si mette tensione ruota; se si fa ruotare genera tensione. Si può, in questo modo, anche recuperare energia. Per questo motivo un semplice collegamento prevede un diodo in parallelo al motore, con il catodo (il -) sul positivo del motore, e l'anodo (il +) sul negativo. Questo serve per evitare un ritorno dannoso di corrente nel motore, quando si comporta anche come generatore ("rinculo").

Tinkering

Secondo Massimo Banzi, co-fondatore di Arduino, il miglior modo per imparare l'elettronica e la robotica è il "tinkering". Questa nuova parola inglese si può tradurre con armeggiare, metterci le mani, "smanettare": un'arte in cui i nostri Omaristi sono campioni.

Controllo di un piccolo motore in CC con Arduino

Un piccolo e semplice motore in continua si può controllare come se fosse un LED (ripassare le parti I e II del corso). Useremo all'uopo un transistor di media potenza del tipo TIP120, un diodo come 1N4001, un resistore da 470 ohm, Arduino e il piccolo motore che alimentiamo a 5 volt. Con il breadboard colleghiamo il pin 9 di Arduino al resistore, l'altro capo di questo alla base del transistor; poi l'emittore a massa, il collettore al meno del motore insieme con l'anodo del diodo; infine il catodo insieme con il più del motore a 5 volt.

Useremo l'istruzione analogWrite per comandare il motore.

Ora scriviamo lo sketch (il programma in C) per Arduino:

```
//Inserire un valore fra 0 e 9 per stabilire la velocità del motore attraverso il serial monitor

int motorPin = 9;
int val = 0;          // variable to store the data from the serial port

void setup() {
  pinMode(motorPin,OUTPUT);    // declare the motor's pin as output
  Serial.begin(19200);        // connect to the serial port
  Serial.println("Welcome to SerialMotorSpeed!");
  Serial.println("Enter speed number 0-9:");
}

void loop () {
  val = Serial.read();        // read the serial port
  if (val >= '0' && val <= '9' ) {
    val = val - '0';          // convert from character to number
    val = 28 * val;           // convert from 0-9 to 0-255 (almost)
    Serial.print("Setting speed to ");
    Serial.println(val);
    analogWrite(motorPin,val);

    Serial.println("Enter speed number 0-9:");
  }
}
```

I motori in continua, senza trasduttori che rivelano la posizione, cioè comandati ad anello aperto, sono troppo imprecisi per gli usi che trattiamo qui. Invece, si devono usare motori passo – passo o servomotori, o motori in CC con trasduttori di posizione e velocità.

Controllo di servomotori in PWM (Pulse Width Mulation)

I servomotori, molto usati in robotica e nel modellismo, si dividono in due categorie:

- bloccati: si muovono in senso orario e antiorario per meno di un intero giro

-sbloccati: si muovono in modo continuo (360 gradi).

Tutti i servomotori sono comandati attraverso un impulso: di solito circa 1 milli secondo per girare in un senso, circa 2 per il senso opposto e 1,5 per raggiungere lo zero di macchina, cioè la posizione di riferimento. L'intervallo fra gli impulsi inviati determina la velocità.

La libreria servo

Permette di controllare 12 motori e 48 con la scheda Arduino Mega. Come abbiamo già detto, i servomotori contengono trasduttori di posizione. Con la libreria è possibile leggere la posizione corrente dell'asse del motore, attraverso la funzione

read().

La funzione write() sposta l'asse del motore di un certo angolo.

La funzione writeMicroseconds() invia un impulso di un valore in microsecondi atto al comando del motore.

Come si può vedere, la libreria servo di Arduino rende il controllo dei servomotori facilissimo.

Un semplice programma che controlla il movimento di un servomotore

```
/*
 * Servo Move Simple
 * -----
 * Move an R/C servo back and forth (0 to 180 degrees) using
 * delayMicroseconds() for pulse and delay() for time between pulses.
 *
 * Created 18 October 2006
 * copyleft 2006 Tod E. Kurt <tod@todbot.com>
 * http://todbot.com/
 *
 * Adapted from Daniel @
 * http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1160470155/0
 * Rotates servo through 180 degrees, using "servoPulse" function
 * adapted from "Temporary Servo Function" by Tom Igoe and Jeff Gray
 */

int servoPin = 7;           // R/C Servo connected to digital pin
int myAngle;               // angle of the servo (roughly in degrees) 0-
180
int pulseWidth;           // function variable

void servoPulse(int servoPin, int myAngle) {
  pulseWidth = (myAngle * 9) + 700; // converts angle to microseconds
  digitalWrite(servoPin, HIGH);    // set servo high
  delayMicroseconds(pulseWidth);   // wait a very small amount
  digitalWrite(servoPin, LOW);     // set servo low
  Serial.print("pulseWidth: "); Serial.println(pulseWidth);
  delay(20);                       // refresh cycle of typical servos (20
ms)
}

void setup() {
  pinMode(servoPin, OUTPUT);       // set servoPin pin as output
  Serial.begin(19200);
}
```

```

void loop() {
  // cycle through every angle (rotate the servo 180 slowly)
  for (myAngle=0; myAngle<=180; myAngle++) {
    servoPulse(servoPin, myAngle);
  }
  delay(1000);
}

```

Un programma per il controllo di un servomotore con l'uso della libreria Servo.h

```

#include <Servo.h>

int laserPin = 4;
Servo servoV;
Servo servoH;

int x = 90;
int y = 90;
int minX = 10;
int maxX = 170;
int minY = 50;
int maxY = 130;

void setup()
{
  servoH.attach(3);
  servoV.attach(2);
  pinMode(laserPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char ch;
  if (Serial.available())
  {
    ch = Serial.read();
    if (ch == '0')
    {
      digitalWrite(laserPin, LOW);
    }
    else if (ch == '1')
    {
      digitalWrite(laserPin, HIGH);
    }
    else if (ch == '-')
    {
      delay(100);
    }
    else if (ch == 'c')
    {
      x = 90;
      y = 90;
    }
    else if (ch == 'l' || ch == 'r' || ch == 'u' || ch == 'd')

```

```

    {
        moveLaser(ch, 1);
    }
    else if (ch == 'L' || ch == 'R' || ch == 'U' || ch == 'D')
    {
        moveLaser(ch, 5);
    }
}
servoH.write(x);
servoV.write(y);
}

void moveLaser(char dir, int amount)
{
    if ((dir == 'r' || dir == 'R') && x > minX)
    {
        x = x - amount;
    }
    else if ((dir == 'l' || dir == 'L') && x < maxX)
    {
        x = x + amount;
    }
    else if ((dir == 'u' || dir == 'U') && y < maxY)
    {
        y = y + amount;
    }
    else if ((dir == 'd' || dir == 'D') && y > minY)
    {
        y = y - amount;
    }
}
}

```

Un secondo programma che controlla il movimento di un servomotore

Examples > Servo Library

Sweep

Sweeps the shaft of a RC servo motor back and forth across 180 degrees. This example makes use of the Arduino **servo library**.

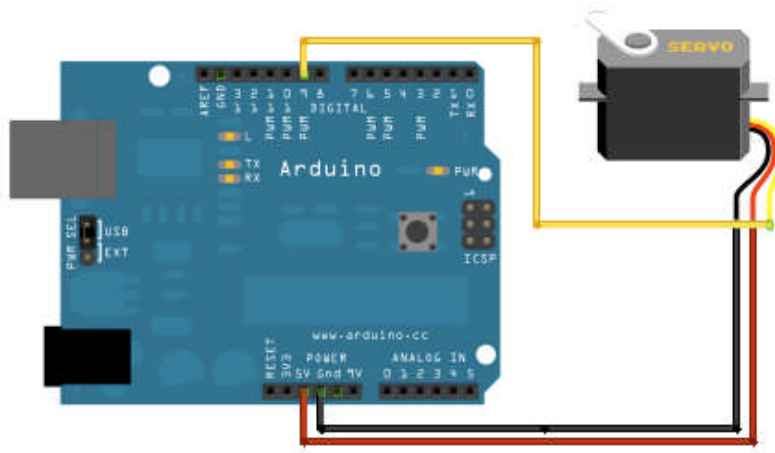
Hardware Required

- + Arduino Board
- + (1) Servo Motor
- + hook-up wire

Circuit

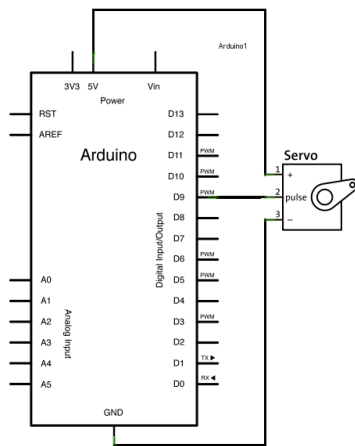
Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to pin 9 on the Arduino board.

click the images to enlarge



images developed using [Fritzing](#). For more circuit examples, see the [Fritzing project page](#)

Schematic



Code

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.

#include <Servo.h>

Servo myservo; // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0; // variable to store the servo position

void setup()
{
  myservo.attach(9); // attaches the servo on pin 9 to the servo
  object
}
```

```

void loop()
{
  for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180
degrees
  {
    myservo.write(pos); // tell servo to go to
position in variable 'pos'
    delay(15); // waits 15ms for the servo
to reach the position
  }
  for(pos = 180; pos>=1; pos-=1) // goes from 180 degrees to 0
degrees
  {
    myservo.write(pos); // tell servo to go to
position in variable 'pos'
    delay(15); // waits 15ms for the servo
to reach the position
  }
}

```

[\[Get Code\]](#)

Un terzo programma per il controllo di un servomotore con l'uso della libreria Wire. Ricordiamo che il linguaggio di Arduino (C e C++) è basato su Wire. L'ambiente di sviluppo (IDE) è basato su Processing (Java).

```

/*
 * NunchuckServo
 *
 * 2007 Tod E. Kurt, http://todbot.com/blog/
 *
 * The Wii Nunchuck reading code is taken from Windmeadow Labs
 * http://www.windmeadow.com/node/42
 */

#include <Wire.h>

int ledPin = 13;
int servoPin = 7; // Control pin for servo motor

int pulseWidth = 0; // Amount to pulse the servo
int refreshTime = 20; // the time in millisecs needed in between pulses
long lastPulse;
int minPulse = 700; // minimum pulse width
int loop_cnt=0;

void setup()
{
  Serial.begin(19200);
  pinMode(servoPin, OUTPUT); // Set servo pin as an output pin
  pulseWidth = minPulse; // Set the motor position to the minimum

  nunchuck_init(); // send the initialization handshake
  Serial.print("NunchuckServo ready\n");
}

```



```

}

void loop()
{
  checkNunchuck();
  updateServo(); // update servo position

  if( nunchuck_zbutton() ) // light the LED if z button is pressed
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin,LOW);

  delay(1); // this is hear to give a known time per loop
}

void checkNunchuck()
{
  if( loop_cnt > 100 ) { // loop()s is every 1msec, this is every 100msec

    nunchuck_get_data();
    nunchuck_print_data();

    float tilt = nunchuck_accelx(); // x-axis, in this case ranges from
~70 - ~185
    tilt = (tilt - 70) * 1.5; // convert to angle in degrees, roughly
    pulseWidth = (tilt * 9) + minPulse; // convert angle to microseconds

    loop_cnt = 0; // reset for
  }
  loop_cnt++;
}

// called every loop().
// uses global variables servoPin, pulsewidth, lastPulse, & refreshTime
void updateServo()
{
  // pulse the servo again if rhe refresh time (20 ms) have passed:
  if (millis() - lastPulse >= refreshTime) {
    digitalWrite(servoPin, HIGH); // Turn the motor on
    delayMicroseconds(pulseWidth); // Length of the pulse sets the motor
position
    digitalWrite(servoPin, LOW); // Turn the motor off
    lastPulse = millis(); // save the time of the last pulse
  }
}

//
// Nunchuck functions
//

static uint8_t nunchuck_buf[6]; // array to store nunchuck data,

// initialize the I2C system, join the I2C bus,
// and tell the nunchuck we're talking to it
void nunchuck_init()
{
  Wire.begin(); // join i2c bus as master

```

```

Wire.beginTransaction(0x52); // transmit to device 0x52
Wire.send(0x40);           // sends memory address
Wire.send(0x00);           // sends sent a zero.
Wire.endTransmission();   // stop transmitting
}

// Send a request for data to the nunchuck
// was "send_zero()"
void nunchuck_send_request()
{
  Wire.beginTransaction(0x52); // transmit to device 0x52
  Wire.send(0x00);             // sends one byte
  Wire.endTransmission();     // stop transmitting
}

// Receive data back from the nunchuck,
// returns 1 on successful read. returns 0 on failure
int nunchuck_get_data()
{
  int cnt=0;
  Wire.requestFrom (0x52, 6); // request data from nunchuck
  while (Wire.available ()) {
    // receive byte as an integer
    nunchuck_buf[cnt] = nunchuk_decode_byte(Wire.receive());
    cnt++;
  }
  nunchuck_send_request(); // send request for next data payload
  // If we recieved the 6 bytes, then go print them
  if (cnt >= 5) {
    return 1; // success
  }
  return 0; //failure
}

// Print the input data we have recieved
// accel data is 10 bits long
// so we read 8 bits, then we have to add
// on the last 2 bits. That is why I
// multiply them by 2 * 2
void nunchuck_print_data()
{
  static int i=0;
  int joy_x_axis = nunchuck_buf[0];
  int joy_y_axis = nunchuck_buf[1];
  int accel_x_axis = nunchuck_buf[2]; // * 2 * 2;
  int accel_y_axis = nunchuck_buf[3]; // * 2 * 2;
  int accel_z_axis = nunchuck_buf[4]; // * 2 * 2;

  int z_button = 0;
  int c_button = 0;

  // byte nunchuck_buf[5] contains bits for z and c buttons
  // it also contains the least significant bits for the accelerometer data
  // so we have to check each bit of byte outbuf[5]
  if ((nunchuck_buf[5] >> 0) & 1)
    z_button = 1;
  if ((nunchuck_buf[5] >> 1) & 1)
    c_button = 1;

  if ((nunchuck_buf[5] >> 2) & 1)

```

```

    accel_x_axis += 2;
if ((nunchuck_buf[5] >> 3) & 1)
    accel_x_axis += 1;

if ((nunchuck_buf[5] >> 4) & 1)
    accel_y_axis += 2;
if ((nunchuck_buf[5] >> 5) & 1)
    accel_y_axis += 1;

if ((nunchuck_buf[5] >> 6) & 1)
    accel_z_axis += 2;
if ((nunchuck_buf[5] >> 7) & 1)
    accel_z_axis += 1;

Serial.print(i,DEC);
Serial.print("\t");

Serial.print("joy:");
Serial.print(joy_x_axis,DEC);
Serial.print(",");
Serial.print(joy_y_axis, DEC);
Serial.print(" \t");

Serial.print("acc:");
Serial.print(accel_x_axis, DEC);
Serial.print(",");
Serial.print(accel_y_axis, DEC);
Serial.print(",");
Serial.print(accel_z_axis, DEC);
Serial.print("\t");

Serial.print("but:");
Serial.print(z_button, DEC);
Serial.print(",");
Serial.print(c_button, DEC);

Serial.print("\r\n"); // newline
i++;
}

// Encode data to format that most wiimote drivers except
// only needed if you use one of the regular wiimote drivers
char nunchuk_decode_byte (char x)
{
    x = (x ^ 0x17) + 0x17;
    return x;
}

// returns zbutton state: 1=pressed, 0=notpressed
int nunchuck_zbutton()
{
    return ((nunchuck_buf[5] >> 0) & 1) ? 0 : 1; // voodoo
}

// returns zbutton state: 1=pressed, 0=notpressed
int nunchuck_cbutton()
{
    return ((nunchuck_buf[5] >> 1) & 1) ? 0 : 1; // voodoo
}

```

```

// returns value of x-axis joystick
int nunchuck_joyx()
{
    return nunchuck_buf[0];
}

// returns value of y-axis joystick
int nunchuck_joyy()
{
    return nunchuck_buf[1];
}

// returns value of x-axis accelerometer
int nunchuck_accelx()
{
    return nunchuck_buf[2];    // FIXME: this leaves out 2-bits of the data
}

// returns value of y-axis accelerometer
int nunchuck_accely()
{
    return nunchuck_buf[3];    // FIXME: this leaves out 2-bits of the data
}

// returns value of z-axis accelerometer
int nunchuck_accelz()
{
    return nunchuck_buf[4];    // FIXME: this leaves out 2-bits of the data
}

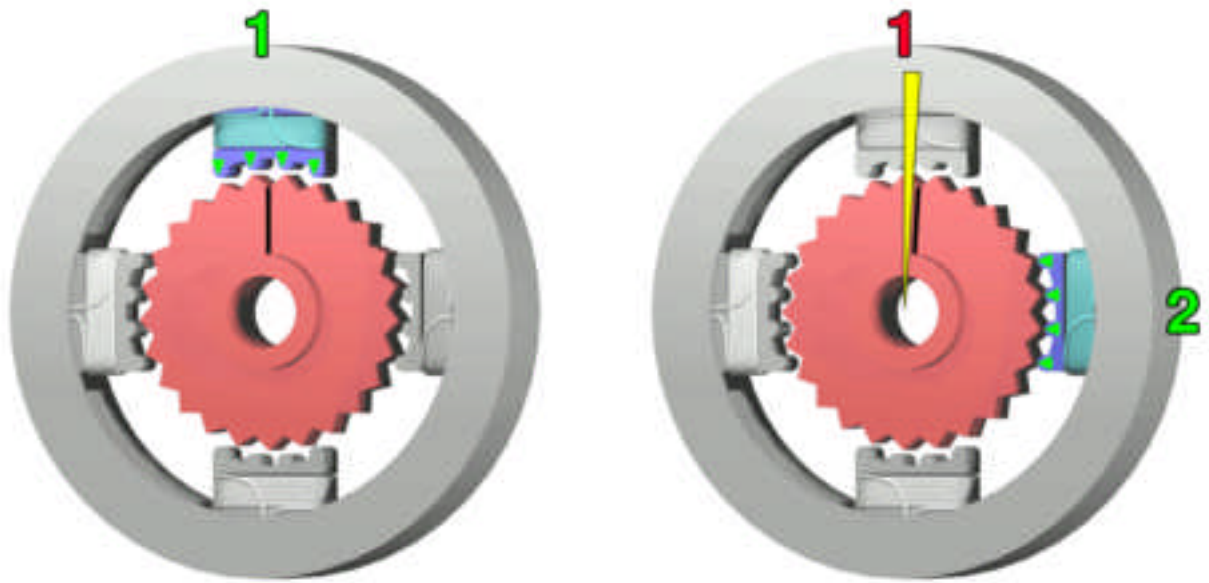
```

Il controllo di motori passo - passo

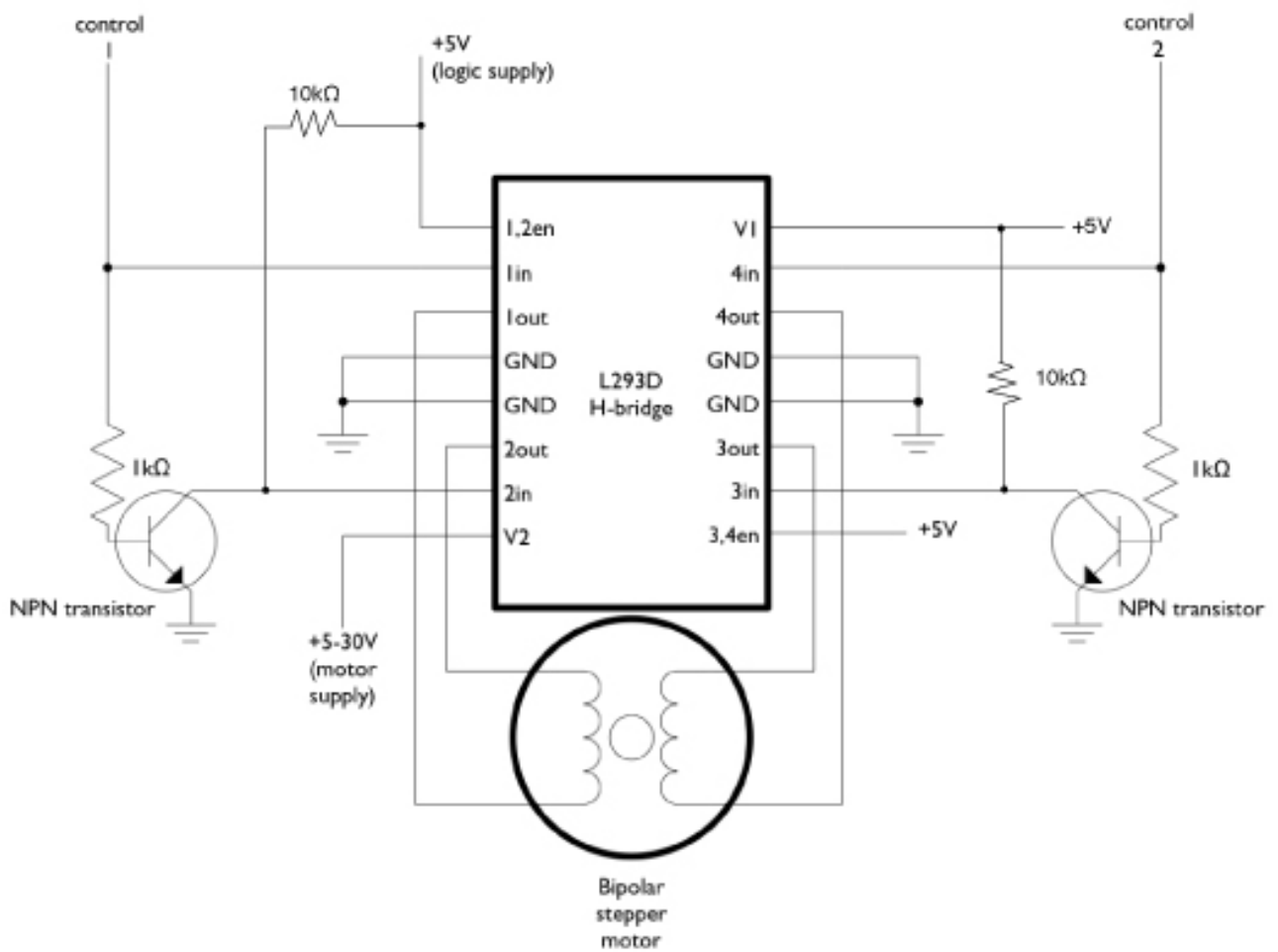
I motori passo - passo sono motori professionali presenti in Robotica, nelle stampanti, nei lettori CD, DVD, ecc. Il controllo è difficile: nella nostra provincia viene spiegato solo all'ITIS Omar di Novara. Con Arduino è un po' più facile. Come suggerisce il nome i motori passo – passo si muovono di un precisato passo alla volta, ad esempio 1,8 gradi, oppure 2 gradi (ma ne esistono anche demoltiplicati con 0,02 gradi per passo). Ciò è ottenuto attivando in modo appropriato le bobine di statore. Il campo magnetico sposta il rotore, a magneti permanenti. I motori passo – passo non hanno spazzole e sono di estrema precisione. Si possono, in linea di principio, comandare ad anello aperto, senza retroazione e sensori. Tuttavia, spesso vengono muniti di fine corsa e zero di macchina. Il loro controllo viene progettato solo in pochi laboratori specializzati nell'industria, e, in campo scolastico, solo all'Omar (da oltre 30 anni). Si dividono in

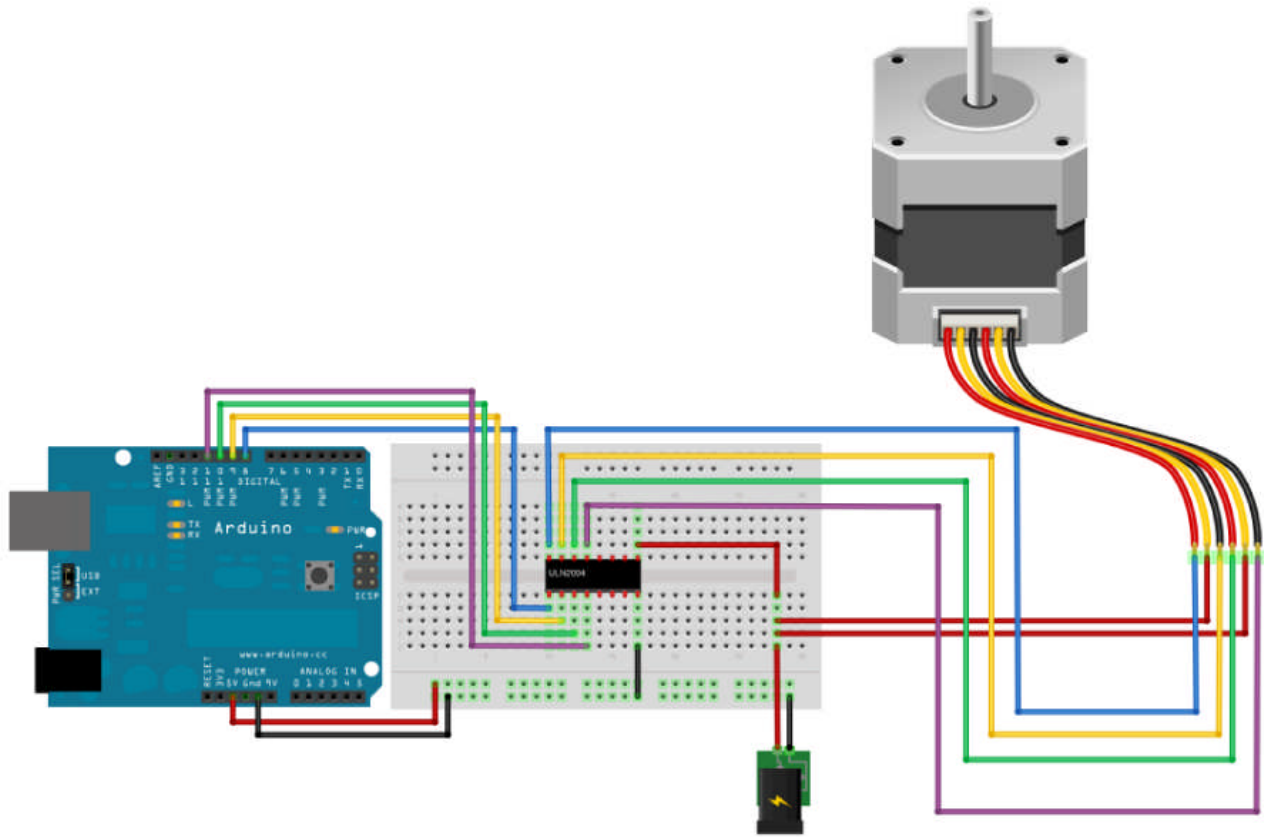
- unipolari: la corrente nelle bobine scorre sempre in una direzione
- bipolari: la corrente scorre in 2 direzioni

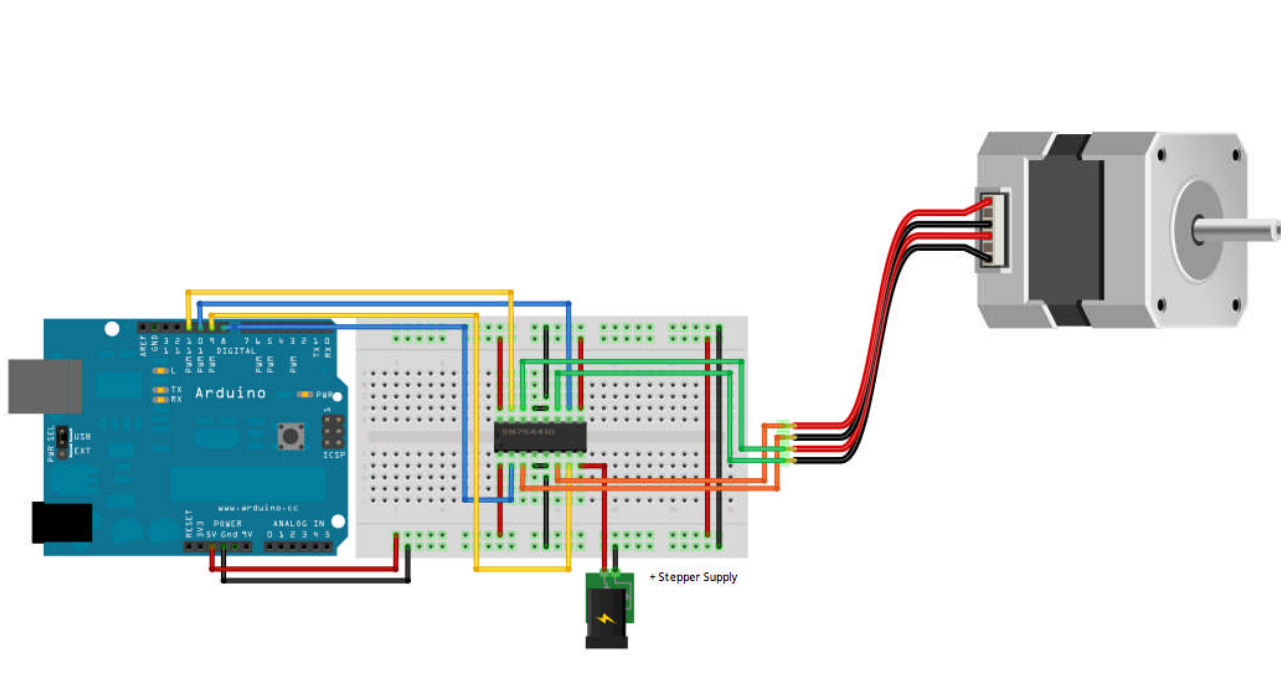
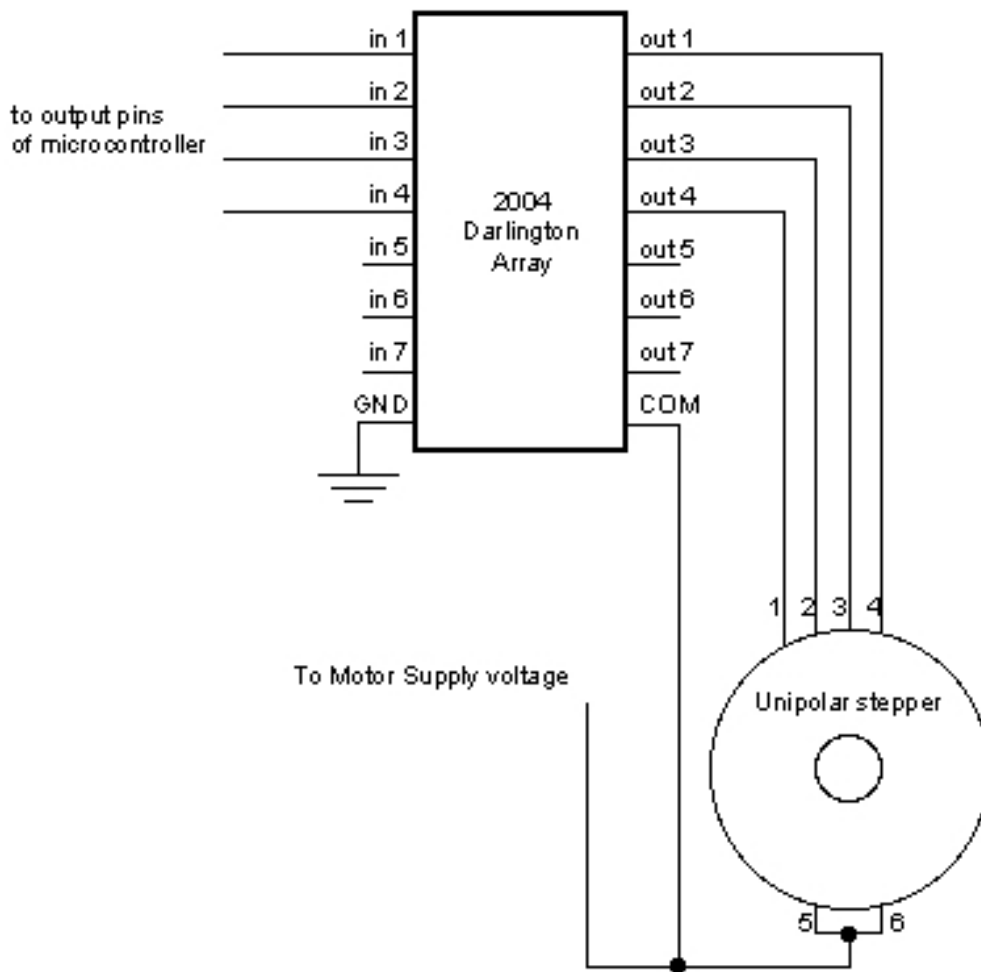
Il comando può essere a passo intero o anche a mezzo passo (attivazione bobine A, A-B, B, ecc.).



ecc.







La libreria di Arduino (C++) per i motori passo – passo (stepper)

Stepper(steps,pin1,pin2,pin3,pin4)
Stepper(steps,pin1,pin2)

Viene create un'istanza della classe Stepper per muovere il motore di un numero steps di passi

setSpeed(rpms)

Fissa la velocità in giri al minuto

Step(steps)

Muove un motore precedentemente definito di un certo di numero passi alla velocità definita prima con setSpeed

Un programma per il comando di motori passo – passo

```
(per entrambi i circuiti)
/*
 * MotorKnob
 *
 * A stepper motor follows the turns of a potentiometer
 * (or other sensor) on analog input 0.
 *
 * http://www.arduino.cc/en/Reference/Stepper
 * This example code is in the public domain.
 */

#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 100

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 9, 10, 11);

// the previous reading from the analog input
int previous = 0;

void setup()
{
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(30);
}

void loop()
{
  // get the sensor value
  int val = analogRead(0);

  // move a number of steps equal to the change in the
```



```
// sensor reading
stepper.step(val - previous);

// remember the previous value of the sensor
previous = val;
}
```

Schede aggiuntive per il controllo di motori (Arduino shield)

Il successo mondiale di Arduino ha fatto sì che siano state costruite molte schede aggiuntive, chiamate shield (scudi) da incastrare sulla basetta Arduino. Coprono tutte le applicazioni: reti, GPS, satelliti Galileo, sensori di ogni tipo, ecc.; e, naturalmente il controllo di motori. Come per Arduino, rappresentano lo stato dell'arte e sono le più convenienti al mondo.



Uno 'shield' per il controllo di motori.

Bibliografia

- M. Banzi, Getting started with Arduino, O'Reilly, Cambridge, Beijing, 2009
- Tod E. Kurt, Bionic Arduino, MachineProject, 2007
- S. Monk, 30 Arduino Projects, McGrawHill, New York, 2010
- M. Schmidt, Arduino, A Quick-Start Guide, The Pragmatic Programmers, 2011
- www.arduino.cc

Grazie per l'attenzione. Buon apprendimento.