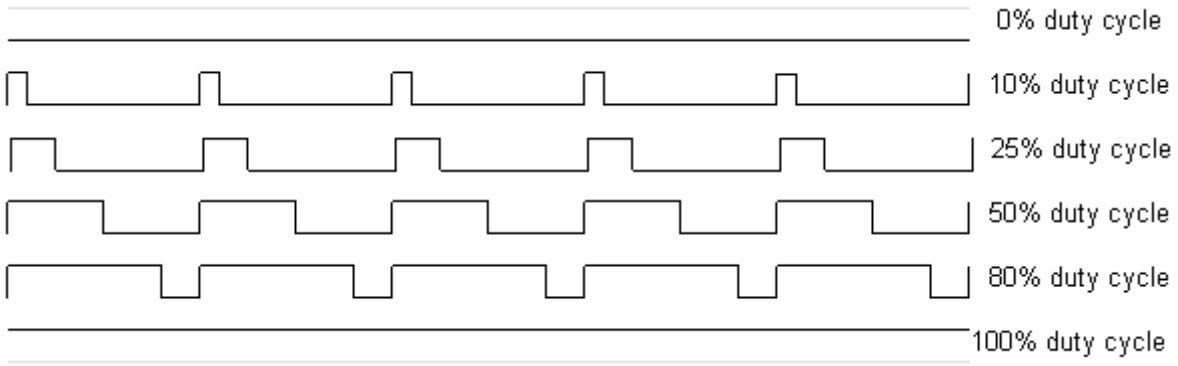
Corso base su arduino – Quarta parte

Uscite analogiche - hardware

Il microcontrollore di arduino Uno non è dotato di convertitori digitali analogici (DAC) in senso stretto. Quando si parla di uscite analogiche ci si riferisce in realtà a piedini che possono operare con la tecnica PWM, modulazione a larghezza d'impulso. Al carico viene inviata una forma d'onda periodica rettangolare, con livelli 0V e 5V, il cui valore medio è proporzionale al duty-cycle. Se il carico è costituito da un LED, oppure da un motore in corrente continua, passo – passo, brushless, l'effetto risultante è praticamente lo stesso di quello che si otterrebbe con un'uscita analogica.
Per quanti non conoscessero il funzionamento del PWM, consiglio l'ottimo articolo presente a questo indirizzo web: http://www.maffucci.it/2011/11/29/arduino-lezione-06-modulazione-di-larghezza-di-impulso-pwm/

La figura seguente, tratta dal sito www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM mostra invece il segnale in uscita in funzione del duty – cycle:



Con un duty – cycle dello 0% otteniamo in uscita una tensione stabile a 0 V, mentre con un duty – cycle del 100%, la tensione è stabile a 5 V.
Con un filtro passa basso è possibile estrarne il valor medio, ma il filtro non serve nel caso di segnalatori ottici (LED, display) a causa della persistenza delle immagini sulla retina e nemmeno nel caso di piccoli motori, dato che si comportano essi stessi da filtro.

I piedini della scheda abilitati al PWM sono identificabili tramite il simbolo ~ posto accanto al numero del piedino stesso. Nell'arduino Uno sono in tutto 6 : 3, 5, 6, 9, 10, 11 e sono evidenziati in colore verde nella figura che segue, tratta dal sito: http://www.mauroalfieri.it/elettronica/frequenza-pwm-arduino-duty-cycle.html

Può risultare utile conoscere la frequenza dell'onda rettangolare che non è uguale per tutti i piedini. Il 5 ed il 6 operano a 976 Hz, mentre per i rimanenti piedini la frequenza è dimezzata quindi pari a 488 Hz. Tali segnali vengono generati dai 3 timer interni del microcontrollore: il timer 0 gestisce il PWM sui piedini 5 e 6, il timer 1 sul 9 e sul 10, il timer 2 sui piedini 3 e 11.

A chi conosce il funzionamento dei motori in tecnica PWM non sarà sfuggito che le frequenze sono troppo basse per un uso professionale. In genere si opera a frequenze ultrasoniche. Occorre tenere presente che la scheda arduino Uno è nata per realizzare prototipi e comunque, non ho notato inconvenienti con i LED o con piccoli motori in corrente continua. Per i più esigenti: è possibile modificare la frequenza agendo sui registri dei timer, si veda http://www.mauroalfieri.it/elettronica/frequenza-pwm-arduino-duty-cycle.html .

Tenete presente che il timer 0 è utilizzato anche dalle funzioni delay( ) e millis( ), pertanto è sconsigliabile modificare la frequenza sui piedini 5 e 6 se fate uso di tali funzioni nel vostro programma.

Le uscite analogiche – software

Il PWM viene attivato tramite la funzione *analogWrite(piedino, valore)*, dove *piedino* identifica una delle uscite descritte in precedenza e *valore* è una costante o una variabile intera da 0 a 255. Non è necessario definire il piedino come uscita tramite la funzione *pinMode*. Sul piedino specificato verrà generata un'onda rettangolare che cesserà in seguito ad una chiamata a *digitalRead* o *digitalWrite*. Sui piedini 5 e 6 il duty – cycle risulterà maggiore del previsto e questo lo si noterà con valori bassi (0 – 10). Ciò è dovuto al fatto che il timer utilizzato per il PWM su tali piedini è usato anche dalle funzioni *delay* e *millis*.

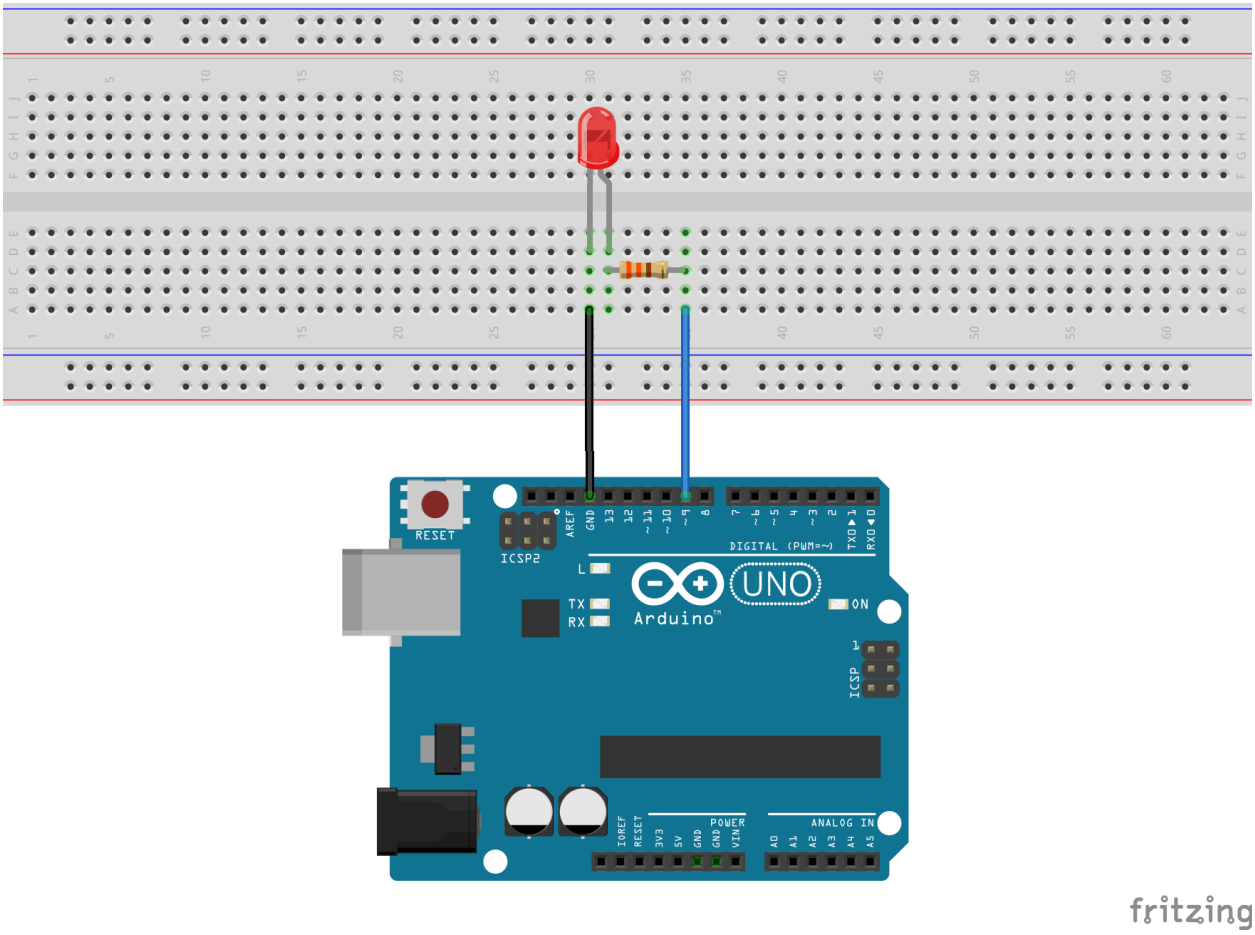
Esempi:

```
analogWrite(9, 128);  //Genera sul piedino 9 un'onda rettangolare con duty – cycle 50%

for (int i = 0; i < 256; i++) { //incrementa il duty – cycle sul piedino 3 gradualmente
        analogWrite (3, i);     //da 0% al 100% con ritardo tra un passo e l'altro di 30 ms
        delay(30);
}
```

Alcune applicazioni laboratoriali

La prima semplice applicazione consiste nel far variare la luminosità di un LED dal minimo al massimo e poi dal massimo al minimo, ciclicamente. L'esempio è tratto da www.arduino.cc/en/tutorial/fading . Il LED è collegato tra il piedino 9, che è abilitato all'uscita in PWM, e massa con in serie una resistenza di limitazione della corrente (220Ω – 470Ω).

Data la semplicità dei collegamenti, non riporto lo schema circuitale ma solo quello di cablaggio.



Il programma esegue un ciclo completo in circa 15 secondi e viene riportato qui di seguito:

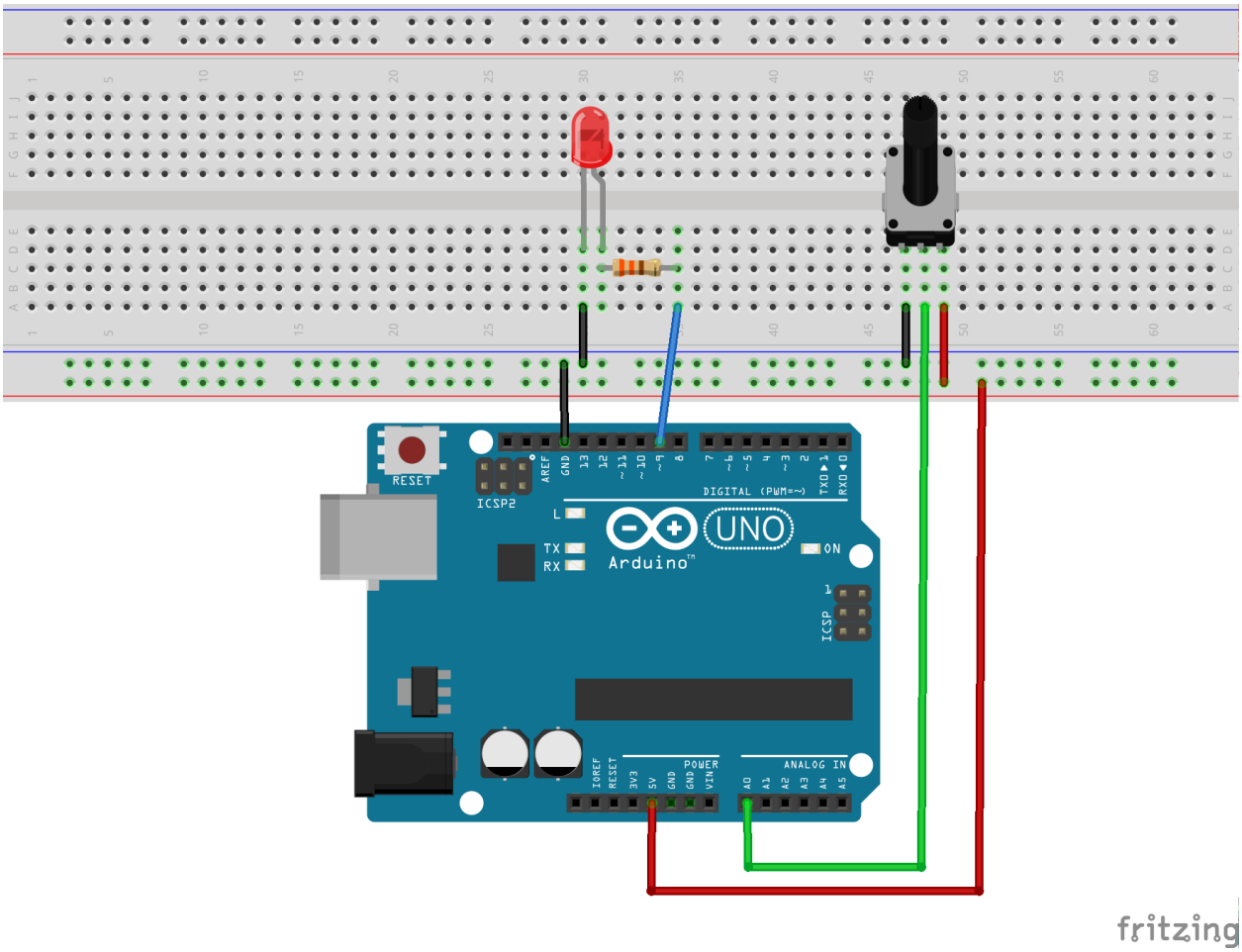
```
int ledPin = 9; // LED collegato al piedino 9

void setup() {
  pinMode(ledPin, OUTPUT); // Facoltativo
}

void loop() {
  // dal minimo al massimo con incrementi di 5 unità
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
    // faccio variare il valore (range da 0 a 255):
    analogWrite(ledPin, fadeValue);
    // attesa di 30 millisecondi per vedere l'effetto
    delay(30);
  }

  // dal massimo al minimo con decrementi di 5 unità
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
    // faccio variare il valore (range da 255 a 0):
    analogWrite(ledPin, fadeValue);
    // attesa di 30 millisecondi per vedere l'effetto
    delay(30);
  }
}
```

Aumentiamo leggermente la complessità del programma precedente inserendo un comando manuale della luminosità mediante un potenziometro collegato all'ingresso analogico A0. Ruotando il cursore del potenziometro varieremo la luminosità del LED, realizzando di fatto quello che viene

comunemente denominato “Dimmer”. C’è però un problema: l’ingresso analogico utilizzato per il potenziometro fornisce valori da 0 a 1023, mentre la funzione *analogWrite( )* richiede un range da 0 a 255. Si può risolvere in due modi: dividendo per 4 il valore letto dal potenziometro, oppure usando la funzione *map( )* e riconducendo l’intervallo 0 – 1023 nell’intervallo 0 – 255.

Al posto di un LED di piccola potenza è possibile comandare una striscia di LED oppure un LED di potenza superiore interponendo un transistor tra l’uscita di arduino ed il carico.

Anche in questo caso, data la semplicità dei collegamenti, non riporto lo schema circuitale ma solo quello di cablaggio.



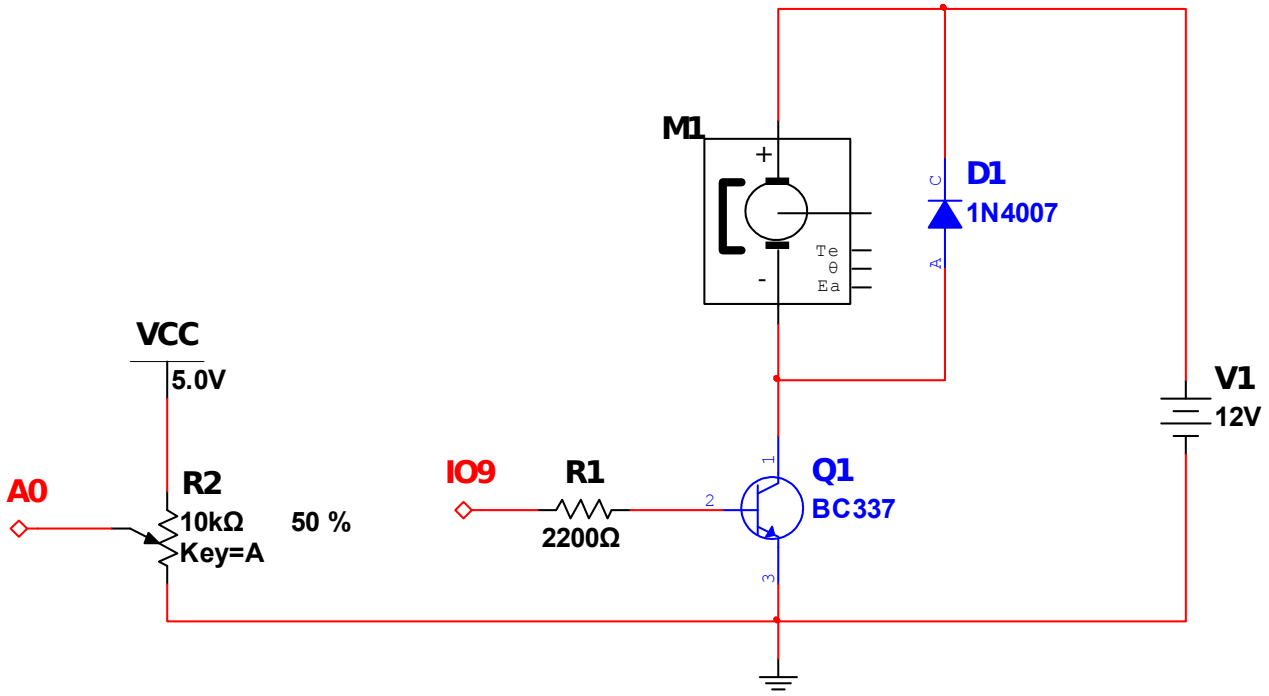
Il programma che lo gestisce è il seguente:

```
/* Fa variare la luminosità di un LED collegato al piedino 9
   in base alla posizione del cursore di un potenziometro.
   Il potenziometro ha i capi estremi collegati a +5V e GND
   e il cursore centrale collegato all’ingresso analogico A0
*/

int LED = 9;  //piedino dove è collegato l’anodo del LED
int potenza = 0; //variabile che contiene il valore letto da A0

void setup() {
  pinMode(LED, OUTPUT); //Facoltativo
}
```

```
void loop(){
  potenza = analogRead(A0); //Leggo la tensione dal potenziometro (0-1023)
  potenza = potenza / 4;    //Riconduco all'intervallo 0-255
  analogWrite(LED, potenza);//Mando il valore al LED
  delay(50);                //Aspetto 50 ms prima di rileggere il potenziometro
}
```

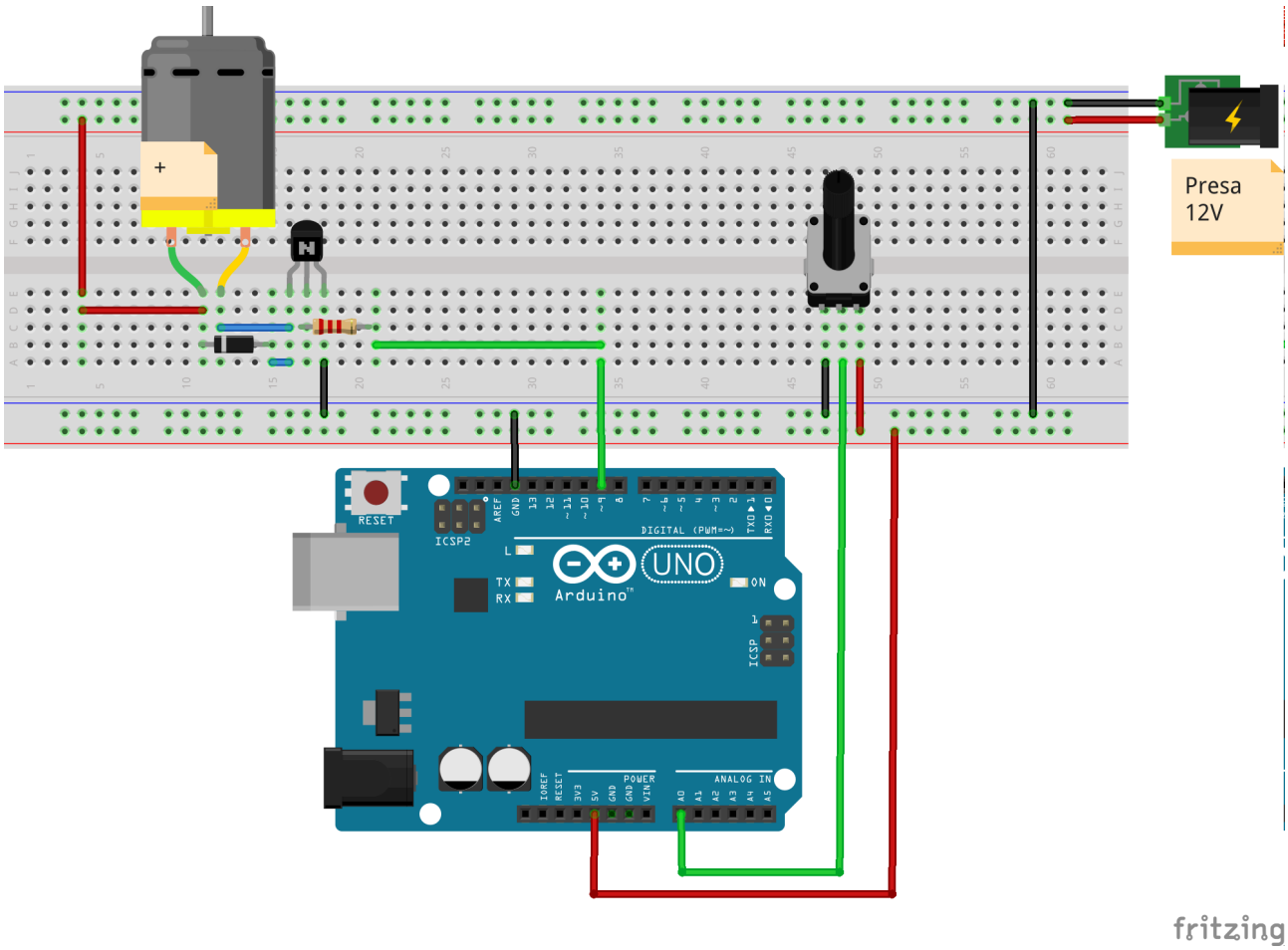
Allo stesso modo possiamo regolare la velocità di una ventola da PC, ma con alcuni accorgimenti. Le ventole utilizzate nei PC sono mosse da motori brushless (senza spazzole) e contengono al loro interno una parte elettronica di comando, pertanto hanno una polarità precisa che non può essere invertita. Il filo rosso della ventola va al positivo dell'alimentazione, mentre il filo nero va al negativo. Richiedono tensioni e correnti che i piedini d'uscita di arduino non sono in grado di fornire (tensione 12V, corrente di circa 120-150mA). É vero che alcune riescono a ruotare anche con tensioni minime di 3V, ma la corrente assorbita rischia di essere sempre oltre i limiti della scheda arduino. Allora si rende necessario l'utilizzo di un amplificatore di corrente che, in questo caso, sarà costituito da un transistor funzionante come interruttore di potenza. Un transistor adatto allo scopo è il BC337. Nello schema è presente anche un diodo che ha lo scopo di proteggere il transistor (1N4007).

Lo schema circuitale è il seguente:



Il componente M1 rappresenta la ventola; la tensione di 5V si ricava da arduino, mentre la tensione di 12V deve provenire da un alimentatore esterno. Occorre prestare la massima attenzione alla polarità dei componenti: transistor, diodo, ventola.

Di seguito viene riportato lo schema di cablaggio. Nel disegno, la ventola viene rappresentata come un piccolo motore in corrente continua (non ho reperito un componente più adatto) e l'alimentazione esterna a 12V mediante la presa in alto a destra. Il transistor è il componente con la lettera N e la sua piedinatura rispetta quella del BC337.
Da notare anche che il negativo dell'alimentazione a 12V deve essere collegato al GND di arduino.



Il programma si comporta come il precedente, ma va tenuto presente che la ventola non potrà funzionare con tensioni inferiori a qualche volt. Allora, per una parte della corsa del potenziometro, la ventola rimarrà ferma. Si può rimediare mappando il range 0-1023 del potenziometro sul range 80-255 (dipende dalla ventola).

```
/* Fa variare la velocità di una ventola collegata al piedino 9
   in base alla posizione del cursore di un potenziometro.
   Il potenziometro ha i capi estremi collegati a +5V e GND
   e il cursore centrale collegato all'ingresso analogico A0.
*/

int ventola = 9; //piedino dove è collegata la ventola (transistor)
int potenza = 0; //variabile che contiene il valore letto da A0

void setup() {
  pinMode(ventola, OUTPUT); //Facoltativo
}

void loop(){
  potenza = analogRead(A0); //Leggo la tensione dal potenziometro (0-1023)
  potenza = potenza / 4;    //Riconduco all'intervallo 0-255
  analogWrite(ventola, potenza);//Mando il valore alla ventola
  delay(50);                //Aspetto 50 ms prima di rileggere il potenziometro
}
```

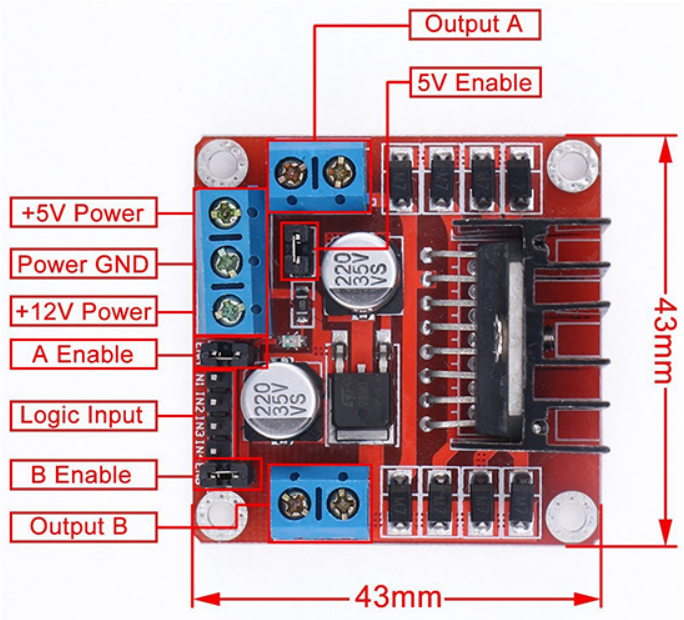
Inserendo un sensore di temperatura LM35 e modificando opportunamente il programma precedente è possibile attivare la ventola ad un valore prestabilito di temperatura e farla ruotare ad una velocità proporzionale alla temperatura stessa.

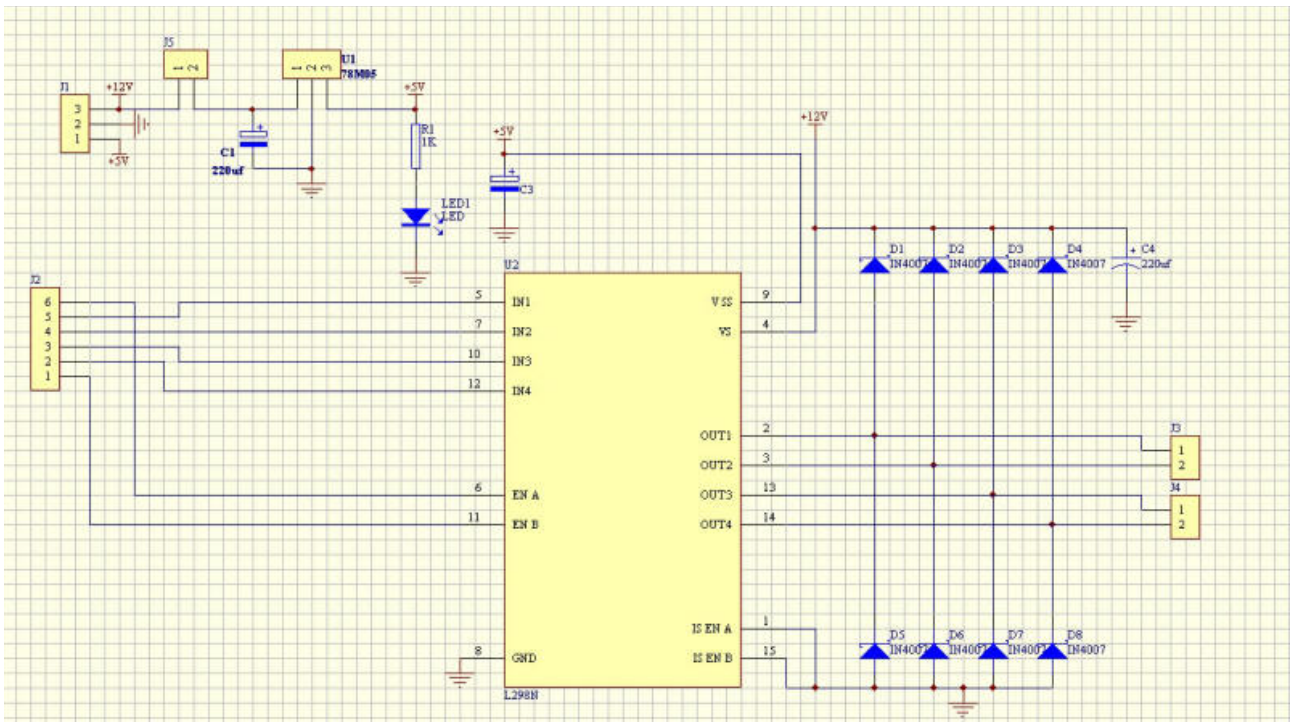
Al posto della ventola si può collegare un piccolo motore in corrente continua a magnete permanente. In tal caso, il verso di rotazione dipenderà dalla polarità con cui è inserito il motore nel circuito. A differenza del motore brushless infatti, il motore in DC inverte il verso di rotazione se si inverte la polarità della tensione che lo alimenta.

Qualora volessimo controllare anche il verso di rotazione del motorino in continua tramite software, dovremmo impiegare un circuito noto come ponte H e che si trova in commercio sotto forma di circuito integrato. Oltre ai piedini di alimentazione, il ponte H dispone di 3 ingressi: enable, in1 e in2. Il funzionamento può essere schematizzato secondo la tabella seguente:

enable	**in1**	**in2**	**motore**
0	X	X	fermo
1	0	0	fermo
1	1	1	fermo
1	0	1	Ruota in un senso
1	1	0	Ruota in senso opposto

1 rappresenta un livello alto; 0 rappresenta un livello basso; X rappresenta 0 oppure 1 indifferentemente.
Si può far variare la velocità di rotazione comandando l'ingresso di enable con un piedino di uscita PWM di arduino. Due integrati comunemente usati con arduino sono L293 e L298 ed entrambi contengono al loro interno 2 ponti H. É preferibile usare L298 perché può erogare fino a 2 ampere a 2 motori e arriva a tensioni massime di 46V.
In commercio si possono reperire dei moduli come quello riportato qui sotto, tratto dal sito
http://www.lombardoandrea.com/l298n-pilotare-un-motore-dc-con-arduino/



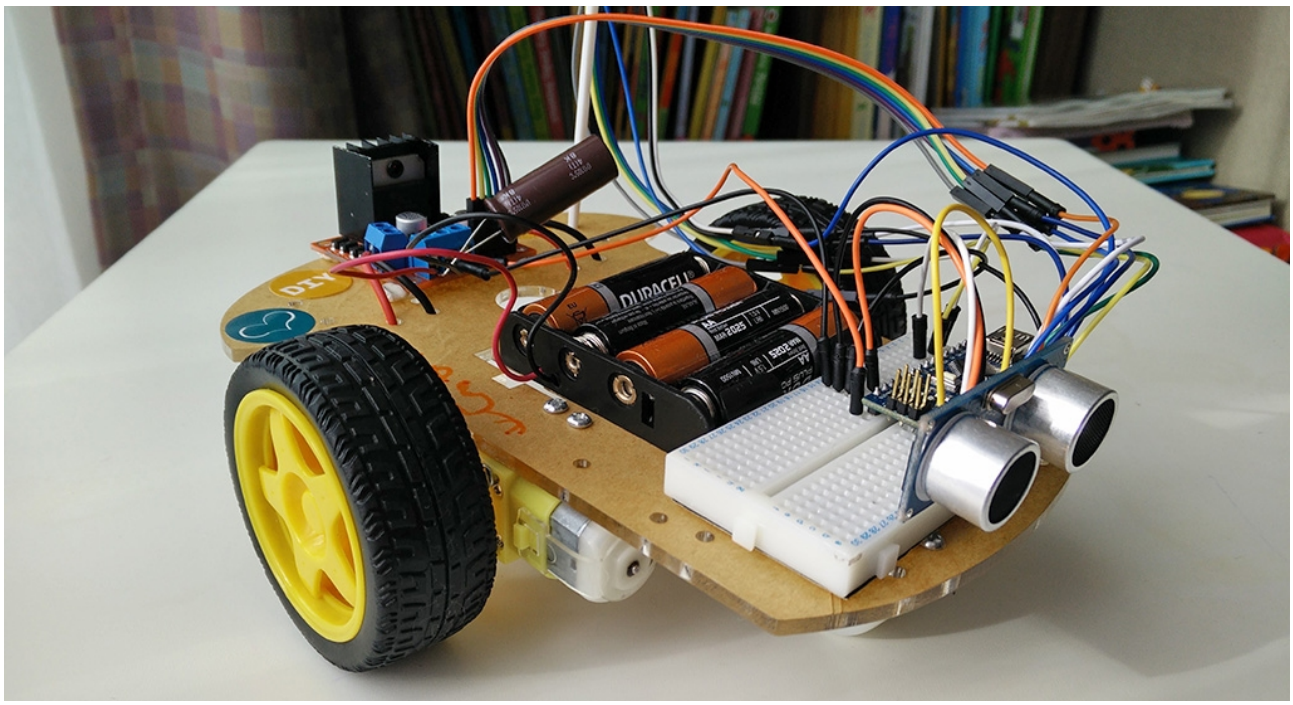


Schema circuitale del modulo L298

Occorre fare attenzione quando si usa questo modulo perché si deve fornire una tensione di almeno 7,5 V tra il terminale +12V e power GND. Uno stabilizzatore di tensione a tre terminali provvede a ricavare la tensione di +5V a partire da +12V. Quest'ultima (+12V) sarà la tensione che alimenterà i motori. Il terminale +5V power serve per alimentare la scheda arduino e dovrà essere collegato al piedino +5V di arduino. Anche la massa del modulo (Power GND) andrà collegata alla massa (GND) di arduino. La coppia di terminali Output A va collegata al primo motore e la coppia Output B al secondo motore. Poi, sono disponibili i terminali IN1, IN2 e A Enable per pilotare il primo motore (togliere il ponticello da A Enable) e IN3, IN4 e B Enable per il secondo motore (togliere il ponticello da B Enable).
Prima di collegare la scheda arduino al PC per trasferire il programma è necessario staccare la tensione dal terminale +12V Power, **altrimenti si può danneggiare arduino**.

Utilizzando questo modulo ed un paio di economici chassis con due motori in corrente continua dotati di riduttore, gli studenti hanno realizzato due piccoli robot. Il primo, grazie ad un sensore di distanza ad ultrasuoni, si muoveva autonomamente cercando di evitare gli ostacoli; il secondo era invece comandato da uno smartphone via bluetooth.

In rete si trovano molti esempi di realizzazioni di questo tipo, ne riporto uno che mi sembra interessante: http://www.webondevices.com/arduino-robot-car-obstacle-avoidance/.
Nella foto qui sotto si può vedere lo chassis impiegato che è reperibile anche su Amazon, ebay, ecc… ad un prezzo intorno ai 16€, completo di motori e ruote. I nostri esemplari sono alimentati in modo diverso da quello della foto: al posto delle 4 stilo AA abbiamo utilizzato due batterie 18650 in serie (3,8V, 2200mAh ciascuna).



Nel robot della foto viene impiegato un arduino nano, mentre noi abbiamo optato per una scheda arduino Uno. É opportuno notare che per far avanzare il robot i due motori devono ruotare in senso opposto l'uno rispetto all'altro. Per farlo ruotare a destra o a sinistra è sufficiente cambiare il verso di rotazione di uno dei due motori per un tempo che dev'essere determinato sperimentalmente in base alla velocità.

Ed ecco il programma che gestisce il robot evita ostacoli:

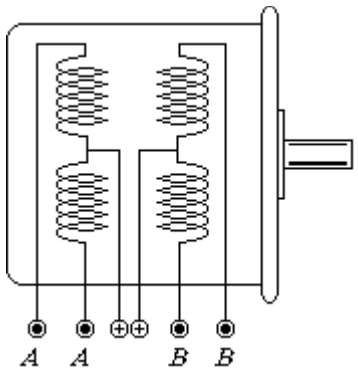
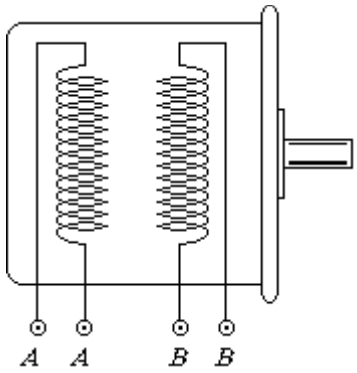
```
// Collegamenti dei piedini di arduino al modulo L298

int enable1 = 10; //piedino di attivazione motore 1
int enable2 = 9;  //piedino di attivazione motore 2
int in1 = 2;      //1°piedino ponte H motore 1
int in2 = 3;      //2°piedino ponte H motore 1
int in3 = 4;      //1°piedino ponte H motore 2
int in4 = 5;      //2°piedino ponte H motore 2

// Collegamenti dei piedini di arduino al modulo ultrasuoni HC-SR04

int trigger = 11; //piedino per inviare ultrasuoni
int echo = 12;    //piedino per ricevere distanza

void setup(){
// Modulo motori L298
  pinMode(enable1, OUTPUT);
  pinMode(enable2, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
// Sensore ultrasuoni
  pinMode(trigger,OUTPUT);
  pinMode(echo, INPUT);
// Inizialmente, robot fermo
```

```
  digitalWrite(enable1, LOW);
  digitalWrite(enable2, LOW);
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}

void loop(){
  digitalWrite(enable1, HIGH); //motore 1 alla massima velocità
//alternativamente: analogWrite(enable1, 128) dimezza velocità
  digitalWrite(enable2, HIGH); //motore 2 alla massima velocità
//alternativamente: analogWrite(enable2, 128) dimezza velocità

//Calcolo distanza ostacoli
  digitalWrite(trigger, LOW); //Invio impulso 10us ultrasuoni
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger, LOW);
  long durata = pulseIn(echo, HIGH); //lettura durata impulso ultras.
  long distanza = 0.034*durata/2; //calcolo distanza ostacolo
  if (distanza < 30) {     // se distanza ostacolo < 30 cm ruota
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
    delay(300);            // tempo impiegato per ruotare
  }
  else {                   // altrimenti vai avanti
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
  }
} //fine loop()
```

Arduino e i motori passo-passo

I motori passo-passo sono utilizzati in ambito industriale dove c'è necessità di ottenere un posizionamento molto preciso senza necessità di retroazione. Sono impiegati nei masterizzatori per spostare la testina con il laser, nelle stampanti a getto d'inchiostro e laser, nelle stampanti 3D, nelle macchine a controllo numerico (fresatrici, torni,…), negli scanner, ecc…
Hanno anche degli inconvenienti: rapporto elevato peso/potenza resa, bassa velocità di rotazione e correnti più elevate di funzionamento rispetto ai motori in corrente continua o ai motori brushless.
Si suddividono in due categorie: unipolari e bipolari. Gli unipolari sono facilmente distinguibili dai bipolari perché hanno 5 o 6 terminali rispetto ai 4 terminali dei bipolari.
I bipolari sono più diffusi perché più semplici da costruire e perché sono in grado di fornire una coppia maggiore. Per quanto concerne il principio di funzionamento vi rimando al sito web http://www.logicaprogrammabile.it/motore-passo-passo-bipolare-driver-l298n/ . Risultano particolarmente semplici anche le spiegazioni presenti sul sito www.adrirobot.it/elettronica/stepper/stepper_motor_theory.htm da cui ho tratto alcune delle immagini riportate qui di seguito.
Le grandezze che li caratterizzano sono: il numero di passi per giro, che ne determina la risoluzione (un motore con 200 passi per giro ruota il proprio albero di 360°/200 = 1,8° per ogni impulso applicato), la corrente massima degli avvolgimenti e la loro tensione massima applicabile.

I motori bipolari possiedono due avvolgimenti separati che devono essere comandati tramite 2 ponti H. Per interfacciarli con arduino torna utile il modulo L298 visto in precedenza. Per far ruotare l'albero è necessario fornire ai due avvolgimenti una sequenza precisa di impulsi (4 diverse combinazioni alto – basso alle due coppie di terminali); cambiando la sequenza si può modificare il verso di rotazione. Per variare la velocità di rotazione si modifica il ritardo tra un passo ed il successivo della sequenza, con la particolarità che per portarli alla velocità di regime è necessario farli partire piano e poi accelerare gradualmente (viceversa per rallentarli).

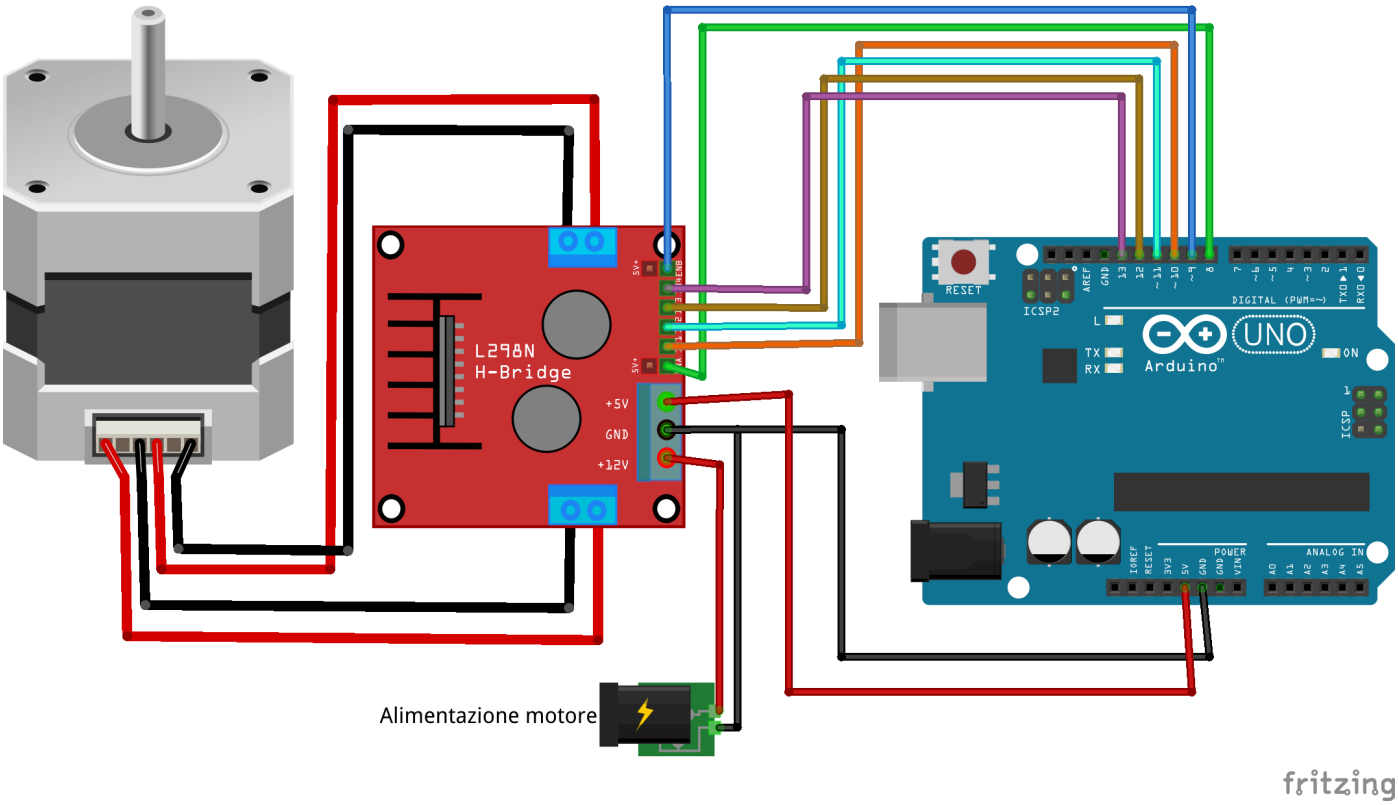
	
Motore passo – passo unipolare	Motore passo – passo bipolare

Motore passo – passo bipolare

Vediamo allora come possiamo comandarli tramite arduino ed un modulo L298.

Nel primo esempio si comandano direttamente le uscite che forniscono corrente ai due avvolgimenti (A e B) del motore passo – passo. Nel codice sono state inserite 5 funzioni ausiliarie: le prime 4 fanno ruotare di un passo l'alberino del motore; la quinta toglie corrente agli avvolgimenti, fermandolo. Gli ingressi di enable servono per attivare o disattivare le uscite e potrebbero essere collegati ad un'unica uscita digitale di arduino, dato che vengono sempre impostati allo stesso livello logico.
La funzione loop( ) fa ruotare l'alberino di 200 passi in un senso di rotazione e altri 200 nel senso opposto; poi, dopo 2 secondi di pausa, ricomincia. Notate la sequenza con cui devono essere attivate le uscite per far girare l'alberino in un senso e nel senso opposto.

Il modulo L298 è alimentato a 12V (tensione del motore stepper) e dal suo stabilizzatore interno si ricavano i +5V che alimentano la scheda arduino. Sono stati rimossi i ponticelli sugli ingressi di enable ed è stato lasciato il ponticello che attiva l'uscita dei 5V dal modulo L298.

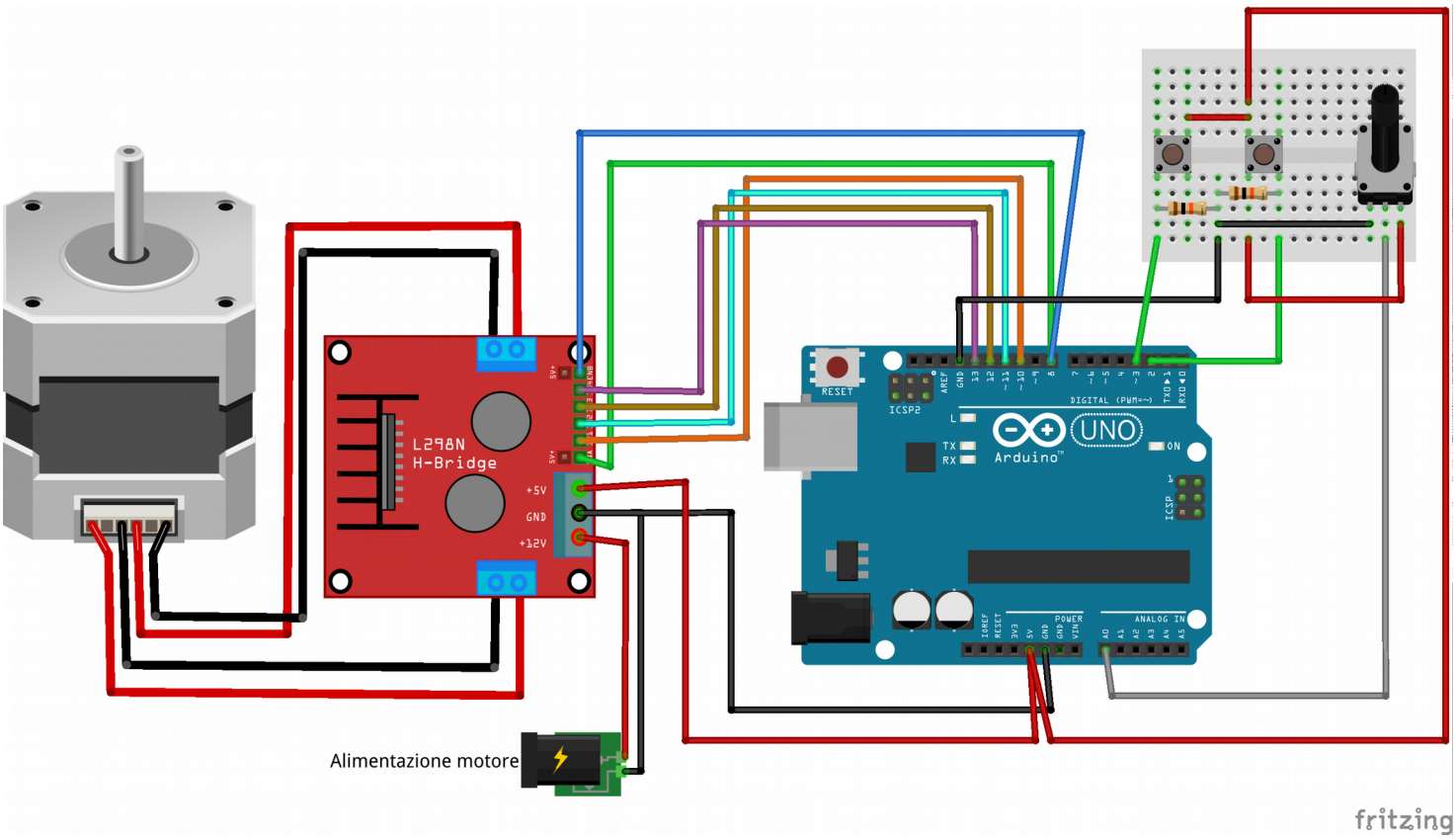



```
/* Programma dimostrativo per il comando di un motore passo-passo bipolare.
* I due avvolgimenti (A e B) sono collegati alle uscite del modulo L298.
* Il programma fa ruotare di 200 passi in un senso e di 200 passi nel senso
* opposto l'albero del motore. Le uscite 8 e 9 di arduino comandano
* rispettivamente enable A e enable B. Le uscite 10 e 11 di arduino
* pilotano, tramite L298, l'avvolgimento A. Le uscite 12 e 13 di arduino
* pilotano l'avvolgimento B.
*/

int inA1 = 10; // input 1 del motore stepper
int inA2 = 11; // input 2 del motore stepper
int inB1 = 12; // input 3 del motore stepper
int inB2 = 13; // input 4 del motore stepper
int enable1 = 8;
int enable2 = 9;

int stepDelay = 50; // Ritardo tra i singoli passi in millisecondi

void setup() {
  pinMode(inA1, OUTPUT); //avvolgimento A
  pinMode(inA2, OUTPUT);
  pinMode(inB1, OUTPUT); //avvolgimento B
  pinMode(inB2, OUTPUT);
  pinMode(enable1, OUTPUT);
  pinMode(enable2, OUTPUT);
}
```

```
void step1() {
  digitalWrite(inA1, LOW);
  digitalWrite(inA2, HIGH);
  digitalWrite(inB1, HIGH);
  digitalWrite(inB2, LOW);
  delay(stepDelay);
}
void step2() {
  digitalWrite(inA1, LOW);
  digitalWrite(inA2, HIGH);
  digitalWrite(inB1, LOW);
  digitalWrite(inB2, HIGH);
  delay(stepDelay);
}
void step3() {
  digitalWrite(inA1, HIGH);
  digitalWrite(inA2, LOW);
  digitalWrite(inB1, LOW);
  digitalWrite(inB2, HIGH);
  delay(stepDelay);
}
void step4() {
  digitalWrite(inA1, HIGH);
  digitalWrite(inA2, LOW);
  digitalWrite(inB1, HIGH);
  digitalWrite(inB2, LOW);
  delay(stepDelay);
}
void stopMotor() {                //tutte le fasi a 0 volt
  digitalWrite(inA1, LOW);
  digitalWrite(inA2, LOW);
  digitalWrite(inB1, LOW);
  digitalWrite(inB2, LOW);
}

// Ripeti per sempre:
void loop() {
  digitalWrite(enable1,HIGH); //abilita uscite
  digitalWrite(enable2,HIGH);
  for (int i=0; i<=50; i++){ //ruota di i*4 passi in avanti
    step1();
    step2();
    step3();
    step4();
  }
  stopMotor();                    // ferma tutto
  digitalWrite(enable1,LOW);
  digitalWrite(enable2,LOW);
  delay(2000);

  digitalWrite(enable1,HIGH); //abilita uscite
  digitalWrite(enable2,HIGH);
  for (int i=0; i<=50; i++){ //ruota di i*4 passi all'indietro
    step3();
    step2();
    step1();
    step4();
  }
```

```
  stopMotor();                    // ferma tutto
  digitalWrite(enable1,LOW); //disattiva le uscite
  digitalWrite(enable2,LOW);
  delay(2000);                    //attendi 2 secondi e ricomincia
}
```

L'esempio successivo utilizza la libreria *stepper* già inclusa nell'ambiente di sviluppo di arduino. Questa semplifica di molto la gestione del motore ed è in grado di comandare sia gli unipolari che i bipolari. La libreria possiede funzioni che permettono di impostare la velocità di rotazione in giri al minuto e il numero di passi in una direzione ed in quella opposta.
Il programma prevede l'utilizzo di due pulsanti che permettono di scegliere il senso di rotazione (orario, antiorario) e di un potenziometro mediante il quale si imposta la velocità. Risulta sempre necessario interporre un modulo di potenza L298 tra la scheda arduino ed il motore bipolare. I due ingressi di enable del modulo L298 sono collegati entrambi al piedino 8 di arduino, mentre gli avvolgimenti del motore sono pilotati come nell'esempio precedente. I pulsanti sono attivi alti e vengono collegati ai piedini 2 e 3 di arduino. Qui sotto viene riportato lo schema di cablaggio disegnato con Fritzing.



Segue il programma:

```
/* Regolazione velocità con potenziometro da 10K tra Vcc, GND e A0
 * Motore bipolare comandato da integrato L298
 */
#include <Stepper.h>   //Libreria stepper di arduino

int orario = 2;        //pulsante attivo alto, senso orario
int antiorario = 3;    //pulsante attivo alto, senso antiorario
int passi = 200;       //numero di passi per giro del motore
```

```
int enable = 8;             //abilita uscite ponti H al motore
Stepper motore(passi, 10,11,12,13); //avvolgimento A (10,11), B (12,13)

void setup() {
  pinMode(orario,INPUT);
  pinMode(antiorario,INPUT);
  pinMode(enable,OUTPUT);
}

void loop() {
      int Speed = analogRead(A0); //Leggo velocità dal potenziometro
      int RPM = map(Speed, 0, 1023, 0, 100); //Converto intervallo 0-1023 in
                                             //0-100
      int avanti = digitalRead(orario);      //Leggo pulsanti
      int indietro = digitalRead(antiorario);
      digitalWrite(enable,LOW);              //Disabilito uscite
      if(avanti == 1 && indietro == 0 && RPM > 5){ // avanti
            digitalWrite(enable,HIGH);  // abilito motore
            motore.step(1);             // 1 step in avanti
            motore.setSpeed(RPM);       // alla velocità RPM
            delay(10);
      }
      if( indietro == 1 && avanti == 0  && RPM > 5){ // indietro
            digitalWrite(enable,HIGH);  // abilito motore
            motor.step(-1);             // 1 step all'indietro
            motor.setSpeed(RPM);        // alla velocità RPM
            delay(10);
      }
      delay(5);
}
```

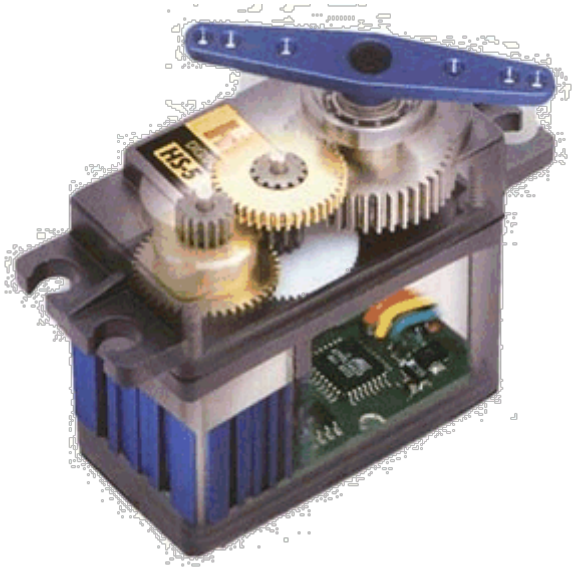
Dal listato si può notare che, dopo aver incluso la libreria con **`#include <Stepper.h>`**, è necessario definire un oggetto di tipo *Stepper* che nell'esempio sopra viene chiamato *motore*. L'istruzione **`Stepper motore(passi, 10,11,12,13);`** contiene all'interno delle parentesi rispettivamente: il numero di passi per giro del motore (che bisogna conoscere) ed i piedini ai quali sono collegati gli avvolgimenti A e B.
Le funzioni che la libreria mette a disposizione sono: *motore.step(numero di passi da effettuare)*, dove il numero di passi è una costante o una variabile intera ed il cui segno specifica il senso di rotazione; *motore.setSpeed(giri al minuto)*, che consente di specificare la velocità di rotazione in giri al minuto. I delay sono stati aggiunti per permettere al motore di assestarsi sui nuovi valori.

In rete si trova una libreria più sofisticata che consente di far variare l'accelerazione del motore e supporta più motori contemporaneamente. La libreria è denominata *AccelStepper* e la documentazione si trova all'indirizzo web //www.airspayce.com/mikem/arduino/AccelStepper/

I servomotori

Un altro tipo di motori utilizzati spesso con arduino sono i servomotori (spesso abbreviati in servo). Questi vengono impiegati nei modelli radiocomandati e consentono di effettuare rotazioni dell'alberino di 180°. Perchè non si usano al loro posto i motori passo – passo? Perchè i servo sono molto più leggeri, assorbono meno corrente e basta un'unica uscita per comandarli. Con le uscite disponibili sulla scheda arduino Uno si possono comandare diversi motori di questo tipo, bisogna però avere l'accortezza di alimentarli mediante una sorgente esterna. Per contro, sono meno precisi.

Un ottimo tutorial che illustra il funzionamento dei servomotori è presente sul sito http://www.gandotech.net/servomotori-come-quando-usarli/ . Nelle prossime righe cercherò di sintetizzarne le informazioni essenziali.
Si interfacciano con 3 terminali: 2 di alimentazione ed uno di comando. Al loro interno c'è un circuito elettronico che elabora il segnale di controllo, attiva il motore che fa ruotare l'alberino e ne controlla la posizione mediante feedback. Il segnale di controllo è costituito da un treno di impulsi periodici con frequenza pari a 50 Hz (T = 20 ms). La durata degli impulsi a livello alto determina la posizione dell'alberino; essa può variare da 1 ms fino a 2ms. Con impulsi di durata pari a 1ms l'alberino si posiziona a 0°, mentre con 2 ms si posiziona a 180°. Occorre fornire sempre gli impulsi di comando ai servo, altrimenti l'alberino è libero di ruotare in funzione delle forze dovute al carico esterno applicato. Mai far ruotare il servo oltre i finecorsa, pena la rottura degli ingranaggi.



Struttura interna di un servomotore

In commercio si trovano servomotori per tutti i tipi di applicazioni, anche per uso industriale come quelli riportati nella fotografia seguente.

Servomotori industriali APCServo-ABB

I servo possono essere utilizzati per applicazioni nel campo della robotica, ad esempio per realizzare un bipede semovente o un braccio robotico; ma anche per azionare lo sterzo di una macchinina, un sensore ad ultrasuoni, una webcam. Chi realizza in casa il proprio presepe natalizio li può impiegare per dare movimento ai personaggi, ecc…

Un modello particolarmente interessante per uso didattico è SG90, piccolo, può essere alimentato direttamente dalla scheda arduino e costa poco (10 servo a 23€ su Amazon Prime). Non è particolarmente robusto dato che gli ingranaggi sono di plastica.

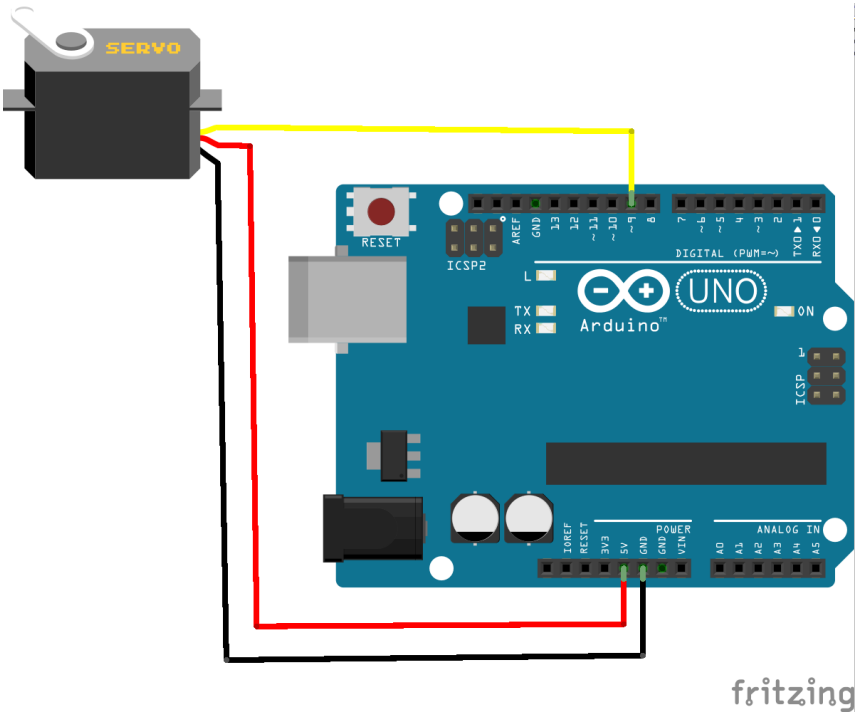
Le specifiche tecniche di questo motore sono le seguenti:

Tensione di lavoro: 4,8 ~ 6,0V
Velocità: 0,12secondi/60 gradi (a 4,8V) ~0,1s/60° (a 6V)
Coppia: 1,6 Kg/cm (a 4,8V)
Peso: 30g
Dimensioni: 21,5 x 11,8 x 22,7 mm

Il filo di colore rosso va collegato a +5V, il marrone a massa e il cavetto di colore arancio è il terminale di comando. Tenete presente che ogni costruttore ha un proprio standard per la colorazione dei terminali; è sempre meglio consultare il datasheet del componente.

Vediamo ora come si può interfacciare un servo di questo tipo con arduino Uno. A tale scopo è disponibile una libreria già inserita nell'ambiente di sviluppo, denominata *Servo*.
La libreria contiene le seguenti principali funzioni:
attach(piedino di arduino) : consente di specificare a quale piedino della scheda è collegato il terminale di controllo del servo.
detach( ) : funzione opposta alla precedente, il piedino non è più gestito dalla libreria.
write(angolo in gradi) : sposta l'alberino nella posizione specificata in gradi (da 0 a 180°).
writeMicroseconds(µs) : come la precedente, ma consente di specificare la durata dell'impulso a livello alto. In un servo, normalmente 1000 µs portano l'alberino in posizione 0° e 2000 µs lo portano in posizione 180°. Purtroppo esistono servo che non rispettano tali specifiche e questa funzione è stata sviluppata proprio per tali dispositivi.
read( ) : ritorna la posizione in gradi passata nell'ultima chiamata alla funzione *write( )*.

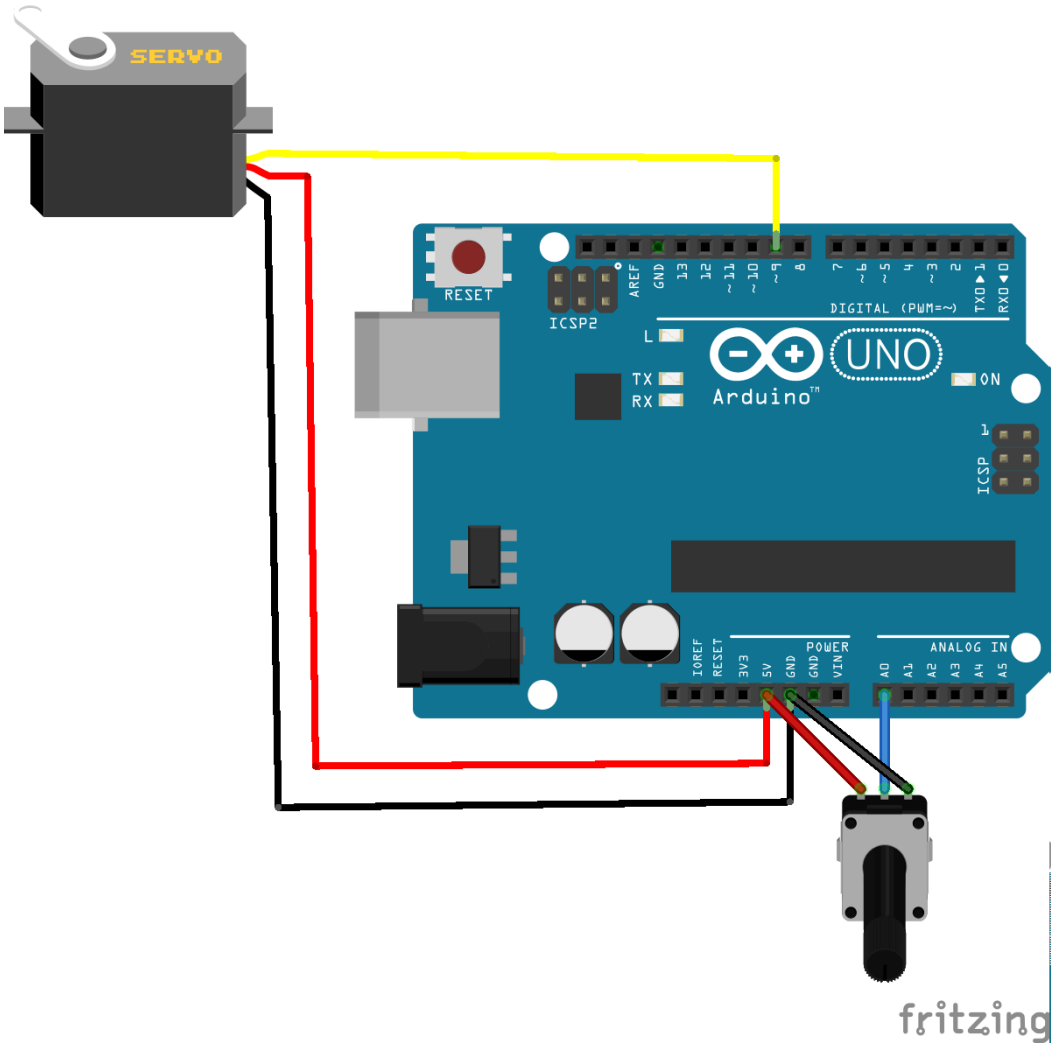
Vediamo un esempio in cui l'alberino del servo viene posizionato prima a 45°, poi a 90°, a 135° e di nuovo a 90°. Infine viene fatto ruotare da 0° a 180° e da 180° a 0° a passi di 1°, con un ritardo tra i passi di 50ms. I tre terminali del servo vanno collegati in questo modo: rosso a +5V, marrone a GND e arancio al piedino 9 di arduino.




```
#include <Servo.h> //Inclusione libreria gestione servo

int servoPin = 9; //Piedino di arduino a cui è collegato il terminale di
                  //controllo del servo
Servo mio_servo;  //mio_servo è la variabile di tipo Servo

int servoAngle = 0;   //posizione iniziale del servo in gradi

void setup()
{
  mio_servo.attach(servoPin); //associa pin 9 al mio_servo
}

void loop()
{
//controlla posizione del servo

  mio_servo.write(45);   // porta alberino a 45 gradi
  delay(1000);           // aspetta 1 secondo
  mio_servo.write(90);  // porta alberino a 90 gradi(posizione centrale)
  delay(1000);           // aspetta 1 secondo
  mio_servo.write(135);  // porta alberino a 135 gradi
  delay(1000);           // aspetta 1 secondo
  mio_servo.write(90);   // riporta alberino a 90 gradi
  delay(1000);            // aspetta 1 secondo

//fine controllo posizione
```

```
 //Rotazioni di 180°

//cambiando il valore di delay si modifica la velocità di rotazione
  for(servoAngle = 0; servoAngle < 180; servoAngle++)
  {
    mio_servo.write(servoAngle);
    delay(50);
  }

  for(servoAngle = 180; servoAngle > 0; servoAngle--)
  {
    mio_servo.write(servoAngle);
    delay(10);
  }
} //fine loop()
```

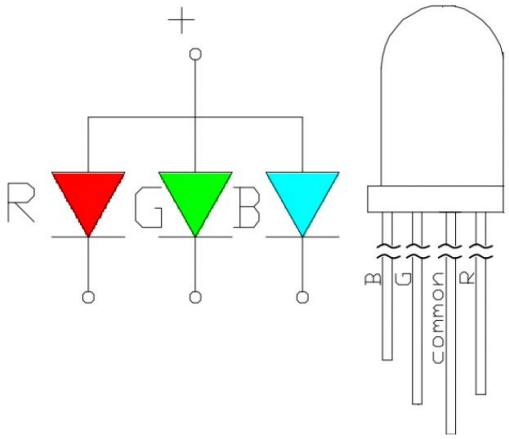
Nel secondo esempio (tratto dal sito ufficiale www.arduino.cc/en/Tutorial/Knob) viene usato un potenziometro da 10KΩ per posizionare l'alberino del servo. Il potenziometro viene collegato con i terminali estremi a GND e a +5V e con il cursore all'ingresso analogico A0. Dall'ingresso analogico viene letto un valore intero da 0 a 1023 che viene quindi ricondotto al range 0 ~ 180 e inviato al servo. Il terminale di controllo del servo è ancora collegato al piedino 9 della scheda arduino.



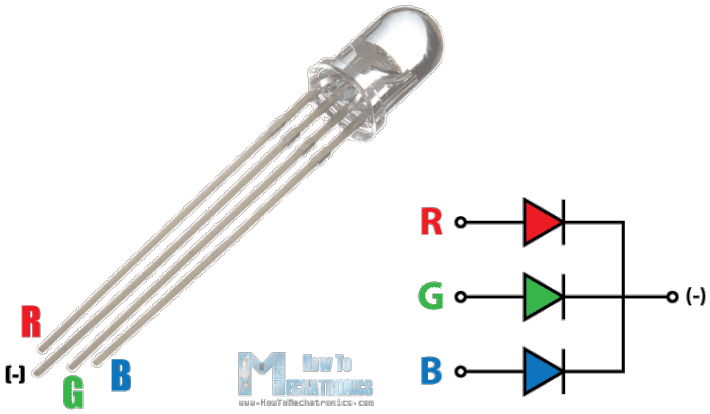
```
#include <Servo.h>

Servo mio_servo;

int potpin = A0;  //piedino a cui è collegato il cursore del potenz.
int val;         //variabile usata per leggere il valore dal pin A0

void setup() {
  mio_servo.attach(9);
}

void loop() {
  val = analogRead(potpin);  //leggo valore da A0 (tra 0 e 1023)
  val = map(val, 0, 1023, 0, 180); // riporto a 0 - 180
  mio_servo.write(val);      //imposto posizione del servo
  delay(15);                 //aspetto che il servo si posizioni
}
```

I LED RGB

Sono LED particolari, nel senso che in un unico contenitore da cui fuoriescono 4 terminali sono disposti, a distanza molto ravvicinata, 3 LED con i colori primari rosso, verde e blu. Facendo variare l'intensità di tali colori si ottengono moltissime variazioni possibili del colore risultante. Questi LED sono disponibili sia ad anodo che a catodo comune e sono molto diffuse anche le strisce di lunghezza pari a 5 metri con 300 LED, ritagliabili in spezzoni più corti.



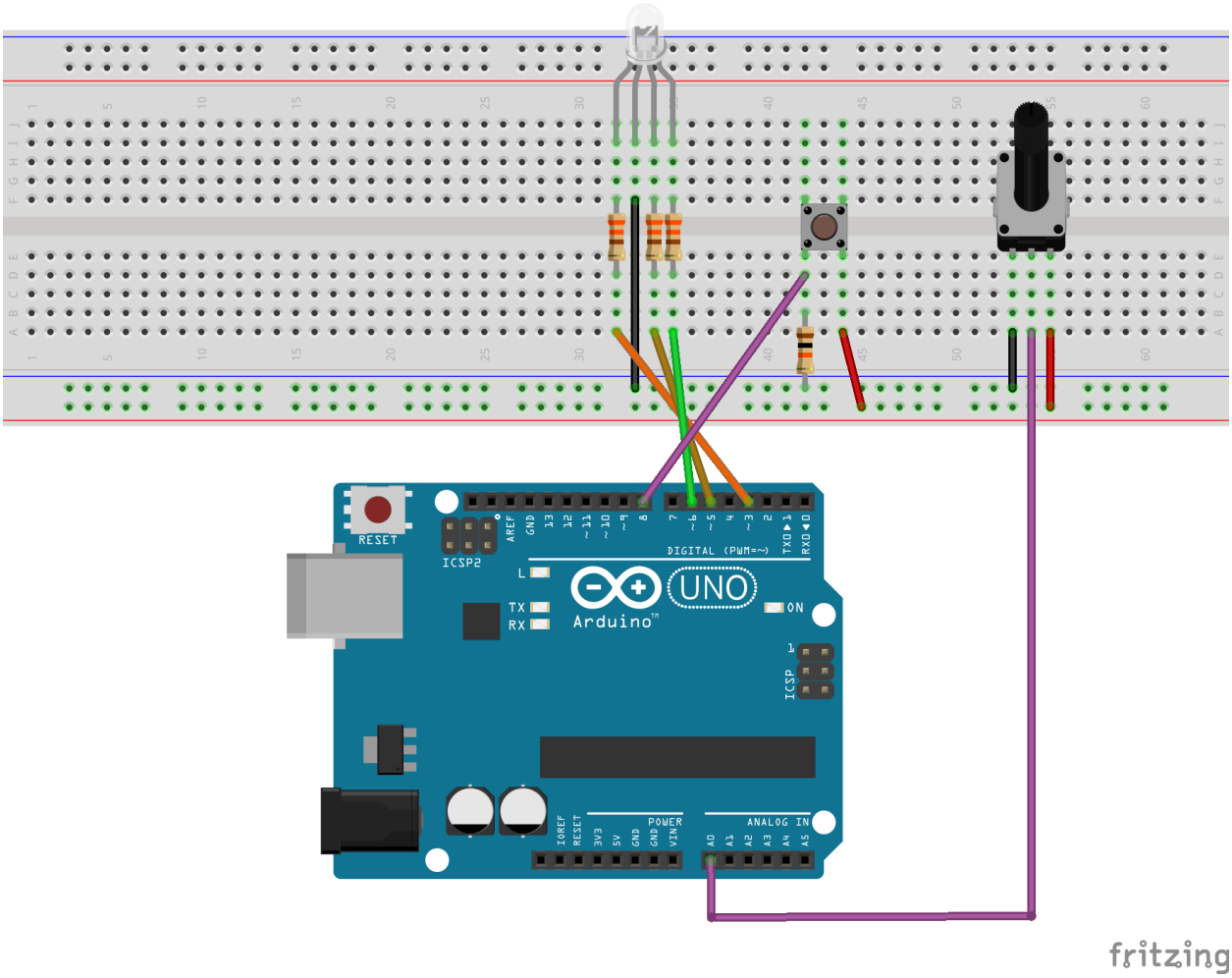
LED RGB anodo comune (sopra) e catodo comune (sotto)



A cosa possono servire? Quelli singoli vengono utilizzati come indicatori in cui il colore rappresenta, ad esempio, un valore di una grandezza fisica (temperatura, pressione, tensione, resistenza, ecc…). Le strisce possono essere impiegate per illuminare piccoli ambienti in modo suggestivo con luce che cambia colore in base ad un programma. Diciamo che le applicazioni dipendono dalla fantasia del progettista.
Una particolarità delle strisce di LED è che devono essere alimentate a 12V e sono ad anodo comune.
Questi componenti sono molto adatti ad essere pilotati in tecnica PWM: l'intensità di ciascun colore primario viene fatta dipendere in modo proporzionale al duty-cycle di un'onda rettangolare.

L'esempio che propongo consiste in un circuito con un potenziometro, un pulsante attivo alto e un LED RGB a catodo comune tutti collegati ad una scheda arduino Uno. Inizialmente il potenziometro permette di regolare l'intensità del rosso; premendo il pulsante si regola l'intensità del verde e premendolo ancora si regola l'intensità del blu.

Agli anodi del LED RGB sono collegate in serie delle resistenze di valore pari a 330Ω (in realtà il rosso ha una tensione di soglia inferiore agli altri due colori primari e richiederebbe un resistore di valore leggermente più alto, ma la differenza non si nota granchè). Il resistore in serie al pulsante è da 10 KΩ, così come il valore del potenziometro.



Segue il programma di gestione del LED RGB:

```
// Esempio analogWrite. LED RGB catodo comune

int  ROSSO = 3; //pin di arduino collegato all'anodo del rosso
int  VERDE = 5; //pin di arduino collegato all'anodo del verde
int  BLU = 6;   //pin di arduino collegato all'anodo del blu
int  POT = A0;  //pin di arduino collegato al cursore del potenziometro
int  PULSANTE = 8; //pin di arduino collegato al pulsante
int  ValRosso, ValBlu, ValVerde; //valori assegnati ai vari colori
int  val_pot = 0; //valore letto dal potenziometro
int  val_col = 0; //valore letto dal pulsante
int  old_val_col = 0; //necessario per anti-rimbalzo
int  color = 0; //scelta del colore primario con il pulsante

void setup() {  //definizione dei piedini di ingresso e di uscita
  pinMode(ROSSO, OUTPUT);
  pinMode(VERDE, OUTPUT);
  pinMode(BLU, OUTPUT);
  pinMode(PULSANTE, INPUT);
}

void loop() {
  // Regolazione RGB con PULSANTE e POT
  val_pot = analogRead(POT); //leggo potenziometro (0 ~ 1023)
  val_col = digitalRead(PULSANTE); //leggo pulsante
  if(val_col==HIGH && old_val_col==LOW){ //se pulsante premuto
     color++;         //incremento numero del colore da regolare
     delay(50);        //ritardo anti-rimbalzo
  }
  old_val_col = val_col; //aggiorno stato del pulsante

  if(color == 0) {
    ValRosso = val_pot/4; //riconduco intervallo a 0~255
    analogWrite(ROSSO, ValRosso); //e lo invio al rosso
  }
  else if(color == 1) { //pulsante premuto una volta
    ValVerde = val_pot/4;   //riconduco intervallo a 0~255
    analogWrite(VERDE, ValVerde); //e lo invio al verde
  }
  else if(color == 2) { //pulsante premuto due volte
    ValBlu = val_pot/4;     //riconduco intervallo a 0~255
    analogWrite(BLU, ValBlu);  //e lo invio al blu
  }
  if (color == 3)     //se pulsante premuto tre volte
    color = 0;        //riporto color a zero
} // fine loop()
```

Il programma successivo fa tutto da solo; all'inizio si vede una variazione dal verde al rosso, poi dal rosso al blu, quindi dal blu al verde e infine ricomincia daccapo. Il circuito è lo stesso del precedente, ma senza il pulsante e senza il potenziometro; rimangono soltanto il LED RGB, i tre resistori in serie e la scheda arduino Uno.
Vediamone il listato:

```
int  ROSSO = 3; //pin di arduino collegato all'anodo del rosso
int  VERDE = 5; //pin di arduino collegato all'anodo del verde
int  BLU = 6;   //pin di arduino collegato all'anodo del blu
int  ValRosso, ValBlu, ValVerde; //valori assegnati ai vari colori

void setup() {  //definizione dei piedini di ingresso e di uscita
  pinMode(ROSSO, OUTPUT);
  pinMode(VERDE, OUTPUT);
  pinMode(BLU, OUTPUT);
}

void loop() {
// variazione da verde a rosso

  ValRosso = 255;
  ValBlu = 0;
  ValVerde = 0;
  for(int i = 0 ; i < 255 ; i ++) {
    ValVerde += 1;
    ValRosso -= 1;
    analogWrite( VERDE, 255 - ValVerde ); //diminuisce verde
    analogWrite( ROSSO, 255 - ValRosso ); //aumenta rosso
    delay(50); //pausa di 50 ms
  }

  // variazione da rosso al blu

  ValRosso = 0;
  ValBlu = 255;
  ValVerde = 0;
  for(i = 0 ; i < 255 ; i ++) {
    ValRosso += 1;
    ValBlu -= 1;
    analogWrite( ROSSO, 255 - ValRosso );
    analogWrite( BLU, 255 - ValBlu );
    delay(50);
  }
  // variazione da blu al verde

  ValRosso = 0;
  ValBlu = 0;
  ValVerde = 255;
  for(i = 0 ; i < 255 ; i ++) {
    ValBlu += 1;
    ValVerde -= 1;
    analogWrite( BLU, 255 - ValBlu );
    analogWrite( VERDE, 255 - ValVerde );
    delay(50);
  }
  ValVerde = 0;
  analogWrite(VERDE, ValVerde);  // spegne RGB
}
```