Il servo da Modellismo

Nati per impiego modellistico, si trovano molto diffusi oggigiorno anche in applicazioni elettroniche, grazie alla loro versatilità e facilità di gestione in tutte quelle situazioni in cui è necessario eseguire un movimento meccanico di precisione. Dal costo contenuto e forniti in svariate grandezze, dipendenti essenzialmente dalla loro potenza, possono essere impiegati in tantissime applicazioni pratiche: una tra tutte, la movimentazione di piccole videocamere per le riprese a bordo di robot (pan/tilt). Per comprendere il funzionamento di un servo da modellismo, occorre osservarlo al suo interno: esso è costituito da un piccolo motore in corrente continua che, grazie ad un sistema di ingranaggi, fa ruotare un perno sul quale è calettato un piccolo potenziometro: la lettura del valore resistivo di questo potenziometro fornisce la posizione esatta del perno. Un circuito elettronico realizzato in tecnologia SMD provvede al controllo bidirezionale del motore ed al corretto posizionamento del perno in relazione al segnale elettrico di comando.

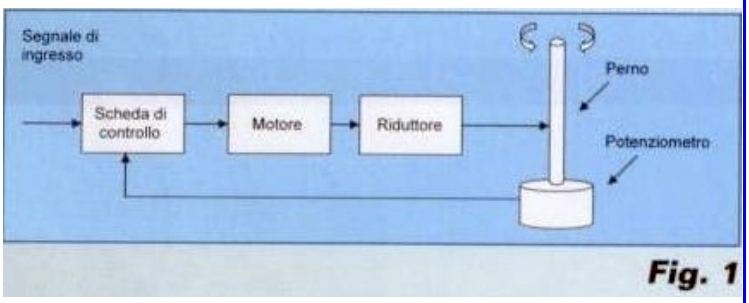


Fig. 1

Il tutto funziona secondo lo schema riportato nella **Fig. 1**, dal quale appare evidente che il posizionamento avviene confrontando il valore in tensione fornito dal potenziometro con quello ricavato dal segnale di ingresso e ruotando di conseguenza il motore sino a quando questi due valori non coincidono perfettamente.

colore	funzione
Nero o marrone	Negativo di alimentazione (GND)
rosso	Positivo di alimentazione (+Vcc)
Giallo o bianco	Segnale di comando (ingresso)

Tabella 1

In questo modo si ottiene un controllo di posizione molto veloce e preciso, comandabile con semplici segnali elettrici. Il cavetto di collegamento è composto da un filo di riferimento (GND), un filo per l'alimentazione (da 4,8 a 6 volt) ed un filo per il segnale di comando (**Tabella 1**) Occorre dire che non è prevista la rotazione continua (salvo casi particolari) del perno, ma solo di ±60 rispetto alla posizione iniziale, anche se è possibile espandere la rotazione sino a ±90. Il segnale di controllo è di tipo PWM (Pulse Wide Modulation) formato da impulsi ad onda rettangolare ripetuti ogni 20 ms, la cui "larghezza" permette di impostare la posizione del perno del servo. La posizione centrale si ottiene quando gli impulsi hanno una durata di 1.5 ms (**Fig. 2**). Questo tipo di segnale digitale si presta benissimo ad essere generato da una logica programmabile, quindi i servo possono essere comodamente gestiti dai microcontrollori.

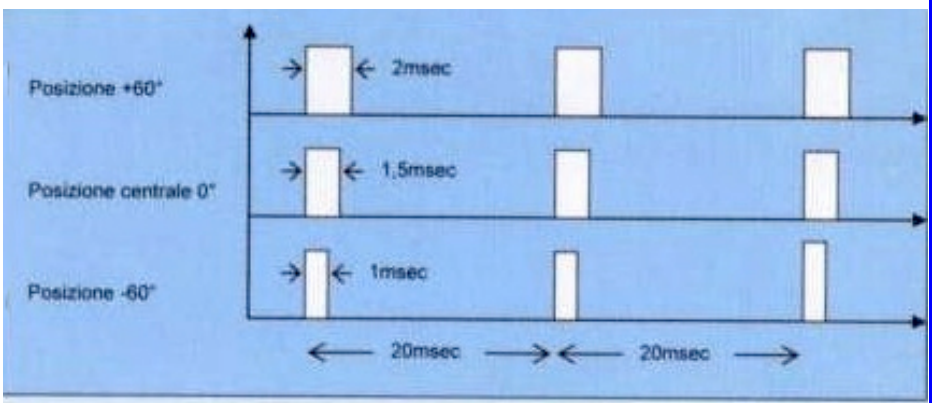


Fig. 2 - *Il segnale di comando.*

Istruzioni *Attach()*

Associa la variabile servo ad uno specifico pin. In Arduino 0016 e precedenti, sono supportati solo due servo collegati ai pin 9 and

10. Sintassi *servo.attach(pin)*, *servo.attach(pin, min,max).*

Parametri servo: variabile di tipo Servo. Pin: numero del pin hardware utilizzato.

Min (opzionale): durata minima dell'impulso, in microsecondi, corrispondente al minimo grado di rotazione (0 gradi) del servo (il valore predefinito è 544).

Max (opzionale): durata massima dell'impulso, in microsecondi, corrispondente alla massima rotazione (180 gradi) del servo (il valore predefinito è 2400).

Istruzioni *Attached()*

Verifica l'associazione tra la variabile servo ed il pin.

Sintassi: *servo.attached().*

Parametri servo: variable di tipo Servo.

Ritorno: vero se il servo è associato al pin; falso in caso contrario.

Istruzioni Detach()

Dissocia la variabile servo al pin specificato. Se tutte le variabili servo non sono associate, i pin 9 e 10 possono essere usati come uscite PWM con l'istruzione *analogWrite()*.

Sintassi: *servo.detach()*.

Parametrri: servo= variabile di tipo servo.

Istruzione Read()

Legge l'attuale posizione del servo corrispondente all'ultima posizione passata con l'istruzione write().

Sintassi: servo.read().

Parametri servo: variabile di tipo servo.

Ritorno: l'angolo del servo da 0 a 180 gradi.

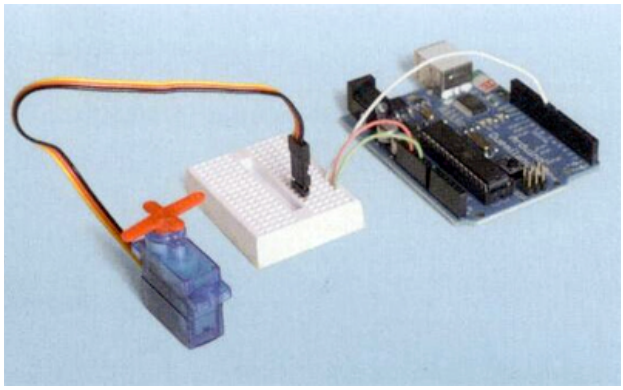
Istruzione Write()

Invia il valore in gradi relativo alla posizione del perno del servo. Un valore 0 corrisponde alla massima rotazione a sinistra, mentre 180 equivale alla massima rotazione a destra; il valore 90 indica la posizione centrale. L'esatta corrispondenza tra valore in gradi inviato e l'effettiva rotazione del servo viene specificata dai valori Max e Min nella dichiarazione dell'istruzione *attach()*; tali valori devono essere ricavati mediante prove pratiche, in quanto possono anche variare da servo a servo.

Sintassi: *servo.write(angle)*.

Parametri servo: variabile di tipo servo.

Angle: valore corrispondente alla rotazione in gradi.



Istruzione Writemicroseconds()

Imposta la posizione del servo come valore relativo alla durata dell'impulso espressa in microsecondi. Normalmente un valore 1000 corrisponde alla massima rotazione a sinistra, 2000 alla massima rotazione a destra ed il valore 1500 corrisponde alla posizione centrale (neutro)

Sintassi writeMicroseconds(µS)
Parametri servo: variabile di tipo servo.
µS: valore in microsecondi relativo alla posizione del servo.

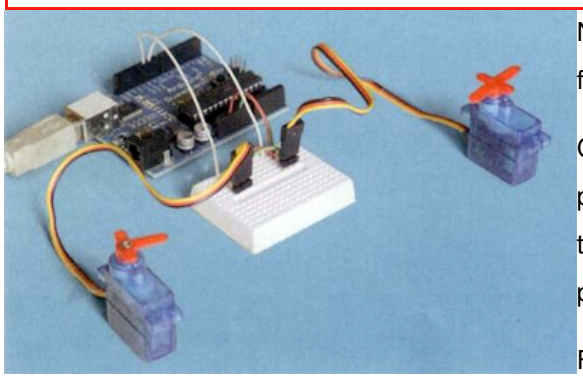
Installando il software Arduino-18, vi ritroverete con due esempi già pronti relativi all'utilizzo dei servo: il primo, denominato Knob, permette di posizionare il servo a seconda della posizione di trimmer cablati sulla scheda Arduino, mentre il secondo, denominato Sweep, permette di far girare l'alberino del servo alternativamente dalla posizione minima a quella massima. In entrambi gli sketch il servo è cablato connettendo il positivo al pin +5 V di Arduino, la massa al pin GND e l'ingresso di comando al pin 9. Solo per il primo esempio è necessario collegare anche un trimmer con i contatti esterni connessi uno a 5 V e l'altro a GND ed il centrale (cursore) al pin 0 di Arduino.

Listato 1

{codecitation class:"brush:cpp"}

```
Include <Servo.h>

Servo myservo;                    // crea un oggetto di  tipo servo con nome myservo
Int pos = 0;                  //  variabile contenente  11   valore della  posizione del   servo
void setup()

{

myservo.attach(9) ;               //  associa  l'oggetto myservo    al   pin 9
}

void loop()
{

for(pos= 0;  pos < 180; pos += 1)           // loop partendo da 0 fino a 180 gradi
{                                  //  a passi   di   un grado
myservo.wr1te(pos);               //  Imposta   la posizione del   servo
delay(15);                        // attende che il   servo raggiunga   la  posizione
}                          // dal   minimo al   massimo sono necessari   15msX180=2,7secondi
for(pos =  180;   pos>=1;   pos-=1)               //   loop da   180 fino a  zero gradi
{
myservo.write(pos);                //  imposta la posizione del  servo

delay(15);                        // attende che il  servo abbia raggiunto la posizione

}

}
```

{/codecitation}



Nel **Listato 1** riportiamo il codice relativo al secondo esempio e ne descriviamo il funzionamento, allo scopo di comprendere meglio l'utilizzo di questa libreria.

Come si vede da tale listato, la libreria semplifica notevolmente il lavoro di programmazione; le istruzioni chiave sono quelle che definiscono un oggetto di tipo servo *myservo=Servo* che successivamente sarà associato ad uno specifico pin *myservo.attach(9)*.

Fatto questo, per impostare la posizione del servo è sufficiente utilizzare il comando *myservo.write(pos)* con il parametro che può valere tra 0 e 180 (corrispondenti ad una rotazione tra 0° e 180°).

Possiamo ora realizzare uno sketch che ci permetta di impostare la posizione di un servo direttamente dal PC, inviando la posizione tramite il tool SerialMonitor.

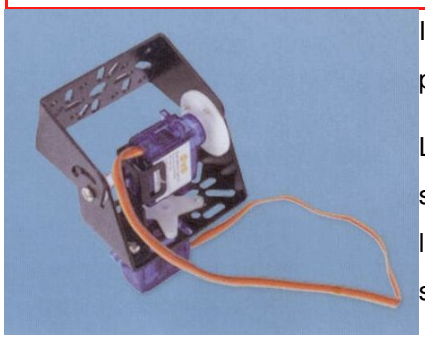
Il programma per questa funzione ne è descritto dal **Listato 2**.

Listato 2

{codecitation class:"brush:cpp"}

```
include <Servo.h>
Servo myservo: // crea un oggetto di tipo servo con nome myservo
void setup()
{
Serial.begin(9600);  // imposta comunicazione a 115200 baud
Serial.println("Pronto!");
}
void loop()

{
static int  v = 0;
if  ( Serial.available())

{
char ch = Serial.read();
switch(ch)

{
case  '0'... '9':
v =  (ch - '0')*20;     //   '0'=0°   '9'=180°
myservo.write(v);
break;
case  'd':
myservo.detach() ;
break;
case 'a':
myservo.attach(9);
break;

}

}

}
```

{/codecitation}

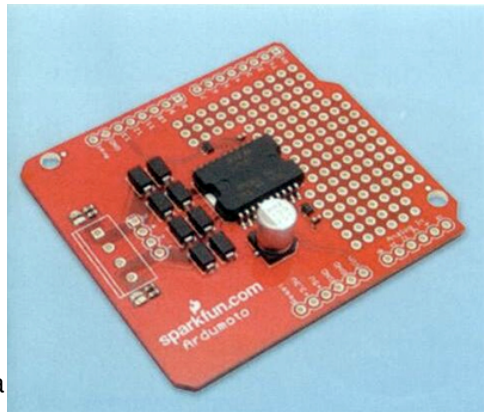


Il listato, peraltro molto semplice, prevede di inizializzare un servo "myservo" ed associarlo al pin 9; viene quindi abilitata la comunicazione seriale con il comando *Serial.begin(9600)*.

La riga di codice *char ch = Serial.read()* permette di attendere l'arrivo di un carattere e di salvarlo nella variabile *ch* che successivamente, con l'istruzione *case*, viene usata per eseguire le istruzioni di associazione Servo ('a'), dissociazione servo ('a') oppure posizionamento dal servo (numeri da '0' a '9').

Per provare questo programma lasciate il servo connesso al pin 9 ed avviate *Serial Monitor* dal menu *Tools*; assicuratevi di aver impostato una velocità di comunicazione di 9.600 baud. Aspettate che si evidenzi la scritta "pronto!" inviata dalla scheda Arduino

appena terminata la programmazione, quindi spedite il carattere "a" per abilitare il servo; successivamente digitato un numero tra 0 e 9 ed inviatelo (pulsante *send*). Il servo viene posizionato tra 0° e 180° in passi di 20°.



La gestione di due servocomandi e altrettanto facile, essendo sufficiente dichiarare due oggetti di tipo servo ad esempio *Servo_1* e *Servo_2*, associarli alle uscite 9 e 10 e comandarli con le istruzioni *Servo_1.write(pos1)* e *Servo_2.write(pos2)*. La gestione di un sistema pan/tilt per il puntamento di una videocamera risulta molto semplice; un esempio di sketch lo troverete assieme ai sorgenti di questa puntata, con il nome di *motor_2*. Per la parte meccanica consigliamo di utilizzare due servocomandi (codice SERVO206) in abbinamento al pan/tilt bracket kit di codice PANTILTKIT, il tutto reperibile presso la ditta Futura Elettronica (www.futurashop.it).

Arduino	Motorshield
Pin 12	Controllo direzione motore A
Pin 10	Segnale PWM per controllo velocità motore A
Pin 13	Controllo direzione motore B
Pin 11	Segnale PWM per controllo velocità motore B
	Morsetti 1 e 2 collegamento motore A
	Morsetti 3 e 4 collegamento motore B

Tabella 2

Vediamo ora come sia possibile gestire dei motori a spazzole in corrente continua, di quelli, per intenderci, che normalmente vengono usati nei giocattoli e che spesso troviamo in molte applicazioni di robotica. Per questo è disponibile uno specifico hardware denominato ArduMoto (la versione V12 nel nostrocaso) disponibile presso la Futura Elettronica (il codice del prodotto è 7300-MOTOR-SHIELD).

Questa scheda viene fornita già montata con componenti in SMD, e basata sul chip L298 e permette di controllare direzione e velocità di 2 motori DC con una corrente massima di 2 ampere ciascuno. Alimentata direttamente dalla linea Vin di Arduino Duemilanove, Arduino uno o Seeeduino, ogni sua uscita dispone di un LED blu e uno giallo per indicare la direzione di rotazione del motore. Tutte le linee di uscita del chip L298 sono protette da un diodo.

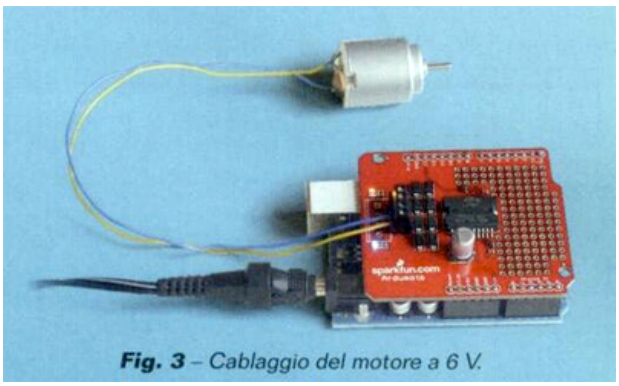


***Fig. 3** – Cablaggio del motore a 6 V.*

Con questa scheda è possibile gestire ciascun motore in entrambi i sensi di marcia, mentre la velocità di rotazione viene regolata con la tecnica del PWM. Ricordiamo in breve che, con il termine PWM, si intende una tecnica di modulazione in cui il segnale in uscita, in questo caso la tensione al motore, viene applicata e poi tolta ad intervalli regolari e molto velocemente. Maggiore è il tempo in cui è presente la tensione in uscita, rispetto al tempo in cui è assente, più il motore girerà velocemente e v iceversa.

Rapporto di riduzione	1:120
Giri a vuoto(3V)	100 RPM
Giri a vuoto(6V)	200 RPM
Corrente a vuoto(3V)	60 mA
Corrente a vuoto(6V)	71 mA
Corrente a rotore bloccato(3V)	260 mA
Corrente a rotore bloccato(6V)	470 mA
Coppia (3V)	1,2 kgcm
Coppia (6V)	1,92 kgcm
Dimensioni	55 x 48,3 x 23 mm
Peso	45 g

Tabella 3

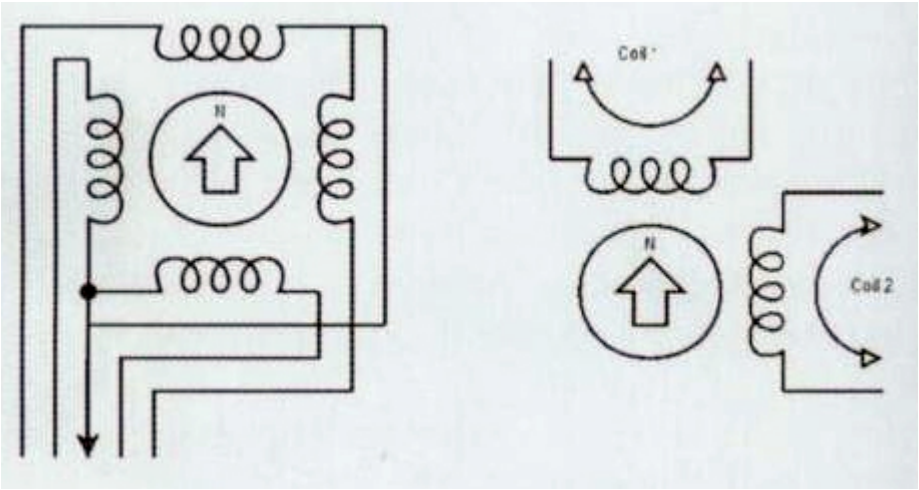
La ArduMoto si installa direttamente al di sopra della scheda Arduino, con un cablaggio predefinito e riepilogato nella **Tabella 2**.

Allo scopo è necessario che vi procuriate degli strip maschio passo 2,54 ed una morsettiera per contatti passo 2,54 oppure 3,5 mm per il collegamento dei motori; la loro saldatura non presenta alcuna difficoltà.

Per la scelta dei motori, da poter collegare, dovete considerare che l'alimentazione viene prelevata dalla linea Vin della scheda Arduino, che a sua volta coincide con l'alimentazione esterna applica al Plug di alimentazione e può avere un valore compreso tra 7 e 14 V. Per quanto riguarda l'assorbimento dei motori, ricordiamoche in un motore elettrico oltre alla corrente assorbita in funzionamento normale è importante considerare anche la corrente massima richiesta in fase di avvio (corrente di spunto) che può arrivare anche a 3 o 4 volte la corrente nominale.

I motori passo-passo

I motori passo-passo fanno sempre parte della grande famiglia dei motori in corrente continua, però, a differenza di quelli a spazzole, i loro avvolgimenti sono esterni (statore) mentre i magneti sono interni (rotore). Quindi non necessitano di spazzole per portare corrente al rotore. Questo implica alcuni vantaggi tra cui l'assenza di usura, minimi disturbi (proprio per la mancanza dei contatti sulle spazzole), possibilità di bloccare il rotore in una specifica posizione; per contro, non è sufficiente collegare il motore ad un alimentatore ma bisogna comandarlo con un apposito driver.



Il driver deve essere in grado di fornire corrente alternativamente con una sequenza prestabilita su ogni fase del motore (di solito quattro) alla quale corrisponde la rotazione dell'albero motore di uno step (da cui, appunto, il nome di "step-motor"). A seconda del tipo di motore, si possono avere 64, 100, 200, 360 o più step con il vantaggio che è possibile mantenere il motore fermo in un determinato step semplicernente fermando la sequenza di alimentazione ma continuando a mantenere alimentato il motore (con una corrente impostabile sul driver di comando). Viste le esigue potenze realizzabili in pratica, questi motori vengono usati per movimenti di precisione in stampanti o bracci robotizzati per lo spostamento di piccoli oggetti. La gestione da parte di un microcontrollore risulta assai agevole, dovendo dialogare con un driver "intelligente" al quale è sufficiente indicare la direzione di rotazione e fornire un impulso per ogni step che si vuole ottenere. Per come funzionano, i motori passo-passo possono ruotare il proprio perno anche solo per porzioni di giro, con estrema precisione. I motori passo-passo si dividono in due categorie (unipolari e bipolari) a seconda della configurazione degli avvolgimenti interni. In quelli unipolari sono presenti quattro avvolgimenti e la corrente li percorre in un solo senso; il cablaggio consiste in quattro fili, uno per ciascun avvolgimento, più un filo comune (5 fili totali) oppure quattro fili, uno per ciascun avvolgimento più due fili ciascuno il comune di due avvolgimenti (6 fili totali). In quelli bipolari sono presenti solo due avvolgimenti ma la corrente può andare in entrambi i sensi; il cablaggio consiste in due fili per ciascun avvolgimento, per un totale di 4 fili.

Se prevedete un utilizzo intenso dei motori con continue accelerazioni e frenate, è importante che anche questa corrente non superi il valore consigliato dei 2 A. Per tale ragione sceglierete dei motori con corrente nominale non superiore a 500 mA. Per i primi esperimenti abbiamo utilizzato un piccolo motore da 6 V (corrente massima di 100 mA) ricavato da un'automobilina giocattolo dismessa, cablato come visibile nella **Fig. 3**. Per l'alimentazione abbiamo usato un alimentatore universale non stabilizzato da 5 W, impostato per tornire una tensione di 6 V, sufficienti per alimentare sia la scheda ArduMoto che Arduino.

Come primo sketch facciamo in modo da attivare entrambi i motori con la seguente sequenza che si ripete all'infinito: motore A e B avanti a mezza velocità, motore A e B avanti a piena velocità, motore A e B fermi, motore A e B indietro a mezza velocità.

Una possibile ed interessante applicazione è il controlio delle ruote di una piattaforma robotica come quella commercializzala dalla ditta Futura Elettronica (codice 7300-2WDP-MA) perfettamente compatibile con le schede Arduino. L'assemblaggio di questa piattaforma robotica è davvero agevole, in quanto essa è già fornita di tutte le viti necessarie al fissaggio.

Per i più tecnici forniamo anche le caratteristiche elettriche dei motori, che trovate nella **Tabella 3**.

Il parametro "a rotore bloccalo" si riferisce al valere della corrente assorbita dal motore quando la ruota e ferma, ovvero quando al robot da fermo viene data tensione per farlo partire o quando lo stesso robot dovesse andare a sbattere contro una parete rimanendovi bloccato. E' in pratica la massima corrente assorbita dai motori ed anche il punto in cui i motori sono maggiormente sollecitali, ma come potete vedere, anche alla

massima tensione di 6 volt, siamo ampiamente al disotto del valore dei 2 A sopportato dal driver di ArduMotor. Utilizzando la piattaforma robotizzata a quattro ruote, i motori di ogni lato potranno essere connessi in parallelo funzionando all'unisono come in un carro armato, rispettando ancora una volta il limite massimo di 2A.

Per l'alimentazione ci affidiamo a 4 batterie ricaricabili da 1,2 V, le quali, completamente cariche, forniranno 1,5x4=6 volt (il massimo consentito dai motori). Tuttavia non potremo sfruttarle al massimo, perchè già ad 1,1 volt per cella la tensione totale sarà di soli 4.4 V. Inserite le batterie in un portabatterie e realizzate un cavetto di alimentazione che abbia da una parte il plug per la scheda Arduino e dall'altra la clip per il portabatterie. Sulla confione della base robotica troverete anche tutte le viti per il fissaggio, un plug maschio ed un interruttore, utile per realizzare un cablaggio più raffinato.

Abbiamo messo appunto uno sketch apposito denominato *motor_ 4*, che permette di comandare i motori in direzione e velocità tramite comandi dal PC, ovviamente con il cavo di programmazione connesso.

Anche se in modo limitalo, è comunque possibile testare le Funzionalità della piattaforma, se non vi soddisfa il senso di rotazione è comunque possibile invertire i fili del motore interessato.

Ci rimane, adesso, un'ultima parte riguardante i motori passo-passo, non essendo disponibile una vera e propria motor shield specifica per questi motori abbiamo optato per l'utilizzo di un driver siglato EasyDriver, fornitoci dalla ditta Futura (codice 7300-EASYDRIVER) e reperibile sul sito www.futurashop.it.

Questo driver è basato sul chip A3967SLB della Allegro ed è in grado di controllare un singolo motore passo-passo bipolare con possibilità di selezionare quattro modalità di controllo del motore: passo, 1/2 passo, 1/4 di passo e un 1/8 di passo. Consente di impostare la corrente in uscita tramite l'apposito trimmer (montato sul circuito).

La versione di EasyDriver usata in questo esempio è la 4.3 ed i rispettivi collegamenti sono illustrati nella **Tabella 4**.

pin	con	funzione
Motor A	Jp3	Avvolgimento A.
Motor A	Jp3	Avvolgimento A.
Motor B	Jp3	Avvolgimento B.
Motor B	Jp3	Avvolgimento B.
GND	Jp1	Massa alimentazione motori.
M+	Jp1	Alimentazione positiva motori (8-30volt).
GND	Jp4	Massa integrato A3967.
+5V	Jp4	Alimentazione integrato A3967 (per impostazione di fabbrica, è ricavata dalla tensione motori tramite stabilizzatore interno).
Cur Adj		Imposta la corrente di riposo dei motori (150÷750 mA).
APWR	Sj1	Ponticello (normalmente chiuso) alimentazione integrato dalla tensione motori. Tagliare per alimentare l'integrato dal connettore JP4.
3/5V	Sj2	Ponticello per impostare alimentazione integrato su 5 o 3,3 V e di conseguenza i livelli logici dei segnali di comando. Per impostazione predefinita, normalmente aperto Vcc=5 V e i segnali di comando sono 0÷5 V.
SLP	Jp5	Pone in standby l'integrato. Di default la linea è alta e l'integrato è attivo.
MS1	Jp5	Impostazioni funzione passo: MS1=1 e MS2=1 1/8 di passo (predefinito); MS1=0 e MS2=1 1/4 di passo; MS1=1 e MS2=0 1/2 di passo; MS1=0 e MS2=0 passo intero.
MS2	Jp6	
ENABLE	Jp6	Abilitazione integrato. Per impostazione predefinita, la linea è a livello basso e l'integrato è abilitato.
RST	Jp7	Reset dell'integrato. Per impostazione predefinita la linea è a livello alto e l'integrato non è resettato.
PFD	Jp7	Imposta la rapidità di variazione di corrente tra un passo ed il successivo. Permette di ottimizzare la funzionalità ad alto numero di giri. Preimpostato su di un valore intermedio.
GND	Jp2	Massa segnale di comando.
STEP	Jp2	Impulso di comando corrispondente ad uno step.
DIR	Jp2	Imposta la direzione di rotazione. La direzione di rotazione dipende anche dal cablaggio degli avvolgimenti.

***Tabella 4** – Collegamenti della EasyDriver 4.3.*

Non dovete spaventarvi di tutti i contatti di cui dispone, in quanto il driver è già impostato in modo ottimale per la maggior delle applicazioni; l'unica raccomandazione è non collegare e scollegare il motore quando il driver e alimentalo, per non danneggiare l'integrato A3967.

Angolo passo-passo	1,8° (200 passi)
Numero di fasi	2 (bipolare)
Resistenza per fase	55 ohm
Induttanza per fase	80 mH
Resistenza d'isolamento	100 Mohm min. (500 Vcc)
Classe d'isolamento	B
Inerzia del rotore	54 g.cm²
Massa	0,23 kg
Alimentazione	max. 15,4 V
Consumo	0,28 A
Coppia di tenuta (coppia che, con motore alimentato, si oppone alla rotazione)	2,4 kg x cm
Coppia residua (coppia che si oppone alla rotazione dell'albero di un motore non alimentato)	120 g x cm
Dimensioni	42,3 x 42,3 x 37 mm

***Tabella 5** – Caratteristiche del motore 7300-STEPMOT01 della Futura Elettronica.*

Il motore utilizzato in questo esempio è distribuito dalla ditta Futura Elettronica (codice 7300-STEPMOT01); esso ha le caratteristiche descritte nella **Tabella 5**.

In ogni caso, assicuratevi che il motore che vi accingete ad usare non assorba una corrente superiore a 750 mA e possa essere alimentato con una tensione compresa tra 8 e 30 volt, oltre ad essere di tipo bipolare.

Per sapere se il motore sia efettivamente bipolare è sufficiente verificare se ha almeno quattro fili; allo scopo utilizzate un tester e misurate la continuità tra i vari fili per identificare i due avvolgimenti A e B. Se avete recuperato il motore in qualche discarica e non trovate i suoi dati, assicuratevi almeno che la resistenza di ogni fase sia sufficientemente alta per poter garantire una corrente non superiore ai 750 mA. Ipotizzando una tensione di alimentazione di 12 voli, la resistenza di ciascuna fase dovrà essere di almeno 16ohm; se trovate un valore più basso, il motore non è adatto al nostro driver.

Appurata l'idoneità del motore, non vi resta che cablare il tutto seguendo le indicazioni riportate nella **Tabella 6**; in essa, per comodità ai pin del driver EasyDriver sono stati saldati alcuni strip maschio per poterlo utilizzare con una piccola breadboard. Facciamo in modo che l'alimentazione tramite il plug di Arduino possa alimentare anche EasyDriver, sfruttando il pm Vin allo scopo. Per far funzionare l'insieme, è sufficiente utilizzare il solito alimentatore non stabilizzato impostato per una tensione di uscita di 6÷9 V. Un pezzetto di nastro adesivo renderà chiaramente visibile la rotazione del perno del motore.

Pin	Collegamento
Motor A	Avvolgimento A
Motor A	Avvolgimento A
Motor B	Avvolgimento B
Motor B	Avvolgimento B
GND	Massa alimentazione motori. Da collegarsi al pin GND di Arduino.
M+	Alimentazione positiva motori. Da collegarsi al pin vin di Arduino.
Cur Adj	Regoliamola al minimo, risparmieremo corrente, anche se il motore avrà meno forza a rotore fermo.
GND	Massa segnale di comando. Da collegare al pin GND della scheda Arduino.
STEP	Impulso di comando corrispondente ad un step. Da collegare al pin 9 della scheda Arduino.
DIR	Imposta la direzione di rotazione. Da collegare al pin 10 della scheda Arduino.

***Tabella 6** – Connessioni di EasyDriver da usare per la nostra applicazione. Tutti gli altri pin non sono utilizzati.*

Per quanto riguarda il programma di test, è sufficiente che generi una sequenza di impulsi alla frequenza desiderata, per il numero impostato. Lo sketch prevede una procedura denominata *Rotate*, alla quale passare come parametri la direzione, il numero di impulsi ed il ritardo tra un impulso ed il successivo. Il suo utilizzo è molto semplice: se, ad esempio, con il motore in nostro possesso volessimo compiere un giro intero, non dovremmo fare altro che generare 1.600 impulsi. Questo perchè il motore è un 200 step, ognuno dei quali è ampio 1,8° ; sapendo che tramile i pin MS1 e MS2, Easydriver è impostato ad 1/8 di passo, vediamo che effettivamente per un giro (formato da 200 passi) servono 200•8=1600 impulsi. Volendo compiere un giro in due secondi, otteniamo anche la variabile Delay: Delay=2/1.600=1.250µs.

In generale i Delay=(durata della rotazione)/(numero di impulsi) e steps=(Numero di giri)/1600.

Listato 3

```
{codecitation class:"brush:cpp"}
int StepPin = 9;
int DlrPin = 10;


void setup()

{
pinMode(StepPln. OUTPUT);
pinMode(DirPin, OUTPUT);
}
vold loop()
delay(1000);
Rotate(fa1se,1000, 1250);  // Richiama funzione per la rotazione

}
vold Rotate(boolean dir,int steps, int Delay)

{
// Dir direzione true o false
// Steps numero di Impulsi
// Delay ritardo tra un impulso ed il successivo


digitalWrite(DirPin,dir);
delay(50);
for(int   i=0;i<steps;1++)

{                                    //  ripete steps  volte
digitalWrite(StepPin, HIGH);                  //  pone alto   11   pin
delayMicroseconds(Delay/2);                       // attende
digitalWrite(StepPin, LOW);                     pone basso 11   pin
delayMicroseconds(Delay/2);                 // attende
}

}

{/codecitation}
```

Il **Listato 3** espone il codice necessario alla gestione del motore e impiegante la variabile Delay. La libreria "stepper.h" non e compatibile con questo driver, ma è invece adatta ad un driver più semplificato, senza logica, in quanto la sequenza dell'alimentazione delle fasi del motore viene generata internamente dal firmware; sul silo ufficiale Arduino potrete trovare alcuni esempi applicativi in cui è specificato il tipo di hardware da usarsi.