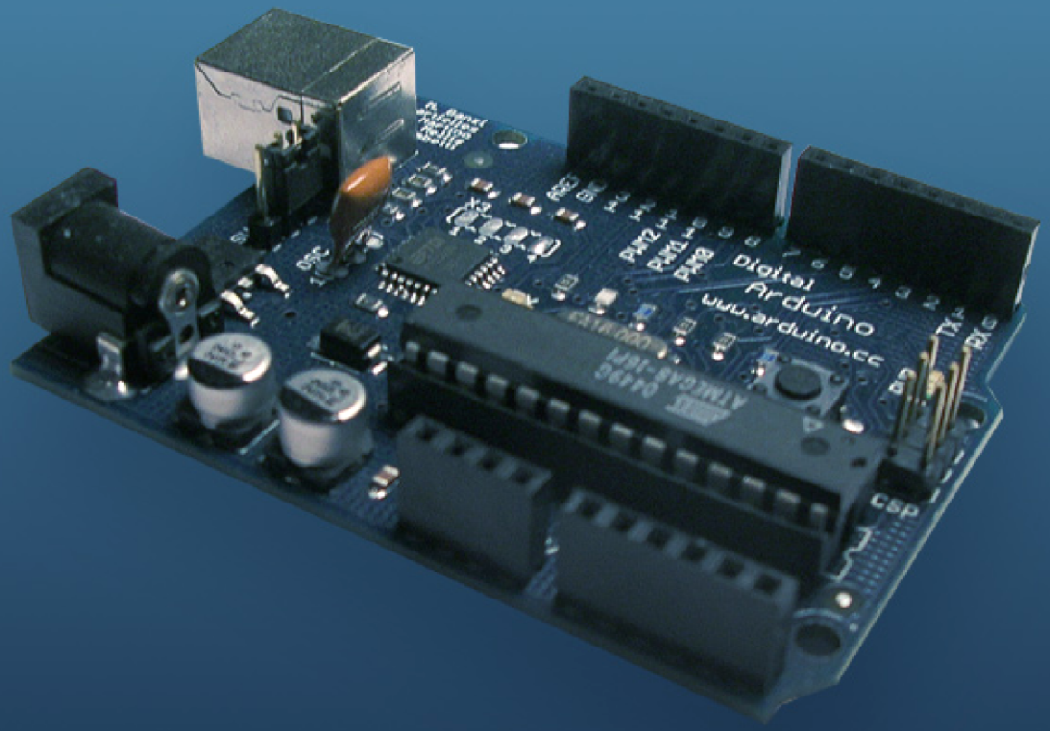
Arduino

Physical Computing I/O board



8^ parte: Pilotare Motori passo-passo bipolari usando l'integrato SN754410NE

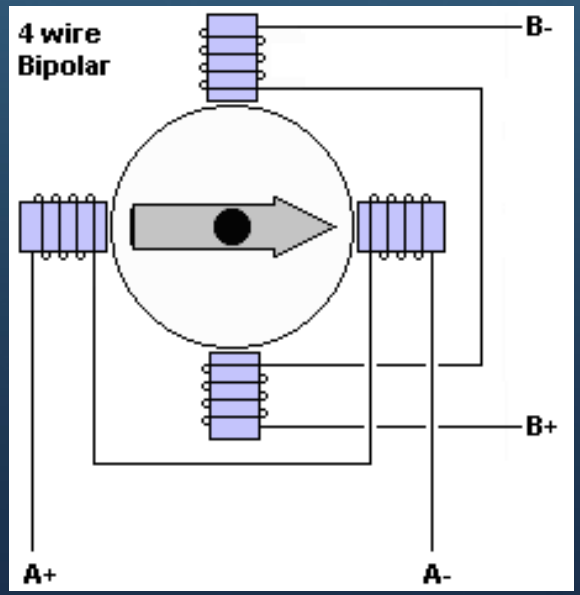
Author: Ing. Sebastiano Giannitto (ITIS "M.BARTOLO" –PACHINO)

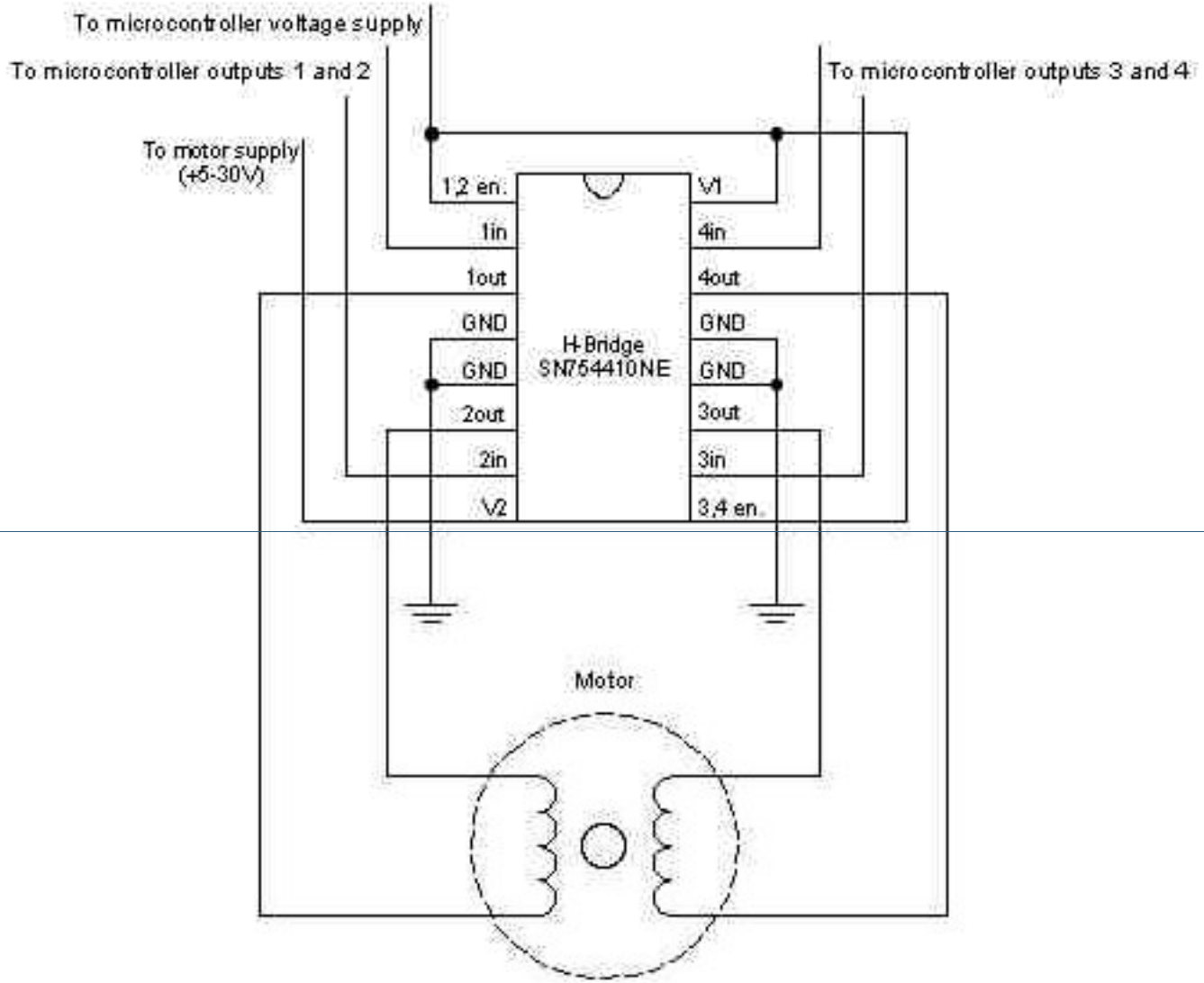
Esperienza n° 7

Lo scopo del progetto è riuscire a gestire un motore passo-passo bipolare con **Arduino e controllarne anche la velocità**.

I motori passo-passo bipolari di base presentano 4 avvolgimenti collegati come in figura, questo significa che se hai un motore a 5 o 6 fili puoi utilizzarlo come bipolare, facendo una forzatura e non utilizzando i 2 comuni, o se hai un motore a 8 fili puoi eseguire i collegamenti tra gli avvolgimenti per poterlo utilizare come bipolare.

A questo punto ti occorre un driver per pilotare questo tipo di motori, sul sito ufficiale Arduino si trova una pagina interessante, in cui sono rappresentati due integrati o driver utilizzabili per pilotare questi motori, per semplicità elettronica, non ci sono componenti extra da utilizzare.
Noi abbiamo deciso di utilizzare l'integrato SN754410NE:





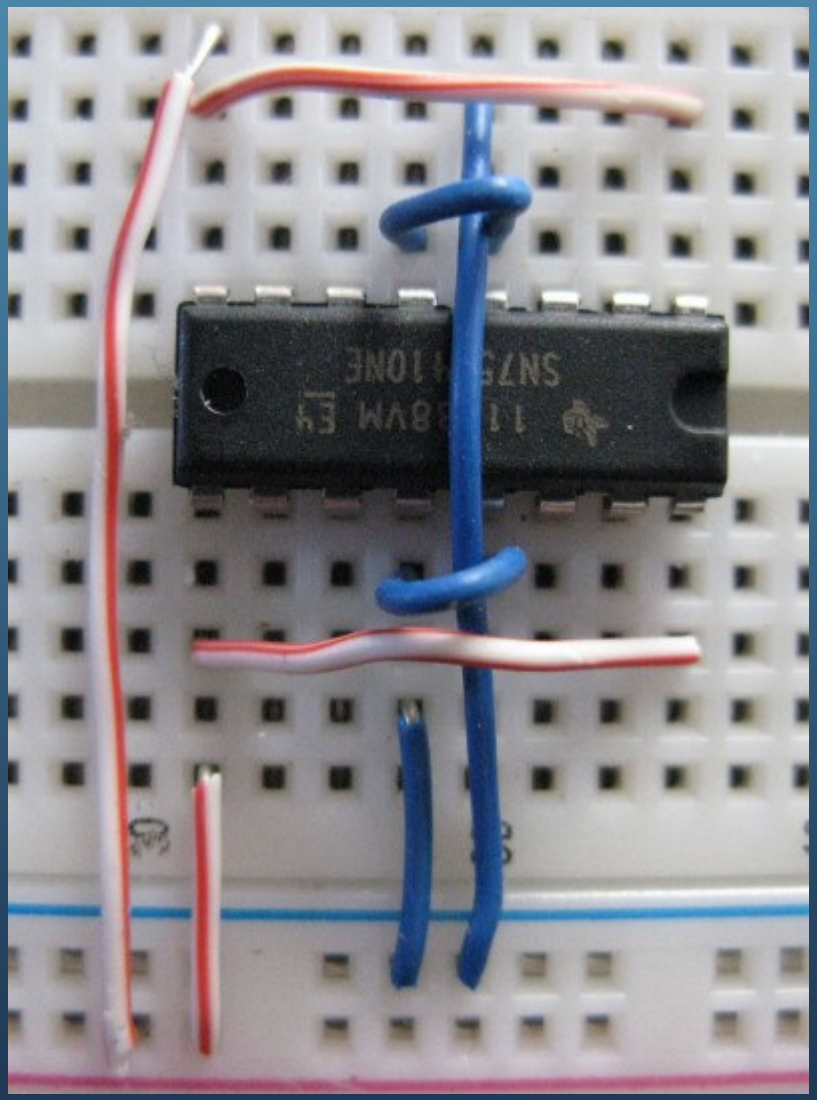
motori bipolari stepper a quattro pin SN754410NE

Image 1 of 9

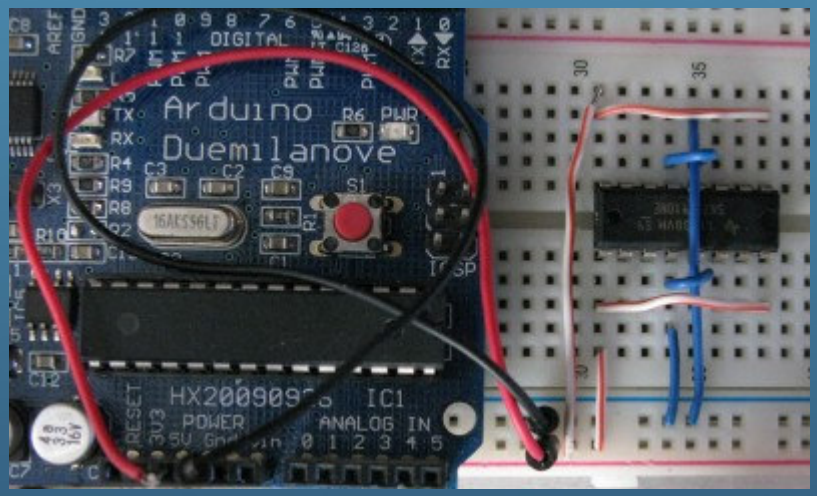
Su breadboard:

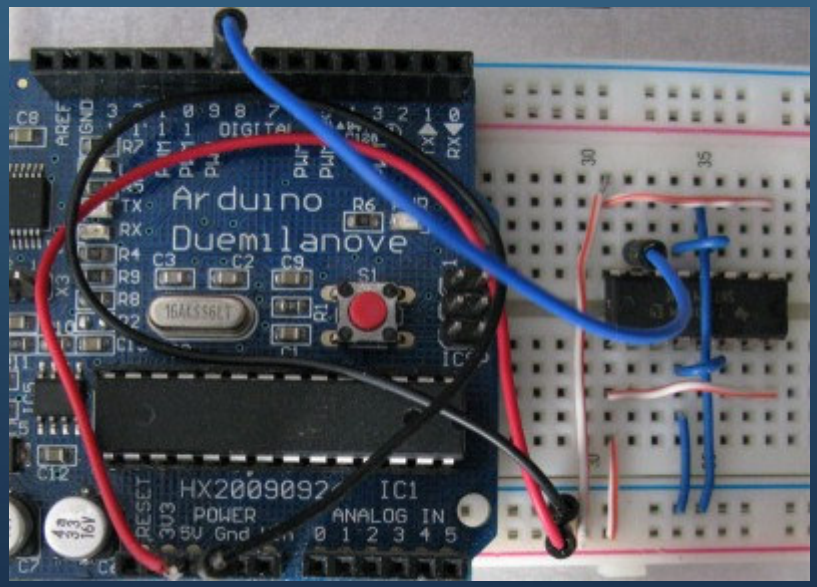
Collegare i pin 4,5,12 e 13 a massa (GND), il pin 16 a Vcc (+5v) ed i pin 1 e 9 ai +5v ricordando che questi ultimi due sono l'enable del primo e del secondo avvolgimento, per cui potremmo decidere di pilotarli con un segnale digitale da Arduino, per semplicità nello schema sono collegati al +5v ossia sempre abilitati.

Il pin 8 è connesso al +5v, questo pin dell'SN754410NE è destinato all'alimentazione esterna del motore, per cui se il tuo motore stepper funziona a 12v è a questo pin che dovremo connettere tale alimentazione, nel nostro esempio è collegato a +5V perché abbiamo utilizzato un motore funzionante a tale voltaggio.

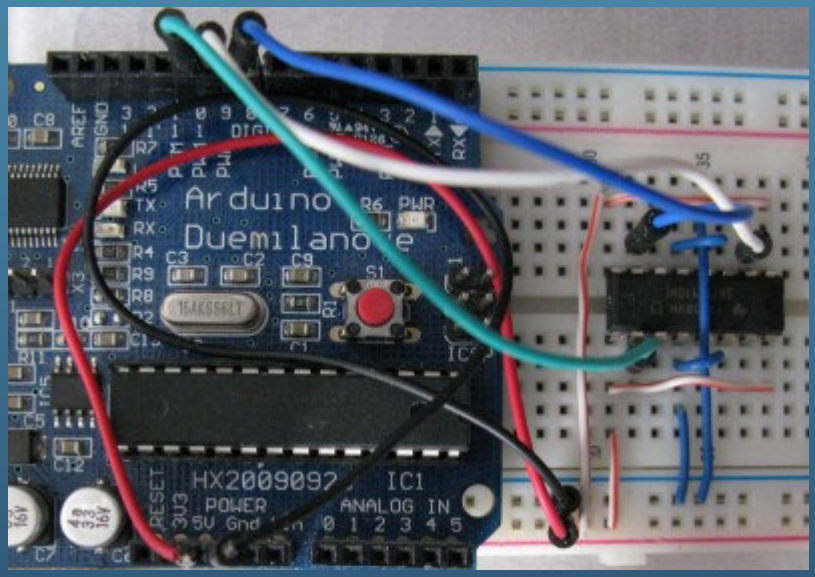


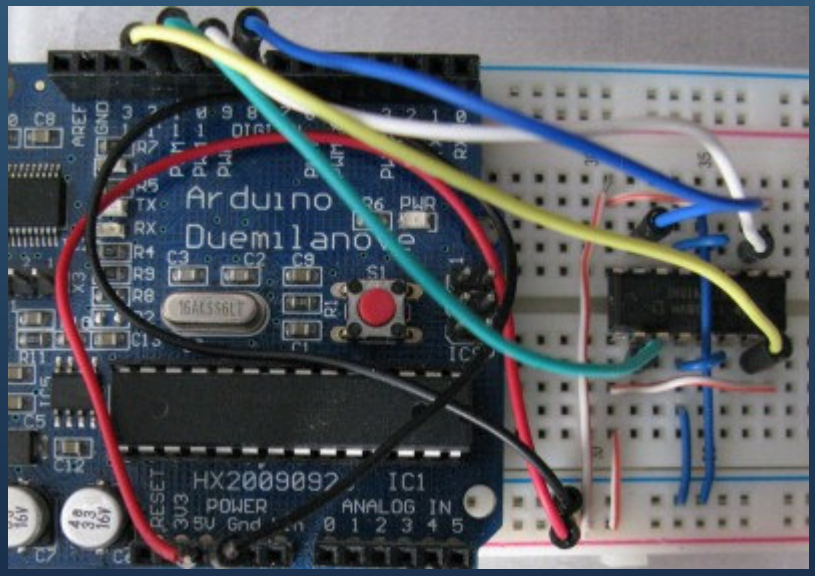
Ecco come collegare l'SN754410NE ad Arduino.
Come anticipavamo se anche il nostro circuito funziona a 5v collegheremo il positivo ai +5v ed il negativo a massa (GND) di Arduino.
Se il circuito non funziona a 5v allora collegheremo i +5v di arduino ai pin 1,9 e 16 e il pin 8 all'alimentazione esterna;
Il negativo del circuito dovremo collegarlo sia al GND di Arduino sia al negativo dell'alimentazione esterna, altrimenti non otterremo nessun movimento del motore.
Passiamo a collegare ad uno ad uno i pin di INPUT dell'SN754410NE ad Arduino, rispettivamente 7,2,10 e 15 del driver sui pin 8,9,10 e 11 di Arduino, come nelle figure seguenti:



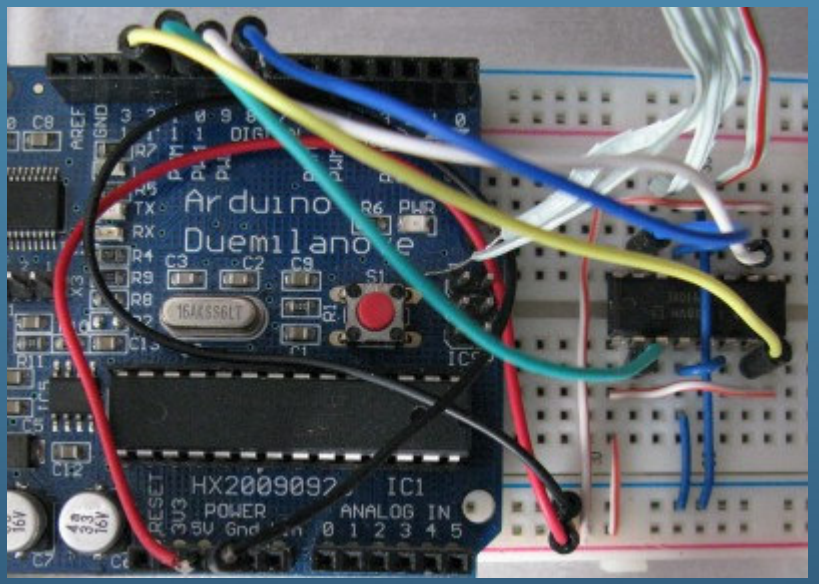


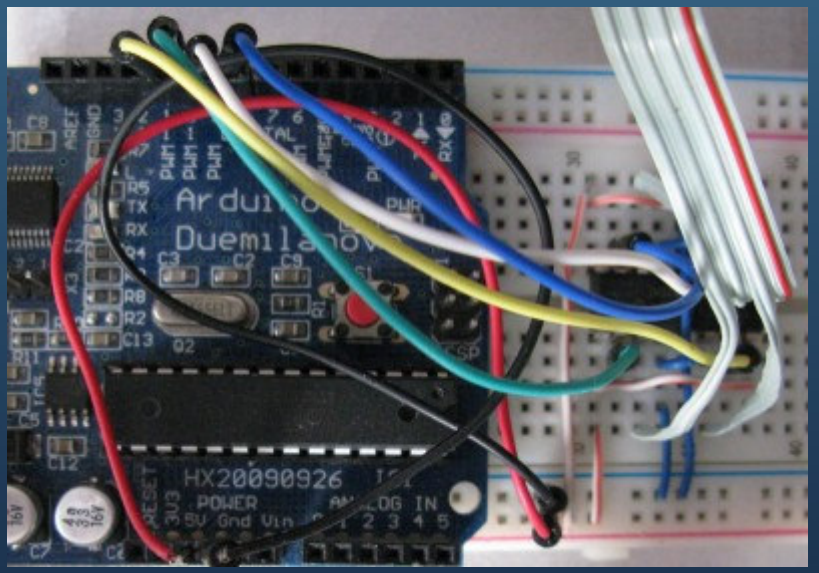
Non ci resta che collegare il motore passo-passo al driver, prima di eseguire il collegamento dovremo aver chiaro quali siano i due distinti avvolgimenti del motore.
Tuttavia occorreranno alcune prove invertendo i collegamenti mentre Arduino è in funzione, per determinare quali siano i pin corretti relativi al primo avvolgimento A+ e ad A- così anche per il secondo.





Ecco come collegarli:
Se tutto è collegato correttamente quando caricheremo lo sketch su Arduino vedremo muoversi il motore passo-passo bipolare.





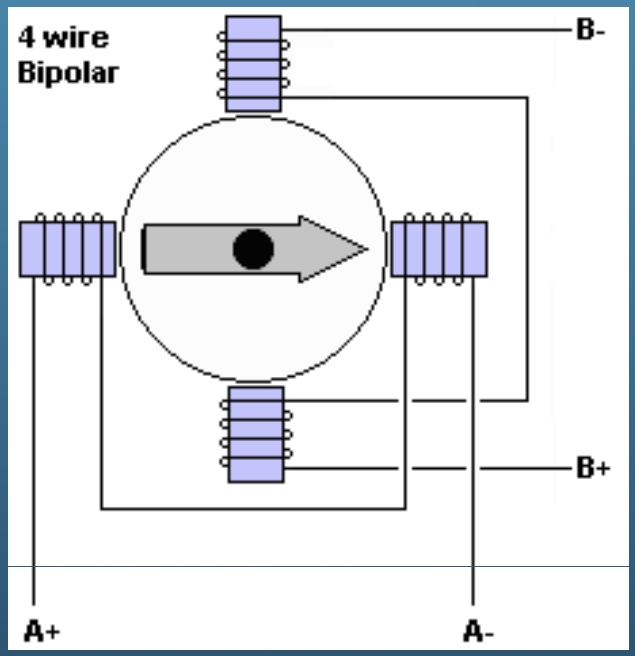
Il codice

```
int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 100;

void setup()
{
  Serial.begin(9600);
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}
```

```
void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

Lo sketch proposto non è complesso, ripete una sequenza di segnali in cui il valore HIGH si sposta lungo i 4 pin a cui è connesso il driver SN754410NE e di conseguenza il motore passo-passo.
Come avrai già letto il driver utilizzato non ha una sua intelligenza specifica, il suo scopo è consentirti di utilizzare tensioni e correnti superiori a quelle gestibili con Arduino e quindi pilotare motori con coppia elevata.
Tutta l'intelligenza del movimento proviene dallo sketch e da Arduino.
Prima di iniziare a sviscerare il programma linea per linea cerchiamo di capire il concetto con cui il motore esegue i suoi passi successivi: trattandosi di 4 bobine collegate a due a due poste agli antipodi il movimento è dato dalla polarizzazione delle bobine stesse:



Nella posizione in cui si trova il magnete centrale permanente, se volessi far spostare l'albero del motore in senso orario dovrei polarizzare correttamente la bobina in basso, applicare quindi un segnale positivo al B+ ed uno negativo al B- se l'albero ruota in senso orario abbiamo ottenuto quanto desiderato e sai che la bobina inferiore attiva il magnete centrale quando riceve corrente positiva su B+ e negativa da B- attraverso la bobina ai suoi antipodi.
Lo stesso concetto vale per A+ e A-, immaginando che il magnete si trovi rivolto verso il basso e che si voglia continuare la sua rotazione in senso orario applicheremo il positivo ad A+ ed il negativo ad A-; per spostarlo quindi verso l'alto dovremo attirare il magnete centrale (permanente) verso l'alto eccitando la bobina in alto, sapendo che per eccitare quella in basso abbiamo applicato positivo a B+ e negativo a B-, applichiamo i poli inversi (positivo a B- e negativo a B+) e la bobina in alto sarà eccitata facendo spostare l'alberino in senso orario di un altro step (passo).
Ripetendo il gioco dell'inversione dei poli su A+ e A- (positivo su A- e negativo su A+) l'alberino si sposta nella posizione iniziale.

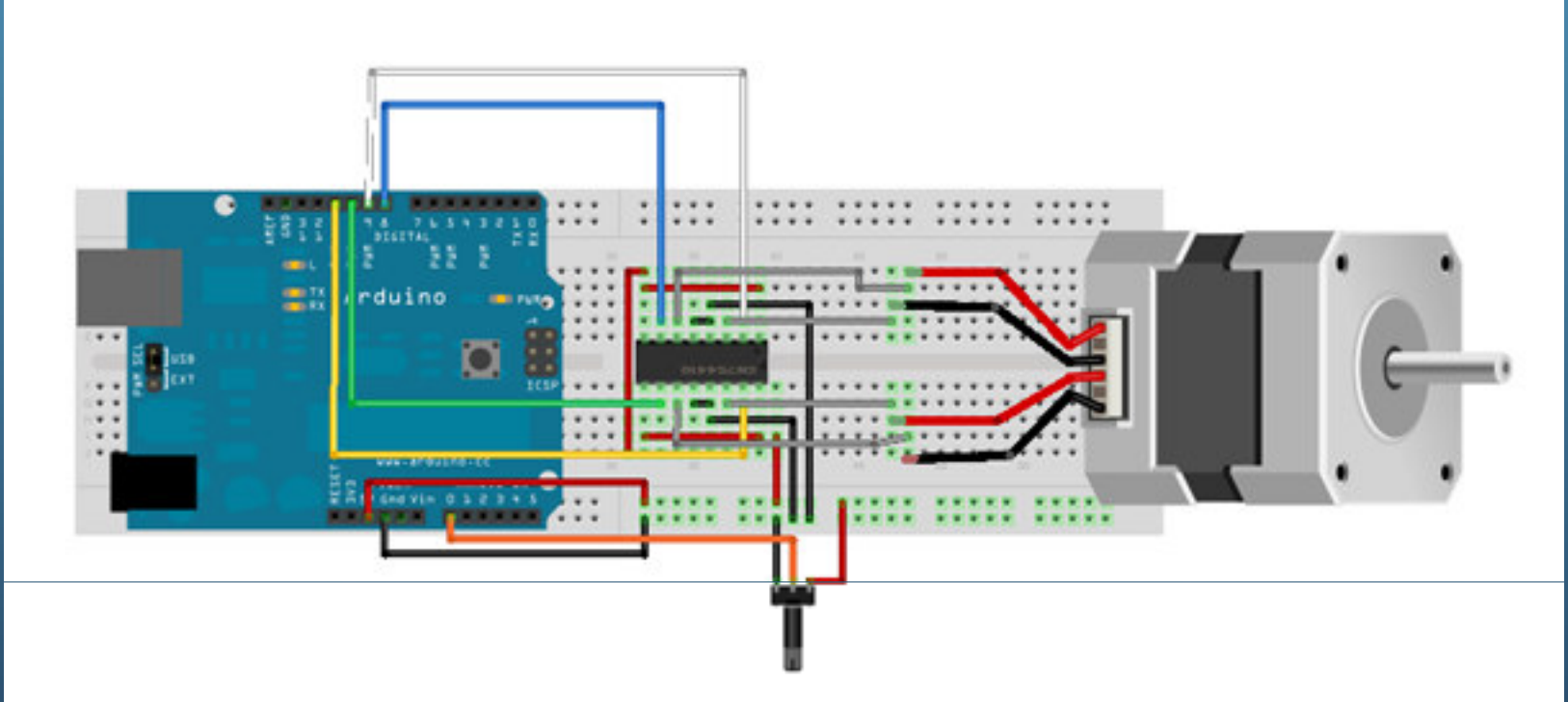
La sequenza che abbiamo appena utilizzato per eseguire un giro completo è:

	B+	B-	A+	A-
Step1	high	low	low	low
Step2	low	low	high	low
Step3	low	high	low	low
Step4	low	low	low	high

Iniziamo con la definizione dei pin a cui collegare il driver SN754410NE:
linee 01-04: definisci le quattro variabili di tipo *integer* ciascuna per uno dei 4 pin a cui è connesso il motore;
linea 05: definisci un tempo di attesa tra uno step ed il successivo, lo scopo è attendere questo tempo per darti modo di comprendere il senso di rotazione e verificare i collegamenti tra Arduino ed il driver/motore;
linee 07-12: definisci la funzione *setup()* in cui indichi ad Arduino che ciascuno dei 4 pin dichiarati nelle linee 01-04, è da utilizzare come OUTPUT ossia per inviare segnali;
linea 14: definisci la funzione *loop()* in cui esegui i comendi di rotazione vera e propria, la funzione *loop()* è divisa in 4 blocchi simili, in cui ciascun blocco ricalca uno degli step della Tabella sopra;
linee 15-18: definisci il primo blocco o step che il motore esegue, per farlo invii un segnale HIGH sul pin 8 che avrai avuto cura di collegare al B+ (vedi tabella) e LOW per tutti gli altri pin;

linea 19: attendi il tempo definito alla linea 05, prima di eseguire il passo successivo, ricorda che tale tmpo è definito in millisecondi;
linee 20-23: è un blocco del tutto simile al precedente che invia il segnale HIGH al pin 10, a cui avrai avuto cura di collegare A+;
linea 24: attendi un tempo identico a *delayTime* prima di compiere il passo successivo;
linee 25-28: compie un nuovopasso in avanti inviando HIGH al pin 9, e LOW a tutti gli altri, il pin 9 deve essere collegato a B-;
linea 29: attendi un tempo identico a *delayTime* prima di compiere il passo successivo;
linee 30-33: compie l'ultimo passo del nostro ciclo prima di ripetere tutta la sequenza, per eseguirlo invii al pin 11 il segnale HIGH e LOW a tutti gli altri;
linea 34: attendi il *delayTime* prima di ricominciare il ciclo di rotazione.
Ora che sai esattamente come pilotare questi motori potrai sperimentare altri sketch e realizzare nuovi progetti con Arduino e i motori bipolari, ricorda che se ne trovano molti nelle vecchie stampanti, non ti costa nulla e puoi divertirti davvero tanto.

Come regolarne la velocità.



Per collegare il potenziometro seguire lo schema in alto, in cui gli estremi del potenziometro sono connessi al positivo e negativo dell'alimentazione, in pratica la resistenza fissa di 10KΩ è tra positivo e negativo ed il pin centrale, quello relativo al cursore del potenziometro lo si collega al pin analogico 0 (A0) di Arduino. Si utilizzerà questo pin per rilevare il valore della resistenza assunta dal potenziometro.
Dovremo fare anche alcune modifiche allo sketch proposto in precedenza:
Aggiungere una nuova variabile al gruppo delle variabili già impostate:
int pinPot = 0;

serve a definire a quale pin hai collegato il potenziometro, nell'esempio è collegato al pin 0;
Nella funzione *setup()* devi aggiungere una nuova linea dopo le altre che indichi ad Arduino di utilizzare il pin 0 (pinPot) come INPUT, ossia per leggere il valore di resistenza impostato:

```
pinMode(pinPot, INPUT);
```

Ed infine alcune modifiche alla funzione *loop()*, in primo luogo aggiungi subito dopo la definizione della funzione loop() le seguenti linee:

```
int readPin = analogRead(pinPot);
int delayTime=map(readPin,0,1024,0,500);
```

Il cui significato è per la linea 01 l'impostazione di una nuova variabile di tipo *integer* in cui farai confluire il valore letto sul pinPot mediante l'utilizzo della funzione *analogRead*, se vuoi maggiori dettagli su questa funzione e sul suo utilizzo puoi leggere l'articolo Programmare con Arduino – leggere un segnale analogico; la linea 2 imposta il*delayTime*, in italiano tempo di attesa, uguale alla proporzione matematica tra lo 0 letto e lo 0 impostato e il valore 1024 letto e il 500 come valore massimo, in pratica la funzione *map()* restituisce in tempo reale la rimappatura del valore minimo e massimo letto rispetto ai valori minimo e massimo desiderati, quando sul pin 0 leggerai un valore 0 -> map restituirà 0; quando leggerai 1024 -> map restituirà 500; quando leggerai 512 -> map restituirà 250; e così via per tutti i valori intermedi da 0 a 1024.
Non occorrono altre modifiche in quanto il tempo calcolato su delayTime è già impostato come quello da attendere tra un passo e quello successivo.

Ora hai chiaro anche il perchè della proporzione inversa tra resistenza e velocità di rotazione, il motivo è che al crescere della resistenza aumenta il tempo di attesa tra un passo ed il successivo per cui una velocità più lenta; al diminuire della resistenza diminuisce anche il tempo di rotazione comportando una velocità maggiore di rotazione.

```
int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int pinPot = 0;

void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);

  pinMode(pinPot, INPUT);
}
```

```
void loop() {
  int readPin = analogRead(pinPot);
  int delayTime=map(readPin,0,1024,0,500);

  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

https://www.youtube.com/watch?x-yt-ts=1422327029&x-yt-cl=84838260&v=bn-D3t7O4bA