Pilotare un motore passo passo con Arduino e il driver A4988

DI ANDREA · 27 APRILE 2017

Scammenting in corso… Come pilotare un motore passo passo con Arduino e il driver A4988.

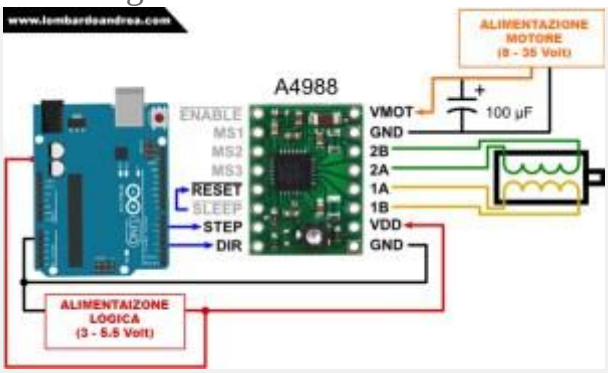
Nel mio caso utilizzerò un modulo Pololu A4988. Si tratta di un driver completo e molto semplice da utilizzare. Questo integrato è progettato per poter comandare i nostri motori in modalità full-, half-, quarter-, eighth-, e sixteenth-step. Ovvero possiamo far ruotare il nostro asse di uno step completo oppure di 1/2, 1/4, 1/8 e 1/16. Grazie a questa funzionalità possiamo affinare di parecchio i movimenti del motore.

La semplicità di utilizzo sta nel fatto che al nostro driver in una configurazione base, basteranno solo due PIN (non considerando l'alimentazione logica) ai quali far arrivare due segnali per deciderne la velocità ed il senso di marcia del motore.

La velocità, tramite il pin STEP richiede un alternanza (pulse) di stati logici alto/basso per poter far muovere il motore, mentre tramite il pin DIR, decideremo il senso di marcia.

Per maggiori dettagli e caratteristiche ti invito a leggere **qui** il datasheet.Ti anticipo solo che la capacità di carico arriva fino a 35 Volt con una corrente di uscita di ±2 Ampere.

Ecco come si presenta una configurazione base:



Configurazione base

Nell'immagine che hai appena visto i PIN come MS1, MS2, MS3, ed ENABLEnon sono connessi. Molto velocemente ti spiego a cosa servono.

I primi 3, ovvero MS1, MS2, MS3, decretano la risoluzione di rotazione del motore passo passo in base alla configurazione riportata in tabella:

MS1	MS2	MS3	Risoluzione Motore
Low	Low	Low	1 – Full step

High	Low	Low	1/2 – Half step
Low	High	Low	1/4 – Quarter step
High	High	Low	1/8 – Eighth step
High	High	High	1/16 – Sixteenth step

Il pin ENABLE serve ad abilitare o disabilitare le uscite dei FET che pilotano il motore. Se impostato ad un valore alto, le uscite saranno disabilitate, ovvero il motore verrà rilasciato. Se impostato ad un valore logico basso, le uscite saranno attivate. Questo pin gestisce soltanto le uscite relative al motore. Indipendentemente dal suo stato, il resto dell'elettronica rimarrà attivo.
Se vogliamo invece mettere a nanna il nostro driver e con esso tutta la sua elettronica, possiamo utilizzare il pin SLEEP. Impostandolo ad un valore logico basso, l'intera circuiteria (compresa quella dei FET) verrà messa a riposo, con conseguente risparmio energetico.
Un altro aspetto molto importante del modulo Pololu A4988 è la possibilità di regolare la corrente in uscita verso il motore. Abbiamo infatti visto in altri esempi (L298N Pilotare un motore passo passo con Arduino) che pilotando un motore passo passo, senza l'utilizzo di un limitatore di corrente o non dando la giusta corrente, il risultato sfocia in un surriscaldamento del motore e del driver stesso.
Per regolare la corrente di uscita, il modulo presenta un piccolo trimmer onboard. Guarda il video del produttore per capire come fare.
Durante i miei test non ho fatto nessuna regolazione, sarà fortuna o sarà un caso, ma né il modulo né il motore sembrano risentirne.

Bando alle ciance e vediamo un esempio pratico

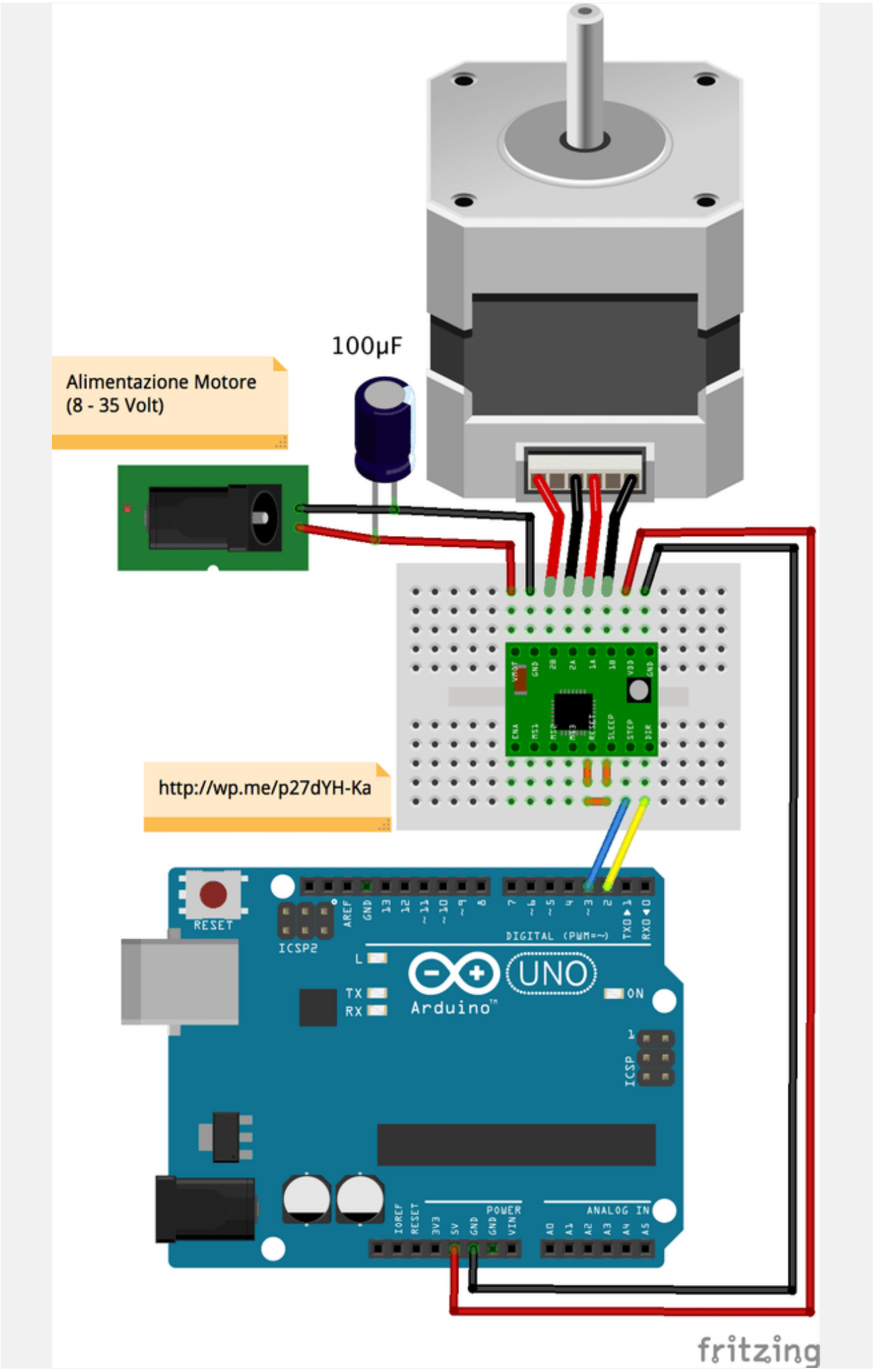
Come al solito ho messo i commenti nel codice, ma se hai dei dubbi usa pure la sezione commenti.

Il codice

```
1.  //Definizione dei PIN
2.  const int pinDir = 2;
3.  const int pinStep = 3;
4.
5.  const int numStepMotore = 200; //E' il numero di step per rotazione del motore (potrebbe differire in base al modello)
6.  const long velocita = 1000; //Si tratta di microsecondi tra un impulso e l'altro sul pin STEP
7.
8.  void setup() {
9.    //inizializzo i PIN come OUTPUT
10.   pinMode(pinStep, OUTPUT);
11.   pinMode(pinDir, OUTPUT);
12. }
13. void loop() {
14.
```

```
  //definiamo la direzione del motore
  digitalWrite(pinDir, HIGH);

  //esegue un giro completo in un senso
  for (int x = 0; x < numStepMotore; x++) {
    digitalWrite(pinStep, HIGH);
    delayMicroseconds(velocita);
    digitalWrite(pinStep, LOW);
    delayMicroseconds(velocita);
  }

  //aspetto 2 secondi
  delay(2000);

  //cambio la direzione di marcia
  digitalWrite(pinDir, LOW);

  //rieseguo un altro giro completo nel senso opposto
  for (int x = 0; x < numStepMotore; x++) {
    digitalWrite(pinStep, HIGH);
    delayMicroseconds(velocita);
    digitalWrite(pinStep, LOW);
    delayMicroseconds(velocita);
  }

  //aspetto altri 2 secondi
  delay(2000);
}
```

Lo schema dei collegamenti


100µF
Alimentazione Motore
(8 - 35 Volt)
http://wp.me/p27dYH-Ka
RESET
ICSP2
DIGITAL (PWM=~)
UNO
Arduino™
ON
ICSP
POWER
ANALOG IN
fritzing

Arduino A4988 Connessione Base

Per dovere di cronaca riporto due avvertimenti importanti presenti sul sito del produttore del modulo A4988.

Warning: *This carrier board uses low-ESR ceramic capacitors, which makes it susceptible to destructive LC voltage spikes, especially when using power leads longer than a few inches. Under the right conditions, these spikes can exceed the 35 V maximum voltage rating for the A4988 and permanently damage the board, even when the motor supply voltage is as low as 12 V. One way to protect the driver from such spikes is to put a large (at least 47 µF) electrolytic capacitor across motor power (VMOT) and ground somewhere close to the board.*
Warning: *Connecting or disconnecting a stepper motor while the driver is powered can destroy the driver. (More generally, rewiring anything while it is powered is asking for trouble.)*

In sostanza… usate il condensatore come riportato negli esempi, e non collegate o scollegate il modulo se l'alimentazione è presente.

Bene… in occasione di questo articolo, ne approfitto per rispondere ad una richiesta proposta da un utente di YouTube su un altro mio video.
L'obbiettivo è quello di pilotare un motore passo passo tramite tre pulsanti, due utilizzati per deciderne la direzione, l'altro per determinarne lo stop.
Anche se la richiesta è stata fatta su un video che mostra come pilotare un motore passo passo con Arduino ed un modulo L298N, ho deciso che per ovvi motivi legati al surriscaldamento da troppa corrente, è bene iniziare ad usare i driver giusti al posto giusto.
Per questo progetto ho usato due librerie nuove (per me). Si chiamano AccelStepper e Bounce2.
La prima offre un sacco di metodi e funzionalità per comandare motori passo passo anche in maniera sincrona e soprattutto, a differenza della classica libreria Stepper di Arduino, questa non presenta blocking functions, ovvero funzioni che bloccano l'esecuzione del codice fin quando non hanno completato la loro operazione.
La seconda invece, serve ad evitare quei fastidiosi casi in cui la pressione di un bottone viene interpretata da Arduino come una successione di pressioni, dovute alla natura elettrica dei contatti.

Per installare le librerie, dall'IDE di Arduino andare su Sketch->Includi Libreria->Gestione Librerie ed utilizzare il campo filtra per cercarle.
Una volta installate le due librerie, passiamo al codice:

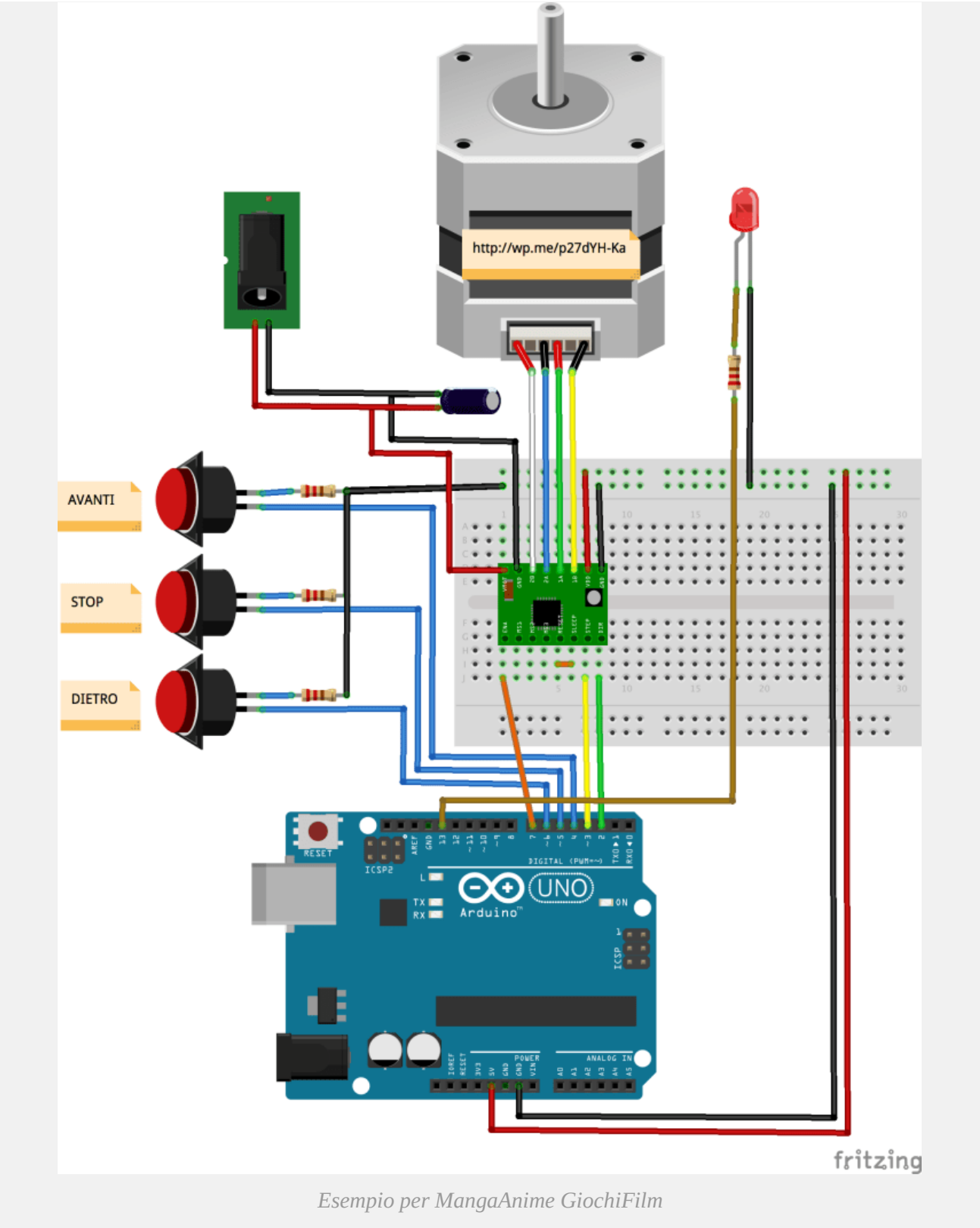
```
/*
  Pilotare un motore passo passo mediante tre pulsanti.
  I pulsanti determinano il senso di marcia e lo stop.
  Progetto realizzato su richiesta di:
  MangaAnime GiochiFilm : https://www.youtube.com/channel/UCmMfiWIApA-eHCxsMiBcpOQ

  Autore  : Andrea Lombardo
  Web     : http://www.lombardoandrea.com
  Post    : http://wp.me/p27dYH-Ka
*/

//Inclusione delle librerie
#include <Bounce2.h>
#include <AccelStepper.h>

//definizione dei pin
const int motoreDir = 2;
const int motoreStep = 3;

const int pinBtnAvanti = 4;
const int pinBtnStop = 5;
const int pinBtnDietro = 6;
const int pinEnable = 7;

//uso il led onboard di Arduino, per avere un riscontro visivo dello stato di marcia o di stop
const int pinLedStato = 13;

/*
  Velocità del motore in numero di step per secondo
  Come riportato nella descrizione di AccelStepper, massimo 4000 per un Arduino UNO.
  4000 steps per second at a clock frequency of 16 MHz on Arduino such as Uno
*/
const float velocita = 1000;

//Millisecondi per il debounce dei pulsanti
const int debounceMs = 10;

//Stato e direzione
boolean abilitato, direzione;

//Creazione istanza del motore
AccelStepper motore(AccelStepper::DRIVER, motoreStep, motoreDir);

//Creazione istanze dei pulsanti
Bounce btnAvanti = Bounce();
Bounce btnStop = Bounce();
Bounce btnDietro = Bounce();

void setup() {
  //imposto stato e direzione iniziali (motore fermo)
  abilitato = direzione = false;

  //definizione della modalità dei pin
  pinMode(pinBtnAvanti, INPUT);
```

```
55.   pinMode(pinBtnStop, INPUT);
56.   pinMode(pinBtnDietro, INPUT);
57.   pinMode(pinEnable, OUTPUT);
58.   pinMode(pinLedStato, OUTPUT);
59.
60.   //impostazione pullUp per i pin dei bottoni
61.   digitalWrite(pinBtnAvanti, HIGH);
62.   digitalWrite(pinBtnStop, HIGH);
63.   digitalWrite(pinBtnDietro, HIGH);
64.
65.   //setta stato del pin enable e del led di stato
66.   digitalWrite(pinEnable, !abilitato);
67.   digitalWrite(pinLedStato, abilitato);
68.
69.   //configurazione parametri motore
70.   motore.setMaxSpeed(velocita);
71.   motore.setSpeed(velocita);
72.
73.   //inizializzazione bottoni
74.   btnAvanti.attach(pinBtnAvanti);
75.   btnAvanti.interval(debounceMs);
76.
77.   btnStop.attach(pinBtnStop);
78.   btnStop.interval(debounceMs);
79.
80.   btnDietro.attach(pinBtnDietro);
81.   btnDietro.interval(debounceMs);
82.
83. }
84.
85. void loop() {
86.
87.   //refresh stato dei pulsanti
88.   btnAvanti.update();
89.   btnStop.update();
90.   btnDietro.update();
91.
92.   //leggi valore dei pulsanti
93.   int valStop = btnStop.read();
94.   int valAvanti = btnAvanti.read();
95.   int valDietro = btnDietro.read();
96.
97.   //determina azioni
98.   if (valStop == LOW) {
99.     abilitato = false;
100.  }
101.
102.  if (valAvanti == LOW) {
103.    abilitato = true;
104.    direzione = true;
105.  }
106.
107.  if (valDietro == LOW) {
108.    abilitato = true;
109.    direzione = false;
110.  }
111.
112.  //cambia stato ai pin ENABLE e LedStato
113.  digitalWrite(pinEnable, !abilitato);
```

```
114.   digitalWrite(pinLedStato, abilitato);
115.
116.   //se abilitato == true
117.   if (abilitato) {
118.
119.     //in base al valore di direzione
120.     if (direzione) {
121.       //imposta velocità (e senso)
122.       motore.setSpeed(velocita);
123.     } else {
124.       //imposta velocità negata (inversione di marcia)
125.       motore.setSpeed(-velocita);
126.     }
127.
128.     //muovi il motore
129.     motore.runSpeed();
130.   } else { //se abilitato == false
131.     //in realtà questa istruzione viene ignorata inquanto il pin ENABLE del modulo
132.     //è stato impostato ad HIGH e quindi le uscite dei FET sono spente
133.     motore.stop();
134.   }
135.
136. }
```

Come hai potuto notare, in questo esempio ho utilizzato anche la funzionalità del pin ENABLE del modulo A4988. Quando premiamo il pulsante di stop, lo stato di ENABLE viene impostato ad HIGH, con conseguente rilascio del motore dovuto allo "spegnimento" dei FET che gestiscono le uscite 1A, 1B, 2A, 2B.

Lo schema dei collegamenti



Esempio per MangaAnime GiochiFilm

Spero che questo post ti sia stato utile! Per qualsiasi domanda o chiarimento, fai riferimento alla sezione commenti.

Al più presto vorrei realizzare un video tutorial

Come sempre

- Assicurati che tutti i collegamenti siano corretti;

- Ricordati di impostare la porta COM del tuo Arduino;
- Utilizza le tensioni corrette;
- E ricorda che io non mi assumo nessuna responsabilità per eventuali danni o disastri che causi