D'angelo, Bonsi, Magro, Lipartiti, Rossi classe 5Bea

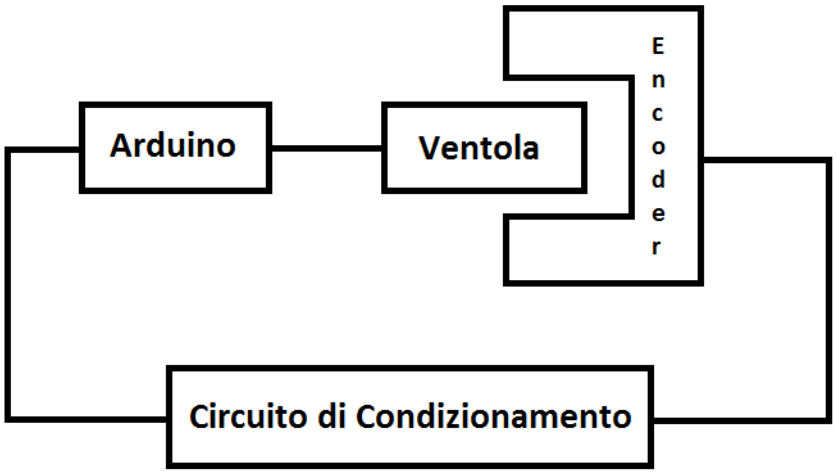
Controllo automatico della velocità di un motore in DC

Scopo:
Lo scopo dell'esperienza è mantenere ad un valore prestabilito la velocità di un motore DC, anche nel caso in cui venga applicato un carico (al motore).

Materiale utilizzato:
-breadbord
-cavi di collegamento
-led ricevente (phototransistor)
-led emittente (infrarosso)
-resistenza elettriche
-2 x integrati UA741
-diodo IN4007
-motore DC (ventola 12 [V])
-Arduino UNO
-Motor Shield Arduino

Strumenti utilizzati:
-oscilloscopio
-alimentatore
-generatore di funzioni
-tester
-ambiente di sviluppo Arduino 1.7.7 (software)
-attrezzi vari (forbici, pinza, cacciavite, ecc...)

Realizzazione del progetto:

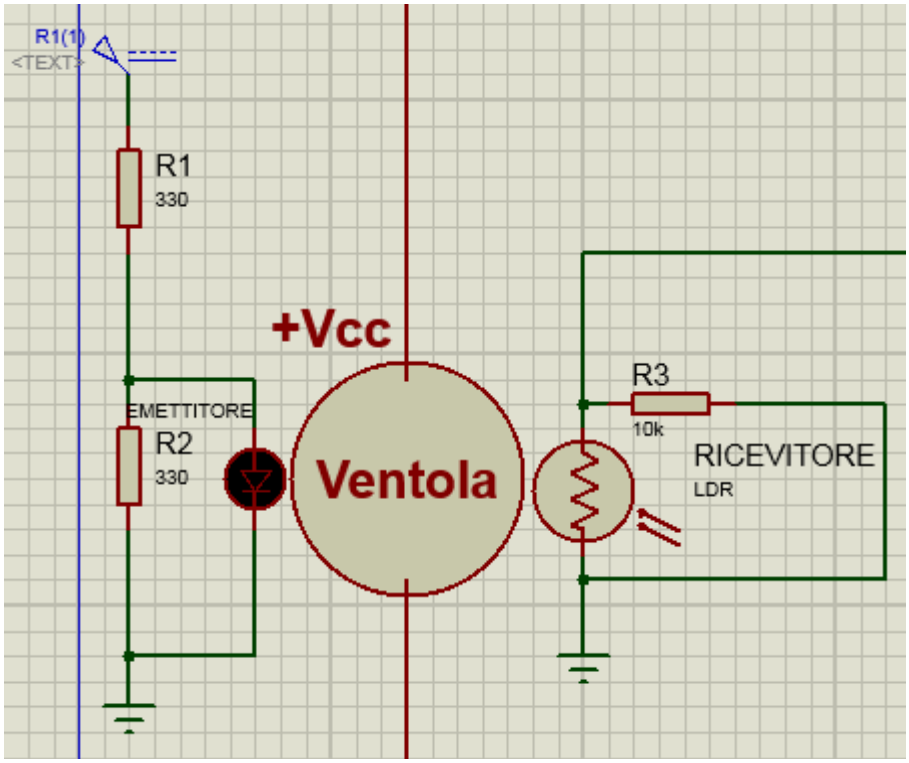


Il progetto è composto da 3 fasi:
1) misurazione della velocità della ventola
2) confronto della velocità reale con quella ideale

3) regolazione della velocità

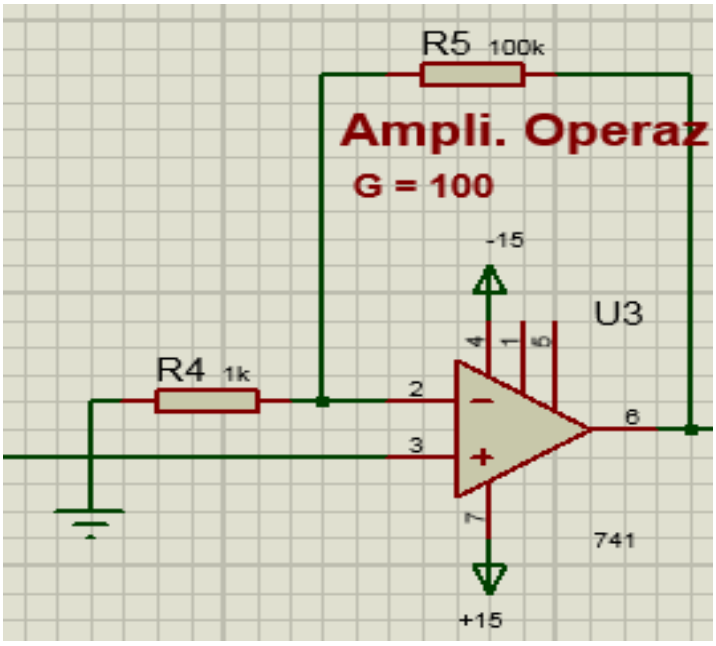
Misurazione della velocità della ventola

Come si può vedere nello schema a blocchi, per misurare la velocità della ventola, è necessario utilizzare un encoder, un sensore in grado di restituire(attraverso opportuni calcoli) la velocità di rotazione del motore a cui è fissato.
L'encoder che abbiamo realizzato è composto da due led, uno in grado di inviare un'onda infrarosso e l'altro di riceverla. Quando il secondo riceve l'onda, genera un segnale costante di circa 4 [mV], quando, invece, viene posta una barriera tra i due componenti (nel nostro caso le pale della ventola), genera una tensione costante di 2 [mV]. Il segnale d'uscita, quindi, è un'onda quadra, dovuta all'alternanza della presenza delle pale tra i due led, che varia la propria frequenza al variare della velocità di rotazione della ventola.
Misurando il periodo del segnale $V_{encoder}$, ricaviamo una frequenza che corrisponde circa al numero di giri (in realtà al numero di giri reale moltiplicato per il numero delle pale). Ecco il circuito elettrico dell'encoder:



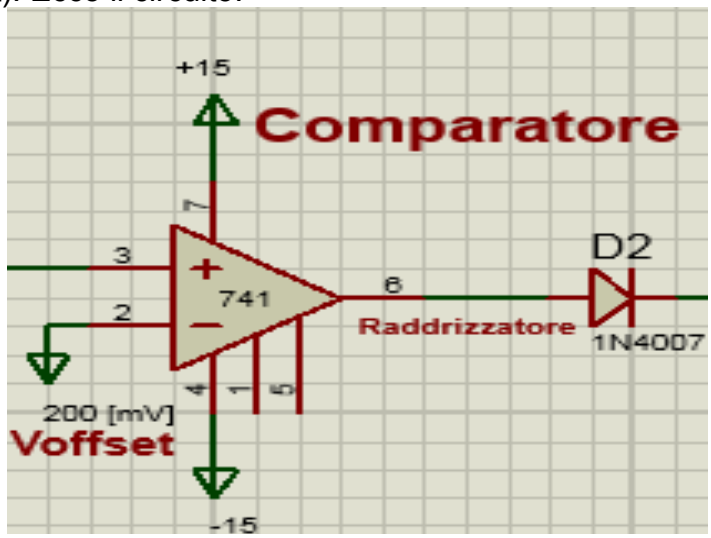
Il partitore formato da R1 e R2 permettere di ottenere una caduta di tensione ai capi del led emettitore di 2,5 [V] (la tensione d'alimentazione è di 5 Volt). La resistenza R3, invece, serve per ottenere una caduta di tensione quando il led ricevitore conduce.

La $V_{encoder}$, però, non può essere inviata direttamente al microcontrollore (Arduino UNO) a causa della sua ampiezza, infatti, Arduino, nei pin digitali può percepire solamente tensioni da 0 (valore logico 0) a 5 [V] (valore logico 1), quindi il nostro segnle verrebbe consierato 0 sempre. Dunque, abbiamo amplificato il segnale di 100 volte, attraverso un amplificatore UA741 in configurazione non invertente. Quindi, resistenza di ingresso = 1 [Kohm] e resistenza di feedback = 100 [Kohm], ecco il circuito elettrico:



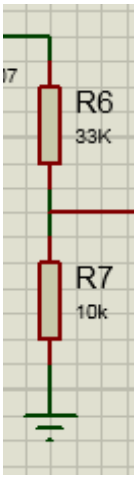
Abbiamo tenuto conto della frequenza di taglio dell'amplificatore, nettamente maggiore di quella del segnale di ingresso, ricavata attravero metodo empirico (circa 33 [Hz], l'argomento sarà trattato succesivamente).
Il segnale, adesso, varia tra 0,2 [V] e 0,4 [V].

Dopo aver amplificato il segnale, è necessario squadrarlo. Utilizziamo, quindi, un UA741, in configurazione comparatore. L'uscita dell'integrato è +15 [V] nel caso in cui il segnale di ingresso sia maggiore di quello di riferimento, altrimenti è -15 [V]. Al piedino 3 dell'integrato è applicata la $V_{encoder}$, mentre al piedino 2 è applicata la Voffset di 0,2 [V], tramite il generatore di funzioni, dato che il segnale proveniente dall'encoder non va da 0 a 0,4 [V] ma da 0,2 a 0,4 [V]. Essendo la tensione d'uscita sia positiva che negativa, è necessario raddrizzarla attraverso un diodo IN4007 (Arduino non può ricevere un segnale negativo come input). Ecco il circuito:

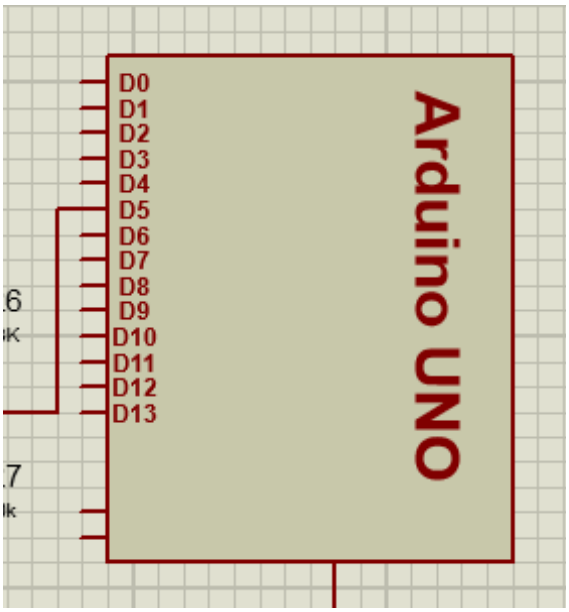


Infine, occorre ridurre la tensione da 15 [V] a 5 [V], per poter essere “accettata” da Arduino, attraverso un partitore di tensione con le resistenze pari a 10 [Kohm] e 33

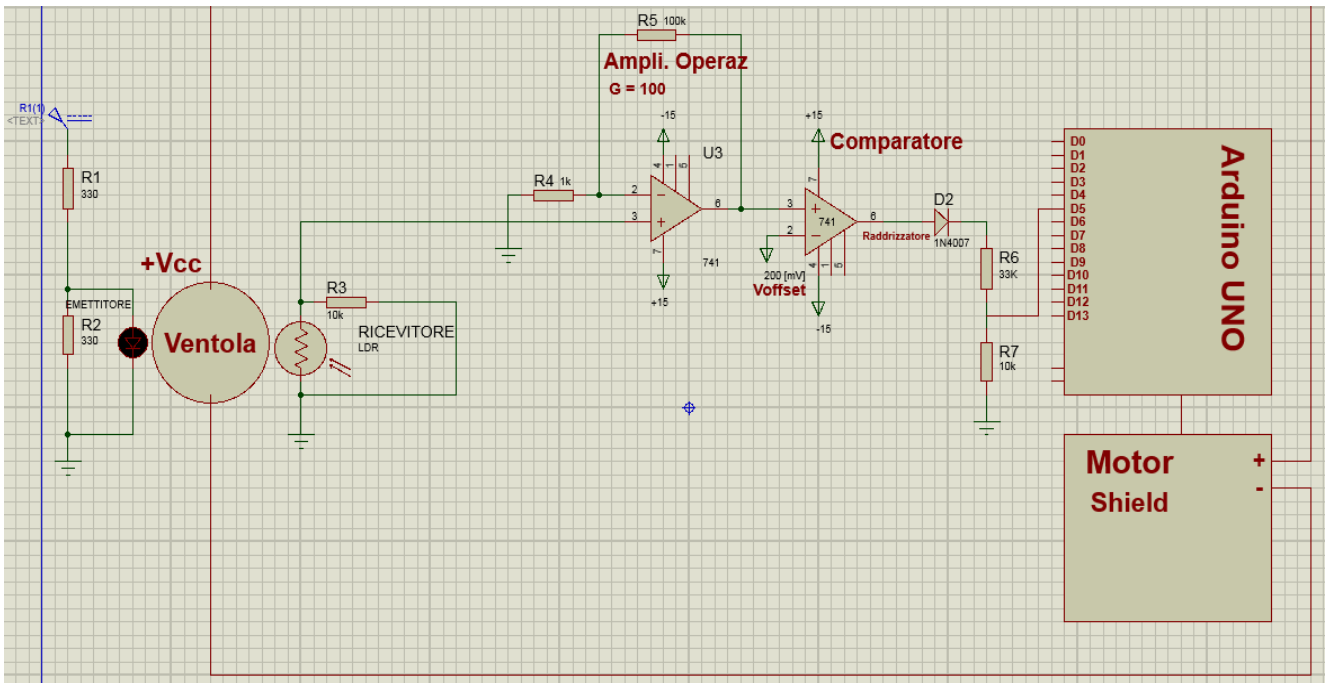
[Kohm]. Ecco il circuito:



Adesso, il segnale dell'encoder varia da 0 a 5 [V], con una frequenza che varia al cambiare della velocità. Questo segnale possiamo finalmente mandarlo ad un pin digitale di Arduino, nel nostro caso il pin 5.

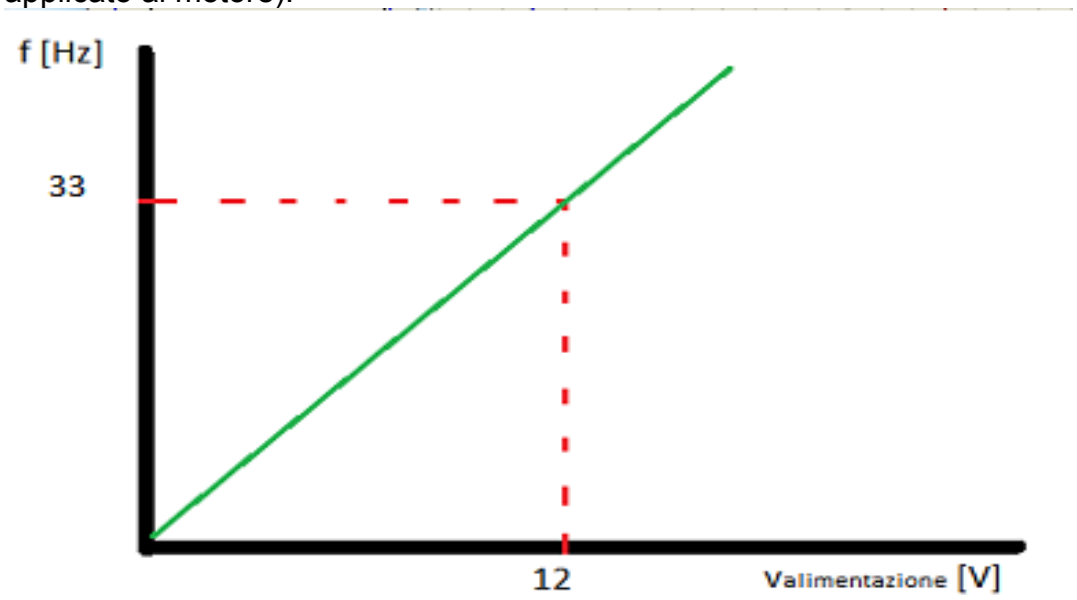


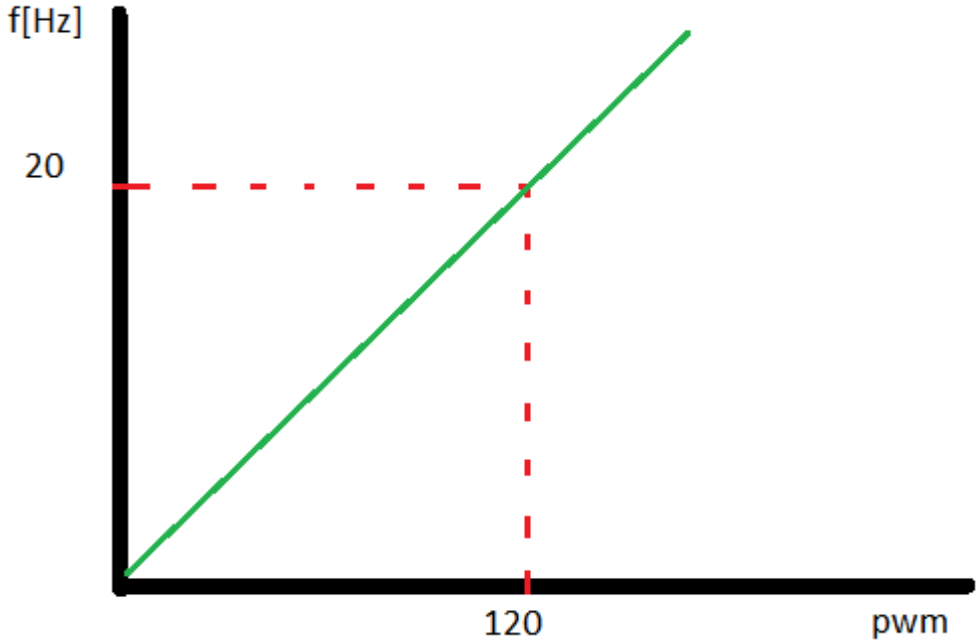
Sotto viene riportato lo schema elettrico generale:



Confronto della velocità reale con quella ideale

Per confrontare la velocità reale con quella ideale, bisogna ricavare la frequenza del segnale di ingresso, calcolando quanto tempo rimane a livello alto (5[V]) e poi a livello basso (0[V]), per poi eseguire l'inverso della somma dei due tempi. Ottenuta la frequenza, la dividiamo per il numero di pale della ventola, ossia 7. In seguito, la sottraiamo dal valore ideale di frequenza. Il valore ideale è stato scelto tracciando la caratteristica della ventola (frequenza in funzione della tensione d'alimentazione e frequenza in funzione del pwm applicato al motore):





Questi grafici sono stati tracciati grazie a misurazioni sperimentali, ossia alimentando la ventola e calcolando la sua frequenza di rotazione, mediante l'alimentatore ed il programma implementato con Arduino, e inviando con la Motor Shield un segnale pwm per osservare la frequenza di rotazione corrispondente.

In base a queste caratteristiche, abbiamo scelto un valore di frequenza pari a 20 [Hz], ossia 120 di pwm, come valore ideale da mantenere costante.

Programmazione

Attraverso l'implementazione di un programma software, non solo ricaviamo la frequenza del segnale encoder, ma sviluppiamo un regolatore proporzionale per la gestione della velocità della ventola.
La prima parte consiste nell'inizializzazione di tutte le variabili necessarie nel programma, illustrate nel commento adicente ad esse:

```
//pin per il motore
const byte dirA = 12,
           pwmA = 3;

float x = 20, //valore ideale della frequenza [Hz]
      e,      //inizializzazione della variabile per il segnale errore
      k;      //costante di proporzionalità

int vp; //variabile per la gestione del pwm del motore

unsigned long Th,          //tempo parte alta
              Tl,          //tempo parte bassa
              T;           //periodo totale
double F,                  //frequenza
      Ts;                  //variabile d'appoggio del periodo per la conversione in secondi
```

Nel setup() abbiamo impostato il pin 5 come INPUT, dato che riceverà il segnale proveniente dall'encoder, abilitato la connesione seriale Arduino-PC e fatto girare la

ventola per 3 secondi, in modo che il regolatore non agisse nella fase transitoria in cui il motore passa da velocità zero a quella ideale.

```
void setup()
{
  pinMode(5, INPUT);        //pin 13 settato come input
  Serial.begin(9600);       //inizializzazione comunicazione seriale
  digitalWrite(dirA, HIGH);
  analogWrite(pwmA, 120);
  delay(3000);
}
```

Nella prima parte del loop calcoliamo la frequenzza del segnale in ingresso al pin 5 misurando per quanto tempo si ha una tensione "alta" (5 [V]) e "bassa" (0) sul pin, attraverso la funzione pulseIn(pin, HIGH/LOW), facendo poi la somma dei due tempi ed infine l'inverso:

```
Th = pulseIn(5, HIGH); //uso la funzione "PulseIn" che mi ritorna il tempo in uS in cui il segnale rimane alto
Tl = pulseIn(5, LOW);  //uso la funzione "PulseIn" che mi ritorna il tempo in uS in cui il segnale rimane basso
T = Th + Tl;           //sommo i due tempi per ottenere il periodo totale
Ts = T /1000000.0;     //conversione da microsecondi a secondi
F = 1.0/Ts;            //calcolo Frequenza del segnale dell'encoder
F /= 7.;               //numero effettivo di giri (frequenza diviso il numero di pale)
```

Infine calcoliamo il segnale errore sottraendo dal valore di frequenza ideale quello reale e moltiplichiamo il risultato per la costante di proporzionalità, in modo da ottenere il valore del regolatore. Questo valore però è espreso in Hz, è necessario convertirlo in un valore pwm per il motore. Attraverso la caratteristica frequenza in funzione del pwm, ricaviamo che 120 : 20Hz = x : e
dove e è il segnale errore. Eseguendo i calcoli ricaviamo che è sufficiente moltplicare l'errore per 6 e poi per Kp. Il valore ottenuto va passato come parametro alla funzione analogWrite(pin, pwm):

```
k = 50;                 //costante Kp del regolatore proporzionale
e = x - F;              //calcolo del segnale errore (valore ideale meno valore reale)
vp =(e*k)*6;            //calcolo del pwm per il controllo automatico
if (vp >= 255)          //dato che il pwm può variare da 0 a 255, definiamo valori limite
  vp = 255;
else if(vp < 0)
  vp = 0;

//funzioni che permettono l'azionamento della ventola
digitalWrite(dirA, HIGH);
analogWrite(pwmA, vp);

//stampo i valori di F e del valore del pwm
Serial.print(F);
Serial.print("Hz");
Serial.print(" ");
Serial.println(vp);
```

I due if servono per impedire che il regolatore mandi un segnale pwm maggiore di 255 o minore di 0, dato che la funizione analogWrite acceta solo valori guaoli o compresi tra questi due numeri.
Programma completo:

Controllo_ventola §

```
//BONSI MAGRO D'ANGELO LIPARTITI ROSSI

//pin per il motore
const byte dirA = 12,
           pwmA = 3;

float x = 20, //valore ideale della frequenza [Hz]
      e,      //inizializzazione della variabile per il segnale errore
      k;      //costante di proporzionalità

int vp; //variabile per la gestione del pwm del motore

unsigned long Th,           //tempo parte alta
              Tl,           //tempo parte bassa
              T;            //periodo totale
double F,                   //frequenza
      Ts;                   //variabile d'appoggio del periodo per la conversione in secondi

void setup()
{
  pinMode(5, INPUT);        //pin 13 settato come input
  Serial.begin(9600);       //inizializzazione comunicazione seriale
  digitalWrite(dirA, HIGH);
  analogWrite(pwmA, 120);
  delay(3000);
}
void loop()
{
  Th = pulseIn(5, HIGH); //uso la funzione "PulseIn" che mi ritorna il tempo in uS in cui il segnale rimane alto
  Tl = pulseIn(5, LOW);  //uso la funzione "PulseIn" che mi ritorna il tempo in uS in cui il segnale rimane basso
  T = Th + Tl;           //sommo i due tempi per ottenere il periodo totale
  Ts = T /1000000.0;     //conversione da microsecondi a secondi
  F = 1.0/Ts;            //calcolo Frequenza del segnale dell'encoder
  F /= 7.;               //numero effettivo di giri (frequenza diviso il numero di pale)

  k = 50;                //costante Kp del regolatore proporzionale
  e = x - F;             //calcolo del segnale errore (valore ideale meno valore reale)
  vp =(e*k)*6;           //calcolo del pwm per il controllo automatico
  if (vp >= 255)         //dato che il pwm può variare da 0 a 255, definiamo valori limite
    vp = 255;
  else if(vp < 0)
    vp = 0;

  //funzioni che permettono l'azionamento della ventola
  digitalWrite(dirA, HIGH);
  analogWrite(pwmA, vp);

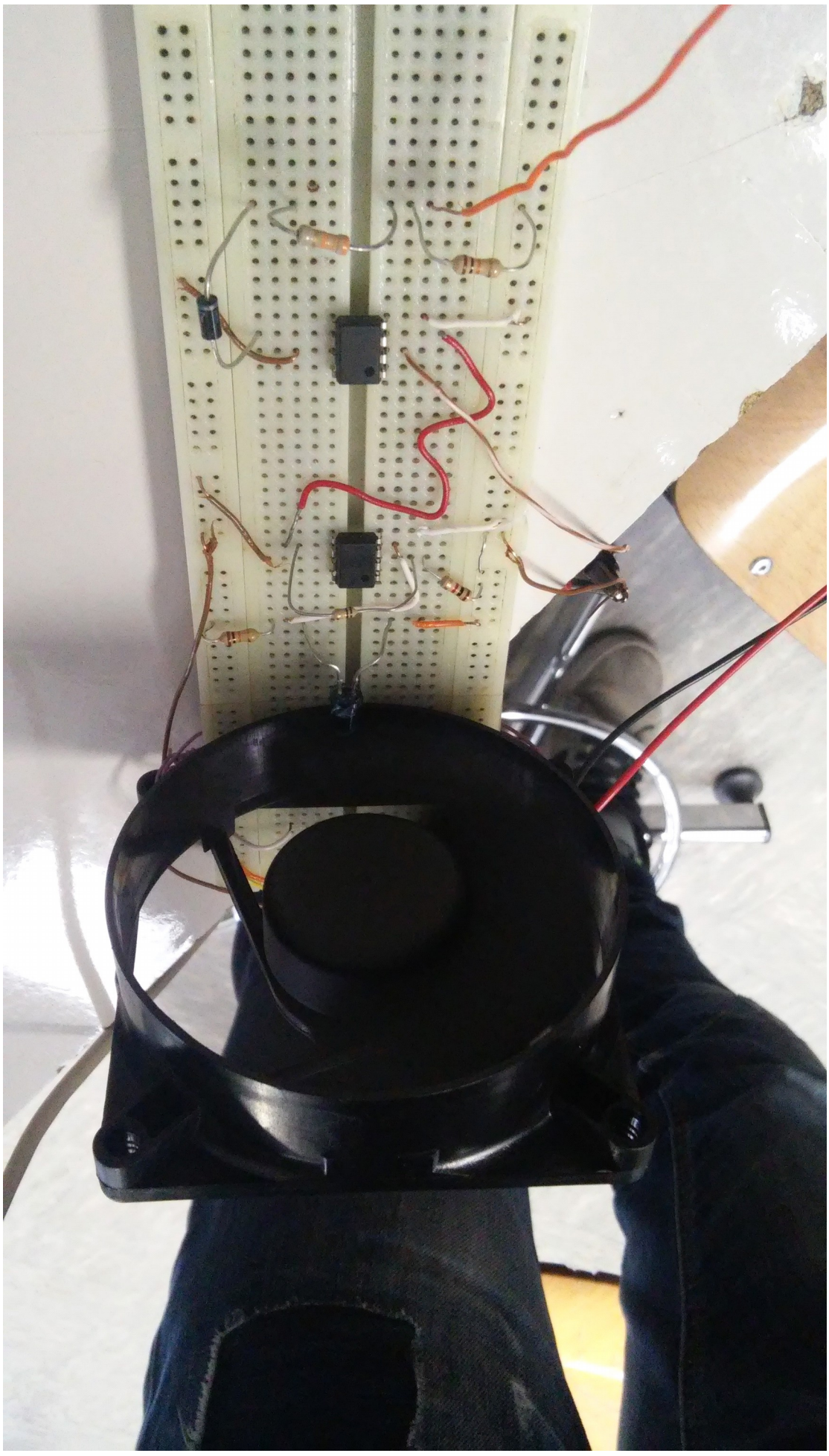
  //stampo i valori di F e del valore del pwm
  Serial.print(F);
  Serial.print("Hz");
  Serial.print(" ");
  Serial.println(vp);
}
```

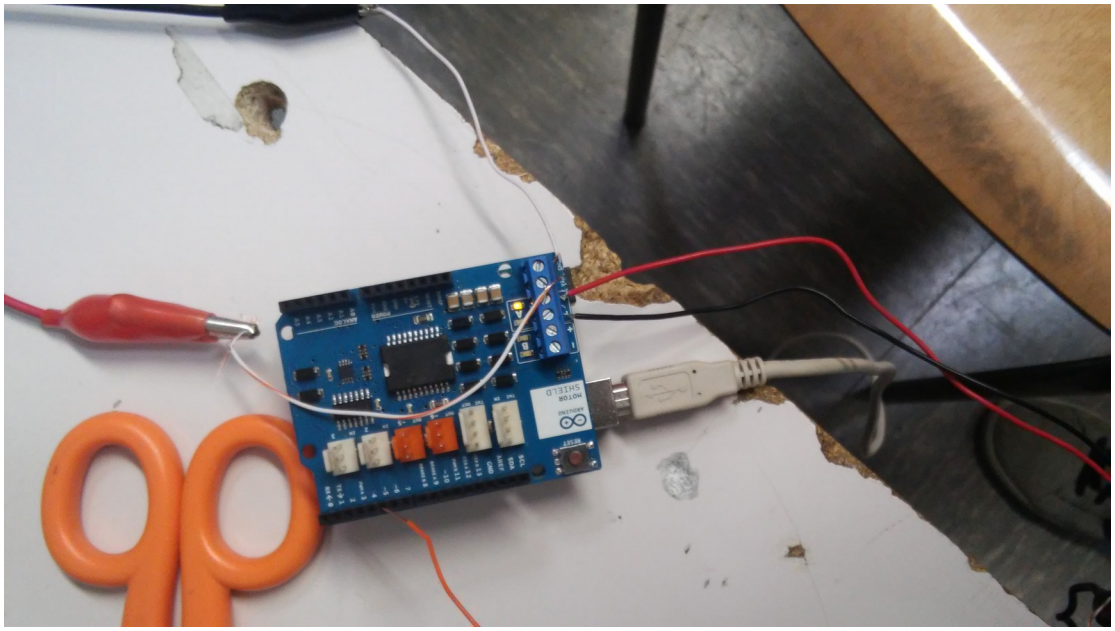
Regolazione della velocità

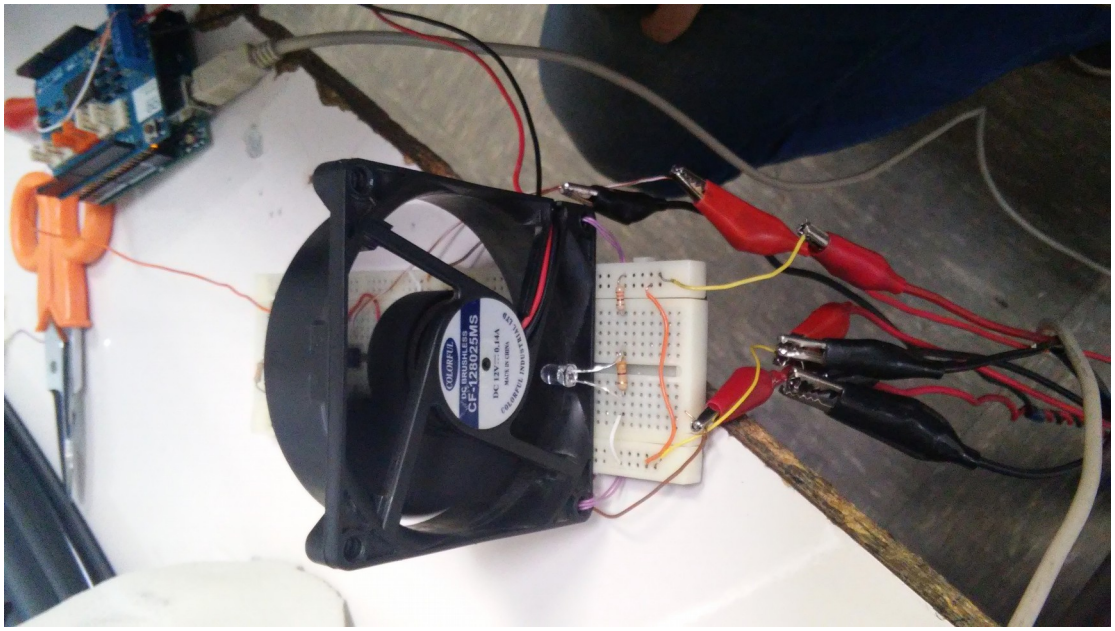
Abbiamo adesso costruito e programmato un controllo automatico della velocità di una

ventola, utilzzando un regoltore proporzionle. E' fondamentale, però, determinare il valore di Kp, punto nevralgico del regolatore, su cui si basa l'efficacia del controllo automatico. Per stimarne il valore, anche in queto caso abbiamo utilizzato il metodo sperimentale (dato che non esistono espressioni matematiche per ricavarlo). E' stato sufficiente alimentare tutti i circuiti e controllare con la comunicazione seriale la frequenza del segnale ed il suo pwm attuale, in particolare interessava osservare la risposta del motore applicando un carico (come un dito posto sul centro della ventola). E' bastato modificare il valore di Kp, fino a trovare quello con cui si riusciva ad ottenere sempre la velocità ideale.
Imprimendo, però, col dito, una pressione molto forte, che simulasse un carico elevato, il controllo non riusciva ad aumentare il pwm sufficientemente per poter ristabilire la velocità ideale. Ciò è dovuto in parte al pwm, delimitato da un valore massimo di 255, in parte alle scarse prestazioni di un regolatore composto solamente dalla da una regolazione proporzionale. Infatti, è matematicamente dimostrato che il regolatore P non è in grado di portare la velocità del sistema oltre il 50% del valore ideale. Solamente grazie all'aggiunta di un regolatore derivativo e/o integrativo si possono ottenere risultati concreti.
In conclusione, il controllo della velocità del motore è stato sviluppato pienamente, ma senza ottenere il risultato sperato.

Ecco le immagini reali dei vari circuiti:




MOTOR
SHIELD


COOLERTEK
CF-12802SMS


TOPWARD ELECTRIC INSTRUMENTS CO., LTD.
MODEL : TPS-4000
DUAL - TRACKING DC POWER SUPPLY
POWER
CV
CC
SLAVE
VOLTAGE
CURRENT
TRACKING
INDEPENDENT
MASTER
5V/3A
ITALIA INSTRUMENTS