BACK TO THE FUTURE

Allievo: Ventimiglia Luca
Tutor: Serpe Antonio

presentano...

HOVERSKATE

ITIS GALILEO FERRARIS
via A. Labriola, Lotto 20 - Scampia - Napoli

INTRODUZIONE

HoverSkate è un self-balancing (auto-bilanciante) skateboard elettrico. Fu introdotto per la prima volta nel mercato dall'imprenditore Kyle Doerksen e distribuito dalla *Future Motion* nel gennaio del 2014. Ha caratteristiche molto simili a quelle di un comune skate con la differenza di possedere soltanto una ruota posta al centro della tavola e di non necessitare dell'arcaica spinta a piede per muoversi grazie al motore di 250W di cui è provvisto. E' un mezzo di locomozione veloce ed ecologico che va avanti o indietro basandosi sulla lettura dell'inclinazione della tavola rispetto al centro della terra da parte di un giroscopio. Il pilotaggio del motore avviene attraverso il coordinamento tra una scheda di controllo Arduino e un ponte H a MOSFET.
Un oggetto utile e divertente, rivolto ad un pubblico giovanile e in grado di trasportare una persona di peso non superiore agli 80kg.
Di seguito due modelli attualmente disponibili sul mercato:

OneWheel

HoverBoard

L'idea è nata dalla seconda parte della famosa trilogia “Ritorno al futuro” del regista Robert Zemeckis quando il protagonista, Marty McFly, viaggiando nel futuro, arriva nel lontano 15 ottobre del 2015 dove prova uno skateboard a levitazione magnetica.

Una scena di "Ritorno al futuro: parte seconda"

Siamo nel 2016, ormai, ma non esistono (ancora...) dei modelli a levitazione magnetica. Iniziamo, però, a diminuire il numero di ruote da quattro ad una: è già un inizio!

MECCANICA

Forse la parte più complicata dell'intero progetto. Inizialmente, per prendere le misure, infatti, lo skateboard è stato realizzato in cartone come si può vedere in figura:

Successivamente, la forma degli assi laterali è stata del tutto modificata perché inadatta ad ospitare le batterie e la parte elettronica.
Per quanto riguarda il materiale: per le barre laterali è stato scelto del semplice compensato data la sua facile reperibilità, mentre, per le pedane, il mogano, più soggette ad usura dato che devono ospitare i piedi dello skater. Sono stati incollati anche due strati di gomma antiscivolo su entrambe le pedane che, in aggiunta, le proteggono ulteriormente.

Le pedane sono state fissate internamente alle barre laterali con delle staffette ad “L” che, però, sono risultate inadatte per ragioni di resistenza. Per questo, si è deciso di fissare esternamente le pedane con sei viti per legno cadauna.

L'asse, in alluminio, di lunghezza 22cm circa è stato acquistato presso un tornitore. Sono stati praticati due fori ai centri delle due barre laterali per ospitare le due estremità dell'asse, alle quali sono stati inseriti dei cuscinetti. L'asse non poggia direttamente sul legno ma su degli appositi supporti in alluminio fissati e incollati ai centri delle due barre laterali: questo per evitare il contatto diretto tra l'asse e il compensato.
La corona è stata fissata con 4 viti ad un supporto di forma quadrata costituito da una boccola, collegata direttamente alla ruota con una semplice vite.

RAPPORTO DI TRASMISSIONE E INGRANAGGI
Il rapporto di trasmissione scelto è stato 1/7 circa (numero denti del pignone 11, numero denti corona 80). Questa scelta ha presentato enormi vantaggi anche per la parte elettronica, in quanto il numero di giri è diminuito e la corrente necessaria per sviluppare la coppia che serve per movimentare il carico è diminuita anch'essa con un “sospiro di sollievo” per il circuito di potenza, dato che il motore “sforza” di meno.
La catena utilizzata è a passo corto con una lunghezza di 32 cm circa (se aperta).
La corona presenta un diametro di 12cm mentre il pignone ha, all'incirca, lo stesso diametro dell'albero motore a cui è fissato.

ELETTRONICA

I moduli hardware utilizzati sono stati:
- **Arduino Nano,** come scheda di controllo;
- **Driver Motori DC con BTS7960,** per il circuito di potenza;
- **Motore DC brushed**;
- **Accelerometro MPU-6050**, per la lettura dell'inclinazione;

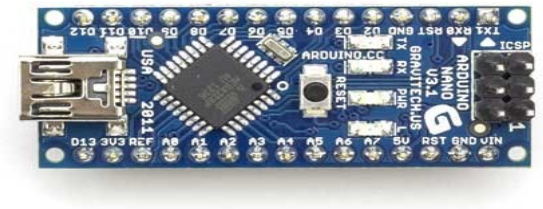
Di seguito una breve intro a ciascun componente (i datasheet possono essere consultati alla fine):

ARDUINO NANO

La Arduino Nano 3.3 è una scheda basata sul microcontrollore ATMEGA328. La scheda Arduino è dotata di 14 pin di input/output digitali (6 dei quali possono essere usati come segnali PWM), 8 input analogici, un quarzo a 16MHz, un connettore Mini-B USB, un connettore per la programmazione ICSP ed un pulsantino per il reset della scheda. La scheda inoltre fornisce tutto ciò che è necessario per supportare il funzionamento del microcontrollore.

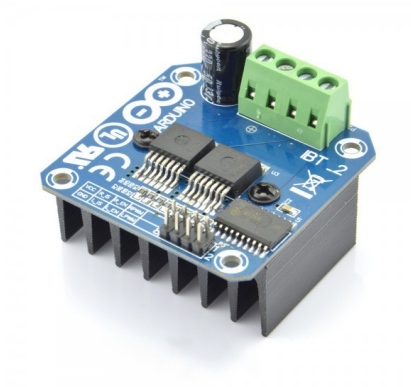
Per cominciare ad utilizzare la Arduino Nano 3.3 è semplicemente necessario connettere la scheda ad un PC tramite un cavo USB oppure fornire una tensione di alimentazione esterna non regolata compresa fra 6V e 20V collegandola al pin 30 oppure fornire una tensione di alimentazione regolata a 5V collegandola al pin 27. La sorgente di alimentazione viene selezionata automaticamente scegliendo quella che fornisce il valore di tensione maggiore.

La scheda Arduino Nano 3.3 è compatibile con molte Shield progettate per la Arduino Duemilanove o Diecimila.



DRIVER MOTORI DC CON BTS7960

Questo driver è un ponte H integrato in grado di sopportare correnti in modo continuativo fino a 43A. La tensione a cui lavora è 24V, con possibilità di effettuare un controllo PWM a 25kHz della velocità del motore e di effettuare la cosiddetta "frenata elettrica". E' possibile anche lasciare il rotore libero di ruotare attraverso lo spegnimento dei pin L_EN ed R_EN. Arduino è perfettamente interfacciabile grazie alle basse tensioni logiche necessarie per utilizzarlo.



MOTORE DC BRUSHED MY1016

Con potenza nominale 250W, tensione nominale di alimentazione 24V, corrente assorbita in condizioni nominali 13,7A, numero di giri nominali 2650 RPM. Viene utilizzato principalmente per le biciclette a pedalata assistita.

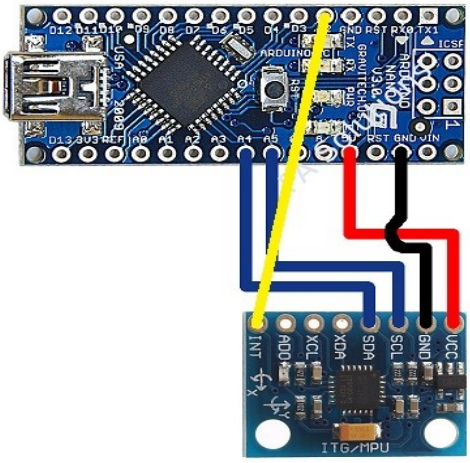
MPU-6050

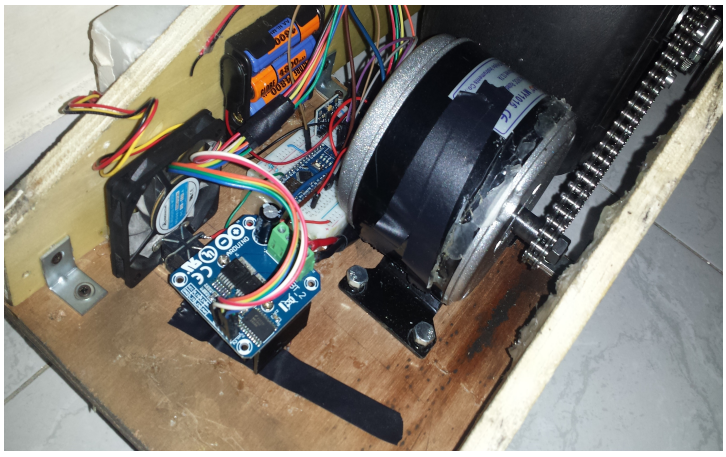
Il sensore InvenSense MPU-6050 contiene, in un singolo integrato, un accelerometro MEMS a 3 assi ed un giroscopio MEMS a 3 assi. Con lo giroscopio possiamo misurare l'accelerazione angolare di un corpo su di un proprio asse, mentre con l'accelerometro possiamo misurare l'accelerazione di un corpo lungo una direzione. È molto preciso, in quanto ha un convertitore AD (da analogico a digitale) da 16 bit per ogni canale. Perciò cattura i canali x, y e z contemporaneamente. Il sensore possiede un protocollo di comunicazione standard I2c, quindi facile da interfacciare con il mondo arduino.

Il sensore MPU-6050 non è nemmeno costoso, forse è il più economico sul mercato, soprattutto in considerazione del fatto che combina un accelerometro e un giroscopio.

CIRCUITO

Per ragioni di tempo il circuito di comando è stato realizzato su breadboard, su cui è stato collegato Arduino Nano e il sensore MPU-6050 come da schema:





La trasmissione e la ricezione dei dati avviene secondo il protocollo I2c. Di seguito alcuni cenni utili sul protocollo e di come avviene tale comunicazione(fonte *internet*):

Protocollo I2C

Il protocollo I2C (Inter Integrated Circuit) utilizza un bus di 2 fili bidirezionali, rispettivamente SDA(dati) e SCL(clock). I segnali che transitano sulle linee sono in codice binario quindi avremo 2 livelli: 1 livello alto, 0 livello basso. Le linee sono a livello alte di default, quindi collegate a resistenze di Pull Up, in modo che, quando nessuno trasmette, non si abbiano livelli di incertezza dovuti al rumore.

Questo protocollo nasce negli anni 80 e fu brevettato dalla Philips inizialmente per far comunicare dispositivi all interno di tv e pc, successivamente fu standardizzato e reso più veloce (oggi si parla di 3.4Mbit/s), ma soprattutto la Philips non ebbe più il monopolio su questo standard.

I principali dispositivi che possiamo trovare nei cataloghi sono diversi, ma ricordiamoci che con l'I2C non avremo mai comunicazioni in REAL TIME di grosse quantità di dati.

Gli attori principali della comunicazione sono 2, il MASTER e lo SLAVE. Non avremo mai 2 MASTER sulla stessa linea, eccetto casi particolari e in genere il MASTER è un microcontrollore, gli altri dispositivi sono SLAVE. Il master gestisce la tempistica e inizia la

trasmissione, lo slave gestisce le richieste del master.

Ogni dispositivo sul bus ha un indirizzo univoco a 7 o 10 bit.

Il master trasmette sulla linea data l'indirizzo dello slave sia che voglia ricevere dati, sia che li voglia trasmettere . Dopo il bit di acknowledge il master procede a trasmettere i dati, oppure li trasmette lo slave se il master lo richiede. L'Ack viene trasmesso dopo ogni byte.

E' importante osservare come la comunicazione inizi con una condizione di Start (o repeated START) e finisca con una condizione di Stop.

Una volta trasferiti i dati con il secondo byte, se vogliamo trasmettere o ricevere nuovamente, invece di stoppare la trasmissione con la condizione di STOP, ripetiamo la condizione di START. In questo modo il Master non smette mai di trasmettere sul bus e nessun altro eventuale master può prenderne il controllo. La condizione di Repeated Start nel caso in cui abbiamo un solo master è ha la stessa funzione della condizione di STOP.

In una comunicazione I2C avremo quindi tre possibili tipi di trasmissioni:

1) Il master vuole trasmettere dati allo slave

- *Il master trasmette l'indirizzo dello slave a cui vuole trasmettere su SDA*
- *Il master invia i dati che vuole trasmettere*
- *Il master termina la comunicazione*

In questo caso è lo slave che genera il segnale di ack, quando un segnale di NOT ACKNOWLEDGE viene ricevuto il master dovrà generare una condizione di STOP o di REPEATED START.

2) Il master vuole ricevere dati dallo slave

- *Il master trasmette l'indirizzo dello slave da cui vuole ricevere dati su SDA*
- *Lo slave trasmette i dati al master*
- *Il master termina il trasferimento*

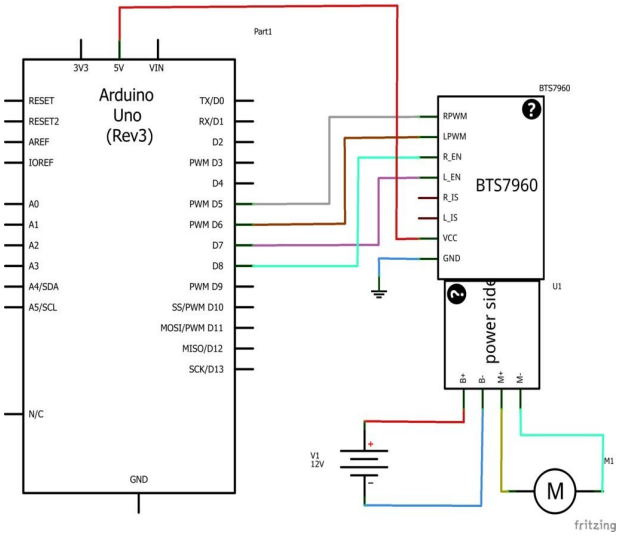
In questo caso il master genera l'ACK dopo aver ricevuto i dati, ma per indicare allo slave che non ha più bisogno di dati genera un NOT ACKNOLEDGE.

3) Comunicazione mista (ogni parte di questa comunicazione è scomponibile in uno dei due casi precedenti).

Nei casi sopracitati possiamo osservare che la comunicazione da master a slave è gestita completamente dal master e a seconda di come poniamo il bit di indirizzo possiamo leggere o scrivere dati nello slave.

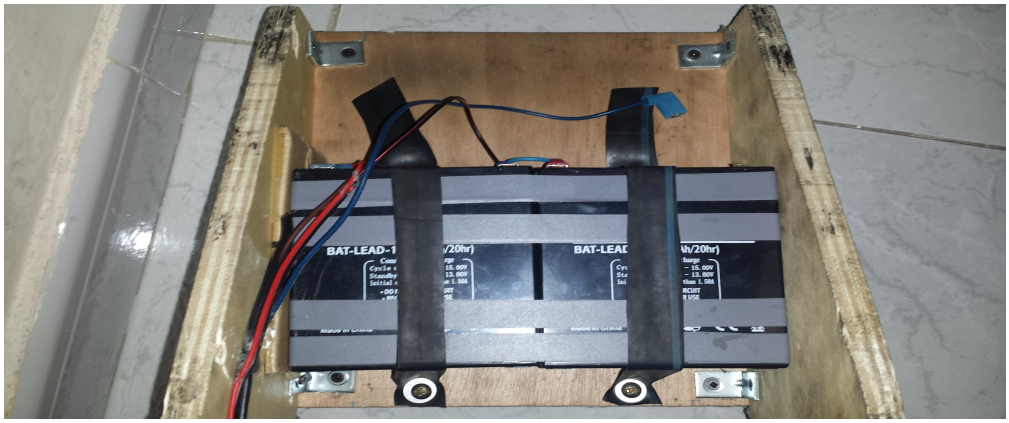
Altra fonte- *Il pin Int ("interrupt") avverte Arduino che i dati grezzi sono arrivati nel registro il quale può essere letto dal controllore.*

Il driver per il controllo dei motori è stato collegato come da schema utilizzando dei cavetti maschio-femmina*:



**Considerando nello schema che è stato utilizzato un Arduino Nano e che l'alimentazione del motore è di 24V.*

Le **batterie** utilizzate sono due comuni batterie al piombo da 12V collegate in serie per ottenere la 24V necessaria per alimentare il motore, ognuna con una capacità di 5Ah. Considerando un assorbimento di circa 14A continui, queste batterie durano, all'incirca 20 min (questo calcolo tuttavia è, di parecchio, peggiorativo, dato che la corrente che assorbe il motore è molto minore del valore su indicato). Per evitare complicazioni, si è scelto di separare l'alimentazione del ciruito di potenza da quella di comando: per alimentare Arduino è stato utilizzato un pacco batterie con quattro pile stilo ricaricabili collegate in serie da 1,2V ciascuna.

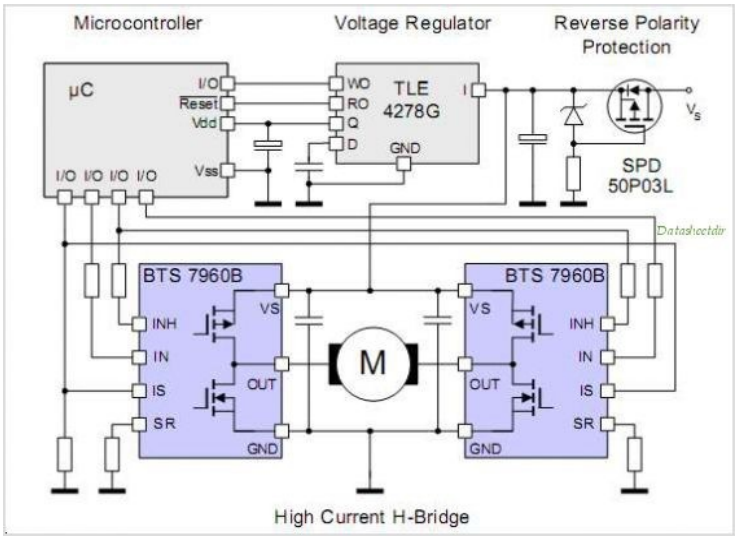


COME FUNZIONA UN PONTE H?

Un **ponte H** serve, essenzialmente, per **regolare la velocità e il verso di un motore** che richiede correnti troppo elevate per poter essere erogate direttamente da una scheda di controllo come Arduino (che al massimo può fornire 40mA in uscita dalle sue porte logiche).

E' un circuito elettronico formato, nella sua forma più semplice, da 4 transistor di cui due di tipo pnp e due di tipo npn (se sono MOSFET due saranno p-channel e due n-channel).

Il driver utilizzato sfrutta due mezzi ponte H, i BTS7960, che collegati opportunamente insieme costituiscono un ponte H completo. Riporto un esempio di applicazione:



BT7960: mezzo ponte H

L'idea iniziale era quella di realizzare un ponte H tutto artigianale per controllare il motore
Un ponte H del genere, tuttavia, non è di facile realizzazione.

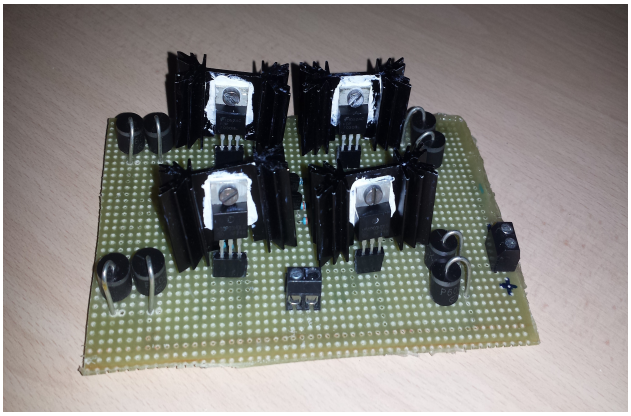
Il primo problema è stato trovare dei MOS Logic Level, che potevano essere pilotati facilmente con le tensioni logiche di Arduino e che potessero drenare le elevate correnti in gioco.

La mia attenzione è ricaduta su due componenti:

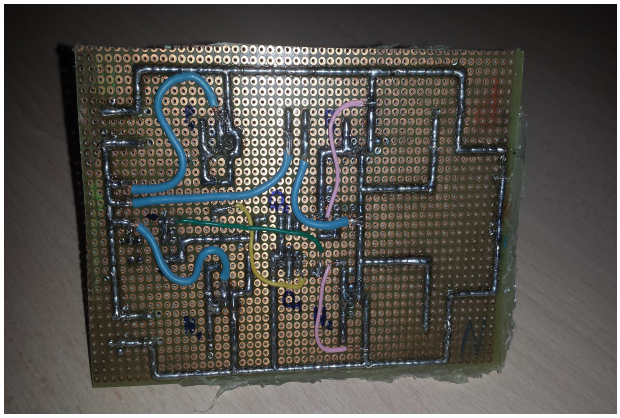
- **FDP7030BL** come n-channel;
- **MTP50P03HDLG** come p-channel.

Se si osservano i grafici della tensione tra gate e source Vgs e della corrente di drain Id, ci si rende subito conto che essi sono in grado di pilotare correnti molto elevate e con tensioni dell'ordine dei 4/5V. Per pilotare i p-channel, tuttavia, è stato realizzato un doppio stadio con dei 2n2222 perchè per interdirli è necessario portare la tensione Vgs a -0.6/-0.7V e questo significava che sul gate ci doveva essere un potenziale molto elevato, di circa 23V (se si considera il potenziale del drain di 24V) che Arduino, da solo, non era in grado di fornire. Nel circuito sono state inserite anche delle resistenze di polarizzazione per evitare la conduzione e lo spegnimento involontario rispettivamente dei p-channel e degli n-channel. Sono stati, inoltre, aggiunti dei dissipatori, uno per ogni MOS, con tanto di pasta termica per migliorare lo smaltimento del calore.

Ecco la scheda terminata:

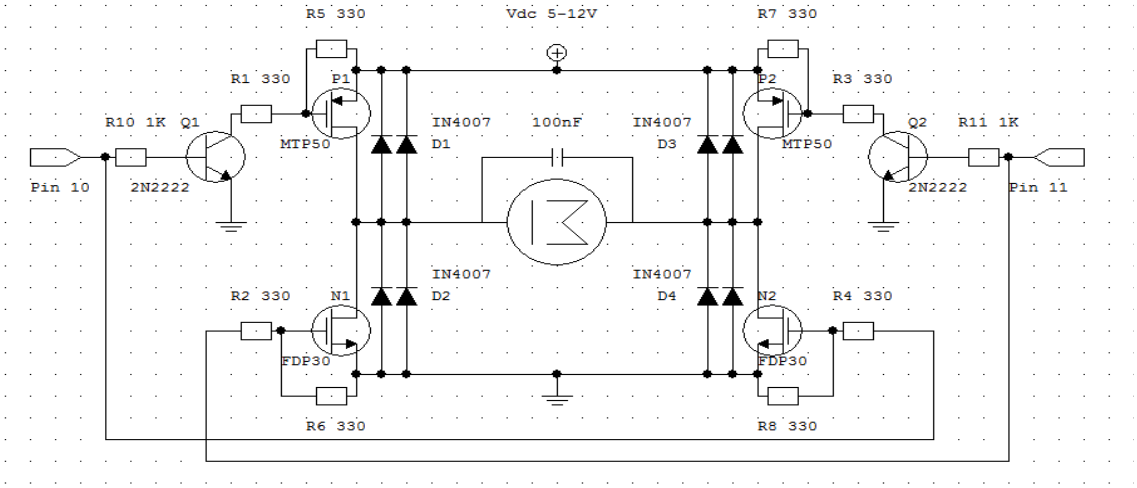


Fronte



Retro

Qui lo schema elettrico*:



**Con tensione di alimentazione 24V*

Sebbene il ponte H funzioni, per ragioni di smaltimento del calore e di isolamento degli ingressi, si è scelto di utilizzare il BTS che garantisce al 100% entrambe le cose.

SOFTWARE

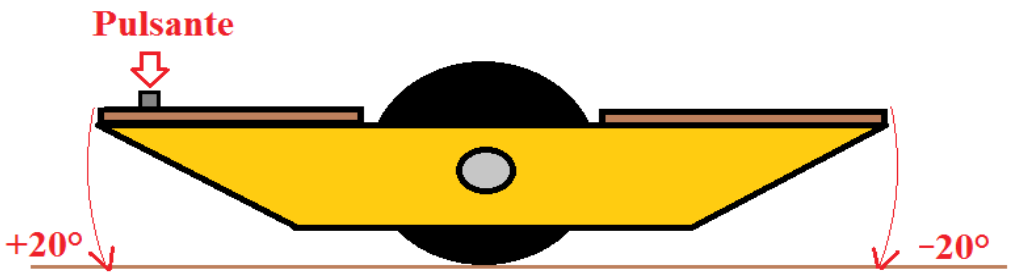
Come primo step, è stato necessario interfacciare l' MPU5060 ad Arduino, interpretando i dati "grezzi" del sensore in gradi di inclinazione (l'inclinazione che ci interessa sarà il cosiddetto "rollio"). L'interpretazione dei dati è stata realizzata sfruttando la libreria "I2Cdev", scritta e realizzata da Jeff Rowberg**,** che, sfruttando quaternioni e formule matematiche complesse, restituisce i gradi di inclinazione con una precisione del centesimo di grado.
L'utilizzo della libreria I2Cdev si è rivelata particolarmente vincente, in quanto, i dati "grezzi" provenienti dal sensore che variavano da 0/16000 da un lato e da 0/-16000 dall'altro lato erano praticamente ingestibili data la velocità con cui variavano.
Per ragioni di lunghezza del codice, dovuta, principalmente, alla libreria "I2Cdev", si riporta in seguito solo la parte di codice che sfrutta il segnale proveniente dal sensore per monitorare il motore. I commenti chiariranno il significato di ogni riga di codice:

```
void loop() {
  int vel; // variabile contenente il segnale PWM da inviare al ponte H che va da 0-255
  int buttonState=digitalRead(button); // variabile che legge lo stato del pulsante ON/OFF
  float y=ypr[2] * 180/M_PI; //variabile contenente il valore dell'angolo di pitch/rollio
  Serial.print("                    y=   "); //stampa y sul monitor seriale
  Serial.println(y);
  Serial.print("VEL=  "); //stampa vel sul monitor seriale
  Serial.println(vel);

  digitalWrite(R_EN,HIGH); // Abilita il motore a girare in senso orario
  digitalWrite(L_EN,HIGH);// Abilita il motore a girare in senso antiorario

     if(buttonState==1) { //se il pulsante è premuto...
     if(y>=-0.9f) { // e il valore dell'angolo di pitch è maggiore o uguale di un certo grado
       vel = map(int(y * 10), -9, 199.3, 0, 255); // fai la proporzione tra inclinazione e velocità del motore
       analogWrite(LPWM,vel); // scrivi il valore di velocità per la rotazione antioraria
       }
       else analogWrite(LPWM,0); //altrimenti porta la velocità a 0
    if(y<-1.0f) {// e il valore dell'angolo di pitch è minore di un certo grado
        vel = map(int(y * 10), -10, -201.0, 0, 255);// fai la proporzione tra inclinazione e velocità del motore
       analogWrite(RPWM,vel);// scrivi il valore di velocità per la rotazione oraria
       }
       else analogWrite(RPWM,0); //altrimenti porta la velocità a 0
       }
       else  {digitalWrite(R_EN,LOW); digitalWrite(L_EN,LOW);} //altrimenti disabilita il motore a girare in ambo i sensi
```

Inizialmente, avevo optato per un controllo PID ma, nonostante fossi riuscito ad implementare nel programma anche un controllo integrativo e derivativo, trovare i valori di Kp, Ki e Kd si è rivelato più difficile del previsto. Per questo, ho scelto di implementare un controllo che fosse solo proporzionale. Sfruttando la comoda funzione map() di Arduino, che fa la proporzione automaticamente, è stato possibile "rimappare", appunto, i valori di inclinazione provenienti dal sensore in valori di velocità per il motore.



Ad esempio, se, tenendo premuto il pulsante, mi inclino di -20° (inclinazione massima di esempio),

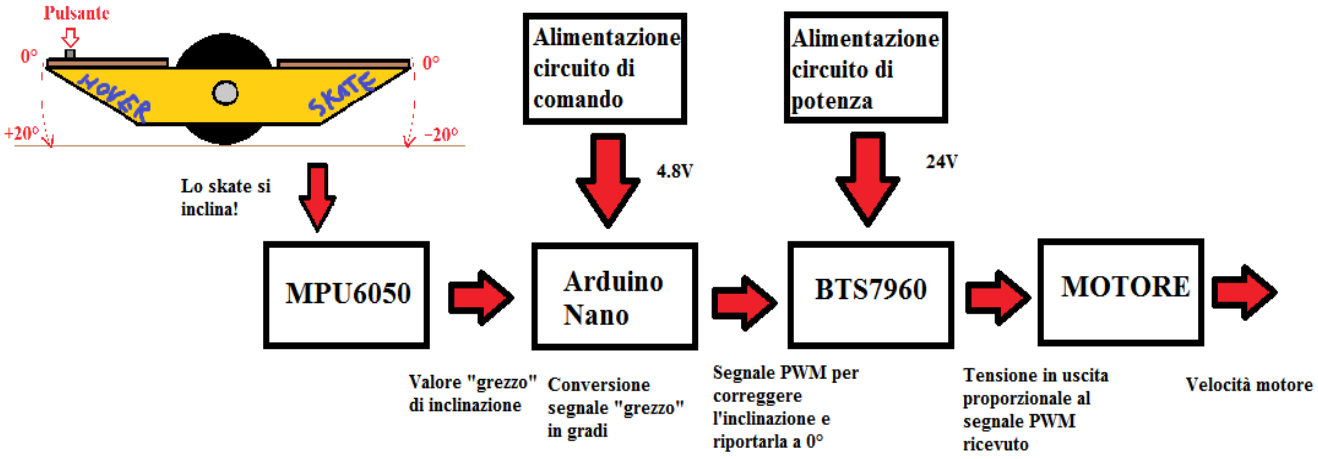
Arduino invierà in PWM il valore 255 al ponte H (corrispondente al valore massimo di velocità per il motore) e il motore girerà in senso antiorario per cercare di riportare lo skate a 0° di inclinazione.

Siccome, però, questa funzione lavora con numeri interi e siccome la variabile y è un float (numero con la virgola), è stato necessario convertire il valore di y in int:

int(y*10)

come si può notare nella funzione map().

SCHEMA A BLOCCHI



CONCLUSIONE

Lo scopo del progetto è realizzare qualcosa di nuovo, divertente, creativo e, per chi ha particolare destrezza, anche utile. Mi piace l'idea che le persone possano interagire con le mie creazioni. Purtroppo, questo, non è un mezzo così semplice da portare, considerando anche la sua artigianalità, e richiede una vera e propria preparazione atletica oltre che uno sviluppato senso dell'equilibrio. Andare su un **HoverSkate** è come “surfare”: il problema, però, è imparare a guidarlo!