Gulliver: A self balancing robot

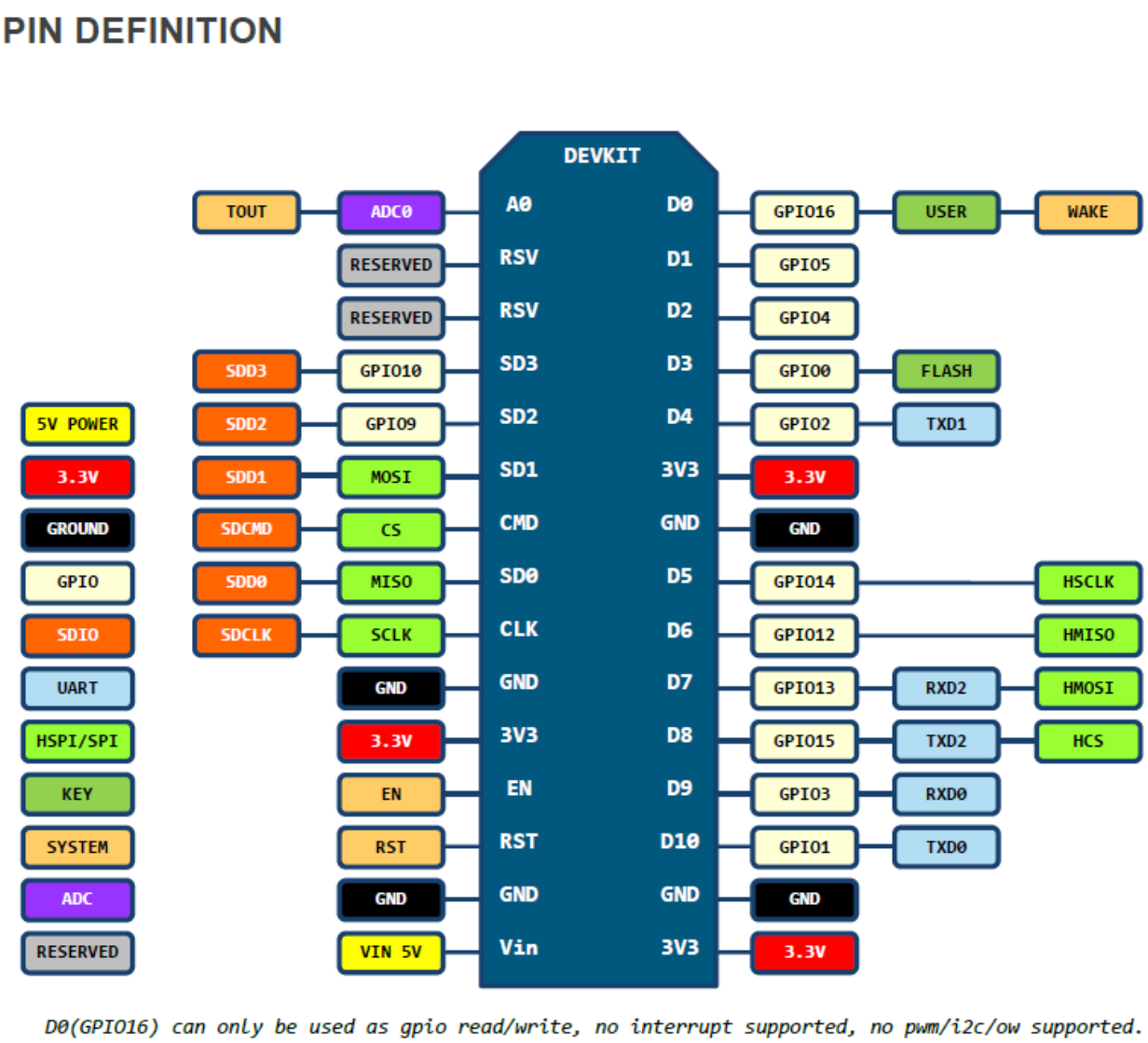
Ciao a tutti, io mi chiamo Davide e sono uno studente della università di Salerno, oggi voglio presentarvi un progetto già visto ma basato interamente su un microcontrollore molto economico, ovvero lo ESP8266.

Lo ESP8266 è un particolare microcontrollore prodotto dalla expressiff che è più economico di un clone di arduino uno, basato sul processore Tensilica L106 32-bit in grado di lavorare fino a 160mhz. La particolarità di questo microcontrollore è quella di avere incorporato un antenna WI-FI ed annessi protocolli per la connessione 802.11 b/g/n già presenti nel sistema operativo di base forniti da expressiff la quale fornisce una libreria per la programmazione di questo microcontrollore.

Tuttavia grazie ad alcuni utenti è possibile programmare lo ESP8266 utilizzando l'IDE di arduino, sfruttando la sintassi già conosciuta con qualche semplice libreria aggiuntiva creata ad hoc per sfruttare il microcontrollore al meglio.

Per chi fosse interessato è possibile programmare questo microcontrollore usando anche il linguaggio LUA. Esistono molte versioni di questo microcontrollore che possiedono il convertitore seriale per la programmazione, quello usato per questo progetto è il NodeMCU 1.0 (ESP-12E Module) ovvero con il convertitore cp2102, provvisto anche di uno stabilizzatore di tensione interno a 3,3V.

Nonostante i numerosi pin, quelli utilizzabili come GPIO sono solo 11.

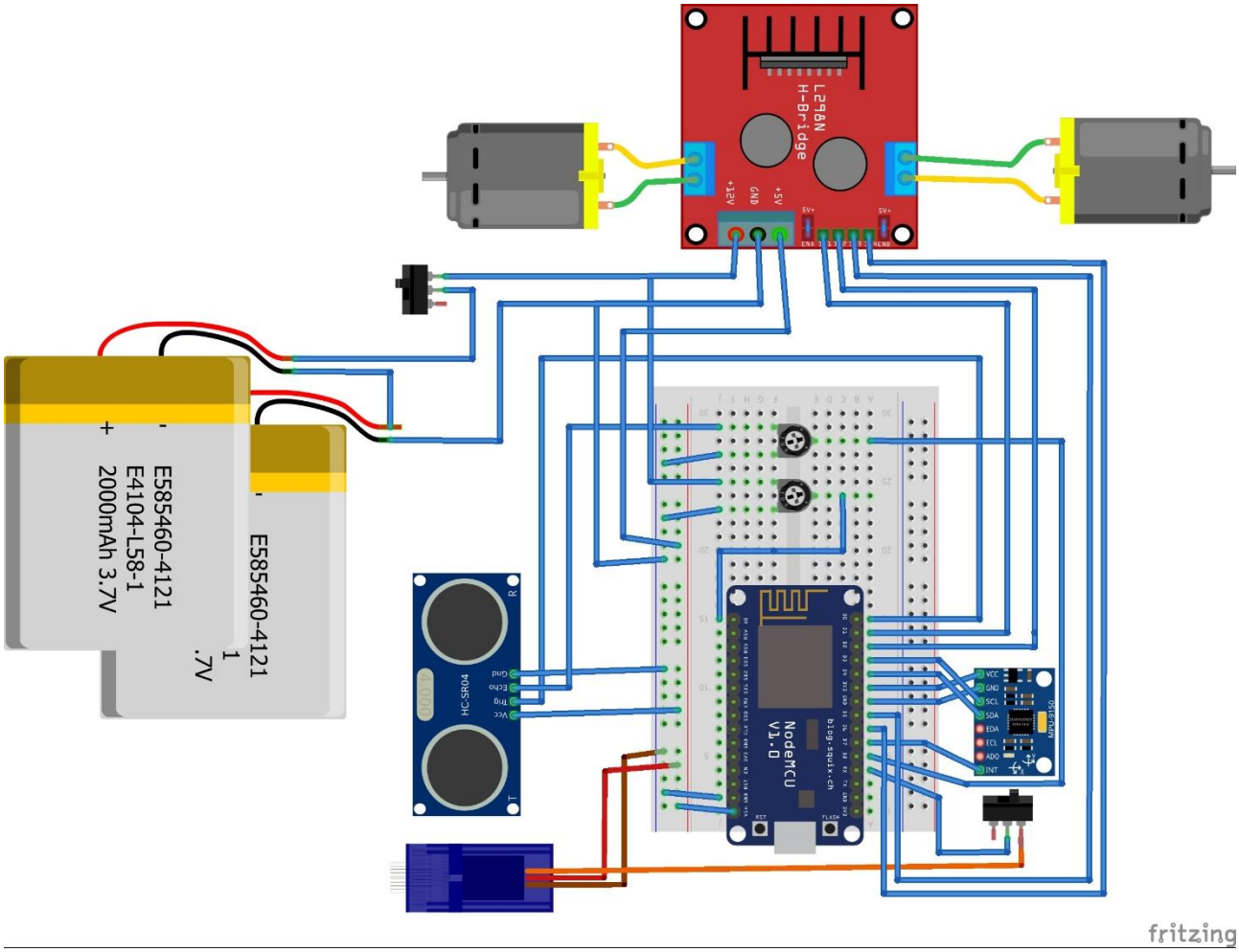


Ovvero dal D0 al D10 e A0 che ha solo funzione di ADC 10 bit.

I pin D9 e D10 sono quelli usati di default per la comunicazione seriale con il convertitore, perciò in caso li si voglia usare per altri scopi come per il PWM input-output etc bisogna ricordarsi di scollegare il dispositivo a cui sono connessi durante la programmazione, inoltre sono fondamentali nel caso si voglia usare un terminale seriale per scrivere o leggere eventuali messaggi di errore, per cui in tal caso non possono essere utilizzati, inoltre tutti i pin non possono superare i 3.3V in ingresso.

Il circuito del self balancing robot da me realizzato sfrutta 10 pin su 11.

Schema:



Lista dei componenti:

NodeMCU (12E module)

HC-SR04

Sevo motore da 9g SG90

MPU9250

2 interruttori

Trimmer 2.2Kohm

Trimmer 4.7Kohm

Interfaccia di potenza basato su un modulo pre-assemblato che usa un L298

2 motori

Batteria lipo a 2S (>1000mAh)

Breadboard da 400 punti

Struttura:

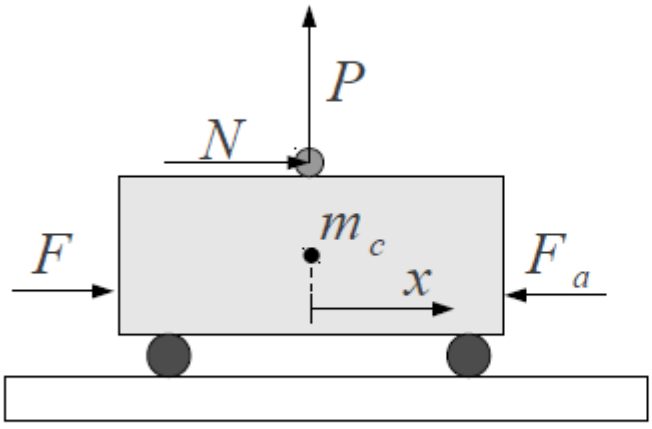
La struttura del self balancing robot è stata progettata in 123D Design e successivamente stampata con una stampante 3D

L'idea:

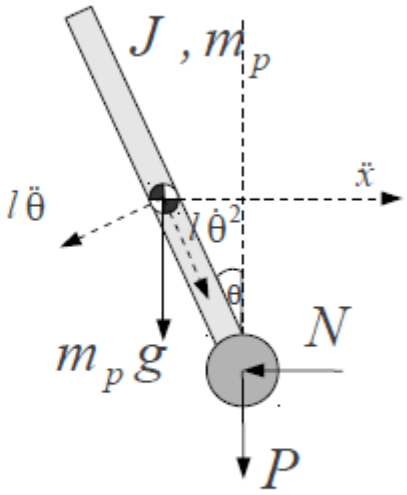
La mia idea era quella di realizzare un self balancing robot, interamente basato sul ESP8266 senza l'ausilio di microcontrollori esterni, che fosse in grado di evitare ostacoli e che potesse essere controllato da remoto tramite il WI-FI.

Il principio fisico alla base del self balancing robot è il pendolo inverso.

Nel caso di un pendolo, grazie alla forza di gravità che ci è amica porta il pendolo a fermarsi esattamente al centro mentre nel caso del pendolo inverso, la gravità è il nostro peggior nemico in quanto dobbiamo contrastarla spostando la base del pendolo, nei limiti del sistema che ci permette di raggiungere tale scopo ovvero i motori + giroscopio.



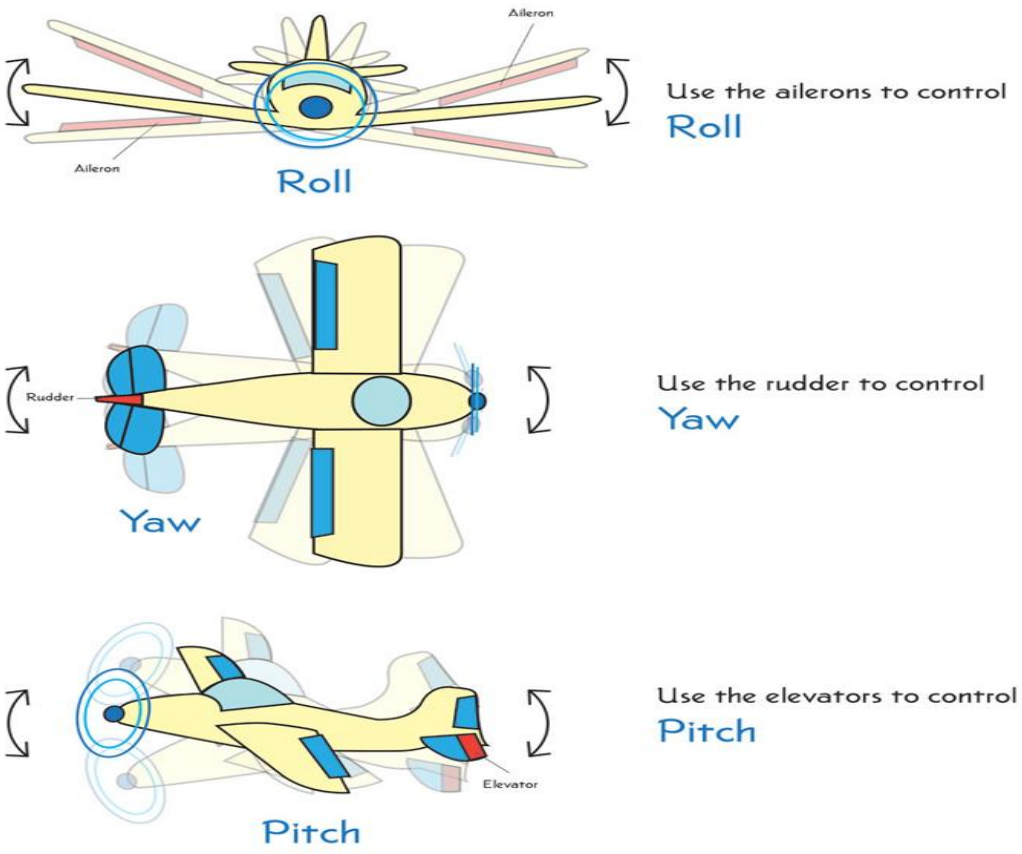
(a)



(b)

Principio di funzionamento:

Quindi principio di funzionamento del robot è molto semplice, il giroscopio usato ovvero lo MPU9250 è a 9 gradi libertà in grado di fornire 3 assi per l'accelerometro, 3 assi per il giroscopio, 3 assi per il magnetometro ed è inoltre in grado di misurare la temperatura dell'ambiente circostante, tutti valori letti sono memorizzati come raw ovvero un numero a 16 bit, che però possono essere convertiti in unità di misura a noi note, la scala di sensibilità del accelerometro può andare da -2/+2 g fino a +16/-16 g la scala del giroscopio da +250/-250 deg/s fino a +2000/-2000 deg/s mentre il magnetometro ha una sensibilità di +4800/-4800 uT, combinando le letture del accelerometro e del giroscopio possiamo calcolare il quaternione che è costituito da quattro variabili che contengono la rotazione del oggetto rispetto ai tre assi con una rappresentazione matematicamente più semplice, il quale poi va in pasto ad un filtro che ci restituisce il posizionamento spaziale statico, ovvero la rotazione del oggetto in gradi rispetto ai 3 assi, questi valori sono normalmente definiti come pitch, roll e yaw. Tuttavia questo particolare MPU(motion processing unit) ci permette di leggere direttamente il quaternione grazie al DMP(dynamic motion processing) incorporato evitando sfasamenti di posizione dovuti a problemi di sincronizzazione all'interno del codice. Noi siamo interessati solo ad uno, ovvero quello relativo al pitch.



Successivamente questo valore viene dato in ingresso al PID, il quale ci permette grazie alla regolazione di tre costanti una proporzionale P, una integrativa I e una derivativa D di controllare i motori affinchè questi possano mantenere in equilibrio il robot.

La variabile P si occupa di moltiplicare per un valore costante l' ingresso letto, in questo caso i gradi misurati dal MPU, fornendo in uscita un valore che poi sarà inviato al comando analogWrite, tuttavia il solo valore proporzionale non è in grado di rendere il sistema affidabile in quanto questo non tiene conto di variabili come il peso, velocità di reazione, insomma tutte le variabili reali che il sistema deve gestire, per questa ragione si aggiunge una costante derivativa che ci permette di gestire la velocità di reazione del sistema al variare del suo input, inoltre la correzione ottimale viene fatta aggiungendo la variabile integrativa che ci permette di aggiustare il posizionamento per la raggiunta di un equilibrio ottimale, a questo è stato aggiunto un algoritmo di windup per evitare che il valore integrativo impatti le performance del sistema quando i valori in output raggiungono la saturazione, ovvero la potenza massima.

Può bastare la regolazione del PID?:

In teoria potrebbe bastare se il sistema è leggero e si ha calibrato bene il punto di centro, ma nel caso del mio progetto, essendo basato su motori molto economici, e pesando intorno ai 600 grammi la semplice regolazione del PID non è bastata. Per far fronte a questo problema c'erano due strade che potevo intraprendere, o aggiungevo un sistema per rilevare la velocità che poi sarebbe andato in ingresso ad un secondo PID che avrebbe variato il baricentro del robot dinamicamente, strada che non ho intrapreso perché avevo esaurito tutti gli ingressi a mia disposizione a meno che avessi voluto aggiungere un altro microcontrollore, così ho analizzato il sistema e come ho scritto poc'anzi il problema era relativo al fatto che il baricentro del robot si spostava e quindi ho trovato un modo alternativo seppur limitato per farlo mantenere in equilibro, nella pratica ho misurato l'angolo raggiunto prima del punto di equilibrio, poi l'angolo raggiunto dopo il punto di equilibrio, ne ho fatto la differenza e l'ho divisa per due, questo piccolo calcolo è il nuovo baricentro del robot e di conseguenza ho potuto regolare il PID con i parametri giusti.

Cosa fa in più questo self balancing robot?

Il self balancing robot da me progettato è in grado di gestire un servo, il quale fa ruotare un sensore di distanza, se rileva un ostacolo fa ruotare il robot nella posizione senza ostacoli, inoltre lo ESP8266 è stato configurato come access point al quale possiamo collegarci, è possibile comunicare con il robot grazie al protocollo UDP con il quale impostare tutti i parametri necessari per la calibrazione ed inoltre possiamo comandare il robot a distanza facendolo andare avanti, indietro a destra o a sinistra.

Al momento per poter comunicare con il robot bisogna usare un terminale UDP, tuttavia sto sviluppando una applicazione android apposita che sarà disponibile sul mio link di git-hub insieme ai file STL per stampare la struttura del robot.

GIT-HUB LINK:

https://github.com/Dave4675/MPU-9250DMP