UNIVERSITÀ
degli STUDI
di CATANIA

DIPARTIMENTO DI INGEGNERIA
ELETTRICA ELETTRONICA E INFORMATICA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Giuseppe Emanuele Di Franco

IMPLEMENTAZIONE DI UN CONTROLLO PID PER SELF-BALANCING ROBOT

Relatore:
Chiar.mo Prof. Ing. Giuseppe Sutera

Anno Accademico 2024/2025

INDICE

ABSTRACT

Il presente lavoro di tesi illustra lo studio, la progettazione e lo sviluppo di un sistema di controllo per robot auto-bilanciante su due ruote. Tali sistemi, noti per la loro instabilità intrinseca, richiedono algoritmi di controllo avanzati per mantenere l'equilibrio e garantire un funzionamento sicuro ed efficiente.

La ricerca si concentra sull'implementazione di un controllore PID al fine di ottimizzare la stabilità e la risposta dinamica del robot.

Per la realizzazione del prototipo è stato adottato un approccio basato sull'utilizzo di componenti hardware facilmente reperibili al fine di garantire un sistema funzionale e completamente open source. L'utilizzo dell'ambiente di sviluppo *Arduino IDE* (Integrated Development Environment) ha permesso una programmazione intuitiva e flessibile, facilitando l'integrazione e la gestione dei dispositivi hardware in modo tale da rendere il progetto accessibile a un'ampia comunità di sviluppatori.

Il controllo è validato attraverso simulazioni e test sperimentali.

INTRODUZIONE

Un Self-Balancing Robot è un tipo di robot progettato per mantenere il proprio equilibrio autonomamente, nonostante sia in movimento o in condizioni di generica instabilità [1]. Attualmente, esempi di Self-Balancing Robot si ritrovano nei mezzi di trasporto personale, come Segway e Hoverboard.

(a) (b)

Figura 1. Esempi di Self-Balancing Robot: (a) Segway, (b) Hoverboard

Un robot di questo genere si basa su principi di controllo dinamico e feedback per stabilizzare la propria posizione. La sfida principale è quella di bilanciarsi su una base instabile. Questo tipo di comportamento richiede una continua correzione dell'orientamento rispetto alla forza di gravità e altre forze esterne, sfruttando sensori per rilevare l'inclinazione e attuatori per correggerla in tempo reale. La stabilizzazione è realizzata attraverso algoritmi di controllo che interpretano i dati sensoriali e attivano i motori per modificare l'inclinazione del robot, mentendolo così in equilibrio.

L'obiettivo di questa tesi è quello di progettare e realizzare un prototipo di Sel-Balancing Robot a cui applicare un controllo di tipo PID (Proportional, Integrative, Derivative) per la stabilità, mediante l'integrazione di un sensore inerziale IMU (Inertial Measurement Unit) - il quale sfrutta due sensori interni (accelerometro e giroscopio) per il calcolo dell'angolo di inclinazione - con un microcontrollore e due attuatori. Inoltre, è stato previsto l'inserimento di due sensori di distanza al fine di salvaguardare il dispositivo da eventuali ostacoli.

L’approccio proposto nella realizzazione di questo progetto prevede la ricerca e assemblaggio delle componenti meccaniche ed elettroniche, lo studio del sistema pendolo inverso, ovvero la rappresentazione ideale del modello fisico del robot. Allo studio fisico seguirà la realizzazione del controllo PID e, infine, i test sulla stabilità e sul movimento del prototipo.

Il presente lavoro è così strutturato: nel primo capitolo verrà proposto lo studio del sistema fisico e teorico del modello utilizzato per la realizzazione del controllo; nel secondo capitolo sarà fornita una panoramica strutturale del prototipo e le componenti hardware utilizzate; il terzo capitolo sarà dedicato alla spiegazione del codice implementato per le funzioni principali; nel capitolo 4 seguiranno gli esperimenti condotti che definiranno l’effettivo funzionamento del robot. Infine, sarà riservato uno spazio riguardante le conclusioni dedotte e saranno offerti spunti per miglioramenti futuri.

STATO DELL'ARTE

I Self-Balancing Robot [1] rappresentano una classe di robot mobili che utilizzano controlli in tempo reale per mantenere l'equilibrio dinamico su due ruote. Questi sistemi basano il loro fondamento teorico sul comportamento di un pendolo inverso su base mobile [2] a causa della capacità di quest'ultimo nel simulare instabilità e per la semplicità con cui può essere controllato in tempo reale.

Inoltre, questo genere di dispositivi ricerca la propria stabilità attraverso un'accurata rilevazione del proprio angolo di inclinazione rispetto alla verticale. In [3] vengono forniti dei metodi per il calcolo dell'angolo attraverso un sensore IMU MPU6050 e, in particolar modo, sono messi in risalto due modalità di calcolo: il DMP (Digital Motion Processor) [8] e il filtro complementare [9]. Dall'analisi qualitativa dei risultati, l'autore specifica come il DMP fornisce stime dell'orientamento più rapide e reattive, con una latenza minore rispetto al filtro complementare ma, allo stesso tempo, il filtro garantisce stime accurate per gli angoli di beccheggio (pitch) e rollio (roll). Alla luce di queste analisi, per il progetto di tesi, è stato scelto di utilizzare il filtro complementare in modo tale da avere maggior precisione al costo di una risposta leggermente più lenta.

In un Self-Balancing Robot gioca un ruolo fondamentale il sistema di locomozione il quale, nel prototipo, è composto da motoriduttori DC, driver a ponte H e regolatore di tensione. In particolare, l'utilizzo dei motoriduttori garantisce un'elevata coppia anche a velocità ridotta, indispensabile per il mantenimento sull'asse verticale. Il collegamento di questi al driver (tipicamente L298N) consente la regolazione bidirezionale della tensione applicata, oltre alla possibilità di modulare la velocità attraverso segnali PWM (Pulse With Modulation). Infine, il regolatore di tensione permette il voltaggio adeguato delle componenti, garantendo la massima resa dei motoriduttori (alimentazione a 12V) ed un funzionamento a 5V nel resto del circuito.

In [4] e [5] vengono forniti degli esempi mediante l'utilizzo del microcontrollore Arduino UNO e Arduino Pro Mini, rispettivamente. Di particolare interesse, in [4],

è la struttura del sistema di locomozione, nel quale i motoriduttori sono collegati al driver L298N, alimentato a 12V e, il regolatore di tensione integrato nella scheda Arduino converte la tensione in ingresso in 5V.

Lo studio dello stato dell'arte ha quindi permesso di gettare le basi per la costruzione del prototipo sia dal lato meccanico che dal lato elettronico permettendo, inoltre, di capire in quali punti poter attuare delle migliorie. Tra queste, è di notevole importanza l'utilizzo del microcontrollore ESP32, anziché della scheda Arduino, sia allo scopo di creare una connessione da remoto con il robot attraverso il modulo Wi-Fi integrato nel microcontrollore sia per le sue caratteristiche tecniche quali dimensioni ridotte, processore a prestazioni più elevate e modalità di sospensione per un basso consumo energetico; quest'ultimo aspetto è da prendere in seria considerazione nel momento in cui si decidesse di alimentare il robot con una batteria.

1. MODELLO FISICO E TEORICO DEL CONTROLLO

Il seguente capitolo descrive dettagliatamente il tipo di approccio utilizzato per la realizzazione del prototipo. Inizialmente, ci si è focalizzati sullo studio del modello fisico di riferimento di cui è stata ricavata la funzione di trasferimento, in regime di Laplace. Una volta constatata l'effettiva instabilità del sistema, si è virato, attraverso taratura empirica, all'implementazione del controllore e al tipo di utilizzo di quest'ultimo al fine di garantire la corretta stabilità del sistema.

1.1. Modello fisico – Il pendolo inverso

Il Self-Balancing Robot può essere rappresentato dal modello matematico del pendolo inverso dal momento che esso rappresenta un sistema intrinsecamente instabile e richiede di un controllo attivo per poter mantenere la posizione verticale.

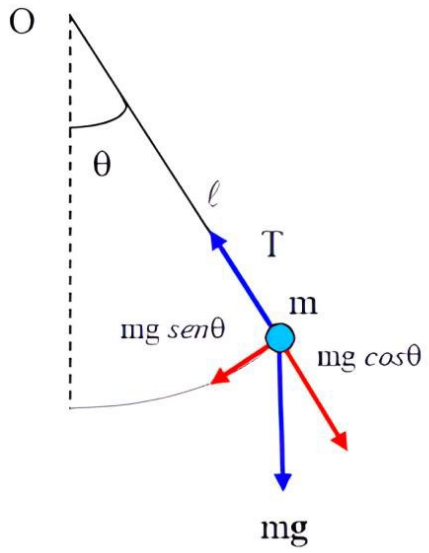


Figura 1.1. Pendolo semplice

Un pendolo semplice è costituito da un filo inestensibile alla cui estremità è appesa una massa puntiforme che può oscillare attorno a un punto fisso detto polo: la

componente della forza peso lungo il filo controbilancia la tensione del filo stesso, mentre la componente della forza peso perpendicolare al filo funge da forza di richiamo e produce il moto oscillatorio del pendolo.

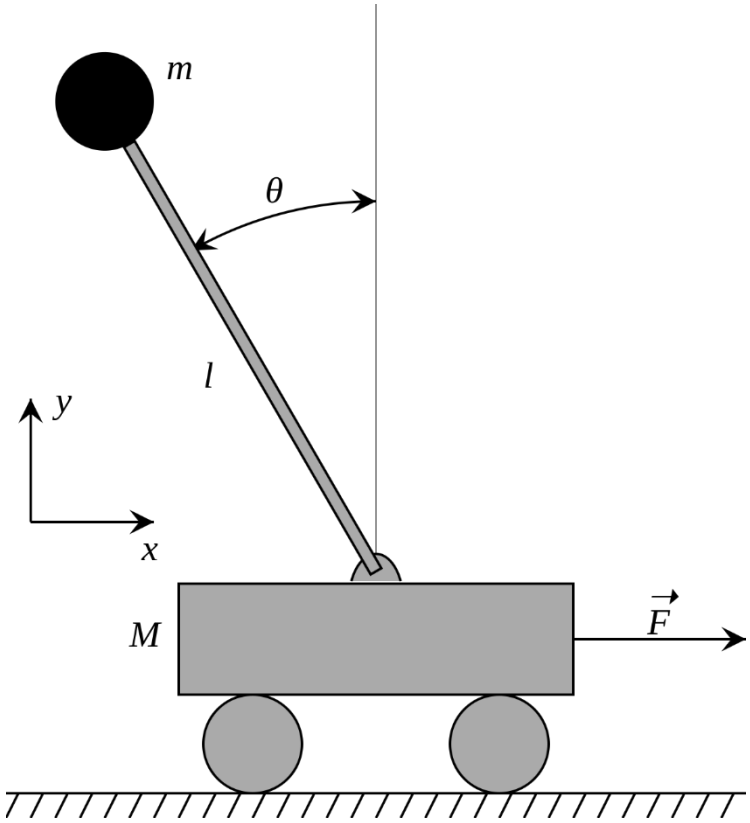


Figura 1.2 Pendolo inverso

Il pendolo inverso rappresenta un pendolo semplice rovesciato, rigido e privo di punto fisso: la parte più bassa può dunque muoversi per bilanciare le oscillazioni della parte più alta, garantendo così l'equilibrio. Il problema del controllo si riconduce dunque a voler stabilizzare la posizione di un'asta vincolata ad un carrello libero di muoversi mediante il movimento della coppia di ruote.

Si definiscono: m la massa dell'asta, M la massa del carrello, I il momento di inerzia dell'asta rispetto al baricentro, L la lunghezza dell'asta e g l'accelerazione gravitazionale. Siano, inoltre, P(·) la forza agente lungo l'asse verticale e N(·) la forza orizzontale, esercitate sull'asta dal carrello.

Si ricavano le equazioni delle due forze come segue:

$$N(t) = m\frac{d^2}{dt^2}\big(x(t) - L\,sin\,\varphi(t)\big) \tag{1}$$

$$P(t) - mg = m\frac{d^2}{dt^2}(L\cos\varphi(t)) \tag{2}$$

Inoltre, risulta necessario ricavare le equazioni del moto di rotazione relativo al baricentro e del moto del carrello:

$$LP(t)\,sin\,\varphi(t) + LN(t)\cos\varphi(t) = I\frac{d^2\varphi(t)}{dt^2} \tag{3}$$

$$M\frac{d^2x(t)}{dt^2} = \ -N(t) + u(t) - b\frac{dx}{dt} \tag{4}$$

Il termine b presente in (4) corrisponde al coefficiente di attrito viscoso.

Andando a sostituire le equazioni (1) e (2) in (3) e (4), si ottiene:

$$L\left[m\frac{d^2}{dt^2}(L\cos\varphi(t)) \ + mg\right]\sin\varphi(t) + L\left[m\frac{d^2}{dt^2}\big(x(t) - L\,sin\varphi(t)\big)\right] \tag{5}$$

$$\frac{Md^2x(t)}{dt^2} = \ -m\frac{d^2}{dt^2}(x(t) - L\sin\varphi(t)) + \ u(t) - b\frac{dx(t)}{dt} \tag{6}$$

Ipotizzando un valore di $\varphi(t)$ molto piccolo, valgono le seguenti approssimazioni:

$$sin\big(\varphi(t)\big) \approx \ \varphi(t), \qquad \cos\big(\varphi(t)\big) \approx \ 1.$$

Con le suddette approssimazioni, le equazioni (5) e (6) diventano:

$$(I + mL^2)\frac{d^2\varphi(t)}{dt^2} - Lmg\varphi(t) = Lm\frac{d^2x(t)}{dt^2} \tag{7}$$

$$(M+m)\frac{d^2x(t)}{dt^2} - mL\frac{d^2\varphi(t)}{dt^2} + b\frac{dx(t)}{dt} = u(t) \qquad (8)$$

Ottenute le equazioni (7) e (8), diventa necessario effettuare la trasformazione nel regime di Laplace per poter studiare la stabilità del sistema:

$$[(I+mL^2)s^2 - Lmg]\Phi(\mathrm{s}) = Lms^2X(s) \qquad (9)$$

$$(M+m)s^2X(s) - mLs^2\Phi(\mathrm{s}) + bsX(s) = U(s) \qquad (10)$$

Dalla (9) si ricava il valore di X(s):

$$X(s) = \frac{(I+mL^2)s^2 - Lmg}{Lms^2}\Phi(\mathrm{s}) \qquad (11)$$

Sostituendo X(s) nella (10) si trova:

$$[(M+m)s+b]s\left[\frac{(I+mL^2)s^2 - Lmg}{Lms^2}\Phi(\mathrm{s})\right] - mLs^2\Phi(\mathrm{s}) = U(s)$$

$$\Rightarrow \left\{\frac{[(M+m)s+b](I+mL^2)s^2 - Lmg}{Lms} - mLs^2\right\}\Phi(\mathrm{s}) = U(s) \qquad (12)$$

Si può così ricavare la funzione di trasferimento G(s):

$$G(s) = \frac{\Phi(\mathrm{s})}{\mathrm{U(s)}} = \frac{Lms}{[(M+m)s+b][(I+mL^2)s^2 - Lmg] - m^2L^2s^3} = \qquad (13)$$

$$= \frac{Lms}{[(M+m)(I+mL^2) - m^2L^2]s^3 + b(I+mL^2)s^2 - mgL(M+m)s - mgLb}$$

La seguente tabella illustra le grandezze considerate nel progetto del prototipo:

Massa del carrello	$M = 0.554\ kg$
Massa dell'asta	$m = 0.308\ kg$
Coefficiente d'attrito viscoso	$b = 0.1\ N/ms$
Lunghezza dell'asta	$L = 0.13\ m$
Accelerazione gravitazionale	$g = 9.81\ m/s^2$
Momento di inerzia $\left(I = \frac{mL^3}{3}\right)$	$I = 0.00201227\ kg\ m^2$

Tabella 1. Grandezze fisiche del prototipo

Sostituendo i dati forniti all'equazione letterale, otteniamo la funzione di trasferimento del pendolo:

$$G(s) = \frac{9.1\ s}{s^3 + 0.16\ s^2 - 76.95\ s - 8.93} \tag{14}$$

Lo studio della funzione di trasferimento è stato eseguito mediante la piattaforma di calcolo MATLAB. In seguito, è fornito il codice:

>> *num* = [9.1 0];

>> *den* = [1 0.16 -76.95 -8.93];

>> *pend = tf(num,den)*;

>> *root(den);*

Gli operatori *num* e *den* rappresentano, rispettivamente, numeratore e denominatore della funzione di trasferimento; attraverso *pend* si rappresenta la stessa. Il comando *root*, invece, esegue il calcolo degli autovalori della funzione.

Così, sono stati trovati i poli, pari a:

8.7505, -8.7944, -0.116.

È evidente la presenza di un polo a parte reale positiva che garantisce l'instabilità del sistema. Questa caratteristica è anche verificabile analizzando come il sistema reagisce ad un ingresso impulsivo e a gradino, attraverso i comandi:

>> *t* = 0 : 0.01 : 5;

>> *impulse (pend, t);*

>> *stepplot (pend, t);*

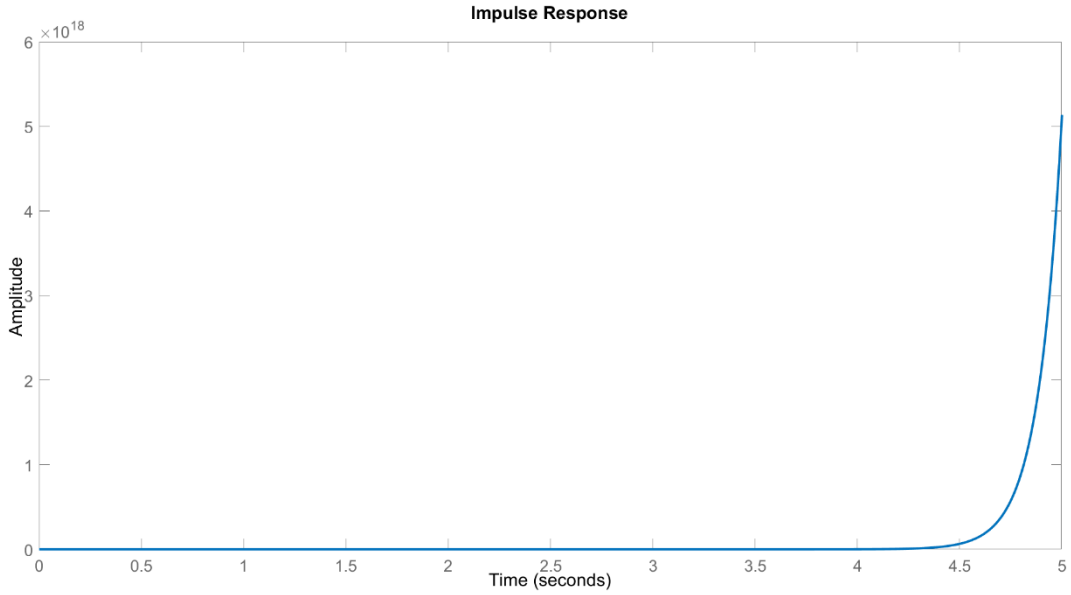


Grafico 1. Risposta impulsiva del sistema

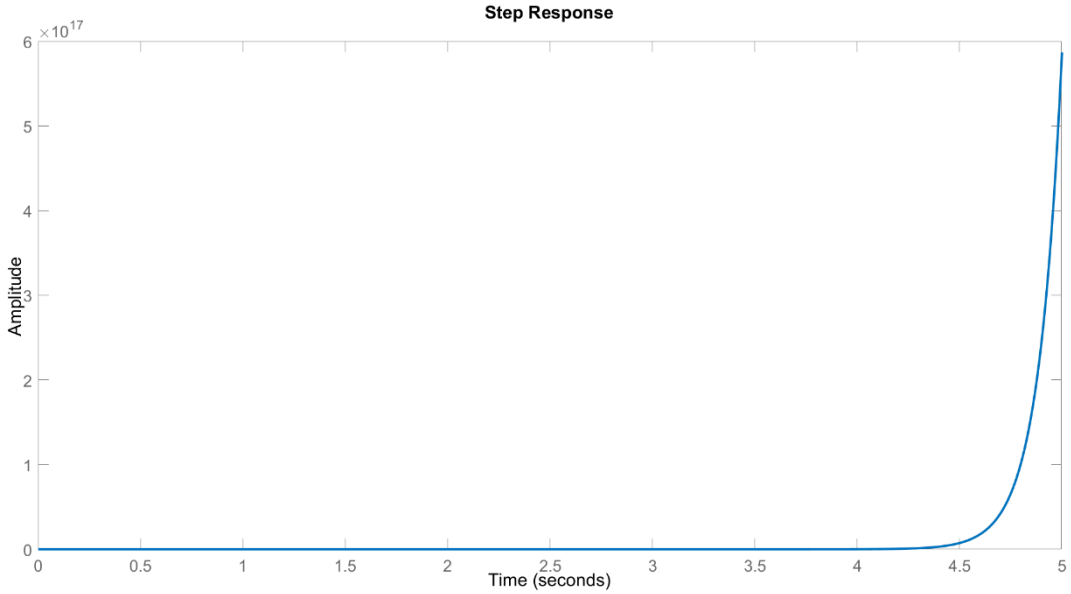


Grafico 2. Risposta al gradino del sistema

Avendo dimostrato l'effettiva instabilità del sistema, si è potuto procedere allo studio e realizzazione del controllore.

1.2. Implementazione del controllore PID

Il controllore PID [7] è un regolatore in retroazione negativa, ampiamente utilizzato nell'automazione industriale e nei sistemi di controllo. Il suo scopo è quello di minimizzare l'errore tra il valore desiderato (*SetPoint*) e quello misurato dalla variabile di processo, regolando opportunamente l'uscita dell'attuatore.

La retroazione negativa è un principio fondamentale del PID: il segnale d'errore, ottenuto confrontando il SetPoint con l'uscita del sistema, viene elaborato dal regolatore per generare un'azione correttiva che tende a ridurre progressivamente l'errore stesso. Questo meccanismo garantisce la stabilità ed evita oscillazioni incontrollate.

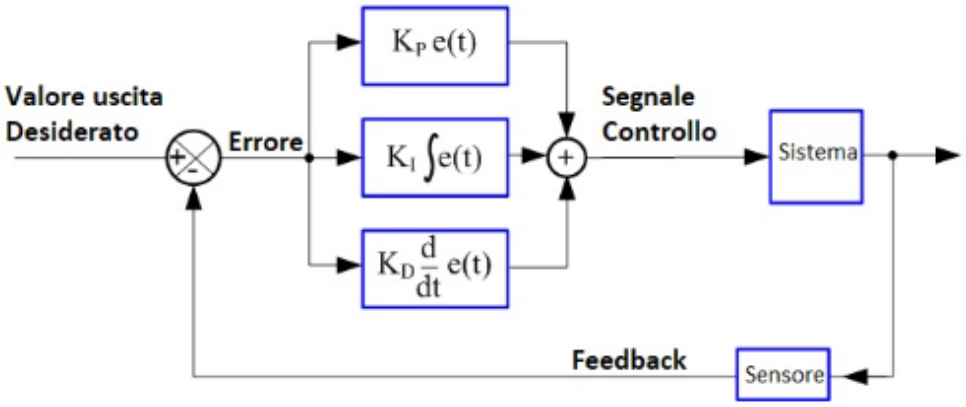


Figura 1.3. Schema a blocchi del controllore PID

L'equazione che definisce il controllore PID è:

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{15}$$

Il PID combina tre contributi distinti: proporzionale, integrale e derivativo. Il contributo proporzionale risponde proporzionalmente all'errore attuale tra il valore desiderato e il valore effettivo, fornendo una risposta rapida alle variazioni dell'errore e ottimizzando la velocità di risposta del sistema. Il parametro caratteristico associato è il K_p, chiamato coefficiente dell'azione proporzionale. Il contributo integrale, invece, compensa l'accumulo di errore nel tempo, correggendo in maniera graduale gli errori passati, così da aiutare a eliminare l'errore in stato stazionario, garantendo che il sistema raggiunga il valore desiderato nel tempo. Il parametro caratteristico è K_i, chiamato coefficiente dell'azione integrale. Infine, il contributo derivativo prevede il comportamento futuro dell'errore, basandosi sulla sua velocità di cambiamento. Il suo obiettivo è ridurre l'oscillazione del sistema, anticipando e attenuando le variazioni dell'errore, garantendo un miglioramento della stabilità e della risposta dinamica del sistema. Il parametro caratteristico è K_d, chiamato coefficiente dell'azione derivativa.

Portando la funzione di trasferimento del controllore PID in regime di Laplace, risulta:

$$C(s) = K_p + \frac{K_i}{s} + K_d\, s = \frac{K_d s^2 + K_p s + K_i}{s} \tag{16}$$

La funzione a ciclo aperto del sistema diventa:

$$F(s) = C(s)G(s) = \frac{K_d s^2 + K_p s + K_i}{s} \frac{9.1\, s}{s^3 + 0.16\, s^2 - 76.95\, s - 8.93} =$$

$$= \frac{\left(K_d s^2 + K_p s + K_i\right)9.1}{s^3 + 0.16\, s^2 - 76.95\, s - 8.93} \tag{17}$$

Il sistema complessivo deve essere caratterizzato da un errore a regime tendente allo zero. Nella seguente tabella viene rappresentata la relazione tra il tipo di sistema e l'ingresso.

tipo \ ingresso	$\frac{1}{s}$	$\frac{1}{s^2}$	$\frac{1}{s^3}$
0	$\frac{1}{1+K_{st}}$	∞	∞
1	0	$\frac{1}{K_{st}}$	∞
2	0	0	$\frac{1}{K_{st}}$

Tabella 2. Errore a regime

La funzione di trasferimento G(s) del sistema pendolo inverso è di tipo 0, motivo per cui si ha la necessità di utilizzare il contributo integrale con coefficiente K_i: viene posizionato un polo nell'origine e la funzione diventa di tipo 1.

Dalla tabella si evince come per un ingresso a gradino unitario $\left(\frac{1}{s}\right)$ l'errore a regime sarà nullo.

A questo punto, per sviluppare il controllore, rimangono da determinare i valori dei coefficienti K_p, K_i e K_d. Vista la complessità della funzione di trasferimento ad anello aperto F(s) si è optato per un metodo di taratura empirica, nello specifico il metodo di Ziegler-Nichols. Attraverso questo tipo di taratura, il sistema, instabile a ciclo aperto, viene considerato stabile a ciclo chiuso.

1.2.1. Metodo di Ziegler-Nichols

Il metodo di taratura empirica di Ziegler-Nichols per i controllori PID, le cui prime forme sono risalenti al 1942, è sostanzialmente un algoritmo atto a trovare il

“guadagno critico” del sistema, dal quale vengono derivati i tre parametri del PID, K_p, K_i e K_d. Il metodo di Ziegler-Nichols richiede sempre in ingresso un gradino e la procedura standard prevede di definire il K_p al minimo e annullare i contributi di K_d e K_i. Dopodiché si va ad aumentare il K_p fino ad ottenere delle oscillazioni persistenti, quindi non smorzate. Il K_p che le provoca prende il nome di K_p “critico” (K_p*). Al di là del valore critico verrà superata la condizione del margine di stabilità che porta il sistema all’instabilità.

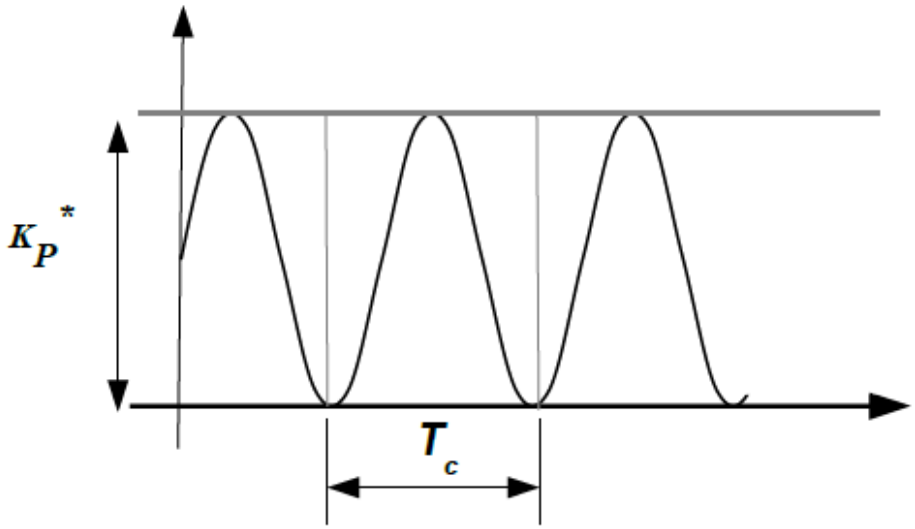


Figura 1.4. Rappresentazione grafica del margine di stabilità

Anche in questo caso si è usufruito di MATLAB per la visualizzazione oscillazioni del sistema.

```
>> contr = tf ( [ Kd  Kp  Ki ], [1 0] );

>> sys_cl = feedback ( pend, contr );

>> t = 0 : 0.01 : 5;

>> stepplot ( sys_cl, t );
```

Una volta definiti i coefficienti del controllore, si è usato il comando *contr* per definire la funzione di trasferimento del controllore. Successivamente, si è

utilizzato il comando *feedback* che ha permesso di porre il controllore in retroazione e, infine, si è calcolata la risposta all'ingresso a gradino.

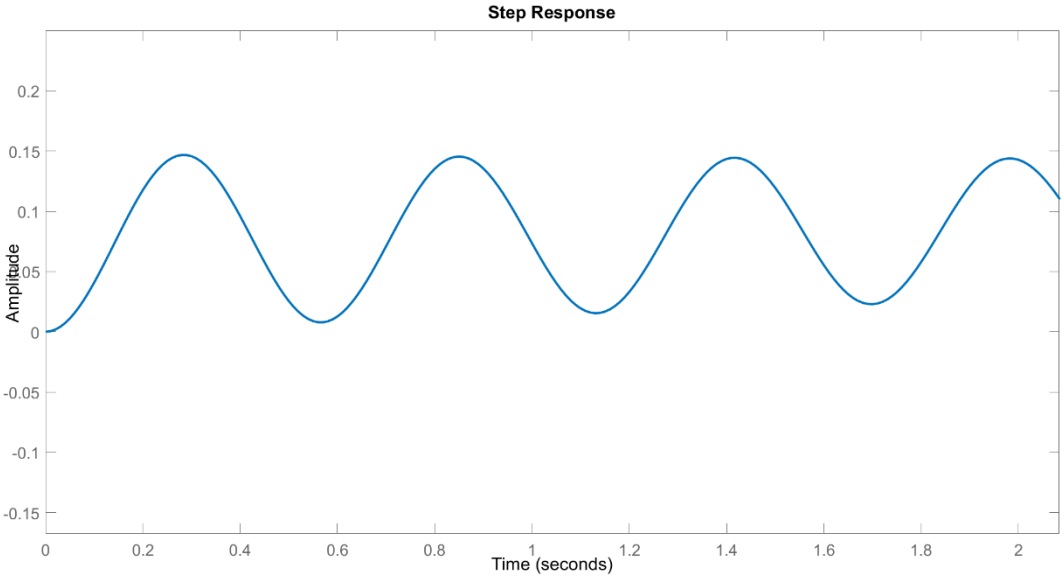


Grafico 3320. Risposta al gradino dopo l'applicazione di Ziegler-Nichols

Si è riusciti a trovare delle oscillazioni non smorzate per $K_p^* = 22.0$ con un periodo di oscillazione T_C pari a 0.55 secondi.

Il metodo di Ziegler-Nichols utilizza delle tabelle nelle quali sono indicati i valori da assegnare ai coefficienti del PID.

	K_P	τ_I	τ_D
P	$0.5\,K_P^*$	–	–
PI	$0.4\,K_P^*$	$0.8\,T_C$	–
PID	$0.6\,K_P^*$	$0.5\,T_C$	$0.125\,T_C$

Tabella 3 Tabella dei valori di Ziegler-Nichols

I parametri τ_I e τ_D sono, rispettivamente, il tempo dell'azione integrativa e il tempo dell'azione derivativa e vengono definiti dai seguenti rapporti:

$$\tau_I = \frac{K_p}{K_i}, \qquad \tau_D = \frac{K_d}{K_p}$$

Dalle ricerche effettuate sui Self-Balancing Robot si è ritenuto opportuno procedere con la realizzazione di un controllo PID affinché il robot possa riuscire anche a fornire una risposta anticipata, grazie al contributo derivativo, evitando oscillazioni e rendendo il movimento più fluido; allo stesso tempo, risulta di fondamentale importanza il contributo integrativo, il quale riesce a migliorare la stabilità statica correggendo gli errori nel tempo, come, ad esempio, la tendenza del robot a pendere da un lato.

Prendendo in riferimento la terza riga della tabella si è calcolato il valore dei tre parametri:

$$K_p = 0.6\, K_p^* = 13.2$$

$$\tau_I = \frac{K_p}{K_i} = 0.5\, T_C = 0.275 \quad \Rightarrow\ K_i = \frac{K_p}{\tau_I} = \frac{13.2}{0.275} = 48$$

$$\tau_D = \frac{K_d}{K_p} = 0.125\, T_C = 0.06875 \quad \Rightarrow\ K_d = K_p \tau_D = 13.2 \cdot 0.06875 \approx\ 0.91$$

Si è, così, riusciti a realizzare, in modo ideale, un controllore di tipo PID il cui obiettivo è quello di mantenere l'asse del robot sulla verticale. I parametri dei contributi trovati sono i seguenti:

$$\begin{cases} K_p = 13.2 \\ K_i = 48 \\ K_d = 0.91 \end{cases}$$

Dal momento che si tratta di una raffigurazione ideale, i parametri potrebbero non essere definitivi. Sarà solo in fase di sperimentazione che si riuscirà a trovare il settaggio preciso.

2. STRUTTURA DEL PROTOTIPO

In questo capitolo verrà trattata la modalità di realizzazione del prototipo attraverso una breve descrizione delle componenti utilizzate, suddivise in parti meccaniche e parti elettroniche. La parte meccanica introdurrà i processi di realizzazione del telaio del robot in cui è stata rivolta notevole attenzione alla idealizzazione dei supporti per attuatori, sensori, microcontrollore e display, le cui posizioni saranno giustificate nella seconda parte del capitolo.

2.1. Struttura meccanica del prototipo

Il telaio è stato progettato su Autodesk Fusion 360, software di modellazione. A questo scopo si è prestata particolare attenzione nella progettazione degli alloggi per i sensori e gli attuatori del robot.

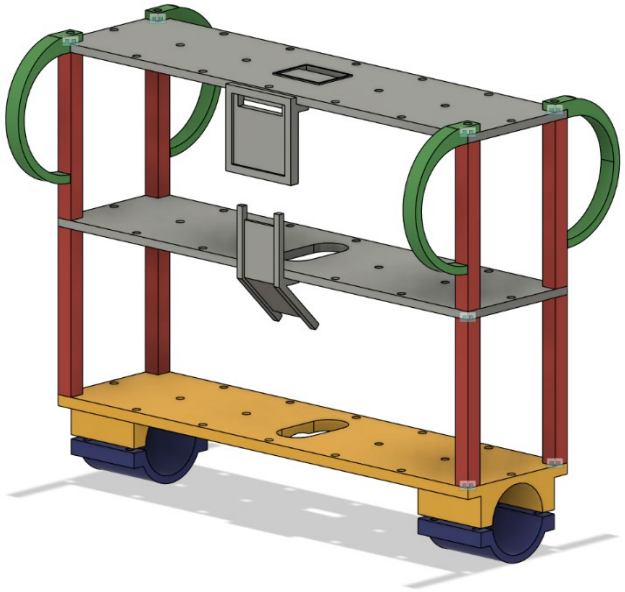
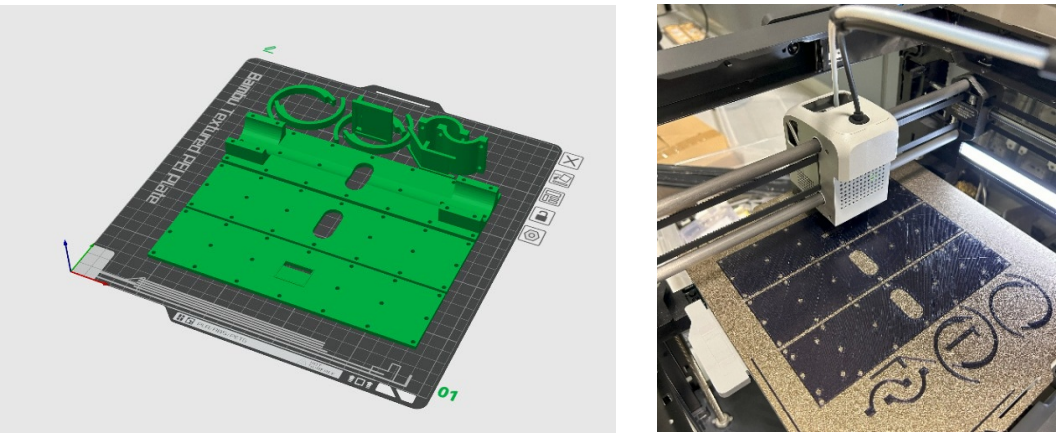


Figura 2.1. Modello 3D Autodesk Fusion 360

Una volta conclusa la progettazione è stata utilizzata una stampante 3D, messa a disposizione dal Rosys Group dell'Università degli Studi di Catania. Si è preferito optare per una stampa 3D per la sua prototipazione rapida per cui si riesce a

progettare e stampare il telaio in poche ore, testarlo e apportarne eventuali modifiche. Questo permette un ciclo di sviluppo molto più rapido rispetto ai metodi tradizionali, consentendo di risparmiare tempi e costi di produzione e di migliorare continuamente il design.

Il materiale utilizzato per la stampa è il PLA (Polylactic Acid) [6], composito in bobina per stampa 3D secondo tecnologia FFF (Fused Filament Fabrication), efficace per la sua resistenza nonché per la leggerezza finale del telaio. Questo ultimo aspetto è di notevole importanza per un robot auto-bilanciante in cui il peso influisce sul comportamento del bilanciamento.



(a) (b)

Figura 2.2. Procedura di stampa 3D: (a) Modello STL, (b) Stampa in corso

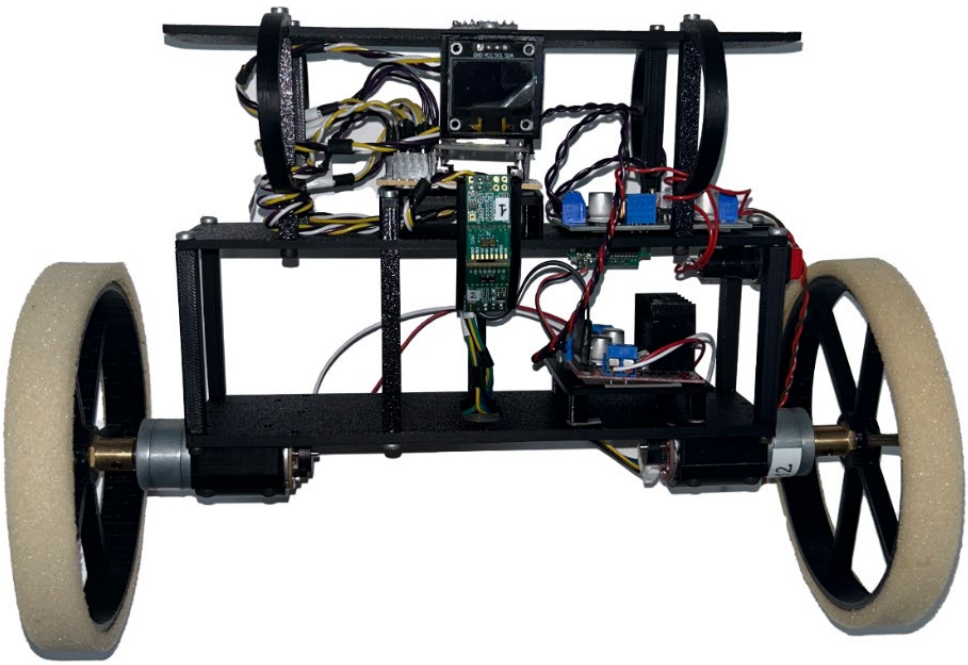


Figura 2.3. Prototipo stampato e assemblato

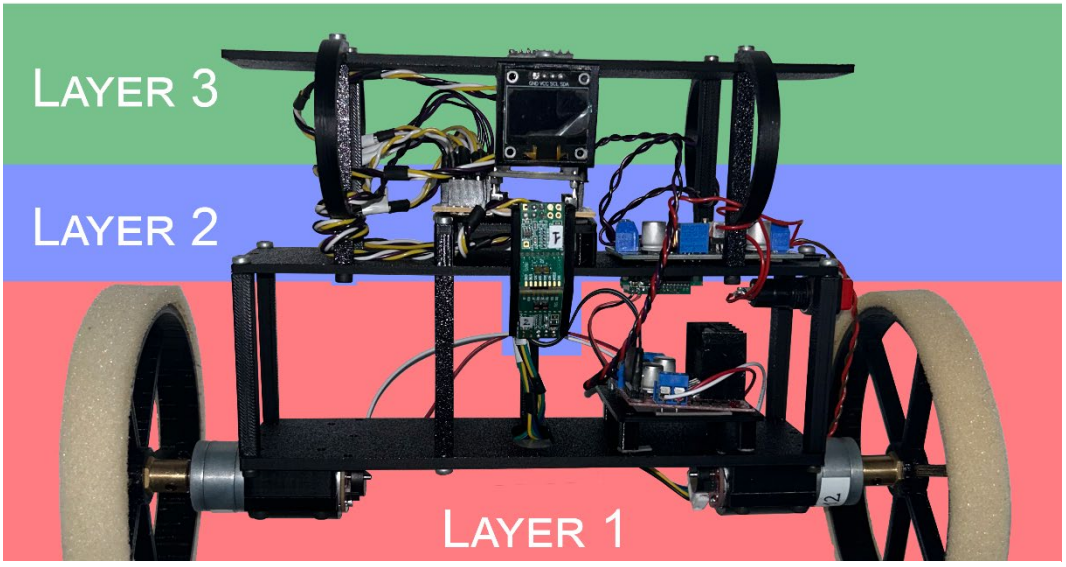


Figura 2.4. Disposizione in Layer del prototipo

Il telaio si compone di tre layer, come mostrato in figura. Nel primo vi è l'alloggio per i due motori DC cui sono calettate le ruote e il supporto per il driver di controllo dei motori. Nel secondo vi è un regolatore di tensione, il supporto per la board del microcontrollore e il supporto per i due sensori di distanza. Quest'ultimo è stato progettato in maniera tale da favorire una diversa funzionalità dei due sensori: il *sensore 1*, posto in verticale, rileva la distanza da un ostacolo guardando in profondità, mentre il *sensore 2*, inclinato di 45°, rileva la presenza del suolo al di sotto del robot. Infine, nel terzo layer, vi è un supporto per un display OLED e l'alloggio per il sensore IMU.

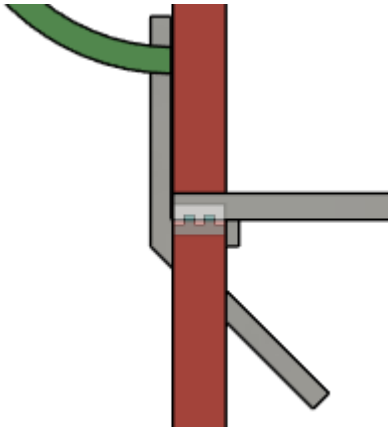


Figura 2.5. Sezione Layer 2: focus supporto dei sensori di distanza

2.2. Struttura elettronica del prototipo

I sensori e gli attuatori costituiscono la parte hardware, fondamentale per permettere al robot di percepire l'ambiente, reagire a stimoli esterni, e adattarsi dinamicamente alle condizioni in tempo reale.

I sensori consentono al robot di raccogliere dati, come la posizione, la velocità, l'orientamento, la distanza da oggetti o altre variabili cruciali per il suo funzionamento, trasformando una grandezza fisica in informazioni digitali, ovvero in segnali elettrici.

Per questo progetto, è stato indispensabile l'utilizzo del sensore GY-521, al cui interno è installato l'IMU MPU6050.

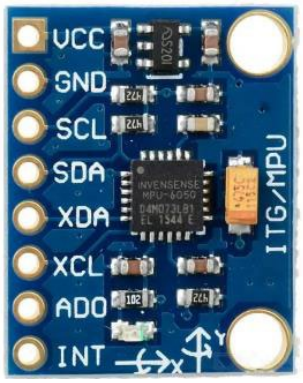


Figura 2.6. Sensore GY-521 MPU6050

Le caratteristiche principali di questo modulo sono la presenza di un giroscopio a 3 assi per la misurazione della velocità angolare e di un accelerometro a 3 assi per la misurazione dell'accelerazione lineare; queste due componenti permettono di individuare l'angolo di inclinazione in maniera accurata, istante per istante. Ciononostante, i valori restituiti dal giroscopio sono soggetti a deriva a lungo termine (drift) e l'accelerometro fornisce dati molto affidabili in condizioni stazionarie ma è sensibile a forze esterne.

Il modulo utilizza un processo di Sensor Data Fusion per combinare i dati provenienti dal giroscopio e dall'accelerometro, ottenendo una stima più accurata dell'orientamento (angoli di pitch, roll, yaw) e del movimento. Il sensore MPU6050 utilizza un algoritmo di fusione, generalmente un Filtro di Kalman o un Filtro

Complementare. Quest'ultimo è stato utilizzato nel progetto, garantendo il bilanciamento dei due set di dati (giroscopio e accelerometro) per produrre una stima stabile dell'orientamento.

Nella realizzazione di questo robot ci si è serviti di due sensori di distanza VL53L1X-SATEL sviluppati da ST-Microelectronics. Questi moduli utilizzano la tecnologia ToF per la misurazione della distanza. La tecnologia ToF calcola la distanza tra il sensore e un oggetto (fino a 4 metri), attraverso un laser a infrarossi (IR), misurando il tempo impiegato dalla luce emessa per viaggiare fino all'oggetto e ritornare al sensore. L'uso dell'IR permette di ottenere misurazioni precise senza il contatto diretto con l'oggetto e con una bassa interferenza da altre fonti di luce.

Figura 2.7. Sensore VL53L1X-SATEL

Per la comunicazione dei tre sensori con il microcontrollore ci si è serviti del protocollo di comunicazione seriale sincrono I2C (Inter Integrated Circuit), il cui vantaggio è quello di impiegare solo due linee per la comunicazione: SDA (Serial Data), per i dati, e SCL (Serial CLock) per il clock; la presenza di quest'ultimo rende l'I2C un bus sincrono composto da almeno un master (che emette il segnale di clock) ed uno slave (che si sincronizza sul clock senza poterlo controllare).

Gli attuatori sono dispositivi che convertono un segnale di controllo in un movimento fisico o in forza meccanica. Nel progetto in studio sono stati utilizzati due motori elettrici in corrente continua (DC) che trasferiscono il movimento alle due ruote calettate ad essi.

Figura 2.8. Motoriduttore DC con encoder

I due attuatori utilizzati dispongono di un motoriduttore a 370 rpm, la cui funzione principale è quella di ridurre la velocità del motore per poterne aumentare la coppia, ovvero la forza esercitata sulle ruote. Inoltre, gli attuatori sono dotati di encoder magnetici.

Gli encoder basati sulla tecnologia magnetica, per rilevare i movimenti rotatori, utilizzano sensori a effetto Hall, fenomeno fisico per il quale si osserva una differenza di potenziale trasversale in un conduttore attraversato da corrente elettrica in verso longitudinale e sottoposto a un campo magnetico perpendicolare. I sensori rilevano l'orientamento di un magnete permanente fissato all'albero dell'encoder e un microprocessore calcola l'angolo di rotazione dell'albero in base ai segnali del sensore.

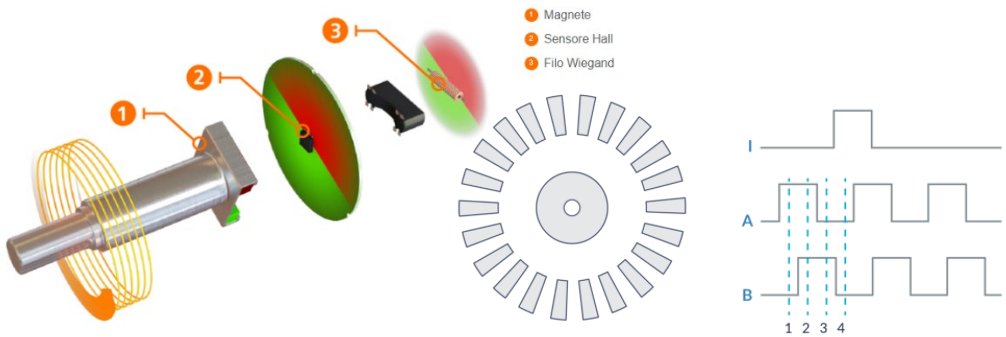


Figura 2.9. Encoder magnetico ad effetto Hall con raffigurazione degli impulsi

I segnali A e B sono di tipo impulsivo e sfasati di 90°, il che permette di determinare la direzione di rotazione (incrementale con codifica in quadratura): se il canale A anticipa B, il motore ruota in una direzione e viceversa. Vengono generati undici impulsi per giro dell'albero motore.

La scheda di controllo dei motori è un dispositivo che sfrutta la combinazione di driver quali il ponte H e un regolatore di tensione integrato, in grado di fornire un'alimentazione di 5 Volt. Un ponte H è costituito da quattro interruttori (transistor o MOSFET) disposti in una configurazione a "H". A seconda di quali interruttori sono attivati, la corrente scorre in una direzione o nell'altra, determinando la rotazione del motore in avanti o all'indietro.

La scheda di controllo è responsabile di gestire il funzionamento e il collegamento dei due motoriduttori. Nel prototipo proposto è stata utilizzata la scheda L298N, basata sul driver Dual H-bridge L298. Quest'ultimo è un circuito elettronico utilizzato per controllare la direzione e la velocità dei motori elettrici indipendentemente. Il modulo L298N si serve del ponte H duale L298 per invertire la polarità della tensione applicata ai motoriduttori, consentendogli di muoversi in entrambe le direzioni.

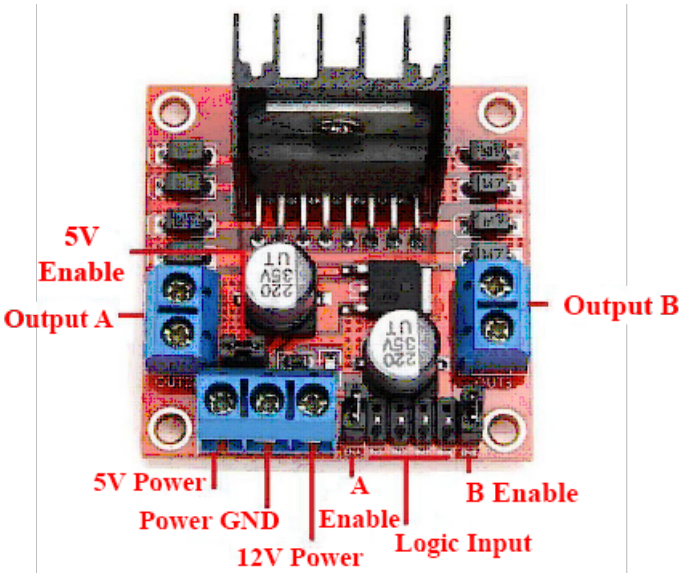


Figura 2.10. Ponte-H L298N

Il regolatore di tensione HW-316 V6 è un modulo DC-DC step-down (buck converter) che permette di ridurre una tensione in ingresso più alta a una tensione di uscita più bassa. È utile ai fini di questo progetto dal momento che per sfruttare la massima potenza dei motori DC è necessaria una tensione pari a 12 Volt, mentre il microcontrollore lavora a tensione di 5 Volt e i sensori a 3.3 Volt.

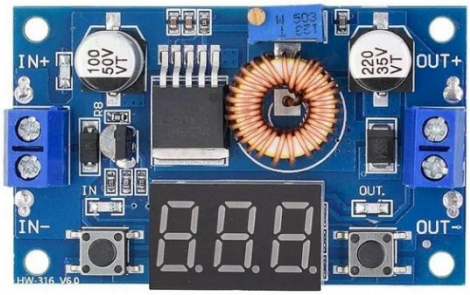


Figura 2.11. Regolatore di tensione HW-316

Per la realizzazione del prototipo è stato utilizzato il microcontrollore ESP32, sviluppato da Espressif Systems. Esso è dotato di connettività Wi-Fi e Bluetooth, basso consumo energetico e alte prestazioni. Di notevole importanza è stata la connettività Wi-Fi dal momento che il microcontrollore è stato utilizzato sia come Access Point che Web Server, così da gestire le richieste http, provenienti da dispositivi connessi alla rete Wi-Fi da esso creata.

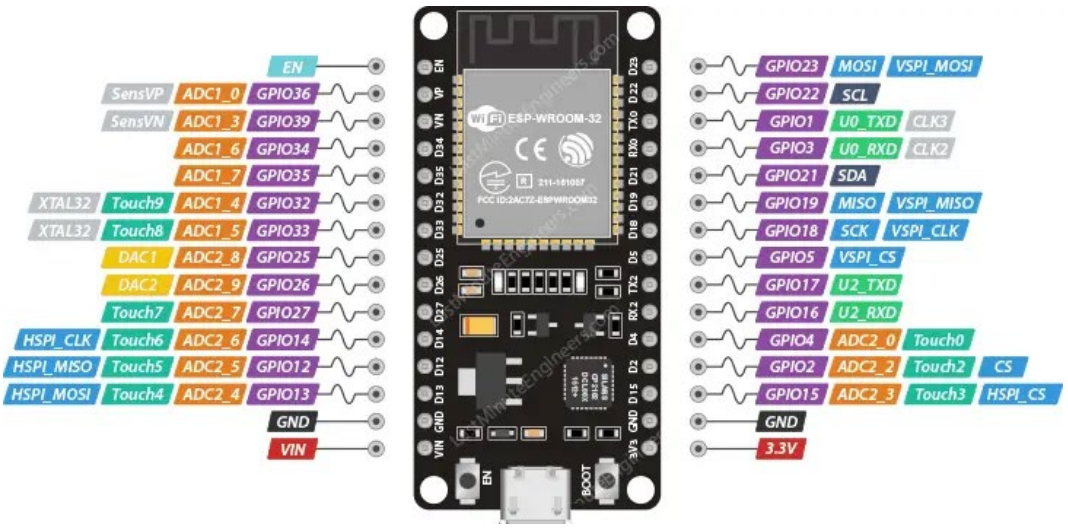


Figura 2.12. PinOut ESP32-Wroom-32

La scheda dispone, nella sua totalità, di 30 pin di cui due per l'alimentazione, rispettivamente a 3.3V e 5V. Tra i vantaggi che sono stati riportati dall'utilizzo di questa scheda spicca la vasta gamma di pin GPIO (General Purpose Input/Output) presenti i quali hanno permesso di gestire e posizionare al meglio le comunicazioni PWM e I2C per attuatori e sensori.

A causa delle sue notevoli dimensioni è stato necessario costruire una board apposita per la realizzazione di tutti i collegamenti con le altre componenti hardware.

Attraverso la piattaforma Fritzing si è rappresentata una schematizzazione dettagliata del circuito elettrico che illustra, in maniera chiara, le connessioni dei vari componenti. Questo modo di operare ha permesso di minimizzare gli spazi occupati dalla nuova board riducendo l'utilizzo di canali al massimo doppi nella parte sinistra della scheda e di un massimo di otto canali nella parte destra.

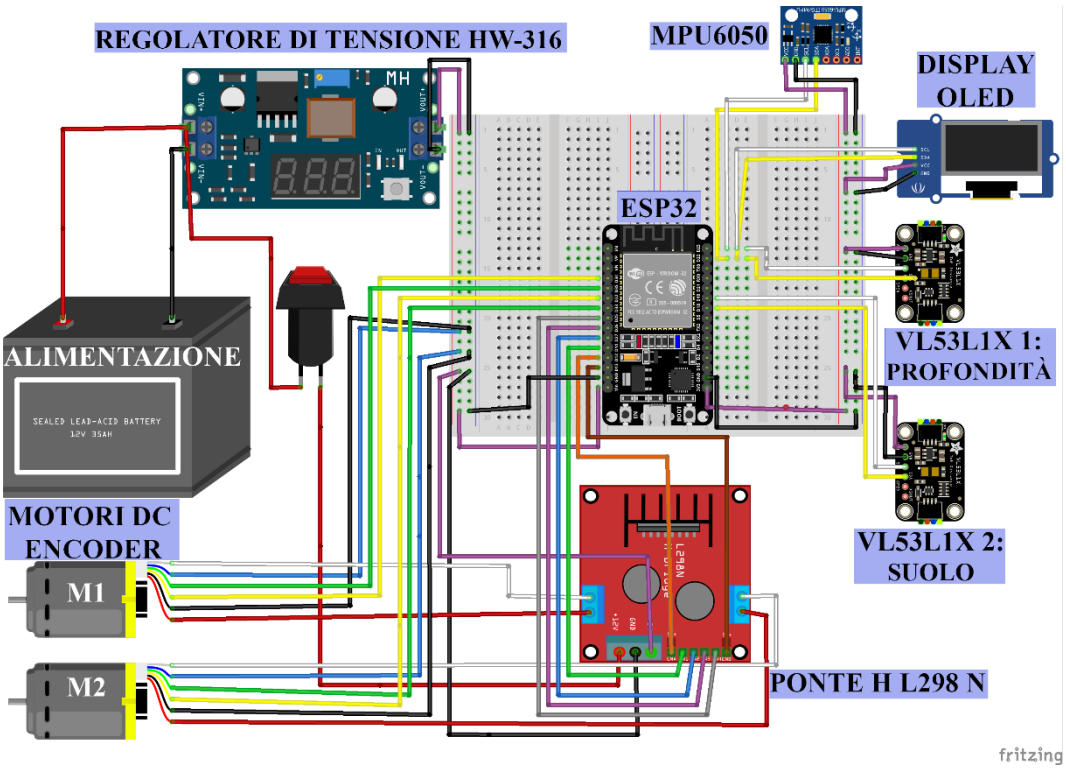
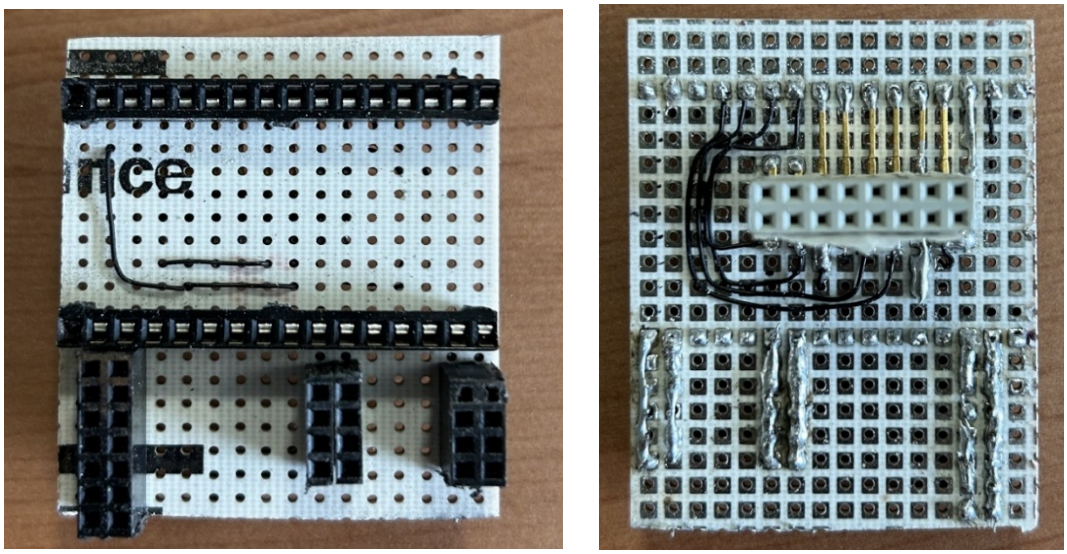


Figura 2.13. Modello del circuito elaborato su Fritzing



(a) (b)

Figura 2.14. Prototipo su mille fori per ESP32: (a) fronte, (b) retro

Come si evince dalle figure 2.14a e 2.14b si è realizzato un binario su cui collegare i pin del microcontrollore. Nel binario destro dell'Esp32 (Figura 2.14b, in basso) sono stati realizzati i canali per le alimentazioni a 3.3V dei sensori e GND, due canali sui GPIO 18 e 19 (SDA, SCL) per la comunicazione I2C del secondo sensore VL53L1X e altri due canali sui GPIO 22 e 23 (SDA, SCL) per la comunicazione I2C dell'IMU, del display OLED e del primo sensore VL53L1X. La separazione della comunicazione tra i tre sensori è stata necessaria al fine di evitare interferenze tra di essi.

Nel binario sinistro (Figura 2.14a, in alto) è stato previsto un canale doppio per le alimentazioni a 5V e GND, rispettivamente per gli Encoder e per il regolatore di tensione. Inoltre, al fine di minimizzare il più possibile gli spazi, si è creata una diramazione dei pin 13,12, 14, 27, 26, 25, 33, 32, 35 e 34 mediante l'utilizzo di un alloggio per pin connettori collocato precisamente sotto al binario su cui giace il microcontrollore; i pin in questione riguardano, rispettivamente, i collegamenti di EnablePin1, ForwardPin, BackwardPin, ForwardPin2, BackwardPin2, EnablePin2, Encoder1Pin1, Encoder1Pin2, Encoder2Pin1, Encoder2Pin2. Il doppio vantaggio sta nel fatto che, oltre a ridurre gli spazi, si è potuto sfruttare l'incavo presente al centro del layer, rendendo più ordinati i collegamenti.

3. ALGORITMI DI CONTROLLO E SVILUPPO SOFTWARE

Nel seguente capitolo verrà fornita una descrizione specifica delle tecniche di calcolo dell'angolo di inclinazione, dell'implementazione del controllo PID e della realizzazione della applicazione web per il controllo da remoto del prototipo. Parte dei risultati ottenuti saranno visualizzati sul display OLED per un controllo in tempo reale dei dati. Ci si focalizzerà sulla spiegazione delle funzioni principali; il corpo del codice completo è visualizzabile in Appendice A.

3.1. Calcolo dell'angolo di inclinazione

Alla base del funzionamento di un robot auto-bilanciante vi è l'angolo di pitch o beccheggio, il quale rappresenta l'inclinazione del robot rispetto all'asse verticale. Nel prototipo realizzato ci si è serviti del sensore inerziale MPU6050 per il calcolo dell'angolo. L'unità inerziale IMU si avvale dell'utilizzo di due diversi sensori, un accelerometro e un giroscopio. Dal lato software, è stata utilizzata la libreria "MPU6050.h" per elaborare i dati forniti dal sensore.

Di seguito sono riportate le descrizioni del codice 1.1 riportato in Appendice A. Nella funzione *setup()* è stata innanzitutto inizializzata la comunicazione I2C tra sensore e microcontrollore; successivamente, si è progredito con l'inizializzazione del sensore e la calibrazione di accelerometro e giroscopio. Infine, è stato applicato un filtro passa-basso digitale (DLPF) con una banda di 20Hz per ridurre il rumore.

Nel *loop()*, la funzione getMotion6(·) consente di leggere i dati (raw) di accelerometro e giroscopio; precisamente, legge sei valori e li memorizza nelle variabili passate per riferimento. L'accelerazione (*ax, ay, az*) viene misurata in unità LSB (Least Significant Bit), da convertire in m/s^2. Le accelerazioni misurate sono quelle degli assi x, y e z. L'angolo di inclinazione calcolato dalla funzione atan2(·) è quello che si forma tra gli assi z e x, rispettivamente l'asse longitudinale e l'asse sagittale. La velocità angolare (*gx, gy, gz*), anch'essa in LSB, è misurata lungo i tre assi ma, per il calcolo dell'inclinazione, è necessaria solo la componente *gy*.

Infine, viene sfruttato il filtro complementare, il quale combina gli angoli trovati dai due sensori, regolandoli per mezzo di un coefficiente di smorzamento alpha: viene così mantenuto il contributo del giroscopio e aggiunto quello dell'accelerometro in maniera tale da correggere la deriva del giroscopio. Il filtro agisce come un passa-alto per il giroscopio e come un passa-basso sull'accelerometro proprio per filtrare la deriva e il rumore della misurazione.

3.2 Implementazione del codice del controllore

Il controllo PID è stato sviluppato con l'aiuto della libreria "PID_v1.h". L'implementazione del codice consta di tre parti, divise in dichiarazione, inizializzazione e utilizzo delle variabili. Il corpo è presentato nel codice 1.2 dell'Appendice A.

Tutte le variabili definite sono state prese come riferimento ai parametri passati al costruttore dell'oggetto PID. Di notevole importanza è il parametro "DIRECT" che fornisce la direzione dell'azione del controllo: in questo caso l'output aumenta all'aumentare dell'errore, viceversa si utilizza il parametro "REVERSE".

Le funzioni di inizializzare il PID sono rispettivamente:

- myPID.SetMode(AUTOMATIC) imposta il PID in modalità automatica, permettendo il calcolo dell'uscita;
- myPID.SetOutputLimits(-255, 255) definisce i limiti per l'uscita del PID rappresentando la massima velocità in una direzione e nella sua opposta;
- myPID.SetSampleTime(sampleTime*1000) imposta l'intervallo di campionamento del PID, ovvero il tempo tra un calcolo e l'altro.

Nel ciclo continuativo viene specificato il passaggio della variabile raffigurante l'angolo corrente in input al PID, ottenuto dai dati elaborati dal sensore IMU. Infine, viene calcolata l'uscita PID con myPID.Compute(), basandosi sull'errore tra *SetPoint* e *input*.

La velocità dei motori in base all'output è settata, in valore assoluto, ad un massimo di 110 su 255 al fine di evitare movimenti troppo bruschi a causa dell'importante potenza massima degli stessi (§ paragrafo 2.2.).

3.3 Realizzazione dell'applicazione Web

Il microcontrollore Esp32 integra un modulo Wi-Fi conforme allo standard IEEE 802.11 b/g/n, operante nella banda 2.4GHz con supporto Dual-mode Wi-Fi: può operare sia come Access Point (AP) che come Client (STA). Questa caratteristica si è mostrata di rilevante importanza nella realizzazione dell'applicazione Web per la gestione del prototipo.

Si è configurato il microcontrollore come un Access Point, il quale ospita un server web in maniera tale da poter gestire le richieste Client su una rete Wi-Fi fornita direttamente dalla scheda. Di seguito verrà fornita una descrizione del codice 1.3 dell'Appendice A, sviluppato col contributo della libreria "Wifi.h" e "WebServer.h".

Inizialmente, sono state configurate le variabili riguardanti il nome della rete Wi-Fi che l'Esp32 creerà (SSID – Service Set Identifier) e la rispettiva password di accesso. Successivamente, è stato creato un server HTTP sulla porta 80, porta standard per i server web.

La stringa *webpage* contiene il codice HTML da inviare ai client quando accedono alla rete Wi-Fi (). L'uso di PROGMEM permette di salvare la pagina web nella memoria flash della ESP32 con lo scopo di non caricare eccessivamente la RAM.

Nella funzione di inizializzazione avvengono le configurazioni dell'Access Point, tramite la funzione WiFi.softAP(), e delle rotte del server Web. Infine, il server viene avviato, rendendolo pronto a ricevere le richieste dai dispositivi connessi.

La funzione handleClient() gestisce le richieste HTTP in arrivo dai client; è fondamentale la sua presenza nel ciclo infinito poiché deve essere chiamata continuamente, altrimenti il server non fornirà risposte.

Una volta effettuato l'accesso alla rete Wi-Fi, la Web App si presenterà all'indirizzo http://192.168.4.1.

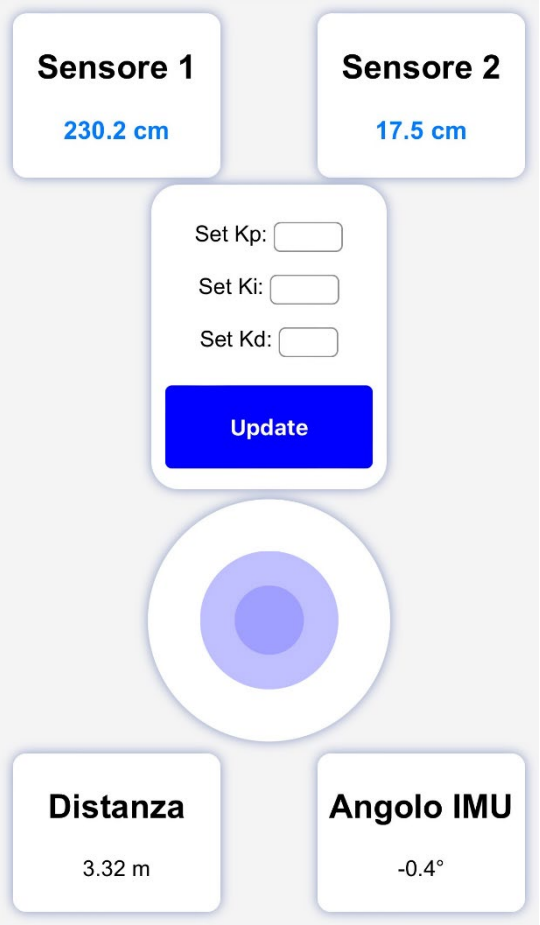
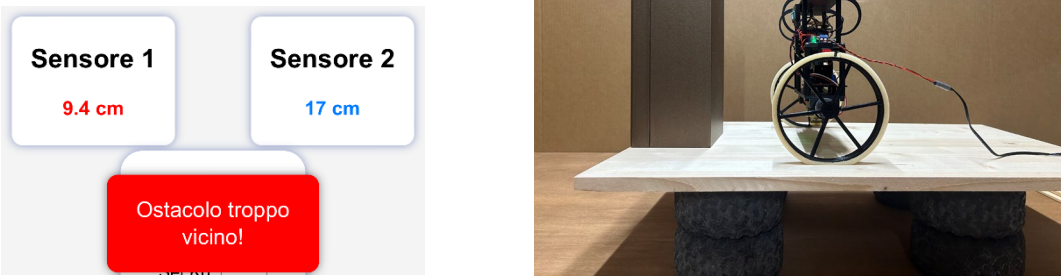


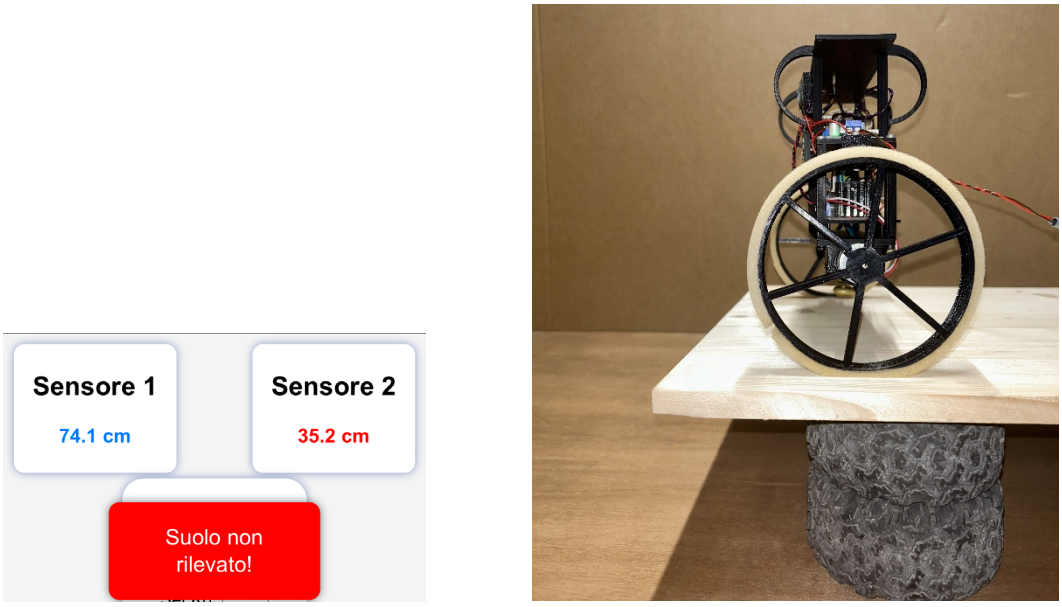
Figura 3.1. Raffigurazione dell'applicazione Web

Una delle principali funzionalità dall'applicazione è la visualizzazione della distanza misurata dai sensori VL53L1X. Rispettivamente, il sensore 1 misura la distanza in profondità con la particolarità che, qualora trovasse un ostacolo nelle vicinanze, mandi un avviso al pilota, fermando i motori nel caso di ostacolo troppo vicino. Il sensore 2 misura la distanza dal suolo e, qualora non rilevasse più il terreno di fronte a lui, blocca i motori lanciando un avviso al pilota. Il tutto è gestito dalla funzione handleData().



(a) (b)

Figura 3.2. Allarme sensore 1: (a) Visualizzazione messaggio, (b) Robot e ostacolo in profondità



(a) (b)

Figura 3.3. Allarme Sensore 2: (a) Visualizzazione messaggio, (b) Robot prossimo al vuoto

Nel riquadro centrale si rende possibile effettuare la modifica dei singoli parametri del controllore PID, i quali vengono caricati nel microcontrollore tramite il pulsante Update, gestita dalla funzione handleSet() e dalla funzione SetTunings(), presente nel loop(), la quale aggiorna continuamente i parametri del controllore al valore scelto.

Inoltre, sono stati inseriti dei riquadri informativi riguardanti la visualizzazione della distanza percorsa dal robot, rilevata grazie agli encoder presenti negli

attuatori, e dell'angolo di pitch misurato dalla IMU, gestite, rispettivamente, da handleData() e handleIMU().

Per ultimo è stato realizzato un Joystick virtuale per il controllo del movimento del robot nelle quattro direzioni con particolare attenzione nei metodi di curva, i quali possono far ruotare il robot su sé stesso (joystick tutto a destra/sinistra) oppure farlo curvare dolcemente (joystick leggermente a destra/sinistra). La gestione avviene mediante handleControl().

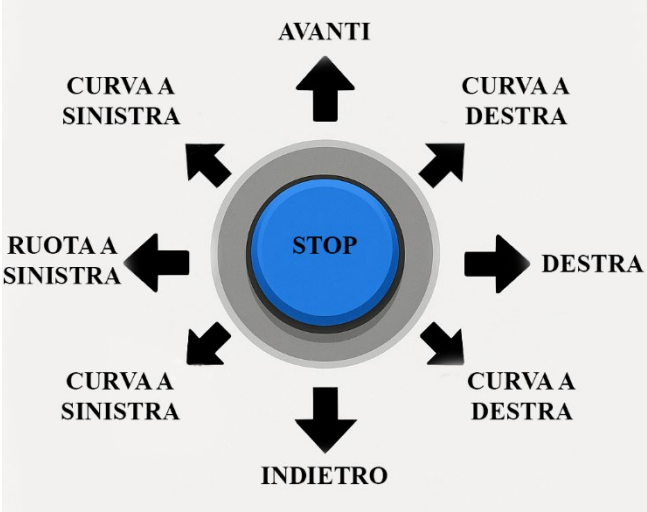


Figura 3.4. Direzioni Joystick

Il modello dal quale si è preso spunto nell'implementazione del movimento è quello degli Hoverboard. Questi strumenti sfruttano due motori elettrici, uno per ciascuna ruota, controllati da sensori giroscopici e sensori di inclinazione. Quando il pilota si inclina in avanti o indietro, i sensori rilevano il cambiamento di inclinazione e attivano i motori per far muovere il dispositivo.

Adottando il principio appena citato, si è implementato un nuovo *SetPoint*, variabile da -5° a + 5°, il quale si attiva soltanto nel momento in cui l'utente muove il Joystick virtuale nelle direzioni avanti, indietro, curva a sinistra e curva a destra; nel caso di rotazione sul posto, il *SetPoint* rimane fissato a 0°.

4. RISULTATI SPERIMENTALI

Nel seguente capitolo saranno descritti gli esperimenti condotti sul prototipo al fine di raggiungere gli obiettivi preposti. Inizialmente si è condotta una verifica in merito al raggiungimento della stabilità sia in quiete che a seguito di perturbazioni. Inoltre, sono stati effettuati dei test su superfici inclinate. Successivamente, si è effettuata una modifica nei codici per sincronizzare il controllo PID con il movimento, in modo tale da garantire un controllo adeguato agli impulsi forniti dai motoriduttori.

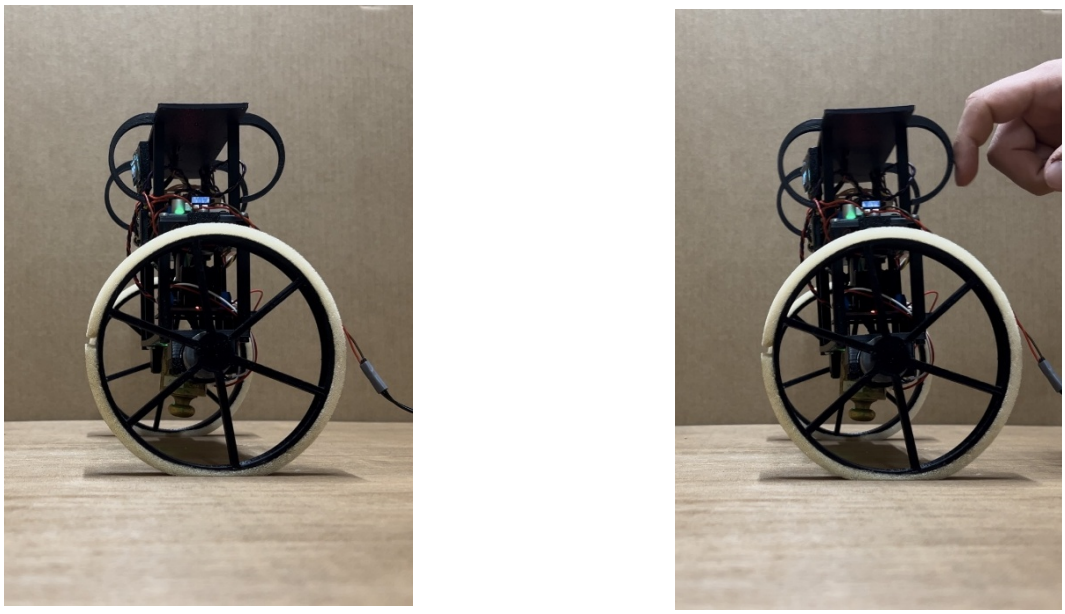
Nello specifico, a partire dai parametri ideali ottenuti dallo studio del modello fisico (§1.2.1.) è stato effettuato il tuning del PID. I parametri iniziali sono risultati inadeguati al fine del bilanciamento e si è proseguito, per tentativi, alla correzione degli errori. A questo scopo, inoltre, è stato aggiunto un carico di 200g nella base inferiore del Layer 1 in maniera tale da abbassare il baricentro del prototipo e, quindi, di garantire il corretto raggiungimento della stabilità. A seguito di questi accorgimenti, sono stati ricavati i tre parametri corretti per il controllo:

$$\begin{cases} K_p = 18.0 \\ K_i = 12.0 \\ K_d = 0.10 \end{cases}$$

Si è rivelata corretta la scelta di posizionare il sensore IMU nel Layer 3. Se fosse stato posizionato molto vicino ai motoriduttori sarebbero stati riscontrati problemi in seguito alla generazione di notevoli vibrazioni meccaniche dei motori, le quali sarebbero state percepite come rumore dall'accelerometro e avrebbero influenzato il giroscopio a causa di micro-movimenti rapidi e non voluti, introducendo dei disturbi nel calcolo dell'angolo mediante filtro complementare. Così, avendolo posizionato più in alto, meccanicamente isolato, le vibrazioni si sono potute attenuare lungo la struttura. Inoltre, con questa disposizione, sono stati ridotti i disturbi dovuti ai campi elettromagnetici generati dai motori, soprattutto quando questi ultimi commutano rapidamente la corrente; dal punto di vista cinematico, il posizionamento alto del sensore ha portato ad un aumento della leva rispetto al punto di rotazione, così da rendere i cambi di inclinazione più pronunciati nella

lettura dell'accelerometro, migliorando la precisione nelle piccole variazioni dell'angolo.

L'ultimo test effettuato sul prototipo è stato quello di sottoporlo a delle sollecitazioni esterne, come una spinta, al fine di osservare la sua reazione a seguito delle perturbazioni. Inizialmente, si è fornita la massima potenza ai motori: i risultati hanno dimostrato come fosse eccessiva, causando uno slittamento delle ruote ed eccessive oscillazioni e impedendo la stabilizzazione. Al fine di ottenere il corretto funzionamento, la potenza è stata ridotta a 100/255 PWM, regolando l'intensità della velocità.



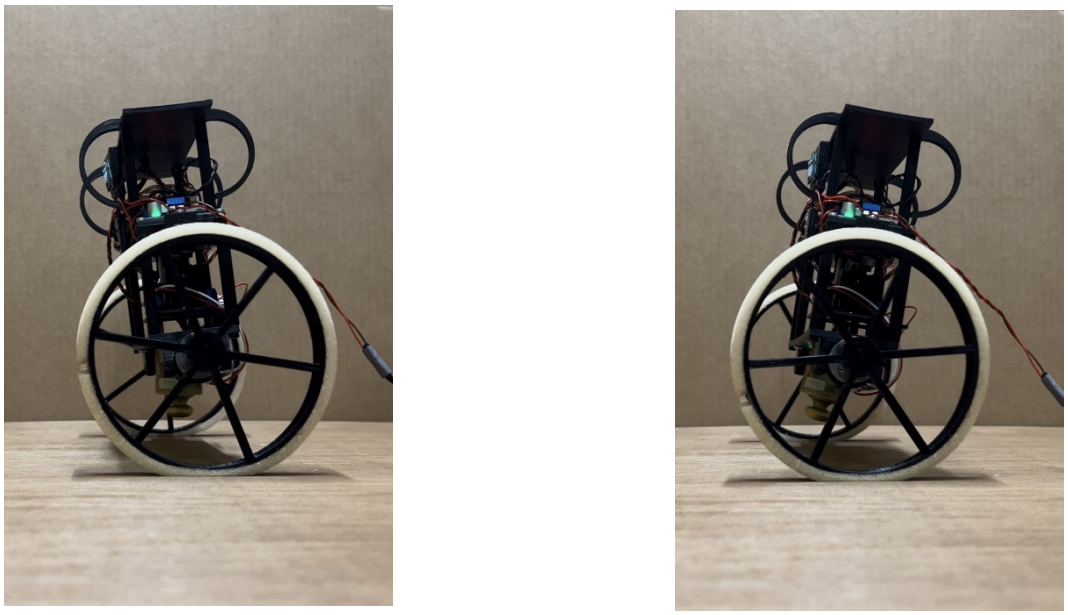
(a) (b)

Figura 4.1. Test di stabilità: (a) Stato di quiete, (b) Perturbazione

In figura 4.1a è raffigurato il robot in fase di quiete -nella quale l'inclinazione rilevata è prossima allo zero-, perfettamente allineato con la verticale del punto di contatto col suolo. Il sistema di controllo mantiene i motori in uno stato di basso intervento sufficiente a contrastare eventuali micro-oscillazioni ambientali.

La perturbazione viene simulata mediante la spinta di un dito (figura 4.1b) provocando un'inclinazione del robot in avanti: l'IMU rileva una variazione

positiva dell'angolo rispetto alla verticale, così il sistema di controllo inizia a reagire aumentando la potenza dei motori nella stessa direzione della perturbazione.



(a) (b)

Figura 4.2. Test di stabilità: (a) Fase di massima inclinazione in avanti, (b) in inversione.

In figura 4.2a viene mostrata la massima inclinazione raggiunta prima dell'intervento correttivo. In questa fase l'angolo di pitch è al suo valore massimo positivo, la velocità è ancora crescente ma il controllo PID ha già calcolato la necessità di invertire la tendenza per evitare la caduta cosicché i motori iniziano a muoversi nella direzione opposta per generare una coppia di riequilibrio. A questo punto il robot inverte la sua traiettoria (figura 4.2b): la direzione del moto è ora retrograda rispetto alla spinta iniziale e i motori lavorano per rallentare la velocità angolare e riportare il robot in posizione verticale, assorbendo l'energia cinetica accumulata. La risposta del sistema è ancora attiva, ma smorzata.

Infine, il robot raggiunge nuovamente la posizione di equilibrio verticale: le oscillazioni residue vengono progressivamente smorzate grazie all'azione del controllore e il sistema torna in quiete.

La risposta alla perturbazione viene descritta nel seguente grafico, realizzato raccogliendo i valori dell'angolo di pitch mel tempo con un campionamento di 20Hz.

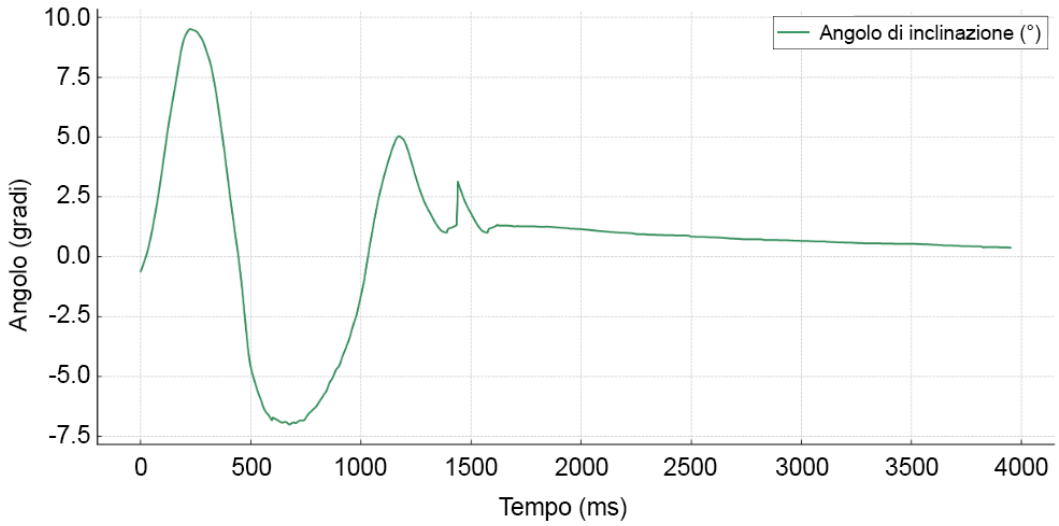


Grafico 4. Andamento dell'inclinazione in seguito a una perturbazione

Nel grafico 4.1 viene mostrata la variazione dell'inclinazione del prototipo rispetto alla verticale in seguito alla perturbazione attuata mediante spinta in cui si evince la rapida risposta fornita dal controllo, quale permette il ritorno sulla verticale in circa 3 secondi.

Al fine di valutare la robustezza del sistema di controllo sviluppato, è stata condotta una serie di esperimenti su superfici inclinate, in particolare con angoli di inclinazione di 5° e 10°. L'obiettivo è stato quello di verificare la capacità del robot di mantenere l'equilibrio anche in presenza di una componente gravitazionale aggiuntiva lungo il piano inclinato.

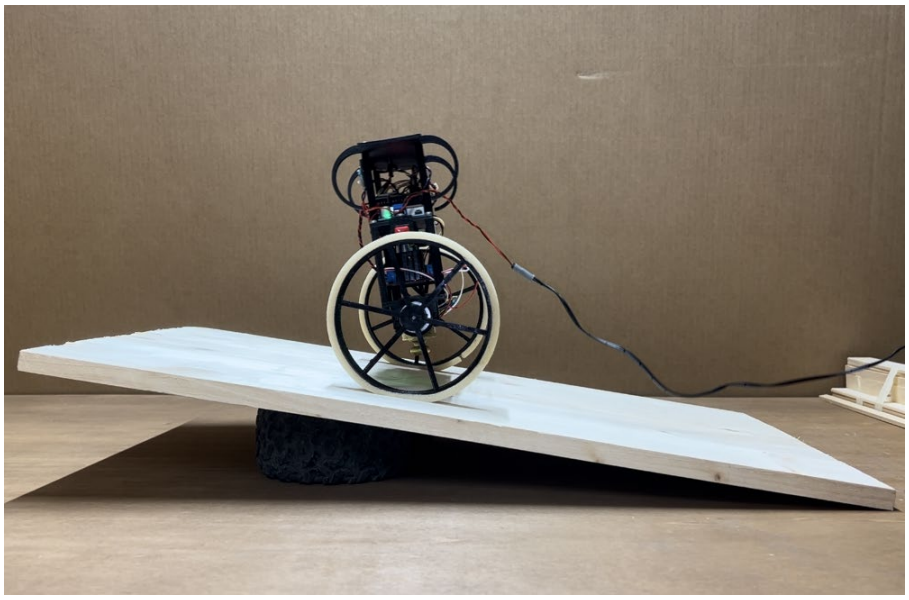


Figura 4.3. Robot in azione su piano inclinato

Durante l’esperimento è stato mantenuto attivo il medesimo algoritmo di controllo utilizzato per il piano orizzontale. In entrambi i casi di inclinazione studiati, il prototipo ha dimostrato la capacità di raggiungere e mantenere la posizione in equilibrio, seppur con un leggero aumento del tempo di stabilizzazione e delle oscillazioni iniziali. Questo comportamento è giustificato dall’incremento della componente della forza peso lungo l’asse del moto, il quale ha richiesto una maggiore azione correttiva da parte del controllore.

CONCLUSIONI

Il presente progetto ha permesso di approfondire in maniera concreta l'implementazione di un controllo PID per la stabilizzazione dinamica di un sistema a due ruote. Attraverso l'utilizzo del filtro complementare per la stima dell'inclinazione, l'elaborazione dei dati provenienti dai sensori e l'integrazione degli attuatori, è stato possibile ottenere un comportamento stabile e reattivo del robot.

La realizzazione dell'interfaccia web ha rappresentato un fattore determinante per lo sviluppo del sistema durante la fase di progettazione e, inoltre, ha reso il sistema più interattivo dimostrando l'efficacia della comunicazione wireless tra il front-end dell'applicazione e il microcontrollore, consentendo un controllo remoto intuitivo e immediato.

Sebbene il robot abbia dimostrato un buon livello di stabilità e reattività, non manca la possibilità di apportare migliorie. Tra queste, ha particolare rilievo la "stima della posizione assoluta" realizzabile attraverso l'integrazione di un modulo GPS o l'impego di tecniche di odometria più avanzate che permetterebbero una migliore localizzazione del robot nello spazio, utile per applicazioni outdoor o navigazione autonoma. Un'altra possibile miglioria potrebbe essere l'implementazione di algoritmi di evitamento ostacoli, evolvendo l'utilizzo dei sensori di distanza mediante tecniche di mappatura come SLAM (Simultaneous Localization and Mapping) che consentirebbe una navigazione autonoma in ambienti complessi.

Le tecnologie alla base del Self-Balancing Robot trovano applicazione concreta in particolare nel settore della micromobilità. Sistemi simili sono impiegati in veicoli elettrici auto-bilanciati come Segway e Hoverboard, dove il bilanciamento dinamico garantisce stabilità, maneggevolezza e sicurezza. Inoltre, l'adattabilità del controllo PID e l'integrazione con sistemi di navigazione e interfacce utente rendono questo tipo di robotica adatta anche a contesti di trasporto autonomo su piccola scala, come in magazzini automatizzati o delivery robot urbani.

APPENDICI

Appendice A – Codici

1.1 Codice per il calcolo dell'inclinazione

```
//Variabili per i dati
int16_t ax, ay, az;   //accelerometro
int16_t gx, gy, gz;   //giroscopio

//Variabili per calcolare gli angoli
float accAngle, currentAngle=0, prevAngle=0, gyroAngle=0;
float lastTime, deltaTime;

//Coefficiente di smorzamento del filtro
float alpha = 0.985;

void setup() {
  Wire.begin(22, 23); //SDA, SCL

  mpu.initialize();
  mpu.CalibrateAccel(10);
  mpu.CalibrateGyro(20);

  mpu.setDLPFMode(MPU6050_DLPF_BW_20);
}

void loop() {
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  // Calcola l'angolo dell'accelerometro (in gradi)
  accAngle = atan2((float)az, (float)ax)* RAD_TO_DEG -90;
  // Calcola il tempo trascorso dall'ultima lettura
  float CurrentTime = millis();
  deltaTime = (CurrentTime - lastTime) /1000.0;
  lastTime = CurrentTime;
  // Calcola l'angolo dal giroscopio (in gradi)
  float gyroRate = (float)gy / 131.0; // 131.0 è la sensibilita
  gyroAngle += gyroRate * deltaTime;  // del giroscopio (±250°/s)

  currentAngle = alpha * (prevAngle + (gyroRate * deltaTime)) +
  (1 - alpha) * (accAngle);
  prevAngle = currentAngle;
  delay(deltaTime);
}
```

1.2 Codice di implementazione controllo PID

```
volatile int pwmSpeed;

//Variabili PID
double Kp=18;
double Ki=15;
double Kd=0.10;
double setPoint = 0;   //Valore desiderato
double targetAngle;
double input, output;
PID myPID(&input, &output, &setPoint, Kp, Ki, Kd, DIRECT);

void setup {
//Inizializza PID
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(-255, 255);
  myPID.SetSampleTime(sampleTime *1000);
}

void loop {
  //Calcola l'input per il PID
  input = currentAngle;
  //Calcola il PID
  myPID.Compute();
  // Limita la      velocità tra 0 e 110
  pwmSpeed = constrain(abs(output), 0, 110);
}
```

1.3 Codice della web App.

```
// Configurazione Access Point ESP32
const char* ssid = "ESP32_AP";
const char* password = "12345678";

WebServer server(80);

const char webpage[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Robot Control</title>
  <meta charset="UTF-8">
   <meta  name="viewport"  content="width=device-width,  initial-
scale=1">
                                                    <script
src="https://cdnjs.cloudflare.com/ajax/libs/nipplejs/0.9.0/nipp
lejs.min.js"></script>
  <style>
```

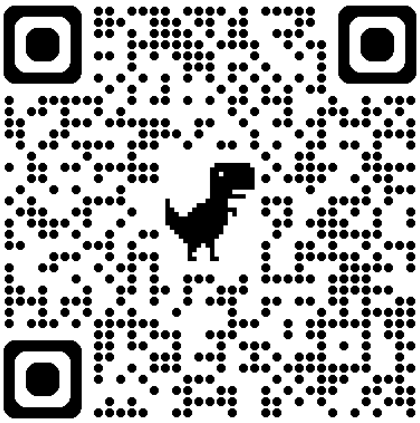
```
  body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    text-align: center;
    margin: 0;
    padding: 0;
  }

  .container {
    position: absolute;
    width: 100%;
    height: 100%;
  }
  .sensor-box,  .motor-box {
    position: absolute;
    width: 140px;
    padding: 5px;
    background: white;
    border-radius: 10px;
    box-shadow: 0 0 8px rgba(30, 60, 150, 0.5);
    text-align: center;
  }
  .popup {
    display: none;
    position: absolute;
    top: 30%;
    left: 50%;
    transform: translate(-50%, -50%);
    background: red;
    color: white;
    padding: 20px;
    font-size: 20px;
    border-radius: 10px;
    box-shadow: 0 0 8px rgba(0, 0, 0, 0.5);
    z-index: 9999;
  }

  #form-container {
  position: relative;
  top: 20%;
  left: 50%;
  transform: translateX(-50%);
  width: 150px;
  padding: 10px;
  background: white;
  border-radius: 20px;
  box-shadow: 0 0 8px rgba(30, 60, 150, 0.5);
  text-align: center;
}


  #sensor1 { top: 10px; left: 10px; }
```

```
    #sensor2 { top: 10px; right: 10px; }
    #motor1 { bottom: 10px; left: 10px; }
    #imu-box { bottom: 10px; right: 10px; }

    form { max-width: 150px; top: 20%; }
     input[type='submit'] { width: 100%; padding: 20px; margin:
5px 0; font-size: 1em; }
    input[type='submit'] { background-color: blue; color: white;
border: none; weight: 20px; }

    .distance { font-size: 18px; font-weight: bold; transition:
color 0.5s ease-in-out; }
    .safe { color: #007BFF; }
    .alert { color: red; }

     #joystick-container {
      position: absolute; top: 67%; right: 20px;
      transform: translate(-50%, -50%);
      width: 175px; height: 175px; background: white;
      border-radius: 50%;
      box-shadow: 0 0 8px rgba(30, 60, 150, 0.5);
    }
  </style>
</head>
<body>
  <div id="popup" class="popup"> </div>
  <div class="container">
    <div class="sensor-box" id="sensor1">
      <h2>Sensore 1</h2>
      <p class="distance safe" id="distance1">-- cm</p>
    </div>
    <div class="sensor-box" id="sensor2">
      <h2>Sensore 2</h2>
      <p class="distance safe" id="distance2">-- cm</p>
    </div>
    <div class="motor-box" id="motor1">
      <h2>Distanza</h2>
      <p id="motorDistance">-- m</p>
    </div>
      <div id="form-container">
        <form action='/set' method='GET'>
           <p>Set Kp: <input type='number' name='Kp' step='0.1'
min='0' max='100' value='%.1f'></p>
           <p>Set Ki: <input type='number' name='Ki' step='0.1'
min='0' max='200' value='%.1f'></p>
          <p>Set Kd: <input type='number' name='Kd' step='0.01'
min='0' max='3' value='%.2f'></p>
          <input type='submit' value='Update'>
        </form>
      </div>
    <div class="motor-box" id="imu-box">
      <h2>Angolo IMU</h2>
```

```
    <p id="imu-angle">--°</p>
    </div>
    <div id="joystick-container"></div>
  </div>
  <script>
     function updateData() {
      fetch('/data')
      .then(response => response.json())
      .then(data => {
        let distance1 = document.getElementById("distance1");
        let distance2 = document.getElementById("distance2");
       let mdistance = document.getElementById("motorDistance");
        distance1.innerHTML = data.sensor1 + " cm";
        distance2.innerHTML = data.sensor2 + " cm";
        mdistance.innerHTML = data.motor1 + " m";

        if (data.sensor1 < 30) {
          distance1.classList.add("alert");
          distance1.classList.remove("safe");
        } else {
          distance1.classList.add("safe");
          distance1.classList.remove("alert");
        }

        if (data.sensor2 > 20) {
          distance2.classList.add("alert");
          distance2.classList.remove("safe");
        } else {
          distance2.classList.add("safe");
          distance2.classList.remove("alert");
        }

        let popup = document.getElementById("popup");
        if (data.suolo) {
          popup.textContent = "Suolo non rilevato!";
          popup.style.display = "block";
        } else if(data.ostacolo) {
          popup.textContent = "Ostacolo troppo vicino!";
          popup.style.display = "block";
        } else {
          popup.style.display = "none";
        }
    });
  }

    function updateIMU() {
      fetch('/imu')
      .then(response => response.json())
      .then(data => {
     document.getElementById("imu-angle").innerHTML = data.angle
+ "°";
      });
```

```
    }

    function sendJoystickData(x, y) {
      fetch(`/control?x=${x}&y=${y}`);
    }

    var joystick = nipplejs.create({
      zone: document.getElementById("joystick-container"),
      mode: "static",
      position: { left: "50%", top: "50%" },
      color: "#0000ff"
    });

    joystick.on("move", function(evt, data) {
      let x = Math.round(data.vector.x * 96);
      let y = Math.round(data.vector.y * 96);
      sendJoystickData(x, y);
    });

    joystick.on("end", function() {
      sendJoystickData(0, 0);
    });

    setInterval(updateData, 100); // Aggiorna i dati ogni secondo
      setInterval(updateIMU,  100);   //  Aggiorna  l'angolo  ogni
millisecondo

  </script>
</body>
</html>
)rawliteral";

void setup() {
  WiFi.softAP(ssid, password);

  server.on("/", []() { server.send_P(200, "text/html", webpage);
});
  server.on("/control", handleControl);
  server.on ("/", handleRoot);
  server.on ("/data", handleData);
  server.on ("/imu", handleIMU);
  server.on ("/set", HTTP_GET, handleSet);

  server.begin();
}

void loop() {
  server.handleClient();
}
```

QR Code del codice completo:



Appendice B – Legende

2.1 Legenda dei cablaggi

FUNZIONE CAVO	COLORE SCHEMA FRITZING
Alimentazione 12V	Rosso
Alimentazione 5V, 3.3V	Viola
Alimentazione 5V Encoder	Blu
Massa (GND)	Nero
IN1	Verde
IN2	Blu
IN3	Viola
IN4	Grigio
Enable Motore 1	Arancione
Enable Motore 2	Blu
Output 1 e 3 attuatori	Rosso
Output 2 e 4 attuatori	Bianco
SDA	Giallo
SCL	Bianco
Pin 1 Encoder	Verde
Pin 2 Encoder	Giallo

INDICE DELLE FIGURE

INDICE DEI GRAFICI

INDICE DELLE TABELLE

BIBLIOGRAFIA

[1] R. S. Martins and F. Nunes, "Control system for a self-balancing robot," *2017 4th Experiment@International Conference (exp.at'17)*, Faro, Portugal, 2017.

[2] Grasser, F., D'Arrigo, A., Colombi, S., & Rufer, A. C. (2002). *Joe: A mobile, inverted pendulum*. IEEE Transactions on Industrial Electronics, 49(1), 107–114.

[3] Debra. *MPU-6050 Redux: DMP Data Fusion vs. Complementary Filter*, GEEK MOM PROJECTS, 2014. Website:

https://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/

[4] T. Thinh. “Self-Balancing Robot ArduinoUno/mpu6050/l298n.” AUTODESK INSTRUCTABLES, 2017. Website:

https://www.instructables.com/Balance-Bot-ArduinoUnompu6050l298n/

[5] S. Midhun. “Arduino Self-Balancing Robot.” AUTODESK INSTRUCTABLES, 2017. Website:

https://www.instructables.com/Arduino-Self-Balancing-Robot-1/

[6] L. Lattanzi “Valutazione delle proprietà meccaniche e degli impatti ambientali di materiali compositi fibrorinforzati stampati mediante tecnologia Fused Filament Fabrication”, 2021.

[7] Migliavacca, Martino. "Progettazione e realizzazione del controllo di una base robotica bilanciante su ruote." 2020.

[8] P. N. Crisnapati, D. Maneetham, Y. Thwe and M. M. Aung, "Enhancing Gimbal Stabilization Using DMP and Kalman Filter: A Low-Cost Approach with MPU6050 Sensor," *2023 11th International Conference on Cyber and IT Service Management (CITSM)*, Makassar, Indonesia, 2023.

[9] Sultana, J. M., Zania, N. H., Azuani, M., Ibrahim, S. Z., & Yusop, A. M. (2022). Analysis of Inertial Measurement Accuracy using Complementary Filter for MPU6050 Sensor. *Jurnal Kejuruteraan*, 34(5), 959–964, 2022.