Fondamenti di Programmazione dei Sistemi Embedded

Relazione progetto

Andrea Alecce

Matricola 214611

a.a. 2020/2021

Indice

1 Introduzione

Obiettivo di questo report è quello di fornire una documentazione tecnica ed esaustiva sui processi, le scelte e le metodologie adottate nella progettazione e realizzazione del dispositivo in oggetto, ovvero un Self Balancing Robot. L'analisi, la modellazione e lo studio di un simile progetto, la cui base concettuale e matematica è quella del classico problema del *pendolo inverso*, è ampiamente portata avanti, in ambito accademico e non solo. La gestione dell'equilibrio in questi tipi di sistemi, è un interessante argomento di ricerca a causa del suo intriseco stato di instabilità, per cui si rende necessario la progettazione di un controllore.

Ciò che differisce per ogni soluzione implementativa è l'approccio al problema, in termini di tecnologie adottate, hardware utilizzato, tipologia di controllo. Si è reso dunque necessario effettuare uno studio preliminare sui sistemi utilizzati, al fine di avere un idea generale sul problema e determinare le migliori soluzioni valutando differenti fattori quali costo dei materiali, tempistiche, risultato finale.

Il progetto è strutturato in alcuni macro-obiettivi, a loro volta organizzati in attività come riportato di seguito.

1. Ricerca e formalizzazione dei processi di sviluppo del progetto
   (a) Stato dell'arte, tecnologie, strategie e metodi utilizzati
2. Definizione delle specifiche
   (a) Definizione delle specifiche funzionali e prestazionali del dispositivo
   (b) Selezione delle tecnologie
   (c) Modellazione matematica del Sistema
   (d) Simulazione, analisi del sistema e controllo
3. Progettazione del sistema
   (a) Progettazione dell'elettronica
   (b) Progettazione del firmware
   (c) Testing e validazione

2 Ricerca e formalizzazione dei processi di sviluppo del progetto

In questa prima fase ci si è concentrati sulla ricerca dello stato dell'arte di dispositivi di interesse per il progetto. Questa ricerca e studio è utile per determinare una idea progettuale di massima. Inoltre, è fonte e ispirazione per quanto riguarda le tecnologie utilizzate, gli approcci realizzativi, le architetture e le soluzioni adottate.

2.1 Stato dell'arte, tecnologie, strategie e metodi utilizzati

In ambito accademico e di ricerca, sono molti i progetti e articoli correlati al *self balancing robot*. In particolare, ci si concentra su sistemi denominati *Two- Wheeled Self-Balancing Robot* (da qui chiamati **TWSBR**).

In [13] vengono affrontate diverse tecniche di modellazione per il TWSBR, confrontando il metodo lagrangiano e il metodo di Kane. In [14] viene affrontato l'intero processo di realizzazione di un TWSBR, basato su motori stepper e controllore PID. In [2] viene realizzato un TWSBR con l'uso di un *filtro di Kalman* per la tecnica di *sensor fusion*. Nell'articolo [10] si descrive la progettazione, la costruzione e controllo di un TWSBR, basato su motori DC, Arduino, un sensore IMU a 6 assi (giroscopio e accelerometro). Inoltre, è implementato un *complementary filter* per compensare gli errori dei vari sensori. Infine, si analizza il controllo sia con un PID sia con LQR. In [9] si illustra lo sviluppo di un TWSBR a scopo didattico, con l'analisi del modello e le simulazioni effettuate. In [12] si descrive la modellazione, la strumentazione e il controllo di un TWSBR, con due ruote indipendenti, basato su microcontrollore ESP32 e IMU MPU6050. Nell'articolo è stato ricavato il modello dinamico, oltre che progettati e simulati un controller PID e un controller LQR su Matlab/Simulink. In [8] si effettua una analisi e modellazione di un TWSBR basato su motori DC.

In [11] si descrive lo sviluppo del modello di un TWSBR e del suo controllo utilizzando varie tecniche, tra cui PID e LQR.

In [18] viene descritto lo sviluppo di un TWSBR, dall'analisi del modello matematico fino alla definizione del controllore, utilizzando come strumento MATLAB.

In [3] è stata affrontata la progettazione di un *segway*, sistema molto affine al TWSBR, descrivendo i metodi di controllo PID e LQR.

Sono stati consultati anche riferimento online, tra blog, forum e siti di interesse tecnico/scientifico. In particolare, in [7] viene descritto tutto il processo di modellazione matematica del sistema e relativo controllo.

Infine, si riporta un progetto ormai ben avviato e maturo, con numerose feature implementative, realizzato da Ascento [5], ovvero un robot capace di salto, compatto e agile progettato per ambienti misti.

3 Definizione delle specifiche e progettazione del sistema

Dall'indagine condotta nel capitolo precedente su TWSBR, è possibile individuare alcuni punti in comune:

1. Necessità di adottare un controllo per stabilizzare il sistema;
2. Alta precisione sulle misure dei sensori;
3. Frequenza di aggiornamento dei dati elevata;
4. Utilizzo di strumenti di simulazione come Matlab/Simulink.

Sulla base di questi punti si definiscono le funzionalità base di cui il dispositivo in fase di realizzazione sarà necessariamente dotato.

3.1 Definizione delle specifiche funzionali e prestazionali del dispositivo

Obiettivo del progetto è quello di realizzare un robot a due ruote motrici, capace di mantenere autonomamente in maniera stabile una posizione di equilibrio in verticale (0° rispetto all'asse perpendicolare al suolo). Qualsiasi disturbo esterno non dovrà comprometterne la posizione verticale. Inoltre, tramite la definizione di un nuovo *set point*, definito come angolo, è possibile far muovere il robot in avanti e indietro. Infine, si propone una semplice soluzione per gestire anche la rotazione a destra e sinistra.

Il robot dovrà essere privo di cavi, che potrebbero influenzarne il controllo, per cui sarà necessario adottare un sistema di alimentazione autonomo basato su batterie e modulo di gestione degli stessi.

3.2 Selezione delle tecnologie

Si rende necessario uno studio sulle tecnologie da utilizzare per la realizzazione del dispositivo. A tal fine, si determinano alcuni punti chiave utili per la scelta, per i componenti di maggiore importanza.

Le scelte saranno basate anche sul vincolo di utilizzo della scheda **STM32 Nucleo-F446RE** per la parte di controllo.

I principali elementi che andranno a definire il progetto sono i motori e la sensoristica.

Per quanto riguarda i motori, si valutano le principali tipologie adottate per la tipologia di sistema in esame, come riportato nella Tabella 1.

Scelta motori		
	Stepper	Motore DC
Alimentazione	12V(0.4A)	3-6V(150mA)
Pin di controllo	4	2
Dimensione (cm)	4.2x4.2x3.4	6.5x1.7x2.2
Velocità (RPM)	600	200
Peso (gr)	230	26
Controllo	Stepper Driver	Driver H-Bridge
Costo (€)	15	1.2

Tabella 1: Tabella di confronto tra motori DC e stepper

Scelta sensori IMU		
	MPU6050	LSM6DS3
Alimentazione (V)	3.3	3.3
Comunicazione	I2C	I2C/SPI
Dimensione (mm)	13x18	13x18
Risoluzione ADC (bit)	16	16
Costo (€)	6.99	7.99

Tabella 2: Tabella di confronto tra motori DC e stepper

Lo stepper richiede una sezione di alimentazione più importante, un numero di segnali per il controllo maggiore, una dimensione e peso maggiori, oltre che il costo. D'altra parte, ne consegue un controllo nettamente migliore oltre che più semplice, soprattutto con i relativi driver. Per questo motivo si è selezionato l'uso di motori stepper. In particolare, è stato considerato uno *Stepper Motor Nema 17 Bipolare modello 17HS13-0404S*. Il driver in uso per controllare il motore stepper è un *TMC2209*, tra i migliori in termini di controllo e rumorosità, spesso adottati come miglioria per dispositivi come stampanti 3D e CNC. Inoltre, questi motori garantiscono una elevata precisione, ragione per cui è possibile non adottare alcun controllo in feedback.

Per quanto riguarda i sensori, l'obiettivo è ricavare una misura quanto più precisa in termini di inclinazione rispetto a un angolo prefissato. Ci si affida dunque a sensori inerziali (*IMU, Inertial Measurement Unit*) per determinare tale angolo, considerandone le principali tipologie come riportato nella Tabella 2.

Come visibile in foto, sono stati selezionati due sensori dotati di caratteristiche simili.

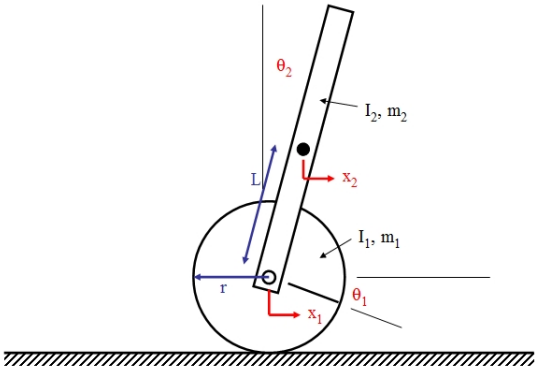


Figura 1: Grafico del modello fisico di un Self Balancing Robot

Altri sensori integrano oltre a giroscopio e accelerometro, anche un magnetometro, arrivando a *9-DOF* (*Degree Of Freedom*). Per gli scopi del progetto, un sensore IMU con 6-DOF è sufficiente. Inoltre, il costo per questi sensori è nettamente maggiore. Si seleziona dunque il sensore **MPU6050**, su board GY-521, in quanto già in mio possesso oltre che avere un costo leggermente inferiore.

3.3 Modellazione matematica del Sistema

Al fine di determinare una corretta legge di controllo, è necessario derivare un modello matematico del sistema e valutarne le caratteristiche. La modellazione di un TWSBR può essere assimilato a quello di un pendolo inverso mobile, di cui si riporta uno schema in Fig. 1.

Si può considerare il TWSBR come un sistema meccanico costituito da due parti accoppiate:

1. Il Corpo, o body, costituito dal corpo principale, dove risiede l'elettronica e l'alimentazione
2. Il Gruppo Ruota, costituito dalle ruote e dai rotori

Si definisce θ_b (o θ_2 nel grafico) come l'angolo formato dalla retta che interseca il baricentro del corpo principale (CM) e l'asse della ruota, con la asse y.

Si riporta nella Tabella 3 i principali parametri, ricavati direttamente tramite misure o indirettamente dalla conoscenza di altri valori.

Gli unici parametri non noti sono i momenti di inerzia relativi alla ruota e al corpo. Nel determinarli, sono state considerate le espressioni dei momenti di inerzia, rispetto all'asse di simmetria passante per il centro di massa, relativi a solidi omogenei notevoli. In particolare,

Parametri progettuali		
Simbolo	Parametro	Valore
R	Raggio della ruota	0.0325 (m)
m_{wa}	Massa della ruota	0.026 (kg)
J_w	Momento di inerzia della ruota	0.0000211 (kgm^2)
l	Altezza corpo	0.02 (m)
m_{wa}	Massa corpo	0.483 (m)
J_b	Momento di inerzia del corpo	0.001304 (kgm^2)
g	Accelerazione gravitazionale	9.81 (ms^{-2})

Tabella 3: Parametri progettuali

per la ruota si è considerato il contributo discreto dato dai raggi, dal cerchione e dallo pneumatico. La massa dello pneumatico, pesata, è pari a 17 grammi, quella della ruota pari a 9 grammi. Supponendo trascurabile, rispetto al peso dello pneumatico, quello della parte esterna, il momento di inerzia viene così calcolato:

$$I = n * I_{spoke} + I_{rim} = 5 * (\frac{1}{3} * m_s * {L_s}^2) + m_r * {L_r}^2 \tag{1}$$

Dove i raggi sono considerati come *spoke*, ovvero un'asta in rotazione su una estremità di lunghezza L_s e massa m_s, mentre lo pneumatico come superficie cilindrica sottile con estremità aperte, di raggio L_r e massa m_r.

Per il corpo si è supposto avere espressione simile a quella di un pendolo inverso, pari a:

$$I_{body} = m_b * {L_b}^2 \tag{2}$$

I passaggi matematici che portano ad un'espressione del problema in termini di equazioni del moto sono state ricavate da [14], di cui si riportano le espressioni 3 e 4 nel dominio del tempo:

$$\ddot{x} = \left[m_b + 2\left(m_{wa} + \frac{J_w}{R^2} - \frac{(m_b l cos(\theta_b)^2)}{m_b l^2 + J_b} \right) \right]^{-1} \left(m_b l sin(\theta_b) {\dot{\theta_b}}^2 - \frac{(m_b l)^2 g cos(\theta_b) sin(\theta_b)}{m_b l^2 + J_b} + 2(\frac{1}{R} + \frac{m_b l cos(\theta_b)}{m_b l^2 + J_b}) T_w \right) \tag{3}$$

$$\ddot{\theta_b} = \left[J_b + m_b l^2 - \frac{\left(m_b l cos(\theta_b)\right)^2}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)} \right]^{-1} \left(m_b l sin(\theta_b) \left[g - \frac{m_b l cos(\theta_b)(\dot{\theta_b})^2}{m_b + 2(m_{wa} + \frac{J_w}{R^2})} \right] - 2 \left[R + \frac{m_b l cos(\theta_b)}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)} \right] \frac{T_w}{R} \right) \tag{4}$$

Linearizzando entrambe le equazioni nell'intorno di $\theta_b = \dot{\theta_b} = 0$ (in quanto si considera il robot inizialmente in verticale), approssimando $\sin\theta_b \approx 0$, $\cos\theta_b \approx 1$ e $\dot{\theta_b} = 0$, si ottiene:

$$\ddot{\theta_b} = C_3\theta_b - C_4 T_w \tag{5}$$

$$\ddot{x} = -C_1\theta_b + C_2 T_w \tag{6}$$

dove:

$$C_1 = \frac{1}{M}\left(\frac{(m_b l)^2 g}{m_b l^2 + J_b}\right) \tag{7}$$

$$C_2 = \frac{2}{M}\left(\frac{1}{R} + \frac{m_b l}{m_b l^2 + J_b}\right) \tag{8}$$

$$C_3 = \frac{m_b g l}{J} \tag{9}$$

$$C_4 = \frac{2}{JR}\left[R + \frac{m_b l}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right] \tag{10}$$

$$C_4 = \frac{2}{JR}\left[R + \frac{m_b l}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right] \tag{11}$$

$$M = m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right) - \frac{(m_b l)^2}{m_b l^2 + J_b} \tag{12}$$

$$J = J_b + m_b l^2 - \frac{(m_b l)^2}{m_b + 2(m_{wa} + \frac{J_w}{R^2})} \tag{13}$$

Considerando l'uso di motori stepper, di cui possiamo controllare la velocità angolare, si ha un legame con la velocità tangenziale del tipo:

$$R\dot{\theta_w} = \dot{x} \tag{14}$$

Per cui le equazioni 5 e 6 diventano:

Parametri numerici	
Parametro	Valore
C_1	1.1927
c_2	145.2181
C_3	70.9901
C_4	2.2728e+03
M	0.5126
J	0.0013

Tabella 4: Parametri numerici ricavati tramite Matlab

$$\ddot{\theta_b} = C_3\theta_b - C_4T_w \tag{15}$$

$$R\ddot{\theta_w} = -C_1\theta_b + C_2T_w \tag{16}$$

Riscrivendo l'equazione 16, si ricava l'espressione di T_w:

$$T_w = \frac{R\ddot{\theta_w} + C_1\theta_b}{C_2} \tag{17}$$

Sostituendo la 17 nella 15, passando nel dominio di Laplace con condizioni iniziali nulle, dopo alcuni passaggi matematici si ottiene una espressione della funzione di trasferimento del sistema:

$$\frac{\theta_b}{s\theta_w(s)} = \frac{-sRC_4}{C_2s^2 + (C_1C_4 - C_2C_3)} \tag{18}$$

dove l'ingresso è la velocità angolare ($\dot{\theta_w}(s) = s\theta_w(s)$) e l'uscita è proprio l'angono θ_b

3.4 Simulazione, Analisi del sistema e Controllo

Una volta ricavata la funzione di trasferimento e i valori dei vari parametri, risulta semplice caratterizzare numericamente il sistema. Si utilizza Matlab per dichiarare e ricavare il valore di ogni parametro. Si ottengono i valori riportati nella Tabella 4.

L'espressione della funzione di trasferimento numerica è riportata nell'espressione 19:

$$\frac{\theta_b}{s\theta_w(s)} = \frac{-73.87s}{145.2s^2 - 7598} \tag{19}$$

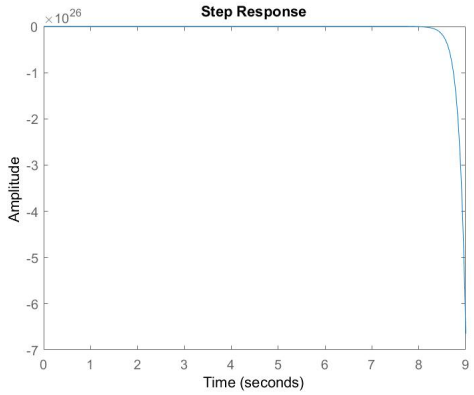


Figura 2: Risposta al gradino della funzione di trasferimento a ciclo aperto

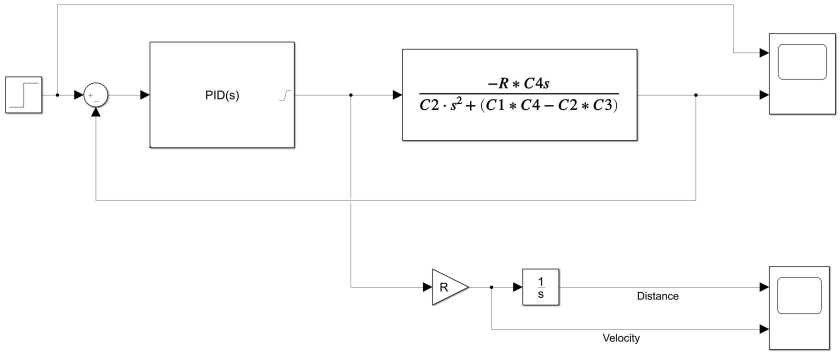


Figura 3: Schema a blocchi del modello del sistema principale

I cui poli a ciclo aperto sono $poles_{openloop} = \pm 7.2335$, da cui si evince la natura instabile del sistema con un polo a parte reale positiva. Ciò è anche apprezzabile dalla risposta al gradino riportata in Fig. 2. La natura instabile del sistema era già comunque facilmente intuibile fisicamente. Si procede con la realizzazione di un controllore PID per rendere il sistema stabile e capace di garantire determinate specifiche progettuali.

Successivamente, si è simulato il sistema retroazionato, senza controllore. Infine, è stato implementato anche il controllore PID. Uno schema a blocchi del sistema retroazionato con controllore PID è stato generato e simulato tramite Simulink, visibile in Fig. 3. Si riporta in Fig. 4 le risposte al gradino dei tre sistemi simulati, a ciclo aperto, con sistema retroazionato e con controllore PID. Tramite PID Tuner, sono stati tarati i valori dei tre parametri Kp, Ki e Kd, per i quali è stata ottenuta la risposta al gradino in Fig. 4.

Si evidenzia nello schema a blocchi realizzato in Simulink la presenza di uno Scope, quest'ultimo utile per rappresentare graficamente, con le dovute conversioni, la distanza e la

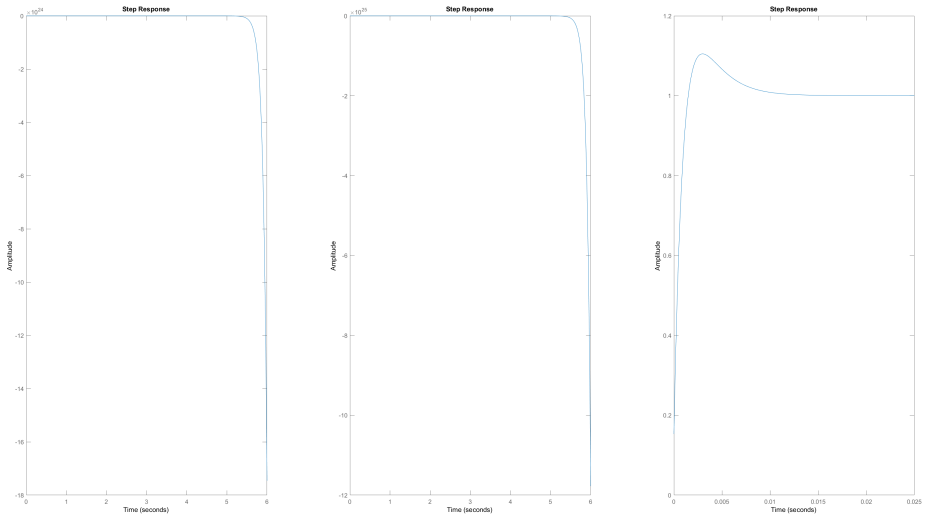


Figura 4: Risposta al gradino PID

velocità in funzione del tempo, riportate in Fig. 5. Il segnale a gradino ha un'ampiezza pari a 2° (0.035 radianti).

Come atteso, la simulazione ha dimostrato che il robot ha risposto muovendosi in avanti con una velocità che aumenta linearmente. Ciò mantiene l'angolo di inclinazione del corpo a 2 gradi. Si nota inoltre che la velocità continuerà ad aumentare se si mantiene un angolo diverso da zero.

4 Progettazione e realizzazione dell'elettronica e del firmware

Sulla base delle caratteristiche elettriche, di alimentazione e comunicazione degli elementi selezionati e definiti nei capitoli precedenti, si progetta un'architettura del sistema, di cui si riporta nella Fig.6 uno schema a blocchi semplificato.

4.1 Progettazione dell'elettronica

Si caratterizzano alcuni aspetti matematici e fisici legati al controllo della rotazione delle ruote, per definire e chiarire le funzioni che andranno poi ad essere implementate nel firmware. Successivamente, di descrivono brevemente le caratteristiche del sensore IMU.

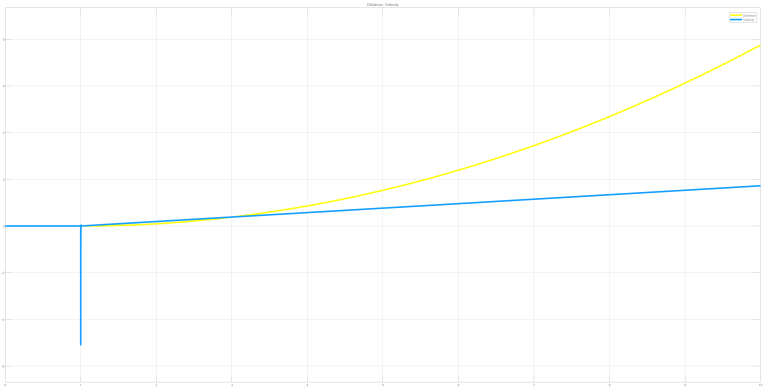


Figura 5: Distanza e velocità in funzione del tempo

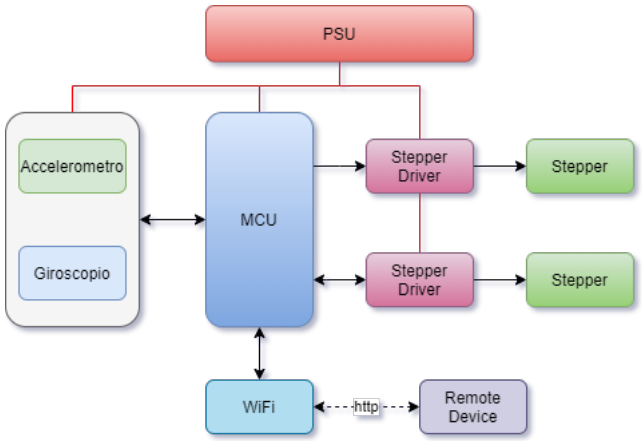


Figura 6: Schema a blocchi principale del sistema

4.1.1 Attuazione: Motori Stepper e Driver

Un motore stepper (o motore passo-passo) è un motore elettrico sincrono in corrente continua pulsata con gestione elettronica senza spazzole (brushless). Si caratterizza per la possibilità di suddividere la propria rotazione in un grande numero di passi (step). La posizione del motore può essere controllata accuratamente senza dover ricorrere al controllo ad anello chiuso (feedback) se la taglia ed il tipo di motore sono scelti in modo adeguato all'applicazione.

Lo stepper è formato da due principali componenti: un rotore e uno statore.

Per ottenere la rotazione del rotore, occorre inviare agli avvolgimento dello statore una serie di impulsi in corrente in modo tale da far spostare, per step o passi successivi, la posizione di equilibrio del rotore.

Siccome tensioni e correnti non sono compatibili direttamente con la tensione logica di board come la Nucleo o Arduino, si rende necessario l'uso di alcuni driver.

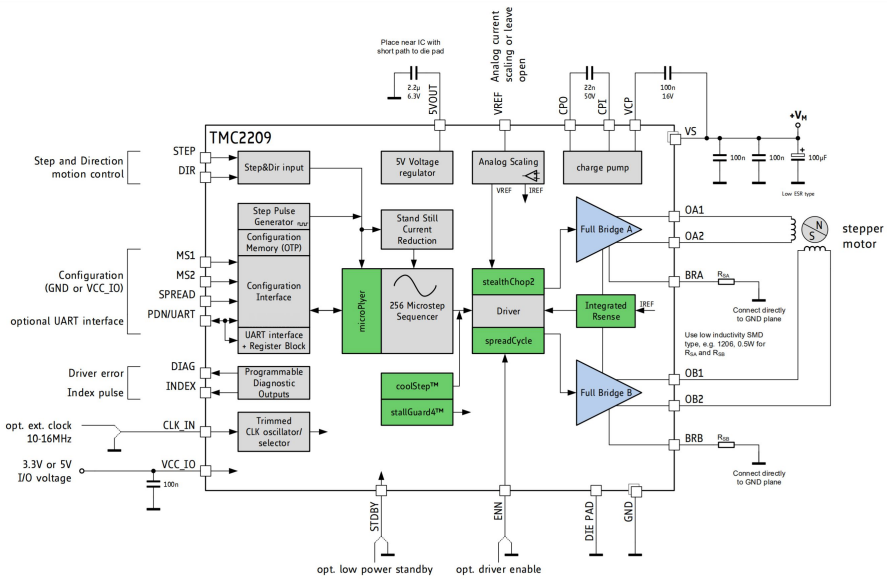


Figura 7: Schema a blocchi interno del Driver TMC2209

TMC2209: Caratteristiche principali	
Parametro	Valore
Alimentazione (V)	5.5-29
Logica I/O	3.3-5.25
Corrente max per avvolgimento (A)	1.4
Frequenza Operativa (KHz)	10-13.4
Potenza (W)	1.4

Tabella 5: Caratteristiche elettriche principali del driver TMC2209

In commercio esistono molte soluzioni, ad esempio i **DRV8825** oppure i **A4988**. Per questo progetto, sono stati selezionati i **TMC2209** della Trinamic, driver ad alte prestazioni generalmente usati come upgrade per i precedenti in contesti quali stampanti 3D o CNC. Uno schema a blocchi interno del TMC2209 è visibile in Fig. 7.

Un riferimento sulle caratteristiche elettriche invece è riportato nella Tabella 5.

Considerati gli aspetti energetici in gioco, si ritiene opportuno inserire dei piccoli dissipatori passivi sui chip dei driver. Gli input da fornire al driver sono la direzione (*dir*) e gli *step*, sottoforma di impulso. Ogni step può essere convertito un passo completo (*full step*) o un micropasso (*microstep*), in cui ci sono 2, 4, 8, 16, 32, 64, 128 o 256 microstep per fullstep. Una tabella interna traduce il valore del contatore nei valori di seno e coseno che controllano

MS1/MS2: Configuration of Microstep Resolution for Step Input		
MS2	**MS1**	**Microstep Setting**
GND	GND	8 microsteps
GND	VCC_IO	32 microsteps (different to TMC2208!)
VCC_IO	GND	64 microsteps (different to TMC2208!)
VCC_IO	VCC_IO	16 microsteps

Figura 8: Configurazioni possibili dei pin per il driver TMC2209

la corrente del motore per il microstepping. Questi parametri possono essere gestiti tramite interfaccia seriale, o tramite la configurazione di alcuni pin (MS1 e MS2) esposti dal driver. In quest'ultimo caso, si hanno le possibilità elencate nella tabella in Fig. 8. Di default, sono tenuti a GND internamente. La possibilità di aumentare i microstep è utile in tutti quei contesti in cui è necessaria una elevata precisione a discapito della massima velocità di rotazione, motivo per cui in questa sede si manterrà la configurazione di default a 8 microstep.

Prima di utilizzare i driver, è necessario calibrare la corrente massima in funzione di quella dichiarata per i motori, ovvero 0.3A. La R_{SENSE} dichiarata da datasheet è pari a 110 mohm. Per effettuare questa operazione, occorre alimentare il driver e mantenere a livello logico alto i pin *step* e *dir*, e regolare un trimmer saldato sulla scheda affinchè il potenziale Vref non sia ad un valore tale per cui la relazione 20: sia verificata:

$$I_{RMS} = \frac{325mV}{R_{SENSE} + 20mV} \cdot \frac{1}{\sqrt{2}} \cdot \frac{V_{ref}}{2.5V} \tag{20}$$

con $I_{RMS} = 0.3A$, da cui si ricava una tensione pari $V_{ref} = 0.43V$. Il motore stepper selezionato è caratterizzato da una risoluzione in gradi di 1.8°/step, da cui si ricava la quantita di step/rev (numero di step per effettuare una rotazione completa) pari a 200. Siccome è stato impostato un valore di microstep pari a 8, l'effettivo numero di step/rev è pari a $200 * 8 = 1600$.

4.1.2 Sensoristica: sensore IMU MPU6050

Caratterizzato il macroblocco di attuazione, si passa a quello di sensoristica. Il sensore IMU selezionato è un **MPU6050**. Tale sensore integra sia un accelerometro che un giroscopio. L'accelerometro può essere configurato con un fondo scala pari a ±2g, ±4g, ±8g o ±16g, con una risoluzione dell'ADC interna pari a 16 bit. Il giroscopio può essere configurato con un fondo scala pari a ±250, ±500, ±1000, and ±2000°/sec, sempre a 16 bit. La trasmissione dei dati avviene tramite standard I2C. Un riferimento sulle caratteristiche elettriche è riportato nella Tabella 6.

Il sensore non richiede particolari interventi circuitali, in quanto si presenta su scheda (modello GY-521) dotata di tutti i componenti (passivi e non) per la regolazione e stabilizzazione

MPU6050: Caratteristiche principali	
Parametro	Valore
Alimentazione (V)	3.3
Logica I/O	0.5-(VDD+0.5)
Corrente (mA)	3.8
Datarate Accelerometro (Hz)	4-1000
Datarate Giroscoopio (Hz)	4-8000
Freq. I2C max (kHz)	400

Tabella 6: Caratteristiche elettriche principali del sensore IMU MPU6050

Aspects	ZigBee	WiFi	Bluetooth
standard	802.15.4	802.11a,b,g	802.15.1
application	automation, control	Web, e-mail, video	replacement of cableing
data rate	50 - 60 kbyte	> 1 Mbyte	> 250 kbyte
battery lifetime	> 1000	1 -5	1 -7
network size	65535	32	7
bandwith (kb/s)	20 - 250	11000	720
transmission distance	100+ m	100 m	10 m
advantage	reliability, performance, cost	data rate, flexibility	cost, comfortable

Figura 9: Tabella di confronto delle principali tecnologie di comunicazione wireless

della tensione, il filtraggio, e le resistenze di pull-up per i segnali I2C. Presente anche un led D1, utile per mostrare la presenza della tensione di alimentazione.

4.1.3 Comunicazione: modulo wifi ESP-8266

Per rendere il robot totalmente autonomo, senza cavi che possano disturbare il suo equilibrio e per renderlo inoltre controllabile da remoto, si opta per un sistema di comunicazione wireless. Esistono differenti soluzioni come lo standard Bluetooth, ZigBee etc. Si riporta in Figura 9 una tabella di confronto di queste principali tecnologie. Considerando gli aspetti energetici, lo standard ZigBee risulta essere il migliore. Inoltre, questo progetto prevede pochi scambi di dati, per cui ZigBee sarebbe comunque accettabile. Tuttavia, il costo di un device ZigBee è estremamente elevato, motivo per cui almeno per questo progetto viene scartato. In sviluppi futuri potrà essere sicuramente considerato come valida soluzione. La seconda scelta ricadrebbe sul Bluetooth, ma richiede lo sviluppo di un sistema per il controllo da remoto (tramite app, per esempio). Si decide di utilizzare quindi la comunicazione WiFi, adottando il modulo ESP-8266 che risulta essere di dimensioni ridotte e dal costo contenuto. Tra le caratteristiche elettriche di

Consumi componenti	
Componente	Consumo max (A)
NUCLEO-F446RE	0.5
MPU6050	0.0038
TMC2209	0.0075
Stepper Motor	0.4 (x2 coil)
ESP8266	0.170
Totale	1.4813

Tabella 7: Caratteristiche elettriche principali del sensore IMU MPU6050

interesse per il progetto, il modulo richiede una tensione di alimentazione e logica pari a 3.3V e richiede una corrente massima, nel caso di trasmissione nel caso peggiore, di 170mA.

4.1.4 Progettazione Elettronica

Per dimensionare correttamente lo stadio di alimentazione, si effettua una stima sulla richiesta energetica del sistema. Si riportano in Tabella 7 i consumi massimi di ogni componente, ricavati dai relativi datasheet.

Si può assumere un consumo teorico massimo pari a 1.5A circa. Ovviamente si tratta di una stima per eccesso del caso peggiore, ma risulta utile per determinare i parametri di alimentazione. Supponendo di voler lavorare con batterie LiPo (tipologia 18650) con tensione nominale 3.7V (totalmente cariche pari a 4.1V) e capacità pari a 2600mAh. Sono richieste due tensioni di alimentazione, ovvero 5V e 12V (l'alimentazione a 3.3V per l'IMU si ottiene dalla scheda Nucleo direttamente). Per ottenere i 12V esistono diverse soluzioni, tra cui:

1. collegare in serie 3 o 4 batterie, per aumentare la tensione;
2. Utilizzare un convertitore DC/DC Step Up

La prima soluzione richiede comunque un circuito di regolazione della tensione, in quanto varierà di molto in funzione dello stato di carica delle batterie. Inoltre si incontreranno effetti di sbilanciamento di carica, problematica tipica nell'uso di celle in serie. Inoltre aumenterà lo spazio, il peso e il costo complessivo. La seconda soluzione risulta meno invasiva, richiede meno risorse e fornisce in uscita già una tensione stabilizzata.

Si considera dunque l'uso di un **convertitore DC/DC Step Up**. Si rende necessario determinare la quantità di batterie da usare, questa volta in parallelo, per garantire un minimo

Table 1. Standard Values of R_{SET} for Common Output Voltages

V_O (Required)	R_{SET} (Standard Value)	V_O (Actual)
5.0 V	Open	5.00 V
9.0 V	4.53 kΩ	9.01 V
12.0 V	1.33 kΩ	12.03 V
15.0 V	60.4 Ω	14.99 V

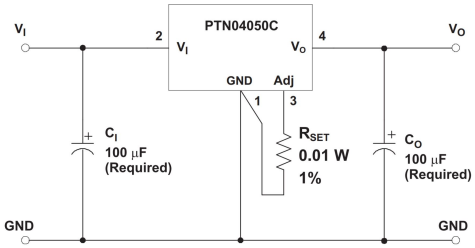


Figura 10: Circuito d'esempio e valori di uscita standard

di autonomia. Si ipotizza di voler mantenere una autonomia di un'ora. Per lo Step Up vale la relazione 21.

$$P_{in} = P_{out} + P_{LOSS} \tag{21}$$

dove P_{in} rappresenta la potenza in ingresso, P_{out} rappresenta la potenza richiesta in uscita e P_{LOSS} rappresenta le perdite. Assumendo nulle le perdite, è possibile ottenere una stima sulla richiesta di potenza:

$$P_{out} = P_{in} = V_{out} I_{out} = 12 * 1.5 = 18W \tag{22}$$

La capacità della singola batteria è 2600mAh, dunque

$$autonomia = \frac{2600mAh}{1500} = 1.73ore \tag{23}$$

ovvero circa un'ora e tre quarti. Basterà dunque una singola batteria per mantenere in autonomia il sistema per un tempo sufficiente.

Lo Step Up selezionato è un **PTN04050C** della Texas Instruments, un piccolo modulo 4 pin con uscita regolabile tramite opportuna resistenza. Nella Figura 10 viene riportato un esempio di circuito di utilizzo e una tabella con valori standard di uscita con i relativi valori di resistenza. Nel caso in esame, verrà utilizzato un resistore da 1.33 kohm per ottenere i 12V in uscita.

Al fine di realizzare uno schematico complessivo dello schematico del circuito, facilmente fruibile, si utilizza un software EDA (Electronic Design Automation). Sul mercato esistono diverse soluzioni, professionali o meno, con diverse funzionalità come servizi cloud e visione

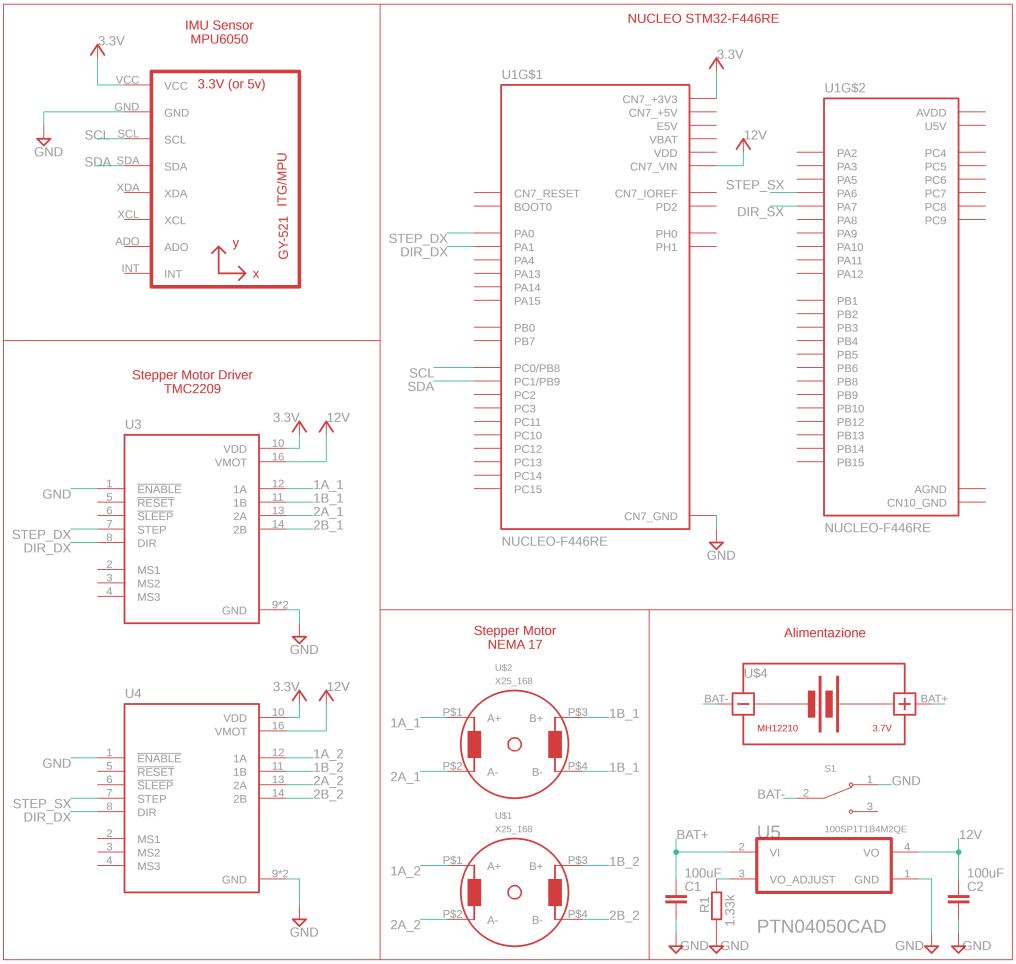


Figura 11: Schematico del circuito realizzato

e modifica dei componenti 3D.Alcuni esempi sono *Altium Designer, Autodesk Eagle, OrCad, KiCado EasyEDA*. Si sceglie di utilizzare **Autodesk Eagle** in quanto fruibile tramite licenza didattica per gli studenti, offre un sistema professionale alla progettazione elettronica. Nella Figura 11, si riporta lo schematico complessivo del sistema realizzato.

E' possibile alimentare la scheda STM32 Nucleo-F446RE tramite VIN dai pin 8 di CN6 o pin 24 di CN7 con una tensione da 7V a 12V e una corrente massima di 800mA. Per alimentare dall'esterno la scheda è necessario spostare il jumper JP5 in posizione E5V, invece per l'alimentazione da USB la posizione sarà U5V. È inoltre necessario lasciare aperto JP1 e connettere l'USB dopo aver alimentato la scheda dall'esterno per poter eseguire la programmazione. Tutte queste informazioni sono riportate nell'**User Manual UM1724** per le schede STM32 Nucleo-64, di cui si illustra un estratto in Figura 12.

Table 7. External power sources

Input power name	Connectors pins	Voltage range	Max current	Limitation
VIN	CN6 pin 8 CN7 pin 24	7 V to 12 V	800 mA	From 7 V to 12 V only and input current capability is linked to input voltage: 800 mA input current when Vin = 7 V 450 mA input current when 7 V < Vin <= 9 V 250 mA input current when 9 V < Vin <= 12 V
E5V	CN7 pin 6	4.75 V to 5.25 V	500 mA	-

Table 8. Power-related jumper

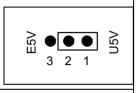
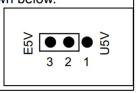
Jumper	Description
JP5	U5V (ST-LINK VBUS) is used as a power source when JP5 is set as shown below (Default setting) 
	VIN or E5V is used as a power source when JP5 is set as shown below. 

Figura 12: Specifiche della scheda STM32 Nucleo-F446RE per l'alimentazione esterna

Si interpone tra il negativo della batteria e il riferimento a massa dello Step Up un interruttore switch per permettere l'accensione dell'intero sistema, scollegando totalmente la batteria nel caso in cui rimanga chiuso.

Infine, si inserisce un modulo di ricarica per la batteria della Adafruit, basato sull'integrato MCP73831, dotato di connettore e cavo JST, ingresso 5V tramite connettore USB Micro-B. La corrente di carica di default è impostata a 100mA, è possibile regolarla a 500mA saldando un ponticello. Ai fini del progetto è sufficiente mantenere la corrente di ricarica a 100mA. Questo modulo di piccole dimensioni permette la ricarica della batteria Lipo senza doverla estrarre: l'uscita è infatti direttamente saldata ai capi del battery holder mentre l'altra estremità è connessa al modulo tramite connettore JST. In fase di utilizzo, può essere facilmente scollegato.

Si riporta in Fig. 13 il sistema realizzato. La struttura su cui è stata integrata l'elettronica è realizzata tramite stampa 3D con tecnica FDM, in materiale PLA. Il design è stato riadattato da un progetto open source differente sempre di self balancing robot. Si è reso ovviamente riarrangiare la disposizione del microcontrollore, della batteria e dei vari elementi.

4.2 Progettazione del firmware

Il firmware per la scheda STM32 Nucleo-F446RE è stato sviluppato nell'IDE (Integrated Development Environment) **STM32CubeIDE** alla versione 1.6.0, con il supporto sull'inizializzazione delle varie periferiche del software integrato **STM32CubeMX**. In Figura 14 si riporta

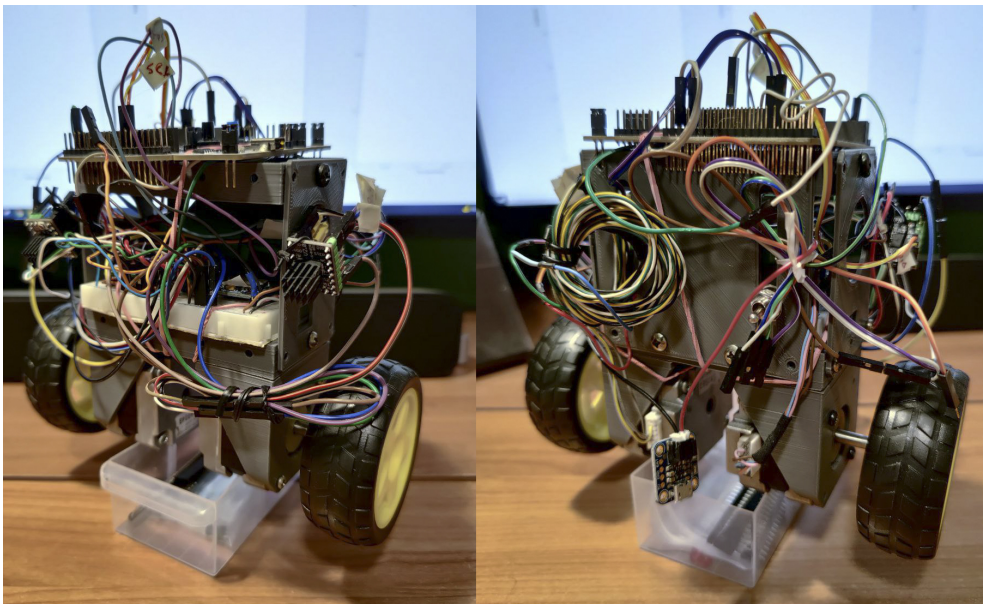


Figura 13: Fronte e retro del prototipo di robot realizzato

il pinout del microcontrollore, con le periferiche utilizzate.

In particolare, vengono utilizzati due Timer (**TIM2** e **TIM3**) per il controllo dei singoli servomotori e TIM6 per la gestione di eventuali operazioni legate al movimento (ad esempio, avanza di 50 cm). Sono stati implementati anche alcune interfacce di comunicazione, tra cui l'I2C per la lettura dei dati dal sensore IMU e due USART, la **USART1** per la comunicazione con il modulo wifi dei comandi e la **USART2** (aperta di default) per la fase di debug con l'IDE. Entrambe le USART vengono utilizzate in modalità Interrupt.

TIM2 e TIM3 sono configurati in modalità **Output Compare**, con *Prescaler*=83 e *Counter Period*=99. La sorgente del segnale di clock è configurata a 84MHz. In questo modo, si ottiene un segnale con frequenza pari a 10kHz (100ns), ordine di grandezza compatibile con i segnali accettati dagli stepper e soprattutto dai driver TMC2209. Questi segnali andranno a controllare la velocità di rotazione degli stepper. Per poter avere un controllo sul questi segnali, al fine di adattare dinamicamente la velocità alla situazione, si abilita l'**Auto-Reload Preload**. Per ogni Timer, si utilizza un solo canale (*Channel 1*) in modalità **Toggle on match**, ottenendo così un segnale ad onda quadra con frequenza predefinita.

Ricordando che gli step per rivoluzione determinati dalla caratterizzazione del sistema stepper+driver era *1600 step/rev*, considerando la caratteristica di toggle, tale valore va moltiplicato per 2 ottenendo un valore effettivo di step per rivoluzione pari a *3200 step/rev*.

TIM6 viene configurato invece in modalità standard, con *Prescaler*=8399 e *Counter Pe-*

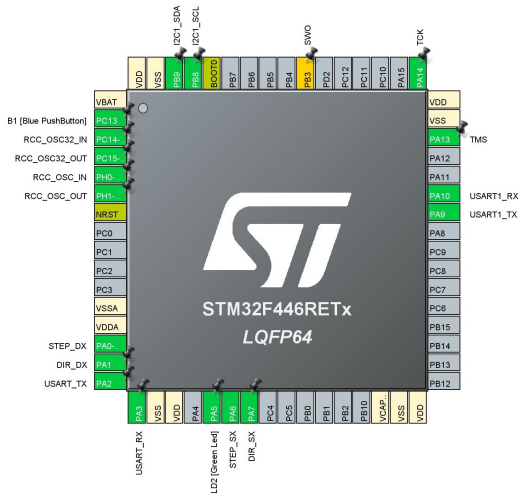


Figura 14: Pinout del microcontrollore

Interfacce utilizzate		
TIM2	controllo servo SX	Output Compare
TIM3	controllo servo DX	Output Compare
TIM6	gestione movimento	Standard Mode
I2C1	lettura dati IMU	Standard Mode
USART1	modulo WiFi	
USART2	Debug	

Tabella 8: Configurazione interfacce

riod=9999. In questo modo, si ottiene un segnale con frequenza pari a 1Hz (1 secondo), ordine di grandezza adatto a determinare la velocità angolare (gradi/secondo) (o indirettamente una velocità tangenziale) da assumere impostando una determinata distanza da percorrere. Tutti i Timer sono implementati in modalità Interrupt. La periferica I2C è configurata in Standard Mode, con frequenza di clock pari a 100KHz.

Si riporta in Tabella 8 un riepilogo delle interfacce utilizzate

Si richiede di caratterizzare i valori massimi e minimi di velocità. Ciò richiede un espressione che leghi la velocità con l'**Auto Reload Register** (**ARR**). Valutando i parametri impostati dei Timer, si può ottenere l'espressione 24 in funzione della variabile x.

$$\frac{84MHz}{84 * 100 * x} = \frac{10kHz}{x} = \frac{10^{-4}}{x} \tag{24}$$

da cui è possibile ricavare la formula 25, che esprime la quantità di secondi per rivoluzione.

$$secondi\ per\ rivoluzione = 3200 * \frac{10^{-4}}{x} = \frac{0.32}{x} \tag{25}$$

Infine, si ricava l'espressione 26 per la velocità angolare

$$\dot{\theta_w} = \frac{2\pi}{secondi\ per\ rivoluzione} = \frac{2\pi}{0.32x} \tag{26}$$

Il codice prevede una prima fase di configurazione e inizializzazione di tutte le periferiche, tramite funzioni generate grazie al *Device Configuration Tool Code Generation* di CubeMX.

Vengono definite alcune funzioni **IRQHandler** per tutti i componenti che utilizzano gli interrupt, oltre che ad alcune funzioni di callback per gestirne l'evento.

```
void USART2_IRQHandler(void);
void USART1_IRQHandler(void);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle);
void TIM2_IRQHandler(void);
void TIM3_IRQHandler(void);
void TIM6_DAC_IRQHandler(void);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
```

In particolare, la callback della UART viene usata per alzare un flag e richiamare un'altra funzione all'interno del loop principale per la ricezione e lettura dei dati, svincolando così l'interrupt. La callback del timer, chiamata allo scadere del TIM6, ha il compito invece di disattivare la generazione dei segnali per i motori, fermando di fatto il robot.

4.2.1 Gestione motori

Per la gestione dei motori, è stata scritta una libreria dedicata. Vengono dichiarati alcuni parametri fisici e costruttivi delle ruote, degli stepper e dei driver. Le funzioni implementate sono le seguenti:

```
void setDirection(uint8_t dir);
void getDirection(float VELOCITY);
void setSpeed(float speed);
void setSpeedPID(float speedPID);
void simpleGO(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3);
void goForward(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3, TIM_HandleTypeDef* htim6, ...
               ... float VELOCITY, float metri);
void goBackward(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3, TIM_HandleTypeDef* htim6, ...
```

```
                ... float VELOCITY, float metri);
void rotateL(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3, TIM_HandleTypeDef* htim6, ...
             ... float VELOCITY, float rad);
void rotateR(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3, TIM_HandleTypeDef* htim6,...
             ... float VELOCITY, float rad);
void stopMotor(TIM_HandleTypeDef* htim2, TIM_HandleTypeDef* htim3, TIM_HandleTypeDef* htim6);
```

Con la funzione ***setDirection()*** si fornisce semplicemente il relativo segnale ai driver degli stepper. Con la funzione ***getDirection()*** si ricava la direzione dall'uscita del PID. Con la funzione ***setSpeed()*** si applica una determinata velocità agli stepper, passata manualmente. La funzione ***setSpeedPID()*** applica la velocità ottenuta dal PID, implementando l'espressione 26. La funzione ***simpleGO()*** viene attivata con la fase di test in autonomia, per cui il sistema avanza indefinitamente in funzione delle letture del sensore e della regolazione del PID. Con la funzione ***goForward()*** si avanza in avanti per una distanza predeterminata. Con la funzione ***goBackward()*** si avanza in dietro per una distanza predeterminata. Con la funzione ***rotateL()*** si ruota a sinistra per un angolo predeterminato. Con la funzione ***rotateR()*** si ruota a destra per un angolo predeterminato. Con la funzione ***stopMotor()*** si bloccano i motori.

4.2.2 Acquisizione sensore IMU e Complementary Filter

Per la gestione del sensore IMU MPU6050, è stata scritta una libreria dedicata. La libreria ***MPU6050.h*** prevede la definizione dell'indirizzo I2C del dispositivo (0xD0) oltre che l'indirizzo dei principali registri. Vengono inoltre definite tutte le variabili necessarie per l'acquisizione e conversione delle misure. Infine, si implementano le seguenti funzioni:

```
void MPU6050_Init (I2C_HandleTypeDef* hi2c1);
void MPU6050_getStatus(I2C_HandleTypeDef* hi2c1);
void MPU6050_Read_Accel (I2C_HandleTypeDef* hi2c1);
void MPU6050_Read_Gyro (I2C_HandleTypeDef* hi2c1);
void MPU6050_Calibration(I2C_HandleTypeDef* hi2c1);
```

Nella funzione ***MPU6050_Init()*** si effettua un controllo sul dispositivo, controllando il registro **WHO_AM_I**: se risponde correttamente con il valore 104 (o 0x68) si procede con l'inizializzazione. Questa fase prevede la configurazione, tramite scrittura di determinati valori, di alcuni registri del sensore. Per attivare il sensore, viene posto a zero tutto il registro di Power Management (0x6B). Successivamente, si imposta il data rate del sensore a 1kHz. Infine, si setta a 2g l'accelerometro e 250°/s il giroscopio.

La funzione ***MPU6050_getStatus()*** controlla semplicemente che alla chiamata del registro WHO_AM_I, il sensore risponda correttamente. Ciò può tornare utile per controllare che il sensore risponda e non abbia problemi.

Con la funzione ***MPU605_Read_Accel()*** si implementa la lettura del valore di accelerazione.

```
HAL_I2C_Mem_Read (hi2c1, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data_Accel, 6, 1000);
Accel_X_RAW = (int16_t)(Rec_Data_Accel[0] << 8 | Rec_Data_Accel [1]);
Accel_Y_RAW = (int16_t)(Rec_Data_Accel[2] << 8 | Rec_Data_Accel [3]);
Accel_Z_RAW = (int16_t)(Rec_Data_Accel[4] << 8 | Rec_Data_Accel [5]);
```

Si richiamano i valori di accelerazione per i 3 assi, rappresentato ognuno con un valore a 16 bit. Risulta necessario richiamare la funzione HAL_I2C_Mem_Read() per 6 volte, in quanto ogni lettura porta 8 bit di dato.

Successivamente, si convertono i valori grezzi ottenuti precedentemente in accelerazione espressa in 'g'.A tal fine, si dividono i valori raw per il valore di fondo scale configurato precedentemente nel registro dell'IMU **FS_SEL**. Siccome FS_SEL = 0, si divide per 16384.0.

La funzione è utilizzata sia in fase iniziale di calibrazione che durante il ciclo di vita del codice. Per identificare questi due momenti operativi, viene utilizzata la variabile ***MPU6050_Calib_status***.

Nel caso in cui non si è in fase di calibrazione, è possibile ricavare l'angolo assunto rispetto ad ogni asse dai valori di accelerazione precedentemente calcolati. Considerando i gradi di libertà con cui il robot può muoversi, si ricavano solo 2 dei 3 angoli. A tal fine, si richiama la funzione matematica **atan2()**, fornita dalla libreria *math.h*.

```
roll=(double)( (atan2(Ay, sqrt(Az*Az+Ax*Ax)))* (180.0 / M_PI) );
pitch=(double)( (atan2(Ax, sqrt(Az*Az+Ay*Ay)))* (-180.0 / M_PI) );
```

La funzione ***MPU6050_Read_Gyro()*** implementa la stessa fase di acquisizione dei dati grezzi.

```
HAL_I2C_Mem_Read (hi2c1, MPU6050_ADDR, GYRO_XOUT_H_REG, 1, Rec_Data_Gyro, 6, 1000);
Gyro_X_RAW = (int16_t)(Rec_Data_Gyro[0] << 8 | Rec_Data_Gyro [1]);
Gyro_Y_RAW = (int16_t)(Rec_Data_Gyro[2] << 8 | Rec_Data_Gyro [3]);
Gyro_Z_RAW = (int16_t)(Rec_Data_Gyro[4] << 8 | Rec_Data_Gyro [5]);
```

Anche in questo caso, si convertono i valori grezzi in gradi al secondo (°/s) dividendo per il valore di fondo scala, precedentemente configurato e pari a 131.0 per il giroscopio con FS_SEL=0.

Anche qui si valuta lo stato della variabile ***MPU6050_Calib_status***: se si è in fase di calibrazione, il valore ottenuto è quello definitivo. Altrimenti si procede con la compensazione dei valori letti rispetto a quelli iniziali (letti appunto in fase di calibrazione).

```
Gx = Gx - Gx_0;
Gy = Gy - Gy_0;
Gz = Gz - Gz_0;
```

Infine, si convertono i valori espressi in gradi al secondo (°/s) in semplici gradi, effettuando una operazione di *integrazione nel discreto*, considerando dTime come l'intervallo di tempo tra due acquisizioni.

```
Gx_angle += (Gx * dTime/1000);
Gy_angle += (Gy * dTime/1000);
Gz_angle += (Gz * dTime/1000);
```

L'ultima funzione implementata è quella di calibrazione, ovvero ***MPU6050_Calibration()***. In questa funzione si acquisiscono una sequenza di valori (100 in questo caso) e si ricava una media aritmetica. Ciò risulta importante soprattutto per il giroscopio che, come si vedrà, risulta soggetto a *errori cumulativi*.

Dalle funzioni sopra descritte e implementate, si ottengono i valori in termini di angoli di tutti gli assi. Si potrebbe dunque pensare di utilizzare semplicemente uno dei due valori ottenuti, o dall'accelerometro o dal giroscopio. Tuttavia, entrambe le misure hanno dei problemi intrinsecamente irrisolvibili.

Generalmente da questi sensori è richiesta una misura in angoli. Come visto, per ottenere l'informazione utile dai giroscopi, è necessario integrarla per ottenere un valore angolare. Anche piccoli errori nella lettura, a causa dell'integrazione possono causare una crescita (accumulo) nei valori finali. Perciò bisogna caratterizzare ed eliminare questi errori, effettuando dei periodici reset.

L'accelerometro invece fornisce una misura in termini di accelerazione, da cui si può facilmente ricavare l'angolo formato, come visto precedentemente.

Purtroppo l'accelerometro lavora bene a basse frequenze ma risulta molto sensibile alle vibrazioni ad alta frequenza.

Dunque, sia i dati dell'accelerometro che quelli del giroscopio sono soggetti a ***errori sistematici***. L'accelerometro fornisce dati precisi a lungo termine, ma è rumoroso a breve termine. Il giroscopio fornisce dati precisi sul cambiamento di orientamento a breve termine,

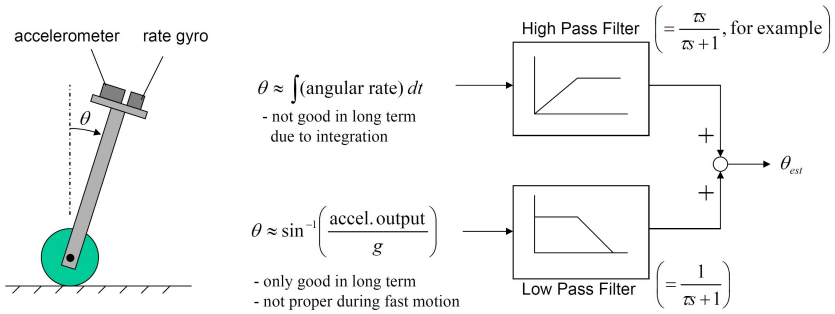


Figura 15: Schema logico di un Complementary Filter per la stima dell'angolo

ma l'integrazione necessaria fa sì che i risultati abbiano un effetto di ***drift*** su scale temporali più lunghe.

La soluzione a questi problemi è fondere insieme i dati dell'accelerometro e del giroscopio in modo tale che gli errori si annullino. Questa tecnica viene chiamata ***Sensor Fusion***.

Una tecnica molto usata per combinare questi due input è con un **filtro di Kalman**, ma risulta piuttosto complessa. Un'altra tecnica, più semplice ed usata per questo progetto, per combinare questi due tipi di dati adotta un ***Complementary Filter***.

Alcuni esempi di applicazione di un Complementary Filter sono riportate in diversi articoli e ricerche [4] [17].

Il *Complementary Filter* è una combinazione di un filtro passa alto e passa basso. La funzione di questo filtro è quello di utilizzare gli aspetti positivi di un input per compensare i difetti dell'altro, e viceversa (da cui complementare). Per cui, il passa basso filtrerà le fluttuazioni dell'accelerometro mentre il passa alto ridurrà gli effetti di drift del giroscopio. In questo modo, si otterrà un dato in uscita più accurato e soprattutto soggetto a meno variazioni e oscillazioni.

Uno schema complessivo di implementazione del Complementary Filter è riportato in Figura 15.

Per l'implementazione software, è stata definita la seguente funzione

```
void complementaryFilter(void){
        angleX = (0.97 * (angleX+Gy*(dTime/1000))) + 0.03*pitch;
        angleY = (0.97 * (angleY+Gx*(dTime/1000))) + 0.03*roll;
}
```

da cui si può notare l'estrema semplicità implementativa. Da notare la presenza, per i due termini dell'accelerometro e giroscopio, di due coefficienti: si tratta di veri e propri *pesi*,

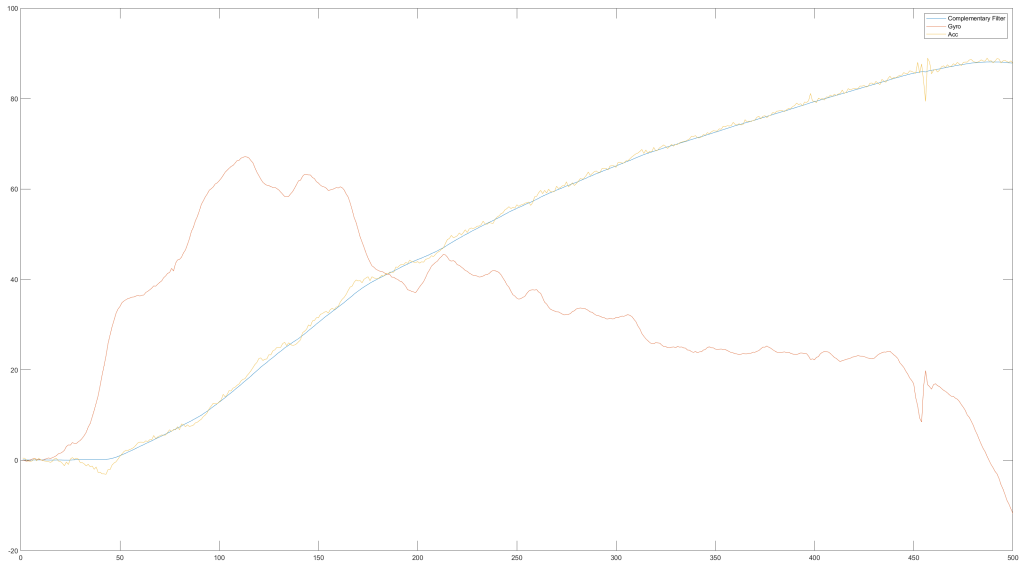


Figura 16: Acquisizione delle misure di angoli tramite Matlab

utili per definire quanta importanza dare a un termine rispetto che all'altro. La somma dei pesi deve necessariamente fare 1.

Dopo l'implementazione sono stati condotti alcuni test per verificare l'effettivo funzionamento del sistema di acquisizione dell'angolo. Una prima prova è stata effettuata considerando gli angoli ottenuti dal solo giroscopio, dal solo accelerometro e dal complementary filter: sono stati trasmessi tramite UART per poter essere apprezzati tramite il Serial Monitor dell'IDE di Arduino. In questo modo si è verificato in maniera semplice e veloce che, facendo assumere una pendenza arbitraria, gli angoli (soprattutto quello in uscita al filtro) seguissero fedelmente e senza disturbi la pendenza. Una successiva prova, per avere una stima globale dell'andamento delle misure, è stata effettuata acquisendo i dati tramite Matlab, per poi essere stampati sotto forma di grafico. Si è portato il Robot da una situazione stabile a zero gradi, ad assumere una angolazione di 90°. Si riporta in Figura 16 il grafico ricavato.

Dalla figura 16 si evince il comportamento supposto per via teorica e si conferma il funzionamento del sistema di acquisizione.

4.2.3 Implementazione PID

Per la gestione dei motori, è stata scritta una libreria dedicata. La libreria ***PID.h*** prevede l'implementazione di una struttura PID, come proposta di seguito:

```
typedef struct {
        /* Controller gains */
```

```
        float Kp, Ki, Kd;
        /* Derivative low-pass filter time constant */
        float tau;
        /* Output limits */
        float limMin, limMax;
        /* Integrator limits */
        float limMinInt, limMaxInt;
        /* Sample time (in seconds) */
        float T;
        /* Controller "memory" */
        float integrator, prevError, differentiator, prevMeasurement;
        /* Controller output */
        float out;
} PIDController;
```

oltre che due funzioni, una per l'inizializzazione del PID ed un'altra per l'aggiornamento dello stesso.

```
void  PIDController_Init(PIDController *pid);
float PIDController_Update(PIDController *pid, float setpoint, float measurement);
```

La funzione di inizializzazione del PID setta semplicemente tutte le variabili utilizzate a zero. La funzione di update invece implementa tutta la logica, aggiornando le variabili e l'uscita del regolatore in funzione del valore precedente dell'ingresso e del *setpoint*. A tal fine, si calcola inizialmente l'errore tra la misura e il setpoint e si calcolano i tre contributi (Proporzionale, Integrale, Derivativo). All'interno di questa funzione è già integrata quella di **Anti-wind-up** per limitare (saturare) il valore dell'effetto integrale. Inoltre, viene ulteriormente limitato il valore dell'uscita complessiva del PID.

4.2.4 Comunicazione WiFi: Esp8266

Come discusso in precedenza, si utilizza per la comunicazione da remoto il modulo Wifi Esp8266.

Obiettivo è quello di realizzare una interfaccia minimale basata su chiamate ***http***.

Per la programmazione di tale dispositivo si è utilizzato l'IDE di ***Arduino***, alla versione 1.8.13. L'IDE non riconosce nativamente la scheda in questione, per cui è necessario installare tali informazioni tramite il gestore schede. Si riporta una guida dettagliata all'installazione, configurazione ed utilizzo iniziale della scheda [1].

Le librerie utilizzate sono:

1. **ESP8266WiFi.h** [6].
2. **ESPAsyncTCP.h** [15].
3. **ESPAsyncWebServer.h** [16].

Forniti i parametri per la connessione (SSID e password del wifi), una volta resettat l'ESP si potrà consultare tramite indirizzo IP da qualsiasi web browser. Tramite delle semplici e specifiche richieste **GET** sarà possibile inviare tramite la seriale dell'ESP il comando desiderato. Ciò avviene implementando le seguenti righe di codice per ogni comando:

```
server.on("/go", HTTP_GET, [] (AsyncWebServerRequest * request) {
Serial.write('G');
request->send(200, "text/plain", "GO eseguito");
});
```

4.2.5 Main Loop

Sfruttando tutte le librerie e funzioni descritte precedentemente, il *Main Loop* risulta notevolmente snellito e di facile lettura.

Inizialmente, si effettua il calcolo dell'intervallo di tempo impiegato ad ogni ciclo. Successivamente, si chiama la funzione relativa alla UART per valutare la presenza di nuovi comandi in arrivo dal modulo wifi o comunque dalla seriale. Sulla base del nuovo comando in arrivo, si determina l'operazione da eseguire tramite il seguente *switch case*:

```
switch(readBuf[0]){
        case STOP:
                stopMotor(&htim2,&htim3, &htim6);
                break;
        case GO:
                goForward(&htim2, &htim3, &htim6, VELOCITY, 0.1);
                break;
        case BACK:
                goBackward(&htim2, &htim3, &htim6, VELOCITY, 0.2);
                break;
        case LEFT:
                rotateL(&htim2, &htim3, &htim6, VELOCITY, M_PI/2);
```

```
                break;
        case RIGHT:
                rotateR(&htim2, &htim3, &htim6, VELOCITY, M_PI/2);
                break;
        case TEST:
                simpleGO(&htim2, &htim3);
                break;
}
```

Si interroga il sensore IMU, controllando che funzioni normalmente tramite la funzione *MPU6050_getStatus()*. Successivamente si acquisiscono i segnali dell'accelerometro e giroscopio, passandoli successivamente al *Complementary Filter*. L'angolo filtrato in uscita viene fornito come valore in input al regolatore PID, assieme al setpoint che, per mantenere l'equilibrio, è impostato a 0.0°.

Dal PID si otterrà in uscità il valore di velocità angolare, da aggiornare alla funzione *setSpeedPID()*. Inoltre, dal valore in uscita del PID si ricava l'informazione sulla direzione da assumere tramite la funzione *getDirection()*, applicandola con *setDirection()*.

Infine, si inoltrano i valori di angolo e velocità calcolati nel ciclo tramite UART, per poi riprendere il ciclo dall'inizio.

Riferimenti bibliografici

[1] A. Alecce. *Costruisci la tua rete domotica con Esp8266 e Raspberry Pi – Primi passi con il modulo wifi ESP8266*. Mar. 2020. URL: https://antima.it/costruisci-la-tua-rete-domotica-con-esp8266-e-raspberry-pi-primi-passi-con-il-modulo-wifi-esp8266/.

[2] Md. Iman Ali e Md. Modasser Hossen. «A two-wheeled self-balancing robot with dynamics model». In: *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)*. 2017, pp. 271–275. DOI: 10.1109/ICAEE.2017.8255365.

[3] Wei An e Yangmin Li. «Simulation and control of a two-wheeled self-balancing robot». In: *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2013, pp. 456–461. DOI: 10.1109/ROBIO.2013.6739501.

[4] Nur Hazliza Ariffin, Norhana Arsad e Badariah Bais. «Low cost MEMS gyroscope and accelerometer implementation without Kalman Filter for angle estimation». In: nov. 2016, pp. 77–82. DOI: 10.1109/ICAEES.2016.7888013.

[5] *Ascento Robotics*. URL: https://www.ascento.ch/.

[6] *ESP8266WiFi library — ESP8266 Arduino Core 3.0.1-1-g421d02ee documentation*. 2017. URL: https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html.

[7] F. *Modeling and Control of Two-Wheels Self Balancing (TWSB) Robot (Updated 27 Dec 2020)*. Giu. 2021. URL: https://fkeng.blogspot.com/2019/03/theory-and-design-of-two-wheels-self.html.

[8] P Frankovský et al. «Modeling of Two-Wheeled Self-Balancing Robot Driven by DC Gearmotors». In: *International Journal of Applied Mechanics and Engineering* 22.3 (2017), pp. 739–747. DOI: 10.1515/ijame-2017-0046.

[9] C Gonzalez, I Alvarado e DML Peña. «Low cost two-wheels self-balancing robot for control education». In: *IFAC-PapersOnLine* 50.1 (2017), pp. 9174–9179. DOI: 10.1016/j.ifacol.2017.08.1729.

[10] Juang Hau-Shiue e Kai-Yew Lum. «Design and control of a two-wheel self-balancing robot using the arduino microcontroller board». In: giu. 2013, pp. 634–639. ISBN: 978-1-4673-4707-5. DOI: 10.1109/ICCA.2013.6565146.

[11] Osama Jamil et al. «Modeling, control of a two-wheeled self-balancing robot». In: apr. 2014, pp. 191–199. ISBN: 978-1-4799-5132-1. DOI: 10.1109/iCREATE.2014.6828364.

[12] A Jiménez, F Jiménez e I Ruge. ≪Modeling and Control of a Two Wheeled Self-Balancing Robot: a didactic platform for control engineering education.≫ In: *Proceedings of the 18th LACCEI International Multi-Conference for Engineering, Education, and Technology: Engineering, Integration, And Alliances for A Sustainable Development" "Hemispheric Cooperation for Competitiveness and Prosperity on A Knowledge-Based Economy"* (2020). DOI: 10.18687/laccei2020.1.1.556.

[13] Sangtae Kim e SangJoo Kwon. ≪Dynamic Modeling of a Two-wheeled Inverted Pendulum Balancing Mobile Robot≫. In: *International Journal of Control, Automation and Systems* 13 (ago. 2015). DOI: 10.1007/s12555-014-0564-8.

[14] Fabian Kung. ≪A Tutorial on Modelling and Control of Two- Wheeled Self-Balancing Robot with Stepper Motor≫. In: *Applications of Modelling and Simulation* 3 (lug. 2019).

[15] M. *me-no-dev/ESPAsyncTCP*. 2019. URL: https://github.com/me-no-dev/ESPAsyncTCP.

[16] M. *me-no-dev/ESPAsyncWebServer*. 2021. URL: https://github.com/me-no-dev/ESPAsyncWebServer.

[17] Kartik Madhira, Ammar Gandhi e Aneesha Gujral. ≪Self balancing robot using complementary filter: Implementation and analysis of complementary filter on SBR≫. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. 2016, pp. 2950–2954. DOI: 10.1109/ICEEOT.2016.7755240.

[18] Sophan Nawawi. ≪Real-Time Control System for a Two-Wheeled Inverted Pendulum Mobile Robot≫. In: nov. 2010. ISBN: 978-953-307-141-1. DOI: 10.5772/10362.