

UNIVERSITÀ POLITECNICA DELLE MARCHE
INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE
Stabilità del quadricottero



Corso di
LABORATORIO DI AUTOMAZIONE
Anno accademico 2023-2024

Studenti:
Gianluca Eremita
Ettore Ricci
Carlo Marchetti

Professore:
Andrea Bonci

Dottorando:
Alessandro Di Biase



Dipartimento di Ingegneria dell'Informazione

Indice

Introduzione	2
1 Sistema	4
1.1 Modello matematico	5
1.1.1 Sistemi di riferimento	5
1.1.2 Caratterizzazione dinamica	9
1.2 Modello di controllo	10
2 Hardware	12
2.1 Scheda STM32 H745 Nucleo-144	12
2.2 Motori	13
2.3 ESC	15
2.4 Batteria	17
2.5 Eliche	18
2.6 IMU	19
2.7 PDB	20
2.8 Radiocomando	21
2.9 Schema dei collegamenti	22
2.10 Prova dei motori	23
3 Software	24
3.1 Diagrammi di flusso	24
3.1.1 Diagramma di flusso degli stati di i	24
3.1.2 Diagramma di flusso del ciclo while	24
3.2 Gestione singoli componenti	26
3.2.1 Gestione TIM1	26
3.2.2 IMU	28
3.2.3 PWM	31
3.2.4 RADIOCOMANDO	35
3.2.5 PID	40
3.3 Funzionamento complessivo	43
4 Test e risultati	47
4.1 Scelta degli offset e dei coefficienti dei PID	47
4.2 Simulazioni	47
4.2.1 Simulazione n.36	48
4.2.2 Simulazione n.12	48
4.3 Simulazione n.26	49
4.3.1 Simulazione n.44	49
4.3.2 Simulazione n.55	50

Conclusioni e sviluppi futuri	51
Appendici	52
I Matlab	52

Introduzione

Il nostro obiettivo per questo progetto è stato quello di implementare un controllo automatico in grado di mantenere in equilibrio un drone quadricottero. Per raggiungere tale obiettivo ci siamo serviti di: quattro motori, ognuno connesso ad un'elica, il sensore IMU e l'ausilio di un radiocomando per armare i motori, farli partire e spegnerli. Dopo aver analizzato quanto fatto dal gruppo precedente, ci siamo occupati di modificare la parte hardware e la parte software.

In particolare a livello hardware, dopo alcune considerazioni, abbiamo modificato la modalità di avvio degli ESC, passando da *normal* e *very soft*.

Inoltre abbiamo abilitato i canali 1 e 2 del radiocomando che permettono di modificare il riferimento dei PID e quindi di poter mantenere stabile il drone non solo sullo zero.

A livello di codice (software), invece, abbiamo aggiunto un controllo al duty del PWM, cambiato la formula per il calcolo del duty e aggiunto alcune funzionalità al radiocomando.

1 Sistema

All'interno di questo capitolo si va ad analizzare nello specifico cos'è un quadricottero, esaminando la sua struttura, il modello matematico su cui si basa e il modello di controllo con cui viene gestito.

Un drone quadricottero, o quadrirotore, è un aeromobile dotato di quattro eliche orizzontali equamente distanziate dal corpo centrale. Le generazioni più moderne, anche grazie alla loro dimensione ridotta, sono progettate per volare utilizzando un sistema di controllo e dei sensori che permettono il volo sia in ambienti interni che esterni. Nel nostro caso non si avrà esperienza di volo libero, ma il nostro obiettivo sarà il raggiungimento e mantenimento dello stato di equilibrio del dispositivo. Le eliche sono azionate a due a due con lo stesso verso di rotazione sullo stesso asse, nel nostro caso 1 e 3 in un verso e 2 e 4 nell'altro (Figura (1)). Dunque, se i motori operano tutti alla stessa velocità, i momenti generati dalle coppie di motori si annullano a vicenda, permettendo al drone di mantenere una direzione durante il volo. L'imbardata, il beccheggio e il rollio, infatti, sono controllati variando le coppie, quindi le velocità di una determinata coppia di rotori: Nel caso dell'imbardata il momento torcente generato dalla prima coppia deve essere contrastato dal momento relativo all'altra coppia di rotori. Questo si traduce in somme di velocità diverse per coppie di rotori. Per quanto riguarda il beccheggio si andranno a modificare le velocità dei motori 1 e 3 mantenendo invariata la loro somma. Per il rollio si andranno a modificare le velocità dei motori 2 e 4 mantenendo invariata la loro somma. L'altitudine è controllata variando la spinta di ogni singolo motore di una stessa quantità.

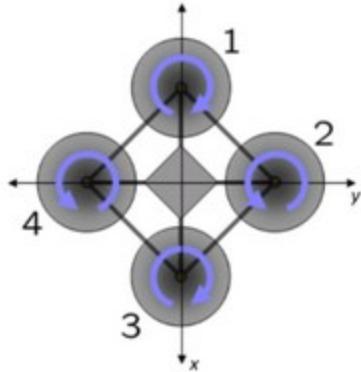


Figure 1.1: Verso di rotazione dei propulsori

Il quadricottero ha una struttura simmetrica, nel nostro caso con un telaio che ha una configurazione a croce. Ogni braccio ospita un motore e un'elica, i quali sono incaricati di garantire la spinta e la stabilità del dispositivo. Il corpo centrale contiene l'elettronica di controllo composta da:

- Una PDB (Power Distribution Board), scheda elettronica progettata per distribuire l'alimentazione elettrica in modo efficiente e sicuro a tutti i componenti del veicolo
- Un sensore IMU, che misura e fornisce informazioni riguardo all'orientamento
- Scheda STM32 H745 NUCLEO-144, posta nella parte superiore del drone
- Quattro ESC, uno per ogni braccio, collegati al controllore STM, per gestire la velocità del relativo motore collegato

La struttura è progettata per essere leggera in modo da garantire la possibilità di volare, ma allo stesso tempo robusta per garantire maggior maneggevolezza e resistenza ad eventuali urti.

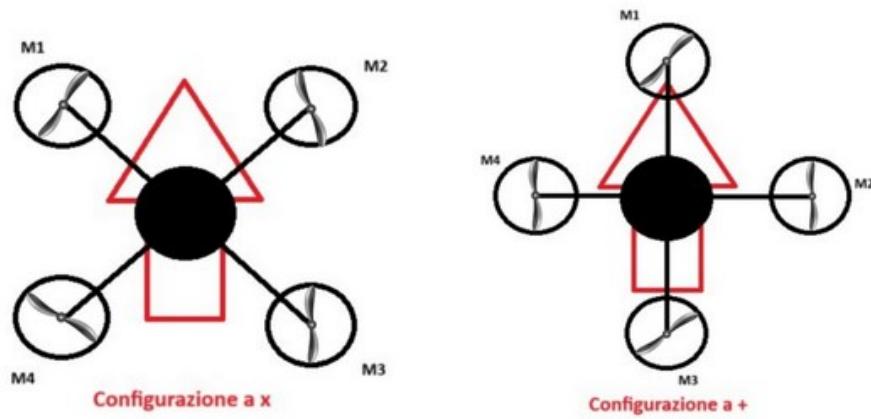


Figure 1.2: Configurazioni droni

Nella Figura 1.2 possiamo notare le differenze tra i due tipi di configurazione del drone, a croce e ad x, queste due configurazioni differiscono per come i motori (M1,M2,M3,M4) vengono accoppiati sullo stesso asse del piano cartesiano e per come il drone si sposta in avanti (freccia rossa) rispetto alla propria struttura

1.1 Modello matematico

1.1.1 Sistemi di riferimento

Introduciamo i sistemi di riferimento necessari per la descrizione dell'assetto e della posizione del drone e che ci porteranno alla definizione delle equazioni che descrivono il veivolo. Possiamo, nel nostro caso, definire due sistemi di riferimento:

$$E_I = [x \ y \ z]$$

$$E_B = [x_B \ y_B \ z_B]$$

Dove:

- EI è il sistema di riferimento inerziale assoluto (solidale al suolo);
- EB è il sistema di riferimento solidale al telaio del drone, l'origine di EB è posta nel centro di massa.

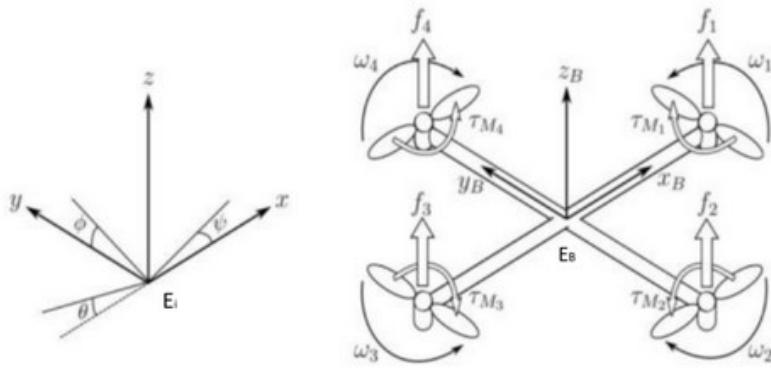


Figure 1.3: sistema inerziale assoluto e sistema inerziale solidale al telaio del drone

Dove abbiamo che:

ϕ = Roll

θ = Pitch

ψ = Jaw

f_i = forza di spinta dei rotori

ω_i = velocità angolari dei motori

M_i = velocità di drag dei motori

Indichiamo ora con ϵ la posizione assoluta del drone e con η il suo assetto nel sistema inerziale E_I . Nell'assetto, in particolare, posso indicare l'angolo di roll ϕ (rotazione intorno all'asse x), l'angolo di pitch θ (rotazione intorno all'asse y) e l'angolo di yaw ψ (rotazione attorno all'asse z). Ottenendo così:

$$\epsilon = [x \ y \ z]^T$$

$$\eta = [\phi \ \theta \ \psi]^T$$

Introduciamo ora due vettori, il vettore rappresentante le velocità lineari V_B e il vettore delle velocità angolari ω del sistema di riferimento inerziale solidale al corpo del drone E_B .

$$V_B = [u \ v \ w]^T$$

$$\omega = [p \ q \ r]^T$$

Supponiamo che il drone sia perfettamente simmetrico, rispetto gli assi x e y. In questo modo gli assi principali d'inerzia coincidono con gli assi di simmetria e abbiamo che $I_{xx} = I_{yy}$

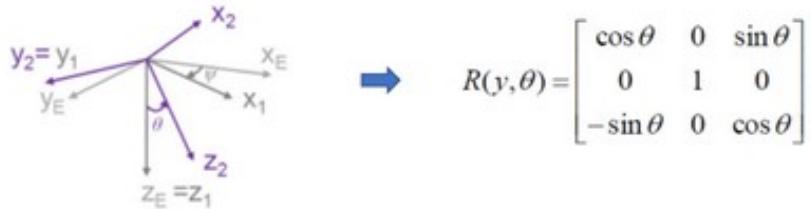
$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{yy} \end{bmatrix}$$

A questo punto utilizzeremo l'approccio Newton-Eulero, che è una metodologia utilizzata nell'analisi del moto di corpi rigidi. Questo approccio combina concetti derivati dalle leggi del moto di Newton con la teoria dei momenti angolari di Eulero. Lo utilizzeremo per definire il comportamento dinamico del drone, per ciò abbiamo bisogno di matrici di rotazione, utili per definire l'orientamento dell'assetto del sistema solidale al telaio riferito al sistema di riferimento inerziale assoluto.

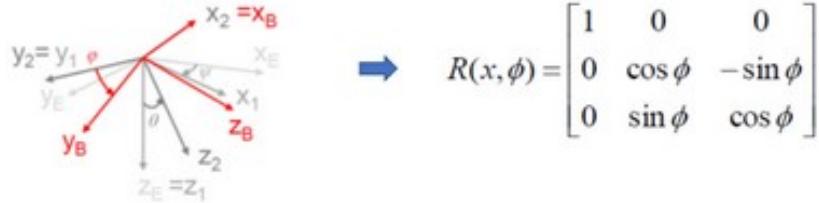
1. YAW (Rotazione attorno all'asse Z dell'angolo ψ attraverso $R(z, \psi)$)



2. PITCH (Rotazione attorno all'asse Y dell'angolo θ attraverso $R(y, \theta)$)



3. ROLL (Rotazione attorno all'asse X dell'angolo ϕ attraverso $R(y, \phi)$)



La composizione delle rotazioni precedenti dà origine ad una matrice di rotazione che permette di rappresentare l'assetto del telaio E_B rispetto al sistema inerziale E_I

$$R = R(x, \phi)R(y, \theta)R(z, \psi) = R_\phi R_\theta R_\psi =$$

$$R(\phi, \theta, \psi) = \begin{bmatrix} \cos \psi \cos \theta & \cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi & \cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi \\ \sin \psi \cos \theta & \sin \psi \sin \theta \sin \phi - \cos \psi \cos \phi & \sin \psi \sin \theta \cos \phi - \sin \phi \cos \psi \\ -\sin \theta & \cos \theta \sin \phi & \cos \theta \cos \phi \end{bmatrix}$$

R = matrice di rotazione da E_B ad E_I

Oraabbiamo che rispetto al sistema di riferimento solidale al telaio, la velocità angolare è caratterizzata dal seguente vettore: $\omega_{E_B} = [p \ q \ r]^T$ e dividendo per le tre rotazioni di base (roll, pitch e yaw), in modo da ottenere

$$\omega = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \omega_{roll} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \omega_{pitch} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega_{yaw}$$

Possiamo scrivere la matrice che mette in relazione i vettori di velocità angolare tra il sistema di riferimento E_I ed il sistema E_B e viceversa:

$$\begin{bmatrix} \phi' \\ \theta' \\ \psi' \end{bmatrix} = T \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = T^{-1} \begin{bmatrix} \phi' \\ \theta' \\ \psi' \end{bmatrix}$$

dove T e T^{-1} sono definite come:

$$T = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & -\sin \theta \\ 0 & \cos \phi & \sin \phi \cos \theta \\ 0 & -\sin \phi & \cos \phi \cos \theta \end{bmatrix}$$

Possiamo infine ricondurre le velocità angolari e lineari alle corrispondenti posizioni assolute e assetto del drone attraverso la matrice di rotazione R e la matrice T^{-1} .

Infatti:

$$\epsilon' = RV_B \quad \eta' = T^{-1}\omega$$

1.1.2 Caratterizzazione dinamica

Di seguito analizziamo le caratteristiche dinamiche del drone. Una volta analizzate le velocità angolari del sistema di riferimento solidale al telaio, si vuole capire come agiscono sul sistema le forze esterne F e i momenti esterni M . Per quanto riguarda le forze esterne che agiscono sul sistema facciamo riferimento alla dinamica traslazionale, in particolare alla seconda legge di Newton:

$$\epsilon'' \cdot m = \sum_{i=1}^N F_i = S$$

dove $F_i = b\omega^2$ rappresenta la forza fornita da ogni motore, la quale si oppone alla forza di gravità, infatti il coefficiente b è detto coefficiente di spinta dell'elica. La sommatoria delle forze dei quattro motori genera una spinta S lungo l'asse z di E_B .

$$S = \sum_{i=1}^4 F_i = b \sum_{i=1}^4 \omega_i^2 \quad S = \begin{bmatrix} 0 \\ 0 \\ S \end{bmatrix} \quad (1.1)$$

Per quanto riguarda i momenti del sistema totale ci basiamo sulla legge di Eulero:

$$\eta'' \cdot I = \sum_{i=1}^N M_i = M$$

L'effetto dato dalla rotazione dell'elica è la creazione di un momento torcente M_i attorno all'asse del motore, pari a: $M_i = d\omega_i^2 + I_M \omega_i$ con d detto coefficiente di resistenza aerodinamica e I_M momento d'inerzia del motore.

Analizziamo gli effetti delle coppie di forze generate dai diversi motori:

- Le spinte dei motori 2 e 4 si contrappongono con fulcro passante per l'asse x , è chiaro che se una delle due spinte prevale sull'altra si ha una variazione dell'angolo di roll ϕ
- Possiamo dire lo stesso per i motori 1 e 3, i quali, però si contrappongono con fulcro passante per l'asse y e uno sbilanciamento delle spinte provocherà una variazione dell'angolo di pitch θ .
- Infine i motori ruotano a 2 a 2 in senso opposto: se le somme delle coppie motrici opposte si egualgiano non si ha una variazione dell'angolo di yaw ψ . Nel caso particolare $\omega_1 = \omega_2 = \omega_3 = \omega_4$ infatti, i momenti torcenti da essi generati si annullano fra loro.

Formulando matematicamente questi concetti si ottiene, M_B :

$$M_B = \begin{bmatrix} M_\phi \\ M_\theta \\ M_\psi \end{bmatrix} = \begin{bmatrix} lb(-\omega_2^2 + \omega_4^2) \\ lb(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^4 M_i \end{bmatrix}$$

dove per l si intende la distanza tra ogni motore e il centro di massa ed è pari a 35 cm.

1.2 Modello di controllo

Dopo aver analizzato la relazione tra velocità di rotazione e momenti che i motori generano, possiamo stabilire le 4 uscite U_i (virtual input) che i controllori dovranno generare per porre il drone nella posizione e nell'assetto desiderato:

$$U_1 = b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$$

la spinta S che i motori devono generare per spostare il drone lungo l'asse z

$$U_2 = lb(-\omega_2^2 + \omega_4^2)$$

momento che modifica l'inclinazione sull'asse x, ovvero provoca una variazione dell'angolo ϕ

$$U_3 = lb(-\omega_1^2 + \omega_3^2)$$

momento che modifica l'inclinazione sull'asse y, ovvero provoca una variazione dell'angolo θ

$$U_4 = b(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2)$$

momento torcente totale, utilizzato per far ruotare il drone attorno l'asse z, ovvero provoca una variazione dell'angolo ψ

(b = coefficiente di spinta, d = coefficiente di resistenza aerodinamica, l = distanza tra motore e centro di massa)

Poniamo i quattro ingressi in forma matriciale ottenendo:



$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -lb & 0 & lb \\ lb & 0 & -lb & 0 \\ -d & d & -d & d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

Invertiamo ora la matrice in modo da ottenere i valori di rotazione dei singoli propulsori, poiché gli ingressi non possono essere inviati direttamente ai motori:



$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & 0 & \frac{1}{2lb} & -\frac{1}{4d} \\ \frac{1}{4b} & -\frac{1}{2lb} & 0 & \frac{1}{4d} \\ \frac{1}{4b} & 0 & -\frac{1}{2lb} & -\frac{1}{4d} \\ \frac{1}{4b} & \frac{-1}{2lb} & 0 & \frac{1}{4d} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

Thrust Roll Pitch Yaw

I coefficienti b e d , sono necessari per tarare i PID nel modo corretto. In particolare il coefficiente di spinta b può essere definito come:

$$b = \frac{S}{m \cdot v} = \frac{S}{\rho \cdot A \cdot v^2}$$

Dove:

- S è la spinta a cui sono sottoposte le eliche del quadricottero durante la rotazione
- m è la massa del flusso d'aria che attraversa l'elica, esprimibile come il prodotto tra la densità dell'aria (ρ), la superficie (circolare) che si forma durante la rotazione (A) e la velocità (v)
- v è la velocità del rotore, esprimibile come prodotto tra la velocità di rotazione (ω) e il raggio della pala (r)

Quindi b diventa:

$$b = \frac{S}{\rho \cdot A \cdot (\omega \cdot r)^2}$$

e nel nostro caso corrisponde al valore di 0.00001163, dove:

$$S = 50 \% \text{ del thrust complessivo}$$

$$r = 0,127m$$

$$\omega = 36606 \frac{rad}{s}$$

$$\rho = 1,225 \frac{kg}{m^3}$$

Mentre la resistenza aerodinamica d può essere definita come:

$$d = \frac{1}{2} \cdot C_d \cdot \rho \cdot A \cdot v^2$$

Con C_d che rappresenta il coefficiente di resistenza aerodinamica, e nel nostro caso d vale 0.000008, dove si ha che:

$$C_d = \frac{d}{q_0 \cdot S} = 1,19 \cdot 10^{-10}$$

$$r = 0,127m$$

$$\omega = 36606 \frac{rad}{s}$$

$$\rho = 1,225 \frac{kg}{m^3}$$

2 Hardware

2.1 Scheda STM32 H745 Nucleo-144

La scheda di sviluppo STM32 H745 Nucleo-144 è una scheda di sviluppo basata su microcontrollori a 32 bit prodotta da STMicroelectronics. Questa scheda fa parte della famiglia di microcontrollori Nucleo, che offre prestazioni elevate e una vasta gamma di funzionalità per applicazioni embedded avanzate. La scheda Nucleo-144 è denominata così a causa del suo layout a 144 pin, che fornisce una varietà di connessioni per periferiche e sensori. Il cuore della scheda è costituito dal microcontroller STM32H745ZIT6, che è basato su due architetture: sull'architettura ARM Cortex-M7 a 32 bit che offre una frequenza di clock fino a 480 MHz e sull'architettura ARM Cortex-M4 a 32 bit che offre una frequenza di clock di 240 MHz. La STM32 H745 Nucleo-144 è progettata per facilitare lo sviluppo rapido di applicazioni embedded, fornendo un ambiente di sviluppo integrato (IDE) e un'ampia gamma di risorse hardware. La scheda offre numerosi connettori e interfacce, tra cui GPIO, UART, SPI, I2C, USB e molti altri, che consentono di collegare facilmente sensori, display, e altre periferiche esterne. Inoltre, è dotata di una memoria Flash di grandi dimensioni, RAM, e molte altre risorse, rendendola adatta per applicazioni complesse.

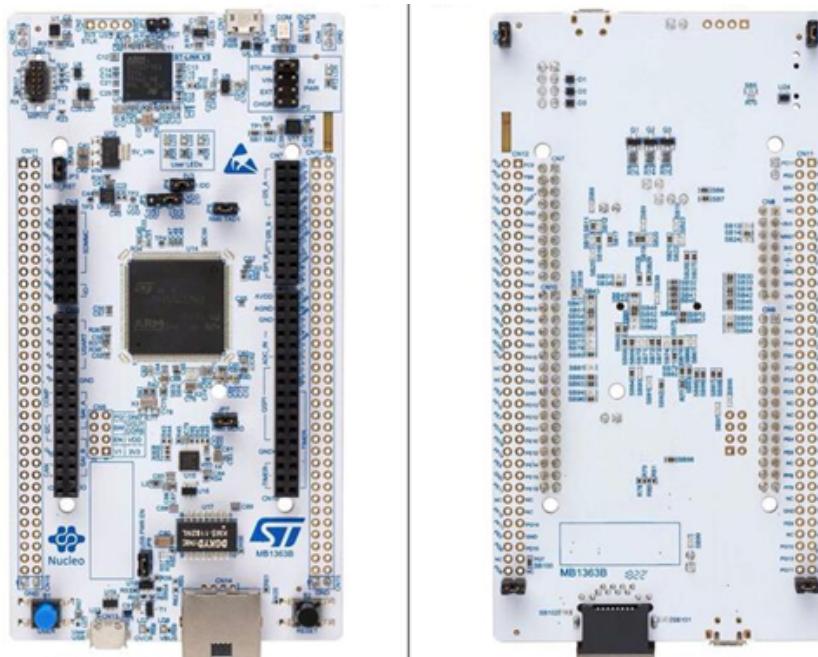


Figure 2.1: fronte e retro della scheda STM144 H745 Nucleo-144

2.2 Motori

I motori brushless sono dispositivi elettrici ampiamente utilizzati in diverse applicazioni, dalle modellistiche agli ambiti industriali. In questo contesto, la serie di motori brushless Turnigy D3536/9 910KV è particolarmente interessante per le sue caratteristiche e prestazioni. Di seguito andremo ad analizzare il funzionamento e come li abbiamo impiegati nel nostro caso specifico. Il quadricottero utilizzato è dotato di quattro motori brushless Turnigy D3536/9 910KV. Questo tipo di motore appartiene alla famiglia dei motori di classe 35, con un diametro di 35 mm e una lunghezza di 36 mm. La sigla "910KV" indica il valore della costante di velocità, ovvero il numero di giri del motore al minuto, per ogni volt applicato senza carico. Altre specifiche chiave includono la corrente massima continua, la corrente di picco e la resistenza interna, queste informazioni sono cruciali per determinare la compatibilità del motore con vari carichi elettrici e batterie. Per quanto riguarda il loro funzionamento, in generale, i motori brushless, sono dispositivi elettronici che convertono l'energia elettrica in energia meccanica senza l'uso di spazzole meccaniche. Il funzionamento dei motori brushless è basato su principi elettromagnetici e richiede un controllo elettronico preciso. Sono costituiti da due principali componenti: il rotore, la parte rotante del motore, che può essere costituita da magneti permanenti e lo statore, la parte fissa del motore, composta da avvolgimenti di filo di rame o bobine. Il funzionamento avviene tramite una commutazione elettronica, infatti il motore funziona grazie a un controllo elettronico che commuta (cambia) la polarità dell'alimentazione delle bobine del motore in modo sincronizzato con la rotazione del rotore. Questa commutazione avviene attraverso un dispositivo noto come regolatore di velocità (ESC), che è in grado di fornire il giusto segnale elettronico alle bobine in base alla posizione del rotore. Nello specifico si distinguono due principali fasi del funzionamento di un motore brushless: la fase di eccitazione e la rotazione. La prima avviene quando una bobina è alimentata con corrente elettrica, perciò si genera un campo magnetico e a seconda della polarità, il rotore verrà attratto o respinto da questo campo; la seconda, invece, avviene quando il controllo elettronico commuta la corrente tra le diverse bobine in modo sincronizzato, inducendo la rotazione del rotore. Questa tecnologia offre numerosi vantaggi, tra cui una maggiore efficienza, una maggiore durata e una minor manutenzione rispetto ai motori tradizionali con spazzole.



Figure 2.2: Motore brushless Turnigy D3536/9 910kV e la sua struttura interna

Di seguito vengono descritte corrente massima, tensione massima e resistenza, le quali rap-

presentano alcune specifiche chiave dei motori brushless. Questi sono parametri importanti che influenzano le prestazioni e l'affidabilità del motore.

Corrente massima - La corrente massima rappresenta il massimo flusso di corrente elettrica che il motore può gestire in modo continuo senza subire danni. È un parametro critico che determina la capacità del motore di gestire carichi elettrici senza surriscaldarsi o deteriorarsi.

Tensione massima - La tensione massima indica il valore massimo di tensione elettrica che il motore può sopportare in modo sicuro. Superare questa tensione può danneggiare irreparabilmente il motore.

Resistenza - La resistenza interna del motore brushless rappresenta l'opposizione che il motore offre al flusso di corrente. Una resistenza interna più bassa significa una maggiore efficienza del motore, in quanto riduce le perdite di energia sotto forma di calore durante il funzionamento.

Queste specifiche sono generalmente fornite dal produttore nelle schede tecniche del motore, forniamo di seguito la scheda tecnica dei motori utilizzati.

RPM	910kv
Fasi	3
Massima corrente	25.5A
Potenza massima	370W a 15V
Peso (compreso connettori)	102g
Batteria	2-4 Cell /7.4 14.8V
Diametro del pozzo	5mm
No corrente di carico	1.5A
Dimensioni	35x36mm
Spinta max	1050g
Dimensione prop.	7.4V/12x5 14.8V/10x7
Resistenza interna	0,063 Ohm

2.3 ESC

ESC è l'acronimo di "Electronic Speed Controller", che in italiano si traduce come "Regolatore di Velocità Elettronico". Gli ESC sono dispositivi elettronici utilizzati nei sistemi di controllo di motori elettrici, in particolare nei motori brushless, come quelli spesso impiegati in aeromodellismo, droni, veicoli elettrici e altre applicazioni. La principale funzione è quella di controllo della velocità, infatti, regolano la velocità di un motore elettrico gestendo la corrente e la commutazione delle fasi. Questo è particolarmente critico in quanto la regolazione elettronica sostituisce la funzione di commutazione meccanica delle spazzole dei motori tradizionali. Nel caso siano utilizzati per il funzionamento di droni, gli ESC comunicano con il sistema di controllo di volo per mantenere il veicolo stabile. La sua capacità di gestire la corrente e di fornire un controllo preciso della velocità contribuisce a una risposta efficiente e fluida del motore durante l'accelerazione e la decelerazione. Nel nostro caso specifico sono stati utilizzati gli ESC Turnigy Plush-30A, uno per ogni motore. Questo tipo di ESC è progettato per gestire una corrente massima continua di 30 ampere, parametro cruciale per determinare la capacità dell'ESC di alimentare il motore elettrico senza surriscaldarsi o danneggiarsi. Mentre per quanto riguarda la tensione nominale, il Turnigy Plush-30A è comunemente progettato per operare con tensioni tipiche delle batterie agli ioni di litio, come 2-4 celle LiPo (7,4-14,8V). Questo tipo di ESC include un Battery Eliminator Circuit (BEC) integrato, ovvero un circuito che fornisce tensione ausiliaria per alimentare ricevitori e servomeccanismi senza la necessità di una batteria ausiliaria.



Figure 2.3: ESC Turnigy Plush - 30A

Di seguito è fornita la scheda tecnica degli ESC utilizzati con le specifiche sopra descritte:

Amperaggio	30 A
Tensione	8.4 – 16.8 V
BEC	5.5 V /4.0 A
Tensione di Cutoff	3.2 V
Peso	24.5g
Dimensioni	36x23.5x10mm
Frequenza	48 MHz

L'ESC è programmabile, questa caratteristica consente agli utenti di personalizzare le diverse impostazioni per adattare il comportamento dell'ESC alle specifiche esigenze dell'applicazione. Le impostazioni programmabili includono:

- **Tipo di batteria:** Impostare il tipo di batteria (LiPo, NiMH, NiCd, etc.)

- **Curva di accelerazione:** Regolare la sensibilità dell'acceleratore.
- **Frenatura:** Impostare il livello di frenatura.
- **Modalità di avvio:** Impostare la modalità di avvio (Normale, Soft, Very Soft).
- **Limite di corrente:** Limitare la corrente massima erogata dall'ESC.

La programmazione di un ESC può essere eseguita in diversi modi, a seconda del modello e del produttore. Tuttavia, il metodo più comune è utilizzare un programma di configurazione o una scheda di programmazione fornita dal produttore. Nel nostro caso la programmazione degli ESC è avvenuta tramite la scheda apposita fornita dal produttore. Nella Figura(6) è mostrata la configurazione considerata migliore per tutti gli ESC.

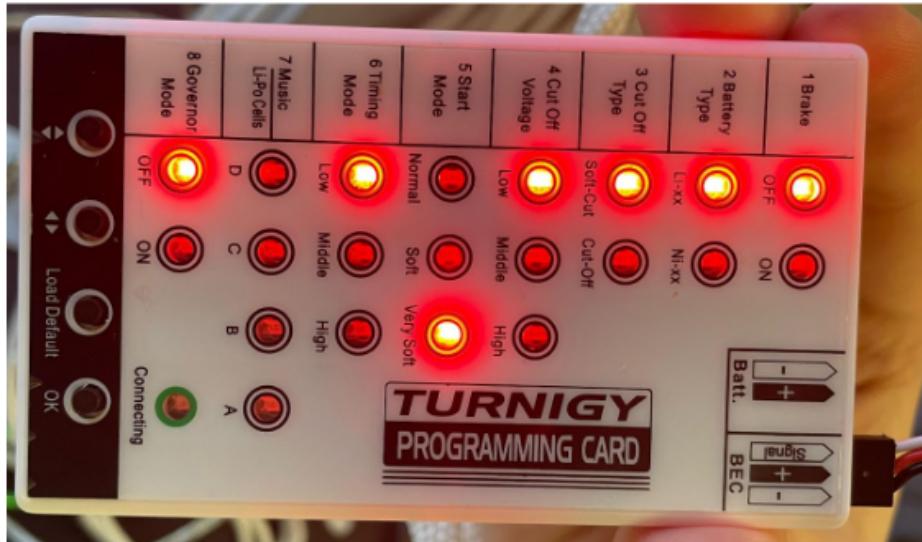


Figure 2.4: Programmatore ESC

2.4 Batteria

Nel quadricottero è stata utilizzata la batteria Turnigy A-Spec 2600mAh, una batteria a quattro celle che è comunemente utilizzata nell'ambito di droni, veicoli radiocomandati e altri dispositivi elettronici.



Figure 2.5: Batteria Turnigy A-Spec 2600mAh

Di seguito è fornita la scheda tecnica della batteria:

Capacità	2600 mAh
Tensione	4S1P / 4 cellule / 14.8 V
Peso (compreso cavo e spina)	297g
Dimensioni	129x39x28 mm

2.5 Eliche

Come accennato in precedenza il drone è dotato di quattro eliche, ciascuna associata ad un motore. La scelta adeguata del tipo di eliche è importante per garantire, insieme al contributo ovviamente essenziale dei motori, la spinta verso l'alto necessaria al drone.

Come raffigurato nello schema sottostante, le quattro eliche sono organizzate in due coppie (insieme ai motori), una delle quali ruota in senso orario mentre l'altra ruota in senso antiorario.

La scelta delle eliche più adatte al nostro progetto ha portato all'impiego delle Eliche Slow Fly Electric Prop 9057SF.

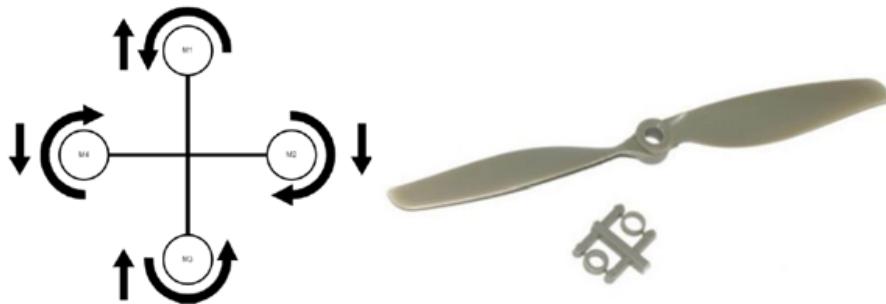


Figure 2.6: Schema del senso di rotazione delle eliche / Elica Slow Fly Electric Prop 9057SF

2.6 IMU

Un IMU, o Inertial Measurement Unit, è un dispositivo elettronico utilizzato per misurare e registrare l'accelerazione lineare e l'accelerazione angolare di un oggetto o di un veicolo in movimento. Questa componente combina una serie di sensori, tra cui giroscopi e accelerometri, al fine di monitorare i cambiamenti di posizione e velocità di un oggetto in uno spazio tridimensionale. L'IMU rileva le accelerazioni lungo gli assi X, Y e Z, oltre alle velocità angolari di rollio (roll), beccheggio (pitch) e imbardata (yaw).

Le informazioni raccolte da un IMU sono fondamentali in numerose applicazioni, ad esempio nei droni e nei veicoli autonomi. L'IMU è utilizzato per raccogliere dati utili a stabilizzare il volo o la guida, consentendo una navigazione precisa e una risposta alle variazioni dell'ambiente.

Nel nostro caso è stato impiegato l'IMU che vede un chip BNO055 montato in una scheda ADAFRUIT; tale componente, a 9 Gradi di libertà, integra un accelerometro triassiale a 14 bit, un giroscopio triassiale a 16 bit con una gamma di ± 2000 gradi al secondo, un sensore geomagnetico triassiale e un microcontrollore Cortex M0+ a 32 bit con installato il software Bosch Sensortec Fusion.

Sulla scheda sono presenti il regolatore di tensione a 3,3 V e gli adattatori di livello per i pin Reset e I2C, è anche presente un cristallo esterno da 32.768KHz da utilizzare per ottenere le migliori prestazioni.

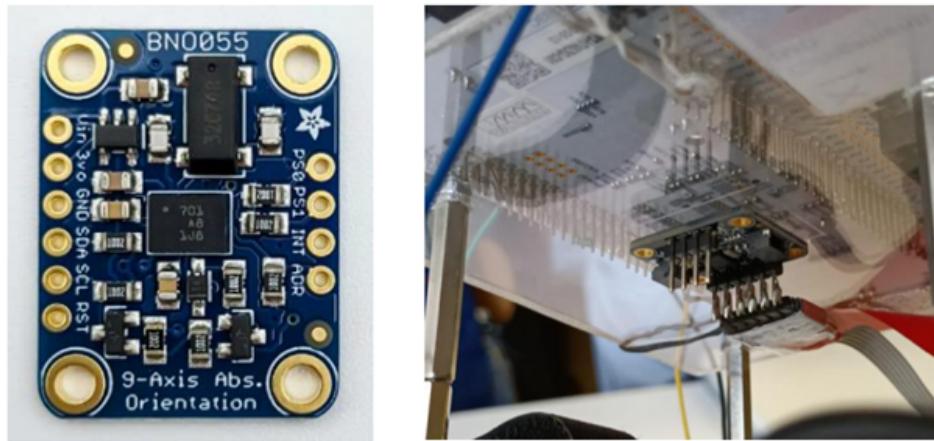


Figure 2.7: IMU / installazione IMU sul quadricottero

2.7 PDB

Il PDB (Power Distribution Board) è un componente che distribuisce in modo uniforme la tensione erogata dalla batteria ai quattro ESC. Il PDB è un componente essenziale nella progettazione di droni e velivoli radiocomandati in generale.

Il PDB dispone di una serie di connettori o piste conduttrive che collegano la batteria agli ESC e ad altri dispositivi elettronici. Questo assicura che ogni ESC riceva la stessa tensione dalla batteria, garantendo che i motori funzionino in modo coerente e che il drone mantenga un volo stabile.

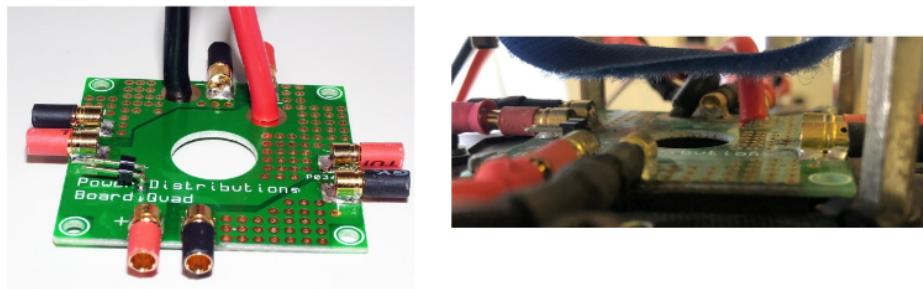


Figure 2.8: PDB / installazione PDB sul quadricottero

2.8 Radiocomando

Come ultimo componente hardware presente nel quadricottero troviamo il radiocomando; nel nostro caso viene impiegato per armare, avviare e spegnere i motori senza la necessità di avvicinare le mani ai motori in movimento, diminuendo così i rischi ed i pericoli.

Un radiocomando è composto da trasmettitore e ricevitore:

Trasmettitore (Tx): Il trasmettitore è l'unità tenuta in mano dall'utente. È dotato di leve, pulsanti e interruttori che rappresentano i comandi di controllo desiderati, come l'accelerazione, la direzione, il sollevamento e altre funzioni specifiche del dispositivo. Quando l'utente modifica una leva o preme un pulsante, il trasmettitore traduce questa azione in segnali elettrici. Questi segnali vengono poi codificati in un formato specifico e inviati tramite onde radio al ricevitore.

Ricevitore (Rx): Il ricevitore è montato sul dispositivo controllato e riceve i segnali radio trasmessi dal trasmettitore. Una volta ricevuti, il ricevitore decodifica i segnali elettrici, traducendoli nuovamente nei comandi specifici per il dispositivo. Questi comandi vengono quindi inviati ai motori del dispositivo per eseguire le azioni richieste.

L'intera operazione avviene in tempo reale, consentendo un controllo accurato e reattivo del dispositivo da parte dell'utente. La comunicazione radio avviene su frequenze specifiche assegnate per evitare interferenze con altri dispositivi e garantire una connessione stabile. Ad esempio, se l'utente utilizza l'interruttore collegato al canale 5 presente sul trasmettitore, il ricevitore tradurrà il segnale e permetterà di armare, far partire o arrestare i motori del dispositivo controllato. Inoltre, abbiamo abilitato i canali 1 e 2 del radiocomando, permettendo di modificare i valori del Pitch e del Roll tramite la levetta di destra, in modo da assegnare un riferimento diverso ai due PID.

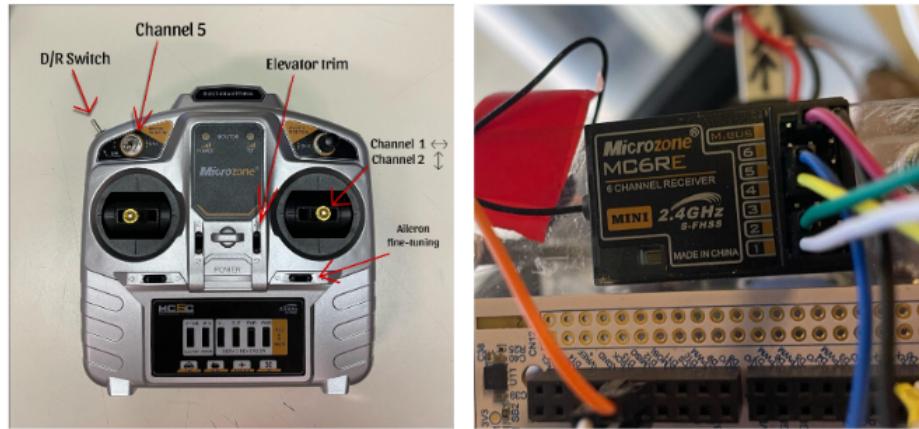


Figure 2.9: Trasmettitore e ricevitore del radiocomando

2.9 Schema dei collegamenti

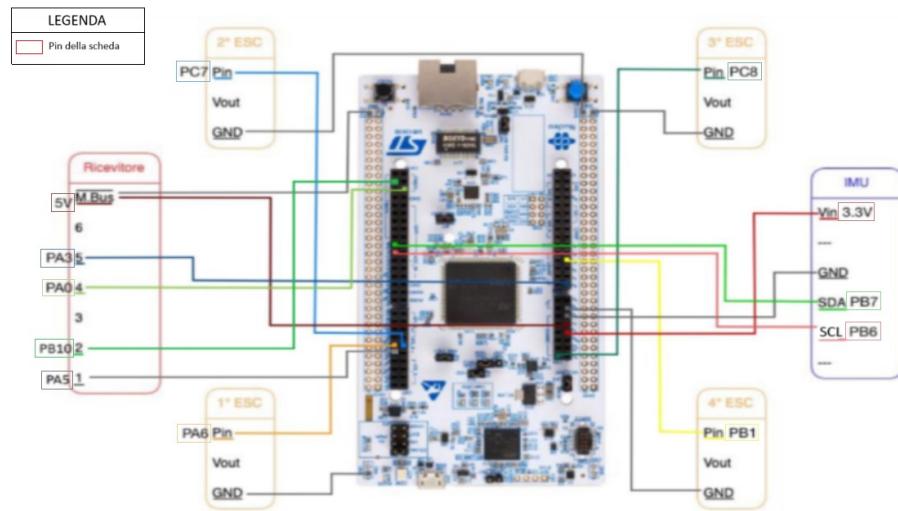


Figure 2.10: Schema dei collegamenti

La scheda STM oltre ad essere alimentata tramite usb, può essere alimentata collegando la porta Vin (sopra al collegamento ground del 4° esc); con un jumper ad una porta Vout presente in tutti gli esc.

Per vedere nello specifico tutti i collegamenti presenti nel nostro programma basta andare nella parte software di tutti i componenti.

2.10 Prova dei motori

La velocità del motore viene controllata incrementando o decrementando il duty cycle che è una misura della frazione di tempo in cui un segnale periodico è attivo o in uno stato "alto" rispetto al suo periodo totale; in altre parole, rappresenta la percentuale di tempo durante la quale il segnale è in uno stato "on" rispetto al tempo totale del ciclo. Prima che il motore entri in movimento occorre armarlo; l'armamento avviene con il 4,75% di duty cycle. Questa fase viene poi "confermata" dai "beep" dell'ESC; sette veloci ed uno più lungo.

$$PWM = DutyCycle \cdot CounterPeriod / 100$$

Sapendo che il rispettivo duty cycle equivale al 4,75% possiamo ottenere il valore in PWM necessario all'armamento:

$$PWM = 4,75 \cdot 20000 / 100$$

A questo punto il motore può iniziare a muoversi tramite un incremento del duty cycle; la percentuale minima per garantire il funzionamento del motore è del 6,2%. La percentuale massima del duty cycle che è stata impiegata è del 6,6%.

I dati sul funzionamento dei motori brushless e sulle percentuali minime e massime di duty cycle da utilizzare sono state prese da test svolti dal gruppo precedente che ha lavorato sul drone, il quale si è servito di un motore da banco per studiare il comportamento del motore in questione e capire come utilizzarlo.



Figure 2.11: Motore da banco

3 Software

Versione IDE utilizzata: 1.13.2

Versione dell'mx: 6.6.1-RC2

3.1 Diagrammi di flusso

3.1.1 Diagramma di flusso degli stati di i

Il diagramma di seguito riportato mostra attraverso quali azioni è possibile cambiare gli stati del sistema per poter effettuare le varie operazioni. E' possibile cambiare stato utilizzando la levetta, che si riferisce al channel 5 della figura 2.10

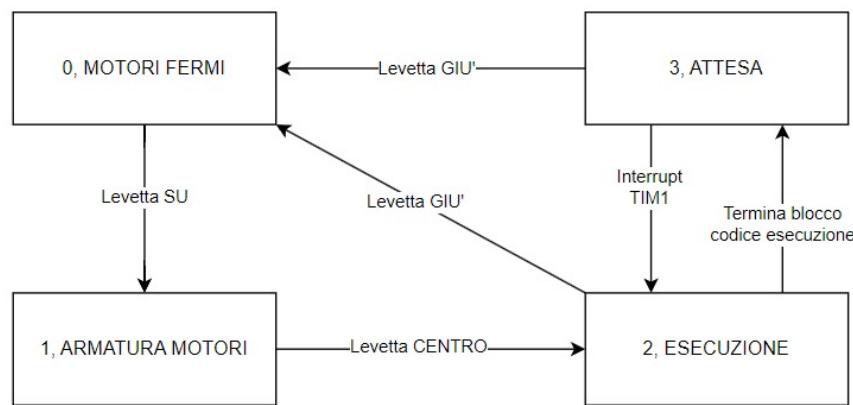


Figure 3.1: Diagramma di flusso dei stati della i

E' possibile passare anche dallo stato 1 allo stato 0 passando, obbligatoriamente, per lo stato 2 poiché la levetta deve fare due movimenti per essere settata in basso.

3.1.2 Diagramma di flusso del ciclo while

Il diagramma di seguito riportato mostra, attraverso una semplificazione, cosa e come viene eseguito all'interno del ciclo while, ovvero la parte principale del programma.

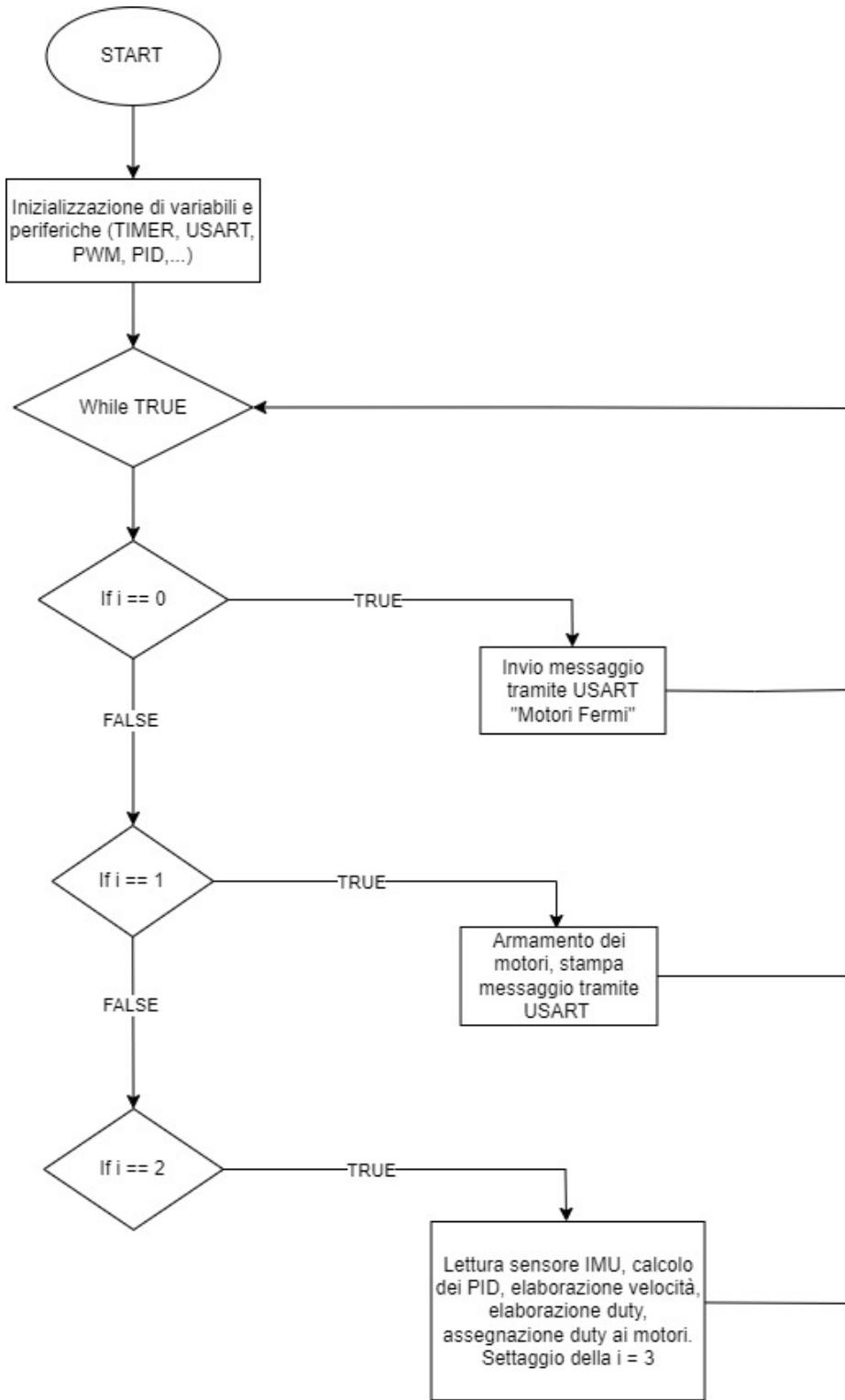


Figure 3.2: Diagramma di flusso del ciclo while

3.2 Gestione singoli componenti

Per quanto riguarda la parte software, questa si divide nell'implementazione di, in ordine cronologico, comunicazione con l'IMU, gestione degli ESC con il PWM e taratura dei PID. Otteniamo così un sistema in grado di leggere gli angoli ed elaborare la risposta con lo scopo di regolare la velocità dei motori.

3.2.1 Gestione TIM1

Come prima cosa abbiamo impostato il timer TIM1 in modalità interrupt per far sì che il codice funzioni ad una frequenza di 100Hz (periodo di 10 ms).

Vediamo successivamente come il timer è stato settato.

The screenshot shows the 'Pinout & Configuration' interface with the 'Clock Configuration' tab selected. On the left, there is a search bar and a table of categories (LPTIM1-13, RTC) with checkboxes. The row for 'TIM1' has both checkboxes checked. On the right, under 'Mode', the 'Cortex-M4' and 'D2' options are selected. Below this, various configuration parameters for TIM1 are listed, such as Slave Mode, Trigger Source, Clock Source, and Channel settings for all six channels (Channel1 to Channel6), all set to 'Disable'.

Pinout & Configuration

Clock Configuration

Pro

▼ Software Packs ▼ Pinout

Tim1 Mode and Configuration

Configuration

Reset Configuration

NVIC Settings DMA Settings
Parameter Settings User Constants

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

- Prescaler (PSC - 16 bits va..240-1)
- Counter Mode Up
- Counter Period (AutoReloa.. 10000-1)
- Internal Clock Division (CK...No Division)
- Repetition Counter (RCR - ...0)
- auto-reload preload Disable

Trigger Output (TRGO) Paramet...

- Master/Slave Mode (MSM ...Disable (Trigger input effect not dela...)
- Trigger Event Selection TR...Reset (UG bit from TIMx_EGR)

Pinout & Configuration

Clock Configuration

Pro

▼ Software Packs ▼ Pinout

Tim1 Mode and Configuration

Configuration

Reset Configuration

NVIC Settings DMA Settings
Parameter Settings User Constants

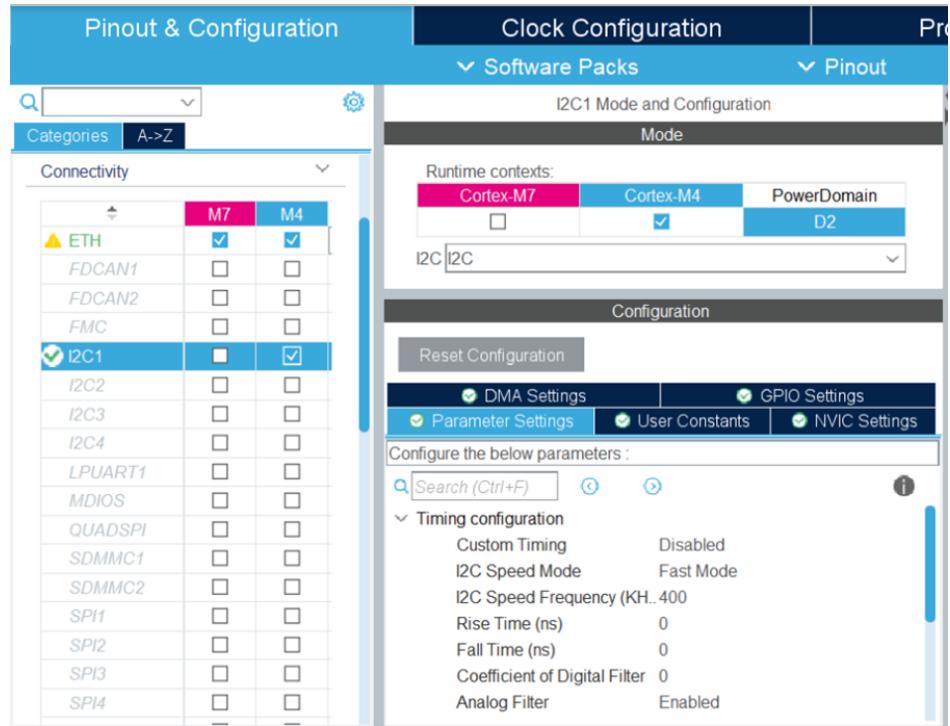
NVIC2 Interrupt Table

	Enabled	Preemption Priority	Sub Priority
TIM1 break interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt	<input checked="" type="checkbox"/>	0	0
TIM1 trigger and commutation interrupts	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input type="checkbox"/>	0	0

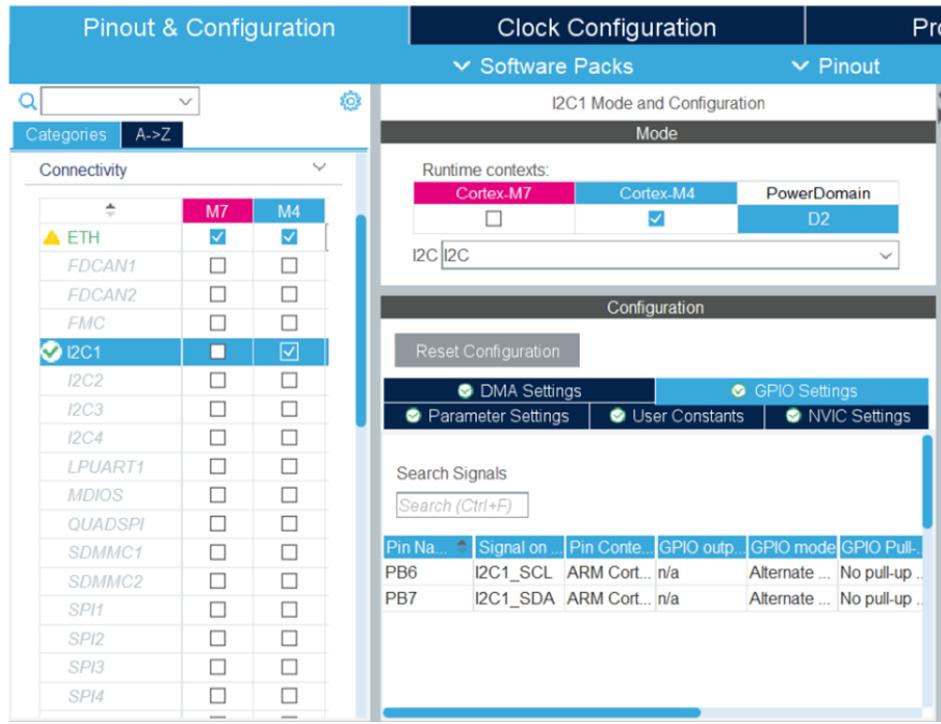
Nel file ‘main.c’ è presente la funzione di callback del timer, mostrata in sezione ‘Funzionamento del codice’.

3.2.2 IMU

Abbiamo abilitato la comunicazione I2C per consentire la connessione all'IMU . Questa comunicazione è stata resa possibile grazie a due porte GPIO che sono state connesse direttamente ai pin SDA e SCL dell'IMU.



In fast mode l'I2C lavora ad una frequenza di 400 KHz. Per acquisire i dati alla frequenza di 100Hz, la lettura avviene attraverso un pooling abilitato grazie all'interrupt generato dal timer 1. Nella seguente immagine riportiamo la configurazione dei pin utilizzati per I2C.



Nelle seguenti immagini sono rappresentate delle funzioni definite nel file IMU.c:

```

60 void bno055_setup() {
61     bno055_reset();
62
63     uint8_t id = 0;
64     bno055_readData(BNO055_CHIP_ID, &id, 1);
65     if (id != BNO055_ID) {
66         printf("Can't find BNO055, id: 0x%02x. Please check your wiring.\r\n", id);
67     }
68     bno055_setPage(0);
69     bno055_writeData(BNO055_SYS_TRIGGER, 0x0);
70
71     // Select BNO055 config mode
72     bno055_setOperationModeConfig();
73     bno055_delay(10);
74 }
```

```

19void bno055_assignI2C(I2C_HandleTypeDef *hi2c_device) {
20    _bno055_i2c_port = hi2c_device;
21 }

19void bno055_setOperationMode(bno055_opmode_t mode) {
20    bno055_writeData(BNO055_OPR_MODE, mode);
21    if (mode == BNO055_OPERATION_MODE_CONFIG) {
22        bno055_delay(19);
23    } else {
24        bno055_delay(7);
25    }
26 }

220bno055_vector_t bno055_getVectorEuler() {
221     return bno055_getVector(BNO055_VECTOR_EULER);
222 }

```

Attraverso l'impiego di queste funzioni vengono forniti i valori degli angoli roll, pitch e yaw ciclicamente. Successivamente troviamo l'implementazione della funzione bno055_getVector utilizzata all'interno della funzione bno055_getVectorEuler, presente nella precedente immagine.

```

172bno055_vector_t bno055_getVector(uint8_t vec) {
173    bno055_setPage(0);
174    uint8_t buffer[8]; // Quaternion need 8 bytes
175
176    if (vec == BNO055_VECTOR_QUATERNION)
177        bno055_readData(vec, buffer, 8);
178    else
179        bno055_readData(vec, buffer, 6);
180
181    double scale = 1;
182
183    if (vec == BNO055_VECTOR_MAGNETOMETER) {
184        scale = magScale;
185    } else if (vec == BNO055_VECTOR_ACCELEROMETER ||
186               vec == BNO055_VECTOR_LINEARACCEL || vec == BNO055_VECTOR_GRAVITY) {
187        scale = accelScale;
188    } else if (vec == BNO055_VECTOR_GYROSCOPE) {
189        scale = angularRateScale;
190    } else if (vec == BNO055_VECTOR_EULER) {
191        scale = eulerScale;
192    } else if (vec == BNO055_VECTOR_QUATERNION) {
193        scale = quaScale;
194    }
195
196    bno055_vector_t xyz = { .w = 0, .x = 0, .y = 0, .z = 0 };
197    if (vec == BNO055_VECTOR_QUATERNION) {
198        xyz.w = (int16_t)((buffer[1] << 8) | buffer[0]) / scale;
199        xyz.x = (int16_t)((buffer[3] << 8) | buffer[2]) / scale;
200        xyz.y = (int16_t)((buffer[5] << 8) | buffer[4]) / scale;
201        xyz.z = (int16_t)((buffer[7] << 8) | buffer[6]) / scale;
202    } else {
203        xyz.x = (int16_t)((buffer[1] << 8) | buffer[0]) / scale;
204        xyz.y = (int16_t)((buffer[3] << 8) | buffer[2]) / scale;
205        xyz.z = (int16_t)((buffer[5] << 8) | buffer[4]) / scale;
206    }
207
208    return xyz;
209 }

```

3.2.3 PWM

La modulazione a larghezza d'impulso è una tecnica di controllo digitale nata per sostituire la regolazione di tensione e di corrente attraverso l'uso di reostati o potenziometri. Un sistema PWM è in grado di modificare il duty cycle di un dispositivo elettrico, ossia il rapporto tra il tempo in cui il segnale resta "alto" rispetto al tempo totale di funzionamento e di mantenerlo così come definito dall'utente. Nel nostro caso i segnali PWM vengono generati mediante un timer, il quale è in grado di generare segnali a impulsi con diversi duty cycle. Il ciclo di lavoro è così calcolabile:

$$\delta = \frac{T_{on}}{T} \cdot 100\%$$

Dove abbiamo che:

T_{on} := intervallo in cui il segnale è attivo

T := periodo effettivo dell' impulso

Se il timer viene impostato in modalità PWM, insieme all'evento viene generata un'uscita che attiva il canale associato al registro CCRx , permettendo di controllare la frequenza di commutazione di ciascun canale associato a tale registro. La relazione tra il registro e il duty cycle è così definita:

$$CCRx = MaxCNT(1 - \delta)$$

con

$MaxCNT(1 - \delta)$:= registro di confronto dell' uscita (registro di ciclo)

Inoltre poiché i valori CCRx memorizzati dipendono dalla frequenza di clock reale del timer (TIMx_CLK), la frequenza del canale può essere definita nel seguente modo:

$$CHx_{Update} = \frac{TIMx_CLK}{MaxCNT}$$

Sono allegate di seguito le immagini relative al settaggio del PWM all'interno del file.ioc chiamato PID_project.ioc:

Pinout & Configuration

Clock Configuration

Software Packs

Pinout

TIM3 Mode and Configuration

Mode

	Cortex-M7	Cortex-M4	PowerDomain
Slave Mode	<input type="checkbox"/> Disable	<input checked="" type="checkbox"/>	D2
Trigger Source	<input type="checkbox"/> Disable	<input type="checkbox"/>	
Clock Source	<input type="checkbox"/> Internal Clock	<input type="checkbox"/>	
Channel1	<input type="checkbox"/> PWM Generation CH1	<input type="checkbox"/>	
Channel2	<input type="checkbox"/> PWM Generation CH2	<input type="checkbox"/>	
Channel3	<input type="checkbox"/> PWM Generation CH3	<input type="checkbox"/>	
Channel4	<input type="checkbox"/> PWM Generation CH4	<input type="checkbox"/>	
Combined Channels	<input type="checkbox"/> Disable	<input type="checkbox"/>	
<input type="checkbox"/> ETR IO as Clearing Source	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/> XOR activation	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/> One Pulse Mode	<input type="checkbox"/>	<input type="checkbox"/>	

Configuration

Reset Configuration

Pinout & Configuration

Clock Configuration

Software Packs

Pinout

TIM3 Mode and Configuration

Mode

	Cortex-M7	Cortex-M4	PowerDomain
Slave Mode	<input type="checkbox"/> Disable	<input checked="" type="checkbox"/>	D2
Trigger Source	<input type="checkbox"/> Disable	<input type="checkbox"/>	
Clock Source	<input type="checkbox"/> Internal Clock	<input type="checkbox"/>	
Channel1	<input type="checkbox"/> PWM Generation CH1	<input type="checkbox"/>	
Channel2	<input type="checkbox"/> PWM Generation CH2	<input type="checkbox"/>	
Channel3	<input type="checkbox"/> PWM Generation CH3	<input type="checkbox"/>	
Channel4	<input type="checkbox"/> PWM Generation CH4	<input type="checkbox"/>	
Combined Channels	<input type="checkbox"/> Disable	<input type="checkbox"/>	
<input type="checkbox"/> ETR IO as Clearing Source	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/> XOR activation	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/> One Pulse Mode	<input type="checkbox"/>	<input type="checkbox"/>	

Configuration

Reset Configuration

DMA Settings **GPIO Settings**

Parameter Settings **User Constants** **NVIC Settings**

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

- Prescaler (PSC - 16 bits v... 240-1)
- Counter Mode Up
- Counter Period (AutoRelo... 20000-1)
- Internal Clock Division (CK. No Division)
- auto-reload preload Disable

Trigger Output (TRGO) Param...

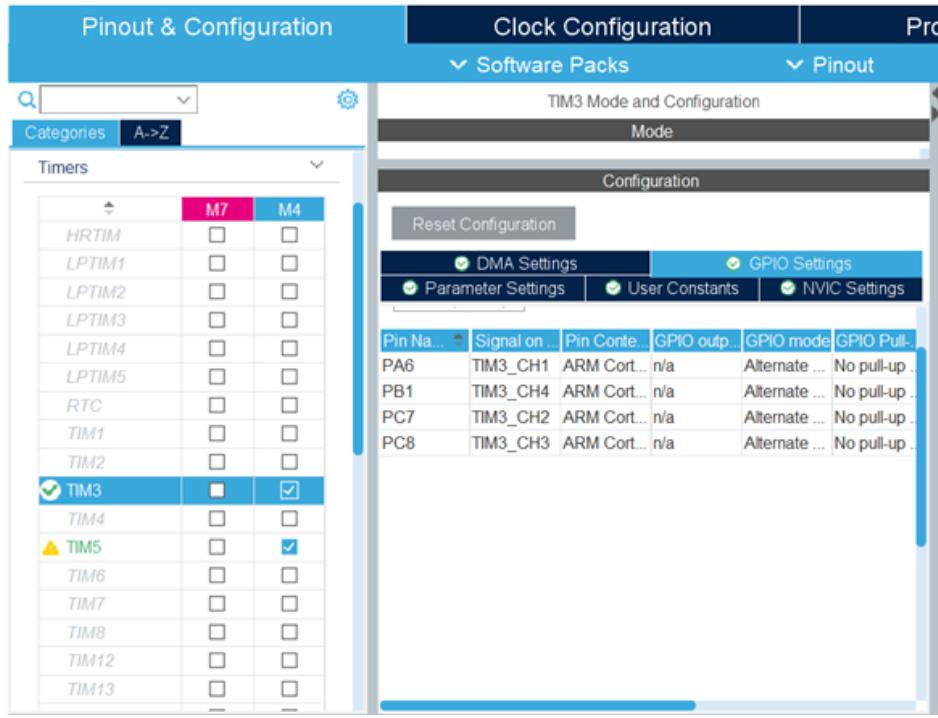
- Master/Slave Mode (MSM .. Disable (Trigger input effect not del...))
- Trigger Event Selection TR. Reset (UG bit from TIMx_EGR)

Clear Input

- Clear Input Source Disable

PWM Generation Channel 1

- Mode PWM mode 1



In questo caso il timer interessato (TIM3) è stato settato con clock di riferimento internal clock mentre i quattro canali sono stati impostati come PWM Generation. Prescaler e counter period sono combinati nella seguente formula:

$$UpdateEvent = \frac{TimerClock}{(Prescaler + 1)(Period + 1)}$$

$$UpdateEvent = \frac{240000000}{(240 + 1)(20000 + 1)} = 50Hz = 20ms$$

Vediamo nella sezione GPIO Settings i quattro pin su cui vengono mandati i segnali PWM in uscita ai quattro ESC connessi ciascuno ad un motore. Questa parte è fondamentale per la realizzazione del codice poiché tramite i quattro pin appena citati riusciamo a differenziare il segnale PWM per ogni motore in modo da garantire il bilanciamento. Analizziamo ora il funzionamento del codice che gestisce i vari PWM allegando di seguito le immagini relative al file map.c:

```

13 float map(float val)
14 {
15
16
17     float duty = ((MaxDuty-MinDuty)*val)/(MaxSpeed-MinSpeed) + ((MinDuty* MaxSpeed)-(MaxDuty*MinSpeed))/(MaxSpeed - MinSpeed);
18
19     if(duty< MinDuty){
20         return MinDuty;
21     }
22
23     if(duty> MaxDuty){
24         return MaxDuty;
25     }
26     //questi if non fanno spegnere il motore in caso di uscita dal range di pwm
27
28     return duty;
29 }
```

La funzione map(float val), prende i valori delle velocità per mapparli in duty tramite la formula:

$$duty = \frac{(MaxDuty - MinDuty) \cdot val}{MaxSpeed - MinSpeed} + \frac{(MinDuty \cdot MaxSpeed) - (MaxDuty \cdot MinSpeed)}{MaxSpeed - MinSpeed}$$

che ci siamo calcolati ipotizzando che sia lineare il rapporto fra duty e velocità. Questo valore verrà poi utilizzato per convertire i duty in PWM tramite la funzione setPwm che analizziamo di seguito.

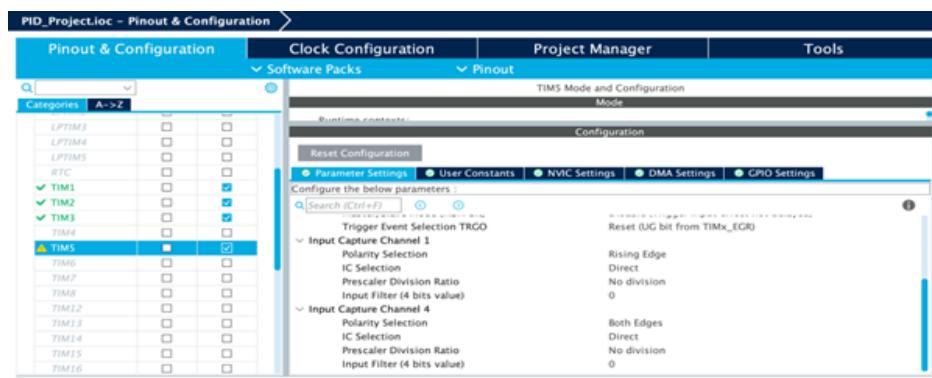
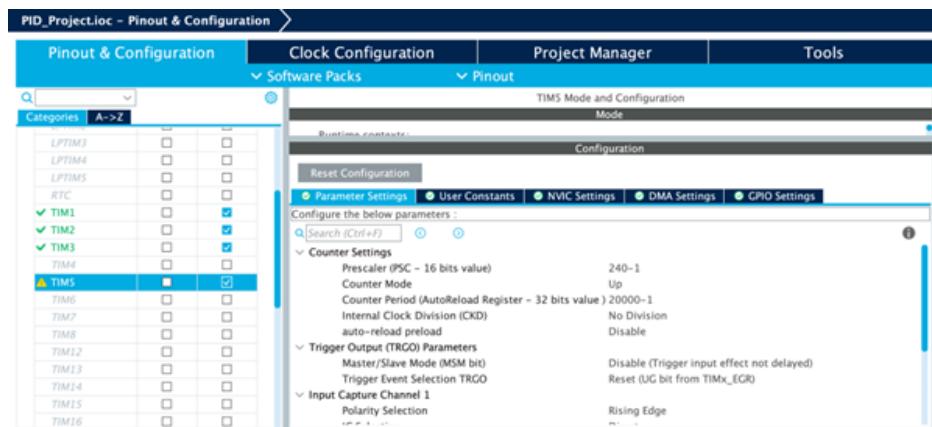
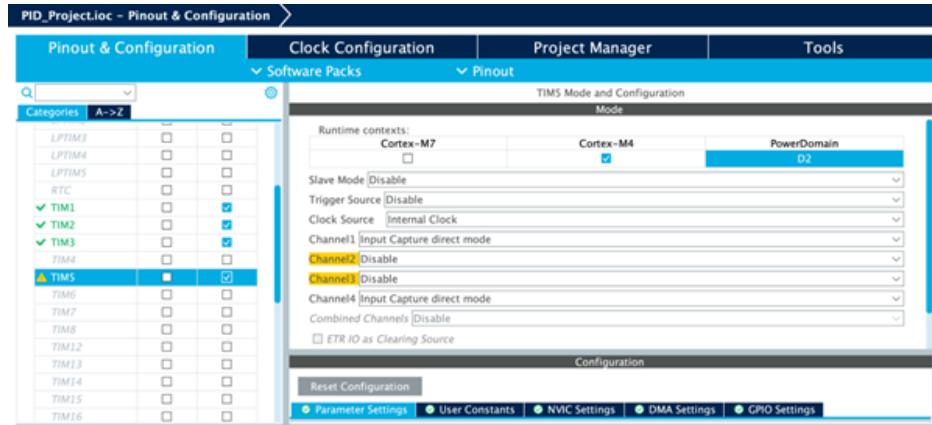
```

31 void setPwm(float pwm1, float pwm2, float pwm3, float pwm4) {
32
33     pwm1=pwm1>MaxDuty?MaxDuty:pwm1<MinDuty?MinDuty:pwm1;
34     pwm2=pwm2>MaxDuty?MaxDuty:pwm2<MinDuty?MinDuty:pwm2;
35     pwm3=pwm3>MaxDuty?MaxDuty:pwm3<MinDuty?MinDuty:pwm3;
36     pwm4=pwm4>MaxDuty?MaxDuty:pwm4<MinDuty?MinDuty:pwm4;
37
38     TIM3->CCR1 = (uint32_t) (TIM3->ARR* pwm1/100); //avgMotor1;
39     TIM3->CCR2 = (uint32_t) (TIM3->ARR* pwm2/100); //avgMotor2;
40     TIM3->CCR3 = (uint32_t) (TIM3->ARR* pwm3/100); // avgMotor3;
41     TIM3->CCR4 = (uint32_t) (TIM3->ARR* pwm4/100); //avgMotor4;
42 }
```

Tale funzione setPwm(float pwm1, float pwm2, float pwm3, float pwm4) ci permette di assegnare i valori di duty agli appositi registri rimanendo in un range di sicurezza. Entrambe le funzioni vengono utilizzate nel ‘main.c’, come mostrato in sezione ‘Funzionamento complessivo’.

3.2.4 RADIOCOMANDO

Per il radiocomando ci siamo serviti di due timer, il TIM5 e il TIM2, configurati entrambi con gli stessi parametri che vengono elencati di seguito.



The screenshot shows the 'Pinout & Configuration' tab selected. In the left sidebar, 'TIM5' is selected under the 'TIM' category. The main area displays a table of pins for PA0 and PA3, with the 'Modified' column checked for both rows.

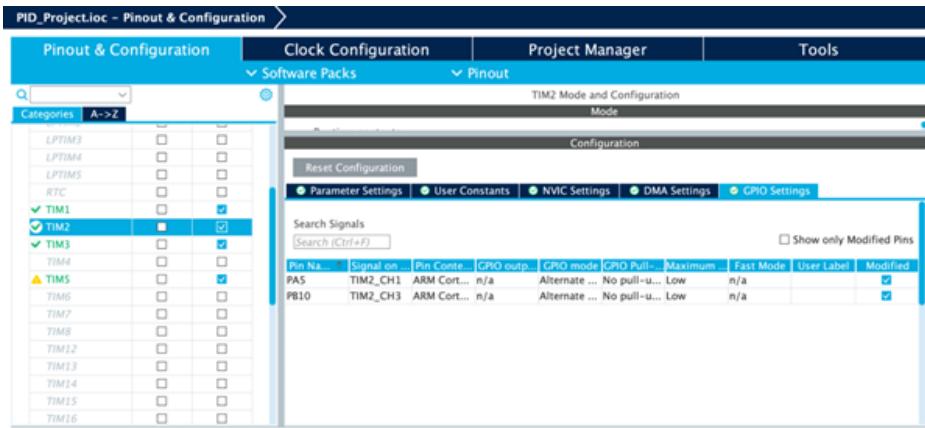
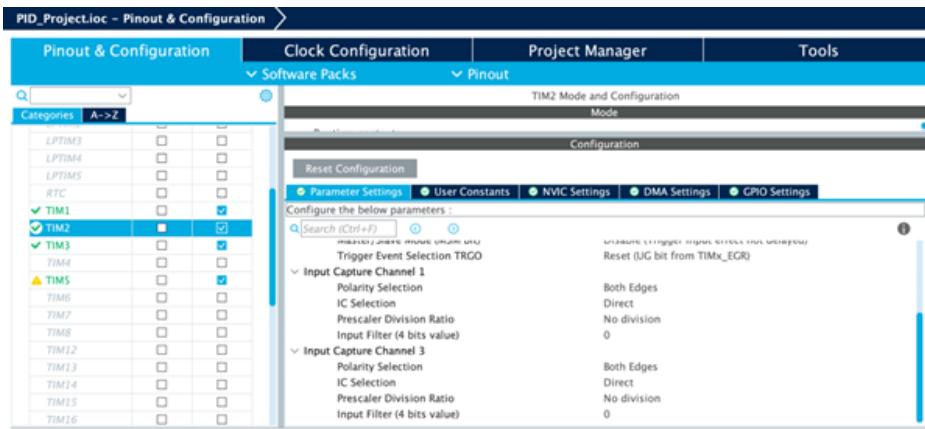
Pin No.	Signal on	Pin Context	GPIO output	GPIO mode	GPIO Pull-up	Maximum ...	Fast Mode	User Label	Modified
PA0	TIM5_CH1	ARM Cortex...	n/a	Alternate ...	No pull-u...	Low	n/a		<input checked="" type="checkbox"/>
PA3	TIM5_CH4	ARM Cortex...	n/a	Alternate ...	No pull-u...	Low	n/a		<input checked="" type="checkbox"/>

Il TIM5_CH4 è collegato al CHANNEL 5 del ricevitore, che ci permette di armare, attivare e spegnere i motori attraverso l'interruttore.

Di seguito vengono mostrati i settaggi del TIMER2.

The screenshot shows the 'Clock Configuration' tab selected. In the left sidebar, 'TIM2' is selected under the 'TIM' category. The main area displays a configuration panel for TIM2, with the 'Modified' column checked for the 'PowerDomain' section.

The screenshot shows the 'Pinout & Configuration' tab selected. In the left sidebar, 'TIM2' is selected under the 'TIM' category. The main area displays a configuration panel for TIM2, with the 'Modified' column checked for the 'Prescaler (PSC ~ 16 bits value)' section.



Il TIM2.CH1 è collegato al CHANNEL 1 del ricevitore, che ci permette di poter effettuare un movimento di roll, e quindi di poter cambiare il riferimento del PID del roll. Il TIM2.CH3 è collegato al CHANNEL 2 del ricevitore, che ci permette di poter effettuare un movimento di pitch, e quindi di poter cambiare il riferimento del PID del pitch. E' possibile effettuare questi due movimenti utilizzando la levetta di destra del radiocomando e, se e solo se, l'interruttore D/R Switch (situato nello spigolo in alto a sinistra) è posizionato verso l'alto. Successivamente vengono mostrati degli snapshot inerenti al codice nel file 'main.c' per il funzionamento del radiocomando.


```

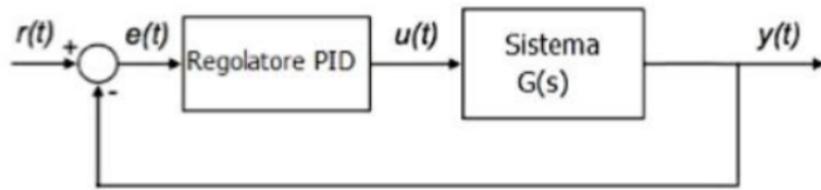
880
881     if(htim == &htim2){
882
883         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1){
884             if (chDuty[IC_CHANNEL1].firstCaptured == 0) // if the first value is not captured
885             {
886                 chDuty[IC_CHANNEL1].firstCaptured = 1; // set the first captured as true
887                 __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
888             }
889         }
890         else{
891             chDuty[IC_CHANNEL1].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
892             chDuty[IC_CHANNEL1].usWidth = (float) (chDuty[IC_CHANNEL1].val) / 1000000; //refClock;
893             chDuty[IC_CHANNEL1].duty = chDuty[IC_CHANNEL1].usWidth * chFrequency.frequency * 100;
894             chDuty[IC_CHANNEL1].duty = floorf(chDuty[IC_CHANNEL1].duty * 100) / 100;
895
896             calcolo_roll=chDuty[IC_CHANNEL1].duty;
897
898             if(calcolo_roll < 15 && calcolo_roll>5){
899                 if(calcolo_roll < 9.8 && calcolo_roll > 8.5)
900                     rif_roll=-1;
901                 if(calcolo_roll <= 8.5 && calcolo_roll > 7.5)
902                     rif_roll = -2;
903                 if(calcolo_roll <= 7.5 && calcolo_roll > 6.3)
904                     rif_roll = -3;
905                 if(calcolo_roll <= 10.3 && calcolo_roll >= 9.8)
906                     rif_roll = 0;
907                 if(calcolo_roll > 10.3 && calcolo_roll <= 11.3)
908                     rif_roll = 1;
909                 if(calcolo_roll > 11.3 && calcolo_roll <= 12.3)
910                     rif_roll = 2;
911                 if(calcolo_roll > 12.3 && calcolo_roll <= 13.5)
912                     rif_roll = 3;
913             }
914
915
916             //provaroll = chDuty[IC_CHANNEL1].duty;
917
918             chDuty[IC_CHANNEL1].firstCaptured = 0;
919         }
920     }
921
922 }
923
924
925
926     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3){
927         if (chDuty[IC_CHANNEL3].firstCaptured == 0) // if the first value is not captured
928         {
929             chDuty[IC_CHANNEL3].firstCaptured = 1; // set the first captured as true
930             __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
931         }
932         else{
933             chDuty[IC_CHANNEL3].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3);
934             chDuty[IC_CHANNEL3].usWidth = (float) (chDuty[IC_CHANNEL3].val) / 1000000; //refClock;
935             chDuty[IC_CHANNEL3].duty = chDuty[IC_CHANNEL3].usWidth * chFrequency.frequency * 100;
936             chDuty[IC_CHANNEL3].duty = floorf(chDuty[IC_CHANNEL3].duty * 100) / 100;
937
938
939             calcolo_pitch=chDuty[IC_CHANNEL3].duty;
940
941             if(calcolo_pitch < 15 && calcolo_pitch>5){
942                 if(calcolo_pitch < 9.8 && calcolo_pitch > 8.5)
943                     rif_pitch=-1;
944                 if(calcolo_pitch <= 8.5 && calcolo_pitch > 7.5)
945                     rif_pitch = -2;
946                 if(calcolo_pitch <= 7.5 && calcolo_pitch > 6.3)
947                     rif_pitch = -3;
948                 if(calcolo_pitch <= 10.3 && calcolo_pitch >= 9.8)
949                     rif_pitch = 0;
950                 if(calcolo_pitch > 10.3 && calcolo_pitch <= 11.3)
951                     rif_pitch = 1;
952                 if(calcolo_pitch > 11.3 && calcolo_pitch <= 12.3)
953                     rif_pitch = 2;
954                 if(calcolo_pitch > 12.3 && calcolo_pitch <= 13.5)
955                     rif_pitch = 3;
956             }
957
958
959             chDuty[IC_CHANNEL3].firstCaptured = 0;
960         }
961     }
962
963 }
964
965
966 }
967

```

NOTA BENE: la mappatura che è stata programmata è valida solo se il segnale emesso dall'analogico di destra del radiocomando è compreso tra 10 e 10.2. Quindi i valori da controllare sono quelli contenuti nelle variabili calcolo_pitch e calcolo_roll. E' possibile centrare quelle variabili nel range descritto prima utilizzando i due tasti di precisione (Elevator Trim, Aileron fine-tuning), anch'essi situati nella parte destra del radiocomando; uno permetterà di modificare calcolo_pitch mentre l'altro calcolo_roll.

3.2.5 PID

I controllori PID sono algoritmi di regolazione da impiegare in sistemi di controllo ad anello chiuso (feedback), cioè a reazione negativa o in controreazione, dove l'ingresso di controllo è dato dalla somma di tre componenti: una proporzionale, una integrale ed una derivativa. Dalle iniziali Proportional-Integral-Derivative si ottiene appunto l'acronimo PID che definisce il tipo di regolatore. L'idea di base di qualsiasi sistema di controllo del feedback è quella di calcolare continuamente il valore dell'errore come differenza tra il setpoint desiderato e la variabile di processo corrente da controllare. Sulla base dell'errore individuato, si calcola quindi il valore della variabile di controllo che spinge il sistema ad avere il comportamento desiderato. Un regolare PID può essere rappresentato nel seguente modo:



Dove
 $r(t)$:= segnale di riferimento in ingresso
 $e(t)$:= errore ($r(t) - y(t)$)
 $y(t)$:= segnale in uscita
 $u(t)$:= ingresso di controllo

Nello schema possiamo notare come il controllore venga posto in serie al sistema da controllare (ovviamente prima di esso perché il controllo abbia effetto sulla grandezza in ingresso al sistema) e come il circuito di retroazione riporti in ingresso il termine $y(t)$, il quale, attraverso un nodo sommatore (che in questo caso assume il ruolo di sottrattore), va a modificare il segnale vero e proprio d'ingresso del controllore (struttura standard). Inoltre completiamo lo schema con la legge di controllo, cioè il legame tra $e(t)$ e $u(t)$, del nostro regolatore

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_{t_0}^t e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

dove i parametri K_P , K_D e K_I rappresentano, rispettivamente, i guadagni delle azioni proporzionale, derivativa e integrale. In realtà è più comunemente usata la seguente formula

$$u(t) = K_P \cdot \left(e(t) + \frac{1}{T_I} \cdot \int_0^t e(\tau) d\tau + T_D \cdot \frac{de(t)}{dt} \right)$$

in cui a fianco al parametro K_P si evidenziano le costanti di tempo delle azioni derivativa e integrale, rispettivamente T_D e T_I . I parametri che identificano il PID dunque sono K_P , T_I (o K_I) e T_D (o K_D) ed essi sono chiamati anche gradi di libertà del controllore.

Se i termini proporzionale, integrale e derivativo vengono scelti in maniera errata, l'ingresso del processo controllato è instabile: in questo caso si otterebbe un'uscita divergente, con oscillazioni. Eseguendo dei calcoli seguendo la struttura del nostro codice siamo risaliti

all'ordine di grandezza dei coefficienti; successivamente andando per tentativi abbiamo visto che il coefficiente integrativo ki non era necessario per la stabilità del drone e siamo infine giunti ai seguenti valori: $Kp_roll = 0,04$, $Kd_roll = 0,04$, $Kp_pitch = 0,05$ e $Kd_pitch = 0,04$. Infatti, è possibile combinare solo le azioni proporzionale e derivativa in quanto solo con la prima il sistema presenta dei problemi di oscillazioni. Nel paragrafo riguardante i test sono presenti descrizioni e immagini dei tentativi eseguiti.

Successivamente troviamo le immagini relative alle funzioni per implementare il controllore PID situate nel file ‘PID.c’. Nelle seguenti righe di codice assegnamo i valori ai coefficienti della struct del PID:

```
13=void PID_Init( PID* conf, float kp, float kd, float ki, float dt, float outMin, float outMax)
14 {
15     conf->kp = kp;
16     conf->kd = kd;
17     conf->ki = ki;
18     conf->dt = dt;
19     conf->Iterm = 0;
20     conf->lastError = 0;
21     conf->outMax = outMax,
22     conf->outMin = outMin;
23 }
```

Ora riportiamo la funzione che calcola il valore che corrisponde al PID conoscendo l'errore in tempo reale.

```

25@ float PID_Compute(float input, float setPoint, PID* conf)
26 {
27     float error = setPoint - input;
28     //calcolo della variabile di errore
29
30     conf->Iterm += (error * conf->dt) * conf->ki;
31@ /*calcola l'integrale dell'errore (aggiungendo il nuovo errore). Noto il valore
32     * precedente dell'integrale dell'errore aggiungiamo l'errore nel nuovo passaggio
33     * quindi otteniamo l'area sotto la funzione di errore passo dopo passo*/
34
35     if((conf->Iterm) > conf->outMax) conf->Iterm = conf->outMax;
36     else if((conf->Iterm) < conf->outMin) conf->Iterm = conf->outMin;
37
38     float error2 = error;
39     float lastError = conf->lastError;
40
41     if (error2 <= 0 && lastError <= 0) {
42         error2 = -error2;
43         lastError = -lastError;
44     }
45
46     float dInput = (error2 - lastError) / conf->dt;
47     //derivata del valore di uscita
48
49     if (dInput > 0) {
50         if (error < 0) dInput=-dInput;
51     }
52     else {
53         if (error < 0) dInput=-dInput;
54     }
55
56     float output = (conf->kp * error) + (conf->Iterm) + (conf->kd * dInput);
57     //calcola l'output del PID comando tutti e tre gli output
58     printf("%.4f, %.4f, %.4f\n", input, conf->kp*error, conf->kd * dInput);
59
60     if(output > conf->outMax) output = conf->outMax;
61     else if(output < conf->outMin) output = conf->outMin;
62
63     conf->lastError = error;
64     //ricorda la variabile per il prossimo ciclo
65
66     return output;
67 }
--
```

Nella successiva funzione ad ogni elemento di Speeds_quad[] viene assegnato un valore che è ottenuto dalle somme riguardanti la matrice di controllo (che troviamo nella sezione Modello Matematico – Modello di Controllo). Successivamente viene eseguita la radice quadrata dei valori appena ricavati, in modo che tali risultati possano essere assegnati agli elementi del vettore Speeds[].

```

52@ float* SpeedCompute(float virtualInputs[])
53 {
54     static float Speeds_quad[4];
55     static float Speeds[4];
56
57     Speeds_quad[0] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
58     Speeds_quad[1] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
59     Speeds_quad[2] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
60     Speeds_quad[3] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
61/*
62     * Calcoliamo le velocità dei motori al quadrato, poichè non possono essere negative. Partendo dal
63     * valore di throttle e seguendo le matrici di controllo dei droni andiamo a sommare e sottrarre le
64     * variabili date tramite il pid per il controllo delle velocità.
65 */
66
67     Speeds[0]= sqrt(Speeds_quad[0]);
68     Speeds[1]= sqrt(Speeds_quad[1]);
69     Speeds[2]= sqrt(Speeds_quad[2]);
70     Speeds[3]= sqrt(Speeds_quad[3]);
71     //Una volta calcolata la velocità dei motori al quadrato, viene eseguita la radice
72
73     return Speeds;
74 }
```

3.3 Funzionamento complessivo

Il codice richiede una prima parte dove vengono inizializzate tutte le strutture necessarie. Prima di tutto, vengono inizializzate i GPIO, per utilizzare un led sulla scheda, I2C, per la comunicazione con l'IMU, e tutti i timer utilizzati.

```
184     /* Initialize all configured peripherals */
185     MX_GPIO_Init();
186     MX_I2C1_Init();
187     MX_TIM1_Init();
188     MX_TIM2_Init();
189     MX_TIM3_Init();
190     MX_TIM5_Init();
```

Successivamente, sotto “USER CODE BEGIN 2”, sono inizializzati la seriale e l'IMU.

```
191     /* USER CODE BEGIN 2 */
192     MX_USART3_UART_Init();
193     bno055_assignI2C(&hi2c1);
194     bno055_setup();
195     bno055_setOperationModeNDOF();
...
```

Vengono poi effettuate le inizializzazioni utili per l'utilizzo del radiocomando.

```
197     Receiver_Init(&chFrequency, chDuty, NUMBER_CHANNELS);
198     waitingForGettingFrequency();
199     jumpHalfPeriod(chFrequency.frequency);
200     startInputCaptureInterruptDutyCycle();
201     startInputCaptureInterruptPitchRoll();
```

La sezione seguente, permette di inizializzare i canali del timer3 in pwm mode.

```
//inizializzazione PWM
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2); } TIM CHANNEL_ALL
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3); }
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4); }
```

Inoltre viene chiamata la funzione PID_Init e vengono quindi assegnati quelli che devono essere i valori del PID. L'angolo yaw non viene impiegato perché nel nostro progetto abbiamo implementato la stabilità esclusivamente per roll e pitch.

```
211     PID_Init(&Pitch_PID, kpp, kdp, Ki, dtempo, -1.3, 1.3);
212     PID_Init(&Roll_PID, Kp, Kd, Ki, dtempo, -1.3, 1.3);
213     //PID_Init(&Yaw_PID, 1.5, 0, 0, 0.05, 0, 1);
214
```

Il codice all'interno del while funziona principalmente con l'ausilio di una variabile chiamata i che può assumere quattro valori diversi:

0. **MOTORI FERMI**
1. **ARMATURA DEI MOTORI**
2. **ESECUZIONE**
3. **ATTESA**

0) Inizialmente la variabile i è pari a 0, il sistema è fermo e ci comunica tramite la seriale che i motori sono disattivati.

```

228     while (1)
229     {
230
231     /* USER CODE END WHILE */
232
233     /* USER CODE BEGIN 3 */
234
235     if(i==0)
236         printf("Motore fermati!\r\n");
237

```

1) Quando viene attivato l'interruttore che permette l'armamento dei motori, la i viene portata al valore 1. Quando la i è uguale ad 1, vengono effettuate le seguenti operazioni: accensione di un led sulla scheda STM, armamento dei motori, comunicazione sulla seriale "Motori armati" e la sequenza dei tipi di dati che verranno inviati successivamente sulla seriale, infine settaggio della i al valore 3. In particolare, per armare i motori, vengono utilizzati i registri CCR del TIM3 con un duty pari al 4,75%. Il valore 950 è il valore del duty corrispondente al 4,75

$$TIMx -> CCRx = \frac{duty(\%) * PeriodTimer}{100} = \frac{4,75 * 20000}{100} = 950$$

```

238     //ARMAMENTO(4,75)
239     if(i == 1){
240
241         //HAL_Delay(5000);
242         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
243
244         TIM3->CCR1=950;
245         TIM3->CCR2=950;
246         TIM3->CCR3=950;
247         TIM3->CCR4=950;
248
249         printf("%s\n", stringa);
250         printf("PWM1, PWM2, PWM3, PWM4, ROLL, VI[1], PITCH, VI[2]\r\n");
251
252         HAL_Delay(5000);
253
254         i=3;
255     }
256

```

Da questo momento, entrano in gioco il TIM1, che programmato in interrupt, cambia il valore della i impostandola a 2. In questo modo viene garantita l'esecuzione del codice solo ad intervallo fisso (10ms).

```
979 //timer ciclo codice
980 void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {
981     if (htim == &htim1) {
982         if (i==3)
983             i=2;
984         /* else
985             if(i==2)
986                 printf("Ciclo codice non terminato!\r\n"); */
987     }
988 }
990
```

2) Il valore della i uguale a 2 permette di poter eseguire la parte più importante del codice all'interno del while, ovvero lettura degli angoli, calcolo dei due PID, elaborazione velocità, elaborazione duty e assegnazione dei vari duty ai 4 motori.

LETTURA DEGLI ANGOLI: Attraverso la funzione bno055_getVectorEuler() vengono acquisiti i valori degli angoli e corretti attraverso un if – else poiché l'IMU è posizionato al centro della scheda ma “sottosopra”. Inoltre abbiamo aggiunto dei valori al roll e al in pitch, perchè è stata rilevata una incongruenza tra i valori riportati dall'IMU e le nostre misure effettuate con la livella.

CALCOLO DEI PID: La funzione PID_Compute, dando come valori in input gli angoli, i valori di riferimento e il puntatore alla struct del PID, calcola i Virtual Input. Questi valori vengono assegnati agli elementi del vettore virtualInputs[].

ELABORAZIONE VELOCITA': La funzione SpeedCompute() permette di convertire i Virtual Inputs generati dai PID in velocità da assegnare ai motori.

ELABORAZIONE DUTY: La funzione map() permette di trasformare i valori delle velocità in duty da assegnare ai vari motori. Sono stati aggiunti e sottratti offset ai diversi AvgMotor che ci hanno permesso di stabilizzare i motori.

ASSEGNAZIONE DUTY: La funzione setPwm() permette di assegnare il duty, precedentemente calcolato, ai vari motori, tenendo conto dei valori massimi e minimi permessi del range di valori utilizzato. La funzione printf permette di comunicare attraverso la seriale le diverse informazioni misurate e calcolate.

```

260     if(i == 2) {
261
262         float virtualInputs[4];
263
264         bno055_vector_t v = bno055_getVectorEuler();
265         pitch=v.y;
266
267         if (v.z<0) {
268             roll= -v.z -180;
269         }
270         else {
271             roll= -v.z +180;
272         }
273         yaw = v.x;
274
275         // printf("Roll: %f\t Pitch: %f\t Head: %f\r\n", pitch, roll, v.x);
276
277         virtualInputs[0] = 15.6;//che equivale alla forza peso del drone;
278         //ipotizziamo che questo valore sia il 50% del thrust complessivo;
279         virtualInputs[1] = PID_Compute(roll,rif_roll, &Roll_PID);
280         virtualInputs[2] = PID_Compute(pitch,rif_pitch, &Pitch_PID);
281         virtualInputs[3] = 0; //PID_Compute(0,0, &Yaw_PID);
282
283         float* Speeds;
284
285         Speeds= SpeedCompute(virtualInputs);
286
287         //Cambiiamo il range di pwm che non è più 1240-2000 ma è 1240-1280
288         //Questo per ovviare alla velocità troppo elevata
289
290         //I valori ottenuti da SpeedCompute vengono mappati in velocità
291
292
293         float avgMotor1 = map*((Speeds+0)) + 0.019; //, MinSpeed, MaxSpeed, (MinDuty/100)*TIM3->ARR, (MaxDuty/100)*TIM3->ARR;
294         float avgMotor2 = map*((Speeds+1)) + 0.0295; //, MinSpeed, MaxSpeed, (MinDuty/100)*TIM3->ARR, (MaxDuty/100)*TIM3->ARR;
295         float avgMotor3 = map*((Speeds+2)) - 0.019; //, MinSpeed, MaxSpeed, (MinDuty/100)*TIM3->ARR, (MaxDuty/100)*TIM3->ARR;
296         float avgMotor4 = map*((Speeds+3)) - 0.0295; //, MinSpeed, MaxSpeed, (MinDuty/100)*TIM3->ARR, (MaxDuty/100)*TIM3->ARR;
297
298         printf("%2f, %2f, %2f, %2f, %f, %f, %f\r\n", avgMotor1, avgMotor2, avgMotor3, avgMotor4, roll, virtualInputs[0]);
299
300         //setPwm(6,2,6,2,6,2);
301         setPwm(avgMotor1, avgMotor2, avgMotor3, avgMotor4);
302
303
304         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_1, GPIO_PIN_SET);
305
306         i=3;
307
308     }
309
310 }
311 /* USER CODE END 3 */
312
...

```

- 3) Quando la i è uguale a 3, non viene eseguita nessuna parte di codice, fino a quando il TIM1 non genera un interrupt riportando il valore della i a 2.

4 Test e risultati

Durante la realizzazione del progetto abbiamo dovuto eseguire diversi test per il funzionamento dei motori tramite la programmazione degli ESC e altri test riguardanti i coefficienti dei PID e inserimento degli offset. Per quanto riguarda gli ESC, abbiamo utilizzato la scheda di programmazione per modificare la Start Mode, impostandola su Very Soft, perché in questa modalità il cambio di potenza dei motori è più leggero in modo da evitare oscillazioni improvvise.

Durante i test sono stati inviati in tempo reale tutti i parametri: pwm dei motori, angoli e ingressi virtuali tramite printf nel codice e textscan di matlab. Tramite un codice matlab che abbiamo elaborato siamo stati in grado di realizzare grafici bidimensionali e tridimensionali che ci hanno permesso di capire l'andamento del drone in base ai diversi coefficienti proporzionali e derivativi. Il codice matlab che abbiamo scritto per salvare i dati in tempo reale e per realizzare i grafici è riportato nella sezione "Appendice".

4.1 Scelta degli offset e dei coefficienti dei PID

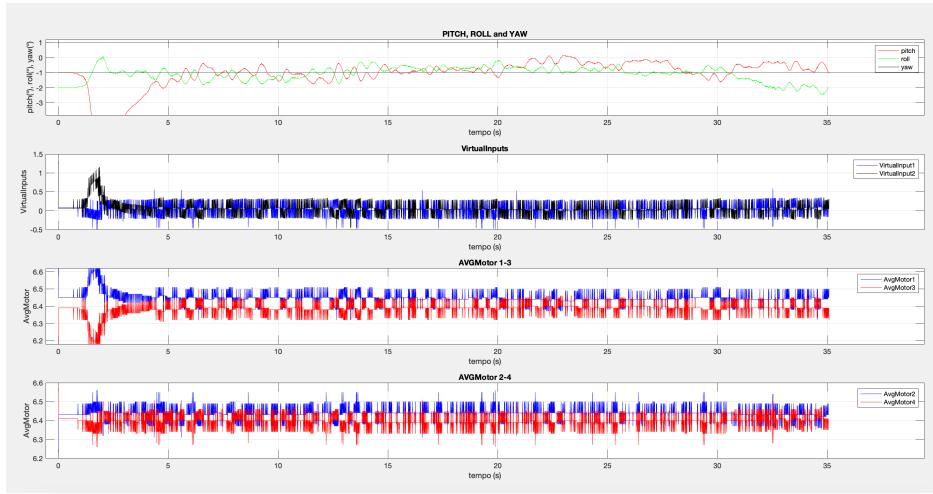
Per i coefficienti dei PID, partendo dall'ordine di grandezza dai valori stabiliti dal gruppo precedente, abbiamo eseguito diversi tentativi per stabilire quali valori numerici fossero migliori per la stabilizzazione del quadricottero. Per la scelta degli offset adeguati analizzando i risultati di varie simulazioni, tramite i grafici di matlab, abbiamo scelto dei specifici valori che ci hanno permesso di raggiungere la stabilità.

Di seguito riportiamo alcune delle simulazioni effettuate.

4.2 Simulazioni

Sono state eseguite diverse simulazioni, ognuna codificata con un numero progressivo. Analizziamo ora i grafici che abbiamo ottenuto delle misurazioni più significative:

4.2.1 Simulazione n.36



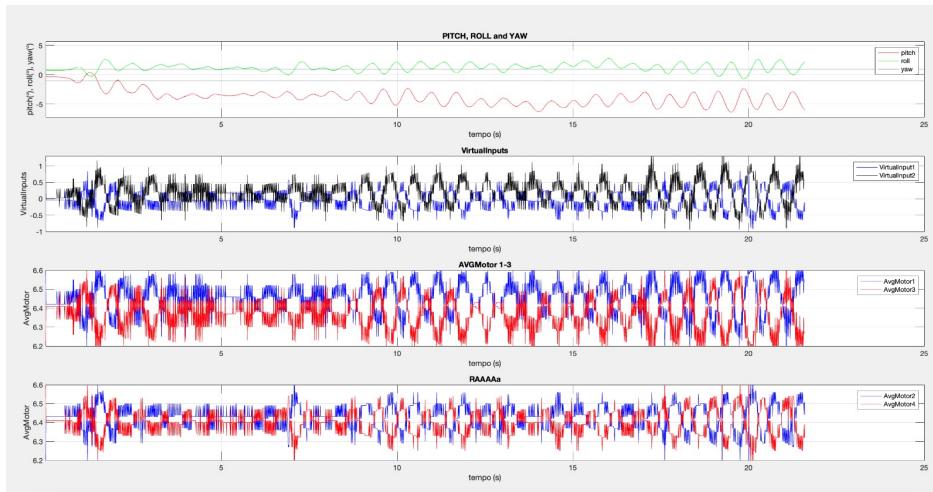
Valori: Kp roll = 0,04 Kd roll = 0,04 Kp pitch = 0,05 Kd pitch= 0,04

ESC: Very Soft

Offset: pwm m1 + 0,018, pwm m3 - 0,018, pwm m2 + 0,03, pwm m4 - 0,03

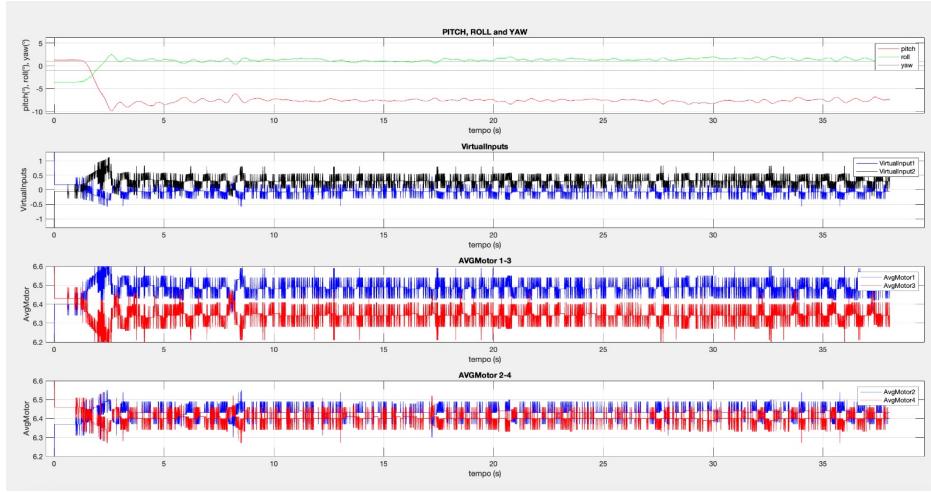
Abbiamo aggiunto un offset ai motori in modo da ottenere il miglior risultato possibile di stabilità, in particolare viene aumentato il pwm del motore 1 di 0,018 e del motore 2 di 0,03 mentre viene diminuito il pwm del motore 3 di -0,018 e del motore 4 di -0,03. In questo caso abbiamo ottenuto una stabilità sia del pitch che del roll intorno a -1° con delle oscillazioni tra 0° e -2°. Tale simulazione ci ha portato ad avere i migliori risultati ottenuti.

4.2.2 Simulazione n.12



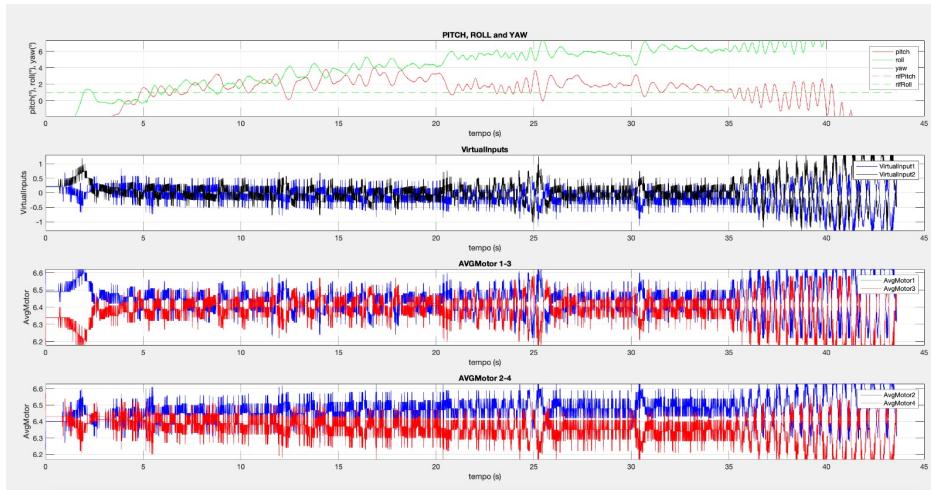
Valori: $K_p = 0,05$ $K_d = 0,045$ ESC: Soft Considerazioni: come possiamo notare dai grafici degli angoli, il pitch si stabilizza intorno ai -5° mentre il roll intorno ad 1° . In questo caso la start mode dell'ESC era in modalità Soft.

4.3 Simulazione n.26



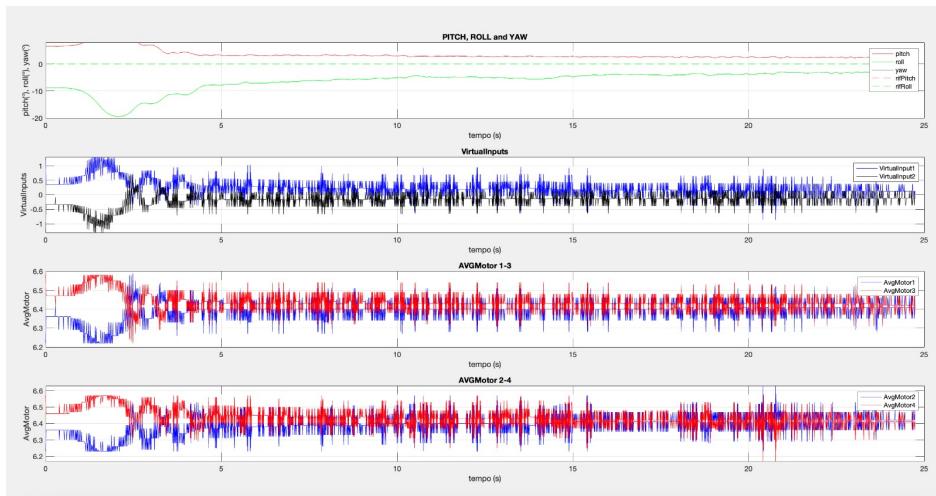
Valori: K_p roll = 0,04 K_d roll = 0,04 K_p pitch = 0,05 K_d pitch= 0,04 ESC: Very Soft Considerazioni: in questa simulazione abbiamo dato dei valori diversi alle costanti derivate e proporzionali dei PID del roll e del pitch e abbiamo impostato la start mode dell'ESC in modalità Very Soft. Possiamo notare che il roll rimane molto stabile intorno ad 1° mentre il pitch si stabilizza sui -8°

4.3.1 Simulazione n.44



Valori: Kp roll = 0,04 Kd roll = 0,04 Kp pitch = 0,05 Kd pitch= 0,04 ESC: Very Soft Offset: pwm m1 + 0,018, pwm m3 - 0,018, pwm m2 + 0,03, pwm m4 - 0,03 In questa simulazione abbiamo tentato di cambiare il riferimento del PID con l'utilizzo del radiocomando, impostando il riferimento ad 1°. Abbiamo ottenuto un pessimo risultato con molte oscillazioni, siamo giunti alla conclusione che il PID con i valori scelti da noi, riesce a trovare velocemente il punto di equilibrio quando il range di potenza dei motori è molto limitato, ciò però comporta l'instabilità nel momento in cui si vuole aumentare il livello di saturazione per far sì che alcuni motori spingano di più per variare l'angolazione e cambiare riferimento.

4.3.2 Simulazione n.55



Valori: Kp roll = 0,04 Kd roll = 0,04 Kp pitch = 0,05 Kd pitch= 0,04 ESC: Very Soft Offset: pwm m1 + 0,018, pwm m3 - 0,018, pwm m2 + 0,03, pwm m4 - 0,03 Riprovando ad ottenere la stabilità con riferimento 0° abbiamo notato un comportamento anomalo del drone, il roll si stabilizza intorno ai -4° mentre il pitch sui 3°, questo dovuto al fatto che l'IMU in seguito a forte vibrazioni non è più coerente con la reale posizione del drone, abbiamo allora tentato di correggere le misure dell'IMU tramite una livella, aggiungendo nel codice +2,8° al pitch e -3,435° al roll, ma non è bastato per ottenere un risultato ottimale.

Conclusioni e sviluppi futuri

Alla fine di questo progetto possiamo concludere che il drone riesce a mantenere la stabilità per quanto riguarda gli angoli di pitch e roll nell'intervallo di 0° e -1° , tuttavia può essere migliorato. Pensiamo che per migliorare la stabilità del drone bisogni risolvere problemi dovuti all'instabilità della base che di fatto non risulta ben salda e rigida enfatizzando le oscillazioni, fissare in maniera opportuna il supporto dell'IMU che in seguito ai diversi test, per via delle vibrazione, si è sicuramente spostato dalla posizione iniziale e in ogni simulazione potrebbe fornire valori non conformi, anche con degli offset, ed infine consigliamo di eseguire i test con la batteria non completamente carica poiché ci è sembrato che dia troppa potenza ai motori generando così delle oscillazioni. Queste ulteriori problematiche devono essere approfondite in futuri progetti, evidenziandone la presenza, determinandone l'origine, e una soluzione per migliorare le prestazioni.

I Matlab

Codice MATLAB per acquisire i dati:

```
%> %% Parametri acquisizione
contatore=20;
drone.data = datetime;
drone.batteriaIniziale = 15.2;
drone.PidPitch.Ki = 0;
drone.PidPitch.Kp = 0.045;
drone.PidPitch.Kd = 0.0425;
drone.PidRoll.Ki = 0;
drone.PidRoll.Kp = 0.045;
drone.PidRoll.Kd = 0.0425;
drone.MinDuty = 6.2;
drone.MaxDuty = 6.6;
drone.formula = 1;

port = "/dev/tty.usbmodem103";
Baund_Rate = 115200;
Data_Bits = 8;
Stop_Bits = 1;
if ~isempty ( instrfind )
    fclose ( instrfind );
    delete ( instrfind );
end
s= serialport(port, Baund_Rate, 'DataBits' , Data_Bits, 'StopBits', Stop_Bits);
while (isempty ( serialportlist )) % attende che la porta sia disponibile end fopen (s);
end
fopen(s);
```

```

%% Acquisizione

i=1;
while true
    testo=readline(s);
    if ~isempty(testo)
        A = cell2mat(textscan(testo,"%f, %f, %f, %f, %f, %f, %f, %f, %f\r\n"));

        if isempty(A)
            fprintf("%s\n", testo)
            if strcmp(testo,sprintf("Motori fermati!\r"))==1
                break;
            end
        else %isnumeric(A) && all(size(A)==0)
            avgMotor(i,:)= [A(1:4)];
            roll(i)= A(5);
            virtualInputs(i, :) = [A(6), A(8)];
            pitch(i)= A(7);
            yaw(i)= A(9);
            i=i+1;
        end
    end
end

%% Chiusura porta felose (s);
fclose(s);
delete (s);
clear s;

%% Salvataggio dati in un file

drone.batteriaFinale=input("Digita la carica finale della batteria in Volt: ");

nomeFile = sprintf("risultati_%d", contatore);
save(nomeFile,"avgMotor","roll", "pitch", "virtualInputs", "yaw", "drone");

```

Successivamente riportiamo il codice MATLAB per la rappresentazione dei grafici 2D di angoli, input virtuali e pwm dei motori. Ne esistono due versioni, una che stampa solo i valori precedentemente nominati, e un'altra versione che stampa anche i riferimenti del pitch e del roll. Da risultati40 (incluso) in poi, è possibile utilizzare il file “StampaDati.m”, per tutti i risultati precedenti, bisogna utilizzare il file “StampaDatiSenzaRiferimento.m”. Poiché i due file sono molto simili, per semplicità si riporta solamente il codice di “StampaDati”.

```

clear all;
clc;

contatore = 39;
nomeFile = sprintf("risultati_%d", contatore);
load(nomeFile);

msg = sprintf(['Dati della simulazione\n\n',...
    'Data = %s\n',...
    'Batteria iniziale = %.1f\n',...
    'PidPitch Ki = %.2f\n',...
    'PidPitch Kp = %.4f\n',...
    'PidPitch Kd = %.4f\n',...
    'PidRoll Ki = %.2f\n',...
    'PidRoll Kp = %.4f\n',...
    'PidRoll Kd = %.4f\n',...
    'MinDuty = %.2f\n',...
    'MaxDuty = %.2f\n',...
    'Batteria finale = %.1f\n',...
    'formula = %d'],...
drone.data, drone.batteriaIniziale, drone.PidPitch.Ki, drone.PidPitch.Kp, ...
drone.PidPitch.Kd, drone.PidRoll.Ki, drone.PidRoll.Kp, drone.PidRoll.Kd, drone.MinDuty,
drone.MaxDuty, drone.batteriaFinale, drone.formula);
msgbox(msg,'Dati Simulazione');

```

```

t=0:0.01:0.010*(length(pitch)-1);
fig = figure;
fig.Name = nomeFile;
tiledlayout(4, 1)
tile(1)=nexttile;
plot(t, pitch, "r", t, roll, "g", t, yaw, "k", t, rif_pitch, "r--", t, rif_roll, "g--")
grid on
title('PITCH, ROLL and YAW')
xlabel('tempo (s)')
ylabel('pitch(°), roll(°), yaw(°)')
%yline(2,'-k');
%yline(-1,'-k');
legend(["pitch", "roll", "yaw", "rifPitch", "rifRoll"])
ylim([-20 8])
%hold on;

```

```

tile(2)=nexttile;
plot(t, virtualInputs(:,1), "b", t, virtualInputs(:,2), "k");
grid on
title('VirtualInputs')
xlabel('tempo (s)')
ylabel('VirtualInputs')
legend(["VirtualInput1", "VirtualInput2"])

```

```

tile(3)=nexttile;
plot(t, avgMotor(:,1), "b", t, avgMotor(:,3), "r");
grid on
title('AVGMotor 1-3')
xlabel('tempo (s)')
ylabel('AvgMotor')
legend(["AvgMotor1", "AvgMotor3"])

tile(4)=nexttile;
plot(t, avgMotor(:,2), "b", t, avgMotor(:, 4), "r");
grid on
title('AVGMotor 2-4')
xlabel('tempo (s)')
ylabel('AvgMotor')
legend(["AvgMotor2", "AvgMotor4"])

linkaxes(tile,'x');

```

Riportiamo ora il codice MATLAB per la rappresentazione 3D della simulazione effettuata:

```
clearvars
close all
clc

Ax_ = [1 0 0];
Ay_ = [0 1 0];
Az_ = [0 0 1];

x_ = [0 0 0 ; 1 0 0];
y_ = [0 0 0 ; 0 1 0];
z_ = [0 0 0 ; 0 0 1];

syms a;
Rx(a) = [1 0 0 ;
          0 cosd(a) -sind(a);
          0 sind(a) cosd(a)];

Ry(a) = [cosd(a) 0 sind(a)
          0 1 0;
          -sind(a) 0 cosd(a)];

Rz(a) = [cosd(a) -sind(a) 0;
          sind(a) cosd(a) 0;
          0 0 1 ];

load("prova.mat","pitch","roll","yaw");

Ax=zeros(length(pitch),3);
Ay=zeros(length(pitch),3);
Az=zeros(length(pitch),3);

l=0;

fprintf("Calcolando il campione: ");
for i=1:length(pitch)
    messaggio = sprintf("%i/%i (%.2f%%);",i,length(pitch),i/length(pitch)*100);
    fprintf("%s%s",sprintf(repmat('\b',1,l)),messaggio);
    l = length(char(messaggio));
    Ax(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Ax_.').';
    Ay(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Ay_.').';
    Az(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Az_.').';
end
```

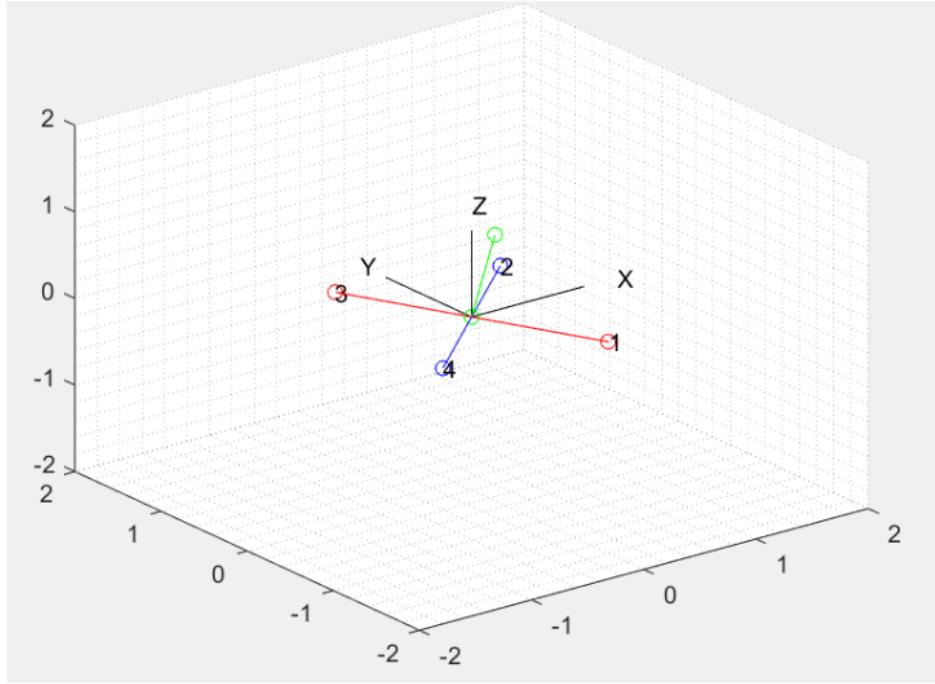
```

Ax=zeros(length(pitch),3);
Ay=zeros(length(pitch),3);
Az=zeros(length(pitch),3);

l=0;

fprintf("Calcolando il campione: ");
for i=1:length(pitch)
    messaggio = sprintf("%i/%i (%.2f%"); i,length(pitch),i/length(pitch)*100);
    fprintf("%s%",sprintf(repmat('\b',1,l)),messaggio);
    l = length(char(messaggio));
    Ax(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Ax_.').';
    Ay(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Ay_.').';
    Az(i,:) = (Rz(yaw(1,i))*Ry(pitch(1,i))*Rx(roll(1,i))*Az_.').';
end

```



Bibliografia