



Università Politecnica delle Marche
Corso di laurea
Ingegneria Informatica e dell'automazione
a.a. 2021 – 2022

Laboratorio di Automazione

Studenti : Prencipe Ilaria Rita
 Ronchini Nicola
 Timmer Valeria

Docente : Prof. Bonci Andrea

SOMMARIO

1. Introduzione	4
1.1 <i>Cenni storici</i>	4
1.2 <i>Fisica di un quadrirotore</i>	5
2. Modello matematico quadrirotore	7
2.1 <i>Sistemi di riferimento e grandezze fondamentali</i>	7
2.2 <i>Caratterizzazione dinamica</i>	8
2.3 <i>Modello di controllo</i>	9
2.4 <i>Conversione dell'uscita di controllo in velocità di rotazione</i>	9
2.5 <i>Coefficienti di spinta e di resistenza aerodinamica</i>	10
2.6 <i>Controllori PID</i>	11
2.6.1 <i>Controllo dell'assetto</i>	13
3. Hardware	15
3.1 <i>Scheda STM32H7 Nucleo-144 boards</i>	15
3.1.1 <i>Cos'è un microcontrollore</i>	15
3.1.2 <i>Caratteristiche</i>	15
3.2 <i>Motori</i>	17
3.3 <i>ESC</i>	18
3.3.1 <i>Cos'è e come funziona</i>	18
3.3.2 <i>Come programmare un ESC</i>	19
3.4 <i>Eliche</i>	20
3.5 <i>Batteria</i>	21
3.5.1 <i>Caratteristiche</i>	21
3.5.2 <i>Lipo Saver</i>	21
3.6 <i>OLED Display</i>	21
3.7 <i>Telaio</i>	22
3.8 <i>Scheda di Potenza</i>	23
3.8.1 <i>Prima versione</i>	23
3.8.2 <i>Seconda versione</i>	24
3.8.2.1 <i>Regolatore di tensione lineare a 5V (LM7805)</i>	24
3.8.2.2 <i>Regolatore di tensione lineare a 3.3V (UA78M33)</i>	25
3.8.2.3 <i>Interruttore a MOSFET comandato tramite OPTO-commutatore</i>	26
3.8.2.4 <i>Adattatore di livello per segnali PWM</i>	29
3.9 <i>Altimetro</i>	30
3.9.1 <i>Posizionamento</i>	30

<i>3.10 IMU</i>	31
<i>3.10.1 Giroscopio: cos'è e come funziona</i>	31
<i>3.10.2 Accelerometro: cos'è e come funziona</i>	32
<i>3.10.3 Magnetometro: cos'è e come funziona</i>	32
<i>3.10.4 Sensore MPU-6050</i>	33
<i>3.10.5 Sensore HMC5883L</i>	34
4. Fonti	34
5. Software	35
<i>5.1 Caratteristiche generali</i>	35
<i>5.1.1 Struttura e main del programma</i>	38
<i>5.1.1.1 .ioc</i>	39
<i>5.1.1.2 Implementazione</i>	40
<i>5.1.2 IMU</i>	41
<i>5.1.2.1 .ioc</i>	41
<i>5.1.2.2 Implementazione</i>	42
<i>5.1.3 Magnetometro</i>	44
<i>5.1.3.1 .ioc</i>	44
<i>5.1.3.2 Implementazione</i>	44
<i>5.1.4 Calcolo angoli Roll, Pitch, Yaw</i>	46
<i>5.1.5 OLED Display</i>	47
<i>5.1.5.1 .ioc</i>	47
<i>5.1.5.2 Implementazione</i>	48
<i>5.2 PID</i>	49
<i>5.2.1 Inizializzazione</i>	50
<i>5.2.2 Esecuzione Real – Time</i>	51
6. PWM	53
<i>6.1 Cos'è e come funziona</i>	53
<i>6.2 Utilizzo</i>	54
<i>6.2.1 .ioc</i>	55
<i>6.2.2 Implementazione</i>	56
7. Connettori	59
8. Attivazione del drone	61
<i>8.1 Prove iniziali di controllo</i>	61
<i>8.2 Prove finali</i>	65
9. Eventuali sviluppi futuri	68

1. Introduzione

1.1 Cenni storici

“Trovo, se questo strumento a vite sarà ben fatto, cioè fatto di tela lina, stopata i suoi pori con amido, e svoltata con prestezza, che detta vite si fa la femmina nell’aria e monterà in alto”
Leonardo Da Vinci, Manoscritto B, foglio 83 v., 1483 – 1486

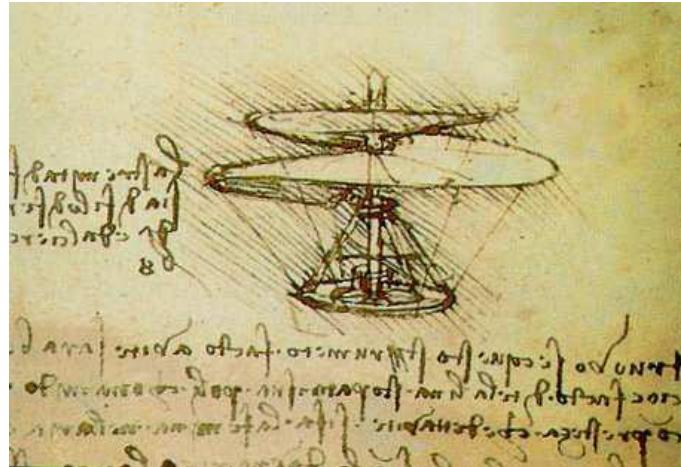


Fig. 1.1.1: Progetto di elicottero di Leonardo

La prima idea di macchina che potesse volare è da attribuire a Leonardo da Vinci, ma dovettero trascorrere secoli prima che la tecnologia potesse dar vita a velivoli dalle notevoli capacità, in grado di gestire autonomamente il volo.

Il drone, infatti è un velivolo caratterizzato dall'assenza di pilota a bordo. Il primo prototipo a radio controllo venne prodotto durante la Prima guerra mondiale, successivamente, durante la guerra fredda le dimensioni dei droni iniziarono a ridursi e, invece, l'uso di tecnologie avanzate, aumentò. Ad oggi è possibile stilare una classifica dei droni, secondo le loro dimensioni e i relativi dati operativi di utilizzo.

Il quadricottero (*dal greco ptéron: ala, “quattro ali”*) fa parte della famiglia dei velivoli multi-rotore (che vanno da tre ad otto rotori), ed è definito infatti come un aerogiro sollevato e spinto da quattro rotori. Esistono due generazioni di modelli quadrirotore: il primo fu progettato per trasportare passeggeri (primi velivoli “più pesanti dell’aria” a decollo e ad atterraggio verticale ad avere successo), il secondo per volare senza pilota a bordo (generazione più recente). I vantaggi dell’attuale generazione sono principalmente di due tipi:

- I velivoli sono caratterizzati da una conformazione semplice ed i costi e i tempi di manutenzione sono ridotti (ciò è dovuto dal fatto che i quadrirotori non necessitano di collegamenti meccanici per variare l’angolo d’attacco del rotore quando ruotano).
- Il danno che verrebbe provocato nel momento in cui i rotori dovessero colpire un oggetto è ridotto (ciò è dovuto dal fatto che il diametro dei singoli rotori è minore di quello di un elicottero equivalente, dunque l’energia cinetica

immagazzinata è anch'essa minore, ed inoltre i rotori possono essere racchiusi all'interno di una struttura di protezione).

Altri vantaggi, non di secondaria importanza riguardano:

- Simmetria del design che consente la centralizzazione dei sistemi di controllo e del payload (mantenere il baricentro nella stessa posizione risulta essere più semplice).
- Ogni rotore contribuisce al raggiungimento della portanza richiesta, fornendo una spinta maggiore ad un elicottero convenzionale (ciò consente di portare payload o piattaforme computazionali più pesanti).

1.2 Fisica di un quadrirotore

Fisicamente il sistema si presenta come un corpo rigido simmetrico, facile da controllare e con una forte stabilità, grazie alla disposizione a croce dei due assi sui quali sono disposti i quattro rotori, a pala fissa, opportunamente dimensionati. Tale simmetria permette di centralizzare il carico sollevabile (payload) al centro del telaio (frame).

È possibile definire sei gradi di libertà nella descrizione del sistema di un quadrirotore: *beccheggio*, in inglese *pitch*, (oscillazione del veicolo attorno all'asse trasversale), *imbardata*, in inglese *yaw*, (oscillazione del veicolo attorno all'asse verticale passante per il baricentro), *rollio*, in inglese *roll*, (oscillazione del veicolo attorno al proprio asse longitudinale), *x* (movimento nella direzione frontale del veicolo), *y* (movimento verso il lato sinistro del veicolo) e *z* (altitudine), ma è controllato utilizzando solo quattro attuatori.

Rotori opposti ruotano nello stesso verso: i motori 2 – 4 ruotano in senso normale (sinistrorso), mentre i motori 1 – 3 in senso inverso (destrorso). In particolare, quando ruotano tutti con la stessa velocità angolare, l'accelerazione angolare attorno all'asse di imbardata è nulla. L'imbardata viene provocata da una discrepanza nel bilanciamento del momento torcente aerodinamico, ovvero controbilanciando opportunamente i comandi di spinta tra le coppie di eliche che ruotano in senso opposto.

È possibile aumentare o diminuire la velocità dei motori, strettamente collegate alla portanza delle pale ed alla forza totale di spinta, andando a controllare il moto del velivolo lungo l'asse *z*.

I movimenti lungo gli assi orizzontali sono controllati agendo sulle coppie dei rotori simmetrici, mantenendo la rotazione dei restanti a velocità costante. In particolare, un aumento delle velocità dei motori 1 – 2 implica una variazione del beccheggio. In modo analogo, una modifica delle velocità dei motori 2 – 3 genera una variazione dell'angolo di rollio.

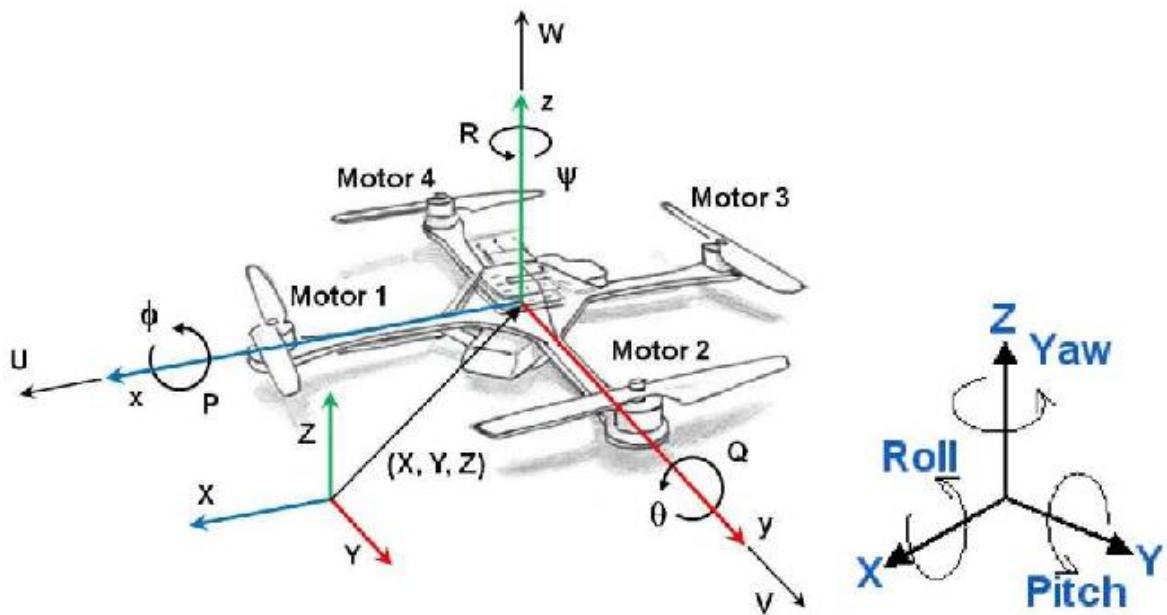


Fig. 1.2.1: Rappresentazione della dinamica del sistema quadrirotore

2. Modello matematico quadrirotore

2.1 Sistemi di riferimento e grandezze fondamentali

Prima di definire le equazioni che descrivono il velivolo, è utile introdurre le coordinate di riferimento, necessarie per la descrizione dell'assetto e della posizione.

Nel caso del quadrirotore è possibile definire due sistemi di riferimento: uno fisso, ed uno mobile, solidale con il telaio del drone.

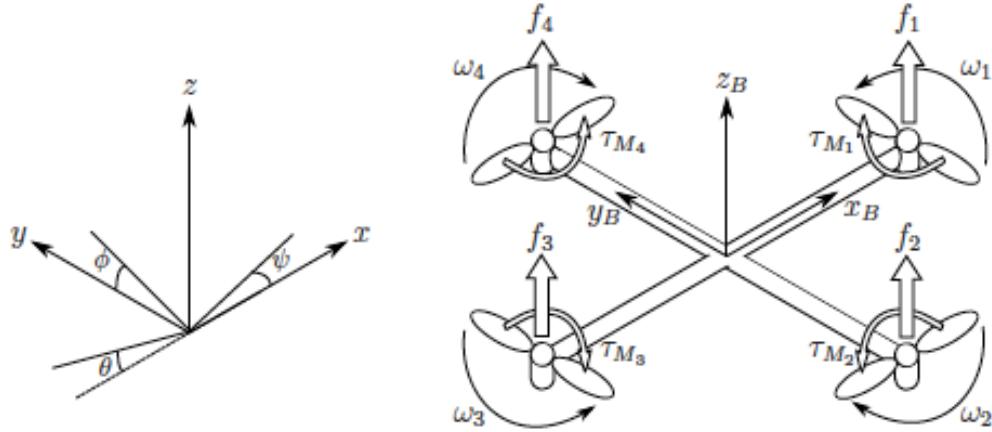


Fig. 2.1.1: Sistema di riferimento inerziale assoluto e sistema di riferimento inerziale solidale con il telaio
Definiamo dunque:

\mathbf{E}_i := sistema di riferimento inerziale assoluto

$$\mathbf{E}_i = [x \ y \ z]$$

\mathbf{E}_B := sistema di riferimento inerziale solidale con il corpo del drone (origine coincidente con il suo centro di massa)

$$\mathbf{E}_B = [x_B \ y_B \ z_B]$$

Indichiamo con ξ la posizione assoluta del velivolo nel sistema inerziale E_i e con η l'assetto del drone in E_i . Analizzando la Fig.1, possiamo, inoltre, indicare gli angoli di *roll* ϕ (rotazione attorno l'asse x), *pitch* θ (rotazione attorno l'asse y) e *yaw* ψ (rotazione attorno l'asse z).

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

Indichiamo con \mathbf{V}_B il vettore rappresentante le velocità lineari e con ω quello delle velocità angolari del sistema E_B :

$$\mathbf{V}_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

La matrice di rotazione dal sistema di riferimento E_B al sistema E_i , può essere definita nel seguente modo:

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

La matrice di trasformazione delle velocità angolari dal sistema di riferimento E_i al sistema E_B , può essere definita nel seguente modo:

$$\mathbf{W}_\eta = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Infatti:

$$\dot{\xi} = \mathbf{R}\mathbf{V}_B, \dot{\eta} = \mathbf{W}_\eta\omega$$

Avendo supposto simmetrica la struttura del velivolo rispetto agli assi x e y, è possibile definire la matrice diagonale di inerzia, nel seguente modo:

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

dove: $I_{xx} = I_{yy}$

2.2 Caratterizzazione dinamica

La velocità angolare del rotore i , chiamata ω_i , genera una forza \mathbf{F}_i lungo la direzione dell'asse del rotore. Essa rappresenta, la spinta fornita da ogni motore che si oppone alla forza di gravità.

La velocità angolare e l'accelerazione del rotore generano, inoltre, un momento torcente τ_{Mi} , attorno all'asse del rotore.

$$\mathbf{F}_i = k\omega_i^2, \tau_{Mi} = b\omega_i^2 + I_M\dot{\omega}_i$$

dove: k := coefficiente di spinta

b := costante di drag

I_M := momento di inerzia del rotore

La combinazione delle forze dei rotori genera una spinta T lungo l'asse z di E_B .

$$T = \sum_{i=1}^4 \mathbf{F}_i = k \sum_{i=1}^4 \omega_i^2$$

$$\mathbf{T}_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

Il momento torcente τ_B può essere espresso nel seguente modo:

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk(-\omega_2^2 + \omega_4^2) \\ lk(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^4 \tau_{Mi} \end{bmatrix}$$

dove: $l :=$ distanza tra il rotore e il centro di massa del quadricottero

Dunque, possiamo dedurre che:

- Si ha variazione dell'angolo di *roll* ϕ , diminuendo la velocità del rotore 2 e aumentando quella del rotore 4.
- Si ha variazione dell'angolo di *pitch* θ , diminuendo la velocità del rotore 1 e aumentando quella del rotore 3.
- Si ha variazione dell'angolo di *yaw* ψ , aumentando le velocità angolari di due rotorì opposti e diminuendo quelle degli altri due.

2.3 Modello di controllo

Dopo aver analizzato la relazione tra velocità di rotazione e momenti che i motori generano, è possibile definire le quattro uscite che i controllori dovranno generare per porre il drone nella posizione desiderata:

$u_1 = b (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$: spinta T 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 dell'asse x_B , ovvero provoca una variazione dell'angolo ϕ

$u_3 = lb (-\omega_1^2 + \omega_3^2)$: momento che modifica l'inclinazione dell'asse y_B , ovvero provoca una variazione dell'angolo θ

$u_4 = d (-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2)$: momento torcente totale, utilizzato per far ruotare il drone attorno all'asse z_B , ovvero provoca una variazione dell'angolo ψ

Dove $b :=$ coefficiente di spinta
 $d :=$ resistenza aerodinamica

2.4 Conversione dell'uscita di controllo in velocità di rotazione

Ponendo il sistema ottenuto nella [sezione 2.3](#) in forma matriciale si ottiene:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -lb & 0 & lb \\ -lb & 0 & lb & 0 \\ -d & d & -d & d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

Invertendo la matrice, si ottiene:

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & 0 & -\frac{1}{2lb} & -\frac{1}{4d} \\ \frac{1}{4b} & -\frac{1}{2lb} & 0 & \frac{1}{4d} \\ \frac{1}{4b} & 0 & \frac{1}{2lb} & -\frac{1}{4d} \\ \frac{1}{4b} & \frac{1}{2lb} & 0 & \frac{1}{4d} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

Ovvero, si ottengono i valori di rotazione dei singoli propulsori, poiché gli ingressi non possono essere inviati direttamente ai motori (ognuno di essi è generato in modo svincolato dagli altri).

2.5 Coefficienti di spinta e di resistenza aerodinamica

Al fine di tarare i PID nel modo corretto è necessario conoscere i coefficienti b e d definiti nella sezione [2.3](#).

Le eliche del quadricottero, durante la rotazione, sono sottoposte a diverse forze e momenti, tra cui la spinta (T). Il coefficiente di spinta (b) può essere così definito:

$$b = \frac{T}{m \cdot v} = \frac{T}{(\rho \cdot A \cdot v) \cdot v}$$

Dove m := 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) (valore costante) e la velocità (v).

v := velocità del rotore

Al fine di generare la spinta necessaria al volo è importante analizzare la velocità (v) che deve assumere il rotore:

$$v = \omega \cdot r$$

Dove ω := velocità di rotazione

r := raggio della pala (valore costante)

Dunque, è possibile ottenere il valore del coefficiente b nel seguente modo:

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

Il coefficiente di resistenza aerodinamica (d) può essere definito come segue:

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

Dove $C_d :=$ resistenza aerodinamica

2.6 Controllori PID

Il sistema di controllo PID (Proporzionale – Integrativo – Derivativo) è il sistema di controllo più comune. Il controllore, un sistema a ciclo chiuso, riesce a definire e correggere l'errore che si viene a creare dalle variabili prese in considerazione. In particolar modo, tale errore, è definito come la differenza tra il valore osservato e quello desiderato. Dunque, l'uscita di un controllore PID è un valore di controllo che tende a portare il sistema più vicino possibile allo stato desiderato.

Un regolatore PID può essere dunque rappresentato nel seguente modo:

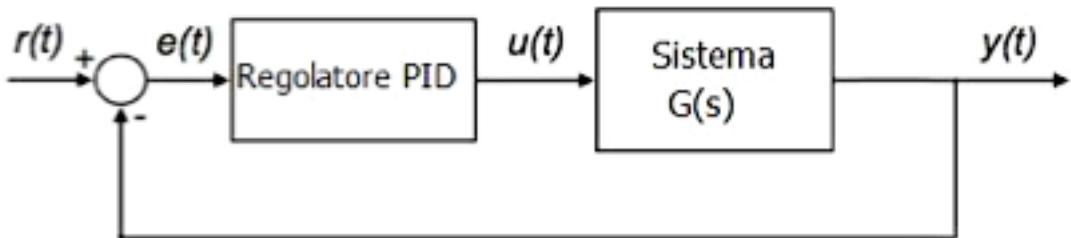


Fig. 2.6.1: Controllore PID

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

Il controllore produce un segnale di output definito come la somma pesata di tre termini:

- *Termine proporzionale (P):* produce un segnale in uscita proporzionale all'errore che si genera tra il valore della variabile attuale e di quella desiderata.

$$P = k_P e(t)$$

Tale relazione esprime il fatto che maggiore sarà l'errore all'ingresso del controllore e maggiore sarà l'azione di controllo svolta dal regolatore.

Aumentando in particolare il valore di k_P , la stabilità migliora ma il tempo di risposta peggiora, viceversa se k_P diminuisce.

- *Termine integrale (I):* restituisce l'integrale dell'errore. L'errore viene azzerato e ogni qual volta risulta essere non nullo, il comando integrativo invia un segnale per farlo convergere a zero.

$$I = k_I \int_0^t e(t) dt$$

Spesso l'azione integrale è associata all'azione proporzionale, realizzando i controllori PI che permettono di ottenere una maggior precisione e velocità di risposta, senza peggiorare la stabilità del sistema. Questi controllori vengono utilizzati soprattutto in sistemi in cui le variazioni di carico avvengono lentamente. Anche in questo caso, aumentando il valore di k_I , la stabilità migliora con un peggioramento del tempo di risposta, viceversa se k_I diminuisce.

- *Termine derivativo (D)*: agisce sulla derivata dell'errore. Il segnale di controllo viene dato non appena l'errore inizia ad aumentare o diminuire.

$$D = k_D \dot{e}(t)$$

L'azione derivativa è in grado di prevedere il comportamento del sistema (per questo motivo viene chiamato anche controllore anticipatore), migliorando il tempo di assestamento e la stabilità dello stesso. In questo caso l'uscita del controllore dipende dalla velocità con cui varia l'errore.

Dove i coefficienti k_P , k_I , k_D sono i guadagni di controllo dei rispettivi termini.

L'ingresso di controllo in funzione del tempo può essere definito nel seguente modo:

$$u(t) = k_P e(t) + k_I \int_0^t e(t) dt + k_D \dot{e}(t)$$

La funzione di trasferimento equivalente nel dominio di Laplace del controllore PID:

$$L(s) = K_P + \frac{K_I}{s} + K_d s$$

Dove $s :=$ variabile complessa

Se i termini proporzionale, integrale e derivativo vengono scelti in maniera errata, l'ingresso del processo controllato può essere instabile: in questo caso si otterrebbe un'uscita divergente, con oscillazioni.

2.6.1 Controllo dell'assetto

Gli angoli di *roll*, *pitch* e *yaw* in output dal controllore, sono ricavati valutando i singoli errori che si ha per ogni angolo relativo al sistema di riferimento del quadricottero.



Figura 2.4.1.1: loop di controllo per l'altitudine



Figura 2.4.1.2: loop di controllo per l'angolo di pitch ϕ

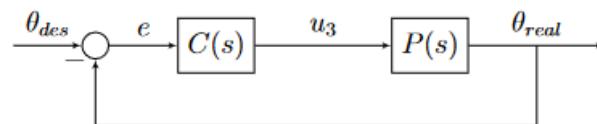


Figura 2.4.1.3: loop di controllo per l'angolo di roll θ



Figura 2.4.1.4: loop di controllo per l'angolo di yaw ψ

È possibile notare che il drone preso in esame ci permette di utilizzare un approccio SISO, descrivendo un loop di controllo per ogni angolo di assetto.

Utilizzando i controllori PID, l'uscita del controllore $C(S)$ può essere schematizzata nel seguente modo:

$$u_1 = K_{P,z}(z_{des} - z_{real}) + K_{D,z} \frac{d(z_{des} - z_{real})}{dt} + K_I \int (z_{des} - z_{real}) dt$$

$$u_2 = K_{P,\phi} (\phi_{des} - \phi_{real}) - K_{D,\phi} \dot{\phi}_{real}$$

$$u_3 = K_{P,\theta} (\theta_{des} - \theta_{real}) - K_{D,\theta} \dot{\theta}_{real}$$

$$u_4 = K_{P,\psi} (\psi_{des} - \psi_{real}) - K_{D,\psi} \dot{\psi}_{real}$$

In questo caso si è considerato un controllore di tipo proporzionale e derivativo (PD) poiché quello che ci interessa è il controllo della dinamica rotazionale nel periodo corto (termine proporzionale e derivativo hanno una maggiore influenza nel breve periodo, dunque il termine integrativo, che ha lo scopo di andare a ridurre l'errore a regime per tempi elevati, può essere trascurato) e gli angoli di roll, pitch e yaw desiderati sono posti a 0.

Le velocità angolari vengono calcolate da un giroscopio digitale ([IMU](#)) e si riferiscono al rateo di velocità angolare rispetto agli assi di riferimento del drone. Successivamente vengono inviate all'[ESC](#) per creare il comando necessario per ottenere la velocità adatta.

3. Hardware

3.1 Scheda STM32H7 Nucleo-144 boards

3.1.1 Cos'è un microcontrollore

Per lo sviluppo del funzionamento del drone preso in analisi è stato utilizzato un microcontrollore, ovvero un dispositivo integrato su un singolo circuito elettronico, progettato per interagire direttamente con il mondo esterno tramite un programma presente nella memoria interna e dei pin specializzati configurati appositamente.

3.1.2 Caratteristiche

Il MCU utilizzato in questo caso, è la scheda **STM23H745 NUCLEO – 144**. Essa è costituita da due processori (core):

- Processore ARM Cortex M4 a 32 bit
- Processore ARM Cortex M7 a 32 bit

In particolare, il progetto è stato sviluppato utilizzando il primo processore.

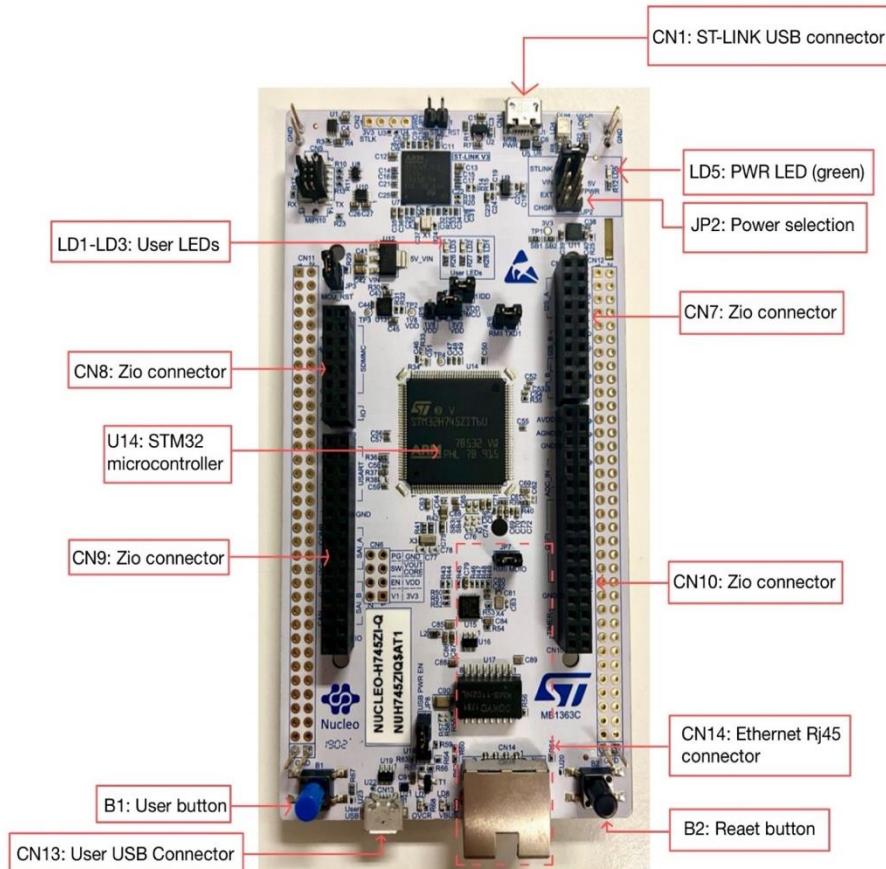


Fig. 3.1.2.1 : Componenti principali della scheda STM32H7 NUCLEO – 144

La scheda ha dimensioni 13.34 x 7 cm ed un peso di 60 g.

Per stabilire il collegamento con la scheda STM32, essa deve essere alimentata collegando l'ingresso CN1 della stessa alla porta USB del PC. Altrimenti può essere fornita alimentazione tramite:

- Un alimentatore esterno da 7-11 V collegato a CN8 pin 15 o CN11 pin 24
- Un alimentatore esterno da 5 V collegato al pin 6 di CN11
- Un caricatore USB esterno da 5 V (5V_USB_CHGR) collegato a CN1
- Un alimentatore esterno da 3,3 V (3V3) collegato a CN8 pin 7 o CN11 pin 16

Anche la scheda è in grado di fornire alimentazione tramite due uscite di alimentazione:

- 5V (CN8 pin 9 o CN11 pin 18): può essere utilizzato come alimentatore di uscita
- 3,3 V (CN8 pin 7 o CN11 pin 16): può essere utilizzato anche come uscita di alimentazione

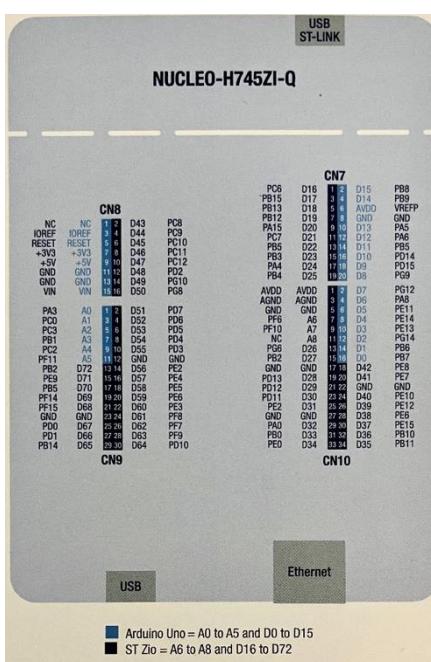
Table 9. External power sources maximum currents

Input power name	Connector pins	Voltage range	Max current	Limitation
VIN	CN8 pin 15 CN11 pin 24	7 V to 11 V	800 mA	From 7 V to 11 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<11 V
EXT	CN11 pin 6	4.75 V to 5.25 V	500 mA	-
CHGR	CN1	5 V	-	-
3.3V	CN8 pin 7 CN11 pin 16	3 V to 3.6 V	-	-

Fig. 3.1.2.2 : Valori di limitazione delle correnti per fonti di alimentazione esterne

La scheda è inoltre dotata di LED, Pulsanti, USART e collegamenti USB, ETHERNET (si rimanda alla Fig. 3.1.2.1).

Ciascun pin della scheda può essere programmato appositamente tramite l'ioc del software CUBE-IDE. Nella scheda STM32H745 i pin sono disposti sui connettori della parte superiore nel seguente modo:



3.2 Motori

Il drone in esame monta 4 motori brushless **Turnigy D3536/9 910KV**.

I motori brushless sono ideali per le applicazioni di piccola media potenza, con rapide accelerazioni e frenate o inversioni del moto, a causa del loro ingombro limitato, a differenza della coppia e della potenza che sono in grado di erogare. Essendo privi di contatti strisciati (spazzole) sull'albero motore, la necessità di manutenzione di tale motore risulta estremamente ridotta.

Il rotore (parte esterna) è dotato di magneti permanenti, i quali costituiscono la parte variabile del motore. La parte fissa è invece definita dallo statore (parte interna). Quest'ultimo è costituito da denti e cave, in cui sono presenti gli avvolgimenti statorici che, opportunamente alimentati, generano il campo magnetico che provoca la rotazione del rotore.



Fig. 3.2.1 : Funzionamento motore brushless

La commutazione di un motore brushless è di tipo elettronica ed è, quindi, effettuata da un banco di transistori gestiti da un microcontrollore ([ESC](#)).



Fig. 3.2.2 : Motore Turnigy D3536/9 910kv a sinistra, ESC Turnigy Plush-30A a destra

Di seguito è fornita 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

3.3 ESC

3.3.1 Cos'è e come funziona

L'ESC, acronimo inglese di "Electronic speed control" (Controllore elettronico della velocità), è un circuito elettronico che controlla e regola la velocità di un motore elettrico. È un componente fondamentale per il corretto funzionamento di un drone quadrirotore, poiché ha la funzione di collegare il controllore di volo con i motori, in modo da consentire la regolazione di questi ultimi. Poiché ogni motore ha un regime stazionario differente rispetto agli altri, ogni motore è dotato di un suo ESC: in particolare il drone preso in esame monta 4 **ESC Turnigy Plush-30A** ([Fig. 3.2.2](#)).

Un motore brushless richiede che la sua velocità venga variata regolando i tempi degli impulsi di corrente erogati ai vari avvolgimenti statorici. Dunque, i sistemi ESC per un motore brushless trifase generano corrente alternata trifase, come in un variatore di frequenza. La fase corretta varia con la rotazione del motore, che viene presa in considerazione dall'ESC (rilevata dalla forza controelettromotrice del motore). Tale dispositivo, permette di gestire e regolare la velocità di rotazione andando a variare la percentuale di segnale PWM inviata ai motori.

In questo caso l'ESC è dotato di un connettore, chiamato BEC (Battery Eliminator Circuit), ovvero un circuito elettronico progettato con lo scopo di fornire una tensione continua e stabilizzata, ricavata dalla batteria centrale che alimenta i motori, dunque senza utilizzare batterie aggiuntive.

Di seguito è fornita la scheda tecnica degli ESC utilizzati:

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

3.3.2 Come programmare un ESC

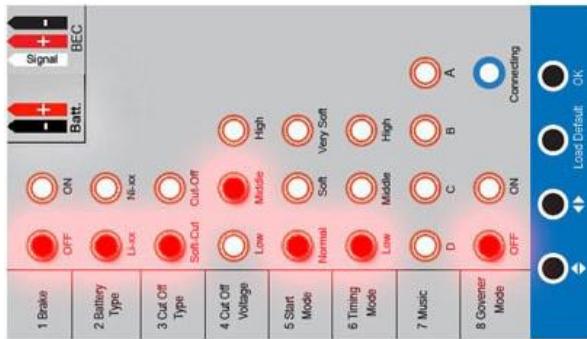


Fig 3.3.2.1 : Scheda di programmazione ESC

Per la programmazione dell’ESC è necessario collegarlo alla scheda di programmazione del cavo BEC e alla batteria, quindi settare i led con gli appositi pulsanti (Fig. 3.3.2.1).

Durante la fase di armamento dei motori è fondamentale inviare un segnale PWM sulla linea dati (BEC) dell’ESC con un duty cycle di ampiezza pari a 4,75%. Se tale fase è terminata correttamente, gli ESC emetteranno un breve segnale audio melodico, altrimenti verranno emessi ripetutamente due BEEP rapidi se si è verificato un problema.

Una volta eseguita tale procedura, è necessario attendere qualche secondo per evitare che i motori si disarmino. Il range di lavoro per i motori utilizzati varia da un duty cycle minimo del 6% ad un massimo del 10%.

Gli ESC vengono collegati alla scheda STM tramite tre collegamenti:

- Alimentazione (uscita 5V della scheda): corrispondente al cavo rosso
- Segnale PWM (si faccia riferimento alla sezione [7.2.1](#)): corrispondente al cavo bianco
- GND: corrispondente al cavo nero



Fig. 3.3.2.2: Collegamenti ESC

3.4 Eliche

Di rilevante importanza per il funzionamento corretto del drone, sono le eliche. È infatti necessario utilizzare una tipologia di eliche adatte al motore, affinché i motori riescano a dare la spinta necessaria al drone per far sì che riesca a volare, questo avviene grazie alla spinta contrapposta generata dalle due coppie di eliche che roteano in senso opposto.

Lo schema di montaggio sarà perciò caratterizzato da due coppie di eliche 1-3 (e motori) che roteano in senso antiorario, e due coppie di eliche 2-4 (e motori) che roteano in senso orario, come possiamo osservare nello schema (Fig.3.4.1).

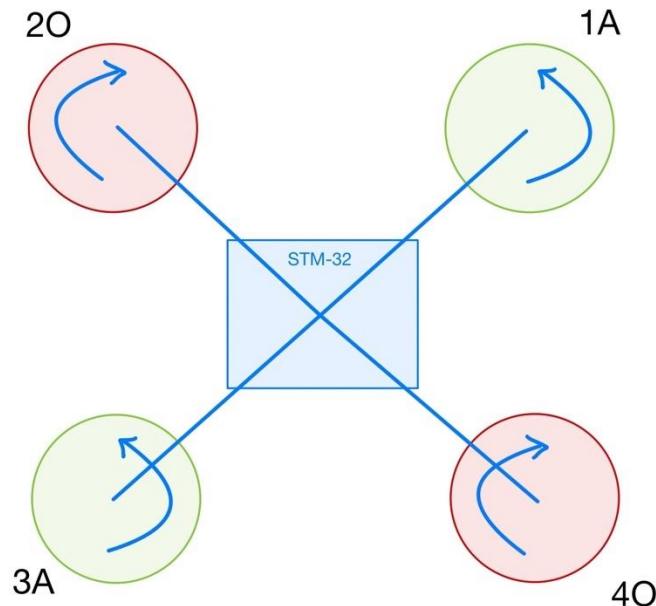


Fig.3.4.1: Schema montaggio eliche

Ai motori sono state collegate in modo assiale eliche bipala **Slow Fly Electric Prop 9057SF** da 9x4.7 pollici, ciascuna in grado di fornire una spinta di 0.632 Kg a piena velocità.



Fig 3.4.2: Elica Slow Fly Electric Prop 9057SF

3.5 Batteria

3.5.1 Caratteristiche

Nel drone preso in considerazione è stata utilizzata la batteria **Turnigy A-Spec 2600mAh**.



Fig. 3.5.1.1 : 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

3.5.2 Lipo Saver

Per evitare che la batteria raggiunga una tensione (per cella) inferiore ai 2.7 V, (ovvero per evitare il danneggiamento) è stata inserita una scheda detta Lipo Saver, che avverte, tramite un diodo, quando la tensione scende sotto dei 3V (per cella).



Fig. 3.5.2.1 : Lipo Saver

3.6 OLED Display

Per la visualizzazione dei dati acquisiti dai sensori si è disposto un OLED (Organic Light Emitting Diode) display, collegato direttamente alla scheda STM32H745, ovvero caratterizzato non solo di una buona luminosità e di un elevato rapporto di contrasto, ma anche di un consumo energetico molto basso.

Il display utilizzato per il drone è del tipo **SSD1306**, il quale ha la caratteristica di utilizzare un’interfaccia seriale I2C che richiede solo due pin di dati. Il circuito di reset è a bordo, per questo motivo non è necessario un pin di reset aggiuntivo.



Fig. 3.6.1 : Display OLED SSD1306

Di seguito sono fornite le caratteristiche del display:

Dimensione	0.96”
Risoluzione	128 x 64 pixel
Tensione	3.3 V – 5 V
Consumo energetico	0.06 W
Durata	Non inferiore a 16000 ore
Materiale schermo	Vetro

3.7 Telaio

Il quadricottero preso in esame monta un telaio del tipo **X666 Glass Fiber Quadcopter Frame**, in alluminio e fibra di vetro. Larghezza 666mm, peso 415g. Il peso del telaio è una caratteristica fondamentale, da scegliere, in base ai motori utilizzati: lo scopo è quello di scegliere la struttura che rende minimo il peso totale del sistema. Tanto minore è la massa, tanto è minore la forza peso, tanto è minore la forza da vincere in condizione di volo livellato. Il materiale del telaio dipende dal rapporto peso/motorizzazione/capacità di carico.

Il telaio si presenta a forma di croce: nelle estremità sono installati i [motori](#), mentre lungo i bracci si trovano gli [ESC](#) e al centro di concentra il sistema di alimentazione e l’elettronica di controllo.



Fig. 3.6.1 : Render del concept del drone

3.8 Scheda di Potenza

La scheda di potenza, posta in posizione centrale nel telaio del quadricottero, sotto la scheda STM32H745 è un elemento molto importante del velivolo in quanto consente un adattamento dei vari segnali e provvede a fornire le alimentazioni ad ogni componente attivo del quadrirotore.

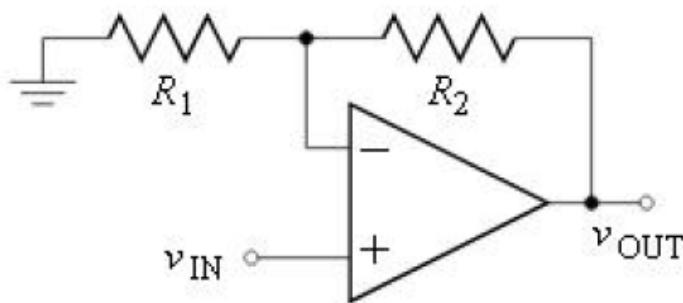
In seguito ad alcune modifiche effettuate a livello hardware si è preferito effettuare un collegamento diretto tra la scheda STM32H745 e i singoli componenti utilizzati per il funzionamento del drone. Per il funzionamento dei motori non è stato necessario inserire un'ulteriore scheda di alimentazione, in quanto alimentati direttamente dalla batteria e regolati opportunamente dall'ESC.

Di seguito vengono fornite due differenti versioni della scheda, progettate al pc tramite il software Cadsoft Eagle, sviluppate su pcb da un'azienda esterna.

3.8.1 Prima versione

La prima versione della scheda è composta da 4 circuiti:

- 1) Regolatore di tensione lineare a 5 V, utilizzato per alimentare la scheda microcontrollore STM32H745.
- 2) Regolatore di tensione a 3.3 V, utilizzato per alimentare eventuali circuiti inseriti in possibili sviluppi futuri.
- 3) Interruttore a MOSFET (comandato tramite adattatore di livello), utilizzato per pilotare l'accensione e lo spegnimento dei motori tramite pin digitale collegato al microcontrollore.
- 4) Adattatore di livello per segnali PWM, utilizzato per adattare i segnali PWM forniti dal controllore, a segnali riconoscibili dagli ESC.



La differenza principale tra questa versione e la seguente sono i blocchi contenenti gli adattatori di livello: nella prima scheda sono stati utilizzati adattatori di livello, il cui principio di funzionamento si basa su amplificatori operazionali in configurazione non invertente, per adattare i segnali in ingresso provenienti dal microcontrollore (0 – 3.3 V) a livelli riconoscibili dai circuiti a valle (0 – 5 V).

La tensione di uscita V_{out} è legata a quella di ingresso secondo la seguente relazione:

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$$

In questo caso, dunque, settando opportunamente i valori delle resistenze, si otterranno in uscita i livelli logici desiderati.

A causa di alcuni errori nella progettazione e nella mancanza di sistemi di sicurezza, si è deciso di riprogettare internamente la scheda, modificandone il layout e la struttura, definendo una seconda versione.

3.8.2 Seconda versione

Composta dagli stessi circuiti della prima, ma costituita da alcuni importanti miglioramenti, quali:

- Inserimento di sistemi di protezione elettrica mediante optoisolatori nei circuiti adattatori di livello
- Correzione di errori di progettazione
- Posizionamento dei connettori, dei fori per le viti di fissaggio
- Posizionamento al centro scheda dell'IMU (indispensabile per il corretto funzionamento del velivolo).
- Eliminazione delle alette di raffreddamento dei MOSFET (in modo da alleggerire il peso del velivolo stesso)

Di seguito vengono descritti i vari blocchi che compongono la scheda.

3.8.2.1 Regolatore di tensione lineare a 5V (LM7805)

Circuito alimentatore che permette, data la tensione di ingresso di 11.1 V proveniente dalla batteria (VDD), di produrre una tensione di uscita continua stabilizzata di 5V (VCC). Nel progetto viene utilizzato per fornire alimentazione alla scheda di controllo STM32H745.

Il principio di funzionamento del circuito si basa sul circuito integrato LM7805 (datasheet allegato) e sui due condensatori C1 e C4 (Fig. 3.8.2.1.1).

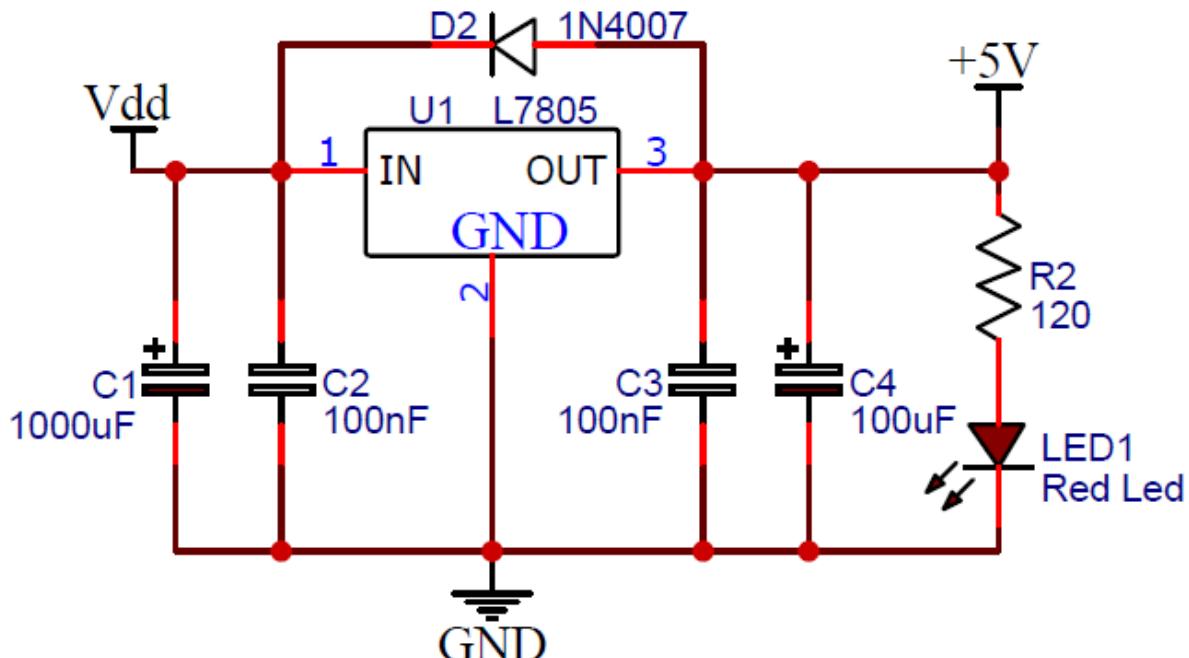


Fig. 3.8.2.1.1 : Regolatore di tensione lineare a 5V

Di seguito sono elencati i componenti che costituiscono il circuito:

- **Regolatore LM7805** (regolatore di tensione positiva, a tre terminali)

Tensione di ingresso	Compresa tra 7V e 20V continui
Tensione di uscita	+5V ($\pm 0.25\text{V}$)
Corrente di uscita massima	1 A
Protezione	Termica per sovraccarico Interna per corto-circuito con limitazione della corrente Package TO220

- **Condensatori (1000uF – 25V, 100uF – 35V, 100nF)**

- 1) $C1$: condensatore elettrolitico di filtro che viene sempre utilizzato in prossimità del regolatore (valore compreso tra 10uF e 4700uF).
- 2) $C2$ e $C3$: condensatori al poliestere, utilizzati per evitare auto oscillazioni (valore compreso tra 1nF e 100nF)
- 3) $C4$: condensatore elettrolitico, utilizzato per ridurre il ripple presente in uscita dal regolatore, rendendo la tensione più stabile (valore compreso tra 1uF e 470uF)

È importante notare che $C1$ deve avere un valore maggiore di $C3$, con un rapporto di 10 a 1, per evitare il danneggiamento del dispositivo.

- **Diodo 1N4007**: il diodo D2 (posto tra l'uscita e l'ingresso) viene utilizzato per proteggere il regolatore ogni qualvolta si spegne il circuito alimentatore. Senza tale diodo, la tensione immagazzinata dal condensatore C4 si scaricherebbe in senso inverso all'interno dell'integrato, cioè dall'uscita verso l'ingresso, danneggiandolo.
- **Led e Resistenza**: il led da 3mme la resistenza R2 vengono utilizzati per un semplice controllo visivo del funzionamento del circuito alimentatore.

3.8.2.2 Regolatore di tensione lineare a 3.3V (UA78M33)

Analogo al precedente, fatta eccezione per il circuito integrato utilizzato che in questo caso è l'UA78M33. Viene utilizzato per fornire una tensione continua stabilizzata di 3.3 V all'IMU.

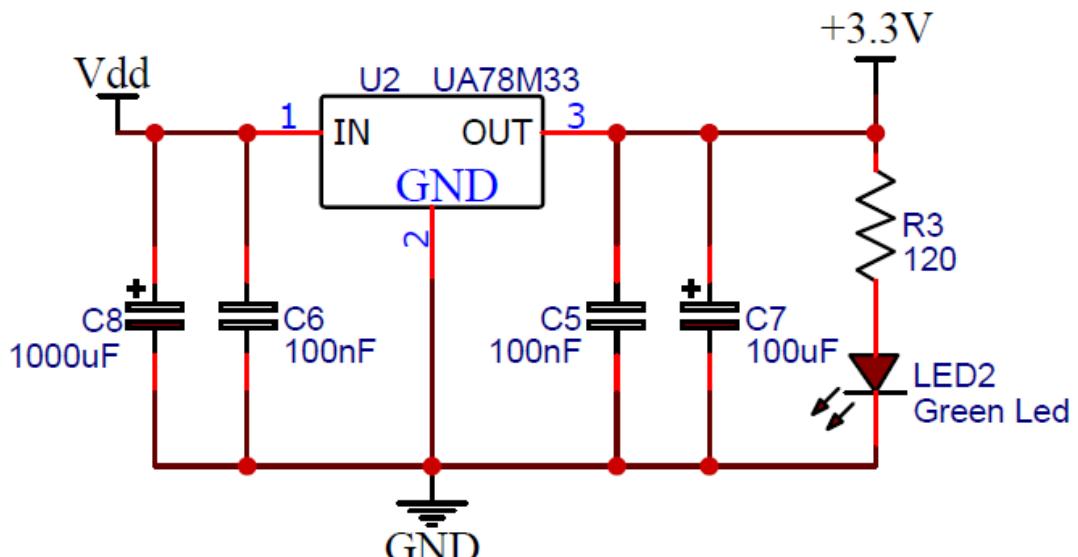


Fig. 3.8.2.2.1 : Regolatore di tensione lineare a 3.3V

Di seguito sono elencati i componenti che costituiscono il circuito:

- **Regolatore UA78M33**

Tensione di ingresso	Compresa tra 5.3 V e 25 V
Tensione di uscita	+3.3V ($\pm 0.1V$)
Corrente di uscita massima	0.5 A

3.8.2.3 Interruttore a MOSFET comandato tramite OPTO-commutatore

Il circuito è un interruttore a stato solido, comandato da un pin digitale della scheda di controllo tramite un adattatore di livello 3.3V – 5V. L'interruttore si è reso necessario, in quanto, l'interruttore a stato solido necessita di una tensione di gate VGS di almeno 4.5V. Tale circuito, agendo direttamente sull'alimentazione degli ESC, è stato sviluppato per permettere una corretta accensione degli stessi. Ciò si è reso necessario in quanto, studiando i datasheet, è emerso che per un corretto funzionamento bisognava fornire l'alimentazione principale solo dopo aver portato un segnale PWM di un determinato periodo sulla linea dati (armamento dei motori); per questo è stato deciso di realizzare questo interruttore, facilmente controllabile via software, che permette di interrompere e far circolare corrente sulla linea di alimentazione degli ESC.

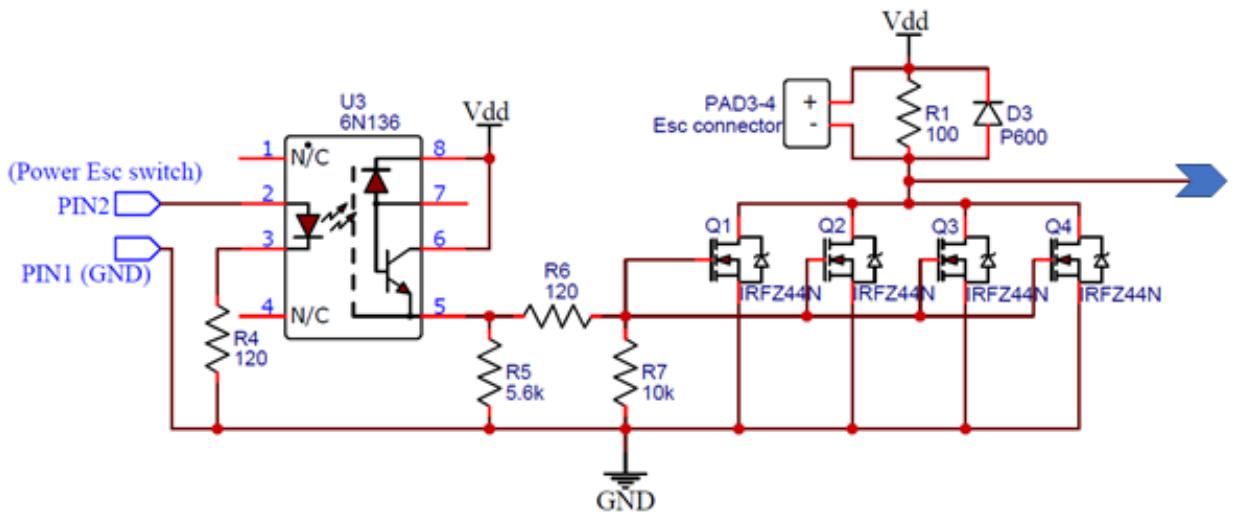


Fig. 3.8.2.3.1 : Interruttore a MOSFET comandato tramite OPTO – commutatore

Il circuito è costituito da:

- **MOSFET IRFZ44N:** il tipo di MOSFET è stato scelto, dopo alcune prove in base alle caratteristiche richieste dal progetto:
 - 1) Alta potenza dissipabile, con conseguente corrente Id maggiore di 10 A, in grado di sostenere l'elevata corrente assorbita dai motori (misurata sperimentalmente e risultata superiore ai 4 A per motore a piena potenza).
 - 2) Bassa resistenza interna, in modo da avere minori cadute di tensione a correnti elevate
 - 3) Facilità di parallelizzazione
 - 4) Possibilità di controllare il gate con segnali a 5V

L'IRFZ44N soddisfa tutte le caratteristiche elencate sopra:

Corrente massima Id	49 A
Potenza massima dissipabile	94 W
Resistenza interna RDS	17.5 mΩ
Tensione di controllo Vgs	> 4.5 V (Vds = 11.1 V , Id = 15 A)
Altro	Facilmente paralellizzabile, senza aggiunta di altri componenti



Fig. 3.8.2.3.2 MOSFET IRFZ44N

Il MOSFET è costituito da una resistenza interna di canale che, in caso di alte correnti (come il caso preso in esame), porta a cadute di tensione dell'ordine di qualche volt.

Dopo alcuni test si è verificato che, ad alte velocità di rotazione dei motori (con conseguenti correnti assorbite), un solo transistor portava a cadute di potenziale troppo elevate (causavano un abbassamento della tensione di alimentazione degli ESC fino al di sotto di 9 V, tensione minima necessaria per il corretto funzionamento degli stessi), dunque si è optato per inserire 4 MOSFET in parallelo anziché uno, in modo da ridurre la resistenza interna equivalente ad un quarto ($17.5/4 = 4.375\text{mOhm}$). Questa soluzione ha migliorato la dissipazione della potenza elettrica, in quanto ciascun transistor è attraversato da una sola parte della corrente assorbita dai motori (comportamento analogo al partitore di corrente).

Quando la tensione V_{gs} supera i 4.5 V, i MOS entrano in saturazione chiudendo il circuito e permettendo il passaggio della corrente. Quando la V_{gs} scende sotto la tensione di soglia e raggiunge gli 0 V, il transistor entra in zona di Cut – Off rendendo la sua resistenza interna infinita ed aprendo il circuito. Le resistenze R2 e R3 servono come protezione del circuito a monte dell'interruttore e a permettere la scarica a massa delle capacità parassite (che impedirebbero una corretta interdizione del canale del MOSFET).

- **DIODO P600:** Il **diodo** è un componente elettronico passivo non-lineare a due terminali (bipolo). D2 è un diodo di tipo general purpose per alte potenze dalle dimensioni molto ridotte, utilizzato come diodo di libera circolazione. La sua funzione è quella di proteggere il carico da picchi di tensione di verso opposto, dovuti a spegnimenti rapidi di carichi induttivi.

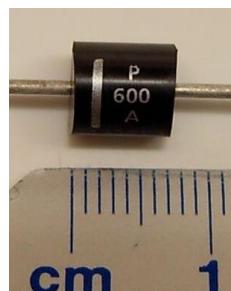


Figura 3.8.2.3.3 D2 DIODO P600

- **OPTOCOMMUTATORE 6N136:** Il circuito adattatore di livello viene utilizzato per modificare i livelli logici in ingresso provenienti dal pin digitale della STM32H74 (0 - 3.3V), innalzandoli a livelli riconoscibili dall'interruttore MOSFET (0–5v).



Figura 3.8.2.3.4 OTOCOMMUTATORE 6N136

3.8.2.4 Adattatore di livello per segnali PWM

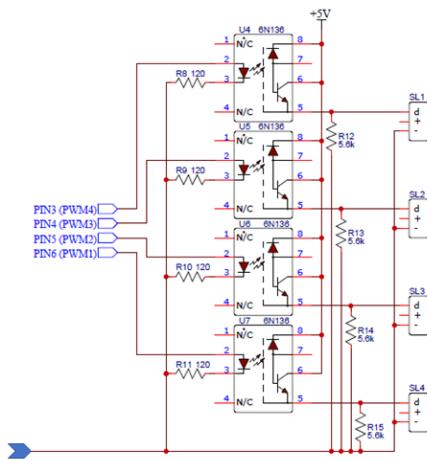
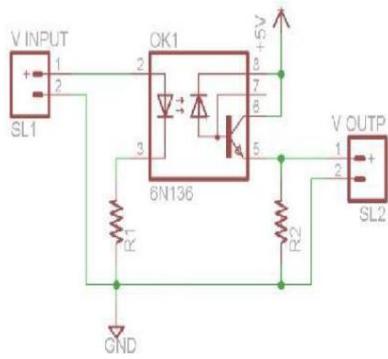


Figura 3.8.2.4.1

Questo circuito viene utilizzato per adattare i livelli in ingresso, provenienti dalle linee digitali e PWM della STM32H74 (0 – 3.3V), a livelli necessari per pilotare gli ESC (0 – 5 V).

Il circuito è costituito da più optoisolatori a rapida commutazione, modello 6N136 collegati come in figura sotto, in grado di fornire, oltre che una protezione elettrica dovuta ad una separazione dei circuiti, un adattamento di livello ottenuto mediante un corretto dimensionamento delle resistenze.



VINPUT	3.3V (Tensione in ingresso all'adattatore)
VOUTPUT	5V (Tensione in uscita)
Vd	1.45V (Tensione di funzionamento dell'emettitore)
IdMAX	20 mA (Corrente massima sull'emettitore)
VOL	0.4V (Tensione LOW Level ricevitore)
IFMAX	8 mA (Corrente massima ricevitore)
CTR	27 (Current Transfer Ratio)

3.9 Altimetro

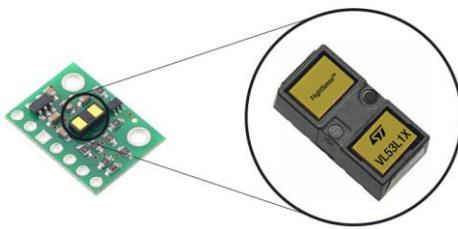


Figura 3.9.1 : Altimetro VL53L1X

L'altimetro è uno strumento di misura che permette di misurare la distanza verticale di un corpo da una superficie di riferimento.

Il VL53L1X (Fig.3.9.1) è un sensore di raggio laser Time-of-Flight (ToF) all'avanguardia di piccole dimensioni (4.9x2.5x1.56 mm), che migliora la famiglia di prodotti ST.

È il sensore ToF miniaturizzato più veloce sul mercato con una portata precisa fino a 4 m e una frequenza di portata rapida fino a 50 Hz.

Alloggiato in un contenitore miniaturizzato e riutilizzabile, integra un array di ricezione SPAD, un emettitore laser invisibile di Classe1 da 940 nm, filtri fisici a infrarossi e ottiche per ottenere le migliori prestazioni in varie condizioni di illuminazione ambientale con una gamma di opzioni di finestra di copertura.

A differenza dei sensori IR convenzionali, il VL53L1X utilizza la tecnologia ToF di ultima generazione di ST che consente la misurazione della distanza assoluta indipendentemente dal colore e dalla riflettanza del target.

È anche possibile programmare la dimensione della ROI sull'array ricevente, consentendo di ridurre il FoV del sensore.

La sua trattazione non era di nostra competenza poiché il drone risulta essere vincolato ad un braccio meccanico di altezza fissa, per cui per chiarimenti e funzionalità approfondite si rimanda al datasheet del prodotto.

3.9.1 Posizionamento

L'altimetro come l'IMU ha una posizione ben precisa (Fig.3.9.2). Il circuitino deve essere posizionato in modo che il sensore VL53L1X sia rivolto verso il basso perché ovviamente è utilizzato per monitorare la distanza del drone dal terreno.

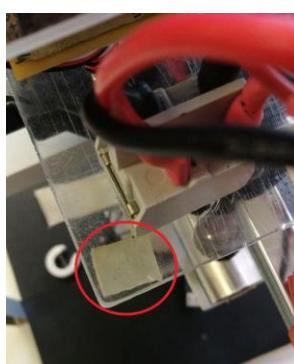


Figura 3.9.1.1 : Posizionamento Altimetro

3.10 IMU

Un'unità di misura inerziale o piattaforma inerziale (nota anche come *inertial measurement unit*, o IMU) è un sistema avionico che implementa il sistema di navigazione inerziale di un aeromobile. E' basato su sensori inerziali, come accelerometri, giroscopi e magnetometri che permettono un monitoraggio della dinamica di un mezzo in movimento.

L'IMU come l'altimetro ha una posizione ben precisa. Il circuitino deve essere posizionato al centro della lastra di plexiglass sopra all'apposito gommino adesivo (Fig.3.9.1), questo perchè il sensore IMU deve essere posizionato perfettamente nel centro di massa del drone.



Figura 3.9.1 Posizionamento IMU

3.10.1 Giroscopio: cos'è e come funziona

Il giroscopio è un dispositivo in grado di misurare la velocità di rotazione di un corpo rispetto ad un sistema di riferimento inerziale. Originariamente inventato nel 1800 da Jean Bernard Leon Foucault per dimostrare la rotazione della terra attorno al proprio asse, il giroscopio ha subito nel corso degli anni continui sviluppi e perfezionamenti sia per quanto riguarda il principio fisico operativo che le tecnologie costruttive adottate per la sua fabbricazione e miniaturizzazione.

I campi di applicazione di tale sensore sono molteplici, alcuni esempi possono essere:

- Settore automotive: controllo di stabilità e d'assetto, navigazione, controllo delle sospensioni, sistemi anticollisione;
- Settore consumer: periferiche del computer, controller per i giochi, equipaggiamento sportivo, videocamere, giocattoli;
- Settore industriale: navigazione autonoma di robot, controllo di sistemi idraulici, stabilizzazione di piattaforme di macchinari pesanti, trasporto umano, controllo di yaw dei mulini a vento;
- Settore aerospaziale: avionica, stabilizzazione nei sistemi di puntamento per antenne, UAV, sistemi di navigazione inerziale.

Il loro campo di applicazione è destinato ad ampliarsi in futuro, in virtù della loro facilità di integrazione e del tasso di incremento delle loro performance che aumentano di un fattore pari a 10 ogni 2 anni.

3.10.2 Accelerometro: cos'è e come funziona

L'accelerometro è un dispositivo in grado di misurare l'inerzia di una massa sottoposta ad un'accelerazione. Il principio alla base del funzionamento di quasi tutti gli accelerometri, nella sua forma più semplice, è la misura dello spostamento relativo nella direzione dell'asse sensibile di una massa di prova, collegata al telaio mediante un'opportuna rigidezza. Tale spostamento risulta direttamente proporzionale all'accelerazione agente sul corpo se la rigidezza non varia in funzione delle condizioni operative.

Di seguito è riportata una breve classificazione degli accelerometri in base al meccanismo di trasduzione impiegato:

- accelerometro estensimetrico (resistivo);
- accelerometro LVDT (Linear Variable Differential Transformer);
- accelerometro piezoelettrico;
- accelerometro termico;
- accelerometro laser;
- accelerometro capacitivo.

Si possono inoltre suddividere gli accelerometri in due macrocategorie:

- accelerometri per misure di accelerazione statica: adatti a rilevare accelerazioni continue statiche come esempio l'accelerazione di gravità.
- accelerometri per misure di accelerazione dinamica: in grado di rilevare solo le accelerazioni che variano nel tempo, come quelle generate da urti o vibrazioni.

3.10.3 Magnetometro: cos'è e come funziona

Il magnetometro è un sensore vettoriale che fornisce sia la direzione che il modulo del vettore del campo magnetico H usufruendo di tre unità sensibili posizionate ortogonalmente tra loro. Sono sensori costruttivamente semplici che non presentano al proprio interno parti in moto a differenza di accelerometri e giroscopi e il principio alla base del loro utilizzo è la determinazione dell'orientazione del vettore di magnetizzazione terrestre.

Le principali tecnologie costruttive permettono di classificare i magnetometri di uso più comune in quattro categorie:

- magnetometri a spirale;
- magnetometri uxgate;
- magnetometri ad effetto hall;
- magnetometri magnetoresistivi MR o AMR.

3.10.4 Sensore MPU-6050



Figura 3.10.4.1 : Scheda GY-86

La MPU-6050(Fig.3.9.4.1) è una piattaforma inerziale a basso costo e dalle dimensioni contenute (4x4x0.9 mm) prodotta da Invensense nel 2010 e utilizzata per la progettazione del nostro quadricottero. Integra al proprio interno un giroscopio a tre assi, un accelerometro a tre assi ed un Digital motion Processor, ovvero un microprocessore in grado di eseguire il sensor fusion tra questi due sensori. Tutto questo è asservito da 6 convertitori analogico digitali, in grado di garantire una risoluzione minima di 61microgrammi per l'accelerometro e 0:0076°/s per il giroscopio. La comunicazione con i registri interni del sensore avviene tramite un bus I₂C ad una velocità massima di 400 KHz ed ulteriori funzionalità sono la presenza di un sensore di temperatura e di un filtro passa basso, la cui frequenza di taglio può esser selezionata dall'utente tra diverse opzioni.

Le principali caratteristiche della MPU-6050 sono di seguito elencate per fornire una panoramica più chiara delle funzionalità di ogni singolo sensore:

Giroscopio:

- misura delle velocità angolari in formato digitale per gli assi X, Y, Z
- convertitore analogico digitale a 16 bit;
- filtro passa basso programmabile a: 256, 188, 98, 42, 20, 10, 5 Hz;
- corrente operativa assorbita: 3.6 mA;
- sensibilità termica (da -40°C a +85°C): _2%.

Accelerometro:

- misura delle accelerazioni in formato digitale per gli assi X, Y, Z
- convertitore analogico digitale a 16 bit;
- filtro passa basso programmabile a: 260, 184, 94, 44, 21, 10, 5 Hz;
- corrente operativa assorbita: 500 microA;
- sensibilità termica (da -40°C a +85°C): _0:02 %/°C.

Altre caratteristiche del sensore:

- comunicazione tramite bus I₂C ad una velocità massima pari a 400 KHz;
- presenza di un bus I₂C ausiliario per permettere la lettura di un sensore esterno (magnetometro);
- VDD operativa compresa tra 2.375 V e 3.46 V;

L'integrato MPU-6050 è saldato all'interno della scheda GY-86 la quale integra l'elettronica di base, un convertitore DC/DC, le resistenze di pull-up per il bus I₂C e le piazzole necessarie ad interfacciarsi facilmente con un microcontrollore.

Per le complete specifiche tecniche della MPU-6050 si rimanda al datasheet del sensore.

3.10.5 Sensore HMC5883L



Figura 3.10.5.1 : Sensore HMC5883L

L' HMC5883L è un magnetometro a basso costo dalle dimensioni contenute (3.0x3.0x0.9 mm) prodotto da Honeywell nel 2013. Progettato per misure di campi magnetici deboli è adatto ad esser utilizzato per determinare il vettore di magnetizzazione terrestre. Al suo interno sono presenti sensori magneto-resistivi ad alta risoluzione e l'elettronica necessaria ad amplificare, filtrare, rimuovere l'offset di misura. La presenza di tre convertitori ADC a 12 bit, uno per ciascun canale di lettura, garantisce una risoluzione minima inferiore a 1 mGauss. Il sensore è in grado di funzionare ad una frequenza massima di 75Hz in polling mode e permette di scambiare facilmente i dati con un microcontrollore attraverso il protocollo I₂C.

Le principali caratteristiche funzionali del magnetometro sono di seguito riportate:

- misura di intensità del campo magnetico in formato digitale per gli assi X, Y, Z;
- convertitore analogico digitale ADC a 12 bit;
- VDD operativa compresa tra 2.16 V e 3.6 V;
- corrente operativa assorbita: 100 microAmpere;
- output data rate programmabile a: 0.75, 1.5, 3, 7.5, 15, 30, 75 Hz;
- velocità massima I₂C: 400 KHz;

L'integrato HMC5883L è saldato all'interno della scheda GY-86, la quale integra l'elettronica di base, un convertitore DC/DC, le resistenze di pull-up per il bus I₂C e le piazzole necessarie ad interfacciarsi al microcontrollore. Per le complete specifiche tecniche del HMC5883L si rimanda al datasheet del sensore.

4. Fonti

- Materiale fornito dal docente
- Teppo Luukkonen - "Modelling and control of quadcopter", Aalto University School of Science

5. Software

5.1 Caratteristiche generali

L'implementazione del sistema è stata effettuata utilizzando il CubeIDE – MX della STMicroelectronics, che permette di effettuare lo sviluppo (configurazione delle periferiche, generazione e compilazione del codice e funzionalità di debug) tutto in uno per microcontrollori e microprocessori.

STM32Cube è un pacchetto di librerie dell'azienda STMicroelectronics, utilizzato con lo scopo di facilitare la produttività e lo sviluppo ai programmati. Tra i pacchetti di librerie del software è possibile definire la STM32Cube hardware abstraction layer (HAL) : libreria API di alto livello orientata alle funzionalità che nascondono la complessità periferica all'utente finale.

Il linguaggio utilizzato per l'implementazione del software è il C/C++, ed è basato sul framework Eclipse, sul toolchain GCC per lo sviluppo e GDB per il debugging.

Per creare un nuovo progetto è necessario cliccare su *File / New / STMProject*, quindi si aprirà una nuova schermata in cui poter selezionare il MCU della STM sul quale effettuare il progetto. In questo caso è stata selezionata la NUCLEO-H745ZI-Q:

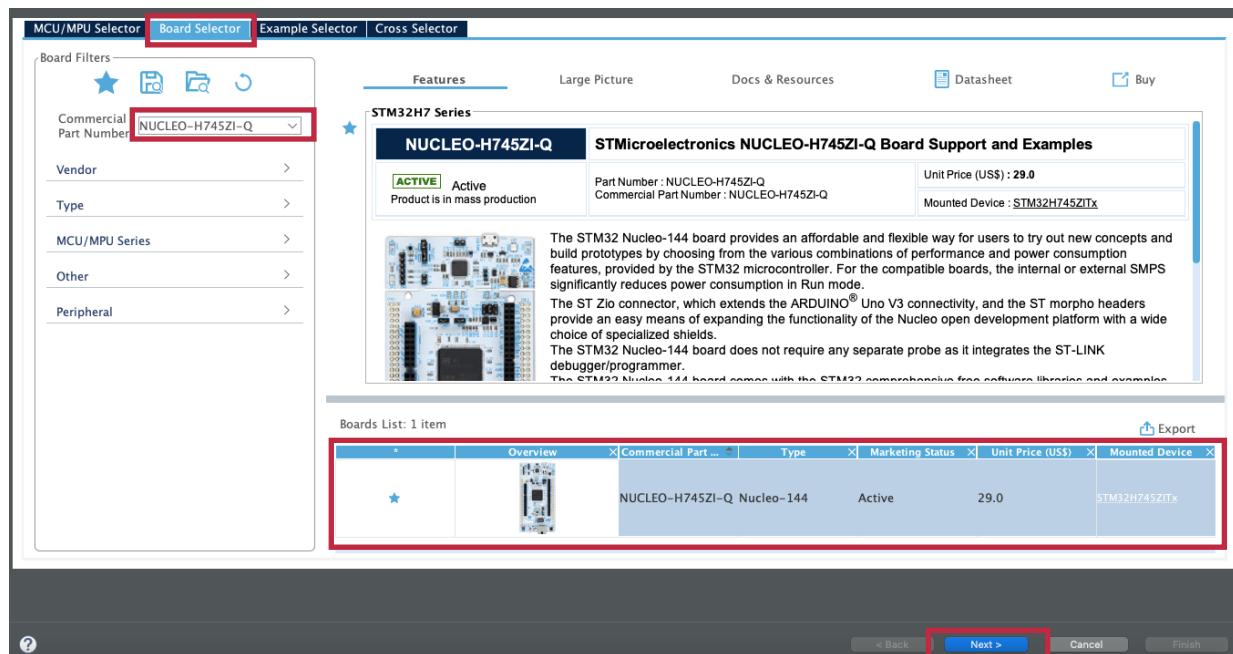


Fig. 5.1.1 : Generazione di un nuovo progetto

Dopo aver selezionato il MCU STM32 desiderato, viene creato il progetto e generato il codice di inizializzazione. In qualsiasi momento, durante lo sviluppo del progetto, il programmatore può tornare all'inizializzazione e rigenerando il codice senza alcun impatto sul codice utente scritto. Il settaggio dei pin e dei registri (scheda STM32) viene facilitato tramite l'utilizzo della configurazione grafica nel file .ioc, la quale permette di generare automaticamente il codice in C associato alle varie modifiche effettuate graficamente.

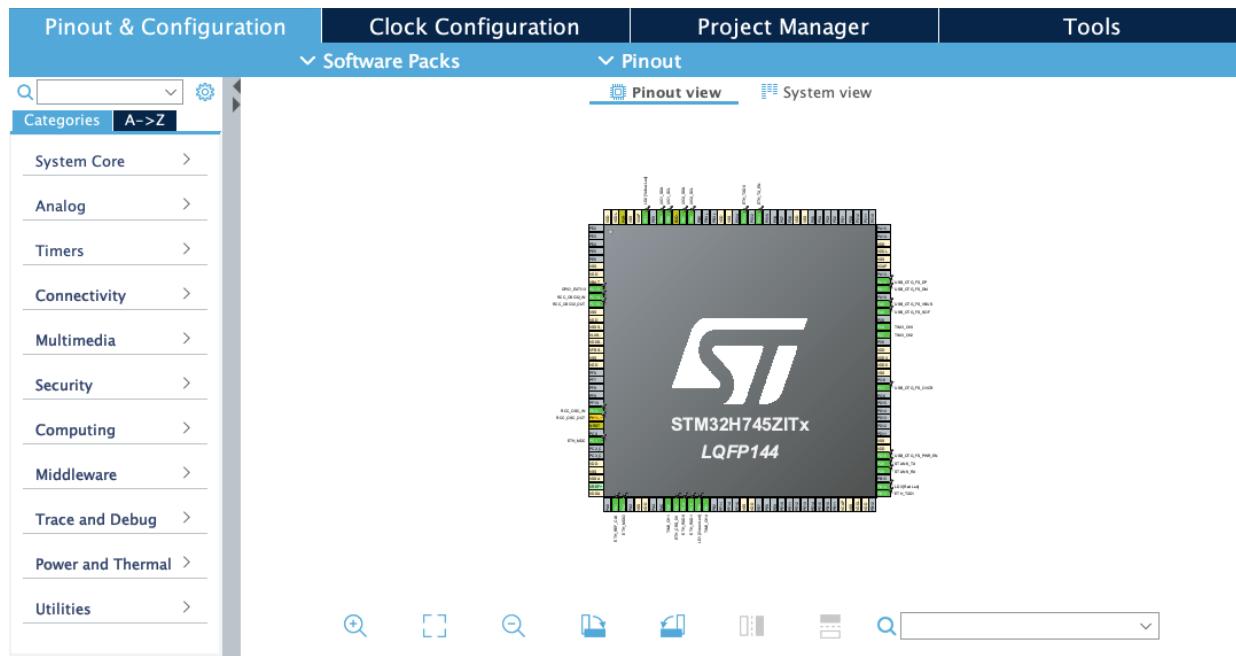


Fig. 5.1.2 : Interfaccia grafica del file .ioc

La configurazione dei pinout, utilizzati nello sviluppo del drone, è stata effettuata nelle seguenti sezioni:

- *System Core*: GPIO (General – Purpose Input Output)
 - *Timers*: TIMER

Per la gestione dei segnali da inviare ai motori (segnali PWM) è stato utilizzato un TIMER. Esso viene utilizzato sia per la fase di armamento dei motori, che per la fase di alimentazione in real time. In questo caso è stato utilizzato un timer di scopo generale (General Purpose Timer) costituito da quattro canali indipendenti necessari per generare i quattro segnali PWM.

- *Connectivity:* PROTOCOLLO I2C

Per la gestione delle comunicazioni con i dispositivi esterni è stato utilizzato il protocollo I2C che permette la connessione seriale sincrona di circuiti integrati. In questo studio la scheda STM32H745 funge da master, mentre i dispositivi esterni (quali IMU, magnetometro e schermo OLED Display) si comportano da slave. Tale protocollo richiede la disponibilità di due linee di trasmissione, una per i dati (SDA) e l'altra per il segnale di clock (SCL). La scheda STM dispone di 4 differenti canali I2C.

Essendo la scheda [STM32H745](#) costituita da due MCU core, al momento della generazione del codice, il software genera due differenti progetti : il primo da sviluppare sul core CM4, il secondo su CM7. Questo progetto è stato sviluppato utilizzando il core M4, dunque è stato necessario inserire il codice seguente (Fig. 4.1.2) nel file “main.c” in M7 per permettere al core M4 di essere il core di partenza predefinito.

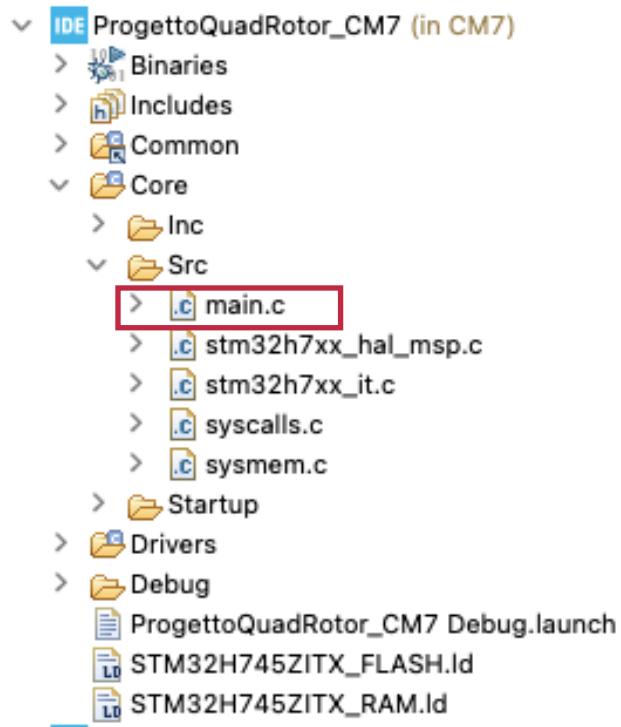


Fig. 5.1.3: ProgettoQuadRotor_CM7/Core/Src/main.c

```

112 /* USER CODE BEGIN Boot_Mode_Sequence_1 */
113 /* Wait until CPU2 boots and enters in stop mode or timeout*/
114 #ifdef DEBUG
115     while((__HAL_RCC_GET_FLAG(RCC_FLAG_D2CKRDY) != RESET));
116 #else
117     timeout = 0xFFFF;
118     while((__HAL_RCC_GET_FLAG(RCC_FLAG_D2CKRDY) != RESET) && (timeout-- > 0));
119     if (timeout < 0)
120     {
121         Error_Handler();
122     }
123 #endif

```

Fig. 5.1.4: Codice inserito nel file “main.c” in M7

È utile precisare che prima di avviare il debug del CM4 per la prima volta, è necessario avviare dapprima il debug sul CM7.

5.1.1 Struttura e main del programma

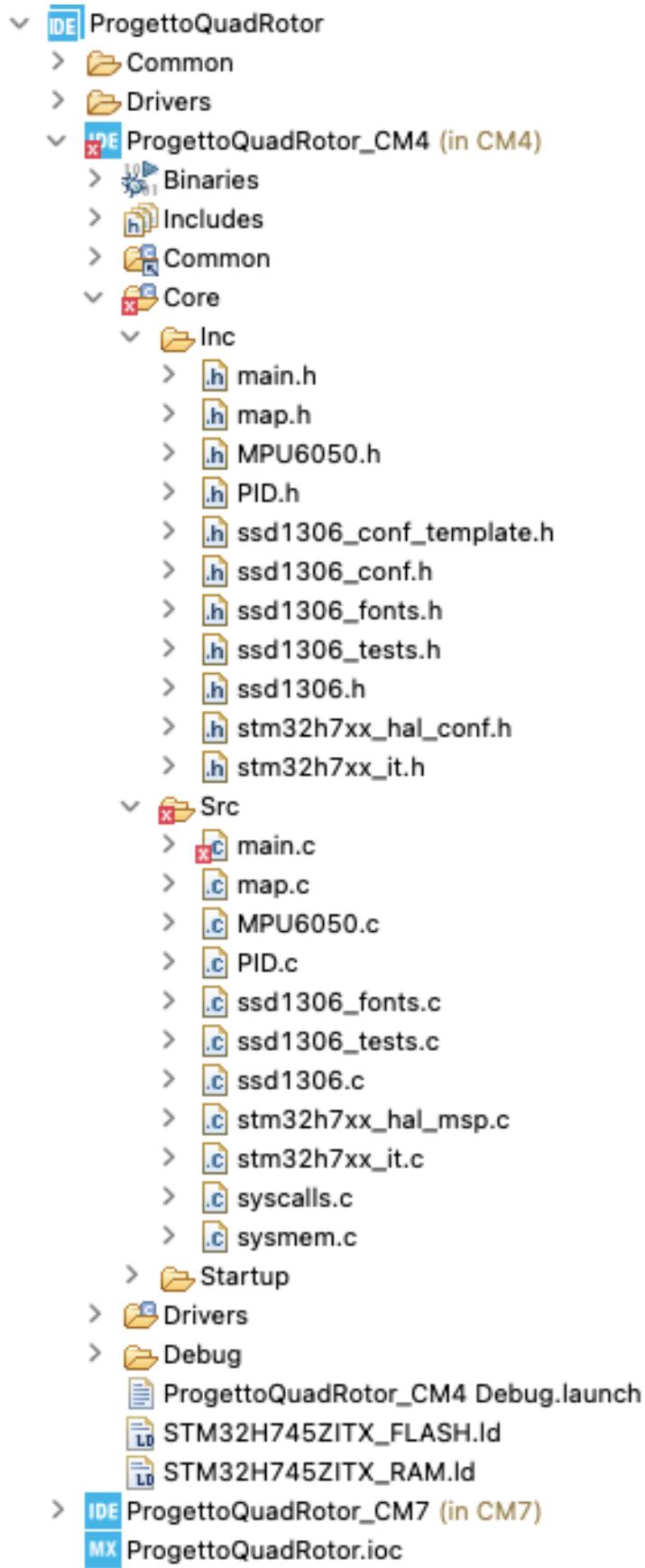


Fig. 5.1.1.1: Struttura progetto

• **ProgettoQuadRotor_CM4/Core/Src/main.c**: il main del programma si compone di tutte quelle funzioni necessarie per il funzionamento del drone nel suo complesso. In particolar modo, nello USER CODE 0, sono state inserite tutte quelle funzioni necessarie al calcolo, alla lettura e alla scrittura dei dati, mentre nello USER CODE 2 sono state definite le inizializzazioni dei TIMER, dei sensori e dei PID. Nello USER CODE 3, interno al while(1), vengono richiamate le funzioni dapprima definite ed è stato definito il codice riguardante il funzionamento dei motori.

L'avviamento dei motori è stato gestito tramite l'utilizzo dello USER BUTTON (tasto azzurro): nel momento in cui viene premuto, si attivano due led (LD2, LD3) e parte la fase di avviamento. Lo spegnimento avviene premendo lo stesso USER BUTTON e viene confermato con il conseguente spegnimento del led.

Per il codice relativo all'attivazione del BUTTON e dei LED si rimanda alla sezione [5.1.1.1](#) e [5.1.1.2](#).

5.1.1.1 .ioc

Nel file **.ioc**, ed in particolare nella sezione *System Core*, è stato attivato sul Cortex - M4 il GPIO corrispondente al led LD3 (red led), ovvero il pin PB14.

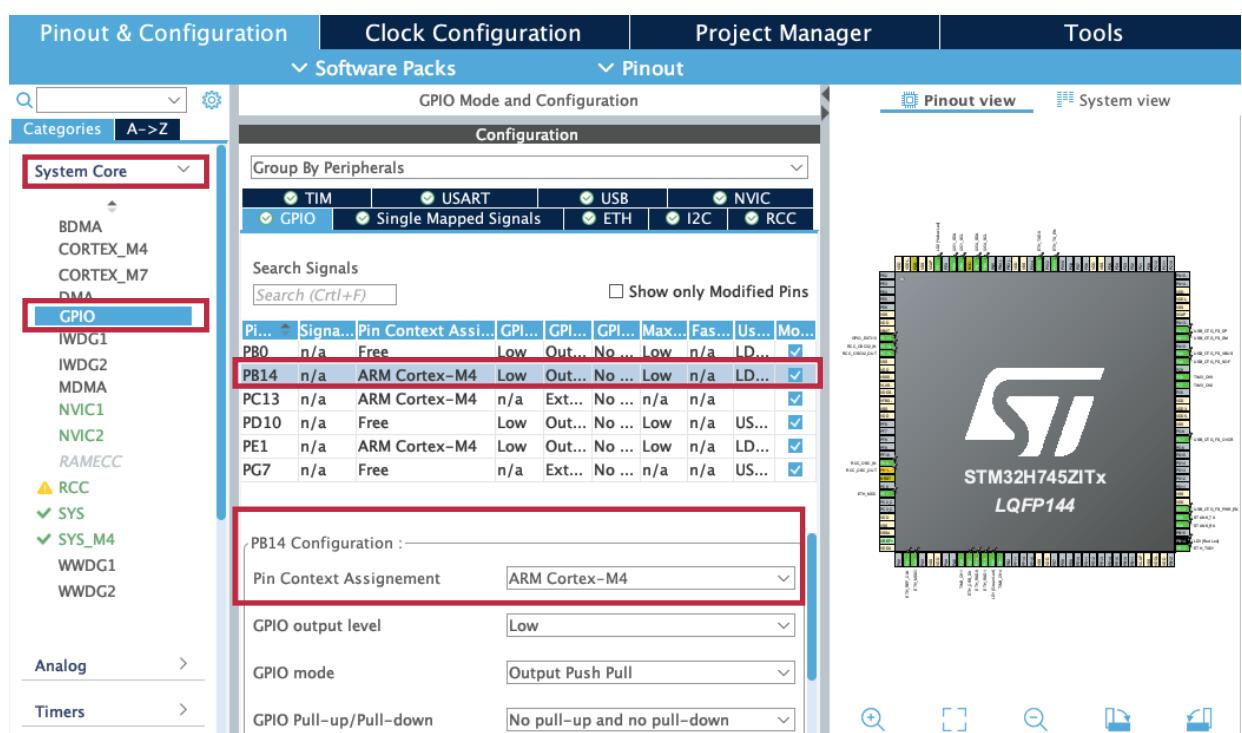


Fig. 5.1.1.1.1: System Core – GPIO – PB14 Configuration

Con la stessa procedura utilizzata per attivare il led LD3, è stato attivato anche il led LD2 (yellow led), ovvero il pin PE1.

Sempre nella sezione *System Core*, è stato attivato sul Cortex – M4 anche il GPIO relativo allo USER BUTTON, a cui corrisponde il pin PC13.

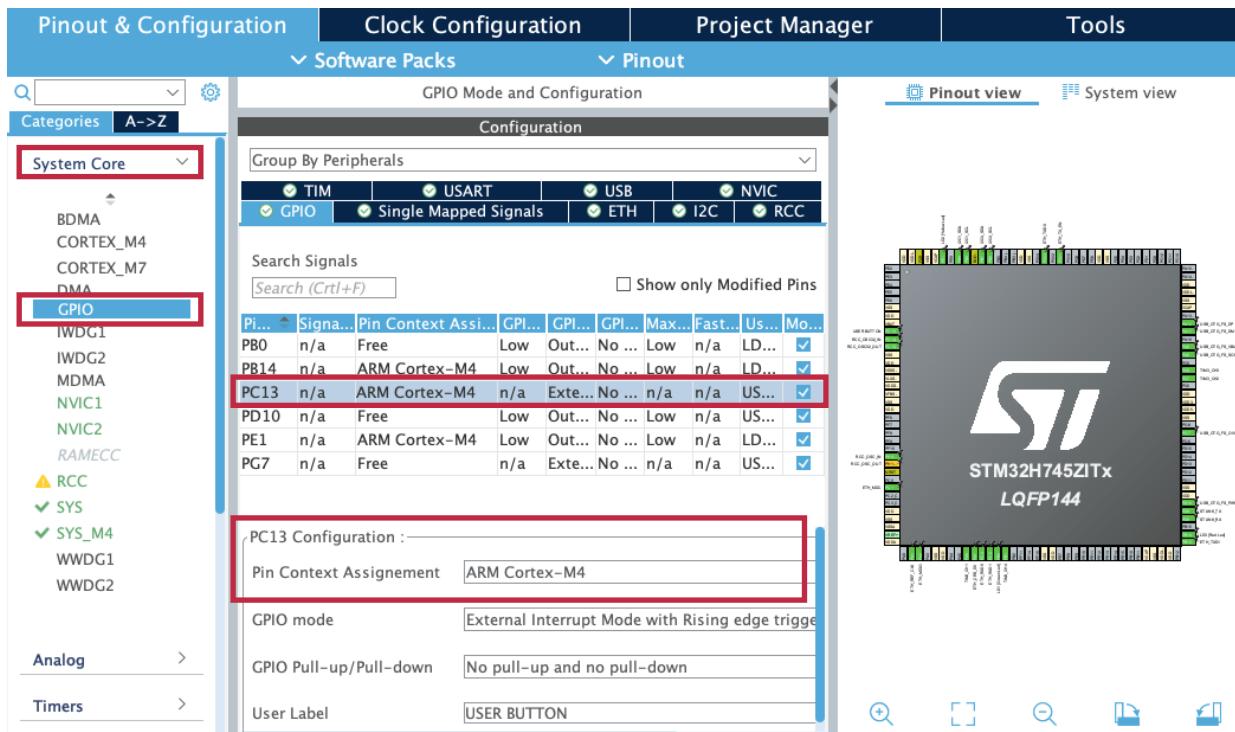


Fig. 5.1.1.1.2: System Core – GPIO – PC13 Configuration

5.1.1.2 Implementazione

- I GPIO vengono inizializzati automaticamente dal software tramite la funzione `MX_GPIO_Init()`.
- L'attivazione del LED LD3 avviene interamente allo USER CODE 3, dentro al while(1) tramite la funzione `HAL_GPIO_WritePin()`, che ha lo scopo di accendere il led del GPIO datogli in ingresso tramite il comando `GPIO_PIN_SET`. Lo spegnimento avviene con la stessa HAL ma utilizzando il comando `GPIO_PIN_RESET`.

```

783 //accendo il led quando entro nell'if
784 HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_SET);
841 HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);

```

Fig. 5.1.1.2.1: Attivazione e spegnimento red led

- L'attivazione del LED LD2 e dello USER BUTTON è stata invece definita dentro lo USER CODE 4. In questo caso è stata utilizzata una variabile intera `flagCallback` (dichiarata nello USER CODE PV), che ha lo scopo di indicare se il pulsante è stato premuto e assume valori pari a 0 ed 1 (funzione di variabile booleana).

```

1218 //funzione utilizzata per controllare se il pulsante blu(User Botton) è stato premuto
1219 void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin) {
1220     if(GPIO_Pin == GPIO_PIN_13)
1221         HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //se il pulsante è stato premuto si accende il LED 2
1222
1223     if (flagCallback == 0)
1224         flagCallback = 1;
1225     else
1226         flagCallback = 0;
1227 }

```

Fig. 5.1.1.2.2: Attivazione User Button

5.1.2 IMU

Come precedentemente descritto, per la connessione dell'IMU alla scheda STM è stato utilizzato il protocollo I2C. In questo caso è stato utilizzato il canale I2C1 (si rimanda alla sezione [5.1.2.1](#)). L'address identificativo del dispositivo è **MPU6050_ADDR**. Per il funzionamento dell'IMU sono state importanti le librerie opportune (si rimanda alla sezione [5.1.2.2](#)).

5.1.2.1 .ioc

Nel file **.ioc**, ed in particolare nella sezione *Connectivity* è stato attivato sul Cortex – M4 il canale I2C1. È necessario selezionare nuovamente la voce I2C nel settore MODE. In questo caso è possibile visualizzare i pinout disponibili, selezionati automaticamente dalla scheda, nel settore CONFIGURATION : PB8 (SCL) e PB9 (SDA)(facilmente individuati nella Fig. 7.1 della [sezione 7](#)).

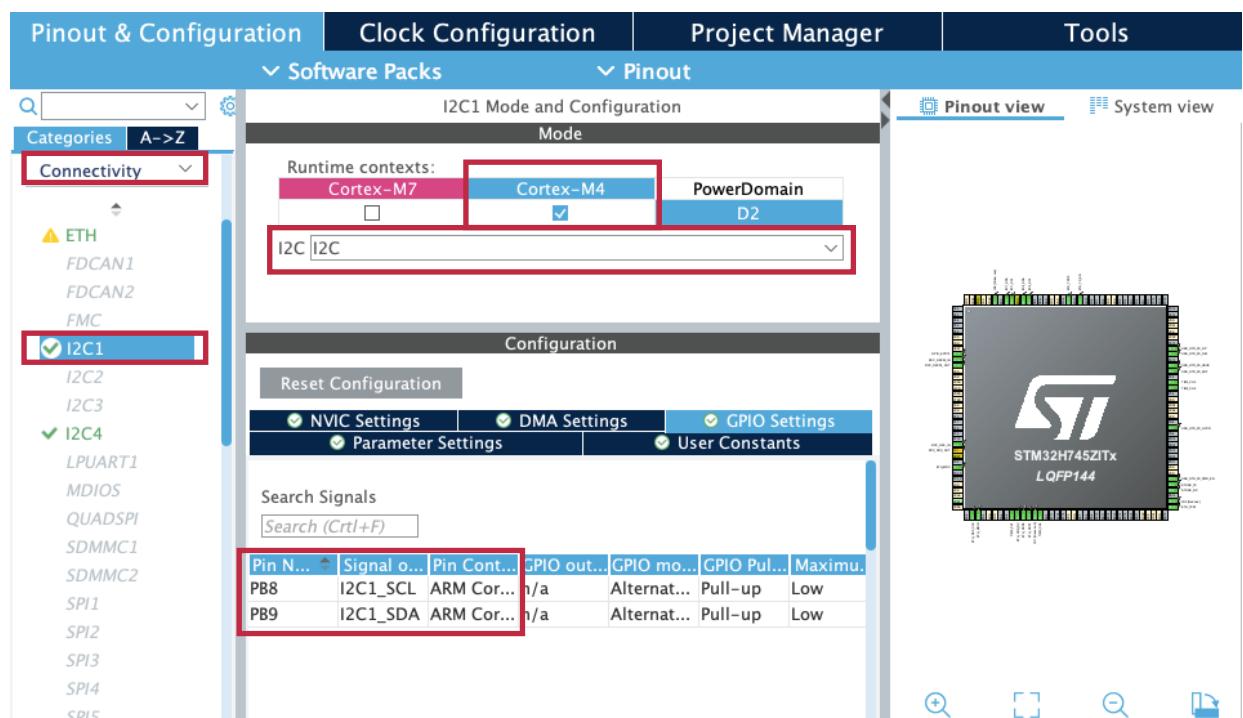


Fig. 5.1.2.1.1 : I2C1 Mode and Configuration – GPIO Settings

In caso di necessità, nella sezione grafica, *Pinout view*, è possibile modificare il canale di uscita I2C a cui sono stati associati i pin, selezionandolo direttamente tra le caratteristiche del GPIO.

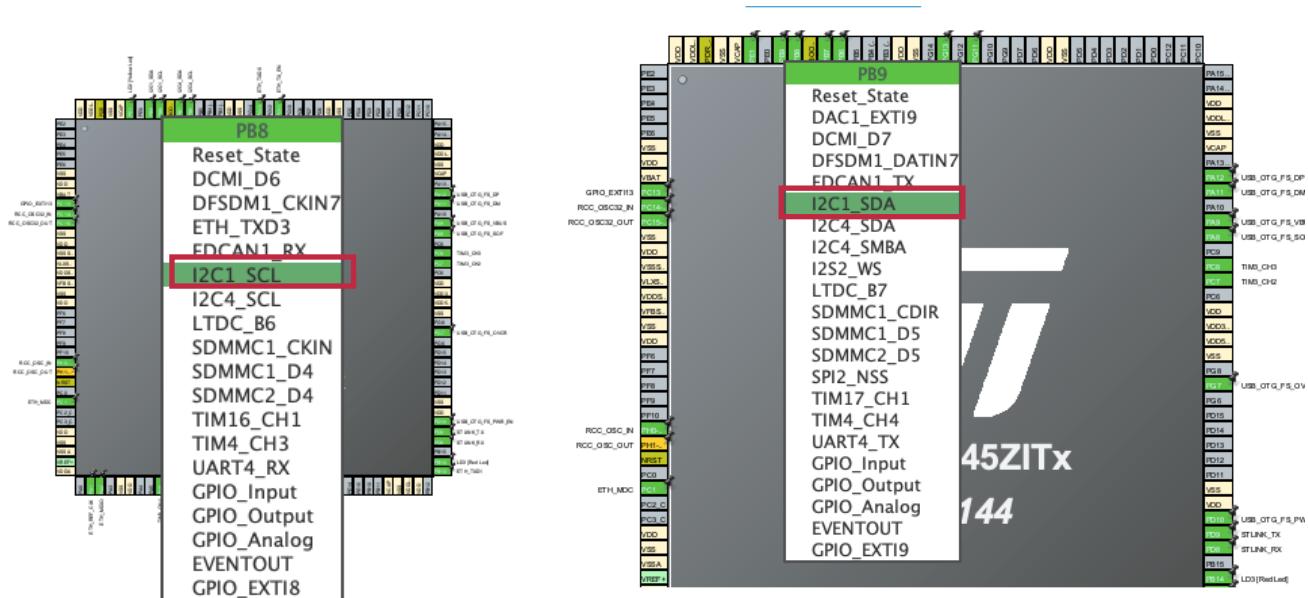


Fig. 5.1.2.1.2 : Pinout view

Nella sezione *Configuration*, è stata modificata la velocità di comunicazione, selezionando la modalità FAST MODE, a cui corrisponde una frequenza di 400KHz.

Configuration

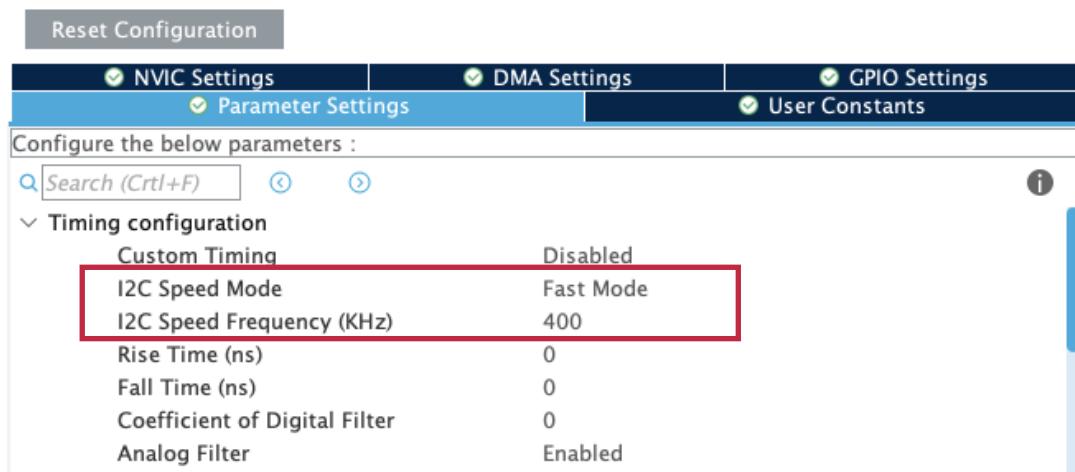


Fig. 5.1.2.1.3 : I2C1 Configuration - Parameter Settings

5.1.2.2 Implementazione

- Nelle cartelle **Src** e **Inc** è possibile visualizzare le librerie importate per il funzionamento dell'IMU: rispettivamente **MPU6050.c** e **MPU6050.h**.
- Nello USER CODE 2 viene inizializzata l'IMU, richiamando la funzione delle librerie, che prende come argomenti:
 - &hi2c1 := canale I2C utilizzato
 - 3, 0, 5 := valori specificati nella funzione **MPU6050_Init()** della libreria

713 //inizializzazione IMU
714 MPU6050_Init(&hi2c1,3,0,5);|

Fig. 5.1.2.2.1 : Inizializzazione IMU

- Nello USER CODE 3, interno al while(1), avviene la lettura dei dati raw (grezzi) dell'IMU (accelerometro e giroscopio) che poi verranno utilizzati e inseriti dapprima nella struttura **MPU6050** tramite la funzione **MPU6050_Parsing()** e successivamente nella struttura **imu_raw**:

```

757      //lettura dati IMU
758      MPU6050_Read_All(&hi2c1, &MPU6050);
759      MPU6050_Parsing(&MPU6050);
760
761      //scrittura dei dati IMU nella struttura imu_raw
762      imu_raw.accRoll = MPU6050.Accel_X_RAW;
763      imu_raw.accPitch = MPU6050.Accel_Y_RAW;
764      imu_raw.accYaw = MPU6050.Accel_Z_RAW;
765      imu_raw.gyrRoll = MPU6050.Gyro_X_RAW;
766      imu_raw.gyrPitch = MPU6050.Gyro_Y_RAW;
767      imu_raw.gyrYaw = MPU6050.Gyro_Z_RAW;
768
769      //modifiche ai valori dell'IMU in base alla sensibilità
770      imu_read(&imu_raw,&imu_sens,&imu_temp);|

```

Fig. 5.1.2.2.2 : Lettura dati IMU

La funzione **imu_read()** viene definita nello USER CODE 0 :

```

570 //lettura dati dall'IMU
571 void imu_read(IMU_raw* imu_raw, IMU_sens* imu_sens, IMU_temp* imu_temp){
572
573     short data_acc[3] = {imu_raw->accRoll,imu_raw->accPitch,imu_raw->accYaw};
574     //mpu_get_accel_reg(data_acc, NULL);
575     imu_raw->accRoll = data_acc[0];
576     imu_raw->accPitch = data_acc[1];
577     imu_raw->accYaw = data_acc[2];
578
579     short data_gyr[3] = {imu_raw->gyrRoll, imu_raw->gyrPitch, imu_raw->gyrYaw};
580     //mpu_get_gyro_reg(data_gyr, NULL);
581     imu_raw->gyrRoll = data_gyr[0];
582     imu_raw->gyrPitch = data_gyr[1];
583     imu_raw->gyrYaw = data_gyr[2];
584
585     imu_temp->accRoll=imu_raw->accRoll / imu_sens->acc_sens;
586     imu_temp->gyrRoll = imu_raw->gyrRoll / imu_sens->gyr_sens * 0.01745329252;
587     imu_temp->accPitch = imu_raw->accPitch / imu_sens->acc_sens;
588     imu_temp->gyrPitch = imu_raw->gyrPitch / imu_sens->gyr_sens * 0.01745329252;
589     imu_temp->accYaw = imu_raw->accYaw / imu_sens->acc_sens;
590     imu_temp->gyrYaw = imu_raw->gyrYaw / imu_sens->gyr_sens * 0.01745329252;
591 }

```

Fig. 5.1.2.2.3: Funzione imu_read()

Tale funzione prende come argomento i puntatori alle strutture definite nello USER CODE PV:

- IMU_raw: contenente i dati grezzi dell'imu

```
172 ⊕ typedef struct {
173     float accRoll;
174     float accPitch;
175     float accYaw;
176     float gyrRoll;
177     float gyrPitch;
178     float gyrYaw;
179 } IMU_raw;
```

Fig. 5.1.2.2.4: struttura IMU_raw

- IMU_sens: contiene i dati del sensore imu

```
167 ⊕ typedef struct {
168     uint16_t acc_sens;
169     float gyr_sens;
170 } IMU_sens;
```

Fig. 5.1.2.2.5: struttura IMU_sens

- IMU_temp: contiene i dati rielaborati

```
181 ⊕ typedef struct {
182     float accRoll;
183     float accPitch;
184     float accYaw;
185     float gyrRoll;
186     float gyrPitch;
187     float gyrYaw;
188 } IMU_temp;
```

Fig. 5.1.2.2.6: struttura IMU_temp

5.1.3 Magnetometro

Essendo il sensore saldato nella scheda GY – 86, come precedentemente citato, questo sensore comunica con la scheda STM32H745 con lo stesso canale I2C utilizzato per la connessione dell'IMU, ovvero il canale I2C1. L'address identificativo del dispositivo è **HMC5883L_ADDRESS**.

5.1.3.1 .ioc

Per la motivazione sopra descritta, si rimanda al paragrafo [5.1.1.1](#).

5.1.3.2 Implementazione

- Per il funzionamento del magnetometro, è necessario settare i registri dell'IMU per abilitare il bypass, fondamentale per raggiungere il magnetometro:

```
715 //abilitazione bypass per raggiungere il magnetometro
716 MPU6050_Bypass(&hi2c1);
```

Fig. 5.1.3.2.1 : Funzione di bypass

- Nello USER CODE PV sono stati definiti i registri del magnetometro (si faccia riferimento al datasheet del dispositivo):

```

114 // HMC5883L - ADDRESS
115 #define HMC5883L_ADDRESS (0x1E << 1)
116 // CONTROL REG A
117 #define HMC5883L_Enable_A (0x78)
118 // CONTROL REG B
119 #define HMC5883L_Enable_B (0xA0)
120 // MODE REGISTER
121 #define HMC5883L_MR (0x00)
122 // HMC5883L - MSB / LSB ADDRESSES
123 #define HMC5883L_ADD_DATAX_MSB (0x03)
124 #define HMC5883L_ADD_DATAX_LSB (0x04)
125 #define HMC5883L_ADD_DATAZ_MSB (0x05)
126 #define HMC5883L_ADD_DATAZ_LSB (0x06)
127 #define HMC5883L_ADD_DATAY_MSB (0x07)
128 #define HMC5883L_ADD_DATAY_LSB (0x08)
129 // SUM (MSB + LSB) DEFINE
130 #define HMC5883L_ADD_DATAX_MSB_MULTI (HMC5883L_ADD_DATAX_MSB | 0x80)
131 #define HMC5883L_ADD_DATAY_MSB_MULTI (HMC5883L_ADD_DATAY_MSB | 0x80)
132 #define HMC5883L_ADD_DATAZ_MSB_MULTI (HMC5883L_ADD_DATAZ_MSB | 0x80)

```

Fig. 5.1.3.2.2 : Registri HMC5883L

- Nello USER CODE 2 viene definita l'inizializzazione del magnetometro con il settaggio dei suoi registri:

```

718 //settaggio registri magnetometro
719 HAL_I2C_Mem_Write(&hi2c1, HMC5883L_ADDRESS, 0x00 , 1, &RegSettingA , 1, 100); //modalità di misura
720 HAL_I2C_Mem_Write(&hi2c1, HMC5883L_ADDRESS, 0x01 , 1, &RegSettingB , 1, 100); // gain magnetometro
721 HAL_I2C_Mem_Write(&hi2c1, HMC5883L_ADDRESS, 0x02 , 1, &RegSettingMR , 1, 100); // modalità continua
---
```

Fig. 5.1.3.2.3 : Inizializzazione magnetometro

- Successivamente, nello USER CODE 2, viene inizializzata la struttura **mag_data** contenente i dati del magnetometro tramite la funzione **maginit()**:

```

732 //inizializzazione struttura contenente i dati del magnetometro
733 maginit();

```

Fig. 5.1.3.2.4: Inizializzazione struttura dati del magnetometro

La cui definizione si trova nello USER CODE PV ed ha la seguente forma:

```

190 typedef struct {
191     short raw[3];
192     float x,y,z;
193     float sens;
194     float bias[3];
195     float scale[3];
196     float ABS;
197 } MAG_data;

```

Fig. 5.1.3.2.5: Struttura MAG_data

- Per il corretto funzionamento del magnetometro, la prima volta che viene acceso, è necessario effettuare la calibrazione nel luogo in cui verrà utilizzato il quadricottero (in questo caso è stata inserita nello USER CODE 2) tramite la funzione **magcal()**:

```

743 //calibrazione magnetometro
744 magcal();

```

Fig. 5.1.3.2.6 : Calibrazione magnetometro

- Nello USER CODE 3, interno al while(1), vengono richiamate le funzioni di lettura dei dati raw del magnetometro e del calcolo dei suoi valori:

```

333 //calcolo valori magnetometro
334 void readmag() {
335
336     mag_data.x = ((float)mag_data.raw[0] * mag_data.sens - mag_data.bias[0]) * mag_data.scale[0];
337     mag_data.y = ((float)mag_data.raw[1] * mag_data.sens - mag_data.bias[1]) * mag_data.scale[1];
338     mag_data.z = ((float)mag_data.raw[2] * mag_data.sens - mag_data.bias[2]) * mag_data.scale[2];
339
340 }
341
342 //lettura dati grezzi magnetometro
343 void readmagraw() {
344
345     // RECEIVE X_axis
346     HAL_I2C_Mem_Read(&hi2c1,HMC5883L_ADDRESS,HMC5883L_ADD_DATAx_MSB_MULTI,1,DataX,2,100);
347     mag_data.raw[0] = ((DataX[1]<<8) | DataX[0]);
348     // RECEIVE Y_axis
349     HAL_I2C_Mem_Read(&hi2c1,HMC5883L_ADDRESS,HMC5883L_ADD_DATAy_MSB_MULTI,1,DataY,2,100);
350     mag_data.raw[1] = ((DataY[1]<<8) | DataY[0]);
351     // RECEIVE Z_axis
352     HAL_I2C_Mem_Read(&hi2c1,HMC5883L_ADDRESS,HMC5883L_ADD_DATAz_MSB_MULTI,1,DataZ,2,100);
353     mag_data.raw[2] = ((DataZ[1]<<8) | DataZ[0]);
354
355 }
```

Fig. 5.1.3.2.7 : Lettura e calcolo dati magnetometro

5.1.4 Calcolo angoli Roll, Pitch, Yaw

Per effettuare il calcolo degli angoli Roll, Pitch e Yaw è stato necessario inserire i dati prelevati dall'IMU e dal magnetometro all'interno di una struttura dati AHRS_data della seguente forma:

```

199 typedef struct{
200     float RollRad;          // Corrisponde all'uscita in radianti dell'asse x di AHRS
201     float PitchRad;         // Corrisponde all'uscita in radianti dell'asse y di AHRS
202     floatYawRad;           // Corrisponde all'uscita in radianti dell'asse z di AHRS
203     float RollDeg;          // Corrisponde all'uscita in gradi dell'asse x di AHRS
204     float PitchDeg;         // Corrisponde all'uscita in gradi dell'asse y di AHRS
205     floatYawDeg;           // Corrisponde all'uscita in gradi dell'asse z di AHRS
206     float omegaRollRad;    // Corrisponde all'uscita in radianti dell'asse x di AHRS (velocità angolare)
207     float omegaPitchRad;   // Corrisponde all'uscita in radianti dell'asse y di AHRS (velocità angolare)
208     float omegaYawRad;     // Corrisponde all'uscita in radianti dell'asse z di AHRS (velocità angolare)
209     float omegaRollDeg;    // Corrisponde all'uscita in gradi dell'asse x di AHRS (velocità angolare)
210     float omegaPitchDeg;   // Corrisponde all'uscita in gradi dell'asse y di AHRS (velocità angolare)
211     float omegaYawDeg;     // Corrisponde all'uscita in gradi dell'asse z di AHRS (velocità angolare)
212 }AHRS_data;
```

Fig. 5.1.4.1: Struttura AHRS_data

Il riempimento di tale struttura avviene nello USER CODE 3 richiamando la funzione getYPR(), definita nello USER CODE 0.

```

743 //riempimento struttura ahhs
744 getYPR(&mag_data ,&imu_temp,&ahrs);
```

Fig. 5.1.4.2: Riempimento struttura AHRS_data

La funzione getYPR ha lo scopo di riempire la struttura AHRS_data e dunque di calcolare le velocità angolari corrispondenti agli angoli, necessarie poi per il calcolo degli angoli Roll, Pitch e Yaw.

```

596 //funzione che richiama il filtro di Magdwick e la funzione del calcolo angoli
597 void getYPR(MAG_data* mag_data ,IMU_temp* imu_temp, AHRS_data* ahrs_data){
598
599     madgwickFilterUpdate(imu_temp->gyrRoll, imu_temp->gyrPitch, imu_temp->gyrYaw, imu_temp->accRoll,
600                         imu_temp->accPitch, imu_temp->accYaw, mag_data->x, mag_data->y, mag_data->z);
601     getAHRS(&ahrs_data->PitchRad, &ahrs_data->YawRad, &ahrs_data->RollRad);
602     ahrs_data->RollDeg = ahrs_data->RollRad * (180.0/PI);
603     ahrs_data->PitchDeg = ahrs_data->PitchRad * (180.0/PI);
604     ahrs_data->YawDeg = ahrs_data->YawRad * (180.0/PI);
605
606     /* Calcolo delle velocità angolari degli angoli */
607     ahrs_data->omegaRollRad = imu_temp->gyrRoll ;/* (PI/180.0);
608     ahrs_data->omegaPitchRad = imu_temp->gyrPitch ;/* (PI/180.0);
609     ahrs_data->omegaYawRad = imu_temp->gyrYaw ;/* (PI/180.0);
610
611     ahrs_data->omegaRollDeg = ahrs_data->omegaRollRad * (180.0/PI);
612     ahrs_data->omegaPitchDeg = ahrs_data->omegaPitchRad * (180.0/PI);
613     ahrs_data->omegaYawDeg = ahrs_data->omegaYawRad * (180.0/PI);
614
615     OmegaYawRad=ahrs_data->omegaYawRad;
616     OmegaYawDeg=ahrs_data->omegaYawDeg;
617 }

```

Fig. 5.1.4.3: Funzione getYPR

Il calcolo dei valori è stato eseguito utilizzando l'algoritmo : **FILTRO DI MADGWICK** (si rimanda alle funzioni **madgwickFilterUpdate()** e **MadgwickAHRSupdateIMU()**, presenti nello USER CODE 0), che utilizza i quaternioni (utilizzati principalmente nella rappresentazione di rotazioni e direzioni nello spazio tridimensionale). Tramite i quaternioni è possibile calcolare i valori desiderati.

```

553 //calcolo di yaw, pitch e roll attraverso i quaternioni precedentemente calcolati
554 void getAHRS(float* pitch, float* yaw, float* roll){
555     *yaw = atan2(2.0f * (q1 * q2 + q0 * q3), q0 * q0 + q1 * q1 - q2 * q2 - q3 * q3);
556     *pitch = -asin(2.0f * (q1 * q3 - q0 * q2));
557     *roll = atan2(2.0f * (q0 * q1 + q2 * q3), q0 * q0 - q1 * q1 - q2 * q2 + q3 * q3);
558
559 }

```

Fig. 5.1.4.4 : Calcolo degli angoli

5.1.5 OLED Display

Come precedentemente descritto, per la connessione dell’OLED Display alla scheda STM è stato utilizzato il protocollo I2C. In questo caso è stato utilizzato il canale I2C4 (si rimanda alla sezione [5.1.5.1](#)). L’address identificativo del dispositivo è **SSD1306_I2C_ADDR**. Per il funzionamento del display sono state importanti le librerie opportune (si rimanda alla sezione [5.1.5.2](#)).

5.1.5.1 .ioc

Nel file **.ioc**, ed in particolare nella sezione *Connectivity* è stato attivato sul Cortex – M4 il canale I2C4. È necessario selezionare nuovamente la voce I2C nel settore MODE. In questo caso è possibile visualizzare i pinout disponibili, selezionati automaticamente dalla scheda, nel settore CONFIGURATION : PB6 (SCL) e PB7 (SDA) (facilmente individuati nella Fig.7.1 della [sezione 7](#)).

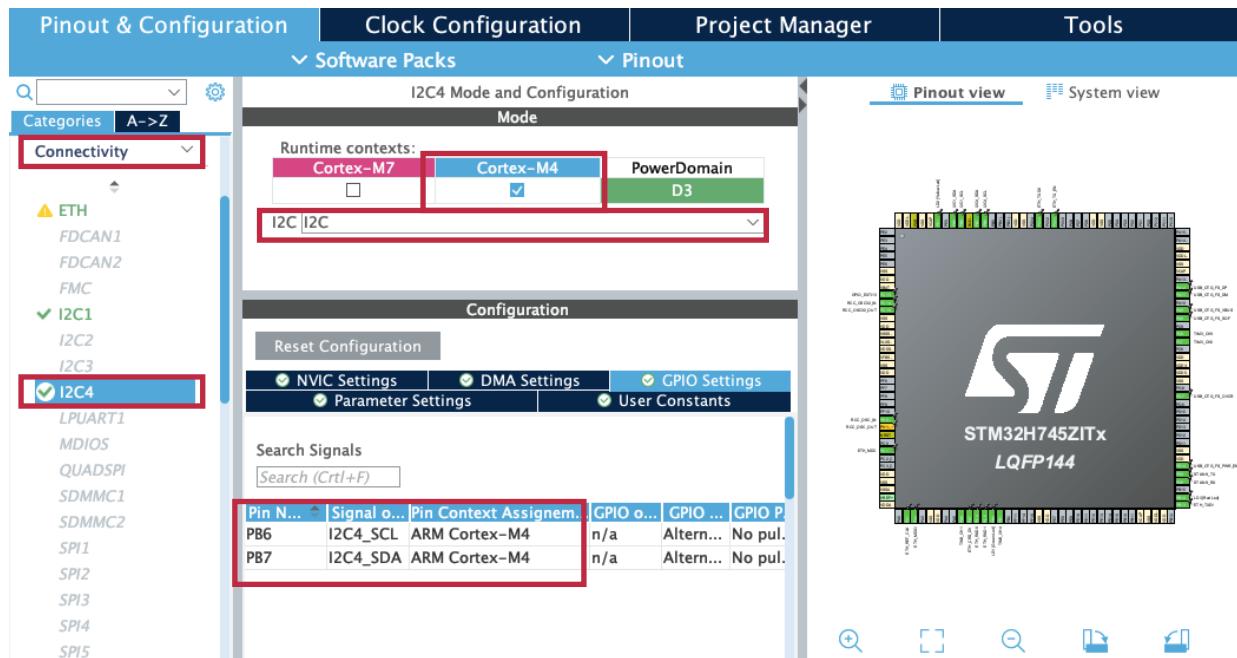


Fig. 5.1.5.1.1 : I2C4 Mode and Configuration – GPIO Settings

Nella sezione *Configuration*, è stata impostata la velocità di comunicazione, selezionando la modalità STANDARD MODE, a cui corrisponde una frequenza di 100KHz.

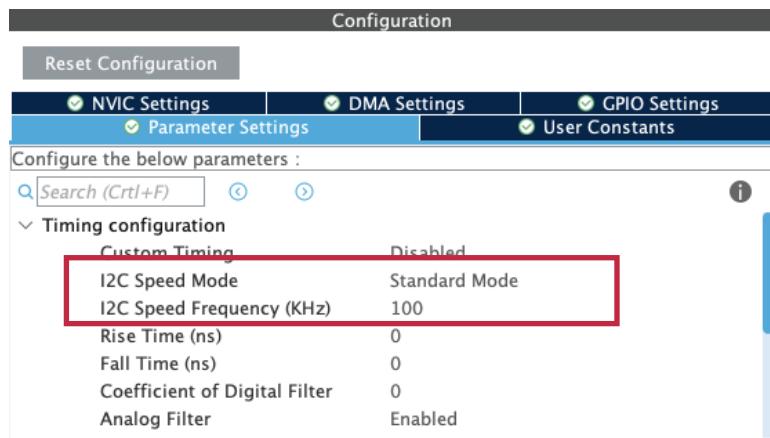


Fig. 5.1.5.1.2: I2C4 Configuration - Parameter Settings

5.1.5.2 Implementazione

- Nelle cartelle **Src** e **Inc** è possibile visualizzare le librerie importate per il funzionamento dello schermo: rispettivamente **ssd1306.c**, **ssd1306_fonts.c**, **ssd1306_tests.c** e **ssd1306.h** , **ssd1306_fonts.h** , **ssd1306_tests.h** , **ssd1306_conf_template.h** .
- Lo schermo stampa i valori degli angoli di Roll, Pitch e Yaw e viene inizializzato nello USER CODE 2, richiamando la funzione **ssd1306_Init()** della libreria **ssd1306.c**.
- La funzione che si occupa della stampa a schermo dei valori degli angoli in gradi è la funzione **Schermo()** definita nello USER CODE 0:

```

352 //stampa a schermo il roll, pitch e yaw
353 void Schermo()
354 {
355
356     uint8_t str[80];
357     uint8_t str1[80];
358     uint8_t str2[80];
359     ssd1306_Fill(Black);
360     ssd1306_SetCursor(2,0);
361     ssd1306_WriteString("Roll:", Font_6x8, White);
362     ssd1306_SetCursor(2,20);
363     ssd1306_WriteString("Pitch:", Font_6x8, White);
364     ssd1306_SetCursor(2,40);
365     ssd1306_WriteString("Yaw:", Font_6x8, White);
366     ssd1306_SetCursor(40, 0);
367     sprintf(str, 80, "%6.3f      ", ahrs.RollDeg);
368     ssd1306_WriteString(str, Font_6x8, White);
369     ssd1306_SetCursor(40, 20);
370     sprintf(str1, 80, "%6.3f      ", ahrs.PitchDeg);
371     ssd1306_WriteString(str1, Font_6x8, White);
372     ssd1306_SetCursor(40, 40);
373     sprintf(str2, 80, "%6.3f      ", ahrs.YawDeg);
374     ssd1306_WriteString(str2, Font_6x8, White);
375     ssd1306_UpdateScreen();
376
377 }

```

Fig. 5.1.5.2.1: Funzione stampa a schermo

La funzione `ssd1306_SetCursor(x,y)` setta la posizione del cursore sullo schermo in base alle coordinate x e y scelte.

La funzione `ssd1306_WriteString(str, font, color)` scrive la stringa desiderata sullo schermo in base al font e al colore scelto.

Per informazioni più dettagliate riguardo alla struttura delle funzioni sopra definite, si rimanda alla libreria `ssd1306.c`.

Tale funzione viene richiamata nello USER CODE 3 (interno al `while(1)`) :

```

746 //scrittura a schermo dei valori di pitch, roll e yaw
747 Schermo();

```

Fig. 5.1.5.2.2: Scrittura e stampa a schermo

5.2 PID

Per realizzare un controllore PID è necessario discretizzare la formula integro-differenziale che lo definisce (si rimanda alla sezione [2.4](#)). In questa sezione viene descritta come avviene la conversione delle velocità in un segnale PWM.

Dapprima è importante calcolare l'errore attraverso la differenza tra l'uscita desiderata e quella attuale:

```
float error = setPoint - input
```

dunque, è possibile calcolare il termine derivativo e integrale:

- **Derivativo** : ottenuto approssimando la derivata dell'errore con il metodo delle differenze finite ($\frac{d(x)}{dt} \cong \frac{x_f - x_i}{T}$, T := tempo di campionamento) :


```
float dInput = (error - conf->lastError) / conf->dt
```
- **Integrale** : ottenuto approssimando l'integrale dell'errore con la formula composita della regola del rettangolo ($\int_a^b f(x)dx \cong h \sum_{i=1}^n f(x_i)$, con $h = \frac{b-a}{M}$, M:= numero di sottointervalli in cui è applicata la regola). In questo caso, h coincide con il periodo di campionamento **dt**. Non conoscendo l'ampiezza dell'intervallo in partenza, è stato sommato progressivamente il prodotto eT , moltiplicato per il coefficiente K_I .


```
conf->ITerm += (error*conf->dt)*conf->ki
```

Dunque è necessario moltiplicare ogni termine per il corrispettivo coefficiente :

```
output = (conf->kp*error)+(conf->ITerm)+(conf->kd*dInput)
```

5.2.1 Inizializzazione

- Nelle cartelle **Src** e **Inc** è possibile visualizzare le librerie importate per il funzionamento dei PID: rispettivamente **PID.c** e **PID.h**.
- Nello USER CODE 2 vengono inizializzati i PID tramite la funzione **PID_Init()** definita nel file **PID.h** ed implementata in **PID.c** nel seguente modo:

```
26 void PID_Init(PID_config* conf, float kp, float kd, float ki, float dt, float outMin, float outMax)
27 {
28     conf->kp = kp;
29     conf->kd = kd;
30     conf->ki = ki;
31     conf->dt = dt;
32     conf->ITerm = 0;
33     conf->lastError = 0;
34     conf->outMax = outMax;
35     conf->outMin = outMin;
36 }
```

Fig. 5.2.1.1: Inizializzazione PID

In particolare, viene riempita la struttura **PID_config** definita nel file **PID.h**:

```
17 typedef struct {
18     float kp, kd, ki; //coefficients of proportional, derivative, integral controller
19     float dt; //step for the discrete control
20     float lastError; //error at time t-1
21     float ITerm; //integral term
22     float outMax;
23     float outMin;
24 } PID_config;
```

Fig. 5.2.1.2: Struttura PID_config

Nello USER CODE PV sono state create tre strutture della tipologia sopra descritta:

Roll_PID, **Pitch_PID** e **Yaw_PID**.

Inoltre, è stato anche inizializzato **dt** = 0.05.

Sono stati dunque scelti i coefficienti **kp**, **kd** e **ki** più appropriati, da correggere in seguito alle prove fisiche effettuate sul drone (si faccia riferimento alla sezione [8.2](#)).

```

727 //inizializzazione PID
728 PID_Init(&Pitch_PID, 0.05, 0.05, 0.5, dt, 0, 1);
729 PID_Init(&Roll_PID, 0.05, 0.05, 0.5, dt, 0, 1);
730 PID_Init(&Yaw_PID, 0.05, 0.05, 0.5, dt, 0, 1);
731

```

Fig. 5.2.1.3: Inizializzazione dei PID

5.2.2 Esecuzione Real – Time

- Nello USER CODE 3, terminata la fase di armamento dei motori, viene eseguito il calcolo delle velocità in base ai valori rilevati dai PID e successivamente viene calcolato il valore del duty cycle da inviare ai motori:

```

808     float virtualInputs[3];
809
810     //calcolo i valori dai PID
811     virtualInputs[0] = PID_Compute(ahrs.PitchDeg, 0, &Pitch_PID);
812     virtualInputs[1] = PID_Compute(ahrs.RollDeg, 0, &Roll_PID);
813     virtualInputs[2] = PID_Compute(ahrs.YawDeg, 0, &Yaw_PID);
814
815     float* Speeds;
816
817     //calcolo le velocità
818     Speeds = SpeedCompute(virtualInputs, B_4,L,D);
819
820     //calcolo il duty cycle dalle velocità
821     avgMotor1 = map(*(Speeds+0), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
822     avgMotor2 = map(*(Speeds+1), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
823     avgMotor3 = map(*(Speeds+2), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
824     avgMotor4 = map(*(Speeds+3), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
825

```

Fig. 5.2.2.1: Calcolo duty cycle da inviare ai motori

- Le funzioni utilizzate per il calcolo dei valori sono definita nella libreria **PID.h** ed implementata in **PID.c**. In particolare gli ingressi virtuali da inviare ai PID vengono inseriti nella struttura **virtualInputs**, utilizzando la funzione **PID_Compute()**, nel seguente modo:

```

44④ float PID_Compute(float input, float setPoint, PID_config* conf)
45 {
46     /*Compute all the working error variables*/
47     float error = setPoint - input;
48
49     /*calculates the integral of error (adding the new error)
50      known the previous value of the integral of the error, we add the error in the new step,
51      thus we obtain the area under the error function step by step*/
52     conf->ITerm += (error - conf->lastError) / conf->dt; //derivative of the exit value (input). ((e(t)-e(t-1))/dt
53
54     if((conf->ITerm) > conf->outMax) conf->ITerm = conf->outMax;
55     else if((conf->ITerm) < conf->outMin) conf->ITerm = conf->outMin;
56     float dInput = (error - conf->lastError) / conf->dt; //derivative of the exit value (input). ((e(t)-e(t-1))/dt
57
58     /*Compute PID Output*/
59     float output = (conf->kp * error) + (conf->ITerm) + (conf->kd * dInput); //sum of the outputs of the 3 PIDs
60
61     if(output > conf->outMax) output = conf->outMax;
62     else if(output < conf->outMin) output = conf->outMin;
63
64     /*Remember some variables for next time*/
65     conf->lastError = error;
66     return output;
67 }

```

Fig. 5.2.2.2: Funzione PID_Compute

Gli ingressi virtuali non possono essere inviati ai motori direttamente, dunque è necessario calcolare le velocità tramite l'utilizzo della funzione **SpeedCompute()**.

Tale funzione risulta essere definita come la traduzione software della matrice d'assetto introdotta precedentemente (si faccia riferimento alla sezione [2.4](#)).

```

69 float* SpeedCompute (float virtualInputs [], float b, float l, float d)
70 {
71
72     static float Speeds_quad[4];
73     static float Speeds[4];
74
75     Speeds_quad[0] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
76     Speeds_quad[1] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
77     Speeds_quad[2] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
78     Speeds_quad[3] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
79
80     /*every speed needs to be mapped in a positive range [0, max speed^2], because the mathematical model
81      *and the control system consider the possibility to invert the rotation, resulting in negative speeds.
82      *This is out the actual possibilities of our physical system because we cannot invert rotation nor
83      *calculate a square root of a negative number.*/
84
85     const float abs_speedQuad_MAX = 1/(4*b) + 1/(2*l*b) + 1/(4*d); //maximum reachable squared speed
86
87     float speedQuad_min = -(1/(2*l*b)) - (1/(4*d)); //lower bound for squared speed 0.
88     float speedQuad_Max = 1/(4*b); //upper bound for squared speed 0.
89     Speeds[0] = sqrt(map(Speeds_quad[0], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));
90
91     speedQuad_min = -(1/(2*l*b)); //lower bound for squared speed 1.
92     speedQuad_Max = 1/(4*b) + 1/(4*d); //upper bound for squared speed 1.
93     Speeds[1] = sqrt(map(Speeds_quad[1], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));
94
95     speedQuad_min = -(1/(4*d)); //lower bound for squared speed 2.
96     speedQuad_Max = 1/(4*b) + 1/(2*l*b); //upper bound for squared speed 2.
97     Speeds[2] = sqrt(map(Speeds_quad[2], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));
98
99     //Speeds_quad[3] can't be negative so we just need to make the square root
100    Speeds[3] = sqrt(Speeds_quad[3]);
101
102    return Speeds;
103
104 }
```

Fig. 5.2.2.3: Funzione SpeedCompute

Infine, il calcolo delle velocità ottenute viene mappato in un range compreso tra i valori minimi e massimi di duty cycle (range compreso tra 1200 e 2000), tramite l'ausilio della funzione **map()** , definita nella libreria **map.h** ed implementata in **map.c** :

```

14 inline float map(float val, float from_src, float to_src, float from_dst, float to_dst)
15 {
16     return (((to_dst-from_dst)/(to_src-from_src))*(val-from_src)) + from_dst;
17 }
18
```

Fig. 5.2.2.4: Funzione map

Dove:

- **val**: valore da convertire (velocità)
- **from_src**: minimo valore di range del primo intervallo (0)
- **to_src**: massimo valore di range del primo intervallo (massimo valore di velocità dei motori)
- **from_dst**: minimo valore di range dell'intervallo finale (1200)
- **to_dst**: massimo valore di range dell'intervallo finale (2000)

6. PWM

6.1 Cos'è e come funziona

La modulazione a larghezza di impulso (Pulse With Modulation) è una tecnica che permette di generare segnali di impulso con diversi duty cycle (intervalli di tempo in cui l'impulso è alto). Nella scheda STM32 i segnali PWM vengono generati mediante l'utilizzo di timers, i quali sono in grado di generare segnali a impulsi con diversi duty cycle. Il duty cycle è così calcolabile:

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

Dove 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 CCR_x , 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:

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

Dove $MaxCNT$:= registro di confronto dell'uscita (registro di ciclo)

Poiché i valori CCR_x memorizzati dipendono dalla frequenza di clock reale del timer ($TIMx_CLK$), dunque la frequenza del canale può essere definita nel seguente modo:

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

È importante evidenziare che i timer (TIM) della STM, utilizzati per la generazione del segnale PWM, sono costituiti da un massimo di quattro canali di uscita indipendenti, dunque sono in grado di generare contemporaneamente 4 PWM sincronizzati su di un singolo timer.

6.2 Utilizzo

Per lo sviluppo del funzionamento del drone preso in esame, è stato attivato un unico timer per la generazione di quattro PWM per l'attivazione dei quattro [motori brushless](#), sia per la fase di armamento che per la fase di esecuzione Real – Time.

Durante la fase di armamento viene inviato ai motori un segnale in PWM a cui corrisponde un duty cycle del 4.75 %, per un periodo di 2s.

Terminata la fase di armamento, durante la fase di esecuzione Real – Time il valore del duty cycle rimane compreso tra un valore minimo del 6% ed un massimo del 10%, per un corretto utilizzo dei motori.



Fig. 6.2.1 : Minimo duty cycle 6%



Fig. 6.2.2 : Massimo duty cycle 10%

Essendo la frequenza di funzionamento dei motori ($TIM_{Update\ Freq.}\ (Hz)$) pari a 50Hz, è stato possibile effettuare il calcolo del prescaler utilizzando la seguente formula:

$$TIM_{Update\ Freq.}\ (Hz) = \frac{Clock}{(Prescaler - 1) \times (Period - 1)}$$

Dove $Clock$:= frequenza del timer (240MHz).

$Period$:= periodo in cui il segnale è alto (20ms).

6.2.1 .ioc

Nel file **.ioc**, ed in particolare nella sezione *Timers* è stato attivato sul Cortex – M4 il TIM3 (Timer general purpose). Per ciascuno dei quattro canali resi disponibili dal Timer, è necessario selezionare la voce “**PWM Generation CHx**”, per generare i segnali PWM da inviare ai motori.

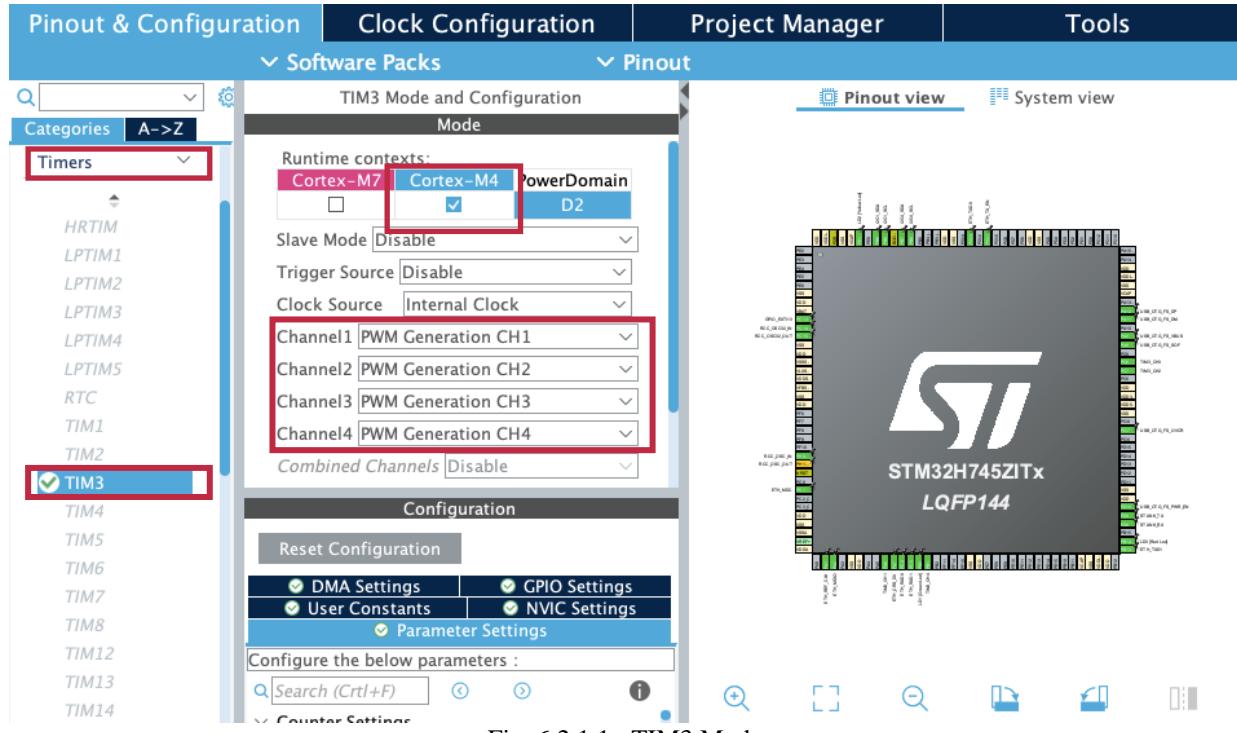


Fig. 6.2.1.1 : TIM3 Mode

Nella sezione *GPIO Settings*, è possibile verificare i pinout della scheda assegnati a ciascun segnale PWM : nel caso preso in esame, al pin PA6 è stato assegnato il canale 1, al PC7 il canale 2, al PC8 il canale 3 ed infine al PB1 il canale 4 (facilmente individuati nella Fig.7.1 della [sezione 7](#)).

Configuration													
Reset Configuration													
<input checked="" type="checkbox"/> NVIC Settings		<input checked="" type="checkbox"/> DMA Settings		<input checked="" type="checkbox"/> GPIO Settings									
<input checked="" type="checkbox"/> Parameter Settings				<input checked="" type="checkbox"/> User Constants									
Search Signals													
<input type="text"/> Search (Ctrl+F) <input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/> Show or													
Pin N...	Signal o...	Pin Context Assigned	GPIO o...	GPIO m...	GPIO Pu...	Maxim...	Fast M...						
PA6	TIM3_CH1	ARM Cortex-M4	n/a	Altera...	No pull...	Low	n/a						
PB1	TIM3_CH4	ARM Cortex-M4	n/a	Altera...	No pull...	Low	n/a						
PC7	TIM3_CH2	ARM Cortex-M4	n/a	Altera...	No pull...	Low	n/a						
PC8	TIM3_CH3	ARM Cortex-M4	n/a	Altera...	No pull...	Low	n/a						

Fig. 6.2.1.2 : TIM3 Configuration - GPIO Settings

Nella sezione *Parameter Settings* sono stati inseriti i valori di Prescaler e Counter Period, come precedentemente calcolati. Inoltre, in questo settore è fondamentale verificare che, per ciascun canale, la voce “Output compare preload” sia ENABLE.

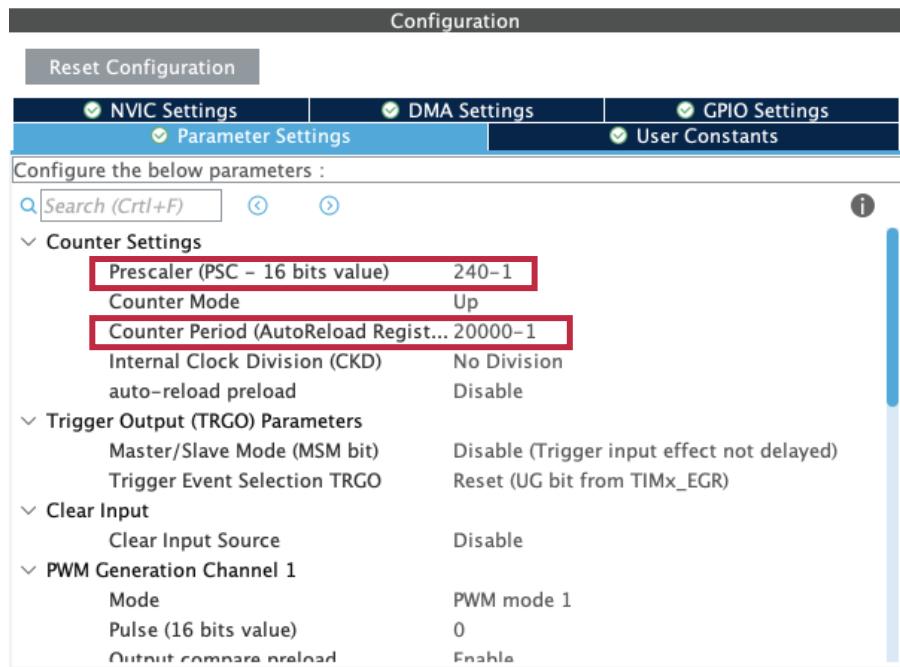


Fig. 6.2.1.3 : TIM3 Configuration - Parameter Settings

6.2.2 Implementazione

- Nello USER CODE PV sono stati definiti i parametri principali riguardanti il funzionamento dei motori utilizzati la taratura dei PID (ad esempio valore min e max di duty, coefficienti di drag e di spinta):

```

90 //definizioni variabili hardware
91 #define MOTOR_MIN_UP 1200 //Min value of duty cycle - 6% Duty (min speed)
92 #define MOTOR_MAX_UP 2000 //Max value of duty cycle - 10% Duty (Max speed)
93 #define MOTOR_MAX_SPEED_3 1220.6926 //sqrt(1/(4*B_3)+1/(2*L*B_3)+1/(4*D))
94 #define MOTOR_MAX_SPEED_4 1220.2435 //sqrt(1/(4*B_4)+1/(2*L*B_3)+1/(4*D))
95 #define B_4 0.000001163 //thrust coefficient 4-cell battery
96 #define B_3 0.000001162 //thrust coefficient 3-cell battery
97 #define L 0.3375 //distance between motor and drone center
98 #define D 0.08 //drag coefficient

```

Fig. 6.2.2.1: Parametri caratteristici motori

- Nello USER CODE 2 vengono inizializzati i PWM:

```

683 //inizializzazione PWM
684 HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
685 HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
686 HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);
687 HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4);

```

Fig. 6.2.2.1 : Inizializzazione PWM

- Nello USER CODE 3, ed in particolare nel momento in cui viene premuto lo USER BUTTON (variabile **flagCallback** = 1), viene attivato il timer (tim3) utilizzato per tenere il tempo per il controllo dei segnali PWM:

```

779 //attivo un timer
780 int timeValue = __HAL_TIM_GET_COUNTER(&htim3);

```

Fig. 6.2.2.2 : Attivazione timer

- Successivamente, nello USER CODE 3 (interno al while(1)) è stata definita la fase di armamento dei motori (durata di 2s) durante la quale, viene inviato ai 4 canali (CCR1, CCR2, CCR3, CCR4) un segnale PWM di 950 (a cui corrisponde un duty cycle del 4,75%).

```

758 //armamento dei motori
759 if (i <= 100000) {
760
761     TIM3->CCR1 = 950;
762     TIM3->CCR2 = 950;
763     TIM3->CCR3 = 950;
764     TIM3->CCR4 = 950;
765
766 }

```

Fig. 6.2.2.3: Fase armamento dei motori

Dove la variabile **i**, definita nello USER CODE 2 (come precisato nella figura seguente), è necessaria per il conteggio del tempo. Dopo numerose prove è stato riscontrato che lo scorrere del tempo del timer utilizzato viene calcolato in microsecondi per di più è necessario dividere il valore per 2, quindi a 2s coincide un valore di 100000.

```

673 //variabile per tenere il tempo
674 int i = 0;

```

Fig. 6.2.2.4: Definizione variabile necessaria per il conteggio del tempo

Terminata tale fase, vengono calcolati i valori dei PID (compresi tra 1200 e 2000), dunque i segnali da inviare ai 4 canali PWM per la fase di esecuzione Real Time.

```

809 //calcolo il duty cycle dalle velocità
810 avgMotor1 = map(*(Speeds+0), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
811 avgMotor2 = map(*(Speeds+1), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
812 avgMotor3 = map(*(Speeds+2), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
813 avgMotor4 = map(*(Speeds+3), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
814
815 //passo il duty cycle ai PWM
816
817     TIM3->CCR1 = avgMotor1;
818
819     TIM3->CCR2 = avgMotor2;
820
821     TIM3->CCR3 = avgMotor3;
822
823     TIM3->CCR4 = avgMotor4;
824

```

Fig. 6.2.2.5 : Segnale PWM

- Nel momento in cui il pulsante viene premuto nuovamente, con lo scopo di spegnere i motori (variabile flagCallback = 0), viene inviato ai motori un segnale di PWM molto

basso (pari a 10), a cui corrisponde un duty cycle dello 0,05%, ma non nullo per rendere lo spegnimento più rapido.

La variabile `i` viene settata a 0, poiché avvenuto lo spegnimento dei motori, non è più necessario mantenere il conteggio del tempo.

```
844     //pulsante spento
845     else if (flagCallback == 0) {
846
847         HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);
848
849         i = 0;
850
851         TIM3->CCR1 = 10;
852         TIM3->CCR2 = 10;
853         TIM3->CCR3 = 10;
854         TIM3->CCR4 = 10;
855
856     }
```

Fig. 6.2.2.6: Fase spegnimento dei motori

7. Connettori

Di seguito viene fornita la tabella dei connettori utilizzati per interfacciare le componenti hardware del quadricottero con la scheda [STM32H745](#):

COLLEGAMENTI SUI CONNETTORI DELLA SCHEDA STM32H745

Periferica esterna	Connettore	PIN	Descrizione
IMU	CN7	2	SCL – PB8
	CN7	4	SDA – PB9
	CN8	7	+3.3 V
	CN10	5	GND
OLED Display	CN10	14	SCL – PB6
	CN10	16	SDA – PB7
	CN8	7	+3.3 V
	CN8	11	GND
MOTORI	CN7	12	PWM1 – PA6
	CN7	11	PWM2 – PC7
	CN8	2	PWM3 – PC8
	CN9	7	PWM4 – PB1
Alimentazione ESC	CN8	9	+5 V
	CN8	11	GND

Viene inoltre fornita una rappresentazione grafica per individuare facilmente i PIN sulla STM32H75:

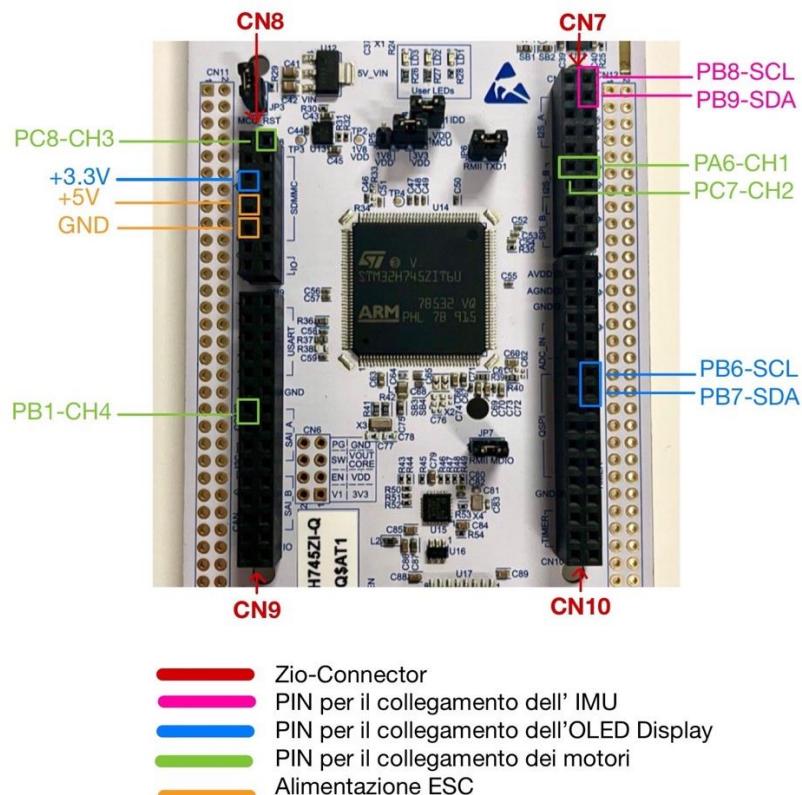
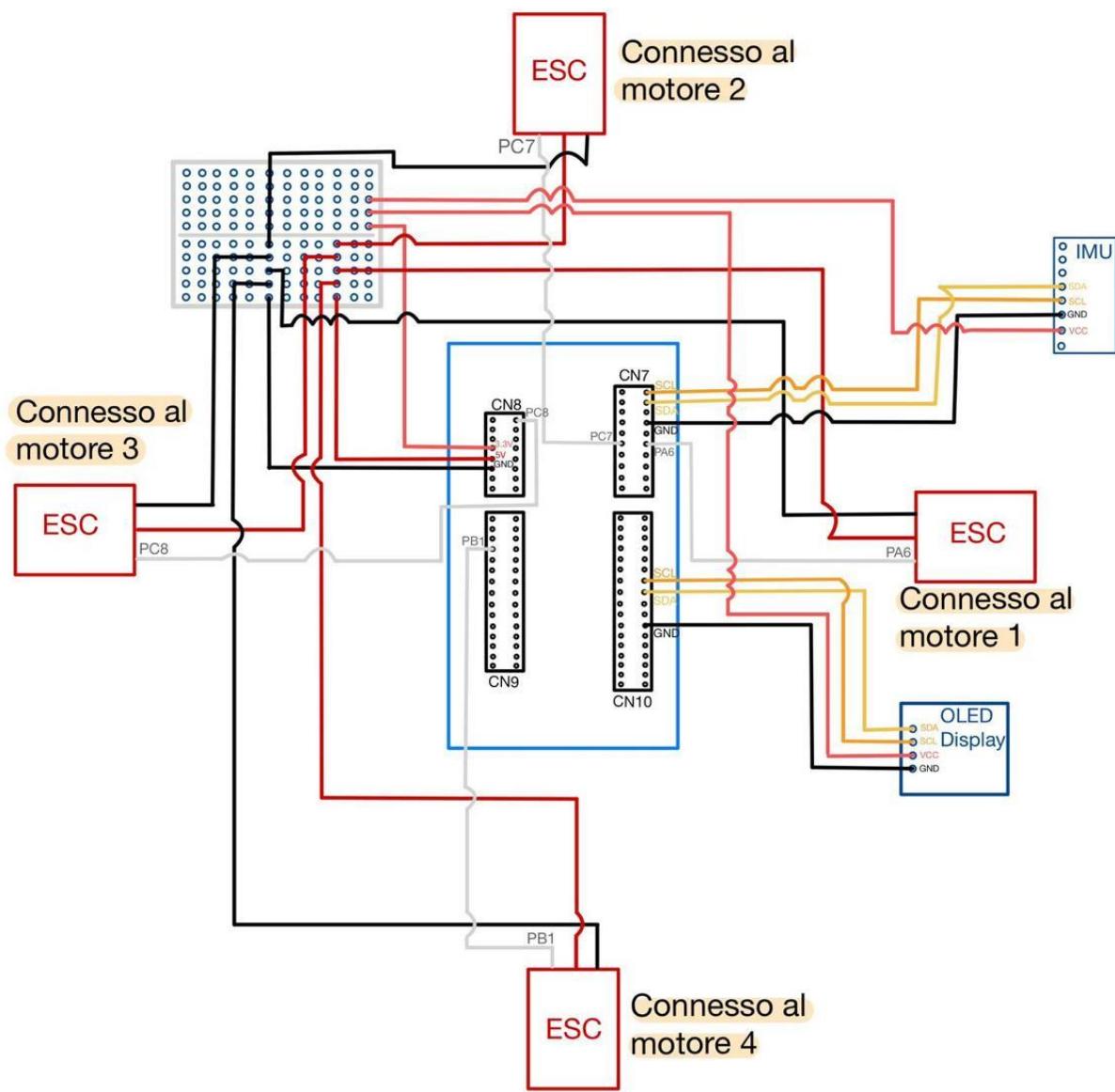


Fig. 7.1: Collegamenti sulla STM32H745

È utile precisare che la scheda STM32H745 è dotata di una singola uscita a +5V e a +3.3V, per questo motivo i collegamenti sono stati resi possibili tramite l'ausilio di una breadboard.

Di seguito è fornito lo schema di collegamento dei singoli componenti alla scheda STM32H745:



8. Attivazione del drone

L'attivazione del drone è stata suddivisa in due fasi:

- 1) Fase iniziale: test dei componenti
- 2) Fase finale: test dell'insieme

È importante ricordare che l'attivazione dei motori avviene premendo il tasto blu (USER BUTTON) sulla scheda STM32H745. Nel momento in cui viene premuto il pulsante si accendono i led LD2 e LD3 per indicare che il segnale è stato inviato ai motori.

Lo spegnimento avviene eseguendo la stessa procedura.

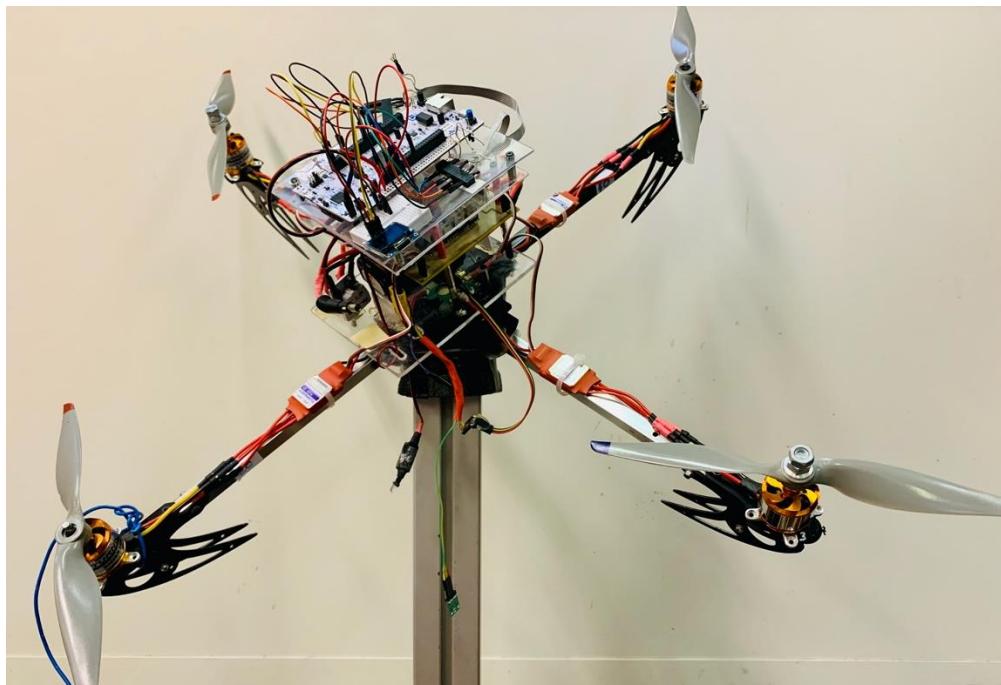


Fig. 8.1: Drone quadricottero

8.1 Prove iniziali di controllo

I test iniziali effettuati riguardano i singoli componenti: motori e sensore MPU – 6050.

• TEST MOTORI

Inizialmente sono stati eseguiti alcuni test per verificare il corretto funzionamento dei motori, inviando opportuni segnali PWM ed analizzando di conseguenza il loro comportamento.

I segnali PWM sono stati prima studiati tramite oscilloscopio: in particolare, i valori esaminati sono stati quelli coincidenti ad un duty cycle di 4.75%, 6% e 10% (range di lavoro dei motori). Ovviamente tale procedimento è stato ripetuto per ciascuno dei quattro PIN della scheda a cui è stato associato un canale di segnale PWM (si rimanda alla sezione [6.2.1](#)).

Solo in un secondo momento, è stato possibile testare i segnali su di un singolo motore (non appartenente all'hardware del drone), collegato opportunamente ad un generatore di tensione (in sostituzione alla batteria a +14.6V), ed è stato possibile osservare la risposta dei motori cambiando opportunamente il valore del duty cycle.

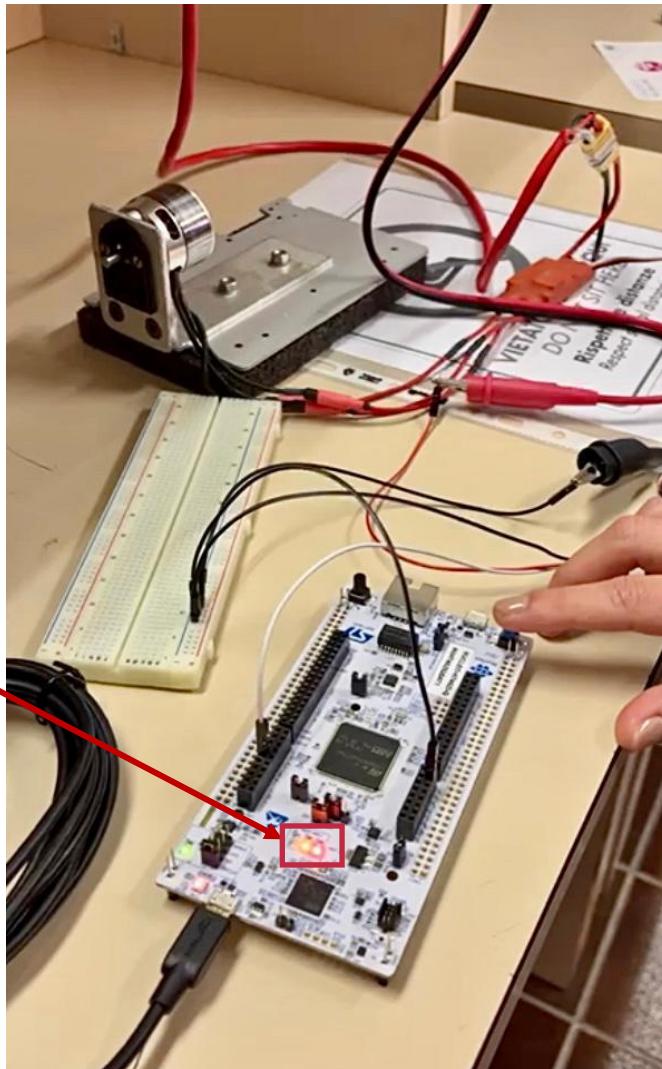


Fig. 8.1.1: Test su di un singolo motore Brushless

- TEST SENSORE MPU-6050

Il sensore MPU-6050 inizialmente è stato testato collegandolo direttamente alla scheda e visualizzando (sullo schermo ad essa collegata) se venivano prelevati dati dall'esterno. Muovendo manualmente il sensore (cercando di riprodurre i movimenti del drone) è stato possibile accertarsi se i dati relativi all'accelerometro e al giroscopio prelevati rispecchiavano le reali condizioni: il primo riportava a schermo un valore pari ad 1,00 (accelerazione gravitazionale normalizzata) in corrispondenza dell'asse del sistema di riferimento del sensore diretto verso il basso e valori nulli per gli altri due assi, mentre il secondo riportava valori variabili (coincidenti a variazioni di velocità angolare) in corrispondenza dell'asse sul quale avveniva il moto.

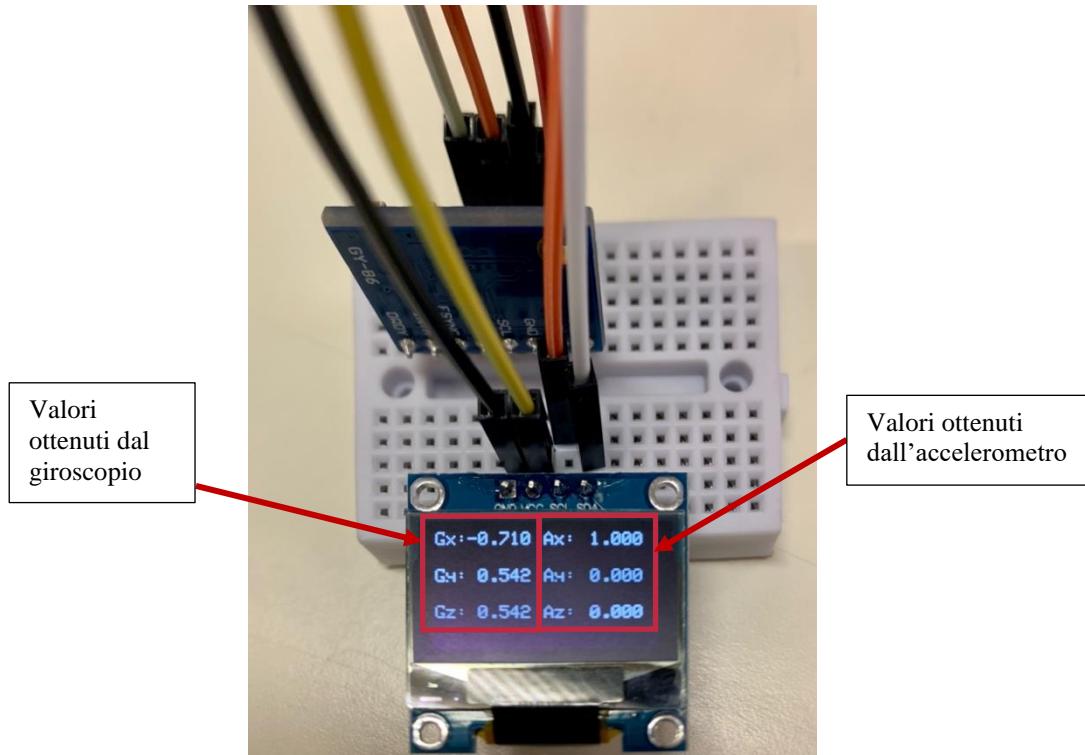


Fig. 8.1.2: Test accelerometro e giroscopio

Successivamente sono iniziati i test del magnetometro. Solo dopo opportune ricerche riguardanti il collegamento tra il sensore HMC5883L e l'IMU e numerose prove è stato possibile ottenere dei risultati riguardanti i valori del campo magnetico lungo i tre assi di riferimento del sensore. Prima di analizzare i valori prelevati dal magnetometro è stata effettuata la calibrazione: muovendo il sensore in tutte le direzioni fino a quando non compaiono tutti i settaggi sul display. La verifica delle variabili è stata effettuata utilizzando una bussola, ovvero analizzando se, l'asse lungo il quale il valore del campo era maggiore in modulo, indicasse il NORD (stando attenti alla vicinanza di dispositivi elettronici e materiali ferrosi in grado di alterare le misurazioni).

Dopo aver verificato il corretto funzionamento dell'IMU e magnetometro è stato possibile calcolare gli angoli di roll, pitch e yaw ed è stato possibile analizzare all'oscilloscopio le variazioni dei valori di duty cycle inviati ai motori muovendo opportunamente il sensore.

Essendo l'oscilloscopio dotato di due ingressi, questo test è stato effettuato collegando a due a due i PIN della scheda corrispondenti ai segnali PWM. Tramite questo test è stato

possibile osservare che il segnale di duty cycle rimaneva compreso tra il 6% ed il 10% (come desiderato) ed inoltre, durante la movimentazione del sensore, ai motori opposti veniva inviato un segnale di duty opposto e complementare.

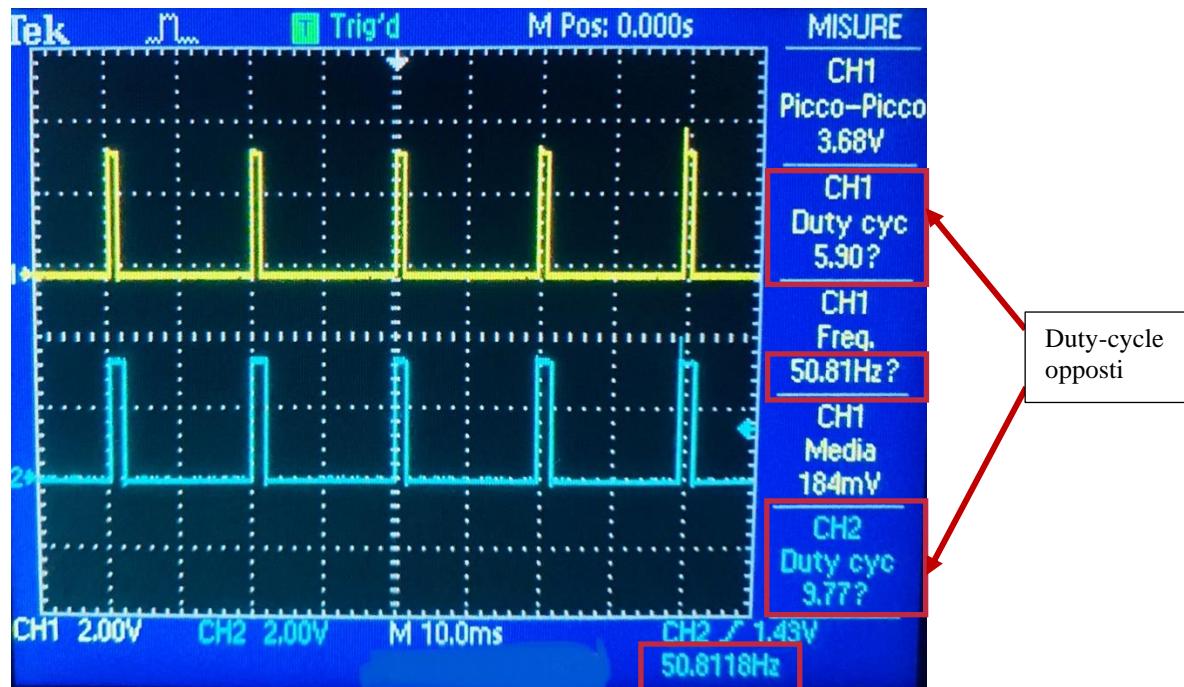


Fig. 8.1.3. Alimentazione motori opposti

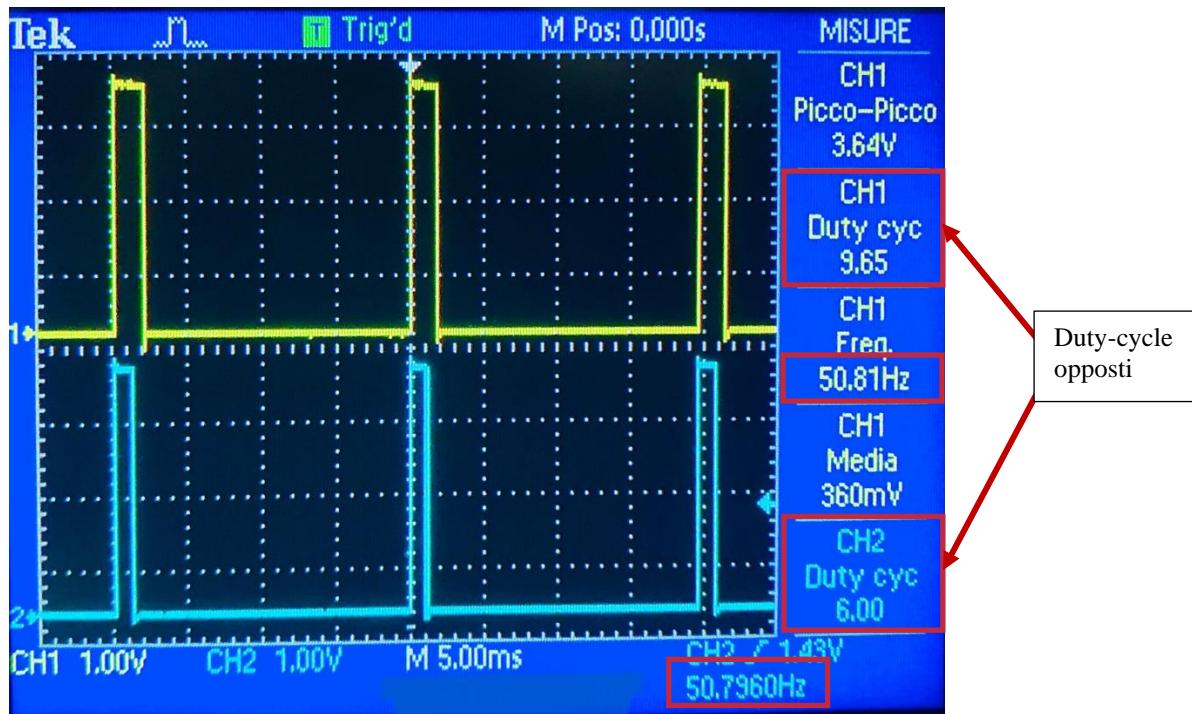


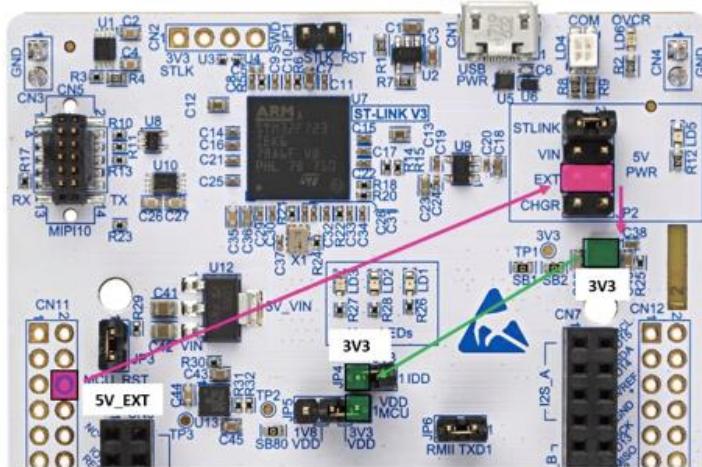
Fig. 8.1.4. Alimentazione motori opposti

Dalle Fig. 8.1.3 e 8.1.4 è inoltre possibile notare che la frequenza del segnale inviato ai due canali rimane pari a 50Hz.

8.2 Prove finali

Prima di procedere alla taratura dei PID è stato verificato che la struttura Hardware fosse correttamente predisposta.

Inizialmente si è pensato di saldare sulla scheda un pin aggiuntivo, in modo tale da alimentare la stessa senza utilizzare direttamente l'alimentazione del PC. Il pin saldato è il PIN 6 del connettore CN11.



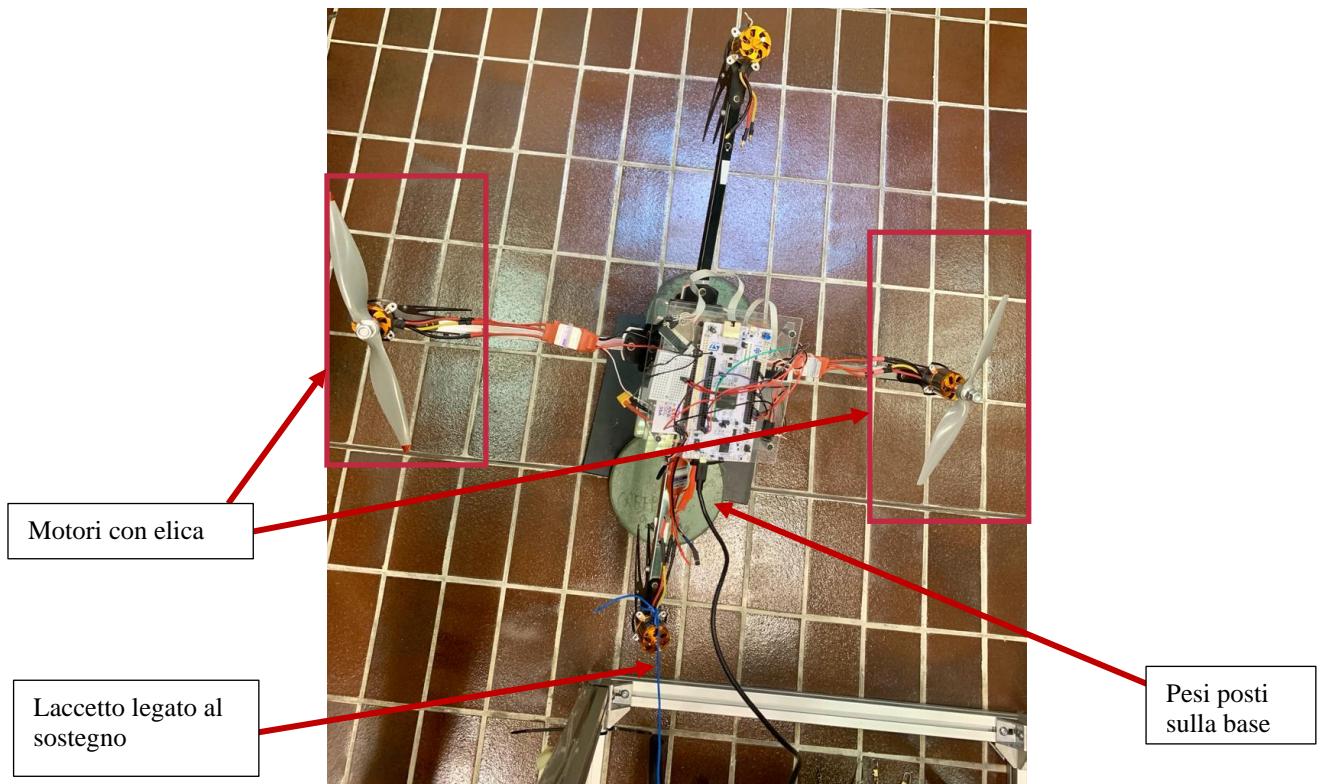


Fig. 8.2.2: Test

Di seguito sono elencate e successivamente descritte le prove principali effettuate per la taratura dei PID:

	DUTY MIN	DUTY MAX	K_P	K_D	K_I
PROVA 1	6%	7%	1	0	0
PROVA 2	6%	7%	1	0	1
PROVA 3	6%	7,5%	1	0	1
PROVA 4	6%	7,5%	1	0.05	1
PROVA 5	6%	7,5%	0.05	0.05	1
PROVA 6	6%	8%	0.05	0.05	0.5

Nelle prime due prove è stato studiato il comportamento del drone a bassa potenza, variando i coefficienti dei PID, successivamente la potenza è stata leggermente aumentata. In questo caso la risposta del drone risultava corretta per piccole variazioni angolari, dunque si è proseguito aumentando ulteriormente il duty max e variando i valori dei PID.

Analogamente si è proseguito cambiando e migliorando ulteriormente i valori.

Dopo numerose prove sono stati riscontrati diverse problematiche nella taratura dei PID probabilmente causati da:

- Vibrazioni dei motori che portano a letture errate da parte dell'IMU
- Acquisizione lenta dei valori da parte dell'IMU, in particolare dei valori del magnetometro

Per i motivi sopra descritti, si è notato che l'utilizzo del filtro di Madgwick risulta molto lento nella lettura dei dati ed in particolare per il calcolo dello Yaw. Questo ritardo causa una risposta lenta nei motori, dunque un comportamento anomalo del drone per grandi variazioni angolari. La taratura dei PID non è stata possibile completarla.

9. Eventuali sviluppi futuri

Per avvicinarci al momento di volo del quadricottero e per effettuare prove più precise dovranno essere prese in considerazione alcune modifiche alla predisposizione dei test.

Per poter svolgere prove ottimali può essere consigliato:

- Utilizzare una gabbia apposita per effettuare le prove provvedendo contemporaneamente tutti i quattro motori di eliche, poiché il supporto utilizzato per le prove preliminari non è in grado di sostenere una potenza maggiore rispetto a quella utilizzata nelle prove (solo due eliche), che non coincide con la potenza massima di lavoro dei motori.
- Predisporre un posizionamento più stabile per l'IMU in modo che non introduca errori nel momento di acquisizione dati (struttura che attutisce le vibrazioni).
- Sviluppare un modulo wi-fi per poter implementare comandi a distanza effettuando prove in sicurezza.
- Implementare ed analizzare anche i valori prelevati dall'altimetro.