Corso di Laurea

Ingegneria Informatica e dell'Automazione

Laboratorio di Automazione

Relazione Quadricottero

Docente: Prof. Bonci Andrea

Anno academico 21/22

Studenti:

Rezkalla Ebram Ebrahim Adib 1093553

Masoud Morcos Safwat Sobhi 1093552

INDICE

INTRODUZIONE

Lo scopo principale di questa relazione consiste nell'implementazione di un controllo automatico in modo da stabilizzare il quadricottero con il sistema dei motori che sopra vengono montate della ale sfruttando un processore che elabora i dati presi dai sensori che ci consentono di determinare l'orientamento del corpo nello spazio tridimensionale.

Cos'è un Quadricottero?

Un quadricottero è un modello simile a un elicottero, che può iniziare e atterrare in verticale, proprio come un elicottero.

A differenza di un elicottero che possiede una meccanica tecnicamente elaborata, un quadricottero è dotato di quattro eliche disposte orizzontalmente.

È un **aeromobile sollevato e spinto da quattro rotori**. I quadrirotori vengono classificati come aerogiri, diversi dagli aeroplani in quanto la portanza è data esclusivamente dai quattro rotori.

Fig 1: Drone quadricottero

Gli schemi più comuni sono: 4/6/8 rotori. Un apparato con tre eliche a rotore è chiamato tricottero; con quattro - un quadricottero; con sei - un esacottero e con otto rotori - un ottocottero. L'uso di ciascuno degli schemi determina sostanzialmente un parametro come - la massa del carico utile o, più semplicemente, il peso che il multirotore può portare a bordo.

Fig 2: Drone tricottero

Fig 3: Drone esacottero

Fig 4: Drone ottocottero

Principio di funzionamento

Il principio di funzionamento di un moderno multirotore è semplice. A seconda dello schema di applicazione (3/4/6/8 rotore), ogni rotore è azionato da un motore elettrico (motore elettrico). Una metà dei motori ruota le viti in senso orario e l'altra in senso antiorario. Grazie a questo approccio, per controllare il multirotore, non è richiesto l'uso di un rotore di coda e di un complesso piatto oscillante nel drone.

Un esempio di controllo di volo

- Aumento del numero di giri su tutti i rotori - sollevamento.
- Diminuzione della velocità su tutti i rotori - diminuzione.
- Aumentando i giri di una metà delle viti e diminuendo i giri dell'altra metà si metterà il movimento da parte.
- Aumentando l'RPM per le eliche in senso orario e diminuendo l'RPM per le eliche in senso antiorario, il drone ruoterà.

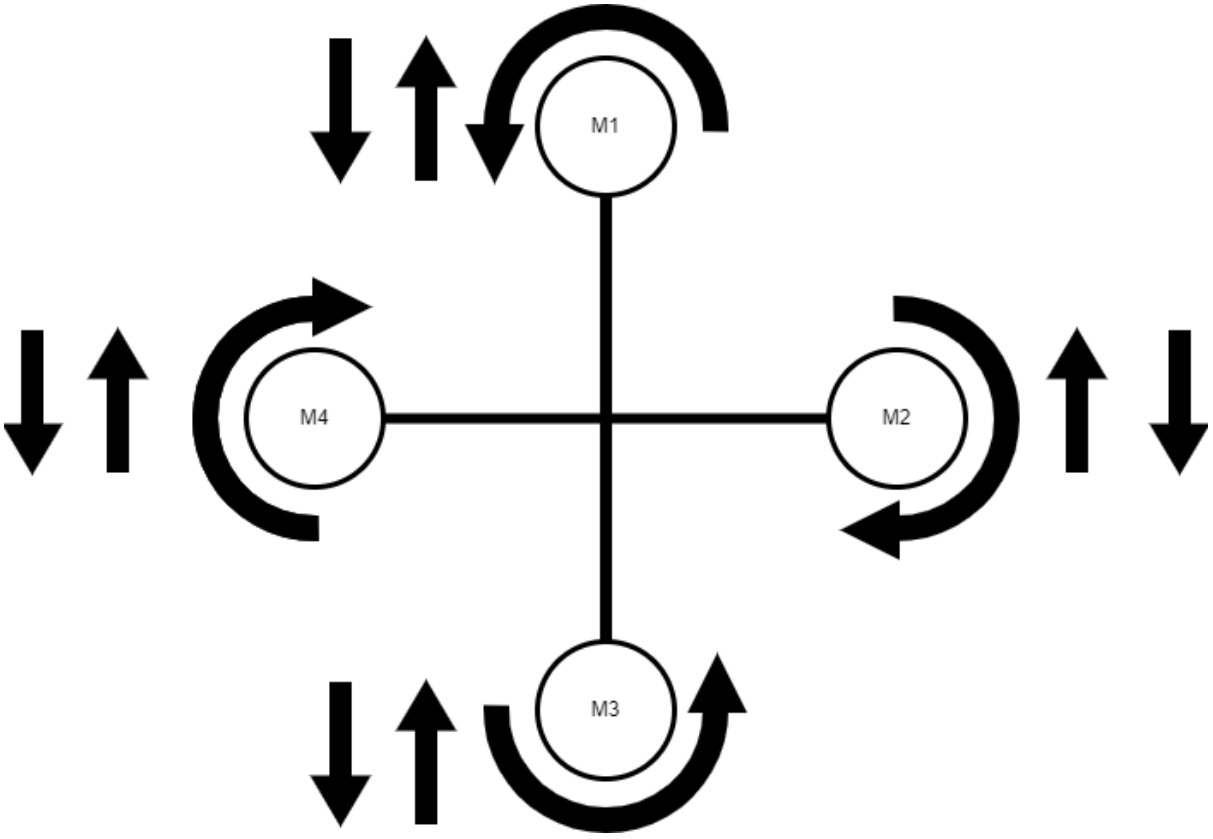


Fig 5: Variazione RPM per i 4 motori per il controllo del volo

Struttura di un quadricottero

Si può quindi definire il drone come un aerogiro radiocomandato a tutti gli effetti. Come già introdotto esistono plurime soluzioni costruttive: tricottero, quadricottero, esacottero ed ottocottero votati semplicemente a diverse condizioni di carico necessario. Per quanto diversi, il principio di funzionamento è simile per tutti i modelli, quindi, si presuppone che tutta la componentistica sia affine. Partiamo dal telaio, lo scheletro del nostro mezzo, con il compito di sostenere il peso e resistere alle sollecitazioni nel corso di tutta la vita operativa. Nello studio in dettaglio ci si è indirizzati sui quadricotteri, i più comuni sul mercato. Di svariate forme, in un quadricottero esso ha una base centrale da cui si estenderanno tanti bracci quanti sono i motori adottati, perciò quattro bracci. L'aspetto fondamentale su cui si insiste è lo stesso che in qualunque tipo di struttura: rigidezza e peso. Se il telaio fosse troppo pesante il payload disponibile sarebbe notevolmente inferiore o richiederebbe una motorizzazione superiore per lo spostamento e questo influirebbe, in più, negativamente su costi e consumi. Se il materiale fosse invece eccessivamente elastico le forze agenti alle estremità dei bracci, dovute al tiro delle eliche, deformerebbero troppo la struttura causando uno spostamento delle forze dalla perpendicolare al piano appartenente al quadricottero. Ne consegue una maggiore instabilità nelle fasi più cruciali del volo o deviazioni dalla traiettoria prevista. Un telaio leggero e rigido in alluminio o materiale composito permette di risparmiare peso, così da incrementare il carico pagante, o di necessitare di meno energia per la stessa manovra aumentando l'autonomia in volo, uno dei punti focali per il futuro sviluppo dei droni. L'autonomia è espressa dalla capacità delle batterie installate che sono sempre posizionate al centro della struttura: coprendo una grossa percentuale del peso complessivo posizionarle nei pressi del baricentro aumenta molto la stabilità e ne diminuisce l'inerzia delle manovre. Attualmente vengono utilizzate quelle a Litio-Polimero (LiPo) per una serie di vantaggi su molte delle formule più comuni in commercio come Litio-Ione (LiIon) o Nichel-Cadmio (NiCd). La mancanza di un solvente organico all'interno le rende innanzitutto più sicure se danneggiate, essendo costituite di un materiale solido: appunto il polimero. Il materiale costituente inoltre non richiede un involucro metallico, per cui sono più leggere, e ha una densità di energia maggiore avendo un impacchettamento più denso. La capacità di essere sagomata a piacimento e un maggiore ciclo di vita infine ne hanno fatto la scelta più vantaggiosa. Possono essere disposte in serie o parallelo, atte a tenere in vita i sistemi di controllo del drone e di raccolta dati. Il primo utilizzatore è sicuramente l'apparato propulsivo. Quest'ultimo è formato da motori, ingranaggi, eliche. I motori sono elettrici, maggiormente economi e gestibili rispetto a quelli a pistoni. Tuttavia, scartiamo i motori trifase per le dimensioni e il peso, e quelli a spazzole per i problemi di scintillio dovuti agli alti regimi di rotazione. Restano quindi i motori elettrici brushless (senza spazzole) a cassa rotante più adatti all'utilizzo in questione. L'energia è fornita dalle batterie ma il giusto voltaggio di cui i propulsori necessitano è garantito dalla presenza di regolatori (ESC). Questi componenti, uno per ciascun rotore, agiscono tramite un input del controllore che da loro l'informazione sul voltaggio corretto. La rotazione impressa dai motori viene trasmessa tramite ingranaggi con un determinato rapporto di marcia alle eliche. La scelta dell'elica deve essere fatta in modo tale da trovare il corretto punto di funzionamento con il motore scelto; quindi, nel regime dove viene espressa la potenza massima. Se così non fosse, l'elica, per quanto prestazionale, non lavorerebbe con la giusta efficienza: in termini pratici un'elica

troppo leggera porterebbe il motore fuori giri, in caso contrario, una troppo pesante non riuscirebbe ad essere sufficientemente trascinata. Spinta e spostamento in ogni direzione sono perciò ottenuti da eliche collegate ai propulsori che generano variazioni di portanza. Eliche a passo fisso sono più comuni per il costo e la semplicità costruttiva ma sono disponibili anche soluzioni più complesse con l'utilizzo di eliche a passo variabile: nelle prime le variazioni di portanza si hanno variando i giri del motore mente nelle seconde modificando l'incidenza delle pale rispetto al flusso. Soluzioni economiche sono costruite in polimero ma si possono avere anche eliche in fibra di carbonio. Il funzionamento corretto di tutti questi organi è coadiuvato dal controllore del nostro drone. Anch'esso posizionato al centro per le dimensioni che questo elemento occupa, il controllore non è altro che una scheda programmabile, molto spesso di tipo Arduino. Su di essa è montato un processore per l'analisi dei dati dai vari input provenienti da piattaforme inerziali (IMU). La IMU (Inertial Measurement Unit) misura accelerazioni e velocità angolari tramite accelerometri e giroscopi che registrano posizione e velocità. Note queste informazioni il processore invia ai regolatori il segnale per il corretto voltaggio da fornire alla motorizzazione utilizzata.

È essenziale per avere un volo stabile che due **motori** ruotino in **senso orario** e due motori in **senso antiorario**. Nell'elicottero la stabilità viene garantita dal rotore di coda che contrasta il movimento rotatorio del rotore principale.

Il quadricottero, avendo i motori e le eliche montate nello stesso piano, deve equilibrare il moto impiegando due motori che ruotano in senso orario e due in senso antiorario. La contrapposizione dei motori permette quindi di annullare l'energia rotatoria degli stessi.

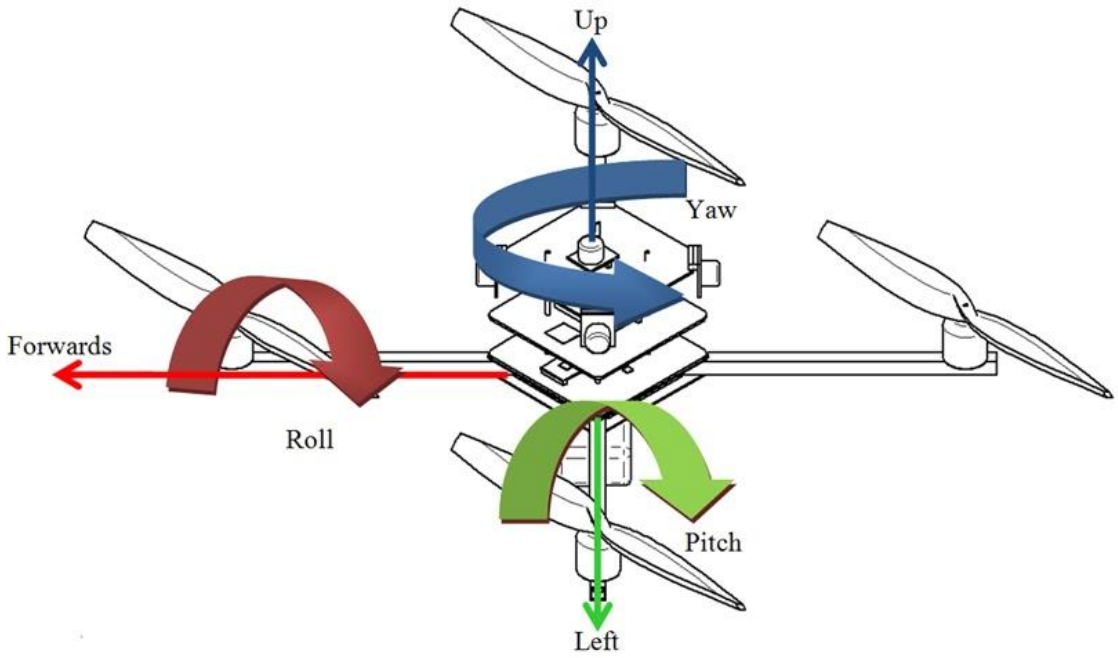


Fig 6: Movimenti del drone Roll, pitch e yaw (movimenti assiali)

CONFIGORAZIONE

Le configurazioni tipiche di un quadricottero sono due e vengono definite dalla direzione di volo in relazione alla posizione dei motori; nella figura seguente potete osservare il modo x ed il modo +:

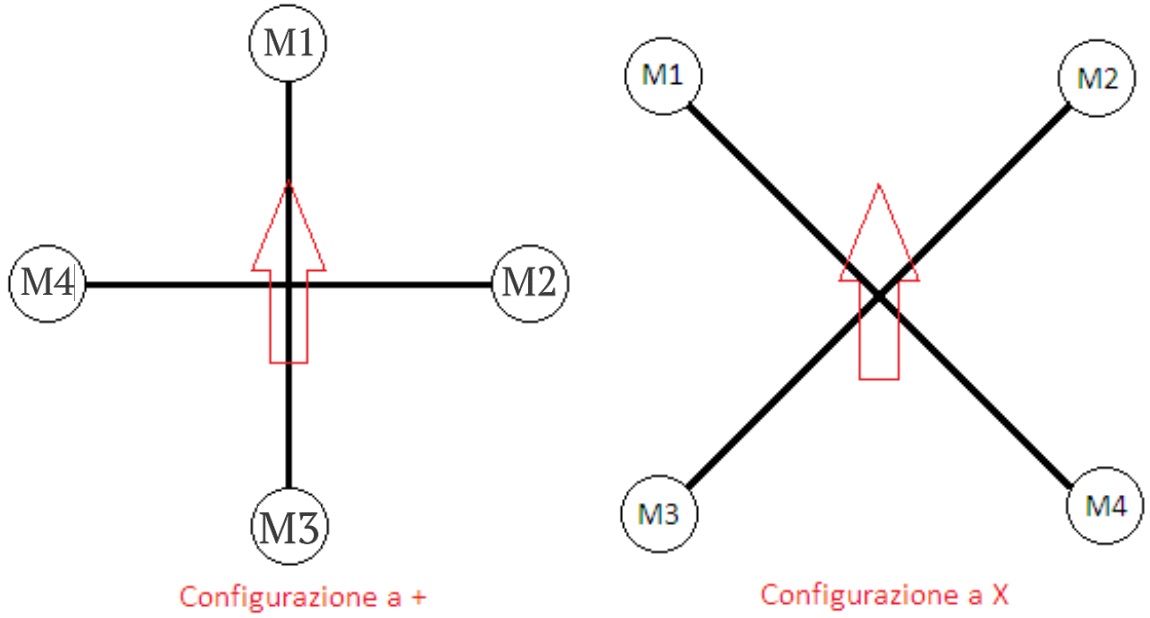


Fig 7: Le due configurazioni + e x

La freccia rossa indica il senso di marcia in avanti

Nella configurazione + (quella che utilizziamo del nostro progetto) la direzione di volo corrisponde all'asse creato dai motori M1 e M3 mentre, nella configurazione x la direzione di volo corrisponde all'asse creato tra i motori M1 e M4 e l'asse creato tra i motori M2 e M3.

I motori, dotati di eliche, sono rappresentati in figura dalle sigle M1, M2, M3, e M4 e vengono montati, tipicamente, alle estremità degli assi del telaio (Frame), l'elettronica di controllo e la batteria sono montati solitamente al centro del drone, dove si intersecano gli assi.

COME VOLA IL QUADRICOTTERO?

È necessario fare una piccola parte teorica per conoscere come vola il nostro quadricottero, questo per me è stato importante sia per capire come montare i vari componenti sia per quanto riguarda la gestione del quadricottero durante primi voli.

Ora, nel sempre nel caso teorico, in cui i pesi del quadricottero sono perfettamente identici in tutti gli assi, e non ci sono elementi esterni che disturbano il volo come folate di vento, la velocità di rotazione dei quattro motori rimane uguale.

Immaginiamo ora che la velocità di rotazione dei motori sia tale da tenere il quadricottero a una quota di 1 metro dal suolo.

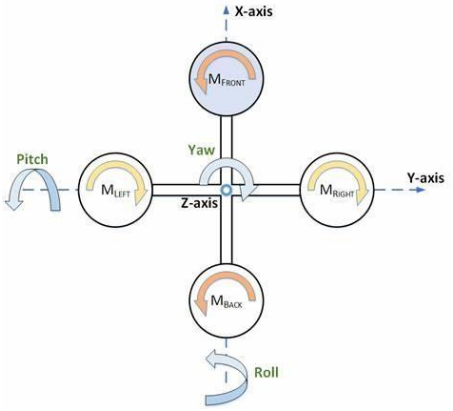


Fig 8: Senso di rotazione dei motori e gli assi di riferimento

Beccheggio

Consideriamo sempre il nostro quadricottero in volo è perfettamente stabile (situazione teorica), per poterlo far muovere in avanti dovremmo agire sul **beccheggio**. Per farlo **muovere in avanti** la centralina di controllo montata al centro del quadricottero, dovrà **aumentare** la velocità di **rotazione** del motore posteriore **M3** e **diminuire** la velocità di **rotazione** del motore anteriori **M1**. In questo caso il modello si inclinerà in avanti permettendone il movimento verso quella direzione.

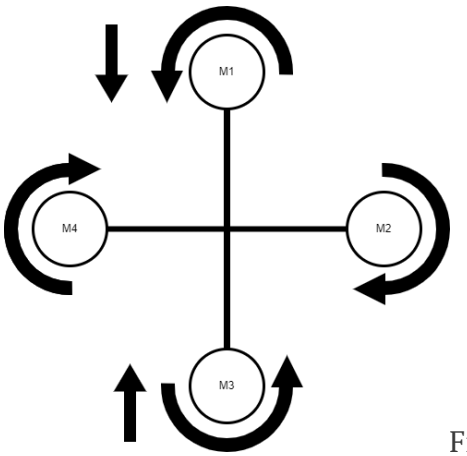


Fig 9: Movimento in avanti

Per **portare** il quadricottero **indietro** la centralina dovrà eseguire l'operazione opposta, ovvero **aumentare** la velocità di **rotazione** del motore **M1** e diminuire la velocità di rotazione del motore **M3.**

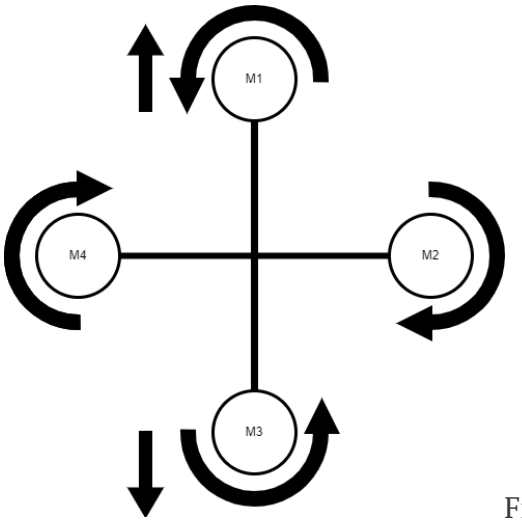


Fig 10: Movimento in dietro

Rollio

Tramite il **rollio** potremo far **spostare** il nostro quadricottero **lateralmente**. Anche in questo caso il movimento del quadricottero avviene variando la velocità dei motori in modo tale da poter inclinare su un lato il quadricottero permettendone quindi lo spostamento a destra o sinistra.

Spostamento a Destra (Relativo all'osservatore posto di fronte al quadricottero) il motore M4 aumenta la velocità di rotazione, il motore M2 diminuisce la velocità di rotazione. Otteniamo una inclinazione a destra con conseguente movimento del quadricottero verso quella direzione.

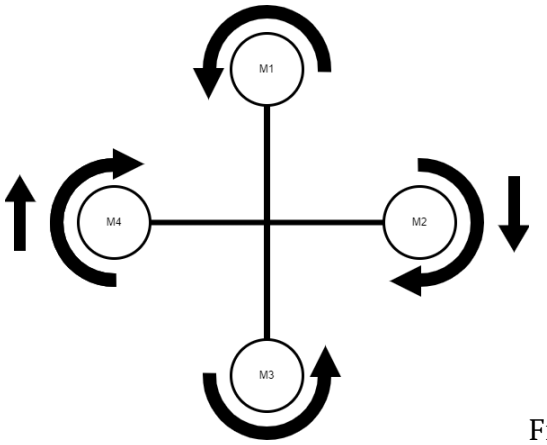


Fig 11: Movimento a destra

Spostamento a Sinistra (Relativo all'osservatore posto di fronte al quadricottero) il motore M2 aumenta la velocità di rotazione, il motore M4 diminuisce la velocità di rotazione. Otteniamo una inclinazione a sinistra con conseguente movimento del quadricottero verso quella direzione.

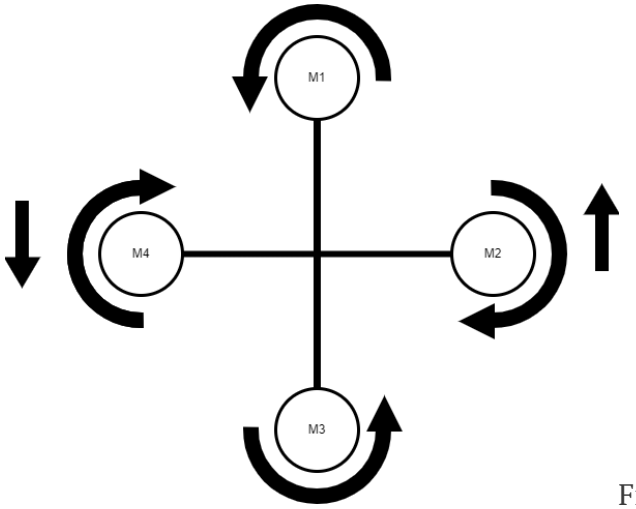


Fig 12: Movimento a sinistra

Imbardata

L'altro movimento che può eseguire il quadricottero è la rotazione su sé stesso, definita imbardata. Il movimento avviene variando la velocità di rotazione di due motori posizionati nello stesso asse.

Rotazione oraria effettuata aumentando la velocità di rotazione dei motori M2 ed M4 e diminuzione della velocità di rotazione dei motori M1 e M3.

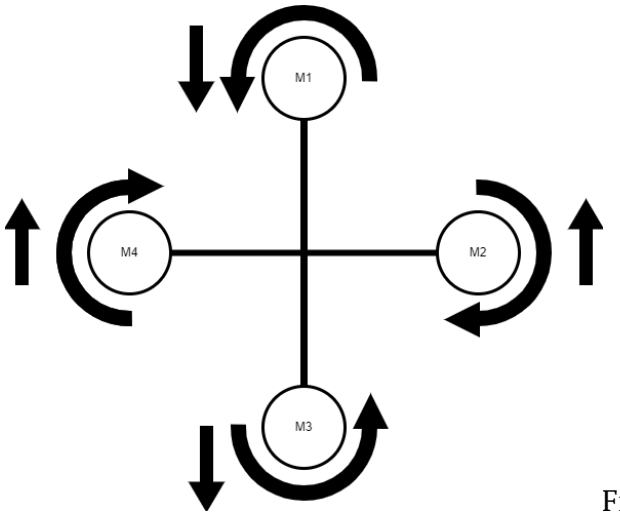


Fig 13: Rotazione oraria

Rotazione antioraria CCW effettuata diminuendo la velocità di rotazione dei motori M2 ed M4 e aumentando la velocità di rotazione dei motori M1 e M3.

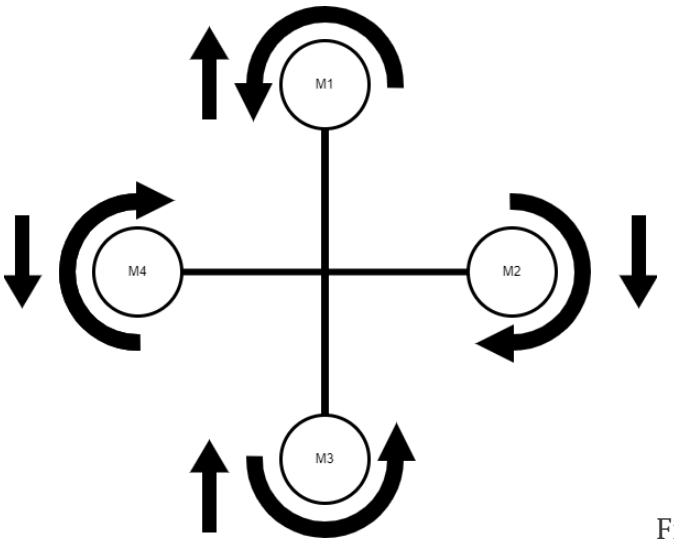


Fig 14: Rotazione antioraria

Credo che per ora queste nozioni siano quelle basilari che dobbiamo conoscere ed imparare per poi affrontare nel prossimo articolo la parte relativa a tutti gli elementi che compongono il quadricottero.

Anche le eliche devono essere montate in modo appropriato, infatti nei due motori che girano in senso orario andranno montate le **eliche CW** mentre le nei due motori che girano in senso antiorario dobbiamo montare le **eliche CCW**.

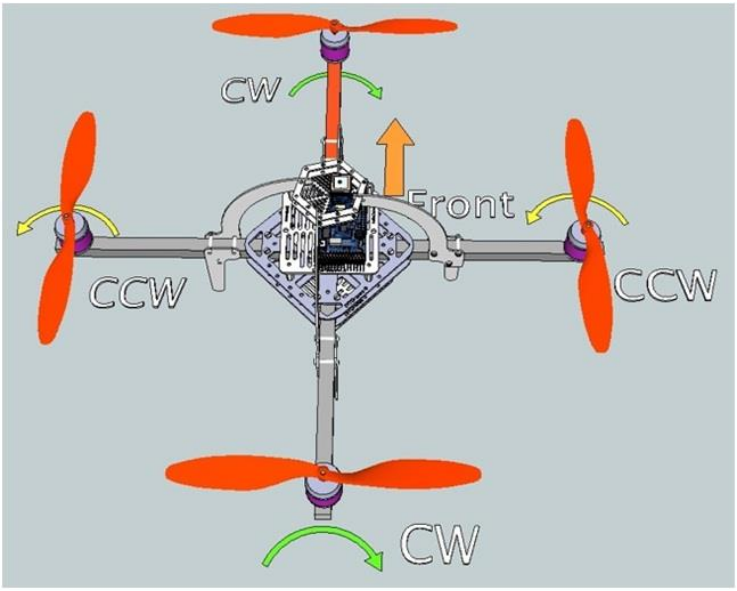


Fig 15: Senso di rotazione con le eliche

MODELLO MATEMATICO QUADRIROTORE

Sistemi di riferimento e grandezze fondamentali

Prima di introdurre il modello matematico del drone, introduciamo i sistemi di riferimento su cui baseremo la trattazione:

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

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

dove:

E_I sistema di riferimento inerziale assoluto (suolo);

E_B sistema di riferimento inerziale solidale al corpo del drone L'origine di EB è posta nel centro di massa.

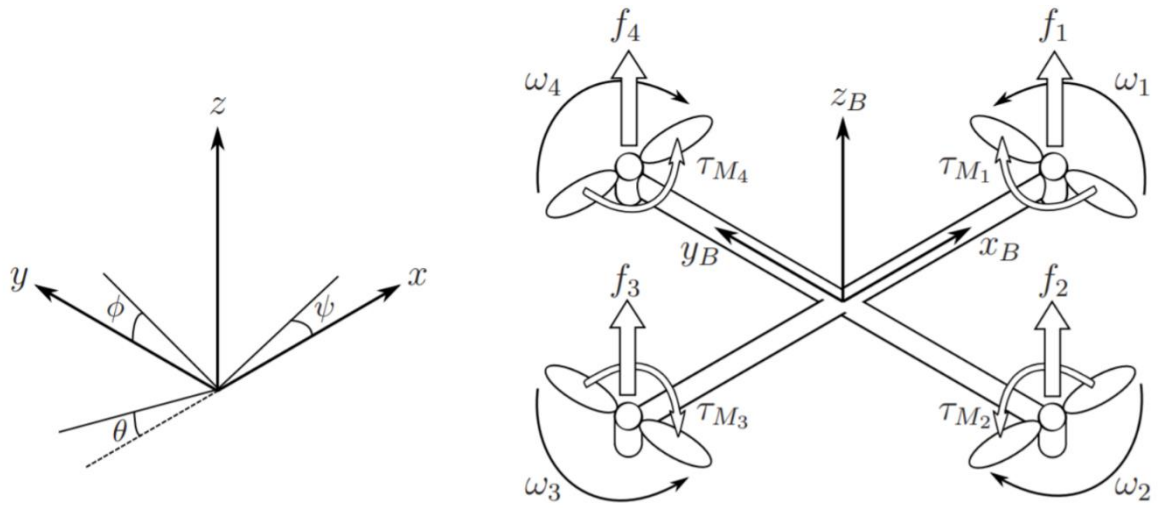


Fig 16: Rappresentazioni del sistema inerziale assoluto e di quello solidale al quadricottero

Inoltre, definiamo ξ la posizione assoluta del drone in E_I e η l'assetto del drone inE_I. Osservando l'assetto più nel dettaglio abbiamo: l'angolo di pitch φ che rappresenta la rotazione del quadricottero attorno all'asse x, l'angolo di roll θ che rappresenta una rotazione attorno all'asse y e l'angolo di yaw ψ che rappresenta una rotazione attorno all'asse z.

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

Ora, possiamo introdurre le velocità lineari VB e angolari ν viste dal sistema EB:

$$V_B = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \qquad v = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Possiamo ricondurre queste velocità alle corrispondenti $\dot{\xi}$ e $\dot{\eta}$, rispettivamente, attraverso la matrice di rotazione R e la matrice W_η:

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

$$W_\eta = \begin{bmatrix} 1 & \sin(\psi)\cos(\phi) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & \sin(\phi) / \cos(\theta) & \cos(\phi) / \cos(\theta) \end{bmatrix}$$

infatti:

$$\dot{\xi} = RV_B \qquad \dot{\eta} = W_\eta v$$

Infine, si suppone che il drone sia perfettamente simmetrico, rispetto agli assi x e y. In questo modo gli assi principali d'inerzia coincidono con gli assi di simmetria e abbiamo che Ixx = Iyy.

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

Caratterizzazione dinamica

È giunto il momento di caratterizzare il quadricottero dal punto di vista dei momenti e delle forze: ogni motore i ruotando alla velocità angolare ω_i genera una forza F_i tale che:

$$F_i = b{\omega_i}^2$$

Fi rappresenta la spinta fornita da ogni motore che si oppone, di fatto, alla forza di gravità. Per questo motivo b è chiamato coefficiente di spinta. in totale 4 motori generano una spinta T lungo l'asse z di E_B

$$T = \sum_{i=1}^{4} F_i = k \sum_{i=1}^{4} {\omega_i}^2, \qquad T_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

L'altro effetto dato dalla rotazione è la creazione di un momento torcente τ_M attorno all'asse del motore pari a:

$$\tau_M = d{\omega_i}^2 + I_M\, \omega_i$$

dove:

d è detto coefficiente di drag e I_M è il momento di inerzia del motore.

Non resta che analizzare gli effetti delle coppie di forze generate dai diversi motori:

• Le spinte dei motori 2 e 4 si contrappongono sull'asse x, è chiaro che se una delle due spinte prevale sull'altra si ha una variazione dell'angolo di roll φ.

• Possiamo dire lo stesso per i motori 1 e 3, i quali, però si contrappongono sull'asse y e uno sbilanciamento delle spinte provocherà una variazione dell'angolo di pitch θ.

• In ultima istanza, come si nota anche dalla Figure 3 alla pagina 6, i motori ruotano a 2 a 2 in senso opposto, in questo modo, i momenti torcenti da essi generati si annullano fra loro se ω1 = ω2 = ω3 = ω4, in caso contrario si ha una variazione dell'angolo di yaw ψ. Formulando matematicamente questi concetti si ottiene, τ_B:

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

dove per l si intende la distanza tra ogni motore e il centro di massa.

Modello di controllo

Ora che abbiamo compreso il rapporto tra le velocità di rotazione dei motori e i momenti che esse generano, possiamo, sulla base di ciò, stabilire le 4 uscite u_i (denominate ingressi virtuali) che i controllori dovranno generare per porre il drone nella posizione e nell'assetto desiderato:

$$\begin{cases} u_1 = b({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2) \\ u_2 = lb(-{\omega_2}^2 + {\omega_4}^2) \\ u_3 = lb(-{\omega_1}^2 + {\omega_3}^2) \\ u_4 = d(-{\omega_1}^2 + {\omega_2}^2 - {\omega_3}^2 + {\omega_4}^2) \end{cases}$$

dove:

• u_1, rappresenta la spinta T che i motori devono generare per spostare il drone lungo l'asse z.

• u_2 e u_3 rappresentano i momenti che modificano l'inclinazione degli assi x_B e y_B (quindi portano a una variazione di φ e θ).

• u_4 rappresenta il momento torcente totale e viene utilizzato per ruotare il drone attorno all'asse z_B (quindi porta a una variazione di ψ).

Controllo dell'assetto

Non resta che definire i loop di controllo che generano le u_1 :

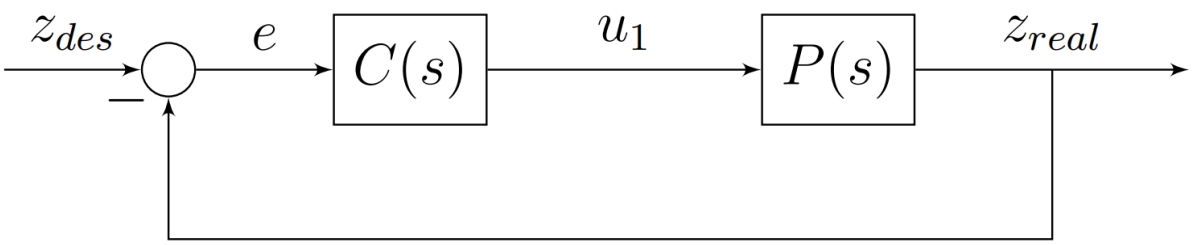


Fig 17: Anello di controllo per l'altitudine z

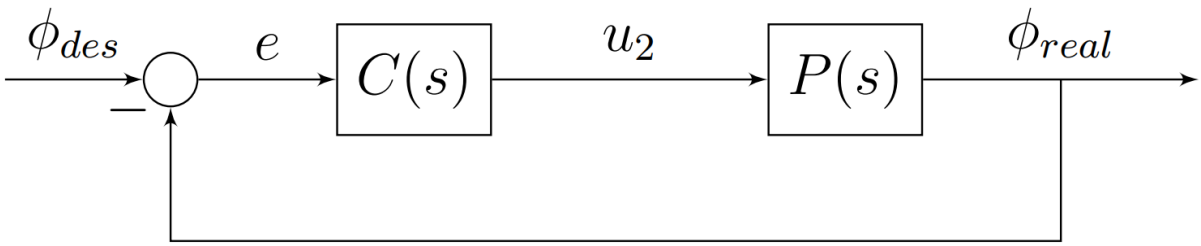


Fig 18: Anello di controllo per l'angolo di pitch φ

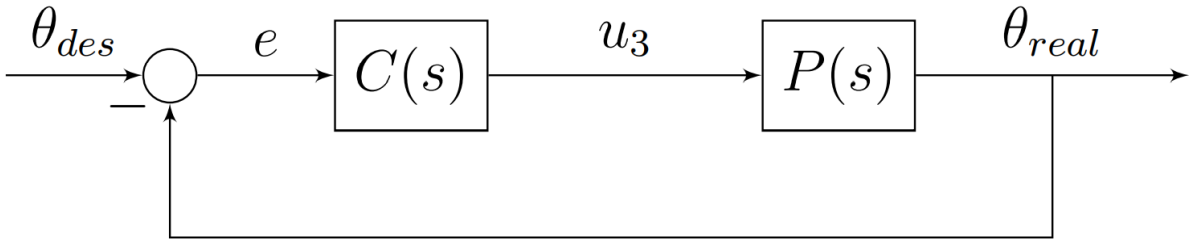


Fig 19: Anello di controllo per l'angolo di roll θ

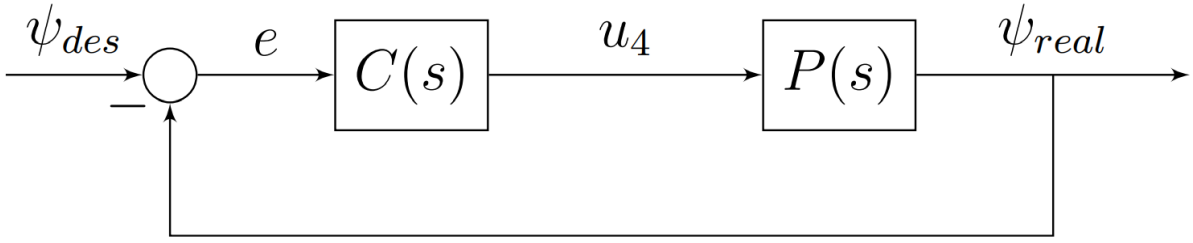


Fig 20: Anello di controllo per l'angolo di yaw ψ

Come si nota, il modello di drone che abbiamo introdotto ci permette di utilizzare un approccio SISO, descrivendo un loop di controllo per ogni angolo di assetto.

Controllori PID

Utilizzando dei controllori PID caratterizzati dalla forma generale $u(t) = K_P e(t) + K_D \frac{de(t)}{dt} + K_I \int e(t)dt$ possiamo schematizzare l'uscita del controllore C(S) come:

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

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

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

$$u_3 = K_{p,\psi}(\psi_{des} - \psi_{real}) - K_{D,\psi}(\psi_{real})$$

Dove si è considerato un controllore proporzionale e derivativo (PD) per l'assetto e che gli angoli di pitch, roll e yaw desiderati fossero posti a 0

Conversione dell'uscita di controllo in velocità di rotazione

Abbiamo definito la dinamica del nostro sistema e abbiamo gli ingressi virtuali agli attuatori (motori). Tuttavia, questi ingressi non possono essere inviati direttamente ai motori perché ognuno di essi è generato in modo svincolato dagli altri. Per ottenere un risultato globale che comporti una variazione generale dell'assetto possiamo avvalerci del sistema introdotto nella sezione del modello di controllo che riportiamo qui sotto in forma matriciale:

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

invertendo la matrice si ottiene:

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

Calcolo dei coefficienti

Al fine di tarare i PID è fondamentale conoscere con precisione i valori dei coefficienti di spinta(b) e di resistenza aerodinamica(d).

Il valore del coefficiente b varia in base al voltaggio della batteria in uso. La batteria maggiormente utilizzata da noi, per i vari test dei componenti del drone, è una Turnigy A-Spec - quattro celle. Durante il nostro percorso, ci siamo ritrovati ad utilizzare anche una Turnigy A-Spec - tre celle e abbiamo quindi deciso di calcolare il coefficiente in tutti e due i seguenti casi:

• Caso batteria Turnigy A-Spec - tre celle:

$$b_3 = \frac{T_3}{{\omega_3}^2} = \frac{1.52}{(1143.53)^2} = 1.162 * 10^{-6}$$

Caso batteria Turnigy A-Spec - quattro celle:

$$b_4 = \frac{T_4}{{\omega_4}^2} = \frac{2.07}{(1334.12)^2} = 1.163 * 10^{-6}$$

b_3 e b_4 rappresentano i coefficienti di spinta calcolati rispetto ad un singolo motore. - T_3 e T_4 rappresentano la spinta del motore, differente a seconda della batteria. - ω_3e ω_4 rappresentano la velocità del motore, anch'essa differente a seconda della batteria.

te d abbiamo inizialmente riscontrato diversi problemi. Da ricerche fatte, abbiamo capito che l'unica formula matematica, attraverso la quale è possibile ricondursi a questo coefficiente, è quella che descrive il calcolo di C_d (Resistenza aerodinamica).

$$C_d = \frac{d}{\frac{1}{2}\rho V^2 S}$$

-ρ: Densità del fluido (nel nostro caso, l'aria)

-V: Modulo velocità del corpo rispetto al fluido indisturbato.

-S: Area di riferimento (nel nostro caso, l'area che l'elica ricopre ruotando)

È possibile calcolare d solamente attraverso l'inversa di tale formula, cosa impossibile nel nostro caso poiché non possediamo il valore di C_d. C_d non è calcolabile senza d e viceversa, ne deduciamo quindi che l'unica soluzione a questo problema è quella di calcolare d mediante via sperimentale o mediante tentativi durante la taratura dei PID.

HARDWARE

Scheda STM32 H745 Nucleo-144

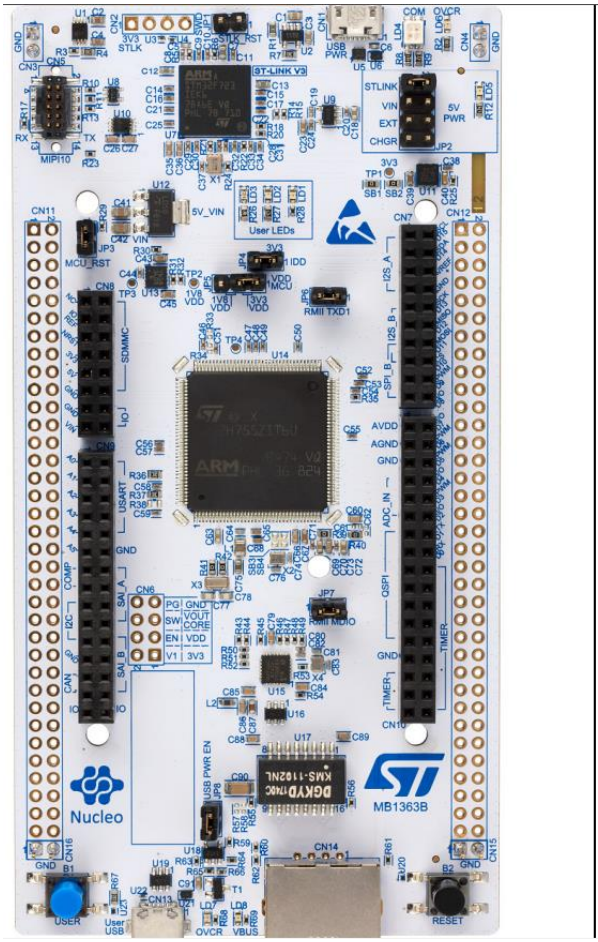


Fig 21: Vista fronte scheda

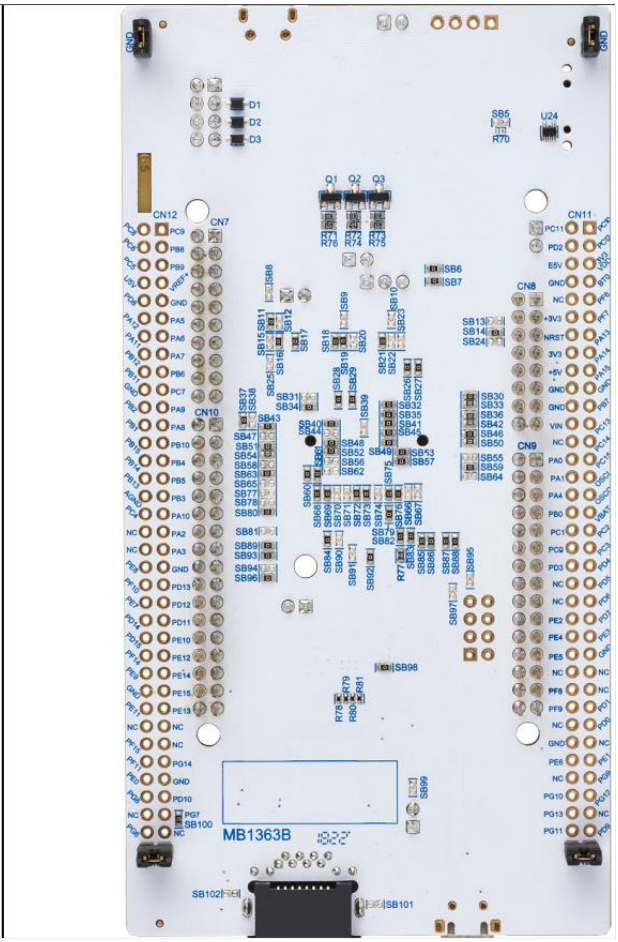


Fig 22: Vista retro-scheda

Per il nostro progetto abbiamo utilizzato la scheda: NUCLEO H745ZI-Q che combina le prestazioni dell'Arm Cortex-M7 (con unità a virgola mobile a doppia precisione) fino a 480 MHz e del core Arm Cortex-M4 (con unità a virgola mobile a precisione singola) fino a 240 MHz, con intervallo di temperatura ambiente esteso opzionale fino a 125 °C.

Nel progetto utilizziamo solo il core Arm Cortex-M4.

Prestazione

FCPU a 480 MHz sul Cortex-M7 e 240 MHz sul Cortex-M4, 3224 CoreMark / 1327 DMIPS in esecuzione dalla memoria Flash, con stati di attesa 0 grazie alla sua cache L1.

Cache L1 (16 Kbyte di I-cache + 16 Kbyte di D-cache) che migliora le prestazioni di esecuzione da memorie esterne.

Sicurezza

Gli MCU STM32H755 includono le seguenti funzionalità di sicurezza aggiuntive:

Accelerazione hardware crypto/hash

Servizi di sicurezza incorporati SFI (Secure Firmware Install) per autenticare e proteggere gli IP software durante l'esecuzione della programmazione iniziale.

Aggiornamento del firmware sicuro all'avvio protetto (SBSFU)

Efficienza energetica

L'architettura multi-power domain consente di impostare diversi domini di potenza in modalità a basso consumo per ottimizzare l'efficienza energetica.

SMPS integrato per ridurre la tensione di alimentazione. Può anche essere utilizzato per fornire circuiti esterni e può anche essere combinato con l'LDO per casi d'uso specifici.

Regolatore USB per alimentare lo strato fisico incorporato (PHY).

145 µ/MHz @VDD tipici = 3,3 V e 25 °C in modalità Run (periferiche spente) e SMPS

2,43 µA tipico in modalità Standby (modalità a basso consumo)

Periferiche integrate

Fino a 35 interfacce di comunicazione tra cui FD-CAN, USB 2.0 ad alta velocità/full-speed, Ethernet MAC, interfaccia telecamera

Gamma di memoria facilmente estendibile utilizzando il controller di memoria flessibile con un'interfaccia parallela a 32 bit o l'interfaccia di memoria Flash seriale Quad-SPI dual-mode

Analogico: DAC a 12 bit, ADC veloci a 16 bit

Timer multipli a 16 e 32 bit in esecuzione fino a 480 MHz sul timer ad alta risoluzione a 16 bit

Motori

Sui moderni multicotteri (droni) sono attualmente utilizzati due tipi di motori elettrici: brushed e brushless. I droni brushless sono principalmente dotati di droni costosi e professionali. Solo le opzioni e i giocattoli economici sono dotati di motori da collezione. Maggiori dettagli sulle loro differenze possono essere trovati qui

- Corrente massima - questa caratteristica mostra quale corrente massima può sopportare l'avvolgimento del motore in un breve periodo di tempo. Se questo tempo viene superato, il guasto del motore è inevitabile. Questo parametro influenza anche la scelta di ESC.
- Tensione massima - oltre alla corrente massima, indica quanta tensione può essere applicata all'avvolgimento per un breve periodo di tempo.
- KV è il numero di giri del motore per volt. Poiché questo indicatore dipende direttamente dal carico sull'albero motore, è indicato nel caso in cui non vi sia carico.
- Resistenza - L'efficienza del motore dipende dalla resistenza. Pertanto, minore è la resistenza, meglio è.

Motori Per questo progetto sono stati utilizzati 4 motori brushless Turnigy D3536/9 910KV. La scelta di utilizzare motori brushless, tipica nelle applicazioni di aeromodellismo, è dovuta sostanzialmente a motivi di sicurezza (dovuta alla mancanza di meccanismo a spazzole), maggior durata e migliori prestazioni in termini di velocità massima di rotazione e minor peso. I motori da noi scelti sono brushless a tre fasi, “outrunner” (a cassa rotante), sensor less ovvero: Sono controllati da 3 fasi che commutano ciclicamente, pilotate a un banco di transistor gestiti da microcontrollore (ESC).

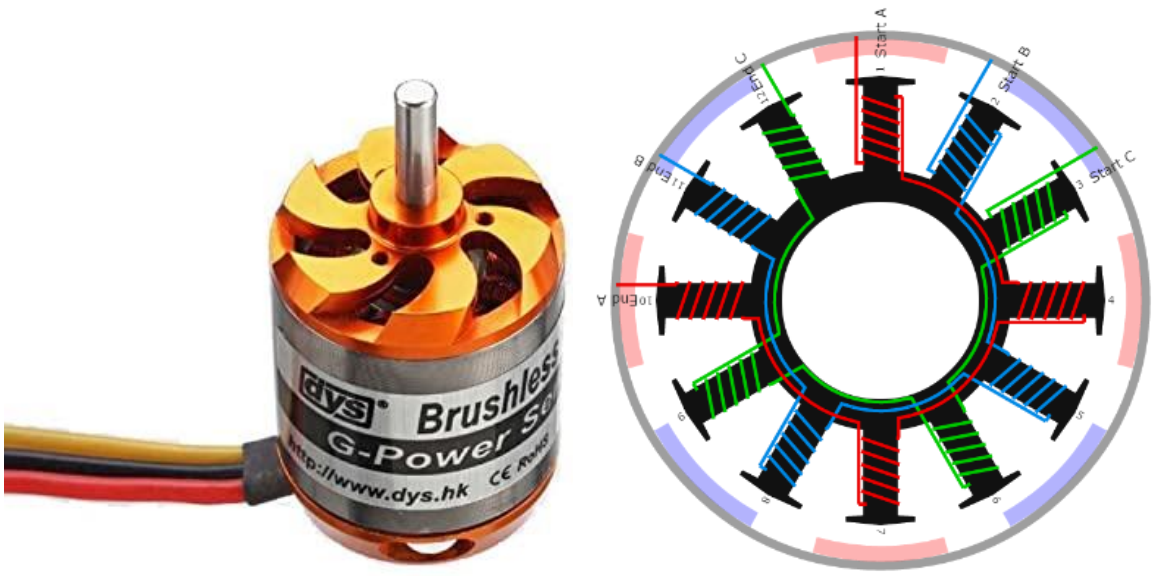


Fig 23: Motore brushless Turnigy D3536/9 910KV

Hanno lo statore fisso, posto internamente insieme agli avvolgimenti delle fasi, ed il rotore esterno, costituito da magneti permanenti, che ruota attorno allo statore. Il vantaggio di utilizzare questa configurazione è l'elevata coppia sostenibile. Non sono dotati di sensoristica che ne determini la posizione angolare (come encoder o resolver), bensì utilizzano un feedback proveniente dalla f.e.m. generatasi nel motore per determinare quando deve avvenire la commutazione. Le caratteristiche tecniche dei motori da noi utilizzati sono le seguenti:

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

Esc

Definizione

I controller di velocità sono chiamati Electric Speed Controller o ESC in breve nella comunità di lingua inglese. Il compito principale dell'ESC è trasferire energia dalla batteria al motore brushless. La necessità del loro utilizzo è nata a causa di alcune caratteristiche del motore BC. In breve, la batteria fornisce corrente continua, mentre il motore brushless accetta corrente alternata trifase.

Modalità d'uso

Il termine ESC deriva dall'acronimo inglese Electronic Speed Control (Controllore Elettronico di Velocità) ed è appunto quell'insieme di microcontrollore e banco di transistor atto a controllare le commutazioni delle fasi che permettono di regolare le performance del motore. Allo stesso tempo, essi provvedono all'alimentazione sia del motore che degli apparati elettronici. Fu introdotto per la prima volta nel campo del radio-modellismo e, proprio per questo, fu stato realizzato appositamente per funzionare in maniera simile ad un servomotore. Un servomotore non è altro che un motore che, ricevendo un comando da una ricevente radio, è in grado di muoversi in determinate posizioni o raggiungere determinate velocità imposte dal trasmettitore.

Tensione di alimentazione	14,6V
Tensione minima di funzionamento	5,6V
Corrente di lavoro	30A
BEC	5v 2A ver.3,1
Peso	42.5g

Fig 24: ESC Turnigy plush 30a

Eliche

Ai motori sono state collegate assialmente delle eliche bipala Slow Fly Electric Prop 9057SF da 9x4.7 pollici capaci di fornire una spinta, a piena velocità, di circa 0.632Kg ciascuna, come indicato dal venditore.

E' stato calcolato, tramite una semplice formula:

$$S_{min} = \frac{Peso\ quadricottero}{4}$$

che la spinta fornita è sufficiente a far volare senza problemi il velivolo.

Fig 25: Eliche da 9x4.7 pollici

Batteria

Caratteristiche Nel nostro progetto è stata utilizzata per l'alimentazione una batteria Turnigy A-Spec 2600 mAh, 4S 25-50 Li-Po Pack. Di seguito sono elencate le caratteristiche:

Capacità	2600 mAh
Voltaggio	4S1P – 4 Celle – 14.8V
Scarica	25C costanti / 65C picco
Dimensioni	106x35x24 mm
Peso	210g

Per una distribuzione rapida della tensione fra i motori vi è stata collegata una scheda di distribuzione chiamata Power Distributor, il cui compito è di mantenere in parallelo le linee di tensione degli ESC.

Fig 27: Batteria Turnigy A-Spec 2600 mAh

Telaio

Il quadricottero da noi costruito monta un telaio del tipo X666 Glass Fiber Quadcopter Frame, in alluminio e fibra di vetro, largo 666mm e dal peso di 415g.

Fig 28: Render per il telaio del drone

Esso è già predisposto per il fissaggio dei motori alle sue estremità, mentre è stato necessario effettuare delle forature per poter fissare la base centrale in plexiglass e adattare ogni altro componente a bordo con opportuni supporti realizzati ad hoc.

La scelta del telaio è stata fatta in funzione principalmente del peso; infatti in questa applicazione è stato necessario cercare di ridurre al minimo tutti i pesi, così da poter avere un certo margine nella scelta delle eliche. Tanto minore è la massa, tanto minore è la forza peso e pertanto, al fine di poter garantire il volo a parità di spinta impressa dalle eliche, tanto minore è la forza da vincere in una condizione di volo livellato. Il telaio scelto, in alluminio anodizzato, si presenta a forma di croce; all'estremità dei segmenti della croce sono installati gli attuatori, lungo i bracci gli ESC e verso il centro si concentra il sistema di alimentazione e l'elettronica di controllo.

Il materiale del telaio varia in base al rapporto peso/motorizzazione/capacità di carico.

IMU

Che cos'è un'unità IMU?

Una unità IMU è una scheda che consente di misurare accelerazione, velocità angolare e campo magnetico. Ogni misura viene fatta su un sistema a 3 assi. Quindi avremo 3 dati per l'accelerazione (x,y,z), 3 dati per la velocità angolare (x,y,z) e 3 dati per il campo magnetico (x,y,z). In tutto avremo 9 variabili che esprimono la posizione nello spazio della nostra scheda. Grazie a queste variabili possiamo stabilire se un oggetto è in movimento, se sta traslando, ruotando o se è inclinato rispetto al pavimento.

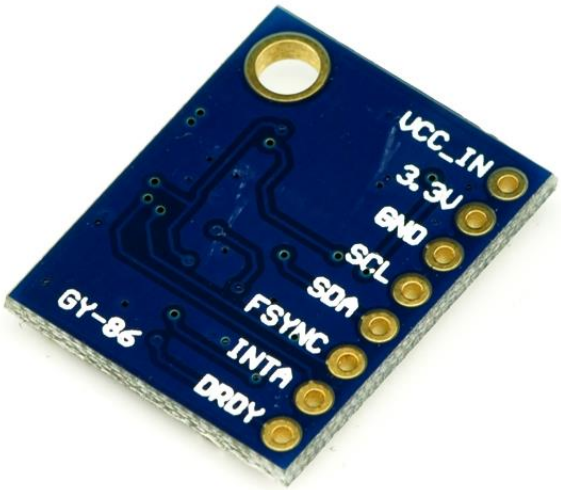


Fig 29: IMU 6050

Come funziona una IMU?

L'IMU può misurare una varietà di fattori, tra cui velocità, direzione, accelerazione, forza specifica, tasso angolare e (in presenza di un magnetometro), campi magnetici che circondano il dispositivo.
Ogni strumento di una IMU è usato per catturare diversi tipi di dati:
– Accelerometro: misura la velocità e l'accelerazione
– Giroscopio: misura la rotazione e il tasso di rotazione
– Magnetometro: stabilisce la direzione cardinale (intestazione direzionale)
L'IMU combina l'input da diversi tipi di sensori per produrre con precisione il movimento.

Posizionamento

L'IMU ha una posizione ben precisa. Il circuitino deve essere posizionato al centro della lastra di plexiglass sopra all'apposito gommino adesivo, questo perchè il sensore IMU deve essere posizionato perfettamente nel centro di massa del drone. Inoltre, il sensore deve essere direzionato nel verso mostrato in Fig 30 dato che il sistema di riferimento dell'IMU deve coincidere con quello del Drone.

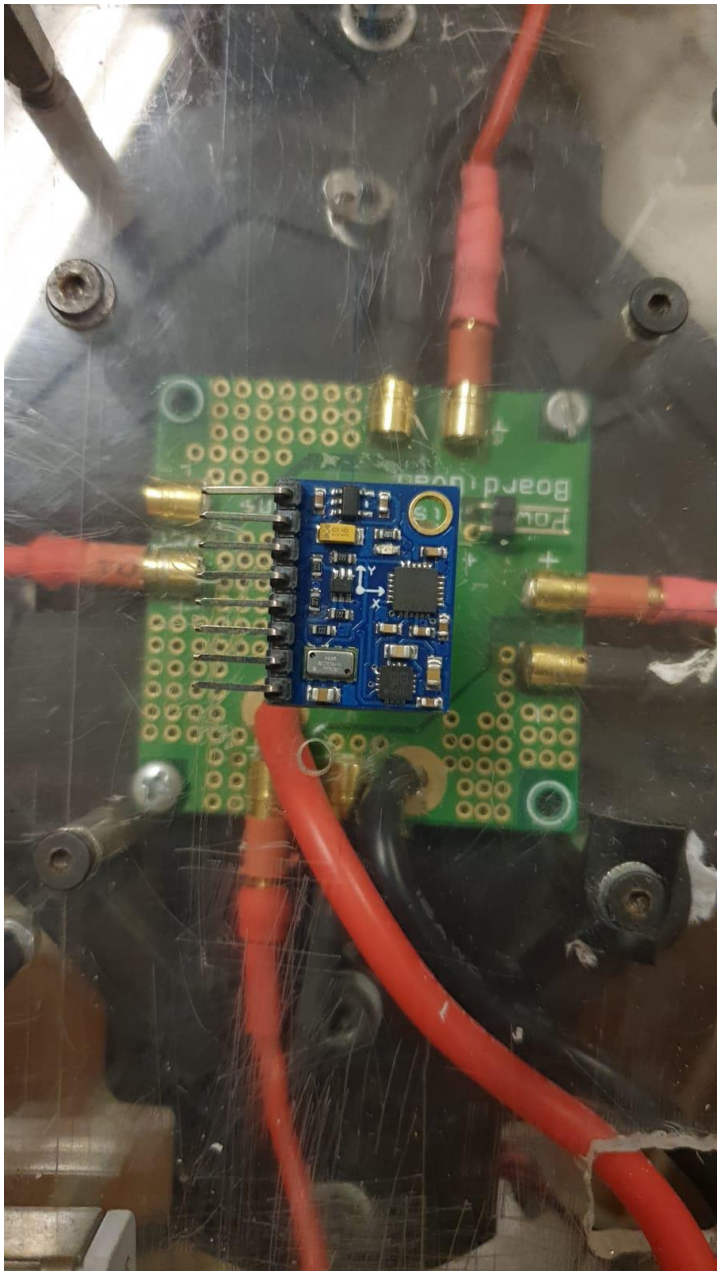


Fig 30: Posizionamento dell'IMU sul drone

Display

Abbiamo programmato un display del tipo oled per stampare i valori del Pitch,Roll e Yaw del quadrotor.

Specifica

Modalità di comunicazione: I2C

Display in scala di grigi: 16 sfumature di grigio.

Supporta sia il display a colori normale che quello inverso.

Tensione di esercizio 3.3/5 V

Matrice di punti 128x128

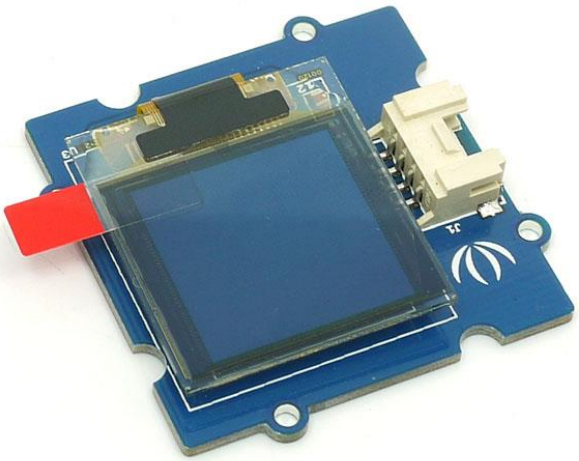


Fig 31: Display (Grove-Oled 1.12 V2)

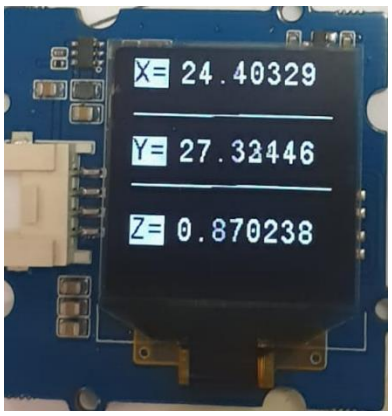


Fig 32: Esempio dati riportati per i tre assi

CONNETTORI

Periferica esterna	Connettore	PIN	Descrizione
IMU	CN8	+3V3	3.3V
	CN8	GND	GND
	CN10	PB10	SCL
	CN10	PB11	SDA
Display	CN8	+3V3	3.3V
	CN8	GND	GND
	CN10	PB6	SCL
	CN10	PB7	SDA
PWM1	CN7	PA6	pwm
PWM2	CN7	PC7	pwm
PWM3	CN8	PC8	pwm
PWM4	CN8	PC9	pwm

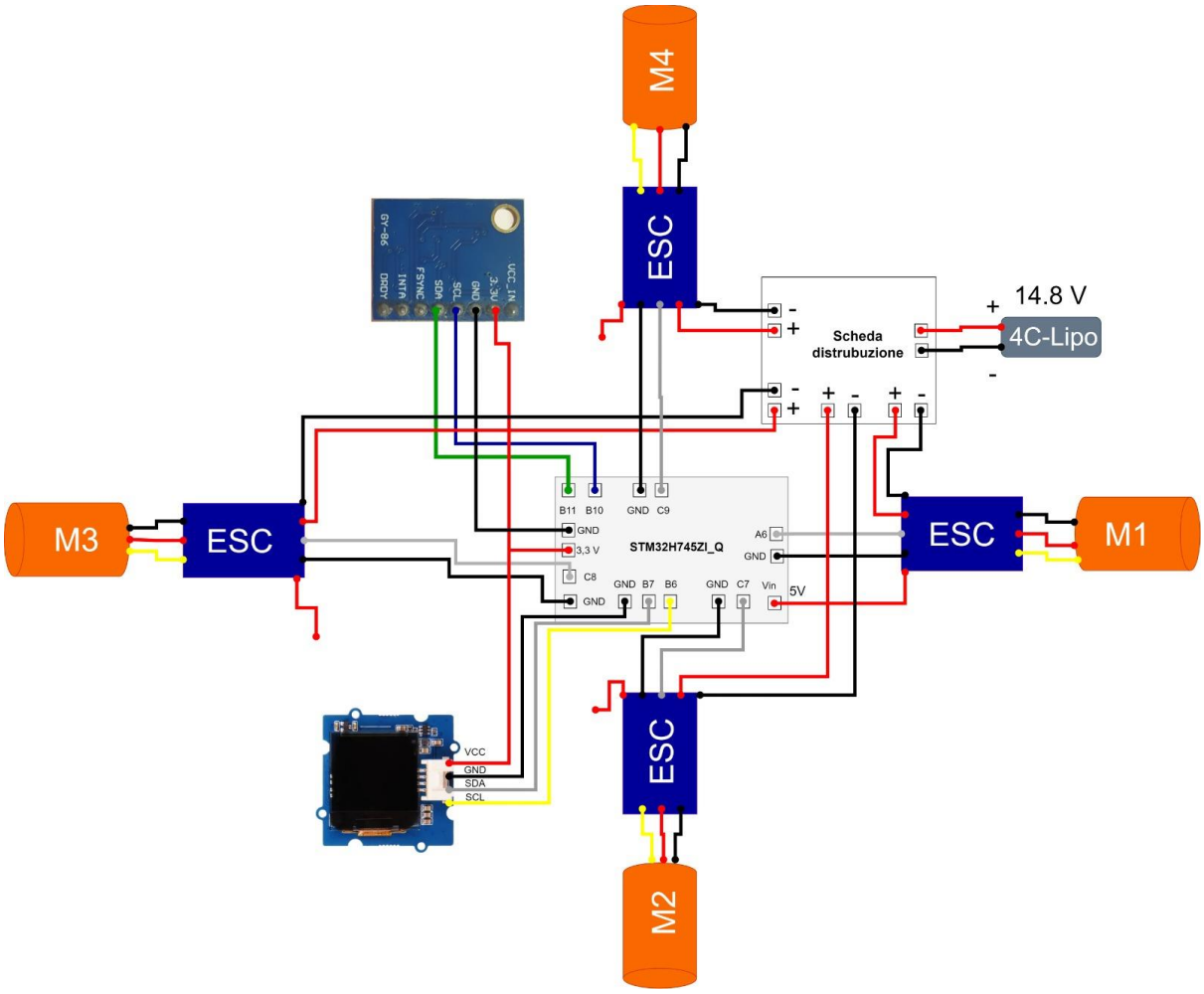


Fig 33: Schema di collegamenti

PROVA MOTORI NEL LABORATORIO

Motori:

i motori brushless funzionano tramite un segnale pwm di frequenza 50 Hz, cambiando il duty cycle saremo in grado di controllare la velocità.

per mettere in rotazione questo tipo di motori prima deve essere armato.

L'armamento è una procedura di sicurezza che consiste nel mandare un certo segnale (pwm) Con duty cycle di 4.75 % quindi nel nostro caso siccome abbiamo il periodo vale 200ms allora

$$4.75 = \frac{\text{up} * 100}{\text{MOTOR. PWM. SIGNAL. PERIOD. UP}}$$

$$\text{up} = \frac{200 * 4.75}{100} = 9.5\ ms$$

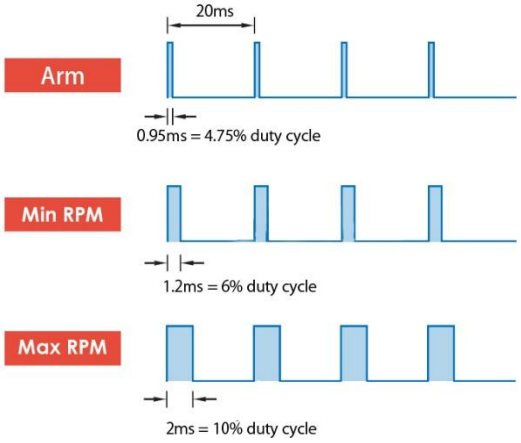
Una volta avvenuta con successo la fase di armamento l'ESC emetterà' due BEEP veloci poi Un BEEP lungo.

Il Range di controllo della velocità è compreso fra 6% (velocità minima) e 10% (velocità massima) di duty cycle. Questi valori però si possono cambiare con la calibrazione del Esc.

La calibrazione è una procedura che consiste nel definire il range del minimo e del massimo valore di duty cycle cioè la minima e la massima velocità del motore.

I passaggi per la calibrazione sono i seguenti:

1) Si manda un segnale PWM del massimo duty cycle desiderato (nel nostro caso 10%).
2) Si connette la batteria all'ESC poi si aspetta 2 secondi
3) L'ESC emetterà due BEEP veloci per segnalare che il massimo duty cycle è stato confermato.
4) Si manda un segnale PWM del minimo duty cycle desiderato (nel nostro caso 6%).
5) L'ESC emetterà due BEEP lungo per segnalare che il minimo duty cycle è stato confermato.



Dopo aver fatto la calibrazione abbiamo mandato il segnale pwm variando il duty cycle in modo tale per testare il motore.

SOFTWARE

Il nostro programma è diviso principalmente in due parti. La prima parte consiste nel leggere i dati dal sensore (IMU) per ottenere la posizione attuale del quadrotor ovvero i vari angoli (pitch, roll, yaw). Invece la seconda parte riguarda l'elaborazione dei dati letti dal sensore tramite il controllo PID ed il funzionamento dei motori per cercare di stabilizzare il drone.

IMU

La trasmissione dei dati avviene tramite il protocollo I2C.

Per leggere la velocità angolare (Gyroscope) e l'accelerazione degli assi, prima dobbiamo inizializzare l'IMU

```
uint8_t MPU6050_Init(I2C_HandleTypeDef *I2Cx) {
    uint8_t check;
    uint8_t Data;

    // check device ID WHO_AM_I

    HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, WHO_AM_I_REG, 1, &check, 1, i2c_timeout);

    if (check == 104)  // 0x68 will be returned by the sensor if everything goes well
    {
        // power management register 0X6B we should write all 0's to wake the sensor up
        Data = 0;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1, i2c_timeout);

        // Set DATA RATE of 1KHz by writing SMPLRT_DIV register
        Data = 0x07;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, SMPLRT_DIV_REG, 1, &Data, 1, i2c_timeout);

        // Set accelerometer configuration in ACCEL_CONFIG Register
        // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 -> ± 2g
        Data = 0x00;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, i2c_timeout);

        // Set Gyroscopic configuration in GYRO_CONFIG Register
        // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 -> ± 250 °/s
        Data = 0x00;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, GYRO_CONFIG_REG, 1, &Data, 1, i2c_timeout);
        return 0;
    }
    return 1;
}
```

Nella prima parte del codice verifichiamo che l'IMU sia connesso correttamente dopo di che si avvia la procedura di inizializzazione scrivendo nei vari registri. Abbiamo settato il fattore di sensibilità del gyroscope di ±250°/s quindi facendo riferimento al manuale andiamo a scrivere il valore 0 all'indirizzo 0x1B.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
GYROSCOPE SENSITIVITY					
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s

invece per quanto riguarda l'accelerazione abbiamo settato il fattore di sensibilità di 2g, facendo sempre riferimento al manuale andiamo a scrivere il valore 0 all'indirizzo 0x1C.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ACCELEROMETER SENSITIVITY					
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*

Una volta inizializzato, possiamo leggere la velocità angolare e l'accelerazione

```
void MPU6050_Read_Gyro(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct) {
    uint8_t Rec_Data[6];

    // Read 6 BYTES of data starting from GYRO_XOUT_H register

    HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, GYRO_XOUT_H_REG, 1, Rec_Data, 6, i2c_timeout);

    DataStruct->Gyro_X_RAW = (int16_t) (Rec_Data[0] << 8 | Rec_Data[1]);
    DataStruct->Gyro_Y_RAW = (int16_t) (Rec_Data[2] << 8 | Rec_Data[3]);
    DataStruct->Gyro_Z_RAW = (int16_t) (Rec_Data[4] << 8 | Rec_Data[5]);

    /*** convert the RAW values into dps (°/s)
         we have to divide according to the Full scale value set in FS_SEL
         I have configured FS_SEL = 0. So I am dividing by 131.0
         for more details check GYRO_CONFIG Register              ****/

    DataStruct->Gx = DataStruct->Gyro_X_RAW / 131.0;
    DataStruct->Gy = DataStruct->Gyro_Y_RAW / 131.0;
    DataStruct->Gz = DataStruct->Gyro_Z_RAW / 131.0;
}
```

Nella parte riportata sopra del codice vengono letti i valori **RAW** della velocità angolare dei tre assi dall'indirizzo **0x43**, ogni asse è composto da 16 bit quindi di totale 6 byte. Dopo di che questi valori **RAW** vengono divisi per **131.0** per avere in fine i valori effettivi della velocità angolare. Il numero su cui vengono divisi i valori **RAW** dipende dal coefficiente di sensibilità scelto prima (nel nostro caso **FS_SEL=0**).

Gyroscope ADC Word Length					
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)

Per quanto riguarda la lettura dell'accelerazione, si segue la stessa procedura di prima.

```
void MPU6050_Read_Accel(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct) {
    uint8_t Rec_Data[6];

    // Read 6 BYTES of data starting from ACCEL_XOUT_H register

    HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data, 6, i2c_timeout);

    DataStruct->Accel_X_RAW = (int16_t) (Rec_Data[0] << 8 | Rec_Data[1]);
    DataStruct->Accel_Y_RAW = (int16_t) (Rec_Data[2] << 8 | Rec_Data[3]);
    DataStruct->Accel_Z_RAW = (int16_t) (Rec_Data[4] << 8 | Rec_Data[5]);

    /*** convert the RAW values into acceleration in 'g'
         we have to divide according to the Full scale value set in FS_SEL
         I have configured FS_SEL = 0. So I am dividing by 16384.0
         for more details check ACCEL_CONFIG Register              ****/

    DataStruct->Ax = DataStruct->Accel_X_RAW / 16384.0;
    DataStruct->Ay = DataStruct->Accel_Y_RAW / 16384.0;
    DataStruct->Az = DataStruct->Accel_Z_RAW / 16384.0;
}
```

Si divide per **16384.0** come riportato nel manuale.

ADC Word Length	Output in two's complement format				
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*
	AFS_SEL=1		8,192		LSB/*g*
	AFS_SEL=2		4,096		LSB/*g*
	AFS_SEL=3		2,048		LSB/*g*

A questo punto abbiamo ottenuto i valori effettivi della velocità angolare in **gradi al secondo** e dell'accelerazione in **g** però per stabilizzare il quadratore ovvero per avere tutti gli angoli a zero gradi, ci serve ricavare gli angoli.

Gli angoli del Pitch e Roll si possono ricavare integrando la velocità angolare nel tempo **dt** (periodo di campionamento) sommando anche il valore del Pitch e Roll precedentemente calcolato.

Il **dt** sarà il tempo di una lettura dall'IMU e quella successiva nel nostro caso **0.005** secondi.

```
/* velocity: angular velocity
    dt: time of integration
    return newangle: the new angle */
float getangles(float velocity, float dt){
    float newangle=0;
    newangle =velocity*dt;

    return newangle;
}
```

$$\dot{\theta} = \frac{d\theta}{dt}$$

```
angleX= angleX+getangles(MPU6050.Gx,dt);
angleY= angleY+getangles(MPU6050.Gy,dt);
```

$$\boldsymbol{\theta}(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_0^t \dot{\theta}(t)T_s$$

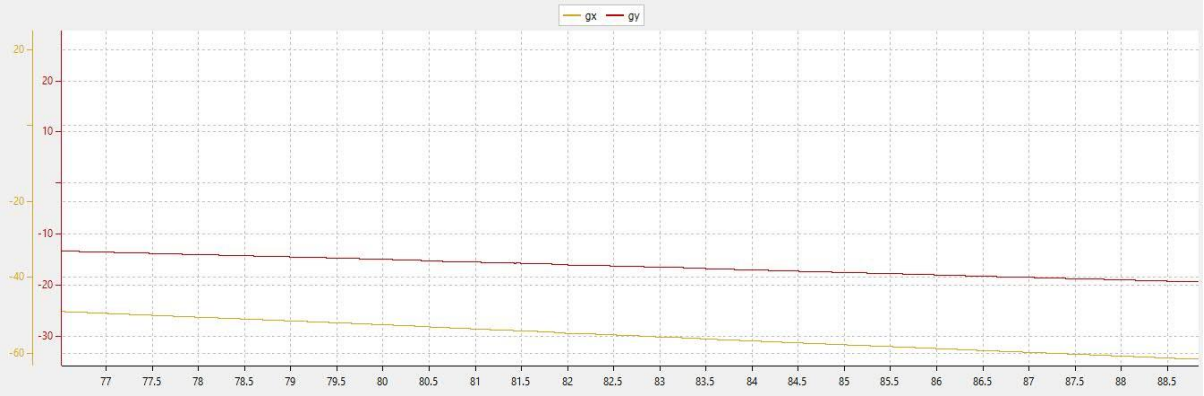
Ora siamo riusciti ad ottenere gli angoli del quadrotore ma abbiamo notato che, firmando quadrotor nella posizione di equilibrio, a lungo andare gli angoli iniziano a deviare. Questo perché non avendo la possibilità di fare un integrale perfettamente continuo utilizzando i sistemi digitali, facciamo l'approssimazione.

L'approssimazione consiste nel prendere la somma di un numero finito di campioni prelevati ad un intervallo costante dt.

Naturalmente questa approssimazione introdurrà errori.

Quando i dati del giroscopio cambiano più velocemente della frequenza di campionamento, non riusciamo a rilevarli e l'approssimazione integrale non sarà corretta.

Questo errore è chiamato **Gyroscope drift**.



Per risolvere questo problema, dobbiamo introdurre prima come si può ricavare gli angoli Pitch e Roll dai valori di accelerazione tramite le seguenti formule di Eulero;

$$tan\phi_{xyz} = \left(\frac{G_{py}}{G_{pz}}\right)$$

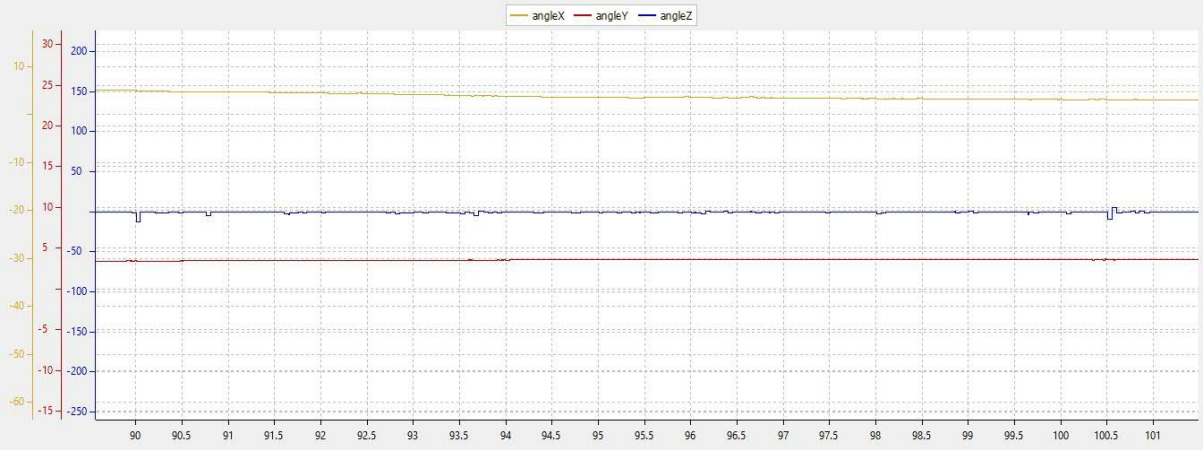
$$tan\theta_{xyz} = \left(\frac{-G_{px}}{G_{py}sin\phi + G_{pz}cos\phi}\right) = \frac{-G_{px}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}}$$

In codice si scrivono in questo modo, viene usato l'arcotangente per ricavare gli angoli in radianti poi vengono moltiplicati per 180/π(RAD_TO_DEG).

```
accel_anglesX = atan2(MPU6050.Ay, MPU6050.Az) * RAD_TO_DEG;

accel_anglesY = atan2(-1 * MPU6050.Ax, sqrt(MPU6050.Ay * MPU6050.Ay +  MPU6050.Az *  MPU6050.Az)) * RAD_TO_DEG;
```

L'asse Z deve essere calibrato a 1g_force.

Ora quando mettiamo il quadrotor nella posizione di equilibrio, notiamo che i valori degli angoli stanno sempre attorno al valore zero e non deviano mai. Anche se qualsiasi movimento del quadrotor causa dei picchi nel diagramma, questo perché l'accelerometro è molto suscettibile alle vibrazioni.



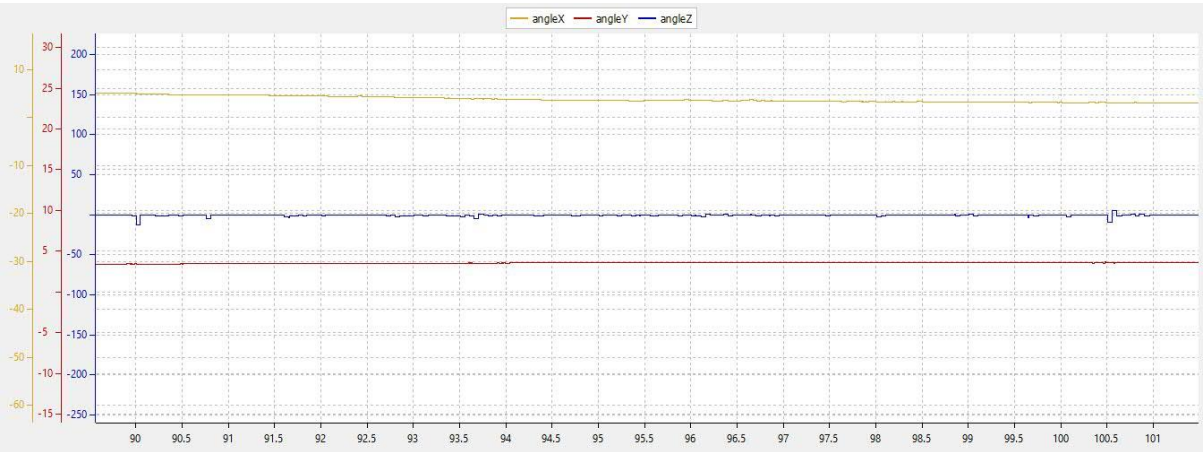
Quindi per risolvere il problema che abbiamo introdotto prima, dobbiamo combinare i due approcci ovvero mettere insieme gli angoli che abbiamo ricavato dal giroscopio e quelli ricavati dall'accelerometro.

Il giroscopio è molto adatto per breve tempo perché risponde velocemente. Invece l'accelerometro è più adatto per lungo tempo in quanto torna sempre a zero e non devia.

Per combinare questi angoli usiamo un filtro chiamato **complementary filter** che consiste nel definire due numeri costanti la loro somma fa uno. Il primo numero è la parte del giroscopio invece il secondo è la parte dell'accelerometro.

```
angleX = GYRO_PART * (angleX + (MPU6050.Gx * dt)) + ACC_PART * accel_anglesX;

angleY = GYRO_PART * (angleY + (MPU6050.Gy * dt)) + ACC_PART * accel_anglesY;
```

Abbiamo scelto **GYRO_PART = 0.995** e **ACC_PART=0.005**.



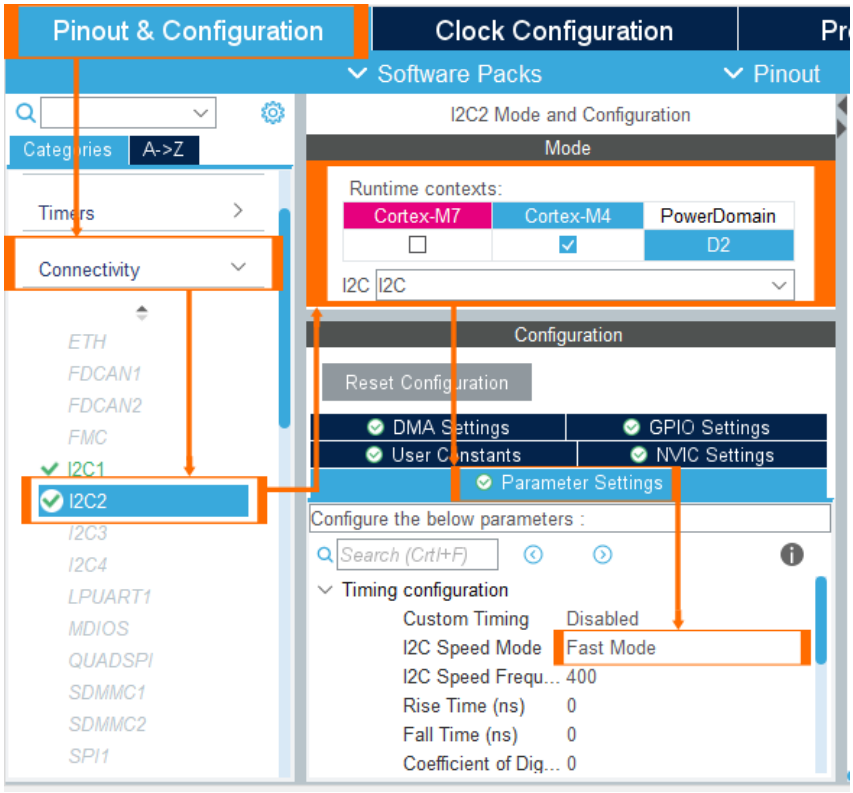
Nel nostro progetto abbiamo usato il filtro complementare però ci sono anche altri tipi di filtri.

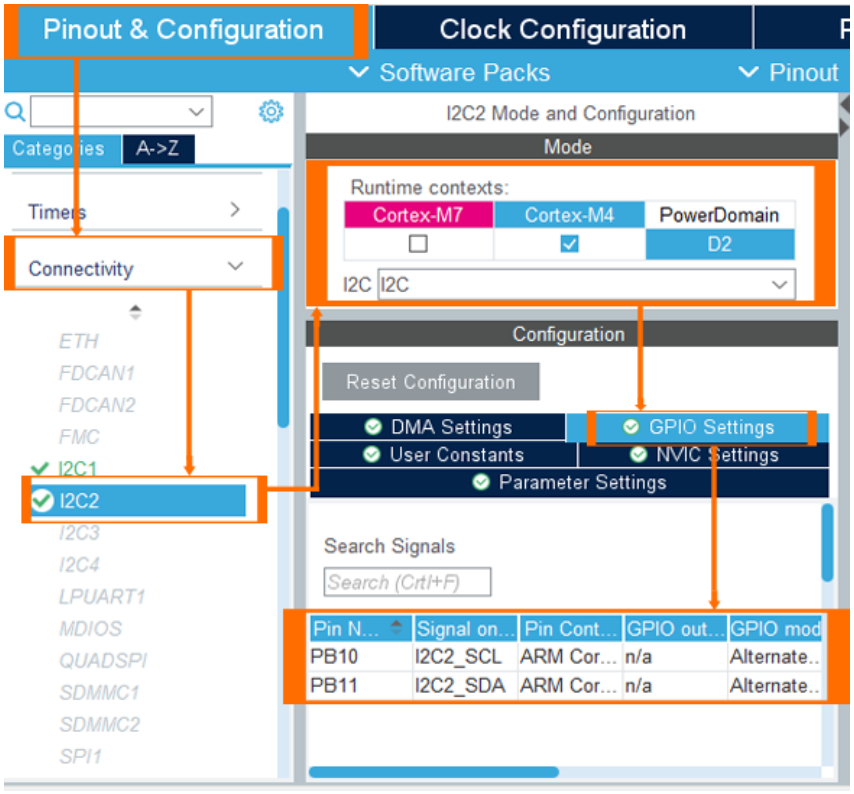
Uno di questi filtri è Kalman filter che è uno strumento per stimare lo stato di un sistema dinamico lineare perturbato da rumore, sulla base di misure (o osservazioni) linearmente dipendenti dallo stato e corrotte da rumore.

Il filtro può essere implementato tramite il seguente codice:

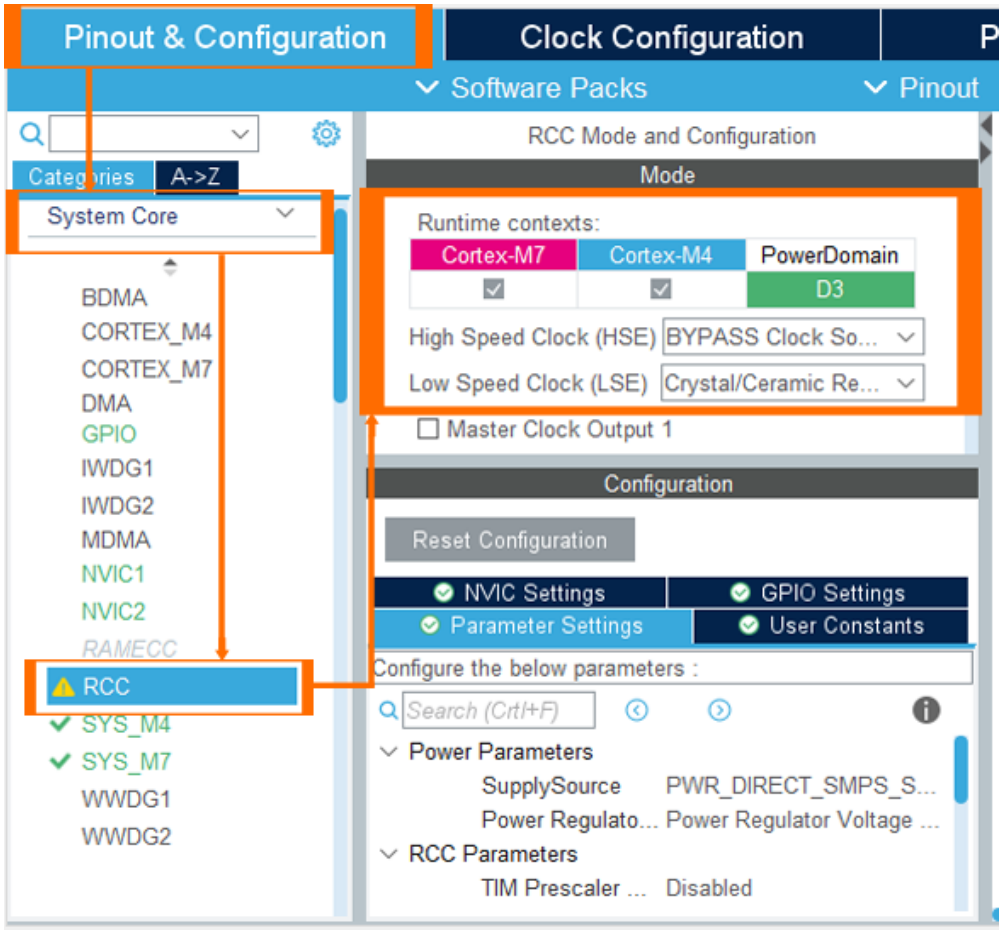
```
double Kalman_getAngle(Kalman_t *Kalman, double newAngle, double newRate, double dt) {
    double rate = newRate - Kalman->bias;
    Kalman->angle += dt * rate;

    Kalman->P[0][0] += dt * (dt * Kalman->P[1][1] - Kalman->P[0][1] - Kalman->P[1][0] + Kalman->Q_angle);
    Kalman->P[0][1] -= dt * Kalman->P[1][1];
    Kalman->P[1][0] -= dt * Kalman->P[1][1];
    Kalman->P[1][1] += Kalman->Q_bias * dt;

    double S = Kalman->P[0][0] + Kalman->R_measure;
    double K[2];
    K[0] = Kalman->P[0][0] / S;
    K[1] = Kalman->P[1][0] / S;

    double y = newAngle - Kalman->angle;
    Kalman->angle += K[0] * y;
    Kalman->bias += K[1] * y;

    double P00_temp = Kalman->P[0][0];
    double P01_temp = Kalman->P[0][1];

    Kalman->P[0][0] -= K[0] * P00_temp;
    Kalman->P[0][1] -= K[0] * P01_temp;
    Kalman->P[1][0] -= K[1] * P00_temp;
    Kalman->P[1][1] -= K[1] * P01_temp;

    return Kalman->angle;
};
```

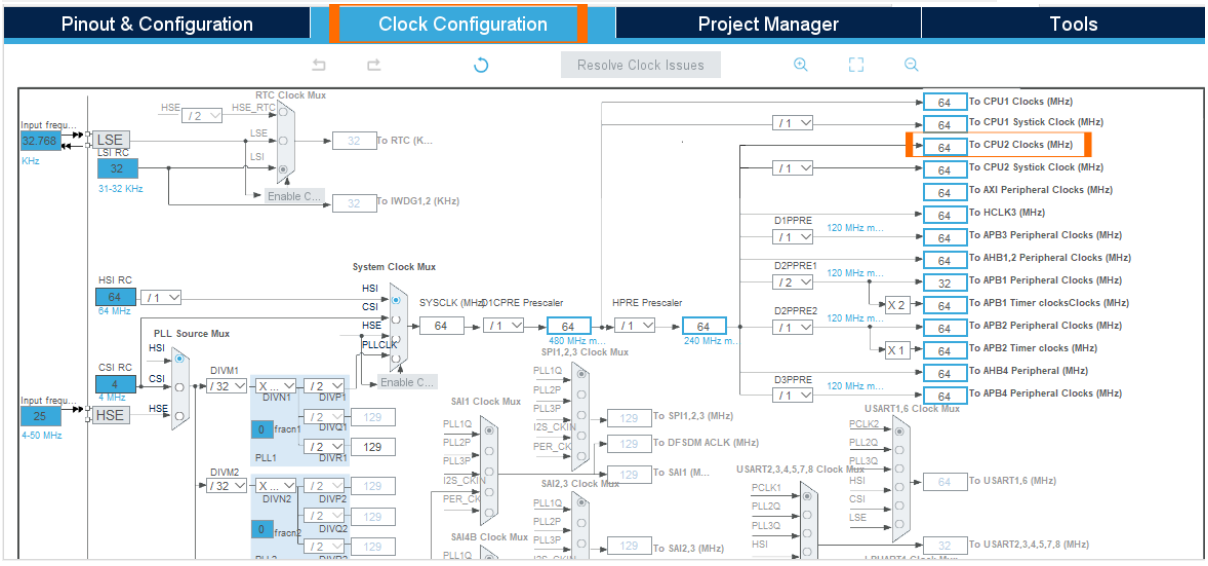
Procedura settaggio del protocollo I2C e del timer tramite CubeIDE
Protocollo I2C per l'IMU:




Pinout & Configuration
Clock Configuration
Software Packs
Pinout
Categories
A->Z
Timers
Connectivity
ETH
FDCAN1
FDCAN2
FMC
I2C1
I2C2
I2C3
I2C4
LPUART1
MDIOS
QUADSPI
SDMMC1
SDMMC2
SPI1
I2C2 Mode and Configuration
Mode
Runtime contexts:
Cortex-M7
Cortex-M4
PowerDomain
D2
I2C
I2C
Configuration
Reset Configuration
DMA Settings
GPIO Settings
User Constants
NVIC Settings
Parameter Settings
Search Signals
Search (Crtl+F)
Pin N...
Signal on...
Pin Cont...
GPIO out...
GPIO mod
PB10
I2C2_SCL
ARM Cor...
n/a
Alternate..
PB11
I2C2_SDA
ARM Cor...
n/a
Alternate..

Configurazione clock e RCC:

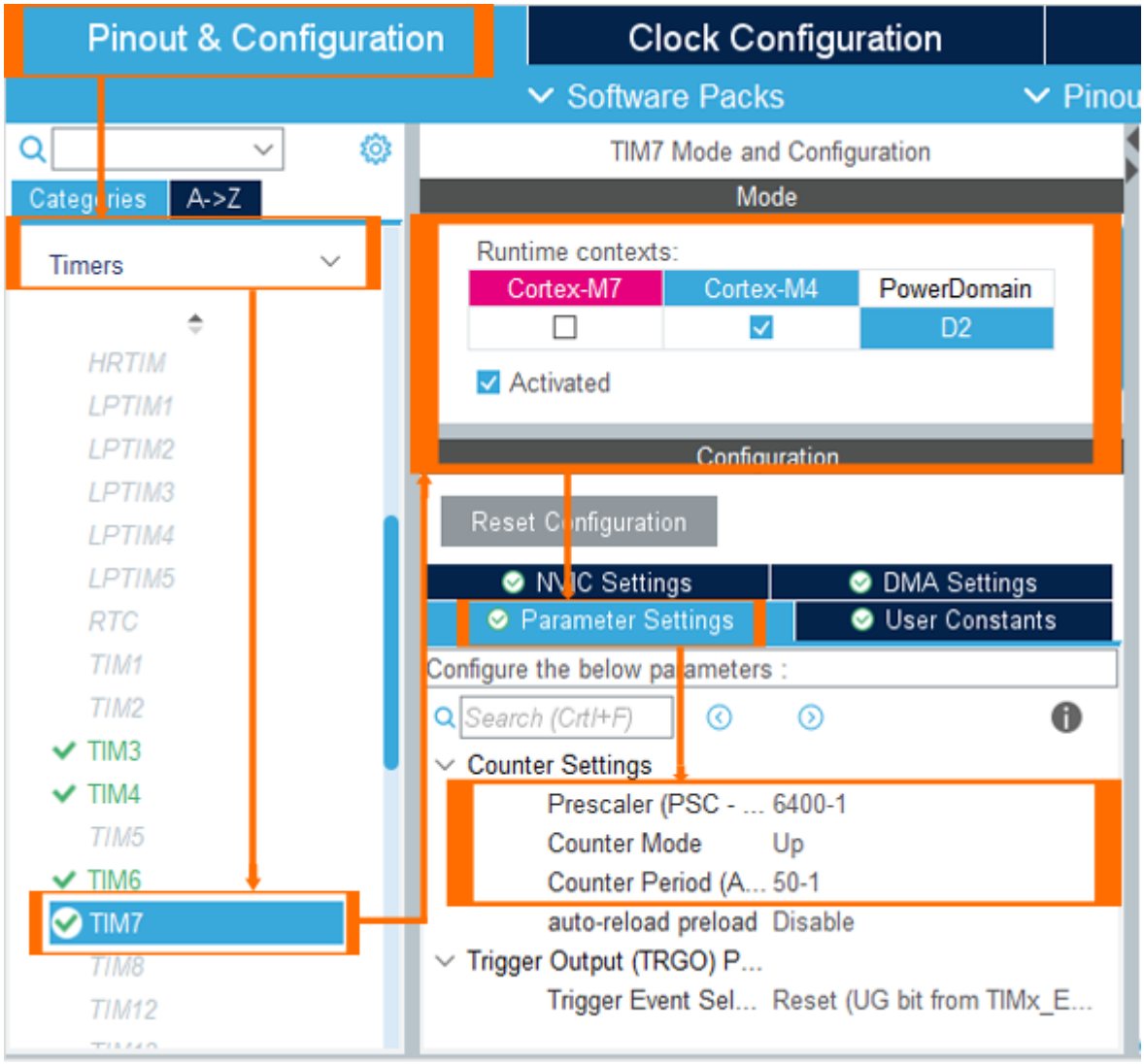


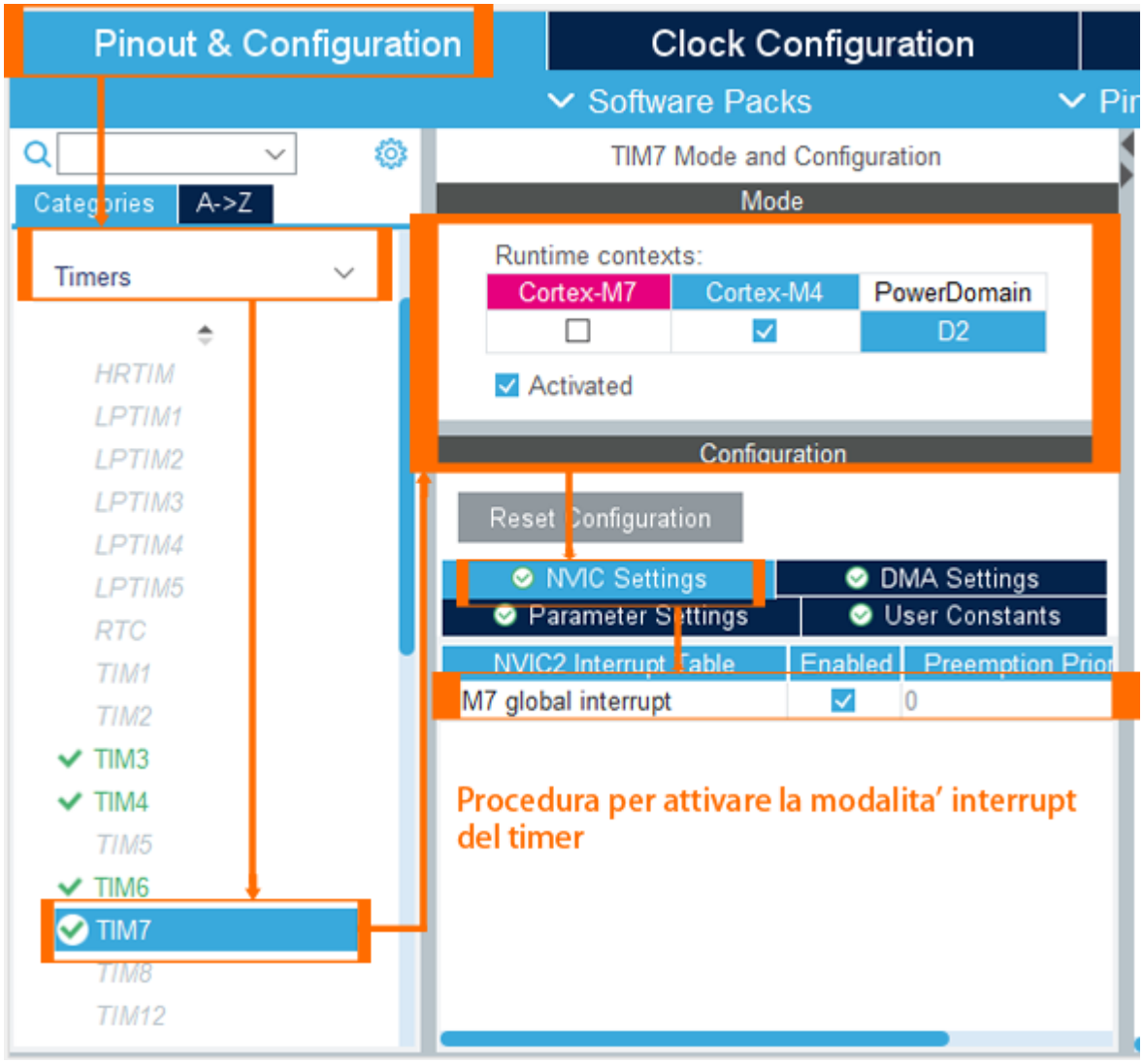


Configurazione Timer7 di frequenza **200 HZ** per la lettura dei dati dall'IMU:

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

$$UpdateEvent = \frac{64000000}{(6400 + 1)(50 + 1)} = 200HZ = 5ms$$





Per inizializzare il timer nella modalità interrupt si utilizza il seguente codice:

```
        HAL_TIM_Base_Start_IT(&htim7); //inizialize timer7 in interrupt mode

 //callback to detect the intrrupt event
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
 //verify if the intrrupt comes from timer7
    if(htim==&htim7){
        //Read data from IMU (gyro,accel)
        MPU6050_Read_All(&hi2c2, &MPU6050);

        // get angle x from acceleration
        accel_anglesX  = atan2(MPU6050.Ay, MPU6050.Az) * RAD_TO_DEG;

        // get angle y from acceleration
        accel_anglesY  = atan2(-1 * MPU6050.Ax, sqrt(MPU6050.Ay * MPU6050.Ay +  MPU6050.Az *  MPU6050.Az)) * RAD_TO_DEG;

        // complementary filter
        angleX = GYRO_PART * (angleX + (MPU6050.Gx * IMU_dt)) + ACC_PART * accel_anglesX;
        angleY = GYRO_PART * (angleY + (MPU6050.Gy * IMU_dt)) + ACC_PART * accel_anglesY;
        angleZ=MPU6050.Gz;

        }
```

Nella prima parte del codice si inizializza il timer7 in modalità interrupt poi con la funzione

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) è la callback con la quale si verifica l'interrupt. Invece la condizione **IF** verifica che l'interrupt sia avvenuto dal timer 7.

La funzione **MPU6050_Read_All** (&hi2c2, &MPU6050) legge i dati dall'IMU poi li memorizza nella struttura **MPU6050.**

ELABORAZIONE DATI E PID

Finora siamo in grado di simulare gli angoli Pitch, Roll e Yaw in modo preciso. Allora possiamo andare ad azionare i motori per stabilizzare il quadrotor però prima dobbiamo calcolare di quanto si deve aggiustare ogni motore la sua velocità per renderlo stabile. Questo ultimo avviene tramite il controllore **PID.**

Per realizzare un controllo **PID,** calcoliamo l'errore sottraendo l'angolo desiderato da quello corrente.

```
/*Compute all the working error variables*/
float error = setPoint - input;
```

Successivamente calcoliamo il termine derivativo e integrale:

-derivativo

```
float dInput = (error - conf->lastError) / conf->dt; //derivative of the exit value (input).
```

-Integrativo

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

ora moltiplichiamo ogni termine per il proprio coefficiente.

```
/*Compute PID Output*/
float output = (conf->kp * error) + (conf->ITerm) + (conf->kd * dInput); //sum of the outputs of the 3 PIDs
```

I coefficienti kp,kd e ki dipendono dalla struttura del quadrotor e vengono determinati tramite tecniche di sperimentazioni.

In totale abbiamo bisogno di tre controlli PID(Pitch, Roll e Yaw).

```
virtualInputs[1] = PID_Compute(currentpitch, desiredpitch, &pitchPid);
virtualInputs[2] = PID_Compute(currentroll,  desiredroll, &rollPid);
virtualInputs[3] = PID_Compute(currentyaw, desiredyaw,  &yawPid);
```

Questi ingressi virtuali non possono essere inviati ai motori direttamente. Abbiamo, quindi predisposto una funzione **SpeedCompute** per la traduzione software della matrice d'assetto introdotta nel modello matematico.

```
float* SpeedCompute (float virtualInputs [], float b, float l, float d)
{

    static float Speeds_quad[4];
    static float Speeds[4];

        Speeds_quad[0] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
        Speeds_quad[1] = (1/(4*b))*virtualInputs[0] - (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
        Speeds_quad[2] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[2] - (1/(4*d))*virtualInputs[3];
        Speeds_quad[3] = (1/(4*b))*virtualInputs[0] + (1/(2*l*b))*virtualInputs[1] + (1/(4*d))*virtualInputs[3];
```

Tuttavia, ciò non è sufficiente, poiché il modello matematico presuppone che la rotazione del motore possa essere invertita, non è possibile da attuare nella pratica, anche perché, nel passaggio da velocità al quadrato a velocità si incorrerebbe nel calcolo di una radice di un numero negativo. Per ovviare a tutto ciò abbiamo ampliato la funzione nel modo seguente:

```
const float abs_speedQuad_MAX = 1/(4*b) + 1/(2*l*b) + 1/(4*d); //maximum reachable squared speed

float speedQuad_min = -(1/(2*l*b)) - (1/(4*d)); //lower bound for squared speed 0.
float speedQuad_Max = 1/(4*b); //upper bound for squared speed 0.
Speeds[0] = sqrt(map(Speeds_quad[0], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));

speedQuad_min = -(1/(2*l*b)); //lower bound for squared speed 1.
speedQuad_Max = 1/(4*b) + 1/(4*d); //upper bound for squared speed 1.
Speeds[1] = sqrt(map(Speeds_quad[1], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));

speedQuad_min = -(1/(4*d)); //lower bound for squared speed 2.
speedQuad_Max = 1/(4*b) + 1/(2*l*b); //upper bound for squared speed 2.
Speeds[2] = sqrt(map(Speeds_quad[2], speedQuad_min, speedQuad_Max, 0, abs_speedQuad_MAX));

//Speeds_quad[3] can't be negative so we just need to make the square root
Speeds[3] = sqrt(Speeds_quad[3]);

return Speeds;
```

A ogni velocità al quadrato è stata applicata una trasformazione che sposta il suo dominio in uno esclusivamente positivo. Il nuovo intervallo [0, abs_speedQuad_MAX] è stato scelto considerando 0 come limite sinistro e il valore massimo, raggiungibile dal sistema di equazioni, derivante dalla matrice d'assetto come limite destro.

Infine, le velocità ottenute vengono mappate in un range compreso tra 6% e 10% di duty cycle (nel nostro caso tra 12 e 20) perché possano essere percepite dai motori con un duty consono:

```
PWM_1 = map(*(Speeds+0), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
PWM_2 = map(*(Speeds+1), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
PWM_3 = map(*(Speeds+2), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
PWM_4 = map(*(Speeds+3), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
```

Questi valori di PWM vengono mandati ai motori gia' armati(l'armamento avviene come e' stato speigato precendentmente) tramite il seguente codice:

```
MotorsArm(&htim3, TIM3);//arm motors
```

Funzione armamento motori che prende come parmaetri Timer3.

```
void MotorsArm(TIM_HandleTypeDef *htim,TIM_TypeDef *TIM){

      HAL_TIM_PWM_Start(htim,TIM_CHANNEL_1);
       TIM->CCR1=9.5;
       HAL_TIM_PWM_Start(htim,TIM_CHANNEL_2);
       TIM->CCR2=9.5;
       HAL_TIM_PWM_Start(htim,TIM_CHANNEL_3);
       TIM->CCR3=9.5;
       HAL_TIM_PWM_Start(htim,TIM_CHANNEL_4);
        TIM->CCR4=9.5;

  }
```

(armamento motori)

```
TIM3->CCR1=PWM_1;
TIM3->CCR2=PWM_2;
TIM3->CCR3=PWM_3;
TIM3->CCR4=PWM_4;
```

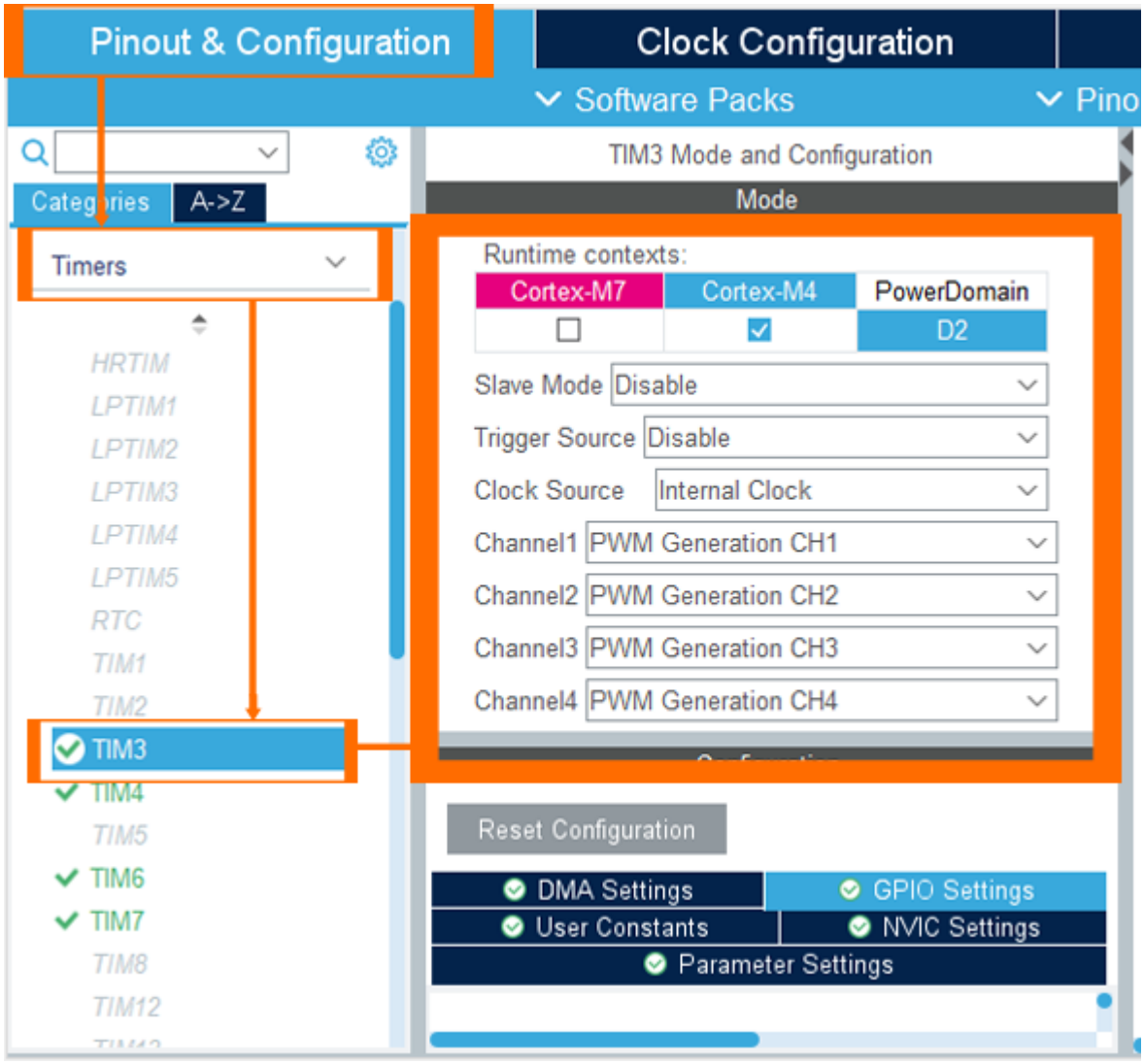
(mandare PWM ai motori)

Variando **CCR** si varia il duty cycle come spiegato nella formula precedentemente.

Procedura settaggio del PWM per i quattro motori e del timer per l'elaborazione dei PID:

PWM

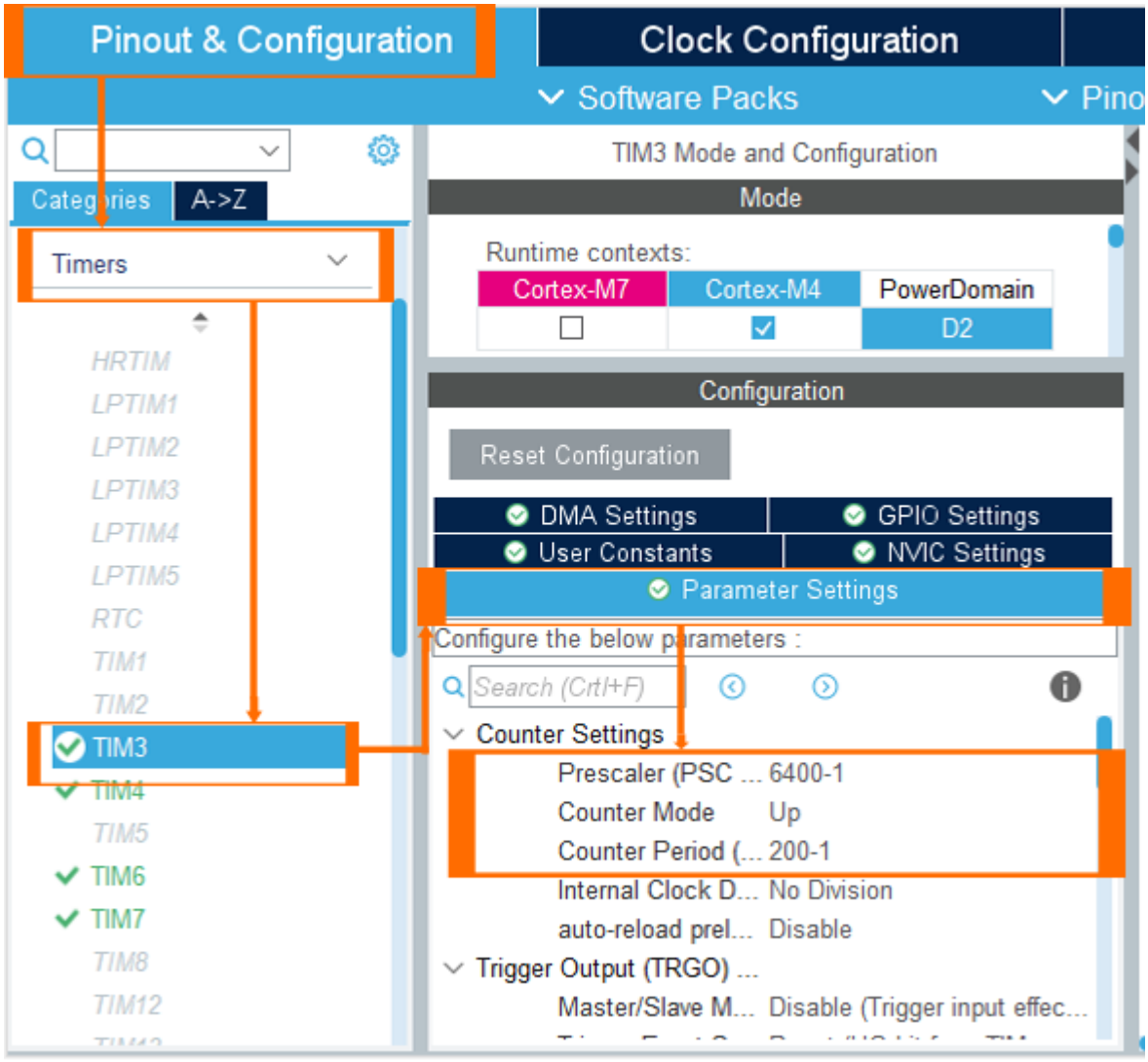
Abbiamo scelto Timer3 in quanto ci dà a disposizione 4 canali output per mandare segnale PWM.



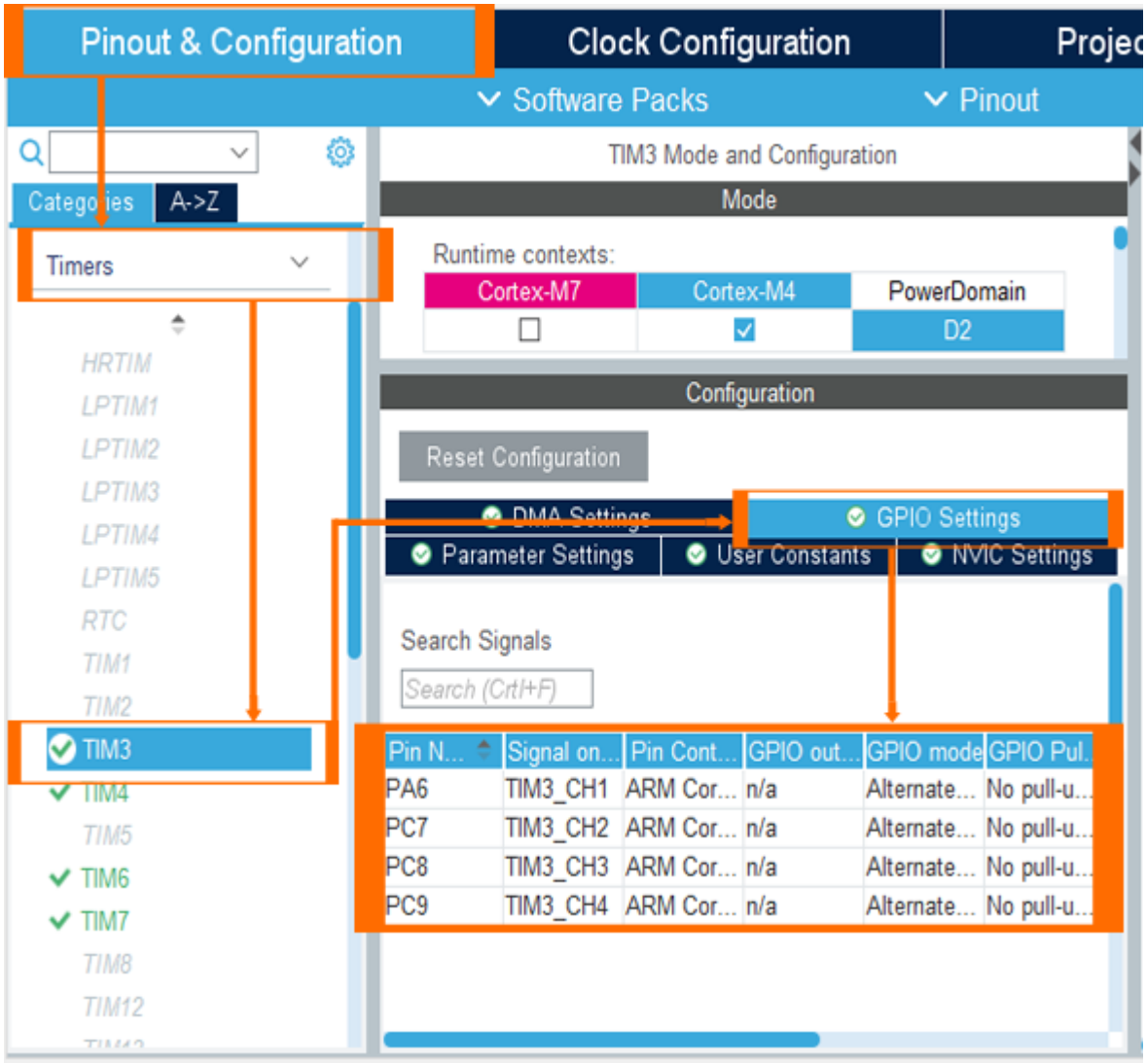
Configurazione Timer3 di frequenza **50HZ** per il segnale PWM

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

$$UpdateEvent = \frac{64000000}{(6400 + 1)(200 + 1)} = 50HZ = 20ms$$



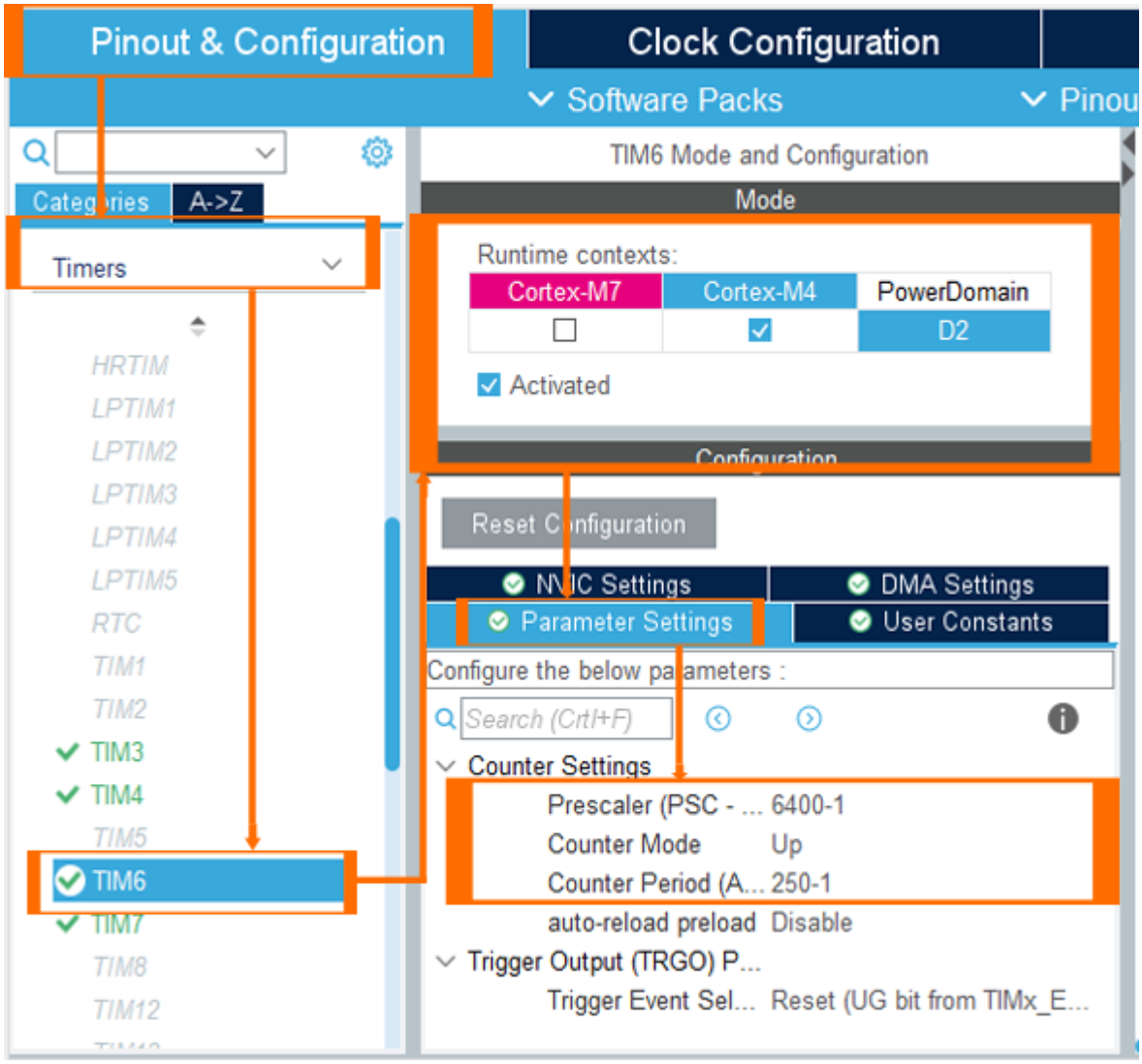
Questi sono i pin su qui vengono mandati in segnali PWM in uscita.

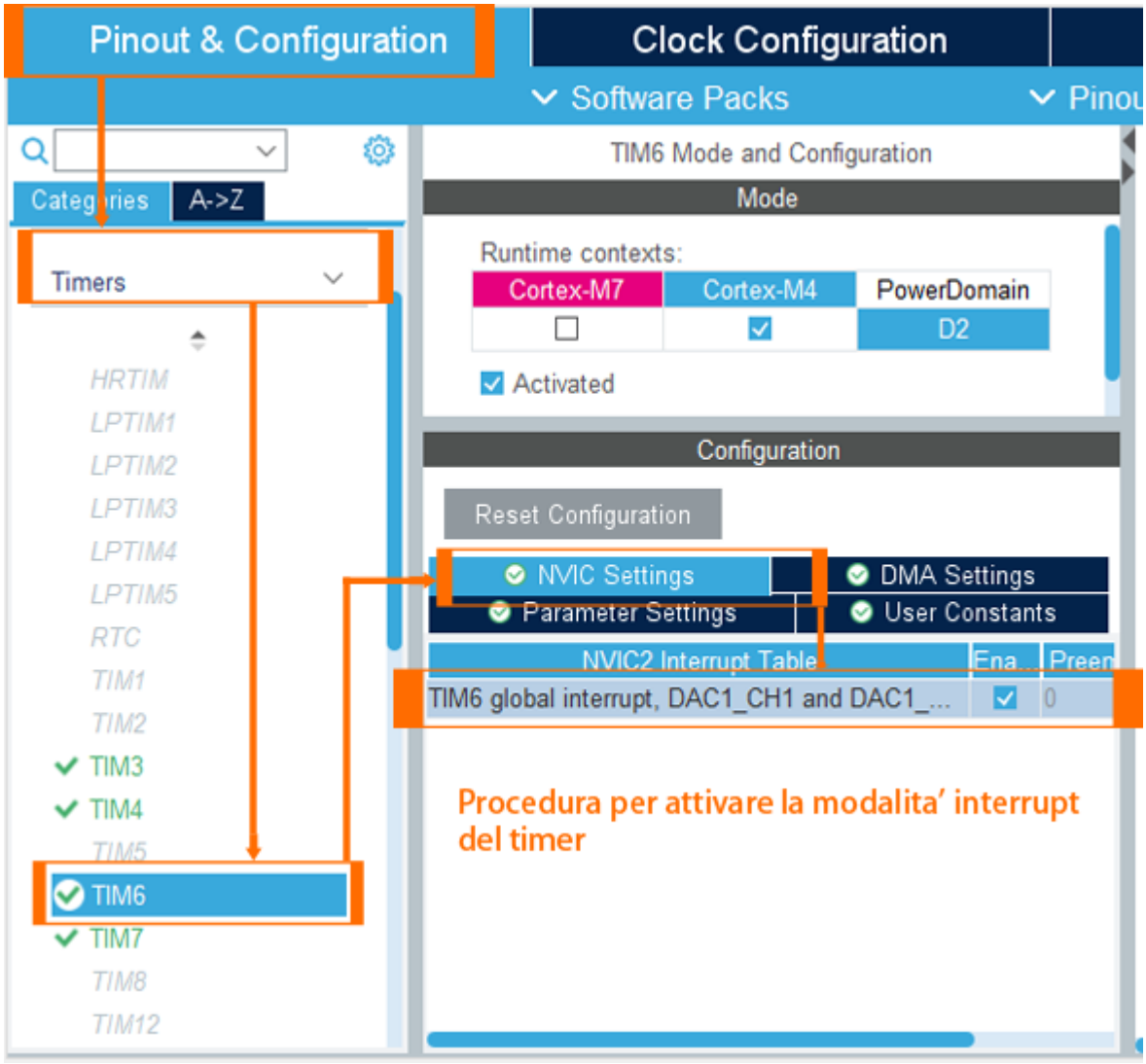


Configurazione Timer6 di frequenza **40 HZ** per il controllore PID:

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

$$UpdateEvent = \frac{64000000}{(6400 + 1)(250 + 1)} = 40HZ = 25ms$$





Per inizializzare il timer6 nella modalità interrupt si utilizza la seguente funzione della libreria HAL:

```
HAL_TIM_Base_Start_IT(&htim6);//inizialize timer7 in interrupt mode
```

```
//callback to detect the intrrupt event
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
//verify if the intrrupt comes from timer7
    if(htim==&htim7){
        //Read data from IMU (gyro,accel)
        MPU6050_Read_All(&hi2c2, &MPU6050);

        // get angle x from acceleration
        accel_anglesX  = atan2(MPU6050.Ay, MPU6050.Az) * RAD_T

        // get angle y from acceleration
        accel_anglesY  = atan2(-1 * MPU6050.Ax, sqrt(MPU6050.A

        // complementary filter
        angleX = GYRO_PART * (angleX + (MPU6050.Gx * IMU_dt))
        angleY = GYRO_PART * (angleY + (MPU6050.Gy * IMU_dt))
        angleZ=MPU6050.Gz;

        }
//verify if the intrrupt comes from timer6
    if(htim==&htim6){
```

Come è stato spiegato precedentemente la funzione

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) è la callback con la quale si verifica l'interrupt. Al contrario di prima la condizione **IF** (evidenziata in blu) verifica che l'interrupt sia avvenuto dal timer 6.

Display (Grove-Oled 1.12 V2)

abbiamo programmato un display del tipo oled per stampare i valori del Pitch,Roll e Yaw del quadrotor.

Prima si scarica una libreria fatta per inizializzare e stampre sullo schermo dal seguente link: https://www.micropeta.com/video19

Poi si modifica la parte di inizializzazione dello schermo perche' e' specifica per ogni schermo prendendo riferimento il manuale:

https://files.seeedstudio.com/wiki/Grove_OLED_1.12/resources/SH1107G_datasheet.pdf

Il codice di inizializzazione è il seguente:

```
/* Init LCD */
            SSD1306_WRITECOMMAND(0x0ae); //display off
            SSD1306_WRITECOMMAND(0x0dc); //Set Memory Addressing Mode
            SSD1306_WRITECOMMAND(0x000); //00,Horizontal Addressing Mode;
            SSD1306_WRITECOMMAND(0x081); // Set contrast control
            SSD1306_WRITECOMMAND(0x02f);
            SSD1306_WRITECOMMAND(0x020); // Set row address
            SSD1306_WRITECOMMAND(0x0a0); // Segment remap
            SSD1306_WRITECOMMAND(0x0c0); // Set Common scan direction
            SSD1306_WRITECOMMAND(0x0a8);
            SSD1306_WRITECOMMAND(0x7f);
            SSD1306_WRITECOMMAND(0x0d5); // Set Dclk
            SSD1306_WRITECOMMAND(0x050); // 100Hz
            SSD1306_WRITECOMMAND(0x0d9); // Set phase leghth
            SSD1306_WRITECOMMAND(0x022); //
            SSD1306_WRITECOMMAND(0x0db); // Set Vcomh voltage
            SSD1306_WRITECOMMAND(0x035);
            SSD1306_WRITECOMMAND(0x0b0); //--set normal display
            SSD1306_WRITECOMMAND(0x0da);// Set external VCC
            SSD1306_WRITECOMMAND(0x012);
            SSD1306_WRITECOMMAND(0x0a4); // Set Entire Display ON
            SSD1306_WRITECOMMAND(0x0a6);// Normal display
            SSD1306_WRITECOMMAND(0xaf);//Display ON
```

La frequenza di stampa sullo schermo e' di **10HZ** e la stampa avviene tramite il seguente codice:

```
void Print_XYZ(){

    SSD1306_GotoXY (0,0);
        SSD1306_Puts ("X=", &Font_11x18, 0); //print "X=" in (0.0) coordinate
        SSD1306_DrawLine(0,35,128,35,1); //print a line with coordinates (0,35,128,35)
        SSD1306_GotoXY (0,50);
        SSD1306_Puts ("Y=", &Font_11x18, 0); //print "Y=" in (0.50) coordinate
        SSD1306_DrawLine(0,80,128,80,1); //print a line with coordinates (0,80,128,80)
        SSD1306_GotoXY (0,100);
        SSD1306_Puts ("Z=", &Font_11x18, 0);//print "Y=" in (0.50) coordinate
        SSD1306_UpdateScreen(); //upadat Displat to print
}
```

La funzione **Print_XYZ()** viene chiamta solo una volta durante il funzionamento del codice in quanto queste stringhe sono fisse.

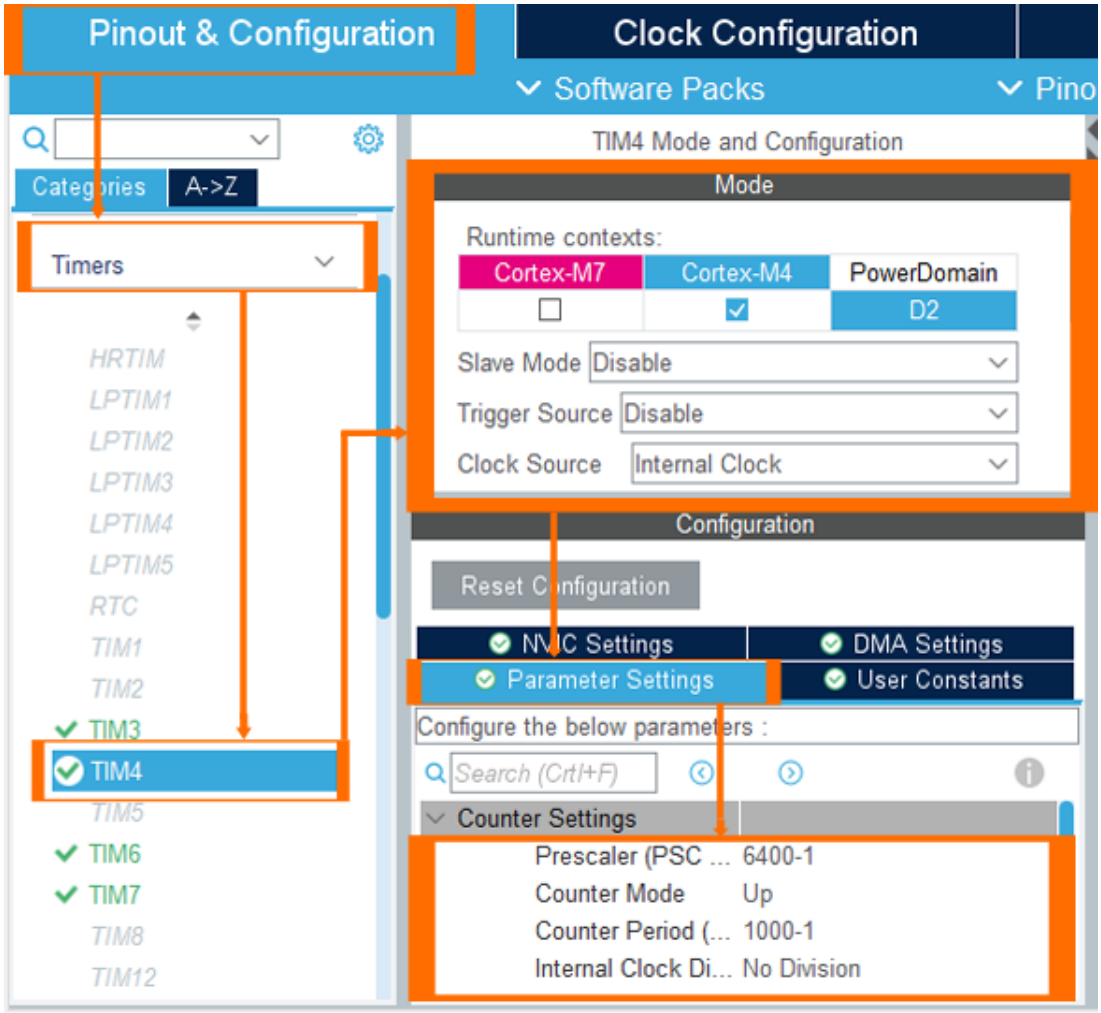
```
  while (1)
  {
     int timerValue=__HAL_TIM_GET_COUNTER(&htim4); // get value of timer count
      if(timerValue==1000){ //if counter value equal to 1000 so the if statment is right

      char bufx[100]; // vector to save the x value
      char bufy[100];// vector to save the y value
      char bufz[100];// vector to save the z value

      SSD1306_GotoXY (30,0);
      gcvt(angleX, 5,bufx); // convert from float to string and save in bufx
      SSD1306_Puts (bufx, &Font_11x18, 1); //print the x value in (30,0) coordinate
      SSD1306_GotoXY (30,50);
      gcvt(angleY, 5,bufy);// convert from float to string and save in bufy
      SSD1306_Puts (bufy, &Font_11x18, 1);//print the y value in (30,50) coordinate
      SSD1306_GotoXY (30,100);
      gcvt(angleZ, 5,bufz);// convert from float to string and save in bufz
      SSD1306_Puts (bufz, &Font_11x18, 1);//print the y value in (30,100) coordinate
      SSD1306_UpdateScreen();// upadate dispaly to print

      }
```

Procedura settaggio del Timer in modalita' **Polling** tramite CubeIDE:

Configurazione Timer4 di frequenza **10 HZ**:

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

$$UpdateEvent = \frac{64000000}{(6400 + 1)(1000 + 1)} = 10HZ = 100ms$$



Per inizializzare il timer4 nella modalità **Polling** si utilizza la seguente funzione della libreria HAL:

```
HAL_TIM_Base_Start(&htim4);//inizialize timer4 in Polling mode
```

```
while (1)
{
   int timerValue=__HAL_TIM_GET_COUNTER(&htim4); // get value of timer count
    if(timerValue==1000){ //if counter value equal to 1000 so the if statment is right

    char bufx[100]; // vector to save the x value
    char bufy[100];// vector to save the y value
    char bufz[100];// vector to save the z value

    SSD1306_GotoXY (30,0);
    gcvt(angleX, 5,bufx); // convert from float to string and save in bufx
    SSD1306_Puts (bufx, &Font_11x18, 1); //print the x value in (30,0) coordinate
    SSD1306_GotoXY (30,50);
    gcvt(angleY, 5,bufy);// convert from float to string and save in bufy
    SSD1306_Puts (bufy, &Font_11x18, 1);//print the y value in (30,50) coordinate
    SSD1306_GotoXY (30,100);
    gcvt(angleZ, 5,bufz);// convert from float to string and save in bufz
    SSD1306_Puts (bufz, &Font_11x18, 1);//print the y value in (30,100) coordinate
    SSD1306_UpdateScreen();// upadate dispaly to print

    }
```

Una volta inizializzato il timer in modalita' polling, andiamo nel ciclo while ad utilizzare la funzione **__HAL_TIM_GET_COUNTER**(&htim4) della libreria HAL per avere il valore del timer e lo salviamo nella variabile di appoggio **timerValue.** La condizione **IF** verifica che il valore del timer (**timerValue**) sia uguale a 1000 (0.1 secondi) che e' il valore del **Counter Period**, se la condizione e' verificata allora inizia a stampre sullo schermo i vari valori di X,Y e Z.

TARARATURA PID

Modificare i valori PID altera il comportamento del quadricottero in modi diversi. In questa sezione cerchiamo di capire gli effetti di ogni termine.

L'effetto di ogni termine

Termine P (proporzionale)

Il termine P tiene conto del valore attuale dell'errore . Una costante proporzionale alta farà si che l'azione di controllo sia grande anche in caso di errori piccoli , mentre una costante proporzionale bassa renderà meno rilevante il valore attuale dell'errore , privilegiando invece il valore che l'errore ha avuto in passato (azione integrale) e le dinamiche di variazione dell'errore nel tempo futuro (azione derivativa).

In generale, per un valore di P troppo alto, il quadricottero diventa troppo sensibile e cercherà di correggere eccessivamente gli errori. In questi casi, la correzione dell'errore va troppo oltre ed il quadricottero oscillerà ad alta frequenza.

Per ridurre le oscillazioni, una possibilità è quella di ridurre il termine P, ma non troppo altrimenti inizierà a risultare troppo lento nelle risposte.

Termine I (integrale)

ll termine I tiene appunto conto del valore assunto dall'errore nel passato.

ll termine I fornisce l'azione necessaria per eliminare l'errore di stato stazionario. Integra l'errore per un periodo di tempo fino a quando il valore dell'errore non raggiunge lo zero.

Quando I diventa troppo alto, il quadricottero sembrerà molto rigido e non agile. Un valore eccessivo di I potrebbe causare anche oscillazioni a bassa frequenza.

Termine D (Derivativo)

Il termine D serve a **smorzare e ridurre la correzione eccessiva** causati da termine P.

Un valore alto di D, può introdurre vibrazioni nel drone perchè amplifica il rumore all'interno del sistema, e questo porta al surriscaldamento dei motori.

Un altro effetto collaterale di un D troppo alto è una diminuzione nella risposta del quadricottero.

Taratura del Pitch

Nell laboratorio abbiamo tarato prima l'asse x.

Abbiamo iniziato la taratura inizializzando P=1,I=0,D=0 tramite la funzione **PID_Init().**

```
void PID_Init(PID_config* conf, float kp, float kd, float ki, float dt, float outMin, float outMax)
{
    conf->kp = kp;
    conf->kd = kd;
    conf->ki = ki;
    conf->dt = dt;
    conf->ITerm = 0;
    conf->lastError = 0;
    conf->outMax = outMax;
    conf->outMin = outMin;
}
```

Dato che il quadricottero rimbalzava in maniera continua, abbiamo diminuito il valore del **P** di 0.2 fino a raggiungere il valore di 0.4; il quadricottero ha cessato cosi` il rimbalzo ma la risposta dei motori era molto lenta, quindi abbiamo deciso di aumentare il valore del **P** fino a 0.8.

Per ridurre l'eccessiva compensazione dell`errore, data da quest`ultimo valore (0.8), abbiamo settato il valore del **I** a 0.2 con conseguente non rimbalzo del quadricottero anche se, manipolandolo per verificarne la risposta, impiegava troppo tempo a recuperare l`angolo desiderato (nel nostro caso di 0 gradi).

Per ovviare a questo problema abbiamo introdotto il valore del **D**, aumentandolo gradualmente di 0.1, notando poi un miglioramento nella reattivita` del recupero dell`angolo desiderato, anche se ancora distante dal risultato ottimale.

Di conseguenza siamo tornati ad aumentare ultieriormente il valore del **P** di 0.2, arrivando ad 1.5, e poi del **D** di 0.1, portandolo a 0,5.

Ecco che finalmente otteniamo i risultati desiderati ed ottimali da parte del quadricottero, che risponde perfettamente alle variazioni dell`angolo, stabilizzandosi.

Taratura del Roll

I valori finali ottenuti li abbiamo poi riapplicati all'asse Y e il qudricottero rispondeva in maniera analoga.

Ricapitolando, i valori finali del **PID** sia per **Pitch** (asse X) che per **Roll** (asse Y) sono i seguenti:

- **P=1.5**

- **I=0.2**

- **D=0.5**