



# UNIVERSITÀ POLITECNICA DELLE MARCHE

**Facoltà di Ingegneria Informatica e dell'Automazione**

**RELAZIONE LABORATORIO DI AUTOMAZIONE**

---

## **Drone: Radiocomando**

---

*Autori:*

Bravi Nicolas

Rachiglia Francesco

*Docente:*

Bonci Andrea

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Cos'è un quadrirotore? . . . . .	3
1.1.1	Cenni Storici . . . . .	3
1.1.2	Fisica di un quadrirotore . . . . .	4
<b>2</b>	<b>Modello matematico di un quadri-rotore</b>	<b>5</b>
2.1	Sistemi di riferimento . . . . .	5
2.2	Dinamica . . . . .	6
2.3	Controllo . . . . .	9
<b>3</b>	<b>HARDWARE</b>	<b>10</b>
3.1	Scheda . . . . .	10
3.2	Motori . . . . .	11
3.3	ESC . . . . .	12
3.3.1	Cosa è e come funziona . . . . .	12
3.3.2	Componenti ESC . . . . .	13
3.3.3	Programmare l'ESC . . . . .	14
3.4	Eliche . . . . .	15
3.5	Batteria Lipo Saver . . . . .	15
3.6	Telaio . . . . .	15
3.7	IMU . . . . .	17
3.7.1	Posizionamento . . . . .	17
3.8	Trasmettitore e Ricevitore del Radiocomando . . . . .	18
3.8.1	Caratteristiche Remote Controller . . . . .	18
3.8.2	Come funziona? . . . . .	20
3.9	Connettori . . . . .	21
<b>4</b>	<b>SOFTWARE</b>	<b>23</b>
4.1	Test Ricevitore . . . . .	23
4.1.1	Impostazioni Timer . . . . .	24
4.1.2	Calcolo frequenza . . . . .	24
4.1.3	Calcolo duty cycle . . . . .	25
4.2	Test Brushless . . . . .	27
4.3	Struttura del codice . . . . .	29
4.4	Gestione Interrupt . . . . .	30
4.5	Operazioni iniziali . . . . .	33
4.6	Loop infinito . . . . .	35
<b>5</b>	<b>Conclusione</b>	<b>39</b>
<b>6</b>	<b>Bibliografia e Sitografia</b>	<b>40</b>

# 1 Introduzione

L'obiettivo di questa relazione è esporre come stabilizzare un quadrirotore attraverso l'uso di un micro-controllore STM, in particolare la **Nucleo-144 STM32H7 45 zi-q**. La stabilizzazione del quadrirotore è stata effettuata attraverso un controllo PID dei segnali PWM che vengono inviati dal radiocomando, nello specifico abbiamo utilizzato i segnali PWM per stabilire le rotazioni e la velocità del drone che saranno poi gestite da un controllore PID implementato da un altro gruppo.

Di seguito è spiegato cosa è, quale è la struttura e i vari componenti elettronici che sono stati utilizzati per la realizzazione del quadrirotore.

Infine poniamo il focus sul radiocomando, in particolar modo sulla parte software, ovvero sul codice utilizzato per la realizzazione di un controllo remoto.

## 1.1 Cos'è un quadrirotore?

### 1.1.1 Cenni Storici

Nel corso del tempo si sono sviluppate varie tipologie di velivoli aerei, tra cui: veicoli a rotore centrale, verticale, come gli elicotteri, con o senza rotore di coda, (ovvero quel dispositivo che consente di annullare il momento torcente dovuto alla rotazione della pala centrale, in quest'ultimo caso, si adopera un secondo rotore, che ruota in modo opposto al primo e si parla di doppio rotore, a configurazione coassiale); velivoli a rotore centrale, orizzontale, come lo sono i vecchi aeroplani; velivoli birotori come possono essere i veicoli sofisticati in dotazione agli eserciti militari; velivoli multi-rotore che vanno da tre a otto rotori.

In questa ultima categoria rientra il nostro quadrirotore, dotato, appunto, di un sistema a quattro rotori disposti a croce, come in figura 1,e che, oggi giorno, costituisce uno dei velivoli più popolari. Per la prima volta fu realizzato come veicolo per il trasporto di passeggeri, ma ben presto uscì di scena a causa del peso eccessivo dovuto alle strutture necessarie a sostenere i 4 rotori ed alla scarsa stabilità che gravava sulla manovrabilità del pilota.

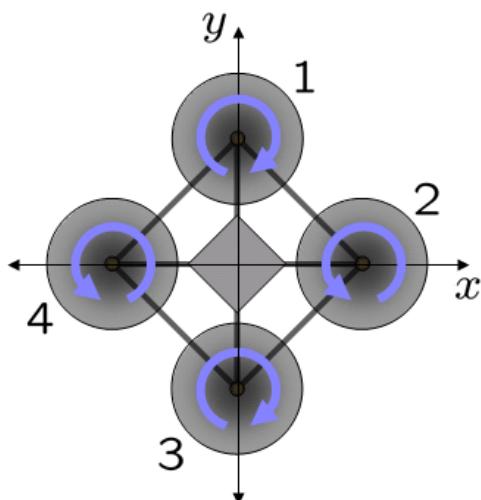


Figura 1: Schema drone

Recentemente con il diffondersi di veicoli denominati **UAV** (Unmanned Aerial Vehicle) questi sistemi hanno trovato svariate applicazioni in ambito civile tra cui: la didattica e l'esplorazione di ambienti impervi o pericolosi per gli esseri umani. I sistemi UAV, introdotti in ambito militare per sorvegliare aree pericolose o inaccessibili a causa di condizioni ambientali avverse, utilizzano sistemi di controllo computerizzati che mediante l'elaborazione di sensori di posizione e orientamento sono in grado di pilotare il velivolo autonomamente o di rendere possibile la guida da terra, mediante radio-controlli.

Le potenzialità e i vantaggi di questi mini-veicoli derivano dalla loro manovrabilità, dalla possibilità di operare in situazione di hovering, (ovvero praticamente immobili), "autobilanciati" dal sistema di controllo autonomo. Per di più, se dotati di un controllo abbastanza efficace, possono essere utilizzati anche all'interno di un ambiente chiuso, e quindi, si rendono a tutti gli effetti disponibili per i più disparati servizi.

### 1.1.2 Fisica di un quadrirotore

Fisicamente, questo sistema si presenta come un corpo rigido simmetrico (frame), formato da quattro aste orizzontali che montano alle estremità quattro rotori, a pala fissa, opportunamente dimensionati. La simmetria del dispositivo consente di centralizzare tutto il carico sollevabile (payload), massimizzato rispetto ad un tradizionale elicottero, al centro del frame. La dinamica del movimento è completamente attuata agendo solo sui contributi di ogni singolo motore e, ulteriore differenza con i comuni velivoli aerei, le pale dei quattro rotori ruotano a due a due in senso contrario e sono di tipo normale (sinistrorso), per i motori 2-4 e, inverso (destrorso), per i motori 1-3. Questa caratteristica, consente di annullare il momento angolare dovuto agli effetti di rotazione delle pale, e, quindi, di annullare l'angolo di imbardata, yaw, quando tutti i motori girano con la stessa velocità (caratteristica spiegata in modo più approfondito nel capitolo successivo).

Nella medesima situazione, mediante relazioni matematiche è possibile aumentare o diminuire le velocità dei motori, che sono strettamente collegate alla portanza delle pale ( $f$ ) ed alla forza totale di spinta ( $u$ ). Tramite ciò, è possibile controllare il moto del velivolo lungo l'asse  $z$ , ovvero: l'altitudine.

Per quanto riguarda, invece, i movimenti lungo gli assi orizzontali, essi sono controllati agendo sulle coppie di rotori simmetrici, mantenendo la rotazione dei restanti alla stessa velocità. Come possibile osservare dalla figura 2 vediamo che un aumento della velocità dei motori 1-2 implica una rotazione attorno all'asse trasversale, che, secondo un termine derivato dall'aeronautica viene definita: beccheggio, in inglese, **PITCH**.

Allo stesso modo, accade per i motori 2 e 3, come si può evincere dalla figura 2, dove si genera l'angolo di rollio, inglese, **ROLL**.

In basso a destra, figura 2, è, invece, rappresentata la configurazione adibita alla rotazione sul piano  $x,y$  di cui abbiamo già esaminato il caso in cui le velocità sono uguali. Qui, due motori opposti (2-3) diminuiscono la loro velocità, quindi, il momento torcente dominante risulta essere quello generato dai restanti rotori (1-4). In effetti quello che si verifica è la generazione di una rotazione attorno all'asse  $z$ , che viene definita imbardata, in inglese, **YAW**.

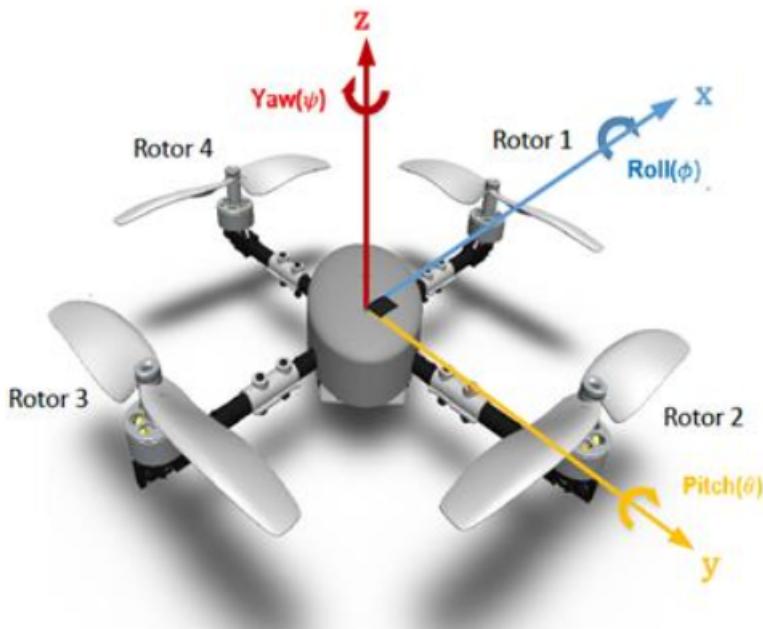


Figura 2: Angoli di Pitch, Roll e Yaw

## 2 Modello matematico di un quadri-rotore

### 2.1 Sistemi di riferimento

Il modello presentato nella figura 3 mostra i sistemi di riferimento del quadri-rotore, le velocità angolari, le forze e i momenti sprigionati dai rotori. Di seguito spieghiamo il modello del quadrirotore ma per ulteriori approfondimenti si riporta al seguente sito: [https://www.academia.edu/33664128/Modelling\\_and\\_control\\_of\\_quadcopter\\_School\\_of\\_Science\\_Mat\\_2\\_4108\\_Independent\\_research\\_project\\_in\\_applied\\_mathematics\\_A](https://www.academia.edu/33664128/Modelling_and_control_of_quadcopter_School_of_Science_Mat_2_4108_Independent_research_project_in_applied_mathematics_A)

La posizione assoluta del quadri-rotore è definita nel sistema di riferimento inerziale dagli assi x-y-z che possiamo esprimere con  $\varepsilon$ . L'assetto, per esempio la posizione angolare è definita nel sistema di riferimento inerziale con gli angoli di Eulero  $\eta$ . L'angolo di pitch  $\theta$  determina le rotazioni attorno all'asse y. L'angolo di roll  $\phi$  determina le rotazioni attorno all'asse x e l'angolo di yaw  $\psi$  determina le rotazioni attorno all'asse z.

$$\varepsilon = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad (1)$$

L'origine del sistema inerziale solidale con il quadri-rotore è nel centro di massa. Nel sistema solidale con il corpo le velocità sono determinate da  $V_B$  e le velocità angolari sono espresse con  $\nu$ .

$$V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \quad \nu = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (2)$$

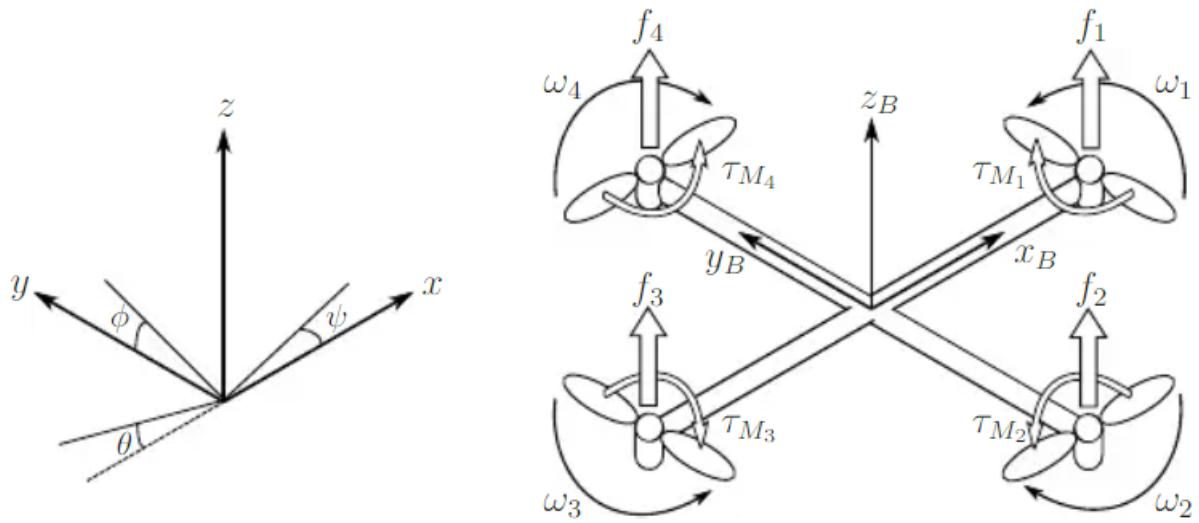


Figura 3: Sistema inerziale assoluto e solidale con il quadri-rotore.

La matrice di rotazione per passare dal sistema solidale con il corpo al sistema di riferimento inerziale è:

$$\mathbf{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \quad (3)$$

in cui  $C_x = \cos(x)$  e  $S_x = \sin(x)$ . La matrice di rotazione  $\mathbf{R}$  è ortogonale, quindi  $R^{-1} = R^T$  che è la matrice di rotazione dal sistema inerziale al sistema solidale con il quadri-rotore.

Abbiamo assunto inoltre che il quadri-rotore abbia una struttura simmetrica con le quattro eliche allineate con il corpo sugli assi x e y. In questo modo otteniamo una matrice di inerzia diagonale in cui  $I_{xx} = I_{yy}$ .

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (4)$$

## 2.2 Dinamica

Il quadri-rotore è un corpo rigido quindi possiamo utilizzare le equazioni di Eulero e Newton per descrivere la sua dinamica.

La velocità angolare del rotore i, che chiameremo  $w_i$ , ci consente di ottenere le forze  $f_i$  in direzione degli assi dei rotori. La velocità angolare e l'accelerazione dei rotori crea anche un momento  $\tau_{M_i}$ , quindi:

$$f_i = k w_i \quad (5)$$

$$\tau_{M_i} = bw_i^2 + I_M \dot{w}_i \quad (6)$$

in cui:

- $k$  è la costante che consente di sollevare il quadri-rotore
- $b$  è il coefficiente di attrito
- $I_M$  è il momento di inerzia del rotore

Il valore di  $\dot{w}_i$  è generalmente piccolo e quindi può essere trascurato. La somma delle forze  $\sum_{i=1}^4 f_i$  dei rotori crea un impulso  $\mathbf{T}$  nella direzione dell'asse delle z nel sistema di riferimento solidale con il quadri-rotore.

Il momento  $\tau_B$  è dato dalla somma dei momenti  $\tau_\phi, \tau_\theta$  e  $\tau_\psi$  nelle direzioni corrispondenti agli angoli del sistema di riferimento solidale con il quadri-rotore.

$$T = \sum_{i=1}^4 f_i = k \sum_{i=1}^4 w_i^2, \quad T^B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \quad (7)$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk(-w_2^2 + w_4^2) \\ lk(-w_1^2 + w_3^2) \\ \sum_{i=1}^4 \tau_{M_i} \end{bmatrix} \quad (8)$$

In cui  $l$  è la distanza tra il rotore e il centro di massa. Quindi il movimento dell'angolo di roll è ottenuto diminuendo  $w_2$  e aumentando  $w_4$ . Analogamente il movimento dell'angolo di pitch è ottenuto diminuendo  $w_1$  e aumentando  $w_3$ . Per quanto riguarda invece l'angolo di jaw, il movimento è ottenuto aumentando la velocità di due rotori opposti e diminuendo la velocità degli altri due.

Nel sistema di riferimento solidale con il corpo l'espressione della dinamica può essere espressa come:

$$m\dot{V}_B + \nu \times mV_B = R^T G + T_B \quad (9)$$

in cui:

- $m\dot{V}_B$  è la forza per l'accelerazione della massa.
- $\nu \times mV_B$  è la forza centrifuga
- $R^T G$  è la forza di gravità
- $T_B$  impulso dei propulsori

Nel sistema di riferimento inerziale, la forza centrifuga viene annullata. Quindi solo la forza gravitazionale, l'impulso e la direzione dei rotori contribuiscono all'accelerazione del quadri-rotore.

$$m\ddot{\epsilon} = G + RT_B \quad (10)$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + T/m \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi C_\theta C_\phi \end{bmatrix} \quad (11)$$

Nel sistema solidale con il corpo, l'accelerazione angolare è data da  $\mathbf{I}\dot{\nu}$ , le forze centripete  $\nu \times (\mathbf{I}\nu)$  e le forze misurate dal giroscopio sono uguali al momento delle forze.

$$I\dot{\nu} + \nu \times I\nu + \Gamma = \tau \quad (12)$$

$$\dot{\nu} = I^{-1} \left( - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx}p \\ I_{yy}q \\ I_{zz}r \end{bmatrix} - Ir \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w_\Gamma + \tau \right) \quad (13)$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} I_{yy} - I_{zz}qr/I_{xx} \\ I_{zz} - I_{xx}pr/I_{yy} \\ I_{xx} - I_{yy}pq/I_{zz} \end{bmatrix} - Ir \begin{bmatrix} q/I_{xx} \\ -p/I_{yy} \\ 0 \end{bmatrix} w_\Gamma + \begin{bmatrix} \tau_{phi}/I_{xx} \\ \tau_{theta}/I_{yy} \\ \tau_{psi}/I_{zz} \end{bmatrix} \quad (14)$$

in cui  $w_\Gamma = w_1 - w_2 + w_3 - w_4$ . Le accelerazioni angolari nel sistema di riferimento inerziale sono collegate alle accelerazioni nel sistema di riferimento solidale con il quadri-rotore con la matrice di trasformazione  $W_\eta^{-1}$  e la sua derivata rispetto al tempo.

$$\ddot{\eta} = (W_\eta^{-1}\nu) = d/dt(W_\eta^{-1})\nu + (W_\eta^{-1})\dot{\nu} = \begin{bmatrix} 0 & \dot{\phi}C_{phi}T_\theta + \dot{\theta}S_{phi}/C_\theta^2 & -\dot{\phi}S_{phi}C_\theta + \dot{\theta}C_{phi}/C_\theta^2 \\ 0 & \dot{\phi}S_\phi & -\dot{\phi}C_\phi \\ 0 & \dot{\phi}C_{phi}/C_\theta + \dot{\phi}S_\phi T_\theta/C_\theta & -\dot{\phi}S_\phi/C_\theta + \dot{\theta}C_\phi T_\theta/C_\theta \end{bmatrix} \nu + W_\eta^{-1}\dot{\nu} \quad (15)$$

dove  $W_\eta$  è la matrice che ci consente di passare dalle velocità angolari nel sistema di riferimento inerziale al sistema di riferimento solidale con il quadri-rotore ed è:

$$W_\eta = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \quad (16)$$

che è invertibile se  $\theta \neq (2k - 1)\pi/2$ ,  $k \in \mathbb{Z}$ .

## 2.3 Controllo

Per stabilizzare un quadri-rotore viene utilizzato un controllore di tipo PID. I vantaggi di utilizzare questo tipo di controllo sono una struttura semplice e la facile implementazione del controllore. In generale un controllore PID è (per ulteriori informazioni si rimanda al link nella sezione 2.1 ):

$$e(t) = x_d(t) - x(t), \quad u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} \quad (17)$$

in cui  $u(t)$  è l'ingresso virtuale di controllo,  $e(t)$  è la differenza tra lo stato desiderato  $x_d(t)$  e lo stato attuale  $x(t)$ , e  $K_P$ ,  $K_I$  e  $K_D$  sono i parametri per il controllore proporzionale, integrale e derivativo.

Nei quadri-rotori, ci sono sei stati, ma solo quattro ingressi di controllo, ovvero le velocità angolari  $w_i$  dei rotori. Le interazioni tra gli stati, l'impulso totale dei propulsori e i momenti generati sono visibili dalla dinamica del quadri-rotore che sono descritti dalle equazioni (11), (12), (13), (14) e (15).

Per l'assetto viene utilizzato un controllore PD scelto nel seguente modo:

$$T = (g + K_{z,D}(\dot{z}_d - \dot{z}) + K_{z,P}(\ddot{z}_d - \ddot{z})) \frac{m}{C_\phi C_\theta} \quad (18)$$

$$\tau_\phi = (K_{\phi,D}(\dot{\phi}_d - \dot{\phi}) + K_{\phi,P}(\ddot{\phi}_d - \ddot{\phi})) I_{xx} \quad (19)$$

$$\tau_\theta = (K_{\theta,D}(\dot{\theta}_d - \dot{\theta}) + K_{\theta,P}(\ddot{\theta}_d - \ddot{\theta})) I_{yy} \quad (20)$$

$$\tau_\psi = (K_{\psi,D}(\dot{\psi}_d - \dot{\psi}) + K_{\psi,P}(\ddot{\psi}_d - \ddot{\psi})) I_{zz} \quad (21)$$

in cui  $\mathbf{g}$  è l'accelerazione di gravità,  $\mathbf{m}$  è la massa del quadri-rotore e  $\mathbf{I}$  è il momento di inerzia.

Le velocità angolari corrette  $w_i$  possono essere calcolate a partire dalle equazioni (7), (8), (18), (19), (20), (21).

$$w^2_1 = \frac{T}{4k} - \frac{\tau_\theta}{2kl} - \frac{\tau\psi}{4b}, \quad (22)$$

$$w^2_2 = \frac{T}{4k} - \frac{\tau_\phi}{2kl} + \frac{\tau\psi}{4b}, \quad (23)$$

$$w^2_3 = \frac{T}{4k} + \frac{\tau_\theta}{2kl} - \frac{\tau\psi}{4b}, \quad (24)$$

$$w^2_4 = \frac{T}{4k} + \frac{\tau_\phi}{2kl} + \frac{\tau\psi}{4b} \quad (25)$$

### 3 HARDWARE

#### 3.1 Scheda

La scheda utilizzata, come detto in precedenza, è la **Nucleo-144 STM32H7 45 zi-q**.

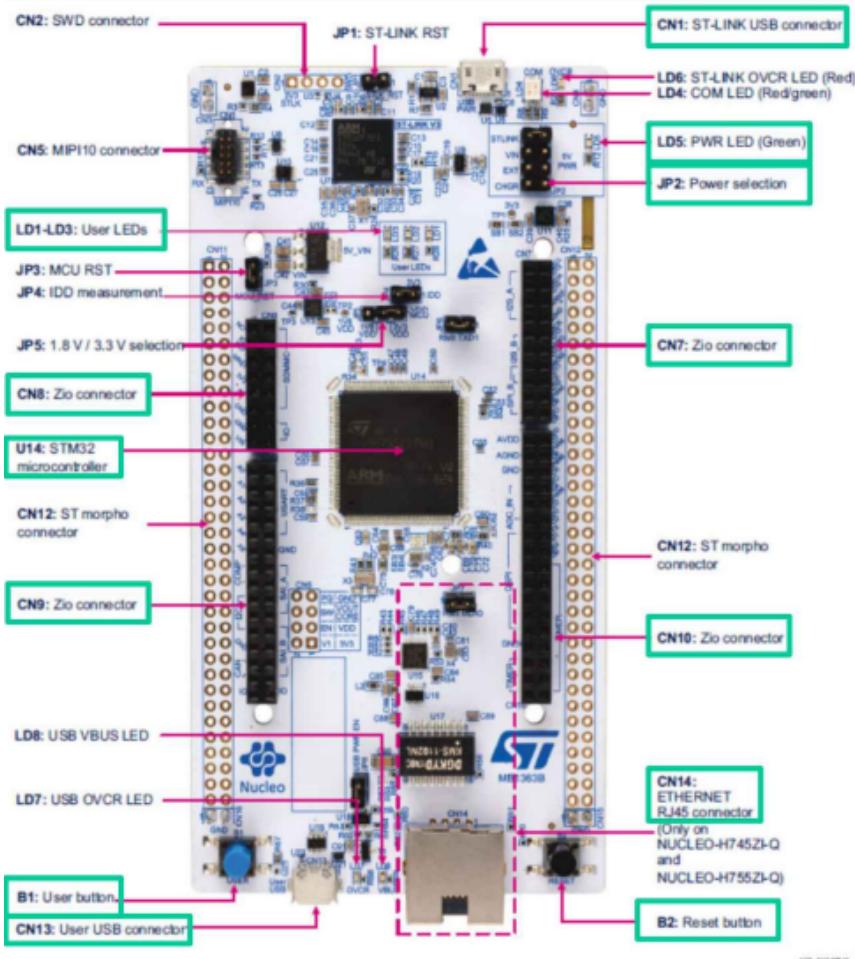


Figura 4: Nucleo-144 STM32H7 45 zi-q

#### Specifiche

Caratteristiche	Valori
Peso	60g
Dimensioni	13.34cm x 7cm
Processore	ARM Cortex
Supporto IDE	Sì

La scheda monta due processori ARM Cortex che sono:

- ARM Cortex M7 32-bit
- ARM Cortex M4 32-bit

Troviamo inoltre una porta Ethernet, una porta USB, 3 LEDs. La scheda monta inoltre connettori SWD, ST Zio expansion including ARDUINO Uno V3, ST morpho expansion.

La scheda è dotata di un SMPS ovvero un alimentatore in modalità commutata per generare una tensione logica  $V_{core}$ , e connettori del tipo Ethernet RJ45 e USB OTG full speed.

Abbiamo anche a disposizione un ambiente di sviluppo integrato (IDE) che include:

- IAR Embedded Workbench,
- MDK-ARM,
- STM32CubeIDE

### 3.2 Motori

Nel drone sono presenti quattro motori brushless Turnigy D3536/9 910KV.



Figura 5: Motore Brushless Turnigy D3536/9 910KV

#### Caratteristiche:

Sono motori trifase, “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).

Ogni motore è collegato a un’elica (propeller) che roteando permette al drone di alzarsi da terra. I motori installati brushless, ovvero ”senza spazzole”, installati sul drone non necessitano di contatti elettrici strisciati per funzionare, in questo modo otteniamo diversi vantaggi, quali:

- Minore resistenza meccanica
- Peso inferiore
- Non si formano le scintille dovute al contatto spazzole-collettore
- Minore necessità di manutenzione

Questi motori però necessitano di ulteriori componenti elettronici, come ad esempio l'ESC (Electronic Speed Controller).

### Caratteristiche specifiche Turnigy D3536/9 910KV

Caratteristiche	Valori
Peso	0.117kg
Batteria	2 4 Cell /7.4 14.8V
RPM	910kv
Corrente massima	25.5A
Corrente a vuoto	1.5A
Potenza massima	370W
Resistenza interna	0.063 ohm
Diametro dell'albero motore	5mm
Dimensioni	35x36m
Dimensione prop	7.4V/12x5 14.8V/10x7
Impulso massimo	1050g

## 3.3 ESC

### 3.3.1 Cosa è e come funziona

L'ESC (Electronic Speed Controller) è la parte del drone che collega i motori brushless al flight controller ed è costituita da un insieme di cavi nella cui parte centrale è posizionata una piccola scheda elettronica. Questa scheda, seppur piccola, consente ai motori di girare in entrambi i sensi e a velocità variabili.

L'ESC funge da intermediario tra la batteria e il motore elettrico. Controlla la rotazione del motore fornendo segnali elettrici temporizzati che vengono tradotti in variazioni di velocità.

Il controller dell'acceleratore del veicolo viene utilizzato per variare la velocità del motore, sia che si tratti di un'auto elettrica, di un aereo o di un drone. Aumentando l'acceleratore aumenta la potenza di uscita, che modifica la velocità con cui gli interruttori si aprono e si chiudono nel circuito dell'ESC.

Esistono diversi protocolli di consegna del segnale utilizzati per trasmettere le informazioni sull'acceleratore dal telecomando all'ESC. Ogni protocollo ha prestazioni leggermente diverse, i più comuni sono PWM (da noi utilizzato), Oneshot, Multishot e Dshot.

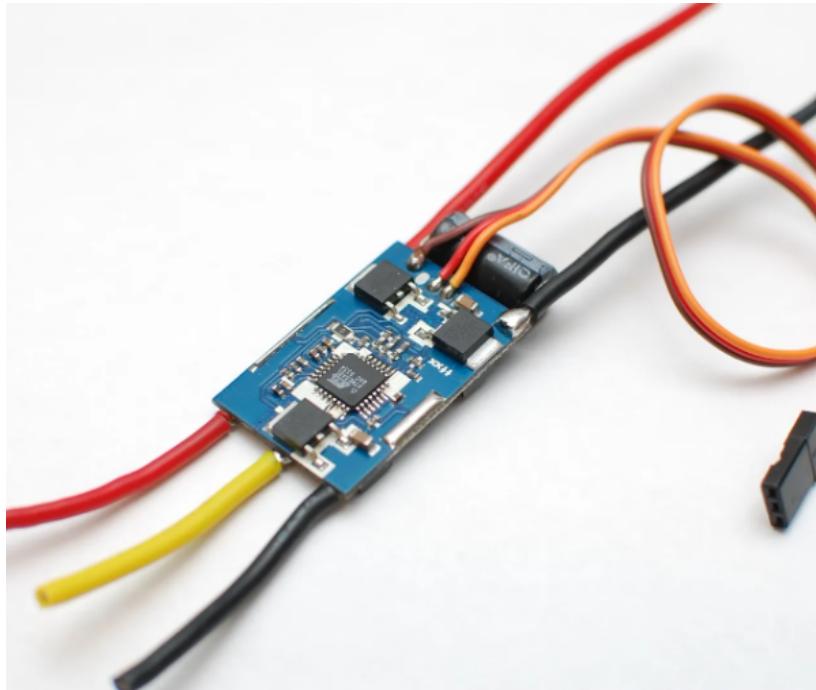


Figura 6: ESC-Electronic Speed Controller

### 3.3.2 Componenti ESC

All'interno dell'ESC ci sono una serie di componenti importanti come MCU, gate driver e i MOSFET, figura 7, ma anche un circuito di eliminazione della batteria e l'adattatore di gestioni di dispositivi in alcuni casi.

#### Microcontrollore MCU

Il microcontrollore svolge tre ruoli chiave nel funzionamento dell'ESC:

- 1) Alloggiamento del firmware che interpreta il segnale dal controller e lo alimenta in un loop di controllo;
- 2) Tenere traccia della posizione del motore per garantire un'accelerazione regolare;
- 3) Inviare impulsi al gate driver per ottenere il comando desiderato.

Il microcontrollore determina anche la posizione del motore attraverso un sistema sensorizzato o sensorless. I sistemi con sensori utilizzano sensori elettronici nel motore per tracciare la posizione del rotore, ideale per applicazioni a bassa velocità e coppia elevata.

**MOSFET** I MOSFET sono gli interruttori che forniscono energia al motore.

L'ESC ha 6 di questi transistor e ogni filo del motore è collegato a due di essi. Ricevono segnali dal microcontrollore e forniscono energia al motore in modo che ognuna delle sue bobine sia in una delle tre fasi: alta tensione, bassa tensione o terra.

Mentre il motore ruota, i segnali dei MOSFET commutano le fasi delle bobine in modo che il rotore continui a girare.

**Circuito di eliminazione della batteria** Gli ESC hanno spesso un circuito di eliminazione della batteria (BEC) incorporato, che non elimina la necessità di una batteria, ma funge da regolatore di tensione per eliminare la necessità di una batteria separata per l'elettronica di bordo.

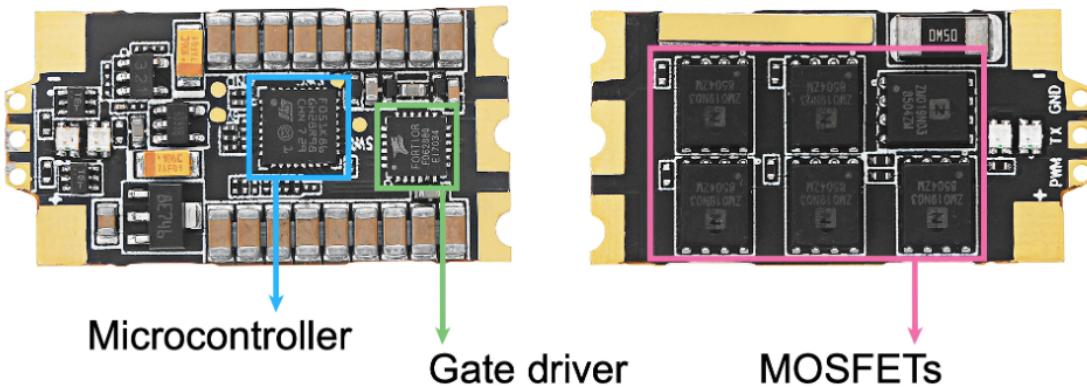


Figura 7: Componenti ESC

### 3.3.3 Programmare l'ESC

Collegare l'ESC alla scheda di programmazione dal cavo BEC e alla batteria, settare poi i led come in figura 8. All'accensione prima di far partire i motori è necessario eseguire una procedura di "armamento". Tale procedura consiste nell'inviare un segnale PWM sulla linea dati (BEC) dell'ESC prima di fornire l'effettiva tensione di alimentazione. Quando la procedura di armamento sarà eseguita correttamente, gli ESC emetteranno un segnale audio, mentre se riscontrano un problema nella procedura di armamento saranno emessi due BEEP ogni due secondi. Le istruzioni software sono riportate nella sezione 4.2.

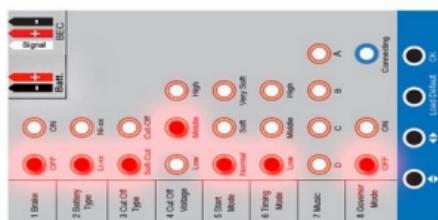


Figura 8: Scheda di programmazione ESC

### 3.4 Eliche

Ai motori sono stati collegate 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.

La spinta, che è sufficiente a far volare il quadricottero senza problemi è stata calcolata con una semplice formula:

$$S_{min} = \frac{PesoQuadricottero}{4} \quad (26)$$

### 3.5 Batteria Lipo Saver

Nel quadricottero è montata 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.



Figura 9: Lipo Saver

Uno dei principali problemi delle batterie LiPo è il controllo della carica minima. Infatti ogni cella della batteria non deve scendere al di sotto dei 2.7V, per evitare il danneggiamento della stessa. Per fare ciò è stata adottata una piccola scheda detta LiPo Saver, utilizzata per verificare la carica della batteria, che, attraverso un diodo led, ci avverte quando la tensione delle singole celle della batteria scendono al di sotto dei 3V.

### 3.6 Telaio

Il nostro gruppo non ha lavorato sulla parte Hardware del drone, le seguenti informazioni sono state infatti prese dalla tesina "Relazione Altimetro e controllo Motori" di Matteo Ferretti, Alessandro De Toni, Angelo D'Agostino Bonomi, Simone Di Rado.

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.



Figura 10: 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, da una vista in pianta 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.

### 3.7 IMU

Un IMU è una scheda elettronica 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. 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)



Figura 11: IMU Adafruit BNO055

#### 3.7.1 Posizionamento

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, 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 figura 19 dato che il sistema di riferimento dell'IMU deve coincidere con quello del Drone. Nell'immagine 19 mostriamo come deve essere posizionata l'IMU, ma da notare che nell'immagine è stata utilizzata una scheda Renesas e che quindi il posizionamento è diverso nella scheda da noi utilizzata ovvero la Nucleo-144 STM32H7 45 zi-q.

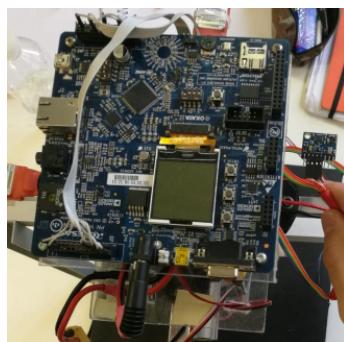


Figura 12: Caption

### 3.8 Trasmettitore e Ricevitore del Radiocomando

Il radiocomando che abbiamo utilizzato per inviare segnali PWM al fine di controllare il drone con un remote controller è il **"Microzone 6CH Remote Transmitter Receiver System for Drones/Quadcopters"** modello: **"MC6C"**



Figura 13: Microzone 6CH Remote Transmitter Receiver System for Drones/Quadcopters

#### 3.8.1 Caratteristiche Remote Controller

Il radiocomando ha sei canali e la banda va dai 2,401GHz a 2,479GHz, ha una distanza di controllo fino a 800m e utilizza 4 batterie xAA.

Il remote controller è dotato di un ricevitore e un trasmettitore che hanno le seguenti caratteristiche:

- **Trasmettitore**

Potenza di trasmissione	$\leq 70\text{MW}$
Range per il controllo	800m
Tensione (DC)	+6V
Peso	550g



Figura 14: Microzone 6CH Remote Transmitter Receiver System

- **Ricevente**

Frequenza	2.400GHz ~ 2.483GHz
Tensione	4.5V ~ 6V
Segnale	PWM/SBUS
Antenna	All'interno del ricevitore
Peso	9.6g
Dimensioni	45*45*10mm

Come possibile notare in figura 15, per una ricevente a 6 canali ci sono 7 righe di pin. La prima in alto funge da bus, per cui è l'unica da alimentare. Accanto alle altre 6, ci sono dei numeri che specificano di quale canale si tratti. Per ricevere il segnale, non è necessario alimentare ogni canale, ma è sufficiente utilizzare i pin di sinistra.



Figura 15: Ricevente MC6RE

### 3.8.2 Come funziona?

I due dispositivi comunicano attraverso onde radio, che sono il vettore di un codice input generato dal trasmettitore e recapitato al ricevitore grazie all'antenna incorporata. Quest'ultimo, genera dei segnali PWM, uno per canale, con cui fornisce informazioni relative all'interazione di un presunto pilota con il radiocomando tramite la variazione del duty cycle del segnale definito in un certo intervallo.

Per prima cosa abbiamo associato le levette del radiocomando agli angoli che ci consentono di muovere il drone, ovviamente le leve non state scelte a caso ma abbiamo adottato delle scelte standard per associarle alle variazioni di velocità e inclinazione del drone. Nel dettaglio abbiamo associato la levetta destra agli angoli di Roll e Pitch, lo spostamento orizzontale della levetta comporta una variazione del PWM che viene inviato per gestire l'angolo di **Roll**, mentre lo spostamento verticale comporta una variazione del PWM che viene inviato per gestire l'angolo di **Pitch**. Per quanto riguarda invece la levetta di sinistra, abbiamo associato lo spostamento verticale della levetta al **Throttle**, mentre lo spostamento orizzontale è stato associato all'angolo di **Yaw**.

Abbiamo scelto di associare la levetta in alto a sinistra per l'armamento dei motori, può fare due scatti, spostandola al centro, i motori già armati all'accensione del drone, lo rendono pronto per l'utilizzo, riportandola invece verso l'alto o il basso i motori vengono spenti, quindi ricapitolando i motori vengono armati e spenti con lo stesso Duty Cycle, spostando la levetta in mezzo il duty aumenta e i motori sono pronti per far volare il drone. La levetta a destra invece non è stata utilizzata, di conseguenza non è associata a nessun comportamento del drone.

Infine, ci sono i "trigger", situati sotto e di fianco le levette. Il loro scopo è quello di traslare l'intervallo del duty cycle e sono presenti solo per i quattro canali principali. Possono essere utilizzati durante la fase di volo per correggere un problema hardware e ritrovare l'equilibrio.



Figura 16: Levette del radiocomando utilizzate

Nella figura 16 sono state evidenziate le levette del radiocomando nel seguente modo:

- Verde la levetta di destra (**Roll** e **Pitch**);
- Giallo la levetta di sinistra (**Throttle** e **Yaw**);
- Blu la levetta del quinto canale in alto a sinistra (per armare i motori);
- Rosso i trigger che consentono di aumentare o diminuire il segnale.

### 3.9 Connettori

I connettori che sono stati utilizzati sono riportati nella seguente tabella:

Periferica esterna	PIN	Descrizione
Radiocomando	PA5	Channel 1
Radiocomando	PB3	Channel 2
Radiocomando	PB10	Channel 3
Radiocomando	PB11	Channel 4
Radiocomando	PA3	Channel 5
Radiocomando	PA0	Channel 5
Brushless1	PA6	PWM1
Brushless2	PC7	PWM2
Brushless3	PC8	PWM3
Brushless4	PB1	PWM4

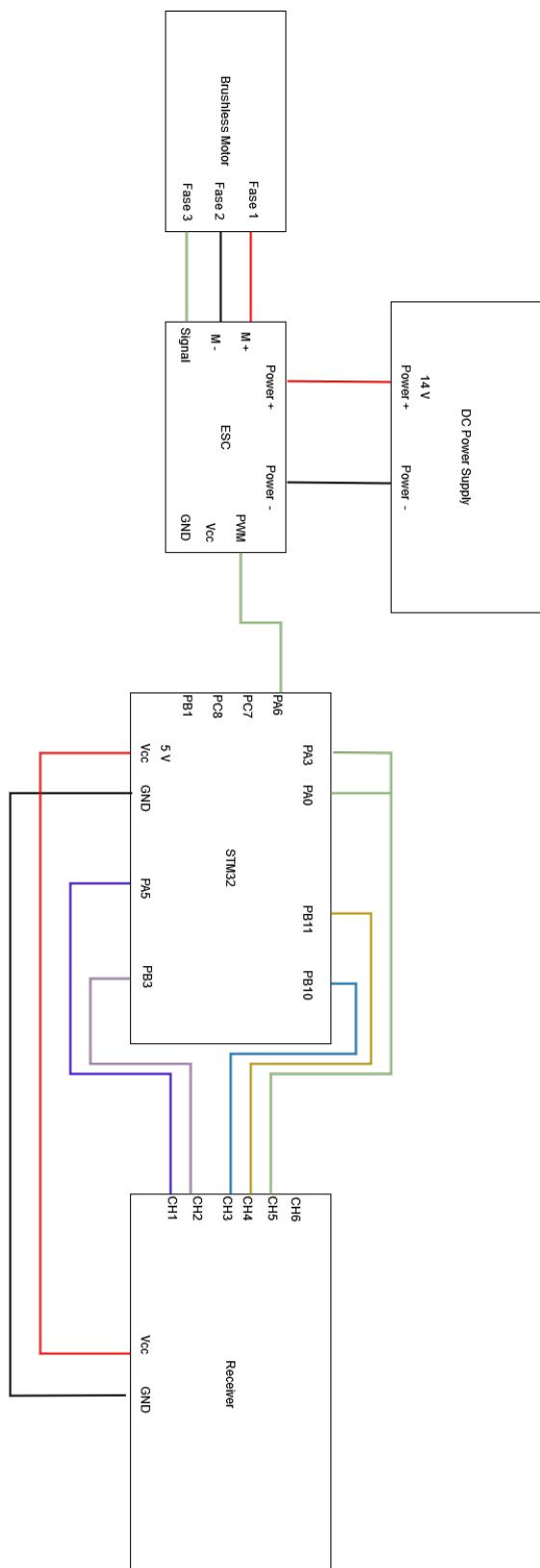


Figura 17: Collegamenti

## 4 SOFTWARE

Lo scopo di questo progetto è di leggere i valori dei canali del radiocomando per poi controllare un motore brushless. In ottica di possibili sviluppi futuri, si è deciso di analizzare solo i primi 5 canali, scartandone l'ultimo in quanto non necessario. I primi 4 possono essere utilizzati per pilotare il velivolo, mentre la levetta a tre livelli (quinto canale) viene utilizzata per specificare lo stato del drone. Il primo livello rappresenta la fase di armatura, il secondo livello rappresenta la fase di pilotaggio ed il terzo livello rappresenta la fase in cui i motori vengono bloccati. Come è possibile notare dal codice, dal punto di vista pratico, l'armatura e lo stop del motore, vengono effettuati allo stesso modo, per garantire un bloccaggio più efficiente.

### 4.1 Test Ricevitore

Poiché l'obiettivo principale di questa parte di progetto è far comunicare il radiocomando con la scheda STM32, si è deciso di alimentare il ricevitore e di studiare l'output proveniente da ogni canale utilizzando un oscilloscopio.

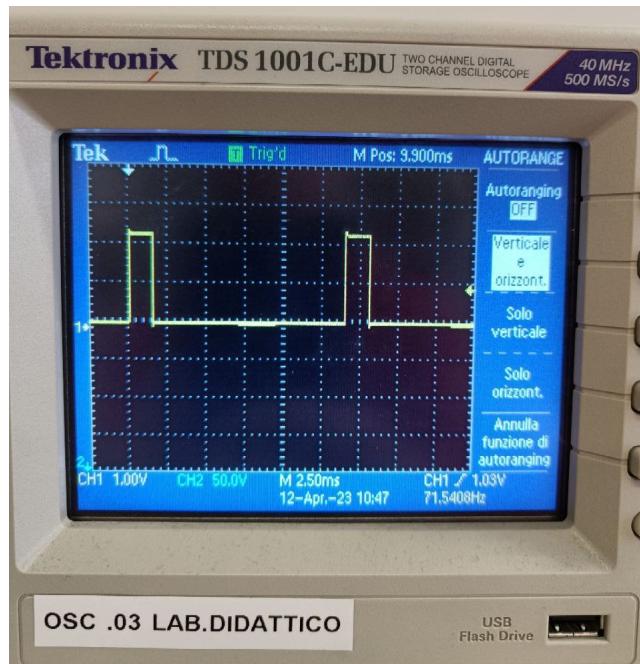


Figura 18: Oscilloscopio con segnale PWM

Come possibile osservare in foto, ogni canale restituisce un segnale PWM con un duty cycle variabile tra un valore minimo ed un valore massimo. Per cui, lo scopo è di ottenere la frequenza ed il duty cycle tutti i canali.

A tale fine, viene utilizzato un Timer in modalità **InputCapture** che è una particolare modalità pensata appositamente per permettere la misura della frequenza e/o dell'ampiezza degli impulsi di un segnale ad onda quadra esterno preso come input. In questa modalità il timer genera infatti un segnale di interrupt, catturato dall'Interrupt Service Routine HAL\_TIM\_IC\_CaptureCallback(), quando rileva un fronte di salita e/o di discesa sul segna-

le dato in ingresso. Il/i fronte/i da monitorare vengono specificati scegliendo il parametro "Polarity Section" del timer nel file ".ioc" tra i seguenti:

1. Rising Edge: fronte di salita
2. Falling Edge: fronte di discesa
3. Both Edges: fronte di salita e di discesa

Con l'aiuto di un oscilloscopio, si nota come tutti i segnali siano sincronizzati tra loro, ovvero hanno stessa frequenza e non sono sfasati tra loro. Questa osservazione è stata utilizzata per semplificare notevolmente la misura dei duty cycle dei vari canali. Come sarà più chiaro in seguito, avendo i segnali la stessa frequenza è sufficiente infatti misurare il periodo di un singolo canale per ottenere dall'ampiezza degli impulsi dei singoli canali i relativi duty cycle.

#### 4.1.1 Impostazioni Timer

Come facile intuire, per ottenere i valori desiderati dalla ricevente è necessario misurare il tempo, ciò avviene tramite l'utilizzo di un Timer. Per cui è necessario trovare delle impostazioni che rendano possibile ed affidabile la lettura di questi valori. Per l'utilizzo dell'InputCapture è stato deciso di utilizzare un Timer con un clock di 240 MHz, che viene diviso dal "Prescaler" settato a  $240 - 1$ . Facendo ciò, si opera con un clock di 1 MHz, che è la frequenza massima misurabile. Inoltre, poiché la frequenza minima leggibile è data da TIMCLOCK/COUNTERPERIOD, si è deciso di lasciare il "Counter Period" al massimo, ovvero 0xffffffff essendo un Timer a 32 bit.

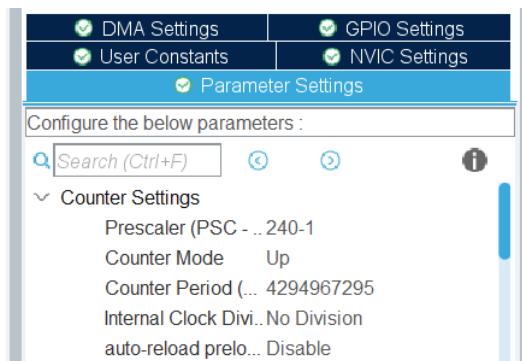


Figura 19: Parametri Timer

#### 4.1.2 Calcolo frequenza

Per poter misurare la frequenza, si setta la "Polarity Section" a **Rising Edge**. L'obiettivo è di misurare la differenza di tempo tra un fronte di salita ed il successivo ricavando il periodo. Con l'aiuto del flag **firstCaptured**, nel primo fronte si va a resettare il contatore, mentre nel secondo si va a leggere quanto tempo è passato dall'ultima volta che è stato azzerato con la funzione `HAL_TIM_ReadCapturedValue()`. Poiché si vuole operare con il tempo in secondi, è necessario dividere il valore ottenuto per il **refClock** che dipende dalle impostazioni del timer:  $refClock = TIMCLOCK/PRESCALER = 240000000/240 = 1000000$ . In questo modo si

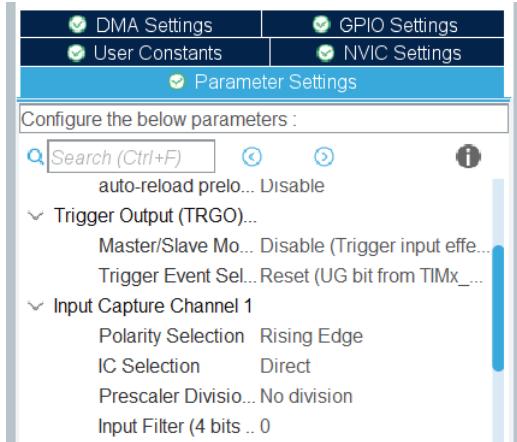


Figura 20: Polarity Section frequenza

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) // if the interrupt source is channel1
    {
        if (firstCaptured == 0) // if the first value is not captured
        {
            firstCaptured = 1; // set the first captured as true
            __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
        }
        else // if the first is already captured
        {
            val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read value

            frequency = (float)refClock/val;
            frequency = floorf(frequency * 100) / 100;

            firstCaptured = 0; // set it back to false
        }
    }
}

```

otterrebbe il periodo, quindi si calcola il reciproco trovando la frequenza in Hz che arrotondiamo al secondo decimale con la funzione floorf().

#### 4.1.3 Calcolo duty cycle

Per ottenere il duty cycle, si setta la "Polarity Section" a **Both Edges**. L'obiettivo è di misurare il tempo tra il fronte di salita e quello di discesa  $T_{ON}$  per poi calcolare il duty cycle con la formula  $\delta = T_{ON}/(T_{ON} + T_{OFF})$ . Il tempo  $T_{ON} + T_{OFF}$  non è altro che il periodo, ricavabile come spiegato sopra.

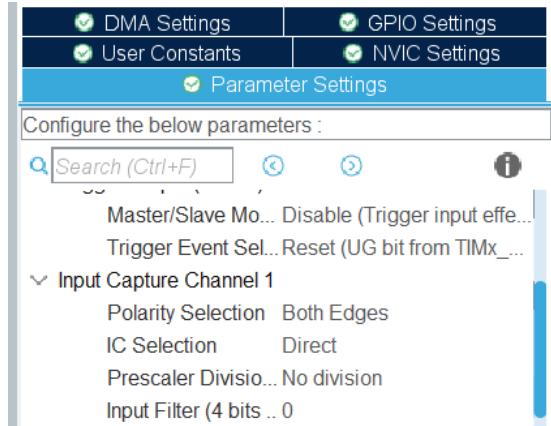


Figura 21: Polarity Section duty cycle

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) // if the interrupt source is channel1
    {
        if (firstCaptured == 0) // if the first value is not captured
        {
            firstCaptured = 1; // set the first captured as true
            __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
        }
        else // if the first is already captured
        {
            val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read value

            usWidth = (float) val/refClock;
            duty = usWidth*frequency*100;
            duty = floorf(duty*100) / 100;

            firstCaptured = 0;
        }
    }
}

```

Il duty cycle deve essere poi convertito in un valore utile a questo programma, per cui è necessario trovarne il valore minimo ed il valore massimo per ogni canale. Per farlo non basta spostare ai limiti la manopola del radiocomando, ma per trovare i valori estremi bisogna anche spostarsi con il relativo trigger, in quanto possono traslare il range del duty (vedi descrizione Hardware). Poiché i canali sono stati progettati tutti allo stesso modo, è sufficiente ricavare questi valori per un solo segnale, in quanto si è verificato in laboratorio che sono gli stessi per tutti gli altri. Si definiscono quindi le seguenti costanti:

```
#define MIN_RANGE_DUTY 6.2
#define MAX_RANGE_DUTY 15.4
```

## 4.2 Test Brushless

Prima di interfacciarsi al programma principale, abbiamo effettuato delle prove per sperimentare il funzionamento di un motore brushless a cui, forniamo alimentazione con un generatore e, come specificato nel capitolo dedicato all'Hardware, ci interfacciamo con l'utilizzo di un ESC (Electronic speed control). Quest'ultimo lavora con un segnale PWM con frequenza di 50 Hz. Dunque, si è utilizzato un Timer per la generazione di un segnale PWM a questa frequenza. Si è scelto il TIM3 avente  $TIMCLOCK = 240MHz$  settando  $PRESCALER = 240 - 1$  e  $COUNTERPERIOD = 20000 - 1$ . Variando il duty cycle è possibile controllare la velocità del motore e ciò è possibile tramite la seguente formula:

$$\delta = \frac{up \times 100}{MOTOR\_PWM\_SIGNAL\_PERIOD\_UP} = \frac{TIM \rightarrow CCR * 100}{TIM \rightarrow ARR} \quad (27)$$

Per poter avviare il motore, è necessaria prima la fase di armatura. Si capisce che il motore è stato armato grazie ad una sequenza di suoni che emette, di cui uno finale più lungo. Dalle prove effettuate dai gruppi precedenti, abbiamo scoperto che tale operazione avviene con un duty cycle del 4,5%. Per cui, avendo MOTOR\_PWM\_SIGNAL\_PERIOD\_UP settato a 20000 (periodo di 20 ms) utilizzando la formula scritta sopra, si ha:

$$up = \frac{20000 \times 4.5}{100} = 900ms \quad (28)$$

Effettuando delle prove, abbiamo scoperto che il motore inizia a girare a partire da un duty cycle del 5% che corrisponde quindi alla velocità minima, mentre, per motivi di sicurezza, si è deciso di impostare come velocità massima, quella ottenuta con un duty cycle del 10%. Nel caso in cui si decidesse di sostituire i motori scelti con altri meno potenti, si invita ad aumentare il valore di massimo duty. Utilizzando sempre la formula precedente, è possibile definire l'intervallo di "time up":

```
#define MIN_VALUE_MOTOR 1000      //5%
#define MAX_VALUE_MOTOR 2000      //10%
```

Per il bloccaggio del motore, abbiamo fornito inizialmente un  $\delta = 0\%$ , ottenendo un risultato non molto efficace. Dopo alcune prove, abbiamo scoperto come la frenata sia molto più performante con  $\delta = 4,5\%$  ovvero lo stesso duty cycle utilizzato per la fase di armatura.

<pre><b>void armMotor()</b> {     TIM3-&gt;CCR1 = 900; }</pre>	<pre><b>void stopMotor()</b> {     TIM3-&gt;CCR1 = 900; }</pre>
--	---

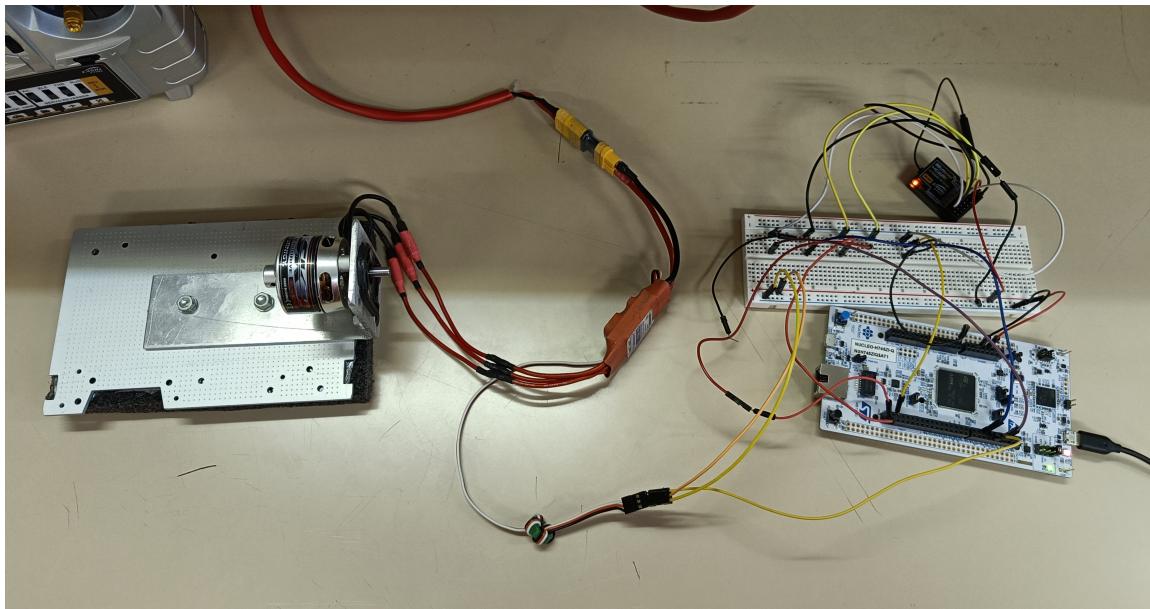


Figura 22: Prova su banco del Brushless

Azionando il motore, ci siamo accorti che dal generatore, non arrivava abbastanza corrente per controllarlo al meglio. Infatti, da un singolo canale del ricevitore, viene erogata al massimo una corrente di 3A. Per cui, grazie alle leggi di Kirchhoff alle correnti, abbiamo trovato la soluzione facendo un collegamento in parallelo come nella figura(23), ottenendo una corrente massima di  $3A + 3A = 6A$  che risulta più che sufficiente.



Figura 23: Collegamento in parallelo generatore

### 4.3 Struttura del codice

Nonostante la scheda STM32 utilizzata sia dual core, per motivi di semplicità si è deciso di utilizzare solo il core M4. I file principali sono:

1. main.c: il main del programma presenta delle operazioni di configurazione iniziali, sia per i motori che per la lettura dei valori dal ricevitore, per poi finire in un loop infinito in cui è possibile modificare lo stato del drone interagendo con il radiocomando.
2. ReceiverChannel.h: file di intestazione della libreria ReceiverChannel.c contenente prototipi di funzione, definizioni e strutture dati per immagazzinare le informazioni necessarie all'interazione con il ricevitore. Per la lettura della frequenza è stata definita la struct **ChannelForFrequency\_Data**, mentre per la lettura di un singolo canale è stata definita la struct **ChannelForDuty\_Data**. L'utilizzo delle funzioni mostrate in [24](#) è spiegato nei paragrafi successivi.

```
void Receiver_Init (ChannelForFrequency_Data* ch, ChannelForDuty_Data chDuty[], int ne);

float correctThrottle (float throttle, float maxThrottleOff);

float correctAngle(float angle, float error);
```

Figura 24: Prototipi funzione

```
typedef struct ChannelForDuty_Data{
    uint32_t val;
    int firstCaptured;
    float usWidth;
    float duty;
}ChannelForDuty_Data;

typedef struct ChannelForFrequency_Data{
    uint32_t val;
    int firstCaptured;
    int flagFirstFrequency;
    float frequency;
}ChannelForFrequency_Data;
```

Figura 25: Struct

```
ChannelForDuty_Data chDuty[NUMBER_CHANNELS];
ChannelForFrequency_Data chFrequency;
```

Figura 26: Variabili nel main.c

3. ReceiverChannel.c: libreria utilizzata per la comunicazione con il ricevitore. Il suo utilizzo principale avviene nell'inizializzazione delle variabili presenti nelle strutture dati descritte nell'header riportato sopra e nella correzione dell'errore presente nella lettura del throttle e dei valori angolari, nel caso in cui siano tendenti a zero. Lo scopo di quest'ultima funzionalità è di garantire maggiore precisione nel caso in cui il pilota voglia stabilizzare il drone in volo.

## 4.4 Gestione Interrupt

Il codice per la gestione degli Interrupt è molto simile a quello descritto nelle prove effettuate. La differenza nell'utilizzo delle strutture dati definite nel file "ReceiverChannel.h". Oltre a questo, il tutto avviene utilizzando una particolare gestione di due timer. I primi quattro canali del ricevitore vengono gestiti dal **TIM2**, mentre la frequenza ed il quinto canale vengono gestiti dal **TIM5**. Quindi il canale 5 del ricevitore è fisicamente collegato a 2 diversi pin della scheda, uno per leggerne il duty e uno per leggerne la frequenza (come riportato in figura 17). Poiché i segnali tra loro hanno stessa fase e frequenza, la "Polarity Section" per i canali del TIM2 non è la stessa per tutti, nonostante venga utilizzato solo per la misura dei duty cycle. Se fosse settata a "Both Edges" per ognuno (come descritto nelle prove effettuate), il contatore si resetterebbe ogni volta, per cui solo il duty cycle più piccolo verrebbe misurato correttamente. Quindi, si è deciso di utilizzare il primo canale come "segnale master" con polarità a "Both Edges", mentre i successivi tre hanno polarità a "Falling Edge". Il quinto canale del ricevitore, associato al quarto canale del TIM2, ha invece "Polarity Section" settata a "Both Edges" in quanto, come spiegato dettagliatamente nel paragrafo 4.5, non è disturbato dall'Interrupt per la misura della frequenza poiché questi non operano in contemporanea, ma come descritto dopo, quest'ultimo viene interrotto una volta ottenuto un valore accettabile.

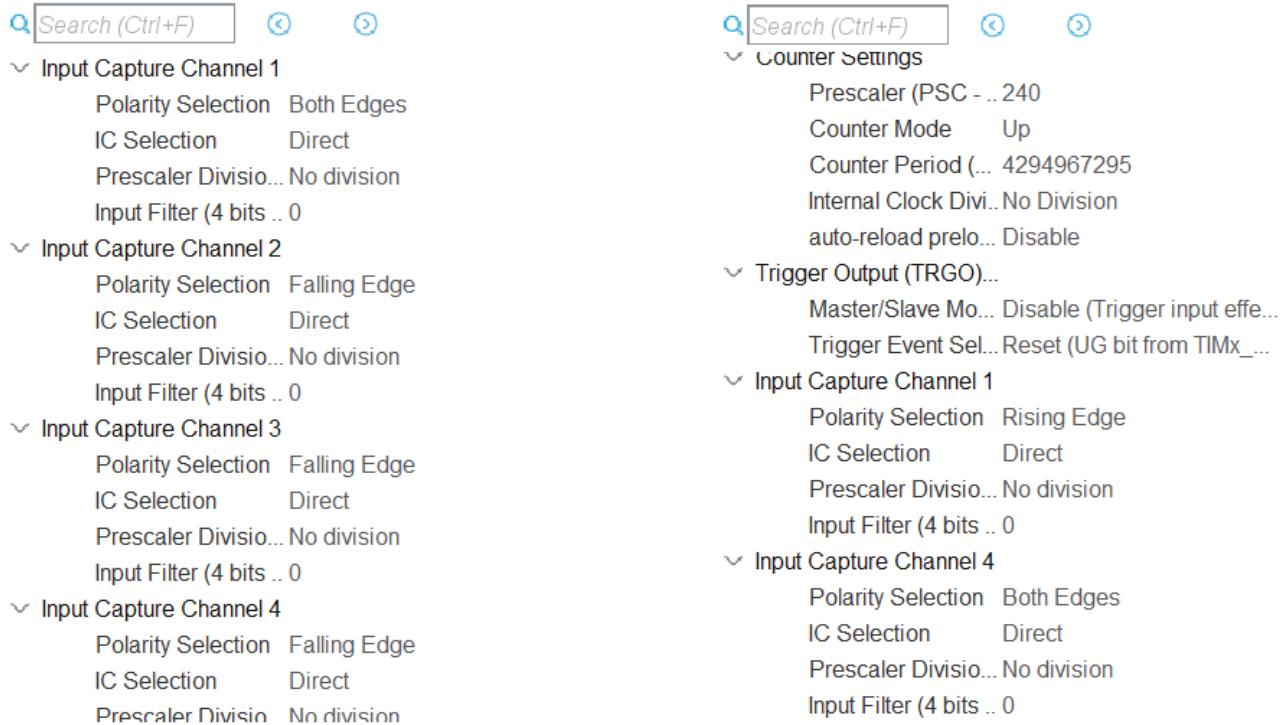


Figura 27: Timer 2 e 5 modalità Input Capture

Così facendo, l'evento del fronte di salita viene considerato solo per il primo canale, per poter resettare il contatore che è unico per il timer. Dopodiché ci si aspetta di avere quattro eventi sul fronte di discesa: ognuno per canale. A questo punto si procede banalmente come specificato nelle prove effettuate, quindi si legge il valore del contatore sul fronte di caduta per ricavare l'ampiezza del tempo  $T_{ON}$  e ottenere il valore in percentuale.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim == &htim5)
    {

        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) // if the interrupt source is channel1
        {
            if (chFrequency.firstCaptured == 0) // if the first value is not captured
            {
                chFrequency.firstCaptured = 1; // set the first captured as true
                __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
            }

            else // if the first is already captured
            {
                chFrequency.val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read value

                chFrequency.frequency = (float)refClock/chFrequency.val;
                chFrequency.frequency = floorf(chFrequency.frequency * 100) / 100;

                chFrequency.firstCaptured = 0; // set it back to false
                chFrequency.flagFirstFrequency = 0;
            }
        }

        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4) // if the interrupt source is channel4
        {
            if (chDuty[IC_CHANNEL5].firstCaptured == 0) // if the first value is not captured
            {
                chDuty[IC_CHANNEL5].firstCaptured = 1; // set the first captured as true
                __HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
            }

            else // if the first is already captured
            {
                chDuty[IC_CHANNEL5].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // read value

                chDuty[IC_CHANNEL5].usWidth = (float) (chDuty[IC_CHANNEL5].val) / refClock;
                chDuty[IC_CHANNEL5].duty = chDuty[IC_CHANNEL5].usWidth * chFrequency.frequency * 100;
                chDuty[IC_CHANNEL5].duty = floorf(chDuty[IC_CHANNEL5].duty * 100) / 100;

                chDuty[IC_CHANNEL5].firstCaptured = 0; // set it back to false
            }
        }
    }
}
```

```
}

else if(htim == &htim2)
{

    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1 ) // if the interrupt source is channel1
    {

        if (chDuty[IC_CHANNEL1].firstCaptured == 0) // if the first value is not captured
        {
            chDuty[IC_CHANNEL1].firstCaptured = 1; // set the first captured as true
            _HAL_TIM_SET_COUNTER(htim, 0); // reset the counter
        }

        else // if the first is already captured
        {
            chDuty[IC_CHANNEL1].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read value

            chDuty[IC_CHANNEL1].usWidth = (float)(chDuty[IC_CHANNEL1].val) / refClock;
            chDuty[IC_CHANNEL1].duty = (chDuty[IC_CHANNEL1].usWidth * chFrequency.frequency * 100);
            chDuty[IC_CHANNEL1].duty = floorf(chDuty[IC_CHANNEL1].duty * 100) / 100;

            chDuty[IC_CHANNEL1].firstCaptured = 0; // set it back to false
        }
    }

    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2 ) // if the interrupt source is channel2
    {
        chDuty[IC_CHANNEL2].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2); // read value

        chDuty[IC_CHANNEL2].usWidth = (float)(chDuty[IC_CHANNEL2].val) / refClock;
        chDuty[IC_CHANNEL2].duty = chDuty[IC_CHANNEL2].usWidth * chFrequency.frequency * 100 + ROLL_ERROR_CALIBRATION;
        chDuty[IC_CHANNEL2].duty = floorf(chDuty[IC_CHANNEL2].duty * 100) / 100;
    }

    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3 ) // if the interrupt source is channel3
    {
        chDuty[IC_CHANNEL3].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3); // read value

        chDuty[IC_CHANNEL3].usWidth = (float)(chDuty[IC_CHANNEL3].val) / refClock;
        chDuty[IC_CHANNEL3].duty = chDuty[IC_CHANNEL3].usWidth * chFrequency.frequency * 100;
        chDuty[IC_CHANNEL3].duty = floorf(chDuty[IC_CHANNEL3].duty * 100) / 100;
    }

    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4 ) // if the interrupt source is channel4
    {
        chDuty[IC_CHANNEL4].val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // read value

        chDuty[IC_CHANNEL4].usWidth = (float)(chDuty[IC_CHANNEL4].val) / refClock;
        chDuty[IC_CHANNEL4].duty = chDuty[IC_CHANNEL4].usWidth * chFrequency.frequency * 100;
        chDuty[IC_CHANNEL4].duty = floorf(chDuty[IC_CHANNEL4].duty * 100) / 100;
    }
}
}
```

Come possibile notare dal codice, nella lettura del duty cycle del secondo canale, rappresentato dal movimento orizzontale della manopola destra del radiocomando, è stato aggiunto un valore descritto dalla costante ROLL\_ERROR\_CALIBRATION, per correggere un errore hardware presente nel radiocomando. Infatti, sotto la manopola destra sono presenti delle molle tali per cui essa tende a tornare al centro se la si sposta. Ciò significa che, se la manopola destra viene lasciata autostabilizzarsi, ci si aspetta lo stesso valore di duty cycle dai i primi due canali (Pitch e Roll). In realtà si ha una piccola differenza e per una questione di precisione è stato preferito diminuirla il più possibile, altrimenti quando il pilota prova a stabilizzare il drone, rischia di avere un movimento minimo verso un solo asse.

## 4.5 Operazioni iniziali

Poiché nel segnale proveniente da ciascun canale del ricevitore varia solo il duty cycle nel tempo, mentre la frequenza rimane la stessa e costante per tutti, si è deciso di procedere nel seguente modo.

```
Receiver_Init(&chFrequency, chDuty, NUMBER_CHANNELS);

waitingForGettingFrequency();
jumpHalfPeriod(chFrequency.frequency);
startInputCaptureInterruptDutyCycle();

//si attende di ottenere i primi valori dal radiocomando, prima di entrare nel loop
waitingForFirstValues(chDuty, NUMBER_CHANNELS);

startPwmMotors();
```

Per prima cosa viene chiamata la funzione **Receiver\_Init** della libreria ReceiverChannel.c per inizializzare le variabili necessarie per la misura dei valori in InputCapture.

```
void Receiver_Init (ChannelForFrequency_Data* ch, ChannelForDuty_Data chDuty[], int ne)
{
    int i = 0;

    ch->firstCaptured = 0;

    for(i = 0; i<ne; i++)
    {
        chDuty[i].firstCaptured = 0;
        chDuty[i].duty = 0;
    }

    ch->flagFirstFrequency = 1;
    ch->frequency = 0;
}
```

Poi, viene avviato solo l'Interrupt per la misura della frequenza, per poi disattivarlo una volta ottenuto un valore accettabile. Per cui viene utilizzato il primo canale del TIM5 per la lettura della frequenza con "Polarity Section" settata a "Rising Edge", mentre viene utilizzato il quarto canale del timer con "Polarity Section" settata a "Both Edges". Il primo valore della frequenza ottenuto viene scartato per salvaguardarsi da possibili problemi hardware del ricevitore ottenibili con il tempo ed è stato effettuato utilizzando la flag **flagFirstFrequency**. L'attesa del secondo valore avviene tramite la funzione **waitForGettingFrequency**.

```
float waitingForGettingFrequency()
{
    HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_1);

    while (chFrequency.frequency == 0 || chFrequency.frequency > 100 || chFrequency.flagFirstFrequency == 1) { }

    HAL_TIM_IC_Stop_IT(&htim5, TIM_CHANNEL_1);

    return chFrequency.frequency;
}
```

Successivamente, prima di attivare gli Interrupt per la misura del duty cycle dei primi 5 canali, per garantire il corretto funzionamento è necessaria la funzione **jumpHalfPeriod**. Come specificato nel capitolo delle prove effettuate, per misurare il duty cycle, si analizza la differenza di tempo tra il fronte di salita e di discesa dell'onda quadra. Poiché gli Interrupt per la misura dei duty cycle vengono attivati subito dopo aver ottenuto la frequenza, il primo fronte che si avrebbe senza l'utilizzo di questa funzione, sarebbe quello di discesa. Il suo scopo è di "saltare mezzo periodo" per poter avere come primo Interrupt un fronte di salita, evitando il primo di discesa. Questo risultato è ottenuto semplicemente utilizzando la funzione **HAL\_Delay** per inserire un delay di durata pari a mezzo periodo, espresso in ms. Quindi, tale operazione è necessaria solo la prima volta, dopodiché il programma prosegue regolarmente.

```
void jumpHalfPeriod (float frequency)
{
    HAL_Delay(1000/(2*frequency));
}
```

Una volta evitato il primo fronte di discesa, si possono avviare gli Interrupt per la lettura dei duty cycle. Dopodichè, prima di entrare nel loop infinito, si attende che venga letto almeno un volta il valore per ogni canale con la funzione **waitingForFirstValues**.

```
void startInputCaptureInterruptDutyCycle()
{
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1); //ch1
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2); //ch2
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3); //ch3
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4); //ch4
    HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_4); //ch5
}

void waitingForFirstValues(ChannelForDuty_Data values[], int ne)
{
    int i;
    do
    {
        for(i=0; i<ne; i++)
        {
            if(values[i].duty == 0)
                break;
        }
    }while(i != ne);
}
```

## 4.6 Loop infinito

Terminate le operazioni iniziali, si può dire per certo di arrivare alla prima iterazione del loop avendo già letto almeno un valore del duty cycle per ogni canale della ricevente. Per cui, quello che bisogna fare è trasformare queste percentuali in informazioni utili, ovvero:

- **Pitch desiderato** espresso in gradi dal primo canale del radiocomando;
- **Roll desiderato** espresso in gradi dal secondo canale del radiocomando;
- **Throttle** con un valore da 0 a 100 dal terzo canale del radiocomando;
- **Yaw desiderato** espresso in radianti/secondo dal quarto canale del radiocomando;
- **Livello levetta** con un valore tra 1,2,3 dal quinto canale del radiocomando;

Per effettuare queste conversioni, utilizziamo la funzione **map**, per "mappare" un valore contenuto in un range, in un altro range.

Angolo	Valore Duty Cycle	Valore Angolo
Pitch	Da 6.2% a 15.4%	Da -30° a 30°
Roll	Da 6.2% a 15.4%	Da -30° a 30°
Yaw	Da 6.2% a 15.4%	Da -10rad/s a 10rad/s
Throttle	Da 6.2% a 15.4%	Da 0 a 100

```
desiredPitch = map(chDuty[IC_CHANNEL1].duty, MIN_RANGE_DUTY, MAX_RANGE_DUTY, -MAX_ANGLE, MAX_ANGLE);
desiredRoll = map(chDuty[IC_CHANNEL2].duty, MIN_RANGE_DUTY, MAX_RANGE_DUTY, -MAX_ANGLE, MAX_ANGLE) ;
throttle = map(chDuty[IC_CHANNEL3].duty, MIN_RANGE_DUTY, MAX_RANGE_DUTY, 0, MAX_THROTTLE);
desiredYaw = map(chDuty[IC_CHANNEL4].duty, MIN_RANGE_DUTY, MAX_RANGE_DUTY, -MAX_YAW_SPEED, MAX_YAW_SPEED);
level = getLevel(chDuty[IC_CHANNEL5].duty);
```

```
float map(float val, float from_src, float to_src, float from_dst, float to_dst)
{
    return (((to_dst-from_dst)/(to_src-from_src))*(val-from_src)) + from_dst;
}
```

Il duty cycle del quinto canale del radiocomando, essendo una levetta, può assumere solo 3 valori differenti, ovvero uno corrispondente ad ogni livello. Per questa motivazione, è stata utilizzata la funzione **getLevel** in cui viene controllato in quale range appartiene il duty cycle.

```
#define LEVEL1_DUTY 7
#define LEVEL2_DUTY 10
#define LEVEL3_DUTY 14
#define error_level 1
```

```
int getLevel(float dutyLevel)
{
    int level;

    if(dutyLevel > LEVEL1_DUTY - error_level && dutyLevel < LEVEL1_DUTY + error_level)
        level = 1;
    else if(dutyLevel > LEVEL2_DUTY - error_level && dutyLevel < LEVEL2_DUTY + error_level)
        level = 2;
    if(dutyLevel > LEVEL3_DUTY - error_level && dutyLevel < LEVEL3_DUTY + error_level)
        level = 3;

    return level;
}
```

Per avere maggiore stabilità, prima di prendere questi valori per buoni si è deciso di effettuare un'ulteriore correzione dell'errore oltre a quella descritta nel paragrafo dell'Interrupt. Come specificato prima, i canali 1,2 e 4, corrispondenti a Pitch, Roll e Yaw, hanno delle molle sotto le manopole del radiocomando per autocentrarsi quando non vengono spostate, in cui si aspetta un valore desiderato nullo, ma non è così. Infatti, quello che si ottiene sono dei valori che oscillano intorno allo zero e poiché stiamo parlando di valori generati dal software, si è deciso di utilizzare il metodo **correctAngle** della libreria dedicata al ricevitore. I valori delle costanti scritte sotto, sono stati scelti visualizzando in debug le variabili desiderate ottenute e sperimentando con il radiocomando.

```
#define MAX_ERROR_STABILIZATION 0.35
#define MAX_ERROR_YAW 0.2

desiredPitch = correctAngle(desiredPitch, MAX_ERROR_STABILIZATION);
desiredRoll = correctAngle(desiredRoll, MAX_ERROR_STABILIZATION);
desiredYaw = correctAngle(desiredYaw, MAX_ERROR_YAW);

float correctAngle(float angle, float error)
{
    if(angle > -error && angle < error)
        angle = 0;

    return angle;
}
```

Tali molle non sono inserite invece per il throttle, rappresentato dal movimento verticale della manopola sinistra. Tuttavia, è stata effettuata anche qui una correzione: se il throttle è sotto un certo valore dato dalla costante MAX\_THROTTLE\_OFF, allora viene settato a zero, altrimenti gli viene sottratto il valore della costante stessa, in modo tale che il valore positivo del throttle parta da 1 e non da MAX\_THROTTLE\_OFF+1. Il valore di tale costante è settato a 15 ed è stato scelto sperimentando il controllo del motore. Il suo scopo è quello di ridurre il rischio durante la fase di pilotaggio del drone, assumendo valore nullo fino ad un certo punto del posizionamento della manopola sull'asse verticale ed un valore compreso nell'intervallo [0, 85]. In questo modo, con il throttle al minimo, i motori non girano. Questa correzione viene effettuata sempre tramite l'intervento della libreria dedicata all'interazione con il radiocomando, con il metodo **correctThrottle**.

```
throttle = correctThrottle(throttle, MAX_THROTTLE_OFF);

float correctThrottle (float throttle, float maxThrottleOff)
{
    if(throttle < maxThrottleOff)
        throttle = 0;
    else
        throttle -= maxThrottleOff;

    return throttle;
}
```

Per concludere, rimane da analizzare il livello della levetta (canale 5) per controllare lo stato del drone. Se è in alto viene avviata l'armatura dei motori, in basso vengono bloccati i motori, mentre al centro vengono azionati.

```
switch(level)
{
    //arms
    case 1:
        armMotors();
        break;

    //move
    case 2:
        if(throttle == 0)
            stopMotors();
        else
            moveMotors(throttle);
        break;

    //stop
    case 3:
        stopMotors();
        break;
}
```

Come specificato nel capitolo 4.2, le funzioni **armMotors** e **stopMotors** fanno la stessa cosa, in quanto il bloccaggio dei motori è più efficace con un duty cycle del 4,5%. Per cui, potenzialmente è possibile armare e fermare i motori sia con la levetta in su che con la levetta in giù.

```
void armMotors()
{
    TIM3->CCR1 = 900;
    TIM3->CCR2 = 900;
    TIM3->CCR3 = 900;
    TIM3->CCR4 = 900;
}

void stopMotors()
{
    TIM3->CCR1 = 900;
    TIM3->CCR2 = 900;
    TIM3->CCR3 = 900;
    TIM3->CCR4 = 900;
}
```

Invece, **moveMotors** rispetto alle altre è una funzione provvisoria in quanto lo scopo di questo progetto è solo quello di testare il radiocomando. Di conseguenza, qualora si voglia riciclare questo codice per far volare il drone, è necessario modificare il contenuto di questa funzione per fare in modo che il velivolo si adatti ai valori desiderati inviati dal pilota. In questa parte di progetto, abbiamo invece controllato il motore seguendo le considerazioni effettuate nel capitolo 4.2, facendo variare quindi il duty cycle tra il valore minimo del 5% ed il valore massimo del 10%.

```
void moveMotors()
{
    TIM3->CCR1 = MIN_VALUE_MOTOR + (throttle/100)*(MAX_VALUE_MOTOR - MIN_VALUE_MOTOR);
}
```

## 5 Conclusione

In questa relazione si spiega come è possibile ricevere ed analizzare dati da un qualsiasi radiocomando a più canali con l'utilizzo di una scheda STM32. Poiché c'era la necessità di sperimentarne il funzionamento, abbiamo deciso anche di controllare uno dei quattro motori brushless che vanno montati sul drone, alimentandolo con un generatore. La prima parte consiste nell'ottenere il duty cycle dei canali della ricevente collegata alla scheda, utilizzando la modalità InputCapture del timer. Dopo di che, abbiamo trasformato i valori ottenuti in valori desiderati relativi allo stato del drone, di cui sarà possibile controllarne il moto in sviluppi futuri utilizzando i primi quattro canali del radiocomando, mentre il quinto canale rappresentato dalla levetta serve solo per attivarlo/disattivarlo. Infatti, una volta alimentato il motore, con la levetta in su è possibile armarlo, con la levetta in mezzo è possibile controllarlo e con la levetta in giù è possibile fermarlo. Come specificato all'interno della relazione, è possibile armare e fermare il motore sia con la levetta in alto che in basso, in quanto il segnale elettrico utilizzato per questi due scopi è lo stesso. Per pilotare il drone utilizzando questo software, è necessario modificare la funzione **moveMotors** che, per questo progetto, viene utilizzata per il controllo di un solo motore brushless senza badare ai valori di Pitch, Roll e Yaw desiderati, ma basandosi esclusivamente sul valore della levetta e del throttle (canale 3).

## 6 Bibliografia e Sitografia

- Teppo Luukkonen, Modelling and control of quadcopter, School of Science, Espoo, August 22, 2011
- <https://deepbluembedded.com/stm32-pwm-example-timer-pwm-mode-tutorial>
- <https://deepbluembedded.com/stm32-input-capture-frequency-measurement-example-timer-input-capture-mode>
- <https://controllerstech.com/input-capture-in-stm32>
- Slide fornite dal docente