

Università Politecnica delle Marche

Corso di Laurea
Ingegneria Informatica e dell'Automazione



Laboratorio di Automazione Relazione Altimetro e controllo Motori

Docente: Prof. Bonci Andrea

A.A 2019/20

Studenti:

Matteo Ferretti - 1083630
Alessandro De Toni - 1082031
Angelo D'Agostino Bonomi - 1082444
Simone Di Rado - 1081639

Indice

1 INTRODUZIONE	4
1.1 Cos'è un Quadricottero?	4
1.1.1 Cenni Storici	4
1.1.2 Fisica di un quadrirotore	5
2 MODELLO MATEMATICO QUADRIROTORE	6
2.1 Sistemi di riferimento e grandezze fondamentali	6
2.2 Caratterizzazione dinamica	8
2.3 Modello di controllo	9
2.3.1 Controllo dell'assetto	10
2.4 Controllori PID	11
2.4.1 Conversione dell'uscita di controllo in velocità di rotazione	11
2.5 Fonti	12
2.6 Calcolo dei coefficienti	13
3 HARDWARE	14
3.1 Scheda Renesas YRDK RX63N	14
3.2 Motori	15
3.3 ESC	16
3.3.1 Cos'è e come funziona	16
3.3.2 Modalità di uso	17
3.3.3 Come programmare un ESC	17
3.4 Eliche	18
3.5 Batteria	18
3.5.1 Caratteristiche	18
3.5.2 Lipo Saver	19
3.6 Telaio	19
3.7 Scheda di Potenza	20
3.7.1 Scheda 1 _a Versione	20
3.7.2 Scheda 2° Versione	21
3.7.2.1 Regolatore di tensione lineare a 5V (LM7805)	21
3.7.2.2 Regolatore di tensione lineare a 3.3V (UA78M33)	23
3.7.2.3 Interruttore a MOSFET comandato tramite Opto-commutatore	23
3.7.2.4 Adattatore di livello per segnali PWM	26
3.8 ALTIMETRO	28
3.8.1 Posizionamento	29
3.9 IMU	30
3.9.1 Posizionamento	30
3.10 CONNETTORI	31
4 SOFTWARE	34
4.1 Struttura generale	34
4.1.1 Generiche	35
4.1.2 Altimeter	36
4.1.3 IMU	38
4.2 PID	39
4.3 Funzionamento del codice	40

4.3.1	Fase di Inizializzazione	40
4.3.2	Fase di esecuzione Real-Time	42
5	Segnale PWM	46
6	Procedura per l'attivazione del Drone	48
7	SVILUPPI FUTURI	50

1 INTRODUZIONE

L'obiettivo primario di questa relazione è focalizzare l'attenzione su alcuni aspetti fondamentali dell'implementazione di un controllo automatico, atto a stabilizzare un sistema ad ala rotante quale è un quadrirotore.

I punti fondamentali trattati sono rispettivamente il processo di elaborazione dei dati sensoriali, che ci consente di determinare l'orientamento del corpo nello spazio tridimensionale attraverso una loro opportuna fusione, ed il sistema di generazione della propulsione del velivolo ottenuta per mezzo di eliche a motore.

1.1 Cos'è un Quadricottero?

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 (Fig.1.1), e che, oggi giorno, costituisce uno dei velivoli più popolari. Esso, fu per la prima volta 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.

Recentemente con il diffondersi di veicoli denominati UAV (Unmanned Aerial Vehicle) questi sistemi hanno trovato svariate applicazioni in ambito civile tra cui: la didattica, l'hobbismo 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.



Figura 1: Drone quadrotor DJI Mavic pro.

1.1.2 Fisica di un quadrirotore

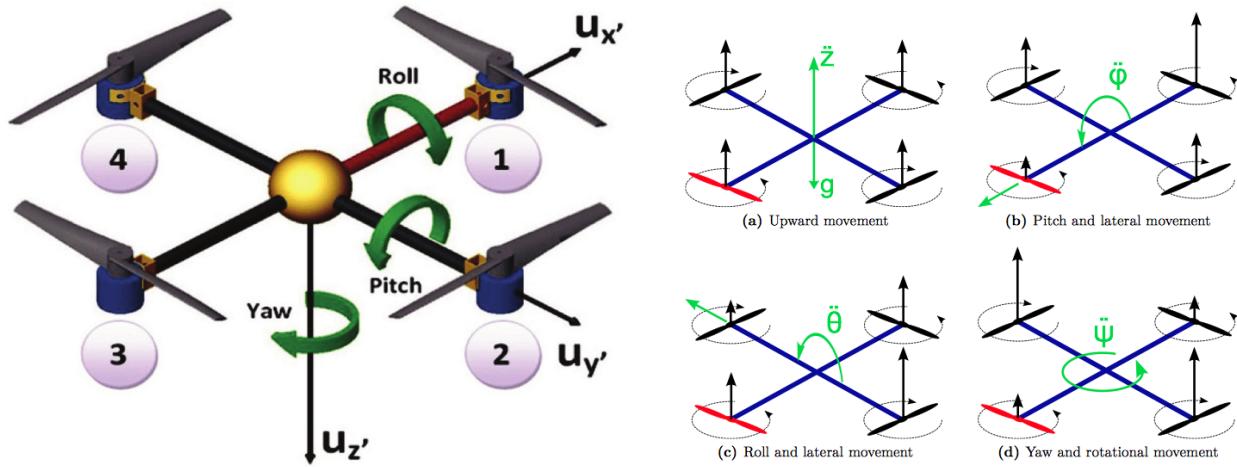


Figura 2: Movimenti del Drone che si ottengono modificando la velocità di determinati motori.

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 (Fig 2), 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. In riferimento alla Fig. 2(b), 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 Fig. 2(c), dove si genera l'angolo di rollio, inglese, roll.

In basso a destra, Fig. 2(d), 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.

2 MODELLO MATEMATICO QUADRIROTORE

2.1 Sistemi di riferimento e grandezze fondamentali

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

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

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

dove:

E_I sistema di riferimento inerziale assoluto (suolo);

E_B sistema di riferimento inerziale solidale al corpo del drone

L'origine di E_B è posta nel centro di massa.

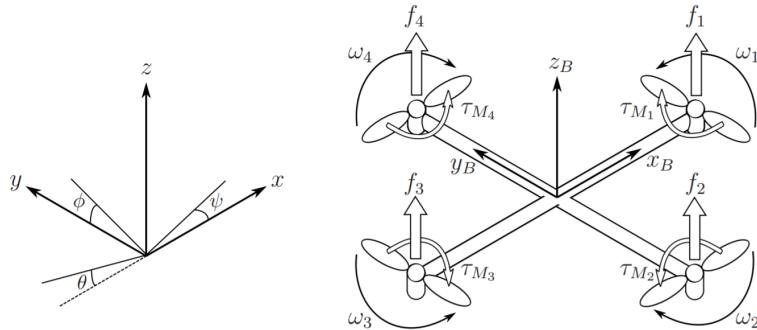


Figura 3: Rappresentazioni (rispettivamente) del sistema inerziale assoluto e di quello solidale al quadricottero

Inoltre, definiamo ξ la posizione assoluta del drone in E_i e η l'assetto del drone in E_i . Observando 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.

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

Ora, possiamo introdurre le velocità lineari \mathbf{V}_B e angolari $\boldsymbol{\nu}$ viste dal sistema E_B :

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

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

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

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

infatti:

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

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 $I_{xx} = I_{yy}$.

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

2.2 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 \mathbf{F}_i tale che:

$$\mathbf{F}_i = b\omega_i^2$$

F_i 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

$$T = \sum_{i=1}^4 \mathbf{F}_i = k \sum_{i=1}^4 \omega_i^2, \quad \mathbf{T}_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

L'altro effetto dato dalla rotazione è la creazione di un momento torcente $\boldsymbol{\tau}_M$ attorno all'asse del motore pari a:

$$\boldsymbol{\tau}_M = d\omega_i^2 + I_M \dot{\omega}_i$$

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 Figura 3 alla pagina 6, i motori ruotano a 2 e 2 in senso opposto, in questo modo, i momenti torcenti da essi generati si annullano fra loro se $\omega_1 = \omega_2 = \omega_3 = \omega_4$, in caso contrario si ha una variazione dell'angolo di yaw ψ .

Formulando matematicamente questi concetti si ottiene, $\boldsymbol{\tau}_B$:

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

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

2.3 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:

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

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).

2.3.1 Controllo dell'assetto

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

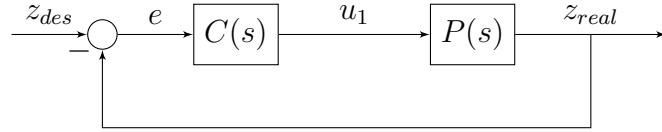


Figura 4: Loop di controllo per l'altitudine z

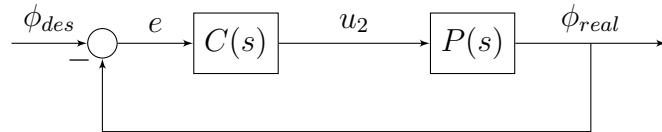


Figura 5: Loop di controllo per l'angolo di pitch ϕ

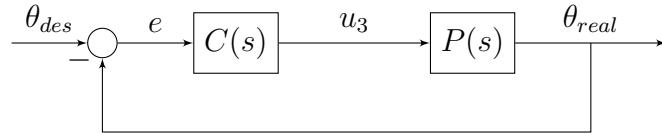


Figura 6: Loop di controllo per l'angolo di roll θ

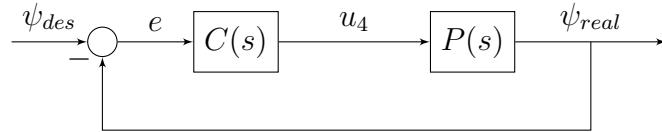


Figura 7: Loop di controllo per l'angolo di yaw ψ

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.

2.4 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:

$$\begin{aligned} u_1 &= K_{p,z}(z_{des} - z_{real}) + K_{D,z} \frac{d(z_{des} - z_{real})}{dt} + K_I \int (z_{des} - z_{real}) dt \\ u_2 &= K_{p,\phi}(\phi_{des} - \phi_{real}) - K_{D,\phi}(\dot{\phi}_{real}) \\ u_3 &= K_{p,\theta}(\theta_{des} - \theta_{real}) - K_{D,\theta}(\dot{\theta}_{real}) \\ u_4 &= K_{p,\psi}(\psi_{des} - \psi_{real}) - K_{D,\psi}(\dot{\psi}_{real}) \end{aligned}$$

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.

2.4.1 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 2.3 e 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:

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

Nella sezione 4.2 vedremo come le velocità appena definite saranno convertite in un segnale PWM.

2.5 Fonti

- Teppo Luukkonen - "Modelling and control of quadcopter", Aalto University School of Science
- Hossein Bolandi, Mohammad Rezaei, Reza Mohsenipour, Hossein Nemati, Seed Majid Smailzadeh - "Attitude Control of a Quadrotor with Optimized PID Controller"
- Slide fornite dal docente

2.6 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(*Per maggiori dettagli, consultare la sezione 3.5 - Batteria*). 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:

$$b_3 = \frac{T_3}{\omega_3^2} = \frac{1,52}{(1143,53)^2} = 1,162 \times 10^{-6}$$

- Caso batteria Turnigy A-Spec - quattro celle:

$$b_4 = \frac{T_4}{\omega_4^2} = \frac{2,07}{(1334,12)^2} = 1,163 \times 10^{-6}$$

- 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_3 e ω_4 rappresentano la velocità del motore, anch'essa differente a seconda della batteria.

Per il coefficiente 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)

E' possibile calcolare d solamente attraverso l'inversa di tale formula, cosa impossibile nel nostro caso poichè 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.

3 HARDWARE

3.1 Scheda Renesas YRDK RX63N

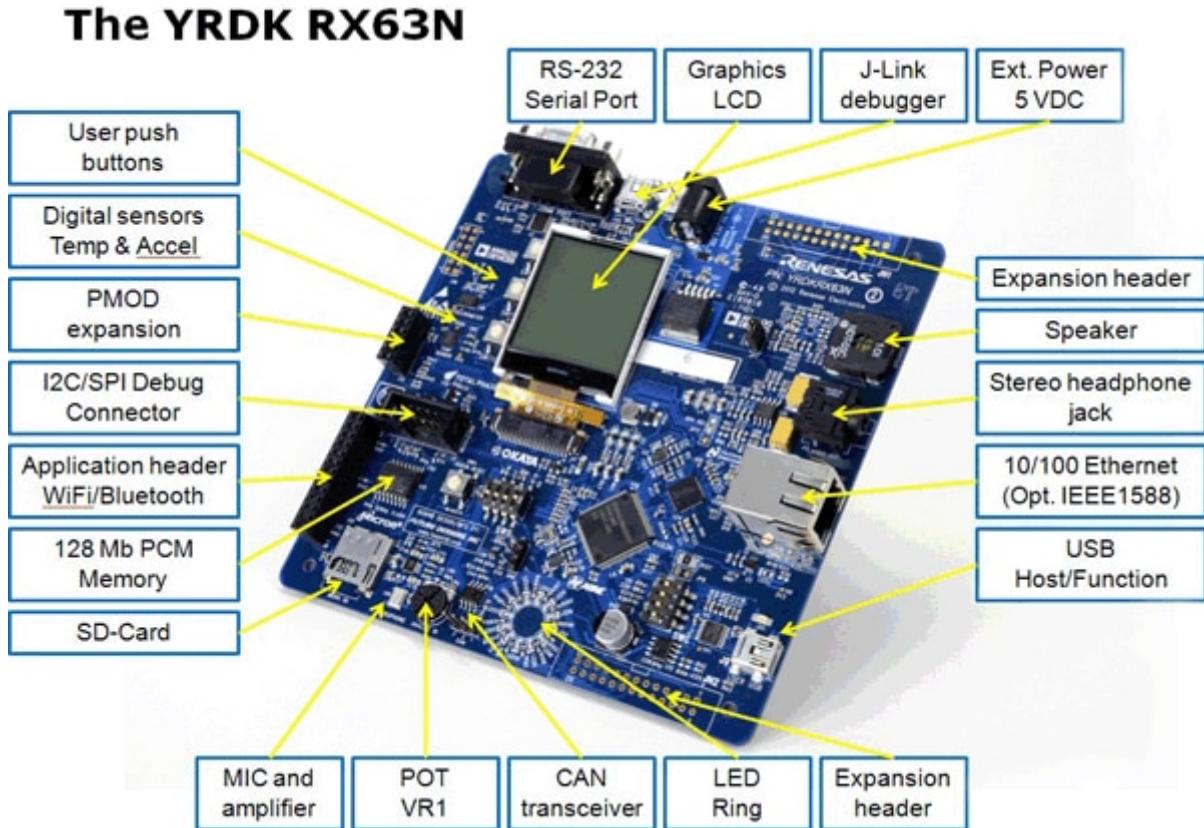


Figura 8: Scheda Renesas TRDK RX63N

Per il progetto di laboratorio è stata utilizzata la scheda “Demonstration Kit RX63N della Renesas” .

La scheda monta a bordo, oltre ad un microcontrollore a 32 bit, chip di memoria esterna di tipo flash SDRAM, oscillatori vari (tra cui l'oscillatore che fornisce il clock alla CPU) , un' interfaccia seriale RS232 e I2C con sda e scl, un piccolo display LCD, ed una serie di dispositivi quali pulsanti, led, buzzer e potenziometri utilizzabili a discrezione del programmatore.

Sono inoltre presenti connettori (JN1, JN2, ecc.), collegati alla varie porte del microcontrollore, con i quali è possibile interfacciarsi facilmente con l'esterno e dai quali si può prelevare o fornire segnali di vario tipo. Tra i vari tipi di segnali con i quali si può interagire con la scheda troviamo: segnali digitali (0– 3.3V), segnali analogici leggibili tramite ADC interno, segnali PWM, segnali di tipo I2C, segnali di tipo seriale standard RS232.

3.2 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).

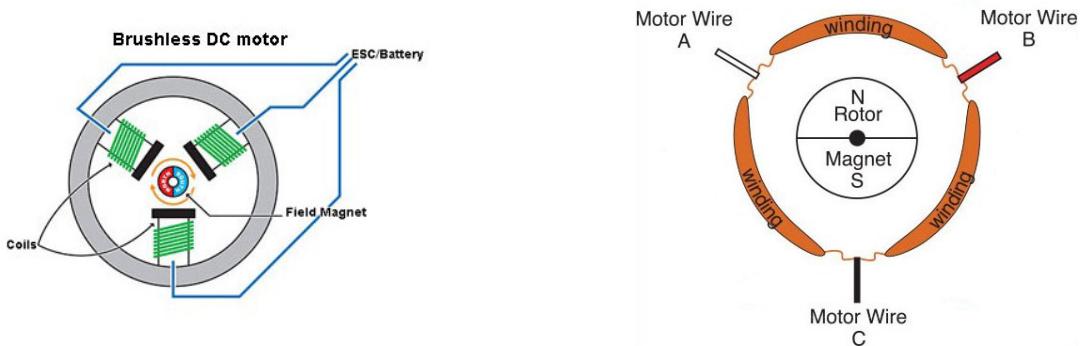


Figura 9: Brushless motor internal circuit.

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:

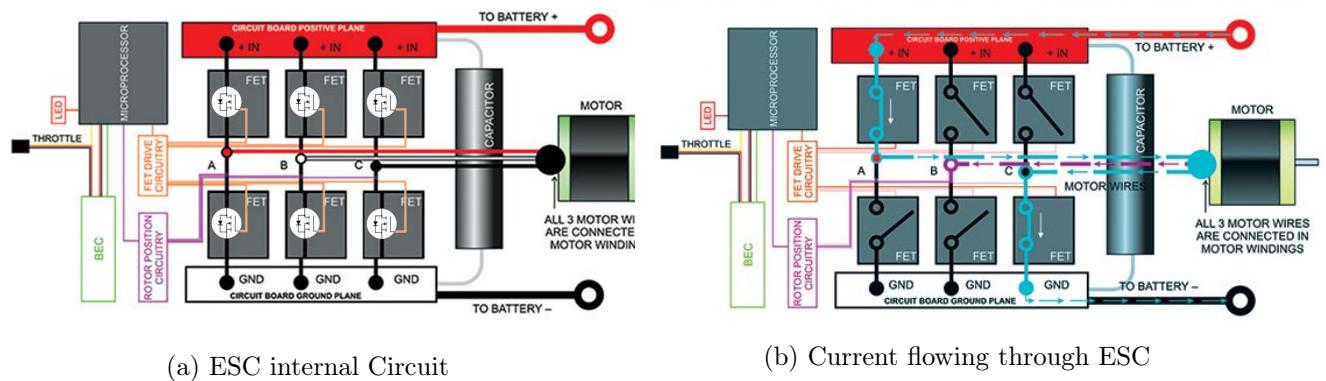
RPM	910kv
Fasi	3
Massima corrente	25.5A
Potenza massima	370W a 15V
Peso	102g
Batteria	2 ~ 4 Cell /7.4~ 14.8V
Diametro del pozzo	5mm
No corrente di carico	1.5A
Dimensioni	35x36mm
Spinta max	1050g
Dimensione prop.	7.4V/12x5 14.8V/10x7
Resistenza interna	0,063 Ohm

3.3 ESC

3.3.1 Cos'è e come funziona

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.



Analogamente un ESC è provvisto di un connettore standardizzato da cui è possibile fornire i comandi al motore direttamente dalla ricevente radio o mediante un opportuno collegamento ad una scheda di controllo.

Oltre a semplificare il comando, il connettore, chiamato **BEC** (*Battery Eliminator Circuit*) ovvero circuito elimina batteria, ha la potenzialità di fornire ai componenti, che vi sono connessi, una tensione continua e stabilizzata ricavata dalla batteria centrale che alimenta i motori. Ad esempio, nei due casi sopra illustrati, l'ESC consente di alimentare o la ricevente o la scheda di volo.

Anche questo è un risultato pensato appositamente per le applicazioni aereo-modellistiche in cui le problematiche di peso e dimensioni non sono trascurabili.

Per controllare i motori sono stati acquistati 4 ESC Turnigy PLUSH-30A con le seguenti caratteristiche:

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

3.3.2 Modalità di uso

Tali dispositivi possono essere programmati mediante una scheda di programmazione apposita che ne permette il settaggio delle caratteristiche di:

Freno (on/off): rappresenta la modalità di frenata dei motori. Quando è on i motori appena raggiunta la posizione minima si fermano immediatamente, altrimenti rallentano naturalmente

Tipologia di Cut-Off (soft-cut/cut-off): consente di ridurre immediatamente la potenza in uscita dei motori una volta raggiunta la soglia imposta al seguente punto, se settato su cut-off, altrimenti la riduzione è graduale.

Tensione di Cut-Off (low/middle/high): definisce il voltaggio al di sotto del quale i motori non vengono azionati. Per batterie da 2 a 4 celle i valori sono 2.85V/3.15V/3.30V.

Modalità di avvio (normal/soft/very soft): in base al parametro impostato, la velocità dei motori raggiungerà il valore massimo in un tempo più o meno breve. Normal è solitamente adatto agli aeroplani, mentre soft e very soft vengono generalmente usati per i velivoli a decollo verticale.

Modalità di timing (low/middle/high)

Numero di celle della batteria (auto/2/3/4): determina le celle del tipo di batteria che si sta usando.

3.3.3 Come programmare un ESC

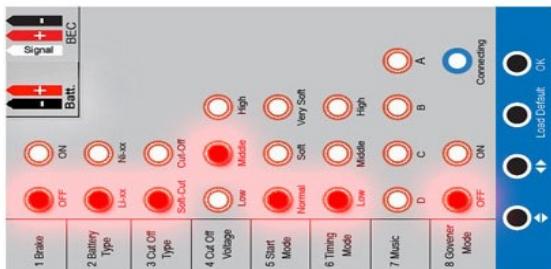


Figura 11: Scheda di programmazione ESC.

Per programmare l'ESC bisogna collegarlo alla scheda di programmazione dal cavo BEC e alla batteria, quindi settare i led come da figura 11 tramite gli appositi pulsanti. All'accensione, per far partire i motori, è necessario eseguire una procedura detta "Armamento". Tale procedura consiste nell'inviare un segnale PWM sulla linea dati (BEC) dell'ESC prima di fornire la linea di alimentazione dei 14.6 V della batteria. Ciò ha comportato la necessità di creare un circuito interruttore a MOSFET (descritto a 3.7.2.3) che desse il tempo al processore di generare il giusto segnale PWM. Quando la procedura di armamento risulta eseguita correttamente, gli ESC emetteranno un breve segnale audio melodico, mentre se viene riscontrato un problema, emetteranno due BEEP rapidi ogni 2 secondi.



(a) Motore Turnigy d3536/9 910kv



(b) ESC Turnigy plush 30a

3.4 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:

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

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

3.5 Batteria

3.5.1 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.

3.5.2 Lipo Saver



Uno dei problemi principali nell'utilizzo delle batterie LiPo è il controllo della carica minima.

Questo tipo di batteria infatti non deve raggiungere una tensione (per cella) inferiore ai 2.7V, per evitare il danneggiamento della stessa.

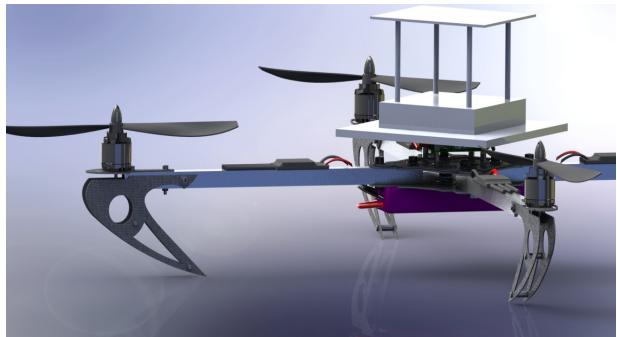
Per questo è stato adottata una piccola scheda detta Lipo Saver, la quale, mediante un diodo led ci avverte quando la tensione scende al di sotto dei 3V per cella.

3.6 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.



(a) Render1



(b) Render2

Figura 13: Un primo render del concept 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 (fig. 1.1) 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 Scheda di Potenza

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

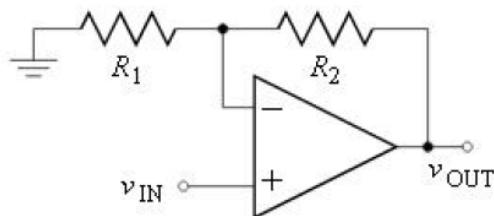
Per il funzionamento dei motori, non è stato necessario inserire una scheda “alimentatore”, in quanto l’alimentazione degli stessi proviene direttamente dalla batteria e la sua regolazione è comandata dagli ESC.

Nel corso del progetto sono state sviluppate e realizzate 2 differenti versioni della scheda. Esse sono state progettate al pc tramite il software Cadsoft Eagle, quindi sono state sviluppate su pcb da un’azienda esterna.

3.7.1 Scheda 1_a Versione

La prima versione della scheda elettronica è composta essenzialmente da 4 circuiti:

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



La differenza principale tra questa versione e la seguente sono i blocchi contenenti gli adattatori di livello: nella prima scheda sono stati utilizzati adattatori di livello, il cui principio di funzionamento si basa su amplificatori operazionali in configurazione non invertente, per adattare i segnali in ingresso provenienti dal microcontrollore

(0 – 3.3V) a livelli riconoscibili dai circuiti a valle (0 – 5V).

La tensione di uscita Vout è legata a quella di ingresso secondo la relazione:

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

In questo caso, quindi, settando opportunamente i valori delle resistenze, avremo in uscita i livelli logici da noi richiesti.

A causa di alcuni errori nella progettazione ed alla mancanza di sistemi di sicurezza (che hanno causato la rottura di un pin di output della scheda Renesas) si è deciso di riprogettare interamente la scheda, modificandone il layout e la struttura.

Si è quindi realizzata la 2° Versione, attualmente in uso.

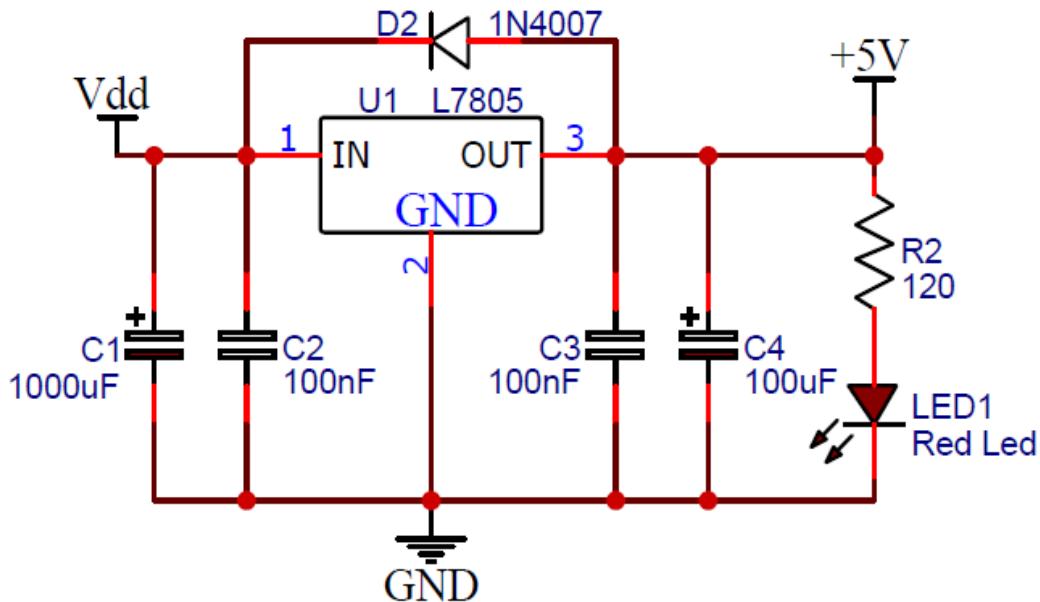
3.7.2 Scheda 2° Versione

La seconda versione della scheda, sostanzialmente è composta dagli stessi circuiti della prima, tuttavia introduce degli importanti miglioramenti, quali l'inserimento di sistemi di protezione elettrica mediante optoisolatori nei circuiti adattatori di livello, la correzione di alcuni errori di progettazione, un miglior posizionamento dei connettori, dei fori per le viti di fissaggio ed il posizionamento al centro scheda dell'IMU, indispensabile per il corretto funzionamento del velivolo.

È stato inoltre deciso di eliminare le alette di raffreddamento dei MOSFET, in quanto sono stati utilizzati componenti di potenza sovradimensionati che non necessitano di dissipatori, in modo da alleggerire il peso del velivolo stesso.

In questo capitolo verranno quindi descritti, in maniera approfondita i vari blocchi che compongono la scheda.

3.7.2.1 Regolatore di tensione lineare a 5V (LM7805)



Il regolatore di tensione lineare, qui utilizzato, è un semplice circuito alimentatore che permette, data la tensione di ingresso di 11.1V proveniente dalla batteria (VDD), di produrre una tensione di uscita continua stabilizzata di 5V (VCC).

Nel nostro progetto, viene utilizzato per fornire l'alimentazione alla scheda di controllo Renesas.

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

Osserviamo più in dettaglio i componenti che compongono il circuito:

- **Regolatore LM7805:**

L'LM7805 è un regolatore di tensione positiva, a tre terminali, avente le seguenti caratteristiche:

Tensione di ingresso	compresa tra 7V e 20 V continui
Tensione di uscita	+5V ($\pm 0.25V$)
Corrente di uscita massima	1°
Protezione	Protezione termica per sovraccarico e Protezione interna per corto-circuito con limitazione della corrente Package TO220

- **Condensatori: 1000uF-25V, 100uF-35V, 100nF**

Il circuito dispone di 4 condensatori:

C1 è un condensatore elettrolitico di filtro, che viene sempre utilizzato in prossimità del regolatore (valore compreso tra 10uF e 4700uF).

C2 e C3 sono condensatori al poliestere, utilizzati per evitare auto oscillazioni (valore compreso tra 1nF e 100nF).

C4 è un condensatore elettrolitico, utilizzato per ridurre il ripple presente in uscita dal regolatore, rendendo la tensione più stabile (valore compreso tra 1uF e 470uF).

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

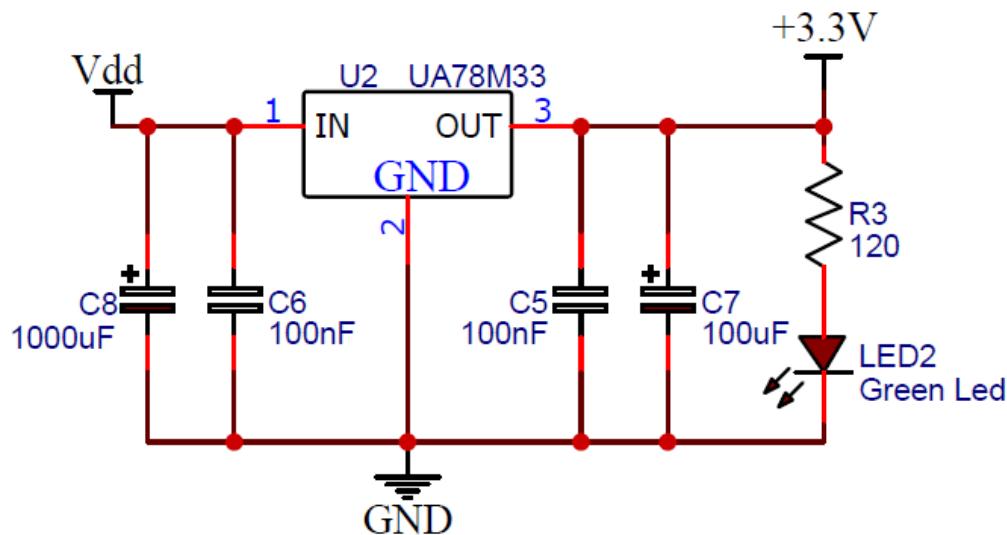
- **Diodo: 1N4007**

Il diodo D2, posto tra l'uscita e l'ingresso, viene utilizzato per proteggere il regolatore ogni qualvolta si spegne il circuito alimentatore. Senza tale diodo, la tensione immagazzinata dal condensatore C4 si scaricherebbe in senso inverso all'interno dell'integrato, cioè dall'uscita verso l'ingresso, danneggiandolo.

- **Led e Resistenza**

Il led da 3mm e la relativa resistenza vengono utilizzati per un semplice controllo visivo del funzionamento del circuito alimentatore.

3.7.2.2 Regolatore di tensione lineare a 3.3V (UA78M33)



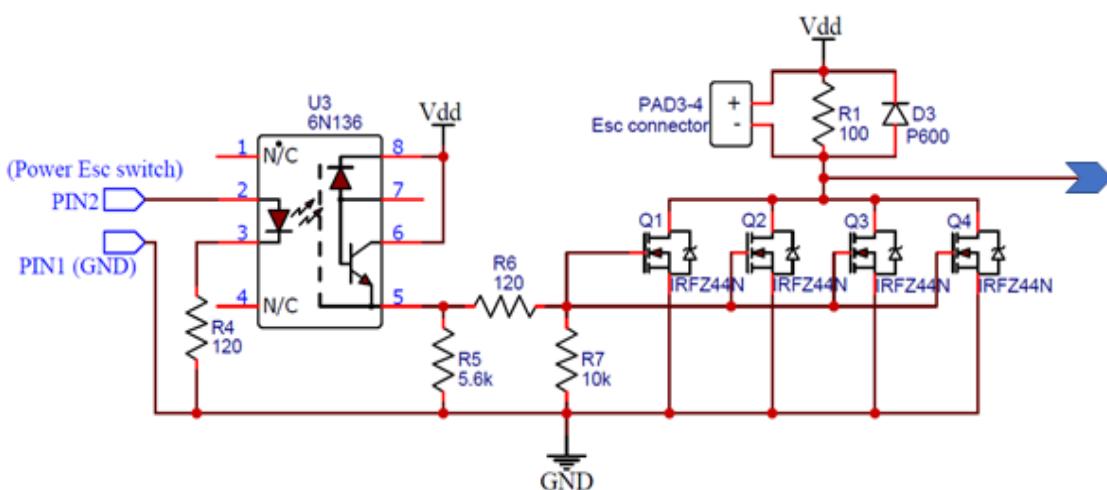
Il circuito regolatore di tensione lineare a 3.3V è del tutto analogo al precedente, fatta eccezione per il circuito integrato utilizzato che in questo caso è l' UA78M33.

Viene utilizzato per fornire una tensione continua stabilizzata di 3.3 V all' IMU e all'Altimetro.

- **Regolatore: UA78M33**

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

3.7.2.3 Interruttore a MOSFET comandato tramite Opto-commutatore



Il circuito qui descritto, è un interruttore a stato solido, comandato da un pin digitale della scheda di controllo Renesas tramite un adattatore di livello 3.3V – 5V.

L'adattatore si è reso necessario in quanto, come spiegato nel successivo capitolo "MOSFET", l'interruttore a stato solido necessita di una tensione di gate VGS di almeno 4.5V, valore non raggiungibile dai pin digitali del microcontrollore (la cui tensione massima di output è 3.3V).

Tale circuito, agendo direttamente sull'alimentazione degli ESC, è stato sviluppato per permettere una corretta accensione degli stessi. Ciò si è reso necessario in quanto, come descritto nel capitolo 3.3, studiando i datasheets, è emerso che per un corretto funzionamento bisognava fornire l'alimentazione principale solo dopo aver portato un segnale PWM di un determinato periodo sulla linea dati (armamento dei motori); per questo è stato deciso di realizzare questo interruttore, facilmente controllabile via software, che permette di interrompere e far circolare corrente sulla linea di alimentazione degli ESC.

Il circuito è composto da:

- **MOSFET: IRFZ44N**

I componenti principali del circuito interruttore sono i 4 MOSFET di potenza.

Il tipo di MOSFET è stato scelto dopo alcune prove, in base alle seguenti caratteristiche richieste dal progetto:

Alta potenza dissipabile, con conseguente corrente Id maggiore di 10A in grado di sostenere l'elevata corrente assorbita dai motori (misurata sperimentalmente e risultata superiore ai 4° per motore a piena potenza).

Bassa resistenza interna, in modo da avere minori cadute di tensioni a correnti elevate.

Facilità di parallelizzazione.

Possibilità di controllare il gate con segnali a 5V.

Il componente che si adatta perfettamente alle nostre esigenze è l'IRFZ44N, in quanto ha le seguenti caratteristiche:

Corrente massima Id	49A
Potenza massima diss.	94W
Resistenza interna RDS	17.5mΩ
Tensione di controllo Vgs	(con Vds=11.1V e Id≈ 15A) maggiore di 4.5V.
Altro	Facilmente parallelizzabile, senza aggiunta di altri componenti.



Figura 14: MOSFET IRFZ44N.

Come ogni MOSFET, anche questo è dotato di una resistenza interna di canale che, in caso di alte correnti (come nella situazione in oggetto), porta a cadute di tensione dell'ordine di qualche volt.

Dopo alcuni test si è verificato che, ad alte velocità di rotazione dei motori con conseguenti elevate correnti assorbite, un solo transistor portava a cadute di potenziale troppo elevate che causavano l'abbassamento della tensione di alimentazione degli ESC fino al di sotto dei 9V, tensione minima necessaria per il corretto funzionamento degli stessi. Si è quindi optato per inserire 4 MOSFET in parallelo anziché 1, in modo da ridurre la resistenza interna equivalente ad un quarto ($17.5/4=4.375\text{m}\Omega$). Tale soluzione ha anche migliorato la dissipazione della potenza elettrica in quanto ciascun transistor è attraversato da una sola parte della corrente assorbita dai motori (funzionamento analogo ad un partitore di corrente).

Tornando al circuito interruttore, il suo principio di funzionamento è molto semplice: quando la tensione V_{gs} supera i 4.5V, i MOS entrano in saturazione chiudendo il circuito e permettendo il passaggio della corrente. Quando la V_{gs} scende sotto la tensione di soglia e raggiunge gli 0V, il transistor entra in zona di Cut-Off rendendo la sua resistenza interna teoricamente infinita ed aprendo il circuito. Le due resistenze R_2 ed R_3 servono rispettivamente come protezione del circuito a monte dell'interruttore ed a permettere la scarica a massa delle capacità parassite che impedirebbero una corretta interdizione del canale del MOSFET.

• DIODO: P600

D_2 è un diodo P600 di tipo general purpose per alte potenze, utilizzato come diodo di libera circolazione. La sua funzione è quella di proteggere il carico da picchi di tensione di verso opposto, dovuti a spegnimenti rapidi di carichi induttivi.

• OPTOCOMMUTATORE: 6N136

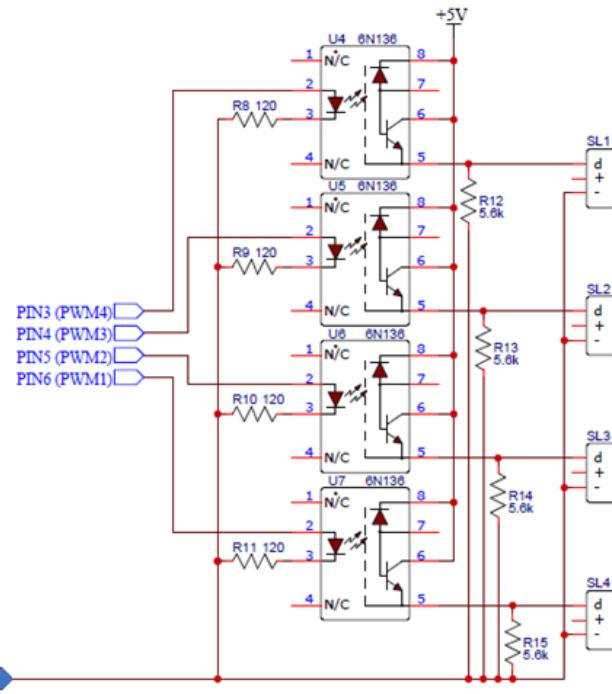
Il circuito adattatore di livello viene utilizzato per modificare i livelli logici in ingresso provenienti dal pin digitale della Renesas (0 - 3.3V), innalzandoli a livelli riconoscibili dall'interruttore MOSFET (0-5v).

Il principio di funzionamento verrà descritto nel paragrafo successivo.



Figura 15: Optocommutatore 6N136.

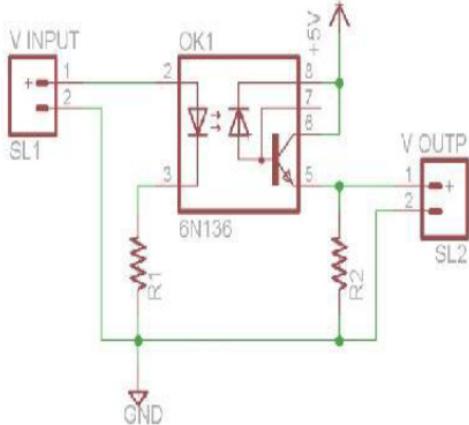
3.7.2.4 Adattatore di livello per segnali PWM



Come per il segnale di controllo dell'interruttore MOSFET, questo circuito viene utilizzato per adattare i livelli in ingresso, provenienti dalle linee digitali e PWM della Renesas (0 – 3.3V), a livelli necessari per pilotare gli ESC (0 – 5 V).

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

Dati:



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

$$R_1 > \frac{V_{input}-V_d}{I_{dMax}} = \frac{3.3 - 1.45}{0.020} > 92 \Omega$$

$$R_2 > \frac{(V_{output}-V_{ol}) * 100}{I_{fMax}*} = \frac{(5 - 0.4) * 100}{0.008 * 27} > 2129 \Omega$$

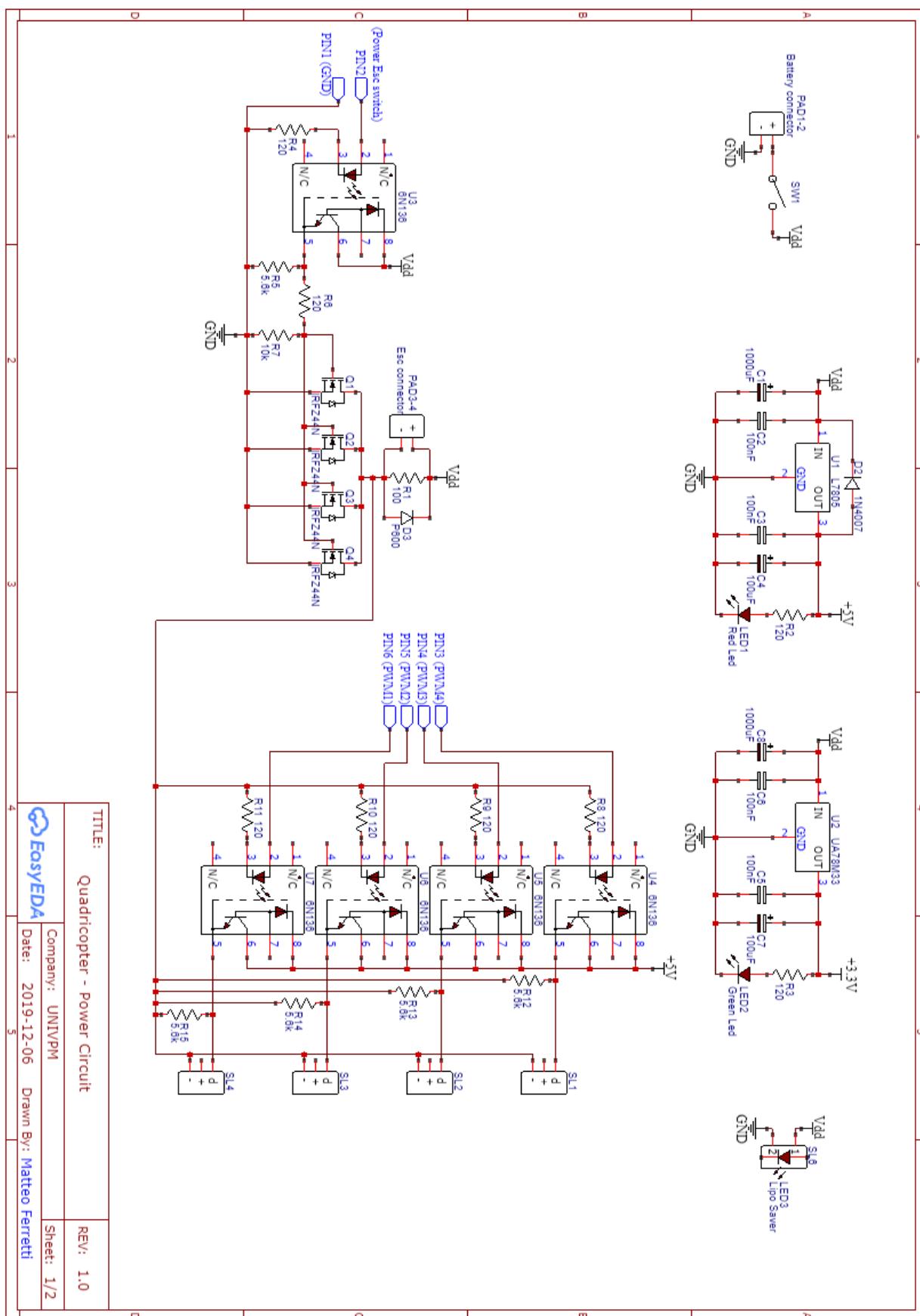


Figura 16: Schematic completo

3.8 ALTIMETRO

Il sensore di distanza VL53L1X STMicroelectronics (mostrato in Figura 18a) utilizza la tecnologia "Time of Flight" (ToF) di ultima generazione. Tale tecnologia consente una misura della distanza indipendente dalla riflettanza dell'obiettivo, a differenza di quanto avviene con la classica tecnologia a Infrarosso (IR) che si limita a misurare l'intensità del segnale riflesso. Grazie a questa tecnologia il sensore riesce ad offrire una misura di distanza accurata da 4 cm fino a 4 m ad una frequenza massima di 50 Hz. Il dispositivo offre anche la possibilità di programmare la riduzione e lo spostamento del Field of View (FoV) del sensore.

La tecnologia ToF, su cui si basa il funzionamento del sensore, consiste nel misurare la distanza sensore-oggetto utilizzando il "tempo di volo", cioè il tempo che l'impulso luminoso generato da un emettitore impiega per raggiungere l'ostacolo, essere riflesso e tornare indietro eccitando il ricevitore. Data la velocità della luce (c) e misurato il tempo trascorso tra l'impulso inviato e quello ricevuto (Δt), il valore della distanza (d) viene calcolato semplicemente con la formula:

$$d = \frac{c\Delta t}{2}$$

Il sensore mette inoltre a disposizione la possibilità di scegliere tra 3 diverse Ranging Options (o Distance Mode): Short, Medium e Long. La scelta della modalità determina una diversa configurazione del sensore che consente di ottenere i migliori risultati in base alle distanze di interesse e alle condizioni di illuminazione dell'ambiente. In particolare la modalità Long permette di raggiungere la massima distanza di 4 m, ma è fortemente influenzata dalla luce dell'ambiente; la modalità Short è la più immune ai disturbi luminosi ma copre un range massimo di 1,3 m.

Lo studio approfondito del sensore Altimetro VL53L1X è già stato eseguito negli anni precedenti quindi per maggiori informazioni su di esso e sul suo funzionamento si rimanda all'allegato "Relazione Altimetro" presente nella cartella "Datasheet Componenti Altimetro".

Abbiamo testato il programma dell'altimetro tramite oscilloscopio osservando che il PWM in uscita dalla Renesas dipende dalla distanza misurata tra il sensore e l'ostacolo: maggiore è l'altezza, minore è il duty-cycle del PWM, quindi minore sarà la velocità di rotazione dei motori; viceversa, minore è l'altezza, maggiore è il duty-cycle e quindi la velocità con cui gireranno. Per capire meglio il funzionamento leggere la sezione riguardante la parte SOFTWARE: 4.

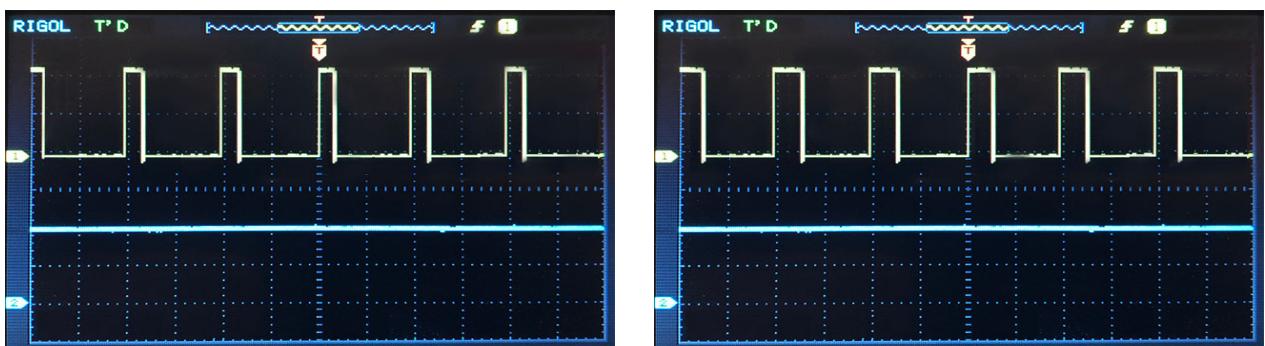
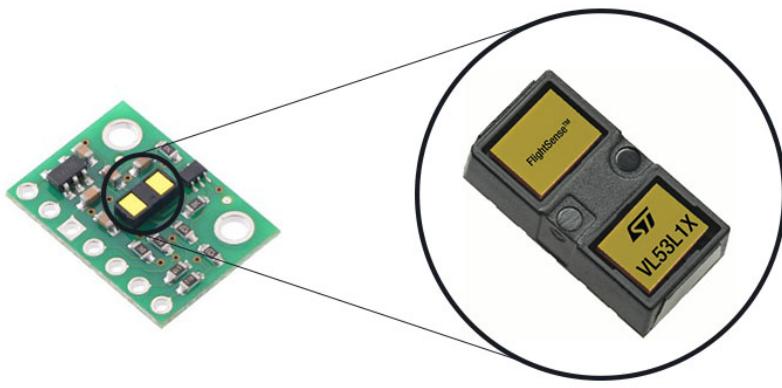


Figura 17: PWM duty-cycle range.

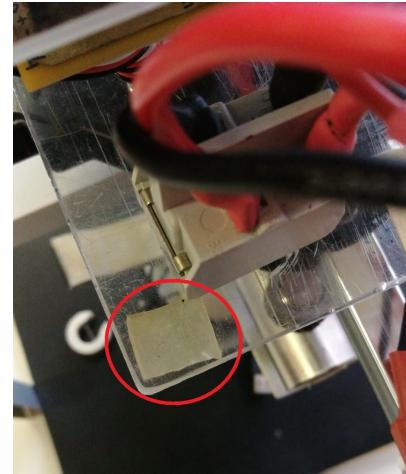
3.8.1 Posizionamento

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

Si può notare in un angolo del plexiglass un gommino adesivo dove deve essere agganciato il sensore.



(a) Altimeter VL53L1X



(b) Posizione dell'Altimetro

3.9 IMU

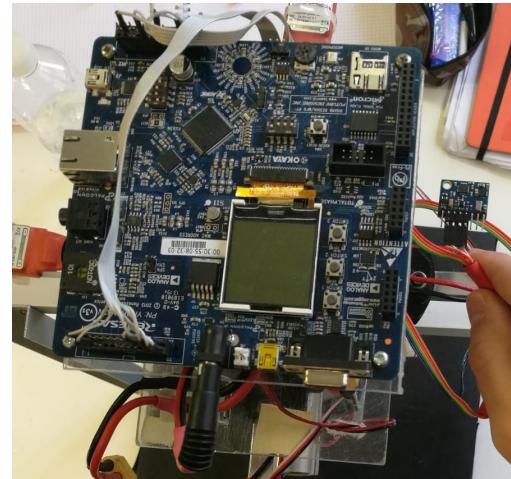
La trattazione dell'IMU non era di nostra competenza. Se si desiderano maggiori informazioni su di essa, si possono trovare nella relazione specifica per il componente.

3.9.1 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 19b dato che il sistema di riferimento dell'IMU deve coincidere con quello del Drone.



(a)



(b)

Figura 19: Posizione e verso del sensore IMU.

3.10 CONNETTORI

Per interfacciare la scheda Renesas con le altre componenti hardware del quadricottero, sono stati utilizzati diversi connettori schematizzati di seguito:

Collegamenti sui connettori della Renesas:

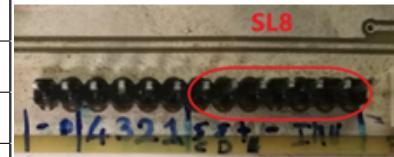
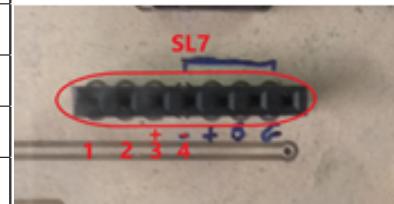
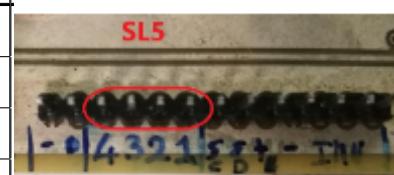
Periferica Esterna	Connettore	PIN	Descrizione
IMU	JN1	2	GND
	JN1	5	3.3V
	JN1	25	SDA
	JN1	26	SCL
Altimetro	JN1	2	GND
	JN1	5	3.3V
	JN1	25	SDA
	JN1	26	SCL
Optocommutatore	JN2	4	GND
Alimentazione ESC	JN2	14	3.3V
PWM1	JN1	23	pwm
PWM2	JN2	9	pwm
PWM3	JN2	22	pwm
PWM4	JN2	23	pwm
MODULO WIFI	JN1	2	GND
	JN1	5	3.3V
	JN1	20	Data Input
	JN1	21	Data Output



Figura 20: I componenti utilizzati per la raccolta e trasferimento dati

Collegamenti sui connettori della Scheda di Potenza:

Funzione	Connettore	PIN	Descrizione	
Collegamento PWM1-ESC	SL1	PIN 1	Dati PWM	
		PIN 2	NC	
		PIN 3	GND	
Collegamento PWM2-ESC	SL2	PIN 1	Dati PWM	
		PIN 2	NC	
		PIN 3	GND	
Collegamento PWM3-ESC	SL3	PIN 1	Dati PWM	
		PIN 2	NC	
		PIN 3	GND	
Collegamento PWM4-ESC	SL4	PIN 1	Dati PWM	
		PIN 2	NC	
		PIN 3	GND	
Collegamento Renesas-PWM	SL5	PIN 1	PIN 23 - JN1	
		PIN 2	PIN 9 - JN2	
		PIN 3	PIN 22 - JN2	
		PIN 4	PIN 23 - JN2	
LIPO Saver	SL6	PIN 1	VDD(Batteria)	
		PIN 2	GND	
Collegamento Sensori-SL8	SL7	PIN 1	SCL	
		PIN 2	SDA	
		PIN 3	+3.3V	
		PIN 4	GND	
		PIN 5	DATI WIFI	
		PIN 6	DATI WIFI	
Collegamento SL8-Renesas	SL8	PIN SC	PIN 26 - JN1	
		PIN SD	PIN 25 - JN1	
		PIN +	PIN 5 - JN1	
		PIN -	PIN 2 - JN1	
		PIN W1	PIN 21 - JN1	
		PIN W2	PIN 20 - JN1	
Collegamento Renesas-Ali.ESC	SL10	PIN 1	PIN 14 - JN2	
		PIN 2	PIN 4 - JN2	



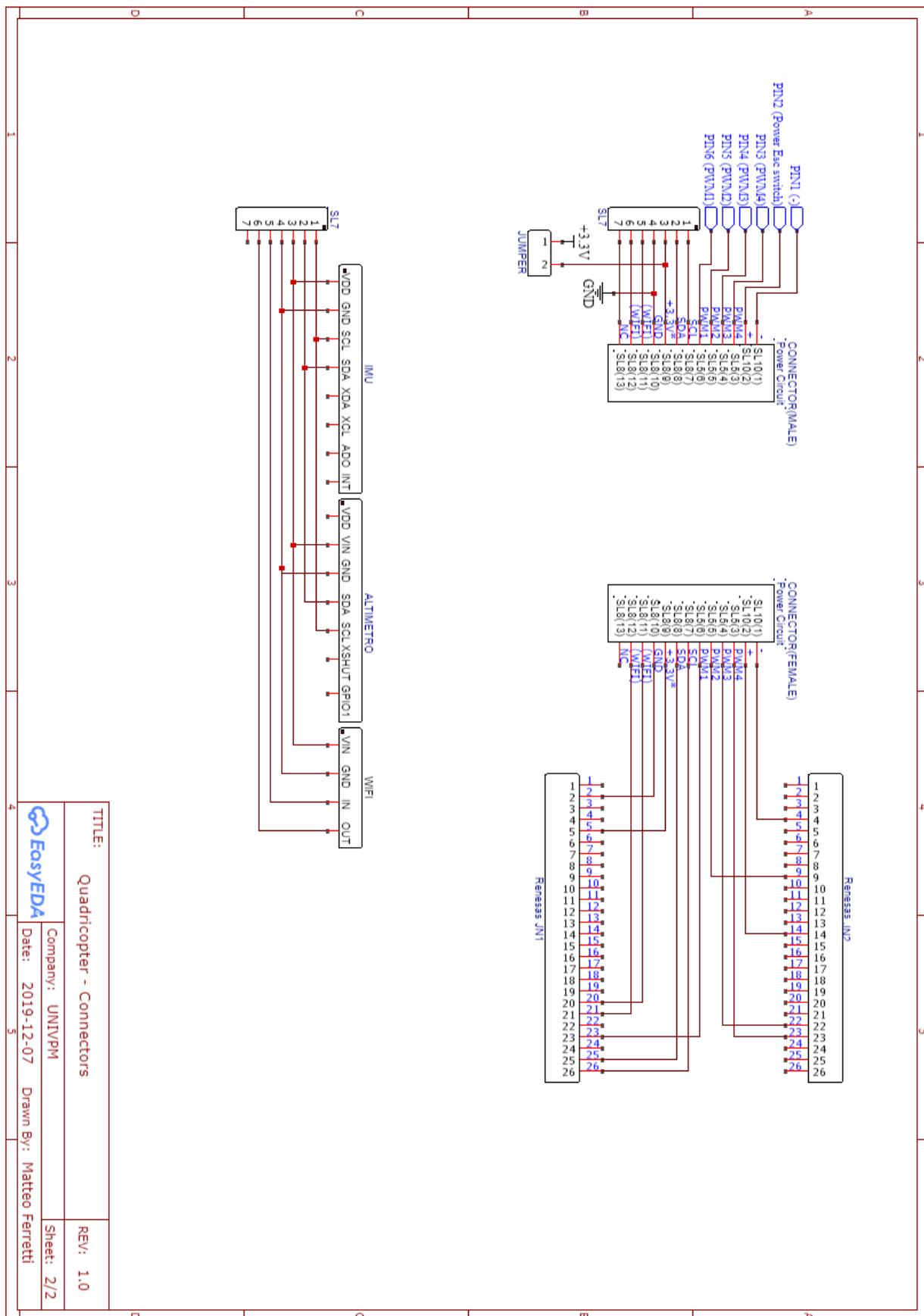


Figura 21: Schematic dei connettori

4 SOFTWARE

4.1 Struttura generale

In seguito all'unione dei codici di gestione del sensore IMU e del sensore Altimetro, la struttura è stata suddivisa in 3 componenti principali:

La prima riguarda l'Altimetro (cartella **Altimeter**), la seconda per l'IMU (cartella **IMU**), e la terza di carattere generico, priva di una propria cartella, definita direttamente all'interno di **src**. (vedere la Figura 22 riportata qui di seguito).

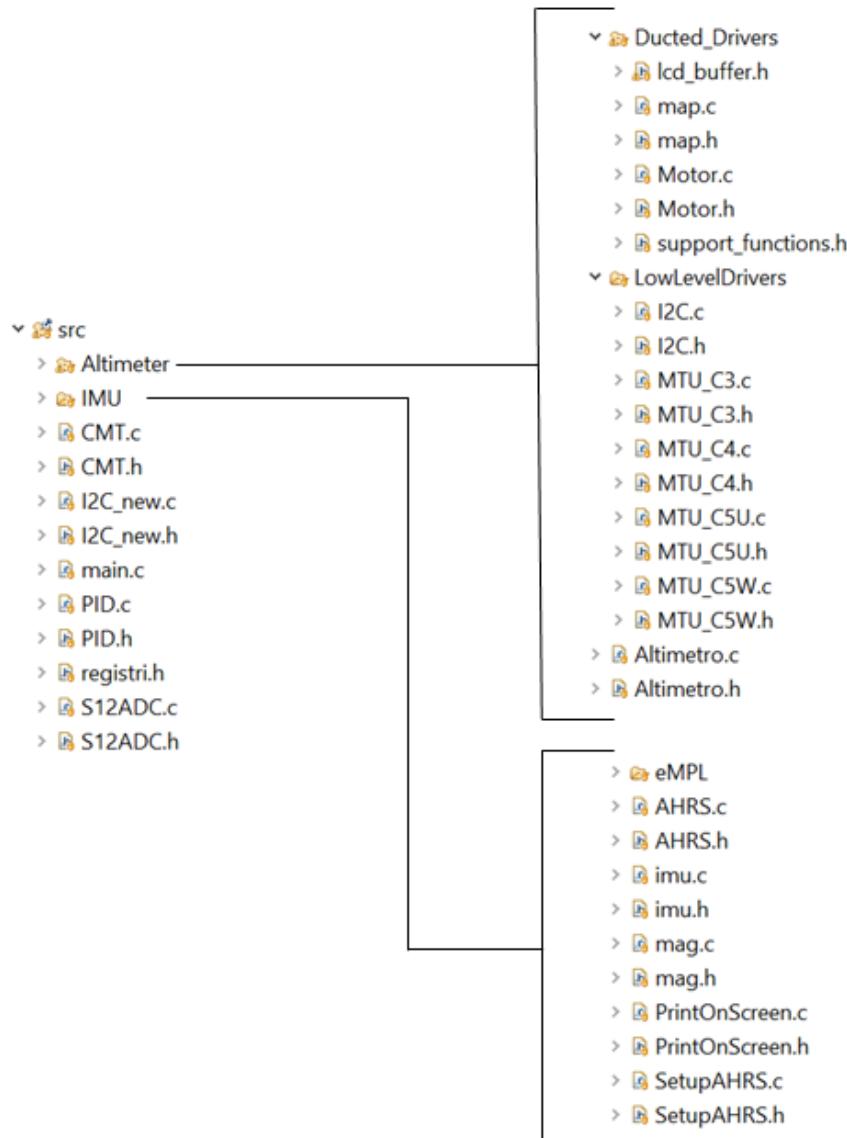


Figura 22: Struttura generale del codice

Riportiamo di seguito le parti salienti del codice di nostra competenza.

4.1.1 Generiche

- `main.c`: il main del programma si compone di una fase di inizializzazione in cui: si armano i motori e si inizializzano i PID (funzione `Setup_Motor_PID()`), si predispongono altimetro e IMU al corretto funzionamento (funzioni `Altimeter_init()`, `Setup_MARG(&ahrs)`).

IMPORTANTE!!! È necessario fare prima l'armamento dei motori e successivamente il setup dell'IMU, altrimenti, scambiando l'ordine delle funzioni i motori riceveranno il segnale di armamento troppo tardi (come descritto nel paragrafo 3.3.3).

Segue, poi, un loop infinito in cui sono state predisposte una serie di callback di cui, solo 4 operative: `Callback_5ms()`, `Callback_10ms()`, `Callback_50ms()` e `Callback_100ms()`. La prima viene utilizzata per leggere i dati peculiari dell'assetto attraverso l'IMU; la seconda, avvia/ferma i motori; la terza, stampa i risultati a schermo; la quarta, valuta l'assetto desiderato, i risultati delle letture dei due sensori e le inserisce all'interno della funzione `PID_Compute(float input, float setPoint, PID_config* conf)`. Questa funzione genera l'ingresso virtuale u_i trattato nella sezione 2.3. A questo punto, u_i è convertito in velocità di rotazione attraverso `SpeedCompute(float virtualInputs[], float b, float l, float d)`, e, la velocità appena ottenuta, viene trasferita sotto forma di segnale PWM ai motori (funzione `Motor_Write_up(int channel, float up)`).

- `I2C_new.c`: la scheda Renesas per gestire le comunicazioni con i componenti esterni utilizza il protocollo i^2C , essa funge da master mentre i componenti esterni si comportano da slave. Per mantenere la compatibilità con i codici precedenti, i due file `i2C.c` (appartenente al codice dell'IMU) e `I2C.c`, `I2C_new.c` (appartenente al codice dell'altimetro) sono stati riuniti in unico file, lasciando invariate le funzioni di read e write dei due sensori. Gli addresses identificativi dei due dispositivi sono:
 - per l'IMU `HMC5983_ADDRESS` (macro contenente il numero 0x1E)
 - per l'altimetro `VL53L1X_ADDRESS` (macro contenente il numero 0x29)

Inoltre il sensore VL53L1X possedeva un indirizzo di default (0x52) che è stato cambiato in 0x29 (come riportato sopra). La velocità massima di comunicazione permessa dal sensore è di 400 Kbps (I2C fast mode) ma non è stata sfruttata nel progetto che utilizza invece una velocità di soli 100Kbps (I2C slow mode).

- `CMT.c`: il CMT è un componente hardware e più precisamente, un timer, che si occupa di generare interrupt ogni qualvolta il timer stesso raggiunga un valore prestabilito (per es. 1,5,10,20 ecc.). Come è intuitibile, i codici precedenti disponevano, ognuno, di un file di gestione del CMT proprietario, essendo inutile possedere due files di questo tipo, essi sono stati unificati.
Alcune informazioni utili: Il prescaler è settato a 128, la priorità di interrupt è settata al livello 4.
- `PID.c`: L'implementazione dei PID è stata attuata a livello software (per approfondire l'argomento si rimanda alla sezione 4.2. `PID.c` si compone di due funzioni:
 - `PID_Init((PID_config* conf, float kp, float kd, float ki, float dt, float outMin, float outMax)` riempie una struttura del tipo `PID_config` in modo da avere sempre disponibili i coefficienti fondamentali necessari al calcolo degli ingressi virtuali. `outMin` e `outMax`, rappresentano il range di valori ammissibili in uscita dai PIDs.

- `PID_Compute(float input, float setPoint, PID_config* conf);` riferendosi alla struttura precedentemente riempita esegue il calcolo dell'ingresso virtuale (vedere la sezione 4.2).

4.1.2 Altimeter

- *DuctedDrivers* :

- `map.c`: Il calcolo delle velocità che poi sarà trattato in seguito in modo approfondito, fornisce un risultato, esso deve essere convertito in un range di valori apprezzabile dai motori. Per questo motivo è stata implementata la funzione `map` che non fa altro che prendere un valore (`val`) appartenente all'intervallo `[from_src, to_src]` e inserirlo in un altro intervallo, `[from_dst, to_dst]`.

```
inline 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;
}
```

Figura 23: funzione map

- `lcd_buffer.h`: semplice buffer per il display lcd.
- `Motor.h`: Questo header contiene non solo i prototipi di `Motor.c` ma anche qualche interessante definizione:

```
#define MOTOR_1 1
#define MOTOR_2 2
#define MOTOR_3 3
#define MOTOR_4 4
#define MOTOR_PWM_SIGNAL_PERIOD_UP 20000.00 //20ms
#define MOTOR_ARM_UP 950 //Duty cycle 4.75% to arm up the motor
#define MOTOR_MIN_UP 1200 //Min value of duty cycle - 6% Duty (min speed)
#define MOTOR_MAX_UP 2000 //Max value of duty cycle - 10% Duty (Max speed)
#define B_4 0.000001163 //thrust coefficient 4-cell battery
#define B_3 0.000001162 //thrust coefficient 3-cell battery
#define L 0.3375 //distance between motor and drone center
#define D 0.08 //drag coefficient
#define MOTOR_MAX_SPEED_3 1220.6926 //sqrt(1/(4*B_3)+1/(2*L*B_3)+1/(4*D))
#define MOTOR_MAX_SPEED_4 1220.2435 //sqrt(1/(4*B_4)+1/(2*L*B_4)+1/(4*D))
```

Figura 24: panoramica variabili definite in `Motor.h`

- * `MOTOR_1,_2,_3,_4`, sono delle semplici macro che utilizziamo per richiamare i diversi canali dei motori.
- * `MOTOR_PWM_SIGNAL_PERIOD_UP`, definisce il periodo del PWM, come si vede in questo momento è impostato a 20ms.
- * `MOTOR_ARM_UP` rappresenta il duty cycle necessario per armare i motori.
- * `MOTOR_MIN_UP` rappresenta il minimo duty cycle inviabile senza che venga a meno la rotazione.
- * `MOTOR_MAX_UP` rappresenta il massimo duty cycle inviabile ai motori.
- * `B_3, B_4` coefficienti di spinta per le batterie a 4 e a 3 celle.
- * `L` lunghezza bracci del drone, coincide con la distanza tra un motore e il centro.
- * `D` coefficiente di drag dell'elica (è necessario stimarlo in modo più approfondito).

- * MOTOR_MAX_SPEED_3,_4 rappresentano la massima velocità in uscita dalla matrice d'assetto 2.4.1) a cui è stata applicata la radice quadrata.
- Utilizzare "3" o "4" a seconda del numero di celle della batteria utilizzata.
- Il calcolo effettuato è il seguente:

$$\sqrt{\left(\frac{1}{4b} + \frac{1}{2lb} + \frac{1}{4d}\right)}$$

e pone le sue basi sulla matrice d'assetto, considerando i contributi dei vari coefficienti come positivi e gli ingressi virtuali t.c. $u_1 = u_2 = u_3 = u_4 = 1$.

Si rimanda alla sezione 2.6 per la comprensione dei valori assegnati ai coefficienti B_3 , B_4 , L , D .

- **Motor.c:** Questo codice sorgente contiene tutte le funzioni atte all'utilizzo dei motori:
 - * `void Motors_Init()` inizializza le unità MTU3, MTU4 settandone tutti i parametri richiesti
 - * `void StartCount_MTUs()` attiva il conteggio delle unità MTU3 e MTU4.
 - * `void HaltCount_MTUs()` ferma il conteggio delle unità MTU3 e MTU4.
 - * `void Motor_Arm(int channel)` è la funzione per l'armamento dei motori, channel rappresenta quale motore deve essere armato. Essa agisce mediante un richiamo alla funzione `Motor_Write_up(...)`
 - * `void Motor_Write_up(int channel, float up)` a sua volta, richiama la funzione `Motor_Write_PWM(...)` passandogli come valori il numero identificativo del motore a cui inviare il PWM e il duty cycle.
- Quest'ultimo è calcolato come:

$$\frac{up * 100}{MOTOR_PWM_SIGNAL_PERIOD_UP}$$

- * `void Motor_Write_PWM(int channel, float value)` funzione che setta il nuovo PWM da inviare al motore che viene indicato dal `channel`. Il nuovo PWM avrà un duty cycle a `value`.
- * `void Motors_Off()` serve per spegnere i motori, effettua una chiamata a `Motor_Write_PWM(...)` impostando `value` allo 0%.

- `support_functions.h`:

- *LowLevelDrivers :*

- **MTU_C3.c:** nell' MTU_C3 vengono definite le caratteristiche dei PWM che vengono inviati ai motori 1 e 2. Per entrambi si setta un valore di prescaler di CLK/16 e vengono impostati in PWM mode 1.
per quanto riguarda il motore 1 viene utilizzato il canale MTIOC3B e viene settato come output il pin 23 della JN1
per quanto riguarda il motore 2 viene utilizzato il canale MTIOC3D e viene impostato come output il pin 9 della JN2
- **MTU_C3.h:** Questo header contiene i prototipi delle funzioni utilizzate da MTU_C3, inoltre definisce anche i valori del periodo di TGR, ovvero 20 ms, e del duty cycle di default: dello 0%.

- **MTU_C4.c:** nell'MTU_C4 sono definite le caratteristiche dei PWM che vengono inviati ai motori 3 e 4. come per i motori 1 e 2 è stato scelto come valore di prescaler CLK/16 e come tipo di PWM il PWM mode 1.
per quanto riguarda il motore 3 viene utilizzato il canale MTIOC4A e viene settato come output il pin 22 della JN2
per quanto riguarda il motore 4 viene utilizzato il canale MTIOC4C e viene settato come output il pin 23 della JN2
- **MTU_C4.h:** Questo header contiene i prototipi delle funzioni utilizzate da MTU_C4, inoltre definisce anche i valori del periodo di TGR, ovvero 20 ms, e del duty cycle di default: dello 0%.
- **Altimetro.c:** l'altimetro utilizzato dal drone non è provvisto di documentazione che indichi come configurarlo ed utilizzarlo per cui è stata utilizzata una libreria API sviluppata per l'utilizzo su Arduino che è stata adattata per funzionare su e2studio per maggiori informazioni consultare la relazione allegata.
- **Altimetro.h:** contiene i prototipi delle funzioni e definisce le variabili utilizzate da Altimetro.c

4.1.3 IMU

La trattazione della parte software dell'IMU non era di nostra competenza. Se si desiderano maggiori informazioni su di essa, si possono trovare nella relazione specifica per il componente, mentre per quanto riguarda le parti essenziali, utili alla comprensione generale del codice, si rimanda alla parte superiore della relazione.

4.2 PID

Per realizzare un controllore PID è sufficiente discretizzare la formula integro-differenziale che lo definisce.

Innanzitutto è necessario calcolare l'errore attraverso la differenza tra l'uscita desiderata e quella attuale.

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

successivamente possiamo calcolare il termine integrale e derivativo:

- **derivativo** - approssimiamo la derivata dell'errore con il metodo delle differenze finite ($\frac{d(x)}{dt} \simeq \frac{x_f - x_i}{T}$, dove T è il tempo di campionamento).

```
float dInput = (error - conf->lastError) / conf->dt
```

- **integrale** - approssimiamo l'integrale dell'errore con la formula composita della regola del rettangolo ($\int_a^b f(x)dx \simeq h \sum_{i=1}^n f(x_i)$, con $h = \frac{b-a}{M}$ e M := numero di sottointervalli in cui è applicata la regola). Nel nostro caso, poniamo h pari al periodo di campionamento `dt` e, non conoscendo in partenza l'ampiezza dell'intervallo, sommeremo progressivamente il prodotto $e*T$ e per ovvi motivi moltiplichiamo già per il coefficiente K_I .

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

A questo punto, non resta che moltiplicare ogni termine per il corrispettivo coefficiente:

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

4.3 Funzionamento del codice

Dopo aver trattato in dettaglio la struttura del codice vediamo come le varie parti interagiscono fra di loro, descrivendo più approfonditamente alcune funzioni cruciali (alcune di queste sono già state introdotte in modo approssimativo).

4.3.1 Fase di Inizializzazione

Per prima cosa è necessario inizializzare i timers, CMT_init() esegue le operazioni tipiche di accensione della periferica, abilitazione alla ricezione degli interrupt, inizio del conteggio. Poi, inizializziamo la PORT4 che utilizzeremo come switch per poter spegnere e accendere i motori senza dover spegnere la scheda, inizializziamo l'LCD e lo ripuliamo da eventuali scritte per sicurezza.

```
void main(void) {

    /* One time initialization instructions */
    CMT_init();

    /* Dichiarazione switch 3*/
    PORT4.PODR.BIT.B4 =0;
    PORT4.PDR.BIT.B4 =0;
    PORT4.PMR.BIT.B4 =0;

    lcd_initialize();
    lcd_clear();

    /*arms the motors, initializes PIDs*/
    Setup_Motor_PID();

    /* Altimeter */
    Altimeter_init();

    /* IMU */
    /* LCD, CMT, accelerometer, gyroscope, magnetometer setup and calibration
    Setup_MARG(&ahrs);

    /* necessary time to arm motor 1, 2, 3 and 4 */
    while(!timers.timer_2000mS) {
        //lcd_display(LCD_LINE2, " 2s to arm ");
    }
    timers.timer_2000mS = 0;
```

Figura 25: Prima parte del main

A questo punto passiamo a `Setup_Motor_PID()` che riportiamo in figura 26.

Grazie a questa funzione, possiamo inizializzare i timer per i motori e proseguire all'armamento. Inoltre predisponiamo i PID una volta per tutte con i coefficienti di cui necessitano.

N.B: i PID che riguardano l'IMU devono essere calibrati correttamente, per fare ciò è sufficiente cambiare i coefficienti delle ultime tre `PID_init()` Per quanto riguarda il resto, non ci dilunghiamo

```
void Setup_Motor_PID() {
    /* Display message on LCD */
    lcd_display(LCD_LINE2, "      SETUP      ");

    /* Initialize motors */
    Motors_Init();
    /* Turn on motors relay */
    StartCount_MTUs();
    /* Send arm signal to motors */
    Motor_Arm(MOTOR_1);
    Motor_Arm(MOTOR_2);
    Motor_Arm(MOTOR_3);
    Motor_Arm(MOTOR_4);

    // Initialize PID structures used for PID properties
    // with their respective coefficients for proportional,
    // derivative and integrative
    PID_Init(&z_axis_PID, 0.7, 0.05, 0.3, dt, 0, 1);

    desiredState.key.motor_diff_us = 0; // variable to control the rotation
    desiredState.key.abs.pos.z = 0.20;

    /*IMUs PID: they need to be tuned properly, right now Kp and Kd are set to 1*/
    PID_Init(&Pitch_PID, 1, 1, 0, dt, 0, 1);
    PID_Init(&Roll_PID, 1, 1, 0, dt, 0, 1);
    PID_Init(&Yaw_PID, 1, 1, 0, dt, 0, 1);
}
```

Figura 26: Armamento dei motori e settaggio dei PID

ulteriormente, `Altimeter_init()` inizializza l'altimetro secondo le condizioni introdotte nella sezione 3.8 e predispone l'i2C, mentre `Setup_MARG(&ahrs)` si occupa dell'IMU.

A questo proposito, Il gruppo che si è adoperato per il funzionamento del sensore inerziale ha aggiunto la possibilità di evitarne la calibrazione a ogni accensione, ora, è sufficiente premere SW2 quando richiesto, come mostra l'LCD.

Concludiamo la fase di inizializzazione aggiungendo 2 secondi di busy waiting per dare il tempo ai motori di armarsi.

4.3.2 Fase di esecuzione Real-Time

Siamo giunti al core vero e proprio di questo software. Per implementare un sistema real-time sul nostro quadricottero ci appoggiamo all'utilizzo del compare match timer della scheda. Per questo, troviamo all'interno di un loop infinito (tipico dei sistemi embedded) una serie di operazioni eseguite ogni volta che è passato un certo intervallo di tempo.

```
while (1) {

    if (timers.timer_1ms) {
        timers.timer_1ms = 0;
        //Callback_1ms();                                //Operations to do every 1ms

    if (timers.timer_5ms) {
        timers.timer_5ms=0;
        Callback_5ms();                               // Operations to do every 5ms

    if (timers.timer_10ms) {
        timers.timer_10ms = 0;
        //Callback_10ms();                            // Operations to do every 10ms
    }
}
```

Figura 27: Parte del loop infinito

Abbiamo già introdotto ciò che segue nella sezione 4.1.1, proseguiamo, ora a trattarlo in dettaglio.

In questo momento, terminata, cioè, la fase di inizializzazione, il drone è ancora fermo, ma i motori sono armati, è, quindi, sufficiente premere SW3 per metterli in rotazione. Infatti, un interrupt generato dalla funzione `Callback_50ms()`, provvede a cambiare la variabile `motor_switch` a "true" (era settata di default a "false").

Come vedremo di qui a poco, questa variabile permette di entrare in una sezione di codice che invia il segnale PWM ai motori.

```
if((1 != PORT4.PIDR.BIT.B4)&&(motors_switch==true)) //Press SW3 to send the pwm signal to the ESC
{
    motors_switch=false;
}
else if((1 != PORT4.PIDR.BIT.B4)&&(motors_switch==false)) //Press SW3 to stop the pwm signal sent
{
    motors_switch=true;
    cont++;
}
```

Figura 28: Accensione/spegnimento motori

Passiamo, quindi, a comprendere come viene generato questo segnale PWM:

Ci spostiamo all'interno della funzione `Callback_50ms()`.

In prima istanza, viene effettuata una misura dell'altezza in cui si trova il drone (`Read(temp)`). Notare che da questa versione la misura viene convertita nella distanza vera e propria dal terreno anche se il dispositivo non si trova con i bracci perpendicolari al terreno.

Questo risultato è stato ottenuto prelevando le misure relative all'assetto attraverso l'IMU:

```
/*gets the angles measured by IMU*/
currentState.key.angle.pitch=ahrs.ahrs_data.PitchDeg;
currentState.key.angle.roll=ahrs.ahrs_data.RollDeg;
currentState.key.angle.yaw=ahrs.ahrs_data.YawDeg;
```

N.B: le strutture di ahrs vengono riempite ogni 5 ms in callback_5ms() utilizzando la funzione di "read" dell'IMU.

E applicando una semplice somma vettoriale:

$$\sqrt{h_{pitch}^2 + h_{roll}^2}$$

Dove:

- $h_{pitch} = h \cos \phi$
- $h_{roll} = h \cos \theta$
- h è la misura effettuata dall'altimetro

Tutto ciò pone il suo fondamento sul fatto che se pensiamo ad h come l'ipotenusa di un triangolo rettangolo formato da un cateto parallelo alla superficie terrestre e da un altro cateto perpendicolare ad essa, l'altezza del drone coincide con il secondo cateto. Il nostro obiettivo, dunque, è trovare questo cateto e per un discorso di angoli simili, ciò consiste nell'applicare la funzione coseno agli angoli di pitch e di roll e nel moltiplicare per h .

Per una maggiore chiarezza riguardo agli angoli di pitch e di roll si rimanda alla figura 3

I concetti matematici appena discussi sono stati tradotti in C come segue:

```
//calculates the altitude value when pitch and roll are not null
if (currentState.key.angle.pitch!=0||currentState.key.angle.roll!=0)
{
    float hpitch =distanza_metri*cos(currentState.key.angle.pitch);
    float hroll =distanza_metri*cos(currentState.key.angle.roll);

    distanza_metri=sqrt(pow(hpitch,2)+pow(hroll,2));
}
```

Ora, abbiamo una misura dello stato in cui si trova il drone, vogliamo che raggiunga lo stato desiderato (altezza desiderata (z), ϕ, θ, ψ desiderati).

Lo stato desiderato, ovviamente, deve essere comunicato al sistema, per farlo riempiamo delle strutture:

```
/* Setting desired values */
desiredState.key.abs.pos.z = altitudeValue;
desiredState.key.angle.pitch = pitchValue;
desiredState.key.angle.roll = rollValue;
desiredState.key.angle.yaw = yawValue;
```

in questo momento, sono stati scelti dei valori costanti ($\phi = \theta = \psi = 0, z = 0.5$).

Tra gli sviluppi futuri si può sicuramente pensare di controllare questo stato via wireless.

Bene, conosciamo in che stato si trova il drone, sappiamo in quale si deve posizionare: possiamo calcolare gli ingressi virtuali attraverso i PID.

```
/* computing PIDs results*/
float virtualInputs [4]; // Temporary storage for PID results

virtualInputs[0] = PID_Compute(distanza_metri, desiredState.key.abs.pos.z, &z_axis_PID);
virtualInputs[1] = PID_Compute(currentState.key.angle.pitch, desiredState.key.angle.pitch, &Pitch_PID);
virtualInputs[2] = PID_Compute(currentState.key.angle.roll, desiredState.key.angle.roll, &Roll_PID);
virtualInputs[3] = PID_Compute(currentState.key.angle.yaw, desiredState.key.angle.yaw, &Yaw_PID);
```

Come già detto più volte, questi ingressi virtuali non possono essere inviati ai motori direttamente. Abbiamo, quindi predisposto una funzione (`SpeedCompute(float virtualInputs [], float b, float l, float d)`), traduzione software della matrice d'assetto introdotta nella sezione 2.4.1.

```
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 (ciò è rappresentato dalla possibilità di ottenere velocità negative attraverso le equazioni, poiché, infatti, le uscite dei PID sono settate in modo tale che appartengano all'intervallo $[0,1]$, è sufficiente che $u_1 = 0$ e $u_3, u_4 < 0$ perché ω_1 sia minore di 0). Questo, però, 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 $[0, \text{abs_speedQuad_MAX}]$ 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 (esso coincide con la quarta equazione in cui si è considerato $u_1 = u_2 = u_4 = 1$).

Infine, le velocità ottenute vengono mappate in un range compreso tra 1200 e 2000 perché possano essere percepite dai motori con un duty consono:

```
//converts the speed in a measure that can be read by the motors
desiredState.key.avg_motor1_us = map(*(Speeds+0), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
desiredState.key.avg_motor2_us = map(*(Speeds+1), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
desiredState.key.avg_motor3_us = map(*(Speeds+2), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
desiredState.key.avg_motor4_us = map(*(Speeds+3), 0, MOTOR_MAX_SPEED_4, MOTOR_MIN_UP, MOTOR_MAX_UP);
```

E viene inviato il segnale PWM ai motori:

```

if(motors_switch==true)
{
    if(cont>=2) StartCount_MTUs();

    // writes new results to motors and servos
    Motor_Write_up(MOTOR_1, desiredState.key.avg_motor1_us);
    Motor_Write_up(MOTOR_2, desiredState.key.avg_motor2_us);
    Motor_Write_up(MOTOR_3, desiredState.key.avg_motor3_us);
    Motor_Write_up(MOTOR_4, desiredState.key.avg_motor4_us);
}
else if(motors_switch==false)
{
    if(cont>=2) HaltCount_MTUs();
    Motors_Off();
}

```

Come si nota, come introdotto qualche pagina addietro, abbiamo inserito la possibilità di spegnere e accendere i motori attraverso la pressione del pulsante SW3 della scheda: se la pressione setta `motor_switch` a "true", viene eseguita la funzione `Motor_Write_Up(...)` per ogni motore e viene inviato il segnale PWM. Altrimenti viene inviato un PWM con duty allo 0% e vengono spenti gli MTU con la funzione `HaltCount_MTUs()`.

Per comprendere il legame tra il range [1200,2000] e la percentuale di duty cycle e per avere maggiori informazioni, su come, effettivamente, il segnale PWM viene inviato ai motori, abbiamo stilato la sezione 5 che si trova alla pagina successiva.

5 Segnale PWM

Sia la fase di armamento dei motori che la fase di esecuzione del Real-Time fanno utilizzo del segnale PWM. la procedura per generare il PWM per armare i motori coinvolge tre funzioni:

1. **Motor_Arm**: La variabile `channel` rappresenta il motore che viene armato, l'unica cosa che fa questa funzione è invocare la funzione `Motor_Write_up(...)` passandogli come valori `channel` e `MOTOR_ARM_UP` (= 950) che corrisponde ad un duty cycle del 4,75% .

```
void Motor_Arm(int channel)
{
    //channel value must be 1 or 4
    if(channel < 1 || channel > 4)  return;

    Motor_Write_up(channel, MOTOR_ARM_UP);
}
```

2. **Motor_Write_up**: quello che fa questa funzione è eseguire il calcolo del valore in percentuale del duty cycle tramite la formula

$$\frac{up * 100}{MOTOR_PWM_SIGNAL_PERIOD_UP}$$

dove `MOTOR_PWM_SIGNAL_PERIOD_UP` è settato a 20000, cioè un periodo di 20 ms.

e passarlo alla funzione `Motor_Write_PWM` che si occupa di generare il PWM

```
void Motor_Write_up(int channel, float up)
{
    //channel value must be 1 or 4
    if(channel < 1 || channel > 4)  return;

    Motor_Write_PWM(channel, (up*100)/MOTOR_PWM_SIGNAL_PERIOD_UP);
}
```

infatti per quanto riguarda l'armamento:

$$\frac{950 * 100}{20000} = 4.75$$

3. **Motor_Write_PWM**: è la funzione che si occupa della generazione del PWM mediante settaggi dei registri delle MTU.

dopo aver effettuato un controllo per verificare che il duty sia espresso in percentuale si prendono i valori dei registri TGRA dell'MTU3 e MTU4; successivamente si entra in uno switch case che in base al numero del motore su cui generare il PWM setta registri diversi, rispettivamente:

- TGRC dell' MTU4 per il motore 1
- TGRD dell' MTU4 per il motore 2
- TGRC dell' MTU3 per il motore 3
- TGRD dell' MTU3 per il motore 4

```
void Motor_Write_PWM(int channel, float value)
{
    //channel value must be 1 or 4
    if(channel < 1 || channel > 4)  return;

    //check if value is a percentage, if not in range 0-100 return
    if(value<=0 || value>=100)  return;

    uint16_t tgr_a_val3,tgr_a_val4, tgr_b_val3, tgr_d_val3,tgr_b_val4, tgr_d_val4;
    tgr_a_val4 = GetTGR_A_MTU_U0_C4();
    tgr_a_val3 = GetTGR_A_MTU_U0_C3();

    // calculate tgrb and tgrd for both MTUs value from duty cycle
    switch(channel){
        case 1:
            tgr_b_val4 = (tgr_a_val4*value)/100;
            SetTGR_B_MTU_U0_C4(tgr_b_val4);
            break;
        case 2:
            tgr_d_val4 = (tgr_a_val4*value)/100;
            SetTGR_D_MTU_U0_C4(tgr_d_val4);
            break;
        case 3:
            tgr_b_val3 = (tgr_a_val3*value)/100;
            SetTGR_B_MTU_U0_C3(tgr_b_val3);
            break;
        case 4:
            tgr_d_val3 = (tgr_a_val3*value)/100;
            SetTGR_D_MTU_U0_C3(tgr_d_val3);
            break;
    }
}
```

per quanto riguarda la fase di esecuzione del Real-Time che il valore di duty cicle viene calcolato mediante i PID con le procedure spiegate precedentemente ed i valori vengono inviati alla funzione Motor_Write_PWM. Bisogna effettuare la mappatura delle velocità poichè i motori utilizzati dal drone funzionano solamente con un duty cicle compreso tra il 6% ed il 10% che corrispondono rispettivamente ad un valore di 1200 e 2000:

6 Procedura per l'attivazione del Drone

Per attivare il Drone è necessario seguire in ordine i passaggi riportati di seguito:

- 1 - Prima di tutto si deve caricare il programma "Progetto_Drone_2019-COMPLETO" tramite PC nella Renesas;
- 2 - Ora bisogna montare la Renesas sul drone e collegarla alla scheda di potenza con i connettori JN1, JN2 (figura 21) e alimentarla ai 5V della scheda di potenza;



Figura 29: JN1, JN2 e Alimentazione

- 3 - Adesso bisogna collegare i sensori (IMU e Altimetro) agli appositi connettori; **(FARE ATTENZIONE A NON CONFONDERLI!!!)**

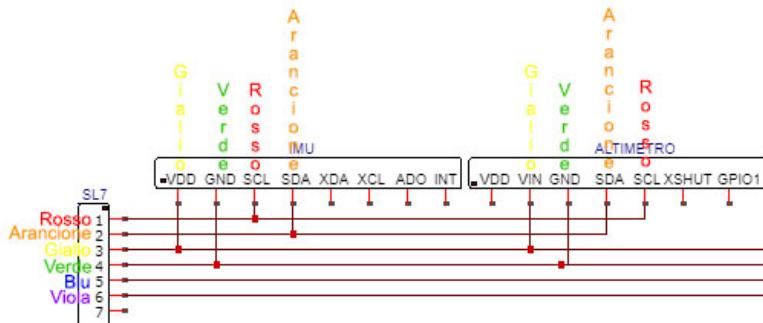


Figura 30: Connettori IMU e Altimetro

- 4 - Collegare la batteria al drone;
- 5 - Ora che tutto è stato collegato, si procede con l'accensione, quindi, attivare il drone tramite l'interruttore generale;

6 PROCEDURA PER L'ATTIVAZIONE DEL DRONE

- 6 - Come si può notare, sul display, viene richiesto di selezionare una tra le due opzioni riguardanti la calibrazione dell'IMU: se si desidera ricalibrare l'IMU (calibrazione manuale), premere switch1, oppure, se si vuole selezionare i valori già predisposti all'interno del codice (calibrazione automatica) premere switch2.

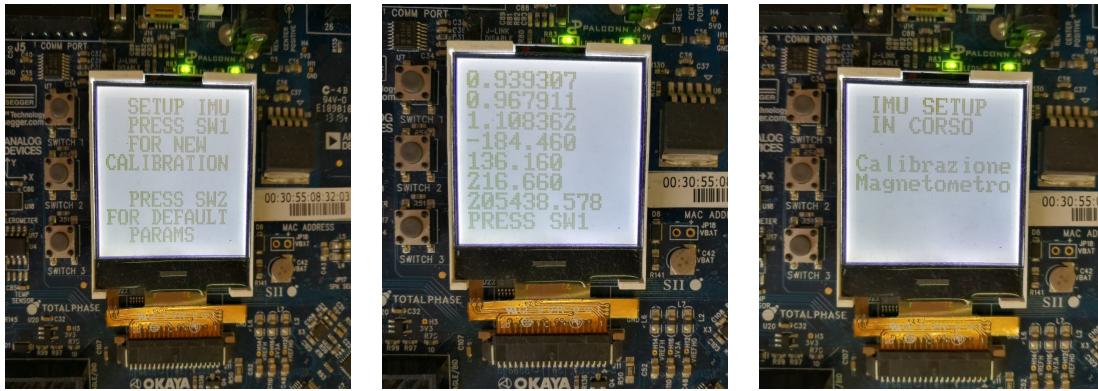


Figura 31: Da sinistra: abbiamo la scelta della calibrazione, al centro i valori predisposti all'interno del codice e a destra la calibrazione manuale

La calibrazione va rieseguita solo nel caso in cui il drone venga attivato in un luogo distante rispetto a quello in cui è stata fatta la calibrazione che si trova all'interno del codice (ovvero il laboratorio). Quindi, nel caso in cui il drone venga spostato, è necessario ricalibrare l'IMU, muovendo il sensore in tutte le direzioni, fino a quando non compaiono tutti i settaggi sul display.

- 7 - Dopo aver eseguito la calibrazione (automatica o manuale che sia) premere lo switch1 per confermare le impostazioni e armare i motori;
- 8 - A questo punto, basta premere lo switch3 per avviare o stoppare i motori in qualsiasi momento. Se, invece, si vuole variare la velocità dei motori, è sufficiente inclinare o alzare il drone, poiché essa varia in base alla distanza da terra e agli angoli di inclinazione che vengono calcolati tramite ai sensori;

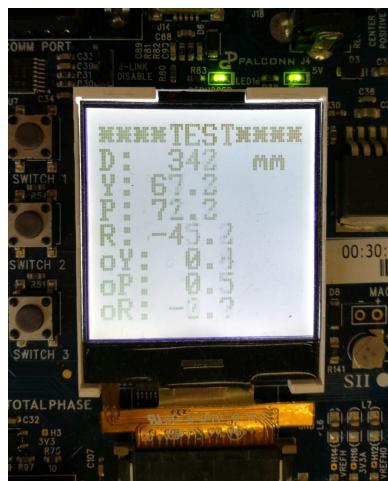


Figura 32: Dall'alto: la distanza, gli angoli di Yaw, Pitch e Roll e le velocità angolari

- 9 - Una volta conclusa la prova, fermare i motori e spegnere il Drone dall'interruttore generale.

7 SVILUPPI FUTURI

Il drone per come si presenta allo stato attuale è pronto per i test preliminari che lo porteranno al volo.

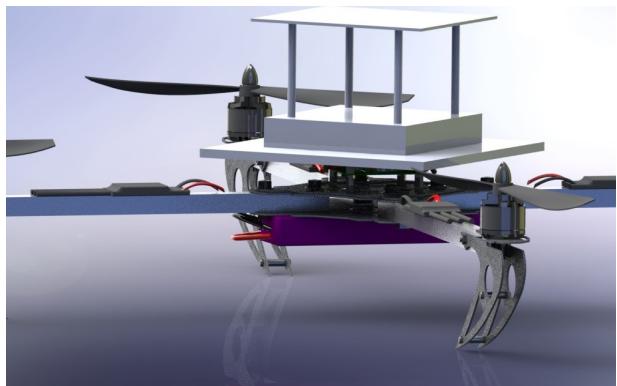
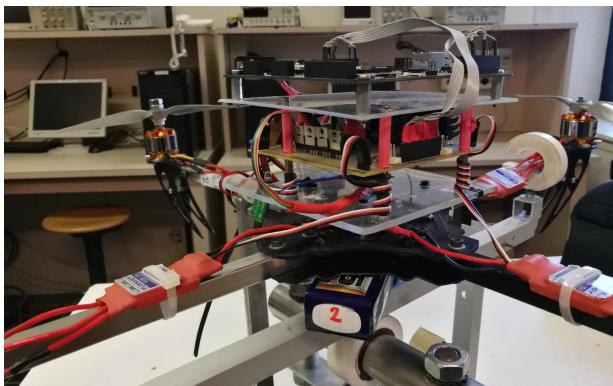
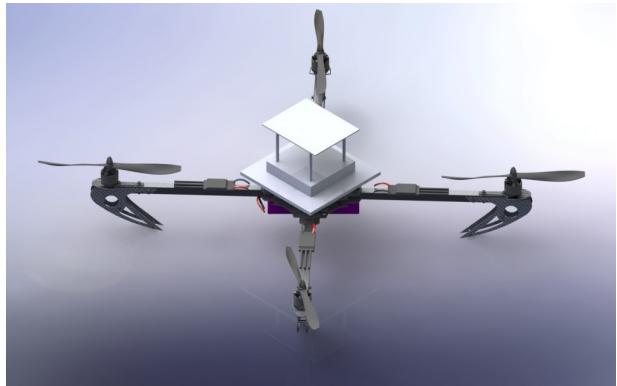


Figura 33: Comparazione del Drone Reale del 2019 con il Render Ideale del 2012.

Per questo motivo è necessario sostituire la struttura su cui è posto con una adatta al testing. Essenziale è tarare i PID che si occupano delle posizioni angolari del drone e controllare che la taratura del PID per l'altitudine sia corretta.

Successivamente, occorre unire il codice che disporremo in allegato con quello elaborato dal gruppo che si è occupato del modulo wi-fi.

Fatto ciò, si può, senza meno, implementare dei comandi a distanza che permettano di controllare il moto del quadricottero.