UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Studio e sviluppo di una piattaforma per il test di droni

Development of a testing platform for drones

Relatore:
Prof. Andrea Bonci

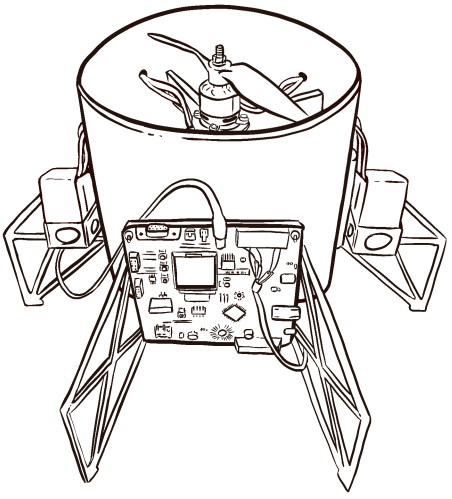
Tesi di Laurea di:
Abrami Fabio

Anno Accademico 2020/2021

Abstract

di Fabio Abrami

L’elaborato tratta, continuando uno studio precedente, i vari passaggi per riuscire ad eseguire test in sicurezza con un ducted fan. Partendo da una descrizione del drone e del codice, si giunge prima alle modalità di costruzione della gabbia, ed infine alle prove effettuate e all’ analisi dei dati raccolti .



Indice

Capitolo 1

Introduzione al drone

Lo scopo della tesi è stato quello di costruire una gabbia per il fissagio del drone e per la sicurezza degli operatori, per poi effettuare delle prove di test per studiare il volo del dispositivo e analizzarne i dati raccolti per la taratura dei PID.
Con l'utilizzo di droni che negli ultimi anni sta crescendo sempre di più è utile studiare un design che permetta di rendere sempre più sicuro il volo di questi velivoli senza avere la necessità di preoccuparsi dell'ambiente circostante. I droni a flusso convogliato riescono a soddisfare questa necessità in quanto le eliche sono poste all'interno della struttura che, oltre a convogliare correttamente l'aria per generare un flusso sufficiente a far si che il drone si sollevi dal suolo, funge essa stessa da protezione per le eliche verso l'ambiente circostante.
Lo sviluppo del software è stato realizzato andando a studiare ed incorporare nella maniera corretta parti di codice dei singoli componenti e sensori presenti nel drone.
La tesi è strutturata come segue:

- nel secondo capitolo, *Codice* e^2 studio, vengono descritte le parti principali del codice utilizzato;
- nel terzo capitolo, *Costruzione gabbia*, vengono elencati i materiali e i passaggi utilizzati per la costruzione della gabbia;
- nel quarto capitolo, *Salvataggio e rielaborazione dati*, viene spiegato come salvare i dati da e2studio e l'utilizzo di uno script di matlab per la pulizia e la rielaborazione dei dati raccolti;

- nel quinto capitolo, *Prove di volo e analisi dei dati*, vengono riportate le prove di test effettuate, i problemi riscontrati e le soluzioni adattate grazie all' analisi dei dati;
- nel sesto capitolo, *Conclusioni*, vengono riportati i risultati ottenuti e i possibili sviluppi futuri;

1.1 Struttura del drone

Il ducted fan è caratterizzato da una struttura cilindrica all'interno della quale sono presenti tutte le componenti necessarie per il moto del velivolo. Questa configurazione permette una maggiore sicurezza nel volo, poiché tutte le componenti in movimento del drone sono protette dall'ambiente circostante dalla struttura stessa. Nella configurazione che è stata presa in esame per questa tesi si va a studiare un ducted fan a doppia elica con un singolo livello di flap situato nella parte bassa della struttura. Quest'ultima è stata progettata e successivamente stampata in PVC mediante l'utilizzo di una stampante 3D. Una volta ottenute le singole componenti è stato possibile assemblarle tra di loro per ottenere la struttura finale.
Si prosegue con lo studio della struttura del drone, trovando nella parte interna superiore del ducted fan i due motori. Questi sono ancorati ad una struttura a croce utilizzata sia come base per i motori stessi sia come rinforzo alle sollecitazioni che può subire il drone. Realizzata e stampata anch'essa in PVC, fornisce una maggior rigidità alla parte superiore della struttura che altrimenti potrebbe essere alterata dai flussi di aria generati dalle eliche che potrebbero portare la struttura a collassare su se stessa.
Nella parte inferiore del ducted fan troviamo i due flap, uno per la gestione dell'angolo di *pitch* e un altro per l'angolo di *roll*. Entrambi i flap sono collegati ad un'estremità al servomotore che ne gestisce il movimento, mentre nell'altra sono inseriti in un apposito foro presente nella struttura che lo mantiene parallelo al terreno. Per ottenere un posizionamento corretto dei due flap, a 90° tra di loro, è stata utilizzata un'apposita placca a scorrimento, pensata e realizzata su misura per il ducted fan, alla quale sono agganciati i servomotori. La placca permette al blocco servomotore-flap di essere spostato di qualche millimetro per ottenere una posizione ottimale che, una volta trovata, si può fissare con i bulloni presenti nella placca che garantiranno il mantenimento della corretta posizione.
All'esterno della struttura si trovano le varie componenti hardware e sei piedi che forniscono un appoggio stabile al terreno per l'intero drone. Sono distanziati in egual misura l'uno dall'altro per una corretta distribuzione dei pesi e sono agganciati alla struttura mediante due bulloni

opportunamente posizionati con il dado all'esterno e la testa all'interno per evitare che il flusso d'aria generato nel cilindro potesse in qualche modo allentare i dadi e causare problemi di rottura o indebolimento nel tempo. I piedi sono stati anch'essi realizzati su misura e successivamente stampati in PVC per poi essere assemblati come sopra descritto.

1.2 Componenti Hardware

Di seguito verranno descritte tutte le componenti hardware utilizzate per la realizzazione del progetto. Per la gestione del software e per l'acquisizione dei dati dai sensori necessari per il corretto funzionamento dei motori e dei flap del drone è stata utilizzata la scheda Renesas Demonstration Kit YRDKRX63N. Prima di essere montato a bordo della struttura, ogni componente è stato testato singolarmente.

1.2.1 Renesas Demonstration Kit YRDKRX63N

La Renesas Demonstration Kit YRDKRX63N è una scheda che si occupa della gestione del software implementato sul microcontrollore RX63N. La sua programmazione avviene mediante l'ambiente di sviluppo e^2 *studio*, tramite il quale si eseguono anche codifica e debugging del codice da implementare.

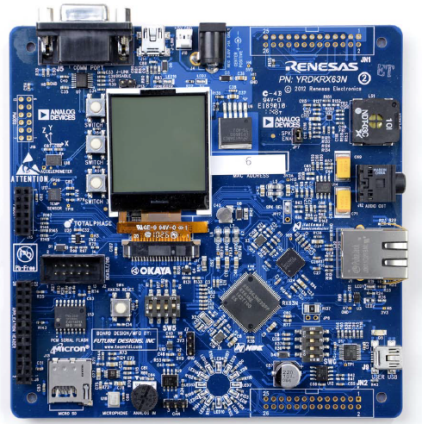


FIGURA 1.1: Renesas Demonstration Kit YRDKRX63N

1.2.2 Motori ed ESC

Per questo progetto sono stati utilizzati dei motori brushless Turnigy Aerodrive SK3-3536 da 1400kv.[1] Di seguito sono riportate le principali caratteristiche tecniche:

- Tensione: 11.1V~16.8V;
- Velocità angolare massima $\approx 2345.72 \frac{rad}{s}$;
- Potenza massima: 590W;
- Corrente massima: 40A;
- Peso: circa 105gr.

FIGURA 1.2: Turnigy Aerodrive SK3-3536

Per il controllo dei motori sono stati utilizzati degli ESC[2], dispositivi che permettono di gestire e regolare le velocità di rotazione andando a variare la percentuale di segnale PWM inviata ai motori.

FIGURA 1.3: Turnigy Plush-40A

[1] Con kv si indica il rapporto RPM/V
[2] Electronic Speed Control

Per il corretto funzionamento dei motori è necessaria una procedura di armamento che consiste nell'invio di un segnale PWM con un duty cycle di ampiezza del 4.75%. L'esito di queste operazioni può essere stabilito grazie a dei segnali acustici (*beep*) emessi dall'ESC. Di seguito sono riportati quelli più comuni:

- Un *beep* ogni due secondi indica l'attesa di un segnale PWM idoneo per l'armamento iniziale dei motori;
- Tre *beep* consecutivi seguiti da un *beep* prolungato indicano l'esito positivo delle operazioni di armamento;
- Una ripetizione rapida di *beep* sta ad indicare il mancato armamento dei motori dovuto ad un segnale in ingresso errato.

Una volta eseguita la procedura di armamento dei motori è necessario attendere qualche secondo per evitare che questi si disarmino.
Dopo varie prove di test si è stabilito come range di lavoro per il motore un segnale PWM che va da un duty cycle minimo del 6% ad un massimo del 10% quindi, per variare la velocità, si deve trovare il valore di duty cycle necessario per fornire la spinta adatta. Valori superiori al 10% di duty cycle causano il disarmamento dei motori.

1.2.3 Eliche

Una delle scelte più importanti da fare per la corretta realizzazione del progetto è sicuramente quella delle eliche da utilizzare. Sono, abbinate al motore, la fonte primaria di erogazione della spinta necessaria per far si che il drone riesca a sollevarsi da terra. É stato quindi necessario uno studio per la corretta selezione in base a due parametri fondamentali che caratterizzano tutte le tipologie di eliche, il diametro e il passo. Con il primo si indica il diametro della circonferenza che l'elica produce quando è in rotazione, con il secondo si indica la distanza che viene percorsa dall'elica dopo un giro completo. É molto importante dimensionare nella maniera corretta questi due parametri in quanto valori troppo bassi potrebbero portare a difficoltà nel far sollevare il drone da terra, al contrario, valori troppo alti, potrebbero non sfruttare a pieno le capacità dell'elica stessa. Solitamente queste due grandezze vengono espresse in pollici[3] e servono per indicare il modello di elica utilizzata. Per questo progetto sono state scelte le GEMFAN GF 9060 9"x6".

Figura 1.4: Eliche GEMFAN GF 9060

1.2.4 Batteria e Power Safe module

Per ottimizzare la posizione e la corretta distribuzione dei pesi sono state utilizzate due batterie poste in parallelo tra di loro. Di seguito vengono elencate le principali caratteristiche di ciascuna batteria:

- Tensione: 14.8V;
- Capacità: 1300mAh;

[3] *inch* in inglese, vengono indicati mediate il simbolo "

- Capacità di scarica[4]: 75C/150C;
- Numero di celle: 4.

FIGURA 1.5: Batteria Tattu

Essendo le due batterie poste in parallelo è necessario utilizzare un power safe module che permetta il collegamento nella maniera più sicura. Questo, inoltre, fa si che le batterie si scarichino in egual misura evitando problemi di erogazione della corrente dovuti alla diversa quantità di carica delle batterie stesse. Per evitare il danneggiamento della batteria è consigliabile non far scendere mai la tensione della batteria al di sotto dei 12V, quindi, considerando le 4 celle a disposizione, è consigliabile mantenere il valore di ogni cella sempre al di sopra sopra dei 3V.

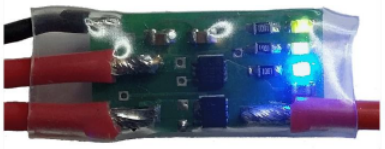


FIGURA 1.6: MRN-electronic Power safe module

1.2.5 Convertitore di tensione

Per questo progetto sono stati utilizzati diversi dispositivi ognuno dei quali necessita di una tensione di alimentazione differente. É stato quindi necessario ricorrere ad un convertitore di tensione per distribuire nella maniera corretta le tensioni di alimentazione ai vari dispositivi. Il convertitore di tensione riceve in ingresso la tensione dalle batterie ed eroga in uscita quattro diversi valori di tensione a seconda dei pin che si sceglie di utilizzare. La prima soluzione che si può adottare è quella di fornire in uscita la stessa tensione che si ha in ingresso, in questo caso il dispositivo funge solamente da ponte per il passaggio della tensione. Un'altra soluzione è

[4]Il primo valore indica la capacità di scarica costante, mentre il secondo è quella di picco

quella di utilizzare degli appositi pin sul dispositivo dai quali può essere erogata una tensione di 3.3V o di 5V, selezionabile mediante un apposito switch; queste due tensioni in uscita non possono essere erogate contemporaneamente in quanto la selezione è vincolata alla posizione dello switch. Infine ci sono dei pin che erogano in uscita una tensione variabile mediante un dimmer che permette di selezionare valori intermedi di tensione fino ad un massimo pari alla tensione di ingresso. Questa soluzione è stata utilizzata per alimentare i servomotori con una tensione di 6.5V.

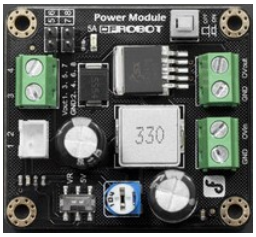


Figura 1.7: DFRobot Power module

1.2.6 Servomotori

Sono stati utilizzati dei servomotori DS3115 Digital, i quali permettono il movimento dei flap per la gestione degli angoli di *pitch* e di *roll* cosicché il drone rimanga stabile in aria a seguito di sollecitazioni che potrebbero portarlo in uno stato di movimento non controllato. Le specifiche di questi servomotori indicano come range di funzionamento tensioni che vanno da 5V a 6.8V, per evitare complicazioni, è stata scelta una tensione di utilizzo di 6.5V, la quale permette comunque una risposta in tempi brevi da parte del servomotore. Tensioni al di sotto dei 6V possono portare ad un aumento dei tempi di risposta che potrebbero, quindi, procurare problemi nella gestione del drone.
La frequenza di lavoro dei servomotori è 330Hz. Il valore del duty cycle varia tra un minimo del 17% e un massimo dell'82%.
I servomotori sono alimentati da un segnale PWM che fa variare la posizione dei flap mediante opportune conversioni effettuate nel codice. Il range completo di movimento, che va da 0° a 180°, è stato limitato per problemi di sicurezza che potevano compromettere la struttura, in quanto, essendo i due flap sovrapposti, un movimento troppo ampio da parte di uno dei due avrebbe potuto provocare una possibile rottura della struttura. Il range di movimento è stato quindi fissato, tramite codice, tra 60° e 120° con la posizione di partenza impostata a 90° rispetto

al terreno, risultando così perpendicolare ad esso. Per una gestione ottimale degli angoli di *pitch* e di *roll* i servomotori devono essere disposti a 90° tra di loro e fissati alla struttura del drone mediante apposite placche a scorrimento per ottenere un posizionamento ottimale.

FIGURA 1.8: Servomotore DS3115 Digital

1.2.7 Sensore di altezza

Per il rilevamento della quota del drone è stato utilizzato il sensore di distanza VL53L1X STMicroelectronics che utilizza la tecnologia *Time of Flight* (ToF) per l'acquisizione dei dati. In questa maniera è possibile effettuare misure indipendentemente dalla riflettenza della superficie interessata, a differenza di tecnologie precedenti, come ad esempio le misurazioni ad infrarossi, le quali si limitavano a misurare l'intensità del segnale riflesso e quindi potevano dare misure alterate dalla superficie stessa. La tecnologia ToF misura la distanza sensore-oggetto calcolando il tempo che l'impulso luminoso, generato da un emettitore presente sul sensore, impiega per raggiungere l'oggetto e tornare indietro. In questo modo è possibile ricavare la distanza utilizzando la formula $d = \frac{c\Delta t}{2}$, dove c è la velocità della luce e Δt è il tempo trascorso da quando l'impulso viene inviato a quando viene ricevuto.
É possibile selezionare quattro modalità di lettura della distanza che andranno ad influire sulla precisione delle misure. Ogni modalità è più adatta ad un certo range di misure ed è caratterizzata da due parametri, il *timing budget* e l'*inter measurement*. Il primo sta ad indicare il tempo che il sensore impiega per effettuare la misura, mentre il secondo il tempo che intercorre tra una misura e la successiva. Di seguito sono riportati i range di misura più adatti alle varie distanze con le relative tempistiche:

Modalità	Range minimo [mm]	Range massimo [mm]	Tempo di misura [ms]	Timing budget [ms]
Short	0	1300	50	20
Medium	1301	1800	110	50
Long	1801	3400	110	50
LongLong	3401	4500	290	140

Tabella 1.1: Range di valori sensore di altezza

Il sensore per il rilevamento dell'altezza è posizionato sulla base del drone in parallelo rispetto al terreno e misura solamente la distanza perpendicolare tra esso e il primo oggetto che incontra. Principalmente i risultati delle misure servono per incrementare o meno la potenza da fornire ai motori per raggiungere una determinata quota di altezza, però, se i valori degli angoli di *pitch* e di *roll* sono diversi da zero, il drone sarà inclinato rispetto al terreno, di conseguenza la misura che fornirà il sensore, ovvero quella perpendicolare ad esso, non sarà quella dell'effettiva distanza tra drone e terreno. Dovrà, quindi, essere manipolata mediante il codice per ottenere il risultato corretto, come spiegato nel paragrafo "modifiche al codice" del prossimo capitolo.

Figura 1.9: sensore VL53L1X

1.2.8 IMU

L'IMU (Inertial Measurements Unit) è una scheda elettronica capace di misurare specifiche caratteristiche ambientali o di movimento. É un componente fondamentale per i sistemi di guida inerziali e viene spesso utilizzata in velivoli come droni per il controllo degli angoli di *roll*, *pitch*, e *yaw*.
In questo progetto è stata utilizzata una IMU Drotek 10DoF costituita dalle seguenti componenti:

- accelerometro: MPU-6050 di TDK InvenSense;
- giroscopio: MPU-6050 di TDK InvenSense;

- magnetometro: HMC5983 di Honeywell;

FIGURA 1.10: IMU Drotek 10DoF

1.2.8.1 Accelerometro e giroscopio

Nel circuito integrato presente all'interno del sensore MPU-6050 troviamo un accelerometro e un giroscopio entrambi a tre assi, quindi fornirà un'uscita a sei valori; l'accelerometro misura l'accelerazione del corpo lungo le tre direzioni, il giroscopio misura le accelerazioni angolari del corpo introno ai suoi tre assi.

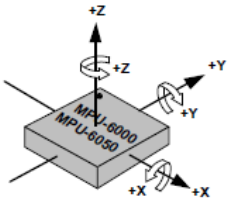


FIGURA 1.11: Orientamento degli assi di rotazione e direzione

Come già detto, l'accelerometro è un dispositivo in grado di misurare l'accelerazione andando a ricavare la forza rilevata rispetto ad una massa. Esistono diverse tipologie di accelerometri ma in questo progetto è stato utilizzato uno sviluppato con la tecnologia MEMS[5], una tecnologia capacitiva che sfrutta la variazione di capacità elettrica come misura dell'accelerazione. Nello specifico si avrà una massa formata da un'armatura mobile mentre un'altra è realizzata sul dispositivo ed è fissata ad esso. La prima massa viene sospesa su una membrana elastica che tramite un apposito circuito rileva la capacità del condensatore e genera un segnale elettrico proporzionale alla posizione della massa. Quando si verifica un'accelerazione lungo uno dei tre assi la membrana elastica subirà uno spostamento che determina una variazione di capacità nel

[5]Micro Electro-Mechanical Systems

condensatore alla quale corrisponde un preciso valore di accelerazione. Il sensore MPU-6050 utilizza masse separate per ognuno dei tre assi ottenendo così valori più precisi delle diverse accelerazioni.
Il giroscopio invece va a misurare l'accelerazione angolare lungo i tre assi di rotazione. Anche questo, come l'accelerometro, è stato realizzato utilizzando la tecnologia MEMS, in particolare utilizza l'effetto Coriolis per effettuare le proprio misure. Anche qui è presente una massa che si muove costantemente in una direzione con una determinata velocità; quando è applicata una forza esterna ci sarà uno spostamento perpendicolare della massa che causerà una variazione di capacità elettrica che verrà misurata e tramutata in velocità angolare rispetto ad uno dei tre assi.

1.2.8.2 Magnetometro

Il magnetometro è uno strumento utilizzato per la misurazione dell'intensità di un campo magnetico, in particolar modo quello terrestre. É formato da sensori magnetoresistivi che rilevano il cambiamento del flusso di materiali ferromagnetici nei quali avviene un cambiamento di resistenza quando viene applicato un campo magnetico perpendicolarmente al flusso di corrente in una striscia di materiale ferroso.
In questo progetto è stato utilizzato un magnetometro HMC5983 della Honeywell, un magnetometro con circuito integrato a tre assi con compensazione della temperatura.
Per il corretto funzionamento del magnetometro questo deve essere calibrato prima di ogni utilizzo andando ad eliminare tutti i contributi di campo magnetico che vadano ad influire su quello terrestre che si vuole misurare. Poiché questi contributi sono costanti e indipendenti dall'orientamento in ogni punto dello spazio, possono essere individuati e poi sottratti. Per individuare il contributo su ogni asse è necessario allineare il dispositivo in modo tale che gli assi del sensore corrispondano a quelli del campo magnetico terrestre, tuttavia in fase di calibrazione non sempre sarà nota la direzione del campo magnetico terrestre, di conseguenza è necessario registrare dati per un certo periodo di tempo. I valori che si avrebbero in corrispondenza dei tre assi possono essere individuati andando ad identificare i valori di picco rilevati. Per avere un'acquisizione ottimale dei dati di consiglia di effettuare questa procedura per una decina di secondi ruotando il sensore in modo tale che assuma tutte le posizioni rispetto ai tre assi di rotazione. Essendo la calibrazione un'operazione necessaria prima di ogni utilizzo che però può essere scomodo effettuare una volta che il sensore è montato a bordo del drone, si è optato

per il salvataggio dei valori necessari alla calibrazione in apposite costanti che vengono settate tramite codice durante l'inizializzazione dei vari sensori.

Capitolo 2

Codice e^{2}studio

Per lo sviluppo completo del software per il controllo del ducted fan è stato necessario implementare nella maniera corretta e più funzionale i codici per la gestione dei vari componenti hardware. Come prima cosa è stato quindi necessario uno studio delle singole parti di codice per poter effettuare in un secondo momento il *merge* del codice completo.

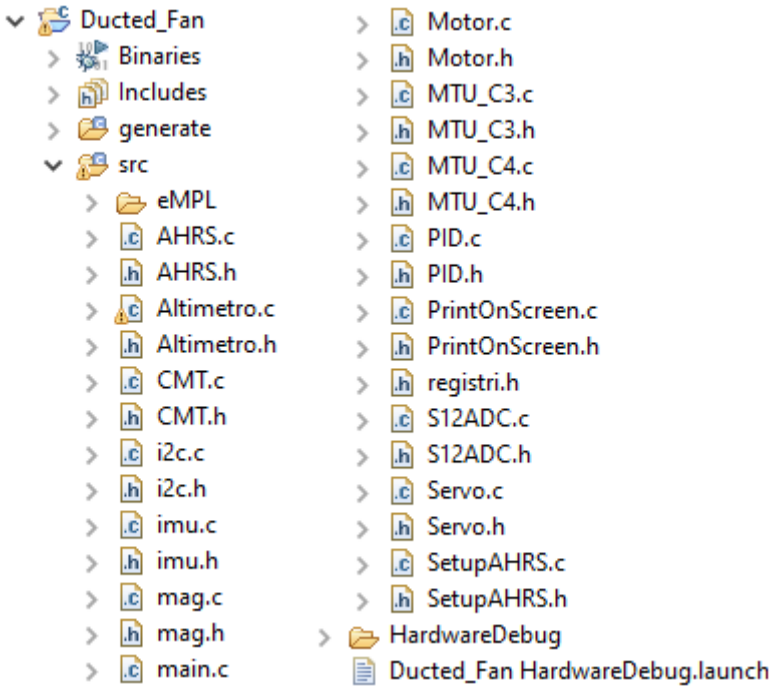


Figura 2.1: Progetto

Come ambiente di sviluppo è stato usato il software e^2 *studio* che viene utilizzato appositamente per sviluppare codice da implementare sui microcontrollori presenti sulle schede RENESAS,

come quella utilizzata per questo progetto. Nella figura 2.1 è mostrata la suddivisione del codice nei rispettivi file, ognuno dei quali si occupa della gestione di un componente specifico del drone. Sono presenti sia file sorgente, .c, che file header, .h, per far si che il codice sia più snello e di facile lettura e comprensione, anche in ottica di sviluppi futuri.
Nel prosieguo di questo capitolo verranno descritte in maniera più dettagliata le parti di codice più importanti per lo sviluppo del progetto.

2.1 IDE

Per lo sviluppo completo del software è stato necessario importare i singoli progetti per il controllo dei componenti hardware e fare il *merge* del codice completo. Questo ha portato a errori iniziali in fase di compilazione che verranno descritti brevemente per evitare problemi futuri in caso di sviluppi sul codice.
Essendo errori dovuti a impostazioni su e^2 *studio* è possibile che non possano verificarsi in egual modo per ogni progetto o potrebbero addirittura non presentarsi.

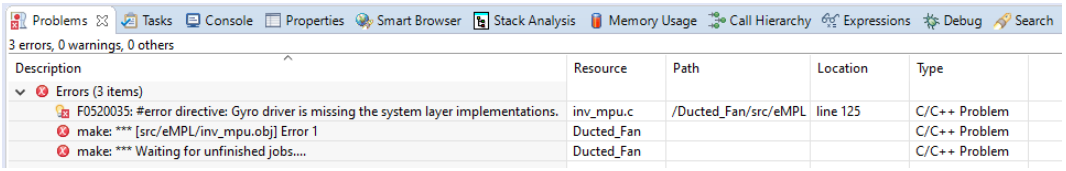


FIGURA 2.2: error directive

L'errore nella figura 2.2 deriva da una mancata definizione, nelle proprietà del progetto, di alcune MACRO necessarie per la corretta compilazione. Per risolvere il problema bisogna seguire i seguenti passaggi:

- Aprire *File*/ *Properties*/ *C/C++ Build*/ *Settings*;
- Nella sezione *Tool Settings* selezionare dall'elenco *Compiler*/ *Source*;
- Nella sezione *Macro definition* inserire le seguenti MACRO
  - EMPL_TARGET_RENESAS;
  - DROTEK_IMU_10DO_V2;
  - MPU6050;

Per applicare le modifiche bisogna, quindi, salvare ed eseguire una nuova compilazione del codice. Le MACRO da aggiungere potrebbero variare a seconda dell'errore specifico da dover risolvere.

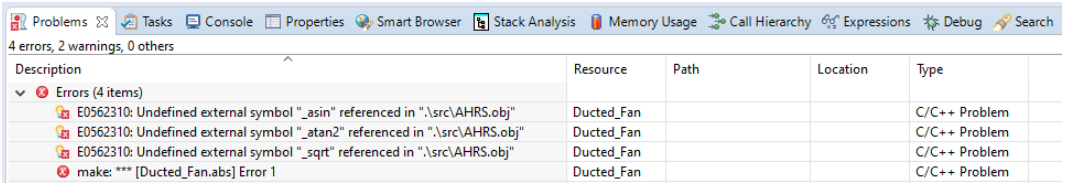


FIGURA 2.3: Undefined external symbol

L'errore nella figura 2.3 è dovuto ad una mancata selezione di librerie matematiche che non vengono quindi inserite in fase di compilazione. Per qualsiasi errore di questo genere, anche con diverse funzioni non rilevate, può essere applicato questo metodo di risoluzione:

- Aprire *File*/ *Properties*/ *C/C++ Build*/ *Settings*;
- Nella sezione *Tool Settings* selezionare dall'elenco *Library Generator*/ *Standard Library*;
- Aggiungere all'elenco di librerie presenti le seguenti:
  - math.h;
  - mathf.h

Così facendo queste due librerie, che si occupano della gestione di funzioni matematiche, verranno aggiunte in fase di compilazione.

2.2 Main

Inizialmente si trovano tutte le inclusioni necessarie per la corretta compilazione del codice, ovvero tutte le librerie utilizzate e gli header file all'interno dei quali sono presenti i prototipi di tutte le funzioni.

```
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <mathf.h>
#include <machine.h>
#include "platform.h"
#include "CMT.h"
#include "PID.h"
#include "Altimetro.h"
#include "Motor.h"
#include "SetupAHRS.h"
#include "i2c.h"
#include "PrintOnScreen.h"
#include "Servo.h"
#include "S12ADC.h"
#include "r_switches.h"
```

FIGURA 2.4: Inclusioni

Successivamente vengono dichiarate tutte le variabili globali presenti all'interno del progetto e i prototipi delle funzioni utilizzate all'interno del file che, per motivi di compilazione, non è stato possibile dichiarare all'interno di specifici header file. Una volta che tutto ciò di cui si ha bisogno è stato dichiarato si procede alla scrittura della funzione principale andando ad implementare il codice per il controllo del ducted fan.

Si procede con l'inizializzazione di tutti i dispositivi hardware che verranno poi utilizzati per il controllo del drone. Queste operazioni sono necessarie affinché, prima di ogni utilizzo del drone, tutti i dispositivi tornino al setup iniziale, che permette l'avvio del drone stesso in maniera sicura. Nelle prime fasi un ruolo cruciale è dato dall'utilizzo degli switch presenti sulla scheda RENESAS che verrà descritto in maniera più approfondita nel paragrafo *Switch*.

Il core del main è costituito dal ciclo while che si occupa di gestire tutte le operazioni di lettura dei sensori e di invio dei dati ai motori e ai servomotori.

Essendo la condizione del ciclo while sempre vera è stato necessario inserire al suo interno una funzione per terminare le operazioni. Questo è stato possibile utilizzando un contatore che, una volta raggiunto il valore impostato da codice, spegne in sicurezza i motori terminando il ciclo.

L'operazione che viene svolta implementando l'attuale codice sulla scheda RENESAS è il raggiungimento di una quota di altezza prefissata nel codice che, grazie all'utilizzo del PID per la gestione dei motori, deve essere raggiunta nel minor tempo possibile. Una volta raggiunta la quota desiderata, il drone rimane in volo finché il ciclo while non è terminato mediante l'utilizzo del contatore che viene descritto precedentemente.

Di seguito viene riportato un diagramma che illustra il funzionamento logico del main.

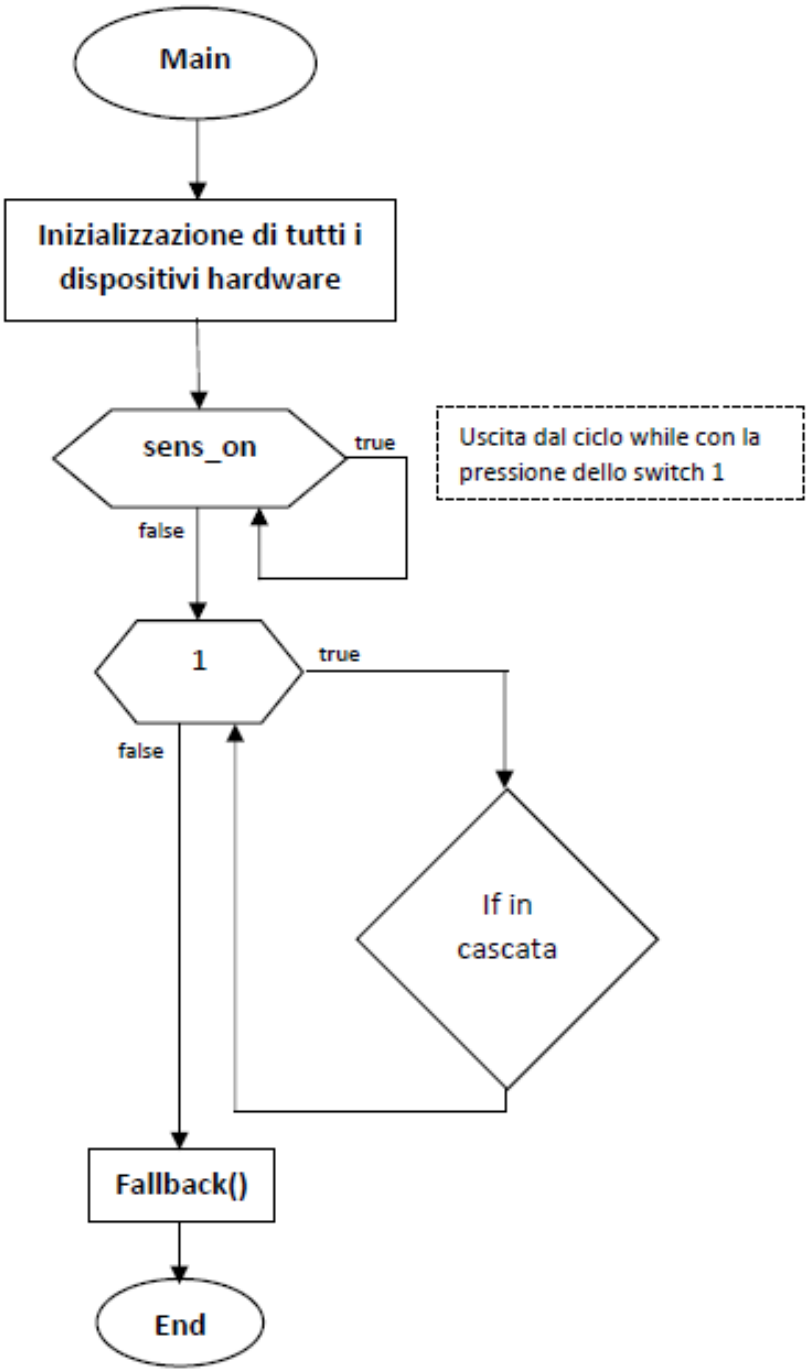


FIGURA 2.5: Diagramma di flusso

L'if in cascata viene riportato nel dettaglio nella pagina successiva.

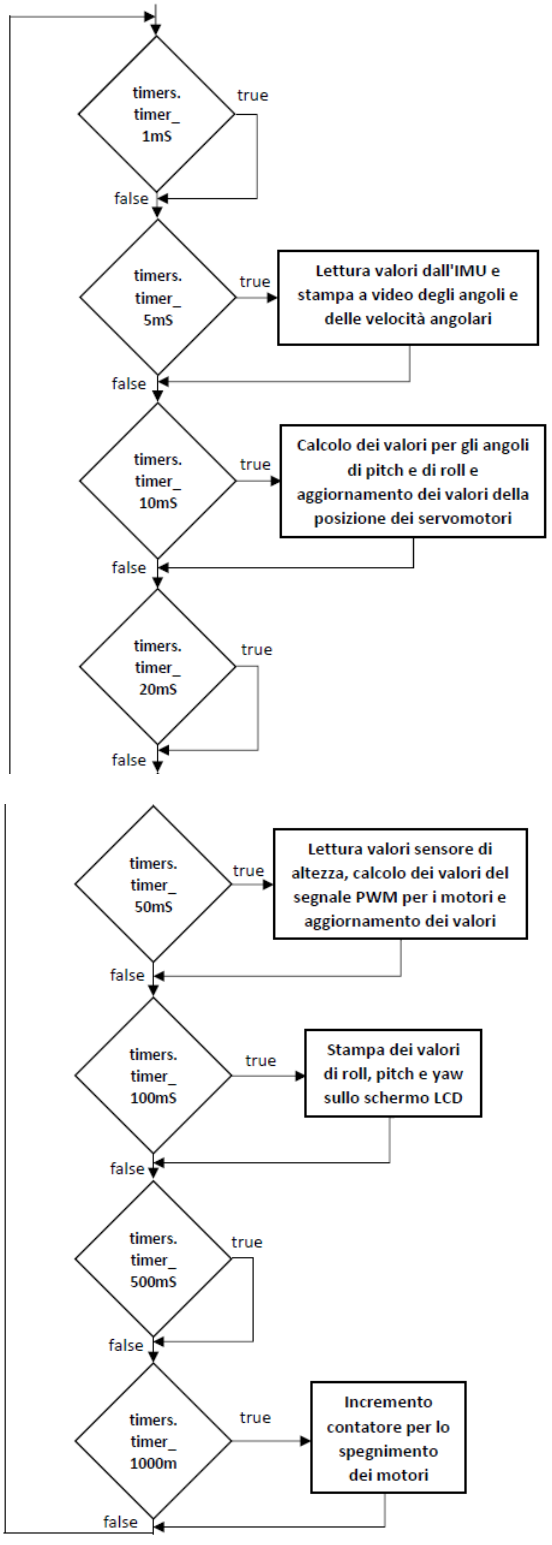


FIGURA 2.6: If in cascata

2.3 Switch

Un ruolo fondamentale per le operazioni di avvio del drone viene svolto dagli switch[1]. Gestiti tramite interrupt inviati alla scheda RENESAS, si occupano di avviare le operazioni principali per il sollevamento in quota del drone.

```
//variabili per regolare accensione motori e sensori
int motore=0;
int dist=1;
int sens_on=1;
```

FIGURA 2.7: Variabili per la gestione degli switch

Nella figura 2.7 viene mostrata la definizione e l'inizializzazione delle variabili utilizzate per il controllo degli switch.

Quando viene alimentata la scheda RENESAS montata a bordo del drone, il software caricato al suo interno viene avviato e le operazioni di inizializzazione descritte nel paragrafo *Main.c* vengono eseguite. Prima di proseguire con l'avvio del ciclo while principale, il software rimane in attesa dell'accensione di tutti i sensori. Questa operazione avviene quando, dopo la pressione dello switch 1, viene generato un segnale di interrupt che verrà gestito dalla scheda RENESAS impostando la variabile $sens\_on = 0$. Questo fa si che l'esecuzione del software possa proseguire entrando nel ciclo while principale.

All'interno del ciclo inizierà l'acquisizione dei dati da parte dei sensori, ma i motori saranno alimentati dal segnale PWM che riesce solamente a tenerli armati, di conseguenza il drone non potrà sollevarsi per raggiungere la posizione desiderata. Anche l'avvio dei motori viene gestito tramite la pressione di uno switch dedicato e si divide in due fasi.

Nella prima, dopo la pressione dello switch 2, viene generato un nuovo segnale di interrupt che verrà, anch'esso, gestito dalla scheda RENESAS impostando la variabile $motore = 1$. Questo fa si che il segnale PWM inviato ai motori sarà uguale al valore necessario a far girare i motori alla velocità minima. Questo è possibile andando ad utilizzare come valore di quota da raggiungere quello letto dal sensore di altezza, in questo modo il PID dei motori rileva come errore, ovvero la differenza tra riferimento e lettura, un valore pari a zero e fornisce in uscita un valore che, dopo opportune conversioni, è dato in ingresso alla funzione *Motor_Write_up*, che verrà descritta più nel dettaglio nel paragrafo *Motori*, che si occupa di aggiornare il segnale di ingresso che gestisce la rotazione dei motori. Una seconda pressione dello switch 2 riporta i motori ad essere

[1] Si trovano alla sinistra dello schermo LCD presente sulla scheda RENESAS

alimentati con un segnale che permette loro di rimanere armati ma che non è sufficiente per farli ruotare alla velocità minima.

Nella seconda fase, che può essere avviata con la pressione dello switch 3, a seguito del segnale di interrupt, la scheda RENESAS imposta la variabile $dist = 0$. Questa operazione avviene dopo che il ciclo while presente nella funzione di gestione dell'interrupt, *sw3_callback()*, arriva al termine, in quanto è stato deciso di fornire all'operatore che effettua la pressione dello switch del tempo per distanziarsi dal drone, evitando problemi di qualsiasi natura, dalla rottura di parti della struttura all'impatto del drone sull'operatore stesso. A seguito di questa operazione il valore di quota effettivo da raggiungere viene dato come riferimento al PID che, dopo aver effettuato i calcoli necessari, fornirà alla funzione *Motor_Write_up* il valore che gestisce la rotazione dei motori.

Quando queste operazioni sono state svolte nella maniera corretta il drone proverà a raggiungere la quota desiderata nel minor tempo possibile e, una volta raggiunta, cercherà di non discostarsi troppo da essa.

```
//pressione switch1, accensione sensori
void sw1_callback()
{
    sens_on=0;
}

//pressione switch2, accensione motori a velocità minima
void sw2_callback()
{
    if(sens_on==0)
    {
        if(motore==0)
            {
                //Avvio motori al minimo
                motore=1;
            }
            else
            {
                motore=0;
            }
    }
}

//pressione switch3, avvio completo dei motori per raggiungimento quota impostata
void sw3_callback()
{
    if(motore==1)
    {
        /* ciclo per ritardare di qualche secondo l'avvio dei motori,
           aggiunto per permettere all'operatore di allontanarsi dal drone
           e mettersi in sicurezza */
        for(int i=0; i<10000000; i++);
        dist=0;
    }
}
```

FIGURA 2.8: Funzioni per la gestione degli interrupt

La pressione degli switch deve avvenire nell'ordine descritto precedentemente ma, qualora questo non dovesse avvenire, il codice è stato implementato in modo tale che non ci siano problemi dovuti ad un ordine errato di pressione.

2.4 Timer

La temporizzazione del ciclo while principale viene gestita dalla scheda RENESAS mediante l'utilizzo del timer presente al suo interno che, una volta inizializzato, è in grado di gestire le operazioni a seconda dell'istante di tempo in cui devono essere eseguite.

```
#pragma interrupt (CMT_isr(vect = VECT(CMT0, CMI0)))
static void CMT_isr(void)
{
    general_timer_mS++;
    timers.timer_1mS = 1;
    if (!(general_timer_mS % 5)) {
        timers.timer_5mS = 1;
        if (!(general_timer_mS % 10)) {
            timers.timer_10mS = 1;
            if (!(general_timer_mS % 20)) {
                timers.timer_20mS = 1;
            }
            if (!(general_timer_mS % 50)) {
                timers.timer_50mS = 1;
                if (!(general_timer_mS % 100)) {
                    timers.timer_100mS = 1;
                    if (!(general_timer_mS % 500)) {
                        timers.timer_500mS = 1;
                        if (!(general_timer_mS % 1000)) {
                            timers.timer_1000mS = 1;
                            if (!(general_timer_mS % 2000)) {
                                timers.timer_2000mS = 1;
                                general_timer_mS = 0;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

FIGURA 2.9: Definizione del timer

Nella figura 2.9 è definita, all'interno del file *CMT.c*, la funzione che si occupa di gestire le funzionalità del timer. Utilizzando la medesima struttura di if in cascata all'interno del main e impostando le condizioni per la verifica del raggiungimento di un certo istante di tempo, è possibile gestire nella maniera corretta tutte le funzioni necessarie per l'acquisizione dei dati da parte dei sensori, indispensabili per la gestione dei motori e dei servomotori.

```
struct timerClocks
{
    uint8_t timer_1mS;
    uint8_t timer_5mS;
    uint8_t timer_10mS;
    uint8_t timer_20mS;
    uint8_t timer_50mS;
    uint8_t timer_100mS;
    uint8_t timer_500mS;
    uint8_t timer_1000mS;
    uint8_t timer_2000mS;
    uint8_t timer_4000mS;
};
```

FIGURA 2.10: struct timerClocks

Per la verifica delle condizioni temporali sopracitate, viene utilizzata la struttura nella figura 2.10, dichiarata nel file *CMT.h*.

```
// Structure containing timer flags
extern struct timerClocks timers;
```

FIGURA 2.11: Definizione struttura

Definendo all'interno del main la struttura *timers* è possibile utilizzare i campi presenti nella figura 2.10 come requisiti per la verifica delle temporizzazioni impostando come condizioni degli if la seguente: $timers.timer\_ * mS$[2].
É fondamentale, quindi, stabilire le priorità di esecuzione delle singole funzioni che sono ripartite nel seguente modo:

- acquisizione dei dati dall'IMU e stampa sullo schermo LCD dei valori ogni 5ms;
- calcolo dei valori da impostare per i flap che gestiscono gli angoli di *pitch* e di *roll* ogni 10ms;
- acquisizione dei valori dal sensore di altezza e calcolo del segnale PWM da impostare sui motori ogni 50ms;
- stampa sullo schermo LCD degli angoli di *roll*, *pitch* e *yaw* ogni 100ms;
- incremento del contatore che si occupa dello spegnimento dei motori dopo il tempo prestabilito ogni 1000ms.

[2]Il valore * va sostituito con il relativo valore di tempo

Tutte le funzionalità sono gestite mediante l'utilizzo di interrupt ogni qualvolta la condizione di temporizzazione risulti verificata, ovvero quando è passato un lasso di tempo necessario a ripetere le operazioni sopracitate. Ogni interrupt viene gestito richiamando la funzione $Callback\_*ms()$[3] ad eccezione dell'acquisizione dei valori dell'IMU con successiva stampa sullo schermo LCD, che vengono eseguite direttamente all'interno dell'if senza passare per una funzione esterna al main, come avviene in tutti gli altri casi. É stato optato per questa soluzione poiché la stampa a video è onerosa in termini di tempi, di conseguenza, essendo la lettura dell'IMU un passaggio fondamentale per la stabilità in volo del drone, si è preferito evitare ulteriori perdite di tempo che potevano derivare dall'utilizzo di una funzione di Callback.
Per ottimizzare i tempi, una volta certi che le misure acquisite fossero corrette, sono state eliminate le stampe sullo schermo LCD, rendendo tutto il processo di gestione del drone molto più reattivo nelle operazioni da svolgere.

2.5 PID

Per verificare il corretto raggiungimento di determinati valori, sia per la gestione dei motori che per i servomotori, è stato implementato un PID a livello software. Preso in ingresso un valore di riferimento da inseguire, mediante l'utilizzo di costanti proporzionali, integrative e derivative opportunamente scelte, è possibile gestire l'uscita per raggiungere nel minor tempo possibile il valore desiderato. All'interno del codice è presente sia un PID per la gestione dei motori, sia uno per la gestione dei servomotori.
Il vantaggio che si ha nell'utilizzo del PID per il controllo del sistema, è costituito dall'utilizzo della medesima struttura in entrambe le situazione, variando solamente i dati che la funzione prende in ingresso.

```
typedef struct
{
    float kp, kd, ki;
    float dt;
    float lastError;
    float ITerm;
    float outMax;
    float outMin;
} PID_config;
```

FIGURA 2.12: Dichiarazione struttura PID

[3] Il valore * va sostituito con il relativo valore di tempo

Nella figura 2.12 è dichiarata, all'interno del file *PID.h* la struttura del PID.

```
/* Create PID structure used for PID properties */
PID_config z_axis_PID;
PID_config Pitch_PID;
PID_config Roll_PID;
```

FIGURA 2.13: Definizione strutture

Per differenziare i PID utilizzati per la gestione degli angoli di *pitch*, di *roll* e per la quota, è necessario definire tre strutture differenti all'interno del main, come riportato nella figura 2.13.

```
*Function name: PID_Compute
float PID_Compute(float input, float setPoint, PID_config* conf);
```

FIGURA 2.14: Dichiarazione $PID\_Compute$

Nella figura 2.14 viene mostrata la dichiarazione, all'interno del file *PID.h*, della funzione utilizzata in tutti e tre i casi sopracitati. Prende in ingresso le variabili *input*, *setPoint* e la struttura *conf* che, a seconda della funzionalità che avranno i PID, assumeranno valori differenti.
Quando è necessario il controllo sull'angolo di *pitch*, la funzione prende come variabile *input* il valore letto dall'IMU dell'angolo di *pitch*; il valore associato alla variabile *setPoint* sarà pari a 0, poiché l'obiettivo è quello di riportare il flap, che gestisce le variazioni di angolo, nella posizione di partenza, ovvero perpendicolare al terreno e con un angolo di 0° rispetto all'asse verticale del drone. La struttura *conf* che viene passata in ingresso sarà *Pitch_PID*, definita nella figura 2.13.
Quando il PID è utilizzato per la gestione dell'angolo di *roll*, alla variabile *input* verrà associato il valore letto dall'IMU dell'angolo di *roll*; la variabile *setPoint* sarà pari a 0 poiché il principio di funzionamento di entrambi i flap è il medesimo, ovvero riportarsi perpendicolari al terreno per fornire stabilità al drone; la struttura *conf* che viene passata alla funzioni è *Roll_PID*, definita nella figura 2.13.
Quando il PID è usato per il controllo di quota, invece, la variabile *input* assume il valore della distanza dal terreno letta dal sensore di altezza, la variabile *setPoint* sarà uguale al valore di quota da raggiungere prestabilito nelle definizioni iniziali del codice. La struttura *conf* avrà in ingresso la struttura z_axis_PID, definita nella figura 2.13.

```
float PID_Compute(float input, float setPoint, PID_config* conf)
{
	/*Compute all the working error variables*/
	float error = setPoint - input;
	conf->ITerm += (error * conf->dt)* conf->ki;
	if((conf->ITerm) > conf->outMax) conf->ITerm = conf->outMax;
	else if((conf->ITerm) < conf->outMin) conf->ITerm = conf->outMin;
	float dInput = (error - conf->lastError) / conf->dt;

	/*Compute PID Output*/
	float output = (conf->kp * error) + (conf->ITerm) + (conf->kd * dInput);

	if(output > conf->outMax) output = conf->outMax;
	else if(output < conf->outMin) output = conf->outMin;

	/*Remember some variables for next time*/
	conf->lastError = error;
	return output;
}
```

FIGURA 2.15: $PID\_Compute$

Nella figura 2.15 viene riportato il codice utilizzato per lo sviluppo software del PID. A seconda dei valori che la funzione riceve in ingresso, descritti precedentemente, genera un'opportuna uscita che permette di raggiungere il valore desiderato.
Quando il PID è utilizzato per il controllo degli angoli di *pitch* e di *roll* l'uscita viene fornita in ingresso alla funzione $Servo\_Write\_deg$, che si occupa di aggiornare i valori della posizione dei servomotori e, di conseguenza, dei flap corrispondenti.
Nel controllo di quota, l'uscita del PID, viene fornita in ingresso alla funzione $Motor\_Write\_up$ che imposta il nuovo valore per la rotazione dei motori che andranno, poi, a generare la spinta sufficiente per far si che il drone raggiunga la quota desiderata.
Per un utilizzo ottimale dei PID è necessario tararli nella maniera ottimale, ovvero in modo tale che i tempi di riposta siano i più brevi possibili per arrivare a regime, facendo si che la risposta sia stabile e non produca troppe oscillazioni attorno al valore desiderato. É necessario, quindi, assegnare i valori alle costanti K_p, K_i e K_d che sono, rispettivamente, il termine proporzionale, integrativo e derivativo, utilizzati per il calcolo dell'uscita del PID.

2.6 Motori

Il funzionamento dei motori è gestito dai file *Motor.h* e *Motor.c*. All'interno del primo si trovano le dichiarazioni di tutte le costanti e le funzioni che vengono utilizzate per svolgere tutte le azioni necessarie per far si che il motore produca, insieme alle eliche, la spinta necessaria per

far si che il drone raggiunga la quota stabilita. All'interno del secondo si trovano le definizioni delle funzioni con il relativo sviluppo del codice.

```
/* Initialize motors */
Motors_Init();

/* Turn on motors relay */
Motors_On();

/* Send arm signal to motors */
Motor_Arm(MOTOR_UPPER);
Motor_Arm(MOTOR_BOTTOM);
```

FIGURA 2.16: Inizializzazione dei motori

Come tutti i dispositivi hardware utilizzati all'interno del progetto, anche per i motori è necessaria un'inizializzazione che deve essere eseguita prima di ogni utilizzo, nella figura 2.16 vengono riportate le funzioni utilizzate. $Motors\_Init()$ si occupa di inizializzare tutti i registri presenti all'interno della scheda RENESAS che si occupa della gestione dei segnali che vengono inviati in ingresso al motore, $Motors\_On()$ avvia il conteggio del timer che gestisce la temporizzazione di tutte le operazioni effettuate sui motori per poter lavorare correttamente sui registri al suo interno, infine, $Motor\_Arm()$ gestisce l'armamento dei motori. Per qualsiasi operazione effettuata sui motori è necessario distinguere su quale dei due motori si sta lavorando, per questo sono state introdotte due costanti, *MOTOR_UPPER* e *MOTOR_BOTTOM*, che identificano, rispettivamente, il motore superiore e il motore inferiore.
Una volta che le inizializzazioni di tutti i dispositivi hardware sono state effettuate ed il codice entra all'interno del ciclo while principale, i motori dovranno variare la loro potenza per raggiungere la quota prestabilita. Queste operazioni vengono svolte dalla funzione *Motor_Write_up()* alla quale vengono passati due valori, il primo per indicare il motore su cui si deve lavorare e il secondo che rappresenta il valore che deve essere impostato nei registri di gestione del motore. Il secondo parametro viene calcolato prendendo come riferimento l'uscita del PID per il controllo di quota e convertendo quel valore, mediante un'opportuna proporzione, in un corrispondente valore di segnale PWM. Il range di valori ammissibili per la gestione dei motori tramite segnale PWM è impostato da codice ed è rappresentato dalle costanti *MOTOR_MIN_UP* e *MOTOR_MAX_UP* i cui valori sono, rispettivamente, 1200 e 1750. Il valore che viene fornito in ingresso alla funzione *Motor_Write_up()* sarà compreso nel range appena descritto e verrà impostato nei registri della scheda RENESAS che erogherà al motore il segnale PWM corrispondente.

2.7 Servomotori

La gestione software dei servomotori è implementata mediante i file *Servo.h* e *Servo.c*, all'interno dei quali sono dichiarate e definite tutte le funzione necessarie. Il loro utilizzo è strettamente collegato ai flap, in quanto si occupano di gestire la loro posizione durante tutta l'esecuzione del software.

```
void Servos_Init()
{
    /* Set all parameter needed by MTU4 unit and start count */
    Initialize_MTU_C4();
    StartCount_MTU_C4();
    Servo_Write_deg(SERVO_PITCH, START);
    Servo_Write_deg(SERVO_ROLL, START);

}
```

FIGURA 2.17: Inizializzazione dei servomotori

Nella figura 2.17 è riportata la sequenza di inizializzazione dei servomotori, necessaria prima di ogni utilizzo per inizializzare tutti i registri nella maniera corretta e per riportare i flap nella loro posizione iniziale, gestita mediante la costante *START* definita uguale a 0, che fa si che i flap siano perpendicolari al terreno.
Il comportamento dei servomotori e la conseguente posizione dei relativi flap viene gestito mediante la funzione *Servo_Write_deg()* che prende in ingresso due valori, il primo per identificare il servomotore utilizzato e il secondo che rappresenta l'angolo che il flap deve raggiungere. Il primo ingresso può assumere due valori, *SERVO_PITCH* e *SERVO_ROLL* per identificare, rispettivamente, i servomotori che gestiscono la variazione degli angoli di *pitch* e di *roll*. Il secondo ingresso assume il valore di uscita calcolato dal PID di controllo dei servomotori.
Rispetto alla posizione di partenza i flap hanno un range di movimento di $\pm 30°$, che è anche il range di valori che fornisce il PID in uscita. Prima di essere però convertiti in valori che poi verranno impostati nei registri della scheda RENESAS, è necessario aggiungere un *OFFSET* di 90° che permette di traslare il range di lavoro in quello ammissibile per il servomotore, che va da 0° a 180°. Per questo progetto è però stato limitato tra 60° e 120° per evitare problemi strutturali al drone.
Il segnale PWM utilizzato per la gestione dei servomotori ammette un valore minimo e un valore massimo, rispettivamente pari a 500 e 2500, che indica il valore in micro secondi delle

posizioni che il servomotore può assumere. Sono definiti utilizzando le costanti *SERVO_MIN_US* e *SERVO_MAX_US*.

2.8 IMU

Tutte le funzioni di inizializzazione e gestione dell'IMU sono presenti all'interno dei file *IMU.h* e *IMU.c*.

Come tutti i dispositivi hardware è presente un'inizializzazione per favorire il corretto funzionamento del dispositivo prima di ogni nuovo utilizzo. Questa procedura è effettuata dalla funzione *imu_init()*, che abilita il funzionamento del giroscopio e dell'accelerometro presenti all'interno del dispositivo e inizializza tutti i registri dei sensori.

Una volta inizializzato, il dispositivo verrà utilizzato all'interno del ciclo *while* principale.

```
int imu_read(IMU_raw* imu_raw, IMU_sens* imu_sens, IMU_temp* imu_temp){

    short data_acc[3] = {imu_raw->accRoll,imu_raw->accPitch,imu_raw->accYaw};
    mpu_get_accel_reg(data_acc, NULL);
    imu_raw->accRoll = data_acc[0];
    imu_raw->accPitch = data_acc[1];
    imu_raw->accYaw = data_acc[2];

    short data_gyr[3] = {imu_raw->gyrRoll, imu_raw->gyrPitch, imu_raw->gyrYaw};
    mpu_get_gyro_reg(data_gyr, NULL);
    imu_raw->gyrRoll = data_gyr[0];
    imu_raw->gyrPitch = data_gyr[1];
    imu_raw->gyrYaw = data_gyr[2];

    imu_temp->accRoll=imu_raw->accRoll / imu_sens->acc_sens;
    imu_temp->gyrRoll = imu_raw->gyrRoll / imu_sens->gyr_sens * 0.01745329252;
    imu_temp->accPitch = imu_raw->accPitch / imu_sens->acc_sens;
    imu_temp->gyrPitch = imu_raw->gyrPitch / imu_sens->gyr_sens * 0.01745329252;
    imu_temp->accYaw = imu_raw->accYaw / imu_sens->acc_sens;
    imu_temp->gyrYaw = imu_raw->gyrYaw / imu_sens->gyr_sens * 0.01745329252;
    return 0;
}
```

FIGURA 2.18: Acquisizione dati IMU

Nella figura 2.18 vengono riportate le operazioni per l'acquisizione dei dati da parte dell'IMU. All'interno della funzione si trovano tre strutture differenti, *imu_raw* che contiene i dati grezzi letti dall'IMU; *imu_sens* che contiene i dati dei sensori dell'IMU e *imu_temp* che contiene i dati rielaborati delle letture effettuate.

Tutte le operazioni e il funzionamento del dispositivo vengono descritte nel capitolo 1, *Introduzione al drone*, paragrafo *IMU*.

2.9 Modifiche al codice

Ora verranno elencate e spiegate le modifiche apportate al codice per migliorarlo.

2.9.1 Cont

```
if(cont>=15)
{
    Motor_Write_up(MOTOR_UPPER, 0);
    Motor_Write_up(MOTOR_BOTTOM, 0);
    Motors_Off();
```

FIGURA 2.19: Modifica al metodo cont

Nella figura 2.19 si può vedere un perfezionamento effettuato al metodo cont, ovvero lo spegnimento automatico dei motori dopo 15 secondi. La modifica è stata quella di sostituire il "==" con ">=", così che se per qualche motivo la variabile cont salta lo step in cui essa vale 15, il drone può spegnersi anche negli istanti successivi.

2.9.2 Atterraggio

```
if(cont>=15)
{
    Motor_Write_up(MOTOR_UPPER, 0);
    Motor_Write_up(MOTOR_BOTTOM, 0);
    Motors_Off();

    while(1)
        nop();
    //commentare le 5 righe di codice sopra e decommentare tutto l'if sotto
    //per usare l'atterraggio controllato

    //atterraggio=1;
}
    //if(cont==25)
    //{
    //   Motor_Write_up(MOTOR_UPPER, 0);
    //   Motor_Write_up(MOTOR_BOTTOM, 0);
    //   Motors_Off();

    //   while(1)
    //   nop();
    //}
```

FIGURA 2.20: Codice atterraggio

Nella figura 2.20 viene mostrato il codice utilizzabile per far atterrare il drone dopo 25 secondi, ovvero si imposta la quota minima al drone che rallenterà diminuendo la potenza fornita ai

motori, per poi spegnerli, evitando un forte impatto col suolo. Ovviamente questa soluzione va utilizzata in alternativa al "metodo cont" e soltanto dopo la taratura perfetta dei PID.

2.9.3 Rimozione stampe a video

Nella figura 2.21 si può vedere la riga di codice da commentare/decommentare per visualizzare o meno le stampe a video dei dati racconti in tempo reale. Considerando che i dati vengono raccolti e salvati tramite pc, si è deciso di eliminare le stampe sullo schermo LCD, rendendo tutto il processo di gestione del drone molto più reattivo nelle operazioni da svolgere.

```
//display_results(distanza);
```

FIGURA 2.21: Stampe a video commentate

2.9.4 Calcolo distanza effettiva

Nella figura 2.22 è mostrato lo script matlab di come si utilizzano le matrici di rotazione per ottenere la quota reale del drone partendo da quella vista dai sensori, dato che quando il ducted fun è inclinato l'altezza letta è falsata.

```
syms c_phi s_phi c_th s_th x y z

R_phi=[1 0 0
       0 c_phi -s_phi
       0 s_phi c_phi];

R_th=[c_th 0 s_th
      0 1 0
     -s_th 0 c_th];

P=[x
   y
   z];

R=transpose(R_phi*R_th);

P_f= R*P;
```

FIGURA 2.22: Calcolo distanza rot

```
//lettura valori sensori di altezza in mm
distanza = Read(temp);
//conversione tramite rotazioni
distanza_rot= cos(roll_deg*PI/180)*cos(pitch_deg*PI/180)*distanza;
//conversione valori in m
distanza_metri = (distanza_rot/1000);
```

FIGURA 2.23: Calcolo distanza rot

Nella figura 2.23 viene applicata la formula, dove roll deg e pitch deg sono gli angoli di inclinazione del drone e distanza è la quota letta dell'imu.

Capitolo 3

Costruzione gabbia

In questo capitolo verrà affrontato l'argomento della gabbia: materiali utilizzati, tecniche di costruzione e modifiche successive. La struttura ha una doppia funzione:fare da sostegno al drone durante le prove e proteggere gli operatori da eventuali elementi che potrebbero staccarsi e schizzare a forte velocità verso l'esterno. Come si evince dalle immagini 3.1 , 3.2 e 3.3, la

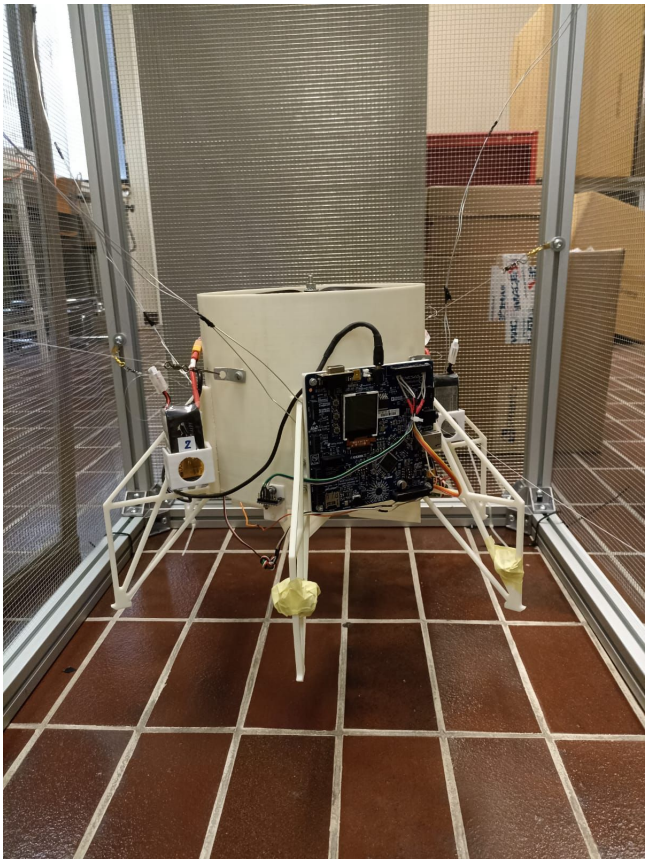


Figura 3.1: Gabbia vista laterale ravvicinata

FIGURA 3.2: Gabbia vista dall'alto

struttura principale è costituita da 12 profili in metallo, di cui 8 di lunghezza 76cm a formare le facce parallele al terreno e 4 di lunghezza 1m posti perpendicolarmente al suolo.
Per tenere uniti i profili sono state usate 20 squadre con altrettanti dadi e viti.
Il parallelepipedo è chiuso per 5 facce su 6 con una rete di maglia 6mm, infatti la base è semplicemente aperta e appoggiata al pavimento.
Una delle facce laterali si chiude con un pannello facilmente rimovibile, tramite dei ganci, così da consentire all' operatore di interagire col drone.
Per coprire eventuali parti taglienti sulle sporgenze, è stato utilizzato del cartone sulla parte superiore e sul pannello.
Delle fascette sono state applicate ai profili per sostenere la rete e il cartone. Per evitare un brusco impatto col terreno allo spegnimento dei motori sono stati legati al drone 3 fili di ferro che escono dalla parte superiore della gabbia, così che l'operatore può tenerlo sollevato dei pochi cm che servono.
Per fissare il drone ai 4 profili verticali, così da limitarne il volo prima e durante della taratura dei PID, sono stati utilizzati moschettoni da pesca e fili di nylon, sfruttando 4 ganci fissati

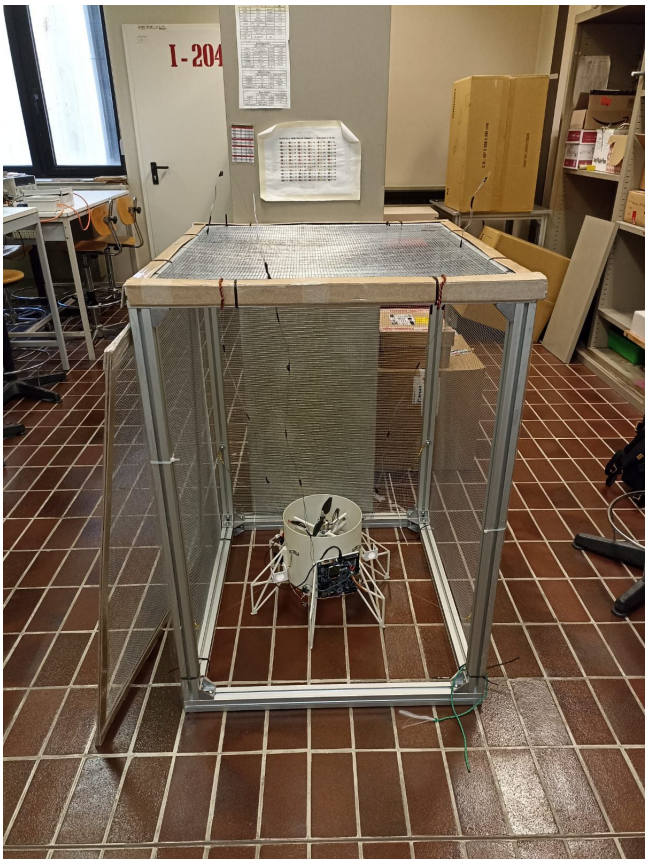


FIGURA 3.3: Gabbia vista laterale distante

al drone precedentemente. Successivamente, durante la taratura dei PID, sono stati aggiunti altri fili per limitare i gradi di libertà del ducted fun e concentrarsi su uno specifico movimento.

Capitolo 4

Salvataggio e rielaborazione dati

In questo capitolo verrà spiegato come acquisire i dati durante le prove di volo e come pulirli ricavandone dei grafici da analizzare. In particolare verranno elencati i vari passaggi per far partire la registrazione dei dati su e^2 *studio*, dove recuperarli finita la prova di volo, come funziona lo script di matlab e come utilizzarlo, ed infine un riassunto dei limiti riscontrati e le soluzioni applicate.

4.1 Registrazione dati su e^2studio

Sulla destra della schermata di e^2 *studio* c'è l'elenco delle variabili "in watch", cioè quelle che potremo visionare in diretta con la prova e che potremo registrare. Per aggiungere una variabile basta cliccare su "Add new expression" e scriverne il nome. Quando una variabile viene aggiunta, bisogna cliccarci sopra con il tasto destro e selezionare dal menù "Real-time Refresh" per attivarne la visione in live (l'operazione va effettuata quando si è in debug mode) .
Dopo aver effettuato le procedure iniziali, che vedremo dettagliatamente nel prossimo capitolo, e prima di agire sugli switch del drone, bisogna attivare la registrazione dei dati: basta fare tasto destro su una delle variabili in watch e selezionare "start recording", inserire il filename, selezionare Timestamp, salvare e confermare. Tutti questi passaggi sono visibili nelle immagini 4.1 , 4.2.

Finita la prova di volo, bisogna fare tasto destro su una delle variabili e cliccare su "Stop Recording" per fermare la registrazione, poi si può terminare il programma.

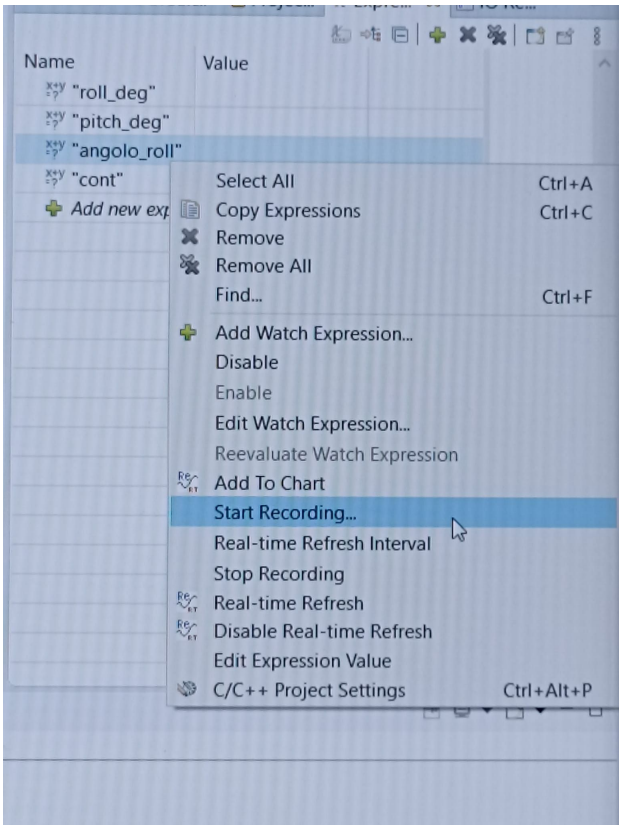


FIGURA 4.1: Start recording

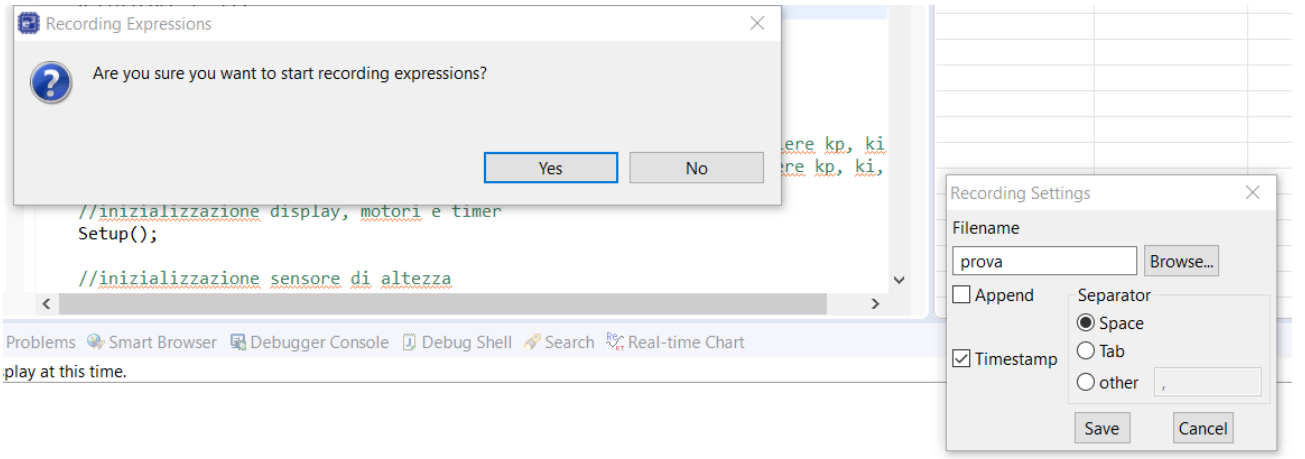


FIGURA 4.2: Timestamp

4.2 Script Matlab

Una volta completata la prova di volo bisogna recuperare i dati salvati nella cartella "eclipse", che si trova nella cartella "e2studio" che a sua volta è all'interno della cartella "Renesas". Il documento va spostato all'interno della cartella in cui si trova lo script che andremo ad utilizzare. Fatto ciò si può aprire lo script su matlab, che apparirà come nelle immagini 4.3, 4.4 e 4.5: nella

seconda riga, fra virgolette, va inserito il nome del file desiderato.

Il primo comando che incontriamo è "readtable", che sistema i nostri dati in una tabella con in

```
clear

dati = readtable('a11');

indice=1;
dati1=table2array(dati);
dif=dati1(1,5);
dati1(:,5)=dati1(:,5)-dif;
T=dati1(:,5);


Pitch_deg=(dati1(:,2));
figure(1)
plot(T,Pitch_deg)
title('Pitch deg')
xlabel('Tempo [ms]')
ylabel('Pitch deg [°]')

Roll_deg=(dati1(:,1));
figure(2)
plot(T,Roll_deg)
title('Roll deg')
xlabel('Tempo [ms]')
ylabel('Roll deg [°]')
%
```

FIGURA 4.3: Script matlab

ascissa le variabili(più il timestamp) e in ordinata il numero della lettura.

Il secondo comando è "table2array" che trasforma il tipo dei dati da table a double.

Nelle due righe successive viene risolto un problema del timestamp. Il primo valore che risulta è un numero molto elevato, perchè conta il "tempo di vita" della scheda, e poi viene incrementato ad ogni misurazione: basta sottrarre ad ogni valore del timestamp il primo valore da lui registrato in quella prova, così da ottenere una scala progressiva partente da zero. Nel fare questa operazione bisogna fare attenzione ad utilizzare la colonna del timestamp(in questo caso la numero 5), che è sempre l'ultima, quindi basta contare le variabili ed aggiungere uno(nel caso in figura 4 variabili più il timestamp 5). Nella riga 9 salviamo il vettore dei tempi delle misurazioni in una variabile chiamata T per comodità.

Il resto del codice serve per ottenere i vari plot delle variabili, facendo attenzione a decommentare quelle non prese in esame in quella misurazione. Nel caso in figura, ad esempio, abbiamo 4 variabili misurate(PitchDeg, RollDeg, AngoloRoll e cont) più il timestamp. Dalla figura 4.1 possiamo vedere in che ordine sono inserite, così da rispettarlo in matlab: per la variabile

RollDeg, che è la prima, dobbiamo quindi prendere la prima colonna della tabella dei dati, e così via per tutte le altre variabili. Per ognuna di esse, per ottenere i grafici basati sulle misurazioni e il loro tempo, basta utilizzare il comando plot. Per una migliore visualizzazione ad ogni grafico è stato assegnato un titolo ed il nome degli assi.
Per quanto riguarda il comando "figure()", necessario in ogni plot, basta che ci sia un numero diverso per ogni variabile.

```
Angolo_roll=(dati1(:,3));
figure(10)
plot(T,Angolo_roll)
title('Angolo roll')
xlabel('Tempo [ms]')
ylabel('Angolo roll [°]')
%
% Angolo_pitch=(dati1(:,5));
% figure(11)
% plot(T,Angolo_pitch)
% title('Angolo pitch')
% xlabel('Tempo [ms]')
% ylabel('Angolo pitch [°]')
%
% Distanza_rot=(dati1(:,2));
% figure(12)
% plot(T,Distanza_rot)
% title('Distanza rot')
% xlabel('Tempo [ms]')
% ylabel('Distanza rot [mm]')
```

FIGURA 4.4: Script matlab pt.2

```
% Out_value=(dati1(:,3));
% figure(6)
% plot(T,Out_value)
% title('Out value')
% xlabel('Tempo [ms]')
% ylabel('Out value')
%
cont=(dati1(:,4));
figure(21)
plot(T,cont)
title('cont')
xlabel('Tempo [ms]')
ylabel('cont')
```

FIGURA 4.5: Script matlab pt.2

4.3 Limiti e soluzioni

Durante i primi test di volo, per registrare i dati veniva utilizzata un' altra funzione fornita da e^2 *studio*: il Real-time Chart, ovvero un grafico che si aggiorna in live al susseguirsi delle misurazioni. Per aggiungere una variabile basta metterla in watch, e col tasto destro selezionare "add to chart".

Finita la prova, prima di terminare il programma, bisogna cliccare col tasto destro sul grafico e fare "export to CSV".

Questi passaggi sono visibili nelle immagini 4.6 e 4.7.

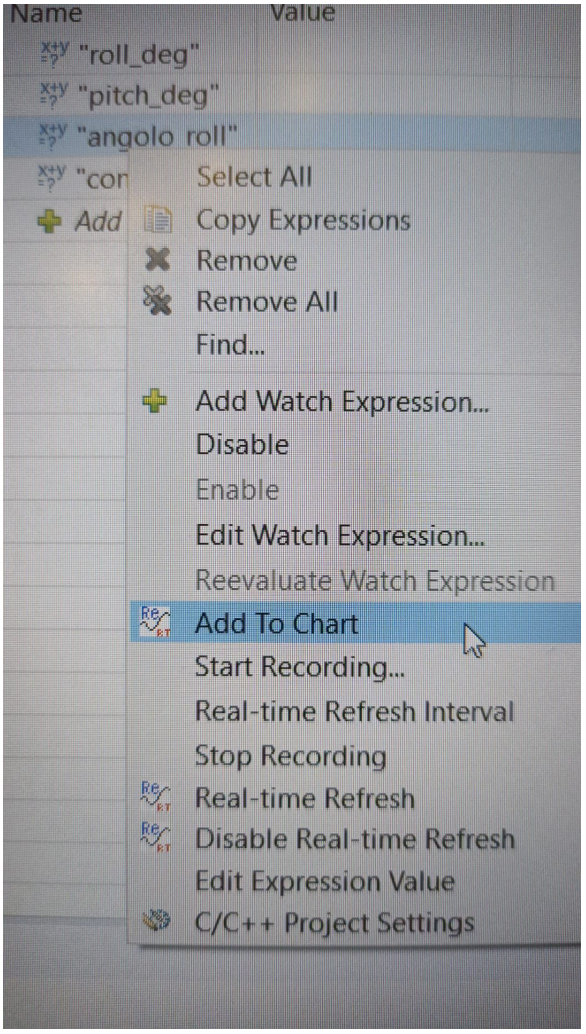


FIGURA 4.6: Add to chart

Nonostante questo metodo di salvataggio permette di avere più letture nell'unità di tempo, ha il problema che ci sono dei buchi temporali in cui non registra o perde dei valori, rendendo difficile pulirli con matlab e falsando l'analisi dei grafici.

Utilizzando "start recording" invece, la lettura dei dati è perfetta, anche se meno accurata in

quanto il tempo tra una misurazione e la successiva è maggiore. Riducendo il numero di variabili in watch contemporaneamnte si riesce ad avere dei tempi di lettura paragonabili agli altri, per cui si è deciso di concentrarsi su tre o quattro variabili per volta, come vedremo nel prossimo capitolo, durante la taratura dei PID.

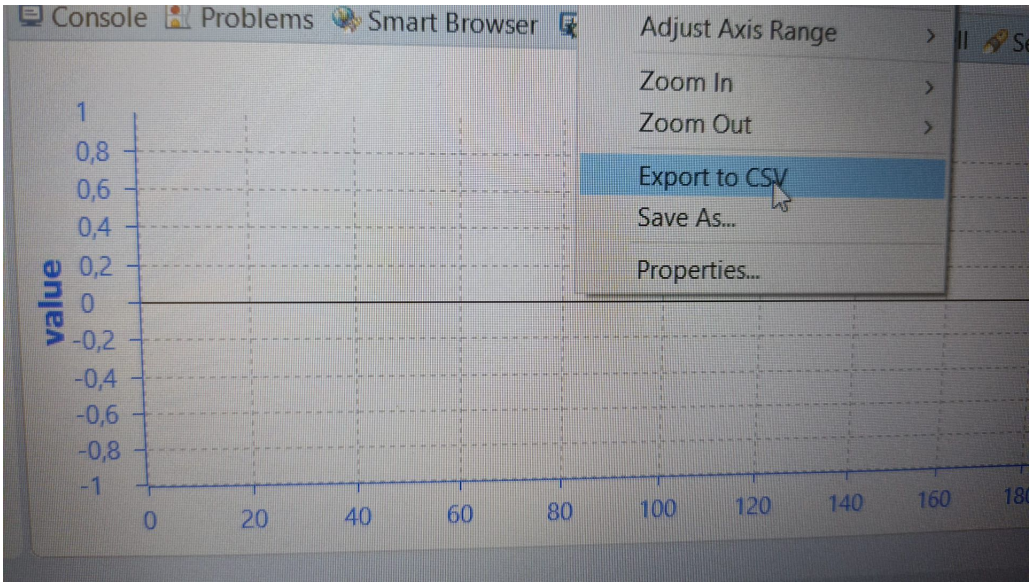


FIGURA 4.7: Export to CSV

Capitolo 5

Prove di volo e analisi dei dati

In quest' ultimo capitolo si descriverà l'evoluzione dei test col ducded fun, dalle prime prove fino al settaggio dei vari PID, analizzando ogni volta i problemi incontrati, le soluzioni applicate e le modifiche alla gabbia.

5.1 Come si fa una prova di volo

In questa sezione vengono elencati tutti i passaggi da fare per eseguire un test, dalla preparazione iniziale ai grafici finali.
1) Mettere in watch tutte le variabili che si vogliono monitorare.
2) Collegare le 2 batterie al drone tramite il ponte, poi attivare servomotori tramite un tastino facilmente rintracciabile, ed infine con l'altro tastino si attiva il resto dell' hardware.
3) Collegare il pc alla scheda tramite cavo usb, chiudere la gabbia, ed eseguire il debug da e^2 *studio* (premere il simbolo dello scarafaggio e poi due volte il tasto play).
4) Mettere le variabili in real time(ciò va fatto solo alle variabili appena aggiunte). Iniziare la registrazione tramite start recording.
5) Tramite lo sportello apribile agire sugli switch: premere il primo per accendere i sensori, verificando che le variabili in real time stiano variando, premere il secondo per far girare i motori al minimo, premere il terzo per far partire i motori al massimo(o alla potenza impostata). Se necessario tenere i fili per evitare l'impatto allo spegnimento dei motori. Si noti che premere troppo lo switch due, dato che agisce tramite interrupt, equivale a premerlo due volte, ed il risultato è che le eliche partono e si fermano subito dopo. In tal caso si può riprendere la prova

premendo di nuovo lo switch due.
6) Una volta spenti i motori, fare stop recording e poi termina(quadratino rosso). Aprire la gabbia per scollegare il cavo usb, premere il tasto dei servomotori, poi l'altro ed infine staccare le batterie(ricordarsi di controllare la tensione delle batterie ogni tanto, così da non scendere sotto la soglia).
7) Finita la prova, recuperare il documento con i dati registrati, spostarlo nella cartella dello script e aprire matlab. Inserire il nome del documento nello script, verificare che le variabili in watch corrispondano a quelle non commentate nello script, e fare run(triangolo verde). Si avranno in output i grafici desiderati.

5.2 Test iniziali di controllo

Nei primi test effettuati l'obiettivo era fare un check del dispositivo e visionare che tutto funzionasse a dovere.
Subito si è notato che il drone pendeva totalmente da un lato, senza cercare di reagire. Tramite un video fatto dall'alto si è capito che i flap si inclinavano dalla parte sbagliata, quindi spingendo il ducted fun nel verso in cui già stava andando. Controllando il codice si è visto che nella formula dell'output dei flap c'era un segno meno di troppo, è bastato toglierlo per vedere subito i miglioramenti alla prova successiva.
Il resto era tutto funzionante: codice, gabbia e hardware.

5.3 Prime prove di volo

Le prove successive sono state fatte per prendere confidenza con i procedimenti da eseguire, imparare a registrare i dati e vedere sommariamente il comportamneto del drone.
Per la registrazione, come spiegato precedentemente, è stato usato un metodo diverso inizialmente, poi abbandonato per quello attuale (start recording). Infatti dopo qualche prova in cui i dati erano mal registrati, si è cercata un'alternativa. Per ritrovare il documento con i dati è servita una ricerca su tutte le cartelle del pc, dato che e^2 *studio* non specifica dove venissero salvati.
Il ducded fun invece reagiva in maniera confusionaria, dovuto alla non taratura dei PID(quota e

flap) e ad altri perfezionamenti che vedremo successivamente.

5.4 Primi tentativi di taratura

Con l'obiettivo di tarare il PID di quota, sono state applicate le matrici di rotazione alla lettura della quota fatta dall'IMU. In questo modo si considerano gli angoli di inclinazione del drone e si può risalire alla reale quota raggiunta.
Inizialmente si è cercato di tarare il PID di quota dando un'altezza fissata come obiettivo, tenendo in watch la distanza e gli angoli di inclinazione del drone e dei flap. Osservando i loro grafici sono stati modificati i parametri del PID(proporzionale, integrativo, derivativo), ma nonostante questo non c'erano miglioramenti perchè non essendo tarati i flap, il ducted fun non riusciva a stabilizzarsi e pendendo da un lato faceva troppa fatica a raggiungere e mantenere la quota stabilita.

5.5 Taratura PID flap

Dopo le prove appena descritte, si è deciso di concentrarsi sulla taratura dei flap, e solo successivamente tornare al PID di quota.
Per farlo, bisogna focalizzarsi su un solo flap, così sono stati messi in tensione i due fili paralleli al flap del roll(asse x), in modo da consentire movimenti solo attorno a quell'asse(il servomotore del pitch è stato spento-vedi figura 5.1). Per mettere in equilibrio il drone(roll drone=0) dopo

```
//aggiornamento valori servomotori
Servo_Write_deg(SERVO_ROLL, angolo_roll);
//Servo_Write_deg(SERVO_PITCH, angolo_pitch);
```

FIGURA 5.1: Servomotore pitch spento

aver tirato i fili, sono stati aggiunti 2 pesetti nel lato della schehda.
Dopo alcune prove fatte così, visti gli scadenti risultati, si è deciso di fissare la potenza dei motori, alla ricerca della velocità di hovering: si cercava la percentuale della potenza massima da dare ai motori affinchè il ducted fun si alzasse un minimo e restasse a quella quota.
La potenza in questione però era troppo poca per generare un flusso in grado di far tornare il drone ad angolo zero(roll=0 era l'obiettivo da mantanere)quando si inclinava dopo la partenza.

Per cui il drone è stato sollevato più in alto, così da aumentare la distanza tra eliche e terreno per avere più spinta a parità di potenza.
Le prove successive sono state fatte imponendo un setpoint all'angolo di inclinazione del drone, pochi gradi di roll, per vedere se fosse in grado di spostarsi verso l'obiettivo e rimanerci. Inoltre si è aumentata la potenza dei motori, dato che quella di hovering non era sufficiente. Questo aumento ha causato un altro problema: i pochi centrimetri che il ducted fun aveva di libertà verso l'alto scatenavano un effetto molla, perchè andando in tensione i fili il drone iniziava a rimbalzare, falsando totalmente i dati raccolti. Per ovviare a ciò sono stati aggiunti dei fili perpendicolari a quelli tesi in modo da impedire totalmente l'effetto molla(i nuovi fili sono stati agganciati sullo stesso punto del drone di quelli tesi, così da non influenzare il movimento di roll, che rimane l'unico possibile).
Dopo tutti questi aggiustamenti si sono iniziati a vedere i primi miglioramenti, infatti dopo qualche prova, analizzando i dati e variando i parametri del PID, si è riusciti a raggiungere il set point.

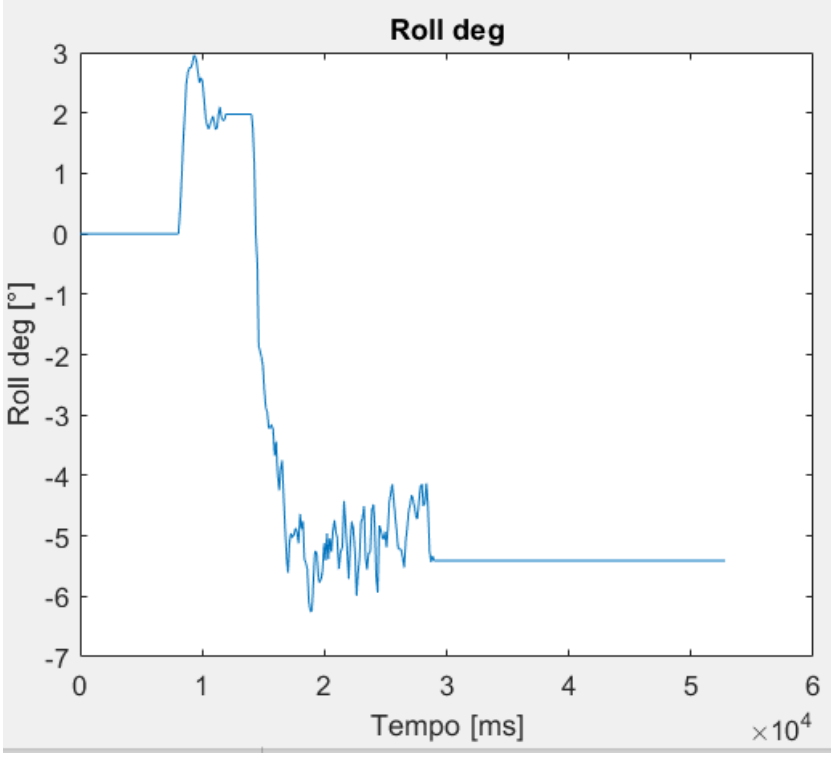


FIGURA 5.2: SetPoint roll raggiunto

Come si può vedere dalle immagini 5.2 e 5.3, il ducted fun ha raggiunto l'obiettivo dei -5° senza saturare il flap(intervallo[-30°; 30°]). Si può notare che all'inizio il flap è più inclinato(-20°circa) perchè il drone è lontano dal setpoint, ma più quest'ultimo arriva in prossimità dei -5° e meno si inclina il flap, fino a che entrambi trovano l'equilibrio restando circa costanti(drone -5°,flap -9°).

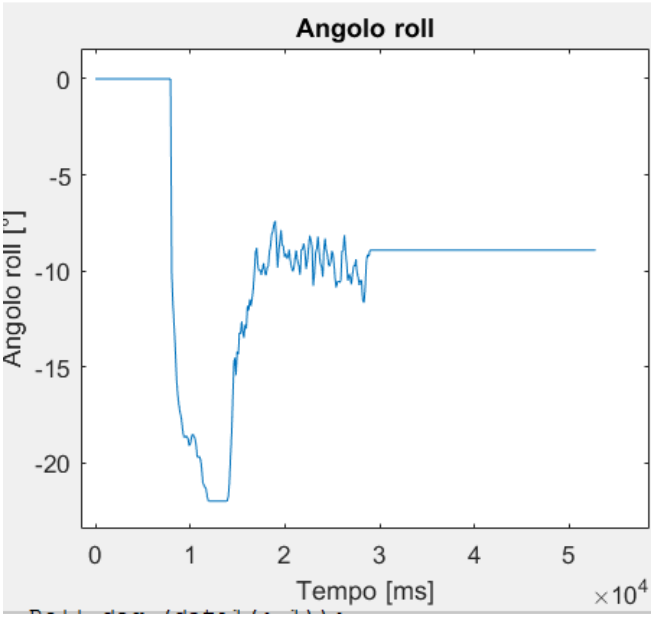


FIGURA 5.3: Flap roll non saturo

Dopo aver raggiunto questo setpoint, si è provato ad arrivare a -10°, ma senza successo come si può notare dalle immagini 5.4 e 5.5: il drone arriva a circa -8° nonostante il flap sia saturo, ovvero ha dato il suo massimo contributo per raggiungere l'obiettivo. Si può affermare che quindi il flusso prodotto non è sufficiente a far inclinare il drone di -10.

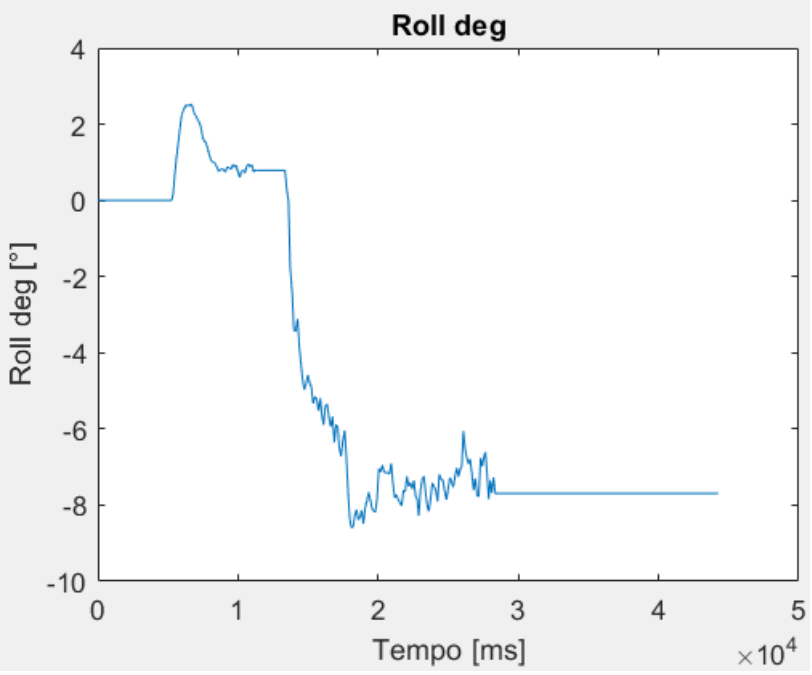


FIGURA 5.4: Setpoint non raggiunto

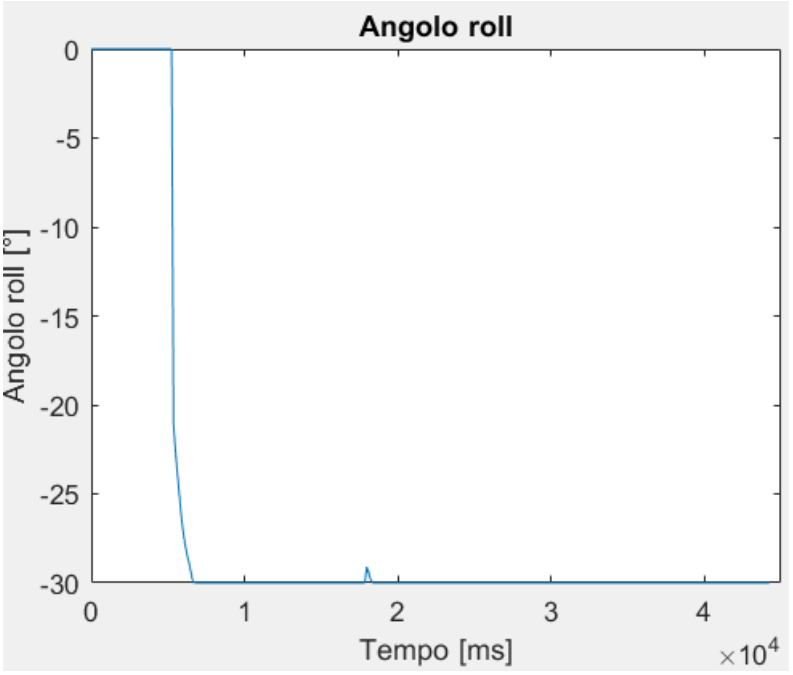


FIGURA 5.5: Flap roll saturo

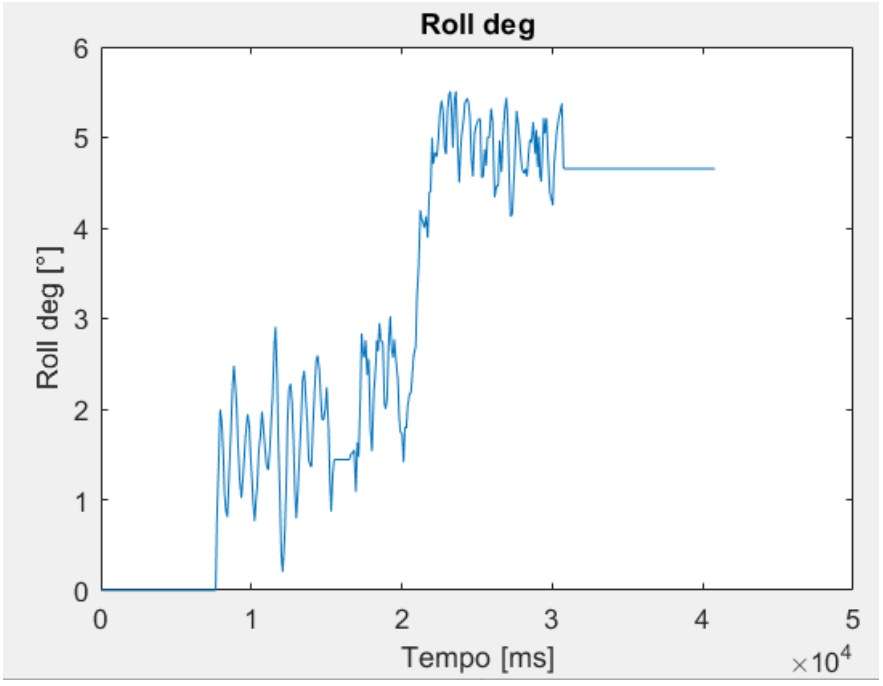


FIGURA 5.6: Setpoint raggiunto

Fallita la prova a -10°, si è provato un setpoint positivo: +5°.
Stranamente il drone non riesce ad arrivarci, escluso un test(figure 5.6 e 5.7), nonostante i gradi di cui inclinarsi siano gli stessi della controparte negativa.

Ciò è probabilmente causato da un non perfetto allineamento tra il flap del roll ed i fili tesi, che

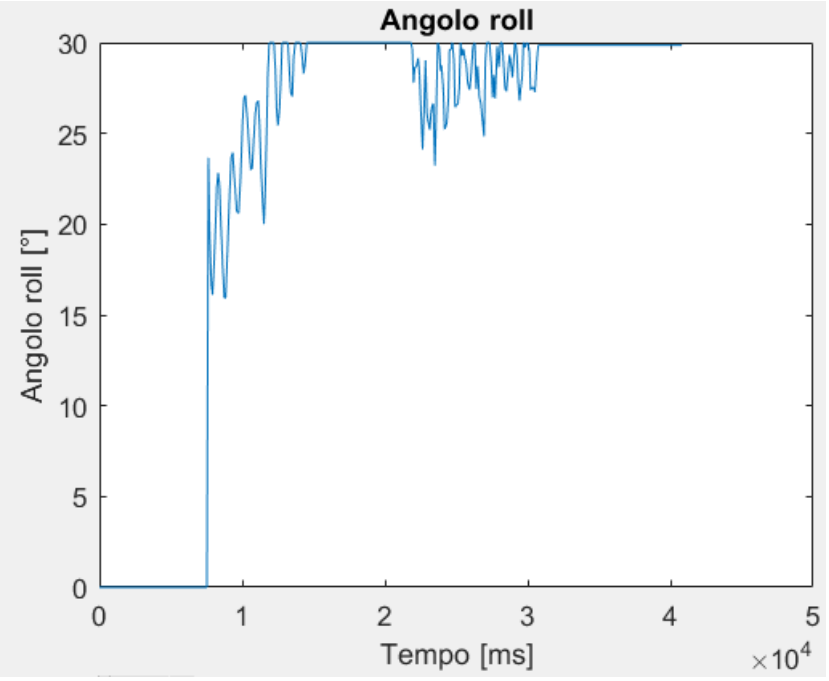


FIGURA 5.7: Flap quasi totalmente saturo

potrebbe rendere più difficoltosa la rotazione in un verso piuttosto che nell' altro.

Questa è certamente la prossima modifica da effettuare per poter proseguire la taratura del flap.

Capitolo 6

Conclusioni

Si conclude questo elaborato in cui è stato affrontato il problema di come tarare i PID di un ducted fun in assoluta sicurezza.
Sono state analizzate tutte le componenti hardware utilizzate e, successivamente, è stato descritto lo sviluppo del software implementato sulla scheda RENESAS per la gestione del drone. In particolare, sono state analizzate nel dettaglio tutte le sezioni di codice importanti per il corretto funzionamento del drone e sono state riportate le linee di codice ritenute cruciali nello sviluppo di parti essenziali del software.
E' stata spiegata la procedura per la costruzione della gabbia, fondamentale per la protezione dell' operatore e per il sostegno al drone.
E' stato mostrato come registrare i dati durante le prove di volo e come utilizzare matlab per ricavarne grafici da analizzare.
Infine sono state descritte le modalità dei test svolti, con particolare attenzione alle problematiche e alle possibili soluzioni, e sono stati riportati i grafici dei maggiori risultati ottenuti.
Per il continuo di questo progetto bisognerà prima di tutto allineare i fili tesi con il flap del roll e poi sostituire i flap con degli altri più lunghi, così da sfruttar meglio il flusso e poter raggiungere angoli di inclinazione maggiori. Dopo aver tarato il PID del flap del roll, quello del pitch dovrebbe avere parametri simili. Infine ci si potrà concentrare sul PID di quota, sfruttando il fatto che il ducted fun sarà più stabile.

Bibliografia

1)Matteo Biagiola. Modellazione e Simulazione di un Ducted Fan. versione italiana edition,2014.

2)Daniele Di Muzio. Implementazione firmware di un microcontrollore per il controllo del volo di un drone a flusso convogliato - Tesi di Laurea triennale

Ringraziamenti

Innanzitutto vorrei ringraziare il Prof. Andrea Bonci per avermi dato la possibilità di svolgere questo percorso di tirocinio. Ringrazio il tecnico Antonio Pinelli per l'aiuto nelle parti più pratiche e ringrazio anche l'Ing. Giuseppe Antonio Scala e l'Ing. Giacomo Nabissi per tutti i consigli ricevuti.

Ringrazio la mia famiglia per avermi supportato, non solo economicamente, in questo percorso e soprattutto nella vita di tutti i giorni.

Ringrazio la mia ragazza Francesca e tutti gli amici, vecchi e nuovi.

Un grazie anche a tutti i professori, che a modo loro hanno fatto parte di questa avventura.

L'ultimo ringraziamento, forse il più grande, va a Daniele. Conosciuto da poco, proprio grazie a questo tirocinio, e già diventato un riferimento, un amico. Grazie per il grande aiuto, per tutti i consigli e soprattutto per la compagnia.