



UNIVERSITÀ POLITECNICA DELLE MARCHE

DIPARTIMENTO INGEGNERIA
DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica e
dell'Automazione

**CONTROLLO DI DRONI MULTIROTORE TRAMITE
APPROCCIO HARDWARE-IN-THE-LOOP.**

CONTROL OF MULTIROTOR DRONES USING HARDWARE-IN-THE-LOOP APPROACH.

Laureando:

Di Antonio Davide

Relatore:

Prof. Freddi Alessandro

Correlatore:

Dott. Baldini Alessandro

Sessione di Laurea Straordinaria

Anno Accademico 2021/2022

*“Per correr miglior acque alza le vele
omai la navicella del mio ingegno,
che lascia dietro a sé mar sì crudele.”*

D. Alighieri

Indice

1	Introduzione	4
1.1	Il veicolo aereo senza pilota	4
1.2	I sistemi reali: il problema odierno	5
1.3	Hardware-In-The-Loop (HITL)	5
1.4	Obiettivo della tesi	6
2	Requisiti di progetto	7
2.1	Architettura PX4	7
2.2	Software di progetto	10
2.3	Hardware di progetto	12
3	Configurazione del sistema HITL	15
3.1	Configurazione PixHawk tramite QGroundControl	16
3.2	Hardware setup tramite UAV Toolbox Support Package for PX4 Autopilots	18
4	Il modello del quadrotore	21
4.1	Equazioni del moto	23
4.2	Parametri e caratteristiche del modello simulato	25
5	Il controllo del quadrotore	27
5.1	Controllo nativo di posizione e di velocità nell'Earth Frame (Outer-Loop)	31
5.2	Controllo nativo dell'angolo e della velocità angolare nel Body Frame (Inner-Loop)	32
5.3	Allocazione e scalatura in PWM (Mixer)	34
5.4	Definizione della nuova legge di controllo	36
5.5	Nuovo controllo di posizione e di velocità nell'Earth Frame (Outer-Loop)	36
5.6	Nuovo controllo dell'angolo e della velocità angolare nel Body Frame (Inner-Loop)	38
6	Implementazione e analisi dei dati	40
6.1	Implementazione della nuova legge di controllo	40
6.2	Analisi della nuova legge di controllo	44
6.3	Analisi del tempo di esecuzione	49
7	Conclusioni	52

Capitolo 1

Introduzione

1.1 Il veicolo aereo senza pilota



Figura 1.1: Quadrirotore generico [5]

Un veicolo aereo senza equipaggio (Unmanned Aerial Vehicle - UAV) è un aereo senza esseri umani a bordo (Figura 1.1). Utilizzati dal XX secolo in campo militare, gli UAV sono oggi impiegati anche per molte applicazioni non militari: fotografia aerea, agricoltura, monitoraggio ambientale, sorveglianza, trasporto e intrattenimento.

Secondo il Dipartimento della Difesa degli Stati Uniti, gli UAV sono classificati in 5 categorie, a seconda della dimensione, del peso al decollo, dell'altitudine operativa e del campo di velocità in cui il veicolo può operare. Ad esempio:

- per un drone di categoria 1:
 - Misura: piccolo;
 - Peso: $< 10Kg$;
 - Altitudine operativa: $< 370m$;
 - Velocità: $< 190Km/h$.

- per un drone di categoria 5:
 - Misura: maggiore;
 - Peso: $> 600kg$;
 - Altitudine operativa: $> 5Km$;
 - Velocità: qualsiasi.

Il controllo di un drone oggi avviene mediante componenti **hardware** e **software**:

- L'**hardware** è generalmente basato su microprocessore primario, su un processore secondario (failsafe), su attuatori, includendo anche l'uso di regolatori di velocità elettronici (ad esempio gli ESCs), e sull'uso di sensori basati su un certo numero di gradi di libertà (DOF): per sensori 6DOF ad esempio si usano giroscopi e accelerometri a 3 assi (ad esempio l'IMU); per sensori invece a 11DOF si usano anche barometro, bussola e navigatore GPS.
- Il **software** è generalmente chiamato **autopilota** o **stack di volo**, con lo scopo di eseguire una missione di volo in modo autonomo o tramite input da remoto. Essendo sistemi in tempo reale, gli UAV richiedono alta frequenza nella modifica dei dati sensoristici: per questo il software si basa generalmente su firmware (gestore del codice macchina, dell'esecuzione del processore e dell'accesso alla memoria), su middleware (gestore della comunicazione per il controllo di volo) e su un sistema operativo (ad esempio ROS, Nuttx, Linux).

1.2 I sistemi reali: il problema odierno

È sempre più complesso eseguire test di verifica e di validazione sui sistemi fisici reali integrati con unità di controllo (veicoli unmanned). Per essere competitivi nel mercato odierno, tali sistemi possono includere diversi gruppi sensoristici (IMU, georadar, termocamere) a seconda del tipo di sistema e del suo utilizzo, e ciò conduce a lavorare con grandi moli di dati: effettuare dunque test che prevedano il caricamento di un software su macchina reale per poi testarlo nel tempo e nello spazio può essere molto costoso. La soluzione pertanto deve fornire la possibilità di effettuare test completi senza utilizzare direttamente sul campo un prodotto finale assemblato. Una tecnica che può essere utilizzata a tal fine è detta Hardware-In-The-Loop, e costituisce la colonna portante del presente lavoro di tesi.

1.3 Hardware-In-The-Loop (HITL)

Con Hardware-In-The-Loop si indicano tutte quelle tecniche per la verifica del funzionamento di unità hardware connesse ad un banco in grado di simulare il comportamento del sistema reale. In questo modo, è possibile eseguire test di algoritmi di controllo già in fase di progettazione, senza attendere la disponibilità del prodotto finale, evitando i tempi ed i costi dei test fisici. L'approccio HITL permette di:

- Creare e simulare un'implementazione virtuale dei componenti fisici;
- Eseguire algoritmi di controllo sul simulatore che interagisce con il dispositivo fisico tramite i canali di I/O.

Risulta essere particolarmente utile quando testare l'algoritmo di controllo sul sistema fisico reale diventa costoso e/o pericoloso, ed è ampiamente utilizzata nei seguenti settori:

- Aerospazio e difesa: simulatore e controllo della dinamica di volo, dove è troppo complesso testare l'algoritmo di controllo su mezzo reale;

- Automotive: dinamica e controllo di un veicolo, dove non è pratico testare continuamente la funzionalità su strada;
- Automazione industriale: controllo di un impianto, quando si interrompe la produzione o la linea di assemblaggio per testare gli algoritmi di controllo, che può richiedere un'enorme quantità di tempo e può comportare delle perdite a livello economico.

1.4 Obiettivo della tesi

L'obiettivo della presente tesi è quello di implementare e testare in ambiente virtuale una legge di controllo per un drone multirottore mediante approccio HITL: il controllore, implementato su PixHawk in PX4, entrerà in comunicazione con un simulatore, tramite cui verrà definita ed implementata una nuova legge di controllo di volo. In questo modo sarà possibile testare tale legge ed analizzarne i dati in uscita, prima di effettuare dei test su un drone reale. La procedura completa si svolge mediante un'attività divisa in due parti: la prima destinata alla configurazione del controllore PixHawk e alla sua comunicazione con il calcolatore; la seconda destinata allo sviluppo di una legge di controllo del volo e all'analisi dei risultati.

Capitolo 2

Requisiti di progetto

In questo capitolo vengono definiti i concetti e gli strumenti base necessari alla configurazione del *setup* sperimentale, oggetto di tesi, e all'implementazione della legge di controllo su tale *setup*.

2.1 Architettura PX4

PX4 [3] è un software open source per il controllo di veicoli unmanned. Esso fornisce un set di vari strumenti agli sviluppatori, oltre ad un supporto software e hardware per le applicazioni. In Figura 2.1 viene rappresentata una panoramica di alto livello del controllore di volo implementato in PX4:

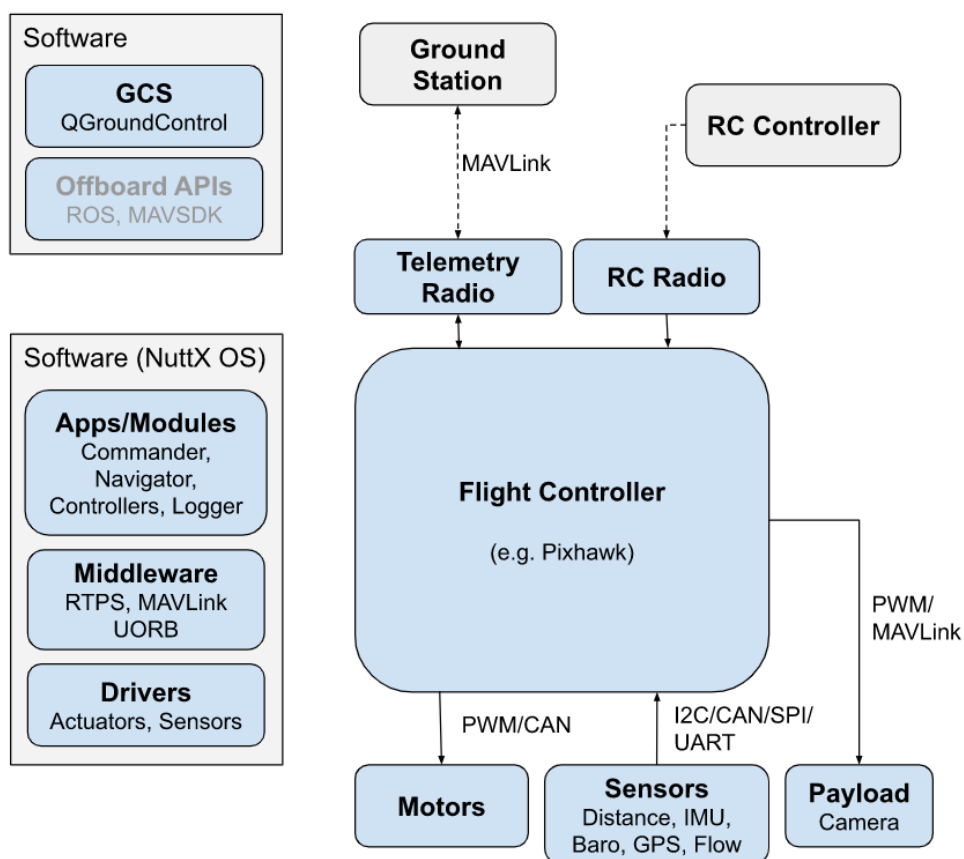


Figura 2.1: Architettura di alto livello del controllore di volo in PX4 [1].

Per il lato hardware abbiamo:

- **Controllore di volo (PixHawk):** esegue lo stack PX4;
- **ESC (Electronic Speed Control):** circuiti che regolano la velocità e la direzione di rotazione dei motori. Sono collegati alle uscite PWM del controller;
- **Sensori:** sono connessi al controller tramite bus I2C o UART;
- **Fotocamera:** viene collegata tramite canale PWM o protocollo MAVLink;
- **RC Controller:** occorre per il controllo manuale remoto (Joypad).

Per quanto riguarda il lato software invece:

- **QGroundControl:** usato come stazione di controllo da computer host;
- **Stack di volo PX4:** viene mandato in esecuzione sul controllore. Esso include moduli di comunicazione, drivers e middleware. In Figura 2.2 viene riportata la sua architettura di alto livello.

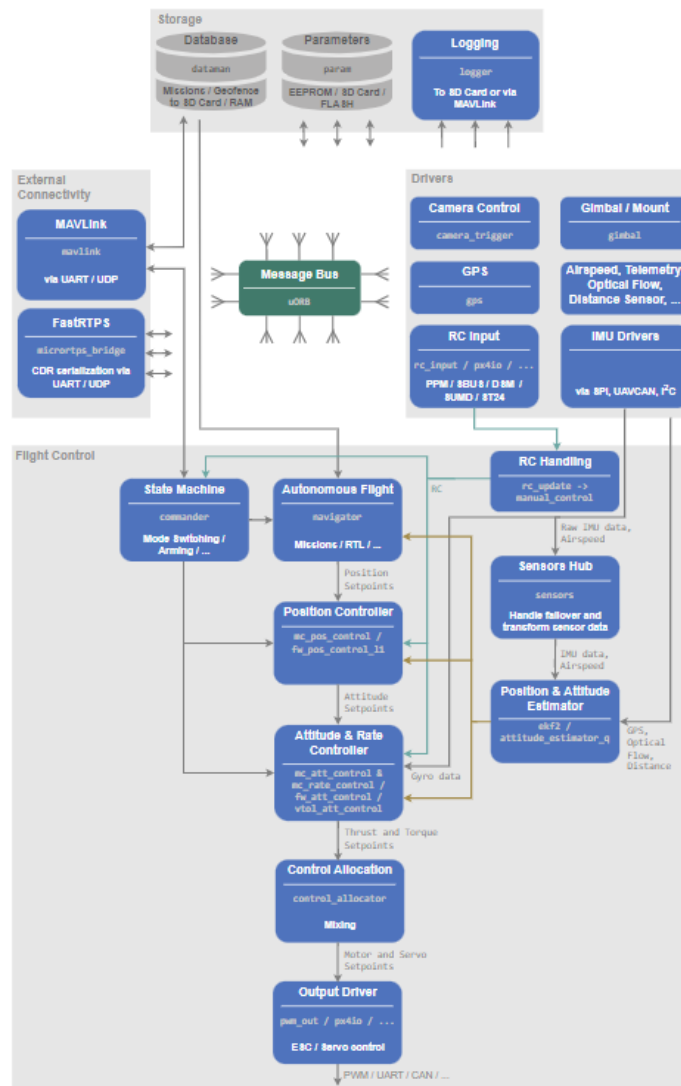


Figura 2.2: Architettura di alto livello dello stack di volo PX4 [8].

Più nel dettaglio, i moduli principali dello stack sono:

- **Commander**: contiene le informazioni sullo stato del drone e quelle relative alle modalità di volo, inclusa la failsafe;
- **Navigator**: Contiene i setpoint di volo, quali launch points, takeoff e waypoints, che sono i punti principali per costruire la traiettoria di volo;
- **Mc_controller**: è costituito da 2 moduli principali (position e attitude_and_rate controller) per il controllo del drone;
- **Mixer**: Rappresenta l'interfaccia tra i segnali di controllo generati dai moduli del mc_controller e i valori inviati agli attuatori del drone;
- **Sensors Hub**: è il cuore dei segnali misurati dai sensori: i dati in output di basso livello vengono resi disponibili grazie ai drivers e vengono messi a disposizione per il controller;
- **Ekf2 (Extended Kalman Filter)**: effettua la stima di posizione e degli angoli tramite un filtro di Kalman esteso;
- **Drivers**: costituiscono l'interfaccia tra i sensori fisici e l'architettura del firmware;
- **Mavlink**: implementa il MAVLink, ovvero il middleware di comunicazione, ed invia messaggi specifici (e. assetto degli angoli del drone) tramite il bus uORB;
- **Simulatore**: è il simulatore del drone in 3D che si interfaccia con il firmware tramite protocollo MAVLink. Il progetto si basa sul simulatore jMAVSim.

Le ragioni principali per cui gli sviluppatori apprezzano PX4 sono:

- **Architettura modulare**: PX4 è altamente modulare, sia a livello hardware sia a livello software. Utilizza un'architettura basata su porte: ciò permette agli sviluppatori di aggiungere delle componenti supplementari;
- **Open Source**: PX4 è sviluppato in collaborazione con una comunità globale. Il flight-stack è un toolkit generale, ed è ampiamente utilizzato in aziende e laboratori;
- **Configurabilità**: PX4 offre API e SDK ottimizzati per gli sviluppatori. Tutti i moduli sono autonomi e le funzionalità sono facili da implementare e riconfigurare;
- **Validità**: migliaia di sistemi commerciali basati su PX4 sono stati implementati in tutto il mondo. Parallelamente, un team di test di volo dedicato accumula migliaia di ore di volo ogni mese eseguendo test hardware e software per garantire la sicurezza e l'affidabilità del codice;
- **Funzioni di sicurezza**: sono ottime le funzionalità di sicurezza, tra cui il comportamento automatico di sicurezza, il supporto per diverse modalità di ritorno, paracadute, ecc.. sono già incluse per impostazione predefinita nella base di codice. Le funzionalità sono facilmente configurabili e regolabili per sistemi personalizzati;
- **Interoperabilità**: oltre ad essere un solido flight stack, PX4 offre un ecosistema di dispositivi supportati. Il progetto guida anche lo sforzo di standardizzazione per il progresso delle comunicazioni, l'integrazione delle periferiche e le soluzioni di gestione dell'alimentazione.

2.2 Software di progetto

Per il lato software si va a lavorare su sistema operativo **Microsoft Windows 10 Pro** (versione 10.0.19044). Inoltre, vengono utilizzati:

- **QGroundControl (versione 4.2.3)**: Fornisce il supporto per la configurazione dei veicoli unmanned pilotati con PX4, ed è di fatto la stazione per la pianificazione e l'esecuzione del percorso che il veicolo ad esso collegato deve eseguire. In Figura 2.3 è riportata la home principale del software.

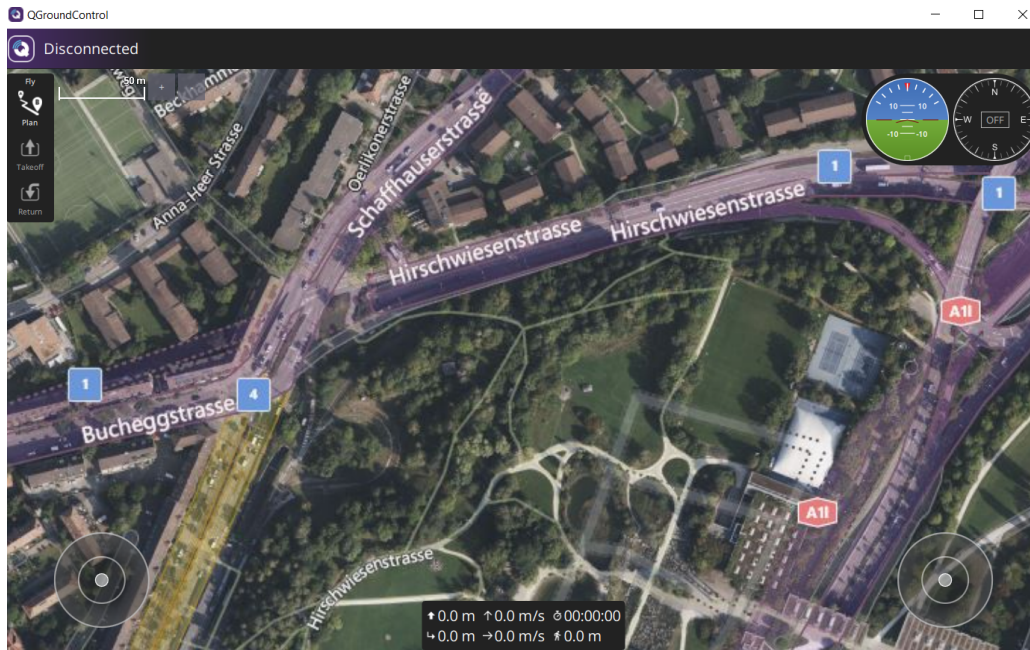


Figura 2.3: Home del software QGroundControl (4.2.3)

Da qui è possibile accedere a quattro sezioni fondamentali:

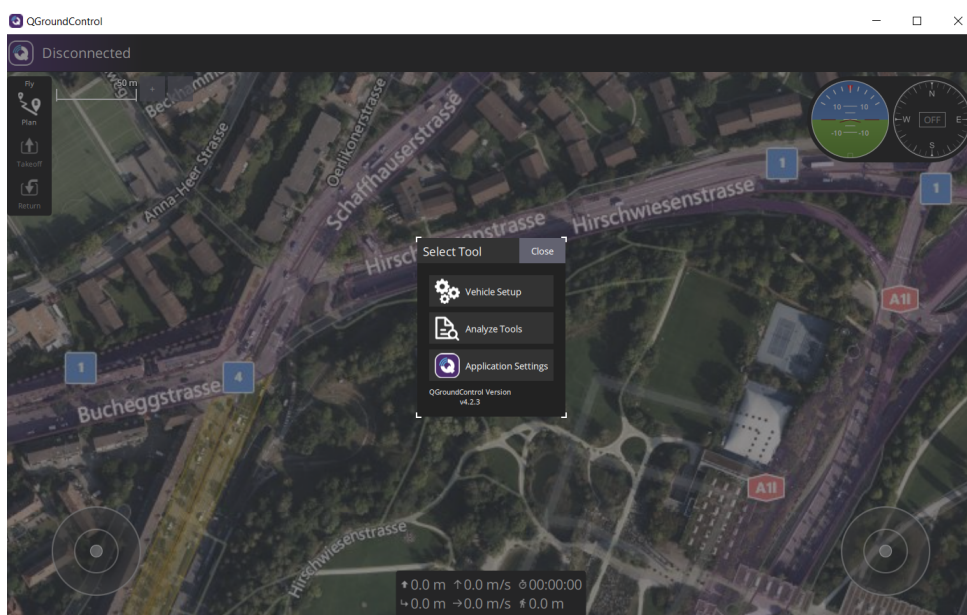


Figura 2.4: Sezioni principali del software QGroundControl.

- **Fly Plan:** permette di pianificare il volo (missione) di un veicolo unmanned. Il volo si basa sulla definizione del punto di **Launch**, cioè il punto da cui il veicolo prende quota prima di iniziare il volo, dei **Takeoff**, ovvero i punti sulla mappa da cui il veicolo deve iniziare la sua missione di volo, e dei **Waypoints**, ovvero i punti che definiscono una variazione della traiettoria durante la missione di volo. Essi guidano passo dopo passo il percorso che il veicolo deve seguire, dallo stato di Takeoff a quello finale;
 - **Vehicle Setup:** si tratta della sezione dedicata all’upload del Firmware, al settaggio dei parametri per la missione e alla definizione dell’airframe del veicolo, in base a ciò che occorre all’utente;
 - **Analyze Tools:** è la sezione dedicata al download dei file di volo .tlog e al MAVLink Inspector, ovvero la finestra in cui sono leggibili i messaggi MAVLink (dati sul controllore, dati dai sensori, ecc.);
 - **Application Settings:** è la sezione dedicata alle impostazioni per la comunicazione con il controllore fisico, per le unità di misura delle grandezze fisiche misurate e per i file di volo: è possibile da qui scegliere se il download dei file può avvenire automaticamente e se averli anche in formato csv.
- **PX4.Windows.Cygwin.Toolchain.0.8.msi:** è l’unico software ufficialmente supportato per il download del codice sorgente PX4 su Windows;
 - **Matlab (versione R2022b):** l’ambiente di calcolo fornito da MathWorks su cui verrà costruito l’algoritmo di controllo e su cui verrà effettuata l’analisi dei risultati. Deve essere integrato con i seguenti toolbox:
 - **Simulink:** è il tool per la modellazione, simulazione dei sistemi dinamici: qui sarà sviluppata la legge di controllo della quale sarà eseguita la *build* sul controllore PixHawk;
 - **UAV Toolbox Support Package for PX4 Autopilots:** è il pacchetto che permette di accedere alle periferiche PX4 Autopilot direttamente da Matlab/Simulink. Tuttavia, questo è possibile con l’integrazione di altri pacchetti, necessari per la generazione automatica di codice C per sistemi embedded e per l’integrazione di algoritmi. Essi sono riportati di seguito, in Figura 2.5.



















Name	Author
 Aerospace Blockset version 5.3	 MathWorks
 Aerospace Toolbox version 4.3	 MathWorks
 Embedded Coder version 7.9	 MathWorks
 MATLAB Coder version 5.5	 MathWorks
 Navigation Toolbox version 2.3	 MathWorks
 Simulink version 10.6	 MathWorks
 Simulink Coder version 9.8	 MathWorks
 UAV Toolbox version 1.4	 MathWorks
 UAV Toolbox Support Package for PX4 Autopilots version 22.2.0	 MathWorks

Figura 2.5: Tools necessari all'implementazione.

- **jMAVSim**: è il simulatore della realtà utilizzato nel progetto per testare ed osservare il volo, e supporta la simulazione di veicoli aerei a 4 rotori in PX4. Viene lanciato dal firmware automaticamente durante il *run* del controllore eseguito in Simulink, e l'interfaccia è semplice ed intuitiva da utilizzare e configurare. In Figura 2.6 è riportata la finestra principale del simulatore durante il *landing* del drone simulato:



Figura 2.6: Finestra principale del simulatore JMAVSim - *landing*.

2.3 Hardware di progetto

Per il lato hardware vengono utilizzati:

- **PixHawk 2.1 Cube (Black)**: è il controllore su cui si va implementata la legge di controllo. In Figura 2.7 è riportato il suo design (**FMUv3**). In Figura 2.8 sono riportate le porte per il cavo micro-USB e per l'SD Card: il micro-USB è necessario per il collegamento

con il pc e per l'upload del firmware, mentre l'SD Card viene utilizzata per memorizzare file di configurazione, prove di volo e informazioni sulle missioni.



Figura 2.7: PixHawk 2.1 Cube (Black) [4].

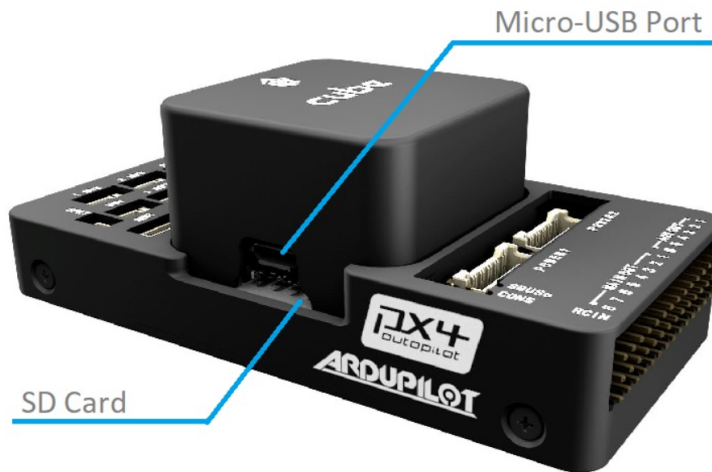


Figura 2.8: Porte per micro-USB e SD card su PixHawk 2.1 Cube (Black) [6].

Le caratteristiche sono riportate in Tabella 2.1: vengono evidenziati processori, RAM, sensori e le interfacce principali.

Processori	Core ARM Cortex M32 (4 bit) con FPU, coprocessore failsafe (32 bit).
RAM	256KB.
Sensori	3 IMU ridondanti IvenSense (accelerometri, giroscopi, bussola), 2 barometri MS5611 ridonanti.
Interfacce	14x uscite servo PWM, uscita servo S.Bus, 5 porte seriali (2 TELEM, 2 GPS, 1 USB, 1 Serial 5) 2 porte I2C.

Tabella 2.1: Caratteristiche principali PixHawk 2.1 Cube (Black).

- **Convertitore seriale CP2102/9 (from USB to TTL):** serve in fase di configurazione e di collegamento tra il controllore e l'host pc. Tramite il convertitore è possibile eseguire il deploy dell'algoritmo di controllo dal calcolatore e di osservare il modello Simulink in esecuzione sull'hardware in destinazione. In Figura 2.9 un'illustrazione del convertitore:



Figura 2.9: Convertitore seriale CP2102/9 [17].

Le caratteristiche principali sono le seguenti:

- Gli input e gli output analogici funzionano a 3.3V. Può essere alimentato a 5V o a 3.3V;
 - Ha bisogno di un driver reperibile da un URL della documentazione ufficiale per interfacciarsi con l'host computer a cui è collegato: dopo l'installazione, il convertitore sarà visibile nella lista delle porte COM, all'interno della sezione "Gestione dispositivi" del pc.
- **Connettore JST:** un connettore da 6 pin a due terminali (femmine) disponibile in laboratorio e che permette di collegare la PixHawk con il convertitore a partire da una porta opportuna (illustrata nel capitolo 4): un terminale verrà inserito nella porta della PixHawk destinata alla comunicazione con l'host, mentre l'altro è costituito da 6 fili liberi, di cui 4 sono necessari per il collegamento con il convertitore: uno rosso per l'alimentazione (VCC), uno blu per il collegamento a massa (GND), due gialli per la ricezione (RXD) e la trasmissione (TXD). In Figura 2.10 è riportato quello utilizzato per l'esperimento.

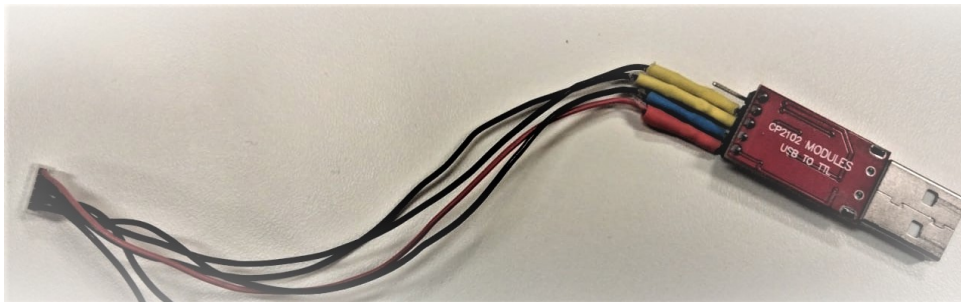


Figura 2.10: JST Connector (6 pin).

Capitolo 3

Configurazione del sistema HITL

Durante la simulazione tramite tecnica HITL, sia QGroundControl (QGC) sia il modello del sistema sviluppato in Simulink devono comunicare con l'hardware Pixhawk tramite seriale. Tuttavia, solo una di queste applicazioni può mantenere occupata la porta seriale e comunicare con l'hardware PX4 durante la simulazione. Pertanto, una terza componente software, che costituisce il simulatore in 3D del modello, è necessaria per mettere in comunicazione le due applicazioni: il simulatore (jMAVSim nel nostro caso) fa da ponte per condividere i dati MAVLink tra l'impianto Simulink e QGC. In Figura 3.1 e 3.2 sono rappresentati i workflow per l'HITL in PX4:

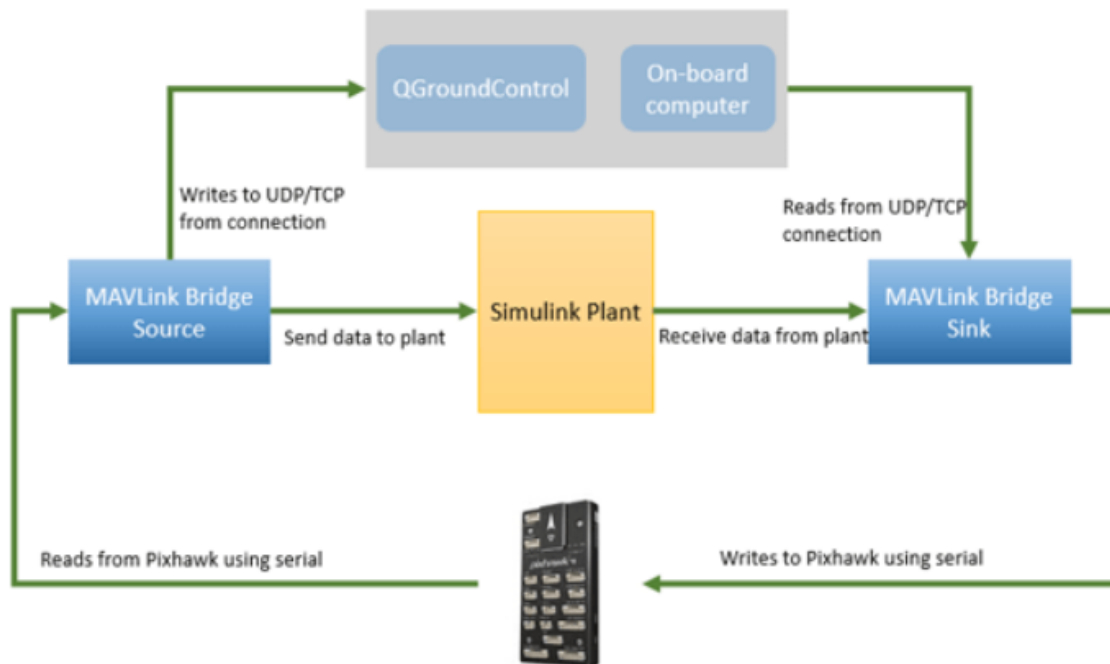


Figura 3.1: Workflow per l'HITL in PX4 [9].

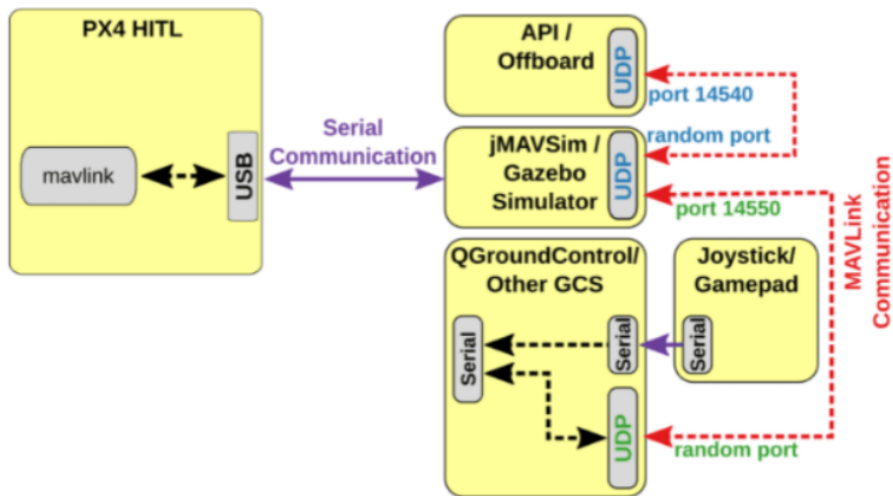


Figura 3.2: Workflow per l'HITL in PX4 [10].

Di seguito verranno illustrati i passaggi per arrivare a questo schema di comunicazione ed entrerà in gioco anche il simulatore della realtà virtuale jMAVSim.

3.1 Configurazione PixHawk tramite QGroundControl

È necessario, come prima cosa, fare l'upload del firmware opportuno tramite QGroundControl: la versione di Matlab 2022b è compatibile con le releases del firmware fino ad arrivare alla v.1.12.3, reperibile in rete [7]. Per questo progetto, è stato eseguito il download della release v1.12.3 stabile denominata `px4_fm-v3_default.px4`, ed è stata caricata tramite il software:

- Si apre QGC, e dalla sezione **Vehicle Setup** si va su **Firmware**;
- Si collega la board PixHawk al pc tramite micro-USB;
- Dalla colonna di destra per eseguire l'upload, si procede con il caricamento del firmware personalizzato: si seleziona dal pc locale la release del firmware ottenuta e si aspetta il completamento dell'upload.

Da questo momento, è possibile configurare la board per l'HITL, quindi:

- Dalla sezione **Safety** si abilita la modalità HITL;
- Dalla sezione **Airframe** si seleziona e si esegue il veicolo **HIL Quadcopter X**;
- Dalla sezione **Flight Modes** si impostano 6 canali per le modalità di volo (la scelta è lasciata all'utente).

Ora, si accede all'area **Application Settings**. Nella sezione **Autoconnect to the following devices** si tolgono tutte le spunte eccetto **UDP** (in Figura 3.3).

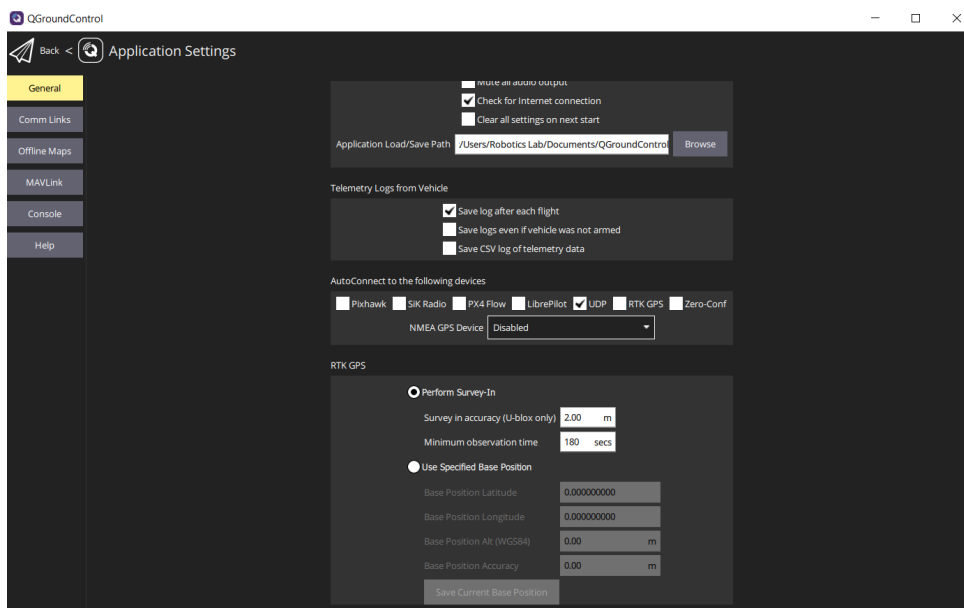


Figura 3.3: Application Settings - porta UDP.

Questa porta interrompe la comunicazione tra QGC e la PixHawk tramite micro-USB, possibile solo grazie al simulatore "ponte", come detto in precedenza¹. Successivamente, vanno configurate le seguenti impostazioni [2]:

- Si accede alla sezione **Vehicle Setup** e si va nell'area **Parameters** in fonda alla sidebar;
- Si digitano nella barra di ricerca, uno per volta, i seguenti parametri²:
 - **COM_RC_IN_MODE**, che va impostato su **Joystick/No RC Checks**;
 - **COM_DISARM_LAND**, che va settato a -1 per evitare che un potenziale failsafe si inneschi durante il *landing* del drone, ovvero quando il drone rimane fisso in un punto preciso dello spazio;
 - **NAV_ACC_RAD**, che va settato a 10. Questo parametro definisce il raggio di accettazione entro cui si considera che il veicolo abbia raggiunto il *waypoint*. Il valore di questo parametro può essere aumentato per rendere il volo più fluido e per permettere al drone di continuare il volo anche se non dovesse raggiungere perfettamente il *waypoint*.

Ora la sidebar principale nella sezione **Vehicle Setup** (in Figura 3.4) non deve riportare elementi marchiati in rosso, eccetto quello relativo alla potenza: se quella sezione è in rosso, non rappresenta un problema per la simulazione.

¹È possibile tornare a comunicare tramite micro-USB spuntando anche solo l'opzione **PixHawk**.

²Il setting è stato effettuato seguendo le istruzioni sulla documentazione ufficiale MathWorks relativa all'HITL.

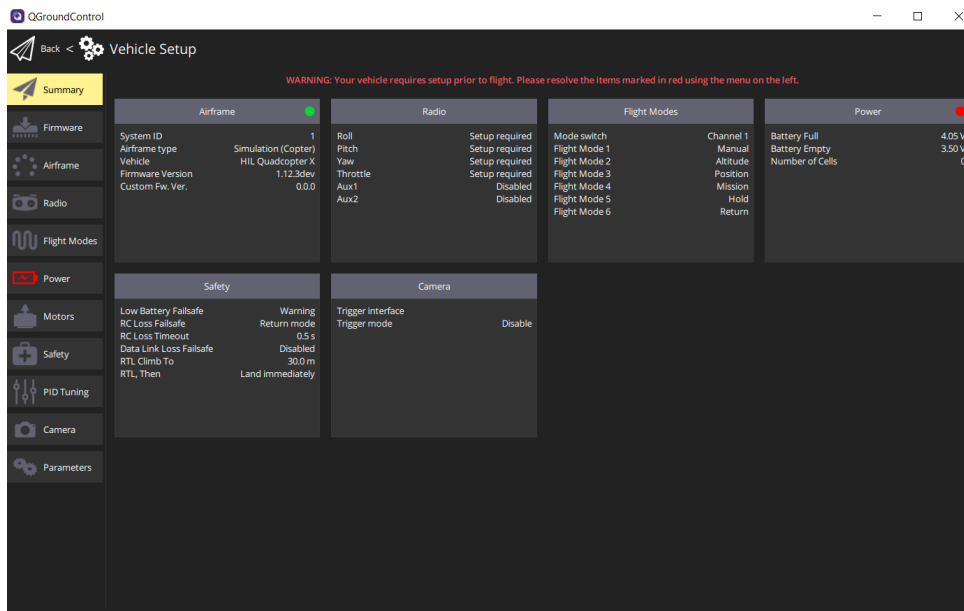


Figura 3.4: Vehicle Setup al termine della configurazione.

3.2 Hardware setup tramite UAV Toolbox Support Package for PX4 Autopilots

Dopo le impostazioni effettuate tramite QGC, si passa al software Matlab. Dalla home principale, tramite il tasto **Manage Add-Ons**, è possibile configurare il pacchetto UAV per PX4 Autopilots [2]:

- si va su **Hardware Setup** e si procede con l'installazione di Toolchain, seguendo le istruzioni: è importante, durante la procedura, mettere la spunta su **Clone PX4 repository and Start Simulation** che permetterà la creazione della cartella "Firmware" all'interno della cartella su cui viene installato Toolchain. La cartella "Firmware" contiene anche i codici relativi al tool jMAVSim (risiedente nel firmware stesso) che saranno importanti per avere informazioni sul modello simulato, sul sistema di riferimento utilizzato in ambiente virtuale, sui parametri del drone, ecc;
- si segue la procedura tenendo spuntate le opzioni di default fino alla sezione per eseguire il *build* del firmware in ambiente Matlab/Simulink;
- una volta completata la building, prima di procedere con **Next**, ci si assicura che la PixHawk sia scollegata dal pc: si va in "Gestione dispositivi" di Windows, si ricollega la PixHawk e si fa attenzione alla porta COM visualizzata. La versione utilizzata in laboratorio visualizza COM4 per il bootloader della PixHawk e, dopo 4 secondi, visualizza COM5 per la PixHawk pronta all'uso. Una volta avuto queste informazioni, si torna su Matlab e si scollega di nuovo la PixHawk. Si va su **Next**, quindi si seleziona manualmente l'**Host Serial Port for upload** (COM4 in questo caso), si ricollega la PixHawk ed entro 4 secondi si clicca su **Upload Firmware**. Al termine della procedura, è possibile accedere ad esempi già implementati in Simulink per l'uso dell'HITL.

Una volta completata questa operazione, dalla console comandi di Matlab, si digita il comando `open_system('px4demo_QGCWaypointFollower_hitl')`: viene visualizzato in Simulink lo schema di controllo nativo collegato al navigatore in QGC, riportato di seguito in Figura 3.5.

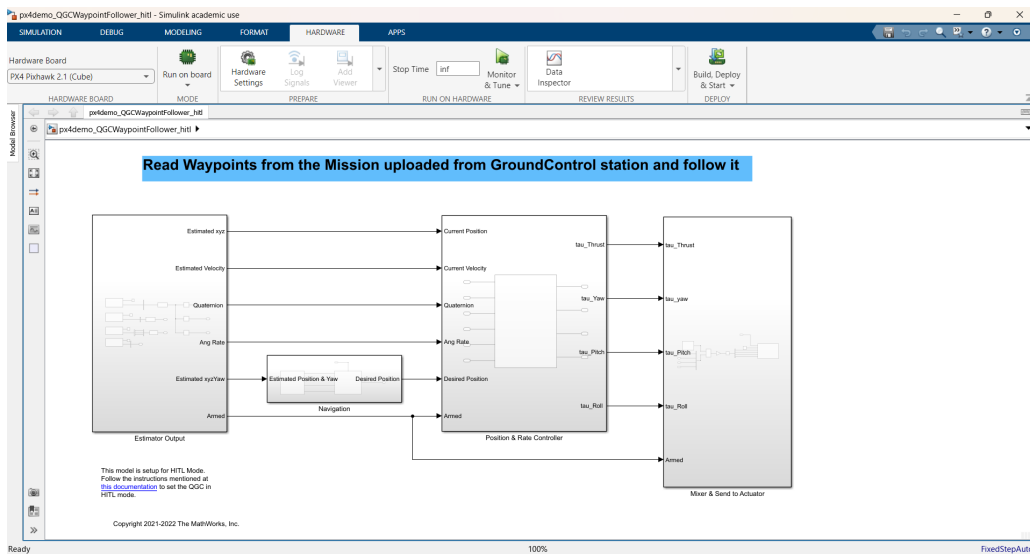


Figura 3.5: Modello Simulink del controllo nativo in PX4.

Di fatto, il comando lanciato dalla console serve ad aprire un modello per la navigazione a waypoints: tale modello verrà utilizzato come punto di partenza per testare il *setup* sperimentale, in particolare verrà studiato il controllo *built-in*, ovvero quello nativo già presente nel controllore, e verrà poi sostituito con una legge di controllo *ad-hoc* basata sul modello matematico del drone in esame. Una volta fatto l'accesso al modello in Simulink, è necessario fare delle impostazioni:

- Si va nella sezione **HARDWARE**, quindi in **Hardware Settings**;
- Si seleziona la board in uso (PX4 PixHawk 2.1 (Cube) in questo caso);
- Dalla sezione **Target hardware resources**:
 - Da **Build options** ci si assicura che il CMake configuration sia impostato su `px4_fmu-v3_default`, e si seleziona la COM opportuna per l'upload del firmware;
 - Da **HITL** si abilita la modalità HITL e si imposta jMAVSim come simulatore virtuale che, insieme a Gazebo (non opzionabile in Simulink), è il più utilizzato come ambiente virtuale;
 - Da **External mode** si impostano l'**Hardware serial port** e l'**Host serial port**: questi parametri indicano, rispettivamente, la porta scelta sulla PixHawk per la connessione seriale con l'host e la COM associata al convertitore.

La COM è visualizzabile una volta collegato il convertitore CP2102/9, mentre la porta sulla PixHawk scelta è la GPS2 (6 pins). Nelle seguenti tabelle 3.1 e 3.2 sono riportate le informazioni per le porte e per il collegamento con il convertitore: un terminale da 6 fili (uno per pin) va inserito nella porta GPS2, mentre solo 4 fili (femmine) sono necessarie per il convertitore (Figura 2.10):

Labels	UART/USART port number
USB	/dev/tty/ACM0
TELEM 1	/dev/ttyS1
TELEM 2	/dev/ttyS2
GPS 1	/dev/ttyS3
GPS 2	/dev/ttyS6
Serial 5 (NSH port)	/dev/ttyS5

Tabella 3.1: Porte seriali PixHawk 2.1 Cube (Black)

Pin	Signal	Volt
1 (red)	VCC	+5V
2 (blk)	TX (OUT)	+3.3V
3 (blk)	RX (IN)	+3.3V
4 (blk)	SCL I2C2	+3.3V
5 (blk)	SDA I2C2	+3.3V
6 (blk)	GND	GND

Tabella 3.2: Pins della porta GPS 2 di PixHawk 2.1 Cube (Black)

Nel nostro caso, dunque, l'Hardware board serial port è la `/dev/ttyS6`, mentre l'Host serial port, associata alla COM del convertitore, è la COM6;

- Da **MAVLink** ci si assicura che ci sia la spunta su **Enable MAVLink**;
- Da `/dev/ttyS6` si imposta il baud rate a 115200.

Ora il modello è pronto per il *building* (che verrà ripreso nel Capitolo 6).

Capitolo 4

Il modello del quadrirotore

Il quadrirotore è un UAV che viene solitamente progettato secondo due configurazioni, definite a + oppure ad **X**: questo progetto in PX4 si basa sull'Airframe con configurazione ad X (Figura 4.1 [23]), ed è progettato su un sistema di riferimento globale (**Earth Frame xyz**) NED (North-East-Down).

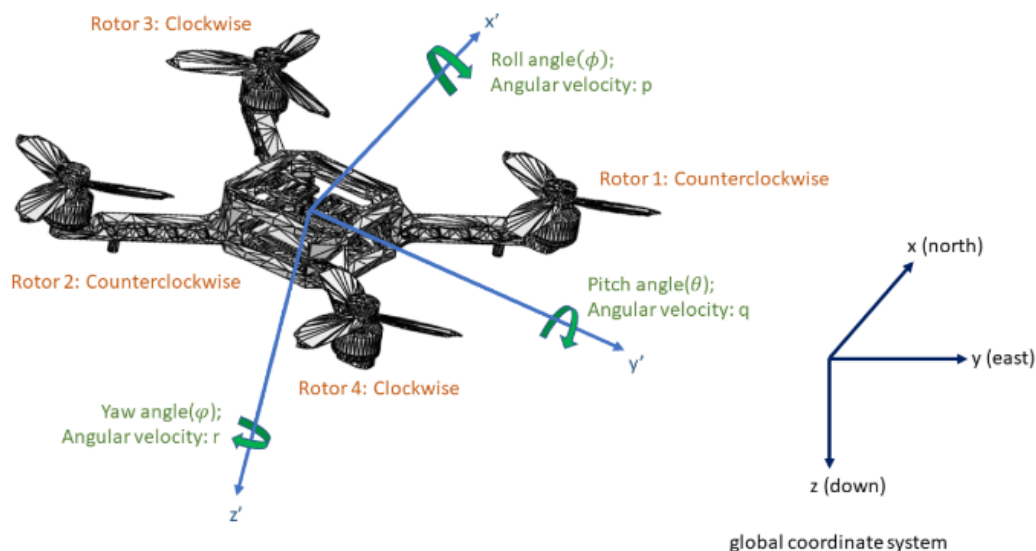


Figura 4.1: Modello Quadrirotore a X in PX4.

Questa configurazione può essere verificata in documentazione PX4 oppure si esegue una prova di volo in QGC tramite il modello in Simulink e si osservano i dati sul **MAVLink Inspector** in **Analyze Tools**: le misure leggibili nelle sezioni **NED**, **ATTITUDE** e **ALTITUDE** durante un volo generico confermano che il drone simulato rispetta tale sistema di riferimento. Inoltre:

- il drone su jMAVSim segue il percorso impostato su QGC, e il sistema NED in ambiente virtuale è confermato nel file **Environment.java** nella Firmware folder scaricata in precedenza;
- la disposizione dei rotori e le loro direzioni (Clockwise e Counterclockwise) è dettata dalla documentazione ufficiale PX4: il modello del quadrirotore ad X dell'Airframe **Quadrotor X** ha i rotori nella configurazione di Figura 4.1.

I 4 motori disposti sulle bisettrici del piano $x'y'$ del **Body Frame $x'y'z'$** (il sistema di riferimento il cui origine coincide con il centro geometrico del drone) devono ruotare in opportune

direzioni, sia per stabilizzare il drone su una certa quota (asse z), sia per bilanciare le coppie che mettono in rotazione il drone stesso: parliamo dunque di 4 ingressi di controllo e 6 gradi di libertà. Una volta stabilito il sistema di riferimento e il verso di rotazione dei motori, prima di procedere al modello matematico, bisogna stabilire l'orientamento del body frame rispetto all'earth frame utilizzato in PX4: la sequenza delle rotazioni stabilisce la matrice di rotazione R molto importante ai fini del modello, e deve essere univoca. All'interno della cartella **matrix** della cartella "Firmware", il file **Euler.hpp** illustra la sequenza delle rotazioni (coincidente con quella classica) seguite in PX4:

```

4  * All rotations and axis systems follow the right-hand rule
5  *
6  * An instance of this class defines a rotation from coordinate frame 1 to coordinate frame 2.
7  * It follows the convention of a 3-2-1 intrinsic Tait-Bryan rotation sequence.
8  * In order to go from frame 1 to frame 2 we apply the following rotations consecutively.
9  * 1) We rotate about our initial Z axis by an angle of _psi.
10 * 2) We rotate about the newly created Y' axis by an angle of _theta.
11 * 3) We rotate about the newly created X'' axis by an angle of _phi.
12 *
13 * @author James Goppert <james.goppert@gmail.com>
14 */
15

```

Figura 4.2: Sequenza delle rotazioni in PX4.

- si prende l'Earth Frame xyz e si fa ruotare l'asse z di un angolo ψ (Figura 4.3) secondo la matrice di rotazione:

$$R(z, \psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

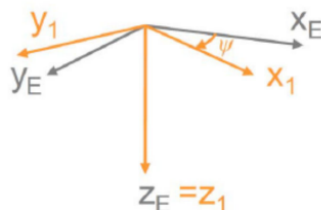


Figura 4.3: Rotazione attorno all'asse z di un angolo ψ [14].

- si prende il nuovo sistema ruotato e si fa ruotare il nuovo asse y di un angolo θ (Figura 4.4) secondo la matrice di rotazione:

$$R(y, \theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

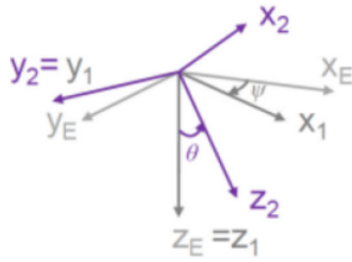


Figura 4.4: Rotazione attorno all'asse y_1 di un angolo θ [14].

- si prende il nuovo sistema ruotato e si fa ruotare il nuovo asse x di un angolo φ (Figura 4.5) secondo la matrice di rotazione:

$$R(x, \varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix}$$

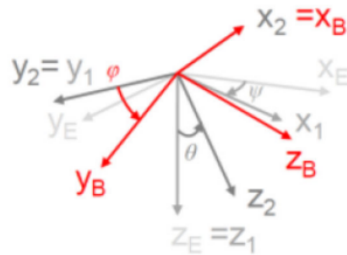


Figura 4.5: Rotazione attorno all'asse x_2 di un angolo φ [14].

Allora la sequenza di rotazione definisce una matrice di rotazione finale dal **Body Frame** all'**Earth Frame**¹:

$$R = R(z, \psi)R(y, \theta)R(x, \varphi) = \begin{bmatrix} c(\theta)c(\varphi) & -c(\theta)s(\psi) + c(\psi)s(\theta)s(\varphi) & s(\psi)s(\varphi) + c(\psi)c(\varphi)s(\theta) \\ c(\theta)s(\varphi) & c(\psi)c(\varphi) + s(\theta)s(\varphi)s(\psi) & -s(\varphi)c(\psi) + c(\varphi)s(\theta)s(\psi) \\ -s(\theta) & c(\theta)s(\varphi) & c(\theta)c(\varphi) \end{bmatrix}$$

dove $c = \cos$ ed $s = \sin$. Inoltre, specifichiamo che:

- $-\pi < \varphi < \pi$ e $-\pi < \psi < \pi$: gli angoli φ (roll) e ψ (yaw) coprono 2π radianti. L'intervallo così definito è per indicare che per valori maggiori o uguali a 0 l'angolo ruota in senso antiorario, seguendo la regola della mano destra, viceversa per quelli minori di 0;
- $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$: l'angolo θ (pitch) copre invece π radianti, e l'intervallo viene così definito per le stesse ragioni viste per gli altri due angoli;

4.1 Equazioni del moto

Il modello matematico è non lineare ed è determinato tramite un approccio Newton-Euleriano. Esso viene descritto dalle seguenti equazioni:

¹Verificabile nel file **Dcm.hpp** all'interno della cartella "Firmware".

- **Traslazione:** si scrivono le equazioni della dinamica relative allo spostamento del centro di massa del drone nello spazio euclideo rispetto all'Earth Frame. La dinamica è influenzata dalle forze esterne che agiscono sul corpo (in questo caso parliamo delle forze di spinta dovute ai rotori (*thrust forces*) e dalle forze resistenti dovute all'attrito con l'aria. Di seguito, le equazioni scritte in forma matriciale

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ -u_1 \end{bmatrix} - k_1 I \begin{bmatrix} V_x |V_x| \\ V_y |V_y| \\ V_z |V_z| \end{bmatrix}$$

dove m è la massa del corpo, $g = 9.81[m/s^2]$, u_1 è la prima componente del vettore di controllo virtuale, I è la matrice identità definita come:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

, k_1 è il coefficiente di attrito lineare, mentre V_x , V_y e V_z sono rispettivamente le velocità del centro di massa del drone lungo gli assi x , y e z dell'Earth Frame. In forma compatta

$$m\dot{V} = P + RF_B - F_A$$

dove \dot{V} è il vettore delle accelerazioni, ovvero la derivata del vettore di velocità nell'Earth Frame):

$$V = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

P è il vettore della forza peso (g ha il segno positivo in quanto siamo su sistema di riferimento NED); R è la matrice di rotazione definita in precedenza; F_B è il vettore delle Thrust forces (dirette lungo l'asse z : il segno di u_1 significa che la spinta avviene verso l'alto in un sistema NED) ed F_A è la forza di resistenza aerodinamica.

- **Rotazione:** si scrivono le equazioni della dinamica relative alla rotazione del drone nello spazio euclideo, influenzate dalle coppie esterne che agiscono sul corpo (parliamo delle coppie dovute ai rotori per stabilizzare il drone durante il volo) e dalle coppie resistenti dovute all'attrito con l'aria. Di seguito, le equazioni scritte in forma matriciale

$$\begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}lu_2 \\ \frac{\sqrt{2}}{2}lu_3 \\ u_4 \end{bmatrix} - k_2 I \begin{bmatrix} p|p| \\ q|q| \\ r|r| \end{bmatrix}$$

dove J_x , J_y e J_z sono rispettivamente i momenti di inerzia calcolati rispetto agli assi x' , y' e z' del Body Frame; p, q ed r sono rispettivamente le velocità angolari rispetto agli assi x' , y' e z' di Figura 4.1; u_2 , u_3 e u_4 sono le 3 componenti rimanenti del vettore di controllo virtuale²; l è la lunghezza del braccio del drone, ovvero della distanza tra il centro di massa del drone e uno dei suoi rotori, mentre k_2 è il coefficiente di attrito angolare. In forma compatta

$$J\dot{\omega} + \omega \times J\omega = M_B - M_A$$

dove $J = \text{diag}(J_x, J_y, J_z)$ è la matrice diagonale di inerzia (la terna principale di inerzia è

²Nel Capitolo 6, i segnali virtuali $u_i (i = 1...4)$ verranno mappati in maniera opportuna.

giustificata dal fatto che il corpo è simmetrico rispetto agli assi passanti per il suo centro di massa); il vettore di velocità angolare del drone nel Body Frame:

$$\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$\dot{\omega}$ è il vettore delle derivate delle componenti di ω ; M_B è il vettore delle coppie esterne dovuto ai motori (determinate configurazioni dei rotori stabilizzano le coppie attorno a φ , θ e ψ e che dipendono dalla lunghezza l dei bracci del drone), mentre M_A è il vettore delle coppie resistive dovute all'attrito con l'aria.

- **Cinematica:** Si scrivono le equazioni della cinematica che legano le velocità angolari nel Body Frame con quelle nell'Earth Frame. In forma matriciale:

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & t(\theta)s(\varphi) & t(\theta)c(\varphi) \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & \frac{s(\varphi)}{c(\theta)} & \frac{c(\varphi)}{c(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

con $t = \tan$, $s = \sin$ e $c = \cos$. In forma compatta

$$\eta = T\omega$$

dove η è il vettore delle velocità angolari nell'Earth Frame e T è la matrice di trasformazione cinematica.

4.2 Parametri e caratteristiche del modello simulato

Il drone caricato in ambiente virtuale è caratterizzato da parametri specifici necessari per il calcolatore che deve generare le forze e le coppie opportune per il controllo. Per ottenere questi dati, può essere aperta una sidebar (mediante il tasto F1 della tastiera) all'interno del simulatore JMAVSim riportante dati e parametri settati, oppure si possono perlustrare due files nella Firmware folder:

- **Quadcopter.java:** Qui viene definito il costrutto generico di un Quadcopter mediante la classe Quadcopter. Al suo interno, all'attributo Quadcopter vengono passati i parametri riportati in Figura 4.6:

```

17  /**
18   * Generic quadcopter constructor.
19   *
20   * @param world      world where to place the vehicle
21   * @param modelName  filename of model to load, in .obj format
22   * @param orientation "x" or "+"
23   * @param style      rotor position layout style. "default"/"px4" for px4, or "cw_fr" CW sequential layout starting at front motor
24   * @param armLength  length of arm from center [m]
25   * @param rotorThrust full thrust of one rotor [N]
26   * @param rotorTorque torque at full thrust of one rotor in [Nm]
27   * @param rotorTimeConst spin-up time of rotor [s]
28   * @param rotorsOffset rotors positions offset from gravity center
29   * @param showGui     false if the GUI has been disabled
30   */
31  public Quadcopter(World world, String modelName, String orientation, String style,
32                   double armLength, double rotorThrust, double rotorTorque,
33                   double rotorTimeConst, Vector3d rotorsOffset, boolean showGui) {
34      super(world, modelName, showGui);

```

Figura 4.6: Parametri passati alla classe Quadcopter.

- **Simulator.java**: Qui vengono inizializzati i parametri dell'oggetto Quadcopter che servono per costruire il modello ed il controllore in fase di implementazione, in particolare nel metodo `buildMulticopter()`:

```

438     private AbstractMulticopter buildMulticopter() {
439         Vector3d gc = new Vector3d(0.0, 0.0, 0.0); // gravity center
440         AbstractMulticopter vehicle = new Quadcopter(world, DEFAULT_VEHICLE_MODEL, "x", "default",
441                                                     0.33 / 2, 4.0, 0.05, 0.005, gc, SHOW_GUI);
442         Matrix3d I = new Matrix3d();
443         // Moments of inertia
444         I.m00 = 0.005; // X
445         I.m11 = 0.005; // Y
446         I.m22 = 0.009; // Z
447         vehicle.setMomentOfInertia(I);
448         vehicle.setMass(0.8);
449         vehicle.setDragMove(0.01);
450         SimpleSensors sensors = new SimpleSensors();
451         sensors.setGPSInterval(50);
452         sensors.setGPSDelay(200);
453         sensors.setNoise_Acc(0.05f);
454         sensors.setNoise_Gyo(0.01f);
455         sensors.setNoise_Mag(0.005f);
456         sensors.setNoise_Prs(0.1f);
457         vehicle.setSensors(sensors, getSimMillis());
458         //v.setDragRotate(0.1);
459
460         return vehicle;
461     }

```

Figura 4.7: Definizione dei parametri passati all'oggetto Quadcopter.

Di seguito, in Tabella 4.1, sono riportati i parametri utilizzati nella legge di controllo: le quantità T_{max} e Q_{max} verranno introdotte e spiegate nel Capitolo 5.

Descrizione	Parametro	Valore
Lunghezza del braccio del drone	l	0.165 [m]
Massa del drone	m	0.8 [Kg]
Momento di inerzia rispetto all'asse x'	J_x	0.005 [Kgm^2]
Momento di inerzia rispetto all'asse y'	J_y	0.005 [Kgm^2]
Momento di inerzia rispetto all'asse z'	J_z	0.009 [Kgm^2]
Spinta massima (per rotore)	T_{max}	4 [N]
Coppia massima (per rotore)	Q_{max}	0.05 [Nm]
Coefficiente di attrito lineare	k_1	0.01 [N/(m/s)]
Coefficiente di attrito angolare	k_2	0 [Nm/(rad/s)]

Tabella 4.1: Parametri del quadrirotore

Capitolo 5

Il controllo del quadrotore

In questo capitolo vengono esposti prima architettura e schema di controllo nativo alla base del controllore *built-in* già implementato in PX4: viene definita una panoramica generale del controllore per poi entrare più nel dettaglio con le definizioni delle variabili e dei parametri in gioco. Infine, si passa alla definizione del nuovo controllore dal punto di vista matematico. L'implementazione in Simulink dello schema di controllo nativo è riportata in Figura 5.1): esso è costruito in tempo discreto, con tempo di campionamento pari a 0.01 [s].

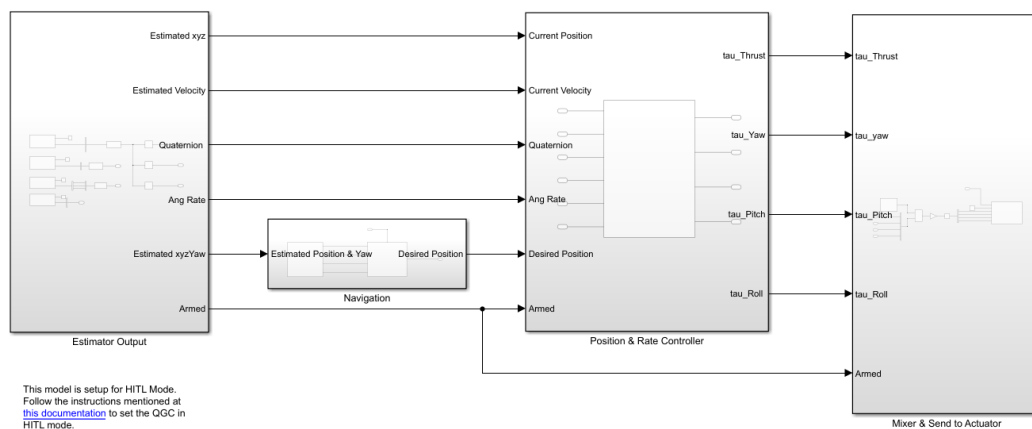


Figura 5.1: Schema di controllo in Simulink.

In particolare, è costituito da 4 blocchi:

- il blocco **Estimator Output** fornisce le misure nel sistema internazionale (SI) di posizione (Earth Frame), angoli e velocità angolari (Body Frame) ottenute durante la simulazione di volo: i dati sono estraibili tramite l'accesso al bus uORB dello stack PX4 e costituiscono lo stato del drone;

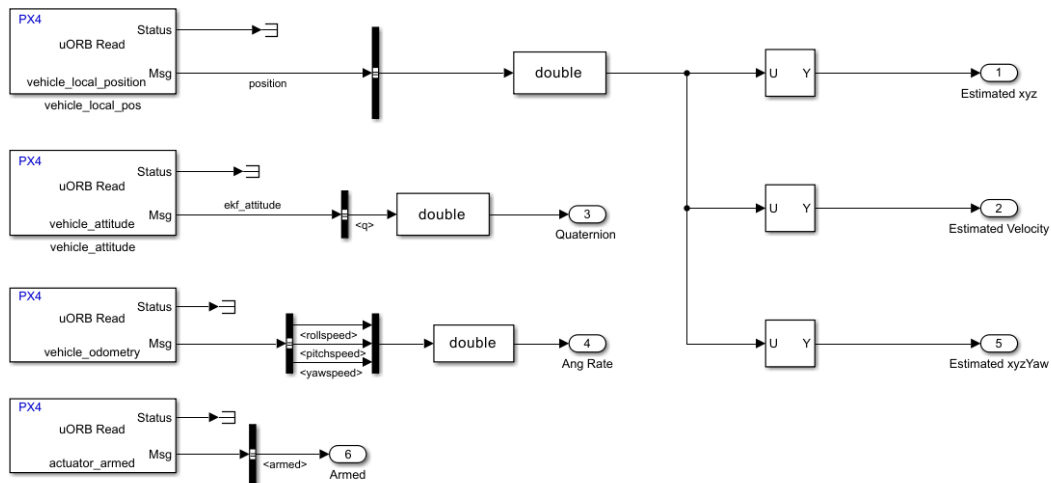


Figura 5.2: Contenuto del blocco **Estimator Output**.

- il blocco **Navigation** prende in ingresso i seipoint del piano di volo impostato sul software QGroundControl e genera i riferimenti necessari al controllore per l'inseguimento di una specifica traiettoria;

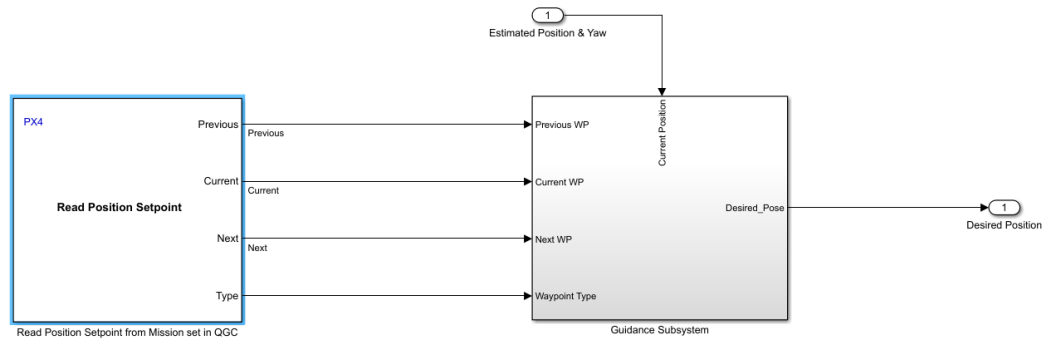


Figura 5.3: Contenuto del blocco **Navigation**.

- il blocco **Position and Rate Controller** costituisce il cuore del controllo implementato in PX4 ed è costruito con un'architettura basata su **Outer-Loop** per il controllo della posizione e della velocità nell'Earth Frame e su **Inner-Loop** per il controllo degli angoli e delle velocità angolari nel Body Frame: il blocco prende in ingresso i riferimenti e le misure dai sensori per generare in output i segnali da mandare ai motori;

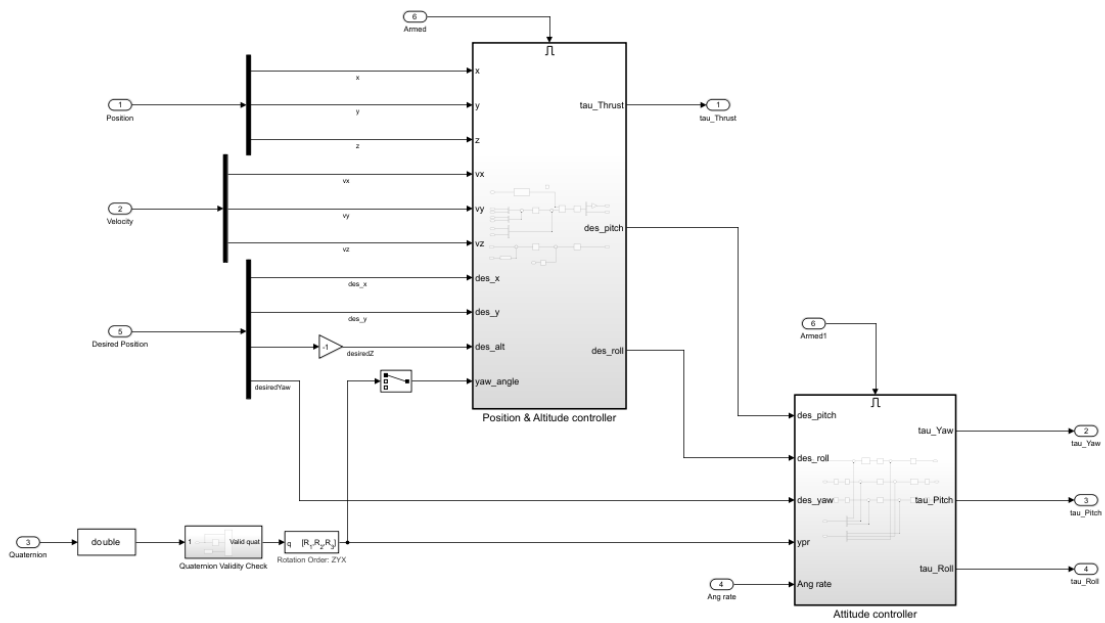


Figura 5.4: Contenuto del blocco **Position and Rate Controller**.

- il blocco **Mixer and Send to Actuator** contiene l'*allocation matrix* e la scalatura in PWM dei segnali generati dal controllore. Difatti, date forze e coppie desiderate, il sistema calcola quanta forza di sollevamento deve generare ogni motore: i segnali dal controllore vengono prima scalati e poi passano per un blocco gestito da jMAVSim che non solo simula il piano di volo del drone una volta avviata la missione, ma anche ESCs e motori;

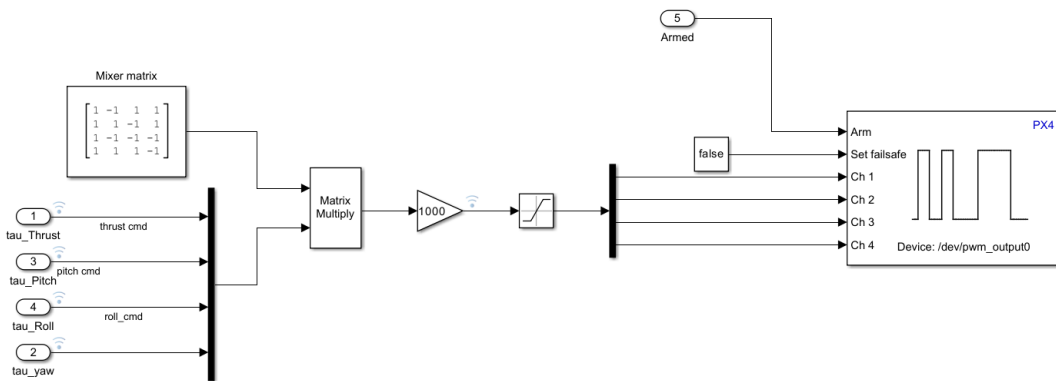


Figura 5.5: Contenuto del blocco **Mixer and Send to Actuator**.

In definitiva, dunque, lo schema nativo del controllo in PX4 per un quadrirotore è composto da due schemi principali riportati di seguito:

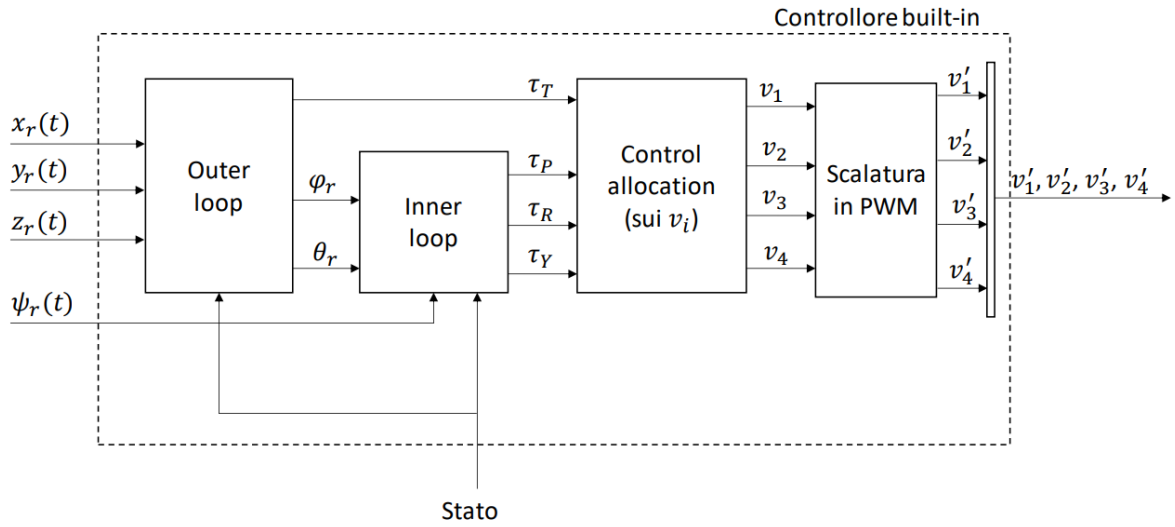


Figura 5.6: Controllore Built-In.

- $x_r(t)$, $y_r(t)$ e $z_r(t)$ sono i riferimenti della posizione del drone rispettivamente lungo gli assi x , y e z e provengono dal navigatore: sono ottenibili impostando altitudes e i diversi waypoints durante il volo in QGC;
- φ_r , θ_r e ψ_r sono i riferimenti degli angoli di assetto del drone: ψ_r è anch'esso ottenibile dal blocco del navigatore ed è calcolato automaticamente con il settaggio dei waypoints, oppure può essere impostato arbitrariamente;
- il vettore dello **Stato**, $X = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \varphi, \theta, \psi, p, q, r]^T \in \mathbb{R}^{12}$, costituisce le misure lineari e angolari del drone in termini di posizione e di velocità;
- τ_T, τ_R, τ_P e τ_Y sono i segnali di controllo uscenti dal controllore e che in fase di allocazione vengono convertiti prima di andare a pilotare i rotori. Verranno ripresi più nel dettaglio in seguito;
- v_1, v_2, v_3 e v_4 sono i segnali convertiti a partire dai τ_i e che passano per una scalatura in PWM prima di accedere agli ESCs dei motori. Verranno definiti nella sezione relativa all'allocazione e alla scalatura in PWM;
- v'_1, v'_2, v'_3 e v'_4 sono i segnali scalati che pilotano effettivamente gli ESCs simulati. Verranno definiti nella sezione relativa all'allocazione e alla scalatura in PWM.

Delle librerie presenti nel firmware gestiscono automaticamente il passaggio dai segnali degli ESCs alle spinte T_i per permettere al drone simulato di mantenere quota e assetto desiderato: lo schema è riportato in Figura 5.7

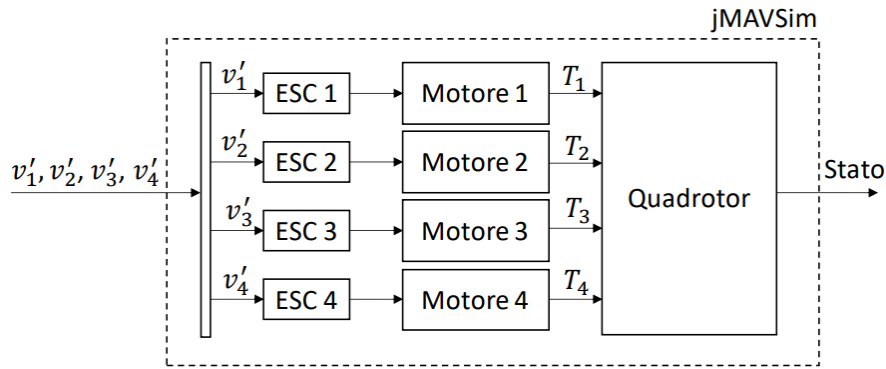


Figura 5.7: Blocco gestito da jMAVSim.

5.1 Controllo nativo di posizione e di velocità nell'Earth Frame (Outer-Loop)

Il controllo di posizione e di velocità nell'Earth Frame avviene tramite due regolatori PID in cascata per ogni componente: il primo regola la posizione, mentre il successivo regola la velocità. Gli input all'Outer-Loop sono i valori di riferimento presi dal QGroundControl mediante il navigatore, e lo stato del drone ottenuto tramite lo stimatore, mentre gli output generati dal blocco **Position and Altitude controller** sono i valori di riferimento φ_r e θ_r necessari all'Inner-Loop e la prima componente utile al controllo dell'altitudine, τ_T (T sta per *Thrust*). In particolare:

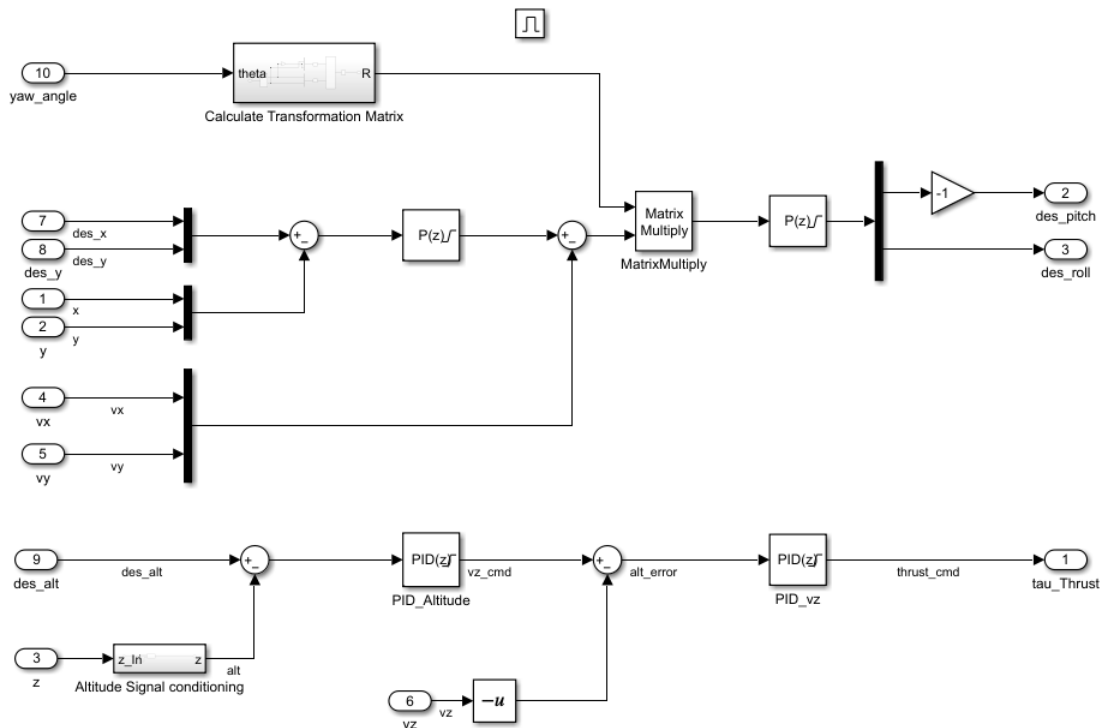


Figura 5.8: Outer-Loop *built-in*.

Nell'Outer-Loop, come si vede, vengono utilizzate due coppie di regolatori. La prima permette di estrarre φ_r e θ_r a partire dal confronto con i riferimenti in x e in y dell'Earth Frame: qui i regolatori P accolgono in ingresso dei vettori, gestendo le singole componenti in maniera

indipendente. La seconda permette invece di generare τ_T a partire dal confronto con il riferimento in z dell'Earth Frame. Nel dettaglio:

- in x e in y :
 - il primo regolatore è un P (doppio, uno per il confronto lungo x e uno per quello lungo y) con k_p proporzionale pari a 0.6, e vi è settato un range di saturazione di velocità compreso tra -4 e 4. L'uscita dal regolatore è confrontata con il vettore di velocità lungo x e y e mandata in ingresso al secondo regolatore;
 - il secondo regolatore è un P con k_p proporzionale pari a 0.3, e vi è settato un range di saturazione per gli angoli φ_r e θ_r compreso tra $-\frac{\pi}{9}$ e $\frac{\pi}{9}$, corrispondenti a -20° e 20° .
- in z :
 - il primo regolatore è un PID con k_p proporzionale pari a 1.5, k_i integrativo pari a 0.01 e k_d derivativo pari a 0.01. Vi è settato inoltre un range di saturazione di velocità compreso tra -2 e 2. L'uscita viene confrontata con la velocità lungo z per poi entrare nel secondo regolatore;
 - il secondo regolatore è un PID con k_p proporzionale pari a 0.5, k_i integrativo pari a 0.1 e k_d derivativo pari a 0.05. Vi è settato inoltre un range di saturazione per il primo segnale di controllo compreso tra 1 e 2, ed è il range che in fase di analisi dei risultati deve essere rispettato per questo segnale:

$$1 < \tau_T < 2$$

5.2 Controllo nativo dell'angolo e della velocità angolare nel Body Frame (Inner-Loop)

Anche qui vengono utilizzati regolatori PID in cascata (3 coppie di PID, una per angolo), in particolare il primo per regolare gli angoli di assetto, il secondo per il confronto di velocità angolare.

I dati in input sono quelli relativi a φ_r e θ_r uscenti dall'Outer-Loop, quello relativo a ψ_r uscente dal navigatore, e quelli relativi agli angoli di assetto (φ , θ e ψ) e alle velocità angolari definite nel Body Frame (p , q ed r) provenienti dallo stimatore.

I segnali di uscita sono le tre componenti rimanenti per il controllo dell'assetto angolare: τ_R , τ_P e τ_Y . In dettaglio:

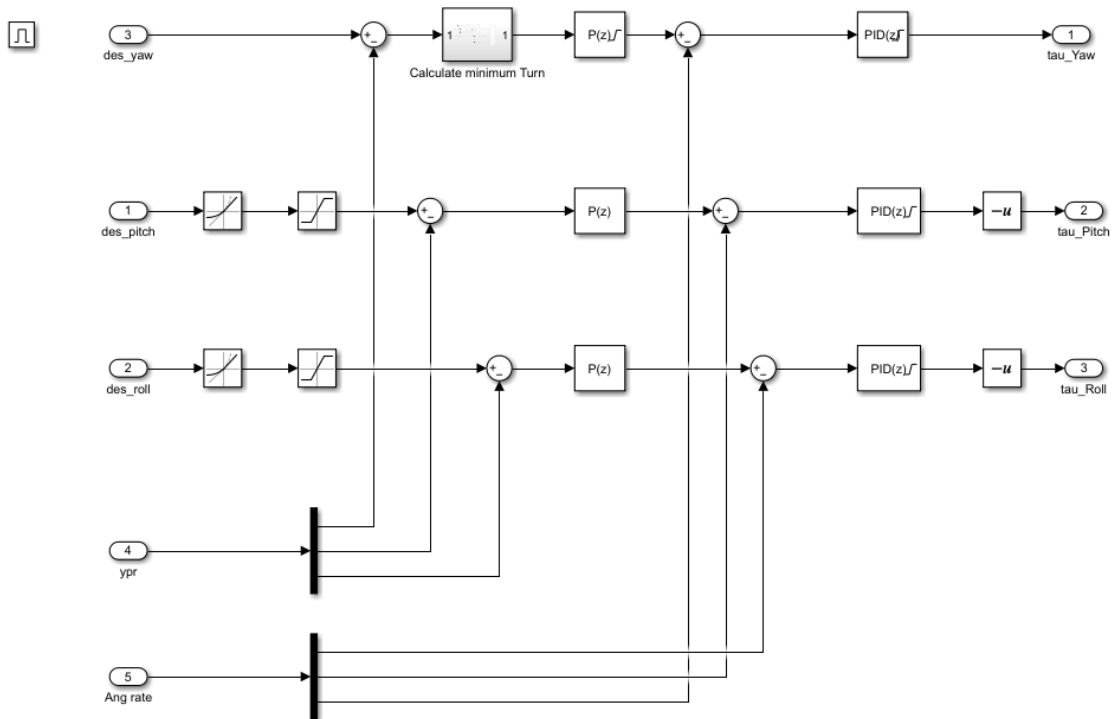


Figura 5.9: Inner-Loop *built-in*.

- per l'angolo ψ (*Yaw*):
 - il primo regolatore è un P con k_p pari a 2, e vi è settato un range di saturazione per la velocità angolare r compreso tra $-\frac{5\pi}{18}$ e $\frac{5\pi}{18}$, ovvero tra -0.87 e 0.87 [rad/s] (-50 e 50 [deg/s] circa). L'uscita viene confrontata con r dell' **Ang rate** ed entra nel secondo regolatore;
 - il secondo regolatore (di velocità) è un PID con k_p proporzionale pari a 0.2, k_i integrativo pari a 0 e k_d derivativo pari a 0, ma ha un coefficiente di filtraggio N pari a 100. Vi è settato un range di saturazione per τ_Y compreso tra -0.1 e 0.1 che deve essere rispettato nella legge definita nel Capitolo 6.
- per l'angolo θ (*Pitch*):
 - il primo regolatore è un P con k_p pari a 4, ma non vi è settato un range di saturazione per la velocità angolare q : questo viene definito nel **rate limiter** prima del regolatore, ed è compreso tra $-\frac{2\pi}{9}$ e $\frac{2\pi}{9}$, ovvero tra -0.7 e 0.7 [rad/s] (-40 e 40 [deg/s] circa). L'uscita viene confrontata con q dell' **Ang rate** ed entra nel secondo regolatore;
 - il secondo regolatore (di velocità) è un PID con k_p proporzionale pari a 0.035, k_i integrativo pari a 0 e k_d derivativo pari a 0.0025, ma ha un coefficiente di filtraggio N pari a 50. Vi è settato un range di saturazione per τ_P compreso tra -0.2 e 0.2 che deve essere rispettato nella legge definita nel Capitolo 6.
- per l'angolo φ (*Roll*):
 - il primo regolatore è un P con k_p pari a 4, ma non vi è settato un range di saturazione per la velocità angolare p : questo viene definito nel **rate limiter** prima del regolatore, ed è compreso tra $-\frac{2\pi}{9}$ e $\frac{2\pi}{9}$, ovvero tra -0.7 e 0.7 [rad/s] (-40 e 40 [deg/s] circa). L'uscita viene confrontata con p dell' **Ang rate** ed entra nel secondo regolatore;

- il secondo regolatore (di velocità) è un PID con k_p proporzionale pari a 0.035, k_i integrativo pari a 0 e k_d derivativo pari a 0.0025, ma ha un coefficiente di filtraggio N pari a 50. Vi è settato un range disaturazione per τ_P compreso tra -0.2 e 0.2 che deve essere rispettato nella legge definita nel Capitolo 6.

Quindi abbiamo:

$$\begin{cases} -0.1 < \tau_Y < 0.1 \\ -0.2 < \tau_P < 0.2 \\ -0.2 < \tau_R < 0.2 \end{cases}$$

5.3 Allocazione e scalatura in PWM (Mixer)

Riprendiamo il modello matematico in forma compatta visto nel Capitolo 5:

$$\begin{cases} m\dot{V} = P + RF_B - F_A \\ J\dot{\omega} + \omega \times J\omega = M_B - M_A \\ \eta = T\omega \end{cases}$$

ed in particolare ricordiamo che:

$$F_B = \begin{bmatrix} 0 \\ 0 \\ -u_1 \end{bmatrix} \quad M_B = \begin{bmatrix} \frac{\sqrt{2}}{2}lu_2 \\ \frac{\sqrt{2}}{2}lu_3 \\ u_4 \end{bmatrix}$$

dove u_i (per $i = 1, \dots, 4$) sono gli ingressi virtuali di controllo. Il modello va riscritto in funzione dei segnali τ_k che escono dal blocco del controllore built-in. Considerando le seguenti premesse:

- La forza generabile dal motore i -esimo è $T_i \in [0, T_{max}]$, con $T_{max} = 4$ (Capitolo 4);
- La coppia generabile dal motore i -esimo è $Q_i = \frac{c_D}{c_L}T_i \in [0, Q_{max}]$, con $Q_{max} = 0.05$ (Capitolo 4) e dove c_D è il coefficiente di attrito lineare che si oppone alla spinta delle pale dei rotori, mentre c_L è il coefficiente di attrito angolare che si oppone alla rotazione delle pale dei rotori.

La forza (normalizzata) generata dal motore i -esimo è $v_i = \frac{T_i}{T_{max}} \in [0, 1]$. Il legame tra gli ingressi virtuali u_i , i segnali di controllo $\tau_T, \tau_R, \tau_P, \tau_Y$ e le forze generate dai motori [23] é:

$$\begin{cases} u_1 = 4T_{max}(\tau_T - 1) = T_1 + T_2 + T_3 + T_4 \\ u_2 = -4T_{max}\tau_R = -T_1 + T_2 + T_3 - T_4 \\ u_3 = -4T_{max}\tau_P = T_1 - T_2 + T_3 - T_4 \\ \frac{c_L}{c_D}u_4 = 4T_{max}\tau_Y = T_1 + T_2 - T_3 - T_4 \end{cases}$$

Dividendo tutto per T_{max} , otteniamo il sistema:

$$\begin{cases} 4(\tau_T - 1) = v_1 + v_2 + v_3 + v_4 \\ -4\tau_R = -v_1 + v_2 + v_3 - v_4 \\ -4\tau_P = v_1 - v_2 + v_3 - v_4 \\ 4\tau_Y = v_1 + v_2 - v_3 - v_4 \end{cases}$$

Possiamo riscrivere il sistema in forma matriciale come segue:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 4(\tau_T - 1) \\ -4\tau_R \\ -4\tau_P \\ 4\tau_Y \end{bmatrix}$$

Isolando il vettore v_i otteniamo:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \tau_T \\ \tau_R \\ \tau_P \\ \tau_Y \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

dove la matrice:

$$M = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

è proprio la **Mixer matrix** del blocco **Mixer and Send to Actuator** di Figura 5.5. Ora, in tale blocco, avviene anche la scalatura in PWM, in modo da generare il vettore:

$$v' = \begin{bmatrix} v'_1 \\ v'_2 \\ v'_3 \\ v'_4 \end{bmatrix}$$

che andrà direttamente in ingresso agli ESCs simulati per generare la forza e le coppie di spinta per controllare il drone. La scalatura è definita come [28]:

$$v'_i = v_i(P_{max} - P_{min}) + P_{min} \in [P_{min}, P_{max}]$$

con $P_{max} = 2000$ e $P_{min} = 1000$ ¹. Per cui abbiamo:

$$v'_i = 1000v_i + 1000$$

e di conseguenza:

$$v_i = \frac{v'_i}{1000} - 1$$

In definitiva, si riporta in Figura 5.10 lo schema completo dell'allocazione, dai segnali di controllo τ_i alle forze di attuazione, passando per le loro scalature in PWM e per i loro rispettivi ESCs:

¹Verificabile anche nel file `motor_params.c` della cartella "Firmware".

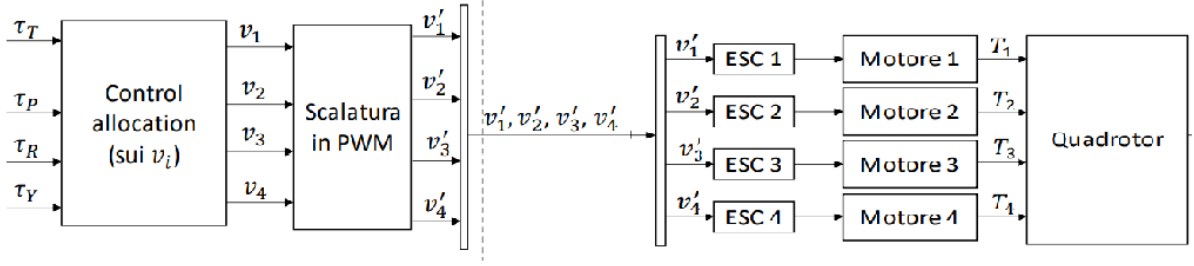


Figura 5.10: Allocazione + scalatura PWM (parte sinistra) e modello ESCs + motori + drone (parte destra).

5.4 Definizione della nuova legge di controllo

Sono diverse le tecniche utilizzate per il controllo di un drone multirottore, come ad esempio:

- **Sliding Mode:** tecnica di controllo robusto per sistemi non lineari: lo stato del sistema viene fatto "scivolare" su un intorno di una superficie di *sliding* opportunamente scelta in modo da stabilizzare asintoticamente il sistema [23];
- **Proportional-Integral-Derivative (PID):** tecnica di controllo che si basa sulla retroazione, in particolare sul confronto tra un segnale in uscita dal sistema in esame e un suo valore di riferimento. L'errore, definito come differenza tra i due, costituisce l'input del regolatore PID, il quale restituisce in uscita il nuovo ingresso del sistema [30];
- **Feedback Linearization:** tecnica di controllo per sistemi non lineari che effettua una linearizzazione del sistema retroazionato mediante un nuovo ingresso opportunamente costruito. Successivamente, viene utilizzata una tecnica di controllo lineare per stabilizzare asintoticamente il sistema stesso [19];

L'obiettivo in questo progetto è costruire ed implementare una legge di controllo di traiettoria di un drone basata su modello matematico. A tal proposito, la definizione della nuova legge avviene prima mediante **Feedback Linearization**, ovvero linearizzando il sistema mediante un feedback in funzione del vettore dei segnali di controllo τ_i . Poi, dal sistema linearizzato, si usa una tecnica di controllo lineare per renderlo asintoticamente stabile: quella usata in questo progetto è la **retroazione dallo stato** del sistema errore linearizzato, ed è utilizzata per ricostruire sia l'Outer-Loop sia l'Inner-Loop.

5.5 Nuovo controllo di posizione e di velocità nell'Earth Frame (Outer-Loop)

Prendiamo la forma matriciale delle equazioni relative alla traslazione nell'Earth Frame:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ -u_1 \end{bmatrix} - k_1 I \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \\ \dot{z}|\dot{z}| \end{bmatrix}$$

sapendo che:

$$R = \begin{bmatrix} c(\theta)c(\varphi) & -c(\theta)s(\psi) + c(\psi)s(\theta)s(\varphi) & s(\psi)s(\varphi) + c(\psi)c(\varphi)s(\theta) \\ c(\theta)s(\psi) & c(\psi)c(\varphi) + s(\theta)s(\varphi)s(\psi) & -s(\varphi)c(\psi) + c(\varphi)s(\theta)s(\psi) \\ -s(\theta) & c(\theta)s(\varphi) & c(\theta)c(\varphi) \end{bmatrix}$$

con $c = \cos$ ed $s = \sin$.

Trattiamo \ddot{z} da una parte e $[\ddot{x}, \ddot{y}]^T$ da un'altra, mentre i segnali di ingresso e di uscita dell'Outer-Loop sono gli stessi visti precedentemente:

- scriviamo l'equazione scalare in z :

$$\ddot{z} = g - \frac{1}{m}(\cos(\theta)\cos(\varphi))u_1 - \frac{k_1}{m}\dot{z}|\dot{z}|$$

con $u_1 = T_1 + T_2 + T_3 + T_4 = 4T_{max}(\tau_T - 1)$. Ora, sia $z_r(t)$ il riferimento per z . La legge di controllo è quindi scelta come:

$$\tau_T = 1 - \frac{m}{4T_{max}(\cos\varphi)(\cos\theta)}(-g + k_1\dot{z}|\dot{z}| + v_z)$$

dove $v_z = \ddot{z}_r - k_{z_1}(\dot{z} - \dot{z}_r) - k_{z_0}(z - z_r)$. Infatti, sostituendo τ_T nel modello viene:

$$\ddot{z} = v_z$$

da cui, usando l'errore $e_z = z - z_r$ si ha:

$$\ddot{e}_z + k_{z_1}\dot{e}_z + k_{z_0}e_z = 0.$$

Arriviamo quindi ad un sistema errore lineare e tempo invariante: la scelta di k_{z_1} e $k_{z_0} \in \mathbb{R}$ deve essere tale per cui il polinomio

$$P(s) = s^2 + k_{z_1}s + k_{z_0}$$

abbia tutte le radici (dette anche poli) a parte reale strettamente minore di 0. Essendo un sistema del secondo ordine, scegliamo opportunamente 2 poli, p_{z_1} e p_{z_2} , e di conseguenza $k_{z_1} = -(p_{z_1} + p_{z_2})$ e $k_{z_0} = p_{z_1}p_{z_2}$. **Il ragionamento è analogo per le altre leggi di controllo trattate di seguito.**

- le equazioni in x e in y le scriviamo in forma matriciale considerando l'approssimazione per piccoli angoli di ϕ e θ (ad esempio, per ϕ avremo $\cos(\phi) \approx 1$ e $\sin(\phi) \approx \phi$). Per cui il sistema diventa:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = -\frac{4T_{max}(\tau_T - 1)}{m} \begin{bmatrix} \sin\psi & \cos\psi \\ -\cos\psi & \sin\psi \end{bmatrix} \begin{bmatrix} \varphi \\ \theta \end{bmatrix} - \frac{1}{m} \begin{bmatrix} k_1 & 0 \\ 0 & k_1 \end{bmatrix} \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \end{bmatrix}$$

L'espressione per la τ_T è stata già determinata, quindi abbiamo tutti i dati per poter imporre un opportuno vettore $[\phi, \theta]^T$ in modo che il sistema sia asintoticamente stabile. Riscriviamo il sistema in modo più semplice, imponendo i seguenti passaggi per via delle grandezze già note:

$$F = -\frac{4T_{max}(\tau_T - 1)}{m} \begin{bmatrix} \sin\psi & \cos\psi \\ -\cos\psi & \sin\psi \end{bmatrix}$$

$$G = \frac{1}{m} \begin{bmatrix} k_1 & 0 \\ 0 & k_1 \end{bmatrix} \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \end{bmatrix}$$

per cui abbiamo:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = F \begin{bmatrix} \varphi \\ \theta \end{bmatrix} - G$$

La legge di controllo scelta a questo punto è:

$$\begin{bmatrix} \varphi \\ \theta \end{bmatrix} = F^{-1}(V_{xy} + G)$$

dove

$$V_{xy} = -k_{xy1} \begin{bmatrix} (\dot{x} - \dot{x}_r) \\ (\dot{y} - \dot{y}_r) \end{bmatrix} - k_{xy0} \begin{bmatrix} (x - x_r) \\ (y - y_r) \end{bmatrix} + \begin{bmatrix} \ddot{x}_r \\ \ddot{y}_r \end{bmatrix}$$

Infatti, sostituendo viene:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = FF^{-1}(V_{xy} + G) - G = V_{xy}$$

e quindi:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = -k_{xy1} \begin{bmatrix} (\dot{x} - \dot{x}_r) \\ (\dot{y} - \dot{y}_r) \end{bmatrix} - k_{xy0} \begin{bmatrix} (x - x_r) \\ (y - y_r) \end{bmatrix} + \begin{bmatrix} \ddot{x}_r \\ \ddot{y}_r \end{bmatrix}$$

Ora abbiamo le uscite τ_T , φ_r e θ_r dall'Outer-Loop: φ_r e θ_r vanno in ingresso all'Inner-Loop, mentre il segnale τ_T va direttamente nel Mixer.

5.6 Nuovo controllo dell'angolo e della velocità angolare nel Body Frame (Inner-Loop)

Prendiamo le equazioni relative alle rotazioni e alla cinematica:

$$\begin{cases} \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} l u_2 \\ \frac{\sqrt{2}}{2} l u_3 \\ u_4 \end{bmatrix} - k_2 I \begin{bmatrix} p|p| \\ q|q| \\ r|r| \end{bmatrix} \\ \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & t(\theta)s(\varphi) & t(\theta)c(\varphi) \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & \frac{s(\varphi)}{c(\theta)} & \frac{c(\varphi)}{c(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \end{cases}$$

e la sua forma compatta:

$$\begin{cases} J\dot{\omega} + \omega \times J\omega = M_B \\ \eta = T\omega \end{cases}$$

tenendo conto che $k_2 = 0$ (Capitolo 4). Sviluppiamo ora la dinamica della legge cinematica:

$$\dot{\eta} = \dot{T}\omega + T\dot{\omega} = \dot{T}\omega - TJ^{-1}C - TJ^{-1}H \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

dove

$$C = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \quad H = \begin{bmatrix} \frac{\sqrt{2}}{2} l & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} l & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ricordando che:

$$\begin{cases} u_2 = -4T_{max}\tau_R = -T_1 + T_2 + T_3 - T_4 \\ u_3 = -4T_{max}\tau_P = T_1 - T_2 + T_3 - T_4 \\ u_4 = 4Q_{max}\tau_Y = Q_1 + Q_2 - Q_3 - Q_4 \end{cases}$$

allora possiamo scrivere come segue:

$$\dot{\eta} = \dot{T}\omega - TJ^{-1}C - TJ^{-1}H' \begin{bmatrix} \tau_R \\ \tau_P \\ \tau_Y \end{bmatrix}$$

dove

$$H' = H \begin{bmatrix} -4T_{max} & 0 & 0 \\ 0 & -4T_{max} & 0 \\ 0 & 0 & 4Q_{max} \end{bmatrix}$$

A questo punto, possiamo definire la legge di controllo come:

$$\begin{bmatrix} \tau_R \\ \tau_P \\ \tau_Y \end{bmatrix} = (TJ^{-1}H')^{-1}(V_{rpy} - \dot{T}\omega + TJ^{-1}C)$$

dove

$$V_{rpy} = -K_{rpy_1}T(\omega - \omega_r) - K_{rpy_0}(\alpha - \alpha_r) + \ddot{\alpha}_r,$$

con

$$K_{rpy_1} = \begin{bmatrix} k_{rp_1} & 0 & 0 \\ 0 & k_{rp_1} & 0 \\ 0 & 0 & k_{y_1} \end{bmatrix}, \quad T(\omega - \omega_r) = T \begin{bmatrix} (p - p_r) \\ (q - q_r) \\ (r - r_r) \end{bmatrix} = \begin{bmatrix} (\dot{\varphi} - \dot{\varphi}_r) \\ (\dot{\theta} - \dot{\theta}_r) \\ (\dot{\psi} - \dot{\psi}_r) \end{bmatrix} = (\eta - \eta_r),$$

$$K_{rpy_0} = \begin{bmatrix} k_{rp_0} & 0 & 0 \\ 0 & k_{rp_0} & 0 \\ 0 & 0 & k_{y_0} \end{bmatrix}, \quad (\alpha - \alpha_r) = \begin{bmatrix} (\varphi - \varphi_r) \\ (\theta - \theta_r) \\ (\psi - \psi_r) \end{bmatrix}, \quad \ddot{\alpha}_r = \begin{bmatrix} \ddot{\varphi}_r \\ \ddot{\theta}_r \\ \ddot{\psi}_r \end{bmatrix}$$

Infatti, sostituendo viene:

$$\dot{\eta} = \dot{T}\omega - TJ^{-1}C - TJ^{-1}H'(TJ^{-1}H')^{-1}(V_{rpy} - \dot{T}\omega + TJ^{-1}C) = V_{rpy}$$

ovvero

$$\dot{\eta} = -K_{rpy_1}(\eta - \eta_r) - K_{rpy_0}(\alpha - \alpha_r) + \ddot{\alpha}_r$$

Capitolo 6

Implementazione e analisi dei dati

In questo capitolo viene implementata in Matlab/Simulink la legge di controllo vista nel capitolo precedente. Essa è costituita da 3 blocchi: 2 per l'Outer-Loop (un blocco per il controllo dell'altitudine (asse z) e uno per il controllo nel piano (x, y)) e uno per l'Inner-Loop (per il controllo degli angoli di assetto).

6.1 Implementazione della nuova legge di controllo

Nel blocco del modello Simulink **Position and Rate Controller** (Figura 5.4) si entra nel sottoblocco per il controllo della posizione e della velocità nell'Earth Frame per costruire i primi due blocchi di codice dell'Outer-Loop:

- **controllo lungo z** : il blocco riceve in ingresso i seguenti segnali:
 - **des_alt**: è l'altitudine desiderata impostata su QGC: essa viene calcolata nel tempo dal navigatore in base alla quota da raggiungere e alla velocità di lancio. Va tuttavia cambiata di segno in quanto siamo in sistema NED e il riferimento viene impostato con il segno $+$ sull'asse z ;
 - **z** : è la misura stimata dell'altitudine durante il volo proveniente dallo stimatore (Figura 5.2);
 - **vz** : è la misura stimata della velocità durante il lancio proveniente dallo stimatore;
 - **angle**: è il vettore degli angoli ϕ, θ e ψ stimati, proveniente dallo stimatore.

Il blocco restituisce in output il segnale τ_T per il controllo dell'altitudine. Di seguito sono riportati schema (Figura 6.1) e codice:

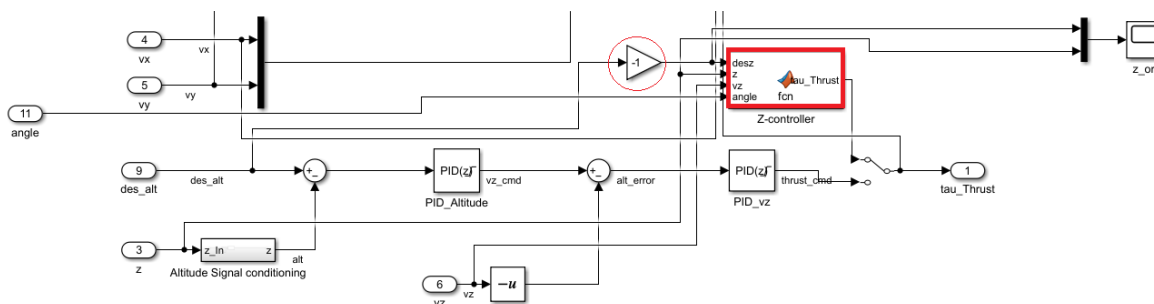


Figura 6.1: Schema per il controllo su z implementato.

Listing 6.1: Controllo in z

```
function tau_Thrust = fcn(desz , z , vz , angle)

%Parametri quadrirotore
m = 0.8;
g = 9.81;
k1 = 0.01;
Tmax = 4;
T = 4*Tmax;
roll = angle(3);      % Misura (stimata) di roll [rad]
pitch = angle(2);    % Misura (stimata) di pitch [rad]

%Allocazione autovalori
pz1 = -0.8;
pz2 = -0.6;
kz0 = pz1*pz2;
kz1 = -(pz1 + pz2);

%Controllo su asse Z
Vz = -kz1*vz - kz0*(z-desz);
u1 = (m*(g - Vz) - k1*vz*abs(vz))/(cos(roll)*cos(pitch));
tau_Thrust = (u1/T) + 1;

end
```

- **controllo su x e y :** il blocco riceve in ingresso il segnale di controllo τ_T calcolato in precedenza e i seguenti segnali stimati provenienti dal navigatore:
 - **des_x** e **des_y:** sono i riferimenti lungo gli assi x e y , e vengono calcolati nel tempo in base ai waypoint impostati e alla velocità di volo;
 - **x** e **y:** sono le misure stimate lungo gli assi x e y durante il volo e vengono dallo stimatore (Figura 5.2);
 - **vx** e **vy:** sono le misure stimate delle velocità lungo gli assi x e y e provengono dallo stimatore;
 - **yaw_angle:** è la misura stimata dell'angolo ψ .

Il blocco restituisce in uscita i valori di riferimento ϕ_r e θ_r necessari per l'Inner-loop. Di seguito sono riportati schema (Figura 6.2) e codice:

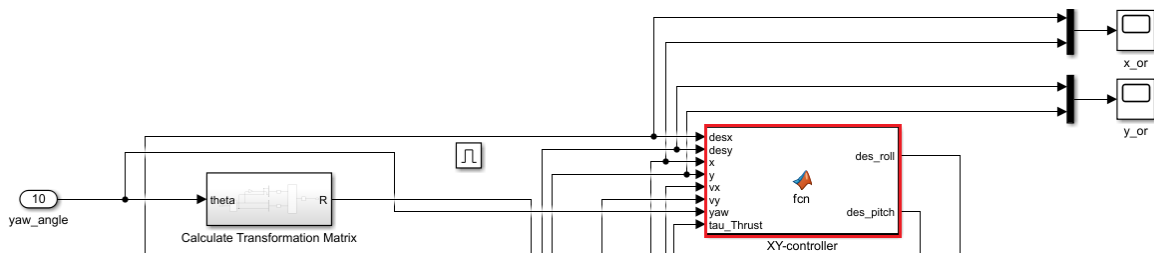


Figura 6.2: Schema per il controllo su xy implementato.

Listing 6.2: Controllo in (x, y)

```
function [des_roll, des_pitch] =
fcn(desx, desy, x, y, vx, vy, yaw, tau_Thrust)

%Parametri quadrirotore
m = 0.8;
k1 = 0.01;
Tmax = 4;
T = 4*Tmax;
u1 = T*(tau_Thrust - 1);

%Allocazione autovalori
p1xy = -0.8;
p2xy = -0.6;
kxy0 = p1xy*p2xy;
kxy1 = -(p1xy+p2xy);

%Controllo su assi X e Y
F = (-u1/m)*[sin(yaw), cos(yaw); -cos(yaw), sin(yaw)];
G = (k1/m)*[vx*abs(vx); vy*abs(vy)];
Vxy = -kxy1*[vx;vy] - kxy0*[x-desx; y-desy];
out = F\(Vxy + G);
des_roll = out(1);
des_pitch = out(2);

end
```

Per quanto riguarda il controllo degli angoli di assetto nell'Inner-Loop, viene utilizzato un singolo blocco, i cui input sono:

- **roll, pitch** e **yaw**: sono le componenti del vettore ypr , ovvero delle misure stimate degli angoli provenienti dallo stimatore (Figura 5.2);
- **des_roll, des_pitch** e **des_yaw**: i primi due riferimenti vengono dall'Outer-Loop visto in precedenza, mentre il riferimento per ψ viene dal navigatore;
- **omega**: è il vettore delle velocità angolari nel Body Frame, ovvero p, q e r provenienti dallo stimatore (Ang rate).

Gli output del blocco sono le 3 componenti di controllo τ_R, τ_P e τ_Y che, insieme a τ_T calcolato nell'Outer-Loop, vanno nel mixer. Di seguito sono riportati schema (Figura 6.3) e codice dell'Inner-Loop:

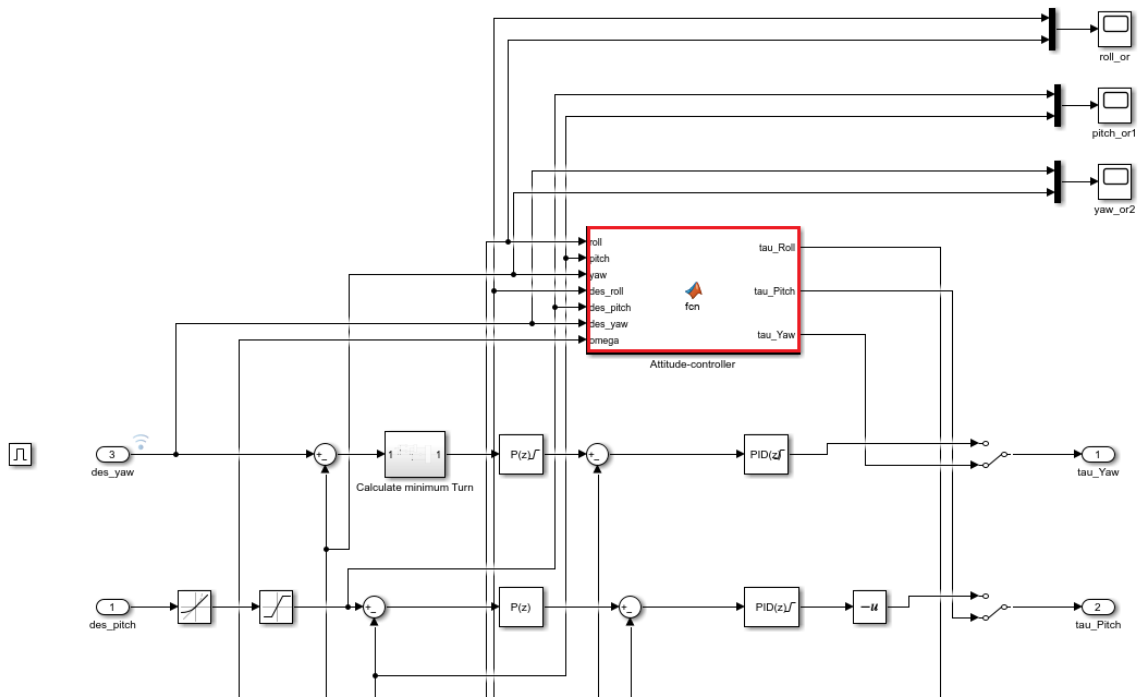


Figura 6.3: Schema per il controllo degli angoli di assetto implementato.

Listing 6.3: Controllo di φ , θ , ψ

```
function [tau_Roll,tau_Pitch,tau_Yaw] =
fcn(roll ,pitch ,yaw ,des_roll ,des_pitch ,des_yaw ,omega)

%Parametri quadrirotore
Jx = 0.005;
Jy = 0.005;
Jz = 0.009;
l = 0.165;
Tmax = 4;
Qmax = 0.05;
TM = 4*Tmax;
QM = 4*Qmax;

%Cinematica
T = [1 sin(roll)*tan(pitch) cos(roll)*tan(pitch);
0 cos(roll) -sin(pitch);
0 sin(roll)/cos(pitch) cos(roll)/cos(pitch)];

dangle = T*omega;
droll = dangle(1);
dpitch = dangle(2);
dT = [0 (dpitch*roll+droll*pitch) dpitch;
0 0 -droll;
0 droll 0]; % Derivata della matrice T (approssimata per piccoli angoli)

angle = [roll; pitch; yaw];
```

```

des_angle = [des_roll; des_pitch; des_yaw];
J = diag([Jx Jy Jz]);
H = diag([sqrt(2)*1/2 sqrt(2)*1/2 1]);
C = cross(omega, J*omega);

%Allocazione autovalori
p_rp1 = -6;
p_rp2 = -5;
p_y1 = -0.6;
p_y2 = -0.5;
krp1 = -(p_rp1+p_rp2);
ky1 = -(p_y1+p_y2);
krp0 = p_rp1*p_rp2;
ky0 = p_y1*p_y2;

%Controllo degli angoli di assetto
Vrpy = -diag([krp1 krp1 ky1])*T*omega
-diag([krp0 krp0 ky0])*(angle-des_angle);
out = (T*J\H)\(Vrpy - dT*omega + T*J\C);
u2 = out(1);
u3 = out(2);
u4 = out(3);
tau_Roll = -u2/TM;
tau_Pitch = -u3/TM;
tau_Yaw = u4/QM;

end

```

6.2 Analisi della nuova legge di controllo

Per l'analisi della legge di controllo è riportata di seguito una prova di volo impostata tramite il software QGC. Una volta aperto il modello in Matlab/Simulink e dopo aver fatto le configurazioni riportate nel Capitolo 3, si connette la PixHawk al pc tramite USB e convertitore, si accede alla sezione **HARDWARE** di Simulink e si procede con **Monitor and Tune**: automaticamente si apriranno QGC e il simulatore jMAVSim.

A questo punto, spostandoci sul software QGC, è possibile impostare il punto di lancio (*launch*) da cui il drone inizia il decollo e gli altri waypoints per definire il percorso che il drone dovrà seguire. Viene impostata una traiettoria chiusa come in Figura 6.4

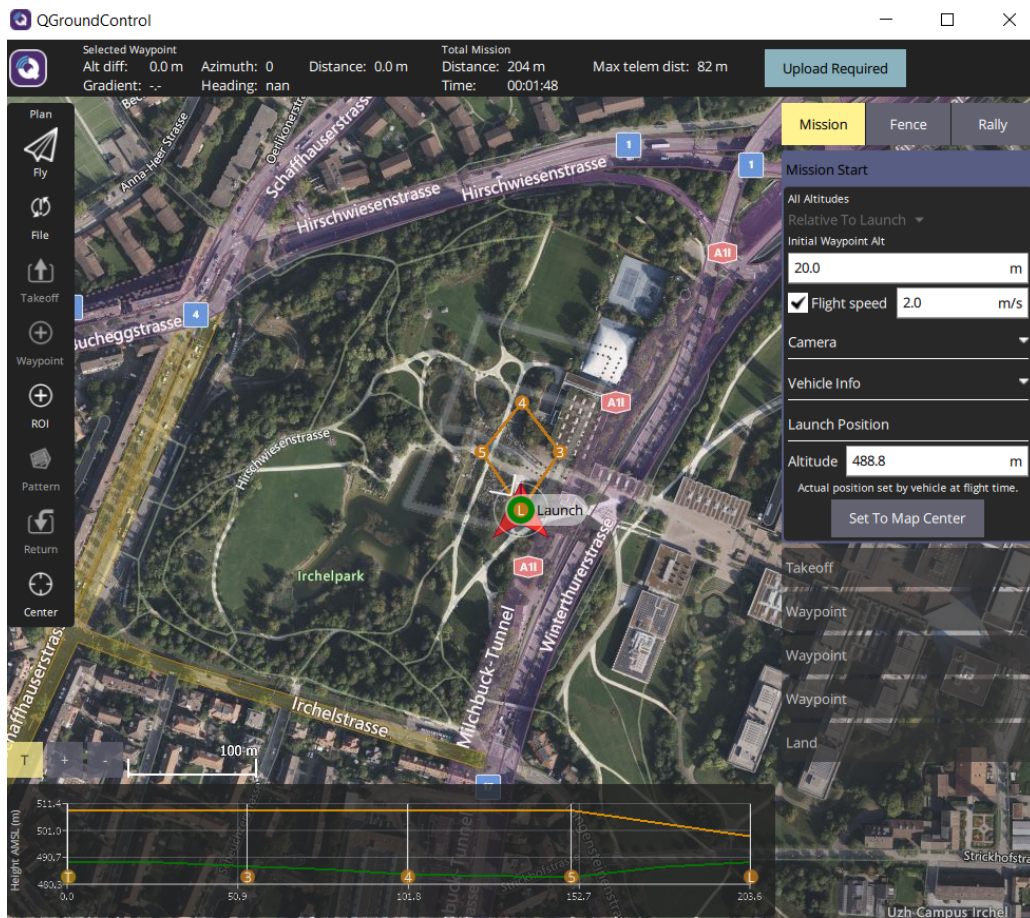


Figura 6.4: Percorso di volo impostato.

Come si può notare, prima di avviare la missione, si procede con la definizione dei seguenti punti sulla mappa:

- **Mission Start (punto 1):** imposta il punto di *launch*, ovvero il punto della mappa da cui il drone inizierà il decollo lungo l'asse z fino ad arrivare all'altitudine desiderata: una volta raggiunta, la missione partirà. Per la simulazione è stata definita un'altitudine di partenza di $20m$.
- **Takeoff (punto 2):** è il punto da cui inizia la missione di volo: spesso viene fatto coincidere con l'altitudine raggiunta dopo il lancio iniziale (sulla mappa, i punti 1 e 2 coincidono);
- **WayPoints (punti 3, 4 e 5):** sono i punti raggiunti dal drone durante la missione e definiscono i riferimenti per x , y e z . I riferimenti vengono calcolati in base alla distanza tra un punto e un altro e alla velocità di volo (impostata di default a $2m/s$ in simulazione); infatti, occorre un tempo t per raggiungere i punti di riferimento.
- **Return (punto 6):** il punto di fine corsa e di ritorno al punto di partenza, di cui si può scegliere la modalità. Quella in simulazione è stata configurata sul *landing* del drone, ovvero nel punto 6 il drone rimarrà nel tempo fisso ad una quota desiderata: quella impostata in simulazione è pari a $10m$.

Una volta definiti i punti, si procede con l'upload della missione e si fa partire la simulazione per testare il controllo. In Figura 6.5 è riportato il drone durante il *landing* finale: a sinistra il drone in 3D in jMAVSIM, mentre a destra il percorso effettuato dal drone su QGC segnato in rosso.

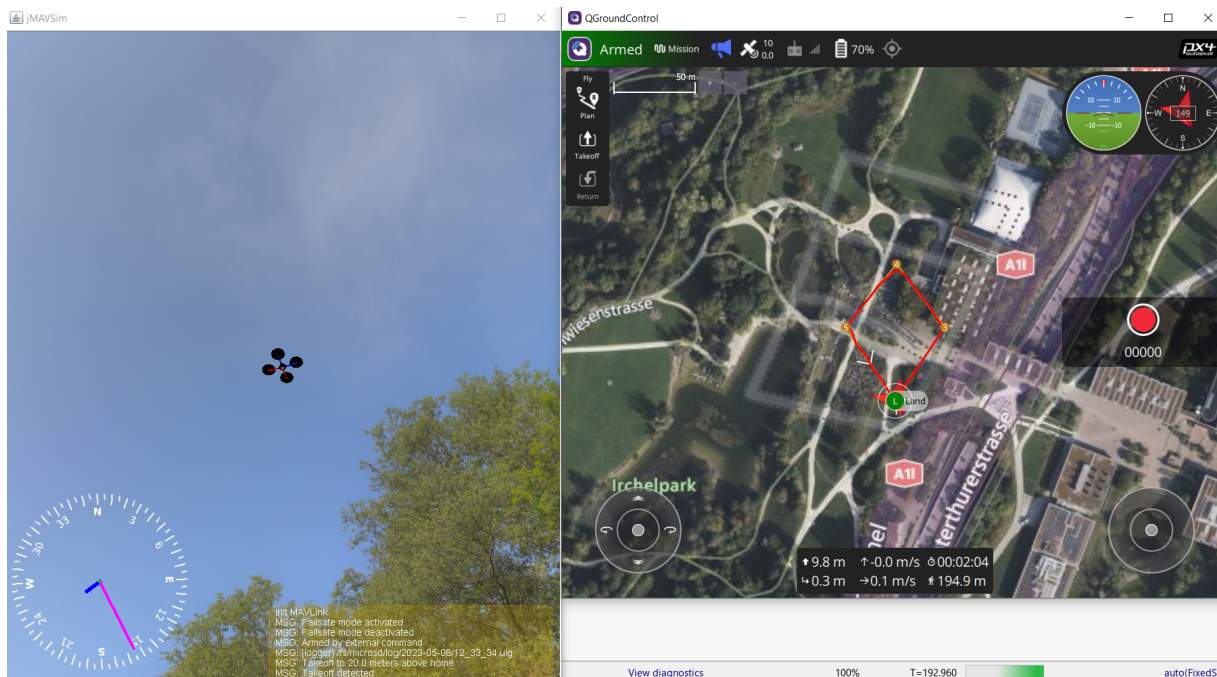


Figura 6.5: Stato finale del percorso - *landing*.

I dati di volo sono stati riportati sul workspace di Matlab tramite degli *scope* messi opportunamente per registrare i valori dei segnali di controllo τ_i , quelli delle misure stimate durante il volo e quelli dei riferimenti. I *plot* vengono riportati di seguito:

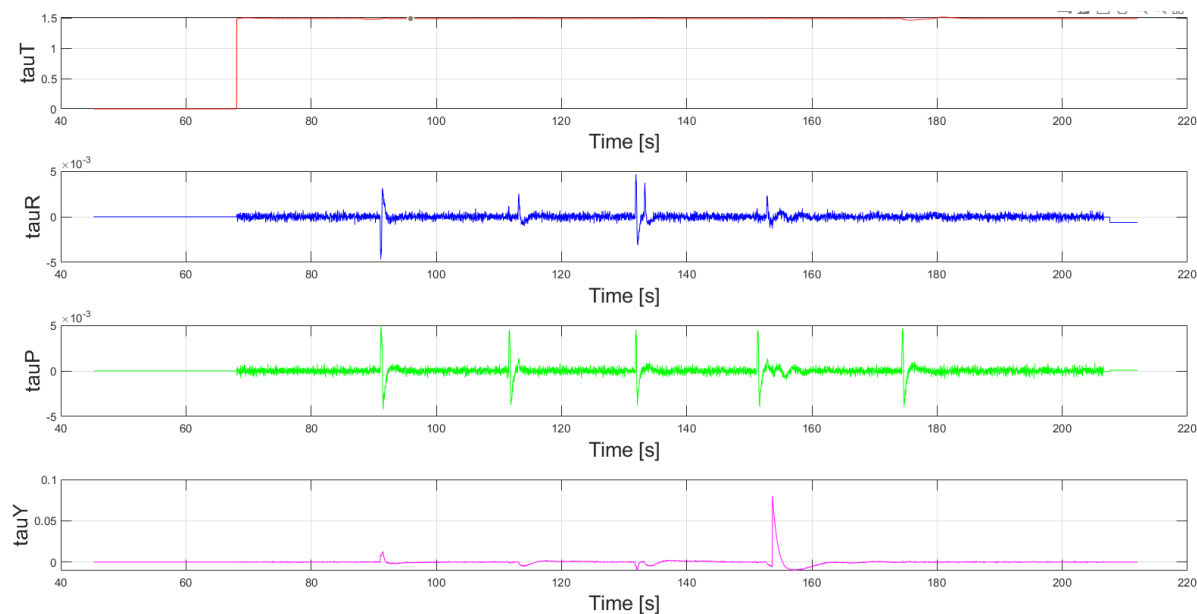


Figura 6.6: *Plotting* dei segnali τ_i nel tempo.

La simulazione è stata avviata dopo circa $68s$ dalla sincronizzazione con il simulatore e il software QGC. Questo giustifica i tratti nulli o pari a 0 prima di quell'istante, mentre quelli nulli dopo circa $210s$, ovvero dopo il *landing*, sono dovuti alla desincronizzazione. Come si può vedere in Figura 6.6, i segnali di controllo rientrano nei range stabiliti nel Capitolo 5, in particolare:

- τ_T sale non appena il drone prende quota e rimane circa costante per tutta la durata del volo in modo da poterla mantenere: dal momento in cui si stabilizza in prossimità della quota desiderata, la varianza (σ^2) di τ_T rimane piccola;
- τ_R , τ_P e τ_Y rimangono prossimi allo 0 (σ^2 piccola) ma con dei picchi in prossimità dei waypoints per il cambio della traiettoria. In particolare, τ_Y raggiunge il suo picco massimo nel waypoint 5: questo perchè in quel punto l'angolo ψ va in prossimità di $-\pi$ radianti, ovvero la sua soglia minima (Capitolo 4). Non riuscendo a riconoscere π radianti come punto vicino, l'angolo ψ viene ripercorso al contrario sotto l'azione di uno sforzo significativo e questo rappresenta un problema.

Considerando che, dato un vettore del tipo $X = [x_1 \dots x_n]^T$, il suo valor medio μ_X è dato da:

$$\mu_X = \frac{1}{n} \sum_{i=1}^n x_i$$

e che la sua varianza $\sigma^2(X)$ è definita come:

$$\sigma^2(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)^2$$

i segnali τ_i sono vettori colonna con 9307 elementi nel workspace di Matlab. Riportiamo i dati nella seguente Tabella 6.1: si può vedere come sia valor medio sia varianza dei τ_i sono racchiusi in un intervallo molto piccolo, per cui per tutta la durata del volo, essi mantengono mediamente valori stabili.

	τ_T	τ_R	τ_P	τ_Y
Max	1.516	0.005	0.005	0.080
Min	0	-0.005	-0.004	-0.001
μ_{τ_i}	1.491	-1.904e ⁻⁰⁵	3.187e ⁻⁰⁶	-5.458e-06
$\sigma^2(\tau_i)$	2.647e ⁻⁰⁵	1.472e ⁻⁰⁷	2.574e ⁻⁰⁷	1.667e ⁻⁰⁵

Tabella 6.1: Tabella dei risultati per τ_i .

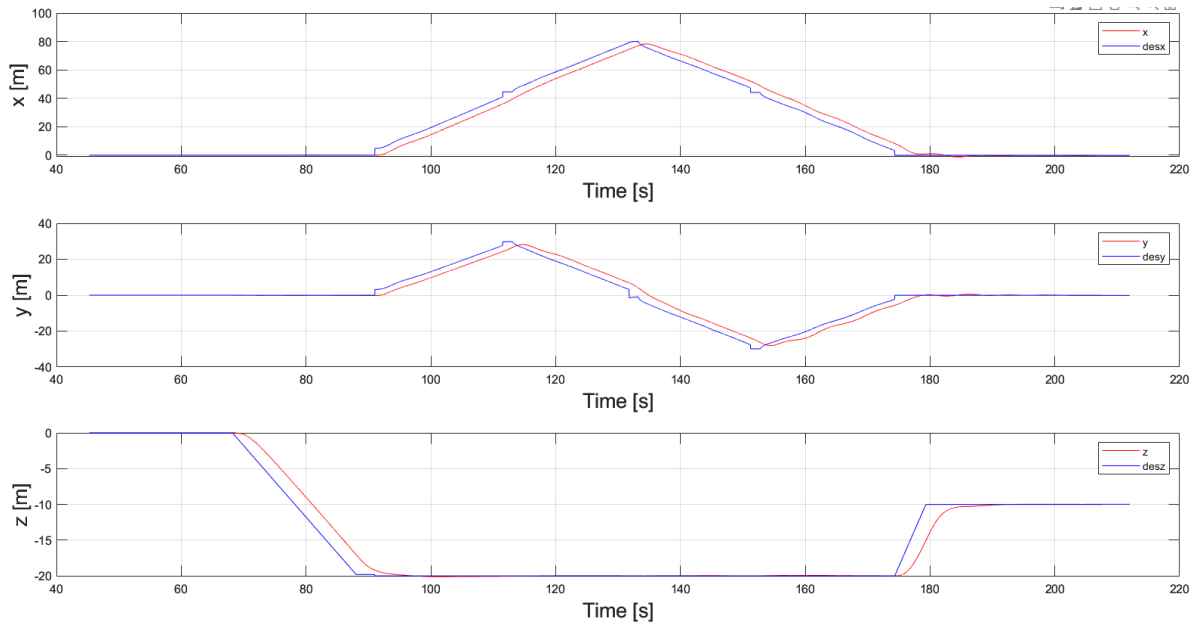


Figura 6.7: *Plotting* delle traiettorie nel tempo.

In Figura 6.7 vengono messe a confronto le traiettorie stimate con i loro riferimenti. Questi ultimi, in particolare, vengono calcolati nel tempo come:

$$WayPoint_f - WayPoint_i = V * \Delta t$$

dove V è la velocità di volo: in questa simulazione, la velocità lungo x e y è pari a circa $2m/s$, mentre quella su z è pari a $1m/s$ circa in fase di decollo, pari a $2m/s$ circa in fase di atterraggio (di default); $WayPoint_f$ è il waypoint di arrivo; $WayPoint_i$ è il waypoint di partenza e Δt è il tempo per percorrere il tratto.

Per cui i riferimenti non "saltano" da un punto ad un altro istantaneamente (difatti, il riferimento non è un gradino ma una rampa). Dopo 68s dalla sincronizzazione inizia il lancio e il drone sale in prossimità della quota desiderata ($20m$) per poi iniziare la missione attraverso i waypoints impostati. Il volo termina con il ritorno del drone nella posizione di partenza, per poi eseguire il *landing*: qui il drone passa da una quota di circa $20m$ ad una quota di circa $10m$ e vi rimane.

In Tabella 6.2 riportiamo i risultati dell'analisi basata su valor medio e varianza degli errori calcolati come:

$$\begin{cases} e_x = x - desx \\ e_y = y - desy \\ e_z = z - desz \end{cases}$$

Tenendo presente che le traiettorie e anche i riferimenti sono vettori colonna con 9307 elementi salvati nel workspace di Matlab, avremo:

	e_x	e_y	e_z
μ	0.014	0.016	0.173
σ^2	12.767	6.722	1.392

Tabella 6.2: Tabella dei risultati per x , y e z .

Gli errori medi in x e y risultano più piccoli di quello in z per la simmetria del percorso scelto nel piano (x, y) . Inoltre, le varianze sono tanto più grandi quanto più è lungo il tempo

per arrivare al waypoint successivo e quanto più è corto il tempo in cui il drone rimane fermo in quel waypoint: in x e y risultano relativamente grandi in quanto le misure seguono, per quasi tutta la durata del tragitto, i riferimenti partendo in ritardo rispetto a questi ultimi (per ogni waypoint) e solo nel tratto finale (*landing*) la varianza dell'errore diventa piccola (come anche l'errore stesso), in quanto in quel tratto l'errore è sempre più prossimo al suo valore medio. .

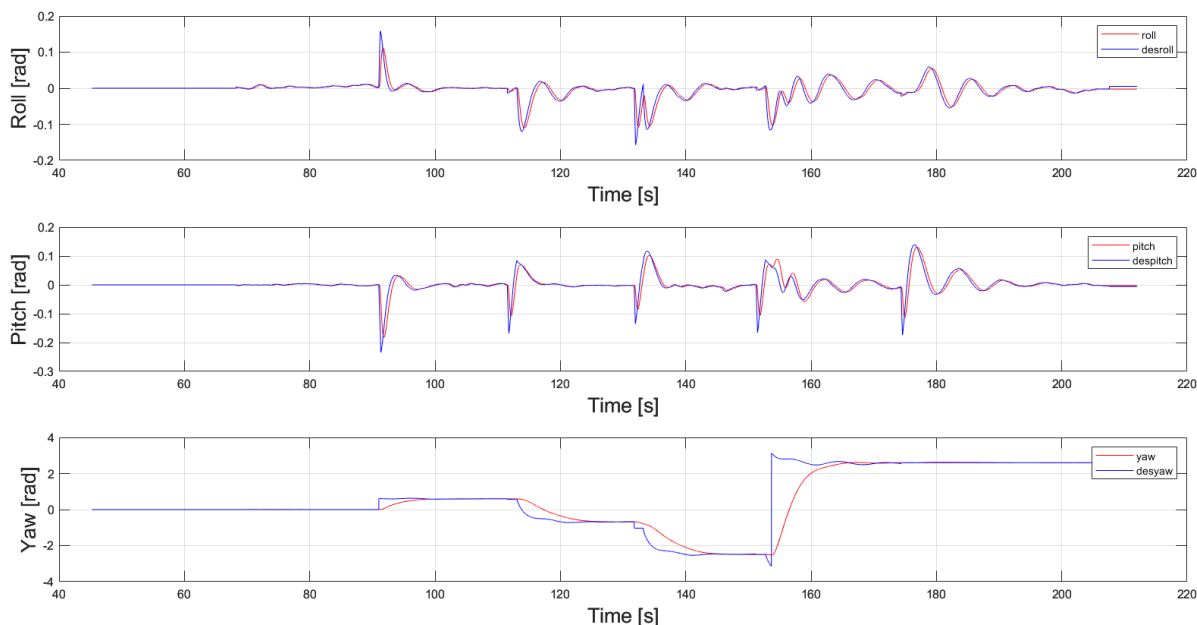


Figura 6.8: *Plotting* degli angoli nel tempo.

In Figura 6.8 vengono messe a confronto le misure degli angoli con i loro riferimenti. In particolare, come si è visto nell'analisi dei segnali di controllo, l'angolo ψ desiderato commuta da $-\pi$ a π istantaneamente: l'azione di controllo fa sì che ψ segua il riferimento ma facendo ripercorre l'angolo in senso contrario (orario) e questo è un effetto indesiderato. Considerando che anche gli angoli sono vettori colonna di 9307 elementi, definiamo la Tabella 6.3 che è del tutto analoga alla Tabella 6.2:

	e_φ	e_θ	e_ψ
μ	$-1.608e^{-04}$	$1.877e^{-05}$	-0.051
σ^2	$1.320e^{-04}$	$3.072e^{-04}$	0.427

Tabella 6.3: Tabella dei risultati per φ , θ e ψ

Come riporta la tabella, l'errore medio e la varianza dell'errore relativo all'angolo ψ sono molto più grandi rispetto agli altri due angoli: questo è dovuto sia perchè ψ è soggetto a rotazioni più importanti e che richiedono tempo, sia perchè si trova di fronte ad una commutazione (da $-\pi$ a π) che il controllore non riesce a gestire, per cui rimanda indietro la rotazione di ψ che con il *landing* si stabilizza.

6.3 Analisi del tempo di esecuzione

Dopo aver implementato il *setup* sperimentale HITL e averne verificato il funzionamento con una legge di controllo personalizzata basata sul modello matematico, è stato valutato il tempo di esecuzione del controllore tramite un algoritmo per il *fault detection* (FD). Come trattato

nell'articolo riportato in bibliografia [13], grazie al toolbox UAV di Matlab, l'algoritmo è stato costruito, compilato e distribuito sulla PixHawk del progetto durante una prova di volo in modalità HITL. Considerando che PixHawk 2.1 Cube (Black) è un hardware dotato con potenza di calcolo e memoria limitate, il calcolo delle funzionalità principali (*features*) e del filtraggio (*filtering*) è stato impossibile per ragioni legate alla complessità, sia temporale sia spaziale. D'altra parte, sono state apportate delle modifiche all'algoritmo che permettono di ottenere un sufficiente risparmio di memoria ed una riduzione del tempo computazionale: i buffer, ovvero gli stati dei filtri (FIR) sono stati implementati come buffer circolari. In Figura 6.9 è riportata l'implementazione effettiva del calcolo delle *features*.

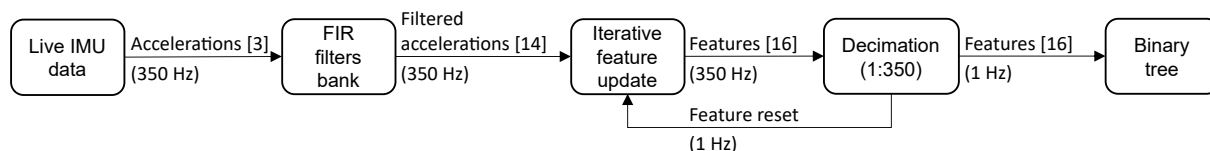


Figura 6.9: Schema per il calcolo delle *features*

Il test per il calcolo del tempo computazionale è stato effettuato implementando 70 filtri e 140 *features*: in Figura 6.10 è riportato il tempo sperimentale per calcolare il filtraggio ed estrarre le *features*.

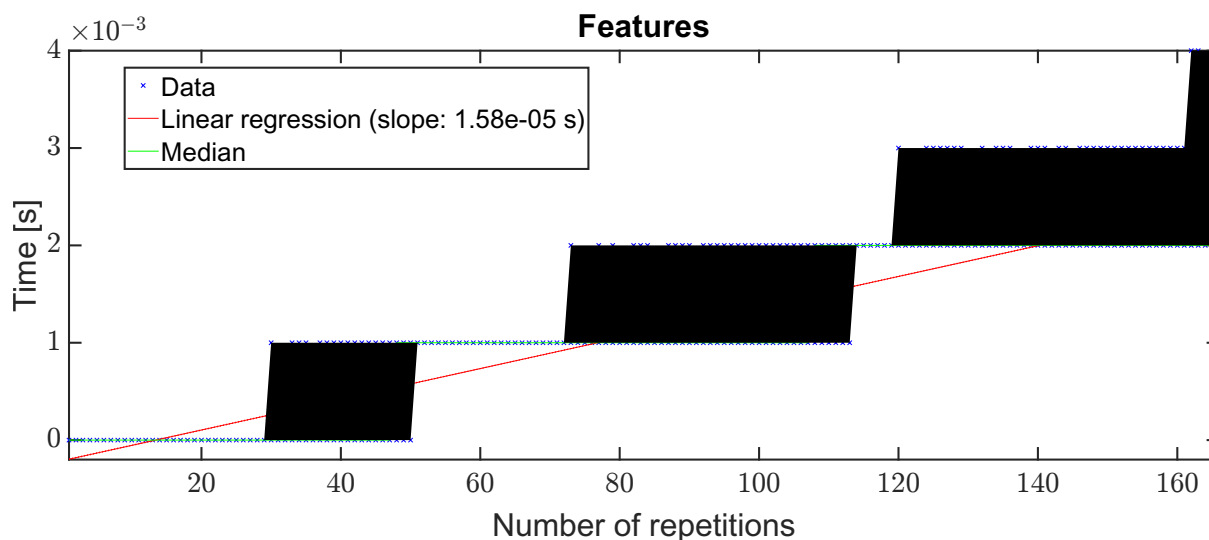


Figura 6.10: Tempo di esecuzione su PixHawk 2.1 Cube (Black) durante la prova in HITL

Nell'ascissa è riportato il numero di calcoli ripetuti (identici) di filtri e *features*: l'attività può essere ripetuta 28 volte nella stessa iterazione dello scheduler, senza raggiungere la risoluzione di un $1ms$ della libreria *time.h* di Matlab, mentre il tempo mediano è inferiore a $1ms$ fino a 48 ripetizioni.

In Figura 6.11 è riportato il tempo di computazione per il classificatore LSVM con 70 *features*.

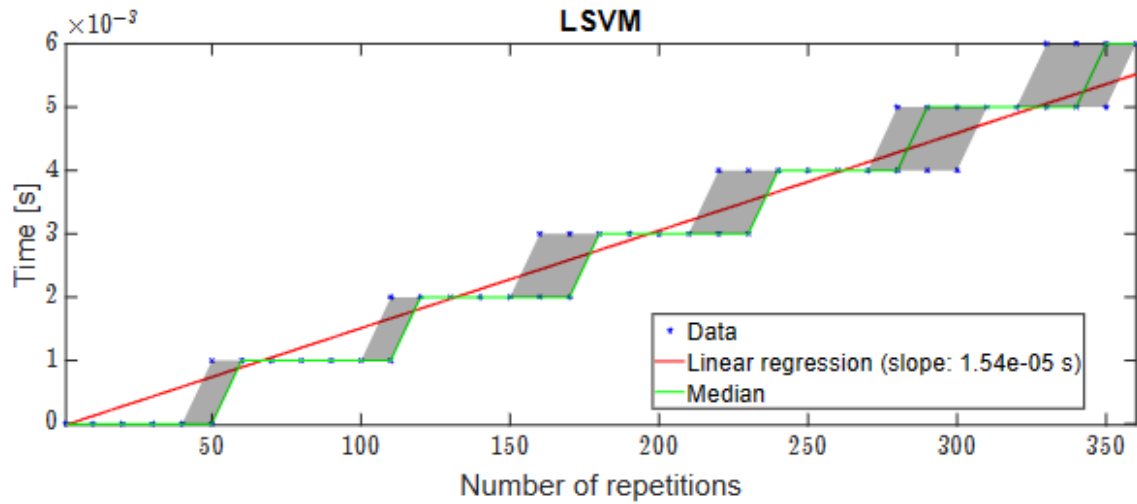


Figura 6.11: Classificazione durante la prova di volo in HITL

Ancora una volta, notiamo che l'attività può essere ripetuta 40 volte nella stessa iterazione dello scheduler senza raggiungere la risoluzione di 1 [ms] della libreria *time.h*, e si deduce che il tempo medio per eseguire il classificatore è di $15.4 \mu\text{s}$. Per fare il punto, nella tabella 6.4 riportiamo il tempo medio di esecuzione:

Features	# Filters	# Features	Task	HIL execution time Average
filters	70	140	Feature computation	0.0158 ms
filters	70	140	LSVM classification	0.0154 ms
filters	70	140	Total over 1 s window	5.55 ms

Tabella 6.4: Tempo medio di esecuzione durante la prova HITL

Capitolo 7

Conclusioni

In questa tesi abbiamo visto come sia possibile configurare e collegare un controllore PixHawk tramite pc, e come implementare e distribuire un algoritmo di controllo tramite il calcolatore Matlab su versioni di PixHawk compatibili con esso. I risultati della tecnica di controllo basata su *Feedback Linearization* hanno portato a risultati soddisfacenti per prove di volo mediante approccio HITL. Questo apre le porte alle opportunità sia di implementare tecniche di controllo robusto, sia di testare queste ultime per analizzarne i dati. Inoltre, evidenziamo i seguenti punti:

- PX4 mette a disposizione le informazioni necessarie per implementare una legge di controllo basata sul modello: avere a disposizione un modello almeno in Grey-Box¹ permette di ricorrere a tecniche di controllo più precise;
- è possibile scrivere righe di codice all'interno dei modelli messi a disposizione dal pacchetto PX4 di Matlab. In questo modo, per implementare ciò che occorre possiamo non adoperare la programmazione tramite blocchi Simulink. Ciò risulta un vantaggio in quanto tramite codice scritto da noi in Matlab conosciamo come avviene il trasferimento input-output e possiamo modificarlo o manipolarlo come riteniamo opportuno (controllo, regolazione, filtraggio, iniezione guasti, forzatura di un comportamento del sistema...);
- testare e impostare i parametri di controllo richiedono diverse prove di volo. La tecnica HITL qui ci permette di risparmiare tempo e di evitare guasti o danni concreti pur avendo a disposizione un controllore reale.

I principali problemi riscontrati sono i seguenti:

- il sistema operativo Windows 11 può creare problemi durante la sincronizzazione tra il Simulink, QGC e il simulatore jMAVSim, a seconda delle versioni. Alcune di esse non permettono l'avvio automatico dei software di volo e di simulazione, e bisogna a quel punto procedere manualmente oppure ricorrere ad un altro sistema operativo;
- come già detto nel Capitolo 6, durante le prove di volo il drone riscontra dei problemi nel punto di *switching* da $-\pi$ a π radianti. Un'idea potrebbe essere quella di imporre un segnale di controllo quando l'angolo ψ rientra in un range del tipo $-\pi + \epsilon < \psi < \pi - \epsilon$, con ϵ scelto opportunamente.

¹Un sistema Grey-Box è un sistema di cui conosciamo il modello, i segnali in uscita sono misurabili ma i parametri sono incerti.

Bibliografia

- [1] Architettura px4. https://docs.px4.io/main/en/concept/px4_systems_architecture.html. Accessed: 2023-05-12.
- [2] Documentazione ufficiale mathworks. https://it.mathworks.com/help/supportpkg/px4/index.html?s_tid=CRUX_lftnav. Accessed: 2023-05-12.
- [3] Documentazione ufficiale px4. <https://docs.px4.io/main/en/>. Accessed: 2023-05-12.
- [4] Immagine pixhawk 2.1 cube. <https://ardupilot.org/copter/docs/common-thecube-overview.html>. Accessed: 2023-05-12.
- [5] Immagine quadrirotore. <https://www.amazon.ca/Quadcopter-Real-Time-Transmission-Positioning-1-dp/B07NWNQ5KK?th=1>. Accessed: 2023-05-12.
- [6] Immagine(2) pixhawk 2.1 cube. https://docs.px4.io/main/en/flight_controller/pixhawk-2.html. Accessed: 2023-05-12.
- [7] Releases del firmware px4. <https://github.com/PX4/PX4-Autopilot/releases>. Accessed: 2023-05-12.
- [8] Stack px4. <https://docs.px4.io/main/en/concept/architecture.html>. Accessed: 2023-05-12.
- [9] Workflow per hitl in px4. <https://it.mathworks.com/help/supportpkg/px4/ug/mavlink-bridge-hitl.html>. Accessed: 2023-05-12.
- [10] Workflow per hitl in px4. <https://it.mathworks.com/help/supportpkg/px4/ug/px4-hitl-system-architecture.html>. Accessed: 2023-05-12.
- [11] A.Baldini, R. Felicetti, A. Freddi, S. Longhi, and A. Monteriù. *Direct position control of an octarotor unmanned vehicle under wind gust disturbance*. 2019.
- [12] A.Baldini, R.Felicetti, A.Freddi, S.Longhi, and A.Monteriù. *Dynamic Surface Control for Multirotor Vehicles*. 2018.
- [13] A.Baldini, R.Felicetti, F.Ferracuti, A.Freddi, S.Iarlori, and A.Monteriù. *Real-time propeller fault detection for multirotor drones based on vibration data analysis*. 2023.
- [14] A.Bonci. *UNMANNED AERIAL VEHICLES DYNAMICS - 1*. Department of Information Engineering, Ancona - Italy.
- [15] M. Acharya, R. George, and J. Antoniou. *Simulate and Deploy UAV Applications with SIL and HIL Workflows, Mathworks*. MATLAB EXPO.
- [16] A.Saibi, R. Boushaki, and H. Belaidi. *Backstepping Control of Drone*. 2022.

- [17] AZ-Delivery. *HW-598 USB auf Seriell Adapter mit CP2102 Chip Datenblatt*. 2015.
- [18] M. Bacic. *On hardware-in-the-loop simulation*. 2005.
- [19] E.Kuantama, I. Tarca, and R. Tarca. *Feedback Linearization LQR Control for Quadcopter Position Tracking*. 2018.
- [20] D. Erdos, A. Erdos, and S. E. Watkins. *An experimental UAV system for search and rescue challenge*. 2013.
- [21] G.Enriquez. *Design of Hardware in the Loop (HIL) Simulation for Small Unmanned Aerial Vehicles (sUAV)*. Aerospace Engineering Department Embry-Riddle Aeronautical University, 2015.
- [22] C. Ha, Z. Zuo, F. Choi, and D. Lee. *Passivity-based adaptive backstepping control of quadrotor type UAVs*. 2014.
- [23] Y. Jing, X. Wang, J. Heredia-Juesas, C. Fortner, C. Giacomo, R. Sipahi, and J. Martinez-Lorenzo. *PX4 Simulation Results of a Quadcopter with a Disturbance-Observer-Based and PSO-Optimized Sliding Mode Surface Controller*. 2022.
- [24] H. M. Kandeel, E. A. Abdelmaksod, and A. O. Elnady. *Modeling and Control of X-Shape Quadcopter*. 2022.
- [25] P. Liu, R. Ye, K. Shi, and B. Yan. *Full Backstepping Control in Dynamics Systems with Air Disturbances Optimal Estimation of a Quadrotor*. 2021.
- [26] M. Pantalone. *MODELLAZIONE E SIMULAZIONE DI UN QUADRICOTTERO MULTIROTORE*. scuola di ingegneria e architettura di Forlì, 2014-15.
- [27] E. Saif and I. Eminoglu. *Modelling of quad-rotor dynamics and Hardware-in-the-Loop simulation*. 2022.
- [28] S.Montemurro. *Quadcopter HOSM Control on PX4 Firmware Architecture*. SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING, Politecnico di Milano, 2020-21.
- [29] A. Stama. *Modellazione matematica e simulazione del sistema di controllo di un drone*. Department of Information Engineering, Ancona - Italy, 2018-19.
- [30] V.Gomez, N. Gomez, J. Rodas, E. Paiva, M. Saad, and R. Gregor. *Pareto Optimal PID Tuning for Px4-Based Unmanned Aerial Vehicles by Using a Multi-Objective Particle Swarm Optimization Algorithm*. 2020.
- [31] S. Wang, X. Dai, C. Ke, and Q. Quan. *RflySim: A Rapid Multicopter Development Platform for Education and Research Based on Pixhawk and MATLAB*. 2021.