

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Approccio Hardware-In-The-Loop per la simulazione di guasti e la validazione di moduli diagnostici in velivoli senza pilota

Hardware-In-The-Loop approach for fault simulation and validation of diagnostic modules in Unmanned Aerial Vehicles

Relatore:

PROF. FREDDI ALESSANDRO

Studente:

CARDONI LORENZO

ANNO ACCADEMICO 2023-2024

*Dedico questa tesi al me diciottenne
per aver deciso di intraprendere questo percorso.*

Indice

1	Introduzione	6
1.1	Hardware-In-The-Loop (HIL)	7
1.2	Definizioni	8
1.3	Confronto tra Pilotaggio Manuale e Volo Autonomo nel Contesto Diagnostico	10
1.4	Stato dell'Arte	10
1.5	Obiettivo Tesi	11
2	Requisiti Progetto	12
2.1	Architettura PX4	12
2.2	Integrazione con l'architettura PX4	13
2.3	Software di progetto	17
2.4	Hardware di progetto	19
2.5	Workflow Hardwar-in-the-Loop	20
3	Modello Matlab/Simulink	22
3.1	Flight Controller	22
3.2	UAV Dynamics Model	24
3.2.1	Blocco Simulatore IMU	26
3.2.2	Blocco Simulatore GPS	28
3.2.3	Struttura del Test	30
3.2.4	Blocco simulatore barometro	31
3.3	Log dei Segnali	31
3.3.1	Log dei Segnali dal Flight Controller	32
3.3.2	Log dei Segnali dal Modello di Dinamica UAV	32
3.4	Post-Simulazione e Visualizzazione dei Dati	33
4	Guasti UAV	35
4.1	Guasti negli Attuatori	36
4.1.1	Lock-in-Place	37
4.1.2	Float Failure	37
4.1.3	Hardover	38
4.1.4	Loss of Effectiveness	38
4.2	Guasti nei Sensori	39
4.2.1	Sensor Bias	39
4.2.2	Sensor Drift	40
4.2.3	Loss of Accuracy	40
4.2.4	Freezing	41
4.2.5	Calibration Error	41
4.3	Classificazione dei guasti per la loro implementazione	42
4.3.1	Tipologie di Guasti	42
4.3.2	Dinamiche dei Guasti	42

4.3.3	Tabella di Classificazione dei Guasti	43
5	Modulo di Generazione Guasti	44
5.1	Struttura dei Moduli	44
5.1.1	Moduli Fault Injection IMU	45
5.1.2	Modulo Fault Injection GPS	49
5.1.3	Modulo Fault Injection Attuatori	51
5.1.4	Interfaccia Principale del Fault Injection	55
5.2	Validazione del Modulo	56
6	Modulo Diagnostico	62
6.1	Guasti da Diagnosticare	62
6.1.1	Simulazione dei Voli	63
6.1.2	Raccolta e Bilanciamento del Dataset	63
6.2	ETL (Extract, Transform, Load)	63
6.2.1	Sincronizzazione dei Dati	64
6.2.2	Selezione dei Dati Utili	64
6.2.3	Creazione della Tabella Finale	65
6.2.4	Visualizzazione dei dati	65
6.3	Design ed Implementazione di un Modello di Machine Learning su Calcolatore	67
6.3.1	Estrazione delle Feature	67
6.3.2	Classificazione e Selezione delle Feature	68
6.3.3	Addestramento del Classificatore	68
6.3.4	Implementazione del Classificatore	69
6.4	Design ed Implementazione di un Modello di Machine Learning su Dispositivo Microcontrollore	72
6.4.1	Implementazione del Classificatore su Microcontrollore	74
6.4.2	Validazione in Volo e Risultati su Flight Controller	74
7	Conclusioni e Sviluppi Futuri	76
A	Configurazione dell'Ambiente di Sviluppo il Sistema PX4 Hardware-in-the-Loop (HIL)	78
A.1	Configurazione Toolchain Cygwin e Download Codice Sorgente PX4	78
A.2	Configurazione Firmware PX4 per l'Hardware-in-the-Loop	80
A.3	Configurazione QGroundControl	88
A.4	PX4 HIL Simulation with UAV Dynamics in Simulink	89
A.5	Avvio Simulazione HIL con Monitor & Tune	90
A.5.1	Collegamento Hardware	90
A.5.2	Configurazione del Modello del Controller PX4 in Simulink per Monitor & Tune	91
A.5.3	Avviare Monitor & Tune per il Flight Controller PX4, Run del UAV Dynamics model, Upload Mission in QGroundControl e volo dell'UAV	92
Bibliografia		95

Sommario

Il presente lavoro di tesi ha l'obiettivo di sviluppare un sistema di diagnosi dei guasti per UAV (*Unmanned Aerial Vehicles*), con particolare attenzione all'implementazione di algoritmi di machine learning sia nel simulatore della dinamica del drone che sui microcontrollori. L'obiettivo principale è garantire l'affidabilità e la sicurezza dei droni in scenari operativi critici, riducendo i rischi di malfunzionamenti attraverso l'uso di moduli diagnostici avanzati.

L'approccio adottato prevede l'impiego di una metodologia di simulazione *Hardware-In-The-Loop* (HIL) per la raccolta dati e la validazione dei modelli diagnostici. In questo contesto, è stato sviluppato un toolbox per la *Fault Injection*, che consente di simulare in modo realistico guasti nei sensori di bordo, come accelerometri, giroscopi, magnetometri e GPS, e generare un dataset utile per l'addestramento dei modelli di classificazione.

La pipeline di sviluppo ha seguito un processo di *Extract, Transform, Load* (ETL) per la preparazione dei dati, comprendendo il calcolo di errori di posizione, l'estrazione e la selezione delle feature più rilevanti tramite test statistici. Un aspetto chiave è stato l'adeguamento della frequenza di campionamento tra simulazione e dispositivo reale.

Due implementazioni distinte del classificatore sono state sviluppate su Calcolatore utilizzato per la simulazione della dinamica del drone e su Microcontrollore.

Per migliorare la robustezza della diagnosi e ridurre i falsi positivi, sono stati implementati buffer circolari e tecniche di filtraggio delle predizioni basate su finestre temporali.

I risultati ottenuti dimostrano la fattibilità dell'implementazione di modelli diagnostici a bordo degli UAV, garantendo un buon compromesso tra accuratezza, prestazioni computazionali e capacità di rilevazione dei guasti. Questo approccio contribuisce allo sviluppo di sistemi autonomi più sicuri e affidabili.

Il presente documento è così strutturato:

- **Capitolo 1:** Introduzione ai concetti chiave della diagnosi dei guasti negli UAV e motivazioni del lavoro.
- **Capitolo 2:** Definizione dei requisiti di progetto hardware e software.
- **Capitolo 3:** Descrizione dei modelli MATLAB & Simulink, sia per la simulazione della dinamica che per il Flight Controller.
- **Capitolo 4:** Analisi dei guasti presenti in letteratura per gli UAV.
- **Capitolo 5:** Implementazione e valutazione del toolbox per la *Fault Injection*.
- **Capitolo 6:** Sviluppo e implementazione dei moduli diagnostici, sia per il calcolatore della simulazione della dinamica del drone che per il microcontrollore, con analisi delle prestazioni.
- **Capitolo 7:** Conclusioni e sviluppi futuri per il miglioramento del sistema diagnostico.

Capitolo 1

Introduzione

L’evoluzione delle tecnologie per la progettazione e il controllo di sistemi autonomi, come i droni, ha aperto nuove prospettive in una vasta gamma di settori, tra cui logistica, monitoraggio ambientale, ispezioni industriali e sicurezza. Tuttavia, l’affidabilità e la sicurezza di questi sistemi sono fondamentali per la loro applicazione in contesti critici. In particolare, la capacità di rilevare, diagnosticare e gestire guasti durante il funzionamento rappresenta una sfida cruciale per garantire prestazioni ottimali e prevenire incidenti che potrebbero comportare rischi per persone, strutture o l’ambiente.

Gli UAV (*Unmanned Aerial Vehicles*) rappresentano sistemi complessi in cui sensori e attuatori giocano un ruolo fondamentale per garantire la stabilità. Tuttavia, la natura stessa di questi componenti li rende vulnerabili a diverse tipologie di guasto, che possono compromettere il funzionamento dell’intero sistema. Per rendere i sistemi affidabili e robusti ai guasti, è fondamentale la progettazione di moduli diagnostici che permettono di:

- **Prevenire malfunzionamenti critici:** rilevando tempestivamente anomalie, è possibile intervenire per evitare danni irreparabili o incidenti.
- **Ridurre i costi di manutenzione:** l’identificazione precisa di guasti consente interventi mirati, riducendo il tempo e le risorse necessarie per la manutenzione.
- **Garantire la sicurezza operativa:** in applicazioni sensibili, come il trasporto di merci o il monitoraggio in ambienti urbani, è fondamentale mantenere elevati standard di sicurezza.
- **Ottimizzare le prestazioni del sistema:** un UAV che opera in condizioni ottimali garantisce efficienza, maggiore autonomia e migliori risultati nelle missioni.

La simulazione svolge un ruolo fondamentale nello sviluppo e nella validazione di algoritmi di diagnosi di guasti per sistemi autonomi come gli UAV. Essa consente di testare diverse condizioni operative e scenari di guasto senza dover ricorrere a test sperimentali sul campo, che risultano spesso costosi, complessi e potenzialmente pericolosi. Grazie alla simulazione, è possibile generare un ampio dataset contenente dati sia nominali che guasti, utile per approcci *data-driven* o *model-based*. Inoltre, permette di valutare l’efficacia di un sistema diagnostico in condizioni controllate, riducendo il rischio di guasti imprevisti in applicazioni reali.

Tuttavia, una delle principali sfide della simulazione è garantire un elevato livello di realismo, in modo da replicare fedelmente il comportamento del sistema fisico e l’influenza di disturbi e guasti reali. Per affrontare questa problematica, si fa spesso ricorso alla metodologia *Hardware-In-The-Loop* (HIL).

Di seguito verranno fornite le definizioni relative agli UAV (*Unmanned Aerial Vehicles*) e alla metodologia di lavoro HIL (*Hardware-In-The-Loop*), viene discusso lo stato dell’arte, e infine delineati gli obiettivi della presente tesi.

1.1 Hardware-In-The-Loop (HIL)

L'HIL è una tecnica che consente di integrare componenti fisici reali con un ambiente simulato, creando un banco di prova in cui è possibile testare algoritmi e moduli diagnostici in condizioni vicine a quelle operative. L'approccio HIL risulta particolarmente utile in contesti in cui il test diretto sul sistema fisico reale è impraticabile, costoso o pericoloso. Di conseguenza, questa metodologia trova applicazione in diversi settori chiave:

- Aerospace e difesa: utilizzato per simulare e controllare la dinamica di volo di velivoli, in scenari dove i test su mezzi reali possono essere troppo complessi o rischiosi.
- Automotive: impiegato per analizzare la dinamica e il controllo di veicoli, evitando la necessità di test continui su strada che possono essere costosi e logisticamente impegnativi.
- Automazione industriale: permette di verificare algoritmi di controllo su impianti produttivi senza interrompere la produzione o il funzionamento di una linea di assemblaggio, riducendo al minimo il rischio di perdite economiche.

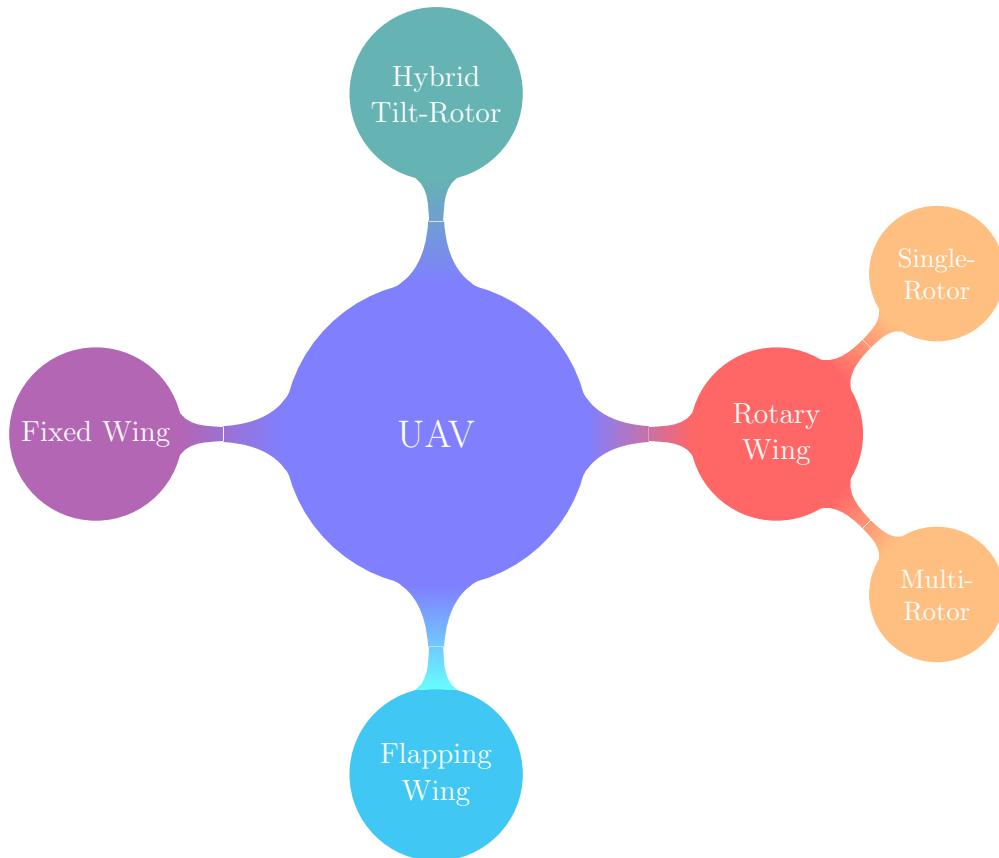
Nel contesto della diagnosi guasti per UAV, l'HIL permette di interfacciare il firmware di controllo con una simulazione accurata della dinamica del drone, dei sensori e degli attuatori. Ciò consente di valutare la capacità del sistema diagnostico di rilevare e classificare guasti in tempo reale, senza il rischio di compromettere un UAV fisico. Inoltre, l'HIL facilita l'ottimizzazione e la calibrazione dei modelli di diagnosi prima della loro implementazione a bordo del veicolo reale. Grazie all'HIL è possibile:

- Creare e simulare una rappresentazione virtuale dei componenti fisici, consentendo un'analisi dettagliata e controllata delle loro interazioni;
- Eseguire algoritmi di controllo e di monitoraggio su un simulatore, che interagisce direttamente con l'hardware fisico attraverso canali di ingresso/uscita (I/O).

Con queste caratteristiche, l'impiego della simulazione e dell'HIL risulta essenziale per lo sviluppo di sistemi diagnostici affidabili, permettendo di accelerare il processo di progettazione, ridurre i costi di test e aumentare la sicurezza delle operazioni.

1.2 Definizioni

Gli UAV possono essere classificati principalmente in diverse categorie. Queste tipologie presentano differenze significative in termini di progettazione, funzionamento e ambiti di utilizzo. Di seguito si riportano le tipologie di UAV più popolari, nella presente tesi, l'attenzione sarà focalizzata sui droni ad ala rotante.



- **Fixed Wing UAV:** Caratterizzati da una struttura simile a quella degli aeroplani, con ali rigide che forniscono portanza grazie al movimento in avanti.

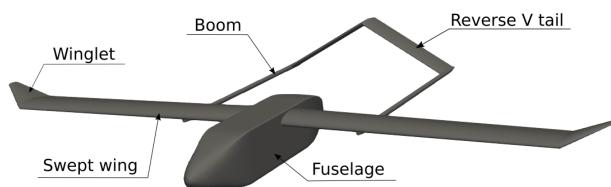


Figura 1.1: Fixed Wing UAV [5]

- **Rotary Wing UAV:** Includono i multirotori (come quadricotteri e esacotteri), che utilizzano eliche rotanti per generare portanza e manovrare in tutte le direzioni. Sono più manovribili e possono decollare e atterrare verticalmente, rendendoli adatti per applicazioni in spazi limitati o ambienti urbani.

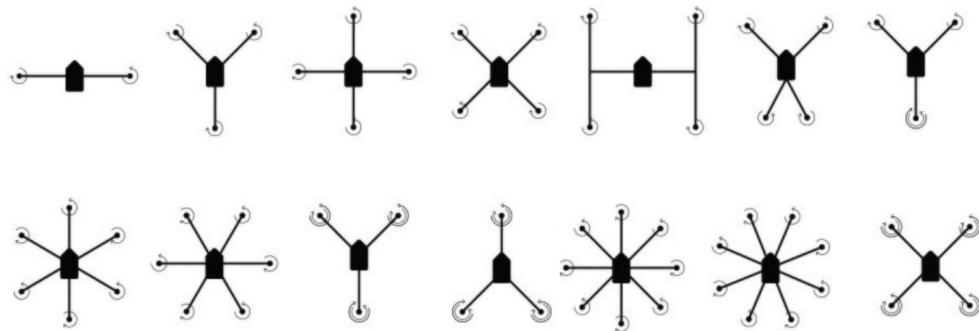


Figura 1.2: Tipi di disposizioni delle eliche multirotore. Nella riga superiore ci sono, in ordine da sinistra: bicottero, tricottero, quad +, quad X, quad H, quad V, quad Y. Nella riga inferiore ci sono, in ordine da sinistra: hexa +, hexa X, hexa Y6, hexa IY, octo +, octo X, octo X8.

- **Tilt-Rotor UAV:** Combinano le caratteristiche dei droni ad ala fissa e ad ala rotante. I loro rotori possono inclinarsi per consentire sia il decollo e atterraggio verticale sia il volo orizzontale ad alta efficienza.

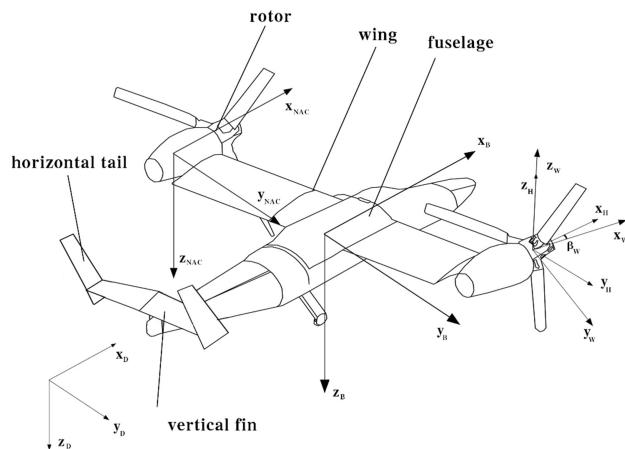


Figura 1.3: Tilt-rotor UAV [13]

- **Flapping Wing UAV:** Ispirati al volo degli uccelli e degli insetti, utilizzano ali battenti per generare portanza e spinta.

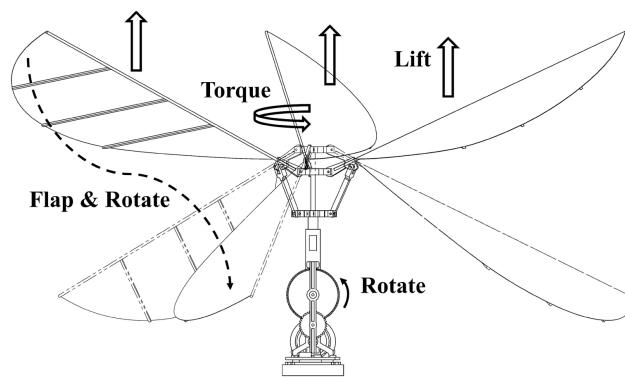


Figura 1.4: Flapping Wing UAV [6]

1.3 Confronto tra Pilotaggio Manuale e Volo Autonomo nel Contesto Diagnostico

Nel contesto dell'uso degli UAV, è possibile distinguere due principali modalità operative: il **pilotaggio manuale** e il **volo autonomo**. Questa distinzione ha un impatto significativo sulla progettazione e sull'implementazione dei sistemi di diagnosi guasti, in quanto il grado di automazione influisce sulle strategie di rilevamento e mitigazione delle anomalie.

Nel **pilotaggio manuale**, il controllo dell'UAV è affidato direttamente a un operatore umano, il quale prende decisioni in tempo reale basandosi sul feedback visivo e strumentale. In questa modalità, la diagnosi guasti ha un ruolo di supporto, fornendo all'operatore informazioni sulle condizioni del drone per facilitare eventuali interventi correttivi. Tuttavia, la reazione a un guasto dipende interamente dalla prontezza e dall'esperienza del pilota, il che può comportare tempi di risposta variabili e un margine di errore più elevato. Inoltre, eventuali segnali di guasto potrebbero essere mascherati dalle manovre del pilota, rendendo più complessa l'identificazione automatica dei problemi.

Nel **volo autonomo**, invece, il drone esegue missioni predefinite seguendo algoritmi di navigazione e controllo senza necessità di intervento umano diretto. In questo scenario, la diagnosi guasti assume un ruolo critico, poiché il sistema deve essere in grado di rilevare, classificare e, se possibile, compensare eventuali anomalie in tempo reale. Un guasto non rilevato potrebbe compromettere l'intera missione e, nei casi più gravi, causare la perdita del velivolo. Per questo motivo, nei droni autonomi vengono implementati sistemi di *Fault Detection and Isolation* (FDI) che, oltre alla semplice identificazione del guasto, possono attivare strategie di *Fault Tolerant Control* (FTC) per mitigare gli effetti dell'anomalia.

Dal punto di vista sperimentale, il testing e la validazione dei moduli diagnostici differiscono tra le due modalità. Nel pilotaggio manuale, i guasti vengono generalmente analizzati in scenari controllati con l'intervento di un operatore, mentre nel volo autonomo si rende necessaria una simulazione più approfondita, come quella offerta dall'HIL, per valutare il comportamento del sistema diagnostico in condizioni realistiche. Inoltre, nel volo autonomo è spesso richiesta una maggiore integrazione tra il modulo diagnostico e il sistema di controllo, in modo da attuare eventuali contromisure in maniera automatizzata.

1.4 Stato dell'Arte

La rilevazione e gestione dei guasti nei sistemi UAV (*Unmanned Aerial Vehicles*) è un'area di ricerca cruciale che ha visto notevoli progressi nell'ultimo decennio. Questa sezione fornisce una panoramica dei principali contributi scientifici che hanno influenzato il presente lavoro.

I singoli guasti degli UAV sono stati approfonditi nel libro di Halim Alwi, Christopher Edwards, e Chee Pin Tan, *Fault Detection and Fault-Tolerant Control using Sliding Modes* (Springer, 2011) [1]. Questo libro fornisce una descrizione dettagliata di ciascun guasto, considerando in particolare gli UAV ad ala fissa, e riporta esempi reali di incidenti aerei.

Il capitolo *Failure Detection, Identification, and Reconfiguration in Flight Control* di Jovan D. Bošković e Raman K. Mehra, contenuto nel volume *Springer Berlin Heidelberg* (2003) [4], tratta i modelli matematici dei guasti e le tecniche di reconfigurazione dei sistemi di controllo in volo. Questo contributo è stato fondamentale per sviluppare i modelli di guasto utilizzati nella presente tesi.

L'articolo di Daoliang Li et al. (*Recent Advances in Sensor Fault Diagnosis: A Review*, Sensors and Actuators A: Physical, 2020) [7] fornisce una revisione approfondita degli ultimi sviluppi nella diagnosi dei guasti dei sensori, evidenziando l'importanza di tecniche *data-driven* per la rilevazione precoce e la gestione dei guasti.

La review "UAV Fault Detection Methods, State-of-the-Art" [12], ha fornito un'ampia panoramica su tutti i guasti presenti in letteratura, sui tipi di UAV e sulle metodologie di identificazione dei guasti più diffuse. Le metodologie di identificazione dei guasti sono principalmente di due tipi: *model-based* e *data-driven*. Quest'ultima ha acquisito crescente popolarità nell'ultimo decennio, però la metodologia model-based rimane ancora quella più utilizzata in ambito UAV. Viene anche riportato che i guasti più trattati riguardano gli attuatori, rispetto ai sensori, e che non c'è molta differenza tra gli UAV ad ala fissa o ala rotante.

1.5 Obiettivo Tesi

L'obiettivo di questa tesi è lo sviluppo di un toolbox avanzato in Matlab SIMULINK per l'iniezione e la simulazione di tutti i principali guasti documentati in letteratura relativi agli UAV. I guasti saranno inizialmente rappresentati mediante modelli matematici accurati, che permettano di descrivere in dettaglio il loro comportamento e l'impatto sul sistema. I modelli matematici saranno integrati in simulazioni *Hardware-In-The-Loop* (HIL), consentendo di replicare condizioni di volo realistiche e verificare l'effetto dei guasti in scenari operativi. Saranno sviluppati algoritmi dedicati per introdurre in modo controllato i guasti nei sensori e negli attuatori, garantendo una gestione precisa del processo di iniezione. Una volta inseriti i guasti, sarà verificata la coerenza dei comportamenti simulati con quanto atteso. Per valutare l'efficacia delle simulazioni, sarà implementato un modello di *machine learning* avanzato per la classificazione e la diagnosi automatica di uno o più guasti. Questo approccio consentirà di analizzare i dati ottenuti dalla simulazione, identificare le tipologie di guasti e migliorare la capacità di rilevamento.

Il risultato finale sarà un toolbox completo e versatile, in grado di supportare la ricerca e lo sviluppo di strategie per la diagnosi e la mitigazione dei guasti negli UAV, contribuendo a migliorare la sicurezza e l'affidabilità di questi sistemi.

Capitolo 2

Requisiti Progetto

In questo Capitolo si andrà ad approfondire la documentazione del "UAV Toolbox Support Package for PX4 Autopilots" fornita dalla MathWorks.

2.1 Architettura PX4

PX4 rappresenta un software open source specializzato nel controllo di veicoli unmanned, offrendo ai programmati una vasta gamma di strumenti e un supporto completo sia a livello software che hardware. Nella Figura 2.1, si presenta un quadro panoramico ad alto livello del controllore di volo implementato in PX4.

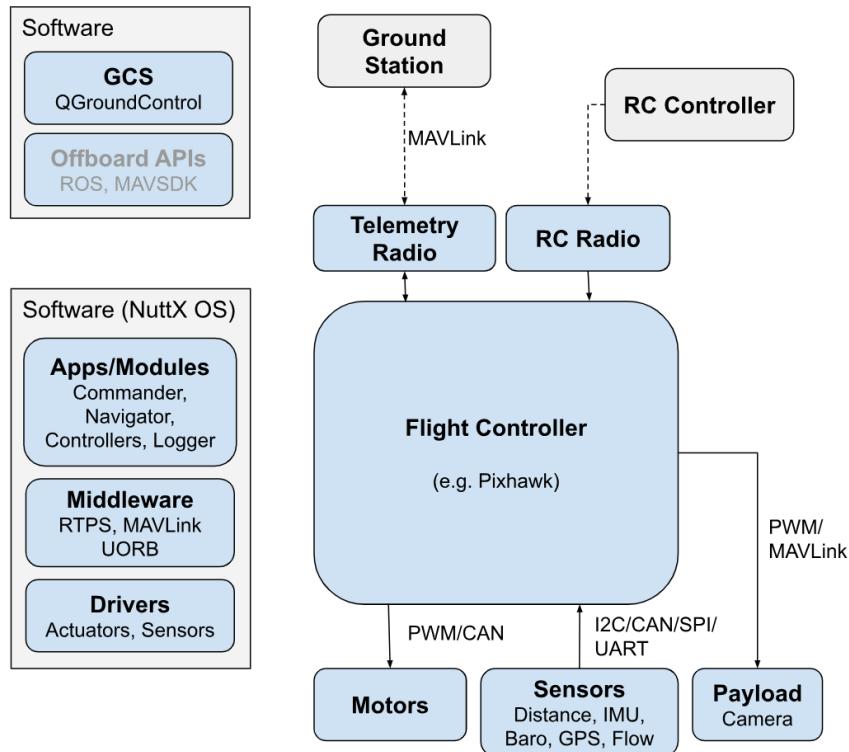


Figura 2.1: Architettura di alto livello del controllore di volo in PX4 [3]

Dal lato hardware, la configurazione comprende diversi componenti chiave:

- **Controllore di volo (PixHawk):** Esegue lo stack PX4, fornendo un'interfaccia fondamentale per la gestione complessiva del drone.
- **ESC (Electronic Speed Control):** Questi circuiti regolano la velocità e la direzione di rotazione dei motori, collegandosi alle uscite PWM del controller.

- **Sensori:** Connessi al controller tramite bus I2C o UART, forniscono dati essenziali per la navigazione e il controllo.
- **Fotocamera:** Collegata tramite canale PWM o protocollo MAVLink, consente la raccolta di dati visivi e può essere integrata nelle attività di controllo del drone.
- **RC Controller:** Necessario per il controllo manuale remoto attraverso un Joypad.

Sul fronte software, la configurazione include:

- **QGroundControl:** Utilizzato come stazione di controllo da un computer host, fornisce un'interfaccia utente per monitorare e comandare il drone.
- **Stack di volo PX4:** Eseguito sul controllore di volo, comprende moduli di comunicazione, driver e middleware. La sua architettura di alto livello è rappresentata nella Figura 2.2 ed in seguito verrà spiegata nel dettaglio.

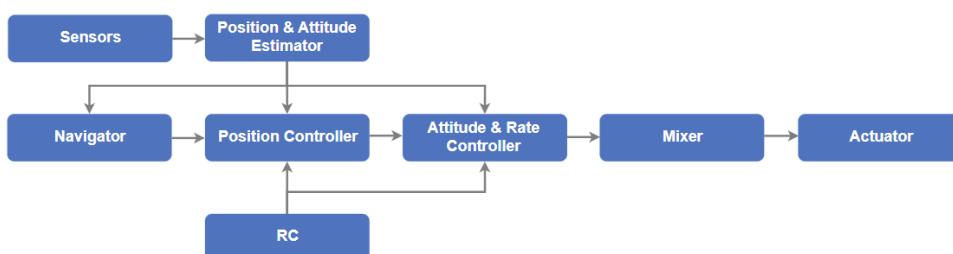


Figura 2.2: Architettura di alto livello del controllore di volo in PX4 [2]

2.2 Integrazione con l'architettura PX4

Il "UAV Toolbox Support Package for PX4 Autopilots" consente di progettare controllori, stimatori e navigatori in Simulink e di implementarli nelle schede PX4 Autopilot. È possibile integrare il codice generato dai modelli Simulink con lo stack di volo PX4 e quindi integrarli nelle schede PX4 Autopilot.

L'architettura software di alto livello del PX4 comprende moduli per l'archiviazione, la connettività esterna, i driver, il bus di messaggi *publish-subscribe uORB* e i componenti dello stack di volo, mostrata in Figura 2.3.

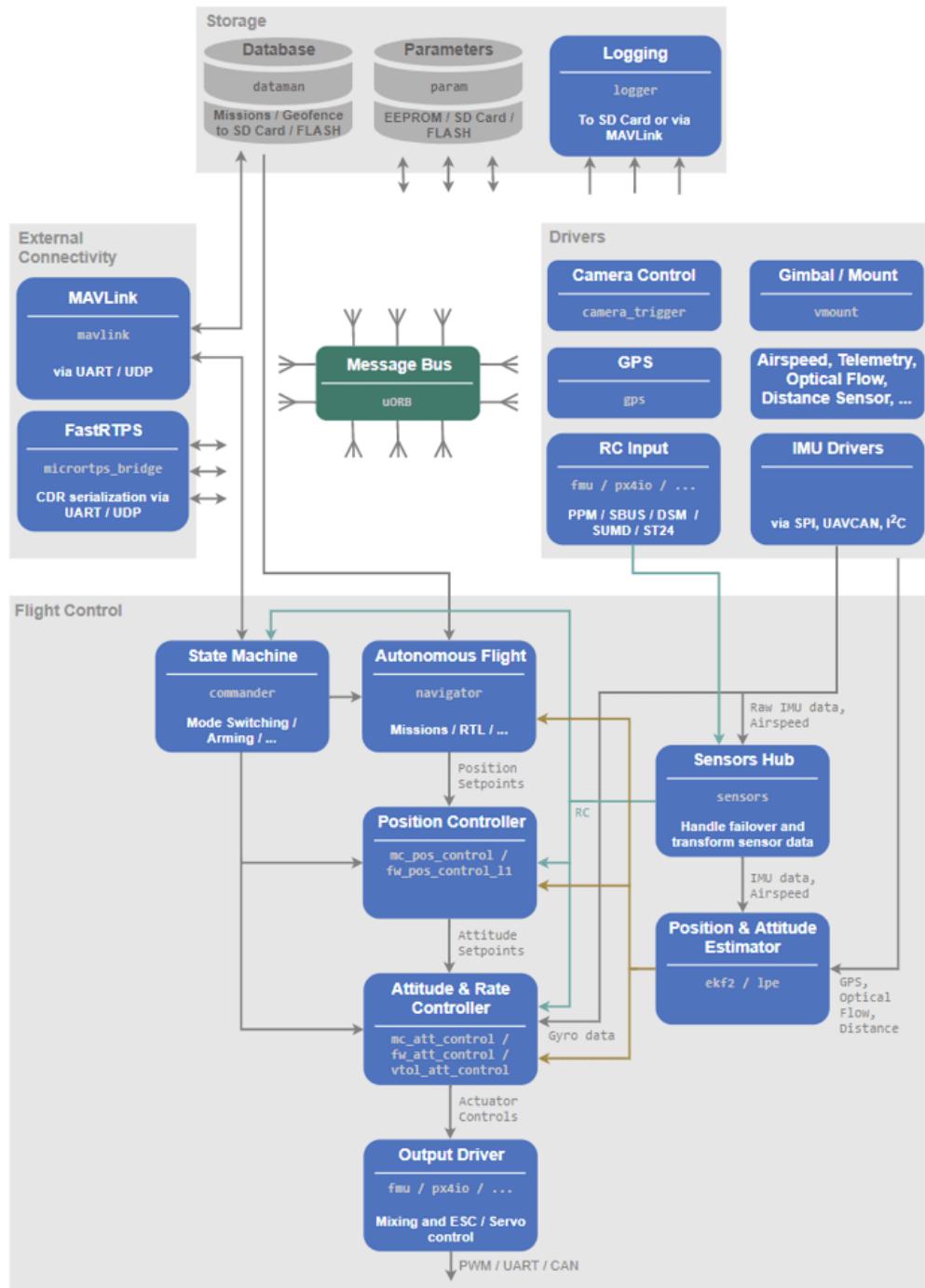


Image courtesy: <https://dev.px4.io/v1.9.0/en/concept/architecture.html>

Figura 2.3: Architettura di alto livello del controllore di volo in PX4 [2]

In modo più dettagliato, lo stack di componenti principali comprende:

- **Commander:** Questo modulo detiene informazioni cruciali sullo stato del drone e sulle modalità di volo, incluso il sistema failsafe, che entra in azione in situazioni di emergenza.
- **Navigator:** Il Navigator contiene i setpoint di volo fondamentali, quali punti di lancio, decollo e waypoints. Questi punti sono essenziali per stabilire la traiettoria di volo del drone.
- **Mc_controller:** Questo modulo è suddiviso in due componenti principali:

- **Position controller:** Responsabile del controllo della posizione del drone.
- **Attitude_and_rate controller:** Gestisce la dinamica dell'assetto del drone.
- **Mixer:** Questo modulo funge da intermediario tra i segnali di controllo generati dai moduli del Mc_Controller e i valori effettivi inviati agli attuatori del drone.
- **Sensors Hub:** Questo è il nucleo dei segnali rilevati dai sensori del drone. I dati di basso livello, provenienti dai sensori, sono elaborati dai driver e resi disponibili per il controller.
- **Ekf2 (Extended Kalman Filter):** Utilizza un filtro di Kalman esteso per stimare la posizione e gli angoli del drone, fornendo così una stima precisa della sua posizione e orientamento.
- **Drivers:** Questi moduli costituiscono l'interfaccia tra i sensori fisici del drone e l'architettura del firmware, consentendo al sistema di acquisire e utilizzare i dati provenienti dai sensori.
- **Mavlink:** Implementa il protocollo MAVLink, che funge da middleware di comunicazione. Tramite questo protocollo, il sistema invia messaggi specifici, come ad esempio l'assetto degli angoli del drone, utilizzando il bus uORB.
- **Simulatore:** Si tratta di un simulatore del drone in ambiente tridimensionale, che si interfaccia con il firmware attraverso il protocollo MAVLink. Questo simulatore è utile per testare il comportamento del drone in condizioni controllate prima del volo effettivo. In questo progetto è stato usato il Visualizzatore 3D Unreal Engine presente in Simulink nel UAV Toolbox.

Alcuni dei moduli e delle interfacce dell'architettura generale di PX4 possono essere integrati con il "UAV Toolbox Support Package for PX4 Autopilots", come mostrato in Figura 2.4

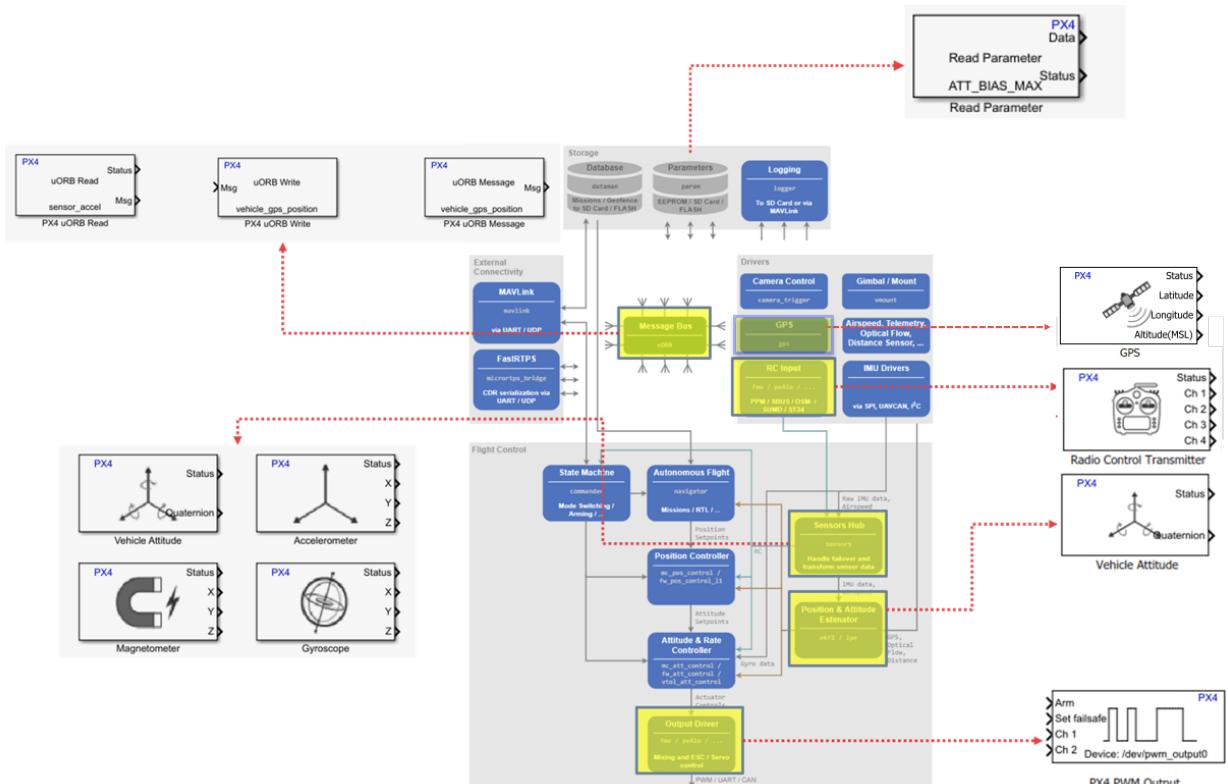


Figura 2.4: Blocchi Simulink Supportati che si interfacciano con i Moduli PX4 [10]

Inoltre, è possibile sostituire i moduli "Position Controller e Attitude & Rate Controller" presenti nell'architettura generale del PX4 con algoritmi di controllo definiti dall'utente e sviluppati utilizzando il "UAV Toolbox Support Package for PX4 Autopilots", come mostrato in Figura 2.5.

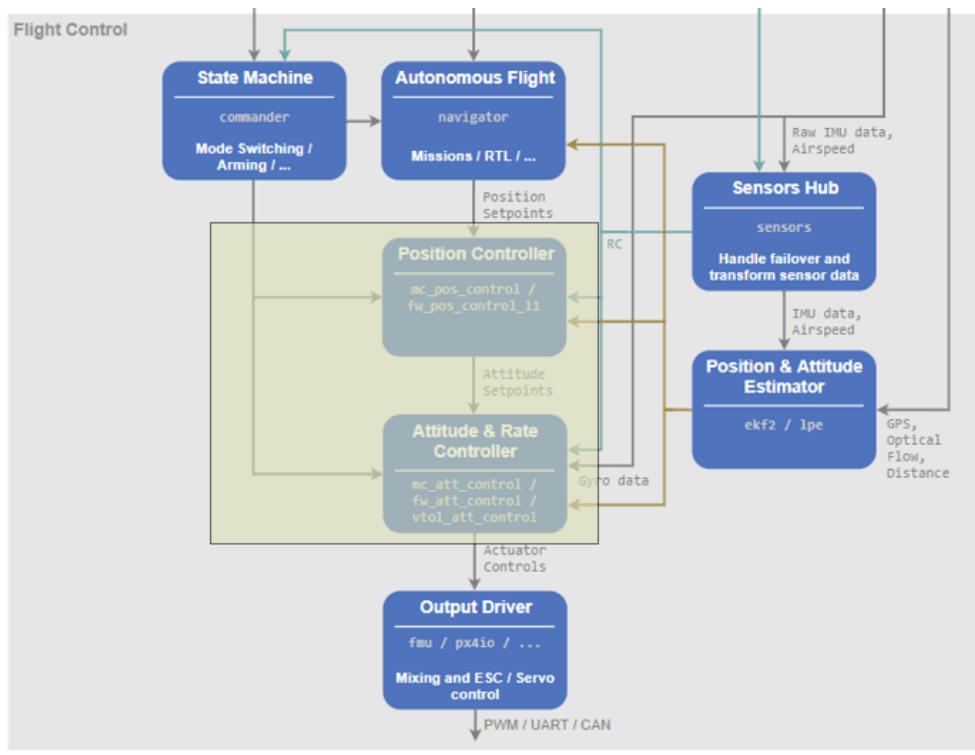


Figura 2.5: Moduli supportati che possono essere sostituiti con algoritmi definiti dall'utente [10]

2.3 Software di progetto

Per il lato software si va a lavorare su sistema operativo Microsoft Windows 11. 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 2.6 è riportata la home principale del software

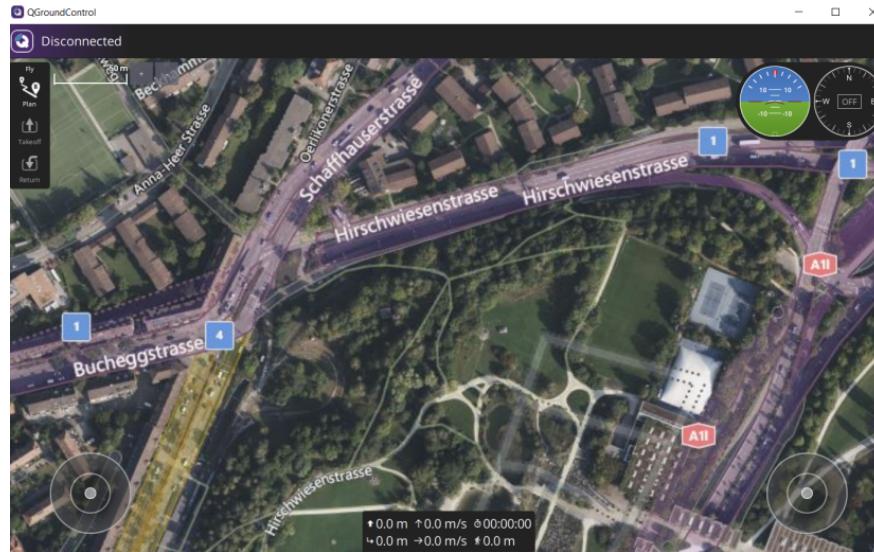


Figura 2.6: Home del software QGroundControl

Da qui è possibile accedere a quattro sezioni fondamentali:

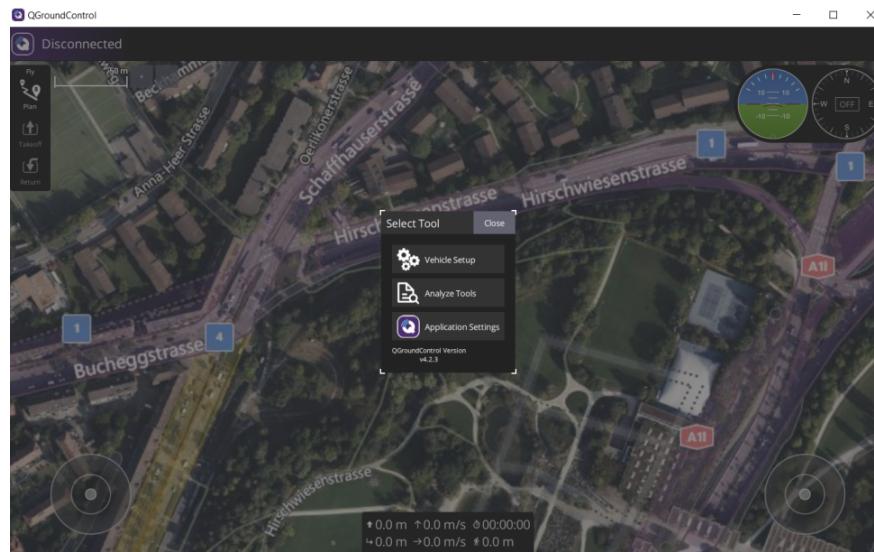


Figura 2.7: Sezioni principali del software QGroundControl

- **Fly Plan:** Questa funzionalità consente di pianificare il volo di un veicolo a pilotaggio remoto (UAV). La pianificazione si basa sulla definizione di tre elementi chiave:
 - * **Punto di Launch:** Il punto di partenza dal quale il veicolo prende quota prima di iniziare il volo.

- * **Takeoff:** I punti sulla mappa da cui il veicolo deve avviare la sua missione di volo.
 - * **Waypoints:** I punti che determinano variazioni nella traiettoria durante il volo. Questi punti guidano il veicolo passo dopo passo lungo il percorso, dal momento del decollo fino alla destinazione finale.
 - **Vehicle Setup:** Questa sezione è dedicata alla preparazione del veicolo. Incluse l'upload del Firmware, la configurazione dei parametri per la missione e la definizione del telaio (airframe) del veicolo, adattandolo alle esigenze specifiche dell'utente.
 - **Analyze Tools:** Questa sezione fornisce strumenti per l'analisi dei voli. Include il download dei file di volo .tlog e il MAVLink Inspector, una finestra attraverso la quale è possibile leggere i messaggi MAVLink (come dati dal controllore e dai sensori).
 - **Application Settings:** Questa sezione è dedicata alle impostazioni dell'applicazione. Qui è possibile configurare la comunicazione con il controllore fisico, selezionare le unità di misura per le grandezze fisiche e gestire i file di volo, consentendo all'utente di personalizzare l'esperienza di volo secondo le proprie preferenze.
- **PX4.Windows.Cygwin.Toolchain.0.8.msi:** è l'unico software ufficialmente supportato per il download del codice sorgente PX4 su Windows;
 - **Matlab (versione R2023b):** l'ambiente di calcolo fornito da MathWorks su cui verrà costruito l'algoritmo di rilevamento guasti 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.
 - **Diagnostic Feature Designer:** è uno strumento integrato in MATLAB che consente di analizzare, progettare e selezionare *feature* diagnostiche per sistemi meccatronici e non solo. Questo tool è particolarmente utile nei contesti di manutenzione predittiva e diagnosi dei guasti, dove è fondamentale identificare indicatori robusti e discriminanti per classificare stati normali e anomali di un sistema.
 - **ClassificationLearner:** è un'applicazione interattiva progettata per addestrare, valutare e ottimizzare modelli di machine learning per problemi di classificazione. Fa parte della toolbox *Statistics and Machine Learning* e offre un'interfaccia grafica che consente agli utenti di esplorare dataset, selezionare algoritmi di classificazione, confrontare le prestazioni dei modelli e generare codice MATLAB per automatizzare i processi.

2.4 Hardware di progetto

Per il lato hardware vengono utilizzati:

- **Holybro Pixhawk 6X**: è il più recente controller di volo Pixhawk, sul quale verrà implementato il controllore di volo e l'algoritmo di rilevamento guasti. In Figura 2.8 è riportato il suo design (FMUv6X). La scheda ha la porta USB-C per il collegamento con il pc e per l'upload del firmware e lo slot per l'SD Card, la quale viene utilizzata per memorizzare i file di configurazione, prove di volo e informazioni sulle missioni. Le caratteristiche sono riportate in Tabella 2.1: vengono evidenziati processori, RAM,



Figura 2.8: PixHawk 6x

sensori e le interfacce principali.

Processori	FMU Processor: STM32H753; IO Processor: STM32F100
RAM	1MB RAM ed 8KB SRAM
Flash Memory	2MB
Sensori	3x IMU ridondanti 2x Barometri ridondanti 1x Magnetometro
Interfacce	16x uscite servo PWM 8 porte seriali (3 TELE, 2 GPS, Debug console, UART4&I2C, PX4IO/RC 2 I2C

Tabella 2.1: Caratteristiche principali PixHawk 6x

- **Convertitore Seriale FT232 FTDI**: 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 sul controller PixHawk e di osservare il modello Simulink del Flight Controller in esecuzione sull'hardware in destinazione. Il modulo è mostrato in Figura 2.9



Figura 2.9: FT232 FTDI

Il modulo ha uno switch per scegliere la tensione in uscita da dare ai pin. Dato che gli input e gli output analogici funzionano a 3.3V, si seleziona lo switch su 3.3V.

- **Connettore JTS:** un connettore da 6 pin a due terminali (femmine) che permette di collegare il convertitore alla PixHawk tramite una porta seriale opportuna (si è scelta la porta TELEM3). 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 per l'alimentazione (VCC), uno per il collegamento a massa (GND), due per la ricezione (RXD) e la trasmissione (TXD); i pin CTS ed RTS vengono lasciati liberi. Il collegamento è mostrato in Figura A.25.

2.5 Workflow Hardwar-in-the-Loop

Durante la simulazione tramite tecnica HIL, 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, l'applicazione che comunica con l'hardware Pixhawk deve anche fare da ponte per la comunicazione MAVLink tra il Pixhawk e le altre applicazioni, come il flusso di lavoro del computer di bordo.

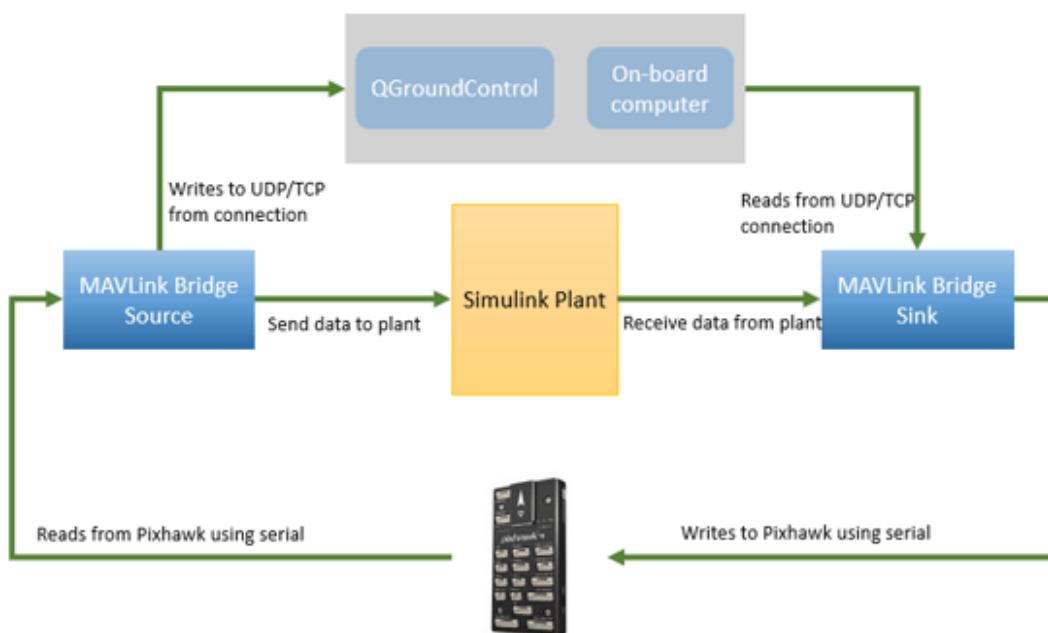


Figura 2.10: Workflow per l'HIL in PX4 [11]

La connettività MAVLink fornisce un modo per instradare il flusso di dati MAVLink dall'hardware Pixhawk al software di pianificazione della missione QGroundControl (QGC), all'hardware del computer di bordo, al Simulink Plant e viceversa nel flusso di lavoro HIL. In Figura 2.10 è rappresentato il workflow per l'HIL in PX4.

Per configurare il firmware PX4 per la simulazione Hardware-in-the-Loop (HIL) tramite MATLAB & Simulink, si riportano i passaggi nell'Appendice A.

Capitolo 3

Modello Matlab/Simulink

L'attività di tesi si è focalizzata su due principali modelli, che risultano implementati in Simulink:

1. Flight Controller
2. UAV Dynamics Model

3.1 Flight Controller

Il modello Simulink del Flight Controller rappresenta la logica di controllo utilizzata per gestire il volo del drone. Questo componente è progettato per essere compatibile con il firmware PX4, rendendolo ideale per le simulazioni Hardware-in-the-Loop (HIL). Le principali caratteristiche includono:

- *Implementazione dei loop di controllo*: Stabilizzazione (roll, pitch, yaw) e controllo della traiettoria.
- *Integrazione con PX4*: Consente il caricamento diretto del codice generato dal modello sul firmware PX4.
- *Supporto HIL e SITL*: Usato per testare i controllori sia in simulazione che con l'hardware reale.

L'implementazione in Simulink dello schema di controllo (Flight Controller) è riportato in Figura 3.1. In particolare, è composto da 4 blocchi:

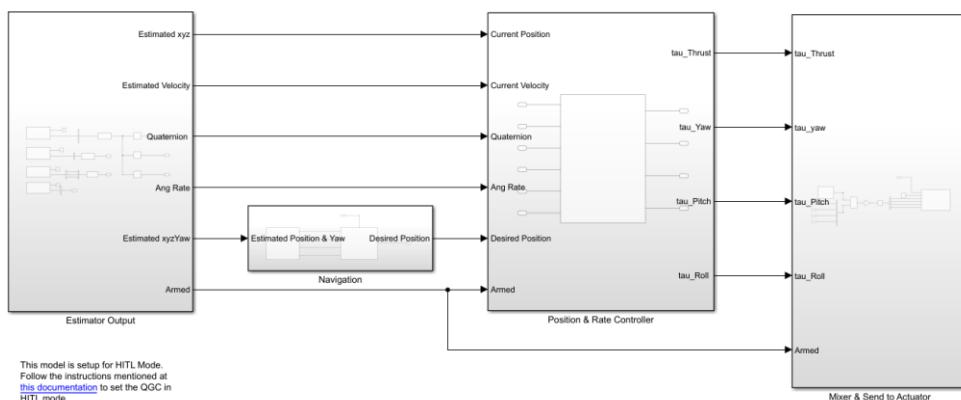


Figura 3.1: Flight Controller in Simulink

- **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;

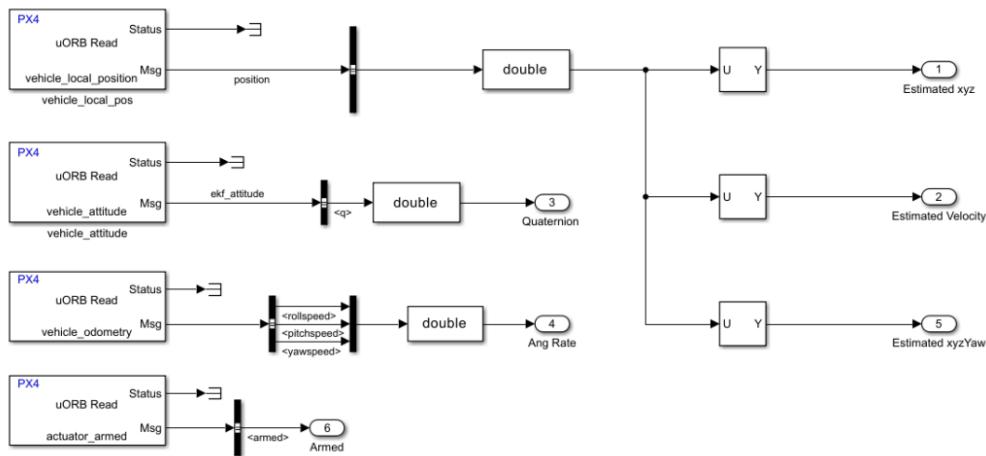


Figura 3.2: Contenuto del blocco Estimator Output

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

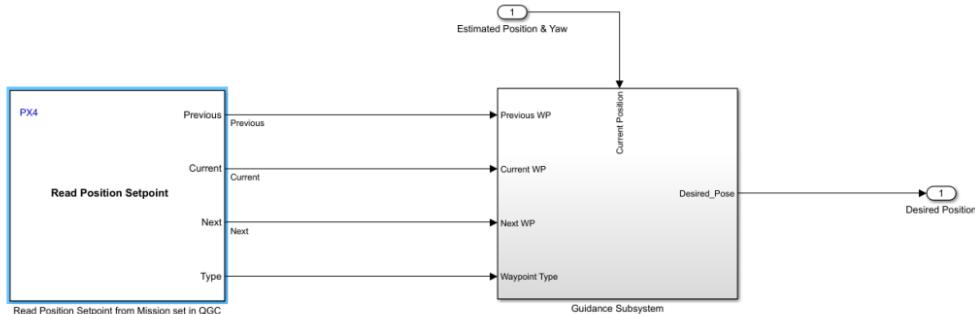


Figura 3.3: Contenuto del blocco Navigation

- **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;
- **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 direttamente dalla PixHwak.

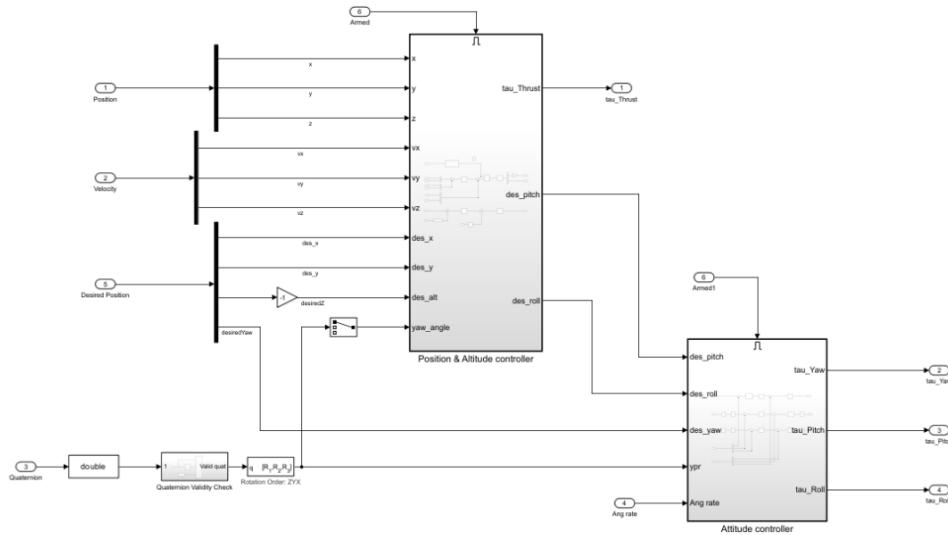


Figura 3.4: Contenuto del blocco Position and Rate Controller

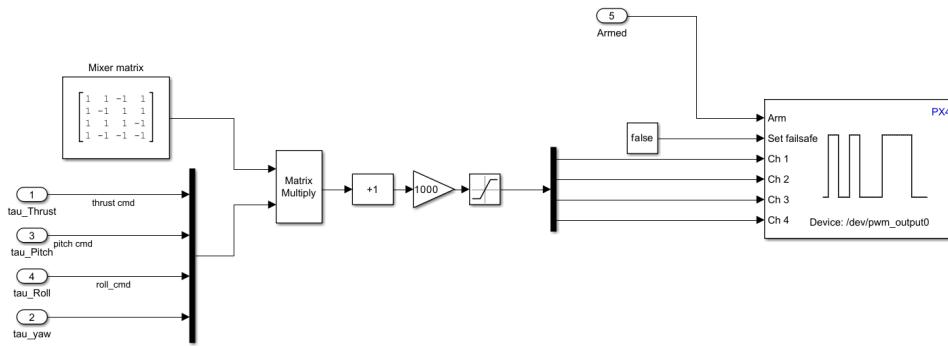


Figura 3.5: Contenuto del blocco Mixer and Send to Actuator

3.2 UAV Dynamics Model

Il modello UAV Dynamics Model rappresenta la dinamica fisica e i sensori del drone. Le principali funzionalità includono:

- *Simulazione delle dinamiche del volo*: Riproduce accuratamente il comportamento del drone, inclusi aerodinamica, motori e forze ambientali.
- *Modelli dei sensori*: Simula i dati di sensori come accelerometri, giroscopi, magnetometri, GPS e barometri, essenziali per l'integrazione con il Flight Controller.
- *Ambiente personalizzabile*: Permette di configurare diversi scenari di volo, inclusi guasti, disturbi atmosferici e traiettorie personalizzate.
- *Compatibilità con HIL*: Funziona come il modello fisico virtuale utilizzato dal Flight Controller durante i test HIL, consentendo una verifica completa del sistema.

Il modello simulink, mostrato in Figura 3.6, è composto da:

- **MAVlink Bridge Source & Sink**: Selezionando la porta COM corrispondente alla PixHawk, permettono il flusso di dati dal Plant Model di Simulink alla PixHawk e vicesa.

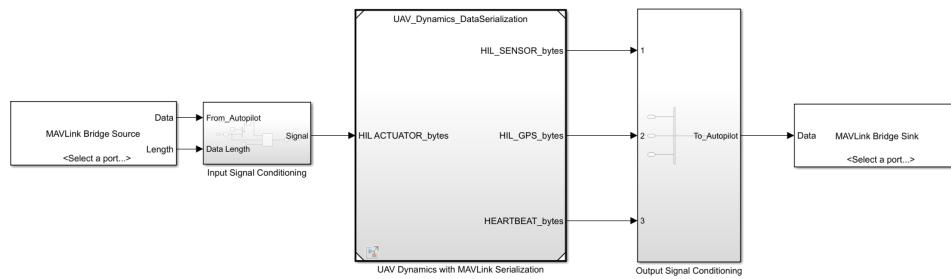


Figura 3.6: UAV Dynamics Model

- **Input/Output Signal Conditioning:** sono utilizzati per manipolare i segnali di input e output per garantire che siano nel formato corretto per essere processati dai blocchi successivi nel modello.
- **UAV Dynamics with MAVLink Serialization:** Contiene il modello matematico del UAV e la generazione dei segnali dovuti ai sensori simulati per l'HIL (IMU, GPS, Barometr)

Quest'ultimo punto lo si può notare meglio nella Figura 3.7

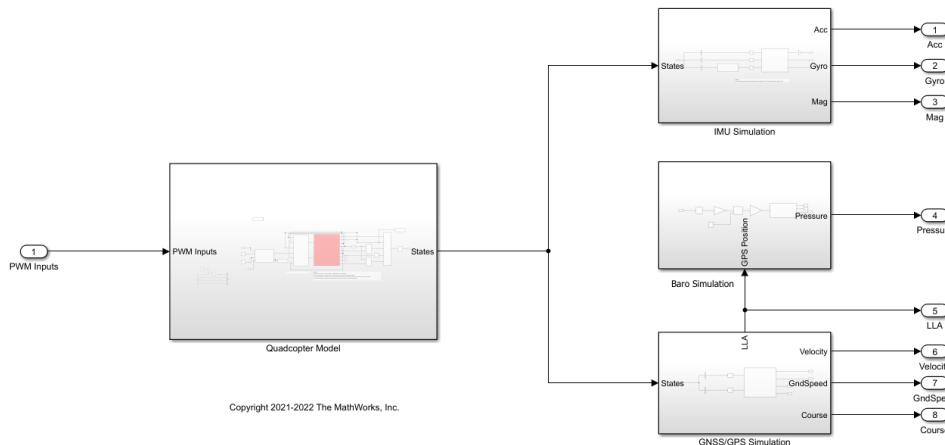


Figura 3.7: Contenuto del UAV Dynamics Model

In Figura 3.8 è riportato il modello matematico dell'UAV, che in questa tesi è un quadricottero.

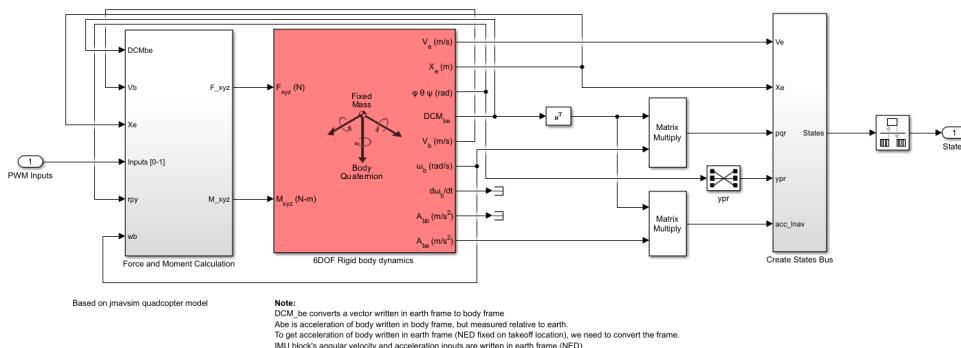


Figura 3.8: Modello matematico del UAV

Il drone per poter funzionare correttamente, ha bisogno dei segnali generati dai sensori IMU,

GPS e Barometro, ma dato che si sta svolgendo una simulazione Hardware In The Loop, questi segnali devono essere simulati. Questo avviene grazie a dei blocchetti Simulink appositamente creati per simulare i segnali dei sensori citati prima, in base al comportamento del drone durante il tragitto che sta compiendo su QGroundControl.

Nelle Figure 3.9, 3.10, 3.11, sono riportati i blocchi Simulink che generano i segnali simulati dei sensori IMU, GPS e Barometro.

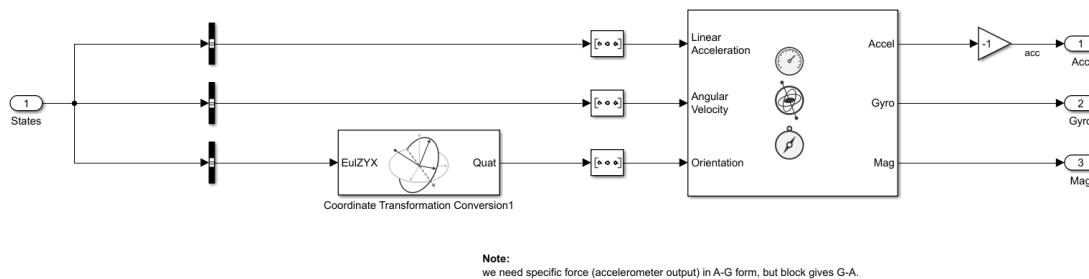


Figura 3.9: IMU Simulation

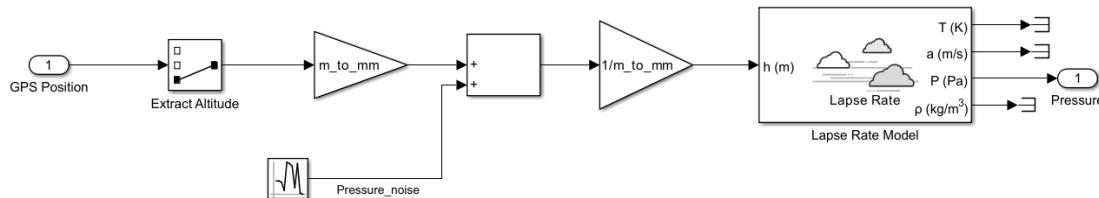


Figura 3.10: Baro Simulation

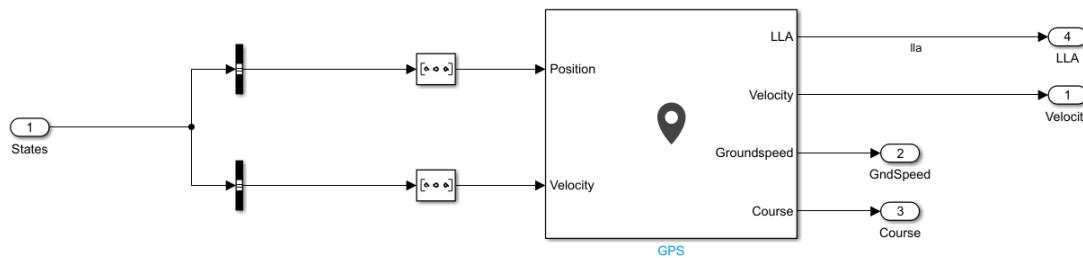


Figura 3.11: GPS Simulation

3.2.1 Blocco Simulatore IMU

Il blocco simulatore dell'IMU (Figura 3.9) utilizza dei *bus selector* per separare i segnali relativi alle accelerazioni lineari, velocità angolari e angoli di Eulero. Le variabili di stato, inizialmente in formato vettoriale, vengono trasformate in un formato $1 \times N$, dove N è il numero di elementi in input, attraverso un'operazione di *reshape*.

Gli angoli di Eulero vengono poi convertiti in quaternioni utilizzando il blocco *Coordinate Transformation Conversion*. Successivamente, tutti i dati vengono inviati al blocco IMU, che simula i tre principali sensori inerziali, accelerometro, giroscopio e magnetometro. Per ognuno di questi viene riportato rispettivamente lo schema del funzionamento dell'algoritmo di simulazione nelle Figure 3.12, 3.13 e 3.14.

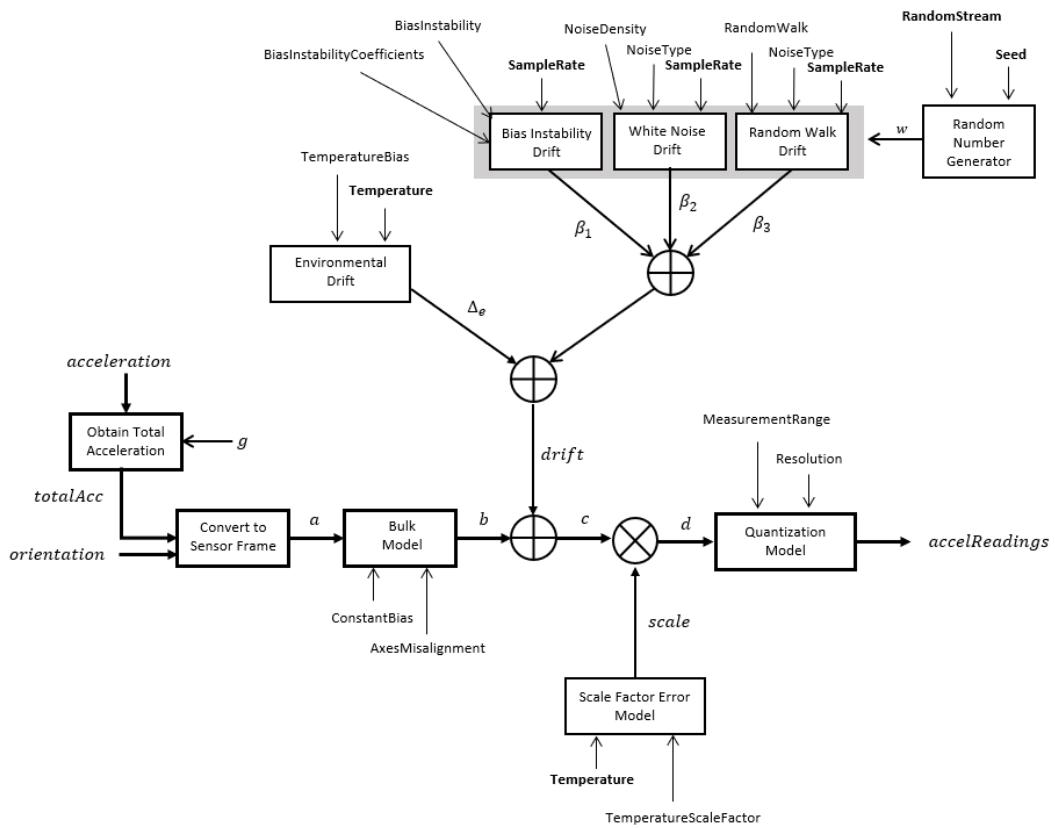


Figura 3.12: Accelerometer Algorithm [9]

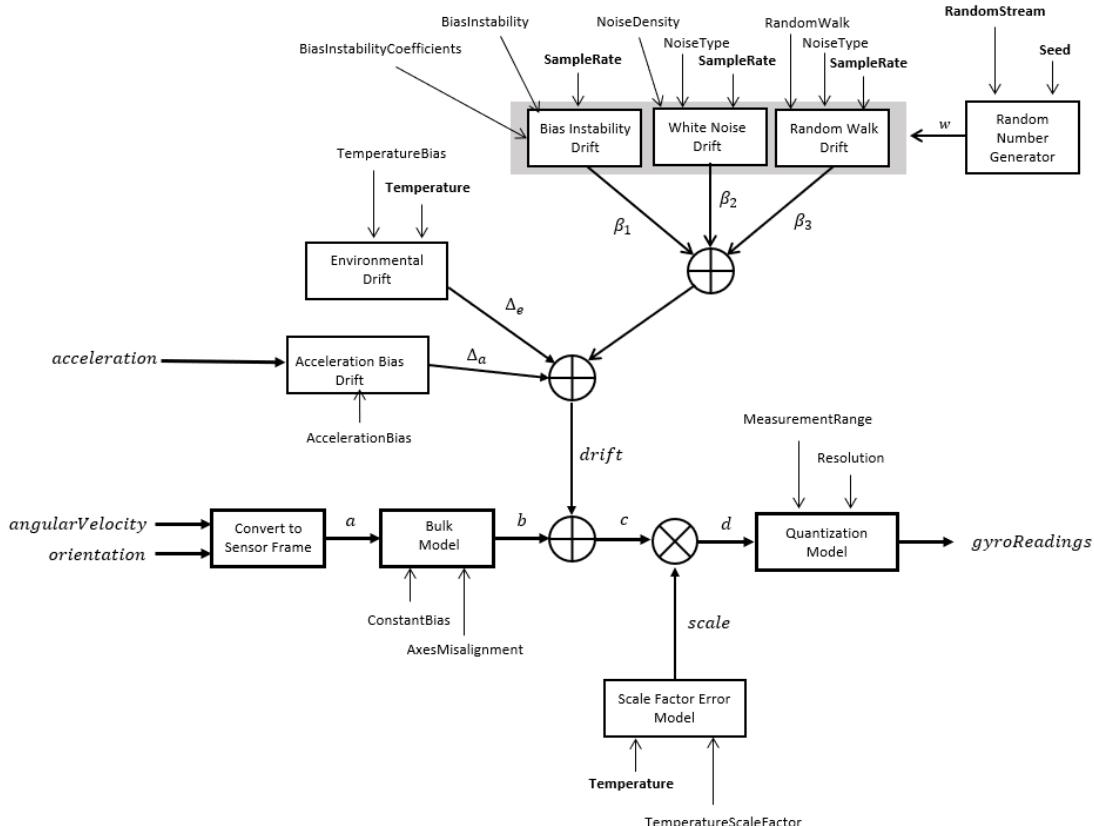


Figura 3.13: Gyroscope Algorithm [9]

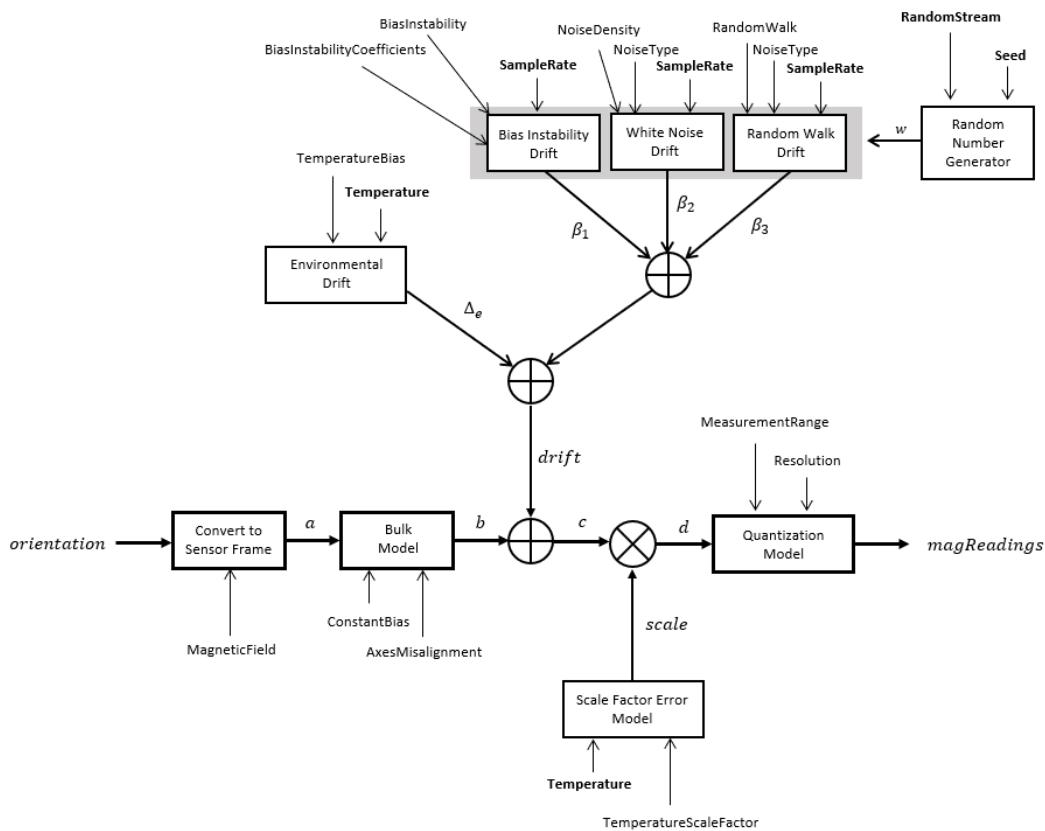


Figura 3.14: Magnetometer Algorithm [9]

3.2.2 Blocco Simulatore GPS

Il blocco di simulazione del GPS riceve in input le posizioni e le velocità lineari. Questo blocco permette di configurare diversi parametri, tra cui:

- **Accuratezza della posizione:** definisce il livello di precisione con cui vengono simulate le coordinate geografiche (latitudine, longitudine e altitudine).
- **Accuratezza della velocità:** determina il grado di precisione nella simulazione delle velocità lineari del drone.
- **Fattore di decadimento:** simula la perdita di accuratezza nel tempo, utile per modellare scenari realistici in cui il segnale GPS può degradarsi.

In output, il blocco fornisce le seguenti grandezze fondamentali per la navigazione:

- **Longitudine, Latitudine e Altitudine (LLA):** le coordinate geografiche simulate del drone.
- **Velocità:** il vettore di velocità lineare calcolato.
- **Groundspeed:** la velocità orizzontale al suolo.
- **Course angle:** l'angolo di direzione del drone rispetto al nord geografico.

Non avendo a disposizione le informazioni di come vengono simulate le coordinate GPS dalle posizioni lineari xyz, si è andato a cercare nel codice sorgente della PX4 la conversione di delle coordinate da LLA a xyz e viceversa. In questo modo si avranno delle informazioni disponibili in più in fase di iniezione di guasti o di studio del guasto.

Per la simulazione della posizione geografica, il file sorgente `geo.cpp` include una funzione denominata `map_projection_project`, che converte la posizione geografica in un riferimento locale utilizzando la Azimuthal Equidistant Projection. Questa proiezione permette di rappresentare le posizioni su una mappa 2D locale, centrata su un punto di riferimento specifico, come la posizione iniziale del drone o un punto definito dall'utente.

Equazioni della Proiezione

La proiezione `map-projection` calcola le coordinate cartesiane x, y a partire da latitudine (ϕ) e longitudine (λ). I parametri di riferimento ($\phi_{\text{ref}}, \lambda_{\text{ref}}$) sono inizializzati dalla funzione `initReference`. Le equazioni principali sono:

$$\cos \Delta\lambda = \cos(\lambda - \lambda_{\text{ref}}), \quad (3.1)$$

$$\arg = \text{constrain}(\sin(\phi_{\text{ref}}) \cdot \sin(\phi) + \cos(\phi_{\text{ref}}) \cdot \cos(\phi) \cdot \cos \Delta\lambda, -1, 1), \quad (3.2)$$

$$c = \arccos(\arg), \quad (3.3)$$

dove la funzione `constrain` garantisce che l'argomento di `arccos` rimanga nell'intervallo accettabile $[-1, 1]$:

$$\text{constrain}(val, \min, \max) = \begin{cases} \min & \text{se } val < \min, \\ \max & \text{se } val > \max, \\ val & \text{altrimenti.} \end{cases} \quad (3.4)$$

Il fattore di scala k è definito come:

$$k = \begin{cases} 1 & \text{se } c = 0, \\ \frac{c}{\sin(c)} & \text{altrimenti.} \end{cases} \quad (3.5)$$

Le coordinate proiettate x, y sono calcolate come segue:

$$x = k \cdot (\cos(\phi_{\text{ref}}) \cdot \sin(\phi) - \sin(\phi_{\text{ref}}) \cdot \cos(\phi) \cdot \cos \Delta\lambda) \cdot R, \quad (3.6)$$

$$y = k \cdot \cos(\phi) \cdot \sin(\lambda - \lambda_{\text{ref}}) \cdot R, \quad (3.7)$$

dove R è il raggio della Terra (`CONSTANTS_RADIUS_OF_EARTH` = 6371000).

Conversione Inversa

Per convertire x, y in latitudine e longitudine:

$$x_{\text{rad}} = \frac{x}{R}, \quad (3.8)$$

$$y_{\text{rad}} = \frac{y}{R}, \quad (3.9)$$

$$c = \sqrt{x_{\text{rad}}^2 + y_{\text{rad}}^2}. \quad (3.10)$$

In base al valore di c :

- Se $c > 0$:

$$\text{lat}_{\text{rad}} = \arcsin \left(\cos(c) \cdot \sin(\text{lat}_{\text{ref}}) + \frac{x_{\text{rad}} \cdot \sin(c) \cdot \cos(\text{lat}_{\text{ref}})}{c} \right), \quad (3.11)$$

$$\text{lon}_{\text{rad}} = \text{lon}_{\text{ref}} + \arctan 2(y_{\text{rad}} \cdot \sin(c), c \cdot \cos(\text{lat}_{\text{ref}}) \cdot \cos(c) - x_{\text{rad}} \cdot \sin(\text{lat}_{\text{ref}}) \cdot \sin(c)). \quad (3.12)$$

- Se $c = 0$:

$$\text{lat}_{\text{rad}} = \text{lat}_{\text{ref}}, \quad \text{lon}_{\text{rad}} = \text{lon}_{\text{ref}}. \quad (3.13)$$

Calcolo dell'Altitudine

Nel file `ALTITUDE.hpp`, l'altitudine viene derivata dalla coordinata z (posizione lungo l'asse verticale) del sistema locale NED. L'equazione utilizzata è:

$$\text{altitudine_amsl} = -\text{local_pos.z} + \text{local_pos.ref_alt}. \quad (3.14)$$

Se z non è valida, è possibile stimarla come:

$$z = \text{altitudine}_{\text{ref}} - \text{altitudine}. \quad (3.15)$$

Struttura del Test e Calcolo dell'Errore

Per verificare la correttezza delle conversioni tra coordinate cartesiane (x, y, z) e geografiche $(\text{lat}, \text{lon}, \text{alt})$, sono stati effettuati dei test confrontando i risultati della trasformazione con i dati di riferimento ottenuti dalle prove di volo.

3.2.3 Struttura del Test

Il test è stato strutturato come segue:

- Sono stati forniti dati di volo simulati contenenti coordinate (x, y, z) e i corrispondenti valori di latitudine, longitudine e altitudine simulati.
- È stata applicata la trasformazione da (x, y) a (lat, lon) utilizzando il modello matematico basato sulla *Azimuthal Equidistant Projection*.
- È stata verificata la conversione inversa da (lat, lon) a (x, y) per garantire la coerenza del modello.
- Per l'altitudine, è stata utilizzata la relazione:

$$z = \text{alt}_{\text{ref}} - \text{alt} \quad (3.16)$$

dove:

- alt_{ref} è l'altitudine di riferimento,
- alt è l'altitudine ottenuta dalla conversione,

- z è la coordinata verticale nel sistema locale.

L'errore tra le coordinate convertite e quelle reali è stato valutato attraverso la differenza assoluta tra il valore calcolato e quello atteso. Gli errori sono stati calcolati come:

$$e_{\text{lat}} = |\text{lat}_{\text{calcolata}} - \text{lat}_{\text{reale}}| \quad (3.17)$$

$$e_{\text{lon}} = |\text{lon}_{\text{calcolata}} - \text{lon}_{\text{reale}}| \quad (3.18)$$

$$e_z = |z_{\text{calcolato}} - z_{\text{reale}}| \quad (3.19)$$

Inoltre, è stato calcolato l'errore medio assoluto per ciascuna grandezza:

$$\bar{e}_{\text{lat}} = \frac{1}{N} \sum_{i=1}^N e_{\text{lat},i} \quad (3.20)$$

$$\bar{e}_{\text{lon}} = \frac{1}{N} \sum_{i=1}^N e_{\text{lon},i} \quad (3.21)$$

$$\bar{e}_z = \frac{1}{N} \sum_{i=1}^N e_{z,i} \quad (3.22)$$

dove N rappresenta il numero di campioni analizzati.

Dai risultati ottenuti, gli errori medi sono risultati nell'ordine di 10^{-8} a 10^{-6} gradi, corrispondente a circa $1mm$ - $10cm$, per la latitudine e la longitudine, indicando una discrepanza molto contenuta. La conversione da LLA a xy per errore medio assoluto è risultato pari a $0.81cm$ lungo x e $3.49cm$ lungo y . Per la coordinata z , l'errore è stato verificato rispetto alla differenza tra altitudine di riferimento e altitudine calcolata, con un errore medio assoluto di $0.76cm$ confermando la validità della trasformazione.

I risultati mostrano che la conversione tra i due sistemi di coordinate è affidabile, con errori trascurabili rispetto alla precisione richiesta per l'applicazione considerata.

3.2.4 Blocco simulatore barometro

La Longitudine, Latitudine e Altitudine (LLA) vengono passate al blocco di simulazione del barometro, dove viene selezionata la sola Altitudine. A questa viene aggiunto un rumore casuale al valore di pressione stocastico. Successivamente, attraverso il blocco *Lapse Rate Model*, vengono simulate diverse variabili ambientali come temperatura, pressione e velocità del suono. Tra queste, l'output di interesse è la pressione.

3.3 Log dei Segnali

Uno degli obiettivi principali della tesi era implementare il log dei segnali sia dal *Flight Controller* caricato sulla PixHawk tramite *Monitor & Tune*, sia dal modello di dinamica dell'UAV (*UAV Dynamics Model*) eseguito sull'Host PC.

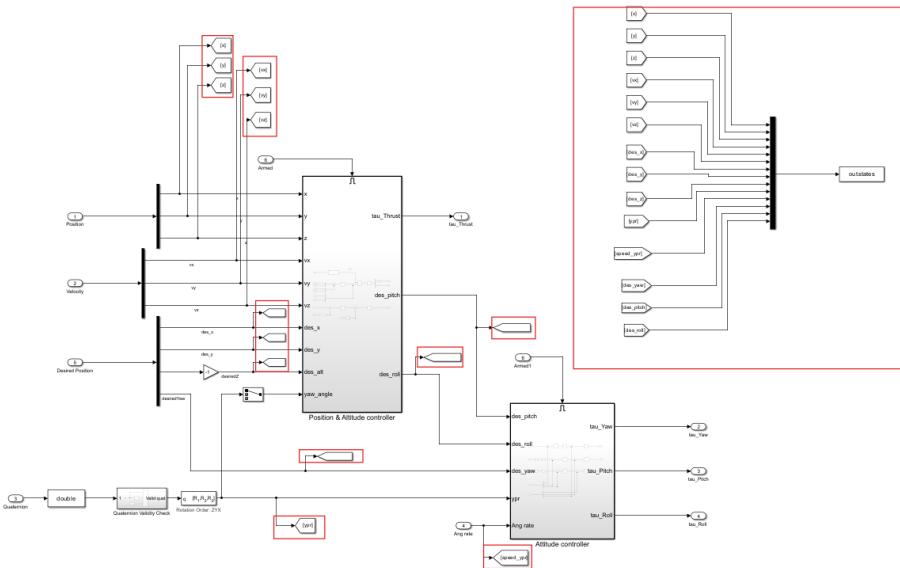


Figura 3.15: Log dei Segnali dal Flight Controller

3.3.1 Log dei Segnali dal Flight Controller

In Figura 3.15 è illustrato il processo di log dei segnali provenienti dal *Flight Controller*. I dati vengono salvati nello *Workspace* di Matlab nella variabile `out.Estimated_Input`.

I segnali registrati dal *Flight Controller* includono:

- **Posizione Stimata:** X, Y, Z .
- **Velocità Lineare Stimata:** dx, dy, dz .
- **Posizione Desiderata:** des_x, des_y, des_z .
- **Angoli di Eulero Stimati:** $Pitch, Roll, Yaw$.
- **Velocità Angolari Stimate:** $Vroll, Vpitch, Vyaw$.
- **Angoli di Eulero Desiderati:** $des_Pitch, des_Roll, des_Yaw$.

3.3.2 Log dei Segnali dal Modello di Dinamica UAV

I segnali loggati dal *UAV Dynamics Model* sono suddivisi in tre categorie, la prima riguarda i PWM calcolati dal Firmware della PX4 poi ricevuti nella dinamica attraverso il blocco *uORB*. La seconda categoria sono gli **stati reali del drone simulati dalla dinamica** e la terza sono gli **stati simulati dai sensori**. I dati vengono salvati nello *Workspace* di Matlab.

Stati reali del UAV calcolati dalla dinamica

- **Posizione:** X, Y, Z .
- **Angoli di Eulero:** $Yaw, Pitch, Roll$.
- **Velocità Lineari:** dx, dy, dz .
- **Accelerazioni Lineari:** ddx, ddy, ddz .
- **Velocità Angolari:** p, q, r .

Stati loggati dai Sensori

- **Accelerazioni Lineari:** Misurazioni delle accelerazioni lungo i tre assi principali del drone (x , y , z), fornite dall'accelerometro.
- **Gyro:** Fornisce la velocità angolare attorno agli assi principali ($roll$, $pitch$, yaw).
- **Mag:** Letture del magnetometro, che misura il campo magnetico terrestre.
- **LLA (Latitudine, Longitudine, Altitudine):** Coordinate geografiche fornite dal GPS o da altri sensori di navigazione. Questi dati permettono di localizzare il drone nello spazio globale e di pianificare percorsi basati su waypoint.
- **Velocity:** Velocità del drone calcolata dal GPS.
- **Ground Speed:** Velocità del drone rispetto al terreno, che include sia la componente orizzontale che quella verticale.
- **Course:** Direzione del movimento rispetto al nord magnetico o geografico, indicata in gradi.

Questi dati risultano fondamentali per l'analisi e la valutazione del comportamento e delle prestazioni del drone durante le simulazioni di volo. Essi forniscono un quadro completo delle dinamiche del sistema e supportano lo sviluppo di algoritmi di controllo e diagnosi.

3.4 Post-Simulazione e Visualizzazione dei Dati

Dopo aver configurato il percorso della simulazione e avviato il processo, il passo successivo consiste nell'attendere il termine della simulazione per analizzare i dati acquisiti e valutare le prestazioni del drone. Questa fase utilizza script in MATLAB per automatizzare il processo di ETL (*Extract, Transform, Load*) e successivamente visualizzare i dati ottenuti. La procedura è formalizzata come segue:

1. Termine della Simulazione

- Una volta avviata la simulazione, si attende il completamento del processo, assicurandosi che tutti i dati rilevanti siano stati registrati nel workspace (Fig. 3.16).
- Durante questa fase, il sistema registra la telemetria relativa sia alla dinamica del drone sia al Flight Controller.

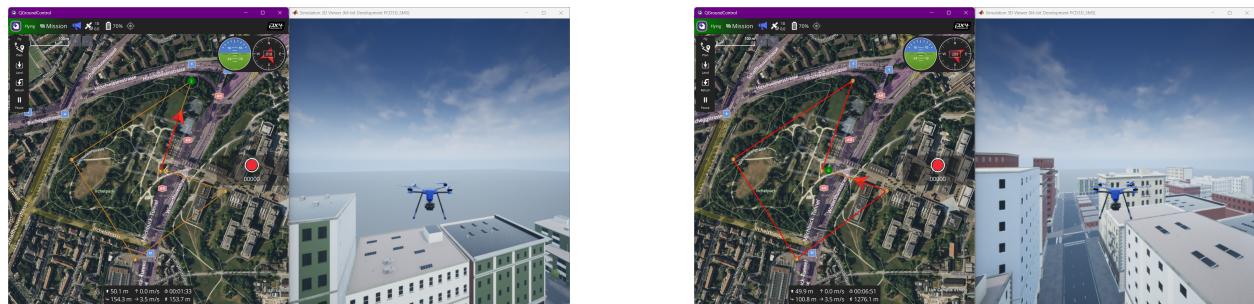


Figura 3.16: Percorso di prova durante e dopo la simulazione.

2. Salvataggio e Plot dei Dati

- Al termine della simulazione, uno script MATLAB preconfigurato automatizza il processo di ETL:
 - Verifica il tempo medio di campionamento del Flight Controller, che non sempre mantiene il valore di 100 Hz.
 - Riempie i campioni mancanti tramite interpolazione in avanti (*forward-filling*).
 - Sincronizza temporalmente i dati registrati dal Flight Controller con quelli della dinamica simulata del drone, poiché le due serie temporali possono iniziare in momenti diversi.
- Una volta processati, i dati vengono salvati in un file .mat per utilizzi futuri.
- Infine, è possibile generare plot personalizzati per analizzare le grandezze desiderate.

3. Plot 3D del Percorso

- Utilizzando le coordinate di posizione (X , Y , Z) registrate, uno script MATLAB genera un plot tridimensionale del percorso seguito dal drone durante la simulazione.
- Questo plot permette un confronto visivo tra il percorso effettivamente seguito e quello previsto, offrendo una chiara rappresentazione delle prestazioni del drone (Fig. 3.17).

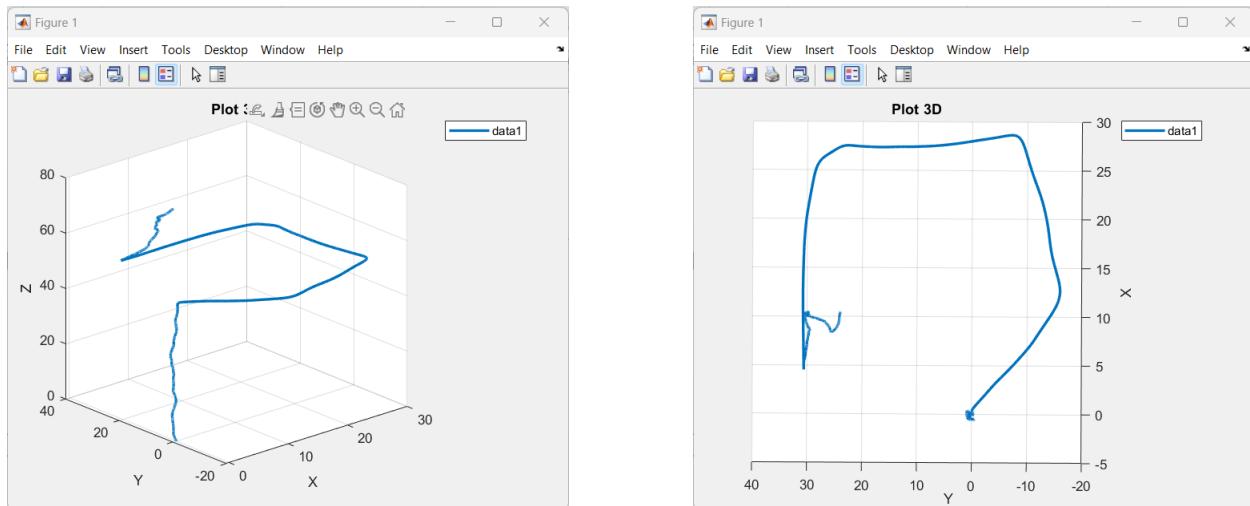


Figura 3.17: Esempio di plot 3D del percorso seguito dal drone.

Capitolo 4

Guasti UAV

Questo capitolo si concentra sull'analisi delle principali categorie di guasti che possono verificarsi in sensori e attuatori di un UAV. Per ciascun tipo di guasto, verranno fornite descrizioni dettagliate, esempi pratici e modellazioni matematiche che consentono di simulare il comportamento del sistema in presenza di guasti.

La modellazione matematica dei guasti è essenziale per:

- Comprendere le dinamiche introdotte da malfunzionamenti nei sottosistemi.
- Simulare il guasto.
- Progettare algoritmi di diagnosi e controllo robusti.

I guasti nei droni possono essere classificati in diverse tipologie, che si distinguono principalmente in base alla loro origine. Una prima suddivisione, proposta in [1] e [4], differenzia i guasti tra quelli che interessano gli attuatori e quelli che colpiscono i sensori. Questa classificazione risulta essenziale per simulare i guasti in modo accurato e sviluppare strategie di rilevamento e mitigazione. La suddivisione delle tipologie di guasto è rappresentata schematicamente in Figura 4.1.

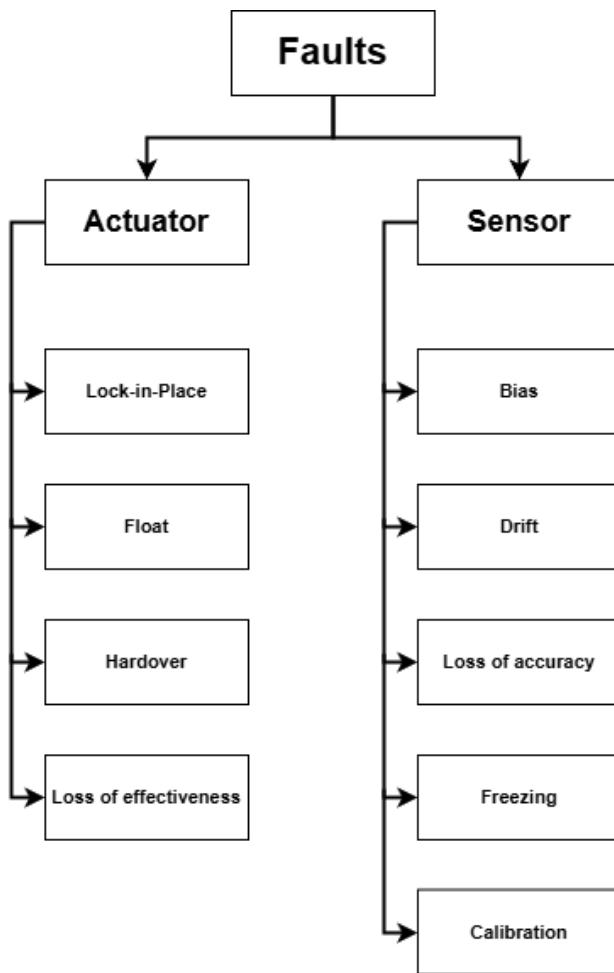


Figura 4.1: Tipologie di guasti [12]

4.1 Guasti negli Attuatori

I guasti negli attuatori possono compromettere la capacità di controllo e stabilità del drone. Un attuatore prende in ingresso un valore di comando ideale $u_c(t)$, e restituisce una variabile di forzamento $u(t)$ che può differire da quella di comando in caso di guasto. La Figura 4.2 rappresenta graficamente la precedente descrizione.

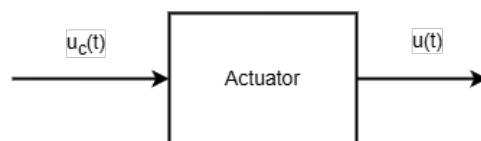


Figura 4.2: Actuator input/output

Per riportare queste tipologie di guasto in una simulazione, si necessita un modello matematico per ognuno di questi. I guasti negli attuatori possono essere modellati con le seguenti equazioni.

$$u(t) = \begin{cases} u_c(t) & \text{for all } t \geq 0 \quad (\text{No-Failure Case}) \\ k(t)u_c(t) & 0 < \epsilon \leq k(t) < 1, \text{ for all } t \geq t_F \quad (\text{Loss of Effectiveness}) \\ x_e + d(t), & |d(t)| \leq \bar{d}, \text{ for all } t \geq t_F \quad (\text{Float Type of Failure}) \\ u_c(t_F) & \text{for all } t \geq t_F \quad (\text{Lock-in-Place Failure}) \\ (u_c)_{\min} \text{ or } (u_c)_{\max} & \text{for all } t \geq t_F \quad (\text{Hard-Over Failure}) \end{cases}$$

4.1.1 Lock-in-Place

Il *Lock-in-Place* si verifica quando un attuatore si blocca e diventa inamovibile, rendendo impossibile il controllo del sistema. Questo guasto può essere causato da problemi meccanici, come l'assenza di lubrificazione, o da ostruzioni fisiche che impediscono il movimento dell'attuatore. La condizione di *Lock-in-Place* rappresenta un rischio significativo per i sistemi di controllo, poiché il blocco permanente compromette le prestazioni complessive del sistema.

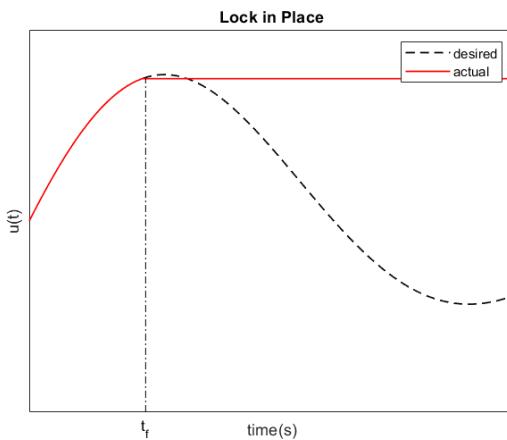
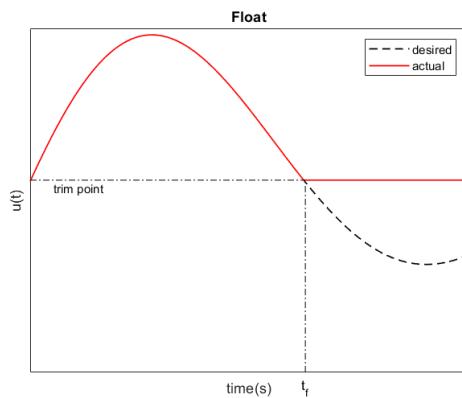
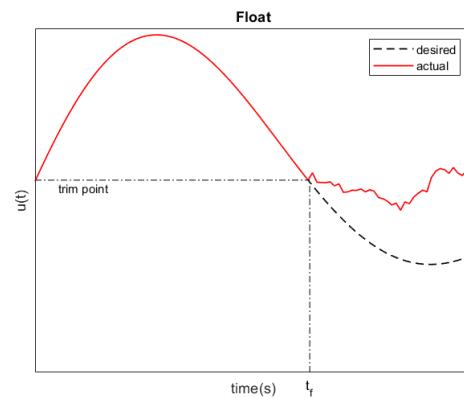


Figura 4.3: Lock in Place

4.1.2 Float Failure

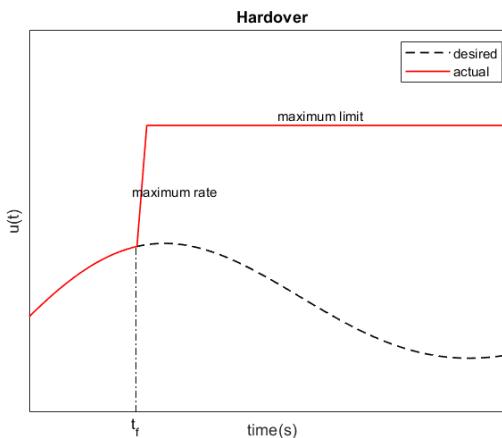
Il guasto noto come "float failure" si verifica quando una superficie di controllo si muove liberamente senza fornire il momento aerodinamico necessario per mantenere il controllo del volo. Un esempio tipico è la perdita di fluido idraulico. Questo tipo di guasto si manifesta in modo diverso negli UAV a **rotary wing** rispetto ai velivoli **fixed wing**.

- **Negli aerei (fixed wing)**, la superficie di controllo spesso si blocca in una posizione neutra (condizione di "lock in place"), portando il momento aerodinamico a un valore nullo, come mostrato in Figura 4.4.
- **Nei droni (rotary wing)**, invece, il guasto si manifesta come un comportamento instabile: l'elica "galleggia" attorno al punto di equilibrio. In questo caso, lo stato dell'elica non è completamente nullo, ma varia leggermente intorno al valore di riferimento, generando oscillazioni che complicano il controllo, come mostrato in figura 4.5.

**Figura 4.4:** Float fixed wing**Figura 4.5:** Float rotary wing

4.1.3 Hardover

L'*Hardover* è una situazione di guasto in cui una superficie di controllo si sposta fino a raggiungere il valore massimo del suo range operativo. Questo evento può verificarsi a causa di un malfunzionamento nei componenti elettronici, come un segnale errato inviato all'attuatore, che provoca una deviazione improvvisa e non desiderata della superficie verso i suoi limiti fisici. La condizione di *Hardover* può causare instabilità nel sistema e richiede interventi per prevenire ulteriori danni.

**Figura 4.6:** Hardover

4.1.4 Loss of Effectiveness

La *Loss of Effectiveness* si riferisce a un degrado progressivo delle prestazioni dell'attuatore. Questo guasto può essere attribuito a diversi fattori, tra cui usura meccanica, aumento dell'attrito o malfunzionamenti nei circuiti elettronici. La perdita di efficacia riduce la capacità dell'attuatore di rispondere adeguatamente ai comandi, compromettendo la precisione e l'affidabilità del sistema.

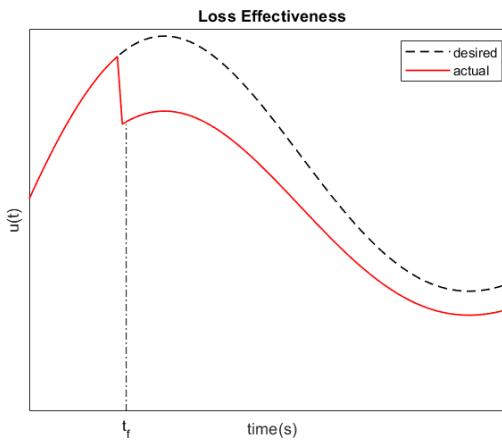


Figura 4.7: Loss of Effectiveness

4.2 Guasti nei Sensori

I sensori, al pari degli attuatori, possono essere soggetti a diverse tipologie di guasto. Un sensore che misura un segnale fisico $x(t)$ produce in uscita una misura digitale $y(t)$, la quale dipende dalla presenza o assenza di guasti. Questa relazione è rappresentata schematicamente in Figura 4.8.

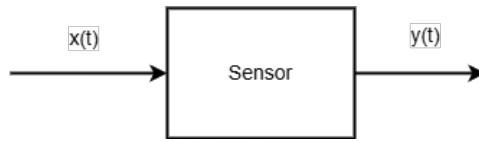


Figura 4.8: Sensor input/output

Nel seguito della sezione, vengono presentate le modellazioni matematiche proposte per le principali tipologie di guasti, che saranno utilizzate successivamente nelle simulazioni.

$$y_i(t) = \begin{cases} x(t), & \text{for all } t \geq 0 \quad (\text{No-Failure Case}) \\ x(t) + d, & \dot{d}(t) \equiv 0, d(t_F) \neq 0 \quad (\text{Bias}) \\ x(t) + d(t), & |d(t)| = \lambda t, 0 < \lambda \ll 1, \text{ for all } t \geq t_F \quad (\text{Drift}) \\ x(t) + d(t), & |d(t)| \leq \bar{d}, \dot{d}(t) \rightarrow 0, \text{ for all } t \geq t_F \quad (\text{Loss of Accuracy}) \\ x(t_F), & \text{for all } t \geq t_F \quad (\text{Sensor Freezing}) \\ cx(t), & \text{for all } t \geq t_F \quad (\text{Calibration Error}) \end{cases}$$

Dove t_F rappresenta il tempo di guasto, d è il coefficiente di accuratezza tale che $d \in [-\epsilon_{min}, \epsilon_{max}]$, dove $\epsilon_{min}, \epsilon_{max} > 0$, e c rappresenta coefficiente di efficacia, con $c \in [\hat{c}, 1]$, dove $\hat{c} > 0$. \hat{c}

4.2.1 Sensor Bias

Il *Sensor Bias* si riferisce a un errore o offset costante che esiste tra il valore misurato dal sensore e il valore reale del parametro che si intende rilevare. Questo tipo di errore può derivare da imperfezioni nella fabbricazione del sensore, errori di calibrazione o condizioni ambientali non ideali. Una caratteristica del bias è che, essendo costante, può essere compensato una volta identificato attraverso una calibrazione accurata.

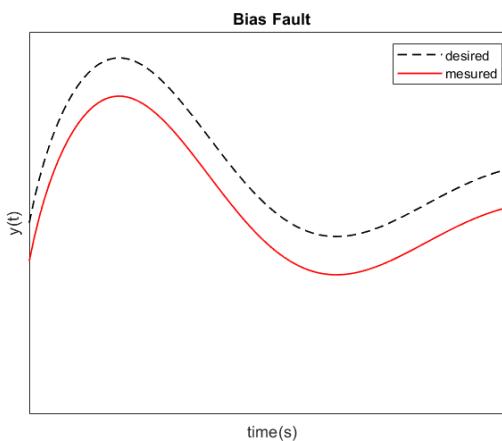


Figura 4.9: Bias

4.2.2 Sensor Drift

Il *Sensor Drift* descrive un errore di misura che cresce nel tempo. Questo fenomeno è spesso causato da cambiamenti fisici nel sensore, come la perdita di sensibilità dovuta all'usura, all'accumulo di contaminanti, o a variazioni ambientali. Il drift rende il sensore sempre meno affidabile con il passare del tempo, ed è per questo motivo che la manutenzione e la ricalibrazione periodica dei sensori sono fondamentali.

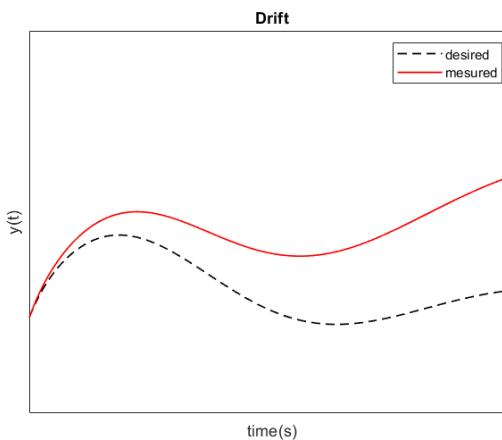


Figura 4.10: Drift

4.2.3 Loss of Accuracy

La *Loss of Accuracy* indica una condizione in cui le misure fornite dal sensore non rappresentano mai fedelmente i valori reali del segnale rilevato. Questa perdita di accuratezza può essere causata da diversi fattori, tra cui interferenze ambientali, degrado del sensore o errori di progettazione. L'accuracy è una caratteristica critica per i sistemi di misura, specialmente in applicazioni dove la precisione è essenziale per la sicurezza o l'efficienza.

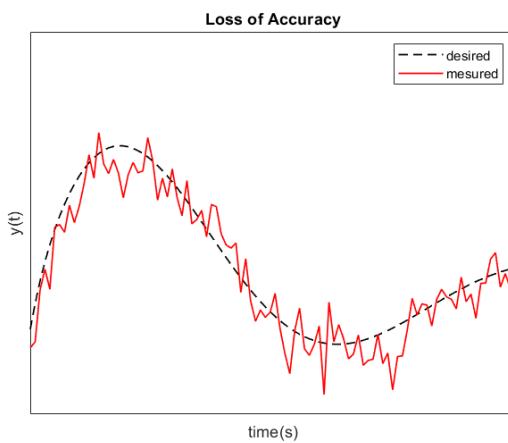


Figura 4.11: Loss of Accuracy

4.2.4 Freezing

Il *Freezing* si verifica quando un sensore si blocca, continuando a fornire costantemente lo stesso valore indipendentemente dalle variazioni reali nel sistema. Questo problema può essere causato da malfunzionamenti hardware, errori software o condizioni operative estreme. La rilevazione di un sensore bloccato è essenziale per prevenire errori sistematici nei dati raccolti e per garantire l'affidabilità del sistema.

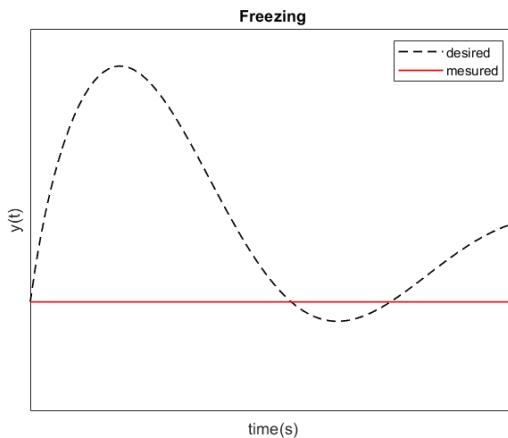


Figura 4.12: Freezing

4.2.5 Calibration Error

L'*Calibration Error* si manifesta quando un sensore rappresenta in modo errato il significato fisico del segnale misurato. Questo errore è spesso il risultato di un processo di calibrazione non corretto o incompleto, e può portare a una traduzione imprecisa delle grandezze fisiche misurate. Correggere un errore di calibrazione richiede una revisione del processo di calibrazione e, talvolta, una nuova caratterizzazione del sensore.

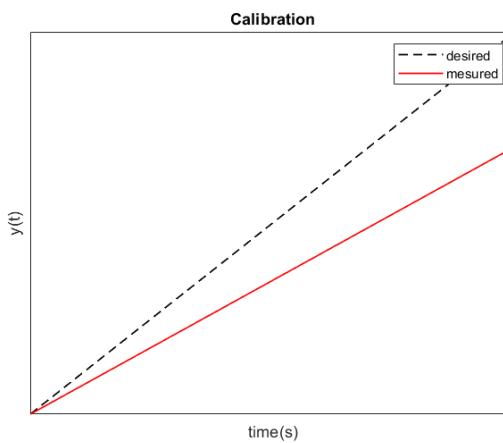


Figura 4.13: Calibration Error

4.3 Classificazione dei guasti per la loro implementazione

Per implementare i guasti, è stato necessario definire i modelli matematici che li rappresentano. Ogni guasto viene categorizzato in base alla sua tipologia. I guasti possono essere classificati come **additivi** o **moltiplicativi**, e inoltre possono manifestarsi con dinamiche **a gradino** o **a rampa**.

4.3.1 Tipologie di Guasti

Guasti Additivi: I guasti additivi si manifestano come un’alterazione diretta sul segnale del sensore o dell’attuatore. In questo caso, al valore nominale del segnale viene sommato un errore, che può essere costante, variabile nel tempo, o seguire una certa dinamica (gradino o rampa). Esempio: un sensore che riporta costantemente una lettura errata con uno scostamento fisso.

Guasti Moltiplicativi: I guasti moltiplicativi alterano il segnale attraverso un fattore di scala, moltiplicando il valore nominale per un coefficiente. Questo tipo di guasto può derivare, ad esempio, da una variazione nel guadagno del sensore o da un’alterazione delle sue proprietà fisiche.

4.3.2 Dinamiche dei Guasti

Guasti a Rampa (o "Incipient Failures"): I guasti a rampa si manifestano come un errore crescente nel tempo. Inizialmente l’errore è di piccola entità, ma aumenta gradualmente, rendendo il guasto più difficile da rilevare in una fase iniziale. Questo comportamento è tipico di un degrado progressivo del sensore.

Guasti a Gradino (o "Abrupt Failures"): I guasti a gradino si verificano improvvisamente, portando a un cambiamento netto e immediato nel segnale. Spesso sono causati da un danno fisico o da un malfunzionamento grave, e sono più facilmente rilevabili grazie alla loro natura evidente.

4.3.3 Tabella di Classificazione dei Guasti

La seguente tabella riassume le tipologie di guasti considerate, specificando per ciascuno se si è trattato come un tipo di guasto additivo o moltiplicativo, e se la loro dinamica è stata trattata come a gradino o a rampa. Questa classificazione consente di modellare i guasti in maniera precisa, facilitando l'implementazione nei sistemi di simulazione.

Guasto	Componente	Additivo	Moltiplicativo	Rampa	Gradino
Loss of Accuracy	Sensore	✓			✓
Freezing	Sensore	✓			✓
Calibration Error	Sensore		✓		✓
Bias	Sensore	✓			✓
Drift	Sensore	✓		✓	
Lock-in-Place	Attuatore		✓		✓
Float	Attuatore		✓		✓
Handover	Attuatore		✓		✓
Loss of Effectiveness	Attuatore		✓		✓

Tabella 4.1: Classificazione dei guasti in base alla natura additiva o moltiplicativa

Capitolo 5

Modulo di Generazione Guasti

L'introduzione dei moduli di *Fault Injection* (o generazione di guasti) rappresenta un passaggio cruciale per simulare in modo realistico i malfunzionamenti che possono verificarsi durante un volo reale. La capacità di riprodurre guasti verosimili consente di sviluppare e testare algoritmi avanzati, inclusi quelli basati su tecniche di intelligenza artificiale, per rilevare e gestire tempestivamente malfunzionamenti in scenari operativi.

Partendo dalle modellazioni matematiche e dalle classificazioni dei guasti presentate nel Capitolo 4, si implementano guasti specifici sia per gli attuatori sia per i sensori, integrandoli in punti strategici del sistema. Questi moduli sono progettati per offrire una rappresentazione flessibile e parametrizzabile dei guasti, con l'obiettivo di generalizzarne l'applicazione a diversi tipi di droni o altri sistemi aerei. Per facilitare l'interazione e l'integrazione nei flussi di lavoro, ogni blocco di iniezione guasti sarà corredata da una maschera grafica (GUI), sviluppata in MATLAB, che permetterà di configurare in modo intuitivo uno o più guasti.

La generalizzazione dei moduli è stata pensata per garantire l'adattabilità del framework a una vasta gamma di piattaforme. Tuttavia, vi sono componenti per i quali l'approccio è necessariamente specifico (e.g. i motori). L'iniezione di guasti nei motori è stata progettata specificamente per il quadricottero utilizzato in questo studio. La configurazione dei guasti dipende strettamente dal numero e dalla disposizione degli attuatori, che variano tra le diverse tipologie di UAV.

Per quanto riguarda le unità inerziali (**IMU**), i guasti sono stati progettati in modo generico, rendendoli applicabili a qualsiasi dispositivo che utilizzi questa tecnologia. Ciò consente di estendere l'applicabilità del modulo di guasto anche a droni con architetture diverse. Stessa argomentazione vale per il GPS a cui saranno applicati guasti di tipo *loss of accuracy* e una *packet loss*.

Questa architettura modulare non solo consente di testare algoritmi di rilevamento e mitigazione dei guasti in modo efficace, ma rappresenta anche uno strumento fondamentale per l'addestramento e la validazione di sistemi autonomi, contribuendo a migliorare la sicurezza e l'affidabilità dei droni in condizioni operative complesse.

5.1 Struttura dei Moduli

Ogni modulo o blocco di iniezione guasti opera sulla linea di segnale associata alla grandezza soggetta al guasto. L'implementazione varia a seconda del componente o sensore considerato. Ad esempio:

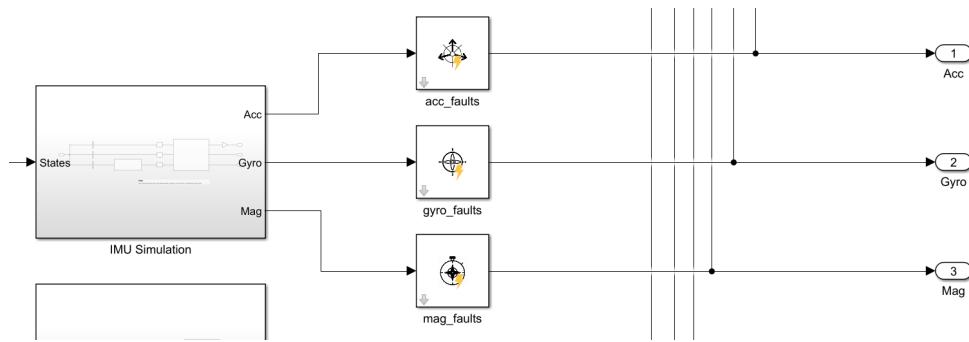


Figura 5.1: IMU Fault Modules

- Per i motori, il modulo agisce direttamente sul segnale *PWM* (Pulse Width Modulation) inviato agli attuatori.
- Per l'accelerometro, giroscopio e magnetometro il guasto può essere iniettato su uno o più assi del sistema di riferimento cartesiano (*XYZ*).
- Per la longitudine, latitudine e altitudine si agisce sulla accuratezza del segnale e sulla sua perdita.

Ogni modulo è strutturato principalmente come segue:

- **Segnale di input:** La grandezza misurata o il comando che sarà soggetto al guasto.
- **Parametri di configurazione:** Parametri configurabili che indicano la presenza o assenza del guasto e, dove necessario, la sua intensità o entità.
- **Logica di iniezione del guasto:** Un'unità che elabora il segnale in ingresso e applica il guasto in base ai parametri definiti.
- **Segnale di output:** La grandezza risultante, che può essere alterata dal guasto o lasciata invariata, a seconda della configurazione.

L'output finale è quindi il segnale modificato per includere il guasto, qualora configurato, oppure il segnale originale non alterato. Questa struttura modulare consente di adattare facilmente i blocchi a diverse tipologie di guasti e applicazioni.

5.1.1 Moduli Fault Injection IMU

L'IMU, essendo costituita da accelerometro, giroscopio e magnetometro, include tre moduli distinti per l'iniezione dei guasti, uno per ciascun sensore. Come illustrato in Figura 5.1, ogni sensore ha un modulo di iniezione guasti associato all'output. Questi moduli condividono la stessa struttura sia a livello modulare sia a livello algoritmico. I guasti simulabili per tutti e tre i sensori includono *bias*, *drift*, *loss of accuracy* e *freezing*. Le differenze tra i moduli riguardano unicamente la nomenclatura specifica per ciascun sensore.

In Figura 5.2 è mostrata la struttura interna del modulo di *fault injection* per l'accelerometro, che rappresenta un modello generale anche per i moduli dedicati al giroscopio e al magnetometro. Questo approccio consente di configurare guasti per assi specifici del sensore, grazie all'introduzione di tre variabili booleane che permettono di selezionare l'asse interessato.

Ogni modulo include i seguenti componenti principali:

- t_f : Variabile che rappresenta il tempo di guasto specificato dall'utente.

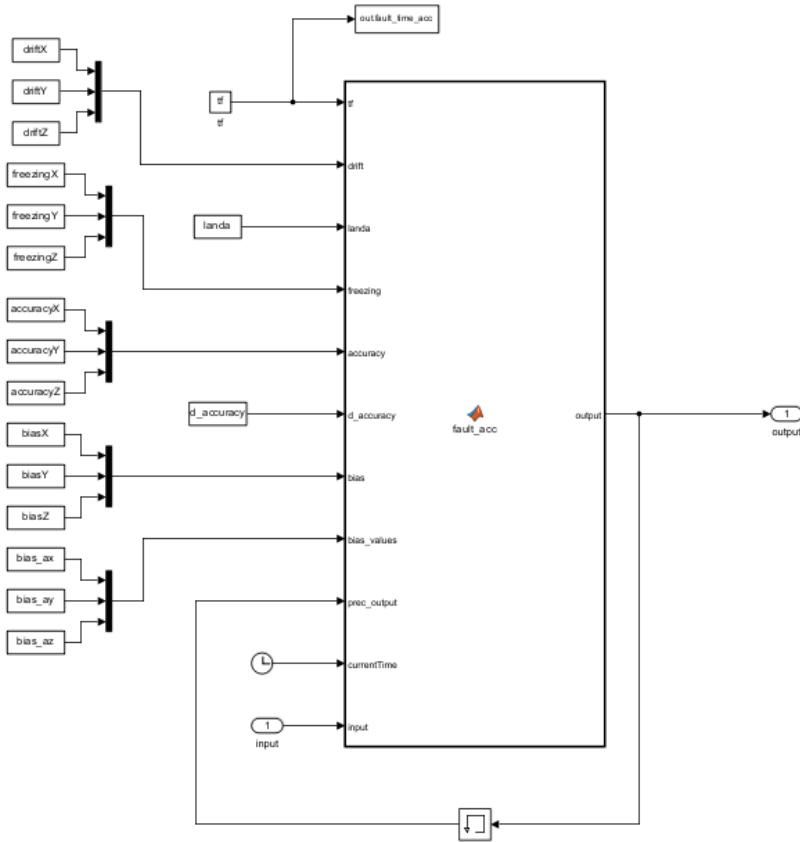


Figura 5.2: Accelerometer Fault Module

- **Current Time:** Tempo attuale della simulazione, utilizzato per confrontarlo con il tempo di guasto t_f .
- **Input:** Segnale in uscita dal sensore (ad esempio, l’accelerazione nel caso dell’accelerometro).
- **Lambda e $d_accuracy$:** Parametri che definiscono rispettivamente l’entità del guasto di tipo *drift* e la perdita di accuratezza (*loss of accuracy*).
- **Prec_output:** Valore di output del modulo al tempo precedente, ottenuto tramite un blocco di memoria (*memory block*) che introduce un ritardo temporale di un passo simulativo.

Tutte queste variabili vengono fornite come input alla funzione MATLAB che implementa l’algoritmo di iniezione del guasto. Il risultato è un segnale di output che incorpora gli effetti del guasto configurato, oppure, in assenza di guasti, restituisce il segnale originale non alterato.

```

1 function output = fault_acc(tf, drift, landa, freezing, accuracy,
2 d_accuracy, bias, bias_values, ...
3 prec_output, currentTime, input)
4
5 % output preallocation
6 output = input;
7
8 % check the fault time
9 if currentTime >= tf

```

```
9      % calculate the time from the start of the fault
10     time_since_fault = currentTime - tf;
11
12     % ---- Linear Drift ----
13     if any(drift)
14         if drift(1) % Drift X
15             output(1) = output(1) + landa * time_since_fault;
16         end
17         if drift(2) % Drift Y
18             output(2) = output(2) + landa * time_since_fault;
19         end
20         if drift(3) % Drift Z
21             output(3) = output(3) + landa * time_since_fault;
22         end
23     end
24
25     % ---- Accuracy ----
26     if any(accuracy)
27         if accuracy(1) % Accuracy X
28             output(1) = output(1) + ((rand - 0.5) * 2) *
29                 d_accuracy;
30         end
31         if accuracy(2) % Accuracy Y
32             output(2) = output(2) + ((rand - 0.5) * 2) *
33                 d_accuracy;
34         end
35         if accuracy(3) % Accuracy Z
36             output(3) = output(3) + ((rand - 0.5) * 2) *
37                 d_accuracy;
38     end
39
40     % ---- Freezing ----
41     if any(freezing)
42         if freezing(1) % Freezing X
43             output(1) = prec_output(1);
44         end
45         if freezing(2) % Freezing Y
46             output(2) = prec_output(2);
47         end
48         if freezing(3) % Freezing Z
49             output(3) = prec_output(3);
50         end
51     end
52
53     % ---- Bias ----
54     if any(bias)
55         if bias(1) % Bias X
56             output(1) = output(1) + bias_values(1);
57         end
58         if bias(2) % Bias Y
59             output(2) = output(2) + bias_values(2);
```

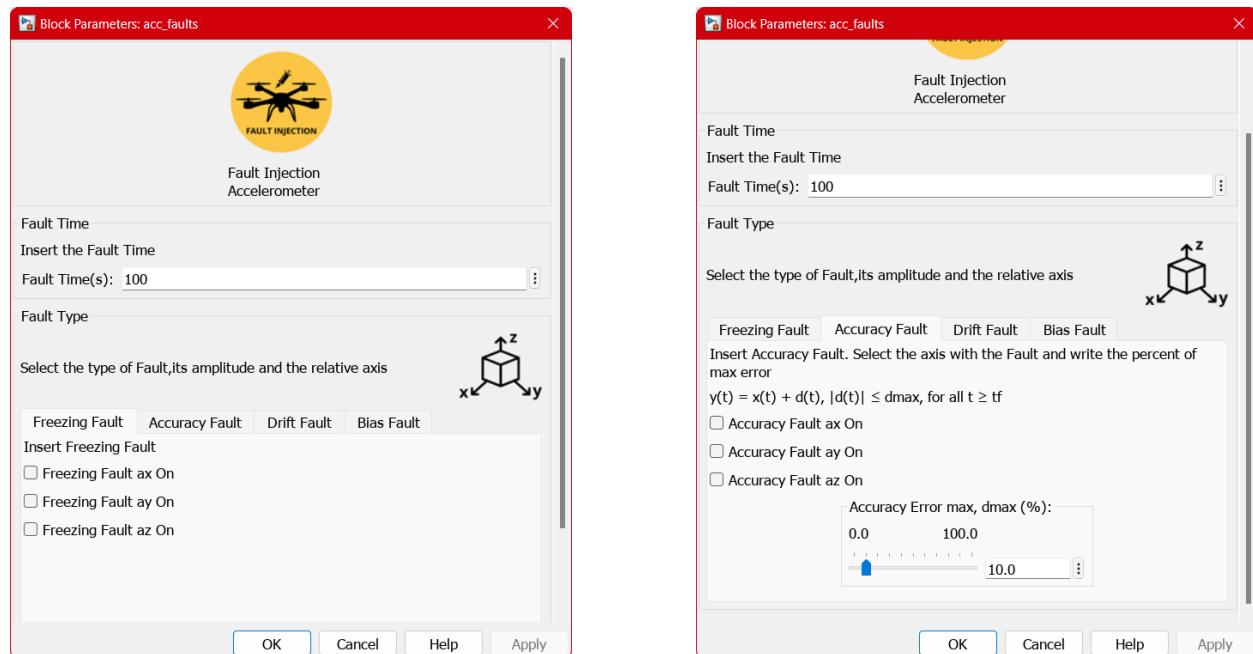
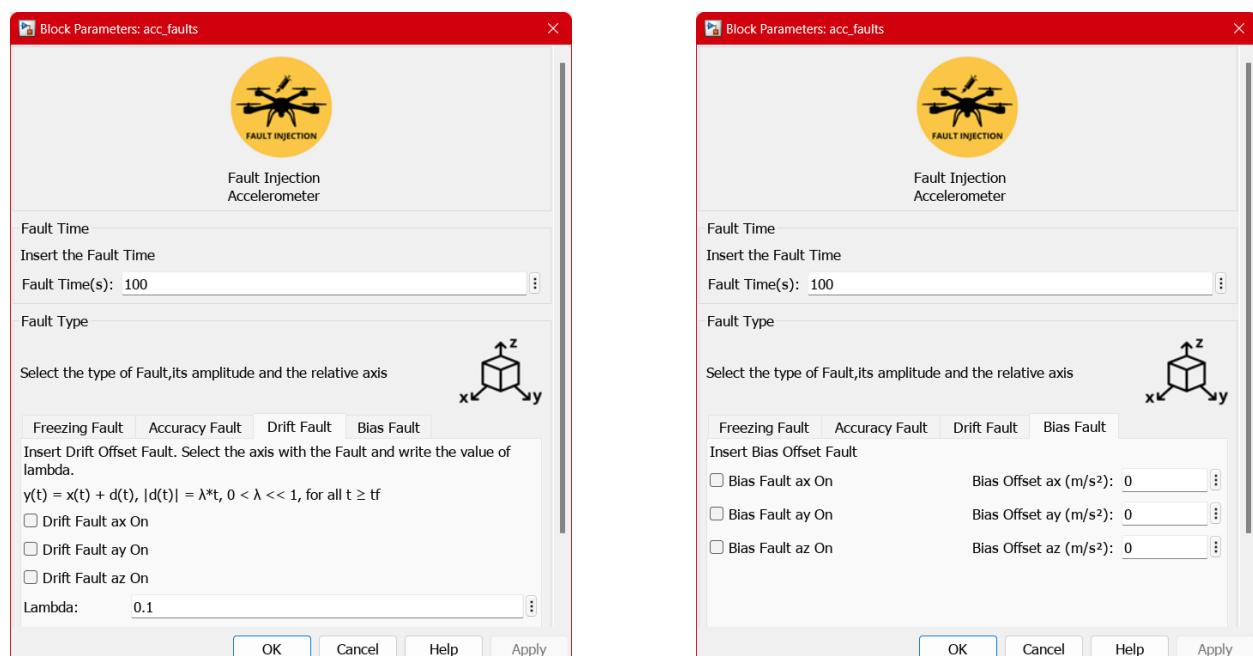
```

60         output(3) = output(3) + bias_values(3);
61     end
62 end
63 end
64 end

```

Listing 5.1: Fault Injection Accelerometer

L'uscita della MATLAB Function fornisce l'output del sensore con o senza guasto. Per rendere il tutto più accessibile all'utente si è progettato di inserire una maschera a questi moduli. Prima di far partire la simulazione l'utente dovrà inserire il tempo di guasto, selezionare i guasti voluti, e l'ampiezza del guasto dove richiesto.

**Figura 5.3:** Maschere Fault Injection per l'accelerometro per i guasti Freezing e Loss of Accuracy**Figura 5.4:** Maschere Fault Injection per l'accelerometro per i guasti Drift e Bias

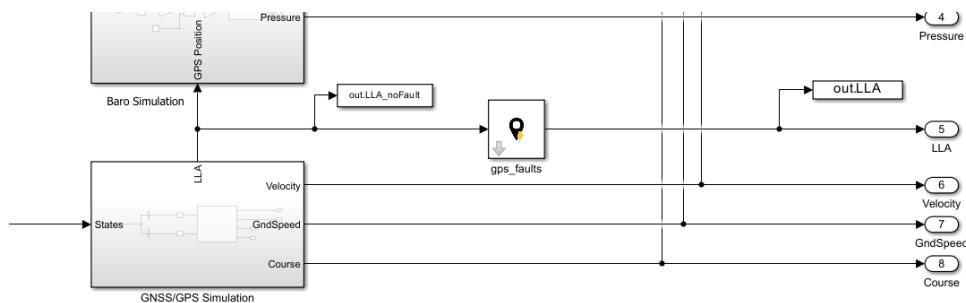


Figura 5.5: Modulo Fault Injection GPS

I moduli di Fault Injection per il giroscopio e masnetometro non sono stati descritti e riportati, perché simili a quelli dell'accelerometro

5.1.2 Modulo Fault Injection GPS

Il modulo di iniezione dei guasti per il GPS opera sulla linea che trasporta le informazioni di Latitudine, Longitudine e Altitudine (LLA) generate dal simulatore GPS, come mostrato in Figura 5.5. Questo modulo è stato progettato per introdurre guasti realistici che possano compromettere l'affidabilità delle informazioni di posizione e altitudine durante la simulazione.

Per il modulo di Fault Injection GPS, sono stati considerati due tipi principali di guasti:

- **Loss of Accuracy (Perdita di Accuratezza):** Questo guasto rappresenta un errore di accuratezza nelle misurazioni di latitudine e longitudine. L'entità dell'errore, fornita dall'utente in metri, viene applicata direttamente ai valori di latitudine e longitudine, simulando uno spostamento geografico rispetto alla posizione reale. Questo tipo di guasto permette di valutare la robustezza degli algoritmi contro errori graduali o improvvisi nei dati di posizione.
- **Packet Loss (Perdita di Pacchetti):** Questo guasto simula l'interruzione o la perdita di pacchetti di dati GPS, spesso causata da problemi di comunicazione o di segnale. L'utente può specificare una probabilità di perdita del segnale (*lossProbability*) e, durante il tempo di guasto, il modulo assegna valori NULL ai dati LLA in base alla probabilità specificata. Questo permette di testare la capacità del sistema di gestire situazioni di perdita intermittente del segnale.

La struttura interna del modulo Fault Injection GPS è illustrata in Figura 5.6. I parametri e le variabili utilizzate sono descritti di seguito:

- **tempo_guasto (*t_f*):** Indica il tempo a partire dal quale il guasto viene attivato.
- **gps_accuracy e loss_signal:** Variabili booleane che abilitano rispettivamente i guasti di Loss of Accuracy e Packet Loss.
- **accuracyErrorMeters:** Specifica l'entità dell'errore di accuratezza in metri da applicare a latitudine e longitudine.
- **lossProbability:** Indica la probabilità percentuale di perdita di segnale durante un intervallo di guasto.
- **currentTime:** Tempo attuale della simulazione, utilizzato per determinare se il guasto deve essere attivato.

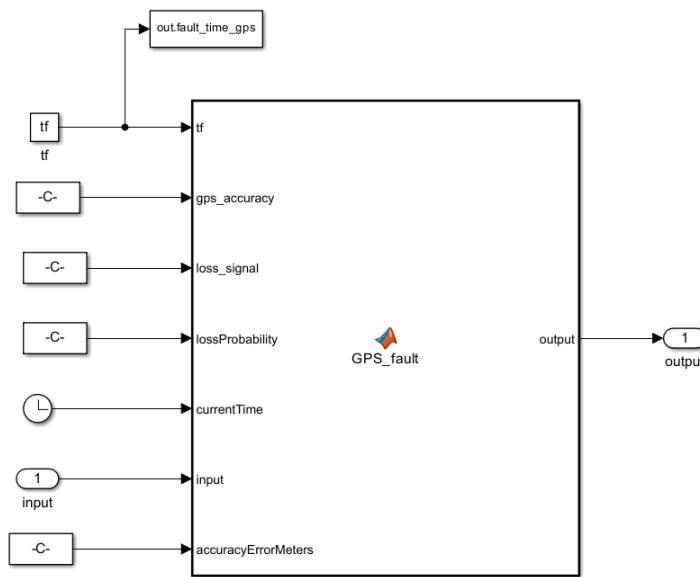


Figura 5.6: Interno del modulo Fault Injection GPS

- **input:** Contiene i valori di Latitudine, Longitudine e Altitudine generati dal simulatore GPS.
- **output:** Contiene i valori di LLA modificati dai guasti, oppure i valori originali in assenza di guasti.

L'implementazione della logica dei guasti è gestita all'interno di una **MATLAB Function**, che riceve in ingresso le variabili sopra descritte e produce i dati di LLA con o senza guasto. Per il guasto di *Loss of Accuracy*, la funzione aggiunge un errore casuale ai dati di latitudine e longitudine, proporzionale a *accuracyErrorMeters*. Per il guasto di *Packet Loss*, la funzione confronta un valore casuale con *lossProbability* per determinare se i dati devono essere annullati.

```

1 function output = GPS_fault(tf, gps_accuracy, loss_signal,
2     lossProbability, ...
3     currentTime, input, accuracyErrorMeters)
4
5     output = input;
6
7     if (currentTime >= tf)
8         % Packet Loss Simulation
9         if(loss_signal)
10             lossProbability = lossProbability/100;
11             % rand returns a random scalar drawn from the uniform
12             % distribution in the interval (0,1)
13             if rand < lossProbability
14                 output = zeros(size(input));
15             else
16                 output = input;
17             end
18         end
19
20         if(gps_accuracy)
21             meters_per_degree_lat = 111320; % Latitude approximation
22         end
23     end
  
```

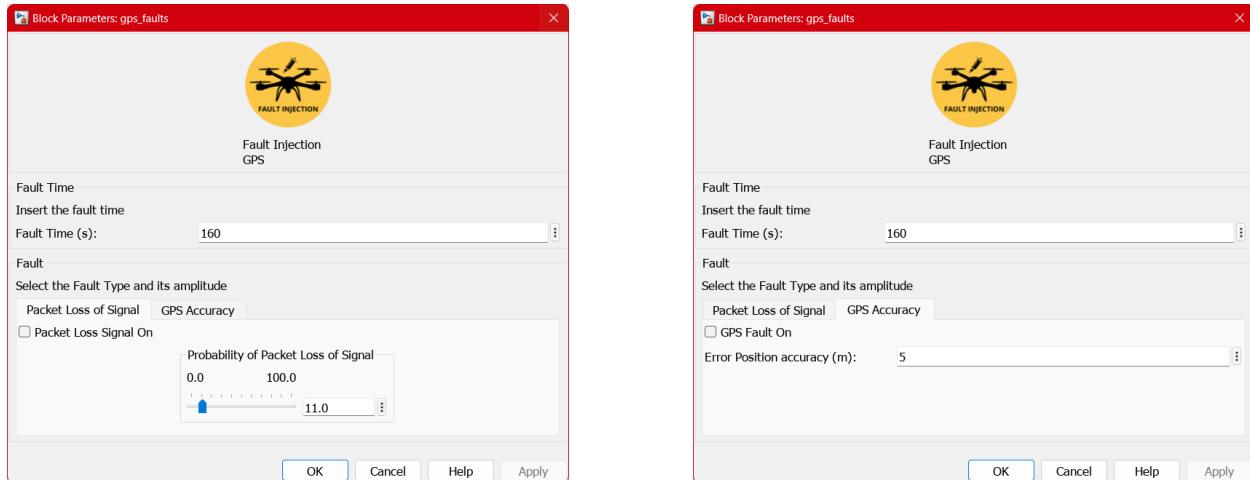
```

20         meters_per_degree_lon = meters_per_degree_lat * cosd(
21             output(2));
22
23     % Generating random error
24     error_lat = randn(1) * (accuracyErrorMeters /
25         meters_per_degree_lat);
26     error_lon = randn(1) * (accuracyErrorMeters /
27         meters_per_degree_lon);
28
29     % Add error to coordinates
30     output(1) = input(1) + error_lat;
31     output(2) = input(2) + error_lon;
32
33 end

```

Listing 5.2: Fault Injection GPS

Anche per il Fault Injection del GPS è stata proposta una maschera per l'utente riportata in Figura 5.7.

**Figura 5.7:** Maschere Fault Injection per l'accelerometro per i guasti Packet Loss e Loss of Accuracy

5.1.3 Modulo Fault Injection Attuator

Il modulo di iniezione guasti per gli attuatori opera sul valore di *PWM* dei singoli motori. Questo modulo è posizionato nella linea che trasporta i segnali *PWM* verso il modello del quadcopter, come mostrato in Figura 5.8. Ogni motore è identificato da un numero, come indicato in Figura 5.9, e il modulo consente di simulare guasti su uno o più motori specifici.

Il modulo Fault Injection Attuator utilizza i seguenti parametri e variabili:

- **input:** Contiene i segnali *PWM* dei quattro motori.
- **tf (*tempo_guasto*):** Specifica il tempo a partire dal quale il guasto viene attivato.
- **currentTime:** Tempo attuale della simulazione, utilizzato per determinare se il guasto deve essere attivato.

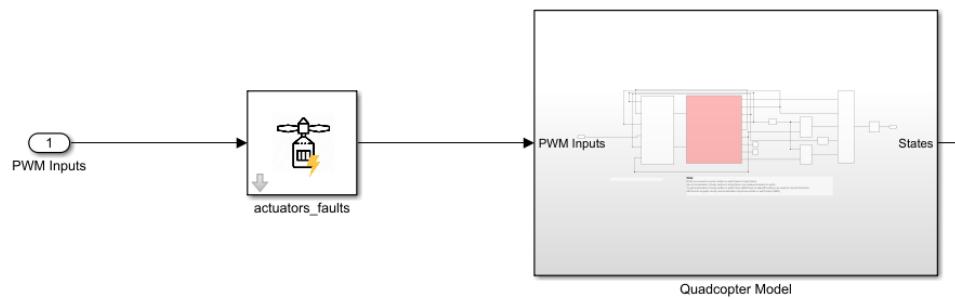


Figura 5.8: Modulo Fault Injection Attuatori

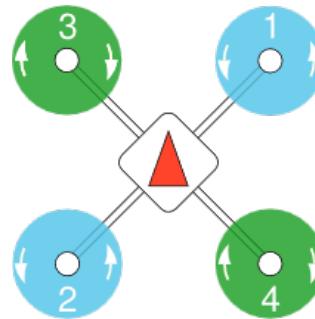


Figura 5.9: Identificazione dei motori nel drone

- **num_motore:** Variabile che indica il motore soggetto al guasto. Assume un valore tra 0 e 4, dove 0 indica l'assenza di guasti e i numeri da 1 a 4 identificano rispettivamente i motori guasti.
- **loss_effectiveness:** Variabile booleana che abilita il guasto di *Loss of Effectiveness* (perdita di efficacia).
- **perc:** Percentuale di efficienza persa per il guasto di *Loss of Effectiveness*.
- **lock_in_place:** Variabile booleana per il guasto di *Lock in Place* (blocco del motore al valore precedente).
- **prec_output:** Valore *PWM* del motore al passo di simulazione precedente, utilizzato per implementare il guasto *Lock in Place*.
- **float:** Variabile booleana per il guasto di *Float* (interruzione completa del segnale *PWM* del motore).
- **hardover:** Variabile booleana per il guasto di *Hardover* (blocco del *PWM* al valore massimo).

Il modulo consente di simulare i seguenti tipi di guasto:

- **Loss of Effectiveness:** Riduzione dell'efficacia del motore, proporzionale alla percentuale specificata dall'utente (*perc*).
- **Lock in Place:** Il motore si blocca sul valore *PWM* calcolato al passo di simulazione precedente, simulando un malfunzionamento meccanico o elettronico.

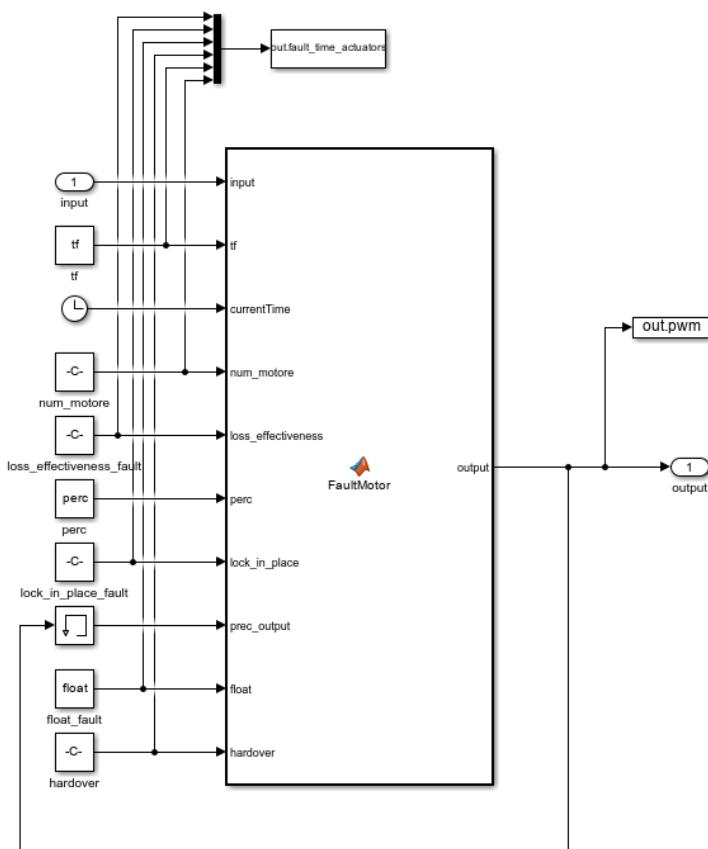


Figura 5.10: Struttura interna del modulo Fault Injection Attuatori

- **Float:** Il valore *PWM* viene portato ad un valore che oscilla attorno al suo punto di equilibrio pari a zero.
- **Hardover:** Il *PWM* del motore viene bloccato al valore massimo, simulando un comando errato o un corto circuito.

La struttura interna del modulo è mostrata in Figura 5.10. La logica dei guasti è implementata all'interno di una **MATLAB Function**, il cui codice è riportato in *Listing 5.3*. La funzione valuta il tipo di guasto attivo e modifica il segnale *PWM* del motore specificato da *num_motore*.

```

1 function output = FaultMotor(input, tf, currentTime, num_motore,
2 loss_effectiveness, ...
3 perc, lock_in_place, prec_output, float, hardover)
4
5 output = input;
6 if (currentTime >= tf)
7
8     if lock_in_place
9         if num_motore == 0
10            output = input;
11        else
12            output(num_motore) = prec_output(num_motore);
13        end
14    end
15
16    if float

```

```

16         if num_motore == 0
17             output = input ;
18         else
19             output(num_motore) = 0;
20         end
21     end
22
23     if hardover
24         if num_motore == 0
25             output = input ;
26         else
27             output(num_motore) = 1;
28         end
29     end
30
31     if loss_effectiveness
32         if num_motore == 0
33             output = input ;
34         else
35             output(num_motore) = ((100 - perc) / 100)* output(
36                 num_motore);
37         end
38     end
39 end

```

Listing 5.3: Fault Injection Actuators

Il modulo include una **GUI** (Figura 5.11) per configurare i parametri di guasto in modo intuitivo. L'utente può specificare:

- Il tempo di attivazione del guasto (*tf*).
- Il motore soggetto al guasto (*num_motore*).
- Il tipo di guasto (*loss_effectiveness*, *lock_in_place*, *float*, *hardover*).
- La percentuale di perdita di efficacia per il guasto di *Loss of Effectiveness*.

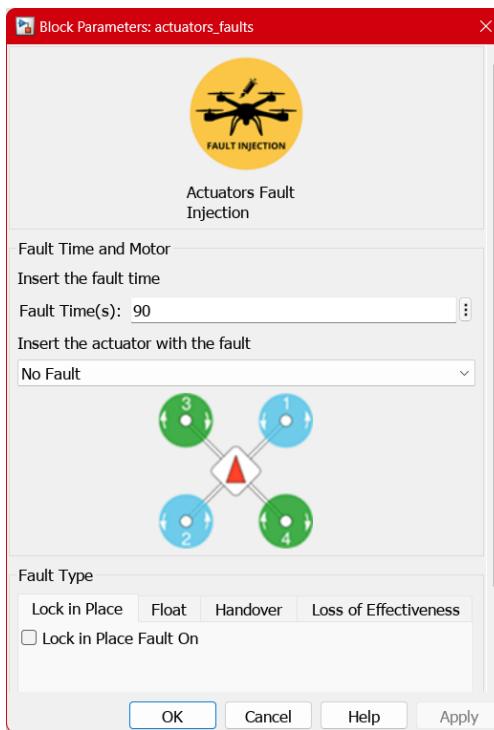


Figura 5.11: Maschera per l'inserimento dei parametri di guasto degli attuatori

5.1.4 Interfaccia Principale del Fault Injection

L'interfaccia principale del modulo di *Fault Injection* è progettata per fornire un punto di accesso centralizzato a tutti i moduli di iniezione dei guasti. Questa interfaccia è integrata nel blocco *UAV Dynamics Model*, come mostrato in Figura 5.13, e consente di navigare facilmente tra i diversi tipi di guasti configurabili per il sistema UAV.

L'interfaccia collega ciascun tipo di guasto (guasti all'IMU, GPS, attuatori) alle rispettive *GUI*, permettendo agli utenti di configurare rapidamente i parametri richiesti.



Figura 5.12: Modulo Fault Injection

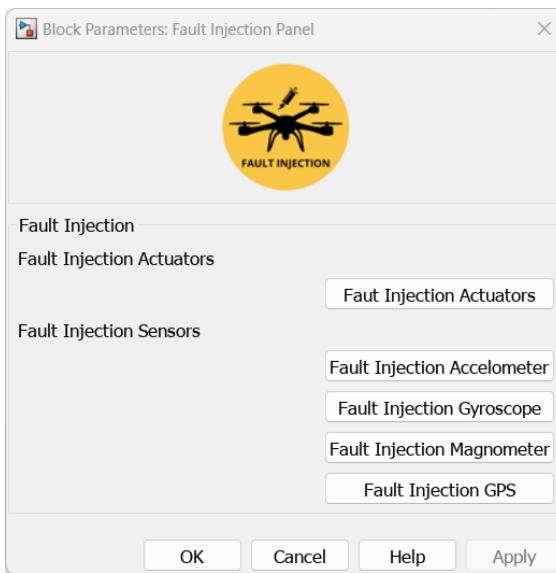


Figura 5.13: Interfaccia Principale del Modulo Fault Injection

5.2 Validazione del Modulo

Per validare il modulo di *Fault Injection*, sono state condotte simulazioni di volo in cui ogni tipologia di guasto è stata introdotta e analizzata singolarmente. L’obiettivo è verificare che ciascun guasto sia correttamente simulato e che il comportamento del sistema UAV corrisponda a quello atteso in presenza di un’anomalia.

La procedura della validazione è stata la seguente:

1. Esecuzione delle simulazioni:

Per ogni tipologia di guasto (ad esempio, *freezing*, *bias*, *loss of accuracy*), sono state configurate e avviate simulazioni specifiche, introducendo i guasti in momenti prestabiliti.

- I parametri del guasto, come il tempo di attivazione (t_f), l’asse interessato o il motore coinvolto, sono stati definiti tramite le GUI dei moduli di *Fault Injection*.
- Il comportamento dei sensori e attuatori è stato monitorato durante l’intero periodo di simulazione.

2. Salvataggio dei dati:

Al termine di ogni simulazione, i dati raccolti, come segnali dei sensori o risposte degli attuatori, sono stati salvati in un file *.mat*. Questi dati includono sia i segnali originali (senza guasti) sia quelli alterati dai guasti iniettati.

3. Analisi dei dati:

Tramite uno script MATLAB, i dati salvati sono stati analizzati graficamente. I plot generati consentono di osservare il comportamento delle variabili soggette a guasto e di verificare che il guasto sia stato iniettato correttamente.

Si riportano di seguito alcuni esempi di validazione.

Freezing sull’asse a_x

Per un guasto di tipo *freezing* lungo l’asse a_x dell’accelerometro, è stato verificato che il segnale si mantenga costante dopo l’attivazione del guasto, indipendentemente dal movimento

reale dell'UAV. I plot hanno mostrato chiaramente il momento di attivazione del guasto e il mantenimento del valore costante.

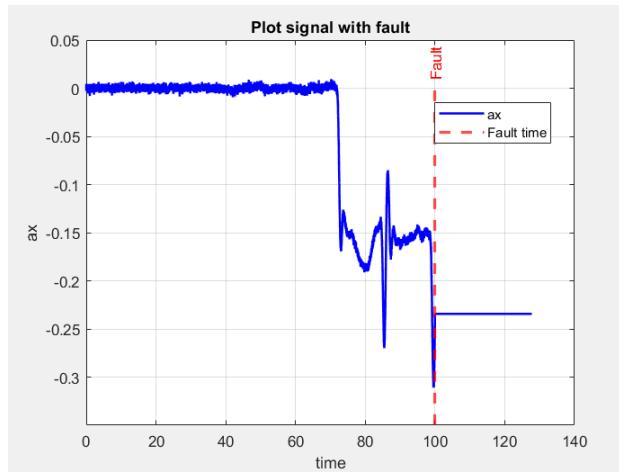


Figura 5.14: Freezing sull'asse a_x

Fault *bias* sull'asse *y* dell'accelerometro

Nel caso di un guasto di tipo *bias* introdotto sull'asse a_y , è stata osservata una deviazione costante aggiunta al segnale del sensore dopo il tempo di attivazione. Questo comportamento è stato confermato graficamente, mostrando che il guasto ha prodotto l'effetto previsto.

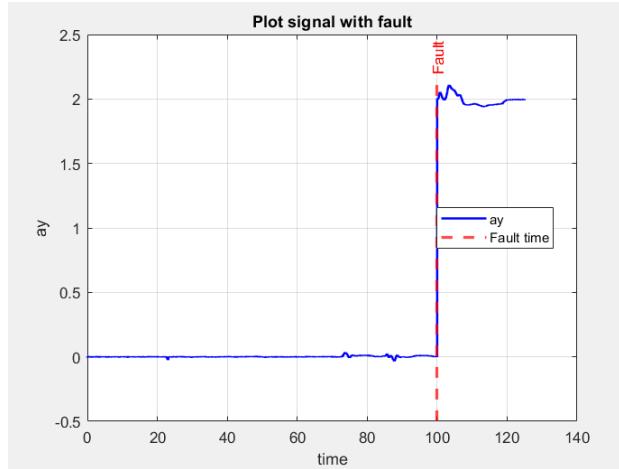


Figura 5.15: Fault *bias* sull'asse *y* dell'accelerometro

Drift Fault sull'asse *x* dell'accelerometro

Come test del guasto drift su un sensore è stato settato il lambda a valore 0.01 ed applicato all'asse x dell'accelerometro. Durante l'analisi post simulazione si osserva che l'accelerazione "misurata" ha un'andamento crescente nel tempo dopo l'avvenimento del guasto.

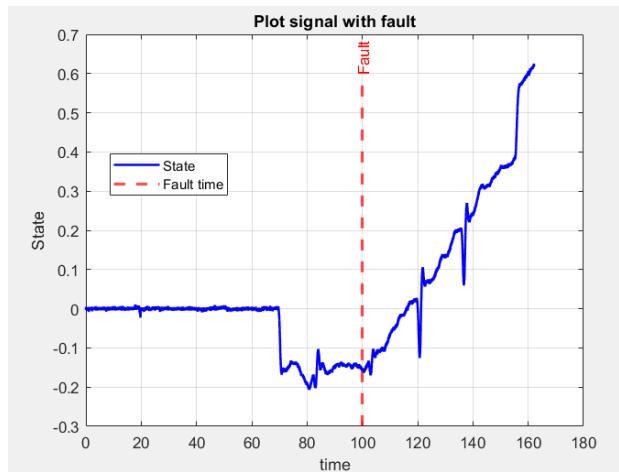


Figura 5.16: Drift Fault sull'asse x dell'accelerometro

Accuracy Fault sull'asse x del magnetometro

Per il guasto sull'accuratezza del segnale si è inherito un errore di accuratezza del 10% sull'asse x del magnetometro, ottenendo il risultato atteso. Stesso effetto è stato riscontrato per gli altri sensori dell'IMU e per il GPS.

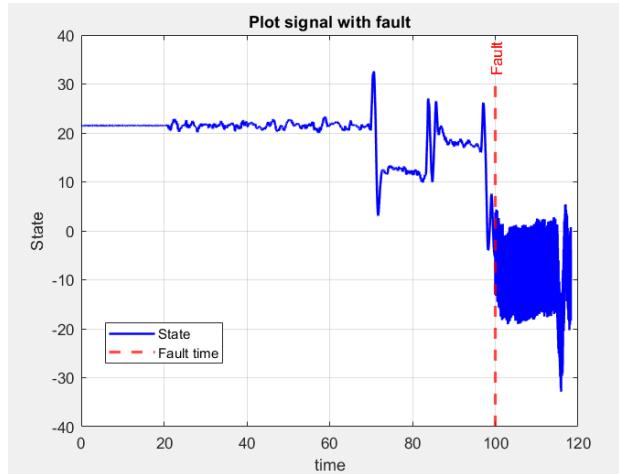


Figura 5.17: Loss of Accuracy Fault sull'asse x del magnetometro

Packet Loss GPS

Per la packet Loss del GPS si è impostata una perdita del 10% dei dati al momento del guasto. In Figura 5.18 è possibile notare che al momento del guasto la Latitudine inizia a perdere valori, vedendo in output valori Nan .

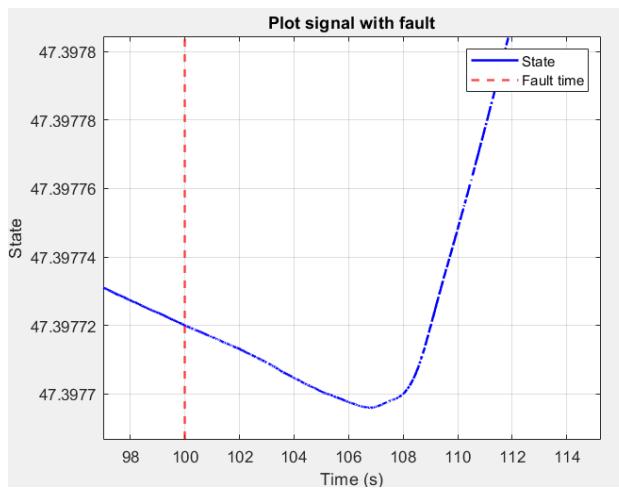


Figura 5.18: Packet Loss Latitudine

Hardover fault attuatore

Per i guasti introdotti nei motori, come l'*hardover*, è stato verificato che il segnale PWM fosse alterato correttamente secondo i parametri impostati. I plot hanno evidenziato le differenze rispetto al comportamento normale. L'inserimento di tale guasto ha portato il drone all'impatto con il terreno.

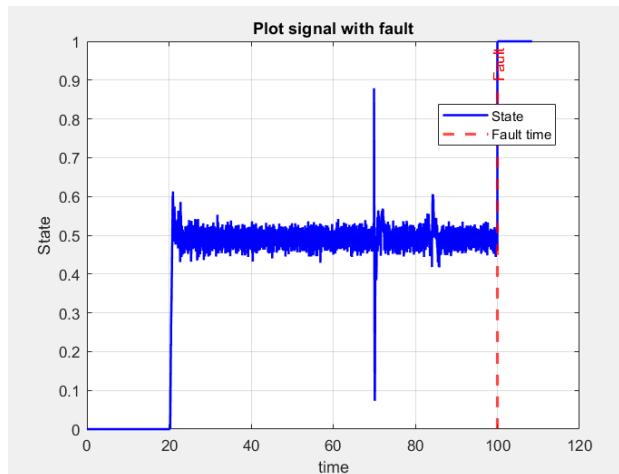


Figura 5.19: Hardover Fault Attuatore

Lock in Place Fault Attuatore

Introducendo il guasto di blocco del motore l'effetto sul PwM è quello mostrato in Figura 5.20 e che ha portato allo schianto del UAV.

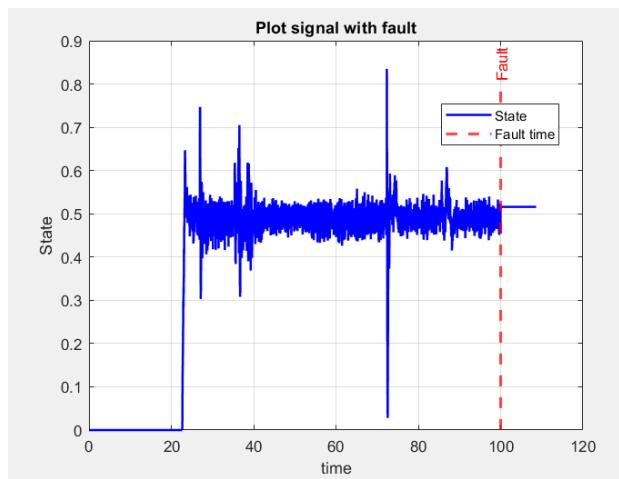


Figura 5.20: Lock in Place Fault Attuatore

Float Fault Attuatore per UAV Fixed Wing

Si è voluto anche riportare nella implementazione il guasto Float per gli UAV Fixed Wing. Quindi all'avvento del guasto l'attuatore si vede in input un PwM nullo (Figura 5.21).

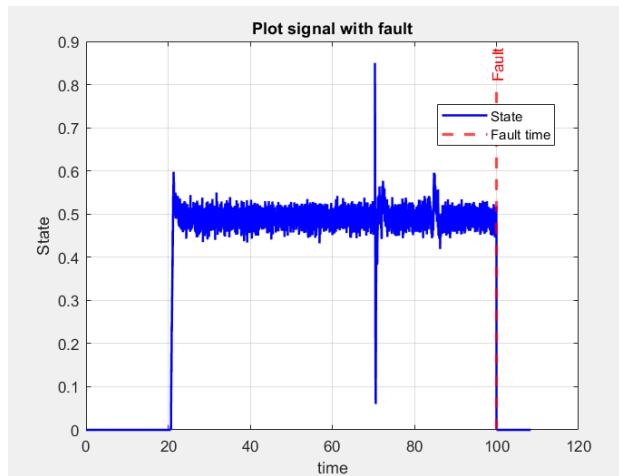


Figura 5.21: Float Fault Attuatore UAV Fixed Wing

Loss of Effectiveness Fault Attuatore

Per vedere l'effetto del guasto di perdita di efficienza del motore, si è posto a monte un log di dati dei PwM senza guasti. In tale maniera è possibile osservare in Figura 5.22 che il PwM desiderato, ovvero quello che arriva dall'uscita del controllore, cerca di compensare la perdita di efficienza fornendo un segnale più alto. In Figura WF indica il segnale PwM con Guasto (With Fault), mentre WOF indica il segnale PwM senza Guasto (WithOut Fault).

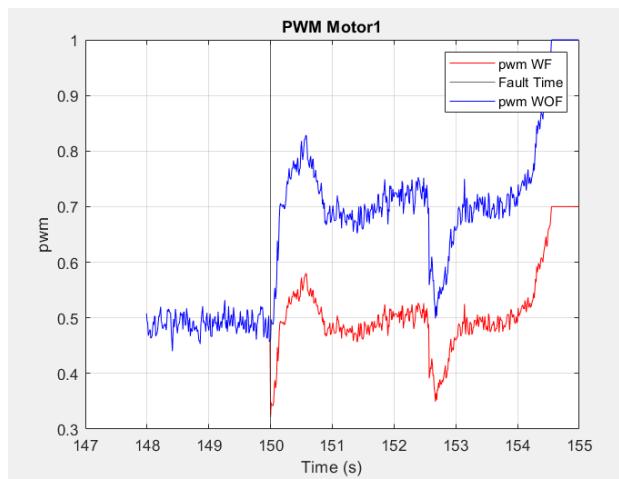


Figura 5.22: Loss of Effectiveness Fault Attuatore

L’analisi dei dati tramite grafici ha dimostrato che ogni guasto è stato correttamente iniettato e il comportamento risultante del sistema è conforme alle aspettative. Questo processo di validazione garantisce che i moduli di *Fault Injection* possano essere utilizzati per testare algoritmi di rilevamento e gestione dei guasti in modo affidabile ed efficace.

Capitolo 6

Modulo Diagnostico

Per fornire un esempio pratico di iniezione di guasti, in questo capitolo viene presentato lo sviluppo di un modulo di diagnosi basato su un modello di machine learning. Il capitolo è strutturato come segue:

- Definizione di quattro guasti specifici da diagnosticare.
- Simulazione di diversi scenari di volo per la raccolta dei dati necessari alla creazione del dataset.
- Esecuzione di una fase di ETL (Extract, Transform, Load) per la preparazione del dataset.
- Design ed Implementazione di un Modello di Machine Learning per la diagnosi guasti su Calcolatore per la simulazione della dinamica del drone.
- Design ed Implementazione di un Modello di Machine Learning per la diagnosi guasti su Microcontrollore.

6.1 Guasti da Diagnosticare

Per questo studio, si è scelto di concentrarci sulla diagnosi di guasti relativi ai sensori UAV. I sensori considerati sono l'IMU (comprendente accelerometro, giroscopio e magnetometro) e il GPS. Sono stati selezionati i seguenti guasti specifici:

- **Accelerometro:** Guasto di tipo *drift* sull'asse *x*.
- **Giroscopio:** Guasto di tipo *bias* sull'asse *y*.
- **Magnetometro:** Guasto di tipo *Loss of Accuracy* sull'asse *z*.
- **GPS:** Guasto di tipo *Packet Loss*.

Questa selezione permette di coprire una gamma di tipologie di guasto, inclusi quelli di tipo additivo, moltiplicativo, a rampa e a gradino. Per ciascun guasto sono state definite tre intensità: bassa, intermedia e alta. I parametri per ogni guasto sono:

- **Drift dell'accelerometro:** Valore del parametro λ (coefficiente di crescita del guasto) pari a 0.01, 0.05 e 0.1.
- **Bias del giroscopio:** Valore del bias in rad/s pari a 0.01, 0.04 e 0.07.
- **Loss of Accuracy del magnetometro:** Percentuale di perdita del segnale pari a 5%, 10% e 20%.
- **Packet Loss del GPS:** Percentuale di perdita di pacchetti pari a 5%, 10% e 20%.

6.1.1 Simulazione dei Voli

Per costruire un dataset diversificato e rappresentativo, i guasti sopra descritti sono stati simulati in tre diversi scenari di volo, come illustrato nelle Figure 6.1, 6.2 e 6.3.

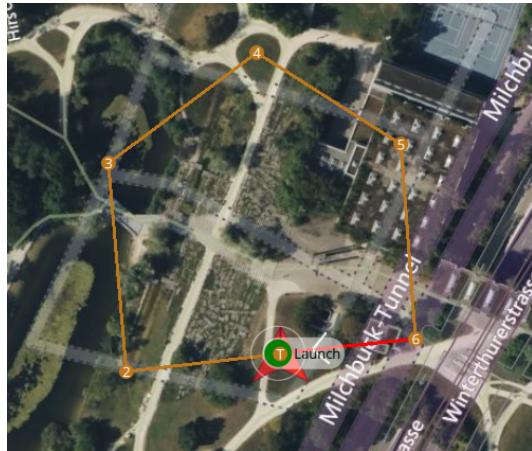


Figura 6.1: Volo 1

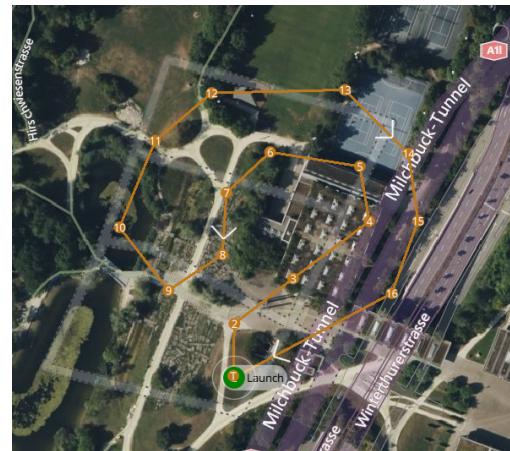


Figura 6.2: Volo 2

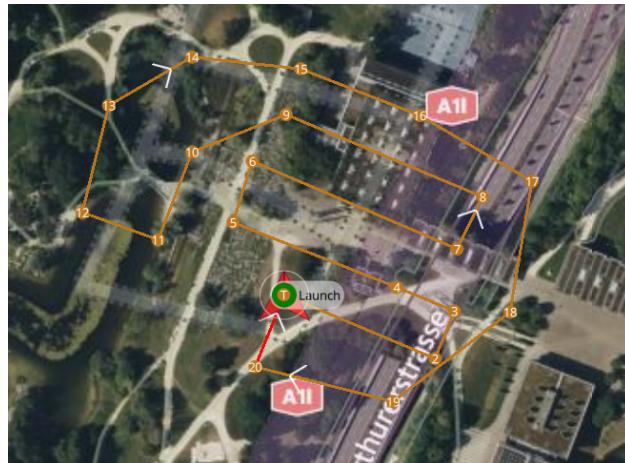


Figura 6.3: Volo 3

6.1.2 Raccolta e Bilanciamento del Dataset

Per garantire un dataset bilanciato e rappresentativo, sono stati simulati complessivamente 45 voli, comprendendo Voli senza guasti (baseline) e Voli con guasti a diverse intensità e in diverse configurazioni.

Questo approccio ha permesso di raccogliere dati che rappresentano sia condizioni normali che condizioni di guasto, offrendo una base solida per l'addestramento e la validazione dei modelli di machine learning.

6.2 ETL (Extract, Transform, Load)

Il processo di ETL è stato necessario per preparare i dati raccolti durante le simulazioni così da ottenere un dataset bilanciato adatto per l'addestramento del modello diagnostico. Il sistema prevede due simulazioni di Simulink: una per il monitor and tune del Flight Controller (FC) e un'altra per la simulazione della dinamica del drone (UAV Dynamics).

L'obiettivo principale è stato garantire la sincronizzazione dei dati tra le due simulazioni e prepararli per l'analisi successiva.

6.2.1 Sincronizzazione dei Dati

Il primo passo è stato quello di verificare il tempo di campionamento nelle due simulazioni.

- La simulazione della dinamica del drone rispetta il tempo di campionamento impostato a 100 Hz.
- La simulazione del Flight Controller, invece, non mantiene il tempo di campionamento di 100 Hz, registrando un valore effettivo di circa 65 Hz.

Per sincronizzare i dati, è stato necessario individuare un intervallo temporale comune utilizzando variabili condivise, nello specifico i valori dell'accelerometro. Il toolbox PX4 mette a disposizione un blocco uORB (Figura 6.4), che consente di ricevere nel Flight Controller i valori simulati dell'accelerometro generati dalla dinamica. Questo blocco è stato sfruttato per allineare temporalmente i dati.

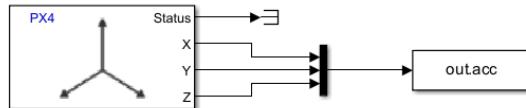


Figura 6.4: uORB

Quando i dati tra due serie temporali provenienti da fonti diverse non hanno la stessa frequenza di campionamento, è possibile modificare la frequenza dei timestamp nella raccolta dei dati. Si parla di *upsampling* (sovracampionamento) per aumentare la frequenza dei timestamp.

Una volta identificati i riferimenti temporali, è stata applicata una tecnica di *forward filling* per riempire i campioni mancanti nel dataset del Flight Controller, uniformandone il numero a quello della dinamica.

6.2.2 Selezione dei Dati Utili

Per i voli con guasto simulati, è stata estratta solo la porzione temporale in cui il guasto è stato effettivamente iniettato. Questo approccio riduce il rumore nei dati e focalizza l'analisi sui momenti critici.

Prima di procedere con l'estrazione delle feature e la selezione delle più rilevanti, sono stati calcolati gli errori di posizione tra i dati stimati e quelli desiderati. Questo passaggio è stato effettuato per evitare che il modello venga addestrato sulle specifiche traiettorie utilizzate durante la fase di training, garantendo così una maggiore generalizzazione sulle traiettorie non viste.

- È stato calcolato l'errore tra le posizioni lineari stimate (x_{stimato} , y_{stimato} , z_{stimato}) e quelle desiderate ($x_{\text{desiderato}}$, $y_{\text{desiderato}}$, $z_{\text{desiderato}}$):

$$e_x = x_{\text{desiderato}} - x_{\text{stimato}}$$

$$e_y = y_{\text{desiderato}} - y_{\text{stimato}}$$

$$e_z = z_{\text{desiderato}} - z_{\text{stimato}}$$

- È stato calcolato l'errore delle posizioni di assetto ($roll$, $pitch$, yaw):

$$e_{roll} = roll_{\text{desiderato}} - roll_{\text{stimato}}$$

$$e_{pitch} = pitch_{\text{desiderato}} - pitch_{\text{stimato}}$$

$$e_{yaw} = yaw_{\text{desiderato}} - yaw_{\text{stimato}}$$

Per i dati del GPS, latitudine e longitudine non sono stati considerati direttamente per evitare di addestrare il modello sulle traiettorie effettuate. Questi dati sono stati normalizzati per ridurre la variabilità e migliorarne l'analisi, utilizzando la seguente formula:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma},$$

dove x è il valore del segnale, μ e σ sono rispettivamente la media delle misurazioni e la deviazione standard del buffer di campioni considerato. Ogni 128 campioni di Latitudine inseriti in un apposito buffer, ne è stata calcolata la media e la varianza per la normalizzazione dei singoli segnali di quel buffer. Stesso procedimento per la Longitudine

6.2.3 Creazione della Tabella Finale

I dati sincronizzati e normalizzati della dinamica del drone (*UAV Dynamics*) e del Flight Controller sono stati uniti per ogni volo, generando una **timetable** finale. Ad ogni record della **timetable** è stato associato un codice di guasto (**faultCode**), che verrà utilizzato successivamente per l'addestramento supervisionato del modello di machine learning:

- 0: Nessun guasto.
- 1: Guasto di tipo drift lungo l'asse x dell'accelerometro.
- 2: Guasto di tipo bias lungo l'asse y del giroscopio.
- 3: Perdita di accuratezza lungo l'asse z del magnetometro.
- 4: Perdita di pacchetti del GPS.

L'estrazione delle feature è stata realizzata utilizzando il *Diagnostic Feature Designer* di MATLAB. Questo strumento richiede come input un formato tabellare, dove:

- La prima colonna contiene le **timetable** con i dati registrati durante i voli.
- La seconda colonna contiene i codici di guasto (**faultCode**) associati a ogni intervallo temporale.

6.2.4 Visualizzazione dei dati

Per valutare se sia possibile individuare in modo euristico delle feature significative per la diagnosi dei guasti, è stata effettuata una rappresentazione grafica dei dati del dataset finale. In particolare, sono stati analizzati i seguenti segnali:

- Accelerazione lungo l'asse x dell'IMU (a_x);
- Velocità angolare lungo l'asse y del giroscopio (ω_y);
- Campo magnetico lungo l'asse z del magnetometro (m_z);

- Latitudine normalizzata;
- Longitudine normalizzata.

Come mostrato nelle Figure 6.5 e 6.6, è possibile identificare visivamente le porzioni del dataset in cui è presente un determinato guasto e valutare il suo impatto sui segnali analizzati. Alcuni comportamenti caratteristici osservati sono i seguenti:

- Il **drift** dell'accelerometro lungo l'asse x mostra una crescita progressiva nel tempo, con valori finali dipendenti dall'intensità del guasto iniettato.
- Il **bias** introdotto nel giroscopio lungo l'asse y risulta meno evidente a un'analisi preliminare, suggerendo la necessità di un'elaborazione più approfondita per una corretta diagnosi.
- Il **loss of accuracy** del magnetometro lungo l'asse z si manifesta con un aumento della deviazione standard del segnale, proporzionale alla percentuale di accuratezza persa.
- La **perdita di pacchetti GPS** influenza direttamente sulla latitudine e longitudine normalizzate: in condizioni nominali, i valori si mantengono attorno allo zero, mentre all'aumentare della percentuale di perdita di pacchetti si osserva una riduzione della media dei valori registrati.

Questa analisi preliminare evidenzia la presenza di schemi ricorrenti nei dati acquisiti, suggerendo la possibilità di estrarre feature significative per la classificazione automatica dei guasti.

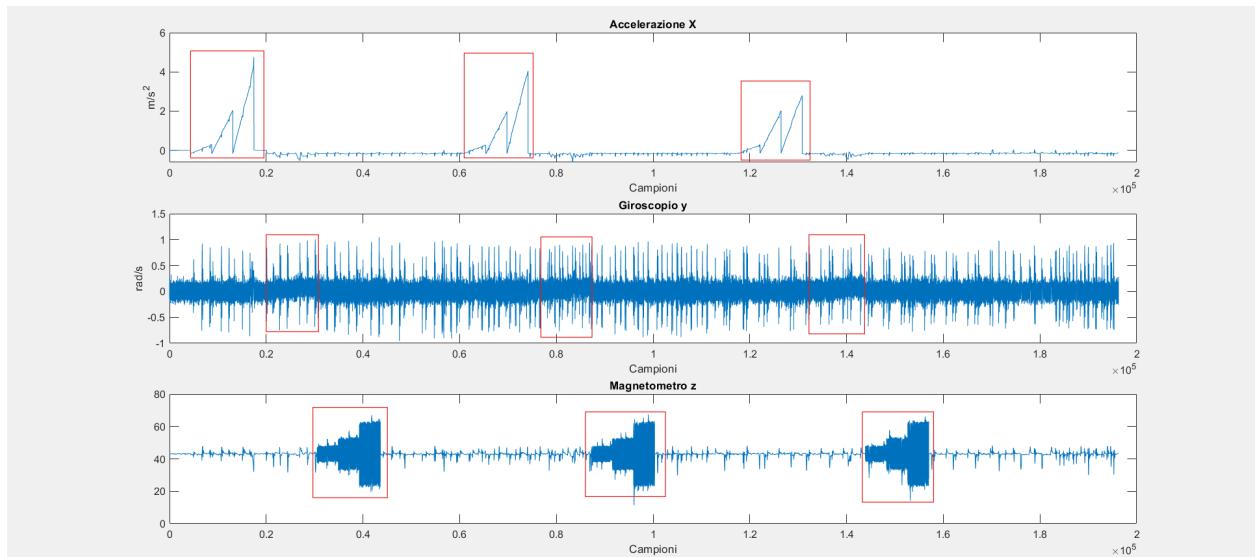


Figura 6.5: Visualizzazione di a_x , ω_y , m_z del dataset finale.

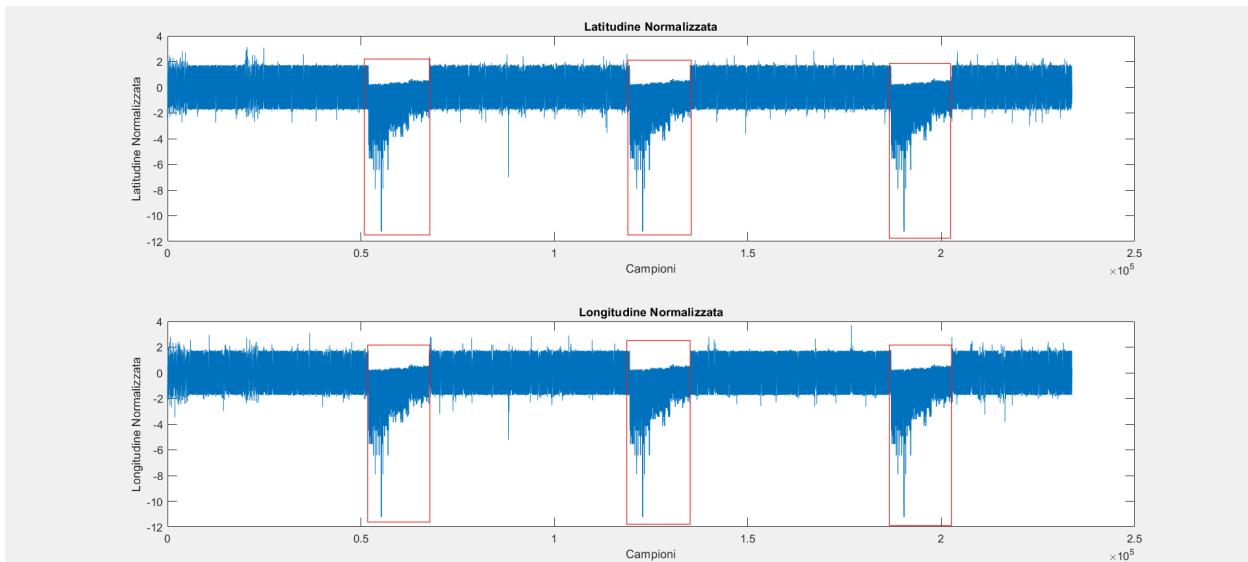


Figura 6.6: Visualizzazione della Latitudine e Longitudine normalizzate del dataset finale.

6.3 Design ed Implementazione di un Modello di Machine Learning su Calcolatore

L’obiettivo di questa fase è l’implementazione di un primo modello di diagnosi dei guasti nel calcolatore per la simulazione della dinamica del drone (*UAV Dynamics*). Integrare un modello di machine learning direttamente in questa simulazione equivale a simulare l’implementazione del modulo di diagnosi su un dispositivo Edge, come ad esempio una Raspberry Pi, una NVIDIA Jetson Nano o una Google Coral Dev Board. Assumendo di avere a disposizione una di queste schede, è possibile trasferire l’intera struttura del modulo diagnostico sulla scheda stessa tramite la generazione automatica di codice. Questo processo avviene attraverso l’uso di un transcompiler, che consente di tradurre il codice in un linguaggio compatibile con l’architettura del dispositivo target. Questi dispositivi sono particolarmente adatti per l’esecuzione di algoritmi di intelligenza artificiale, grazie alle loro capacità di calcolo ottimizzate e al supporto per accelerazione hardware. Questa implementazione nella dinamica del drone ci permetterà di effettuare un confronto tra le prestazioni di un modello implementato in un dispositivo edge ed un modello implementato in un microcontrollore.

6.3.1 Estrazione delle Feature

Come discusso nella Sezione 2.3, l’estrazione delle feature è stata effettuata utilizzando l’applicazione *Diagnostic Feature Designer* di MATLAB. Questo strumento consente la generazione di feature sia attraverso metodi predefiniti sia mediante strategie personalizzate. Tra le categorie di feature estratte si annoverano:

- **Statistiche di segnale:** media, deviazione standard, valore di picco, ecc.
- **Metriche non lineari:** skewness, kurtosis, ecc.
- **Metriche specifiche per macchinari rotanti:** Crest Factor, Clearance Factor, ecc.
- **Metriche spettrali:** potenza di banda, spettro di frequenza, ecc.

Per la generazione delle feature è stato impostato un criterio di segmentazione temporale, definendo una *frame policy* con una finestra temporale di 1.28 secondi e una frequenza di aggiornamento dei frame pari a 1.28 secondi.

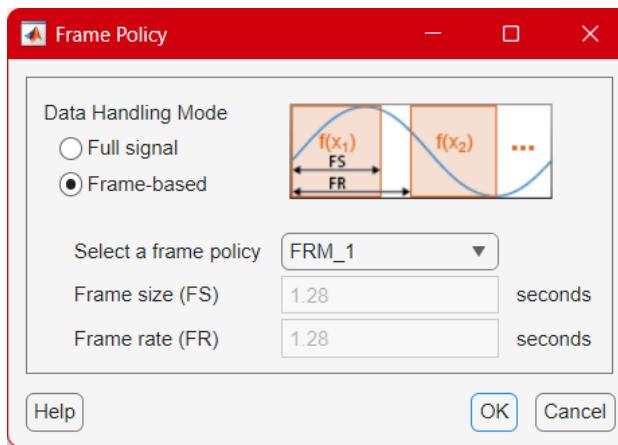


Figura 6.7: Definizione della Frame Policy.

Successivamente, per ogni segnale acquisito, sono state estratte le feature statistiche nel dominio del tempo. In seguito, sono state ricavate le feature nel dominio della frequenza mediante la generazione dello spettro di potenza di ciascun segnale. Il range di frequenza selezionato per l'analisi spettrale è compreso tra 1Hz e 50Hz , coerente con la banda informativa dei segnali registrati.

6.3.2 Classificazione e Selezione delle Feature

L'efficacia delle feature estratte è stata valutata attraverso tecniche di classificazione supervisionata, con lo scopo di identificare le più adatte alla discriminazione tra stati nominali e stati di guasto. In particolare, è stato applicato il **One-Way ANOVA Test**, un test statistico che consente di determinare se esistono differenze significative tra le medie di tre o più gruppi indipendenti. Questo test verifica l'ipotesi nulla H_0 secondo cui tutte le medie dei gruppi siano uguali, contro l'ipotesi alternativa H_1 , che afferma che almeno una media differisce dalle altre.

Sulla base dei risultati ottenuti, sono state selezionate le 21 feature con un *ranking score* normalizzato superiore a 0.4. Inoltre, sono state aggiunte due feature aggiuntive, selezionate in maniera euristica, per un totale di 23 feature finali, elencate in Tabella 6.1.

Gli errori di posizione ed assetto hanno ottenuto un *ranking score* normalizzato inferiore a 0.3, indicando una scarsa rilevanza per la classificazione dei guasti. Questo suggerisce che altre feature contengono informazioni più discriminanti per il modello di machine learning.

a_x	ω_y	m_z	Lat e Lon
Media	Media	Clearance Factor	Crest Factor
Valore di picco	Frequenza di picco	Crest Factor	Valore di picco
RMS	Potenza di banda	Impulse Factor	Shape Factor
Deviazione standard		Valore di picco	Skewness
		Shape Factor	Potenza di banda
		Deviazione standard	

Tabella 6.1: Feature selezionate per l'addestramento del classificatore su Calcolatore

6.3.3 Addestramento del Classificatore

Le feature selezionate sono state successivamente esportate nel tool *Classification Learner* di MATLAB, descritto nella Sezione 2.3. Questo strumento consente di addestrare diversi modelli di machine learning e deep learning e di confrontarne le prestazioni.

In questo lavoro, l'attenzione si è focalizzata su modelli ad albero decisionale, poiché questi risultano facilmente implementabili su microcontrollori embedded, garantendo al contempo una buona interpretabilità e prestazioni in tempo reale.

Il miglior modello ottenuto è stato un **Medium Tree**, che ha raggiunto le seguenti prestazioni:

- Accuratezza in validazione: **94%**.
- Accuratezza su dati di test: **93.7%**.
- Velocità di predizione: **33'000 osservazioni al secondo**.

La matrice di confusione del modello è riportata in Figura 6.8, evidenziando un'elevata capacità di classificazione per le diverse tipologie di guasto.

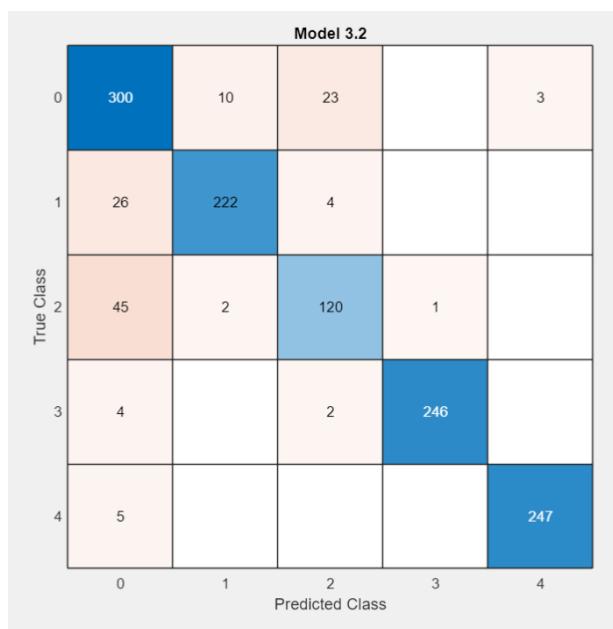


Figura 6.8: Matrice di confusione del modello Medium Tree per il Calcolatore

6.3.4 Implementazione del Classificatore

Per l'implementazione del classificatore, è stato necessario definire una struttura efficiente per la gestione dei dati in tempo reale. A tal fine, si è adottato l'utilizzo di **buffer circolari** per i segnali di accelerometro (a_x), giroscopio (ω_y), magnetometro (m_z), latitudine e longitudine. Questi buffer consentono di mantenere un flusso continuo di dati con un'allocazione di memoria costante, evitando problemi di overflow o inefficienza nella gestione degli input.

Descrizione dei Buffer Circolari

Un **buffer circolare** (o *ring buffer*) è una struttura dati che utilizza un'area di memoria fissa e la gestisce come se fosse circolare. È composto principalmente da due operazioni fondamentali:

- **Scrittura:** i nuovi campioni vengono inseriti in coda al buffer, sostituendo i dati più vecchi quando la capacità massima è raggiunta.
- **Lettura:** i dati vengono letti secondo l'ordine di arrivo, garantendo una gestione FIFO (*First-In, First-Out*).

Nel nostro caso, ogni buffer è stato configurato per immagazzinare **128 campioni** e aggiornarsi con uno *slide* a ogni nuovo campione ricevuto, garantendo così una finestra mobile di dati sempre aggiornata.

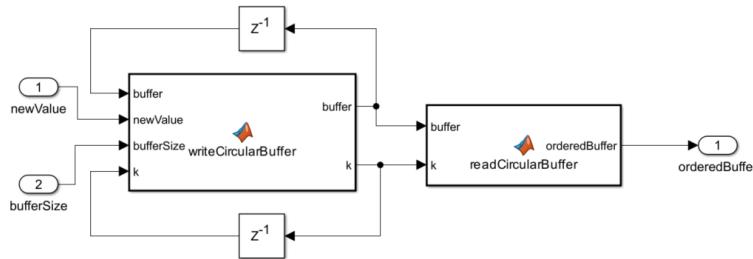


Figura 6.9: Struttura Simulink Buffer Circolare

Estrazione delle Feature e Inferenza del Modello

Una volta aggiornati i buffer circolari, i dati vengono passati a un **estrattore di feature**, il quale esegue l'estrazione delle feature nel **dominio del tempo** (statistiche come media, deviazione standard, ecc.) e l'estrazione delle feature nel **dominio della frequenza**, mediante **Trasformata di Fourier (FFT)**, necessaria per ottenere lo spettro di potenza del segnale.

Le feature calcolate vengono quindi organizzate in un vettore di input conforme al modello di machine learning precedentemente addestrato nel *Classification Learner*.

Trascrizione dell'albero decisionale

Per ottimizzare il consumo di memoria del calcolatore, non è stato caricato direttamente il modello di machine learning addestrato, bensì si è optato per una **conversione manuale in codice if-else**. Questa scelta è stata motivata dalla necessità di ridurre il footprint di memoria, in quanto il modello decisionale compatto dell'albero occupava circa **14 KB**, mentre la trascrizione in codice if-else ha ridotto la dimensione a **4 KB**.

La conversione è stata effettuata mediante uno *script MATLAB* che ha analizzato la struttura dell'albero decisionale e generato il codice corrispondente, rappresentando ogni nodo dell'albero come una condizione **if** con la relativa label di classificazione.

Filtraggio delle Etichette con Buffer Circolare

Poiché il Modello di Machine Learning genera una nuova label di classe di guasto ogni **0.01 secondi**, è emerso un problema di **falsi positivi** dovuti alla variabilità istantanea delle predizioni. Per mitigare questo effetto, è stato implementato un ulteriore **buffer circolare per le etichette**, della dimensione di **500 campioni** (equivalenti a **5 secondi** di simulazione). Il codice è riportato in *Listing 6.1*.

```

1 function filteredLabel = filterUsingBuffer(currentLabel)
2 %#codegen
3
4 bufferSize = 150; % Dimensione del buffer
5 maxLabels = 5; % Assumiamo un numero massimo di etichette
6 conosciuto

```

```

7      % Variabili persistenti per memorizzare il buffer e l'indice
8      % corrente
9      persistent labelBuffer bufferIndex labelCounts initialized
10
11      % Inizializza il buffer alla prima esecuzione
12      if isempty(initialized)
13          labelBuffer = zeros(bufferSize, 1); % Buffer inizializzato a
14          % 0
15          bufferIndex = 1; % Indice per la scrittura nel buffer
16          labelCounts = zeros(maxLabels, 1); % Contatore fisso per le
17          % etichette
18          initialized = true;
19      end
20
21      % Rimuove l'etichetta precedente dal conteggio
22      oldLabel = labelBuffer(bufferIndex);
23      if oldLabel > 0 && oldLabel <= maxLabels
24          labelCounts(oldLabel) = labelCounts(oldLabel) - 1;
25      end
26
27
28      labelBuffer(bufferIndex) = currentLabel; % Inserisce l'etichetta
29      % corrente nel buffer in modo circolare
30      bufferIndex = mod(bufferIndex, bufferSize) + 1; % Incrementa 1'
31      % indice (circolare)
32
33      % Aggiunge la nuova etichetta al conteggio
34      if currentLabel > 0 && currentLabel <= maxLabels
35          labelCounts(currentLabel) = labelCounts(currentLabel) + 1;
36      end
37
38      % Trova l'etichetta con il massimo conteggio
39      [~, filteredLabel] = max(labelCounts);
40  end

```

Listing 6.1: Label Filter

Il processo di filtraggio avviene nel seguente modo:

1. Ad ogni iterazione, la nuova etichetta predetta viene inserita nel buffer nella posizione indicata da `bufferIndex`.
2. Si calcolano tutte le etichette uniche presenti nel buffer.
3. Si contano le occorrenze di ciascuna etichetta utilizzando un ciclo e la funzione `sum`.
4. L'etichetta più frequente nel buffer viene determinata con la funzione `max` e restituita come output definitivo.

Questo approccio consente di stabilizzare la predizione del modello, migliorando l'affidabilità della classificazione ed evitando variazioni indesiderate dovute a rumore o transizioni rapide nei dati (Figura 6.11).

Validazione in Volo e Risultati su Calcolatore

Unendo tutti i componenti descritti precedentemente per l'implementazione del classificatore, quello che si ottiene è la struttura riportata in Figura 6.10, in cui si hanno 5 buffer 5 circolari,

uno per ogni segnale da passare all'estrattore delle feature. L'estrattore delle Feature calcola il vettore delle feature da passare all'albero che effettua un classificazione. L'output ottenuto viene fornito in input al filtraggio descritto precedentemente.

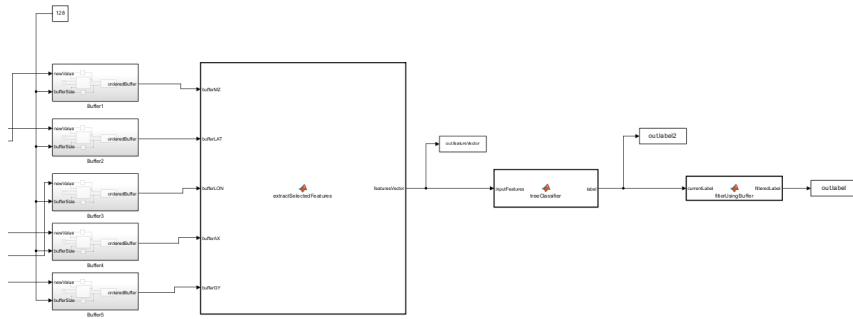


Figura 6.10: Struttura del modulo diagnostico in un calcolatore

L'implementazione è stata testata con un **test di volo** utilizzando una traiettoria differente da quelle impiegate per la generazione del dataset, al fine di verificare la capacità del classificatore di generalizzare a nuovi scenari. Per la validazione, è stato selezionato un guasto specifico: **perdita di accuratezza nell'asse z del magnetometro**. L'individuazione del guasto e la sua classificazione è avvenuta con successo. Si possono notare anche gli effetti del filtraggio effettuato dalla Figura 6.11, in cui vengono tolti la maggior parte dei falsi positivi.

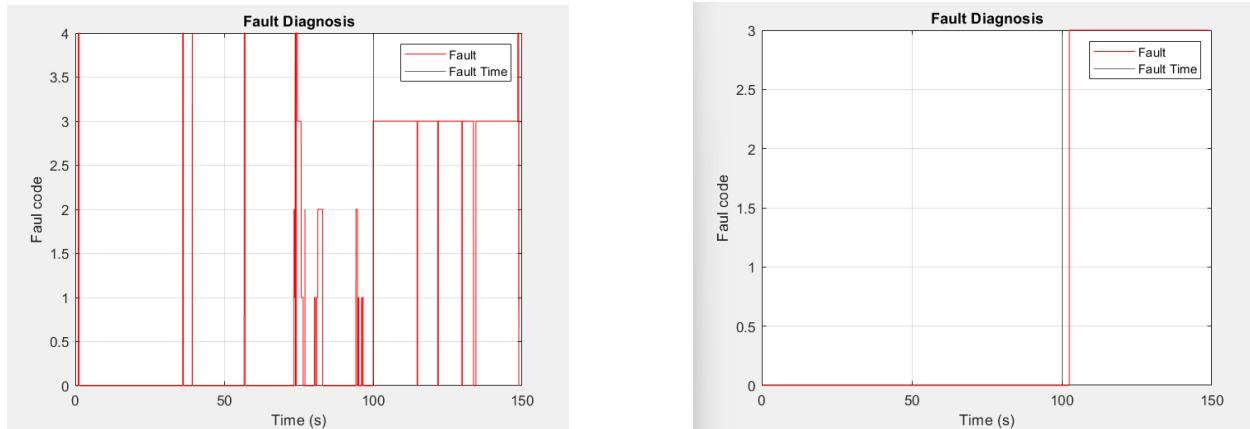


Figura 6.11: Confronto tra rilevamento guasti senza filtraggio (sinistra) e con filtraggio di 5 secondi (destra)

6.4 Design ed Implementazione di un Modello di Machine Learning su Dispositivo Microcontrollore

Per evitare l'utilizzo su un UAV di una scheda edge come Raspberry Pi, che potrebbe aumentare il peso del drone, aumentare il consumo della batteria e i costi, si può pensare di trovare un compromesso, inserendo un modulo di diagnosi basato su machine learning, all'interno del Flight Controller.

È importante considerare che l'intero firmware caricato nel Flight Controller occupa quasi il 94% della memoria disponibile e che il tempo di campionamento massimo raggiungibile è pari a 65 Hz, mentre la dinamica simulata opera a 100 Hz. Questo disallineamento nei tempi

di campionamento ha impedito l'implementazione del modello di machine learning addestrato per un Calcolatore, poiché avrebbe compromesso la corretta rilevazione dei guasti basati su feature nel dominio della frequenza, come la perdita di accuratezza del magnetometro e la perdita di pacchetti del GPS.

Alla luce di queste limitazioni, si è deciso di addestrare un nuovo modello di machine learning, adattato alle specifiche del Flight Controller. A tal fine, è stato necessario eseguire un *downsampling* del dataset a 65 Hz mediante uno script MATLAB. Dopo questa operazione, si è proceduto alla normalizzazione delle coordinate di longitudine e latitudine, come descritto nella Sezione 6.2.2.

Successivamente, si è ripetuto il processo di estrazione delle feature, aumentando la dimensione del frame (*frame size*) e la frequenza di aggiornamento (*frame rate*) a 2 secondi. Dopo il calcolo del ranking delle feature e la loro selezione, si è scelto di mantenere gli stessi segnali utilizzati per l'implementazione su Calcolatore. In questo caso, sono state considerate le 24 feature riportate in Tabella 6.2.

Anche per il modello di machine learning del microcontrollore, non sono state considerate le feature derivanti dagli errori lineari e di assetto che hanno ricevuto un *ranking score* al di sotto a 0.3. Poiché l'implementazione del modello avviene su un microcontrollore con risorse limitate, è importante ridurre il numero di feature per minimizzare il carico computazionale. Escludere feature poco rilevanti aiuta a ottimizzare memoria e tempi di inferenza.

a_x	ω_y	m_z	Lat e Lon
Media	Media	Clearance Factor	Crest Factor
Valore di picco	Frequenza di picco	Crest Factor	Valore di picco
RMS		Impulse Factor	Shape Factor
Deviazione standard		Valore di picco	Skewness
		Shape Factor	Potenza di banda
		Deviazione standard	
		Frequenza di picco	

Tabella 6.2: Feature selezionate per l'addestramento del classificatore su dispositivo microcontrollore

Dopo l'esportazione delle features selezionate nel Classification Learner si sono addestrati i modelli di machine learning basati su albero decisionale, per la loro bassa complessità computazionale e la loro semplicità. La complessità dei decision trees durante la fase di inferenza è relativamente efficiente in termini di tempo. Per classificare una nuova istanza, la complessità temporale è $O(d)$, dove d è la profondità del decision tree. Questo perché, al peggio, è necessario attraversare l'intero albero dalla radice a una foglia.

Il modello di albero decisionale ottenuto ha avuto le seguenti metriche:

- Accuratezza in validazione: **90%**.
- Accuratezza su dati di test: **85%**.
- Velocità di predizione: **20'000 osservazioni al secondo.**

La matrice di confusione del modello è riportata in Figura 6.12, evidenziando un peggioramento rispetto al modello di machine learning addestrato per il Calcolatore della dinamica del drone.

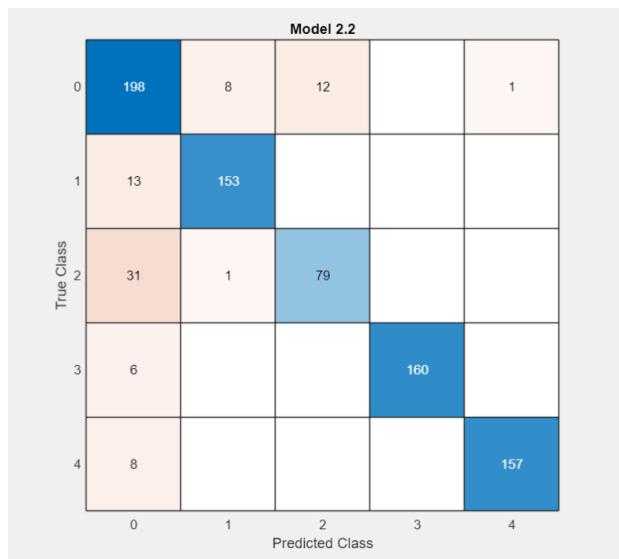


Figura 6.12: Matrice di confusione del modello Medium Tree per il dispositivo Microcontrollore

6.4.1 Implementazione del Classificatore su Microcontrollore

L'implementazione dell'albero decisionale sul *flight controller* è stata realizzata in modo analogo a quanto discusso nella Sezione 6.3.4. Tuttavia, alcune modifiche sono state necessarie per adattare il modello ai vincoli hardware e software del microcontrollore.

Per garantire una maggiore stabilità nel riconoscimento delle condizioni operative, si è aumentata la dimensione dei **buffer circolari** dei segnali, passando da **128 a 130 campioni**. Questa variazione consente di migliorare la capacità di catturare le dinamiche del segnale senza compromettere la reattività del sistema.

Il filtraggio delle etichette, già introdotto nella Sezione 6.3.4, è lo stesso riportato in *Listing 6.1*). Questo assicura che la gestione delle label avvenga in modo deterministico e compatibile con le restrizioni del codice embedded.

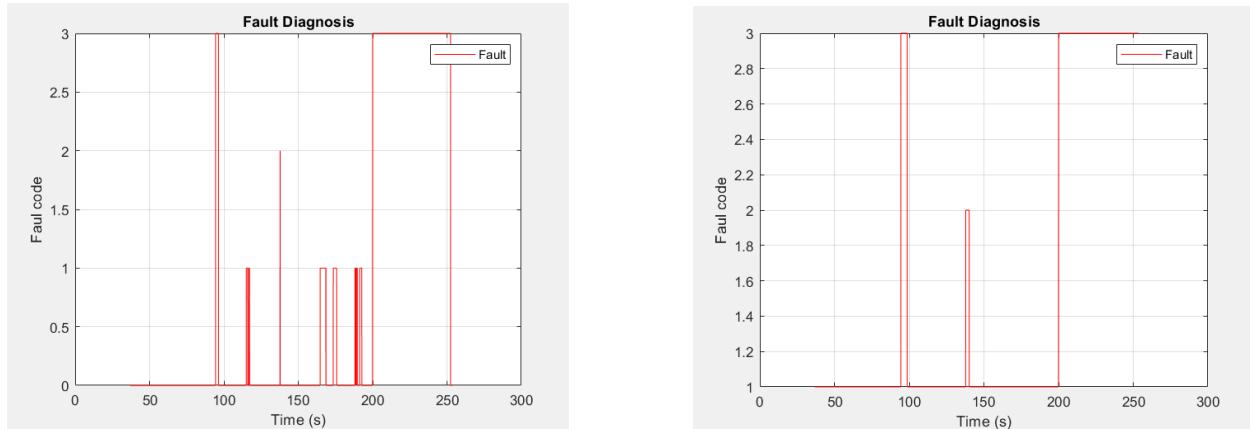


Figura 6.13: Confronto tra rilevamento guasti senza filtraggio (sinistra) e con filtraggio di 5 secondi (destra)

6.4.2 Validazione in Volo e Risultati su Flight Controller

Per la validazione del modulo di diagnosi inserito, si è eseguito lo stesso volo e con stessa tipologia di guasto. I risultati ottenuti dal sistema di rilevamento dei guasti sono riportati in Figura 6.13, dove:

- **A sinistra:** identificazione dei guasti **senza filtraggio**, che mostra una maggiore variabilità nelle predizioni.
- **A destra:** identificazione dei guasti **con filtraggio della durata di 5 secondi** (333 labels filtrate), evidenziando una maggiore stabilità nella classificazione.

Per eliminare del tutto i falsi positivi residui, è possibile aumentare la dimensione della finestra di filtraggio in base alle proprie esigenze.

Capitolo 7

Conclusioni e Sviluppi Futuri

Uno degli aspetti chiave della Tesi è stato lo sviluppo di un **toolbox per la Fault Injection**, progettato per introdurre guasti simulati nei sensori dell'UAV e analizzarne l'impatto sulle prestazioni di un classificatore. Questo strumento ha permesso di replicare guasti tipici, come la perdita di accuratezza del magnetometro, la perdita di pacchetti GPS e il drift dell'accelerometro, fornendo un ambiente controllato per valutare l'affidabilità del modello diagnostico.

La valutazione di tale toolbox ha previsto la progettazione di un'architettura diagnostica basata su algoritmi di machine learning, con un'attenzione particolare alla selezione delle feature più efficaci per il riconoscimento dei guasti. Attraverso l'uso di strumenti software come MATLAB e la sua applicazione *Diagnostic Feature Designer*, sono state estratte e selezionate feature significative per discriminare tra condizioni operative nominali e difettose. L'analisi è stata validata tramite il test statistico *One-Way ANOVA*, permettendo di identificare le feature più rilevanti. Tali feature identificate sono state utilizzate per addestrare *Classificatori ad Albero Decisionale* tramite il *Classification Learner* di MATLAB.

L'implementazione del classificatore è stata eseguita dapprima utilizzando il calcolatore per la simulazione della dinamica del drone, in maniera simile a quanto si farebbe con un dispositivo edge, e successivamente adattata per l'integrazione diretta su un flight controller. A causa dei vincoli computazionali del microcontrollore, è stato necessario effettuare un downsampling del dataset, rivedere la selezione delle feature e ottimizzare il modello decisionale, convertendolo in una struttura *if-else* per ridurre il consumo di memoria. Inoltre, per garantire una maggiore affidabilità nella classificazione, è stata introdotta una tecnica di filtraggio basata su buffer circolari, in grado di ridurre i falsi positivi e migliorare la stabilità delle predizioni.

L'iniezione di guasti ha evidenziato l'importanza di un'adeguata selezione delle feature e della calibrazione dei parametri di filtraggio delle predizioni. Senza un'adeguata finestra di filtraggio, il sistema tendeva a generare falsi positivi, riducendo l'affidabilità della diagnosi. L'implementazione del filtro su buffer circolare ha migliorato la stabilità del sistema, permettendo di ottenere risultati più coerenti nel tempo.

L'accuratezza del modello diagnostico è stata valutata su due piattaforme hardware distinte:

- **Calcolatore:** il modello implementato sul calcolatore per la simulazione della dinamica del drone, che emula un dispositivo Edge a bordo del velivolo, ha raggiunto un'accuratezza del **93.7%** sui dati di test. Questo risultato è stato possibile grazie all'utilizzo di feature più complesse e a un modello decisionale più sofisticato.
- **Microcontrollore:** a causa delle limitazioni computazionali e di memoria, il modello è stato semplificato e adattato in una struttura *if-else*. Nonostante l'ottimizzazione,

l'accuratezza è scesa all'**85%**, principalmente a causa del minor numero di feature disponibili e della ridotta frequenza di campionamento (da 100 Hz a 65 Hz).

Questa differenza di prestazioni evidenzia la necessità di bilanciare la complessità del modello con le risorse hardware disponibili. Sebbene i dispositivi edge offrano una maggiore potenza di calcolo e quindi una diagnosi più accurata, l'implementazione su microcontrollore è cruciale per applicazioni real-time a bordo di UAV con vincoli di peso, consumo energetico e memoria.

Per migliorare ulteriormente il sistema diagnostico, futuri sviluppi potrebbero includere:

- L'integrazione di tecniche di *Fault Tolerant Control* (FTC) per rendere il drone in grado di reagire autonomamente ai guasti rilevati.
- L'ottimizzazione dell'implementazione su microcontrollore attraverso tecniche di compressione del modello e quantizzazione delle feature.
- L'ampliamento del dataset includendo una gamma più estesa di guasti per migliorare la robustezza del sistema diagnostico.
- L'uso di tecniche di apprendimento automatico avanzate, come le reti neurali leggere (*TinyML*), per migliorare l'accuratezza della diagnosi mantenendo un basso impatto computazionale.
- L'ottimizzazione della gestione della memoria nei microcontrollori per migliorare la scalabilità del modello diagnostico senza compromettere le prestazioni.

In conclusione, il lavoro svolto ha dimostrato l'efficacia di un approccio basato su machine learning per la diagnosi guasti in UAV, evidenziando al contempo le sfide legate all'implementazione su hardware con risorse limitate. La metodologia adottata, combinando simulazione, test su dispositivi edge e implementazione su microcontrollore, rappresenta un contributo significativo per lo sviluppo di sistemi diagnostici affidabili e applicabili in scenari reali.

Appendice A

Configurazione dell'Ambiente di Sviluppo il Sistema PX4 Hardware-in-the-Loop (HIL)

A.1 Configurazione Toolchain Cygwin e Download Codice Sorgente PX4

Questa sezione spiega il compito da completare come parte del passo "Configurazione del Toolchain Cygwin e Download del Codice Sorgente" del processo di configurazione hardware (utilizzando le schermate di configurazione hardware).

Nota: Assicurati che il tuo PC sia connesso a una connessione internet attiva prima di procedere con questo passaggio. Per configurare il toolchain Cygwin™ e scaricare il codice sorgente PX4® utilizzato nel pacchetto di supporto per PX4 Autopilots nel UAV Toolbox, seguire questi passaggi:

Scaricare la versione 0.8 di PX4 Cygwin Toolchain MSI Installer, che è compatibile con PX4 Firmware v1.12.3, disponibile a questo link: <https://github.com/PX4/PX4-windows-toolchain/releases/tag/v0.8>. Nota: Il pacchetto di supporto UAV Toolbox per PX4 Autopilots supporta solo la versione v0.8 di PX4 Windows® Cygwin Toolchain MSI Installer, anche se una versione più recente potrebbe essere disponibile.

Eseguire il programma di installazione MSI e avviare l'installazione del toolchain (Fig. A.1).

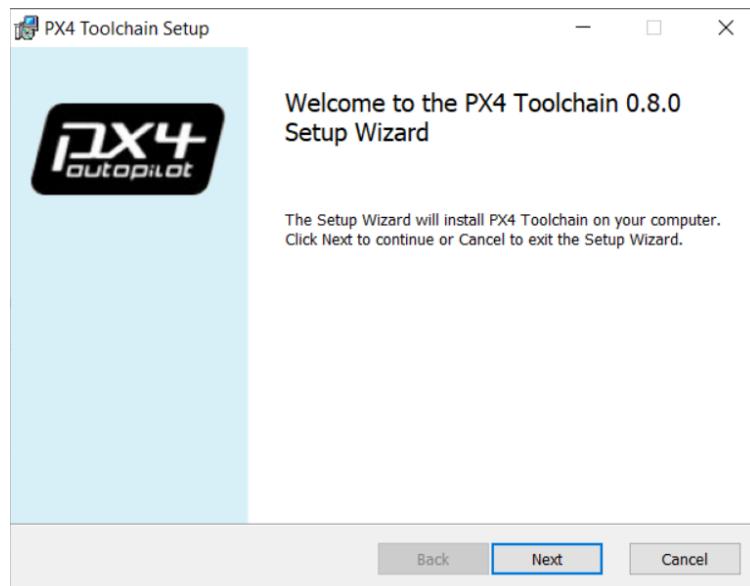


Figura A.1: PX4 Toolchain Setup

Cambiare la cartella di installazione per Cygwin in una qualsiasi cartella locale (ad esempio, C:\px4), quindi fare clic su OK (Fig. A.2).

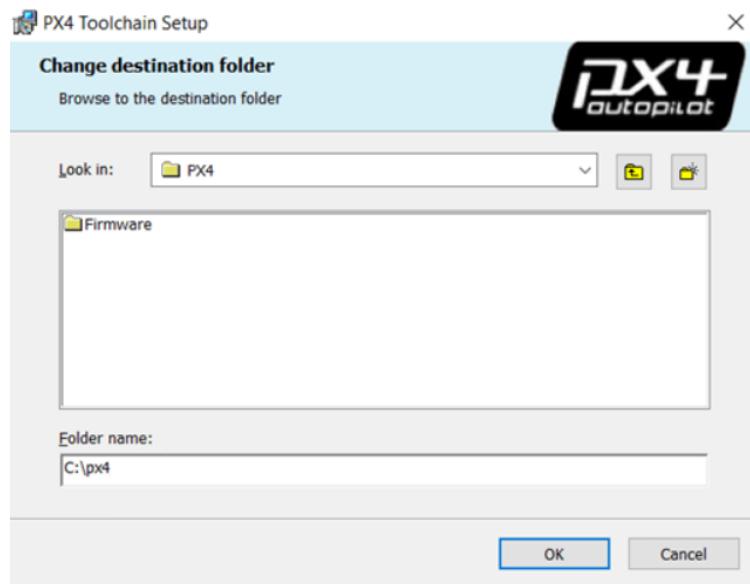


Figura A.2: Selezione della cartella di installazione

All’ultimo passaggio della procedura guidata di configurazione del toolchain PX4, fare quanto segue:

- Se non si dispone del Codice Sorgente PX4 (Firmware Autopilota PX4 v1.12.3) scaricato nel computer host, selezionare l’opzione "Clona repository PX4 e Avvia Simulazione", quindi fare clic su Fine. Questa opzione clona il firmware PX4 attuale. Quando si fa clic su Verifica Installazione nel passaggio 6 di seguito, il firmware viene automaticamente controllato alla versione v1.12.3.
- Se il Codice Sorgente PX4 (Firmware Autopilota PX4 v1.12.3) è già disponibile nel computer host, fare clic su Fine senza selezionare l’opzione "Clona repository PX4 e Avvia Simulazione" (Fig. A.3).

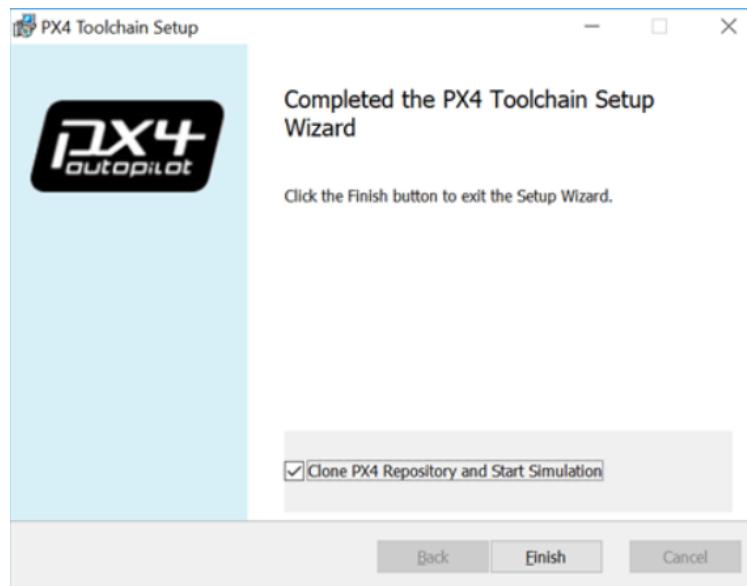


Figura A.3: Clona repository PX4 e Avvia Simulazione

Se si è selezionata l'opzione "Clona repository PX4 e Avvia Simulazione" e si è fatto clic su Fine, viene avviata una shell bash che avvia la clonazione del firmware (Fig. A.4).

```
bash --login -c trap bash SIGINT; git clone --recursive -j8 https://github.com/PX4...
Cloning into 'Firmware'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (55/55), done.
Receiving objects: 33% (88391/267849), 40.22 MiB | 1.13 MiB/s
```

Figura A.4: Shell bash che avvia la clonazione del firmware

Attendere il completamento della clonazione del firmware. Dopo che il firmware è stato clonato, la Simulazione viene avviata in jMAVSim. È possibile chiudere la shell bash in questa fase.

Il firmware PX4 viene clonato all'interno di una cartella denominata home, all'interno della cartella Cygwin selezionata durante l'installazione (ad esempio, C:\px4\home).

A.2 Configurazione Firmware PX4 per l'Hardware-in-the-Loop

Ora si andranno ad elencare i passaggi per la generazione del firmware PX4 utilizzando il supporto per PX4 Autopilots nel UAV Toolbox. Questo firmware permetterà poi di eseguire la simulazione HIL e verificare gli algoritmi di controllo implementati usando Matlab/Simulink.

Selezione del Firmware PX4 e del Hardware di Destinazione:

Accedi alla schermata di selezione del firmware PX4 e del hardware di destinazione nel UAV

Toolbox Support Package for PX4 Autopilots. Clicca su "Manage" e successivamente su "setup". Una volta aperto l'Hardware Setup, comparirà la schermata mostrata in Figura che chiederà se Python 3.8.2 è installato sul PC.

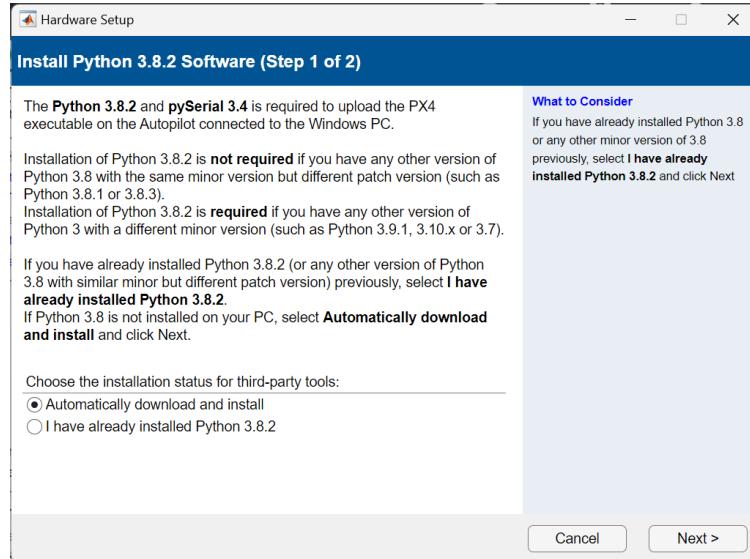


Figura A.5: Pyhton 3.8.2

Se non è installato, selezionare "Automatically download and install". Una volta fatto, comparirà una schermata. Premere su "Validate" ed attendere che esca la scritta "Python 3.8.2 validation successful".

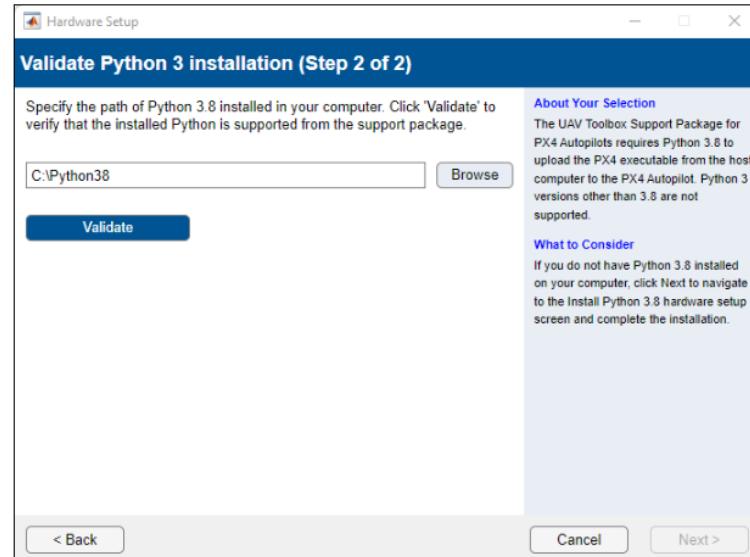


Figura A.6: Validare Python 3.8.2

In caso positivo, clicca "Next". Successivamente, l'Hardware Setup chiederà di inserire il percorso utilizzato per l'installazione del toolchain Cygwin (stesso percorso della Figura A.2), quindi fare clic su Verifica Installazione (Fig. A.7). Una volta uscita una spunta verde, clicca Next.

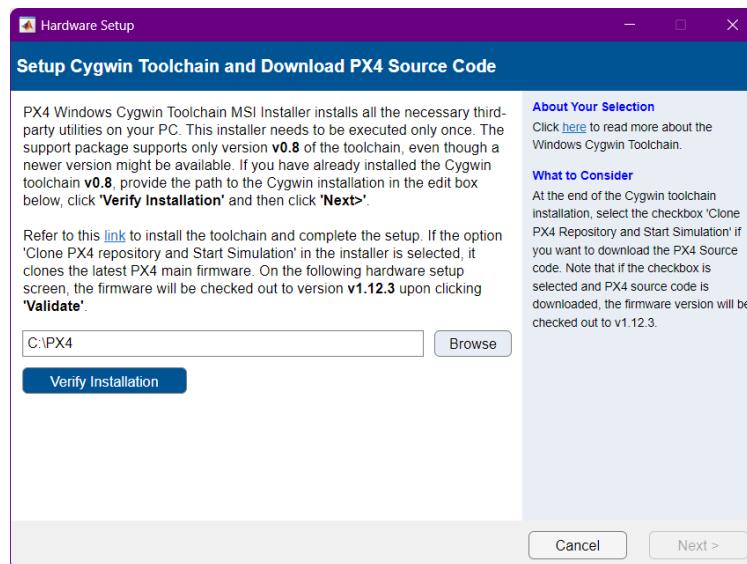


Figura A.7: Verifica Percorso

Selezionare la cartella di download per il codice sorgente della PX4 (Fig. A.8). Aspettare che esca la scritta "Firmware validation successful" e clicca "Next".

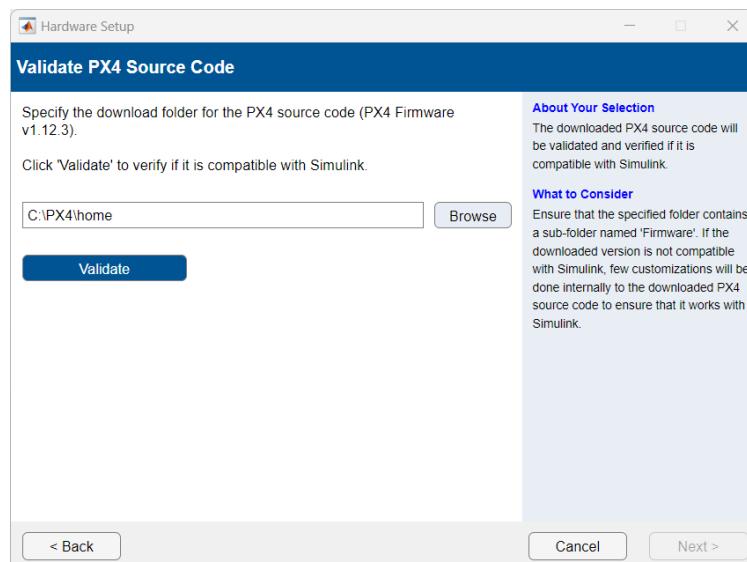


Figura A.8: Cartella di download per il codice sorgente della PX4

Selezionare "Design Flight Controller in Simulink" e clicca "Next" (Fig. A.9)

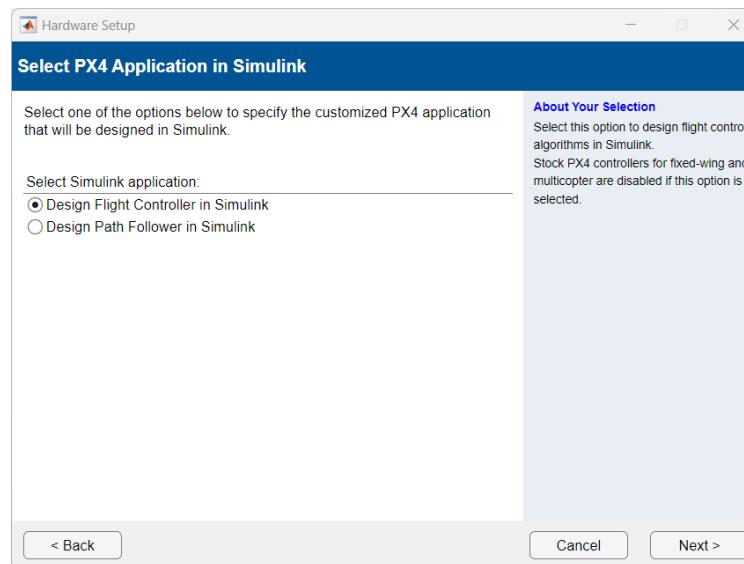


Figura A.9: Design Flight Controller in Simulink

Selezionare la "PX4 Pixhawk 6x" con build target "px4_fmuv6x_multicopter" e clicca "Next" (Fig. A.10).

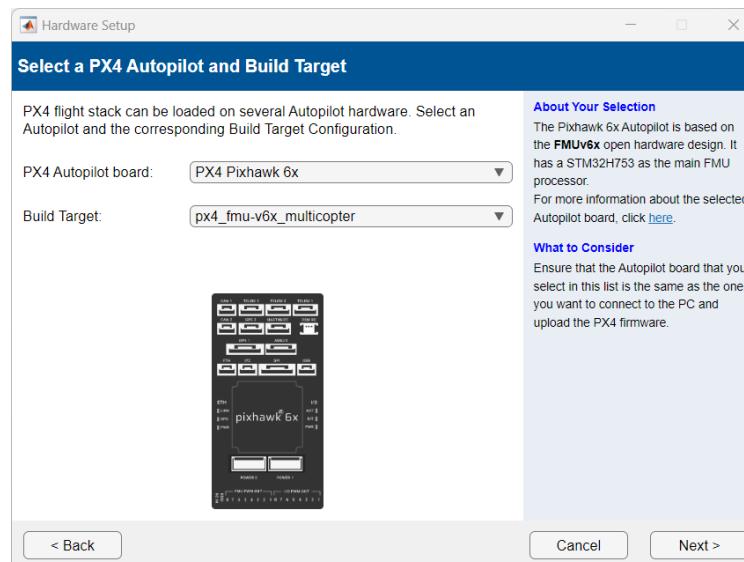


Figura A.10: Select a PX4 Autopilot and Build Target

Selezionare "Use default starup script" e clicca "Next" (Fig. A.11).

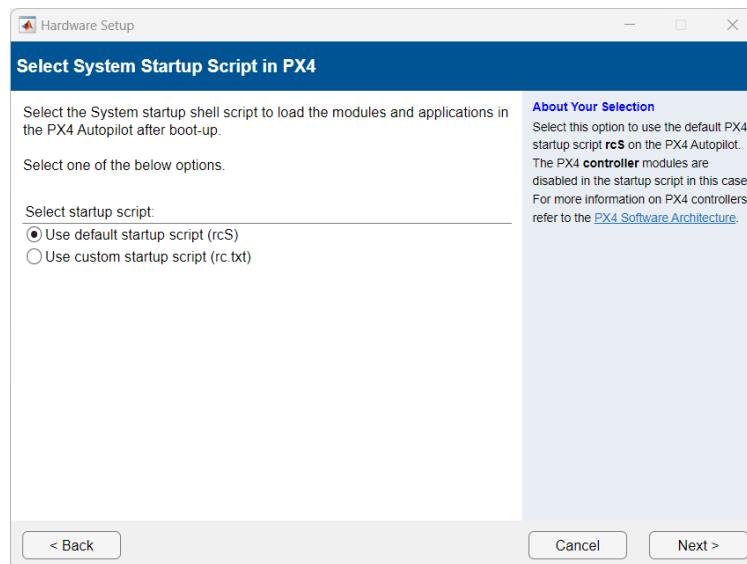


Figura A.11: Select System Startup Script in PX4

Verificare la installazione di QGroundControl e cliccare "Next" (Fig. A.12 e Fig. A.13). In caso QGroundControl non fosse stato installato, lo si può scaricare dal link presente nella schermata e poi ripetere la verifica dell'installazione.

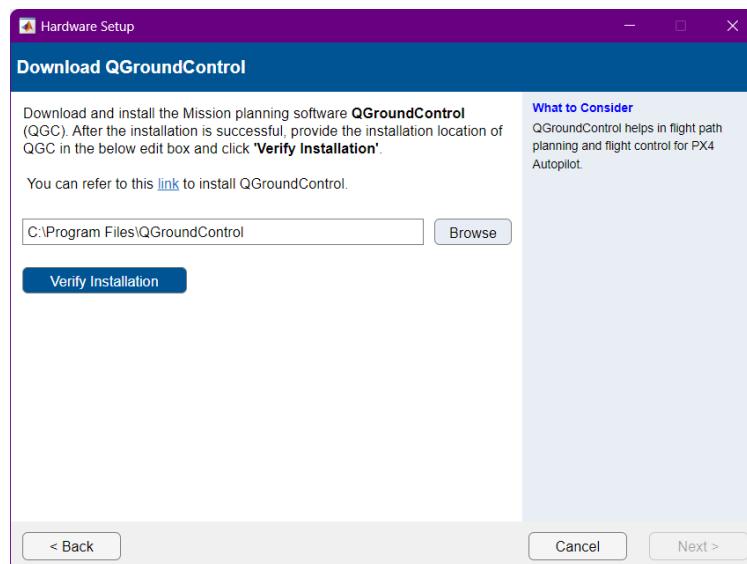


Figura A.12: Download QGroundControl

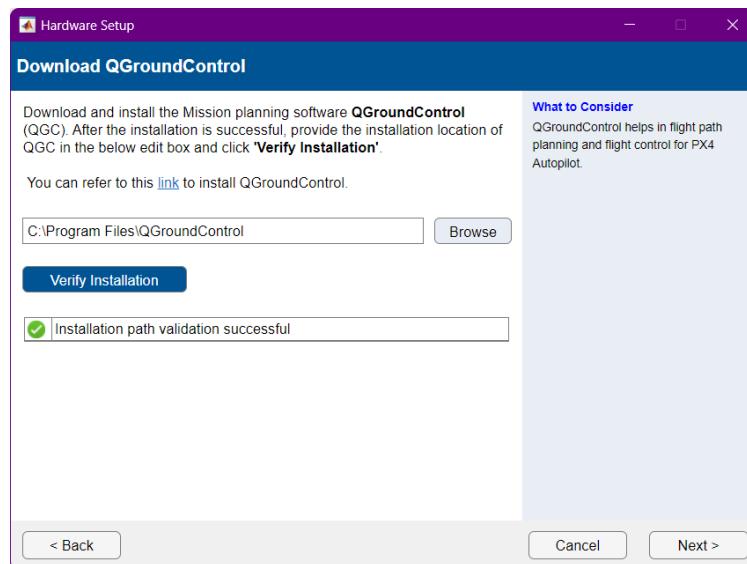


Figura A.13: Download QGroundControl Successful

La selezione dell'Airframe e la configurazione di QGroundControl per abilitare il HIL avverrà in una sezione apposita successivamente. Per ora, cliccare "Next" (Fig. A.14).

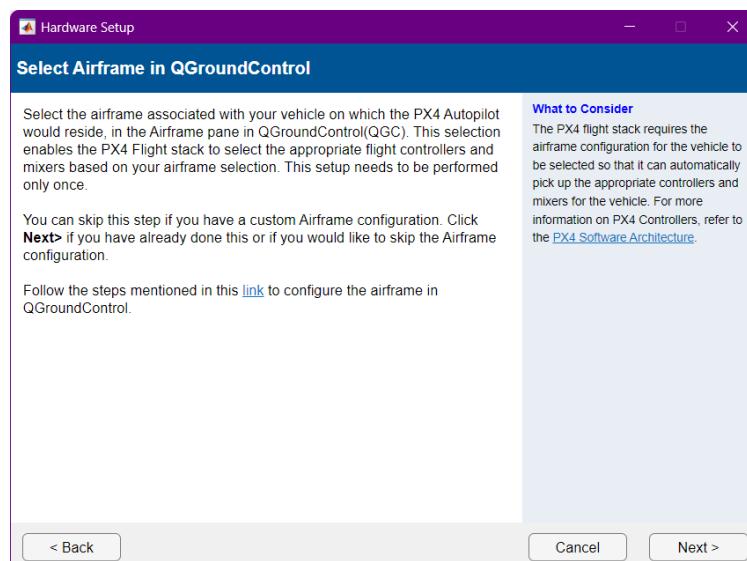
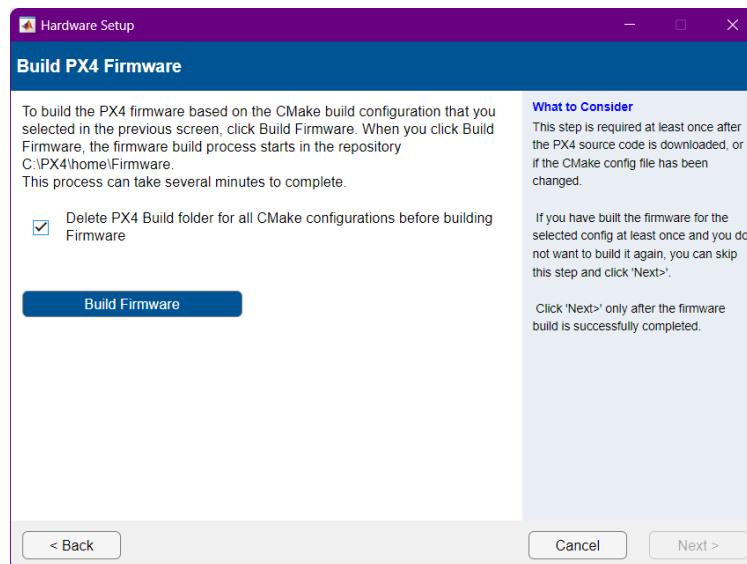


Figura A.14: Select Airframe in QGroundControl

Spuntere la casella "Delete PX4 Build folder for all CMake Configurations before building Firmware" ed eseguire il Build del Firmware (Fig. A.15).

**Figura A.15:** Build Firmware

Aspettare finchè non esce la scritta "Firmware Build Successful" e clicca "Next" (Fig. A.16 e Fig. A.17). Selezionare la COM a cui è collegata la PX4 ed eseguire il caricamento del

What to Consider

This step is required at least once after the PX4 source code is downloaded, or if the CMake config file has been changed.

If you have built the firmware for the selected config at least once and you do not want to build it again, you can skip this step and click 'Next'.

Click 'Next' only after the firmware build is successfully completed.

Build Firmware

What to Consider

This step is required at least once after the PX4 source code is downloaded, or if the CMake config file has been changed.

If you have built the firmware for the selected config at least once and you do not want to build it again, you can skip this step and click 'Next'.

Click 'Next' only after the firmware build is successfully completed.

Figura A.16: Firmware Building**Figura A.17:** Firmware build successful

firmware (Fig. A.18).

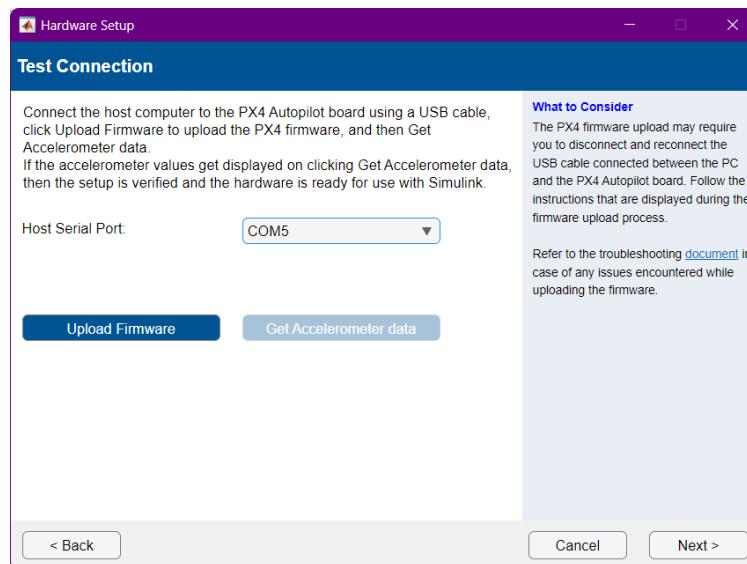


Figura A.18: Test Connection

Uscirà il messaggio mostrato in figura A.19, clicca "OK" e scollegare la PX4 dal pc. Quindi togliere il cavo USB e collegarlo nuovamente per il riavvio della PX4.

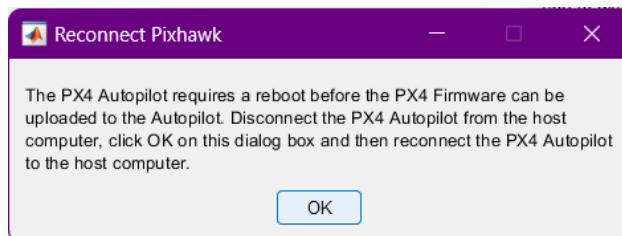


Figura A.19: Reconnect Pixhawk 6x

A questo punto verrà effettuato il caricamento del firmware nella PX4, quindi aspettare il suo completamento (Fig. A.20 e Fig. A.21).

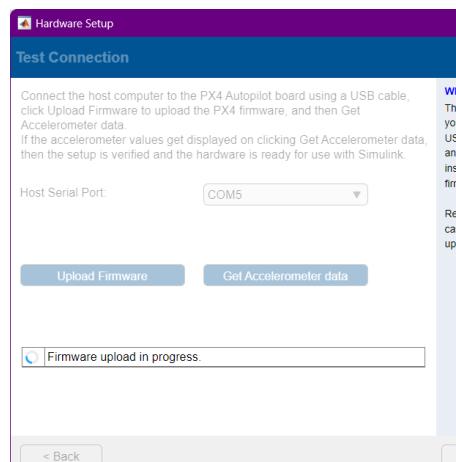


Figura A.20: Firmware uploading

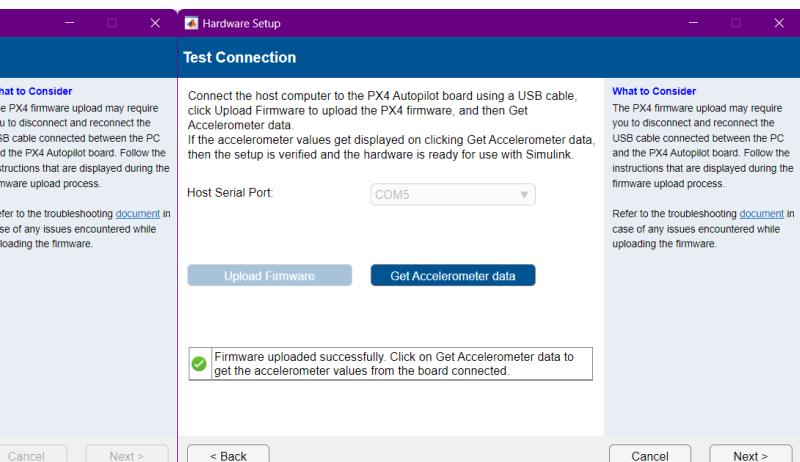


Figura A.21: Firmware upload successful

A questo punto è terminata la configurazione del firmware per la PX4.

A.3 Configurazione QGroundControl

Nella schermata di download di QGroundControl per la configurazione hardware, scarica e installa QGC. Dopo l'installazione, clicca su "Verify Installation" per verificare che l'installazione sia avvenuta correttamente. Successivamente, clicca su "Next" per procedere.

Seguire questi passaggi per configurare e impostare QGroundControl (QGC):

- Collegare la PixHawk 6x a QGC tramite USB.
- Abilitare la modalità HIL.
 - Aprire Setup > sezione Safety.
 - Selezionare Enabled dalla lista HIL Enabled.
- Selezionare il telaio (Airframe) del drone.
 - Aprire Setup > sezione Airframes.
 - Selezionare HIL QuadCopter X per simulare un quadricottero. Fare clic su Apply and Restart nella parte superiore destra della pagina di configurazione del telaio.
- Nella scheda Generale del menu delle impostazioni, deselezionare tutte le opzioni di AutoConnect tranne UDP.

Nota: Questa selezione interrompe la comunicazione tra QGC e l'autopilot PX4 tramite USB. QGC può ora comunicare solo tramite UDP (di solito gestito da un Simulatore che funge da ponte tra QGC e l'autopilot e collega i dati MAVLink tra QGC e l'autopilota tramite UDP).

 - Nella scheda Generale, dal menu Impostazioni, selezionare l'opzione Joystick virtuale.
 - Per configurare il Joystick ed il Failsafe, si deve accedere alla sezione Vehicle Setup, si va nell'area Parameters in fondo alla sidebar e si digitano nella barra di ricerca, uno per volta, i seguenti parametri:
 - 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_RCL_act va disabilitato.
 - 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.

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

- Chiudere QGC.

A.4 PX4 HIL Simulation with UAV Dynamics in Simulink

All'interno della documentazione del "UAV Toolbox Support Package for PX4 Autopilots", vi è la sottosezione "Hardware-in-the-loop Simulation (HIL) with PX4". Al suo interno vi sono vari esempi per effettuare l'Hardware In The Loop con PX4 usando Matlab/Simulink. In questo progetto è stato usato l'esempio "PX4 HIL Simulation with UAV Dynamics in Simulink".

Esistono due metodi per eseguire la simulazione HIL su PX4 da Simulink:

- 1. Build, Deploy and Start:** Il Flight Controller è implementato sulla PixHawk 6x e il modello dinamico dell'UAV viene eseguito su MATLAB su un computer host che comunica con la PixHawk 6x tramite seriale e tramite UDP con QGroundControl. Si possono acquisire solo i segnali provenienti dal Plant Model, dato che il Flight controller è stato caricato sulla PixHawk 6x e non è in comunicazione con l'host PC. Lo schema a blocchi è mostrato in Figura A.22

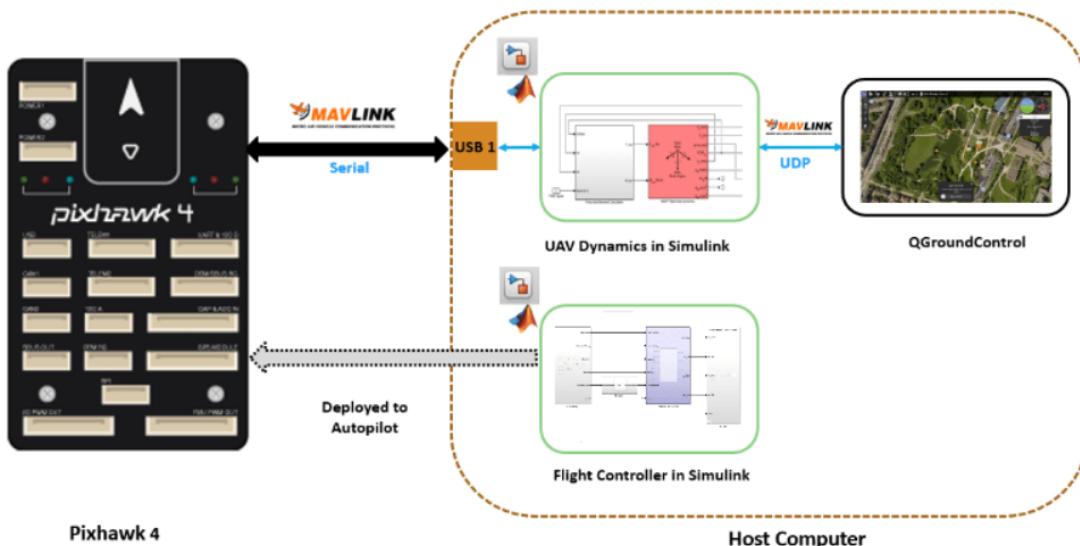


Figura A.22: Controller Deployed on Autopilot over Normal Mode

- 2. Monitor & Tune:** Il Flight Controller è implementato su PixHawk 6x e comunica con MATLAB sul computer host utilizzando la simulazione Monitor & Tune ed il convertitore FTDI. Il modello dinamico dell'UAV viene eseguito su MATLAB sul computer host che comunica con PixHawk 6x tramite seriale e tramite UDP con QGroundControl. Si possono acquisire sia i segnali provenienti dal Flight Controller caricato sulla PixHawk grazie al convertitore FTDI, sia dal Plant Model che è in Run sull'host PC. Lo schema a blocchi è mostrato in Figura A.23

Per entrambe le tipologie di simulazione HIL, è possibile utilizzare un Visualizzatore 3D integrato nel Matlab UAV Toolbox. Lo schema a blocchi è mostrato in Figura A.24. In questo progetto, è stato utilizzato il metodo Monitor & Tune, così da poter effettuare il log dei segnali sia dal Flight Controller che dal Plant Model, i quali serviranno poi per la simulazione e rilevamento guasti alle componenti di attuazione e sensoristica.

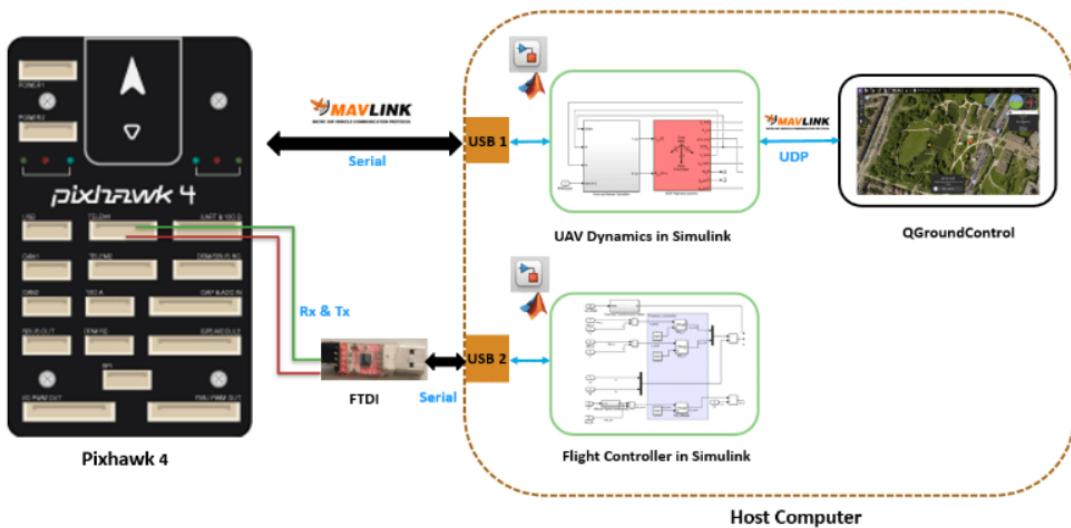


Figura A.23: Controller deployed on Autopilot for Monitor & Tune Simulation

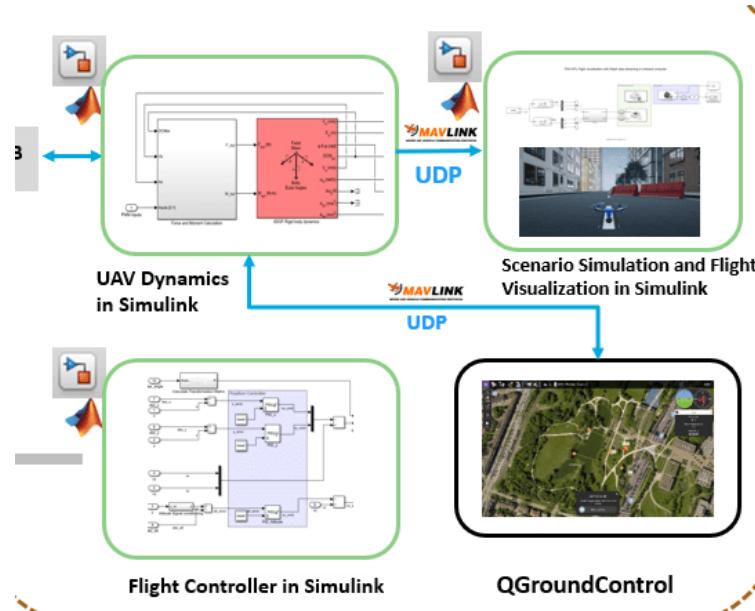


Figura A.24: Scenario Simulation and Flight Visualization

A.5 Avvio Simulazione HIL con Monitor & Tune

A.5.1 Collegamento Hardware

La simulazione Monitor & Tune permette di mettere a punto l'algoritmo in tempo reale sull'autopilota e registrare i segnali in tempo reale in Simulink. In questo caso, è necessaria una porta seriale aggiuntiva per stabilire la comunicazione di Monitor & Tune con l'autopilota. Poiché la porta USB è già occupata per il trasferimento dei dati MAVLink tra il computer host e l'Autopilota in modalità HIL, è necessario utilizzare una delle altre porte seriali disponibili sull'Autopilota. In questo esempio, la porta TELEM3 (/dev/ttyS1) viene utilizzata sul Pixhawk 6x per comunicare con Simulink in esecuzione sul computer host utilizzando la simulazione Monitor & Tune.

Come primo passo si collega la Pixhawk al computer host usando il cavo USB c, successivamente si collega la porta seriale a TELEM3 (/dev/ttyS1) e la porta USB sul computer host

utilizzando un convertitore Serial-to-USB FTDI come mostrato nell'immagine di esempio qui sotto (Fig. A.25).

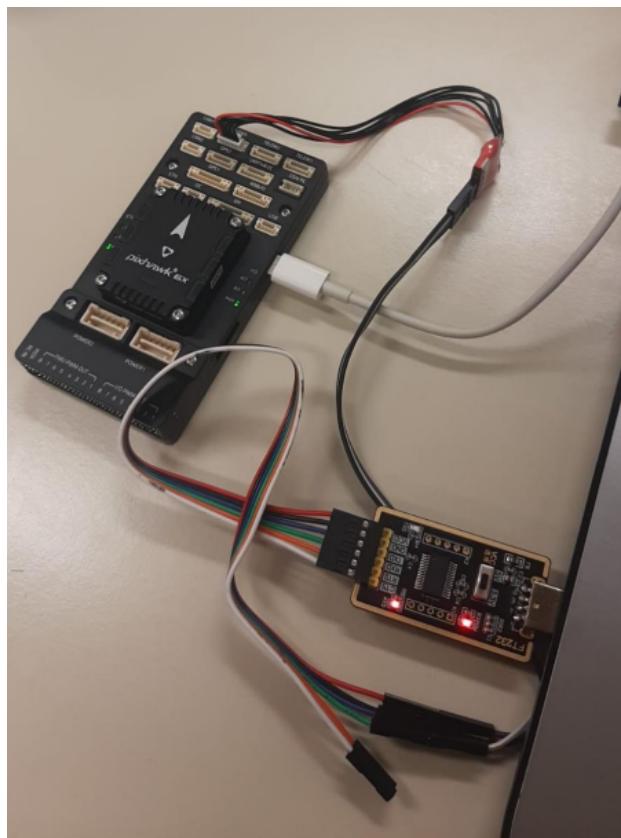


Figura A.25: Collegamento PX4 Pixhawk 6x

A.5.2 Configurazione del Modello del Controller PX4 in Simulink per Monitor & Tune

Per aprire l'esempio "PX4 HIL Simulation with UAV Dynamics in Simulink", digitare il comando `openExample('px4/PX4HILSimulationSimulinkPlantExample')`.

Una volta che il progetto Simulink si è aperto, clicca su "Project Shortcuts" sulla finestra di MATLAB e clicca "Open Autopilot Controller" per aprire il Controller PX4 chiamato "Quadcopter_ControllerWithNavigation".

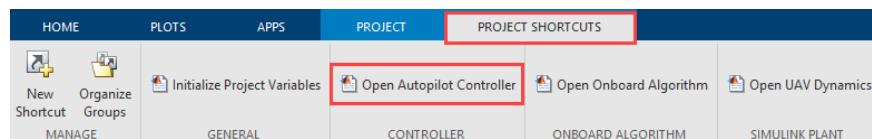


Figura A.26: Open Autopilot Controller

Ora si deve configurare il Flight Controller PX4 per poterlo caricare all'interno della PixHawk6x e per abilitare la simulazione Monitor & Tune:

1. Vai su Modeling > Model Settings per aprire la finestra di dialogo Configuration Parameters.
2. Apri il pannello Hardware Implementation e seleziona la scheda PX4 Pixhawk 6x.

3. Espandi le risorse hardware di destinazione per quella scheda.
4. Fai clic su HIL e quindi seleziona Enable HIL Mode. Abilitando la modalità HIL, si potrà scegliere tra il simulatore jMAVSIM e Simulink. Seleziona "Simulink"
5. Fai clic su External mode e seleziona /dev/ttyS1 come porta seriale della scheda hardware. Nel parametro Host Serial port, inserisci il numero di porta seriale host sul computer host per la comunicazione in modalità esterna.
6. Fai clic su MAVLink e assicurati che sia selezionata l'opzione Enable MAVLink on /dev/ttyACM0.
7. Fai clic su /dev/ttyS1 e imposta il Baud rate su 921600. Non modificare altri parametri.
8. Fai clic su Apply e poi su OK.

Dopo aver configurato correttamente il Flight Controller PX4, nella "Project Shortcuts", clicca "Open UAV Dynamics" per aprire il modello dinamico dell'UAV in Simulink chiamato "UAV_Dynamics_Autopilot_Communication".

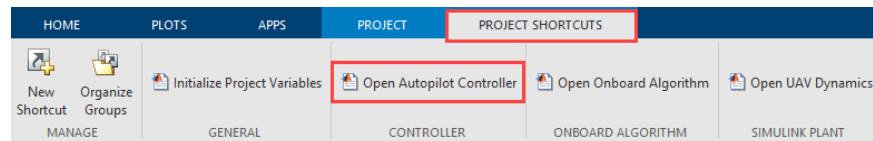


Figura A.27: Open UAV Dynamics

A.5.3 Avviare Monitor & Tune per il Flight Controller PX4, Run del UAV Dynamics model, Upload Mission in QGroundControl e volo dell'UAV

1. Nella scheda Hardware, nella sezione Mode, seleziona Run on board e poi fai clic su Monitor & Tune.

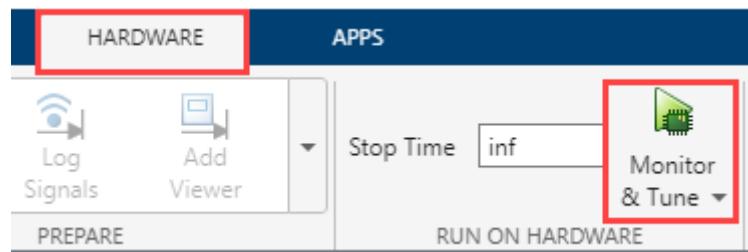


Figura A.28: Monitor & Tune

2. Questo avvia QGC. (In caso di errore, controllare l'appendice A)
3. Nella barra Simulink del Plant model (UAV_Dynamics_Autopilot_Communication), su Simulation, premere Run per simulare il modello.

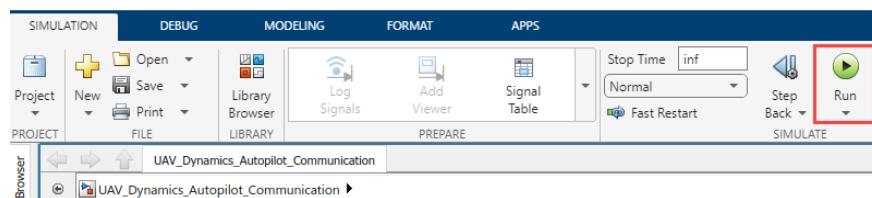


Figura A.29: Run UAV_Dynamics_Autopilot_Communication

4. In QGC, vai alla vista Plan.
5. Crea una missione o carica la missione pre-pianificata, Mission.plan, disponibile con questo in Figura A.30.

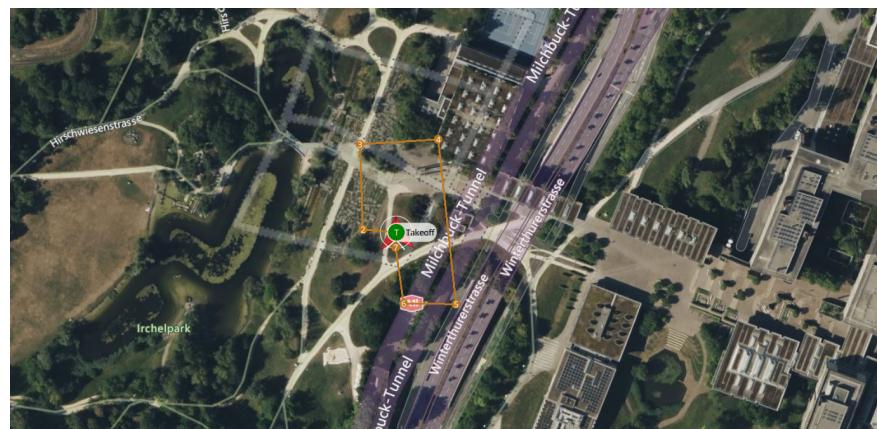


Figura A.30: Pianificazione percorso

6. Se si vuole caricare una missione: Fai clic su Open Example nella parte superiore di questa pagina per salvare il file di piano sul tuo computer. Dopo aver salvato il file .plan, avvia QGC, e fai clic su File > Open per caricare il piano su QGC. Dopo aver caricato il piano, la missione è visibile in QGC.
7. Fai clic sul pulsante Upload nell'interfaccia QGC per caricare la missione da QGroundControl.
Nota: Se la scheda hardware viene riavviata dopo che la missione è stata caricata, la missione potrebbe non funzionare se QGC, UAV Dynamics e 3D Visualization with Unreal Engine del modello Simulink sono aperti. Riavvia QGC, UAV Dynamics e 3D Visualization with Unreal Engine, successivamente carica nuovamente la missione per risolvere il problema.
8. Vai alla vista Fly per visualizzare la missione caricata.
9. Se si desidera visualizzare il drone in 3D, seguire i passaggi seguenti:
 - Aprire una nuova sessione di MATLAB.
 - Aprire il progetto esistente.
 - Aprire il file Simulink "3D Visualization with Unreal Engine" (Fig. A.31).

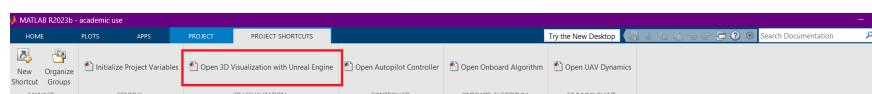


Figura A.31: Nuova sessione MATLAB per la visualizzazione 3D

- Eseguire il comando "Run" in Simulink (Fig. A.32).

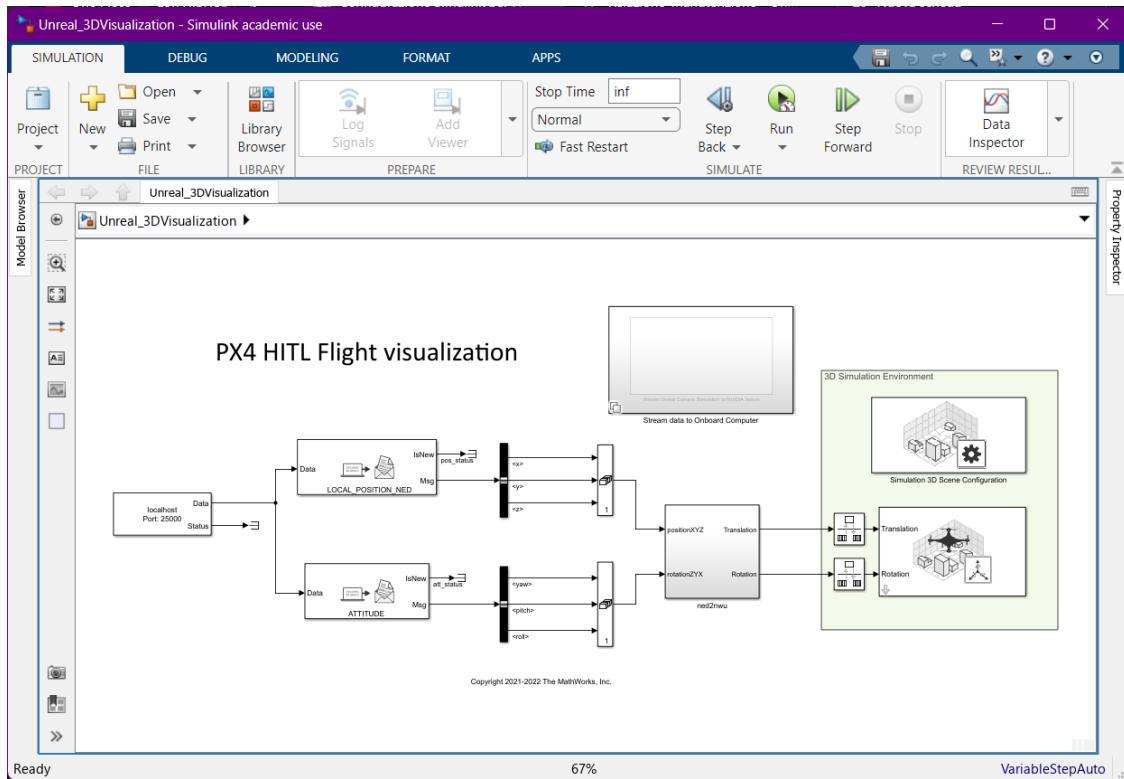


Figura A.32: 3D Visualization with Unreal Engine

- A questo punto vi si aprirà un nuova finestra con la visualizzazione 3D del drone (Fig. A.33).

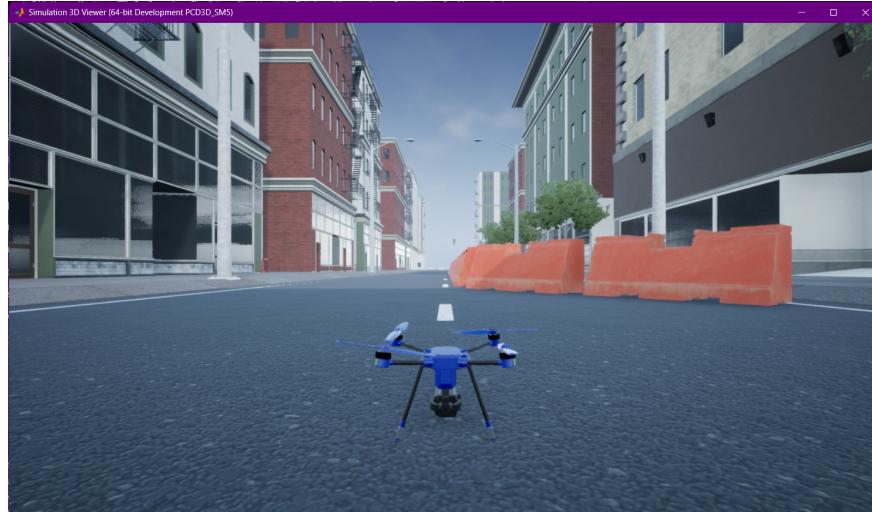


Figura A.33: Drone 3D

- Avvia la Missione in QGC. Dopo l'avvio della missione, il drone decolla e segue l'insieme di waypoints da QGC.

Bibliografia

- [1] Halim Alwi, Christopher Edwards, and Chee Pin Tan. *Fault detection and fault-tolerant control using sliding modes*. Springer, 2011.
- [2] PX4 Autopilot. Px4 architectural overview, 2024. Accessed: 2025-01-09.
- [3] PX4 Autopilot. Px4 system architecture, 2024. Accessed: 2025-01-09.
- [4] Jovan D. Bošković and Raman K. Mehra. *Failure Detection, Identification and Re-configuration in Flight Control*, pages 129–167. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [5] Lorenzo Cacciatori, Carlo Brignoli, Benedetto Mele, Federica Gattere, Celeste Monti, and Maurizio Quadrio. Drag reduction by riblets on a commercial uav. *ResearchGate, Applied Sciences*, 12(10), 2022.
- [6] Xin Dong, Ziyu Wang, Fangyuan Liu, Song Li, Fan Fei, Daochun Li, and Zhan Tu. Visual-inertial cross fusion: A fast and accurate state estimation framework for micro flapping wing rotors. *MDPI, Drones*, 6(4), 2022.
- [7] Daoliang Li, Ying Wang, Jinxing Wang, Cong Wang, and Yanqing Duan. Recent advances in sensor fault diagnosis: A review. *Elsevier, Sensors and Actuators A: Physical*, 309:111990, 2020.
- [8] Hamid Maqsood, Muhammad Taimoor, Zahid Ullah, Nihad Ali, and Muhammad Sohail. Novel sensor fault detection and isolation for an unmanned aerial vehicle. In *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, pages 486–493. IEEE, 2021.
- [9] MathWorks. Imu, 2024. Accessed: 2025-01-09.
- [10] MathWorks. Integration with general px4 architecture, 2024. Accessed: 2025-01-09.
- [11] MathWorks. Mavlink connectivity for qgc, on-board computer and simulink plant introduction, 2024. Accessed: 2025-01-09.
- [12] Radosław Puchalski and Wojciech Giernacki. Uav fault detection methods, state-of-the-art. *MDPI, Drones*, 6(11):330, 2022.
- [13] Hanlin Sheng, Chen Zhang, and Yulong Xiang. Mathematical modeling and stability analysis of tiltrotor aircraft. *MDPI, Drones*, 6(4), 2022.

Elenco delle figure

1.1	Fixed Wing UAV [5]	8
1.2	Tipi di disposizioni delle eliche multirotore. Nella riga superiore ci sono, in ordine da sinistra: bicottero, tricottero, quad +, quad X, quad H, quad V, quad Y. Nella riga inferiore ci sono, in ordine da sinistra: hexa +, hexa X, hexa Y6, hexa IY, octo +, octo X, octo X8.	9
1.3	Tilt-rotor UAV [13]	9
1.4	Flapping Wing UAV [6]	9
2.1	Architettura di alto livello del controllore di volo in PX4 [3]	12
2.2	Architettura di alto livello del controllore di volo in PX4 [2]	13
2.3	Architettura di alto livello del controllore di volo in PX4 [2]	14
2.4	Blocchi Simulink Supportati che si interfacciano con i Moduli PX4 [10]	15
2.5	Moduli supportati che possono essere sostituiti con algoritmi definiti dall'utente [10]	16
2.6	Home del software QGroundControl	17
2.7	Sezioni principali del software QGroundControl	17
2.8	PixHawk 6x	19
2.9	FT232 FTDI	20
2.10	Workflow per l'HIL in PX4 [11]	20
3.1	Flight Controller in Simulink	22
3.2	Contenuto del blocco Estimator Output	23
3.3	Contenuto del blocco Navigation	23
3.4	Contenuto del blocco Position and Rate Controller	24
3.5	Contenuto del blocco Mixer and Send to Actuator	24
3.6	UAV Dynamics Model	25
3.7	Contenuto del UAV Dynamics Model	25
3.8	Modello matematico del UAV	25
3.9	IMU Simulation	26
3.10	Baro Simulation	26
3.11	GPS Simulation	26
3.12	Accelerometer Algorithm [9]	27
3.13	Gyroscope Algorithm [9]	27
3.14	Magnetometer Algorithm [9]	28
3.15	Log dei Segnali dal Flight Controller	32
3.16	Percorso di prova durante e dopo la simulazione.	33
3.17	Esempio di plot 3D del percorso seguito dal drone.	34
4.1	Tipologie di guasti [12]	36
4.2	Actuator input/output	36
4.3	Lock in Place	37

4.4	Float fixed wing	38
4.5	Float rotary wing	38
4.6	Hardover	38
4.7	Loss of Effectiveness	39
4.8	Sensor input/output	39
4.9	Bias	40
4.10	Drift	40
4.11	Loss of Accuracy	41
4.12	Freezing	41
4.13	Calibration Error	42
5.1	IMU Fault Modules	45
5.2	Accelerometer Fault Module	46
5.3	Maschere Fault Injection per l'accelerometro per i guasti Freezing e Loss of Accuracy	48
5.4	Maschere Fault Injection per l'accelerometro per i guasti Drift e Bias	48
5.5	Modulo Fault Injection GPS	49
5.6	Interno del modulo Fault Injection GPS	50
5.7	Maschere Fault Injection per l'accelerometro per i guasti Packet Loss e Loss of Accuracy	51
5.8	Modulo Fault Injection Attuatori	52
5.9	Identificazione dei motori nel drone	52
5.10	Struttura interna del modulo Fault Injection Attuatori	53
5.11	Maschera per l'inserimento dei parametri di guasto degli attuatori	55
5.12	Modulo Fault Injection	55
5.13	Interfaccia Principale del Modulo Fault Injection	56
5.14	Freezing sull'asse a_x	57
5.15	Fault <i>bias</i> sull'asse y dell'accelerometro	57
5.16	Drift Fault sull'asse x dell'accelerometro	58
5.17	Loss of Accuracy Fault sull'asse x del magnetometro	58
5.18	Packet Loss Latitudine	59
5.19	Hardover Fault Attuatore	59
5.20	Lock in Place Fault Attuatore	60
5.21	Float Fault Attuatore UAV Fixed Wing	60
5.22	Loss of Effectiveness Fault Attuatore	61
6.1	Volo 1	63
6.2	Volo 2	63
6.3	Volo 3	63
6.4	uORB	64
6.5	Visualizzazione di a_x , ω_y , m_z del dataset finale.	66
6.6	Visualizzazione della Latitudine e Longitudine normalizzate del dataset finale.	67
6.7	Definizione della Frame Policy.	68
6.8	Matrice di confusione del modello Medium Tree per il Calcolatore	69
6.9	Struttura Simulink Buffer Circolare	70
6.10	Struttura del modulo diagnostico in un calcolatore	72
6.11	Confronto tra rilevamento guasti senza filtraggio (sinistra) e con filtraggio di 5 secondi (destra)	72
6.12	Matrice di confusione del modello Medium Tree per il dispositivo Microcontrollore	74

6.13 Confronto tra rilevamento guasti senza filtraggio (sinistra) e con filtraggio di 5 secondi (destra)	74
A.1 PX4 Toolchain Setup	79
A.2 Selezione della cartella di installazione	79
A.3 Clona repository PX4 e Avvia Simulazione	80
A.4 Shell bash che avvia la clonazione del firmware	80
A.5 Python 3.8.2	81
A.6 Validare Python 3.8.2	81
A.7 Verifica Percorso	82
A.8 Cartella di download per il codice sorgente della PX4	82
A.9 Design Flight Controller in Simulink	83
A.10 Select a PX4 Autopilot and Build Target	83
A.11 Select System Startup Script in PX4	84
A.12 Download QGroundControl	84
A.13 Download QGroundControl Successful	85
A.14 Select Airframe in QGroundControl	85
A.15 Build Firmware	86
A.16 Firmware Building	86
A.17 Firmware build successful	86
A.18 Test Connection	87
A.19 Reconnect Pixhawk 6x	87
A.20 Firmware uploading	87
A.21 Firmware upload successful	87
A.22 Controller Deployed on Autopilot over Normal Mode	89
A.23 Controller deployed on Autopilot for Monitor & Tune Simulation	90
A.24 Scenario Simulation and Flight Visualization	90
A.25 Collegamento PX4 Pixhawk 6x	91
A.26 Open Autopilot Controller	91
A.27 Open UAV Dynamics	92
A.28 Monitor & Tune	92
A.29 Run UAV_Dynamics_Autopilot_Communication	93
A.30 Pianificazione percorso	93
A.31 Nuova sessione MATLAB per la visualizzazione 3D	93
A.32 3D Visualization with Unreal Engine	94
A.33 Drone 3D	94

Ringraziamenti

Al termine di questa tesi, desidero esprimere la mia gratitudine a tutte le persone che hanno contribuito a rendere possibile questo risultato.

In primo luogo, ringrazio il Professore Alessandro Freddi. Grazie alla sua guida, sono riuscito a superare tutte le fasi critiche della tesi e a raggiungere i risultati ottenuti. Il suo approccio metodico e la sua disponibilità sono stati per me un riferimento preziosa in questo percorso. Un sentito ringraziamento va alla mia famiglia, mio padre Andrea, mia madre Francesca e le mie sorelle Giorgia ed Eleonora, per il loro sostegno in questo periodo così intenso e impegnativo. Il loro affetto e la loro vicinanza sono stati fondamentali.

Ringrazio i miei nonni, quelli che oggi sono con me e quelli che continuano ad esserlo comunque. A loro devo i ricordi d'infanzia, i loro racconti e soprattutto il loro affetto.

Ringrazio di cuore Debora, che è sempre stata presente in questi due anni di magistrale. Mi ha sempre sostenuto, spronandomi nei momenti difficili e gioendo con me per ogni piccolo traguardo.

Un grazie agli amici Federico, Riccardo, Cristian ed Emanuele, che hanno saputo regalarmi momenti di leggerezza in questi anni universitari.

Infine, un ringraziamento a tutti gli amici dell'Università, senza i quali non avrei raggiunto questo traguardo. Le nostre lunghe sessioni di studio e le pause piene di risate hanno reso questo percorso meno faticoso e molto più piacevole. Grazie alla loro compagnia, sono stati 5 anni di Università divertenti.