UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'automazione

Progetto E2: Software-In-The-Loop PX4 – MATLAB UAV Toolbox

Candidato:
Cichella Lorenzo
Antonini Adelio
Garzarella Fiore

Relatore:
Prof. Freddi Alessandro

Anno Accademico 2022-2023

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

I droni sono sistemi complessi che sono soggetti a una serie di potenziali guasti. I guasti possono essere causati da una varietà di fattori, tra cui errori di programmazione, malfunzionamenti hardware, interferenze o azioni umane. I guasti nei droni possono avere una serie di conseguenze negative, come incidenti o il danneggiamento di persone e cose. Lo studio e il rilevamento dei guasti è un aspetto fondamentale per la sicurezza di questi dispositivi.
Il seguente progetto ha come obiettivo la realizzazione di una connessione Software-In-The-Loop tra il software PX4 e il MATLAB UAV Toolbox, al fine di sfruttare le librerie messe a disposizione, effettuare il log dei segnali e simulare l'iniezione di guasti alle componenti di attuazione e sensoristica di un drone quadrirotore.
I simulatori software offrono una serie di vantaggi per il rilevamento guasti nei droni. Tra questi vantaggi, si possono evidenziare:

- Flessibilità: i simulatori software consentono di simulare una vasta gamma di guasti in un ambiente controllato. Questo rende i simulatori software un modo ideale per testare i sistemi di rilevamento guasti e per migliorare la sicurezza dei droni.
- Economicità: i simulatori software sono generalmente meno costosi rispetto alla simulazione di guasti sull'hardware. Questo rende i simulatori software una soluzione ideale per le aziende che vogliono migliorare la sicurezza dei propri droni senza dover spendere ingenti somme di denaro.
- Rapidità: i simulatori software consentono di simulare guasti in tempi molto rapidi.
- Sicurezza: la simulazione di guasti in dispositivi hardware necessita di più accortezza e presenta rischi maggiori. Le simulazioni software consentono di testare i droni in un ambiente sicuro senza mettere a rischio persone o cose.

Capitolo 2

Materiali e metodi

In questo capitolo verranno discussi gli strumenti software utilizzati per la realizzazione del progetto e le attività iniziali di configurazione necessarie per il corretto funzionamento delle simulazioni.

2.1 PX4-Autopilot

PX4 Autopilot è un sistema completo di controllo di volo open source progettato per droni e altri veicoli autonomi. È uno dei progetti più popolari e ampiamente utilizzati nel campo dei droni.
PX4 Autopilot offre una vasta gamma di funzionalità e capacità avanzate per il controllo di volo, la navigazione e il monitoraggio dei veicoli aerei senza pilota. Queste funzionalità includono il controllo di stabilità, il posizionamento GPS, l'automazione delle missioni, il supporto per i sensori di bordo, la gestione della telemetria e molto altro.
Il sistema PX4 Autopilot è basato su un'architettura modulare che consente agli sviluppatori di personalizzare e adattare il software alle specifiche esigenze del drone. Questa flessibilità rende PX4 adatto a una vasta gamma di applicazioni, tra cui la fotografia aerea, la mappatura, l'agricoltura di precisione, l'ispezione industriale e molto altro.

Figura 2.1: Logo PX4

PX4 Autopilot supporta diversi tipi di droni, inclusi multicotteri, aerei a decollo e atterraggio verticali (VTOL), aerei a singola ala e elicotteri. È compatibile con una varietà di piattaforme hardware e offre una vasta scelta di opzioni di controllo radio e telemetria.

Essendo un progetto open source, PX4 Autopilot offre numerosi vantaggi agli sviluppatori. L'accesso al codice sorgente consente di personalizzare il sistema, condividere le modifiche con la comunità e contribuire al miglioramento continuo del software. Inoltre, la comunità di PX4 offre supporto attivo, documentazione completa e una vasta gamma di risorse per aiutare gli sviluppatori a ottenere il massimo dal sistema.

2.2 MATLAB UAV Toolbox

UAV Toolbox fornisce strumenti e applicazioni di riferimento per la progettazione, la simulazione, il test e la distribuzione di applicazioni per droni e aeromobili a pilotaggio remoto (APR, o UAV in inglese). È possibile progettare algoritmi per il volo autonomo, missioni APR e controller di volo. L'app Flight Log Analyzer consente di analizzare in modo interattivo percorsi di volo 3D, dati di telemetria e le letture dei sensori dai registri di volo nei formati più diffusi.

Figura 2.2: UAV Toolbox features

Per le simulazioni desktop (SITL) e i test Hardware-In-the-Loop (HIL) degli algoritmi di volo autonomo e i controller di volo, è possibile generare e simulare scenari con APR. È possibile simulare output di sensori GPS, di fotocamere, LIDAR e IMU in un ambiente 3D fotorealistico oppure in un ambiente di simulazione 2.5D.
Il toolbox supporta la generazione di codice C/C++ per la prototipazione rapida, i test HIL e la distribuzione standalone su hardware come il pilota automatico Pixhawk® (con Embedded Coder).

2.3 PX4 support package for UAV Toolbox

Il "PX4 Support Package for UAV Toolbox" è un pacchetto di supporto software sviluppato da MathWorks, che consente l'integrazione tra l'ambiente di sviluppo MATLAB/Simulink e il sistema di controllo di volo PX4 Autopilot.
Questo pacchetto offre una serie di blocchi di Simulink predefiniti e funzioni di generazione del codice per facilitare lo sviluppo, la simulazione e l'implementazione di algoritmi di controllo di volo per droni utilizzando PX4. Consente agli ingegneri e agli sviluppatori di utilizzare l'ambiente di sviluppo Simulink per progettare e sperimentare algoritmi di controllo, testarli in simulazione e generare automaticamente il codice per essere eseguito su PX4 Autopilot. Il PX4 Support Package for UAV Toolbox semplifica il processo di sviluppo di algoritmi di controllo complessi e l'integrazione con il sistema di controllo di volo PX4. Offre un'interfaccia intuitiva per configurare i parametri di volo, acquisire dati dai sensori a bordo e interagire con

i componenti del drone. Inoltre, il pacchetto supporta funzionalità di simulazione software-in-the-loop (SITL) per testare e validare gli algoritmi di controllo in un ambiente simulato prima di essere implementati su un drone reale.
L'architettura software di alto livello di PX4 include moduli per l'archiviazione, la connettività esterna, i driver, il bus di messaggi di pubblicazione-sottoscrizione uORB e i componenti dello stack di volo.

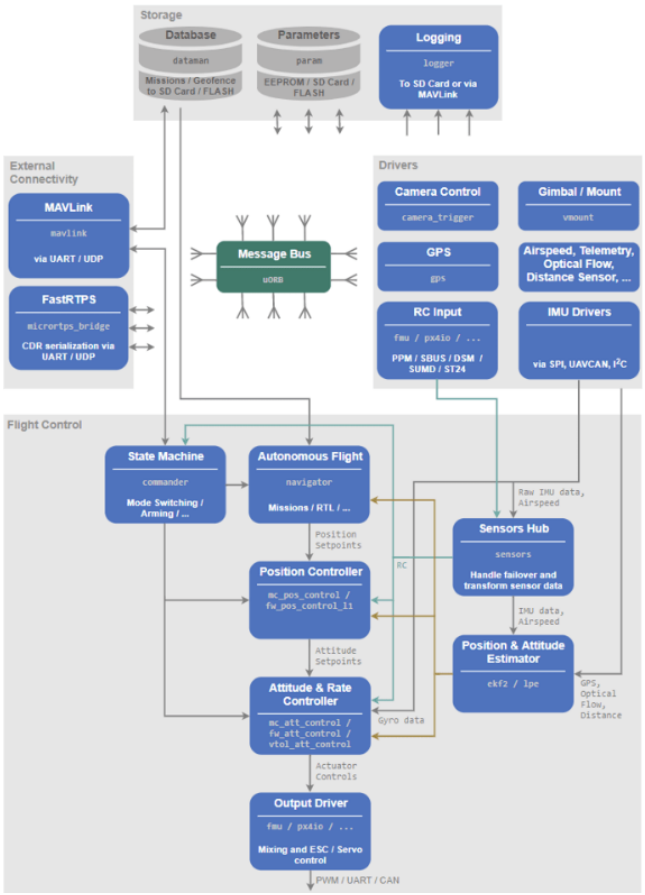

Figura 2.3: Architettura PX4

Infine, i modelli presenti nel toolbox possono essere modificati o sostituiti con algoritmi definiti dall'utente.

2.4 Installazione e configurazione di PX4-Autopilot

Si precisa che il seguente elaborato è stato sviluppato sul sistema operativo Windows 11. Per configurare il Cygwin toolchain e scaricare il codice sorgente di PX4 che verrà utilizzato nel PX4 support package, è necessario seguire i seguenti passaggi:

1. Scaricare la versione 0.8 di PX4 Cygwin Toolchain MSI Installer, che è compatibile con PX4 Firmware v1.12.3, disponibile a questo link [1].

2. Al termine dell'installazione, spuntare la casella "clone PX4 repository and start simulation". In questo modo verrà copiato il codice sorgente del repository contenente il firmware di PX4 direttamente nella cartella specificata per l'installazione. Al termine della compilazione verrà avviata automaticamente la simulazione.

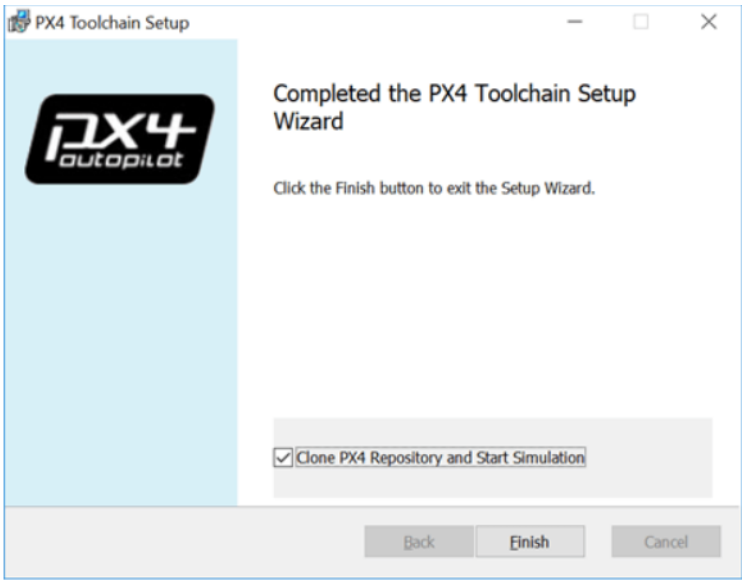

Figura 2.4: Setup PX4

Si fa notare che la versione 0.8 di PX4 è l'unica supportata dal PX4 Support package di MATLAB.

2.5 Installazione e configurazione del MATLAB UAV Toolbox e del PX4 support package for UAV Toolbox

Per l'installazione del MATLAB UAV Toolbox è necessario aprire MATLAB, e nella sezione "Home" selezionare Add-Ons -> Get Add-Ons. Digitare nella barra di ricerca "UAV Toolbox" e cliccare sul primo risultato. Infine selezionare su "Install", comincierà l'installazione del toolbox al termine della quale MATLAB verrà riavviato. Per quanto riguarda il PX4 support package for UAV Toolbox invece, è necessario seguire i seguenti step:

1. Installare il PX4 support package allo stesso modo dell'UAV Toolbox, digitando il nome completo nella barra di ricerca degli Add-Ons.

2. Una volta effettuata l'installazione e riavviato MATLAB, recarsi in Add-Ons -> Magage Add-Ons, scorrere fino a trovare il PX4 support package e cliccare sulla rotellina nella parte destra dello schermo, come mostrato in figura 2.5

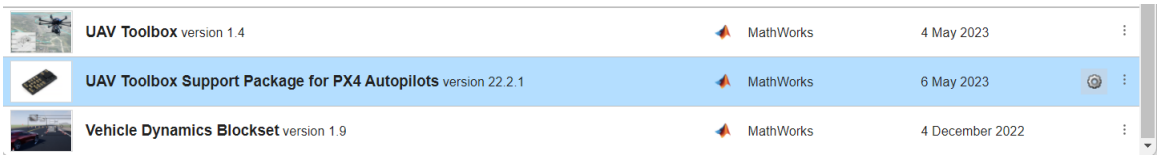

Figura 2.5: Configurazione PX4 support package

3. All'aprirsi dell'Hardware setup, seguire la procedura guidata specificando le directory di installazione di PX4 e del firmware per effettuare la validazione.

4. Proseguire nell'installazione lasciando tutto come di default fino ad arrivare alla pagina "Build PX4 Firmware", cliccare su "build firmware". Questa operazione potrebbe richiedere diversi minuti, a seconda delle prestazioni della macchina.

5. Una volta completata la compilazione del firmware, cliccare su "Lauch PX4 Host Target and jMAVSim" per avviare una prima simulazione SITL tra MATLAB e il simulatore. Dopo pochi secondi, questa verrà lanciata come mostrato in figura 2.6

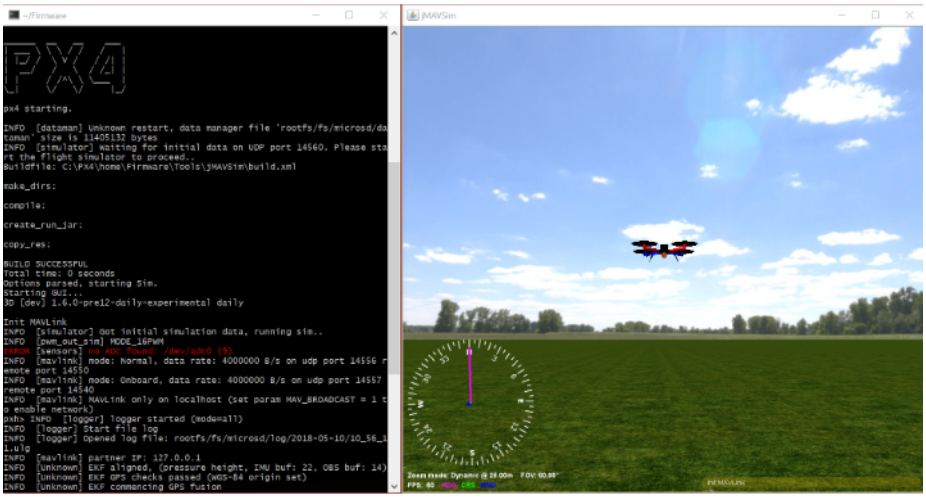

Figura 2.6: Test simulazione

Capitolo 3

Sviluppo del progetto

Lo sviluppo è stato incentrato in 4 fasi principali:

1. Studio della documentazione Matlab/Simulink per comprendere le features offerte dall'UAV Toolbox e il funzionamento dei blocchi Simulink da usare nelle simulazioni software e della documentazione PX4, approfondendo il protocollo MAVLink, necessario per l'interazione tra la console PX4 e Simulink durante la simulazione.

2. Analisi dei log di volo acquisiti durante le simulazioni, partendo dal modello Matlab "Position tracking for X-configuration quadcopter" e apportando delle modifiche per realizzare una traiettoria replicabile del drone, da testare in presenza di guasti e non.

3. Iniezione di guasti in tempo reale alle componenti sensoriali del drone, e successiva analisi dei dati per confrontare il caso con guasti e senza guasti.

4. Simulazione di un guasto moltiplicativo alle componenti di attuazione del drone e analisi dei dati acquisiti dai sensori.

3.1 Modello MATLAB di partenza: Position Tracking for X-configuration quadcopter

Questo modello è una completa applicazione del PX4 support package for UAV Toolbox per progettare un controllore di posizione per un drone quadricottero con una configurazione a X.
Il modello implementa un controllore PID per regolare la posizione e l'assetto del drone. Ad ogni passo temporale l'algoritmo controlla la velocità di rotazione dei diversi rotori per tracciare l'assetto desiderato, in base all'errore di posizione. In questo esempio si considera l'influenza delle raffiche di vento sul quadricottero durante il volo. Quest'ultimo deve cercare di mantenere la posizione e l'altitudine desiderate. In base all'errore di posizione, vengono generati i comandi di beccheggio e rollio e viene modificata l'uscita ai motori dell'attuatore. Gli input slider presenti nella parte sinistra del modello possono essere utilizzati per fornire le coordinate desiderate al drone.

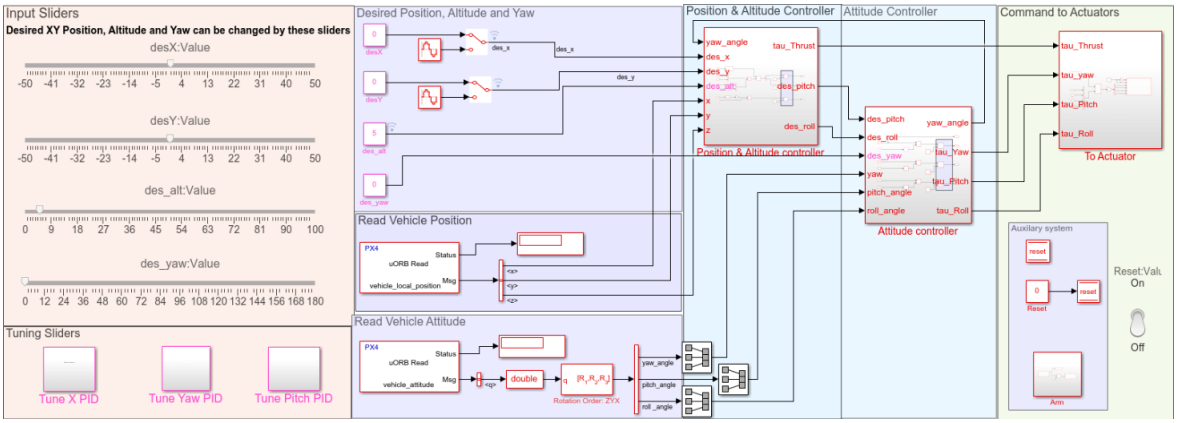

Figura 3.1: Modello Simulink

La variabile des_alt rappresenta un'altitudine alla quale il veicolo staziona. Il valore dell'altitudine può essere impostato utilizzando il rispettivo slider o modificando direttamente il valore della costante des_alt. La posizione desiderata per X e Y invece può essere impostata assegnando i valori desiderati alle costanti des_x e des_y o utilizzando i rispettivi slider.
Nell'area "Desired Position, Altitude and Yaw" è inoltre possibile fornire input dinamici (anziché solo input statici) per testare le capacità di tracciamento del controller di posizione. In questo esempio, l'onda sinusoidale viene utilizzata come ingresso dinamico e può essere selezionata utilizzando i rispettivi interruttori manuali.

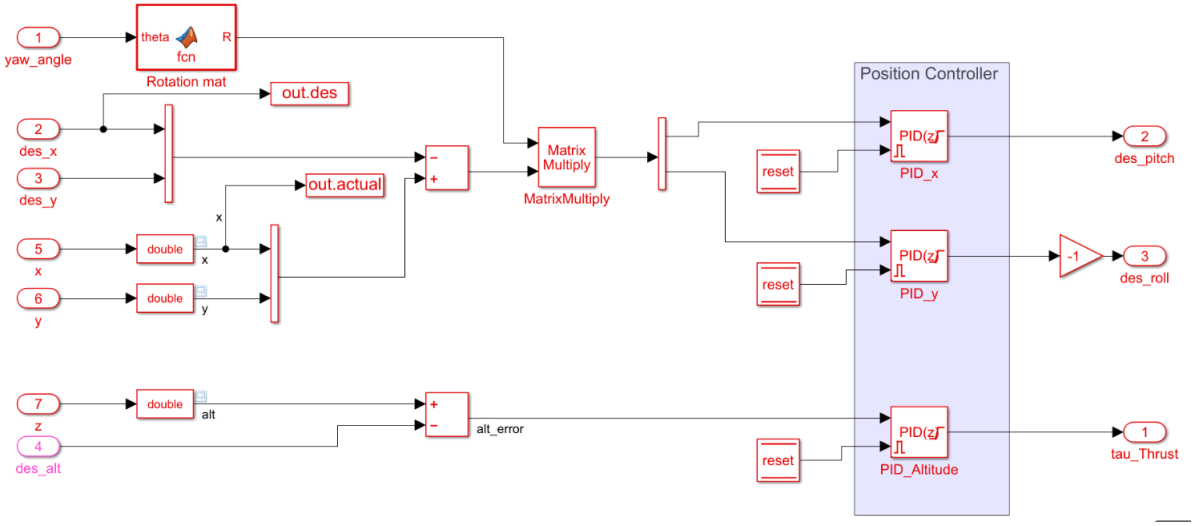

Figura 3.2: Position & altitude controller

Nel sottosistema Position & Altitude Control , tre blocchi PID separati utilizzano la differenza tra il valore desiderato e il valore corrente (che viene letto utilizzando il blocco di lettura uORB) per generare gli angoli di rotazione, beccheggio e rollio. Un'uscita di ciascun blocco PID è limitata tra i valori massimo e minimo predefiniti

per limitare gli angoli di rollio e beccheggio. Il limite è impostato a 50 gradi per questo esempio. Il limite massimo dell'angolo di beccheggio e rollio è generalmente dedotto dalle caratteristiche del veicolo e, per un veicolo più agile, può essere più elevato.

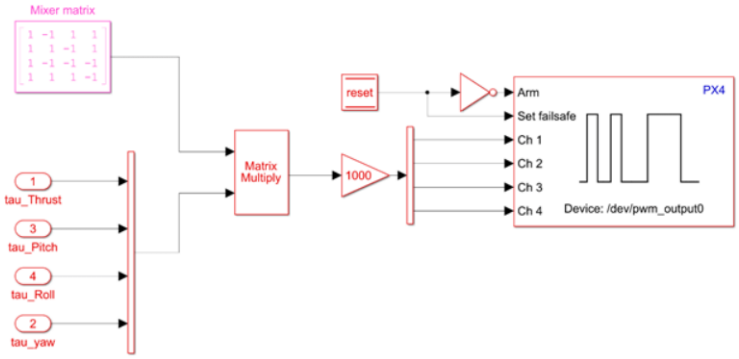

Figura 3.3: Blocco "To actuator"

Dopo l'intervento del controllore, il segnale finale viene dato in input al blocco "To actuator", visibile nella figura 3.3, attraverso 4 segnali PWM. I 4 canali Ch1,Ch2,Ch3 e Ch4 corrispondono ai motori del drone.

3.2 Iniezione guasti sui sensori

In questa sezione viene illustrata l'iniezione di guasti di sistema, essa è una pratica utilizzata per introdurre intenzionalmente guasti o errori in un sistema al fine di osservare come risponde in diverse situazioni di guasto.
L'iniezione di errori di sistema consente di indurre diversi tipi di errori di sensori e di sistema, a livello di programmazione utilizzando il plug-in di errore MAVSDK, o "manualmente" tramite una console PX4 come la shell MAVLink . Per questo progetto è stata adottata la seconda opzione. Prima dell'attività di simulazione dei guasti, il modello Simulink di partenza è stato opportunamente modificato per renderlo più idoneo allo scopo del progetto, in particolare sono stati apportati due cambiamenti principali:

- Sostituzione degli slider input con un blocco "Signal editor" per realizzare una traiettoria replicabile del drone da testare in presenza di guasti o meno.
- Aggiunta al modello di blocchi uORB read della libreria del PX4 support package per l'acquisizione dei parametri durante la simulazione.

Per quanto riguarda la prima modifica, è stato necessario progettare una traiettoria replicabile in catena aperta, in modo da poterla utilizzare come punto di riferimento per la simulazione dei guasti e la conseguente analisi dei risultati. A tal proposito, sono stati eliminati i blocchi "Input Sliders" e sostituiti con un blocco "Signal editor",

all'interno del quale è stata definita la traiettoria. Nell'immagine 3.4 è possibile vedere l'andamento della traiettoria: inizialmente il drone si alza in volo sull'asse z di una quantità pari a 10 metri. Successivamente in avanti lungo l'asse x di 10, poi lungo y di 10, indietro du x di -10 e su y di -10. Infine il drone compie un giro di 360° prima di atterrare.

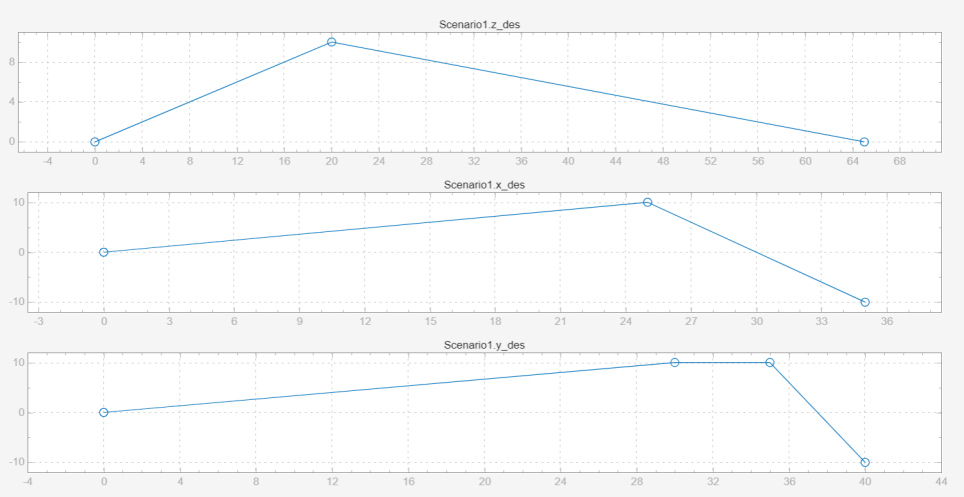

Figura 3.4: Traiettoria replicabile

Il nuovo blocco "Signal editor" appare come in figura 3.5. E' stato scelto un tempo di campionamento di 0.01.

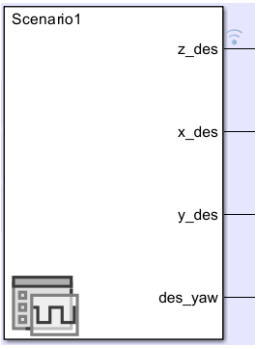

Figura 3.5: Blocco Signal Editor

Successivamente, sono stati aggiunti dei blocchi per il monitoraggio in tempo reale e l'acquisizione di alcuni parametri interessati durante la simulazione. In particolare, i blocchi "Giroscopio" e "Accellerometro" direttamente dalla libreria del PX4 support package, per andare a leggere le velocità di rotazione e le accellerazioni lungo gli assi. In seguito, sono stati utilizzati blocchi uORB read per acquisire parametri come il PWM dei motori, le velocità lineari e angolari e le posizioni. Più nel dettaglio, sono stati configurati i blocchi per la lettura delle istanze "actuator_outputs", "vehicle_odometry" e "vehicle_local_position.

Per fare ciò questi blocchi sfruttano l'uORB, un sistema di messaggistica leggero utilizzato dal framework PX4 per la comunicazione tra i moduli del firmware.

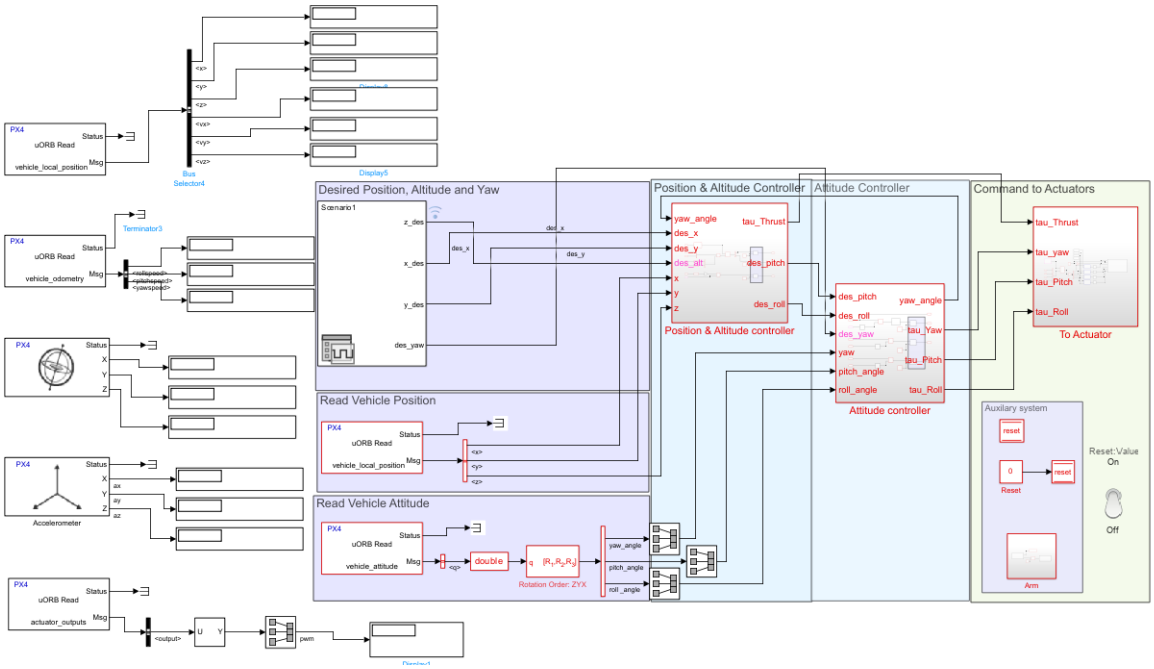

Figura 3.6: Modello modificato

Il modello modificato è quello nella figura 3.6. Durante la simulazione, i blocchi di lettura uORB read mostrano il variare dei parametri, che a fine simulazione vengono salvati nel Data Inspector, in cui è possibile vedere i grafici. Durante la pianificazione della traiettoria replicabile, è stato riscontrato un bug presente nel modello Simulink tale per cui, quando si impostava lo Yaw su un valore superiore ai 180°, il drone si comportava in modo anomalo iniziando a girare su se stesso. Il problema si verifica perchè quando l'angolo di beccheggio desiderato viene impostato su 180 gradi (o pi greco (3,14) radianti), l'angolo stimato da PX4 tende a superare i 180 gradi. Tuttavia, in questo caso, invece di raggiungere i 182 o 183 gradi, questo valore viene segnalato come -178 o -177 gradi (o -3,14 radianti). Di conseguenza, il calcolo dell'errore tra l'angolo desiderato e quello effettivo, che idealmente dovrebbe essere vicino a 0, diventa 2pi o 6,28 nel modello che viene quindi fornito al PID. Quest'ultimo, per compensare l'errore elevato, introduce ulteriori comandi attuatori compensativi che causano la rotazione incontrollata del drone.

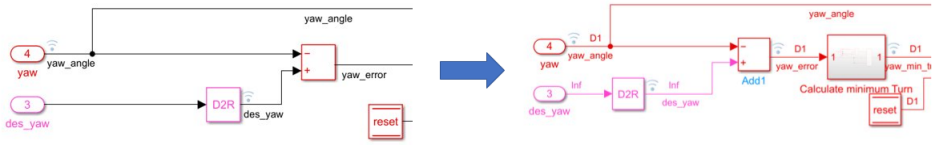

Figura 3.7: Correzione modello

Per rimediare al bug del modello è stato introdotto un sottosistema, visibile nella figura 3.7, che calcola la minima svolta che il drone deve compiere per raggiungere l'angolo di beccheggio desiderato; nella maggior parte dei casi è solo la differenza tra "des_yaw" e l'angolo di beccheggio stimato, tuttavia per casi particolari come quello sopra, calcoliamo la minima svolta che si avvicina a 0. Nel grafico "yaw_min_turn" della diapositiva precedente, anche se lo ''yaw_error" calcolato era 6.28, l'errore "min_turn" era vicino a 0 e quindi il drone non ruota più in modo incontrollato.

3.2.1 Comando Failure System

Nel seguente elaborato gli errori sono stati iniettati utilizzando il comando failure system direttamente sulla console/shell PX4, specificando sia la destinazione che il tipo di errore.
Prima di procedere con la sintassi del comando è necessario abilitare il parametro SYS_FAILURE_EN, come mostrato in figura 3.8 per iniettare i guasti sul sistema, dal momento che questo parametro di default è disabilitato.

```
pxh> param set SYS_FAILURE_EN 1
```

Figura 3.8: Set parametro SYS_FAILURE_EN

La sintassi completa del comando failure è:
failure <***componente***> <***tipo di errore***> [**-i** <***numero di istanza***>]
Dove:

- *componente* :
  - Sensori:
    - `gyro` : Giroscopio.
    - `accel` : Accelerometro.
    - `mag` : Magnetometro
    - `baro` : Barometro
    - `gps` : GPS
    - `optical_flow` : Flusso ottico.
    - `vio` : Odometria visiva inerziale.
    - `distance_sensor` : Sensore di distanza (telemetro).
    - `airspeed` : Sensore di velocità dell'aria.
  - Sistemi:
    - `battery` : Batteria.
    - `motor` : Il motore.
    - `servo` : Servo.
    - `avoidance` : Evitare.
    - `rc_signal` : Segnale RC.
    - `mavlink_signal` : Segnale MAVLink (telemetria dati).
- *tipo_errore* :
  - `ok` : Pubblica come di consueto (disabilita l'iniezione di errore).
  - `off` : interrompe la pubblicazione.
  - `stuck` : Segnala lo stesso valore ogni volta (*potrebbe* indicare un malfunzionamento del sensore).
  - `garbage` : Pubblica rumore casuale. Sembra leggere una memoria non inizializzata.
  - `wrong` : pubblica valori non validi (che sembrano ancora ragionevoli/non sono "spazzatura").
  - `slow` : Pubblica a tariffa ridotta.
  - `delayed` : Pubblica dati validi con un notevole ritardo.
  - `intermittent` : Pubblica a intermittenza.
- *numero di istanza* (facoltativo): numero di istanza del sensore interessato. 0 (predefinito) indica tutti i sensori del tipo specificato.

Figura 3.9: Tipi di guasto

Nella figura 3.9 sono riportate le componenti su cui possono essere iniettati i guasti, che si dividono in sensori e sistemi. La tipologia di guasto, come per esempio "off" che interrompe la pubblicazione dei valori acquisiti, simulando un malfunzionamento. Infine, con il numero di istanze è possibile specificare il numero di sensori o sistemi che saranno affetti dal guasto.

Di seguito viene riportato un esempio che simula un errore nell'accelerometro utilizzando il comando failure. Inanzitutto bisogna abilitare il parametro SYS_FAILURE_EN, Fig. 3.8; nel caso si vuole visualizzare il valore corrente del parametro SYS_FAILURE_EN bisogna lanciare il comando riportato in Figura 3.10.

```
pxh> param show SYS_FAILURE_EN
Symbols: x = used, + = saved, * = unsaved
x + SYS_FAILURE_EN [601,1168] : 1

 614/1441 parameters used.
```

Figura 3.10: Lettura parametro SYS_FAILURE_EN

Di seguito viene riportata la sintassi del comando failure sull'accellerometro (Fig. 3.11) :

```
pxh> failure accel off
WARN  [failure] inject failure unit: accel (1), type: off (1), instance: 0
WARN  [simulator] CMD_INJECT_FAILURE, accel 0 off
WARN  [simulator] CMD_INJECT_FAILURE, accel 1 off
WARN  [simulator] CMD_INJECT_FAILURE, accel 2 off
```

Figura 3.11: Tipi di guasto

In questo esempio, è stato disattivato il sensore accellerometro in modo che nel modello Simulink venisse bloccata la pubblicazione dei valori acquisiti. Alla fine della simulazione, è possibile plottare tutti i parametri in "Data Inspector" e analizzare come variano i vari segnali come ad esempio la posizione del veicolo, velocità lineare o angolare, ect. con e senza l'iniezione del guasto sul sistema. L'analisi del seguente esempio viene riportato successivamente nel capitolo dei risultati.
Dopo numerose simulazioni sui diversi tipi di sensori e diversi tipi di guasto iniettati ad essi, si è andati incontro a un errore durante l'iniezione di guasti.
A conferma di ciò, è stato analizzato il file sorgente "SimulatorMavlink.cpp" presente nel repository di PX4 [2], ed è stato appurato che la maggior parte dei guasti non sono stati ancora definiti dagli sviluppatori di PX4-Autopilot. Inoltre nella documentazione sull'iniezione di guasti [3] è presente a inizio pagina un avviso (Fig. 3.12) che indica che la suddetta funzione è ancora in fase di sviluppo.

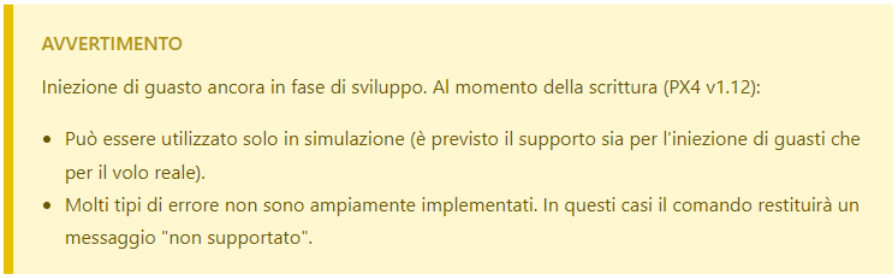

Figura 3.12: Avviso Iniezione guasti

In Figura 3.13, vengono riportati vari test per verificare l'indisponibilità dei guasti, provando ad iniettare dei guasti che non sono definiti sul file "SimulatorMavlink.cpp" e, come già anticipato, si incorre in un errore del comando failure.
L'errore indica che il sistema non riconosce tali guasti o che potrebbe essere necessario un ulteriore sviluppo o configurazione per abilitare questa funzionalità in modo corretto.

```
pxh> failure gyro wrong
WARN  [failure] inject failure unit: gyro (0), type: wrong (4), instan
ce: 0
ERROR [failure] Result: 3
Command 'failure' failed, returned 1.
pxh> failure gyro slow
WARN  [failure] inject failure unit: gyro (0), type: slow (5), instanc
e: 0
ERROR [failure] Result: 3
Command 'failure' failed, returned 1.
pxh> failure gyro wrong
WARN  [failure] inject failure unit: gyro (0), type: wrong (4), instan
ce: 0
ERROR [failure] Result: 3
Command 'failure' failed, returned 1.
pxh> WARN  [ekf2] primary EKF changed 0 (timeout) -> 4
extmodeEvent error: code -10
pxh> failure accel garbage
WARN  [failure] inject failure unit: accel (1), type: garbage (3), ins
tance: 0
ERROR [failure] Result: 3
Command 'failure' failed, returned 1.
pxh> failure accel slow
WARN  [failure] inject failure unit: accel (1), type: slow (5), instan
ce: 0
ERROR [failure] Result: 3
Command 'failure' failed, returned 1.
```

Figura 3.13: Failure error

In conclusione, nella figura 3.14 è riportata una tabella riassuntiva in cui vengono riportati i guasti iniettabili:

	OK	OFF	STUCK	GARBAGE	WRONG	SLOW	DELAYED	INTERMITTENT
Gyro	✓	✓	✓	×	×	×	×	×
Accell	✓	✓	✓	×	×	×	×	×
Mag	✓	✓	✓	×	×	×	×	×
Baro	✓	✓	✓	×	×	×	×	×
GPS	✓	✓	×	×	×	×	×	×
Optical flow	×	×	×	×	×	×	×	×
Vio	✓	✓	×	×	×	×	×	×
Distance sensor	×	×	×	×	×	×	×	×
Airspeed	✓	✓	×	×	×	×	×	×
Battery	×	×	×	×	×	×	×	×
Motor	×	×	×	×	×	×	×	×
Servo	×	×	×	×	×	×	×	×
Avoidance	×	×	×	×	×	×	×	×
RC signal	×	×	×	×	×	×	×	×
Mavlink signal	×	×	×	×	×	×	×	×

Figura 3.14: Guasti iniettabili

3.3 Iniezione di guasti moltiplicativi sulle componenti di attuazione

Dal momento che la maggior parte dei guasti presenti nella libreria di PX4 sono ancora in fase di implementazione, è stato possibile iniettare dalla shell di PX4 solo guasti di tipo off e stuck su alcuni sensori come il giroscopio e l'accelerometro. Per questo motivo, sono stati implementati anche dei guasti di tipo moltiplicativo agendo

direttamente sul blocco "To Actuator" del modello Simulink. Questo ha permesso di introdurre dapprima in uno solo dei canali, poi estendendolo agli altri, un guadagno dal valore inferiore a 1 per simulare la perdita di potenza dei motori.

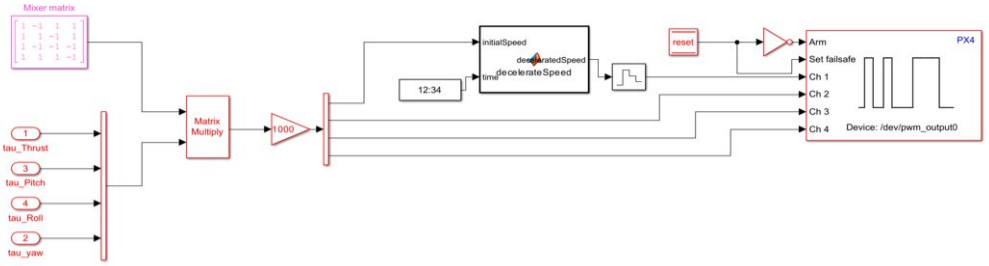

Figura 3.15: Guasto moltiplicativo

Nella figura 3.15 è presente il contenuto del blocco "To Actuator" modificato. Per prima cosa è stata definita una funzione "DeceleratedSpeed" utilizzando il blocco "Matlab Function". Questa funzione (figura 3.16) permette una graduale delecerazione dei motori, ed ha come ingressi la velocità iniziale e il tempo della simulazione corrente. L'uscita della funzione viene discretizzata da una TOZ (tenuta di ordine zero) visto che il modello è a tempo discreto.

```
function deceleratedSpeed = decelerateSpeed(initialSpeed, time)
    persistent decelerationStarted;

    if isempty(decelerationStarted)
        decelerationStarted = false;
    end

   if time >= 30 && ~decelerationStarted
        decelerationStarted = true;
    end

    if decelerationStarted
        deceleratedSpeed = initialSpeed * exp(-0.07);
    else
        deceleratedSpeed = initialSpeed;
    end
end
```

Figura 3.16: Funzione "DeceleratedSpeed"

E' stata utilizzata una funzionale esponenziale decrescente per ridurre gradualmente la velocità del motore. Nell'esempio riportato la decelerazione inizia dopo 30 secondi di simulazione e andiamo a decrementare la velocità iniziale del 7% sul motore 1. Successivamente il guasto moltiplicativo è stato esteso gradualmente a tutti i motori,

a distanza di tot secondi l'uno dall'altro. Il blocco "To Actuator" si modifica come in figura 3.17

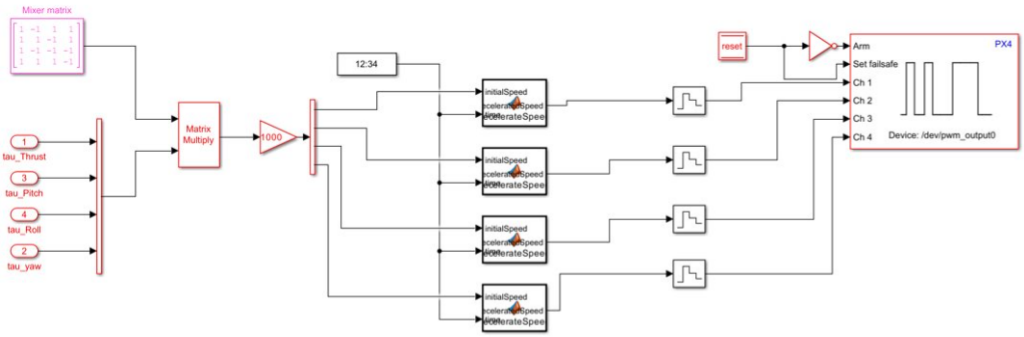

Figura 3.17: Guasto moltiplicativo su 4 motori

Capitolo 4

Risultati

Prima di presentare i vari risultati è bene precisare che all'interno del blocco uORB Read è possibile modificare quali parametri di riferimento vengono usati per il controllo del drone, abbiamo quattro instanze diverse:

- *Vehicle attitude groundtruth*: rappresenta l'orientamento di riferimento del veicolo e fornisce un valore esatto senza l'uso di algoritmi di stima. In altre parole vengono forniti direttamente i valori corretti e non la stima di PX4 ottenuta dai sensori.
- *Vehicle attitude*: rappresenta l'orientamento del drone misurato direttamente dai sensori a bordo, come giroscopi e accellerometri. Questo va a stimare quindi il rollio (roll), il beccheggio (pitch) e l'imbardata (yaw). I sensori forniscono misure grezze, che possono essere affette da rumore o errori di misurazione.
- *Estimator attitude*: rappresenta l'orientamento stimato dal filtro di Kalman. Questo viene stimato combinando le misurazioni provenienti dai sensori a bordo, come giroscopi e accellerometri.
- *Vehicle vision attitude*: rappresenta l'orientamento del veicolo stimata utilizzando informazioni provenienti dai sensori di visione a bordo del drone, che possono includere telecamere o sensori di profondità. In base a questi strumenti il drone stima la sua posizione usando punti di riferimento visivi.

4.1 Vehicle attitude groundtruth

Con l'istanza *vehicle attitude groundtruth* è stato notato che l'effetto dei guasti off e stuck, in giroscopio e accellerometro, non influivano sui residui degli angoli di assetto roll, pitch e yaw, in quanto non vengono usati algoritmi di stima e questi ultimi non venivano usati per l'assetto del drone in quanto il simulatore utilizzava i valori esatti e non quelli forniti dai sensori, pertanto il drone non veniva minimamenti influenzato. Qui riportati i vari grafici dei residui.

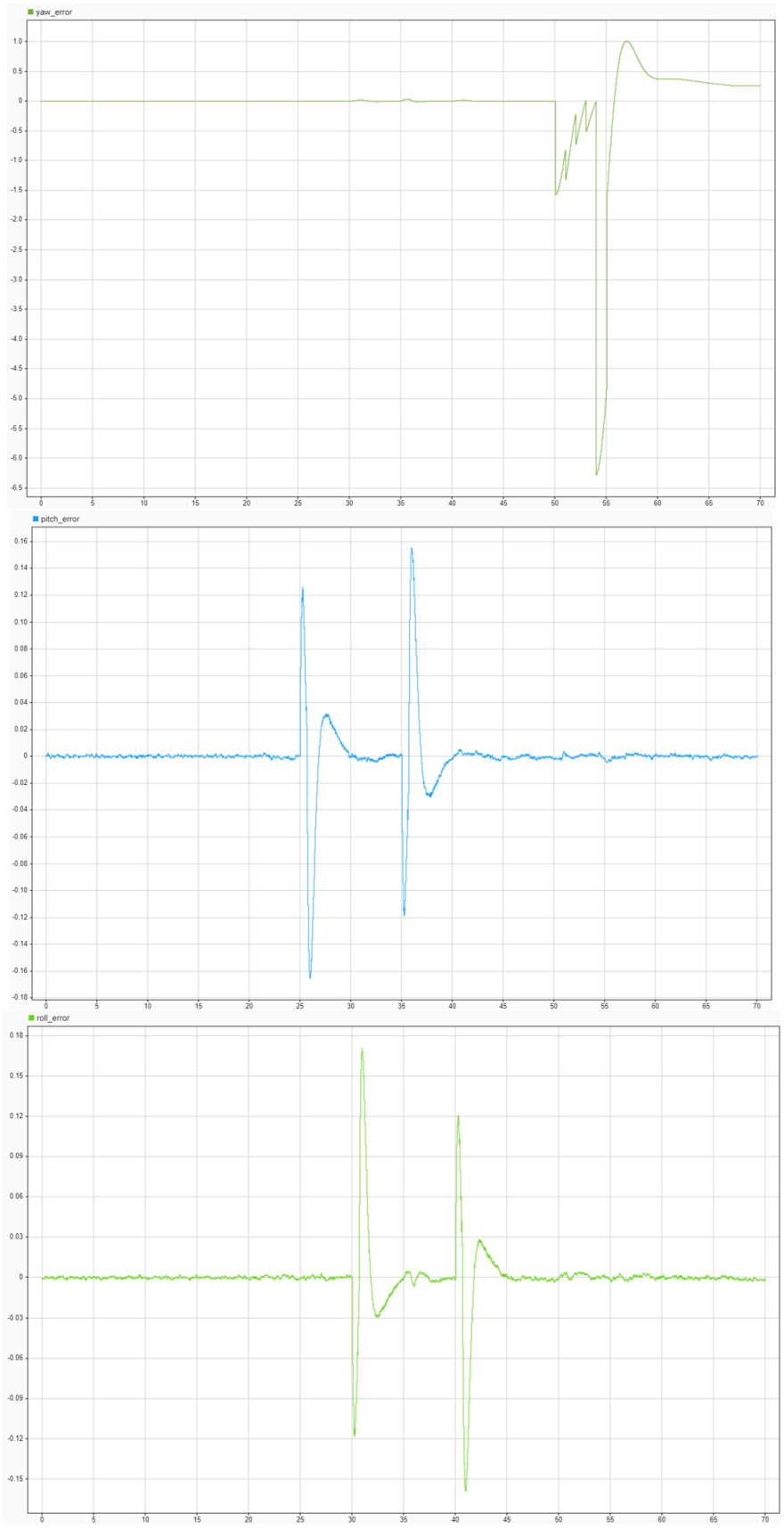

Figura 4.1: Residui con guasti off e stuck (asse y: rad , asse x: s)

4.2 Estimator attitude

Con l'istanza *Estimator attitude* viene stimato l'orientamento del drone attraverso l'utilizzo del filtro di Kalman che riceve le misure dai sensori a bordo del drone come giroscopio ed accelerometro pertanto innescando guasti su quest'ultimi il drone in simulazione ne risente, modificando la sua traiettoria o cadendo a terra. Infatti iniettando i vari guasti è possibile notare come i residui vengano modificati rispetto a un andamento senza guasti.

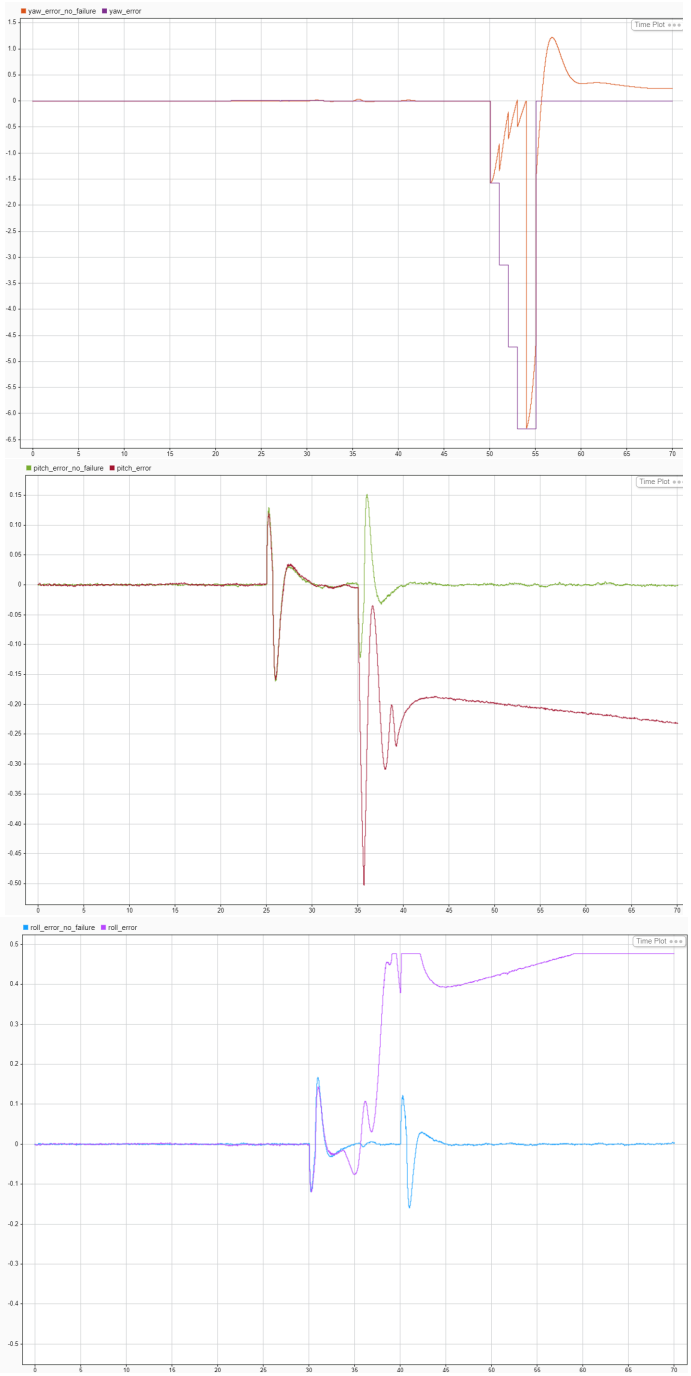

Figura 4.2: Residui con guasto off su accelerometro (asse y: rad , asse x: sec)

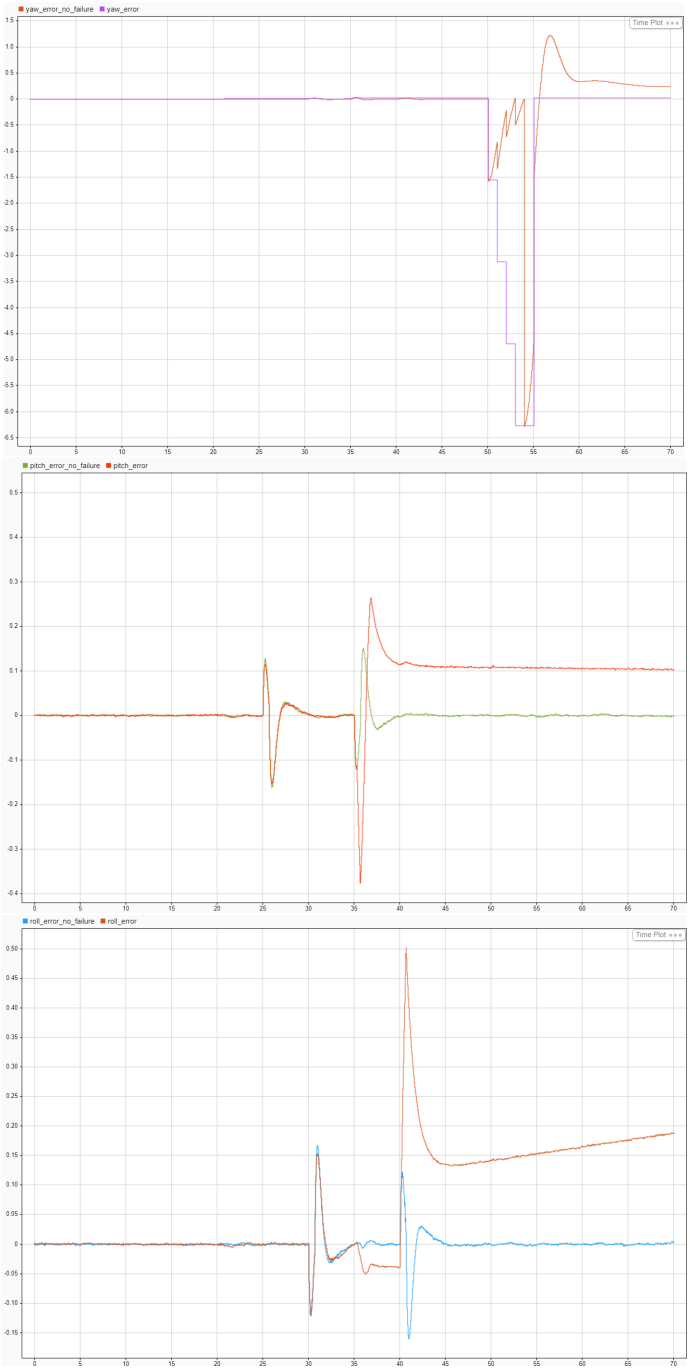

Figura 4.3: Residui con guasto stuck su accelerometro (asse y: rad , asse x: sec)

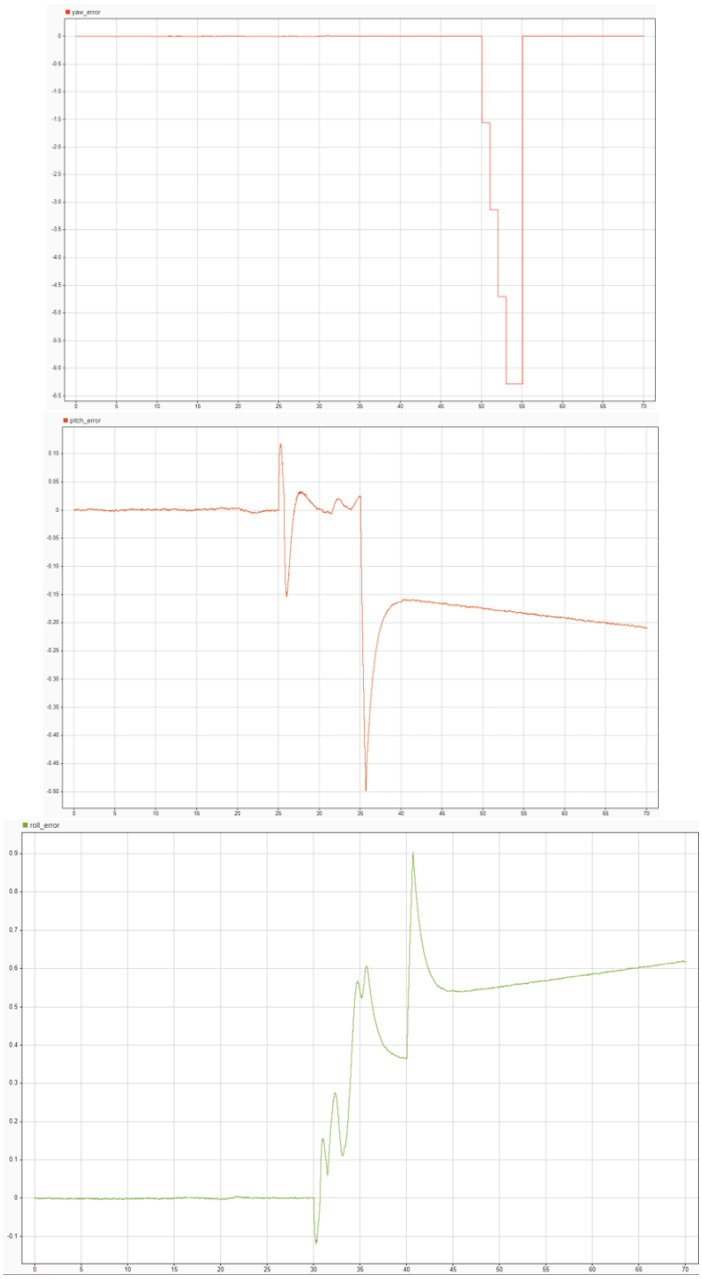

Figura 4.4: Residui con guasto off su giroscopio (asse y: rad , asse x: sec)

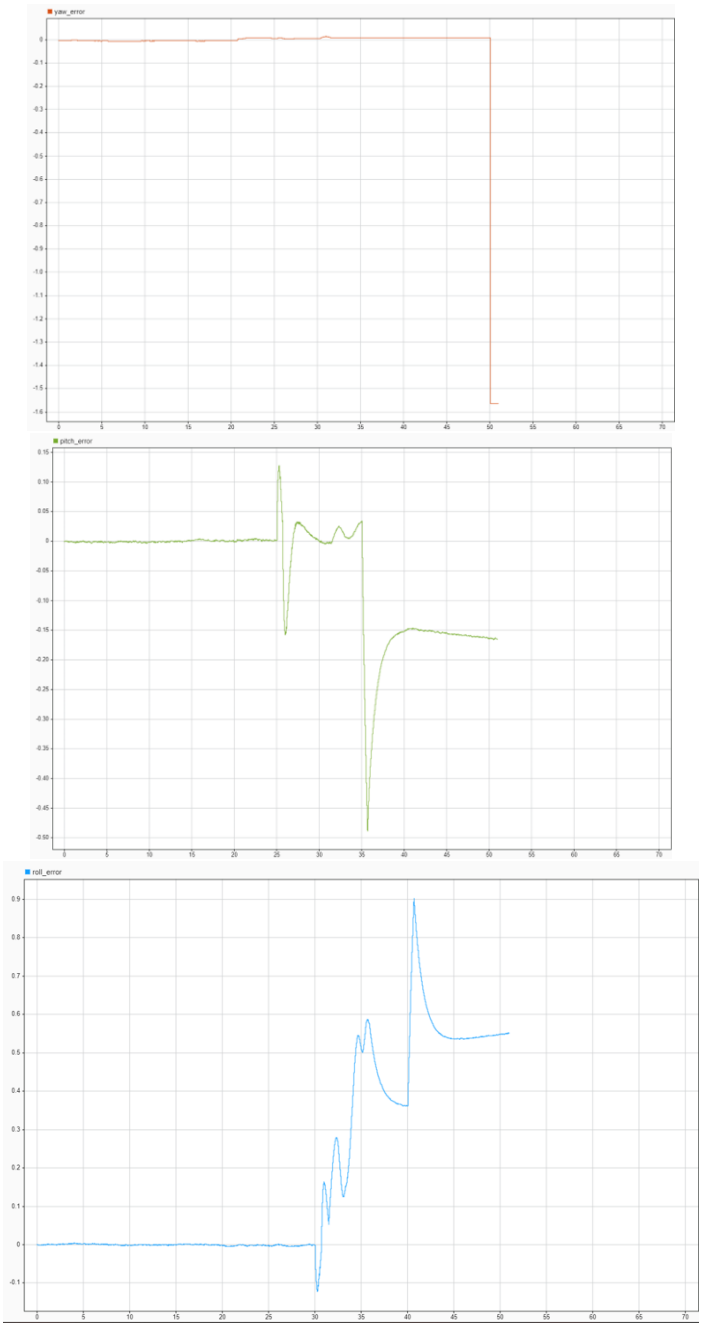

Figura 4.5: Residui con guasto stuck su giroscopio (asse y: rad , asse x: sec)

4.3 Vehicle attitude

Attraverso l'istanza *Vehicle attitude* l'orientamento del drone viene misurato e stimato direttamente dai sensori presenti a bordo del drone come appunto giroscopio ed accelerometro pertanto si è notato che iniettando guasti su questi ultimi il drone in simulazione non manteneva la propria traiettoria o precipitava, pertanto i guasti influivano sul drone, come si puo vedere dai residui qui riportati.

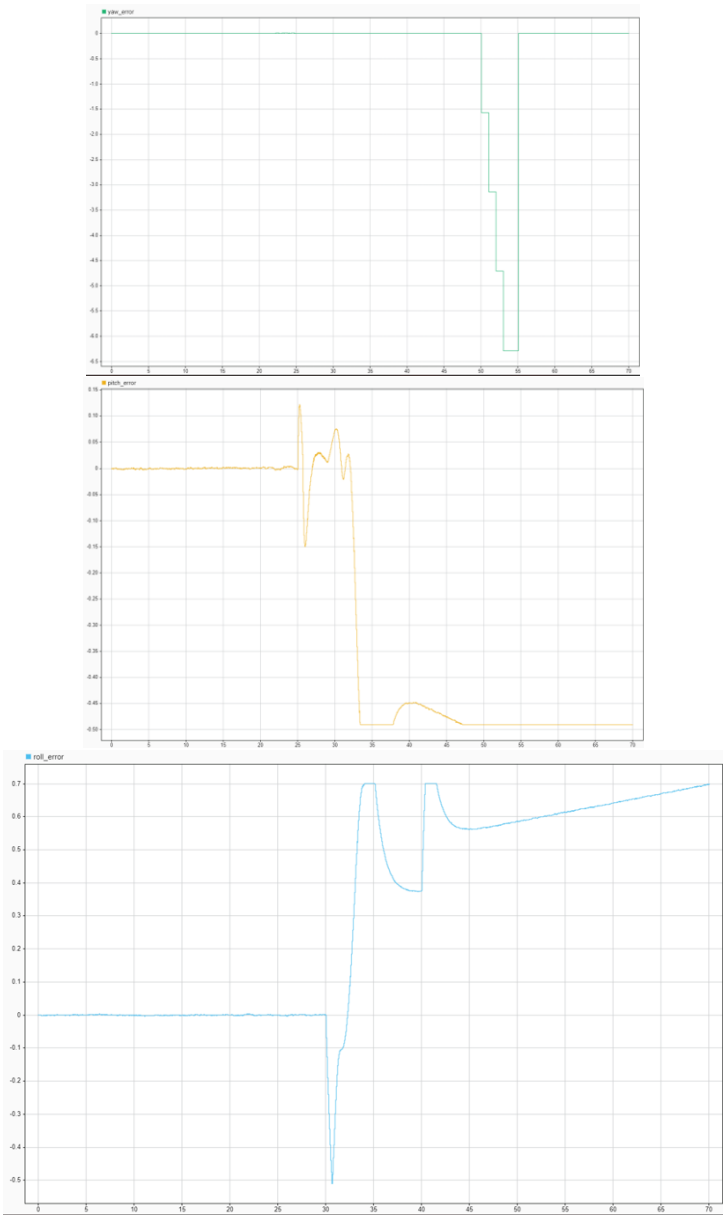

Figura 4.6: Residui con guasto off su accelerometro (asse y: rad , asse x: sec)

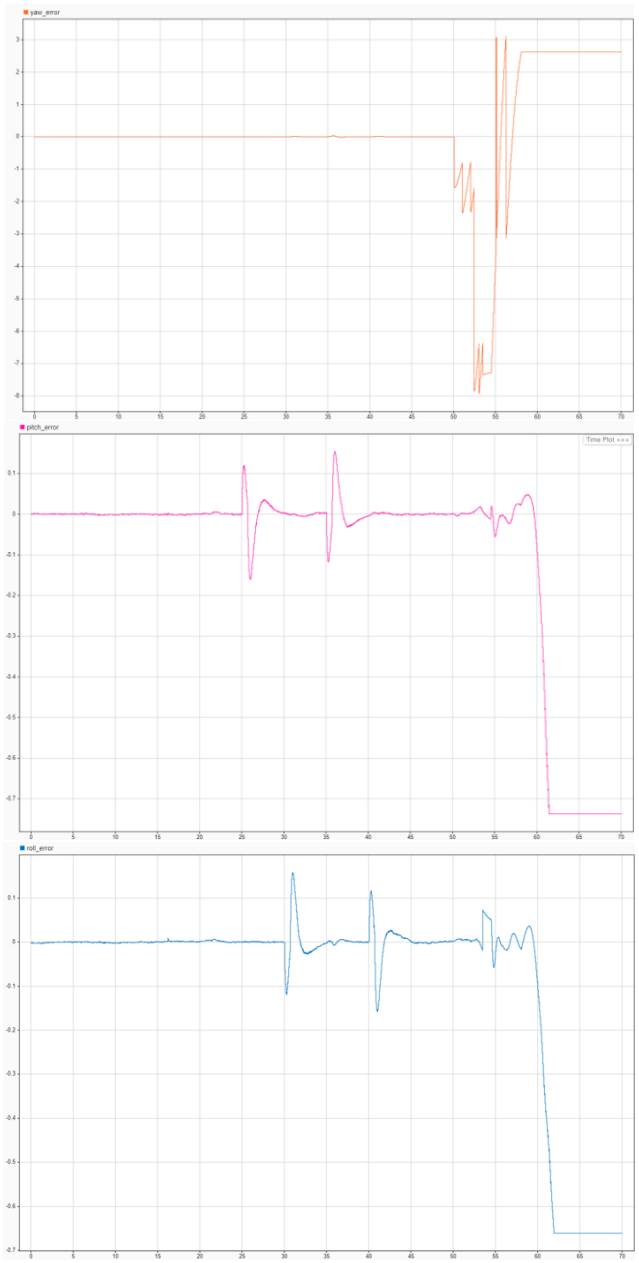

Figura 4.7: Residui con guasto stuck su accelerometro (asse y: rad , asse x: sec)

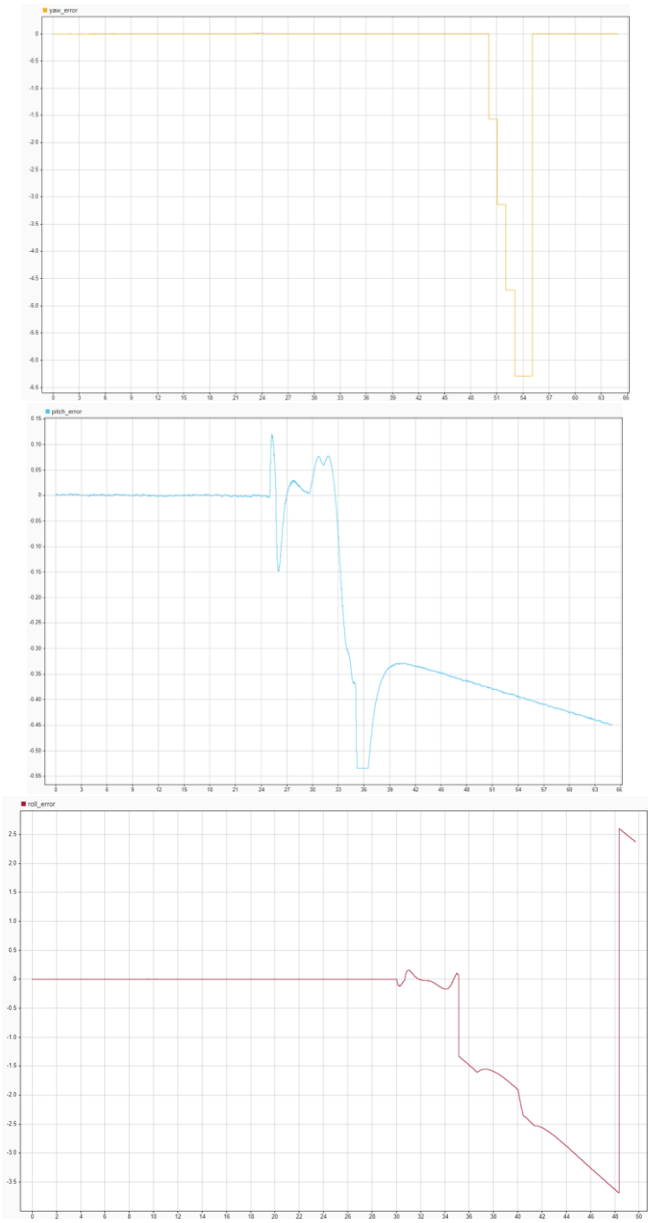

Figura 4.8: Residui con guasto off su giroscopio (asse y: rad , asse x: sec)

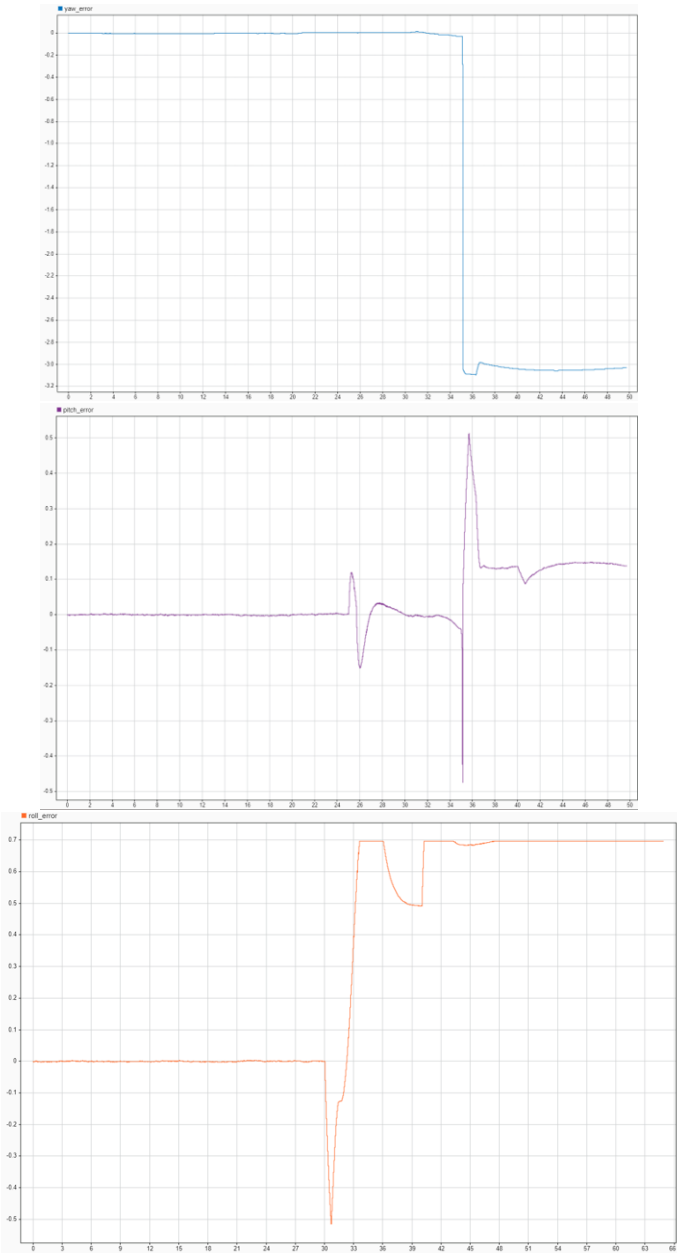

Figura 4.9: Residui con guasto stuck su giroscopio (asse y: rad/s , asse x: sec)

4.4 Guasto moltiplicativo sui motori

Sono state effettuate varie simulazione di guasto per individuare il guadagno massimo iniettabile sul singolo motore entro il quale il drone riesce a rimanere in equilibro e a percorrere la traiettoria desiderata. Al momento dell'iniezione del guasto, il drone oscilla, e impiega un certo intervallo di tempo per ristabilire la traiettoria, in qunato il guasto viene iniettato ad inizio simulazione. Qui riportate le simulaizoni di guasto sul motore 1 con una percentuale del 7 % .

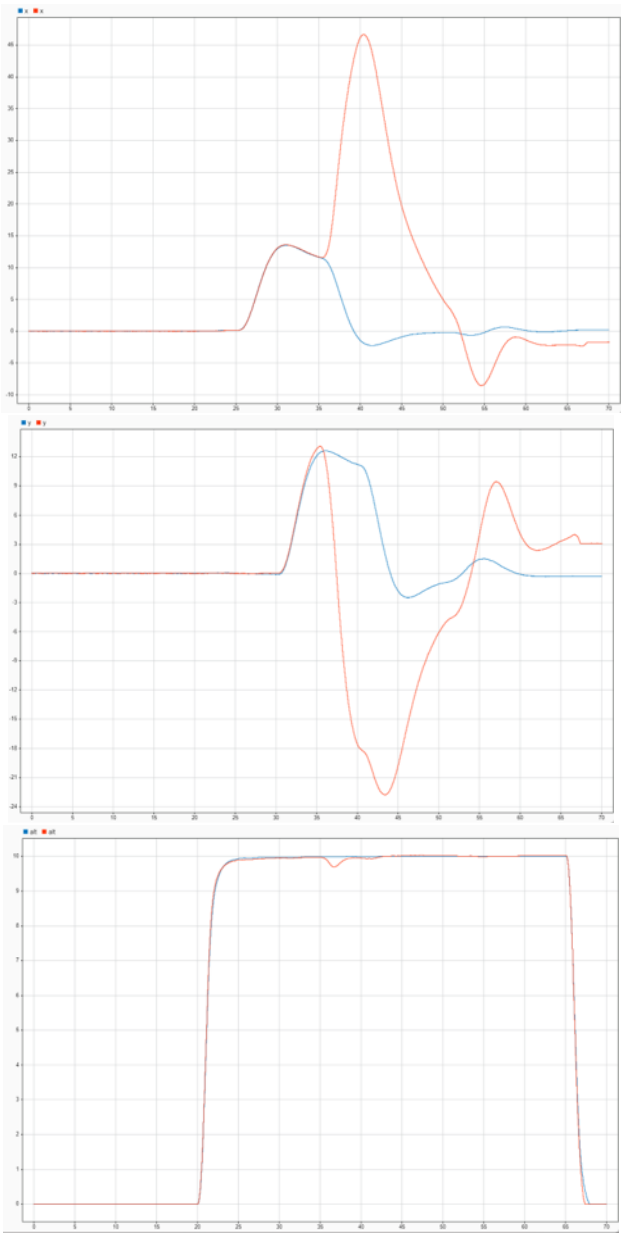

Figura 4.10: In rosso x,y,z con guasto, in blu x,y,z senza guasto (asse y: metri , asse x:sec)

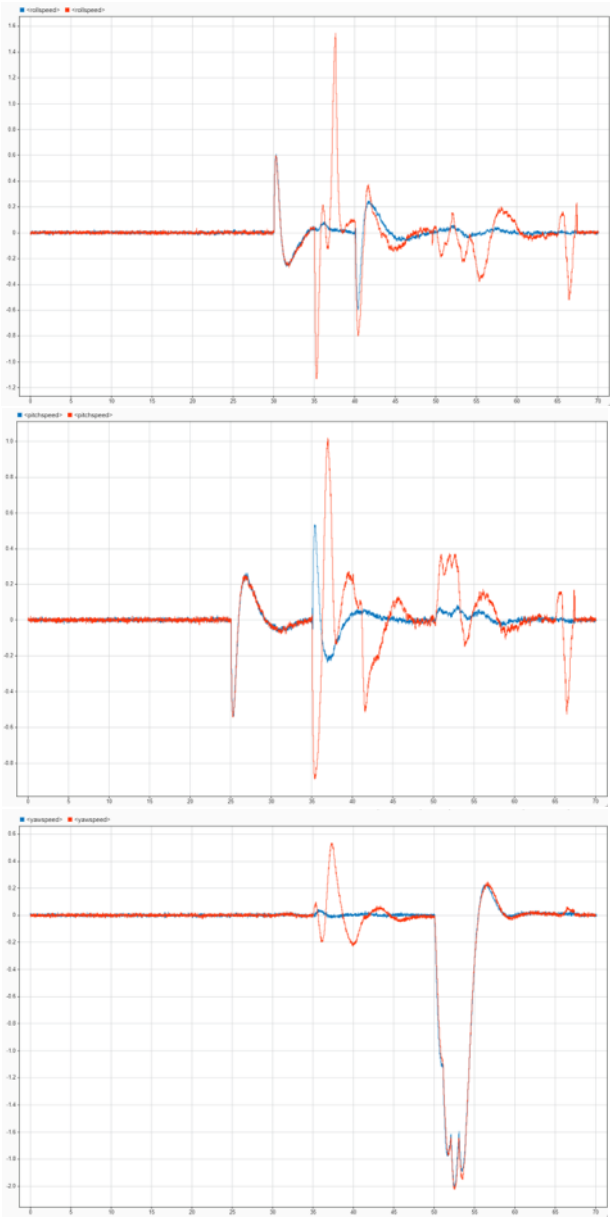

Figura 4.11: In rosso le velocità angolari con guasto, in blu le velocità angolari senza guasto (asse y: rad/s , asse x: sec)

Poi è stata effettutata una simulazione con un guasto al 20 % su tutti e 4 i motori nello stesso istante. Abbiamo notato che iniettando il guasto simultaneamente su tutti i motori, essendo quest'ultimo distribuito uniformemente, il drone mantiene la traiettoria desiderata senza troppi problemi.

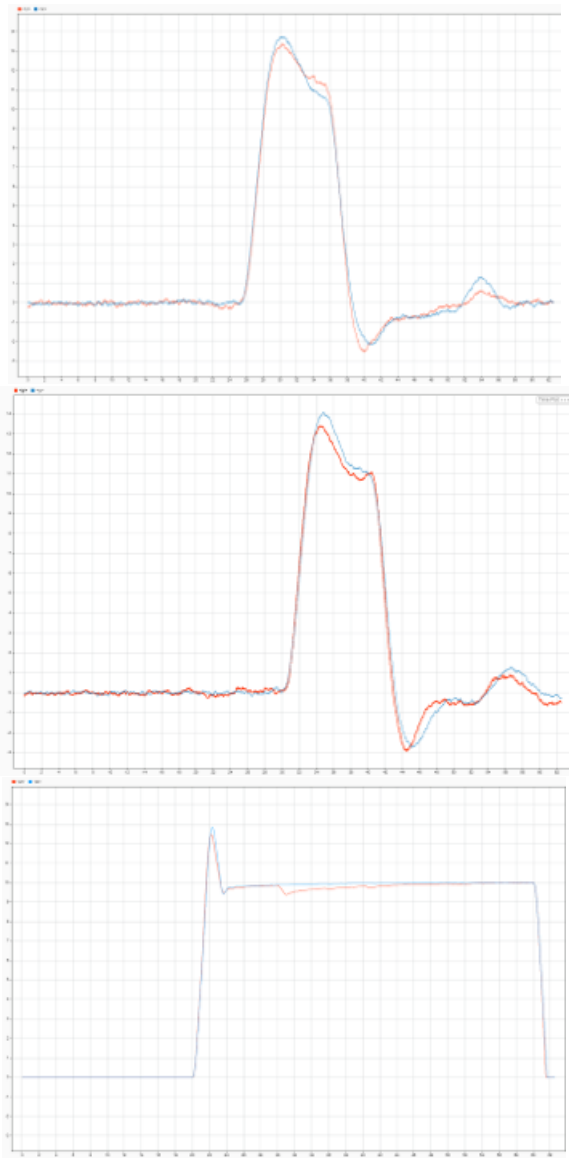

Figura 4.12: In rosso x,y,z con guasto, in blu x,y,z senza guasto (asse y: metri , asse x: sec)

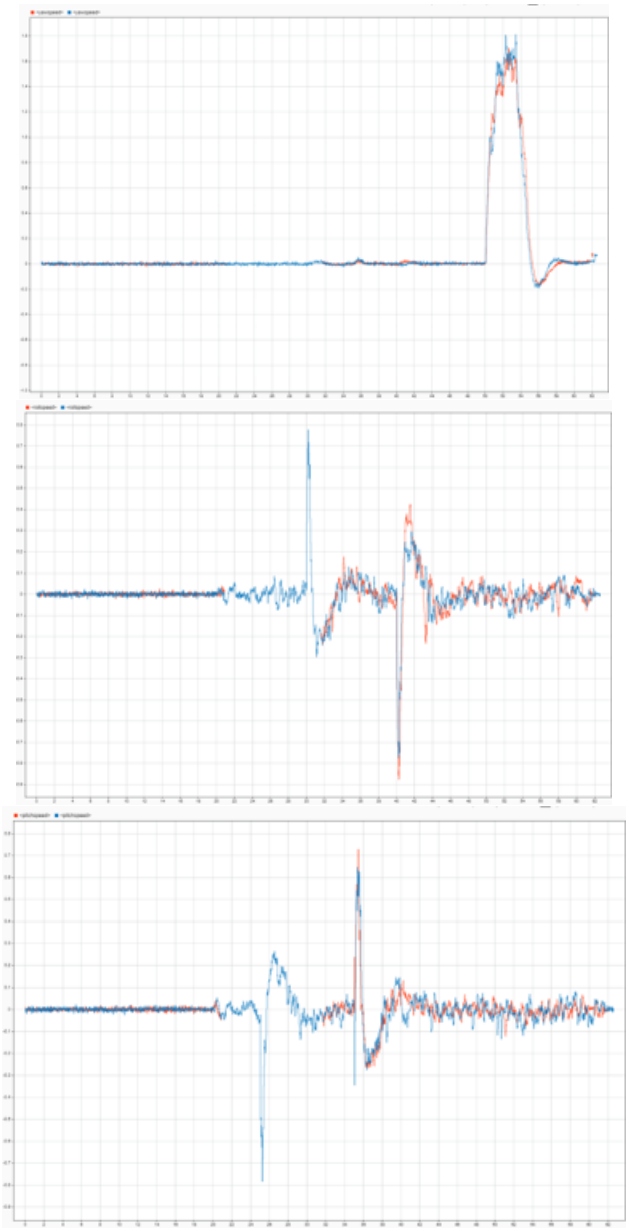

Figura 4.13: In rosso le velocità angolari con guasto, in blu le velocità angolari senza guasto (asse y: rad/s , asse x: sec)

Infine è stata eseguita un ultima simulazione con un guasto al 3% a scalare ogni due istanti di tempo su ogni motore.

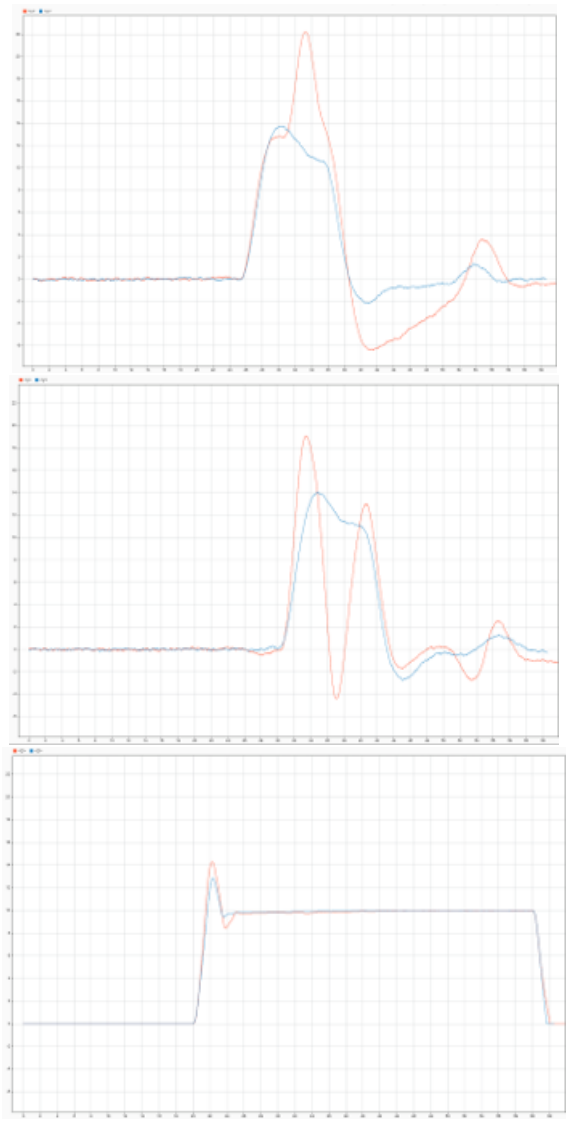

Figura 4.14: In rosso x,y,z con guasto, in blu x,y,z senza guasto (asse y: metri , asse x: sec)

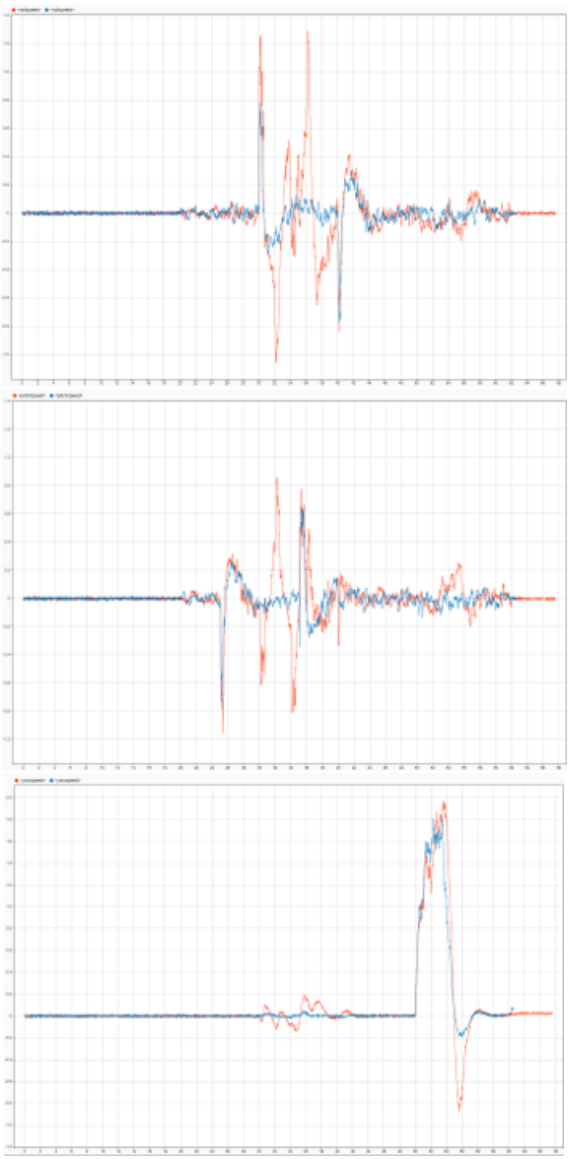

Figura 4.15: In rosso le velocità angolari con guasto, in blu le velocità angolari senza guasto (asse y: rad/s , asse x: sec)

Capitolo 5

Conslusioni e sviluppi futuri

Con lo sviluppo di questo progetto è stata realizzata una connessione Software-In-The-Loop tra Matlab Simulink e PX4, sfruttando il PX4 support package for UAV Toolbox e le sue librerie. Più nel dettaglio, nelle prime fasi ci si è concentrati sul registrare e acquisire i log di volo del drone, per poi avere una base di dati su cui lavorare. Ad esempio sono stati monitorati sensori presenti sul drone, come accelerometro e giroscopio, e anche i segnali PWM generati per i 4 motori. Successivamente è stata realizzara una traiettoria replicabile da utilizzare come base per l'iniezione di guasti, in modo tale da confrontare l'andamento "fault free" da quello con guasti. Infine, questi ultimi sono stati iniettati prima sulle componenti di sensoristica tramite comandi da shell PX4 e poi sulle componenti di attuazione, simulando un guasto moltiplicativo con una funzione realizzata su Matlab per decelerare i motori durante la simulazione.

5.1 Sviluppi futuri

Questo progetto ha posto le basi per lo sviluppo e l'implementazione di simulazioni per il volo di droni in un ambiente controllato, con la possibilità di studiare l'effetto dei guasti su questi dispositivi prima di passare ai test sull'hardware. Vi sono ancora degli elementi che possono essere introdotti per migliorare e approfondire l'effetto dei guasti. In primo luogo, la mancanza di una completa implementazione dei guasti applicabili attraverso i comandi Mavlink dalla shell di PX4 ha limitato la possibilità di approfondire l'iniezione di guasti sulle componenti di sensoristica e attuazione. Una parentesi aperta che può senza dubbio essere approfondita con l'intento di esplorare tipologie di guasti differenti e su componenti e sistemi diversi, come mostrato in figura 3.9.
In secondo luogo, è aperta la possibilità di includere nella simulazione con PX4 il software QGroundControl, un'applicazione open source sviluppata da Dronecode, che offre un'interfaccia grafica utente (GUI) molto intuitiva, e mette a disposizione numerose features:

- Pianificazione del volo, attraverso la definizione di waypoints, percorsi, altezza, velocità e altre impostazioni di volo.

- Interfaccia utente intuitiva: QGroundControl offre un'interfaccia utente user-friendly con elementi visivi e comandi intuitivi, rendendo più facile l'interazione con il software.
- Mappa interattiva: La mappa interattiva integrata consente agli utenti di visualizzare la posizione del veicolo in tempo reale, oltre a mostrare la traiettoria della missione programmata.
- Telemetria e monitoraggio: Il software fornisce dati di telemetria in tempo reale, come l'altitudine, la velocità, la posizione GPS e altri parametri di stato del veicolo.
- Modalità di volo: QGroundControl offre diverse modalità di volo, come il volo manuale, il volo autonomo e altre modalità specializzate a seconda delle funzionalità del veicolo.

QGroundControl potrebbe essere uno strumento aggiuntivo interessante da integrare nella simulazione tra PX4 e Simulink.

5.2 Conclusioni

Lo sviluppo di questo progetto è stato un'esperienza molto formativa che ha permesso di approfondire le conoscenze sull'iniezione di guasti e sul loro impatto sul comportamento di un drone quadrirotore. Il lavoro svolto rappresenta un importante punto di partenza per futuri progetti, introducendo nuove funzionalità e obiettivi. In generale, questo progetto ha fornito un'ottima occasione per acquisire nuove conoscenze e competenze, che possono essere utilizzate in futuro per migliorare la progettazione e lo sviluppo di sistemi di controllo per droni.
Nel seguente repository Github sono presenti i modelli Matlab utilizzati nel progetto e i grafici relativi alle simulazioni effettuate: https://github.com/adelioA/Manutenione-preventiva-2022-2023

Bibliografia

[1] Dronecode foundation. Px4 installer. https://github.com/PX4/PX4-windows-toolchain/releases/tag/v0.8.

[2] Repository di px4. https://github.com/PX4/PX4-Autopilot/blob/main/src/modules/simulation/simulator_mavlink/SimulatorMavlink.cpp.

[3] Documentazione sull'iniezione guasti. https://docs.px4.io/main/en/debug/failure_injection.html.