UNIVERSITA' POLITECNICA DELLE MARCHE

REALIZZAZIONE OSSERVATORE DIAGNOSTICO SU ARDUPILOT SITL

Cingolani Cristian

Caprari David

Anno Accademico 2021-22

INDICE

1 INTRODUZIONE

1.1 OBIETTIVO PROGETTO

L'obiettivo del progetto è quello di replicare il modello di un drone specifico all'interno del simulatore AirSim, progettare un osservatore interno al simulatore e confrontare le uscite del simulatore con quelle dell'osservatore, anche in caso di guasto simulato.

1.2 TEORIA OSSERVATORE

Nel progetto viene impiegato un osservatore per l'individuazione dei guasti. Quindi è necessaria una introduzione sui principi di base che stanno dietro a questa tecnica.

Si vuole individuare il guasto a partire da un confronto tra il comportamento del processo reale e quello di un modello in grado di descrivere il comportamento del processo. Rientra, dunque, assieme alle equazioni di parità, tra le tecniche basate su modello. Nel caso particolare si ha un approccio mediante rappresentazione in spazio di stato in quanto l'osservatore ricostruisce le variabili di stato in funzione delle condizioni iniziali e dell'evoluzione temporale delle misure di ingresso e di uscita del processo reale.

Si parte da un processo LTI che si può descrivere nel seguente modo:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

Figura 1

L'osservatore permette poi di ricostruire lo stato del processo:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + He(t)$$
$$e(t) \doteq y(t) - C\hat{x}(t)$$

Figura 2

Se il sistema è osservabile si giunge alla seguente formula:

$$\dot{\hat{x}}(t) = [A - HC]\hat{x}(t) + Bu(t) + Hy(t)$$

Figura 3

L'errore dello stato, che rappresenta il residuo utile per l'individuazione dei guasti, viene calcolato come differenza tra lo stato del processo reale e quello stimato dall'osservatore. Un osservatore stabile deve portare l'errore ad assumere un valore che tende a zero per tempi che vanno all'infinito.

1.3 DESCRIZIONE STRUMENTI

La simulazione virtuale del drone oggetto di progetto viene gestita dal software AirSim, diffuso simulatore per droni, auto e altro, sviluppato sulle spalle del motore grafico Unreal Engine.

AirSim è uno strumento open-source e cross platform che permette l'implementazione in Software-In-The-Loop o Hardware-In-The-Loop di diversi tra i più popolari controllori di volo. Si poggia su Unreal Engine in modo che la simulazione possa essere facilmente implementata in un Unreal Environment, cioè un ambiente di simulazione fisico e grafico che può, con l'utilizzo degli strumenti del motore grafico, rappresentare ambienti realistici. Gli sviluppatori del pacchetto si auspicano che in questo modo sia possibile svolgere ricerca su diversi fronti del volo autonomo legato ad algoritmi di intelligenza artificiale e computer vision.

Unreal Engine è un potente motore grafico che permette la generazione di spazi virtuali realistici. Utilizza diversi algoritmi avanzati per la gestione della fisica, il che lo rende ottimale per la gestione della simulazione scopo del progetto.

ArduPilot è un progetto open-source che funge da contenitore di molti e svariati tool per la gestione e il controllo di veicoli, autonomi e non. Implementa, tra i tanti, gli strumenti di controllo e comunicazione attraverso il protocollo MAVLink che consente la comunicazione con piccoli veicoli senza pilota. Grazie quindi alle interfacce messe a disposizione dallo strumento, funge da vera e propria Ground Control Station in grado di dialogare col veicolo e ottenerne dati di volo.

Al fine di elaborare opportunamente i dati ottenuti dall'analisi dei parametri di volo del drone e al fine di implementare uno strumento che faccia anch'esso da Ground Control Station in cui implementare le soluzioni software previste dallo svolgimento di progetto, si è scelto di creare uno script in linguaggio Python. Per questo motivo, sono state utilizzate diverse librerie. Per quanto riguarda la realizzazione della dinamica lineare dell'osservatore diagnostico è stata utilizzata la libreria Python Control che mette a disposizione strumenti di base per l'analisi e il design di sistemi di controllo in feedback.
Per la gestione di vettori e strumenti per la visualizzazione grafica dei dati sono state utilizzate le librerie NumPy e MatPlotLib.

2 INSTALLAZIONE

2.1 INSTALLAZIONE STRUMENTI

Gli strumenti descritti precedentemente sono stati installati in ambiente GNU/Linux su una macchina con Ubuntu 20.04. La scelta è dovuta a criteri di preferenze personali e dalla ristretta compatibilità di alcuni strumenti con distribuzioni differenti.

Per l'installazione di AirSim e Unreal Engine è stata seguita la guida presente a questo link.

Successivamente, per l'installazione di ArdupilotSITL è stata seguita la guida presente a questo link.

Python3 è distribuito di default nel sistema operativo Ubuntu utilizzato, ma non il suo gestore dei pacchetti *pip* che è stato installato con l'utilizzo del comando:

sudo apt install python-pip

Attraverso *pip* è stato possibile installare la libreria Python Control con l'utilizzo del comando:

pip install control

Lo stesso comando viene utilizzato anche per l'installazione delle librerie adite alla stampa dei grafici, MatPlotLib, e per le librerie necessarie alla comunicazione col sistema MAVProxy, pymavlink, nonché per le librerie per la gestione di vettori e matrici, NumPy.

2.2 PREDISPOSIZIONE DELLA SIMULAZIONE

Affinché sia possibile simulare il volo del drone in una specifica posizione e settando i giusti parametri di AirSim, è necessario inserire nella cartella */home/Documents/AirSim* un file in estensione .json in cui porre alcune iniziali configurazioni di cui ne è riportato un esempio nell'immagine seguente.

```
{
  "SettingsVersion": 1.2,
  "LogMessagesVisible": true,
  "SimMode": "Multirotor",
  "OriginGeopoint": {
    "Latitude": 43.311035,
    "Longitude": 12.944218,
    "Altitude": 324
  },
  "Vehicles": {
    "Copter": {
      "VehicleType": "ArduCopter",
      "UseSerial": false,
      "LocalHostIp": "127.0.0.1",
      "UdpIp": "127.0.0.1",
      "UdpPort": 9003,
      "ControlPort": 9002
    }
  }
}
```

Figura 4

Queste impostazioni serviranno a specificare la tipologia del drone da simulare nonché la posizione di partenza della simulazione, insieme ad altre ovvie configurazioni di comunicazione tra gli strumenti.

3 IMPLEMENTAZIONE

3.1 STRUTTURA DEL PROGRAMMA

A seguito di una breve analisi dei requisiti di progetto, si è scelto di optare per un'implementazione del codice relativo all'osservatore diagnostico al di fuori del codice di simulazione di AirSim.
La scelta è stata fatta in ottica di un'implementazione reale: al fine di ottenere un eventuale drone più leggero di hardware e "scarico" di codice possibile, si è scelto di optare per una soluzione che possa spostare il carico di lavoro relativo all'analisi e al calcolo dell'osservatore diagnostico a terra. La soluzione scelta, quindi, ricalcherà l'implementazione di una aggiuntiva Ground Control Station in cui scaricare i dati di volo del drone ed effettuare le analisi opportune.

In breve, lo script soluzione di progetto si connetterà alla simulazione del drone, scaricherà i dati di volo, li elaborerà visualizzandone lo stato parziale e mostrerà al termine del processo i dati raccolti ed analizzati in totale indipendenza rispetto al volo del veicolo e al veicolo stesso. Parte dell'elaborazione verrà fatta quindi in modalità *online*, in modo che la rilevazione del guasto possa essere il più vicina possibile al guasto stesso e non al termine del percorso del drone.

Per necessità di sviluppo, come soluzione parziale, sono stati implementati due script aggiuntivi volti ad eseguire l'analisi dell'osservatore in modalità *offline*. Il primo dei due si occuperà di effettuare la raccolta dati attraverso la connessione col drone a mo' di log crawler e salvarli su di un dataset in forma testuale.
Il secondo script, raccolto il dataset, elaborerà gli stati e le uscite dell'osservatore diagnostico a partire dalle entry dei log precedentemente generati. In questo modo si è altamente favorita la portabilità di sviluppo, rendendo possibile l'analisi implementativa e la correzione dei bug del codice dell'osservatore senza dover necessariamente lanciare di nuovo la simulazione e rimettere in campo un volo di prova.

La seconda soluzione viene consegnata insieme al codice finale di progetto, in allegato a questo documento.

3.2 IMPLEMENTAZIONE CONNESSIONE E RACCOLTA DATI

Per ottenere i dati di volo del drone è necessario creare e interrogare il controllore attraverso un canale di comunicazione. A questo fine si è scelto di utilizzare la libreria PyMavLink, attraverso la quale è possibile sfruttare il protocollo di comunicazione MavLink e decodificare i messaggi di stato dell'apparecchio.

Nel codice Python che andrà a formare lo script sarà quindi possibile utilizzare la classe pymavlink.mavutil.MavlinkConnection per definire una connessione su di un protocollo di connessione, che nel caso del progetto in discussione è il protocollo UDP sulla porta 14501.

L'oggetto connessione così creato consente l'utilizzo del metodo recv_match() che consente di attendere l'arrivo sul canale di comunicazione di un messaggio che corrisponde ad un certo pattern da cui poi estrarre informazione. Nel nostro caso sono stati utilizzati i pattern "SERVO_OUTPUT_RAW", "LOCAL_POSITION_NED", "ATTITUDE", "AHRS2" per ottenere rispettivamente i parametri: rcou dei 4 motori da cui ottenere attraverso opportune trasformazioni gli ingressi uf, up, uq, ur relativi alla spinta del drone in direzione ascendente e nelle direzioni relative agli angoli di attitude; la posizione nello spazio e le velocità di spostamento lungo gli assi X, Y e Z; le velocità di rotazione in roll, pitch e yaw; gli angoli di roll pitch e yaw.

I parametri di volo ottenuti in questo modo verranno poi depositati su strutture dati apposite che ne consentano l'accesso successivo e l'elaborazione dei dati conseguente.

3.3 IMPLEMENTAZIONE OSSERVATORE

Per poter implementare l'osservatore è necessario calcolare le matrici A0, B0, C0, D0 e H a partire dalle matrici A, B, C, D a disposizione. Queste ultime si ipotizzano conosciute e rappresentano le matrici che caratterizzano il sistema reale.

Nel caso particolare del quadricottero in esame si ha a disposizione il seguente modello matematico:

$$
\begin{aligned}
m\ddot{x}_F &= [\cos(\varphi)\sin(\theta)\cos(\psi) + \sin(\varphi)\sin(\psi)]u_f - k_t\dot{x}_F \\
m\ddot{y}_F &= [\cos(\varphi)\sin(\theta)\cos(\psi) - \sin(\varphi)\sin(\psi)]u_f - k_t\dot{y}_F \\
m\ddot{z}_F &= \cos(\varphi)\cos(\theta)u_f - mg - k_t\dot{z}_F \\
I_x\dot{p} &= -k_r p - qr(I_z - I_y) + u_p \\
I_y\dot{p} &= -k_r q - pr(I_x - I_z) + u_q \\
I_z\dot{r} &= -k_r r - pq(I_y - I_x) + u_r \\
\dot{\varphi} &= p + q\sin(\varphi)\tan(\theta) + r\cos(\varphi)\tan(\theta) \\
\dot{\theta} &= q\cos(\varphi) - r\sin(\varphi) \\
\dot{\psi} &= [q\sin(\varphi) + r\cos(\varphi)]/\cos(\theta),
\end{aligned}
$$

Figura 5

In questo modello uf, up, uq e ur rappresentano gli input del sistema, mentre xF,yF,zF, ẋF, ẏF, żF,p,q,r,φ,θ,ψ rappresentano gli output.

Si possono così definire le matrici A, B, C, D:

```
Ix = 0.008154;
Iy = 0.008154;
Iz = 0.017458;
Kt = 0.001;
Kr = 0.001;
m = 1.22;
g = 9.81;
F = 90;

A = [
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 -Kt/m 0 0 0 0 0 g*sin(F) g*cos(F) 0
0 0 0 0 -Kt/m 0 0 0 0 -g*cos(F) g*sin(F) 0
0 0 0 0 0 -Kt/m 0 0 0 0 0 0
0 0 0 0 0 0 -Kr/Ix 0 0 0 0 0
0 0 0 0 0 0 0 -Kr/Iy 0 0 0 0
0 0 0 0 0 0 0 0 -Kr/Iz 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0
];
B = [
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
1/m 0 0 0
0 1/Ix 0 0
0 0 1/Iy 0
0 0 0 1/Iz
0 0 0 0
0 0 0 0
0 0 0 0
];

C = eye(12);

D = 0;
```

Figura 6

L' operazione di calcolo delle matrici A0, B0, C0, D0 e H è stata svolta grazie all'utilizzo del software MATLAB R2021b e la schermata riportata in precedenza è frutto delle definizioni delle matrici di partenza nel software appena citato.

Per il calcolo della matrice H è possibile utilizzare la funzione *place* che prende in ingresso le matrici A e C ed i poli desiderati. Per ottenere un risultato soddisfacente sono stati scelti dodici poli compresi tra -4 e -5 e differenti tra loro in modalità pseudocasuale.

Le altre matrici dell'osservatore sono state calcolate nel modo seguente:

$$A0 = A-H*C;$$
$$B0 = [B,H];$$
$$C0 = C;$$
$$D0 = D;$$

Le matrici ottenute sono poi ricopiate nel codice Python dello script.

A questo punto, ottenute le matrici A0, B0, C0, D0 che caratterizzano il sistema Lineare Tempo Invariante osservatore in rappresentazione in spazio di stato, va lanciata una funzione risolutrice che sia in grado di ottenere l'uscita nel tempo del sistema a partire dagli ingressi forniti. Per implementare una soluzione che sia in grado di effettuare il calcolo dell'uscita in modalità online, quindi il più possibile contemporaneamente all'effettivo volo del drone, si è scelto di adoperare la seguente metodologia:

Allo stato temporale *t* vengono catturati gli ingressi forniti al sistema al tempo *t* e al tempo *t-1*. Si genera quindi un vettore che contenga i due vettori di ingresso al tempo *t-1* e *t* rispettivamente. Questo vettore verrà dato in pasto alla funzione della libreria Python Control control.matlab.lsim che ricalca fedelmente il comportamento della stessa lsim all'interno di MatLab. Alla funzione nel codice Python verrà detto di effettuare l'elaborazione su di un tempo caratterizzato da un LinearSpace a 2 valori, *t-1* e *t* prendendo come stato iniziale dell'elaborazione X0 lo stato ottenuto dall'elaborazione precedente effettuata tra i tempi *t-2* e *t-1*. Ciclando questo blocco per tutto lo spazio lineare definito dagli istanti temporali di elaborazione si ottiene una implementazione iterativa in grado di risolvere il problema.

Affinché fosse possibile verificare la correttezza di questa scomposizione iterativa, sono stati implementati due script aggiuntivi basati su matrici A0, B0, C0, D0 di prova, più semplici di quelle riportate precedentemente: il primo script avrebbe implementato la funzione di elaborazione come appena descritta, il secondo invece avrebbe effettuato un'elaborazione gemella off-line e quindi a partire da dati interamente precedentemente caricati, il tutto su di un LinearSpace che rappresenta interamente il tempo di elaborazione. Il confronto tra l'output fornito dai due script conferma la corretta metodologia applicata e viene riportato nelle due immagini successive.

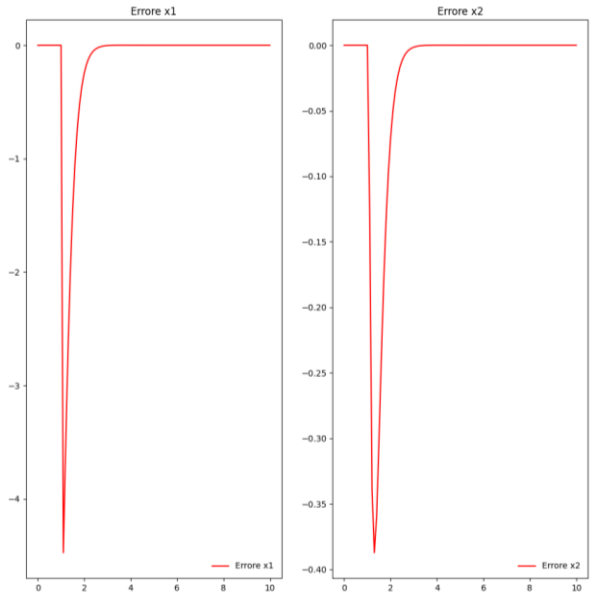

Figura 7 Risposta al gradino (codice online)

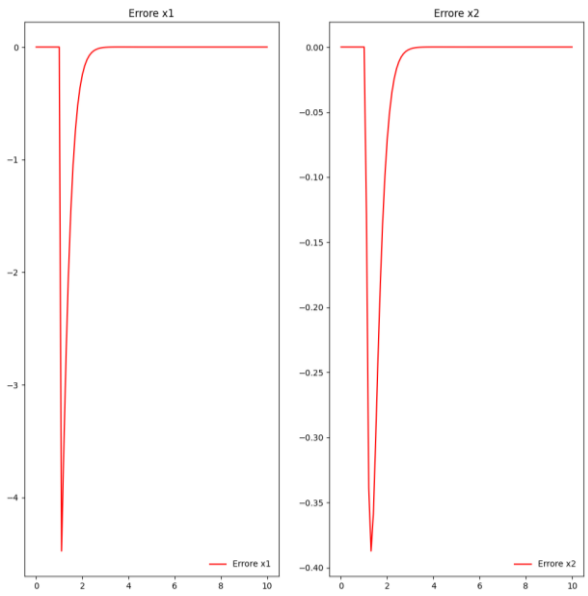

Figura 8 Risposta al gradino (codice offline)

I risultati forniti dal processo di elaborazione vengono poi utilizzati per ricavare le singole differenze tra i parametri della simulazione e quelli del modello; alla sottrazione tra i due viene dato il nome di residui. La rilevazione del guasto utilizzerà quindi l'aumento o la diminuzione dei parametri residuali per constatare un ipotetico malfunzionamento.

3.4 IMPLEMENTAZIONE ELABORAZIONE DATI

Per poter terminare la fase di osservazione e stampare i grafici di interesse è necessario introdurre una soluzione che permetta di uscire dal ciclo "while True" che si occupa di acquisire dati dal volo, stimare lo stato del sistema tramite osservatore e generare i residui. Si è optato per un semplice try - except KeyboardInterrupt che permette l'uscita dal ciclo alla pressione da tastiera dei tasti ctrl+c.

Si ha dunque a disposizione un vettore *graph_vector* appartenente alla classe GraphVector() e contenente tutti i dati di volo da utilizzare per creare e stampare i grafici. Quindi, questo viene passato alla funzione *printGraphs* che provvede a stampare tre finestre indicizzate.

Per la creazione di queste finestre sono state utilizzate le librerie matplotlib.pyplot e mpl_toolkits.mplot3d.

La prima finestra è costituita da 5 subplots contenenti informazioni generiche sul volo del drone:

1. Evoluzione della posizione del drone (X,Y,Z) nel tempo
2. Evoluzione della velocità del drone (Vx,Vy,Vz) nel tempo
3. Evoluzione della velocità angolare del drone (P,Q,R) nel tempo
4. Evoluzione degli angoli del drone (Phi,Theta,Psi) nel tempo
5. Evoluzione degli ingressi del drone (UF,UP,UQ,UR) nel tempo

La seconda finestra traccia la traiettoria del drone nello spazio 3D durante il volo.

La terza e ultima finestra è costituita da 13 subplots contenenti i residui ottenuti:

1. Evoluzione del residuo della posizione X nel tempo
2. Evoluzione del residuo della posizione Y nel tempo
3. Evoluzione del residuo della posizione Z nel tempo
4. Evoluzione del residuo della velocità lungo X nel tempo
5. Evoluzione del residuo della velocità lungo Y nel tempo
6. Evoluzione del residuo della velocità lungo Z nel tempo

7. Evoluzione del residuo della velocità angolare P nel tempo
8. Evoluzione del residuo della velocità angolare Q nel tempo
9. Evoluzione del residuo della velocità angolare R nel tempo
10. Evoluzione del residuo dell' angolo Phi nel tempo
11. Evoluzione del residuo dell' angolo Theta nel tempo
12. Evoluzione del residuo dell' angolo Psi nel tempo
13. Evoluzione del residuo della velocità orizzontale ottenuto come radice quadrata della somma del residuo della velocità lungo X al quadrato e del residuo della velocità lungo Y al quadrato

Grazie a questi grafici è possibile verificare la coerenza dei residui ottenuti con il volo intrapreso, andando ad individuare in modo semplice eventuali guasti del drone.

3.5 GUIDA AL SOFTWARE

Entrambe le soluzioni fornite assieme a questa relazione, testate solamente in ambiente GNU/Linux, richiedono che sia eseguito e lanciato il simulatore AirSim attraverso un Unreal Environment e Unreal Engine stesso e che ArduPilotSITL sia avviato correttamente.

Il comando utilizzato per l'avvio di UnrealEngine e AirSim installato all'interno della cartella /home è:

./UnrealEngine/Engine/Binaries/Linux/UE4Editor

Successivamente al comando è necessario confermare la scelta di un Unreal Environment in cui definire la simulazione. Per gli scopi di questo progetto è sempre stato utilizzato l'environment predefinito Blocks.

All'avvio dell'enviroment è necessario premere il pulsante Play nella barra in alto della finestra di Unreal Engine per predisporre l'attivazione dello spazio di simulazione.

L'avvio del drone all'interno della simulazione viene effettuato a partire dalla cartella *ardupilot/ArduCopter/* con il comando:

../Tools/autotest/sim_vehicle.py -v ArduCopter -f airsim-copter --console –map

A questo punto, la simulazione di drone predefinita da parte di ArduPilotSITL effettua un campionamento dei dati di volo a 4hz. Per ottenere una elaborazione più efficace da parte dell'osservatore, attraverso le impostazioni di configurazione degli script MAVProxy, lanciati dall'esecuzione di ArduPilotSITL, si può modificare la frequenza di campionamento del canale di comunicazione principale aumentandola a circa 100hz, con un cap massimo di funzionamento ottimale testato definito in 200hz.

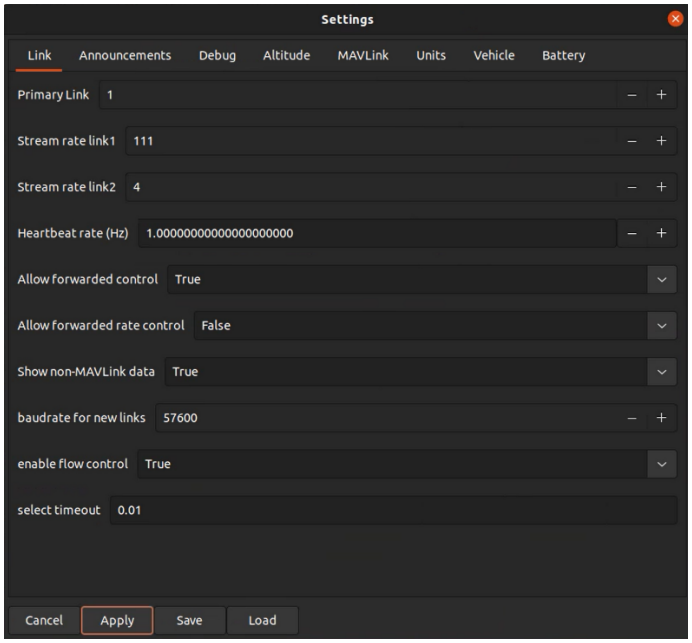

Figura 9

Da qui, utilizzando la finestra contenente la mappa di volo è possibile inizializzare e definire un percorso da far eseguire al drone attraverso l'utilizzo della funzione Mission/Draw accessibile al clic destro.
La funzione grossolanamente definita può quindi essere modificata e raffinata attraverso Mission/Editor.

Per attivare il drone ed armarne le pale, dal terminale con cui è stato lanciato ArduPilotSITL si può eseguire il comando:

arm throttle

Dopo aver atteso che il drone si sia attivato e che siano state simulate le entry di connessione al segnale GPS, per alzare in volo il drone a 30 metri da terra possono essere eseguiti i comandi:

mode guided

takeoff 30

Appena il drone è giunto all'altezza desiderata, per eseguire il piano di volo precedentemente definito può essere lanciato il comando:

long MISSION_START

In fasi successive di testing, per valutare l'efficacia dell'osservatore diagnostico, sono stati inseriti diversi guasti.
Per definire guasti relativi agli attuatori delle pale, va prima selezionato il motore a cui attribuire il guasto con il comando:

param set SIM_ENGINE_FAIL [id_motore]

dove [id_motore] va sostituito con i possibili valori 0, 1, 2, 3 che definiscono i motori in base alla configurazione del drone predefinito. Nella configurazione di base utilizzata nello svolgersi del progetto, guardando il drone dall'alto verso il basso, ortogonalmente allo stesso, l'identificatore 0 rappresenta il motore in alto a destra, il motore 1 è quello in basso a sinistra, il motore 2 in alto a sinistra ed il 3 in basso a destra. Dopodiché al motore scelto può essere modificato un moltiplicatore che definisce la forza di spinta dell'attuatore con il comando:

param set SIM_ENGINE_MUL [valore]

dove al posto di [valore] può essere inserito un valore compreso tra 0 e 1 per diminuire la forza di spinta o maggiore di 1 per aumentarla. Nei test effettuati si è notato che per valori compresi tra 1 e 0.80 il controllore del drone simulato riesce a compensare il guasto in maniera ottimale. I primi comportamenti non attesi si ottengono per valori di guasto al singolo motore che vanno tra 0.80 e 0.65. Su valori intorno a 0.60 la spinta fornita dal motore non è più sufficiente a mantenere in volo il drone, che inizia a perdere quota.

Per l'inserimento di guasti ai sensori e testing del comportamento dell'osservatore a malfunzionamenti di questo tipo, per la modifica dei parametri del sensore di posizione GPS lungo l'asse Y è stato utilizzato il comando:

param set SIM_GPS_GLITCH_Y [valore]

Mentre, per la modifica dei parametri di funzionamento del sensore acceleratore sempre lungo l'asse Y è stato inserito il guasto attraverso il comando:

param set SIM_ACC1_BIAS_Y [valore]

Terminato il piano di volo, o terminata la fase rilevante dell'analisi di comportamento dell'osservatore diagnostico in risposta ai guasti, per far atterrare il drone è necessario inserire il comando:

mode land

ed il drone avvierà la procedura di landing automatico.

Noti i comandi necessari a manovrare il drone, l'avvio degli script relativi all'osservatore implementato come soluzione a questo progetto richiedono che sia avviato un ulteriore terminale e siano dati i seguenti comandi:

python PATH/TO/SCRIPT/observer.py (per l'osservatore online)

oppure:

python PATH/TO/SCRIPT/crawler.py
(per l'implementazione di raccolta dati offline, da terminare con ctrl-c)

python PATH/TO/SCRIPT/observer_mock.py (per l'analisi offline dei dati appena raccolti)

Entrambe le soluzioni, al termine della raccolta dei dati, mostreranno i grafici dei dati raccolti secondo quanto definito nei precedenti paragrafi.

4 TEST GUASTI

4.1 VOLO DI RIFERIMENTO

Per ottenere raffronti da utilizzare nella comprensione dei dati ricavati dai voli con guasto, si è scelto innanzitutto di riportare un primo volo in cui i malfunzionamenti non sono presenti.

Per questo volo, e per i successivi, si è scelto di considerare una traiettoria pianificata di movimento sull'asse X e, a causa del processo di linearizzazione del modello del drone che porta ad una maggiore efficacia dell'osservatore in condizioni vicine a quelle di hover, si è scelta una velocità relativamente blanda di 0,20m/s. Il drone quindi si muoverà a 100m di altezza in direzione frontale per un minuto.

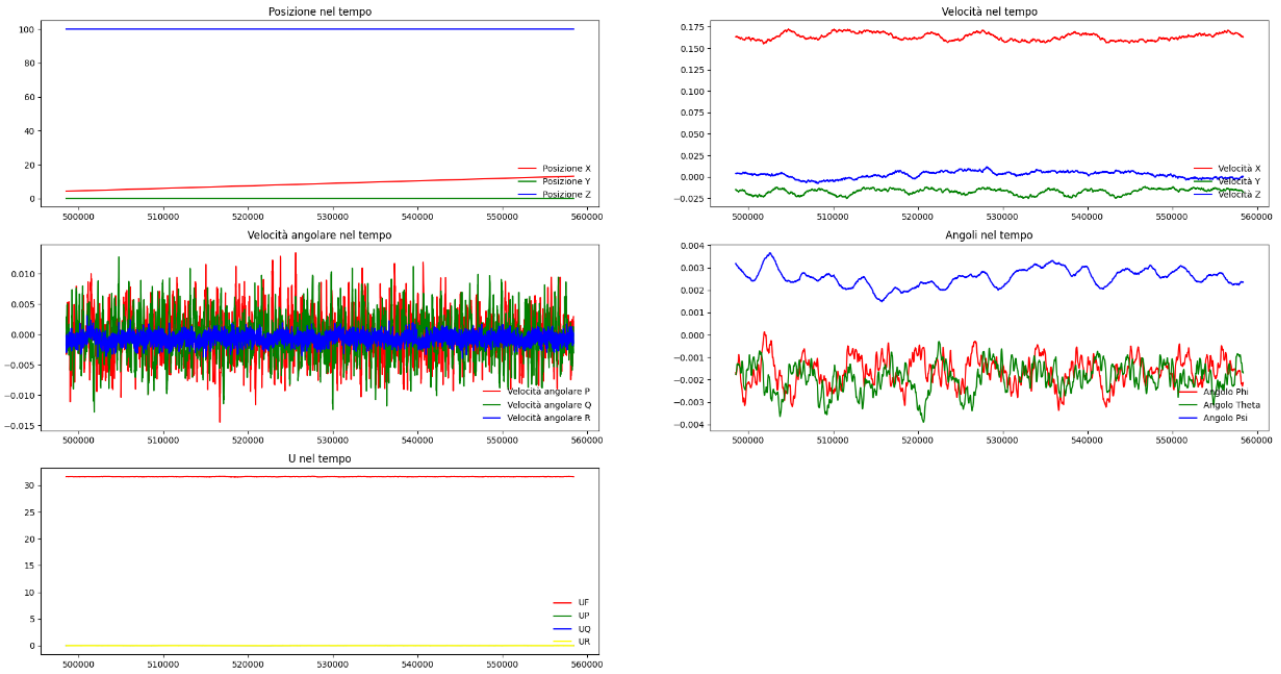

Figura 10

Come si può notare dai grafici riportati, la lettura dei parametri di volo avviene correttamente.

Per un volo relativamente semplice come quello utilizzato per riferimento è abbastanza evidente la presenza di rumore sui segnali.

Si ricorda che, soprattutto nel caso di parametri che poco mutano nel tempo, i metodi utilizzati per la stampa dei grafici adottano funzioni che modificano automaticamente lo scaling degli assi.

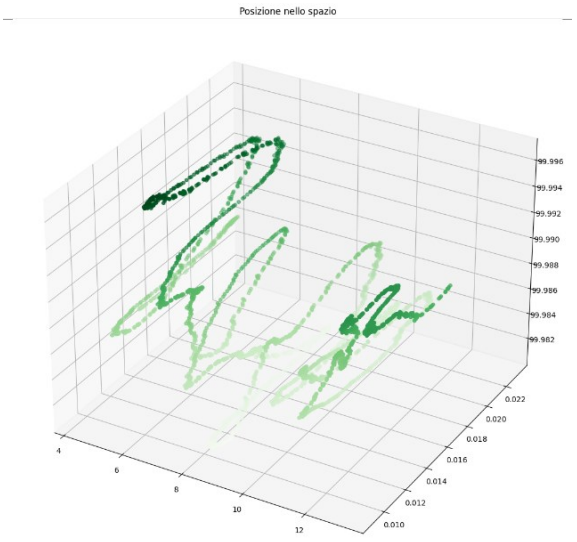

Figura 11

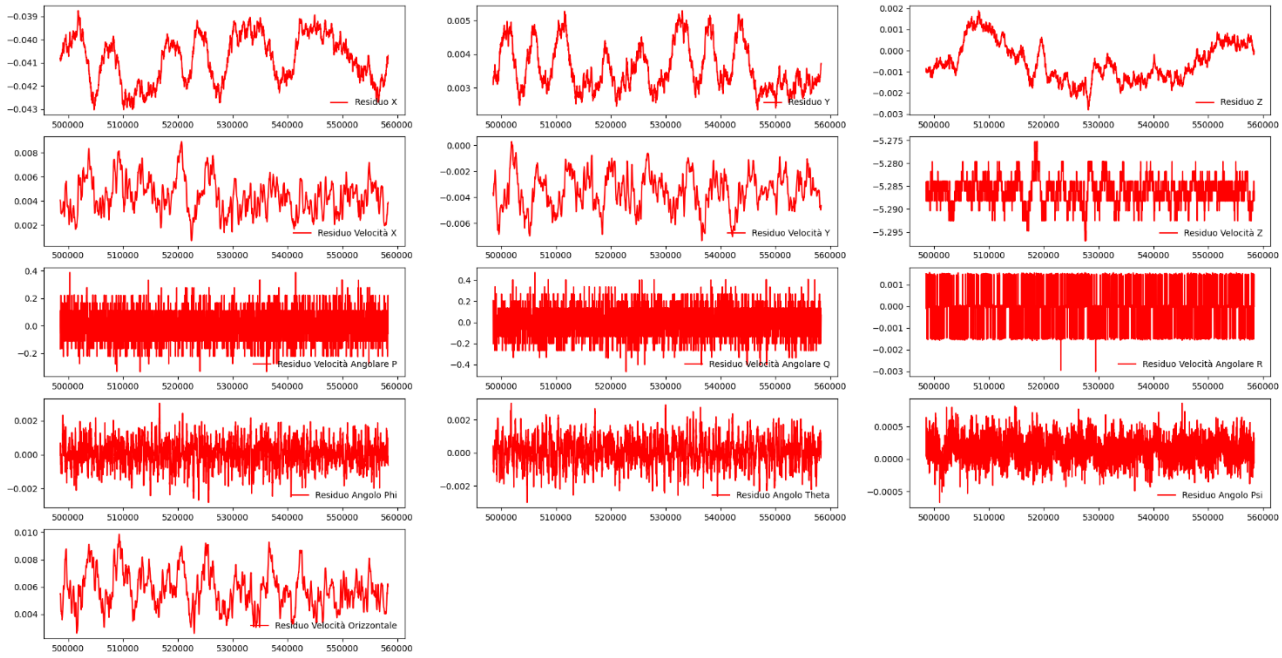

Figura 12

Si può notare come i residui calcolati in un volo in assenza di guasti varino per tutte le misure tra valori relativamente piccoli ed accettabili a causa dell'ovvio rumore presente nei parametri della simulazione.

4.2 GUASTO AL MOTORE 1

Il primo volo con guasto ha il seguente piano di volo: ci si muove in linea retta frontalmente a 0.20m/s per venti secondi, viene inserito un guasto al motore 1 (frontale di destra) che ne riduca la capacità di spinta al 70%, valore significativo in modo che il controllore non sia in grado di compensare efficacemente, e dopo 20 secondi di volo con guasto esso viene rimosso per tornare ad un volo in condizioni standard.

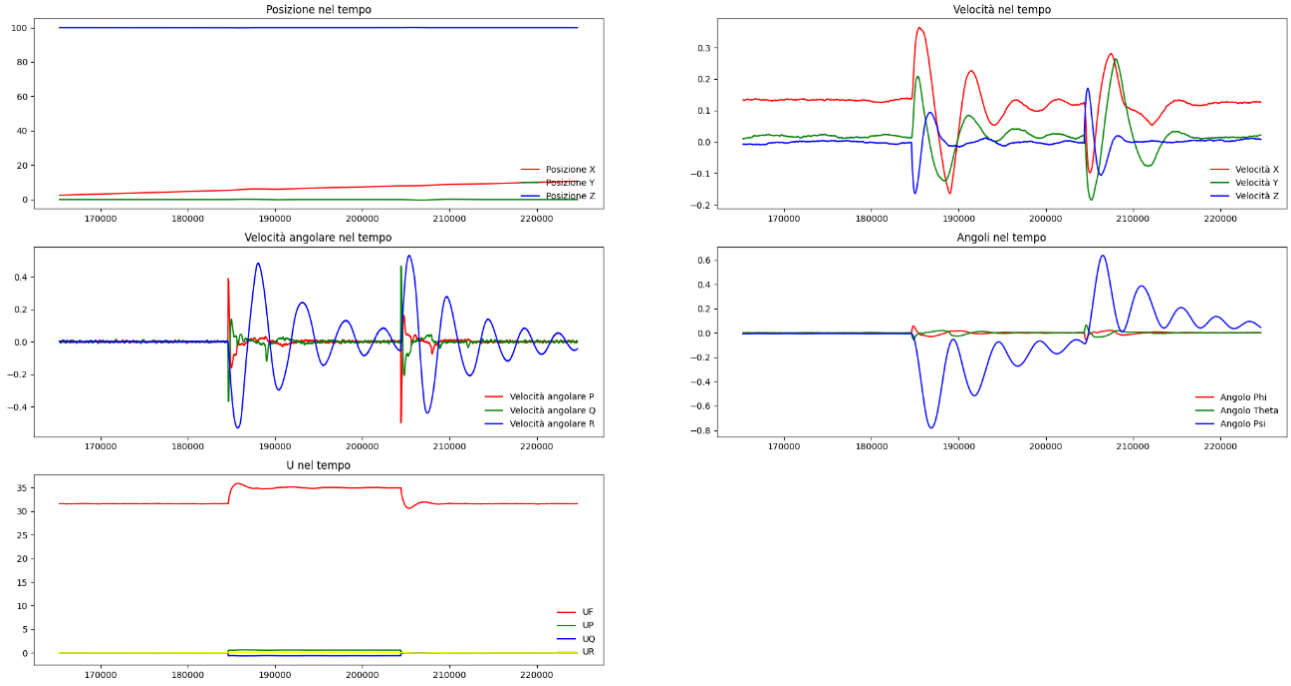

Figura 13

In questo caso, si può notare come a differenza del volo di riferimento, sia chiaro l'inserimento del guasto sia nelle variabili di ingresso U, sia nel comportamento relativo all'attitude del drone e quindi nella sua posizione nello spazio.

Nel grafico 3D di dimensione nello spazio si nota in maniera ottimale come sia l'inserimento del guasto che la sua rimozione giochino in una evidente modifica della traiettoria di volo e addirittura in una prima perdita di altitudine nel momento dell'inserimento e poi di un sovraguadagno nei momenti successivi alla rimozione del comportamento anomalo.

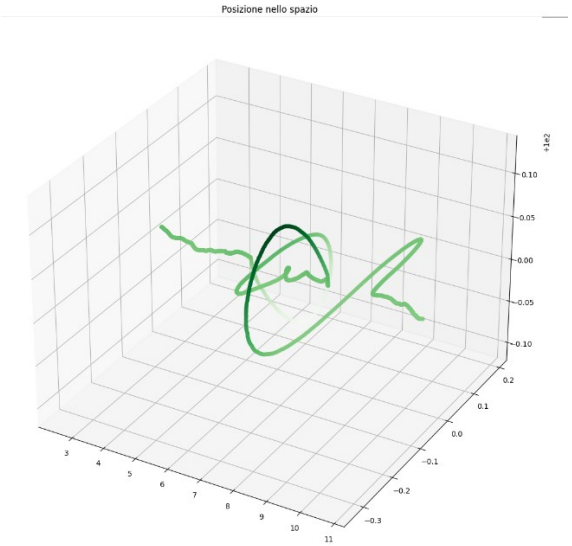

Figura 14

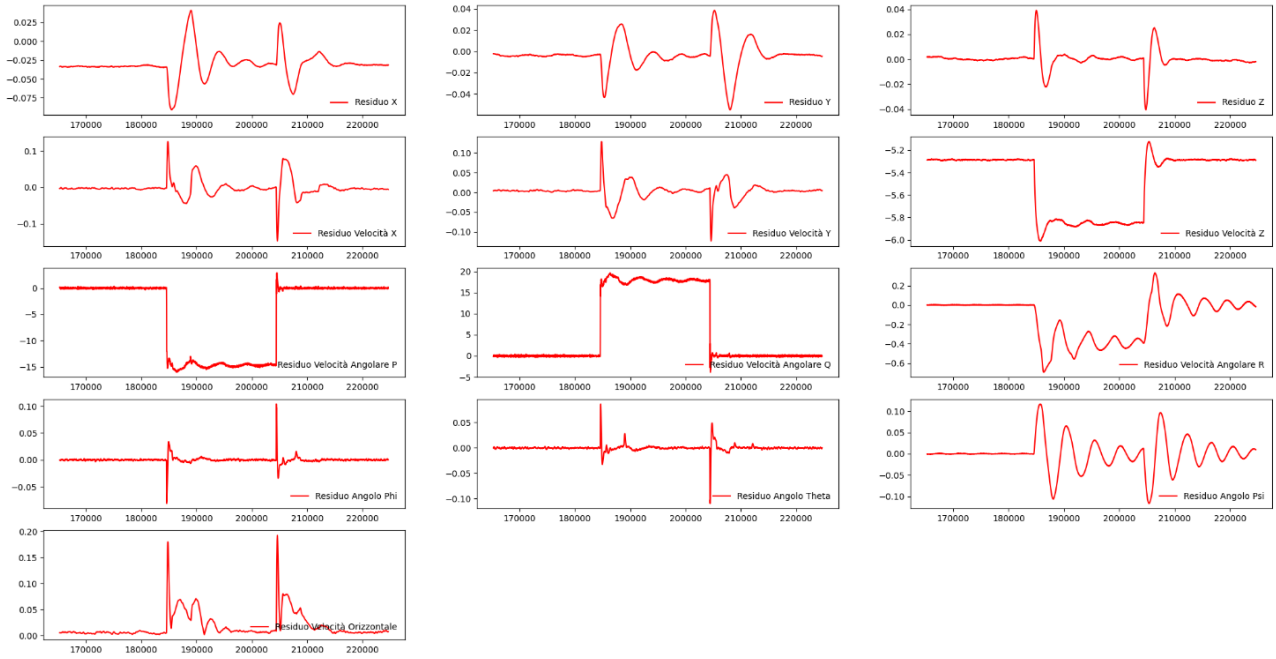

Figura 15

A confermare l'efficacia della soluzione adottata, tutti i grafici relativi ai residui riportano una brusca variazione di comportamento sia all'inserimento che alla rimozione del comportamento anomalo del motore, alcuni di questi mantenendo elevata per tutti i 20 secondi di guasto una evidente differenza tra quanto rilevato e quanto atteso dal modello di processo.

4.3 GUASTO AL SENSORE GPS

Il secondo volo con malfunzionamento segue il piano dei voli precedenti, con l'inserimento nel periodo centrale di un guasto alla coordinata Y del sensore di GPS di 0.0001 latitudine, circa 11.1m nella lettura della posizione.

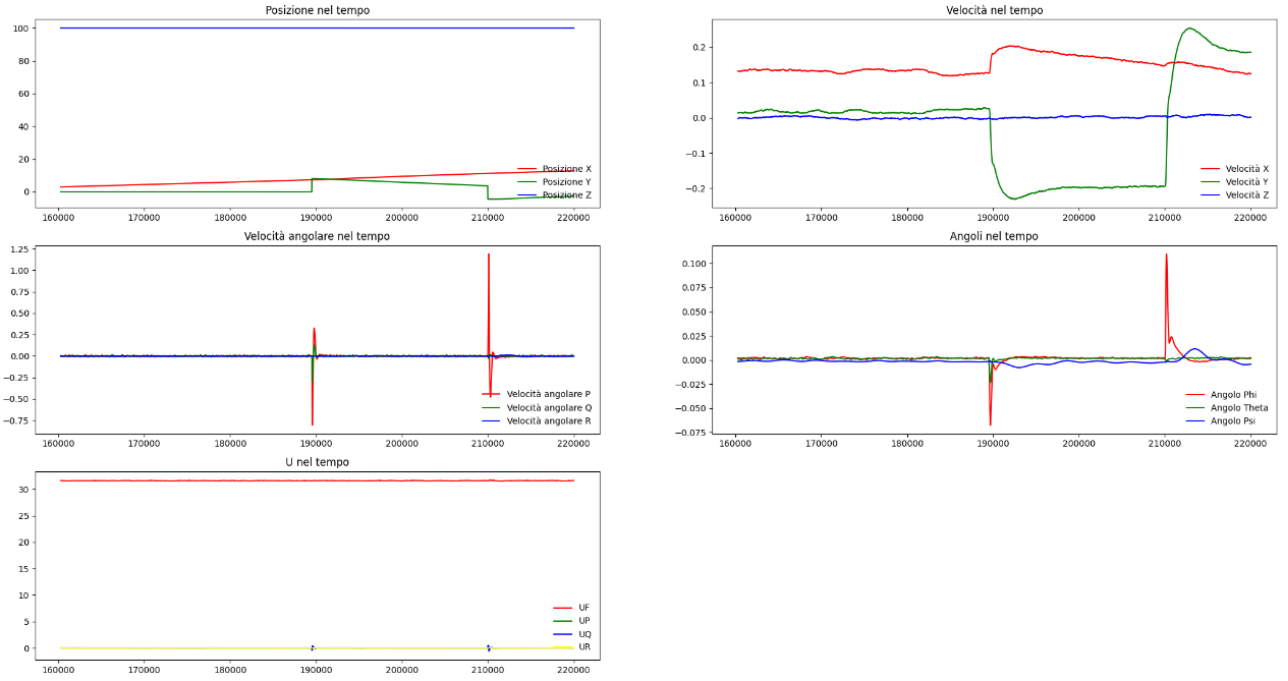

Figura 16

Anche in questo caso si mostrano evidenti l'inserimento e la rimozione del malfunzionamento al sensore. Si nota anche come effettivamente, inserito il guasto, il drone provi a correggere il suo misplacement muovendosi lentamente nel tentativo di riportare il valore in lettura sull'asse Y a quello previsto.

Avendo inserito un guasto relativo alla posizione del drone nello spazio ci si aspettava che questa rispecchiasse un andamento a 3 periodi distinti e separati esattamente come poi è stato ottenuto nel grafico qui riportato.

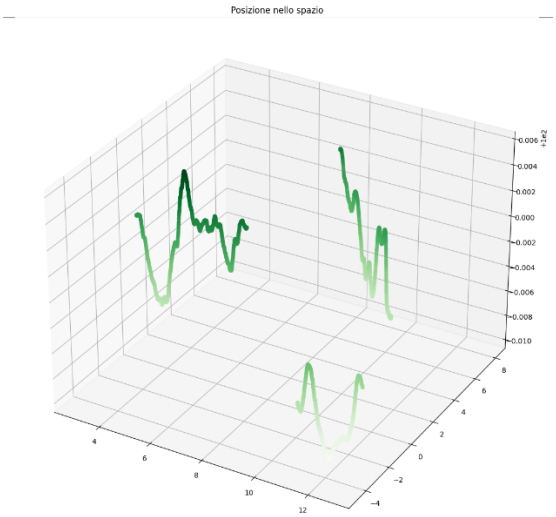

Figura 17

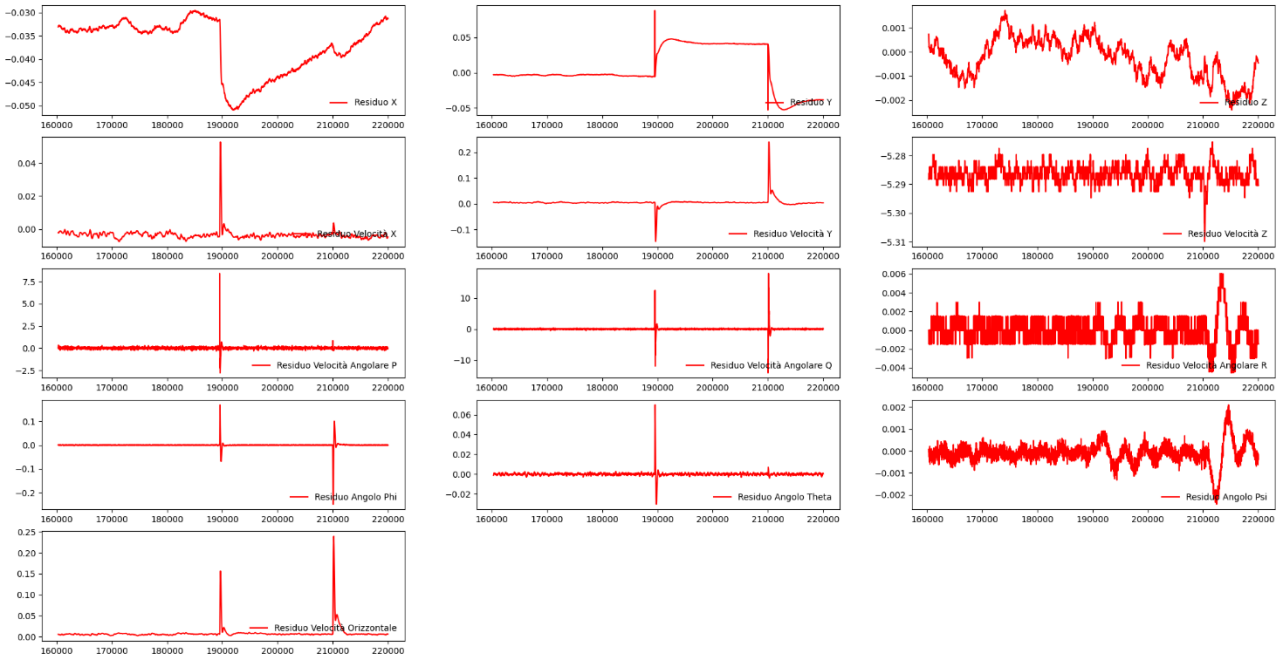

Figura 18

Anche in questo caso, a dimostrare la discreta efficacia della soluzione adottata, si hanno alcuni comportamenti bruschi riscontrati su più residui riguardo e all'inserimento del comportamento inatteso.

4.4 GUASTO AL SENSORE ACCELEROMETRO

L'ultimo volo con guasto è stato ottenuto inserendo nei 20 secondi centrali del minuto di volo di un guasto relativo alla componente sull'asse Y dell'accelerometro di bordo del drone, quindi anche in questo caso inserendo un fittizio malfunzionamento ad un sensore.

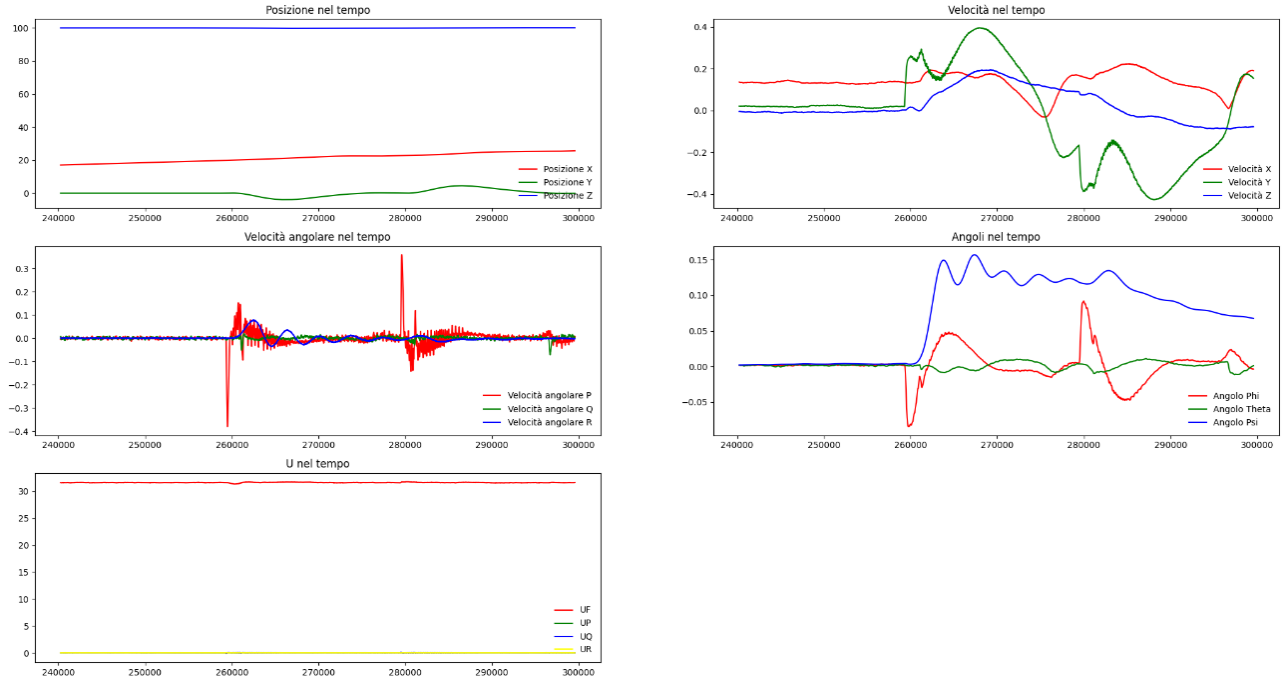

Figura 19

Malfunzionamento più che evidente anche questa volta dai grafici basati sui dati di volo qui riportati.

Come visibile dal grafico 3D della posizione nello spazio, un guasto all'accelerometro comporta che il controllore del drone tenda a compensare muovendosi nella direzione sull'asse che subisce il guasto risultando nel comportamento ad "S" riportato.

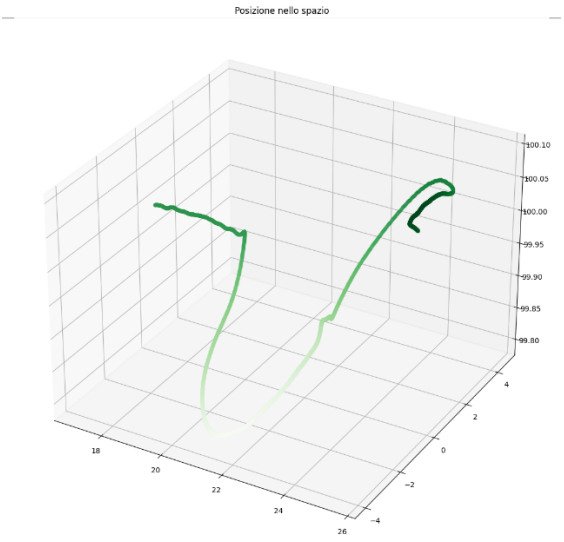

Figura 20

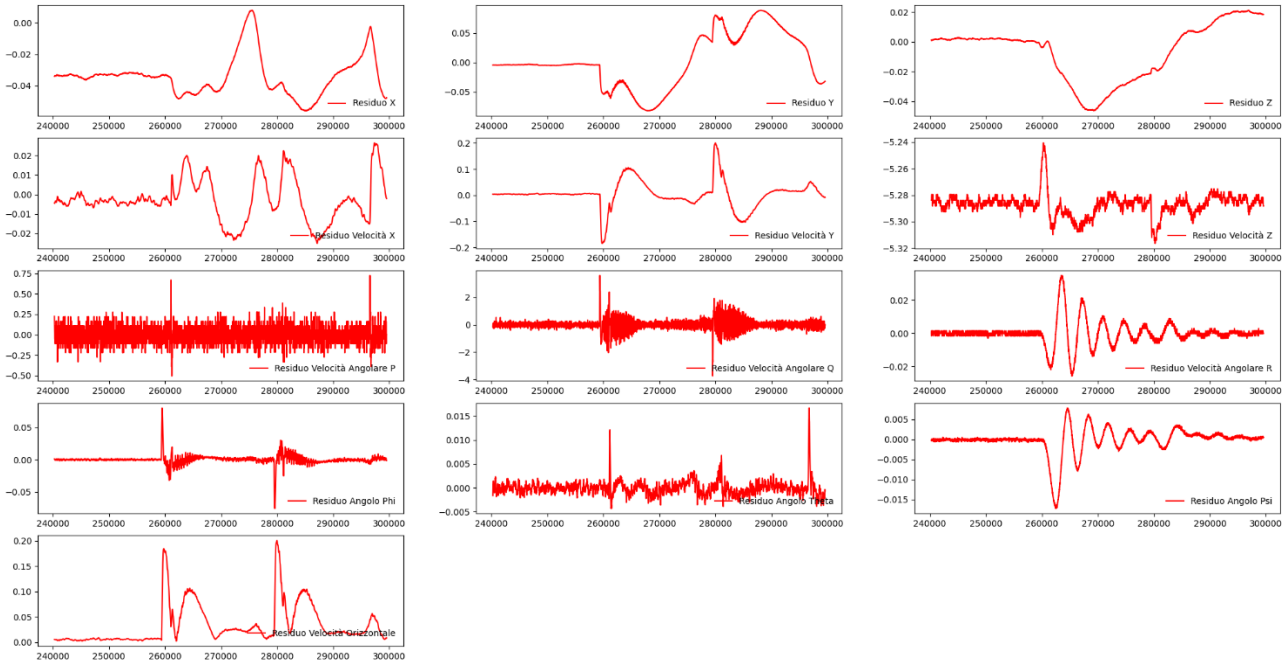

Figura 21

Anche nel caso del guasto all'accelerometro la soluzione adottata riesce a rispondere correttamente con la visualizzazione dei comportamenti inattesi sia all'inserimento che alla rimozione del guasto in molti dei parametri di riferimento.

5 CONCLUSIONI

Si conclude quindi che la soluzione adottata, con tutti i limiti del caso di studio, riesca ad effettuare una vera e propria rilevazione dei guasti inseriti.

Il comportamento dei residui in tutti i voli di prova con guasto si differenzia notevolmente dal volo di riferimento, sebbene per ogni residuo possa essere considerata una discreta componente relativa al rumore sui segnali.

In voli di prova non riportati sono stati riscontrati accettabili risultati di rilevamento anche in condizioni lontane dal volo in hover, sebbene il rumore e i disturbi introdotti nel sistema osservatore da condizioni così lontane da quelle studiate ed ottimizzate per il suo funzionamento, rendano più aleatorio il comportamento dei residui e quindi più difficile l'individuazione a partire dall'analisi visiva del segnale.

Per questi motivi, l'implementazione della soluzione riportata risulterebbe discretamente accettabile anche come prima base di partenza di rilevamento guasti per un drone reale, potendo superare le ovvie difficoltà di riadattamento del sistema all'esterno di una simulazione virtuale.

6 INDICE FIGURE