UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Studio e sviluppo di tecniche di controllo per l'inseguimento di percorsi da parte di droni

Study and development of control techniques for tracking paths by drones

Tesi di Laurea di:

Mariotti Manuel

Relatore:

Prof. Gianluca Ippoliti

A.A. 2022 / 2023

SOMMARIO

L’utilizzo dei droni nel mondo odierno è in continua crescita, con applicazioni che vanno dalle attività di pronto intervento ad attività di logistica, dalle applicazioni nel settore agricolo alle attività di intrattenimento per citarne alcune.

Il fattore comune di tali applicazioni è il controllo della traiettoria che essi devono seguire e nelle pagine successive si tratterà “come nel mio piccolo”, sia riuscito anche io in questo intento.

Indice generale

Indice delle figure

Indice delle tabelle

Introduzione

Il controllo del drone per l'inseguimento di una traiettoria è stato effettuato mediante il software Matlab&Simulink; in particolare si è utilizzato un package di supporto per i droni dell'azienda costruttrice *Parrot,* il quale offriva diversi progetti "base" da cui partire per lo sviluppo.

La verifica del corretto inseguimento della traiettoria (percorso rosso su fondo nero) è stata effettuata mediante la simulazione in un ambiente virtuale tridimensionale offerto proprio dal package in questione, il quale offriva le rappresentazioni del drone e del percorso citato.

Si approfondirà il package ed il progetto utilizzato nella sezione dedicata.

I Droni

Generalità e breve excursus storico

I droni sono definiti “aeromobili a pilotaggio remoto”, cioè senza la presenza di un pilota a bordo ma comandati grazie ad un computer di controllo interno o tramite radiocomando da un operatore posto a terra o in altre posizioni.

Questa caratteristica consente al drone di essere impiegato nelle applicazioni più pericolose garantendo così sicurezza per la vita umana.

Nascono come velivoli per usi militari nella Prima guerra mondiale con i modelli “Aerial Terget” (comandato tramite radiofrequenze) e lo “Sperry Aerial Torped” (comandato tramite dei giroscopi posti all’interno dello stesso), per poi diffondersi anche negli ambiti civili a partire dagli anni duemila.

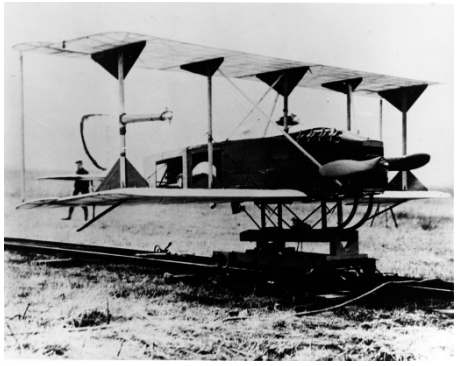


Figura 1 Sperry aerial Torped

Figura 2 Aerial Target

Oggi sono impiegati in svariate applicazioni come la consegna di pacchi e materiali in zone difficili da raggiungere, nella ricerca di feriti a seguito di disastri naturali e il pronto soccorso degli stessi, nell’aerofotogrammetria, nelle riprese cinematografiche o nell’agricoltura con il trattamento delle coltivazioni per citarne alcune; inoltre non mancano di certo droni con impieghi meno professionali con scopi ludico ricreativi.

Figura 3 Da sinistra verso destra, un drone per il cinema, per la consegna di medicinali e ad uso hobbistico.

Una curiosità: fino al 1946 il termine “Drone” veniva dalla lingua inglese con il significato di “Fuco”, il maschio dell’ape domestica; da quel momento in poi si è scelto di associarla anche gli aeromobili in questione a causa del loro rumore che ricorda il ronzio di un’ape.

Caratteristiche tecniche dei droni commerciali

In questa sezione si elencheranno le diverse caratteristiche tecniche che caratterizzano i droni commerciali, descritti dalla seguente tabella:

Tabella 1 Specifiche droni commerciali

Specifiche	**Valori**
Dimensione	< 50 cm
Peso	< 900 g
Raggio d’azione	1-15km
Altitudine	< 300 m
Costo	20 – 1700 €

Infatti, il drone utilizzato per il progetto rientra in questa categoria.

Modalità di volo

Una prima classificazione che si può fare riguarda modalità di volo: può essere ad ala fissa o a elica/ala rotante;

I droni ad ala fissa hanno la forma del telaio come quella di un aeroplano, dove si sfruttano le ali per generare portanza e propulsione permettendo così di volare a velocità elevate e coprire distanze maggiori rispetto ai droni ad ala rotante.

Figura 4 Drone ad ala fissa in volo

Gli svantaggi di questi modelli sono quelli del mancato decollo verticale e della mancata possibilità di rimanere in volo stazionario (situazione di *hovering*).

I secondi, quelli ad ala rotante, coprono gli svantaggi dei droni ad ala fissa compiendo manovre simili a quelle di un elicottero grazie alla loro configurazione di eliche.

L'alto grado di manovrabilità li rende in grado di operare in spazi ristretti e in ambienti urbani.

In base al numero di eliche presenti, coincidenti con il numero di motori, essi prendono il nome di *tricotteri*, *quadricotteri*, *esacotteri*, *octacotteri* etc.

Figura 5 Un octacottero su di un piedistallo

Componentistica dei droni ad elica rotante
Di seguito si elencano i componenti fondamentali di ogni drone ad elica rotante.

Entrando più nel dettaglio, troviamo:

- ***Telaio:*** rappresenta la struttura portante del drone e varia a seconda del numero di motori presenti. A partire da esso si sviluppa tutto "il progetto drone" e di conseguenza è quell'elemento che dona robustezza, resistenza e stabilità e che rappresenta la buona parte del suo peso finale. Può essere di diversi materiali, tra

cui legno, plastica, alluminio o carbonio e in base al materiale scelto aumenteranno o meno le qualità sopra citate.

Figura 6 Telaio in fibra di carbonio di un quadricottero

- ***Motori:*** sono la parte di vitale importanza di tutti gli aeromobili, senza i quali non sarebbe possibile l'azione di volo.
  Sono generalmente motori elettrici di tipo "*brushless*", letteralmente "senza spazzole", ossia dei contatti elettrici striscianti sull'albero motore responsabili della tensione di alimentazione impartita al motore (bobine interne).
  La tensione di alimentazione viene applicata grazie alla presenza di un circuito di eccitazione formato da transistor responsabili dell'eccitamento delle bobine interne e di conseguenza al senso di rotazione del motore.
  I *motori brushless* offrono una minore resistenza meccanica, hanno un peso inferiore e caratteristica più importante, non sono interessati da eventuali scintille prodotte dallo sfregamento dei contatti elettrici sull'albero motore come nel caso delle spazzole.

Figura 7 Un motore brushless ed un suo spaccato esemplificativo

- **Eliche:** montate sui motori, svolgono la funzione di generare la *portanza*, ossia la differenza di pressione tra la parte superiore e inferiore delle pale non appena queste iniziano a girare; questa differenza di pressione si converte in una spinta responsabile del sollevamento e del successivo volo del drone.

  Costruttivamente, la generazione di tale spinta viene effettuata realizzando l'elica con la parte superiore curva, mentre quella inferiore piatta o comunque meno curva.

  I fattori che determinano l'intensità di spinta sono la velocità di rotazione dei motori, il diametro e il passo delle eliche; quest'ultimo non è altro che lo spostamento perpendicolare al loro piano prodotto da una rotazione di 360°.

  A parità di portanza, eliche con un passo/diametro minore girano più velocemente di quelle con passo/diametro maggiore.

  I materiali utilizzati per la loro costruzione sono principalmente plastica e fibra di carbonio.

Figura 8 Un'elica montata

- **ESC, regolatori di velocità:** sono schede elettroniche che collegano i *motori brushless* al *flight controller*; si occupano dell'interazione dei due componenti per permettere il controllo del senso di rotazione e della velocità.

Figura 9 Un Esc con i suoi cavi di collegamento

- **Flight controller:** rappresenta il computer interno del drone.
  È una scheda integrata con all'interno una *CPU* (Central Processing Unit) in cui è caricato il software di controllo responsabile del volo.
  Interagisce con i sensori di bordo fornendo così una risposta algoritmica per le varie situazioni da controllare.

Figura 10 USAQ Naze32 Flight Controller

- **Batteria:** è quel componente che fornisce energia elettrica a tutti i dispositivi elettrici ed elettronici presenti nel drone.
  Produce in uscita corrente continua (DC) e per la maggior parte dei casi è di tipologia *LI-PO* (Lithium polymerBatteries) ovvero batterie a Polimeri di Litio.
  Questi tipi di batterie sono composte da celle al litio contenenti elettroliti solidi o gelatinosi e sono prive di componenti in metallo, due caratteristiche per le quali è aumentato il loro impiego a causa di una maggiore efficienza energetica a parità di peso rispetto alle altre tipologie di batterie.
  La tensione in uscita è solitamente specificata come una configurazione di celle, ad esempio 3.7V per una singola cella di Litio.

Figura 11 Batteria LiPo da 4200 mAh (quantità di energia immagazzinabile)

- **Sensori:** Sono dispositivi per la misura delle grandezze fisiche che regolano l'azione di volo. Sono montati sul telaio e a ogni specifico lasso temporale, forniscono informazioni al *flight controller* per le azioni di controllo.
  I più comuni che si trovano a bordo sono: sensore ad ultrasuoni, sensore di pressione, sensore per campo magnetico, giroscopio e IMU (Unità di misura inerziale).

Oltre a questi componenti principali, di modello in modello possiamo anche trovare videocamere, termocamere, GPS, sensori di gas etc.

Disposizione motori quadricotteri

La parte centrale di un quadricottero è chiamata *body frame* e a questa sono collegati quattro bracci alle cui estremità si trovano i motori con le relative eliche.

Considerando il sistema di riferimento solidale al drone (B.F.F.R.- Body fixed Frame of Reference) in cui l'asse x è orientato verso la parte anteriore del drone, l'asse y verso la parte laterale destra e quello z verso la parte inferiore, si definiscono due tipologie di configurazioni:

- **Configurazione a più o '+'**: i motori sono allineati con gli assi x e y del sistema di riferimento del drone.
  Viene mosrato in figura 12 per maggiore chiarezza.

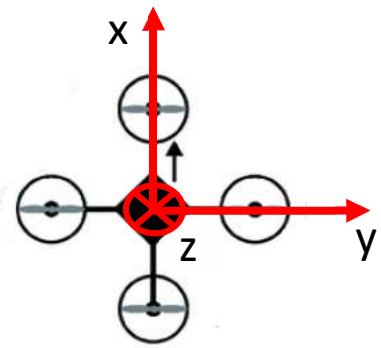


Figura 12 Configurazione a '+'

- **Configurazione a croce:** In questo caso i motori sono disallineati dagli assi x e y per un angolo pari a 45°, come mostrato in figura 13.

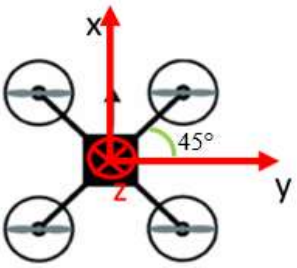


Figura 13 Configurazione a croce

Dinamica di volo di un quadricottero

Descriviamo ora tutti i possibili movimenti che può compiere un quadricottero.

Il sistema drone è un sistema a sei gradi di libertà (in riferimento al B.F.F.R.) composto da:

- *Beccheggio (Pitch)*: lo spostamento angolare attorno l'asse y;
- *Imbardata (Yaw):* lo spostamento angolare attorno l'asse z;
- *Rollio (Roll):* spostamento angolare attorno l'asse x;
- *X:* spostamento lungo l'asse x;
- *Y:* spostamento lungo l'asse y;
- *Z:* spostamento lungo l'asse z.

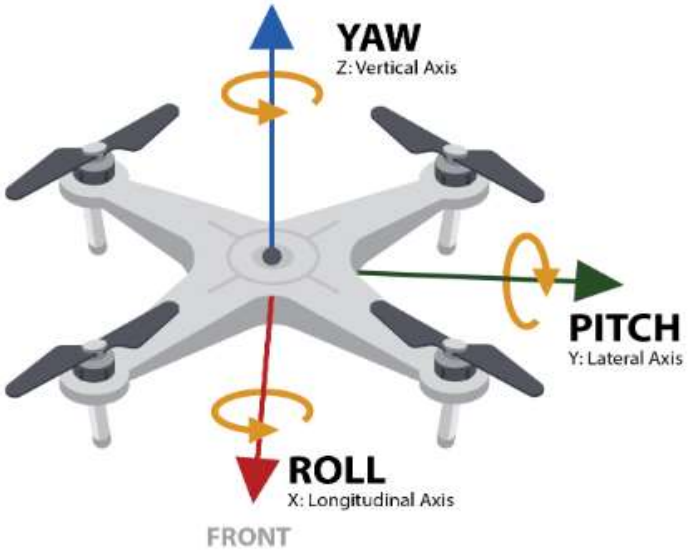


Figura 14 Il BFFR con i relativi movimenti

Da qui si evince il fatto di essere un sistema *sotto-attuato*, ossia che il numero di variabili di controllo (motori) è minore di quelle da controllare (i sei movimenti).

Vediamo ora come vengono realizzati i movimenti descritti; partiamo dalla figura 15:

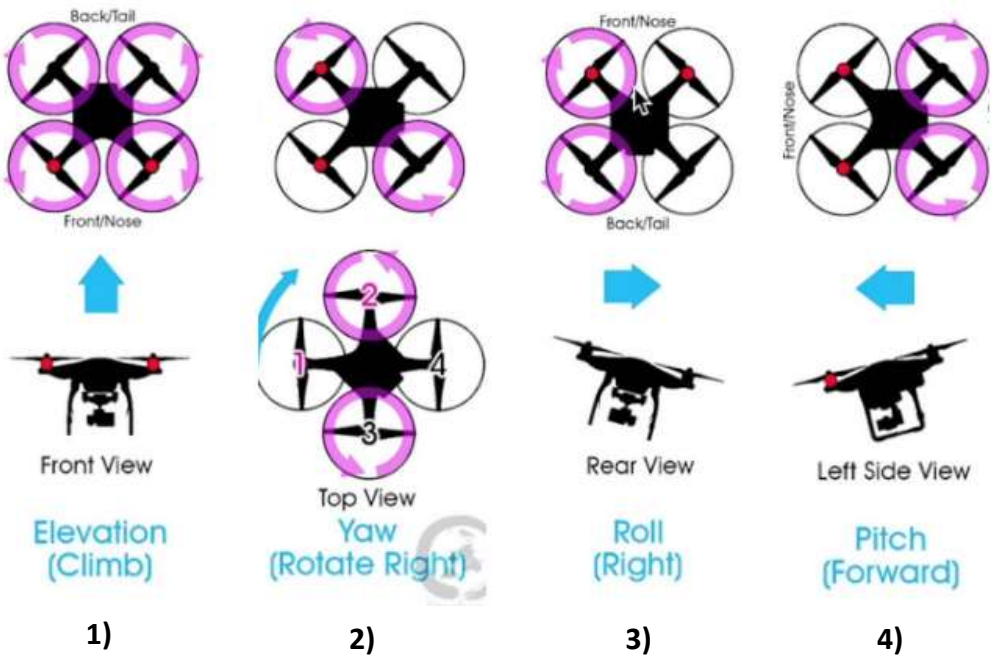


Figura 15 Schema esemplificativo dei possibili movimenti di un quadricottero a croce

In una prima istanza si nota che il controllo dei movimenti avviene attraverso quello del senso di rotazione di ogni motore e che risulta importante anche la velocità di rotazione di questi (in fucsia, le velocità maggiori).

In più si nota anche che il senso di rotazione di motori adiacenti risulta discorde, questo a causa dell'*effetto giroscopio* di cui si lascia al lettore un eventuale approfondimento.

In 1) si genera un movimento verticale facendo ruotare tutti i motori a stessa velocità.

Se la somma delle forze generate da ogni singolo motore supera la forza peso del drone, si avrà un movimento verso l'altro coincidente con l'asse delle z negative in accordo con il sistema di riferimento solidale; si avrà invece un movimento simmetrico qualora la forza totale risultasse minore della forza peso.

Se la forza risultante è di valore pari a quella della forza peso, si raggiunge la situazione di volo stazionario, ovvero quella di *hovering* in cui il drone mantiene la sua posizione raggiunta.

In 2) si ottiene una rotazione attorno l'asse delle z aumentando la velocità di una coppia di motori disposti lungo lo stesso asse.

In 3) si provoca uno spostamento lungo l'asse x aumentando la velocità di una coppia di motori adiacenti; questo genera una differenza di spinta tra i due lati del drone causando un'inclinazione attorno l'asse x, appunto il *Roll*.

In 4) si mostra come un procedimento uguale a 3) ma considerando l'asse y, generi un'inclinazione attorno l'asse y ottenendo uno spostamento lineare lungo l'asse in questione.

Parrot Mambo Minidrone

Il drone utilizzato nel progetto in Matlab&Simulink è il *Parrot Mambo Minidrone* prodotto dell'omonima azienda francese *Parrot SA*.

Si tratta un quadricottero con configurazione a croce di piccole dimensioni, ma non per questo carente in termini di efficienza e prestazioni.

Sotto si riportano delle immagini ed una tabella che riassume le sue caratteristiche tecniche principali:

Figura 16 Parrot Mambo con e senza carene di protezione

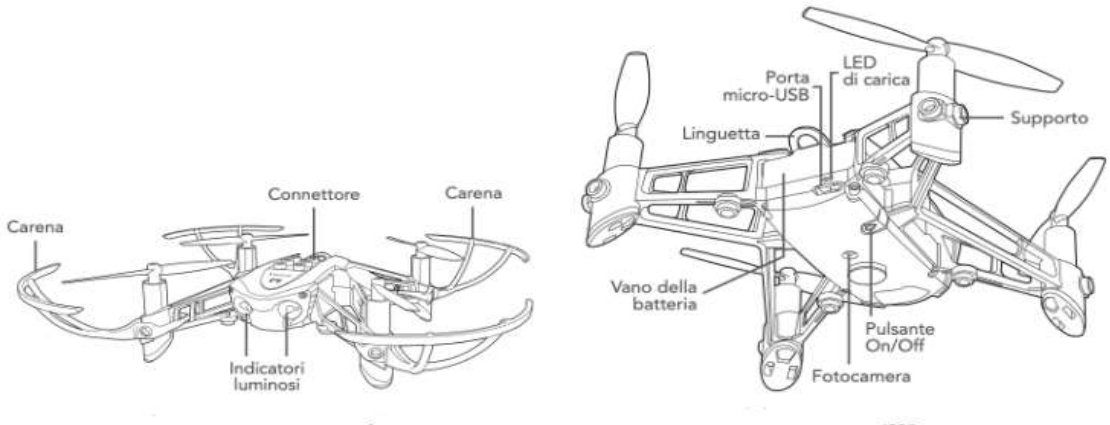


Figura 17 Diagrammi annotati

Tabella 2 Specifiche tecniche principali

Specifiche	Valori
Dimensioni	18 x 18 x 5 cm (con carene)
Peso	63 g
Batteria	LiPo 660 mAh
Autonomia	10 min ca
Velocità massima	30 km/h
Quota massima	4 m
Portata bluetooth massima	100 m
Sensori e componenti	IMU, sensore ad ultrasuoni, sensore di pressione, videocamera 720p 60 FPS, motori coreless[1] brushless

Delle specifiche tecniche approfondiamo la funzione del bluetooth.

Appena il drone esce dalla fabbrica, esso dispone di un *firmware* per il controllo tramite app (Free Flight Mini); per permettere il funzionamento tramite un codice da noi sviluppato in Matlab, si deve procedere con la sostituzione dello stesso.

La procedura in questione è denominata *Hardware Setup* della quale si riportano i passaggi (in Windows):

1. Attivare il bluetooth del pc tramite *Impostazioni* > *Dispositivi* > *Bluetooth e altri dispositivi;*
2. Aprire MATLAB® quindi *Add-Ons* > *Manage Add-Ons* > click sull'icona di ingranaggio del *Simulink Support Package for Parrot Minidrones* precedentemente installato;
3. Dalla schermata apparsa, selezionare l'utilizzo dei driver ***Bluetooth*** del pc e procedere avanti;

[1] I motori Coreless hanno gli avvolgimenti interni di statore avvolti su sé stessi invece che su di un nucleo ferromagnetico. Sono anche brushless per la mancanza delle "spazzole".

4. Connettere il drone al pc tramite cavo USB e selezionare il modello “Mambo” dal menu a tendina; riconnettere se non apparso;
5. Seguire gli step successivi della schermata.

Una volta terminati, procedere con i successivi che sono intermedi al caricamento finale del codice sviluppato all’interno del Flight Controller[2]:

1. Dalla barra delle applicazioni di Window, selezionare l’icona bluetooth quindi *Aggiungi a Personal Area Network;*
2. In *Dispositivi e stampanti*, click su *Aggiungi dispositivo Bluetooth;* Il computer cercherà un dispositivo bluetooth nelle vicinanze;
3. Selezionare il dispositivo “Parrot Mambo” apparso; se non ripetere con *Aggiungi dispositivo Bluetooth*;
4. Click destro sul dispositivo appena associato, quindi *Connect using > Access point;*
5. Ora che il minidrone è correttamente collegato con il pc tramite bluetooth, si deve settare l’IP per permettere la comunicazione tra i due dispositivi;
6. Ritornare quindi nella schermata di *Hardware Setup* precedente > *next;*
7. Eseguire il *Test connection;*

Una volta completati anche i medesimi, procedere con il caricamento del proprio codice seguendo i supporti MATLAB online relativi a codici esempio, tipo il seguente:

https://it.mathworks.com/help/supportpkg/parrot/ug/fly-a-parrot-minidrone-using-the-hover-simulink-model.html

[2] Si ricorda che il Flight Controller è la scheda di bordo contenente gli algoritmi di controllo del drone.

Simulink Support Package for Parrot Minidrones

Introduzione al package

Il package introdotto a grandi linee nella parte introduttiva è il "*Support Package for Parrot Minidrones*";

Come detto, mette a diposizione vari progetti da cui partire per la propria implementazione, raggiungibili digitando specifici comandi nella *Command Window* di Matlab.

Prima di conoscere le specifiche del progetto utilizzato per il seguente elaborato, vediamo di quali componenti aggiuntivi (*toolbox*) si necessita per il corretto funzionamento.

Aprendo la finestra principale di Matlab e recandosi in *Add-Ons>Get Add-Ons*, si dovranno installare, qualora mancassero, i seguenti t*oolbox*:

- *Aerospace Blockset*
- *Aerospace Toolbox*
- *Computer Vision Toolbox*
- *Control System Toolbox*
- *Data Acquisition Toolbox*
- *DSP System Toolbox*
- *Image Processing Toolbox*
- *Signal Processing Toolbox*
- *Simulink*
- *Simulink 3D Animation*
- *Simulink Control Designer*
- *Simulink Library: Performance Index*
- *Simulink Support Package for Parrot Minidrones*

Per i loro approfondimenti, si rimanda alla descrizione presente nella fase di istallazione di ognuno.

A installazione completata, è possibile aprire uno dei progetti messi a disposizione dal package, che nel mio caso, è stato fatto digitando nella *Command Window* "**parrotMinidroneCompetitionStart**".

A seguito di ciò verranno create tutte le cartelle e i file costituenti del progetto *parrotMinidroneCompetition* salvati in:

C:\Utenti\nome_utente\MATLAB\Projects\examples\parrotMinidroneCompetition

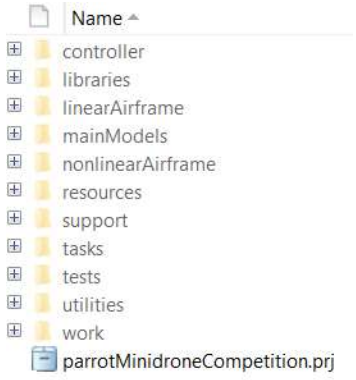


Figura 18 Cartelle e file del progetto in Matlab

Il file importante per la trattazione è quello con estensione ". prj" di figura 18;

Al click di questo, appariranno in una prima vista il modello complessivo del drone in Simulink (composto dal modello matematico e da tutti quei blocchi per la simulazione), l'ambiente virtuale e la visuale della fotocamera del drone.

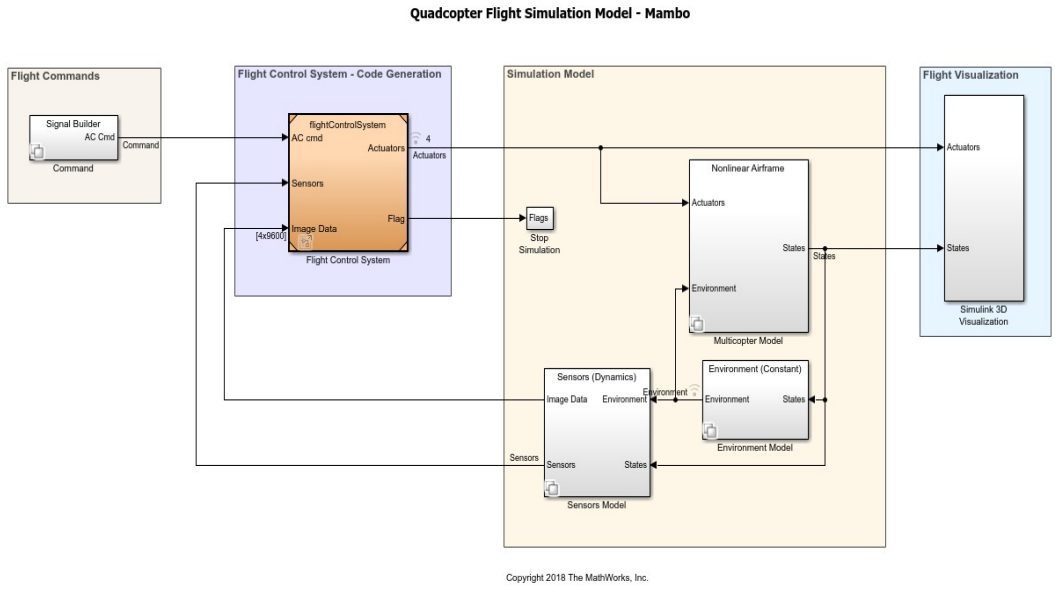


Figura 19 Modello complessivo del drone

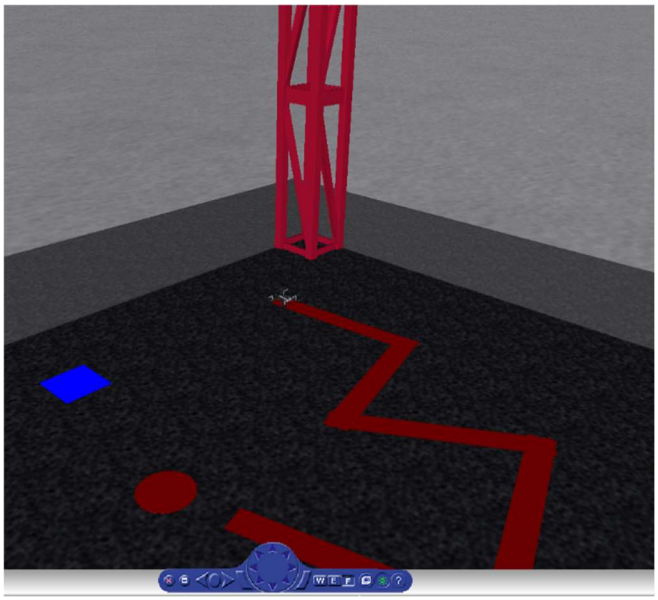


Figura 21 Ambiente virtuale, vista isometrica

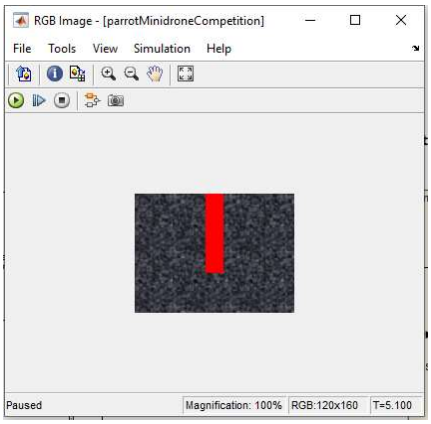


Figura 20 Visuale fotocamera (situazione di volo)

Descrizione del progetto "parrotMinidroneCompetition"

Come vediamo dalla figura 19, i blocchi costituenti del modello Simulink sono sette:

- *Command*
- *Flight Control System*
- *Stop Simulation*
- *Multicopter Model*
- *Sensors Model*
- *Enviroment Model*
- *Simulink 3D Visualization*

Per comprendere il funzionamento complessivo del modello, daremo una descrizione di ciascun blocco elencato.

Command

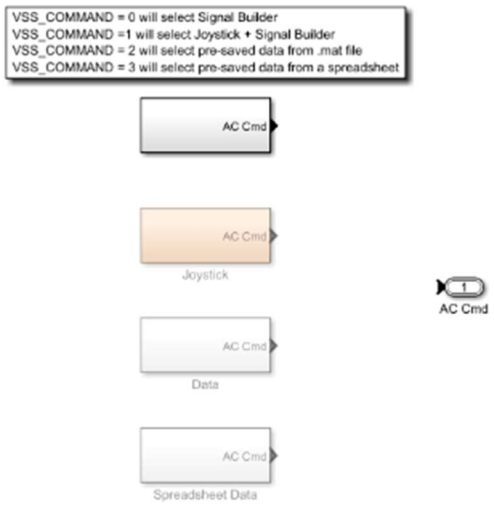


Figura 22 Blocco Command

Il blocco in questione ha il compito di assegnare i *segnali di riferimento* alle diverse variabili da controllare; tali segnali possono essere generati attraverso una delle quattro modalità selezionabili tramite il valore della variabile *"**VSS_COMMAND**"* presente all'interno del *Workspace*.

Le quattro modalità sono:

- *Signal Builder*: è la modalità preimpostata che indica la scelta dei diversi segnali di riferimento tramite l'inizializzazione degli stessi all'interno del sotto blocco *Signal Builder* di figura 22;
  In questo caso, il valore della variabile *VSS_COMMAND* è uguale a 0.
- *Joystick*: prevede l'assegnazione dei segnali mediante una periferica esterna che può essere per l'appunto, un joystick o una tastiera;
  VSS_COMMAND=1.
- *Data*: In questo caso i segnali vengono impostati tramite valori passati con un file ". mat"; esso viene opportunamente gestito per la fase di conversione "Valore-Segnale";
  VSS_COMMAND=2.
- *Spreadsheet*: Modalità simile a "*Data*" dove si prevede l'utilizzo di un foglio di calcolo anziché di un file ". mat";
  VSS_COMMAND=3.

La modalità scelta per il seguente elaborato è quella del *Signal Builder* con i valori di riferimento di default; essi sono dei *segnali a gradino* di valore 0 per *X, Y, Yaw, Roll e Pitch* e di valore -1.5 per la *Z*.

La loro modifica avviene all'interno del sotto blocco "*Position/Attitude Reference*" all'interno del *Signal Builder.*

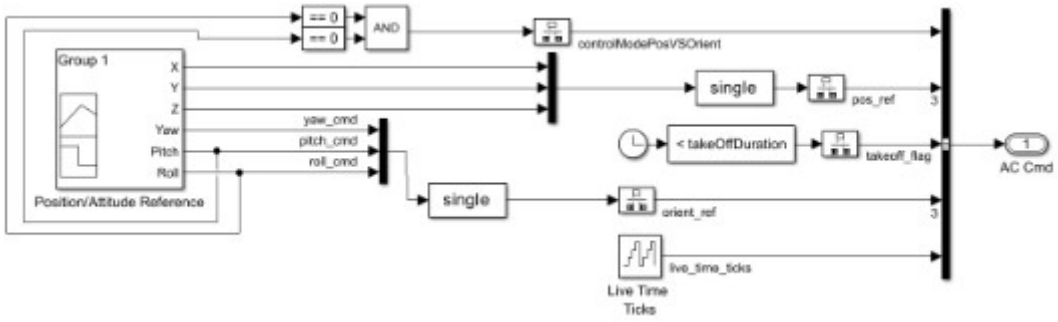


Figura 23 Blocco Signal Builder

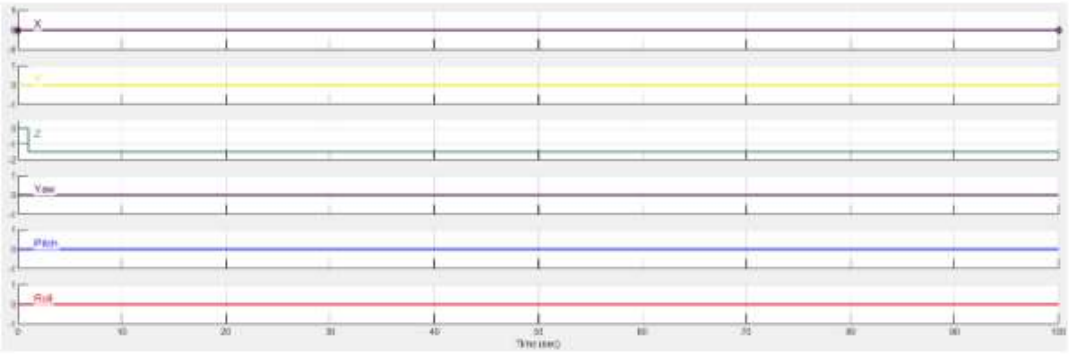


Figura 24 Segnali di riferimento di default (Position/Attitude Reference)

Flight Control System

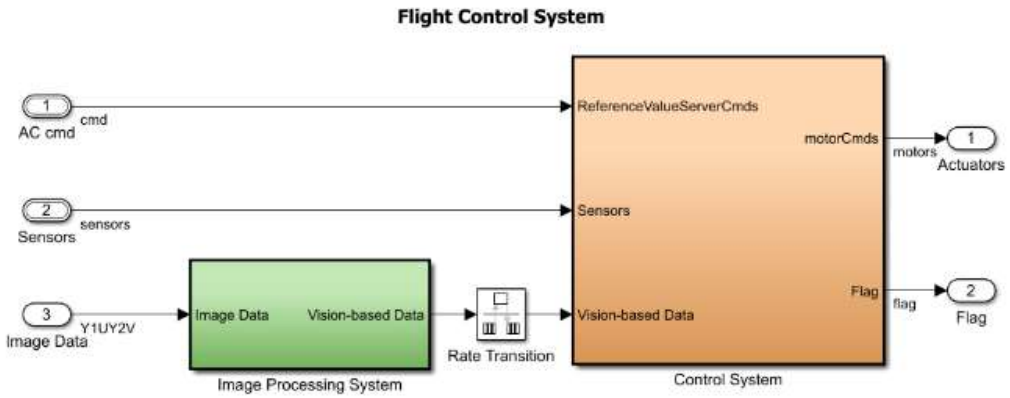


Figura 25 Flight Control System

È il blocco di maggiore interesse per la trattazione;

infatti, al suo interno, troviamo *l'Image Processing System* ed il *Control System* responsabili rispettivamente del processamento delle immagini (videocamera drone) e del controllo del volo.

Sotto si riporta una tabella degli *inputs* e *outputs* del blocco:

Tabella 3 Inputs-Outputs Flight Control System

Input (porte dati**)**	**Descrizione**
AC cmd	Segnali di riferimento blocco *Command*
Sensors	Segnali ottenuti dai sensori
Image Data	Dati videocamera in formato Y1UY2V
Output (porte dati)	**Descrizione**
Actuators	Segnali di controllo motori
Flag	Flags dal blocco *Crash Prediction Flags*

Sempre all'interno del blocco è presente il *Rate Transition*, un blocco Simulink con la funzione di gestire la transizione di segnali con diverso *periodo di campionamento*; infatti, il *periodo di campionamento* dell'*Image processing* è di 0.2 s mentre quello del *Control System* di 5 ms.

Nell'utilizzo dei blocchi in questione, per *periodo di campionamento* si intende la frequenza di esecuzione del blocco interessato.

All'interno dell'*Image processing* si trova l'algoritmo da me implementato per la rilevazione del percorso; se ne discuterà nella sezione dedicata per maggiore chiarezza.

Andando avanti con la descrizione, in figura 26 possiamo osservare l'interno del blocco *Control System:*

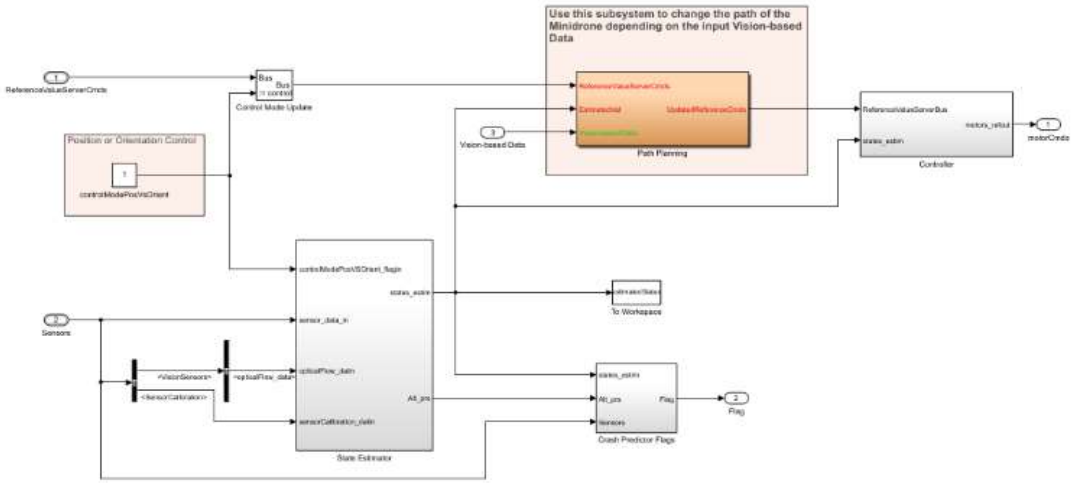


Figura 26 Blocco Control System

In esso troviamo il *Path Plannig*, lo *State Estimator,* il *Controller* e il *Crash Predictor Flags*;

Iniziamo col dire che il P*ath Planning* è il blocco all'interno del quale si trova l'altro algoritmo di mia competenza, quello con la funzione di prendere come *inputs* i dati del percorso estrapolati grazie all'*Image Processing System* e di processare quest'ultimi al fine di ottenere la risposta di controllo del drone costituita dalle coordinate x,y,z.

Anche in questa occasione, si rimanda alla sezione dedicata per l'approfondimento dello stesso.

Lo *State Estimator*, prendendo come ingressi i dati provenienti dai sensori, stima tutti quei parametri d'interesse alla dinamica del drone; essi sono x, y, z, yaw, pitch, roll e le rispettive variazioni temporali dx, dy, dz, p, q ed r.

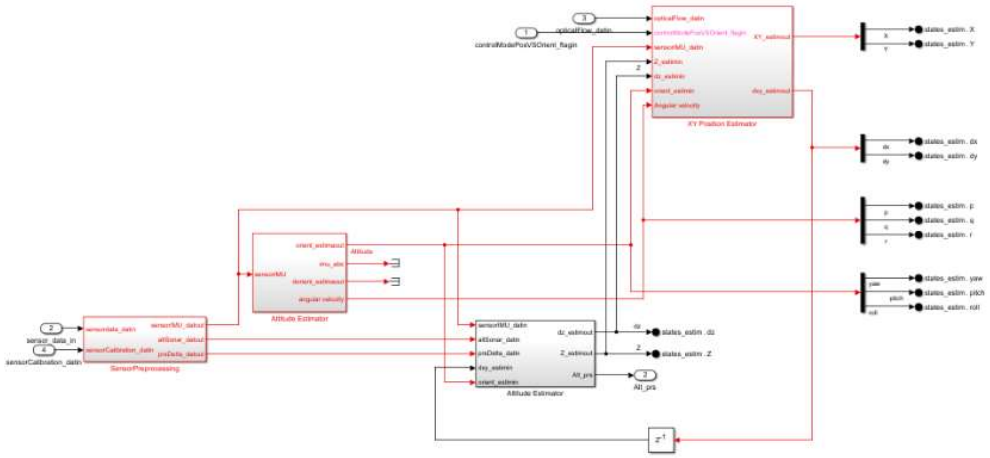


Figura 27 State Estimator

I presenti parametri sono utilizzati in vari punti del modello come, ad esempio, nel blocco *Crash Prediction Flags* che utilizza tali stime per sollevare un flag d'interruzione della simulazione qualora il drone stesse per entrare in conflitto con l'ambiente circostante; è possibile un loro utilizzo anche in altre circostanze (come vedremo per il P*ath Planning*) in base alle proprie necessità.

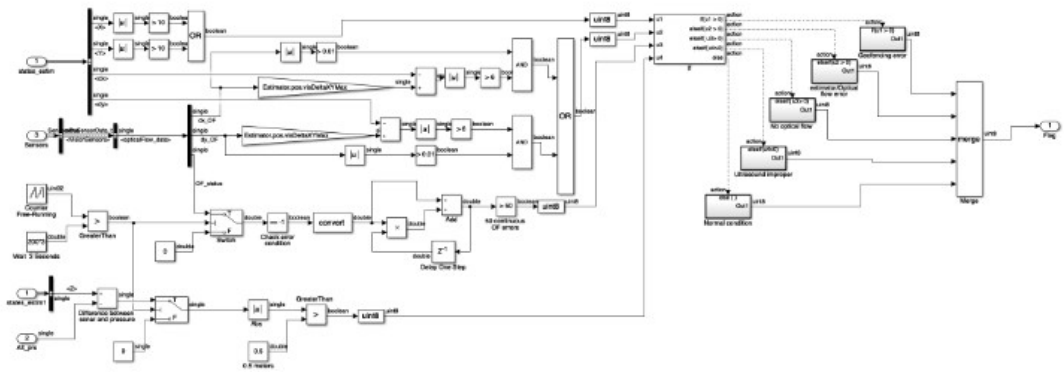


Figura 28 Crash Predictor Flags

L'ultimo blocco da trattare è il *Controller,* che si può esemplificare come un blocco contenente tutti i sistemi di controllo (a retroazione negativa) delle variabili da controllare interessate al volo del drone; l'azione di controllo si basa quindi sull'entità del valore dell'errore tra il segnale di riferimento della variabile interessata (*Signal Builder*) ed il suo valore stimato (*State Estimator*).

Si ricorda che il movimento del drone nel piano x-y è interessato dalle variazioni di *Pitch* e *Roll* e che quindi il controllo della posizione nel piano "passa" attraverso queste due grandezze, motivo per il quale nella figura 29, dove si mostra l'interno del blocco, troviamo il sotto blocco *Altitude Controller* per il controllo dei suddetti movimenti.

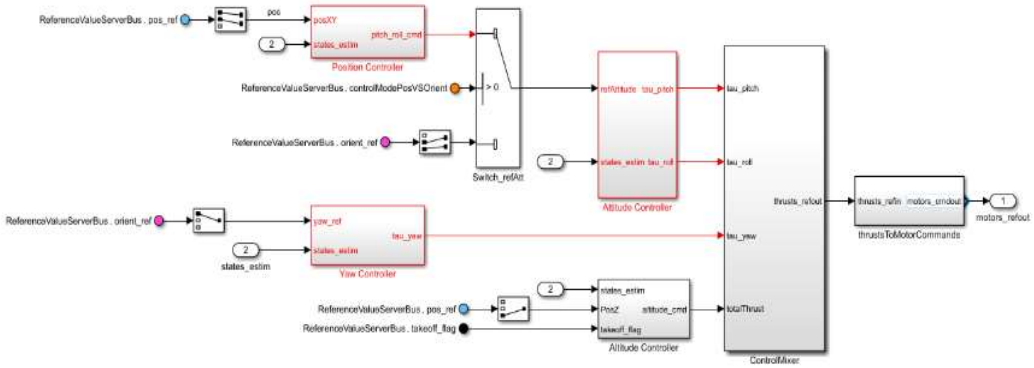


Figura 29 Blocco Controller

Inizialmente nei sistemi a retroazione negativa (interni ai vari sotto blocchi) si trovano dei controllori *PID* ma per meglio ottimizzare le azioni di controllo, si sono sostituiti questi ultimi con controllori ottenuti per mezzo di *sintesi con luogo delle radici* e di *sintesi in frequenza* di cui si discuterà nell'apposita sezione.

Multicopter Model

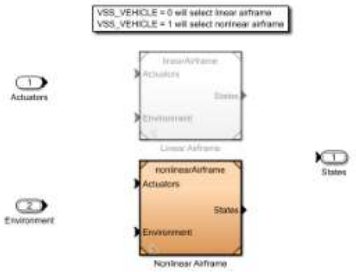


Figura 30 Multicopter Model

Prendendo come ingressi i dati provenienti dal *Flight Controller*, il *Multicopter Model* si occupa di codificare il modello matematico del drone; si può scegliere se trattarlo come un sistema lineare *(Linear Airframe)* o non lineare *(Nonlinear Airframe)* selezionando il valore di "*VSS_VEHICLE*" rispettivamente a 0 o 1.

A seconda del valore assegnatogli, verrà evidenziato il blocco scelto (figura 30);

Il valore preimpostato della variabile all'apertura del progetto è 1 e di conseguenza il modello viene trattato come non lineare.

Volendo trasmettere al lettore le principali informazioni del modello per riuscire a muoversi in autonomia, si discuteranno di entrambi i modelli.

Partendo dal *Nonlinear Airframe*, di cui vediamo la rappresentazione nella figura 31,

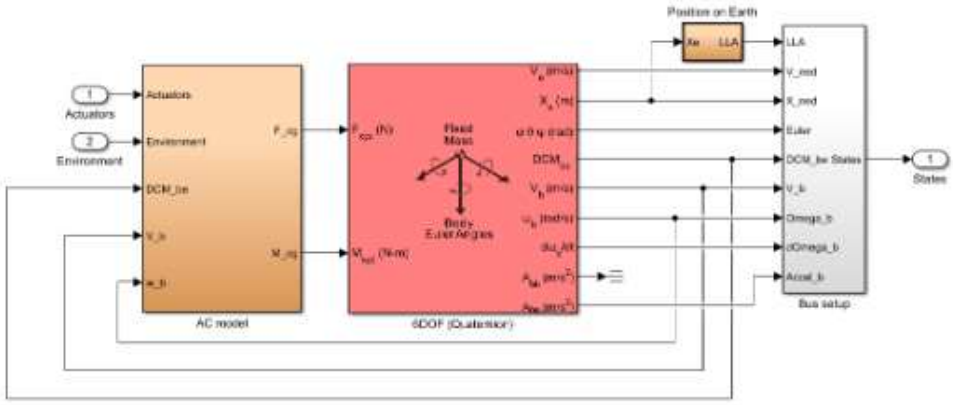


Figura 31 Nonlinear Airframe

questo è regolato dalle leggi fisiche di natura non lineare che regolano l'azione di volo di un drone.

Si notano tre blocchi principali costituenti: *AC Model*, *6DOF* e *Bus Setup*.

- ***AC Model***: prende in ingresso i dati dei motori (*Actuators*), quelli dell'ambiente virtuale (*Environment*), la matrice dei coseni che descrive la rotazione del drone rispetto ai suoi assi di riferimento (*DCMbe*) e i vettori delle velocità angolari e lineari (rispettivamente *wb* e *Vb*); tramite questi, calcola le forze e i momenti agenti sul drone (rispettivamente *F_cg* e *M_cg*);

- ***6DOF***: Calcola, a partire dai dati forniti dall'*AC model*, una serie di grandezze organizzate nella tabella 4;
- ***Bus setup***: si occupa di organizzare tutte le uscite del blocco 6DOF in unico bus dati sotto il nome di *States.*

Tabella 4 Outputs 6DOF

Grandezza	Descrizione
V_e	Velocità lineare nel sistema di riferimento North-East-Down che tiene conto dell'orientamento degli assi
X_e	Posizione X rispetto al sis. di riferimento terrestre
$\varphi\ \vartheta\ \Psi$	Angoli di assetto Pitch, Roll e Yaw (sis. di riferimento solidale)
DCMbe	Matrice dei coseni di rotazione
V_b	Vettore delle velocità lineari relativo al sistema di riferimento solidale
ωb	Vettore delle velocità angolari sempre facenti riferimento con il sistema di riferimento solidale
$d\omega b$	Accelerazioni angolari (idem per il sis. di riferimento)
A_{be}	Accelerazioni lineari

Vediamo ora il modello lineare *Linear Airframe*;

Come per tutti i modelli lineari, si tratta della *linearizzazione* attorno ad un punto di lavoro delle equazioni non lineari del modello con il quale si sta operando.

Nel nostro caso, tale compito è svolto dalla funzione **trimLinearizeOpPoint.m** contenuta all'interno della cartella "LinearAirframe" del *Workspace*.

Un sistema lineare è della forma generale di figura 33:

$$\begin{cases} \dot{x} = Ax + By \\ y = Cx + Du \end{cases}$$

Figura 33 Sistema Lineare descritto dalle matrici A, B, C e D

Il sistema *Linear Airframe* del modello Simulink prende le sembianze di figura 32:

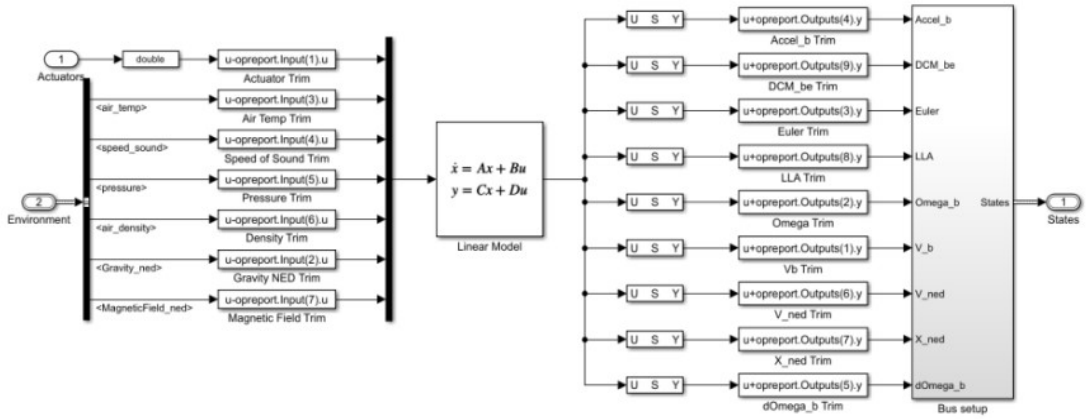


Figura 32 Blocco Linear Airframe

Enviroment Model

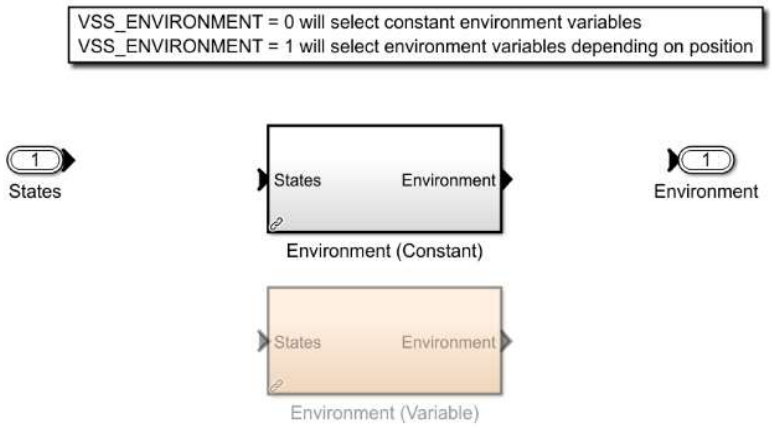


Figura 34 Enviroment Model

Dal nome del blocco si intuisce che lo scopo di quest'ultimo è quello di modellare l'ambiente circostante in cui il drone si troverà ad agire;

In particolare, modella le caratteristiche ambientali quali temperatura, pressione, coefficiente di gravità, densità dell'aria, velocità del suono e campo magnetico.

È possibile scegliere tra due diverse configurazioni che riguardano il grado di accuratezza dei parametri nei punti dello spazio; infatti, impostando il valore della variabile *VSS_ENVIROMENT* a 0, lo spazio modellato avrà le caratteristiche fisiche uguali in tutti i punti, di cui è possibile vederne i valori all'interno del blocco stesso.

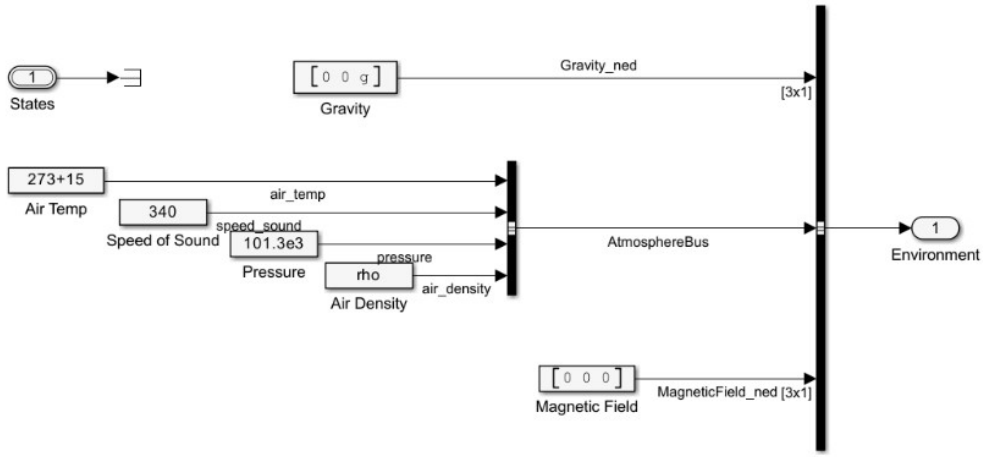


Figura 35 Blocco Enviroment (Costant)

Impostando invece il valore della variabile a 1, lo spazio modellato avrà caratteristiche fisiche differenti in base alla posizione del drone all'interno di questo; ciò è possibile recuperando le informazioni del vettore di stato del drone (blocco *Estimator*).

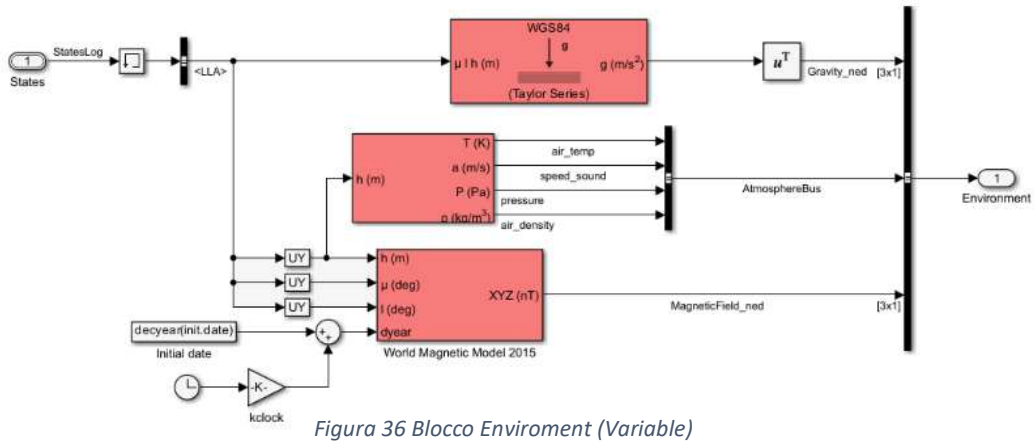


Figura 36 Blocco Enviroment (Variable)

Sensors Model

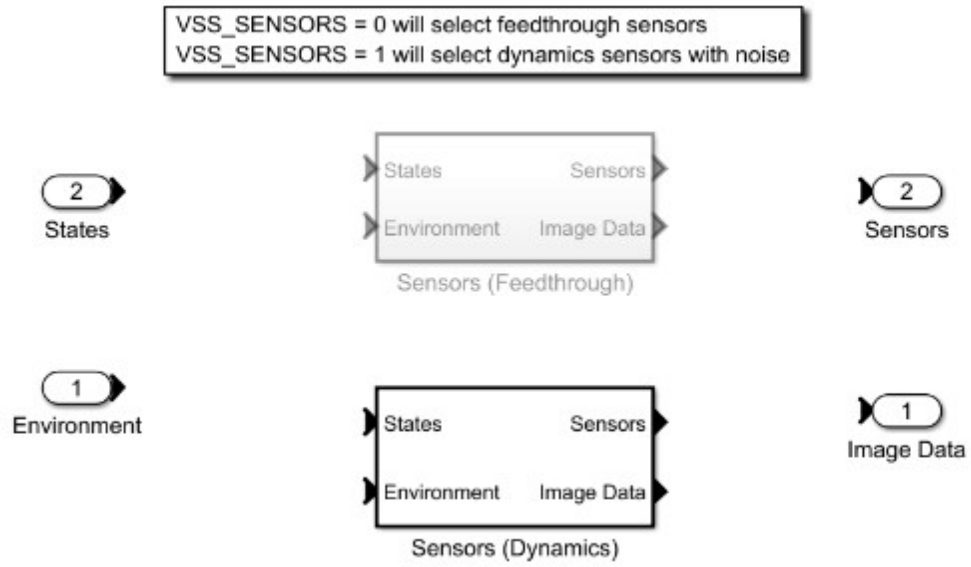


Figura 37 Sensors Model

Il blocco *Sensors Model* si occupa della modellazione dei sensori a bordo del drone; si può scegliere se valutare quest'ultimi come ideali *(Feedthrough)*, quindi come segnali senza disturbi esterni, oppure come reali (*Dynamics*) con la presenza di rumore bianco.

La scelta dei due modelli viene effettuata attraverso la variabile *VSS_SENSOR* grazie alla quale è possibile selezionare la configurazione ideale impostando il suo valore a 0 o la configurazione reale con il valore a 1; la scelta per il progetto personale è ricaduta nel modello *Dynamics* per avvicinarsi alle condizioni reali.

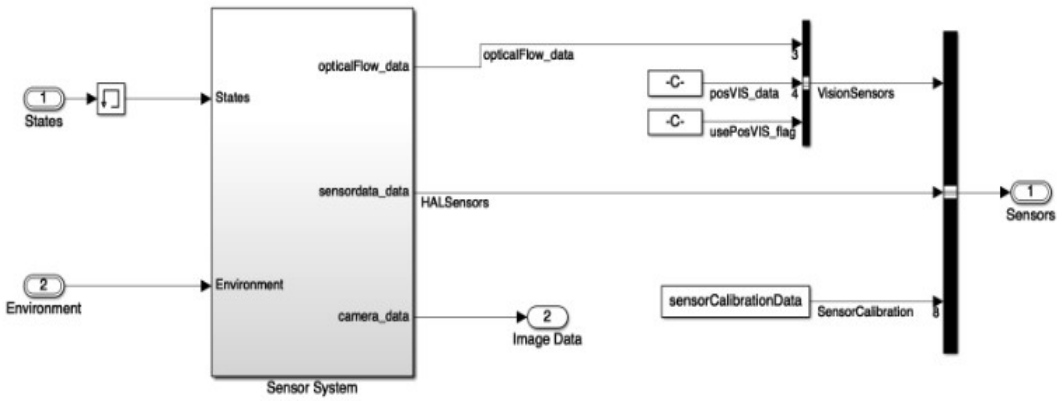


Figura 38 Sensor Model (Dynamics)

Dalla figura 38 si nota che il blocco responsabile dell'elaborazione dei segnali dei sensori risulta essere il *Sensor System,* il quale restituisce i dati di ciascun sensore.

Nell'introdurre questi, risulta comodo analizzare l'interno del blocco;

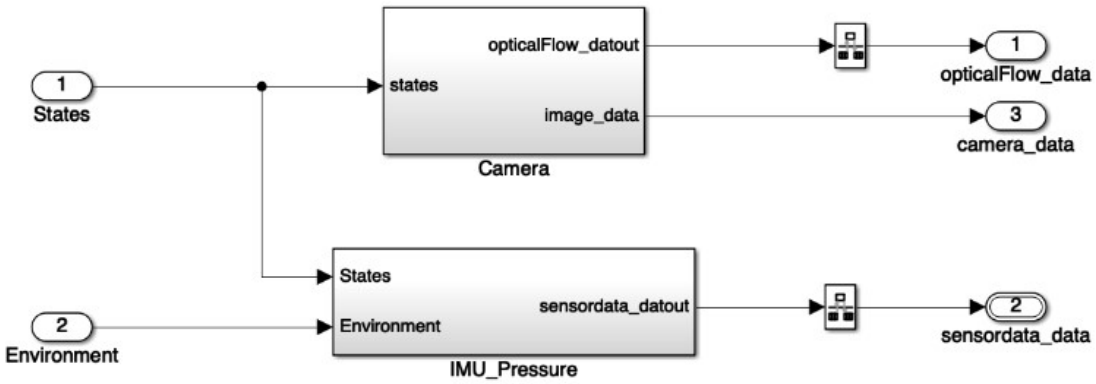


Figura 39 Sensor System

I due blocchi che troviamo all'interno riguardano la videocamera di bordo (*Camera*) e l'IMU (*IMU_Pressure*).

Da *Camera* ritroviamo in uscita l'immagine in formato Y1UY2V (*camera_data*) e i risultati del flusso ottico (*opticalFlow_data*).

Il flusso ottico è "una tecnica utilizzata nella visione artificiale per stimare il movimento relativo di oggetti o del drone in base ai cambiamenti di luminosità tra i fotogrammi consecutivi di una sequenza di immagini tramite un sensore o un algoritmo".

Nel nostro caso, il flusso ottico è utile per stimare la velocità di volo e per monitorare il movimento del body frame.

Dall' *IMU_Pressure* vengono invece ritornati i dati dell'IMU (*sensordata_data*).

Simulink 3D Visualization

Il presente blocco ha lo scopo grafico di visualizzare istante per istante la simulazione del drone che insegue il percorso.

Il blocco è costituito a sua volta da due sotto blocchi con le funzioni di visualizzare il movimento del drone nell'ambiente virtuale (*Extract Flight Instruments*) e di stampare a schermo di alcuni valori numerici inerenti alla dinamica del drone (*Simulink 3D*).

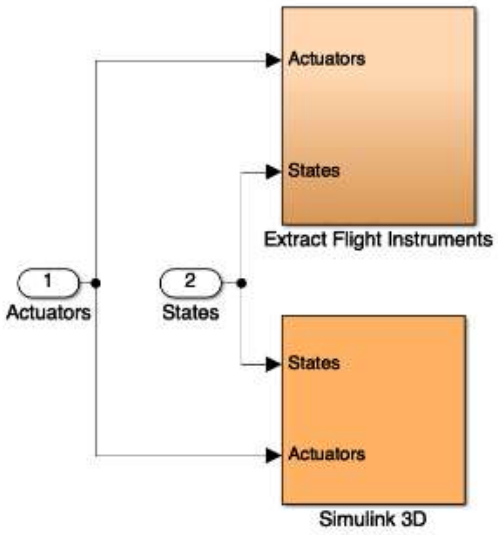


Figura 40 Simulink 3D Visualization

L'*Extract Flight Instruments* si occupa di acquisire i valori della posizione z, delle variabili responsabili dell'assetto, delle velocità lineari e angolari per poterle visualizzare

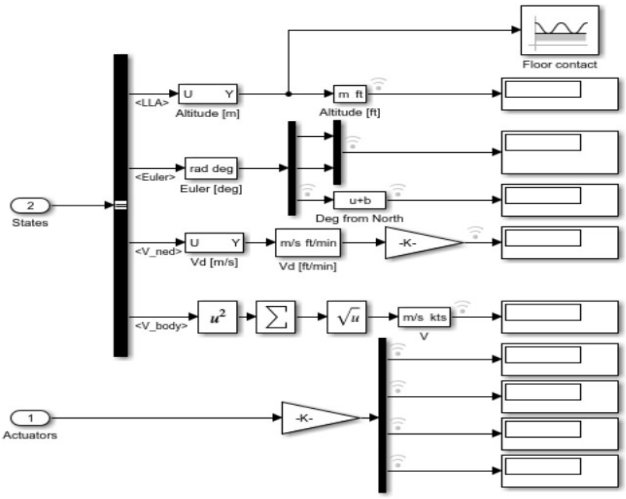


Figura 41 Extract Flight Instruments

successivamente a schermo in modo da avere un riscontro numerico sullo stato della simulazione; in più, con queste, gestisce anche il contatto con il suolo del drone (sempre per scopi grafici).

Il *Simulink 3D* permette la visualizzazione del movimento del drone grazie agli algoritmi implementati nel *Flight Controller*.

È grazie quindi a questo blocco che possiamo osservare il drone impegnato a seguire il percorso interessato.

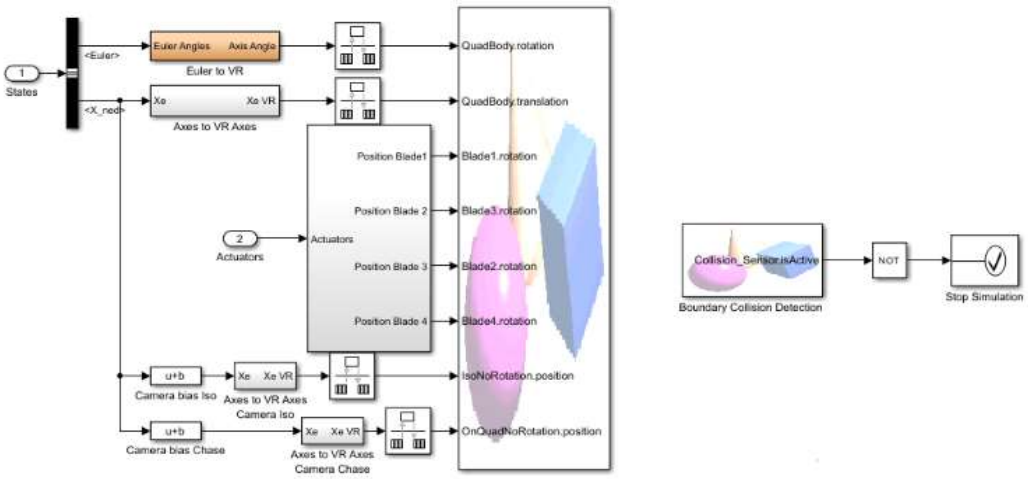


Figura 42 Simulink 3D

Implementazione Image Processing System

Introduzione

Entriamo nella parte del progetto di mia competenza; iniziamo *dall'Image Processing System*.

Ricordiamo che esso è il blocco dedicato al processamento delle immagini del *parrotMinidroneCompetition*, contenuto all'interno del *Flight Contoller,* di cui si riporta la vista generale.

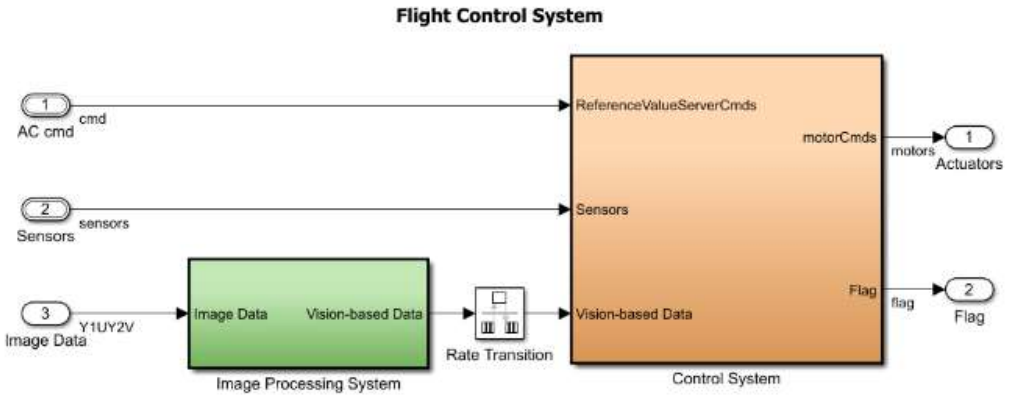


Figura 43 Flight Control System

Il mio compito è stato quello di estrapolare dei dati utili al controllo del "drone inseguitore" dalle inquadrature provenienti dalla videocamera in ogni instante di tempo; si sono estrapolate così le caratteristiche salienti del percorso.

La struttura dell'algoritmo implementato fa riferimento allo pseudocodice di figura 44 tratto dall'articolo scientifico "A Vision-Based Algorithm for a Path Following Problem"[3];

nelle prossime righe si analizzerà quest'ultimo per guidare il lettore alla comprensione del modello Simulink posto a fine analisi.

[3] Autori: Mario Terlizzi, Giuseppe Silano, Luigi Russo, Muhammad Aatif, Amin Basiri, Valerio Mariani, Luigi Iannelli, Luigi Glielmo.

Pure Pursuit Algorithm

In figura 44 si mostra lo pseudodice a cui si è fatto riferimento:

Algorithm 1 Image Processing System

```
IMG ← channelConv(IMG),
IMG ← binarization(IMG),
IMG ← erosion(IMG),
Flag_VTP ← detectTrack(IMG),
Flag_marker ← detectMarker(IMG)
if Flag_VTP then
  x_VTP, y_VTP ← vtp(frame)
  e_x ← x_VTP − x_CoM
  e_y ← y_VTP − y_CoM
else
    if Flag_marker then
      x_MARK, y_MARK ← cgMarker(frame)
      e_x ← x_MARK − x_CoM
      e_y ← y_MARK − y_CoM
return e_x, e_y, Flag_VTP, Flag_marker
```

Figura 44 Pseudocodice utilizzato per l'implementazione in Simulink

si tiene presente che nella procedura sono stati utilizzati concetti teorici (evidenziati in grassetto) che verranno approfonditi nella sezione "approfondimenti teorici".

1. Come primo passo, si converte l'immagine dal formato RGB allo **spazio colori *HSV*** per motivi pratici; questa, in termini computazionali, è una matrice 120x160 i cui elementi sono i pixel digitali rappresentanti il colore;
2. Si esegue una binarizzazione della "matrice-immagine": si impostano ad 1 i pixel rossi (facenti parte del percorso) e a 0 tutti gli altri;
3. Si effettua l'***erosione*** della matrice: per ora ci basti sapere che è un'operazione matematica per la "pulizia dell'immagine".

   Infatti, qualora non fosse chiaro, la matrice ottenuta nel punto 2 restituisce il percorso "visto" dal drone rappresentato numericamente da uni; con l'operazione in questione si eliminano eventuali elementi non desiderati dovuti ad errori d'elaborazione.

La "matrice-immagine" a questo punto ottenuta rappresenta la versione più fedele del percorso acquisito dalla videocamera del drone; si procede quindi alla sua analisi per la ricerca di un punto appartenente al percorso con l'obiettivo futuro di raggiungerlo.

Questo modo di operare rientra nella categoria degli algoritmi di tipo "P*ure Pursuit Algorithms*", letteralmente "A*lgoritmi di Puro Inseguimento".*

Continuando con i passi computazionali:

4. Dopo l'analisi della matrice, si setta un flag di riconoscimento (*Flag_VPT*) per distinguere il caso in cui si è trovato un punto appartenente al percorso (set a 1, il percorso è stato rilevato) dal caso contrario (set a 0, nessun percorso rilevato);
5. Viene eseguita una seconda analisi della matrice qualora non venga rilevato il percorso (evento che avviene alla fine del percorso a causa della distanza che intercorre tra il tracciato e il disco di atterraggio); viene settato un altro flag di riconoscimento (*Flag_marker*) a 1 nel caso in cui viene trovato un punto nel disco d'atterraggio, a 0 negli altri due casi (percorso rilevato nel punto 4 o nessun punto trovato per l'atterraggio);
6. Si verifica se il percorso è stato o meno rilevato attraverso il relativo flag; nel caso affermativo:
    a) Si recuperano le coordinate x, y del punto precedentemente determinato (X_{VPT}, Y_{VPT}) e il baricentro del drone (X_{com}, Y_{com}) (supposto coincidente con il centro della videocamera);
    b) Si calcola la coordinata x dell'errore (e_x);
    c) Si calcola la coordinata y dell'errore (e_y);
7. Nel caso contrario, si verifica se è stato rilevato il punto d'atterraggio; se sì:
    a) Si recuperano le coordinate x, y del punto determinato nell'area di atterraggio (X_{MARK}, Y_{MARK});
    b) Si calcola la coordinata x dell'errore (e_x);
    c) Si calcola la coordinata y dell'errore (e_y) [4];
8. Si ritornano come outputs dell'intero blocco le grandezze e_x, e_y, *Flag_VPT, Flag_marker*.

[4] Specifichiamo che il calcolo dell'errore viene associato alle variabili e_x e e_y sia per il caso del rilevamento del percorso sia per il caso dell'atterraggio in quanto a cambiare saranno le coordinate del punto da raggiungere (in un primo caso VPT nell'altro MARK).

Si riporta ora una figura per approfondire il calcolo del punto VPT appartenente al percorso:

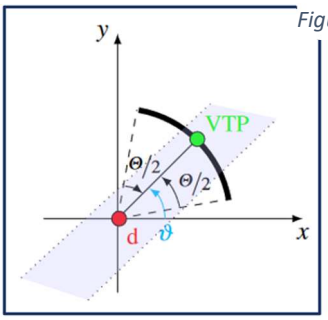


Figura 45 Diagramma ricerca punto da raggiungere

d = (H/2 ,W/2) = (60, 80)
H,W = height and Width of camera
VTP = Virtual target point
θ = arctg(VTP)
Θ = FOV = Field of View

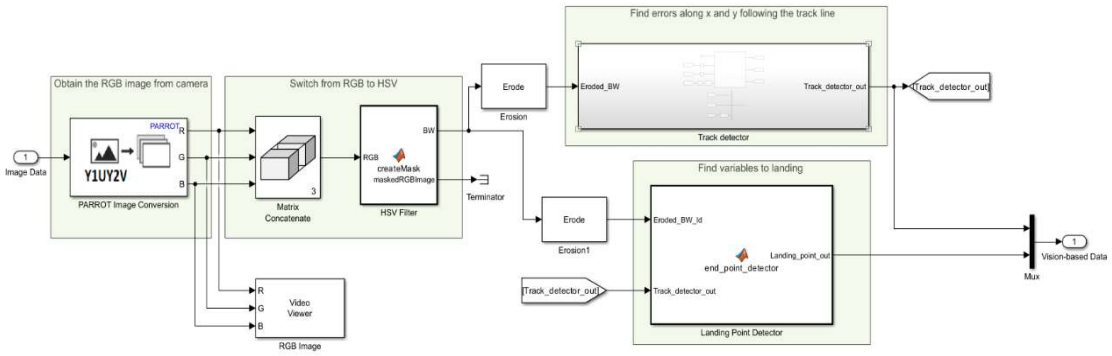
Viene analizzato il procedimento all'istante k con k ∈ N_0;

L'efficienza computazionale è dovuta dal fatto che non viene analizzata tutta l'immagine ma bensì soltanto una sua porzione; a partire dal valore VPT_{k-1} si risale all'angolo del percorso che si sta inseguendo (ϑ) e da questo si forma un'area di ricerca (*FOV*) delimitata da un limite destro e sinistro rispettivamente diminuendo e aumentando il valore dell'angolo ϑ di una costante.

Con questi aspetti, il valore del punto da raggiungere VPT è uguale al punto medio di tutti quei valori appartenenti al percorso presenti all'interno della FOV con l'ulteriore specifica di essere compresi nell'arco di circonferenza posto ad una distanza fissa dalla posizione *d* del drone all'istante k. *H* e *W* che compaiono nelle coordinate di *d* rappresentano rispettivamente l'altezza e la larghezza della visuale della videocamera.

Modello Simulink dell'Image Processing implementato

Dopo la comprensione della struttura implementata, possiamo passare all'implementazione Simulink dell'*Image Processing System*; nella pagina successiva si riporta la vista del modello a seguito della traduzione dello pseudocodice.



T.C. = 0.2 s

Figura 46 Image Processing System con il relativo tempo di campionamento

esso è stato implementato grazie all'utilizzo di blocchi presenti all'interno della libreria software, tra cui le "Matlab Function", blocchi che permettono la scrittura di codice utente in linguaggio MATLAB (questi verranno mostrati nella sezione "Codici del modello").

Da sinistra verso destra, ciò che si incontra per primo è l'input del modello: *l'Image Data*, ovvero le informazioni della videocamera provenienti dal blocco dei sensori.

Ad un primo livello esse sono nel formato "Y1UY2V" mentre successivamente,

mediante un blocco Simulink già presente al primo avvio del progetto, queste vengono convertite nella rappresentazione "RGB".

A seguito della conversione, si ottengono i tre canali rappresentanti il grado d'intensità di rosso (R), verde (G) e blu (B), ognuno dei quali codificato su 8 bit; il blocco *Video Viewer* li riceve in ingresso occupandosi della visualizzazione a schermo della visuale della videocamera.

In seguito, attraverso il blocco *Matrix Concatenate,* si uniscono i tre canali in un'unica matrice: i parametri del blocco sono quelli della modalità di concatenazione (impostata su *Multidimension Array*) e della relativa dimensione (*Concatenate dimension*, impostata su 3 in modo da essere tridimensionale); in questo modo la matrice prodotta ha dimensioni 120x160.

La matrice risultante contiene elementi con valori compresi tra 0 e 255, poiché i colori di ciascun pixel sono rappresentati nel formato RGB.

Il passo successivo consiste nel rappresentare i colori dei pixel nello spazio colori HSV tramite il blocco *HSV Filter,* la prima *Matlab Function* che incontriamo.

Il codice MATLAB implementato deriva dalla generazione di codice dopo l'utilizzo del *toolbox Color Thresholder;* tramite una prima interfaccia grafica, esso permette all'utente di regolare dei valori di soglia per la segmentazione dei colori di un'immagine.

Ricevendo quindi come ingresso la "matrice-RGB", i valori di soglia scelti per la segmentazione (visionabili nel codice) sono relativi al colore rosso del percorso; in più, il codice in questione, si occupa anche della binarizzazione della matrice, l'altro importante passaggio del *work-flow*.

I blocchi che si osservano in seguito sono quelli relativi alle E*rosioni*: è il momento di "pulire" l'immagine.

La matrice binaria derivante dal filtro HSV viene passata come input ai blocchi *Erosion* e *Erosion1 (*entrambi blocchi Simulink "Erode") che si occupano, rispettivamente, dell'erosione del percorso e dell'area di atterraggio.

I parametri dei blocchi sono gli *elementi strutturanti* di forma quadrata (*Erosion*) e circolare (*Erosion1*), cosicché venga ottimizzata la matrice rappresentante la visuale della posizione del drone; si ottiene la citata "matrice-immagine" dello pseudocodice.

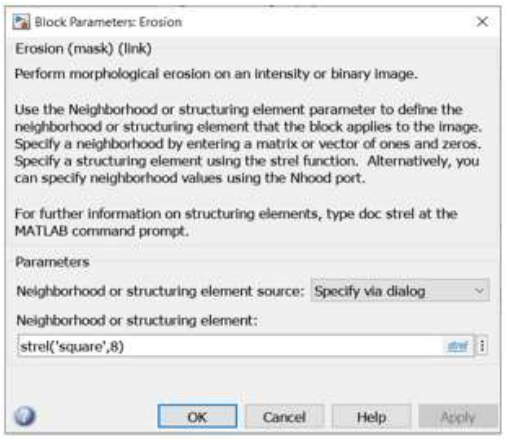


Figura 47 Blocco Erosion per la rilevazione del percorso

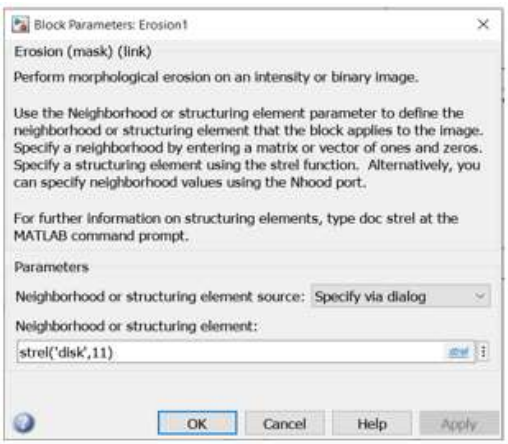


Figura 48 Blocco Erosion1 per la rilevazione del punto d'atterraggio

Si specifica che le due operazioni vengono svolte in parallelo ad ogni ciclo di esecuzione dell'intero blocco e che quindi il risultato di una delle due matrici erose (in modo mutuamente esclusivo, dipendente dalla posizione del drone) è la matrice nulla; infatti

qualora il drone si trovi all'inizio del percorso o comunque sopra di questo, il risultato dell'erosione per mezzo del disco (di raggio 11) comporta la scomparsa di tutti gli uni dalla matrice non essendo questo in nessun modo incluso; mentre ne vedremo diminuire il loro numero per mezzo dell'erosione con l'elemento strutturante quadrato (di lato 8).

Le due matrici erose vengono passate come ingresso ai blocchi che si occupano delle rilevazioni del percorso (*Track detector*) e del disco d'atterraggio (*Landing Point Detector*), estrapolandone i dati salienti per il controllo futuro dell'inseguimento.

Iniziando dal *Track Detector,* esso è organizzato in un sottosistema (blocco Simulink *Subsystems*) in cui all'interno troviamo la *Matlab Function* operante (*errors_flag_generator*);

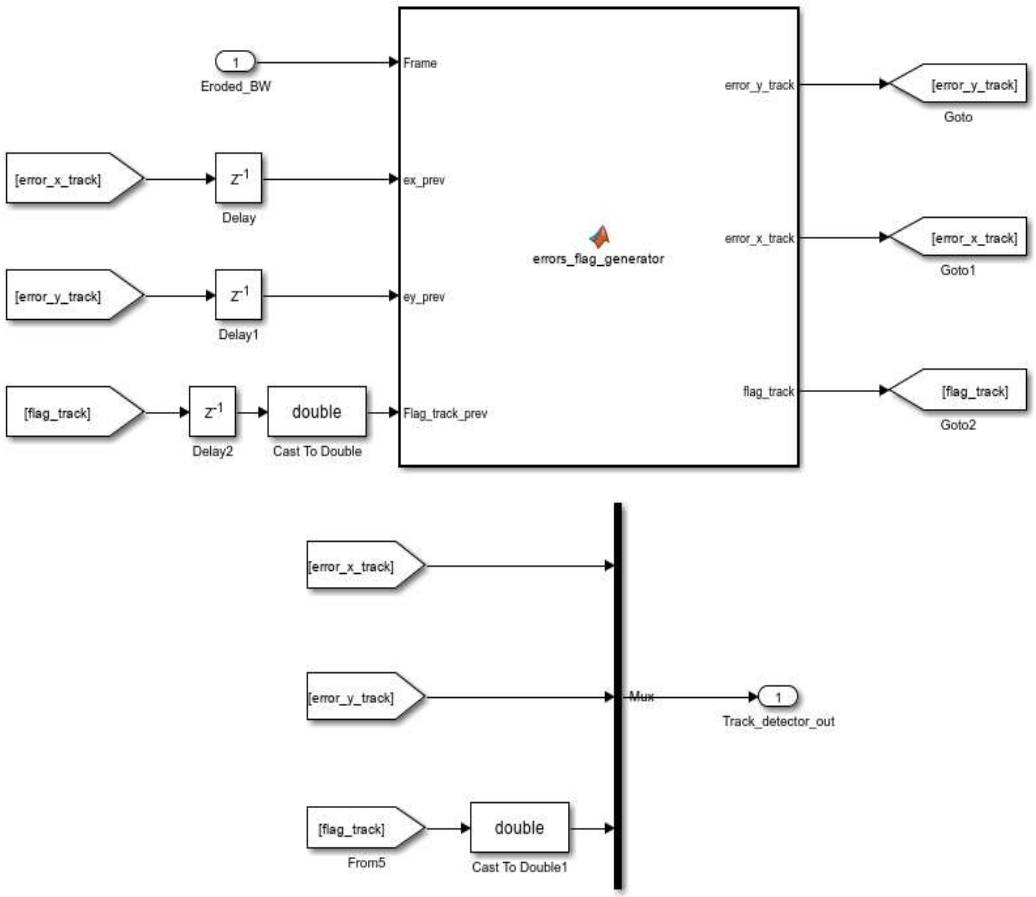


Figura 49 Track Detector

si osserva che l'implementazione della stessa è di tipo *ricorsivo*, ossia che i risultati in un generico istante $k \in N_0$ dipendono da quelli nell'istante k-1; questo è reso possibile utilizzando i blocchi della libreria software *Goto* (per le uscite della funzione) e *From (*per gli ingressi*)*, i quali permettono di prelevare e riportare delle grandezze in vari punti del modello.

La scelta di proporre un algoritmo di tipo ricorsivo è motivata dal fatto che la ricerca di un nuovo punto da raggiungere viene svolta utilizzando l'inclinazione del percorso che si sta inseguendo (ottenuta per mezzo della *cotangente* del punto (*ex_prev*, *ey_prev)*), come da specifica dello pseudocodice implementato.

Essendo "un'impostazione ricorsiva", le uscite riportate in ingresso devono essere ritardate di un certo periodo di tempo per evitare che nel caso iniziale, all'inizio della simulazione, non si disponga dei valori degli ingressi.

A questo pensa il blocco *Delay,* riportato per ogni ingresso interessato.

Il blocco riceve come parametri la condizione iniziale della grandezza da ritardare (nel nostro caso impostata per tutte a 0) e il tempo di ritardo; quest'ultimo, dovendolo impostare su un valore intero come da specifica del blocco, si è impostarlo ad 1 essendo il valore utile più vicino al tempo di campionamento di 0.2 s.

L'*errors_flag_generator* restituisce quindi i valori delle coordinate x-y dell'errore e del flag di riconoscimento a seguito dell'analisi della "matrice-immagine" del percorso.

Le uscite vengono poi coinvolgiate in un multiplexer per creare un unico canale di uscita sotto il nome di *Track_detector_out*.

Nota: è stato necessario un cast di tipo double per il valore del flag in quanto l'utilizzo del *mux* richiede che i *tipi* di dato in ingresso siano gli stessi.

Mentre il *Track_detector* è impegnato nella rilevazione del percorso*,* il *Landing Point detector* lo è nella rilevazione del disco di atterraggio; esso è rappresentato in maniera diretta dalla funzione *end_point_detector* che prende in ingresso la "matrice-immagine" del blocco *Erode1* e il canale di uscita *Track_detector_out,* utilizzato per conoscere il valore del flag di riconoscimento del percorso*;* Grazie a questi, procede nell'analisi della

matrice restituendo le coordinate degli errori lungo x-y e il flag di riconoscimento del disco di atterraggio in base alla posizione del drone.

L'output finale dell'*Image Processing System* prende il nome di *Vision-based Data*, un canale d'uscita che trasmette verso l'esterno gli *outputs* dei blocchi *Track_detector* e *Landig Point Detector*.

Il *Vision-based Data* verrà poi richiamato all'interno del *Control System* per il controllo dell'inseguimento del percorso da parte del drone.

Codici del modello

Sotto si riportano i codici delle *Matlab Functions* incontrate nel modello.

createMask HSV Filter

```
function [ BW , maskedRGBImage ] = createMask ( RGB )
% CreateMask Threshold RGB image using auto - generated code from
% colorThresholder app . [ BW , MASKEDRGBIMAGE ] = createMask ( RGB )
% thresholds image RGB using auto - generated code from the
% colorThresholder app . The colorspace and range for each channel of
% the colorspace were set within the app . The segmentation mask is
% returned in BW , and a composite of the mask and original RGB
% images is returned in maskedRGBImage.

% Auto - generated by colorThresholder app on 04 - Jan -2023
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Convert RGB image to chosen color space
I = rgb2hsv ( RGB ) ;

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.960;
channel1Max = 0.037;
% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.108;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.150;
channel3Max = 1.000;

% Create mask based on chosen histogram thresholds
sliderBW = ( ( I (: ,: ,1) >= channel1Min ) | ( I (: ,: ,1) <= channel1Max ) ) &...
( I (: ,: ,2) >= channel2Min ) & ( I (: ,: ,2) <= channel2Max ) & ...
( I (: ,: ,3) >= channel3Min ) & ( I (: ,: ,3) <= channel3Max ) ;
BW = sliderBW ;

% Initialize output masked image based on input image .
maskedRGBImage = RGB ;

% Set background pixels where BW is false to zero .
maskedRGBImage ( repmat (~ BW ,[1 1 3]) ) = 0;

end
```

Figura 50 Funzione createMask

errors_flag_generator

```
function [error_y_track, error_x_track, flag_track] = ...
    errors_flag_generator(Frame, ex_prev, ey_prev, Flag_track_prev, ...
    MIN_RADIUS_CROWN, MAX_RADIUS_CROWN, FOV, X_COM, Y_COM)
% La seguente funzione calcola gli errori lungo x e y tra il VPT (ottenuto tramite una maschera circolare) e la
% posizione del drone rappresentata dal suo centro di gravità;
%
% Inputs:
% - Frame: Immagine erosa--> matrice binaria i cui elementi rappresentano
%          pixel rossi (1) o meno (0)
% - ex_prev: valore precedente dell'errore lungo x; è utilizzato come
%            referenza per il calcolo di quello successivo.
% - ex_prev: valore precedente dell'errore lungo y; è utilizzato come
%            referenza per il calcolo di quello successivo.
% - flag_track_prev: valore precedente dello stato del drone: 1 se sopra alla traccia 0 altrimenti

% Parameters:
% - MIN_RADIUS_CROWN: Valore del raggio interno della maschera.
% - MAX_RADIUS_CROWN: Valore del raggio interno della maschera.
% - FOV: Costante per il calcolo della porzione della maschera da
%        analizzare.
% - X_COM: Coordinata x del centro di massa del drone.
% - Y_COM: Coordinata y del centro di massa del drone.

% Outputs:
% - error_y_track: Errore corrente lungo x
% - error_x_track: Errore corrente lungo y
% - track_angle_prev: valore precedente dell'angolo del percorso.
% - flag_track: valore corrente dello stato del drone: 1 se sopra alla traccia 0 altrimenti

persistent x_accumulator
persistent y_accumulator
persistent pixel_count
persistent Initial_time

if isempty(x_accumulator)
    x_accumulator = 0;
end

if isempty(y_accumulator)
    y_accumulator = 0;
end

if isempty(pixel_count)
    pixel_count = 0;
end

if isempty(Initial_time)
    Initial_time = tic;
end
```

Le variabili di tipo *persistent* mantengono la loro dichiarazione tra le chiamate successive della funzione.

```
scan_orientation = atan2(-ex_prev,ey_prev);
scan_orientation = mod(scan_orientation,2*pi);

if Flag_track_prev == 1

   % set dei pixel dentro la maschera---> inizialmente un quadrato di lato l=2*MAX_RADIUS_CROWN
   for i = (X_COM-MAX_RADIUS_CROWN):(X_COM+MAX_RADIUS_CROWN)

       for j = (Y_COM-MAX_RADIUS_CROWN):(Y_COM+MAX_RADIUS_CROWN)

           norm = sqrt(((i-(X_COM-0.5))^2)+((j-(Y_COM-0.5))^2)); %L'aggiunta di 0.5 assicura che il punto centrale
           % sia posizionato esattamente nel centro di una cella.

           if norm >= MIN_RADIUS_CROWN && norm <= MAX_RADIUS_CROWN %se la distanza rientra all'interno dei due raggi

               axis_x = (i-(X_COM-0.5));
               axis_y = (j-(Y_COM-0.5));

               yaw_point_temp = atan2(axis_x,axis_y); %angolo rispetto al centro della corona: COM
               yaw_point_temp = mod(yaw_point_temp,2*pi);

               %Calcolo dei limiti della porzione della maschera da considerare (FOV)
               threshold_1 = mod((scan_orientation+(pi/FOV)),2*pi); %si aumenta l'angolo di circa 60 gradi (fov=2.9 e 90 = pi/2 )
```

```
threshold_2 = mod((scan_orientation-(pi/FOV)),2*pi); %si diminuisce l'angolo di circa 60 gradi (fov=2.9 e 90 = pi/2 )

% a questo punto si verifica se l'angolo yaw_point_temp è compreso all'interno della FOV per mezzo dell'angolo precedente;
% in caso affermativo si setta flag_threshold a 1
if threshold_1 > threshold_2

    flag_threshold = yaw_point_temp <= threshold_1  && yaw_point_temp >= threshold_2;

else

    flag_threshold = (yaw_point_temp <= 2*pi  && yaw_point_temp >= threshold_2) || (yaw_point_temp <= threshold_1  && yaw_point_temp >= 0);

end
```

```
    if flag_threshold %se il punto all'interno della FOV

        if frame(i,j)==1 % si verifica che il pixel appartenga al tracciato

            pixel_count = pixel_count + 1;
            x_accumulator = x_accumulator + i;
            y_accumulator = y_accumulator + j;

        end

    end

end

    end
end
```

```
elseif toc(Initial_time) <= 4

    % Parte di codice al primo avvio: tracciato non rilevato
    for i = (X_COM-MAX_RADIUS_CROWN):(X_COM+MAX_RADIUS_CROWN)

        for j = (Y_COM-MAX_RADIUS_CROWN):(Y_COM+MAX_RADIUS_CROWN)

            norm=sqrt(((i-(X_COM-0.5))^2)+((j-(Y_COM-0.5))^2));

            if norm>=MIN_RADIUS_CROWN && norm <= MAX_RADIUS_CROWN

                if frame(i,j)==1
                    pixel_count=pixel_count+1;
                    x_accumulator=x_accumulator+i;
                    y_accumulator=y_accumulator+j;
                end
            end
        end
    end
end
```

```
X_VPT = round(x_accumulator/pixel_count);
Y_VPT = round(y_accumulator/pixel_count);

ex_temp = -(X_VPT-X_COM);
ey_temp = (Y_VPT-Y_COM);


% Gestione delle variabili di errore iniziali di tipo NaN ("Not a Number")

if isnan(ex_temp)

    error_x_track = 0;
    flag_error_x = false;

else

    flag_error_x = true;
    error_x_track = ex_temp;

end
```

```
if isnan(ey_temp)

    error_y_track = 0;
    flag_error_y = false;

else

    error_y_track = ey_temp;
    flag_error_y=true;

end

flag_track = flag_error_x || flag_error_y; % Flag di rilevamento tracciato

pixel_count = 0;
x_accumulator = 0;
y_accumulator = 0;

end
```

Figura 51 Funzione errors_flag_generator

end_point_detector

```
function Landing_point_out = end_point_detector(Eroded_BW_ld, Track_detector_out, ...
    X_COM, Y_COM, FRAME_SIZE_HEIGHT, FRAME_SIZE_WIDTH)

% La seguente funzione si attiva quando si è arrivati alla fine del percorso;
% si calcolano gli errori lungo x e y per raggiumgere il centro di atterraggio.
%
%
% Inputs:
% - Eroded_BW: Immagine erosa
% - Track_detector_out : output del blocco "Track detector"
%
% Outputs:
% - error_y_end_marker: Errore lungo y tra le coordinate y del centro di
%                       massa del drone con il VPT dell'END_MARKER (punto di atterraggio)
% - error_x_end_maker: Errore lungo x tra le coordinate x del centro di
%                      massa del drone con il VPT dell'END MARKER.
% - flag_end_marker: Flag true quando viene rilevato l'END MARKER
%
%
% Parameters:
% - X_COM: Coordinata x del centro di massa del drone
% - Y_COM: Coordinata y del centro di massa del drone
% - FRAME_SIZE_WIDTH: Larghezza dell'immagine, numero colonne matrice
% - FRAME_SIZE_HEIGHT: Altezza dell'immagine, numero righe matrice

persistent column_accumulator
persistent count_pixel
persistent row_accumulator

if isempty(column_accumulator)
    column_accumulator = 0;
end

if isempty(count_pixel)
    count_pixel = 0;
end

if isempty(row_accumulator)
    row_accumulator = 0;
end
```

```
flag_track = Track_detector_out(3);

% Check della fine del percorso
if flag_track == 0

     for i=1:FRAME_SIZE_HEIGHT

        for j=1:FRAME_SIZE_WIDTH

            %check all the pixel belonging to the end marker
            if Eroded_BW_ld(i,j)==1
                count_pixel = count_pixel +1;
                column_accumulator = column_accumulator + j;
                row_accumulator = row_accumulator + i;

            end

        end
    end

    if count_pixel ~= 0
        flag_end_marker=1;
        X_VPT_END = round(row_accumulator/count_pixel);
        Y_VPT_END = round(column_accumulator/count_pixel);
    else
        flag_end_marker=0;
        X_VPT_END=1;
        Y_VPT_END=1;
    end

else
    X_VPT_END=0;
    Y_VPT_END=0;
    flag_end_marker=0;
end

error_x_end_marker = -(X_VPT_END-X_COM);
error_y_end_marker = (Y_VPT_END-Y_COM);

count_pixel = 0;
column_accumulator = 0;
row_accumulator = 0;

Landing_point_out = [error_x_end_marker, error_y_end_marker, flag_end_marker];


end
```

Figura 52 Funzione end_point_detector

Approfondimenti teorici

Spazio dei colori HSV

I sistemi che mirano alla classificazione dei colori dal punto di vista della loro rappresentazione e codifica nei sistemi digitali vengono chiamati “formati colore”.

Data la loro possibilità di essere mappati in spazi a due, tre o quattro dimensioni simili ai sistemi di coordinate cartesiane, vengono anche denominati “spazi colore”; l’utilizzo di uno ad esclusione di altri dipende dall’applicazione e dalle esigenze specifiche, come ad esempio il formato RGB impiegato nei display e nelle applicazioni multimediali.

Il formato HSV permette un controllo intuitivo sulle proprietà dei colori migliorandone la loro manipolazione e selezione (*segmentazione*).

Proprio per la selezione intuitiva dei colori interessati da un’immagine, l’HSV è divenuto il formato colore da noi utilizzato per la selezione del colore rosso del percorso.

Questo tipo di rappresentazione si avvicina più fedelmente alla modalità con cui noi essere umani interpretiamo un colore; infatti, l’occhio umano è sensibile a diversi aspetti di un colore, tra cui la tonalità, la saturazione e la luminosità, proprio i parametri che rappresentano un colore nel formato HSV.

Vediamoli:

- **H** (*Hue*-tonalità): rappresenta il tipo di colore interessato, ovvero, riferendoci ad un oggetto, quello che in gergo diciamo essere ad esempio “Rosso” anziché “Blu”;
- **S** (*Saturation*-saturazione): indica la quantità di grigio presente nel colore; è quella caratteristica che dona al colore interessato più o meno *vividezza*;
- **V** (*Value*-luminosità): descrive la luminosità del colore, ossia la quantità dei bianchi che rende il colore interessato “chiaro” o “scuro”.

La rappresentazione grafica più comune dello spazio colori HSV è quella di un cono, in cui i valori di *Saturation* e *Value* sono compresi tra 0 e 1 e la tonalità (*Hue)* varia da 0 a 360°.

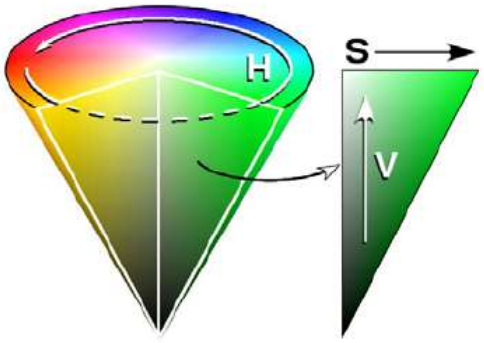


Figura 53 Rappresentazione HSV

Nella tabella 5 si specificano i valori degli angoli di *tonalità*:

Tabella 5 Range di valori di Tonalità

Colore (*Hue*)	**Angolo**
Rosso	[0º, 60°]
Giallo	[60º, 120°]
Verde	[120º, 180°]
Ciano	[180º, 240°]
Blu	[240º, 300°]
Magenta	[300º, 360°]

Erosione Morfologica

Nella rappresentazione digitale, le immagini vengono comunemente rappresentate come matrici i cui elementi sono i pixel che le compongono; i consecutivi valori sono espressi in termini del formato scelto per la rappresentazione dei colori.

Ad esempio, in una matrice che rappresenta un'immagine codificata nel formato RGB, troveremo valori compresi tra 0 e 255.

Sulle immagini vengono svolte svariate operazioni, in particolare quelle che ricadono nel campo della *Morfologia* mirano allo studio delle geometrie che le compongono.

Ciò è reso possibile utilizzando forme "modello" dette *elementi strutturanti*, le cui dimensioni sono sensibilmente importanti per il risultato finale dell'immagine processata; le forme più comuni sono bidimensionali includendo cerchi, linee, rettangoli, quadrati e forme a diamante.

L'E*rosione* è un'operazione morfologica utilizzata per la rimozione di piccoli dettagli o per accentuare delle caratteristiche di forma utilizzando gli *elementi strutturanti*.

Matematicamente, data l'immagine binaria F e l'elemento strutturante S, l'immagine erosa (*F eroso S*) è uguale a:

$$F \ominus S = \{q \mid (S)_q \subseteq F\}$$

Ovvero la nuova immagine è l'insieme dei pixel tali che, traslando in essi S, l'intero elemento strutturante è contenuto in F.

L'effetto di erosione è dovuto al fatto che, quando l'elemento strutturante S viene traslato vicino ai bordi, esso non è completamente contenuto in F.

Sotto si riportano alcuni esempi:

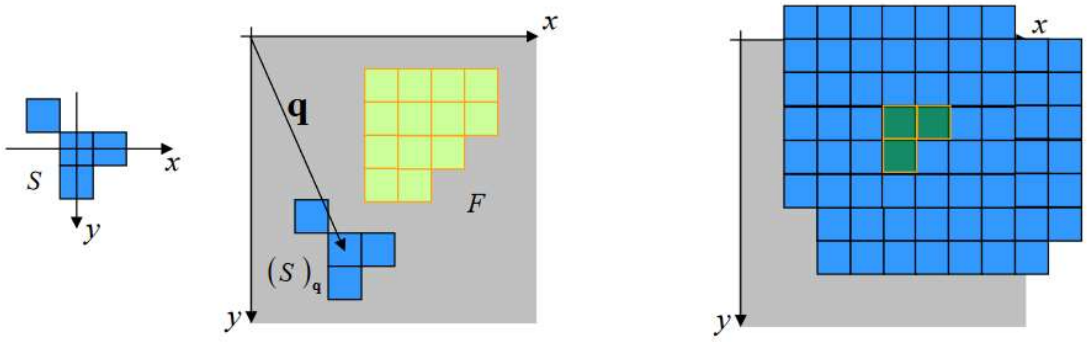


Figura 54 Esempio 1 erosione

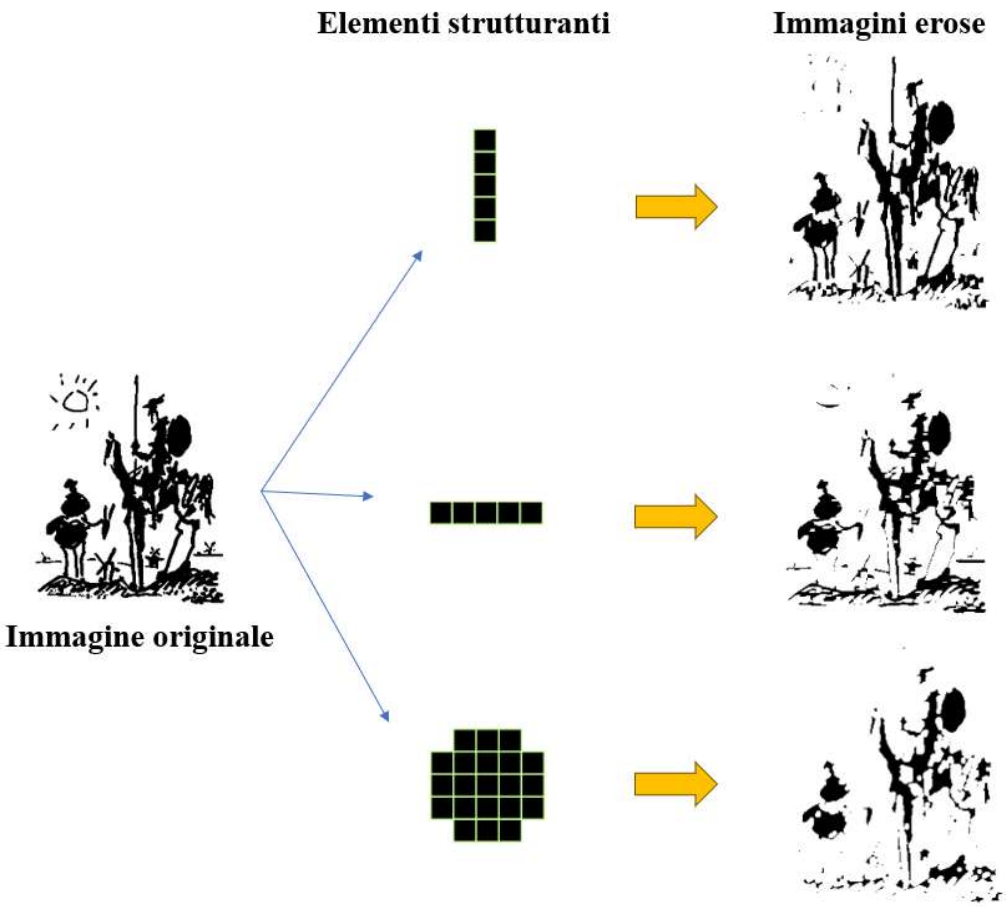


Figura 55 Esempio 2 erosione

Il prossimo esempio che si farà riguarda più da vicino l'applicazione dell'*erosione* del progetto;

Prendendo un facsimile[5] della "matrice-immagine" rappresentante il percorso, vediamo come viene modificata per mezzo dell'*erosione*.

In figura 56 si mostra la *Command Window* di Matlab in cui si è simulato, tramite comandi che svolgono le stesse operazioni, ciò che avviene nel modello Simulink del progetto.

[5] Si intende una matrice con affinità in riguardo alla disposizione degli elementi (1) rappresentanti il percorso, ma con dimensioni ridotte rispetto a quella utilizzata nel progetto (120x160).

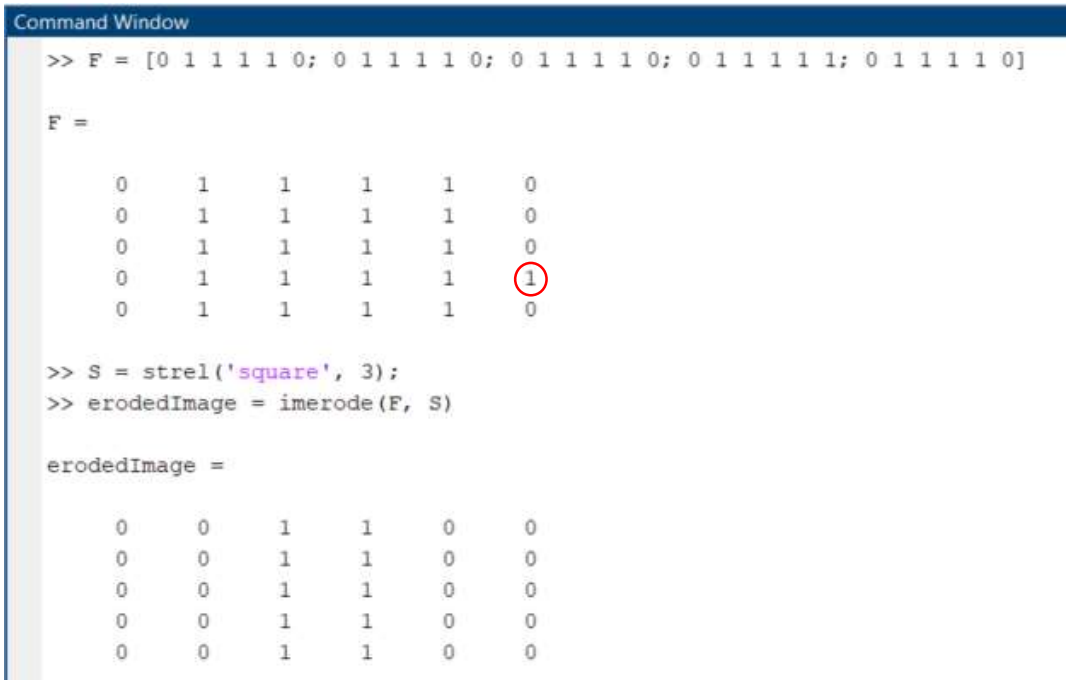


Figura 56 Erosione mediante Command Window 1

In riferimento alla figura 56, si è presa una "matrice-immagine" *F* che rappresenta la visuale del drone nel bel mezzo dell'inseguimento del percorso, in cui compare un pixel (evidenziato in rosso) per evidenziare come può apparire un eventuale errore d'elaborazione; questa viene erosa per mezzo dell'*elemento strutturante* S di forma quadrata e, dalla struttura della matrice risultante *erodedImage*, si nota come l'operazione abbia eliminato l'errore in questione evidenziando anche la forma del percorso riducendo i pixel del percorso ininfluenti in termini della caratteristica finale del percorso.

Si riporta anche il caso in cui l'elemento strutturante fosse stato di dimensioni maggiori per sottolineare l'importanza che ricopre quest'ultimo nei risultati finali;

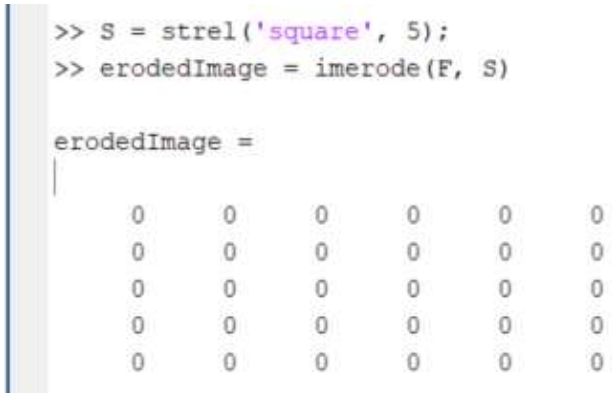


Figura 57 Erosione mediante Command Window 2

Implementazione del Path Planning

Introduzione all'algoritmo

Nel progetto *parrotMinidroneCompetition* della nostra trattazione, è presente un blocco Simulink chiamato *Path Planning* in cui è modellato l'algoritmo di controllo che permette il movimento del drone lungo il percorso.

Il termine "Path Planning" viene anche utilizzato per descrivere tutti quegli algoritmi che pianificano e controllano l'inseguimento di un percorso da parte dei dispositivi mobili su cui agiscono.

Andremo ad illustrare il modello Simulink del blocco, analizzandone l'algoritmo, approfondendo anche degli argomenti utili alla comprensione finale.

Prima però, si mostrano delle immagini della simulazione che raffigurano i vari stadi dell'inseguimento del percorso da parte del drone, così da introdurre al lettore i risvolti pratici della trattazione teorica.

Al primissimo avvio della simulazione, la vista grafica sarà quella di figura 58 in cui viene mostrato il drone a terra allineato con il percorso di prova;

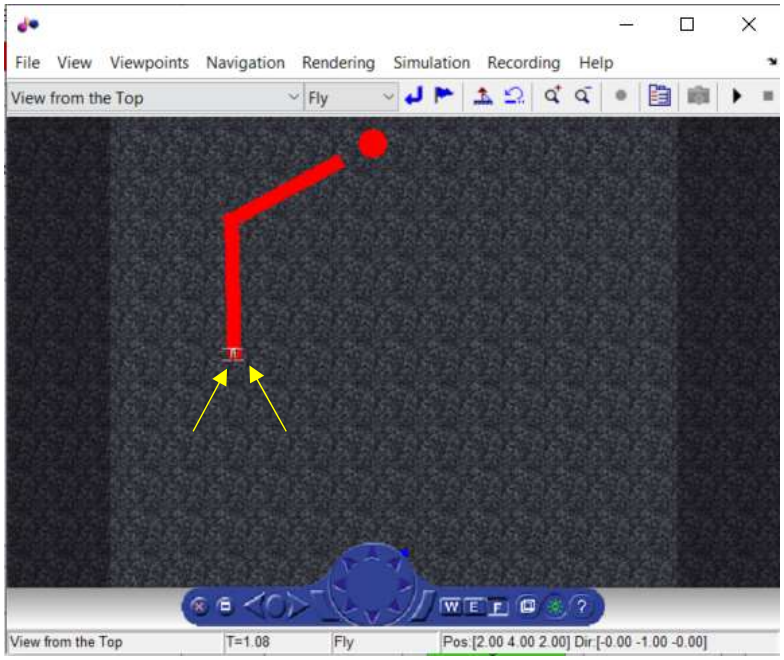


Figura 58 Drone al momento dell'avvio della simulazione

Dopo pochi secondi, il drone si solleva da terra e raggiunge la situazione di *hovering* (volo stazionario) in attesa della rilevazione del percorso;

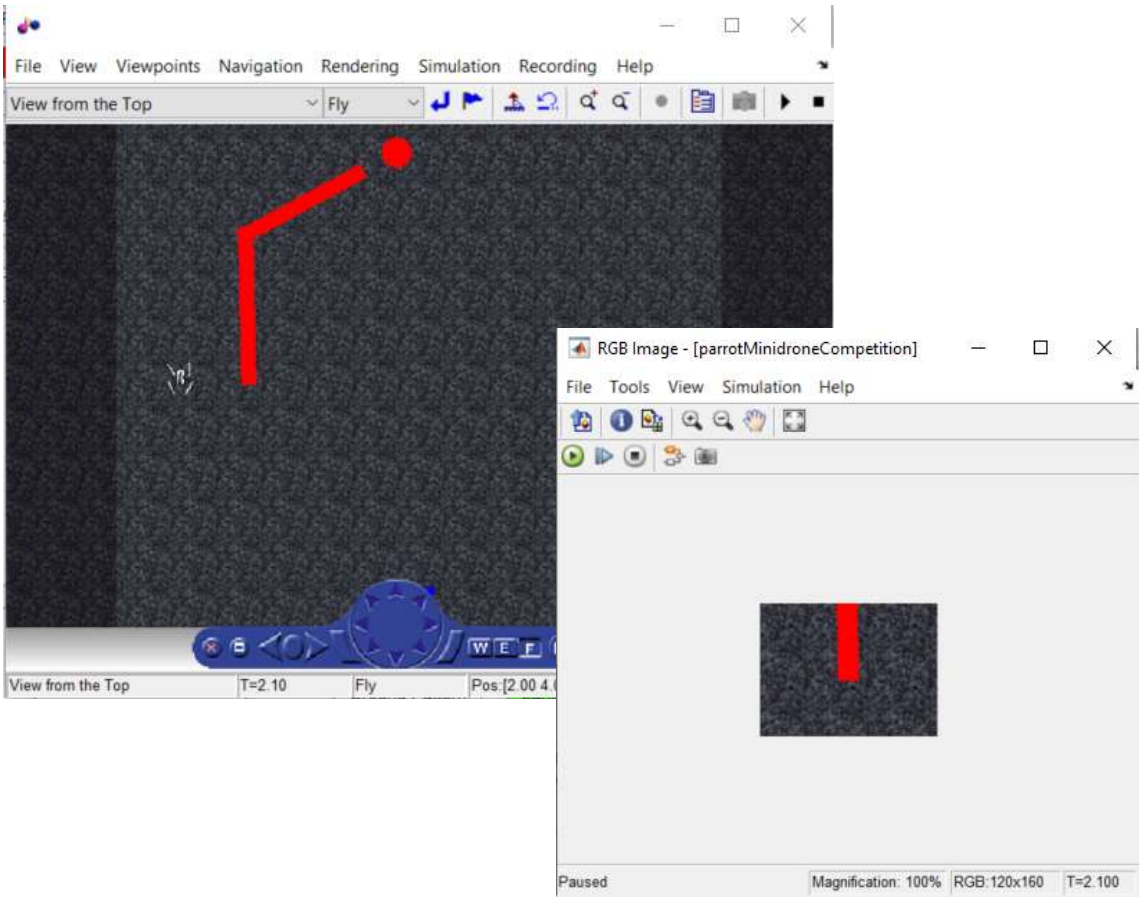


Figura 59 Prima situazione di hovering del drone

D'ora in poi, per la valutazione dell'algoritmo, si faccia sempre riferimento alla visuale della videocamera nell'apposita finestra poiché quella dell'ambiente potrebbe a prima vista trarre in inganno mostrando il drone apparentemente disallineato dal percorso per motivi di prospettiva.

Una volta rilevato il percorso, inizia l'inseguimento:

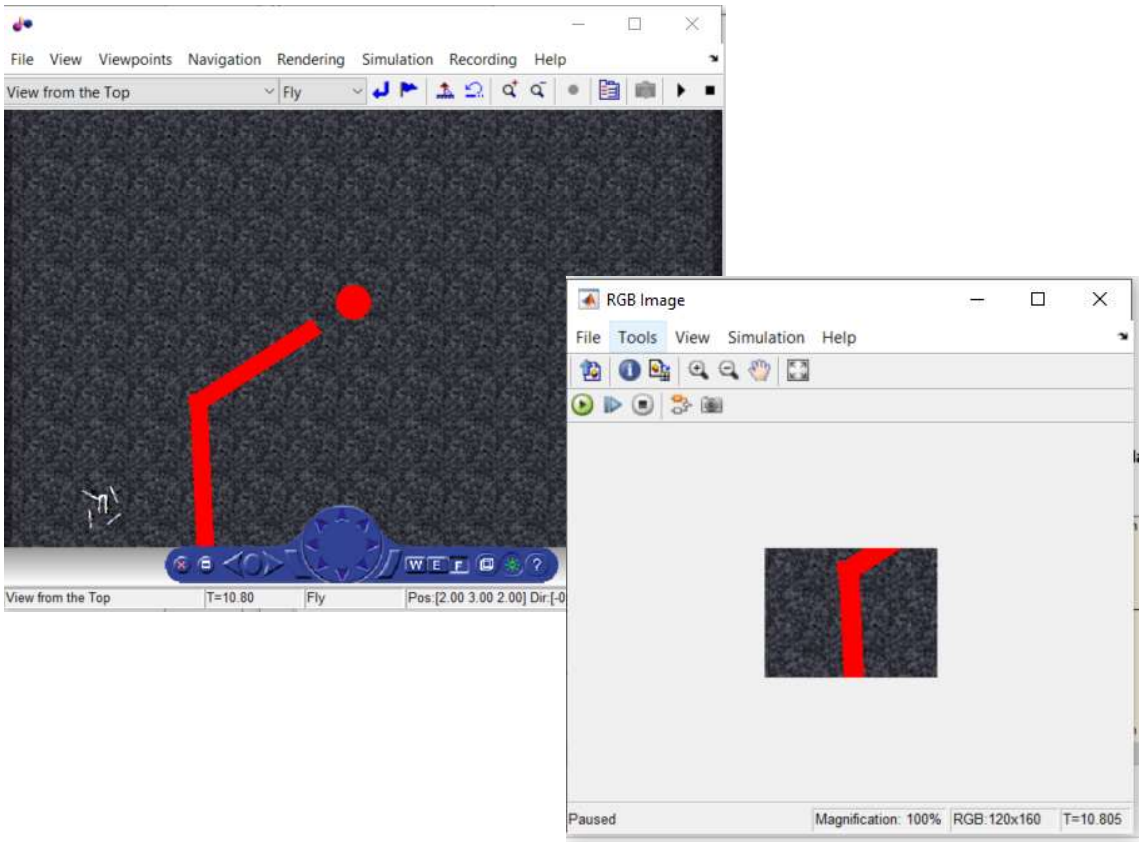


Figura 60 Stato di inseguimento del percorso

Per la prima parte rettilinea, il drone è orientato secondo il percorso; in presenza di curve (figura 61) il drone non ruota insieme al percorso, ma continua l'inseguimento traslando. Le specifiche iniziali del progetto erano quelle di farlo ruotare, ma per motivi di tempistica e per problemi con il controllo in sé, si è optato per la soluzione proposta.

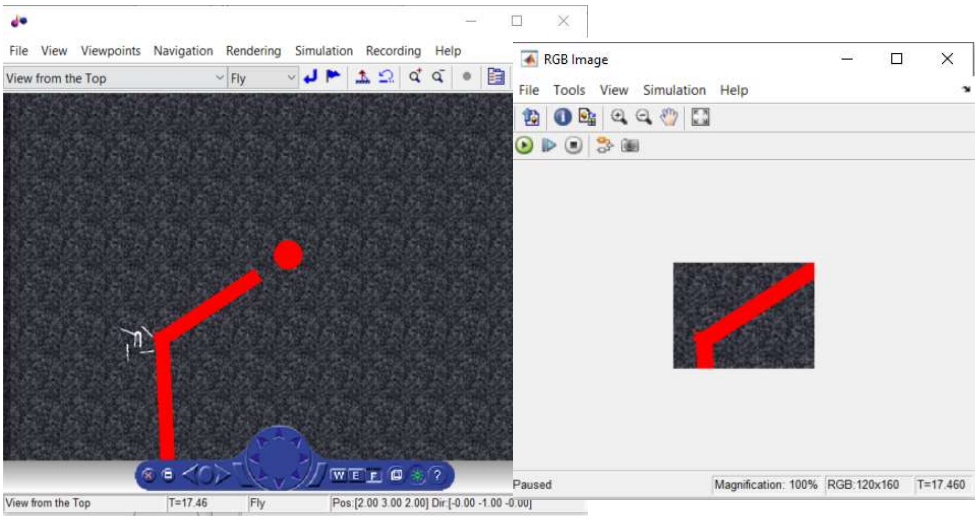


Figura 61 Situazione in curva

Il resto dell'inseguimento continuerà quindi con il drone orientato dello stesso angolo iniziale, fino al raggiungimento del punto finale di atterraggio dove procederà con la discesa.

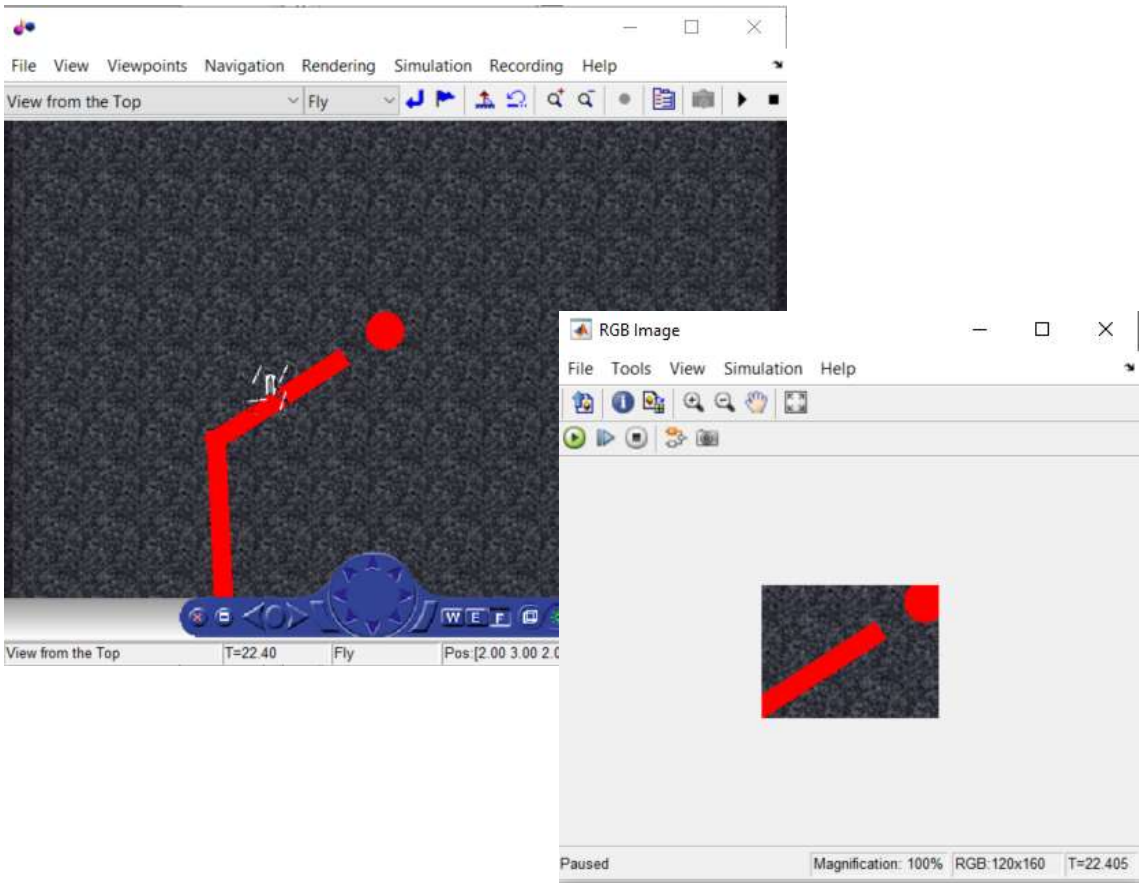


Figura 62 Inseguimento per traslazione; avvicinamento al punto d'atterraggio

Descrizione del modello Simulink

A questo punto della trattazione si dispongono delle grandezze estrapolate grazie all'elaborazione delle immagini della videocamera; tramite queste si sono ricavate le variabili di controllo X, Y e Z responsabili del movimento del drone.

Il blocco in cui si fa ciò è il *Control System,* che a sua volta si divide in più sotto blocchi, come mostrato in figura 64.

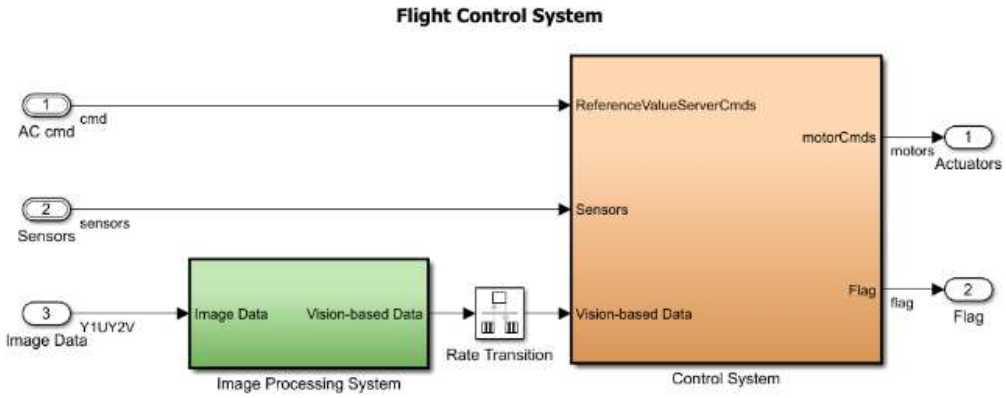


Figura 63 Flight Control System

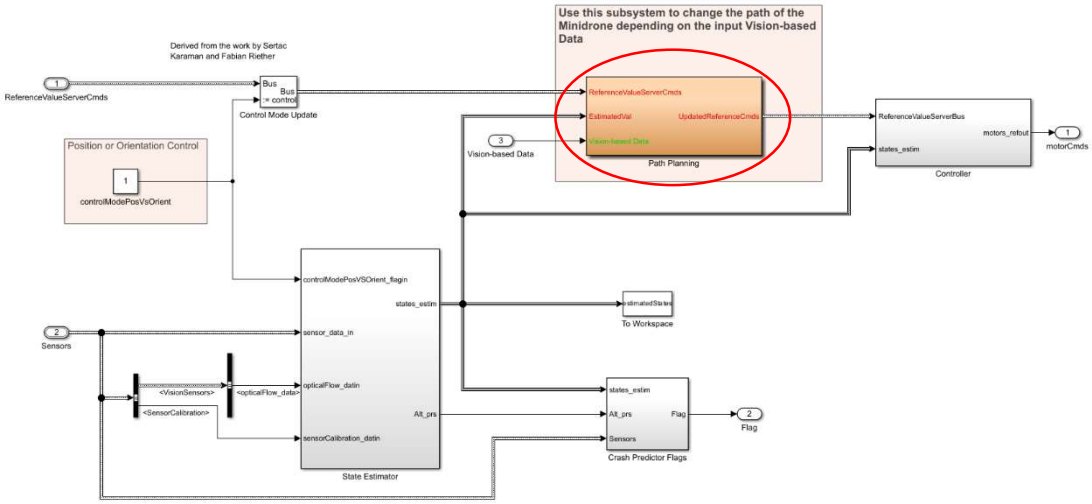


Figura 64 Control System con evidenziato il Path Planning

Il *Path Planning,* il sotto blocco evidenziato in figura 64, è responsabile dell'assegnazione delle variabili sopra citate.

Questo è stato implementato con codice utente mediante l'utilizzo di una *Matlab Function* e di uno *Stateflow chart*, un blocco della libreria Simulink che utilizza un linguaggio di programmazione grafico includendo *diagrammi di transizione di stato*, *flow charts*, *tabelle di transizione di stato* e *tabelle di verità;* lo si approfondirà nella sezione dedicata.

Il *Path Plannig* riceve quindi in ingresso l'uscita dell'*Image Processing System* (ricordiamo che è la linea dati contenente i valori dopo l'analisi del percorso*)* insieme ai dati delle stime dei sensori; esso prende la seguente forma:

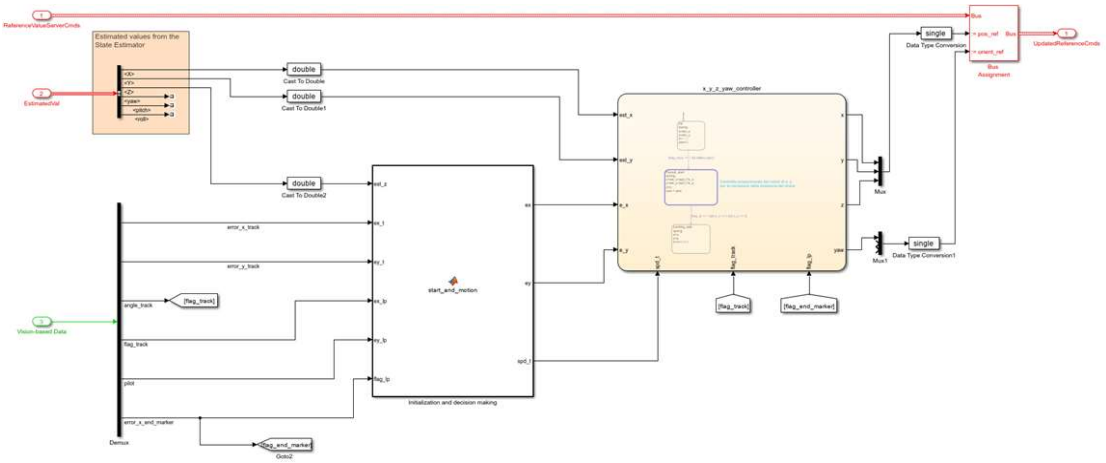


Figura 65 Path Planning

L'idea generale dell'implementazione è quella di passare, istante per istante (riferendosi all'avanzare del tempo di simulazione), i valori di X, Y e Z al *Bus Assignement*, il quale si occupa di dirigere tali valori in unico bus sotto il nome di *UpdateReferenceCmds*; questo verrà poi richiamato all'interno di *Controller* (si veda la figura 64) in cui sono presenti i controllori del "processo Drone" per l'effettivo controllo del movimento per mezzo dei motori.

I valori da assegnare alle coordinate del piano vengono determinati utilizzando un *controllo proporzionale*, che rappresenta l'essenza del calcolo degli errori svolto nell'*Image Processing System*.

Il *controllo proporzionale* è una tecnica di controllo in cui la grandezza da controllare varia in modo proporzionale all'errore tra il suo valore desiderato e il suo valore attuale.

Data la grandezza "X", la sua espressione di controllo risulta essere:

$$X = X(0) + Kp*e$$

dove:

- **X (0)** rappresenta la condizione iniziale di "X", ovvero il suo valore iniziale o stato iniziale;
- **Kp** è il coefficiente di proporzionalità che determina l'effetto dell'errore sull'uscita;
- **e** rappresenta l'errore definito come la differenza tra il valore desiderato e il valore attuale della grandezza al momento del controllo.

Entriamo ora nei dettagli del blocco per sviluppare una trattazione più specifica.

Osserviamo che, attraverso un demultiplexer, si selezionano tutti i valori del *Vision-based Data* in modo da averli disponibili per gli scopi del progetto; ciò viene fatto anche per l'*EstimatedVal,* il quale ritorna le variabili di controllo stimate dello *State Estimator*.

In un primo momento, si inizializza l'algoritmo di controllo tramite *l'Inizialization and decion making,* una *Matlab Function* che implementa la funzione *start_and_motion.*

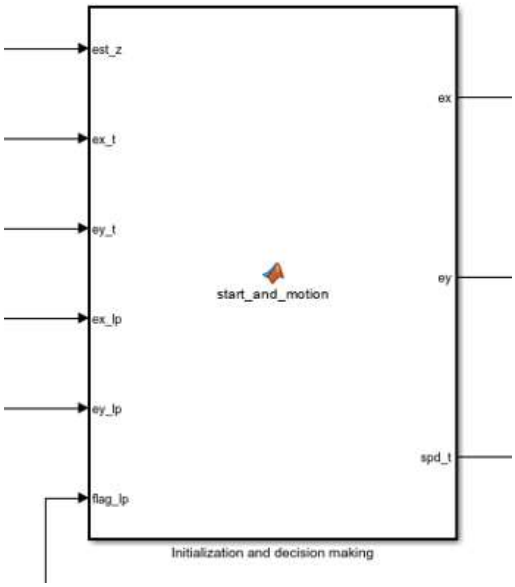


Figura 66 Initialization and decision making più da vicino

Questa ha un primo scopo di avviare l'algoritmo di controllo solo qualora il drone si sia sollevato in volo ad un'altezza di 1.1 m (gestita all'interno del *chart*) e successivamente, ad evento verificato, di assegnare alle variabili di controllo "ex" ed "ey" i valori coerenti provenienti dalla rilevazione del percorso o da quella del punto di atterraggio.

Le variabili di ingresso e di uscita della funzione sono organizzate nelle tabelle 6 e 7, dove sono anche elencati i ruoli che ricoprono all'interno della stessa;

Tabella 6 Variabili di ingresso Inizialization and decision making

Var. di ingresso	**Funzione**
est_z (Stima della z)	Verifica dell'effettiva quota di volo di 1.1 m
ex_t (errore lungo x proveniente dal *Track detector*)	"Essere pronto" all'assegnazione della relativa uscita qualora siano verificate le condizioni
ey_t	//
ex_lp (errore lungo x proveniente dal *Landing Point detector*)	//
ey_lp	//
flag_lp (flag di riconoscimento del disco d'atterraggio)	Gestione dell'assegnazione delle uscite "ex" e "ey"

Tabella 7 Variabili di uscita Inizialization and decision making

Var. di uscita	**Funzione svolta**
ex	Errore lungo x per il controllo proporzionale del drone
ey	Errore lungo y per il controllo proporzionale del drone
spd_t	*Peso* del controllo proporzionale

Sotto, se ne riporta il codice MATLAB:

```
function [ex, ey, spd_t] = start_and_motion(est_z, ex_t, ey_t, ex_lp, ey_lp, flag_lp, Flag_control_start)
% All'interno di questa funzione si eseguono due operazioni:
%   - si setta un flag quando il drone si trova nello stato di hovering
%     dato dalla sua posizione fissa ad una altezza di 1.1 metri.
%   - si passano all'algoritmo di controllo (PATH PLANNER) i valori degli errori in base
%   agli altri due stati che il drone può assumere: inseguimento percorso, fase di atterraggio
%
% Inputs:
% - z_est: Stima della coordinata z del drone data dai sensori a bordo.
% - ex_t: coordinata x dell'errore proveniente dal TRACK DETECTOR
% - ey_t: coordinata y dell'errore proveniente dal TRACK DETECTOR
% - ex_lp: coordinata x dell'errore proveniente dall' END MARK DETECTOR
% - ey_lp: coordinata Y dell'errore proveniente dall' END MARK DETECTOR
% - angle_track_prev: angolo punto VPT precedente del TRACK DETECTOR
% - flag_t: flag di cheeck percorso
% - flag_lp: fòag di check punto di atterraggio

% Parametres:
% - Flag_control_start: flag logico pre-impostato per decidere se attivare il PATH PLANNER prima o dopo la fase di decollo.
%
% Outputs:
% - ex: coordinata x errore di track/end_point a seguito della scelta in base allo stato del drone
% - ey: coordinata y errore di track/end_point a seguito della scelta in base allo stato del drone
% - spd_angle: peso controllo proporzionale dell'angolo yaw del drone
% - spd_t: peso controllo proporzionale delle coordinate x e y del drone
% - angle_track_error: errore tra l'angolo del percorso e quello del drone in movimento
% - angle_track: angolo del percorso in ogni istate di esecuzione


if Flag_control_start == 0

    if est_z < (-1.0) && est_z > (-1.2) % soglie z_low e z_high

        Flag_start_pp = 1 ; % Flag di hovering
    else

        Flag_start_pp = 0 ;

    end

else

   Flag_start_pp = 1 ;

end
if Flag_start_pp == 1 %Quando il drone è in hovering possiamo dare i comandi per l'inizio dell'inseguimento

    %Condizioni if a seconda delle quali si passano gli errori del percorso o del punto finale di atterraggio
    if flag_lp == 0

        ex=ex_t;
        ey=ey_t;

        % valore peso controllo proporzionale x,y
        spd_t = 0.004;

    else
        ex=ex_lp;
        ey=ey_lp;
        spd_t = 0.004;

    end

else

    ex=0;
    ey=0;
    spd_t = 0;

end
end
```

Figura 67 Codice del blocco Initialization and decision making che implementa la funzione start_and_motion

Prima di esaminare lo *Stateflow chart* del progetto, ne diamo una sua panoramica generale.

Lo Stateflow in Simulink

Lo *State Flow* è un *toolbox* di Matlab che permette la modellazione di una logica decisionale utilizzando *macchine a stati finiti*, *flow charts, tabelle di transizione di stato* e *tabelle di verità;*

Nei modelli Simulink, nella libreria software, si trova il blocco *charts*, il quale permette di modellare una propria logica decisionale utilizzando *macchine a stati finiti* che implementano la semantica *Mealy*, *Moore* o tempo continuo; le loro azioni ed eventi vengono espressi in codice MATLAB.

Di seguito si elencano le caratteristiche utili alla comprensione del *chart* utilizzato nel progetto.

Una volta all'interno di un modello Simulink, avendo selezionato "chart*"* dalla libreria, vedremo comparire il seguente blocco:

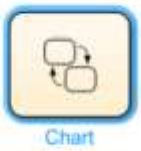


Figura 68 Prima vista di un blocco chart

L'operazione successiva sarà quella di definire i suoi *inputs* e *outputs* che serviranno alla costruzione del modello al suo interno;

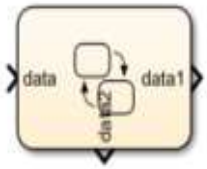


Figura 69 chart con un ingresso e due uscite

questo viene fatto intuitivamente con il mouse del pc passando vicino al blocco, fino alla comparsa del simbolo "+" che indica l'azione di connessione di un segnale.

Nella figura 69 si mostra un chart con un segnale in ingresso e due in uscita, i cui nomi sono quelli di default della prima connessione; si possono rinominare secondo i propri scopi.

Al click del *chart*, saremo "trasportati" nell'ambiente di progettazione, inizialmente vuoto.

Dal menù a tendina a fianco della schermata apparsa, si individuano i principali elementi utili alla modellazione; tra questi, ciò che ci interessa ai fini del progetto sono gli *Stati* e le *Transizioni*.

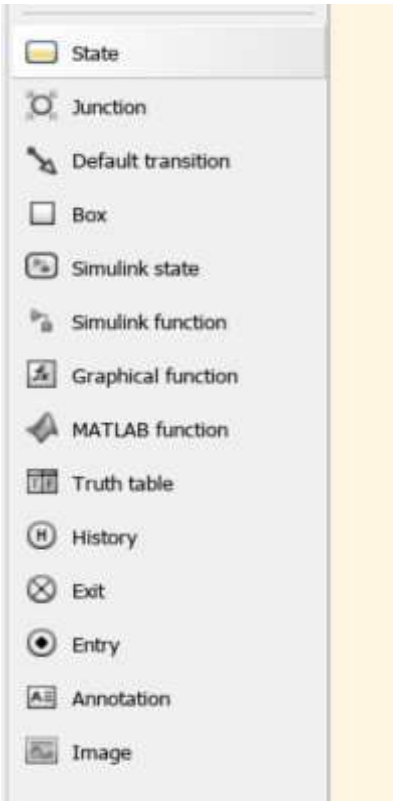


Figura 70 Elementi per la progettazione del chart

Le *Transizioni* connettono *Stati* tra loro ed ogni *Stato* contiene comandi (*Azioni*) da eseguire; si inizia quindi con l'inserire il primo stato all'interno del *chart,* il quale sarà anche quello ad essere eseguito per primo non appena il *chart* diventerà attivo insieme alla simulazione del progetto che si sta sviluppando; lo stato in questione viene notato dalla sua connessione con la *Default Transition*, quella con il "pallino" in figura 71.

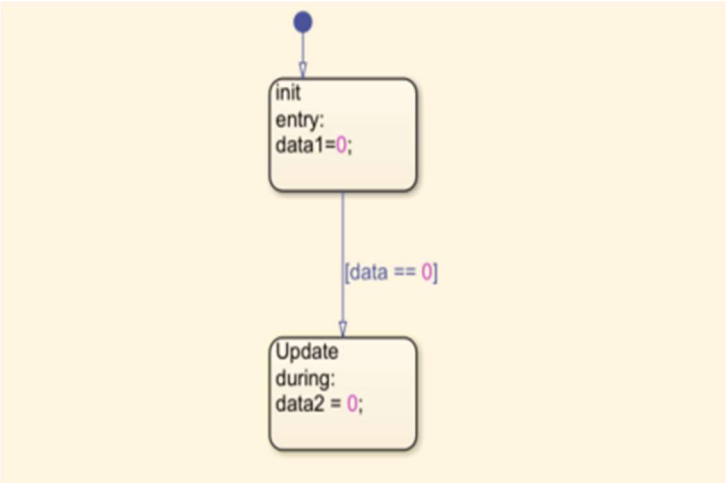


Figura 71 Due stati connessi per mezzo di una transizione (inputs e outputs in riferimento a fig. 63); Default Transition

Anche in questo caso, la connessione tra gli stati avviene in modo intuitivo: è sufficiente avvicinare il cursore del mouse allo stato di partenza desiderato per creare la transizione.

Nella figura 71 si mostra la condizione che, se verificata, attiva la transizione per il cambio di stato; a seconda della sua forma si otterrà un modello con semantica *Moore* o *Mary*.

La differenza tra l'uno e l'altro risiede nella disposizione degli *outputs*: in un caso si inseriscono nella transizione stessa, nell'altro si inseriscono all'interno dello stato entro il quale si è transitato.

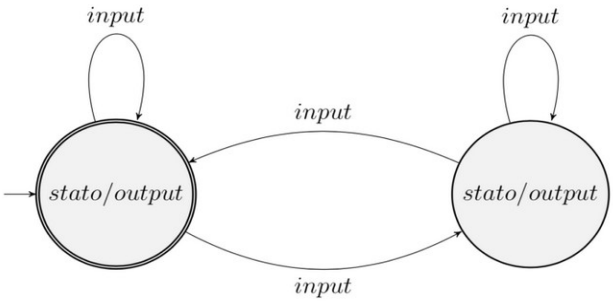


Figura 72 Macchina a stati finiti di Moore

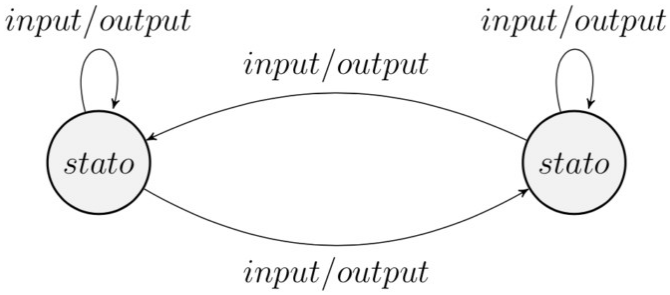


Figura 73 Macchina a stati finiti di Mary

La forma generale di una transizione è la seguente:

event_or_message[condition]{condition_action}/transition_action

la sua costruzione permetterà quindi i due modelli di diversa semantica.

- ***event_or_message***: specifica l'evento che attiva la transizione quando questo è vero ('1' logico). Si possono inserire gli operatori AND (&) e/o OR (||) per comprendere più eventi;
- ***condition***: compresa tra due parentesi quadre, serve a verificare se la transizione può essere effettuata o meno. Generalmente possono essere funzioni, variabili booleane, condizioni temporali (in tabella 8) etc.
- ***condition_action***: compresa tra due parentesi graffe, si tratta di un'operazione svolta prima del passaggio di stato, quando la condizione risulta vera ma la transizione non è considerata ancora attiva.
- ***transition_action:*** si esegue dopo che la transizione verso un nuovo stato viene dichiarata valida.

Tabella 8 Tabella degli operatori temporali

Operatore	Sintassi	Descrizione
after	after(n,E)	Restituisce *True* se l'evento E è avvenuto almeno n volte dopo l'attivazione dello stato.
before	before(n,E)	Restituisce *True* se l'evento E è avvenuto meno di n volte dopo l'attivazione dello stato.
at	at(n,E)	Restituisce *True* se l'evento E è avvenuto per l'n-esima volta dopo l'attivazione dello stato.
every	every(n,E)	Restituisce *True* ad ogni n-sima attivazione dell'evento E.
temporalCount	temporalCount(E)	Incrementa di 1 e ritorna il valore dell'evento E ad ogni sua attivazione nello stato considerato.

Sotto invece si riportano, tramite una tabella, gli operatori che consentono la gestione delle *Azioni* contenute all'interno degli *Stati.*

Tabella 9 Operatori di gestione delle azioni

Azione	Abbreviazione	Descrizione
entry	en	Consente le operazioni per una sola volta quando si entra nello stato.
exit	ex	Consente le operazioni per una sola volta quando si esce dallo stato.
during	du	Consente le operazioni finché si permane nello stato.
bind	-	Collega un evento o un dato allo stato in modo che solo questo stato possa gestire l'evento o modificare il dato.
event-name	-	Consente l'esecuzione solo se si osserva l'evento specificato.

Lo Stateflow chart del progetto

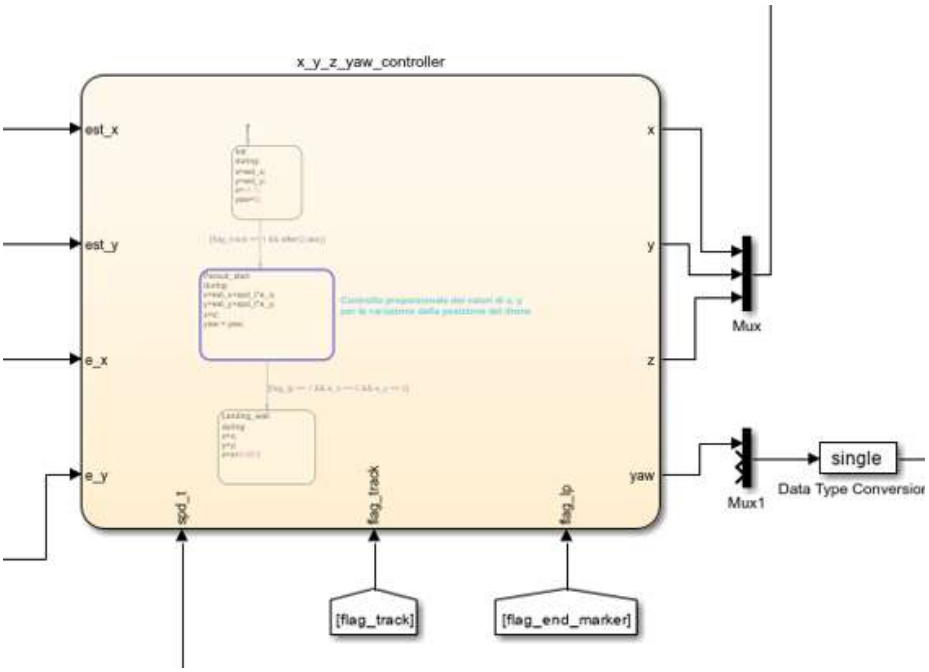


Figura 74 Chart del progetto

In figura 74 si mostra più da vicino lo *Stateflow chart* del progetto, servitoci per un'implementazione più diretta ed intuitiva del codice d'assegnazione delle variabili che si controllano.

Vediamo che il blocco ha come ingressi i segnali provenienti delle stime di X e Y (*est_x, est_y*), i valori delle coordinate degli errori nel piano x-y assegnati istante per istante dall'*Initialization and decision making* (*ex, ey*) e i flag segnalativi del percorso e del disco di atterraggio, rispettivamente *flag_track* e *flag_lp*.

Le uscite sono le grandezze X, Y, Z e l'angolo YAW del "processo Drone".

La modellazione interna al blocco rappresentata da una macchina a stati finiti di *Moore*, viene raffigurata in figura 75:

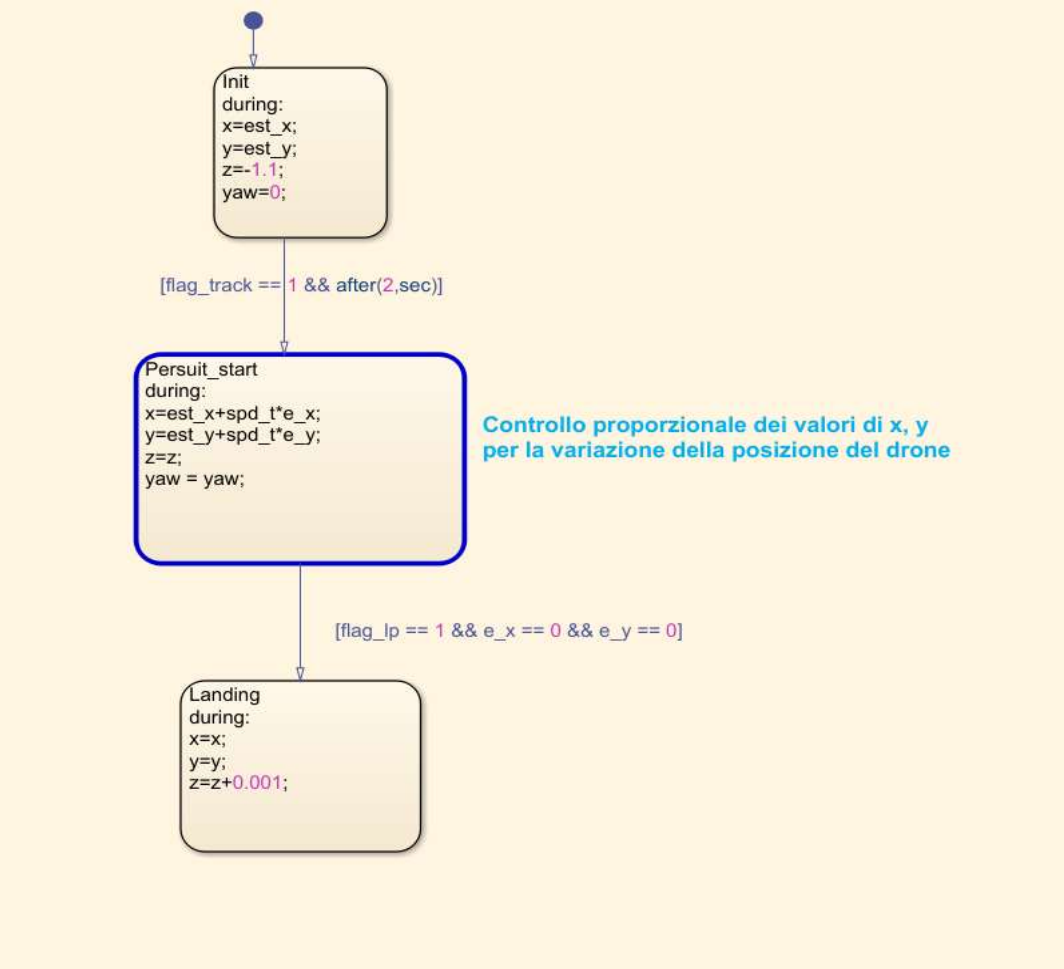


Figura 75 Modellazione del chart

Osserviamo la presenza di tre stati, ognuno dei quali rappresenta una condizione di volo:

- ***Init***: è lo stato in cui si inizializzano le variabili X e Y secondo la posizione iniziale del drone all'interno dell'ambiente virtuale (in modo che si sollevi in volo sopra l'inizio del percorso), dove si invia il comando per il sollevamento in volo alla quota di 1.1 m e dove si setta l'angolo YAW a 0 (radianti). È il primo stato ad essere eseguito dopo l'avvio del chart (si ricorda la *Default Transition*);
- ***Persuit_start***: dopo la rilevazione del percorso e dopo che sono passati almeno due secondi, si transita nel presente stato responsabile del movimento del drone lungo il percorso. Qua si implementa il *controllo proporzionale* delle variabili X e Y per tutta la durata della permanenza (si ricordano le azioni di tipo *During*).
  I valori di Z e YAW restano invariati.

Si resta in questo stato anche alla fine del percorso, in prossimità del punto di atterraggio, permettendo al drone di disporsi al centro del disco, dopo che i valori degli errori passati sono relativi all'analisi della "matrice-immagine" dell'*End Marker*;

- ***Landing***: è il blocco che cura la fase di atterraggio. Una volta che il drone si trova in prossimità del centro del disco segnalato dai valori nulli degli errori (i punti del centro della videocamera e quello del disco corrispondono) si procede alla discesa del drone variando la Z e mantenendo costanti i valori di X e Y raggiunti.

I valori di uscita vengono poi immessi nel bus *UpdetedReferenceCmds* in modo poterli richiamare all'interno del blocco *Controller* per l'effettivo controllo "fisico" tramite i sistemi di controllo.

Vedremo l'implementazione dei controllori del progetto nel prossimo capitolo.

Tecniche di controllo del Processo Drone

Generalità sui sistemi di controllo

Un sistema di controllo è un apparato che risponde ad un *problema di controllo*, ossia l'azione o l'insieme delle azioni volte a far assumere a una grandezza fisica un valore o un andamento temporale desiderato.

Si pensi ad esempio al controllo della climatizzazione di un edificio, della portata e pressione di una caldaia o ad esempio al controllo della velocità di un motore elettrico;

l'oggetto del controllo prende il nome di *Processo*, e le sue grandezze che si vogliono controllare prendono il nome di *variabili controllate*.

Idealmente si vuole che le *variabili controllate* coincidano con degli andamenti temporali da noi desiderati, detti *segnali di riferimento*, avendo supposto la possibilità di aggiungere delle variabili esterne al processo, le *variabili di controllo*, in modo da riuscire nell'intento descritto.

Nella realtà, le *variabili controllate* non dipendono esclusivamente da quelle di controllo o dai *segnali di riferimento*, ma anche da altri ingressi non manipolabili dall'utente che prendono il nome di *disturbi*.

Il problema di controllo si trasforma quindi nel riuscire a condizionare le variabili del *Processo* da controllare in modo che esse si avvicinino il più possibile a valori da noi prestabiliti; a tal proposito compare un'altra definizione che ricorre nella teoria dei sistemi: quella dell'*errore*.

L'*errore* è la differenza tra il *segnale di riferimento* e la *variabile controllata* e in base al sistema che si studia, saranno preventivate delle *specifiche* per rendere l'errore "accettabilmente piccolo" in accordo con la natura del processo da controllare.

Per rispondere a queste specifiche si svolge la *Sintesi del Controllore*, ovvero alla progettazione di quel blocco in ingresso al *Processo* che ne modifica il comportamento.

La connessione del *Controllore* con il *Processo* da luogo al *sistema di controllo* ed esso può essere di due tipologie:

- **A catena aperta**, in cui l'azione di controllo (l'uscita del *Controllore*) è determinata in base all'entità dei disturbi, ammesso che questi siano misurabili;
- **A catena chiusa o in controreazione**, dove l'azione di controllo si basa sul confronto fra l'uscita del sistema (*variabile controllata* in uscita dal *Processo*) e l'ingresso (il *segnale di riferimento*).
  È con questo approccio che si mira all'annullamento dell'*errore*, tenendo conto di tutte le possibili cause che possono intromettersi negativamente tra il valore desiderato e la variabile da controllare; questo processo è comunemente noto come *feedback* dell'uscita.

Il sistema di controllo maggiormente utilizzato nella teoria dei sistemi è quello *a catena chiusa*, di cui ne vediamo rappresentato un generico schema a blocchi in figura 76;

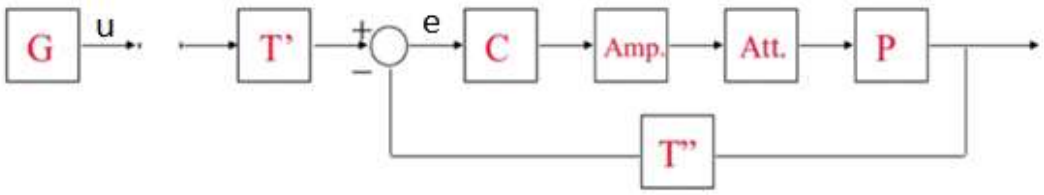


Figura 76 Schema di controllo in controreazione

I blocchi fondamentali sono:

- ***G***: è il blocco per la generazione del segnale di riferimento **u** in ingresso al sistema (comunemente di natura fisica come temperatura, pressione, velocità etc);
- ***T' e T''***: sono i *Trasduttori* del sistema, ovvero blocchi per la trasduzione dei segnali; trattano la conversione di segnali da natura fisica in elettrica (*T'*) e viceversa (*T''*);
- ***C***: è il *Controllore* per l'azione di controllo (comunemente un algoritmo software);
- ***Amp.***: il blocco per l'amplificazione del segnale di controllo per il corretto interfacciamento con l'*Attuatore* del sistema;
- ***Att.***: l'*Attuatore*; svolge la funzione di attuare l'azione di controllo sul *Processo*. Possono essere ad esempio motori elettrici;
- ***P***: *Processo*, ovvero l'oggetto del controllo.

Lo studio del sistema, e la sua successiva rappresentazione a blocchi, si basa dapprima sullo studio del modello matematico del *Processo* nella variabile temporale **t**;

- Sistemi lineari e stazionari a dimensione finita:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{P}\boldsymbol{z}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) + \boldsymbol{Q}\boldsymbol{z}(t) \end{cases}$$

Figura 77 Modello matematico processo lineare e stazionario

per poi passare nel dominio **s=jω** per mezzo della *Trasformata di Laplace*:

- Trasformata di Laplace:

$$\begin{cases} \boldsymbol{x}(s) = \boldsymbol{\Phi}(s)\boldsymbol{x}(t_0) + \boldsymbol{H}(s)\boldsymbol{u}(s) + \boldsymbol{H}_z(s)\boldsymbol{z}(s) \\ \boldsymbol{y}(s) = \boldsymbol{\Psi}(s)\boldsymbol{x}(s) + \boldsymbol{W}(s)\boldsymbol{u}(s) + \boldsymbol{W}_z(s)\boldsymbol{z}(s) \end{cases}$$

Figura 78 Processo dopo la trasformata di Laplace

Si fa notare al lettore che il *Processo* è esso stesso un *sistema*, ovvero "un'interconnessione di componenti che lavorano insieme per raggiungere un obiettivo specifico".

Viene effettuato il passaggio nel dominio di *Laplace* per semplificare i passaggi matematici nella *sintesi del controllore*, l'altro importante passaggio per il dimensionamento del sistema di controllo ricercato.

Il modello del sistema di controllo in controreazione nel dominio della frequenza diventa quello raffigurato in figura 79;

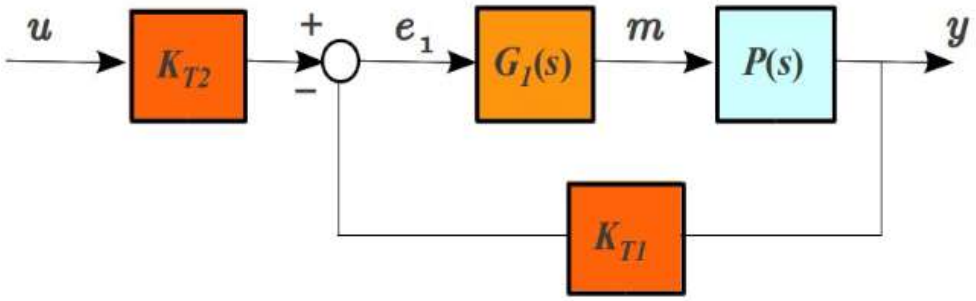


Figura 79 Sistema di controllo nel dominio di Laplace

I *Trasduttori* vengono rappresentati come delle costanti (data la trasduzione di natura proporzionale) mentre i blocchi relativi all'amplificazione e all'attuazione vengono integrati, rispettivamente, all'interno del *Controllore* e del *Processo*.

Il sistema di figura 74 può essere riscritto nella seguente forma,

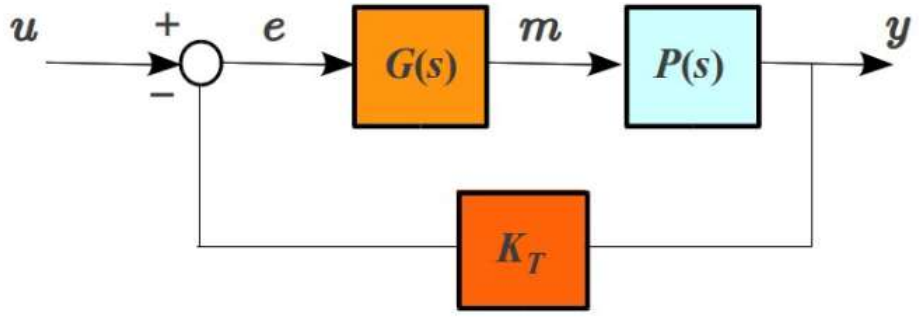


Figura 80 Sistema di controllo "riscritto"

in cui comparare un solo trasduttore di valore $K_T = \frac{K_{T2}}{K_{T1}}$

Questa è la forma più utilizzata per la semplificazione matematica che ne deriva.

Controllori PID

Di seguito si espongono i controllori *PID* utili alla comprensione delle parti successive della trattazione;

I controllori *PID* (*Proportional-Integrative-Derivative*) sono una tipologia di controllori largamente utilizzati nel campo dell'automazione nei *Processi* industriali, in quanto hanno una semplice struttura.

Sono controllori predefiniti, ovvero il progettista, agendo su determinati parametri, ne regola l'azione di controllo in base alle proprie esigenze.

Accettando in ingresso l'*errore*, sviluppano tre azioni di controllo: *proporzionale*, *integrativo* e *derivativo*.

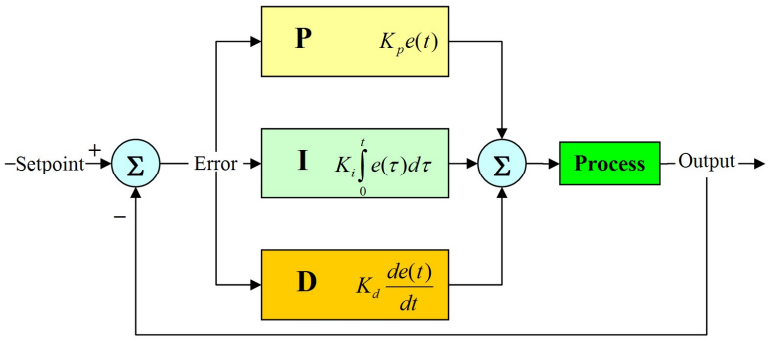


Figura 81 Sistema di controllo con controllore PID

Si possono anche trovare varie configurazioni di PID, come ad esempio PI o PD, dove risultano nulli i contributi, rispettivamente, derivativo e integrativo.

- ***Il controllo proporzionale*** fornisce un'uscita proporzionale all'errore *e*.
  è possibile regolare, utilizzando solo quest'ultimo, processi talvolta instabili; tuttavia, non è possibile regolare un'azione di controllo che garantisca un errore nullo: questo perché l'azione U_P è possibile sole se *e* è diverso da zero.
- ***Il controllo integrativo*** fornisce un'uscita proporzionale all'integrale dell'errore (quindi proporzionale al suo valor medio).
  Tale controllo fa sì che il controllore abbia memoria dei valori passati del segnale d'errore; ciò significa che l'uscita integrale non sia necessariamente nulla se è nullo l'errore, fornendo così il contributo necessario per portare il processo al

riferimento richiesto senza il quale, con la sola azione proporzionale, non sarebbe possibile.

- ***Il controllo derivativo*** fornisce un'uscita proporzionale alla derivata di e(t).
  Viene utilizzato per la pronta compensazione delle variazioni del segnale errore: all'aumentare di quest'ultimo, l'azione derivativa cerca di compensare rapidamente la variazione in ragione della sua velocità di cambiamento senza aspettare che l'errore diventi significativo (azione proporzionale) o che persista per un certo tempo (azione integrale).
  L'azione derivativa viene però spesso tralasciata in quanto può rendere il PID troppo sensibile in quelle applicazioni dove l'errore può variare in modo "brusco".

L'uscita generale del *PID*, in ingresso al Processo, assume dunque la forma:

$$u(t) = K_P\, e(t) + K_I \int_0^t e(\tau)\, d\tau + K_D \frac{de(t)}{dt}$$

dove:

K_P = guadagno proporzionale

K_I = guadagno integrale

K_D = guadagno derivativo

Un altro modo di riscriverla, molto più utilizzato, è il seguente:

$$u(t) = K_P \left(e(t) + \frac{1}{T_I} \int_0^t e(\tau)\, d\tau + T_D \frac{de(t)}{dt} \right)$$

in cui:

$T_I = \frac{K_P}{K_I}$ è detto tempo integrale

$T_D = \frac{K_D}{K_P}$ è detto tempo derivativo

Nel dominio di Laplace, la legge di controllo del *PID reale* si ottiene filtrando l'azione derivativa:

$$U(s) = K_P(1 + \frac{1}{T_I s} + \frac{T_D s}{1 + s\frac{T_D}{N}})E(s)$$

ottenendo così una legge fisicamente realizzabile.

La sintesi dei controllori *PID* si basa sulla scelta dei parametri delle tre azioni di controllo, ovvero quelli dei guadagni delle tre leggi matematiche.

Per far ciò, si utilizzano leggi empiriche, come ad esempio le *tecniche di Ziegler-Nichols*;

quest'ultime risalgono al 1942 e si basano sulla determinazione del *guadagno critico*, valore dal quale si deducono i valori dei parametri ricercati.

I passi da seguire sono i seguenti:

- Si controlla il *Processo* interessato con la sola azione proporzionale (i contributi integrativo e derivativo vengono impostati a 0);
- Si aumenta gradualmente il guadagno K_P;
- Si determina il valore del guadagno K_U in modo che la variabile controllata presenti oscillazioni sostenute che persistano nel tempo, senza scomparire dopo un periodo transitorio.
- Si registra il periodo critico P_U di tali oscillazioni;
- Si determinano i parametri per il controllore P, PI o PID seguendo la tabella 10.

Tabella 10 Metodo Ziegler-Nichols

Tipo	**K_P**	**τ_i**	**τ_d**
P	$0{,}5K_P$	-	-
PI	$0{,}45K_U$	$P_U / 1{,}2$	-
PID	$0{,}6K_U$	$P_U / 2$	$P_U / 8$

Controllori del progetto originale

All'apertura del progetto *parrrotMinidroneCompetition*, i controllori implementati sono contenuti all'interno del blocco *Control System* del *Flight Control*;

Il sotto blocco interessato è il *Controller*, evidenziato in figura:

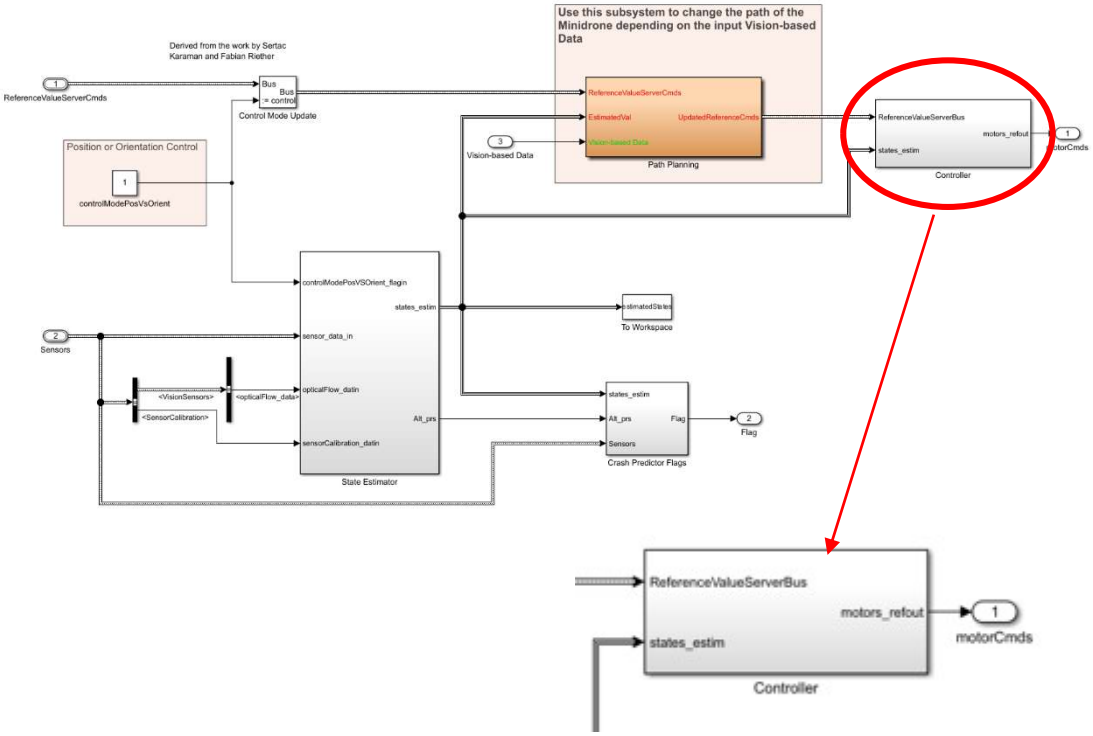


Figura 82 Blocco Controller in risalto

esso prende in ingresso il bus dati proveniente dal blocco *Command (*segnali di riferimento*)* e quello relativo alle stime del blocco *State Estimator*.

Il bus d'uscita *motorCmds* contiene le azioni di controllo dei controllori, in modo che le *variabili controllate* siano più fedeli possibile a quelle desiderate.

Vediamo la relazione interna che lega l'uscita con gli ingressi;

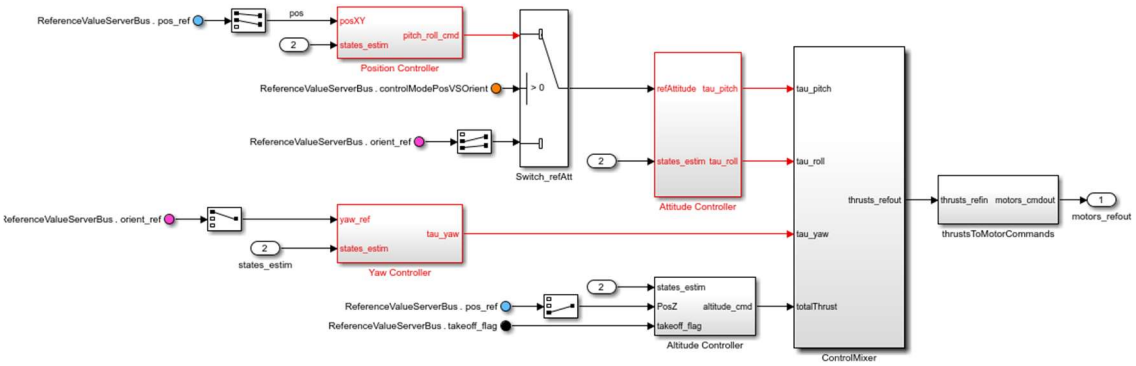


Figura 83 Blocco Controller

Il modello Simulink del blocco è riportato in figura 83, in cui si distinguono i due blocchi fondamentali utili alla trattazione, l'*Attitude Controller* e lo *Yaw Controller*.

Essendo gli spostamenti lineari del drone lungo gli assi X e Y controllati dagli spostamenti angolari attorno ai medesimi, l'*Attitude Controller* è "l'incaricato" del controllo dei movimenti di Pitch e Roll implementando i relativi controllori;

Si è riportato lo *Yaw Controller* (non vedremo la sua composizione) per indicare l'importanza che avrebbe assunto qualora si fosse progettato un algoritmo che includeva la rotazione del drone in corrispondenza delle curve del percorso.

Per un richiamo, si riporta la figura 84 che rappresenta un'illustrazione del drone compresa dei movimenti angolari per eseguire gli spostamenti lineari:

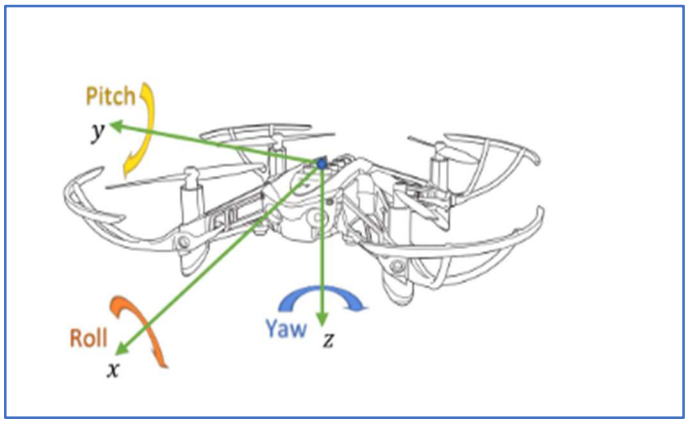


Figura 84 Movimenti di Pitch, Roll e Yaw in accordo con il sis. di riferimento solidale

Lo schema di controllo iniziale dell'*Attitude Controller* era il seguente:

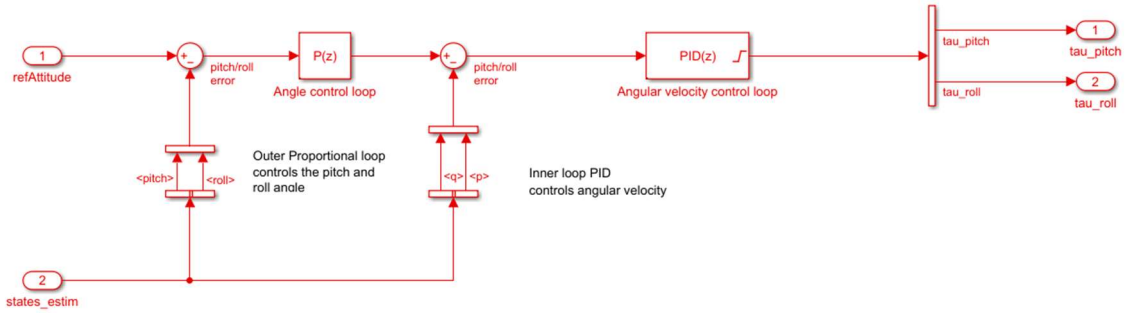


Figura 85 Schema di controllo iniziale di Pitch e Roll

in cui si osserva che implementava un controllore *PID* sulla base dell'*errore* tra i valori dei segnali di riferimento interessate (*refAttitude*) e quelli delle variabili indicative dello stato del drone (*states_estim*).

Le *variabili controllate tau_pitch* e *tau_roll* saranno poi richiamate nella sezione relativa alla visualizzazione grafica del drone, in modo da osservare il suo movimento lungo il percorso.

Il comportamento dinamico che assumeva il sistema viene rappresentato dalla figura 86, in cui vengono mostrate le *variabili controllate Pitch* e *Roll* in funzione dei valori di X e Y dell'algoritmo di *Path Planning* durante l'inseguimento di un percorso di prova;

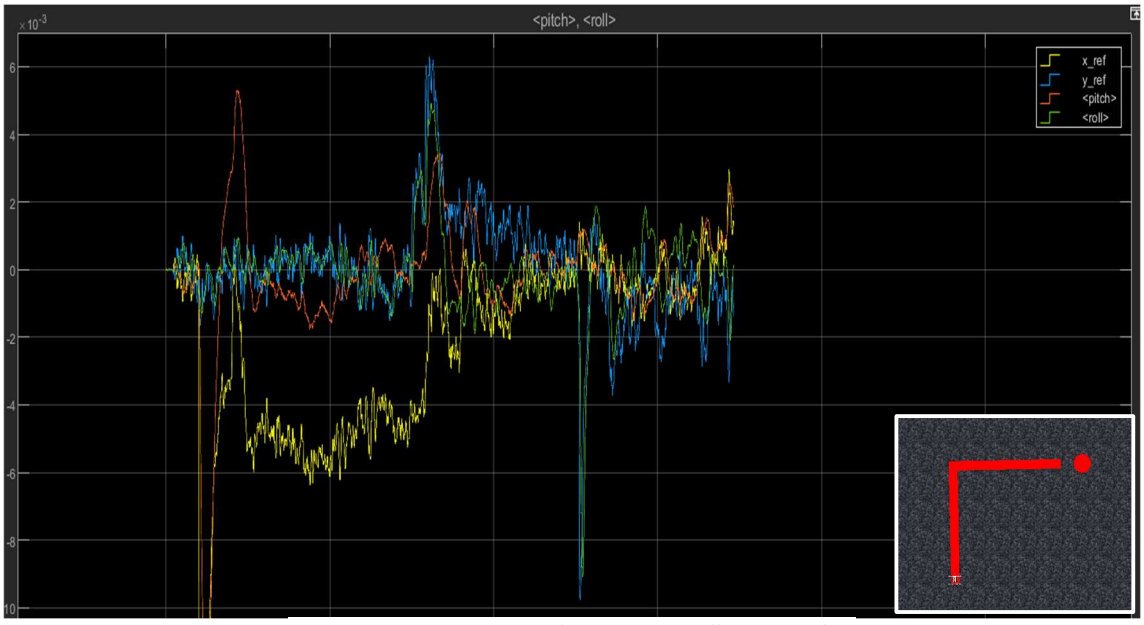


Figura 86 Comportamento dinamico controllori originali

Le sovra elongazioni accentuate sono relative agli elevati sforzi di controllo dovuti alla poca “precisione” dei controllori e alla curva di metà percorso da compiere.

Controllori dopo le sintesi

Osservando il comportamento dinamico originale, ci si è chiesto se sarebbe stato possibile ottenere una migliora risposta sintetizzando dei propri controllori attraverso delle *sintesi in frequenza* o con *luogo delle radici*.

Dall’elaborato “Gizzarone Manuel, Studio e sviluppo di sistemi di controllo lineari per l’assetto di droni”, in cui il caso di studio era sempre il controllo del *Parrot Minidrone*, si sono implementati i controllori proposti per l’angolo di *Pitch* e *Roll* in sostituzione dei primi.

Il nuovo comportamento ottenuto è il seguente (basato sullo stesso percorso di prova iniziale):

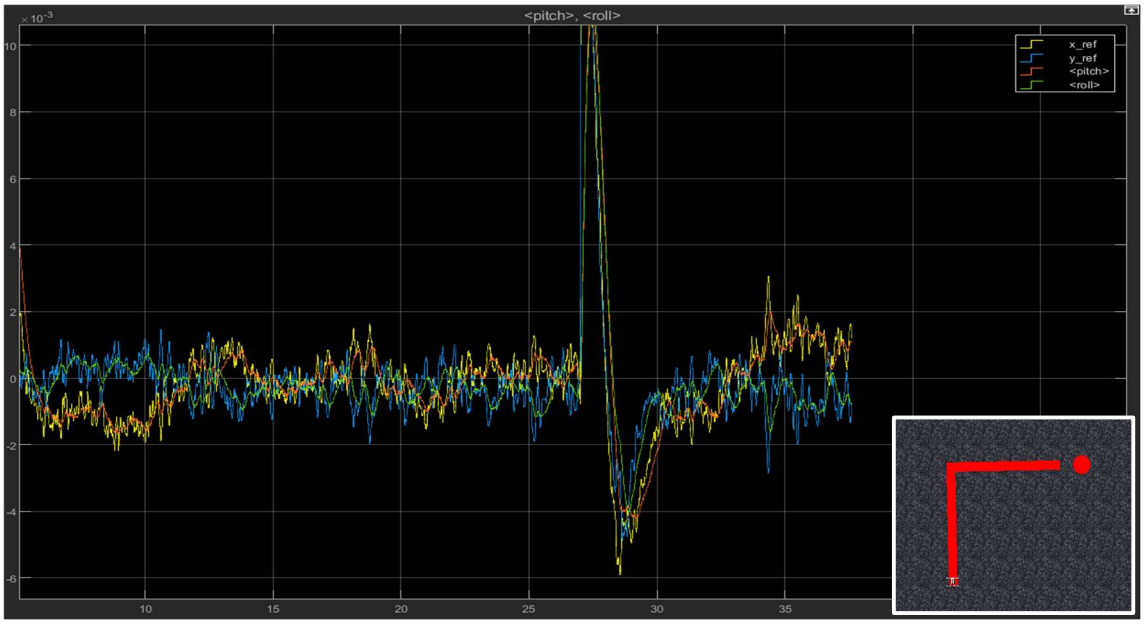


Figura 87 Comportamento dinamico del nuovo sistema

Dal confronto si nota come i valori delle grandezze risultano essere più ravvicinate nell’ordine d’ampiezza comunicato.

Nella pagina successiva si riporta lo schema di controllo del modello Simulink ottenuto:

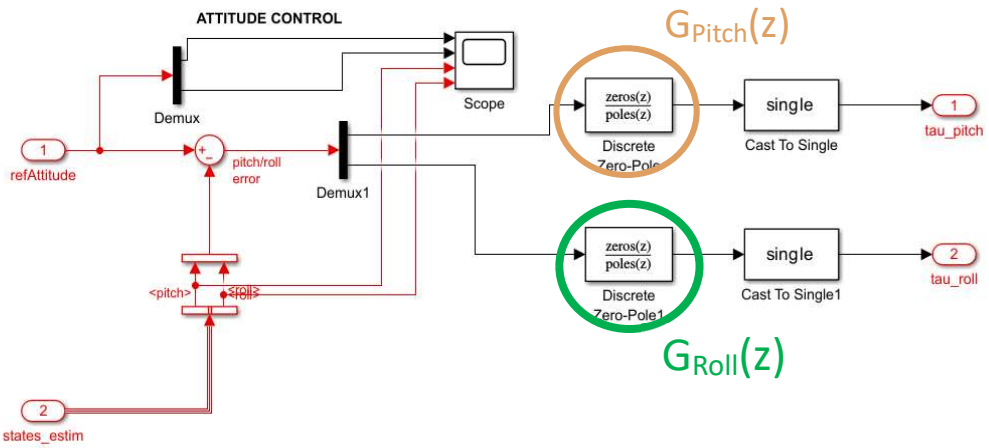


Figura 88 Schema di controllo a seguito delle sintesi

I controllori di *Pitch* e *Roll* nella variabile "z" [6] sono:

$$G_{roll}(z) = \frac{0.018113(z-0.9987)(z-0.9909)}{(z-1)(z-0.8521)}$$

$$G_{pitch}(z) = \frac{0.02(z-0.987)}{(z-0.7408)}$$

Figura 89 Controllori "Gizzarone Manuel, Studio e sviluppo di sistemi di controllo lineari per l'assetto di droni"

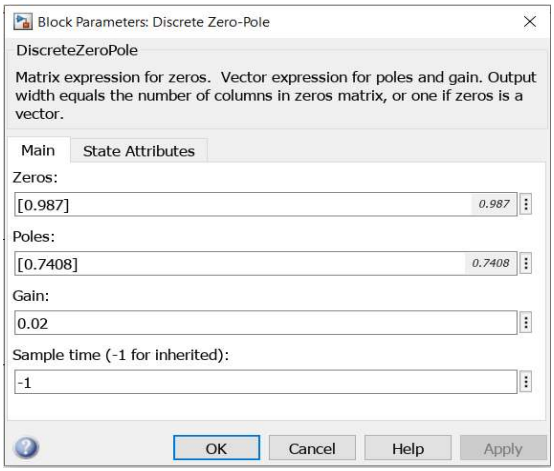


Figura 91 Blocco Simulink del controllore di Pitch

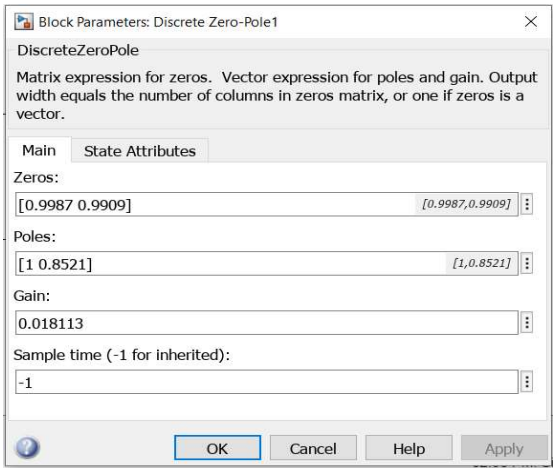


Figura 90 Blocco Simulink del controllore di Roll

[6] Si precisa che dopo la sintesi in "s" è stata necessaria una *discretizzazione* per riportare i controllori nelle stesse variabili del modello ("z", ad indicare la natura discreta del progetto).

Le sintesi utilizzate dall'elaborato di riferimento sono, rispettivamente, quella della *sintesi in frequenza* per il controllore dell'angolo di *Roll* e la *sintesi per luogo delle radici* per il controllore dell'angolo di *Pitch*; sotto si riportano le specifiche che si sono imposte:

- **Specifiche univoche**
  - Tipologia di sistema desiderato: sistema di tipo $k = 1$
  - Condizione sull'errore di inseguimento a regime: $|e_1(t)| \leq 0.01$
  - Astatismo rispetto a disturbi costanti agenti in catena diretta in uscita dal processo $P(s)_{roll}$
- **Specifiche lasche**
  - Modulo alla risonanza: $M_r \leq 2dB$
  - Banda passante: $B_3 \approx 0.5Hz$

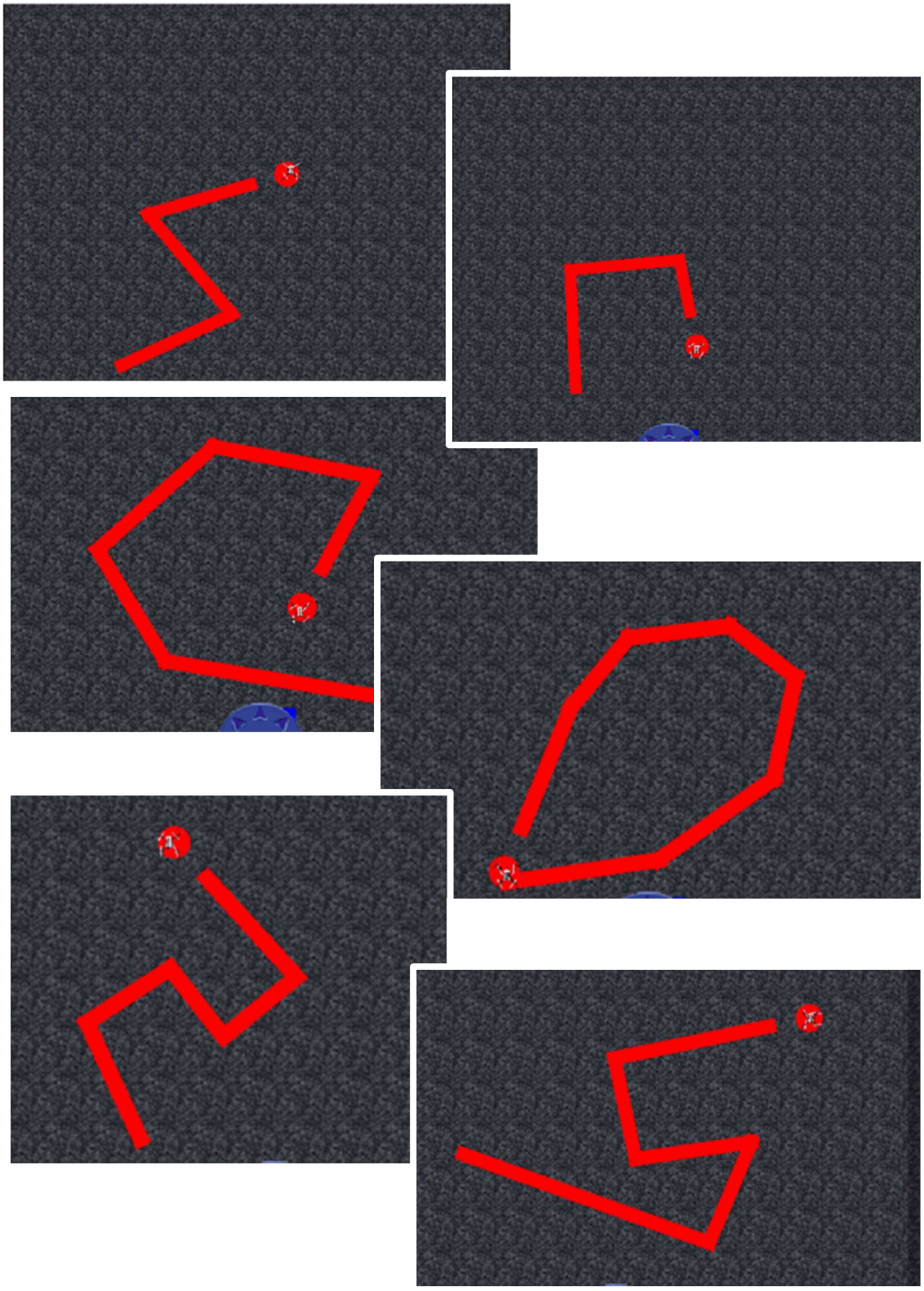
Figura 93 Specifiche sintesi $G_{Roll}(s)$

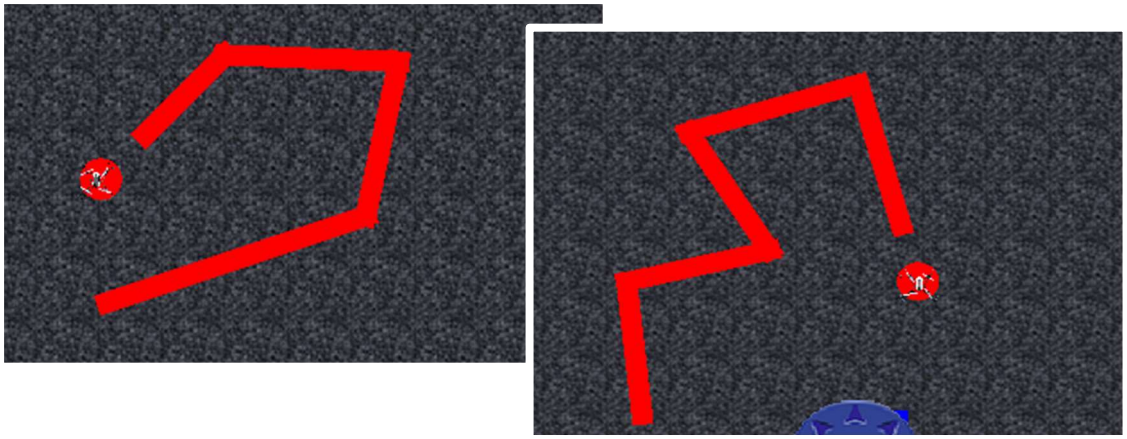
- **Specifiche univoche**
  - Tipologia di sistema desiderato: sistema di tipo $K = 1$
  - Astatismo rispetto disturbi costanti agenti in catena diretta in uscita dal processo $P(s)_{pitch}$.
- **Specifiche lasche**
  - i poli p_i della funzione di trasferimento in catena chiusa devono avere tutti parte reale minore di -1, ovvero $\Re(p_i) \leq -1 \quad \forall i$

Figura 92 Specifiche sintesi $G_{Pitch}(s)$

Alcuni percorsi della simulazione

Arrivati a questo punto della trattazione, si possono mostrare alcuni percorsi utilizzati per le prove dell'algoritmo di inseguimento.





Tutti i percorsi mostrati sono stati tracciati utilizzando l'app *Track Builder;* essa consente di raffigurare e di caricare all'interno dell'ambiente virtuale, percorsi di proprio interesse utili per i test degli algoritmi implementati.

L'app è disponibile solo all'interno dei progetti *parrotMinidroneCompetition*, raggiungibile all'apertura del progetto in Matlab seguendo *Project Shortcuts* > *Open Track Builder;*

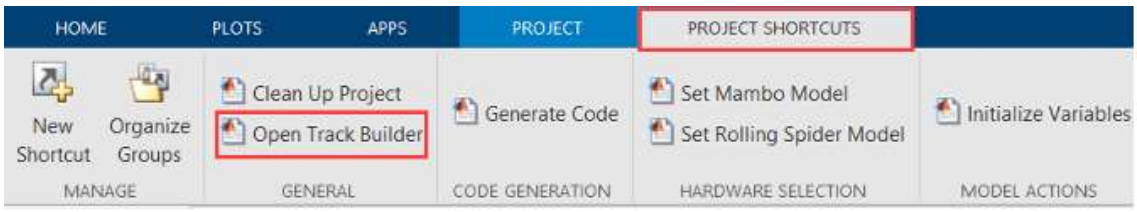


Figura 94 Passi per l'apertura dell'app Track Builder

All'apertura dell'app, si costruisce il proprio percorso facendo click con il mouse nell'area vuota: si selezionano così i vari punti di congiunzione facenti parte del percorso; questi (le loro coordinate) saranno mostrati nel riquadro a fianco.

Attraverso i pulsanti *Clear* e *Show Landing Marker* si può scegliere di eliminare il percorso fino a quel momento tracciato e di mostrare o meno il punto d'atterraggio posto alla fine del percorso.

Il colore predefinito del percorso è il rosso, ma è possibile sceglierne uno di diverso tipo immettendo nel relativo campo il suo codice in esadecimale.

Infine, attraverso il pulsante *Update Virtual World*, vedremo l'aggiornamento del nostro percorso all'interno dall'ambiente virtuale di simulazione.

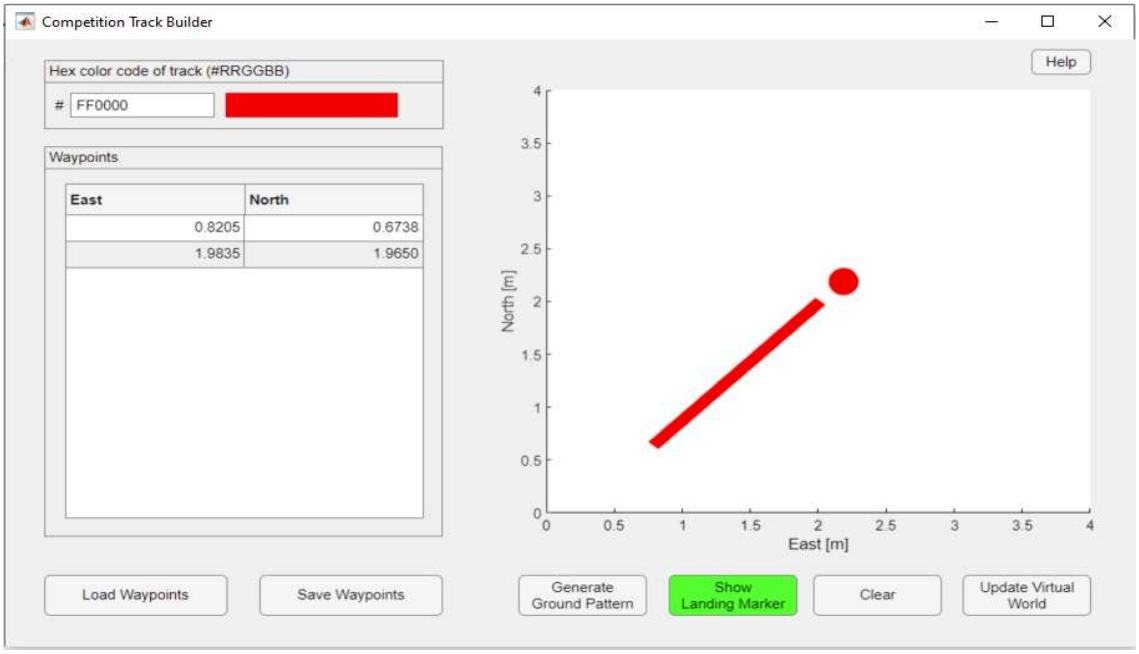


Figura 95 App Track Builder

Conclusioni

Siamo arrivati alla fine della trattazione; spero che il lettore abbia trovato scorrevole la lettura e che sia passata la mia voglia di spiegare quanto maneggiavo con più cura.

Mi piacerebbe che qualcuno prenda in mano questa tesi, e che integrando le proprie competenze con le tecniche descritte finora, riesca a far ruotare il drone in corrispondenza delle curve.

Se può essere d'aiuto, la strada che avevo intrapreso per riuscire nell'intento era quella di normalizzare l'angolo di controllo *Yaw* con *l'angle_track* rilevato *nell'Image Processing System*; infatti, quando quest'ultimo assume valore pari a 90° significativo del caso in cui il percorso è allineato verticalmente con il centro della videocamera, la variabile *Yaw* deve essere posta a 0° (questo da specifica del modello).

Dallo stesso ragionamento, se si sta procedendo in direzione rettilinea (in accordo con la visuale della videocamera) e si incorre in una curva a gomito di 90°, *l'angle_track* rilevato corrisponde a 0° se la curva è a destra o a 180° se la curva è a sinistra.

Dopo queste considerazioni, vi lascio con l'ultima frase che mi ha ispirato "Things that matter are difficult; if we want to do things that matter, it's going to take effort" - Simon Sinek.

Bibliografia

Battiato, P. (s.d.). *Morfologia e Image Processing.*

Caduceo, A. (2022). *Studio e sviluppo di algoritmi di visione per l'inseguimento di traiettorie per droni*, Tesi triennale, Università politecnica delle Marche.

Cappelli, R. (s.d.). *Morfologia Matematica.* Università di Bologna.

Carlizza, P. (2018). *Come è fatto e come funziona un drone*. Tratto da ElectroYou: https://www.electroyou.it/paolo.carlizza/wiki/come-funziona-un-drone

De Tommasi, G. (2012). *Regolatori PID.*

Gaudeni, F. (2019). *Studio e sviluppo di un algoritmo per l'inseguimento di traiettoria da parte di un minidrone*,Tesi triennale, Università Politecnica delle Marche.

Gizzarone, M. (2023). *Studio e sviluppo di sistemi di controllo lineari per l'assetto di droni. Tesi triennale,* Tesi triennale, Università Politecnica delle Marche.

Massimo, N. (2023, giugno 14). *Applicazioni dei droni ad oggi e nel futuro*. Tratto da Wondershare: https://filmora.wondershare.it/drones/drone-applications-and-uses-in-future.html

MathWorks. (s.d.). *Simulink Support Package for Parrot Minidrones*. Tratto da mathworks.com: https://it.mathworks.com/help/supportpkg/parrot/index.html?s_tid=CRUX_lftnav

MathWorks. (s.d.). *Stateflow*. Tratto da mathworks.com: https://it.mathworks.com/help/stateflow/index.html?s_tid=CRUX_lftnav

Orlando, G. (2023). *Generalità sui sistemi di controllo.*

Paperi, L. (2019). *Progetto e sviluppo di sistemi di controllo per droni*, Tesi triennale, Università Politecnica delle Marche.

Terlizzi, M., Russo, L., Silano, G., & Aatif, M. (2021). *A Vision-Based Algorithm for a Path Following Problem.*

Treccani. (s.d.). *drone.* Tratto da TRECCANI: https://www.treccani.it/vocabolario/drone/

Wikipedia. (s.d.). *Aeremobile a pilotaggio remoto*. Tratto da Wikipedia.org: https://it.wikipedia.org/wiki/Aeromobile_a_pilotaggio_remoto

Wikipedia. (s.d.). *Controllo PID*. Tratto da Wikipedia.org: https://it.wikipedia.org/wiki/Controllo_PID