UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Studio e sviluppo di un algoritmo per l'inseguimento di traiettoria da parte di un minidrone

Study and development of a line follower algorithm for a minidrone

Relatore:

Prof. Gianluca Ippoliti

Correlatori:

Prof. Giuseppe Orlando

Prof. Adriano Mancini

Candidato:

Francesco Gaudeni

Matricola 1078800

Anno accademico 2018-2019

A Chiara, Marcella e Andrea

Sommario

Il seguente elaborato si apre con una panoramica sul mondo dei quadricotteri. Successivamente si illustra il modello MATLAB su cui si è lavorato per la realizzazione di un line follower drone, ovvero il parrotMinidroneCompetition, utilizzato per partecipare alla competizione MathWorks Minidrone Competition. Infine si descrive il capitolo sulla realizzazione pratica del codice Simulink.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'obiettivo del presente elaborato è quello di descrivere il lavoro svolto sul minidrone Mambo Parrot, ovvero la creazione di un *line follower*, facendo una panoramica sui droni e il loro funzionamento in primis, per poi passare alla descrizione del modello utilizzato e del codice sviluppato. In letteratura si possono trovare lavori relativi allo sviluppo di *line followers*, come [1] o [9], ma non possiedono le caratteristiche presenti in questo progetto, che è stato completamente sviluppato tramite i software MATLAB e Simulink appositamente per il drone Mambo Parrot.
Nel Capitolo 2 si parla quindi dei quadrirotori, partendo dal loro utilizzo per arrivare ad una breve descrizione del loro funzionamento.
Il Capitolo 3, invece, introduce la competizione a cui è stato inviato il seguente lavoro, descrivendone modalità e regole.
Nel Capitolo 4 vengono analizzati sia il modello matematico di un quadricottero sia il modello MATLAB utilizzato per lo sviluppo di questo progetto.
I Capitoli 5 e 6 servono a presentare sia il "linguaggio" utilizzato per la creazione del codice sia il codice vero e proprio.
Infine, all'interno del Capitolo 7, si fa riferimento a quelli che potrebbero essere gli sviluppi futuri di questo lavoro.

Capitolo 2

Droni e quadrirotori

2.1 Panoramica

Ciò che comunemente viene chiamato drone è un cosiddetto aeromobile a pilotaggio remoto (APR), ovvero un velivolo il cui pilota non si trova a bordo ma a terra o su di un altro veicolo. Questa peculiarità fa sì che l'utilizzo dei droni sia molto più versatile e polivalente rispetto ai tradizionali mezzi aerei, permettendoci ad esempio di impiegarli in situazioni di pericolo per l'uomo, come in zone di guerra. I droni, infatti, nascono principalmente per uso militare, a cui con il passare del tempo e l'avanzamento della tecnologia si sono aggiunti gli usi civili e ricreativi. Un altro esempio di uso a cui ci si può facilmente riferire è quello in zone post sismiche, in cui oltre ad aiutare nella ricerca dei sopravvissuti e a documentare i danni, i droni agevolano la ricostruzione tridimensionale degli edifici danneggiati senza mettere a rischio la vita di alcuna persona. Per quanto riguarda l'utilizzo ricreativo, al giorno d'oggi esistono svariati modelli in commercio, per lo più quadricotteri (o quadrirotori, figura 2.1a), ovvero droni dotati di quattro motori. Questa tecnologia è la più diffusa in quanto, rispetto a droni con cinque o sei motori (figura 2.1b), si hanno costi e pesi contenuti.

(a) Quadrirotore [link]

(b) Esacottero [link]

Figura 2.1: Due tipologie di droni

2.2 Funzionamento

Il drone utilizzato per lo sviluppo di questo lavoro, il Parrot Mambo, ricade proprio nella categoria dei quadrirotori. Come si evince dalla figura 2.2, i quattro motori non girano tutti nello stesso verso, bensì due in senso orario e due in senso antiorario; ciò permette di comandare indipendentemente gli angoli di *pitch, roll, yaw* e il *thrust*, ovvero il beccheggio, il rollio, l'imbardata e la spinta.

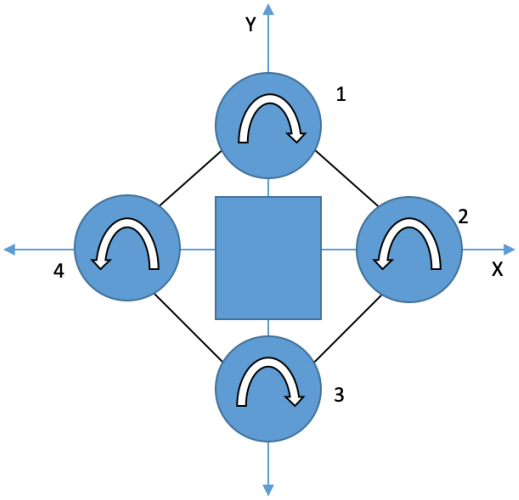
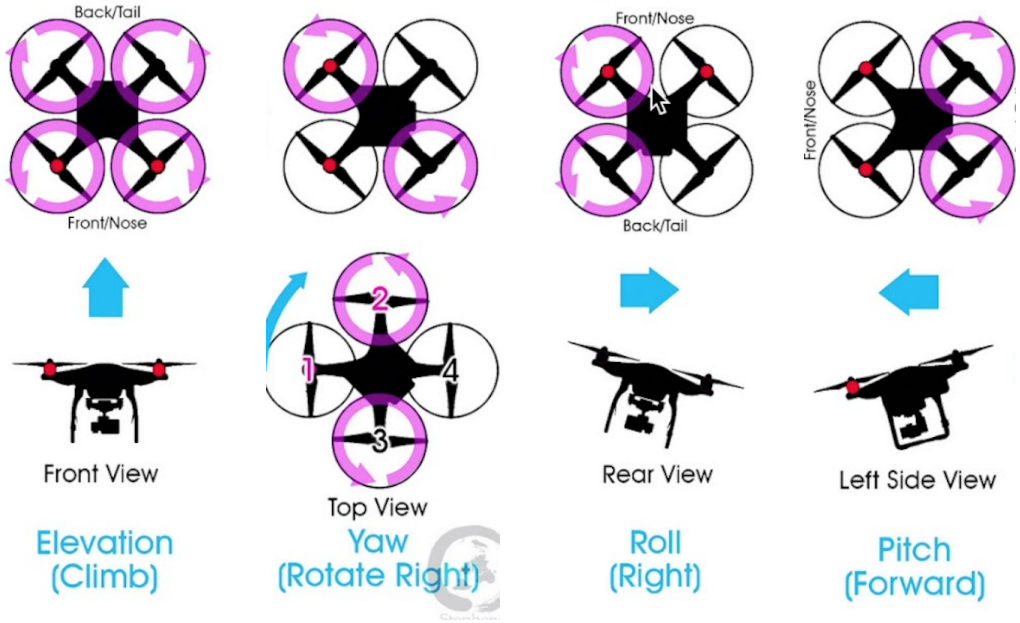


Figura 2.2: Schema esplicativo di un quadrirotore

Un motore produce spinta semplicemente con la rotazione delle eliche, spingendo l'aria verso il basso e generando di conseguenza una forza F verso l'alto.
Se il motore è posto nel centro di gravità di un oggetto, allora quest'ultimo si muoverà di sola traslazione; quando la forza di spinta verrà equilibrata dalla forza di gravità, allora l'oggetto sarà in equilibrio e quindi in una situazione di *hovering*. Invece, quando la forza è applicata su di un lato, verranno prodotte sia una traslazione che una rotazione, il cui risultato sarà quello di far ruotare l'oggetto per poi farlo cadere al suolo, dato che la componente verticale della forza non è in grado di equilibrare la gravità. Questo problema può essere risolto applicando due forze (di intensità F/2) anziché una, ai due lati del nostro modello, in modo tale che le forze e le coppie in gioco si annullino a vicenda.
Questa configurazione, però, genera un altro problema, ovvero il fatto che così non si genera coppia nella terza dimensione, per cui non avremmo rollio.
La configurazione finale prevede quindi quattro motori, che generano quattro spinte, le cui eliche girano in sensi opposti a coppie, come appunto nelle figure 2.2 e 2.3a; in questo modo si può modificare l'angolo di imbardata in modo indipendente dagli altri angoli, semplicemente diminuendo la velocità di rotazione di due motori opposti e aumentando l'altra velocità, come si vede nella figura 2.3b.

Così facendo, quindi, la spinta totale rimane la stessa. Altrimenti, posizionando i motori con stessa rotazione vicini si avrebbe una spinta differente sui lati del drone, che modificherebbe quindi l'angolo di rollio o di beccheggio, a seconda delle posizioni dei motori stessi.
Per modificare invece l'angolo di rollio è necessario diminuire la velocità di due motori "laterali" e incrementare gli altri (figura 2.3c). Infine, per il beccheggio, bisogna incrementare le velocità dei motori posteriori e diminuire quelle degli anteriori, o viceversa, come si intuisce dalla figura 2.3d.



(a) Spinta (b) Imbardata (c) Rollio (d) Beccheggio

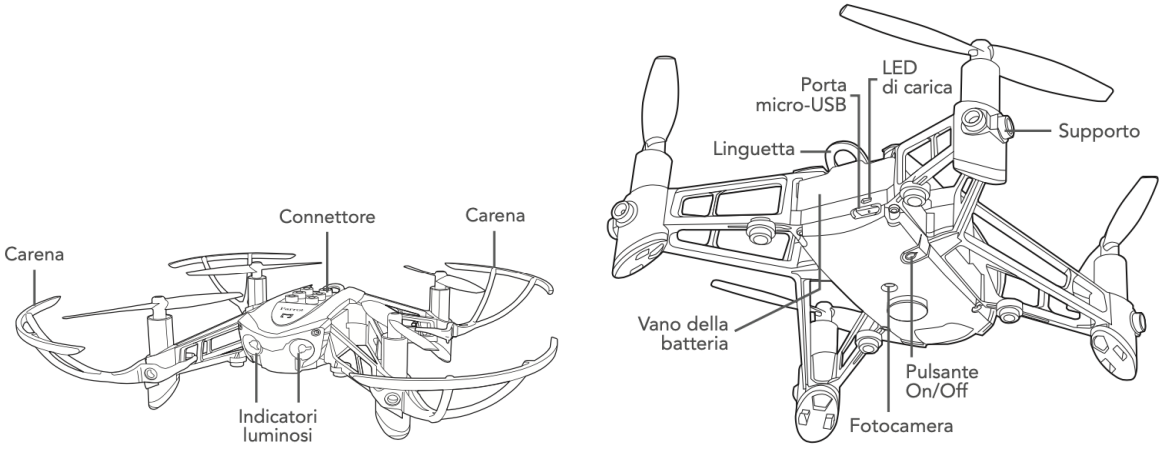
Figura 2.3: Meccanica dei droni
[link]

2.3 Parrot Mambo

Il minidrone utilizzato per lo sviluppo di questo progetto è il Mambo, prodotto dall'azienda francese Parrot. Il vantaggio principale di questo quadricottero è quello di poter essere programmato interamente tramite MATLAB e Simulink, grazie al *Simulink Support Package for Parrot Minidrones*, che permette di sviluppare algoritmi di volo per il minidrone, attraverso programmi preesistenti quali asbQuadcopter o parrotMinidroneCompetition.
Pur essendo un drone di piccole dimensioni, 18×18 cm per 63 grammi di peso (senza paraurti o accessori), il Mambo presenta diversi sensori, che permettono un'esperienza di volo completa.

Sensori:

- Sensore ad ultrasuoni: serve a misurare le distanze dalle superfici. Invia segnali ad alta frequenza al terreno e calcola quanto tempo ci mettono per rimbalzare e tornare indietro.
- Camera: acquisisce fotogrammi a 60 fps, attraverso cui si ricava l'*horizontal motion* e la velocità.
- Sensore di pressione: serve a misurare l'altezza di volo.
- IMU (*Inertial Measurement Unit*): composto da un accelerometro a 3 assi e un giroscopio anch'esso a 3 assi, che calcolano l'accelerazione lineare e la velocità angolare.



(a) Vista semi-frontale. (b) Vista inferiore

Figura 2.4: Parrot Mambo
[link]

Dalla figura 2.4b si possono vedere le posizioni dei vari sensori, tranne quella dell'IMU (situata nella *Printed Circuit Board*): al centro del modello si trovano camera e sonar, mentre in posizione avanzata c'è il sensore di pressione.

Per quanto riguarda gli attuatori, il Mambo presenta quattro motori di tipo *Coreless* (cioè con rotore privo di armatura in ferro), divisi in Motori A (perché girano in senso antiorario, *Anti-clockwise*) e Motori C (senso orario, *Clockwise*). Le eliche soprastanti sono bianche (davanti) e nere (dietro), solamente per distinguere la parte anteriore da quella posteriore: i motori, infatti, hanno una configurazione a X.
La batteria, invece, è di tipo LiPo 660 mAh, con circa dieci minuti di autonomia.
Infine, come si vede dalla figura 2.4a, il modello presenta due indicatori luminosi, che segnalano i diversi stati del Mambo, descritti nella tabella 2.1.

Colore	Comportamento
Arancio fisso	Il Parrot si sta avviando
Verde fisso	Il Parrot è pronto per l'uso
Rosso lampeggiante	La batteria è scarica
Rosso fisso	Il Parrot ha rilevato un problema

Tabella 2.1: Comportamento degli indicatori luminosi

2.3.1 Collegamento

Il Mambo può essere pilotato sia tramite smartphone (attraverso l'app FreeFlight Mini) sia con il joystick Flypad. Inoltre, ed è questo il caso di nostro interesse, il drone può essere collegato tramite bluetooth al computer per permettere a Simulink di caricare il codice direttamente sul Parrot, facendolo così volare con il programma appena sviluppato. Per fare ciò è necessario collegare una prima volta il Mambo al computer, seguendo una specifica procedura chiamata *Hardware Setup*, in modo da sostituire il firmware di fabbrica (che permette il volo con l'app). Una volta effettuati i seguenti passaggi (per sistema operativo Mac OS) e dopo aver collegato il drone al Mac (come un normale dispositivo bluetooth) si potrà far volare il Mambo attraverso i programmi sviluppati in MATLAB & Simulink.

Procedura:

- Dalla home di MATLAB cliccare su *Manage Add-Ons*. Si aprirà così una finestra attraverso la quale si possono gestire tutti i pacchetti aggiuntivi di MATLAB. Scorrere fino al *Simulink Support Package for Parrot Minidrones* e cliccare sull'icona dell'ingranaggio sulla destra per avviare la procedura vera e propria.
- Connettere il minidrone Parrot al computer attraverso il cavo USB. Cliccare su Next.
- Selezionare dalla lista a cascata il minidrone Parrot. Qualora non fosse presente, riconnettere il modello al computer e aspettare che i LED si stabilizzino; cliccare poi su Detect e selezionare il minidrone. A questo punto si può andare avanti, con il pulsante Next.

- Disconnettere il minidrone. Aspettare che i LED diventino arancioni e inizino a lampeggiare alternativamente. Ciò indica che si sta aggiornando il firmware. Solo una volta che i LED siano diventati verdi e siano rimasti di questo colore per almeno dieci secondi, si può ritenere il firmware aggiornato con successo.
- L'ultimo passaggio verifica la corretta esecuzione della procedura, ovvero il corretto aggiornamento.

Capitolo 3

MathWorks Minidrone Competitions

MathWorks, società produttrice del software MATLAB, organizza in giro per il mondo competizioni per studenti chiamate *MathWorks Minidrone Competitions*; l'obiettivo di chi partecipa è quello di creare un algoritmo che permetta al minidrone Parrot Mambo di seguire una linea tracciata sul suolo.
La competizione si svolge in due round: il primo, chiamato "*Simulation Round*", consiste nello sviluppare l'algoritmo usando Simulink. Il programma creato verrà poi analizzato dagli ingegneri della *MathWorks* attraverso il simulatore di volo presente sullo stesso software, valutando i seguenti aspetti:

- Capacità del modello di generare codice.
- Numero di tracciati completati.
- Atterraggio effettuato con successo.
- Accuratezza del tracciato percorso rispetto al tracciato dell'ambiente 3D di Simulink.
- Tempo percorso per completare il tracciato.

Qualora il proprio programma venisse selezionato, il team creatore passerebbe al secondo round, chiamato "*Simulation and Hardware Deployment Round*", in cui partecipanti sono tenuti a testare sul drone i propri algoritmi. Anche questa fase si divide in due parti: *Practice Round* e *Live Round*.

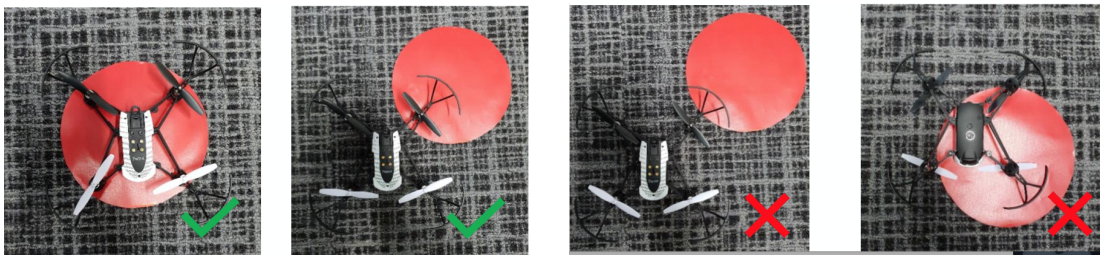
- **Practice Round**:
  - Ogni team ha fino a due slot da 15 minuti ciascuno nell'arena per calibrare il modello.
  - Le performance del minidrone in questo round non vengono considerate per dichiarare il vincitore.

- **Live Round**
  - Ogni team avrà un ulteriore slot da 15 minuti nell'arena in cui far volare il minidrone.
  - Ogni team ha fino a 7 possibilità durante i 15 minuti per far volare il minidrone.
  - Alla fine dello slot il team deve indicare il volo finale che dovrà essere valutato dai giudici.

In questo secondo round, il criterio di valutazione usato dai giudici si baserà sul massimo numero di sezioni che il minidrone ha completato. Ad ogni sezione è associata una fase, per cui:

- **Fase 0**: Decollo completato.
- **Fase 1**: Sezione 1 completata.
- **Fase 2**: Sezione 2 completata.
- ...
- ...
- **Fase N**: Sezione N del tracciato completata.
- **Fase Finale**: Atterraggio regolare sul marker rotondo.

Il fattore tempo verrà considerato solo quando il Mambo è nella Fase Finale, per cui ha completato tutte le sezioni precedenti ed è atterrato con successo sul cerchio; l'atterraggio con successo prevede che ci sia almeno un'elica a contatto col marker e che il drone non sia capovolto.



(a) Atterraggi con successo (b) Atterraggi falliti

Figura 3.1: Tipi di atterraggi
[link]

Per quanto riguarda il tracciato, si conoscono in anticipo tutte le informazioni necessarie ad eccezione del colore, che è rosso (#FF0000, codifica esadecimale) solo nel

simulatore. Il percorso è costituito da una linea spezzata, in cui ogni coppia di segmenti consecutivi forma un angolo di ampiezza variabile. Non sono previste curve "dolci".
Ogni segmento avrà una larghezza di 10 cm e la distanza tra l'ultimo segmento e il centro della circonferenza d'atterraggio, il cui diametro sarà di 20 cm, è pari a 25 cm, come in figura 3.2.

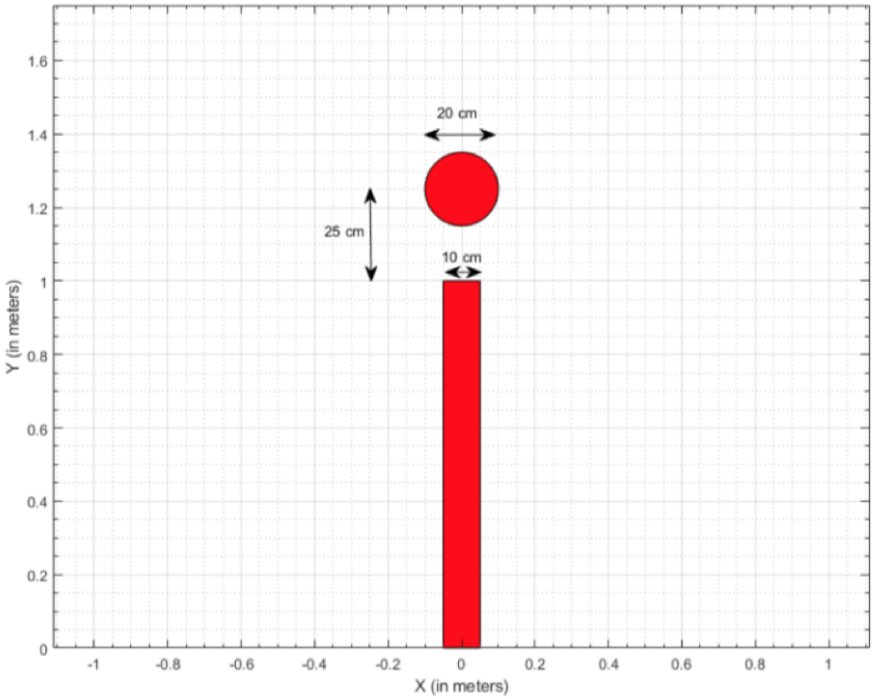


Figura 3.2: Misure del tracciato
[link]

Capitolo 4

Modello parrotMinidroneCompetition

Prima di parlare del modello parrotMinidroneCompetition è conveniente fare una breve introduzione sul modello matematico che sta dietro ad un quadrirotore. Innanzitutto, dato che il drone utilizzato si può modellare secondo le leggi di un corpo rigido sottoposto a forze e coppie, è utile dare la definizione di corpo rigido (secondo [3]): *"Un corpo rigido è un insieme (anche infinito) di punti le cui distanze reciproche non variano nel tempo e nello spazio. Un corpo rigido è sempre riconducibile al sistema di riferimento ortogonale che lo caratterizza".*

4.1 Modello matematico

4.1.1 Angoli di Eulero

Per analizzare l'orientamento nello spazio di un corpo rigido è necessario introdurre gli angoli di Eulero, ovvero angoli che descrivono gli sfasamenti angolari tra una terna di riferimento solidale con il corpo rigido (in rosso in figura 4.1) e una terna di riferimento fissa (in blu in figura 4.1), attraverso una serie di rotazioni elementari. I due sistemi di riferimento coincidono nell'origine, come in figura 4.1.
Gli angoli di Eulero sono di solito denotati con α, β, γ oppure con ϕ, θ e ψ ed esistono fino a dodici possibili sequenze di terne di rotazione, divise in due gruppi:

- **Angoli di Eulero classici**: $(z{-}x{-}z, x{-}y{-}x, y{-}z{-}y, z{-}y{-}z, x{-}z{-}x, y{-}x{-}y)$
- **Angoli di Tait-Bryan**: $(x{-}y{-}z, y{-}z{-}x, z{-}x{-}y, x{-}z{-}y, z{-}y{-}x, y{-}x{-}z)$

Questi ultimi sono quindi un sottoinsieme dei più generici angoli di Eulero e tra loro possiamo trovare gli angoli di imbardata, rollio e beccheggio. L'imbardata (*yaw*) è una rotazione di un angolo ψ intorno all'asse z.
Il rollio (*roll*) è una rotazione di un angolo ϕ intorno all'asse x e infine il beccheggio (*pitch*) è una rotazione di un angolo θ intorno all'asse y.

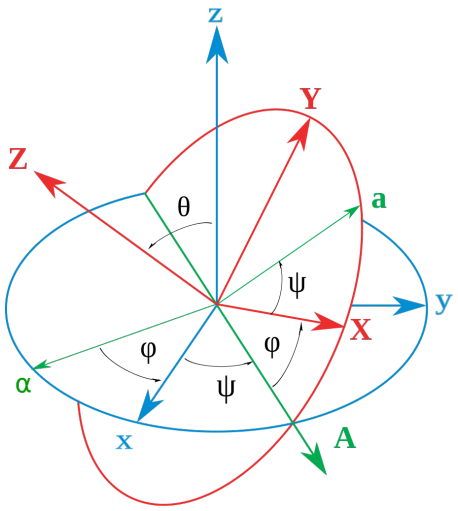


Figura 4.1: Schema degli angoli di Eulero
[link]

4.1.2 Matrici di rotazione

Per passare dal sistema di riferimento mobile a quello fisso bisogna considerare tre matrici di rotazione, ciascuna corrispondente ad un angolo, e quindi ad una manovra di rollio, beccheggio e imbardata. Queste matrici sono:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\phi & -S\phi \\ 0 & S\phi & C\phi \end{bmatrix} \tag{4.1}$$

$$R_y(\theta) = \begin{bmatrix} C\theta & 0 & S\theta \\ 0 & 1 & 0 \\ -S\theta & 0 & C\theta \end{bmatrix} \tag{4.2}$$

$$R_z(\psi) = \begin{bmatrix} C\psi & -S\psi & 0 \\ S\psi & C\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

con Cψ=cos(ψ) e Sψ=sin(ψ). La matrice di rotazione complessiva, chiamata *Direct Cosine Matrix* (DCM,4.5) sarà data allora dalla moltiplicazione fra le precedenti:

$$R(\phi, \theta, \psi) = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi) \tag{4.4}$$

$$R(\phi, \theta, \psi) = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi - S\psi C\phi & C\psi S\theta C\phi + S\psi S\phi \\ S\psi C\theta & S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi - S\phi C\psi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix} \tag{4.5}$$

4.1.3 Equazioni della dinamica

Come approfondito in [14] esistono diversi approcci per descrivere il modello dinamico di un sistema a sei gradi di libertà. In questo elaborato è stato seguito il metodo di Newton-Eulero. Innanzitutto si fanno le seguenti ipotesi:

- il drone ha una struttura perfettamente simmetrica e la matrice d'inerzia è diagonale,
- la struttura fisica è un corpo rigido equipaggiato con quattro motori,
- il centro di gravità del drone è fisso e coincide con il sistema di riferimento solidale col drone,
- la spinta è proporzionale al quadrato della velocità,
- il modello è tempo-invariante per cui non ci sono cambiamenti di massa durante il movimento,
- tutti i motori sono identici, per cui si può analizzarne solamente uno senza perdere di generalità,
- effetti aerodinamici come il *blade-flapping* (inclinazione della lama dell'elica per compensare l'effetto dell'aria) sono considerati nulli.

Poi, partendo dalla seguente equazione (di Newton-Eulero):

$$\begin{bmatrix} mI_{3,3} & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} \dot{\mu} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega \times m\omega \\ \omega \times J\omega \end{bmatrix} = \begin{bmatrix} F \\ \tau \end{bmatrix} \tag{4.6}$$

si arriva al sistema 4.7 che descrive in maniera completa la dinamica di un quadrirotore, come spiegato in [4] e [5].

$$\begin{cases} \ddot{X} = (S_\phi S_\psi + C_\psi S_\theta C_\phi)\frac{U_z}{m} \\ \ddot{Y} = -(S_\phi C_\psi + S_\psi S_\theta C_\phi)\frac{U_z}{m} \\ \ddot{Z} = -g + C_\theta C_\phi \frac{U_z}{m} \\ \ddot{\phi} = \frac{1}{J_{xx}}(U_\phi + (J_{yy} - J_{zz})\dot{\theta}\dot{\psi} + \dot{\theta}\Omega J_{tp}) \\ \ddot{\theta} = \frac{1}{J_{yy}}(U_\theta + (J_{zz} - J_{xx})\dot{\phi}\dot{\psi} - \dot{\phi}\Omega J_{tp}) \\ \ddot{\psi} = \frac{1}{J_{zz}}(U_\psi + (J_{xx} - J_{yy})\dot{\phi}\dot{\theta}) \end{cases} \tag{4.7}$$

4.2 Descrizione generale parrotMinidroneCompetition

Il progetto sviluppato si basa sul modello Simulink che può essere lanciato dalla *Command Window* di MATLAB digitando la seguente istruzione: parrotMinidroneCompetitionStart. Facendo ciò si va a creare una cartella all'interno del workspace denominata parrotMinidroneCompetition, al cui interno si troveranno tutti i file utili per lo sviluppo del lavoro.
Una volta aperto il progetto compariranno a video sia il simulatore Minidrone Flight Visualization sia la finestra di Simulink (figura 4.2) da cui iniziare a modificare il programma.

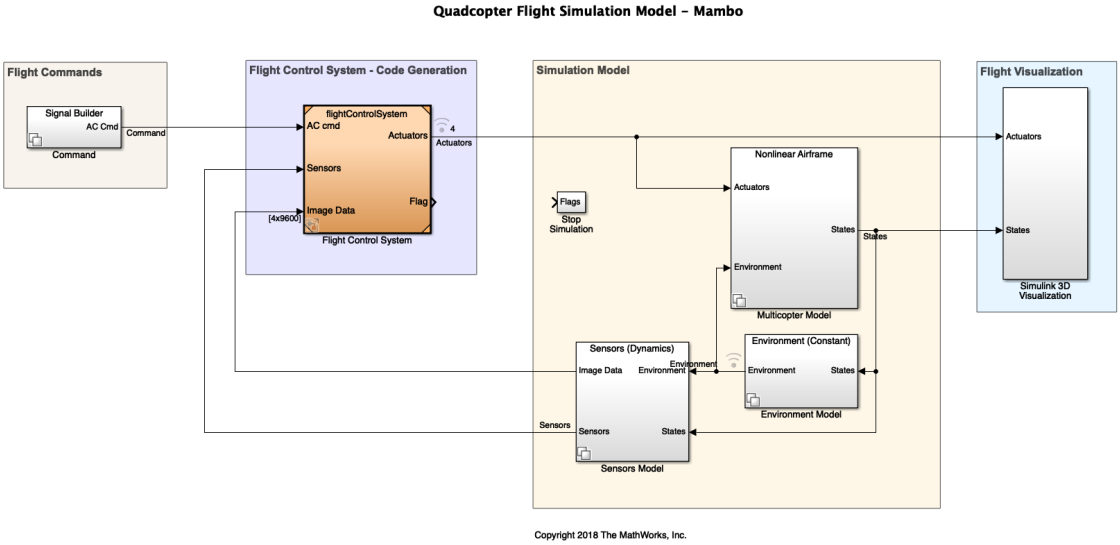


Figura 4.2: Modello completo che si apre all'avvio del progetto

Come si intuisce direttamente dall'immagine, il modello è composto da sette blocchi:

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

4.3 Command

Aprendo il blocco Command compare una schermata formata da quattro blocchi (figura 4.3), ovvero le quattro possibilità attraverso cui è possibile generare i riferimenti per il Mambo:

- Signal Editor: si impongono i riferimenti attraverso il *Signal Builder* di MATLAB.
- Joystick: il Mambo viene pilotato tramite tastiera, senza imporre riferimenti.
- Data: vengono imposti i riferimenti al drone tramite file di formato *.mat*.
- Spreadsheet Data: vengono imposti i riferimenti al drone tramite foglio di calcolo.

La scelta può essere effettuata andando a modificare il valore della VSS_COMMAND, direttamente dalla *command window* di MATLAB.

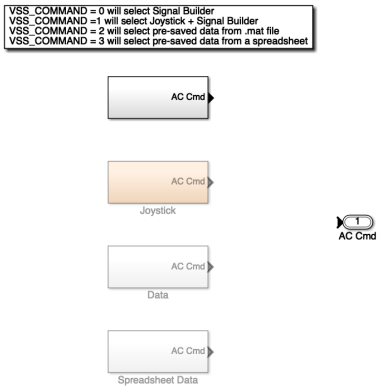


Figura 4.3: Command

Nella parte sinistra della figura 4.4 si può quindi vedere il blocco Position/Attitude Reference, deputato a creare i sei segnali di riferimento, che si uniranno poi nei vettori *pos_ref* e *orient_ref*.

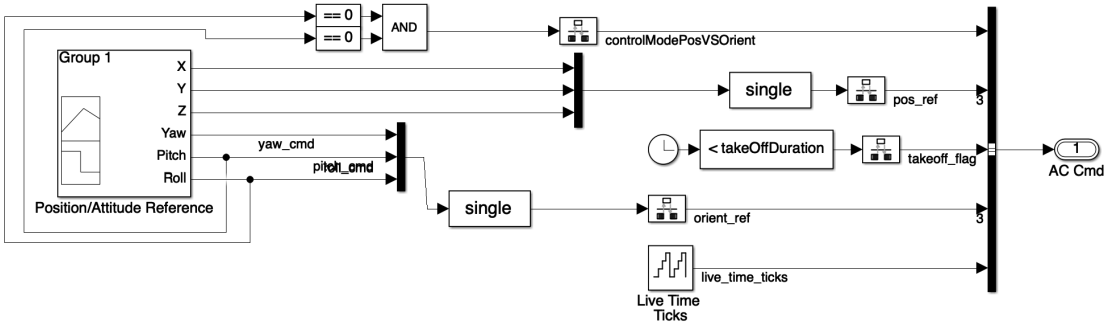


Figura 4.4: Signal Builder

4.4 Flight Control System

Il blocco Flight Control System merita di essere descritto in maniera più approfondita rispetto agli altri perché è quello in cui l'algoritmo vero e proprio del line follower viene implementato. Infatti, questo è l'unico blocco il cui codice viene compilato e caricato sul drone (via bluetooth). Il Flight Control System ha tre input, che sono:

- AC cmd, segnali di riferimento provenienti da Command,
- Sensors, segnali dei sensori provenienti da Sensors Model,
- Image Data, immagini della camera provenienti da Sensors Model,

e due output

- Actuators, nuovi segnali di riferimento per i motori, vanno in Multicopter Model e Simulink3D Visualization,
- Flag, segnale di stop che va nel blocco Stop Simulation.

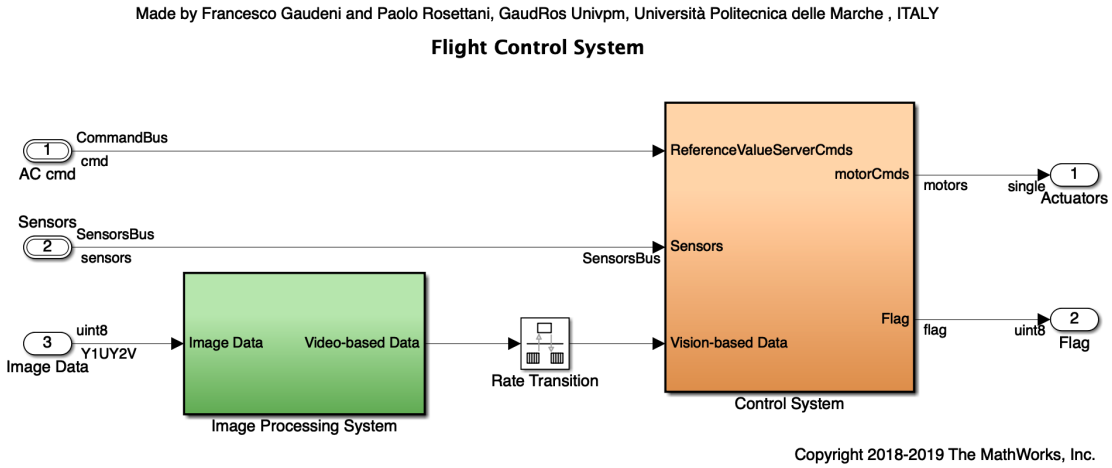


Figura 4.5: Flight Control System

Il blocco si divide in due sottoblocchi, l'Image Processing System e il Control System, collegati tra loro da un Rate Transition, dato che questi operano a due sample time (parametro che indica ogni quanto, durante la simulazione, il blocco produce output) differenti. I sottoblocchi sono approfonditi nelle seguenti sezioni.

4.4.1 Image Processing System

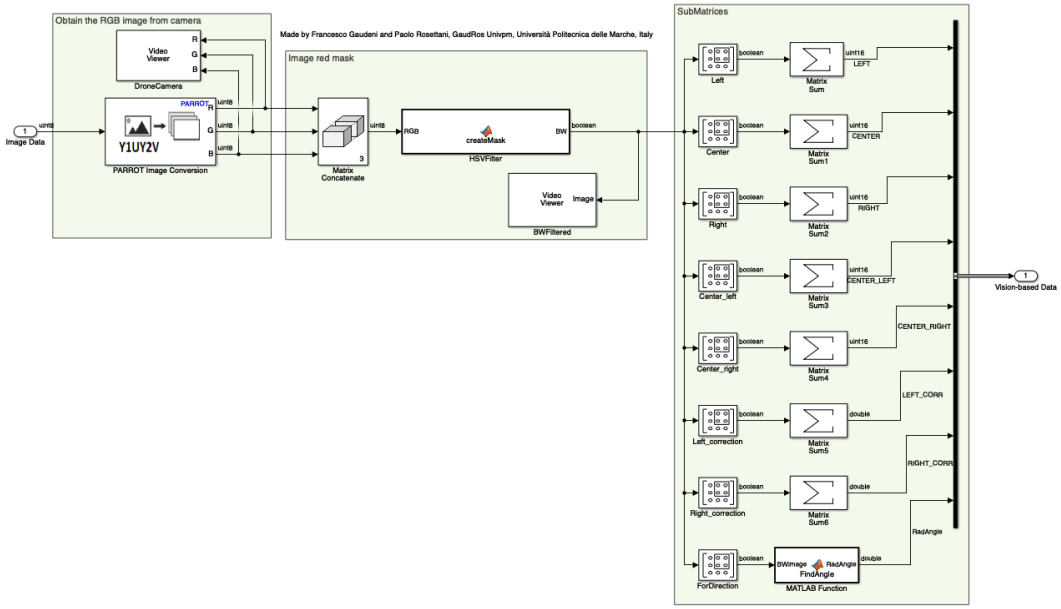


Figura 4.6: Image Processing System

Prima di parlare di questa sezione è bene fare tre digressioni, che aiuteranno il lettore a comprendere meglio tutte le dinamiche.

Spazio di colore HSV

L'HSV è un modello di colorazione che utilizza la tonalità (*Hue*), la saturazione (*Saturation*) e la luminosità (*Value*) anziché definire le componenti di colore come nel RGB (textitRed, Green, Blue) o nel CMY (textitCyan, Magenta, Yellow). Come approfondito in [17], una rappresentazione tridimensionale dello spazio di colore HSV è l'*hexacone* (cono con in cima un esagono, figura 4.7), dove l'asse verticale centrale rappresenta l'intensità, la tonalità è definita come un angolo tra 0 e 2π e la saturazione è misurata come la distanza radiale dall'asse centrale.

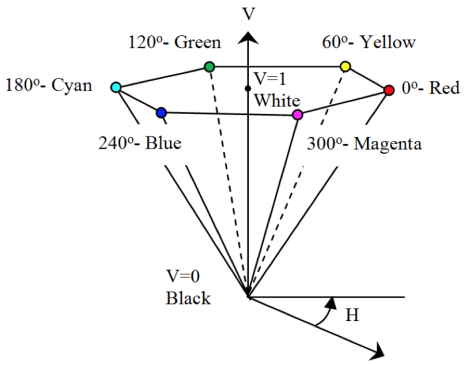


Figura 4.7: Hexacone [link]

```
function [BW] = 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 11-Jan-2020
%------------------------------------------------------

% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.882;
channel1Max = 0.086;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.037;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.848;
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;
end
```

Figura 4.8: Funzione HSVFilter

Algoritmo di Canny

Questo algoritmo è un metodo per determinare i bordi degli elementi presenti in un'immagine e, come approfondito in [10], si basa su tre criteri:

- Basso tasso di errore: bassa probabilità di non individuare un bordo reale ed una bassa probabilità di individuare un bordo falso.
- Buona capacità di localizzazione: la distanza fra i pixel dei bordi trovati dall'algoritmo e il bordo reale sia minima.
- Risposta ad un singolo bordo: l'algoritmo dovrebbe fornire una risposta univoca in corrispondenza di un bordo reale.

Trasformate di Hough

La trasformata di Hough "è una tecnica che permette di riconoscere particolari configurazioni di punti presenti nell'immagine, come segmenti, curve o altre forme prefissate" ([18]). La trasformata si basa sul fatto che la forma cercata può essere rappresentata da una funzione che contiene specifici parametri. La trasformazione avviene perché una retta nel piano cartesiano viene rappresentata come un punto nello spazio dei parametri, e viceversa.

A questo punto si hanno le conoscenze necessarie per parlare del blocco Image Processing System, che ha una sola freccia di input, l'Image Data, e una sola freccia di output, il Video-based Data. Attraverso il blocchetto PARROT Image Conversion si ottiene una conversione dall'immagine ottenuta dalla camera, in formato Y1UY2V, all'immagine in RGB, sotto forma di tre matrici 120×160 di tipo uint8 (una per ogni componente di colore). Queste matrici vengono poi passate al blocchetto Matrix Concatenate, che restituisce un'unica matrice, attraverso cui è più immediato lavorare. Infatti, questa viene data in input alla funzione HSVFilter (vedi figura 4.8), la quale, attraverso il modello di colorazione HSV, riesce ad estrarre una maschera in bianco e nero (1 e 0) dall'immagine colorando di bianco solamente la linea rossa (nel caso del simulatore). Da qui si creano tutte le otto finestre (figura 4.9, non completa per questioni di dimensioni) che servono a far orientare il drone sopra la linea.
Per avere un unico dato che rappresentasse la quantità di colore vista da ogni finestra è stato utilizzato il blocchetto Matrix Sum, che somma tutti gli elementi di una matrice; in questo modo, ad ogni variabile che compone il bus del Video-based Data (tranne la RadAngle) si può associare un valore univoco, che sarà quindi direttamente proporzionale alla quantità di bianco. Di conseguenza ad ogni finestra è associata ad una variabile, il cui scopo è illustrato più avanti.
Qui invece ci si può soffermare sull'ultima finestra ForDirection, che viene data in input alla funzione FindAngle di figura 4.10 anzichè al blocchetto Matrix Sum; questo perché, attraverso l'algoritmo di Canny e le trasformate di Hough spiegati precedentemente, si vuole ottenere in output il valore in radianti dell'angolo compreso tra la sezione che il Mambo sta percorrendo e la sezione appena successiva, che verrà utilizzato nel codice della curva.

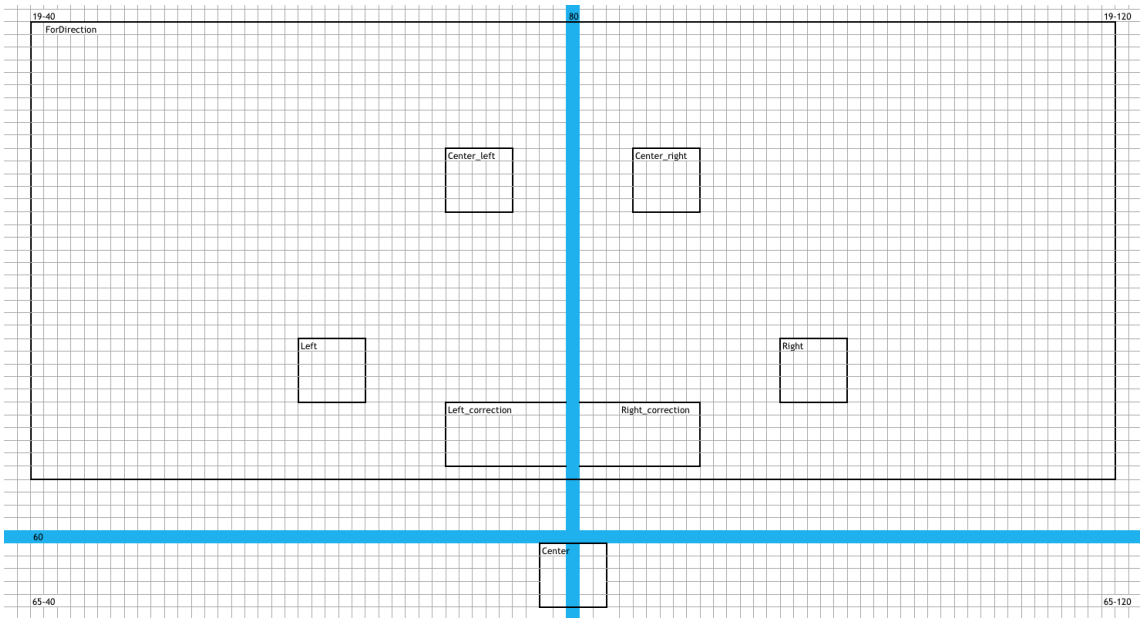


Figura 4.9: Divisione dell'immagine

```
function RadAngle   = FindAngle(BWimage)
BW = edge(BWimage); %Canny
[H, T] = hough(BW); %Hough
P = houghpeaks(H,2); %Punti
 RadAngle = deg2rad(mean(T(P(:,2))));
%DegAngle = T(P(1,2));
end
```

Figura 4.10: Funzione FindAngle

4.4.2 Control System

Come si può vedere nella figura 4.5, il Control System prende in ingresso tre input e restituisce due output. Quattro di questi cinque dati sono già stati spiegati nell'introduzione della sezione, mentre l'input Video-based Data è il risultato del blocco approfondito precedentemente.

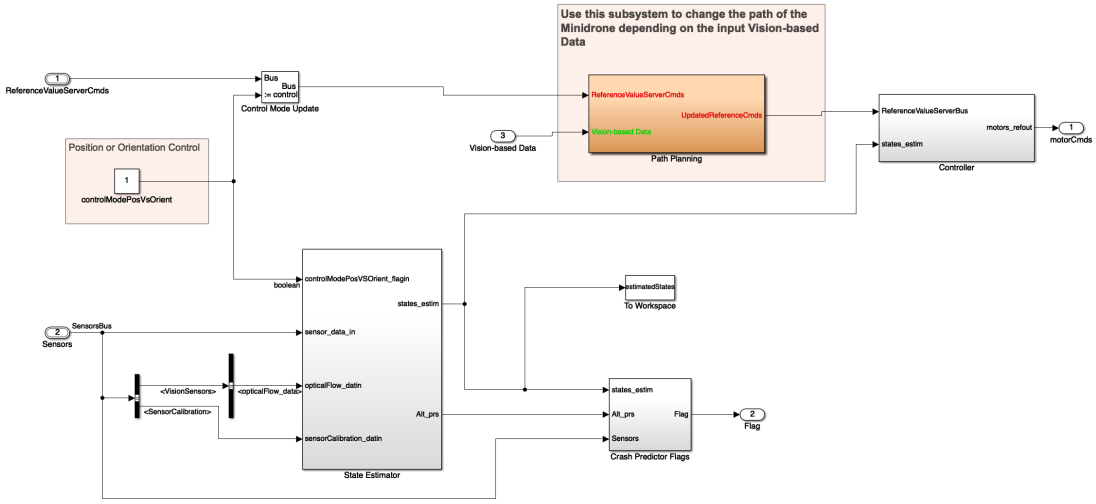


Figura 4.11: Control System

All'interno del Control System rappresentato in figura 4.11, è possibile trovare il Path Planning, ovvero il blocco su cui si basa il lavoro di questo elaborato, il Controller, lo State Estimator e il Crash Predictor Flags. Il primo tra questi blocchi verrà analizzato nel dettaglio più avanti, mentre per quanto riguarda gli altri tre si può dire che:

- il Controller (rappresentato in figura 4.12) riporta al suo interno tutti gli algoritmi di controllo (PID o PD) per l'altitudine e per gli angoli di rollio, beccheggio e imbardata. L'errore utilizzato per il calcolo è dato dalla differenza tra la lettura reale e lo stato stimato;
- lo State Estimator realizza le stime di altitudine, posizione del piano xy e angoli di rollio, beccheggio e imbardata;
- il Crash Predictor Flags, figura 4.13, è il blocco il cui output va nello Stop Simulation spiegato nella sezione successiva.

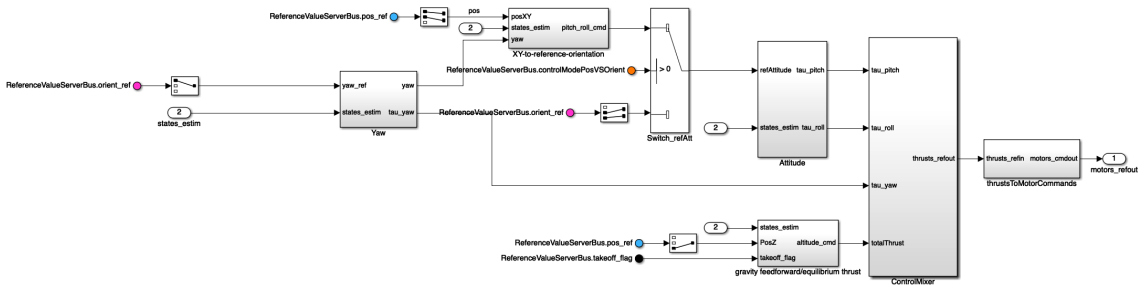


Figura 4.12: Controller

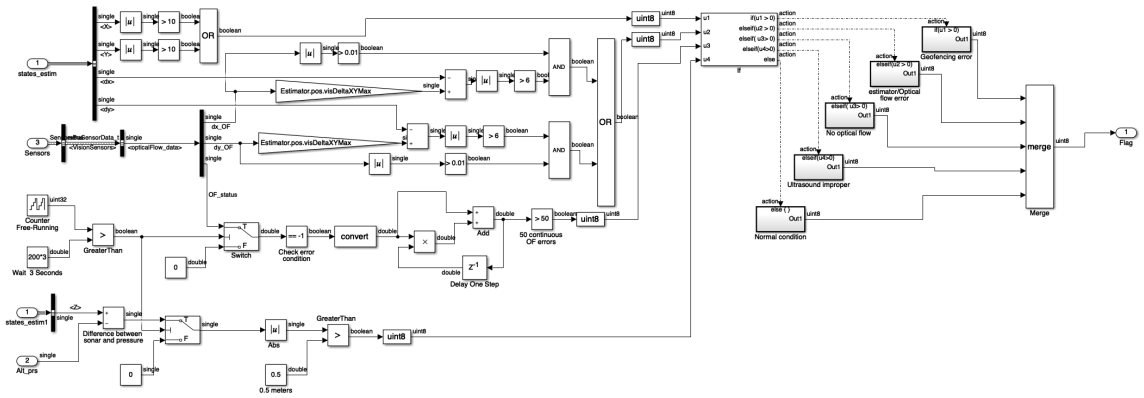


Figura 4.13: Crash Predictor Flags

4.5 Stop Simulation

Questo blocco serve a valutare il flag proveniente dal Crash Predictor Flags, che effettua un controllo su determinate condizioni del velivolo per prevenire eventuali comportamenti da evitare, quali cadute improvvise o ribaltamenti. Nella figura 4.14 si nota come non ci sia collegamento tra il Flight Control System e lo Stop Simulation; questo perché, altrimenti, nel simulatore avrebbe interrotto l'atterraggio del Mambo rilevando una caduta improvvisa o un incidente.

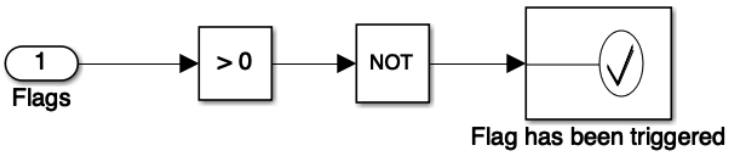


Figura 4.14: Stop Simulation

4.6 Multicopter Model

Anche qui è lasciata all'utente una scelta, ovvero quella di operare su un modello lineare o su uno non lineare, grazie al cambio di valore della variabile VSS_VEHICLE (figura 4.15). Selezionando il modello non lineare si aprirà il blocco di figura 4.16, formato da tre elementi:

- AC model, che contiene la parte della dinamica e restituisce sia le forze F_cg che i momenti M_cg sui tre assi.
- 6DOF (Quaternion), che permette di calcolare velocità, accelerazioni e angoli grazie alle equazioni 4.7.
- Bus setup, che organizza tutte le grandezze provenienti dal 6DOF (Quaternion) in un unico bus.

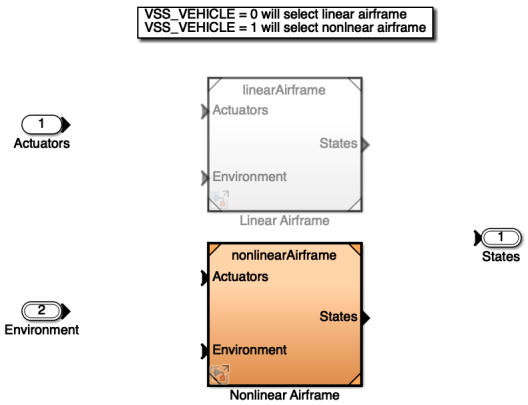


Figura 4.15: Multicopter Model

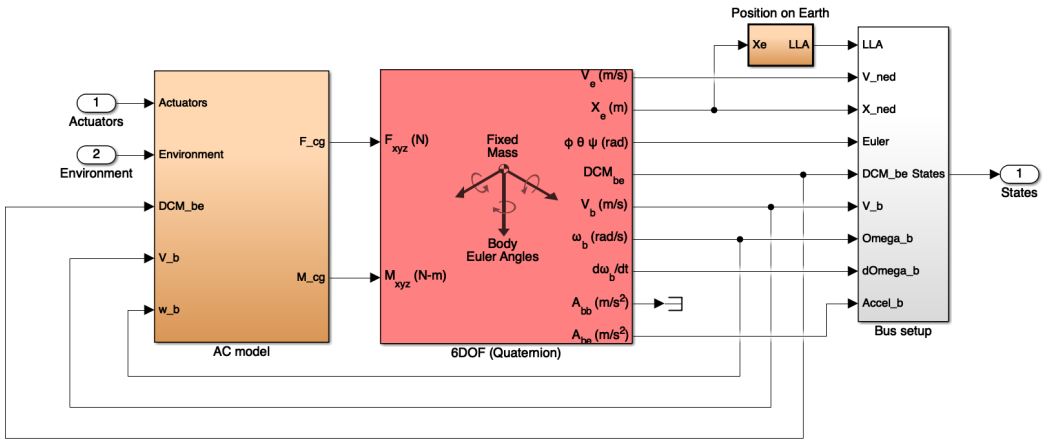


Figura 4.16: Nonlinear Model

4.7 Sensors Model

Come nel precedente blocco, anche in Sensors Model è possibile scegliere fra due modelli, il primo con sensori di tipo ideale, mentre il secondo con sensori che presentano rumore e imprecisione, ovvero più vicini alla realtà.

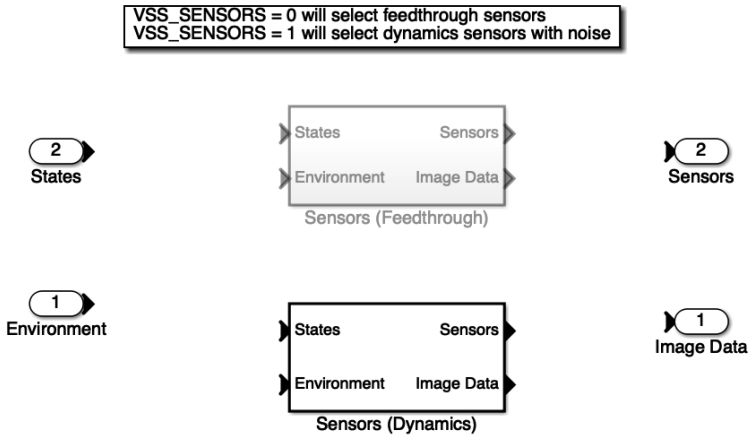


Figura 4.17: Sensors Model

Aprendo il Sensors (Dynamics) (figura 4.17) e successivamente il Sensor System (figura 4.18) si possono trovare i blocchi Camera e IMU_Pressure che, rispettivamente, acquisiscono i dati dalla fotocamera e dall'IMU.

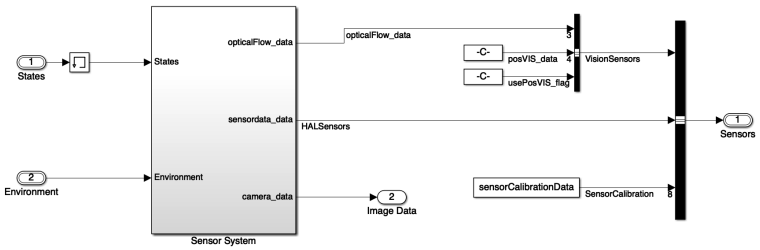


Figura 4.18: Sensors

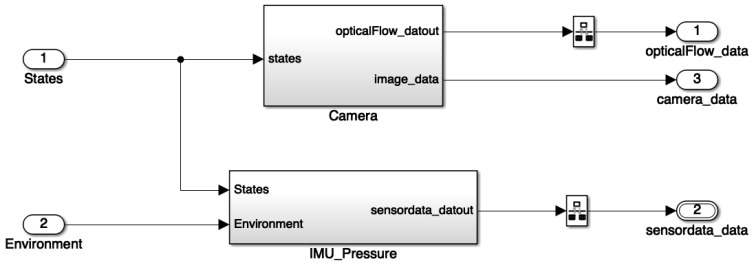


Figura 4.19: Sensors System

4.8 Enviroment Model

Anche qui si hanno due opzioni, selezionabili tramite la variabile VSS_ENVIROMENT (figura 4.20). Impostandola a 0 l'ambiente circostante al drone verrà selezionato come costante, mentre settando il valore a 1 si avrà una rappresentazione più realistica della realtà, con la presenza di ostacoli o edifici ad esempio.

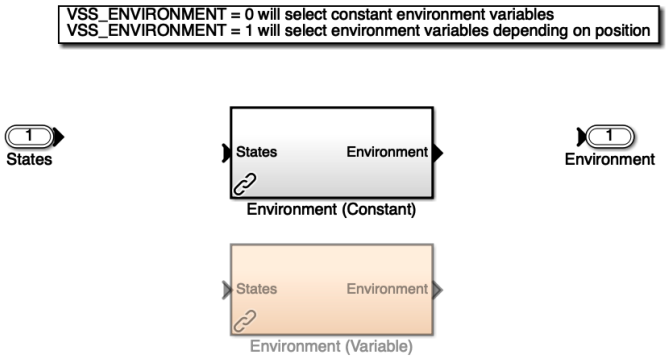


Figura 4.20: Enviroment Model

Come si vede dalla figura 4.21, tutte le variabili "ambientali" come la temperatura dell'aria, la velocità del suono, la pressione o la densità dell'aria sono impostate a valori costanti e confluiscono nell'AtmosphereBus.

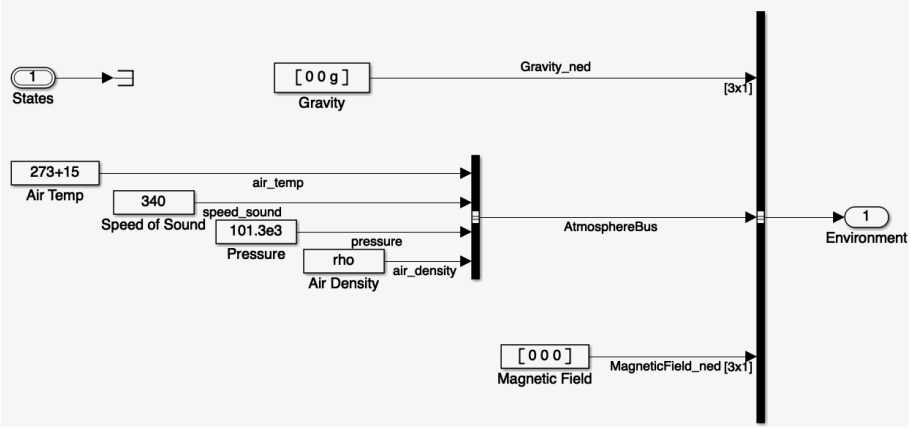


Figura 4.21: Enviroment

4.9 Simulink 3D Visualization

Insieme allo Stop Simulation, il Simulink 3D Visualization è l'unico blocco del modello a non dare dati in output; questo perché il presente blocco serve esclusivamente alla simulazione del sistema. In figura 4.23a si può vedere l'Extract Flight Instruments, che permette all'utente di controllare alcuni parametri come altitudine, orientamento o velocità dei quattro motori, espressa in giri al minuto. Gli stessi input, ovvero Actuators e States, vengono passati anche al blocco di figura 4.23b, che è quello deputato a creare il simulatore 3D attraverso il quale sono stati svolti tutti i test del codice.

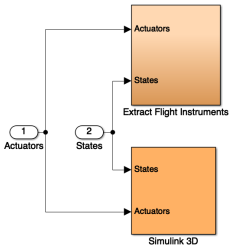
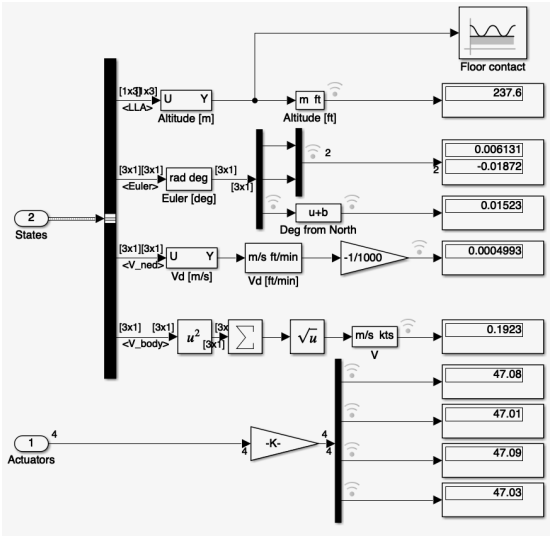
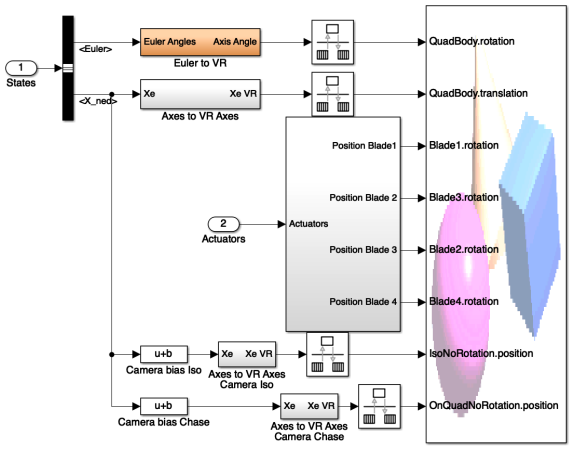


Figura 4.22: Simulink 3D Visualization



(a) Extract Flight Instruments.



(b) Simulink3D

Figura 4.23: Blocchi del Simulink 3D Visualization

Capitolo 5

Stateflow

Il Path Planning è stato programmato principalmente in Stateflow, ovvero un linguaggio che attraverso diagrammi di flusso e macchine a stati permette di descrivere l'algoritmo che sta alla base del *line follower* (che verrà analizzato nel capitolo successivo). Come si vede in figura 5.1, l'ambiente di Stateflow è integrato in uno schema Simulink attraverso un blocco specifico, denominato Chart, a cui vengono passati tutti gli input e che restituisce tutti gli output.

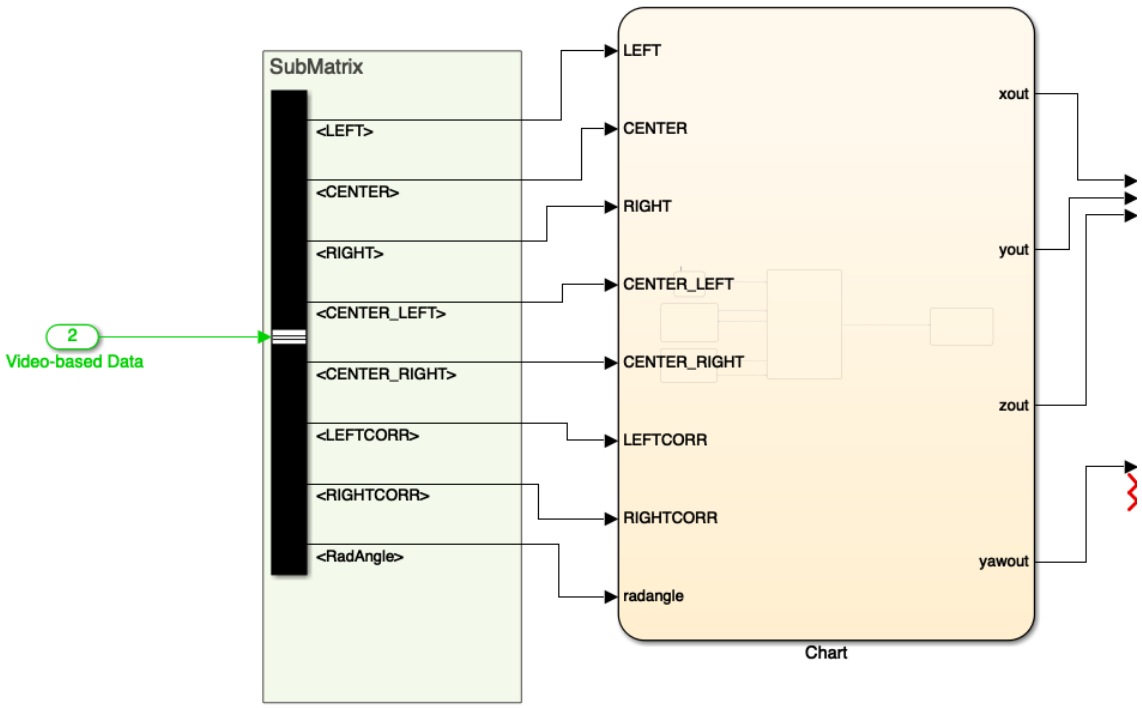


Figura 5.1: Blocco Chart Stateflow

Dal blocco Chart è possibile accedere al Symbols Pane, in figura 5.2, da cui si possono modificare le caratteristiche di ogni variabile, vale a dire indicarne il tipo (input, local, output, constant, data memory store, parameter data, temporary data), il valore iniziale (value) e la porta (qualora si tratti di una variabile di ingresso o di uscita).

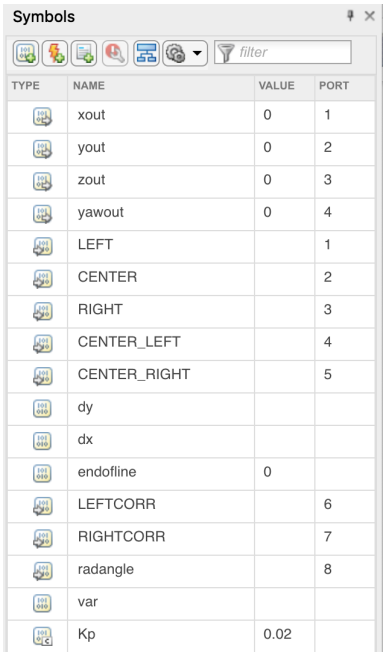


Figura 5.2: Symbols Pane (parziale)

Gli elementi base di un diagramma Stateflow sono blocchi e frecce. Ogni blocco rappresenta uno stato del nostro processo, contenente quindi tutte le operazioni inerenti a quella specifica situazione di volo; le operazioni e i calcoli all'interno di ogni blocco possono essere divisi in differenti categorie, analizzate nella tabella 5.1.
Le frecce, invece, rappresentano le transizioni, ovvero le condizioni che, se verificate, permettono di passare da uno stato origine ad uno destinazione. Ogni freccia può contenere anche più di una condizione, con collegamenti formati da operatori logici OR e/o AND. Una transizione particolare è quella di default, che non ha bisogno di alcuna condizione per verificarsi ed è collegata con il primo stato da cui parte il programma, in questo caso *TAKE_OFF*. Il formato generico di una transizione è il seguente: *event_or_message[condition]condition_action/transition_action*.

- **event_or_message**: specifica un evento o un messaggio che attiva la transizione quando la condizione è vera.
- **condition**: compresa tra parentesi quadre, è la condizione booleana che se verificata attiva la transizione.
- **condition_action**: compresa tra parentesi graffe, è l'azione che avviene prima di passare allo stato destinazione, quando la condizione è vera, ma prima che la transizione sia dichiarata valida.
- **transition_action**: si esegue dopo che la transizione alla destinazione è dichiarata valida.

Azione	Abbreviazione	Descrizione
entry	en	Si esegue quando lo stato diventa attivo
exit	ex	Si esegue quando lo stato si disattiva
during	du	Si esegue quando lo stato si attiva e avviene uno specifico evento
bind	/	Associa un evento o un dato allo stato cosicché solo questo stato possa gestire l'evento o modificare il dato
on *event_name*	/	Si esegue quando lo stato è attivo e riceve un *event_name*
on *message_name*	/	Si esegue quando è disponibile un *message_name*
on after(n, *event_name*)	/	Si esegue quando lo stato è attivo riceve un numero n di *event_name*
on before(n, *event_name*)	/	Si esegue quando lo stato è attivo e prima di ricevere un numero n di *event_name*
on at(n, *event_name*)	/	Si esegue quando lo stato è attivo e riceve esattamente un numero n di *event_name*
on every(n, *event_name*)	/	Si esegue quando lo stato è attivo e al ricevimento di esattamente n *event_name*

Tabella 5.1: Insieme di tutte le azioni

Capitolo 6

Path Planning

6.1 Idea generale

Il Path Planning è il blocco Simulink su cui i partecipanti alla competizione devono lavorare per sviluppare il *line follower*. Prende in ingresso due vettori, ReferenceValue-ServerCmds, che setta i riferimenti che il drone deve seguire, e Video-BasedData, ovvero i dati provenienti dalla fotocamera sottostante. Attraverso questi due input produce il vettore di output UpdateReferenceCmds che contiene le nuove indicazioni per i motori, come da figura 6.1.

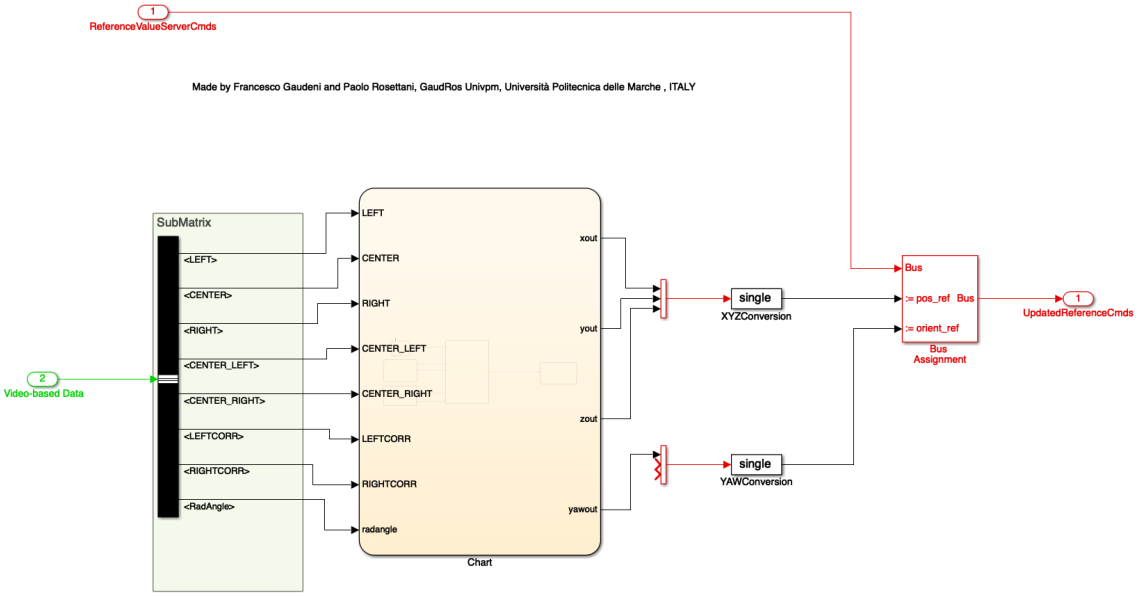


Figura 6.1: Blocco Path Planning

L'obiettivo di questo programma è quello di permettere al Mambo di seguire in autonomia una linea tracciata sul pavimento; per cui il lavoro da fare è stato suddiviso in cinque diverse fasi, che complessivamente riassumono il volo del drone, ovvero il decollo, il movimento, le due rotazioni e l'atterraggio.

L'idea che sta dietro al codice è essenzialmente questa: il drone decolla e si muove in avanti incrementando le coordinate x (in questo caso corrisponde all'asse verticale) e y (asse orizzontale). Una volta che incontra una curva viene calcolato l'angolo preciso tra la due linee sottostanti, grazie all'algoritmo di Canny e alle trasformate di Hough, e si modifica l'imbardata del Mambo di un angolo pari all'angolo appena misurato. Per evitare scostamenti dalla linea è stato implementato un controllore di tipo PI, proporzionale-integrale, che permette il riallineamento. Infine si ha l'atterraggio, che dovrà essere fatto nella circonferenza alla fine del tracciato. Tutto il codice è spiegato in dettaglio nella seguente sezione.

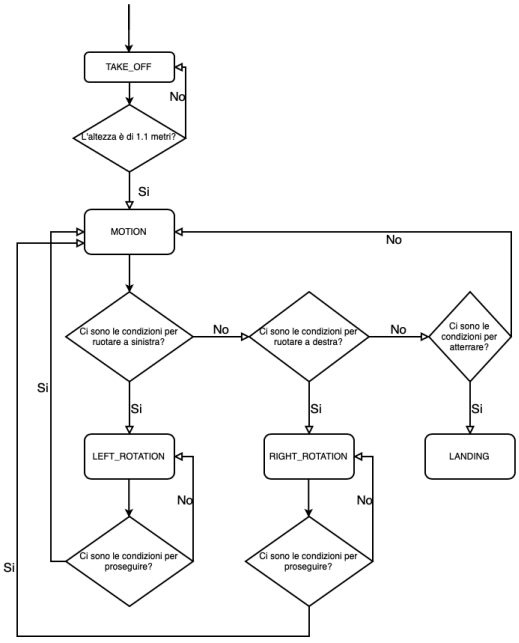


Figura 6.2: Diagramma di flusso

Figura 6.3: Diagramma Stateflow completo

6.2 Sviluppo

Il diagramma in figura 6.3 contenente il codice per l'inseguimento della traiettoria è formato da cinque stati:

- *TAKE_OFF*
- *LEFT_ROTATION*
- *RIGHT_ROTATION*
- *MOTION*
- *LANDING*

Innanzitutto va fatta la lista di tutte le variabili utilizzate, divise per tipologia.
Si ricorda che tutte le otto variabili di input derivano dalle otto finestre in cui è stata divisa l'immagine di 120×160 pixel che la fotocamera acquisisce. Per questo, tra parentesi è indicato il posizionamento di ogni finestra (righe-colonne).

Input:

- LEFT: utilizzata per la curva a sinistra e per l'atterraggio (45-49 60-64).
- CENTER: utilizzata per l'atterraggio (61-65 78-82).
- RIGHT: utilizzata per la curva a destra e per l'atterraggio (45-49 96-100).
- CENTER_LEFT: utilizzata per la curva a destra e per l'atterraggio (30-34 71-75).
- CENTER_RIGHT: utilizzata per la curva a sinistra e per l'atterraggio (30-34 85-89).
- LEFTCORR: utilizzata per la correzione (50-54 71-80).
- RIGHTCORR: utilizzata per la correzione (50-54 80-89).
- RADANGLE: utilizzata per le curve (20-55 40-120).

Output:

- XOUT: asse verticale.
- YOUT: asse orizzontale.
- ZOUT: altezza.
- YAWOUT: imbardata.

Locali:

- DX: incremento della x.
- DY: incremento della y.
- ENDOFLINE: flag di fine percorso.
- VAR: variabile di appoggio.
- IY: termine integrale del controllore della y.
- IX: termine integrale del controllore della x.
- IY_INST: termine integrale attuale del controllore della y.
- IX_INST: termine integrale attuale del controllore della x.
- PX: termine proporzionale del controllore della x.
- PY: termine proporzionale del controllore della y.
- PIX: controllore della x.
- PIY: controllore della x.
- TURN: variabile di appoggio per la rotazione.
- C: variabile di appoggio.
- ANGLE: variabile di appoggio per la rotazione.

Costanti:

- KP: guadagno proporzionale, pari a 0,02.
- KI: guadagno integrale, pari a 0,002.
- LOWERIX_LIMIT: limite inferiore parte integrale di x, pari a -0,1.
- LOWERIY_LIMIT: limite inferiore parte integrale di y, pari a -0,1.
- UPPERIX_LIMIT: limite superiore parte integrale di x, pari a 0,1.
- UPPERIY_LIMIT: limite superiore parte integrale di y, pari a 0,1.
- VEL: incremento massimo.

6.3 Take_Off

Il *TAKE_OFF* è lo stato da cui lo sviluppo dell'algoritmo ha inizio; questo si può facilmente vedere dalla figura 6.4, dato che sopra al blocco è presente la transizione di default, che indica appunto il primo stato in esecuzione.
Inoltre, questo è lo stato più semplice del diagramma perché contiene una sola istruzione, ovvero zout=zout-0.01. Attraverso questa riga di codice si riesce a far sollevare il Mambo di 0.01 metri alla volta.
L'unica transizione collegata a questo stato ha come condizione zout<=-1.1 e porta allo stato *MOTION*. Ciò vuol dire che una volta raggiunta l'altezza di 1.1 m il flusso uscirà dallo stato *TAKE_OFF* per passare allo stato *MOTION*.

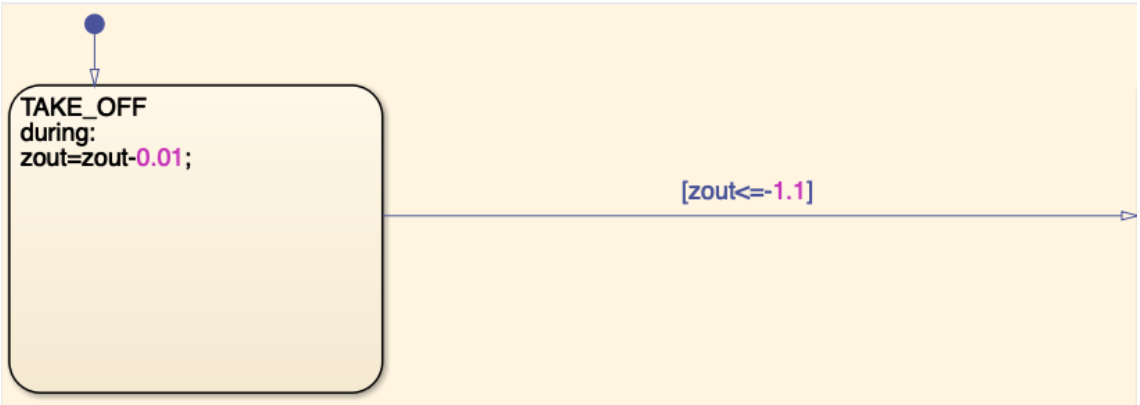


Figura 6.4: Stato Take_Off

6.4 Motion

MOTION è lo stato più complesso all'interno del programma, dato che contiene molte più istruzioni degli altri stati. Ciò è dovuto al fatto che attraverso questo stato si gestisce tutto il movimento del drone al di sopra della linea, esclusa la parte della curva e dell'atterraggio. A differenza dello stato *TAKE_OFF*, che ha solo la parte di during, questo stato ha anche l'entry, in cui vengono inizializzare a zero le variabili DX, DY, VAR e C.

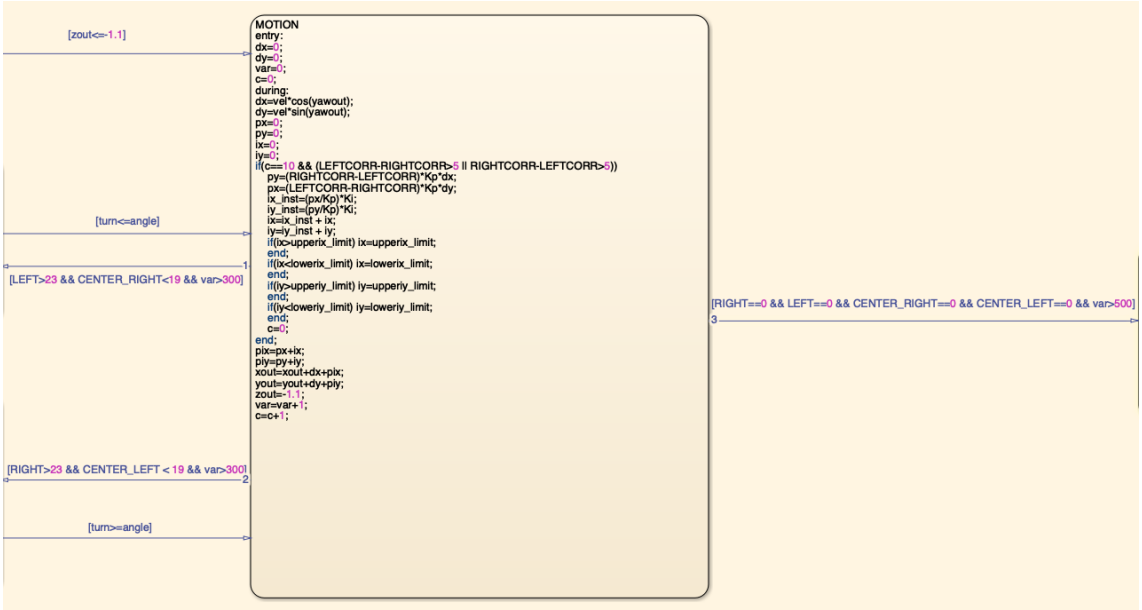


Figura 6.5: Stato Motion

Le prime due istruzioni di during sono quelle che assegnano a DX e DY i rispettivi valori, con i comandi

$$\text{DX} = \text{VEL} \cdot cos(\text{YAWOUT}); \tag{6.1}$$

$$\text{DY} = \text{VEL} \cdot sin(\text{YAWOUT}); \tag{6.2}$$

Ogni sezione del percorso può essere immaginata come l'ipotenusa di un triangolo rettangolo, per cui risulta facile calcolarne i due cateti, che corrispondono allo spostamento orizzontale e a quello verticale. I casi degeneri corrispondono alle sezioni verticali e orizzontali, in cui DX e DY assumono i valori massimi (0.00055) e minimi (0). E' stato fatto ciò in modo da avere in ogni sezione lo stesso incremento e, di conseguenza, la stessa velocità lungo tutto il percorso. Inizialmente era stato previsto un sistema di calcolo con la tangente, poi scartato a causa del comportamento difficile da gestire a $\pi/2 + k\pi$ (dove tende ad infinito).
Nella tabella 6.1 che segue si può trovare un riassunto di tutti i casi possibili.

Sezione	Variabile	Valore
Verticale	DX	$0.00055 \cdot cos(0°) = 0.00055$
Verticale	DY	$0.00055 \cdot sin(0°) = 0$
Orizzontale	DX	$0.00055 \cdot cos(90°) = 0$
Orizzontale	DY	$0.00055 \cdot sin(90°) = 0.00055$
Obliqua	DX	$vel \cdot cos(yawout)$
Obliqua	DY	$vel \cdot sin(yawout)$

Tabella 6.1: Calcolo DX e DY per ogni sezione

In seguito si trova un ciclo if che gestisce il posizionamento del minidrone sopra alla linea, in modo da correggere qualsiasi variazione di posizione.
La condizione 6.3 per entrare all'interno del ciclo è formata da più espressioni, collegate tra loro da un AND e un OR. Dopo vari test attraverso il simulatore sono stati trovati i valori più adatti a gestire il controllo del posizionamento; per questo, da una parte bisogna avere almeno dieci "cicli di codice" e dall'altra serve un errore tra le finestre destra e sinistra (e viceversa) pari ad almeno cinque. Ciò serve ad evitare di correggere la posizione ad ogni ciclo, dando così il tempo al comando precedente di agire, evitando di sovrapporre più condizioni.

$$if(\text{C} == 10 \&\& (\text{LEFTCORR} - \text{RIGHTCORR} > 5 || \\ ||\text{RIGHTCORR} - \text{LEFTCORR} > 5)) \quad (6.3)$$

La correzione è di tipo PI, quindi proporzionale-integrale. Il proporzionale è calcolato moltiplicando l'errore, vale a dire la differenza tra RIGHTCORR e LEFTCORR (in PY,viceversa per PX) per la costante KP e l'incremento DX (DY nel caso di PX). Le due finestre che poi producono le variabili usate per calcolare l'errore sono LEFT_CORRECTION e RIGHT_CORRECTION che dividono la linea sottostante a metà. L'ultimo elemento serve a dare alla variabile proporzionale un elemento per annullarla nei casi limite: infatti nel caso di un percorso verticale la correzione deve essere fatta solo sull'asse y (orizzontale), per cui la correzione PX è nulla perchè DY è pari a zero (sin(0°)=0).

$$\begin{aligned} py &= (\text{RIGHTCORR} - \text{LEFTCORR}) \cdot \text{KP} \cdot \text{DX}; \\ px &= (\text{LEFTCORR} - \text{RIGHTCORR}) \cdot \text{KP} \cdot \text{DY}; \end{aligned} \quad (6.4)$$

Successivamente si calcola la parte integrale del controllore, utilizzando le stesse variabili di PX, ma moltiplicando per KI anziché per KP. La variabile IX è poi confrontata con un limite superiore ed uno inferiore, per realizzare il cosiddetto schema anti win-

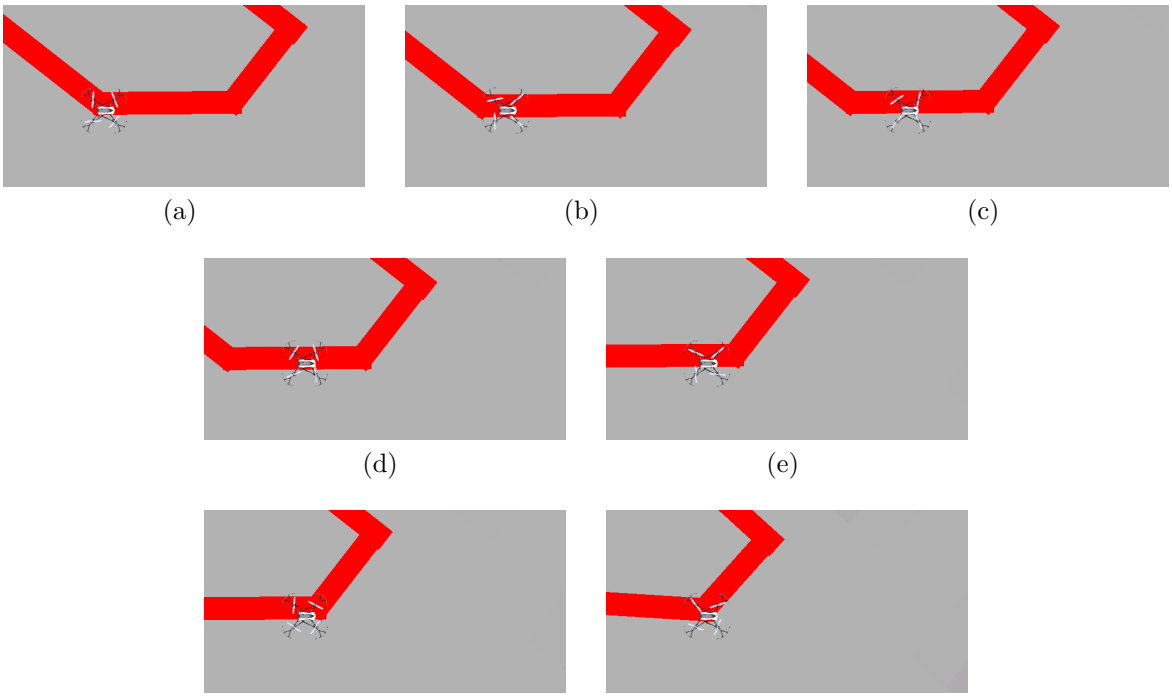
dup. Come approfondito in [11] e [16], il windup è quel fenomeno dovuto all'integratore del PID che continua ad integrare l'errore creando un segnale di controllo saturo.

$$
\begin{aligned}
&\text{IX\_INST} = (px/Kp) \cdot \text{KI};\\
&\text{IY\_INST} = (py/Kp) \cdot \text{KI};\\
&\text{IX} = (\text{IX\_INST} + \text{IX});\\
&\text{IY} = (\text{IY\_INST} + \text{IY});\\
&if(\text{IX}>\text{UPPERIX\_LIMIT})\ \text{IX} = \text{UPPERIX\_LIMIT};end;\\
&if(\text{IX}<\text{LOWERIX\_LIMIT})\ \text{IX} = \text{LOWERIX\_LIMIT};end;\\
&if(\text{IY}>\text{UPPERIY\_LIMIT})\ \text{IY} = \text{UPPERIY\_LIMIT};end;\\
&if(\text{IY}<\text{LOWERIY\_LIMIT})\ \text{IY} = \text{LOWERIY\_LIMIT};end;
\end{aligned}
\tag{6.5}
$$

Infine si calcola la variabile di controllo totale, data da PX+IX e si aggiornano sia i valori di XOUT e YOUT, sommando il valore di x e y precedente, l'incremento DX e DY e il controllo PIX e PIY, sia le variabili di appoggio VAR e C.

$$
\begin{aligned}
&\text{PIX}= \text{PX} + \text{IX};\\
&\text{PIY}= \text{PY} + \text{IY};\\
&\text{XOUT}= \text{XOUT} + \text{DX} + \text{PIX};\\
&\text{YOUT}= \text{YOUT} + \text{DY} + \text{PIY};\\
&\text{ZOUT}= \text{-1.1};\\
&\text{VAR}= \text{VAR} + 1;\\
&\text{C}= \text{C} + 1;
\end{aligned}
\tag{6.6}
$$

Questo stato riceve tre transizioni in ingresso, una da *TAKE_OFF* e due dagli stati delle rotazioni. Inoltre, ha tre transizioni in uscita, una per *LEFT_ROTATION*, una per *RIGHT_ROTATION* e l'ultima per *LANDING*. Come si intuisce dal diagramma di flusso 6.2, il programma prima controlla la validità della condizione per la curva a sinistra, poi per quella a destra e infine per l'atterraggio. La stessa cosa si può verificare dai numeri delle transizioni che escono dallo stato *MOTION*, che indicano appunto l'ordine di esecuzione.
Attraverso le immagini che seguono si può vedere l'effetto della correzione, che tende a riportare il drone verso il centro del percorso. Infatti, partendo dalla stessa posizione (figure 6.6a e 6.7a), si arriva ad avere una posizione più centrata nel drone con correzione (figura 6.7) rispetto a quello senza correzione, che rimane attaccato al bordo inferiore del tracciato (figura 6.6).
Più la sezione sarà lunga e più si avrà un effetto visibile, dato che già di per se il Mambo tende ad avere un'oscillazione durante il volo.

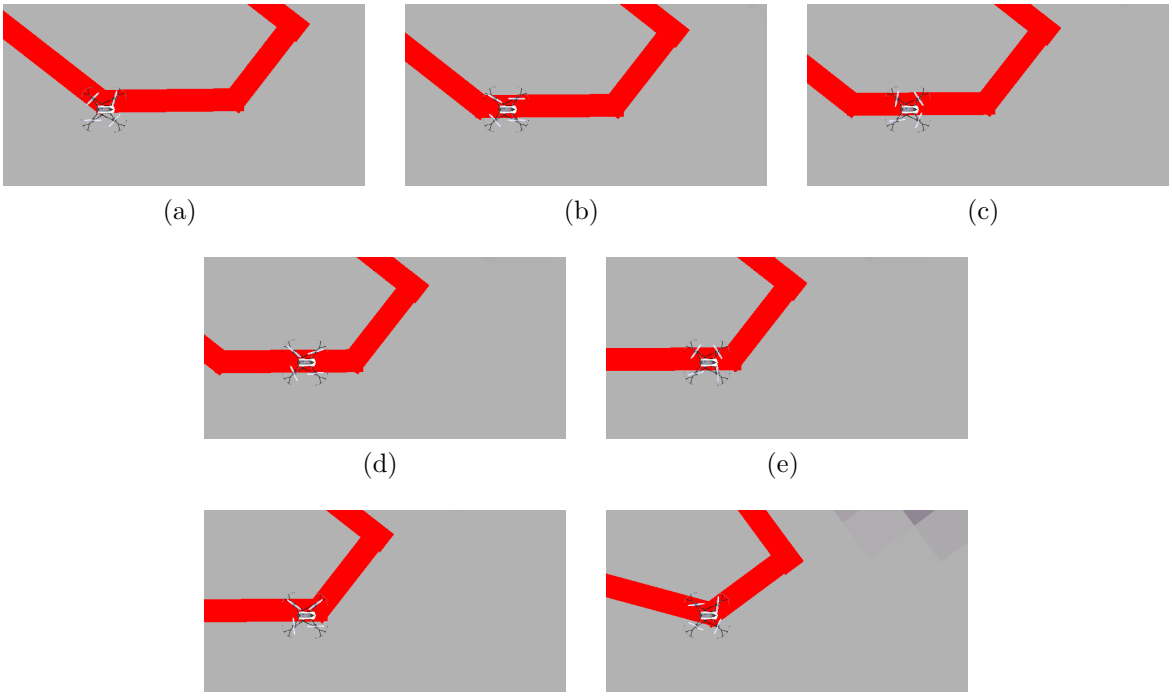


(a) (b) (c)

(d) (e)

(f) (g)

Figura 6.6: Sezione senza correzione



(a) (b) (c)

(d) (e)

(f) (g)

Figura 6.7: Sezione con correzione

6.5 Left_Rotation

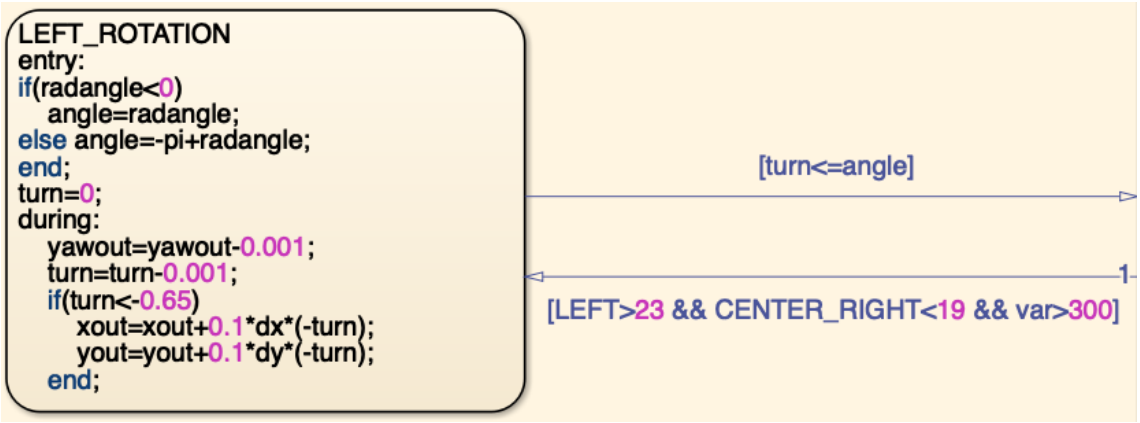


Figura 6.8: Stato Left_Rotation

Questo stato serve appunto per la curva a sinistra e vi si entra da *MOTION* quando:

$$\text{LEFT}>23 \;\&\&\; \text{CENTER\_RIGHT}<19 \;\&\&\; \text{VAR}>300 \tag{6.7}$$

ovvero quando la finestra sinistra vede una buona quantità di rosso, quella di centro-destra ne vede meno e la variabile contatore VAR abbia superato 300 (ciò evitava in alcuni casi particolari di permettere al Mambo di eseguire due rotazioni in successione). Anche in questo caso abbiamo delle entry actions, che servono a valutare il valore dell'angolo della curva a cui ci si sta avvicinando. Se l'angolo è negativo allora lo si può associare direttamente alla variabile ANGLE. Qualora l'angolo risulti non negativo bisogna sottrarre 180° al valore letto, questo perché l'algoritmo di Canny restituisce valori positivi quando l'angolo reale è minore di -90°.
Le during action servono a far ruotare il drone sopra la curva, con il comando $yawout = yawout - 0.001$. Inoltre, dato che nel simulatore il drone tende ad arretrare leggermente mentre curva, è stata aggiunta una condizione per cui superato un certo angolo (pari a -0,65 radianti), il Mambo continua a muoversi nella stessa direzione da cui proveniva, facendo in modo che si centri meglio sopra la curva. Questa funzione è illustrata per la rotazione a destra nel paragrafo successivo, attraverso le figure 6.10 e 6.11.

$$\begin{aligned} &\text{IF}(\text{TURN}<\text{-}0.65)\\ &\text{XOUT}=\text{XOUT} + 0.1 \cdot \text{DX} \cdot (\text{-TURN});\\ &\text{YOUT}=\text{YOUT} + 0.1 \cdot \text{DY} \cdot (\text{-TURN});\\ &\textit{end};\end{aligned} \tag{6.8}$$

Come si può notare dal codice, l'ultimo fattore corrisponde proprio all'angolo, per cui ad un angolo maggiore si avrà una maggiore "spinta" sopra la curva.
La condizione di uscita da questo stato è TURN<=ANGLE, ovvero quando il drone ha ruotato di un valore pari all'angolo calcolato prima della curva, allora può ripartire da *MOTION*.

6.6 Right_Rotation

Per quanto riguarda lo stato *RIGHT_ROTATION*, il funzionamento è lo stesso del precedente, con la differenza che si hanno delle finestre diverse sulla condizione (RIGHT e CENTER_LEFT) e dei segni invertiti all'interno del codice, come in RADANGLE e nei calcoli di YAWOUT, TURN, XOUT e YOUT. Questo blocco è collegato in uscita solamente con *MOTION*, attraverso TURN>=ANGLE.

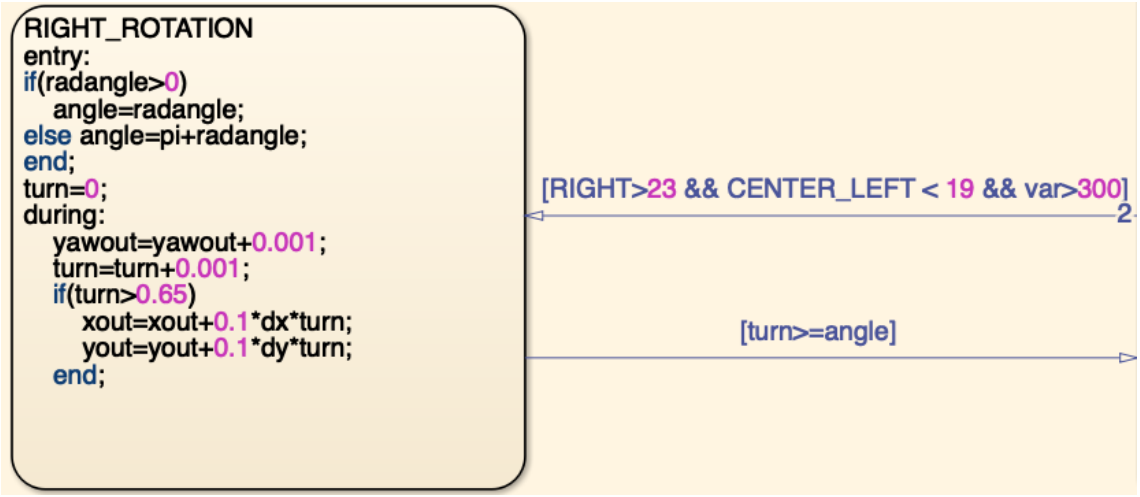
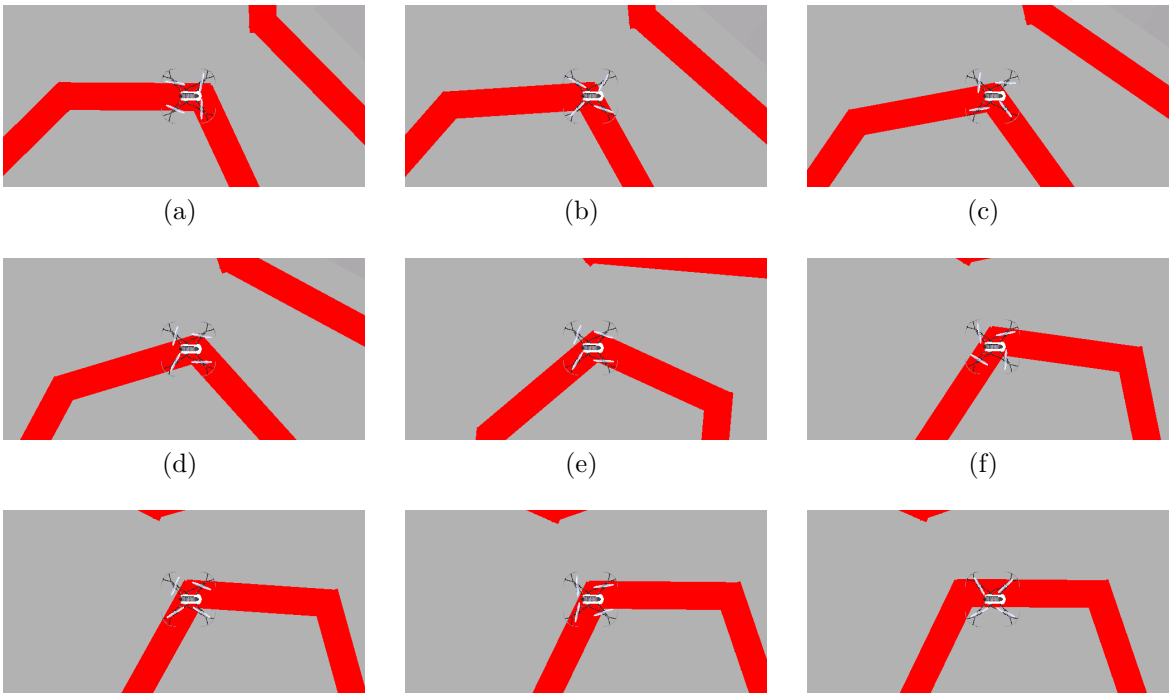


Figura 6.9: Stato Right_Rotation

Attraverso le figure sottostanti è possibile vedere l'evoluzione di una curva a destra. Qualora invece si togliessero dal codice della curva le ultime quattro istruzioni (le istruzioni 6.8 nel caso di curva a sinistra), nel simulatore il Mambo tornerebbe leggermente indietro mentre esegue la rotazione, risultando poi fuori centro nel momento in cui deve ripartire per la sezione successiva, come si può notare dalla seguente sequenza, e in particolare confrontando le figure 6.10i e 6.11i, dove si può vedere come nella prima il drone sia esattamente al centro della riga, mentre nel secondo caso sia più vicino al bordo inferiore.

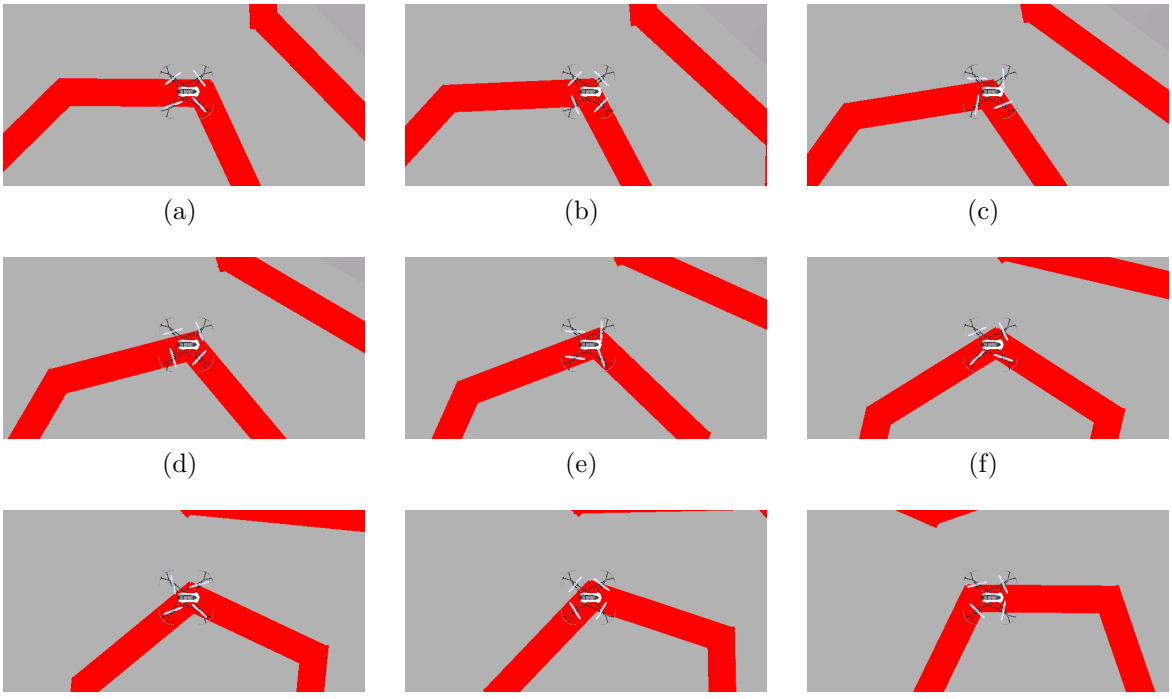


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 6.10: Curva senza correzione



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 6.11: Curva con correzione

6.7 Landing

La condizione che fa entrare nello stato di atterraggio è la seguente:

$$\text{RIGHT}==0 \;\&\&\; \text{LEFT}==0\&\& \\ \&\&\text{CENTER\_RIGHT}==0 \;\&\&\; \text{CENTER\_LEFT} \;\&\&\; \text{VAR} > 500 \quad (6.9)$$

Ciò si verifica quando il drone ha superato l'ultima sezione del percorso, per cui ai suoi lati non vede rosso e neanche attraverso le due finestre centrali, come in 6.13. Questo stato fa in modo che il Mambo continui il suo percorso nello stessa direzione da cui proveniva, attraverso le prime due righe di codice. Poi, valutando sia la quantità di rosso che vede la finestra CENTER sia il flag ENDOFLINE (che deve essere true, ovvero il percorso è terminato) si può far atterrare il drone, settando a 0 l'incremento lungo gli assi x e y e decrementando gradualmente la variabile z dell'asse verticale, attraverso l'istruzione 6.10:

$$\text{ZOUT}=\text{ZOUT}+0.007; \quad (6.10)$$

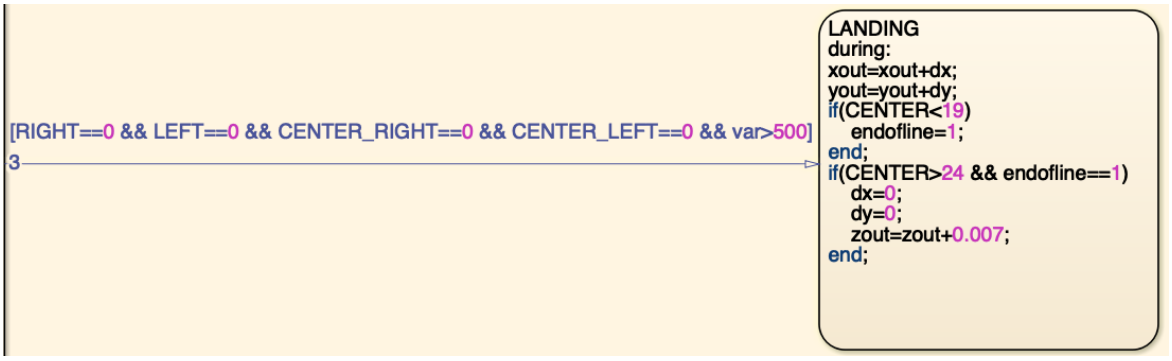
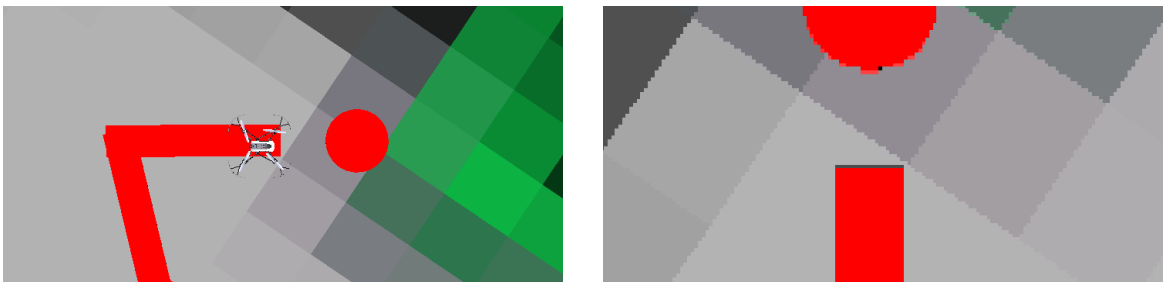


Figura 6.12: Stato Landing



(a) Vista dall'alto (b) Vista dal drone

Figura 6.13: Atterraggio

Capitolo 7

Conclusioni e sviluppi futuri

Come si può vedere dai seguenti link, il Mambo risponde bene a diverse tipologie di percorso, seguendo la traiettoria in modo preciso e correggendo la propria posizione laddove ce ne fosse bisogno.

Primo percorso Secondo percorso Terzo percorso

Questo però non implica che non possano essere fatte delle migliorie, sia per quanto riguarda la parte del simulatore sia per il volo reale, il quale ad esempio presenta problemi differenti rispetto al volo su Simulink. Infatti nella realtà si incontrano maggiori difficoltà soprattutto per quanto riguarda la lettura dell'immagine vista dal drone (sarebbe meglio aver un pavimento non uniforme a livello di colore e una luce che non varia tra le sezioni del percorso) e per una diversa risposta alla pesantezza computazionale del codice (nel volo reale c'è una maggiore lentezza).
In entrambi i casi, comunque, due miglioramenti realizzabili sono un upgrade della correzione, ovvero del controllo PI, e una modifica della logica della curva, che permette al drone di non fermarsi sopra alla curva e poi ruotare ma di eseguire un movimento continuo.

Bibliografia

[1] M. Basso and E. P. de Freitas. A uav guidance system using crop row detection and line follower algorithms. *Journal of Intelligent & Robotic Systems*, pages 1–17, 2019.

[2] Á. Bello Guisado. *Diseño de controladores de vuelo para un dron modelo PARROT Mambo Minidrone*. PhD thesis, 2019.

[3] B. Bona. Rotazioni http://www.ladispe.polito.it/corsi/meccatronica/02jhcor/2011-12/slides/rotazioni.pdf, 2008.

[4] S. Bouabdallah. Design and control of quadrotors with application to autonomous flying. Technical report, Epfl, 2007.

[5] T. Bresciani. Modelling, identification and control of a quadrotor helicopter. *MSc Theses*, 2008.

[6] R. Bucher. Introduzione a stateflow, 2013.

[7] G. Carratelli and M. Del Duca. Controllo di un quadcopter, 2012.

[8] A. Chovancová, T. Fico, L. Chovanec, and P. Hubinsk. Mathematical modelling and parameter identification of quadrotor (a survey). *Procedia Engineering*, 96(2014):172–181, 2014.

[9] D. García-Olvera, A. Hernández-Godínez, B. Nicolás-Trinidad, and C. Cuvas-Castillo. Line follower with a quadcopter. *Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI*, 7:30–36, 01 2020.

[10] B. Green. Canny edge detection tutorial. *Retrieved: March*, 6:2005, 2002.

[11] J. L. Guzmán, K. Astrom, S. Dormido, T. Hagglund, and Y. Piguet. Interactive learning modules for pid control. *ACE*, 6, 2006.

[12] T. Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.

[13] MathWorks. Mathworks minidrone competitions, 2019.

[14] S. Musa. Techniques for quadcopter modelling & design: A review. 5, 05 2018.

[15] A. RUSSO. Adaptive control of multirotor uavs. 2016.

[16] H.-B. Shin and J.-G. Park. Anti-windup pid controller with integral state predictor for variable-speed motor drives. *IEEE Transactions on Industrial Electronics*, 59(3):1509–1516, 2011.

[17] S. Sural, G. Qian, and S. Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. In *Proceedings. International Conference on Image Processing*, volume 2, pages II–II. IEEE, 2002.

[18] F. Tortorella. La trasformata di hough.

[19] Unicas. Estrazione dei bordi.

Ringraziamenti
Vorrei ringraziare per primi il Professor Ippoliti, il Professor Mancini ed il Professor Orlando per il supporto e la disponibilità di questi mesi.
Poi un grande grazie va ai coinquilini di questi tre anni: Marco, Loy, Davide, Francesco, Filippo e Nicolò. La convivenza con loro ha sicuramente reso questo viaggio più leggero e divertente. Non posso non ringraziare tutti i ragazzi e le ragazze del gruppo 'Osso Ioide' e non che in questi anni hanno sopportato le mie vicende universitarie. Come non ringraziare poi tutti i compagni ingegneri con cui ho condiviso "gioie e dolori"; menzione speciale per Sandro, compagno di mille progetti e avversario di altrettanti calcetti, e Paolo, collega di questi ultimi mesi.
Last but not least, il grazie più grande va a mia sorella, mia madre e mio padre, per aver appoggiato ogni mia scelta e per aver avuto sempre una parola di incoraggiamento quando serviva (ma soprattutto per avermi fatto Interista).