Universitá Politecnica delle Marche

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Studio e sviluppo di algoritmi di elaborazione delle immagini per minidroni

Study and development of an image processing algorithm for minidrones

Relatore:
Prof. Gianluca Ippoliti

Correlatore:
Prof. Giuseppe Orlando

Candidato:
Fabrizio Pilone
Matricola 1086830

Anno Accademico 2020-2021

Alla mia famiglia, che grazie a tanti sacrifici mi ha portato fin qui,
in particolare a mia madre.
Ai miei amici, che hanno reso questo percorso meraviglioso.
Grazie a tutti voi.

Sommario

All'interno della tesi si procederà, come prima cosa, ad una panoramica generale del mondo dei quadricotteri al fine di comprendere il soggetto della trattazione. Si procederà quindi alla descrizione del moto dei velivoli e delle equazioni che ne regolano la dinamica. Una volta comprese le forze e le equazioni che si stanno utilizzando, si darà una presentazione del modello parrotMinidroneCompetition, un modello Simulink creato da MathWorks all'interno dell'omonima competizione. Tale modello permette di simulare un drone Mambo dell'azienda Parrot e quindi di progettare algoritmi di inseguimento, algoritmi di elaborazione delle immagini e controllori per i 6 gradi di libertà del drone con la semplicità data da Simulink, e quindi di poter testare in ambiente virtuale le features progettate al calcolatore, prima di implementarle sul drone reale. Proprio grazie a questo ambiente è stato possibile simulare ed utilizzare la videocamera posta nella parte inferiore del drone Mambo, al fine di realizzare un algoritmo che permettesse al drone di seguire un percorso realizzato al suolo. Tale algoritmo si compone di una parte di elaborazione delle immagini e analisi dei dati (che sfrutta l'algoritmo di Canny, le trasformate di Hough e i relativi picchi per identificare i bordi del percorso), e una parte di pilotaggio del drone basato sulle informazioni raccolte nello stadio precedente. Una volta analizzato l'algoritmo, al fine di dimostrare le numerose possibilità date dal modello Simulink, è stato realizzato un controllore per uno dei gradi di libertà del drone per sostituire il PID montato a bordo del velivolo.

Indice

Elenco delle tabelle III

Elenco delle figure IV

1 Droni e quadricotteri 1
- 1.1 Introduzione . . . 1
- 1.2 Caratteristiche di volo . . . 4
- 1.3 Realizzazione dei movimenti . . . 6
- 1.4 Parrot Mambo . . . 9
  - 1.4.1 Motori coreless . . . 11

2 Modello matematico del drone 13
- 2.1 Angoli di Tait-Bryan . . . 13
- 2.2 Equazioni matematiche . . . 15

3 Modello Simulink 23
- 3.1 Installazione e avviamento del progetto . . . 23
- 3.2 Flight Simulation Model . . . 25
  - 3.2.1 Command . . . 26
  - 3.2.2 Sensors Model . . . 27
  - 3.2.3 Environment Model . . . 28
  - 3.2.4 Multicopter Model . . . 30
  - 3.2.5 Simulink 3D Visualization . . . 35
  - 3.2.6 Flight Control System . . . 36

4 Algoritmo di elaborazione delle immagini 41
- 4.1 HSV . . . 41
- 4.2 Rilevamento dei bordi . . . 42
- 4.3 Stateflow . . . 45
- 4.4 Image Processing System . . . 46
- 4.5 Path Planning . . . 49
- 4.6 Percorsi scelti . . . 52

5 PID e controllori 55
5.1 PID 55
5.2 Luogo delle radici 58
5.3 Sintesi di un controllore per la z con il luogo delle radici 60

6 Conclusioni 69

A Modello lineare MATLAB 71

B Codice sviluppato 75
B.1 HSV Filter 75
B.2 Center of gravity 76
B.3 Area of piloting 77
B.4 Edge and angle detection 78

Elenco delle tabelle

1.1 Tabella dei requisiti per droni 2
1.2 Specifiche del Mambo 9
1.3 Indicatori luminosi Mambo 10
4.1 Tabella delle azioni di uno Stateflow 45

Elenco delle figure

1.1 Il mito di Icaro . . . 1
1.2 La vite aerea di Leonardo . . . 1
1.3 Il drone Ingenuity su suolo marziano . . . 2
1.4 Alcune tipologie di drone . . . 3
1.5 Tipologie di configurazione del quadricottero . . . 4
1.6 Roll, pitch e yaw . . . 5
1.7 Movimento lungo l'asse z . . . 6
1.8 Pitch configurazione a più (in avanti e all'indietro) . . . 7
1.9 Pitch configurazione a croce (in avanti e all'indietro) . . . 7
1.10 Roll configurazione a più (a destra e a sinistra) . . . 8
1.11 Roll configurazione a croce (a destra e a sinistra) . . . 8
1.12 Yaw configurazione a croce . . . 9
1.13 Motori coreless e ironcore a confronto . . . 11
2.1 Terna inerziale e terna mobile . . . 14
2.2 Angoli di Tait-Bryan . . . 15
3.1 Installazione nuovo pacchetto . . . 24
3.2 Flight Simulation Model . . . 25
3.3 Blocco Command . . . 26
3.4 Sotto-sistema Signal Builder . . . 27
3.5 Blocco Sensors Model . . . 27
3.6 Sensori camera, IMU e pressione . . . 28
3.7 Sensore camera . . . 28
3.8 Sensori IMU nelle due modalità . . . 29
3.9 Environment Model . . . 29
3.10 Environment nelle due modalità . . . 30
3.11 Scelta modello quadricottero . . . 31
3.12 Modello non lineare . . . 32
3.13 Blocco AC Model . . . 32
3.14 Forza di gravità ed effetto drag . . . 33
3.15 Calcolo di forze e momenti generati dai rotori . . . 34
3.16 Modello lineare . . . 35
3.17 Blocco di visualizzazione . . . 35

3.18 Processamento immagini e sistema di controllo 36
3.19 Control System 37
3.20 Filtro di Kalman per l'altitudine 37
3.21 Stimatori degli stati 38
3.22 PID per gli angoli di pitch e roll 39
3.23 Controllori del quadricottero 40
4.1 Spazio colore HSV 42
4.2 Algoritmo di Canny su immagine di giunto a vapore 44
4.3 Trasformata di Hough 44
4.4 Image Processing System 46
4.5 Sezione rettilinea di percorso 47
4.6 App colorThresholder 47
4.7 Divisione in zone del percorso precedente 49
4.8 Sottosistema Path Planning 50
4.9 Diagramma a stati in Path Planning 51
4.10 Trasformazione da sforzo di controllo in spinta 52
4.11 3 percorsi per la fase di testing dell'algoritmo 53
5.1 Diagrammi di Bode di F(s) 62
5.2 Luogo delle radici di F(s) 63
5.3 Risposta al gradino di Fs 63
5.4 Blocco controllore asse z 64
5.5 Errore sulla z con PID e luogo delle radici 65
5.6 Indici di prestazione con PID e luogo delle radici 66
5.7 Stima della z con PID e luogo delle radici 67
6.1 Utilizzo dell'algoritmo in campo agrario 69
A.1 Matrice A 71
A.2 Matrice B 71
A.3 Matrice C 72
A.4 Matrice D 73
A.5 Vettori di stato, di ingresso e di uscita 74

Capitolo 1

Droni e quadricotteri

1.1 Introduzione

L'interesse dell'uomo per gli oggetti volanti nasce in tempi remoti (basti pensare al mito di Icaro o alla vite aerea di Leonardo) e giunge praticamente inalterato ai giorni nostri dopo secoli di studi ed invenzioni in tal senso.

Figura 1.1: Il mito di Icaro

Figura 1.2: La vite aerea di Leonardo

Un aeromobile a pilotaggio remoto (APR), comunemente detto drone (in inglese UAV-Unmanned Aerial Veichle, o RPA-Remotely Piloted Aircraft) è un apparecchio volante caratterizzato dall'assenza del pilota a bordo il cui volo è governato da varie tipologie di Flight Control System, gestiti in remoto da piloti a terra o automaticamente da calcolatori.

Il primo tentativo di costruzione di un APR risale addirittura al 1849, quando gli Austriaci attaccarono Venezia utilizzando dei palloni caricati di esplosivo. Come facilmente intuibile, lo sviluppo principale a tale tecnologia si ebbe in ambito militare, prima durante la I Guerra Mondiale, e poi durante la II Guerra Mondiale, la Guerra Fredda e la Guerra in Vietnam.

Tuttavia, a partire dagli anni duemila, grazie all'enorme progresso tecnologico registrato, gli APR hanno cominciato ad essere impiegati in numerosi campi in ambito civile, come il monitoraggio di siti archeologici, di impianti industriali e

della fauna, o il telerilevamento e l'aerofotogrammetria, o ancora le operazioni di ricerca e soccorso, la fotografia e in generale a scopo ludico.

Anche in ambito spaziale iniziano ad avere sempre più importanza, tanto che nell'ultima missione verso Marte è stato utilizzato un drone, *Ingenuity*, come dimostratore tecnologico al fine di ottenere informazioni dagli strumenti a bordo e immagini dalle numerose telecamere montate a bordo del drone stesso e sul rover *Perseverance* (immagini consultabili sul sito della *NASA*).

Figura 1.3: Il drone Ingenuity su suolo marziano

Il sempre crescente interesse nei confronti di questi oggetti volanti, ha spinto le industrie a progettare droni sempre più piccoli ed economici, fino ad arrivare agli attuali droni in commercio: delle dimensioni di una mano, pesanti qualche centinaio di grammi e costosi poche decine di euro.

Alcune delle caratteristiche tipiche di un drone sono quelle riportate in tabella 1.1

Tabella 1.1: Tabella dei requisiti per droni

Specifica	Requisiti
Grandezza	<15cm
Peso	<100g
Raggio	tra 1 e 10 km
Durata	60min
Altitudine	<150m
Carico	20g
Costo	<1500€

La prima, e più evidente, classificazione dei droni può essere fatta in base al sistema di volo utilizzato: possono essere ad ala fissa o ad ala rotante.

I droni ad ala fissa sono simili a piccoli aerei e sono caratterizzati da una struttura relativamente semplice. Il controllo in volo è realizzato grazie a superfici realizzate nell'ala e nella coda. Le principali caratteristiche di questa tipologia di droni sono:

- Elevata autonomia di volo: sono per questo utilizzati ad esempio nella ricognizione e nello spionaggio
- Elevata sicurezza di volo: in caso di avaria del motore possono planare attutendo l'impatto con il suolo
- Carico utile relativamente grande
- Costo elevato
- Impossibilità di rimanere in volo stazionario
- Impossibilità di decollo e atterraggio verticale (VTOL-Vertical Take-Off and Landing)

I droni ad ala rotante sono caratterizzati da eliche (a passo fisso o variabile) che, girando ad alte velocità, generano la spinta necessaria al sollevamento del velivolo. Possono essere dotati di un solo rotore, il che li rende simili a piccoli elicotteri a pilotaggio semplificato, o di più rotori, e vengono per questo chiamati multicotteri. Il multicottero più facile da trovare in commercio è il quadricottero, caratterizzato da elevata stabilità grazie ai 4 rotori posti in simmetria, possibilità di trasportare carichi relativamente elevati e basso costo, limitando tuttavia l'autonomia di volo.

(a) *Drone ad ala fissa* (b) *Tricottero* (c) *Quadricottero*

Figura 1.4: Alcune tipologie di drone

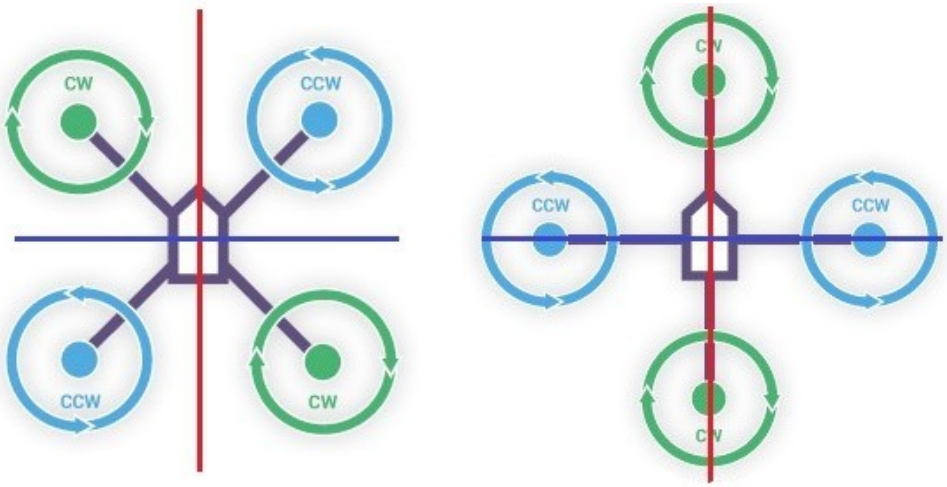
1.2 Caratteristiche di volo

In questa sezione si analizzeranno le caratteristiche di volo dei quadricotteri, riferendosi in particolare al drone Parrot Mambo.

I quadricotteri sono costituiti da un corpo centrale in metallo leggero e 4 bracci, ognuno dei quali terminante con un motore elettrico che mette in rotazione un'elica; rotori opposti rispetto al braccio generano una coppia

La prima classificazione che è possibile fare sui quadricotteri, è la posizione dei 4 rotori rispetto al sistema di riferimento del corpo:

- Configurazione a più: i 4 rotori sono allineati agli assi x e y del corpo, ovvero si hanno 2 rotori in corrispondenza del naso e della coda del drone, e gli altri due sono posti ortogonalmente ai precedenti
- Configurazione a croce: i 4 rotori sono disallineati di 45° rispetto agli assi x e y del corpo, ovvero il naso, così come la coda, si trova in posizione intermedia tra 2 rotori



(a) *Configurazione a più* (b) *Configurazione a croce*

Figura 1.5: Tipologie di configurazione del quadricottero

La seconda classificazione riguarda le eliche montate sul quadricottero, le quali, come è possibile osservare dalla Figura 1.5, possono essere di due tipi e sono poste sullo stesso braccio:

- *Tipo A* (Antiorario): producono spinta verso l'alto girando in senso antiorario. Nella figura precedente sono segnati dal colore celeste
- *Tipo O* (Orario): producono spinta verso l'alto girando in senso orario. Nella figura precedente sono segnati dal colore verde

L'effetto di questa distinzione è quello di bilanciare la rotazione generata dalla conservazione del momento angolare (effetto giroscopico): le eliche, ruotando, generano un momento in verso opposto al verso di rotazione delle eliche stesse; se le 4 eliche ruotano tutte alla stessa velocità, il momento generato dalle due eliche di tipo A viene bilanciato da quello delle eliche di tipo O. Questo è proprio il compito del motore di coda di un normale elicottero, che pertanto viene reso non necessario nei quadricotteri.

Grazie alla loro modalità costruttiva, i quadricotteri hanno a disposizione 6 gradi di libertà, 3 per movimenti lineari e 3 per movimenti angolari:

1. Movimento lineare longitudinale
2. Movimento lineare trasversale
3. Movimento lineare verticale
4. Movimento angolare attorno all'asse trasversale (angolo di pitch, o beccheggio)
5. Movimento angolare attorno all'asse longitudinale (angolo di roll, o rollio)
6. Movimento angolare attorno all'asse verticale (angolo di yaw, o imbardata)

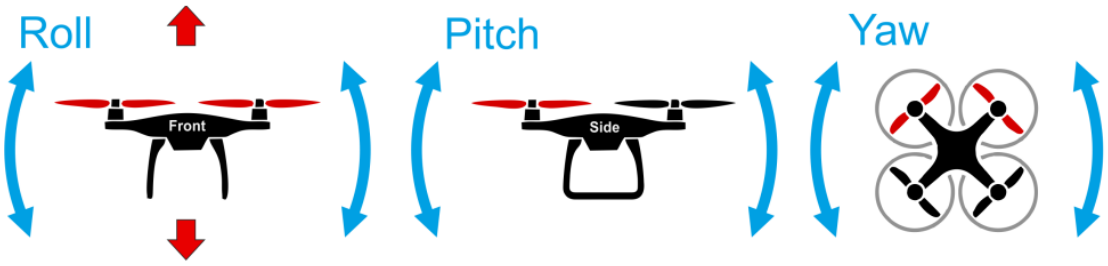


Figura 1.6: Roll, pitch e yaw

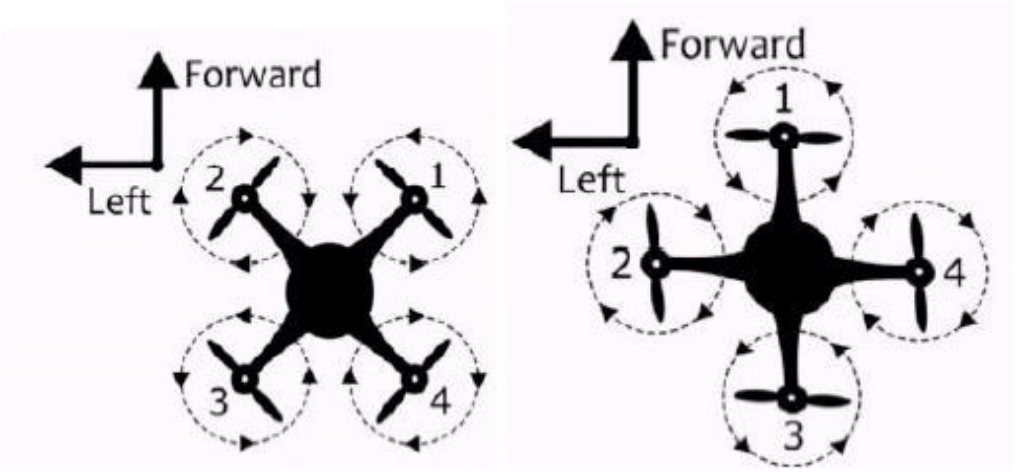
Come è possibile notare, si hanno 6 gradi di libertà (DoF) ma solo 4 attuatori, il quadricottero è cioè un sistema sotto-attuato in cui non è possibile raggiungere un determinato set-point per tutti i 6 DoF, ma solo per 4 di essi. 4 gradi di libertà sono infatti accoppiati: una variazione dell'angolo di pitch è associata ad un movimento frontale, una variazione dell'angolo di roll è associata ad un movimento laterale, mentre movimento verticale e angolo di yaw sono indipendenti.

1.3 Realizzazione dei movimenti

Vediamo quindi come realizzare nella pratica i movimenti descritti nella sezione precedente.

- Movimento verticale

  Per ottenere un movimento solo lungo l'asse z, è sufficiente che tutti i rotori girino ad uguale velocità. In questo modo, per quanto descritto in precedenza, si annullerà l'effetto giroscopico e si genererà una spinta verso l'alto dipendente dalla velocità di rotazione delle eliche. Se ogni attuatore genera una spinta pari ad un quarto della forza peso a cui è sottoposto il drone, si otterrà un moto stazionario a mezz'aria, anche detto hovering; aumentando (o diminuendo) la velocità dei motori, si aumenterà (o diminuirà) la spinta generata dagli stessi, e si avrà così movimento verso l'alto (o verso il basso)



(a) *Configurazione a croce* (b) *Configurazione a più*

Figura 1.7: Movimento lungo l'asse z

- Movimento frontale

  Per analizzare il caso di movimento sull'asse longitudinale del drone, è necessario distinguere la configurazione a più da quella a croce

  – Configurazione a più

    Nella configurazione a più si ottiene movimento "in avanti" aumentando la velocità di rotazione del motore posto sulla coda del drone e diminuendo quella del rotore sul naso (rispettivamente i motori 3 ed 1 in Figura 1.8). In questo modo la spinta totale rimarrà inalterata permettendo al drone di mantenere la quota, e si otterrà una variazione dell'angolo di pitch che

metterà in moto il drone. Per far muovere il drone "all'indietro" basterà fare l'opposto, ovvero aumentare i giri del motore anteriore e ridurre quelli del motore posteriore

– Configurazione a croce

Nella configurazione a croce è necessario modificare la velocità di tutti e 4 i motori: i due posti nella parte anteriore diminuiranno i giri, i due posti sulla coda li aumenteranno (rispettivamente i motori 1-2 e 3-4 in Figura 1.9), viceversa per il movimento all'indietro

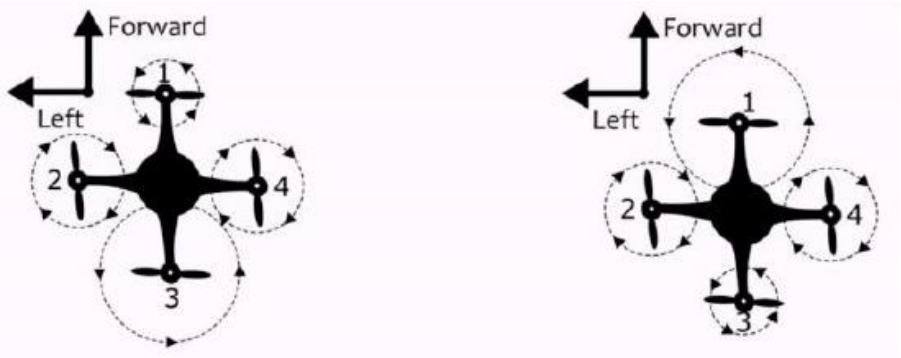


Figura 1.8: Pitch configurazione a più (in avanti e all'indietro)

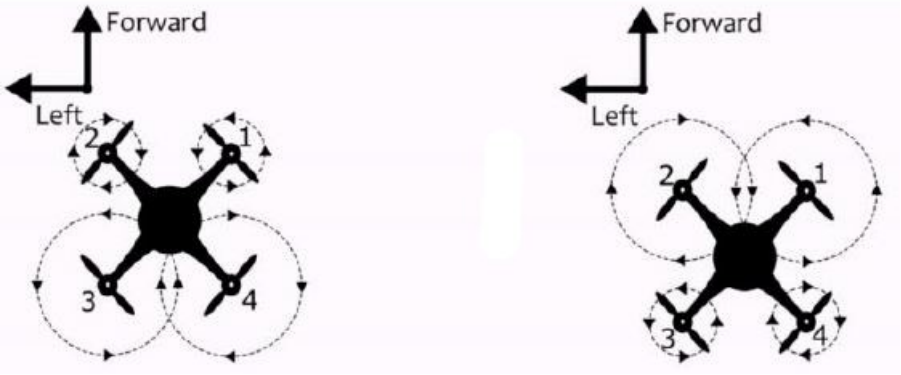


Figura 1.9: Pitch configurazione a croce (in avanti e all'indietro)

- Movimento laterale

Anche in questo caso distinguiamo le due configurazioni

– Configurazione a più

Come per il precedente DoF, anche per questo è sufficiente la variazione della velocità di soli due motori, quello a sinistra e quello a destra del corpo

del drone (rispettivamente i motori 4 e 2 in Figura 1.10), per ottenere una variazione dell'angolo di roll e quindi mettere in moto il drone

– Configurazione a croce

Analogamente, per la configurazione a croce sarà necessario intervenire sui 4 motori, aumentando i giri dei motori opposti alla direzione in cui si vuole far muovere il drone

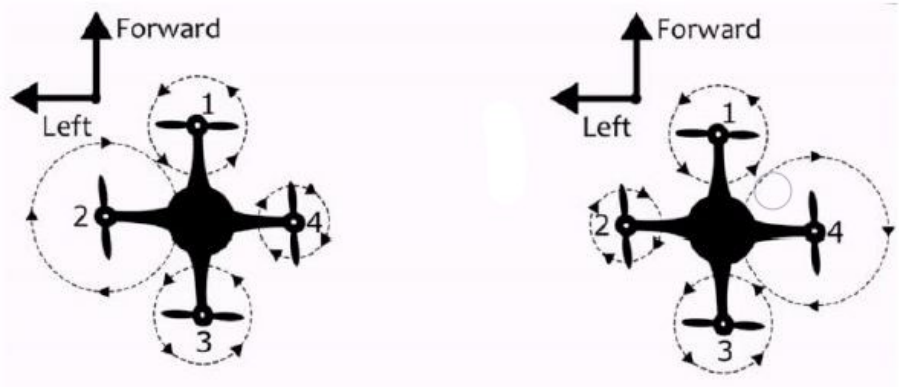


Figura 1.10: Roll configurazione a più (a destra e a sinistra)

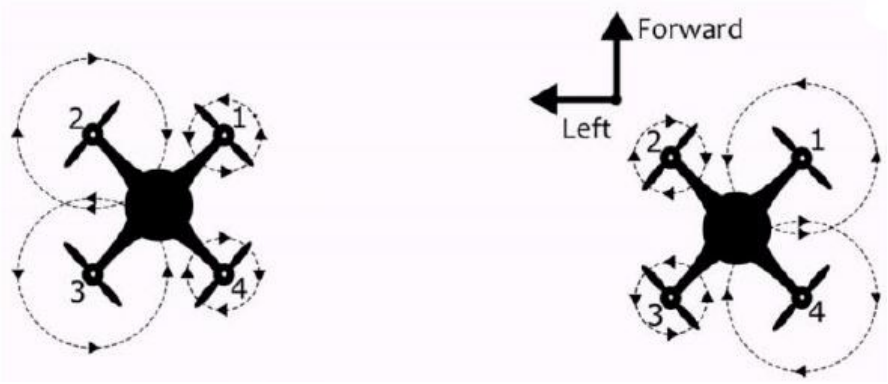


Figura 1.11: Roll configurazione a croce (a destra e a sinistra)

- Angolo di yaw

  Infine, per ottenere una variazione dell'angolo di yaw, e quindi una rotazione del drone intorno all'asse verticale, è necessario aumentare la velocità delle eliche che ruotano in un verso, e diminuire quella delle eliche che girano in senso opposto. Ad esempio per ottenere una rotazione in senso orario, è necessario aumentare la velocità dei motori che ruotano in senso antiorario e viceversa (motori 1 e 3 in Figura 1.12)

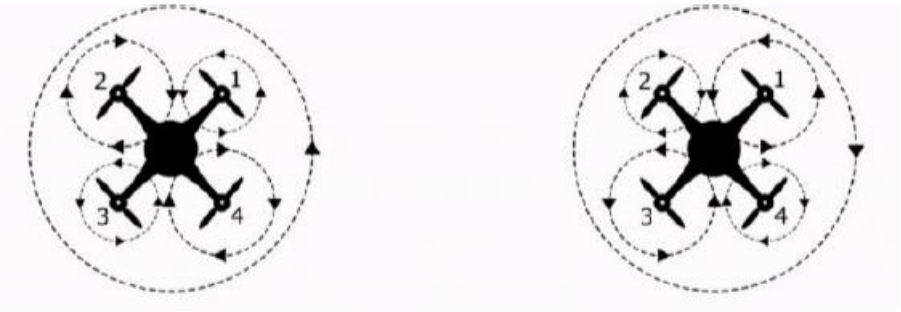


Figura 1.12: Yaw configurazione a croce

1.4 Parrot Mambo

Nella presente trattazione si è preso a modello il drone Parrot Mambo, particolarmente adatto alle nostre esigenze grazie alle dimensioni ridotte e al basso costo. Ma il principale vantaggio di questo quadricottero è la libera disponibilità di un pacchetto Simulink, il *Simulink Support Package for Parrot Minidrones* disponibile al seguente link: https://www.mathworks.com/hardware-support/parrot-minidrones.html, grazie al quale è possibile programmare il drone su un modello Simulink e testarlo in sicurezza in ambiente virtuale, per poi caricare il codice sorgente direttamente sul controllore a bordo del drone.

Tabella 1.2: Specifiche del Mambo

Specifica	Valore
Grandezza [cm]	18x18cm con paraurti
Peso	63g
Batteria	LiPo 660 mAh
Autonomia	circa 8min
Costo	circa 60€
Fotocamera	0.3 Megapixel
Velocità max	30 km/h

Il Mambo è inoltre dotato di numerosi sensori, in particolare:

- IMU (Inertial Measurement Unit) per misurare le velocità angolari e le accelerazioni lineari. É composto da un giroscopio a 3 assi e da un accelerometro a 3 assi

- Sensore di pressione per la misura dell'altitudine

- Sensore ad ultrasuoni per la stima dell'altitudine, calcola il tempo intercorso tra l'emissione del segnale e il ritorno dello stesso dopo essere stato riflesso al suolo

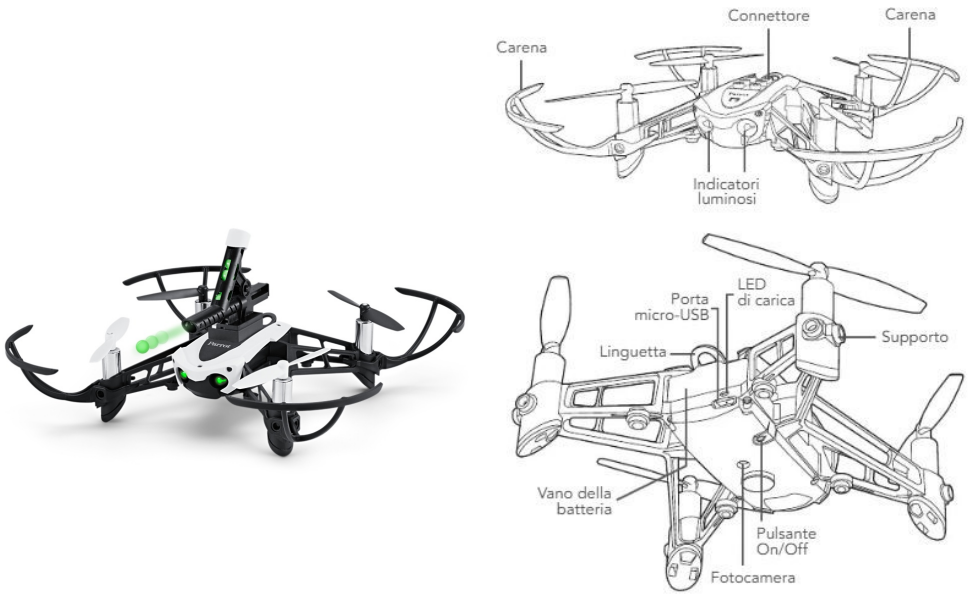
- Camera in grado di girare video a 120° con risoluzione 720p

Al fine di distinguere il naso del drone dalla coda, le due eliche anteriori sono di colore bianco mentre le eliche posteriori sono nere. Inoltre il rotore anteriore sinistro e il posteriore destro girano positivamente rispetto all'asse z, i restanti due girano negativamente rispetto all'asse z.

Infine, oltre alla disponibilità di numerosi accessori, come un piccolo cannone che spara palline da Softair, una videocamera aggiuntiva FPV (First Person View), paraurti per la sicurezza delle eliche e molti altri, è dotato di due indicatori luminosi.

Tabella 1.3: Indicatori luminosi Mambo

Indicatore	Significato
Arancione fisso	Drone in avviamento
Verde fisso	Drone pronto
Rosso lampeggiante	Batteria scarica
Rosso fisso	Problema rilevato



(a) *Parrot Mambo* (b) *Schema componenti Mambo*

Dopo aver descritto le caratteristiche delle eliche e il moto del Mambo, potrebbe risultare interessante capire che tipo di motori movimentano tali eliche. Per questo motivo è stata realizzata una piccola digressione sui motori coreless.

1.4.1 Motori coreless

Il principio generale di un motore elettrico a corrente continua prevede il passaggio di corrente in un avvolgimento, che genera un campo magnetico attorno ad un elemento in grado di ruotare (rotore) il quale si mette in moto grazie all'interazione con il campo magnetico di un magnete permanente (statore). Ad esempio, nei motori di tipo tradizionale, si hanno dei lamierini in materiale ferromagnetico sagomati a forma di corona circolare con cave e denti, impilati uno sull'altro e isolati tra loro con del materiale isolante, su cui è avvolto del filo in rame.

Nei motori coreless è assente il nucleo in ferro nel rotore, l'avvolgimento ha un design "obliquo" o "a nido d'ape" per creare una trama fitta e autoportante che viene tenuta in posizione con della resina epossidica. É possibile costruirli sia con le spazzole (motori DC coreless brushed) sia privi di spazzole (motori DC coreless brushless)

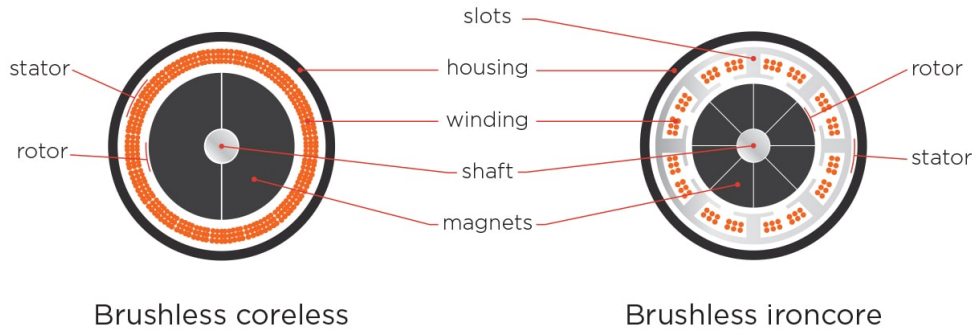


Figura 1.13: Motori coreless e ironcore a confronto

Vantaggi dei motori coreless:

- Azzeramento del fenomeno del cogging: si tratta dello "scatto" che si verifica nel passaggio da una posizione di equilibrio alla successiva (che si ha con l'allineamento del dente al magnete opposto). La sua assenza comporta maggior fluidità nel movimento

- Basso livello di interferenza elettromagnetica, che risulta utile essendo in presenza di altri segnali EM, come il radiocomando o lo smartphone

- Assenza di saturazione del ferro, che comporta elevata capacità di sovraccarico, e assenza di perdite nel ferro
- Lunga durata, bassa inerzia, coppia e velocità elevate
- Con la versione brushless si elimina lo scintillio tipico dello "strisciamento" delle spazzole, con questa configurazione è possibile utilizzare il drone in ambienti saturi di gas o di materiali potenzialmente esplosivi

Capitolo 2

Modello matematico del drone

Nel seguente capitolo verranno illustrare le equazioni matematiche che regolano il moto del drone. In questo modo risulterà più facile comprendere il modello Simulink che verrà utilizzato, oltre che avere una visione quanto più completa possibile dei fenomeni in gioco.

Per iniziare, essendo possibile approssimare il drone come un corpo rigido, se ne dà la definizione:

Definizione 1. Un corpo rigido è un oggetto materiale le cui parti sono soggette al vincolo di rigidità, ovvero sia quando è fermo che quando è in moto, non si deforma mai. In altre parole è un insieme di punti le cui distanze reciproche non variano nel tempo e nello spazio.

2.1 Angoli di Tait-Bryan

Per poter descrivere le equazioni che governano il moto di un velivolo, è necessario definire i sistemi di riferimento in cui si descrivono l'assetto e la posizione dello stesso. Nel caso del drone verranno prese in considerazione due terne:

- Terna fissa: detta anche terna mondo o terna inerziale, è indicata con O_{NED}, dove il pedice NED indica la direzione degli assi: North, East, Down
- Terna mobile: solidale con il corpo del drone, detto anche body frame, la terna mobile ha gli assi diretti verso il naso del drone (asse x), il fianco destro del drone (asse y) e verso il basso (asse z)

Le origini delle due terne coincidono tra loro e con il baricentro del drone.

Definite le terne di riferimento, è possibile definire gli assi di rotazione: in qualsiasi situazione, componendo 3 rotazioni elementari attorno a questi assi, è possibile

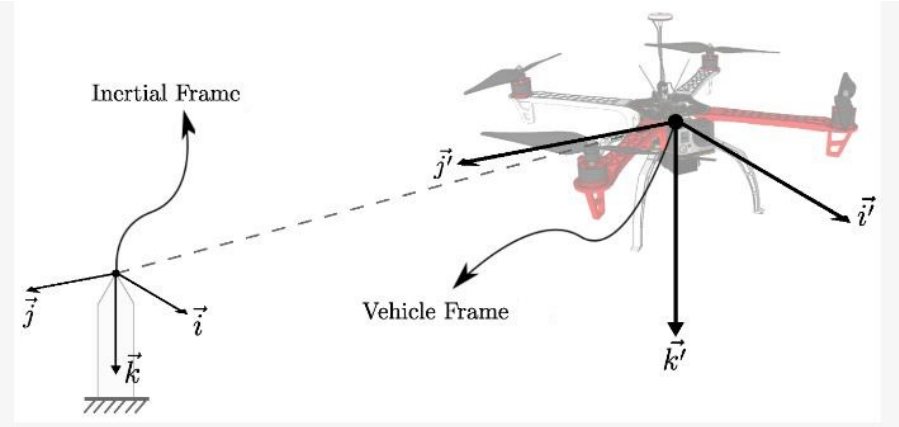


Figura 2.1: Terna inerziale e terna mobile

raggiungere qualsiasi terna di destinazione. Esistono 12 possibili sequenze di assi di rotazione:

- Angoli di Eulero propri: 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

In particolare, indicando con x,y,z gli assi della terna fissa, con X,Y,Z gli assi della terna mobile e con N l'asse dei nodi, definito come l'intersezione dei piani xy e XY (se coincidono allora N coincide con X), è possibile definire gli angoli di Tait-Bryan come segue:

- ϕ (α) : angolo di roll (o di precessione), è compreso tra asse x e asse N
- θ (β) : angolo di pitch (o di nutazione), è compreso tra asse z e asse Z
- ψ (γ) : angolo di yaw (o di rotazione propria), è compreso tra asse N e asse X

É quindi possibile definire le matrici di rotazione che permettono il passaggio da una terna di riferimento all'altra:

1. Rotazione dell'asse x di un angolo ϕ

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}$$

2. Rotazione dell'asse y di un angolo θ

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}$$

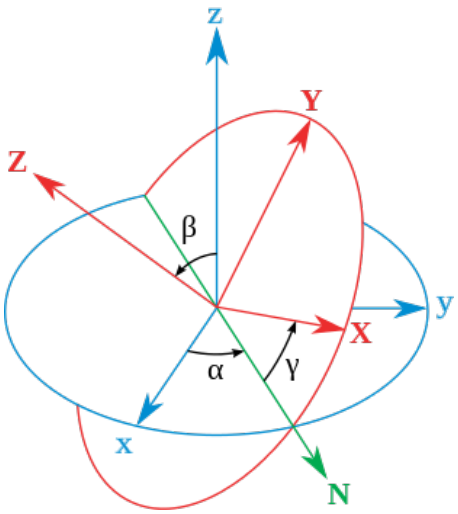


Figura 2.2: Angoli di Tait-Bryan

3. Rotazione dell'asse z di un angolo ψ

$$R_z(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Una rotazione attorno a tutti e 3 gli assi è data dal prodotto delle 3 matrici precedenti:

$R_{xyz}(\phi, \theta, \psi) = R_x(\phi)R_y(\theta)R_z(\psi) =$

$$= \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.1}$$

Tale matrice è spesso chiamata DCM (Direction Cosine Matrix)

2.2 Equazioni matematiche

Stabiliti i sistemi di riferimento, si può quindi procedere con la definizione delle variabili necessarie.

Nel sistema di riferimento inerziale consideriamo il vettore delle posizioni lineari, P_E, e il vettore degli angoli di assetto, A_E. Nella terna mobile introduciamo il vettore delle velocità lineari, V_B, e il vettore delle velocità angolari, ω_B. Per distinguerli si utilizzerà il pedice E, Earth, per la terna inerziale e il pedice B, Body frame, per la terna mobile. Si ottiene dunque

$$P_E = \begin{vmatrix} x & y & z \end{vmatrix}^T \tag{2.2}$$

$$A_E = \begin{vmatrix} \phi & \theta & \psi \end{vmatrix}^T \tag{2.3}$$

$$V_B = \begin{vmatrix} u & v & w \end{vmatrix}^T \tag{2.4}$$

$$\omega_B = \begin{vmatrix} p & q & r \end{vmatrix}^T \tag{2.5}$$

Analogamente alla matrice DCM, anche per le trasformazioni angolari è presente una matrice, detta ROT:

$$ROT = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{2.6}$$

É quindi possibile notare che

$$\dot{P_E} = DCM \cdot V_B$$
$$\dot{A_E} = ROT \cdot \omega_B$$

Da cui, scrivendo in forma estesa

$$\begin{cases} \dot{x} = u \cdot \cos\theta\cos\psi + v \cdot (\sin\theta\sin\phi\cos\psi - \cos\phi\sin\psi) + w \cdot (\sin\theta\cos\phi\cos\psi + \sin\phi\sin\psi) \\ \dot{y} = u \cdot \cos\theta\sin\psi + v \cdot (\sin\theta\sin\phi\sin\psi + \cos\phi\cos\psi) + w \cdot (\sin\theta\cos\phi\sin\psi - \sin\phi\cos\psi) \\ \dot{z} = -u \cdot \sin\theta + v \cdot \cos\theta\sin\phi + w \cdot \cos\theta\cos\phi \end{cases} \tag{2.7}$$

$$\begin{cases} \dot{\phi} = p + q \cdot \tan\theta\sin\phi + r \cdot \tan\theta\cos\phi \\ \dot{\theta} = q \cdot \cos\phi - r \cdot \sin\phi \\ \dot{\psi} = q \cdot \sin\phi/\cos\theta + r \cdot \cos\phi/\cos\theta \end{cases} \tag{2.8}$$

Guardando l'equazione relativa alla dinamica dell'angolo di yaw è possibile notare come esso sia strettamente legato agli altri due angoli, questo implica che una variazione dell'angolo di roll o dell'angolo di pitch potrebbe generare anche una variazione dell'angolo di yaw.

Prima di iniziare con l'analisi delle equazioni della dinamica è necessario fare alcune ipotesi di lavoro:

- Il drone ha una struttura perfettamente simmetrica, per cui il tensore di inerzia è diagonale
- La spinta generata da un rotore è proporzionale al quadrato della velocità: $F_t = C\Omega^2$
- Il centro di gravità del drone coincide con l'origine dei due sistemi di riferimento
- Il modello è tempo-invariante

- I motori sono perfettamente uguali, per cui trattarne un solo alla volta non fa perdere di generalità

- Effetti aerodinamici minori, come blade-flapping ed effetto suolo, sono trascurati

Si è infine pronti ad iniziare con le equazioni, che in questa trattazione verranno maneggiate con il formalismo di Newton-Eulero.

Dall'equazione di Newton si ha

$$\begin{vmatrix} F_B \\ \tau_B \end{vmatrix} = \begin{vmatrix} mI_{3x3} & 0 \\ 0 & J_{CM} \end{vmatrix} \cdot \begin{vmatrix} \dot{V_B} \\ \dot{\omega_B} \end{vmatrix} + \begin{vmatrix} \omega_B \times (mV_B) \\ \omega_B \times (J_{CM}\omega_B) \end{vmatrix} \tag{2.9}$$

Dove I_{3x3} indica la matrice identità di ordine 3 e J_{CM} indica il tensore di inerzia, che per le ipotesi precedenti risulta essere:

$$J_{CM} = \begin{vmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{vmatrix}$$

Possiamo riscrivere l'equazione 2.9 in forma matriciale. A tal scopo definiamo il vettore delle velocità generalizzato nel B-Frame (ν), la matrice di rotazione generalizzata (J_Θ), il vettore generalizzato delle forze e dei momenti (Λ) e il vettore generalizzato della posizione e dell'assetto (ξ):

$$\begin{aligned} \nu &= \begin{vmatrix} V_B & \omega_B \end{vmatrix}^T = \begin{vmatrix} u & v & w & p & q & r \end{vmatrix}^T \\ \Lambda &= \begin{vmatrix} F_B & \tau_B \end{vmatrix}^T = \begin{vmatrix} F_x & F_y & F_z & \tau_x & \tau_y & \tau_z \end{vmatrix}^T \\ \xi &= \begin{vmatrix} P_E & A_E \end{vmatrix}^T = \begin{vmatrix} x & y & z & \phi & \theta & \psi \end{vmatrix}^T \\ J_\Theta &= \begin{vmatrix} DCM & 0_{3x3} \\ 0_{3x3} & ROT \end{vmatrix} \end{aligned}$$

Si ottiene quindi

$$M_B\dot{\nu} + C_B\nu = \Lambda \tag{2.10}$$

Dove M_B è detta "System inertia matrix" e C_B è la "Coriolis-centripetal matrix"

e il cui valore è

$$M_B = \begin{vmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & J_{CM} \end{vmatrix} = \begin{vmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & J_x & 0 & 0 \\ 0 & 0 & 0 & 0 & J_y & 0 \\ 0 & 0 & 0 & 0 & 0 & J_z \end{vmatrix}$$

$$C_B = \begin{vmatrix} 0_{3_3} - mS(V_B) \\ 0_{3x3} - S(J_{CM}\omega_b) \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & 0 & 0 & 0 & J_z r & -J_y q \\ 0 & 0 & 0 & -J_z r & 0 & J_x p \\ 0 & 0 & 0 & J_y q & -J_x p & 0 \end{vmatrix}$$

Dove l'operatore S è l'operatore antisimmetrico, ovvero dato un vettore $k = \begin{vmatrix} k_1 & k_2 & k_3 \end{vmatrix}^T$ si ha che

$$S(k) = -S^T(k) = \begin{vmatrix} 0 & -k_3 & k_1 \\ k_3 & 0 & -k_1 \\ -k_2 & k_1 & 0 \end{vmatrix}$$

Si noti che il risultato a cui siamo giunti è a carattere totalmente generale e riguarda quindi tutti i corpi rigidi soggetti alle ipotesi precedenti. Dato che stiamo trattando il caso specifico di un quadricottero, vediamo i contributi che compongono il vettore Λ.

1. Vettore gravitazionale: $G_B(\xi)$

   Il primo contributo trattato, probabilmente il primo che possa venire in mente, è quello dovuto all'accelerazione di gravità $g = 9.81[m/s^2]$. Come facilmente intuibile, non influenza il momento totale del corpo ma solo le forze a cui è soggetto e in particolare, riferendosi alla terna inerziale, all'asse z:

   $$G_B(\xi) = \begin{vmatrix} F_G^B \\ 0_{3x1} \end{vmatrix} = \begin{vmatrix} DCM^{-1} \cdot F_G^E \\ 0_{3x1} \end{vmatrix} = \begin{vmatrix} DCM^T \cdot \begin{vmatrix} 0 \\ 0 \\ -mg \end{vmatrix} \\ 0_{3x1} \end{vmatrix} = \begin{vmatrix} mg\sin\theta \\ -mg\cos\theta\sin\phi \\ -mg\cos\theta\sin\phi \\ 0 \\ 0 \\ 0 \end{vmatrix} \tag{2.11}$$

   Dove con F_G^E e F_G^B sono indicati rispettivamente il vettore peso nella terna inerziale e nella terna del Body Frame. Inoltre, dato che la DCM è una matrice orto-normale, si ha $DCM^{-1} = DCM^T$

2. Effetto giroscopico

L'effetto giroscopico è un fenomeno fisico dovuto alla conservazione del momento angolare e generato quindi dal movimento delle eliche. Dato che due dei 4 motori ruotano in senso orario e gli altri due in senso antiorario, si ha uno squilibrio complessivo se la somma algebrica delle velocità dei motori non è nulla. Indichiamo quindi con Ω_{tot} la somma delle velocità che, ricordando quanto descritto nella sezione 1.4 a proposito dei versi di rotazione delle eliche, equivale a $\Omega_{tot} = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$, e con $\vec{\Omega}$ il vettore $\vec{\Omega} = \begin{vmatrix}\Omega_1 & \Omega_2 & \Omega_3 & \Omega_4\end{vmatrix}^T$. Oltre allo squilibrio delle velocità, si può ottenere momento giroscopico anche a causa di velocità angolari non nulle per le altre due variabili di assetto, si ha infatti:

$$\begin{aligned} O_B(\nu)\cdot\vec{\Omega} &= \begin{vmatrix} 0_{3x1} \\ -\sum_{k=1}^{4} J_{TP}\left(\omega_B \times \begin{vmatrix}0\\0\\1\end{vmatrix}\right)(-1)^k\Omega_k \end{vmatrix} = \begin{vmatrix} 0_{3x1} \\ J_{TP}\begin{vmatrix}-q\\p\\0\end{vmatrix}\Omega_{tot} \end{vmatrix} \\ &= J_{TP}\begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ q & -q & q & -q \\ -p & p & -p & p \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix}\vec{\Omega} \end{aligned} \tag{2.12}$$

Dove J_{TP} indica il momento di inerzia totale intorno all'asse del motore.

3. Forze e momenti prodotti dal drone

   L'ultimo termine analizzato riguarda gli effetti degli input riguardanti i movimenti principali.

   Definendo la distanza tra il centro di un rotore e il centro di gravità del quadricottero, l, e i coefficienti di spinta del motore e di drag, b e d, si ottiene:

$$U_B(\vec{\Omega}) = \begin{vmatrix} 0 \\ 0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ b(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2) \\ bl(\Omega_1^2+\Omega_2^2-\Omega_3^2-\Omega_4^2) \\ bl(-\Omega_1^2+\Omega_2^2+\Omega_3^2-\Omega_4^2) \\ d(\Omega_1^2-\Omega_2^2+\Omega_3^2-\Omega_4^2) \end{vmatrix} \tag{2.13}$$

   Dove per la componente U_4 si sono trascurati i termini dati da $\dot{\vec{\Omega}}$.

L'equazione 2.10 diventa quindi:

$$M_B\dot{\nu} + C_B(\nu)\nu = G_B(\xi) + O_B(\nu)\vec{\Omega} + E_B\vec{\Omega}^2 \tag{2.14}$$

da cui, ponendola in funzione di $\dot{\nu}$:

$$\dot{\nu} = M_B^{-1}\left(-C_B(\nu)\nu + G_B(\xi) + O_B(\nu)\vec{\Omega} + E_B\vec{\Omega}^2\right) \tag{2.15}$$

É quindi possibile esplicitare le componenti in forma di sistema:

$$\begin{cases} \dot{u} = (vr - wq) + g\sin\theta \\ \dot{v} = (wp - ur) - g\cos\theta\sin\phi \\ \dot{w} = (uq - vp) - g\cos\theta\sin\phi + \dfrac{U_1}{m} \\ \dot{p} = \dfrac{J_y - J_z}{J_x}qr - \dfrac{J_{TP}}{J_x}q\Omega_{tot} + \dfrac{U_2}{J_x} \\ \dot{q} = \dfrac{J_z - J_x}{J_y}pr + \dfrac{J_{TP}}{J_y}p\Omega_{tot} + \dfrac{U_3}{J_y} \\ \dot{r} = \dfrac{J_x - J_y}{J_z}pq + \dfrac{U_4}{J_z} \end{cases} \tag{2.16}$$

il quale è completamente espresso nel sistema di riferimento del Body Frame.

A volte tuttavia, potrebbe risultare utile esprimere le equazioni lineari nella terna inerziale e le equazioni angolari nella terna del Body Frame, ottenendo così un sistema di riferimento ibrido che chiameremo H.

Esprimendo il nuovo vettore generalizzato delle velocità come:

$$\zeta = \begin{vmatrix} \dot{P}_E & \omega_B \end{vmatrix} = \begin{vmatrix} \dot{x} & \dot{y} & \dot{z} & p & q & r \end{vmatrix}$$

L'equazione 2.14 diventa:

$$M_H\dot{\zeta} + C_H(\zeta)\zeta = G_H + O_H(\zeta)\vec{\Omega} + E_H(\xi)\vec{\Omega}^2 \tag{2.17}$$

Ridefiniamo quindi le matrici:

$$M_H = M_B$$

$$C_H(\zeta) = \begin{vmatrix} 0_{3x3} & 0_{3x3} \\ 0_{3x3} & -S(J_{CM}\omega_B) \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & J_z r & -J_y q \\ 0 & 0 & 0 & -J_z r & 0 & J_x p \\ 0 & 0 & 0 & J_y q & -J_x p & 0 \end{vmatrix}$$

$$G_H = \begin{vmatrix} F_G^E \\ 0_{3x1} \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ -mg \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

$$O_H(\zeta)\vec{\Omega} = O_B(\nu)\vec{\Omega}$$

$$E_H(\xi)\vec{\Omega}^2 = \begin{vmatrix} DCM & 0_{3x3} \\ 0_{3x3} & I_{3x3} \end{vmatrix} E_B\vec{\Omega}^2 = \begin{vmatrix} (\sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi)U_1 \\ (-\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi)U_1 \\ (\cos\theta\cos\phi)U_1 \\ U_2 \\ U_3 \\ U_4 \end{vmatrix}$$

Quindi, ponendo l'equazione 2.17 in funzione di $\dot{\zeta}$

$$\dot{\zeta} = M_H^{-1}\left(-C_H(\zeta)\zeta + G_H + O_H(\zeta)\vec{\Omega} + E_H(\xi)\vec{\Omega}^2\right) \tag{2.18}$$

ed espandendo in forma di sistema, si ottiene il sistema di equazioni nella terna H, ovvero:

$$\begin{cases} \ddot{x} = \left(\sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi\right)\dfrac{U_1}{m} \\ \ddot{y} = \left(-\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi\right)\dfrac{U_1}{m} \\ \ddot{z} = -g + \left(\cos\theta\cos\phi\right)\dfrac{U_1}{m} \\ \dot{p} = \dfrac{J_y - J_z}{J_x}qr - \dfrac{J_{TP}}{J_x}q\Omega_{tot} + \dfrac{U_2}{J_x} \\ \dot{q} = \dfrac{J_z - J_x}{J_y}pr + \dfrac{J_{TP}}{J_y}p\Omega_{tot} + \dfrac{U_3}{J_y} \\ \dot{r} = \dfrac{J_x - J_y}{J_z}pq + \dfrac{U_4}{J_z} \end{cases} \tag{2.19}$$

che, insieme a 2.7 e 2.8, esprime tutte le equazioni necessarie.

Capitolo 3

Modello Simulink

Nel seguente capitolo verrà mostrato e spiegato il modello Simulink utilizzato per la progettazione dell'algoritmo di inseguimento della traiettoria e di un controllore sostitutivo del PID standard a bordo del quadricottero.

3.1 Installazione e avviamento del progetto

La prima operazione da compiere per poter lavorare sul progetto, è installare il pacchetto di supporto messo a disposizione da MathWorks. Tale pacchetto è chiamato *Simulink Support Package for Parrot Minidrones*, disponibile al seguente link.

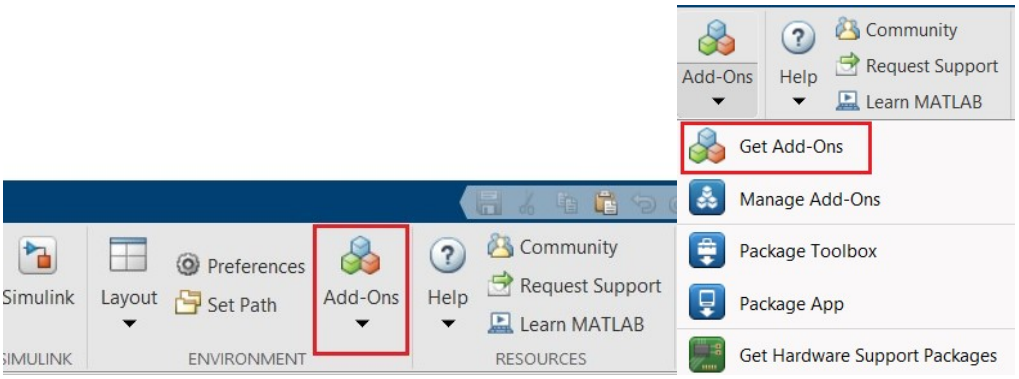
La procedura di installazione più veloce è la seguente:

- Aprire MATLAB
- Fare click su "Add-Ons"
- Fare click su "Get Add-Ons"
- Nella barra di ricerca digitare il nome del pacchetto desiderato, in questo caso *Simulink Support Package for Parrot Minidrones*
- Fare click sul pacchetto e infine su "Installa"

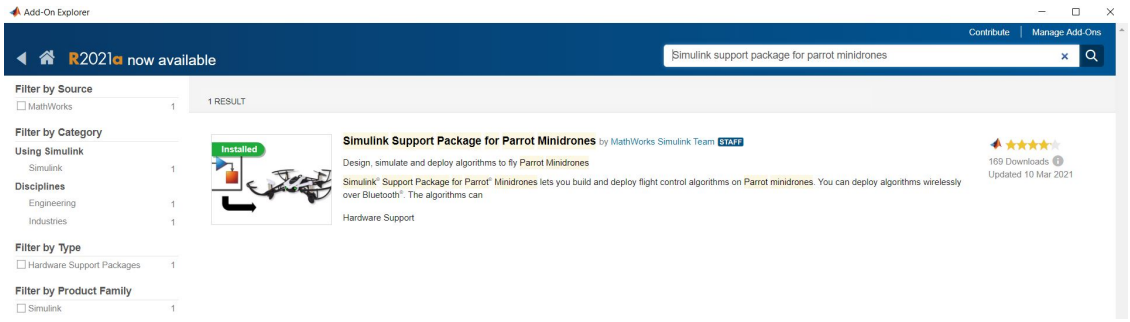
Il procedimento è riassunto in figura 3.1

I pacchetti raccomandati per la corretta gestione e visualizzazione del progetto sono numerosi, in generale però, nel caso di un pacchetto mancante, sarà il progetto stesso ad informare l'utente con un messaggio d'errore e a circondare con un bordo rosso il sotto-sistema che lo ha generato, esplicitando inoltre il nome del pacchetto mancante. I pacchetti fondamentali sono:

- *Simulink Support Package for Parrot Minidrones*
- Simulink 3D Animation



(a) *Sezione Add-Ons* (b) *Get Add-Ons*



(c) *Pacchetto di interesse*

Figura 3.1: Installazione nuovo pacchetto

- Aerospace Blockset
- Aerospace Toolbox
- Computer Vision Toolbox
- Simulink Control Design
- Signal Processing Toolbox
- Image Processing Toolbox
- Control System Toolbox
- Data Acquisition Toolbox
- DSP System Toolbox

tutti installabili come descritto in precedenza.

Una volta installati tutti i pacchetti, è possibile procedere con la creazione del progetto; per farlo è sufficiente scrivere nella Command Window di MATLAB il

comando *parrotMinidroneCompetitionStart*, si creerà così una cartella all'interno del workspace di MATLAB contenente i file necessari per avviare il progetto.

Aprendo il progetto su MATLAB compariranno varie schede: il Simulation Model, il Minidrone Flight Visualization e il gestore del progetto.

3.2 Flight Simulation Model

Passiamo quindi all'analisi del modello Simulink riguardante il Parrot Mambo.

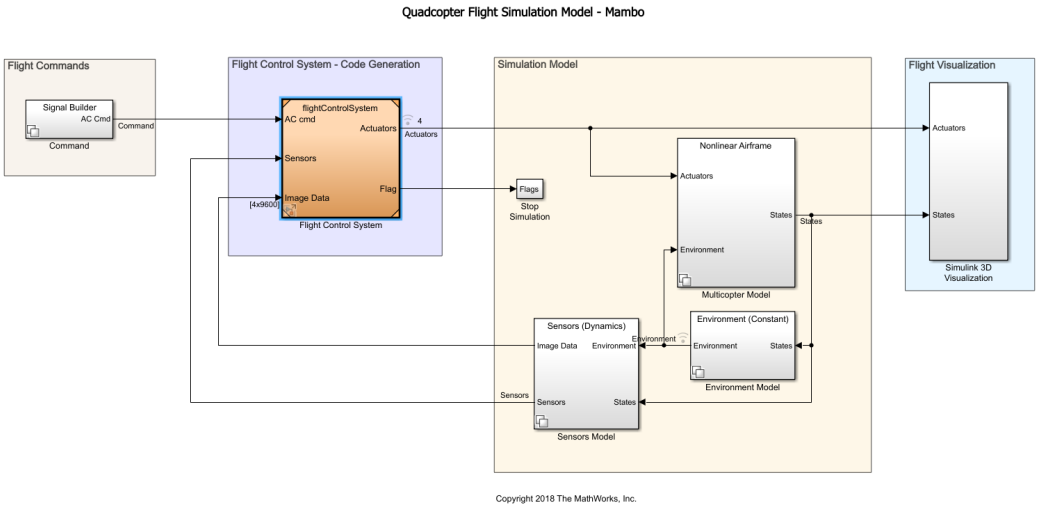


Figura 3.2: Flight Simulation Model

In figura 3.2 è possibile vedere la struttura generale del modello, in cui è evidente la divisione in sotto-sistemi. Da sinistra a destra si ha:

- Command
- Flight Control System
- Sensors Model
- Environment Model
- Multicopter Model
- Simulink 3D Visualization

É possibile notare la presenza di un ultimo blocco, Stop Simulation, che serve ad interrompere la simulazione nel caso in cui il drone dovesse rilevare malfunzionamenti o altri problemi.

3.2.1 Command

Il primo blocco in analisi è il blocco Command, che genera il segnale di riferimento per il drone.

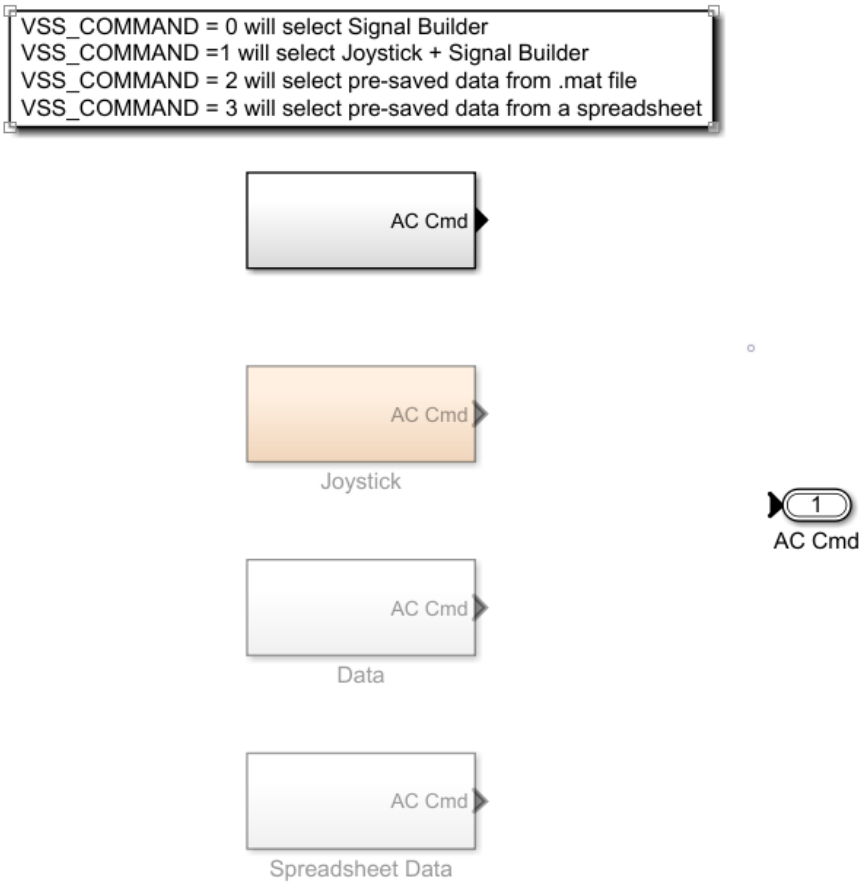


Figura 3.3: Blocco Command

Come è possibile notare in Figura 3.3, il blocco Command è composto da 4 sotto-sistemi, dei quali uno solo per volta può essere attivo e la cui selezione avviene attraverso la variabile *VSS_COMMAND* del workspace.

- Signal Builder: genera segnali costanti attraverso il blocco *Signal Builder*
- Joystick: permette il pilotaggio del drone tramite tastiera
- Data: il drone viene pilotato attraverso riferimenti passati con un file .mat
- Spreadsheet Data: il riferimento viene imposto da un foglio di calcolo

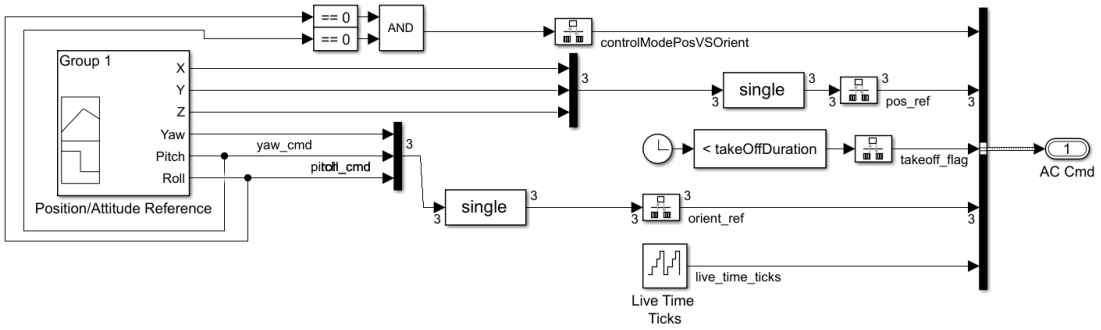


Figura 3.4: Sotto-sistema Signal Builder

3.2.2 Sensors Model

Il Sensors Model contiene il modello dei sensori a bordo del drone. Anche in questo caso è possibile selezionare due modalità attraverso la variabile *VSS_SENSORS*:

- Feedthrough: sono utilizzati modelli ideali di sensori
- Dynamics: sono utilizzati modelli dinamici di sensori, ovvero soggetti a rumori e disturbi

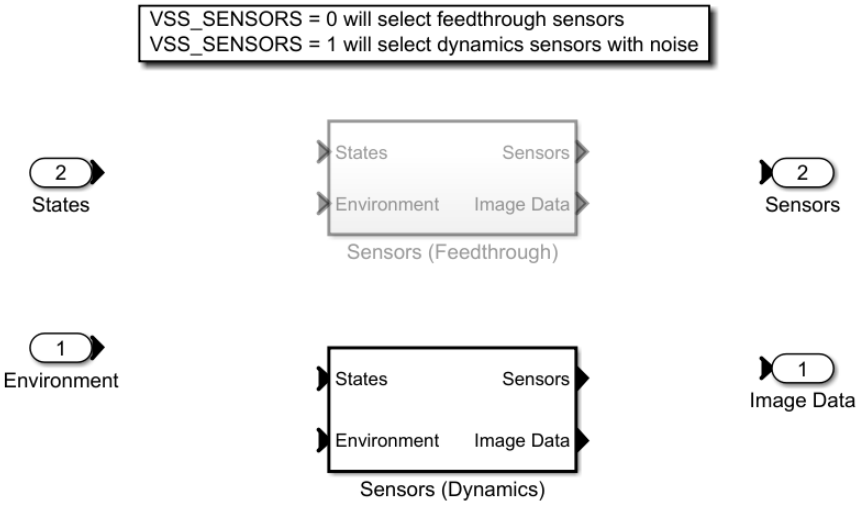


Figura 3.5: Blocco Sensors Model

All'interno del modello *Dynamics*, aprendo il blocco *Sensor System*, si trovano i due sottosistemi relativi ai sensori *Inertial Measurement Unit*, *Pressione* e alla *Camera*.

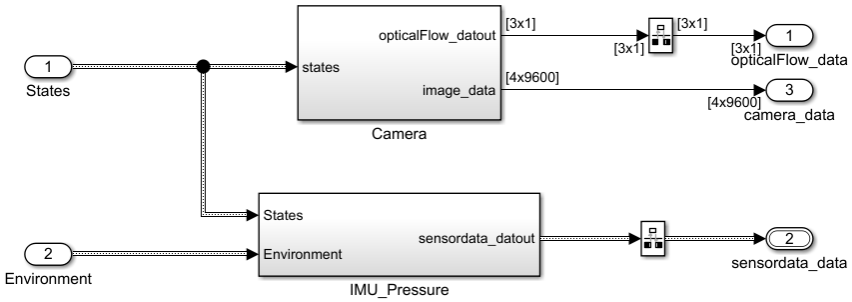


Figura 3.6: Sensori camera, IMU e pressione

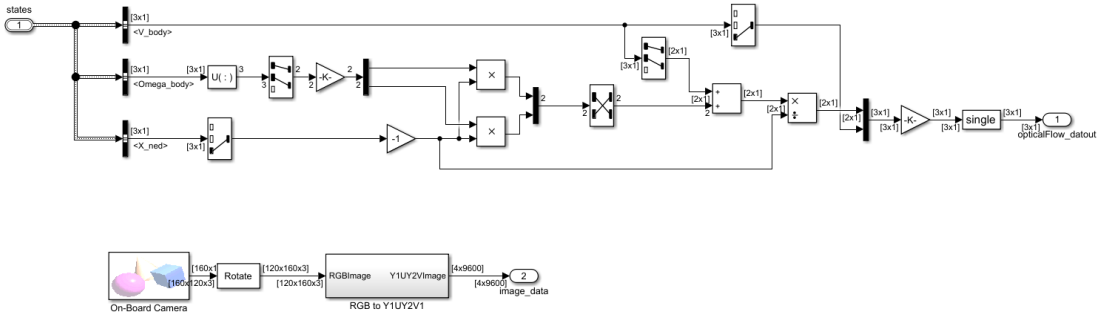


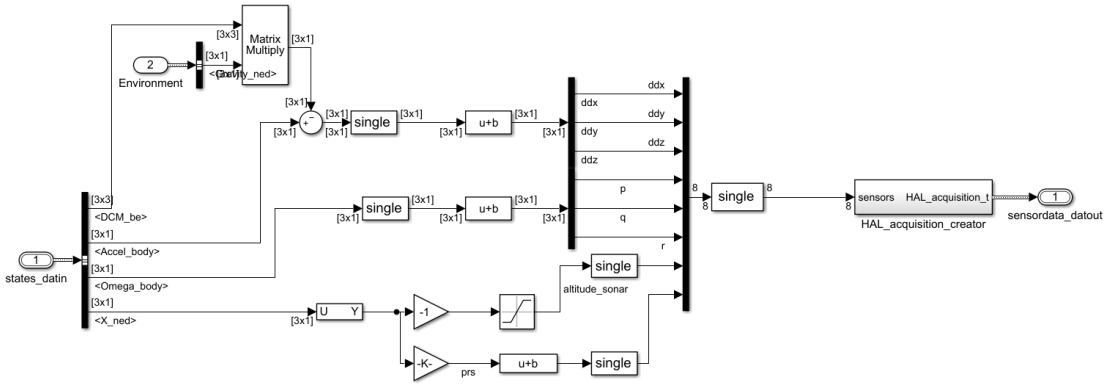
Figura 3.7: Sensore camera

3.2.3 Environment Model

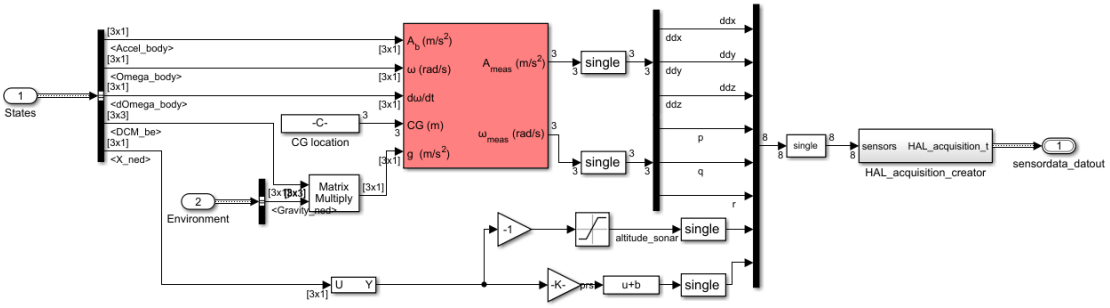
L'Environment Model contiene i dati riguardanti l'ambiente in cui è immerso il drone. Attraverso il valore della variabile *VSS_ENVIRONMENT* è possibile scegliere tra due modalità:

- Constant: l'ambiente è definito come costante
- Variable: l'ambiente è più dinamico, ad esempio pressione e temperatura dell'aria variano con l'altezza e così via.

Prendendo in esame il primo sotto-sistema, è possibile vedere tutte le costanti che definiscono lo stato dell'ambiente, come vettore accelerazione di gravità, temperatura dell'aria, velocità del suono e così via (Figura 3.10).



(a) *Sensore IMU Feedthrough*



(b) *Sensore IMU Dynamics*

Figura 3.8: Sensori IMU nelle due modalità

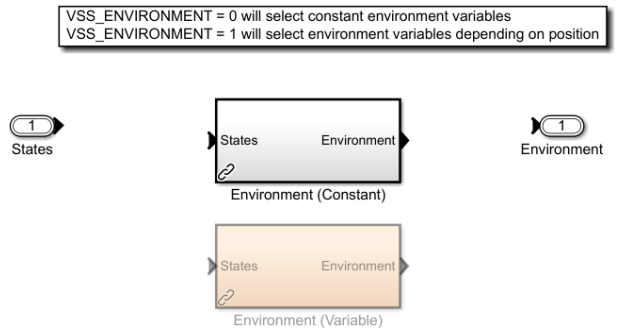
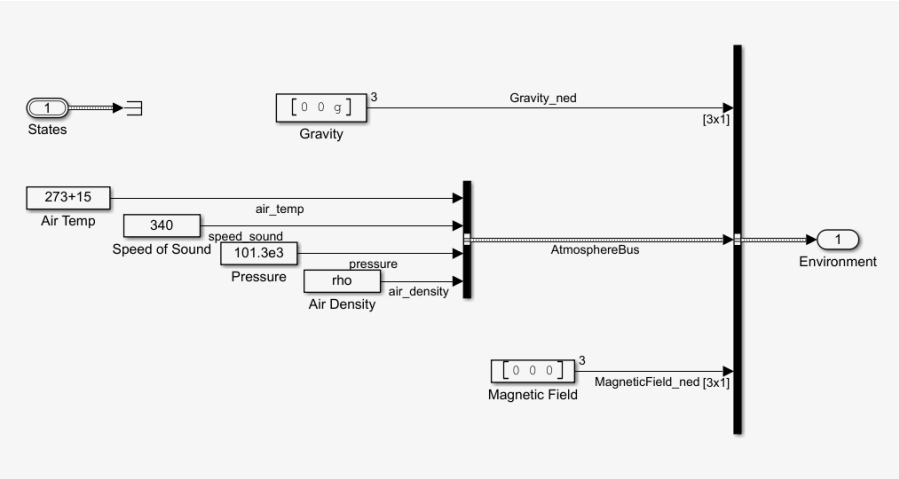
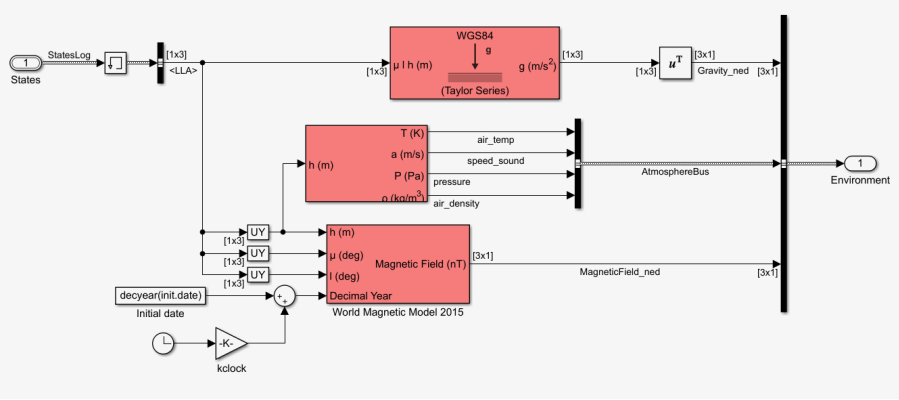


Figura 3.9: Environment Model



(a) *Environment(Constant)*



(b) *Environment(Variable)*

Figura 3.10: Environment nelle due modalità

3.2.4 Multicopter Model

All'interno del sotto-sistema *Multicopter Model* sono contenuti tutti i blocchi necessari al funzionamento del modello del drone. Vi sono presenti il calcolo delle forze e dei momenti, l'implementazione delle equazioni viste nel capitolo 2 e la gestione dei controllori di volo.

Prima di tutto è possibile scegliere che tipo di modello utilizzare tramite la variabile *VSS_VEHICLE*:

- Modello non lineare: implementa le equazioni come viste in questa trattazione, utilizzando quindi formule non lineari

- Modello lineare: implementa equazioni linearizzate sotto forma di sistema in spazio di stato. Una volta selezionato tale modello e avviata la simulazione,

verrà definita nel workspace la struttura *linsys* contenente i vettori e le matrici dello spazio di stato. Questa modalità è utilizzabile ad esempio ai fini della progettazione di controllori con tecniche di controllo lineari.

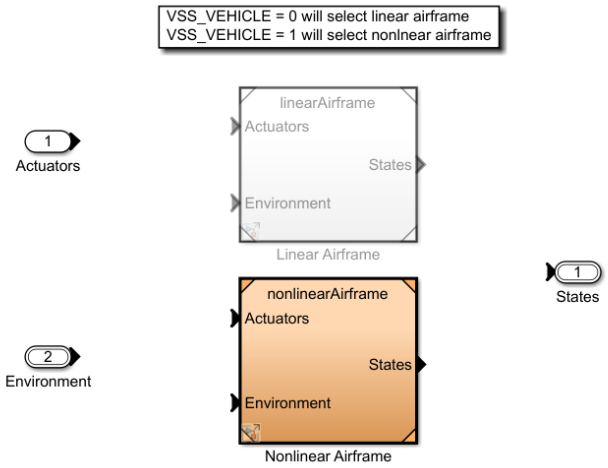


Figura 3.11: Scelta modello quadricottero

Modello non lineare

Il modello non lineare è composto da un blocco Simulink *6DOF* che prende in ingresso le forze e i momenti applicati al corpo, e restituisce i valori aggiornati di

- Posizione nella terna inerziale, $X\_ned(P_E)$
- Velocità lineare nella terna inerziale, $V\_ned(V_E)$
- Angoli di assetto, $Euler(A_E)$
- Velocità lineare nella terna Body frame, V_b
- Velocità angolare nella terna Body frame, $Omega\_b(\omega_B)$
- Accelerazione angolare nella terna B, $dOmega_b(\frac{d}{dt}\omega_B)$
- Accelerazione lineare nella terna B, $Accel\_b(\dot{V}_B)$
- La matrice *DCM*
- La posizione nel sistema *Latitudine-Longitudine-Altitudine*

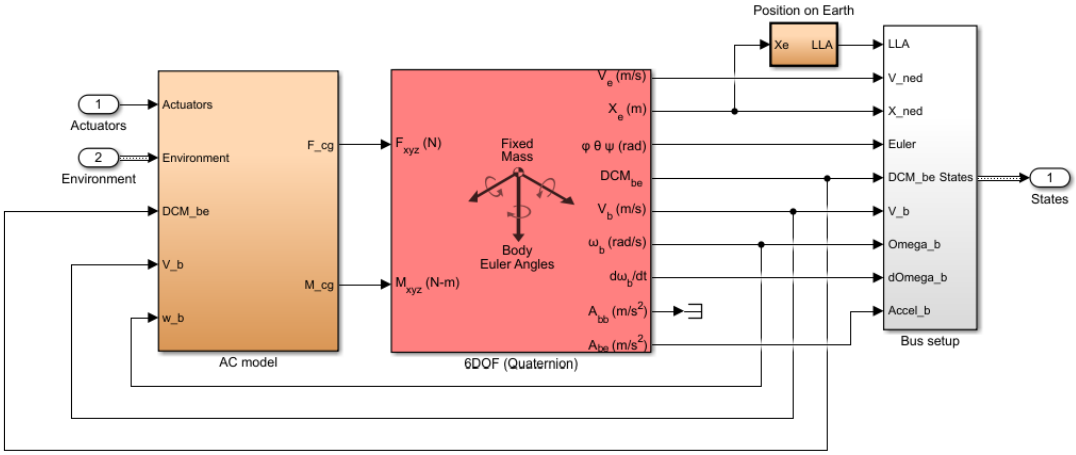


Figura 3.12: Modello non lineare

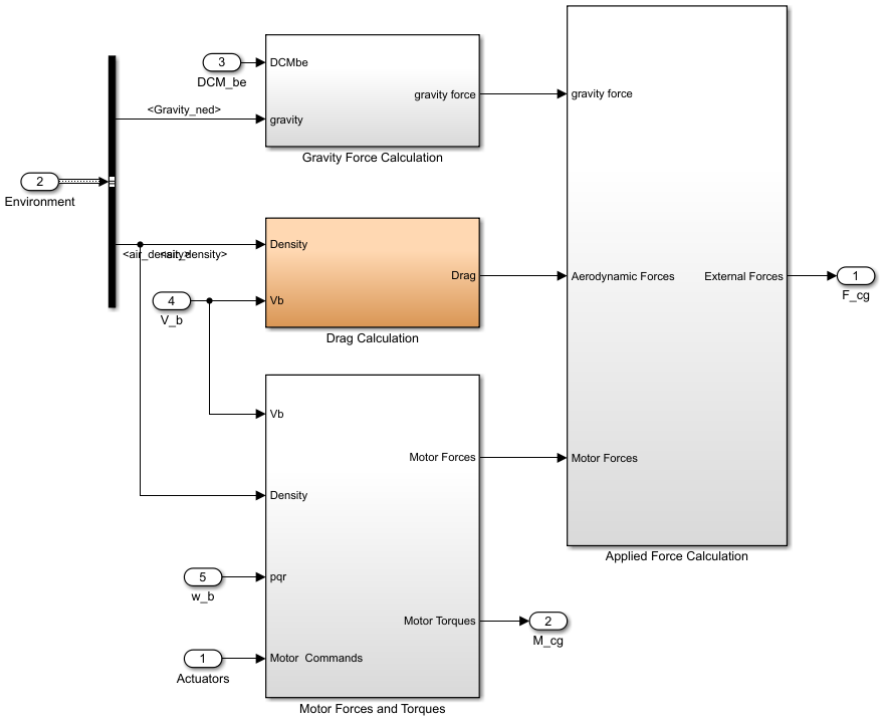
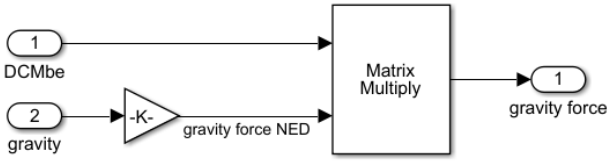


Figura 3.13: Blocco AC Model

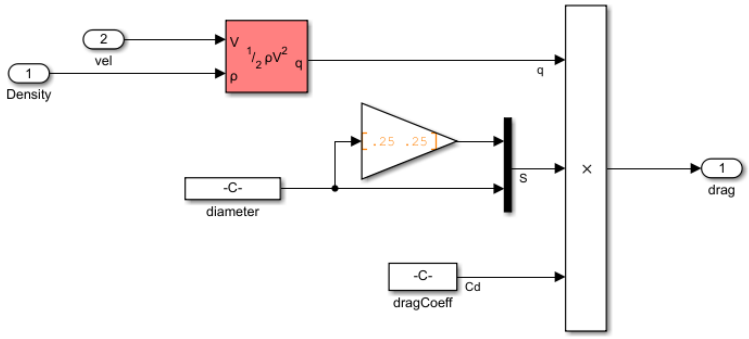
e da un blocco *AC model* che calcola forze e momenti angolari.

AC Model è composto da 3 sotto-sistemi, i quali calcolano il momento totale applicato al corpo e i 3 contributi dati da gravità, attrito viscoso e forze generate dai motori, sommandoli infine in un blocco apposito.

Il primo blocco moltiplica semplicemente il vettore $\begin{bmatrix} 0 & 0 & g \end{bmatrix}$ per la matrice DCM per ottenere il vettore F_G^B. Mentre il secondo, per il calcolo dell'effetto drag, implementa la formula $q_i = C_q \rho A r^2 V_B^2 r$, dove C_q è il coefficiente di drag, A è l'area spazzata dalle eliche, ρ è la densità dell'aria ed r è il raggio del rotore.



(a) *Calcolo forza di gravità*



(b) *Calcolo effetto drag*

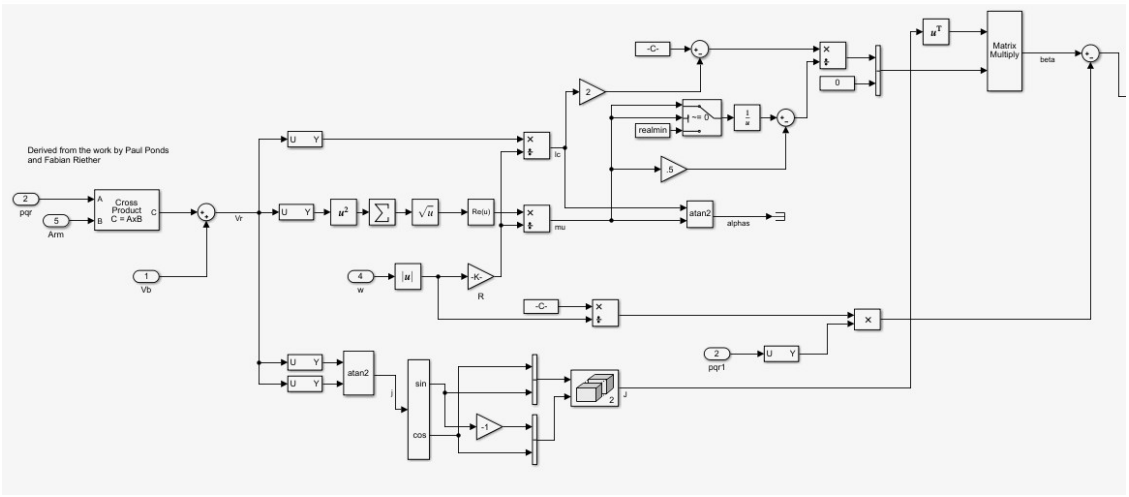
Figura 3.14: Forza di gravità ed effetto drag

Il terzo blocco è il più complesso, in quanto contiene numerosi blocchi Simulink per il calcolo delle forze e dei momenti descritti nel precedente capitolo. É strutturato come un sotto-blocco *For-Each* che svolge le medesime operazioni sui due ingressi, la prima volta per le forze e la seconda per i momenti (Figura 3.15).

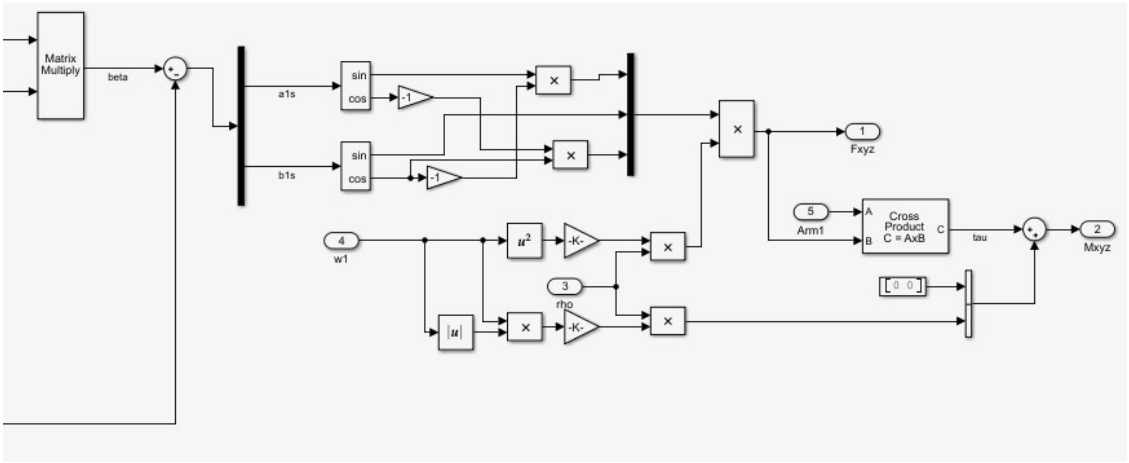
A questo punto il calcolo dei momenti viene direttamente dato in ingresso al blocco *6DOF*, mentre il calcolo delle forze di questo blocco, viene sommato alle componenti calcolate nei blocchi precedenti.

Modello lineare

Per la generazione del modello lineare viene utilizzata la funzione *linearize* il cui utilizzo può essere osservato nel file *trimLinearizeOpPoint.m* nella cartella *linearAirframe* del progetto.



(a) *Parte A*



(b) *Parte B*

Figura 3.15: Calcolo di forze e momenti generati dai rotori

Aprendo il blocco LinearAirframe è possibile analizzare il modello lineare del quadricottero posto in spazio di stato:

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

É possibile visualizzare le matrici A, B, C, D che definiscono il processo aprendo

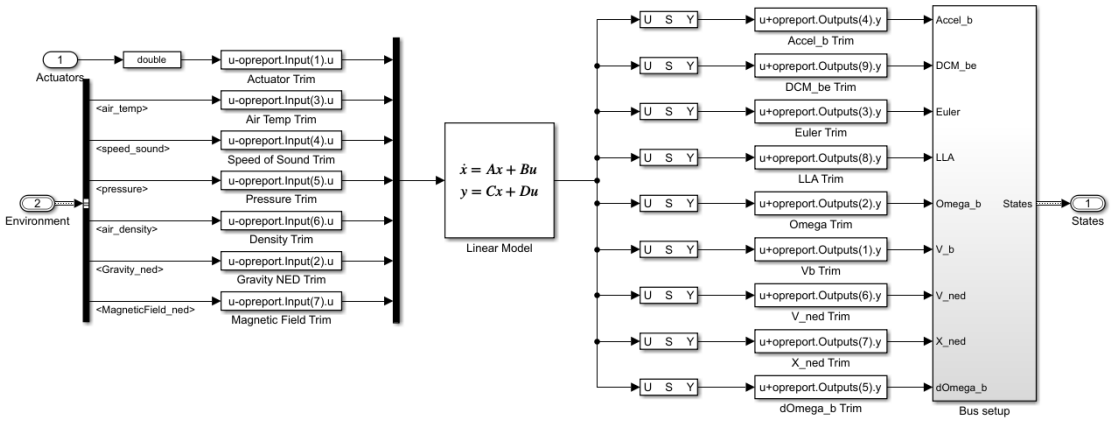


Figura 3.16: Modello lineare

la struttura *linsys* e scegliendo quale componente visualizzare. In particolare le matrici sono visualizzabili nell'appendice A

3.2.5 Simulink 3D Visualization

Il blocco di visualizzazione 3D definisce gli strumenti necessari alla simulazione tridimensionale del drone, in modo da poter visualizzare il comportamento che avrà il drone reale.

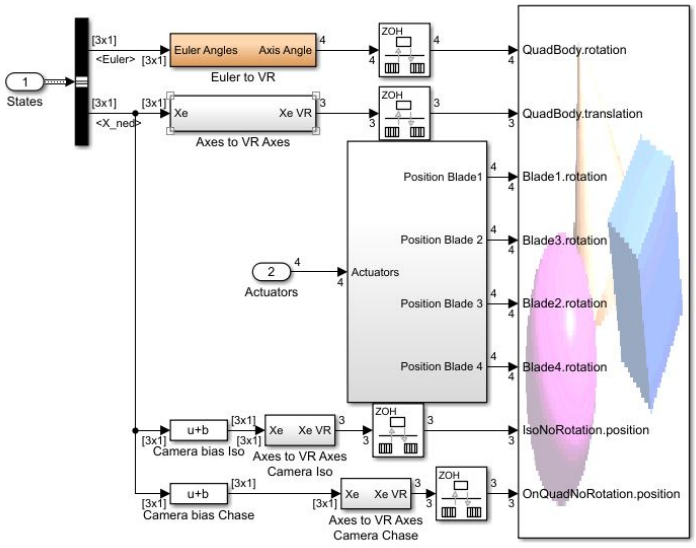


Figura 3.17: Blocco di visualizzazione

3.2.6 Flight Control System

Il Flight Control System è il blocco su cui il presente lavoro si è sviluppato, nonchè uno dei principali del modello stesso. Infatti questo sotto-sistema è diviso in due gruppi: *Image Processing System* (il blocco di logica di processamento delle immagini) e *Control System* (il blocco dei sistemi di controllo), contenente gli stimatori, i controllori, il blocco di logica di inseguimento del percorso e il blocco di previsione delle collisioni. L'*Image Processing System* e il *Path Planning* verranno analizzati in maniera più approfondita nel prossimo capitolo.

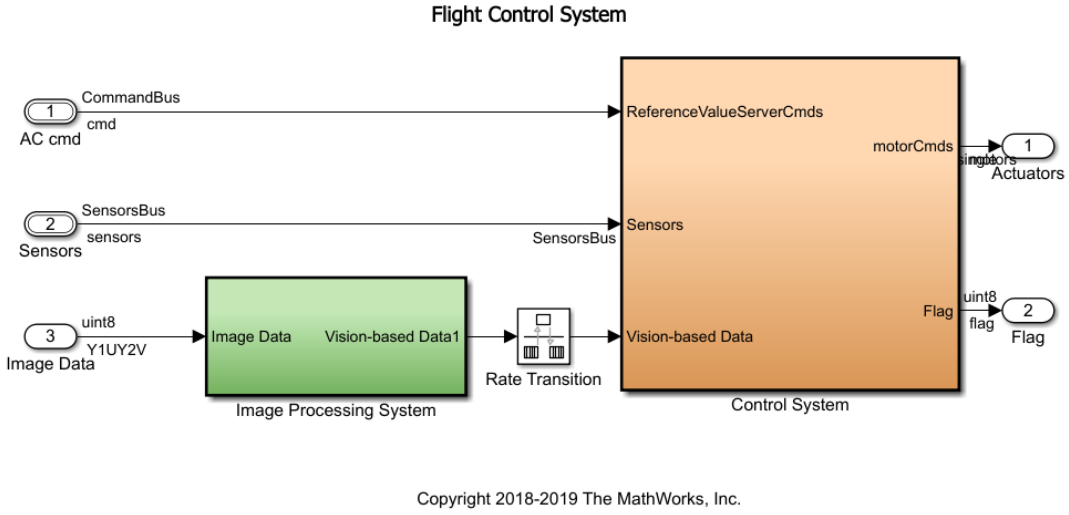


Figura 3.18: Processamento immagini e sistema di controllo

Aprendo il blocco *Control System* ci si trova davanti a 4 sotto-sistemi e ad una costante:

- controlModePosVSOrient regola la modalità di controllo del quadricottero, con riferimento lineare (*Position*) o con riferimento angolare (*Orientation*)
- State Estimator contiene gli stimatori delle variabili di stato del sistema
- Controller contiene i controllori dei gradi di libertà del quadricottero
- Crash Predictor Flags gestisce le informazioni provenienti dai sensori e dagli stati stimati al fine di anticipare eventuali collisioni e fermare la simulazione
- Path Planning contiene la logica che regola l'inseguimento di un percorso disegnato al suolo

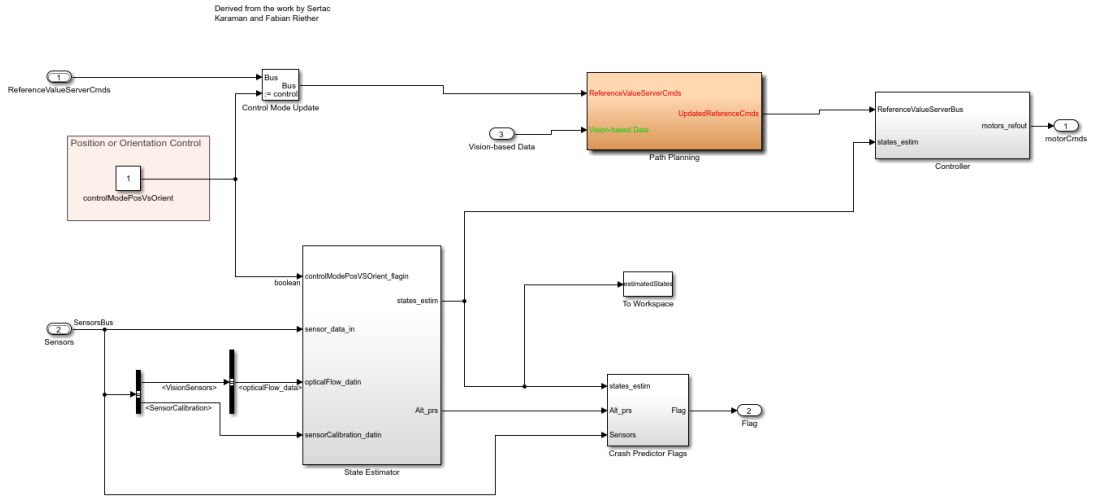


Figura 3.19: Control System

State Estimator

Riceve in ingresso i dati provenienti dai sensori a bordo del drone e, tramite degli stimatori, emette in uscita le stime di:

- Posizione e velocità lineare
- Assetto e velocità angolare

in particolare sono utilizzati dei filtri di Kalman per posizione e velocità lineare e un filtro complementare per l'assetto.

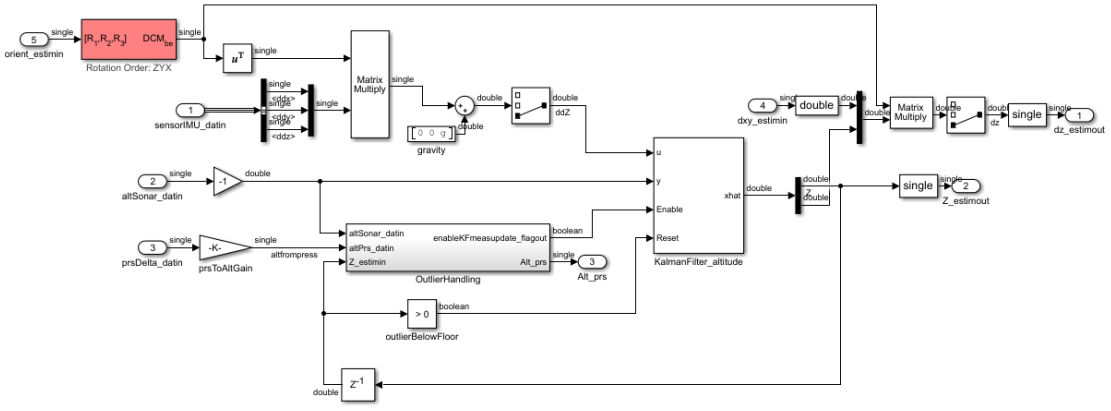


Figura 3.20: Filtro di Kalman per l'altitudine

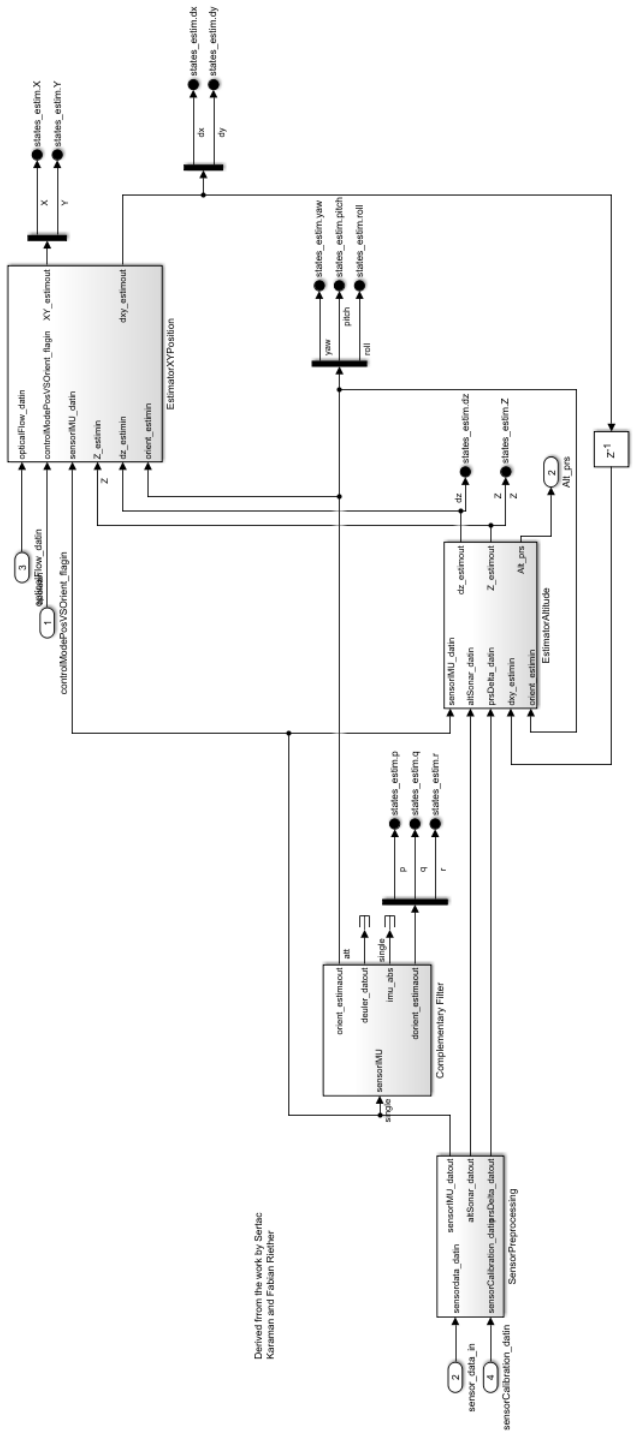


Figura 3.21: Stimatori degli stati

Controller

Infine il sotto-sistema *Controller* contiene i controllori per i 6 gradi di libertà del drone.

Si può notare in Figura 3.23 come pitch e roll siano accoppiati alla posizione x-y, infatti l'uscita del blocco *XY-to-reference-orientation* funge da ingresso nel blocco *Attitude*.

Come controllori standard, il drone Mambo monta:

- PID per il controllo degli angoli di pitch e di roll
- PD per il controllo dell'angolo di yaw
- PD per la posizione North-East-Down

Per una rapida descrizione del funzionamento dei PID si guardi il Capitolo 5.1.

Prendendo come esempio il PID relativo agli angoli di pitch e yaw, è possibile notare 3 componenti:

1. Proporzionale (quella in alto): moltiplica l'errore, dato dalla differenza tra riferimenti e angoli stimati, per un guadagno K_p
2. Integrale (quella in mezzo): moltiplica l'integrale nel tempo dell'errore per un guadagno K_I
3. Derivativa (quella in basso): moltiplica la derivata dell'errore per un guadagno K_d. In questo caso la derivata dell'assetto è semplicemente la velocità angolare

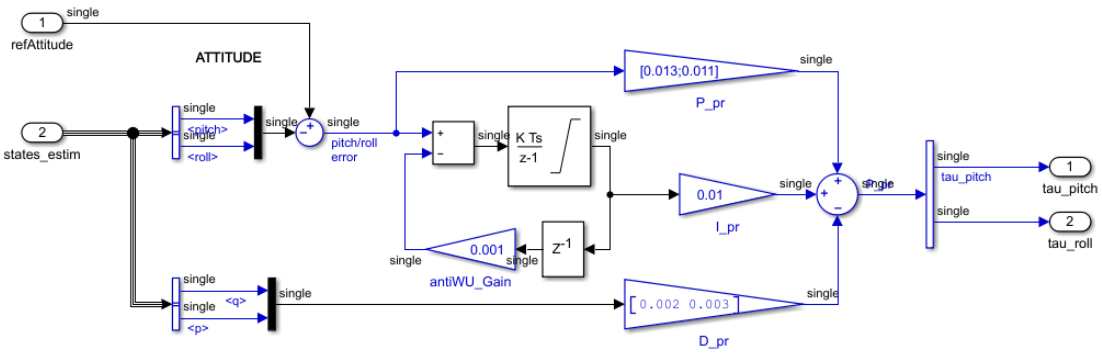


Figura 3.22: PID per gli angoli di pitch e roll

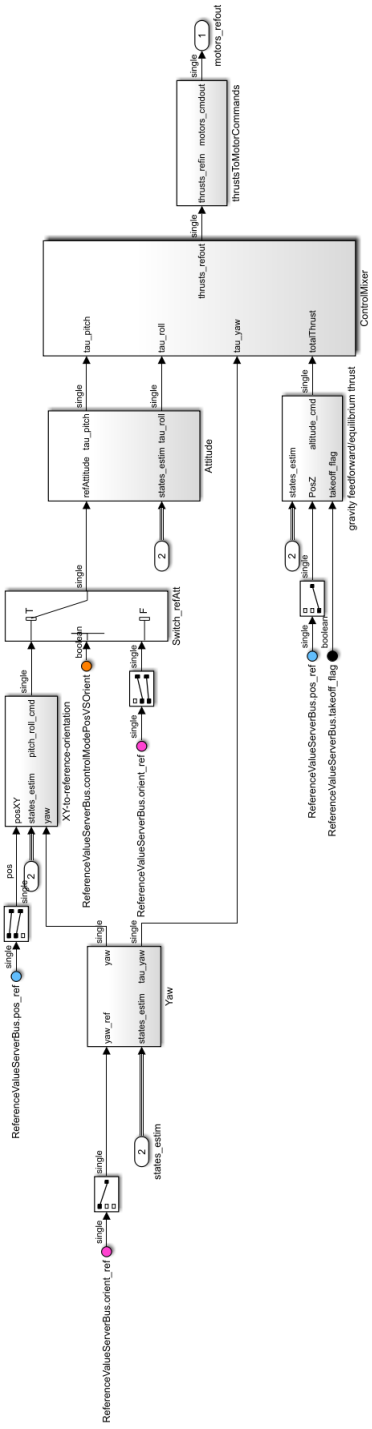


Figura 3.23: Controllori del quadricottero

Capitolo 4

Algoritmo di elaborazione delle immagini

Nella presente sezione si mostrerà e spiegherà l'algoritmo ideato per l'elaborazione delle immagini e l'inseguimento da parte del drone di un percorso lineare creato al suolo.

Come prima cosa è utile analizzare gli strumenti che verranno utilizzati all'interno dell'algoritmo, al fine di comprendere il modo in cui è stato realizzato e per implementare eventuali correzioni e miglioramenti.

4.1 HSV

L'HSV è una modalità di rappresentazione del colore alternativa al più comune RGB. HSV sta per *Hue-Saturation-Value*, ovvero *Tonalità-Saturazione-Valore* in quanto separa le informazioni relative all'intensità (luminanza) dalle informazioni relative al colore (cromaticità). Dato che una variazione nella saturazione di un pixel è visivamente più influente di una variazione nella tonalità, generalmente viene scelta l'intensità o la tonalità come caratteristica dominante basata sulla saturazione, per poi "segmentare" l'immagine raggruppando pixel con caratteristiche comuni.

Lo spazio colore HSV viene spesso rappresentato tridimensionalmente sotto forma di cilindro o cono, in cui l'intensità è rappresentata dall'asse verticale, la tonalità è definita come l'angolo compreso tra $[0, 2\pi]$ in cui il rosso è posizionato in 0 e in 2π, il verde in $2\pi/3$ e il blu in $4\pi/3$, infine la saturazione, che indica la "purezza" del colore, è misurata dalla distanza radiale dall'asse verticale, con valori compresi tra 0, sull'asse, e 1, sul bordo esterno.

Si noti che per S=0, al variare della luminosità, il colore varia dal bianco al nero passando per numerose sfumature di grigio.

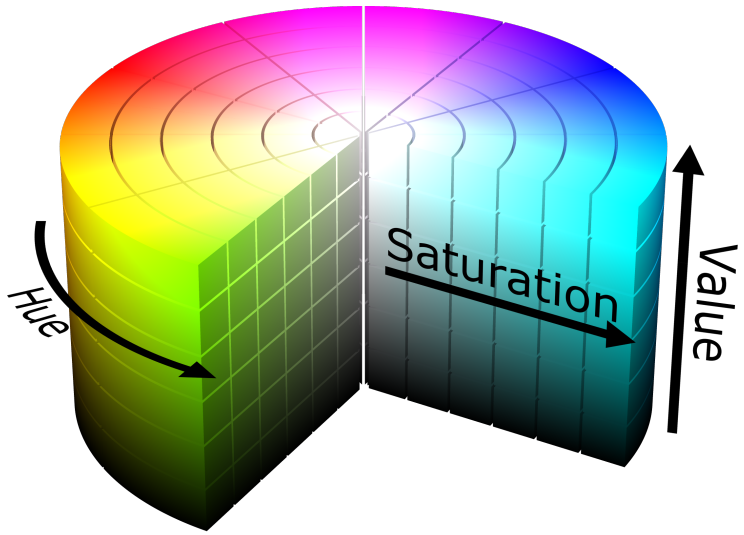


Figura 4.1: Spazio colore HSV

Per gli scopi indicati in questa trattazione, i vantaggi nell'utilizzo dello spazio colore HSV sono principalmente 2:

- La rappresentazione in formato HSV è più vicina alla prospettiva umana, in quanto basata sulla percezione del colore in termini di tinta, sfumatura e luminosità
- Il modello HSV è ideale per il rilevamento del colore, in quanto è possibile stabilire un range di tonalità che quindi risulterà robusto rispetto a variazioni nella luminosità (ad esempio zone più in ombra, o luci non omogenee) e nella saturazione (ad esempio tessuti scoloriti dalle intemperie)

4.2 Rilevamento dei bordi

Per il corretto funzionamento dell'algoritmo di rilevamento del percorso, è stato necessario identificare i bordi del tragitto in modo da poter calcolare gli angoli con cui affrontare le curve.

Algoritmo di Canny

Un bordo è una linea in cui si ha rapida variazione nell'intensità dell'immagine. Il rilevamento dei bordi è stato effettuato utilizzando l'algoritmo di Canny, di cui si darà una rapida spiegazione senza entrare nei formalismi matematici. Tale algoritmo si compone principalmente di 4 passi:

1. Rimozione di eventuale rumore dall'immagine grezza. Per farlo viene utilizzato un filtro gaussiano che si convolve all'immagine originale ottenendo una leggera "sfocatura" gaussiana

2. Calcolo del gradiente di luminosità dell'immagine al fine di trovare la forza dei bordi (*Edge strength*). A tal proposito possono essere utilizzati dai 2 ai 4 filtri da convolvere con l'immagine per identificare i bordi verticali, orizzontali e diagonali

3. Soppressione dei non massimi: si mantengono solo i massimi locali, ovvero i punti per i quali il valore nella mappa dei gradienti è maggiore rispetto a quello dei punti adiacenti nella direzione del gradiente. Inoltre per i massimi locali la derivata del gradiente si annulla

4. Sogliatura con isteresi: alla fine del passo precedente si ottiene una immagine in scala di grigi contenente possibili punti di bordo, bisogna effettuare una sogliatura per verificare quali lo siano effettivamente. Tuttavia se si confrontasse il valore del punto di bordo con una sola soglia T, si rischierebbe di ottenere bordi interrotti nei punti in cui, a causa di disturbi, il valore scende al di sotto di T. Per questo motivo vengono utilizzate due soglie T_1 e T_2:

   - Se il pixel ha un valore superiore alla soglia T_1 allora viene accettato come pixel di bordo e posto ad 1
   - Se il pixel ha un valore inferiore alla soglia T_2 allora viene scartato e posto a 0
   - Se il pixel ha un valore compreso tra T_2 e T_1 ed è connesso ad un punto già accettato come bordo, allora anch'esso viene accettato e posto ad 1, altrimenti viene scartato e posto a 0

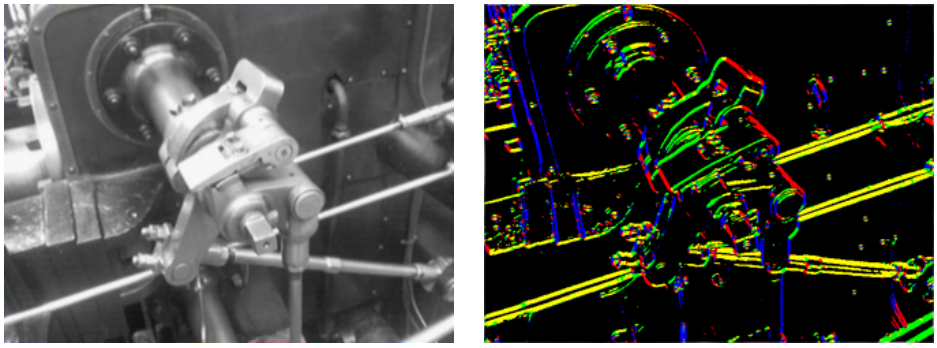
Si guardi a titolo esemplificativo le immagini in Figura 4.2, tratte da [1].

Trasformata di Hough

Il secondo strumento utilizzato è la trasformata di Hough, una tecnica di estrazione basata sul riconoscimento delle linee e sul principio secondo cui una qualsiasi forma può essere espressa attraverso una funzione nota con parametri.

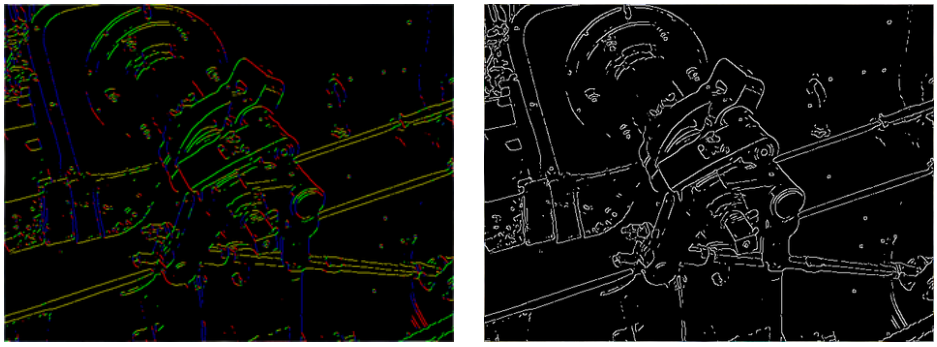
Si consideri ad esempio la rappresentazione classica della retta: $y = mx + q$, ogni retta è definita univocamente dai parametri m e q. Se consideriamo invece la rappresentazione in forma di Hesse della retta, ovvero $\rho = x\cos\theta + y\sin\theta$, l'insieme dei parametri è dato da (ρ, θ). É quindi possibile trasformare una retta nel piano $x - y$ in un punto dello spazio dei parametri $\rho - \theta$.

Una immagine è costituita da un insieme di punti e dato che un punto P non è altro che l'intersezione di più rette, nello spazio dei parametri si avrà una curva formata dai punti immagine delle rette passanti per P. Se nell'immagine in analisi



(a) *Immagine convoluta con filtro gaussiano*

(b) *Mappa binaria degli angoli con operatore Sobel*



(c) *Soppressione dei non-massimi*

(d) *Bordi dell'immagine*

Figura 4.2: Algoritmo di Canny su immagine di giunto a vapore

ci sono due punti A e B allineati su una retta, nello spazio dei parametri si avranno due curve che si intersecano in un punto, tale punto è l'immagine della retta su cui giacciono i punti A e B.

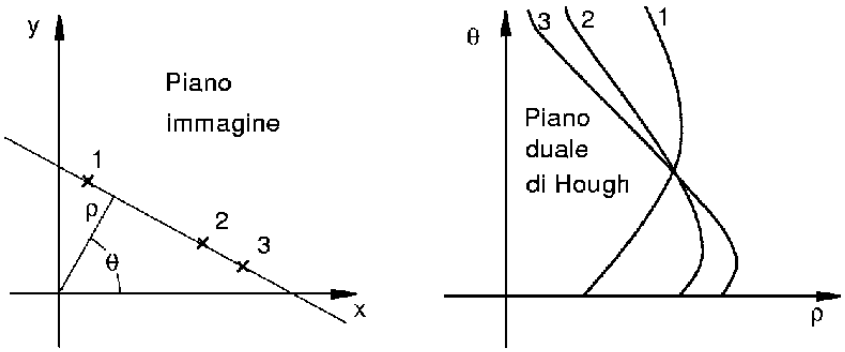


Figura 4.3: Trasformata di Hough

4.3 Stateflow

Stateflow è un tool di logica di controllo sviluppato da MathWorks e utilizzato per modellare sistemi reattivi attraverso macchine a stati e diagrammi flowchart.

Una tipica macchina a stati finiti costruita con *Stateflow* è formata da diversi elementi:

- Stati: possono essere esclusivi, se solo uno può essere attivo, e sono indicati da una linea continua; o paralleli, caratterizzati dalla linea tratteggiata e da un numero che ne indica la sequenza di esecuzione

- Transizioni e giunzioni: le transizioni sono frecce che portano da uno stato ad un altro; sopra di esse possono essere definite le condizioni che generano la transizione. Le giunzioni sono punti in cui arrivano più transizioni e da cui partono più transizioni, sono indicate da un cerchio

- Azioni: le azioni possono avvenire all'interno di uno stato o sulle transizioni. All'interno di uno stato sono raggruppate come segue:

Tabella 4.1: Tabella delle azioni di uno Stateflow

Azione	Effetto
entry	Le operazioni vengono eseguite all'ingresso nello stato
exit	Le operazioni vengono eseguite all'uscita dallo stato
during	Le operazioni vengono eseguite finchè lo stato è attivo
bind	Collega un evento allo stato in modo che sia lo stato a gestirlo

Le azioni associate ad una transizione fanno parte di una sintassi più complessa quale: *event_trigger[condition]condition_ action/transition action*

  - *event_trigger* è l'evento che scatena la transizione, se non specificato allora ogni evento effettua la transizione
  - *condition* è la condizione che determina se la transizione può essere effettuata o meno. Può essere booleana o temporale
  - *condition_action* sono azioni eseguite prima della transizione nel caso in cui la condizione sia vera. Sono inserite tra parentesi graffe e vengono eseguite anche se poi non si raggiungerà il nuovo stato
  - *transition_action* sono le azioni eseguite solo se viene raggiunto il nuovo stato. Sono precedute da /

4.4 Image Processing System

Si possiedono ora tutte le conoscenze necessarie alla comprensione dell'algoritmo sviluppato.

A tal fine, come prima cosa, si mostrerà la logica di analisi delle immagini catturate dalla videocamera posta nella parte inferiore del drone. Il sotto-sistema deputato a tale analisi è l'*Image Processing System* all'interno del blocco *Flight Control System*.

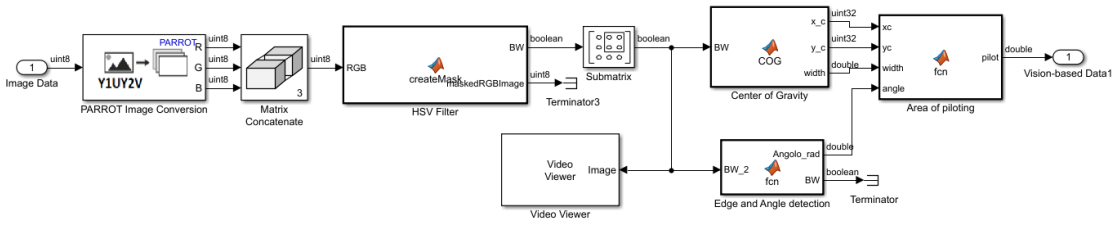


Figura 4.4: Image Processing System

Si procederà all'analisi blocco per blocco da sinistra verso destra:

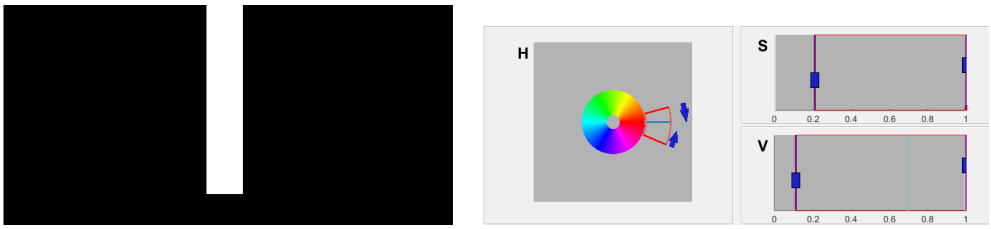
- PARROT Image Conversion: è un semplice convertitore di formato che riceve in ingresso le immagini dalla camera del drone in formato YUV e le restituisce in RGB. Viene scelto l'RGB a causa dell'assenza di un blocco Simulink che permetta la conversione in HSV, tuttavia non è un problema in quanto verrà effettuata in seguito. Si noti che l'immagine in ingresso ha risoluzione 120x160, non molto alta ma efficace al fine di alleggerire il peso computazionale dell'algoritmo

- Matrix Concatenate: riceve in ingresso i 3 vettori corrispondenti ai canali *R-G-B* del formato RGB e restituisce una matrice unica

- HSV Filter: si tratta di una *MATLAB Function* autogenerata dall'ambiente *colorThresholder*. Quest'app è fondamentale in quanto permette di caricare ed elaborare una immagine generando una maschera binaria. Nel nostro caso è stato utilizzato uno screenshot del percorso preso dal simulatore Simulink, come quello in Figura 4.5.

  Una volta caricata l'immagine è possibile scegliere lo spazio colore desiderato, per quanto descritto in precedenza è stato selezionato l'HSV, e regolare quindi gli intervalli di tonalità, saturazione e luminosità da accettare. Se la regolazione è svolta correttamente (non dovrebbe essere difficile data la presenza dell'interfaccia grafica che mostra in tempo reale gli effetti di ogni modifica) in uscita si ottiene una immagine binaria formata da "*1*" per ogni pixel facente parte del percorso e "*0*" per ogni pixel escluso.

Figura 4.5: Sezione rettilinea di percorso



(a) *Effetti della maschera* (b) *Modifiche alla maschera*

Figura 4.6: App colorThresholder

- Submatrix: l'immagine binaria ottenuta viene tagliata in modo da concentrarsi sul percorso da compiere, viene infatti eliminata la parte inferiore (contenente la porzione di tracciato già percorsa) e una parte laterale per evitare che vengano rilevate porzioni del tracciato successive ma molto vicine a quella attuale
- Video Viewer: è un semplice visualizzatore che mostra in finestra l'immagine binaria modificata
- Edge and Angle detection: all'interno di questa *MATLAB Function* si utilizza quanto descritto in precedenza al fine di identificare bordi e curve del percorso:

```
function [Angolo_rad,BW] = fcn(BW_2)
BW = edge(BW_2,'Canny');
[H Theta] = hough(BW);
P = houghpeaks(H,2);
Angolo_rad = deg2rad(mean(Theta(P(:,2))));
end
```

Utilizzando la funzione *edge* con opzione *Canny* si sfrutta l'algoritmo di Canny per il rilevamento dei bordi. All'immagine binaria che ne risulta viene applicata la funzione *hough* che ne calcola la trasformata di Hough e restituisce una matrice H le cui righe e colonne sono composte dai parametri ρ e θ e un

vettore *Theta* contenente l'angolo tra l'asse x e il vettore *rho* in gradi. Si cercano infine i picchi nella matrice *H* grazie alla funzione *houghpeaks* che restituisce una matrice contenente righe e colonne dei picchi. Vengono quindi restituiti l'angolo in radianti della curva e l'immagine elaborata. L'angolo sarà positivo in senso orario (curva a destra) e negativo in senso antiorario (curva a sinistra), tuttavia, dato che l'output di tale funzione è compreso nell'intervallo $[-\pi, \pi]$, per curve ad angolo maggiore di 180° si otterrà un valore negativo e per angoli minori di -180° un valore positivo

- Center of Gravity: questa funzione riceve in ingresso l'immagine mascherata e calcola il centro geometrico del percorso. É strutturata in modo da scorrere le righe alla ricerca degli "*1*" nella maschera, quando ne trova uno incrementa un contatore deputato al conteggio dei pixel del percorso e somma il valore di riga e colonna a due variabili apposite. Terminate le scansioni, divide i due valori totali per il numero di elementi, ottenendo così il centro geometrico:

$$x_c = \frac{x_1 + \cdots + x_n}{n} \qquad y_c = \frac{y_1 + \cdots + y_n}{n}$$

  Vi è infine il calcolo del numero di pixel di percorso nell'ultima riga dell'immagine al fine di ottenere una stima della larghezza del tracciato.

- Area of piloting: l'ultima funzione serve a stabilire la posizione del drone rispetto al percorso e quindi preparare eventuali manovre correttive da effettuare. A tal fine vengono utilizzate la posizione del drone rispetto al percorso e l'angolo calcolato dalla funzione *Edge and Angle detection*.

  Per il primo parametro l'immagine viene divisa in varie sezioni in modo da identificare la posizione del centro geometrico del tracciato rispetto al drone: ad esempio se l'angolo rilevato è nullo e il centro si trova nella fascia centrale dell'immagine, possiamo affermare che il drone sia abbastanza centrato su un percorso rettilineo, se il centro geometrico fosse spostato a destra o a sinistra sarebbe necessario il riallineamento orizzontale del drone. In Figura 4.7 è possibile vedere tale divisione, in particolare ciò che è esterno al riquadro verde viene eliminato da *Submatrix*; la sezione tra linee blu indica se il drone è centrato sulla linea (si è scelto di permettere un errore del 20% rispetto all'ampiezza del percorso); le due sezioni tra linea blu e marrone indicano al drone di girare verso sinistra (o destra) mentre prosegue in avanti; le due sezioni esterne indicano al drone di fermarsi e girare; infine la parte compresa tra le linee blu e inferiore alla linea dorata è la sezione di atterraggio. Nell'esempio riportato il centro geometrico del tracciato è indicato dal puntino nero, per cui il drone è abbastanza centrato.

  In base alla sezione in cui si trova il centro geometrico del percorso e all'angolo rilevato, viene restituito un intero, *pilot*, che serve da segnale di pilotaggio per

il drone. Ad esempio, nel caso in cui il centro del tracciato cada all'interno della zona centrale e l'angolo rilevato fosse molto piccolo (minore di 3°), si avrebbe $pilot = 0$, con un angolo maggiore di 3° e coordinate del centro geometrico del percorso interne alla seconda fascia, si avrebbe $pilot = 1$ e così via.

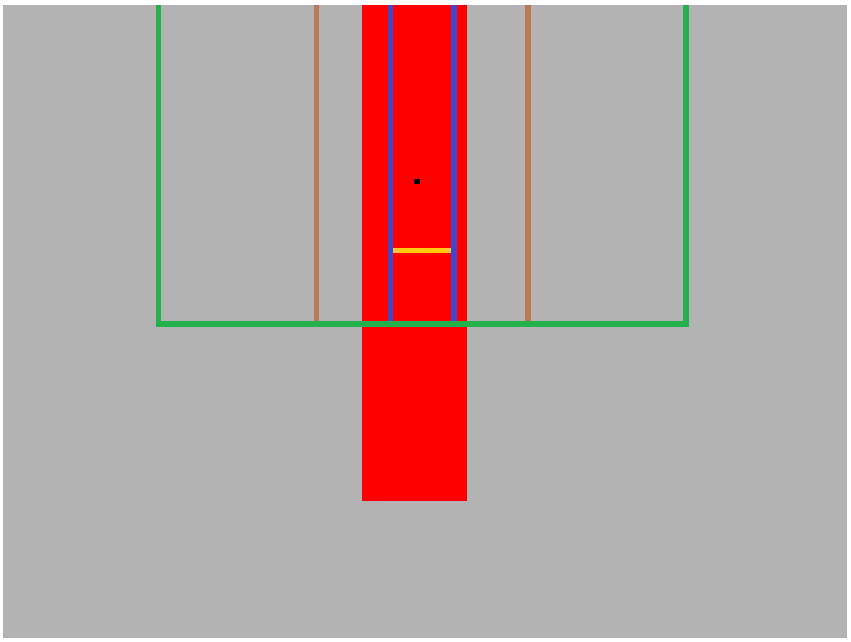


Figura 4.7: Divisione in zone del percorso precedente

4.5 Path Planning

Riassumiamo brevemente quanto descritto nel precedente paragrafo: l'immagine in arrivo dal drone viene filtrata alla ricerca del percorso, di cui viene calcolata la posizione del centro geometrico; in base al settore di appartenenza di tale punto all'interno dell'immagine, si attua una risposta codificata dalla variabile intera *pilot*. Prima di essere processata, la variabile *pilot*, output del sottosistema *Image Processing*, viene passata attraverso il blocco *Rate Transition* che ne converte il rate da quello originario a quello del sistema *Control System*. A questo punto è pronta a svolgere il proprio compito e a dare indicazioni al drone.

All'interno del sistema di *Path Planning* si trova la logica che trasforma le indicazioni qualitative date da *pilot* in indicazioni quantitative da dare ai controllori. Tale logica è implementata attraverso uno *Stateflow* composto da 9 stati di cui quello predefinito è quello di decollo (indicato dal pallino blu). Analizziamo quindi i vari stati:

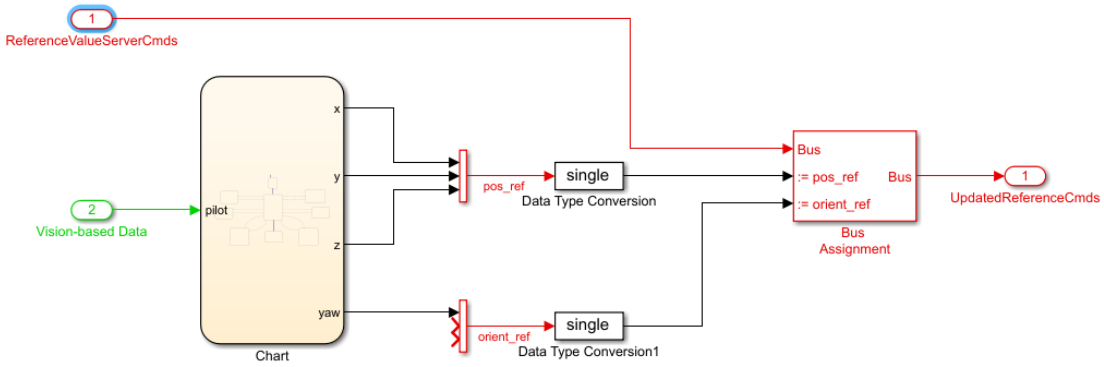


Figura 4.8: Sottosistema Path Planning

- Take_off: all'ingresso dello stato vengono posti a 0 i valori di x e y, mentre z viene posta a -1 al fine di far alzare il drone di un metro (l'asse z è rivolto verso il basso, per cui 1m di altezza equivale a -1). Terminata la fase di *entry*, dopo 2 secondi di attesa, si ha la transizione verso lo stato di navigazione

- Cruise1: lo stato *Cruise1* equivale a *pilot* = 0, ovvero avanzamento in linea retta. A tal fine si inizializzano a 0 due variabili, *dx* e *dy*, che indicano rispettivamente lo spostamento orizzontale e verticale da compiere e si riassegna all'angolo di yaw il valore che aveva precedentemente (per evitare rotazioni indesiderate). Il calcolo degli spostamenti deve tener conto del fatto che i valori di x e di y sono espressi nella terna inerziale, per questo, da semplici considerazioni trigonometriche, è possibile osservare che lo spostamento totale sul percorso corrisponde all'ipotenusa di un triangolo rettangolo con cateti gli spostamenti verticale e orizzontale. Definendo come costante lo spostamento totale, e pari a 0.0005, si ha semplicemente che:

  ```
  dx=0.0005*cos(yaw);
  dy=0.0005*sin(yaw);
  x=x+dx;
  y=y+dy;
  z=z;
  ```

  Ad ogni nuova immagine catturata dal Mambo viene ricalcolato il valore di *pilot* che quindi porterà lo stateflow ad una transizione verso lo stato corrispondente.

- Turn_left: gli spostamenti lineari rimangono gli stessi dello stato *Cruise1*, in più si ha una variazione costante dell'angolo di yaw che viene decrementato di 0.001 radianti per ciclo. Questo stato è ideale per curve lievi, in quanto

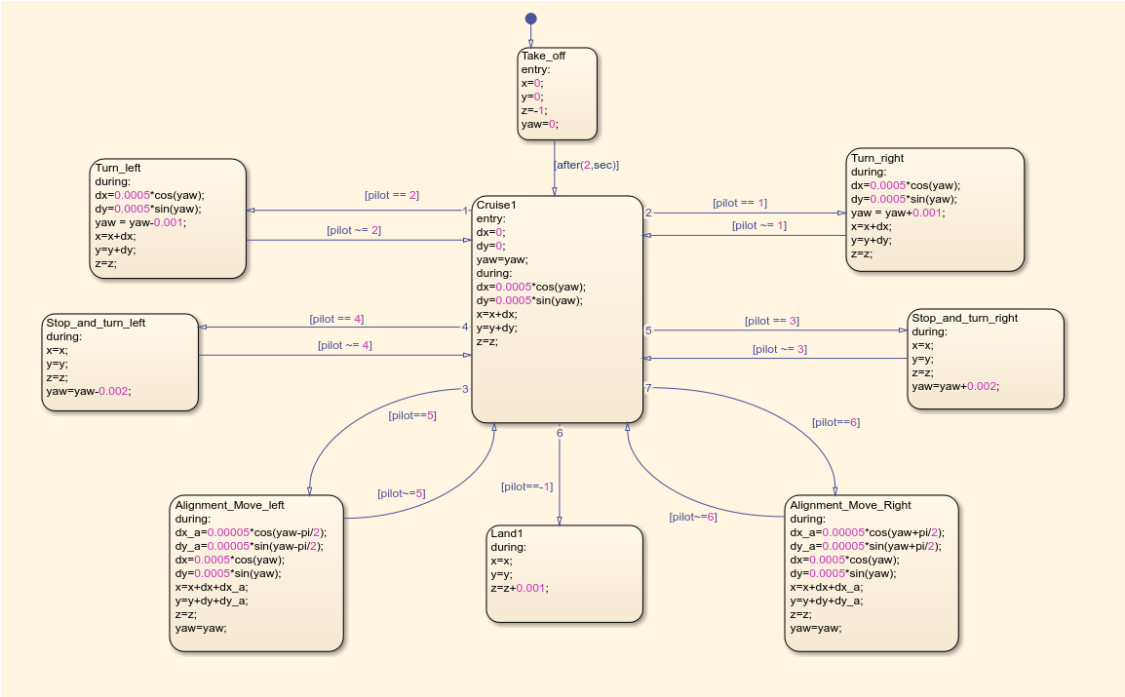


Figura 4.9: Diagramma a stati in Path Planning

permette la rotazione senza la necessità di fermare il drone, rendendo quindi più veloce e fluida la manovra. Corrisponde a $pilot = 2$

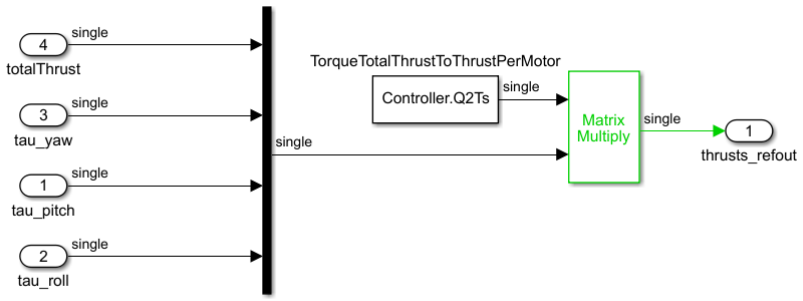
- Stop_and_turn_left: contrariamente allo stato precedente, in questo caso il drone si ferma nella posizione corrente e ruota in senso antiorario più velocemente rispetto a prima, in modo da evitare di andare fuori percorso muovendosi in corrispondenza di una curva troppo stretta. Corrisponde a $pilot = 4$

- Alignment_Move_left: questo stato è adibito al riallineamento del drone su un percorso rettilineo, ovvero si attiva quando il percorso non presenta curve ma il drone è spostato verso destra rispetto al centro geometrico del tracciato. In tal caso è sufficiente un movimento orizzontale per riposizionarsi correttamente. Per ottenere tale movimento vengono definiti due ulteriori spostamenti, dx_a e dy_a (a sta per "alignment"), da sommare ai due definiti in precedenza. Il ragionamento alla base della loro valorizzazione è molto semplice: per ottenere un movimento orizzontale verso sinistra, idealmente il drone dovrebbe effettuare una rotazione di 90° in senso antiorario, muoversi in avanti e poi ruotare nuovamente di 90° in senso orario; tale effetto si ottiene sottraendo $\pi/2$ all'angolo di yaw e facendo muovere il drone come se dovesse andare avanti. Corrisponde a $pilot = 5$

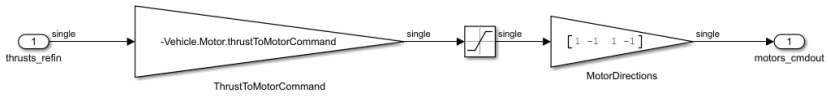
- Land1: l'ultimo stato è quello di atterraggio, il quale si attiva quando il centro geometrico del percorso si trova nel settore centrale e verso il fondo dell'immagine (ricordando che l'immagine è stata "ritagliata" in precedenza, il fondo corrisponde alla zona centrale dell'immagine originale, ovvero al centro di gravità del drone). Il quadricottero inizia quindi a ridurre lentamente la quota. Corrisponde a $pilot = -1$

I movimenti verso destra sono analoghi a quelli già descritti ma con una variazione dei segni.

I nuovi valori di x, y, z e yaw calcolati dal chart, vengono utilizzati per modificare il bus diretto al sottosistema *Controller* che li utilizzerà come riferimenti per i controllori progettati. Gli sforzi di controllo così calcolati, vengono quindi trasformati in riferimenti per i motori



(a) *Control Mixer*



(b) *Spinta dei motori*

Figura 4.10: Trasformazione da sforzo di controllo in spinta

4.6 Percorsi scelti

Per testare l'algoritmo sono stati scelti 3 percorsi diversi.

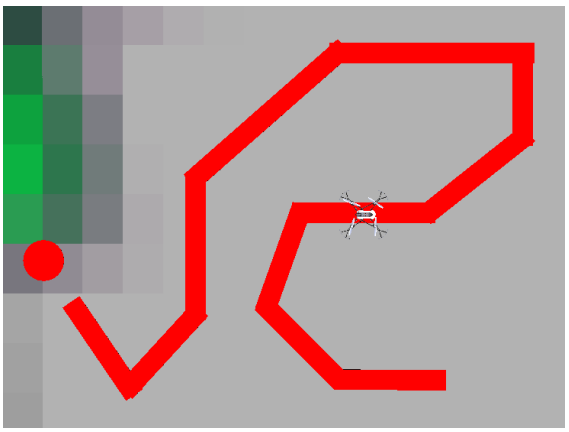
Il primo percorso presenta un rettilineo iniziale abbastanza lungo, fatto per verificare la capacità del drone di tenere la rotta in modo preciso; subito dopo il rettilineo si trova una curva dall'angolo molto stretto, fatta appositamente per portare al limite l'algoritmo, che nonostante qualche difficoltà (i valori di *pilot* che impongono al drone di ruotare senza muoversi generano un movimento all'indietro

invece di tenerlo perfettamente immobile, per questo quando riparte ruotando "taglia" un pezzetto di curva) riesce a far affrontare la curva al drone e a riallinearlo con il percorso. Sono poi presenti una serie di curve più dolci e di 90° che non generano problemi.

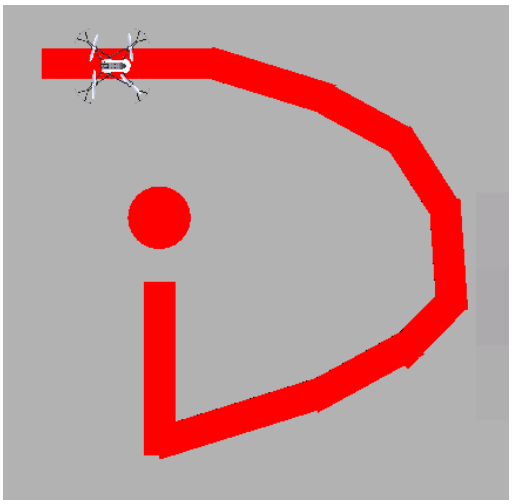
Il secondo percorso presenta curve meno pronunciate ma testa la capacità di mantenere l'allineamento con il percorso attuale, propone infatti due sezioni di tracciato molto ravvicinate, che inizialmente provocavano il cattivo funzionamento del drone. Proprio per risolvere tale problema è stata proposta la modifica dell'immagine attraverso il blocco *Submatrix*.

Il terzo percorso esamina il comportamento del drone su curve lievi, è infatti realizzato quasi come un semicerchio con una curva molto stretta alla fine.

(a) *Percorso 1: link 1 link 2*



(b) *Percorso 2: link 1 link 2*



(c) *Percorso 3: link 1 link 2*

Figura 4.11: 3 percorsi per la fase di testing dell'algoritmo

Capitolo 5

PID e controllori

In questa sezione si darà una analisi rapida dei PID e si progetterà un controllore alternativo per un grado di libertà del quadricottero.

5.1 PID

I PID, o regolatori industriali, sono strutture di controllo semplici e molto usate nell'automazione industriale. Si tratta di strutture predefinite dove il progettista deve scegliere alcuni parametri eseguendo esperimenti sull'impianto.

Si assume inoltre che il modello matematico del processo non sia noto, ma che il processo sia stabile esternamente (tutti i poli a parte reale negativa) e che non abbia poli nell'origine.

Il nome PID deriva dalle 3 componenti che lo costituiscono:

- Controllore proporzionale (P)
- Controllore integrale (I)
- Controllore derivativo (D)

P

Il controllore proporzionale fornisce un'uscita direttamente proporzionale all'ingresso:

$$m(t) = K_G e(t) \qquad G(s) = \frac{M(s)}{E(s)} = K_G \tag{5.1}$$

K_G è il guadagno del termine proporzionale.

Si noti che con un controllore di questo tipo non si può ottenere errore a regime permanente nullo per ingresso a gradino in quanto, se il processo non ha poli in $s = 0$, il sistema in catena aperta non ha poli nell'origine. Aumentando il guadagno si riduce l'errore ma si potrebbero avere problemi di stabilità, infine, comportandosi

come un amplificatore, è soggetto a saturazione dovuta ai limiti fisici dei dispositivi che si controllano.

I

Il controllore integrale fornisce un'uscita proporzionale all'area sottesa dalla curva errore-tempo:

$$m(t) = K_I \int_0^t e(\tau)d\tau \qquad G(s) = \frac{M(s)}{E(s)} = \frac{K_I}{s} \tag{5.2}$$

Il vantaggio è che si ottiene un sistema in catena chiusa di tipo 1, pertanto $\tilde{e} = 0$ e si ha astatismo rispetto ai disturbi in catena diretta. Lo svantaggio è che si inserisce un ritardo di fase di 90°. In generale peggiora il margine di fase, e quindi la stabilità, ma migliora le prestazioni di regime.

D

Il controllore derivativo fornisce un'uscita che non dipende dall'errore presente o passato ma dalla velocità alla quale esso varia:

$$m(t) = K_D \frac{d}{dt} e(t) \qquad G(s) = \frac{M(s)}{E(s)} = K_D s \tag{5.3}$$

Fornisce un anticipo di fase di 90° migliorando il margine di fase e quindi la stabilità del sistema in catena chiusa, ma peggiora le prestazioni di regime.

Combinando le equazioni 5.1, 5.2 e 5.3 si ottiene

$$m(t) = K_G e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \tag{5.4}$$

$$m(s) = K_G e(s) + K_I \frac{e(s)}{s} + K_D s e(s) \tag{5.5}$$

$$G(s) = \frac{m(s)}{e(s)} = K_G + \frac{K_I}{s} + K_D s = K_G\Big(1 + \frac{1}{\tau_I s} + \tau_D s\Big) \tag{5.6}$$

L'equazione 5.6, detta anche *textbook controller*, presenta alcuni problemi e criticità.

1. Il primo problema è la parte derivativa. Infatti l'equazione 5.3, da cui è composto il PID, presenta uno zero e nessun polo; questo, oltre a renderlo non fisicamente realizzabile, genera nel diagramma di Bode del modulo una semiretta a pendenza +20 dB/dec che quindi amplifica i segnali ad alta frequenza e dato che ad alta frequenza agiscono i rumori, diventa un amplificatore di rumori.

Per risolvere tale problematica si sostituisce l'espressione del controllore derivativo con

$$\tau_D s \longrightarrow \frac{\tau_D s}{1 + \frac{\tau_D s}{N}}$$

con $3 \leq N \leq 20$

In questo modo si inserisce un polo in $s = -\frac{N}{\tau_D}$ abbastanza lontano dallo zero, in modo da non interferire con l'azione derivatrice ma che allo stesso tempo sia in grado di tagliare l'amplificazione ad alte frequenze.

2. Il secondo problema riguarda sempre l'azione derivatrice, la quale, essendo un operatore illimitato, potrebbe generare uno sforzo di controllo troppo elevato. Infatti se consideriamo un ingresso $r(s)$ che varia come un gradino, otterremo un errore che varia quasi come un gradino (l'uscita ha una propria dinamica più lenta dell'ingresso, per cui cambia più lentamente) la cui derivata tende ad infinito, pertanto lo sforzo di controllo $m(t) = K_D \frac{d}{dt} e(t)$ assume valori molto elevati.

   Per risolvere tale problematica è possibile porre l'azione derivatrice in controreazione dall'uscita, ottenendo così

   $$m(s) = K_P e(s) + \frac{K_P}{\tau_I s} e(s) - \frac{K_P \tau_D s}{1 + \tau_1 s} y(s)$$

   In realtà dato che per valori di K_P abbastanza elevati anche l'azione proporzionale potrebbe generare sforzi di controllo grandi, a volte la si pone anch'essa in controreazione dall'uscita.

3. La terza criticità riguarda l'azione integrale. Ogni attuatore è assimilabile ad un blocco proporzionale, e in quanto tale può entrare in saturazione (consideriamo ad esempio come attuatore una valvola, una volta che è totalmente aperta non è possibile aumentare ancora lo sforzo di controllo). Se consideriamo un errore costante nel tempo, il suo integrale aumenta al passare del tempo e quindi aumenta lo sforzo di controllo a cui è sottoposto l'attuatore. Se l'attuatore va in saturazione è come se si interrompesse la controreazione in quanto l'uscita dell'attuatore rimane costante anche se lo sforzo di controllo aumenta o diminuisce, finchè non torna ad un valore inferiore a quello che manda in saturazione l'attuatore.

   Per risolvere tale problematica è possibile adottare due tecniche:

   - Azzeramento dell'integratore: l'azione integrale viene interrotta quando $m(t) = m_{max}$
   - Integrazione condizionata: si interrompe l'azione integrale prima di u_{max} in modo da evitare che sia l'azione proporzionale a mandare in saturazione l'attuatore (es. $u_{max} = 100$, azzero l'integrale per $u(t) = 70$)

Definita l'espressione generale del PID è facile notare che i parametri che caratterizzano un PID sono K_P,K_D e K_I i quali devono essere ricavati attraverso esperimenti sull'impianto. I modi per farlo sono due:

- Prima tecnica di Ziegler-Nichols (catena aperta): Si annullano le azioni integrale e derivativa e si aumenta K_G fino a portare il sistema in oscillazione (quasi instabilità). Si osservano quindi il periodo T_0 dell'oscillazione e il guadagno K_{G0} ottenuto. Attraverso formule empiriche si ricavano i valori dei parametri
- Seconda tecnica di Ziegler-Nichols (catena chiusa): Si assume che la risposta al gradino del sistema sia assimilabile a quella di un processo semplificato, quale

  $$P(s) = \frac{Ke^{-T_m s}}{1+Ts}$$

  e si cercano i valori di K,T_m e T che rendono le due risposte più simili possibile. A questo punto, attraverso formule empiriche, si ricavano i parametri cercati.

5.2 Luogo delle radici

Consideriamo l'espressione della funzione di trasferimento in catena chiusa

$$W(s) = \frac{F(s)}{1+F(s)} = \frac{k\frac{\prod_{i=0}^{m}(s-z_i)}{\prod_{i=1}^{n}(s-p_i)}}{1+k\frac{\prod_{i=0}^{m}(s-z_i)}{\prod_{i=1}^{n}(s-p_i)}} = \frac{k\prod_{i=1}^{m}(s-z_i)}{\prod_{i=1}^{n}(s-p_i)+k\prod_{i=1}^{m}(s-z_i)} \tag{5.7}$$

Si definisce *luogo delle radici* il luogo dei punti percorsi dalle radici dell'equazione

$$f(s,k) = \prod_{i=1}^{n}(s-p_i) + k\prod_{i=1}^{m}(s-z_i) = 0$$

nel piano di Gauss al variare di $k \in (-\infty, +\infty)$. In particolare, per $k > 0$ si ha il *luogo positivo*, mentre per $k < 0$ si ha il *luogo negativo*. Si noti che le radici dell'equazione 5.7 sono i poli della funzione di trasferimento in catena chiusa.

É possibile ricavare regole generali per tracciare nel piano di Gauss la variazione dei poli di $W(s)$ al variare di k:

1. Per k=0 il luogo delle radici parte dai poli in catena aperta (per k=0 i poli di W(s) coincidono con i poli di F(s))
2. Il luogo delle radici è simmetrico rispetto all'asse reale
3. Tutto l'asse reale appartiene al luogo. In particolare appartengono al luogo positivo le porzioni di asse reale che lasciano alla propria destra un numero dispari di poli e zeri di F(s) contati con la loro molteplicità; e appartengono al luogo negativo quelle che lasciano alla propria destra un numero pari di poli e zeri

4. I punti singolari sono le radici multiple dell'equazione $f(s,k)=0$:

$$\begin{cases} f(s,k)=0 \\ \dfrac{\partial f}{\partial s}(s,k)=0 \end{cases}$$

5. Per $k \longrightarrow +\infty$ m rami convergono sugli zeri e gli altri $n-m$ rami divergono all'infinito lungo $n-m$ semirette, detti asintoti del luogo, che partono da un punto sull'asse reale, detto centro degli asintoti.

$$s_0 = \frac{\sum_{i=1}^{n} p_i - \sum_{i=1}^{m} z_i}{n-m}$$

Definite le regole di tracciamento del luogo delle radici, è possibile passare alla sintesi di un controllore utilizzando tale grafico.

Sintesi in s

La sintesi con il luogo delle radici è un tipo di sintesi detto "per tentativi" in quanto si procede con la progettazione di vari controllori al fine di soddisfare, volta per volta, sempre più specifiche aggiungendo delle funzioni compensatrici.

Per le sintesi di questo tipo si procede prima di tutto con l'individuazione delle specifiche richieste, ad esempio specifiche sulla stabilità, sull'errore di regime permanente, sull'azione dei disturbi, sul transitorio o sulla sensibilità alle variazioni parametriche.

Si identificano poi le specifiche univoche (quelle che possono essere soddisfatte solo in un modo, ad esempio per ottenere astatismo rispetto ai disturbi in catena diretta è necessario un polo in $s=0$ nella funzione di trasferimento in catena diretta) e le specifiche lasche (ottenibili in più modi). Le specifiche univoche sono rappresentate da particolari richieste riguardanti la funzione di trasferimento, ad esempio la presenza di poli nell'origine, mentre le specifiche lasche sono definite come appartenenza dei poli ad una regione del piano.

Si procede quindi con la progettazione di un controllore di primo tentativo che soddisfi le specifiche univoche, composto da un guadagno K_G e da r poli in $s=0$.

$$\hat{G}(s) = \frac{K_G}{s^r}$$

A questo punto si traccia il luogo delle radici della funzione di trasferimento a ciclo chiuso ottenuta con il controllore di primo tentativo

$$\hat{F}(s) = \hat{G}(s)P(s) \tag{5.8}$$

e si verifica se siano soddisfatte anche le specifiche lasche.

In caso di risposta negativa sarà necessario progettare delle funzioni compensatrici che adeguino la funzione di trasferimento senza intaccare le specifiche univoche.

Per raggiungere l'obiettivo senza aumentare la complessità della funzione di trasferimento, e quindi quella del luogo delle radici, è possibile introdurre delle coppie polo-zero tali da spostare il centro degli asintoti all'interno della zona di specifica, in modo che i rami, che divergono seguendo gli asintoti, convergano all'interno della porzione di piano richiesta per valori di K accettabili.

L'introduzione delle coppie polo-zero risulta utile anche al fine di semplificare la funzione di trasferimento, in quanto è possibile scegliere poli e zeri coincidenti rispettivamente con zeri e poli della $\hat{F}(s)$ e interni alla zona di specifica, al fine di semplificarli e rendere più semplice il grafico.

Soddisfatte tutte le specifiche, otterremo la funzione di trasferimento del controllore:

$$G(s) = \hat{G}(s)\frac{\prod_{i=1}^{k1}(s - z_i)}{\prod_{i=1}^{k2}(s - p_i)}$$

5.3 Sintesi di un controllore per la z con il luogo delle radici

In questa trattazione si è scelto di mostrare le potenzialità del Simulink nella progettazione di sistemi di controllo, utilizzando il toolbox apposito di MATLAB per elaborare un sistema che controlli un grado di libertà del drone, in particolare la z. Prima di tutto è necessario calcolare la funzione di trasferimento relativa alla z.

Ricordando quanto descritto nelle equazioni 2.7 e 2.19 riguardo la dinamica della z, e considerando una condizione di hovering in cui il drone è approssimativamente fermo a mezz'aria, si ottiene che gli angoli di pitch e roll sono prossimi a 0, pertanto $\cos\theta = \cos\phi \simeq 1$ da cui

$$\begin{aligned} \ddot{z} &\simeq -g + \frac{U_1}{m} \\ \dot{z} &\simeq w \end{aligned}$$

Possiamo quindi considerare la z come approssimativamente disaccoppiata rispetto agli altri gradi di libertà, isolandone la dinamica come se fosse un processo a parte.

Scegliendo come vettore di stato $\begin{vmatrix} z \\ \dot{z} \end{vmatrix}$, e ricordando l'espressione di un sistema in spazio di stato:

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

si ottiene

$$\begin{cases} \begin{vmatrix} \dot{z} \\ \ddot{z} \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ 0 & 0 \end{vmatrix} \cdot \begin{vmatrix} z \\ \dot{z} \end{vmatrix} + \begin{vmatrix} 0 \\ \frac{1}{m} \end{vmatrix} U_1 + \begin{vmatrix} 0 \\ -g \end{vmatrix} \\ y = \begin{vmatrix} 1 & 0 \end{vmatrix} \begin{vmatrix} z \\ \dot{z} \end{vmatrix} \end{cases}$$

Osservando le matrici nel sistema linearizzato di MATLAB (Appendice A) è possibile notare alcune differenze: per prima cosa il blocco costante dato dall'accelerazione di gravità viene trascurato nelle matrici e aggiunto in seguito, inoltre sono presenti alcune costanti moltiplicative come *Vehicle.Motor.thrustToMotorCommand* o quella presente nella matrice *Controller.Q2Ts*, che rendono la matrice B diversa.

In particolare, prendendo in considerazione i valori non nulli nelle matrici A, B, C e D delle righe corrispondenti alla z (riga 9) ed a w (riga 6), si ottiene:

$$\begin{cases} \begin{vmatrix} \dot{z} \\ \ddot{z} \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ 0 & 0 \end{vmatrix} \cdot \begin{vmatrix} z \\ \dot{z} \end{vmatrix} + \begin{vmatrix} 0 \\ 0.0096 \end{vmatrix} U_1 \\ y = \begin{vmatrix} 1 & 0 \end{vmatrix} \begin{vmatrix} z \\ \dot{z} \end{vmatrix} \end{cases}$$

Da cui, per calcolare la funzione di trasferimento del processo

$$P(s) = C\Big(sI - A\Big)^{-1} B + D = \begin{vmatrix} 0 & 1 \end{vmatrix} \cdot \left(s \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} - \begin{vmatrix} 0 & 1 \\ 0 & 0 \end{vmatrix} \right)^{-1} \cdot \begin{vmatrix} 0 \\ 0.0096 \end{vmatrix} =$$
$$= \begin{vmatrix} 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1/s & 1/s^2 \\ 0 & 1/s \end{vmatrix} \cdot \begin{vmatrix} 0 \\ 0.0096 \end{vmatrix} = \begin{vmatrix} 1/s & 1/s^2 \end{vmatrix} \cdot \begin{vmatrix} 0 \\ 0.0096 \end{vmatrix} = 6/625 s^2$$

Moltiplicando per le costanti viste precedentemente, il cui valore è consultabile su MATLAB scrivendo nella Command Window il loro nome, o cercandole all'interno delle strutture dati definite nel workspace, dato che

$$Vehicle.Motor.thrustToMotor = 1530.7$$
$$Controller.Q2Ts = 0.25$$

si ottiene che la funzione di trasferimento totale è data da

$$P(s) = 1530.7 \cdot 0.25 \cdot \frac{0.0096}{s^2} = \frac{3.67}{s^2} \tag{5.9}$$

Per progettare un controllore con il luogo delle radici è stato utilizzato il toolbox *Control System Design* che fornisce l'ambiente *sisotool* sul quale sono disponibili varie funzionalità, come i diagrammi di bode, la risposta al gradino e, per l'appunto, il luogo delle radici.

Progettare un controllore in questo modo è molto intuitivo in quanto è sufficiente aggiungere quanti poli e gli zeri si desidera e spostarli nella posizione voluta.

In tempo reale sarà possibile osservare la variazione dei diagrammi di Bode, del luogo delle radici e della risposta al gradino nonchè l'espressione della funzione di trasferimento del controllore.

Tramite questo ambiente si è realizzato il seguente controllore:

$$G(s) = \frac{100(s+4)(s+16.5)}{(s+80)(s+40)}$$

Dato che il sottosistema *Flight Control System* lavora a tempo discreto con tempo di campionamento di $Ts = 0.005sec$ è stato necessario discretizzare il controllore appena descritto. Anche in questo caso MATLAB ha reso il compito molto più semplice grazie alla sua funzione predefinita *c2d*, per cui tramite il comando *c2d(Gs,0.005,'zoh')* si è ottenuto il controllore a tempo discreto con tempo di campionamento pari a quello del sottosistema:

$$G(z) = \frac{100(z-0.9873)(z-0.9031)}{(z-0.8187)(z-0.6703)}$$

La funzione di trasferimento in catena chiusa è quindi data da

$$F(s) = G(s)P(s) = \frac{100(s+4)(s+16.5)}{(s+80)(s+40)} \cdot \frac{3.67}{s^2} = \frac{367(s+4)(s+16.5)}{s^2(s+40)(s+80)}$$

Ne possiamo osservare i diagrammi di Bode, il luogo delle radici e la risposta a gradino:

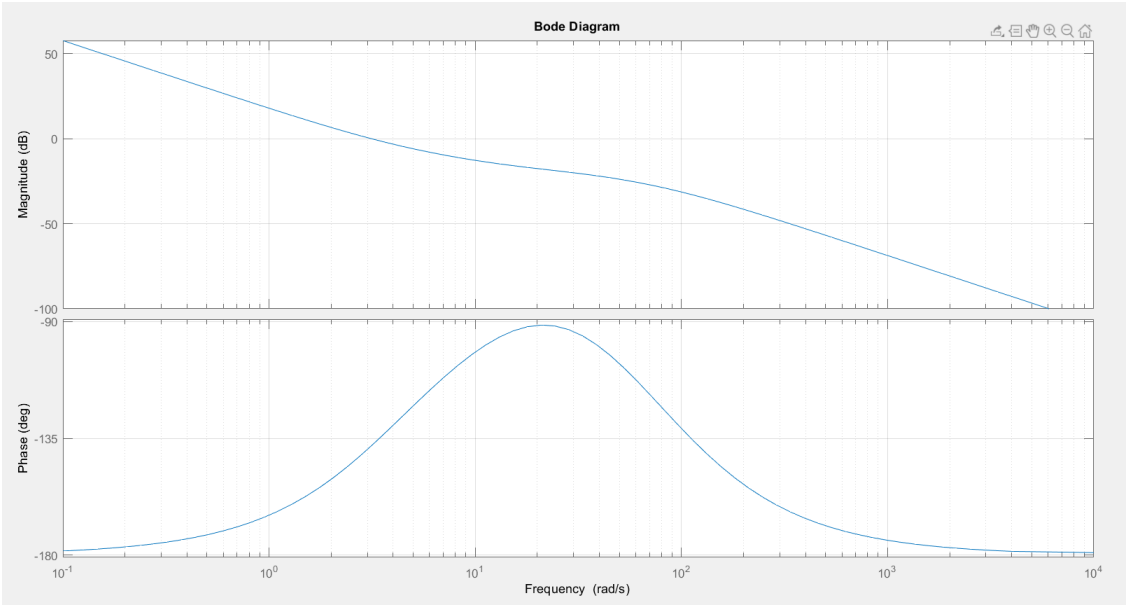


Figura 5.1: Diagrammi di Bode di F(s)

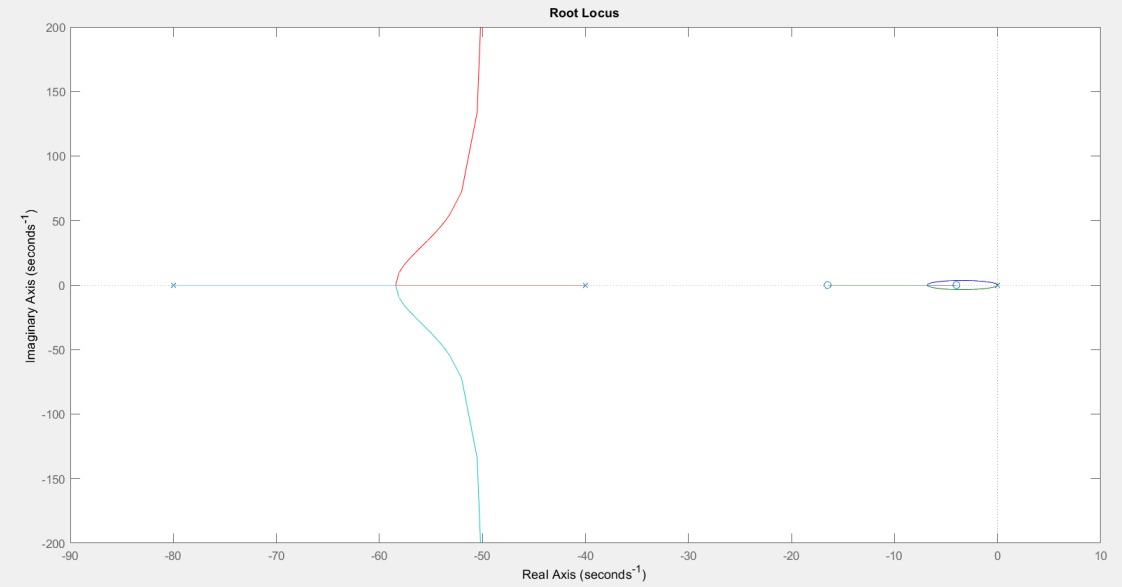


Figura 5.2: Luogo delle radici di F(s)

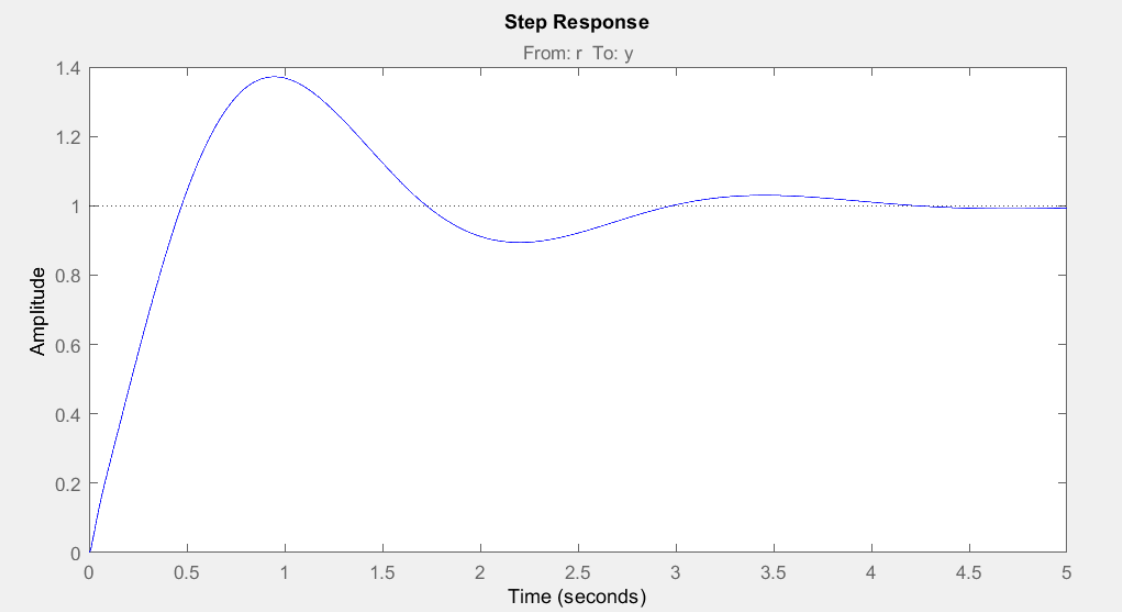


Figura 5.3: Risposta al gradino di Fs

É possibile utilizzare tale controllore dal sottosistema *Controller* ed entrando nel blocco dedicato all'asse z, ovvero *gravity feedforward/equilibrium thrust*. In Figura 5.4 è possibile notare come il segnale di controllo sia diviso in due: il primo riguarda

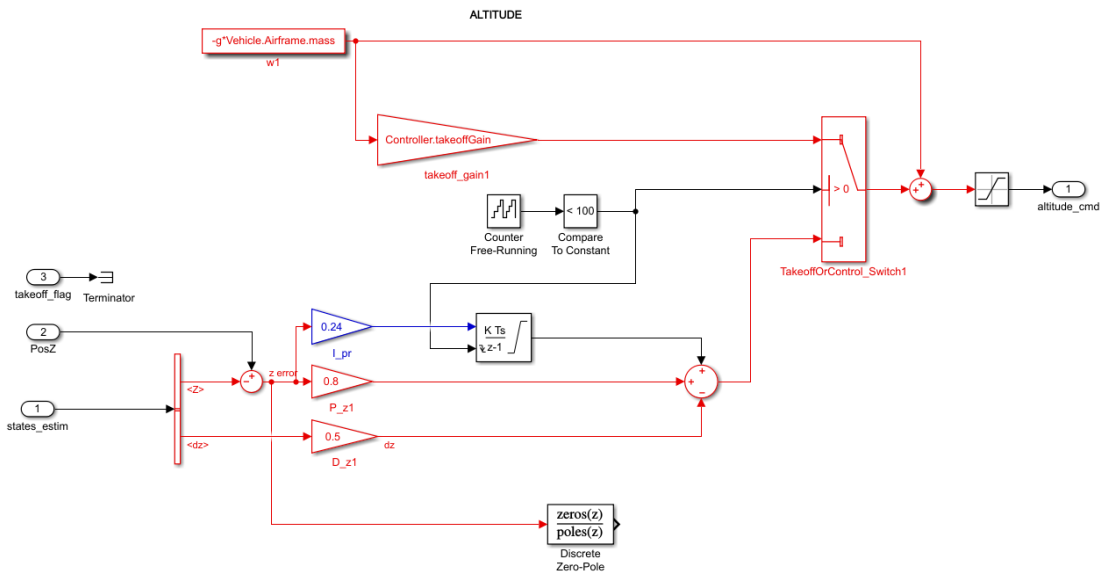


Figura 5.4: Blocco controllore asse z

l'azione di decollo ed ha un proprio guadagno, il secondo è l'azione di controllo post-decollo ed è collegata all'uscita del PID. Lo sforzo di controllo corretto viene scelto tramite uno switch, e qualunque esso sia, gli viene sottratta la forza peso, calcolata come $-g \cdot m$.

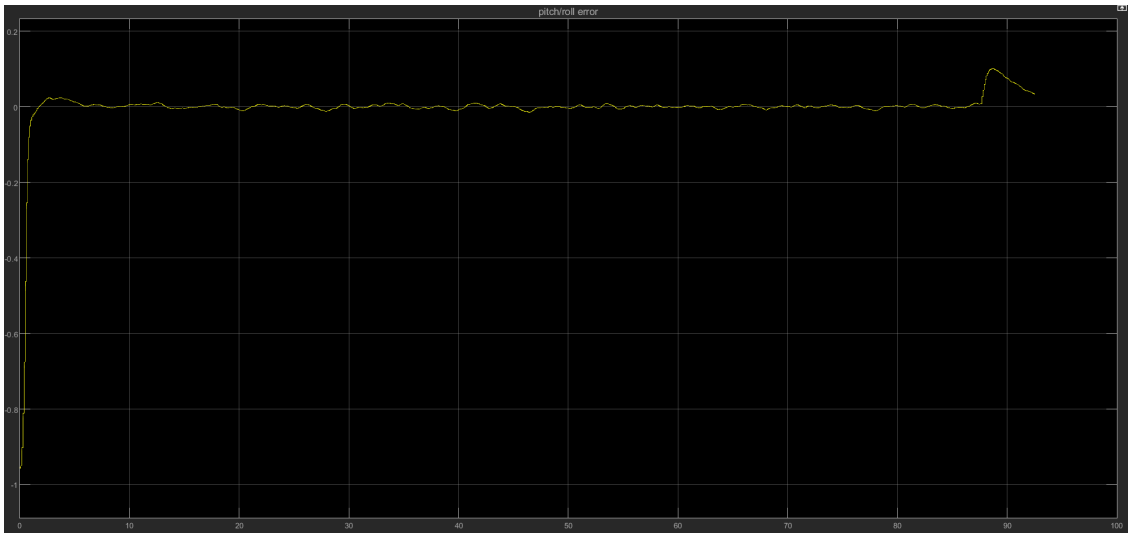
Per utilizzare il controllore progettato è quindi necessario eliminare il collegamento tra il PID e lo switch e collegare un blocco *Discrete Zero-Pole* con ingresso l'errore della z e funzione di trasferimento quella del controllore appena visto.

Per osservare il miglioramento apportato è sufficiente visualizzare l'andamento dell'errore sulla z, definito come la differenza tra il riferimento imposto dall'algoritmo e il valore stimato dai sensori.

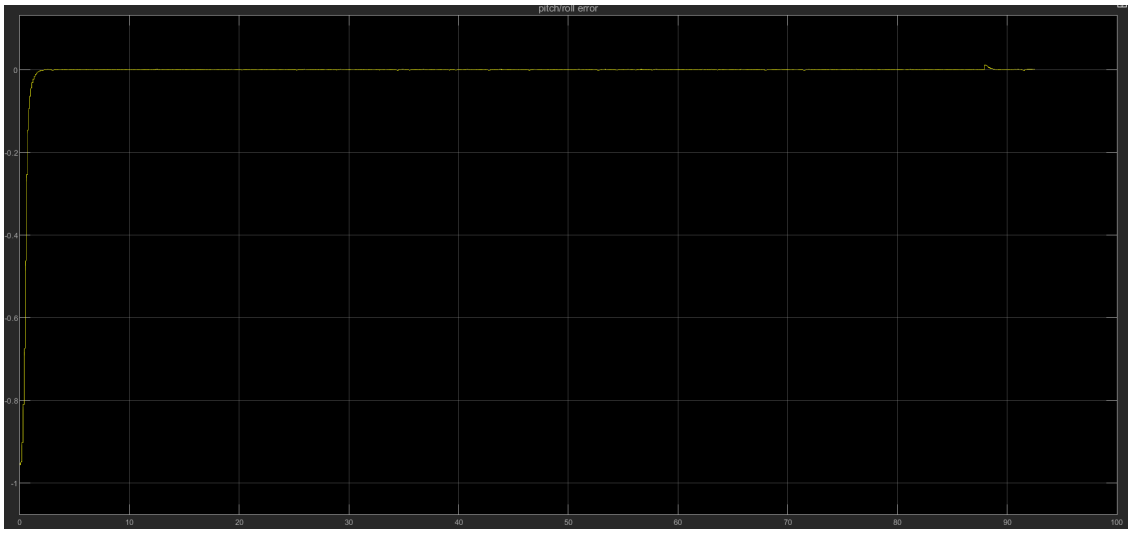
Dalla figura 5.5 è facilmente visibile come il PID, sebbene riesca a portare l'errore a 0 in tempi accettabili, sia sottoposto a continue oscillazioni che portano il drone a movimenti continui sull'asse z. Per quanto riguarda il controllore progettato con il luogo delle radici, l'andamento dell'errore è molto più costante e lineare, permettendo quindi al drone di mantenere la quota in modo più preciso e di limitarne le oscillazioni.

Un secondo parametro da prendere in considerazione sono gli indici prestazionali:

- IAE (Integral Absolute Error): è semplicemente l'integrale del modulo dell'errore nel tempo
$$IAE = \int_0^t |e(\tau)| d\tau$$
- ISE (Integral Squire Error): integra il quadrato dell'errore nel tempo, in modo da amplificare l'effetto degli errori di grandi dimensioni e ridurre quello degli



(a) *Errore sulla z con il PID*



(b) *Errore sulla z con il controllore progettato*

Figura 5.5: Errore sulla z con PID e luogo delle radici

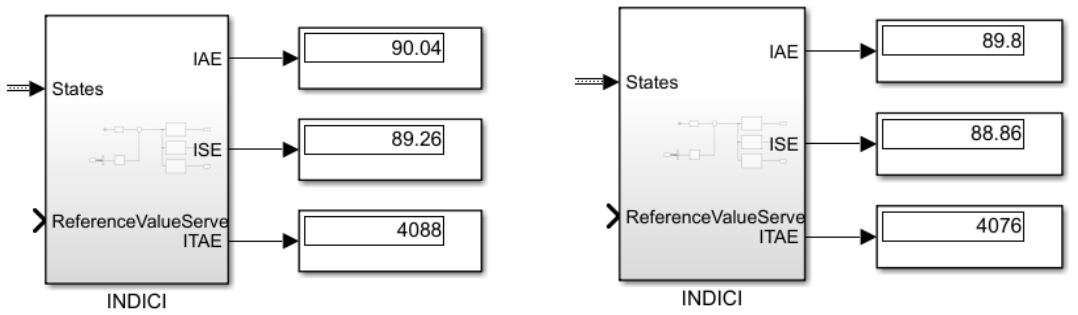
errori piccoli

$$ISE = \int_0^t e(\tau)^2 d\tau$$

- ITAE (Integral Time Absolute Error): integra il valore assoluto dell'errore nel tempo pesandolo con il tempo stesso, in modo da penalizzare l'effetto degli errori ad istanti lontani

$$ITAE = \int_0^t \tau \cdot |e(\tau)| d\tau$$

Anche in questo caso, confrontando i valori degli indici di prestazione ottenuti con il PID e quelli ottenuti con il controllore progettato con il luogo delle radici, si nota un miglioramento fornito da quest'ultimo.

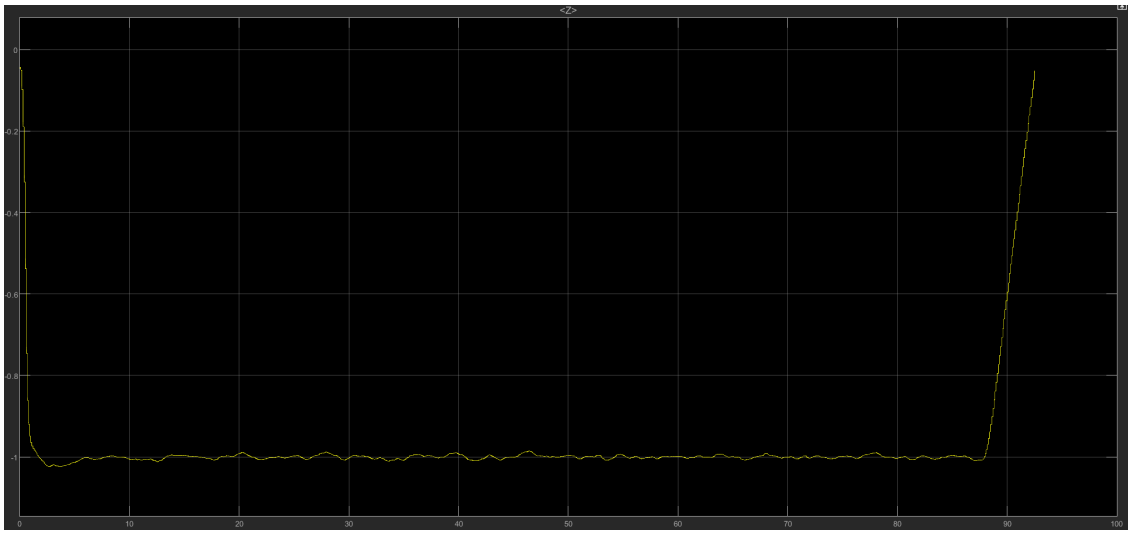


(a) *Indici di prestazione PID* (b) *Indici di prestazione luogo delle radici*

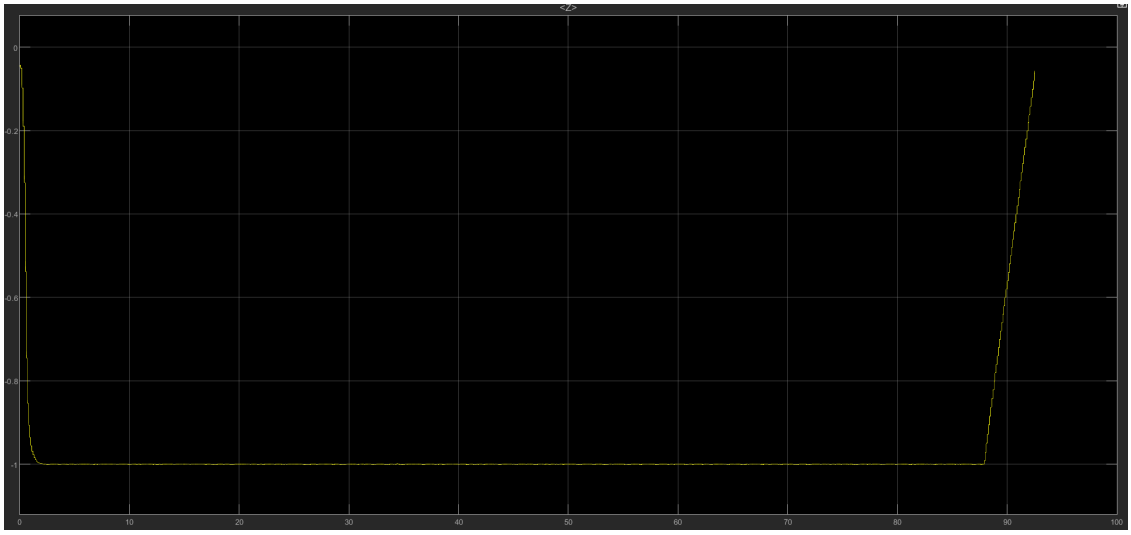
Figura 5.6: Indici di prestazione con PID e luogo delle radici

Infine, nonostante sia facilmente intuibile da quanto detto in precedenza, come ulteriore verifica è possibile operare un confronto tra la stima della z nel caso del PID e nel caso del controllore progettato. Analogamente a quanto osservato per l'errore, anche in questo caso il controllore progettato con il luogo delle radici riduce le oscillazioni ottenendo un grafico più preciso e lineare.

Grazie al modello Simulink è quindi stato possibile progettare con relativa semplicità un controllore che migliori le prestazioni del PID.



(a) *Stima della z con PID*



(b) *Stima della z con luogo delle radici*

Figura 5.7: Stima della z con PID e luogo delle radici

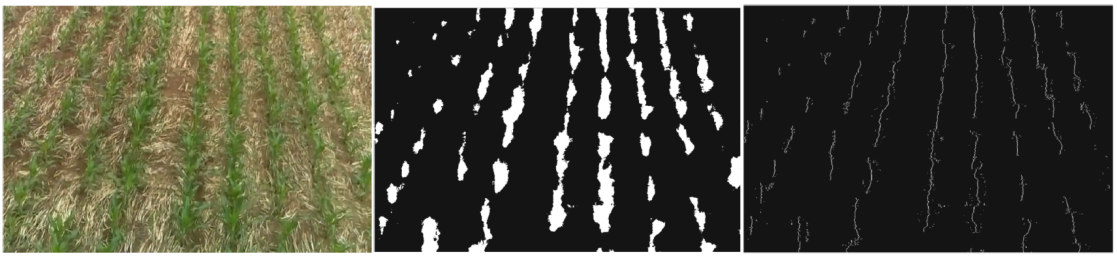
Capitolo 6

Conclusioni

In questa trattazione si è visto il funzionamento generale dei quadricotteri e la modalità del loro movimento, per poi passare alla trattazione del modello *Simulink* implementato da MATLAB per il drone *Mambo Parrot*. Tale modello è stato utilizzato per l'implementazione di un algoritmo di inseguimento della traiettoria e di un controllore per la z utilizzando la sintesi per tentativi con il luogo delle radici.

L'implementazione dell'algoritmo mostrata è facilmente adattabile a numerose altre tipologie di problemi, ad esempio al posto di un percorso al suolo si potrebbe avere un percorso a mezz'aria individuato da cornici circolari, quadrate ecc; in tal caso basterebbe utilizzare un drone con videocamera frontale e modificare la logica di inseguimento affinchè il centro geometrico del quadrato risulti esattamente al centro dell'immagine.

Alcuni possibili utilizzi dell'algoritmo proposto potrebbero essere in campo agricolo, ad esempio con quadricotteri spargi-semi la cui rotta può essere stabilita tramite una linea sul terreno o tramite una serie di paletti colorati, o per l'analisi e il controllo delle piantine; o ancora in ambito stradale utilizzando la segnaletica orizzontale. Si riportano come esempio le immagini in Figura 6.1 tratte da [3].



(a) *Immagine piantine* (b) *Immagine binaria* (c) *Immagine finale*

Figura 6.1: Utilizzo dell'algoritmo in campo agrario

Appendice A

Modello lineare MATLAB

Per visualizzare il modello lineare del drone in Simulink, è sufficiente aprire dal workspace la struttura *linsys*. Lì saranno presenti le matrici A, B, C e D, e i vettori di stato, di ingresso e di uscita:

linsys.A

	1	2	3	4	5	6	7	8	9	10	11	12
1	1.0000	1.3319e-14	0	-8.1378e-13	-8.1378e-13	-4.0064e-19	0	0	0	0.0050	1.2699e-14	-2.4237e-14
2	1.0186e-14	1.0000	0	6.2251e-13	6.2251e-13	-3.4672e-19	0	0	0	9.7068e-15	0.0050	2.4235e-14
3	-3.4247e-15	5.1406e-26	1	-3.1414e-24	-2.8361e-24	7.1449e-24	0	0	0	-7.3027e-18	-2.4119e-14	0.0050
4	-4.3785e-13	-0.0490	0	1.0000	-1.7828e-11	3.4346e-15	0	0	0	-2.7566e-13	5.9100e-04	-2.2402e-15
5	0.0490	4.6538e-13	0	-1.8944e-11	1.0000	-3.4340e-15	0	0	0	-5.9028e-04	2.9325e-13	-2.3572e-15
6	8.3991e-17	8.3991e-17	0	-3.4247e-15	3.4247e-15	1	0	0	0	1.6382e-15	1.7480e-15	-6.9372e-31
7	-7.3027e-16	-1.2262e-04	1.6494e-15	0.0050	-4.4633e-14	-2.4233e-14	1	0	0	-6.9596e-16	1.5822e-06	-9.8112e-19
8	1.2262e-04	7.7632e-16	1.7578e-15	-4.7440e-14	0.0050	2.4232e-14	0	1	0	-1.5809e-06	7.4030e-16	-1.0026e-18
9	-2.2438e-15	-2.3522e-15	0	2.4233e-14	-2.4232e-14	0.0050	0	0	1	7.6626e-18	7.6688e-18	1.0344e-31
10	-7.9832e-12	7.9832e-12	0	-3.2485e-10	-3.2485e-10	-1.5993e-16	0	0	0	0.9878	5.0285e-12	1.5793e-15
11	6.1068e-12	-6.1068e-12	0	2.4859e-10	2.4859e-10	-1.3846e-16	0	0	0	3.8437e-12	0.9900	-2.0429e-15
12	1.7746e-24	1.2198e-24	0	-4.9746e-23	7.2352e-23	2.8580e-21	0	0	0	4.8862e-16	4.2715e-16	1.0000

Figura A.1: Matrice A

linsys.B

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	6.1566e-06	6.1566e-06	-6.1566e-06	-6.1566e-06	3.8353e-19	0	0	0	-1.3577e-15	-1.3577e-15	-6.6842e-22	0	0	0
2	5.0092e-06	-5.0092e-06	-5.0092e-06	5.0092e-06	-1.7863e-19	0	0	0	1.0384e-15	1.0384e-15	-5.7834e-22	0	0	0
3	-1.9711e-07	-1.9711e-07	-1.9711e-07	-1.9711e-07	-1.3553e-20	0	0	0	-5.2402e-27	-4.7315e-27	1.1908e-26	0	0	0
4	6.3614e-07	-6.3614e-07	-6.3614e-07	6.3614e-07	-5.8827e-16	0	0	0	0.0050	-4.4633e-14	2.4250e-14	0	0	0
5	-7.8184e-07	-7.8184e-07	7.8184e-07	7.8184e-07	5.8197e-16	0	0	0	-4.7440e-14	0.0050	-2.4249e-14	0	0	0
6	-5.1848e-05	5.1848e-05	-5.1848e-05	5.1848e-05	-0.0414	0	0	0	-8.5618e-18	8.5617e-18	0.0050	0	0	0
7	1.0953e-09	-1.0953e-09	-1.0953e-09	1.0953e-09	5.0071e-16	0	0	0	1.2500e-05	-7.4441e-17	1.4311e-20	0	0	0
8	-1.3465e-09	-1.3465e-09	1.3465e-09	1.3465e-09	-5.0072e-16	0	0	0	-7.9136e-17	1.2500e-05	-1.4309e-20	0	0	0
9	-1.2962e-07	1.2962e-07	-1.2962e-07	1.2962e-07	-1.0357e-04	0	0	0	6.0589e-17	-6.0587e-17	1.2500e-05	0	0	0
10	0.0025	0.0025	-0.0025	-0.0025	2.3451e-16	0	0	0	-8.1378e-13	-8.1378e-13	-4.0064e-19	0	0	0
11	0.0020	-0.0020	-0.0020	0.0020	3.9397e-17	0	0	0	6.2251e-13	6.2251e-13	-3.4672e-19	0	0	0
12	-7.8843e-05	-7.8843e-05	-7.8843e-05	-7.8843e-05	-3.4694e-18	0	0	0	-1.2434e-25	1.8090e-25	7.1449e-24	0	0	0

Figura A.2: Matrice B

linsys.C

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	-9.8100	0	-3.5815e-09	-3.5815e-09	1.9948e-15	0	0	0	-5.6860e-11	0.1434	8.6632e-16
2	9.8100	0	0	-3.8094e-09	-3.8094e-09	-1.8754e-15	0	0	0	-0.1434	6.0478e-11	-8.6632e-16
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	-4.8483e-12	-4.8481e-12	-1.0000	0	0	0	0	0	0	0	0	0
6	1.0034e-18	1.0000	-4.8481e-12	0	0	0	0	0	0	0	0	0
7	0	0	1.0000	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	-1.0000	1.0034e-18	-4.8483e-12	0	0	0	0	0	0	0	0	0
10	0	-1.0000	0	0	0	0	0	0	0	0	0	0
11	1.0000	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	0	0
14	0	1	0	0	0	0	0	0	0	0	0	0
15	0	0	1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	9.0026e-06	0	0	0	0	0
17	0	0	0	0	0	0	0	1.2127e-05	0	0	0	0
18	0	0	0	0	0	0	0	0	-1.0000	0	0	0
19	0	0	0	0	0	0	0	0	0	1	0	0
20	0	0	0	0	0	0	0	0	0	0	1	0
21	0	0	0	0	0	0	0	0	0	0	0	1
22	0	0	0	1	0	0	0	0	0	0	0	0
23	0	0	0	0	1	0	0	0	0	0	0	0
24	0	0	0	0	0	1.0000	0	0	0	0	0	0
25	1.5993e-24	1.0649e-19	3.2988e-13	1	-1.0033e-18	-4.8483e-12	0	0	0	0	0	0
26	-1.0649e-19	1.0685e-37	3.5156e-13	1.0033e-18	1	4.8481e-12	0	0	0	0	0	0
27	-3.2988e-13	-3.5156e-13	0	4.8483e-12	-4.8481e-12	1.0000	0	0	0	0	0	0
28	0	0	0	0	0	0	1	0	0	0	0	0
29	0	0	0	0	0	0	0	1	0	0	0	0
30	0	0	0	0	0	0	0	0	1	0	0	0
31	0	0	0	-6.5370e-08	-6.5370e-08	-3.2183e-14	0	0	0	-2.4600	1.0378e-09	3.1780e-13
32	0	0	0	4.9967e-08	4.9967e-08	-2.7830e-14	0	0	0	7.9327e-10	-2.0000	-4.1062e-13
33	0	0	0	-9.9439e-21	1.4476e-20	5.7159e-19	0	0	0	9.8327e-14	8.5858e-14	-9.0652e-09

Figura A.3: Matrice C

linsys.D

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	-7.7532e-17	7.7532e-17	-7.7532e-17	7.7532e-17	-1.0342e-13	0	0	0	1.0000	1.0033e-18	4.8483e-12	0	0	0
2	7.5998e-17	-7.5998e-17	7.5998e-17	-7.5998e-17	1.0219e-13	0	0	0	-1.0033e-18	1.0000	-4.8481e-12	0	0	0
3	-0.0104	0.0104	-0.0104	0.0104	-8.2855	0	0	0	0	0	1.0000	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0.4945	0.4945	-0.4945	-0.4945	4.2633e-14	0	0	0	0	0	0	0	0	0
32	0.4021	-0.4021	-0.4021	0.4021	1.4211e-14	0	0	0	0	0	0	0	0	0
33	-0.0158	-0.0158	-0.0158	-0.0158	-4.4409e-16	0	0	0	0	0	0	0	0	0

Figura A.4: Matrice D

linsys.StateName

	1
1	phi theta psi(1)
2	phi theta psi(2)
3	phi theta psi(3)
4	ub,vb,wb(1)
5	ub,vb,wb(2)
6	ub,vb,wb(3)
7	xe,ye,ze(1)
8	xe,ye,ze(2)
9	xe,ye,ze(3)
10	p,q,r(1)
11	p,q,r(2)
12	p,q,r(3)

(a) *Vettore di stato*

linsys.InputName

	1
1	Actuators(1)
2	Actuators(2)
3	Actuators(3)
4	Actuators(4)
5	AtmosphereBus.air_density
6	AtmosphereBus.air_temp
7	AtmosphereBus.pressure
8	AtmosphereBus.speed_sound
9	Gravity_ned(1)
10	Gravity_ned(2)
11	Gravity_ned(3)
12	MagneticField_ned(1)
13	MagneticField_ned(2)
14	MagneticField_ned(3)

(b) *Vettore degli ingressi*

linsys.OutputName

	1
1	Accel_body(1)
2	Accel_body(2)
3	Accel_body(3)
4	DCM_be(1)
5	DCM_be(2)
6	DCM_be(3)
7	DCM_be(4)
8	DCM_be(5)
9	DCM_be(6)
10	DCM_be(7)
11	DCM_be(8)
12	DCM_be(9)
13	Euler(1)
14	Euler(2)
15	Euler(3)
16	LLA(1)
17	LLA(2)
18	LLA(3)
19	Omega_body(1)
20	Omega_body(2)
21	Omega_body(3)
22	V_body(1)
23	V_body(2)
24	V_body(3)
25	V_ned(1)
26	V_ned(2)
27	V_ned(3)
28	X_ned(1)
29	X_ned(2)
30	X_ned(3)
31	dOmega_body(1)
32	dOmega_body(2)
33	dOmega_body(3)

(c) *Vettore di uscita*

Figura A.5: Vettori di stato, di ingresso e di uscita

Appendice B

Codice sviluppato

Di seguito il codice sviluppato all'interno delle MATLAB function.

B.1 HSV Filter

```
function [BW,maskedRGBImage] = createMask(RGB)
        %createMask  Threshold RGB image using auto-generated code from colorThresholder
        %  [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 07-Mar-2021
        %------------------------------------------------------


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

        % Define thresholds for channel 1 based on histogram settings
        channel1Min = 0.940;
        channel1Max = 0.046;

        % Define thresholds for channel 2 based on histogram settings
        channel2Min = 0.361;
        channel2Max = 1.000;

        % Define thresholds for channel 3 based on histogram settings
        channel3Min = 0.524;
        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
```

B.2 Center of gravity

```
function [x_c,y_c,width] = COG(BW)
        [l_x l_y]=size(BW);     %l_x=61, l_y=160
        xc=uint32(0);
        yc=uint32(0);
        n_el=uint32(0);
        width=0;
        a=0;b=0;
        for i=1:l_x
                for j=1:l_y
                        if(i==54 & j>1)
                                if(BW(i,j)==1 & BW(i,j-1)==0)
                                        a=j;
                                else if (BW(i,j)==0 & BW(i,j-1)==1)
                                        b=j;
                                end
                        end
                        width = (b-a);
                end
                if(BW(i,j)~=0)
                        xc = xc + j;
                        yc = yc + i;
                        n_el = n_el + 1;
                        end
                end
        end
        x_c = xc/n_el;
        y_c = yc/n_el;
end
```

B.3 Area of piloting

```
function pilot = fcn(xc, yc, width, angle)
        ll = 50 - width*20/100; %left limit=20% of the line's width from the center
        rl = 50 + width*20/100; %right limit=20% of the line's width from the center
        %% Cruise in a straight line
        if (abs(angle) < 0.055 & (xc > ll & xc < rl))
                pilot = 0;
        %% Turn right
        elseif (angle > 0.055 & xc >= rl & xc < 70)
                pilot = 1;
        %% Turn left
        elseif (angle < -0.055 & xc <= ll & xc > 30)
                pilot = 2;
        %% Stop and turn right
        elseif (xc >= 70 & angle > 0.055)
                pilot = 3;
        %% Stop and turn left
        elseif (xc <= 30 & angle < -0.055)
                pilot = 4;
        %% Reallineate on left
        elseif (abs(angle) < 0.055 & xc <= ll)
                pilot = 5;
        %% Reallineate on right
        elseif (abs(angle) < 0.055 & xc >= rl)
                pilot = 6;
        %% Land
        elseif (xc > ll & xc < rl & yc > 47)
                pilot = -1;
        %% Special cases
        else
                if (xc > rl & (angle < -0.055 & angle >= -0.8))
                        pilot = 1;
                elseif (xc < ll & (angle > 0.055 & angle <= 0.8))
                        pilot = 2;
                elseif (xc > 60 & angle < -0.8)
                        pilot = 3;
                elseif (xc < 40 & angle > 0.8)
                        pilot = 4;
                else
                pilot = 0;
        end
end
```

B.4 Edge and angle detection

```
function [Angolo_rad,BW] = fcn(BW_2)
        BW = edge(BW_2,'Canny');
        [H Theta] = hough(BW);
        P = houghpeaks(H,2);
        Angolo_rad = deg2rad(mean(Theta(P(:,2))));
end
```

Bibliografia

[1] *Algoritmo di Canny*. 2020. URL: https://it.wikipedia.org/wiki/Algoritmo_di_Canny.

[2] *Angoli di Eulero*. 2010. URL: https://it.wikipedia.org/w/index.php?oldid=33218760.

[3] Maik Basso. "A UAV Guidance System Using Crop Row Detection and Line Follower Algorithms". In: *Journal of Intelligent and Robotic Systems* (2019).

[4] Samir Bouabdallah. *Design and Control of quadrotors with application to autonomous flying*. École Polytechnique Fédérale De Lausanne, 2007.

[5] Alexandre Brandao, Felipe N. Martins e Higor B. Soneguetti. "A Vision-based Line Following Strategy for an Autonous UAV". In: *SCITEPRESS (Science and Technology Publications)* (2015).

[6] Roberto Bucher. *Introduzione a Stateflow*. Scuola universitaria professionale della Svizzera italiana, 2013.

[7] *Canny Edge Detection Tutorial*. 2002. URL: http://masters.donntu.org/2010/fknt/chudovskaja/library/article5.htm.

[8] Paolo Ceppi. "Model-based Design of a Line-tracking Algorithm for a Low-cost Mini Drone through Vision-based Control". Politecnico di Torino, 2020.

[9] David Garcia-Olvera et al. "Line follower with a quadcopter". In: *Pädi Boletin Cientifico de Ciencias Basicas e Ingenierias del ICBI* (2020).

[10] Teppo Luukkonen. *Modelling and control of quadcopter*. Aalto University, 2011.

[11] Neetu Sharma Monika Deswal. "A Fast HSV Image Color and Texture Detection and Image Conversion Algorithm". In: *International Journal of Science and Research* (2014).

[12] Matteo Pantalone. "Modellazione e simulazione di un quadricottero multirotore". Alma Mater Studiorum - Università di Bologna, 2015.

[13] *Quadcopter project*. URL: https://www.mathworks.com/help/aeroblks/quadcopter-project.html?s_tid=mwa_osa_a.

[14] Paolo Rosettani. *Manuale Parrot Mambo*. 2020.

[15] Marc Pujol Rubio. "Multi Shot Recording using Consensus-Based Cooperative Control for a Multiquadrotor System". Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, 2019.

[16] Shamik Sural, Gang Qian e Sakti Pramanik. "Segmentation and histogram generation using the HSV color space for image retrieval". In: *IEEE* (2002).

[17] Francesco Tortorella. *La trasformata di Hough*. 2003.