

UNIVERSITÀ DEGLI STUDI DI BERGAMO



Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Triennale

Sviluppo di un algoritmo di line following per quadricottero in ambiente Simulink

Relatore

prof. Antonio Ferramosca

Laureando

Marco Mustacchi

Correlatore:

dott. Marco Polver

Settembre 2021

a mio nonno Vittorio per aver sempre creduto in me
ai miei genitori per avermi aiutato nei momenti più difficili

Indice

Elenco delle figure	vii
Elenco dei listati	ix
1 Introduzione	1
1.1 Applicazioni	1
1.1.1 Militare	2
1.1.2 Spaziale	2
1.1.3 Intrattenimento	2
1.1.4 Trasporto	2
1.1.5 Risposta alle emergenze e aiuti umanitari	2
1.1.6 Soccorso in caso di disastri naturali	3
1.2 Sommario	3
2 Movimento Quadricottero	5
2.1 Analisi qualitativa	5
2.2 Modello Matematico	8
3 Mathworks Minidrone Competition	19
3.1 Regole della competizione	19
3.2 Sistemi controllati in retroazione	21
3.3 Sensori	22
3.3.1 Ultrasound	22
3.3.2 Camera	22
3.3.3 Sensore di pressione	22
3.3.4 IMU	22
3.3.5 Filtro Complementare	24
3.4 Analisi modello Simulink	24
3.5 Simulation Model	26
3.6 Controllore	28
3.6.1 PID	28
3.6.2 PID nel modello Simulink	30

3.6.3	Taratura regolatori PID	34
4	Image Processing	37
4.1	Elaborazione digitale delle immagini	37
4.2	Canny Edge Method	38
4.2.1	Levigazione dell'immagine	39
4.2.2	Image gradient	41
4.2.3	Soppressione dei non massimi	42
4.2.4	Tracciamento dei bordi tramite isteresi	42
4.3	Trasformata di Hough	43
4.4	Image Processing System	44
5	Pianificazione dello spostamento	49
5.1	Path Planning	49
5.2	Stateflow	50
5.3	Assestamento	53
6	Simulazione	57
6.1	Segnale di riferimento	59
6.2	Velocità dei propulsori	60
6.3	Velocità del drone	61
6.4	Confronto tra valori stimati e valori reali	62
6.5	Traiettorie	63
7	Conclusioni	67
A	Valore Parametri	69
B	Codice Grafici	71
C	Tracks Simulazione	75
	Riferimenti bibliografici	77

Elenco delle figure

2.1	Direzioni traslazionali	6
2.2	Direzioni rotazionali	6
2.3	a partire dall'alto: Thrust, Roll, Pitch, Yaw	7
2.4	Drone come corpo rigido e come corpo flessibile	8
2.5	Sistema di riferimento inerziale e del body del drone	9
2.6	Rappresentazione degli assi di riferimento e delle coppie agenti	11
3.1	Path Rules	20
3.2	Controllo in retroazione	21
3.3	Schema di controllo in retroazione di un drone	21
3.4	Schema grafico per Accelerometro e per Giroscopio	24
3.5	Schema di un Filtro Complementare	25
3.6	Modello Simulink	25
3.7	Flight Control System	26
3.8	Modello lineare	27
3.9	Modello non lineare	27
3.10	Control System	28
3.11	Schema Controllore	31
3.12	Controller Yaw	31
3.13	Controller XY to reference orientation	32
3.14	Controller Attitude	32
3.15	Controller Gravity	33
3.16	Blocco Control Mixer	33
3.17	Blocco thrustsToMotorCommands	34
3.18	Modello per taratura PID altitudine	34
3.19	Risposta del sistema linearizzato	35
4.1	Istogramma canali colore RGB	38
4.2	da sinistra: Immagine RGB, Maschera RGB, Maschera binaria	39
4.3	Distribuzione funzione gaussiana	40
4.4	Convoluzione tra Matrice dell'immagine e matrice Kernel	41
4.5	Image Processing System	44

5.1	Path Planning	50
5.2	Sistema di riferimento solidale	51
5.3	Stateflow	52
5.4	Analisi del movimento per Assestamento	54
5.5	Schema Assestamento	55
6.1	Simulation	57
6.2	Simulazione Tracciato 6	58
6.3	Fase di cambiamento della direzione	58
6.4	Fase di fine percorso e atterraggio	59
6.5	Segnale di Yaw e rispettivo riferimento	59
6.6	Velocità rotazione dei propulsori [<i>RPM</i>]	60
6.7	Velocità rispetto a ognuno dei 3 assi	61
6.8	Traiettoria	62
6.9	Distanza	62
6.10	Yaw reale e stimato	62
6.11	Velocità X	63
6.12	Velocità Y	63
6.13	Posizione rispetto a ognuno dei 3 assi	64
6.14	Traiettoria reale dall'alto [m]	65
6.15	Traiettoria reale 3D [m]	65
A.1	Avvio del progetto	69

Elenco dei listati

4.1	Color Threshold	45
4.2	Edge Detection	45
4.3	Find Angle	46
5.1	Critic Condition	51
5.2	Assestamento X	54
5.3	Assestamento Y	55
B.1	Codice grafici simulazioni	71

Capitolo 1

Introduzione

Unmanned aerial vehicles (UAV) are a class of aircrafts that can fly without the onboard presence of pilots. Unmanned aircraft systems consist of the aircraft component, sensor payloads and a ground control station. They can be controlled by onboard electronic equipments or via control equipment from the ground.

(in Wireless Public Safety Networks 1, 2015)

Questa la definizione in letteratura per gli aeromobili a pilotaggio remoto (UAV), chiamati comunemente droni. Come possiamo leggere, la caratteristica di questi dispositivi consiste nell'assenza del pilota e nel controllo remoto, che costituiscono la base della guida a navigazione autonoma, obiettivo cardine di questa tesi, grazie a componenti sensoristici montati su di essi. Prima di addentraci però nello specifico su cosa è stato analizzato per sviluppare questo tipo di tecnica, mostriamo, elencando alcune delle principali applicazioni di utilizzo attuali e future, per quale motivo i droni hanno oggi assunto una sempre maggior importanza con enormi investimenti da parte di stati e multinazionali.

1.1 Applicazioni

Negli ultimi anni, le tecnologie legate allo sviluppo di sistemi volanti autonomi ha subito una impennata rapidissima. In particolare lo sviluppo tecnologico nell'ambito della sensoristica e dell'intelligenza artificiale, ha permesso di equipaggiare i droni con molteplici sensori, come camere nello spettro del visibile e dell'infrarosso (camere termiche), fino ad arrivare a sensori più evoluti come ad esempio sensori Lidar o per il monitoraggio della qualità dell'aria.

1.1.1 Militare

Mentre i droni militari sono stati inizialmente utilizzati più di cinquant'anni fa, in particolare i grandi UAV per sorveglianza e ricognizione, droni più piccoli e portatili sono ora impiegati dalle forze di terra su base regolare. Utilizzati come esche per bersagli, per missioni di combattimento, ricerca e sviluppo e per la supervisione, i droni sono ormai parte integrante delle forze militari di tutto il mondo grazie alla loro agilità, versatilità ed efficienza.

1.1.2 Spaziale

Negli ultimi anni l'esplorazione spaziale che ha interessato per la prima volta anche aziende private, grazie alla crescita esponenziale dell'intelligenza artificiale, ha permesso di rimpiazzare l'invio di astronauti sui corpi solari interessati. Il drone svolge la funzione di esploratore, scattando foto e raccogliendo dati sul terreno del pianeta soprattutto nelle aree che il rover non può raggiungere.

1.1.3 Intrattenimento

Siamo ormai abituati a riprese dall'alto mozzafiato nelle competizioni sportive, ottenute mediante braccia robotiche e droni, ma quest'ultimi possono essere sfruttati anche come intrattenimento stesso, per esempio attraverso spettacoli di luci sincronizzate usate prevalentemente nei parchi divertimento più famosi. I droni possono essere utilizzati anche in ambito videoludico, non soltanto per la creazione di videogiochi, in cui i droni permettono di sviluppare mappe iper-realistiche per videogiochi sempre più complessi, ma anche come componenti di gioco. Si stanno sempre più instaurando infatti competizioni agonistiche a livello internazionale in cui i partecipanti si sfidano in gare a ostacoli attraverso la FPV, o first person view.

1.1.4 Trasporto

I droni vengono utilizzati per attività di logistica di multinazionali, per consegnare le merci dai rivenditori locali e dai centri di adempimento, permettendo trasporti veloci e autonomi. Essi però non vengono utilizzati soltanto per il trasporto tradizionale di prodotti di consumo, ma anche, per esempio, per diffondere fertilizzanti sul terreno, impollinare fiori e per la raccolta e il trasporto di rifiuti, soprattutto dagli oceani.

1.1.5 Risposta alle emergenze e aiuti umanitari

Le squadre di pronto intervento lavorano spesso contro il tempo e in condizioni che riducono notevolmente la loro mobilità sul terreno. In questo scenario, i droni possono essere estremamente utili per aiutare i servizi di emergenza. Identificare

le vittime difficili da individuare a occhio nudo attraverso l’uso di termocamere, consegna veloce di defibrillatori su richiesta, aiutare a valutare la direzione in cui si sta dirigendo l’inondazione, consegnare articoli essenziali come cibo, acqua a chi ne ha bisogno, sono solo alcuni dei potenziali utilizzi dei droni. Essi possono svolgere tutti questi compiti molto più rapidamente delle vie tradizionali, come la terra e l’acqua, spesso difficilmente attraversabili in seguito a disastri, oltre ad avere l’ulteriore vantaggio di tenere il personale aggiuntivo lontano da aree potenzialmente pericolose.

1.1.6 Soccorso in caso di disastri naturali

Oltre alla risposta alle emergenze, i droni si sono dimostrati utili in caso di disastri naturali. In seguito a uragani e terremoti, per esempio, gli UAV sono stati utilizzati per valutare i danni e fornire aiuti e soccorsi. In alcune circostanze inoltre, stanno aiutando a prevenire del tutto i disastri. Per aiutare a monitorare e combattere gli incendi boschivi per esempio, i droni di sorveglianza dotati di termocamere vengono impiegati per rilevare le temperature anomale delle foreste. Così facendo, le squadre sono in grado di identificare le aree più inclini agli incendi boschivi o di identificare degli incendi prima che questi si manifestino. I droni, quindi, riescono ad essere in grado di aumentare drasticamente i tassi di sopravvivenza sia nelle aree rurali che in quelle urbane di tutto il mondo.

1.2 Sommario

Una volta appresa l’importanza attuale e futura di questi dispositivi volanti, possiamo introdurre come è stato deciso di strutturare questa tesi. Nel capitolo 2 analizzeremo prima dal punto di vista qualitativo e poi analitico e matematico, la dinamica di un quadricottero, ovvero di un drone composto da 4 propulsori. Confronteremo poi nel capitolo 3 i risultati ottenuti con il modello Simulink fornito da Mathworks, all’interno del quale è già presente tutta la parte di modellazione matematica dell’ambiente e del drone, chiamato Parrot Mambo Fly. In particolare ci soffermeremo sull’analisi del controllore e del plant all’interno della catena chiusa di retroazione. Cercheremo poi nel capitolo 4 di creare attraverso il modello Simulink fornito un algoritmo in grado di far seguire al drone un percorso di uno specifico colore, obiettivo della Mathworks Minidrone Competition. Introdurremo prima in modo teorico il sistema di acquisizione e elaborazione dell’immagine catturata dal minidrone, in particolare con riferimento ad alcune tecniche di image processing, e poi la soluzione proposta per la competizione. Nel capitolo 5 spiegheremo l’algoritmo in grado di far muovere correttamente il drone lungo il percorso in relazione ai dati derivanti dall’image processing della fase precedente. Faremo nel capitolo 6 una breve analisi dei risultati ottenuti mediante simulazione, confrontandoli con i

risultati attesi derivanti dallo studio teorico e infine nel capitolo 7 trarremo le conclusioni evidenziando il resoconto di questa ricerca di tesi e gli eventuali problemi riscontrati.

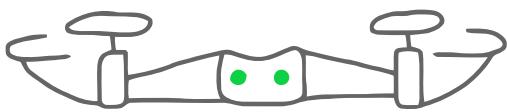
Capitolo 2

Movimento Quadricottero

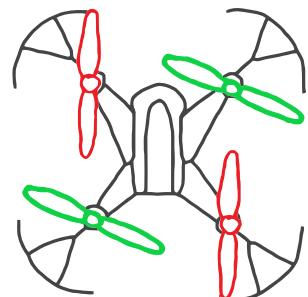
In questo capitolo descriveremo prima in modo qualitativo, e successivamente in modo analitico matematico, il movimento di un quadricottero. L'obiettivo finale sarà trovare un sistema di equazioni in forma di state space che descriva la dinamica del modello.

2.1 Analisi qualitativa

Iniziamo descrivendo in modo qualitativo il movimento di un quadricottero. Innanzitutto viene definito quadricottero poiché composto da quattro rotori, posizionati alle quattro estremità del veicolo che gli permettono il movimento, come mostrato in figura 2.1a e 2.1b. La prima particolarità di questo modello che è un sistema



(a) Drone vista frontale



(b) Drone vista dall'alto

sottoattuato, infatti presenta 6 gradi di libertà, ma soltanto 4 attuatori (i motori). Poiché non abbiamo un attuatore per ogni tipo di movimento, questo comporta che i movimenti traslazionali e i movimenti rotazionali non saranno tutti controllabili in qualunque momento. Per esempio, il drone non è capace di muoversi in avanti, se prima non lo ruotiamo in quella direzione. Per quanto riguarda i rotori,



Figura 2.1: Direzioni traslazionali

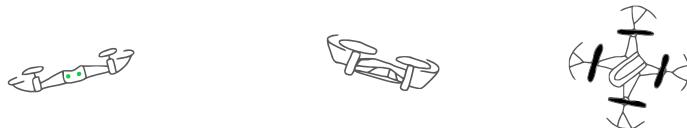


Figura 2.2: Direzioni rotazionali

viene sfruttata una strategia che permette di disaccoppiare i movimenti permessi: i propulsori posizionati simmetricamente rispetto al centro del drone ruotano nella stessa direzione. Il movimento è ottenuto attraverso il principio di azione-reazione: la rotazione delle eliche spinge l'aria verso il basso e questo provoca la reazione di spinta verso l'alto del drone.

Analizziamo adesso come è possibile ottenere le varie tipologie di movimento, Thrust, Roll, Pitch e Yaw, con i soli 4 motori dei propulsori.

- La forza di Thrust consiste in una forza verso l'alto che permette al drone di alzarsi o abbassarsi di quota. Per fare ciò è sufficiente aumentare o diminuire la velocità di rotazione di tutti e 4 i propulsori per, rispettivamente, muovere il drone verso l'alto o verso il basso. Nel caso la forza di spinta coincida con la forza di gravità, il drone si manterrà alla stessa quota.
- Il momento di Roll provoca una rotazione del drone intorno all'asse X. È necessario aumentare o diminuire i due propulsori a sinistra o destra per effettuare una manovra di rollo.
- Il momento di Pitch provoca una rotazione del drone intorno all'asse Y. È necessario aumentare o diminuire i due propulsori davanti o dietro per effettuare una manovra di beccheggio.
- Il momento di Yaw provoca una rotazione del drone intorno all'asse Z. È necessario aumentare o diminuire i due propulsori simmetrici rispetto al centro per effettuare una manovra di imbardata.

In figura 2.3 vengono riassunte le precedenti manovre descritte, in cui la freccia rossa indica la velocità di rotazione dei motori antiorari, mentre quella verde dei motori orari. La freccia blu e gialla invece indicano rispettivamente la forza di Thrust risultante e la direzione della rotazione.

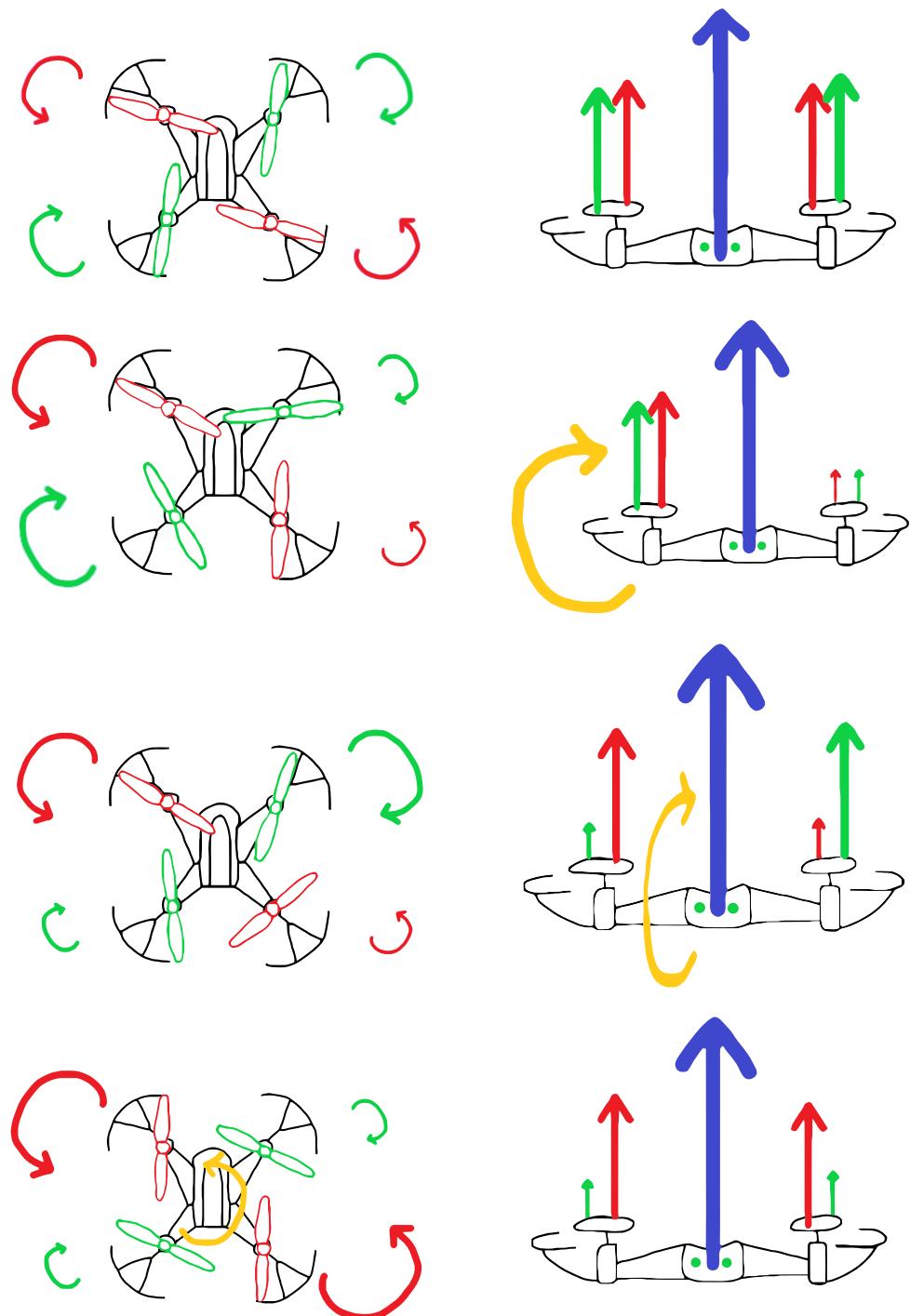


Figura 2.3: a partire dall'alto: Thrust, Roll, Pitch, Yaw

2.2 Modello Matematico

Affrontiamo ora la modellazione matematica della dinamica di un quadricottero. Lo studio di un modello può essere affrontato con due tecniche: *white box* e *black box*. Il primo metodo consiste nel derivare direttamente il modello costituito dalle variabili di stato attraverso una qualunque tecnica tra cui la meccanica newtoniana, la meccanica lagrangiana, la meccanica hamiltoniana, ecc. Per fare ciò dobbiamo capire quali componenti sono importanti da considerare, e quali possono essere tralasciate per non appesantire troppo il modello. Viene definito white perché si conosce il modello a priori, a differenza del black in cui il modello non si conosce o è troppo difficile da derivare direttamente. In questo caso si utilizza la tecnica del system identification: immettendo nel modello alcuni input e mettendoli in relazione con gli output, è possibile fare inferenza per capire la struttura del modello incognito. Esiste infine una tecnica che possiamo definire intermedia tra le due precedenti chiamata *grey box* in cui conosciamo il modello del sistema che vogliamo identificare, ma sono incogniti il valore dei parametri. Questi ultimi vengono identificati sulla base dei dati in nostro possesso. In questa tesi viene scelto di utilizzare il primo metodo e in particolare di sfruttare la meccanica newtoniana. La prima assunzione che dobbiamo fare prima di addentrarci nella modellazione matematica, è di considerare il drone come un corpo rigido. Nella realtà dovremmo considerare il corpo come flessibile, che come possiamo vedere nella figura a 2.4, comporta una distanza (vettore \mathbf{r}) dal centro di massa che dipende dalla flessione. Nel nostro caso, approssimandolo come corpo rigido, consideriamo la distanza di ogni punto del corpo rispetto al centro di massa costante.



Figura 2.4: Drone come corpo rigido e come corpo flessibile

Le variabili di stato sono il più piccolo numero di variabili di sistema che possono rappresentare l'intero stato del sistema in qualunque istante di tempo. Solitamente, il minor numero di variabili di stato necessarie è uguale all'ordine delle equazioni differenziali che definiscono il sistema, ma non è sempre necessariamente così. Se invece il sistema è rappresentato come funzione di trasferimento, il minor numero di variabili di stato necessarie è uguale all'ordine del denominatore della funzione di trasferimento dopo che è stata ridotta a minimi termini. Poiché nel nostro caso il drone è libero di muoversi nello spazio, abbiamo 6 gradi di libertà (6DOF), per cui serve conoscere la posizione 3-D e l'orientamento 3-D del veicolo. Inoltre è interessante capire come questi cambiano durante il tempo. Il modello dinamico che verrà analizzato in questo capitolo è stato ottenuto dallo studio delle fonti [6],

[8], [15], [16]. Le variabili di stato che interessano per il sistema sono:

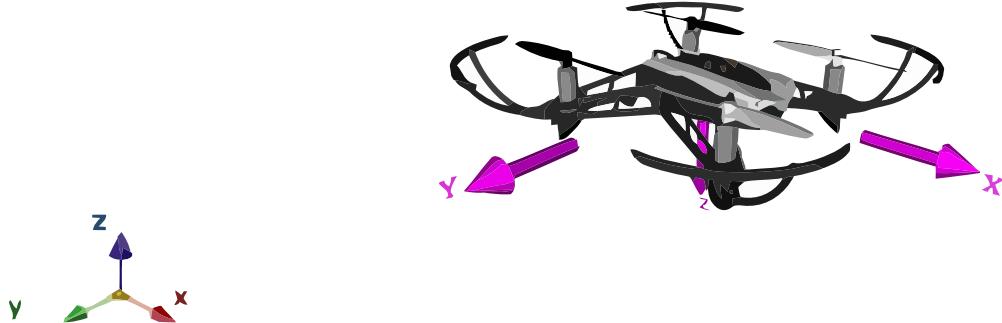


Figura 2.5: Sistema di riferimento inerziale e del body del drone

$$\mathbf{v}^b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b \quad \boldsymbol{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}^b \quad \boldsymbol{\Phi} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^n \quad (2.1)$$

dove \mathbf{v}^b rappresenta la velocità nel sistema di riferimento del body del veicolo, la $\boldsymbol{\omega}$ la velocità angolare nel riferimento del body, la $\boldsymbol{\Phi}$ gli angoli di Eulero e \mathbf{x} la posizione rispetto al sistema di riferimento inerziale. Inoltre per ognuna di esse dovremo calcolare la derivata prima per capire il cambiamento nel tempo. Calcoliamo adesso le forze e i momenti agenti dal punto di vista fisico: la forza dovuta alla spinta del propulsore i -esimo, che sappiamo essere diretta lungo l'asse Z del veicolo, può essere espressa come:

$$F_i = \frac{1}{2} \rho A C_i R^2 w_i^2 \quad (2.2)$$

che dipende oltre che dalla velocità angolare w_i , dalla costante di proporzionalità della coppia Ct , dalla densità del fluido (in questo caso l'aria) ρ , dall'area spazzata dal rotore A e dal raggio dell'elica R . Possiamo notare come, a meno della velocità angolare, siano tutte costanti, per cui possiamo semplificarla come proporzionale al quadrato della velocità angolare dei motori:

$$F_i = b w_i^2 \quad (2.3)$$

dove i indica il numero dell'elica raffigurata e b è una costante di proporzionalità¹. Per quanto riguarda la coppia, essa può essere ottenuta come la somma dovuta alla

¹I valori reali sono riportati in Appendice A insieme ad altri parametri utilizzati nel modello

resistenza fluidodinamica e all'inerzia dei motori, e sappiamo essere diretta lungo l'asse Z :

$$M_i = k\omega_i^2 + I_M \dot{\omega}_i \quad (2.4)$$

dove I_M è il momento di inerzia dei rotori e k un coefficiente di resistenza aerodinamica definito come:

$$k = C_t \rho A_{\text{cross}} R^3 \quad (2.5)$$

dipendente quindi dalla costante adimensionale C_t , dalla densità del fluido (in questo caso l'aria) ρ , e infine dall'area della sezione trasversale delle eliche A_{cross} attraverso il raggio dell'elica R . Tuttavia nella condizione di steady state flight, ossia quando il drone rimane alla stessa quota (che nel nostro caso vedremo sarà la maggior parte del tempo), $\dot{\omega} \approx 0$ poiché deve mantenersi circa costante per contrapporsi alla forza di gravità. Di conseguenza possiamo ignorare il secondo termine e approssimare, ottenendo quindi:

$$M_i = (-1)^{i+1} k \omega_i^2 \quad (2.6)$$

Una volta che abbiamo espresso le forze in funzione dei nostri input del sistema (ciò che possiamo effettivamente controllare) ovvero le velocità di rotazione delle eliche, possiamo esprimere i momenti delle forze rispetto alle forze generate dai 4 rotori. Questo perché vogliamo fare in modo di ricavare le forze dei 4 rotori attraverso i momenti disaccoppiati tra loro e la forza di thrust totale che deve opporsi alla forza di gravità.

Con coerenza rispetto al modello Simulink che vedremo successivamente, rappresentiamo in figura 2.6 le forze agenti e le direzioni degli assi. Quindi calcoliamo le forze di spinta dei propulsori agenti lungo le tre direzioni, date da:

$$\mathbf{F}_{\text{prop}} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -F_1 - F_2 - F_3 - F_4 \end{bmatrix} \quad (2.7)$$

poiché le forze dei propulsori sono solo lungo l'asse Z e dirette verso l'alto, per cui il segno – è dovuto alla convenzione che l'asse Z del sistema di riferimento scelto punta verso il basso. Queste forze corrisponderanno ai nostri input nel modello utilizzato per la simulazione in ambiente Simulink.

Successivamente studiamo i momenti assiali, ovvero i momenti rispetto all'asse X (Roll), all'asse Y (Pitch) e all'asse Z (Yaw). È importante tenere presente la direzione dei momenti delle forze rispetto agli assi, per ricavare i segni corretti. Sempre facendo riferimento alla figura 2.6 e utilizzando la relazione 2.6, i momenti

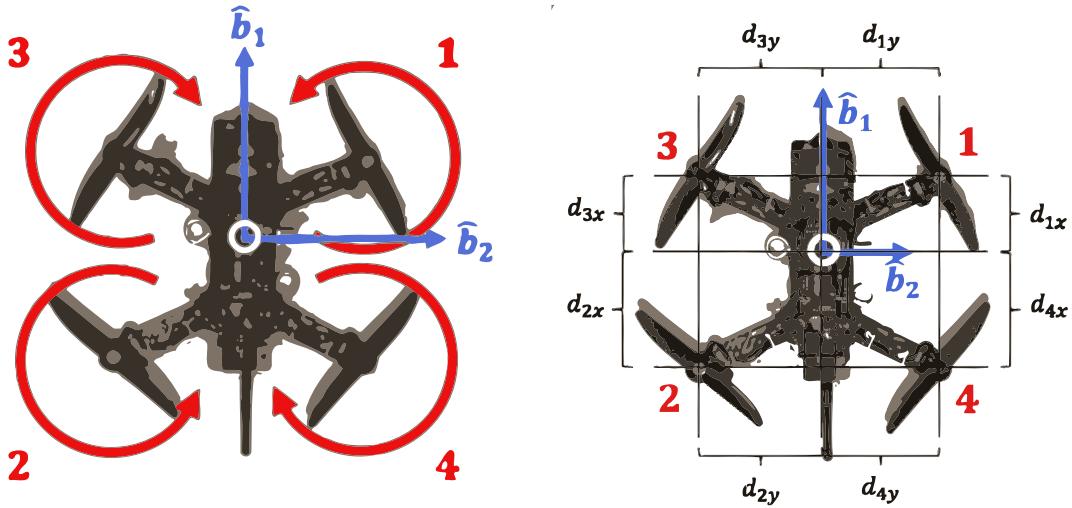


Figura 2.6: Rappresentazione degli assi di riferimento e delle coppie agenti

assiali saranno quindi:

$$\begin{aligned}
 M_\psi &= \sum_{i=1}^4 k\omega_i^2 \\
 &= M_1 - M_2 + M_3 - M_4 \\
 &= (k\omega_1^2) - (k\omega_2^2) + (k\omega_3^2) - (k\omega_4^2) \\
 &= k(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)
 \end{aligned} \tag{2.8}$$

$$\begin{aligned}
 M_\phi &= -F_1d_{1y} + F_2d_{2y} + F_3d_{3y} - F_4d_{4y} \\
 &= d_y(-F_1 + F_2 + F_3 - F_4)
 \end{aligned} \tag{2.9}$$

$$\begin{aligned}
 M_\theta &= F_1d_{1x} - F_2d_{2x} + F_3d_{3x} - F_4d_{4x} \\
 &= d_x(F_1 - F_2 + F_3 - F_4)
 \end{aligned} \tag{2.10}$$

Possiamo quindi esprimere la relazione tra i momenti assiali e le forze di Thrust di ognuno dei quattro propulsori in forma matriciale:

$$\begin{aligned}
 \begin{bmatrix} T_{\text{total}} \\ M_\psi \\ M_\theta \\ M_\phi \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ -k/b & -k/b & k/b & k/b \\ d_x & -d_x & d_x & -d_x \\ -d_y & d_y & d_y & -d_y \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \\
 &= \mathbf{M} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}
 \end{aligned} \tag{2.11}$$

per cui semplicemente invertendo l'equazione precedente, troviamo infine la relazione cercata per facilitare l'attuazione del controllore:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} T_{\text{total}} \\ M_\psi \\ M_\theta \\ M_\phi \end{bmatrix} \quad (2.12)$$

Infine esprimiamo la relazione tra le forze di Thrust appena calcolate e i comandi di ognuno dei 4 motori:

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = -K_M \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \quad (2.13)$$

Possiamo ora cominciare ad analizzare le relazioni tra gli stati del sistema rispetto alle loro derivate, esprimibili con equazioni differenziali. Prima di fare ciò dobbiamo introdurre un metodo per permettere di passare dalle coordinate di un vettore espresse rispetto al sistema di riferimento inerziale a quelle di un vettore rispetto al sistema di riferimento del body del veicolo, e viceversa. Introduciamo quindi le cosiddette matrici di rotazione, in cui gli angoli di ϕ θ e ψ corrispondono ai tre angoli di Eulero:

$$\mathbf{R}_\phi^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \mathbf{R}_\theta^T = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \mathbf{R}_\psi^T = \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Per brevità, nelle successive matrici di rotazione, utilizzeremo l'abbreviazione $s = \sin$ e $c = \cos$. Per la trasformazione diretta tra sistema di riferimento inerziale a sistema di riferimento del body, è sufficiente moltiplicare le tre matrici di rotazione, secondo quanto riportato [15]. Nel nostro caso, poiché passiamo dal sistema di riferimento del body a quello inerziale, dovremo moltiplicare le corrispettive matrici inverse. In questo caso le matrici inverse sono facilmente ottenibili dato che le matrici di rotazione sono ortogonali, per cui la matrice trasposta coincide con la matrice inversa. Quindi moltiplicando otteniamo matrice di rotazione complessiva:

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ -\dot{h}^E \end{bmatrix} = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b \quad (2.14)$$

in cui abbiamo sostituito $z = -h$ dato che nel nostro caso l'asse Z è rivolto verso

il basso. Svolgendo i prodotti riga per colonna:

$$\begin{bmatrix} \dot{x}^E \\ \dot{y}^E \\ -\dot{h}^E \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & -c_\theta s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b \\ = \mathbf{T}_b^n \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b \quad (2.15)$$

in cui la \mathbf{T}_b^n indica che sto passando dalle coordinate del sistema di riferimento del body a quello inerziale terrestre. Sviluppando i prodotti riga per colonna dalla 2.15 otteniamo quindi:

$$\begin{aligned} \dot{x}^E &= c_\theta c_\psi u^b + (-c_\phi s_\psi + s_\phi s_\theta c_\psi) v^b + (s_\phi s_\psi + c_\phi s_\theta c_\psi) w^b \\ \dot{y}^E &= c_\theta s_\psi u^b + (c_\phi c_\psi + s_\phi s_\theta s_\psi) v^b + (-s_\phi c_\psi + c_\phi s_\theta s_\psi) w^b \\ \dot{h}^E &= s_\theta u^b - s_\phi c_\theta v^b - c_\phi c_\theta w^b \end{aligned} \quad (2.16)$$

La seconda relazione esprime i ratei di variazione degli angoli di Eulero rispetto alle componenti del vettore velocità di rotazione del drone nel sistema di riferimento del body. Facendo riferimento a [6], questa relazione è esprimibile come:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} \\ + \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \quad (2.17)$$

che può essere rappresentata, alla stesso modo di 2.15, in forma compatta come:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (2.18)$$

quindi da 2.17 e svolgendo i prodotti riga per colonna otteniamo:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} -\dot{\psi} s_\theta \\ \dot{\psi} c_\theta s_\phi \\ \dot{\psi} c_\phi c_\theta \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\theta} c_\phi \\ -\dot{\theta} s_\phi \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \quad (2.19)$$

che possiamo esprimere come:

$$\begin{aligned} p &= \dot{\phi} - \dot{\psi} \sin \theta \\ q &= \dot{\theta} \cos \phi + \dot{\psi} \cos \theta \sin \phi \\ r &= \dot{\psi} \cos \phi \cos \theta - \dot{\theta} \sin \phi \end{aligned} \quad (2.20)$$

espressione che possiamo esprimere anche per le derivate degli angoli di Eulero

$$\begin{aligned}\dot{\phi} &= p + (q \sin \phi + r \cos \phi) \tan \theta \\ \dot{\theta} &= q \cos \phi - r \sin \phi \\ \dot{\psi} &= (q \sin \phi + r \cos \phi) \sec \theta\end{aligned}\quad (2.21)$$

Le successive relazioni possono essere ricavate applicando la prima e la seconda equazione cardinale della dinamica. La prima equazione cardinale della dinamica descrive il moto traslatorio di un sistema e può essere espressa come:

$$\mathbf{F}^{(e)} = \frac{d\mathbf{P}}{dt} \quad (2.22)$$

dove $\mathbf{F}^{(e)} = \sum_{i=1}^n \mathbf{F}_i^{(e)}$ rappresenta la somma delle forze esterne agenti sul sistema. In questo caso possiamo considerare la massa costante, quindi $\frac{d\mathbf{P}}{dt} = m\mathbf{a}$ e la prima equazione cardinale della dinamica può essere riscritta come:

$$\mathbf{F}^{(e)} = m\mathbf{a} \quad (2.23)$$

È necessario sottolineare che questa relazione vale solo se fatta rispetto al sistema di riferimento inerziale (non è possibile farla rispetto a quello non inerziale, in cui le leggi di inerzia non valgono). Tuttavia possiamo scegliere se svolgerla con le coordinate del sistema di riferimento inerziale terrestre, oppure come nel nostro caso, con le coordinate del sistema di riferimento del body. Esplicitando adesso le forze agenti sul sistema:

$$\mathbf{F}_{\text{prop}} + \mathbf{F}_{\text{grav}} + \mathbf{F}_{\text{drag}} = m\ddot{\mathbf{x}}_B \quad (2.24)$$

dove \mathbf{F}_{prop} è la spinta dei propulsori, \mathbf{F}_{grav} è la forza peso e \mathbf{F}_{drag} è la resistenza fluidodinamica. In particolare:

$$\mathbf{F}_{\text{grav}}^n = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \quad (2.25)$$

è definita rispetto al sdr inerziale, di conseguenza possiamo usare l'inversa della matrice di rotazione 2.15, che permette di passare dal sistema di riferimento inerziale a quello espresso tramite le coordinate del body:

$$\mathbf{F}_{\text{grav}}^b = \mathbf{T}_n^b \mathbf{F}_{\text{grav}}^n \quad (2.26)$$

che esplicitando:

$$\begin{aligned}\mathbf{F}_{\text{grav}}^b &= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ -c_\theta s_\psi + s_\phi s_\theta c_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & s_\phi c_\theta \\ s_\phi s_\psi + c_\phi s_\theta c_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \\ &= \begin{bmatrix} -mg \sin(\theta) \\ mg \sin(\phi) \cos(\theta) \\ mg \cos(\phi) \cos(\theta) \end{bmatrix}\end{aligned}\quad (2.27)$$

La resistenza fluidodinamica è quella forza che si oppone al movimento di un corpo in un fluido (nel nostro caso tra il corpo e l'aria). Può essere espressa al solito come un certo coefficiente moltiplicato per la velocità.

$$F_{\text{drag}} = -k_d \dot{\mathbf{x}} \quad (2.28)$$

dove:

$$k_d = C_d \frac{\rho \mathbf{v}^2}{2} A \quad (2.29)$$

questo coefficiente dipende quindi dalla densità dell'aria ρ , dal quadrato della velocità \mathbf{v}^2 , dalla dimensione del corpo A e dal coefficiente C_d , che solitamente è determinato sperimentalmente, e che tiene conto della viscosità e dalla compressibilità dell'aria, della forma del corpo e dell'inclinazione del corpo rispetto al flusso del fluido. Possiamo riportare il valore di questa forza in forma matriciale lungo le tre direzioni:

$$F_{\text{drag}} = \begin{bmatrix} -k_d \dot{x} \\ -k_d \dot{y} \\ -k_d \dot{z} \end{bmatrix} \quad (2.30)$$

Infine esprimiamo l'accelerazione come derivata del vettore velocità, quindi:

$$\ddot{\mathbf{x}}^b = \dot{\mathbf{v}}_{\text{inertial}}^b \quad (2.31)$$

dove $\mathbf{v}_{\text{inertial}}^b$ è la nostra variabile di stato velocità definita in 2.1 rispetto al sistema di riferimento inerziale ma con le coordinate del body. Per trovare l'accelerazione basterà derivare la velocità, ma poiché è esplicitata rispetto al body, dovremo derivare anche i versori che dipendono anch'essi dal tempo. Applichiamo quindi il teorema di Coriolis:

$$\begin{aligned} \dot{\mathbf{v}}_{\text{inertial}}^b &= \left(\frac{d\mathbf{v}^b}{dt} \right)_{\text{inertial}} \\ &= \left(\frac{du}{dt} \hat{b}_1 + \frac{dv}{dt} \hat{b}_2 + \frac{dw}{dt} \hat{b}_3 \right) + \left(u \frac{d\hat{b}_1}{dt} + v \frac{d\hat{b}_2}{dt} + w \frac{d\hat{b}_3}{dt} \right) \\ &= (\dot{u}\hat{b}_1 + \dot{v}\hat{b}_2 + \dot{w}\hat{b}_3) + (u\mathbf{w} \times \hat{b}_1 + v\mathbf{w} \times \hat{b}_2 + w\mathbf{w} \times \hat{b}_3) \\ &= (\dot{u}\hat{b}_1 + \dot{v}\hat{b}_2 + \dot{w}\hat{b}_3) + \mathbf{w} \times (\hat{u}\hat{b}_1 + \hat{v}\hat{b}_2 + \hat{w}\hat{b}_3) \\ &= \dot{\mathbf{v}}_n^b + \boldsymbol{\omega}_n^b \times \mathbf{v}^b \end{aligned} \quad (2.32)$$

Esso indica che dobbiamo considerare anche l'accelerazione dovuta alla rotazione del sistema di riferimento mobile del body frame rispetto al sistema di riferimento fisso. Il prodotto vettoriale possiamo riscrivere sotto forma di matrice simmetrica

obliqua:

$$\begin{aligned}\dot{\mathbf{v}}_{\text{inertial}}^b &= \dot{\mathbf{v}}^b + \boldsymbol{\omega}_n^b \times \mathbf{v}^b \\ &= \dot{\mathbf{v}}^b + [\boldsymbol{\omega} \times] \mathbf{v}^b \\ &= \dot{\mathbf{v}}^b + \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \mathbf{v}^b\end{aligned}\quad (2.33)$$

e infine se sostituiamo $\mathbf{v}^b = [u, v, w]^T$ e $\boldsymbol{\omega}^b = [p, q, r]^T$, otteniamo:

$$\begin{aligned}\dot{\mathbf{v}}_{\text{inertial}}^b &= \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}^b \\ &= \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix}\end{aligned}\quad (2.34)$$

Sostituendo tutti i valori trovati nell'equazione 2.24:

$$\begin{aligned}\mathbf{F}_{\text{prop}} + \mathbf{F}_{\text{grav}} + \mathbf{F}_{\text{drag}} &= m \dot{\mathbf{v}}_{\text{inertial}}^b \\ \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} -F_{\text{drag},x_B} \\ -F_{\text{drag},y_B} \\ -F_{\text{drag},z_B} \end{bmatrix} &= m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \\ \begin{bmatrix} -mg \sin(\theta) \\ mg \sin(\phi) \cos(\theta) \\ -F_1 - F_2 - F_3 - F_4 + mg \cos(\phi) \cos(\theta) \end{bmatrix} &= m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix}\end{aligned}\quad (2.35)$$

e riorganizzando otteniamo:

$$\begin{aligned}\dot{u} &= -g \sin(\theta) + rv - qw - F_{\text{drag},x_B} \\ \dot{v} &= g \sin(\phi) \cos(\theta) - ru + pw - F_{\text{drag},y_B} \\ \dot{w} &= \frac{1}{m} (-F_z) + g \cos(\phi) \cos(\theta) + qu - pv - F_{\text{drag},z_B}\end{aligned}\quad (2.36)$$

Possiamo applicare adesso la seconda equazione cardinale della dinamica, che descrive il moto rotatorio di un sistema e che può essere espressa come:

$$\mathbf{M}^{(e)} = \frac{d\mathbf{L}}{dt} \quad (2.37)$$

dove $\mathbf{M}^{(e)} = \sum_{i=1}^n \mathbf{M}_i^{(e)}$ rappresenta la somma dei momenti esterni agenti sul sistema. Anche in questo caso possiamo considerare la massa costante, quindi $\frac{d\mathbf{L}}{dt} = \mathbf{I} \frac{d\boldsymbol{\omega}}{dt}$ e la seconda equazione cardinale della dinamica può essere riscritta come:

$$\mathbf{M}^{(e)} = \mathbf{I} \frac{d\boldsymbol{\omega}}{dt} \quad (2.38)$$

dove \mathbf{I} è il momento d'inerzia, definito come:

$$\mathbf{I} = \int_V \rho r^2 \, dV \quad (2.39)$$

dove ρ è la densità del corpo. Esso può essere riscritto come matrice d'inerzia:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \quad (2.40)$$

e poiché possiamo considerare il drone simmetrico rispetto a tutti e tre gli assi X , Y e Z , la matrice può essere semplificata come:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (2.41)$$

ovvero si mantengono solo i momenti d'inerzia lungo la diagonale principale. Come prima, poiché la velocità angolare è espressa con le coordinate del sistema di riferimento del body, per la derivazione possiamo applicare il teorema di Coriolis e il teorema di Poisson e ottenere la seconda equazione cardinale come:

$$\begin{aligned} \mathbf{M} &= \mathbf{I}^b \dot{\boldsymbol{\omega}}_n^b + \boldsymbol{\omega}_n^b \times \mathbf{I}^b \boldsymbol{\omega}_n^b \\ &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \end{aligned} \quad (2.42)$$

esplicitando il vettore \mathbf{M} e svolgendo prodotti matriciali otteniamo:

$$\begin{aligned} \begin{bmatrix} L \\ M \\ N \end{bmatrix} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \\ &= \begin{bmatrix} I_{xx}\dot{p} \\ I_{yy}\dot{q} \\ I_{zz}\dot{r} \end{bmatrix} + \begin{bmatrix} -I_{yy}qr + I_{zz}qr \\ I_{xx}pr - I_{zz}pr \\ -I_{xx}pq + I_{yy}pq \end{bmatrix} \end{aligned} \quad (2.43)$$

e infine riorganizzando otteniamo:

$$\begin{aligned} \dot{p} &= \frac{1}{I_{xx}} (L + (I_{yy} - I_{zz}) qr) \\ \dot{q} &= \frac{1}{I_{yy}} (M + (I_{zz} - I_{xx}) pr) \\ \dot{r} &= \frac{1}{I_{zz}} (N + (I_{xx} - I_{yy}) pq) \end{aligned} \quad (2.44)$$

Infine mettiamo insieme i risultati trovati nelle equazioni 2.16, 2.21, 2.36, 2.44, e otteniamo un sistema di equazioni che rappresenta la dinamica del drone sia rispetto

alle coordinate del sistema di riferimento inerziale, sia a quello del body:

$$\begin{aligned}
 \dot{u} &= -g \sin(\theta) + rv - qw - F_{dragx_B} \\
 \dot{v} &= g \sin(\phi) \cos(\theta) - ru + pw - F_{dragy_B} \\
 \dot{w} &= \frac{1}{m} (-F_z) + g \cos(\phi) \cos(\theta) + qu - pv - F_{dragz_B} \\
 \dot{p} &= \frac{1}{I_{xx}} (L + (I_{yy} - I_{zz}) qr) \\
 \dot{q} &= \frac{1}{I_{yy}} (M + (I_{zz} - I_{xx}) pr) \\
 \dot{r} &= \frac{1}{I_{zz}} (N + (I_{xx} - I_{yy}) pq) \\
 \dot{\phi} &= p + (q \sin \phi + r \cos \phi) \tan \theta \\
 \dot{\theta} &= q \cos \phi - r \sin \phi \\
 \dot{\psi} &= (q \sin \phi + r \cos \phi) \sec \theta \\
 \dot{x}^E &= c_\theta c_\psi u^b + (-c_\phi s_\psi + s_\phi s_\theta c_\psi) v^b + (s_\phi s_\psi + c_\phi s_\theta c_\psi) w^b \\
 \dot{y}^E &= c_\theta s_\psi u^b + (c_\phi c_\psi + s_\phi s_\theta s_\psi) v^b + (-s_\phi c_\psi + c_\phi s_\theta s_\psi) w^b \\
 \dot{h}^E &= -1 \left(-s_\theta u^b + s_\phi c_\theta v^b + c_\phi c_\theta w^b \right)
 \end{aligned} \tag{2.45}$$

Il risultato è una modellazione non lineare della dinamica del drone che potrà essere usata, come vedremo, per definire il modello in ambiente Simulink per la simulazione.

Il sistema di equazioni che descrive la dinamica del sistema è di tipo non lineare, non rappresentabile quindi in forma matriciale. Esso sarà nella forma di state space del tipo:

$$\begin{aligned}
 \dot{\mathbf{x}}(t) &= \mathbf{f}(t, x(t), u(t)) \\
 \mathbf{y}(t) &= \mathbf{h}(t, x(t), u(t))
 \end{aligned}$$

Di conseguenza, a questo punto mancherà definire le nostre uscite $\mathbf{y}(t)$ corrispondenti alle variabili misurate.

Possiamo inoltre linearizzare il sistema nell'intorno di un punto di equilibrio per renderlo lineare e rappresentarlo in forma matriciale di state space più nota:

$$\begin{aligned}
 \dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \\
 \mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)
 \end{aligned}$$

Il modello linearizzato, anch'esso presente, come vedremo, nel progetto Simulink, consentirà di utilizzare tutte le strategie note nell'ambito del controllo per la realizzazione del regolatore.

Capitolo 3

Mathworks Minidrone Competition

Mathworks, una multinazionale specializzata nella produzione di software per calcoli matematici, organizza una competizione studentesca con l'obiettivo di creare un algoritmo di line following per permettere a un quadricottero di seguire un percorso di uno specifico colore e di atterrare all'interno di un cerchio finale, dello stesso colore. È fornito inizialmente da parte di Mathworks un modello Simulink del sistema, necessario per simulare il comportamento del drone, e che dovrà essere modificato opportunamente secondo lo scopo della competizione. In seguito, nel Round 2, il flight control system verrà convertito in codice C++ e testato direttamente sull'hardware di un mini-drone, chiamato Parrot Mambo Fly, durante un evento dal vivo. Questo tipo di drone infatti possiede un microcontrollore in grado di poter eseguire real time il controllore, una volta automaticamente linearizzato da Simulink, inserendo il codice tramite una semplice chiavetta USB.

3.1 Regole della competizione

L'obiettivo della competizione è quello di studiare e progettare un algoritmo per seguire autonomamente un percorso di un particolare colore all'interno di un'area quadrata chiamata arena e atterrare all'interno del cerchio finale. Per la valutazione del risultato elaborato saranno prese in considerazione la precisione nel seguire la pista, se l'atterraggio avviene all'interno di un marcitore circolare e il tempo compiuto per completare il tracciato. Di seguito le regole della competizione:

- L'arena è uno spazio di 4 m * 4 m chiuso da reti su tutti i lati.
- La pista dell'arena è larga 10 cm.
- Il marcitore circolare di atterraggio ha un diametro di 20 cm.

- La pista del line follower consiste solo in segmenti di linea collegati e non ha alcuna curva liscia alle connessioni.
- L'angolo tra due sezioni di pista può avere un valore compreso tra 10 gradi e 350 gradi.
- Il binario può avere da 1 a 10 segmenti di linea collegati. La posizione iniziale del drone è sempre all'inizio della linea. Tuttavia, la parte frontale del drone potrebbe non essere sempre rivolta verso la direzione della prima linea del tracciato.
- La distanza dalla fine della traccia al centro del cerchio è di 25 cm.
- Lo sfondo intorno al percorso può non essere di un solo colore e ha una texture.
- Il colore e la pista per la competizione di persona nel round 2 saranno resi noti il giorno della competizione.
- La pista per il turno di pratica e il turno dal vivo può essere diversa nel caso della competizione di persona nel round 2.

In figura 3.1 possiamo vedere riassunte alcune di queste regole, nella rappresentazione della fine di un generico percorso all'interno dell'arena quadrata.

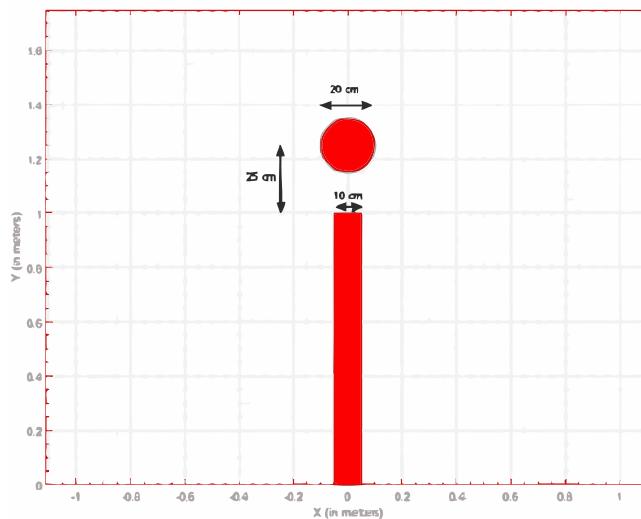


Figura 3.1: Path Rules

3.2 Sistemi controllati in retroazione

Prima di analizzare il modello fornito da Mathworks, ricordiamo innanzitutto il classico controllo industriale retroazionato. Esso, rappresentato in figura 3.2 è composto nel caso più semplice possibile, da tre blocchi, chiamati Controller, Plant e Sensors. I primi, rispettivamente il controllore e il sistema su cui andiamo ad agire costituiscono la cosiddetta linea d'andata d'anello. Con l'aggiunta dei sensori che prendono in ingresso l'uscita del sistema, si forma il cosiddetto schema ad anello chiuso. Nel nostro specifico caso in cui il plant è costituito dal drone che dovremo

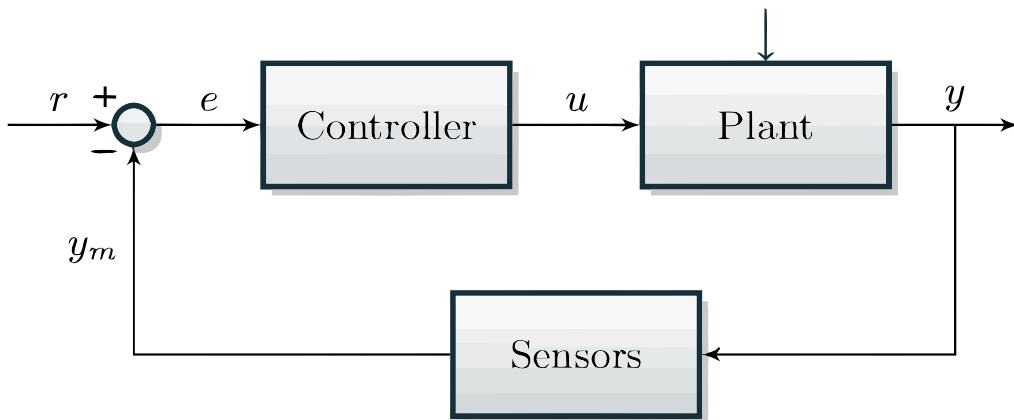


Figura 3.2: Controllo in retroazione

controllare, l'anello potrà essere rappresentato come in figura 3.3, in cui al posto dei sensori sostituiamo appunto i sensori presenti all'interno del drone e descritti in seguito, mentre in uscita dal controllore gli attuatori saranno i 4 motori che possono essere controllati in velocità.

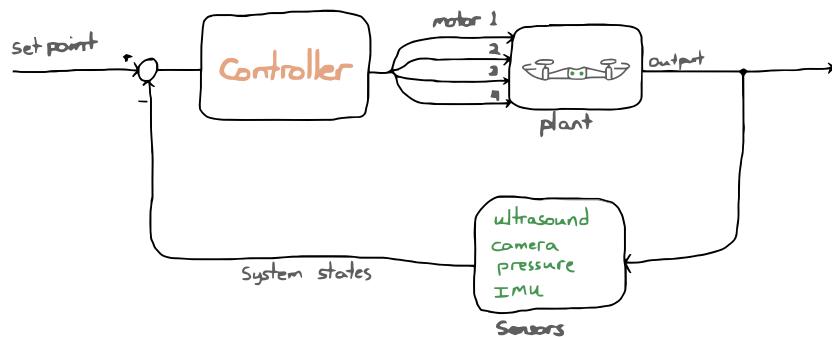


Figura 3.3: Schema di controllo in retroazione di un drone

3.3 Sensori

Il quadricottero della competizione è composto da quattro tipi di sensori:

- Ultrasound
- Camera
- Sensore di pressione
- IMU

3.3.1 Ultrasound

Il primo tipo di sensore è un Sonar, che attraverso l'emissione di ultrasuoni permette di avere una misura della distanza del drone dal terreno, semplicemente calcolando quanto tempo ci vuole affinché il segnale riflesso venga nuovamente percepito dal sensore. Il valore massimo di funzionamento di questo dispositivo è di 13 piedi, al di sopra infatti il suono riflesso dal terreno è troppo debole per poter essere acquisito dal drone.

3.3.2 Camera

La camera, posizionata al di sotto del drone, permette di acquisire immagini in formato Y1UY2V, con 4 array di dimensione 9600 di tipo `uint8`. Attraverso queste immagini acquisite, è possibile sfruttare tecniche di image processing che nel nostro caso, come vedremo, permetteranno di identificare il percorso identificato da un colore, individuare i suoi bordi e calcolare l'angolo durante i cambiamenti di direzione.

3.3.3 Sensore di pressione

Poiché con l'aumentare dell'altitudine la pressione atmosferica diminuisce, grazie a questo sensore otteniamo una misura indiretta dell'altitudine dal terreno.

3.3.4 IMU

Il sensore più importante è il cosiddetto IMU. L'IMU (Inertial measurement unit) è un dispositivo composto da un'unione di diverse tipologie di sensore: accelerometro a 3 assi e da un giroscopio a tre assi, formando così il cosiddetto IMU a 6 assi, che si distingue dall'IMU a 9 assi composto anche da un magnetometro. Questo dispositivo permette di avere una stima dell'orientamento nello spazio dei velivolo.

L'accelerometro, in questo caso di tipo MEMS, si basa sul fatto che qualsiasi corpo, sulla superficie terrestre, risente della accelerazione di gravità rivolta sempre

verso il basso. Quando il corpo ruota, misurando l'accelerazione lungo gli assi di riferimento del veicolo e ipotizzando ancora che agisca solo la forza di gravità, è possibile risalire all'inclinazione del corpo calcolando come si distribuisce il vettore accelerazione di gravità lungo gli assi secondo delle semplici formule trigonometriche, che nel caso bidimensionale e facendo riferimento alla figura 3.4:

$$\vartheta = \arctan\left(\frac{A_x}{A_z}\right)$$

Nel nostro caso tridimensionale, i tre angoli sono ottenuti come:

$$\begin{aligned}\vartheta &= \arctan\left(\frac{A_x}{\sqrt{A_y^2+A_z^2}}\right) \\ \varphi &= \arctan\left(\frac{A_y}{\sqrt{A_x^2+A_z^2}}\right) \\ \Phi &= \arctan\left(\frac{\sqrt{A_x^2+A_y^2}}{A_z}\right)\end{aligned}$$

dove l'angolo Φ è l'angolo formato dall'asse Z e non l'angolo di yaw, poiché quest'ultimo non è ricavabile direttamente con un accelerometro. Nell'accelerometro di tipo MEMS, in particolare, la misura di come si distribuisce la forza di gravità è ottenuta mediante lo spostamento di una massa movibile causato dall'accelerazione, che comporta una variazione delle capacità dei condensatori all'interno del dispositivo.

Il secondo sensore, chiamato giroscopio, facendo riferimento alla figura 3.4, effettua una stima della posizione angolare attorno ai principali assi integrando la velocità angolare misurata:

$$\theta = \int \omega_y dt$$

Il motivo per cui non si utilizza la misura di un singolo sensore è dovuto agli errori di ognuno degli strumenti. L'accelerometro funziona bene quando abbiamo solo il vettore accelerazione di gravità, ovvero in condizioni statiche e semi-statiche, quindi con frequenze di variazione dell'accelerazione basse. Al contrario un giroscopio ha un errore intrinseco durante l'integrazione, integrando anche i random noise e l'offset, che altrimenti sarebbero trascurabili; di conseguenza avremo una misura che diverge nel tempo. Questo offset è generalmente in bassa frequenza, quindi un giroscopio funziona bene in alta frequenza dove l'errore dato dal rumore elettronico ha tendenzialmente media nulla. Per risolvere queste problematiche si utilizza generalmente un filtraggio detto in frequenza, in cui si vanno a prendere solamente le componenti utili in frequenza dei nostri sensori per ottenere una misura robusta delle stime. Il filtraggio di questo tipo più comune è il filtro di Kalman, ma in questo caso in cui abbiamo solo l'accelerometro e il giroscopio, possiamo utilizzare un particolare filtro di Kalman di regime del primo ordine, chiamato filtro complementare.

Infine il motivo per cui generalmente si utilizza anche un magnetometro è dovuto al fatto che l'accelerometro ha problemi nella misurazione dell'angolo di Yaw, ovvero della rotazione lungo l'asse Z.

Il magnetometro specifica rispetto al nord magnetico come siamo posizionati, e quindi fornisce un valore di aggiornamento al giroscopio nel caso di utilizzo di un filtro di Kalman.



Figura 3.4: Schema grafico per Accelerometro e per Giroscopio

3.3.5 Filtro Complementare

Il filtro complementare esegue il filtraggio di un segnale utilizzando la tecnica del sensor fusion, ovvero del processo di combinare dati provenienti da più sensori per ottenere una stima che ha minor incertezza rispetto alle singole misure. In particolare, il filtro complementare è un utile filtro per l'integrazione della misura di soli due sensori. Viene definito complementare perché utilizza appunto due filtri tra loro complementari, generalmente un filtro passa alto e un filtro passa basso. Nel nostro caso si utilizza un filtro passa basso per l'accelerometro in modo da attenuare le componenti in alta frequenza, maggiormente soggette ad errori, e lasciar passare inalterate le componenti in bassa frequenza; mentre si utilizza un filtro passa alto per il giroscopio, che avrà comportamento opposto, come si può notare in figura 3.5. Un filtro complementare può essere descritto dalla seguente formula:

$$\vartheta = \vartheta_{\text{accel}} \mathbf{K} + \vartheta_{\text{gyro}} (\mathbf{1} - \mathbf{K})$$

in cui la scelta del parametro $K \in [0,1]$ influenza la banda passante dei filtri. La regolazione, infatti, viene fatta semplicemente modificando le frequenze di taglio dei due filtri, che chiaramente dovranno essere medesime, in modo da dare maggior importanza all'una o all'altra misurazione del parametro d'interesse.

3.4 Analisi modello Simulink

Analizziamo adesso il modello Simulink fornito da Mathworks. Esso si presenta come un normale sistema retroazionato come mostrato in figura 3.6, a cui però viene

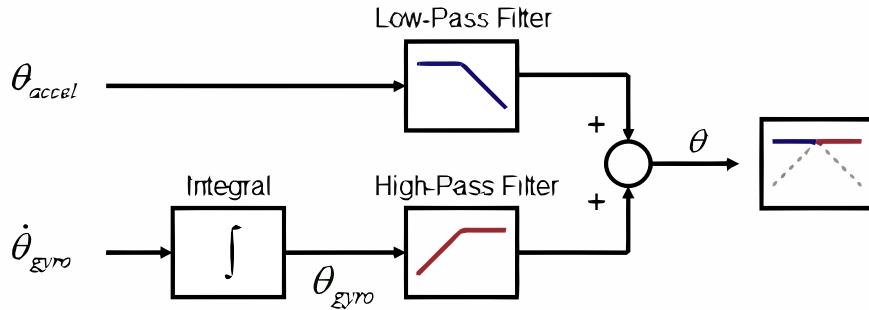


Figura 3.5: Schema di un Filtro Complementare

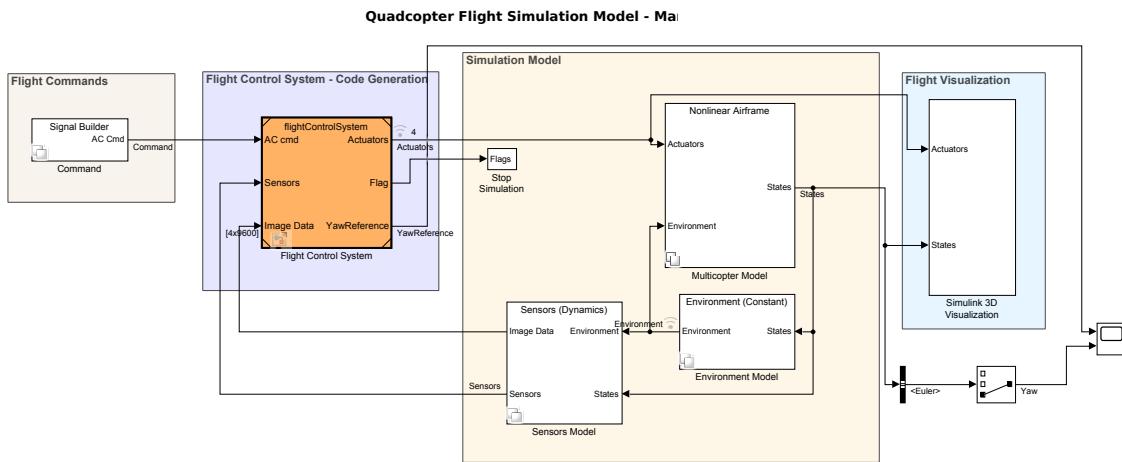


Figura 3.6: Modello Simulink

aggiunto un blocco per la visualizzazione della simulazione, chiamato **Simulink 3D Visualization**. Il sistema **Signal Builder** genera periodicamente il segnale di riferimento del sistema che potremo modificare successivamente per specificare nuovi set point.

Il sistema **Flight Control System**, che è ciò che effettivamente andremo a inserire direttamente nel drone, è costituito, come mostrato in figura 3.7 da due parti: l'**Image Processing System** e il **Control System**. Poiché i due sistemi lavorano a un differente sample time, essi sono connessi attraverso un blocco **Rate Transition**, che permette di trasferire correttamente i dati tra i due sistemi.

Il blocco chiamato **Simulation Model** rappresenta il plant, ossia la modellazione del sistema, ed è composto dal modello lineare e non lineare del minidrone, dal modello dell'ambiente e dal modello dei sensori all'interno del drone.

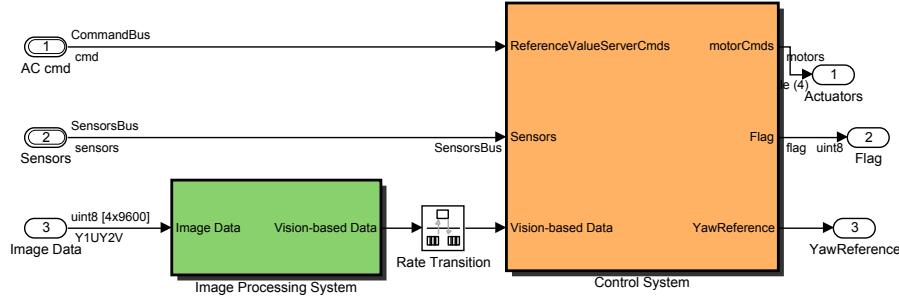


Figura 3.7: Flight Control System

3.5 Simulation Model

All'interno dell'immenso schema del modello di simulazione, contenente i modelli matematici della dinamica del quadricottero, dei sensori e dell'ambiente, soffermiamo la nostra analisi sul modello utilizzato per descrivere il comportamento del drone. All'interno del blocco **NonlinearAirframe** è possibile scegliere se usare la modellizzazione matematica lineare oppure non lineare.

La prima è rappresentata in figura 3.8 sotto forma di spazio di stato. Essa verrà utilizzata per definire il controllore, poiché molte delle tecniche utilizzate (tra cui il controllore PID implementato) prevedono di utilizzare un modello lineare.

La seconda, invece, approssimando meglio il modello alla realtà, verrà utilizzata per la simulazione. Il modello non lineare rappresentato in figura 3.9 è costituito da due blocchi principali: l'**AC model** e il **6DOF**. Tutto ciò che può creare una forza o un momento al drone è calcolato nel blocco **AC model**, il quale quindi riceve in ingresso non solo le velocità degli attuatori derivanti dal controllore, ma anche i disturbi dovuti all'ambiente, come la gravità, la temperatura e la densità dell'aria, la pressione atmosferica, la velocità del suono e il campo magnetico¹. L'**AC model** inoltre acquisisce anche le attuali velocità lineari e angolari con coordinate del body, in retroazione dal blocco successivo **6DOF** insieme alla DCM, ovvero la matrice di rotazione. Il blocco **6DOF**, ovvero 6 gradi di libertà, permette, a partire dalla forza e dal momento del drone, di ricavare le equazioni che descrivono la dinamica del sistema, come abbiamo mostrato precedentemente nel capitolo 2. Ciò che restituirà infatti saranno le derivate delle variabili di stato (ognuna delle quali descritta da un vettore 3×1) sia rispetto al riferimento inerziale della Terra, sia rispetto al riferimento solidale al drone Body, che descrivono come gli stati del sistema variano nel tempo. Infine questi valori calcolati vengono raggruppati tutti in un bus comune

¹I valori reali sono riportati in Appendice A insieme ad altri parametri utilizzati nel modello

chiamato *States*. Inoltre questo blocco permette non solo di impostare alcuni parametri costanti come le masse o i momenti d'inerzia, ma anche scegliere se descrivere l'orientamento del minidrone sia attraverso gli angoli di Eulero, come abbiamo fatto noi in precedenza studiando la modellizzazione matematica della dinamica, sia attraverso i quaternioni, che, come i precedenti, forniscono una rappresentazione di orientamenti e rotazioni di oggetti in tre dimensioni. Tuttavia mentre i primi forniscono una rappresentazione a tre parametri, i secondi, più complessi, rappresentano l'orientamento tramite una quadrupla e permettono di superare alcune limitazioni come il problema del Gimbal Lock.

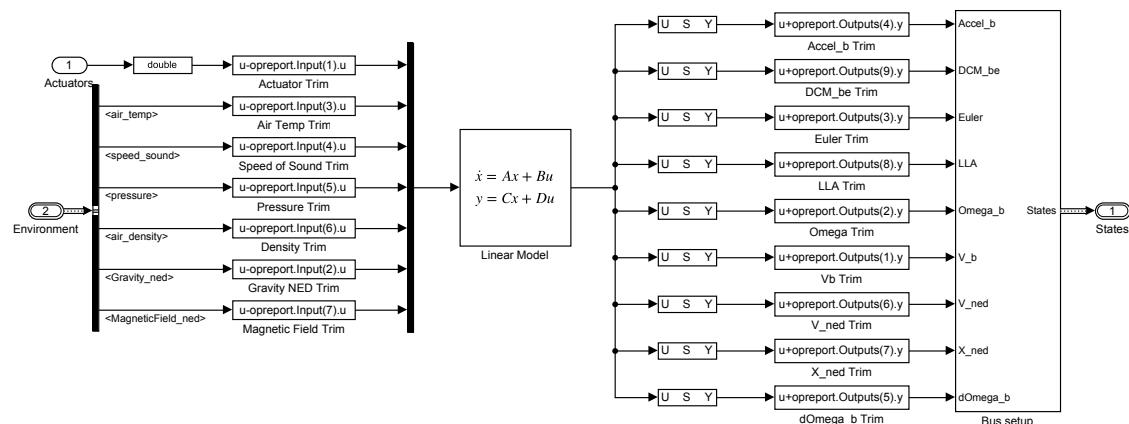


Figura 3.8: Modello lineare

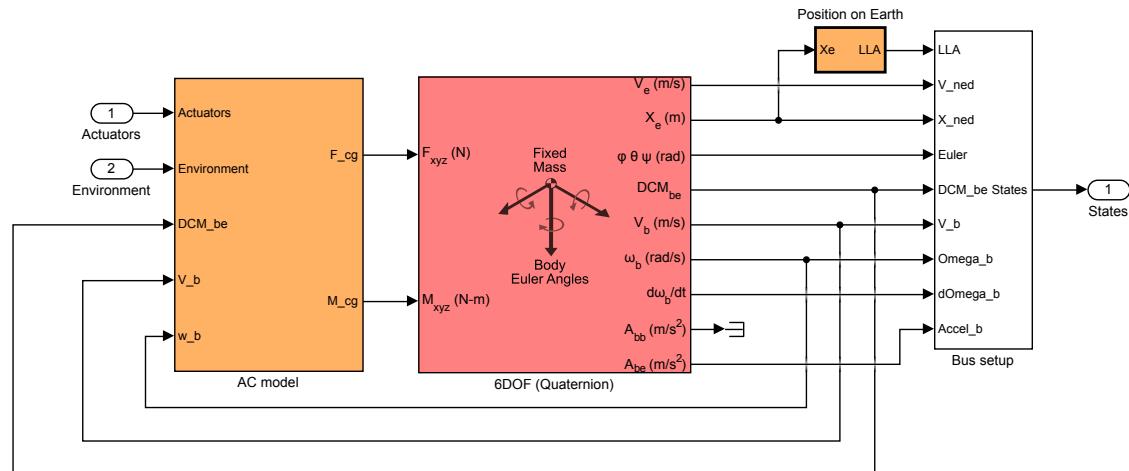


Figura 3.9: Modello non lineare

3.6 Controllore

Analizziamo adesso il **Control System** fornito facendo riferimento alla figura 3.10.

Esso non è formato solamente dal controllore come è lecito aspettarsi, ma da altri sottosistemi con funzioni altrettanto importanti. Il blocco *Crash Predictor Flags* viene utilizzato come logica di protezione da eventuali guasti. Esso riceve in ingresso gli stati stimati, come le posizioni e le velocità, e controlla che non ci siano anomalie, confrontandoli con dei valori soglia. Nel caso un'anomalia si verifichi, viene impostata una variabile FLAG a 1 per spegnere il minidrone.

Il blocco *State Estimator* riceve in ingresso le variabili misurate dai sensori e stima lo stato del sistema mediante algoritmi di sensor fusion, come il filtro complementare e il filtro di Kalman, che forniscono stime di variabili incognite attraverso la composizione di misurazioni osservate nel tempo da parte di più sensori. Questo stato servirà come ingresso al blocco *Path Planning* insieme al reference value per la logica del movimento del drone, e nel blocco *Controller* per poter confrontare lo stato attuale del sistema con quello di riferimento in uscita dal Path Planning in modo da spostare il drone secondo la logica desiderata.

In particolare, all'interno del blocco *Controller* abbiamo in totale 6 PID controller, alcuni dei quali in cascata. Prima di proseguire nella descrizione della funzione di ognuno di essi, facciamo un rapido richiamo teorico.

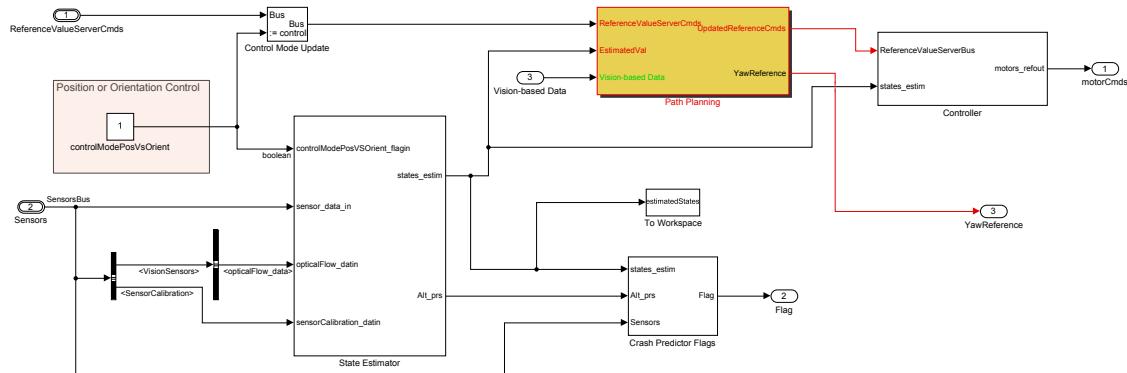


Figura 3.10: Control System

3.6.1 PID

Il controllore PID è uno dei controllori più usati perché semplice ma anche efficace. Riceve in ingresso l'errore tra il riferimento e la variabile di processo misurata e utilizza tre parametri, proporzionale, integrale e derivativo per portare questo errore a zero. Queste tre componenti vengono calcolate separatamente e poi sommate

algebricamente secondo la relazione:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

definendo $u(t)$ come l'output del controllore.

Analizziamo ora singolarmente ognuno dei tre parametri:

- P → PROPORZIONALE $C(s) = K_p$: Il termine P è proporzionale al valore attuale dell'errore $e(t)$.
Ha l'effetto di ridurre il tempo di salita (che è ciò che solitamente vogliamo ottenere) e riduce ma non elimina l'errore a regime.
- I → INTEGRALE $C(s) = K_i/s$: Il termine I tiene conto dei valori passati dell'errore $e(t)$ e li integra nel tempo per produrre il termine I. Per esempio, se c'è un errore residuo dopo l'applicazione del controllo proporzionale, il termine integrale cerca di eliminare l'errore residuo aggiungendo un effetto di controllo dovuto al valore cumulativo storico dell'errore.
Ha l'effetto di eliminare completamente l'errore a regime per un ingresso costante o per il gradino unitario, ma può rendere la risposta transitoria più lenta.
- D → DERIVATIVO $C(s) = K_d s$: Il termine D è la migliore stima della tendenza futura dell'errore $e(t)$, basata sul suo attuale tasso di variazione, poiché cerca effettivamente di ridurre l'effetto dell'errore esercitando un'influenza di controllo generata dal tasso di variazione dell'errore: Più rapido è il cambiamento, maggiore è l'effetto di controllo o di smorzamento.
Ha l'effetto di aumentare la stabilità del sistema, riducendo la sovraelongazione e migliorando la risposta transitoria.

Possiamo riassumere gli effetti aumentando il guadagno K di ciascun parametro di un controllore PID con la seguente tabella:

Parametro	Tempo di salita	Sovraelongazione	Tempo di assestamento	Errore a regime
K_p	Diminuisce	Aumenta	Piccola variazione	Diminuisce
K_i	Diminuisce	Aumenta	Aumenta	Eliminato
K_d	Piccola variazione	Diminuisce	Diminuisce	Invariato

Tabella 3.1: Schema azioni parametri regolatore PID

Inoltre, azzerando alcuni dei parametri, è possibile ottenere delle varianti dei controllori PID, come per esempio: il controllore PI (per $K_d = 0$), il controllore PD (per $K_i = 0$), oppure il controllore P (per $K_i = K_d = 0$).

Un aspetto importante da tenere presente è che il controllore PID non è fisicamente realizzabile perché la sua funzione di trasferimento è impropria, come si evince dalle sue rappresentazioni per un PID ideale in forma parallela (a sinistra) e per un PID ideale in forma standard (a destra):

$$R_{\text{PID}}(s) = \frac{K_d s^2 + K_p s + K_i}{s} \quad \text{e} \quad R_{\text{PID}}(s) = K_p \frac{(T_i T_d s^2 + T_i s + 1)}{T_i s}$$

dove $T_i = \frac{K_p}{K_i}$ e $T_d = \frac{K_d}{K_p}$ sono rispettivamente la costante di tempo integrale (o di reset) e la costante di tempo derivativo. Stiamo considerando una funzione di trasferimento impropria poiché il grado del numeratore è maggiore del grado del denominatore. Nella pratica è allora necessario introdurre un polo in alta frequenza che non modifica il funzionamento del controllore rispetto ad una risposta a un gradino, ma che renda propria la funzione di trasferimento, altrimenti non fisicamente realizzabile. Infatti una generica funzione di trasferimento differenziale del tipo:

$$G(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_0}{s^n + a_{n-1} s^{n-1} + \dots + a_0}$$

con $m > n$, ossia con un numero di zeri maggiore del numero di poli, corrisponde ad un'equazione differenziale del tipo:

$$\frac{d^n}{dt^n} y(t) + \frac{d^{n-1}}{dt^{n-1}} y(t) + \dots + a_0 = \frac{d^m}{dt^m} u(t) + \frac{d^{m-1}}{dt^{m-1}} u(t) + \dots + b_0$$

per cui l'uscita al generico istante t dipende dall'ingresso in quell'istante (quando i due esponenti coincidono), dagli ingressi precedenti e anche da uno o più ingressi futuri (poiché $m > n$) che chiaramente non sono realizzabili dal punto di vista fisico.

3.6.2 PID nel modello Simulink

Analizziamo in questa sezione lo schema del controllore all'interno del modello Simulink e mostrato in figura 3.11. Il modello del controllore è costituito da 6 diversi tipi di PID, alcuni dei quali in cascata, con l'obiettivo di mandare i giusti comandi agli attuatori in modo da mantenere il drone nell'altezza, nella posizione e con l'orientamento desiderato. Abbiamo visto nel capitolo 1 come possiamo agire su Thrust, Roll Pitch e Yaw indipendentemente, senza influenzare le altre azioni. Di conseguenza, possiamo creare un controllore per ognuna delle 4 azioni, quindi avremo sicuramente almeno 4 regolatori PID indipendenti.

Iniziamo ad analizzare uno ad uno i regolatori PID, con il relativo schema in figura, contenuti all'interno del modello ². Nel blocco 1 Yaw mostrato in figura 3.12,

²Gli schemi dei regolatori PID contengono i parametri già tarati come descritto in 3.2

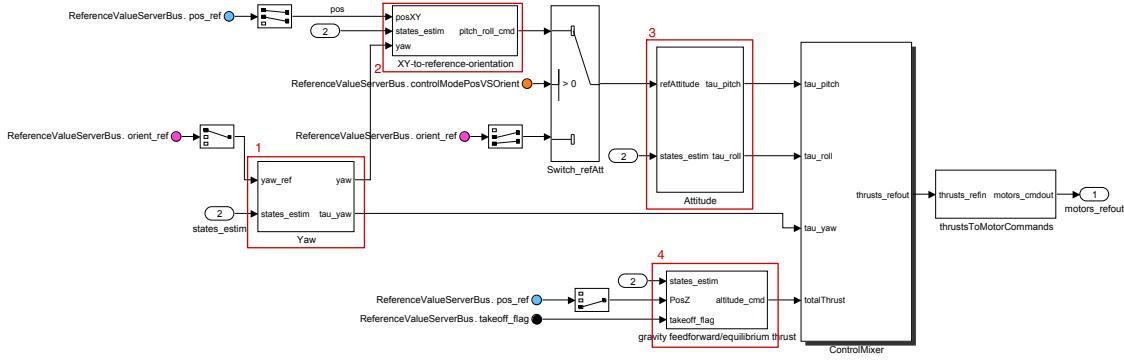


Figura 3.11: Schema Controllore

vi è il primo PID. Esso ricevendo in ingresso il valore di riferimento dell'angolo di Yaw e l'angolo di Yaw stimato attraverso lo State Estimator, restituisce in uscita non soltanto il momento necessario per portare l'errore dello Yaw tendente a zero come un classico controllore, ma restituisce nuovamente anche lo Yaw stimato in ingresso. In cascata, il blocco 2 in figura 3.13 riceve in ingresso la posizione stimata

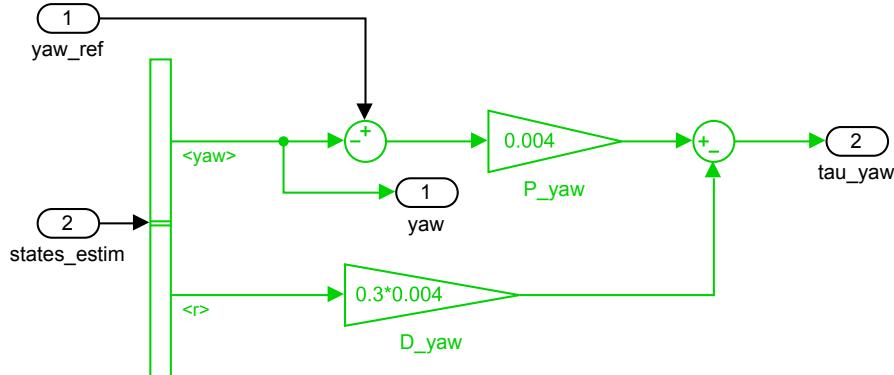


Figura 3.12: Controller Yaw

di X e Y e i loro rispettivi riferimenti e l'angolo di Yaw del blocco precedente. In questo caso i due PID hanno il compito di fornire gli angoli di Roll e Pitch di riferimento. L'obiettivo del blocco 2 in cascata è di permettere di mantenere il drone nella posizione d'interesse. Esso permette infatti di riconoscere se il drone si sta spostando a causa di disturbi esterni (come il vento) e fare le dovute correzioni per riportarlo nella posizione di riferimento X e Y. Di conseguenza in uscita avremo gli angoli di Roll e Pitch di riferimento per permettere di spostarsi nella posizione desiderata che serviranno da ingresso per il controllore rispettivamente degli angoli di Roll e Pitch successivo (infatti come abbiamo visto il drone per muoversi in avanti o a destra, deve prima compiere una rotazione intorno all'asse x o all'asse y rispettivamente). Questo tipo di controllori sono detti a cascata. Inoltre dobbiamo

usare l'angolo di Yaw poiché la posizione di X e Y è riferita rispetto al sistema di riferimento inerziale, mentre gli angoli di Roll e Pitch sono relativi al sistema di riferimento del drone. Quindi l'angolo di Yaw permette di passare tra i due sistemi di riferimento definiti. Il blocco 3 in figura 3.14 chiamato Attitude ha il compito

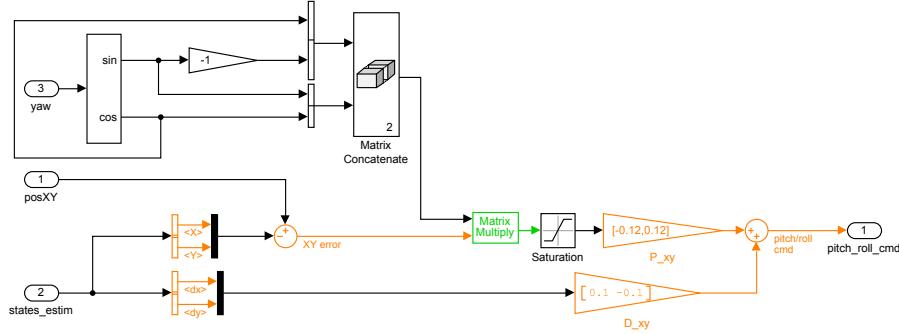


Figura 3.13: Controller XY to reference orientation

di mantenere l'orientamento del drone rispetto al riferimento degli angoli di Roll e Pitch. Esso riceve in ingresso proprio il riferimento generato dal controllore del blocco 2 e restituisce in output i momenti necessari per mantenere l'orientamento di Roll e Pitch. Infine il blocco 4 mostrato in figura 3.15 riceve in ingresso l'al-

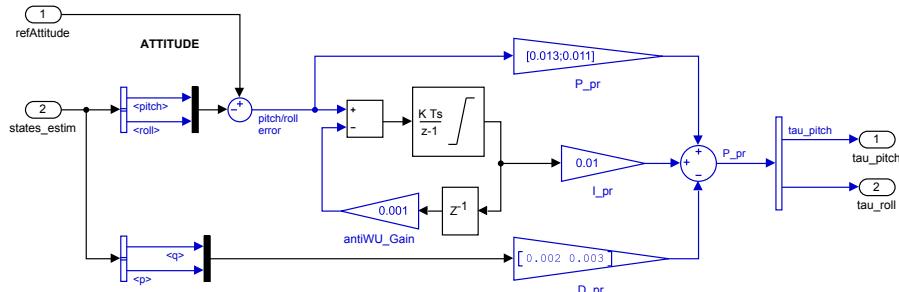


Figura 3.14: Controller Attitude

tezza di riferimento e il suo valore stimato dallo State Estimator per generare la forza di Thrust necessaria a mantenere il valore desiderato tenendo conto anche dell'accelerazione di gravità. Come possiamo notare dalle figure degli schemi dei controllori, ognuno di essi tiene conto anche della saturazione dei motori mettendo prima dell'effettiva uscita del momento anche un blocco di saturazione. Una particolarità della modellazione dei PID all'interno del progetto esistente, è che invece di utilizzare il blocco specifico fornito da Mathworks per rappresentare un PID, lo ottiene manualmente derivando il segnale e moltiplicandolo per il guadagno del parametro derivativo e dall'altro lato, integra il segnale che verrà poi moltiplicato per il guadagno integrale. Le 4 uscite dei controllori PID indipendenti sono dati in ingresso al blocco in figura 3.16 chiamato **Control Mixer**, che ha il compito di

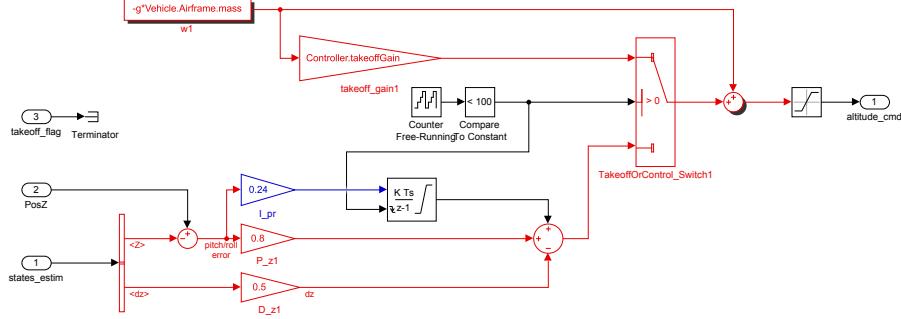


Figura 3.15: Controller Gravity

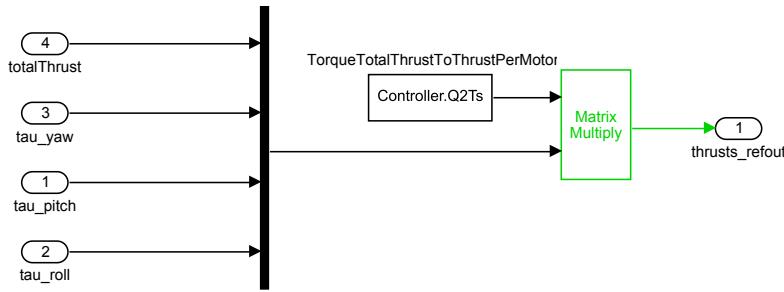


Figura 3.16: Blocco Control Mixer

convertire, secondo quanto visto nell’equazione 2.12, tutti i momenti rispetto agli angoli e le forze di Thrust nelle rispettive forze di ognuno dei 4 propulsori:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} T_{\text{total}} \\ M_\psi \\ M_\theta \\ M_\phi \end{bmatrix}$$

e infine nel blocco in figura 3.17 chiamato **ThrustToMotorCommands** la forza di ognuno dei propulsori viene convertita nella rispettiva velocità in RPM attraverso la seguente relazione:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{b} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

ottenuta da 2.3 a cui viene applicato un blocco **Saturation** per tenere conto della saturazione degli attuatori, e poi moltiplicata per il vettore:

$$\vec{s} = [1 \ -1 \ 1 \ -1]$$

per ottenere l’opportuno segno in relazione alla direzione di rotazione.

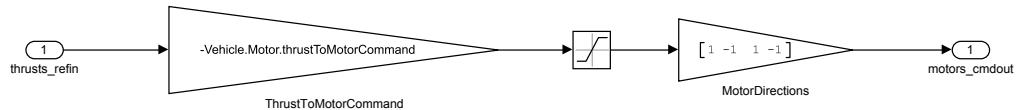


Figura 3.17: Blocco thrustsToMotorCommands

3.6.3 Taratura regolatori PID

Una volta descritto l’obiettivo di ciascun regolatore PID, proviamo a fare la taratura dei suoi parametri. Come detto in precedenza, sapendo che i i movimenti di Thrust, Roll, Pitch e Yaw sono disaccoppiati, i 4 regolatori PID sono indipendenti e quindi possiamo effettuare la taratura di un controllore alla volta senza modificare il comportamento degli altri. Per brevità, verrà riportato il procedimento di taratura solo di uno di essi, poiché il ragionamento è analogo anche per gli altri. In particolare, per la taratura del regolatore PID che controlla l’altitudine del veicolo, il modello è riportato in figura 3.18. Rispetto al modello originale in figura 3.6,

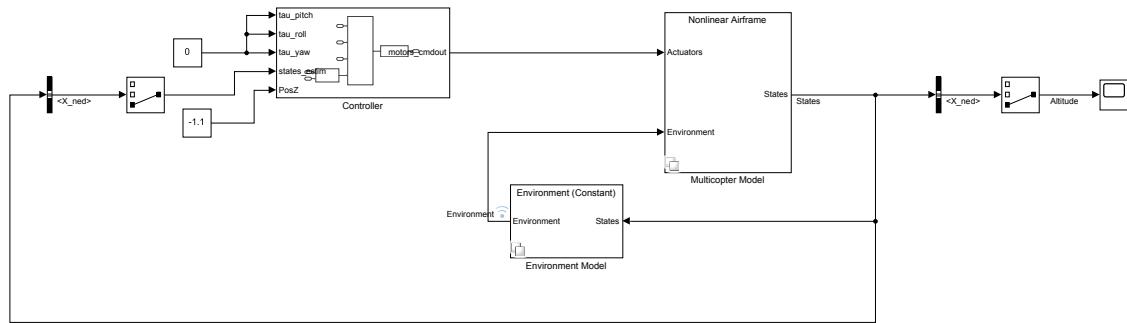


Figura 3.18: Modello per taratura PID altitudine

abbiamo mantenuto solo il modello del drone lineare, il modello dell’ambiente (in particolare con parametri costanti) e il sistema del controllore. Esso corrisponde al classico feedback control in cui il reference non viene generato automaticamente, ma viene fornito da noi, di conseguenza abbiamo eliminato i sensori dal modello di partenza. Per effettuare la taratura utilizziamo la PID Tuner App di Simulink, una feature integrata nel programma. Inizialmente, essa linearizza tutto il sistema, successivamente mostra il grafico in figura 3.19, in cui la linea tratteggiata rappresenta la risposta del sistema prima della taratura del controllore, mentre la linea continua mostra come risponde il sistema al variare dei guadagni del controllore PID. È quindi un metodo interattivo, che consente di modificare il valore dei parametri e vedere come il sistema risponde. Possiamo notare come nella risposta precedente la taratura presentava un overshoot nel transitorio, tipico dei controllori PID. Modificando il valore dei parametri siamo stati in grado di eliminare la sovraelongazione

a discapito di un leggero ritardo. Nel caso della taratura del regolatore PID per l'altitudine, manteniamo semplicemente come riferimento il valore della posizione Z, che settiamo uguale a 1.1 come durante la simulazione per la competizione, e estraiamo dagli stati solamente la variabile indicante l'altezza. Settiamo invece a 0 tutti i momenti per gli angoli di Pitch, Roll e Yaw in quanto non siamo interessati a controllarli. In questo modo abbiamo disaccoppiato il controllo solo rispetto alla traslazione lungo l'asse Z. Con lo stesso procedimento, possiamo tarare tutti gli

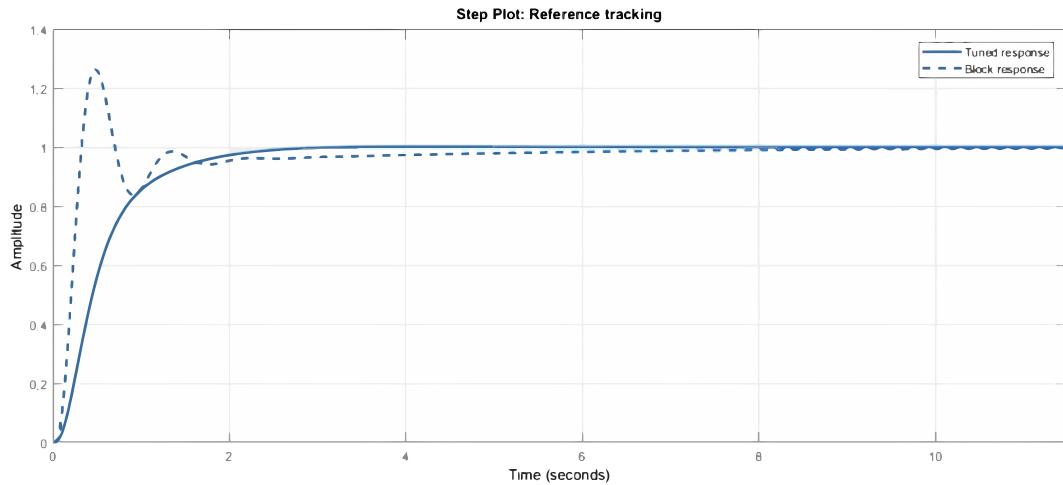


Figura 3.19: Risposta del sistema linearizzato

altri regolatori PID all'interno del modello.

Mostriamo quindi in tabella 3.2 il valore di tutti i parametri dei regolatori opportunamente tarati in cui i segni sono in accordo con la convenzione del riferimento del frame.

A questo punto possono essere eseguiti dei test in anello aperto per validare il modello matematico rispetto alle sue ipotesi e semplificazioni, e dei test in anello chiuso per validare tutto il sistema del controllo.

Tipo di controller	P	I	D
Yaw	0.004	/	0.3*0.004
X-Y to reference orientation	[-0.12,0.12]	/	[0.1,-0.1]
Attitude	[0.013,0.011]	0.01	[0.002,0.003]
Altitude	0.8	0.24	0.5

Tabella 3.2: Parametri tarati regolatori PID

Capitolo 4

Image Processing

Il primo step della competizione è fare in modo che il drone acquisisca immagini attraverso la sua telecamera rivolta verso il basso, per poi modificarle con diverse tecniche di elaborazione delle immagini. In questo capitolo mostreremo quindi la logica implementata all'interno dell'ambiente Simulink per l'image processing. Per ognuna di queste tecniche verrà prima spiegato il suo concetto teorico e successivamente la sua implementazione sotto forma di blocchi Simulink oppure di codice MATLAB. In seguito questi risultati serviranno per spostare il drone correttamente lungo il percorso secondo uno specifico algoritmo di controllo, ottenuto modificando periodicamente il riferimento di posizione e orientamento.

4.1 Elaborazione digitale delle immagini

Abbiamo visto durante la spiegazione dei sensori che il drone è capace di catturare un'immagine in formato YUY2V. Essa viene per prima cosa convertita in immagine RGB, la cui profondità colore delle immagini, ovvero il numero di valori di luminosità (che vanno dal nero al bianco) di ogni pixel, è di 8 bit, quindi avremo 256 livelli di intensità. Una qualunque immagine RGB può essere rappresentata come 3 matrici, ognuna delle quali contenente il valore da 0 a 255 rispettivamente del colore Rosso, Verde e Blu. L'unione di queste matrici identificherà il colore di ogni pixel in formato RGB. Poiché sappiamo che il path percorso che vogliamo identificare è di colore rosso, dovremo identificare quello specifico colore all'interno delle nostre immagini acquisite.

Per fare ciò, tramite un'immagine fittizia, effettuiamo un color Threshold mediante la Color Thresholder App che permette di identificare l'istogramma dei 3 canali colore.

L'istogramma in figura 4.1, ottenuto tramite questo strumento dopo aver selezionato la visualizzazione dei colori con modalità RGB, è la rappresentazione grafica sul modo in cui sono distribuiti i pixel di alta e bassa intensità luminosa

nell’immagine.

Sull’asse orizzontale sono rappresentati i valori di luminosità (che vanno dal nero al bianco). Sull’asse verticale viene rappresentato il numero complessivo di pixel corrispondenti ad ogni livello di luminosità. Anche in questo caso i valori di luminosità sull’asse orizzontale sono campionati a 8 bit per canale, dunque un istogramma mostrerà complessivamente 256 linee verticali. Quando si osserva l’istogramma di un’immagine è possibile visualizzarlo in un unico grafico (dove viene fatta una media dei valori luminosi RGB contenuti in ogni pixel), oppure (come in questo caso) gli istogrammi di ogni singolo canale RGB. Possiamo notare che nell’immagine fittizia scelta, abbiamo il picco sia all’estremo destro sia all’estremo sinistro dell’istogramma, dovuto, come mostrato nella prima immagine in figura 4.2, alla maggior presenza dei colori primari rosso, verde e blu, del bianco ma anche del nero, dato dalla mancanza di sfondo nell’immagine e quindi convertito automaticamente dal programma in colore nero. Possiamo muovere i cursori presenti nell’istogramma

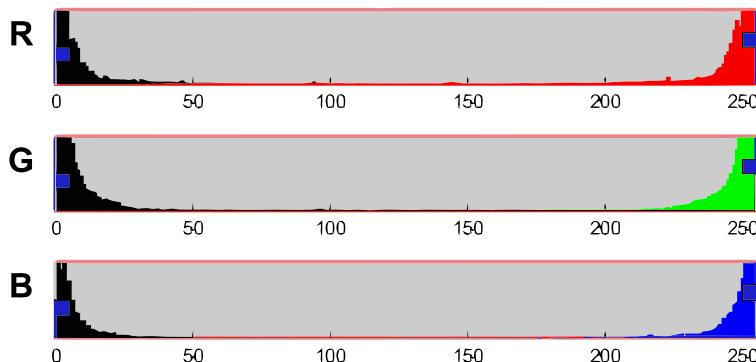


Figura 4.1: Istogramma canali colore RGB

per scegliere un valore di threshold per ogni canale. Successivamente, effettuiamo una segmentazione dell’immagine, in modo da assegnare ai pixel identificati con il colore rosso il valore di 1 (bianco), mentre assegniamo a 0 (nero) tutti gli altri; questo per avere minor calcolo computazionale durante l’esecuzione real time. Infine possiamo generare il codice MATLAB che fornirà una funzione che potremo utilizzare successivamente nel nostro modello Simulink. In conclusione, come mostrato in figura 4.2, l’immagine RGB acquisita in partenza è stata trasformata prima in una maschera in cui identifichiamo solo il colore rosso, e poi in un’immagine binaria, con soli valori di 1 quando abbiamo il colore rosso e 0 altrimenti.

4.2 Canny Edge Method

Il metodo Canny per l’Edge Detection é composto da vari step:

1. Importazione dell’immagine RGB

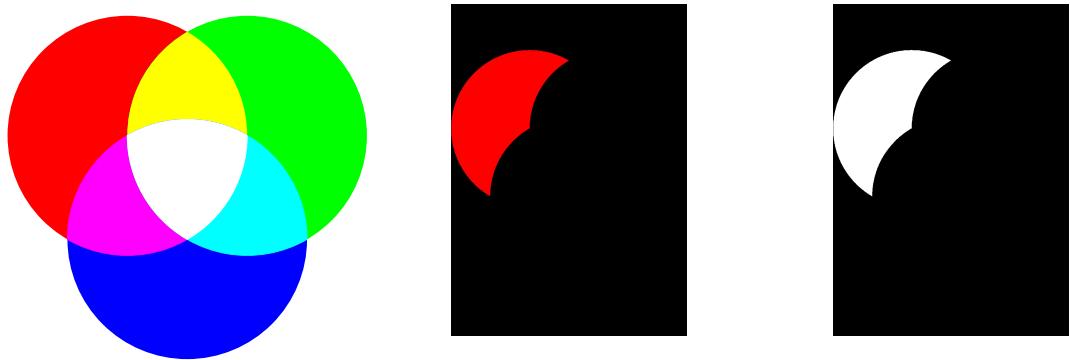


Figura 4.2: da sinistra: Immagine RGB, Maschera RGB, Maschera binaria

2. L’immagine RGB viene convertita nella corrispondente scala di grigi
3. Levigazione dell’immagine
4. Gradiente dell’immagine
5. Soppressione dei non massimi
6. Tracciamento dei bordi tramite isteresi

Poiché la nostra immagine in input è già stata segmentata, i primi due step vengono saltati.

4.2.1 Levigazione dell’immagine

L’obiettivo di questo step è di ridurre il rumore in alta frequenza presente all’interno dell’immagine. Esso è ottenuto mediante un Gaussian blur, ovvero attraverso la convoluzione dell’immagine con un Kernel di tipo gaussiano. Esso deriva dalla funzione Gaussiana, definita come:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

(conosciuta anche con il nome di distribuzione normale in statistica).

L’idea della levigazione tramite Kernel gaussiano è quella di utilizzare la distribuzione 2-D di una funzione gaussiana, definita come:

$$G_\sigma(m, n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{m^2+n^2}{2\sigma^2}\right)}$$

dove m saranno poi le righe e n le colonne della matrice Kernel. Essa può essere rappresentata come in figura 4.3 e i valori provenienti da questa distribuzione sono

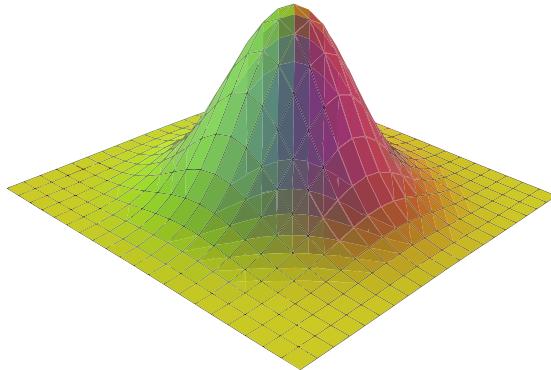


Figura 4.3: Distribuzione funzione gaussiana

discretizzati e usati per costruire la matrice di convoluzione applicata all’immagine originale. Tuttavia, in teoria, la funzione gaussiana è definita per qualsiasi numero reale (il suo insieme di definizione è \mathbb{R} allo stesso modo di una funzione esponenziale), ma nella pratica computazionale non è possibile includere tutti i valori. Viene perciò fatta un’approssimazione, per cui i numeri a una distanza maggiore di 3σ vengono considerati nulli. La matrice kernel diventa quindi della forma:

$$\frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

in cui possiamo notare come presenta, allo stesso modo della funzione gaussiana, il massimo nella parte centrale e diminuisce simmetricamente quando la distanza dal centro aumenta.

La convoluzione tra l’immagine originale e la matrice Gaussiana è un processo iterativo, ossia viene calcolato per ogni elemento della matrice originale secondo la semplice formula seguente:

$$y(t) = x(t) * h(t)$$

che perciò viene esplicitata dalla formula:

$$x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau$$

Nel nostro caso tuttavia la convoluzione viene fatta tra due matrici, ossia in 2D, tramite la formula

$$g[m, n] = x[m, n] * h[m, n] = \sum_{j=-a}^a \sum_{i=-b}^b x[i, j] \cdot h[m - i, n - j]$$

dove h rappresenta la risposta all'impulso (matrice Kernel). Essa, come mostrato nell'immagine in figura 4.4, consiste nella sovrapposizione delle due matrici, in cui il punto di applicazione del Kernel (ovvero il centro) viene fatto scorrere su ogni pixel dell'immagine.

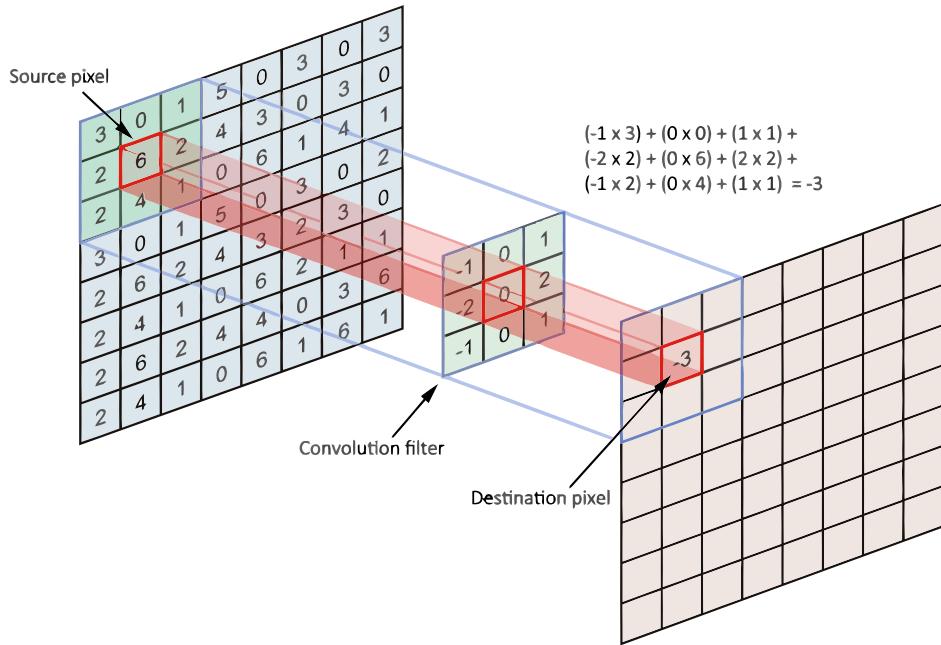


Figura 4.4: Convoluzione tra Matrice dell'immagine e matrice Kernel

4.2.2 Image gradient

Il gradiente è il vettore delle derivate parziali. Nel caso di un'immagine esso viene utilizzato per ottenere i cambiamenti repentina di intensità di colore. Anche in questo caso viene effettuata la convoluzione con una matrice di Kernel, chiamata operatore di Sobel, una comune approssimazione della derivata della Gaussiana, ossia due matrici 3x3 definite come:

$$g_m = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{e} \quad g_n = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

per ottenere il valore il modulo del gradiente, che definirà la forza del bordo

$$M(m, n) = \sqrt{g_m^2(m, n) + g_n^2(m, n)}$$

e la direzione del gradiente, data dall'angolo

$$\theta(m, n) = \arctan\left(\frac{g_n(m, n)}{g_m(m, n)}\right)$$

la quale verrà poi arrotondato a uno dei quattro angoli rappresentanti la verticale, l'orizzontale e le due diagonali (0° , 45° , 90° e 135° rispettivamente)

4.2.3 Soppressione dei non massimi

In questo step viene deciso se un punto è un massimo locale. Per fare ciò viene confrontato per ogni punto (ossia ogni pixel) il valore del modulo del gradiente ottenuto nella fase precedente, con il valore dei suoi due vicini a seconda dell'angolo trovato prima. Infatti:

- se l'angolo è di 0° , ossia rappresentabile con $-$, viene confrontato con i pixel alla sua destra e alla sua sinistra.
- se l'angolo è di 45° , ossia rappresentabile con $/$, viene confrontato con i pixel in alto a destra e in basso a sinistra
- se l'angolo è di 90° , ossia rappresentabile con $|$, viene confrontato con i pixel superiore e inferiore
- se l'angolo è di 135° , ossia rappresentabile con \backslash , viene confrontato con i pixel in alto a sinistra e in basso a destra

Se un massimo è rilevato, il valore del pixel viene mantenuto, altrimenti viene impostato il valore a 0.

4.2.4 Tracciamento dei bordi tramite isteresi

Durante l'ultimo step vengono definiti due threshold, uno di valore alto e uno di valore basso. Ogni pixel dell'immagine viene quindi comparato con questi due valori. Se il valore nel pixel è maggiore del threshold alto, allora viene assegnato il valore di 255, se invece è minore del threshold basso viene assegnato il valore di 0. Nel caso infine il pixel sia compreso tra i due valori di threshold, allora effettuiamo un ulteriore controllo:

- esso viene settato a 255 se è connesso a pixel che hanno un valore di threshold maggiore del massimo
- esso viene settato a 0 se è connesso a pixel che hanno un valore di threshold minore del massimo

Questo viene fatto in modo tale da avere dei bordi puliti, ossia senza troppe discontinuità e quindi avere un valore assegnato a tutti i pixel.

4.3 Trasformata di Hough

L’obiettivo della trasformata di Hough è di trovare le linee in un’immagine in una qualsiasi direzione. Essa si basa sul seguente principio: una qualsiasi linea può essere rappresentata tramite la relazione seguente:

$$\rho = x \cos(\theta) + y \sin(\theta)$$

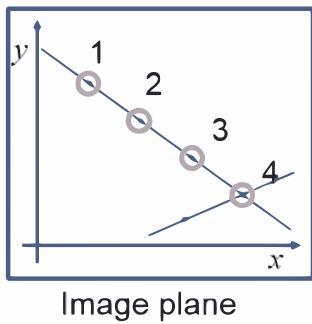
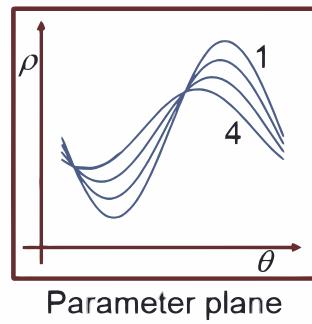


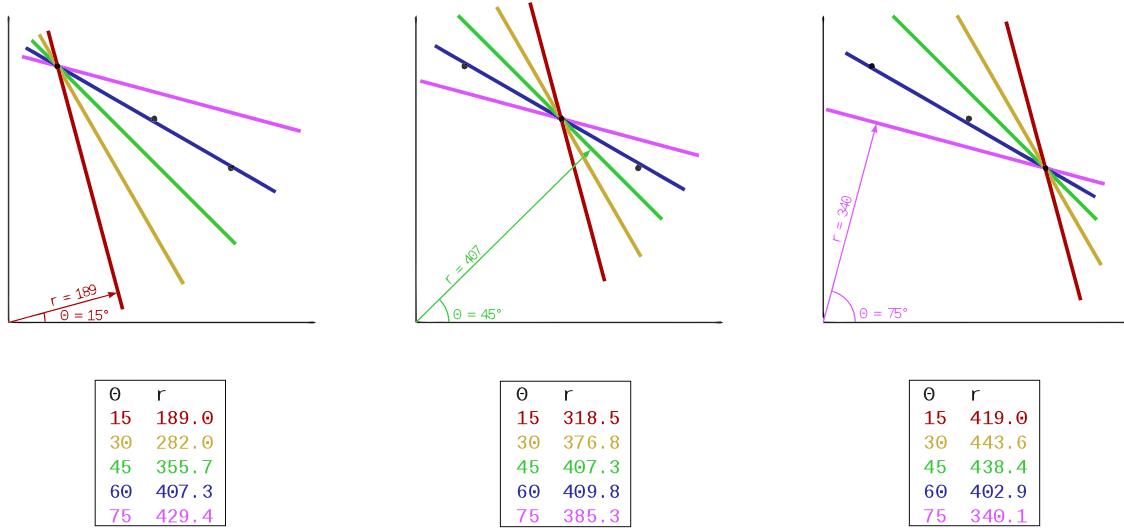
Image plane



Parameter plane

dove x e y sono le coordinate del punto nell’immagine, θ è l’angolo della linea che collega il punto all’origine, ρ rappresenta la perpendicolare tra la linea che collega il punto, e rappresenta la distanza del punto dall’origine.

Successivamente, per ogni punto vengono tracciate diverse linee a tutti gli angoli differenti, come mostrato nella figura successiva. Per ognuna di queste linee vengono



riportati in una matrice di coordinate ρ e θ nell’indice corrispondente il numero di volte che i due valori sono stati trovati. Da notare che nel caso di MATLAB, poiché i

valori di ρ e θ sono potenzialmente infiniti, i valori di θ sono raggruppati in intervalli di 4, mentre viene usato un sample time di 1 per ρ , fornendo così una matrice di 45x180. Successivamente per ogni coppia (ρ, θ) trovata, si aumenta di 1 il valore nella casella corrispondente. Infine con il comando `houghpeaks`, vengono estratte le coordinate ρ e θ del valore accumulato maggiore, che corrisponderà al nostro angolo individuato. Un aspetto molto importante da tenere in considerazione è che gli angoli definiti dalla trasformata sono compresi nell'intervallo $\theta \in [-90^\circ, 90^\circ]$. Questo comporterà considerazioni successive nell'algoritmo di Path Planning.

4.4 Image Processing System

Iniziamo ad analizzare la soluzione proposta in figura 4.5, in cui viene mostrata la logica per l'Image Processing System.

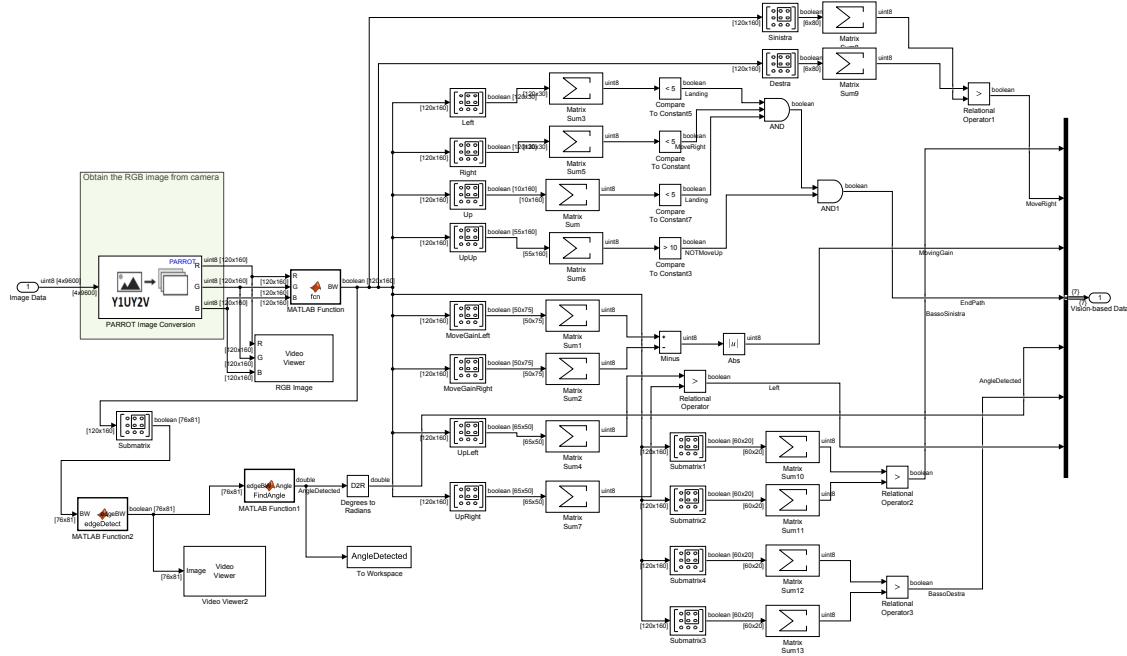


Figura 4.5: Image Processing System

L'unico ingresso del sottosistema è l'immagine ottenuta dai sensori che viene inviata al blocco **Parrot Image Conversion** dove l'immagine catturata dal drone in formato YUY2V viene convertita in formato RGB cosicché in uscita abbiamo tre linee, ognuna dello specifico colore: Rosso, Verde e Blu. Questi tre valori indicanti come è fatta l'immagine, vengono dati in ingresso a una funzione MATLAB **Color Threshold** mostrata nel listato 4.1, con il compito di definire i threshold per ogni canale colore e ottenere l'immagine in forma binaria in cui avremo 1 in

corrispondenza di pixel completamenti rossi e 0 altrimenti. Questo sarà utile anche per svolgere la somma dei pixel nelle sottomatrici nei passaggi successivi.

```
1 function BW = Color Threshold(R, G, B)
2 I = cat(3,R,G,B);
3
4 % Define thresholds for channel 1 based on histogram settings
5 channel1Min = 150.000;
6 channel1Max = 255.000;
7
8 % Define thresholds for channel 2 based on histogram settings
9 channel2Min = 0.000;
10 channel2Max = 5.000;
11
12 % Define thresholds for channel 3 based on histogram settings
13 channel3Min = 0.000;
14 channel3Max = 5.000;
15
16 % Create mask based on chosen histogram thresholds
17 sliderBW = (I(:,:,1) >= channel1Min) & (I(:,:,1) <= channel1Max) & ...
18     (I(:,:,2) >= channel2Min) & (I(:,:,2) <= channel2Max) & ...
19     (I(:,:,3) >= channel3Min) & (I(:,:,3) <= channel3Max);
20 BW = sliderBW;
21
22 end
```

Listato 4.1: Color Threshold

Successivamente restringiamo l’immagine modificata con un blocco **SubMatrix**, in modo da mantenere solo la parte centrale dell’immagine che viene messa in ingresso a una nuova funzione MATLAB **edgeDetected**, mostrata nel listato 4.2, con lo scopo di identificare i bordi del percorso, secondo la tecnica descritta nel paragrafo 4.2.

```
1 function edgeBW = edgeDetected(BW)
2
3 edgeDet = edge(BW, 'canny'); %Canny edge method;
4
5 edgeBW = edgeDet;
```

Listato 4.2: Edge Detection

Infine attraverso l’ultima funzione MATLAB chiamata **FindAngle**, mostrata nel listato 4.3, calcoliamo gli angoli dei bordi nell’immagine attraverso la trasformata di Hough, e preleviamo nella matrice il picco corrispondente all’angolo più significativo nell’immagine. Infine convertiamo in gradi l’angolo stimato in radianti attraverso il blocco **Rad2Deg**.

```

1 function Angle = FindAngle(edgeBW)
2
3 [H, theta , rho ] = hough(edgeBW); %Hough Transform ;
4 peaks = houghpeaks(H,1); %peak values ;
5 DegAngle = max( theta ( peaks (:,2) )); %find angle value ;
6
7 Angle = DegAngle;

```

Listato 4.3: Find Angle

Nello stesso sub system iniziamo a sviluppare delle logiche di controllo che saranno utili per il Path Planning. Quindi a partire dall'immagine completa binaria in uscita alla funzione **Color Threshold** estraiamo delle sottomatrici con funzioni specifiche. Per esempio prendiamo la sottomatrice *UpLeft* e *UpRight*, in ognuna delle quali facciamo la somma dei pixel attraverso il blocco **matrixSum**, e confrontiamo con operatore relazionale quale delle due è maggiore, che indicherà la direzione del percorso: verso sinistra se l'uscita è 1 oppure verso destra se l'uscita è 0.

Allo stesso modo per capire se il percorso rispetto al drone si trova in basso a destra o in basso a sinistra, sfruttiamo la stessa logica precedente estraendo rispettivamente la sottomatrice in basso a destra e in alto a destra, e la sottomatrice in basso a sinistra e in alto a sinistra, che verranno confrontate con l'operatore relazionale maggiore.

Con la stessa logica abbiamo infine le sottomatrici *Destra* e *Sinistra* che confrontano il numero di pixel a 1 dovuti alla presenza del percorso a destra e a sinistra del drone.

Abbiamo poi le sottomatrici *MoveGainLeft* e *MoveGainRight* il cui risultato, dopo aver fatto le rispettive somme matriciali, viene sottratto e preso in valore assoluto con lo scopo di capire la differenza di pixel a destra e a sinistra del drone. Maggiore sarà questo valore, più il drone sarà lontano a destra o a sinistra dal centro esatto del percorso da seguire.

Infine eseguiamo la logica che permetterà di definire la fine del percorso e quindi il successivo abbassamento del drone all'interno del cerchio finale. Per fare ciò estraiamo le sottomatrici *Left*, *Right* e *Up*, ognuna delle quali dopo la somma matriciale viene confrontata con un threshold di 5, se minore restituisce 1, altrimenti 0. Ognuno di questi 3 output viene poi inserito in ingresso a un blocco operatore logico **AND**, che di conseguenza restituirà il valore 1 (true) solamente se ognuno degli ingressi è vero. Poco più sotto abbiamo anche la sottomatrice *UpUp*, a cui a sua volta viene fatta la somma matriciale, ma in questo caso viene confrontato se maggiore del valore di threshold di 10. Infine il risultato di questa operazione e del operatore AND precedente viene messo in ingresso a un nuovo AND che di conseguenza restituirà il valore 1 di true solo se tutte le condizioni precedenti sono verificate. Il significato di quest'ultima serie di blocchi è, come detto, di capire se siamo arrivati alla fine del path da seguire. Infatti sappiamo che il percorso, secondo la regole definite dalla

competizione, termina con una linea, come mostrato in figura 3.1. Di conseguenza verifichiamo che sia a destra, che a sinistra, che poco sopra il centro dell’immagine (blocco *Up*) catturata dal drone, abbiamo un numero totale di pixel rossi minore di 5, che indicherà la linea nera prima del cerchio; e sopra l’immagine (blocco *UpUp*) abbiamo ancora molti pixel bianchi (> 10), indicando che c’è un nuovo percorso in seguito a una discontinuità e che, di conseguenza, sarà appunto il cerchio finale.

Ognuno di questi risultati viene mandato in un blocco **Bus Creator**, che crea appunto un bus con definiti all’interno tutti i risultati che saranno trasmessi con il blocco **Outport**, al sottosistema successivo, il Control System per il Path Planning.

Capitolo 5

Pianificazione dello spostamento

In questo capitolo mostriamo la logica implementata in Simulink per controllare il movimento del drone in relazione ai dati acquisiti dalla camera ed elaborati con le tecniche di image processing, e successivamente una strategia per permettere al quadricottero di correggere la traiettoria nel caso esso non sia al centro del tracciato. Entrambi questi metodi sono ottenuti modificando periodicamente i valori di riferimento della posizione e dell'orientamento riferiti al sistema di riferimento inerziale.

5.1 Path Planning

Analizziamo in questo capitolo il Path Planning all'interno del controllore mostrato in figura 5.1. A differenza del precedente, analizzato nel capitolo 4, in questo caso il sottosistema è formato da più ingressi, ognuno dei quali ottenuto da differenti parti del modello Simulink attraverso il blocco **Iport**. Il **ReferenceValueServerCmds** acquisisce i valori di riferimento generati dal **Signal Builder**, l'**EstimatedVal** corrisponde agli stati stimati dallo State Estimator attraverso gli algoritmi di state fusion a partire dai dati ricevuti dai sensori, e infine la **Iport Vision-based Data** riporta i valori in uscita determinati dallo schema dell'Image Processing. Obiettivo di questo schema è di aggiornare i valori di riferimento, che serviranno da ingresso al controllore, per far sì che il drone aggiorni il suo stato seguendo il percorso della gara. Di conseguenza, l'unica uscita è il blocco **Outport UpdatedReferenceCmds**. Tuttavia, di tutti i parametri stimati dallo State Estimator visibili in uscita al blocco **Bus Selector**, che come abbiamo visto corrispondono agli stati definiti nel capitolo 4, ossia le tre posizioni X, Y e Z, le tre velocità dx , dy e dz , l'orientamento con gli angoli di Roll, Pitch eYaw e infine le velocità angolari p , q , r ; per l'algoritmo di Path Planning basteranno solo le posizioni X, Y e Z e l'orientamento del veicolo

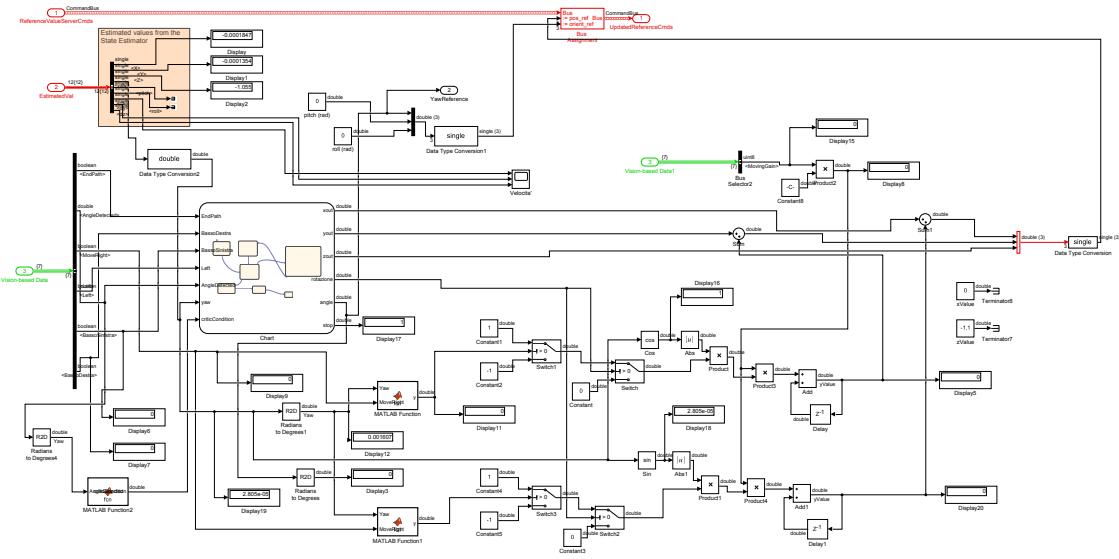


Figura 5.1: Path Planning

dato dal solo angolo di Yaw. Questi tre stati saranno modificati a nostro piacimento attraverso il **Bus Assignment**, che riassegna i valori di riferimento all'interno del bus.

Lo schema del Path Planning possiamo dividerlo in due parti:

- Lo Stateflow per definire il movimento direzionale del veicolo
- una logica di condizioni che permettono il raddrizzamento della traiettoria rispetto al percorso

Prima di descrivere il funzionamento della soluzione, è importante ricordare il sistema di riferimento in simulazione del drone, che, come già visto in figura 2.5, è riportato con vista dall'alto in figura 5.2. Di conseguenza gli angoli nella direzione in alto a destra saranno positivi, mentre nella direzione in alto a sinistra avranno X positiva e Y negativa.

5.2 Stateflow

Stateflow è uno strumento logico di controllo utilizzato all'interno di un modello Simulink. Nello Stateflow, chiamato anche macchina a stati finiti, un solo stato può essere attivo. Il passaggio di stato avviene al verificarsi di una certa condizione, che può essere logica ma anche temporale. Inoltre all'interno di uno stato possiamo definire tre azioni: *entry* in cui le condizioni vengono eseguite una sola volta quando lo stato diventa attivo, *during* in cui le condizioni vengono eseguite fintanto che lo

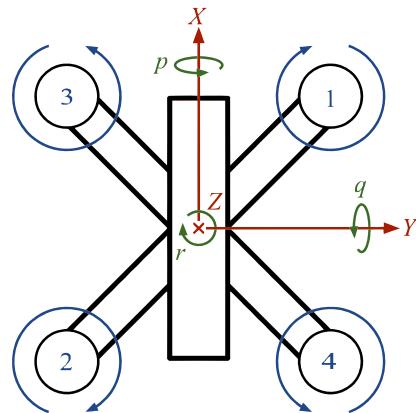


Figura 5.2: Sistema di riferimento solidale

stato è attivo, quindi possono essere eseguite più volte, e *exit* in cui le condizioni vengono eseguite una sola volta all'uscita dello stato, prima di passare a quello successivo.

Lo Stateflow riceve in ingresso alcune delle condizioni che abbiamo ricavato durante l'Image Processing System. Inoltre riceve in ingresso l'angolo di Yaw stimato attraverso lo State Estimator e la condizione *criticCondition* in uscita dall'omonima funzione MATLAB mostrata nel listato 5.1. Questa funzione ha il compito di controllare se l'angolo individuato attraverso la trasformata di Hough è nell'intorno dei $\pm 90^\circ$, valore critico, e in caso affermativo mettere in uscita il valore di 1.

```

1 function criticCondition = fcn(AngleDetected)
2
3 if (AngleDetected <= 90 & AngleDetected >= 85) | (AngleDetected >=
-90 & AngleDetected <= -85)
    %mi considera entrambi i casi (sia sopra che sotto) l'asse delle
    %ascisse
    u = 1;
else
    u = 0;
end
criticCondition = u;
```

Listato 5.1: Critic Condition

Infatti in questo intervallo, sappiamo che la trasformata di Hough calcola correttamente l'angolo, ma individua in maniera errata il verso. Inoltre poiché l'angolo stimato dalla trasformata di Hough è espresso in radianti, mettiamo a monte un blocco **Rad2Deg** per avere gli angoli definiti in gradi. La nostra macchina a stati finiti è composta da 7 stati: TAKEOFF, MOVING, ROTATING, ASSESTAMENTO, ENDPATH,

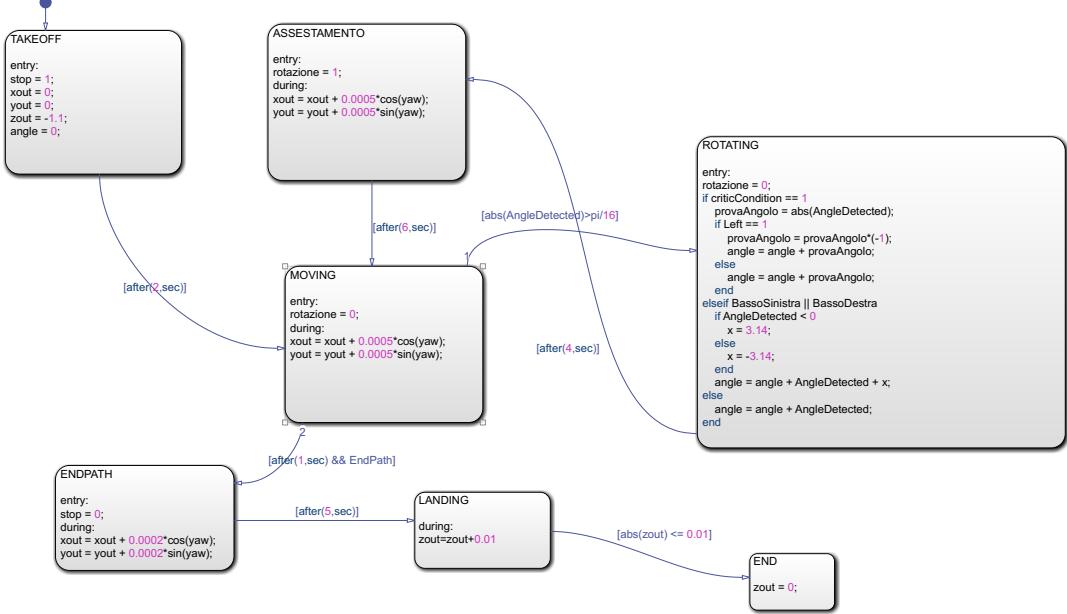


Figura 5.3: Stateflow

LANDING e **END**. All'avvio della macchina a stati, per definizione, entriamo nello stato connesso con una freccia (le transizioni) a un cerchio pieno, chiamato in questo caso **TAKEOFF**. All'entrata, portiamo zout al valore di -1.1 (ricordiamo che l'asse Z è rivolto verso il basso), in modo da portare il nuovo riferimento di Z e far sollevare il drone alla corretta altezza, e settiamo la variabile stop a 1. Dopo 2 secondi, la condizione di transizione viene verificata e si passa allo stato **MOVING**, in cui dato l'angolo di Yaw stimato, settiamo i nuovi riferimenti per la X e per la Y. È necessario moltiplicare rispettivamente per il $\cos(\text{yaw})$ e per il $\sin(\text{yaw})$ poiché i valori di riferimento saranno tutti rispetto al sistema di riferimento inerziale terrestre. A questo punto gli stati di uscita sono 2: **Rotating** e **EndPath**. Nel caso venga individuato un angolo, in radianti, il cui valore assoluto è maggiore o uguale a $\pi/16$ si passa nello stato **ROTATING**. Inizialmente si verifica se siamo nella condizione critica, se lo siamo prima ricaviamo il valore assoluto dell'angolo, e poi se **Left==1**, ossia abbiamo più rosso a sinistra che a destra, vuol dire che il nostro percorso prosegue a sinistra, e di conseguenza moltiplichiamo l'angolo per -1 , altrimenti il percorso sarà verso destra e quindi manteniamo l'angolo che sarà positivo. Se non siamo nella condizione critica, verifichiamo prima che **BassoSinistra** e **BassoDestra** siano entrambi falsi. Se non lo sono infatti, vuol dire che abbiamo più rosso in basso che nel rispettivo alto, e quindi dovremo sommare un angolo di $+180^\circ$ se la rotazione è verso destra o -180° se è verso sinistra (in modo da fare il minor angolo di rotazione possibile). Il motivo è che, come detto nel capitolo 4, la trasformata di Hough è in grado di

identificare angoli compresi solo tra $\theta \in [-90^\circ, 90^\circ]$, e quindi un angolo di $+60^\circ$ potrebbe in realtà corrispondere all'angolo nel quadrante opposto, in questo caso -150° . Se invece nessuna delle due condizioni è verificata, il percorso dovrà trovarsi in avanti e quindi potremo semplicemente sommare il valore di angolo individuato. Alla fine, il valore assegnato a angle, andrà a modificare il riferimento dell'angolo di Yaw, per cui il drone potrà effettuare una rotazione lungo l'asse Z. Dopo 4 secondi, tempo necessario per superare il transitorio del controllore, si passa nello stato di **ASSESTAMENTO**. Questo stato è identico al precedente stato di **MOVING**, con l'eccezione di settare a 1 la variabile rotazione, che servirà nella seconda parte dello schema descritta nella prossima sezione. Dopo 6 secondi, si ritorna nello stato di **MOVING**, resettando la variabile rotazione a 0. Nel caso invece che durante lo stato di **MOVING**, passino 1 secondo e la condizione di EndPath sia vera, proseguiamo nello stato di **ENDPATH**. L'obiettivo di questo stato è far diminuire la velocità del drone, in quanto in prossimità del cerchio finale in cui dovremo atterrare, e settare la variabile stop a 0, variabile che fino ad ora era sempre rimasta a 1 dopo averla settata nel primo stato di **TAKEOFF**. Infine conoscendo la distanza dal centro del cerchio, specificata nelle regole della competizione e mostrata in figura 3.1, e la velocità del drone, definita da noi, è possibile ricavare il tempo. Di conseguenza, con una certa approssimazione, sappiamo che dopo 5 secondi saremo al di sopra del centro del cerchio, e quindi sarà la condizione per passare allo stato successivo **LANDING**. In questo stato facciamo scendere di quota il drone, incrementando la variabile zout di 0.01, in modo che la discesa non sia troppo brusca (avremmo infatti potuto settare fin da subito il valore di zout a 0, ma con il rischio di danneggiare il drone a causa di overshoot molto probabile dovuto all'azione del controllore). Giunti a un valore di zout in valore assoluto minore o uguale di 0.01, possiamo finalmente transitare nello stato finale **END**, in cui settiamo finalmente il valore di zout a 0, per raggiungere il terreno, evitando così di far continuare a incrementare il valore di zout al di sotto dello 0 se fossimo rimasti nello stato precedente.

5.3 Assestamento

L'obiettivo della logica che stiamo per descrivere è quello di far cambiare dinamicamente la traiettoria del drone per correggere gli eventuali ritardi o anticipi nella rotazione, poiché rispetto a quanto descritto fino ad ora, una volta identificato un angolo, fino a questo punto siamo in grado solamente di seguire la direzione descritta da esso, senza la possibilità di correggerla. Con lo schema di controllo proposto, cercheremo di mantenere il drone al centro del percorso. Per fare ciò, dobbiamo tenere presente rispetto a quale sistema di riferimento i nostri stati stimati dallo State Estimator e le condizioni nella logica dello schema dell'Image Processing, sono definiti. Infatti mentre tutti gli stati sono definiti rispetto al sistema di riferimento inerziale, in particolare tramite simulazioni abbiamo dedotto che il sistema di riferimento fisso è definito come mostrato in basso a sinistra della figura 6.2, i

sensori del minidrone (in particolare la camera che acquisisce l’immagine del percorso) sono definiti rispetto al drone, poiché chiaramente montati su di esso. Facendo riferimento alla figura 5.4, possiamo definire quattro specifiche situazioni:

- Nel caso il nostro drone stia percorrendo una traiettoria con un angolo definito all’interno del *primo quadrante* → per allinearci alla destra del drone dovremo **diminuire** la X e **aumentare** la Y, per allinearci a sinistra dovremo **aumentare** la X e **diminuire** la Y.
- Nel caso il nostro drone stia percorrendo una traiettoria con un angolo definito all’interno del *secondo quadrante* → per allinearci alla destra del drone dovremo **aumentare** la X e **aumentare** la Y, per allinearci a sinistra dovremo **diminuire** la X e **diminuire** la Y.
- Nel caso il nostro drone stia percorrendo una traiettoria con un angolo definito all’interno del *terzo quadrante* → per allinearci alla destra del drone dovremo **aumentare** la X e **diminuire** la Y, per allinearci a sinistra dovremo **diminuire** la X e **aumentare** la Y.
- Nel caso il nostro drone stia percorrendo una traiettoria con un angolo definito all’interno del *quarto quadrante* → per allinearci alla destra del drone dovremo **diminuire** la X e **diminuire** la Y, per allinearci a sinistra dovremo **aumentare** la X e **aumentare** la Y.

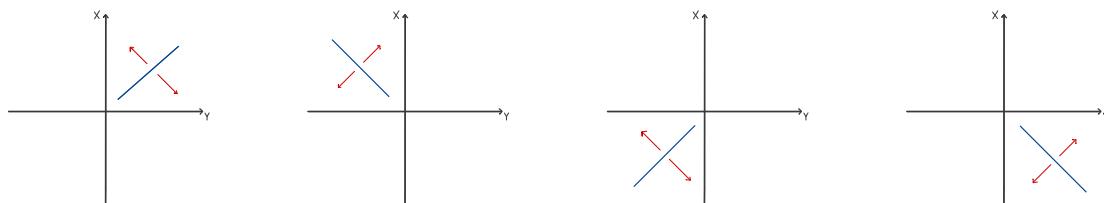


Figura 5.4: Analisi del movimento per Assestamento

La logica appena descritta viene riassunta in figura 5.5 raffigurante in un unico grafico come cambia la X e la Y per ogni quadrante.

Quindi inizialmente facciamo un controllo con le funzioni MATLAB **AssestamentoX** e **AssestamentoY** da noi definite.

Nella prima, riportata nel listato 5.2, verifichiamo se siamo in una regione in cui se dovendo muoverci a destra abbiamo un incremento della X oppure dovendo muoverci a sinistra abbiamo un incremento della X.

```

1 function y = AssestamentoX(Yaw, MoveRight)
2
3 if (MoveRight==1 & (Yaw<1 & Yaw>-181)) | (MoveRight==0 & (Yaw>-1 &
    Yaw<181))

```

```

4 % quadranti in cui per muoversi a destra e' necessario
5     incrementare X
6 out = 1;
7 else
8     out = 0;
9 end
10 y = out;

```

Listato 5.2: Assestamento X

Nella seconda, riportata nel listato 5.3, verifichiamo se siamo in una regione in cui se dovendo muoverci a destra abbiamo un incremento della Y oppure dovendo muoverci a sinistra abbiamo un incremento della Y.

```

1 function y = AssestamentoY(Yaw, MoveRight)
2
3 if (MoveRight==1 & (Yaw>-91 & Yaw<91)) | (MoveRight==0 & ((Yaw>89 &
4     Yaw<181) | (Yaw<-89 & Yaw>-181)))
    % quadranti in cui per muoversi a destra e' necessario
    % incrementare Y
    out = 1;
6 else
7     out = 0;
8 end
9
10 y = out;

```

Listato 5.3: Assestamento Y

In entrambi i casi, se le condizioni sono verificate, in uscita avremo la variabile y a 1, altrimenti a 0. Questa variabile farà cambiare stato al primo blocco **Switch**, i cui due ingressi sono le costanti 1 e -1. In entrambi i casi, in cascata abbiamo poi un

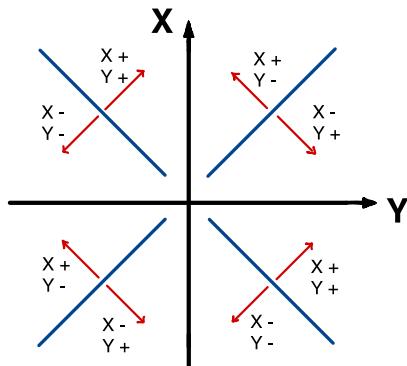
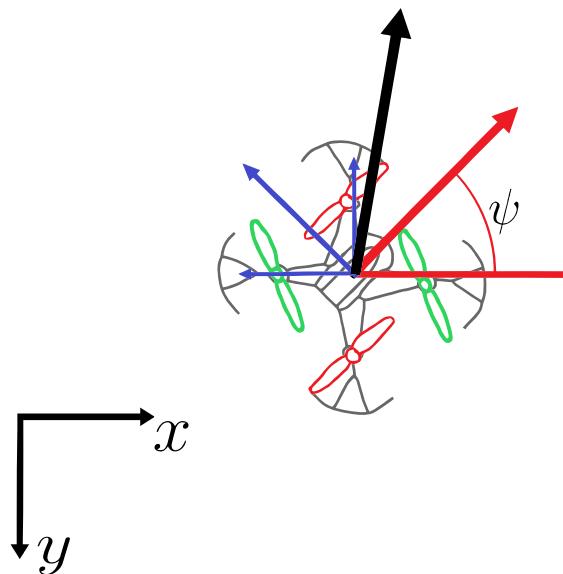


Figura 5.5: Schema Assestamento

secondo **Switch**, la cui condizione è la variabile rotazione definita all'interno dello Stateflow. Se rotazione è uguale a 1, allora facciamo passare l'uscita del precedente **Switch**, altrimenti facciamo passare 0. Questo perché, mentre siamo nello stato **ROTATING**, ovvero viene modificato il riferimento dello Yaw per permettere la rotazione del drone durante un cambio di direzione del percorso, non vogliamo alcun assestamento, che rischierebbe di compromettere la rotazione e neanche durante la fase di **MOVING**, poiché dalla simulazione ci siamo accorti che uno spostamento ortogonale rispetto al percorso in prossimità di una deviazione di percorso, potrebbe far calcolare alla trasformata di Hough un angolo errato. Successivamente moltiplichiamo per il seno dell'angolo di Yaw stimato per il cambiamento della coordinata X (poiché lo spostamento è ortogonale alla traiettoria) e per il coseno dell'angolo per il cambiamento della coordinata Y, a cui segue in entrambi i casi un valore assoluto, dato che il segno lo abbiamo già definito prima in base al quadrante. Infine il risultato viene moltiplicato per un guadagno determinato dalla differenza di pixel tra la sottomatrice destra e sinistra che avevamo definito nel capitolo precedente **MoveGainLeft** e **MoveGainRight**, moltiplicato per una costante per definire lo spazio di spostamento. Arrivati a questo punto sfruttiamo in entrambi i casi il blocco **Delay**, che combinato con il blocco **Sum** fornisce la classica operazione incrementale. La logica si conclude sommando questo valore (che potrà essere positivo o negativo) al corrispettivo valore di $xout$ e $yout$ che modificherà il riferimento attuale consentendo al drone di aggiustare la propria traiettoria. Il risultato sarà quindi una somma vettoriale che modifica la traiettoria del drone in movimento come riportato nella figura successiva ipotizzando di essere nel primo quadrante di figura 5.5 e di volerci spostare verso sinistra. Il vettore nero sarà quindi la direzione risultante del moto.



Capitolo 6

Simulazione

In quest'ultimo capitolo faremo delle considerazioni rispetto alle simulazioni eseguite in ambiente Simulink. Il risultato di ognuna di queste simulazioni grafiche può essere visionato cliccando sulla didascalia delle immagini dei tracciati in Appendice C. In questo capitolo ci soffermeremo solamente sul tracciato numero 6. Lo schema di simulazione, rappresentato in figura 6.1, è costituito dal blocco a destra che riceve in ingresso tutte le informazioni relative alla posizione e orientamento del drone e le informazioni sul percorso. Esso permette di visualizzare l'andamento

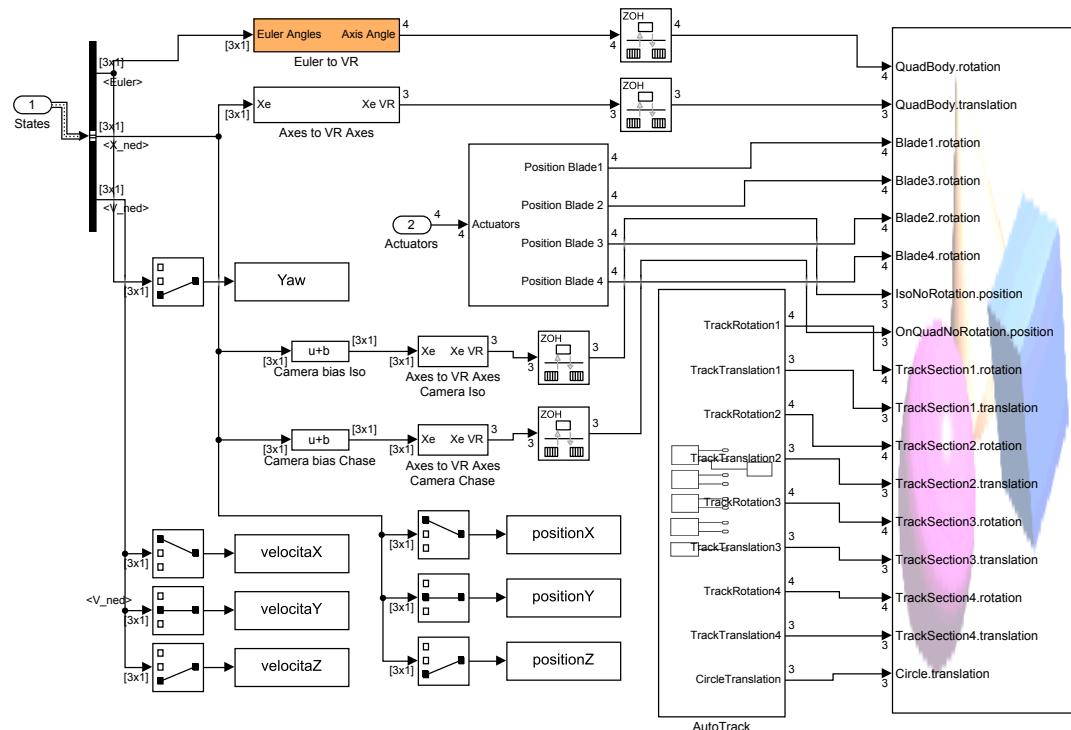


Figura 6.1: Simulation

della simulazione in un ambiente 3D, come mostrato in figura 6.2. Il tracciato scelto è il numero 6 rispetto alla lista riportata in Appendice C.



Figura 6.2: Simulazione Tracciato 6

Mostriamo inoltre in figura 6.3 come si comporta il drone durante un cambio di direzione, come si nota nel frame 2. Vediamo nel frame 3 che la rotazione avviene con un leggero ritardo e infatti il drone è spostato più verso la parte destra del percorso, ma poi nel frame 4 e 5 come inizia a spostarsi verso il centro grazie all'algoritmo di assestamento.

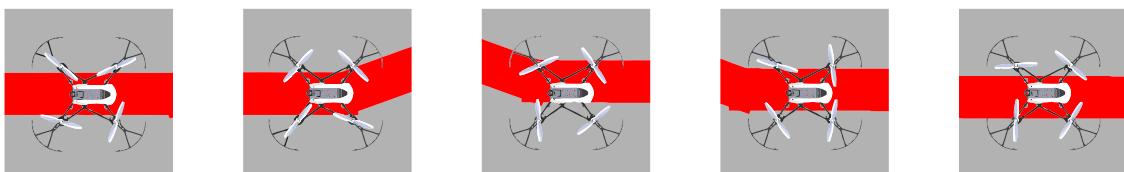


Figura 6.3: Fase di cambiamento della direzione

Infine in figura 6.4 viene rappresentata la fase di atterraggio alla fine del percorso. Una volta individuata dall'algoritmo di Image Processing la fine del percorso nel frame 1, il drone decelera nel frame 2 e 3 per poi iniziare a scendere di quota e atterrare correttamente all'interno del cerchio finale nel frame 4.

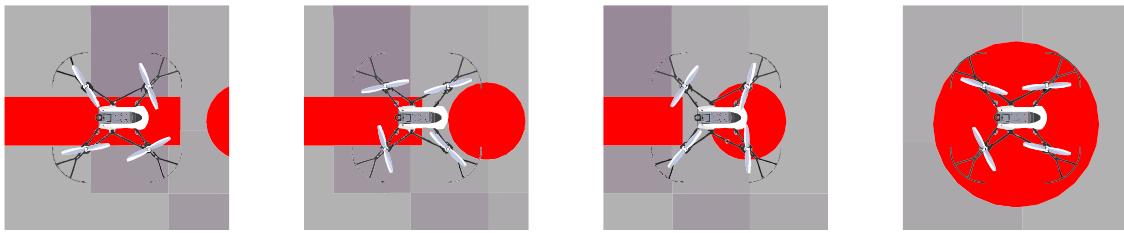


Figura 6.4: Fase di fine percorso e atterraggio

6.1 Segnale di riferimento

Il primo blocco del modello Simulink è il generatore del segnale di riferimento. In figura 6.5 viene rappresentato il segnale di Yaw di riferimento generato dalla trasformata di Hough e il segnale di Yaw reale del drone. Possiamo notare come quest'ultimo segue perfettamente il riferimento, avevamo visto in figura 3.19, senza alcun overshoot tipico dei controllori PID, ma a discapito di un leggero ritardo. Questo risultato è una conseguenza del comportamento che avevamo deciso di dare al controllore durante la taratura, modificando opportunatamente i suoi parametri. Inoltre è facile individuare dal grafico le tre variazioni di direzione del percorso. In particolare avremo un prima variazione di circa -48° , poi una di circa $+27^\circ$ e infine un'ultima variazione di circa -78° . Questo corrisponde al tracciato in figura 6.2 dove ogni angolo di Yaw è riferito al sistema di riferimento inerziale.

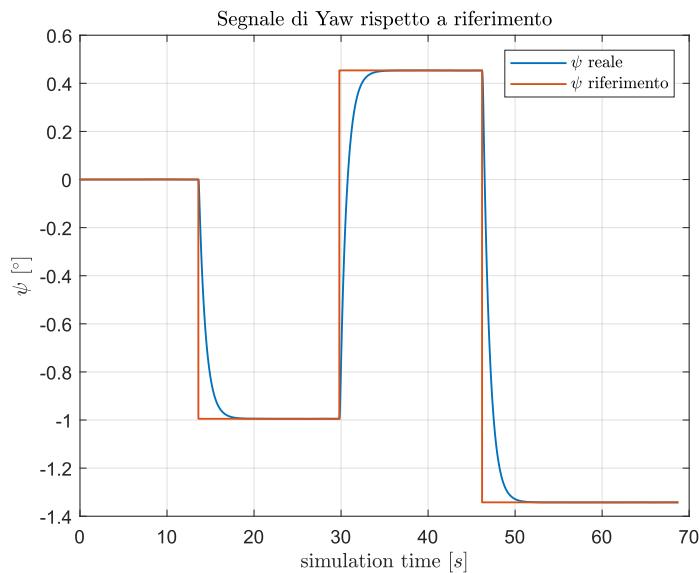


Figura 6.5: Segnale di Yaw e rispettivo riferimento

6.2 Velocità dei propulsori

Analizziamo il comportamento dei propulsori dal grafico in figura 6.6 durante la simulazione. Innanzitutto i motori 3 e 4 hanno valori positivi di RPM perché ruotano in senso antiorario, che per convenzione impostiamo positivo, mentre i motori 1 e 2, girando in senso orario, hanno RPM negativi. Per quanto riguarda i grafici dei motori, possiamo notare (anche riferendoci alle immagini in cui viene mostrato come ottenere i movimenti) che abbiamo un picco iniziale, dovuto alla grossa accelerazione necessaria per far alzare di quota il drone. Successivamente notiamo altri 3 balzi di valori, dovuti alle 3 rotazioni nei 3 angoli di percorso. Notiamo come le coppie in diagonale, le cui eliche ruotano alla stessa direzione hanno valore pressoché coincidente (permette di ruotare con Yaw), quindi se verso sinistra avremo un aumento dei RPM dei motori 3 e 4, mentre se verso destra un aumento negli RPM dei motori 1 e 2. Confrontandoci quindi con il percorso 6 in Appendice, notiamo che il primo netto cambio di direzione è verso sinistra, e infatti il secondo picco nei grafici coinvolge un aumento dei motori 3 e 4, mentre il successivo cambio di direzione, verso destra, induce un picco nei motori 1 e 2. È importante inoltre che in questi istanti anche i due motori non direttamente coinvolti nello Yaw subiscono una diminuzione in modulo, ovvero diminuiscono gli RPM (attenzione ai valori sull'asse delle ordinate). Questo perché il valore totale del momento, che deve contrapporsi alla forza di gravità deve rimanere lo stesso, e poiché aumentiamo la coppia di due motori, dovremo necessariamente diminuire la coppia degli altri due, in modo da far rimanere il drone alla stessa altezza dal terreno.

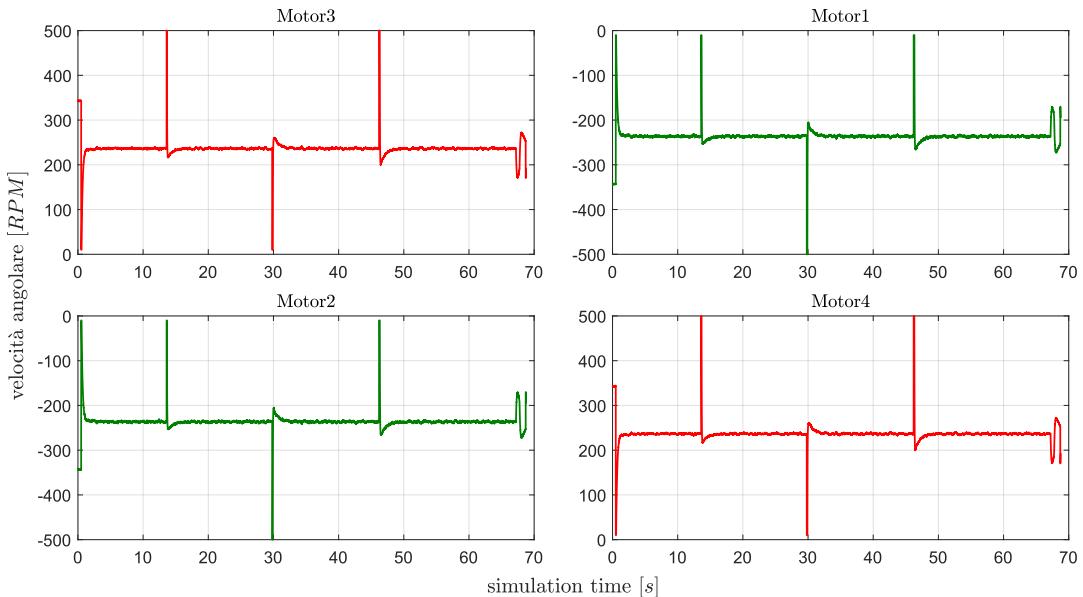


Figura 6.6: Velocità rotazione dei propulsori [RPM]

6.3 Velocità del drone

Analizziamo il grafico in figura 6.7 rappresentante le velocità del drone rispetto al sistema di riferimento inerziale. Possiamo notare come la dz , ovvero la velocità rispetto all'asse Z , ha due picchi. Il primo, all'inizio dovuto all'alzata di quota del drone, mentre l'ultimo, a fine della simulazione, dovuto all'abbassamento del drone all'interno del cerchio finale. Durante il resto delle simulazioni, la velocità dz si mantiene pressoché nulla, poiché il drone deve rimanere alla stessa quota. È possibile notare però tre piccole variazioni negative (ossia dei leggeri incrementi di velocità verso l'alto) i cui tempi coincidono esattamente con i tre cambi di direzione ottenuti con una rotazione intorno all'asse Z . Questo può essere dovuto sia al comportamento del controllore che fornisce una leggera sovraelongazione, sia a delle approssimazioni fatte durante il calcolo di attuazione, per cui l'aumento delle coppie simmetriche di propulsori e la conseguente diminuzione delle altre due, non bilancia perfettamente la forza di gravità rivolta verso il basso.

Per quanto riguarda invece le velocità dx e dy , notiamo fino a circa $15s$, tempo nel quale abbiamo il primo cambio di direzione, la dx , a meno di un rapido picco iniziale, si mantiene costante a 0.1 mentre la dy si mantiene nulla. Questo poiché nella prima parte di percorso, la traiettoria è solo lungo l'asse X del sistema di riferimento inerziale. Successivamente le due componenti variano parecchio a causa sia degli angoli di direzione del percorso rispetto al sistema di riferimento fisso, sia a causa della logica di assestamento.

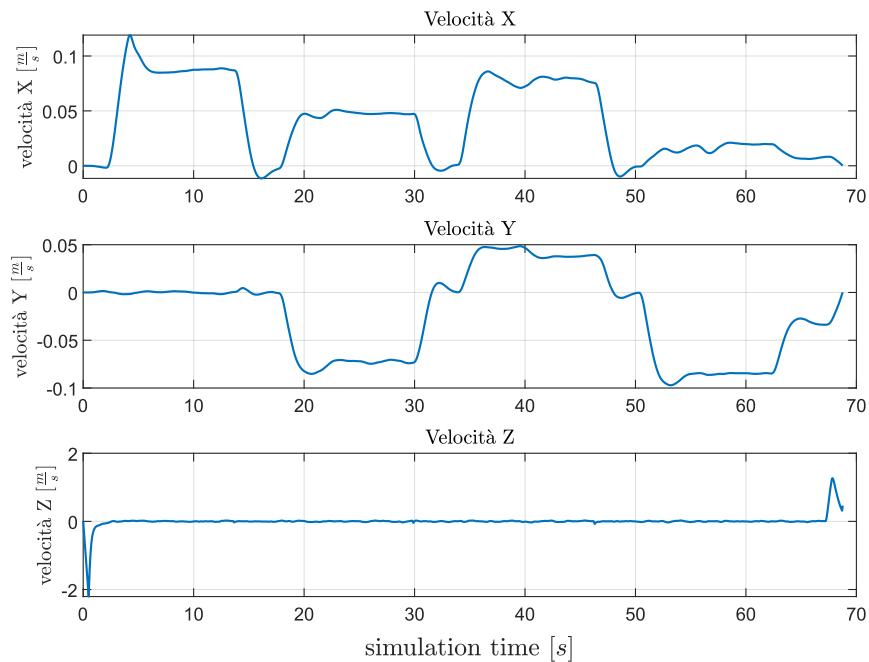


Figura 6.7: Velocità rispetto a ognuno dei 3 assi

6.4 Confronto tra valori stimati e valori reali

In questa sezione mostriamo i confronti tra i valori reali ottenuti mediante il modello non lineare dell'intero sistema e i valori stimati ottenuti mediante l'algoritmo dello State Estimator. In figura 6.8 vengono rappresentate la traiettoria del drone reale e la traiettoria stimata. In figura 6.9 viene calcolata l'errore tra i due tragitti come distanza tra i corrispettivi punti. Possiamo notare come con i cambi di direzione l'errore viene incrementato, generando un certo offset tra le due traiettorie. In

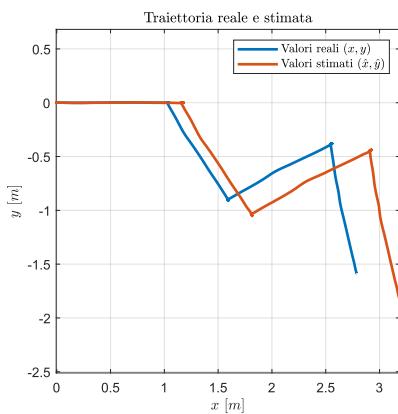


Figura 6.8: Traiettoria

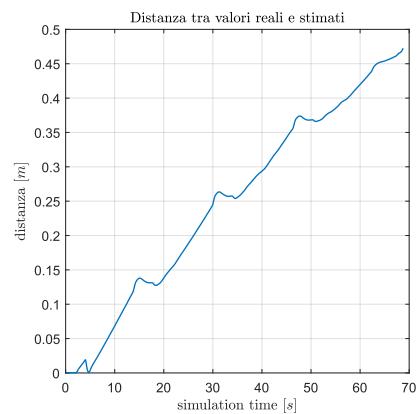


Figura 6.9: Distanza

figura 6.10 viene rappresentato il valore dell'angolo di Yaw reale e stimato rispetto al sistema di riferimento inerziale. In questo caso notiamo come la stima risulti molto accurata, con la stima che segue quasi perfettamente il valore reale. Infine

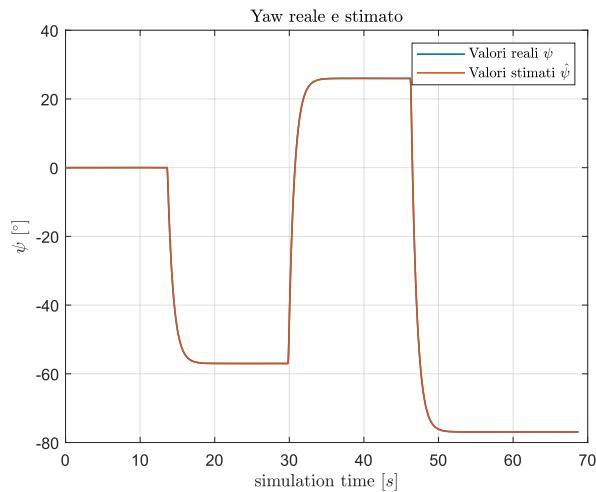


Figura 6.10: Yaw reale e stimato

nelle figure 6.11 6.12 rappresentiamo i valori di velocità reali e stimati. Abbiamo potuto notare da diverse simulazioni opportunamente costruite, che, mentre il drone prosegue linearmente effettuando solo movimenti traslazionali l'errore è basso, tuttavia durante le rotazioni del quadricottero dovute ai cambi di direzione, lo stimatore non è più in grado di generare una stima corretta della velocità, e l'errore tra il valore reale e quello stimato è molto alto.

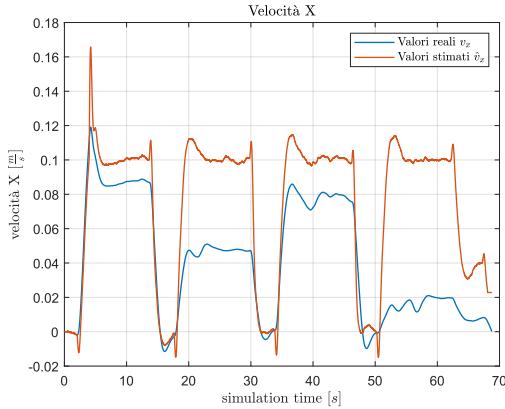


Figura 6.11: Velocità X

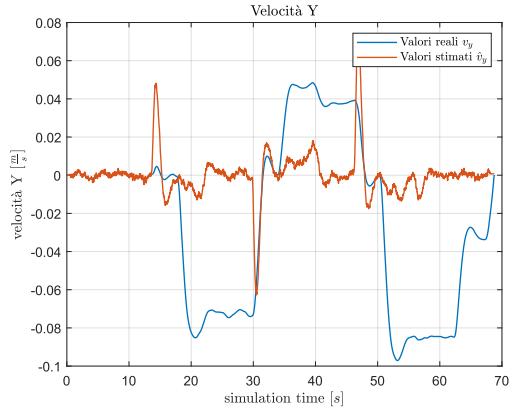


Figura 6.12: Velocità Y

6.5 Traiettorie

Mostriamo in quest'ultima sezione le traiettorie percorse dal drone rispetto al tempo. In figura 6.13 vengono mostrate le traiettorie reali della posizione X , Y e Z rispetto al tempo. In particolare è stato preso il valore assoluto per la posizione rispetto all'asse Z , quindi otteniamo un valore sempre positivo, che si differenzia da quanto trovato per la velocità rispetto allo stesso asse. Notiamo come la posizione X sia completamente crescente, in quanto il nostro percorso vada continuamente in avanti, mentre la posizione Y sia sempre negativa, in quanto rispetto all'origine (la partenza) il percorso si trova tutto alla sua destra. Infine rispetto alla posizione Z è facile osservare che all'inizio il minidrone si porti alla quota di 1.1 m durante la fase di sollevamento e torni a 0 m una volta raggiunto il cerchio finale nella fase di atterraggio, mantenendosi per tutto il tracciato alla quota costante di 1.1 m, per permettere alla camera di acquisire al meglio il percorso e mantenere l'algoritmo di image processing uniforme. In figura 6.15 abbiamo riportato la traiettoria in 3D, dove l'asterisco verde rappresenta la partenza, mentre il pallino rosso rappresenta il cerchio finale su cui atterrare. In figura 6.14 viene invece mostrata la traiettoria reale vista dall'alto. In entrambe le figure è possibile infine notare alcune leggere oscillazioni, dovute all'assestamento del drone per portarsi al centro del percorso individuato.

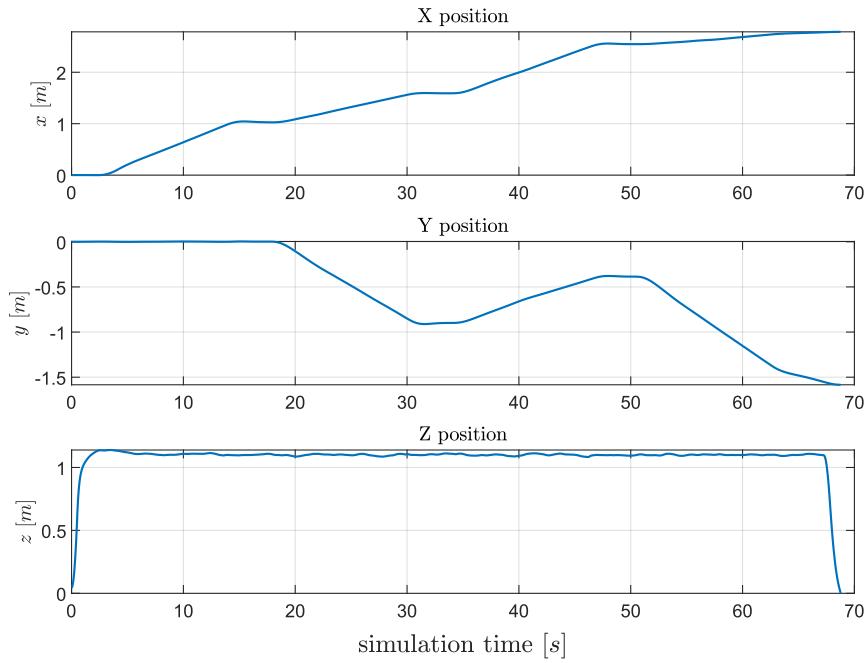


Figura 6.13: Posizione rispetto a ognuno dei 3 assi

Una volta terminato il nostro algoritmo di acquisizione e controllo della traiettoria e dopo aver testato attraverso la simulazione il comportamento dinamico rispetto a tutti i percorsi nella figura in Appendice C, il codice può finalmente essere generato. Simulink, una volta generato automaticamente il modello linare del sistema, convertirà quindi tutto lo schema del blocco **Flight Control System** in codice C++ che potrà essere inserito all'interno del drone tramite una chiavetta USB e testato nella realtà.

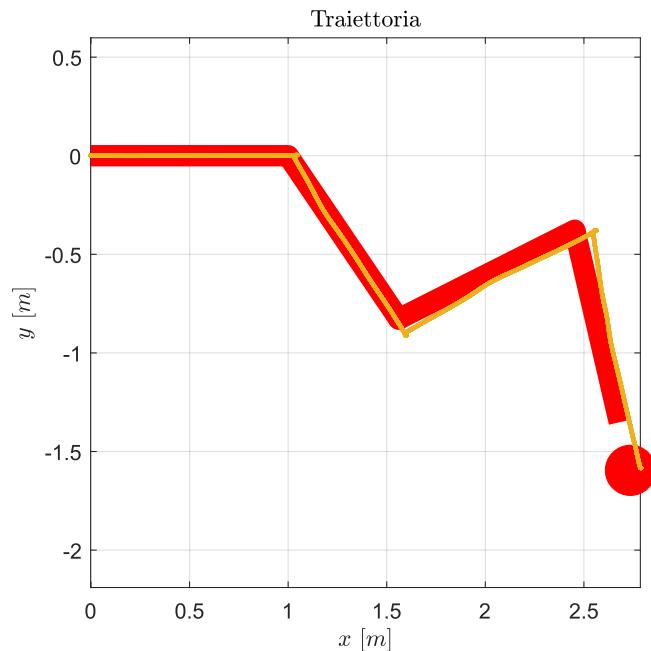


Figura 6.14: Traiettoria reale dall’alto [m]

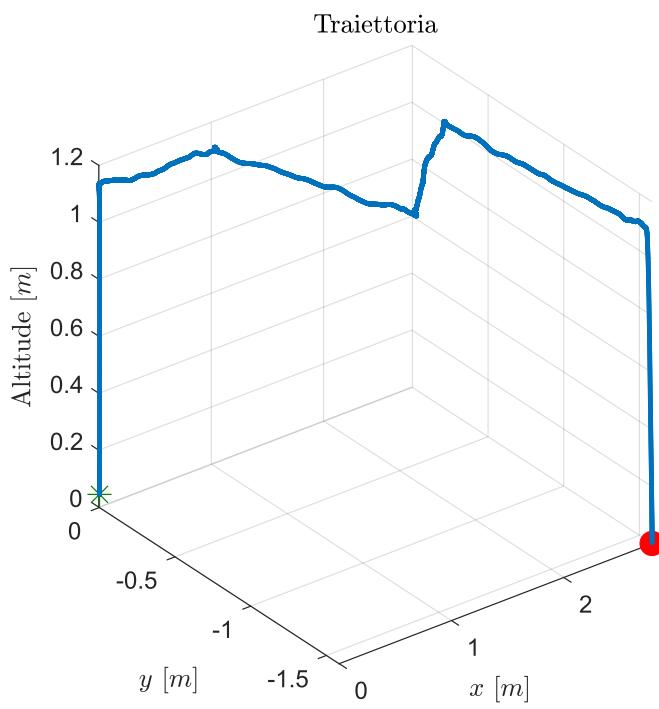


Figura 6.15: Traiettoria reale 3D [m]

Capitolo 7

Conclusioni

Riassumiamo in quest'ultimo capitolo tutto ciò che è stato analizzato in questa tesi. Partendo quindi da una descrizione intuitiva del funzionamento dinamico del drone, in particolare capendo come si è in grado di ottenere ognuno dei possibili movimenti traslazionali e rotazionali, siamo passati alla modellazione matematica in forma di state space della dinamica, ossia dello studio del movimento considerando anche le cause che l'hanno provocato, sfruttando in un caso reale nozioni di meccanica introdotte in altri corsi. Abbiamo quindi ottenuto un sistema dinamico di 12 equazioni in 12 incognite che permette di descrivere completamente la dinamica del quadricottero. Successivamente abbiamo introdotto la competizione organizzata da Mathworks, la MathWorks Minidrone Competitions, che ha permesso di approfondire il funzionamento dell'ambiente Simulink e che ha stimolato nel cercare e perfezionare continuamente un algoritmo in grado di far seguire autonomamente al drone un percorso di uno specifico colore. Abbiamo quindi analizzato nel capitolo 4 le tecniche di image processing usate nel modello Simulink, tra cui il metodo Canny, per identificare e tracciare i bordi nell'immagine catturata dal sensore, e la trasformata di Hough, per calcolare l'angolo durante i cambi di direzione. Successivamente abbiamo mostrato il nostro algoritmo per permettere al drone di muoversi lungo il percorso, modificando in opportunamente in real-time il riferimento della sua posizione e del suo orientamento. Infine abbiamo descritto la logica utilizzata per l'assestamento della traiettoria, in modo da permettere al minidrone di riportarsi verso il centro del percorso se, soprattutto durante i cambi di direzione durante le rotazioni, si trovava più all'esterno del tracciato. Nonostante la soluzione sembri concettualmente corretta, i risultati delle simulazioni hanno mostrato alcuni problemi, come ritardi nei cambi di direzione e nell'azione di assestamento anche una volta raggiunto il centro del percorso. Questi problemi possono essere causati dalle frequenze sia dei sensori montati sul drone, sia del sample time delle varie funzioni nel modello. Tuttavia queste analisi sono state volutamente trascurate. Un ultimo aspetto che può essere migliorato è lo schema con il filtro di Kalman per la

stima dello stato del sistema. Infatti abbiamo potuto notare come le stime della posizione del drone accumulino un errore, che diventa via via maggiore con il passare del tempo; e che le stime delle velocità, che in assenza di rotazioni del drone sembrano comportarsi correttamente, in seguito ad una qualunque rotazione del drone rispetto all'asse Z , comportano un errore irrimediabile e non più trascurabile.

Appendice A

Valore Parametri

All'avvio del nostro progetto in MATLAB, verranno eseguiti automaticamente alcuni script che inizializzeranno tutte le variabili di progetto, i bus, il sampling time e la scelta dello specifico percorso. Lo script probabilmente più importante è chiamato

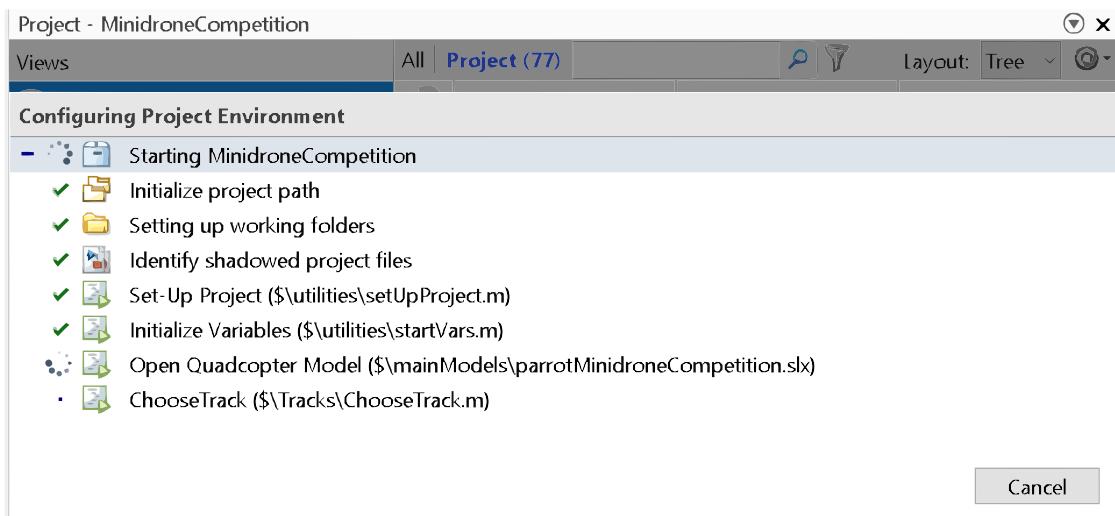


Figura A.1: Avvio del progetto

startVars.m all'interno della cartella *tasks* che definisce tutti i parametri che verranno utilizzati nel progetto. Vengono riportati sotto i più importanti per quanto riguarda il veicolo:

- massa $m: 0.0632[kg]$
- momenti d'inerzia $\mathbf{I}: [0.0000582857, 0.0000716914, 0.0001][kg \cdot m^2]$
- raggio dei propulsori $r = 0.033[m]$;
- coefficiente di proporzionalità tra Thrust e la velocità angolare $b: 4.72 \times 10^{-8}$

- coefficiente di drag k : 1.1393×10^{-10}

e per quanto riguarda i parametri ambientali:

- Temperatura dell'aria: $283[K]$
- Pressione atmosferica: $1[atm]$
- Densità dell'aria: $1.184 \left[\frac{\text{kg}}{\text{m}^3} \right]$
- Velocità del suono: $340 \left[\frac{\text{m}}{\text{s}} \right]$
- Accelerazione di gravità: $[0,0,9.81] \left[\frac{\text{m}}{\text{s}^2} \right]$
- Campo magnetico: $[0,0,0][T]$

Infine viene specificato che i motori hanno range di funzionamento con velocità angolari $w \in [10,500]$.

Appendice B

Codice Grafici

Di seguito viene mostrato il codice utilizzato per plottare i grafici delle simulazioni:

```
1 close all
2 clc
3
4 % ATTENZIONE: prende dati dal progetto Simulink, quindi stessa directory.
5 % Inoltre i dati sono quelli che vengono acquisiti dopo la simulazione,
6 % quindi serve prima simulare il modello almeno una volta.
7
8 set(0, 'defaultFigureRenderer', 'painters') % per alta risoluzione
9
10 %% Plot in 3d valori REALI
11 % ATTENZIONE: metto abs() per la z perche' e' rivolta verso il basso (vedi sistema di
12 % riferimento drone)
13 positionY_NORMALIZZATO = positionY - 95;
14 positionX_NORMALIZZATO = positionX - 57;
15
16 plot3(positionX_NORMALIZZATO,positionY_NORMALIZZATO,abs(positionZ), 'LineWidth',2)
17 hold on
18 plot3(positionX_NORMALIZZATO(1,1),positionY_NORMALIZZATO(1,1),abs(positionZ(1,1)), 'Marker', '*', 'LineStyle', 'none', 'MarkerFaceColor', [1 0 0], 'MarkerSize',10, 'MarkerEdgeColor',[0, 0.5, 0])
19 hold on
20 plot3(positionX_NORMALIZZATO(end,1),positionY_NORMALIZZATO(end,1),abs(positionZ(end,1)), 'Marker', 'o', 'LineStyle', 'none', 'MarkerFaceColor', [1 0 0], 'MarkerSize',10, 'MarkerEdgeColor', 'red')
21 pbaspect([1 1 1])
22 grid on
23 xlabel('$x\$ [$m\$]', 'Interpreter', 'latex')
24 ylabel('$y\$ [$m\$]', 'Interpreter', 'latex')
25 zlabel('Altitude [$m\$]', 'Interpreter', 'latex')
26 title('Traiettoria', 'Interpreter', 'latex')
27
28 %% Plot in 2d SOVRAPPOSTO CON PATH dei valori REALI
29 figure
30 plot(yTrack, xTrack, 'color', 'r', 'LineWidth', 10);
31 hold on
32 plot(Ay.C, Ax.C, '.', 'color', 'r', 'MarkerSize', 80);
33
34 hold on
35 plot3(positionX_NORMALIZZATO,positionY_NORMALIZZATO,abs(positionZ), 'LineWidth',2)
36 grid on
37 axis equal
38 pbaspect([1 1 1])
39 xlabel('$x\$ [$m\$]', 'Interpreter', 'latex')
40 ylabel('$y\$ [$m\$]', 'Interpreter', 'latex')
41 zlabel('Altitude [$m\$]', 'Interpreter', 'latex')
42 title('Traiettoria', 'Interpreter', 'latex')
43
44 view(2) % per vedere plot 3d dall'alto (in 2D)
45
46 %% Plot in 2d dei vari casi valore REALE
47 figure
48 t = tiledlayout(3,1) % Combine multiple plots
49 t.Padding = 'compact';
50 t.TileSpacing = 'compact';
```

```

53 ax1 = nexttile;
54 plot(ax1,estimatedStates.time(1:end-1),positionX_NORMALIZZATO,'LineWidth',1)
55 grid on
56 ylabel('$x\text{ [m]}$','Interpreter','latex')
57 title('X position','Interpreter','latex')
58
59 ax2 = nexttile;
60 plot(ax2,estimatedStates.time(1:end-1),positionY_NORMALIZZATO,'LineWidth',1)
61 grid on
62 ylabel('$y\text{ [m]}$','Interpreter','latex')
63 title('Y position','Interpreter','latex')
64
65 ax3 = nexttile;
66 plot(ax3,estimatedStates.time(1:end-1),abs(positionZ),'LineWidth',1)
67 grid on
68 ylabel('$z\text{ [m]}$','Interpreter','latex')
69 title('Z position','Interpreter','latex')
70 xlabel(t,'simulation time [$s$]','Interpreter','latex')
71
72 %% Sovrapposizione plot tra valori REALI e valori STIMATI
73 figure
74 plot(positionX_NORMALIZZATO,positionY_NORMALIZZATO,'LineWidth',2)
75
76 hold on
77 plot(estimatedStates.signals.values(:,1),estimatedStates.signals.values(:,2),'LineWidth',2)
78 hold on
79 plot(estimatedStates.signals.values(1,1),estimatedStates.signals.values(1,2))
80 hold on
81 plot(estimatedStates.signals.values(end,1),estimatedStates.signals.values(end,2))
82 grid on
83 axis equal
84 pbaspect([1 1 1])
85 xlabel('$x\text{ [m]}$','Interpreter','latex')
86 ylabel('$y\text{ [m]}$','Interpreter','latex')
87 legend('Valori reali $(x,y)$','Valori stimati $\hat{(x,y)}$','Interpreter','latex')
88 title('Traiettoria reale e stimata','Interpreter','latex')
89
90 %% Errore tra valori REALI e valori STIMATI
91 % le due matrici devono avere la stessa dimensione, quindi bisogna eliminare l'ultima riga per
92 % i valori stimati
93
94 distance_between_two_points = sqrt((positionX_NORMALIZZATO -
95 estimatedStates.signals.values(1:end-1,1)) ^ 2 + (positionY_NORMALIZZATO -
96 estimatedStates.signals.values(1:end-1,2)) ^ 2);
97
98 figure
99 plot(estimatedStates.time(1:end-1),distance_between_two_points,'LineWidth',1)
100 grid on
101 pbaspect([1 1 1])
102 xlabel('simulation time [$s$]','Interpreter','latex')
103 ylabel('distanza [m]','Interpreter','latex')
104 title('Distanza tra valori reali e stimati','Interpreter','latex')
105
106 %% Velocita X
107 figure
108 plot(estimatedStates.time(1:end-1),velocitaX,'LineWidth',1)
109 hold on
110 plot(estimatedStates.time,estimatedStates.signals.values(:,7),'LineWidth',1)
111 grid on
112 xlabel('simulation time [$s$]','Interpreter','latex')
113 ylabel('velocit\`a X ['$\frac{m}{s}$]','Interpreter','latex')
114 legend('Valori reali $v_x$','Valori stimati $\hat{v}_x$','Interpreter','latex')
115 title('Velocit\`a X','Interpreter','latex')
116
117 %% Velocita Y
118 figure
119 plot(estimatedStates.time(1:end-1),velocitaY,'LineWidth',1)
120 hold on
121 plot(estimatedStates.time,estimatedStates.signals.values(:,8),'LineWidth',1)
122 grid on
123 xlabel('simulation time [$s$]','Interpreter','latex')
124 ylabel('velocit\`a Y ['$\frac{m}{s}$]','Interpreter','latex')
125 legend('Valori reali $v_y$','Valori stimati $\hat{v}_y$','Interpreter','latex')
126 title('Velocit\`a Y','Interpreter','latex')
127
128 %% Angolo Yaw
129 % Attenzione: tutti i valori di angoli trovati sono rispetto al sistema di
130 % riferimento inerziale, non rispetto al drone
131 Yaw_deg = rad2deg(Yaw); % trasformo rad to deg
132 YawHAT_deg = rad2deg(estimatedStates.signals.values(:,4));
133
134 figure

```

```

133 plot(estimatedStates.time(1:end-1),Yaw_deg,'LineWidth',1)
134 hold on
135 plot(estimatedStates.time,YawHAT_deg,'LineWidth',1)
136 grid on
137 xlabel('simulation time [$$]','Interpreter','latex')
138 ylabel('$\psi^{\circ}$','Interpreter','latex')
139 legend('Valori reali $\psi$','Valori stimati $\hat{\psi}$','Interpreter','latex')
140 title('Yaw reale e stimato','Interpreter','latex')
141
142 %% Motori
143 figure
144 t = tiledlayout(2,2) % Combine multiple plots
145 t.Padding = 'compact';
146 t.TileSpacing = 'compact';
147
148 ax3 = nexttile;
149 plot(ax3,estimatedStates.time(1:end),Motor1,'r','LineWidth',1)
150 grid on
151 title('Motor3','Interpreter','latex')
152 xticks(0:10:70);
153 xlim([0 70])
154
155 ax1 = nexttile;
156 plot(ax1,estimatedStates.time(1:end),Motor2,[0, 0.5, 0],'LineWidth',1)
157 grid on
158 title('Motor1','Interpreter','latex')
159 xticks(0:10:70);
160 xlim([0 70])
161
162 ax2 = nexttile;
163 plot(ax2,estimatedStates.time(1:end),Motor4,[0, 0.5, 0],'LineWidth',1)
164 grid on
165 title('Motor2','Interpreter','latex')
166 xticks(0:10:70);
167 xlim([0 70])
168
169 ax4 = nexttile;
170 plot(ax4,estimatedStates.time(1:end),Motor3,'r','LineWidth',1)
171 grid on
172 title('Motor4','Interpreter','latex')
173 xticks(0:10:70);
174 xlim([0 70])
175 xlabel(t,'simulation time [$$]','Interpreter','latex')
176 ylabel(t,'velocit\`a angolare [RPM]','Interpreter','latex')
177
178 %% Velocita reali
179 figure
180 t = tiledlayout(3,1) % Combine multiple plots
181 t.Padding = 'compact';
182 t.TileSpacing = 'compact';
183
184 ax1 = nexttile;
185 plot(ax1,estimatedStates.time(1:end-1),velocitaX,'LineWidth',1)
186 grid on
187 ylabel('velocit\`a X [$\frac{m}{s}]','Interpreter','latex')
188 title('Velocit\`a X','Interpreter','latex')
189
190 ax2 = nexttile;
191 plot(ax2,estimatedStates.time(1:end-1),velocitaY,'LineWidth',1)
192 grid on
193 ylabel('velocit\`a Y [$\frac{m}{s}]','Interpreter','latex')
194 title('Velocit\`a Y','Interpreter','latex')
195
196 ax3 = nexttile;
197 plot(ax3,estimatedStates.time(1:end-1),velocitaZ,'LineWidth',1)
198 grid on
199 ylabel('velocit\`a Z [$\frac{m}{s}]','Interpreter','latex')
200 title('Velocit\`a Z','Interpreter','latex')
201 xlabel(t,'simulation time [$$]','Interpreter','latex')
202
203 %% Yaw reale e reference
204 figure
205 plot(estimatedStates.time(1:end-1),Yaw,'LineWidth',1)
206 hold on
207 plot(estimatedStates.time(1:end-1),YawReference,'LineWidth',1)
208 grid on
209 xlabel('simulation time [$$]','Interpreter','latex')
210 ylabel('$\psi$ reale','Interpreter','latex')
211 legend('$\psi$ reale','$\psi$ riferimento','Interpreter','latex')
212 title('Segnali di Yaw rispetto a riferimento','Interpreter','latex')

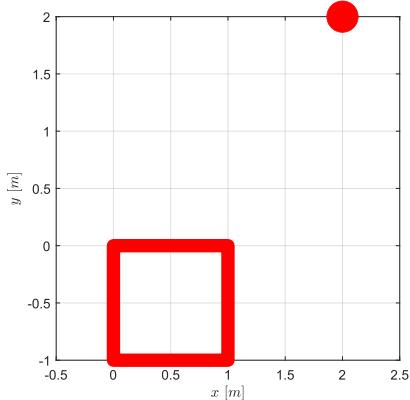
```

Listato B.1: Codice grafici simulazioni

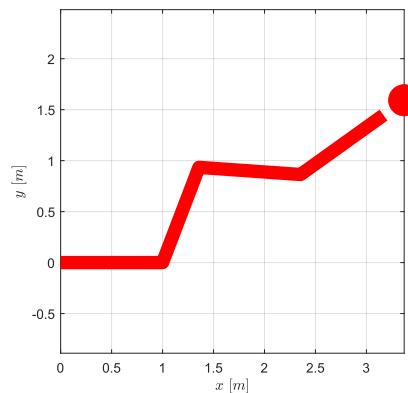
Appendice C

Tracks Simulazione

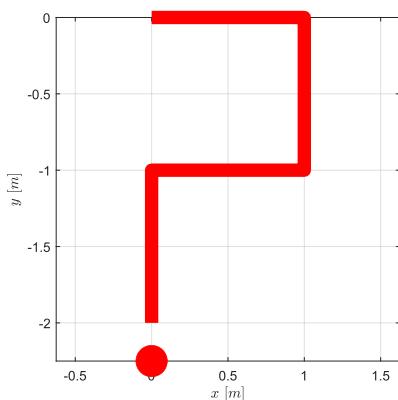
Mostriamo l'elenco dei percorsi che sono stati testati durante la fase di simulazione.



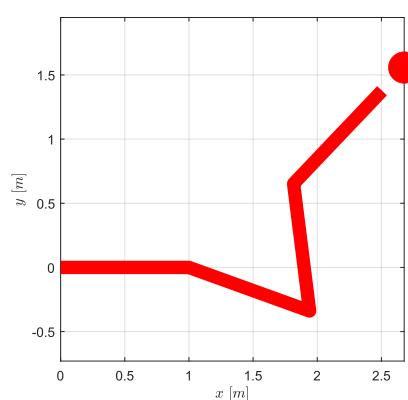
Track 1



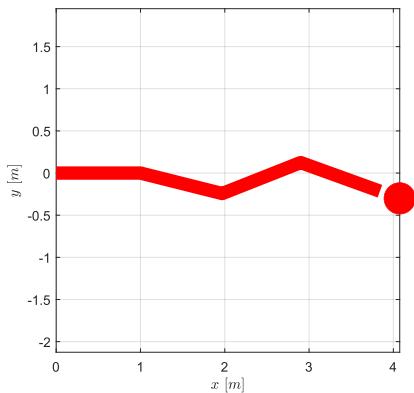
Track 2



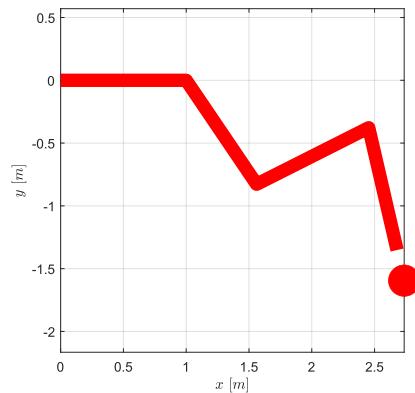
Track 3



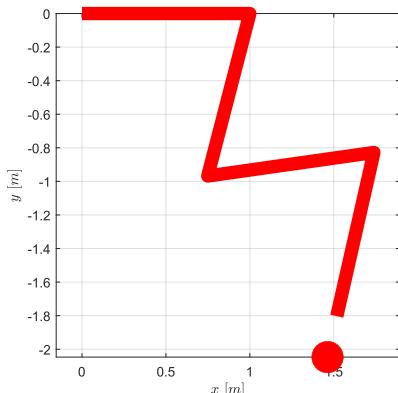
Track 4



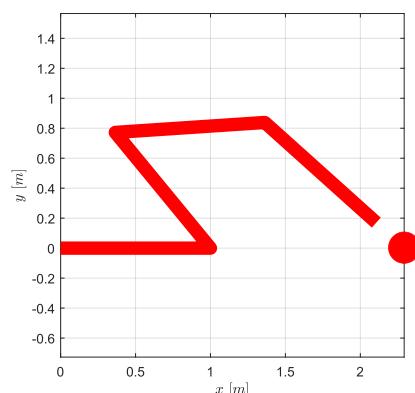
Track 5



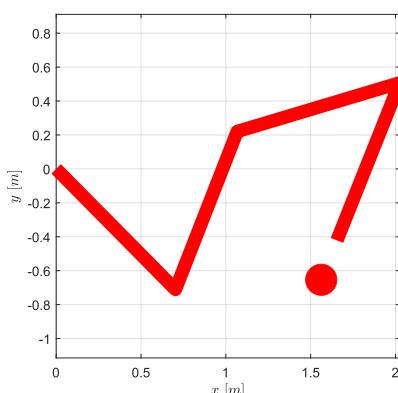
Track 6



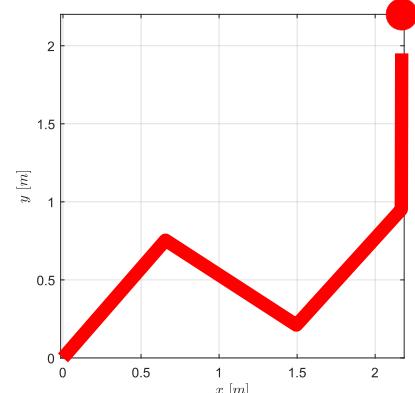
Track 7



Track 8



Track 9



Track 10

Bibliografia

- [1] Song Ho Ahn. *Example of 2D Convolution*. URL: http://www.songho.ca/dsp/convolution/convolution2d_example.html.
- [2] Aydin Ayanzadeh. *Canny edge detection method*. URL: <https://medium.com/@a.ayanzadeh/canny-edge-detection-method-23a23b282ac0>.
- [3] Eduardo Fuentevilla Blanco. «Control strategies evaluation forstabilizing a quadcopter docking on a side wall». Master's Thesis. KTH Royal Institute of Technology, 2019, pp. 25–36. URL: <https://www.diva-portal.org/smash/get/diva2:1366443/FULLTEXT01.pdf>.
- [4] Paolo Ceppi. «Model-based Design of a Line-tracking Algorithm for a Low-cost Mini Drone through Vision-based Control». Master's Thesis. Politecnico di Torino, 2020. URL: <https://webthesis.biblio.polito.it/16018/1/tesi.pdf>.
- [5] Gabriele Danesi. *I Canali Colore RGB e l'Iistogramma*. URL: <https://gabrieledanesi.com/blog/?canali-colore-rgb-istogramma>.
- [6] Agostino De Marco e Domenico P. Coiro. «Orientamento del velivolo e trasformazione di assi». Appunti. Università degli Studi di Napoli, 2017, pp. 4–12. URL: http://wpage.unina.it/agodemar/DSV-DQV/DSV-DQV_Quaderno_2.pdf.
- [7] Brian Douglas. *Drone Simulation and Control*. URL: https://www.youtube.com/playlist?list=PLn8PRpmsu08o0LBVYYIwwN_nvuyUqEjrj.
- [8] Andrew Gibiansky. *Quadcopter Dynamics, Simulation, and Control*. URL: <https://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf>.
- [9] Marco Giordani. «Progettazione di un sistema di “cruise control” con implementazione MATLAB/Simulink». Bachelor's Thesis. Università degli Studi di Padova, 2013, pp. 13–22. URL: http://tesi.cab.unipd.it/44164/1/TESI_Marco_Giordani.pdf.
- [10] CB Insights. *38 Ways Drones Will Impact Society: From Fighting War To Forecasting Weather, UAVs Change Everything*. URL: <https://www.cbinsights.com/research/drone-impact-society-uav>.

BIBLIOGRAFIA

- [11] Socret Lee. *Lines Detection with Hough Transform*. URL: <https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549>.
- [12] MathWorks. *MathWorks Minidrone Competition*. URL: <https://it.mathworks.com/videos/series/mathworks-minidrone-competition.html>.
- [13] MathWorks. *What Is Image Segmentation? 3 things you need to know*. URL: <https://it.mathworks.com/discovery/image-segmentation.html>.
- [14] Robert Fisher e Simon Perkins e Ashley Walker e Erik Wolfart. *Gaussian Smoothing*. URL: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm>.
- [15] Charles Tytler. *Modeling Vehicle Dynamics Euler Angles*. URL: <https://charlestytler.com/modeling-vehicle-dynamics-euler-angles/>.
- [16] Charles Tytler. *Modeling Vehicle Dynamics Quadcopter Equations of Motion*. URL: <https://charlestytler.com/quadcopter-equations-motion/>.
- [17] Wikipedia. *Canny edge detector*. URL: https://en.wikipedia.org/wiki/Canny_edge_detector.