Universita Politecnica delle Marche

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Tesi di laurea triennale

PROGETTO E SVILUPPO DI ALGORITMI DI CONTROLLO PER DRONI

Design and development of control algorithms for drones

Relatore:
Gianluca Ippoliti

Correlatore:
Giuseppe Orlando

Candidato:
Alex Giacomini
Matricola S1076647

Anno Accademico 2019 - 2020

"Alla mia famiglia."

Abstract

Questa tesi contiene l'elaborato riguardante la progettazione di un sistema di controllo per la traiettoria di un mini drone. Partendo dallo studio del modello matematico, che permette l'analisi fisica e dinamica del quadricottero presentato nelle pagine seguenti, si ricavano le equazioni del sistema che ne regolano il movimento. Dopodiché si passa all'analisi della competizione che ha come obiettivo l'inseguimento di un percorso da parte del mini drone. Infine si arriva alla descrizione dell'algoritmo realizzato nell'ambiente di sviluppo Simulink, con annesse le simulazioni e i test di verifica.

Indice

Elenco delle figure

Elenco delle abbreviazioni

APR	Aeromobile a Pilotaggio Remoto
RGB	Red Green Blu (definition)
HSV	Hue Saturation Value
LiPo	Accumulatore litio-polimero
RC	Radiocomandato
DOF	Degrees of freedom
BW Image	Black White Image

Capitolo 1

Introduzione

1.1 Droni - APR

I droni, elementi di studio di questa tesi, sono un affascinante invenzione, nata da una brillante intuizione che aveva come obiettivo la ricerca di strumenti per il supporto delle azioni compiute dall'essere umano. In particolare, può essere sicuramente considerato uno straordinario risultato, in quanto simboleggia la perseveranza dell'essere umano nel cercare di espandere sempre più i propri orizzonti, andando oltre quello che c'é di concreto e provando a sfruttare al meglio la vastità di cielo che ci circonda. Infatti risulta essere proprio questo scenario a fare da base principale per l'evoluzione tecnologica degli ultimi vent'anni, e che sicuramente otterrà sempre più importanza nelle vite di ciascuno di noi in futuro; basti guardare i risultati ottenuti dalle grandi compagnie aeree, piuttosto che agenzie di programmi spaziali come la NASA e via dicendo. Nel caso di studio trattato in questa tesi si considerano come anticipato gli APR. I primi prototipi furono realizzati per scopo militare, al fine di trasportare carichi esplosivi , con forme più somiglianti a quelle di un pallone che alle sembianze assunte dai droni che si trovano in circolazione oggi. Ma dall'inizio del nuovo millennio questo tipo di tecnologia é poi esplosa, ampliando i propri campi d'applicazione ed espandendosi in molteplici settori. Si passa dalla consegna di pacchi direttamente alla propria abitazione, all'impiego in ambito cinematografico. Ma la parte di principale sviluppo e beneficio, riguarda il loro utilizzo nell'ambito della telemedicina, passando per l'analisi ambientale e il supporto agricolo, fino

ad arrivare all'ambito industriale e di lavoro. Quindi possono essere considerati un grande successo dato il loro utilizzo per il monitoraggio della sicurezza, quanto il loro utilizzo per scopo ricreativo. Come ultimo aspetto ma non di minore importanza, bisogna valutare gli aspetti economici e di inquinamento, che risultano essere molto inferiori rispetto a strumenti utilizzati in passato, ma anche aspetti di sicurezza civile. Quelli in commercio si differenziano per le loro varie caratteristiche, ad esempio dimensione, forma, numero di motori e peso. In questa trattazione si é utilizzata una specifica categoria molto diffusa, quella dei mini droni, In particolare il mini drone *Mambo* dell'azienda francese *Parrot*. Di seguito é rappresentato il *Parrot Mambo Fly* Fig. 1.1.

Figura 1.1: Parrot Mambo Fly

1.1.1 Parrot Mambo Fly

La peculiarità di questi ultimi é la dimensione ridotta pari a 18×18 cm e la maneggevolezza, senza dimenticare la possibilità di essere programmato attraverso Matlab e Simulink scaricando determinati pacchetti di supporto. É dotato di quattro propulsori collegati opportunamente a 4 eliche rotanti, una per ciascun motore, con un peso complessivo che si aggira intorno ai 60 g. Queste caratteristiche descritte lo fanno rientrare nella categoria dei mini droni. Come componenti aggiuntivi, questa tipologia possiede 4 sensori, distribuiti equamente tra la parte superiore e inferiore dello stesso. I due sensori che si trovano sul dorso del mini drone, sono un sensore di pressione e un'IMU. Il primo rileva appunto la pressione, mentre l'IMU, formata

da accelerometro a 3 assi e giroscopio a 3 assi, misura le accelerazioni lineari e le velocità angolari. Per quanto riguarda i sensori che si trovano sulla parte inferiore, uno di essi é ad ultrasuoni, con obiettivo la misura della distanza dal suolo, ottenuta basandosi sulla misura di un intervallo temporale tra l'istante d'invio di un segnale ad alta frequenza verso il terreno e quello in cui viene ricevuto indietro dallo stesso, dopo aver raggiunto il suolo, con un'altezza massima di 4 m. La telecamera, sensore su cui si basa parte rilevante dell'operato, lavora a 60 fps e si occupa della stima della velocità e del movimento, come si vedrà nei capitoli successivi. L'obbiettivo generale é quello di analizzare il modello matematico e fisico su cui tale drone si basa e successivamente implementare un algoritmo in grado di seguire correttamente il percorso casuale. Illustrazione dei componenti del *Parrot Mambo Fly* Fig. 1.2.
Per altre caratteristiche consultare [**1**].

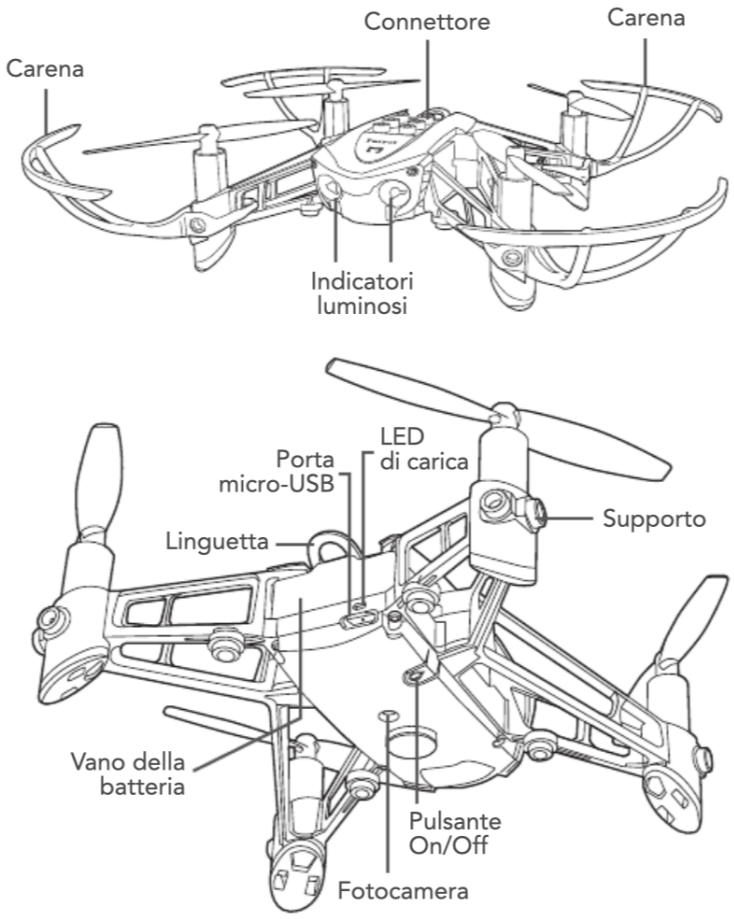

Figura 1.2: Parrot Mambo Components

1.2 Color Thresholder App

Questa applicazione Aggiuntiva di Matlab consente di modificare le sogli di colori delle immagini a colori, manipolando i componenti dei singoli colori attraverso determinati filtri, come RGB o HSV. Usando questa app, si può quindi creare una maschera di segmentazione per un'immagine a colori. In questo elaborato verrà impiegato per lo studio di un percorso che il mini drone dovrà inseguire autonomamente. Un esempio dell'app *Color Thresholder* Fig. 1.3.

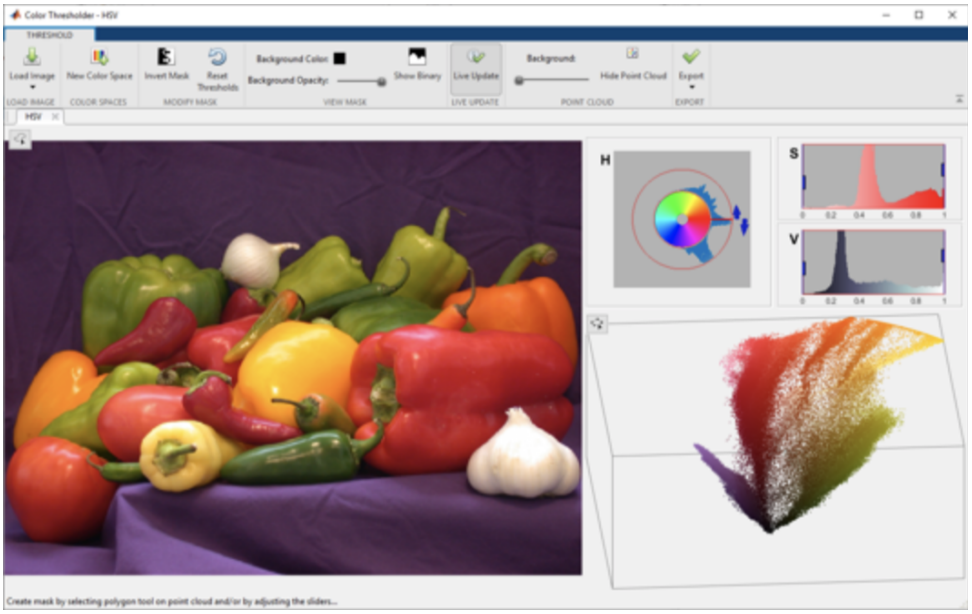

Figura 1.3: Color Thresholder App

1.3 Stateflow

Questo strumento sviluppato da MathWorks é un linguaggio a controlli logici implementato per la modellazione di sistemi con stati variabili nel tempo, Fig. 1.4. Il passaggio tra i diversi stati assumibili dal sistema deve essere sagomato attraverso delle transizioni mutualmente esclusive, che tipicamente separano stati con funzionalità completamente differenti tra loro, in modo da ottenere un codice chiaro e un funzionamento soddisfacente e pulito.

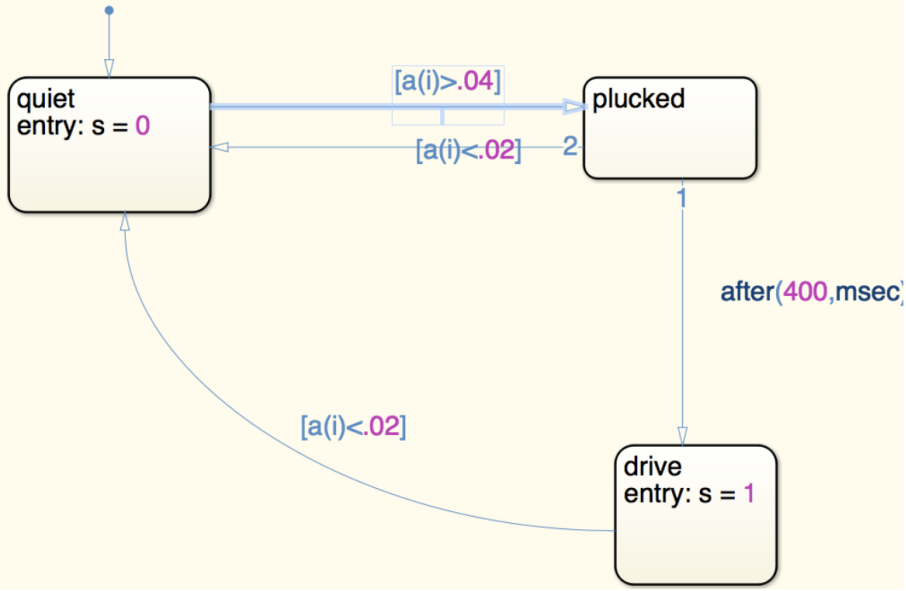

Figura 1.4: Esempio Stateflow

É quindi un linguaggio di programmazione del tipo stato-transizione [**2**], [**3**] che viene generalmente utilizzato per la progettazione di controllori discreti in sistemi ibridi, e in questo caso specifico viene utilizzato per la modellazione dell' inseguimento di un percorso variabile che lo specifico mini drone descritto in precedenza deve compiere senza alcuna azione di controllo telecomandata.

Capitolo 2

Modello del sistema

2.1 Componenti principali

2.1.1 Batteria power per Mambo

Batteria Power LiPo da 660 mAh, che garantisce un'autonomia di volo maggiore rispetto alle normali batterie, Fig. 2.1.

Figura 2.1: batteria LiPo Parrot Mambo Fly

Questa tipologia di batterie ricaricabili sono comunemente usate in ambito APR e nel mondo RC e sono un'opzione molto più valida rispetto alla combustione interna. Queste ultime possiedono tre caratteristiche principali che le rendono la scelta perfetta per gli RC:

- le batterie LiPo sono leggere e possono essere realizzate in qualsiasi forma e dimensione;
- le batterie LiPo sono in grado di immagazzinare grandi quantità di potenza in una piccola area;
- le batterie LiPo hanno elevato tasso di scarico elettrico per alimentare i motori più esigenti.

In altre parole, questo tipo di batterie consentono un alto accumulo di energia rapportata ad un peso limitato, fattore che ovviamente incide particolarmente in settori comprendenti diverse categorie e modelli di RC. Esistono al contempo alcuni difetti:

- le batterie LiPo sono ancora care rispetto le tradizionali NiCad e NiMH;
- sono tutt'ora una soluzione con un'autonomia limitata e allo stesso tempo una durata che varia dai 300 ai 500 cicli di ricarica, fino ad arrivare ad un massimo di 1000 cicli con un rigido rispetto delle regole di manutenzione;
- in rarissime ma possibili situazioni, se non vengono appropriatamente trattate, rischiano di scoppiare e/o prendere fuoco.

2.1.2 Motori A/C

Entrambe le tipologie di motori A e C, Fig. 2.2, sono di tipo Coreless, con dimensioni contenute, e progettate esclusivamente per i Parrot Mambo in modo da garantire alte prestazioni e una lunga durata di vita.

- $Motore\ A,\ (Anti - Clock)$ distinguibile dal filo di colore bianco, che gira in senso antiorario, e che assume la posizione anteriore destra e posteriore sinistra nella configurazione fisica.
- $Motore\ C,\ (Clock)$ distinguibile dal filo di colore rosso, che gira in senso orario, e che assume la posizione anteriore sinistra e posteriore destra nella configurazione fisica.

Figura 2.2: Rispettivamente motore A - motore C

Come anticipato in precedenza, queste due categorie di motori ricadono nella categoria Coreless, di cui si mostra una panoramica sintetica nelle prossime righe.

Motori Coreless

É un *motore elettrico a corrente continua*, che prevede il passaggio di corrente attraverso avvolgimenti di rame, che generano un campo magnetico intorno al rotore. Quest'ultimo elemento entra in movimento a seguito dell'interazione con un magnete permanente, che generando una forza negli avvolgimenti rotorici permette la rotazione della parte rotorica. Ma nello specifico, la caratteristica che contraddistingue questa categoria, come suggerisce il nome, è l'assenza del nucleo ferroso interno al rotore, infatti l'avvolgimento viene realizzato in una configurazione a cilindro , ruotando esternamente ad un magnete anch'esso di forma cilindrica, Fig. 2.3.

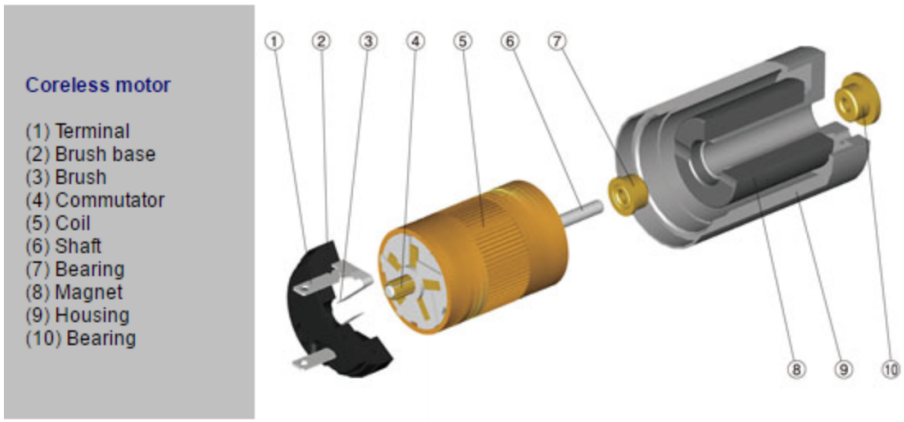

Figura 2.3: Struttura generale motore coreless

Nella figura precedente, è illustrata in particolare la struttura di un motore coreless specifico per droni. Tra i fattori che lo rendono migliore rispetto altre possibili categorie troviamo:

- Non presenta alcuna coppia residua e questo permette un controllo fluido del posizionamento e della velocità, con maggiore efficienza complessiva.

- Ha coppia e potenza estremamente elevate in relazione al peso e le dimensioni del motore.
- Presentano una relazione lineare tra carico e velocità, favorendo regolazioni precise e sensibili.
- Inerzia più bassa, che implica una dinamica di avviamento e arresto superiori.

2.1.3 Scheda madre

Dotata del più recente chipset SIP6 Parrot con ARM A9 da 800MHz. La scheda madre di questo minidrone utilizza Linux, e come illustrato precedentemente possiede diversi sensori e allo stesso tempo diversi attuatori, che insieme permettono il controllo del sistema, Fig. 2.4.

Figura 2.4: Vista anteriore e posteriore della scheda madre

2.2 Modello Strutturale

2.2.1 Configurazione fisica

Per quanto riguarda l'assetto del quadricottero, come suggerisce il nome, prevede una struttura formata da quattro braccia con collegati i quattro motori elencati in precedenza, con configurazione a croce, Fig. 2.5

Questa disposizione presenta 6 DOF, di cui 3 gradi di libertà dati dalle 3 coordinate cartesiane X,Y,Z e tre riguardanti le possibili rotazioni attorno a tali assi denominati angoli di *roll*, *pitch e yaw*, ovvero *rollio*, *beccheggio e imbardata*. Essendo però il drone formato da quattro attuatori, si possono controllare solo 4 DOF, ottenendo

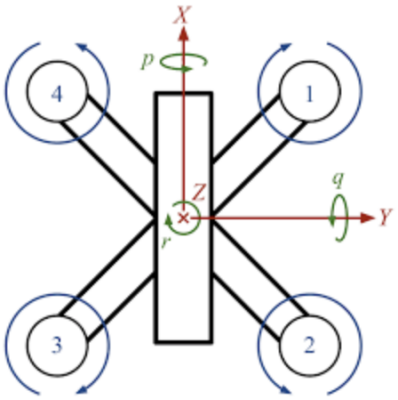

Figura 2.5: Configurazione a croce

comunque un buon controllo, scegliendo ϕ, θ, ψ *e* Z.

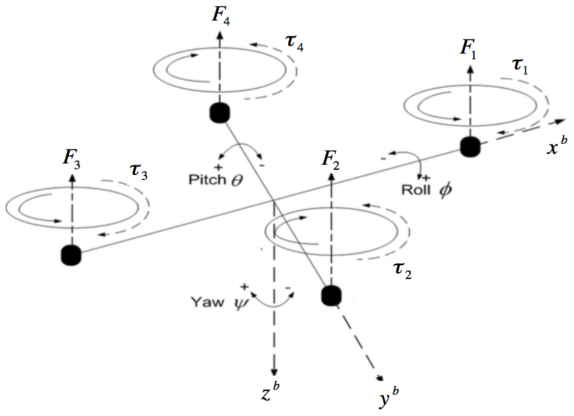

Figura 2.6: Gradi di libertà

Andando a variare le velocità di rotazione di tutte le eliche con un certo criterio, si riescono ad ottenere manovre secondo i gradi di libertà illustrati. Ad esempio, per ottenere una spinta del drone verso l'alto in direzione Z, si aumenta la velocità di rotazione della stessa quantità a tutte le eliche, permettendo anche le manovre di Hovering e Landing, ovvero partenza e atterraggio. Invece, per compiere una modifica dell'angolo di rollio, la velocità delle eliche del lato sinistro/destro devono aumentare e al contempo, il lato opposto deve diminuire. Discorso similare per quanto riguarda gli ultimi due gradi di libertà, con l'unica differenza di avere coppie di eliche differenti per poter compiere le manovre desiderate, Fig. 2.7. (La testa del drone si trova nella parte in alto del foglio).

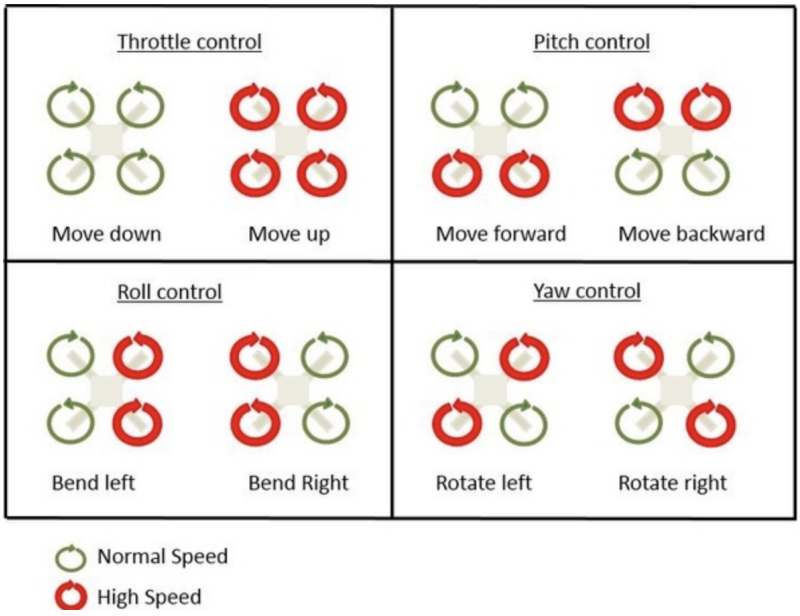

Figura 2.7: Manovre di volo per la configurazione a croce

Andiamo ora ad analizzare le equazioni matematiche alla base del sistema che regolano la dinamica e la cinematica.

2.3 Modello matematico

Sezione fondamentale alla esplicitazione delle equazioni matematiche senza le quali sarebbe impossibile ottenere un sistema computabile e al contempo realizzare un modello eseguibile materialmente; Questa parte di elaborato permette in altre parole, di dare significato a ciascuna parte del modello. Esso si serve di diversi strumenti matematici, che sono le basi per ricavare le equazioni finali, tra cui il formalismo di Newton-Eulero e gli angoli di Tait-Bryan.

2.3.1 Angoli di Tait-Bryan

Sono una variante degli angoli di Eulero, tra cui possiamo trovare gli angoli di roll, pitch e yaw. Il rollio (roll) è una rotazione di un angolo ϕ intorno all'asse X, il beccheggio (pitch) invece rappresenta una rotazione θ intorno all'asse Y ed infine

l'imbardata (yaw), che rappresenta una rotazione ψ intorno all'asse Z. Quindi grazie a questi ultimi è possibile descrivere l'orientamento di un corpo rigido nello spazio, senza cui ovviamente sarebbe impossibile realizzare questo tipo di sistemi. In particolare vige una convenzione secondo cui le terne che rappresentano gli angoli di Tait-Bryan sono quelle in cui compaiono tutti tre gli assi X, Y e Z disposti in modo differente, mentre quelli di Eulero sono le terne in cui il primo elemento della terna coincide con l'ultimo.

Angoli di Eulero, Fig. 2.8:

- z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y

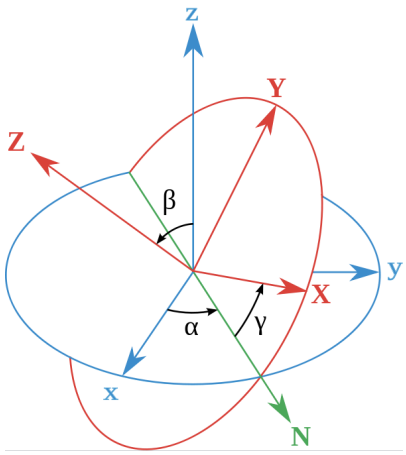

Figura 2.8: Angoli di Eulero

Questi angoli, come si evince dalla immagine, rappresentano la rotazione di un sistema di riferimento mobile, solidale col centro di massa del minidrone, rispetto il sistema di riferimento assoluto, che risulta fisso e centrato in un certo punto dello spazio. In particolare, gli angoli di Eulero descrivono quindi la posizione del sistema di riferimento mobile solidale col drone rispetto una serie di rotazioni. I due sistemi devono avere le origini coincidenti, altrimenti si traccia una retta passante per le due origini che viene detta *linea dei nodi* (N); questi angoli sono:

- α (angolo di precessione), è l'angolo tra l'asse z e la linea dei nodi N.
- β (angolo di mutazione), è l'angolo tra l'asse z e l'asse Z.

- γ (angolo di spin), è l'angolo tra la linea dei nodi N e l'asse X.

Per quanto riguarda invece gli *angoli di Tait − Bryan*, Fig. 2.9:

- x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z

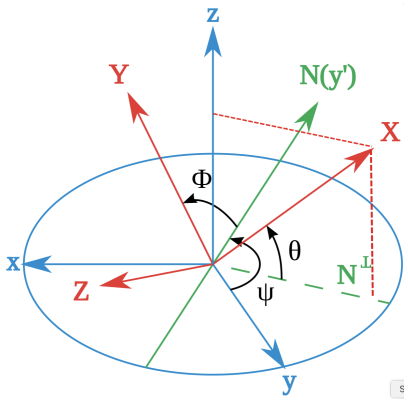

Figura 2.9: Angoli di Tait-Bryan

Le rotazioni intorno i tre assi permettono quindi di effettuare i movimenti di rollio, beccheggio e imbardata come descritto precedentemente.
Il sistema di riferimento mobile viene ricavato dal sistema di riferimento assoluto grazie a tre matrici di rotazione, ciascuna corrispondente ad una delle manovre. L'origine del sistema assoluto viene indicato O_E (Earth), mentre l'origine del sistema solidale col centro di massa del minidrone O_B (Barycenter).
Le tre matrici di rotazione sono (2.1), (2.2), (2.3):

$$Rx(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{2.1}$$

$$Ry(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{2.2}$$

$$Rz(\psi) = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

La matrice complessiva in R^3 si ricava moltiplicando le matrici (2.1), (2.2), (2.3) e per cui si ha (2.4):

$$R_{B->E}(\phi,\theta,\psi) = \begin{bmatrix} cos\theta cos\psi & cos\theta sin\psi & -sin\theta \\ -cos\phi sin\psi + sin\phi sin\theta cos\psi & cos\phi cos\psi + sin\phi sin\theta sin\psi & sin\phi cos\theta \\ sin\phi sin\psi + cos\phi sin\theta cos\psi & -sin\phi cos\psi + cos\phi sin\theta sin\psi & cos\phi cos\theta \end{bmatrix} \tag{2.4}$$

Essendo quest'ultima ortogonale con le righe/colonne linearmente indipendenti, la sua matrice inversa é la semplice trasposizione di $R_{B->E}$ (2.4) anche definita come DCM (2.5).

$$R_{E->B}(\phi,\theta,\psi) = \begin{bmatrix} cos\theta cos\psi & -cos\phi sin\psi + sin\phi sin\theta cos\psi & sin\phi \sin\psi + cos\phi sin\theta cos\psi \\ cos\theta sin\psi & cos\phi cos\psi + sin\phi sin\theta sin\psi & sin\phi cos\psi + cos\phi sin\theta sin\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \tag{2.5}$$

Un'ultima matrice rotazionale di cui si necessita nei passaggi per ottenere le equazioni finali della dinamica/cinematica è la seguente (2.6).

$$ROT = \begin{bmatrix} 1 & sin\phi tg\theta & cos\phi tg\theta \\ 0 & cos\phi & -sin\phi \\ 0 & \frac{sin\phi}{ctg\theta} & \frac{cos\phi}{ctg\theta} \end{bmatrix} \tag{2.6}$$

Quest'ultima ci permette di trovare una relazione tra il vettore delle velocità angolari $\omega_B = \begin{bmatrix} p & q & r \end{bmatrix}^T$ collegate alla terna del corpo rigido ed il vettore delle velocità angolari $\dot{A}_{\mathrm{E}} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$ collegate alla terna mondo (assoluta).
Ne segue la relazione:

$$\dot{A}_{\mathrm{E}} = ROT\omega_B \tag{2.7}$$

Queste sono le matrici di interesse che descrivono il modello cinematico del sistema.

2.3.2 Metodo di Newton-Eulero

Approccio utilizzato per modellare il sistema a sei DOF, che permette di ricavare dodici equazioni, di cui sei riguardanti il sistema di riferimento assoluto O_E e le restanti sei il sistema O_B. Per cominciare si definiscono i vettori di velocità traslazionale V_B (2.8) e della velocità angolare ω_B (2.9) relativi al body-frame del drone, cioè del sistema di riferimento solidale col centro di massa del minidrone O_B.

$$V_B = \begin{bmatrix} u & v & w \end{bmatrix}^T \tag{2.8}$$

$$\omega_B = \begin{bmatrix} p & q & r \end{bmatrix}^T \tag{2.9}$$

Parallelamente, per quanto riguarda il sistema di riferimento assoluto O_E, si utilizzano dei vettori posizione P_E (2.10) e un'altro riguardante l'orientamento spaziale del drone attraverso gli angoli di Tait-Brayn A_E (2.11) accennati in precedenza.

$$P_E = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{2.10}$$

$$A_E = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \tag{2.11}$$

A questo punto, si possono ottenere diverse relazioni tra le matrici esplicate, la prima tra il vettore delle velocità lineari $\dot{P}_E$ e il vettore delle velocità lineari V_B attraverso l'impiego della matrice DCM indicata in (2.5), ottenendo la seguente equazione (2.12):

$$\dot{P}_E = DCM \cdot V_B \tag{2.12}$$

Mentre quella che lega le velocità angolari ω_B con le velocità $\dot{A}_E$ è stata già descritta in (2.7).

Volendo Esplicare gli elementi dei due vettori si ha per $\dot{P}_E$ (2.13) e per $\dot{A}_E$ (2.14):

$$\begin{cases} \dot{x} = u \cdot cos\theta cos\psi + v \cdot (-cos\phi sin\psi + sin\phi sin\theta cos\psi) + w \cdot (sin\phi sin\psi + cos\phi sin\theta \cos\psi) \\ \dot{y} = u \cdot cos\theta cos\psi + v \cdot (cos\phi sin\psi + sin\phi sin\theta sin\psi) + w \cdot (sin\phi cos\psi + cos\phi sin\theta sin\psi) \\ \dot{z} = u \cdot -sin\theta + v \cdot sin\phi cos\theta + w \cdot cos\phi cos\theta \end{cases} \tag{2.13}$$

$$\begin{cases} \dot{\phi} = p + sin\phi tg\theta \cdot q + cos\phi tg\theta \cdot r \\ \dot{\theta} = cos\phi \cdot q - sin\phi \cdot r \\ \dot{\psi} = \frac{sin\phi}{ctg\theta} \cdot q + \frac{cos\phi}{ctg\theta} \cdot r \end{cases} \tag{2.14}$$

(2.13), (2.14) sono le equazioni che regolano la cinematica del sistema. Le restanti sei equazioni riguardano invece la dinamica del sistema e si ricavano attraverso il formalismo di Newton-Eulero (2.15), che tiene conto di alcuni dei vettori precedentemente descritti.

$$\begin{bmatrix} m \cdot I_{3x3} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{V}_B \\ \dot{\omega}_B \end{bmatrix} + \begin{bmatrix} \omega_B \times m \cdot V_B \\ \omega_B \times I \cdot \omega_B \end{bmatrix} = \begin{bmatrix} F \\ \tau \end{bmatrix} \tag{2.15}$$

dove m rappresenta la massa del quadricottero, I_{3X3} la matrice identità 3×3, I il momento di inerzia, F le forze agenti sul drone e τ il momento meccanico (coppia motrice) risultante.

Anche in questo caso, volendo chiarire le equazioni, si possono effettuare dei passaggi, ottenendo le prime tre equazioni della dinamica dalla prima riga della (2.15)

$$\dot{V}_B + \omega_B \times V_B = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \dot{u} + q \cdot w - r \cdot v \\ \dot{v} + r \cdot u - p \cdot w \\ \dot{w} + p \cdot v - q \cdot u \end{bmatrix} \tag{2.16}$$

$$\sum F = F_{g_B} + F_{rotori} = m \cdot (\dot{V}_B + \omega_B \times V_B) \tag{2.17}$$

$$F_g = \begin{bmatrix} 0 \\ 0 \\ m \cdot g \end{bmatrix} \qquad F_{g_B} = DCM \cdot F_g = \begin{bmatrix} -m \cdot sin\theta \\ m \cdot g \cdot sin\phi \cdot cos\theta \\ m \cdot g \cdot cos\phi \cdot cos\theta \end{bmatrix} \tag{2.18}$$

$$F_{rotori} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -(F_1 + F_2 + F_3 + F_4) \end{bmatrix} \tag{2.19}$$

$$F_g + F_{rotori} = \begin{bmatrix} -m \cdot sin\theta \\ m \cdot g \cdot sin\phi \cdot cos\theta \\ m \cdot g \cdot cos\phi \cdot cos\theta - (F_1 + F_2 + F_3 + F_4) \end{bmatrix} \tag{2.20}$$

$$m \cdot \dot{V}_B = m \cdot \begin{bmatrix} \dot{u} + q \cdot w - r \cdot v \\ \dot{v} + r \cdot u - p \cdot w \\ \dot{w} + p \cdot v - q \cdot u \end{bmatrix} \tag{2.21}$$

$$\begin{bmatrix} -m \cdot sin\theta \\ m \cdot g \cdot sin\phi \cdot cos\theta \\ m \cdot g \cdot cos\phi \cdot cos\theta + F_1 + F_2 + F_3 + F_4 \end{bmatrix} = m \cdot \begin{bmatrix} \dot{u} + q \cdot w - r \cdot v \\ \dot{v} + r \cdot u - p \cdot w \\ \dot{w} + p \cdot v - q \cdot u \end{bmatrix} \tag{2.22}$$

Arrivando ad altre tre equazioni del modello, (2.23)

$$\begin{cases} \dot{u} = -g \cdot sin\theta - q \cdot w + r \cdot v \\ \dot{v} = g \cdot sin\phi cos\theta - r \cdot u + p \cdot w \\ \dot{w} = -\frac{F_z}{m} + g \cdot cos\phi c0s\theta + q \cdot u - p \cdot v \end{cases} \tag{2.23}$$

Mentre prendendo ora la seconda riga della (2.15), e svolgendo alcuni passaggi si ottiene, assumendo il veicolo simmetrico rispetto all'asse quindi con la matrice di inerzia avente forma (2.24):

$$I = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \tag{2.24}$$

$$\tau = I \cdot \dot{\omega}_B + \omega_B \times I \cdot \omega_B \tag{2.25}$$

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \cdot \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \cdot \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.26}$$

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{XX} \cdot \dot{p} \\ I_{YY} \cdot \dot{q} \\ I_{ZZ} \cdot \dot{r} \end{bmatrix} + \begin{bmatrix} -I_{YY} \cdot q \cdot r + I_{ZZ} \cdot r \\ I_{XX} \cdot p \cdot r - I_{ZZ} \cdot p \cdot r \\ -I_{XX} \cdot p \cdot q + I_{YY} \cdot p \cdot q \end{bmatrix} \tag{2.27}$$

Da cui ottengo le ultime tre equazioni (2.28) che completano il modello.

$$\begin{cases} \dot{p} = \frac{\tau_x + (I_{YY} - I_{ZZ}) \cdot q \cdot r}{I_{XX}} \\ \dot{q} = \frac{\tau_y + (I_{ZZ} - I_{XX}) \cdot p \cdot r}{I_{YY}} \\ \dot{r} = \frac{\tau_z + (I_{XX} - I_{YY}) \cdot p \cdot q}{I_{ZZ}} \end{cases} \tag{2.28}$$

2.3.3 Forze e momenti esterni al sistema

Si modificano leggermente le formule trovate in precedenza, aggiungendo i contributi di eventuali forze momenti esterni che modificano il comportamento effettivo del drone. In particolare, utilizzando una particolare modellazione, si definisce una corrispondenza tra la spinta fornita da ciascun motore e il quadrato della velocità angolare del singolo motore (2.29):

$$F_i = C_T \cdot \omega_i \tag{2.29}$$

Complessivamente si ottiene quindi, modificando la (2.19), attraverso il legame appena illustrato.

$$F_{rotori} = \begin{bmatrix} 0 \\ 0 \\ -(F_1 + F_2 + F_3 + F_4) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -(C_T \cdot (\omega_1 + \omega_2 + \omega_3 + \omega_4)) \end{bmatrix} \tag{2.30}$$

Sostituendo ora la (2.30) nella (2.20) si arriva alla (2.31).

$$F_g + F_{rotori} = \begin{bmatrix} -m \cdot sin\theta \\ m \cdot g \cdot sin\phi \cdot cos\theta \\ m \cdot g \cdot cos\phi \cdot cos\theta - (C_T \cdot (\omega_1 + \omega_2 + \omega_3 + \omega_4)) \end{bmatrix} \tag{2.31}$$

Inoltre, ogni motore possiede un momento aerodinamico, (2.32):

$$\tau_i = C_q \cdot \omega_i^2 \tag{2.32}$$

dove C_q in questo caso è il coefficente di coppia. Si possono esprimere gli effetti di coppia su ciascun asse, utilizzando la regola della mano destra in riferimento al corpo del drone, (2.33).

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} F_4 \cdot d - F_2 \cdot d = C_T \cdot d \cdot (\omega_4^2 - \omega_2^2) \\ F_3 \cdot d - F_1 \cdot d = C_T \cdot d \cdot (\omega_3^2 - \omega_1^2) \\ C_q \cdot (-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{2.33}$$

Sostituendo (2.33) nella (2.28) si trovano le nuove equazioni (2.34)

$$\begin{cases} \dot{p} = \frac{C_T \cdot d \cdot (\omega_4^2 - \omega_2^2) + (I_{YY} - I_{ZZ}) \cdot q \cdot r}{I_{XX}} \\ \dot{q} = \frac{C_T \cdot d \cdot (\omega_3^2 - \omega_1^2) + (I_{ZZ} - I_{XX}) \cdot p \cdot r}{I_{YY}} \\ \dot{r} = \frac{C_q \cdot (-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) + (I_{XX} - I_{YY}) \cdot p \cdot q}{I_{ZZ}} \end{cases} \tag{2.34}$$

2.4 Equazioni finali del sistema

$$\begin{cases} \dot{x} = u \cdot cos\theta cos\psi + v \cdot (-cos\phi sin\psi + sin\phi sin\theta cos\psi) + w \cdot (sin\phi sin\psi + cos\phi sin\theta \cos\psi) \\ \dot{y} = u \cdot cos\theta cos\psi + v \cdot (cos\phi sin\psi + sin\phi sin\theta sin\psi) + w \cdot (sin\phi cos\psi + cos\phi sin\theta sin\psi) \\ \dot{z} = u \cdot -sin\theta + v \cdot sin\phi cos\theta + w \cdot cos\phi cos\theta \\ \dot{\phi} = p + sin\phi tg\theta \cdot q + cos\phi tg\theta \cdot r \\ \dot{\theta} = cos\phi \cdot q - sin\phi \cdot r \\ \dot{\psi} = \frac{sin\phi}{ctg\theta} \cdot q + \frac{cos\phi}{ctg\theta} \cdot r \\ \dot{u} = -g \cdot sin\theta - q \cdot w + r \cdot v \\ \dot{v} = g \cdot sin\phi cos\theta - r \cdot u + p \cdot w \\ \dot{w} = -\frac{F_z}{m} + g \cdot cos\phi c0s\theta + q \cdot u - p \cdot v \\ \dot{p} = \frac{\tau_x + (I_{YY} - I_{ZZ}) \cdot q \cdot r}{I_{XX}} \\ \dot{q} = \frac{\tau_y + (I_{ZZ} - I_{XX}) \cdot p \cdot r}{I_{YY}} \\ \dot{r} = \frac{\tau_z + (I_{XX} - I_{YY}) \cdot p \cdot q}{I_{ZZ}} \end{cases} \tag{2.35}$$

Tali equazioni descrivono il sistema non lineare.

Capitolo 3

Parrot Minidrone Competition

3.1 Descrizione della competizione

Questo elaborato deriva dalla competizione proposta da *Mathworks*, azienda da cui sono nati prodotti come MATLAB e SIMULINK, che ha sviluppato questa piattaforma per scopo didattico, al fine di confrontare algoritmi diversi e contendersi il premio finale.
La competizione si divide in due turni:

- Round 1, turno della simulazione, durante il quale le squadre partecipanti programmano il proprio algoritmo di inseguimento della traiettoria, attraverso ambiente *Simulink* e sfruttando il Model-Based Designed che si rispecchia nel linguaggio del Stateflow.

- Round 2, in cui le squadre che abbiano ottenuto un buon risultato durante le simulazioni del turno precedente vengono invitate ad eseguire il codice realizzato sul drone fisico, implementando in altre parole la parte Software all'interno dell'Hardware.

Al termine della seconda fase, ci sarà la valutazione dei partecipanti e la selezione dei vincitori da parte della giuria prescelta, attraverso specifici criteri di valutazione.

3.1.1 Specifiche di volo

- il drone parte da terra, posizionato all'inizio del percorso, e dopo aver raggiunto la quota di 1,1 metri di altezza dal suolo, deve mantenerla fino alla fase di atterraggio.
- la larghezza del percorso da seguire di 10 centimetri.
- il diametro del cerchio di atterraggio di 20 centimetri.
- la distanza tra l'origine del cerchio e la fine del percorso ammonta a 25 centimetri.
- il percorso non contiene curve ma solo segmenti collegati tra di loro che formano angoli diversi.
- il colore del tracciato può variare, con il colore di default rosso.
- il background non avrà tinta unita per evitare eventuali problemi alla ricezione dell'immagine dal sensore fotocamera del minidrone.

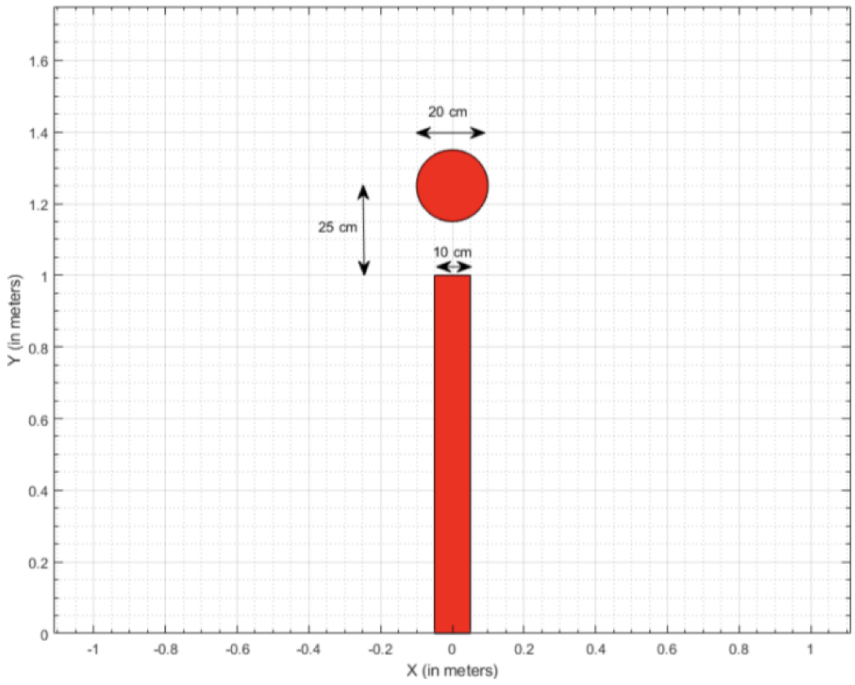

Figura 3.1: Layout della parte terminale del percorso da seguire

3.1.2 Atterraggio

Per essere valido, l'atterraggio deve rispettare almeno la specifica di contenere almeno un paraurti completamente, nell'eventuale possibilità in cui il completo corpo del drone finisca fuori dalla circonferenza. Inoltre, l'atterraggio viene accettato solo se il dorso del drone rimane verso l'alto, cioè non deve capovolgersi. Nelle immagini (3.2), (3.3), si osservano rispettivamente dei casi di successo e insuccesso.

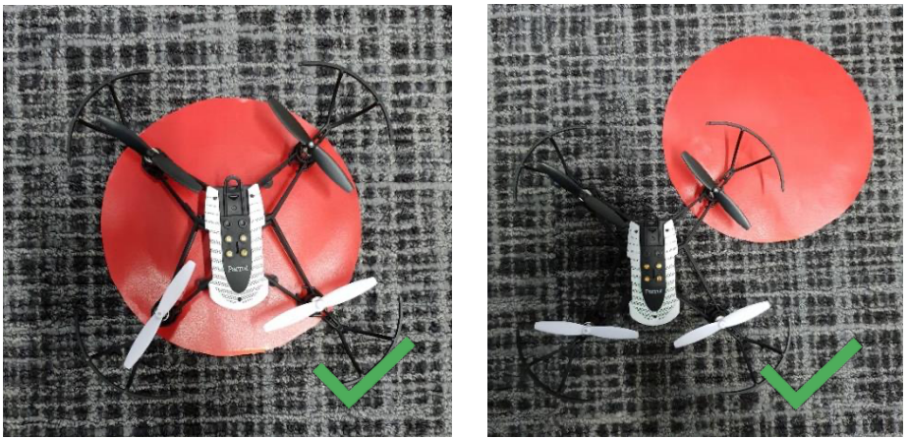

Figura 3.2: Atterraggio corretto

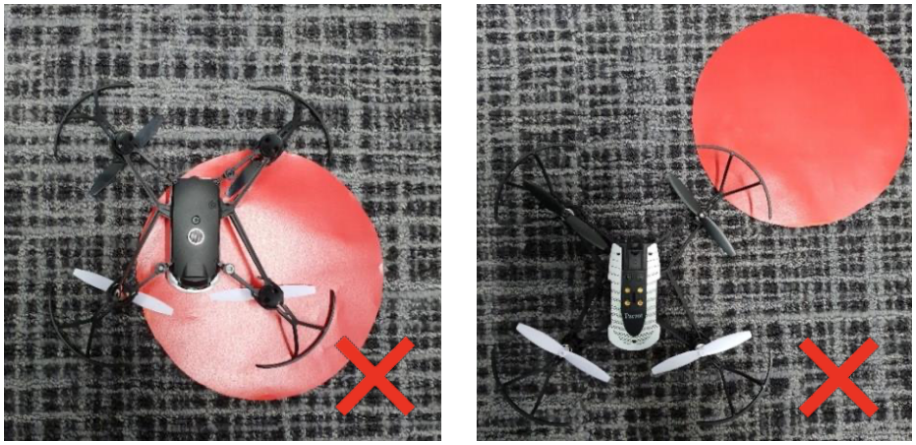

Figura 3.3: Atterraggio errato

Per ulteriori informazioni sulla competizione, consultare link [**8**] , [**9**] in bibliografia.

3.2 Modello Simulink della competizione

3.2.1 Creazione del progetto

Per creare un nuovo progetto basato sulla competizione a cui si è fatto riferimento in precedenza, digitare nella *Command Window* di MATLAB il comando seguente, (dopo aver installato tutti i pacchetti necessari):

- parrotMinidroneCompetitionStart

Tale operazione crea una cartella di lavoro nel Workspace di Matlab comprendente tutti i file necessari alla realizzazione del lavoro, dopodiché si accinge all'apertura del modello di lavoro in ambiente SIMULINK.

3.2.2 Descrizione del modello Simulink

Una volta aperto il modello, la schermata principale del programma ha la forma seguente (3.4).

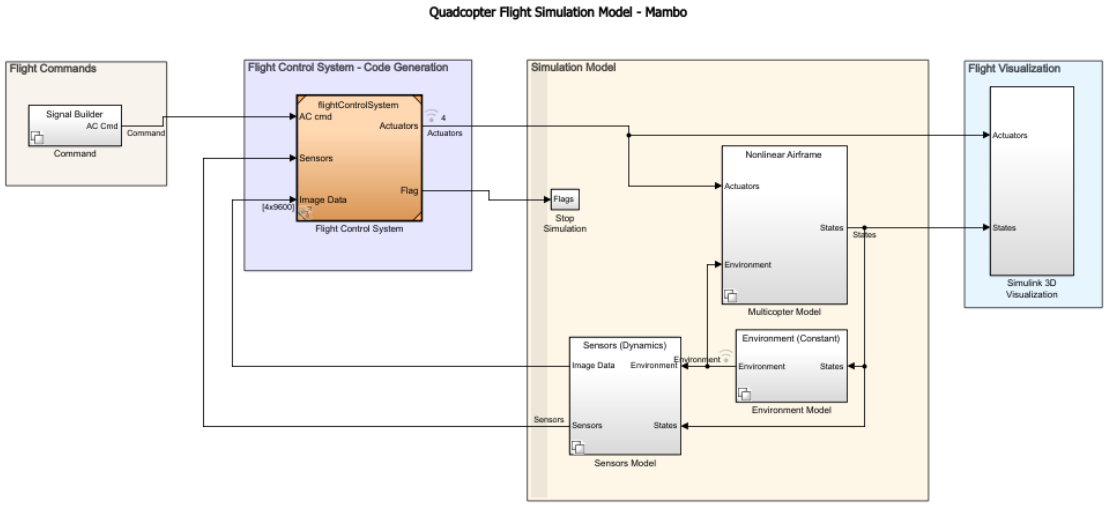

Figura 3.4: Quadcopter Flight Simulation Model - Mambo

Questo modello a blocchi traduce quello matematico all'interno dei vari blocchi che interagiscono per formare il sistema nella sua completezza. In particolare si osservano sette blocchi principali, ognuno formato da possibili sotto-blocchi e costituenti

diverse funzionalità. Il blocco che si deve modificare per completare il progetto si trova nel blocco arancione denominato *FlightControlSystem*, che verrà analizzato successivamente. I componenti appena annunciati sono i seguenti:

- *Command*, che gestisce i comandi da remoto in ingresso al minidrone.
- *Flags*, composto da diverse condizioni utili per evitare eventuali rischi dovuti a logiche sbagliate o malfunzionamenti del codice. Serve inoltre per evitare di arrecare danni al drone in questione.
- *Sensors*, che si occupa dell'acquisizione dei dati dai quattro sensori che possiede il minidrone, come ad esempio le immagini dalla fotocamera o i dati dall' accelerometro e dal giroscopio compresi nell'IMU.
- *Non linear Airframe*, contenente la struttura del velivolo.
- *Environment*, che contiene le costanti ambientali.
- *Flight Visualization*, che permette di analizzare e visualizzare i risultati in uscita, in particolare sulla finestra di simulazione, se non si possiede il drone fisico per dei test.
- *Flight Control System*, contenente le parti da modificare come anticipato in precedenza, dove realizzare la logica di inseguimento della traiettoria.

Si vanno ora ad analizzare uno ad uno i blocchi appena specificati, e distinguibili chiaramente nell'immagine soprastante.

3.2.3 Command - Signal Builder

Si hanno diverse possibilità di input assumibili dall' *AC cmd* a seconda del valore che si impone alla variabile $VSS\_COMMAND$. Si assume per le simulazioni l'opzione predefinita del Signal Builder, come intuibile dalla illustrazione in Fig. (3.5).

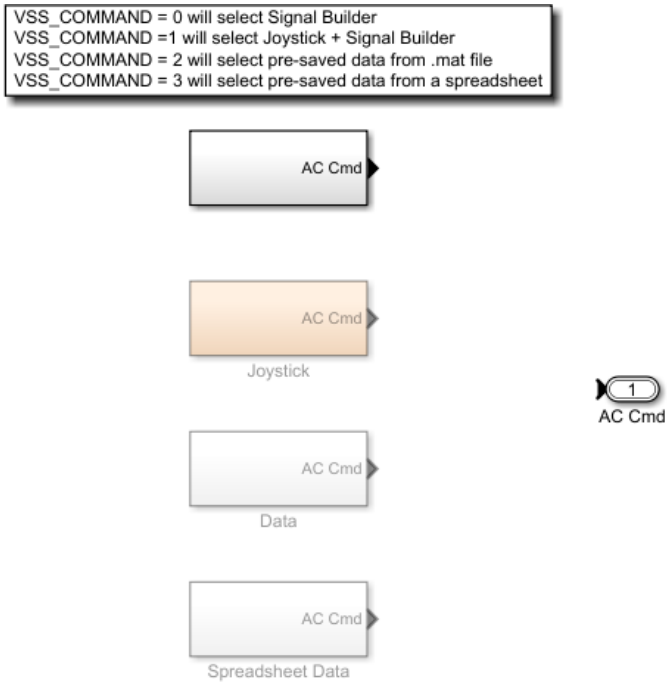

Figura 3.5: Blocco Command

Entrando più a fondo nel blocco *AC Cmd* si osserva la forma del Signal Builder, Fig. (3.6).

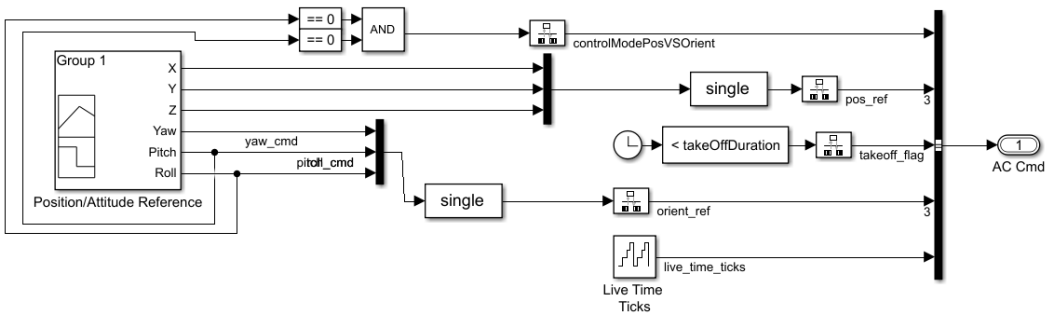

Figura 3.6: Blocco Signal Builder

Si osserva che, partendo dalla lettura dei dati di posizione e altitudine di riferimento, si operano semplici operazioni e si mandano in uscita le stesse informazioni manipolate a seconda del tipo di variabili che servono in uscita, raggruppate tutte in un bus dati.

3.2.4 Sensors

Subentrando nei sotto-blocchi interni al blocco principale, passando per il blocco *Sensors* (*Dynamics*) si arriva fino alla schermata di Fig. (3.7).

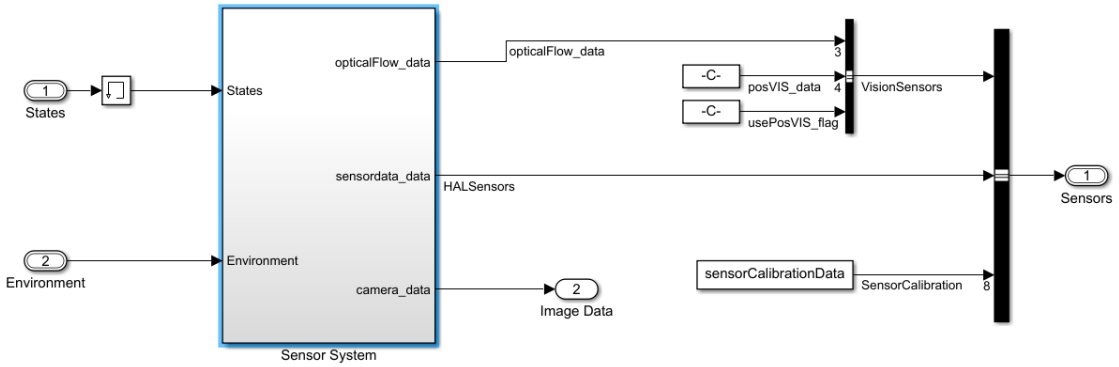

Figura 3.7: Blocco Sensors

Si osserva il blocco *Sensor System* che si occupa della parte di lettura delle informazioni, provenienti dai terminali fotocamera e IMU.

3.2.5 Non Linear Airframe

Scendendo internamente si trova il modello non lineare di figura (3.8).

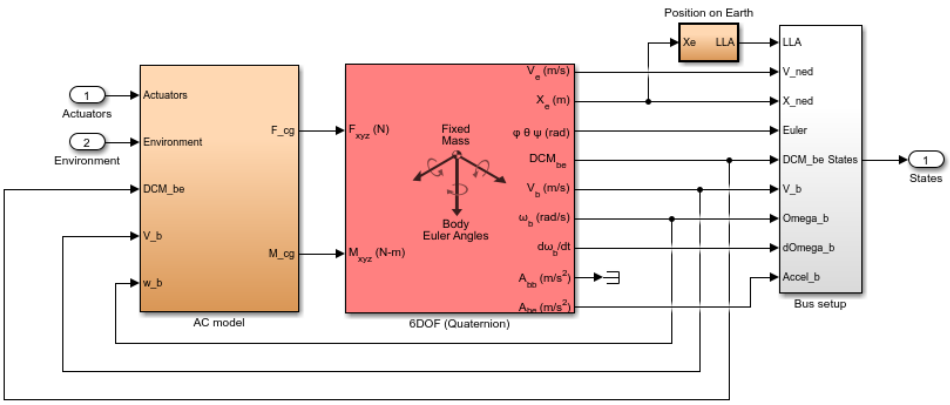

Figura 3.8: Blocco Command

Quest'ultimo è formato da ulteriori blocchi tra cui *AC Model*, che contiene la par-

te della dinamica e dal quale si ricavano le forze e i momenti agenti sui tre assi sotto forma di uscite. Queste ultime a loro volta fungono da ingressi per il blocco $6DOF$ (*Quaternion*), che implementa le equazioni del moto a sei gradi di libertà rispetto agli assi del corpo, già presentate nella sezione (2.3). Permette in altre parole di calcolare velocità, accelerazioni e angoli grazie a quelle determinate equazioni. Infine, si osserva il blocco *Bus setup* in cui si raccolgono tutti dati precedentemente calcolati.

3.2.6 Environment

Settore che comprende le definizioni delle costanti e delle variabili ambientali che possono agire ed influire sulla dinamica del sistema, variabili che di default sono impostate a valori costanti attraverso la variabile $VSS\_ENVIRONMENT$ settata a valore 0. Nel caso in cui venga posta pari a 1, si tenderà ad avere una rappresentazione più realistica, con coefficienti variabili nel tempo. Scendendo in profondità tra i vari sotto-blocchi si arriva alla parte desiderata. Si mostra in Fig. (3.9) la definizione delle variabili ambientali costanti.

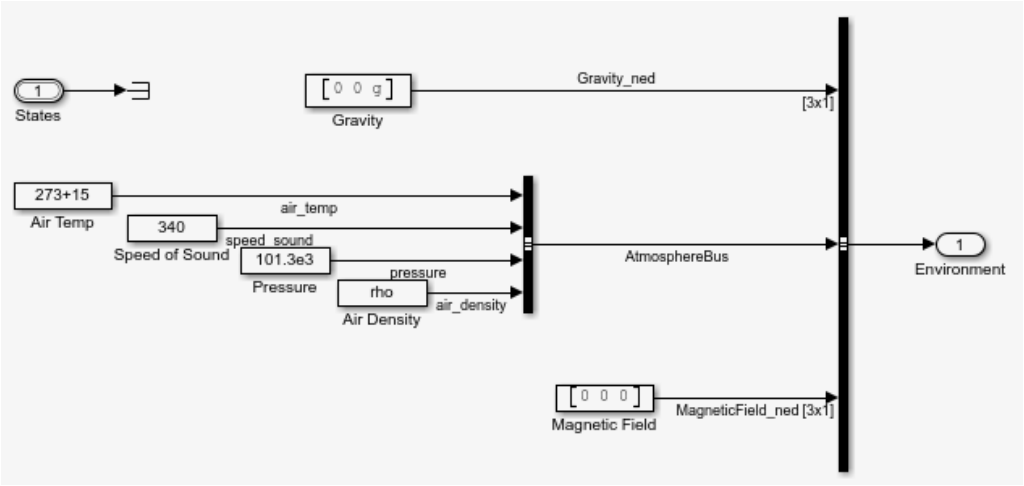

Figura 3.9: Blocco Environment (Constant)

Si osservano elementi come costante gravitazionale con contributo esclusivamente in direzione dell'asse Z, coefficienti come la temperatura dell'aria, la velocità del

suono, la pressione atmosferica e l densità dell'aria, fino ad arrivare al contributo di eventuali campi magnetici, che per questo studio, vengono considerati nulli.

3.2.7 Flight Visualization

Sezione del codice che si occupa essenzialmente della simulazione del progetto, infatti così come suggerisce il nome, permette di visualizzare i risultati in una schermata video illustrata in Fig. (3.10).

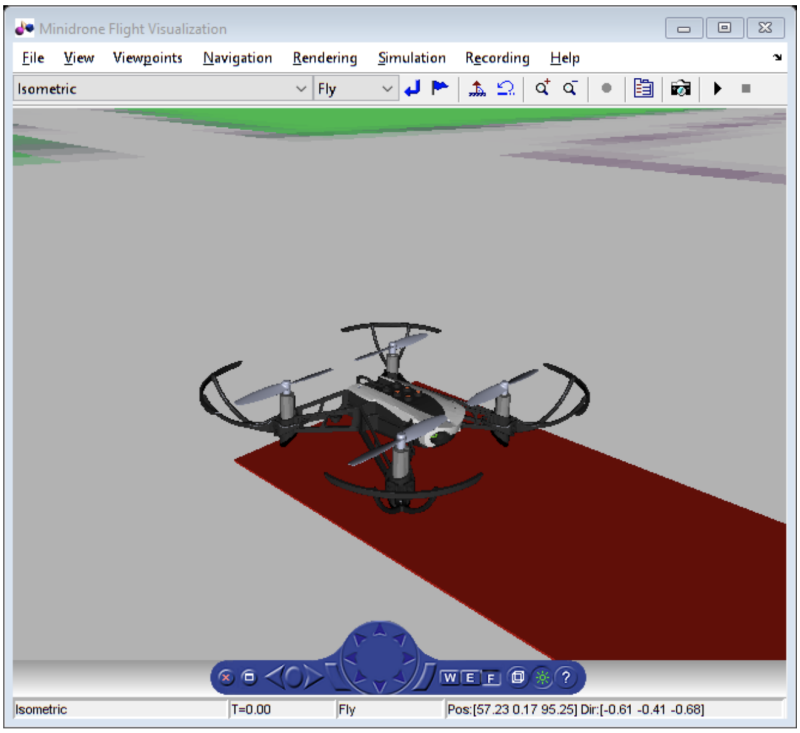

Figura 3.10: Minidrone Flight Visualization

Per ulteriori informazioni sui blocchi che gestiscono la visualizzazione video della simulazione ci si muova all'interno dei vari sotto blocchi e si interpreti la logica di funzionamento.

3.2.8 Flight Control System

Come ultima sezione del codice si analizza il blocco su cui si pone la principale attenzione di questo elaborato, che come già anticipato comprende le aree di sviluppo del codice per la realizzazione e il completamento del lavoro. Entrando di un livello all' interno del blocco sopracitato si raggiunge la schermata illustrata in Fig. (3.11), che risulta a sua volta essere composta da due sezioni le quali saranno argomento dei prossimi due capitoli, ovvero il blocco *Image Processsing System* e il blocco *Control System*. Si osservi in particolare la presenza di un elemento denominato *Rate Transition* tra i due, con la funzione di adattare il tempo di campionamento dei blocchi, dato che lavorano a frequenze differenti.

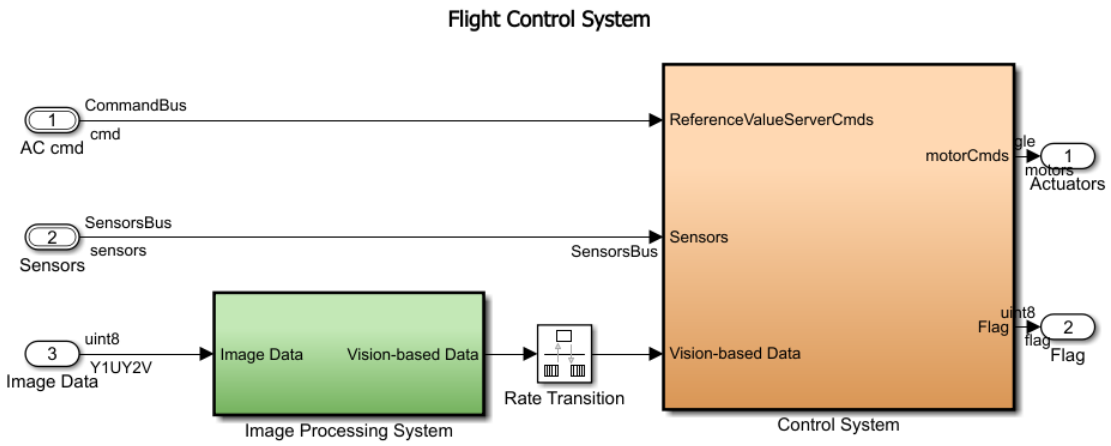

Figura 3.11: Blocco Flight Control System

Ora che si possiede una base concreta dell'ambiente circostante e delle leggi che regolano il complesso, si procede alla spiegazione della logica di inseguimento realizzata.

Capitolo 4

Image Processing System

Questo capitolo descrive lo studio e l'analisi dell'immagine catturata dal sensore fotocamera sottostante al mini drone, attraverso determinati metodi per la traduzione dell'immagine a colori in immagine BW, per poi arrivare a distinguere i bordi dell'immagine BW attraverso degli algoritmi ed ottenere l'angolo formato dalla traiettoria. Si parta con la descrizione teorica dei filtri RGB e HSV, utilizzati per lo studio e confrontati per capire quale fosse il migliore tra di essi in questa particolare circostanza di lavoro, per poi analizzare praticamente i risultati.

4.1 App Color Thresholder

Applicativo Matlab che permette di manipolare le immagini rendendole di tipo binario, in base al colore che si vuole isolare, in questo elaborato in base al colore del percorso. Per aprire l'App si digiti il comando "colorThresholder" direttamente sulla *Command Window*. Una volta aperta la pagina di interesse, si importi l'immagine desiderata da analizzare, e si scelga il tipo di filtro desiderato per manipolare l'immagine, tutto visibile in Fig. (4.1).

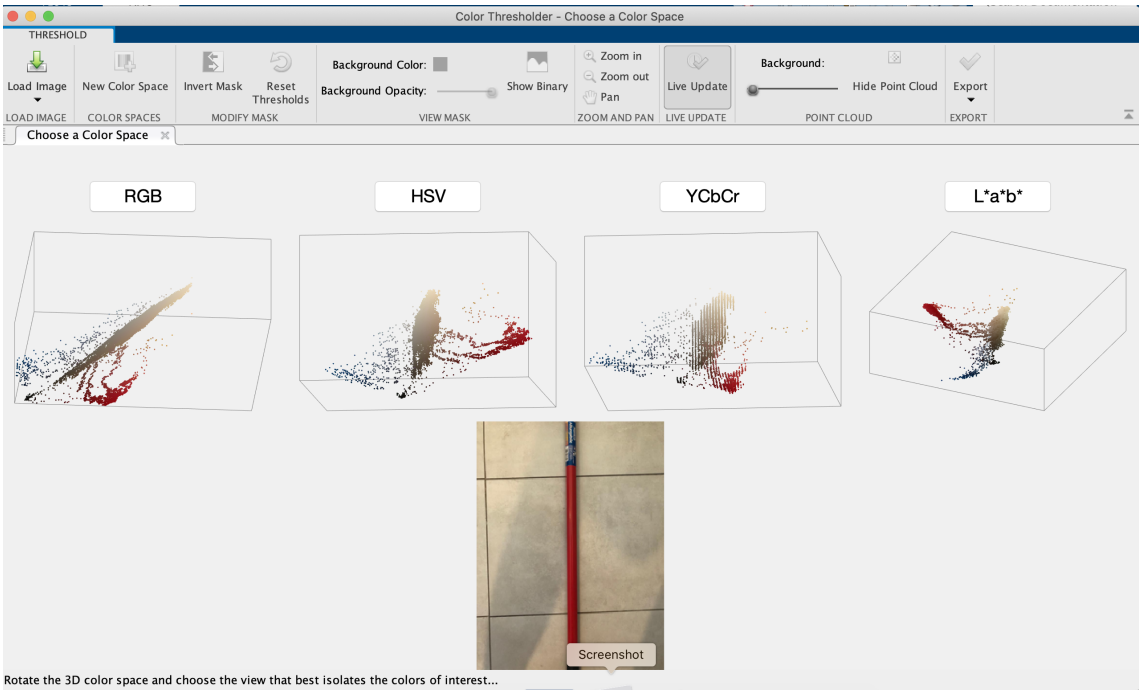

Figura 4.1: App Color Thresholder

Come anticipato, i due spazi colore prescelti per il lavoro sono l'RGB e l'HSV, di cui ora si accenna il funzionamento. Per lo studio del colore rosso (colore di default del percorso), si utilizza un manico di scopa di materiale lucido, grazie a cui si riuscirà ad effettuare una considerazione più stringente sulla scelta del filtro migliore attuabile, dato che quella tipologia particolare di materiale può riflettere la luce e creare errori sul filtraggio del colore.

4.2 filtri RGB - HSV

4.2.1 Filtro RGB

É un modello di colori di tipo additivo, come somma dei tre colori *rosso* (R), *verde* (G) e $blu(B)$, che combinandoli in modi differenti ci portano a colori altrettanto differenti, Fig. (4.2). Principalmente, risulta un modello particolarmente

adatto per le immagini di dispositivi elettronici, tuttavia potrebbe utilizzando lo stesso filtro su dispositivi diversi potrebbe essere visualizzata in maniera diversa, a seconda delle circostanze e in particolare delle condizioni luminose che possono determinare errori ed incertezze nella traduzione dell'immagine.

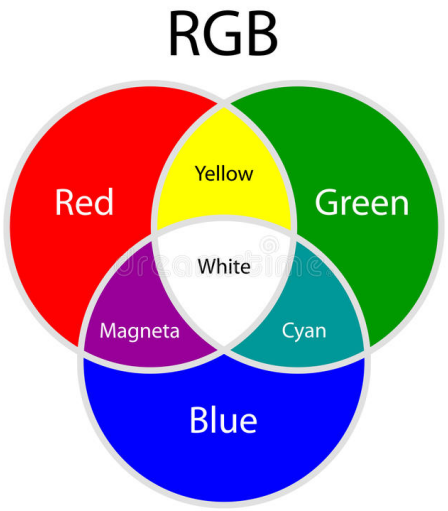

Figura 4.2: Filtro RGB

Si pone ora l'attenzione sul filtro HSV, che tiene conto di aspetti più utili nello studio dell'immagine (nel progetto in discussione) e concreti dal punto di vista dei risultati.

4.2.2 Filtro HSV

Al contrario del precedente, il modello HSV separa la luminanza, ovvero l'intensità dell'immagine, dalle informazioni sul colore. Questo risulta molto utile quando si desidera lavorare sulla componente d' intensità, piuttosto che le componenti dei colori, Fig. (4.3).
Come si osserverà, in confronto all'RGB, il quale risulta essere molto sensibile alle ombre e alla luce in particolare, il filtro HSV è in grado di ottenere risultati più soddisfacenti dal punto di vista di possibili agenti luminosi agenti sul percorso, essendo in grado di limitare l'effetto di questi ultimi.

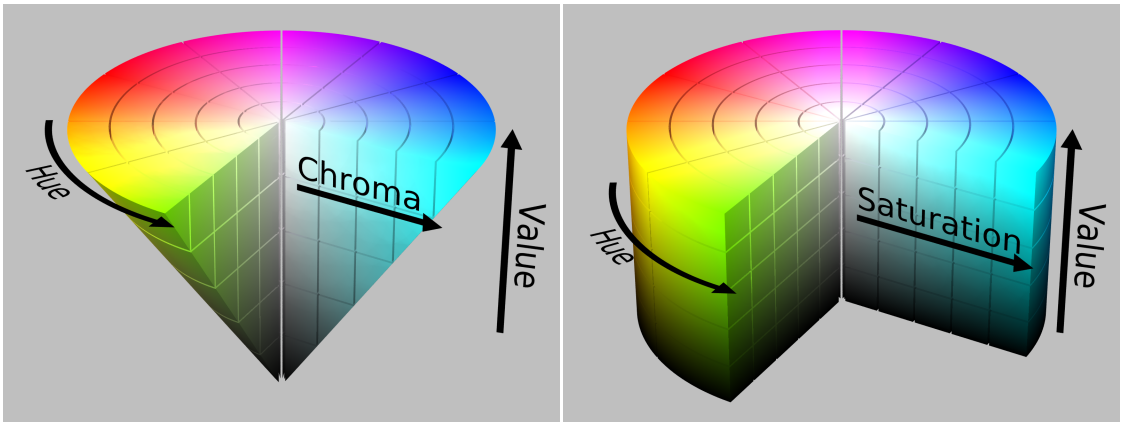

Figura 4.3: Filtro HSV in due diverse rappresentazioni

4.2.3 Confronto dei due spazi di colore RGB e HSV

Si proceda ad effettuare un test con entrambi i modelli presentati, ottenendo i due risultati di Fig. (4.4), e Fig. (4.5)

Al contrario dell'RGB il quale fatica ad isolare le ombre, si immagini come esempio esplicativo due zone differenti con lo stesso colore sottoposte ad intensità luminosa differente; nello spazio di colore RGB, la parte con ombra avrà probabilmente caratteristiche molto diverse rispetto alla parte senza ombre. Nello spazio di colori HSV invece é più probabile che la componente di tonalità di entrambe la patch sia simile, infatti l'ombra influenzerà principalmente il valore, ovvero la componente di saturazione, mentre la tonalità, indicando il colore primario non dovrebbe cambiare molto. Inoltre è chiaramente osservabile dalle immagini dei risultati ottenuti, che il filtro HSV riesce ad isolare meglio i dettagli, rendendo l'immagine più pulita e quindi con funzionamento più efficace.
Si proceda dunque all'esportazione della funzione in formato codice Matlab di Fig. (4.6), pronta per essere inserito all'interno del progetto, nel blocco che verrà spiegato nel corso della prossima sezione.

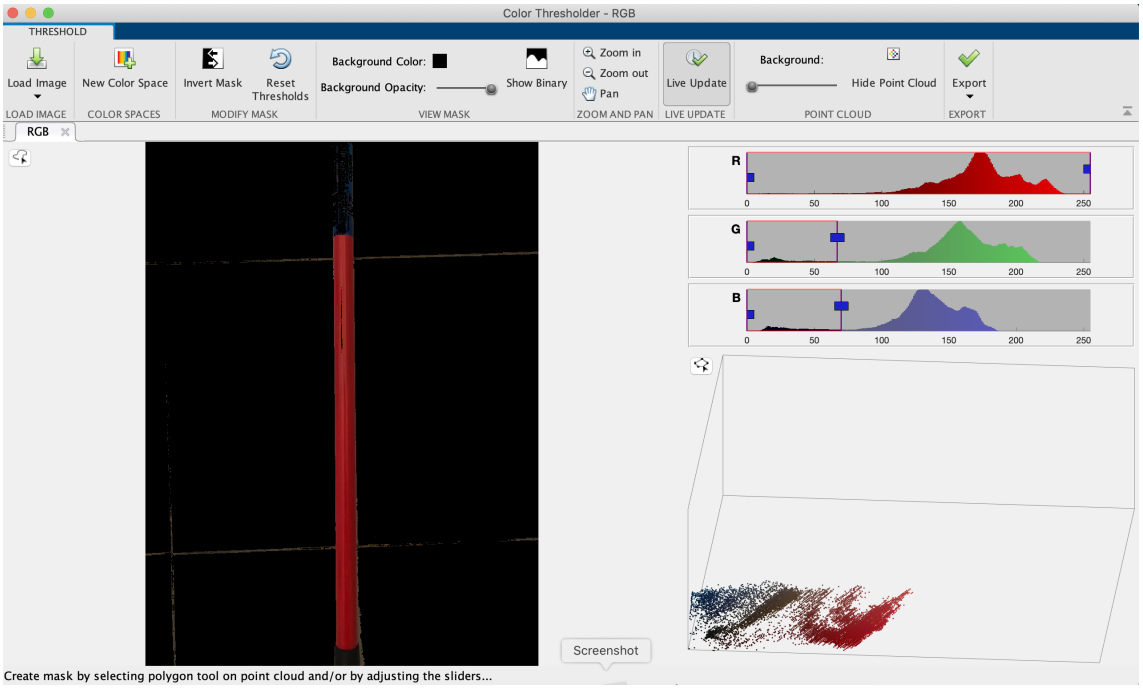

Figura 4.4: Filtro RGB in azione

Per adattarlo al codice basta eliminare la parte evidenziata del codice per essere funzionante.

4.3 Blocco Image Processing System

Nella illustrazione di Fig. (4.12), è rappresentato il blocco nella sua interezza, in cui si passa dalla traduzione della immagine catturata dalla fotocamera in immagine binaria (elementi sono 0 e 1) attraverso la funzione di Fig. (4.6), relativa al blocco denominato *Colour Filter RGB*, alla manipolazione dell'immagine appena citata in sotto matrici, ognuna con diverse funzionalità, utili nell'elaborazione della logica riguardante la progettazione dell'algoritmo di inseguimento della traiettoria, (sotto matrici relative ai riquadri verdi successivi al blocco precedente, ovvero *Colour Filter RGB*).

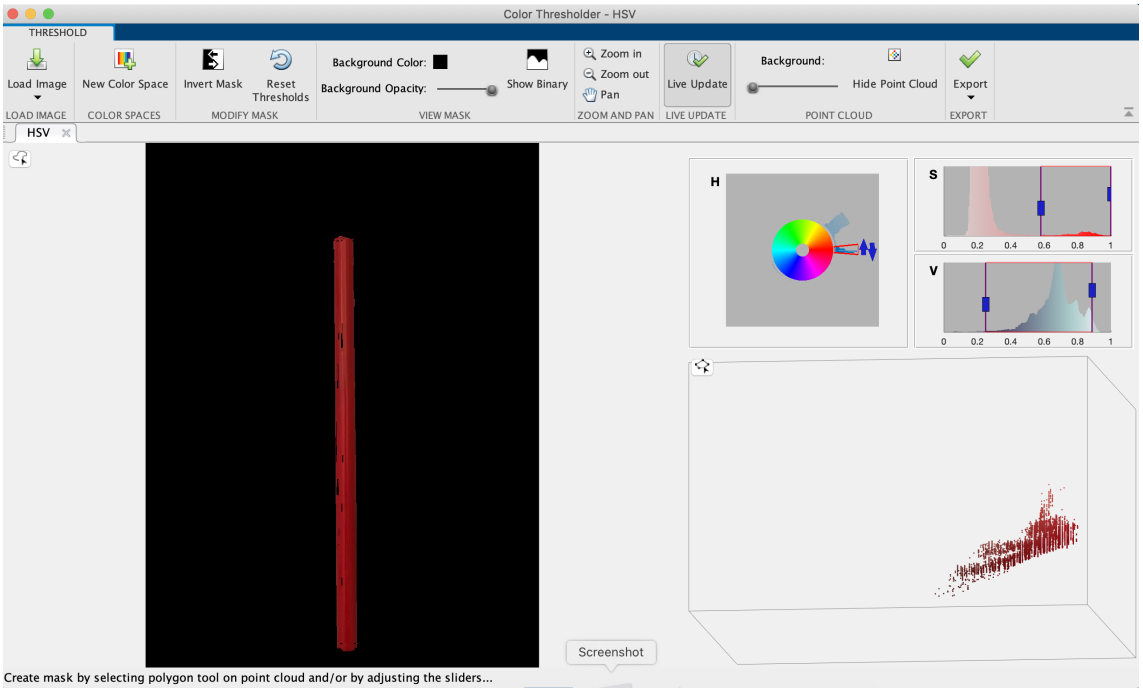

Figura 4.5: Filtro HSV in azione

La matrice BW principale corrispondente all'immagine binaria in uscita dalla funzione sopra citata ha dimensioni 120×160 e ciascuna sottomatrice forma un sottoinsieme di dimensione indipendente dalle altre. Le varie sottomatrici e le relative dimensioni sono rappresentante in una griglia di dimensione ridotta, che rappresenta la parte interna della matrice BW (Fig. (4.7)).

Le sottomatrici illustrate coprono un ruolo importante nello sviluppo della logica di inseguimento, come si vedrá nel capitolo successivo. In particolare, si effettuano particolari verifiche sugli elementi delle sottomatrici, che corrispondono ai pixel dell'immagine tradotta, condizioni che ci permettono di gestire con una determinata logica il controllo su determinati aspetti di volo.
Un' ultima parte fondamentale al completamento del progetto risiede nel blocco matematico *FindDircetion*, all'interno di uno dei rettangolo verdi, il cui interno è formato dalle righe di codice di Fig. (4.8)

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

% Auto-generated by colorThresholder app on 15-May-2020
%------------------------------------------------------


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

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.987;
channel1Max = 0.016;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.581;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.251;
channel3Max = 0.888;

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

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

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

end
```

Figura 4.6: Funzione su ambiente Matlab HSV

```
function ThetaAverage = FindDirection(Matrix2)

BW = edge(Matrix2,'Roberts');           %ricavo i bordi della sottomatrice BW StopTurning
[H,theta,rho] = hough(BW);              %effettuo la trasformata di Hough
peaks  = houghpeaks(H,2); %n=2          %cerco i due picchi principali
ThetaAverage = mean(theta(peaks(:,2))); %calcolo l'angolo della traiettoria

end
```

Figura 4.8: Funzione su ambiente Matlab FindDirection

La funzione $edge(I, method)$, richiamabile per modificare una certa immagine

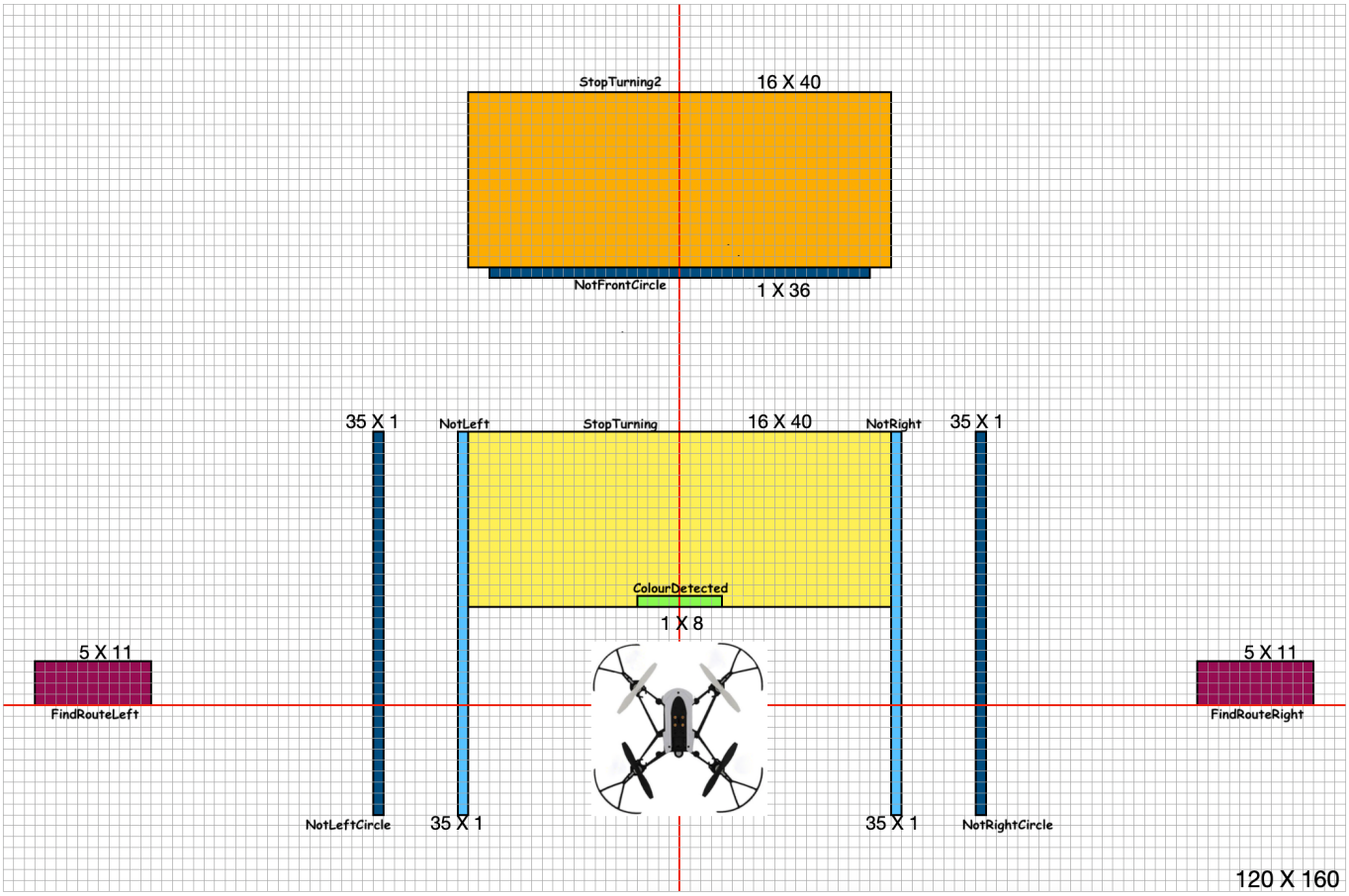

Figura 4.7: Immagine BW con le sottomatrici progettate

binaria, ha come parametro principale l'immagine binaria da sottoporre alla lavorazione, mentre come parametro secondario la scelta di un metodo per ricavare i bordi dell'immagine precitata; in particolare si è effettuato un test di prova con tutte le opzioni disponibili su di una porzione di percorso con angolo retto, (si veda Fig. (4.9)), da cui si è scelto per questo elaborato il metodo di Roberts perché risulta visivamente piú pulito e preciso rispetto i suoi concorrenti. Infine attraverso la funzione $mean(A)$ si calcola la media tra gli angoli relativi ai due picchi trovati dalla funzione $houghpeaks(H, numpeaks)$. Si osserva visivamente che il risultato dell'angolazione risulta essere corretto, in quanto l'angolo in questione è retto. Nell'immagine di Fig. (4.9), sono stati effettuati dei test con un codice Matlab, in cui si arriva fino a stampare a video l'angolo di una traiettoria ipotetica che potrebbe verificarsi. Si analizzano in breve le funzioni matematiche presenti nel codice.

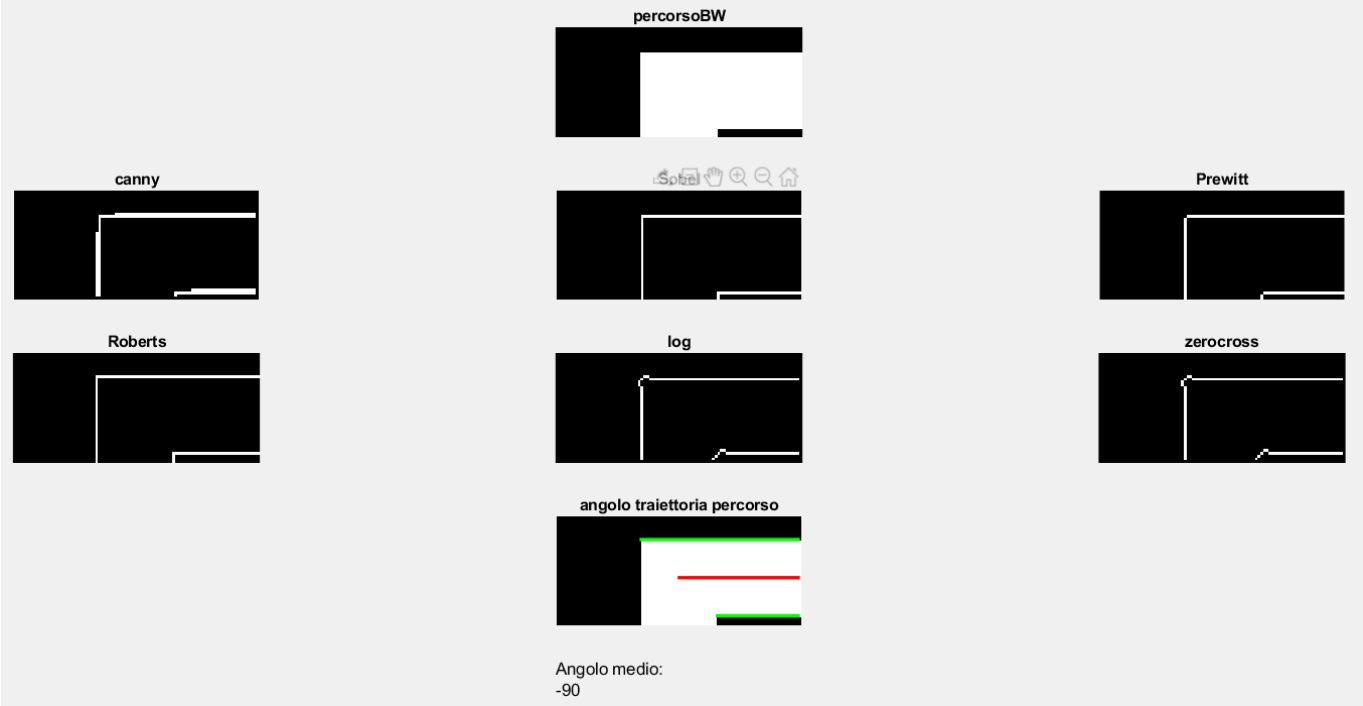

Figura 4.9: Funzionamento grafico del codice di Fig. (4.8)

4.3.1 Funzione edge

Con la parole angolo-sassone "edge", ovvero contorno (bordo), si indica una curva di dimensione variabile a seconda del contesto e dell'oggetto in discussione, che separa zone di spazio che si differenziano per una rapida variazione di colori, in particolare delle varie intensitá che si possono verificare. Infatti, i bordi tipicamente sono conosciuti come quell'elemento che delimita un qualsiasi oggetto definendone il perimetro. Quindi la funzione *edge* serve per analizzare determinate immagini in cui le intenistà di colorazione variano rapidamente e di conseguenza, attraverso criteri che sfruttano le derivate di determinate funzioni rappresentanti i contorni dell'immagine analizzata. Il criterio più performante esistente è il metodo di Canny, ma come visto in precedenza, talvolta potrebbe essere più semplice e preciso un metodo diverso a seconda del contesto in esame. Tenere in considerazione che se l'ambiente circostante fosse affetto da rumori o disturbi di vario genere, risulta una buona idea utilizzare il metodo di Canny, che isola meglio questi ultimi fattori negativi. Dettagli ulteriori

in [**11**], [**12**]. Esempio esplicito di utilizzo della funzione *edge* in Fig. (4.10).

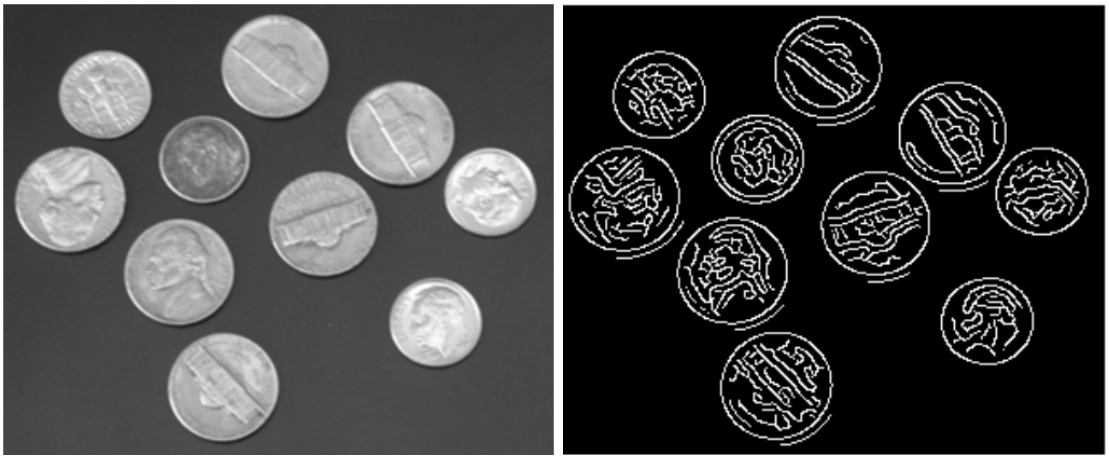

Figura 4.10: Esempio in azione della funzione *edge*

4.3.2 Funzione hough

Funzione che sfrutta il principio della trasformata di Hough, la quale riguarda una tecnica di estrazione utilizzata nel campo dell'elaborazione digitale di immagini. Nella sua forma principale, pone il suo funzionamento sul riconoscimento delle linee principali dell'immagine, ma è poi stata estesa anche a forme più complesse. La funzione sfrutta la rappresentazione parametrica di una linea, ovvero:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{4.1}$$

ρ rappresenta la distanza della linea dall'origine degli assi, distanza misurata su di un segmento perpendicolare alla linea in questione; θ invece rappresenta l'angolo in gradi (deg) che si forma tra il segmento perpendicolare pre-enunciato e l'asse X, Fig. (4.11). L'insieme dei valori assumibili da θ vale $-90° \leq \theta < 90°$. Ulteriori informazioni in [**13**].

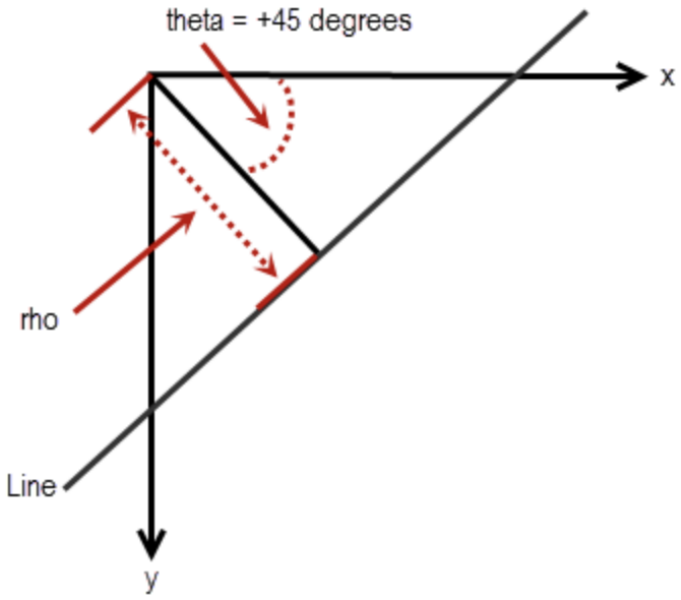

Figura 4.11: Rappresentazione della trasformata di Hough

4.3.3 Funzione houghpeaks

Attribuisce a determinati punti dell'immagine dei picchi calcolati dalla trasformata di Hough. A seconda di quanti picchi si vogliono ricavare dall'immagine si riusciranno ad ottenere un determinato numero di linee; in dettaglio, più picchi si richiedono alla funzione, più linee che li attraversano si riescono ad ottenere. Per chiarire questo aspetto si osservi l'immagine in Fig. (4.9), in cui vengono richiesti due picchi, quindi si riesce ad ottenere un'unica linea rossa passante per quei due punti. Visitare **[14]** per ulteriore documentazione.

4.3.4 Rappresentazione del codice

Si rappresenta nella sua completezza il blocco di codice riguardante lo studio e la manipolazione dell'immagine descritto nelle pagine precedenti, Fig. (4.12).

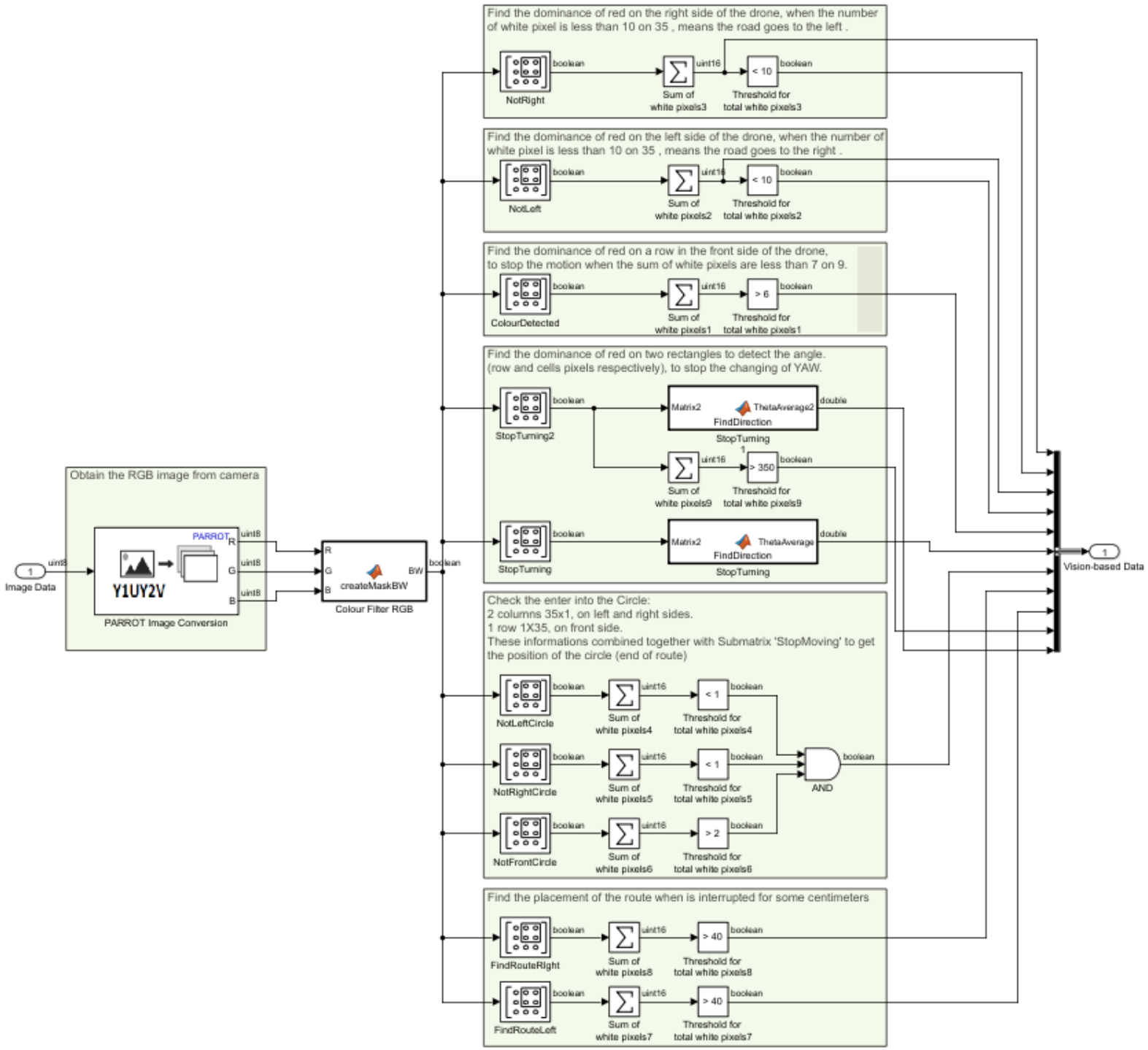

Figura 4.12: Blocco Image Processing System

Capitolo 5

Control System

Ultimo capitolo di questa tesi, comprende la descrizione della logica di volo del minidrone, cioè l'intuizione attraverso l'APR insegue la traiettoria predisposta dalla competizione. In dettaglio, si analizzano le sottocartelle su cui muoversi per arrivare al percorso desiderato, per poi analizzare le fasi di volo progettate separatamente. Tutto il codice di questa parte è realizzato in linguaggio di Stateflow, ovvero un linguaggio che sfrutta il $ModelBasedDesign$,

5.1 Model Based Designed

Si tratta di un metodo matematico per progettare logiche di controllo di sistemi automatizzati, ma anche nella elaborazione di segnali o sistemi di comunicazione, Fig. (5.1). Fornisce un approccio a dir poco efficiente nello stabilire una rete di comunicazione tra diversi componenti di un sistema attraverso il processo di progettazione descritto da quattro fasi principali:

- modellare una piantina da seguire, ovvero lo scheletro/schema della logica da seguire;
- analizzare e sintetizzare un controller per lo scheletro ideato;
- simulare il controller progettato per quello schema

- integrare tutte le fasi precedenti in un software applicativo da eseguire, non appena tutte le simulazioni e i test di verifica restituiscono responsi positivi.

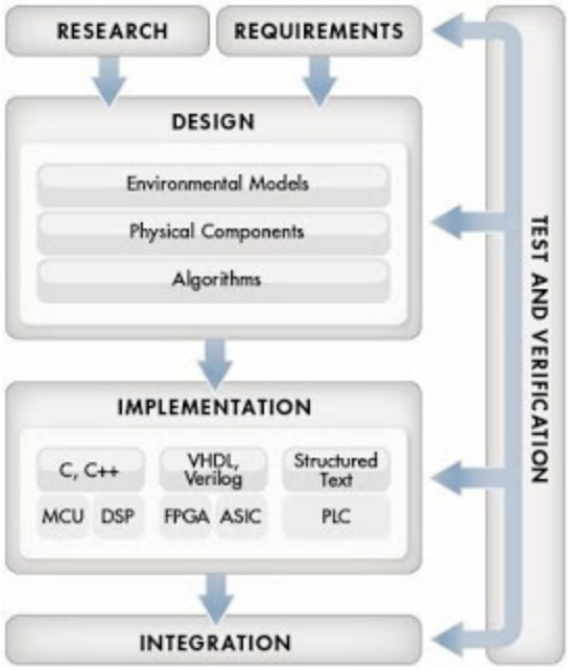

Figura 5.1: Model Based Design

Come vantaggi principali, tiene particolare rilevanza il fatto dell'esistenza di un ambiente di lavoro realizzato al fine di facilitare tutte le operazioni necessarie allo svolgimento di un qualsiasi progetto; inoltre come qualsiasi buon ambiente, permette la modifica degli errori che si presentano in fase di programmazione, indicando la posizione dell'errore e talvolta suggerendo delle azioni da apportare; come ultimo aspetto, può essere considerato un buon mezzo attraverso il quale migliorare o aggiornare dei sistemi realizzati in altri modelli di programmazione, dato il maggior numero di possibili strumenti da utilizzare. Si fa riferimento alla sezione di *Mathworks* che si occupa della parte del *Model Based Design* in [**15**].

5.1.1 Stateflow

É una *toolboxdiMatlab* che permette la modellazione e la simulazione di macchine a stati e diagrammi di flusso. Stateflow viene integrato in uno schema di Simulink,

come verrà visualizzato nelle prossime pagine. In altre parole, a parte della categoria dei linguaggi basati sul Model Based Design, con la peculiarità rispetto ad altri linguaggi di basarsi sulla notazione di macchina a stati finiti (automa), cioè stati che dopo un certo lasso di tempo (dipendente dalle condizioni che legano ciascuna parte) passano ad altri stati senza mai fermarsi all'infinito sullo stesso. Come verrà spiegato nelle prossime sezioni, ogni stato si occuperà di azioni di volo differenti, le quali legate da determinate condizioni (transizioni) formeranno il Flowchart complessivo, rappresentato in Fig. (5.2).

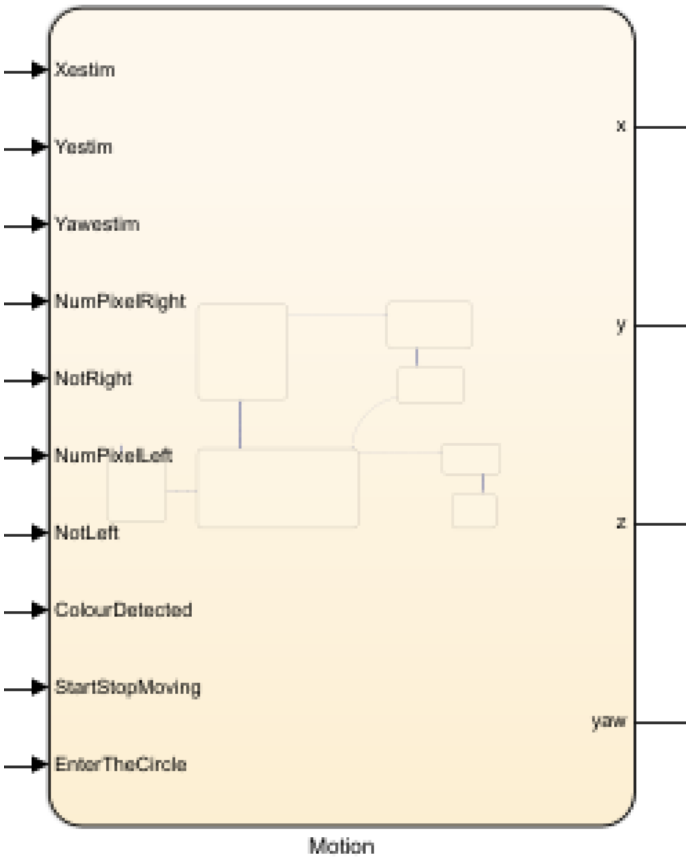

Figura 5.2: Blocco Flowchart realizzato nel Control System

5.2 Blocco Path Planning

Per arrivare al percorso esatto dove realizzare il codice in linguaggio Stateflow, subentrare come primo passaggio nel blocco principale *ControlSystem*, dopodiché subentrare nuovamente nel blocco denominato *PathPlanning* che porterà alla schermata di Fig. (5.3), ovvero la destinazione desiderata.

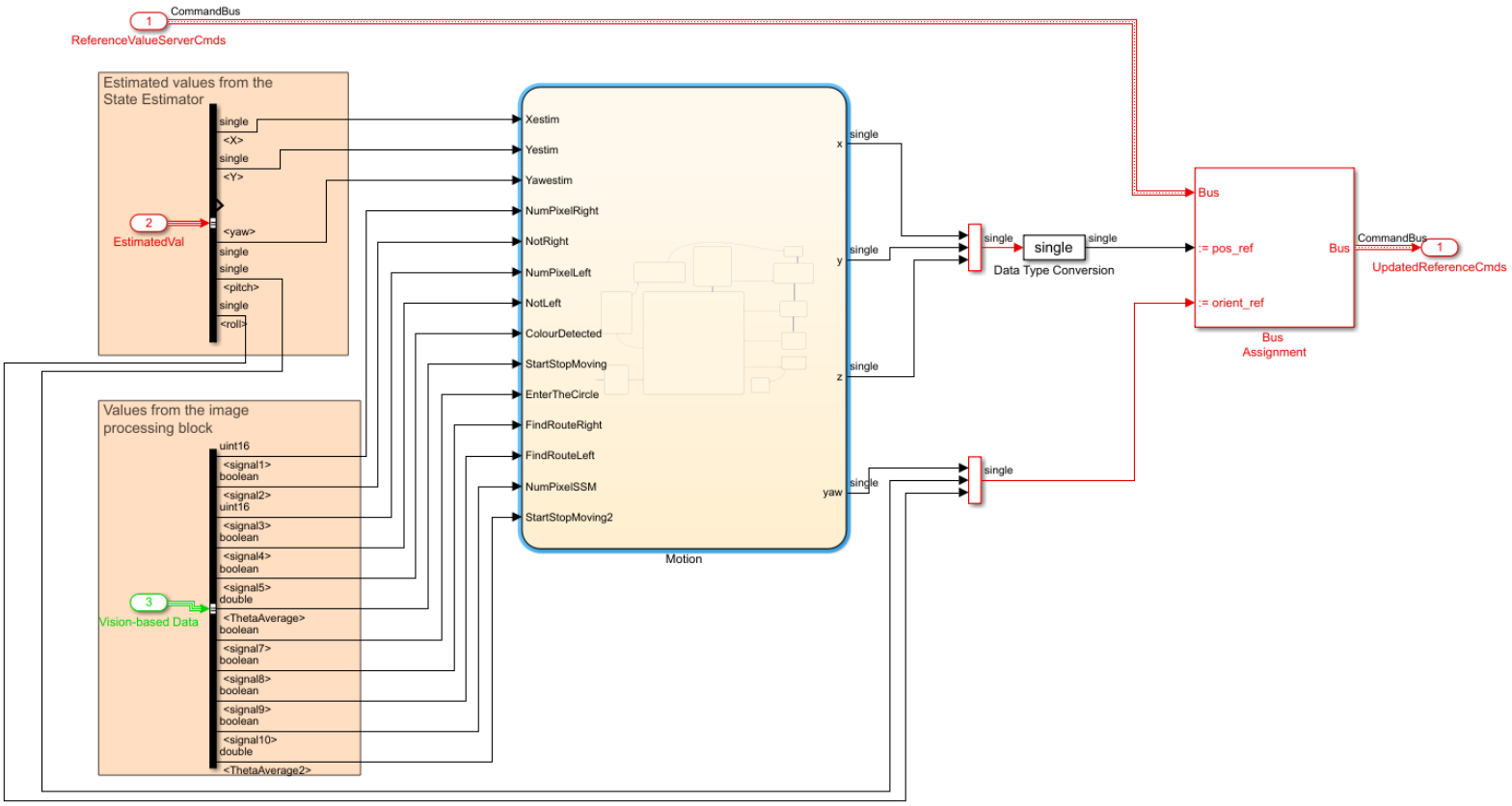

Figura 5.3: Blocco Path Planning

Si osservano chiaramente due riquadri arancioni che rappresentano gli ingressi derivanti rispettivamente dai sensori, ovvero $EstimatedVal$, e le sottomatrici rappresentate nel capitolo precedente in Fig. (4.7), ognuna con importanza specifica nella realizzazione delle condizioni e delle transizioni, e la cui presenza risulta obbligatoria dato che lo scopo del progetto è l'inseguimento della traiettoria sfruttando la teleca-

mera del minidrone, altresì $Vision-basedData$. Tutte le informazioni derivanti dal Flowchart realizzato e denominato $Motion$, vengono infine passate ad un Bus dati che le raggrupperá ciclicamente per poi passarle agli attuatori per essere elaborate. Si analizza infine nello specifico il blocco $Motion$, partendo da una prima illustrazione in Fig. (5.3) per arrivare a descrivere singolarmente ciascuna fase componente la struttura.

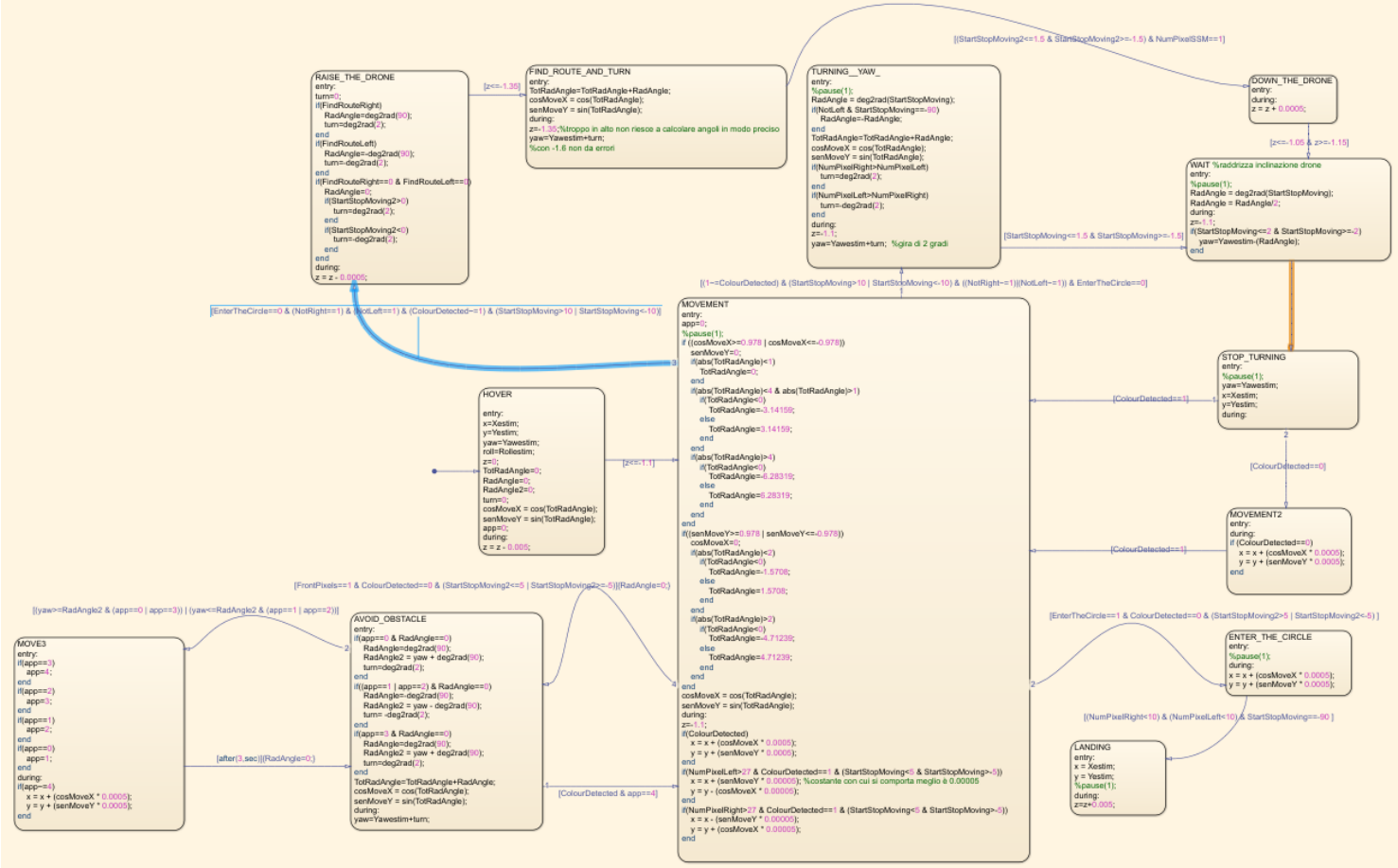

Figura 5.4: Blocco Motion

Gli stati presenti all'interno della composizione sono i seguenti:

- $Hover$, che rappresenta la fase di salita (decollo) del minidrone;
- $Movement$, che esprime la fase di spostamento rettilineo lungo un tratto del percorso;

- $Turning\_Yaw\_$, che segna la fase di rotazione lungo l'asse Z del minidrone;

- $Wait$, ovvero la fase di attesa prima della ripartenza lungo un nuovo tratto del percorso individuato;

- $Stop\_Turning$, che serve all'aggiornamento di determinate variabili locali terminata la fase di rotazione in Z;

- $Enter\_The\_Circle$, che rappresenta la fase di approccio alla parte terminale del percorso, rappresentata dal cerchio descritto in sezione (3.1);

- $Landing$, ovvero l'atterraggio, cioé l'ultima fase di volo dell'APR;

- $Raise\_The\_Drone$, che rappresenta la fase in cui si presenta una discontinuità nel percorso da seguire, che deve essere riconosciuta e seguita, nel mentre che si alza il drone per trovare dove continua il percorso;

- $Find\_Route\_And\_Turn$, impiegata alla rotazione del drone dopo aver riconosciuto una discontinuitá;

- $Down\_The\_Drone$, per riportare il drone all'altezza predefinita di 1.1 metri;

- $Movement2$, che serve per seguire il piccolo tratto di percorso di discontinuità individuato;

- $Avoid\_Obstacle$, utile alla gestione di ostacoli che si possono incontrare nel corso del percorso, in particolare per effettuare le rotazioni intorno all'ostacolo;

- $Move3$, che insieme allo stato precedente serve per completare il movimento, ovvero gli spostamenti rettilinei intorno all'ostacolo, per poi ricominciare il percorso;

Si procede ora all'analisi logica della composizione in singoli parti.

5.3 Fasi del movimento

5.3.1 Hover

Stato che controlla il decollo del mini drone, che si occupa della semplice salita da terra di quest'ultimo, e l'inizializzazione di una parte delle variabili locali, Fig. (5.5).

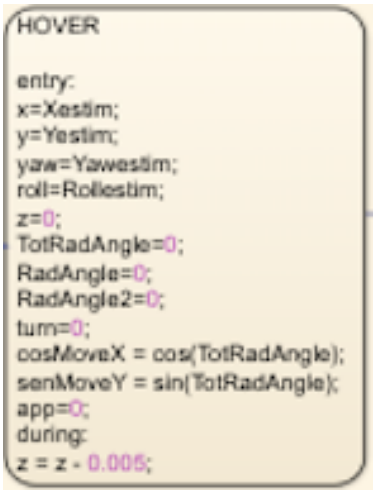

Figura 5.5: Fase di decollo

Nella fase *entry* vengono inizializzate le seguenti variabili:

- variabile x, inizializzata al valore fornito dai sensori della posizione in X rispetto il sistema assoluto, cioè della posizione in X iniziale;
- variabile y, con le stesse considerazioni della variabile precedente, con asse Y.
- variabile yaw, inizializzata al valore dell'angolo letto dall' IMU rispetto il sistema di riferimento solidale col baricentro del drone, che serve quindi nella fase di rotazione intorno all'asse Z.
- variabile z, che rappresenta la posizione verticale del drone, ponendola al valore 0 che indica il posizionamento a terra del minidrone.
- variabile $TotRadAngle$, che serve per la memorizzazione della somma degli angoli formati tra diversi segmenti del percorso;

- variabile *RadAngle* che serve per la memorizzazione dell'angolo calcolato del prossimo segmento che si presenta e che il drone dovrà percorrere terminata la rotazione;

- variabile *turn*, che verrà posta sempre al valore di 2 radianti, con lo scopo di modificare ciclicamente il valore di yaw fino alla raggiunta di un'angolatura pressoché nulla del segmento successivo;

- variabile $cosMoveX$, fondamentale alla memorizzazione del contributo totale rispetto cui il minidrone deve spostarsi nella direzione X;

- variabile $senMoveY$, come il precedente, ma con contributo riguardante gli spostamenti nella direzione Y;

- variabile *app*, utile solo nella gestione dell'ostacolo in aria.

Per quanto riguarda il corpo del blocco, cioè la fase *during*, l'unica operazione che deve svolgere, come anticipato, é la diminuzione del contributo in Z, tale al fine di alzare da terra ciclicamente il mini drone di un piccolo contributo fino ad un'altezza di 1.1 metri, (specifica della competizione), momento in cui si attiverá la transizione di Fig. (5.6), portando il sistema al nuovo stato $Movement$.

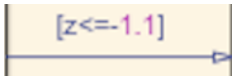

Figura 5.6: Transizione che porta dallo stato di Hover a quello di Movement

5.3.2 Movement

Stato che controlla il movimento lungo ciascuno dei segmenti rettilinei che compongono il percorso, con funzionamento descritto in Fig. (5.8). La fase *entry*, ha come ultimo compito gestire eventuali condizioni di verifica per eliminare l'incertezza che si può presentare dalla lettura dell'angolo con la telecamera, e quindi eventualmente andando ad agire sulla variabile che memorizza l'angolo totale in TotRadAngle. In particolare, effettuando delle verifiche sui valori di cosMoveX e senMoveY, con i

valori numerici indicati opportunamente nell'illustrazione, ovvero multipli di angoli di 90° o nulli rispetto il sistema di riferimento assoluto del drone. Mentre la fase *during* descrive una logica semplice che ottiene buoni risultati. Come primo aspetto, si mantiene costante l'altezza da terra del drone ad 1.1 metri, come richiesto dalle specifiche; successivamente la prima condizione, basandosi sulla sottomatrice di colore verde chiaro $ColourDetected$ di Fig. (4.7), permette lo spostamento rettilineo del drone fino a quando la fotocamera rileva nella posizione relativa a quella sottomatrice almeno 7 pixel di traiettoria su 9. Le due successive condizioni servono invece come aggiustamento a probabili piccoli errori che si possono presentare nel corso dell'esecuzione, dato che talvolta, l'angolo rilevato dalla fotocamera può non essere perfettamente identico ma avere un'incertezza fino a 5 radianti; in questo modo durante lo spostamento rettilineo dell'APR se quest'ultimo dovesse cominciare ad allontanarsi dalla mediana relativa ai due lati che comprendono la traiettoria da percorrere le condizioni provvedono a riportare con piccoli contributi il mini drone verso la posizione della mediana precitata. Questo stato termina la sua esecuzione nel momento in cui la transizione di Fig. (5.7) viene verificata.

Figura 5.7: Transizione che porta dallo stato di Movement a quello di Turning

Questa specifica transizione si attiva nel momento in cui il drone arrivato alla fine del segmento rileva meno di 7 pixel su 9; ciò significa che da quel punto inizia una nuova tratta con un'angolazione diversa dalla precedente, quindi si deve procedere alla modifica dell'angolo di yaw del minidrone, riportandolo in posizione rettilinea rispetto la nuova tratta. Le altre condizioni presenti nella transizione sono secondarie, e servono per rendere il passaggio tra gli stati mutualmente esclusivo, evitando che possa entrare in stati differenti. In particolare, con la variabile $StartStopMoving$, si provvede a verificare se l'angolo rilevato dalla fotocamera nella sottomatrice gialla di Fig. (4.7) sia maggiore di 1 in modulo, evitando così possibili incertezze; le variabili $NotRight$ e $NotLeft$, relative alle matrici di colore azzurro di Fig. (4.7)

invece verificano che effettivamente ci sia un cambio di direzione, infine la variabile $EnterTheCircle$ ci assicura l'assenza del cerchio che indica la parte terminale del percorso.

Questa specifica transizione porta il sistema verso lo stato di $Turning\_Yaw\_$.

```
MOVEMENT
entry:
app=0;
%pause(1);
if ((cosMoveX>=0.978 | cosMoveX<=-0.978))
   senMoveY=0;
   if(abs(TotRadAngle)<1)
      TotRadAngle=0;
   end
   if(abs(TotRadAngle)<4 & abs(TotRadAngle)>1)
      if(TotRadAngle<0)
         TotRadAngle=-3.14159;
      else
         TotRadAngle=3.14159;
      end
   end
   if(abs(TotRadAngle)>4)
      if(TotRadAngle<0)
         TotRadAngle=-6.28319;
      else
         TotRadAngle=6.28319;
      end
   end
end
if((senMoveY>=0.978 | senMoveY<=-0.978))
   cosMoveX=0;
   if(abs(TotRadAngle)<2)
      if(TotRadAngle<0)
         TotRadAngle=-1.5708;
      else
         TotRadAngle=1.5708;
      end
   end
   if(abs(TotRadAngle)>2)
      if(TotRadAngle<0)
         TotRadAngle=-4.71239;
      else
         TotRadAngle=4.71239;
      end
   end
end
cosMoveX = cos(TotRadAngle);
senMoveY = sin(TotRadAngle);
during:
z=-1.1;
if(ColourDetected)
   x = x + (cosMoveX * 0.0005);
   y = y + (senMoveY * 0.0005);
end
if(NumPixelLeft>27 & ColourDetected==1 & (StartStopMoving<5 & StartStopMoving>-5))
   x = x + (senMoveY * 0.00005); %costante con cui si comporta meglio è 0.00005
   y = y - (cosMoveX * 0.00005);
end
if(NumPixelRight>27 & ColourDetected==1 & (StartStopMoving<5 & StartStopMoving>-5))
   x = x - (senMoveY * 0.00005);
   y = y + (cosMoveX * 0.00005);
end
```

Figura 5.8: Fase di spostamento

5.3.3 Turning_Yaw_

Stato che controlla la rotazione rispetto l'asse verticale Z, descritto dall'angolo di yaw, il tutto mostrato in Fig. (5.9).

```
TURNING__YAW_
entry:
%pause(1);
RadAngle = deg2rad(StartStopMoving);
if(NotLeft & StartStopMoving==-90)
   RadAngle=-RadAngle;
end
TotRadAngle=TotRadAngle+RadAngle;
cosMoveX = cos(TotRadAngle);
senMoveY = sin(TotRadAngle);
if(NumPixelRight>NumPixelLeft)
   turn=deg2rad(2);
end
if(NumPixelLeft>NumPixelRight)
   turn=-deg2rad(2);
end
during:
z=-1.1;
yaw=Yawestim+turn;  %gira di 2 gradi
```

Figura 5.9: Fase di rotazione rispetto asse Z

Nella fase di *entry* si procede con la memorizzazione dell'angolo rilevato del nuovo segmento di percorso, e alla traduzione di quest'ultimo in radianti, data la circostanza del sistema che controlla le variabili di yaw, pitch, roll in radianti piuttosto che in gradi. Successivamente si verifica la direzione di un possibile angolo retto, dato che la Trasformata di Hough contiene un insieme di valori analizzati nella sottosezione (4.3.2), e che quindi il valore di $-90°$ rappresenta angoli retti in entrambe le direzioni. Successivamente, dopo la modifica delle variabili d'appoggio per le successive fasi volo, si procede a modificare l'angolo di yaw di 2 radianti alla volta, fino alla posizione finale che attiva la transizione di Fig. (5.10).

Figura 5.10: Transizione che porta dallo stato di Turning a quello di Wait

Intuitivamente, l'illustrazione mostra che la transizione si attiva quando la matrice gialla di Fig. (4.7), dopo la rotazione raggiunge una angolatura tra i due angoli descritti, considerando quindi un margine di incertezza molto piccolo, e con la consapevolezza che il drone si troverà in una posizione pressoché rettilinea. Questa transizione porta a due nuovi stati, ovvero *Waiting* e *Stop_Turning*, che essenzialmente fungono da piccola pausa nello spostamento del codice con qualche piccolo accorgimento aggiuntivo.

5.3.4 Waiting e Stop_Turning

Stati che come descritto nelle righe precedenti hanno la funzione di sospendere i movimenti del drone per istanti brevissimi prima di ricominciare il movimento seguendo un nuovo segmento del percorso, quindi sono dei buoni mezzi per ottenere dei risultati migliori, considerando che la fotocamera, per velocità troppo grandi di movimento, potrebbe riscontrare maggiori problemi e di conseguenza valutazione errata dei dati dalla fotocamera. Si rappresentano i due stati in Fig. (5.11).

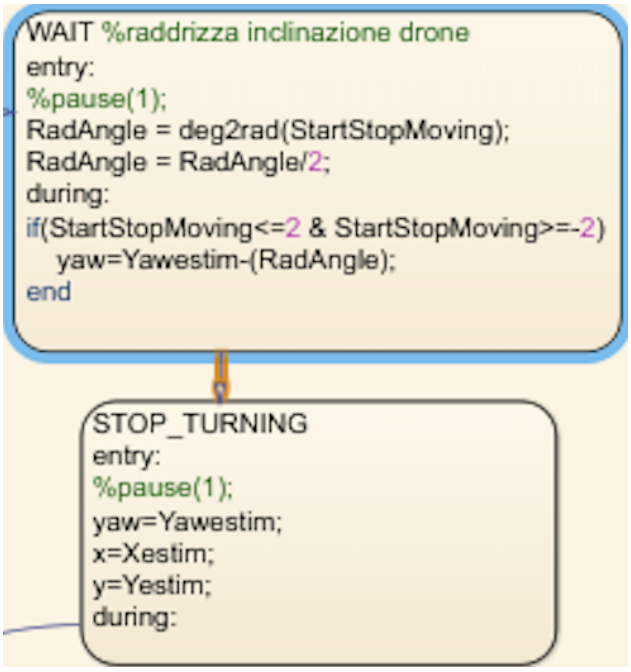

Figura 5.11: Fasi di attesa e di aggiornamento delle variabili locali

In dettaglio, nel primo stato si aggiusta lievemente un possibile piccolo disorientamento lasciato durante la rotazione nello stato precedente, raddrizzando quindi di una piccola quantità aggiuntiva il minidrone. Per quanto riguarda invece il secondo stato, come si osserva visivamente, serve come già anticipato per l'aggiornamento delle variabili locali, utili all'inizio di un nuovo segmento di volo. Al termine di questi due stati la transizione si attiva all'istante con la condizione di figura, come si osserva in Fig. (5.12), e si ricomincia con una nuova tratta.

Figura 5.12: Transition che porta da Stop_Turning a Movemen

Questa condizione serve solo per rendere mutualmente esclusiva l'uscita da questo stato, differenziando la rotazione normale di un nuovo segmento dalla rotazione che si verifica con una discontinuitá.

Gli stati descritti in queste prime sezioni si ripeteranno ciclicamente, seguendo tutta la traiettoria del percorso messo a disposizione dalla competizione, fino a quando il minidrone si troverà ad affrontare una delle seguenti possibilità:

- *atterraggio*, (5.3.5), ovvero il minidrone si trova nelle vicinanze del cerchio, che indica la fine del percorso, attraverso la transizione di Fig. (5.13) che porta dallo stato di *Movement* a quello di *Enter_The_Circle*.

Figura 5.13: Transizione dallo stato di Movement a quello di Enter_The_Circle

Le condizioni utilizzate che permettono di distinguere la fine del percorso sfruttano l'impiego delle sottomatrici giá illustrate ripetutamente nel corso delle

sezioni.

- *discontinuitá di percorso*, in sotto sezione (5.3.7), ovvero la mancata presenza di un eventuale piccolo segmento nel corso del percorso come si osserva nell'immagine in Fig. (**??**), e raggiungibile dalla transizione di Fig. (5.14).

Figura 5.14: Transizione dallo stato di Movement a quello di Raise_The_Drone

La differenza che lo rende esclusivo rispetto quello di Fig. (5.15), è la variabile EnterTheCircle.

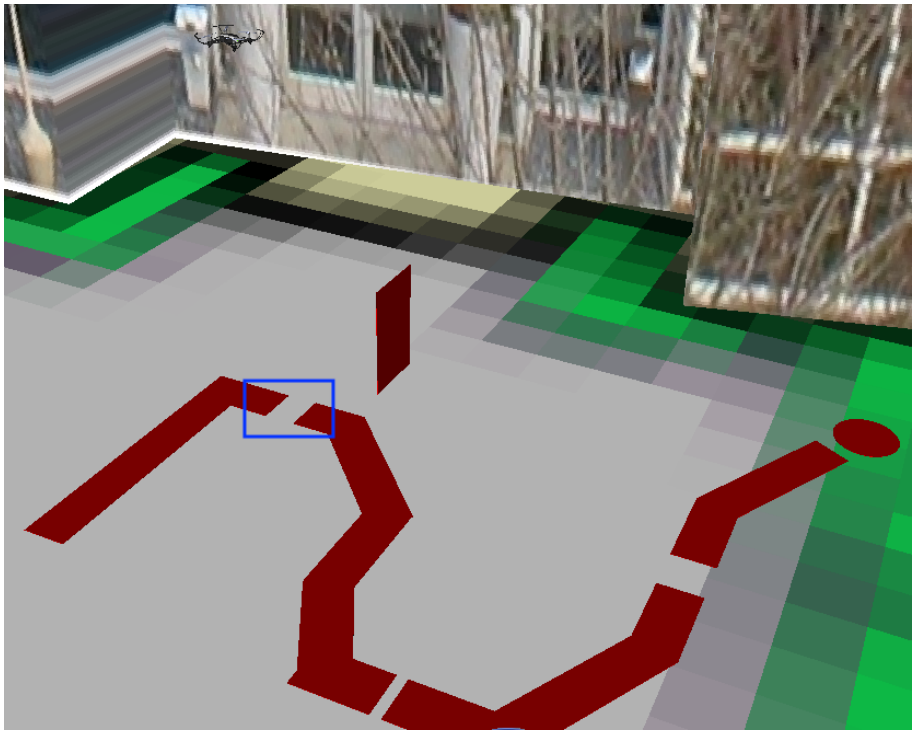

Figura 5.15: Illustrazione di un esempio di interruzione di percorso

- *ostacolo in aria*, in (5.3.11), ovvero la possibile presenza di un ostacolo da evitare nel percorso, con un esempio esplicativo in Fig. (5.17), e raggiungibile dalla transizione di Fig. (5.16).

```
[FrontPixels==1 & ColourDetected==0 & (StartStopMoving2<=5 | StartStopMoving2>=-5)]{RadAngle=0;}
```

Figura 5.16: Transizione dallo stato di Movement a quello di Avoid_Obstacle

Si osserva nuovamente che la differenza dalle altre due sta nell'utilizzo della matrice che calcola l'angolo di inclinazione della tratta appena successiva all'ostacolo, individuata da una leggerissima discontinuità necessaria per la mancanza di telecamere anteriori al drone per la gestione di un controllo in maniera differente.

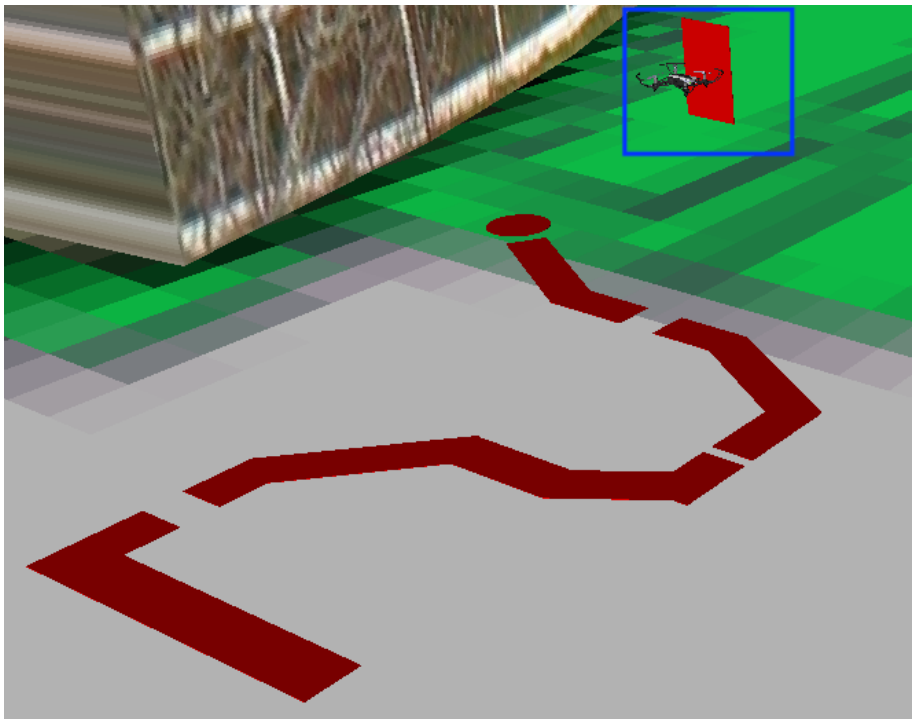

Figura 5.17: Illustrazione di un esempio di ostacolo nel percorso

Si analizzano ora gli stati raggiungibili dalle tre transizioni poc'anzi discusse.

5.3.5 Enter_The_Circle

Stato che serve al minidrone per completare lo spostamento prima dell'atterraggio e raggiungere quindi un'area soprastante il cerchio di fine percorso. In particolare

deve effettuare gli spostamenti anche se non viene rilevato il colore dalla sotto matrice ColourDetected, Fig. (5.18)

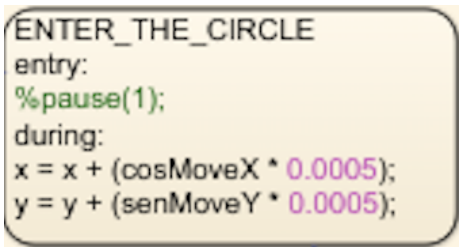

Figura 5.18: Transizione dallo stato di Movement a quello di Enter_The_Circle

Questo stato continua fino quando non si attiva la transizione di Fig. (5.19)

Figura 5.19: Transizione dallo stato di Movement a quello di Enter_The_Circle

Si osserva che la transizione si attiva quando l'angolo rilevato dalla sotto matrice StartStopMoving é uguale a ang-90, ovvero quando non conterrà più sezioni di percorso, simboleggiando che il drone si trova con la testa alla fine del cerchio di atterraggio, con le condizioni restanti utili ad un controllo più preciso. A quel punto si entra nella vera e propria fase di atterraggio.

5.3.6 Landing

Questa fase serve molto semplicemente a modificare il valore del contributo nell'asse verticale Z, in modo opposto rispetto il decollo, per compiere l'abbassamento del mini drone e completare la competizione, Fig. (5.20).

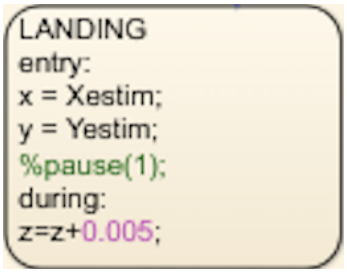

Figura 5.20: Transizione dallo stato di Enter_The_Circle a quello di Landing

Si passa ora alla descrizione del caso di una presenza di discontinuità nel percorso.

5.3.7 Raise_The_Drone

Nella fase di *entry* si procede alla definizione di certe variabili, che descrivono la posizione individuata dove il percorso ricomincia dopo l'interruzione momentanea, per poi andare ad alzare l'altezza del drone per favorire la lettura dalla fotocamera nel caso in cui il percorso sia eccessivamente distante. É importante considerare il fatto di non poter aumentare eccessivamente l'altezza dal suolo del radiocomandato altrimenti la fotocamera in dotazione per la competizione non otterrebbe più risultati soddisfacenti nel calcolo delle angolature, (5.21).

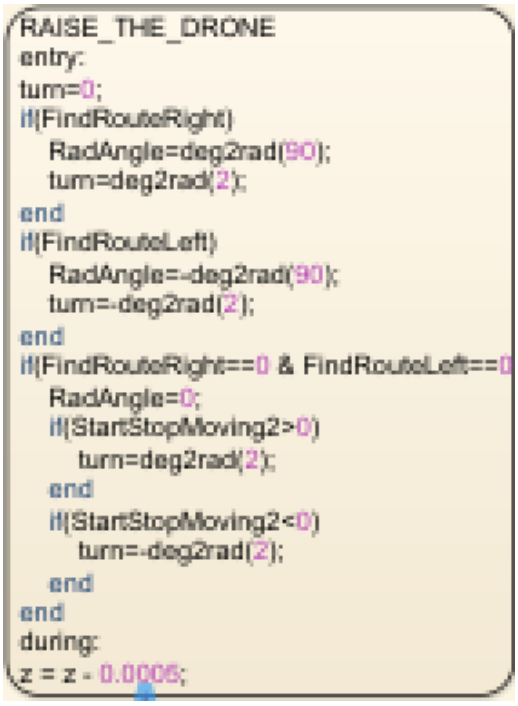

Figura 5.21: Stato di Raise_The_Drone

questo stato si eseguirà fino al raggiungimento di un'altezza di circa 1.35 metri dal suolo, come si osserva in Fig. (5.22), che porta direttamente allo stato successivo di rotazione rispetto la nuova traiettoria.

Figura 5.22: Transizione dallo stato di Raise_The_Drone a quello di Find_Route_And_Turn

5.3.8 Find_Route_And_Turn

Nella fase di *entry* procede alla modica dell'angolo rilevato del nuovo segmento di percorso, per poi ruotare il minidrone della quantità desiderata mantenendo la nuova altezza raggiunta, Fig. (5.23)

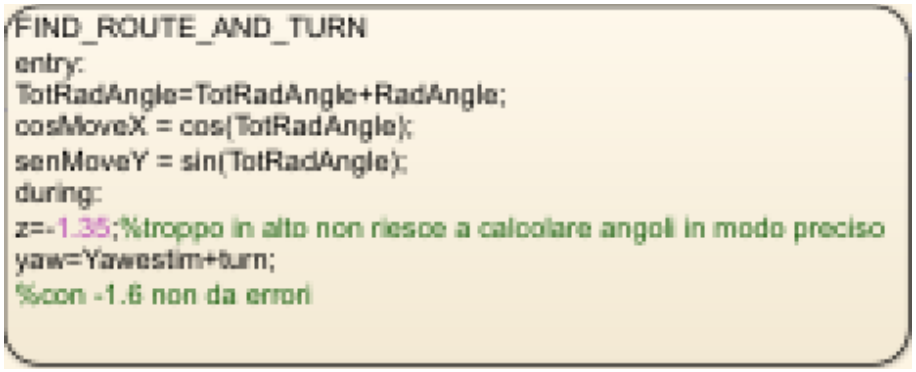

Figura 5.23: Stato di Find_Route_And_Turn

Al termine della rotazione, si attiva la transizione seguente, Fig. (5.24).

Figura 5.24: Transizione dallo stato di Find_Route_And_Turn a quello di Down_The_Drone

La condizione ulteriore descritta dalla variabile $NumPixelSSM$ booleana si attiva quando pixel rilevati dalla matrice che rileva l'angolo sono di un numero superiore a diverse centinaia, per assicurare la corretta esecuzione della rotazione, fermandosi in linea parallela alla mediana del nuovo segmento, evitando così possibili errori iniziali quando la sotto matrice comincia a rilevare il percorso. Successivamente si arriva allo stato di *Down_The_Drone*.

5.3.9 Down_The_Drone

Unico e semplice compito di questo stato é di riportare l'altezza dal suolo alle specifiche standard della competizione di 1.1 metri e continuare il percorso, Fig. (5.25).

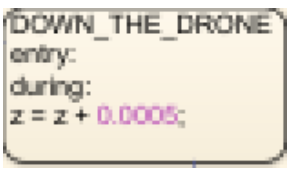

Figura 5.25: Stato di Down_The_Drone

Si passa poi al nuovo stato raggiungendo l'altezza standard come annunciato in precedenza, per poi entrare nello stato di *Wait* e di *Stop_Turning* già descritti nelle pagine precedenti. Successivamente però prima di tornare allo stato principale di Movement si è realizzato uno stato necessario al completamento della tratta senza rilevamento del percorso, dato che la sotto matrice ColourDetected non rileverebbe alcun percorso per un lasso di tempo finito. Quindi passando dalla transizione di Fig. (5.26), si arriva allo stato di *Movement*2.

Figura 5.26: Transizione dallo stato di Stop_Turning a quello di Movement2

5.3.10 Movement2

Provvede solamente allo spostamento in linea rettilinea rispetto il nuovo segmento fino a quando non viene rilevata la nuova tratta per poi tornare allo stato principale di Movement e continuare lo spostamento.

Si passa ora all'analisi dell'ultimo caso che si può verificare, ovvero quello della presenza di un ostacolo in aria.

5.3.11 Avoid_Obstacle

Nella fase di *entry* si effettuano delle verifiche riguardanti il numero di rotazioni intorno all'ostacolo che servono per evitarlo, in particolare la variabile *app*, nel corso delle rotazioni assume i valori da uno a quattro progressivamente, indicanti il numero di rotazione effettuata, per poi compiere uno spostamento rettilineo e ripetere la procedura fino al completamento dell'ovviazione dell'ostacolo. Si osserva in particolare che le rotazioni si compiono esattamente di $90°$ o $-90°$ gradi per rendere l'esecuzione più pulita e precisa. Nella fase di *during* quindi si procede alla rotazione effettiva del drone variando l'angolo di yaw, come mostrato in Fig. (5.27)

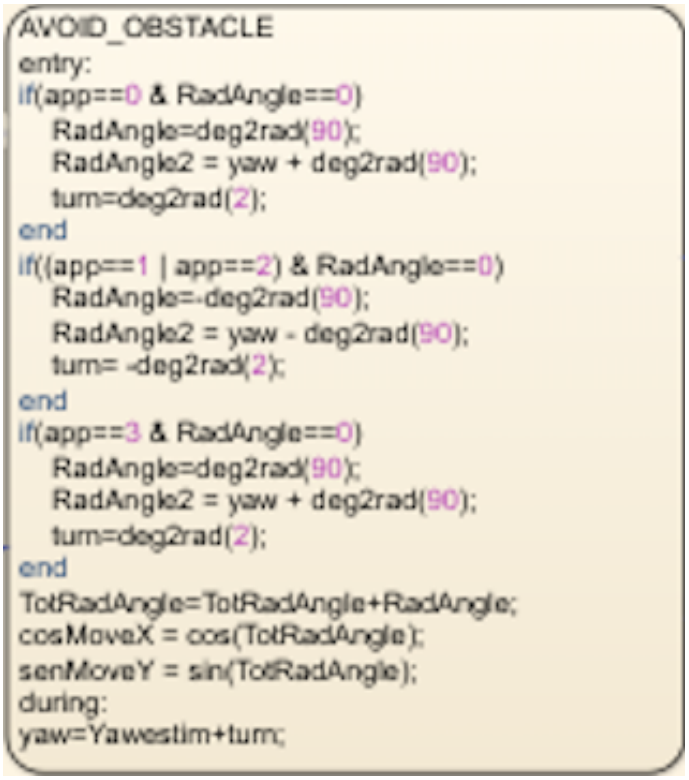

Figura 5.27: Stato di Avoid_Obstacle

Questo stato è complementare a quello direttamente collegato ad esso e denominato $Move3$, che serve per effettuare lo spostamento dopo aver compiuto la rotazione di un angolo retto intorno all'ostacolo, a cui si arriva attraverso la transizione di FIg. (5.28)

Figura 5.28: Transizione dallo stato di Avoid_Obstacle a quello di Move3

Si osserva che si attiva nel momento in cui il valore dell'angolo di yaw che si ottiene girando il minidrone arriva ad essere maggiore di 90° rispetto il vecchio valore aumentato di 90°, se ruota in senso orario, altrimenti di $-90°$ se ruota in senso antiorario. L'interpretazione del giro in senso orario o antiorario viene gestita attraverso la variabile *app*, con interpretazione diretta e intuitiva. Si arriva infine allo stato di Move3 come anticipato, e illustrato in Fig. (5.29).

Figura 5.29: Stato di Move3

Nella fase di *entry* si procede semplicemente all'incremento della variabile *app*, mentre nella fase di *during* si mette il minidrone in movimento come già descritto, fino a quando passano 3 secondi, tornando cosí alla fase di rotazione intorno all'ostacolo, processo con durata di 4 cicli, Fig. (5.30)

Figura 5.30: Transizione dallo stato di Move3 a quello di Avoid_Obstacle

Con questa ultima transizione si conclude la descrizione del codice studiato e successivamente progettato.

Capitolo 6

Conclusioni

Grazie a questo progetto è stato possibile muoversi all' interno dell'ambiente di lavoro *Simulink*, e allo stesso tempo dell' applicativo che lo implementa, ovvero *Matlab*. In particolare, questa tesi è servita all' apprendimento generale e specifico della tecnica di *Model Based Design* la quale in questo ambito di sviluppo, ovvero algoritmi di controllo per droni, si traduce in linguaggio di *Stateflow*. Tutti i test effettuati hanno portato dei risultati soddisfacenti nella gestione delle dinamiche di volo, adattandosi a diverse strutture di percorsi spesso molto differenti tra loro. Per quanto riguarda possibili upgrade del progetto, sicuramente un primo passo sarebbe la prova effettiva del codice realizzato sul drone fisico, il che presenta ovviamente problematiche ben più rilevanti di una simulazione con il programma. Successivamente si potrebbe ricorrere a funzioni e logiche alternative al conseguimento degli obiettivi proposti dalla competizione o al miglioramento di quelli già realizzati. In conclusione, si puó affermare che questo elaborato é servito all' apprendimento di molteplici aspetti, tra cui leggi matematiche che regolano la dinamica e la cinematica di un APR, un' infarinata generale sugli argomenti riguardanti lo studio e la manipolazione di immagini, e il miglioramento delle capacitá di programmazione attraverso lo sviluppo delle dinamiche di volo. Si puó affermare certamente che l' ambito di studio di questa tesi sia di grande attualitá e interesse, considerando che gli ambienti i quali scopriranno e sfrutteranno questa tecnologia diventeranno sempre piú vasti nel tempo a venire, quindi é un concreto punto di partenza per chi vuole immergersi nel mondo della robotica e dello studio degli automi.

Bibliografia

[1] Parrot,https://www.parrot.com/it/droni/parrot-mambo-fpv

[2] Mathworks,https://it.mathworks.com/products/stateflow.html

[3] Mathworks,https://it.mathworks.com/help/stateflow/ug/state-transition-tables-in-stateflow.html

[4] Wikipedia,https://it.wikipedia.org/wiki/Accumulatore_litio-polimero

[5] Recensioni Droni,https://recensionidroni.com/

[6] Adamant-Namiki,https://adamant-namiki.eu/motore-coreless-come-funziona/

[7] Wil Selby,https://www.wilselby.com/research/arducopter/modeling/

[8] Mathworks,https://it.mathworks.com/videos/series/mathworks-minidrone-competition.html

[9] Mathworks,https://it.mathworks.com/videos/series/mathworks-minidrone-competition.html

[10] Mathworks,https://it.mathworks.com/help/images/ref/im2bw.html

[11] Mathworks,https://it.mathworks.com/help/images/ref/edge.html

[12] Mathworks,https://it.mathworks.com/help/images/edge-detection.html

[13] Mathworks,https://it.mathworks.com/help/images/ref/hough.html

[14] Mathworks,https://it.mathworks.com/help/images/ref/houghpeaks.html

[15] Mathworks,https://it.mathworks.com/solutions/model-based-design.html

[16] Roberto-Bucher,http://robertobucher.dti.supsi.ch/wp-content/uploads/2017/03/stateflow.pdf

[17] Mary K. Pratt, Digital Image Processing

[18] Alasdair McAndrew, An introducing to digital image Processing with Matlab Notes