
UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI INGEGNERIA

Corso di Laurea Triennale in

INGEGNERIA INFORMATICA ED ELETTRONICA

# Studio e identificazione di modelli per la dinamica di un quadrirotore

Relatore:

**Prof. Paolo Valigi**

Correlatore:

**Ing. Mirko Leomanni**

Candidato:

**Giovanni Versiglioni**

ANNO ACCADEMICO 2022/2023

a Camilla

# SOMMARIO

Il quadrirotore è la piattaforma aerea maggiormente utilizzata ai fini di ricerca sui velivoli autonomi per la sue caratteristiche dinamiche che gli conferiscono un'ottima manovrabilità. Inoltre, sebbene sia instabile a ciclo aperto (come la maggior parte dei velivoli ad ala rotante), il quadrirotore mostra un buon grado di disaccoppiamento, cioè l'azione su uno dei segnali di controllo influisce quasi esclusivamente sul corrispondente grado di libertà.

La crescente complessità dei sistemi di controllo per questo tipo di mezzi necessita di modelli matematici estremamente accurati per la progettazione e la simulazione di relative leggi di controllo.

Lo scopo della seguente tesi è duplice. Il primo è quello di presentare un modello matematico che descriva rigorosamente la dinamica di un quadrirotore. Il secondo è quello di identificare un modello sufficientemente accurato per via sperimentale secondo un approccio a scatola nera, cioè esclusivamente sulla conoscenza di dati input-output. Nello specifico, sfruttando la proprietà di disaccoppiamento, si cercherà di identificare delle funzioni di trasferimento tempo continue in grado di descrivere il movimento del drone longitudinale, laterale e verticale (cioè lungo gli assi principali).

**Parole Chiave** *Unmanned Aerial Vehicle, Quadcopter, Quaternion, Robot Operating System, Gazebo, ArduCopter, MAVROS, System Identification, Black Box*

# INDICE

# LISTA DELLE FIGURE

# ACRONIMI

| | |
|---|---|
| AGV | Automated Guided Vehicle |
| AMR | Autonomous Mobile Robot |
| UAV | Unmanned Aerial Vehicle |
| DoF | Degree of Freedom |
| LTP | Local Tangent Plane |
| ENU | East North Up |
| NED | North East Down |
| ABC | Aircraft Body Center |
| DCM | Direct Cosine Matrix |
| ROS | Robot Operating System |
| API | Application Programming Interface |
| DNS | Domain Name System |
| RPC | Remote Procedure Call |
| SLERP | Spherical Linear Interpolation |
| RViz | ROS Visualization |
| SITL | Software In The Loop |
| SISO | Single Input Single Output |
| MIMO | Multi Input Multi Output |
| LTI | Linear Time Invariant |
| NRMSE | Normalized Root Mean Square Error |

# 1

# INTRODUZIONE

Nell'ultimo decennio l'industria robotica è cresciuta anno dopo anno, assumendo un ruolo decisivo nel più ampio campo dell'industria tecnologica. Secondo il rapporto "State of the robotics market" condotto da ABI Research [1], nel 2018 l'investimento totale nel settore della robotica è stato di circa 4 miliardi di dollari (US$).

In particolare, la robotica mobile rappresenta uno dei trend maggiormente in crescita, con i veicoli a guida automatica - Automated Guided Vehicle (AGV) - che ne rappresentano un ruolo chiave nel mercato. Questi ultimi sono utilizzati principalmente in campo industriale per la movimentazione di prodotti all'interno di uno stabilimento. Tra le aziende leader di questo mercato primeggia Amazon, con investimenti continui volti a migliorare la produttività dei propri centri di distribuzione.

ABI sostiene che nel prossimo futuro gli AGV continueranno ad avere un ruolo chiave nel settore della robotica mobile, ma prevede che i robot mobili autonomi - Autonomous Mobile Robot (AMR) - li supereranno (in numero) entro il 2030. Questa ultima tipologia di robot mobili è caratterizzata da automi in grado di comprendere e spostarsi nell'ambiente esterno senza essere gestiti direttamente da un operatore.

In questo contesto, gli aeromobili a pilotaggio remoto - Unmanned Aerial Vehicle (UAV) o droni - rappresentano un settore in via di sviluppo e trovano riscontro in un numero crescente di applicazioni: dal settore militare al settore civile, dal settore commerciale al settore dell'intrattenimento.

### BREVE SOMMARIO

L'elaborato si articola nei seguenti cinque capitoli:

- **Capitolo 1** introduce la robotica mobile e le relative classificazioni
- **Capitolo 2** espone nel dettaglio la classe dei droni (UAV), quindi descrive il modello fisico-matematico e il controllo di un quadrirotore
- **Capitolo 3** introduce ROS e ne spiega il funzionamento entrando nel dettaglio dei vari livelli che lo costituiscono, quindi descrive l'architettura utilizzata per simulare il volo di un quadrirotore
- **Capitolo 4** riporta le simulazioni di volo effettuate e i relativi dati input-output registrati, quindi spiega il procedimento seguito per ottenere modelli matematici SISO che legano gli ingressi e le uscite fortemente dipendenti tra loro (e.g. rollio e posizione lungo y, beccheggio e posizione lungo x, spinta e posizione lungo z)
- **Capitolo 5** discute i risultati ottenuti e presenta eventuali sviluppi futuri

# 2

# UAV

Un aeromobile a pilotaggio remoto (UAV) è un velivolo caratterizzato dall'assenza di un pilota umano a bordo. Il suo volo può essere controllato da un computer a bordo del mezzo oppure da remoto, seguendo i comandi di un pilota a terra.

## 2.1 CLASSIFICAZIONE

In base al numero e alla configurazione dei rotori si possono distinguere diverse tipologie di droni [2], come mostrato in Figura 2.1.

Il modello di drone più semplice è quello a singolo rotore. Presenta un solo rotore al suo interno e un secondo rotore di coda necessario per fornire il controllo. Questo modello si rivela un'ottima soluzione nel caso si debba trasportare dei carichi particolarmente pesanti per un breve tempo di volo.

Aumentando il numero di eliche a tre e disponendole in modo da formare una Y, si ottiene la configurazione del tricopter. Questo è costituito da tre motori, tre controllori, quattro giroscopi e un servomotore. I motori sono posizionati alle estremità dei tre bracci e ognuno presenta un sensore di posizione. È in grado di rimanere stabile lungo un determinato percorso grazie ai sensori di posizione e non necessita di correzione manuale.

La configurazione più diffusa è quella a quattro eliche, che caratterizza la tipologia del quadrirotore (quadcopter). Le quattro eliche possono assumere la disposizione a + (QUAD I) oppure a X (QUAD X), come mostra Figura 2.1.

Esistono anche configurazioni a sei e otto rotori. Queste configurazioni sono sicuramente più stabili rispetto a quelle con un numero minore di eliche e consentono inoltre di controllare il velivolo con più precisione.

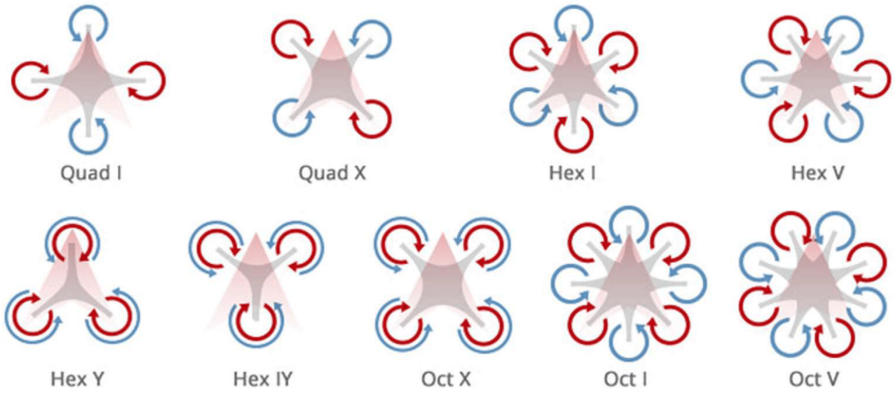


**Figura 2.1:** Classificazione multi-rotori in base al numero e alla configurazione dei rotori.

Nel seguito si prenderà come riferimento un quadrirotore con struttura a X dato che presenta un modello fisico-matematico semplificato e rende pertanto più semplice il progetto di schemi di controllo e l'identificazione del sistema. Inoltre, questa disposizione dei rotori minimizza la variazione di posizione del baricentro e risulta pertanto più semplice tenere traccia della traiettoria del drone nello spazio.

## 2.2 MODELLAZIONE MATEMATICA DI UN QUADRIROTORE

Un robot, nello specifico un quadrirotore, può essere considerato come un corpo rigido che trasla e ruota in uno spazio tridimensionale. Pertanto, è necessario costruire un modello matematico in grado di descrivere in modo compatto il suo spostamento.

Il modello fisico-matematico di un quadrirotore si può trovare con diversi formalismi, quali quello di Newton-Eulero e quello di Lagrange [3] [4] [5]. Nel seguito si descrive il primo.

Nello specifico, si tratta di un sistema a sei gradi di libertà - 6 Degree of Freedom (DoF) - quindi è descrivibile utilizzando dodici stati[1]: sei per la posizione, sei per l'assetto. Il sistema pertanto risulta sotto-attuato perché è caratterizzato da sei gradi di libertà, ma il suo movimento nello spazio può essere controllato soltanto da quattro attuatori (uno per ogni elica). In particolare, variando le velocità angolari delle eliche del velivolo è possibile effettuare tutti i movimenti.

1 Per ogni grado di libertà si considerano due stati: uno descrive la variabile considerata in un determinato istante di tempo t, l'altro ne descrive l'evoluzione nel tempo. Nel caso di un sistema tempo discreto è sufficiente calcolare la variabile nell'istante discreto successivo $t + T_C$, dove $T_C$ è il periodo di campionamento (Metodo Differenze Finite). Siccome $T_C$ è un tempo costante si può normalizzare il periodo di campionamento rendendolo unitario. Nel caso di un sistema tempo continuo si estende lo stesso ragionamento tramite l'operatore di derivata (Metodo Eulero). Se ad esempio la variabile considerata rappresenta una posizione, la sua evoluzione è una velocità.

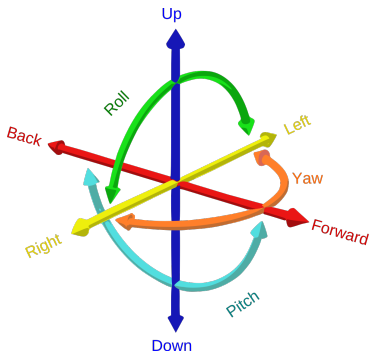


**Figura 2.2:** Sei gradi di libertà (6 DoF) di un corpo rigido in uno spazio tridimensionale.

### 2.2.1 *Sistemi di Riferimento*

Prima di poter descrivere le equazioni che governano un velivolo, come il quadrirotore, è necessario definire le coordinate di riferimento in cui si intende farlo. Sono necessari due sistemi di riferimento per descrivere la posizione e l'assetto del drone nello spazio: un sistema *mondo* (earth frame o E) inerziale, e un sistema *drone* (body frame o B) solidale al velivolo in movimento, le cui dinamiche possono essere espresse rispetto alla terna fissa. La terna fissa (inerziale) è una terna dove si può considerare valida la prima legge di Newton.

Il sistema *mondo* può essere definito secondo le coordinate Local Tangent Plane (LTP), relative al piano tangente ad ogni punto della superficie terrestre. Esistono due convenzioni: il sistema East North Up (ENU) è definito da una terna i cui versori sono orientati verso Est, Nord (piano tangente) e verso l'esterno della terra (Figura 2.3), il sistema North East Down (NED) è definito analogamente con Est e Nord scambiati e punta verso l'interno della terra.

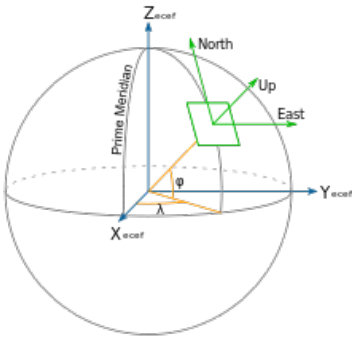


**Figura 2.3:** Sistema di riferimento ENU, tangente alla superficie terrestre.

Il sistema *drone* ha origine nel centro di massa del quadrirotore e in letteratura viene definito come Aircraft Body Center (ABC). Questa terna mobile varia a seconda della configurazione dei rotori del quadrirotore, come mostra Figura 2.4. Nel caso di

un quadrirotore con disposizione a + si considerano gli assi x e y coincidenti con i due assi principali del quadrirotore. Nel caso in oggetto (disposizione a X), invece, gli assi x e y corrispondono alle bisettrici dei due assi principali del quadrirotore.

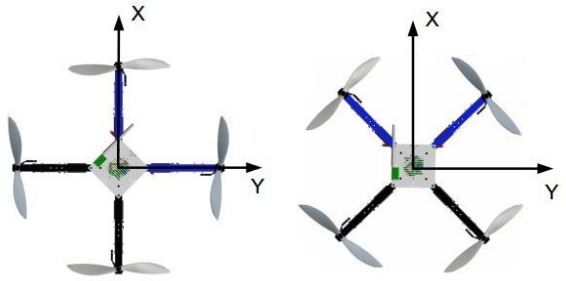


**Figura 2.4:** ABC di un QUAD I e di un QUAD X.

Figura 2.5 presenta i due sistemi di riferimento *mondo* e *drone*, secondo le ipotesi formulate.

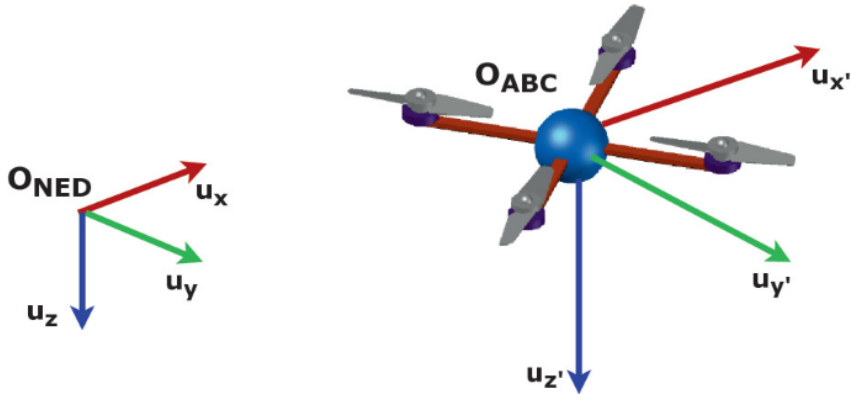


**Figura 2.5:** Sistemi di riferimento NED e ABC, utilizzati nella modellazione di un quadrirotore.

### 2.2.2 *Trasformazioni*

Dato che il drone trasla e ruota, è necessario stabilire per entrambi i moti un modo per poter passare dal sistema di riferimento *mondo* al sistema di riferimento *drone* e viceversa.

#### TRASLAZIONI

Per le traslazioni è sufficiente utilizzare la somma vettoriale nello spazio a tre dimensioni di riferimento. Infatti, dato un generico punto P rappresentato nel sistema di riferimento del velivolo $O_{ABC}$ dal vettore $v^{P}_{ABC}$, esiste un vettore di traslazione che

permette di rappresentare lo stesso punto nel sistema di riferimento a terra. Nello specifico:

$$v^p_{NED} = v^p_{ABC} + t_{NED}$$

dove $v^P_{NED}$ è il vettore che rappresenta il punto P nel sistema di riferimento a terra e $t_{NED}$ è il vettore di traslazione.

### ROTAZIONI

Le rotazioni possono essere descritte assumendo il corpo rigido fisso (in termini di posizione) nell'origine di un sistema di riferimento, con i tre versori del sistema che cambiano sotto l'effetto di rotazioni. Quindi, la rotazione è una relazione tra un sistema di riferimento fisso e lo stesso sistema (cioè con stessa origine) ruotato rispetto ai tre angoli di Eulero (cioè rispetto ai tre assi). La rotazione di un angolo intorno ad un asse può essere rappresentata da una matrice le cui colonne corrispondono ai versori del sistema ruotato. Considerando i tre possibili angoli di rotazione relativi ai tre assi del sistema di riferimento, si ottengono le seguenti tre matrici di rotazione ortonormali, cioè tali che $R^\top = R^{-1}$.

- Rotazione intorno all'asse x:

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

- Rotazione intorno all'asse y:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2}$$

- Rotazione introno all'asse z:

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Quindi, dato un generico punto P rappresentato nel sistema di riferimento traslato ma non ruotato $O_{xyz}$ dal vettore $v^P_{xyz}$, la trasformazione che consente di rappresentare lo stesso punto nel sistema di riferimento $O_{x'y'z'}$ ruotato intorno all'origine rispetto ad uno specifico asse (e.g. asse x) è la seguente:

$$v^P_{x'y'z'} = R_x(\bar{\phi})v^P_{xyz}$$

dove $v^P_{x'y'z'}$ è il vettore che rappresenta P nel sistema ruotato intorno all'asse x di un angolo $\bar{\phi}$, come indicato nella matrice di rotazione.

A questo punto, si definisce la Direct Cosine Matrix (DCM) come il prodotto delle tre matrici di rotazione per passare da un sistema di riferimento all'altro. Per semplificare la notazione, le funzioni trigonometriche $\sin(\cdot)$ e $\cos(\cdot)$ sono talvolta indicate come $s(\cdot)$ e $c(\cdot)$, rispettivamente.

Per passare dal sistema *mondo* al sistema *drone* si considera la seguente matrice di trasformazione:

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

$$= \begin{bmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & s(\phi)s(\theta)c(\psi) + c(\phi)c(\psi) & s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{bmatrix} \tag{4}$$

Per passare dal sistema *drone* al sistema *mondo*, invece, si considera la seguente matrice di trasformazione:

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

$$= \begin{bmatrix} c(\theta)c(\psi) & -c(\phi)s(\psi) + s(\phi)s(\theta)c(\psi) & s(\phi)s(\psi) + c(\phi)s(\theta)c(\psi) \\ s(\psi)c(\theta) & c(\theta)c(\psi) + s(\phi)s(\theta)s(\psi) & s(\theta)c(\phi)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{5}$$

### 2.2.3 *Modello Non Lineare*

I sei stati di posizione descrivono la posizione del drone e la sua evoluzione nel tempo, cioè come il velivolo trasla rispetto al sistema a terra. I sei stati di assetto descrivono l'assetto del drone e la sua evoluzione nel tempo, cioè come il velivolo ruota introno all'origine del sistema di riferimento traslato. A tal proposito, in aeronautica si utilizzano gli angoli di Eulero: rollio (roll o $\phi$), beccheggio (pitch o $\theta$) e imbardata (yaw o $\psi$); che indicano rispettivamente una rotazione intorno agli assi Y, X e Z, come mostra Figura 2.6.

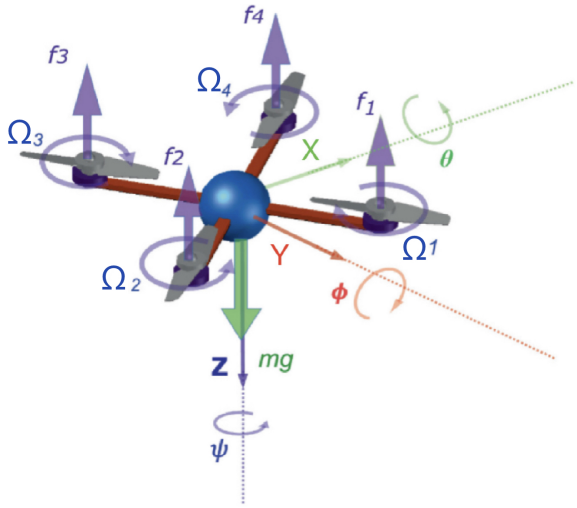


**Figura 2.6:** Modello fisico del quadrirotore, dove $\Omega_i$ e $f_i$ ($i = 1, 2, 3, 4$) sono rispettivamente le velocità angolari dei rotori e le relative forze di spinta generate.

Si definiscono quattro vettori tridimensionali per rappresentare i dodici stati del modello. I primi due vettori sono definiti rispetto al sistema di riferimento *mondo* (E) e rappresentano rispettivamente il vettore di posizione che identifica la posizione del drone nello spazio (6) e il vettore di assetto che descrive (secondo gli angoli di Eulero) come il drone è ruotato intorno agli assi coordinati (7).

$$\Gamma_E = \begin{bmatrix} x & y & z \end{bmatrix}^\top \tag{6}$$

$$\Theta_E = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top \tag{7}$$

I restanti vettori sono definiti rispetto al sistema di riferimento *drone* (B) e rappresentano rispettivamente la velocità lineare (8) e la velocità angolare (9) del quadrirotore.

$$V_B = \begin{bmatrix} u & v & w \end{bmatrix}^\top \tag{8}$$

$$\omega_B = \begin{bmatrix} p & q & r \end{bmatrix}^\top \tag{9}$$

Il moto di traslazione del velivolo è definito dall'equazione 10 seguente, che mette in relazione $\Gamma_E$ e $V_B$ tramite la matrice $R_{BE}$ (DCM definita dalla 5).

$$\dot{\Gamma_E} = R_{BE} V_B \tag{10}$$

Il moto di rotazione, invece, mette in relazione $\Theta_E$ e $\omega_B$. In questo caso entra in gioco una nuova matrice $J$ che consente di trasformare le variazioni degli angoli di Eulero intorno ai rispettivi assi $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ nel vettore di velocità angolare del quadrirotore $\omega_B$, come mostra l'equazione 11.

$$J\dot{\Theta_E} = \omega_B \tag{11}$$

Considerando la matrice inversa (13) si ottiene l'equazione del moto di rotazione cercata (12).

$$\dot{\Theta_E} = J^{-1} \omega_B \tag{12}$$

$$J^{-1} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\sin(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{13}$$

Prima di poter formulare il modello matematico del quadrirotore è necessario fare un'ipotesi semplificativa. Ogni sistema rotante, infatti, risente del cosiddetto effetto giroscopico, cioè un fenomeno che tende a mantenere il sistema sul suo asse di rotazione. Risulta quindi più difficile cambiare l'assetto, ma siccome l'effetto è tanto più forte quanto più la velocità di rotazione è grande, e in questo caso non ci sono grandi

velocità, l'effetto è trascurabile. Si può dimostrare che trascurandolo, il modello può essere quindi scritto con il seguente sistema di equazioni nel sistema di riferimento *drone*:

$$\begin{cases} \dot{u} = (vr - wq) + g\sin((\theta) \\ \dot{v} = (wp - ur) - g\cos(\theta)\sin(\phi) \\ \dot{w} = (uq - vp) - g\cos(\theta)\sin(\phi) + \frac{U_1}{m} \\ \dot{p} = \frac{I_{YY} - I_{ZZ}}{I_{XX}} qr + \frac{U_2}{I_{XX}} \\ \dot{q} = \frac{I_{ZZ} - I_{XX}}{I_{YY}} pr + \frac{U_3}{I_{YY}} \\ \dot{r} = \frac{I_{XX} - I_{YY}}{I_{ZZ}} pq + \frac{U_4}{I_{ZZ}} \end{cases} \tag{14}$$

dove i termini $U_i$ $(i = 1, 2, 3, 4)$ sono i termini di controllo del quadrirotore e sono infatti definiti in funzione delle velocità angolari dei rotori $\Omega_i$ $(i = 1, 2, 3, 4)$, come mostra la 15.

$$\begin{cases} U_1 = b(\Omega_1{}^2 + \Omega_2{}^2 + \Omega_3{}^2 + \Omega_4{}^2) \\ U_2 = lb(-\Omega_2{}^2 + \Omega_4{}^2) \\ U_3 = lb(-\Omega_1{}^2 + \Omega_3{}^2) \\ U_4 = d(-\Omega_1{}^2 + \Omega_2{}^2 - \Omega_3{}^2 + \Omega_4{}^2) \end{cases} \tag{15}$$

Inoltre, nel modello (14) sono presenti i termini $I_{XX}$, $I_{YY}$, $I_{ZZ}$, cioè gli elementi diagonali della matrice di inerzia I, indicata nella 16. Quest'ultima risulta diagonale perché per ipotesi l'origine del sistema di riferimento del drone è nel suo centro di massa e gli assi corrispondono agli assi principali di inerzia.

$$I = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \tag{16}$$

Infine, vale la pena spiegare il significato dei coefficienti b e d, che legano le velocità angolari dei rotori ai termini di controllo.

Il termine b è il cosiddetto coefficiente di thrust. È un coefficiente che lega la spinta alla velocità angolare secondo la seguente equazione:

$$b = \frac{T}{\Omega^2} \tag{17}$$

dove $\Omega$ è la velocità di un generico rotore e T è la spinta che genera verso l'alto. Si misura in $[Ns^2]$ ed è considerato costante e uguale per tutti i rotori. La spinta verso l'altro è pertanto descritta dal termine di controllo $U_1$ nella 15.

Il termine d è il cosiddetto coefficiente di drag. Quando le pale di un rotore girano in un certo verso, nasce una forza dovuta alla resistenza del vento nel senso opposto che genera un momento. Nella dinamica del quadrirotore, il coefficiente d incide sull'angolo di imbardata, legato alla variazione dei momenti. Anche questo coefficiente è considerato costante e uguale per tutti i rotori, si misura in $[Nms^2]$ ed è definito come segue:

$$d = \frac{Q}{\Omega^2} \tag{18}$$

dove Q è il momento della forza di trascinamento. Nella 15 il termine $U_4$ tiene conto di questo contributo.

Poi, i termini $U_2$ e $U_3$ sono legati rispettivamente agli angoli di roll e pitch. Nello specifico, $U_2$ è legato alla differenza delle spinte date dai rotori di indice due e quattro, cioè (dalla 17) ai prodotti $b{\Omega_2}^2$ e $b{\Omega_4}^2$, poi moltiplicata per il braccio l. Analogo è il discorso per $U_3$, ma chiaramente interessa i rotori di indice uno e tre.

Si può anche scrivere il modello dinamico riferito al sistema *mondo* tramite il sistema di equazioni definito nella 19.

$$\begin{cases} \ddot{X} = (\sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi))\frac{U_1}{m} \\ \ddot{Y} = (-\cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi))\frac{U_1}{m} \\ \ddot{Z} = -g + (\cos(\theta)\cos(\phi))\frac{U_1}{m} \\ \dot{\phi} = p + \sin(\phi)\tan(\theta)q + \cos(\phi)\tan(\theta)r \\ \dot{\theta} = \cos(\phi)q + \cos(\theta)\sin(\phi)r \\ \dot{\psi} = -\sin(\phi)q + \cos(\theta)\cos(\phi)r \end{cases} \tag{19}$$

Si nota facilmente che il modello dinamico ottenuto (definito dalla 14 nel sistema solidale al drone e dalla 19 nel sistema solidale alla terra) è di tipo non lineare.

## 2.3 ANGOLI DI EULERO E QUATERNIONI

Come spiegato nel paragrafo precedente, ogni rotazione di un corpo rigido può essere descritta mediante i tre angli di Eulero, che specificano una sequenza di tre rotazioni successive intorno agli assi coordinati. Da un lato questa rappresentazione risulta facile da comprendere e, dal punto di vista dell'implementazione, necessita semplicemente di tre numeri reali floating point. Dall'altro soffre il problema del gimbal lock (o blocco cardanico).

### 2.3.1 *Gimbal Lock*

Si tratta di un fenomeno problematico dei giroscopi causato dall'allineamento di due assi rotanti verso la stessa direzione. Il blocco causa la perdita del grado di libertà corrispondente all'asse bloccato.

Nello specifico, il problema si verifica quando il valore di alcuni angoli raggiunge valori critici. Infatti, se ad esempio il valore dell'angolo di pitch ($\theta$) dovesse raggiungere i 90 gradi, il sensore a bordo del velivolo non sarebbe in grado di distinguere tra roll ($\phi$) e yaw ($\psi$). Osservando la Figura 2.7 si nota come l'orientazione relativa al caso B possa essere frutto di due rotazioni differenti: un beccheggio seguito da un rollio oppure un'imbardata seguita da un beccheggio.

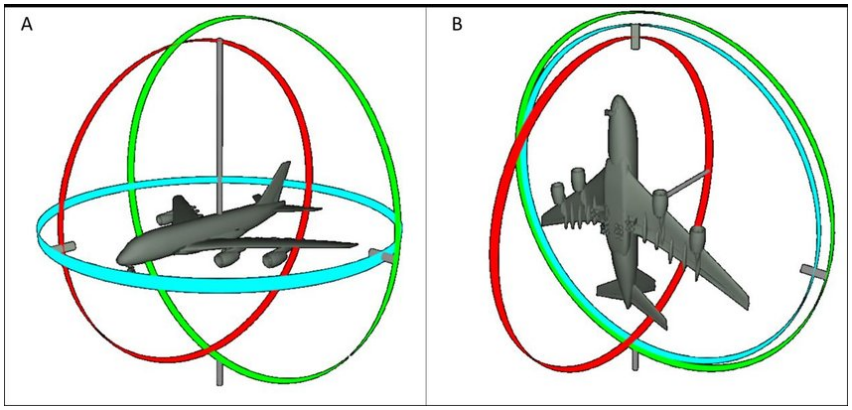


**Figura 2.7:** Problema del blocco cardanico. Caso A: non si verifica il blocco cardanico. Caso B: rollio (roll) e imbardata (yaw) sono bloccati e risultano indistinguibili, quindi si perde un grado di libertà (blocco cardanico).

Questa problematica si riscontra facilmente anche nel modello matematico. Infatti, osservando la matrice $T$ che descrive il moto rotazionale del quadrirotore, si nota come per $\theta = \pm 90\,\text{deg}$ questa risulta singolare (determinante nullo).

Esistono due metodi differenti per risolvere questo problema. Il primo prevede semplicemente di utilizzare gli angoli di Eulero ad una condizione: gli angoli misurati dai sensori devono mantenersi piccoli. Il secondo, molto utilizzato, consiste nell'utilizzo dei quaternioni.

### 2.3.2 *Quaternioni*

Introdotto da William Rowan Hamilton nel 1843 come estensione dei numeri complessi, un quaternione è un elemento scrivibile come:

$$a + bi + cj + dk$$

dove $a, b, c, d$ sono numeri reali e $i, j, k$ si comportano in modo simile all'unità immaginaria dei numeri complessi. Si può dire che la prima coordinata $a$ costituisce la parte scalare o reale del quaternione, mentre le rimanenti tre coordinate $b, c, d$ ne costituiscono la parte vettoriale o immaginaria.

Pertanto, un quaternione è identificato da quattro numeri reali e può essere scritto come un vettore di $\mathbb{R}^4$ come segue.

$$q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\top$$

con $q_0, q_1, q_2, q_3 \in \mathbb{R}$.

Un quaternione unitario è un quaternione di norma unitaria.

$$\|q\| = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$$

La totalità dei quaternioni unitari forma una ipersfera quadrimensionale in $\mathbb{R}^4$.

$$S^3 = \{(q_0, q_1, q_2, q_3) \in \mathbb{R}^4 | q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1\}$$

Questi particolari quaternioni possono essere utilizzati per descrivere l'assetto del quadrirotore nel sistema di riferimento ABC rispetto al sistema di riferimento inerziale fisso [6].

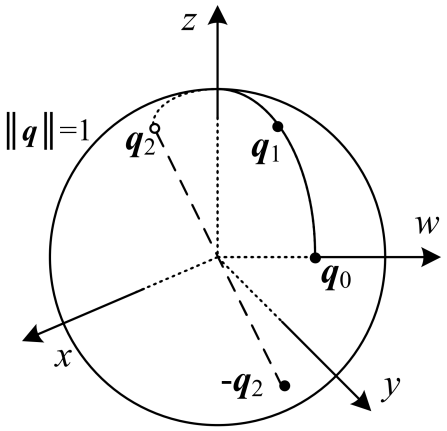


**Figura 2.8:** Rappresentazione grafica di una rotazione tramite un quaternione unitario.

Somma e prodotto di due quaternioni sono definiti tenendo conto delle relazioni fondamentali[2]:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{20}$$

2 William Rowan Hamilton scrisse per la prima volta questa formula il 16 Ottobre 1843 camminando lungo il Broom Bridge, che attraversa il Royal Canal, a Cabra, Dublino, Irlanda. Tale evento è ad oggi commemorato da una targa collocata sul ponte che recita: "Mentre qui passeggiava, il 16 ottobre 1843 Sir William Rowan Hamilton, in un lampo d'ispirazione scoprì la formula fondamentale per la moltiplicazione dei quaternioni (20) e la incise su una pietra di questo ponte."

Le relazioni che legano gli angoli di Eulero $(\phi, \theta, \psi)$ al quaternione unitario $(q)$ corrispondente sono mostrate nelle equazioni 21 e 22.

$$\begin{cases} q_0 = c(\frac{\phi}{2})c(\frac{\theta}{2})c(\frac{\psi}{2}) + s(\frac{\phi}{2})s(\frac{\theta}{2})s(\frac{\psi}{2}) \\ q_1 = s(\frac{\phi}{2})c(\frac{\theta}{2})c(\frac{\psi}{2}) - c(\frac{\phi}{2})s(\frac{\theta}{2})s(\frac{\psi}{2}) \\ q_2 = c(\frac{\phi}{2})s(\frac{\theta}{2})c(\frac{\psi}{2}) + s(\frac{\phi}{2})c(\frac{\theta}{2})s(\frac{\psi}{2}) \\ q_3 = c(\frac{\phi}{2})c(\frac{\theta}{2})s(\frac{\psi}{2}) - s(\frac{\phi}{2})s(\frac{\theta}{2})c(\frac{\psi}{2}) \end{cases} \tag{21}$$

$$\begin{cases} \phi = \tan^{-1}\left[\frac{2(q_2q_3+q_0q_1)}{q_0^2-q_1^2-q_2^2+q_3^2}\right] \\ \theta = \sin^{-1}\left[-2(q_1q_3 + q_0q_2)\right] \\ \psi = \tan^{-1}\left[\frac{2(q_1q_2+q_0q_3)}{q_0^2+q_1^2-q_2^2-q_3^2}\right] \end{cases} \tag{22}$$

Come detto, questa soluzione è ampiamente utilizzata in ambito informatico e robotico. Infatti, molti linguaggi di programmazione e molte librerie forniscono direttamente operazioni di conversione tra angoli di Eulero e quaternioni unitari, come si vedrà nel seguito.

## 2.4 CONTROLLO DI UN QUADRIROTORE

Un quadrirotore a X presenta quattro rotori posizionati come in Figura 2.9. Essendo i motori e le eliche sullo stesso piano, è necessario equilibrare il moto del drone impiegando due motori che ruotano in senso orario e due che ruotano in senso antiorario. Nello specifico, i rotori M1 e M4 ruotano in senso orario (clockwise o CW), mentre i rotori M2 e M3 ruotano in senso antiorario (counterclockwise o CCW). Questa contrapposizione permette di annullare l'energia rotatoria dei motori e quindi eventuali coppie di reazione.

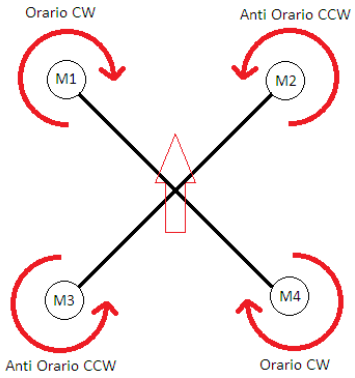


**Figura 2.9:** Disposizione e funzionamento dei rotori di un quadrirotore con eliche disposte a X.

### 2.4.1 *Manovre di Spinta, Rollio, Beccheggio e Imbardata*

Nel seguito si descrive il funzionamento delle quattro principali manovre di controllo del quadrirotore. Tre di queste manovre sono dovute al cambiamento dell'assetto del drone nel tempo, cioè alle variazioni degli angoli di Eulero (rollio, beccheggio e imbardata). È possibile alterare questi angoli modificando le velocità dei quattro rotori in modo specifico. La quarta e ultima manovra di controllo, invece, riguarda la spinta del quadrirotore verso l'alto, mantenendone fisso l'assetto.

#### MANOVRA DI SPINTA

Quando il vettore di spinta (thrust) è allineato alla verticale (direzione ortogonale al piano individuato dal quadrirotore) e ha modulo maggiore della forza peso del drone ($F_p = mg$ dove $m$ è la massa del drone e $g$ è l'accelerazione gravitazionale), si verifica una traslazione verticale lungo l'asse Z. Come specifica la relazione 17, la spinta generata è legata al quadrato delle velocità angolari dei rotori. In condizioni di equilibrio statico (hovering) ciascun rotore genera una spinta pari ad un quarto della forza peso del drone. Per aumentare il modulo della spinta verso l'alto (asse Z positivo) si aumentano le velocità di tutti i rotori di una stessa quantità, in modo da non creare differenze che possano modificare l'assetto del drone. In Figura 2.10 è mostrato il concetto.

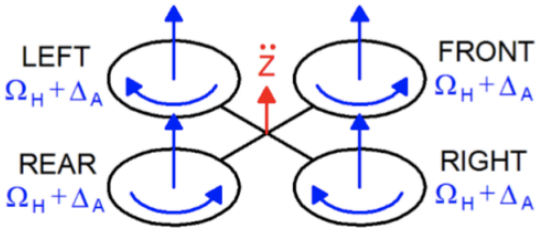


**Figura 2.10:** Manovra di spinta: la velocità di tutti i rotori aumenta di una quantità costante.

### MANOVRA DI ROLLIO

Il rollio contribuisce allo spostamento del drone lungo l'asse Y (destra e sinistra). Si ottiene lasciando invariate le velocità dei motori anteriore (FRONT) e posteriore (REAR), e cambiando di una certa quantità le velocità dei motori di destra (RIGHT) e sinistra (LEFT); come mostrato in Figura 2.11.

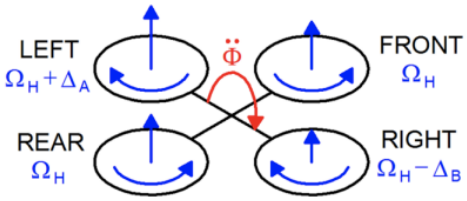


**Figura 2.11:** Manovra di rollio: cambia la velocità dei rotori di destra (RIGHT) e sinistra (LEFT).

### MANOVRA DI BECCHEGGIO

Il beccheggio contribuisce allo spostamento del drone lungo l'asse X (avanti e indietro). Si ottiene, analogamente al rollio, lasciando invariate le velocità dei motori di destra (RIGHT) e sinistra (LEFT), e cambiando di una certa quantità le velocità dei motori anteriore (FRONT) e posteriore (REAR); come mostrato in Figura 2.12.

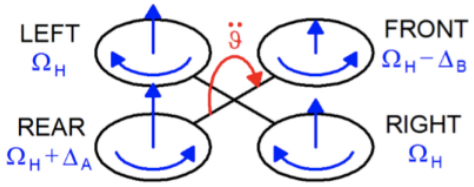


**Figura 2.12:** Manovra di beccheggio: cambia la velocità dei rotori anteriore (FRONT) e posteriore (REAR).

#### MANOVRA DI IMBARDATA

L'imbardata è il movimento più complesso. L'idea fondamentale è cambiare il momento rispetto all'asse z, immaginato uscente dal piano dei rotori, e mantenere costante la spinta verso l'alto. Per ottenere questo, si aumenta la velocità dei rotori di due motori opposti, e si diminuisce la velocità dei due restanti rotori. In questo modo si avranno, per esempio, due rotori che girano velocemente in senso orario, e due che girano più lentamente in senso antiorario. Nel complesso ci sarà dunque un momento che fa girare il drone intorno all'asse z. Se le velocità sono opportunamente scelte, non ci sarà un cambiamento della spinta verso l'alto. La Figura 2.13 riassume quanto detto.

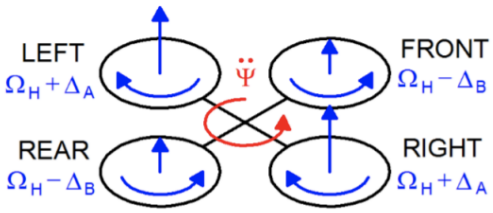


**Figura 2.13:** Manovra di imbardata.

### 2.4.2 *Controllo Stabilizzato e Acrobatico*

Esistono due modalità di volo e quindi di controllo di un quadrirotore. La modalità stabilizzata, anche nota come self-level, e la modalità acrobatica, anche nota come rates e utilizzata per droni da gara.

Nel primo caso si controlla il drone agendo sugli angoli di Eulero, come esposto nel paragrafo precedente. Nel secondo caso, invece, il segnale di controllo inviato al drone non contiene gli angoli di Eulero, ma le velocità angolari. Questo significa che, controllando il drone ad esempio con un radiocomando e lasciando andare le levette di roll e pitch verso il centro (cioè portandole a zero), nel primo caso il velivolo ritorna in posizione neutrale (piana ed allineata con l'orizzonte), mentre nel secondo caso si mantiene l'angolo di rotazione sugli assi di roll e pitch impostati.

# 3

# ROS

Robot Operating System (ROS) [7] nasce nel 2007 sotto il nome di Switchyard presso lo Stanford Artificial Intelligence Laboratory (SAIL), centro di eccellenza per la ricerca sull'intelligenza artificiale della Stanford University, in California. Nello specifico, viene sviluppato inizialmente da due dottorandi dell'università a capo del programma Personal Robotics (PR) per fornire supporto al progetto Stanford Artificial Robot (STAIR). Nel 2008 lo sviluppo del progetto progredisce presso il Willow Garage (diventando ROS), ente di ricerca e incubatore nel campo della robotica, che versa il proprio contributo approfondendo i principi con cui il progetto nasce e realizzando le prime versioni testate.

Brevemente, si tratta di un middleware open source basato su sistemi operativi Linux (e.g. Ubuntu) che consente comunicazione tra processi attraverso un meccanismo basato sul passaggio di messaggi. Praticamente, comprende un insieme di librerie, strumenti e convenzioni che semplificano lo sviluppo di software impiegato nei robot.

ROS non è stato inizialmente progettato e rilasciato per un uso di tipo industriale, ma con l'obiettivo di velocizzare la ricerca sulla robotica. Per questo motivo, alcuni aspetti come la sicurezza oppure il supporto per programmi real-time non sono stati considerati essenziali. L'espansione di ROS in ambito industriale ha dato vita alla creazione di ROS 2 [8]: una riprogettazione completa del framework che mantiene le funzioni principali e risolve le carenze riscontrate nella prima versione.

## 3.1 FILESYSTEM LEVEL

Il *Filesystem Level* è l'organizzazione del framework all'interno della macchina. Comprende tutte le risorse utilizzate: *packages*, *metapackages*, *package manifests*, *repositories*, *message types*, *service types*.

### 3.1.1 *Packages*

I *packages* sono l'unità di organizzazione del software a questo livello in ROS. Contengono file di configurazione e file eseguibili (*nodes*, trattati nel seguito) e contribuiscono, insieme alle *repositories*, alla modularità del framework. Infatti, si cerca di creare e mantenere *packages* di dimensioni contenute e semplici da utilizzare.

## 3.2 COMPUTATION GRAPH LEVEL

Il *Computation Graph Level* è dove viene processata ed elaborata l'informazione. Si tratta di una rete peer-to-peer costituita da tutti i processi attivi in ROS che stanno processando ed elaborando dati insieme.

### 3.2.1 *Nodes*

Un *node* è un processo che compie una qualsiasi attività computazionale all'interno del sistema ROS.

Essendo ROS progettato per essere modulare, tipicamente un sistema basato su di esso comprende numerosi nodi e in tal contesto sono interpretabili come moduli software, ognuno dei quali è incaricato di gestire un aspetto del comportamento del robot (e.g. parte decisionale, movimento, azionamento motori, ecc.).

Un sistema il cui carico computazionale venga ripartito tra i vari nodi di cui è costituito ha innanzitutto il vantaggio di una maggiore tolleranza agli errori, potendo gestire indipendentemente il malfunzionamento del singolo nodo. La complessità del codice è ridotta se confrontata coi sistemi monolitici e i dettagli implementativi sono nascosti in quanto i singoli nodi offrono un'interfaccia composta da una Application Programming Interface (API) minimale.

### 3.2.2 *Master*

In un sistema basato su ROS, il *master* è un server centralizzato che offre ai nodi un servizio di registrazione e di naming, similmente all'informazione data da un server Domain Name System (DNS). Permette infatti al singolo nodo di contattarne un secondo attraverso una metodologia peer-to-peer. Tiene inoltre traccia per ogni singolo *topic* (trattati nel seguito) dei relativi nodi publisher e subscriber. Il *master* offre anche il parameter server, cioè un server che offre un servizio di memorizzazione e

consultazione di parametri a tempo di esecuzione (runtime) ai nodi che ne richiedono i servizi attraverso una API di rete.

### 3.2.3 *Topics*

I *topic* costituiscono il mezzo di comunicazione asincrono, unidirezionale, per lo scambio di messaggi tra nodi, secondo una semantica di tipo publish/subscribe (pubblica/sottoscrivi). Ci possono essere più nodi publisher e subscriber concorrenti per un singolo *topic* e un singolo nodo può pubblicare e/o sottoscriversi a più *topics*. In generale, i nodi publisher e subscriber non sono consapevoli dell'esistenza degli altri, disaccoppiando così la produzione dell'informazione dal suo consumo.

Ogni *topic* è fortemente tipizzato dal tipo di messaggio che viene pubblicato e i nodi possono ricevere esclusivamente messaggi il cui tipo sia compatibile. Ciò determina che all'interno del *topic* sia possibile scrivere o leggere un solo tipo di messaggio.

Il protocollo di trasporto utilizzato per lo scambio dei messaggi viene definito in fase di esecuzione e può essere di due tipi: TCPROS o UDPROS.

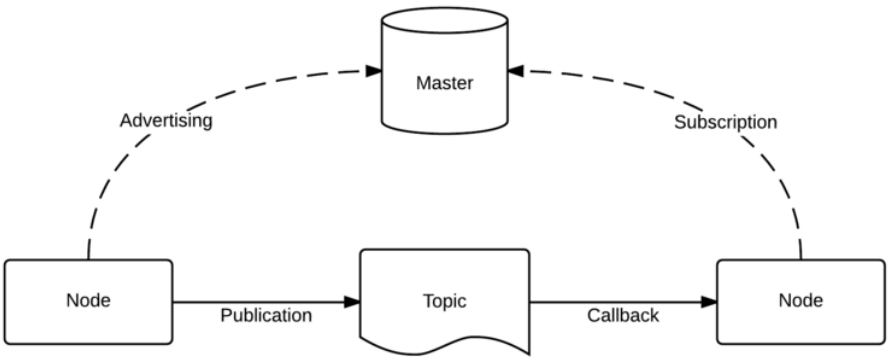


**Figura 3.1:** Funzionamento di *master*, *node* e *topic* in ROS.

### 3.2.4 *Services*

I *services* rappresentano il mezzo di comunicazione sincrono secondo una semantica di tipo request/reply (richiesta/risposta), implementando in ROS una funzionalità di tipo Remote Procedure Call (RPC). Pertanto, si definiscono nella rete nodi provider, che forniscono servizi, e nodi client, che usufruiscono di tali servizi.

Esistono funzioni per verificare la presenza di un service provider nella rete. Nel dettaglio, il nodo client invia dei dati che prendono il nome di request a un nodo server, aspettando la risposta di quest'ultimo. Il nodo server, una volta ricevuta, processa la request del client e gli riscontra dei dati in risposta (response).

### 3.2.5 *tf*

Quando si eseguono compiti con un robot, è fondamentale che il robot sia consapevole di dove si trova e di dove si trova il resto del mondo rispetto a se stesso. La libreria *tf* [9] è stata progettata per fornire un modo standard per tenere traccia dei riferimenti di coordinate e trasformare i dati all'interno di un intero sistema in modo tale che gli utenti dei singoli componenti possano essere sicuri che i dati si trovino nel frame di coordinate desiderato, senza richiedere la conoscenza di tutti i riferimenti di coordinate nel sistema. Man mano che i sistemi robotici diventano sempre più complicati, essere in grado di concentrarsi con precisione sul task frame, cioè un frame di coordinate che può essere collegato a diversi oggetti che devono essere manipolati, e sui frame di coordinate rilevanti diventa fondamentale. La maggior parte dei sistemi robotici fonde i dati di molti sensori diversi con frame di coordinate differenti.

La libreria *tf* è stata sviluppata come pacchetto ROS per fornire questa capacità ed è composta da due moduli standard: un broadcaster (emittente) e un listener (ascoltatore). La prima parte si occupa della diffusione delle informazioni di trasformazione all'intero sistema. La seconda parte riceve le informazioni di trasformazione e le archivia per un uso successivo. Quindi è in grado di rispondere a domande sulla trasformazione risultante tra diversi frame di coordinate.

Le trasformazioni e i frame di coordinate possono essere espressi come un grafo con le trasformazioni come archi e i frame di coordinate come vertici. In questa rappresentazione la trasformata netta è semplicemente il prodotto degli archi che collegano due nodi qualsiasi. Il grafo può esistere con uno o più sottografi disconnessi e la trasformazione può essere calcolata tra nodi all'interno dei sottografi (intesi come componente connessa), ma non tra sottografi disconnessi. Tuttavia, in un grafo arbitrario, due nodi possono avere più cammini tra di loro, risultando in due o più potenziali trasformazioni nette che rendono ambiguo il risultato della query. Per evitare questo il grafo deve essere aciclico, in modo che ogni sottografo connesso è un albero. Limitare il grafo a un albero consente una rapida ricerca della connettività. Questo diventa importante con l'aumentare della complessità del grafo. Una struttura ad albero ha anche il vantaggio di consentire modifiche dinamiche alla struttura senza

utilizzare informazioni aggiuntive rispetto agli archi del grafo diretto.

Il modulo broadcaster è stato progettato in modo molto semplice. Trasmette messaggi ogni volta che si verifica un aggiornamento su una trasformazione specifica.

Il modulo listener raccoglie i valori in un elenco ordinato e, quando interrogato, può interpolare tra i due valori più vicini. La frequenza di trasmissione delle trasformate da parte dell'emittente deve essere selezionata sufficientemente alta da consentire la Spherical Linear Interpolation (SLERP) [10].

Per calcolare le trasformazioni tra due nodi qualsiasi, viene calcolato il cosiddetto spanning set. Per calcolare lo spanning set tra un frame di origine e quello di destinazione, il modulo listener risale l'albero (attraverso la relazione di genitore) finché non viene trovato un nodo genitore comune che forma uno spanning set. Se non viene trovato alcun genitore comune, la ricerca fallisce e restituirà un errore. Se la ricerca ha esito positivo, il listener calcolerà la trasformazione netta degli archi dal frame di origine al frame di destinazione lungo il cammino percorso.

### 3.2.6 *RViz*

ROS Visualization (RViz) è uno strumento di visualizzazione 3D di ROS. Consente di visualizzare il robot, il suo orientamento, i sistemi di riferimento, le matrici di covarianza e molto altro. Nell'ambito della tesi è risultato molto utile nelle simulazioni di volo per visualizzare in tempo reale la posizione del drone rispetto al sistema di riferimento a terra.

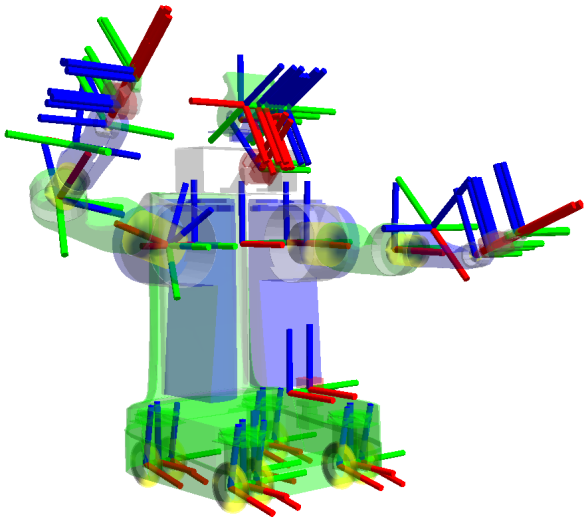

**Figura 3.2:** Task frames nel robot PR2 di Willow Garage, visualizzate in RViz. I cilindri RGB rappresentano gli assi X, Y e Z dei riquadri di coordinate.

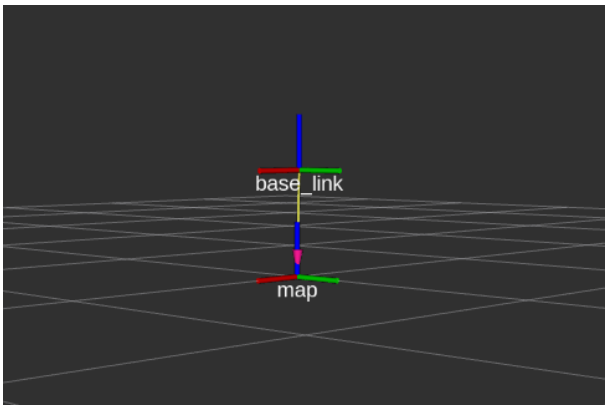


**Figura 3.3:** Task frames del quadrirotore simulato, visualizzate in RViz. Il riferimento del sistema a terra è indicato con *map*, quello solidale al quadrirotore con *base_link*.

## 3.3 ARCHITETTURA DI SIMULAZIONE

In questa sezione si entra nel dettaglio dell'architettura di simulazione utilizzata in ROS, partendo dalla scelta del sistema operativo. È stato utilizzato il sistema operativo Linux Ubuntu versione 20.04 (Focal Fossa) in quanto compatibile e consigliato per la versione ROS adottata (Noetic Ninjemys, rilasciata a Maggio 2020).

### 3.3.1 *Gazebo*

Gazebo è un simulatore 3D. Consente di creare uno scenario tridimensionale sul computer con robot, ostacoli e altri oggetti utili alla simulazione del robot in questione.

In questo lavoro di tesi, si è fatto uso della repository *iq_sim* di Intelligent Quads, disponibile su GitHub [11]. Questa repository contiene mondi gazebo per vari scenari e configurazioni di droni. Inoltre, è specificamente progettata per funzionare con il sistema di controllo ArduPilot e utilizza il plug-in gazebo ArduPilot per consentire al software di controllo ArduPilot di interfacciarsi e controllare il drone in gazebo.

Dopo aver clonato la repository sul computer utilizzato, per lanciare gazebo con il mondo desiderato basta eseguire da terminale il seguente comando.

```
roslaunch iq_sim runway.launch
```

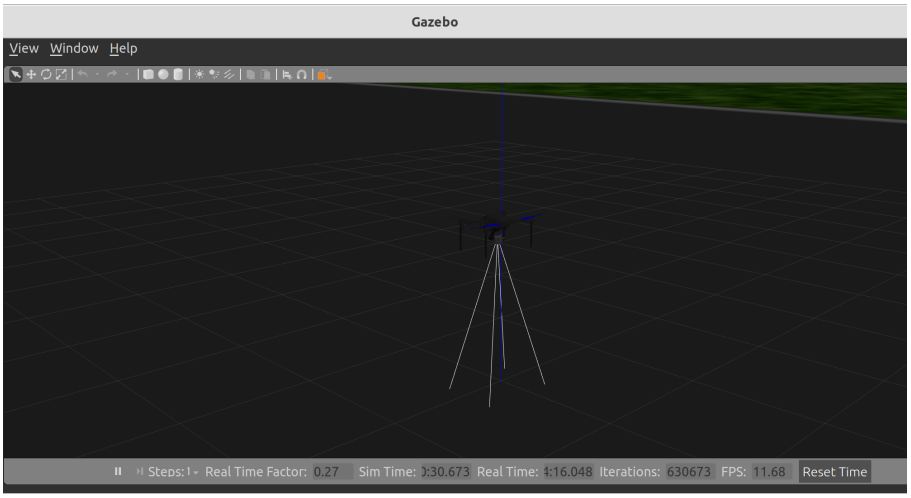


**Figura 3.4:** Quadrirotore nel mondo simulato *runway.world* (pista di decollo) di Intelligent Quads, in Gazebo.

### 3.3.2 *ArduCopter*

Gli aeromobili a pilotaggio remoto (UAV) sono sistemi che richiedono una rapida risposta al variare dei dati forniti dai sensori a bordo, per cui risulta fondamentale per il controllo del volo un software di autopilotaggio. Esistono diverse soluzioni, ma per questa tesi è stato considerato il software ArduCopter, cioè la versione per UAV del più generale software ArduPilot.

Software In The Loop (SITL) [12] è un simulatore che consente di eseguire ArduCopter sul computer, senza l'utilizzo di alcun hardware. Per lanciare ArduCopter si esegue il seguente comando da terminale, assicurandosi di essere nella directory corretta di ArduPilot. Le Figure 3.5 e 3.6 mostrano le interfacce grafiche contenenti tutte le informazioni sullo stato del velivolo e altri parametri.

```
./sim_vehicle.py -v ArduCopter -f gazebo-iris --console --mavproxy-args="--streamrate
  =100"
```

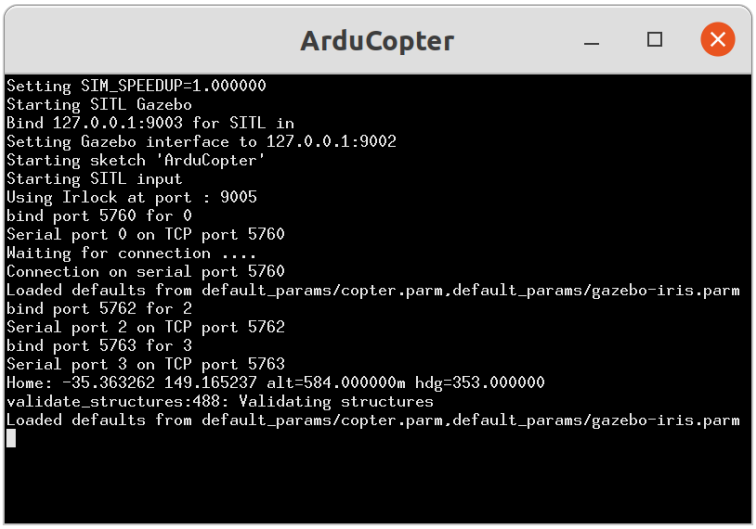


**Figura 3.5:** Interfaccia grafica ArduCopter.

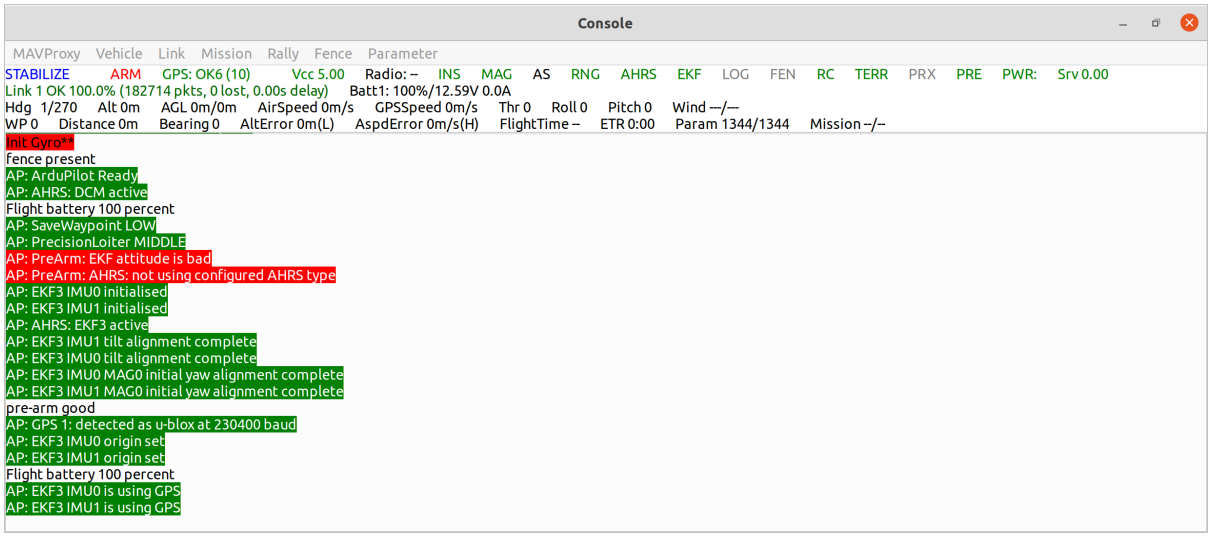


**Figura 3.6:** Interfaccia grafica console ArduCopter.

### 3.3.3 *MAVLink e MAVROS*

MAVLink (Micro Air Vehicle Link) è un protocollo di comunicazione per UAV di piccole dimensioni. I pacchetti utilizzati nella comunicazione MAVLink sono strutture in linguaggio C, trasmesse molto efficientemente attraverso dei canali di comunicazione seriale. I punti a suo favore sono sicuramente la velocità di comunicazione, la facilità con cui si possono realizzare nuovi messaggi e il fatto di essere open source. Per quanto riguarda la possibilità di creare nuovi messaggi, MAVLINK permette la definizione mediante file XML (chiamati dialetti), i quali vengono convertiti in codice sorgente in diversi linguaggi in base alle esigenze.

MAVROS [13] è l'estensione di MAVLink in ROS. Tra le tante caratteristiche di MAVROS vale la pena menzionare il fatto che è in grado di convertire i riferimenti di tipo NED in riferimenti di tipo ENU e viceversa, in modo da essere conforme allo standard adottato in ROS.

Per completare la configurazione del sistema di simulazione si lancia il file *apm.launch* con il seguente comando da terminale.

```
roslaunch mavros apm.launch
```

Questo si occupa di stabilire una connessione tra l'istanza ROS in questione e il simulatore configurato precedentemente (Gazebo e ArduCopter). Una delle operazioni più rilevanti, svolta da questo eseguibile, è quella di inviare dati relativi ai *tf* (task frames) coinvolti. In particolare, invia con una certa frequenza di trasmissione dati

sui *tf* in modo da mantenerli aggiornati in tempo reale. I *tf* possono essere visualizzati non solo in RViz, come mostra la Figura 3.3, ma anche nell'albero dei *tf* (tf_tree), come mostra la Figura 3.7.

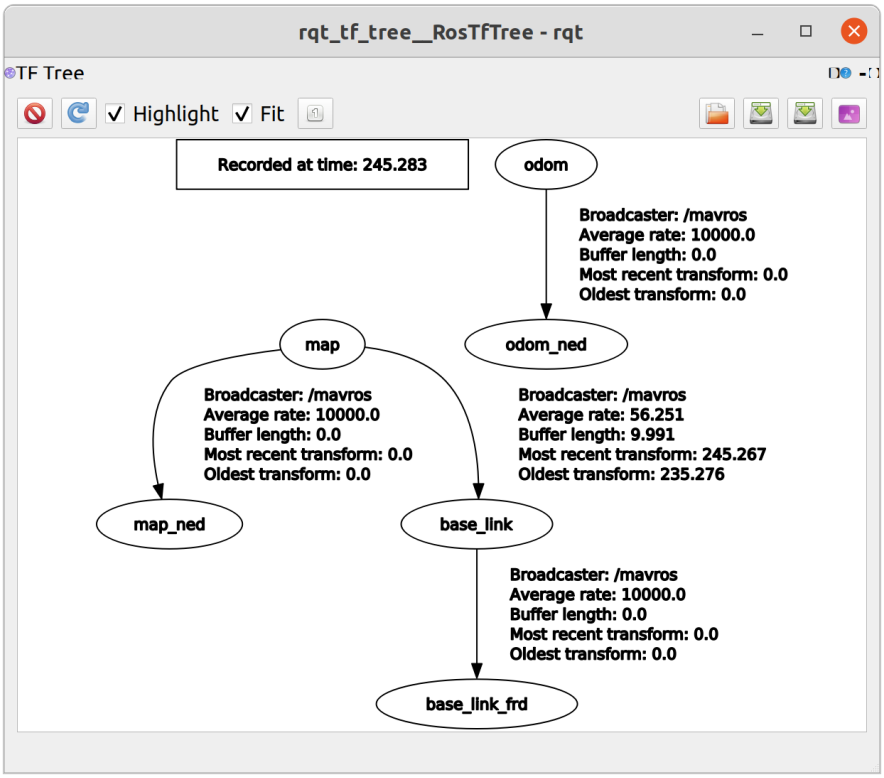


**Figura 3.7:** Task frames del quadrirotore simulato, visualizzate nel tf_tree. Il riferimento del sistema a terra è indicato con *map*, quello solidale al quadrirotore con *base_link*.

# 4

# IDENTIFICAZIONE

Data la crescente necessità di disporre di quadrirotori ad alte prestazioni, è necessario disporre di modelli accurati che descrivano al meglio il sistema reale, al fine di simulare ed implementare leggi di controllo ad alte prestazioni. Per questo motivo, il campo dell'identificazione di sistema è stato molto discusso negli ultimi decenni. In questo capitolo si cerca di identificare il modello dinamico di un quadrirotore.

## 4.1 IDENTIFICAZIONE DI SISTEMI DINAMICI

L'identificazione di sistema consiste nella costruzione di modelli matematici partendo da dati sperimentali (input-output). Le tecniche di identificazione di sistema possono essere caratterizzate in termini di struttura del modello, tipo del modello e dominio dei dati input-output.

Esistono tre possibili tecniche di identificazione di sistema, che differiscono per quanto riguarda il grado di conoscenza e accesso al sistema oggetto di studio.

Il primo approccio è quello a scatola bianca (white box) e si applica quando si ha una conoscenza completa e dettagliata del sistema in esame. Si conoscono infatti tutte le equazioni matematiche e/o le leggi fisiche che governano il comportamento del sistema. Pertanto, l'identificazione si basa semplicemente sulla stima dei parametri delle equazioni conosciute, utilizzando dati sperimentali per ottimizzare o confermare il modello. Si tratta di una tecnica molto precisa, ma che richiede una conoscenza approfondita del sistema.

Il secondo approccio è quello a scatola grigia (gray box) e si applica quando si ha una conoscenza parziale del sistema in esame. Potrebbe essere noto solo un sottoinsieme delle equazioni o dei dettagli del sistema, mentre altri aspetti sono sconosciuti o approssimati. L'idea è quella di combinare la conoscenza parziale con dati sperimentali per costruire un modello che rappresenti accuratamente il sistema. Questo

approccio è utile quando non si dispone di una comprensione completa del sistema, ma si hanno alcune informazioni chiave.

Il terzo approccio è quello a scatola nera (black box) e si applica quando non si ha conoscenza delle equazioni o dei dettagli interni del sistema. Il sistema viene trattato appunto come una scatola nera in cui si osservano le relazioni tra l'input e l'output, ma non si cerca di comprendere il funzionamento interno. L'obiettivo è identificare un modello empirico o una funzione di trasferimento (come si vedrà nel seguito) che descriva il comportamento del sistema basandosi solo su dati sperimentali. Questo approccio è utilizzato quando il sistema è molto complesso o non è possibile ottenere informazioni dettagliate sul suo funzionamento.

In questa tesi si utilizzerà MATLAB System Identification Toolbox [14] [15], uno strumento fornito da MATLAB per l'identificazione di sistemi dinamici. Questo, tramite un'interfaccia grafica intuitiva, consente di inserire dati input-output e identificare il sistema secondo l'approccio black box. Le tre principali funzioni del tool sono le seguenti:

1. **Data Preprocessing**. I dati input-output registrati possono essere elaborati in vario modo. Ad esempio possono essere filtrati, possono essere trasformati in segnali a media nulla oppure si può ridurre la finestra di interesse ad un determinato intervallo temporale. Queste tecniche sono molto utili in quanto i dati, così come sono stati raccolti, non sempre sono sufficienti per identificare un modello accurato.

2. **System Identification**. Esistono numerosi tipi di modelli identificabili. Si possono identificare sia modelli Single Input Single Output (SISO) che modelli Multi Input Multi Output (MIMO).

3. **System Validation**. Una volta identificato il modello si è interessati a capire se il suo comportamento è effettivamente simile (ed eventualmente quanto) a quello reale. MATLAB fornisce una funzione che simula la risposta del modello identificato (inserendo nel sistema l'input misurato) e la sovrappone all'uscita misurata [16]. Inoltre, mostra anche la misura del Normalized Root Mean Square Error (NRMSE), che descrive la bontà dell'adattamento tra la risposta simulata e i dati di misurazione di output ed è definito come mostra la 24.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}} \tag{23}$$

$$\text{NRMSE} = \frac{\text{RMSE}}{y_{max} - y_{min}} \tag{24}$$

## 4.2 SIMULAZIONI DI VOLO

Per poter identificare al meglio il sistema del quadrirotore tramite delle funzioni di trasferimento SISO è necessario effettuare delle simulazioni di volo che siano in grado di descrivere i singoli aspetti coinvolti nella dinamica del sistema.

Il quadrirotore utilizzato nell'ambiente di simulazione ROS è nello specifico un sistema ad anello aperto instabile. Questo non consente di effettuare delle simulazioni valide per l'identificazione del sistema, in quanto rende impossibile effettuare una traiettoria di volo desiderata agendo direttamente sugli angoli di Eulero.

Per questo motivo, si è deciso di chiudere l'anello del sistema con un controllore in grado di stabilizzare il drone.

### 4.2.1 *Controllore di Posizione*

Si è deciso di utilizzare un controllore, fornito già implementato, in grado di mantenere il drone in stato di hovering una volta decollato e in grado di farlo muovere verso uno specifico punto dello spazio. L'anello più interno è responsabile del controllo di assetto (Attitude Controller), quello più esterno del controllo di posizione (Position Controller), come mostra Figura 4.1. Per un corretto funzionamento di questo schema di controllo è necessario il principio di disaccoppiamento in frequenza: la larghezza di banda del sistema di controllo di assetto (anello interno) deve essere significativamente maggiore rispetto a quella del sistema di controllo di posizione (anello esterno). In altre parole, la dinamica dell'anello interno deve rispondere molto più velocemente rispetto alla dinamica di quello esterno. In questo modo, il regolatore di posizione può essere progettato in modo indipendente dal regolatore di assetto. Questo schema di controllo è ampiamente utilizzato in ambito aeronautico e nello specifico per il controllo di quadrirotori [3] [4] [5] [17] [18] [19].

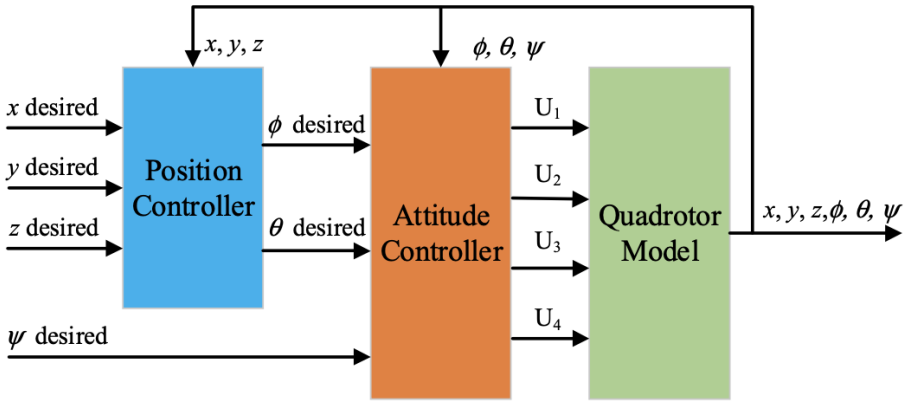


**Figura 4.1:** Schema di controllo di un quadrirotore. L'anello più interno è responsabile del controllo di assetto (Attitude Controller), quello più esterno del controllo di posizione (Position Controller).

Le variabili considerate come input e output del modello sono rispettivamente quelle di controllo e di misura nello schema mostrato in Figura 4.1. Come già spiegato, il quadrirotore è controllabile variando le velocità angolari dei quattro rotori ($\Omega_i$, $i = 1, 2, 3, 4$). Tuttavia, le forze e i momenti generati dai rotori sono funzioni quadratiche delle velocità angolari, quindi considerare le velocità angolari come variabili di ingresso del modello comporterebbe sia una forte non linearità nelle equazioni (a causa della dipendenza quadratica), sia un forte accoppiamento tra gli ingressi e la risposta dei singoli assi (poiché le velocità angolari dei quattro rotori vengono sempre variate simultaneamente, indipendentemente dal grado di libertà da controllare). Per evitare queste difficoltà solitamente si considerano come ingressi i segnali di controllo $U_i$, $i = 1, 2, 3, 4$ definiti nell'equazione 15 del Capitolo 2. Siccome questi quattro segnali agiscono sulle quattro manovre principali del velivolo, si possono equivalentemente considerare gli angoli di Eulero e la forza di spinta come ingressi del sistema. Non è oggetto della tesi l'identificazione della dinamica dell'angolo di yaw, quindi si considerano in ingresso al modello la forza di spinta e gli angoli di roll e pitch. Infine, le variabili in uscita sono esattamente quelle di misura, cioè le coordinate del velivolo nello spazio.

Dal punto di vista implementativo, il controllore è un pacchetto ROS contenente nodi (Python) di varia utilità. Alcuni nodi gestiscono l'algoritmo matematico di controllo, altri si occupano di presentare all'utente un'interfaccia grafica per controllare lo spostamento del drone. Come mostra Figura 4.2, l'utente può effettuare le seguenti operazioni, che corrispondo a servizi ROS nel codice sorgente:

- **Take Off** fa decollare il drone all'altezza specificata e lo stabilizza (stato di hovering);

- **Go To Point** imposta il setpoint da raggiungere verso le coordinate desiderate (rispetto al sistema di riferimento specificato, che può essere sia quello a terra che quello solidale al velivolo);
- **Go Home** riporta il drone nelle componenti del piano XY iniziali in cui è avvenuto il decollo;
- **Land** fa atterrare il drone nel punto del piano XY dove si trova, disarmando i motori.


**Figura 4.2:** Interfaccia grafica del controllore.

### 4.2.2 *Registrazione e Memorizzazione Dati*

I dati di interesse sono gli angoli di Eulero responsabili dei movimenti laterali del drone e la forza di spinta per quanto riguarda gli input e le coordinate di posizione per quanto riguarda gli output. Per registrare e memorizzare queste informazioni sono state aggiunte alcune righe di codice in alcuni nodi ROS del controllore. In particolare, per la gestione dei dati sono state utilizzate le librerie NumPy [20] e SciPy [21].

Prima di tutto sono stati inizializzati sei array NumPy (tre per gli input e tre per gli output), poi sono stati riempiti all'interno del ciclo di controllo del drone utilizzando la funzione *append()*, fornita dalla libreria NumPy. Il ciclo di controllo viene aggiornato con una frequenza di 100 Hz, quindi, al termine di una simulazione di volo ogni array sarà un segnale del tempo campionato con tempo di campionamento pari a 0.01 secondi. Il numero di campioni dipende dalla durata della simulazione e, ovviamente, tanto più è grande tanto più sarà precisa l'identificazione del modello. Ipotizzando

che ogni volo registrato è durato in media due minuti, ogni segnale è caratterizzato da circa 12000 campioni. Sono state effettuate dieci simulazioni di volo in modo da poter identificare più modelli relativi a distinti insiemi di dati e confrontarli tra loro.

Per trasferire i dati registrati in MATLAB, dove poi avvengono i tre passi necessari per identificare e validare il sistema, si è fatto uso della funzione *savemat()* [22] della libreria input-output di SciPy (*scipy.io*).

## 4.3 IDENTIFICAZIONE DI MODELLI SISO

L'obiettivo di questa sezione è quello di identificare tre funzioni di trasferimento tempo continue, nel dominio di Laplace, che descrivano le dinamiche del quadrirotore sugli assi principali. Infatti, il quadrirotore mostra un buon grado di disaccoppiamento, cioè l'azione su uno dei segnali di controllo influisce quasi esclusivamente sul corrispondente grado di libertà [17] [23].

### 4.3.1 *Rappresentazione di Sistemi Lineari*

Nel campo dell'analisi di sistemi, un sistema dinamico SISO, lineare e tempo-invariante - Linear Time Invariant (LTI) - è descritto da un'equazione differenziale lineare a coefficienti costanti in $u(t)$ e $y(t)$ (rispettivamente ingresso e uscita). Assumendo condizioni iniziali nulle e applicando la trasformata di Laplace, si ottiene un'equazione algebrica del tipo:

$$Y(s) = W(s)U(s)$$

dove $U(s)$ e $Y(s)$ sono i segnali di ingresso e uscita trasformati e $W(s)$ è la funzione di trasferimento che li lega. Inoltre, una equazione differenziale lineare a coefficienti costanti genera sempre una funzione di trasferimento razionale fratta, cioè esprimibile come il rapporto di due polinomi.

### 4.3.2 *Funzione di Trasferimento angolo di Rollio e posizione lungo Y*

I dati utilizzati per identificare la dinamica del velivolo lungo l'asse Y sono il segnale che descrive le variazioni dell'angolo di rollio (input) e il segnale che descrive le variazioni di posizione lungo lo stesso asse (output). Entrambi i segnali si presentano già a media nulla[1], quindi l'unica operazione di preprocessing effettuata è stata quella di considerare uno specifico intervallo temporale (dove il segnale è significativo). I segnali utilizzati sono mostrati in Figura 4.3. Il segnale di input, relativo all'angolo di rollio, mostra una brusca variazione prima di assestarsi a zero. Questo accade perché al termine della simulazione di volo, quando si fa atterrare il drone, il controllore cerca di stabilizzare il drone nel punto del piano XY dove si trova. Ma non appena il drone inizia il processo di atterraggio, il controllore smette di inviare controlli relativi all'assetto del velivolo e l'angolo di rollio (in questo caso) rimane nullo fino al termine della simulazione.

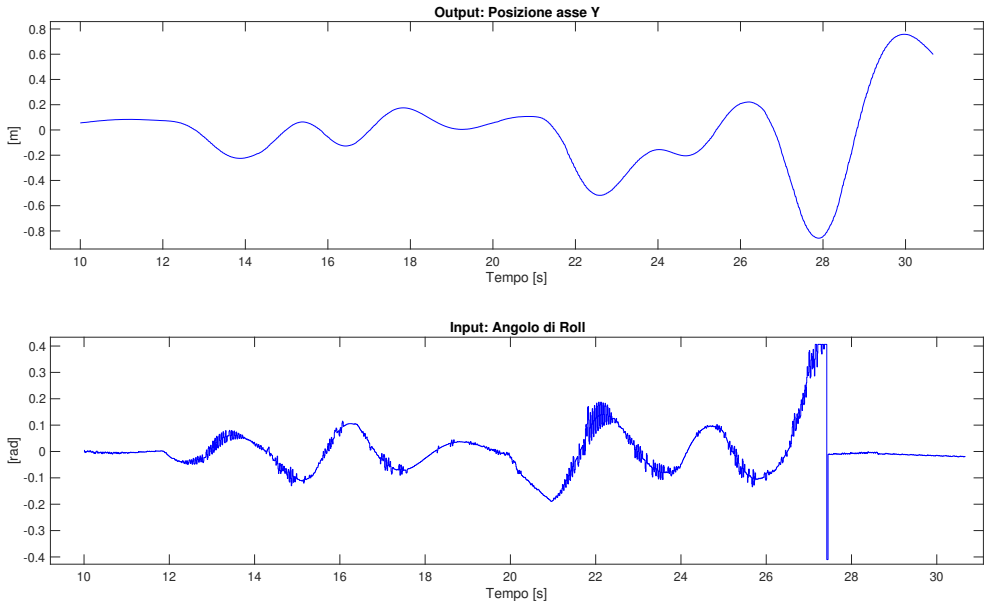


**Figura 4.3:** Dati input-output per identificare la dinamica del rollio. Input: variazione dell'angolo di rollio nel tempo. Output: variazione della posizione lungo l'asse Y nel tempo.

La funzione di trasferimento identificata che lega l'angolo di rollio (roll) allo spostamento del velivolo lungo l'asse Y è definita nell'equazione 25.

1 Il processo che elimina la media di un segnale e lo rende a media nulla viene denominato detrending e consente di rimuovere gli effetti del valor medio per mostrare solo le differenze significative. Consente quindi di identificare in modo più preciso l'andamento del segnale.

$$W_Y(s) = \frac{-1.022s^2 + 0.6857s + 1.873}{s^3 + 0.008794s^2 + 1.456s + 0.1566} = \frac{-1.022(s - 1.73)(s + 1.059)}{(s + 0.1068)(s - 0.0490 \pm j1.2098)} \quad (25)$$

Si tratta di un sistema lineare tempo continuo del terzo ordine con un polo reale e due poli complessi e coniugati.

Completata la fase di identificazione, si passa alla terza e ultima fase, cioè la validazione del sistema. La funzione *compare()* [16] mostra una buona compatibilità del modello identificato con quello reale, come mostra Figura 4.4. Il NRMSE risulta in un 71.3% di compatibilità.

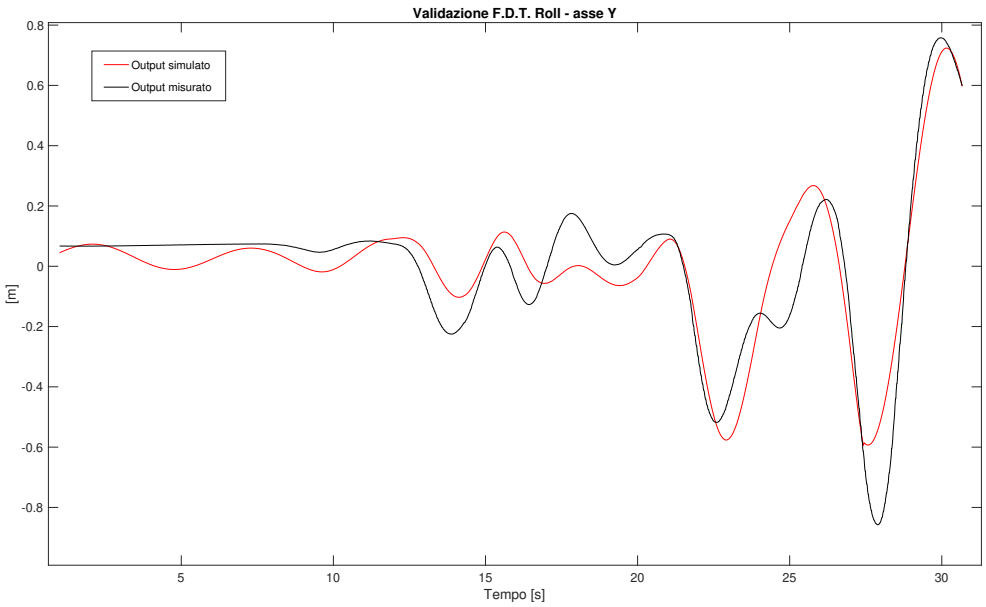


**Figura 4.4:** Validazione F.D.T. tra rollio e posizione lungo Y.

### 4.3.3 *Funzione di Trasferimento angolo di Beccheggio e posizione lungo X*

I dati utilizzati per identificare la dinamica del velivolo lungo l'asse X sono il segnale che descrive le variazioni dell'angolo di beccheggio (input) e il segnale che descrive le variazioni di posizione lungo lo stesso asse (output). Anche qui, entrambi i segnali si presentano già a media nulla, quindi l'unica operazione di preprocessing effettuata è stata quella di considerare uno specifico intervallo temporale (dove il segnale è significativo). I segnali utilizzati sono mostrati in Figura 4.5.

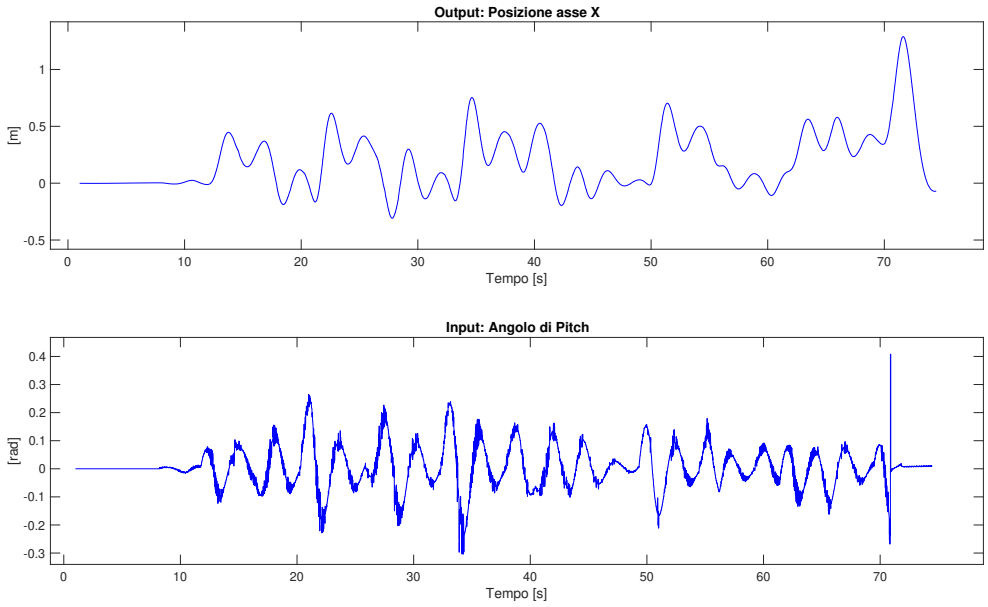


**Figura 4.5:** Dati input-output per identificare la dinamica del beccheggio. Input: variazione dell'angolo di beccheggio nel tempo. Output: variazione della posizione lungo l'asse X nel tempo.

La funzione di trasferimento identificata che lega l'angolo di beccheggio (pitch) allo spostamento del velivolo lungo l'asse X è definita nell'equazione 26.

$$W_X(s) = \frac{-3.224s^2 + 3.491s + 1.71}{s^3 + 1.16s^2 + 0.1916s + 0.1929} = \frac{-3.224(s - 1.449)(s + 0.3661)}{(s + 1.14)(s - 0.0714 \pm j1.0995)} \tag{26}$$

Anche in questo caso (non sorprendentemente per la simmetria del quadrirotore rispetto ai movimenti lungo Y e X) si ottiene un sistema lineare tempo continuo del terzo ordine con un polo reale e due poli complessi e coniugati.

Per quanto riguarda la validazione, la funzione *compare()* [16] mostra ancora una buona compatibilità del modello identificato con quello reale, come mostra Figura 4.6. Il NRMSE risulta in un 69.7% di compatibilità.

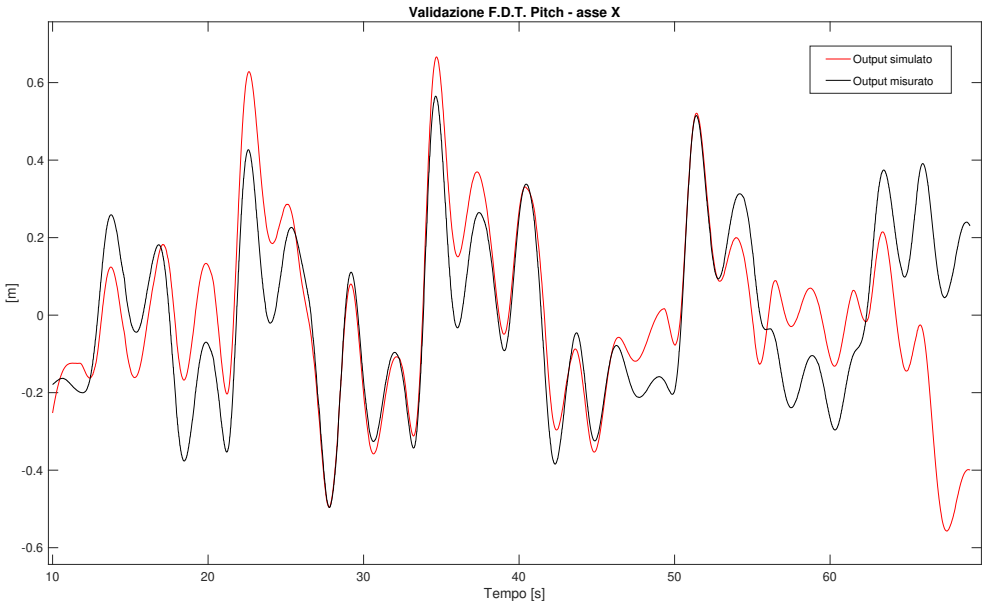


**Figura 4.6:** Validazione F.D.T. tra beccheggio e posizione lungo X.

### 4.3.4 *Funzione di Trasferimento forza di Spinta e posizione lungo Z*

I dati utilizzati per identificare la dinamica del velivolo lungo l'asse Z sono il segnale che descrive le variazioni della forza di spinta (input) e il segnale che descrive le variazioni di posizione lungo lo stesso asse (output). Il drone subito dopo il decollo entra in stato di hovering, cioè rimane fisso sull'origine del piano XY ma ad una certa quota lungo l'asse Z. Questo significa che, a differenza dei due casi precedenti, in questo caso i segnali input-output non sono a media nulla. Pertanto, nel preprocessing si rendono entrambi a media nulla, come mostra Figura 4.7.

Il segnale di ingresso contenente le varie spinte applicate nel tempo è caratterizzato da una serie di gradini, di diversa ampiezza, in modo da evidenziare in maniera marcata il comportamento del sistema. Si è cercato di fare la stessa cosa per gli ingressi utilizzati nei casi precedenti, ma il meccanismo di stabilizzazione è risultato meno efficiente per spostamenti sul piano XY (rispetto a spostamenti lungo l'asse Z), ed è per questo che i segnali sono sicuramente più complicati da interpretare.

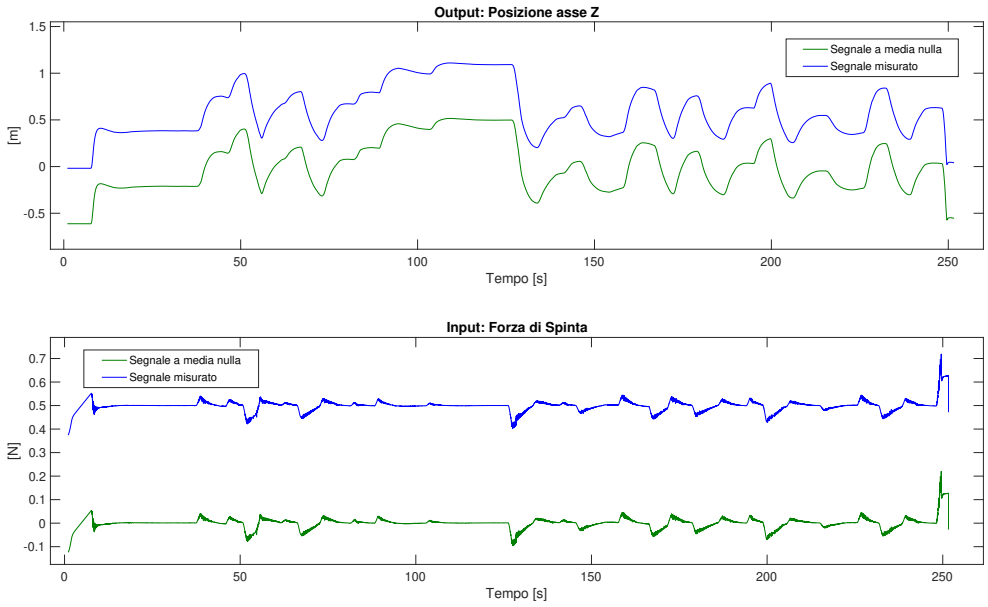


**Figura 4.7:** Dati input-output per identificare la dinamica della spinta. Input: forza di spinta applicata nel tempo. Output: variazione della posizione lungo l'asse Z nel tempo.

La funzione di trasferimento identificata che lega la forza di spinta allo spostamento del velivolo lungo l'asse Z è mostrata nell'equazione 27.

$$W_Z(s) = \frac{-144.02s^2 + 97.89s + 25.18}{s^3 + 39.98s^2 + 12.35s + 0.7329} = \frac{-144.02(s - 0.8787)(s + 0.199)}{(s + 39.67)(s + 0.231)(s + 0.07999)} \quad (27)$$

Si ottiene ancora un sistema lineare tempo continuo del terzo ordine, ma con tre poli reali negativi.

Passando alla validazione, la funzione *compare()* [16] mostra un'ottima compatibilità del modello identificato con quello reale, come mostra Figura 4.8. Il NRMSE risulta in un 78.4% di compatibilità.

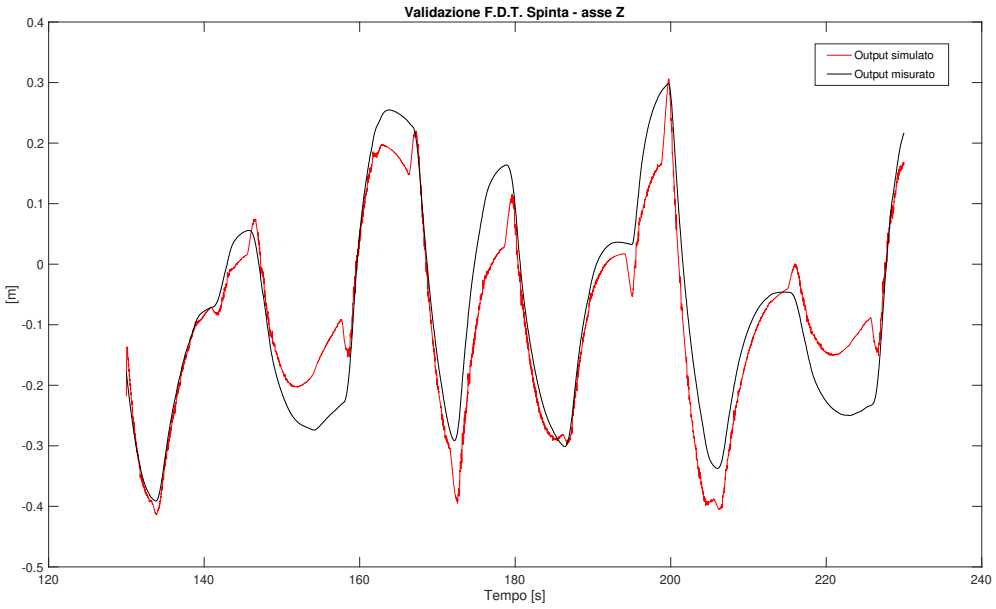


**Figura 4.8:** Validazione F.D.T. tra spinta e posizione lungo Z.

Si noti che nella validazione dei modelli è stato riportato il segnale in uscita (e la risposta del modello identificato) ristretto ad un preciso intervallo di tempo. Tale intervallo è stato scelto in modo tale da ottenere la migliore compatibilità del modello con i dati registrati.

### 4.3.5 *Analisi in Frequenza*

In questa sezione si discutono e confrontano le risposte in frequenza relative alle tre funzioni di trasferimento identificate.

Come mostra Figura 4.9 e come si nota dalla funzione di trasferimento $W_Z(s)$ definita nella 27, la risposta del quadrirotore lungo l'asse verticale è asintoticamente stabile e non oscillatoria, essendo caratterizzata esclusivamente da poli reali situati nel semiasse negativo.

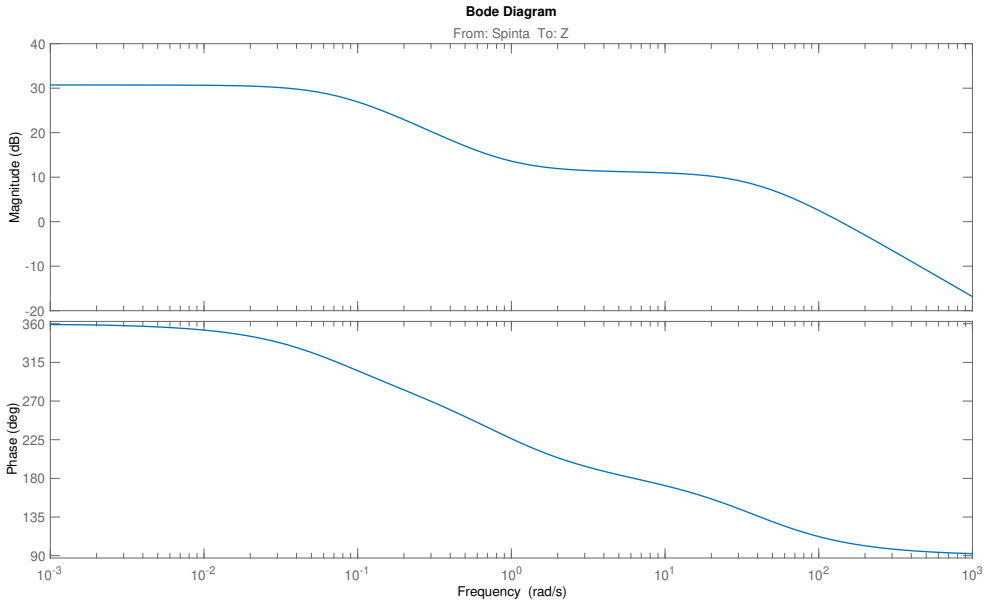


**Figura 4.9:** Diagrammi di Bode (modulo e fase) della F.D.T. tra spinta e posizione lungo Z.

D'altra parte, non sorprendentemente, le risposte in frequenza relative all'asse laterale e longitudinale (rispettivamente Figura 4.10 e Figura 4.11) sono instabili (poli a parte reale positiva) e caratterizzate da modi oscillatori (poli complessi e coniugati). Inoltre, c'è una forte simmetria tra questi due modelli. In particolare, osservando le funzioni di trasferimento $W_X(s)$ e $W_Z(s)$ definite dalle equazioni 26 e 27, si nota una simile dislocazione di poli e zeri nel piano complesso e questo denota che il modello identificato è preciso abbastanza da aver catturato la forte simmetria del quadrirotore.

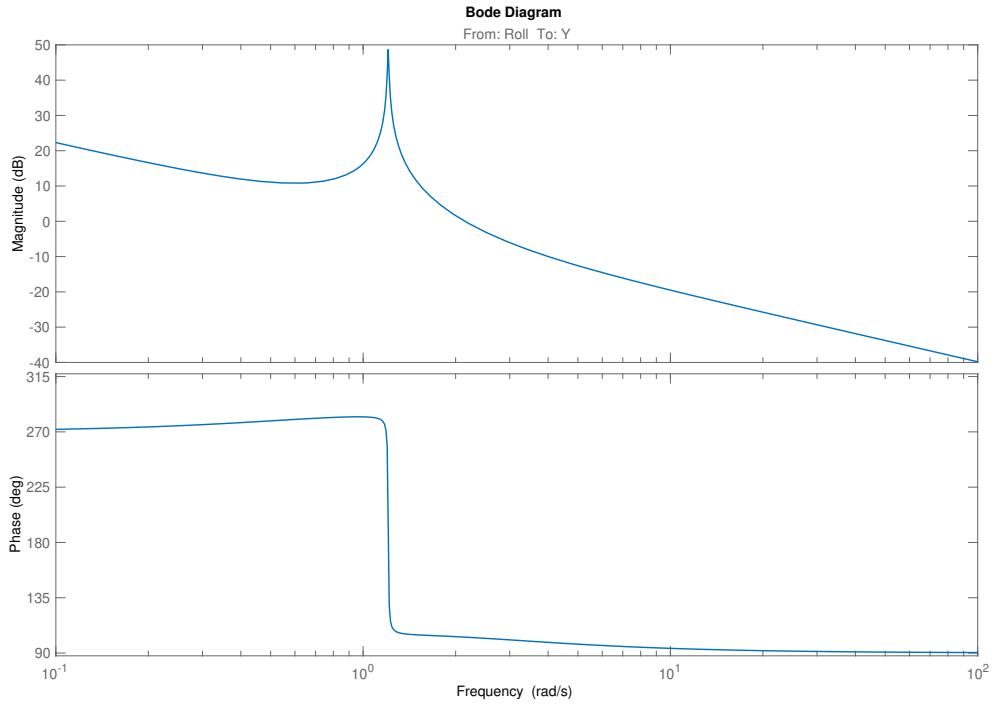


**Figura 4.10:** Diagrammi di Bode (modulo e fase) della F.D.T. tra rollio e posizione lungo Y.

A causa delle limitazioni nella durata degli esperimenti e dell'azione a bassa frequenza del controllore in retroazione, che maschera la vera dinamica del sistema, ci si aspetta una significativa incertezza nella parte a bassa frequenza delle risposte dei modelli SISO identificati. Questo si verifica maggiormente nei due modelli relativi agli spostamenti del drone nel piano XY (roll e pitch), essendo modelli instabili. Il modello lungo l'asse Z, infatti, essendo stabile, potrebbe anche essere simulato a ciclo aperto e non risentirebbe affatto dell'azione del controllore a bassa frequenza.

Ad ogni modo, quest'ultima problematica non è così importante dato il modello identificato è principalmente destinato alla progettazione di leggi di controllo. Quindi, è fondamentale avere un modello accurato introno alla frequenza di taglio, ma si può accettare un certo grado di incertezza a basse (e alte) frequenze.

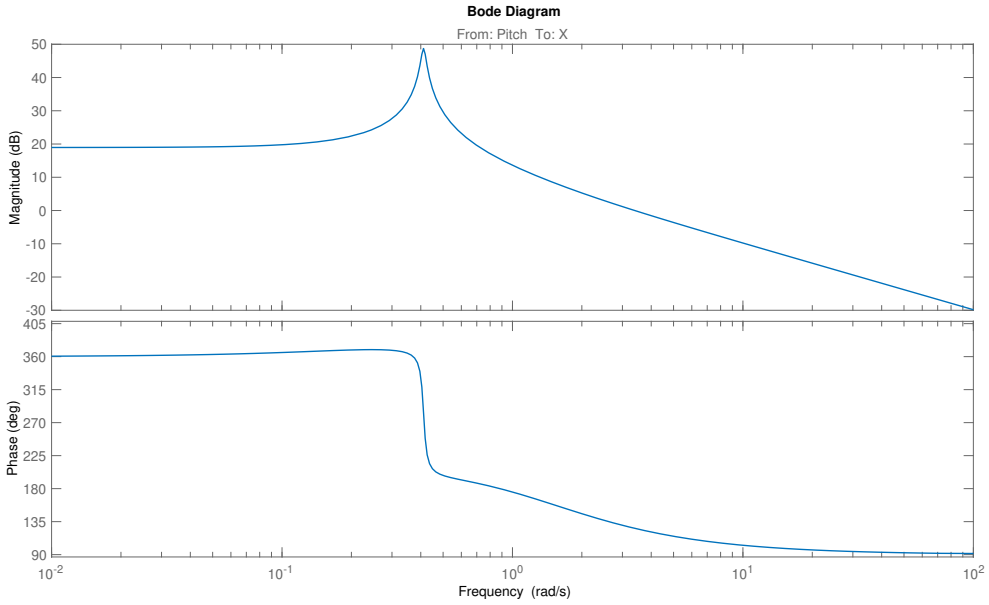


**Figura 4.11:** Diagrammi di Bode (modulo e fase) della F.D.T. tra beccheggio e posizione lungo X.

# 5

# CONCLUSIONI E SVILUPPI FUTURI

## 5.1 CONCLUSIONI

Nel Capitolo 2 si è descritto nel dettaglio il modello matematico di un quadrirotore con disposizione delle eliche a X. Nel Capitolo 3, dopo aver spiegato brevemente il funzionamento di ROS, sono stati descritti e spiegati i moduli software utilizzati per configurare il simulatore. Infine, nel Capitolo 4 sono state identificate le tre funzioni di trasferimento SISO cercate e, visto che per le simulazioni di volo si è fatto uso di un controllore di posizione, si è anche parlato del funzionamento e dell'implementazione di tale controllore.

Le funzioni di trasferimento ricavate risultano sufficientemente compatibili con i dati input-output registrati nelle simulazioni e colgono le caratteristiche dinamiche essenziali del quadrirotore lungo tutti gli assi.

## 5.2 SVILUPPI FUTURI

Questo lavoro è un importante punto di partenza nello studio della dinamica di un quadrirotore ed è sicuramente utile per la progettazione e la simulazione di eventuali leggi di controllo. Tuttavia, può essere approfondito ulteriormente.

Infatti, non solo l'identificazione di sistema dipende dal tipo del modello (white box, gray box e black box), ma varia anche in base alla sua struttura. In questo lavoro sono stati identificati tre modelli lineari tempo continui nel dominio di Laplace, ma esistono molte altre strutture e sono spiegate nel dettaglio nella documentazione ufficiale di MATLAB System Identification Toolbox [14] [15].

Per un'identificazione più approfondita si potrebbe utilizzare un modello nello spazio di stato di tipo non lineare, in grado di descrivere la dinamica del sistema completa e non disaccoppiata lungo i tre assi principali. Poi, per ottenere una misura

del grado di accuratezza delle funzioni di trasferimento SISO individuate, sarebbe interessante confrontare il comportamento del sistema MIMO ristretto a tali dinamiche disaccoppiate. In questo modo si potrebbe comprendere se effettivamente agendo su un particolare segnale di controllo si influenza solo il corrispondente grado di libertà, come si è assunto in questo lavoro, o se ci sono delle dipendenze più complesse interne al sistema (che potrebbero diventare oggetto di studio).

Esistono anche modelli più avanzati per l'identificazione di sistemi non lineari, come i modelli ARX e i modelli Hammerstein-Wiener. Più informazioni relative a questi ultimi si trovano nella documentazione MATLAB [14] [15].

Inoltre, si potrebbe utilizzare il software Simulink [24] per progettare e simulare schemi di controllo sulla base del modello identificato.

# BIBLIOGRAFIA E SITOGRAFIA


[1] ABI Research. *State of the robotics market*. https://go.abiresearch.com/lp-state-of-the-robotics-market. 2018 (cit. a p. 1).

[2] James S. Aber, Irene Marzolff, Johannes B. Ries, and Susan E.W. Aber. "Chapter 8 - Unmanned Aerial Systems." In: *Small-Format Aerial Photography and UAS Imagery (Second Edition)*. Ed. by James S. Aber, Irene Marzolff, Johannes B. Ries, and Susan E.W. Aber. Second Edition. Academic Press, 2019, pp. 119–139. URL: https://www.sciencedirect.com/science/article/pii/B9780128129425000082 (cit. a p. 3).

[3] Tommaso Bresciani. *Modelling, Identification and Control of a Quadrotor Helicopter*. 2008 (cit. a pp. 4, 32).

[4] Samir Bouabdallah. *Design and Control of Quadrotors with Application to Autonomous Flying*. 2007 (cit. a pp. 4, 32).

[5] M Islam, M Okasha, and M M Idres. *Dynamics and control of quadcopter using linear model predictive control approach*. 2017 (cit. a pp. 4, 32).

[6] Leandra Vicci. *Quaternions and Rotations in 3-Space: The Algebra and its Geometric Interpretation*. 2001 (cit. a p. 15).

[7] *ROS*. https://www.ros.org/ (cit. a p. 20).

[8] *ROS 2*. https://docs.ros.org/en/iron/index.html (cit. a p. 20).

[9] Tully Foote. *tf: The transform library*. 2013 (cit. a p. 23).

[10] Verena Elisabeth Kremer. *Quaternions and SLERP*. 2008 (cit. a p. 24).

[11] Intelligent Quads. *IQ Simulations*. https://github.com/Intelligent-Quads/iq_sim (cit. a p. 26).

[12] *Software In The Loop (SITL)*. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html (cit. a p. 27).

[13] *MAVROS*. https://github.com/mavlink/mavros (cit. a p. 28).

[14] Lennart Ljung. *System identification toolbox: User's guide*. Citeseer, 1995 (cit. a pp. 31, 45, 46).

[15] MATLAB. *System Identification Toolbox*. https://it.mathworks.com/help/ident/ (cit. a pp. 31, 45, 46).

[16] MATLAB. *Compare*. https://it.mathworks.com/help/ident/ref/compare.html (cit. a pp. 31, 37, 38, 40).

[17] Marco Bergamasco and Marco Lovera. "Identification of Linear Models for the Dynamics of a Hovering Quadrotor." In: *IEEE Transactions on Control Systems Technology* 22.5 (2014), pp. 1696–1707 (cit. a pp. 32, 35).

[18] M. Alpen, K. Frick, and J. Horn. "Nonlinear modeling and position control of an industrial quadrotor with on-board attitude control." In: Jan. 2010, pp. 2329 –2334 (cit. a p. 32).

[19] David H. "A flight control system for aerial robots: Algorithms and experiments." In: vol. 35. July 2002, pp. 1263–1263. ISBN: 9783902661746 (cit. a p. 32).

[20] *NumPy*. https://numpy.org/ (cit. a p. 34).

[21] *SciPy*. https://scipy.org/ (cit. a p. 34).

[22] SciPy. *scipy.io.savemat*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.savemat.html (cit. a p. 35).

[23] Rui Miguel Martins de Oliveira. *Identification and Validation of a Quadrotor's Model Dynamics*. October 2014 (cit. a p. 35).

[24] MATLAB. *Simulink*. https://it.mathworks.com/products/simulink.html (cit. a p. 46).

# APPENDICE

Nel seguito si riportano le sezioni più significative del codice Python relativo al controllore di posizione (*controller.py*). In particolare, le modifiche che sono state apportate per poter registrare e memorizzare i dati input-output di interesse.

```
#! /usr/bin/env python3

import rospy
import numpy as np
from scipy import io as sio
import tf2_ros as tf2
from scipy.spatial.transform import Rotation as R

class Controller:

  def __init__(self):

    # INPUTS
    self.controller_roll = np.array([])
    self.controller_pitch = np.array([])
    self.controller_yaw = np.array([])
    self.controller_thrust = np.array([])

    # OUTPUTS
    self.pose_x = np.array([])
    self.pose_y = np.array([])
    self.pose_z = np.array([])

    # TRANSFORMS
    self.tf_buffer = tf2.Buffer()
    self.tf_listener = tf2.TransformListener(self.tf_buffer)
    try:
    transform = self.tf_buffer.lookup_transform(self.world_frame, self.drone_frame,
    rospy.Time(0), rospy.Duration(5))
    except tf2.LookupException:
    print("ERROR: No transformation between {} and {}.".format(self.world_frame, self.
    drone_frame))
    exit(-1)
```

```
  # POSE AND ORIENTATION (QUATERNION)
  self.init_pose = np.array([transform.transform.translation.x,
                             transform.transform.translation.y,
                             transform.transform.translation.z])
  self.init_orientation = np.array([transform.transform.rotation.x,
                            transform.transform.rotation.y,
                            transform.transform.rotation.z,
                            transform.transform.rotation.w])
  self.current_pose = copy.deepcopy(self.init_pose)
  self.current_orientation = copy.deepcopy(self.init_orientation)

def _control_loop(self):
  while not self.kill_threads:

  # ROTATION MATRIX
  r = R.from_quat(current_orientation)
  current_rotation_matrix = np.array(r.as_matrix()).reshape((3, 3))

  if self.current_state != self.drone_states["GROUND"]:

    # REGISTRAZIONE OUTPUT (X,Y,Z)
    xp = np.append(self.pose_x, [current_pose[0]])
    self.pose_x = xp
    yp = np.append(self.pose_y, [current_pose[1]])
    self.pose_y = yp
    zp = np.append(self.pose_z, [current_pose[2]])
    self.pose_z = zp



    if not self.current_state == self.drone_states["FAIL"]:
       if not self.current_state == self.drone_states["TAKING_OFF"]:

          # CONTROLLO STABILIZZATO
          if self.angle_mode:

          # CALCOLO PHI, THETA, PSI, T

          # REGISTRAZIONE INPUT (PHI, THETA, PSI, T)
          r1 = np.append(self.controller_roll, [roll])
          self.controller_roll = r1
          p1 = np.append(self.controller_pitch, [pitch])
          self.controller_pitch = p1
          y1 = np.append(self.controller_yaw, [yaw])
          self.controller_yaw = y1
          t1 = np.append(self.controller_thrust, [trust])
          self.controller_thrust = t1
```

```
            self.publish_attitude_angle(roll, pitch, yaw, trust)

            # CONTROLLO ACROBATICO
            else:
            self.publish_attitude_acro(wtr[0], wtr[1], wtr[2],

        else: # TAKE OFF

          # CALCOLO PHI, THETA, PSI, T

          # REGISTRAZIONE INPUT (PHI, THETA, PSI, T)
          r2 = np.append(self.controller_roll, [roll])
          self.controller_roll = r2
          p2 = np.append(self.controller_pitch, [pitch])
          self.controller_pitch = p2
          y2 = np.append(self.controller_yaw, [yaw])
          self.controller_yaw = y2
          t2 = np.append(self.controller_thrust, [trust])
          self.controller_thrust = t2

          self.publish_attitude_angle(roll, pitch, yaw, trust)

    # DATA DICTIONARY
    input_output_dic = {
    "controller_roll": self.controller_roll,
    "controller_pitch": self.controller_pitch,
    "controller_yaw": self.controller_yaw,
    "controller_thrust": self.controller_thrust,
    "pose_x": self.pose_x,
    "pose_y": self.pose_y,
    "pose_z": self.pose_z
    }

    # MATLAB
    sio.savemat("input_output_data.mat", input_output_dic)

    self.rate_pid.sleep()

if __name__ == "__main__":
    controller = Controller()
    while not rospy.is_shutdown():
        controller._control_loop()
```