



Dipartimento di Ingegneria

Tesi di Laurea Magistrale in
Ingegneria Informatica e Robotica

Implementazione di un Framework Hardware In The Loop per il controllo ottimo in retroazione di un quadricottero

Relatore

Prof. Francesco Ferrante

Correlatore

Dott. Mirko Leomanni

Candidato

Paolo Leopardi

Anno Accademico 2021/2022

Indice

Sommario	6
1 Introduzione	7
1.1 Stato dell'arte	8
2 Modellazione e controllo del quadricottero	11
2.1 Principio di funzionamento	11
2.2 Modello dinamico	13
2.2.1 Modello linearizzato	14
2.3 Sviluppo del controllore	16
2.3.1 Osservatore asintotico dello stato	16
2.3.2 Controllo LQR	16
2.3.3 State feedback	17
2.3.4 Output feedback	18
2.3.5 State feedback con termine integrale	20
3 Autopilota PX4	22
3.1 Concetti generali	22
3.1.1 QGroundControl	23
3.1.2 MAVLink	24
3.2 Architettura software	24
3.2.1 Flight Stack	26
3.3 SITL vs HITL	27
3.4 UAV Toolbox Support Package for PX4 Autopilots	28
4 Implementazione	30
4.1 Simulazione Simulink	32
4.1.1 Output feedback	32
4.1.2 State feedback con termine integrale	34
4.2 Simulazione SITL	35
4.2.1 Output feedback	36
4.2.2 State feedback con termine integrale	38
4.3 Simulazione HITL	39
4.3.1 Output feedback	40
4.3.2 State feedback con termine integrale	41
5 Esperimenti e Risultati	43
5.1 Hovering	44
5.1.1 Simulazione Simulink	44
5.1.2 Simulazione HITL	45
5.2 Hovering con massa variabile dinamicamente	47
6 Conclusioni e Sviluppi Futuri	49

Indice degli acronimi

ANN	Artificial Neural Network
EKF	Extended Kalman Filter
GCS	Ground Control Station
HITL	Hardware In The Loop
ITAE	Integral Time Absolute Error
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
NED	North-East-Down
PID	Proportional-Integral-Derivative
QGC	QGroundControl
RPY	Roll-Pitch-Yaw
SITL	Software In The Loop
UAV	Unmanned Aerial Vehicle

Elenco delle figure

1.1	Drone commerciale DJI Mavic Air 2	7
1.2	Pixracer R15.	8
1.3	Classificazione delle tecniche di controllo.	9
2.1	Senso di rotazione delle eliche.	11
2.2	Forze generate dal quadricottero durante il moto laterale.	12
2.3	Angoli di rollio, beccheggio e imbardata.	13
2.4	Linearizzazione della funzione $y = x^2$	14
2.5	Schema con osservatore per la stima dello stato.	16
2.6	Schema di controllo State feedback.	18
2.7	Schema di controllo Output feedback.	20
2.8	Schema di controllo State feedback con termine integrale.	21
3.1	Logo PX4.	22
3.2	Controller di volo Pixhawk 4, Pixhawk 4 Mini, Pixhawk 3 Pro.	23
3.3	Schermata Fly View di QGroundControl.	23
3.4	Logo MAVLink.	24
3.5	Architettura software PX4.	25
3.6	Flight stack PX4.	26
3.7	Cascata di regolatori PID controller PX4.	26
3.8	Schema simulazione HITL.	27
3.9	Sostituzione dei moduli di controllo PX4	28
4.1	Schema Simulink controller Output feedback.	34
4.2	Schema Simulink controller state feedback con termine integrale.	35
4.3	Schema SITL controller Output feedback.	37
4.4	Sottosistema create PX4 uORB message controller Output feedback.	38
4.5	Schema SITL controller State feedback con termine integrale.	38
4.6	Sottosistema create PX4 uORB message controller State feedback con termine integrale.	39
4.7	Schema del modello dinamico HITL.	40
4.8	Schema HITL controller Output feedback.	41
4.9	Sottosistema controller Output feedback HITL.	41
4.10	Schema HITL controller Output feedback.	42
4.11	Sottosistema controller Output feedback HITL.	42
5.1	Schermata Fly View simulazione HITL.	43
5.2	Quota z Hovering Output feedback Simulink.	44
5.3	Segnali di controllo Hovering Output feedback Simulink.	44
5.4	Quota z Hovering State feedback con termine integrale Simulink.	45
5.5	Segnali di controllo Hovering State feedback con termine integrale Simulink.	45
5.6	Quota z Hovering HITL.	46
5.7	Segnali di controllo Hovering HITL.	46
5.8	Quota z Hovering con massa variabile HITL.	47

5.9 Segnali di controllo <i>Hovering</i> con massa variabile HITL.	47
5.10 Evoluzione di T , m , z controller State feedback con termine integrale HITL.	48

Elenco dei codici

4.1	quadcopter_linearized_dynamics.m	31
4.2	load_model_parameters.m Simulink	32
4.3	output_feedback.m Simulink	32
4.4	integral_state_feedback.m Simulink	34
4.5	load_model_parameters.m SITL	35
4.6	output_feedback.m SITL	36
4.7	load_model_parameters.m HITL	39

Sommario

Il progetto descrive lo sviluppo di un algoritmo di controllo per un generico quadricottero grazie all'implementazione di un framework che consente di effettuare simulazioni su differenti livelli di astrazione hardware. L'obiettivo è quello di implementare uno schema di controllo che garantisca l'*hovering* del velivolo e allo stesso tempo fornire una pipeline che consenta di definire, in maniera del tutto arbitraria, sia l'algoritmo di controllo che il modello da controllare. In particolare, tramite la simulazione Hardware In The Loop, è possibile far interagire il modello dinamico simulato con un controller di volo reale (su cui è implementato lo schema di controllo) così che quest'ultimo invii i segnali di input come se stesse comandando un velivolo reale.

Il controller di volo utilizza lo stack software PX4 interfacciandosi con l'ambiente di sviluppo MATLAB/Simulink grazie al *toolbox* denominato *UAV Toolbox Support Package for PX4 Autopilots*. Così facendo è stato possibile sviluppare algoritmi di controllo ottimo in retroazione grazie all'ambiente Simulink ed implementarli successivamente nel controllore di volo.

Gli esiti delle simulazioni mostrano come gli schemi di controllo proposti consentono, seppur con risultati differenti, di garantire l'*hovering* del velivolo sotto determinate condizioni operative. Inoltre è stato possibile evidenziare la discrepanza fra le simulazioni svolte in ambiente interamente simulato rispetto all'esecuzione delle stesse impiegando però un controller di volo reale.

Il testo è organizzato in capitoli che descrivono i concetti necessari per una piena comprensione del progetto e della tematica analizzata, il contenuto degli stessi è riassunto nel seguente elenco:

Capitolo 1: presentazione generale sui droni, ambiti applicativi e linee di ricerca, introduzione al problema di controllo di un quadricottero, descrizione generale dell'ambiente di sviluppo e panoramica sullo stato dell'arte.

Capitolo 2: illustrazione del principio di funzionamento fisico del velivolo, spiegazione del modello dinamico utilizzato e delle approssimazioni introdotte, sviluppo degli algoritmi di controllo ottimo e dimostrazione matematica della correttezza degli stessi.

Capitolo 3: panoramica dello stack software PX4 con particolare attenzione al *flight stack*, illustrazione delle principali differenze fra simulazione Software In The Loop e Hardware In The Loop, analisi del *toolbox* MATLAB/Simulink per PX4.

Capitolo 4: descrizione dei codici MATLAB e degli schemi Simulink realizzati per l'implementazione del framework.

Capitolo 5: definizione degli esperimenti effettuati, presentazione e analisi dei risultati ottenuti.

Capitolo 6: considerazioni finali riguardanti gli schemi di controllo sviluppati e il framework implementato. Discussione di varie linee di sviluppo del progetto.

Capitolo 1

Introduzione

Un drone è un apparecchio volante caratterizzato dall'assenza del pilota a bordo, tipicamente questi dispositivi vengono indicati con l'acronimo anglosassone *Unmanned Aerial Vehicle* (UAV). L'impiego di questi velivoli è ormai consolidato oltre che in applicazioni militari anche per usi civili e industriali: agricoltura di precisione [1], monitoraggio ambientale [2], [3], operazioni di ricerca e salvataggio [4], [5], trasporto di carichi [6], controllo di gasdotti [7] o del traffico stradale [8]. I droni possono essere classificati in base a differenti metriche [9], una delle più comuni si basa sulla distinzione fra UAVs ad ala fissa e multirotore: i primi ricordano di fatto il design di un comune aeroplano mentre gli altri sfruttano una o più eliche per volare, i droni multirotore possono essere a loro volta catalogati in base al numero di eliche.

Un quadricottero è una particolare tipologia di drone multirotore dotato di quattro motori, controllabili indipendentemente e disposti di norma in maniera simmetrica seguendo una configurazione a '+' oppure ad 'X'. Il successo di questo velivolo è stato determinato dall'aumento delle performance e da un abbassamento dei costi, rendendolo così disponibile anche per scopi commerciali ed hobbyistici. In commercio sono presenti dispositivi di questo tipo aventi pesi molto differenti, da meno di 250g [10] fino a superare gli 11kg [11], così da garantire prestazioni eterogenee in base alle operazioni da svolgere.



Figura 1.1: Drone commerciale DJI Mavic Air 2 [12].

Il controllo di un quadricottero è un problema di interesse a causa della sua complessità: sei gradi di libertà (tre assi di traslazione e tre di rotazione) e solamente quattro input (dipendenti dal numero e dalla velocità delle eliche) rendono il sistema sottoattuato. A causa di questa proprietà vi è un forte accoppiamento fra dinamica traslazionale e rotazionale, di conseguenza aumenta la complessità di task come il mantenimento di uno stato di equilibrio o l'inseguimento di traiettoria. Pertanto la modellazione del velivolo e la definizione delle leggi di controllo rappresentano uno snodo

fondamentale per garantire la corretta esecuzione del compito assegnato in termini di affidabilità e prestazioni.

Per lo sviluppo di un algoritmo di controllo adeguato a quelli che sono i vincoli di progetto è necessario poter simulare il comportamento del velivolo, analizzarne le prestazioni e configurare opportunamente i parametri del controllore prima di implementare lo schema sul controller di volo reale (e quindi nel drone). L'ambiente di sviluppo MATLAB/Simulink [13], [14] consente di simulare il comportamento del velivolo, sviluppare e modificare il controllore in tempi rapidi senza correre il rischio di danneggiare le componenti hardware a fronte di eventuali malfunzionamenti. Il principale svantaggio è quello di operare in un contesto che potrebbe non rispecchiare in maniera sufficientemente accurata le condizioni operative reali del sistema in esame. Queste approssimazioni possono riguardare sia il modello del drone, ad esempio omettendo alcuni disturbi agenti sul velivolo, sia il controller, ignorando limitazioni hardware o eventuali vincoli nella trasmissione di segnali.

Il duplice obiettivo di questo progetto è quello di fornire un framework che consenta di sviluppare un qualsiasi schema di controllo utilizzando delle simulazioni basate su differenti livelli di astrazione hardware e allo stesso tempo implementare, sfruttando il medesimo framework, un algoritmo basato su controllo ottimo che consenta di garantire l'*'hovering'* del quadricottero. Il framework consente di valutare le performance dello schema di controllo eseguito dal *flight controller*, in questo caso Pixracer R15 [15], interfacciato con il modello Simulink del velivolo; così facendo è possibile implementare l'algoritmo direttamente nel controller di volo (che sarà poi montato nel drone reale) e testarne le prestazioni sfruttando i vantaggi del modello simulato. In particolare è possibile definire in maniera del tutto arbitraria il modello dinamico da utilizzare in modo da poter sviluppare modelli più o meno accurati in base alle esigenze di progetto; il controller di volo comunica con il modello Simulink inviando i segnali di controllo come se stesse interagendo con un drone reale.



Figura 1.2: Pixracer R15 [16].

1.1 Stato dell'arte

Il controllo di un quadricottero può avvenire applicando diverse tipologie di tecniche di controllo, in particolare queste possono essere sintetizzate in quattro categorie: controllo lineare, non lineare, basato su intelligenza artificiale e ibrido, ovvero combinando più tecniche.

Uno degli approcci più comunemente usati è quello basato su regolatori PID. In [18] viene proposto un semplice controllore costituito da tre PID in parallelo che consentono di controllare gli angoli di rollio, beccheggio e imbardata¹. In [19] viene proposto uno schema di controllo costituito dalla cascata di regolatori PID per controllare l'orientamento del velivolo più un ulteriore regolatore PID per raggiungere la quota desiderata. La cascata è composta da due regolatori PI in serie: il primo regolatore prende in input l'errore fra l'orientamento desiderato e quello effettivo, la differenza fra

¹Si veda 2.2 per comprendere il significato fisico degli angoli

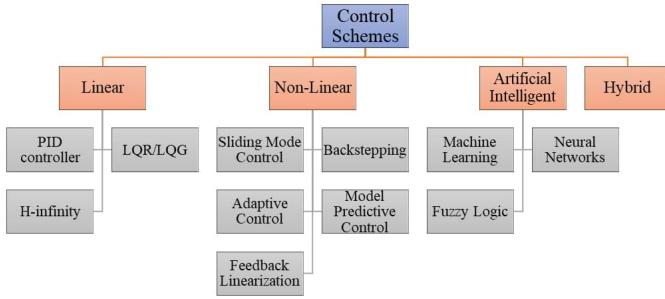


Figura 1.3: Classificazione delle tecniche di controllo [17].

l'output prodotto e la velocità angolare viene quindi utilizzata come segnale di ingresso per il secondo. Il primo controller serve per regolare l'orientamento e il segnale di controllo prodotto è la velocità angolare desiderata, questa serve come riferimento per il secondo regolatore che si occupa della velocità angolare. Le tecniche illustrate finora prevedono quindi che i riferimenti desiderati siano relativi agli angoli *roll*, *pitch*, *yaw* con l'aggiunta della quota *z*, tuttavia è più immediato e intuitivo definire un riferimento nella terna di coordinate spaziali *xyz* assieme all'angolo di *yaw*. L'approccio proposto in [20], sfruttando una sequenza di regolatori PID, consente appunto di specificare i riferimenti *x*, *y*, *z* e l'angolo di imbardata; in particolare il primo regolatore prende in ingresso l'errore fra i valori di riferimento specificati nella terna di coordinate spaziali e i valori misurati, l'uscita di questo primo controller viene utilizzata per generare gli angoli *roll* e *pitch* desiderati. Il secondo regolatore prende in input questi ultimi assieme al valore dello *yaw* desiderato.

Un'altra tecnica di controllo lineare è quella basata su *Linear Quadratic Regulator* (LQR). In [21] e [22] viene proposto un semplice controllo LQR, entrambe le soluzioni si basano su un feedback completo dello stato; [23] propone invece l'utilizzo di LQR per ottimizzare l'operazione di atterraggio del drone Parrot AR.Drone [24]. L'approccio proposto in [25] utilizza un loop esterno per generare gli angoli di riferimento a partire dalla traiettoria da inseguire, i riferimenti angolari desiderati sono controllati dal loop interno; i guadagni dei parametri sono calcolati tramite LQR in modo tale da minimizzare il consumo di energia. In [26] viene proposta un'analisi delle performance di tre tipologie di controllore: PID tarato utilizzando *Integral Time Absolute Error* (ITAE), LQR e PID tarato utilizzando un loop LQR.

Le tecniche analizzate finora si basano su una linearizzazione del modello, tuttavia in letteratura vengono proposti numerosi approcci basati su tecniche di controllo non lineare. In [27] viene proposto un approccio che sfrutta la feedback linearization per effettuare il *tracking* di una semplice traiettoria. Questa tecnica consiste nel trasformare un sistema non lineare in uno completamente o parzialmente lineare tramite un feedback dallo stato e trasformazioni non lineari, in questo modo possono essere applicate strategie di controllo lineare [28]. In [29] viene proposto un approccio che combina la feedback linearization con un controllo LQR: vengono definiti prima i segnali di controllo che consentono di linearizzare la dinamica del sistema e successivamente viene calcolato il guadagno ottimo *K* che consente di stabilizzare il sistema.

Un'altra tecnica impiegata è il controllo adattativo, questa consente di controllare il quadricottero a fronte di incertezze riguardanti i parametri del modello dinamico, come ad esempio il peso oppure eventuali disturbi che possono agire sul velivolo, si pensi a raffiche di vento laterali. In [30] viene proposto un controllo adattativo per la compensazione dei disturbi che agiscono lungo il piano *xy*: il controllo di assetto viene effettuato utilizzando dei regolatori PD, a questi si aggiunge un termine

adattativo con l'obiettivo di compensare le forze incognite agenti sul quadricottero. Le simulazioni mostrano come l'utilizzo dei soli regolatori PD porta ad un errore significativo nell'inseguimento di traiettoria; con l'aggiunta della legge adattativa l'errore di *tracking* viene scalato di un fattore variabile da 3 a 4 rispetto a quello ottenuto con il solo utilizzo dei PD. In [31] viene impiegata una legge adattativa per compensare la variazione di peso dovuta ad operazioni di carico o scarico di oggetti con massa incognita che possono essere trasportati dal velivolo.

Il backstepping è una tecnica di controllo che si basa sull'utilizzo di controller virtuali per la stabilizzazione di un sistema dinamico, in particolare viene utilizzata quando alcuni stati sono controllabili attraverso degli altri; in [32] viene proposto un semplice controllo tramite backstepping. In [33] il sistema viene suddiviso in tre sotto-sistemi interconnessi: il primo modella la relazione fra gli angoli di rollio e beccheggio con le coordinate x , y , il secondo definisce la dinamica della quota z e dell'angolo di imbardata, l'ultimo modella le forze generate dai rotori; l'intero sistema viene quindi stabilizzato tramite backstepping.

Sono presenti inoltre numerosi approcci che utilizzano il machine learning e reti neurali. In [34] vengono impiegati dei regolatori PID in combinazione ad una *Artificial Neural Network* (ANN) composta da un solo *hidden layer*. La rete viene utilizzata per effettuare un aggiornamento online dei guadagni dei regolatori a seguito delle possibili variazioni dei parametri del sistema. Il *continuous tuning* dei guadagni consente di impiegare il regolatore PID per il problema di *tracking* anche per sistemi con parametri variabili nel tempo. Un'altra tecnica per il controllo di droni basata sull'impiego di reti neurali è il reinforcement learning: quest'ultima è una tecnica di machine learning in cui un computer impara a svolgere un'attività tramite l'interazione con l'ambiente, ad ogni azione compiuta viene associata una ricompensa. Il calcolatore impara a prendere decisioni cercando di massimizzare una metrica di *reward*. In [35] viene proposta la stabilizzazione di un quadricottero utilizzando appunto il reinforcement learning.

Capitolo 2

Modellazione e controllo del quadricottero

2.1 Principio di funzionamento

Prima di procedere all'analisi del modello dinamico è bene comprendere il principio di funzionamento che consente al quadricottero di muoversi e orientarsi nello spazio. Il velivolo è costituito da una struttura rigida con configurazione ad 'X' ai cui estremi sono montati quattro rotori controllabili in maniera indipendente, il movimento viene generato da due coppie di eliche che ruotano in direzioni opposte, come illustrato in Figura 2.1. Facendo variare opportunamente la velocità di rotazione delle

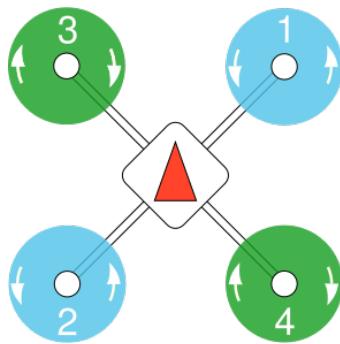


Figura 2.1: Senso di rotazione delle eliche [36].

singole eliche, e di conseguenza la spinta generata dalle stesse, è possibile far muovere il velivolo nello spazio; i movimenti eseguibili sono classificabili in tre tipologie:

- Verticale
- Rotazionale
- Laterale

Nel moto verticale le eliche ruotano tutte alla stessa velocità consentendo al drone di variare la propria quota, in quello rotazionale le coppie di eliche che ruotano in sensi opposti hanno velocità differenti permettendo al velivolo di modificare il proprio orientamento rispetto al suo asse verticale. Il movimento laterale è ottenuto variando la velocità di due eliche su un qualsiasi lato, la spinta creata

sul lato in cui le eliche ruotano più velocemente è maggiore rispetto a quella sul lato opposto. Il risultato è che il quadricottero si muove nella direzione in cui è presente meno spinta [37].

Più in dettaglio, utilizzando come riferimento lo schema in Figura 2.1, ogni elica genera una forza di sollevamento f_i , $i = 1, \dots, 4$ che contribuisce alla spinta totale T , pari quindi alla somma delle singole forze; inoltre la rotazione della singola elica genera una coppia τ_i , $i = 1, \dots, 4$ che, in base alla terza legge di Newton [38], tende a far ruotare il velivolo in direzione opposta a quella di rotazione dell'elica stessa [39]. Nel moto verticale le eliche ruotano tutte alla medesima velocità, così facendo si ottiene:

$$\tau_1 + \tau_2 - \tau_3 - \tau_4 = 0 \quad (2.1)$$

Grazie al bilanciamento delle coppie il drone non ruota attorno al proprio asse verticale; per di più la spinta T dovrà sovrastare la forza gravitazionale agente sul velivolo così da consentire al drone di sollevarsi da terra. Una volta raggiunta la quota di riferimento, per garantire l'*'hovering'* del quadricottero, la forza gravitazionale e la spinta T dovranno compensarsi a vicenda. Dalle considerazioni precedenti è facile intuire che nel caso di moto rotazionale si avrà:

$$\tau_1 + \tau_2 \neq \tau_3 + \tau_4,$$

dipendentemente dal paio di eliche che esercita una coppia maggiore si otterrà una rotazione oraria ($\tau_1 + \tau_2 > \tau_3 + \tau_4$) o antioraria ($\tau_1 + \tau_2 < \tau_3 + \tau_4$). Nel caso di moto laterale una coppia di eliche su un lato ruota a velocità maggiore dell'altra, lo sbilanciamento di forze causa l'inclinazione del drone nel lato in cui la forza complessiva delle due eliche è inferiore. Come mostrato in Figura 2.2 l'inclinazione del drone determina una spinta complessiva (●) scomponibile in una componente verticale ed una orizzontale. Quindi per ottenere un movimento laterale la componente verticale (●) dovrà compensare la forza di gravità così da mantenere la quota, quella orizzontale (●) consente lo spostamento laterale del drone. Le frecce laterali (○) indicano la spinta complessiva generata dalle due coppie di eliche ai lati del quadrirotore.

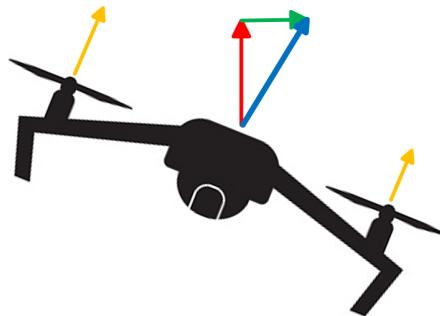


Figura 2.2: Forze generate dal quadricottero durante il moto laterale.

Dalle considerazioni precedenti è possibile intuire il perché della disposizione delle eliche illustrata in Figura 2.1: si consideri ad esempio il caso in cui entrambe le eliche che ruotano in senso orario siano posizionate nella parte anteriore del velivolo e si voglia far muovere in avanti il drone. Le eliche posteriori (ovvero quelle che ruotano in senso antiorario) dovranno generare una spinta complessiva maggiore di quelle anteriori così da far inclinare il drone in avanti, così facendo però l'equazione (2.1) non è più verificata e di conseguenza il velivolo inizierà a ruotare in senso orario compromettendo la stabilità dello stesso.

2.2 Modello dinamico

Per descrivere il modello dinamico vengono definiti due sistemi di riferimento, indicati rispettivamente con le variabili SR_I ed SR_B , espressi entrambi in coordinate *North-East-Down* (NED): il primo è quello inerziale, la cui origine è la posizione di partenza del drone, mentre il secondo è solidale al centro di massa del velivolo. La terna di variabili xyz consente di specificare la posizione dell'origine di SR_B in SR_I , mentre la terna $\varphi\theta\psi$ esprime l'orientamento di SR_B in SR_I tramite la matrice di rotazione

$$\mathbf{R}_{BI} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\varphi - s_\psi c_\varphi & c_\psi s_\theta c_\varphi + s_\psi s_\varphi \\ s_\psi c_\theta & s_\psi s_\theta s_\varphi + c_\psi c_\varphi & s_\psi s_\theta c_\varphi - s_\varphi c_\psi \\ -s_\theta & c_\theta s_\varphi & c_\theta c_\varphi \end{bmatrix}$$

In particolare le variabili φ , θ , ψ indicano gli angoli di rollio (*roll*), beccheggio (*pitch*) e imbardata (*yaw*); questi rappresentano le rotazioni che un veicolo terrestre, marino o aereo può compiere rispettivamente attorno all'asse longitudinale, trasversale e verticale.

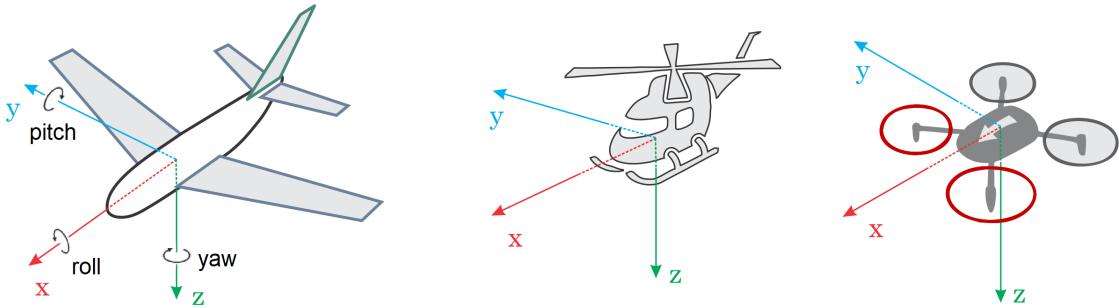


Figura 2.3: Angoli di rollio, beccheggio e imbardata [40].

Dalle precedenti considerazioni, tenendo presente che il quadricottero è di fatto un sistema meccanico, la descrizione completa dello stato avviene in termini di posizione e velocità, in particolare il vettore di stato è il seguente:

$$\begin{bmatrix} x & y & z & \varphi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\varphi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^\top$$

Il modello dinamico utilizzato, proposto in [41], è descritto dal sistema di equazioni (2.2); questo assume che gli effetti giroscopici ed aerodinamici siano irrilevanti, φ e θ siano piccole così che le velocità angolari espresse nei due sistemi di riferimento siano all'incirca uguali, il quadricottero abbia una forma simmetrica e i disturbi siano trascurabili.

$$\begin{cases} \ddot{x} = -(\cos(\psi) \sin(\theta) \cos(\varphi) + \sin(\psi) \sin(\varphi)) \frac{T}{m} \\ \ddot{y} = -(\sin(\psi) \sin(\theta) \cos(\varphi) - \sin(\varphi) \cos(\psi)) \frac{T}{m} \\ \ddot{z} = -\cos(\theta) \cos(\varphi) \frac{T}{m} + g \\ \ddot{\varphi} = \frac{\tau_\varphi}{I_x} \\ \ddot{\theta} = \frac{\tau_\theta}{I_y} \\ \ddot{\psi} = \frac{\tau_\psi}{I_z} \end{cases} \quad (2.2)$$

I parametri I_x , I_y , I_z rappresentano i momenti di inerzia per ciascun asse, m è la massa del quadricottero e g la costante di accelerazione gravitazionale. Gli ingressi che consentono di controllare la

spinta verticale e gli angoli RPY sono indicati con T , τ_φ , τ_θ , τ_ψ . Utilizzando la medesima convenzione impiegata in Figura 2.1, ed indicando con l la distanza fra il centro del velivolo e l'elica, si ottengono quindi le equazioni dei segnali di controllo in funzione delle forze e coppie esercitate dalle eliche:

$$T = f_1 + f_2 + f_3 + f_4 \quad (2.3)$$

$$\tau_\varphi = l \sin\left(\frac{\pi}{4}\right) (f_1 - f_2 - f_3 + f_4) \quad (2.4)$$

$$\tau_\theta = l \sin\left(\frac{\pi}{4}\right) (f_1 - f_2 + f_3 - f_4) \quad (2.5)$$

$$\tau_\psi = (\tau_1 + \tau_2 - \tau_3 - \tau_4) \quad (2.6)$$

Dalle equazioni (2.3) - (2.6) è possibile ricavare il contributo di ogni singolo rotore, e successivamente la velocità angolare di riferimento ω_i , $i = 1, \dots, 4$ grazie alle relazioni:

$$f_i = b\omega_i^2$$

$$\tau_i = d\omega_i^2$$

dove b e d sono dei coefficienti aerodinamici dipendenti dalla geometria delle eliche e denominati rispettivamente *thrust factor* e *drag factor*.

2.2.1 Modello linearizzato

Come si può osservare in (2.2) il modello presenta una dinamica non lineare, tuttavia per lo sviluppo del controllore desiderato è necessario che la dinamica del sistema sia lineare.

La linearizzazione è una tecnica che consente di approssimare una funzione non lineare con una funzione lineare nell'intorno di un qualche punto di lavoro; in prossimità del punto le due funzioni hanno un andamento simile (quindi la funzione lineare è una buona approssimazione), allontanandosi progressivamente dal punto scelto la qualità dell'approssimazione peggiora. Un esempio pratico è proposto in [42], la Figura 2.4 evidenzia come la linearizzazione $y = 2x - 1$ approssima correttamente l'andamento della funzione $y = x^2$ nell'intorno del punto $\bar{x} = 1$.

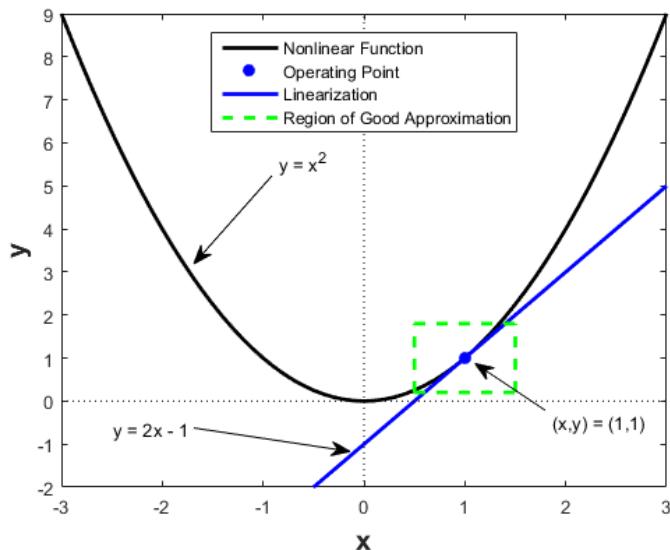


Figura 2.4: Linearizzazione della funzione $y = x^2$ [42].

Il concetto di linearizzazione può essere esteso ad un sistema dinamico tempo invariante del tipo $\dot{x} = f(x, u)$, dove x rappresenta lo stato del sistema e u i segnali di input. Linearizzando nell'intorno di un generico punto di equilibrio \bar{x} e definendo $\delta_x = x - \bar{x}$ e $\delta_u = u - \bar{u}$, dove \bar{u} indica il segnale di controllo all'equilibrio, si ottiene un sistema del tipo

$$\begin{aligned}\dot{\delta}_x &= A\delta_x + B\delta_u \\ A &= \left. \frac{\partial f(x, u)}{\partial x} \right|_{(x, u) = (\bar{x}, \bar{u})} \quad B = \left. \frac{\partial f(x, u)}{\partial u} \right|_{(x, u) = (\bar{x}, \bar{u})}\end{aligned}\quad (2.7)$$

Il modello linearizzato descrive il comportamento del sistema non lineare nell'intorno del punto di equilibrio; l'approssimazione consente di concludere sulla natura di un punto di equilibrio del sistema non lineare basandosi sulla posizione degli autovalori, semplificando notevolmente l'analisi degli equilibri che altrimenti andrebbe condotta tramite la definizione di una funzione di Lyapunov. Progettando un'opportuna legge di controllo che stabilizzi il sistema lineare è possibile stabilizzare il punto di equilibrio del sistema non lineare.

L'idea è quindi quella di linearizzare il sistema (2.2), prima di tutto però deve essere definita la coppia (\bar{x}, \bar{u}) che garantisce l'*hovering* del velivolo ad una quota arbitraria d (ricordando che la convenzione utilizzata per il modello è NED). In particolare si ha:

$$\bar{x} = [0 \ 0 \ -d \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \quad (2.8)$$

$$\bar{u} = [T \ \tau_\varphi \ \tau_\theta \ \tau_\psi] = [mg \ 0 \ 0 \ 0] \quad (2.9)$$

e successivamente applicando la linearizzazione illustrata in (2.7) si ottiene il sistema lineare tempo invariante (LTI - *Linear Time Invariant*):

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}$$

La matrice D è identicamente nulla mentre invece la scelta di C dipende dalle assunzioni sulle variabili misurabili: supponendo ad esempio che si possa accedere all'intero vettore di stato questa sarà una matrice identità di dimensione 12×12 ; se invece si assume di poter misurare solo la posizione del

velivolo nello spazio la struttura sarà

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.3 Sviluppo del controllore

2.3.1 Osservatore asintotico dello stato

Considerando un sistema LTI può accadere di non avere accesso all'intero vettore di stato x ; ha senso dunque chiedersi se, dato un sistema dinamico, è possibile ricostruire lo stato completo a partire dalla conoscenza della variabili di ingresso e uscita. L'osservatore asintotico dello stato consente di generare una stima dello stato, indicata con \hat{x} , a partire dalla conoscenza dei segnali di controllo u e di uscita y . Il modello che descrive l'osservatore è:

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \\ \hat{y} = C\hat{x} + Du \end{cases} \quad (2.10)$$

dove le matrici A, B, C, D definiscono la dinamica del sistema mentre il termine $L(y - \hat{y})$, denominato innovazione, misura lo scostamento tra il valore vero dell'uscita e quella stimata se lo stato reale fosse \hat{x} . Definendo l'errore di stima $\tilde{x} = x - \hat{x}$ si ottiene la dinamica dell'errore:

$$\dot{\tilde{x}} = \dot{x} - \dot{\hat{x}} = Ax + Bu - A\hat{x} - B\hat{u} - L(Cx - C\hat{x}) = (A - LC)\tilde{x} \quad (2.11)$$

La dinamica dell'errore di stima dipende quindi dal termine $(A - LC)$, con L che è un parametro di progetto. Definendo quest'ultimo in modo tale che $\sigma(A - LC) \subset \mathbb{C}^-$ si ottiene una dinamica dell'errore asintoticamente stabile e quindi convergente a zero, di conseguenza \hat{x} converge ad x .

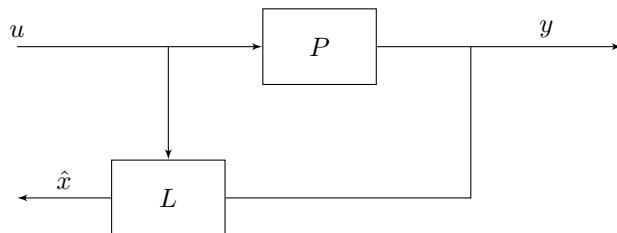


Figura 2.5: Schema con osservatore per la stima dello stato.

2.3.2 Controllo LQR

Nell'ambito dei controlli automatici si definisce controllo ottimo l'insieme di algoritmi che hanno l'obiettivo di stabilizzare un sistema dinamico minimizzando un certo funzionale di costo. In particolare, sotto le ipotesi di sistema con dinamica lineare e funzionale di costo dipendente quadraticamente dallo stato e dall'ingresso, il problema LQR consiste nel determinare il segnale di controllo $u : [0, t_f] \rightarrow \mathbb{R}^m$

che minimizza il funzionale di costo J definito come:

$$\begin{aligned} J &= \int_0^{t_f} (x(s)^\top Q x(s) + u(s)^\top R u(s)) ds + x(t_f)^\top S x(t_f) \\ \text{s.t. } & \dot{x}(t) = Ax(t) + Bu(t), \forall t \in [0, t_f] \\ & x(0) = x_0 \end{aligned}$$

dove $Q \succeq 0$ è il costo dello stato, $R \succeq 0$ il costo del controllo ed $S \succeq 0$ il costo terminale. La soluzione al problema LQR è data dalla legge di controllo in retroazione

$$u(t, x) = \underbrace{-R^{-1}B^\top P^*(t)x}_K$$

dove P^* è la soluzione dell'equazione differenziale di Riccati (DRE - *Differential Riccati Equation*) definita come:

$$\begin{aligned} -\dot{P} &= A^\top P + PA + Q - PBR^{-1}B^\top P \\ P(t_f) &= S, \quad t \in [t_0, t_f] \end{aligned}$$

Il problema LQR può essere definito anche su un orizzonte temporale infinito in cui non esiste un tempo finale t_f ; l'obiettivo è quello di determinare il segnale di controllo $u : [0, \infty] \rightarrow \mathbb{R}^m$ che minimizza il funzionale di costo J definito come:

$$\begin{aligned} J &= \int_0^\infty (x(s)^\top Q x(s) + u(s)^\top R u(s)) ds \\ \text{s.t. } & \dot{x}(t) = Ax(t) + Bu(t), \forall t \geq 0 \\ & x(0) = x_0 \end{aligned}$$

Sotto le ipotesi di controllabilità della coppia (A, B) e di osservabilità della coppia (A, E) , con $Q = E^\top E$, il problema LQR su orizzonte infinito ammette soluzione. In particolare la legge di controllo in retroazione

$$u = \underbrace{-R^{-1}B^\top P^*x}_K$$

garantisce che il sistema a ciclo chiuso $\dot{x} = (A - BK)x$ sia asintoticamente stabile. P^* rappresenta la soluzione dell'equazione algebrica di Riccati (ARE - *Algebraic Riccati Equation*) definita come:

$$0 = A^\top P + PA + Q - PBR^{-1}B^\top P$$

2.3.3 State feedback

La prima tecnica di controllo analizzata si basa sull'utilizzo di un regolatore lineare quadratico tramite retroazione dallo stato per garantire, come anticipato in 2.2.1, l'*hovering* del quadricottero ad una quota arbitraria. Considerando quindi la coppia (\bar{x}, \bar{u}) definita in (2.8) - (2.9) si vuole che tramite un'opportuna legge di controllo u il sistema sia stabilizzato alla quota assegnata, cioè $f(\bar{x}, \bar{u}) = 0$.

Nel caso di feedback completo dallo stato il sistema dinamico è:

$$\begin{cases} \dot{x} = f(x, u) \\ y = x \end{cases}$$

Assumendo che

$$\exists! (\bar{x}, \bar{u}) \mid A\bar{x} + B\bar{u} = 0$$

e considerando il sistema linearizzato nell'intorno di \bar{x} si definiscono le variabili $\tilde{x} = x - \bar{x}$ e $\tilde{u} = u - \bar{u}$ tali per cui

$$\begin{aligned} & \begin{cases} \dot{\tilde{x}} = \dot{x} = Ax + Bu - \underbrace{A\bar{x} - B\bar{u}}_0 = A\tilde{x} + B\tilde{u} \\ \tilde{u} = K\tilde{x} \end{cases} \\ & \Rightarrow \dot{\tilde{x}} = (A + BK)\tilde{x} \\ & \Rightarrow u = K\tilde{x} + \bar{u} \end{aligned}$$

La dinamica di \tilde{x} è determinata quindi da $(A + BK)$, scegliendo il parametro K tramite LQR è possibile rendere la dinamica asintoticamente stabile e quindi $x \rightarrow \bar{x}$. Analizzando la legge di controllo $u = K\tilde{x} + \bar{u}$ questa presenta un termine in retroazione Kx e un termine di feedforward \bar{u} , illustrato in (2.9).

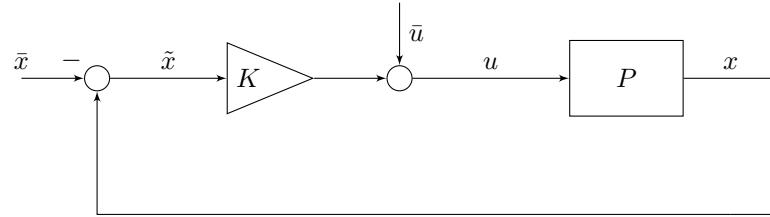


Figura 2.6: Schema di controllo State feedback.

Considerando l'equazione dell'accelerazione \ddot{z} descritta in (2.2), assumendo $\theta = 0$, $\varphi = 0$, e sostituendo T con la rispettiva componente del segnale di controllo u si ottiene:

$$\ddot{z} = -\frac{K(x - \bar{x}) + mg}{m} + g \Big|_{\tilde{x}=0} = -\frac{mg}{m} + g = 0$$

2.3.4 Output feedback

Lo schema di controllo illustrato in 2.3.3 costringe a specificare la traiettoria da inseguire con un numero di componenti pari all'intero vettore di stato, tuttavia nel contesto in esame ha senso specificare un riferimento in termini di posizione e orientamento del velivolo. In questo caso il feedback non è dallo stato ma dall'uscita, si ha quindi $y = Cx$ con

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.12)$$

L'idea rimane quella di applicare la legge di controllo del tipo $u = K\tilde{x} + \bar{u}$ per l'inseguimento del riferimento

$$r = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^\top = \begin{bmatrix} 0 & 0 & -d & 0 \end{bmatrix}^\top \quad (2.13)$$

In questo caso si assume di non poter misurare l'intero vettore di stato e di conseguenza non è possibile ottenere direttamente $\tilde{x} = x - \bar{x}$, viene quindi costruita una stima $\hat{\tilde{x}}$ dell'errore e la legge di controllo sarà

$$u = K\hat{\tilde{x}} + \bar{u}$$

Applicando lo stesso procedimento utilizzato in 2.3.3 si assume che:

$$\exists! (\bar{x}, \bar{u}) \mid A\bar{x} + B\bar{u} = 0$$

tale per cui:

$$\dot{\tilde{x}} = Ax + Bu - \underbrace{A\bar{x} - B\bar{u}}_0 = A\tilde{x} + B(u - \bar{u}) \quad (2.14)$$

Definendo $r = C\bar{x}$ e l'errore di *tracking* come $e = y - r$, considerando inoltre che $x = \tilde{x} + \bar{x}$ si ottiene:

$$\begin{aligned} y &= Cx = C(\tilde{x} + \bar{x}) = C\tilde{x} + C\bar{x} \\ e &= y - r = Cx - C\bar{x} = C\tilde{x} \end{aligned} \quad (2.15)$$

L'obiettivo rimane quello di stimare l'errore \tilde{x} , perciò considerando il modello dell'osservatore descritto in (2.10) e le equazioni (2.14), (2.15)

$$\begin{cases} \dot{\tilde{x}} = A\tilde{x} + B(u - \bar{u}) \\ u = K\hat{\tilde{x}} + \bar{u} \\ e = C\tilde{x} \end{cases}$$

si ottiene la dinamica della stima di \tilde{x} :

$$\begin{aligned} \dot{\hat{\tilde{x}}} &= A\hat{\tilde{x}} + B(u - \bar{u}) + L(e - C\hat{\tilde{x}}) = A\hat{\tilde{x}} + BK\hat{\tilde{x}} + L(e - C\hat{\tilde{x}}) = \\ &= (A + BK - LC)\hat{\tilde{x}} + Le \end{aligned} \quad (2.16)$$

Quindi l'osservatore prende in ingresso solamente l'errore di *tracking* e , definendo inoltre $\varepsilon = \tilde{x} - \hat{\tilde{x}}$, cioè la differenza fra l'errore vero e quello stimato si ricava:

$$\begin{aligned} \dot{\varepsilon} &= \dot{\tilde{x}} - \dot{\hat{\tilde{x}}} = A\tilde{x} + \cancel{B(u - \bar{u})} - A\hat{\tilde{x}} - \cancel{B(u - \bar{u})} - L(e - C\hat{\tilde{x}}) = \\ &= A(\tilde{x} - \hat{\tilde{x}}) - L(C\tilde{x} - C\hat{\tilde{x}}) = A\varepsilon - LC\varepsilon = (A - LC)\varepsilon \end{aligned} \quad (2.17)$$

Si ottiene quindi che la dinamica di ε dipende dal termine $(A - LC)$, è sufficiente quindi scegliere L tramite LQR così da avere che $\sigma(A - LC) \subset \mathbb{C}^-$. Avendo definito ε è possibile riscrivere l'equazione (2.16) come:

$$\begin{aligned} \dot{\hat{\tilde{x}}} &= A\hat{\tilde{x}} + BK\hat{\tilde{x}} + L(e - C\hat{\tilde{x}}) = A\hat{\tilde{x}} + BK\hat{\tilde{x}} + L(C\tilde{x} - C\hat{\tilde{x}}) = \\ &= A\hat{\tilde{x}} + BK\hat{\tilde{x}} + LC\varepsilon \end{aligned} \quad (2.18)$$

Definendo la coppia $(\hat{\tilde{x}}, \varepsilon)$ e considerando (2.17), (2.18) si ottiene:

$$\begin{bmatrix} \dot{\hat{\tilde{x}}} \\ \dot{\varepsilon} \end{bmatrix} = \begin{bmatrix} A + BK & LC \\ 0 & A - LC \end{bmatrix} \begin{bmatrix} \hat{\tilde{x}} \\ \varepsilon \end{bmatrix}$$

Gli autovalori del sistema sono l'unione degli autovalori di $(A + BK)$ e di $(A - LC)$, è possibile quindi progettare K ed L indipendentemente così da garantire che:

$$\begin{aligned} \sigma(A + BK) &\subset \mathbb{C}^- \\ \sigma(A - LC) &\subset \mathbb{C}^- \end{aligned}$$

e quindi che il sistema $(\hat{\tilde{x}}, \varepsilon)$ sia asintoticamente stabile. Di conseguenza si ha:

$$\begin{aligned} \varepsilon \rightarrow 0 &\Rightarrow \hat{\tilde{x}} \rightarrow \tilde{x} \\ \hat{\tilde{x}} \rightarrow 0 &\Rightarrow \tilde{x} \rightarrow 0 \Rightarrow x \rightarrow \bar{x} \end{aligned}$$

La legge di controllo è quindi $u = K\hat{\tilde{x}} + \bar{u}$ con il medesimo termine di feedforward utilizzato in 2.3.3 e indicato in (2.9).

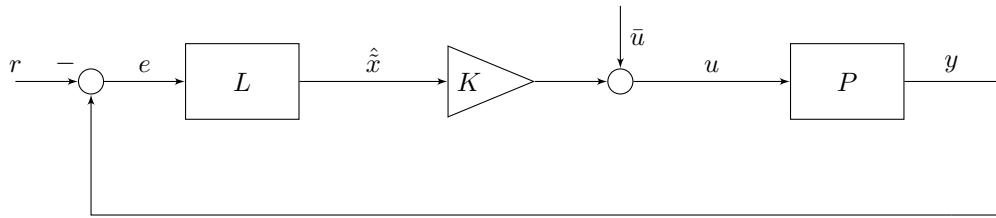


Figura 2.7: Schema di controllo Output feedback.

2.3.5 State feedback con termine integrale

Gli schemi di controllo illustrati in 2.3.3 e 2.3.4 presuppongono una conoscenza esatta dei parametri del modello, in particolare il termine di feedforwad espresso in (2.9) necessita di una conoscenza esatta della massa m . Un'ipotesi di questo tipo è tuttavia molto limitante: potrebbe non essere possibile conoscere con esattezza il peso del velivolo, oppure lo stesso peso potrebbe cambiare dinamicamente nel caso in cui il drone debba trasportare dei carichi durante una missione; viene quindi definita una nuova legge di controllo indipendente dalla massa m . Considerando il sistema linearizzato come mostrato in 2.3.3, assumendo di poter accedere all'intero vettore di stato e specificando lo stesso riferimento r definito in (2.13) viene definita la legge di controllo:

$$\begin{cases} u = K_P x + K_I \sigma \\ \dot{\sigma} = r - y = r - Cx \end{cases}$$

con C espressa in (2.12). Si ottiene quindi la dinamica del sistema:

$$\dot{x} = (A + BK_P)x + BK_I \sigma$$

e applicando ancora una volta il medesimo procedimento utilizzato in 2.3.3 e 2.3.4 si assume che:

$$\exists! (\bar{x}, \bar{\sigma}) \mid A\bar{x} + B\underbrace{(K_P\bar{x} + BK_I\bar{\sigma})}_{\bar{u}} = 0, \quad C\bar{x} = r$$

definendo $\tilde{x} = x - \bar{x}$ e $\tilde{\sigma} = \sigma - \bar{\sigma}$ ed esplicando la dinamica della coppia $(\tilde{x}, \tilde{\sigma})$ si ottiene:

$$\begin{cases} \dot{\tilde{x}} = \dot{x} = Ax + B(K_Px + K_I\sigma) - \underbrace{A\bar{x} + BK_P\bar{x} + BK_I\bar{\sigma}}_0 = (A + BK_P)\tilde{x} + BK_I\tilde{\sigma} \\ \dot{\tilde{\sigma}} = \dot{\sigma} - \dot{\bar{\sigma}} = r - Cx - \underbrace{r - C\bar{x}}_0 = -C(x - \bar{x}) = -C\tilde{x} \end{cases}$$

scrivibile in forma matriciale come:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{\sigma}} \end{bmatrix} = \begin{bmatrix} A + BK_P & BK_I \\ -C & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{\sigma} \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}}_{A_e} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_e} \underbrace{\begin{bmatrix} K_P & K_I \\ 0 & K_e \end{bmatrix}}_{K_e} \begin{bmatrix} \tilde{x} \\ \tilde{\sigma} \end{bmatrix}$$

Di conseguenza basta ricavare la matrice K_e tramite LQR dal sistema esteso (A_e, B_e) in modo tale da rendere asintoticamente stabile il sistema $(\tilde{x}, \tilde{\sigma})$. Così facendo si ottiene quindi:

$$\begin{aligned} \tilde{x} \rightarrow 0 &\Rightarrow x \rightarrow \bar{x} \\ \tilde{\sigma} \rightarrow 0 &\Rightarrow \sigma \rightarrow \bar{\sigma} \end{aligned}$$

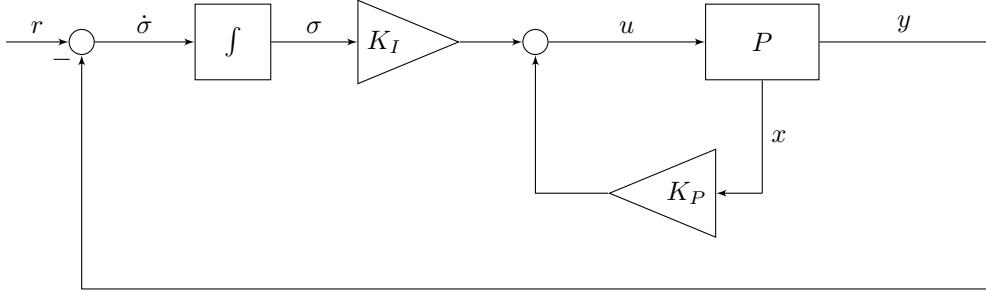


Figura 2.8: Schema di controllo State feedback con termine integrale

Capitolo 3

Autopilota PX4

3.1 Concetti generali

PX4 [43] è un *autopilot flight stack*, ovvero un software di controllo del volo per droni; tuttavia può essere impiegato per controllare veicoli autonomi di varia natura: aerei, sottomarini o di superficie [44]. Consente di determinare, tramite l'utilizzo di sensori, lo stato² del veicolo così da permettere allo stesso di navigare in maniera autonoma. I segnali di output generati dall'autopilota vengono utilizzati per controllare la velocità dei motori tramite un *Electronic Speed Controller* (ESC), quest'ultimo converte i segnali generati dal controller di volo in appropriati livelli di potenza per i motori. Possono inoltre essere generati anche dei segnali che consentano di controllare altre periferiche esterne come fotocamera o paracadute.



Figura 3.1: Logo PX4 [43].

PX4 definisce diverse modalità di volo [45] che determinano come lo stesso autopilota risponde ai comandi impartiti da remoto e come gestisce il movimento del velivolo durante il volo in maniera autonoma. Le modalità forniscono differenti livelli di assistenza da parte dell'autopilota al pilota che variano dall'automazione di alcuni task comuni, come il decollo e l'atterraggio fino a meccanismi che facilitano il mantenimento del veicolo in una certa posizione. Le modalità di volo possono essere catalogate in due categorie: manuali e autonome. In modalità manuale l'utente ha il controllo del velivolo tramite un radiocomando, in modalità autonoma è l'autopilota ad avere il controllo e il radiocomando viene utilizzato solamente per modificare la modalità di volo.

Il *flight stack* può essere eseguito su diverse tipologie di controller di volo [46], quest'ultimo è la componente hardware che costituisce il “cervello” del veicolo. La piattaforma hardware di riferimento per lo stack software PX4 è Pixhawk [47], questa comprende diversi tipologie di controller con differenti caratteristiche in termini di costo, dimensioni e performance. Il controller comprende un set minimo di sensori costituito da giroscopio, accelerometro, bussola e barometro grazie ai quali il software di

²Il concetto di stato varia dipendentemente dal task in esame, può quindi intendersi come stato un insieme di variabili tra le quali: posizione, orientamento, velocità lineare, velocità angolare, stato della batteria, etc.



Figura 3.2: Controller di volo Pixhawk 4 [48], Pixhawk 4 Mini [49], Pixhawk 3 Pro [50].

controllo del volo riesce a determinare lo stato del veicolo; possono essere collegati anche altri sensori addizionali come ad esempio il GPS [51].

Nel velivolo sono chiaramente montate le eliche che possono essere pericolose quando ruotano, per questo motivo è utile definire due stati mutuamente esclusivi in cui il drone può trovarsi: disarmato e armato. Il primo indica che gli attuatori non sono alimentati e quindi il drone non è in grado di volare, nel secondo caso il velivolo è pronto per prendere il volo e quindi gli attuatori sono alimentati.

3.1.1 QGroundControl

QGroundControl (QGC) [52] è una *Ground Control Station* (GCS), ovvero un'applicazione software eseguita su un computer a terra che consente di comunicare con il velivolo. Permette di configurare e modificare i parametri dell'autopilota, pianificare missioni, arfare e disarmare il drone, inviare i comandi, ottenere e visualizzare informazioni real-time sullo stato del velivolo, calibrare i sensori e aggiornare il firmware del controller di volo.

La schermata principale, denominata *Fly View*, viene utilizzata per comandare e monitorare il veicolo mentre è in volo, la schermata *Setup View* consente di configurare un nuovo velivolo effettuando l'aggiornamento del firmware e le calibrazioni dei sensori, del radiocomando e dei parametri dell'autopilota.

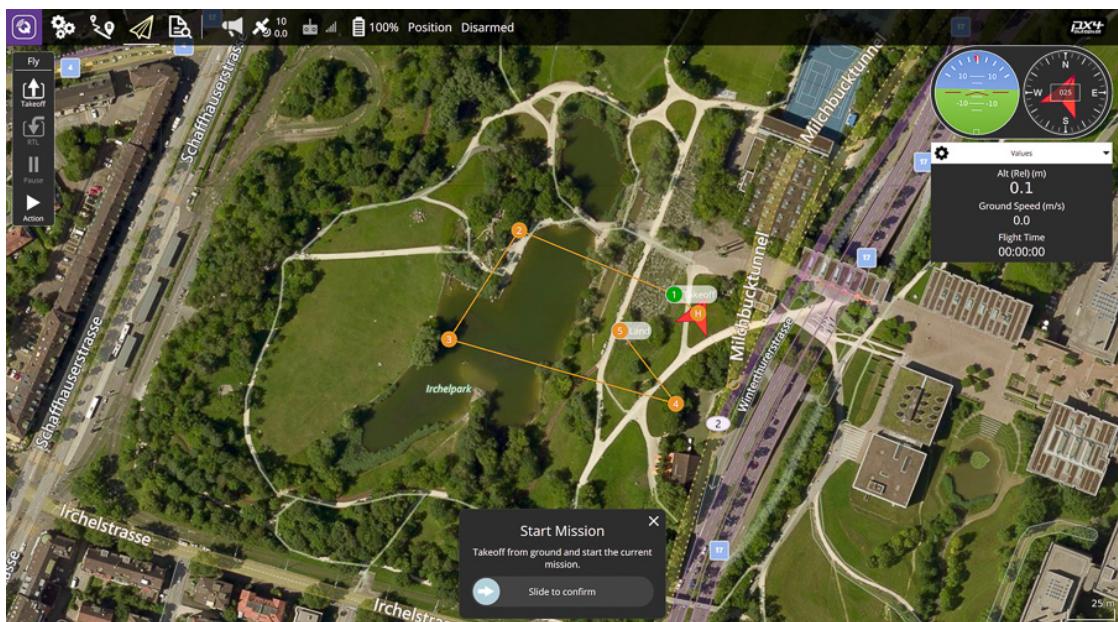


Figura 3.3: Schermata Fly View QGroundControl [53].

3.1.2 MAVLink

MAVLink [54] è un protocollo di comunicazione utilizzato per comunicare con il drone, PX4 utilizza quest'ultimo per scambiare informazioni con la GCS e connettere le componenti del drone esterne al controller di volo [55]. Una rete MAVLink si compone di sistemi (veicoli, GCS, etc.) che a loro volta possono essere composti da più componenti. Ogni sistema ha un identificativo univoco (SYSTEM-ID) così come ogni componente di uno specifico sistema (COMPONENT-ID); questa coppia di campi consente di indirizzare le informazioni a tutti i sistemi, ad un sistema specifico, a tutte le componenti di un sistema o ad una particolare componente.



Figura 3.4: Logo MAVLink [54].

I flussi di dati vengono inviati tramite un meccanismo di comunicazione *publish/subscribe*: il mittente (*publisher*) pubblica i messaggi contenenti le informazioni in un *topic* a cui si sottoscrivono i destinatari (*subscriber*) per ottenere l'informazione pubblicata. Inoltre viene utilizzata una comunicazione di tipo *point-to-point* per lo scambio delle informazioni di configurazione. I messaggi sono definiti in file .XML, ogni file definisce il set di messaggi, detto *dialect*, supportato da un particolare sistema; il set di messaggi implementato nella maggior parte delle GCS è definito nel set `common.xml`.

3.2 Architettura software

Lo stack architettonico prevede la presenza di un controller di volo (sui cui viene eseguito l'autopilota PX4), una GCS, un radiocomando e chiaramente un velivolo equipaggiato opportunamente con motori, sensori e altre periferiche opzionali.

L'autopilota PX4 consiste in due layer principali: il *flight stack* si occupa di controllare il velivolo e il *middleware* ha il compito di gestire la comunicazione con l'hardware. Il sistema è progettato in modo tale che sia reattivo [56], questo significa che:

- tutte le funzionalità vengono suddivise in componenti scambiabili e riutilizzabili;
- la comunicazione viene effettuata tramite uno scambio di messaggi asincrono;
- il sistema riesce a gestire carichi di lavoro variabile [57].

Il diagramma in Figura 3.5 illustra i moduli che costituiscono il software PX4³. Le componenti più in alto contengono i moduli del middleware, questo si compone infatti dei driver per la comunicazione con i sensori e con gli altri dispositivi. Lo scambio di informazioni fra i vari moduli avviene tramite un bus denominato *uORB* che sfrutta un meccanismo di comunicazione *publish/subscribe*, il cui funzionamento è stato illustrato in 3.1.2. La parte in basso del diagramma è il *flight stack* contenente gli algoritmi di stima e controllo necessari per la navigazione.

³Una rappresentazione dettagliata della rete di comunicazione fra i moduli è consultabile in [58].

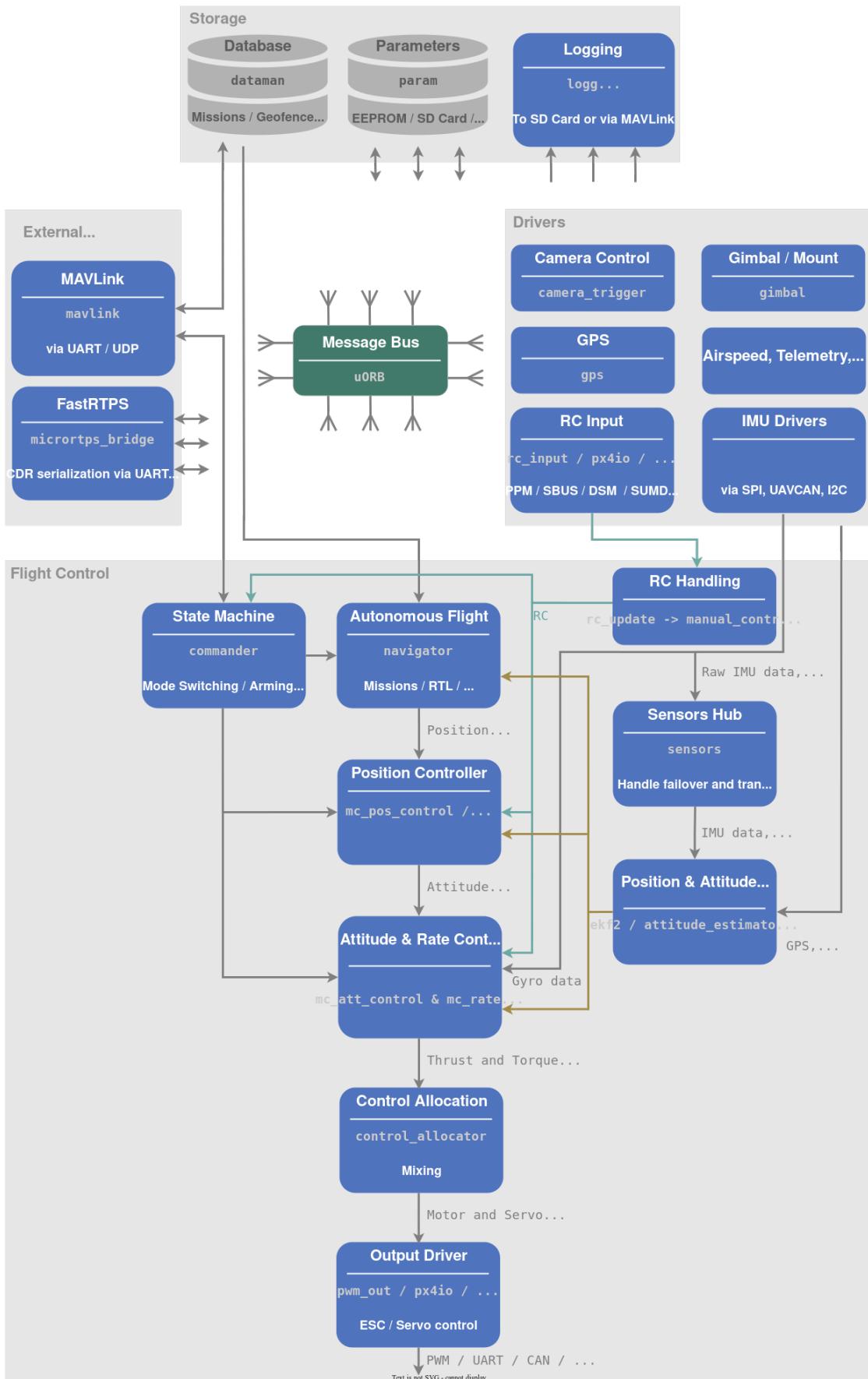


Figura 3.5: Architettura software PX4 [57].

3.2.1 Flight Stack

Il flight stack è una collezione di algoritmi di guida, navigazione e controllo per droni autonomi [57]. Lo stimatore si occupa di calcolare grazie ad EKF [59] lo stato del veicolo basandosi sulle misure fornite dai sensori. Il controller prende in ingresso lo stato stimato assieme ad un setpoint cercando di far raggiungere allo stato il setpoint ricevuto in input, ha quindi l'obiettivo di impostare al sistema il comportamento desiderato tramite il proprio output. Il mixer prende in ingresso i segnali in uscita dal controllore convertendoli nei comandi per ogni singolo motore garantendo che questi siano coerenti con i limiti fisici degli attuatori.

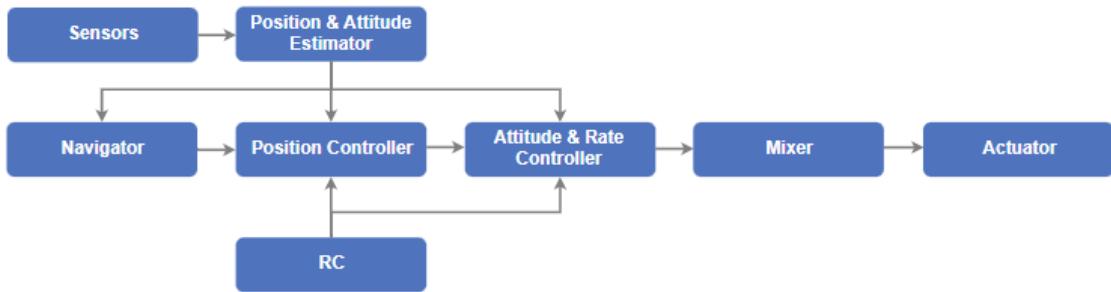


Figura 3.6: Flight stack PX4 [57].

Controller

Il controller impiegato nello stack PX4 è formato da una cascata di regolatori PID [60]. I regolatori di posizione e orientamento sono dei P, mentre invece quelli di velocità lineare e angolare sono dei PID; lo schema complessivo è illustrato in Figura 3.7. Il blocco denominato Acceleration and Yaw to Attitude prende in ingresso il setpoint, espresso in termini di accelerazioni lineari A_{sp} e angolo di yaw ψ_{sp} , producendo in output la spinta verticale $\delta_{T_{sp}}$ ed una rappresentazione dell'orientamento mediante quaternione q_{sp} . Il primo regolatore prende in ingresso il setpoint di posizione X_{sp} e la stima

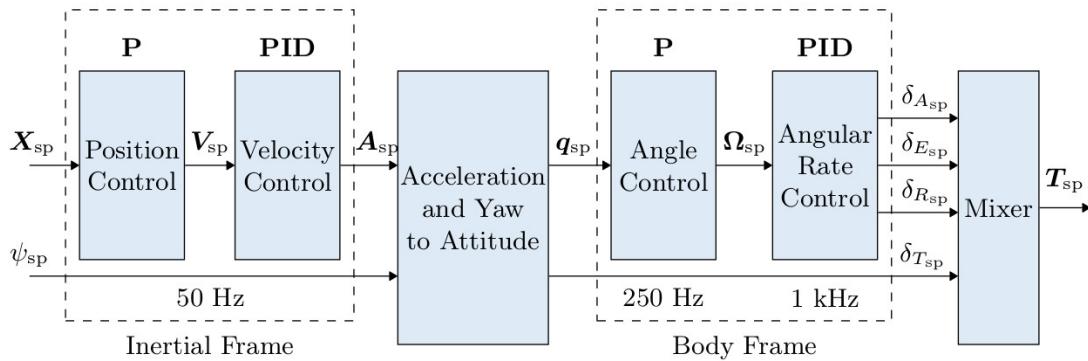


Figura 3.7: Cascata di regolatori PID controller PX4 [60].

della stessa fornita da EKF producendo in uscita il riferimento in velocità V_{sp} ; il controller di velocità seguendo uno schema identico al precedente prende in ingresso la stima della velocità e il riferimento V_{sp} generato dal controller di posizione restituendo in uscita il riferimento in accelerazione. La seconda coppia di regolatori ha una struttura identica alla prima con la differenza che i riferimenti

sono espressi in termini di posizione angolare q_{sp} e velocità angolare Ω_{sp} . In output vengono generati i valori di roll $\delta_{A_{sp}}$, pitch $\delta_{E_{sp}}$ e yaw $\delta_{R_{sp}}$ ⁴ in ingresso al mixer assieme alla spinta $\delta_{T_{sp}}$.

3.3 SITL vs HITL

Per testare le performance di un algoritmo di controllo possono essere utilizzate varie tipologie di simulazione prima di procedere all'impiego del controller di volo sul velivolo reale; il processo di simulazione permette difatti l'interazione fra il modello simulato e il software di controllo, quest'ultimo viene quindi sollecitato come se fosse in condizioni operative reali. Le modifiche apportate possono essere testate in maniera più rapida e sicura prima di volare utilizzando un drone reale. In particolare si distinguono due tipologie di simulazioni: Software In The Loop (SITL) e Hardware In The Loop (HITL).

SITL indica una tecnica di verifica che sfrutta la completa emulazione sia del controllore che del modello da controllare, così facendo è possibile verificare la corretta implementazione del controllore da un punto di vista logico e avere una prima idea dei risultati ottenibili. Il vantaggio risiede nella rapidità con cui è possibile applicare e testare le modifiche effettuate, oltre a non richiedere alcuna disponibilità fisica di componenti hardware o prototipi.

Nella simulazione HITL è previsto l'utilizzo di un controllore reale che comunica con il modello da controllare simulato; pertanto è possibile non solo testare le performance dell'algoritmo di controllo ma anche l'intero stack software implementato dal controller di volo e le interfacce di comunicazione dello stesso. L'algoritmo eseguito in hardware consente di verificare le prestazioni in condizioni più realistiche rispetto a SITL tenendo conto dei vincoli prestazionali del dispositivo impiegato. Il limite di questo approccio è determinato dalla necessità di avere a disposizione fisicamente il controller di volo. Utilizzando questa tecnica i sensori del controller vengono disabilitati così da non avere conflittualità con i dati inviati dal modello.

PX4 supporta sia la simulazione Software In The Loop, in cui lo stack di volo viene eseguito su computer (lo stesso computer o un altro computer sulla stessa rete), sia la simulazione Hardware In The Loop utilizzando un firmware di simulazione su un controller di volo reale [62]. I simulatori supportati da PX4 sono elencati in [62], tuttavia per lo sviluppo del progetto è stata utilizzata la piattaforma MATLAB/Simulink come illustrato in 3.4.

In Figura 3.8 è mostrato lo schema della simulazione HITL: il controller di volo invia i segnali al modello simulato in MATLAB/Simulink che a sua volta aggiorna lo stato dello stesso in base all'input ricevuto; successivamente lo stato del sistema aggiornato entra in input al controllore.

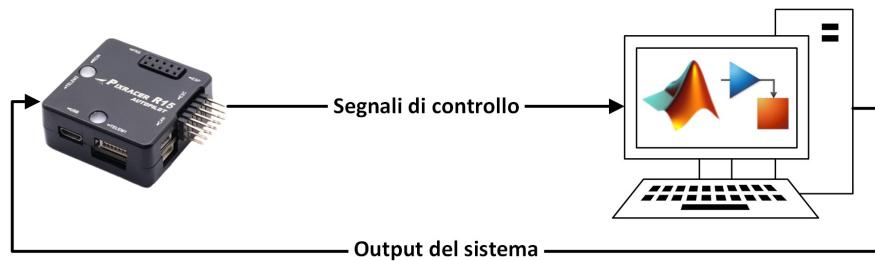


Figura 3.8: Schema simulazione HITL.

⁴Le variabili roll, pitch, yaw vengono rispettivamente indicate anche con i termini *aileron*, *elevator*, *rudder* come mostrato in [61].

3.4 UAV Toolbox Support Package for PX4 Autopilots

Mathworks mette a disposizione il *toolbox* denominato *UAV Toolbox Support Package for PX4 Autopilots* [63] per lo sviluppo di algoritmi di controllo compatibili con lo stack PX4; in particolare tramite l'utilizzo di blocchi Simulink dedicati consente di accedere alle periferiche dell'autopilota ed implementare schemi di controllo direttamente nel controller di volo.

L'integrazione con l'architettura software PX4 illustrata in Figura 3.5 avviene grazie alla sostituzione dei moduli che si occupano di implementare lo schema di controllo, ovvero Position Controller ed Attitude & Rate Controller evidenziati in Figura 3.9. Questi vengono disabilitati e il nuovo modulo denominato px4_simulink_app, generato direttamente dall'Embedded Coder [64] in C++ a partire dal modello Simulink del controller, viene integrato nello stack PX4.

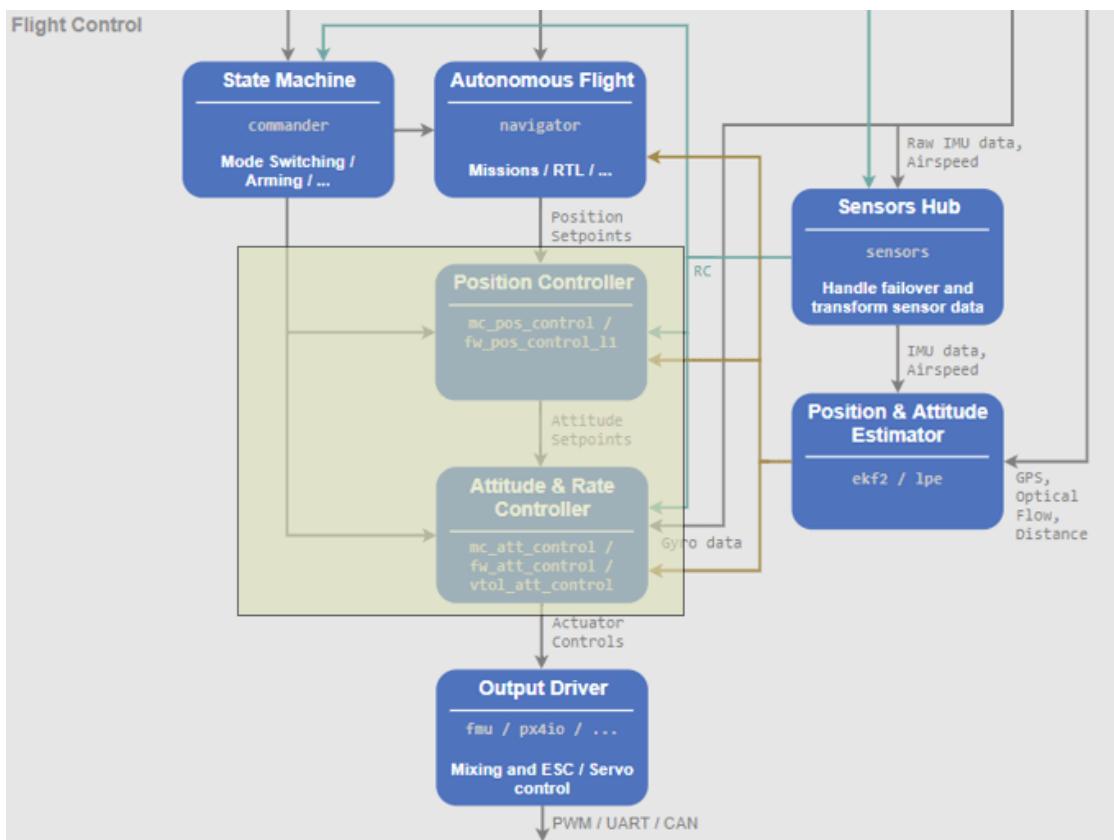


Figura 3.9: Sostituzione dei moduli di controllo PX4 [65].

Il *toolbox* consente inoltre di implementare le simulazioni SITL [66] e HITL [67] così da testare le performance del controllore sviluppato, queste possono essere eseguite secondo due modalità:

- Build
- Monitor & Tune

La prima consente di caricare l'algoritmo di controllo sviluppato nel controllore (simulato o reale) e successivamente eseguire le simulazioni senza poter monitorare i segnali interni al controller o modificare i parametri dello stesso, nel secondo caso oltre ad effettuare l'implementazione dell'algoritmo è possibile visualizzare i segnali interni e variare i parametri mentre la simulazione è in esecuzione.

Quest'ultima tecnica consente quindi di verificare in tempo reale le prestazioni del controllore di volo al variare dei parametri.

Per definire la dinamica del modello da emulare si può utilizzare il simulatore jMAVSim [68] oppure sviluppare il modello dinamico del velivolo tramite Simulink. JMAVSim è un simulatore sviluppato in linguaggio Java che sfrutta il modello indicato in [69] e consente di visualizzare l'animazione 3D del quadricottero; il principale svantaggio nell'impiego di quest'ultimo è dovuto al fatto che l'eventuale modifica del modello necessita di una correzione del codice sorgente. Chiaramente utilizzando il modello Simulink si sopperisce a questo inconveniente essendo la dinamica del sistema definibile dall'utente stesso.

Capitolo 4

Implementazione

Per l'implementazione del modello e degli schemi di controllo illustrati nel Capitolo 2 è stato impiegato l'ambiente di sviluppo MATLAB/Simulink. Sono state definite tre diverse tipologie di simulazione che consentono di emulare, con differenti livelli di astrazione hardware, il comportamento del velivolo soggetto agli schemi di controllo descritti precedentemente. Le simulazioni sviluppate sono:

- Simulink
- Software In The Loop
- Hardware In The Loop

La prima è la tipica simulazione in cui viene definito il modello dinamico e il controllore tramite blocchi Simulink del tutto generici, non vi è alcun riferimento alla piattaforma hardware da utilizzare.

Nella simulazione SITL vengono impiegati dei blocchi dedicati allo stack software PX4 [70] ed al protocollo di comunicazione MAVLink [71], pertanto la tipologia di variabili da utilizzare è vincolata dalla necessaria compatibilità di queste con i sopracitati blocchi, sia per il controllore che per il modello dinamico. La simulazione sviluppata differisce da quella proposta in [72]; questo perché i moduli impiegati in [72] per interfacciare modello e controller introducono un ritardo significativo nella comunicazione fra i due schemi.

La simulazione HITL prevede lo sviluppo di due modelli Simulink distinti: lo schema che descrive la dinamica del quadrirotore e il controller, che verrà implementato nel Pixracer R15 come illustrato in 3.4. È bene specificare che per lo sviluppo del codice nel controller di volo è necessario utilizzare solamente blocchi tempo discreto.

Gli algoritmi di controllo utilizzati in simulazione sono quelli che consentono di specificare la traiettoria di riferimento in termini di x , y , z , e ψ , ovvero Output feedback (Sezione 2.3.4) e State feedback con termine integrale (Sezione 2.3.5). I valori dei parametri utilizzati per il modello dinamico (2.2) sono riportati in Tabella 4.1⁵.

⁵Valori numerici impiegati in [73].

Parametro	Unità di misura	Valore
m	kg	1.2
g	m/s^2	9.81
I_x	kNm^2	$2.353e^{-3}$
I_y	kNm^2	$2.353e^{-3}$
I_z	kNm^2	$5.262e^{-2}$

Tabella 4.1: Parametri del modello.

Prima di procedere allo sviluppo degli schemi di controllo è stato necessario linearizzare il modello come descritto in 2.2.1, lo script che si occupa di effettuare questa operazione è chiamato quadcopter_linearized_dynamics.m, il codice è riportato nel listato 4.1. Questo produce le matrici A e B che vengono salvate nel file linearized_system.mat (riga 42) che verrà impiegato negli script successivi.

```

1 clc;
2 clear;
3
4 syms x xd y yd z zd phi phid theta thetad psi psid T tau_phi tau_theta tau_psi
5 g m Ix Iy Iz
6
7 dX(1:3,1)=[xd, yd, zd];
8
9 dX(4:6,1)=[phid, thetad, psid];
10
11 % linear accelerations
12 dX(7:9,1)=[-(cos(psi)*sin(theta)*cos(phi) + sin(psi)*sin(phi))*(T/m)
13 -(sin(psi)*sin(theta)*cos(phi) - sin(phi)*cos(psi))*(T/m)
14 -(cos(theta)*cos(phi)*(T/m))+g];
15
16 % angular accelerations
17 dX(10:12,1)=[tau_phi/Ix; tau_theta/Iy; tau_psi/Iz];
18
19 A = jacobian(dX, [x y z phi theta psi xd yd zd phid thetad psid])
20 A = subs(A, [x y z phi theta psi xd yd zd phid thetad psid T tau_phi tau_theta
21 tau_psi], ...
22 [0 0 -1 0 0 0 0 0 0 0 0 0 m*g 0 0 0])
23
24 B = jacobian(dX, [T tau_phi tau_theta tau_psi]);
25 B = subs(B, [x y z phi theta psi xd yd zd phid thetad psid], [0 0 -1 0 0 0 0 0
26 0 0 0 0])
27
28 %%%
29 % subs with numerical values
30 %
31 % g = 9.81; % gravitational acceleration
32 %
33 % Ix = 2.353 * exp(-3); % inertia along x-axis
34 % Iy = 2.353 * exp(-3); % inertia along y-axis
35 % Iz = 5.262 * exp(-2); % inertia along z-axis
36

```

```

37 A = double(subs(A, [g m Ix Iy Iz], [9.81 1.2 2.353*exp(-3) 2.353*exp(-3) 5.262*
38   exp(-2)]))
39 B = double(subs(B, [g m Ix Iy Iz], [9.81 1.2 2.353*exp(-3) 2.353*exp(-3) 5.262*
40   exp(-2)]))
41 save("linearized_system.mat", "A", "B")

```

Script 4.1: quadcopter_linearized_dynamics.m

4.1 Simulazione Simulink

Il file MATLAB denominato load_model_parameters.m si occupa di caricare i parametri indicati in Tabella 4.1.

```

1 % script for load model parameters
2
3 clc;
4
5 g = 9.81; % gravitational acceleration
6
7 Ix = 2.353 * exp(-3); % inertia along x-axis
8 Iy = 2.353 * exp(-3); % inertia along y-axis
9 Iz = 5.262 * exp(-2); % inertia along z-axis
10
11 m = 1.2; % mass

```

Script 4.2: load_model_parameters.m Simulink

4.1.1 Output feedback

Lo script output_feedback.m calcola le variabili che consentono di implementare il controllore Output feedback descritto in 2.3.4: carica il modello linearizzato, definisce le matrici C e D , il costo dello stato Q e del controllo R , ed infine calcola le matrici K ed L . Viene inoltre definito l'osservatore tempo discreto poiché, come anticipato, è necessario per la simulazione HITL.

```

1 clc;
2
3 load('../linearized_system.mat')
4
5 C = zeros(4,12);
6 C(1,1) = 1;
7 C(2,2) = 1;
8 C(3,3) = 1;
9 C(4,6) = 1;
10
11 D = zeros(size(C,1), size(B,2));
12
13 Q = eye(size(A)); % state cost
14 R = 0.01; % control cost
15
16 K = -lqr(A, B, Q, R);
17 eig_controller = eig(A+B*K)
18
19 L = -lqr(A', C', Q, R);

```

```

20 L = L';
21
22 eig_observer = eig(A+L*C)
23
24 % observer model
25 A_obs = A+B*K+L*C;
26 B_obs = L;
27 C_obs = eye(size(A_obs));
28 D_obs = zeros(size(C_obs,1), size(B_obs, 2));
29
30 u_bar = m*g; % feedforward term
31
32 %% discrete time
33
34 sys_obs = ss(A_obs, B_obs, C_obs, D_obs);
35
36 Ts = .01; % sample time
37 obs_DT = c2d(sys_obs, Ts, 'zoh');

```

Script 4.3: output_feedback.m Simulink

Si ricavano quindi le matrici K ed L così da garantire che gli autovalori di $(A + BK)$ ed $(A + LC)$ siano posizionati nella parte sinistra del piano complesso. In particolare si ottiene:

$$\begin{aligned}\sigma(A + BK) = \{ & -1.0000 + 0.0000i, -2.2141 + 2.2155i, -2.2141 - 2.2155i, \\ & -85.3556 + 0.0000i, -85.3556 - 0.0000i, -14.0065 + 0.0000i, \\ & -8.2722 + 0.0000i, -2.2141 + 2.2155i, -2.2141 - 2.2155i, \\ & -1.0000 + 0.0000i, -1.0074 + 0.0000i, -1.0026 + 0.0000i \}\end{aligned}$$

$$\begin{aligned}\sigma(A + LC) = \{ & -2.1605 + 2.2681i, -2.1605 - 2.2681i, -2.1605 + 2.2681i, \\ & -2.1605 - 2.2681i, -9.9977 + 0.0000i, -9.9977 + 0.0000i, \\ & -9.9494 + 0.0000i, -9.9494 + 0.0000i, -1.0001 + 0.0000i, \\ & -1.0001 + 0.0000i, -1.0051 + 0.0000i, -1.0051 + 0.0000i \}\end{aligned}$$

È stato quindi implementato lo schema Simulink con i guadagni calcolati precedentemente, il modello è illustrato in Figura 4.1. Dall'immagine si osserva come gli input del modello dinamico, ovvero il blocco `quadcopter_model_simulink`, non sono le velocità dei rotori bensì la spinta T e le coppie τ_φ , τ_θ , τ_ψ ; per questo motivo non è stato necessario definire la lunghezza l riportata nelle equazioni (2.4), (2.5). Lo stesso schema è stato definito a tempo discreto sostituendo il modulo tempo continuo che implementa il modello dell'osservatore con l'equivalente tempo discreto; le matrici A_{obs} , B_{obs} , C_{obs} , D_{obs} calcolate da riga 25 a 28 vengono cambiate con le corrispondenti tempo discreto obs_DT.A , obs_DT.B , obs_DT.C , obs_DT.D ricavate a riga 37.

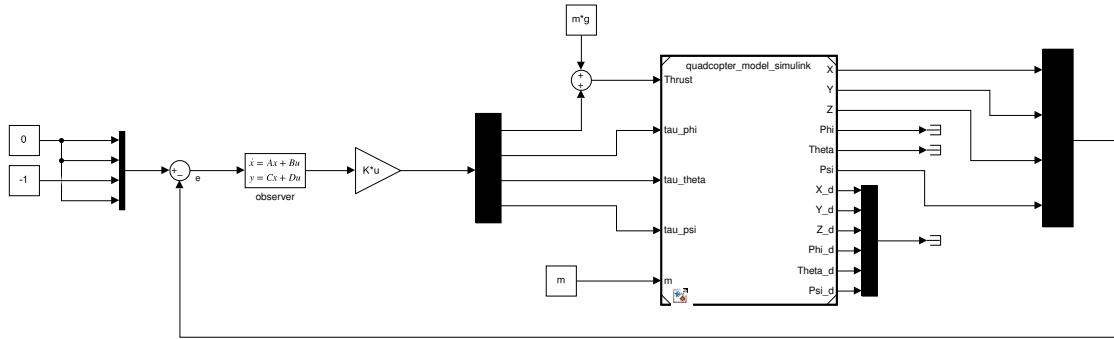


Figura 4.1: Schema Simulink controller Output feedback.

4.1.2 State feedback con termine integrale

Per calcolare i guadagni del controllore è stato utilizzato il file chiamato `integral_state_feedback.m`, in maniera simile a quanto accade per il codice 4.3 vengono caricate le matrici del modello linearizzato, fissati i costi Q ed R , e ricavati i guadagni K_P e K_I .

```

1 clc;
2
3 load('../linearized_system.mat')
4
5 % build extended system
6
7 C = zeros(4,12);
8 C(1,1) = 1;
9 C(2,2) = 1;
10 C(3,3) = 1;
11 C(4,6) = 1;
12
13 D = zeros(size(C,1), size(B,2));
14
15 Ae = [A zeros(12, 4); -C zeros(4, 4)];
16 Be = [B; zeros(4, 4)];
17
18 size(ctrb(Ae, Be))
19 rank(ctrb(Ae, Be))
20
21 E = eye(size(Ae)); % state cost
22 R = 0.01; % control cost
23
24 Ke = -lqr(Ae, Be, E, R);
25
26 KP = Ke(:, 1:12);
27 KI = Ke(:, 13:end);
28
29 eigCLsys = eig([A+B*KP B*KI; -C zeros(size(C, 1), size(B*KI, 2))])
30
31 Ts = .01; % sample time

```

Script 4.4: `integral_state_feedback.m` Simulink

Ancora una volta l'obiettivo è quello di avere il sistema esteso (A_e, B_e) asintoticamente stabile, dalla matrice K_e vengono ricavate le due sottomatrici K_P e K_I in modo tale da avere $\sigma(A_e + B_e K) \subset \mathbb{C}^-$.

$$\begin{aligned}\sigma(A_e + B_e K_e) = \{ & -85.3556 + 0.0000i, -85.3556 + 0.0000i, -14.0066 + 0.0000i, \\ & -8.2731 + 0.0000i, -2.2077 + 2.2104i, -2.2077 - 2.2104i, \\ & -2.2077 + 2.2104i, -2.2077 - 2.2104i, -0.8681 + 0.5037i, \\ & -0.8681 - 0.5037i, -0.8691 + 0.5000i, -0.8691 - 0.5000i, \\ & -0.8691 + 0.5000i, -0.8691 - 0.5000i, -0.8668 + 0.5013i, \\ & -0.8668 - 0.5013i\}\end{aligned}$$

Avendo verificato la correttezza dei guadagni ottenuti è stato implementato lo schema di controllo illustrato in Figura 4.2. Come già spiegato in 4.1.1 non è necessario ricavare la velocità angolare di ogni singola elica. È possibile definire l'equivalente schema tempo discreto sostituendo il termine integrale a valle del primo sommatore con l'integratore tempo discreto.

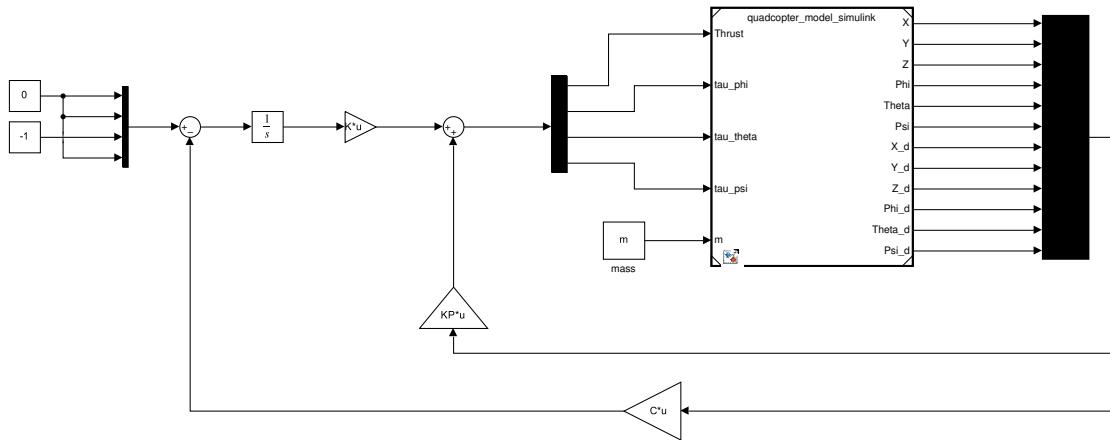


Figura 4.2: Schema Simulink controller state feedback con termine integrale.

4.2 Simulazione SITL

Anche in questo caso è stato utilizzato uno script denominato `load_model_parameters.m` che si occupa di caricare i parametri necessari. A differenza del precedente vengono definite due nuove matrici denominate `mixer_matrix` e `inverse_mixer_matrix` utilizzate per scomporre i valori complessivi dei segnali di controllo nel contributo di ogni singolo motore e, successivamente, ricomporre i segnali così da ricavare T , τ_φ , τ_θ , τ_ψ . La definizione di un tempo di campionamento tramite la variabile `SampleTime` (riga 23) si è resa necessaria per l'utilizzo dei blocchi MAVLink impiegati nella simulazione.

```

1 % script for load model parameters
2
3 clc;
4
5 g = 9.81; % gravitational acceleration
6
7 Ix = 2.353 * exp(-3); % inertia along x-axis
8 Iy = 2.353 * exp(-3); % inertia along y-axis

```

```

9 Iz = 5.262 * exp(-2); % inertia along z-axis
10
11 m = 1.2; % mass
12
13 mixer_matrix = [ 1   1   1   1;
14           1   -1   1   -1;
15           1   1   -1   -1;
16           1   -1   -1   1] / 4;
17
18 inverse_mixer_matrix = [1   1   1   1;
19           1   -1   1   -1;
20           1   1   -1   -1;
21           1   -1   -1   1];
22
23 SampleTime = 0.002;

```

Script 4.5: load_model_parameters.m SITL

4.2.1 Output feedback

Il file che si occupa di calcolare i parametri del controllore è sintatticamente identico al listato 4.3, l'unica differenza è che il tempo di campionamento T_s utilizzato per ricavare la dinamica dell'osservatore tempo discreto è stato fissato uguale a `SampleTime` (riga 36). Di conseguenza anche i parametri K ed L saranno gli stessi portando al medesimo risultato in termini di posizionamento degli autovalori.

```

1 clc;
2
3 load('../linearized_system.mat')
4
5 C = zeros(4,12);
6 C(1,1) = 1;
7 C(2,2) = 1;
8 C(3,3) = 1;
9 C(4,6) = 1;
10
11 D = zeros(size(C,1), size(B,2));
12
13 Q = eye(size(A)); % state cost
14 R = 0.01; % control cost
15
16 K = -lqr(A, B, Q, R);
17 eig_controller = eig(A+B*K);
18
19 L = -lqr(A', C', Q, R);
20 L = L';
21
22 eig_observer = eig(A+L*C);
23
24 u_bar = m*g; % feedforward
25
26
27 % build continuos time observer model
28 A_obs = A + B*K + L*C;
29 B_obs = L;
30 C_obs = eye(12,12);

```

```

31 D_obs = zeros(12, 4);
32 sys_obs = ss(A_obs, B_obs, C_obs, D_obs);
33
34 %% discrete time
35
36 Ts = SampleTime; % sample time
37
38 obs_DT = c2d(sys_obs, Ts, 'zoh'); % discrete time observer model

```

Script 4.6: output_feedback.m SITL

Lo schema di controllo Simulink è illustrato in Figura 4.3. Oltre al modello del controllore e del quadricottero è presente un ulteriore blocco che, prendendo in ingresso le variabili di uscita⁶ del sistema, genera i corrispondenti messaggi uORB da inviare al controller. Quest'ultimo, dopo aver ricavato i segnali di controllo, calcola il contributo di ogni motore e tramite un messaggio MAVLink denominato HIL_ACTUATOR_CONTROL [74] li invia al modello dinamico⁷: Il modello svolgendo l'operazione inversa ricava i segnali T , τ_φ , τ_θ , τ_ψ .

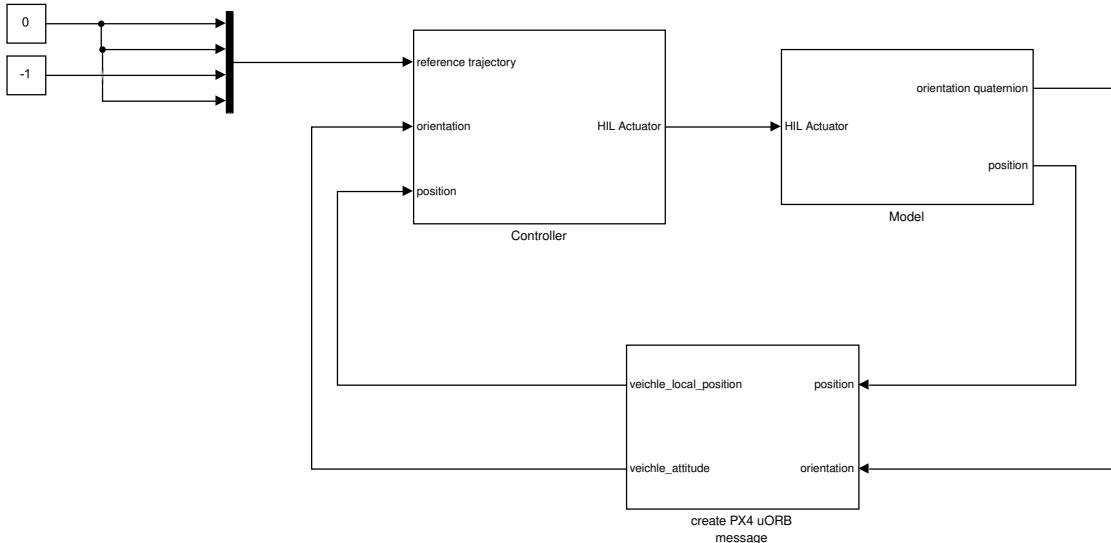


Figura 4.3: Schema SITL controller Output feedback.

⁶Gli angoli vengono espressi come quaternione per garantire la compatibilità con i messaggi uORB.

⁷Il valore del contributo di ogni motore deve essere espresso nel formato uint16.

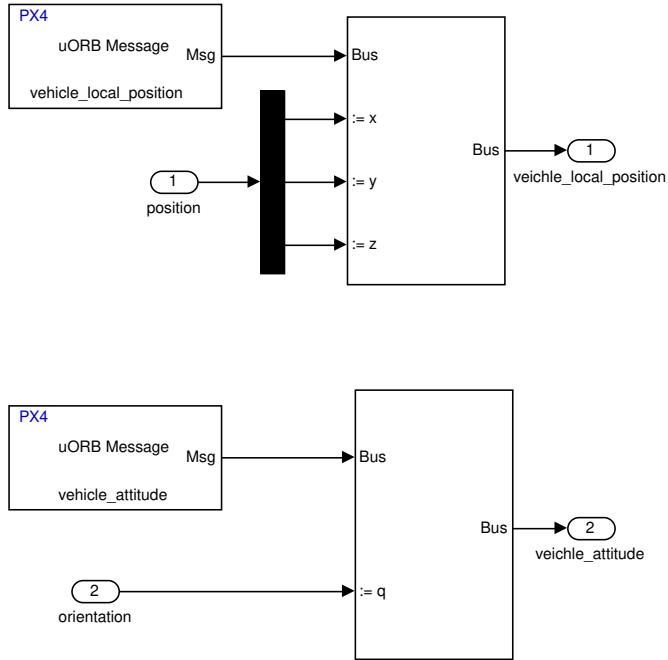


Figura 4.4: Sottosistema create PX4 uORB message controller Output feedback.

4.2.2 State feedback con termine integrale

Il codice che si occupa di calcolare i guadagni del controllore è identico a quello illustrato in 4.4 per la simulazione Simulink semplice, si ottengono quindi gli stessi valori della matrice K e di conseguenza anche delle sottomatrici K_P e K_I . Lo schema Simulink segue lo stesso principio del modello illustrato in 4.3 con la differenza che l'uscita di Model è l'intero vettore di stato. Ancora una volta il blocco denominato `create PX4 uORB message` si occupa di generare i messaggi da passare al controllore.

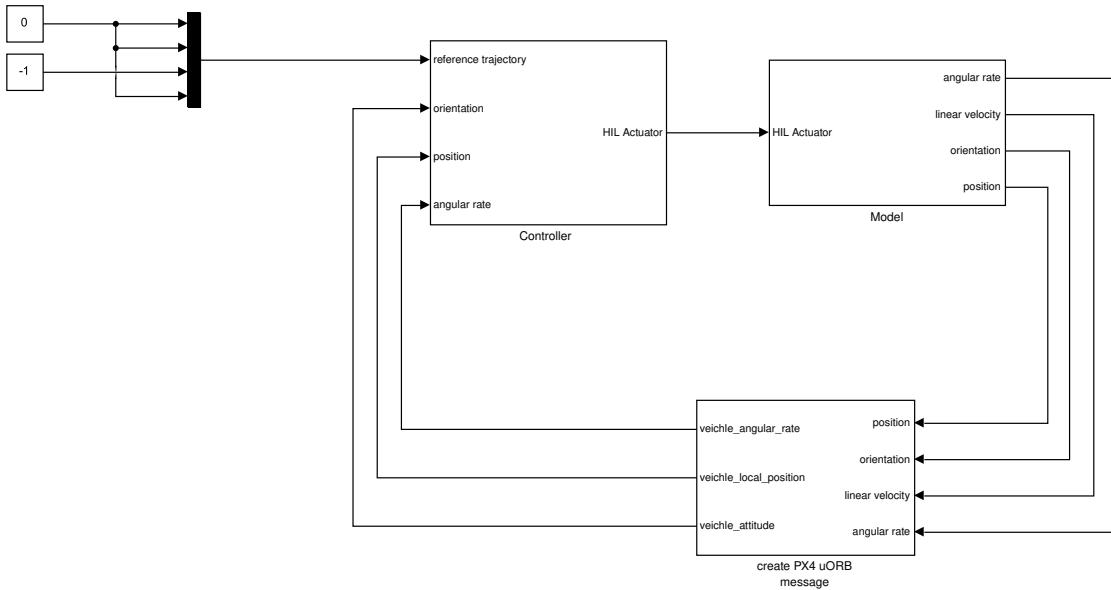


Figura 4.5: Schema SITL controller State feedback con termine integrale.

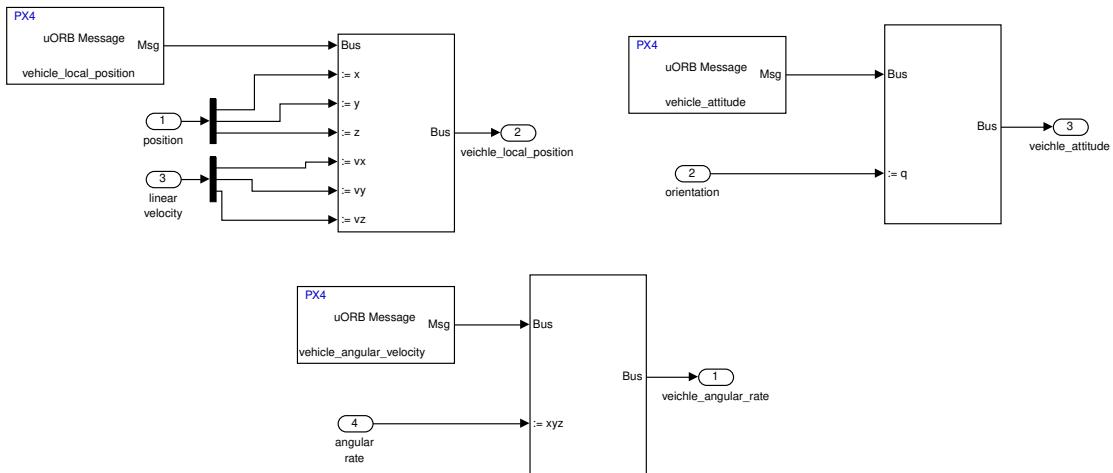


Figura 4.6: Sottosistema create PX4 uORB message controller State feedback con termine integrale.

4.3 Simulazione HITL

Per la simulazione HITL è stato definito il file denominato `load_model_parameters.m`, come già accaduto in 4.1, 4.2. Vengono definiti i parametri necessari per emulare la comunicazione MAVLink oltre alle variabili introdotte nei listati 4.2, 4.5.

```

1 % script for load model parameters
2
3 clc;
4
5 g = 9.81; % gravitational acceleration
6
7 Ix = 2.353 * exp(-3); % inertia along x-axis
8 Iy = 2.353 * exp(-3); % inertia along y-axis
9 Iz = 5.262 * exp(-2); % inertia along z-axis
10
11 m = 1.2; % mass
12
13 mixer_matrix = [ 1 1 1 1;
14                 1 -1 1 -1;
15                 1 1 -1 -1;
16                 1 -1 -1 1] / 4;
17
18 inverse_mixer_matrix = [1 1 1 1;
19                         1 -1 1 -1;
20                         1 1 -1 -1;
21                         1 -1 -1 1];
22
23
24 SampleTime = 0.002;
25
26 % parameters for messages creation
27 m_to_mm = 1000;
28 ref_height = -71.3232;
29 ms_to cms = 100;
30
31 altToPrsGain = 12.0173;

```

```

32 altToPrsBias = 101270.95;
33
34 ms2_to_mg = (1/9.80665)*1000;
35
36 % lat, long ing@UNIPG
37 ref_lat = 43.118926;
38 ref_lon = 12.355420;
39
40 uT_to_gauss = 0.0100;

```

Script 4.7: load_model_parameters.m HITL

In questa tipologia di simulazione gli schemi Simulink del velivolo e del controller vengono definiti in file separati, questo perché lo schema contenente l'algoritmo di controllo sarà poi caricato nel *flight controller* come discusso in 3.4. Il modello che descrive la dinamica del quadricottero è illustrato in Figura 4.7 ed è composto, oltre che da *Quadcopter plant* che implementa le equazioni (2.2), da altri moduli necessari per la comunicazione con il controller. Il modulo denominato MAVLink Bridge Source definisce la porta seriale per ricevere i segnali dal controllore, MAVLink2rotor estrae dal messaggio HIL_ACTUATOR_CONTROL il corrispondente contributo di ogni rotore, questi vengono combinati da *rotor2control* così da ricavare i canonici segnali di controllo. Le variabili di uscita del modello vengono poi utilizzate da *create MAVLink messages* per creare i messaggi MAVLink HIL_SENSOR, HIL_STATE_QUATERNION, HIL_GPS che verranno passati a MAVLink Bridge Sink e inviati al controller.

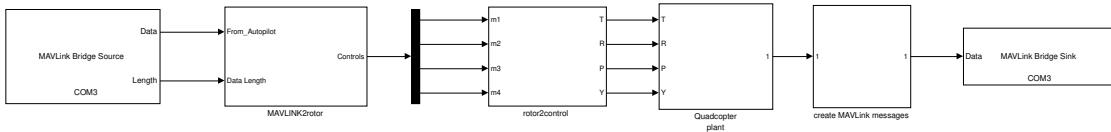


Figura 4.7: Schema del modello dinamico HITL

4.3.1 Output feedback

I parametri del controller sono stati ricavati con il medesimo file utilizzato per la simulazione SITL, riportato in 4.6; i valori di K ed L sono invariati rispetto a quelli calcolati in 4.1.1 e di conseguenza anche la posizione degli autovalori di $(A+BK)$ e $(A+LC)$. In questo caso il controller da implementare deve essere convertito necessariamente nell'equivalente tempo discreto. La Figura 4.10 illustra l'architettura generale dove sono presenti i blocchi *uORB Read* che consentono di leggere i dati inviati dal modello dinamico. A differenza degli schemi precedentemente mostrati è stato impiegato il modulo che si occupa di verificare se il velivolo è armato: solamente in questo caso viene abilitato l'algoritmo di controllo encapsulato nel sottosistema *controller* e mostrato in Figura 4.9. Quest'ultimo non è altro che il medesimo schema illustrato in 4.1.1 e impiegato anche in 4.2.1, la sola differenza è il modello dell'osservatore tempo discreto.

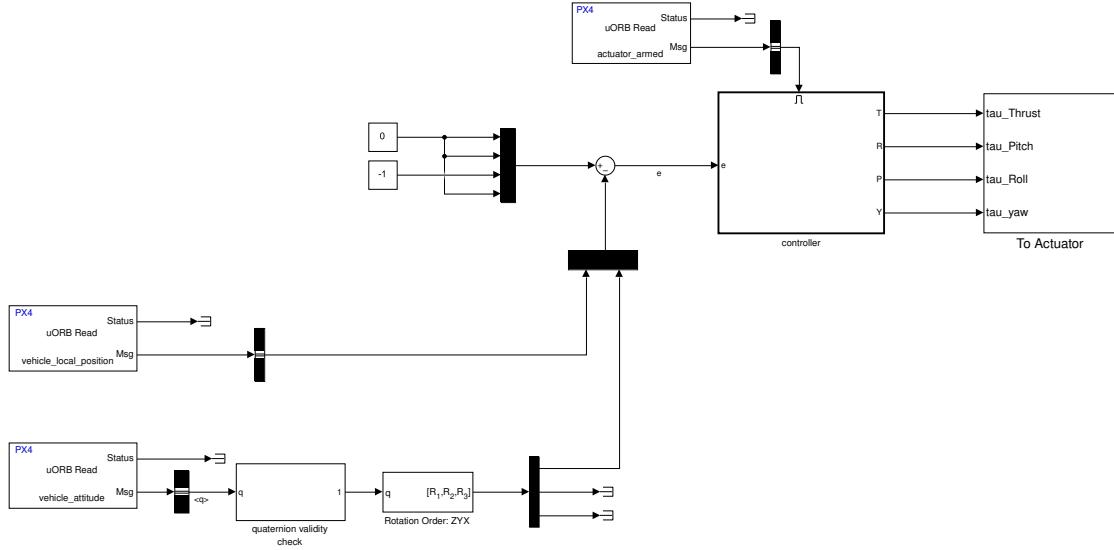


Figura 4.8: Schema HITL controller Output feedback.

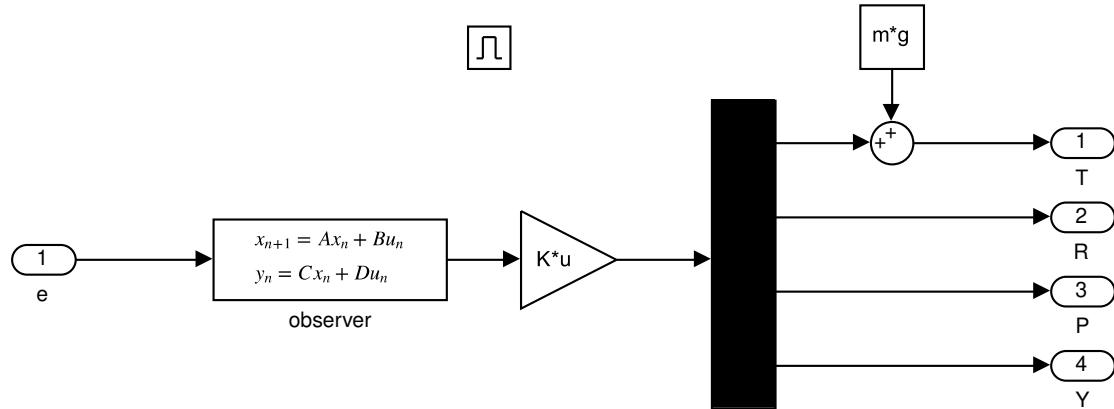


Figura 4.9: Sottosistema controller Output feedback HITL.

4.3.2 State feedback con termine integrale

Il file di configurazione dei parametri è il medesimo della simulazione Simulink e SITL mostrato in 4.4, si ottengono conseguentemente gli stessi valori di K_P e K_I . Lo schema generale del controllore è mostrato in Figura 4.10, in questo caso sono presenti più blocchi uORB Read poiché si ha un feedback completo dello stato. In particolare il messaggio `vehicle_local_position` contiene le informazioni riguardanti la posizione e le velocità lineari; la posizione angolare e le corrispondenti velocità vengono inviate in due topic distinti denominati rispettivamente `vehicle_attitude` e `vehicle_angular_velocity`. Anche in questo caso è presente il modulo che si occupa di leggere lo stato del quadrirotore ed abilitare, in caso di velivolo armato, l'algoritmo di controllo mostrato in Figura 4.11. Come si può immaginare è necessaria la sostituzione dell'integrale tempo continuo con il corrispondente tempo discreto.

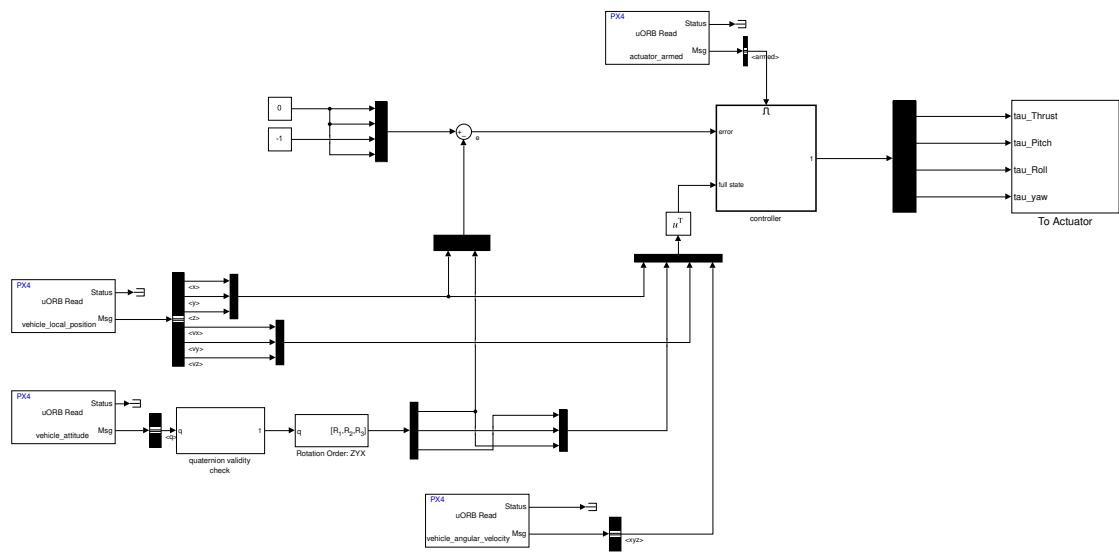


Figura 4.10: Schema HIL controller Output feedback.

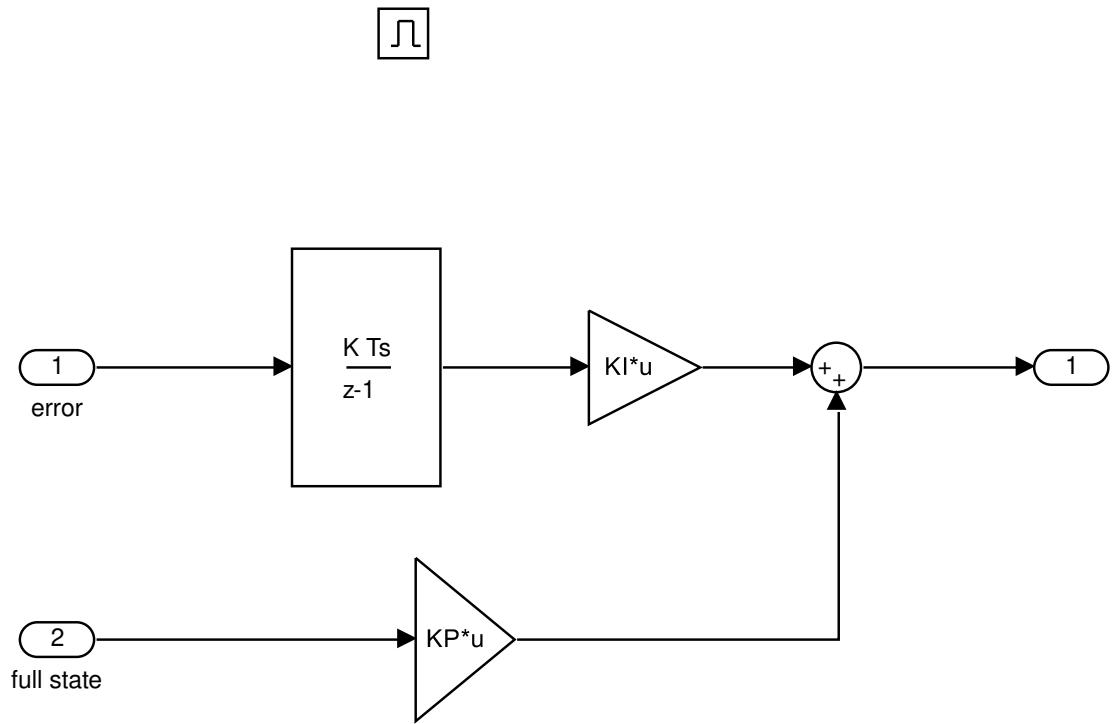


Figura 4.11: Sottosistema controller Output feedback HIL.

Capitolo 5

Esperimenti e Risultati

Per testare gli schemi di controllo illustrati nel Capitolo 4 sono state definite due tipologie di simulazione:

1. *Hovering*
2. *Hovering* con massa variabile

Entrambe hanno l'obiettivo di verificare che l'algoritmo di controllo riesca a garantire l'*hovering* del velivolo alla quota di un metro. Nel primo caso non sono presenti incertezze parametriche, nel secondo la massa del velivolo varia dinamicamente durante la simulazione. Le simulazioni illustrate sono state svolte in modalità HITL ad eccezione della prima tipologia, eseguita anche in Simulink così da evidenziare analogie e differenze fra il controller tempo continuo e tempo discreto.

È interessante notare come nella simulazione HITL il controller necessita comunque di ricevere il comando di *arming* via QGC per far in modo che i segnali inviati abbiano concretamente effetto nel modello dinamico; difatti se lo schema che descrive la dinamica è in esecuzione ma non è ancora stato inviato il comando di *arming* il modello riceve i segnali di controllo identicamente nulli. Inoltre tramite la schermata *Fly View* è possibile monitorare l'andamento della simulazione come se venisse eseguito il task da un velivolo reale.

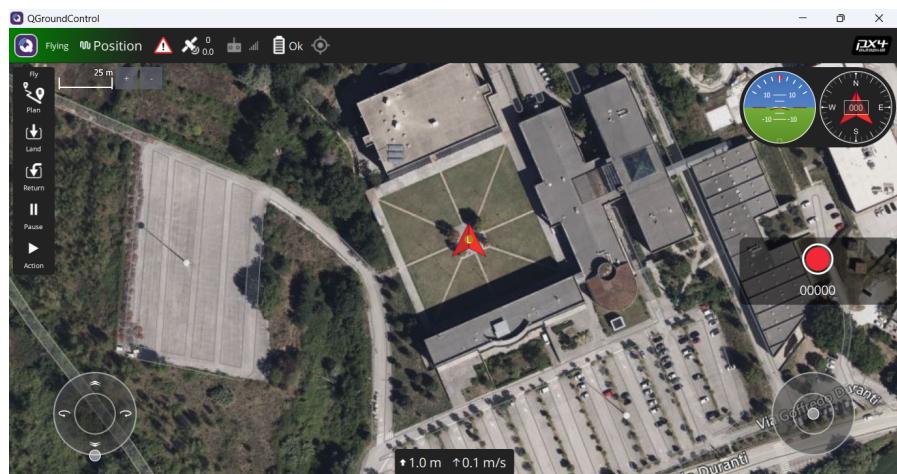


Figura 5.1: Schermata *Fly View* simulazione HITL.

5.1 Hovering

5.1.1 Simulazione Simulink

I risultati ottenuti per la simulazione *Hovering* in Simulink utilizzando lo schema di controllo Output feedback sono mostrati nelle Figure 5.2, 5.3. L'andamento della quota z è praticamente identico in entrambi i casi, la variabile \bar{z} indica la quota di riferimento. Avendo definito un tempo di campionamento pari a 0.01s per la discretizzazione dell'osservatore (come mostrato nel codice 4.3), i segnali di controllo non differiscono nel caso di osservatore tempo continuo e tempo discreto. Il velivolo effettua un moto puramente verticale, di conseguenza $\tau_\varphi, \tau_\theta, \tau_\psi$ sono identicamente nulli per tutta la durata della simulazione.

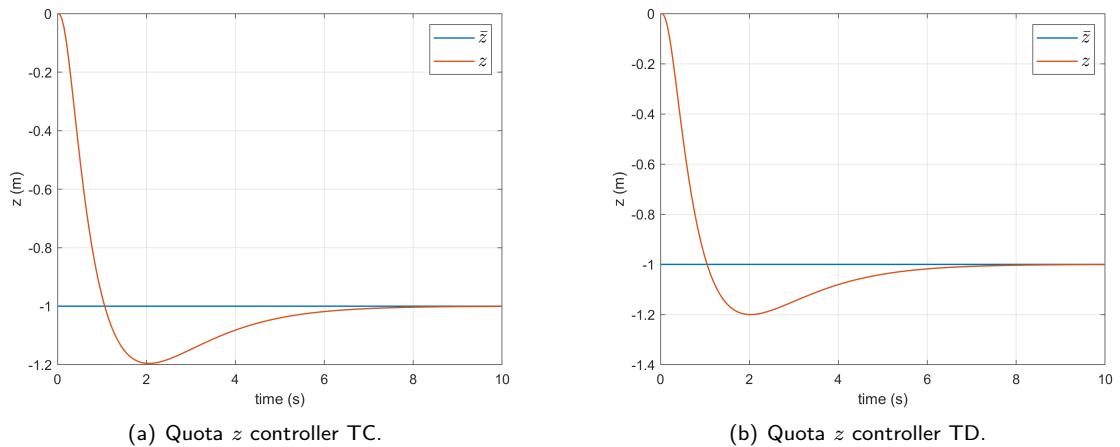


Figura 5.2: Quota z Hovering Output feedback Simulink.

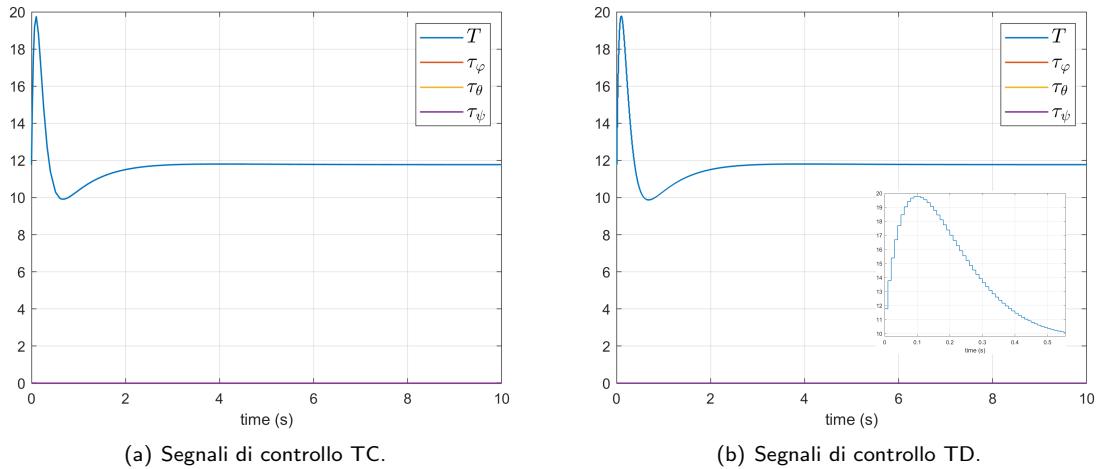


Figura 5.3: Segnali di controllo Hovering Output feedback Simulink.

È stato svolto lo stesso esperimento utilizzando il controller State feedback con termine integrale, impiegando sia l'integratore tempo continuo che il duale tempo discreto. I risultati dell'andamento di z sono mostrati nelle Figure 5.4, 5.5: anche in questo caso non vi è alcuna differenza lungo z nei casi tempo continuo e tempo discreto; il medesimo discorso vale per l'andamento dei segnali di controllo.

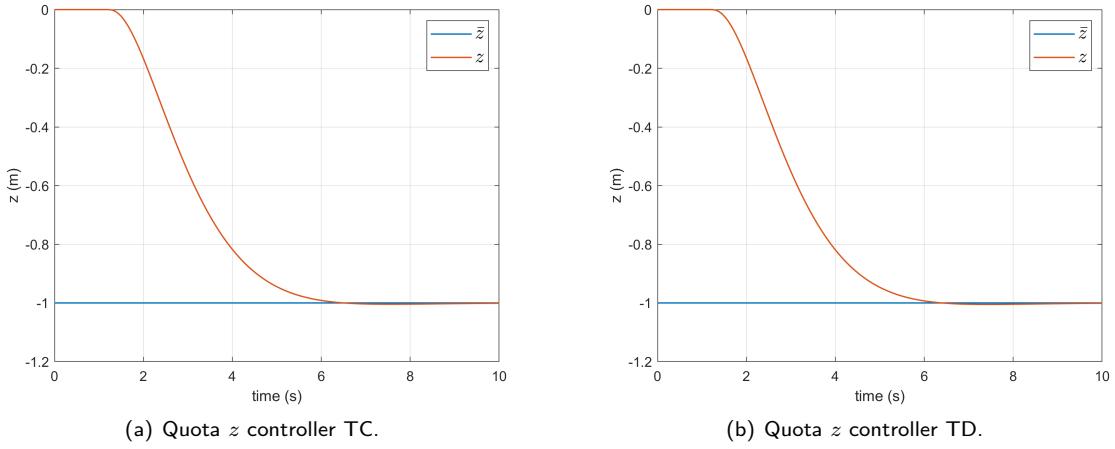


Figura 5.4: Quota z Hovering State feedback con termine integrale Simulink.

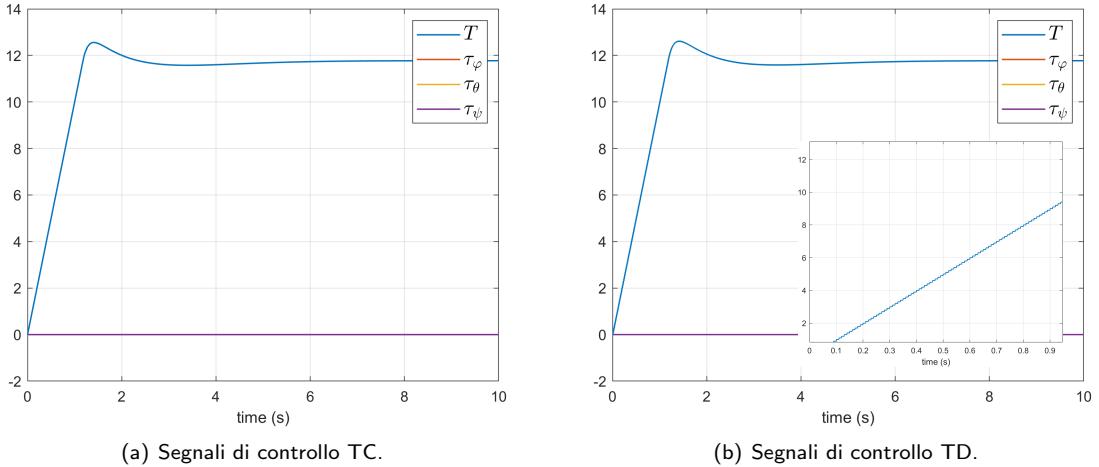


Figura 5.5: Segnali di controllo Hovering State feedback con termine integrale Simulink.

Confrontando i grafici dei due schemi di controllo appare evidente la differenza in termini di performance. Utilizzando come riferimento le simulazioni tempo continuo si osserva come il controller Output feedback riesca a raggiungere in un tempo pari a circa 1s la quota \bar{z} , presentando una sottoelongazione contenuta; il controller State feedback con termine integrale mostra invece un andamento più regolare. Fra i due schemi è chiaramente preferibile la prima soluzione, ovvero Output feedback, poiché si mostra nettamente più presentate in termini di tempo impiegato per salire di quota.

5.1.2 Simulazione HITL

Il medesimo esperimento illustrato in 5.1.1 è stato svolto in modalità HITL, chiaramente in questo caso i segnali di controllo sono solamente a tempo discreto. In questa tipologia di simulazione è necessario esprimere i segnali nel formato `uint16` come precedentemente spiegato in 4.2.1: così facendo si ha una perdita di precisione rispetto alle simulazioni svolte interamente in ambiente Simulink a causa della quantizzazione dei segnali.

Come si osserva in Figura 5.6 il controller Output feedback tende ad inseguire correttamente la traiettoria, tuttavia a circa 25s il segnale di controllo diminuisce leggermente causando l'abbassamento

della quota del drone che si stabilizza a circa 0.8m, questo fenomeno è causato dalla quantizzazione dei segnali; il controller State feedback con termine integrale oscilla invece in maniera quasi impercettibile attorno alla quota di 1m. Analizzando più in dettaglio i segnali di controllo si osserva che nel caso State feedback il valore della spinta oscillava periodicamente attorno ad $mg = 11.77$, questo infatti è il valore esatto che garantisce l'*'hovering'* del quadrirotore; le oscillazioni si ripercuotono chiaramente sull'andamento di z . In questo caso la soluzione con feedback dallo stato garantisce delle performance migliori.

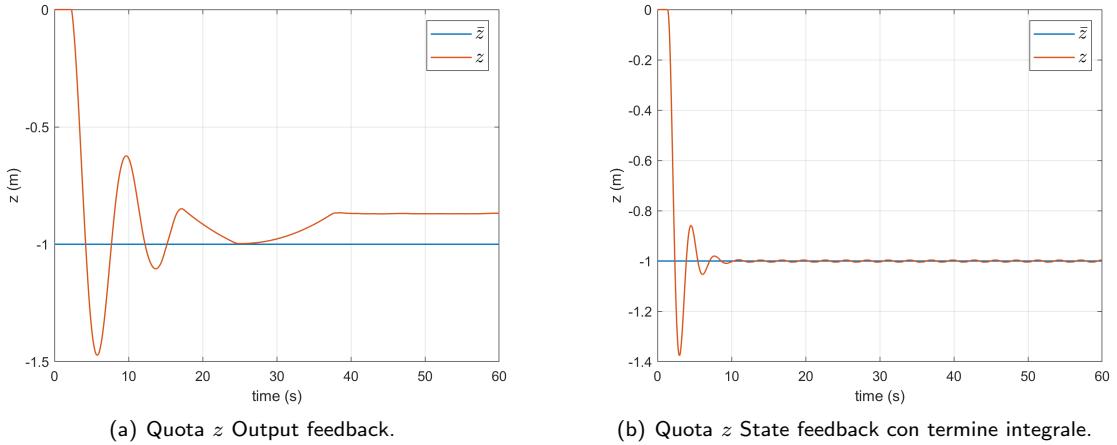


Figura 5.6: Quota z Hovering HITL.

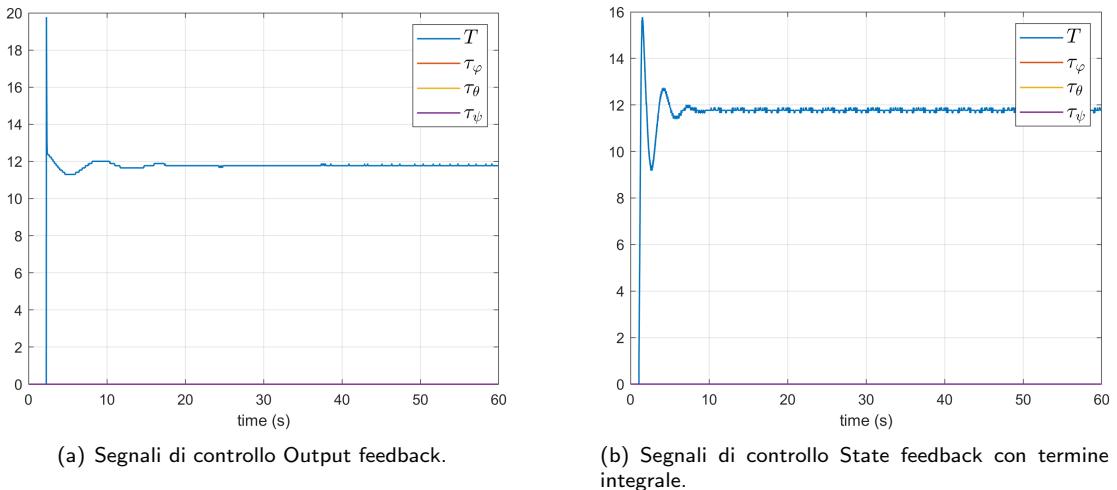


Figura 5.7: Segnali di controllo Hovering HITL.

Confrontando i risultati ottenuti con quelli mostrati in 5.1.1 è interessante notare come, se fossero state effettuate esclusivamente le generiche simulazioni tramite Simulink, si sarebbe presentato il tipico *trade-off* fra prestazioni e robustezza. Utilizzando invece HITL, che rappresenta sicuramente un'emulazione più verosimile dell'ambiente reale, è apparso chiaro che la tipologia di controller Output feedback non offre alcuna garanzia, sia in termini di robustezza (essendo il termine di feedforward \bar{u} dipendente dal valore esatto della massa m) che di performance rispetto al controller State feedback con termine integrale.

5.2 Hovering con massa variabile dinamicamente

In questo esperimento sono state testate le prestazioni degli algoritmi di controllo sviluppati variando dinamicamente la massa del velivolo durante la simulazione. Il valore della massa m viene raddoppiato dopo 20 secondi e triplicato dopo 40, ottenendo quindi una valore che parte da 1.2kg ed arriva a 3.6kg.

Come si può osservare in Figura 5.8 i risultati ottenuti evidenziano la robustezza del controller State feedback con termine integrale rispetto a possibili variazioni della massa a tempo di esecuzione; l'algoritmo impiega un tempo decisamente inferiore per bilanciare la prima variazione rispetto alla seconda. Il controller Output feedback non riesce invece a compensare questo cambiamento poiché il termine di feedforward (2.9) viene calcolato staticamente con quelli che sono i dati iniziali, determinando di conseguenza un comportamento incontrollabile del sistema. I risultati ottenuti mostrano come la conoscenza completa dello stato consenta di ottenere delle performance significativamente migliori rispetto alla sola conoscenza dell'uscita.

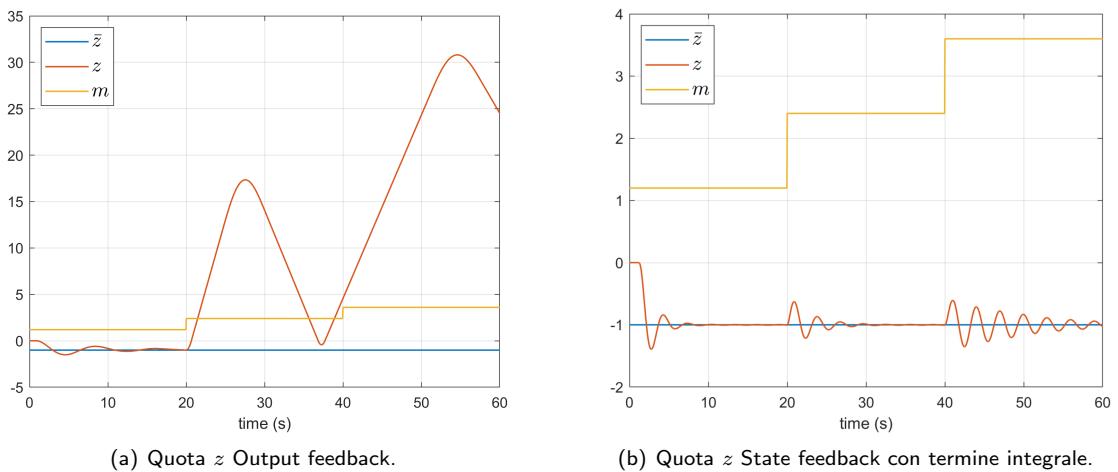


Figura 5.8: Quota z Hovering con massa variabile HITL.

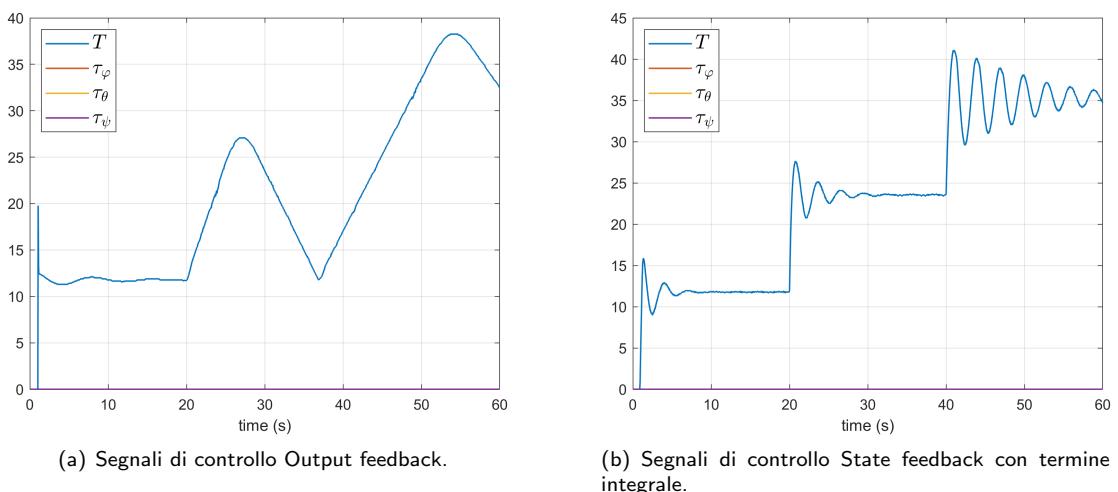


Figura 5.9: Segnali di controllo Hovering con massa variabile HITL.

Un ultimo esperimento è stato svolto facendo variare la massa come spiegato in precedenza e riportandola successivamente al valore originale. Il segnale di controllo T , la massa m e la quota z sono illustrate nell'animazione⁸ in Figura 5.10. Il grafico mostra come il controllore riesca a compensare variazioni significative della massa, sia in positivo che in negativo.

Figura 5.10: Evoluzione di T , m , z controller State feedback con termine integrale HITL.

⁸Le animazioni sono supportate solamente da alcuni lettori PDF: AcrobatReader, PDF-XChange, acroread, and Foxit Reader [75].

Capitolo 6

Conclusioni e Sviluppi Futuri

Nei capitoli precedenti è stato mostrata l'implementazione di un'intera pipeline per lo sviluppo di algoritmi di controllo, con riferimento al mondo dei quadrirotori. Gli schemi di controllo progettati sono stati sintetizzati grazie all'ambiente di sviluppo MATLAB/Simulink e validati secondo differenti tipologie di simulazione. Queste sono state strutturate definendo dapprima delle simulazioni totalmente generiche con l'obiettivo di testare la correttezza degli algoritmi di controllo, successivamente sono stati progressivamente introdotti i moduli dedicati allo stack software PX4 fino a raggiungere una completa compatibilità con il *flight stack*, consentendo quindi di implementare lo schema sviluppato nel controller di volo reale.

I risultati ottenuti hanno mostrato come lo schema Output feedback non riesca a garantire performance affidabili a fronte di incertezze che potrebbero influenzare il modello dinamico, a differenza di quanto mostrato dall'algoritmo State feedback con termine integrale. Un importante spunto di riflessione è dato dai risultati ottenuti in 5.1.2: questi evidenziano come, a parità di modello e di controllore, gli esiti delle sperimentazioni possano essere diametralmente opposti eseguendo delle simulazioni classiche rispetto a quelle HITL. Viene quindi rimarcata la necessità di questa tipologia di simulazione nel caso in cui il fine ultimo sia quello di sviluppare un algoritmo di controllo impiegabile in contesto reale.

In conclusione, il framework implementato consente di validare le performance di un qualsiasi controller (che esegue lo stack software PX4) tramite una pipeline robusta e flessibile allo stesso tempo: gli schemi Simulink illustrati possono essere modificati a piacimento con l'unica accortezza che i blocchi delegati alla comunicazione fra modello e controllore siano implementati correttamente.

Proprio la flessibilità del framework consente a questo progetto di fare da apripista ad un'infinità di possibili sviluppi futuri.

La prima idea potrebbe essere quella di rimuovere dall'algoritmo di controllo State feedback con termine integrale la conoscenza completa dello stato, così facendo andrebbe implementato un controller che sfrutta un osservatore come mostrato per Output feedback. Potrebbero tuttavia essere definite delle tecniche di controllo non lineari e di controllo robusto sperimentando le prestazioni tramite il *tracking* di traiettorie più complesse rispetto al semplice *hovering* a quota assegnata.

Inoltre, avendo definito il modello del velivolo utilizzando Simulink, è possibile modificare quest'ultimo in maniera totalmente arbitraria definendo ad esempio una dinamica più accurata o modificare interamente la stessa così da emulare il comportamento di una diversa tipologia di velivolo (ricordando che PX4 è impiegabile anche per controllare veicoli di superficie o sottomarini [44]).

Le simulazioni sono state svolte considerando il caso in cui il velivolo si trovi ad operare all'esterno, inviando quindi messaggi MAVLink HIL_GPS; sarebbe possibile anche utilizzare messaggi di tipo VISION_POSITION_ESTIMATE e modificare i *topic* del controllore da cui vengono letti i dati dello stato, emulando così uno scenario di navigazione *indoor* tramite un sistema di Motion Capture [76]. Uno sviluppo particolarmente interessante potrebbe essere l'implementazione di uno schema di controllo che racchiude al suo interno due tipologie di controller distinti, uno per la navigazione *outdoor* e l'altro per la navigazione *indoor*.

Un ulteriore possibile utilizzo del framework è quello dell'impiego di HITL in modalità Monitor e Tune, descritta brevemente in 3.4, per effettuare un *fine tuning* dei parametri dell'algoritmo di controllo.

Infine uno sviluppo sicuramente affascinante è quello della messa in opera del controller di volo per pilotare un velivolo reale. In questo caso andrebbero definite tutte le misure di sicurezza necessarie, le modalità di comunicazione con la GCS, come poter inviare al controller una traiettoria di riferimento, e il meccanismo di atterraggio.

Bibliografia e Sitografia

- [1] UM Rao Mogili and BBVL Deepak. Review on application of drone systems in precision agriculture. *Procedia computer science*, 133:502–509, 2018.
- [2] Victor V Klemas. Coastal and environmental remote sensing from unmanned aerial vehicles: An overview. *Journal of coastal research*, 31(5):1260–1267, 2015.
- [3] Jarrod C Hodgson, Shane M Baylis, Rowan Mott, Ashley Herrod, and Rohan H Clarke. Precision wildlife monitoring using unmanned aerial vehicles. *Scientific reports*, 6(1):1–7, 2016.
- [4] Patrick Doherty and Piotr Rudol. A uav search and rescue scenario with human body detection and geolocation. In *AI 2007: Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007. Proceedings* 20, pages 1–13. Springer, 2007.
- [5] Ebtehal Turki Alotaibi, Shahad Saleh Alqefari, and Anis Koubaa. Lsar: Multi-uav collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019.
- [6] Daniel KD Villa, Alexandre S Brandao, and Mário Sarcinelli-Filho. A survey on load transportation using multirotor uavs. *Journal of Intelligent & Robotic Systems*, 98:267–296, 2020.
- [7] Dieter Hausamann, Werner Zirnig, Gunter Schreier, and Peter Strobl. Monitoring of gas pipelines—a civil uav application. *Aircraft Engineering and Aerospace Technology*, 77(5):352–360, 2005.
- [8] Mouna Elloumi, Riadh Dhaou, Benoit Escrig, Hanen Idoudi, and Leila Azouz Saidane. Monitoring road traffic with a uav-based system. In *2018 IEEE wireless communications and networking conference (WCNC)*, pages 1–6. IEEE, 2018.
- [9] Vinay Chamola, Pavan Kotesw, Aayush Agarwal, Navneet Gupta, Mohsen Guizani, et al. A comprehensive review of unmanned aerial vehicle attacks and neutralization techniques. *Ad hoc networks*, 111:102324, 2021.
- [10] DJI - Mini 2, Scheda Tecnica. <https://www.dji.com/it/mini-2/specs>. Data consultazione: 30/01/2023.
- [11] Aero System West - Heavy Lift Multirotor - Quadcopter. <https://www.aerosystemswest.com/product-page/heavy-lift-multirotor-quadcopter>. Data consultazione: 30/01/2023.

- [12] DJI - Mavic Air 2. <https://www.dji.com/it/mavic-air-2>. Data consultazione: 29/01/2023.
- [13] MathWorks - MATLAB. <https://it.mathworks.com/products/matlab.html>. Data consultazione: 30/01/2023.
- [14] MathWorks - Simulink. <https://it.mathworks.com/products/simulink.html>. Data consultazione: 30/01/2023.
- [15] mRo PixRacer R15. <https://store.mrobotics.io/product-p/auav-pxracer-r15-mr.htm>. Data consultazione: 30/01/2023.
- [16] Flying Tech - Pixracer R15 Mini Flight Controller with Wi-Fi Module. <https://www.flyingtech.co.uk/electronics/mini-px4-pixracer-r15-flight-controller-autopilot-wifi>. Data consultazione: 30/01/2023.
- [17] Sherif I Abdelmaksoud, Musa Mailah, and Ayman M Abdallah. Control strategies and novel techniques for autonomous rotorcraft unmanned aerial vehicles: A review. *IEEE Access*, 8:195142–195169, 2020.
- [18] Viswanadhapalli Praveen, Anju S Pillai, et al. Modeling and simulation of quadcopter using pid controller. *International Journal of Control Theory and Applications*, 9(15):7151–7158, 2016.
- [19] Pengcheng Wang, Zhihong Man, Zhenwei Cao, Jinchuan Zheng, and Yong Zhao. Dynamics modelling and linear control of quadcopter. In *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pages 498–503. IEEE, 2016.
- [20] Sabir Abdelhay and Alia Zakriti. Modeling of a quadcopter trajectory tracking system using pid controller. *Procedia Manufacturing*, 32:564–571, 2019.
- [21] Emmanuel Okyere, Amar Bousbaine, Gwangtim T Poyi, Ajay K Joseph, and Jose M Andrade. Lqr controller design for quad-rotor helicopters. *The Journal of Engineering*, 2019(17):4003–4007, 2019.
- [22] Faraz Ahmad, Pushpendra Kumar, Anamika Bhandari, and Pravin P Patil. Simulation of the quadcopter dynamics with lqr based control. *Materials Today: Proceedings*, 24:326–332, 2020.
- [23] Gembong Edhi Setyawan, Wijaya Kurniawan, and Amroy Casro Lumban Gaol. Linear quadratic regulator controller (lqr) for ar. drone's safe landing. In *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, pages 228–233. IEEE, 2019.
- [24] Parrot AR.Drone. https://www.parrot.com/assets/s3fs-public/2021-09/ar-drone_user-guide_uk.pdf. Data consultazione: 31/01/2023.
- [25] Eduardo Gamaliel Hernández-Martínez, Guillermo Fernandez-Anaya, Enrique D Ferreira, José-Job Flores-Godoy, and Alejandro Lopez-Gonzalez. Trajectory tracking of a quadcopter uav with optimal translational control. *IFAC-PapersOnLine*, 48(19):226–231, 2015.
- [26] Lucas M Argentim, Willian C Rezende, Paulo E Santos, and Renato A Aguiar. Pid, lqr and lqr-pid on a quadcopter platform. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6. IEEE, 2013.

- [27] Daewon Lee, H Jin Kim, and Shankar Sastry. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of control, Automation and systems*, 7(3):419–428, 2009.
- [28] Frede Blaabjerg. *Control of Power Electronic Converters and Systems: Volume 2*, volume 2. Academic Press, 2018.
- [29] Zhao Shulong, An Honglei, Zhang Daibing, and Shen Lincheng. A new feedback linearization lqr control for attitude of quadrotor. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1593–1597. IEEE, 2014.
- [30] Anastasia Razinkova, Igor Gaponov, and Hyun-Chan Cho. Adaptive control over quadcopter uav under disturbances. In *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, pages 386–390. IEEE, 2014.
- [31] Bara J Emran, Jorge Dias, Lakmal Seneviratne, and Guowei Cai. Robust adaptive control design for quadcopter payload add and drop applications. In *2015 34th Chinese Control Conference (CCC)*, pages 3252–3257. IEEE, 2015.
- [32] Kadda Meguenni Zemalache, Lotfi Beji, and H Marref. Control of an under-actuated system: application a four rotors rotorcraft. In *2005 IEEE International Conference on Robotics and Biomimetics-ROBIO*, pages 404–409. IEEE, 2005.
- [33] Tarek Madani and Abdelaziz Benallegue. Backstepping control for a quadrotor helicopter. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3255–3260. IEEE, 2006.
- [34] Salman Bari, Syeda Shabih Zehra Hamdani, Hamza Ullah Khan, Mutte ur Rehman, and Haroon Khan. Artificial neural network based self-tuned pid controller for flight control of quadcopter. In *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–5. IEEE, 2019.
- [35] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.
- [36] PX4 User Guide - Airframes Reference. https://docs.px4.io/main/en/airframes/airframe_reference.html. Data consultazione: 10/01/2023.
- [37] National Aeronautics and Space Administration. Advanced Air Mobility: The Science Behind Quadcopters. https://www.nasa.gov/sites/default/files/atoms/files/aam-science-behind-quadcopters-reader-student-guide_0.pdf, 2020. Data consultazione: 10/01/2023.
- [38] Alessandro Catania. TERZO PRINCIPIO DELLA DINAMICA: AZIONE E REAZIONE. <https://www.youmath.it/lezioni/fisica/dinamica/2965-terza-legge-di-newton.html>.
- [39] Lesics. Drones — The complete flight dynamics. <https://youtu.be/C0KBu2ihp-s?t=129>, 2020.
- [40] PX4 User Guide - Flight Controller/Sensor Orientation. https://docs.px4.io/main/en/config/flight_controller_orientation.html. Data consultazione: 15/01/2023.

- [41] Marilena Vendittelli. Modeling and control of multi-rotor UAVs. <http://www.diag.uniroma1.it/~venditt/eir/#Material>. Data consultazione: 30/12/2022.
- [42] MathWorks - Linearize Nonlinear Models. <https://it.mathworks.com/help/slcontrol/ug/linearizing-nonlinear-models.html>. Data consultazione: 16/01/2023.
- [43] PX4 User Guide - PX4 Autopilot User Guide. <https://docs.px4.io/main/en/>. Data consultazione: 26/12/2022.
- [44] PX4 User Guide - Vehicle Selection. https://docs.px4.io/main/en/getting_started/frame_selection.html. Data consultazione: 03/01/2023.
- [45] PX4 User Guide - PX4 Flight Modes Overview. https://docs.px4.io/main/en/getting_started/flight_modes.html. Data consultazione: 04/01/2023.
- [46] PX4 User Guide - Flight Controller (Autopilot) Hardware. https://docs.px4.io/main/en/flight_controller/. Data consultazione: 26/12/2022.
- [47] Pixhawk - The open standard for drone hardware. <https://pixhawk.org/>. Data consultazione: 03/01/2023.
- [48] PX4 User Guide - Pixhawk 4. [https://docs.px4.io/main/en/flight_controller/pixhawk4.html/](https://docs.px4.io/main/en/flight_controller/pixhawk4.html). Data consultazione: 02/01/2023.
- [49] PX4 User Guide - Pixhawk 4 Mini. https://docs.px4.io/main/en/flight_controller/pixhawk4_mini.html. Data consultazione: 03/01/2023.
- [50] PX4 User Guide - Pixhawk 3 Pro. https://docs.px4.io/main/en/flight_controller/pixhawk3_pro.html. Data consultazione: 03/01/2023.
- [51] PX4 User Guide - Sensors. https://docs.px4.io/main/en/getting_started/sensor_selection.html. Data consultazione: 26/12/2022.
- [52] QGroundControl. <http://qgroundcontrol.com/>. Data consultazione: 26/12/2022.
- [53] QGroundControl User Guide. <https://docs.qgroundcontrol.com/master/en/>. Data consultazione: 02/01/2023.
- [54] MAVLink Developer Guide. <https://mavlink.io/en/>. Data consultazione: 26/12/2022.
- [55] PX4 User Guide - MAVLink Messaging. <https://docs.px4.io/main/en/middleware/mavlink.html>. Data consultazione: 04/01/2023.
- [56] The Reactive Manifesto. <https://www.reactivemanifesto.org/>. Data consultazione: 04/01/2023.
- [57] PX4 User Guide - PX4 Architectural Overview. <https://docs.px4.io/main/en/concept/architecture.html>. Data consultazione: 26/12/2022.
- [58] PX4 User Guide - uORB Publication/Subscription Graph. https://docs.px4.io/main/en/middleware/uorb_graph.html. Data consultazione: 09/01/2023.
- [59] PX4 User Guide - Using the ECL EKF. https://docs.px4.io/main/en/advanced_config/tuning_the_ecl_ekf.html. Data consultazione: 04/01/2023.

- [60] PX4 User Guide - Controller Diagrams. https://docs.px4.io/main/en/flight_stack/controller_diagrams.html. Data consultazione: 26/12/2022.
- [61] LEARN.PARALLAX.COM - Understanding Each Control Channel. <https://learn.parallax.com/tutorials/robot/elev-8/how-fly-multirotor-suav/understanding-each-control-channel>. Data consultazione: 06/01/2023.
- [62] PX4 User Guide - Simulation. <https://docs.px4.io/main/en/simulation/>. Data consultazione: 06/01/2023.
- [63] MathWorks - UAV Toolbox Support Package for PX4 Autopilots. https://it.mathworks.com/help/supportpkg/px4/index.html?s_tid=CRUX_lftnav. Data consultazione: 26/12/2022.
- [64] MathWorks - Embedded Coder. <https://it.mathworks.com/products/embedded-coder.html>. Data consultazione: 08/01/2023.
- [65] MathWorks - Integration with General PX4 Architecture. <https://it.mathworks.com/help/supportpkg/px4/ug/px4-capabilities-integration.html>. Data consultazione: 03/01/2023.
- [66] MathWorks - Deploy PX4 on Host Computer with PX4 Host Target (PX4 Software-In-The-Loop or SITL). https://it.mathworks.com/help/supportpkg/px4/setup-and-configuration_mw_3095a711-0234-45ed-9b75-77488e8e577d.html. Data consultazione: 03/01/2023.
- [67] MathWorks - Hardware-in-The-Loop Simulation (HITL) with PX4. https://it.mathworks.com/help/supportpkg/px4/setup-and-configuration_mw_06fa168c-661f-4e54-b3b0-197c970f0ac9.html?s_tid=CRUX_lftnav. Data consultazione: 03/01/2023.
- [68] PX4 User Guide - jMAVSIM with SITL. <https://docs.px4.io/main/en/simulation/jmavsim.html>. Data consultazione: 03/01/2023.
- [69] PX4 User Guide - Simulation-In-Hardware (SIH). <https://docs.px4.io/main/en/simulation/simulation-in-hardware.html>. Data consultazione: 03/01/2023.
- [70] MathWorks - Develop Algorithms & Deploy on PX4 Autopilot. <https://it.mathworks.com/help/supportpkg/px4/develop-algorithm-deploy.html>. Data consultazione: 03/02/2023.
- [71] MathWorks - MAVLink Support. https://it.mathworks.com/help/uav/mavlink-support.html?s_tid=CRUX_lftnav. Data consultazione: 03/02/2023.
- [72] MathWorks - Monitor and Tune PX4 Host Target Flight Controller with Simulink-Based Plant Model. <https://it.mathworks.com/help/supportpkg/px4/ref/simulator-plant-model-example.html>. Data consultazione: 03/02/2023.
- [73] Jun Li and Yuntang Li. Dynamic analysis and pid control for a quadrotor. In *2011 IEEE International Conference on Mechatronics and Automation*, pages 573–578. IEEE, 2011.

- [74] MAVLINK Common Message Set. <https://mavlink.io/en/messages/common.html>. Data consultazione: 04/02/2023.
- [75] The animate package. <https://texblog.org/2018/03/05/the-animate-package/>. Data consultazione: 10/02/2023.
- [76] PX4 User Guide - Using Vision or Motion Capture Systems for Position Estimation. https://docs.px4.io/main/en/ros/external_position_estimation.html. Data consultazione: 10/02/2023.

Ringraziamenti

Paolo Leopardi