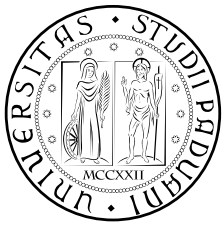


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

Studio e implementazione di algoritmi di localizzazione per UAVs basati su tecniche di Visual Inertial Odometry

Relatore: Dr. Giulia Michieletto

Correlatore: Dr. Stefano Michieletto

Correlatore: Dr. Massimiliano Bertoni

Laureando: Luca Pescante
2016585

ANNO ACCADEMICO: 2021-22

SOMMARIO

Obiettivo di questa tesi è consentire a UAVs la navigazione in ambienti indoor *GPS-denied*, al fine ultimo di poter interagire con l'ambiente circostante. In primo luogo si è analizzato lo stato dell'arte e il funzionamento di sistemi di localizzazione per UAVs, basati su tecniche di *visual inertial odometry*. In secondo luogo si sono implementati e affinati differenti algoritmi in grado di ottenere una maggiore accuratezza nella localizzazione del drone. L'intero sistema è stato validato con prove sperimentali per mezzo di due multirotori differenti.

"It wasn't luck that made them fly; it was hard work and common sense." I'm wondering what all of us could do if we had faith in our ideas and put all our heart and mind and energy into them.

David McCullough, The Wright Brothers

RINGRAZIAMENTI

Il seguente lavoro di tesi è stato al contempo motivante ed appassionante. Non sarebbe stato possibile senza il costante aiuto delle tre "M": Giulia Michieletto, Stefano Michieletto e Massimiliano Bertoni, rispettivamente relatrice e correlatori, sempre disponibili a rispondere alle mie molteplici domande. Un ringraziamento particolare a Giulia Michieletto, sempre propositiva, anche nella correzione delle prime sgangherate bozze dell'elaborato. Il supporto di Stefano Michieletto è stato fondamentale per il lato *software* del sistema, mentre Massimiliano Bertoni è stato sempre presente in tutte le fasi dell'implementazione, condividendo numerosi successi e formativi insuccessi. Un ringraziamento anche ai *meme* di *PlotJuggler*, indispensabili per tener alto il morale durante l'analisi statistica.

L'elaborato non sarebbe giunto alla conclusione in tempo senza il supporto di Arianna, compagna di avventure dall'oramai distante 2016, che mi ha sostenuto nei momenti più critici. Un grazie ai miei genitori, che, sebbene con modalità diametralmente opposte, mi hanno sempre stimolato al raggiungimento della laurea magistrale. Un sentito ringraziamento anche agli amici, con i quali ho condiviso momenti indimenticabili in questi anni, e ai compagni e alle compagne di corso: senza il loro supporto non starei scrivendo ora questo paragrafo.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

1 INTRODUZIONE

Il termine "drone", propriamente «fuco, ronzio», è oramai entrato nell'uso comune per indicare velivoli privi di pilota grazie all'importante diffusione degli stessi, sia in ambito consumistico che in ambito industriale. Nel settore tecnico-scientifico si predilige utilizzare delle sigle più specifiche rispetto al termine "drone". Si elencano i principali acronimi utilizzati nel settore:

- **APR** - acronimo per **A**eromobile a **P**ilotaggio **R**emoto con il quale si identifica il velivolo radiocomandato;
- **SAPR** - acronimo per **S**istema **A**eromobile a **P**ilotaggio **R**emoto, identificante l'intero sistema comprensivo non solo del velivolo ma anche di radiocomando, stazione di terra ed eventuali ulteriori componenti;
- **UAV** - acronimo inglese per ***U**nmanned **A**erial **V**ehicle*, veicolo aereo senza pilota;
- **UAS** - acronimo inglese per ***U**nmanned **A**erial **S**ystem*, come per SAPR si identifica non solo l'aeromobile ma l'intero sistema di volo.

Nel seguente elaborato si prediligerà l'utilizzo dell'acronimo **UAV**.

Già nel celeberrimo romanzo distopico di George Orwell si ipotizzava l'uso di droni per la sorveglianza, mentre al giorno d'oggi le applicazioni UAV sono molteplici e trasversali, come riportato in sezione 1.1. La crescita continua nel settore è principalmente trainata da velivoli atti a operare *outdoor*, sfruttando sistemi di localizzazione satellitari GNSS (*Global Navigation Satellite System*), in grado di determinare la posizione del drone nello spazio

Il presente elaborato si focalizza sull'implementazione di un sistema di localizzazione alternativo che permette di operare con un UAV anche in scenari indoor.

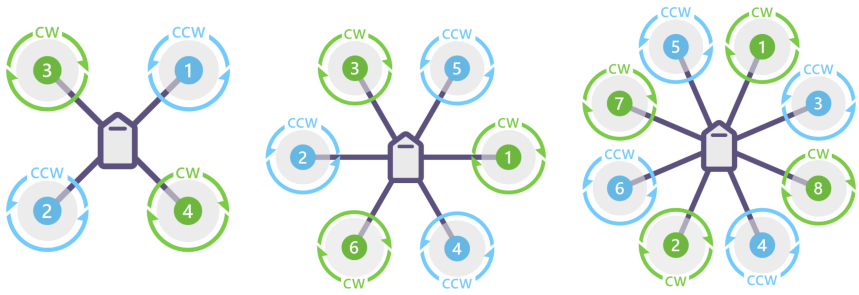
In sezione 1.1 si presentano le applicazioni principali degli UAVs, dopo una breve esposizione sulle varie tipologie esistenti, in sezione 1.2 si analizzano le principali tecniche di localizzazione, evidenziandone pregi e difetti, mentre nelle sezioni 1.3 e 1.4 si espongono l'obiettivo e la struttura della tesi.

1.1 CLASSIFICAZIONE DI DRONI E PRINCIPALI APPLICAZIONI

Una prima classificazione degli UAVs può basarsi sulle caratteristiche aerodinamiche dell'aeromobile [1]: si distinguono i **velivoli ad**

(a) Velivoli ad ala fissa (b) Velivoli ad ala rotante

Figura 1: Classificazione aerodinamica di UAVs [1].



(a) Quadricottero (b) Esacottero (c) Ottacottero

Figura 2: Principali configurazioni di multirotori.

ala fissa da quelli **ad ala rotante**. I primi sono caratterizzati dalla presenza di un profilo alare che genera portanza grazie alla velocità di avanzamento dell'aeromobile. Il controllo di questa tipologia di UAVs è garantito da parti mobili attuate presenti sulle ali. I secondi sono invece caratterizzati da uno o più rotori che producono la spinta appropriata per librarsi in aria. Con il termine rotore si definisce un elemento meccanico composto da più pale che, mediante rotazione, permette il pilotaggio, la propulsione e il sostentamento di un aeromobile. I velivoli ad ala rotante se presentano un rotore principale si definiscono elicotteri, mentre se composti da un numero maggiore di rotori, tipicamente quattro o più, si definiscono **multirotori**. Quest'ultimi possono essere classificati in ulteriori sottocategorie, ad esempio in base al numero di rotori/attuatori, come riportato in in Figura 2. Sono stati sviluppati anche sistemi ibridi, tipicamente caratterizzati sia dalla presenza di più rotori per le fasi di decollo e atterraggio verticali che dalla presenza di ali fisse per aumentare l'efficienza e garantire maggiori distanze di volo e autonomie, definiti in gergo *Vertical Take-Off and Landing* (VTOL).

La grande diffusione degli UAVs è avvenuta grazie ai velivoli ad ala rotante multirotore, che garantiscono un decollo verticale e la possibilità di stazionare in volo (manovra di *hovering*). Questa tipologia di piattaforma, composta solitamente da quattro o più rotori che costituiscono il sistema di attuazione, è caratterizzata da un design strutturale semplice e da un'elevata versatilità e manovrabilità. Tale diffusione è stata resa possibile dal recente avanzamento tecnologico in numerosi campi come la microelettronica, la microelettromeccanica e l'elettronica di potenza. [6].

I campi di utilizzo degli UAVs sono molteplici e disparati. Si presentano ora alcune tra le applicazioni più diffuse [1, 7]:

- **riprese aeree** - la maggioranza dei droni disponibili sul mercato ha come payload una camera, spesso stabilizzata, per effettuare riprese aeree. Grazie al costo contenuto degli UAVs multirotore, l'utilizzo di questi sistemi è non solo industriale ma anche hobbystico. Sono disponibili sul mercato droni di piccole dimensioni in grado di effettuare riprese di qualità molto elevata;
- **agricoltura di precisione** - gli UAVs possono fornire numerosi servizi all'agricoltura, dall'acquisizione di dati su coltivazioni tramite sensori multispettrali al cospargimento di apposite sostanze;
- **trasporto di oggetti** - i veicoli aerei autonomi costituiscono un'alternativa rapida, efficiente e priva di pilota per il trasporto di beni;
- **sorveglianza** - in Australia e in USA i droni sono già utilizzati per monitorare lunghi tratti costieri;
- **servizi internet** - appositi veicoli aerei autonomi ad alimentazione solare sono in grado di fornire una connessione a internet in luoghi remoti e difficilmente accessibili alle consuete infrastrutture.

1.2 LOCALIZZAZIONE DI UAVS

Con il termine localizzazione si intende l'operazione mediante la quale si individua la posizione di un qualunque oggetto nello spazio [8]. La localizzazione è essenziale per un drone per garantire il corretto funzionamento del controllore di posizione: senza una stima di posizione il velivolo risulta sia impossibilitato all'esecuzione di missioni automatiche che difficoltoso da pilotare manualmente, poiché non mantiene autonomamente la posizione.

Comunemente i velivoli sono equipaggiati con piattaforme inerziali, in lingua inglese *Inertial Measurement Unit* (IMU), sistemi di localizzazione satellitare e barometri. I dati forniti dai singoli sensori sono

soggetti a errore, dunque si implementano sistemi di stima dello stato per fornire misure più attendibili. Il metodo di stima dello stato solitamente implementato nei droni è *Extended Kalman Filter* (EKF), una versione del filtro di Kalman applicabile a sistemi non lineari [9], che permette di fondere i dati di più sensori, in inglese *data fusion*.

Generalmente i droni in commercio sono progettati per volare all'esterno, dunque la stima di posizione si basa su sistemi di localizzazione satellitare. In assenza di un valido segnale satellitare ricevuto, la stima di posizione del drone può risultare molto inaccurata, deteriorando le capacità di svolgere efficientemente missioni automatiche e aumentando le possibilità di incorrere in comportamenti dannosi e imprevisti. All'interno di edifici il segnale proveniente da sistemi GNSS non è affidabile, in quanto caratterizzato da un elevato errore. In scenari *indoor* risulta quindi necessario studiare nuove tecniche di localizzazione per operare con tali velivoli, soprattutto dal momento che in ambito industriale le possibili applicazioni sono numerose, come la gestione dell'inventario di magazzino, la logistica interna e la sorveglianza [10].

1.2.1 *Localizzazione indoor*

Comunicare con le reti GNSS risulta infattibile nella maggior parte dei contesti *indoor*, diventa dunque necessario individuare altre fonti per ottenere una corretta stima di posizione. In letteratura si trova un copioso numero di studi e soluzioni per ottenere tale stima, tra cui sistemi basati su radio frequenza (WiFi [11], Bluetooth [12], RFID [13]), su segnali acustici [14], sulla comunicazione a luce visibile [15] e sistemi di visione [16]. Un confronto sulle soluzioni citate è disponibile in Tabella 1, da cui si possono notare le elevate prestazioni in termini di accuratezza dei sistemi di visione. Quest'ultimi sono maggiormente utilizzati per la localizzazione indoor, proprio per l'elevata robustezza e accuratezza [17]: sistemi di *motion capture* basati su architetture di camere a infrarosso possono avere prestazioni di accuratezza elevatissime, fino ai $7\,\mu m$ [18], tuttavia il costo del sistema è molto elevato.

Un'ulteriore tipologia di sistema di visione per la localizzazione si basa su *fiducial markers*, oggetti posti nel campo visivo di una camera installata su un mezzo, usati come riferimento per localizzare il mezzo stesso. In letteratura si possono trovare soluzioni che implementano sistemi di questo tipo applicati agli UAVs [19, 20], tuttavia si focalizzano principalmente sul riconoscimento di singoli *markers* per effettuare manovre di atterraggio molto precise. I sistemi di localizzazione basati su *fiducial markers* sono caratterizzati da un costo minore se comparati a sistemi di *motion capture* basati su architetture di camere a infrarosso.

Sistema	Accuratezza	Costo
Wi-Fi	10 m	basso
Bluetooth	$2-5$ m	medio-basso
RFID	$1-5$ m	basso
Acustico	$0.1-1$ m	basso
Luce visibile	0.1 m	medio
Sistemi di visione	$10^{-5}-10^{-1}$ m	basso-alto

Tabella 1: Contronto tra sistemi di localizzazione indoor [4, 5].

1.3 SCOPO E INTENTO DI TESI

Il lavoro di tesi è stato svolto presso il C-square (Computer and Control Engineering) lab del Dipartimento di Tecnica e Gestione dei Sistemi Industriali (DTG), situato presso l'Elevator Innovation Hub.

Precedenti studi di ricerca e lavori di tesi hanno portato alla realizzazione di due multirotori: un quadrirotore QR01 e un esarotore HR01, riportati in Figura 3. Questi multirotori integrano un sistema di localizzazione basato su tecniche di *Visual Inertial Odometry* (VIO). In tale sistema si localizza il multirotore fondendo l'informazione di sensori inerziali e di odometria visuale, derivante da un sistema di visione basato su *fiducial markers*. Per quanto riguarda i *fiducial markers*, si è deciso di utilizzarne un elevato numero, disponendoli in una mappa. In figura 4 si riporta un'immagine dell'arena di volo, nella quale si può notare la mappa di *markers* poggiata sulla pavimentazione e il multirotore HR01.

Il sistema di localizzazione precedentemente implementato risultava già funzionante, si considerava però un solo *marker* tra quelli nel campo di visione della camera per stimare la posizione del multirotore. La stima così calcolata non si è rivelata sufficientemente accurata, soprattutto in ottica di interazione con l'ambiente circostante, presentando errori fino a 30 cm in valore assoluto. Allo stato dell'arte si trovano diverse soluzioni per ottenere una stima di posizione per velivoli a partire dalla visione di singoli *markers* [19, 20, 21, 22]. In letteratura, tuttavia, la problematica di ricavare una stima di posizione a partire da più *fiducial markers* nel campo di visione, anche di dimensioni o tipologia differente, non è affrontata.

Scopo di questa tesi è migliorare l'accuratezza della stima del sistema di *visual inertial odometry*, sfruttando tutta l'informazione reperibile dalla camera. In particolare, si vuole migliorare l'accuratezza andando ad analizzare e filtrare l'informazione reperita dall'identificazione di tutti i *markers* nel campo di visione della camera. Per raggiungere questo obiettivo si sono sviluppati e installati diversi algoritmi sull'apparato sperimentale a disposizione, così da mantenere

(a) QR01 (b) HR01

Figura 3: Quadrirotore QR01 ed esarotore HR01.

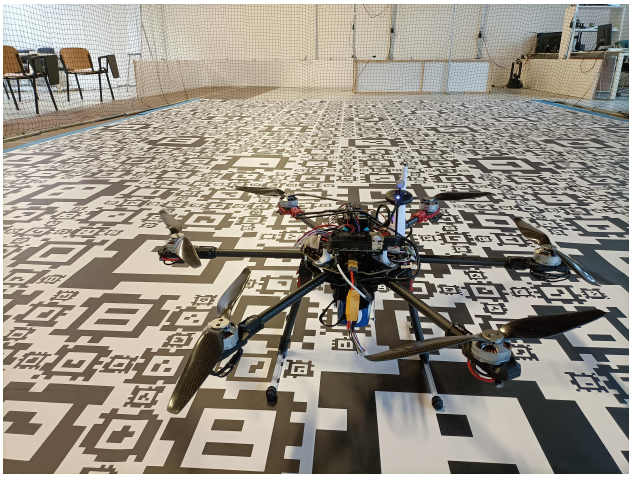


Figura 4: Arena di volo *indoor*, si noti la mappa di *fiducial markers*.

invariato l'hardware e validare i risultati ottenuti.

Il tutto si inserisce in un progetto di ricerca più ampio, volto all'interazione tra l'ambiente (elemento passivo), robot aerei e terrestri (elementi attivi con dinamica prevedibile) e l'uomo (elemento attivo con dinamica imprevedibile) in ambito industriale.

1.4 STRUTTURA DELL'ELABORATO

L'elaborato si compone di tre capitoli:

- nel capitolo 2 si approfondisce la *Visual Inertia Odometry* (VIO), sia da un punto di vista più generale che per applicazioni specifiche a UAV;
- nel capitolo 3 si tratta l'implementazione del sistema di visione, con particolare focus sugli algoritmi scritti per migliorare l'accuratezza di posizione;
- nel capitolo 4 si riportano i risultati derivanti da prove sperimentali atte alla validazione del sistema implementato;

2 VISUAL INERTIAL ODOMETRY

L'uso combinato di camere e sensori inerziali per la stima del moto tridimensionale ha ricevuto una considerevole attenzione nell'ambito della robotica: sia le *Inertial Measurement Unit* (IMU) che le camere sono sensori facilmente reperibili, economici e complementari per frequenze di lavoro e informazioni acquisibili [23].

Grazie all'importante progresso nel settore dei sistemi micro-elettromeccanici, sono disponibili IMU a 6 assi di ridotte dimensioni e capaci di operare a frequenze nell'ordine dei kHz, che misurano accelerazione lineare e velocità angolare. A partire dai dati forniti da sensori inerziali è possibile ricavare l'informazione di **posa** (posizione e orientamento), tuttavia le misure sono corrotte da rumore e distorsioni, che possono risultare in una stima della posa non affidabile. In particolare, la stima può presentare il fenomeno di *drifting*, letteralmente "andare alla deriva" [24]. D'altro canto anche le camere sono di ridotte dimensioni e tramite algoritmi di visione restituiscono una stima della posa esente da problematiche di *drifting*. Le camere sono però sensibili a fenomeni come il *motion blur*, derivante da rapidi movimenti, o la variazione di luminosità dell'ambiente e operano tipicamente a frequenze nell'ordine delle decine di Hz. La fusione delle informazioni fornite da IMU e camere tramite Extended Kalman Filter (EKF) può dunque fornire una stima della posa accurata e sufficientemente veloce per applicazioni *real-time* [25].

Con il termine *Visual Inertial Odometry* (VIO) si intende il processo di determinazione della posa di un oggetto a partire da misure inerziali e informazioni di posizione derivanti da metodi *image-based*. Tali informazioni devono essere unite, al fine di ottenere un'unica stima di posa. Tecnicamente questo processo di fusione delle informazioni provenienti da sensori differenti viene denominato *sensor fusion*.

In Sezione 2.1 si introducono dei modelli matematici per UAVs e per i principali sensori di un multirotore. In Sezione 2.2 viene presentata la *Visual Odometry*, dalle definizioni base alla storiografia, mentre in Sezione 2.3 si approfondisce la *Monocular Visual Odometry* e i principali *fiducial markers* utilizzati. In ultimo, si espone il funzionamento di EKF per *sensor fusion* in Sezione 2.4.

2.1 MODELLO DI MULTIROTORI E DI SENSORI

In primo luogo è importante comprendere i movimenti principali di un multirotore in configurazione *star-shaped*, così definito se presenta i rotori equamente distribuiti su una circonferenza, avente come

centro il Centro di Massa (CdM) del velivolo [26]. Nel seguente elaborato ci si concentrerà su multirotori *star-shaped* con rotori coplanari, in quanto gli UAVs di questa tipologia sono utilizzati per la validazione sperimentale del sistema implementato. In particolare, i rotori si definiscono coplanari se ruotano attorno ad assi di rotazione paralleli tra loro e perpendicolari al piano passante per i centri di massa dei rotori stessi. I movimenti di un multirotore *star-shaped* con rotori coplanari si definiscono con i termini inglesi *lift*, *roll*, *pitch* e *yaw*. Il sistema ha quindi quattro gradi di libertà, che rendono necessari almeno quattro rotori indipendenti. In Figura 5 sono rappresentati i movimenti principali [6], mettendo in evidenza quali rotori devono generare più portanza con delle frecce verdi e quali meno con delle frecce blu in un quadrirotore, al fine di eseguire il movimento indicato in didascalia. Appare poco intuitivo come la rotazione attorno all'asse z denominata *yaw* venga eseguita: per compensare il momento torcente che agisce sul quadricottero a causa della rotazione dei motori, quest'ultimi ruotano a due a due in direzione opposta, come evidenziato in Figura 2a. Andando dunque a diminuire la velocità di rotazione dei rotori che ruotano in senso antiorario e aumentando quella dei rotori che ruotano in senso orario, sul velivolo agisce un momento torcente che provoca una rotazione antioraria dello stesso.

Un ragionamento similare è applicabile anche a un esarotore: sebbene siano presenti sei rotori indipendenti, il sistema complessivo è comunque caratterizzato da quattro gradi di libertà. Dal punto di vista controllistico i droni *star-shaped* con rotori planari sono sistemi sottoattuati, poiché caratterizzati da quattro gradi di libertà a fronte dei sei presenti nello spazio.

Un multirotore si compone di tre principali sottosistemi fondamentali:

- sistema di attuazione - genera forze e coppie tramite i rotori affinché un UAV si libri in volo;
- sistema di *sensing e perception* - comprende principalmente i sensori atti a misurare posizione e orientamento di un UAV;
- sistema di controllo - a partire dalle informazioni del sistema di *sensing e perception* genera dei comandi per gli attuatori al fine di eseguire le traiettorie richieste.

Per quanto riguarda il sistema di attuazione, nella sottosezione seguente si descrive il modello dinamico di un generico multirotore, mentre per il sistema di *sensing e perception* si presenta un modello dei principali sensori *onboard* di interesse, IMU e camera.

2.1.1 *Modello di multirotori*

Per controllare e comprendere i velivoli multirotore è importante disporre di un modello matematico della dinamica degli stessi.

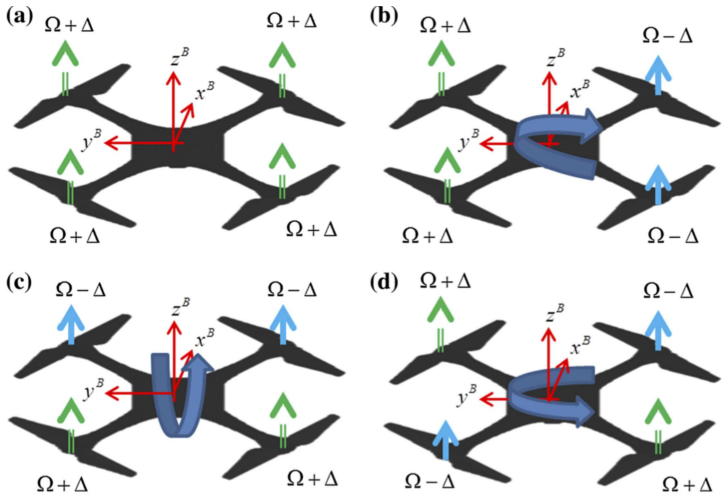


Figura 5: Movimenti base di un drone: a) lift, b) roll, c) pitch, d) yaw.

In primo luogo in aeronautica si definiscono diversi sistemi di riferimento tramite differenti notazioni. Tra le notazioni più usate ricordiamo North East Down (NED), Front Right Down (FRD), East North Up (ENU), Front Left Up (FLU). Queste notazioni rappresentano in ordine la direzione dell'asse x, y, z. Con *world frame* $\mathcal{F}_W$ si intende un sistema di riferimento globale fisso definito secondo la convenzione NED, identificato in Figura 6 dalla base $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. Con *body frame* $\mathcal{F}_B$ si intende invece un sistema di riferimento locale centrato sul centro di massa (CdM) del drone, definito secondo la convenzione FRD e identificato dalla base $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$. $\mathbf{p}_W$ è un vettore in $\mathbb{R}^3$ che identifica la posizione dell'origine del body frame $\mathcal{F}_B$ (coincidente con il CdM del drone) nel world frame $\mathcal{F}_W$, $\mathbf{R}_B$ è una matrice di rotazione appartenente al gruppo speciale ortogonale $SO(3)$ che descrive l'orientamento del body frame rispetto al world frame. La coppia $(\mathbf{p}_W, \mathbf{R}_B)$ è la posa del velivolo, insieme tra posizione e *attitude* (orientamento).

La rotazione di un rotore genera una forza e una coppia dipendente dal quadrato della velocità di rotazione dello stesso. La combinazione tra le coppie e le forze generate dai singoli rotori risultano in una forza e una coppia applicate al CdM, rispettivamente *control force* $\mathbf{F}_B^c$ e *control torque* $\boldsymbol{\tau}_B^c$, espressa in *body frame*. Il modello della dinamica di un generico drone *star-shaped* può essere ricavato mediante le equazioni di Newton-Eulero [17]:

$$m\ddot{\mathbf{p}}_W = -mg\mathbf{e}_3 + \mathbf{R}_B\mathbf{F}_B^c \tag{1a}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}}_B = -\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \boldsymbol{\tau}_B^c \tag{1b}$$

dove $m, g \in \mathbb{R}^+$ sono rispettivamente la massa del velivolo e l'accelerazione di gravità, $\mathbf{J} \in \mathbb{R}^{3x3}$ è il momento di inerzia del drone calcolato rispetto $\mathcal{F}_B$ e $\boldsymbol{\omega}_B \in \mathbb{R}^3$ è la velocità angolare, espressa in *body frame*.

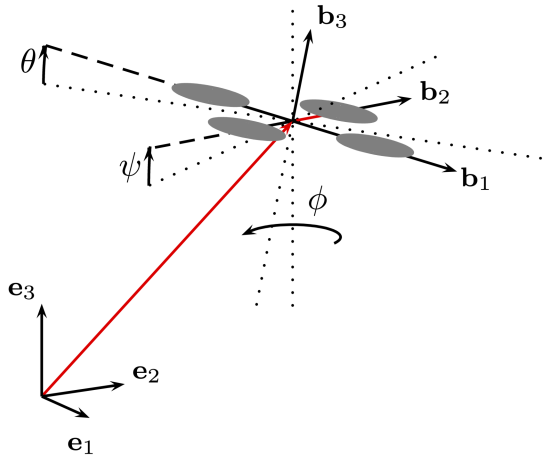


Figura 6: Sistemi di riferimento: *world frame* e *body frame*.

2.1.2 *Modello delle misure di IMU*

Con piattaforma inerziale, in inglese *Inertial Measurement Unit* (IMU), si definisce un sistema basato su sensori inerziali che permettono un monitoraggio della dinamica di un oggetto in movimento. In base alla quantità di sensori inerziali presenti in una piattaforma inerziale si suddividono in:

- **IMU a 3 assi** - composte da un giroscopio a 3 assi, che restituisce la velocità angolare per ciascun asse del *body frame*. I multirotori che comprendono questo sistema sono comunemente chiamati *3-axis drone* e possono essere comandati solamente fornendo un riferimento di velocità angolare tramite un radiocomando;
- **IMU a 6 assi** - comprendono anche un accelerometro a 3 assi, che fornisce misure di accelerazione lineare lungo i tre assi di riferimento del *body frame*. Con *6-axis drone* si intende un multirotore dotato di un'unità inerziale di tale tipologia, che permette di essere comandato fornendo un riferimento di posizione angolare per roll e pitch;
- **IMU a 9 assi** - comprendono anche un magnetometro a 3 assi, che permette di fornire un riferimento di posizione angolare anche per lo *yaw*. I multirotori commerciali solitamente implementano unità inerziali di questa tipologia, così da garantire la massima stabilità del velivolo.

In Figura 7 è riportata l'unità inerziale a sei assi contenuta nell'hardware PIXHAWK, presente nell'apparato sperimentale in uso (Sezione 4.1). Si notino le dimensioni estremamente ridotte del componente.

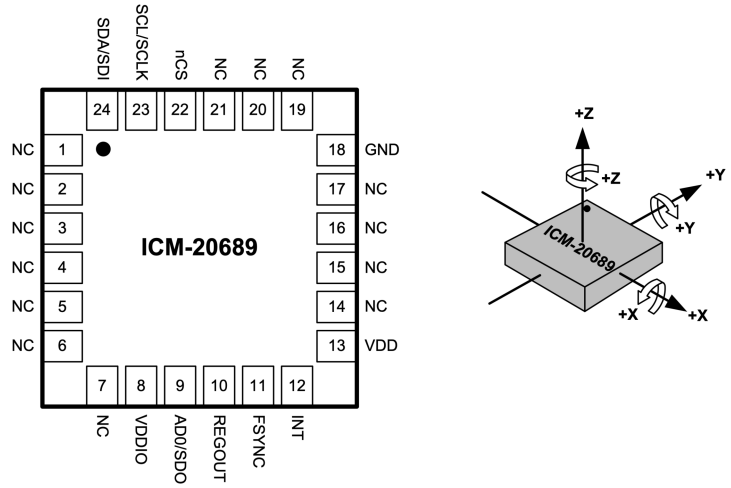


Figura 7: ICM-20689 IMU a 6 assi di dimensioni $4\,\mathrm{mm} \times 4\,\mathrm{mm} \times 0.9\,\mathrm{mm}$.

Le misure rilevate da giroscopi e accelerometri possono essere modellizzate rispettivamente come:

$$\boldsymbol{\omega}_{m,B} = \boldsymbol{\omega}_B + \mathbf{b}_{g,B} + \mathbf{r}_{g,B} \tag{2a}$$

$$\mathbf{a}_{m,B} = \mathbf{R}^T(\ddot{\mathbf{x}}_B - g\mathbf{e}_3) + \mathbf{b}_{a,B} + \mathbf{r}_{a,B} \tag{2b}$$

dove $\boldsymbol{\omega}_{m,B}, \mathbf{a}_{m,B} \in \mathbb{R}^3$ sono velocità e accelerazione misurate in *body frame*, $\boldsymbol{\omega}_B, \ddot{\mathbf{x}}_B \in \mathbb{R}^3$ sono velocità e accelerazione reali in *body frame*, $\mathbf{b}_{g,B}, \mathbf{b}_{a,B} \in \mathbb{R}^3$ sono i *bias* in *body frame*, $\mathbf{r}_{g,B}, \mathbf{r}_{a,B} \in \mathbb{R}^3$ sono sequenze di rumore bianco gaussiano a media nulla in *body frame*, $g \in \mathbb{R}$ è l'accelerazione gravitazionale.

2.2 VISUAL ODOMETRY

Una misura di posizione esente da problematiche di *drifting* può essere restituita da un sistema di visione. Di seguito si è implementato un sistema di *Visual Odometry* (VO), al fine di stimare la posa di UAVs. Per VO si intende il processo di stima della posa di un agente, come veicoli, velivoli, robot, a partire dall'informazione data da uno o più sensori ottici solidali a esso. Il termine è stato coniato da Nister et al. nel 2004 ed è stato scelto per l'affinità con *Wheel Odometry*, la stima del movimento di un veicolo data dall'integrazione del numero di rotazioni di una ruota [28].

2.2.1 *Storia della Visual Odometry*

La problematica della stima della posa di un veicolo è stata affrontata a partire dall'inizio del 1980, ed è interessante osservare come molte tra le pubblicazioni preliminari siano a opera della NASA per rover planetari. In particolare è da citare il lavoro di Moravec, una delle prime implementazioni di VO, sperimentata su un rover equipaggiato di

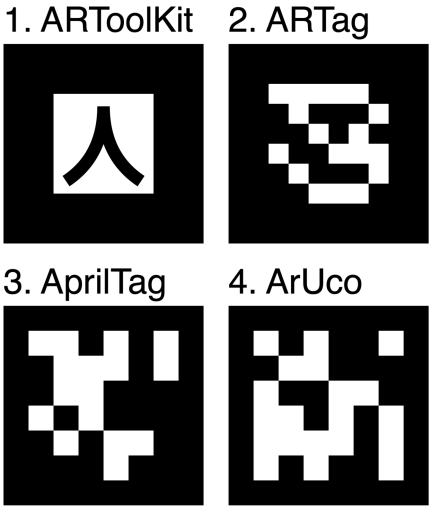


Figura 8: Esempi di *fiducial markers*.

una singola camera posta su una guida lineare. Il lavoro di Moravec appartiene alla categoria degli algoritmi di *stereo VO*, che ricavano il movimento di un veicolo dalla triangolazione di immagini scattate da differenti prospettive tramite più camere o una singola camera mobile. Al contrario algoritmi di *monocular VO* ricavano la posa tridimensionale a partire dall'informazione fornita da un singolo sensore ottico bidimensionale, rendendo quindi necessaria una misura diretta di almeno un elemento nell'ambiente. La ricerca in queste due macrocategorie di algoritmi di *VO* ha progredito in maniera disaccoppiata [28] e per la seguente trattazione la problematica è stata affrontata mediante algoritmi di *monocular VO*: la presenza di una sola camera solidale al velivolo rende il sistema più compatto ed economico, così da poter essere applicato anche a droni di piccole dimensioni.

2.2.2 *Modello delle misure di posizione da visual odometry*

La misura di posizione ricavata da sistemi di *visual odometry* può essere modellizzata tramite la seguente equazione, a prescindere dalla tipologia di sistema implementato:

$$\mathbf{p}_{m,W} = \mathbf{p}_W + \mathbf{b}_{vo,W} + \mathbf{r}_{vo,W} \tag{3}$$

dove $\mathbf{p}_{m,W} \in \mathbb{R}^3$ è la posizione misurata del *body frame*, $\mathbf{p}_W$ è la posizione reale del *body frame*, $\mathbf{b}_{vo,W} \in \mathbb{R}^3$ sono i *bias* e $\mathbf{r}_{vo,W} \in \mathbb{R}^3$ è una sequenza di rumore bianco gaussiano a media nulla, il tutto espresso in *world frame*.

2.3 FIDUCIAL MARKERS E MONOCULAR VISUAL ODOMETRY

Sistemi di *Monocular Visual Odometry* estraggono informazioni di posa a partire da immagini, tuttavia l'informazione bidimensionale non

è sufficiente a ricostruire un movimento tridimensionale nello spazio [28]. È dunque necessario avere a priori un'informazione sull'ambiente circostante, come la misura di un oggetto posto nel campo di visione della camera. Tale oggetto viene definito *fiducial marker*, ovvero un corpo posizionato nel campo di visione della camera usato come riferimento.

Per applicare algoritmi di *Monocular Visual Odometry* agli UAVs, si è deciso di utilizzare più *tags* come *fiducial markers*. Per *tag* si intende un *marker* caratterizzato da un pattern visivo altamente distinguibile che codifica un identificativo (ID). Esempi di *tags* sono AprilTag, ARTag e CLTags, riportati in Figura 8. Il riconoscimento di tali *tags* si basa su due fasi [30]:

1. ***unique feature detection*** - inizialmente si ricerca nell'immagine la "caratteristica unica", ovvero la forma esterna considerando la deformazione prospettica (quadrilatero);

2. ***verification/identification*** - successivamente si controlla l'interno del quadrilatero per verificare sia effettivamente un *marker* e in caso di esito positivo si identifica l'ID dello stesso.

Sia il design che gli algoritmi di *detection* e *identification* sono differenti per ogni tipologia di *tag*, di conseguenza in diverse situazioni si riscontrano diverse prestazioni. Nelle sottosezioni successive si riportano le principali caratteristiche delle tipologie di *tags* analizzate e un confronto tra le stesse.

2.3.1 *ARTag*

Il sistema ARTag contiene 2002 *markers* di forma quadrata. Per la rilevazione l'algoritmo ricerca il bordo del quadrato, individuando i pixel al margine e formando dei segmenti, raggruppati poi in quadrilateri. L'immagine interna è formata da una matrice 6x6 di celle bianche o nere, codificanti 36 bit di informazione. I primi 10 bit identificano l'ID del *marker*, mentre i restanti 26 bit servono per rilevare unicamente l'orientazione e correggere eventuali errori [31].

L'implementazione degli algoritmi di *detection* e *identification* di ARTag si basano su ARToolKit, una libreria *open-source* per la localizzazione rilasciata nel 2004 che permette di calcolare la trasformazione tra camera e *marker* identificato. ARTag implementa un'elaborazione dei dati più complessa e performante, di conseguenza si riscontrano prestazioni migliori al variare della luminosità dell'ambiente e per quanto concerne l'affidabilità, se comparato ad ARToolKit. Come si evince dal nome, tale libreria è stata concepita per applicazioni di *Augmented Reality* (AR).

2.3.2 *AprilTag*

Il sistema AprilTag è stato proposto da Olson nel 2011 ed è utilizzato in vari contesti, tra i quali la calibrazione di camere, la robotica e la realtà aumentata [31]. I *tags* sono di forma quadrata, e possono codificare fino a 4164 ID differenti.

Il pacchetto per la *detection* e l'*identification* è completamente *open-source* e ben documentato. In base alla famiglia di AprilTag scelta, l'informazione codificata varia da 4 a 36 bit. Il processo di riconoscimento si compone di diverse fasi, la prima di ricerca dei segmenti lineari utilizza un approccio simile a ARTag, mentre per la *detection* è stato implementato un algoritmo ricorsivo che riconosce ciascun lato del quadrato. Per quanto riguarda l'identificazione, all'interno del quadrato l'informazione è codificata da un sistema *lexicode* caratterizzato da due parametri: il numero di bit per l'identificativo e la minima distanza di Hamming tra i *markers* [31]. Quest'ultima definisce il numero massimo di bit che possono essere corretti senza produrre un errore di identificazione tra *tags*. Nel sistema AprilTag si può scegliere tra diverse famiglie di *tags* che differiscono proprio per i due citati parametri: per esempio la famiglia "Tag41h12" codifica 41 bit per l'ID con una minima distanza di Hamming di 12 bit tra due *markers*.

2.3.3 *ArUco*

I *markers* Aruco sono anch'essi di forma quadrata e codificano un ID. Il sistema si differenzia nel proporre un metodo per la generazione di dizionari di *markers* configurabili di dimensioni e numero arbitrari, invece che basarsi su una sequenza predefinita di *tags*. Di conseguenza, in base agli ID univoci necessari per l'applicazione è possibile generare un dizionario ad hoc. Inoltre viene proposto anche un metodo per la correzione di errori basato sul dizionario ottenuto.

La libreria ArUco per la *detection* e l'*identification* è disponibile gratuitamente con licenza *open-source*. In Figura 8 sono riportati degli esempi di *markers* di differenti dimensioni [33].

2.3.4 *Confronto tra markers*

In letteratura si trovano diversi confronti sperimentali tra le prestazioni dei *markers* precedentemente citati, in particolare tra AprilTag e ArUco [34, 31, 35, 36]. Da test sperimentali in ambiente reale si nota come l'utilizzo di AprilTag consente di ottenere risultati migliori in termini di accuratezza di posizionamento e rotazione rispetto ad ArUco [34]. Anche per quanto riguarda il tasso di rilevamento il pacchetto AprilTag è più performante comparato ad ArUco [34].

Mediante prove in ambiente simulativo si evince come ArUco presenti una maggiore reiezione a un disturbo Gaussiano di bassa entità,

Intervalli di ID	Lato [cm]	Dimensione
$0 \rightarrow 99$	46	XL
$100 \rightarrow 399$	23	L
$400 \rightarrow 999$	11.5	M
$100 \rightarrow \infty$	5.75	S

Tabella 2: AprilTag presenti nella mappa.

iniettato nelle immagini. Tuttavia, per un disturbo maggiore del 10% sia AprilTag che ArUco non forniscono risultati attendibili [35]. Ulteriori test a differenti condizioni di distanza e luminosità dimostrano come il sistema AprilTag presenti una velocità di rilevamento ed elaborazione maggiore rispetto ad ArUco. [36] Un vantaggio del sistema ArUco è la possibilità di generare dizionari ad hoc, come accennato precedentemente.

Per il sistema di *monocular visual odometry* implementato si è scelto di usare AprilTag, vista la rapidità nella decodifica, le buone prestazioni in condizioni di scarsa luminosità e l'adeguato numero di *markers* a disposizione [3].

2.3.5 *Mappa di fiducial markers*

La strategia di localizzazione discussa in questa tesi si basa sull'utilizzo di una mappa di *AprilTag*. Come descritto nel lavoro di Segattini [3], tale mappa si contraddistingue per l'ampio numero di *tags* presenti, anche di differenti dimensioni. L'uso di una mappa rispetto a *tags* singoli è motivato dal fatto che si vuole utilizzare la VO per navigare nell'ambiente, anche a differenti quote, e non eseguire solamente una manovra di atterraggio preciso in determinate posizioni, come trattato in letteratura [19, 20]. Nel caso la mappa risulti troppo ingombrante a terra è possibile collocarla sul soffitto di un ambiente industriale.

In particolare, è stato scelto di inserire AprilTag di quattro diverse dimensioni (S, M, L, XL) nella mappa, così che nel campo di visione del drone sia sempre presente una dimensione di *markers* a una definizione accettabile per un range di altezze consono a un ambiente industriale. Ogni AprilTag inserito nella mappa deve possedere un ID univoco, così da poterlo associare a una precisa posizione nella mappa, di cui si assume nota l'origine. Si è dunque deciso di inserire l'informazione della dimensione di un *marker* nell'identificativo, suddividendo le taglie in predeterminati intervalli di ID come riportato in Tabella 2. Per quanto riguarda la disposizione delle varie dimensioni di *tags*, si è scelto uno schema base, riportato in Figura 9, che comprende *markers* di tutte e quattro le dimensioni minimizzando lo

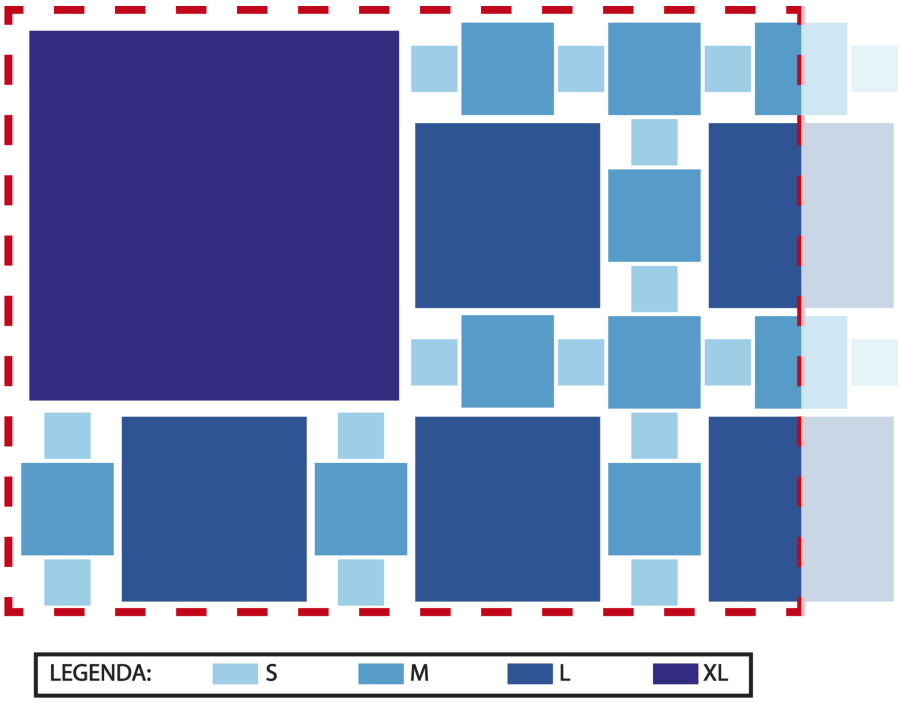


Figura 9: Schema base per la generazione della mappa.

spazio vuoto. Ripetendo il pattern generato è possibile formare una mappa dalle dimensioni desiderate. In Figura 10 è riportata una mappa di dimensioni $4.9\,\mathrm{m} \times 3.45\,\mathrm{m}$, presente nel laboratorio di Padova del gruppo di ricerca SPARCS (*SPace Aerial and gRound Control System*). L'origine della mappa è stata posizionata nell'angolo in basso a sinistra, con asse x verticale orientato verso l'alto, asse y orizzontale orientata verso destra e asse z entrante nel foglio, seguendo dunque la convenzione FRD.

2.4 SENSOR FUSION MEDIANTE EKF

Come accennato in Sezione 1.2, affinché un multirotore sia controllabile in posizione è necessaria un'accurata stima della posa, poiché si stanno considerando sistemi sottoattuati. L'informazione data da diversi sensori dev'essere fusa per ottenere una stima della posa più affidabile e l'approccio più utilizzato è mediante Extended Kalman Filter (EKF) [37]. La stima della posa mediante *visual inertial odometry* consiste dunque, per il sistema implementato, nella fusione con EKF delle misure date da sensori inerziali (descritti in Sezione 2.1) e quelle date dal sistema di VO (presentato in Sezione 2.2). Si espongono di seguito le basi del funzionamento del filtro di Kalman per ottenere la *sensor fusion*.

Si consideri inizialmente un sistema lineare a tempo discreto espresso in spazio di stato [38]:

$$\mathbf{x}(k+1) = \mathbf{F}\mathbf{x}(k) + \mathbf{r}(k) \tag{4}$$

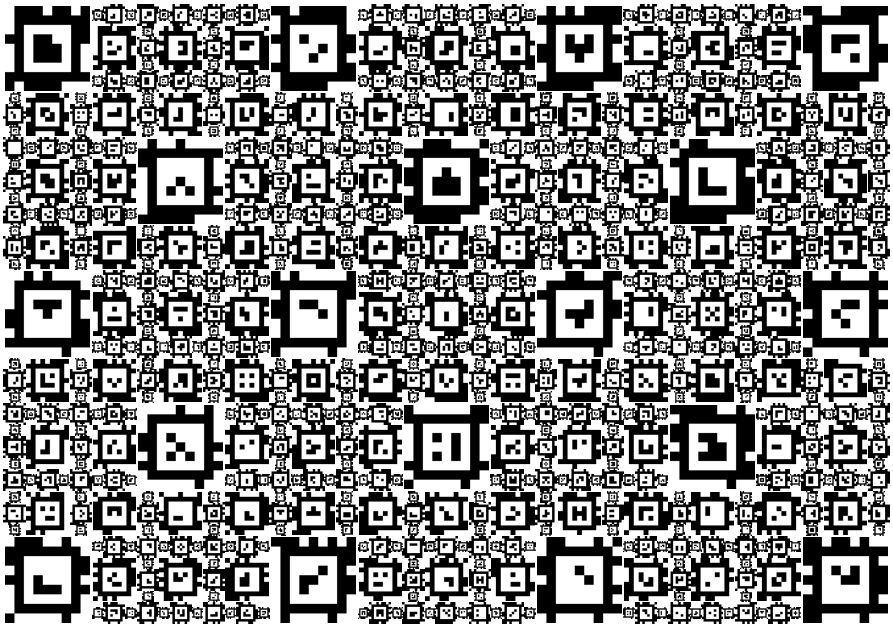


Figura 10: Mappa.

dove $\mathbf{x}(k) \in \mathbb{R}^n$ è il vettore dello stato, $\mathbf{F} \in \mathbb{R}^{n \times n}$ è la matrice di stato, $\mathbf{r}(k) \in \mathbb{R}^n$ è una sequenza di rumore bianco gaussiano a media nulla, per il quale si assume conosciuta la matrice di covarianza $\mathbf{Q}(k) \in \mathbb{R}^{n \times n}$, definita come:

$$\mathbf{Q}(k) = E[\mathbf{r}(k)\mathbf{r}(k)^T] \tag{5}$$

dove l'operatore $E[\cdot]$ denota il valore atteso dell'argomento. Nei sistemi più semplici, le misure rilevate dai sensori possono essere espresse come relazioni lineari del vettore di stato $\mathbf{x}(k)$ e disturbate dalla presenza di rumore gaussiano. Si considerino dunque delle misure acquisite mediante N sensori:

$$\mathbf{z}_i(k) = \mathbf{H}_i\mathbf{x}(k) + \mathbf{r}_i(k), \qquad i = 1, \ldots, N \tag{6}$$

dove $\mathbf{z}_i(k) \in \mathbb{R}^l$ è il vettore delle misure acquisite dall'i-esimo sensore, $\mathbf{H}_i \in \mathbb{R}^{l \times n}$ è la matrice di misura associata all'i-esimo sensore e $\mathbf{r}_i(k) \in \mathbb{R}^l$ è una sequenza di rumore bianco gaussiano a media nulla, per il quale si assume conosciuta la matrice di covarianza $\mathbf{R}_i(k) \in \mathbb{R}^{n \times n}$, definita come

$$\mathbf{R}_i(k) = E[\mathbf{r}_i(k)\mathbf{r}_i(k)^T]. \tag{7}$$

Noto il modello descritto dal sistema in spazio di stato mediante le Equazioni 4 e 6, il funzionamento del filtro di Kalman può essere suddiviso in due fasi:

1. la fase di **stima** - nella quale a partire dalle misure acquisite dai sensori, solitamente rumorose, si aggiorna la stima del vettore di stato;

2. la fase **predittiva** - nella quale il filtro di Kalman calcola una stima del vettore di stato.

Nella fase di stima, $\hat{\mathbf{x}}(k \mid k)$ (ovvero $\hat{\mathbf{x}}$ di k dato k, dunque la stima corrente) si calcola come:

$$\hat{\mathbf{x}}(k \mid k) = \hat{\mathbf{x}}(k \mid k-1) + \sum_{i=1}^{N} \mathbf{K}_i(k)\mathbf{v}_i(k) \tag{8}$$

dove

$$\mathbf{K}_i(k) = \mathbf{P}(k \mid k)\mathbf{H}_i^T \mathbf{R}_i^{-1}(k) \tag{9}$$

è il guadagno variabile per la fusione delle misure associato all'i-esimo sensore, $\mathbf{P}(k \mid k)$ è la matrice di incertezza al passo k sulla stima dello stato, calcolata a partire dalla matrice di incertezza al passo precedente mediante:

$$\mathbf{P}^{-1}(k \mid k) = \mathbf{P}^{-1}(k \mid k-1) + \sum_{i=1}^{N} \mathbf{H}_i^T \mathbf{R}_i^{-1}(k)\mathbf{H}_i \tag{10}$$

e

$$\mathbf{v}_i(k) = \mathbf{z}_i(k) - \mathbf{H}_i\hat{x}(k \mid k-1) \tag{11}$$

è l'innovazione associata alla misura acquisita dall'i-esimo sensore.

Nella fase predittiva, $\hat{\mathbf{x}}(k+1 \mid k)$ (ovvero $\hat{\mathbf{x}}$ di $k+1$ dato k, dunque la predizione) si calcola come:

$$\hat{\mathbf{x}}(k+1 \mid k) = \mathbf{F}\hat{\mathbf{x}}(k \mid k) \tag{12}$$

$$\mathbf{P}(k+1 \mid k) = \mathbf{F}\mathbf{P}(k \mid k)\mathbf{F}^T + \mathbf{Q}(k) \tag{13}$$

Il metodo sopra descritto può essere esteso per sistemi non lineari. Uno dei metodi più usati per la stima della posa di UAV utilizza un estensione del filtro di Kalman denominata EKF, a causa della semplicità computazionale dell'implementazione [6]. Il controllore di volo PX4 Autopilot, in uso nell'apparato sperimentale, implementa proprio EKF per la stima dello stato. In particolare, lo stato $\mathbf{x}(k) \in \mathbb{R}^{24}$ è composto da:

- $\mathbf{q} \in \mathbb{R}^4$ - quaternione che definisce la rotazione di *body frame* in *world frame* [-];
- $\mathbf{v} \in \mathbb{R}^3$ - velocità lineare del *body frame* in *world frame* $[\mathrm{m\,s^{-1}}]$;
- $\mathbf{p} \in \mathbb{R}^3$ - posizione lineare del *body frame* in *world frame* $[\mathrm{m}]$;
- $\mathbf{b}_\alpha \in \mathbb{R}^3$ - bias della posizione angolare del *body frame* $[\mathrm{rad}]$;
- $\mathbf{b}_\nu \in \mathbb{R}^3$ - bias della velocità angolare del *body frame*$[\mathrm{rad\,s^{-1}}]$;

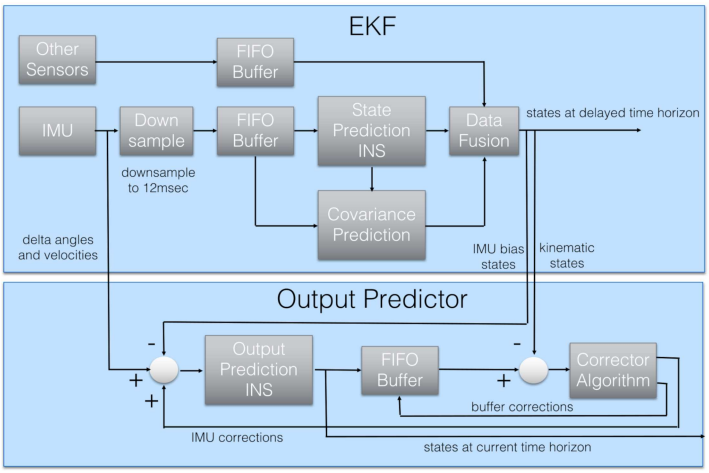


Figura 11: EKF e predittore dell'uscita in PX4 Autopilot.

- $\mathbf{b}_v \in \mathbb{R}^3$ - vettore di intensità del campo magnetico in *world frame* [Gauss];
- $\mathbf{b}_m \in \mathbb{R}^3$ - bias del vettore di intensità del campo magnetico in *body frame* [Gauss];
- $\mathbf{v}_w \in \mathbb{R}^2$ - velocità del vento in direzione Nord e Est $[m\,s^{-1}]$.

In Figura 11 è riportato lo schema a blocchi dell' EKF implementato in PX4 Autopilot. A causa della differente frequenza di campionamento dei differenti sensori, la stima mediante EKF presenterà un ritardo, che può causare instabilità. Di conseguenza nelle implementazioni reali è presente anche un predittore d'uscita, che garantisce una predizione dello stato senza ritardi [17].

3 IMPLEMENTAZIONE

Per implementare il sistema di visione progettato si è scelto di scrivere e integrare nodi **ROS 2** (Robotic Operating System) con il software **PX4 Autopilot**.

In Sezione 3.1 ci si focalizza sul software PX4 Autopilot, in Sezione 3.2 si descrive ROS 2 e perché se ne è scelto l'utilizzo, mentre in Sezione 3.3 si tratta l'integrazione tra i due, con particolare riferimento alla comunicazione di messaggi. Una volta descritti in Sezione 3.4 i nodi ROS 2 necessari al funzionamento del sistema di *Visual Odometry*, nelle Sezioni 3.5 e 3.6 si approfondisce il nodo *apriltag to visual odometry*, scritto in una prima versione da Segattini [3] e in una seconda durante questo stesso lavoro di tesi.

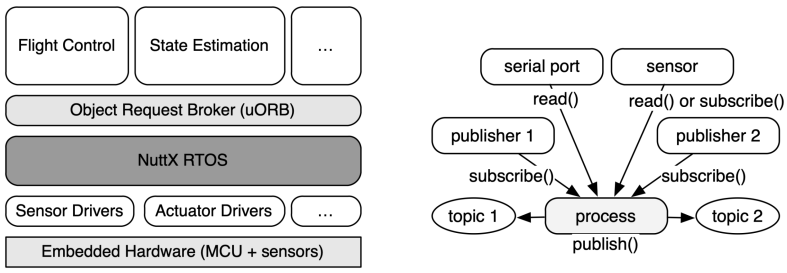
3.1 PX4 AUTOPILOT

PX4 Autopilot è un autopilota *open-source*, sviluppato e supportato sia dal mondo accademico che industriale. Per autopilota si intende un sistema in grado di controllare la traiettoria di un mezzo, senza richiedere un *input* manuale costante da un operatore umano.

3.1.1 *Storia di PX4 e Pixhawk*

Pixhawk è il nome di un gruppo di studenti al politecnico di Zurigo, che vinsero nel 2009 la competizione *European Micro Air Vehicle* grazie a un controllore di volo sviluppato da zero dagli stessi studenti. Il team rilasciò il codice con licenza *open-source*, ed ebbe da subito un importante successo. Lo sviluppo del software del controllore di volo proseguì imperterrito, e si affrontarono varie riscritture da zero del codice. La quarta riscrittura del codice convinse il fondatore Lorenz Meier, così il software che ne risultò venne chiamato **PX4**.

Al giorno d'oggi la comunità di PX4 conta 9600 utenti e più di 600 collaboratori, che contribuiscono alla crescita del codice. Il fondatore sottolinea come il successo del progetto sia stato possibile grazie al carattere *open-source* dell'iniziativa [39]. È importante sottolineare la differenza tra PX4, denominazione del lato software del progetto, e *Pixhawk*, termine ora utilizzato sia per definire lo standard industriale per la progettazione dell'hardware per l'autopilota PX4, che per identificare l'hardware stesso prodotto da *Holybro*.



(a) (b)

Figura 12: Architettura software: in (a) la struttura a livelli, in (b) un generico processo si sottoscrive o pubblica su diversi *topics* [2].

3.1.2 *Framework PX4*

Il Framework PX4 è stato presentato alla comunità scientifica da Meier et al. [2]. L'architettura software si suddivide in quattro differenti livelli riportati in Figura 12a, rendendo il sistema modulare:

- il primo livello comprende i driver e si suddivide in due sottolivelli. Il sottolivello inferiore, denominato *Embedded Hardware*, comprende i driver specifici per i dispositivi connessi, mentre il sottolivello superiore comprende i driver standardizzati facenti parte del sistema operativo;
- il secondo livello è composto dal sistema operativo *NuttX* [40];
- il terzo livello è il *micro Object Request Broker* (µORB), che gestisce la comunicazione garantendo l'integrità dei dati tra processi e thread;
- il quarto livello è l'*application layer*, che comprende applicazioni come il controllore di volo e gli stimatori dello stato.

I vari processi in esecuzione comunicano seguendo lo schema *publish-subscribe*, secondo il quale determinati dati prodotti da un certo processo, il *publisher*, possono essere consumati da uno o più processi in sottoscrizione, i *subscriber*. In Figura 12b si riporta un esempio di comunicazione tra processi.

3.1.3 *Architettura del controllore di volo di PX4*

PX4 implementa un controllore composto da più anelli in cascata a prescindere dal velivolo, mentre la stima dello stato viene fornita da EKF, come approfondito in Sezione 2.4. In particolare, nel caso di un multirotore il controllore ha una struttura a blocchi riportata in Figura 13, composta a partire dall'anello più interno da:

- un **controllore K-PID per la velocità angolare** - riportato in Figura 14, un consueto controllore PID con l'aggiunta di un gua-

dagno K che moltiplica i termini P, I, D. È inoltre presente un limitatore dell'integrale per evitare il fenomeno di *wind-up*, di un filtro passa basso per l'azione derivativa e di un ulteriore limitatore in uscita;

- un **controllore P per la posizione angolare (*attitude*)** - riportato in Figura 15, utilizza i quaternioni per gestire l'informazione di orientamento dell'UAV (si veda appendice A.1) e presenta anch'esso un limitatore in uscita;
- un **controllore PID per la velocità lineare** - riportato in Figura 16a, con l'aggiunta di un filtro passa basso per l'azione derivativa;
- un **controllore P per la posizione lineare** - riportato in Figura 16b, con l'aggiunta di un saturatore in uscita.

3.2 ROS 2

Robotic Operating System (ROS) è un insieme di librerie *open-source* per applicazioni dedicate alla robotica. Le principali caratteristiche del framework ROS possono essere riassunte in [41]:

- *peer-to-peer* - un generico sistema implementato tramite ROS è composto da numerosi processi, potenzialmente in esecuzione su differenti dispositivi, connessi tramite tecnologia *peer-to-peer* in *real-time*;
- *multi-lingual* - differenti programmatori possono avere differenti preferenze riguardo al linguaggio di programmazione da utilizzare, per questo ROS supporta C++, Python, Octave e LISP;
- *tool-based* - per gestire la complessità di ROS è stato implementato un *microkernel*, nel quale differenti e numerosi *tool* sono usati per compilare ed eseguire una gran varietà di elementi di ROS, così da garantire una struttura modulare;
- *thin* - al fine di rendere driver e algoritmi utilizzabili al di fuori del progetto iniziale, si consiglia di spostare la complessità in librerie non dipendenti da ROS, così da poter scrivere eseguibili leggeri e comprensibili, che esulano dalla complessità delle librerie;
- *free* e *open-source* - il codice sorgente di ROS è disponibile pubblicamente, si garantisce dunque un maggior grado di libertà nel *debugging* a qualsiasi livello.

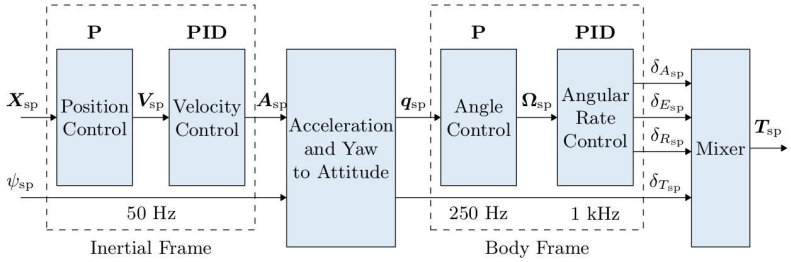


Figura 13: Architettura del controllore di PX4 per multirotori.

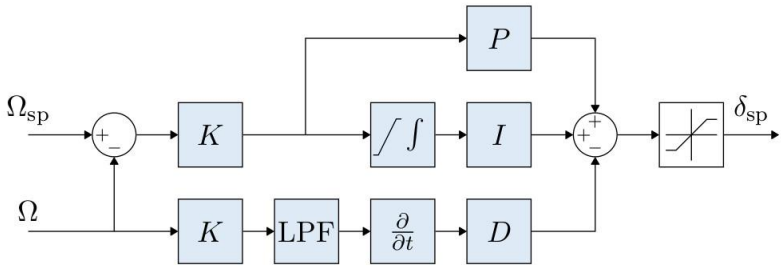


Figura 14: Controllore PID di velocità angolare.

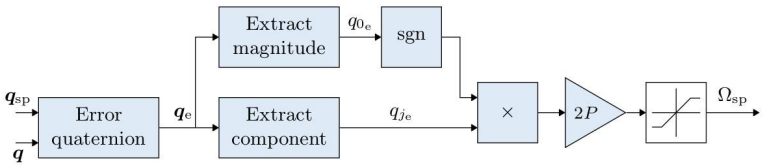
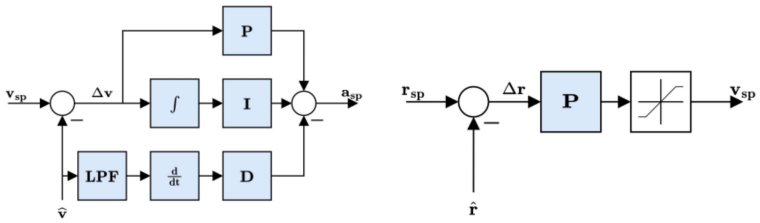
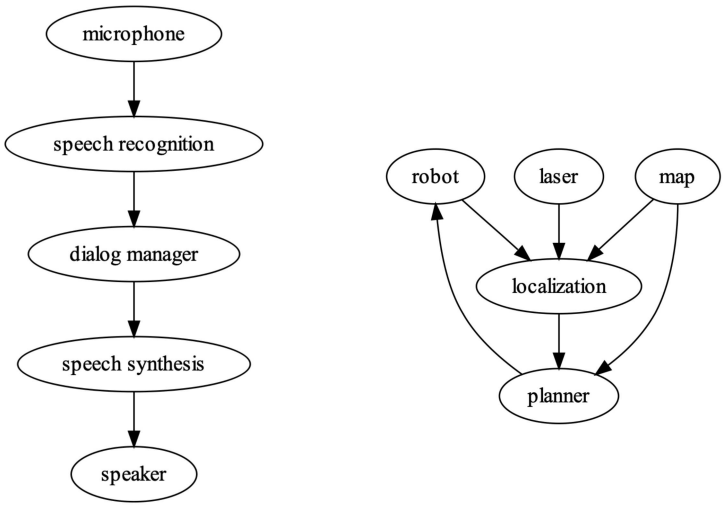


Figura 15: Controllore P di posizione angolare.



(a) Controllore PID di velocità (b) Controllore P di posizione

Figura 16: Controllori di posizione e velocità lineari.



(a) (b)

Figura 17: Esempi di comunicazione tra diversi nodi.

3.2.1 *Nomenclatura*

I concetti fondamentali legati ad un'architettura ROS sono nodi, messaggi, *topic* e servizi [41].

I **nodi** sono processi che eseguono i calcoli. ROS è progettato per essere modulare, di conseguenza solitamente un sistema è composto da più nodi interconnessi tramite una comunicazione *peer-to-peer*.

I nodi comunicano tra loro scambiandosi **messaggi**. Un messaggio è una struttura dati strettamente definita, che supporta i tipi primitivi come integer, floating point, boolean, etc.

I messaggi vengono pubblicati dai nodi su un dato ***topic***, una semplice stringa come "tf" o "visual_odometry". Un nodo interessato a una certa tipologia di dati deve sottoscriversi a un appropriato *topic*. Per un dato *topic* possono esserci più *publisher* e *subscriber* concorrenti, mentre un singolo nodo può sottoscriversi o pubblicare su più *topics*. Esempi di grafici di connessione tra nodi sono riportati in Figura 17.

Il modello *publish-subscribe* basato su *topic* non è appropriato per transazioni sincrone, che possono semplificare il progetto di nodi. In ROS si possono ottenere transazioni sincrone mediante l'utilizzo di **servizi**, definiti mediante una stringa e un paio di tipi di messaggio. Il funzionamento è similare ai *web services* e, al contrario dei *topics*, solamente un nodo può pubblicare un servizio con un certo nome: per esempio può esistere solo un servizio chiamato *classify_image*, come può esistere solo un *web service* a un dato URI (*Uniform Resource Identifier*).

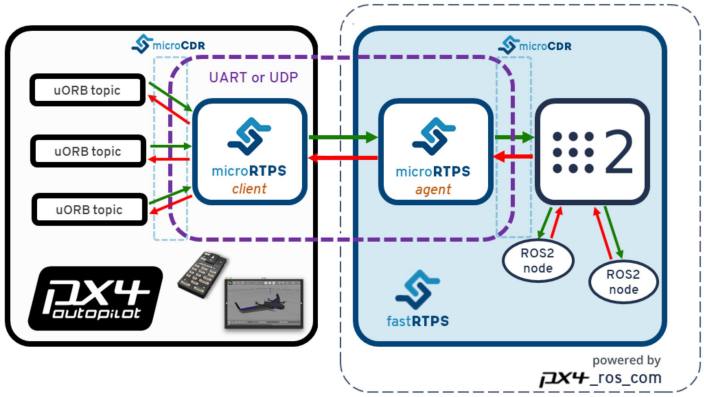


Figura 18: Architettura di PX4-ROS2 bridge.

3.3 PX4-ROS 2 BRIDGE

L'insieme di PX4 e ROS 2 permette la creazione di complesse e performanti applicazioni per UAVs, poiché si possono affiancare all'autopilota PX4 le numerose librerie *open-source* a disposizione del *framework* ROS 2. Il **PX4-ROS 2 bridge** [42] implementa il *translation layer* tra ROS 2 e PX4, che garantisce una connessione bidirezionale tra i messaggi µORB di PX4 e quelli di ROS 2. L'architettura di PX4-ROS 2 bridge è riportata in Figura 18. In particolare, l'interconnessione è data da un **microRTPS client** in esecuzione su PX4 Autopilot e un **microRTPS agent** in esecuzione su un companion computer, in concomitanza con diversi nodi ROS.

Per ROS 2 sono disponibili i pacchetti `px4_msgs` e `px4_ros_com`, che assicurano come le definizioni dei messaggi corrispondenti siano usate per la creazione sia del *client* che dell'*agent*:

- `px4_msgs` contiene le definizione dei messaggi PX4 ROS;
- `px4_ros_com` compila `px4_msgs` e contiene il sorgente dell'agent microRTPS.

Grazie a questa interconnessione è possibile creare nodi ROS 2 che si sottoscrivano o pubblichino direttamente su *topic* µORB di PX4.

3.4 ARCHITETTURA DEL SISTEMA DI VO IN ROS 2

Il sistema di *VO* descritto nel Capitolo 2 è implementato mediante tre nodi ROS scritti in codice C++ che eseguono su un *companion-computer* a bordo dell'UAV. Il codice sorgente è disponibile nella pagina *GitHub* del gruppo di ricerca *C-square*[1]. In Figura 19 si riporta il diagramma dell'architettura del sistema e la tipologia di messaggi scambiati, mentre nelle seguenti sottosezioni si analizzano i singoli nodi.

1 https://github.com/C-square-unipd

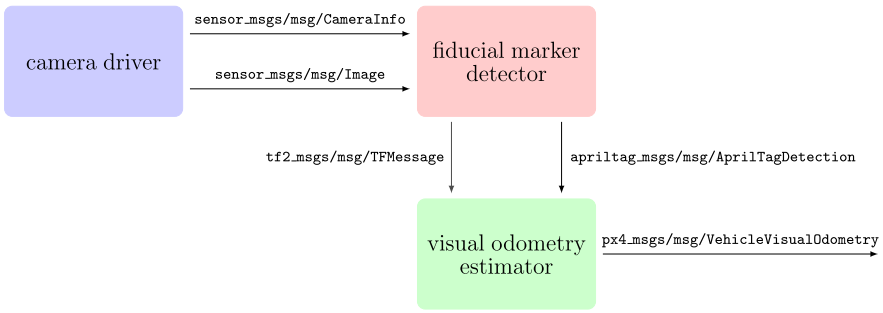


Figura 19: Architettura dei principali nodi ROS per il sistema di *visual odometry*. Si specificano anche i tipi dei messaggi trasmessi o ricevuti.

3.4.1 *Camera driver*

Il nodo *camera driver* pubblica l'informazione delle immagini collezionate dal sensore su un *topic* apposito. Per ogni tipologia di camera utilizzata è necessario avere a disposizione il driver corretto che pubblichi i seguenti messaggi:

- `sensor_msgs/msg/CameraInfo` - definisce le meta informazioni per una camera, tra le quali sono di particolare importanza i parametri di calibrazione;
- `sensor_msgs/msg/Image` - contiene l'immagine non compressa catturata dal sensore, mediante una matrice di `uint8`.

È necessario specificare gli opportuni parametri di configurazione per impostare i frame per secondo (fps) e la risoluzione della camera. I fps coincidono con la frequenza con la quale si calcola la stima della posa, a patto che il *companion-computer* sia abbastanza potente da processare l'informazione alla medesima frequenza. Una maggiore risoluzione consente invece di identificare *markers* di una data dimensione a distanze maggiori, nonché di avere una stima di posizione più accurata. In base al *companion-computer* a disposizione è opportuno trovare un adeguato compromesso tra i due parametri.

3.4.2 *Fiducial marker detector*

In base al *fiducial marker* scelto, un nodo apposito identifica i *tags* nell'immagine fornita dal nodo *camera driver* e calcola la trasformazione tra sensore e *tag* identificato. Si è scelto di implementare il sistema con *markers* AprilTag tramite il nodo *apriltag ros*. Quest'ultimo si sottoscrive ad appositi *topics* per ricevere i messaggi contenenti l'informazione della camera, mentre pubblica:

- `tf2_msgs/msg/TFMessage` - messaggio composto da un vettore di `geometry_msgs/msg/TransformStamped.msg`, nel quale ogni elemento definisce la trasformazione dal sistema di riferimento della camera al centro di un *marker identificato*.

- `apriltag_msgs/msg/AprilTagDetection` - contenente le informazioni 2D e 3D riguardo un singolo AprilTag identificato.

Come analizzato in Sezione 3.6, per rendere più efficiente il sistema si è deciso di pubblicare la trasformazione inversa, dunque dall'AprilTag identificato alla camera, in un *topic* chiamato `/tf_vio`. Con tale modifica si riserva un *topic* solo per le trasformazioni relative alle identificazioni di AprilTag, mentre in un sistema robotico il *topic* `/tf` è spesso utilizzato da altri nodi per la pubblicazione di trasformazioni. In caso si decida di utilizzare un diverso nodo per l'identificazione di *fiducial marker*, è necessario modificare il codice sorgente aggiungendo un *publisher* su `/tf_vio` delle trasformazioni identificate.

Il nodo *apriltag ros* è configurabile tramite la porzione seguente del file *launch*

```
    cfg_41h12 = {
    "image_transport": "raw",
    "family": "Standard41h12",
    "size": 0.027,
    "max_hamming": 0,
    "z_up": False,
    "pub_inv": True,
    "threads": 1,
    "tag_ids": [0, 100, 400, 1000],
    "tag_frames": ['frame0', 'frame100', 'frame400', 'frame1000'],
    "tag_sizes": [0.256, 0.128, 0.064, 0.032]
}
```

nel quale i parametri più importanti da specificare sono:

- *"family"* - specifica che famiglia di AprilTag riconoscere (si veda Sezione 2.3);
- *"tag_ids"* - vettore che specifica il sottoinsieme di ID da identificare;
- *"tag_frames"* - vettore che specifica il nome del frame da assegnare all'id corrispondente al medesimo indice in *"tag_ids"*;
- *"tag_sizes"* - vettore che specifica la dimensione da assegnare all'id corrispondente al medesimo indice in *"tag_ids"*.

Necessariamente i vettori *"tag_ids"*, *"tag_frames"* e *"tag_sizes"* devono avere la stessa dimensione.

3.4.3 *Visual odometry estimator*

Il nodo *visual odometry estimator* si sottoscrive ai *topics* pubblicati dal *fiducial marker detector* ed elabora l'informazione ricevuta per generare la stima di *visual odometry*. Una volta ottenuta, la stima di VO viene inviata a PX4 Autopilot tramite il PX4-ROS 2 bridge, così che possa essere fusa con le misure inerziali per ottenere la stima di *visual inertial odometry*.

Questo nodo è il fulcro del sistema di VO, dunque viene presentato in due sezioni distinte. Nello specifico, una prima versione del nodo è descritta in Sezione 3.5, mentre una seconda è esposta in Sezione 3.6. La seconda versione si differenzia principalmente dalla prima per l'implementazione di algoritmi più avanzati, al fine di stimare più accuratamente la posa del velivolo.

3.5 APRILTAG TO VISUAL ODOMETRY - PRIMA VERSIONE

apriltag to visual odometry è un nodo creato appositamente da Segattini [3], che genera una stima di *visual odometry* e la comunica a PX4 Autopilot per il calcolo della posa del multirotore.

In questa prima versione il nodo si sottoscrive ai *topics*:

- `/tf` - da cui mediante l'oggetto *Transform Listener* il nodo ricava le informazioni relative alla trasformazione tra camera e AprilTag identificato;
- `/fmu/timesync/out` - dal quale il nodo ricava il tempo in µs dall'avvio di PX4, necessario per rendere il sistema composto da autopilota e *companion-computer* isocrono[2];
- `/fmu/vehicle_odometry/out` - dal quale il nodo ricava la posa del velivolo a valle dell'EKF implementato in PX4, dunque la stima di VIO;

e pubblica sui *topics*:

- `/tf` - un vettore di trasformazioni per visualizzare una simulazione dell'ambiente di volo [3];
- `/apriltag_for_estimation` - una stringa di testo contenete una lista con gli ID dei *markers* utilizzati per ricavare la posa dall'avvio dell'algoritmo;
- `fmu/vehicle_visual_odometry/in` - la stima di *visual odometry*, con la quale PX4 calcolerà la stima di *visual inertial odometry* mediante EKF.

In Tabella 3 si specifica a ogni *topic* del precedente elenco che messaggio è associato [3], mentre la Figura 20 rappresenta a quali *topics* il nodo si sottoscrive e pubblica.

La funzione principale del nodo è *on_timer()*, che viene richiamata a una frequenza di 50 Hz. Uno schema logico della funzione è riportato in Figura 21. Obiettivo della funzione *on_timer()* è di ricavare da

2 Un sistema distribuito si definisce isocrono se differenti dispositivi interconnessi condividono l'informazione sul tempo.

3 Si pubblicano le pose statiche relative ai *markers* presenti nella mappa e la posa del *body frame* del multirotore, così tramite l'utilizzo di *RViz* si può comprendere se la stima della posa del velivolo ricavata dal sistema di VIO sia coerente con quella reale.

Topic	Tipo di messaggio associato
/fmu/timesync/out	px4_msgs/msg/Timesync
/fmu/vehicle_odometry/out	px4_msgs/msg/VehicleOdometry
/apriltag_for_estimation	std_msgs/msg/String
/fmu/vehicle_visual_odometry/in	px4_msgs/msg/VehicleVisualOdometry
/tf	tf2_msgs/msg/TFMessage
/tf_static	tf2_msgs/msg/TFMessage

Tabella 3: Topic e relativi messaggi associati di *apriltag to visual odometry v1*

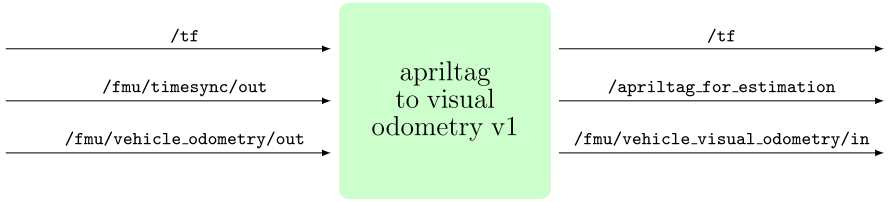


Figura 20: Topic in sottoscrizione (a sinistra) e in pubblicazione (a destra) per il nodo *apriltag to visual odometry* versione 1.

un buffer di *transform* la trasformazione più recente relativa all'AprilTag di dimensione maggiore che, per come sono stati definiti i marker nella generazione della mappa (Sezione 2.3), risulta nell'identificazione dell'AprilTag con ID minore. Il buffer contiene le trasformazioni pubblicate sul *topic* `/tf`, e mediante la classe *Transform Listener*, si può richiedere una certa trasformazione specificando l'ID di un AprilTag. In tal senso, al fine di determinare l'Apriltag con ID minore più recentemente identificato e presente nel buffer, si scorre il buffer stesso mediante un ciclo.

Una volta identificato il *marker* con ID minore, si determina la trasformazione corrispondente contenuta nel buffer. Quest'ultima è la trasformazione tra *AprilTag frame* $\mathcal{F}_A$ e *camera frame* $\mathcal{F}_C$, come evidenziato in Figura 22. La trasformazione di interesse è tra *world frame* $\mathcal{F}_W$ e *body frame* $\mathcal{F}_B$, si definiscono dunque le seguenti matrici di trasformazione:

- $\mathbf{T}_{W,A} \in \mathbb{R}^{4\times4}$ - trasformazione statica tra $\mathcal{F}_W$ e $\mathcal{F}_A$, nota dal progetto della mappa;
- $\mathbf{T}_{A,C} \in \mathbb{R}^{4\times4}$ - trasformazione tra $\mathcal{F}_A$ e $\mathcal{F}_C$, estratta dal buffer;
- $\mathbf{T}_{C,B} \in \mathbb{R}^{4\times4}$ - trasformazione statica tra $\mathcal{F}_C$ e $\mathcal{F}_B$, nota dal progetto del multirotore.

Per giungere alla trasformazione tra *world frame* e *body frame* $\mathbf{T}_{W,B} \in \mathbb{R}^{4\times4}$ da inviare a PX4 Autopilot si calcola:

$$\mathbf{T}_{W,B} = \mathbf{T}_{W,A}\mathbf{T}_{A,C}\mathbf{T}_{C,B} \tag{14}$$

In ultimo si procede alla pubblicazione della trasformazione così ricavata, di modo che PX4 Autopilot possa stimare la posa del velivolo.

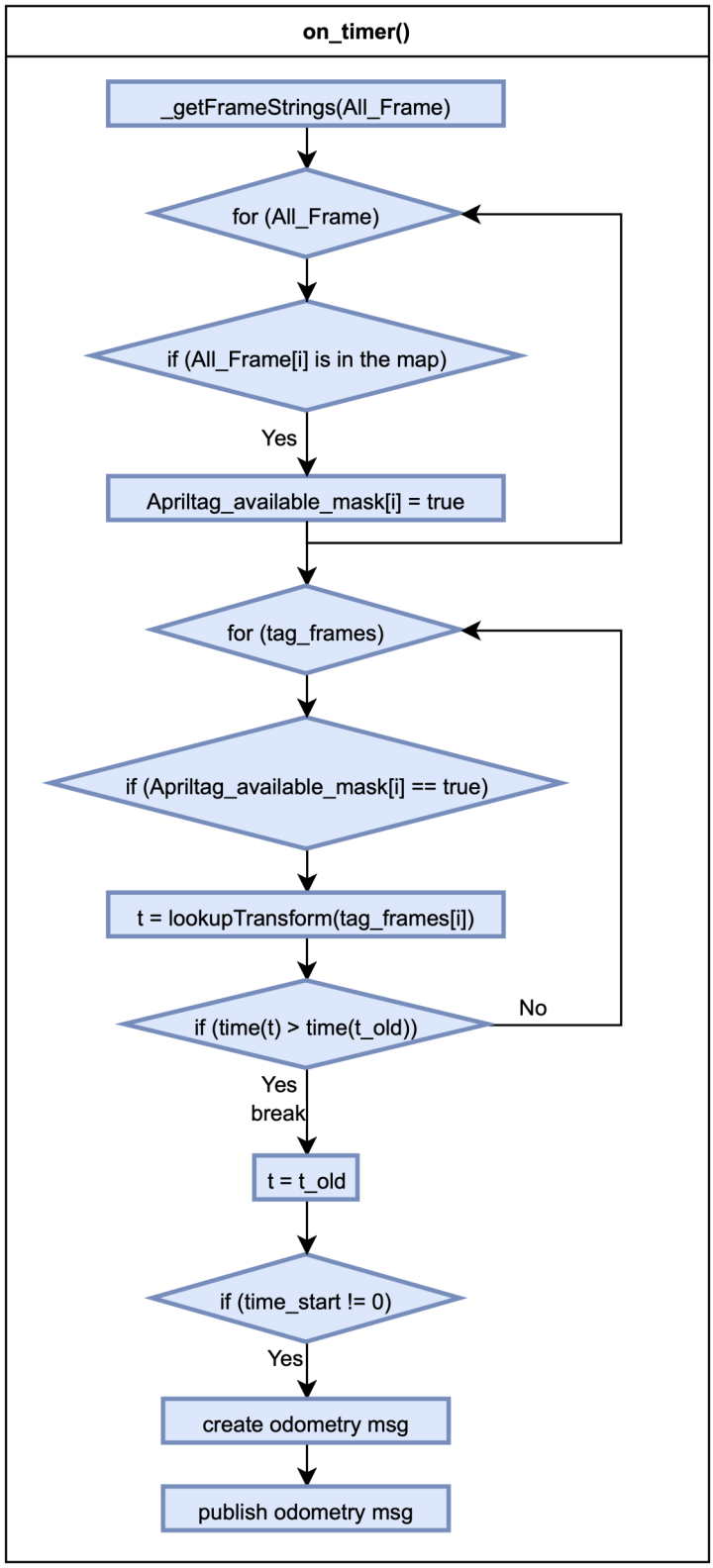


Figura 21: Logica di *on_timer()*, principale funzione del nodo *apriltag to visual odometry* versione 1 [3].

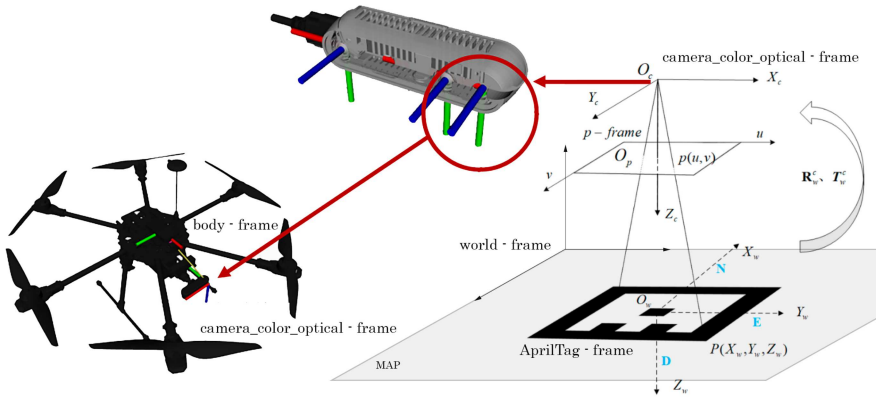


Figura 22: Sistemi di riferimento.

Il sistema così implementato è stato validato tramite prove sperimentali, i cui risultati sono discussi in [17] e nel capitolo 4, ma l'accuratezza di localizzazione non era soddisfacente, soprattutto in ottica di interazione del multirotore con l'ambiente esterno, poiché l'errore della stima di posizione raggiungeva il valore di 30 cm in valore assoluto.

3.6 APRILTAG TO VISUAL ODOMETRY - SECONDA VERSIONE

Al fine di migliorare l'accuratezza della stima di *visual odometry*, è stato implementato un nuovo nodo *apriltag to visual odometry*, apportando le seguenti migliorie:

- **minimizzazione della latenza** - l'obiettivo è quello di accoppiare temporalmente l'invio di un *TFMessage* da *apriltag ros* e l'invio del messaggio di odometria a PX4 da parte di *apriltag to visual odometry*;
- **miglior sfruttamento delle informazioni** - l'idea è quella di non basare la stima sull'informazione derivante da un solo *marker*, ma, per mezzo di una media pesata, si utilizza l'informazione di tutte le trasformazioni a disposizione;
- **rimozione degli *outliers*** - lo scopo è quello di identificare le informazioni non utili alla stima (*outliers*), in modo da escluderle nel computo della media pesata;
- **attenuazione del rumore in alta frequenza** - l'idea è di minimizzare il rumore sulla stima finale attraverso l'implementazione di un filtro a media mobile pesata.

Nelle sottosezioni successive si analizza in dettaglio l'implementazione delle singole funzioni implementate per apportare le migliorie elencate.

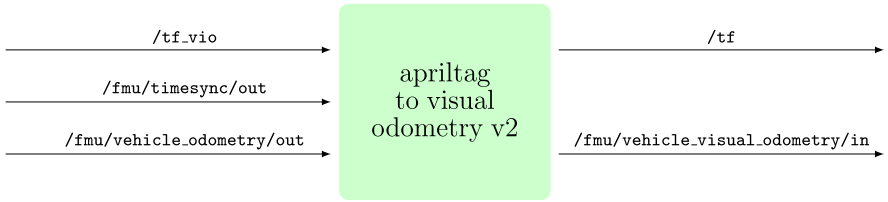


Figura 23: Topic in sottoscrizione (a sinistra) e in pubblicazione (a destra) per il nodo *apriltag to visual odometry* versione 2.

3.6.1 *Minimizzazione della latenza*

Al fine di accoppiare temporalmente l'invio di un *TFMessage* da *apriltag ros* e l'invio del messaggio di odometria a PX4 da parte di *apriltag to visual odometry*, si è deciso di non utilizzare la struttura ad alto livello di *tf2*, ovvero *Buffer* e *Transform Listener*. In particolare, utilizzando *Transform Listener* è necessario specificare l'identificativo di un *marker* affinché venga restituita la relativa trasformazione. Non si è tuttavia a conoscenza di quali AprilTag siano stati identificati, quindi è necessario implementare diversi cicli per ottenere la corretta trasformazione, come si nota nell'implementazione di *on_timer()* in Figura 21.

Per rendere più snello il sistema si è creato un apposito *topic* `tf_vio`, sul quale *apriltag ros* pubblica le trasformazioni identificate, mentre *apriltag to visual odometry* si sottoscrive per riceverle. Di conseguenza, non è più presente una funzione come *on_timer()* che esegue ogni 20 ms, ma non appena un messaggio viene inviato sul *topic* `tf_vio` si esegue la funzione *tf_callback*(), avente come parametro proprio il *TFMessage* inviato da *apriltag ros*, contenente le trasformazioni identificate. L'esecuzione di *tf_callback*() è dunque conseguente all'identificazione di AprilTag nella mappa, con una latenza trascurabile.

Analizzando il nodo *apriltag ros*, si nota come il vettore di trasformazioni contenuto in un messaggio *TFMessage* sia disposto in ordine crescente sulla base degli ID degli Apriltag identificati. Con il sistema così configurato, si può implementare un semplice algoritmo che, non appena viene ricevuto un messaggio, elabori la posa del multirotore a partire dalla trasformazione all'indice iniziale nel vettore ricevuto, corrispondente all'Apriltag con ID minore. Si ottiene così lo stesso risultato rispetto alla soluzione implementata da Segattini, ma in modo più snello ed efficiente.
Il diagramma in Figura 23 schematizza a quali *topics* si sottoscrive e pubblica il nodo. Confrontando il diagramma con quello in Figura 20, si nota come il nodo non si sottoscrive più al *topic* `/tf` mediante la classe *Buffer*, bensì si sottoscrive al *topic* appositamente creato `/tf_vio`, nel quale transitano solo i messaggi relativi alle trasformazioni identificate da *apriltag ros*. Si può notare anche la rimozione del *publisher* su `/apriltag_for_estimation`, in quanto non più necessario.

Come accennato, la funzione principale del codice è *tf_callback*()

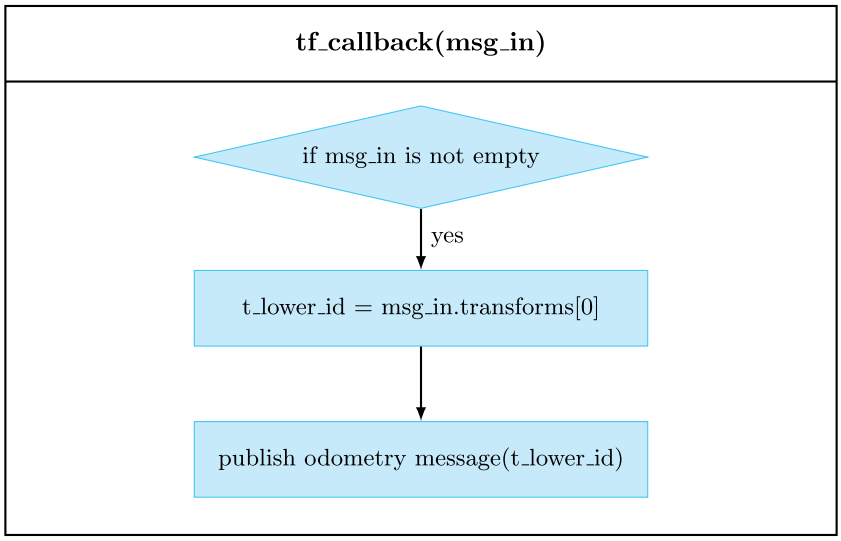


Figura 24: Diagramma logico della funzione *tf_callback()* configurata per acquisire ed elaborare la trasformazione relativa all'AprilTag con ID minore.

configurata in modo tale da considerare solo una tra le trasformazioni relative agli AprilTag identificati. In Figura 24 è riportato un diagramma dell'esecuzione della funzione così configurata. Si noti la semplicità dell'implementazione rispetto alla precedente schematizzata in Figura 21. Inoltre, avendo accoppiato la chiamata della funzione *tf_callback* con la pubblicazione del messaggio *TFMessage* da parte di *apriltag ros*, si minimizza la latenza.

3.6.2 *Media pesata tra più trasformazioni*

Per ogni frame catturato dalla camera, *apriltag ros* invia un messaggio contenente un vettore di trasformazioni, una per ogni AprilTag identificato. È palese come andando a utilizzare un solo AprilTag per la stima di *visual odometry* si stia trascurando una grande quantità d'informazione a disposizione, potenzialmente utile per migliorare l'accuratezza della stima. Nel tipo di messaggio *TFMessage*, l'informazione sulla trasformazione tra *AprilTag frame* $\mathcal{F}_A$ e *camera frame* $\mathcal{F}_C$ è contenuta in due elementi:

- *geometry_msgs/Vector3* - vettore di dimensione 3 di float64, contiene la traslazione nel formato $[x, y, z]$, rispettivamente le componenti x, y, z di $\mathcal{F}_C$ in $\mathcal{F}_A$;
- *geometry_msgs/Quaternion* - quaternione che identifica la rotazione di $\mathcal{F}_C$ in $\mathcal{F}_A$ mediante quattro elementi $[q_0, q_1, q_2, q_3]$, anch'essi float64. Si rimanda in Appendice A per meglio comprendere la rappresentazione di rotazioni mediante quaternioni.

Come visto in Sezione 3.5, la trasformazione di interesse è quella tra il *world frame* $\mathcal{F}_W$ e il *body frame* $\mathcal{F}_B$. Per ottenere tale trasformazione a partire da quelle contenute in *TFMessage* si rimanda all'Equazione 14.

Per incapsulare l'informazione contenuta nel messaggio *TFMessage* si è deciso di utilizzare la classe *tuple*, una struttura di dimensioni fisse che permette di collezionare valori eterogenei [43]. Si è definita una *tupla* di dimensione 9 con il seguente schema, dove il numero nell'enumerazione corrisponde all'indice della *tupla* stessa:

0. elemento q_0 del quaternione rappresentante la rotazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
1. elemento q_1 del quaternione rappresentante la rotazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
2. elemento q_2 del quaternione rappresentante la rotazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
3. elemento q_3 del quaternione rappresentante la rotazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
4. coordinata x della traslazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
5. coordinata y della traslazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
6. coordinata z della traslazione da $\mathcal{F}_W$ a $\mathcal{F}_B$;
7. peso della rototraslazione da applicare nel calcolo della media;
8. ID dell'Apriltag identificato.

Una *tupla* così definita è atta a codificare l'informazione di una trasformazione. Per incapsulare in un'unica struttura le n trasformazioni codificate in *TFMessage*, si è definito un vettore di *tuple* denominato *t_in*, che avrà dunque dimensione n. Definita la struttura dati, l'obiettivo è calcolare la media pesata tra le n trasformazioni incapsulate nel vettore di *tuple* t_in. Si è innanzitutto scelto di disaccoppiare la traslazione dalla rotazione.

Per quanto riguarda la posizione di $\mathcal{F}_B$ in $\mathcal{F}_W$, è sufficiente calcolare la media delle distanze euclidee tra l'origine di $\mathcal{F}_B$ e l'origine di $\mathcal{F}_W$, ottenute a partire dalle n trasformazioni contenute in *t_in*. Si è voluto implementare una **media pesata**, scegliendo un peso proporzionale alla superficie degli AprilTag. In particolare, si sono associati i pesi 1, 4, 16 e 64 rispettivamente ai *markers* di dimensione S, M, L, XL.

Il vettore risultante $\mathbf{v}_{avg} \in \mathbb{R}^3$ si calcola come:

$$\mathbf{v}_{avg} = \frac{\sum_{i=1}^{n} w_i \cdot \mathbf{v}_i}{\sum_{i=1}^{n} w_i} \tag{15}$$

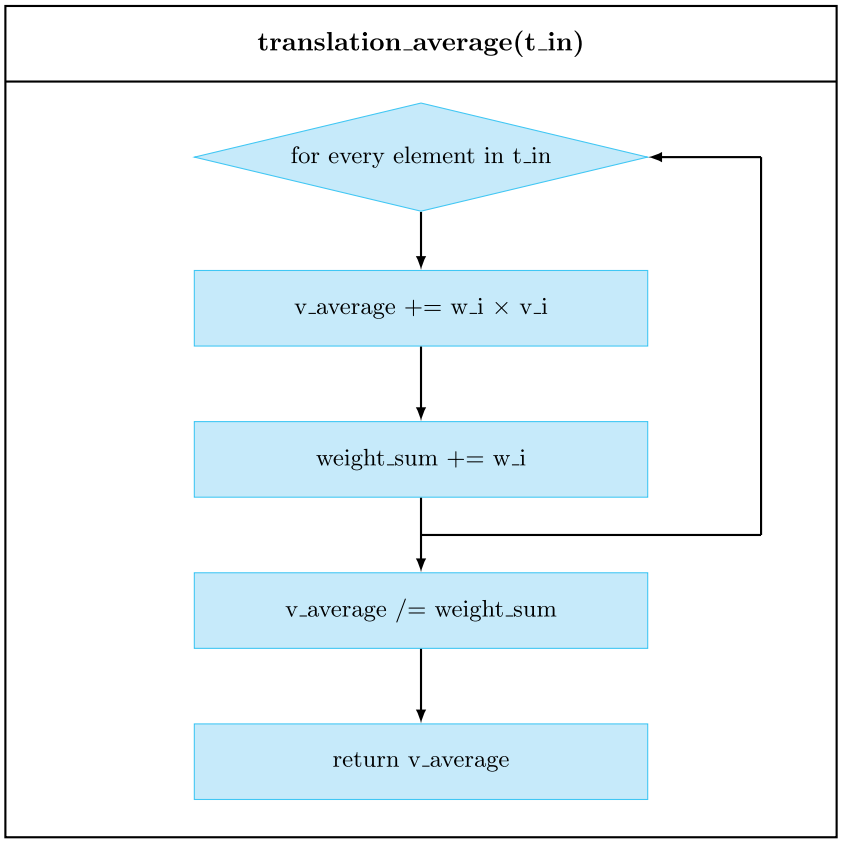


Figura 25: Diagramma logico della funzione *translation_average()*.

dove $\mathbf{v}_i \in \mathbb{R}^3$ è il vettore relativo alla traslazione tra $\mathcal{F}_W$ e $\mathcal{F}_B$ ottenuto dall'i-esima trasformazione contenuta in *t_in*, $w_i \in \mathbb{R}$ è il peso associato all'i-esima trasformazione contenuta in *t_in*. La funzione *translationAverage()* implementa tale formula, restituendo il vettore di stima di posizione derivante dalla media pesata, secondo lo schema logico riportato in Figura 25.

Per quanto riguarda la rotazione espressa mediante quaternioni, ottenere una media pesata tra gli stessi è più complesso. Questa problematica è nota in letteratura, e diversi approcci sono stati proposti per risolverla [44, 45, 46, 47]. In particolare, la soluzione proposta da Markley et al. in [44] si è rivelata atta ad essere implementata mediante un algoritmo ed efficiente. Inizialmente si calcola la matrice $\mathbf{A} \in \mathbb{R}^{4\times4}$ definita come:

$$\mathbf{A} = \frac{\sum_{i=1}^{n} w_i \cdot \mathbf{q}_i \cdot \mathbf{q}_i^T}{\sum_{i=1}^{n} w_i} \tag{16}$$

dove $w_i \in \mathbb{R}$ è il peso associato all'i-esimo quaternione $\mathbf{q}_i \in \mathbb{R}^4$ contenuto in un messaggio di n trasformazioni. Il quaternione medio $\mathbf{q}_{avg} \in \mathbb{R}^4$ equivale all'autovettore associato all'autovalore maggiore della matrice $\mathbf{A}$. Si rimanda in Appendice A.2 per la dimostrazione dell'algoritmo implementato [44]. In Figura 26 si riporta uno

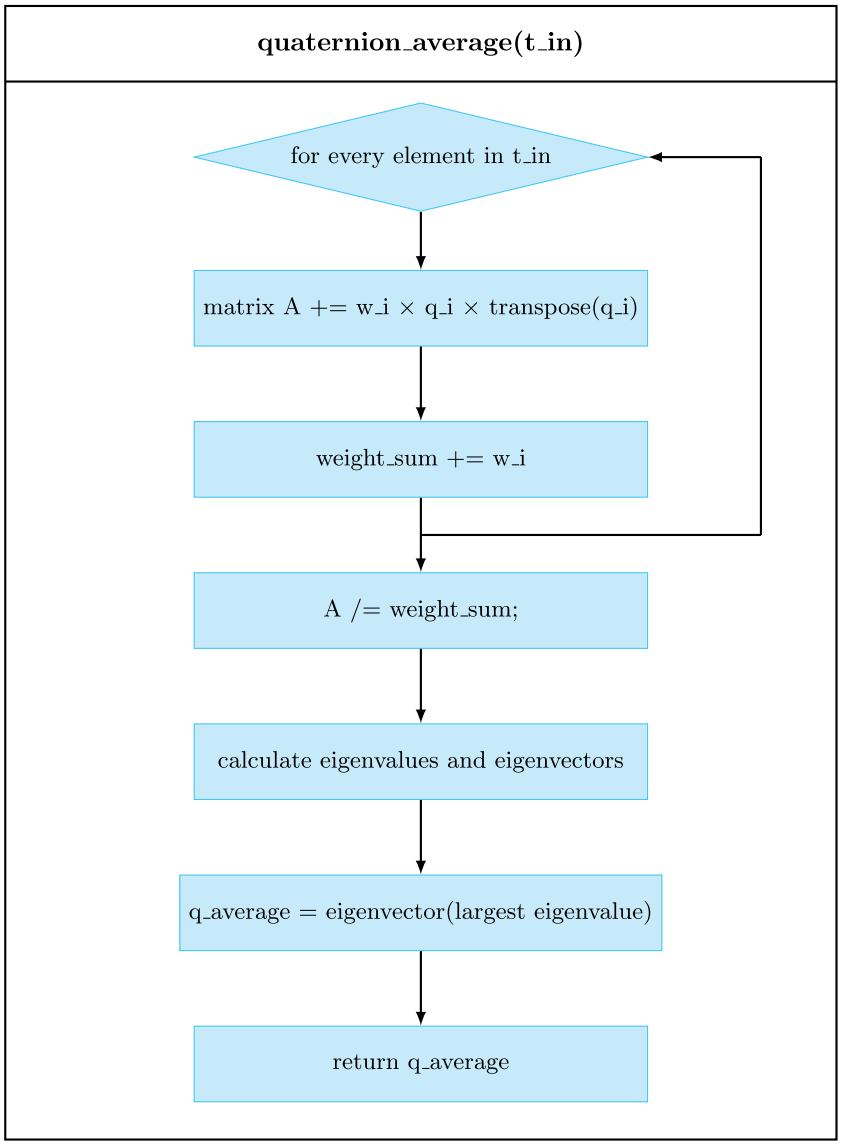


Figura 26: Diagramma logico della funzione *quaternion_average()*.

schema logico della funzione *quaternionAverage()*, che implementa il procedimento descritto.

Si è notato tramite prove sperimentali come mediare tutte le rototraslazioni non dia risultati soddisfacenti in termini di accuratezza di localizzazione, anche attribuendo un peso maggiore agli AprilTag di dimensioni maggiori. Si sono quindi studiati e implementati algoritmi per individuare e rimuovere gli *outliers* dal calcolo della media.

3.6.3 *Individuazione outliers*

DIMENSIONI DI APRILTAG Con il termine *outlier* si definisce un valore anomalo in un set di dati, ovvero un valore chiaramente distante dalle altre osservazioni disponibili. Dal punto di vista pura-

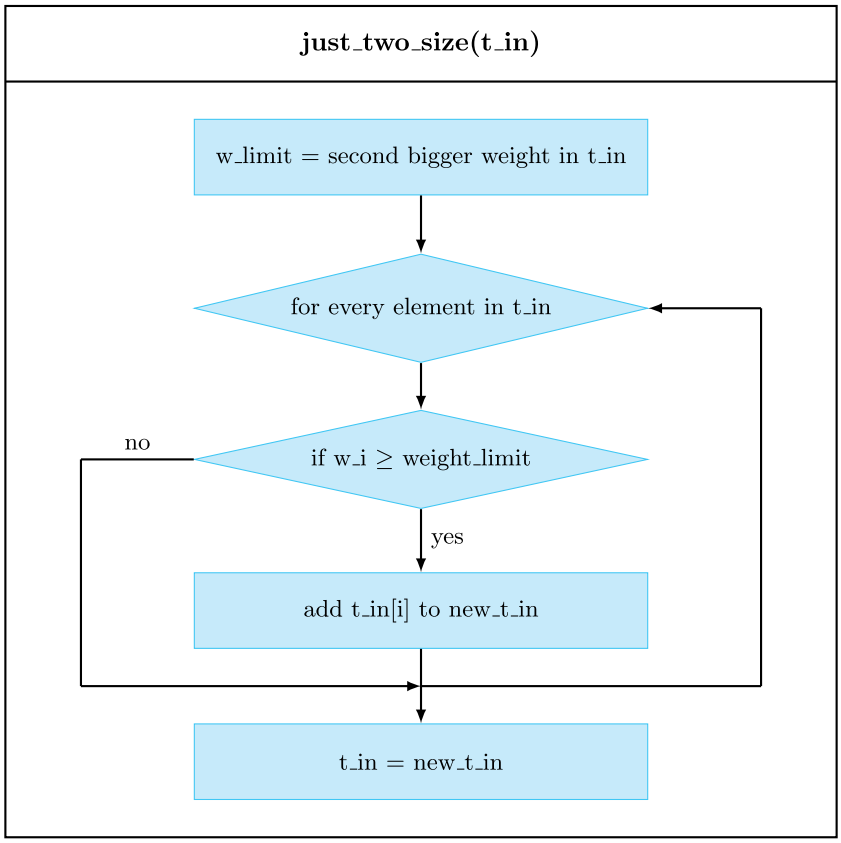


Figura 27: Diagramma logico della funzione *just_two_size()*.

mente implementativo, la rimozione degli *outliers* consiste nel chiamare delle apposite funzioni prima dell'elaborazione della media, che eliminano dal vettore *t_in* gli elementi considerati *outliers*.

Come primo tentativo si è scelto di calcolare la media pesata considerando solamente gli AprilTag aventi dimensioni maggiori, nello specifico prendendo in considerazione i *markers* appartenenti alle due classi di dimensione maggiore tra quelle identificate. Le informazioni ricavate dall'identificazione di *markers* eccessivamente piccoli dal punto di vista della camera risulta rumorosa e poco affidabile, di conseguenza possono essere considerati *outliers*. Si noti come questo modo di operare non tenga in considerazione l'informazione derivante dai *markers*, ma sia basata su una scelta a priori. La funzione *just_two_size()* implementa l'eliminazione degli *outliers* mediante la logica descritta, ed è schematizzata nel diagramma in Figura 27. Questo algoritmo non ha purtroppo migliorato l'accuratezza della stima di posizione, dunque si sono implementati metodi più raffinati per l'individuazione di *outliers*.

SCARTO INTERQUARTILE In statistica il metodo dello scarto interquartile (IQR) è ampiamente utilizzato per l'individuazione di *outliers* in distribuzioni normali di dati. In primo luogo si divide il set di

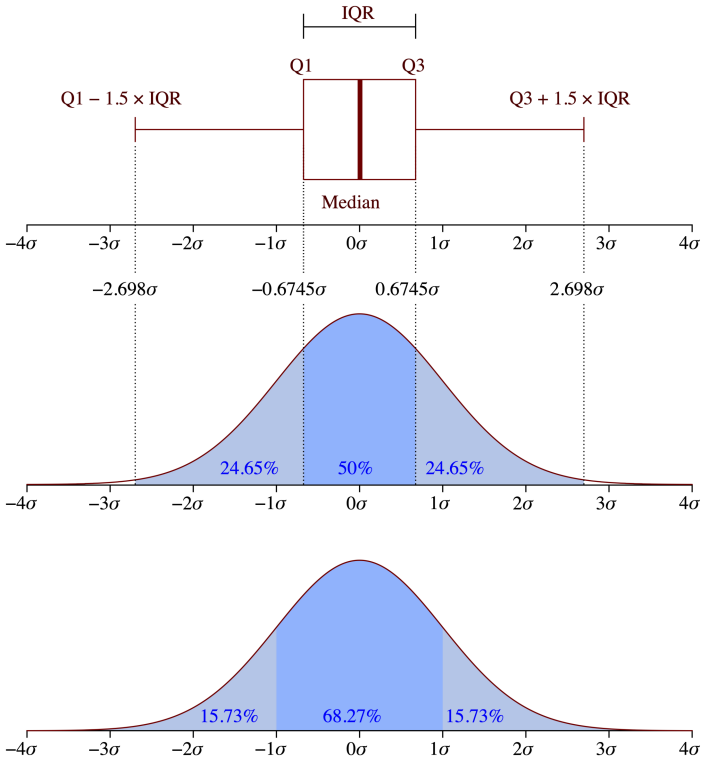


Figura 28: boxplot e densità di probabilità di una distribuzione normale di dati.

dati in quartili, ovvero valori che ripartiscono il set in quattro parti di uguale numerosità. Denominati i quartili come $Q1, Q2, Q3, Q4$ si calcola lo scarto interquartile $IQR = Q3 - Q1$. In secondo luogo si calcolano gli *outliers*, definendoli come i dati compresi nell'intervallo inferiore a $Q1 - 1.5 \cdot IQR$ e nell'intervallo superiore a $Q3 + 1.5 \cdot IQR$. In Figura 28 si rappresenta la procedura applicata a una distribuzione normale di dati.

Nel caso in esame, l'implementazione di questa strategia ha una complessità maggiore rispetto i canonici algoritmi di IQR, poiché le trasformazioni sono dati vettoriali. Si è scelto di individuare gli *outliers* sulla base dell'informazione di posizione, dunque un vettore composto da tre elementi $[x, y, z]$.

L'algoritmo è stato implementato a partire della seguente logica:

1. si ordina in modo crescente *t_in*, sulla base della coordinata x, y e z, definendo dunque tre nuovi vettori di *tuple*, denominati rispettivamente *t_in_x*, *t_in_y*, *t_in_z*;

2. si rimuovono gli *outliers* con il metodo IQR, per ogni vettore ordinato definito nel punto precedente;

3. si considerano *outliers* gli elementi che non sono contenuti nell'intersezione tra i vettori *t_in_x*, *t_in_y* e *t_in_z*.

Nell'implementazione reale, i dati delle trasformazioni vengono pesati sulla base della dimensione dell'AprilTag con cui sono stati ricavati. Uno schema logico della funzione *interquartile_range()* è riportato in Figura 29, che implementa quanto esposto.

DISTANZA EUCLIDEA Si è implementato un ulteriore algoritmo per l'identificazione e la rimozione degli *outliers*, basato sulla distanza euclidea tra la posizione ricavata dalle trasformazioni contenute in *t_in* e la posizione stimata mediante EKF da PX4 Autopilot. In particolare, la distanza euclidea è calcolata come:

$$d_i = \sqrt{(x_{i,in} - x_{EKF})^2 + (y_{i,in} - y_{EKF})^2 + (z_{i,in} - z_{EKF})^2} \quad (17)$$

dove le coordinate i-esime sono riferite alla i-esima posizione contenuta in *t_in*, mentre quelle con pedice EKF sono riferite alla posa calcolata da PX4 Autopilot a valle di EKF.

Affinché una trasformazione contenuta in *t_in* sia considerata *outlier*, la distanza precedentemente calcolata dev'essere maggiore di un valore di soglia:

$$\mathbf{T}_i \in \mathcal{O} \quad \Leftrightarrow \quad d_i(\mathbf{T}_i) > d_{max} \quad (18)$$

dove $\mathbf{T}_i$ è la trasformazione i-esima in ingresso, $\mathcal{O}$ è l'insieme degli *outliers* e d_{max} è la distanza di soglia. La distanza di soglia d_{max} è configurabile mediante il parametro *euc_dist_max*.

Per assicurarsi che una posa venga calcolata nel ciclo di *tf_callback*, si è implementato un meccanismo per il quale, se l'algoritmo elimina troppi elementi, si incrementa d_{max} e si rilancia il ciclo. In particolare, si confronta il rapporto tra il numero di elementi di *t_in* dopo aver eliminato gli *outliers* e il numero di elementi di *t_in* prima dell'eliminazione degli *outliers* con il valore di soglia definito dal parametro *euc_outlier_ratio*. Se il rapporto calcolato è maggiore del valore di soglia, si considera valida l'eliminazione degli *outliers*. In caso contrario, si incrementa la distanza d_{max}, sommandole la distanza specificata dal parametro *euc_dist_to_sum* e si procede nuovamente alla rimozione degli *outliers* con la distanza d_{max} appena calcolata. Con tale logica può verificarsi che il ciclo non termini, di conseguenza, per garantire l'integrità della funzione, è possibile scegliere un limite massimo di iterazioni per il quale si ripristina *t_in* alla versione precedente l'eliminazione degli *outliers*, specificato dal parametro *euc_dist_iteration*. Quest'ultimo meccanismo serve a prevenire il blocco dell'intero sistema e, nel caso intervenga, viene segnalata un'anomalia.

Nell'elenco seguente si riassumono i parametri di configurazione, regolabili dall'utente:

- *euc_dist_max* - massima distanza tra l'i-esima posizione in *t_in* e la stima mediante EKF, affinché l'i-esima trasformazione non sia considerata *outlier*;

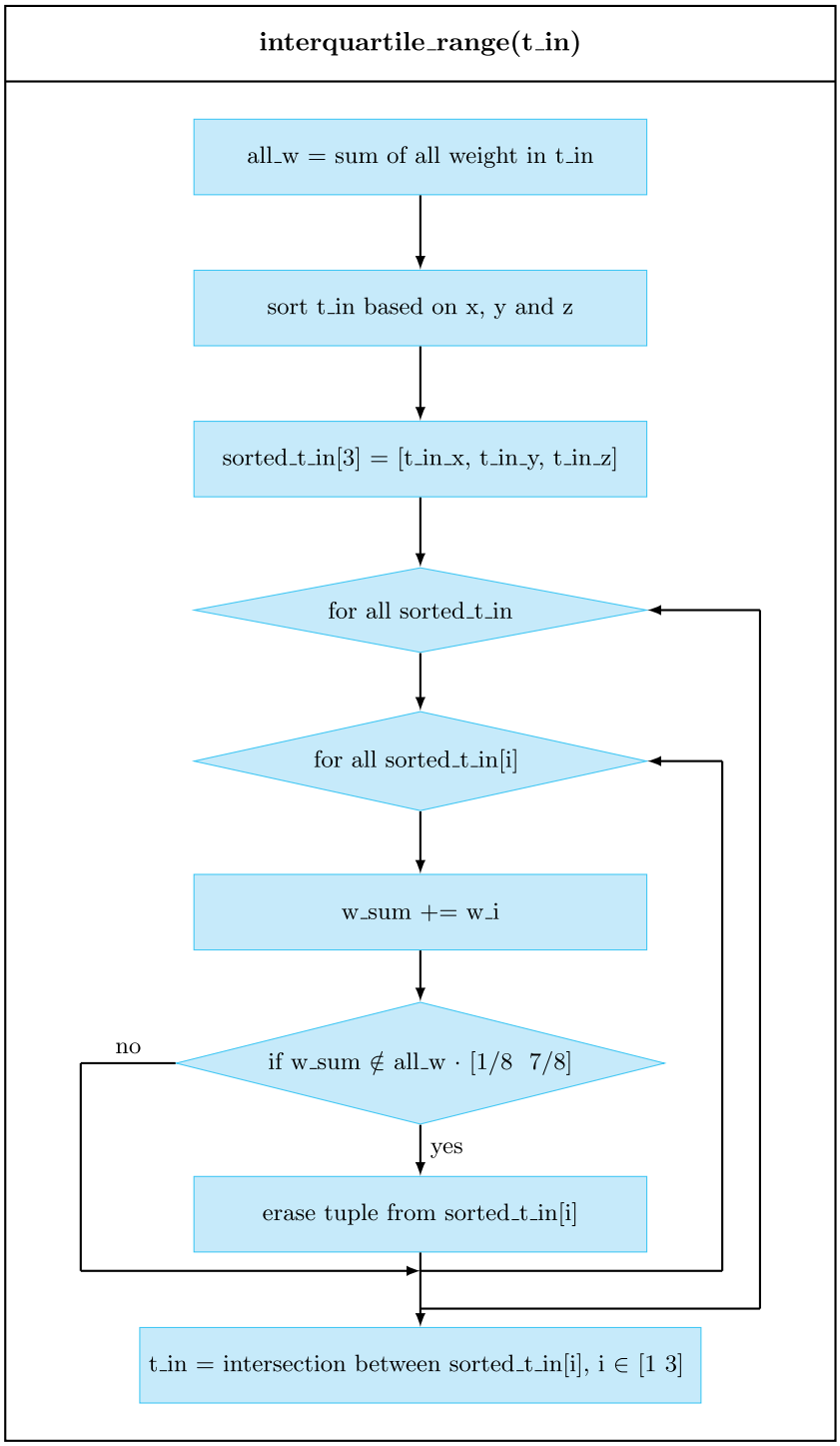


Figura 29: Diagramma logico della funzione *interquartile_range()*.

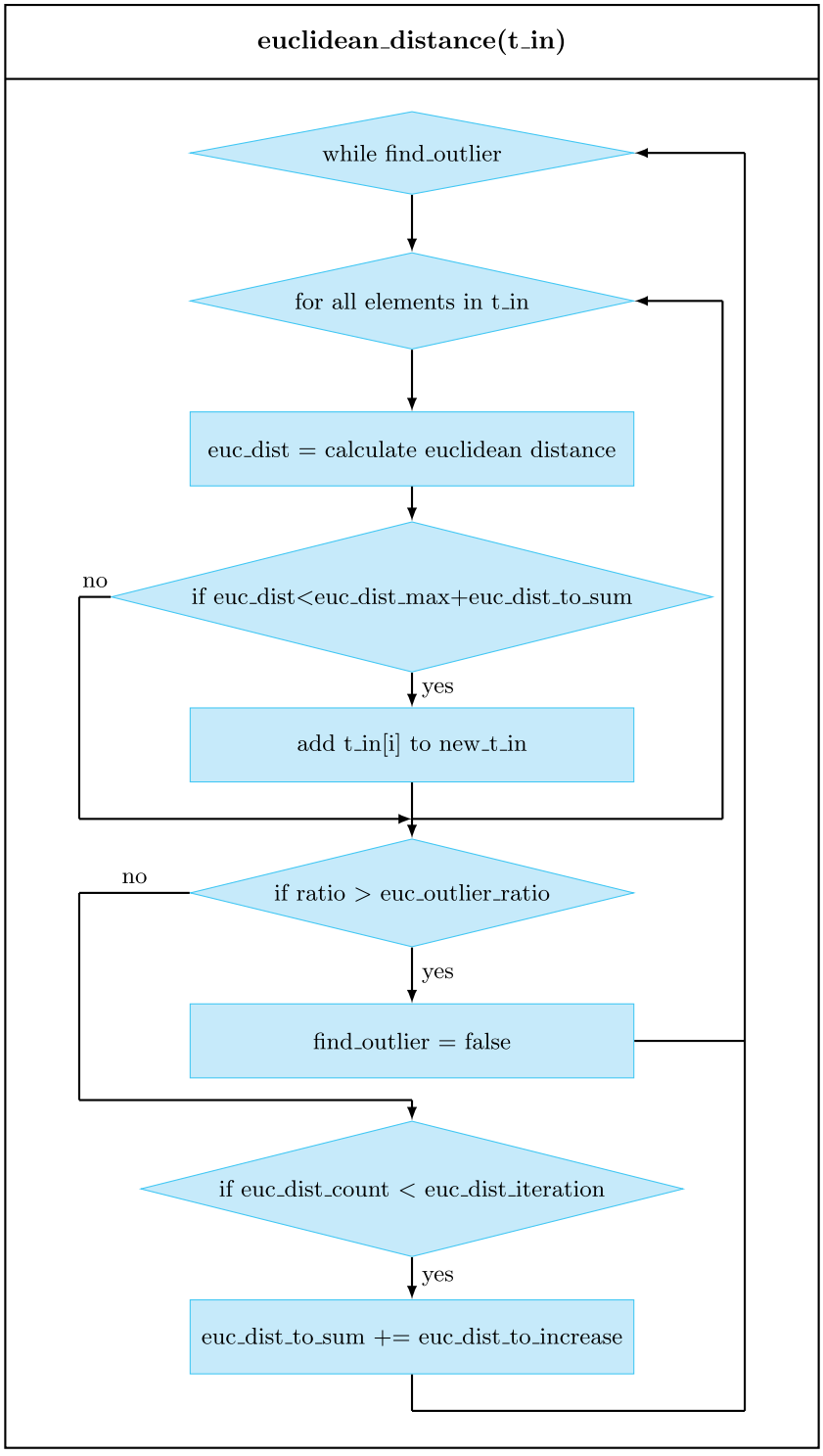


Figura 30: Diagramma logico della funzione *euclidean_distance()*.

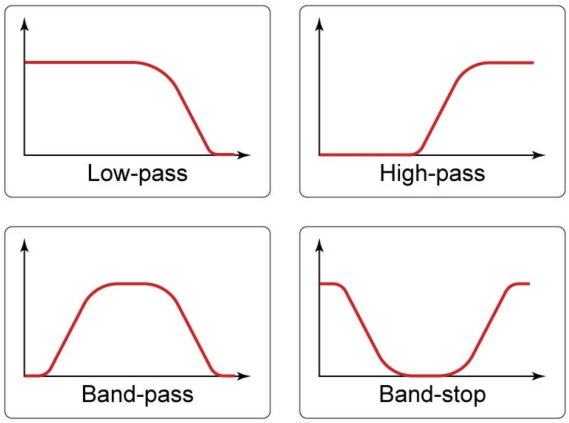


Figura 31: Andamento della risposta in frequenza di filtri passa basso, passa alto, passa banda e elimina banda.

- *euc_outlier_ratio* - valore di soglia del rapporto tra il numero di elementi di *t_in* dopo aver eliminato gli *outliers* e il numero di elementi di *t_in* prima dell'eliminazione degli *outliers*;
- *euc_dist_to_sum* - valore con il quale si incrementa d_{max} nel caso il rapporto tra *outliers* e trasformazioni totali sia troppo elevato;
- *euc_dist_iteration* - numero massimo di iterazioni del ciclo per l'eliminazione degli *outliers*.

L'implementazione dell'algoritmo avviene mediante la funzione *euclidean_distance()*, schematizzata nel diagramma logico in Figura 30.

3.6.4 *Filtro a media mobile*

Dato un certo segnale digitale campionato e quantizzato, è possibile implementare dei filtri digitali per elaborare numericamente l'informazione in ingresso. I filtri sono tipicamente utilizzati per attenuare determinate bande di frequenza indesiderate. Come per i filtri analogici, i filtri digitali possono essere definiti passa basso, passa alto, passa banda o elimina banda a seconda dell'andamento della propria risposta in frequenza, come riportato in Figura 31. Un filtro numerico è esprimibile con una equazione alle differenze:

$$\begin{aligned} y(k) = {} & b_0 x(k) + b_1 x(k-1) + \cdots + b_n x(k-n) + \\ & + a_1 y(k-1) + a_2 y(k-2) + \cdots + a_m y(k-m) \end{aligned} \tag{19}$$

dove $x(k)$ è l'ingresso digitale e $y(k)$ l'uscita. Quando tutti i coefficienti a_i sono nulli il filtro si definisce a risposta impulsiva finita FIR (*finite impulse response*). Se invece almeno un coefficiente a_i non è nullo il filtro si definisce a risposta impulsiva infinita IIR (*infinite impulse response*).

Un filtro con caratteristiche passa basso di tipo FIR è il **filtro a media mobile**:

$$y(k) = \frac{1}{N} \cdot \sum_{i=0}^{N-1} x(k-i) \tag{20}$$

dove N è l'ordine del filtro. Il filtro ha caratteristiche di tipo passa basso, ma la sua risposta si assesta dopo N periodi di campionamento. Come tutti i filtri non ha problemi di stabilità numerica, anche in caso di errata scelta dei coefficienti. Gli ingressi $x(k-1), x(k-2), \ldots, x(k-N)$ possono moltiplicati per dei pesi, così da pesare maggiormente l'ingresso più recente. Si definisce dunque il filtro a media mobile pesata:

$$y(k) = \frac{\sum_{i=0}^{N-1} w_i \cdot x(k-1)}{\sum_{i=0}^{N-1} w_i} \tag{21}$$

dove $w_1, w_2, \ldots, w_N$ sono i pesi.

La stima della posa ottenuta dal sistema di *visual odometry* presentava del rumore in alta frequenza, per attenuarlo si è dunque implementato un filtro a media mobile pesata. Dall'Equazione 21 si può notare come la definizione del filtro FIR corrisponde a una media pesata di N elementi. DI conseguenza, le funzioni *translation_average()* e *quaternion_average()* precedentemente definite per calcolare la media pesata della posa possono essere utilizzate per l'implementazione del filtro a media mobile. In Figura 32 si riporta lo schema della funzione *fir()*, che necessita come unico parametro di configurazione di *fir_weight[N]*, vettore dei pesi di dimensione N, nel quale l'N-esimo elemento pesa la trasformazione più recente. Si noti come il parametro precedentemente definito impone anche l'ordine del filtro, in base al numero di elementi del vettore. La funzione assegna ai vettori *quat_body* e *translation_body* rispettivamente quaternione medio e posizione media calcolati, che vengono codificati nel messaggio da inviare a PX4 Autopilot.

3.6.5 *Nodo AprilTag to visual odometry*

Le funzioni analizzate nelle sezioni precedenti sono implementate nel nodo ROS *apriltag to visual odometry*. Si è deciso di rendere possibile l'attivazione o la disattivazione di determinate funzioni aggiungendo alcuni parametri al file di configurazione del nodo. In particolare, i parametri principali presenti nel file di configurazione sono i seguenti:

- *just_bigger_one* - parametro booleano, se *true* viene considerata solo la trasformazione relativa all'AprilTag di ID minore;
- *just_two_size* - parametro booleano, se *true* si considerano solo le due maggiori taglie di AprilTag identificate, come esposto in sottosezione 3.6.3;

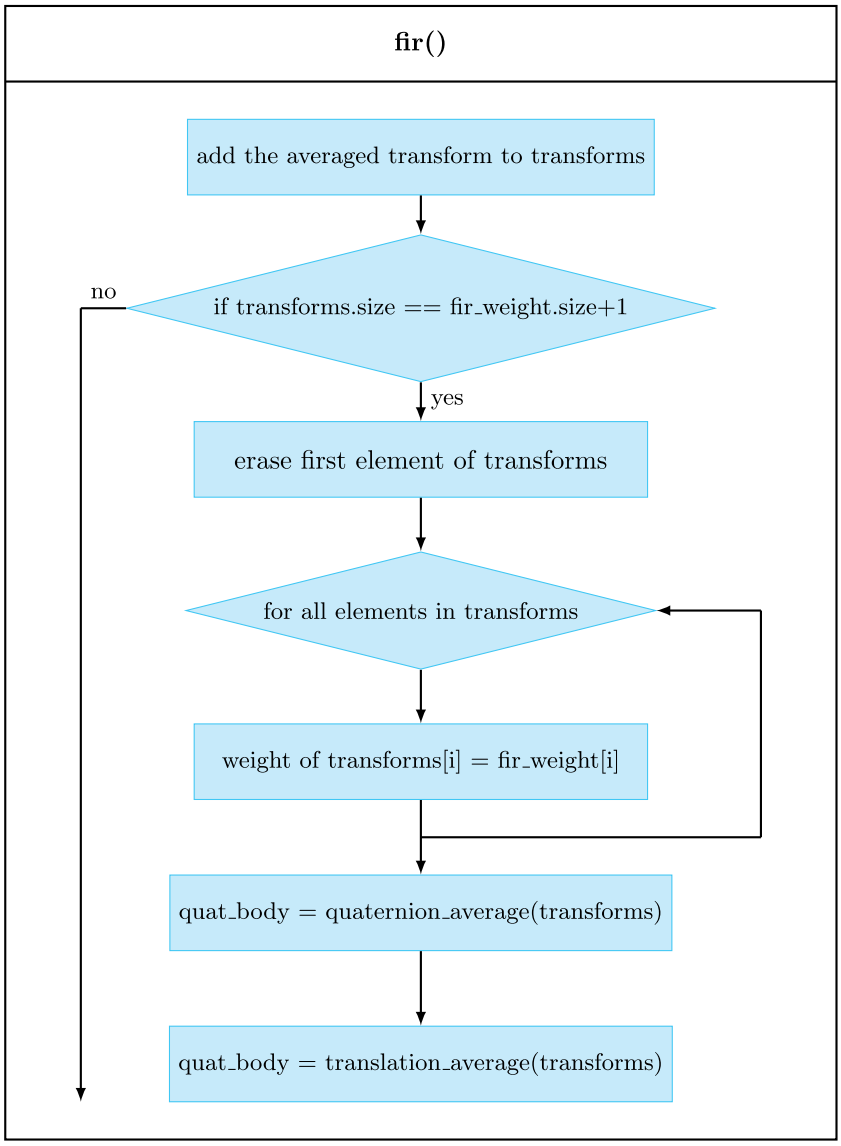


Figura 32: Diagramma logico della funzione *fir()*.

- *iqr_filter* - parametro booleano, se *true* si rimuovono gli *outliers* dal calcolo della media pesata tramite scarto interquartile, come esposto in sottosezione 3.6.3;
- *euc_dist_filter* - parametro booleano, se *true* si escludono gli *outliers* considerando la distanza euclidea rispetto alla stima di EKF, come esposto in sottosezione 3.6.3;
- *fir_weight* - vettore di interi, specifica i pesi per il filtro FIR e di conseguenza l'ordine N del filtro a media mobile, come approfondito in sottosezione 3.6.4;
- *tag_ids* - vettore di interi, elenco in ordine crescente degli identificativi degli AprilTag presenti nella mappa;

- *tags_locations_XL* - vettore di double, specifica le coordinate rispetto l'origine della mappa degli AprilTag di dimensione XL;
- *tags_locations_L* - vettore di double, specifica le coordinate rispetto l'origine della mappa degli AprilTag di dimensione L;
- *tags_locations_M* - vettore di double, specifica le coordinate rispetto l'origine della mappa degli AprilTag di dimensione M;
- *tags_locations_S* - vettore di double, specifica le coordinate rispetto l'origine della mappa degli AprilTag di dimensione S;

Un esempio di configurazione nel file *apriltag_virtual_map.yaml* è il seguente:

```
# filtering parameters
just_bigger_one: false          # consider only the bigger frame
euc_dist_filter: false          # compute euclidean distance for filtering
iqr_filter: true                # weighted median to delete the outliers
just_two_size: false            # consider only the two bigger sizes seen
euc_dist_max: 0.05              # euclidean distance limit for deleting outliers
euc_dist_to_increase: 0.01      # increase the euclidean distance limit if too
                                # frames are deleted
euc_outlier_ratio:  0.3         # ratio between the number of frames that have not
                                # been deleted and all the frames seen
fir_weight: [1, 1, 1, 1, 1]     # weight to apply to fir filter

#Specify the apriltag_map. Example:
#tag_ids: [first_id_XL, ... , last_id_XL, first_id_L, ... ,last_id_L, first_id_M,
     ... , last_id_M, first_id_S, ... , last_id_S]
#tags_locations_<size_tags>: [x_first, y_first, ... , x_last, y_last]

# Apriltag division:
# XL(0 --> 99)
# L(100 --> 399)
# M(400 --> 999)
# S(1000 --> ...)

tag_ids: [0, 1, 100, 101, 400, 401, 1000, 1001]
tags_locations_XL: [0.256, 0.256, 1.716, 0.256]
tags_locations_L: [0.621, 0.256, 0.986, 0.256]
tags_locations_M: [0.6205, 0.073, 0.803, 0.073]
tags_locations_S: [0.53, 0.0735, 0.7125, 0.0735]
```

4

RISULTATI SPERIMENTALI

In questo capitolo si espone il processo di validazione sperimentale del sistema di visione descritto nel Capitolo 3. In particolare, si sono sperimentati i diversi algoritmi delineati per comprendere quale fosse più performante in termini di accuratezza di localizzazione. Dopo una presentazione dell'intero sistema sperimentale in Sezione 4.1, si riportano i risultati dei test nelle sezioni successive, imponendo due diverse traiettorie al controllore di volo PX4 Autopilot.

4.1 DESCRIZIONE DELL'APPARATO SPERIMENTALE

L'apparato impiegato nelle prove di validazione si compone di due elementi macroscopici fondamentali:

- l'**UAV** - un multirotore comprendente il sistema di visione implementato per la localizzazione indoor;
- la **mappa** - composta da un certo numero di AprilTag a seconda della dimensione dell'ambiente di volo, come descritto in Sezione 2.3.

Il Dipartimento di Tecnica e Gestione dei sistemi industriali (DTG) dell'Università di Padova dispone di due multirotori *custom*, denominati **HR01** e **QR01**

4.1.1 *HR01*

L'esarotore HR01 è un robot aereo per scopi di ricerca, caratterizzato sia da componenti standard reperibili sul mercato che da componenti custom appositamente realizzati. L'HR01 si caratterizza per la presenza di un *companion computer* per la gestione dei dati derivanti dal sistema di visione e di una camera fissa orientata verso il pavimento per l'identificazione dei *markers*.

I principali componenti dell'esarotore sono i seguenti:

- *Tarot 680 pro* - telaio progettato per eliche da 13 ″ e caratterizzato da una lunghezza di interasse di 695 mm;
- *Turnigy 6600mAh 6S* - batteria ai polimeri di litio (Li-Po), tensione nominale di 22.2 V e capacità nominale di 6.6 A h;
- *Tarot 6S 380KV 4008* - motori brushless a magneti permanenti, progettati per una tensione nominale di 22.2 V;

Figura 33: Esarotore HR01.

Multirotore	Interasse motori [mm]	Peso [kg]
HR01	695	3.5
QR01	250	0.5

Tabella 4: Dimensioni e masse dei multirotori HR01 e QR01 a confronto

- *Tarot 1355 Carbon Propellers* - eliche in materiale composito caratterizzate da un diametro di 13 ″ e da un passo di 5.5 ″;
- *Holybro Tekko32 ESC 35A* - regolatore di velocità ESC (*Electronic Speed Controller*) per i motori, supporta una corrente nominale di 35 A;
- *Pixhawk 4* - controllore di volo progettato per eseguire il firmware PX4 Autopilot, comprende come sensoristica due IMU a 9 assi e un barometro;
- *Raspberry PI4 model B* - *companion computer* con installato il sistema operativo *Ubuntu 20.04* e le librerie ROS 2, nel quale sono implementati i nodi ROS per il sistema di *visual odometry*;
- *Intel RealSense Depth Camera D435* - stereocamera dalle dimensioni ridotte, collegata al *Raspberry* mediante porta USB 3;
- *RadioLink AT9S Pro* - radiocomando per controllare manualmente il velivolo
- *custom hardware* - supporti per camera e *companion computer* realizzati mediante stampa 3D.

In Figura 33 si riporta un immagine dell'esarotore HR01, mentre nella prima riga della Tabella 4 si riportano i parametri principali del

Figura 34: Quadrirotore QR01.

multirotore. Viste le dimensioni importanti del multirotore, si è preferito sperimentare inizialmente il sistema di visione con un velivolo di dimensioni minori, il QR01.

4.1.2 *QR01*

Il quadrirotore QR01 si distingue per le ridotte dimensioni, riportate nella seconda riga in Tabella 4. Il multirotore è stato assemblato a partire dal kit *Holybro* QAV250[1]. L'utilizzo di tale kit è consigliato anche dalla comunità PX4 [48].

In Figura 34 si riporta un'immagine del QR01, che è composto dai seguenti componenti principali:

- *Carbon fiber 250 airframe with hardware* - telaio in materiale composito dalle dimensioni compatte;
- *Turnigy 2200mAh 4S* - batteria ai polimeri di litio (Li-Po), tensione nominale di 14.8 V e capacità nominale di 2.2 A h;
- *DR2205 KV2300* - motori brushless a magneti permanenti;
- *5″ Plastic Props* - eliche in materiale polimerico tripala, caratterizzate da un diametro di 5 ″;
- *Fully assembled Power Management Board with ESCs* - un unico componente comprende la scheda per la distribuzione della potenza e i quattro ESC per l'azionamento dei motori brushless;
- *Pixhawk 4 Mini* - controllore di volo progettato per eseguire il firmware PX4 Autopilot, comprende come sensoristica due IMU a 9 assi;

1 http://www.holybro.com/product/pixhawk-4-mini-qav250-kit/

- *Raspberry PI4 model B - companion computer* con installato il sistema operativo *Ubuntu 20.04* e le librerie ROS 2, nel quale sono implementati i nodi ROS per il sistema di *visual odometry*;
- *Raspberry Pi Camera* - modulo camera orientato verso il pavimento, si collega al *Raspberry* mediante apposito connettore MIPI;
- *RadioLink AT9S Pro* - radiocomando per controllare manualmente il velivolo;
- *custom hardware* - carrello d'atterraggio e supporti per camera e *companion computer*, realizzati mediante stampa 3D.

Confrontando i componenti di HR01 e QR01, si può notare come i nodi per il sistema di visione vengano eseguiti sul medesimo dispositivo (*Raspberry PI4 model B*), tuttavia le camere connesse sono differenti.

4.2 TRAIETTORIE PER VALIDAZIONE SPERIMENTALE

4.2.1 *Offboard control*

Per comparare le prestazioni del sistema di visione con i differenti algoritmi presentati in Sezione 3.6, si è deciso di imporre delle traiettorie mediante il nodo *offboard control*[2]. Quest'ultimo impone dei riferimenti di posizione al controllore di PX4 Autopilot, generando una traiettoria di volo. In particolare, si utilizzano le due traiettorie definite in [17]. La prima è denominata **traiettoria quadrata**, nella quale il velivolo disegna un quadrato sul piano xy del *world frame*, mantenendo una quota costante lungo l'asse z dello stesso sistema di riferimento (ovvero un'altezza costante dal suolo). La seconda è la **traiettoria step**, per la quale il riferimento sull'asse x e y rimane costante, mentre il riferimento lungo l'asse z varia così da ottenere un volo stazionario a tre diverse altezze.

Per valutare la bontà della stima di posizione anche in volo stazionario, ogni traiettoria considerata si suddivide in movimentazioni. Nel caso di traiettoria quadrata una movimentazione corrisponde alla percorrenza di un lato del quadrilatero, invece per quella step una movimentazione corrisponde al moto tra un riferimento di quota a quello successivo. Per garantire un moto dolce al velivolo, il riferimento nelle componenti x, y e z durante una movimentazione ha un andamento polinomiale di quinto grado. Ambedue le traiettorie sono parametrizzate su un apposito file di configurazione *trajectoryParameters.yaml*, tramite i seguenti parametri:

- *test* - tipologia di traiettoria da seguire, da imporre uguale a 0 per quella quadrata e uguale a 1 per quella step;

2 https://github.com/C-square-unipd/offboard_control

- *MotionStart* - tempo di attesa in secondi da quando si esegue il nodo *offboard control* a quando si impongono i riferimenti di posizione per seguire la traiettoria, per consentire al multirotore di decollare;
- *Tmotion* - durata di ogni movimentazione in secondi;
- *Trest* - periodo di tempo nel quale il multirotore rimane in volo stazionario tra una movimentazione e la successiva.
- *StartingPoint* - posizione di partenza in *world frame* per l'esecuzione della traiettoria, espressa come $[x, y, z]$ in metri;
- *deltaX* - solo per traiettoria quadrata, lunghezza del lato parallelo all'asse x in metri;
- *deltaY* - solo per traiettoria quadrata, lunghezza del lato parallelo all'asse y in metri;
- *deltaZ* - solo per traiettoria step, lunghezza delle singole movimentazioni in metri.

Si riporta di seguito il file configurato per l'esecuzione delle due traiettorie scelte, nel quale si può notare il valore dei parametri precedentemente esposti utilizzati per i test di volo. In fase di test, si può decidere se eseguire la traiettoria quadrata o quella step commentando parte del codice.

```
offboard_control:
ros__parameters:
  debug: true
  ## Square test
  # test: 0
  # MotionStart: 15.0
  # Tmotion: 10.0
  # Trest: 2.0
  # StartingPoint: [3.52, 5.61, -0.8]
  # deltaX: 2.0
  # deltaY: 2.0

  ## Step test
  test: 1
  MotionStart: 15.0
  Tmotion: 3.0
  Trest: 6.0
  StartingPoint: [2.41, 4.9, -0.8]
  deltaZ: 0.3
```

4.2.2 *Acquisizione e analisi dei dati*

Per acquisire i dati relativi alle traiettorie si è utilizzato il comando `rosbag`, che permette di salvare i messaggi che transitano su determinati *topics* in un database locale. In particolare, si è voluto registrare i dati relativi ai seguenti *topics*:

- */fmu/trajectory_setpoint/in* - nel quale transitano i messaggi relativi al riferimento di posizione da imporre a PX4 Autopilot, al fine di generare la traiettoria desiderata;
- */fmu/vehicle_odometry/out* - nel quale transitano i messaggi relativi alla posa del drone a valle del filtro di Kalman di PX4 Autopilot.

Una volta acquisiti i dati mediante il comando `rosbag`, si può generare un file *comma separeted value* (csv) tramite il software *PlotJuggler*. A partire dal file csv, tramite *Matlab* si sono innanzitutto creati dei grafici per un confronto qualitativo tra i diversi algoritmi. Nei tratti stazionari si sono invece calcolati dei valori statistici per confrontare quantitativamente le prestazioni dei differenti algoritmi.

4.3 PRESENTAZIONE DEI RISULTATI

Nella restante parte del capitolo si presentano i risultati ottenuti attivando differenti combinazioni di algoritmi, focalizzandosi sulle due differenti traiettorie e sui due multirotori. Per alleggerire la notazione, si abbrevia il nome degli algoritmi secondo il seguente elenco:

- JBG - *Just Bigger One*, algoritmo che considera solo uno tra i *markers* di dimensione maggiore;
- EUC - *EUClindean distance*, algoritmo che rimuove gli *outliers* mediante distanza euclidea;
- IQR - *InterQuartile Range*, algoritmo che rimuove gli *outliers* mediante scarto interquartile;
- FIR - *Finite Impulse Response*, algoritmo che implementa il filtro a media mobile.

Anche per le traiettorie si introducono delle abbreviazioni:

- T0 - traiettoria quadrata;
- T1 - traiettoria step.

In primo luogo si sono scelti i parametri di configurazione per i differenti algoritmi, in particolare per l'algoritmo EUC:

- *euc_dist_max* è stato impostato a $0.05\,\mathrm{m}$, valutando il rapporto tra *outliers* e trasformazioni totali con tale valore e l'effettiva accuratezza della stima;
- *euc_dist_to_increase* è stato impostato a $0.01\,\mathrm{m}$, valutando il numero di iterazioni del ciclo per l'esclusione degli *outliers*;
- *euc_outlier_ratio* è stato impostato a 0.3, quindi in presenza di un eliminazione del 70% delle trasformazioni in *t_in* si itera nuovamente per il calcolo degli *outliers*.

Per quanto concerne l'algoritmo FIR, il parametro *fir_weight* è stato impostato a $[1, 1, 1, 1, 1]$, implementando così un filtro a media mobile di quinto ordine.

Nelle sezioni successive i risultati sono esposti in ordine decrescente di accuratezza, partendo dunque dall'algoritmo con il quale si sono ottenuti i risultati peggiori. In particolare, l'ordinamento è risultato il medesimo per le differenti combinazioni di traiettorie e multirotori. I test sono dunque esposti con il seguente ordine nelle sezioni successive:

1. JBG - si riportano sia i risultati con filtro FIR attivato che quelli con filtro FIR disattivato;
2. EUC - si riportano solamente i risultati con filtro FIR attivato, poiché si è sempre notata una migliore accuratezza;
3. IQR - si riportano solamente i risultati con filtro FIR attivato, poiché si è sempre notata una migliore accuratezza.

Per l'analisi statistica, riportata in Sezione 4.8, si considerano invece tutte le combinazioni tra algoritmi per la rimozione degli *outliers* e filtro FIR.

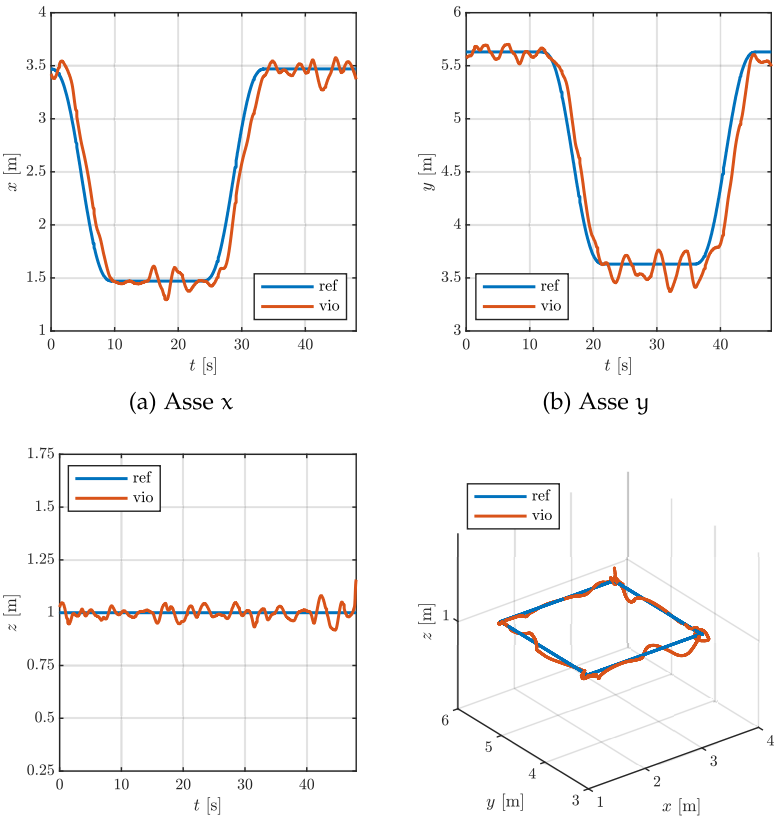
4.4 QR01 - TEST TRAIETTORIA QUADRATA

Le prime prove di volo sono state eseguite con il quadrirotore QR01, imponendo l'esecuzione della traiettoria quadrata. Modificando i parametri del file di configurazione di *apriltag to visual odometry*, si è eseguita la traiettoria quadrata con le varie combinazioni di algoritmi esposti in Sezione 3.6. Si presentano ora i risultati ottenuti, a partire dall'algoritmo più semplice, ovvero quello che considera solo uno tra gli AprilTag di dimensione maggiore inquadrati dalla camera.

4.4.1 *AprilTag di dimensione maggiore (JBG)*

Per scegliere di attivare questo algoritmo per la stima di posizione è sufficiente impostare a `true` il parametro *just_bigger_one* nel file di configurazione di *apriltag to visual odometry*. Se si imposta a `true` questo parametro, vengono automaticamente ignorati gli altri algoritmi per la rimozione degli *outliers*, poiché si considera solo il *marker* con l'ID minore tra quelli identificati, che corrisponde a uno tra gli AprilTag di dimensione maggiore nel campo visivo della camera. Questo stesso risultato si otteneva anche con la prima versione del nodo, scritta da Segattini [3].

La stima di posizione risulta particolarmente rumorosa, come si può notare dai grafici in Figura 35. Ci si aspetta un ritardo tra riferimento e stima di posizione, tuttavia, soprattutto nei tratti in cui



(a) Asse x

(b) Asse y

(c) Asse z

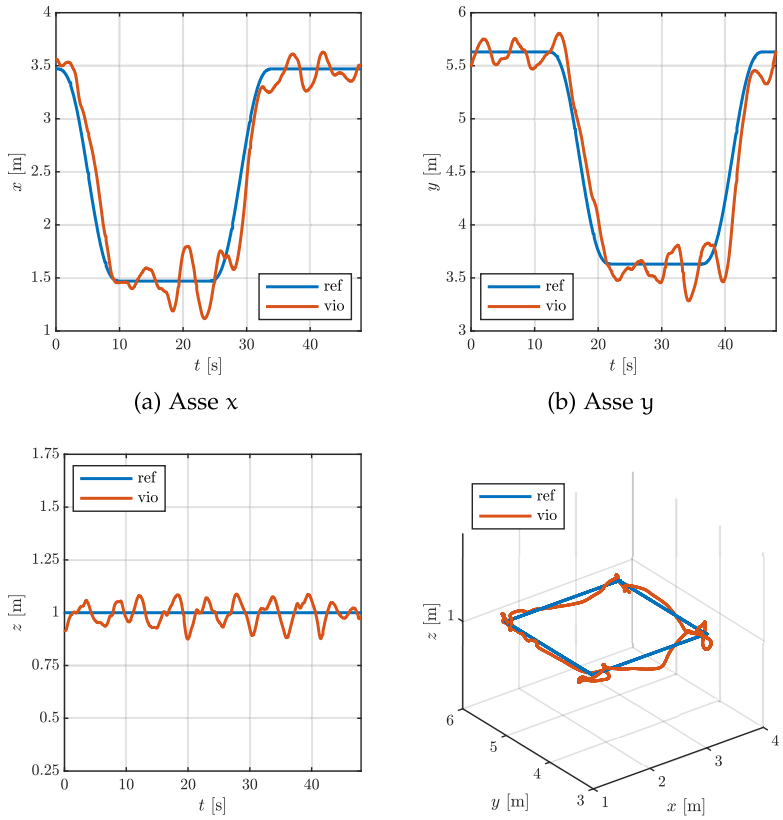
(d) Grafico tridimensionale

Figura 35: Stima di posizione con JBO, To e QR01.

il riferimento è costante in un singolo asse, la stima ha una risposta oscillatoria non accettabile, caratterizzata da un errore massimo di 20 cm, 26 cm e 8 cm rispettivamente per la componente x, y e z. Sia gli andamenti che i valori dell'errore massimo sono coerenti con quanto riportato in [17], nel quale si confrontava l'andamento della traiettoria stimata con l'algoritmo OBG con quella di un sistema di *motion capture*.

Dal grafico tridimensionale si evince come il controllore con tale stima di posizione in ingresso fatichi a imporre la traiettoria quadrata desiderata.

Per migliorare l'accuratezza usando lo stesso algoritmo, si è attivato il filtro a media mobile di quinto ordine. Come esposto in Sottosezione 3.6.4, il filtro a media mobile è un filtro digitale con caratteristiche passa basso, spesso utilizzato per filtrare misure rumorose. In Figura 36 si riportano gli andamenti per ogni asse della stima della posizione del multirotore. Tale stima con l'attivazione del filtro FIR risulta ancora più rumorosa su ogni asse, con sovraelongazioni e picchi ancora più distanti dal riferimento. In particolare, l'errore massimo è di 32 cm, 35 cm e 12 cm rispettivamente per la componente x, y e z. Un aumento medio del 48% rispetto quanto riscontrato senza at-



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

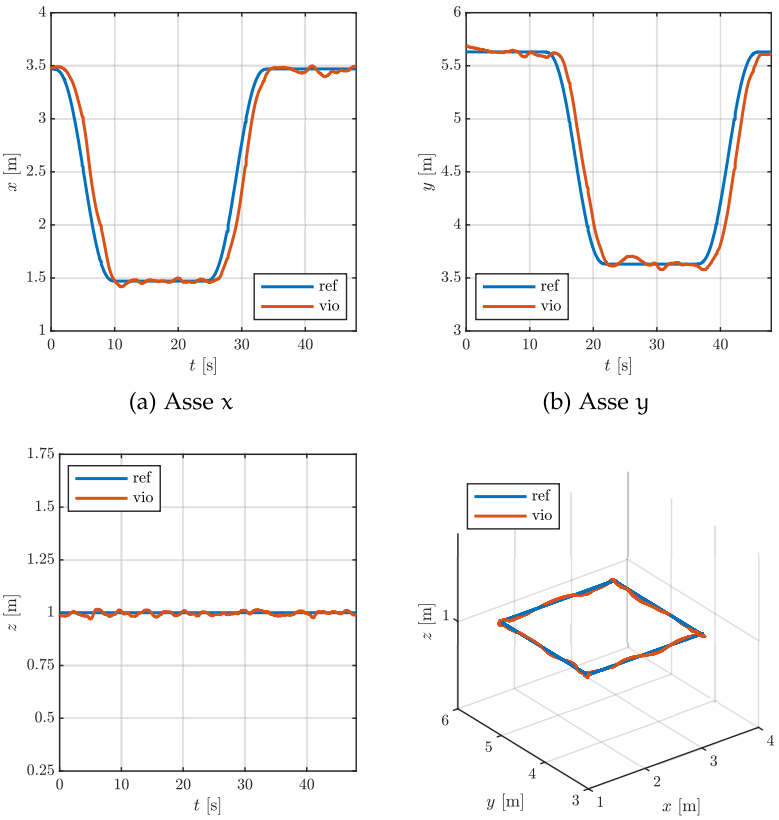
Figura 36: Stima di posizione con JBO+FIR, To e QR01.

tivare il filtro FIR. Si può trovare una spiegazione a tale fenomeno considerando che, nel caso una misura di posizione calcolata in un certo istante di tempo sia soggetta a un grande errore, pesa per il calcolo della stima in uscita per cinque istanti di campionamento, grazie al filtro a media mobile. Di conseguenza, il controllore di posizione corregge la traiettoria basandosi su una stima non accurata per più istanti di tempo, che può causare un aumento dell'errore massimo. Ci si aspetta dunque che il filtro FIR migliori la stima di posizione utilizzando innanzitutto degli algoritmi che rimuovano gli *outliers*.

4.4.2 *Rimozione outliers tramite distanza euclidea (EUC) e FIR*

Per quanto riguarda gli algoritmi per la rimozione degli *outliers*, si è notata un'accuratezza media maggiore del 20% per la stima di posizione nelle prove svolte con il filtro a media mobile attivato rispetto le medesime prove con il filtro disattivato. Per tali algoritmi si riportano dunque solo i grafici della stima di posizione con il filtro a media mobile attivato, al fine di alleggerire la trattazione.

Per attivare questo algoritmo per la stima di posizione è sufficiente impostare a `true` il parametro *euc_dist_filter* nel file di configurazione



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

Figura 37: Stima di posizione con EUC+FIR, To e QR01.

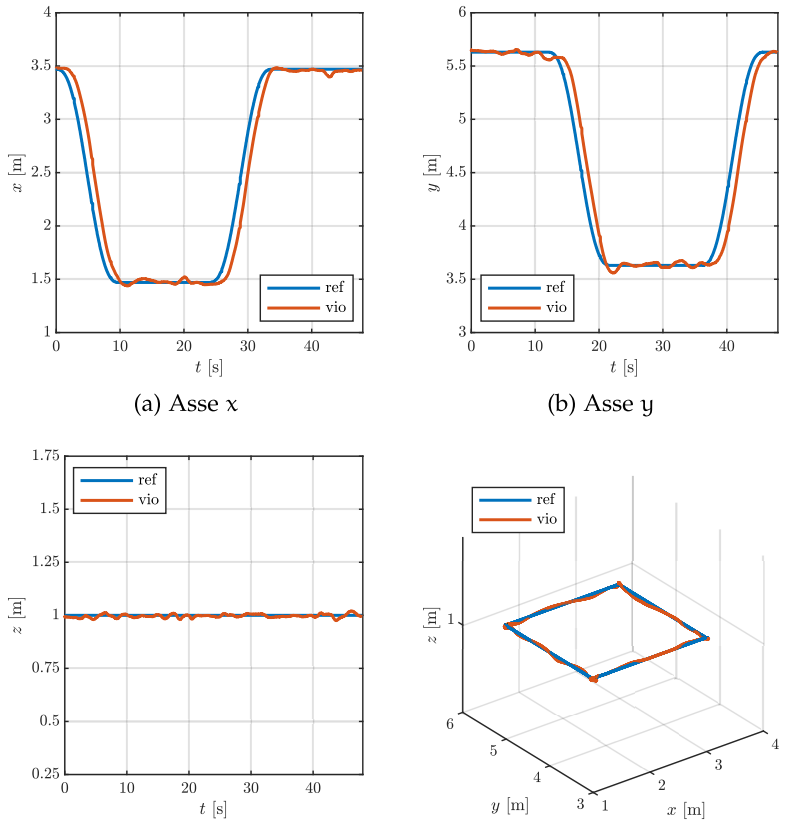
di *apriltag to visual odometry*. L'algoritmo *euclidean distance* rimuove gli *outliers* sulla base della distanza euclidea tra le trasformazioni in ingresso e l'ultima posizione stimata da EKF (Sottosezione 3.6.3).

In Figura 37 si riporta l'andamento della stima di posizione. Si nota da subito un importante miglioramento rispetto all'algoritmo precedentemente trattato: la stima di posizione segue accuratamente il riferimento, tuttavia sono ancora presenti delle oscillazioni, particolarmente evidenti nelle fasi stazionarie delle componenti x e y. Si riscontra un errore massimo di 5 cm, 7 cm e 2 cm rispettivamente per la componente x, y e z della stima di posizione, complessivamente minore del 74% rispetto quello calcolato con l'algoritmo JBO. Dal grafico tridimensionale si evince come la traiettoria venga seguita fedelmente.

4.4.3 *Rimozione outliers tramite IQR e FIR*

La rimozione degli *outliers* tramite scarto interquartile ha permesso di ottenere i migliori risultati in termini di accuratezza, probabilmente a causa del fondamento statistico dell'algoritmo (Sottosezione 3.6.3).

L'andamento della stima di posizione è riportato in Figura 38. Gli



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

Figura 38: Stima di posizione con IQR+FIR, To e QR01.

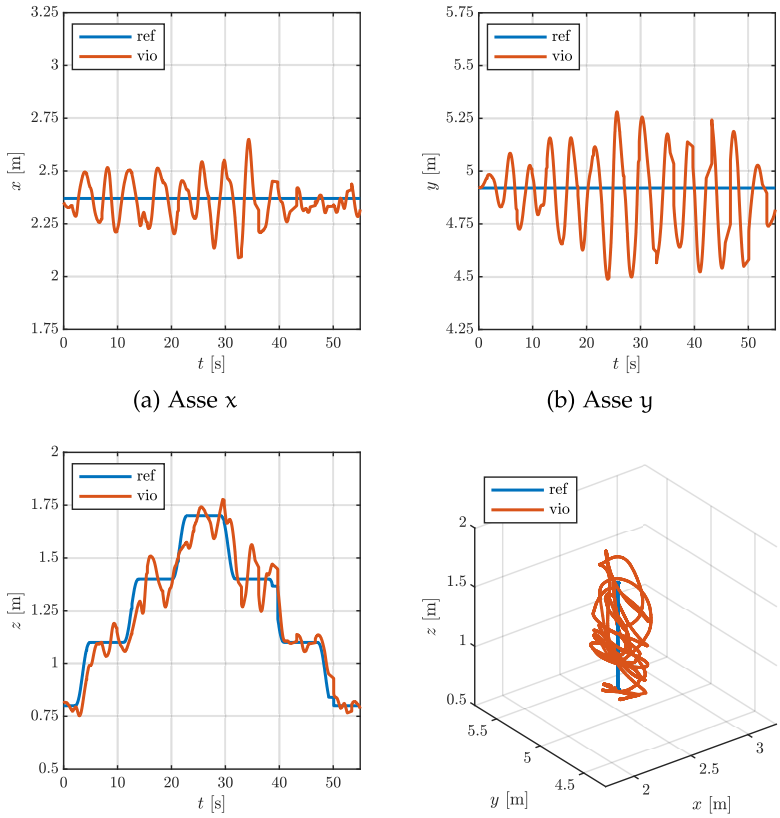
errori massimi per le componenti x, y e z della posizione risultano rispettivamente pari a 4 cm, 7 cm e 2 cm, comparabili a quanto riscontrato con l'algoritmo EUC. Da un confronto qualitativo con i grafici ottenuti tramite l'algoritmo EUC, si riscontra come, soprattutto per l'asse y, l'andamento della stima presenti oscillazioni di ampiezza più contenuta.

Dall'analisi del grafico tridimensionale, si nota come la stima sia ben sovrapposta al riferimento per l'intera traiettoria, anche negli angoli del quadrato dove il velivolo esegue un volo stazionario.

4.5 QR01 - TEST TRAIETTORIA STEP

In una seconda serie di test, si è imposto al multirotore QR01 di eseguire la traiettoria step. Tale traiettoria è stata pensata per misurare l'accuratezza del sistema di *visual inertial odometry* nella condizione di *hovering* statico, a tre differenti altezze. La medesima traiettoria è stata eseguita attivando differenti combinazioni tra gli algoritmi implementati e nelle sezioni successive si riportano i risultati ottenuti.

Si evidenzia che i grafici per gli assi x e y riportati in questa sezione hanno una scala differente rispetto a quelli in Sezione 4.4. Con la tra-



(a) Asse x

(b) Asse y

(c) Asse z

(d) Grafico tridimensionale

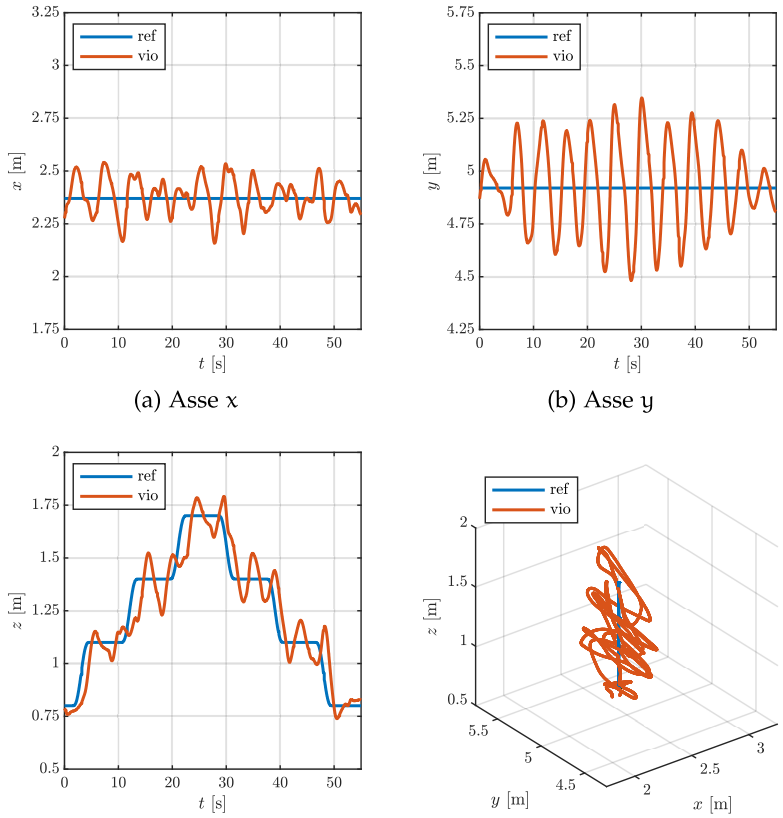
Figura 39: Stima di posizione con JBO, T1 e QR01.

iettoria step i riferimenti su x e y sono costanti, dunque si è preferito diminuire l'intervallo tra il limite superiore e inferiore degli assi, così da evidenziare meglio l'andamento e l'entità dell'errore.

4.5.1 *AprilTag di dimensione maggiore (JBO)*

Attivando questo algoritmo si considera solamente uno tra gli AprilTag di dimensione maggiore. In Figura 39 si riporta l'andamento della stima di posizione con attivato solamente tale algoritmo, mentre in Figura 40 si è attivato anche il filtro a media mobile.

Analizzando gli andamenti, non si nota una particolare variazione relativamente all'attivazione e la disattivazione del filtro a media mobile, tuttavia in entrambi i casi la stima è scadente, caratterizzata da grandi errori di posizione su tutti e tre gli assi. In particolare, l'errore massimo relativo alla componente x, y e z è rispettivamente di 25 cm, 43 cm e 15 cm con filtro FIR disattivato e di 22 cm, 42 cm e 17 cm con filtro FIR attivato. Per l'asse x l'attivazione del filtro a media mobile migliora l'entità dei picchi di errore tra stima e riferimento, mentre per l'asse y l'attivazione del filtro non porta miglioramenti. Si nota, tuttavia, come in ambedue i grafici l'errore della componente y



(a) Asse x (b) Asse y

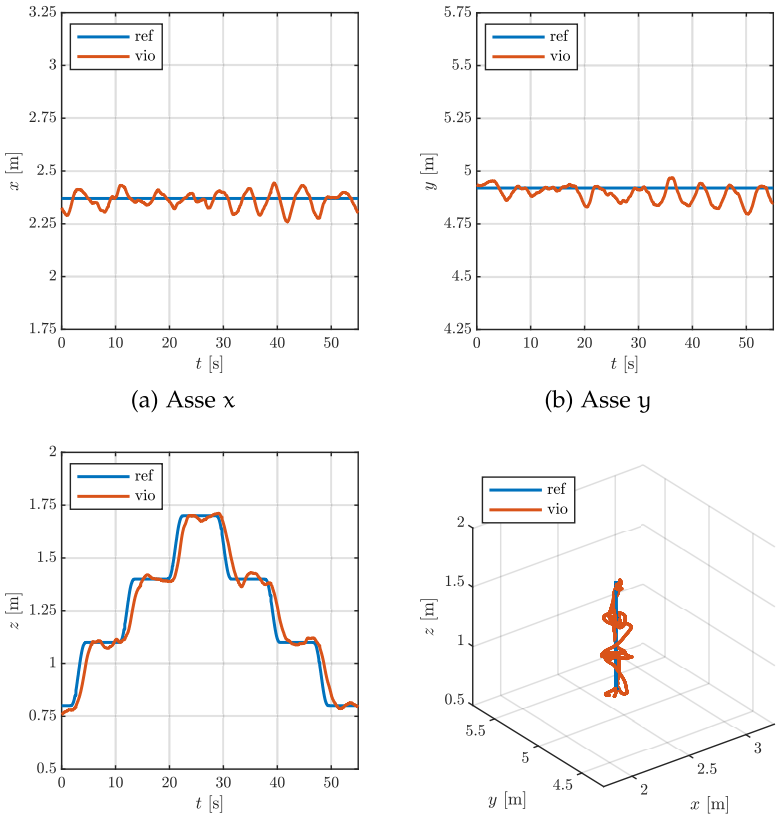
(c) Asse z (d) Grafico tridimensionale

Figura 40: Stima di posizione con JBO+FIR, T1 e QR01.

sia maggiore di quello della componente x. Tale comportamento può essere giustificato analizzando le caratteristiche strutturali del multirotore QR01: come si nota dalla Figura 34, la massa nel velivolo è distribuita principalmente sull'asse x del *body frame*. Di conseguenza, il momento d'inerzia J_x, calcolato lungo l'asse x, risulta minore rispetto al momento d'inerzia J_y. Come ribadito precedentemente, i multirotori utilizzati per la validazione sperimentale sono sistemi sottoattuati, dunque per ottenere una traslazione in direzione y è necessaria una rotazione del velivolo lungo l'asse x. Essendo J_x minore, la dinamica meccanica di rotazione angolare lungo l'asse x è più veloce, di conseguenza un errore sulla stima di posizione in direzione y può generare oscillazioni di maggiore entità.

Anche per quanto riguarda l'asse z le oscillazioni sono importanti, sia nelle fasi di salita che in quelle di discesa. Confrontando i grafici relativi all'asse z con quelli relativi all'asse y, si può notare come raggiungendo la quota massima di 1.7 m le oscillazioni aumentino d'intensità, a causa della maggiore distanza tra camera e mappa di AprilTag. Dal grafico tridimensionale si evince come la traiettoria non sia seguita accuratamente dal multirotore.

In [17] si esegue la medesima traiettoria con lo stesso algoritmo,



(a) Asse x

(b) Asse y

(c) Asse z

(d) Grafico tridimensionale

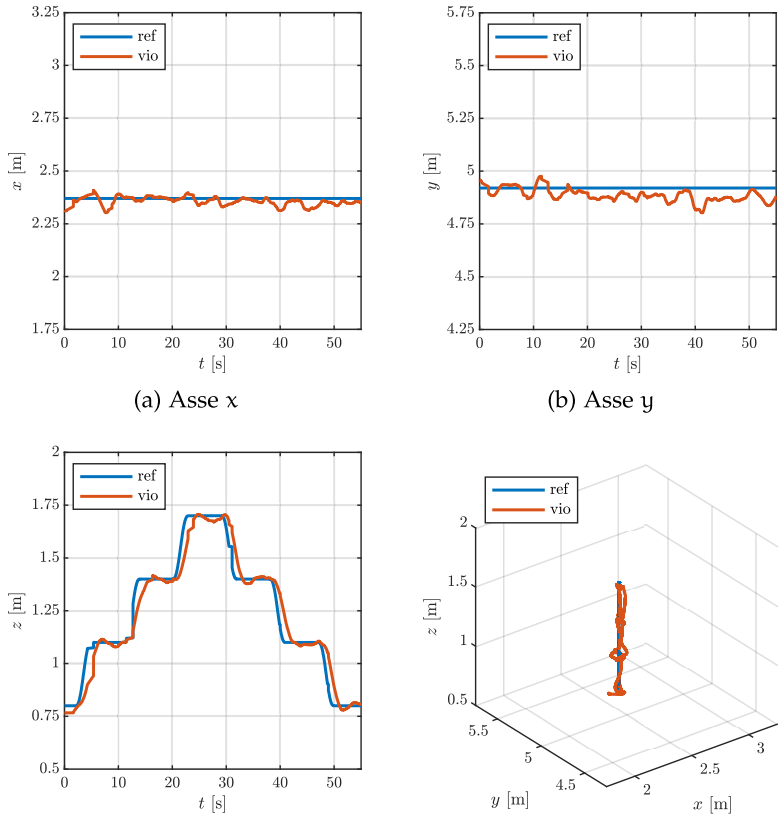
Figura 41: Stima di posizione con EUC+FIR, T1 e QR01.

e si può notare come gli andamenti della stima di posizione risultino molto disturbati, con oscillazioni di ampiezza picco-picco pari a 70 cm sia nella componente x che nella componente y, paragonabili a quelle riscontrate in questo test. Comparando la stima di posizione ottenuta tramite il sistema di *visual inertial odometry* con quella ottenuta da un sistema di *motion capture*, si riscontra un errore massimo pari a circa 30 cm per le componenti x e y.

4.5.2 *Rimozione outlier tramite distanza euclidea (EUC) e FIR*

Anche per quanto riguarda la traiettoria step, attivando gli algoritmi per la rimozione degli *outliers* si è notato un incremento di accuratezza della stima di posizione con il filtro a media mobile attivato, pari al 51 %. Per tali algoritmi si riportano dunque solo i grafici della stima di posizione con il filtro a media mobile attivato, al fine di alleggerire la trattazione.

L'andamento della stima di posizione è riportato in Figura 41. L'errore massimo rilevato per le componenti x, y e z risulta rispettivamente pari a 12 cm, 12 cm e 5 cm, una diminuzione del 78% se comparato all'errore massimo ottenuto con l'algoritmo JBO. La diminuzione



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

Figura 42: Stima di posizione con IQR+FIR, T1 e QR01.

dell'errore massimo è molto elevata e si riscontra anche analizzando qualitativamente gli andamenti. Anche alla quota di 1.7 m la stima è accurata e non presenta un comportamento differente rispetto alle quote più basse. Analizzando il grafico tridimensionale si distinguono ora le tre fasi di hovering, che sono inevitabilmente maggiormente soggette a delle correzioni da parte del controllore.

Sebbene la traiettoria segua accuratamente il riferimento, soprattutto negli assi x e y si riscontrano comunque delle oscillazioni, che raggiungono una distanza picco-picco di 20 cm.

4.5.3 *Rimozione outliers tramite IQR e FIR*

L'attivazione dell'algoritmo per la rimozione degli *outliers* tramite scarto interquartile ha permesso di ottenere i migliori risultati in termini di accuratezza. Il medesimo risultato si era ottenuto anche per la traiettoria quadrata, come visto in Sezione 4.4.

In Figura 42 si riporta l'andamento delle tre componenti della stima di posizione. Confrontando tali andamenti con i grafici ottenuti precedentemente tramite l'utilizzo di diversi algoritmi, si nota da subito come le oscillazioni siano minori, in particolare si riscontra un

errore massimo di 7 cm, 12 cm e 2 cm. Anche in questo caso la stima della componente y della posizione sembra avere un andamento più disturbato: le oscillazioni rimangono di bassa entità, tuttavia sembra che all'aumentare del tempo ci sia un leggero fenomeno di *drift*, probabilmente dovuto alla componente inerziale della stima, in quanto soggetta a tale fenomeno.

Analizzando invece il grafico tridimensionale, l'inseguimento del riferimento di posizione è molto accurato, risultando in una traiettoria con solo leggeri scostamenti. Se si compara l'andamento della stima di posizione con quello ottenuto con i precedenti algoritmi, si nota una maggiore accuratezza con l'algoritmo per la rimozione degli *outliers* mediante scarto interquartile attivato.

4.6 HR01 - TEST TRAIETTORIA QUADRATA

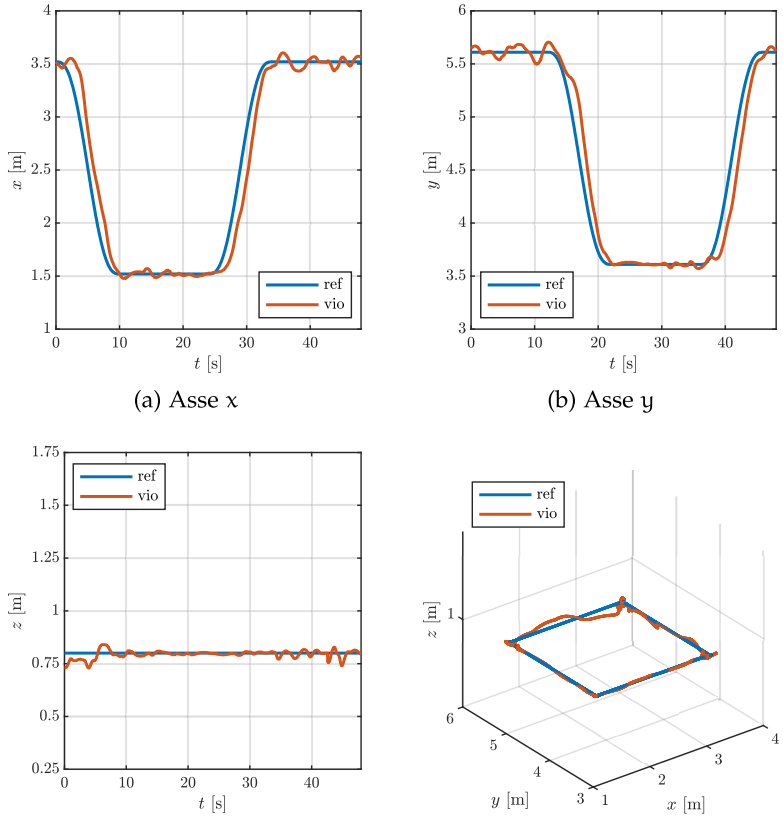
Le medesime prove di volo presentate precedentemente per il QR01 sono state eseguite anche con l'esarotore HR01, in primo luogo imponendo l'esecuzione della traiettoria quadrata.

4.6.1 *AprilTag di dimensione maggiore (JBO)*

Si riporta in Figura 43 l'andamento della stima di posizione al variare del riferimento per ottenere la traiettoria quadrata. Analizzando i grafici, l'inseguimento del riferimento di posizione imposto è preciso, sebbene nei tratti stazionari per le singole componenti siano presenti delle oscillazioni. L'errore massimo della stima equivale a 8 cm, 12 cm e 4 cm rispettivamente per la componente x, y e z.

Per migliorare l'accuratezza usando lo stesso algoritmo, si è attivato il filtro a media mobile di quinto ordine. In Figura 44 si riportano gli andamenti della stima di posizione con il filtro attivato. Confrontando la traiettoria complessiva rappresentata dal grafico tridimensionale con quella ottenuta precedentemente senza FIR attivo (riportata in Figura 43), non si notano particolari differenze. In particolare, l'errore massimo ammonta a 9 cm per la componente x, a 14 cm per la componente y e a 5 cm per la componente z. Si può notare un aumento dell'errore massimo, pari al 18%, tuttavia, da un'analisi qualitativa, l'andamento delle traiettorie stimate è paragonabile.

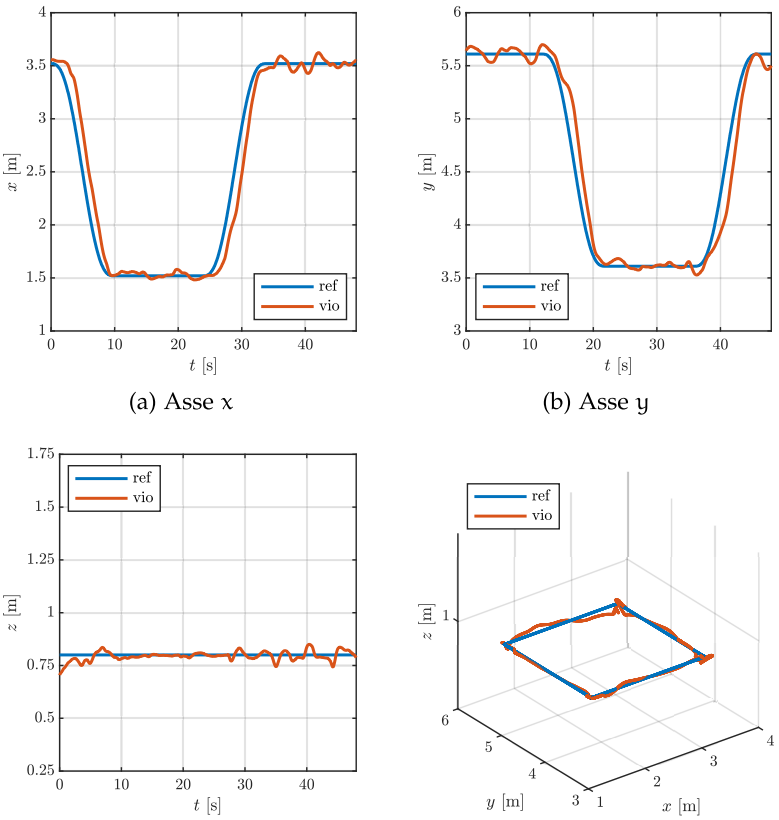
È invece importante notare come le stime di posizione tramite questo algoritmo, attivato nel sistema di volo dell'HR01, siano complessivamente molto più accurate rispetto al medesimo algoritmo attivato su QR01 (i grafici relativi alla stima di posizione con QR01 sono riportati in Figura 35 e 36, con il filtro FIR rispettivamente disattivato e attivato). In particolare, si nota una diminuzione dell'errore massimo pari al 54% con filtro FIR disattivato e al 63% con filtro FIR attivato. Questo differente comportamento tra i due multirotori può essere motivato considerando le masse, le dimensioni e i risultanti momenti



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

Figura 43: Stima di posizione con JBO, To e HR01.



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

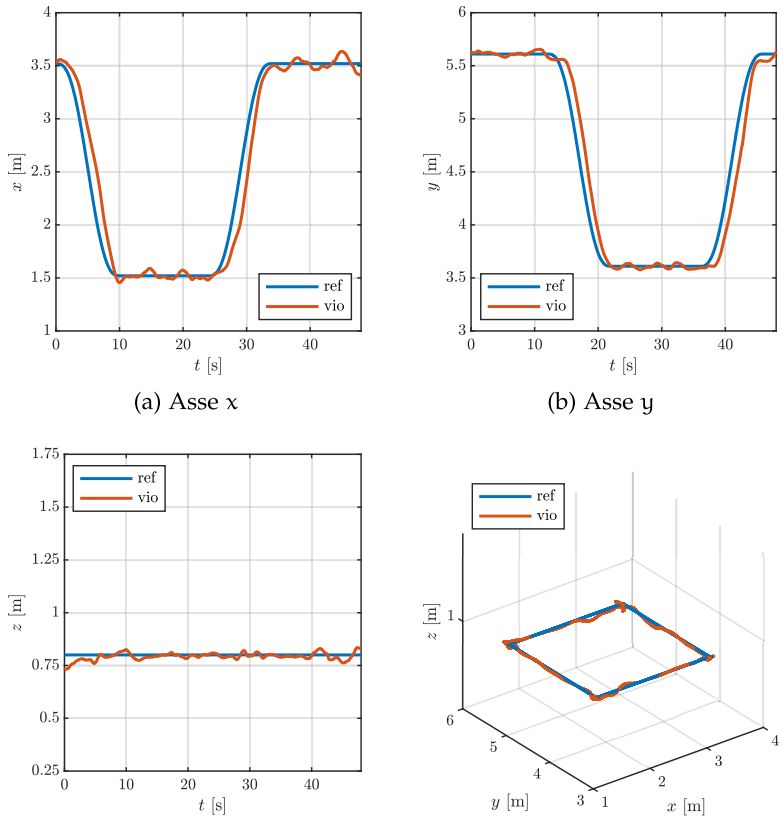
Figura 44: Stima di posizione con JBO+FIR, To e HR01.

d'inerzia dei velivoli. Come riportato in Tabella 4, masse e dimensioni del QR01 sono notevolmente inferiori a quelle dell'HR01. Di conseguenza, la dinamica del QR01 è più rapida, dunque la risposta a una stima errata in un dato istante di tempo risulta in uno scostamento maggiore dal riferimento. In [17] l'errore riscontrato eseguendo la traiettoria quadrata con QR01 è pari a circa 10 cm, coerentemente a quanto rilevato in questo test.

4.6.2 *Rimozione outlier tramite distanza euclidea (EUC) e FIR*

Per quanto riguarda gli algoritmi per la rimozione di *outliers*, anche durante i voli di HR01 si è sempre notata una maggior accuratezza della stima di posizione con il filtro a media mobile attivato. Per tali algoritmi si riportano dunque solo i grafici della stima di posizione con il filtro a media mobile attivato, al fine di alleggerire la trattazione.

In Figura 45 si riporta l'andamento della stima di posizione, con l'algoritmo *euclidean distance* in uso. Anche per l'HR01 si nota un miglioramento della stima di posizione filtrando gli *outliers* mediante l'algoritmo EUC, rispetto a quello precedentemente trattato. Il miglio-



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

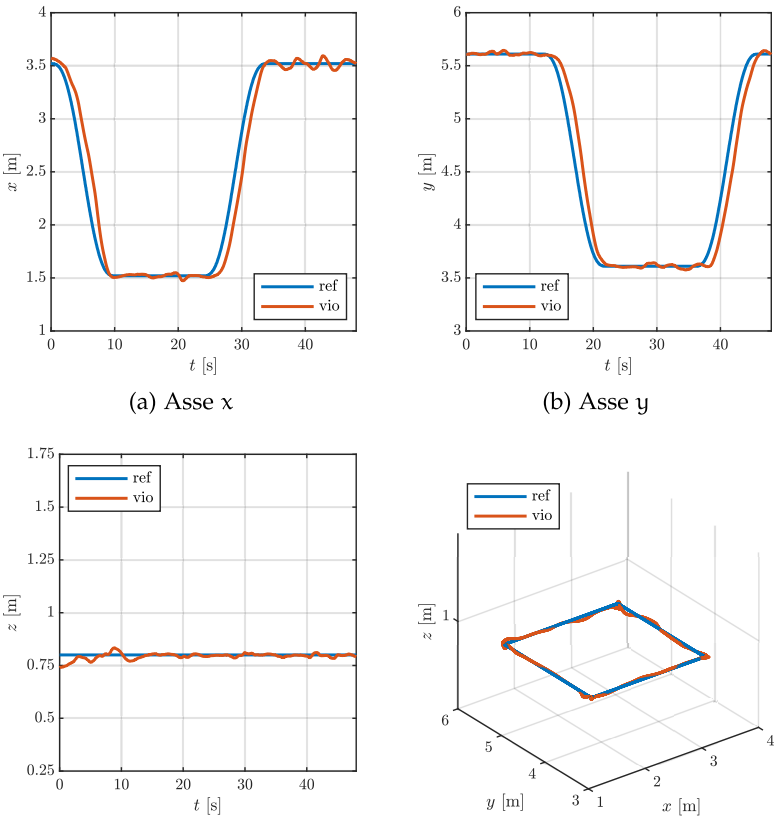
Figura 45: Stima di posizione con EUC+FIR, To e HR01.

ramento è tuttavia esiguo, pari al 12%, poiché si riscontra un errore massimo pari a 8 cm, 11 cm e 3 cm rispettivamente per la componente x, y e z. Da un analisi qualitativa del grafico tridimensionale, si evince come la traiettoria sia complessivamente più accurata, sempre comparata a quella ottenuta con l'algoritmo trattato nella precedente sottosezione.

4.6.3 *Rimozione outliers tramite IQR e FIR*

La rimozione degli *outliers* tramite scarto interquartile si conferma anche in questo caso l'algoritmo per ottenere una stima di posizione più accurata. Si riportano gli andamenti della stima di posizione in Figura 46.

L'inseguimento del riferimento è molto accurato per tutte e tre le componenti. L'errore massimo della stima è di 7 cm, 5 cm e 4 cm rispettivamente per l'asse x, y e z. Si riscontra dunque una diminuzione dell'errore massimo del 33% rispetto all'algoritmo EUC, analizzato nella precedente sottosezione. Osservando qualitativamente l'andamento della traiettoria quadrata, la stima di posizione risulta leggermente più accurata comparata ai risultati precedentemente



(a) Asse x

(b) Asse y

(c) Asse z

(d) Grafico tridimensionale

Figura 46: Stima di posizione con IQR+FIR, To e HR01.

ottenuti.

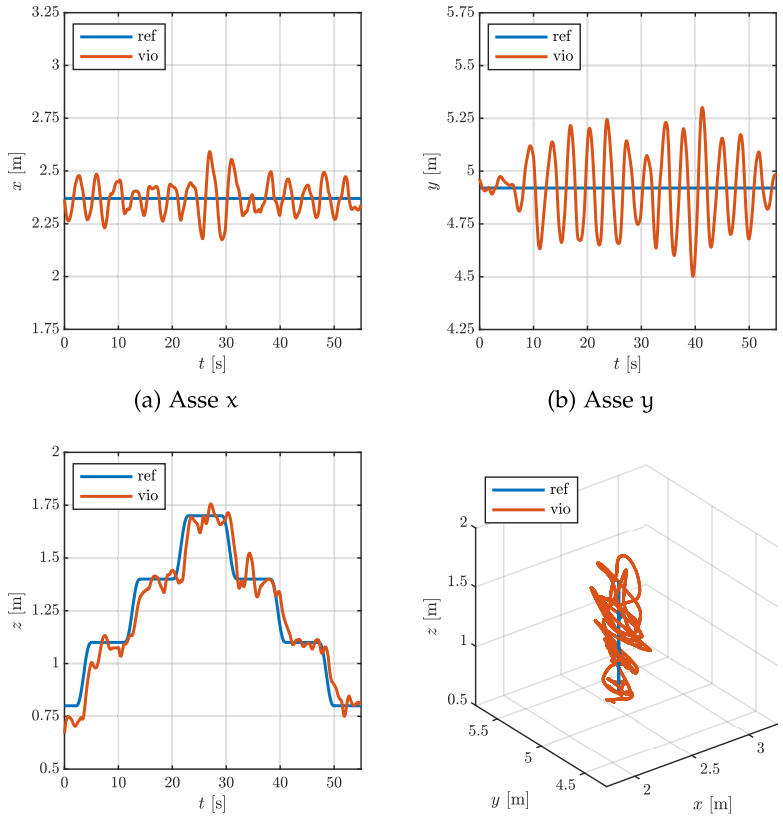
4.7 HR01 - TEST TRAIETTORIA STEP

In una seconda serie di test, si è imposto al multirotore HR01 di eseguire la traiettoria step. Si presentano nelle sottosezioni successive i risultati ottenuti attivando differenti algoritmi.

4.7.1 *AprilTag di dimensione maggiore*

Attivando questo algoritmo si considera solamente uno tra gli AprilTag di dimensione maggiore. In Figura 47 si riporta l'andamento della stima di posizione con attivato tale algoritmo.

Analizzando gli andamenti, la stima di posizione risulta molto imprecisa per tutta la durata del volo. In particolare, per l'asse x e per l'asse y il sistema appare semplicemente stabile, poiché a fronte di un riferimento costante la posizione non converge. L'errore massimo della stima per le componenti x, y e z equivale rispettivamente a 21 cm, 42 cm e 11 cm. Come si è riscontrato durante i test di volo con il multirotore QR01, l'errore della componente y è maggiore di quello della



(a) Asse x (b) Asse y

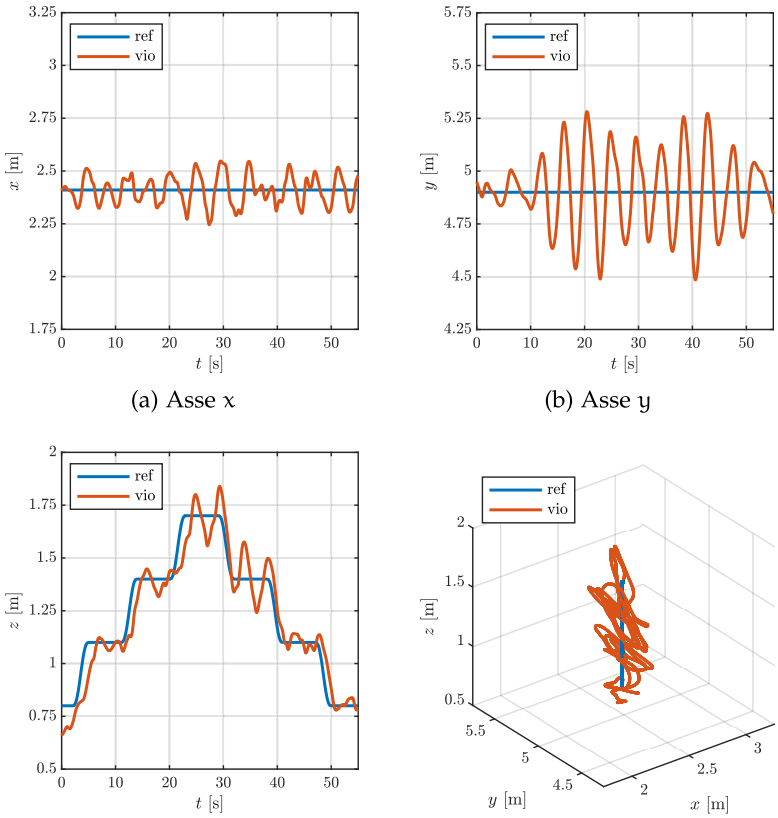
(c) Asse z (d) Grafico tridimensionale

Figura 47: Stima di posizione con JBO, T1 e HR01.

componente x, in particolare confrontando l'entità delle oscillazioni. Anche per quanto riguarda l'asse z le oscillazioni sono importanti, sia nelle fasi di salita che in quelle di discesa.

Tale risultato è inaspettato, poiché con il multirotore HR01 la stima di posizione risultava essere accurata anche con l'algoritmo che considera solamente uno tra gli AprilTag di dimensione maggiore, come esposto in Sezione 4.6 eseguendo la traiettoria quadrata. Imponendo invece un riferimento di posizione tale da eseguire la traiettoria step in esame, l'errore è importante in tutti gli assi. Si evince dunque come l'algoritmo in esame sia particolarmente inadatto al volo stazionario, preponderante nel test mediante traiettoria step, mentre con una traiettoria per la quale il multirotore è in movimento, come quella quadrata, l'accuratezza risulta accettabile. Attivando il filtro a media mobile l'andamento della stima è similare, come riportato in Figura 48. In particolare, si riscontra un errore massimo della stima per le componenti x, y e z pari rispettivamente a 17 cm, 42 cm e 15 cm. Migliora dunque l'errore massimo nella componente x del 19%, tuttavia peggiora del 26% per la componente z.

Considerando gli errori massimi della stima in tutte le componenti, l'andamento complessivo della traiettoria step è dunque inaccettabile,



(a) Asse x (b) Asse y

(c) Asse z (d) Grafico tridimensionale

Figura 48: Stima di posizione con JBO+FIR, T1 e HR01.

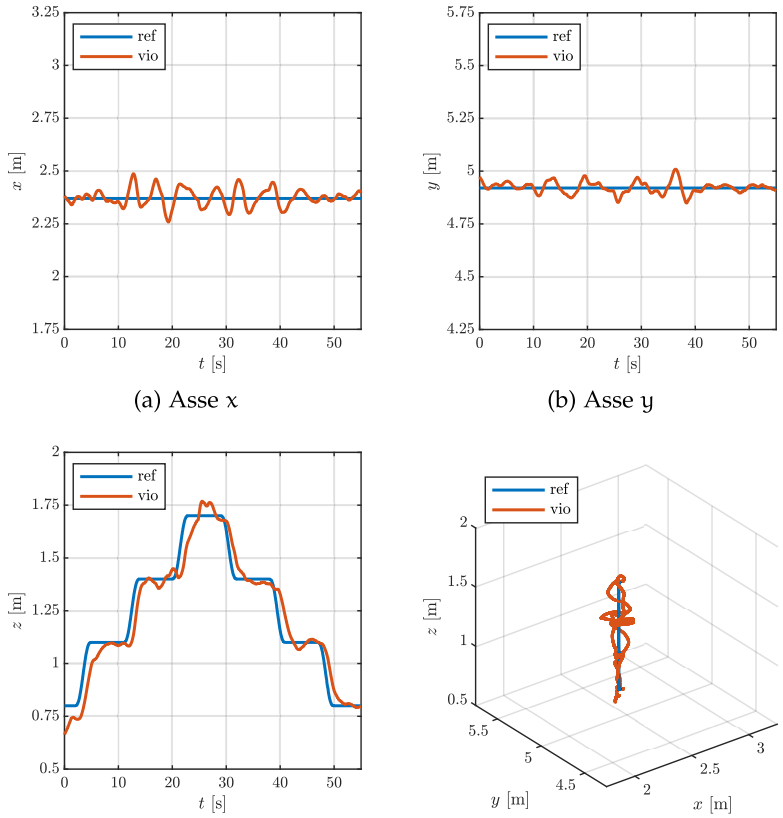
poiché presenta oscillazioni picco-picco di 84 cm, sia nel caso di filtro a media mobile attivato che disattivato, come si evince osservando i grafici tridimensionali in Figura 47 e 48.

4.7.2 *Rimozione outliers tramite distanza euclidea (EUC) e FIR*

Anche per quanto riguarda la traiettoria step, attivando gli algoritmi per la rimozione degli *outliers* si è sempre notata una maggior accuratezza della stima di posizione con il filtro a media mobile attivato, si riportano dunque solo i grafici della stima di posizione relativi a questa condizione.

Attivando l'algoritmo per la rimozione degli *outliers* tramite distanza euclidea e il filtro a media mobile, si riscontra un miglioramento della stima di posizione in tutte e tre le componenti, come riportato in Figura 49. In particolare, l'errore massimo della stima per le componenti x, y e z è pari rispettivamente a 12 cm, 9 cm e 6 cm, minore del 55% se comparato a quanto riscontrato con l'algoritmo JBO.

Sia gli andamenti che gli errori massimi sono comparabili a quelli ottenuti con lo stesso algoritmo e la stessa traiettoria dal quadrirotore QR01 (Sezione 4.5).



(a) Asse x (b) Asse y

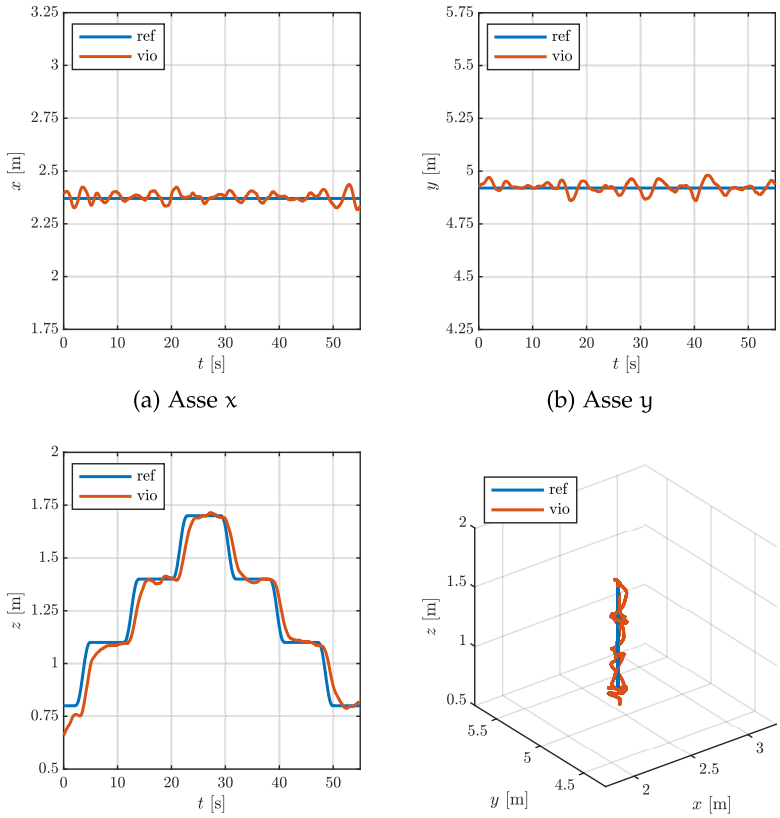
(c) Asse z (d) Grafico tridimensionale

Figura 49: Stima di posizione con EUC+FIR, T1 e HR01.

4.7.3 *Rimozione outliers tramite IQR e FIR*

Attivando l'algoritmo per la rimozione degli *outliers* tramite scarto interquartile si assiste a un ulteriore miglioramento dell'accuratezza della stima di posizione. Tale risultato è coerente con quanto analizzato nelle sezioni precedenti: rimuovendo gli *outliers* mediante IQR e FIR, si è assistito a un miglioramento dell'accuratezza per entrambe le traiettorie eseguite da entrambi i veicoli, come analizzato nelle Sezioni 4.4, 4.5 e 4.6.

In Figura 50 si riporta l'andamento delle tre componenti della stima di posizione. L'errore massimo della stima per le componenti x, y e z è pari rispettivamente a 6 cm, 6 cm e 3 cm. Comparando gli errori massimi a quelli ricavati nella precedente sezione, si nota una diminuzione del 44%. Anche da un'analisi qualitativa del grafico tridimensionale, si evince come la traiettoria imposta sia seguita accuratamente.



(a) Asse x

(b) Asse y

(c) Asse z

(d) Grafico tridimensionale

Figura 50: Stima di posizione con IQR+FIR, T1 e HR01.

4.8 ANALISI STATISTICA DEI RISULTATI

Nelle precedenti sezioni si sono analizzati qualitativamente gli andamenti dei grafici riportati. Per validare i risultati ottenuti, si è eseguita un'analisi statistica dell'errore tra stima di posizione e riferimento, per verificare con quale algoritmo si riscontrassero le prestazioni migliori.

L'analisi è esposta nelle Sottosezioni 4.8.1 e 4.8.2. Nella prima si analizzano i dati relativi all'esecuzione della traiettoria quadrata, confrontando in primo luogo i risultati ottenuti con il quadrirotore QR01 e in secondo luogo quelli ottenuti con il multirotore HR01. Nella seconda sottosezione si confrontano invece i risultati ottenuti seguendo la traiettoria step.

Si sono innanzitutto introdotti gli errori e_x, e_y, e_z, definiti come la differenza tra la componente della stima di posizione a valle dell'EKF e il rispettivo riferimento. Tramite *Matlab* si è calcolata media e varianza degli errori delle componenti cartesiane così definite per i tratti in cui il riferimento è costante.

Si è inoltre voluto calcolare un indice di accuratezza della stima che riassumesse la qualità dell'algoritmo per tutti gli assi. Per raggiunge-

re quest'obiettivo Si è dunque introdotto l'errore complessivo e, ottenuto aggregando tutti gli errori per ogni componente della stima:

$$e = (e_x \mid e_y \mid e_z). \tag{22}$$

Di quest'ultimo errore si è particolarmente interessati alla varianza, utilizzata come indice di accuratezza dell'algoritmo. In particolare, i risultati esposti nelle sottosezioni successive sono posti in ordine crescente rispetto alla varianza dell'errore complessivo.

4.8.1 *Traiettoria quadrata*

Per la traiettoria in esame si sono eseguiti 12 test di volo. Le zone analizzate per l'analisi statistica, evidenziate in ciano in Figura 51, sono di seguito riportate, specificando il numero di campioni considerato per ogni test:

- per la componente x il riferimento è costante in due tratti della durata di 12 s, denominati

  $$X1 = \{x : t \in [11\ 23]\}, \quad 240 \text{ campioni considerati}$$
  $$X2 = \{x : t \in [35\ 47]\}, \quad 240 \text{ campioni considerati;}$$

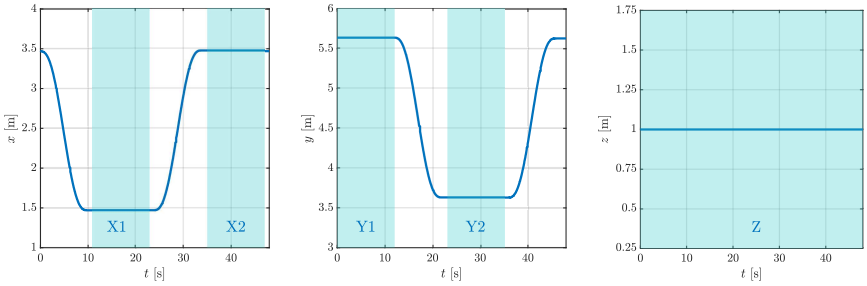
- per la componente y il riferimento è costante in due tratti, anch'essi di durata 12 s, denominati

  $$Y1 = \{y : t \in [0\ 12]\}, \quad 240 \text{ campioni considerati}$$
  $$Y2 = \{y : t \in [23\ 35]\}, \quad 240 \text{ campioni considerati;}$$

- per la componente z il riferimento è sempre costante per l'intera durata del volo, dunque si è considerato l'intero intervallo di tempo $t = [0\ 48]$, denominando il tratto Z, analizzando 960 campioni.

QR01 I risultati ottenuti eseguendo le prove di volo con il quadrirotore QR01 sono riportati in Tabella 5. Le colonne centrali riportano media e varianza dell'errore nelle tre componenti e_x, e_y, e_z nelle zone di analisi X1, X2, Y1, Y2, Z, riportate in apice dell'errore stesso.

La combinazione di algoritmi più performante in termini di accuratezza risulta essere l'insieme tra l'eliminazione degli *outliers* mediante scarto interquartile e il filtro a media mobile, che minimizza il valore della varianza dell'errore complessivo, pari a 1.7 cm. I risultati peggiori sono invece ottenuti dall'algoritmo JBO+FIR, per il quale il valore della varianza dell'errore complessivo è pari a 11.2 cm, mentre, considerando le singole componenti, l'errore e_x risulta particolarmente elevato, presentando una varianza di 18.7 cm nella zona X1.



(a) Quadrato asse x (b) Quadrato asse y (c) Quadrato asse z

Figura 51: Riferimenti di posizione per l'esecuzione della traiettoria quadrata, con evidenziati in ciano i tratti considerati per le analisi statistiche.

Algoritmo	e_x^{X1}	e_x^{X2}	e_y^{Y1}	e_y^{Y2}	e_z^{Z}	σ_e
IQR+FIR	1.0 ± 2.1	-2.0 ± 1.8	-0.4 ± 2.3	0.4 ± 2.1	0.3 ± 1.1	1.7
IQR	-1.0 ± 2.7	-2.8 ± 2.5	-0.1 ± 2.6	-4.7 ± 2.2	0.2 ± 1.4	2.0
EUC+FIR	-0.7 ± 2.3	-1.9 ± 3.2	0.6 ± 3.6	1.4 ± 4.9	0.3 ± 1.7	2.8
EUC	1.6 ± 6.4	1.0 ± 6.7	0.0 ± 11.4	-0.8 ± 2.8	0.0 ± 2.2	5.5
JBO	0.8 ± 6.9	1.5 ± 9.9	1.1 ± 6.3	-1.2 ± 8.5	0.4 ± 6.0	7.1
JBO+FIR	1.1 ± 18.7	-3.5 ± 12.2	-3.0 ± 10.3	-8.9 ± 14.2	0.2 ± 7.4	11.2

Tabella 5: Media e varianza di e in cm con To e QR01.

L'algoritmo EUC senza l'attivazione del filtro FIR presenta una varianza complessiva di 5.5 cm, particolarmente elevata a causa delle componenti x e y. Sempre utilizzando quest'ultimo algoritmo, si può notare un importante incremento dell'accuratezza attivando il filtro FIR, che comporta una diminuzione della varianza dell'errore complessivo del 49%. Da questo comportamento si può comprendere come con l'algoritmo EUC sia presente del rumore in alta frequenza, che viene filtrato dal FIR.

L'incremento di accuratezza riscontrato tra l'algoritmo IQR e l'algoritmo EUC è pari al 63% con il filtro a media mobile disattivato e al 38% con il filtro a media mobile attivato, basandosi sulla varianza dell'errore complessivo. Tale miglioramento di accuratezza è riscontrabile anche considerando le singole componenti e le singole zone di analisi.

In ultimo, si evidenzia la diminuzione della varianza dell'errore complessivo riscontrata attivando l'algoritmo IQR+FIR, rispetto a quella calcolata attivando l'algoritmo JBO, pari al 76%. Quest'ultimo aumento di accuratezza si riscontra dunque paragonando la prima versione di *apriltag to visual odometry* [3] alla versione implementata in questo elaborato.

Algoritmo	e_x^{X1}	e_x^{X2}	e_y^{Y1}	e_y^{Y2}	e_z^Z	σ_e
IQR+FIR	-0.2 ± 1.7	-0.4 ± 3.7	0.5 ± 1.5	-0.2 ± 1.9	0.5 ± 1.9	2.2
IQR	0.6 ± 4.4	0.2 ± 4.5	0.9 ± 3.0	0.2 ± 3.1	0.4 ± 2.0	3.0
EUC+FIR	0.2 ± 3.3	0.4 ± 6.3	1.0 ± 2.4	-0.4 ± 2.1	0.7 ± 2.3	3.2
EUC	0.4 ± 4.8	1.0 ± 5.9	0.9 ± 2.6	-0.6 ± 2.6	1.3 ± 2.2	3.4
JBO	-0.1 ± 3.0	-0.8 ± 5.6	0.2 ± 6.2	0.5 ± 2.3	0.5 ± 2.7	3.7
JBO+FIR	1.5 ± 3.2	-0.5 ± 5.3	0.6 ± 6.1	1.0 ± 3.4	0.4 ± 2.9	3.9

Tabella 6: Media e varianza di e in cm con To e HR01.

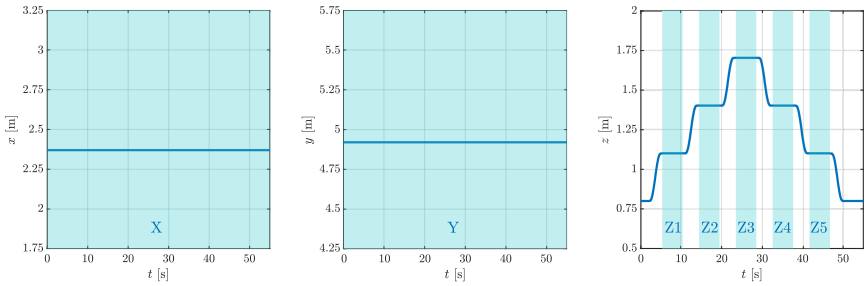
HR01 I risultati ottenuti eseguendo i test di volo con l'esarotore HR01 sono riportati in Tabella 6. Dall'analisi statistica, si evince come l'algoritmo più performante in termini di accuratezza risulta essere l'insieme tra l'eliminazione degli *outliers* mediante scarto interquartile e filtro a media mobile.

Analizzando i grafici ottenuti imponendo la traiettoria quadrata al controllore di HR01 in Sezione 4.6, si è notato come le differenze tra la stima di posizione attivando i diversi algoritmi siano minori a quanto rilevato per il quadrirotore QR01. L'analisi statistica conferma quanto rilevato qualitativamente, poiché l'aumento di accuratezza tra l'algoritmo meno performante (JBO+FIR) e quello più performante (IQR+FIR) è pari al 43%, considerando la varianza dell'errore complessivo. L'aumento è comunque importante, tuttavia non quanto l'aumento riscontrato per QR01, pari all'85%.

È inoltre d'interesse valutare la varianza dell'errore delle singole componenti. Si noti innanzitutto come la varianza non ecceda i 6.2 cm, riscontrati nella zona Y1 attivando l'algoritmo JBO. Si riscontra inoltre un aumento della varianza nella zona X2, a prescindere dall'algoritmo attivato. Questo comportamento può essere motivato considerando che numerosi fattori esterni possono peggiorare la stima di posizione localmente, come la presenza di meno illuminazione o di riflessioni in un frammento di mappa. È tuttavia importante notare come l'algoritmo performante risulti essere comunque quello IQR+FIR, sebbene la varianza sia di 3.7 cm, molto maggiore se comparata a quella nella zona X1, pari a 1.7 cm.

Per quanto riguarda la componente z, sempre costante durante la traiettoria step, si riscontra un aumento costante dell'accuratezza di stima, partendo dalla varianza di 2.9 cm ottenuta dal test con algoritmo JBO+FIR e giungendo alla varianza calcolata nel test con l'algoritmo IQR+FIR, pari a 1.9 cm.

I risultati sono dunque molto coerenti, in particolare confrontando la Tabella 5 con la Tabella 6 si evince come l'ordinamento degli algoritmi sia il medesimo, sebbene i multirotori QR01 e HR01 abbiano caratteristiche strutturali differenti, a testimoniare la robustezza della soluzione proposta.



(a) Step asse x (b) Step asse y (c) Step asse z

Figura 52: Riferimenti di posizione per l'esecuzione della traiettoria step, con evidenziati in ciano i tratti considerati per le analisi statistiche

4.8.2 *Traiettoria step*

Anche per i risultati ottenuti imponendo la traiettoria step si è eseguita un analisi statistica, in primo luogo per i voli eseguiti con il quadrirotore QR01 e in secondo luogo per quelli eseguiti con l'esarotore HR01. Per la traiettoria in esame si sono eseguiti 12 test di volo, considerando per l'analisi le zone in cui il riferimento è costante, evidenziate in ciano in Figura 52:

- per l'asse x il riferimento è sempre costante per l'intera durata del volo, dunque si è considerato l'intero intervallo di tempo $t = [0\ 55]$, denominando il tratto X e acquisendo 1100 campioni per ogni test;
- per l'asse y il riferimento è sempre costante per l'intera durata del volo, dunque si è considerato l'intero intervallo di tempo $t = [0\ 55]$, denominando il tratto Y e acquisendo 1100 campioni per ogni test;
- per l'asse z il riferimento è costante in cinque tratti della durata di cinque secondi, denominati

$$\begin{aligned} Z1 &= \{z : t \in [5\ 10]\}, \quad 100 \text{ campioni considerati} \\ Z2 &= \{z : t \in [14\ 19]\}, \quad 100 \text{ campioni considerati} \\ Z3 &= \{z : t \in [23\ 28]\}, \quad 100 \text{ campioni considerati} \\ Z4 &= \{z : t \in [32\ 37]\}, \quad 100 \text{ campioni considerati} \\ Z5 &= \{z : t \in [41\ 46]\}, \quad 100 \text{ campioni considerati.} \end{aligned}$$

QR01 I risultati ricavati dalle prove di volo eseguite con QR01 sono riportati in Tabella 7. È di particolare interesse la colonna con la varianza dell'errore complessivo, evidenziata in ciano. Anche per la traiettoria step, la diminuzione della varianza dell'errore complessivo utilizzando algoritmi più avanzati è notevole, in particolare si nota

Algoritmo	e_x^X	e_y^Y	e_z^{Z1}	e_z^{Z2}	e_z^{Z3}	e_z^{Z4}	e_z^{Z5}	σ_e
IQR+FIR	-2.0 ± 2.5	-5.4 ± 4.0	1.2 ± 2.8	2.9 ± 5.4	2.1 ± 4.3	-1.8 ± 4.9	-1.6 ± 4.9	3.6
IQR	-1.8 ± 2.9	-4.8 ± 5.0	2.3 ± 2.6	1.2 ± 4.1	1.8 ± 3.3	-0.2 ± 2.1	-0.5 ± 4.3	3.9
EUC+FIR	-1.2 ± 4.6	-3.8 ± 4.7	2.0 ± 2.8	1.1 ± 3.5	1.8 ± 1.9	-1.4 ± 3.3	-1.3 ± 3.5	4.4
EUC	-1.2 ± 7.0	-3.6 ± 10.2	2.7 ± 6.3	1.2 ± 7.5	-0.9 ± 7.2	-3.0 ± 8.4	-2.1 ± 3.3	8.4
JBO	-1.5 ± 10.7	-5.2 ± 20.5	2.8 ± 8.2	3.2 ± 13.9	6.1 ± 10.2	-1.9 ± 14.4	-0.1 ± 3.6	15.5
JBO+FIR	1.4 ± 9.4	1.2 ± 22.5	2.7 ± 5.4	4.8 ± 14.7	1.7 ± 9.5	-0.9 ± 11.4	-1.1 ± 9.3	16.2

Tabella 7: Media e varianza di e in cm con T1 e QR01.

una diminuzione del 78% tra quella calcolata con l'algoritmo meno performante (JBO+FIR) e quello più performante (IQR+FIR). In particolare, attivando l'algoritmo che esclude gli *outliers* tramite scarto interquartile e il filtro a media mobile si ottiene il miglior risultato in termini di accuratezza, ottenendo un valore di varianza dell'errore complessivo pari a 3.6 cm. Comparando i risultati ottenuti imponendo la traiettoria step (Tabella 7) e quelli ottenuti imponendo quella quadrata (Tabella 5), si nota un aumento della varianza dell'errore complessivo, a prescindere dall'algoritmo considerato. Questo dimostra come la traiettoria step necessiti di una stima più accurata per l'esecuzione, probabilmente a causa dei tratti stazionari più lunghi.

Proprio per questa traiettoria si ottiene il peggior valore della varianza, considerando la totalità dei test effettuati, pari a 22.5 cm. Quest'ultimo valore è ottenuto dalla componente y con l'algoritmo JBO+FIR.

È d'interesse valutare anche la differenza tra la varianza dell'errore della componete x e quella della componente y. Sia il riferimento in x che quello in y sono costanti nell'esecuzione della traiettoria step, tuttavia si riscontrano valori molto differenti. In particolare, per la componente x la varianza dell'errore è compresa tra 2.5 cm (ottenuta con algoritmo IQR+FIR) e 10.7 cm (calcolata con algoritmo JBO), mentre per la componente y è compresa tra 4 cm (ottenuta con algoritmo IQR+FIR) e 22.5 cm (calcolata con algoritmo JBO+FIR). Il valore elevato della varianza per la componente y può essere motivato dal differente momento d'inerzia in x e in y del quadrirotore QR01, come visto in sezione 4.5.

HR01 Per quanto riguarda invece i voli eseguiti con HR01, sempre imponendo l'inseguimento della traiettoria step, i risultati ottenuti sono riportati in Tabella 8. L'analisi statistica è coerente a quanto ricavato dall'osservazione dei grafici presenti in Sezione 4.7. L'algoritmo più performante si conferma quello che esclude gli *outliers* tramite scarto interquartile unito al filtro a media mobile. Comparando quest'ultimo algoritmo con quello JBO, implementato nella prima versione di *apriltag to visual odometry*, si riscontra una diminuzione della varianza dell'errore complessivo del 75%. In particolare, la varianza dell'errore complessivo con l'algoritmo IQR+FIR è pari a 3.1 cm.

Algoritmo	e_x^X	e_y^Y	e_z^{Z1}	e_z^{Z2}	e_z^{Z3}	e_z^{Z4}	e_z^{Z5}	σ_e
IQR+FIR	0.9 ± 2.5	0.4 ± 2.8	5.3 ± 4.5	3.6 ± 4.7	2.7 ± 5.1	-2.0 ± 4.9	-3.1 ± 3.7	3.1
IQR	1.5 ± 4.4	0.0 ± 4.2	6.1 ± 6.0	3.7 ± 4.6	3.5 ± 4.7	-2.6 ± 6.3	-2.6 ± 4.0	4.5
EUC+FIR	0.5 ± 4.5	0.3 ± 3.3	4.9 ± 5.3	3.5 ± 3.3	-0.2 ± 10.3	-1.3 ± 5.1	-2.1 ± 7.0	4.6
EUC	0.7 ± 4.9	0.4 ± 3.7	5.8 ± 5.4	3.2 ± 6.4	1.4 ± 9.3	0.1 ± 5.7	-2.5 ± 5.6	4.9
JBO	-0.1 ± 8.8	-0.2 ± 16.3	5.5 ± 7.3	5.3 ± 5.6	2.6 ± 6.0	-1.4 ± 9.3	-2.5 ± 5.6	12.2
JBO+FIR	-0.1 ± 7.8	0.8 ± 19.2	4.4 ± 7.3	3.2 ± 6.8	4.7 ± 12.1	-1.0 ± 16.1	-1.3 ± 4.8	14.0

Tabella 8: Media e varianza di e in cm con T1 e HR01.

È di interesse comparare la varianza dell'errore complessivo calcolata eseguendo la traiettoria quadrata (Tabella 6) e quella con traiettoria step, presentata in questo paragrafo. Si nota innanzitutto la differenza tra l'escursione della varianza eseguendo la traiettoria quadrata, pari a $1.7\,\mathrm{cm}$, e quella eseguendo la traiettoria step, di $10.9\,\mathrm{cm}$. L'aumento dell'escursione conferma come l'esecuzione della traiettoria step necessiti di una stima di posizione più accurata. Questo fenomeno era già stato evidenziato in Sezione 4.7, analizzando qualitativamente gli andamenti della stima di posizione.

Comparando inoltre i risultati ottenuti dall'esecuzione della traiettoria step con QR01 (Tabella 7) e quelli ottenuti con HR01 (Tabella 8), si osserva come le varianze siano simili, sia considerando l'errore complessivo che quello nelle singole componenti. Questo risultato è inaspettato: HR01 ha una massa maggiore dell'86% rispetto QR01, di conseguenza anche le inerzie sono maggiori. Ci si aspetta dunque una dinamica più lenta è un minor errore della stima di posizione, che si è effettivamente riscontrato per la traiettoria quadrata. Si può quindi giustificare la similarità delle varianze considerando come la traiettoria step necessiti di un'accurata stima di posizione per essere eseguita precisamente, a prescindere dal velivolo.

In conclusione, i risultati ottenuti sia nell'analisi statistica quantitativa che nell'analisi qualitativa degli andamenti dei grafici sono coerenti per entrambe le traiettorie analizzate. In particolare, si è notato un miglioramento netto della stima di posizione escludendo gli *outliers* tramite scarto interquartile e implementando un filtro a media mobile. È inoltre importante notare come i medesimi risultati siano stati ottenuti eseguendo voli di prova con due multirotori di massa e dimensioni differenti e seguendo traiettorie differenti.

CONCLUSIONI

Il lavoro di tesi si è focalizzato sul miglioramento di un sistema di localizzazione per UAVs basato su *visual inertial odometry* e *fiducial markers*. Si sono implementati differenti algoritmi per migliorare la stima della posa del velivolo, suddivisibili in quattro categorie fondamentali:

- minimizzazione della latenza - al fine di garantire un'implementazione più snella, veloce ed efficiente;
- miglior sfruttamento delle informazioni - al fine di considerare tutta l'informazione a disposizione;
- rimozione degli *outliers* - al fine di rimuovere le informazioni non utili per il calcolo della stima;
- attenuazione del rumore in alta frequenza - al fine di rendere la stima meno disturbata.

Gli algoritmi presentati sono stati validati sperimentalmente per mezzo di due UAVs, il quadrirotore QR01 e l'esarotore HR01. Dall'analisi sperimentale si è compreso quale combinazione di algoritmi fornisca una stima di posizione più accurata: l'insieme tra l'eliminazione di *outliers* tramite scarto interquartile e il filtro a media mobile (IQR+FIR). Si sottolinea come i dati acquisiti per via sperimentale siano risultati coerenti: sebbene i voli siano stati eseguiti con multirotori e traiettorie differenti, gli algoritmi citati si sono rivelati i più accurati per tutti i test effettuati. Si è voluto quantificare l'aumento di accuratezza ottenuto tramite il sistema implementato in questo elaborato (seconda versione) rispetto a quello rilevato per mezzo del sistema implementato in un precedente lavoro di tesi (prima versione) [3]. Si sono dunque considerati sia l'errore massimo che la variazione della varianza relativa all'errore di posizione nell'esecuzione di due traiettorie. In particolare, con la prima versione del sistema si ottiene un errore massimo di circa 40 cm e una varianza media di 10 cm, aggregando i dati acquisiti con entrambi i multirotori. Utilizzando invece la seconda versione del sistema implementato, si riscontra un errore massimo di circa 10 cm e una varianza media di 3 cm, sempre aggregando i dati acquisiti con entrambi i multirotori. I miglioramenti riscontrati sull'errore massimo e sulla varianza media sono quindi rispettivamente pari al 75% e al 70%.

Il lavoro di tesi si colloca in un progetto di ricerca più ampio, orientato allo studio dell'interazione tra l'ambiente, robot aerei e terrestri e l'uomo in ambito industriale. In caso l'incremento dell'accuratezza della stima di posizione per robot aerei riscontrato non sia

ritenuto sufficiente, sono proposti ulteriori sviluppi per aumentare ulteriormente l'accuratezza:

- *companion-computer* più performante - il Raspberry PI 4B utilizzato ha capacità di calcolo limitate, che consentono l'esecuzione del nodo atto a riconoscere i *fiducial markers* a una frequenza massima di 20 Hz con risoluzione accettabile (720×480). L'utilizzo di un *companion-computer* più performante può consentire un aumento sia della frequenza che della risoluzione;
- *global shutter* - l'otturatore di ambedue le camere utilizzate per la validazione sperimentale è *rolling shutter*, che in condizioni dinamiche può causare una deformazione dell'immagine e l'errato o mancato riconoscimento di *fiducial markers*. L'utilizzo di una camera con otturatore *global shutter* può consentire prestazioni dinamiche migliori;
- implementazione di nuovi algoritmi - ulteriori algoritmi possono essere implementati e validati, sulla base dei risultati ottenuti e delle considerazioni esposte.

APPENDIX

A

(QUATERNIONI E ROTAZIONI)

A.1 RAPPRESENTAZIONE DI ROTAZIONI MEDIANTE I QUATERNIONI

Ogni rotazione nel piano può essere rappresentata mediante i numeri complessi. Infatti, identificando il punto P di coordinate $(x\ y)^T$ con $z = x + iy \in \mathbb{C}$, la rotazione di misura φ intorno all'origine coincide con l'applicazione [49]:

$$\mathbb{C} \to \mathbb{C} : z \mapsto ze^{i\varphi}. \tag{23}$$

I quaternioni consentono di conseguire una rappresentazione analoga delle rotazioni nello spazio. Si rappresenti l'insieme dei quaternioni con $\mathbb{H} = \mathbb{R}^4$. Tipicamente si identificano i quaternioni con la lettera $\mathbf{q}$, e spesso giova rappresentarli come una coppia formata da uno scalare e da un vettore:

$$\mathbf{q} = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{v}) \tag{24}$$

dove $\mathbf{v} = (q_1, q_2, q_3) \in \mathbb{R}^3$ è definita *parte vettoriale*, mentre q_0 è definita *parte scalare*.

Si definisce inoltre *norma* di un quaternione $\mathbf{q}$:

$$\|\mathbf{q}\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}, \tag{25}$$

mentre un quaternione avente norma unitaria si definisce *quaternione unitario*. Per ogni quaternione unitario $\mathbf{q}$, esistono un unico $\varphi \in [0\ 2\pi]$ e almeno un versore $\mathbf{u}$, tali che [49]:

$$\mathbf{q} = \left(\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{u}\right). \tag{26}$$

Il quaternione in Equazione 26 può essere associato a una rotazione di misura φ intorno all'asse orientato individuato da $\mathbf{u}$ passante per l'origine. In particolare, un quaternione è associato a un'univoca rotazione, tuttavia una rotazione può essere associata a due differenti quaternioni, che sono uno l'opposto dell'altro. Tale proprietà viene definita come *corrispondenza due a uno tra quaternioni unitari e rotazioni*. Un'ulteriore importante proprietà dell'associazione tra quaternioni e rotazioni riguarda la composizione di rotazioni: per comporre più rotazioni è sufficiente moltiplicare i quaternioni unitari associati.

A.1.1 *Matrici di rotazione e quaternioni*

Una rotazione nello spazio può essere associata anche a una matrice ortogonale speciale, che si definisce *matrice di rotazione*. La rotazione

di misura φ intorno all'asse orientato individuato da $\mathbf{u} = (h, k, l)$ passante per l'origine può essere rappresentata sia dal quaternione in Equazione 26 che dalla matrice:

$$\mathbf{R} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \in SO(3) \tag{27}$$

dove

$$\begin{aligned} a_1 &= (1 - h^2)\cos\varphi + h^2 \\ a_2 &= hk(1 - \cos\varphi) - l\sin\varphi \\ a_3 &= lh(1 - \cos\varphi) + k\sin\varphi \\ b_1 &= hk(1 - \cos\varphi) + l\sin\varphi \\ b_2 &= (1 - k^2)\cos\varphi + k^2 \\ b_3 &= kl(1 - \cos\varphi) - h\sin\varphi \\ c_1 &= lh(1 - \cos\varphi) - k\sin\varphi \\ c_2 &= kl(1 - \cos\varphi) + h\sin\varphi \\ c_3 &= (1 - l^2)\cos\varphi + l^2. \end{aligned}$$

Nelle implementazioni *software* vengono spesso utilizzati i quaternioni unitari piuttosto che le matrici di rotazione, poiché: [50]:

- i quaternioni sono composti da 4 scalari, al contrario delle matrici di rotazione che sono composte da 9 scalari. Si riduce dunque la memoria necessaria al salvataggio dell'informazione di rotazione e si velocizzano i calcoli riguardanti le rotazioni. In particolare, comporre due rotazioni rappresentate da quaternioni richiede 16 moltiplicazioni scalari, mentre la composizione di due rotazioni associate a matrici necessita di 27 moltiplicazioni scalari;
- utilizzando i quaternioni si possono evitare distorsioni introdotte dall'accumularsi di errori dovuti alla rappresentazione *floating-point* di numeri reali. Al contrario delle matrici di rotazione, un quaternione è facilmente normalizzabile, così da evitare distorsioni dovute a errori numerici;

Software come PX4 Autopilot o librerie come tf2 (inclusa in ROS 2) utilizzano i quaternioni per rappresentare le rotazioni.

A.2 MEDIA PESATA TRA ROTAZIONI ASSOCIATE A QUATERNIONI

Ottenere la media pesata tra più quaternioni è un problema noto in letteratura [44, 45, 46, 47]. La problematica è stata affrontata anche dalla NASA (*National Aeronautics and Space Administration*) per la

stima dell'orientamento di veicoli spaziali in [44], proponendo una soluzione che si presta a essere implementata come algoritmo.

Il fulcro della trattazione si basa sull'osservare come l'obiettivo ultimo sia mediare delle orientazioni anziché i quaternioni. Di conseguenza, il quaternione mediato deve minimizzare la somma pesata della norma di Frobenius al quadrato della differenza tra le matrici di orientazione:

$$\overline{\mathbf{q}} \triangleq \arg\min_{\mathbf{q}\in\mathbb{S}^3} \sum_{i=1}^{n} w_i \left\|\mathbf{A}(\mathbf{q}) - \mathbf{A}\left(\mathbf{q}_i\right)\right\|_F^2 \tag{28}$$

dove $\mathbb{S}^3 = \{\mathbf{q} \in \mathbb{H} : \|q\| = 1\}$ denota la 3-sfera unitaria e la norma di Frobenius è definita:

$$\begin{aligned}\left\|\mathbf{A}(\mathbf{q}) - \mathbf{A}\left(\mathbf{q}_i\right)\right\|_F^2 &= \operatorname{Tr}\left\{\left[\mathbf{A}(\mathbf{q}) - \mathbf{A}\left(\mathbf{q}_i\right)\right]^T \left[\mathbf{A}(\mathbf{q}) - \mathbf{A}\left(\mathbf{q}_i\right)\right]\right\} &(29)\\ &= 6 - 2\operatorname{Tr}\left[\mathbf{A}(\mathbf{q})\mathbf{A}^T\left(\mathbf{q}_i\right)\right] &(30)\end{aligned}$$

dove nell'ultimo passaggio si sono sfruttate le proprietà della traccia di una matrice, denotata da Tr e la definizione di ortogonalità di $\mathbf{A}(\mathbf{q})$ e $\mathbf{A}(\mathbf{q}_i)$. Si può dunque esprimere l'Equazione 28 come:

$$\overline{\mathbf{q}} = \arg\max_{\mathbf{q}\in\mathbb{S}^3} \operatorname{Tr}\left[\mathbf{A}(\mathbf{q})\mathbf{B}^T\right] \tag{31}$$

dove

$$\mathbf{B} \triangleq \sum_{i=1}^{n} w_i \mathbf{A}\left(\mathbf{q}_i\right) \tag{32}$$

è definita *attitude profile matrix*, in quanto contiene tutta l'informazione sull'orientazione.

Come riportato in Equazione 24, un generico quaternione $\mathbf{q}$ può essere rappresentato con una coppia scalare-vettore. Ricordando inoltre che i quaternioni rappresentanti le rotazioni hanno norma unitaria, dunque $q_0^2 + \|\mathbf{w}\|^2 = \mathbf{q}\mathbf{q}^T = 1$, la matrice di rotazione $\mathbf{A}$ è associata al quaternione $\mathbf{q}$ mediante:

$$\mathbf{A}(\mathbf{q}) = \left(q_0^2 - \|\mathbf{v}\|^2\right)\mathbf{I}_{3\times 3} + 2\mathbf{v}^T\mathbf{v} - 2q_0[\mathbf{v}]_\times \tag{33}$$

dove $\mathbf{I}_{3\times 3} \in \mathbb{R}^{3\times 3}$ è la matrice identità e $[\mathbf{v}]\times$ è la matrice del prodotto vettore:

$$[\mathbf{v}]\times \triangleq \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix}. \tag{34}$$

Mediante l'Equazione 33 si può verificare la seguente identità:

$$\operatorname{Tr}\left[\mathbf{A}(\mathbf{q})\mathbf{B}^T\right] = \mathbf{q}^T\mathbf{K}\mathbf{q} \tag{35}$$

dove $\mathbf{K} \in \mathbb{R}^{4\times4}$ è definita come:

$$\mathbf{K} \triangleq \begin{bmatrix} \mathbf{B} + \mathbf{B}^\top - \mathrm{Tr}(\mathbf{B})\mathbf{I}_{3\times3} & \mathbf{z} \\ \mathbf{z}^\top & \mathrm{Tr}(\mathbf{B}) \end{bmatrix} \tag{36}$$

e $\mathbf{z}$ è definito dalla matrice prodotto vettore:

$$[\mathbf{z}\times] \triangleq \mathbf{B}^\top - \mathbf{B}. \tag{37}$$

Sostituendo l'Equazione 33 prima in 32 e poi in 36 si ricava:

$$\mathbf{K} \triangleq 4\mathbf{M} - w_{\mathrm{tot}}\mathbf{I}_{4\times4} \tag{38}$$

dove $w_{\mathrm{tot}} \triangleq \sum_{i=1}^{n} w_i$ è la somma dei pesi relativi alle n orientazioni da mediare, $\mathbf{I}_{4\times4} \in \mathbb{R}^4$ è la matrice identità e $\mathbf{M} \in \mathbb{R}^4$ è definita:

$$\mathbf{M} \triangleq \sum_{i=1}^{n} w_i \mathbf{q}_i \mathbf{q}_i^\top. \tag{39}$$

Dunque, il quaternione medio rappresentante la media pesata tra le orientazioni può essere calcolato risolvendo il seguente problema di massimizzazione:

$$\overline{\mathbf{q}} = \arg\max_{\mathbf{q}\in\mathbb{S}^3} \mathbf{q}^\top \mathbf{M}\mathbf{q}. \tag{40}$$

La soluzione a tale problema di massimizzazione è ben nota in letteratura [51]: il quaternione medio è l'autovettore della matrice $\mathbf{M}$ associato all'autovalore massimo. Il calcolo della media pesata di un orientazione si riduce quindi a un classico problema agli autovalori per la matrice $\mathbf{M} \in \mathbb{R}^4$.

BIBLIOGRAFIA

[1] Panagiotis Radoglou-Grammatikis, Panagiotis Sarigiannidis, Thomas Lagkas, and Ioannis Moscholios. A compilation of UAV applications for precision agriculture. *Computer Networks*, 172: 107148, 2020. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2020.107148. URL https://www.sciencedirect.com/science/article/pii/S138912862030116X.

[2] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.

[3] Martino Segattini. Progettazione di un esarotore e implementazione di tecniche di sensing e perception. Master's thesis, Università degli Studi di Padova, 2021.

[4] Huthaifa Obeidat, Wafa Shuaieb, Omar Obeidat, and Raed Abd-Alhameed. A review of indoor localization techniques and wireless technologies. *Wireless Personal Communications*, 119(1): 289–327, 2021.

[5] Ahmed Azeez Khudhair, Saba Qasim Jabbar, M Qasim Sulttan, and Desheng Wang. Wireless indoor localization systems and techniques: survey and comparative study. *Indonesian Journal of Electrical Engineering and Computer Science*, 3(2):392–409, 2016.

[6] Hyunsoo Yang, Yongseok Lee, Sang-Yun Jeon, and Dongjun Lee. Multi-rotor drone tutorial: systems, mechanics, control and state estimation. *Intelligent Service Robotics*, 10(2):79–93, 2017.

[7] GN Muchiri and S Kimathi. A review of applications and potential applications of uav. In *Proceedings of the Sustainable Research and Innovation Conference*, pages 280–283, 2022.

[8] Istituto della Enciclopedia Italiana fondata da Giovanni Treccani, editor. *Vocabolario della Lingua Italiana*.

[9] Quan Quan. *Observability and Kalman Filter*, pages 173–197. Springer Singapore, Singapore, 2017. ISBN 978-981-10-3382-7. doi: 10.1007/978-981-10-3382-7_8. URL https://doi.org/10.1007/978-981-10-3382-7_8.

[10] Lukas Wawrla, Omid Maghazei, and Torbjørn Netland. Applications of drones in warehouse operations. *Whitepaper. ETH Zurich, D-MTEC*, 2019.

[11] Chouchang Yang and Huai-Rong Shao. Wifi-based indoor positioning. *IEEE Communications Magazine*, 53(3):150–157, 2015.

[12] AKM Mahtab Hossain and Wee-Seng Soh. A comprehensive study of bluetooth signal parameters for localization. In *2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5. IEEE, 2007.

[13] Lionel M Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P Patil. Landmarc: Indoor location sensing using active rfid. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).*, pages 407–415. IEEE, 2003.

[14] A B M Mohaimenur Rahman, Ting Li, and Yu Wang. Recent advances in indoor localization via visible lights: A survey. *Sensors*, 20(5), 2020. ISSN 1424-8220. doi: 10.3390/s20051382. URL https://www.mdpi.com/1424-8220/20/5/1382.

[15] A B M Mohaimenur Rahman, Ting Li, and Yu Wang. Recent advances in indoor localization via visible lights: A survey. *Sensors*, 20(5), 2020. ISSN 1424-8220. doi: 10.3390/s20051382. URL https://www.mdpi.com/1424-8220/20/5/1382.

[16] Anca Morar, Alin Moldoveanu, Irina Mocanu, Florica Moldoveanu, Ion Emilian Radoi, Victor Asavei, Alexandru Gradinaru, and Alex Butean. A comprehensive survey of indoor localization methods based on computer vision. *Sensors*, 20(9):2641, 2020.

[17] Massimiliano Bertoni, Stefano Michieletto, Roberto Oboe, and Giulia Michieletto. Indoor visual-based localization system for multi-rotor uavs. *Sensors*, 22(15):5798, 2022.

[18] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur, and Xavier Savatier. A study of vicon system positioning performance. *Sensors*, 17(7):1591, 2017.

[19] Joshua Springer and Marcel Kyas. Evaluation of april tag and whycode fiducial systems for autonomous precision drone landing with a gimbal-mounted camera. *arXiv preprint arXiv:2203.10180*, 2022.

[20] Joshua Springer and Marcel Kyas. Autonomous precision drone landing with fiducial markers and a gimbal-mounted camera for active tracking. *arXiv preprint arXiv:2206.04617*, 2022.

[21] Mohammad Nahangi, Adam Heins, Brenda McCabe, and Angela Schoellig. Automated localization of uavs in gps-denied indoor construction environments using fiducial markers. In

ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction, volume 35, pages 1–7. IAARC Publications, 2018.

[22] Eduard Mráz, Jozef Rodina, and Andrej Babinec. Using fiducial markers to improve localization of a drone. In *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, pages 1–5, 2020. doi: 10.1109/ISMCR51255.2020.9263754.

[23] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017. doi: 10.1109/TRO.2016.2597321.

[24] Guoquan Huang. Visual-inertial navigation: A concise review. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9572–9582, 2019. doi: 10.1109/ICRA.2019.8793604.

[25] Euntae Hong and Jongwoo Lim. Visual-inertial odometry with robust initialization and online scale estimation. *Sensors*, 18(12), 2018. ISSN 1424-8220. doi: 10.3390/s18124287. URL https://www.mdpi.com/1424-8220/18/12/4287.

[26] G. Michieletto, A. Cenedese, and A. Franchi. Force-moment decoupling and rotor-failure robustness for star-shaped generically-tilted multi-rotors. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 2132–2137, 2019. doi: 10.1109/CDC40024.2019.9030008.

[27] D Nister, O Naroditsky, and J Bergen. Visual odometry. computer vision and pattern recognition, 2004. cvpr 2004. In *Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, 2004.

[28] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part i: The first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4):80–92, 2011.

[29] Hans Peter Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Stanford University, 1980.

[30] Mark Fiala. Comparing artag and artoolkit plus fiducial marker systems. In *IEEE International Workshop on Haptic Audio Visual Environments and their Applications*, pages 6–pp. IEEE, 2005.

[31] Artur Sagitov, Ksenia Shabalina, Roman Lavrenov, and Evgeni Magid. Comparing fiducial marker systems in the presence of occlusion. In *2017 International Conference on Mechanical, System and Control Engineering (ICMSC)*, pages 377–382. IEEE, 2017.

[32] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

[33] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2014.01.005. URL https://www.sciencedirect.com/science/article/pii/S0031320314000235.

[34] Michail Kalaitzakis, Sabrina Carroll, Anand Ambrosi, Camden Whitehead, and Nikolaos Vitzilaios. Experimental comparison of fiducial markers for pose estimation. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 781–789. IEEE, 2020.

[35] Aufar Zakiev, Tatyana Tsoy, Ksenia Shabalina, Evgeni Magid, and Subir Kumar Saha. Virtual experiments on aruco and apriltag systems comparison for fiducial marker rotation resistance under noisy sensory data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2020.

[36] Gaetano C La Delfa, Vincenzo Catania, Salvatore Monteleone, Juan F De Paz, and Javier Bajo. Computer vision based indoor navigation: a visual markers evaluation. In *Ambient Intelligence-Software and Applications*, pages 165–173. Springer, 2015.

[37] Ern J Lefferts, F Landis Markley, and Malcolm D Shuster. Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429, 1982.

[38] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2004.07.002. URL https://www.sciencedirect.com/science/article/pii/S156625350400065X.

[39] The story of px4 and pixhawk. https://auterion.com/company/the-history-of-pixhawk/, 2019.

[40] G. Nutt et al. http://www.nuttx.org, 2014.

[41] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[42] Px4-ros 2 bridge. https://docs.px4.io/main/en/ros/ros2_comm.html.

[43] std::tuple - cppreference.com. https://en.cppreference.com/w/cpp/utility/tuple.

[44] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[45] Jae-Hyung Cho, AD Rollett, and KH Oh. Determination of a mean orientation in electron backscatter diffraction measurements. *Metallurgical and materials transactions A*, 36(12):3427–3438, 2005.

[46] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International journal of computer vision*, 103(3): 267–305, 2013.

[47] Maher Moakher. Means and averaging in the group of rotations. *SIAM journal on matrix analysis and applications*, 24(1):1–16, 2002.

[48] Holybro qav250 + pixhawk4-mini build. https://docs.px4.io/main/en/frames_multicopter/holybro_qav250_pixhawk4_mini.html.

[49] Corrado Zanella. Appunti di geometria.

[50] Ron Goldman. Understanding quaternions. *Graphical Models*, 73(2):21–49, 2011. ISSN 1524-0703. doi: https://doi.org/10.1016/j.gmod.2010.10.004. URL https://www.sciencedirect.com/science/article/pii/S1524070310000172.

[51] James R Wertz. Spacecraft attitude determination and control. *Astrophysics and Space Science Library*, 73:426–428, 1978.