Alma Mater Studiorum - Università di Bologna

Scuola di Ingegneria e Architettura

- Sede di Forlì -

Corso di Laurea in Ingegneria Aerospaziale - Classe L-20

Tesi di Laurea in Dinamica del Volo

Modellazione e controllo avanzato di un velivolo multirotore

Presso il Laboratorio di Meccanica del Volo

Candidato:

Alberto Sodi

Relatore:

Prof. Ing. Fabrizio Giulietti

Anno Accademico 2012/2013

III Sessione

A Rudolf Emil Kálmán

Abstract

Grazie alla loro versatilità, i velivoli multirotore hanno ricevuto sempre più interesse durante gli ultimi anni, in ambito accademico e di recente anche industriale. Il lavoro presentato è volto a studiare e confrontare le moderne tecniche di navigazione e di controllo di questo tipo di velivoli. Difatti, spesso, gli algoritmi utilizzati sono stati limitati dalla capacità di calcolo del processore imbarcato, come dalla qualità dei sensori utilizzati. Negli ultimi anni, però, lo sviluppo della microelettronica ha ricevuto un forte impulso (dovuto principalmente alla ricerca nell'ambito della telefonia), che ha portato all'abbattimento dei costi e alla nascita di progetti opensource, tra i quali le famose schede Arduino prodotte da Olivetti, attorno alle quali si sono sviluppati molti progetti di velivoli opensource. L'importanza di ciò, in ambito accademico, è rilevante, poiché consente l'utilizzo di algoritmi e di configurazioni hardware comprovati, lasciando spazio a modifiche e migliorie. Nel nostro caso, in particolare, si vuole osservare come complessi algoritmi di navigazione, resi possibili da un processore più potente, possano migliorare le prestazioni del noto progetto opensource ArduPilot [3]. Tali miglioramenti possono essere rilevanti in applicazioni per le quali è richiesta una certa precisione nel posizionamento, come ad esempio lo studio di formazioni o la navigazione in ambienti angusti.

Indice

Elenco delle figure

Introduzione

Cenni storici

Recentemente la microelettronica ha ricevuto un impulso notevole, dovuto al naturale processo di evoluzione delle tecniche costruttive e in gran parte all'aumento di dispositivi di consumo che hanno invaso il mercato conquistando l'attenzione dei grandi costruttori. Questo ha dato la possibilità di applicare una grande varietà di controlli automatici a velivoli di piccole dimensioni, migliorandone la controllabilità, come nel caso di velivoli ad ala fissa o di piccoli elicotteri o permettendo il volo di velivoli altamente instabili, che difficilmente sarebbero potuti essere pilotati altrimenti. È il caso, in particolare, dei velivoli multirotore. Nella loro configurazione più diffusa, ovvero quella a quattro rotori, tali velivoli sono stati oggetto di studio fin dagli albori dell'aeronautica, quando Etienne Oehmichen presentò un prototipo funzionante denominato The Oehmichen No.2 (fig. 1). Vari tentativi vennero effettuati in seguito; il prototipo più vicino alla commercializzazione fu il Curtiss-Wright VZ-7 (fig. 2), che era stato pensato per l'esercito americano nel 1958. Tuttavia la duttilità e le potenzialità di questi velivoli non furono sufficienti a garantirne la diffusione, in quanto ingombranti e soprattutto poco manovrabili, poiché la disposizione a X delle eliche comporta una grossa instabilità difficilmente compensabile da un pilota.

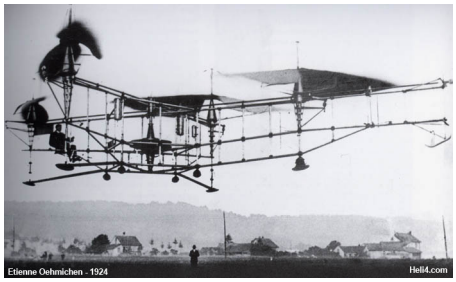


Figura 1: Il velivolo The Oehmichen n.2 in volo

Il problema è ulteriormente accentuato nel caso di piccoli velivoli radiocomandati, a causa di un aumento del tempo di reazione del pilota, il quale ha un feedback visivo e non inerziale, e a causa della dinamica molto rapida del velivolo. Finché nel 2002 si ottiene il primo controllo automatico [11] sul velivolo radiocomandato X4-flyer, gettando le basi per lo sviluppo di questi

Figura 2: Il velivolo Curtiss-Wright VZ-7 durante un test

sistemi. Ad oggi se ne contano svariati modelli, sia prototipi sviluppati da enti e università a scopo di ricerca, sia in commercio, con costi contenuti e dalle svariate caratteristiche.

Descrizione generale del progetto

Il Laboratorio di Meccanica del Volo dell'Università degli Studi di Bologna è da anni impegnato in progetti di controllo ambientale al fianco dell'Istituto Nazionale di Geofisica e Vulcanologia. Per ogni missione, in passato, è stato utilizzato un drone adeguato alle condizioni di volo ed equipaggiato con la necessaria strumentazione, il quale ha sorvolato la zona interessata. Tuttavia, si sta valutando la possibilità di utilizzare più velivoli contemporaneamente, in futuro, in modo tale da avere a disposizione più payload o di poter sopperire a esigenze diverse, utilizzando veicoli dalle differenti caratteristiche. In particolare, potrebbe risultare utile l'utilizzo combinato di un veicolo su ruote, in grado di essere in contatto con il terreno, e di un velivolo quadrirotore, in grado di stazionare e seguire il veicolo nei suoi spostamenti, in modo da avere una panoramica elevata della zona circostante. Da qui nasce la necessità di testare una legge di controllo in grado di comandare in cooperazione due o più veicoli. Lo scopo di questo lavoro è equipaggiare correttamente un velivolo quadrirotore in modo tale che possa, sia volare autonomamente, sia servire da base per un futuro test di volo cooperativo insieme ad un altro veicolo. Per fare questo ci si deve rivolgere, chiaramente, ad una architettura opensource. Qui nasce il lavoro, di studio e simulazione di algoritmi per architetture libere.

1 Sensori

Al fine di rendersi conto di come funziona un sensore reale, in modo da riprodurre fedelmente il funzionamento in ambiente di simulazione, nonché di poter effettuare test con dati realmente campionati, sono stati acquisiti alcuni sensori, della tipologia che normalmente costituisce l'equipaggiamento di un velivolo autonomo. Tutti i sensori sono stati in seguito riprodotti in ambiente Simulink, quale parte integrante del modello finale del velivolo multirotore presentato.

1.1 Sensori inerziali

Il primo strumento analizzato è un sensore inerziale, in grado di misurare accelerazioni e velocità angolari. Nel nostro caso è stato testato un dispositivo ADIS16350, prodotto dalla Analog Devices [10]. Come in ogni sensore, il valore di varianza del rumore dipende sensibilmente dalle condizioni ambientali, in particolare dalla temperatura. Il datasheet ufficiale fornisce dei valori di varianza e covarianza per quanto riguarda il rumore e bias nei tre assi sia del giroscopio che dell'accelerometro. Avendo a disposizione un termometro all'interno del sensore di pressione come vedremo in seguito, è possibile stimare tali valori in tempo reale e, teoricamente, aggiornarli durante la missione del velivolo, nel caso le condizioni ambientali variassero. Tuttavia test sperimentali hanno indicato una certa differenza, non trascurabile, tra i valori stimati e i valori misurati a una data temperatura. Se questo può essere trascurabile per quanto riguarda la covarianza del rumore, non è altrettanto vero per i valori di bias. Per cui si è scelto di valutare in anticipo ogni valore prima del volo. Nell'ottica di un velivolo autonomo, la procedura richiede un certo tempo, tuttavia se si considera che anche il gps deve agganciare i satelliti prima di ogni volo, il tempo di calibratura risulta ininfluente.

In seguito viene presentato lo pseudocodice dell'algoritmo di calibratura. Si noti che la media viene aggiornata ogni ciclo, non viene calcolata al termine del tempo selezionato per la calibratura, poiché la memoria RAM a disposizione nelle schede del tipo Arduino è molto ridotta ed è buona norma salvare in memoria il numero minore possibile di variabili.

```
1 void function valutazione bias
```

```
2  for (i = 1; i<=tempo di valutazione bias; i++) {
3    lettura accelerometro;
4    lettura giroscopio;
5    bias_a[0] = (bias_a[0]*i + acc[0])/(i+1);
6    bias_a[1] = (bias_a[1]*i + acc[1])/(i+1);
7    bias_a[2] = (bias_a[2]*i + acc[2]-g)/(i+1);
8    bias_g[0] = (bias_g[0]*i + gyr[0])/(i+1);
9    bias_g[1] = (bias_g[1]*i + gyr[1])/(i+1);
10   bias_g[2] = (bias_g[2]*i + gyr[2])/(i+1);;
11 }
12 void function valutazione varianze
13 for (i = 1; i<=tempo di valutazione varianze; i++) {
14   lettura accelerometro;
15   lettura giroscopio;
16   var_a[0] = (((acc[0]-bias_a[0])*(acc[0]-bias_a[0])) + var_a[0]*i)/(i+1);
17   var_a[1] = (((acc[1]-bias_a[1])*(acc[1]-bias_a[1])) + var_a[1]*i)/(i+1);
18   var_a[2] = (((acc[2]-bias_a[2]-g)*(acc[2]-bias_a[2]-g)) + var_a[2]*i)/(i+1);
19   var_g[0] = (((gyr[0]-bias_g[0])*(gyr[0]-bias_g[0])) + var_g[0]*i)/(i+1);
20   var_g[1] = (((gyr[1]-bias_g[1])*(gyr[1]-bias_g[1])) + var_g[1]*i)/(i+1);
21   var_g[2] = (((gyr[2]-bias_g[2])*(gyr[2]-bias_g[2])) + var_g[2]*i)/(i+1);
22 }
```

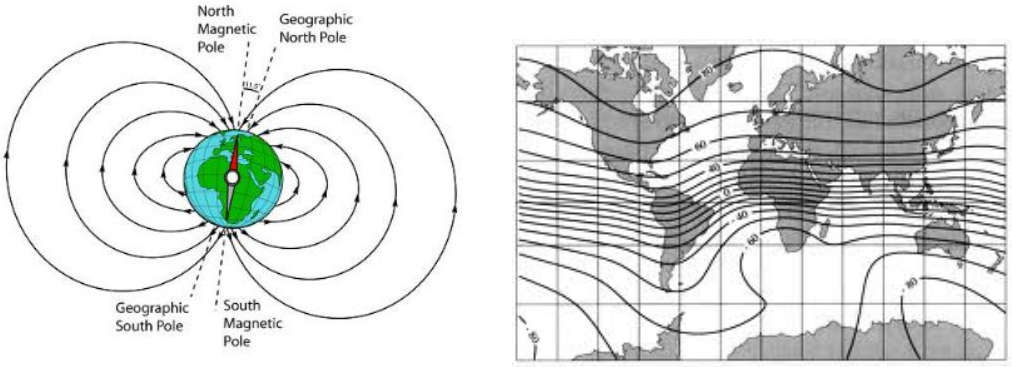
Si noti che lo pseudocodice proposto è valido per i linguaggi di programmazione tradizionali, mentre in ambiente Arduino il programma principale è in continuo loop e quindi il ciclo for va gestito diversamente, integrandolo nel ciclo generale. Inoltre, gli errori lungo i tre assi sono considerabili ragionevolmente indipendenti tra di loro, per cui la matrice di covarianza può essere considerata diagonale senza alcun problema.

Resta comunque il problema che tali valori, per quanto precisi, non tengono conto della deriva che gli strumenti subiranno durante la missione. Per cui bisogna fare due considerazioni importanti: primo, per quanto possa mutare il valore del bias, abbiamo a che fare con missioni della durata di pochi minuti, per cui l'errore finale sarà sempre minore dell'errore che si sarebbe compiuto stimando i valori con le leggi fornite nel datasheet. Non sono poche, infatti, le applicazioni in cui il bias del giroscopio viene considerato semplicemente costante, una volta valutato [17]. Secondo, esistono molto algoritmi per il calcolo dell'assetto e/o della posizione che stimano il bias in tempo reale o, perlomeno, sono robusti a un certo errore. Per cui, come vedremo nel capitolo dedicato, sarà possibile compensare ogni errore di questo tipo e ottenere ottimi risultati di stima, sebbene a discapito della semplicità del sistema e della potenza computazionale richiesta.

1.2 Magnetometro

Altra misura fondamentale per la navigazione autonoma è quella del campo magnetico. Il riferimento magnetico è teoricamente in grado di determinare da solo l'assetto sui tre assi, a meno del caso particolare in cui il corpo sia totalmente allineato al campo magnetico terrestre.

Il campo magnetico terrestre, sebbene venga spesso approssimato come un dipolo (fig. 1.1a), ha una forma più complessa, descritta in figura 1.1b



(a) Il campo come dipolo magnetico (b) La mappa dell'intensità reale

Figura 1.1: Rappresentazioni a confronto

Come si può notare dalla rappresentazione del dipolo magnetico, vi è un certo angolo di inclinazione del campo magnetico dovuto alla sfericità della superficie terrestre. Tale angolo è detto angolo di inclinazione (o dip) e varia con la latitudine. A Forlì è stato valutato essere circa 60.7°. A questo si aggiunge un più piccolo angolo detto declinazione relativo alla componente verso Est del campo magnetico. A Forlì vale 2.5°. Tale informazione può essere richiesta dalla tecnica di filtraggio utilizzata.

Un'altra informazione importante, la si ricava dalla mappa del campo magnetico. In particolare, si nota che il nord indicato dal magnetometro può discostarsi di molto da quello geografico, per cui se ci si affidasse al magnetometro come bussola digitale, si seguirebbe una direzione sbagliata. Specialmente nelle zone dell'America meridionale, infatti, l'aviazione generale non si può affidare a questo sistema di navigazione, a meno dell'utilizzo di un'accurata tabella correttiva. Inoltre, data la presenza di altri due poli di minore intensità, per lunghe tratte si può osservare una non linearità delle linee di flusso, con l'ago della bussola che viene attratto da entrambi i poli con intensità diversa a seconda della vicinanza dell'uno o dell'altro. Tuttavia, nella nostra applicazione e per piccole tratte, siamo semplicemente alla ricerca di un riferimento fisso, per cui affidarci al polo magnetico senza alcuna correzione non comporta alcun rischio.

Inoltre bisogna considerare che il campo magnetico terrestre subisce dei disturbi, i quali vengono suddivisi in due categorie:

- **Hard iron:** si tratta dell'effetto dovuto alla presenza di magneti permanenti in prossimità del sensore. Causa un offset del valore indicato rispetto a quello reale, tuttavia è relativamente semplice da compensare, avendo a disposizione un numero sufficiente di dati sperimentali.

- **Soft iron:** è dovuto principalmente al fatto che i metalli tendono a veicolare le linee di flusso magnetiche meglio dell'aria, per cui le linee di flusso del campo terrestre deviano

per passare all'interno di certi oggetti. In questo caso la compensazione è più difficile, perché varia a seconda della prua del sensore.

Ovviamente è possibile compensare entrambe queste situazioni tramite vari algoritmi presenti in letteratura, tra i quali si segnala uno tra i più utilizzati, presentato da M.J. Caruso nel 2000 [7]. Come vedremo meglio nella sezione del filtraggio, dobbiamo tener conto della presenza di rumori a media nulla (legati alla natura del sensore e non a deformazioni del campo) e/o ritardi al momento di eseguire la stima dell'assetto. Difatti la calibrazione del sensore e il filtraggio dei suoi dati, per quanto forniscano dei dati migliori, possono tradursi in tempi non trascurabili di elaborazione, per cui ci si può trovare di fronte ad una frequenza di aggiornamento dei dati inferiore a quella degli altri sensori

Il sensore che è stato acquisito nell'esperimento è del modello HMR3300 della Honeywell (fig. 1.2). La peculiarità di questo modello è un circuito integrato che elabora i segnali in uscita e toglie gli errori derivati da hard e soft iron, nonché il rumore gaussiano derivato dai disturbi elettronici. Dopo aver effettuato una seconda calibrazione con l'algoritmo di W. Premerlani [16] si è constatata l'effettiva accuratezza del segnale acquisito, nonché la minima varianza del segnale. Questo, tuttavia, diminuisce notevolmente il tempo di campionamento, per cui il sensore supporta al massimo 8 Hz di frequenza, con le conseguenti difficoltà che verranno approfondite nella sezione 3.

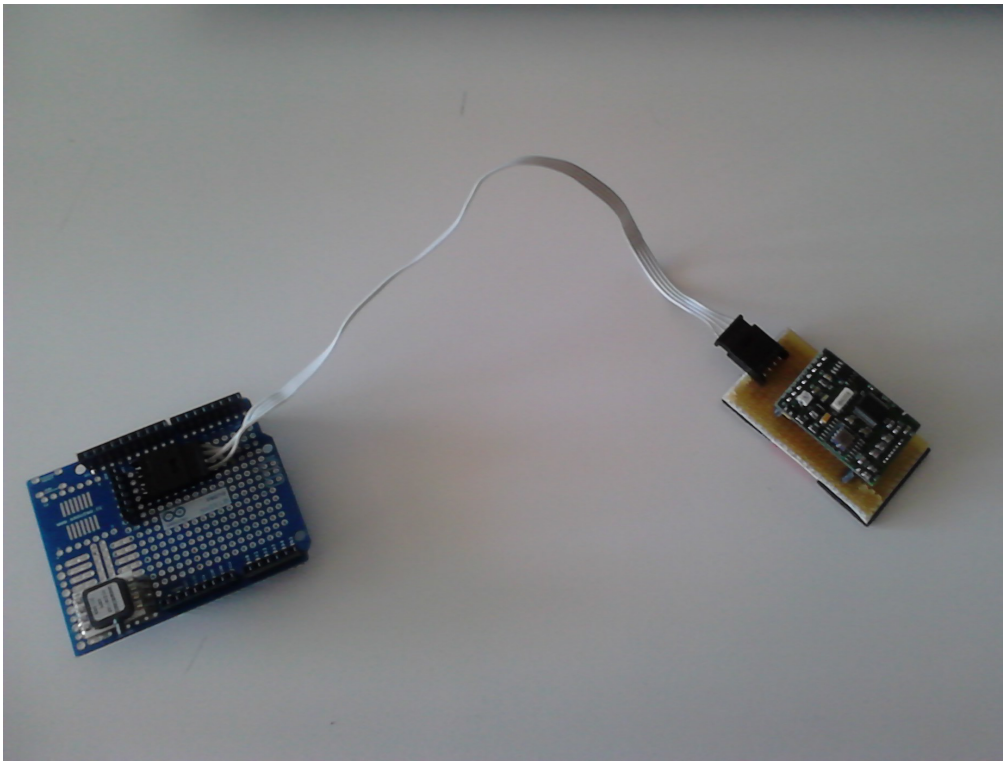


Figura 1.2: Il sensore HMR3300

1.3 GPS

Il sistema di navigazione satellitare, per quanto soffra di notevoli incertezze dovute all'inferiore qualità del segnale civile (canale L1) rispetto a quello utilizzato in ambito militare (canale L2), è comunque un valido ed economico sistema per individuare la posizione nello spazio di un velivolo. Nel nostro caso è stato impiegato un sensore del modello Lassen SQ (fig. 1.3a e 1.3b), prodotto da Trimble. Come il sensore magnetico, questo GPS ha la qualità di filtrare i dati in ingresso, fornendo un risultato molto accurato in termini di posizione e senza il tipico rumore che affligge questo tipo di strumentazione, a discapito della frequenza di aggiornamento che si ferma a 1 Hz. Tuttavia in questo caso è più che sufficiente, dato che si tratta di determinazione della rotta e non di assetto, un ritardo può comportare al più una deviazione dalla rotta desiderata, mentre nel caso dell'assetto un posizionamento sbagliato può avere conseguenze catastrofiche per il velivolo.

(a) Il modulo del ricevitore (b) L'antenna GPS

Figura 1.3: Il ricevitore Trimble LassenSQ

1.3.1 Ricezione dei dati

I GPS solitamente comunicano secondo lo standard della Marina Militare Americana (proprietaria del sistema di posizionamento), denominato NMEA. Tale standard standard prevede che i dati siano inviati all'interno di sentenze prestabilite, di cui ne esistono vari tipi. Il più diffuso è quello denominato GPRMC, per il quale esistono varie librerie di acquisizione opensource per ogni tipo di linguaggio. Una particolarmente utilizzata in ambiente Arduino è quella di M. Lamers [13]. Tuttavia, sebbene il modello Lassen SQ fosse pensato per comunicare con ogni tipo di sentenza NMEA, non è stato possibile modificare le impostazioni originarie, volte a fornire i dati ricevuti tramite sentenze GPGGA (relativa alla posizione) e GPVTG (reativa alla velocità). A questo proposito è stata scritta una libreria apposita, derivata da quella di Lamers e presente in appendice.

1.4 Sensore di pressione

Infine, per migliorare la qualità del posizionamento, si è rivelato necessario l'inserimento di un sensore di pressione. È noto, infatti, che la diluzione di precisione del segnale GPS raggiunge valori molto più elevati sull'asse verticale. Per ovviare a questo problema si può sostituire il valore della pressione atmosferica a quello della posizione verticale e quello della sua derivata al valore di velocità verticale, moltiplicato per un opportuno coefficiente. A tale scopo si consideri l'utile formula, presente in molti manuali di meccanica del volo [8], che lega quota e pressione atmosferica:

$$\frac{p}{p_0} = (1 - \frac{0.0065h}{T_0})^{\frac{1}{0.0065R}} \tag{1.1}$$

Il valore corrispondente al livello del suolo, ovviamente, dipende fortemente dalle condizioni atmosferiche, per cui può essere considerato costante solo per la durata di una breve missione. Nella sezione 3.2.1, dedicato al filtraggio dei dati, vedremo più in dettaglio come può essere utilizzata questa misura. Infine nella 1.1 si nota che necessitiamo della temperatura T_0. A questo scopo, infatti, è stato utilizzato un sensore HCE0611AR (fig. 1.4) che integra sia un barometro che un termometro e comunica con la scheda Arduino MEGA tramite porta seriale.

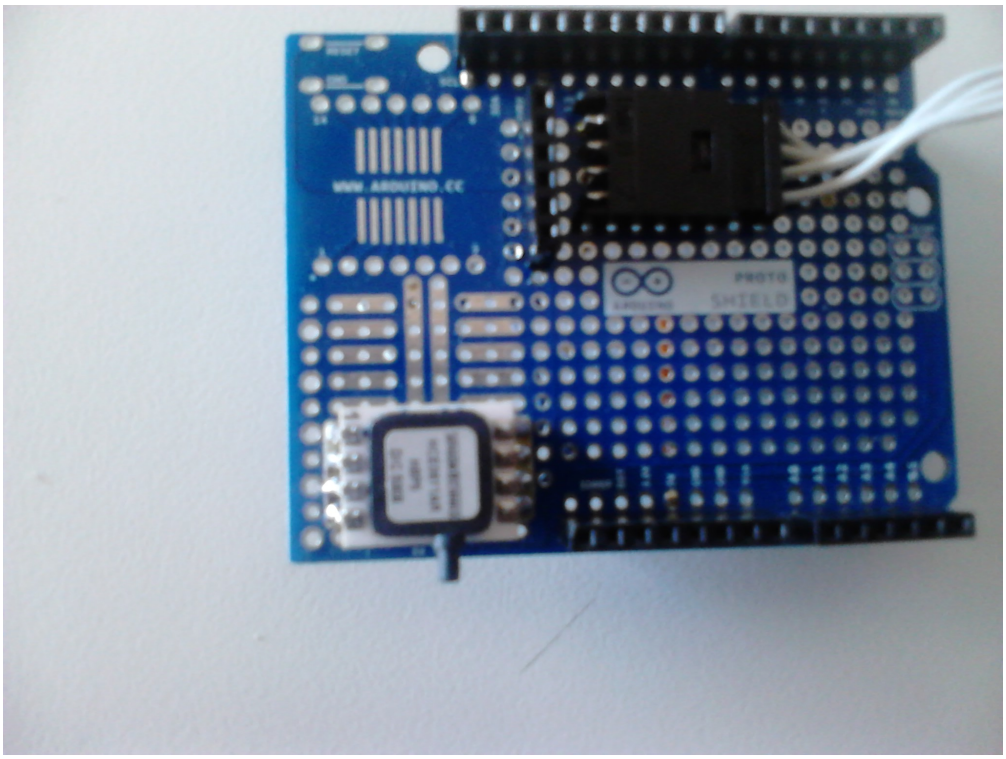


Figura 1.4: Il sensore HCE0611AR montato sullo shield Arduino

2 Modellazione

Al fine di testare in ambiente di sviluppo la legge di controllo che sarà applicata al velivolo, è necessario utilizzare un modello dinamico del velivolo. Si espone quindi di seguito il modello implementato, basato fondamentalmente sul lavoro di T. Hamel *et al.* [11].

2.1 Modello dinamico di un velivolo quadrirotore

Dati i sistemi di riferimento precedentemente definiti $\mathbf{F_b} = \{x_b, y_b, z_b\}$ quello legato agli assi corpo e $\mathbf{F_e} = \{x_e, y_e, z_e\}$ il sistema di navigazione fissato alla terra, abbiamo $\mathbf{R} : \mathbf{F_b} \rightarrow \mathbf{F_e}$ tale che $\mathbf{R} \in SO(3)$. Definiamo, quindi, come la velocità lineare espressa nel sistema assi-corpo e sia Ω la velocità angolare del sistema mobile rispetto a quello fisso. Perciò la derivata rispetto al tempo della posizione dell'origine del sistema assi corpo vale:

$$\dot{\xi} = \mathbf{v}. \tag{2.1}$$

Eseguendo il bilancio delle forze agenti si ottiene:

$$m\dot{\mathbf{v}} = mg\mathbf{e}_3 + \mathbf{RF} \tag{2.2}$$

Dove la dinamica della matrice di rotazione $\mathbf{R}$ è data da:

$$\dot{\mathbf{R}} = \mathbf{R} \cdot sk(\omega) \tag{2.3}$$

dove $sk(\omega)$ è la matrice antisimmetrica tale che $sk(\omega) \cdot \mathbf{b} = \Omega \wedge \mathbf{b}$ per ogni vettore $\mathbf{b}$. Quindi

abbiamo, per quanto riguarda la dinamica della rotazione:

$$\mathbf{I} = -\Omega \wedge \mathbf{I}\Omega + \Gamma \tag{2.4}$$

dove **I** è la matrice di rotazione del quadrirotore e Γ è il momento torcente agente. Per valutare quest'ultimo, dobbiamo conoscere la relazione:

$$\mathbf{I}_r \dot{\omega}_i = \tau_i - Q_i \tag{2.5}$$

Dove $\mathbf{I}_r$ rappresenta l'inerzia del rotore, τ_i il momento torcente applicato e Q_i il momento torcente aerodinamico (per ciascun rotore, i)

Definiamo come al solito la portanza generata da ogni rotore come:

$$L_i = -b\omega_i^2 \mathbf{e}_3 \tag{2.6}$$

dove b è il coefficiente aerodinamico (determinato sperimentalmente). Allo stesso modo il momento generato dalla resistenza aerodinamica misura:

$$Q_i = k\omega_i^2 \tag{2.7}$$

Quindi la trazione totale vale:

$$T = \sum_{i=1}^{4} |L_i| = b(\sum_{i=1}^{4} \omega_i^2) \tag{2.8}$$

mentre il momento torcente aerodinamico è valutato come:

$$\tau^a = (\tau_1^a, \tau_2^a, \tau_3^a) \tag{2.9}$$

dove:

$$\tau_1^a = db(\omega_2^2 - \omega_4^2)\tau_2^a = db(\omega_1^2 - \omega_3?2)\tau_3^a = k(\omega_2^2\omega_4^2\omega_1^2\omega_3^2).$$

Infine c'è un ulteriore momento torcente agente sul rotore. Possiamo esprimere il momento giroscopico come:

$$\mathbf{G}^a = -\sum_{i=1}^{4} \mathbf{I}_r(\Omega \wedge \mathbf{e}_3)\omega_i. \tag{2.10}$$

In pratica abbiamo un modello composto dal seguente set di equazioni:

- $\dot{\xi} = \mathbf{v}$
- $\dot{\mathbf{v}} = g\mathbf{e}_3 - \frac{1}{m}\mathbf{TRe}_3$
- $\dot{\mathbf{R}} = \mathbf{R}sk(\Omega)$
- $\mathbf{I}\dot{\Omega} = -\Omega \wedge \mathbf{I}\Omega + \mathbf{G}^a + \tau^a$
- $\mathbf{I}_r\dot{\omega}_i = \tau_i - k\omega_i$

Le prime quattro espressioni esprimono la dinamica del corpo rigido, mentre l'ultima lega l'input τ_i alle forze e ai momenti agenti sul corpo. Tale equazione può essere riscritta, comprendendo anche l'espressione della spinta, in forma matriciale, risultando:

$$\begin{bmatrix} T \\ \tau_1^a \\ \tau_2^a \\ \tau_3^a \end{bmatrix} = \begin{bmatrix} -b & -b & -b & -b \\ 0 & db & 0 & -db & 0 \\ k & -k & k & -k \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} := \mathbf{A} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}. \tag{2.11}$$

Inoltre, si può dimostrare che se $b, k, d > 0$, $\mathbf{A}$ ha rango massimo. Ciò, nella pratica, è sempre vero, per quindi la soluzione è sempre univocamente determinata.

Le velocità angolari delle eliche, tuttavia, non sono variabili direttamente legate al sistema di controllo. Sarà quindi necessario fornire una valida modellazione che leghi il comando inviato al motore con la velocità imposta alle eliche.

2.2 Modello dinamico delle eliche

La maggior parte dei motori elettrici utilizzati in questo ambito sono controllati in voltaggio. È possibile modellare la relazione che c'è tra il voltaggio in input e la velocità angolare della pala come la somma di due transitori, quello legato alla reattività del motore e quello legato all'aerodinamica dell'elica. Il guadagno totale delle due funzioni è quello che dà la relazione tra voltaggio e velocità angolare in condizioni stazionarie, mentre le due dinamiche sono state approssimate come poli del primo ordine. I valori usati, rispettivamente per: guadagno aerodinamico, costante di tempo aerodinamica, guadagno elettrico, costante di tempo elettrica, valgono:

- $K_a = 0.8$
- $\tau_a = 1/100$
- $K_e = 20$
- $\tau_e = 1/20$

2.2.1 Resistenza aerodinamica

Per ottenere un modello realistico della dinamica nell'atmosfera del velivolo sono state modellate le resistenze aerodinamiche nei tre assi. La formula della resistenza aerodinamica è espressa nella nota forma:

$$D_i = \frac{1}{2}\rho S_i U_i^2 C_D \tag{2.12}$$

dove S_i e U_i rappresentano rispettivamente la superficie frontale e la componente della velocità relativa dell'aria nell'asse i, appartenente alla terna assi corpo. Il valore di C_D dipende dalla geometria del velivolo e deve chiaramente essere testato sperimentalmente. Data l'esilità della struttura in sé, le resistenze saranno principalmente legate alla geometria del rotore, ma comunque variano a seconda del velivolo impiegato.

2.3 Controllo

Data l'elevata instabilità della dinamica dei velivoli multirotore, è fondamentale fornire un controllo automatico adeguato. Molti approcci sono stati presentati in passato, sia lineari che non lineari. Tra questi citiamo:

- PID [5]

- LQ [6]
- PieceWise Affine Model (PWA) [2]
- Backstepping [11]
- Hybrid Backstepping [9]

Tuttavia, indagando a fondo, si scopre che, a meno di particolari richieste di capacità acrobatiche, i risultati forniti dai controllori PID, come ad esempio [5], non sono peggiori delle altre tipologie, come attestato in parte da [6]. In compenso i vantaggi di questo sistema sono molteplici: primo tra tutti la semplicità della teoria, che si traduce anche in semplicità di implementazione e leggerezza di calcolo; secondo, il fatto che l'approccio SISO (Single Input Single Output) consente di semplificare il lavoro di tuning dei parametri, poiché l'operatore che debba eseguire il lavoro può controllare intuitivamente un parametro alla volta.

Infine, per quanto riguarda il controllo della posizione, si possono citare molte diverse teorie presentate in passato, quali [11], basata sul backstepping, o [1], basata sulla tecnica di inversione dinamica. Tuttavia, in questo lavoro ci si è concentrati su altri aspetti di navigazione, quindi per la posizione è stato implementato un semplice ed efficace controllore PID che agisca sui comandi passati al controllo di assetto per modificare o mantenere la posizione laterale e sul voltaggio totale per mantenere la quota desiderata.

2.4 Simulazione

Per osservare effettivamente la dinamica che il velivolo effettua seguendo le leggi descritte, il modello è stato implementato in Simulink e alcuni comandi sono stati forniti per visualizzare il comportamento del velivolo. Per semplificare la visualizzazione, i disturbi dovuti al vento sono stati imposti nulli, sebbene sia possibile inserire fenomeni di vento laterale, raffiche e turbolenze. Il velivolo simulato presenta le seguenti caratteristiche:

- $m = 1.6kg$
- $I = \begin{bmatrix} 0.0815 & 0 & 0 \\ 0 & 0.0815 & 0 \\ 0 & 0 & 0.1624 \end{bmatrix}$
- $I_r = 0.000005$
- $b = 0.00063$
- $d = 0.05$
- $k = 3.1500e-05$.

Le prestazioni di volo sono presentate in fig. 2.1, 2.2, 2.3 e 2.4. Si nota che, almeno qualitativamente, le dinamiche rispecchiano quelle aspettate intuitivamente. Questa può essere considerata come una prova della validità del modello. Si nota, inoltre, come il controllo di assetto e di posizione (qui attivo solo per l'asse verticale), porti il velivolo esattamente nella condizione desiderata.

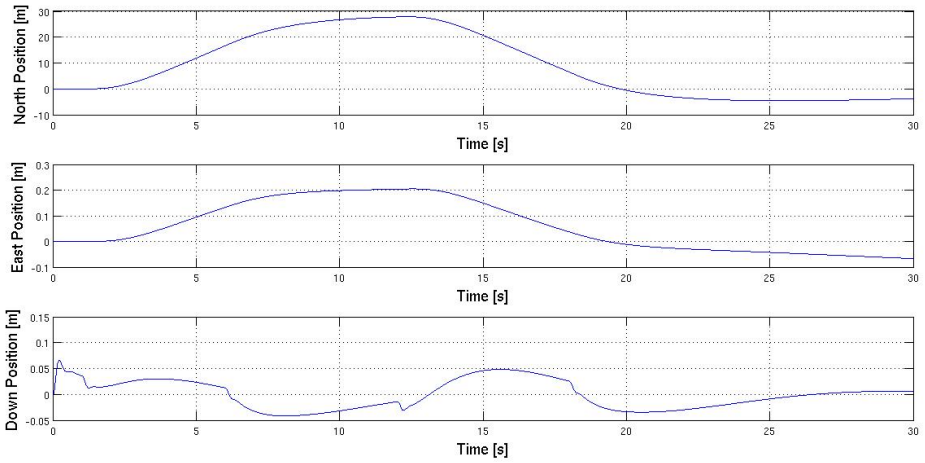


Figura 2.1: Andamento della posizione controllata agendo sull'angolo di beccheggio

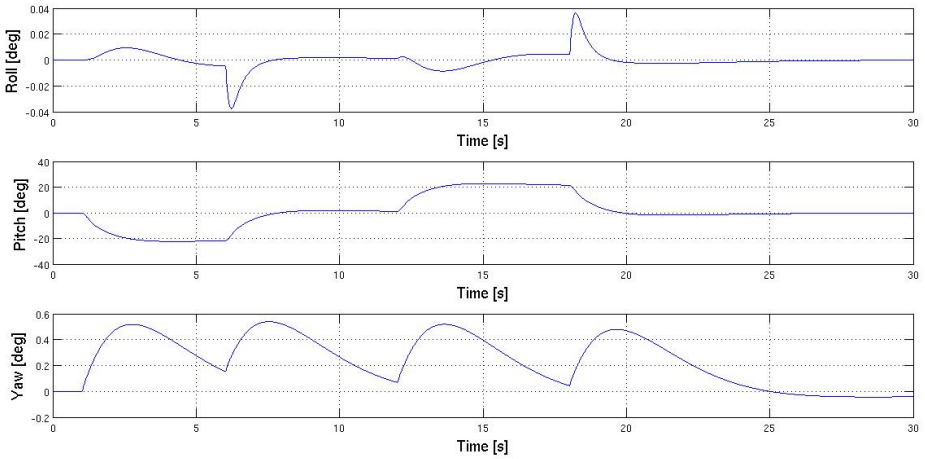


Figura 2.2: Angoli di assetto relativi agli spostamenti nel grafico precedente

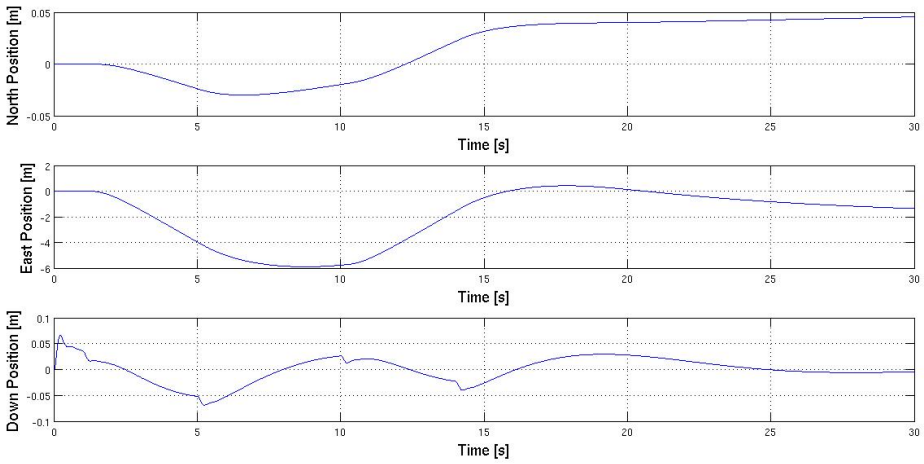


Figura 2.3: Andamento della posizione controllata agendo sull'angolo di rollio

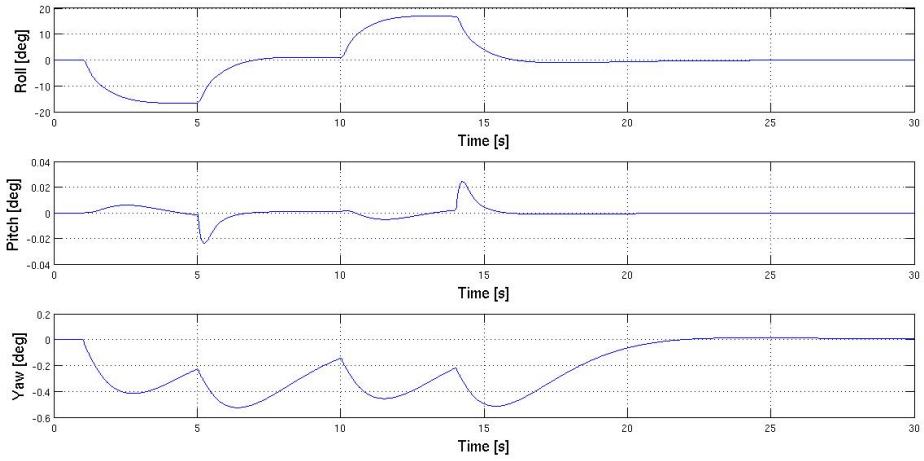


Figura 2.4: Angoli di assetto relativi agli spostamenti nel grafico precedente

3 Filtraggio dei dati

3.1 Determinazione d'assetto

Una parte fondamentale nel progetto di ogni sistema di volo automatizzato o semi-automatizzato è la determinazione dell'assetto del velivolo. I metodi sono svariati, sviluppati a partire dagli anni '60 e variano tra di loro in precisione, complessità di calcolo e stabilità nel tempo. La complessità computazionale richiesta da un filtro di Kalman nonlineare ha portato spesso a cercare valide alternative che richiedessero un algoritmo più semplice. Se oggi il problema si può presentare solo all'interno di piccoli velivoli che sfruttano processori di gamma bassa, lo stesso non valeva negli anni '70, quando si arrivò a confrontare i risultati di un filtro complementare e di un filtro di Kalman per l'utilizzo a bordo dello Space Shuttle [12]. La maggior parte dei progetti commerciali di velivoli quadrirotore, compreso l'autopilota ArduPilot, preso a rifermento in questa trattazione, implementano un filtro complementare. Tuttavia questo presenta notevoli svantaggi, primo tra tutti la necessità di una calibrazione molto precisa dei sensori, che non è un'ipotesi scontata nel caso di missioni più lunghe di qualche minuto, a causa della deriva del bias dei vari sensori. Secondo, il filtro complementare si basa sull'idea di una posizione di trim intorno allo zero e, per quanto questa ipotesi sia poco azzardata nel senso che raramente un quadrirotore non acrobatico dovrebbe raggiungere angoli più elevati di $15-20°$, è preferibile abolire ogni limitazione a priori. Per questo motivo sono state implementate due diverse tipologie di filtro d'assetto che vedremo nel dettaglio, le quali sono state precedentemente testate in fase di simulazione e confrontate con il filtro complementare presente nel firmware ArduPilot. A queste si aggiunge un filtro di Kalman che è attualmente in fase di sviluppo da parte dello studente di dottorato Gastone Ferrarese.

3.1.1 Calibrazione

Sebbene non sia al centro di questa trattazione, occorre spendere due parole sulla calibrazione degli strumenti di bordo. Per quanto riguarda l'accelerometro e il giroscopio, occorre prima di tutto conoscere la distanza che c'è tra il baricentro del velivolo, che sarà il fulcro delle rotazioni e la posizione dei sensori, in modo da poter compensare le accelerazioni apparenti misurate.

Se possibile, chiaramente, in fase di progetto si cercherà sempre di ridurre al minimo questo valore, tuttavia, considerando che tali velivoli vengono impiegati per trasportare payload disparati tra di loro, sarà difficile anche solo conoscere la posizione esatta del centro di massa e questo si tradurrà in un aumento del rumore sulle misure di accelerazione quando la velocità angolare non è nulla. Una volta conosciuta la struttura del velivolo, la cosa più importante diventa identificare i bias momentanei. Per fare ciò è necessario tenere la strumentazione ferma per un certo numero di secondi prima del decollo, seguendo la procedura spiegata nel capitolo 1. Si noti, però che questo non è sufficiente, poiché i bias varieranno durante la missione. Vedremo in seguito come è possibile ovviare a questo problema.

Infine, per quanto riguarda il magnetometro, si richiede sempre una procedura di calibrazione, che tuttavia è più complessa rispetto a quella della IMU e richiede spostamento del sensore, non dei dati fissi, motivo per cui viene eseguita durante tutta la durata della missione. Molti algoritmi sono presenti in letteratura, tra i quali l'ambiente ArduPilot ne implementa uno valido. Si tratta di una versione leggermente modificata dell'algoritmo di W. Premerlani [16]. Le differenze principali rispetto all'algoritmo originale sono due:

- La versione modificata crea un buffer di dati che divide il vettore dei campi magnetici in insiemi da N flussi di dati (20 nel codice ArduPilot). L'algoritmo è eseguito su ogni flusso separatamente
- In ogni flusso vengono calcolate le variazioni solo se il vettore campo magnetico è cambiato di un quantitativo significativo.

In questo modo l'algoritmo valuta velocemente gli offset se i sensori sono poco rumorosi, ma funziona bene, anche se più lentamente, in caso di sensori poco precisi. Oltre a questo algoritmo è stato implementato il più dispendioso filtro di Kalman descritto in [15], che è stato utilizzato in combinazione dei filtri di Kalman per il calcolo dell'assetto.

3.1.2 Filtro complementare

Una volta ottenuti dei dati centrati, questi resteranno comunque rumorosi, nonché privi di significato direttamente interpretabile in termini di assetto o posizione. Per questa ragione si procede alla fase di filtraggio. I filtri complementari, nonostante la loro semplicità, permettono di ottenere ottini risultati in termini di pulizia, a discapito del range valido per l'estimazione, della prontezza e di un degrado delle performance nel tempo. In seguito viene descritto brevemente il suo funzionamento, compatibilmente con la versione adottata dal firmware ArduPilot.

Algoritmo Sfortunatamente, ArduPilot non fornisce una documentazione dettagliata riguardo gli algoritmi utilizzati, per cui l'algoritmo qui descritto è un'estrapolazione del codice

presente nel firmware, anche grazie ad un lavoro, presentato recentemente come tesi presso l'Università di Pisa, che presenta la traduzione in codice Matlab/simulink dell'algoritmo di filtraggio ArduPilot. Vediamone, quindi, i principali passaggi.

L'algoritmo si basa, fondamentalmente, su una stima della matrice di assetto ottenuta integrando le velocità angolari. Questa stima, ovviamente, non può essere inizializzata e comunque soffre di derive dovute alle imprecisioni dei giroscopi. Per questo motivo la stima viene aggiustata tramite dei termini correttivi dovuti alle misure dell'accelerometro e alle misure del magnetometro. In particolare, si può notare che il vettore gravità sarà diretto principalmente verso il basso e quindi darà una buona stima sugli angoli di beccheggio e di rollio, mentre avendo dei piccoli valori lungo gli assi Nord ed Est, una triangolazione in questo senso soffrirebbe molto dei rumori di misura. Per questo motivo, il termine correttivo dell'angolo di imbardata deriva dal magnetometro, che, al contrario dell'accelerometro, viene utilizzato solo per individuare il Nord e quindi la rotazione attorno all'asse verticale. Il fatto di poter utilizzare il magnetometro praticamente come una bussola, elimina il problema di individuare l'angolo di Inclinazione, poiché non influisce sulla misura. L'algoritmo è, quindi, valido in ogni parte del mondo.

La matrice di rotazione é individuata come segue:

$$\mathbf{R_{n+1}} = \mathbf{R_n} + \begin{bmatrix} R(1,2)g(3,1) - R(1,3)g(2,1) & R(1,3)g(1,1) - R(1,1)g(3,1) & R(1,1)g(2,1) - R(1,2)g(1,1) \\ R(2,2)g(3,1) - R(2,3)g(2,1) & R(2,3)g(1,1) - R(2,1)g(3,1) & R(2,1)g(2,1) - R(2,2)g(1,1) \\ R(3,2)g(3,1) - R(3,3)g(2,1) & R(3,3)g(1,1) - R(3,1)g(3,1) & R(3,1)g(2,1) - R(3,2)g(1,1) \end{bmatrix} \tag{3.1}$$

dove g è dato da:

$$g = (gyro_{meas} + \Omega_I + \Omega_P + \Omega_{P,yaw})\Delta t \tag{3.2}$$

I valori Ω_P e Ω_I sono valori correttivi derivanti dalle misure di accelerazione. Se definiamo $\hat{e} = \{\phi,\theta,\psi\}_{acc} - \{\phi,\theta,\psi\}_{gyr}$, ovvero la differenza tra la misura di assetto ottenuta con i giroscopi, veloce e precisa, ma poco accurata, con la misura ottenuta dall'accelerometro, più lenta e rumorosa, ma con errori a media verosimilmente nulla, possiamo valutare tali valori come:

$$\Omega_P = \hat{e} * P_gain(spinrate) * K_p; \tag{3.3}$$

$$\dot{\Omega}_I = \hat{e} * K_i \tag{3.4}$$

Dove K_p e K_i sono guadagni scelti dall'operatore in base alla qualità dei sensori, nel codice ArduPilot possono essere modificati, ma valgono normalmente circa $0.2 \sim 0.3$ e 0.01 rispettivamente. Il termine P_gain viene inerito per aumentare il guadagno nel caso di elevato spin rate, ma normalmente vale 1. Il valore di $\dot{\Omega}_I$, chiaramente, va integrato numericamente per ottenere Ω_I. In pratica, Ω_P e Ω_I e i rispettivi guadagni, rappresentano un controllore proporzionale integrale usato per portare a zero la deriva della matrice di rotazione stimata. Il terzo componente del vettore Ω_I verrà calcolato in maniera analoga, ma basandosi sulla misura del magnetometro. In questo caso i valori di $K_{p,yaw}$ e $K_{i,yaw}$ possono essere leggermente differenti, dato che la qualità degli strumenti è diversa. Si noti che dopo ogni passaggio la matrice di rotazione cambierà il valore della sua norma a causa di errori numerici di troncamento, per cui è preferibile rieseguire una normalizzazione dopo ogni passaggio di integrazione.

Inoltre, poiché si utilizza un integratore numerico, il sistema si comporta egregiamente come filtro passa basso ed elimina il rumore. La costante di tempo del filtro dipende, chiaramente, dal passo di integrazione. Per quanto riguarda il valore correttivo integrale, il sistema ArduPilot, ad esempio, utilizza un tempo di integrazione cinque volte maggiore rispetto a quello proporzionale per evitare di integrare troppo rumore e annullare l'utilità del guadagno integrale.

L'algoritmo presenta le note caratteristiche del filtro complementare, buona precisione e accuratezza al costo di un algoritmo molto leggero, tuttavia non essendo in grado di stimare i bias dell'accelerometro, ci si affida sempre ad una correzione che può non essere del tutto precisa. Inoltre si nota che le prestazioni degradano sensibilmente nel caso di angoli di beccheggio e rollio significativamente maggiori di zero. Nella sezione 3.1.5 esamineremo le prestazioni nel dettaglio

3.1.3 Filtro a gradiente coniugato

Una valida alternativa si deve al lavoro di Madgwick *et al.* [14]. Proposto nel 2011, sfrutta la tecnica di ottimizzazione del gradiente coniugato per valutare il minimo di una funzione che rappresenta l'errore sulla misura di assetto e quindi, indirettamente, calcola l'assetto stesso. In particolare, il filtro esiste in due versioni, la prima che non implica l'utilizzo del magnetometro, la seconda che invece utilizza tutti i sensori a disposizione. Per questa breve introduzione ci riferiamo alla funzione più completa.

Il concetto base è quello di valutare il quaternione di posizione come una semplice integrazione discreta dell'equazione 3.14 e di pesare la misura con quella derivata da accelerometro e magnetometro, in maniera analoga a quanto avviene nei filtri complementari. Tuttavia in questo caso varia la tecnica di estrapolazione dell'informazione dagli ultimi due sensori.

Dati i due vettori $\tilde{g} := \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ e $\tilde{m} := \begin{bmatrix} 0 & b_x & 0 & b_z \end{bmatrix}$ contenenti il valore del campo gravitazionale e del campo magnetico espressi nel sistema di riferimento terrestre e, dati i vettori $\hat{a} := \begin{bmatrix} 0 & a_x & a_y & a_z \end{bmatrix}$ e $\hat{m} := \begin{bmatrix} 0 & m_x & m_y & m_z \end{bmatrix}$ contenenti i valori misurati dal sensore, cercheremo il quaternione di assetto tale per cui la matrice di rotazione corrispondente dia una relazione valida per entrambe queste coppie di vettori. Trovare tale valore è teoricamente impossibile, tuttavia possiamo cercare la funzione che minimizza l'errore. In pratica si tratta di ricavare il valore di $\hat{q}$ che minimizzi la funzione

$$f = \hat{q}^* \otimes \hat{d}_e \otimes \hat{q} - \hat{s}_b \tag{3.5}$$

dove $\hat{d}_e$ è un generico vettore misurato nel sistema di riferimento terrestre e $\hat{s}_b$ è lo stesso vettore nel campo di riferimento legato agli assi corpo, mentre il simbolo $\hat{q}^*$ indica il vettore complesso coniugato di $\hat{q}$. Si noti che questo equivale a moltiplicare il primo vettore per la matrice di rotazione derivata dal vettore $\hat{q}^*$ come indicato nella sezione del filtro di Kalman. Si dimostra ([14]) che tale funzione f vale, nel caso del campo gravitazionale

$$f_g = \begin{bmatrix} 2(q_2 q_4 - q_1 q_3) - a_x \\ 2(q_1 q_2 + q_3 q_4) - a_y \\ 2(\frac{1}{2} - q_2^2 - q_3^2) - a_z \end{bmatrix} \tag{3.6}$$

e, nel caso del campo magnetico

$$f_m = \begin{bmatrix} 2b_x(0.5 - q_3^2 - q_4^2) + 2b_z(q_2 q_4 - q_1 q_3) - m_x \\ 2b_x(q_2 q_3 - q_1 q_4) + 2b_z(q_1 q_2 + q_1 q_4) - m_y \\ 2b_x(q_1 q_3 + q_2 q_4) + 2b_z(0.5 - q_2^2 - q_3^2) - m_z \end{bmatrix} \tag{3.7}$$

Per cui si cerca un minimo alla funzione

$$f_{g,m} := \begin{bmatrix} f_g \\ f_m \end{bmatrix} \tag{3.8}$$

tramite un qualunque algoritmo di ottimizzazione. L'algoritmo di discesa tramite gradiente coniugato si presta particolarmente bene a causa della sua semplicità teorica e implementativa

e richiede la valutazione dello jacobiano della matrice f, dato da:

$$J_{g,m} := \begin{bmatrix} J_g \\ J_m \end{bmatrix} \tag{3.9}$$

Inoltre è necessario ricordare che l'algoritmo del gradiente coniugato è iterativo, quindi va inizializzato e, teoricamente, a ogni valutazione in loop dell'assetto, dovrebbe compiere più iterazioni. Tuttavia i test dimostrano che inizializzando a 0, l'algoritmo non fornisce mai problemi di convergenza e che se ad ogni passo utilizziamo come punto di partenza la condizione di assetto valutata l'istante precedente, l'algoritmo converge ad un valore accettabile alla prima iterazione. Per cui l'unico problema è causato da un transitorio iniziale durante il quale l'algoritmo deve andare a regime, ma questo non influisce minimamente sul compimento della missione.

Si noti infine che l'algoritmo richiede un parametro γ_t che pesi l'importanza dell'assetto ricavato tramite ottimizzazione e di quello ricavato dai giroscopi, in modo che:

$$\hat{q} = \gamma_t \hat{q}_\nabla + (1 - \gamma_t)\hat{q}_\omega \tag{3.10}$$

dove $\hat{q}_\nabla$ e $\hat{q}_\omega$ rappresentano i valori stimati rispettivamente tramite l'ottimizzazione matematica e tramite l'integrazione delle velocità angolari. Si può dimostrare che il valore ottimale di questo peso è funzione del rumore e può essere descritto come:

$$\gamma_t = \frac{\beta}{\frac{\mu_t}{\Delta t} + \beta} \approx \frac{\beta \Delta t}{\mu_t} \tag{3.11}$$

Dove β può essere descritta in funzione del rumore massimo sulla misura di velocità angolare $\omega_{\tilde{max}}$

$$\beta = \sqrt{\frac{3}{4}}\tilde{\omega}_{max}. \tag{3.12}$$

Infine si può notare che l'algoritmo non è in grado di valutare in loop il bias, degli strumenti e quindi richiede una calibrazione iniziale come accade nel caso di un filtro complementare; tuttavia, il fatto che ogni misura si affidi a tutte le tre misurazione, fa sì che si ottengano risultati stabili anche in presenza di un certo offset degli strumenti dovuto alla deriva delle

loro prestazioni, oltre al fatto che la qualità del filtro è indipendente dall'assetto in cui si trova il velivolo. I risultati saranno analizzati nella sezione dedicata.

3.1.4 Filtri di Kalman

Introduzione La tipologia di filtro più diffusa in assoluto quando si tratta di ricostruire uno stato partendo da un modello dinamico e da alcune misurazioni è senza dubbio quella del filtro di Kalman. Nel caso della determinazione d'assetto, risulta sconveniente utilizzare la dinamica del velivolo come funzione di trasferimento all'interno del filtro di Kalman, in quanto ci sono molte incertezze dovute ai comandi che entrano come input, ovvero: se possiamo sapere l'impulso che forniscono i motori alla dinamica del velivolo, dato che li comandiamo noi, sebbene con le incertezze dovute alla dinamica interna dei motori, sarebbe ben più difficile conoscere le forze aerodinamiche che agiscono su un velivolo di piccole dimensioni come quello in questione con una precisione sufficiente.

Per questo ci si riferisce alla dinamica del corpo rigido, dove le velocità angolari sono le derivate temporali degli angoli di Eulero, le accelerazione angolari le derivate di queste ultime e così via. Gli strumenti a disposizione sono in grado di misurare il campo magnetico, il quale può essere confrontato con quello terrestre, l'accelerazione, la quale può essere confrontata con quella presente in un sistema di riferimento inerziale prestabilito e le velocità angolari, le quali possono essere integrate e fornire un aggiornamento più veloce della misura d'assetto. Si noti che l'ultima misurazione è teoricamente superflua, in quanto basterebbero le prime due per determinare univocamente una stima d'assetto, ad esempio col metodo delle triadi, mentre le prime due forniscono un vettore tridimensionale che non è sufficiente a determinare la matrice di rotazione rispetto a un vettore di confronto, in quanto ricade in un caso di parziale indeterminazione se i vettori sono allineati. In realtà, è impossibile che i due vettori si trovino perfettamente allineati, ma ad una situazione di vettori quasi allineati corrisponderebbero errori molto larghi.

La dinamica di assetto del corpo rigido è fortemente nonlineare, per cui è da scartare a priori l'utilizzo di un filtro di Kalman classico. In compenso una grande varietà di algoritmi riconducibili ai cosiddetti Extended Kalman Filter è stata presentata in letteratura. Tra questi, degno di nota è sicuramente l'algoritmo presentato da A.M. Sabatini nel 2006 [17]. Tra le capacità di questo algoritmo c'è quella tipica degli EKF di valutare in loop le variazioni sulla calibratura degli strumenti, semplicemente aumentando la variabile di stato. Tuttavia, questa autorevole versione dell'algoritmo valuta i valori di bias per quanto riguarda il magnetometro e l'accelerometro, ma considera come ipotesi fondante il fatto che il giroscopio possa essere calibrato a priori. Tuttavia, l'esperienza smentisce questa assunzione, almeno per quanto riguarda la strumentazione adottata e le condizioni ambientali in cui è stata testata, perlomeno su scale temporali più lunghe di qualche minuto. Per questo si è deciso di testare un algoritmo differente, pubblicato in tempi più recenti da Y.S. Suh [19]. A seguito si riportano i concetti fondamentali dell'algoritmo. Oltre a questo viene presentato un filtro progettato dal

dottorando Gastone Ferrarese e non ancora pubblicato.

Algoritmo di Y.S. Suh Definiamo due sistemi di riferimento: sia $\mathbf{F_b} = \{x_b, y_b, z_b\}$ il sistema di riferimento legato agli assi corpo e $\mathbf{F_e} = \{x_e, y_e, z_e\}$ il sistema di navigazione fissato alla terra; essi sono legati dalla nota relazione:

$$\mathbf{v_b} = \mathbf{C}(q) * \mathbf{v_e} \tag{3.13}$$

dove **C** è la matrice di rotazione tra i due sistemi di riferimento espressa in funzione del quaternione di assetto $q = \{q_0, q_1, q_2, q_3\}$. La derivata di q è definita da:

$$\dot{q} = \frac{1}{2} q \otimes \omega \tag{3.14}$$

dove ω è la velocità angolare nei tre assi. Integrando l'equazione 3.14 si può ottenere una stima dell'assetto del velivolo, che tuttavia dovrà essere inizializzata e, cosa più importante, aggiornata istante per istante correggendone gli errori derivati da rumori e bias. Per questo si introduce la rappresentazione:

$$q = \hat{q} \otimes \tilde{q}_e \tag{3.15}$$

nella quale $\hat{q}$ rappresenta il quaternione stimato. Per cui il quaternione correttivo diventerà lo stato da valutare all'interno del filtro di Kalman. Si può dimostrare ([19]) che la prima componente è sempre simile a 1, per cui scriveremo:

$$\tilde{q}_e \approx \begin{bmatrix} 1 \\ q_e \end{bmatrix} \tag{3.16}$$

e cercheremo un vettore di tre componenti come incognita. Aumentando lo stato delle incognite con i valori di bias del giroscopio e dell'accelerometro nei tre assi, otteniamo uno

stato $x(t)$ di 9 componenti, la cui dinamica è data da:

$$\dot{x}(t) = Ax(t) + \begin{bmatrix} -0.5v_g \\ v_b, g \\ v_b, a \end{bmatrix} \tag{3.17}$$

dove

$$A := \begin{bmatrix} -[y_g\times] & -0.5I & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

dove per un vettore p la matrice antisimmetrica $p\times$ è definita come

$$p\times := \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix}$$

mentre v_g è il rumore di misura del giroscopio e $v_{b,g}$ e $v_{b,a}$ sono dei rumori di processo fittizi aggiunti affinché la stima di questi due valori non si fermi una volta a regime, poiché il valore potrebbe cambiare successivamente.

Seguendo la teoria classica dell'EKF, si giunge alla forma di un'equazione di aggiornamento della misura, che, tuttavia, viene eseguita in due passi distinti, per l'accelerometro e per il sensore magnetico. Questo per avere la possibilità di utilizzare il sensore magnetico solo per individuare l'angolo di imbardata, dato che per quanto riguarda gli altri angoli di assetto il valore è alquanto rumoroso. Per quanto riguarda la matrice di osservazione dell'accelerometro, consideriamo come misura all'istante k non i dati puri che leggiamo, bensì il valore $z_{a,k}y_a - C(\hat{q})\tilde{g}$, dove $\tilde{g} := \begin{bmatrix} 0 & 0 & g \end{bmatrix}'$ e abbiamo che la matrice di osservabilità vale:

$$H_{a,k} := \begin{bmatrix} 2[C(\hat{q})\tilde{g}\times & 0 & I \end{bmatrix}$$

Mentre per quanto riguarda la misura del magnetometro abbiamo: $z_{m,k} := y_{m,k} - C(\hat{q})\tilde{m}$, con $\tilde{m} := \begin{bmatrix} cos(\alpha) & 0 & -sin(\alpha) \end{bmatrix}'$, dove α rappresenta l'angolo di dip del campo magnetico terrestre. La matrice di osservazione vale:

$$H_{m,k} := \begin{bmatrix} 2[C(\hat{q})\tilde{m}\times & 0 & I \end{bmatrix}$$

In questo caso, tuttavia, per evitare di considerare il magnetometro nel computo degli angoli di rollio e di beccheggio, si sostituisce la matrice classica dei guadagni di Kalman K con la matrice K' data da:

$$K' := \begin{bmatrix} r_3 r_3' & 0_{3,6} \\ 0_{6,3} & 0_{6,6} \end{bmatrix} K \tag{3.18}$$

con $r_3 = C(\hat{q}) \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$.

Si noti, inoltre, che l'algoritmo presentato in [19], presenta anche un valido sistema per stimare l'accelerazione esterna che viene applicata al sensore, in modo tale da eliminarne gli effetti al momento di utilizzare la misura dei dati dell'accelerometro.

L'algoritmo proposto presenta risultati notevoli, come verrà evidenziato nella sezione dedicata ai test, e vanta la notevole caratteristica di richiedere frequenze di aggiornamento molto basse, data la forte nonlinearità dell'algoritmo che permette di prevedere la dinamica dell'assetto con molta precisione. Questo permette di abbassare il carico di lavoro sulla CPU e di utilizzare senza problemi i dati forniti dal magnetometro con frequenza 8 Hz. Tuttavia i tempi di elaborazione si sono dimostrati ancora troppo elevati se implementati all'interno di un microcontrollore come quello di Arduino MEGA 2560.

Calibrazione Come ogni filtro di Kalman, il valore del rumore di processo non è noto a priori, per cui bisogna effettuare delle valutazioni. Il documento fornisce, tuttavia, una supposizione che p stata testata essere relativamente valida. Si tratta di considerare

$$E\{\begin{bmatrix} -0.5v_g(t) \\ v_{bg}(t) \\ v_{ba}(t) \end{bmatrix} \begin{bmatrix} -0.5v_g(s) \\ v_{b_g}(s) \\ v_{b_a}(s) \end{bmatrix}^T \} = Q\delta(t-s) \tag{3.19}$$

e

$$Q := Diag\{0.25R_g, Q_{b_g}, Q_{b_a}. \tag{3.20}$$

Tuttavia, una simulazione Montecarlo è stata effettuata per valutare l'effettiva validità di questa scelta, oltre che per valutare l'entità ottimale dei rumori di processo aggiunti. La simulazione è stata effettuata variando tre parametri, che chiameremo q_1, q_2 e q_3, che rappresentano il rumore di processo relativo alla stima del quaternione, del bias del giroscopio e di quello dell'accelerometro (considerando lo stesso valore per i tre assi). La simulazione si è svolta in tre passi:

- una prima simulazione è stata effettuata variando tutti i valori in un ampio spettro di possibilità
- un valore approssimativo per quanto riguarda q_3 viene fissato dalla prima simulazione e gli altri due valori vengono fatti variare casualmente attorno al punto di minimo individuato nella prima simulazione
- infine, viene fatto variare il parametro q_3 e si valuta l'accuratezza complessiva dell'algoritmo ottenuto

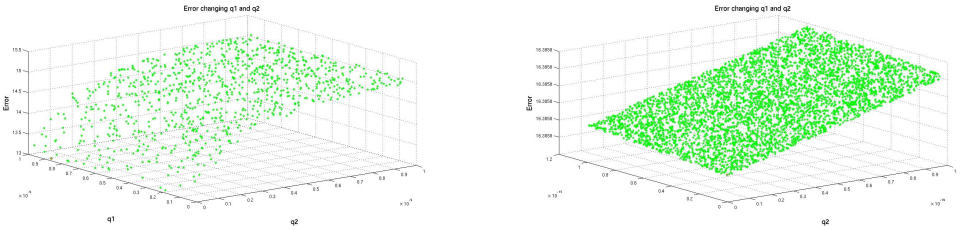
La bontà della calibrazione è stata valutata misurando un parametro proporzionale all'errore commesso dal filtro nella stima dell'assetto e dei bias su una storia di assetto casuale, ottenuta dal simulatore descritto nel capitolo precedente. Tale parametro è stato scelto come segue:

$$e = 5norm(rme_{assetto}) + norm(rme_{bias,acc}) + 20norm(rme_{bias,gyr}) \tag{3.21}$$

In modo tale da far prevalere l'errore sull'assetto, che chiaramente è il parametro che ci interessa, senza però trascurare la precisione sulla stima dei bias e il fatto che a causa della diversa natura delle grandezze descritte, i valori numerici sono differenti.

Il grafico della prima simulazione non può essere riportato, in quanto dovrebbe essere in quattro dimensioni. Per quanto riguarda la seconda simulazione, il risultato ottenuto è riportato in figura 3.1a. In seguito la simulazione è stata ripetuta circoscrivendo i valori intorno all'intervallo di minimo, come si vede in fig. 3.1b.

Si nota che la funzione raggiunge un minimo oltre il quale muta molto lentamente. Per questo motivo i valori che vengono scelti si fermano a 10^{-10} per evitare di inserire troppa rigidità. In fig. 3.2 vediamo, invece, il risultato della terza simulazione.



(a) Prima simulazione, più generale (b) Seconda simulazione, più accurata

Figura 3.1: Esito della seconda simulazione Montecarlo

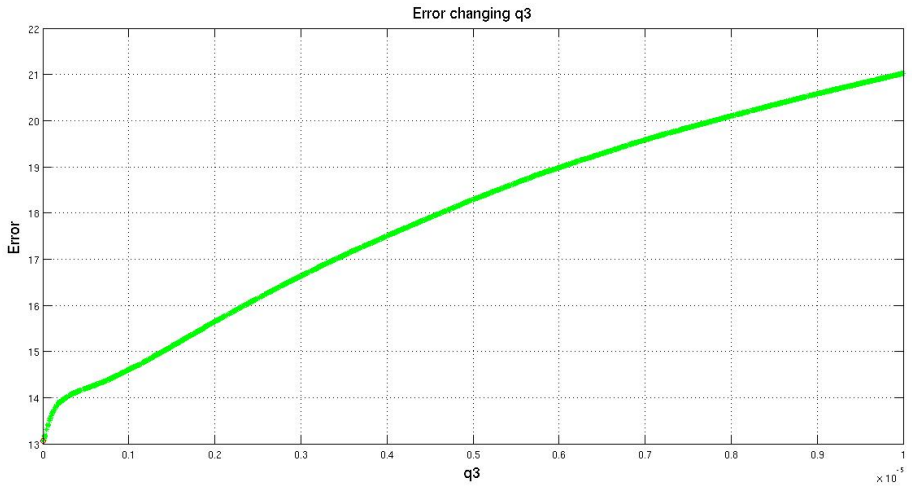


Figura 3.2: Esito della terza simulazione Montecarlo

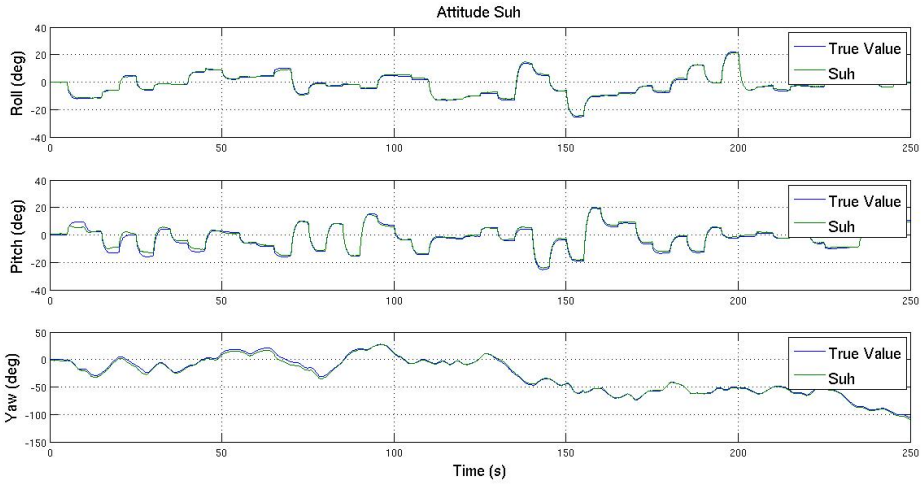


Figura 3.3: Assetto stimato con il filtro ottimizzato

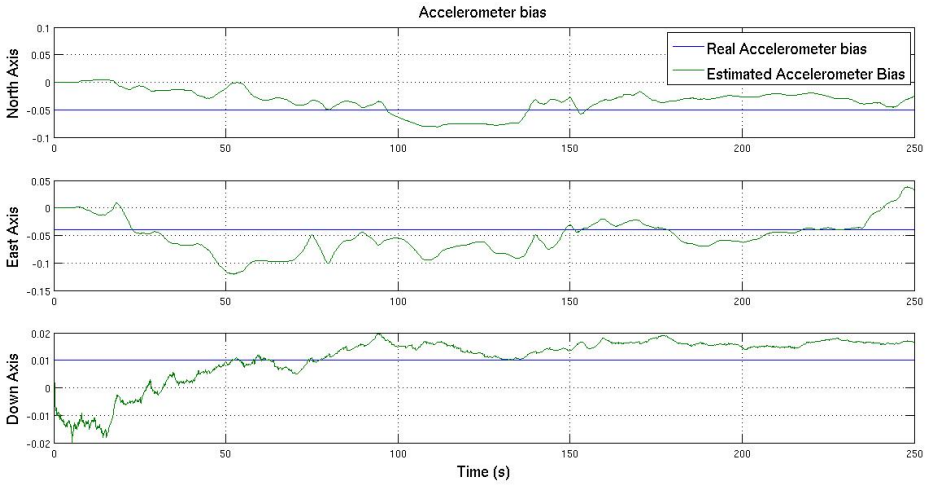


Figura 3.4: Bias dell'accelerometro stimati con il filtro ottimizzato

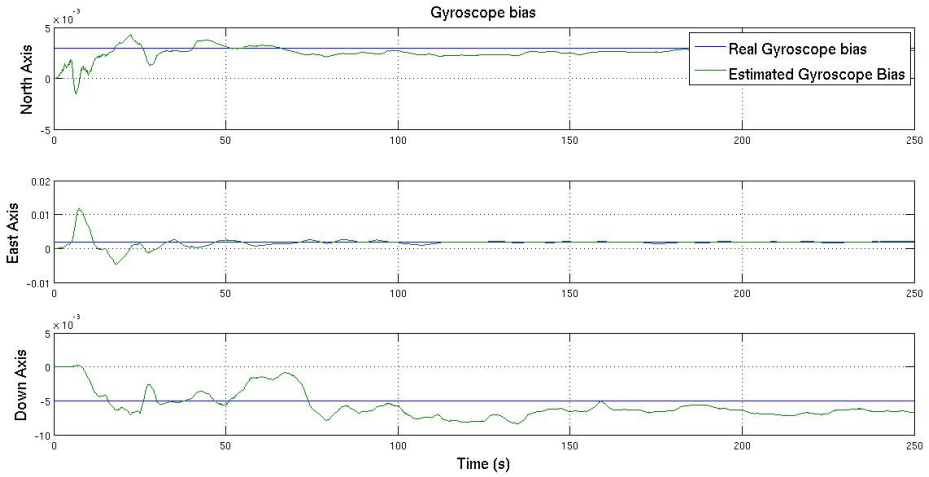


Figura 3.5: Bias del giroscopio stimati con il filtro ottimizzato

Si nota, allo stesso modo che oltre un certo limite il valore muta molto lentamente. È stato scelto, quindi, un valore di $7.6 * 10^{-9}$, di circa tre ordini di grandezza inferiore a quello suggerito. In fig. 3.3, 3.4 e 3.5 vediamo, infine, i risultati per quanto riguarda l'assetto, il bias dell'accelerometro e il bias del giroscopio. Si nota che, nel loro complesso, i valori non si discostano molto da quelli suggeriti nel documento di Suh e nel suo toolbox ufficiale, tuttavia in tabella 3.1 vediamo la differenza che si può ottenere:

Calibrazione	Assetto	Bias accelerometro	Bias giroscopio	Errore totale
Originale	0.7499,1.8439,2.6071	0.0111,0.0111,0.0111	0.0013,0.0013,0.0013	16.4640
Montecarlo	0.5711,1.1999,2.2045	0.0448,0.0448,0.0448	0.0032,0.0032,0.0032	13.0582

Tabella 3.1: Errori quadratici medi per l'assetto (deg) e per i bias (m/s^2, rad/s) secondo le due calibrazioni (assi x,y,z), più parametro di valutazione complessiva.

Algoritmo di Gastone Ferrarese L'algoritmo si basa sulle seguenti equazioni:

$$\dot{\mathbf{x}} = f(\mathbf{x}+\mathbf{w}) \tag{3.22}$$

$$\dot{\mathbf{z}} = h(\mathbf{x}+\mathbf{v}) \tag{3.23}$$

dove $\mathbf{Q} = E(\mathbf{ww})$ e $\mathbf{R} = E(\mathbf{vv})$ sono le matrici di covarianza dei rumori di processo e di osservazione.

$$\mathbf{F} = \frac{\delta f(\mathbf{x})}{\delta \mathbf{x}}|_{x=\hat{x}} \tag{3.24}$$

$$\mathbf{H} = \frac{\delta h(\mathbf{x})}{\delta \mathbf{x}}|_{x=\hat{x}} \tag{3.25}$$

Contrariamente all'algoritmo precedente, la matrice $\mathbf{Q}$ è considerata tempo-variabile ed è data dalla formula

$$\mathbf{Q}_k = \int_0^{T_s} \phi(t)\mathbf{Q}\phi^T(t)\,dt \tag{3.26}$$

dove

$$\phi(t) = e^{\mathbf{F}t} \simeq \mathbf{I} + \mathbf{F}t, \phi(\mathbf{T_s}) := \phi_k.$$

Discorso differente vale per la matrice **R** che è considerata costante, dato che la degradazione degli strumenti durante il tempo di missione è altamente improbabile. Si può ora valutare la matrice dei guadagni di Kalman come segue ed infine lo stato stimato, come somma della componente stimata a priori e dell'innovazione:

$$\mathbf{M}_k = \phi_k \mathbf{P}_k {\phi^{\mathbf{T}}}_k + \mathbf{Q}_k$$

$$\mathbf{K}_k = \mathbf{M}_k \mathbf{H}^{\mathbf{T}} (\mathbf{H}\mathbf{M}_k \mathbf{H}^{\mathbf{T}} + \mathbf{R}_k)^{-1}$$

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{K}_k (\mathbf{z}_k - h(\bar{\mathbf{x}}_k)) \tag{3.27}$$

dove

$$\bar{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + f(\hat{\mathbf{x}}_{k-1})\, T_s$$

Detto questo, non resta che individuare quale sarà lo stato (e la relativa matrice dinamica) e quali saranno le misure.

Stato Lo stato è composto dal quaternione di assetto e dai bias del giroscopio relativi ai tre angoli di eulero:

$$\mathbf{x} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 & b_\phi & b_\theta & b_\psi \end{bmatrix}^T. \tag{3.28}$$

Piccoli rumori di processo sono inseriti anche nei tre bias, per evitare che la stima si fermi subito. La funzione f può essere scritta come:

$$f(\mathbf{x}) = \frac{1}{2} \begin{bmatrix} \mathbf{q}_d \omega \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.29}$$

dove $\mathbf{q}_d$ è la matrice tale per cui:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}_d \omega \tag{3.30}$$

e che vale

$$\mathbf{q}_d = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ 3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}.$$

Si ottengono, quindi, le matrici **F** e **H**:

$$\mathbf{F} = \frac{1}{2}\begin{bmatrix} 0 & -(p-b_p) & -(q-b_q) & -(r-b_r) & q_1 & q_2 & q_3 \\ p-b_p & 0 & r-b_r & -(q-b_q) & -q_0 & q_3 & -q_2 \\ q-b_q & 0 & r-b_r & -(q-b_q) & -q_3 & -q_0 & q_1 \\ r-b_r & q-b_q & -(p-b_p) & 0 & q_2 & -q_1 & -q_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.31}$$

$$\mathbf{H} = \mathbf{I}_{4x4} \tag{3.32}$$

Misure La funzione di trasferimento lascia intendere che il quaternione d'assetto è la misura stessa dell'esperimento, anziché le misure di accelerazioneo di campo magnetico o altro. Questa è la novità principale di questo algoritmo e si basa sul fatto che al posto delle misure dei sensori viene inserito il risultato di un'analisi eseguita con il metodo delle triadi, noto in letteratura [20] e molto utilizzato in ambito spaziale sin dagli anni '60. Per approfondire il metodo si può consultare [18] e [4].

Abbiamo, quindi, una misura della matrice di rotazione tra il vettore gravità registrato e

$$\bar{\mathbf{g}} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$$

pesata con quella tra il campo magnetico registrato e il vettore (trascurando la declinazione)

$$\mathbf{H} = \begin{bmatrix} Hcos(I) & 0 & Hsin(I) \end{bmatrix}^T.$$

La qualità dell'algoritmo dipende ovviamente dalla qualità delle misure impiegate. Per quanto riguarda le accelerazioni, contrariamente a quanto accade nell'algoritmo descritto precedentemente, tutte vengono considerate trascurabili rispetto all'accelerazione di gravità, per cui gli errori di misura dell'accelerometro dipendono solo dai bias e dal rumore. Invece, per quanto riguarda il magnetometro, abbiamo:

$$\mathbf{B}_{sensed} = (\mathbf{I}_{3x3} + \mathbf{D})^{-1}(\mathbf{RH} + b + \varepsilon) \tag{3.33}$$

dove D sono le correzioni Soft Iron, b quelle Hard Iron e ε il rumore bianco. Per questo, il magnetometro richiede una calibrazione, come già spiegato nella sezione 3.1.1.

L'algoritmo richiede la calibrazione del metodo delle triadi e del filtro stesso, ma presenta risultati eccellenti, come vedremo nella sezione dedicata, e risparmia potenza computazionale, rispetto ad altre implementazioni di EKF, anche perché gli Jacobiani di f e h sono calcolati analiticamente.

Calibrazione Anche in questo caso l'algoritmo richiede una precisa calibrazione, che dipende dalla qualità delle misure effettuate e che deve essere effettuata in maniera empirica. Poiché l'algoritmo stima solo il bias dei giroscopi, lasciando la calibrazione del restante affidata solo alla calibrazione iniziale, i valori che sono stati scelti come costante di proporzionalità del rumore di processo sono due, che chiameremo q_1 e q_2. in fig. 3.6 osserviamo i risultati ottenuti:

mentre in fig. 3.7a e 3.7b vediamo l'andamento dell'assetto e del bias con i valori ottimali. I valori ottimali secondo la simulazione Montecarlo sono: $q_1 = 0.0005$ e $q_2 = 0.000004$.

3.1.5 Simulazione

Al fine di validare e confrontare gli algoritmi, sono state effettuate alcune simulazioni in ambiente Matlab/simulink. I dati derivano dall'esecuzione della simulazione del velivolo quadrirotore presentata nel capitolo precedente, per rendere la storia di assetto più verosimile. Ovviamente all'assetto vero sono stati aggiunti rumori e bias in maniera di poter simulare i sensori descritti nel primo capitolo o, perlomeno, dei sensori realistici. Poiché la fase di calibrazione era in grado di eliminare completamente il bias in ambiente di simulazione, ma

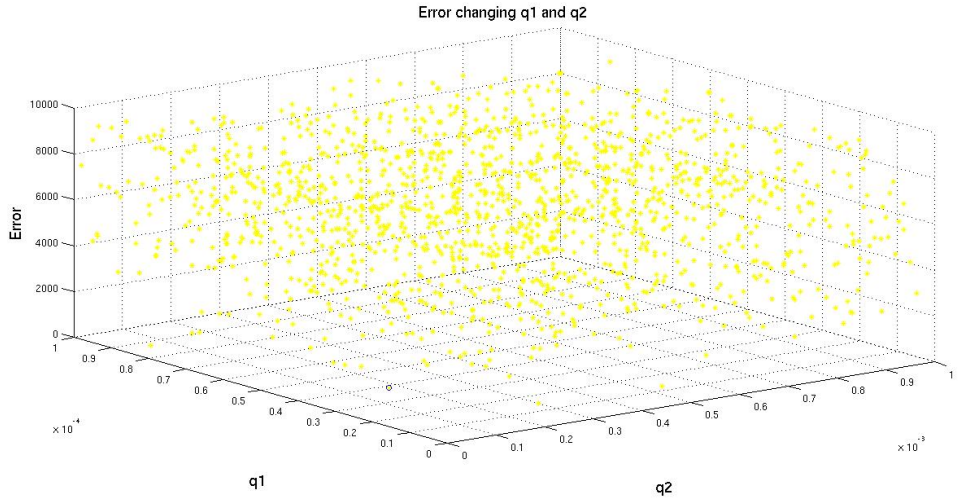
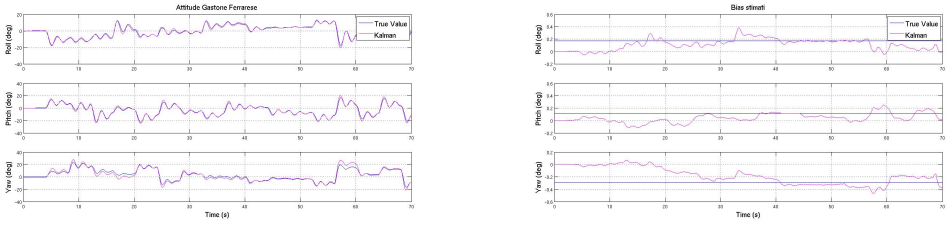


Figura 3.6: Errore in funzione dei parametri q_1 e 1_2



(a) Andamento dell'assetto

(b) Andamento dei bias del giroscopio

Figura 3.7: Esito del filtraggio ottimizzato con la Montecarlo

questo non è del tutto realistico, dei piccoli bias sono stati ulteriormente aggiunti. I valori della simulazione sono:

- $R_a = 0.004 * \mathbf{I}_{3x3}[m/s^2]$
- $R_g = 0.00001 * \mathbf{I}_{3x3}[rad]$
- $R_m = 100 * \mathbf{I}_{3x3}[\mu T]$
- $b_a = [0.0040.0050.001][m/s^2]$
- $b_g = [0.0030.0020.004][rad]$
- $b_m = [100 - 50150][\mu T]$

I risultati sono i seguenti:

Tuttavia, se si aumenta sensibilmente il bias residuo (moltiplicandolo per dieci), i risultati diventano:

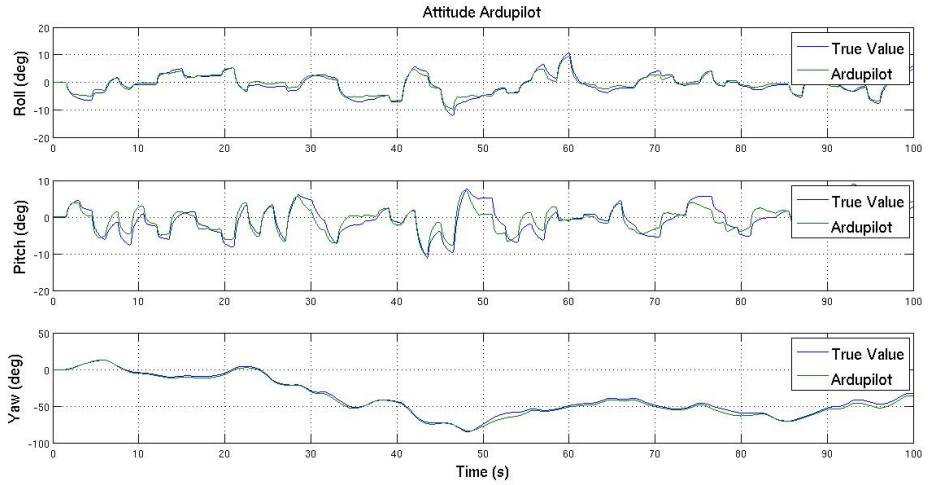


Figura 3.8: Andamento dell'assetto reale e stimato con il filtro di ArduPilot

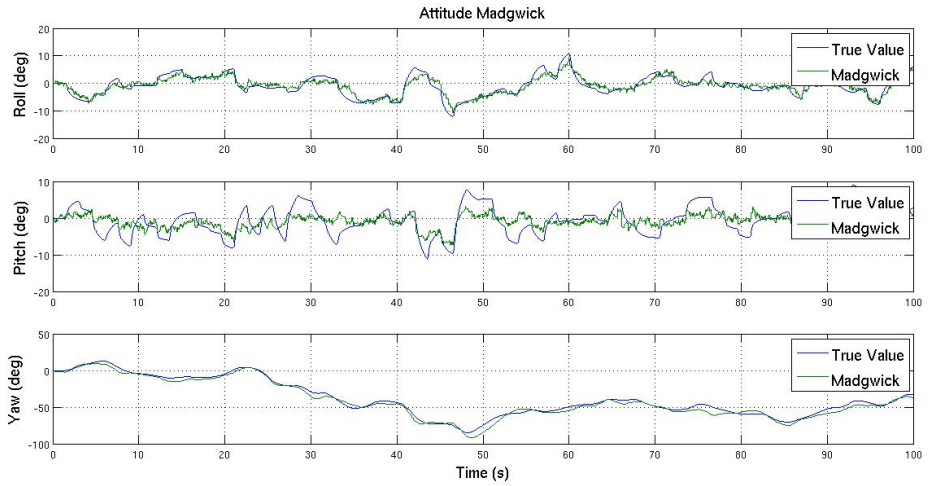


Figura 3.9: Andamento dell'assetto reale e stimato con il filtro di C. Madgwick

Tecnica	Beccheggio	Rollio	Imbardata
ArduPilot	0.8570	1.8986	2.5406
Madgwick	1.5518	2.8113	3.9059
Suh	0.8321	1.8159	3.4131
Ferrarese	0.9055	1.1578	2.3378

Tabella 3.2: Errori quadratici medi per l'assetto (deg).

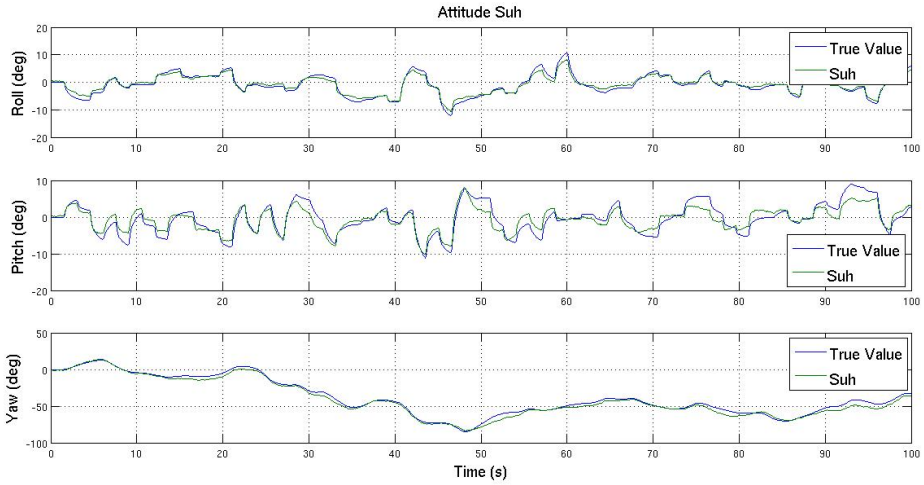


Figura 3.10: Andamento dell'assetto reale e stimato con il filtro di Y.S. Suh

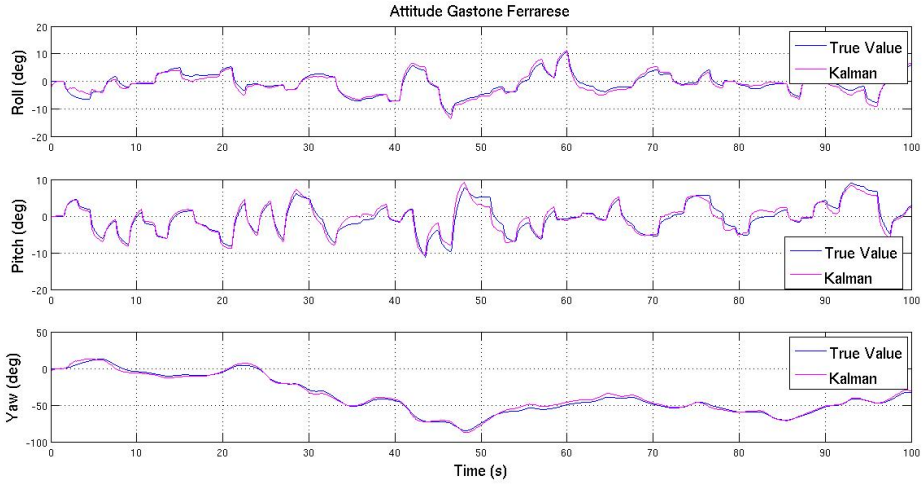


Figura 3.11: Andamento dell'assetto reale e stimato con il filtro di Gastone Ferrarese

Tecnica	Beccheggio	Rollio	Imbardata
ArduPilot	1.8432	2.3980	4.9294
Madgwick	1.6049	2.9099	3.1698
Suh	1.3565	2.1872	3.7612
Ferrarese	0.9383	1.1518	2.3971

Tabella 3.3: Errori quadratici medi per l'assetto (deg).

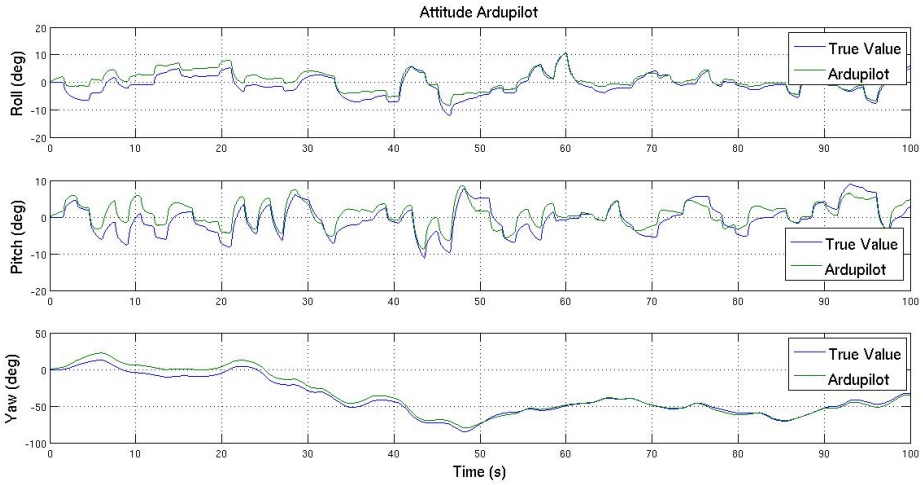


Figura 3.12: Andamento dell'assetto reale e stimato con il filtro di ArduPilot

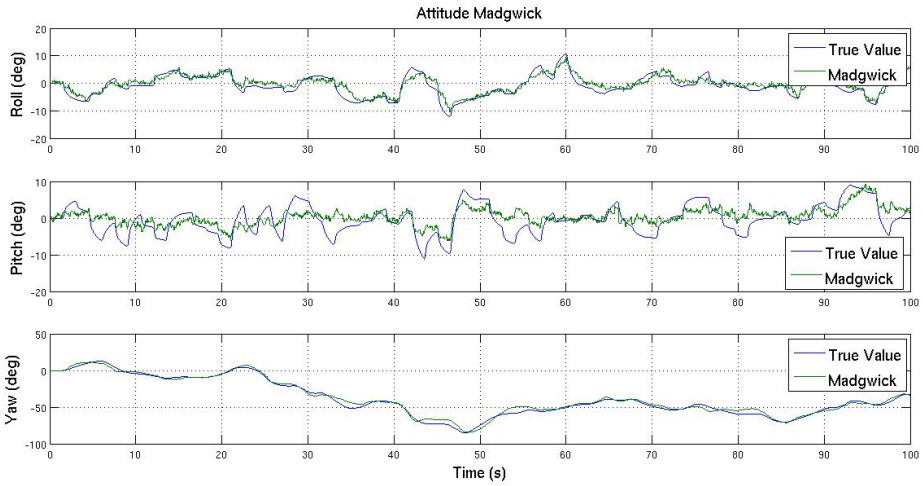


Figura 3.13: Andamento dell'assetto reale e stimato con il filtro di C. Madgwick

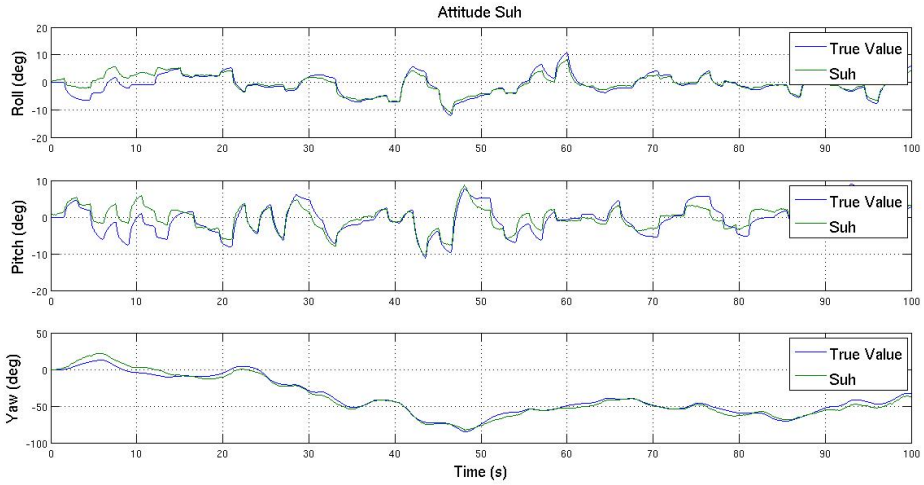


Figura 3.14: Andamento dell'assetto reale e stimato con il filtro di Y.S. Suh

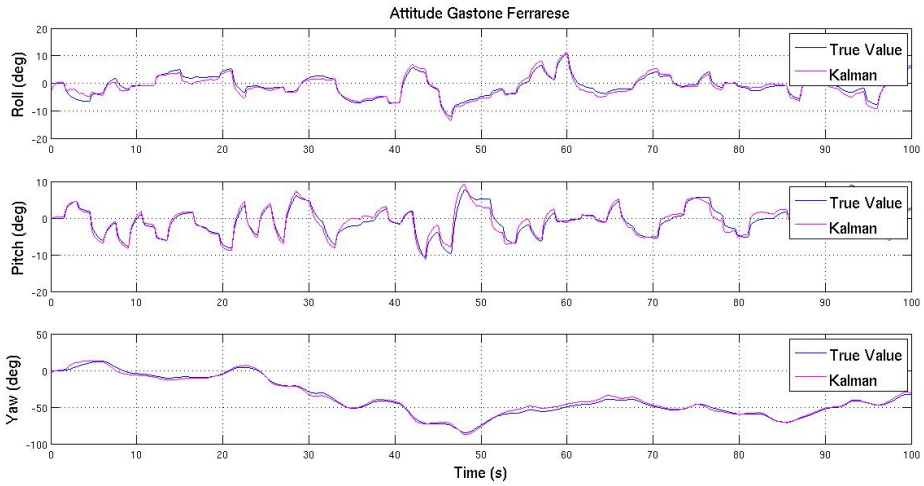


Figura 3.15: Andamento dell'assetto reale e stimato con il filtro di Gastone Ferrarese

Si nota immediatamente che l'algoritmo presente all'interno del codice ArduPilot è valido e preciso, anche se soffre nel caso il bias abbia una deriva ed aumenti di valore. Questo ne conferma la rischiosità nel caso di missioni troppo lunghe. Per quanto riguarda il filtro a gradiente coniugato, la simulazione non ha dato buoni risultati. Il parametro β (l'unico modificabile) è stato ottimizzato più e più volte, ma si è dimostrato molto legato ai parametri della simulazione e ne deriva che l'algoritmo è poco adattabile a situazioni differenti, anche di poco, da quella di progetto. Per quanto riguarda i filtri di Kalman, invece, entrambi hanno dimostrato la capacità di compensare molto bene la deriva del bias. Il filtro di Suh non ottiene risultati eccezionali in termini di precisione, però è in grado di annullare asintoticamente l'errore. Il filtro di Gastone Ferrarese, invece, si dimostra valido in ogni senso e si presenta come un'ottima alternativa ai filtri attualmente utilizzati.

3.1.6 Implementazione

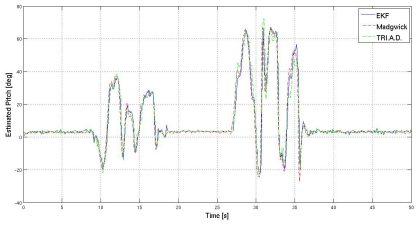
Nella pratica, però, l'implementazione di tali algoritmi può presentare non pochi problemi, primo fra tutti la conversione da linguaggio Matlab a linguaggio c++ su cui si basa l'ambiente di sviluppo Arduino e più o meno tutti gli hardware sul mercato. Prima di tutto, per valutare la pesantezza degli algoritmi a confronto, è stato misurato il tempo di esecuzione degli script Matlab adottati nella sezione precedente. Questo metodo presenta delle lacune legate al fatto che il codice potrebbe essere stato scritto in maniera più o meno ottimizzata, per cui si invita a non ritenere comprovati questi risultati, ma semplicemente indicativi sulla potenza computazionale richiesta. Il test è stato effettuato su un pc notebook Acer, con un processore intel i3 Sandy Bridge, sul quale gira il sistema operativo Debian 7.3 e MatlabR2013a. I dati forniti sono riferiti al tempo di esecuzione dell'algoritmo di stima relativo ad un ciclo di misura e sono dati dalla media di ogni esecuzione dell'algoritmo lungo un periodo di campionamento di 250 secondi a frequenza 20 Hz, eseguito con tre differenti storie di assetto. I risultati sono:

- Ardupilot $3.32465362091145 * 10^5 s$
- Madgwick $8.06306241965343 * 10^5 s$
- Suh $724895622895621 * 10^4$
- Ferrarese $4.95560314061385 * 10^5 s$

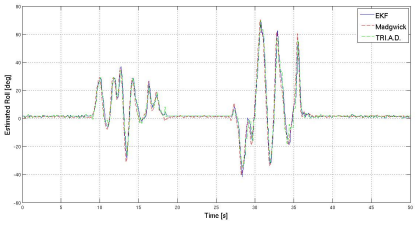
Se questo test, quindi, può non dire niente riguardo alla possibilità di implementare o meno l'algoritmo su una certa architettura, si nota, comunque, che il filtro proposto da Gastone Ferrarese presenta, rispetto agli altri filtri, notevoli vantaggi in termini computazionali, come previsto nella sezione 3.1.4.

Per completare il test, inoltre, si è provato a implementare alcuni algoritmi su una schda Arduino Mega 2560. L'ambiente Matlab/simulink fornisce un valido tool per convertire automaticamente degli schemi funzionanti in una funzione c++, sebbene ciò non fornisca risultati

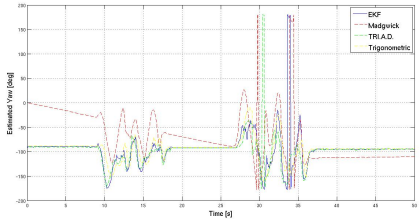
completamente ottimizzati in termine di codice. Tale metodologia è stata utilizzata per convertire lo schema relativo al filtro di Suh, altrimenti troppo complicato. Per quanto riguarda il filtro di Madgwick, la funzione è stato implementato in codice c, avvalendosi del toolbox ufficiale fornito dagli autori e adattandola all'hardware utilizzato. Per controllare la funzionalità di tali script è stato effettuato un breve test per mettere a confronto i risultati ottenuti dallo script di Matlab, del quale si conosce l'errore medio, e quelli ottenuti dallo stesso script funzionante sul microcontrollore Arduino. In particolare, il tempo richiesto dall'elaborazione dei dati su Arduino del filtro di Suh (circa 0.2 secondi per ogni iterazione), è stato ritenuto troppo elevato per fornire dei risultati accettabili, per cui il confronto presentato in questa trattazione riguarda solo il filtro a gradiente coniugato.



(a) Beccheggio

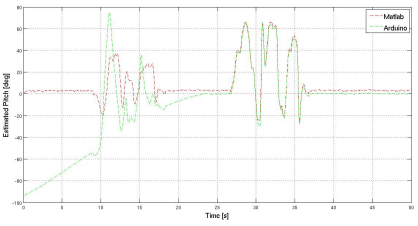


(b) Rollio

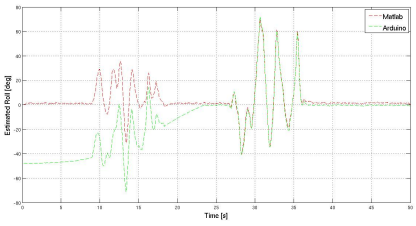


(c) Imbardata

Figura 3.16: Assetto stimato sui sensori reali



(a) Beccheggio



(b) Rollio

Figura 3.17: Assetto stimato sui sensori reali

in fig. 3.16 e si nota l'esattezza del filtro a gradiente coniugato, mentre in fig. 3.17 si nota la differenza che c'è tra lo stesso algoritmo eseguito a posteriori su matlab, con un processore intel i3 sandy bridge e quello eseguito in-loop sulla scheda Arduino Mega 2560, dotata di un processore ATmega2560 che opera a 16 MHz e 8 bit di precisione. Si nota che i valori

discostano l'uno dall'altro, tuttavia il problema è presente principalmente nella parte iniziale, in seguito alla quale i valori convergono. In pratica, per utilizzare questa tipologia di algoritmo, dobbiamo considerare un tempo di convergenza dell'algoritmo molto maggiore di quello ottenuto in simulazione e tenerne conto prima di avviare la missione.

3.2 Determinazione della posizione

Un aspetto fondamentale della capacità di navigazione autonoma è quello della determinazione della posizione del velivolo. A questo scopo i velivoli sono spesso equipaggiati con un sensore GPS il quale, tuttavia, soffre di grosse incertezze dovute alla scarsa qualità delle portanti per uso civile. Al fine di ottenere una misura precisa e aggiornata, viene di seguito presentato un filtro di Kalman che tenta di unire la tecnica cosiddetta di *Dead Reckoning* (la quale consiste nell'ottenere informazioni sulla posizione integrando la velocità, ottenuta a sua volta integrando le misure di accelerazione, disponibili tramite l'accelerometro) e le informazioni del GPS, in modo da combinarne la precisione della prima, con la fondamentale assenza di derive della seconda. Prima di questo però vediamo rapidamente come è strutturato l'algoritmo che identifica la posizione in ArduPilot.

Il concetto di partenza è semplice e in comune con l'algoritmo che verrà presentato in seguito. Si tratta di passare da un sistema di coordinate geodetiche a un sistema a Terra piatta con un'origine vicina (o coincidente) con il punto di partenza della missione. Ciò è possibile sfruttando le proprietà della terra, la cui superficie rappresenta quella che in analisi matematica si chiama varietà. Si prende, quindi, ad esempio, la prima posizione stabilita dal GPS e la si definisce come base. Qui avrà il suo centro un sistema di riferimento NED. Le informazioni del GPS, però sono in gradi, per questo vanno proiettate sulla superficie terrestre. Si ha:

$$d_m = r_e \frac{2\pi}{360}. \tag{3.34}$$

L'algoritmo di ArduPilot, a questo punto, si limita ad integrare le misure dell'accelerometro ruotate ne sistema di riferimento inerziale (Dead Reckonning) e a sommare un termine correttivo dovuto alla differenza tra la posizione stimta e quella misurata dal GPS moltiplicata per un coefficiente definito dall'operatore. Discorso differente vale per la posizione verticale, dove si usa la (1.1) linearizzata:

$$h_{baro} = ln(\frac{p_{ground}}{p}) T 29.271267 \tag{3.35}$$

dove T indica, stavolta, la temperatura assoluta (in gradi Kelvin). Infatti l'algoritmo non è in grado di fornire misure di posizione verticale in caso di assenza del barometro

3.2.1 Filtro di Kalman

Il filtro progettato è un filtro di Kalman che prende come ingressi i dati in uscita dell'accelerometro ruotati nel sistema di riferimento inerziale e i dati del GPS. Vedremo in seguito come sostituire la misura di pressione al dato del GPS relativo all'altitudine.

Al già noto rumore dovuto all'accelerometro, si aggiungerà quindi un rumore di calcolo dovuto alla non perfezione della matrice di rotazione stimata nella sezione precedente. La dinamica relativa a posizione, velocità e accelerazione nei tre assi è fortunatamente lineare, per cui si può utilizzare un filtro lineare senza temere di perdere accuratezza. L'unica variabile che potrebbe essere considerata nonlineare, ovvero bias degli strumenti, non fa parte dello stato del sistema, in quanto si considera già valutato nella sezione precedente. Seguendo la teoria classica di Kalman, abbiamo quindi che le matrici di stato, degli ingressi e di osservazione, valgono rispettivamente

$$A = \begin{bmatrix} \mathbf{I}_{3\times3} & \Delta t \cdot \mathbf{I}_{3\times3} \\ \odot_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.36}$$

dove $\odot$ rappresenta la matrice di soli zeri,

$$B = \begin{bmatrix} \odot_{3\times3} & \odot_{3\times3} \\ \odot_{3\times3} & \Delta t \cdot \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.37}$$

$$H = \begin{bmatrix} \mathbf{I}_{6\times6} \end{bmatrix} \tag{3.38}$$

se sono disponibili i nuovi dati del GPS (il quale ha frequenza di aggiornamento minore rispetto agli altri strumenti e al filtro stesso)

$$H = \begin{bmatrix} \odot_{6\times6} \end{bmatrix} \tag{3.39}$$

altrimenti. La matrice delle uscite è sempre nulla. Valori relativi all'inizializzazione e alle matrici di covarianza (Q) e (R) dipendono dalla strumentazione a disposizione. Di seguito viene mostrato un test effettuato in ambiente Matlab/Simulink con valori vicini a quelli degli strumenti a disposizione.

Per quanto riguarda l'asse verticale, sarebbe consigliabile utilizzare il barometro per migliorare

la stima, poiché è risaputo che il GPS soffre di molte incertezze nella misura dell'altitudine. Tuttavia, nel progetto del filtro, si è preferito tenere un contatto con la misura assoluta del GPS per quanto riguarda la posizione, dato che quella del barometro può dipendere da repentini cambi delle condizioni meteorologiche. La misura del barometro è, pertanto, impiegata solo come stima della velocità. Il fatto di utilizzare uno stato completo, consente di fondere tutte le misure a disposizione, senza precludere nessun dato, come invece accade nel software ArduPilot.

3.2.2 Calibrazione

Anche in questo caso, come per quanto riguardava i filtri di assetto, non si conosce l'entità fisica che si deve assegnare al rumore di processo con assoluta precisione, per cui è stata effettuata un'ulteriore simulazione Montecarlo. In questo caso, tuttavia, data la natura delle variabili utilizzate, la scelta dei parametri da utilizzare all'interno della simulazione si è rivelata più semplice, dovendo variare un solo parametro q. In fig. 3.18 viene rappresentato in un grafico l'andamento della norma degli scarti quadratici medi al variare del parametro q. Si nota che il risultato resta stabile per un vasto intervallo di valori del rumore di processo, il che é un fatto positivo perché significa che il valore scelto risulta ottimale anche nel caso di una qualità differente per i valori di misura e di ingresso, come dimostrato da numerosi test.

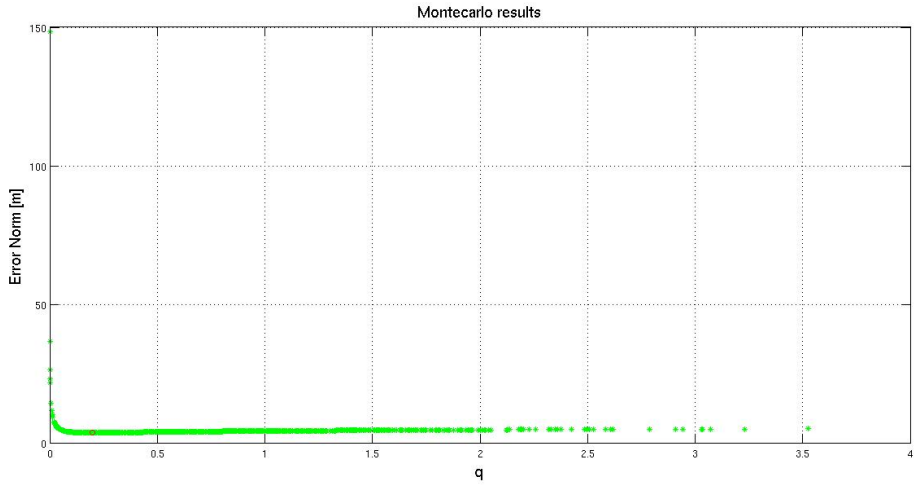


Figura 3.18: Andamento dell'errore in funzione del parametro q

3.2.3 Simulazione

La simulazione è stata effettuata simulando i sensori come nel caso del filtro di assetto e accedendo ad una storia di volo simulata. Il termometro è stato pensato affetto da un rumore di intensità 1 K, mentre il barometro è affetto da rumore bianco di intensità 20 Pa. Come ingresso del filtro di posizione è stato impiegata l'uscita del filtro di assetto di ArduPilot. Di seguito vengono riportati i risultati:

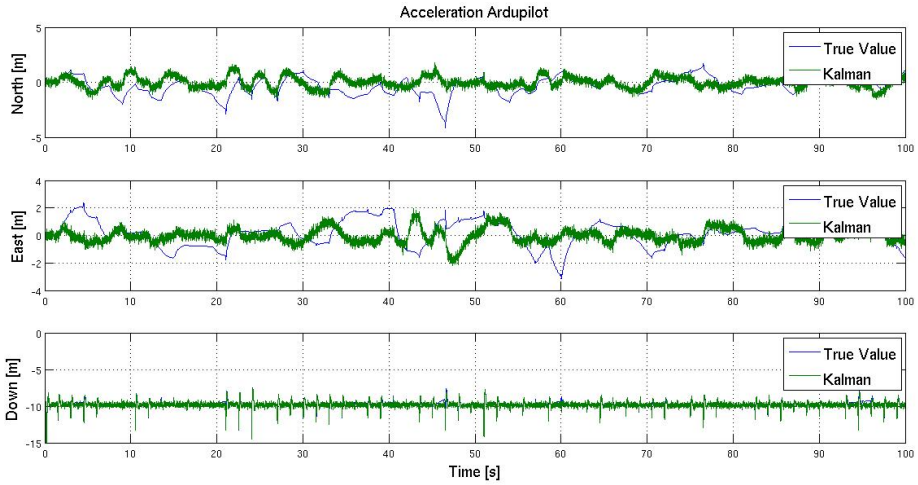


Figura 3.19: Andamento dell'accelerazione reale e stimata da ArduPilot

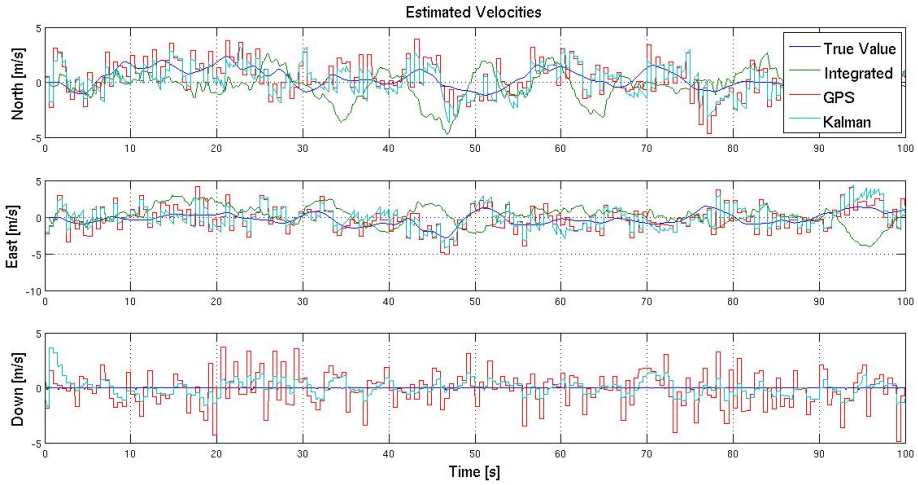


Figura 3.20: Andamento della velocità reale e stimata secondo ArduPilot, GPS e filtro di Kalman

Tecnica	Beccheggio	Rollio	Imbardata
GPS	3.1379	3.1562	3.1699
ArduPilot	3.0122	3.8790	3.364
Kalman	2.4844	2.5652	1.9746

Tabella 3.4: Errori quadratici medi per la posizione (espressi in metri).

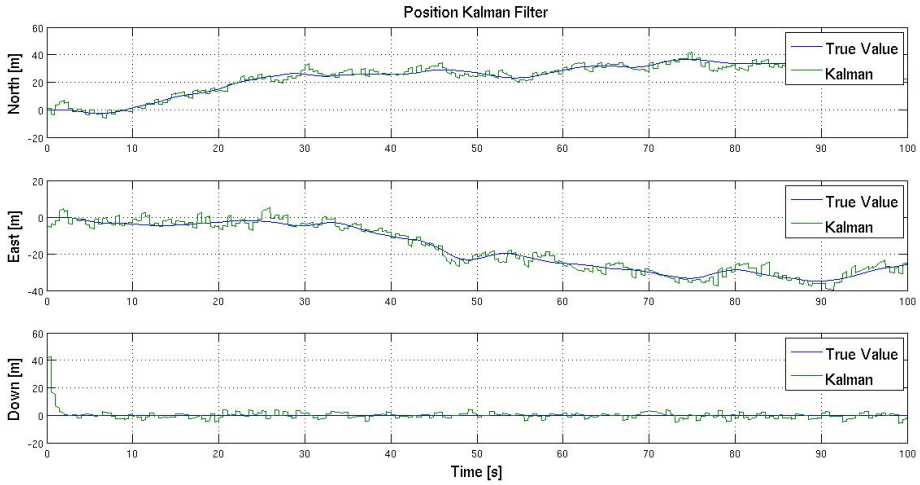


Figura 3.21: Andamento della posizione reale e stimata con il filtro di Kalman

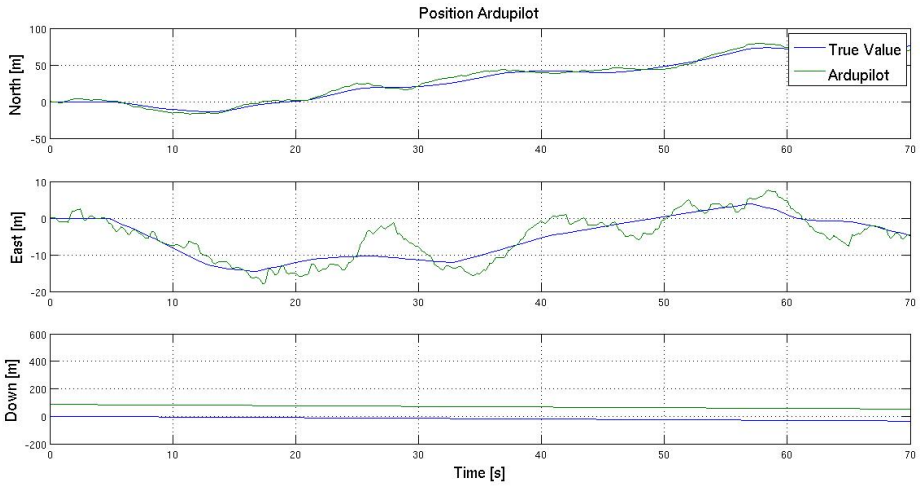


Figura 3.22: Andamento della posizione reale e stimata con il filtro di ArduPilot

Come si può notare, il risultato ottenuto è notevole. Il filtro di ArduPilot non fa molto più che garantire continuità tra una misurazione e l'altra del GPS, mentre riguardo all'asse verticale fornisce comunque una valida stima senza usare il GPS, però si riferisce a una pressione atmosferica a livello del mare che non sarà mai quella reale, per cui soffrirà sempre di un certo offset.

Infine, per quanto riguarda il filtro di Kalman proposto, i risultati sono molto validi e il sistema è, quindi, in grado di fornire una buona stima fondendo i dati di GPS accelerometro e barometro. Il sistema funziona anche in caso di black out temporaneo del GPS e non richiede troppa potenza computazionale, essendo concepito come filtro lineare.

3.2.4 Prestazioni

Anche in questo caso si è proceduto ad effettuare un testbench sui due algoritmi esaminati. Chiaramente, valgono le stesse considerazioni espresse in sezione 3.1.6, ma si è proceduto lo stesso per eseguire un confronto tra i due algoritmi. In questo caso i tempi di esecuzione per ogni step valgono:

- ArduPilot $9.72589630052865 * 10^6 s$
- Kalman $2.61901523939046 * 10^5 s$

In questo caso si può vedere che l'algoritmo proposto richiede un certo incremento di complessità, che però è accettabile se si considera il consistente incremento delle prestazioni dimostrato nella sezione precedente.

4 Conclusione e sviluppi futuri

Al termine del lavoro di ricerca si sono raggiunti tre risultati fondamentali:

1. costruito un simulatore in grado di replicare accuratamente il moto di un velivolo quadrirotore, e, con qualche modifica, di un multirotore in generale
2. effettuato una panoramica dei principali algoritmi presenti in letteratura e utilizzati al momento. Di questi alcuni si sono rivelati utili, altri si sono rivelati validi, ma non infallibili
3. descritto e testato due nuovi algoritmi con ottimi risultati: uno scritto da un dottorando del Laboratorio di Meccanica del Volo, e uno progettato per questa trattazione, ma entrambi nati all'interno del Laboratorio.

Per questi motivi ci si può ritenere soddisfatti ed orgogliosi di aver fatto un passo avanti verso la realizzazione di un sistema autonomo che porti al suo interno una certa innovazione sviluppata all'interno della facoltà. Ciò che manca per la realizzazione del progetto, e con questo veniamo agli sviluppi futuri, è la caratterizzazione accurata di un velivolo tra quelli disponibili: dei suoi momenti di inerzia, della dinamica dei suoi motori e delle sue caratteristiche aerodinamiche. Una volta ottenuto questo, il velivolo sarà programmabile con un firnware standard e avrà la possibilità di implementare dei filtri avanzati che gli consentano di migliorare notevolmente le performance riguardo alla precisione del volo, senza bisogno di modificare il sistema di controllo. A questo punto si dovrebbe avere un velivolo adeguato, o meglio una coppia almeno, per testare con una certa sensibilità una legge di guida che ottemperi le richieste di formazione compatta, nonché sia servibile per altri scopi di monitoraggio, sorveglianza, ispezione, fotografie aeree e tutte le missioni tipiche dei velivoli multirotore.

Appendice

Libreria GPGGA_GPVTG.

```
Header:

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class GPGGA_GPVTG
{
  public:
        GPGGA_GPVTG();                  // constructor for NMEA parser object
        int decode(char c);             // parse output from GPS; returns 1 when
          a GPGGA-GPVTG full combination is decoded
        float   gpvtg_course();         // track-angle-made-good term in last
          full GPRMC sentence
        float   gpvtg_mag_course();         // track-angle-made-good with respect
           to magnetic pole term in last full GPRMC sentence
        float   gpvtg_velocity();           // returns ground speed in kilometers
           per hour
        float   gpvtg_velocity_knots();         // returns ground speed in knots
          per hour
              float   gpgga_utc();                  // returns UTC as
                evaluated by GPS
              float   gpgga_latitude();             // returns latitude
              float   gpgga_longitude();            // returns longitude
              float   gpgga_altitude();             // returns longitude
              float   gpgga_satellites();           // returns number of
                satellites
              float   gpgga_hdop();                 // returns GPS HDOP (
                horizontal dilution of precision)
        int terms();                // returns number of terms (including data
          type and checksum) in last received full sentence
        char*   term(int t);                // returns term t of last received
          full sentence as zero terminated string
        float   term_decimal(int t);        // returns the base-10 converted
          value of term[t] in last full sentence received

  private:
           // properties
        float   _gpgga_latitude;
              float   _gpgga_longitude;
              float   _gpgga_altitude;
              float   _gpgga_satellites;
```

```
34                  float   _gpgga_hdop;
35          float   _gpvtg_velocity;
36          float   _gpvtg_velocity_knots;
37          float   _gpvtg_course;
38          float   _gpvtg_mag_course;
39                  float   _gpgga_utc;
40          char    f_sentence[100];
41          char*   f_term[30];
42          int f_terms;
43          int _terms;
44          char    _sentence[100];
45          char*   _term[30];
46          int n;
47          int _gpvtg_tag;
48                  int     _gpgga_tag;
49          int _state;
50          int _parity;
51          int _nt;
52          float   _degs;
53          int _dehex(char a);
54          float   _decimal(char* s);
55  };
```

Libreria:

```
1
2   /*
3     This library was written by Alberto Sodi and Paolo Lombardi and is
4     based on nmea.h library (Copyright (c) 2008 Maarten Lamers, The Netherlands,
5     All rights reserved)
6
7     This is meant to be useful in order to decode the sentences
8     produced by any GPS that communicates GPGGA and GPVTG sentences.
9     Is meant to be an estension to the library nmea.h which
10    only decodes GPRMC sentences (the most common, by the way)
11
12    This library is free software; you can redistribute it and/or
13    modify it under the terms of the GNU Lesser General Public
14    License as published by the Free Software Foundation.
15
16    This library is distributed in the hope that it will be useful,
17    but WITHOUT ANY WARRANTY; without even the implied warranty of
18    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19    Lesser General Public License for more details.
20
21    You should have received a copy of the GNU Lesser General Public
22    License along with this library; if not, write to the Free Software
23    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
24  */
25
26  #include "gpgga_gpvtg.h"
27
28  #define _GPVTG_TERM    "$GPVTG,"        // GPVTG datatype identifier
29  #define _GPGGA_TERM    "$GPGGA,"        // GPGGA datatype identifier
30
31  //
32  // constructor method
33  //
34
35  GPGGA_GPVTG::GPGGA_GPVTG()
```

```
{
    // private properties
    _gpvtg_velocity = 0;
    _gpvtg_velocity_knots = 0;
    _gpvtg_course = 0;
    _gpvtg_mag_course = 0;
    _gpgga_utc = 0;
    _gpgga_latitude = 0;
    _gpgga_longitude = 0;
    _gpgga_altitude = 0;
    _gpgga_satellites = 0;
    _gpgga_hdop = 0;
    _terms = 0;
    n = 0;
    _state = 0;
    _parity = 0;
    _nt = 0;
    f_sentence[0] = 0;
    f_terms = 0;

    // allocate memory for individual terms of sentence
    for (int t=0; t<30; t++) {
    _term[t] = (char*) malloc (15 * sizeof(char));
    f_term[t] = (char*) malloc (15 * sizeof(char));
    (f_term[t])[0] = 0;
    }
}


//
// public methods
//

int GPGGA_GPVTG::decode(char c) {
  // avoid runaway sentences (>99 chars or >29 terms) and terms (>14 chars)
  if ((n >= 100) || (_terms >= 30) || (_nt >= 15)) { _state = 0; }
  // LF and CR always reset parser
  if ((c == 0x0A) || (c == 0x0D)) { _state = 0; }
  // '$' always starts a new sentence
  if (c == '$') {
    _gpvtg_tag = 0;
    _gpgga_tag = 0;
    _parity = 0;
    _terms = 0;
    _nt = 0;
    _sentence[0] = c;
    n = 1;
    _state = 1;
    return 0;
  }
  // parse other chars according to parser state
  switch(_state) {
  case 0:
    // waiting for '$', do nothing
    break;
  case 1:
    // decode chars after '$' and before '*' found
    if (n < 7) {
     // see if first seven chars match "$GPVTG," or "$GPGGA,"
    if (c == _GPVTG_TERM[n]) { _gpvtg_tag++; }
```

```
96      if (c == _GPGGA_TERM[n]) { _gpgga_tag++; }
97      }
98      // add received char to sentence
99      _sentence[n++] = c;
100     switch (c) {
101     case ',':
102         // ',' delimits the individual terms
103       (_term[_terms++])[_nt] = 0;
104       _nt = 0;
105       _parity = _parity ^ c;
106       break;
107     case '*':
108         // '*' delimits term and precedes checksum term
109       (_term[_terms++])[_nt] = 0;
110       _nt = 0;
111       _state++;
112       break;
113     default:
114         // all other chars between '$' and '*' are part of a term
115       (_term[_terms])[_nt++] = c;
116       _parity = _parity ^ c;
117       break;
118     }
119     break;
120   case 2:
121     // first char following '*' is checksum MSB
122     _sentence[n++] = c;
123     (_term[_terms])[_nt++] = c;
124     _parity = _parity - (16 * _dehex(c));       // replace with bitshift?
125     _state++;
126     break;
127   case 3:
128      // second char after '*' completes the checksum (LSB)
129     _sentence[n++] = c;
130     _sentence[n++] = 0;
131     (_term[_terms])[_nt++] = c;
132     (_term[_terms++])[_nt] = 0;
133     _state = 0;
134     _parity = _parity - _dehex(c);
135      // when parity is zero, checksum was correct!
136     if (_parity == 0) {
137
138          // did we find a gpgga sentence?
139     if (_gpgga_tag == 6) {
140          // copy _sentence[] to f_sentence[]
141     while ((--n) >= 0) { f_sentence[n] = _sentence[n]; }
142      // copy all _terms[] to f_terms[]
143       for (f_terms=0; f_terms<_terms; f_terms++) {
144     _nt = 0;
145     while ((_term[f_terms])[_nt]) {
146         (f_term[f_terms])[_nt] = (_term[f_terms])[_nt];
147         _nt++;
148     }
149         (f_term[f_terms])[_nt] = 0;
150     }
151
152        // store values of relevant terms
153       _gpgga_utc=_decimal(_term[1]);
154       _gpgga_latitude=_decimal(_term[2]);
155       _gpgga_longitude=_decimal(_term[4]);
```

```
156         _gpgga_altitude=_decimal(_term[9]);
157         _gpgga_satellites=_decimal(_term[7]);
158         _gpgga_hdop=_decimal(_term[8]);
159          // Sentence accepted, but still not complete
160         return 0;
161       }
162
163        // did we find a gpvtg sentence?
164       if (_gpvtg_tag == 6) {
165        // copy _sentence[] to f_sentence[]
166       while ((--n) >= 0) { f_sentence[n] = _sentence[n]; }
167        // copy all _terms[] to f_terms[]
168        for (f_terms=0; f_terms<_terms; f_terms++) {
169           _nt = 0;
170           while ((_term[f_terms])[_nt]) {
171           (f_term[f_terms])[_nt] = (_term[f_terms])[_nt];
172           _nt++;
173       }
174           (f_term[f_terms])[_nt] = 0;
175         }
176        // store values of relevant terms
177          _gpvtg_course = _decimal(_term[1]);
178          _gpvtg_mag_course=_decimal(_term[3]);
179          _gpvtg_velocity=_decimal(_term[7]);
180          _gpvtg_velocity_knots=_decimal(_term[5]);
181       // sentence accepted and complete
182          return 1;
183          }
184       }
185       break;
186       default:
187       _state = 0;
188       break;
189     }
190     return 0;
191   }
192
193   float GPGGA_GPVTG::gpvtg_course() {
194       // returns decimal value of track-angle-made-good term in last-known GPVTG
          sentence
195     return _gpvtg_course;
196   }
197
198   float GPGGA_GPVTG::gpvtg_mag_course() {
199       // returns decimal value of track-angle-made-good with respect to magnetic
          pole term in last-known GPVTG sentence
200     return _gpvtg_mag_course;
201   }
202
203   float GPGGA_GPVTG::gpvtg_velocity() {
204           // returns decimal value of ground speed in last-known GPVTG sentence [km
            /s]
205     return _gpvtg_velocity;
206   }
207
208   float GPGGA_GPVTG::gpvtg_velocity_knots() {
209       // returns decimal value of ground speed in last-known GPVTG sentence [kn]
210     return _gpvtg_velocity_knots;
211   }
212
```

```
213 float GPGGA_GPVTG::gpgga_utc() {
214         // returns decimal value of UTC as measured from GPS
215   return _gpgga_utc;
216 }
217
218 float GPGGA_GPVTG::gpgga_latitude() {
219         // returns decimal value of latitude
220   return _gpgga_latitude;
221 }
222
223 float GPGGA_GPVTG::gpgga_longitude() {
224         // returns decimal value of longitude
225   return _gpgga_longitude;
226 }
227
228 float GPGGA_GPVTG::gpgga_altitude() {
229         // returns decimal value of altitude
230   return _gpgga_altitude;
231 }
232
233 float GPGGA_GPVTG::gpgga_satellites() {
234         // returns number of received satellites
235   return _gpgga_satellites;
236 }
237
238 float GPGGA_GPVTG::gpgga_hdop() {
239         // returns decimal value of HDOP (horizontal dilution of precision)
240   return _gpgga_hdop;
241 }
242
243 int GPGGA_GPVTG::terms() {
244     // returns number of terms (including data type and checksum) in last
        received full sentence
245   return f_terms;
246 }
247
248 char* GPGGA_GPVTG::term(int t) {
249     // returns term t of last received full sentence as zero terminated string
250   return f_term[t];
251 }
252
253 float GPGGA_GPVTG::term_decimal(int t) {
254     // returns value of decimally coded term t
255   return _decimal(f_term[t]);
256 }
257
258 int GPGGA_GPVTG::_dehex(char a) {
259     // returns base-16 value of chars '0'-'9' and 'A'-'F';
260     // does not trap invalid chars!
261   if (int(a) >= 65) {
262     return int(a)-55;
263   }
264   else {
265     return int(a)-48;
266   }
267 }
268
269 float GPGGA_GPVTG::_decimal(char* s) {
270     // returns base-10 value of zero-termindated string
271     // that contains only chars '+','-','0'-'9','.';
```

```
272       // does not trap invalid strings!
273     long  rl = 0;
274     float rr = 0.0;
275     float rb = 0.1;
276     boolean dec = false;
277     int i = 0;
278
279     if ((s[i] == '-') || (s[i] == '+')) { i++; }
280     while (s[i] != 0) {
281       if (s[i] == '.') {
282         dec = true;
283       }
284       else{
285         if (!dec) {
286           rl = (10 * rl) + (s[i] - 48);
287         }
288         else {
289           rr += rb * (float)(s[i] - 48);
290           rb /= 10.0;
291         }
292       }
293       i++;
294     }
295     rr += (float)rl;
296     if (s[0] == '-') {
297       rr = 0.0 - rr;
298     }
299     return rr;
300   }
```

Bibliografia

[1] Saif A. Al-Hiddabi. Quadrotor control using feedback linearization with dynamic extension. In *6th International Symposium on Mechatronics and its Applications (ISMA09), Sharjah, UAE*.

[2] Kostas Alexis, George Nikolakopoulosb, and Anthony Tzesa. Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances. *Control Engineering Practice*, 2011.

[3] C. Anderson. ArduPilot Project. http://ardupilot.com/. [Online; accessed 25-Feb-2014].

[4] H.D. BLACK. A passive system for determining the attitude of a satellite. *AIAA journal*, 1964.

[5] Hossein Bolandi, Mohammad Rezaei, Reza Mohsenipour, Hossein Nemati, and Seed Majid Smailzadeh. Attitude control of a quadrotor with optimized pid controller. *Scirp Intelligent Control and Automation*, 2013.

[6] Samir Bouabdallah, André Noth, and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. *International Conference on Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ*, 2004.

[7] M.J. Caruso. Applications of magnetic senors for low cost compass systems. In *Location and Navigation Symposium, IEEE*.

[8] Carlo Casarosa. *Meccanica del volo*. plus edizioni, 2000.

[9] J. Colorado, A. Barrientos, A. Martinez, B. Lafaverges, and J. Valente. Mini-quadrotor attitude control based on hybrid backstepping and frenet-serret theory. *IEEE International Conference on Robotics and Automation*, 2010.

[10] Analog Devices. ADIS16350 Specifications. http://www.analog.com/en/mems-sensors/mems-inertial-sensors/adis16350/products/product.html, 2013. [Online; accessed 16-Aug-2013].

[11] T. Hamel, R. Mahony, and R. Lozano. Dynamic modelling and configuration stabilization for an x4-flyer. In *15th Triennial World Congress, Barcelona, Spain*.

[12] W.T. HIGGINS JR. A comparison of complemetary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, 1975.

[13] M. Lamers. nmea.h library. http://www.maartenlamers.com/nmea/. [Online; accessed 25-Feb-2014].

[14] S.O.H. Madgwick, A.J.L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *IEEE International Conference on Rehabilitation Robotics*.

[15] ohn L. Crassidis, Kok-Lam Lai, and Richard R. Harman. Real-time attitude-independent three axis magnetometer calibration. *AIAA Journal of Guidance, Control and Dynamics*.

[16] W. Premerlani. Magnetometer offset cancellation: Theory and implementation, revisited. 2011. [Online; accessed 25-Feb-2014].

[17] A.M. Sabatini. Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Transactions on Biomedical Engineering*, 2006.

[18] M. D. Shuster and S. D. Oh. Three-axis attitude determination from vector observations. *AIAA Journal of Guidance and Control*.

[19] Y.S. Suh. Orientation estimation using a quaternion-based indirect kalman filter with adaptive estimation of external acceleration. *IEEE Transactions on Instrumentation and Measurement*, 2010.

[20] J. R. Wertz. *Spacecraft Attitude Determination and Control*. Springer, 1978.

Ringraziamenti

Un breve ma sentito ringraziamento a tutte le persone che hanno reso possibile questo lavoro, la mia laurea e il mio conseguente lancio nel mondo del lavoro.

Al Professor Fabrizio Giulietti, che mi segue e conosce da anni, che ha sempre avuto fiducia in me e che mi ha sempre messo a disposizione strutture e conoscenze per completare questo lavoro e la mia formazione.

A tutto il Laboratorio di Meccanica del Volo, Gastone, Matteo e Mauro: persone eccezionali, preparate e disponibili che mi hanno seguito, aiutato e guidato durante tutta la mia permanenza nel Laboratorio.

Ai miei compagni che hanno condiviso con me viaggi, lezioni, pause, serate e momenti belli e brutti. Grazie Ale, Buccio, Campa, Fede, Pado e Pietro (in ordine rigorosamente alfabetico). Anche se finiremo in ogni angolo del mondo, sarà difficile dimenticarvi.

A Monica e Giorgia che hanno dato un volto umano alla routine quotidiana della mensa.

Al team A5-Unibo, che mi ha fatto sperimentare la vita del ricercatore, che mi ha fatto vivere un'esperienza stupenda e con i quali ho condiviso tanti sacrifici, dalle notti passate in stazioni e aeroporti, agli straordinari per consegnare tutti i documenti nonostante stessi lavorando anche sulla tesi.

E infine due righe a parte le merita la mia sorella spagnola, Nani, che ha lavorato fianco a fianco con me quasi ogni giorno in questi due anni e che è sicuramente quanto più mi mancherà di questo luogo. Perché mi ha fatto capire cosa vuol dire avere passione per quello che stai facendo, e perché insieme abbiamo imparato che non si lavora mai con degli ingegneri, degli operai o dei professori, ma che si lavora con delle persone. Grazie, di cuore.

Forlì, 27 Marzo 2014 A. S.