Progetto di Microelettronica
Sviluppo di algoritmi per la stima dell'orientamento di un sensore inerziale

Daniele Comotti - Michele Ermidoro

29 marzo 2011

Indice

Introduzione

Questa documento è la relazione di quanto da noi svolto durante il corso di progetto di *Microelettronica*. L'obiettivo del progetto è quello di ottenere un livello sufficiente di confidenza e conoscenza con il contesto applicativo legato alla stima di rotazioni attraverso l'uso di una piattaforma di misura inerziale, più comunemente nota come *IMU* (*inertial measurement unit*). Le piattaforme di questo tipo integrano al loro interno svariati sensori, le cui misure permettono di risalire a come il dispositivo è orientato nello spazio. Il progetto, durante i 3 mesi di svolgimento, è stato soggetto a numerose variazioni, soprattutto legate alla iniziale difficile comprensione del contesto applicativo e delle tecniche di stima e di integrazione delle misure. Durante l'ultimo mese di lavoro, abbiamo intensificato l'attività, e, anche grazie agli innumerevoli articoli da noi letti, siamo riusciti a produrre degli algoritmi stabili e consistenti.

La relazione si articola come segue.

Nel primo capitolo verrà esposta una panoramica generale circa il contesto applicativo che ha riguardato l'attività, per quel che concerne gli strumenti di misura disponibili e la *IMU* da noi utilizzata e le tecniche matematiche di cui viene fatto maggiormente uso.

Il secondo capitolo vede come oggetto la realizzazione di un primo algoritmo di stima dell'orientamento, basato sugli angoli di *Eulero*.

A seguire, nel terzo capito, verrà esposta una nostra implementazione del fitro di *Kalman*, l'ultimo lavoro che abbiamo realizzato.

Il quarto capitolo interessa l'algoritmo descritto da uno degli articoli che maggiormente ci hanno ispirato [13], che ci ha aiutato parecchio durante lo sviluppo degli altri due algoritmi.

Infine, gli ultimi due capitoli, riguardano rispettivamente l'applicativo realizzato in linguaggio *C#* che, attraverso una simpatica interfaccia grafica, mostra all'utente in tempo reale con l'*IMU* collegato al PC il risultato degli algoritmo, ed i risultati e le prestazioni degli algoritmi discussi.

Capitolo 1

Analisi del Contesto

In questo capitolo verranno esposti tutti gli elementi del contesto applicativo relativo al nostro progetto che abbiamo preso in considerazione e che hanno determinato le scelte implementative.

1.1 Strumenti di Misura in Commercio

Dalla letteratura che abbiamo esaminato [1, 2, 3, 4, 5, 13] sono risultati di frequente utilizzo i seguenti tipi di strumenti di misura:

- Accelerometro - Permette di misurare l'accelerazione agente su di un corpo rilevando l'inerzia di una massa sospesa ad un elemento elastico. Lo spostamento rispetto ad un'altra struttura fissa del dispositivo è valutato per mezzo di un sensore che determina l'accelerazione associata alla massa sulla base dello spazio percorso dalla stessa e che trasforma tale spostamento in un segnale elettrico. In condizioni statiche le accelerazioni misurate si limitano al vettore gravità, mentre in condizioni dinamiche, oltre a queste componenti, si hanno le accelerazioni esterne a cui il corpo è soggetto. Nel nostro caso gli accelerometri analizzati sono caratterizzati da 3 assi di misura x, y, z; per mezzo della trigonometria è possibile risalire, a meno di casi particolari, all'inclinazione di ciascun asse rispetto alla terna originale, come esposto nella sotto-sezione 1.2.2.
  I principali tipi di accelerometro sono:
  - Accelerometro estensimetrico;
  - Accelerometro piezoresistivo;
  - Accelerometro LVDT;
  - Accelerometro capacitivo;
  - Accelerometro piezoelettrico;
  - Accelerometro laser;

  - Gravitometro;

- Magnetometro - Fornisce informazioni circa il campo magnetico che circonda il dispositivo. I magnetometri si suddividono in scalari o vettoriali, rispettivamente se misurano il modulo del campo magnetico oppure la componente del campo magnetico lungo una particolare direzione dello spazio. Il vettore del campo magnetico può essere definito unicamente attraverso la misura su tre direzioni indipendenti. Data la numerosità degli apparecchi elettromagnetici che ci circondano, la misura attraverso questo strumento può essere affetta da errore. L'utilizzo della trigonometria, come mostrato nella sotto-sezione 1.2.3, risulta utile al fine della determinazione dell'orientamento a partire dalla misura del vettore magnetico lungo 3 assi (x, y, z). I principali tipi di magnetometro sono:

  - Magnetometro ad ago;
  - Magnetometro a bobina;
  - Sonda di Hall;
  - Magnetometro a protoni;

- Giroscopio - Questo dispositivo fisico rotante, per effetto della legge di conservazione del momento angolare, tende a mantenere il suo asse di rotazione orientato in una direzione fissa. In questo modo può essere utilizzato per misurare le velocità angolari $[\frac{rad}{sec}]$. Un giroscopio triassiale permette di misurare le velocità di rotazione attorno ai tre assi x, y, z e di ricavare l'angolo di rotazione semplicemente integrando nel tempo la misura di velocità. Ciascun dispositivo è però affetto da un errore che, se integrato, fa divergere linearmente l'angolo. Questo tipo di errore, chiamato *drift*, se non trattato, porta alla completa inconsistenza delle misure;

- GPS - Il segnale GPS (*Global Positioning System*) di posizione e velocità viene spesso accostato alle misure dell'accelerometro per ottenere delle informazioni di maggiore consistenza e per aggiornare in maniera iterativa l'offset che caratterizza un giroscopio.

Dato che l'unità di misura inerziale (IMU) a nostra disposizione non include un dispositivo GPS (fare riferimento alla sezione 1.2) e dato che non ne abbiamo a disposizione uno, l'utilizzo di questo tipo di misura non è stata considerato ai fini dell'attività di progetto.

1.2 Inertial Measurement Unit (IMU): iNemo

Per lo sviluppo di questo progetto ci siamo avvalsi della piattaforma di misura inerziale (o *IMU*) denominata *iNemo* (*iNErtial MOdule*) e prodotta dalla *ST Microelectronics*. In particolare, la distribuzione che abbiamo utilizzato è l'*STEVAL-MKI062V2*, evaluation kit e seconda versione dell'*iNemo*.

L'*iNemo* è basato su *MEMS* (Micro Electro-Mechanical Systems) ed è composto dai seguenti moduli [4, 5]:

- *LPS001DL*, sensore di pressione a *MEMS*;
- *LPR430AL*, giroscopio basato su *MEMS* che fornisce informazioni di *Roll* e *Pitch* (fare riferimento alla sotto-sezione 1.3.1 per dettagli). Questo dispositivo misura la velocità angolare attorno agli assi di *Pitch* e *Roll* con una scala pari a $\pm 300\,\frac{deg}{sec}$ entro una banda passante è di $140\,Hz$. Si basa su una massa controllata mantenuta in costante oscillazione in grado di reagire a velocità angolari sfruttando il principio di Coriolis [9];
- *LY330ALH*, giroscopio basato su *MEMS* che misura le rotazioni di *Yaw* (fare riferimento alla sotto-sezione 1.3.1 per dettagli). Questo giroscopio a singolo asse è caratterizzato da una scala pari a $300\,\frac{deg}{sec}$ ed una banda di $140\,Hz$. Il principio di funzionamento è lo stesso dei giroscopi integrati nell'*LPR430AL* [10];
- *LSM303DLH*, modulo basato su *MEMS* che integra un accelerometro ed un magnetometro di misura, entrambi relativi a 3 assi (x, y, z). Questo sistema integrato presenta un sensore 3D digitale che fornisce misure accelerometriche lineari ed un sensore 3D magnetico. Le scale di misura sono selezionabili dall'utente e sono rispettivamente $\pm 2\,g$ / $\pm\,4g$ / $\pm\,8g$ per l'accelerometro e ± 1.3 / $\pm\,1.9$ / $\pm\,2.5$ / $\pm\,4.0$ / $\pm\,4.7$ / $\pm\,5.6$ / $\pm\,8.1$ *Gauss* per il magnetometro. Nella nostra applicazione abbiamo fatto uso delle prime scale menzionate per entrambi gli strumenti [8];
- *STM32F103RET7*, microcontrollore a 32 bit per la supervisione del modulo.

In figura 1.1 è rappresentata la piattaforma iNemo.

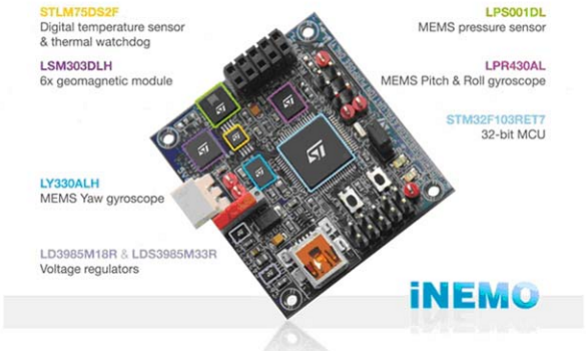


Figura 1.1: Fotografia dell'iNemo dall'alto.

1.2.1 Rotazioni e giroscopio

I giroscopi di cui è dotato l'iNemo permettono di ottenere informazioni relative alla velocità di rotazione attorno ai 3 assi x, y, z. Per ricavare le rotazioni angolari è necessario effettuare una integrazione discreta nel tempo:

$$\Delta \overrightarrow{\theta} = \vec{\omega} \cdot \Delta t$$

dove $\overrightarrow{\theta}$ rappresenta il vettore degli angoli di rotazione attorno agli assi x, y, z della terna corrente, $\vec{\omega}$ è il vettore delle velocità angolari riferite a ciascun asse della terna corrente e Δt la variazione temporale dalla precedente acquisizione.
Tale operazione, effettuata direttamente senza alcun accorgimento, porta, al decorrere del tempo, a delle informazioni inconsistenti; infatti i giroscopi sono caratterizzati da un offset intrinseco che, se integrato, porta ad un errore lineare divergente, come mostrato in figura 1.2.

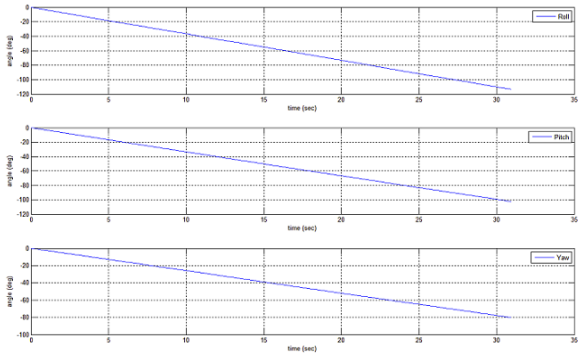


Figura 1.2: Integrazione della velocità angolare fornita dai giroscopi in stato di quiete. Si noti come l'errore relativo all'offset porta ad una inconsistenza della misura.

Come descritto nel [14], il segnale proveniente da un giroscopio può essere modellato attraverso il seguente processo:

$$\omega_m = (1 + s_f) \cdot \omega_t + b_0 + b(t)$$

dove ω_m è la velocità angolare misurata, ω_t è la velocità angolare reale, s_f è un fattore di scala, b_0 è la componente di offset statica e $b(t)$ è un processo stocastico che realizza l'errore legato alla componente variabile di offset ed al rumore di campionamento.
Durante lo svolgimento della nostra attività, ci siamo limitati ad un calcolo statistico di b_0 e $b(t)$ senza considerare il fattore di scala.

1.2.2 Rotazioni e accelerometro

L'accelerometro fornisce le componenti di accelerazione del sistema sui 3 assi x, y, z. Attraverso la trigonometria e supponendo il sistema soggetto soltanto all'accelerazione di gravità, è possibile calcolare l'orientamento del dispositivo rispetto alla terna fissa $\overrightarrow{i}$, $\overrightarrow{j}$, $\overrightarrow{k}$ secondo la convenzione di *Eulero* Yaw, $Pitch$, $Roll$, nel seguente modo:

$$\gamma = atan2(A_y, A_z);\ \phi = atan2\left(A_x, A_z\right);\ \psi = atan2\left(A_y, A_x\right) \tag{1.1}$$

dove γ, ϕ e ψ sono rispettivamente gli angoli di rotazione attorno agli assi x, y, z, A_x, A_y e A_z sono rispettivamente le componenti del vettore gravità lungo i 3 assi e $atan2$ è la funzione arcotangente con immagine $(-\pi, \pi]$.

In alternativa, come esposto spesso nella letteratura [14], è possibile calcolare gli angoli di *Roll* e $Pitch$ attraverso le seguenti relazioni:

$$\gamma = asin\left(\frac{A_y}{|\vec{g}|}\right);\ \phi = asin\left(\frac{A_x}{|\vec{g}|}\right) \tag{1.2}$$

Le equazioni 1.1 e 1.2 soffrono però di alcuni difetti:

1. Quando un asse è allineato con il vettore $\overrightarrow{G}$ non è possibile risalire alle rotazioni attorno allo stesso. Ciò è dovuto al fatto che le componenti utilizzate per calcolare l'angolo di rotazione attorno a tale asse oscillano nell'intorno di zero $[0^-, 0^+]$ con conseguente variazione inconsistente dello stesso tra $[-90, +90]$ (nel caso di utilizzo della funzione $atan$ o $atan2$);

2. La funzione $atan2$ può non distinguere un angolo pari a $\pm 180°$ e $0°$, a causa dei segni degli argomenti;

3. La funzione trigonometrica $asin$ restituisce angoli compresi tra $\pm 90°$.

Per i difetti esposti in precedenza, tali equazioni di misura sono utilizzate principalmente nel settore aeronautico[14] o comunque laddove non sono effettuate manovre estreme, e sono integrate con le informazioni fornite da magnetometro (fare riferimento alla sotto-sezione 1.2.3). Per sopperire ai difetti degli angoli di *Eulero* spesso si ricorre all'utilizzo dei quaternioni (fare riferimento alla sotto-sezione 1.3.3).

Al fine di ottenere misure precise è possibile effettuare una calibrazione dell'accelerometro, come esposto nel [15]. Infatti, le misure di accelerazione fornite possono essere descritte nel seguente modo:

$$\begin{bmatrix} A_{x_1} \\ A_{y_1} \\ A_{z_1} \end{bmatrix} = [A_m] \cdot \begin{bmatrix} \frac{1}{A_{sc_X}} & 0 & 0 \\ 0 & \frac{1}{A_{sc_Y}} & 0 \\ 0 & 0 & \frac{1}{A_{sc_Z}} \end{bmatrix} \begin{bmatrix} A_x - A_{OS_X} \\ A_y - A_{OS_Y} \\ A_z - A_{OS_Z} \end{bmatrix} = [A_{CC}] \cdot \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \begin{bmatrix} A_{CC_{10}} \\ A_{CC_{20}} \\ A_{CC_{30}} \end{bmatrix} \quad (1.3)$$

dove A_{i_1} rappresenta l'accelerazione di gravità reale sull'i esimo asse, A_i è la misura fornita dal sensore sull'i esimo asse, $A_m \in \mathbb{R}^{3\times3}$ è il disallineamento tra gli assi del dispositivo e gli assi dei sensori, A_{SC_i} è la sensibilità del sensore associato all'i esimo asse e A_{OS_i} è l'offset di misura sull'i esimo asse. L'obiettivo della calibrazione è fornire i 12 parametri tali che per ogni misura è possibile risalire ai valori reali di accelerazione. La calibrazione viene effettuata in 6 posizioni stazionarie. Riscrivendo l'equazione 1.3 si ha:

$$\begin{bmatrix} A_{x_1} & A_{y_1} & A_{z_1} \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z & 1 \end{bmatrix} \cdot \begin{bmatrix} A_{CC_{11}} & A_{CC_{21}} & A_{CC_{31}} \\ A_{CC_{12}} & A_{CC_{22}} & A_{CC_{32}} \\ A_{CC_{13}} & A_{CC_{23}} & A_{CC_{33}} \\ A_{CC_{10}} & A_{CC_{20}} & A_{CC30} \end{bmatrix}$$

o in forma compatta:

$$Y = w \cdot X \quad (1.4)$$

dove, per le 6 misurazioni stazionarie, il vettore Y non è altro che il vettore dei valori reali delle accelerazioni lungo i 3 assi. Effettuando n misurazioni (nel nostro caso n è pari a 10) per ogni posizione stazionaria, si ha che w ha dimensione $6 \cdot n \times 4$, così come Y. Effettuando la stima ai minimi quadrati si determina X come:

$$X = [w^T \cdot w]^{-1} \cdot w^T \cdot Y$$

1.2.3 Rotazioni e magnetometro

Il magnetometro fornisce le componenti del campo magnetico presente nell'ambiente lungo i 3 assi x, y, z. L'utilizzo del magnetometro risulta utile in quanto, attraverso la trigonometria e noti gli angoli di rollio γ e beccheggio ϕ, è possibile determinare la rotazione di imbardata ψ [11]:

$$\begin{gathered} M_x = X_h \cdot cos(\phi) + Z_h \cdot sin(\phi); \\ M_y = X_h \cdot sin(\gamma) \cdot sin(\phi) + Y_h \cdot cos(\gamma) - Z_h \cdot sin(\gamma) \cdot cos(\phi); \\ \psi = atan2(M_y, M_x) \end{gathered} \quad (1.5)$$

dove X_i rappresenta la componente magnetica sull'$i-esimo$ asse.
Se un asse è parallelo a $\vec{G}$ allora le equazioni 1.5 si semplificano; ad esempio, se $\overrightarrow{k} \,//\, \overrightarrow{G}$, si ha che $\psi = atan2(Y_h,\, X_h)$.

Le misure fornite dal magnetometro sono soggette ad errori dovuti a distorsioni magnetiche, i quali possono essere ridotti attraverso la calibrazione. Come mostrato nel [16], una calibrazione manuale sufficiente si può ottenere ruotando la piattaforma circolarmente con un asse parallelo al vettore di gravità; nel caso di misure non affette da distorsioni, si otterrebbe una funzione delle misure sui 2 assi non paralleli a $\vec{G}$ molto simile ad una circonferenza di centro l'origine. In realtà, per i motivi esposti in precedenza, spesso tale funzione è più simile ad un ellisse e non ha centro nell'origine. Per correggere le misure provenienti dal sensore è sufficiente applicare la seguente trasformazione:

$$M_r = M_m \cdot M_{sf} + M_{off} \tag{1.6}$$

dove M_m è la misura, M_{sf} è il fattore di scala e M_{off} è l'offset. Per il calcolo di queste ultime due componenti, come mostrato nel [16], è sufficiente effettuare 3 rotazioni, ognuna caratterizzata da un asse diverso parallelo a $\vec{G}$. Al termine di ogni rotazione il fattore di scala e l'offset vengono calcolati come segue:

$$M_{sf\,i} = \frac{max(\vec{M}_j) - min(\vec{M}_j)}{max(\vec{M}_i) - min(\vec{M}_i)}$$

$$M_{off\,i} = \left(\frac{max(\vec{M}_i) - min(\vec{M}_i)}{2} - max(\vec{M}_i) \right) \cdot M_{sf}$$

dove i, j sono i due assi non paralleli a $\vec{G}$ e perpendicolari tra di loro ed$\vec{M}_i$ è la serie di misure ottenute sull'i-esimo asse. La calibrazione del magnetometro permette di ovviare agli effetti di interferenze magnetiche nel frame del sensore. Se il campo magnetico terrestre è affetto da disturbi, attraverso la calibrazione non è possibile rimediare a questi disturbi; per fare ciò è necessario effettuare la compensazione magnetica, spiegata nel [13] e nel capitolo 3.

1.3 Rappresentazione delle Rotazioni nello Spazio

Per definizione, in matematica, una rotazione è una trasformazione del piano o dello spazio euclideo che sposta gli oggetti in modo rigido e che lascia fisso almeno un punto (l'origine dello spazio). I punti che restano fissi nella trasformazione formano un sottospazio: quando questo insieme è un punto (l'origine) o una retta, questi si chiamano rispettivamente il centro e l'asse della rotazione.[14]

La rappresentazione delle rotazioni è stata di nostro interesse per lo sviluppo del progetto per comprendere quale fosse il metodo migliore per definire la posizione del corpo rigido nello spazio.

Esistono 3 principali tecniche di rappresentazione degli angoli di rotazione, ognuna con pro e contro. Nelle prossime sezioni verranno analizzate.

1.3.1 Roll, Pitch e Yaw

Gli angoli di *Roll*, *Pitch* e Yaw furono introdotti dal matematico *Eulero* per rappresentare l'orientazione nello spazio di un corpo rigido. Essi rappresentano le rotazioni attorno agli assi principali mostrate in figura 1.3, e per la precisione:

- *Roll* (Rollio): rotazione attorno all'asse x di un angolo φ;
- *Pitch*(Beccheggio): rotazione attorno all'asse y di un angolo θ;
- Yaw(Imbardata): rotazione attorno all'asse z di un angolo ψ.

Figura 1.3: Rappresentazione delle rotazioni secondo la convezione Roll,Pitch e Yaw.

Con questi tre tipi di rotazioni possono essere rappresentate ogni tipo di trasformazione nello spazio 3D. Questa rappresentazione è molto semplice ed è spesso utilizzata in navigazione (aerei, navi, moto), presenta però dei difetti se si vogliono rappresentare delle rotazioni libere nello spazio. E' necessario prima di tutto definire l'ordine delle rotazioni, ad esempio *Roll*, poi *Pitch* e poi Yaw. Le trasformazioni di *Eulero*, ad esempio, sono delle rotazioni con degli ordini fissati; la convenzione *Eulero I*, mostrata in figura 1.4, impone:

- si ruota la terna origine dell'angolo φ intorno all'asse z;
- si ruota la terna dell'angolo θ intorno all'asse x' corrente;
- si ruota la terna dell'angolo ψ attorno all'asse z'' corrente.

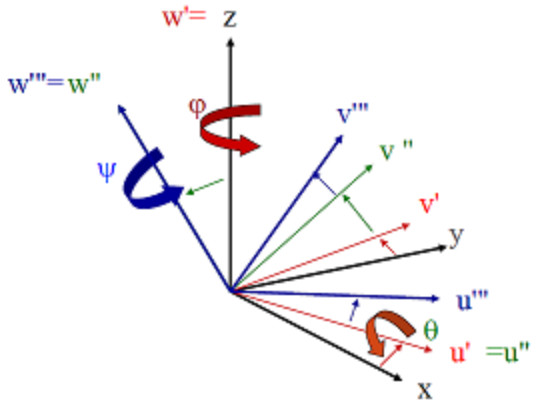


Figura 1.4: Convenzione Eulero 1.

Definito l'ordine di rotazione la rappresentazione *RPY* presenta comunque un problema chiamato *"Gimbal Lock"*; vediamo un esempio:

- Poniamo di avere un aereo che procede con la punta rivolta verso Nord (di conseguenza l'ala sinistra punta a Ovest e la destra a Est);
- Effettuiamo una rotazione di (90,90,0) con (*Roll*, *Pitch* e Yaw) e verso positivo per le trasformazioni antiorarie. La rotazione di *Roll* fa sì che l'ala destra punti verso l'alto e la sinistra verso terra. La rotazione di *Pitch* porta invece l'aereo ad avere la punta dell'areo rivolta verso Ovest;
- Ora effettuiamo un'altra rotazione di (-90,0,0), per riportare le ali in posizione orizzontale. In questo modo abbiamo le ali sul piano orizzontale e l'aereo che punta verso Ovest.
- La rotazione complessiva che è stata effettuata è dunque (0,90,0); applicando però questa rotazione alla posizione originaria otteniamo che il nostro aereo punterà verso l'altro, completamente diverso dalla posizione che è stata raggiunta in seguito alla prima serie di rotazione.

Gimbal Lock è la perdita di un grado di libertà nello spazio 3D quando due assi diventano paralleli, e quindi lo spazio delle rotazioni degenera in uno spazio 2D. Come possiamo vedere rappresentato nelle due immagini di figura 1.5 quando due assi sono paralleli la rotazione di uno o dell'altro asse provoca lo stesso effetto.

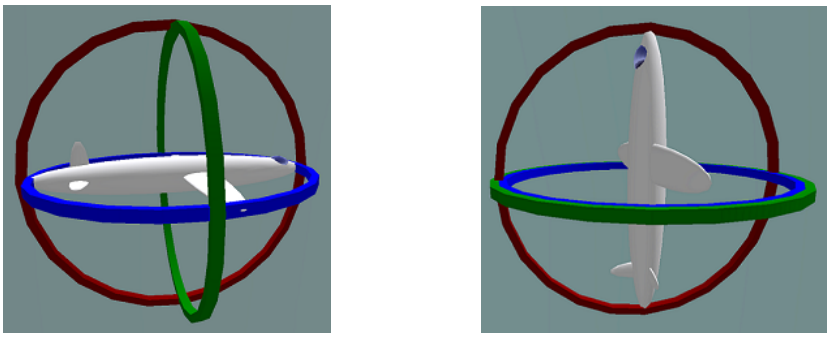


Figura 1.5: Rappresentazione del *Gimbal Lock*; una rotazione di Pitch pari a 90 gradi porta alla pertita di un grado di libertà.

1.3.2 DCM (Direct Cosine Matrix)

*DCM*è l'acronimo di *Direct Cosine Matrix* e rappresenta una matrice utilizzata per rappresentare rotazioni nello spazio tridimensionale. Questa matrice si forma componendo le matrici di singole rotazione degli angoli di *Eulero*. Vediamo come:

- La rotazione attorno all'asse x di un angolo φ è rappresentata dalla seguente matrice:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\varphi & -sin\varphi \\ 0 & sin\varphi & cos\varphi \end{bmatrix}$$

- La rotazione attorno all'asse y di un angolo θ è rappresentata dalla seguente matrice:

$$\begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}$$

- La rotazione attorno all'asse z di un angolo ψ è rappresentata dalla seguente matrice:

$$\begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Una rotazione completa degli angoli di *Eulero* è quindi rappresentata dalla matrice DCM:

$$M_{DCM} = \begin{bmatrix} cos\theta cos\psi & -cos\varphi sin\psi + sin\varphi sin\theta cos\psi & sin\psi sin\varphi + cos\varphi sin\theta cos\psi \\ cos\theta sin\psi & cos\varphi cos\psi + sin\varphi sin\theta sin\psi & -sin\varphi cos\psi + cos\varphi sin\theta sin\psi \\ -sin\theta & sin\varphi cos\theta & cos\varphi cos\theta \end{bmatrix}$$

Con questa rappresentazione è possibile osservare "in forma matematica" il problema del *Gimbal Lock*.

Il problema del *Gimbal Lock* utilizzando le matrici DCM è risolto in quanto permette di "separare" le rotazioni. Se vogliamo applicare una rotazione di (90,90,0) e successivamente di (-90,0,0) come in precedenza, dopo la prima trasformazione viene calcolata la DCM relativa, a cui verrà succesivamente applicata la nuova rotazione.

1.3.3 Quaternioni

I quaternioni sono un'estensione dei numeri complessi introdotti nel 1843 da William Rowan Hamilton. Essi contengono i numeri complessi, e, sul campo reale, sono anche uno spazio vettoriale a 4 dimensioni (analogamente ai complessi, che sono uno spazio sui reali a 2 dimensioni). La loro proprietà principale è la seguente:

$$i^2 = j^2 = k^2 = ijk = -1$$

Da cui derivano le seguenti proprietà moltiplicative tra gli elementi mostrate in tabella 1.1.

$\times$	**1**	**i**	**j**	**k**
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

Tabella 1.1: Proprietà moltiplicative delle componenti di un quaternione.

Un quaternione, come già, detto rappresenta un elemento in $\mathbb{R}^4$ ed è quindi formato da quattro elementi:

$q = (q_0, q_1, q_2, q_3)$da cui possiamo definire il quaternione come una somma:

$q = q_0 + \mathbf{q} = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3$in cui:

- q_0 :parte scalare del quaternione.
- $\mathbf{i}, \mathbf{j}, \mathbf{k}$: sono le basi ortonormali di $\mathbb{R}^3$, cioè $i = (1, 0, 0)$, $j = (0, 1, 0)$ e $k = (0, 0, 1)$;
- $q = \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3$ un ordinario vettore in $\mathbb{R}^3$;
- q_0, q_1, q_2, q_3: sono chiamati i componenti del quaternione.

Altre caratteristiche importanti dei quaternioni sono:

- Il complesso coniugato, definito come segue: $q* = q_0 - q = q_0 - iq_1 - jq_2 - kq_3$;
- La norma di un quaternione è così definita: $N(q) = \sqrt{q * q}$, oppure $N^2(q) = q * q$;
- Il quaternione unitario è invece quello con norma uguale a uno, e quindi: $N^2(q) = 1$;

- Il quaternione puro è caratterizzato da $q_0 = 0$, e cioè è un quaternione composto solamente da parti immaginarie;
- L'inverso di un quaternione è il quaternione tale che $q^{-1}q = 1$, da cui $q^{-1} = \frac{q*}{N^2}$, e per un quaternione unitario $q^{-1} = q*$;

Di particolare interesse per il nostro lavoro sono i quaternioni unitari; difatti ognuno di essi rapresenta una rotazione nello spazio 3D, nel seguente modo:

Per un quaternione unitario vale il seguente teorema:

$$q = q_0 + \mathbf{q} = \cos\theta + \boldsymbol{u}\sin\theta$$

dove $\mathbf{u} = \frac{\mathbf{q}}{|\mathbf{q}|}$ e $\tan\theta = \frac{|\mathbf{q}|}{q_0}$.
A questo punto definiamo l'operatore $L_q(\mathbf{v}) = q \otimes \mathbf{v} \otimes q*$che rappresenta una rotazione di un angolo 2θ del vettore v, usando q come asse di rotazione. Affinchè che questa moltiplicazione rappresenti una rotazione bisogna utilizzare il prodotto tra quaternioni, così definito sfruttando la regola di Hamilton:

$$a \otimes b = [a_0 a_1 a_2 a_3] \otimes [b_0 b_1 b_2 b_3]$$

$$= \begin{bmatrix} a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 \\ a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \\ a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1 \\ a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0 \end{bmatrix}^T$$

Quindi sfruttando la proprietà appena enunciata della moltiplicazione dei quaternioni, e l'operatore di rotazione L possiamo definire la matrice di trasformazione in termini di componenti di quaternioni; siano:

- q: quaternione che definisce la rotazione, composto da $[q_0, q_1, q_2, q_3]$.
- $\mathbf{v}$:vettore appartenente ad $\mathbb{R}^3$che verrà traslato dalla posizione A alla B.
- $\mathbf{v^B} = q \otimes \mathbf{v^A} \otimes q*$ è l'applicazione dell'operatore L al vettore $\mathbf{v}$.

La matrice che rappresenta questa trasformazione è la seguente:

$$\begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & 2q_0^2 - 1 + 2q_2^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix}$$

Da questa matrice possono poi essere calcolati gli angoli di Roll, Pitch e Yaw

$$\varphi = Atan2(2q_2q_3 - 2q_0q_1, 2q_0^2 + 2q_3^2 - 1)$$

$$\theta = -\arcsin(2q_1q_3 + 2q_0q_2)$$

$$\psi = Atan2(2q_1q_2 - 2q_0q_3, 2q_0^2 + 2q_1^2 - 1)$$

1.3.3.1 Vantaggi dei Quaternioni

- I quaternioni non seguono regole "della mano sinistra" o "della mano destra", in quanto sono rappresentazioni di numeri, non sono delle convenzioni;
- Rappresentano una rotazione in modo molto compatto, sfruttando solamente 4 parametri, e la costruzione di un quaternione a partire dall'asse di rotazione e dall'angolo di rotazione è molto semplice al contrario degli altri metodi;
- Tramite un metodo denominato $SLERP$(spherical linear interpolation) permettono di effettuare un'interpolazione tra una rotazione e l'altra per evitare movimenti a scatti. (di particolare interesse per la grafica 3D);
- Evitano il problema del gimbal lock che è stato presentato nei paragrafi precedenti.

Per ulteriori informazioni sui quaternioni si rimanda agli articoli [12, 13].

1.4 Tipi di Algoritmi Descritti nella Letteratura

Negli articoli presenti in letteratura sono esposte svariate tecniche di stima delle rotazioni attraverso informazioni sensoriali fornite da $IMUs$, ognuna con una diversa implementazione. Ciononostante, questi algoritmi possono essere raggruppate in generici gruppi che si differenziano per i criteri di stima e per la teoria matematica utilizzata. Tra questi, quelli che abbiamo analizzato maggiormente e a cui ci siamo ispirati durante il progetto, sono:

- Complementary Filter;
- Quaternion Kalman Filter;
- Quaternion Complementary Filter.

Nei prossimi 3 capitoli verrà esposta, per ciascuna delle tipologie di algoritmi esposte, una particolare implementazione da noi realizzata avente più o meno caratteristiche in comune con quelle descritte nella letteratura. Gli articoli che abbiamo letto e su cui ci siamo basati sono molteplici (fare riferimento alla bibliografia), per cui, quelli che abbiamo cercato di realizzare, sono degli algoritmi che

raggruppano gli aspetti più salienti o chiari delle informazioni presentate, in relazione alla propria classe implementativa.

Capitolo 2

Complementary Filter

In questo capitolo viene esposto il primo algoritmo da noi scritto, prendendo spunto dal [5]. Fra tutti risulta il più semplice da comprendere in maniera concettuale in quanto, oltre che a utilizzare gli angoli per la rappresentazione delle rotazioni, realizza un filtraggio intuitivo e basato su di una matematica poco complessa.

2.1 Descrizione Introduttiva

Il *Complementary Filter*, come lascia intuire il nome stesso, realizza un filtraggio delle informazioni sensoriali in maniera complementare a 1, pesando ciascun segnale con l'adeguato coefficiente (compreso tra 0 e 1). Nell'articolo correlato, gli strumenti di misura di cui vien fatto uso sono l'accelerometro ed il giroscopio. Ciascuno di essi, come descritto nelle sotto-sezioni 1.2.1 e 1.2.2, presenta vantaggi e svantaggi; da una parte, bisogna tenere conto dell'errore maggiormente incidente nelle misure dal giroscopio, l'offset, che rischia di portare alla deriva la stima dell'orientamento, dall'altra è necessario porre attenzione alle eccessive accelerazioni a cui il dispositivo viene sottoposto evitando di stimare rotazioni che in realtà non sono avvenute. Lo schema di principio è mostrato in figura 2.1; la velocità angolare viene integrata e filtrata passa-alto eliminando l'offset, i segnali dell'accelerometro vengono filtrati passa-basso mantenendo soltanto le componenti legate al vettore gravità.

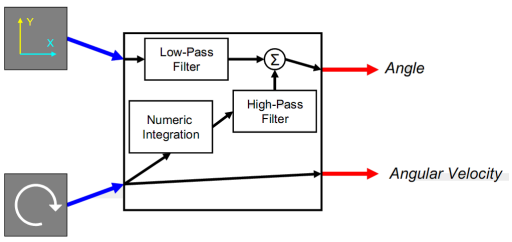


Figura 2.1: Schema di principio del *Complementary Filter*.

Adattando l'algoritmo al nostro contesto applicativo, abbiamo deciso di utilizzare anche le misure fornite dal magnetometro integrandole con quelle dell'accelerometro attraverso le techiche mostrate nelle sotto-sezioni 1.2.2 e 1.2.3. Onde evitare problemi legati alle singolarità delle misure, tale soluzione è stata pensata per gestire rotazioni comprese nell'intervallo $(-90°, +90°)$, calandosi nel contesto di applicazioni aeronautiche, navali, motociclistiche e quant'altro.

2.2 Progetto di Dettaglio

Le due componenti di segnale di cui realizzare il filtraggio sono il giroscopio e l'accelerometro insieme al magnetometro. Di seguito verranno dapprima descritte le techiche matematiche di stima di ciascuna componente, e poi l'algoritmo di filtraggio vero e proprio.

2.2.1 Magnetometro ed Accelerometro

I segnali provenienti da magnetometro ed accelerometro, opportunamente calibrati e normalizzati, sono soggetti ad un primo filtraggio passa-basso, per mezzo di un filtro digitale IIR. Una volta che i segnali sono filtrati e di nuovo normalizzati, gli angoli di Roll e Pitch nella convenzione di *Eulero* sono calcolati con le seguenti relazioni:

$$\gamma = asin\left(\frac{A_y}{|\vec{g}|}\right);\ \phi = asin\left(\frac{A_x}{|\vec{g}|}\right)$$

L'angolo di Yaw viene ottenuto attraverso l'equazione 1.5. Queste equazioni permettono di calcolare il vettore delle tre rotazioni di *Eulero* all'istante corrente, $\vec{\Phi}(t)$. Nelle figure 2.2 e 2.3 sono mostrati rispettivamente i risultati del filtraggio passa basso e del calcolo degli angoli.

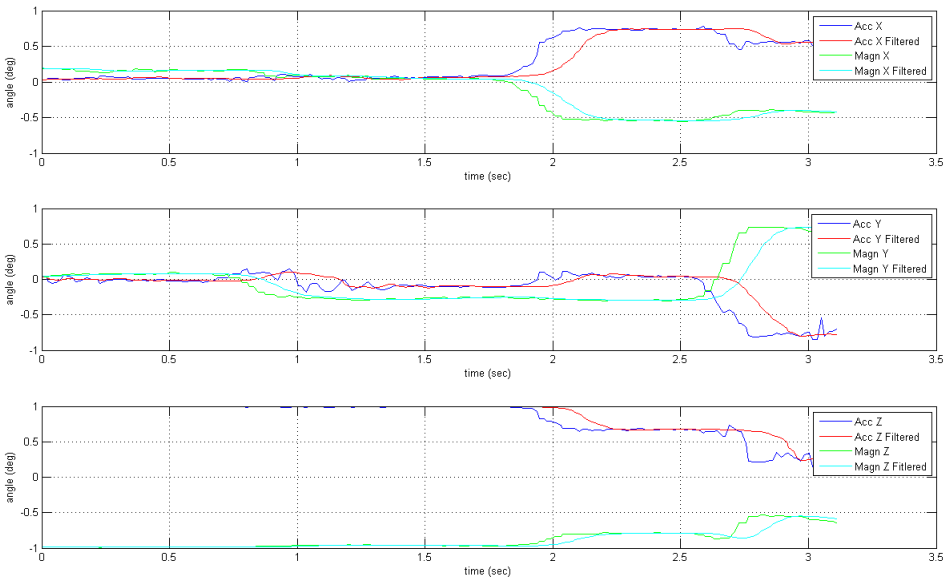


Figura 2.2: Segnali di accelerometro e magnetometro con i relativi filtrati.

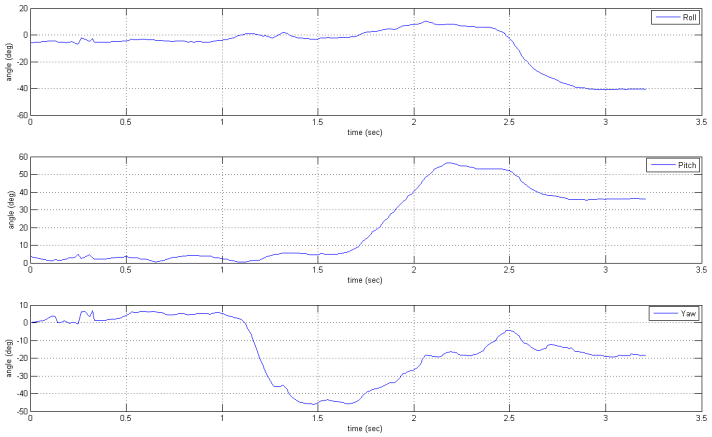


Figura 2.3: Risultato del calcolo degli angoli di *Roll*, *Pitch* e *Yaw*.

2.2.2 Giroscopio

Prima di integrare la velocità angolare, abbiamo calcolato l'offset statico medio caratterizzante ciascun asse del giroscopio, in modo da poterlo sottrarre prima di effettuare il filtraggio. Le rotazioni rispetto alla terna corrente sono quindi calcolate nel seguente modo:

$$\vec{\theta}_t = (\vec{\omega}_r(t) + \vec{\omega}_r(t-1)) \cdot \frac{1}{2} \cdot \Delta t$$

con $\vec{\omega}_r(t) = \vec{\omega}(t) - \vec{\omega}_{OFF}$, dove $\omega(t)$ è la velocità angolare misurata durante l'ultima acquisizione, $\vec{\omega}_{OFF}$ è l'offset statico. Abbiamo deciso inoltre di mediare il segnale con quello all'istante precedente per limitare il $drift$.

Per passare dalle rotazioni espresse rispetto alla terna corrente a quelle di *Eulero*, espresse rispetto alla terna fissa, abbiamo utilizzato la seguente relazione [14]:

$$\begin{bmatrix} \frac{\delta\psi}{\delta t} \\ \frac{\delta\phi}{\delta t} \\ \frac{\delta\gamma}{\delta t} \end{bmatrix} = \frac{1}{cos\,(\gamma)} \cdot \begin{bmatrix} 0 & sin\,(\gamma) & cos\,(\gamma) \\ 0 & cos\,(\gamma)\,cos\,(\phi) & -sin\,(\gamma)\,cos\,(\phi) \\ cos\,(\phi) & sin\,(\gamma)\,sin\,(\phi) & cos\,(\gamma)\,cos\,(\phi) \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.1}$$

dove γ, ϕ, ψ sono gli angoli di *Eulero* di *Roll*, *Pitch* e Yaw rispettivamente, calcolati durante la precedente acquisizione secondo l'equazione 2.2. Moltiplicando il risultato dell'equazione 2.1 per il periodo di acquisizione, si ha che gli angoli di *Eulero* all'istante attuale sono:

$$\vec{\Phi}_{Gyro}(t) = \begin{bmatrix} \psi(t) \\ \phi(t) \\ \gamma(t) \end{bmatrix} = \begin{bmatrix} \psi(t-1) \\ \phi(t-1) \\ \gamma(t-1) \end{bmatrix} + \begin{bmatrix} \frac{\delta\psi}{\delta t} \\ \frac{\delta\phi}{\delta t} \\ \frac{\delta\gamma}{\delta t} \end{bmatrix} \cdot \Delta t \tag{2.2}$$

Tale operazione di moltiplicazione per il periodo di campionamento è possibile perchè le acquisizioni vengono effettuate ad una frequenza che si suppone essere molto maggiore rispetto alla variazione degli angoli stessi. In figura 2.4 è mostrato un esempio di filtraggio dei segnali di giroscopio e di calcolo degli angoli.

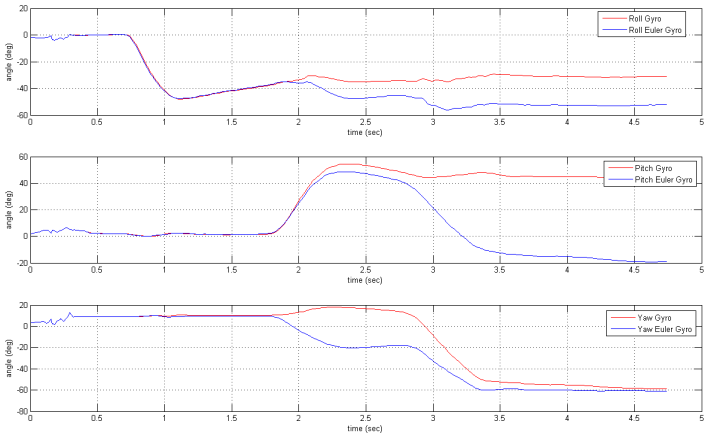


Figura 2.4: Rotazione di $-40°$ attorno a x, di $+40°$ attorno a y e di $60°$ attorno a z rappresentata secondo le rotazioni attorno alla terna corrente (rosso) ed Eulero (blu). Si noti come gli angoli di Eulero differiscono dalla rappresentazione rispetto alla terna corrente, in quanto seguono la convenzione *Yaw*, *Pitch*, *Roll*. In particolare, al termine della rotazione attorno all'asse y (la seconda in ordine cronologico), si nota come la rappresentazione di Eulero necessita di una certa rotazione ($-20°$) attorno all'asse z, anche se questa non è effettivamente avvenuta, mentre al termine della terza, attorno a z, la rappresentazione di Eulero riporta a circa $0°$ l'angolo di *Pitch*.

2.2.3 Filtraggio

Una volta che si dispone di due rappresentazioni nella stessa convenzione (*Eulero*), il filtraggio viene effettuato tenendo conto del fatto che le variazioni a breve termine sono più affidabili se fornite dal giroscopio, mentre la convergenza a regime stazionario è garantita da accelerometro e magnatometro. Si è quindi deciso di calcolare la stima dell'angolo per mezzo della seguente equazione:

$$\hat{\vec{\Phi}}(t) = 0.98 \cdot \vec{\Phi}_{Gyro}(t) + 0.02 \cdot \vec{\Phi}(t) \tag{2.3}$$

dove $\vec{\Phi}_{Gyro}$ è la terna di angoli calcolata attraverso il giroscopio mentre $\vec{\Phi}$ è la terna di angoli calcolata attraverso le misure di accelerometro e magnetometro. Per garantire i requisiti di convergenza a seconda della comportamento dei segnali, le equazioni 2.1 e 2.2 sono state riscritte come segue, introducendo la cosiddetta terna di angoli $\vec{\Phi}_{gyroFilt}$:

$$\begin{bmatrix} \frac{\delta\psi_{gyroFilt}}{\delta t} \\ \frac{\delta\phi_{gyroFilt}}{\delta t} \\ \frac{\delta\gamma_{gyroFilt}}{\delta t} \end{bmatrix} = \frac{1}{cos\,(\gamma)} \cdot \begin{bmatrix} 0 & sin\,(\hat{\gamma}) & cos\,(\hat{\gamma}) \\ 0 & cos\,(\hat{\gamma})\,cos\left(\hat{\phi}\right) & -sin\,(\hat{\gamma})\,cos\left(\hat{\phi}\right) \\ cos\left(\hat{\phi}\right) & sin\,(\hat{\gamma})\,sin\left(\hat{\phi}\right) & cos\,(\hat{\gamma})\,cos\left(\hat{\phi}\right) \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.4}$$

$$\vec{\Phi}_{gyroFilt} = \begin{bmatrix} \psi_{gyroFilt}(t) \\ \phi_{gyroFilt}(t) \\ \gamma_{gyroFilt}(t) \end{bmatrix} = \begin{bmatrix} \hat{\psi}(t-1) \\ \hat{\phi}(t-1) \\ \hat{\gamma}(t-1) \end{bmatrix} + \begin{bmatrix} \frac{\delta\psi_{gyroFilt}}{\delta t} \\ \frac{\delta\phi_{gyroFilt}}{\delta t} \\ \frac{\delta\gamma_{gyroFilt}}{\delta t} \end{bmatrix} \cdot \Delta t \tag{2.5}$$

dove $\hat{\gamma}$, $\hat{\phi}$, $\hat{\psi}$ sono rispettivamente le rotazioni di $Roll$, $Pitch$, Yaw che costituiscono il vettore $\hat{\vec{\Phi}}$ ottenuto dalla 2.3. L'effetto di questa retroazione è di ottenere una bassa risposta sia alle dinamiche di accelerometro e magnetometro che alla stazionarietà (ad esempio offset) del giroscopio. Come mostrato nel [5], la costante di tempo che contraddistingue il sistema è:

$$\tau = \frac{0.98 \cdot \delta t}{0.02}$$

dove δt è il periodo di campionamento. Nel nostro caso δt vale circa $0.02\,sec$, per cui si ottiene una costante di tempo pari a circa $1\,sec$. Per il segnale di accelerometro e magnetometro, la costante di tempo definisce un filtro passa-basso, eliminando segnali ad alta frequenza ovvero variazioni improvvise e ripide dovute ad accelerazioni esterne, mentre per il giroscopio, la costante di tempo caratterizza una sorta di filtro passa-alto attenuando le basse frequenze, come ad esempio il bias che porta all'errore di $drift$. In figura 2.5 sono riportati i risultati applicati ad un esempio.

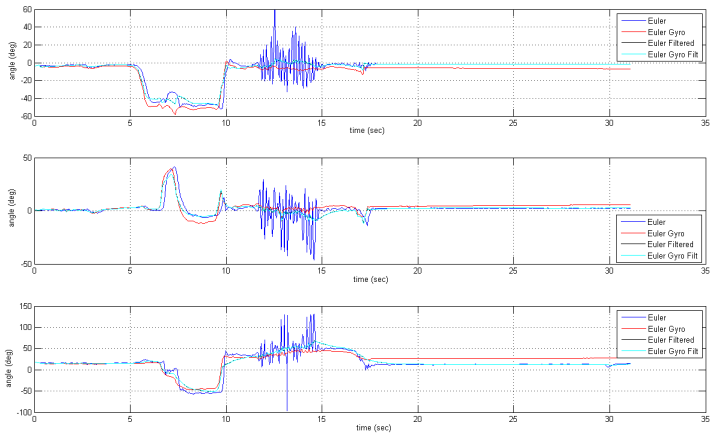


Figura 2.5: Risultati di esempio dell'algoritmo; il segnale filtrato risponde maggiormente alla dinamica del giroscopio piuttosto che a quella di accelerometro e magnetometro. Viceversa, a regime stazionario, il giroscopio porta a minime divergenze dovute all'offset variabile, dato che il maggiore affidamento è legato agli angoli calcolati con accelerometro e magnetometro.

Lo schema a blocchi riassuntivo dell'algoritmo è mostrato in figura 2.6.

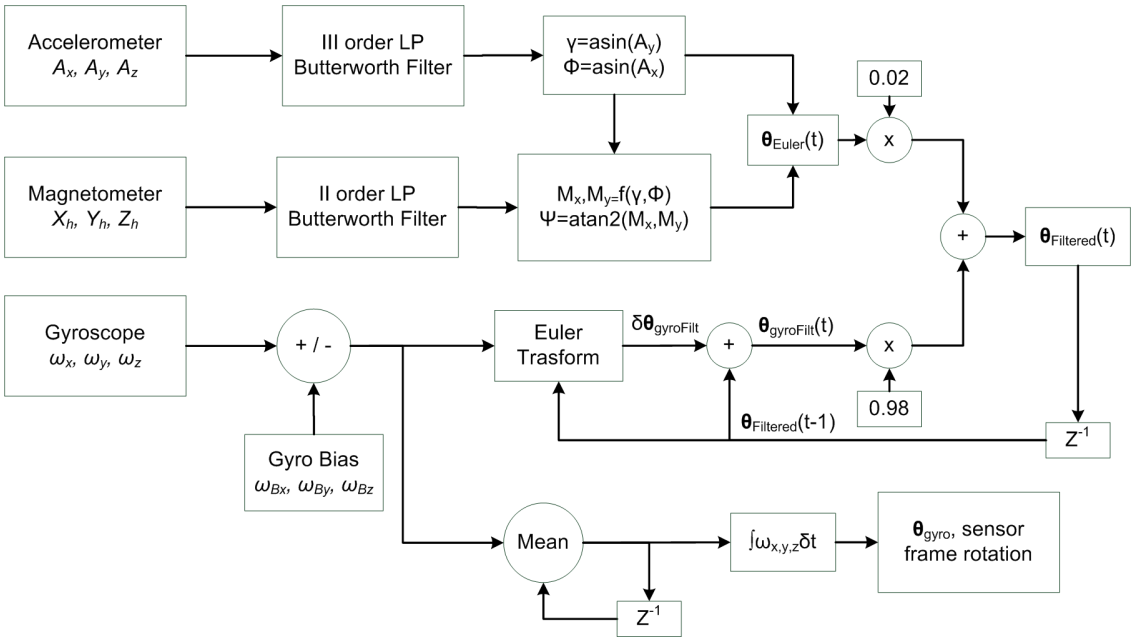


Figura 2.6: Schema a blocchi del *Complementary Filter*.

2.2.4 Progetto C# - La classe ComplementaryFilter

Nel contesto dell'applicativo sviluppato nel linguaggio *C#*, è stata scritta una classe apposita che implementa il *Complementary Filter* spiegato precedentemente, denominata *ComplementaryFilter.cs*. La struttura della classe è la stessa di quella descritta sopra e mostrata in figura 2.6. Nel costruttore vengono inizializzate le variabili che non cambiano nel corso dell'algoritmo, come i filtri *Butterworth*, il periodo di campionamento e la matrice di calibrazione dell'accelerometro. Il metodo denominato *complementaryFiltering*, che riceve come argomenti le triplette di misure provenienti dall'*iNemo*, effettua il filtraggio. In aggiunta vi sono i seguenti metodi:

- *computeAnglesFromAccAndMagn*, che, ricevendo in ingresso le osservazioni da accelerometro e magnetometro filtrate, restituisce la terna di angoli associata alle stesse;
- *computeEulerFromGyro*, che, ricevendo in infresso il vettore delle velocità angolari acquisite e la terna di angoli nella convenzione di *Eulero* calcolati all'iterazione precedente, restituisce la terna di angoli nella convenzione di *Eulero* aggiornata;
- *getAnglesFiltered*, restituisce gli angoli di *Eulero* fitrati con l'algoritmo;
- *limitRotations*, che controlla che la terna di angoli passata in argomento si attenga all'intervallo $[-180°, 180°)$, restituendone una forma corretta nel caso non lo sia.

Di seguito alcuni accorgimenti circa l'utilizzo dell'applicativo:

1. La posizione originale di zero si ottiene allineando l'*iNemo* verso il nord magnetico e con l'asse z parallelo al vettore $\vec{G}$, diretto verso l'alto;

2. Come descritto in precedenza, per motivi realizzativi, le rotazioni devono essere comprese nell'intervallo $-90°<\theta<+90°$, con gli estremi esclusi.

Capitolo 3

Quaternion Kalman Filter

Questo capitolo descrive una nostra implementazione del *Filtro di Kalman* per la stima dell'orientamento. Per poter comprendere al meglio il ruolo di questo algoritmo nel nostro particolare contesto applicativo, abbiamo letto molti degli articoli presenti nella letteratura che ne fanno uso [1, 2, 3, 4, 13, 14] ed abbiamo cercato di realizzare alcuni filtri di *Kalman* seguendo gli stessi articoli. Spesso, a causa dell'alto formalismo utilizzato nelle letture, i risultati si sono rivelati non soddisfacenti, per cui abbiamo preso la decisione di realizzare una nostra versione del filtro di *Kalman*, che prendesse spunto dagli articoli letti ma avesse una struttura pensata da noi. Il risultato di questo lavoro è discusso nelle prossime sezioni. Dato che in questo progetto si fa uso dei quaternioni, si rimanda, per loro proprietà e caratteristiche, alla sotto-sezione 1.3.3.

3.1 Descrizione Introduttiva

Il filtro di *Kalman* è un algoritmo ricorsivo che permette la stima dello stato che caratterizza un sistema dinamico partendo da osservazioni dello stato stesso che sono però non completamente attendibili perchè affette da disturbi. Il modello del sistema dinamico generico a cui applicare il filtro è descritto nella 3.1.

$$\begin{cases} \vec{x}_t = F \cdot \vec{x}_{t-1} + B \cdot \vec{u}_{t-1} + \vec{w}_{t-1} \\ \vec{z}_t = H \cdot \vec{x}_t + \vec{v}_t \end{cases} \tag{3.1}$$

con $\vec{w}_t \sim N(\mu_w, \sigma_w^2)$ innovazione dello stato, ovvero una realizzazione stocastica che modellizza la dinamica dello stato e $\vec{v}_t \sim N(\mu_v, \sigma_v^2)$ il rumore che si sovrappone alle osservazioni dello stato. Il filtro di *Kalman* propone lo stimatore ottimo lineare dello stato basato sulla stima di *Bayes*, introducendo il concetto di probabilità condizionata. Infatti viene fatto uso di propabilità a priori e a posteriori durante la computazione dell'algoritmo. Il filtro di *Kalman* si basa su due principali passi:

1. Il passo di *Predizione*, durante il quale si calcola lo stato a partire da aggiornamenti legati a iterazioni precedenti e attraverso la matrice F che rappresenta l'evoluzione dello stato tra l'istante precedente e quello attuale;

2. Il passo di *Aggiornamento*, durante il quale viene aggiornata la predizione sulla base delle osservazioni z_t.

Inoltre è necessario inizializzare lo stato ad un valore x_0 in modo tale che il filtro possa essere avviato.

La stima della rotazione di una IMU per mezzo del filtro di $Kalman$ può avvenire rappresentando lo spazio degli stati con gli angoli di rotazione (ad esempio gli angoli di $Eulero$) [14] oppure lavorando direttamente con i quaternioni [1, 8, 13, 14]. La decisione dipende dall'applicazione, tenendo presente che i quaternioni sopperiscono ad alcuni problemi legati agli angoli di $Eulero$ (fare riferimento alla sotto-sezione 1.3.3). A prescindere da tale aspetto, l'approccio al filtro di $Kalman$ prevede che il passo di predizione venga compiuto attraverso le velocità angolari del giroscopio, calcolando cioè la variazione dello stato aggiornato al passo precedente sulla base delle misure provenienti dal giroscopio, mentre il passo di aggiornamento vede l'integrazione della predizione con le osservazioni fornite da accelerometro e/o magnetometro. Non è necessario l'utilizzo di un vettore di variabili esogene $\vec{u}$, quindi tale termine viene escluso dal modello del sistema. Dagli articoli da noi analizzati è emerso che il vettore dello stato è rappresentato da due principali gruppi:

$$\vec{x} = \begin{bmatrix} \vec{x}_\omega \\ \vec{x}_{rot} \end{bmatrix} \tag{3.2}$$

dove $\vec{x}_\omega \in \mathbb{R}^{3\times1}$ e rappresenta le velocità angolari associate ai tre assi del giroscopio, e $\vec{x}_{rot}$ rappresenta gli angoli di rotazione; a seconda che si scelga di lavorare con quaternioni piuttosto che con gli angoli di $Eulero$, si avrà che $\vec{x}_{rot} \in \mathbb{R}^{4\times1}$ oppure $\vec{x}_{rot} \in \mathbb{R}^{3\times1}$. Il motivo di mantenere all'interno dello stato anche le componenti delle velocità angolari sta nel fatto che si vuole avere un riscontro diretto sulla stima delle stesse, stante il fatto che sono affette da errore.

Prima di procedere con una nostra versione del filtro di $Kalman$, abbiamo cercato di implementare gli algoritmi descritti dai vari articoli letti, mantenendo la struttura dello stato descritta nella 3.2; i risultati ottenuti non sono stati del tutto soddisfacenti, forse per mancanza di descrizioni poco formali e per una nostra difficoltà nel comprendere concretamente il contesto. In seguito a questi risultati, abbiamo cercato di scrivere il filtro di $Kalman$ basandoci maggiormente su nostri ragionamenti e applicando, dove necessario, le relazioni matematiche della letteratura. Il risultato è mostrato nella seguente sezione.

3.2 Progetto di Dettaglio

La prima decisione da noi presa è stata quella di ridurre e semplificare la dimensione dello stato evitando una rappresentazione esplicita delle componenti di velocità angolare del giroscopio. Per

quanto riguarda lo stato, al fine di ottenere un algoritmo che funzionasse per qualsiasi configurazione di angoli di rotazione, abbiamo deciso di rappresentarlo attraverso i quaternioni:

$$\vec{x} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{3.3}$$

3.2.1 Predizione dello Stato

Come mostrato negli articoli [1, 8, 13, 14] la seguente relazione rappresenta l'evoluzione dei quaternioni in relazione alle velocità angolari:

$$\dot{q} = \frac{1}{2} \cdot \vec{x} \otimes \vec{\omega} \tag{3.4}$$

Abbiamo quindi utilizzato questa equazione per rappresentare la matrice F che lega lo stato $\vec{x}_t$ a $\vec{x}_{t-1}$. Infatti, si ha che:

$$\vec{x}_t = \vec{x}_{t-1} + \dot{q}_t \cdot \delta t = F \cdot \vec{x}_{t-1} \tag{3.5}$$

dove ma la matrice F è stata determinata in maniera coerente ed è pari a:

$$F = \begin{bmatrix} 1 & -\frac{1}{2} \cdot \omega_x \cdot \delta t & -\frac{1}{2} \cdot \omega_y \cdot \delta t & -\frac{1}{2} \cdot \omega_z \cdot \delta t \\ \frac{1}{2} \cdot \omega_x \cdot \delta t & 1 & \frac{1}{2} \cdot \omega_z \cdot \delta t & -\frac{1}{2} \cdot \omega_y \cdot \delta t \\ \frac{1}{2} \cdot \omega_y \cdot \delta t & -\frac{1}{2} \cdot \omega_z \cdot \delta t & 1 & \frac{1}{2} \cdot \omega_x \cdot \delta t \\ \frac{1}{2} \cdot \omega_z \cdot \delta t & \frac{1}{2} \cdot \omega_y \cdot \delta t & -\frac{1}{2} \cdot \omega_x \cdot \delta t & 1 \end{bmatrix} \tag{3.6}$$

dove δt è il periodo di campionamento. Come si nota, la matrice F non è fissata, ma varia per ogni iterazione, definendo un approccio al filtro di *Kalman* di tipo *Esteso*.

Per modellizzare il vettore stocastico $\vec{w}$ abbiamo calcolato la varianza delle velocità angolari associate a ciascun asse del giroscopio, eliminando prima l'offset statico, ed ottenendo il vettore $\vec{\sigma}_\omega = \begin{bmatrix} \sigma_x & \sigma_y & \sigma_z \end{bmatrix}^T$. Per determinare la matrice Q di varianze e covarianze dello stato abbiamo considerato l'incertezza che contraddistingue ciascuna componente dello stato, legata alla varianza del giroscopio:

$$Q = E\left[\begin{bmatrix} -\omega_x - \omega_y - \omega_z \\ \omega_x - \omega_y + \omega_z \\ \omega_x + \omega_y - \omega_z \\ -\omega_x + \omega_y + \omega_z \end{bmatrix} \cdot \begin{bmatrix} -\omega_x - \omega_y - \omega_z & \omega_x - \omega_y + \omega_z & \omega_x + \omega_y - \omega_z & -\omega_x + \omega_y + \omega_z \end{bmatrix} \right] \tag{3.7}$$

Assumendo che $E[\omega_i] = 0$ e $E[\omega_i \cdot \omega_j] = 0$, $\forall i \neq j$, ovvero che le velocità attorno agli assi siano scorrelate l'una con l'altra, si ha che la matrice di varianza e covarianza dello stato è pari a:

$$Q = \begin{bmatrix} \sigma_x^2 + \sigma_y^2 + \sigma_z^2 & -\sigma_x^2 + \sigma_y^2 - \sigma_z^2 & -\sigma_x^2 - \sigma_y^2 + \sigma_z^2 & \sigma_x^2 - \sigma_y^2 - \sigma_z^2 \\ -\sigma_x^2 + \sigma_y^2 - \sigma_z^2 & \sigma_x^2 + \sigma_y^2 + \sigma_z^2 & \sigma_x^2 - \sigma_y^2 - \sigma_z^2 & -\sigma_x^2 - \sigma_y^2 + \sigma_z^2 \\ -\sigma_x^2 - \sigma_y^2 + \sigma_z^2 & \sigma_x^2 - \sigma_y^2 - \sigma_z^2 & \sigma_x^2 + \sigma_y^2 + \sigma_z^2 & -\sigma_x^2 + \sigma_y^2 - \sigma_z^2 \\ \sigma_x^2 - \sigma_y^2 - \sigma_z^2 & -\sigma_x^2 - \sigma_y^2 + \sigma_z^2 & -\sigma_x^2 + \sigma_y^2 - \sigma_z^2 & \sigma_x^2 + \sigma_y^2 + \sigma_z^2 \end{bmatrix} \tag{3.8}$$

Utilizzando quindi la notazione del filtro di *Kalman*, secondo cui la predizione dello stato è $\vec{x}_{t|t-1}$ si ha che il passo di predizione è:

$$\vec{x}_{t|t-1} = F \cdot \vec{x}_{t-1|t-1} \tag{3.9}$$

dove $\vec{x}_{t-1|t-1}$ è il vettore dello stato aggiornato calcolato al passo precedente. La predizione dello stato così ottenuta viene normalizzata. Per quanto riguarda la varianza sull'errore di predizione, questa è calcolata come:

$$P_{t|t-1} = \left(F \cdot Q \cdot F^T\right) + Q \tag{3.10}$$

Le deviazioni standard delle velocità associate agli assi del giroscopio sono state calcolate eliminando dapprima l'offset statico ed effettuando svariate misure in condizioni stazionarie; i risultati ottenuti sono i seguenti:

$$\vec{\sigma}_\omega \backsimeq \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

Un accorgimento che abbiamo preso per evitare un errore rilevante legato all'integrazione nel tempo, è stato quello di mantenere un periodo di campionamento relativamente basso; dopo una serie di prove, abbiamo verificato che un buon valore di δt sta nell'intorno di $7 - 10\,ms$.

3.2.2 Calcolo delle Osservazioni

Una volta effettuato il passo di predizione, è necessario acquisire le osservazioni da accelerometro e magnetometro ed integrarle al fine di ottenerne una rappresentazione nello spazio degli stati. Per fare ciò ci siamo avvalsi del metodo *Gradient Descent*, mostrato in dettaglio nell'Appendice, che restituisce i quaternioni legati alle correnti misure $\vec{x}_t$. Il coefficiente μ viene preso ad ogni iterazione nel seguente modo [13]:

$$\mu = 10 \cdot |\dot{q}| \cdot \delta t$$

dove $\dot{q}$ è ottenuto attraverso l'equazione 3.4.
Per migliorare il funzionamento dell'algoritmo, abbiamo integrato nel *Gradient Descent* i seguenti accorgimenti:

- Raffinamento delle osservazioni di accelerometro e magnetometro attraverso i risultati delle calibrazioni (fare riferimento alle sotto-sezioni 1.2.2 e 1.2.3);
- Compensazione magnetica delle misure dal magnetometro, prendendo spunto dall'articolo [13]. Tale soluzione permette di diminuire gli errori dovuti alle fonti di interferenza del campo magnetico terrestre rispetto alla corrente zona in cui ci si trova, oltre ad evitare che l'algoritmo dipenda da un prefissato vettore di riferimento; in questo modo gli effetti di una errata inclinazione del campo magnetico terrestre misurato (rispetto alla terna fissa) possono essere corretti se il riferimento del campo magnetico terrestre (rispetto alla terna fissa) all'interno dell'algoritmo è soggetto alla stessa inclinazione. Il procedimento è il seguente:

  1. Si calcola il riferimento rispetto alla terna corrente, affetto da interferenze, attraverso lo stato aggiornato al passo precedente ${}^{E}\vec{h}_t = \vec{q}_{t-1|t-1} \otimes^{S} \vec{m}_t \otimes \vec{q}^{*}_{t-1|t-1}$;
  2. Si calcola il riferimento da utilizzare nell'algoritmo ${}^{E}\vec{b}_t = \begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix}$.

I quaternioni così ottenuti vengono poi normalizzati.

3.2.3 Aggiornamento

Il passo di aggiornamento permette di aggiornare la predizione $\vec{x}_{t|t-1}$ per mezzo delle osservazioni e sulla base di quanto affidamento viene dato alle osservazioni stesse piuttosto che alla predizione. Ciò viene rappresentato attraverso il guadagno del filtro di *Kalman*. Questo termine (nel nostro caso una matrice quadrata di ordine 4) è il fattore moltiplicativo della differenza tra osservazioni e predizione. Maggiore sarà l'affidamento sulle osservazioni, maggiore sarà il guadagno e quindi la differenza tra aggiornamento e predizione. Il guadagno, oltre che dipendere dalla varianza dell'errore di predizione, è determinato anche dalla matrice di varianze e covarianze delle osservazioni, R. Più questa matrice è costituita da terrmini piccoli, più l'affidamento alle osservazioni sarà grande. È chiaro che bisogna raggiungere il giusto compromesso per fare in modo che l'aggiornamento sia poco soggetto a variazioni evidenti e allo stesso tempo possa convergere in un tempo relativamente basso. Il guadagno del filtro è definito come segue:

$$K = P_{t|t-1} \cdot H \cdot \left(H \cdot P_{t|t-1} \cdot H^T + R\right)^{-1} \tag{3.11}$$

dove la matrice H che lega le osservazioni provenienti dal *Gradient Descent* allo stato è una identità di ordine 4, in quanto le osservazioni sono esattamente ciò che viene fornito dall'algoritmo di minimizzazione. Il passo di aggiornamento viene quindi calcolato nel seguente modo:

$$\vec{x}_{t|t} = \vec{x}_{t|t-1} + K \cdot \left(\vec{x}_t - \vec{x}_{t|t-1}\right) \tag{3.12}$$

Infine la varianza dell'errore di aggiornamento viene calcolata come segue:

$$P_{t|t} = (I_{4\times4} - K \cdot H) \cdot P_{t|t-1} \tag{3.13}$$

Per quanto riguarda la matrice R, nella letteratura è suggerito di porla pari a $I_{4\times4} \cdot 0.001$. Tale valore si è risultato troppo piccolo, determinando oscillazioni piccole ma evidenti nell'intorno dello stato ottimo, per cui, dopo svariate operazioni di "tuning", è stata posta pari a $I_{4\times4} \cdot 0.05$.

3.2.4 Inizializzazione

Al primo passo d'iterazione il filtro di *Kalman* necessita di valori iniziali dell'aggiornamento e della varianza dell'errore di aggiornamento; nella letteratura lo stato iniziale viene spesso posto a $\begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$ oppure a $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$. Noi abbiamo scelto il primo vettore. Per quanto riguarda la matrice P_0, poco trattata negli articoli, abbiamo effettuato delle operazioni di "tuning" ottenendo un valore appropriato per $P_0 = I_{4\times4} \cdot 2$. Tale valore determina, partendo dal valore iniziale potenzialmente non corretto, un andamento a regime iniziale dell'ordine della decina di secondi. Per ottenere una maggiore velocità di convergenza iniziale, anzichè sollecitare il sistema in maniera automatica, abbiamo deciso di delegare tale compito all'utente; al massimo, in circa 2/3 secondi l'algoritmo è pronto per essere utilizzato.

3.2.5 Schema a Blocchi e Risultati

In figura 3.1 è mostrato lo schema a blocchi dell'algoritmo.

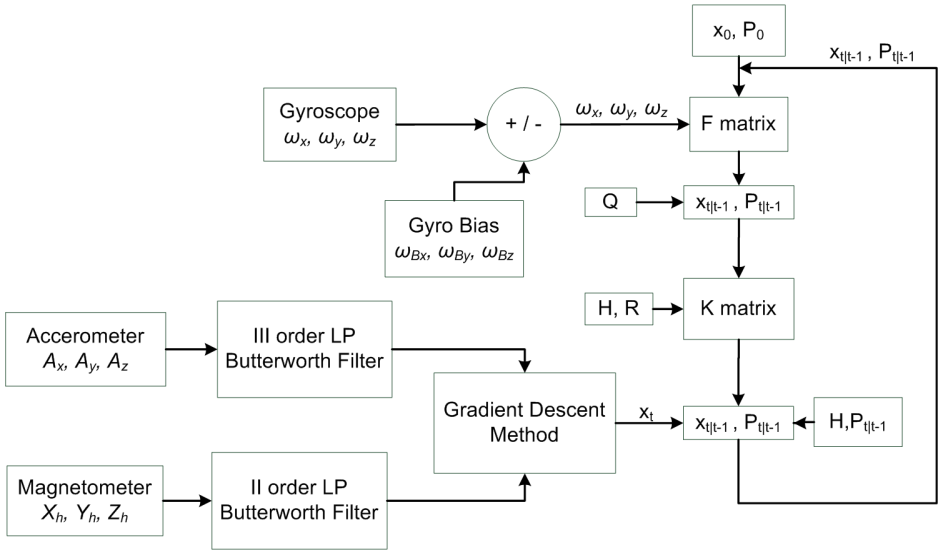


Figura 3.1: Schema a blocchi del filtro di *Kalman* da noi realizzato.

Per quanto riguarda alcuni risultati del filtro, nelle figure 3.2 e 3.3 sono mostrati rispettivamente l'andamento dei quaternioni determinati con il metodo *Gradient Descent* e con il filtro di *Kalman* e i relativi angoli nella convenzion *Yaw*, *Pitch*, *Roll*.

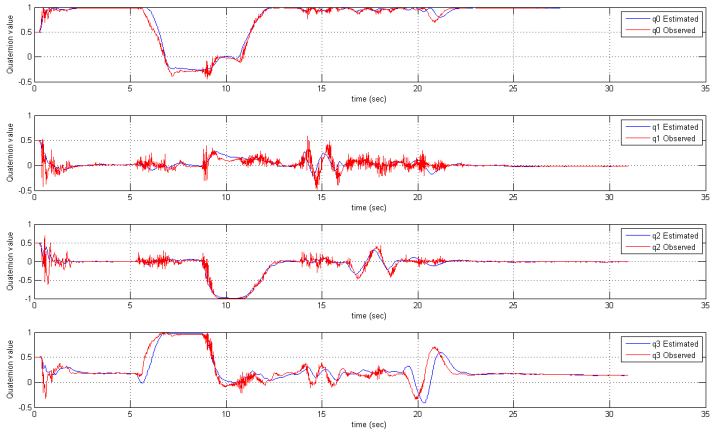


Figura 3.2: Andamento dei quaternioni calcolati con il metodo *Gradient Descent* (in rosso) e con il filtro di *Kalman*. Si noti come le osservazioni siano soggette ad una oscillazione nell'intorno del punto di convergenza (caratteristico "zig-zag" del metodo), caratteristica che non si riperquote sull'aggiornamento. I primi 2/3 secondi sono relativi all'assestamento iniziale dell'algoritmo, che si sposta da $\vec{x}_0 = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$ a regime.

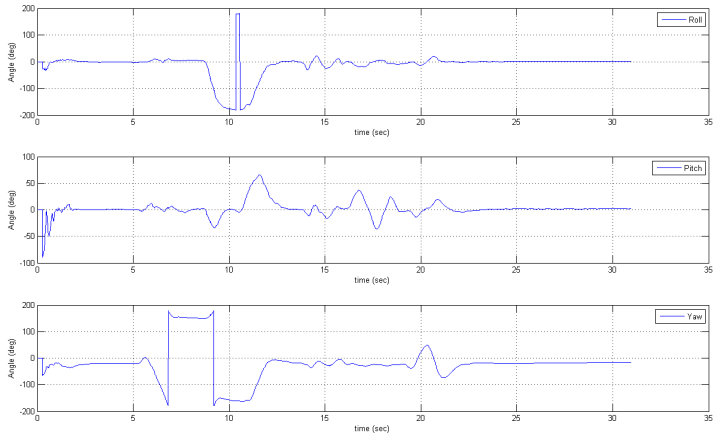


Figura 3.3: Angoli associati ai quaternioni mostrati in figura 3.2. Dapprima l'*iNemo* viene ruotato di 180 gradi attorno all'asse z, poi di circa 180 gradi attorno ad x ed infine di 180 gradi attorno a y. Quest'ultima rotazione è meno evidente perchè rappresentata come composizione di tre rotazioni. Al termine di queste rotazioni seguono alcune operazioni di rollio, beccheggio e imbardata.

3.2.6 Progetto C# - La classe KalmanFilter

Nel contesto dell'applicativo sviluppato nel linguaggio *C#*, è stata realizzata una classe che implementa il filtro di *Kalman* spiegato precedentemente, denominata *KalmanFilter.cs*. La struttura della classe è la stessa di quella descritta sopra e mostrata in figura 3.1.
Nel costruttore vengono inizializzate tutte le variabili che non cambiano nel corso dell'algoritmo (ad esempio, le matrici Q ed R), mentre il metodo *filter_step* realizza i passi di predizione ed aggiornamento del filtro. Tale metodo riceve in argomento la tripletta di misure provenienti dall'*iNemo*. Altri metodo di supporto definiti sono i seguenti:

- *GradientDescent*, calcola le osservazioni dei quaternioni ricevendo in ingresso le misure di accelerometro e magnetometro;
- *QuaternionProduct*, realizza il prodotto di quaternioni ricevendo come argomenti i due quaternioni da moltiplicare;
- *newQuaternion*, che, dati quattro numeri reali di tipo *double*, costruisce il quaternione corrispondente.

Capitolo 4

AHRS Quaternion Estimation

4.1 Descrizione Introduttiva

Il software implementa l'algoritmo *AHRS estimation* basato sui quaternioni per poter visualizzare in real-time l'inclinazione e la posizione della scheda nello spazio. Grazie alle librerie grafiche di *Windows*, si visualizza la scheda come un parallelepipedo che ruota in base ai movimenti che vengon fatti fare al dispositivo, come visibile in figura 4.1

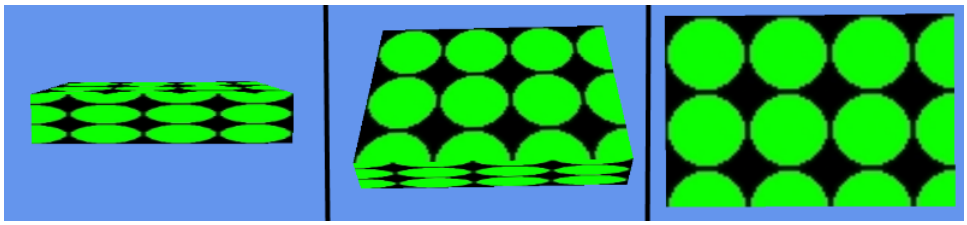


Figura 4.1: Sequenza che rappresenta l'output del software per una rotazione sull'asse di rollio di 90 gradi

4.2 Progetto di Dettaglio

L'algoritmo è stato scritto a partire dall'articolo [13] e dai sorgenti presenti sul sito `http://code.google.com/p/imumargalgorithm30042010sohm/`. Vengono utilizzati i quaternioni per rappresentare le rotazioni ed il funzionamento è descritto dallo schema a blocchi di figura 4.2. Nella figura possiamo notare due zone tratteggiate:

- Group 1: effettua la compensazione del campo magnetico, in modo che se anche variano le condizioni l'algoritmo si adatta ail nuovo ambiente.
- Group 2: effettua la compensazione del drift presente nel giroscopio.

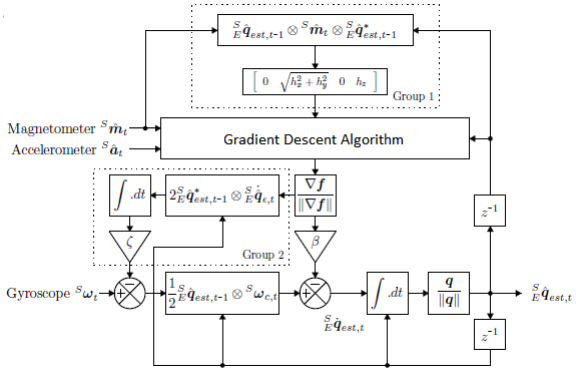


Figura 4.2: Schema a blocchi dell'algoritmo AHRS

L'algoritmo viene chiamato ogni volta che si ricevono dei nuovi dati, e necessita solamente di due parametri dipendenti da ogni scheda per poter funzionare, e questi parametri sono:

- $GyroMeasError$: rappresenta l'errore medio presente nelle misure del giroscopio
- $GyroBiasDrift$: rappresenta un valore che indica il drift che viene introdotto integrando le misure del giroscopio.

A grandi linee l'algoritmo funziona così:
dai segnali del giroscopio si stima la variazione dei quaternioni. Tramite l'accelerometro e il magnetometro viene fatta un'ulteriore stima della stessa variazione, e successivamente le due misure vengono unite usando un metodo similare a quello del *Complementary Filter*. Questa stima della variazione dei quaternioni viene poi integrata sull'intervallo di tempo tra una chiamata al filtro e l'altra. I quaternioni sono poi normalizzati.

Ad ogni passo si sfrutta la stima precedente dei quaternioni per effettuare una compensazione del campo magnetico, in caso di possibili variazioni della direzione del flusso. Inoltre la stima dei quaternioni fatta tramite l'utilizzo di accelerometro e magnetometro viene utilizzata per effettuare una compensazione del drift del giroscopio.

Non entriamo nei dettagli dell'algoritmo in quanto sono ampiamente descritti nell'articolo [13].

4.2.1 Classe C# AHRSalgorithm

E' la classe che implementa l'algoritmo, ed oltre al costruttore e un metodo per settare i parametri dell'algoritmo presenta il metodo *Update* che viene invocato ogni volta che abbiamo a disposizione delle nuove misurazioni dai sensori. Il metodo si sviluppa in modo molto lineare, eseguendo in successione le seguenti operazioni:

- Per prima cosa normalizza i vettori dei segnali provenienti da magnetometro e accelerometro;

- Successivamente calcola le matrici della funzione obbiettivo e la *Jacobiana* per l'applicazione dell'algoritmo *Gradient Descent*. In questa fase si risolve un problema di ottimizzazione, e cioè, a partire dalle misure di accelerometro e magnetometro, si stima quali sono i quaternioni che rappresentano la posizione della scheda minimizzandone l'errore;
- Viene poi effettuata la compensazione del bias del giroscopio;
- A questo punto vengono stimati i quaternioni a partire dalle misure del giroscopio filtrate;
- I quaternioni vengono poi normalizzati e memorizzati;
- A questo punto viene effettuata una compensazione del campo magnetico per poter migliorare la stima tramite il magnetometro nei passi successivi, ed attenuare possibile variazioni nel campo magnetico.

Capitolo 5

Angle Estimation Application

5.1 Implementazioni Matlab

L'ambiente di sviluppo maggiormente utilizzato, soprattutto durante le fase di costruzione e verifica degli algoritmi, è stato *Matlab*. Di seguito gli script implementati durante il progetto:

- Connessione e disconnessione dall'*iNemo*, aspetti non banali che hanno richiesto parecchio tempo;
- Calibrazione del magnetometro;
- Calibrazione dell'accelerometro;
- Stima dell'offset statico medio relativo alla velocità angolare di ciascun asse del giroscopio;
- Stima della varianza della velocità angolare in condizioni statiche per ciascun asse del giroscopio;
- Metodo *Gradient Descent*;
- *Complementary Filter*;
- *Kalman Filter*;
- Molti altri script di contorno o relativi ad altre soluzioni poi abbandonate.

5.2 Implementazioni C#

Per ottenere un riscontro visivo in tempo reale, abbiamo deciso di realizzare una applicazione nel linguaggio *C#* che fosse in grado di mostrare l'orientamento dell'*iNemo* per mezzo di un solido a forma di parallelepipedo, per ciascun algoritmo descritto nei capitoli precedenti. Nelle prossime

sezioni verranno mostrate alcune caratteristiche di questa applicazione ed un breve manuale per il suo utilizzo.

5.2.1 C#

Il C# (si legge C sharp) è un linguaggio di programmazione object-oriented sviluppato da Microsoft all'interno dell'iniziativa .NET. La sintassi del C# prende spunto da quella del Delphi, del C++, di Java e di Visual Basic per gli strumenti di programmazione visuale e per la sua semplicità (meno simbolismo rispetto a C++, meno elementi decorativi rispetto a Java).
Rispetto al C/C++ ha subito modifiche volte ad evitare le disambiguità tipiche della programmzione C, quali ad esempio la gestione dei puntatori o della memoria, che possono essere fatti in C# ma solamente in blocchi di codice definiti "unsafe". Inoltre è presente un garbage collector come in Java, che evita i problemi di dangling pointer o memory leak, il tutto a lieve discapito delle prestazioni. Queste caratteristiche fanno sembrare il C# molto simile a Java; in realtà sono presenti diverse differenze, quali appunto la gestione di puntatori e codice unsafe, oppure il fatto che C# permette che alcune eccezioni non vengano gestite, al contrario di Java che riguardo alle eccezioni è molto rigido.

5.2.2 XNA

Microsoft XNA (che sta per **X**NA is **N**ot **A**cronymed) è un insieme di strumenti per la progettazione, lo sviluppo e la gestione di software per videogiochi reso disponibile da Microsoft per semplificare il processo di creazione, evitando ai programmatori l'inserimento di parti inutili di codice e consentendo di appoggiarsi a un framework unificato, che include tutti i set di istruzioni utili per un videogame.
E' virtualmente supportato da tutti i linguaggi .NET ma è consigliato l'utilizzo di C#. Per la gestione delle primitive grafiche si appoggia sulle DirectX, le librerie grafiche di Windows. XNA facilita la gestione della grafica e delle animazioni, nonostante sia creato per la realizzazione di giochi si può facilmente utilizzare per visualizzare semplici animazioni come nel nostro caso.
La programmazione in XNA si differenzia dalla programmazione ad oggetti, o ad eventi, perchè si basa su di un loop che continua a ripetersi. Difatti la classe Game (tipica di XNA) che gestisce l'interfaccia grafica, richiama periodicamente due metodi (*Update* e *Draw*) che permettono di aggiornare la logica del gioco, il primo, e scrivere sullo schermo il secondo.
Per approfondire la conoscenza si rimanda al sito ufficiale `http://msdn.microsoft.com/en-us/aa937791`.

5.2.2.1 Program.cs

E' la classe di avvio del software, inizializza ed avvia la classe Game che è responsabile della gestione della grafica.

5.2.2.2 Game.cs

E' la classe che è alla base della programmazione XNA, permette di gestire le animazioni e la grafica. Questa classe ha una funzione molto simile ad un Thread che continua a chiamare le stesse due funzioni ad intervalli regolari. Difatti una volta che nella classe Program.cs viene chiamato il metodo Run della classe Game.cs, ad intervalli regolari (60 volte al secondo), vengono chiamati i metodi *Update* e *Draw*. Analizziamo ora i metodi:

- Game(string args_1, string args_2) : costruttore della classe;
- *Initialize()* : metodo che viene chiamato solamente una volta quando viene istanziata la classe ed inizializza tutte le classi e parametri che verranno utilizzate durante il programma. Imposta anche i parametri relativi al parallelepipedo che rappresenterà la nostra scheda tramite il metodo initializeWorld().
- *LoadContent()* : anche questo metodo viene chiamato solamente all'inizio ed è dove vengono caricati tutti i contenuti esterni, quale ad esempio la Texture che verrà visualizzata sul parallelepipedo.
- *PacketReceived(INEMO2_FrameData data)* : invocato periodicamente secondo il periodo di campionamento, acquisisce i dati dall'*iNemo* fornendoli all'algoritmo di stima;
- *Update(GameTime gameTime) :* controlla se viene premuto o il tasto destro o il tasto sinistro del mouse. Il tasto sinistro serve per settare la nuova posizione di zero, mentre il tasto destro è per passare alla modalità di visualizzazione fullscreen.
- *Draw(GameTime gameTime) :* disegna a schermo il parallelepipedo ruotato in base ai quaternioni che di volta in volta vengono calcolati dall'algoritmo.

5.2.2.3 AcquisitionThread.cs

E' la classe delegata all'acquisizione dei dati. E' un thread che ad intervalli pari al periodo di campionamento richiede i dati all'iNEMO. Questo thread rimane sempre attivo finchè è attiva l'applicazione. Si occupa anche della connessione e disconnessione ad alto livello della scheda iNEMO.

5.2.2.4 BasicShape.cs

Definisce il parallelepipedo che rappresenta l'iNEMO, si occupa della definizione delle dimensione e degli aspetti grafici.

5.2.2.5 INEMO2_SDK_Wrapper.cs

Questa classe si occupa del collegamento e dell'accesso all'iNEMO ad un livello più basso. Un livello più basso nel senso che permette di utilizzare le librerie di accesso all'iNEMO scritte in C, in C#.

5.2.3 Funzionamento

La classe *Program* istanzia ed avvia *Game* che gestisce la grafica.
Al decorrere di multipli del periodo di campionamento, la classe *AcquisitionThread* si interfaccia con l'iNEMO tramite l'*iNEMO2_SDK_Wrapper* ed acquisisce i dati da accelerometro, giroscopio e magnetometro. Questi dati li invia alla classe *Game*, la quale aggiorna i quaternioni necessari per la visualizzazione invocando un metodo dell'algoritmo corrente che restituisce i quaternioni stimati all'istante corrente.
La classe *Game* fa uso della *BasicShape* per la visualizzazione a schermo. Lo schema di funzionamento può essere osservato in figura 5.1

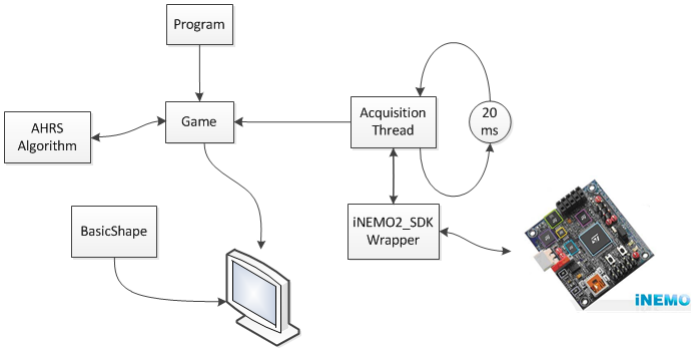


Figura 5.1: Schema di funzionamento del software

5.3 Manuale d'Uso

Prima di avviare il software collegare l'iNEMO ad una porta USB libera e verificare la porta a cui è collegata. Una volta che si è a conoscenza della porta si può avviare il programma, che come prima cosa richiederà a che porta è collegato l'iNEMO e quale algoritmo avviare, come visibile in figura 5.2. Nello spazio riservato al numero della porta inserire solamente il numero, ad esempio se l'iNEMO è collegato alla COM4, scrivere 4.

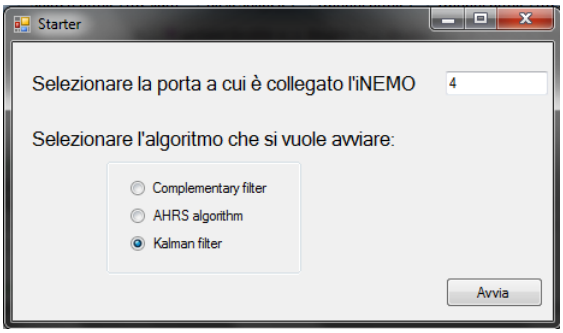


Figura 5.2: Form d'avvio del programma

Una volta cliccato sul pulsante Avvia compare il form di selezione delle impostazioni (figura 5.3), in cui è possibile definire alcuni parametri caratteristici degli algoritmi. In particolare in base all'algoritmo che si è scelto nel precedente form saranno attive diversi parametri:

- Complementary Filter: se si è scelto questo algoritmo saranno disponibili alla scelta gli offset del giroscopio e la frequenza a cui lavorerà il filtro.
- AHRS algorithm: in questo caso i parametri da definire sono solamente due, "Gyro meas error" e "gyro bias drift" come descritto nel capitolo relativo all'AHRS algorithm.
- Kalman filter: in questo caso i parametri che si possono settare sono molti di più. In particolare si possono definire gli offset e le varianze del giroscopio, la frequenza di lavoro e tutti i parametri relativi al magnetometro. In particolare per il magnetometro sono definiti il fattore di scala e l'offset, necessari per la calibrazione, come descritto in [16].

Al contrario di inserire tutti i parametri è possibile avviare gli algoritmi in una configurazione standard, premendo sul pulsante "Usa parametri standard". Se non si vogliono usare questi parametri riempire tutti i campi e poi premere su "Salva parametri".

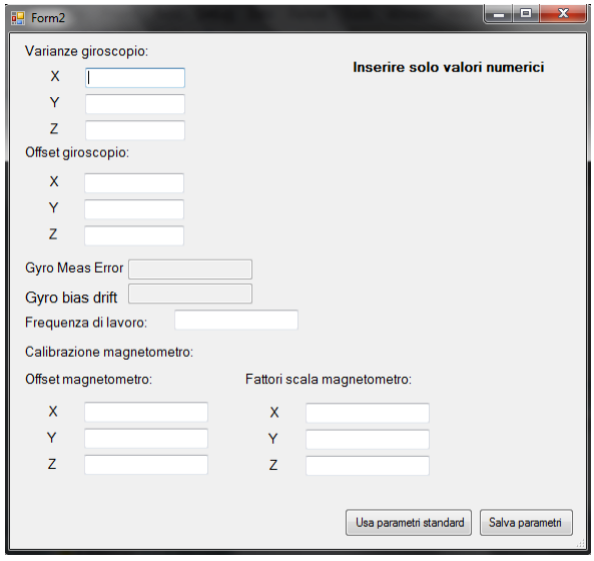


Figura 5.3: Form di inserimento dei parametri

Una volta che si preme su uno dei due pulsanti del form di inserimento di parametri si avvia il software vero e proprio di stima della posizione, in cui si visualizzerà un parallelepipedo che ruoterà in concomitanza con i movimenti da noi effettuati, come visibile in figura 5.4. Per gli algoritmi AHRS ed il filtro di Kalman è necessario una fase di assestamento, in particolare:

- Per l'AHRS tenere l'iNEMO fermo fincheè non si stabilizza anche il parallelepipedo sullo schermo.

- Per il filtro di Kalman invece effettuare una perturbazione muovendo la scheda. Dopo qualche movimento il parallelepipedo si assesterà ed il filtro sarà quindi a regime.

A questo punto per avere un funzionamento migliore si consiglia di "spostare virtualmente" il nord. Difatti il software è progettato per avere il nord rivolto verso la sinistra dello schermo, quindi a meno che non abbiate il nord rivolto verso la sinistra del vostro schermo i movimenti sembreranno sfasati. Per risolvere questo problema basterà premere il tasto sinistro del mouse quando i filtri si sono assestati; in questo modo si ridefinisce la posizione del nord nel software.
Si può anche premere il tasto destro del mouse per potere trasformare a tutto schermo la finestra. Chiudendo la finestra invece si tornerà alla finestra di avvio.

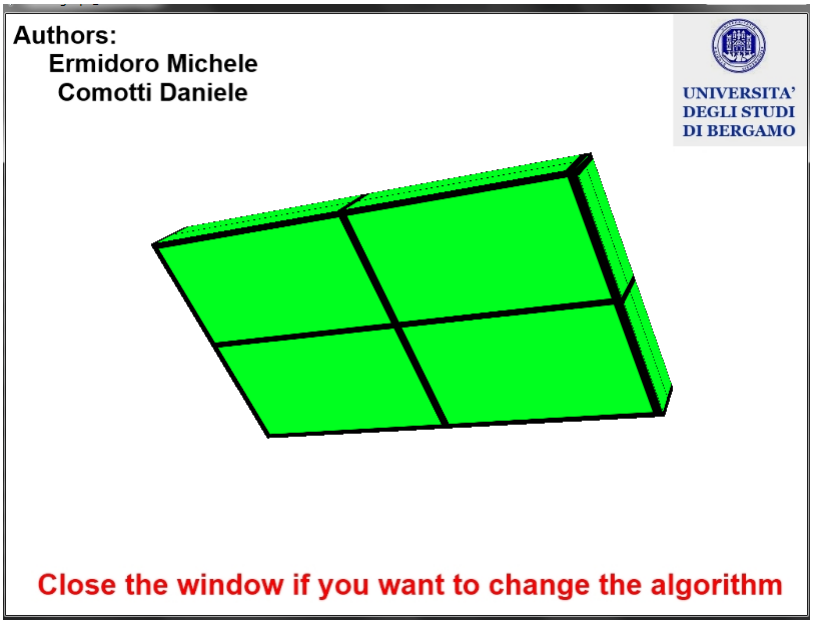


Figura 5.4: Screenshot di funzionamento del filtro di Kalman

Capitolo 6

Risultati e Prestazioni degli Algoritmi

In questo capitolo verranno esposti dei risultati di precisione e velocità dell'algoritmo che implementa il filtro di *Kalman* e del *Complementary Filter*. Per quanto riguarda il terzo algoritmo, dato che non è stato realizzato in *Matlab*, verrà affrontato un breve confronto tra lo stesso e gli altri due applicativi nel contesto del programma scritto in *C#*.

Le prove per il filtro di *Kalman* ed il *Complementary Filter* si sono svolte utilizzando un treppiedi da fotografo su cui abbiamo montato l'*iNemo* in maniera fissa, come mostrato in figura 6.1.

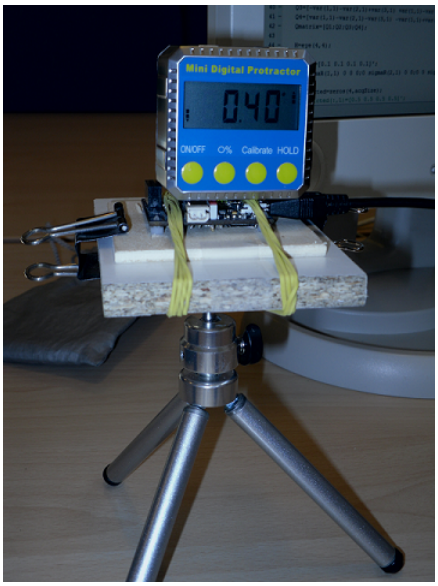


Figura 6.1: L'*iNemo* fissato al treppiedi durante i test di precisione.

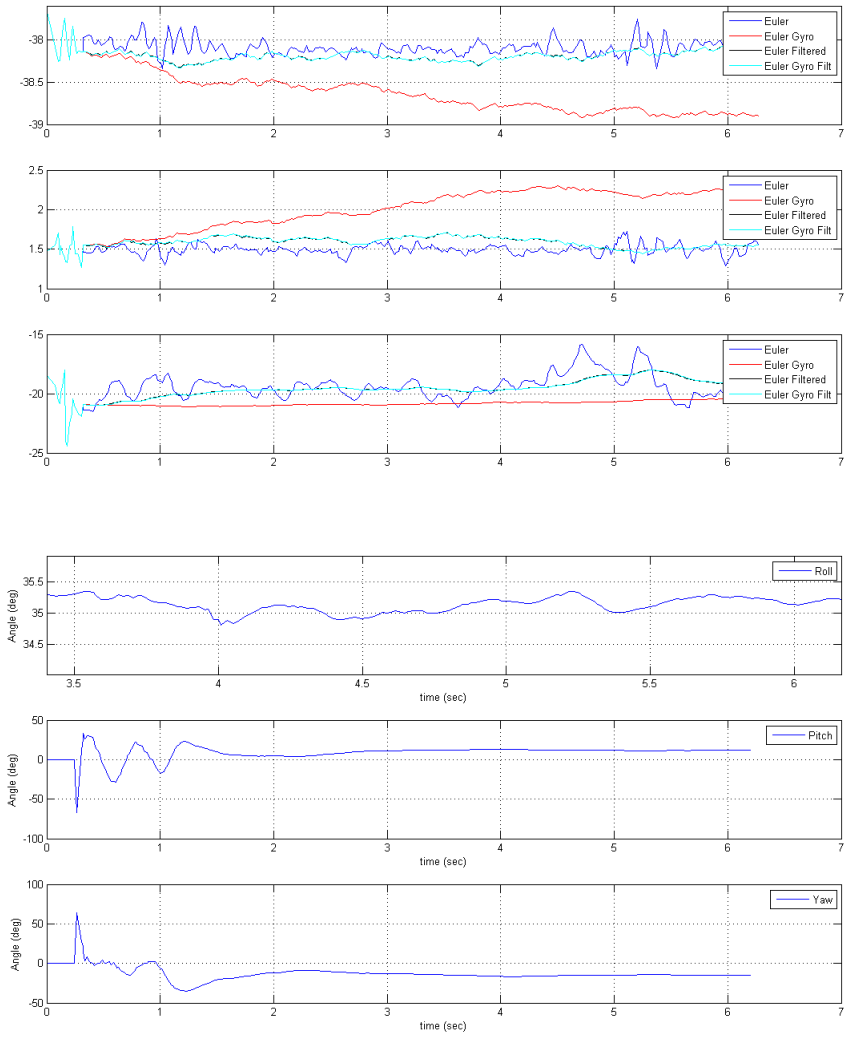
Abbiamo misurato, per ciascuna rotazione attorno ai tre assi, la precisione della stima degli angoli, munendoci di un inclinometro, e la dinamica di rotazione. Il motivo per cui ci siamo limitati a rotazioni su singoli assi è dovuto al fatto che altrimenti le misure sarebbero state più complicate e non

sarebbe bastato l'inclinometro. Per un funzionamento esaustivo si rimanda all'uso dell'applicazione scritta in *C#*.

Nella prossima sezione verrano mostrati i risultati dei test.

6.1 Complementary Filter e Quaternion Kalman Filter

La prima serie di misure che abbiamo effettuato ha riguardato la precisione in condizioni statiche di ciascuno dei due algoritmi; assicurandoci della correttezza delle rotazioni per mezzo dell'inclinometro, abbiamo eseguito gli script *Matlab* di ciascun algoritmo valutando la consistenza degli output. Di seguito sono mostrate le immagini relative.



Rollio, angolo pari a $36,3°$. La prima figura mostra i risultati ottenuti con il *Complementary Filter*, la seconda con il filtro di *Kalman*. Come si nota, il primo algoritmo presenta uno scostamento di circa $+2°$ dall'angolo reale, mentre il secondo di circa $-1°$.

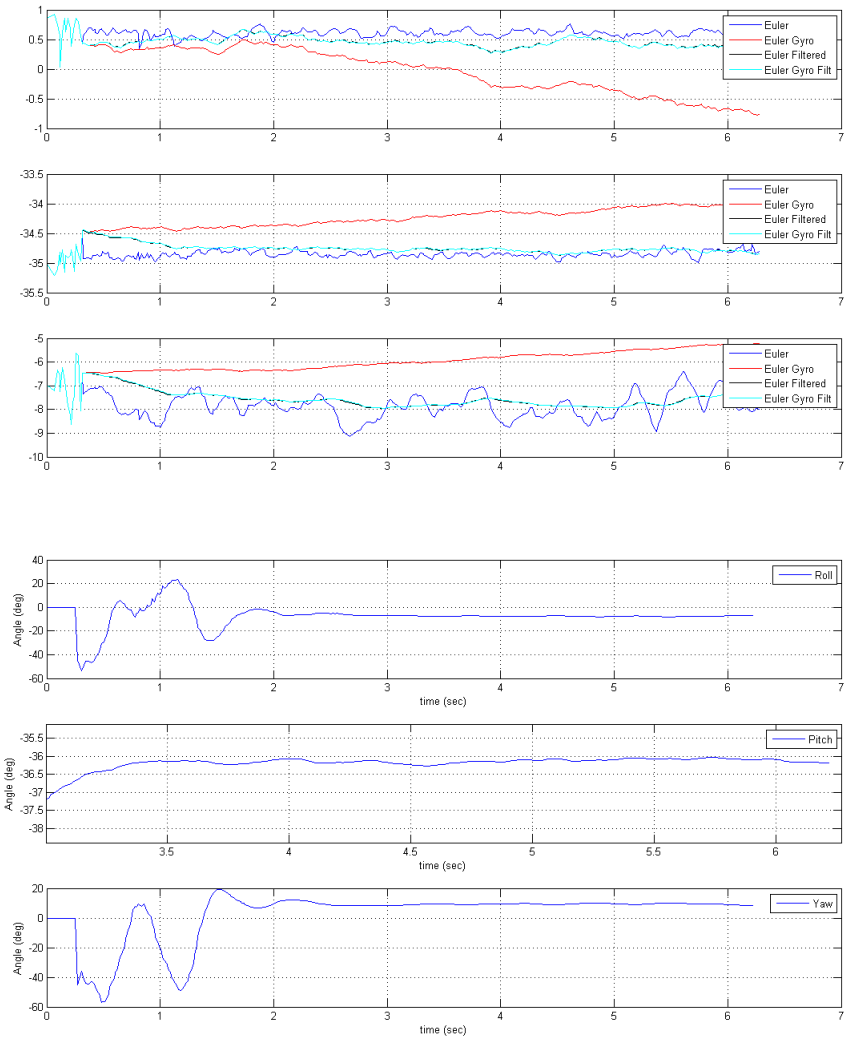


Figura 6.2: *Beccheggio*, angolo pari a $-36,1°$. La prima figura mostra i risultati ottenuti con il *Complementary Filter*, la seconda con il filtro di *Kalman*. Come si nota, il primo algoritmo presenta uno scostamento di circa $+1°$ dall'angolo reale, mentre il secondo risulta molto più preciso.

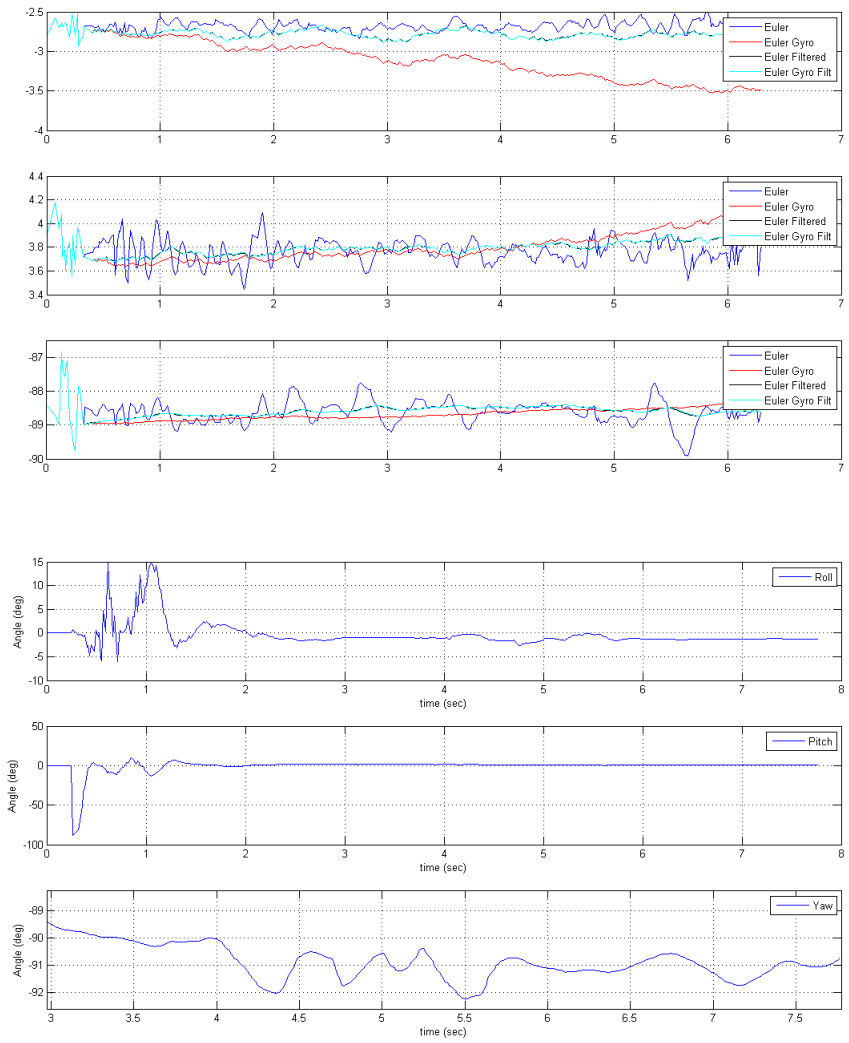


Figura 6.3: *Imbardata*, angolo pari a $-90°$. La prima figura mostra i risultati ottenuti con il *Complementary Filter*, la seconda con il filtro di *Kalman*. Come si nota, entrambi gli algoritmi si muovono nell'intorno dell'angolo reale.

In tabella 6.1 sono mostrati i dati riassuntivi degli esperimenti dinamici.

	Misura Reale [°]	Complem. Filter [°]	Kalman Filter [°]	Errore Complem. Filter [°]	Errore Kalman Filter [°]
Rollio	36.3	38.0	35.2	1.7	1.1
Beccheggio	-36.1	-34.9	-36.2	1.2	-0.1
Imbardata	-90.0	-88.6	-91.0	1.4	-1.0

Tabella 6.1: Dati riassuntivi degli esperimenti in condizioni statiche.

Per quanto riguarda i test dinamici, attraverso il treppiedi abbiamo variato l'orientamento dell'*iNemo* per ciascuno dei tre assi esclusivamente. Di seguito sono mostrate le immagini dell'esperimento.

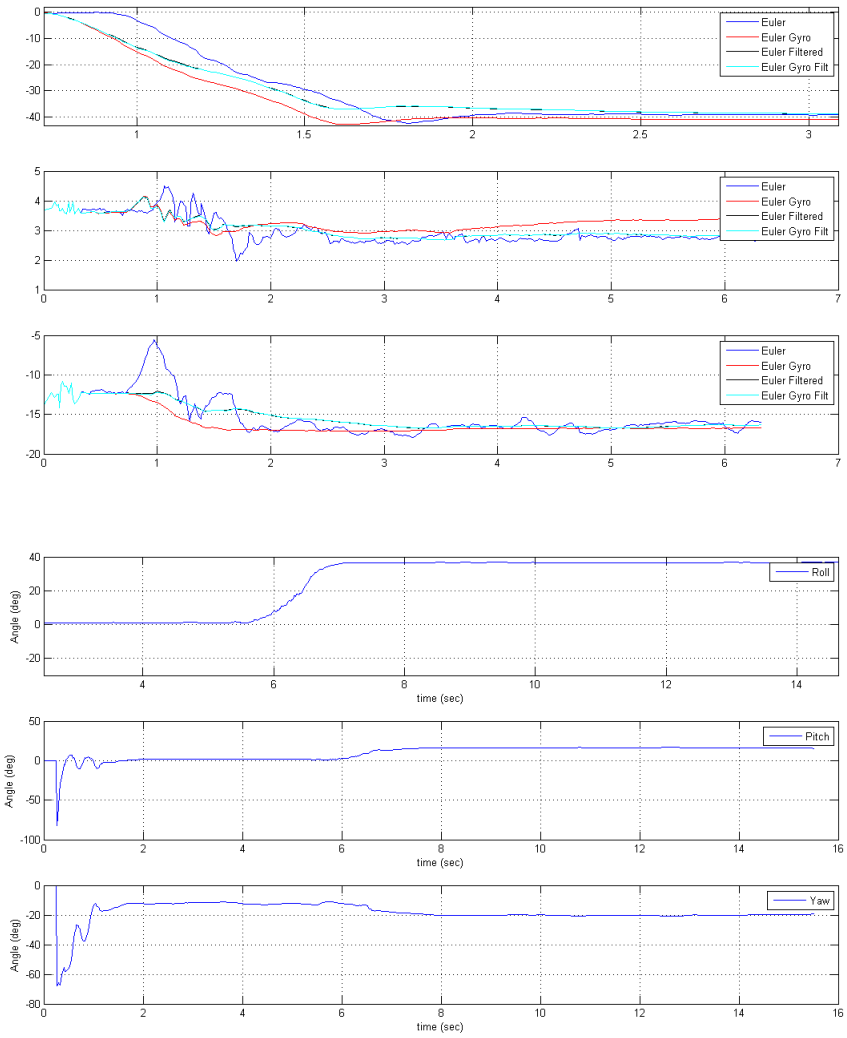


Figura 6.4: Rotazione attorno all'asse x di circa 37°.

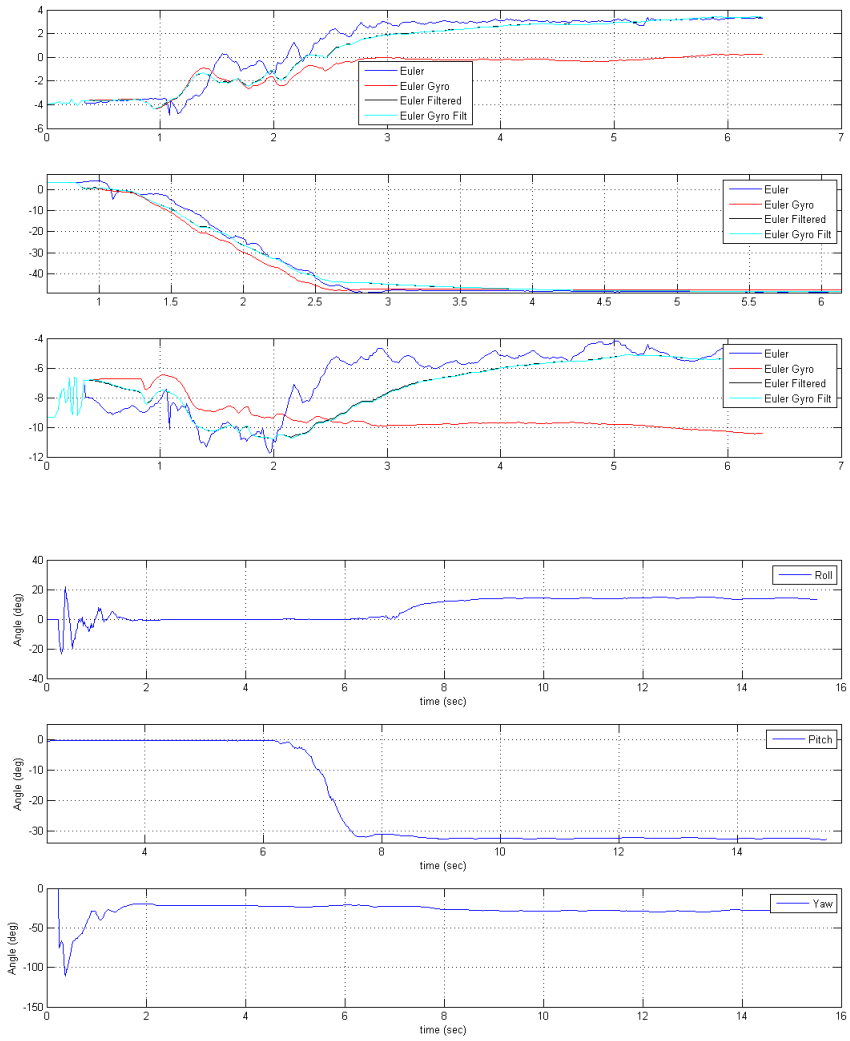


Figura 6.5: Rotazione attorno all'asse y. Gli angoli di rotazione non sono gli stessi per i due algoritmi, in quanto i fini dell'esperimento erano la dinamica e non la precisione relativa.

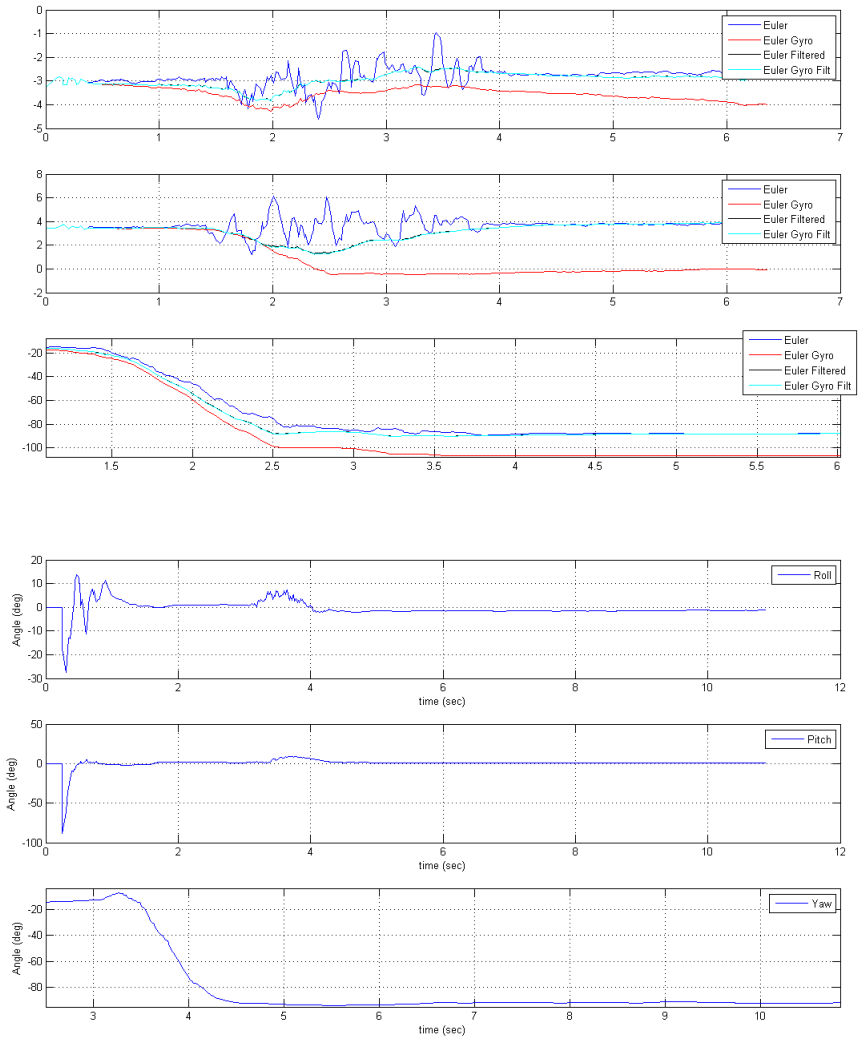


Figura 6.6: Rotazione attorno all'asse z di circa 90°.

Le immagini mostrate nelle precedenti figure sono puramente indicative, dato che ciascuna misura è stata affetta da errore sia umano che da imprecisioni legate agli assi non coinvolti nelle rotazioni.

Un'ultima analisi ha riguardato l'esecuzione parallela dei due algoritmi. Sono state effettate 3 rotazioni attorno ai 3 assi di riferimento. Nelle figure 6.7, 6.8 e 6.9 sono mostrati i risultati di ciascuna rotazione.

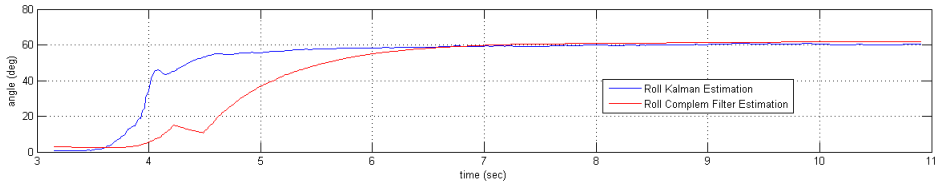


Figura 6.7: Rotazione di Rollio rilevata dal *Kalman Filter* e dal *Complementaray Filter*.

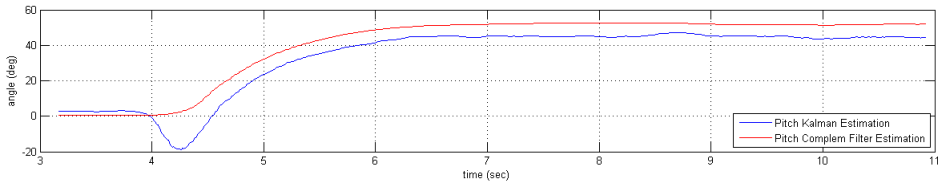


Figura 6.8: Rotazione di Beccheggio rilevata dal *Kalman Filter* e dal *Complementaray Filter*.

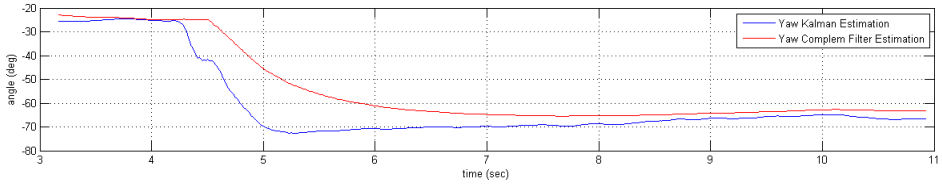


Figura 6.9: Rotazione di Imbardata rilevata dal *Kalman Filter* e dal *Complementaray Filter*.

In tutti i casi si nota come il comportamento a regime sia molti simile, come esposto precedentemente, mentre la dinamica del filtro di *Kalman* è più veloce con un tempo di assestamento inferiore rispetto al *Complementary Filter*. La resa dinamica del filtro di *Kalman* può essere aggiustata attraverso la matrice R, come si nota, la configurazione utilizzata durante i test ha prodotto, per quanto riguarda la rotazione di Pitch, una sottoelongazione iniziale. Infine, in figura 6.10 è mostrato un esperimento completo durato all'incirca 30 secondi.

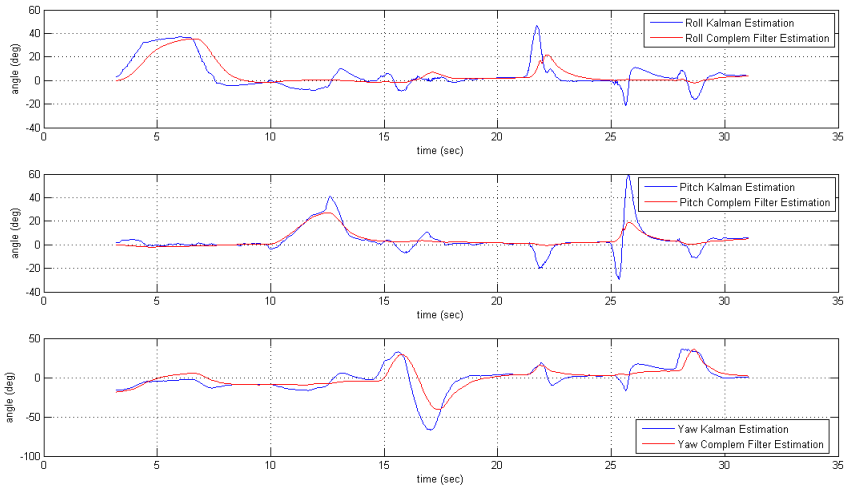


Figura 6.10: *Kalman Filter* e *Complementary Filter* a confronto. Si noti la risposta in condizioni lente sia molto simile tra i due, mentre, durante rotazioni veloci, il secondo algoritmo non segue del tutto l'andamento reale.

6.2 AHRS Quaternion

Per quanto riguarda l'algoritmo *AHRS Estimation*, questo non è stato possibile testarlo come fatto per *Kalman Filter* e *Complementary Filter* in quanto completamente sviluppato in *C#* (la classe è stata scaricata da internet ed adattata), saltando lo sviluppo in *Matlab*. Di conseguenza non è stato possibile ottenere dei grafici come fatto per i due algoritmi precedenti. Possiamo però analizzare il suo funzionamento nella visualizzazione grafica in *C#*. Distinguiamo due fasi principali:

- Statica: quando l'*iNemo* è fermo l'algoritmo identifica la posizione in cui si trova rispetto al nord terrestre, però richiede del tempo per raggiungere questa posizione. Il tempo è nell'ordine di $5 \div 6$ secondi. Il tempo maggiore lo richiede per l'assestamento iniziale, mentre per l'assestamento dopo un movimento il tempo si riduce a un paio di secondi. Le prestazioni in questo caso sono quindi peggiori rispetto al filtro di *Kalman* in cui la fase di assestamento iniziale può durare anche solamente un secondo se si perturba con un movimento la scheda, mentre per l'assestamento successivamente a rotazioni il tempo è pressochè nullo;
- Dinamica: durante i movimenti l'algoritmo segue abbastanza bene i movimenti della scheda. In questo caso le prestazioni sono paragonabili a quelle del filtro di *Kalman*, anche se il secondo è molto più stabile, le vibrazioni vengono attenuate dal filtraggio.

Conclusioni

In questa relazione è stata descritta la nostra esperienza circa l'attività di stima dell'orientamento di una piattaforma inerziale. Il lavoro svolto ha richiesto non poco tempo, soprattutto per quanto riguarda il filtro di *Kalman*, e richiederebbe ulteriore sforzo per ottenere una comprensione completa di quanto trattato. Quella acquisita è una conoscenza generale, una "infarinatura", circa la panoramica di soluzioni relative al tema trattato. Tuttavia, il progetto ci ha coinvolto attivamente e ci ha molto appassionato, a tal punto che ci siamo posti degli obiettivi di sviluppi futuri:

- Realizzazione un *Complementary Filter* che stimi le rotazioni in modo completo sull'intervallo $(180, -180]$;
- Scrittura in *Matlab* dell'algoritmo *AHRS Quaternion Estimation*;
- Raffinamento del filtro di *Kalman* ed estensione dello stato includendo le componenti di velocità del giroscopio;
- Miglioramento dell'applicativo scritto in *C#* ed una migliore ingegnerizzazione del codice relativo.

Bibliografia

[1] João Luís Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee, and Michael J. Zyda, "An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors".

[2] Songlai Han, Jinling Wang, "A novel method to integrate IMU and magnetometers in attitude and heading reference systems".

[3] Widyawardana Adiprawita, "Adang Suwandi Ahmad, Jaka Sembiring, "Development of AHRS for Autonomous UAV".

[4] Demoz Gebre-Egziabher, Gabriel H. Elkaim, J. D. Powell, Bradford W. Parkinson, "A gyro-free quaternion based attitude determination system suitable for implementation using low cost sensors".

[5] Shane Colton, "The Balance Filter A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform".

[6] ST Microelectronics, "STEVAL-MKI062V2 User Manuale".

[7] ST Microelectronics, "STEVAL-MKI062V2 Data brief".

[8] ST Microelectronics, "LSM303DLH Datasheet".

[9] ST Microelectronics, "LPR430AL Datasheet".

[10] ST Microelectronics, "LY330ALH Datasheet".

[11] ST Microelectronics, "AN3192 Application Note".

[12] Verena Elisabeth Kremer, "Quaternions and SLERP".

[13] Sebastian O.H. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays".

[14] DEMOZ GEBRE-EGZIABHER, ROGER C. HAYWARD , J. DAVID POWELL, "Design of Multi-Sensor Attitude Determination Systems".

[15] ST Microelectronics, "AN3182 Application Note".

[16] Michael J. Caruso, "Applications of Magnetoresistive Sensors in Navigation Systems".

Appendice

Gradient Descent Method

Nell'ambito di questo progetto abbiamo utilizzato l'algoritmo di ottimizzazione del primo ordine *Gradient Descent*, che permette di determinare un minimo locale di una funzione. Come si può intuire dal nome, tale algoritmo fa uso del gradiente della funzione da minimizzare procedendo, dato un punto iniziale, il punto di minimo, iterando per passi negativi proporzionali al gradiente calcolato nel punto corrente. In generale, sia $\underline{f}(\underline{x}) : \mathbb{R}^n \to \mathbb{R}^m$ definita e differenziabile in un intorno di a, allora la discesa più veloce è quella che da a si sposta nella direzione negativa del gradiente di $\underline{f}(a)$. Si ha quindi che il nuovo punto b è dato da:

$$b = a - \mu \cdot \nabla \underline{f}(a) \tag{6.1}$$

con $\mu > 0$ e piccolo abbastanza. Agendo ricorsivamente, partendo da $\underline{x_0}$, con la sequenza $\underline{x_0}$, $\underline{x_1}$, $\underline{x_2}$, ... si ha che:

$$\underline{x_{n+1}} = \underline{x_n} - \mu \cdot \nabla \underline{f}(\underline{x_n}) \, , \, n \geq 0 \to \underline{f}(\underline{x_0}) \geq \underline{f}(\underline{x_1}) \geq \underline{f}(\underline{x_2}) \dots \tag{6.2}$$

per cui la sequenza $\underline{x_0}$, $\underline{x_1}$, $\underline{x_2}$, ..., $\underline{x_n}$ converge al desiderato minimo locale.

Il metodo del *Gradient Descent* l'abbiamo applicato nel contesto dell'algoritmo di stima basato sul *Kalman Filter* per determinare i migliori quaternioni caratterizzanti le osservazioni. Dato che gli angoli di *Eulero* soffrono di problemi, in particolare quando un asse è parallelo al campo (gravitazionale o magnetico), l'angolo di rotazione attorno a tale asse, necessario per la rappresentazione basata su quaternioni, può essere determinato formulando un problema di ottimizzazione dove l'orientamento dell'*iNemo*, descritto attraverso un quaternione $\hat{q}_{t-1}$, è tale che effettuando l'operazione di rotazione sulla direzione del campo rispetto alla terna fissa ${}^E\underline{d}$, allinea quest'ultimo con la misura rispetto alla terna corrente ottenuta dall'*iNemo* ${}^S\underline{s}$. Si può quindi determinare il nuovo quaternione q_t come soluzione del seguente problema di minimo:

$$min_{\,\hat{q} \in \mathbb{R}^4} \, \underline{f}(\hat{q}, {}^E\underline{d}, {}^S\underline{s}) \tag{6.3}$$

dove $\underline{f}(\hat{q}_{t-1}, {}^E\underline{d}, {}^S\underline{s})$ è appunto data da:

$$\underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s}) = \hat{q}_{t-1} * \otimes {}^{E}\underline{d} \otimes \hat{q}_{t-1} - {}^{S}\,\underline{s} \tag{6.4}$$

dove $\hat{q}_{t-1}*$ è il coniugato di $\hat{q}_{t-1}$.
In questo modo viene determinato il quaternione che, sulla base delle osservazioni, minimizza l'errore dovuto alla rotazione del campo nella terna fissa per mezzo dello stesso. Nell'articolo [13] è mostrato come calcolare il migliore quaternione associato alle osservazioni, e per farlo viene utilizzato il metodo del *Gradient Descent*. Definendo ${}^{E}\underline{d} = \begin{bmatrix} 0 & d_x & d_y & d_z \end{bmatrix}$ e ${}^{S}\underline{s} = \begin{bmatrix} 0 & s_x & s_y & s_z \end{bmatrix}$, si ha che:

$$q_t = \hat{q}_{t-1} - \mu \cdot \frac{\nabla \underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s})}{\left\| \nabla \underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s}) \right\|} \tag{6.5}$$

dove il gradiente si ottiene, dato che la funzione in questione è caratterizzata da più variabili, moltiplicando la *Jacobiana* per la funzione stessa, ovvero:

$$\nabla \underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s}) = J(\hat{q}_{t-1},\, {}^{E}\underline{d})^T \cdot \underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s})$$

Per ulteriori dettagli circa il calcolo della *Jacobiana* e della funzione $\underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s})$ si rimanda all'articolo , in quanto, per evitare di dilungarsi, sono state omesse in questa relazione.
Dato che le informazioni sensoriali che definiscono le osservazioni provengono da due sensori, il magnetometro e l'accelerometro, sulla base di quanto detto precedentemente, si hanno due coppie di *Jacobiane* e di funzioni da minimizzare $\underline{f}(\hat{q}_{t-1},\, {}^{E}\underline{d},\, {}^{S}\underline{s})$, $\underline{f_g}(\hat{q}_{t-1},\, {}^{S}\underline{a})$ e $J_g(\hat{q}_{t-1})$ per l'accelerometro e $\underline{f_b}(\hat{q}_{t-1},\, {}^{E}\underline{b},\, {}^{S}\underline{m})$ e $J_b(\hat{q}_{t-1},\, {}^{E}\underline{b})$ per il magnetometro. L'algoritmo finale raggruppa queste ultime nel seguente modo:

$$\underline{f_{g,b}}(\hat{q}_{t-1},\, {}^{S}\underline{a}\, {}^{E}\underline{b},\, {}^{S}\underline{m}) = \begin{bmatrix} \underline{f_g}(\hat{q}_{t-1},\, {}^{S}\underline{a}) \\ \underline{f_b}(\hat{q}_{t-1},\, {}^{E}\underline{b},\, {}^{S}\underline{m}) \end{bmatrix}$$

$$J_{g,b}(\hat{q}_{t-1},\, {}^{E}\underline{b}) = \begin{bmatrix} J_g(\hat{q}_{t-1}) \\ J_b(\hat{q}_{t-1},\, {}^{E}\underline{b}) \end{bmatrix}$$

$$\nabla \underline{f_{g,b}}(\hat{q}_{t-1},\, {}^{S}\underline{a}\, {}^{E}\underline{b},\, {}^{S}\underline{m}) = J_{g,b}(\hat{q}_{t-1},\, {}^{E}\underline{b})^T \cdot \underline{f_{g,b}}(\hat{q}_{t-1},\, {}^{S}\underline{a}\, {}^{E}\underline{b},\, {}^{S}\underline{m})$$

$$q_t = \hat{q}_{t-1} - \mu \cdot \frac{\nabla \underline{f_{g,b}}(\hat{q}_{t-1},\, {}^{S}\underline{a}\, {}^{E}\underline{b},\, {}^{S}\underline{m})}{\left\| \nabla \underline{f_{g,b}}(\hat{q}_{t-1},\, {}^{S}\underline{a}\, {}^{E}\underline{b},\, {}^{S}\underline{m}) \right\|} \tag{6.6}$$

Nel calcolo di $\underline{f_g}(\hat{q}_{t-1},\, {}^{S}\underline{a})$ non viene utilizzato il riferimento rispetto alla terna fissa in quanto questo viene assunto pari a ${}^{E}\underline{a} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$, ovvero con l'*iNemo* avente l'asse z diretto verso l'alto e parallelo a $\underline{G}$.