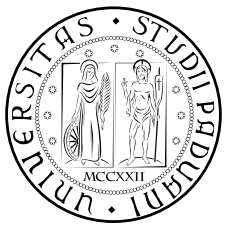


UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

STABILIZZAZIONE DELL'IMMAGINE IN BRANDEGGI MOBILI PER VIDEOSORVEGLIANZA: APPLICAZIONE DI TECNICHE BASATE SULL'ELABORAZIONE D'IMMAGINE

Relatore: Ch.mo Prof. Roberto Oboe

Correlatore: Dott. Riccardo Antonello

Laureando: Simone Viola
626213-IMC

ANNO ACCADEMICO: 2011-12

SOMMARIO

L'obiettivo che ci si è posti in questo lavoro di tesi, è l'implementazione di un controllo per la stabilizzazione dell'immagine acquisita mediante un brandeggio mobile per videosorveglianza in applicazioni offshore su boe e navi.
Tale stabilizzazione è stata ottenuta utilizzando misure inerziali fornite da una IMU (*Inertial Measurement Unit*) e realizzando, dapprima, dei loop di controllo per gli attuatori presenti all'interno del brandeggio e, successivamente, una post-elaborazione a livello video. In particolare attraverso la post-elaborazione video è stato possibile stimare la bontà del controllo implementato.
In aggiunta è stato proposto un metodo per la stabilizzazione video di movimenti di roll non compensabili elettromeccanicamente dal brandeggio.
Obbiettivo non secondario di tale elaborato è stato, inoltre, verificare la fattibilità di questo nuovo sistema senza dover richiedere sostanziali modifiche strutturali e nei componenti del bandeggio stesso.

RINGRAZIAMENTI

Ringrazio il Professor Roberto Oboe e il Dottor Riccardo Antonello per la disponibilità e l'aiuto datami per svolgere questa tesi.
Desidero esprimere la mia riconoscenza anche a tutti i Professori del Corso di Laurea in Ingegneria Meccatronica, per avermi trasmesso, con professionalità e dedizione le loro conoscenze.
Desidero inoltre ringraziare la mia famiglia per essermi stati sempre vicino, umanamente ed economicamente, durante tutti i miei studi.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

1 INTRODUZIONE

1.1 STABILIZZAZIONE PER PIATTAFORME INERZIALI

La stabilizzazione per piattaforme inerziali ISPs (*Inertially Stabilized Platforms*) viene utilizzata ormai da cento anni per stabilizzare e/o mantenere indirizzati una vasta gamma di sensori come fotocamere, videocamere, telescopi, equipaggiamenti di armamento, ecc.... Tali sistemi vengono montati in ogni tipo di veicolo mobile, dai satelliti ai sottomarini, dagli aerei ai veicoli terrestri. Tutto questo per realizzare innumerevoli applicazioni in ambiti che vanno da quello scientifico e commerciale fino quello militare. Queste applicazioni includono videosorveglianza, inseguimento di target, comunicazioni, telescopi astronomici, stabilizzazione di telecamere, missili teleguidati, controllo di sistemi di puntamento [4].
Le configurazioni per la ISP elettromeccanica sono tanto diverse quanto lo sono le applicazioni per qui sono designate. Solitamente questi sistemi, chiamati *gimbal*, sono costituiti da un'insieme di strutture, ingranaggi, e attuatori in cui sono spesso installati molti tipi di sensori, quali giroscopi, accelerometri, magnetometri ed unità GPS. Originariamente, con il termine gimbal si identificava una struttura costituita da cerchi concentrici sostenuti da cuscinetti, mentre oggigiorno questo termine può riferirsi a diversi tipi di meccanismi e dispositivi che permettono la stabilizzazione della linea di vista (*line of sight* LOS) del carico posto su di essi, ruotandolo e controllandolo opportunamente. Attraverso la lettura dei sensori installati nel gimbal si può misurare l'orientazione e la posizione della struttura. Successivamente, basandosi su queste misure, è possibile realizzare qualsiasi controllo di stabilizzazione o di puntamento. Il gimbal è progettato, tipicamente, per permettere il puntamento e la stabilizzazione rispetto due o più assi e molte applicazioni richiedono almeno due gimbal ortogonali tra loro, per ottenere alcuni gradi di libertà addizionali o per ottenere un miglior isolamento rispetto al moto del veicolo o della struttura su cui sono montati.
Anche se i requisiti, per i vari sistemi ISP, variano molto a seconda delle applicazioni, quest'ultimi hanno come obiettivo comune la stabilizzazione o il controllo del LOS di un'oggetto rispetto ad un'altro o ad uno spazio inerziale. Il LOS, per esempio, può essere il bersaglio di un fascio luminoso (laser) o di un'arma, il centro del campo visivo di un telescopio o la direzione in cui è puntato un sensore.

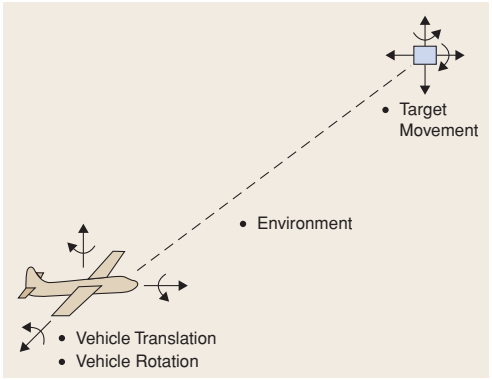


Figura 1: Scenario tipo per applicazioni di ISP

Uno scenario tipico, mostrato in fig. 1, ci permette di capire l'effettiva necessità di tali applicazioni ISP. Come si osserva, ad eccezione di alcune particolari applicazioni scientifiche o di navigazione, il movimento che deve essere controllato e compensato è il movimento tra due oggetti, che spesso sono in movimento uno rispetto all'altro.
Le specifiche, per quanto riguarda questo tipo di applicazioni, si concentrano sostanzialmente intorno ai seguenti due aspetti del controllo:

- errore di inseguimento del target, ossia l'errore che si commette cercando di mantenere puntato il LOS su un target. Se tale errore risulta essere troppo elevato è possibile che il target esca dal campo visivo del sensore;
- jitter, ossia un'oscillazione indesiderata del sensore che può compromettere la qualità dei dati raccolti da quest'ultimo. L'oscillazione può essere causata da vibrazioni generate dal mezzo in cui sono montati i sensori o da disturbi ambientali: terreni sconnessi, vento, onde e quant'altro.

Entrambi questi aspetti sono indesiderati e per questo l'obbiettivo comune dei sistemi ISP è di andare a compensarli e di ridurne gli effetti il più possibile.

1.2 STABILIZZAZIONE DELLE IMMAGINI

La stabilizzazione dell'immagine (*Image Stabilization* IS) è una famiglia di tecniche utilizzate per ridurre le vibrazioni e le sfocature dovute al movimento della fotocamera durante l'esposizione o della videocamera durante la registrazione di un video. In particolare, si cerca di compensare eventuali movimenti di pan e tilt[1] di una fotocamera o di una videocamera o di altro dispositivi come binocoli e telescopi astronomici. Tale esigenza è facilmente comprensibile pensando ai

1 in angoli aeronautici corrispondono rispettivamente ad imbardata/yaw e beccheggio/pitch

problemi che possono insorgere nelle principali applicazioni di acquisizione immagini. In ambito fotografico, per effetto della continua evoluzione, si sono ottenuti obbiettivi con lunghezze focali e tempi di esposizione sempre maggiori e pertanto le vibrazioni della fotocamera sono una problematica sempre più attuale. Con le videocamere, le vibrazioni del corpo macchina causano un jitter visibile frame-to-frame, che compromette la qualità del video registrato. In astronomia, il problema della lent-shake (vibrazione delle lenti) si va ad aggiungere alle variazioni temporali dell'atmosfera e ciò determina posizioni apparenti degli oggetti osservati.

Una prima distinzione che si può effettuare, per quel che riguarda queste tecniche di stabilizzazione, è la seguente:

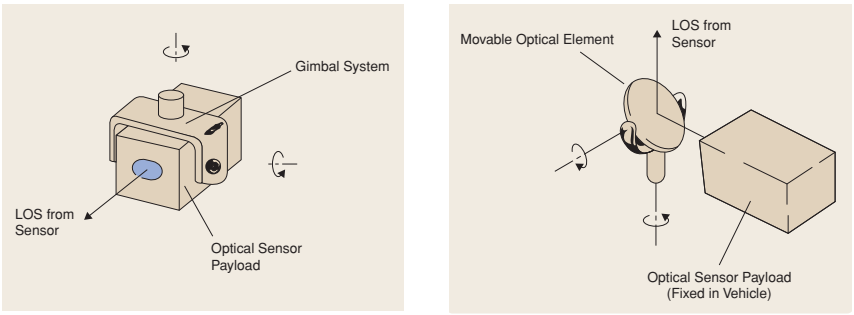
- stabilizzazione elettromeccanica
- stabilizzazione digitale delle immagini

1.2.1 *Stabilizzazione elettromeccanica delle immagini*

I sensori ottici per l'acquisizione di immagini, come fotocamere, videocamere o videocamere a infrarosso, e molti altri, raccolgono informazioni su un target preciso oppure su una regione in uno spazio di interesse. Come già detto in precedenza, al fine di raccogliere tali informazioni, è necessario andare a controllare il puntamento della LOS del sensore ottico e compensare tutte le eventuali vibrazioni esistenti. Controllare tale orientamento con una ISP può essere la risposta a questo problema.
Una tipica ISP, in accordo con [6], è composta da:

- un sistema elettromeccanico, solitamente attuato, capace di fornire un'interfaccia tra la struttura su cui è montata la ISP e il sensore ottico montato su di essa;
- un sistema di controllo il cui obbiettivo primario è quello di mantenere all'interno del campo visivo del sensore ottico il target desiderato e compensare eventuali vibrazioni o movimenti della struttura su cui è montata la ISP stessa;
- un equipaggiamento ausiliario che, a seconda delle applicazioni, può misurare la posizione e l'orientamento del target relativamente al sistema di riferimento della ISP. Inoltre, se la struttura su cui la ISP viene montata è mobile, cosa che si verifica nella maggior parte dei casi, sorge la necessità di conoscere l'orientazione e posizione della ISP stessa. Per misurare questa orientazione si fa spesso uso di un'unità inerziale di misurazione detta IMU (*Inertial Measurement Unit*) e tali dati vengono poi utilizzati dal sistema di controllo.

Per quanto riguarda il sistema elettromeccanico è tipicamente composto da un gimbal attuato che ruota. La stabilizzazione, secondo [6], può avvenire usando sostanzialmente due approcci: platform stabilization e steering stabilization.
Nella *platform stabilization*, come mostrato in fig. 2a, il sensore ottico



(a) platform stabilization

(b) steering stabilization

Figura 2: Esempio di platform stabilization e di steering stabilization

viene montato all'interno del gimbal e viene fatto ruotare opportunamente per rendere il LOS del sensore immune ai movimenti della struttura e per permettere l'inseguimento del target. Il gimbal può essere movimentato con diverse tecniche. Una prima tecnica consiste nell'utilizzo di motori direttamente installati sui singoli assi, mentre una seconda consiste nell'uso di motori collegati ai giunti con trasmissioni meccaniche come ingranaggi, cinghie o catene. L'orientazione del LOS è determinata dallo spostamento angolare del gimbal stesso. Se la struttura ospitante il gimbal si muove o vibra quest'ultimo, se funzionante, ruota in senso opposto in modo da rendere l'orientazione del LOS fissa rispetto ad uno spazio inerziale. In questo caso quindi, il movimento del gimbal determina direttamente la direzione del LOS e il jitter dell'immagine.
Nella *steering stabilization* per cambiare l'orientazione del LOS viene mosso un elemento ottico mobile, come ad esempio uno specchio o un prisma o una lente, ed il sensore ottico, invece, risulta fisso con la struttura. I questo modo, come si può osservare in fig. 2b, l'orientazione del LOS viene determinata indirettamente attraverso il controllo di solo alcune delle ottiche presenti. La relazione geometrica tra l'elemento ottico mobile ed il sensore ottico determinerà la regione dello spazio di interesse vista dal sensore. Quando il target puntato si muove, tale relazione geometrica può essere modificata per inseguirlo. Infine, se la struttura ospitante il sensore ottico si muove o vibra, lo specchio può essere controllato in modo tale da compensare questi spostamenti e renderli invisibili al sensore ottico. In questa configurazione, la posizione nominale dello specchio indirizza il LOS nella direzione desiderata, e movimenti addizionali permettono una compensazione del jitter.

1.2.2 *Stabilizzazione digitale delle immagini*

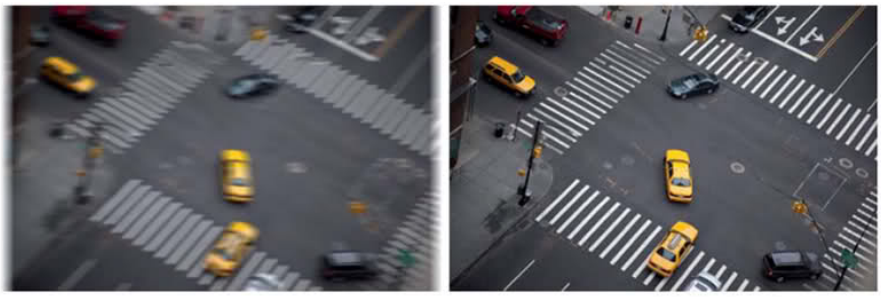


Figura 3: Esempio di stabilizzazione digitale dell'immagine

La stabilizzazione digitale delle immagini, (*Digital Image Stabiliztion* DIS), ha un approccio totalmente differente dalla precedente. Essa non necessità di strutture meccaniche per la compensazione del movimento ma la stabilizzazione dell'immagine avviene esclusivamente via software, senza coinvolgere l'hardware. In aggiunta la DIS può essere utilizzata sia in algoritmi di post-processing che di real-time processing ed è una tecnica semplice da implementare ed a basso costo [14]. Per questi motivi è sempre più utilizzata in tutti i recenti prodotti elettronici commerciali, come fotocamere digitali, videocamere digitali, CCD sensor, smartphone, etc. . . dove aspetti di costo e dimensioni sono determinanti.
La stabilizzazione digitale d'immagine si divide sostanzialmente in due passi: la stima del movimento e la sua compensazione. Una delle tecniche più comuni per stimare il movimento dell'immagine è la *feature detection*. Tale tecnica prevede la rilevazione di figure o forme caratteristiche nell'immagine e il tracking di quest'ultime frame by frame. Per implementare questa tecnica è in corso un continuo sviluppo di nuovi algoritmi con l'obbiettivo di ridurne l'onerosità computazionale, e migliorarne la precisione nel tracking.
Lo svantaggio maggiore dato dall'utilizzo di tale metodo è che risulta molto influenzato dalla qualità dell'immagine e dalla possibilità di distinguerne forme e contorni. Ciò dipende dal tipo di ambiente inquadrato, dalle condizioni di luminosità e atmosferiche [1].

1.3 VIDEOSORVEGLIANZA

Un sistema di videosorveglianza viene definito come uno strumento tecnologico che aiuta l'uomo ad estendere le sue capacità percettive e di ragionamento in situazioni che prevedono il monitoraggio di ambienti. L'uomo può percepire e processare con un tempo definito solo una parte limitata di informazioni ed eventi che riguardano un'area ristretta dello spazio. Le situazioni che possono essere analizzate sono limitate rispetto alla totalità degli eventi che avvengono in istanti

temporali diversi [12]. Le nuove tecnologie di videosorveglianza vengono in aiuto per compensare queste limitazioni dei sensi percettivi dell'uomo. Queste trovano impiego in molte applicazioni tra cui:

- sorveglianza di zone commerciali, militari e private al fine allertare i servizi di sicurezza a causa di rapine in corso o della presenza di persone sospette;
- misura dell'intensità di traffico e individuazione di incidenti;
- rilevazione della densità di persone nei centri commerciali e nei parchi di divertimento;
- Controllo accessi in aree speciali.

Gli ambiti e le tecniche di videosorveglianza possono essere organizzate in tre principali categorie:

1. Identificazione di un particolare target ed il suo inseguimento anche in ambienti dinamici.
2. Analisi del movimento di oggetti e di persone ed identificazione del moto, cioè tecniche che consentono di descrivere accuratamente l'andatura e la postura di una persona.
3. Analisi attiva. Si vuole fare in modo che il sistema di videosorveglianza sia in grado di rivelare e classificare autonomamente gli eventi e le interazioni tra questi.

Le applicazioni future sono principalmente orientate nell'evitare le occlusioni, la fusione di tecniche di tracking 2-D e 3-D, elaborazione di modelli tridimensionali di persone e veicoli, rilevazione di anomalie e predizione di comportamenti, riconoscimento semantico, sorveglianza da remoto [5].

1.3.1 *Brandeggio*

Per brandeggio, in senso generico, s'intende un supporto che può ruotare contemporaneamente sia un asse verticale, sia lungo un asse orizzontale. Esso si lega al concetto di videosorveglianza quando si ha l'esigenza di muovere a distanza una telecamera in tutte le orientazioni possibili. Con il termine brandeggio per videosorveglianza, in campo industriale, s'intende l'insieme di motori, custodie e il sistema di controllo integrato. Quest'ultimo è necessario per i movimenti di pan e tilt[2], e per la gestione delle ottiche motorizzate (zoom, focus, iris), della telemetria e del segnale video.

2 rispettivamente rotazione attorno all'asse verticale ed orizzontale

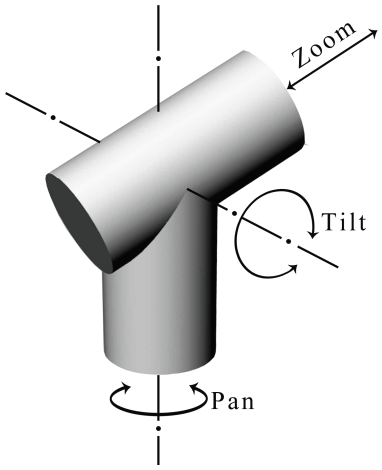


Figura 4: Rappresentazione delle movimentazioni del brandeggio

Questi sistemi prendono anche il nome di telecamere PZT, ossia telecamere in grado di eseguire movimentazioni di PAN, TILT e ZOOM.

1.4 SORVEGLIANZA MARITTIMA

Figura 5: Tipica boa con apparecchiature scientifiche a bordo

Mentre la sorveglianza terrestre e aerea da sempre ha attirato un notevole interesse e un conseguente notevole sviluppo, in ambito marittimo questo tema non ha richiamato un'attenzione significativa. Tuttavia, in questo ambito vi sono molti interessanti aspetti e problematiche riguardanti la stabilizzazione del moto. Le principali applicazioni interessate sono boe per videosorveglianza in zone portuali e aree marine di interesse, boe con apparecchiature scientifiche a bordo, videosorveglianza per navi etc. . . .
Questi sensori galleggianti montati, su boe o navi, sono influenzati da un continuo movimento dovuto alla superficie dell'acqua. Le onde possono avere forme e dimensioni che variano anche molto ra-

pidamente nel tempo. É quindi necessario in alcuni casi montare tali sensori sopra a delle strutture ISP al fine di compensare il moto ondoso. Per quanto riguarda applicazioni con sensori ottici è necessario affrontare altre problematiche che non si presentano in altri ambiti. L'illuminazione marina tipicamente genera un contrasto dell'immagine maggiore rispetto ad immagini terrestri. Ciò è dovuto alla riflessione diretta della luce solare che causa, per sensori ottici con limitati range dinamici, una notevole perdita di dettagli nell'immagine. I rapidi cambiamenti di luce possono essere contrastati attraverso un auto-iris del sensore, ma questo causa dei cambiamenti sull'aspetto dell'oggetto visualizzato e sui suoi dettagli. Infine gli spruzzi dell'acqua possono lasciare delle gocce e depositi di sale nella custodia di protezione dell'ottica, causando distorsioni e confusione nell'immagine.

1.5 DESCRIZIONE DEL PROBLEMA

Dopo questa breve panoramica si vuole ora descrivere lo scopo di questo elaborato cioè la realizzazione di un sistema di controllo per il brandeggio Ulisse Compact finalizzato alla stabilizzazione della telecamera su di esso montata per applicazioni marittime off-shore. A causa del moto ondoso, la boa o la nave su cui il brandeggio è fissato può oscillare compromettendo la qualità del video e il mantenimento del target desiderato all'interno del campo visivo della telecamera. Tale risultato si vuole ottenere senza apportare sostanziali modifiche all'hardware in possesso valutandone quindi la fattibilità ed i limiti imposti dal sistema complessivo in termini di prestazioni.

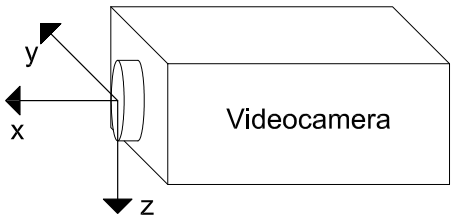


Figura 6: Rappresentazione del sistema di riferimento di una videocamera

Un immagine viene descritta come un insieme di celle, dette pixel, di coordinate (m, n). Il sistema di riferimento di una videocamera lo si può rappresentare come in fig. 6. Utilizzando la rappresentazione angolare di Eulero, è possibile definire roll, pitch, e yaw come gli angoli associati, secondo la regola della mano destra, rispettivamente agli assi x, y, e z. Mentre una rotazione della videocamera di roll si traduce in una rotazione dell'immagine, una rotazione di pitch o yaw si traduce in una traslazione del target lungo le coordinate m o n sul piano immagine. Nel caso molto comune di puntamento di un target lontano, i movimenti che influenzano notevolmente la qualità dell'immagini sono ancora le rotazioni di yaw e pitch, mentre even-

tuali traslazioni della videocamera si possono considerare trascurabili [15] (come si può notare dallo schema semplificativo in fig. 7).

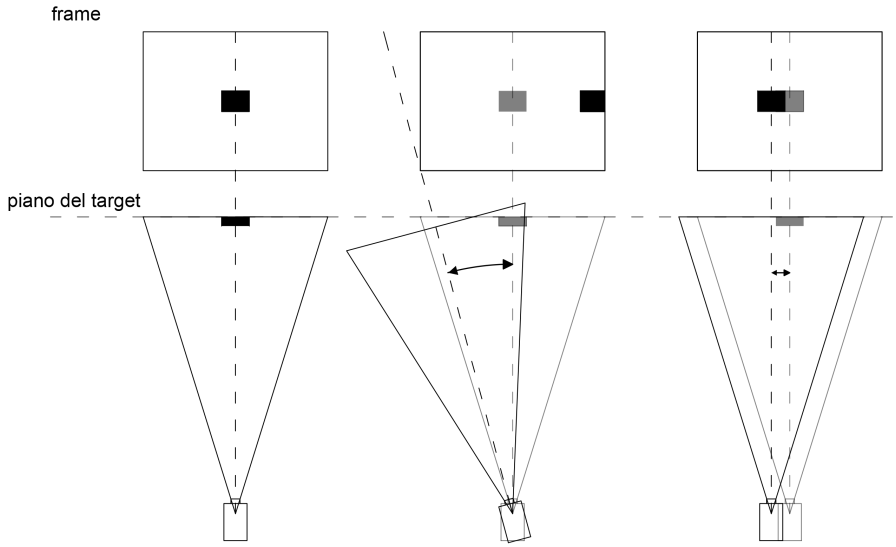


Figura 7: Schema esplicativo degli effetti di rotazioni e traslazioni della videocamera

Tale figura ci permette di ottenere una valutazione qualitativa del diverso effetto di una rotazione o di una traslazione della videocamera. Una valutazione quantitativa si può ricavare, invece, eseguendo dei banali calcoli: basti pensare ad un target posto a 100 metri di distanza dalla videocamera. Nel caso in cui quest'ultima ruoti rispetto all'asse di pan di 15° (come nell'esempio in fig. 7) il LOS si sposta a livello del piano del target di una distanza pari a

$$y = 100 * \tan(15) = 26.79m \tag{1}$$

mentre una traslazione rispetto all'asse y di un metro determina una medesima traslazione del LOS sul piano del target. Come si può capire tali movimenti determinano degli spostamenti del target molto diversi, in particolare le differenze si amplificano proporzionalmente alla distanza del target dalla videocamera.
Ciò permette di capire come sia importante, in un applicazione di videosorveglianza, riporre molta attenzione nella compensazione di eventuali rotazioni di α o β rispetto alle rotazioni di γ o traslazioni in quanto le prime possono procurare una perdita del target. Il brandeggio in questione può essere considerato come una ISP a due gradi di libertà in grado di movimentare la telecamera lungo gli assi di pan e tilt. Può quindi essere utilizzato per la compensazione elettromeccanica di rotazioni rispettivamente di yaw e pitch, garantendo così la stabilizzazione dell'immagine.
Inizialmente si è fatta una valutazione inerente al tipo di metodo da utilizzare per stimare i movimenti da compensare. Una prima pos-

sibilità si basava sull'elaborazione delle immagini provenienti dalla videocamera mentre la seconda si basava sul montaggio di un sensore inerziale aggiuntivo (*Inertial Measurament Unit, IMU*). La scelta finale è stata indirizzata verso la seconda soluzione per i seguenti motivi:

- insensibilità della IMU ai problemi dovuti all'uscita del target osservato dal piano immagine. Ciò può verificarsi per brusche movimentazioni a cui la base del brandeggio può essere soggetta e che non riescono ad essere compensate istantaneamente. Utilizzando il primo metodo l'uscita del target comprometterebbe la possibilità di recuperarlo;
- indipendenza da condizioni atmosferiche, occlusioni del sensore ottico, e dai noti problemi del degrado della qualità dell'immagine in mare;
- banda passante più elevata. I dati dalla IMU vengono trasmessi con una frequenza che oscilla attorno ai 40 Hz. Si riescono quindi ad ottenere informazioni aggiornate più frequentemente rispetto a quelle che si otterrebbero dall'acquisizione video con frequenza di 25 Hz;
- videocamere poco costose non permettono di ottenere buoni risultati attraverso le tradizionali tecniche di stabilizzazione dell'immagine a livello software.

Gli esperimenti, eseguiti in laboratorio, hanno previsto la realizzazione di un simulatore per il moto ondoso costituito da un manipolatore parallelo a sei gradi di libertà (piattaforma di Gough-Stewart). Sono state ricavate delle registrazioni di andamenti in roll, pitch e yaw di boe e navi. Tali riferimenti sono stati poi inviati al simulatore per testare l'efficacia del controllo realizzato. Ulteriori prove sono state poi eseguite con semplici riferimenti sinusoidali su ogni asse di rotazione.

1.5.1 *Contenuto della tesi*

Dopo questa breve introduzione, l'elaborato si sviluppa in quattro capitoli principali. Il primo di questi fornisce una descrizione di tutti i componenti utilizzati e del setup sperimentale nel suo complesso. Il secondo, espone le metodologie utilizzate per l'identificazione del modello del sottosistema che si andrà a controllare. Con il termine sottosistema si intende gli assi di pan e tilt del brandeggio. L'algoritmo di sensor-fusion implementato nella scheda IMU viene analizzato dettagliatamente nel quarto capitolo. Nel quinto capitolo verrà descritto il controllo sviluppato per la compensazione del moto ondoso e verranno descritti i dati sperimentali raccolti nelle varie prove. Infine, nell'ultimo capitolo, viene proposta la compensazione off-line del

video del movimento di roll e una verifica della bontà del controllo sviluppato attraverso un analisi sui video raccolti.

2 DESCRIZIONE DEL SET-UP SPERIMENTALE

2.1 INTRODUZIONE

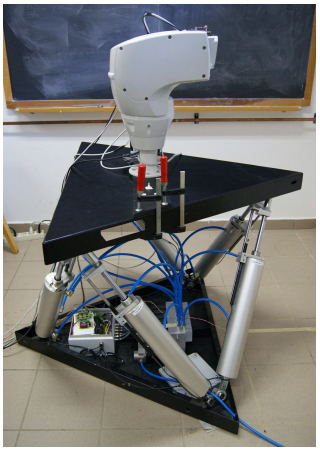


Figura 8: Setup sperimentale completo

Durante lo svolgimento di questo elaborato è stato necessario allestire un apparato sperimentale abbastanza complesso. Tale set-up di lavoro si articola principalmente in due sottosistemi. Il primo di questi è la piattaforma di Gough-Stewart che costituisce il vero e proprio banco di prova mediante il quale viene simulato il movimento ondulatorio a cui può essere sottoposta una telecamera posizionata su una boa in mare. Il secondo sottosistema, invece, è costituito dal dispositivo sotto analisi che, in questo lavoro di tesi, risulta essere il brandeggio Ulisse Compact. A contorno di questi due dispositivi principali sono stati utilizzati una IMU board, schede di potenza e schede di acquisizione dati essenziali per l'implementazione dei vari loop di controllo e per il monitoraggio del sistema nel suo insieme. Qui di seguito vengono descritti uno ad uno i componenti che vanno a costituire il setup completo utilizzato. Il setup sperimentale quindi risulta così composto:

- Piattaforma di Gough-Stewart;
- Brandeggio Ulisse Compact;
- Driver di potenza per il comando dei motore;
- Scheda di acquisizione PCI Sensoray 626;
- Due PC con software Matlab/Simulink completo di toolbox per simulazioni in Real-Time;

- 9DOF-RAZOR IMU;
- Un PC per la registrazione video con software VLC.

2.2 PIATTAFORMA DI GOUGH-STEWART

Figura 9: Simulatore Cyber Air Space.

Il simulatore di moto ondoso che si è voluto realizzare è essenzialmente costituito dal modello *Cyber Air Space* della *Virtogo* (attualmente acquistata dalla *Motion Control*), il quale viene venduto come simulatore nel campo dei videogiochi. Tale manipolatore, come si può osservare dalla fig. 9, è composto principalmente da:

- 2 basi, una fissa appoggiata al pavimento e una mobile;
- 6 attuatori lineari che collegano la base fissa a quella mobile mediante dei giunti rotoidali. Ogni attuatore è costituito da un pistone pneumatico a doppio effetto e da un potenziometro a resistenza variabile che permette di ottenere misure sull'allungamento dei singoli cilindri;
- 6 elettrovalvole proporzionali, una per ogni attuatore, che regolano il flusso dell'aria compressa nelle due camere, inferiore e superiore dei pistoni pneumatici;
- scheda di acquisizione dati che gestisce i segnali input-output tra la piattaforma di Gough-Stewart e il calcolatore;
- scheda per il condizionamento dei segnali che gestisce l'apertura delle 6 valvole;
- scheda di adattamento della tensione per alimentare correttamente i potenziometri e la scheda di condizionamento del segnale.

La struttura così realizzata costituisce un'Hexapod (una variante della piattaforma di Gough-Stewart). Tale manipolatore parallelo garantisce all'end-effector, cioè la base mobile, 6 DOF (degree of liberty).

Model number- *Cyber Air Space*	2.1
DOF	6
Pitch [deg]	25
Roll [deg]	25
Yaw [deg]	9
Alzata [m]	0.88
Max Speed [m/s]	114
Corsa [m]	0.2
Massimo carico applicabile [Kg]	300
Peso a vuoto, compresa poltrona [Kg]	90
Altezza [m]	0.68
Consumo aria (m^3 a 100 PSI)	12.4

Tabella 1: Dettagli del costruttore Cyber Air Space.

Le motivazioni che hanno portato alla scelta di tale manipolatore sono state in particolare: l'elevato rapporto tra carico utile e peso della struttura, le buone prestazioni dinamiche e l'elevata rigidezza della struttura, dovuta al fatto che il carico viene ripartito tra le sei gambe del manipolatore.
Lo svantaggio principale di tale tipo di robot parallelo rispetto a quelli seriali è la limitatezza dello spazio di lavoro in cui possono operare, la presenza di configurazioni singolari al suo interno e la complessità dell'analisi cinematica sia diretta che inversa e il posizionamento non molto accurato a causa dell'attuazione pneumatica.
Per quanto concerne il lavoro sviluppato ci siamo soffermati esclusivamente sulla cinematica inversa, utile per la realizzazione del sistema di controllo e quindi imporre le movimentazioni desiderate alla piattaforma mobile. Viene invece tralasciata la cinematica diretta del manipolatore perché molto più complessa e non essenziale per il raggiungimento degli obbiettivi desiderati.

2.2.1 *Cinematica inversa*

La cinematica inversa è il processo di determinazione, attraverso equazioni cinematiche, della posizione degli attuatori o dei parametri dei giunti di un manipolatore al fine di ottenere una desiderata posizione dell'end-effector. Tali risultati sono necessari per la corretta movimentazione del robot.
Per iniziare la trattazione della cinematica inversa è necessario andare a definire due sistemi di riferimento di cui uno fisso con la base e uno fisso con l'end-effector o piattaforma mobile. In ciascun riferimento vengono poi definite le posizioni dei giunti che legano i pistoni pneu-

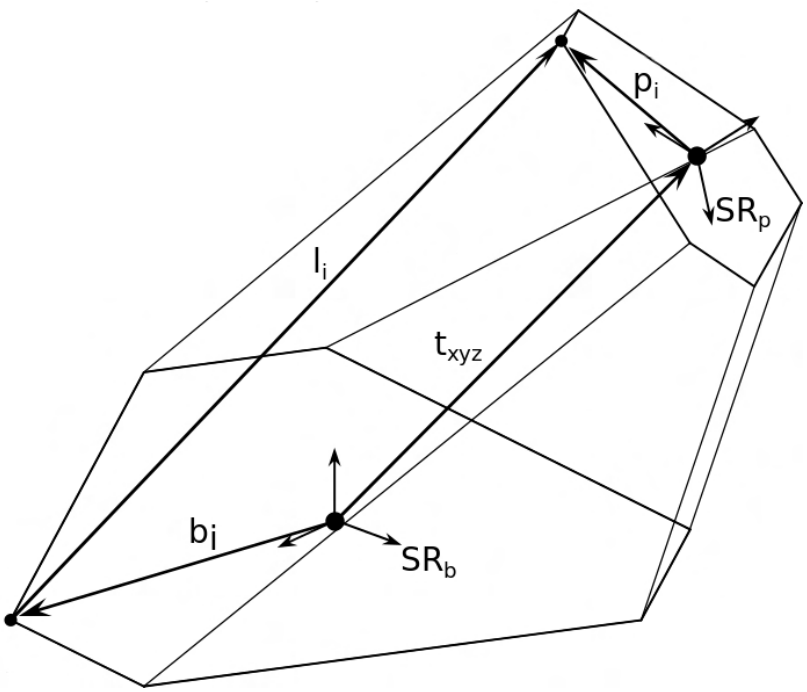


Figura 10: Schema della piattaforma di Gough-Stewart

matici alle due piattaforme definite rispettivamente dai vettori $\mathbf{b_i}$ e $\mathbf{p_i}$ osservabili in fig. 10. Al fine di ottenere le lunghezze desiderate dei diversi cilindri è necessario traslare e ruotare opportunamente il sistema di riferimento della piattaforma mobile ed insieme ad esso anche le posizioni dei giunti superiori.
La trasformazione che lega la descrizione del sistema di riferimento della base mobile SR_p^b rispetto al sistema di riferimento della base fissa SR_b è la seguente:

$$SR_p^b = \mathbf{t_{xyz}} + R_{DCM} SR_b \tag{2}$$

dove il vettore $\overrightarrow{t_{xyz}}$ definisce la posizione dell'origine del sistema di riferimento SR_p rispetto al sistema di riferimento della base e la matrice di rotazione R_{DCM} (dove DCM sta per *Direct Cosine Matrix*) descrive le rotazioni che devono essere eseguite per orientare opportunamente il sistema di riferimento della base mobile. Tale matrice è cosi definita:

$$R_{DCM} = \begin{bmatrix} c\beta c\alpha & s\gamma s\beta c\alpha - c\gamma s\alpha & c\gamma s\beta c\alpha + s\gamma s\alpha \\ c\beta s\alpha & s\gamma s\beta c\alpha + c\gamma c\alpha & c\gamma s\beta s\alpha - s\gamma c\alpha \\ -s\beta & s\gamma c\beta & c\beta c\beta \end{bmatrix} \tag{3}$$

La rotazione della base mobile viene descritta secondo la convenzione yaw-pitch-roll cioè come la successione dei tre rotazioni semplici intorno ad un singolo asse:

- yaw rotazione intorno all'asse z di misura α;
- pitch rotazione intorno all'asse y di misura β;
- roll rotazione intorno all'asse x di misura γ.

Per quanto riguarda i vettori $\mathbf{p_i}$ la loro descrizione rispetto al sistema di riferimento SR_p non cambia, ma ciò che si modifica è la descrizione rispetto il sistema di riferimento della base fissa secondo l'equazione seguente:

$$\mathbf{p_i^b} = \mathbf{t_{xyz}} + R_{DCM} * \mathbf{p_i^p} \tag{4}$$

dove $\mathbf{p_i^p}$ corrisponde al vettore che definisce la posizione del giunto superiore della link i definito nel sistema di riferimento SR_p, mentre $\mathbf{p_i^b}$ rispetto al riferimento SR_b. A questo punto si hanno a disposizione i vettori di tutte le posizioni dei giunti, sia inferiori che superiori, tutti riferiti allo stesso sistema di riferimento SR_b. Per ottenere la lunghezza di ogni singolo attuatore è quindi sufficiente svolgere il seguente calcolo:

$$length_i = |\mathbf{l_i}| = |\mathbf{p_i^b} - \mathbf{b_i}| \tag{5}$$

Il listato di tale algoritmo implementato in *Matlab Function* è riportato in appendice A.1.1.

2.2.2 *Circuito pneumatico e elettrovalvole*

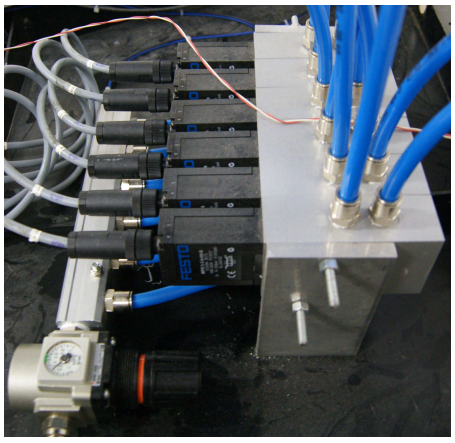


Figura 11: Particolare del circuito pneumatico

Il circuito pneumatico che alimenta i cilindri del manipolatore preleva aria compressa dalla linea di distribuzione presente in laboratorio. A monte del circuito è installata una valvola di controllo che permette la regolazione la pressione in ingresso (vedi fig. 11) e subito a valle di tale valvola vi è un collettore che distribuisce l'aria a tutte e sei le elettrovalvole (modello MPYE-5-...B prodotte della *FESTO*).

Tali elettrovalvole proporzionali hanno da poco sostituito delle vecchie elettrovalvole discrete al fine di migliorare la risposta dinamicha della piattaforma. Queste servovalvole vengono controllate da un segnale elettrico in tensione da 0 a 10 V.
Tale segnale viene utilizzato come riferimento dall'elettronica di controllo di un solenoide. Il campo magnetico prodotto da quest'ultimo condiziona la posizione di una slitta presente all'interno dell'elettrovalvola (vedi fig. 12a). A seconda della posizione in cui si trova è possibile modulare il flusso d'aria in modo continuo e proporzionale al segnale in ingresso. La condizione di riposo della valvola, cioè quella in cui entrambe le uscite sono chiuse si verifica quando la tensione di ingresso è pari a 5 V. Quando il riferimento varia da tale condizione di equilibrio o verso gli 0 V o verso i 10 V inizia a fluire aria in una delle due uscite. In questa maniera viene controllato il flusso d'aria convogliato in una delle due camere del pistone e di conseguenza la sua velocità di estensione o di accorciamento.
Le elettrovalvole devono essere alimentate con una tensione compresa tra 17 e 30 V in DC. Si è optato quindi, di fornire tale alimentazione, attraverso un alimentatore *ECOLINE CP SNT* 250 W che fornisce in uscita una tensione di 24 V in DC e un massimo di 10 A. Tale alimentatore è parecchio sovradimensionato per questa applicazione, ed è per questo che, in seguito, viene utilizzato anche come sorgente per le schede di regolazione della tensione e per i driver dei motori.

(a) Schema elettrovalvola

(b) Curva tensione-% portata fluido in uscita

Figura 12: Elettrovalvole MPYE-5-...B.

In fase di realizzazione del controllo per il manipolatore è sorto subito un problema dovuto alla presenza di una zona morta nell'intorno dei 5 V causata dal meccanismo di funzionamento delle elettrovalvole (vedi fig. 12b). Quest'ultima impedisce la diretta applicazione dei riferimenti ottenuti dai controllori PID sintetizzati nel loop di controllo. Per compensare tale non linearità, è stato necessario introdurre una azione feedforward attraverso una particolare funzione ottenuta empiricamente ed inserita in una Look Up Table (LUT). Come si vede in fig. 13 il riferimento di posizione viene derivato in modo tale da ottenerlo in velocità e successivamente viene fatto passare dalla fun-

zione ottenuta sperimentalmente. La funzione ha un andamento tale per cui anche un piccolo errore di velocità garantisce l'apertura di una delle uscite della valvola evitando così la persistenza, per piccoli errori, all'interno della zona morta.
A tale segnale viene poi sommata l'uscita del controllore PID calcolata sull'errore di posizione.

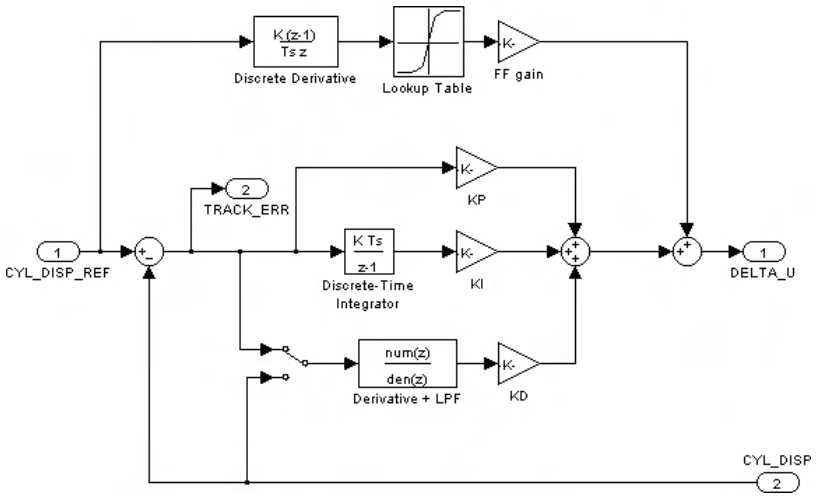


Figura 13: Particolare del controllo dei cilindri pneumatici

2.2.3 *Scheda MultiQ-3*

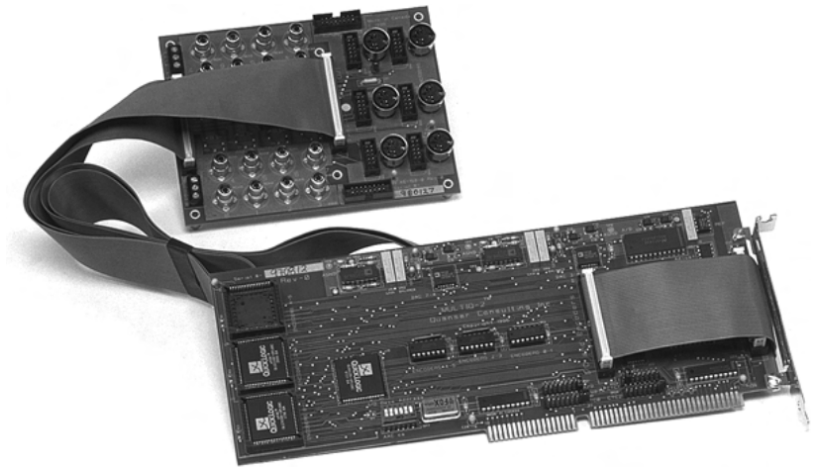


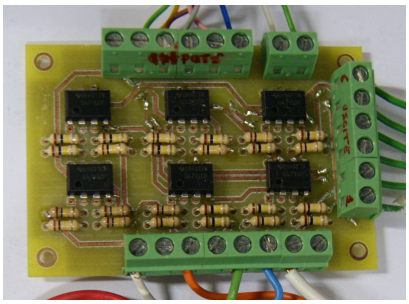
Figura 14: Scheda MultiQ-3.

Il dispositivo MultiQ-3 è una scheda di acquisizione e controllo dei dati multiuso. Essa dispone di 8 ingressi analogici single-ended, 8 uscite analogiche, 16 bit di ingresso digitali, 16 bit di uscita digitali, 3 timer programmabili e fino ad 8 ingressi encoder decodificati in quadratura.
In questo setup sperimentale tale scheda viene utilizzata per realizza-

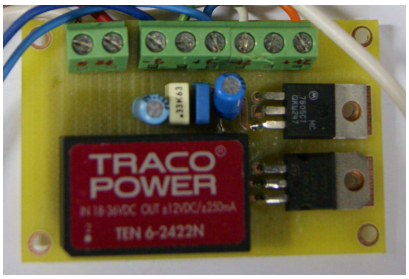
re l'interfacciamento tra calcolatore e piattaforma di Gough-Stewart. Vengono quindi utilizzati sei degli ingressi analogici per leggere le uscite dei potenziometri, e quindi avere una misura della lunghezza degli attuatori, e sei uscite analogiche per imporre i comandi alle servovalvole. Infine viene utilizzata un ulteriore uscita analogica per realizzare la sincronizzazione tra i calcolatori in fase di raccolta dei dati sperimentali (tale aspetto verrà approfondito nei capitoli successivi).
Per installare la scheda è necessario innanzitutto spegnere il computer, inserire i cavi a nastro piatto attraverso il blackplane e poi inserire la scheda MultiQ nello slot ISA della motherboard. Infine vanno inseriti i due cavi a nastro piatto nella morsettiera che permette un più facile accesso agli ingressi e alle uscite della scheda stessa.
A causa dell'assenza di driver disponibili è stato necessario compilare due particolari *S-Function* per interfacciare la scheda al software Matlab. Al loro interno sono state richiamate della funzioni appartenenti alla libreria *WinIO* che permettono, anche in ambiente Windows, di avere accesso agli indirizzi di memoria degli ingressi e uscite di particolari periferiche quale ad esempio la periferica ISA. Le due S-Function implementate servono una per la conversione digitale-analogica dei comandi alle servovalvole e per la conversione analogica-digitale degli ingressi provenienti dei potenziometri.
La conversione analogico-digitale (A/D) della scheda MultiQ-3 è di tipo single ended bipolare con segno a 13 bit (12 bit più segno). È possibile eseguire una conversione in uno degli 8 canali, selezionando il canale e iniziando la conversione. Il bit EOC_I (end of conversion interrupt) nello STATUS_REGISTER indica che il dato è pronto e può essere letto. I dati vengono letti dal AD_DATA register.
I dati restituiti sono due parole a 8 bit che devono essere combinati per dare come risultato una parola a 16 bit con segno. Il range accettabile in ingresso alla scheda di acquisizione è ± 5 V single-ended. Un ingresso di $+5$ V viene codificato con $0 \times \mathrm{FFF}$, uno di 0 V con 0×0 e -5 V con $0 \times \mathrm{FFFF000}$. Per quanto riguarda i convertitori digitali-analogici (D-A) sono a 12 bit senza segno. Un input di -5 V è codificato con 0×000, uno di 0 V con $0 \times 3\mathrm{FF}$ mentre i $+5$ V con $0 \times \mathrm{FFF}$. Per imporre quindi una certa tensione in uscita è necessario scrivere un numero a 12 bit $(0 - 4.095)$ in un apposito registro. I listati di tali funzioni sono riportati in appendice A.3.2 e A.3.1.

2.2.4 *Schede per il condizionamento e gestione dei segnali*

Un problema riscontrato durante lo sviluppo del simulatore, ed in particolare nei cablaggi e nella gestione dei segnali, era costituito da un diverso range di tensioni di lavoro tra la scheda di acquisizione MultiQ-3 (± 5 V) e i segnali di comando alle servovalvole $(0 - 10$ V). È stato quindi necessario andare a progettare una scheda elettronica



(a) Scheda condizionamento del segnale



(b) Scheda per la gestione della tensione

Figura 15: Schede PCB (printed circuite board) realizzate

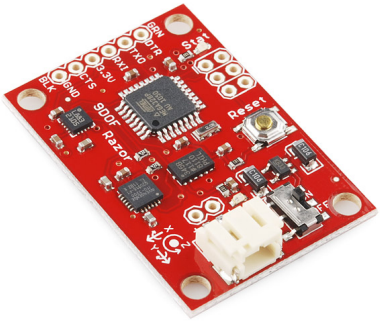
(vedi fig. 15a) al fine di adattare i due diversi range di tensione. La soluzione adottata, scelta per semplicità e economicità costruttiva, è costituita da sei semplici amplificatori operazionali *LM*741 in configurazione invertente e sommatore di tensione. Infatti per passare da ± 5 V a $0 - 10$ V basta andare a sommare 5 V al segnale in ingresso.
Per garantire il corretto funzionamento di tale scheda è necessario andare ad alimentare opportunamente gli operazionali. É stato quindi necessario realizzare un'ulteriore scheda (vedi fig. 15b) che converta la tensione in uscita dall'alimentatore[1] da 24 V ai $\pm$12V necessari. Tale compito viene affidato ad un normale convertitore DC/DC.
Nella stessa scheda di gestione della tensione sono stati aggiunti altri due componenti: *LM*7905 e *LM*7805. Questi componenti sono degli stabilizzatori di tensione rispettivamente a -5 V e a $+5$ V. Queste ulteriori uscite di tensioni servono per alimentare i potenziometri lineari utilizzati nelle misure delle lunghezze dei link del manipolatore. I potenziometri lavorano come dei partitori di tensione variabili. Il potenziometro da in uscita una tensione proporzionale all'allungamento, cioè se il potenziometro è completamente contratto l'uscita si porta alla potenziale di alimentazione più basso mentre, se è completamente esteso tale uscita si porta al potenziale più alto. Si è scelto di alimentarli a ± 5 V in modo tale che le loro uscite si ritrovino sempre all'interno di questo range di tensione e quindi non è stato necessario andare a progettare un'ulteriore scheda di adattamento dei livelli di tensione per l'acquisizione dei segnali dai potenziometri.
Gli schemi elettrici di tali schede PCB sono riportati in appendice B.1 e B.2.

2.3 9 DOF-RAZOR IMU, SEN-10736

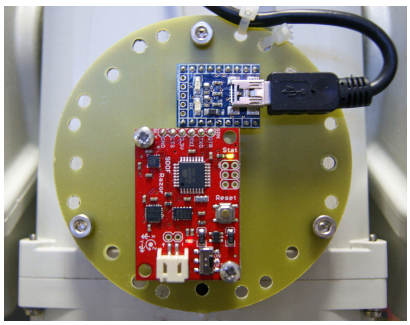
La scheda 9DOF-RAZOR IMU è un sensore inerziale che incorpora al suo interno tre sensori:

- ITG-3200, un giroscopio a tre assi;

1 lo stesso utilizzato per le servovalvole



(a) sensore IMU



(b) sensore IMU con adattatore seriale/USB montati sulla scheda PCB di interfaccia. Il tutto fissato sulla videocamera

Figura 16: 9DOF-RAZOR IMU, SEN-10736

- ADXL345, un accelerometrO a tre assi;
- HMC5883L un magnetometro , anch'esso a tre assi;

Il tutto determina 9 gradi di libertà per quanto riguarda le misure inerziali.
I segnali in uscita vengono processati dalla scheda *ATmega*328 montata a bordo della IMU board. I segnali elaborati vengono poi forniti in uscita attraverso un'uscita seriale. Questo permette alla scheda 9DOF-RAZOR di essere utilizzata come un meccanismo di controllo in tutti quei sistemi dove è richiesto la misura di orientazioni di corpi rigidi quali ad esempio i sistemi di stabilizzazione delle immagini (caso in oggetto), controlli per veicoli autonomi, UAVs (*Unmanned aerial vehicle*). La scheda *ATmega*328 viene programmata attraverso il bootloader Arduino 8 MHz *(stk*500*v*1*)*. È sufficiente connettere la scheda al computer mediante un'adattatore e simulare attraverso una porta USB un ingresso seriale. È possibile utilizzare l'IDE Arduino per programmare il codice, che poi verrà caricato sulla scheda, basta selezionare la board corretta (Arduino Pro o Pro Mini (3.3 V, 8 MHz) w/ATmega328) nel menu a tendina. L'alimentazione alla board viene fornita direttamente attraverso la porta USB. Per interfacciare la morsettiera USB alla board IMU si è realizzata un ulteriore scheda PCB il cui schema è riportato in appendice B.3.

2.4 IL BRANDEGGIO ULISSE COMPACT

I sistemi integrati ULISSE prodotti dalla VideoTec S.p.a. uniscono un brandeggio ad alta velocità con rotazione continua, una custodia per telecamera e un ricevitore di telemetria multiprotocollo. Sono stati

sviluppati vari modelli diversi tra loro per taglia (compatta, media o maxi), per tipo di controllo (analogico e/o IP), per telecamera (integrata, classica, network, termica oppure con illuminatore ad IR). La gamma ULISSE è quindi la soluzione ideale per ogni tipo di applicazione per videosorveglianza in ambiente esterno. Per quanto riguarda l'applicazione in oggetto è stato utilizzato il brandeggio della serie ULISSE denominato ULISSE COMPACT. Tale brandeggio presenta le seguenti caratteristiche tecniche principali [19]:

- Porte analogiche PAL/NTSC e seriali multiprotocollo RS485/RS422;
- IP Video Encoder MPEG4, 25 fps, Full D1;
- Velocità variabili 0.1°-200°/s Pan/Tilt;
- Rotazione orizzontale (o di Pan) continua;
- Rotazione verticale (o di Tilt) compresa tra $-90°$ e $+90°$;
- Day/Night Sony camera 36×, 28×, 1× or 10×;
- Stabilizzatore di immagine avanzato (Stable Zoom Mode);
- Peso, 12.5 kg;
- Dimensioni, $660 \times 330 \times 570$ mm.

La struttura portante è costituita da un telaio in lega leggera al quale vengono fissati tutti i componenti. Le movimentazioni di pan e di tilt vengono eseguite da due hybrid step motor collegati con l'asse da controllare mediante cinghie dentate. I due assi sono posizionati in modo tale da essere ortogonali e indipendenti tra loro e rispetto al LOS.
Sempre all'interno trovano posto anche due schede elettroniche chiamate di Base e di Body [7, 9, 8]. La prima permette l'alimentazione del brandeggio a partire dalla tensione di rete. Essa rende, inoltre, disponibile in uscita il segnale proveniente dalla telecamera e prevede alcuni ingressi che consentono l'azionamento dei motori tramite una tastiera in dotazione assieme al brandeggio. La seconda, ovvero la scheda di Body, racchiude le funzioni di controllo e di comunicazione del brandeggio. Questa permette quindi il pilotaggio dei motori degli assi di rotazione (pan e tilt), e delle ottiche della telecamera (zoom, focus e iris). In questa scheda sono presenti anche un microcontrollore e un FPGA. Il microcontrollore viene utilizzato per gestire i comandi provenienti dalla tastiera. Per quanto riguarda l'FPGA genera le leggi di moto ai motori come riferimenti in corrente che vengono elaborati da un DAC che li converte in segnali analogici. Sull'asse di tilt è presente anche un piccolo encoder che ha la sola funzione di permettere una registrazione iniziale dell'asse stesso e l'individuazione dei limiti massimi di rotazione imposti dalla struttura (calibrazine dell'asse). Esternamente il brandeggio si presenta come in fig. 17, con un

case plastico rigido, che racchiude tutti i componenti sopra descritti, e completamente scomponibile per agire comodamente su di essi. Lungo tutte le giunture sono presenti delle guarnizioni per renderlo impermeabile e quindi adatto ad operare all'esterno. Ai fini del pro-

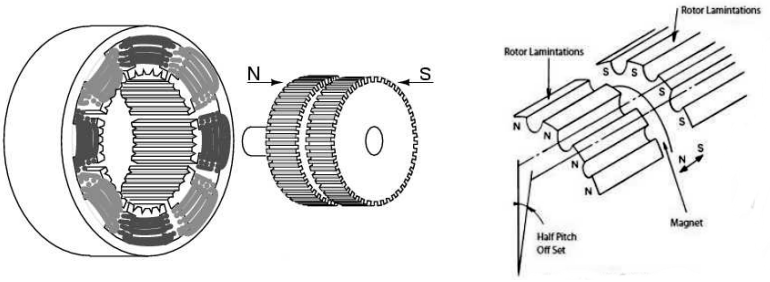
Figura 17: Ulisse Compact

getto si sono scollegate tutte le schede elettroniche interne ed i riferimenti ai motori sono dati attraverso dei driver aggiuntivi comandati direttamente dal PC usato per implementare il controllo degli assi. Per quanto riguarda approfondimenti sul tipo di motori utilizzati e le tecniche di pilotaggio adottate si rimanda al prossimo paragrafo.

2.4.1 *Hybrid step motor*

I motori adottati nell'Ulisse Compact, come precedentemente accennato, sono hybrid step motor a due fasi (o HY). Questa particolare categoria di motori passo passo, sfrutta il principio dei motori a riluttanza variabile (o VR) e contemporaneamente quello dei motori a magneti (o PM). La struttura dello statore è praticamente quella di un motore VR, con la differenza che mentre nel motore VR attorno ad ogni dente trova posto l'avvolgimento di una singola fase, nei motori HY attorno ad ogni dente trovano posto due fasi diverse. Pertanto un dente non è più associabile ad una singola fase, bensì ad una coppia di fasi avvolte generalmente con versi opposti. Il rotore ha una struttura particolare. Il suo nucleo è costituito da un magnete permanente cilindrico, che produce un flusso assiale unipolare come mostrato in fig. 18a. La parte esterna, rappresentata in fig. 18b è composta da una struttura dentata, tipica del motore VR, e i denti delle due sezioni sono disallineati tra loro di mezzo passo di dentatura. Per quanto riguarda la generazione di coppia in tali motori e ulteriori approfondimenti si rimanda a [21].

I motori a passo, in generale, sono quindi caratterizzati da un movimento a scatti regolari, corrispondenti ad un preciso angolo di rotazione detto passo. Nel caso dei motori a passo ibridi, proprio per l'effetto del disallineamento tra le due strutture dentate, si riesce ad ottenere un numero molto maggiore di passi rispetto ad un motore



(a) Spaccato del motore passo ibrido (b) Sezione del rotore

Figura 18: Rotore e Statore di un motore a passo ibrido

PM o VR[2] a scapito però di un maggior costo di costruzione.

Tra i vantaggi principali dei motori HY si possono citare:

- posizionamenti accurati senza alcun sensore di posizione o velocità;
- errore di posizionamento non cumulativo;
- grande risoluzione;
- alto rendimento;
- sono particolarmente semplici e adatti al controllo digitale;
- assenza di contatti striscianti e quindi un'elevata robustezza meccanica;
- elevata coppia di tenuta (o holding torque).
- possono raggiungere elevate velocità (elevata curva di pull-out) se avviati in rampa;
- bassi costi se confrontati con altri tipi di motore con analoghe prestazioni;

Per contro, il principale difetto è la generazione di coppia non costante (ripple di coppia e coppia di cogging) che causa l'instaurarsi di vibrazioni nella movimentazione. Un ulteriore problema di questo tipo di motori è il limite imposto dalla curva di pull-in. Questa curva racchiude la regione di avviamento in cui il motore sotto carico può avviarsi o fermarsi istantaneamente, senza perdita di sincronismo. Il range di avviamento dipende dall'inerzia del carico applicato. In altre parole, le curve di pull-in sono misurate in funzione di diversi momenti di inerzia e dunque forniscono una possibile coppia di avviamento già depurata dalla componente inerziale. La curva di pull-in è quindi un grosso limite perché limita di fatto la velocità di partenza e di arresto del motore. I motori presenti nell'Ulisse Com-

2 tipicamente un motore HY può avere 400/100 passi/giro contro i 48/24 passi/giro di un motore PM o VR

Figura 19: Hybrid step motor

pact atti alla movimentazione degli assi di pan e tilt sono uguali. Su questi non si hanno specifiche provenienti da manuali di supporto o datasheet e quindi ci si è basati su quelle presenti sulla targa del motore che, anche se non molto dettagliata, illustra dati sufficienti per gli scopi del progetto. In particolare, come si può osservare in fig. 19, la targa riporta il passo, 1.8°, e la massima corrente applicabile per ogni fase, 2.8 A. Per quanto riguarda invece la massima velocità raggiungibile e la massima velocità di avviamento sono state ricavate sperimentalmente.
I motori sono gestiti sfruttando un pilotaggio in microstepping. Questa tecnica consiste nell'inviare alle due fasi del motore due correnti sinusoidali in quadratura tra loro. Tanto maggiore sarà la corrente in una fase rispetto a quella nell'altra, tanto più il rotore si posizionerà vicino al dente di statore corrispondente alla fase con corrente più elevata. Questa tecnica consente di ottenere una maggiore fluidità del movimento, un aumento della precisione nel posizionamento, per via dell'aumento della risoluzione, ed una diminuzione delle vibrazioni durante il moto. Per regolare la corrente servono naturalmente circuiti molto complessi. In pratica, nelle applicazioni industriali si usano processori dedicati oppure appositi circuiti integrati. Il vantaggio, va ribadito, consiste nell'eliminazione del funzionamento a scatti, uno dei più marcati difetti dei motori a passo nelle applicazioni di precisione[21].

2.4.2 *Trasmissione*

La trasmissione del moto dai motori ai rispettivi assi di pan e tilt è realizzata tramite un accoppiamento a cinghia dentata, visibile in fig. 20 con rapporto di trasmissione $\tau = 1/7$.

Le cinghie dentate sono spesso usate in quanto permettono di trasmettere grandi potenze senza slittamenti e con rendimenti elevati e, inoltre, permettono di ottenere un movimento poco rumoroso. Per contro, questo sistema di trasmissione del moto porta ad avere inevitabili elasticità dovute alla struttura della cinghia. Questo introduce delle risonanze che portano all'instaurarsi di fenomeni vibratori indesiderati. Nel caso dell'asse di tilt queste vibrazioni sono molto più

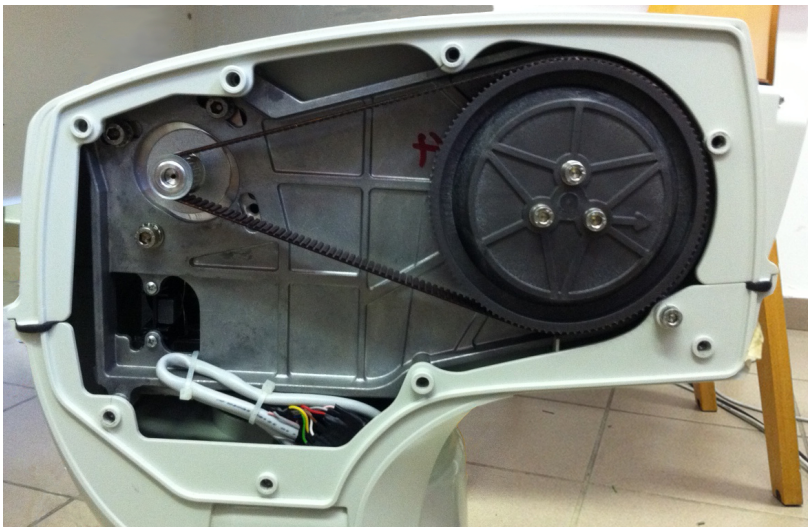
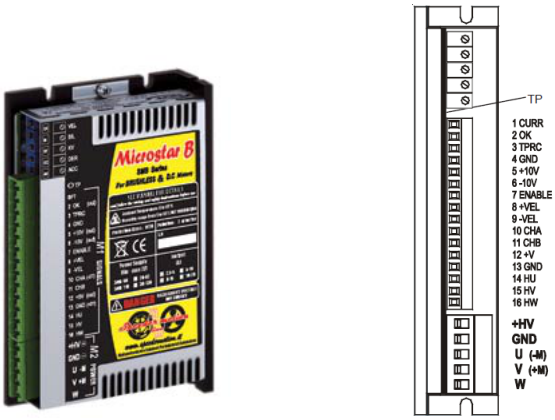


Figura 20: Cinghia di trasmissione del moto di Tilt

accentuate rispetto all'asse di pan. Questo può essere dovuto al fatto che l'asse di tilt presenta un inerzia molto minore e quindi anche una minor capacità di smorzare eventuali vibrazioni.

2.5 DRIVER

Come precedentemente accennato i motori atti alla movimentazione dei due assi sono stati scollegati dalla scheda di Body e vengono pilotati indipendentemente dal PC mediante appositi Driver. I driver a disposizione, fig. 21, sono i *Microstar B* prodotti dall'Agenzia Ing. Pini s.r.l. e fanno parte della linea Speeder Motion.



(a) Driver

(b) Piedinatura Driver

Figura 21: Driver MicroStar-B

Tali convertitori sono azionamenti adatti al pilotaggio sia di moto-

ri brushless che di motori a corrente continua ma le loro molteplici modalità operative li rendono adattabili, con qualche considerazione aggiuntiva, anche a questo tipo di applicazione [11]. Le modalità operative supportate sono:

- controllo di velocità da ingresso analogico in cui il motore è controllato da un riferimento analogico (differenziale o di modo comune) di velocità in ingresso tra i morsetti ±VEL
- controllo di coppia in cui il motore è controllato con un riferimento analogico di coppia. Questa regolazione consente di pilotare il convertitore in torque mode attraverso l'ingresso analogico in modo comune TPRC.

Sfruttando la seconda modalità operativa e applicando all'ingresso TPRC un segnale di tensione V_{ing} di max ±10 V si ottiene in uscita un segnale in corrente con andamento analogo all'ingresso ma scalato di una costante che dipende dalla taglia dell'azionamento. In altre parole l'azionamento può essere visto come un amplificatore di transconduttanza. Essendo l'azionamento utilizzato di taglia 10/20 A la formula per determinare il massimo valore di V_{ing} da applicare in TPRC in modo da ottenere la massima corrente imponibile al motore I_{pk} è la seguente

$$V_{ing} = \frac{10}{20} \cdot I_{pk}. \tag{6}$$

Essendo I_{pk} pari a 2.8 A si ottiene quindi

$$V_{ing} = \frac{10}{20} \cdot 2.8 = \frac{1}{2} \cdot 2.8 = 1.4 \text{ V}. \tag{7}$$

Si osserva quindi che la costante di transconduttanza, K_i, che lega la variazione della corrente in uscita con la variazione della tensione in ingresso risulta pari a 2 A/V.

Nel caso in esame sono stati usati due driver per motore, uno per fase, operanti in modalità controllo di coppia. Così facendo si è riusciti a trasformare i segnali in tensione imposti dal controllo sviluppato in Simulink in segnali di corrente adatti al pilotaggio delle singole fasi dei due motori a passo. Per consentire tale funzionamento è quindi necessario operare su ognuno dei quattro driver alcuni collegamenti, sia per la parte di segnale che per la parte di potenza. Per la parte di segnale oltre a collegare l'ingresso in tensione al pin 3 (i. eq. TPRC) si è collegato il pin 5 con il pin 7, corrispondenti rispettivamente a +10 V e ENABLE, e il pin 13 con il pin 15, corrispondenti rispettivamente a GND e HV. La prima coppia di collegamenti è dovuta alla necessità di abilitare in ogni momento il convertitore mentre la seconda coppia tiene a GND gli ingressi per i segnali di Hall provenienti da un eventuale motore Brushless collegato in modalità di controllo della velocità. Per quanto riguarda la parte di potenza si è collegata l'alimentazione del

driver tra il pin +HV e GND e le uscite verso la fase del motore tra V +M e U -M.

I quattro driver, così collegati sono stati fissati alla base dell'Ulisse Compact come mostrato in fig. 22

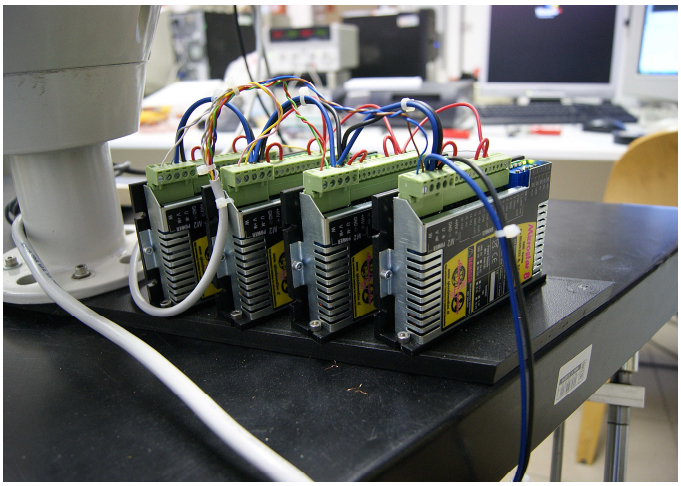
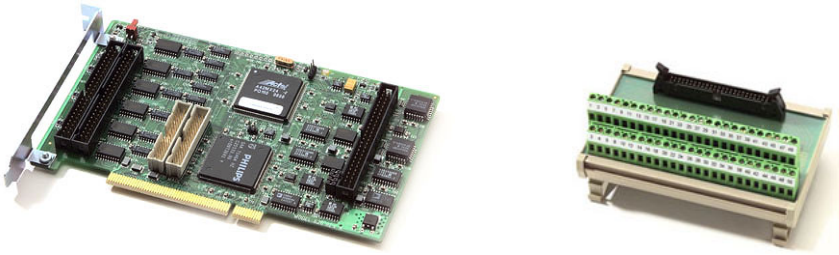


Figura 22: I quattro driver Multistar B

2.6 SCHEDA DI ACQUISIZIONE SENSORAY 626

Per progettare il controllo dell'Ulisse Compact si è fatto uso di Matlab/Simulink associato al toolbox Real Time Windows Target. Questo tool permette di interfacciare in tempo reale il sistema attraverso due blocchi principali di I/O denominati Analog input e Analog Outup. Per acquisire i dati derivanti dal modello fisico si è fatto uso del-



(a) Scheda PCI (b) Morsettiera I/O

Figura 23: Scheda Sensoray 626

la scheda di acquisizione dati Sensoray 626, visibile in fig. 23 e che presenta le seguenti principali caratteristiche [13]:

- 16 ingressi analogici single-ended con risoluzione di 14 bit ed input range selezionabile tra ± 5 V e ± 10 V;
- 4 uscite analogiche con risoluzione di 14 bit ed output range di ± 10 V ;

- 6 Ingressi encoder con risoluzione 24 bit;
- 48 I/O digitali.

Operativamente, in fase di compilazione, il modello Simulink viene trasformato in un file eseguibile, che verrà poi utilizzato durante l'esecuzione in Real-Time. Per una corretta simulazione risulta essenziale settare opportunamente i parametri dei blocchi I/O oltre ovviamente a tutti i parametri generali di simulazione. Verranno di seguito forniti i passi principali per compiere queste due operazioni:

1. digitare nella Command Window di *Matlab* l'istruzione: `rtwintgt -install`;
2. entrare nell'ambiente *Simulink* di *Matlab*;
3. come prima applicazione è sufficiente selezionare dalla libreria di *Simulink* il blocchetto *Analog Output* all'interno del toolbox *Real-time Windows Target* e visualizzare la finestra delle proprietà (fig. 24). Da questa finestra è necessario selezionare *Install*

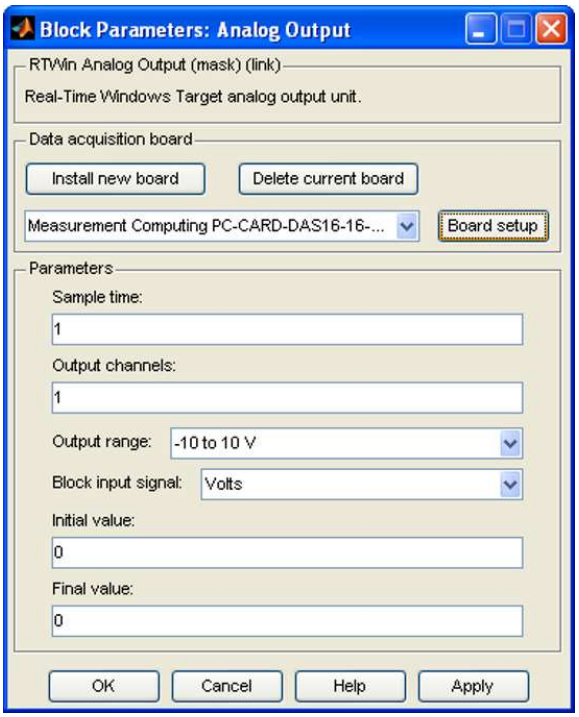


Figura 24: Finestra proprietà *Analog Output*

New Board e scorrere l'elenco delle schede di acquisizione fino a trovare la scheda sopra citata. Fatto ciò, per verificarne il corretto funzionamento, è sufficiente selezionare *Board setup* ed effettuare il *Test*.

4. selezionare dal menù a tendina di *Simulink*: *Simulation* ⇒ *Configuration Parameters* (fig. 25): quindi selezionare *Browse* e cliccare su `Rtwin.tlc`;

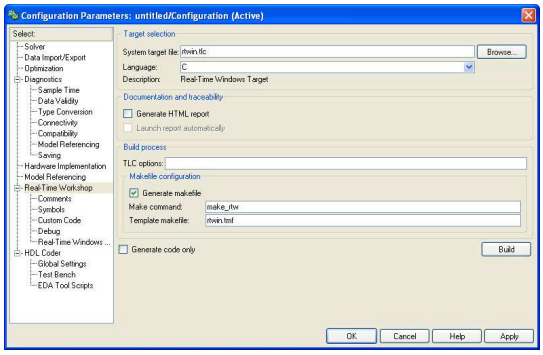


Figura 25: Finestra proprietà *Configuration Parameters*

5. infine, dal menù a tendina di *Simulink* selezionare *Tools* ⇒ *External Mode Control Panel* ⇒*Signal and triggering* (fig. 26): da qui

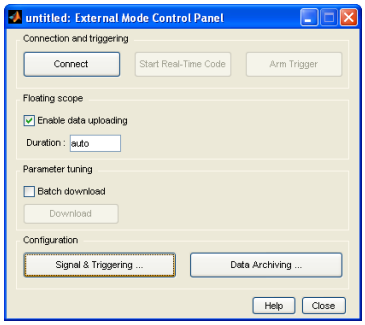


Figura 26: Finestra proprietà *External Mode Control Panel*

   à possibile selezionare la dimensione del buffer su qui verranno salvati i dati delle prove.

6. per la configurazione del blocchetto *Analog Input* di *Simulink* è sufficiente ripetere il punto 3.

2.7 REALIZZAZIONE DEL SETUP SPERIMENTALE COMPLETO

La prima parte del lavoro di tesi ha previsto l'assemblaggio tutti i componenti sopra menzionati al fine di realizzare un setup sperimentale completo. Lo schema a blocchi del setup realizzato è riportato in fig. 27.

Come evidenziato in figura lo schema è composto da tre parti fondamentali indipendenti tra loro:

- Il simulatore del moto ondoso;
- Il sistema di stabilizzazione dell'immagine;
- Il PC per l'acquisizione dei video.

La prima parte è composta dalla piattaforma di Stewart e da tutti i dispositivi che ne permettono il funzionamento quali, PC, la scheda di acquisizione dati MultiQ-3, e le schedine di condizionamento

del segnale. Il riferimento di movimento, dato in roll, pitch e yaw, viene elaborato dal PC e fornito alla piattaforma come spostamento lineare di ogni singolo pistone. Allo stesso tempo, al fine di creare un feedback per i controllori, vengono acquisiti i reali allungamenti dei pistoni e trasformati in segnali di roll, pitch e yaw. Questo loop di controllo permette la movimentazione della piattaforma con qualsiasi segnale di rifermento purché sia fornito come riferimento di rotazione sui tre assi cartesiani e che rientri nei limiti massimi di movimento della struttura.
La seconda parte di schema a blocchi riguarda il sistema di stabilizzazione dell'immagine. Esso è composto dal brandeggio Ulisse Compact, dagli azionamenti citati in 2.5, dal sensore IMU, e dal PC equipaggiato con la scheda per l'acquisizione dei dati Sensoray 626. Il brandeggio è stato installato direttamente sopra la piattaforma cosi come i quattro driver dei motori. Attraverso il sistema di controllo viene stabilizzata la posizione della videocamera interna all'Ulisse Compact attorno ai valori angolari imposti come riferimento. Il feedback di posizione viene effettuato attraverso il sensore IMU. Per quanto riguarda suo posizionamento vi sono due possibili scelte: o sulla piattaforma mobile oppure direttamente sulla videocamera. La prima possibilità è stata subito scartata perché comportava la necessità di introdurre ulteriori sensori sul brandeggio o di realizzare un controllo in catena aperta. Tale controllo non risulterebbe affidabile per la possibilità di una non corretta compensazione nel caso di una perdita di passo dello step motor. Si è quindi optato per il posizionare la IMU direttamente sopra la videocamera in modo tale che gli assi del suo sistema di riferimento siano allineati con gli assi di pan e tilt del brandeggio. La terza parte del setup sperimentale è formata dal PC utilizzato per l'acquisizione dei video. Tale PC non è indispensabile al fine degli obbiettivi del controllo ma ne consente un'immediata ed intuitiva verifica della bontà di quest'ultimo. Per permettere l'acquisizione video è stata installata nel PC in questione una scheda PCI/TV con ingresso video-composito. Le registrazioni sono state effettuate con l'ausilio del software VLC prodotto dalla VIDEOLAN organization e disponibile gratuitamente su web (`www.videolan.org/vlc/`).

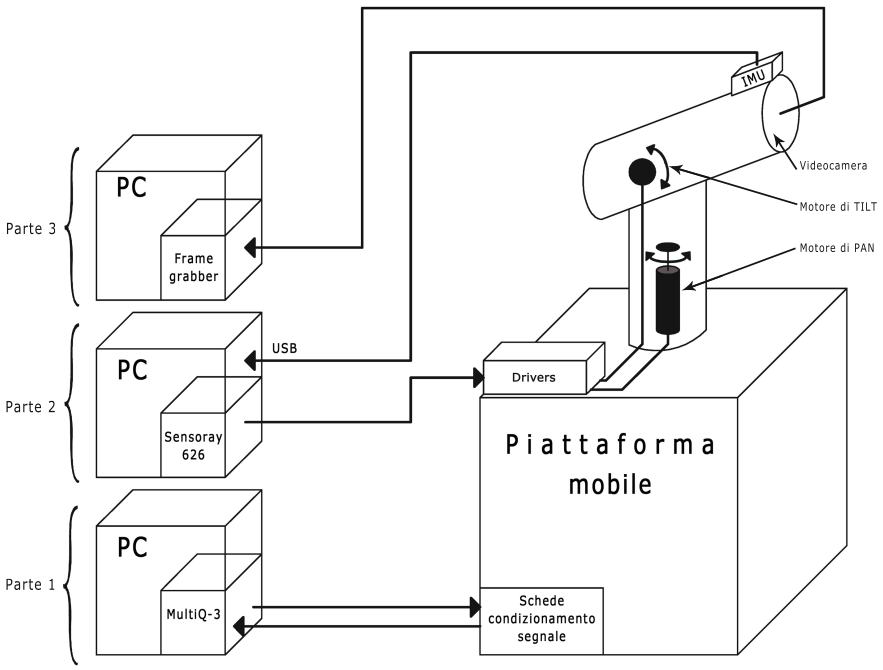


Figura 27: Schema setup sperimentale completo.

3

IDENTIFICAZIONE DEL MODELLO

3.1 INTRODUZIONE

L'obbiettivo di questo capitolo è stimare la funzione di trasferimento che lega il movimento del brandeggio al segnale di comando al fine di sintetizzare un controllo che soddisfi particolari specifiche di prontezza e stabilità.
L'identificazione dei sistemi è una scienza che si prefigge l'obiettivo di stimare modelli di sistemi reali a partire da dati sperimentali. Essa trova grande applicazione nel settore dell'automatica, ma non solo.
Si possono distinguere tre tipologie di procedure per l'identificazione di un modello:

- Procedure a scatola trasparente: il modello viene ricavato dalla descrizione delle parti costituenti il sistema mediante le leggi (fisiche) che regolano il comportamento delle parti stesse. Il problema di tale procedura nasce quando un parametro è incognito e va stimato in base ad osservazioni sperimentali;

- Procedure a scatola grigia: in questo caso il modello viene ricavato parzialmente mediante leggi fisiche e parzialmente dallo studio di ingressi e uscite reali. I parametri che descrivono il modello sono quindi in parte ricavati da relazioni matematiche che descrivono il fenomeno, e pertanto hanno un senso fisico, ed in parte ricavate sperimentalmente senza alcuna interpretazione fisica diretta;

- Procedure a scatola nera: Il modello viene ricavato sperimentalmente sulla base delle uscite ottenute da un particolare input. I parametri di tali modelli non hanno un'interpretazione fisica diretta, ma sono solo un mezzo per descrivere delle relazioni di ingresso-uscita del sistema.

Queste ultime procedure si possono affrontare con due principali metodologie:

- identificazione non-parametrica: tali tecniche pongono come obbiettivo la determinazione dell'andamento del modulo e della fase della risposta in frequenza del sistema;

- identificazione parametrica: tali tecniche, data la struttura matematica atta a modellizzare il sistema dinamico, mirano a determinare il set di parametri che meglio approssima il processo sulla base di un preciso indice di costo.

Nella maggioranza dei casi molti sistemi sono quasi impossibili da modellare tramite le leggi della fisica o se anche lo fossero tale operazione risulterebbe al quanto lunga e complicata. Per questo sempre più spesso si utilizzano procedure per l'identificazione a scatola nera o grigia anziché a scatola trasparente. Tra queste due procedure menzionate ci si concentrerà sulla quella a scatola nera con identificazione non-parametrica. Questa esigenza è data dal fatto che il sistema da modellizzare è difficilmente descrivibile da equazioni fisiche in quanto non si conoscono molte delle caratteristiche dei motori atti alla movimentazione, le inerzie e i fenomeni di attrito presenti sui singoli assi di pan e tilt ecc. . . .
In genere si sceglie un segnale di input di ``qualità´´al quale sottoporre il sistema e si misurano quindi le sequenze in uscita. Con qualità si intende la capacità del segnale di ingresso di eccitare tutte le dinamiche del sistema che si vogliono identificare (quindi opportunamente ricco) e non risentire di eventuali rumori o effetti non lineari come l'attrito. Bisogna quindi fornire al sistema un segnale che contenga una quantità elevata di informazioni a tutte le righe spettrali di interesse e sia adeguatamente potente.
In aggiunta, una procedura di identificazione deve necessariamente prevedere una fase di validazione del modello ottenuto nella quale si stabilisce la bontà del modello ottenuto ed eventualmente, se non accettabile, se ne cambia la struttura o si effettuano nuovi esperimenti. Tale fase di valutazione, si è svolta osservando la funzione di *coherence* che fornisce un'indicazione diretta delle frequenze alle quali l'identificazione presenta un'intervallo di confidenza accettabile. Se il valore di coherence è prossimo allo 0 il segnale di ingresso e uscita sono scorrelati e quindi l'uscita è influenzata prevalentemente dal rumore e quindi poco affidabile. Viceversa, se tale valore tende a 1, ingresso e uscita risultano correlate e la stima del modello è più attendibile.
Un altro aspetto molto importante è costituito dall'eliminazione delle dinamiche note dalla funzione da identificare. Infatti, molte volte, parte delle dinamiche sono note e quindi possono essere eliminate per concertare maggiormente l'attenzione su quelle incognite da identificare.

3.2 IDENTIFICAZIONE DEGLI ASSI DI PAN E TILT

Come precedentemente accennato il sistema elettromeccanico da identificare è costituito dall'asse di pan e tilt che vengono visti come una scatola nera dalla procedura di identificazione.

3.2.1 *Lo schema simulink*

Lo schema Simulink utilizzato è visibile in fig. 28 e corrisponde macroscopicamente ad un classico schema per una simulazione open-loop. La simulazione si è svolta per 60 secondi in Real-Time con

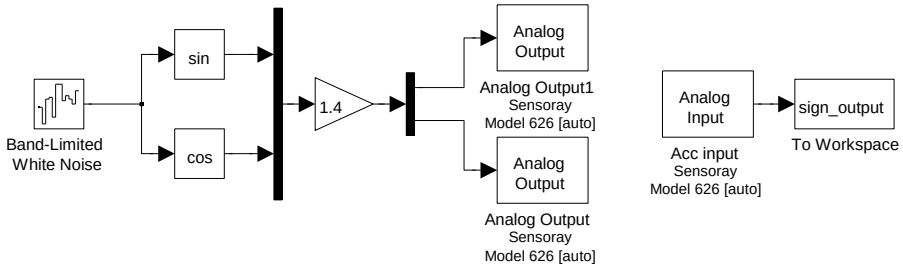


Figura 28: Schema Simulink utilizzato per la procedura di identificazione

un simple time di 1 ms utilizzando la scheda di acquisizione dati Sensoray626 descritta in sezione 2.6. L'utilizzo di tale scheda genera, come è inevitabile che sia, un errore di quantizzazione o di campionamento. Il primo si manifesta come del rumore bianco sovrapposto al segnale discretezzato mentre il secondo come un ritardo di fase approssimabile a circa metà del tempo di campionamento. Per questi motivi si è cercato di imporre il simple time più basso possibile e una quantizzazione molto fine.
Per l'esecuzione delle prove si sono valutati due segnali di ingresso "canonici": il segnale di chirp e un segnale di rumore bianco a banda limitata. Entrambi sono segnali di buona qualità ma dopo alcune prove si è osservato, attraverso una preliminare valutazione della coherence, che il rumore bianco fornisce un identificazione più fedele del sistema reale rispetto al segnale di chirp. Per tale motivo lo si è scelto come unico segnale di input per tutta l'identificazione degli assi del brandeggio. La potenza del segnale (P_{noise}) è stata impostata sperimentalmente in modo tale da impedire che il segnale di riferimento superi l'ampiezza di un passo dello step motor. Ciò porterebbe l'instaurarsi di ripple di coppia che falserebbero le misure acquisite. I valori trovati, dopo prove successive è di $P_{noise} = 0.00001$ per l'asse di pan e di $P_{noise} = 0.00002$ per l'asse di tilt.

Per quanto riguarda l'acquisizione del segnale si è utilizzato l'accelerometro ADXL335, prodotto dalla *Analog Device*, e montato su scheda *Sparkfun electronics*. Tale sensore, di cui è possibile visualizzarne un'immagine in fig. 29, è stato montato sopra la videocamera in modo da ottenere un segnale di misura che corrisponda all'effettivo movimento della videocamera stessa. Le caratteristiche dell'accelerometro sono:

- uscite analogiche su tre assi X,Y,Z;
- fondoscala di ± 3 g;

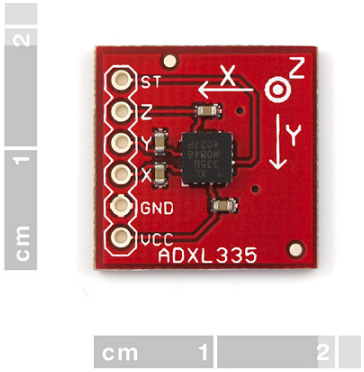


Figura 29: Accelerometro MEMS LIS2L02AS utilizzato per l'identificazione del modello

- 10 g di accelerazione massima sopportabile (senza danneggiamenti);
- sensitività (nominale) di 300 mV/g con alimentazione a 3 V;
- zero g bias level: 1.5 V;
- alimentazione possibile da 1.8 V a 3.6 V;
- basso consumo energetico: 320 A (tipico);
- banda passante di 500 Hz ottenuta con una sostituzione delle capacità sulla scheda[1].

Osservando le specifiche sulla banda passante si nota che con questo sensore è possibile acquisire dati, e quindi identificare il sistema, fino a circa 500 Hz. Ciò in prima impressione può risultare superfluo in quanto la banda passante del sensore IMU è di soli 50 Hz. Tuttavia, così facendo, si riesce ad ottenere un'identificazione su un range maggiore di frequenze che può risultare utile per applicazioni future o in caso di sostituzione del sensore IMU con uno a banda passante maggiore.

Come si può osservare dallo schema in fig. 28, il segnale di noise entra nei blocchi di coseno e seno che permettono il comando in microstepping del motore sotto test ed allo stesso tempo viene salvato nel Workspace di Matlab. La costante di 1,4 posta appena prima dell'uscita adatta il livello di tensione in modo che con la massima tensione si ottenga la massima corrente al motore come descritto in 2.5. I dati provenienti dall'accelerometro vengono acquisiti mediante il blocco di *Analog Input* fornito dalle librerie del toolbox Real-Time Windows Target e salvati nel Workspace di Matlab per le successive elaborazioni.

1 per approfondimenti vedi [20]

3.2.2 *Post-processing dei dati*

Una volta acquisiti i dati si è passati alla fase di elaborazione con l'obbiettivo di ottenere due funzioni di trasferimento che legano gli ingressi in radianti elettrici forniti dal controllo alle uscite in posizione angolare dell'asse meccanico di pan e tilt. In questo modo si vuole ottenere un modello che meglio approssima il comportamento del brandeggio.
Con riferimento a fig. 28, si vuole quindi dapprima identificare il seguente sistema reale con ingresso in posizione espresso in radianti elettrici e uscita in Volt proporzionale all'accelerazione lineare dell'accelerometro installato sulla videocamera. Quest'ultimo segnale, espresso in Volt, non rappresenta nulla di fisico per il sistema reale ma fornisce informazioni sulle accelerazioni a cui è sottoposto il sensore. Differisce dalla reale accelerazione di solo una costante, detta anche constante di *Gain* dell'accelerometro e da un offset.
Per semplificare il processo di identificazione sono state svolte le seguenti approssimazioni:

- le misure di accelerazione lineare ottenute dell'accelerometro sono state approssimate a misure di accelerazione tangenziale. Tale approssimazione può essere considerata valida solo nel caso di piccoli spostamenti angolari degli assi meccanici. Con tali spostamenti, infatti, la direzione del vettore di accelerazione tangenziale rimane praticamente uguale a quella del vettore di accelerazione lineare. Pertanto, posizionando opportunamente la telecamera e il sensore è possibile ottenere la misura desiderata da un solo asse dell'accelerometro trascurando le proiezioni sugli altri due assi;

- integrando le misure di accelerazione lineare è possibile ottenere lo spostamento angolare dell'asse meccanico che differisce dallo spostamento lineare esclusivamente per il raggio di curvatura (ciò vale solo per piccoli spostamenti). Tale costante moltiplicativa, assieme al gain dell'accelerometro, costituiscono un incognita al fine dell'identificazione.

Per prima cosa, il problema dell'offset viene eliminato eseguendo un detrend sul segnale in modo da renderlo a media nulla.
Per quanto riguarda,invece, il problema delle costanti moltiplicative, incognite o non facilmente determinabili, lo si può aggirare scalando il guadagno statico della risposta in frequenza in modo tale da ottenere il guadagno teorico desiderato. Questo guadagno è facilmente calcolabile considerando il fatto che il motore in questione è un hybrid

step motor a due avvolgimenti con passo pari a 1.8 $gradi$ comandato in microstepping[2]. Si può quindi scrivere la seguente relazione:

$$\theta_{el}\,[rad] : 2\pi = \theta_m\,[rad] : 4 \cdot \frac{1.8 \cdot 2\pi}{360} \tag{8}$$

cioè

$$\theta_{el} = \frac{4 \cdot 1.8 \cdot 2\pi}{360} \cdot \theta_m = 50 \cdot \theta_m \qquad \Rightarrow \qquad \frac{\theta_m}{\theta_{el}} = \frac{1}{50} \tag{9}$$

dove θ_{el} è la l'angolo elettrico e θ_m è l'angolo dell'asse del motore considerato.
Va poi tenuto conto del rapporto di trasmissione tra motore e asse di movimento pari a

$$\frac{\theta_{asse}}{\theta_m} = \frac{1}{7} \tag{10}$$

dove θ_{asse} è l'angolo dell'asse cinematico in questione. Si ottiene pertanto un guadagno totale tra riferimento di posizione e spostamento dell'asse comandato pari a

$$\frac{\theta_m}{\theta_{el}} \cdot \frac{\theta_{asse}}{\theta_m} = \frac{1}{50} \cdot \frac{1}{7} \qquad \Rightarrow \qquad \frac{\theta_{asse}}{\theta_{el}} = \frac{1}{50 \cdot 7} \tag{11}$$

L'algoritmo scelto per la procedura di identificazione è l'*etfe* (Empirical Transfer Function Estimate). Tale algoritmo realizza il rapporto tra le trasformate di Fourier discrete (DTFT) del segnale di ingresso e il segnale in uscita dal processo fino ad una frequenza prestabilita.

$$\hat{P}(\omega) = \frac{Y(\omega)}{U(\omega)} \tag{12}$$

Così facendo si ottiene una funzione di trasferimento tra riferimento in posizione e movimento in accelerazione. Purtroppo, tale funzione non identifica il reale sistema da controllare in quanto il sensore IMU che si utilizzerà per il feedback del controllo fornisce una misura in posizione. Per questi motivi risulta necessario aggiungere al modello identificato la dinamica nota pari ad un doppio integratore.
Per quanto riguarda il guadagno in bassa frequenza lo si è aggiustato normalizzando la funzione di trasferimento ottenuta e moltiplicandola poi per la costante in (11). Si ottiene così una funzione di trasferimento con guadagno statico proprio pari a quello atteso espresso in (11).
Le caratteristiche di ingresso-uscita così ottenute rappresentano, a meno di errori nell'identificazione, il vero sistema da controllare.

2 con tale tecnica di comando in ogni periodo di sinusoide data come riferimento il motore esegue precisamente quattro passi.

3.2.3 *Asse di PAN*

La funzione di trasferimento ottenuta dall'analisi etfe, insieme a quella calcolata con l'operazione di fitting, sono rappresentate in modulo e fase in fig. 30. Osservando la funzione di coherence si può avere

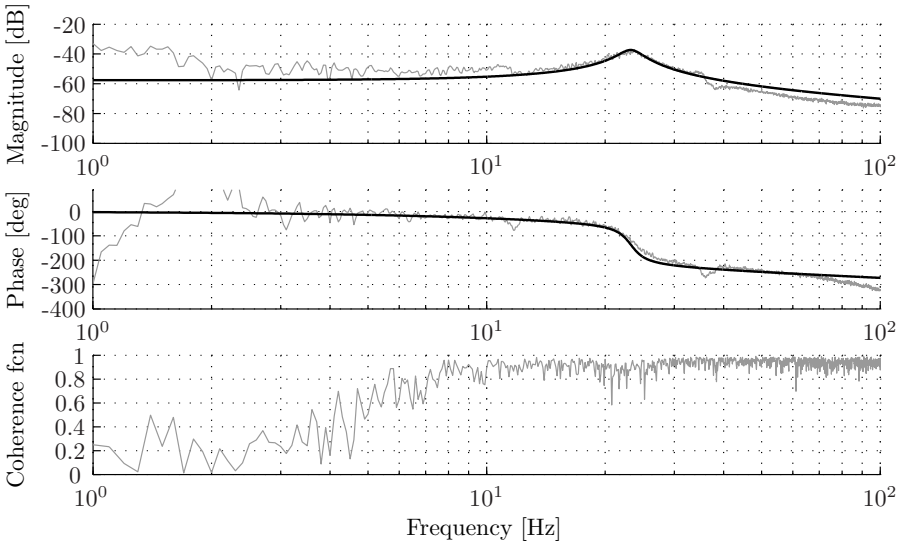


Figura 30: Funzione di trasferimento del sistema tra riferimento in posizione al motore e angolo dell'asse di pan

una stima della bontà del modello identificato. Tale funzione per frequenze superiori a 10 Hz tende a 1 il che è sintomo di una buona correlazione ingresso-uscita e conseguentemente di una buona identificazione del sistema reale. Per basse frequenze la coherence risulta essere tendente a zero, dando l'impressione di una non buona identificazione. Ciò purtroppo però è inevitabile in quanto a basse frequenze entrano in gioco fenomeni come il backlash, l'elasticità della trasmissione, o attriti statici che causano una non buona correlazione tra ingressi e uscite. Sempre con riferimento a fig. 30, si può facilmente distinguere la presenza di un doppio polo risonante a circa 23 Hz e di due coppie di poli-zeri risonanti prima e dopo il doppio polo stesso, rispettivamente a circa 11 Hz e circa 36 Hz. Ipotizzando un controllo in bassa frequenza[3], si è pensato di modellizzare il sistema tenendo conto solo del doppio polo a 23 Hz ritenendo i contributi in modulo e fase introdotte dalle altre due coppie di poli-zeri irrilevanti.

L'operazione di fitting è stata eseguita con l'apposita funzione di Matlab *invfreqz* opportunamente parametrizzata per creare una funzione di trasferimento che abbia un solo zero e due poli. La funzione

3 ipotesi veritiera dettata dal fatto che essendo la banda passante del sensore IMU pari a 50 Hz il controllo necessariamente potrà avere banda passante massima di qualche Hz (2 o 3).

di trasferimento del sistema in tempo discreto con simple time pari a 0.001 s risulta

$$P_{PAN}(z) = \frac{-0.0001629 \cdot z + 0.0001906}{z^2 - 1.96 \cdot z + 0.9809} \tag{13}$$

Si nota che il guadagno statico risulta circa -57 dB il che è vicino ma non uguale al guadagno atteso che sarebbe pari a

$$K_{PAN} = 20 \cdot \log_{10}\left(\frac{1}{50 \cdot 7}\right) = -50.88 \text{ dB} \tag{14}$$

Questo è dovuto al fatto che vi sono alcuni errori nel fitting della caratteristica in uscita dall'algoritmo di etfe causati probabilmente dalla rumorosità di quest'ultima. Per quanto riguarda le alte frequenze il fitting viene eseguito molto fedelmente anche grazie alla parametrizzazione della funzione *invfreqz* che consente di concentrare un maggior numero di campioni attorno alla coppia di poli risonanti.

3.2.4 *Asse di TILT*

La funzione di trasferimento, relativamente all'asse di Tilt, ottenuta dall'analisi etfe, insieme a quella calcolata con l'operazione di fitting, sono rappresentate in modulo e fase in fig. 31.

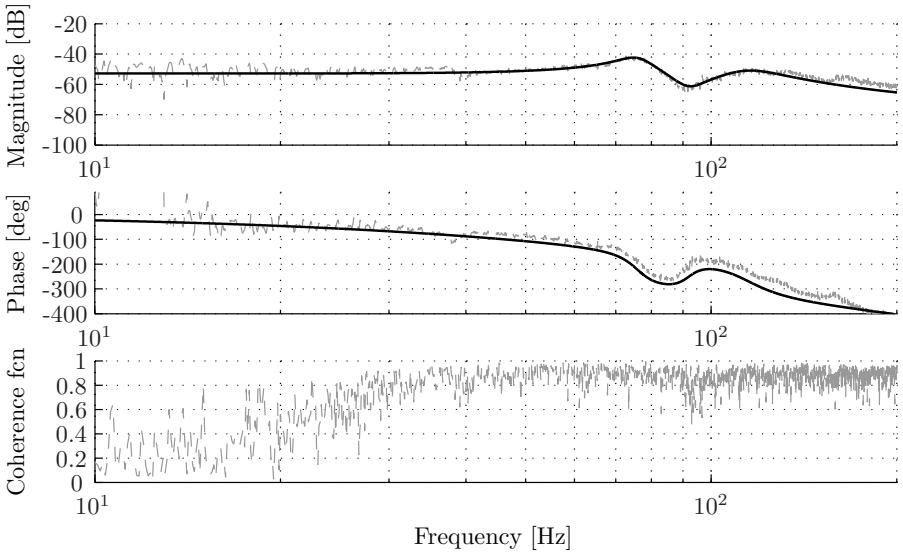


Figura 31: Funzione di trasferimento del sistema tra riferimento in posizione al motore e angolo dell'asse di tilt

Osservando la funzione di coherence si nota che per frequenze superiori a 30 Hz si ottiene una buona correlazione ingresso-uscita e conseguentemente di una buona identificazione del sistema reale. Ciò, come nel caso dell'asse di pan non si verifica per basse frequenze dove la coherence è tendente a zero. Si distingue, inoltre, la presenza

di un doppio polo-zero risonante attorno agli 80 Hz e di un doppio polo a circa 115 Hz. Assumendo l'ipotesi veritiera di un controllo in bassa frequenza fatta precedentemente, si è pensato di modellizzare il sistema tenendo conto solo di questi due contributi. In aggiunta, si osservano altre due coppie polo-zero, una a circa 37 Hz e una a circa 160 Hz i cui contributi in modulo e fase vengono trascurati.
Per tali motivi si è parametrizzata la funzione di fitting in modo da rappresentare il sistema come una funzione di trasferimento a quattro zeri e cinque poli. La FDT del sistema in tempo discreto ottenuta con un simple time pari a 1 ms

$$P_{TILT}(z) = 8.45e-005 \cdot \cdot \frac{z^4 - 0.0005847z^3 + 0.001363z^2 - 0.001325z + 0.0005228}{z^5 - 3.838z^4 + 6.435z^3 - 5.789z^2 + 2.795z - 0.5759} \quad (15)$$

Come nel caso dell'asse di pan il guadagno statico risulta leggermente diverso da quello atteso e precisamente pari a -52 dB per gli stessi motivi menzionati precedentemente.
Per quanto riguarda le alte frequenze il fitting viene eseguito molto fedelmente anche grazie alla parametrizzazione della funzione *invfreqz* che consente di concentrare un maggior numero di campioni attorno tra 10 Hz e 130 Hz cioè dove sono posizionati le dinamiche da seguire.

I programmi di identificazione creati in ambiente Matlab sono consultabili in appendice A.1.2 e A.1.3 rispettivamente.

4

PROGETTAZIONE DEL CONTROLLO PER LA COMPENSAZIONE DEL MOTO ONDOSO

4.1 INTRODUZIONE

In questo capitolo si vuole descrivere la realizzazione del sistema di controllo per gli attuatori del brandeggio Ulisse COMPACT, al fine di compensare le oscillazioni di yaw e pitch della base su cui è montato. Tale compensazione si basa sull'utilizzo di misure inerziali ricevute dalla scheda IMU al fine di generare opportuni segnali di controllo alla ISP, costituita per l'appunto dal brandeggio. La stabilizzazione dell'immagine trattata in questo capito è una stabilizzazione esclusivamente a livello hardware, essendo basata esclusivamente da misure di sensori e dal controllo opportuno degli attuatori e senza alcun contributo a livello di elaborazione video.
Nella prima parte di questo capitolo viene affrontata la scelta del tipo di controllo da implementare e la sua successiva realizzazione pratica, con tutte le accortezze del caso. Infine vengono illustrate e analizzate le misure sperimentali ottenute da alcune prove utilizzando il simulatore di moto ondoso, precedentemente descritto nel paragrafo 2.7.

4.2 SCELTA DEL TIPO DI CONTROLLORE

Una prima osservazione che è necessario effettuare riguarda l'analisi cinematica del brandeggio. Infatti, quest'ultimo costituisce un *gimbal* a due gradi di libertà: pan e tilt. I movimenti che si dovrebbero compensare, per ottenere una buona stabilizzazione delle immagini, invece, sono tre: yaw, pitch e roll. Si può capire che, a livello hardware, in nessun modo si può andare a compensare eventuali oscillazioni di roll sull'asse x del brandeggio. Attraverso il seguente sistema di controllo, è possibile stabilizzare solamente il centro dell'immagine ma non, ad esempio mantenere, fissa la linea dell'orizzonte. Tale aspetto e una sua possibile soluzione verranno descritti nell'ultimo capitolo di questo elaborato.
Gli assi di rotazione che è possibile compensare sono indipendenti e ortogonali tra loro e rispetto al LOS. Questo ha permesso di realizzare dei controlli indipendenti per pan e tilt, non essendoci mutui accoppiamenti. Come ampiamente descritto nel capitolo precedente i due sistemi da controllare sono costituiti da motori passo-passo che movimentano, attraverso delle cinghie elastiche, gli assi cinematici del *gimbal* su cui è montata la struttura in cui è alloggiata la videocame-

ra per la videosorveglianza. L'obbiettivo del controllo è di mantenere puntata la telecamera verso un preciso riferimento, evitando che quest'ultimo esca dal campo di vista. Sostanzialmente il controllo deve mantenere un certo riferimento angolare fisso degli assi di pan e tilt, indipendentemente dal movimento della base del brandeggio.
Nella progettazione dei due anelli di controllo è necessario tenere presente il fatto che i motori passo-passo, utilizzati per le movimentazioni di pan e tilt, sono attuatori comandati in posizione. Inoltre i comandi devono avere velocità di variazione limitate in modo da garantire il mantenimento del punto di lavoro dei motori all'interno della curva di pull-out, evitando così indesiderate perdite di passo. Un modo semplice per applicare tale limitazione consiste nell'utilizzo di un'integratore con ingresso saturato per generare il segnale di comando in posizione (come si può vedere in fig. 32).

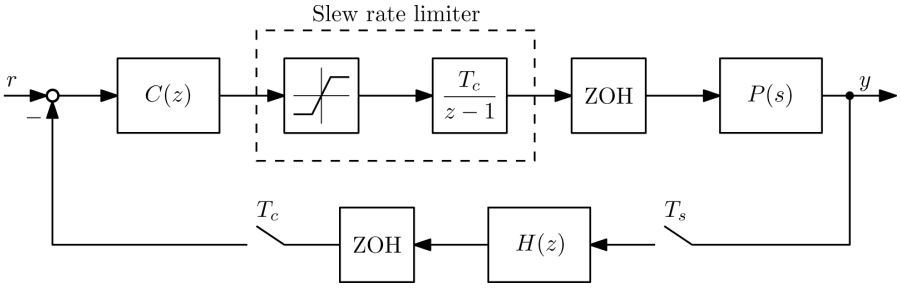


Figura 32: Schema a blocchi semplificato del loop di controllo per gli i motori passo-passo ibridi utilizzati per la stabilizzazione degli assi di pan e tilt del brandeggio. Nella figura, $P(s)$ rappresenta la dinamica di pan/tilt dal comando in angoli elettrici all'uscita in posizione angolare dell'asse del brandeggio (tali funzione è stata identificata nel precedente capitolo) mentre $C(s)$ costituisce il controllo che si desidera implementare.

Dopo tale accorgimento la progettazione dei due sistemi di controllo può avvenire utilizzando metodi convenzionali, tenendo conto, però, che il sistema da controllare ora presenta un integratore aggiuntivo all'interno dalla funzione di trasferimento.
Le funzioni di trasferimento ottenute nel precedente capitolo, con l'aggiunta dell'integratore, sono il punto di partenza per la realizzazione del controllo desiderato (vedi fig. 33). Da tali diagrammi di Bode e dalle considerazioni precedenti è facile dedurre che il controllo ricercato risulta essere un controllo di velocità e un semplice controllore proporzionale è sufficiente per stabilizzare l'anello di controllo in feedback con un soddisfacente margine di fase di circa 90° alla frequenza di attraversamento desiderata.

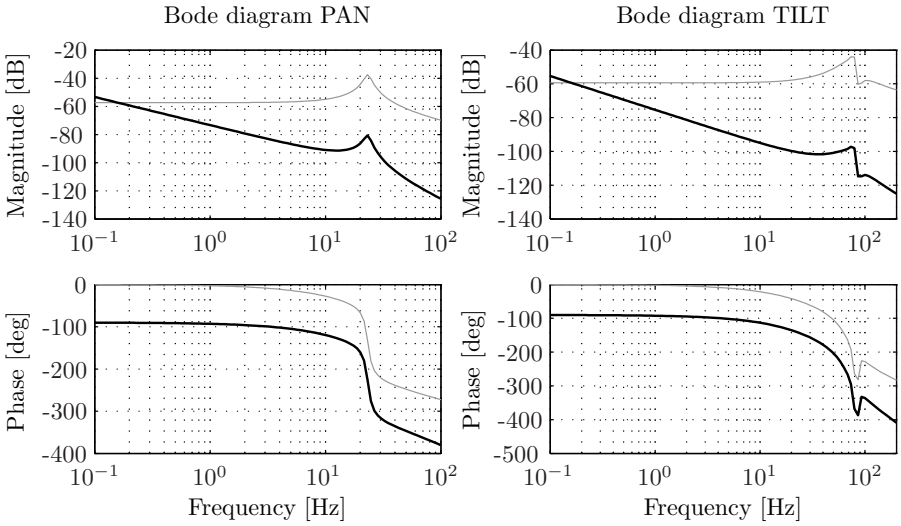


Figura 33: Confronto tra il diagramma di Bode che descrive la dinamica del sistema $P(s)$ (in nero) e quello del processo $P(s)$ con l'integratore discreto (in grigio).

La scelta della banda passante da imporre ai due anelli di controllo deve essere tale da permettere la compensazione della maggior parte delle componenti in frequenza che costituiscono un moto ondoso. Si è, quindi, eseguita una ricerca al fine di trovare dai dati scientifici riguardanti la densità spettrale dell'energia dei moti ondosi. I risultati che trovati [2],[16] sono molto simili tra loro e un grafico che li rappresenta molto bene viene riportato in fig. 34.

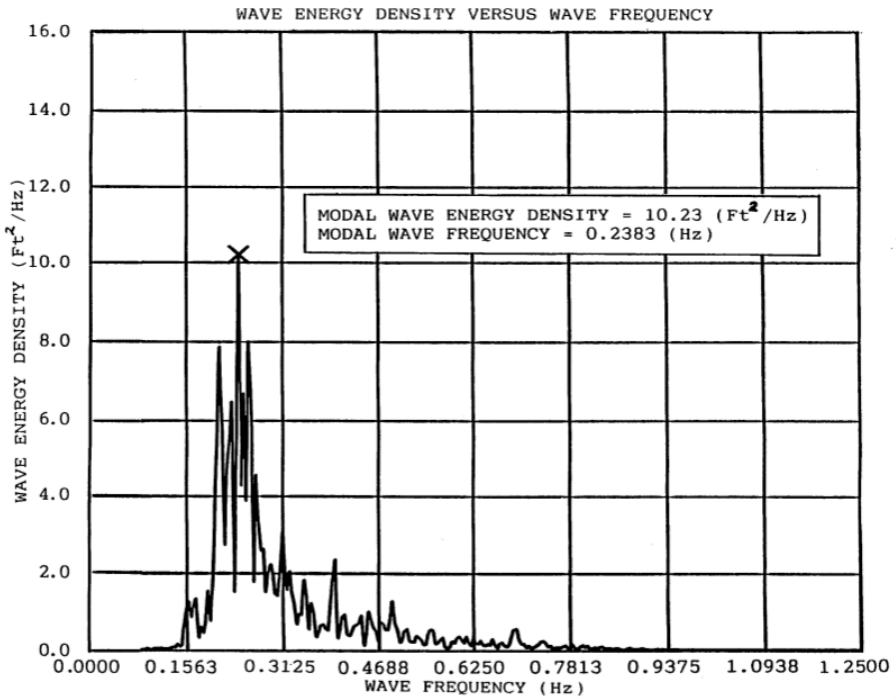


Figura 34: Grafico che riporta un andamento tipico della densità di energia di un'onda in funzione della frequenza.

Come si può notare, la densità spettrale dell'energia del moto ondoso presenta un unico picco nell'intorno di 0.2Hz. È opportuno, quindi, andare a realizzare degli anelli di controllo che presentino una banda passante, se possibile, superiore a tale frequenza in modo tale che le componenti con maggiore energia (conseguentemente quelle che determinano le oscillazioni maggiori) possano essere compensate adeguatamente dal controllo.
Scelto di realizzare un controllore proporzionale in velocità, la determinazione dei giusti guadagni proporzionali è stata ottenuta imponendo l'attraversamento a 0dB, alla frequenza prestabilita,della funzione di anello $L(s)$ per ognuno dei controlli.

$$L(s) = C(s) \cdot P(s) = Kp \cdot P(s) \tag{16}$$

Per imporre tale attraversamento sono stati svolti i seguenti calcoli:

$$Kp = \frac{1}{|P(j\omega_a)|} \tag{17}$$

dove ω_a risulta essere la pulsazione di attraversamento.
Le valori delle frequenze di attraversamento prescelte, per gli assi da controllare, sono rispettivamente 0.75Hz per l'asse di pan e 0.2Hz per l'asse di tilt. Tali scelte sono state fatte sulla base, non solo degli andamenti della densità spettrale di energia delle onde, ma anche sulla base delle prestazioni che si potevano ottenere da tali controlli. C'è da tenere presente, infatti, che aumentare la banda passante dell'anello di controllo rende il sistema più reattivo e pronto ma anche più sensibile ai disturbi. Per l'appunto la banda passante dell'anello di controllo di tilt è stata mantenuta abbastanza bassa per evitare questo tipo di problematica.

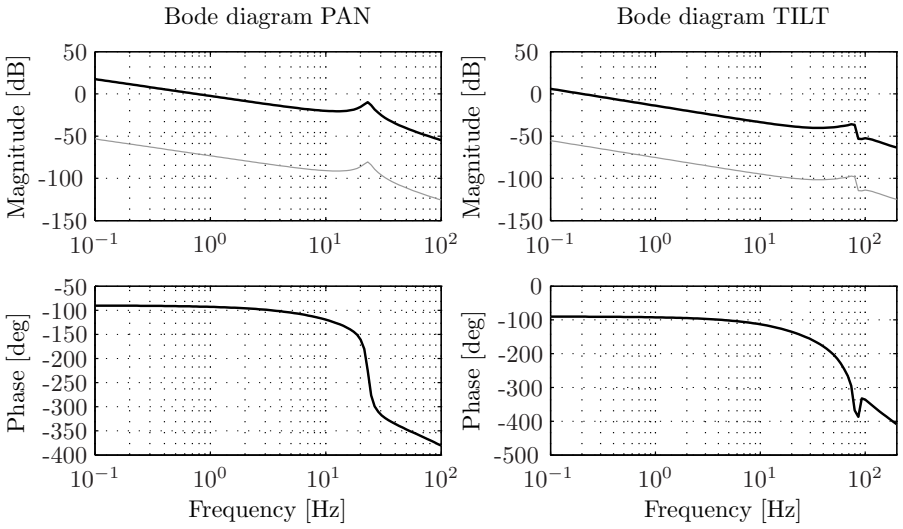


Figura 35: Confronto tra il diagramma di Bode del sistema effettivamente da controllare (in grigio) e il diagramma di Bode della funzione di anello compresa di controllore proporzionale(in nero).

4.3 ALGORITMO DI SENSOR FUSION

La stima dell'orientazione della videocamera viene fornita dal sensore IMU che implementa in se un algoritmo di sensor fusion. La scheda dispone al suo interno di un microcontrollore che permette la lettura dei dati provenienti da 3 sensori a bordo (giroscopio, accelerometro e magnetometro) e ne realizza una prima elaborazione attraverso un algoritmo di sensor fusion. Tutto questo per fornire in uscita dei valori in yaw-pitch-roll che sono più semplici da trasferire ed analizzare.
Sensor fusion è quel processo attraverso il quale dati di singoli sensori o derivanti da più sensori di diversa natura, vengono combinati assieme in modo che le informazioni risultanti siano migliori rispetto a quelle che si sarebbero potute ottenere utilizzando i singoli sensori individualmente. Il termine migliore in alcuni casi può significare più accurato, più completo, più affidabile.
Nelle pagine seguenti si vuole andare ad illustrare l'implementazione di un particolare algoritmo di sensor fusion DCM (Direction-Cosine-Matrix) basato su misure inerziali provenienti dalla scheda IMU montata sulla telecamera.

4.3.1 *Matrici di rotazione-DCM*

In geometria, una rotazione è una trasformazione 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, viene definito come centro o asse di rotazione. Più precisamente, in termini geometrici, una rotazione è un'isometria di uno spazio euclideo.
Secondo il teorema di rotazione di Eulero la rotazione generica di un corpo rigido (o di un sistema di riferimento) rispetto ad un punto fisso può essere descritta in modo univoco da un minimo di tre parametri. Tuttavia, a seconda delle applicazioni, ci sono diversi tipi di rappresentare di una rotazione, quali: matrici di rotazione, angoli di Eulero, quaternioni, ecc. . . . Molte di queste rappresentazioni utilizzano più dei tre parametri minimi necessari, anche se qualsiasi tipo di descrizione rimane comunque a tre gradi di libertà, e ognuna presenta vantaggi e svantaggi nell'utilizzo.
La scelta di utilizzare le matrici di rotazione nello sviluppo dell'algoritmo di sensor fusion ricade nel fatto che tale rappresentazione è priva di approssimazioni e singolarità. Inoltre si integra bene con i dati raccolti dai vari sensori e con la successiva fase di realizzazione del controllo. Una matrice di rotazione, detta anche DCM, è una rappresentazione di una rotazione dello spazio euclideo definita da una matrice 3×3. Tale matrice può pero essere anche interpretata come descrizione dell'orientazione di un sistema di coordinate cartesiane

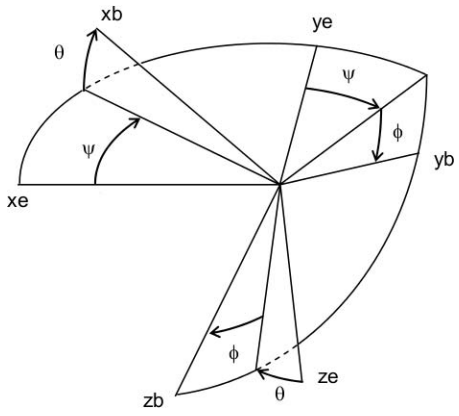


Figura 36: Rotazione tra riferimento di terra (Earth Frame) e riferimento del sensore (Body Frame)

rispetto ad un altro. Nel caso in esame, tali matrici verranno utilizzate per descrivere l'orientazione del sistema di riferimento fisso con il sensore (Body Frame) rispetto ad un sistema di coordinate solidale con il suolo (Earth Frame) (vedi fig. 36). Infatti, queste matrici sono costituite in modo tale che le colonne sono una rappresentazione dei versori[1] del sistema di riferimento, di cui si vuole esprimere l'orientazione, rispetto ad un altro sistema di coordinate, considerato come riferimento fisso, vedi (18).

$$R = \begin{array}{c} \\ \mathbf{xe} \\ \mathbf{ye} \\ \mathbf{ze} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{xb} & \mathbf{yb} & \mathbf{zb} \end{array} \\ \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \end{array} \quad ; \quad R^{-1} = \begin{array}{c} \\ \mathbf{xb} \\ \mathbf{yb} \\ \mathbf{zb} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{xe} & \mathbf{ye} & \mathbf{ze} \end{array} \\ \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \end{array} \tag{18}$$

In questo modo un vettore appartenente al primo sistema di riferimento può essere trasformato nell'altro sistema di coordinate semplicemente moltiplicandolo per tale matrice di rotazione.

$$\mathbf{y^e} = R \cdot \mathbf{y^b} \tag{19}$$

dove $\mathbf{y^e}$ è un vettore definito secondo il sistema di riferimento di terra e $\mathbf{y^b}$ è il medesimo vettore definito secondo il sistema di riferimento del sensore. L'operazione inversa invece richiede l'utilizzo della matrice di rotazione inversa (R^{-1}) che per le particolari proprietà di tale rappresentazione, descritte in seguito, coincide con la matrice trasposta.

$$\mathbf{y^b} = R^{-1} \cdot \mathbf{y^e} \tag{20}$$

$$R \cdot R^{-1} = R^T \cdot R = I \tag{21}$$

Tali matrici presentano due proprietà chiave che verranno utilizzate in seguito all'interno dell'algoritmo di sensor fusion. La prima è l'or-

1 vettori di modulo unitario.

togonalità, secondo la quale due vettori sono perpendicolari in un sistema di riferimento allora lo saranno in ugual modo in ogni sistema di riferimento, purché tra i due sistemi di riferimento sussista un legame determinato da una matrice di rotazione. La seconda proprietà è simile alla prima ma riguarda le lunghezze dei singoli vettori che rimangono invariate al variare del sistema di riferimento scelto.
Le matrici di rotazione presentano 9 elementi, tuttavia solo 3 di loro sono indipendenti. Questo deriva dal fatto che, affinché le colonne definiscano un sistema in coordinate cartesiane è necessario che tale matrice sia ortonormale, ossia che valgano le seguenti proprietà: ortogonalità tra coppie di colonne (o righe) e somma dei quadrati delle singole colonne (o righe) pari a 1. In formule diventa:

$$|\mathbf{xb}| = |\mathbf{yb}| = |\mathbf{zb}| = 1 \tag{22}$$

$$\mathbf{xb} \perp \mathbf{yb} \quad ; \quad \mathbf{xb} \perp \mathbf{zb} \quad ; \quad \mathbf{yb} \perp \mathbf{zb} \tag{23}$$

Questo fa si che i gradi di libertà scendano da 9 a 3 come desiderato. Tutto ciò rende oltretutto la matrice asimmetrica.

Convenzioni assi Per descrivere l'orientazione di un corpo rigido è necessario definire adeguati sistemi di riferimento. Nella maggior parte dei problemi riguardanti questo ambito, vengono utilizzati due sistemi di coordinate: uno fisso a terra, considerato durante l'analisi come un sistema di riferimento inerziale, e un secondo fisso con il corpo rigido di cui si vuole conoscere l'orientazione (come già definito in precedenza).
Viene inoltre utilizzata la convenzione di definire come asse Z quello con direzione uguale al vettore dell'accelerazione di gravità e verso tale per cui quest'ultimo è positivo (vedi fig. 36).
L'orientazione del riferimento mobile rispetto al riferimento fisso può essere definito ipotizzando di partire dal riferimento fisso ed eseguire le seguenti rotazioni al fine di sovrapporre tale riferimento al riferimento mobile:

- rotazione intorno all'asse z di un angolo di yaw pari a α
- rotazione intorno all'asse y di un angolo di pitch pari a β
- rotazione intorno all'asse x di un angolo di roll pari a γ

La composizione di queste tre rotazioni permette di ottenere la medesima matrice DCM di rotazione descritta nei capitoli precedenti per la cinematica del manipolatore parallelo (vedi matrice (3)).

4.3.2 *Procedura di sensor fusion*

L'idea di base di questo algoritmo è l'applicazione delle leggi della cinematica per quanto riguarda rotazioni di corpi rigidi. Da ciò, si

ottiene che le matrici di rotazione, che definiscono l'orientazione del sensore e quindi della telecamera, possono essere ottenute integrando le equazioni differenziali non lineari che descrivono le rotazioni. La cinematica analizza le rotazioni di un corpo rigido e come queste trasformino una configurazione rigida in un'altra. Tali trasformazioni possono essere descritte come composizioni e integrazioni di matrici. L'integrazione numerica, tuttavia, è l'operazione più critica da implementare in modo digitale, perché introduce degli errori che discostano il risultato finale da quello teorico. Una teorica integrazione dei segnali provenienti dai giroscopi, produrrebbe esattamente la corretta matrice di rotazione, ipotizzando comunque di avere segnali esatti dai giroscopi. Inoltre, l'integrazione numerica introduce due tipi di errori:

- errori di integrazione, dovuti al fatto che si sta eseguendo una integrazione discreta e quindi con un tempo di quantizzazione finito, seppure piccolo. Ciò provoca l'introduzione di un'errore che è proporzionale all'accelerazione della rotazione;
- errori di quantizzazione, dovuti all'utilizzo di calcolatori digitali e di conseguenza ad una conversione analogico-digitale dei segnali che ne determinano una loro rappresentazione finita.

Tali errori numerici possono violare le proprietà precedentemente descritte che devono possedere le matrici di rotazione. Ad esempio la matrice risultante dopo la procedura di integrazione potrebbe non essere più ortonormale.
Dal punto di vista operativo l'algoritmo di sensor fusion lavora sostanzialmente nel seguente modo:

1. i giroscopi vengono utilizzati come prima sorgente da cui ricavare le informazioni riguardanti l'orientazione. I giroscopi producono segnali elettrici proporzionali alla velocità di rotazione del sensore intorno ai rispettivi assi del sistema di riferimento fisso con esso. Essendo le rotazioni non commutative l'ordine delle singole rotazioni conta e non è pertanto possibile integrare i singoli segnali dei giroscopi al fine di ottenere gli angoli desiderati. È necessario quindi analizzare accuratamente la cinematica delle rotazioni e ricavare la velocità di variazione di un vettore di orientazione rispetto alla sua velocità angolare. Vengono poi integrate le equazioni differenziali non lineari della cinematica che forniscono la variazione nel tempo dell'orientazione del sensore, e quindi del corpo rigido su cui è montato, rispetto alla sua posizione attuale. Tutto questo viene svolto ad elevate velocità (40Hz), il che è sufficiente per realizzare un buon loop di controllo del sistema complessivo;
2. riconoscimento degli errori numerici dell'integrazione. Drift e offset dei giroscopi gradualmente determinano un accumularsi

di errori negli elementi delle matrici DCM. Tali errori possono violare i vincoli che le matrici di DCM devono soddisfare, come già spiegato precedentemente. È necessario quindi, eseguire piccoli ma regolari aggiustamenti agli elementi delle matrici per soddisfare tali vincoli. É necessario andare a risolvere il problema del drift delle misure. La procedura che si è voluto implementare per raggiungere questo scopo prevede i seguenti passi:

- utilizzo di vettori di riferimento per determinare l'errore di orientazione tra l'orientazione vera e quella ottenuta elaborando i dati dei giroscopi;
- retroazionare l'errore di rotazione attraverso un controllore proporzionale integrale (PI) a fine di generare un aggiustamento della velocità di rotazione per i giroscopi;
- sommare l'uscita del controllore PI all'attuale segnale dei giroscopi.

Il principale requisito che deve possedere un vettore di riferimento per l'orientazione è che non deve essere affetto da drift. Le sue prestazioni dinamiche non sono importanti perché, in tali condizioni, ci sono i giroscopi che forniscono una buona affidabilità nella stima dell'orientazione. Attraverso i sensori aggiuntivi montati sulla IMU, quali magnetometri e accelerometri, è possibile ricavare 2 diversi vettori di riferimento. I magnetometri vengono utilizzati come riferimento per la proiezione sul piano orizzontale dell'asse X (asse di roll) del sensore mentre gli accelerometri forniscono un vettore di riferimento per l'asse Z.
Per ciascuno dei due vettori di riferimento, l'errore di orientazione viene determinato eseguendo il prodotto vettoriale tra il vettore misurato e il vettore stimato dalla matrice DCM. Retroazionandolo attraverso il controllore PI la rotazione stimata, quest'ultima è gradualmente forzata ad inseguire il vettore di riferimento compensando così il drift dei giroscopi.
L'errore del vettore di riferimento torna indietro all'interno dei segnali del giroscopi attraverso la matrice DCM quindi la correzione dipende dall'orientazione della IMU. Quindi, per esempio, il vettore di riferimento dei magnetometri può correggere o X ,Y, Z o combinazioni di questi, e ciò dipende per l'appunto dall'orientazione degli assi rispetto al riferimento di terra.

Per una trattazione più approfondita di tale algoritmo di sensor fusion si rimanda a [10]. I listati del firmware installato all'interno del microprocessore della scheda che implementa l'algoritmo appena descritto, sono riportati in appendice A.2.

4.4 REALIZZAZIONE DEL SISTEMA DI CONTROLLO

Il trasferimento delle informazioni riguardanti l'orientazione della scheda IMU e, di conseguenza, della videocamera vengono trasferite al calcolatore attraverso un protocollo di trasmissione seriale asincrono. Il firmware caricato all'interno della scheda è stato configurato in modo tale che, non appena i dati sono pronti, una successione di misure (yaw, pitch e roll) venga inviata al computer.

4.4.1 *Schema del controllo di Homing*

All'inizio di ogni prova sperimentale il controllo non è in grado di conoscere in che modo risulta essere orientato il brandeggio. La posizione iniziale può essere anche molto differente dalla posizione di home da cui si desidera partire con la prova. Tale posizione di home è stata definita empiricamente in modo da garantire, inizialmente, la centratura del target visivo realizzato.
La presenza di un elevato errore tra riferimento di home e posizione iniziale del brandeggio, utilizzando il semplice controllo prima descritto, in alcuni casi può determinare l'imposizione di una velocità di rotazione iniziale che supera i limiti imposti dalla curva di pull-in del motore, determinando perdite di passo e vibrazioni indesiderate. Si è così implementato un controllo di homing che ha come obbiettivo, esclusivamente, la movimentazione del brandeggio al fine di portarlo in prossimità della posizione di riferimento desiderata. Non si vogliono ottenere alcun tipo di prestazione da tale controllo a parte evitare la perdita del passo. Si è optato, quindi, di realizzare il controllo di homing con la medesima struttura del controllo precedentemente descritto modificando esclusivamente la saturazione del comando in velocità con il limite di velocità definito dalla curva di pull-in degli step motor.

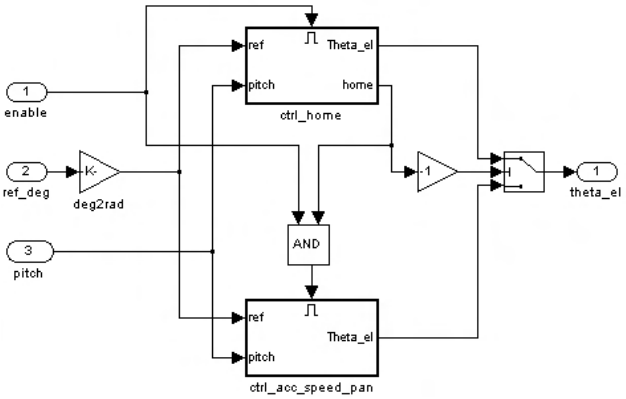


Figura 37: Schema a blocchi per lo switch tra controllo di homing e controllore normale.

Non appena l'errore di posizione si trova all'interno di una certa banda prestabilita, il segnale di *home* passa da 0 a 1 (vedi fig. 37)e il controllo commuta da quello di home a quello regolare che presenta, invece, una saturazione di velocità pari al limite massimo ottenibile dal motore (curva di pull-out).

4.5 SCHEMA SIMULINK COMPLESSIVO PER IL CONTROLLO DEGLI ASSI

I controlli relativi agli assi di pan e tilt presentano la medesima struttura e variano esclusivamente i guadagni dei controllori e alcune costanti moltiplicative. Con riferimento a fig. 38, nella catena di retroazione dei segnali in uscita dalla scheda IMU sono state aggiunte due costanti moltiplicative per convertire tali segnali da gradi a radianti (il controllo viene realizzato utilizzando grandezze in radianti) e per realizzare una prima semplice compensazione di errori derivanti da una mancante calibrazione della IMU.
Il segnale in uscita dai controllori è un riferimento di posizione in angoli elettrici dei motori. Tale riferimento viene utilizzato per comandare in microstepping gli attuatori. Infine viene fatto notare la presenza di una serie di blocchetti (vedi in alto a destra in fig. 38) per la sincronizzazione dell'acquisizione delle misure tra i diversi calcolatori utilizzati nelle prove sperimentali (tale aspetto verrà approfondito nel capito della elaborazione video).

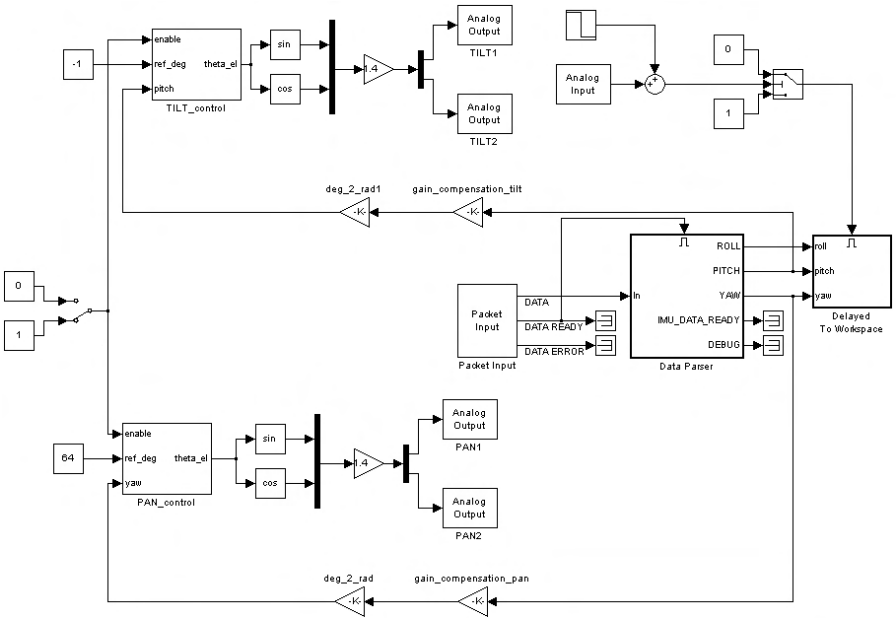


Figura 38: Schema Simulink complessivo per il controllo degli assi di tilt (sopra) e pan (sotto).

4.6 PROVE SPERIMENTALI

Al fine di testare la bontà e le prestazioni del sistema di stabilizzazione dell'immagine appena descritto, sono state svolte delle prove preliminari fornendo dei semplici riferimenti al simulatore di moto ondoso di tipo sinusoidale. In questo modo si è potuto verificare, separatamente, i funzionamenti dei singoli controlli.
Si è voluto poi testare il sistema in condizioni operative realistiche (off-shore in mare). Si è deciso di reperire delle registrazioni di dati riguardanti il movimento e l'orientazione di due differenti supporti galleggianti (in cui potrebbe essere montato il brandeggio): una nave per il pattugliamento della costa e una boa costiera. Tali registrazioni sono state estratte da [16] e sono state utilizzate come riferimenti per generare i movimenti del simulatore al fine di realizzare un moto ondoso il più possibile realistico.
Per ottenere una valutazione quantitativa della stabilizzazione dell'immagine, ogni prova effettuata è costituita, sostanzialmente, dalla movimentazione del simulatore (secondo uno dei riferimenti appena descritti) con montato sopra il brandeggio e dalla raccolta e successivo confronto delle misure inerziale fornite dalla IMU nel caso in cui il controllo sviluppato viene attivato o meno. Ulteriori analisi e considerazioni sono state effettuate eseguendo una post-elaborazione dei video della videocamera raccolti durante l'esecuzione di queste prove, tale aspetto verrà trattato ampiamente nell'ultimo capito.

4.6.1 *Risultati sperimentali*

Il riferimento sinusoidale utilizzato nella prova preliminare è costituito dalle seguente sequenza di movimentazione:

- posizione di *home* (5 sec);
- movimento di solo moto di yaw/pan con ampiezza massima pari a 10° (30 sec);
- posizione di *home* (5 sec);
- movimento di solo moto di roll con ampiezza massima pari a 5° (30 sec);
- posizione di *home* (5 sec);
- movimento di solo moto di pitch/tilt con ampiezza massima pari a 5° (30 sec);
- posizione di *home* (5 sec);
- movimento di movimento simultaneo di yaw e pitch con le relative ampiezze precedenti (30 sec).

I risultati sperimentali, riguardanti le misure inerziali della IMU, ottenuti da tale prova, appena descritta, sono riportati in fig. 39 mentre i relativi valori delle ampiezze picco-picco dei segnali sono riportati in tab. 2.

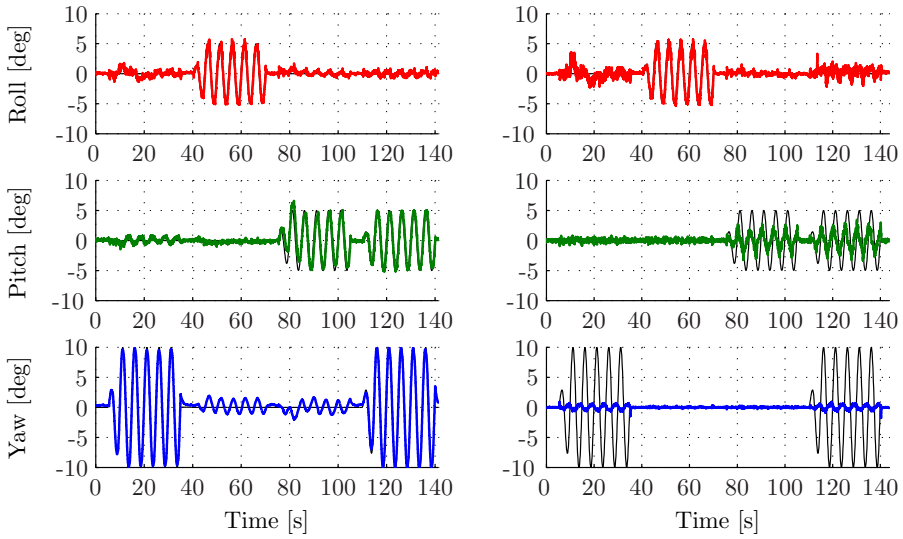


Figura 39: Andamenti di roll, pitch, yaw ottenuti dalla scheda IMU montata sulla videocamera nel caso di controllo per la stabilizzazione disattivato (a sinistra) e attivato (a destra). I riferimenti di orientazione utilizzati dal simulatore per la generazione del moto (linee nere) sono di tipo sinusoidale per ogni rotazione.

Tabella 2: Tabella delle ampiezze picco-picco delle sinusoidi dei segnali di roll, pitch, yaw ottenute con il riferimento sinusoidale. Tali ampiezze sono state calcolate a partire dai segnali prevenienti dalla IMU, fittati attraverso un algoritmo ai minimi quadrati per determinare la sinusoide che meglio li approssima.

Ampiezze picco-picco

Angolo	Simbolo	No comp. [deg]	Comp. [deg]
Roll	σ_R	10	9.95
Pitch	σ_P	10	3.74
Yaw	σ_Y	20	0.96

Un primo aspetto da tenere presente sono i valori esatti delle ampiezze nel caso di controllo non attivo. Ciò è dovuto al fatto che i guadagni in retroazione, al fine di eseguire una prima semplice compensazione di una mancata calibrazione dei sensori, sono stati scelti in maniera tale da ottenere queste ampiezze prestabilite e pari ai riferimenti.
Come era ovvio aspettarsi,inolte , nelle prove dove la stabilizzazione

dell'immagine non è attiva le misure inerziali della IMU sono praticamente sovrapponibili con i riferimenti di moto ondoso, questo perché nessuna azione di controllo viene eseguita. Nel caso invece in cui la compensazione delle oscillazioni è attivata si nota una diminuzione del movimento per quanto riguarda i movimenti di pitch e yaw della telecamera. Ovviamente, per i movimenti di roll la situazione rimane, invece, praticamente invariata rispetto al caso non compensato perché la stabilizzazione dell'immagine rispetto a tali movimenti non può essere effettuata meccanicamente attraverso il brandeggio. A confermare tali considerazioni, vengono riportate nella tab. 2 i valori di deviazione standard espresse in gradi dei segnali ottenuti dalla varie prove. È possibile constatare come la stabilizzazione della telecamera è verificata dalla diminuzione della deviazione standard dei segnali della IMU per quanto riguarda l'asse di yaw e pitch, ed inoltre, come la compensazione delle oscillazioni di yaw è migliore rispetto a quelle di pitch. Una possibile spiegazione a questo fatto può essere ricercata nella minore reattività dell'anello di controllo dell'asse di yaw alle misure rumorose rispetto all'anello di controllo per i movimenti di tilt (questo può essere dovuto, avendo implementato lo stesso tipo di controllo, alla maggiore inerzia posseduta) o alla effettiva maggiore rumorosità delle misure di pitch rispetto a quelle di yaw. Tale ultimo aspetto è confermato dai dati ottenuti dalla IMU in una prova dove il controllo è stato attivato ma il brandeggio è stato mantenuto fisso (vedi fig. 40).

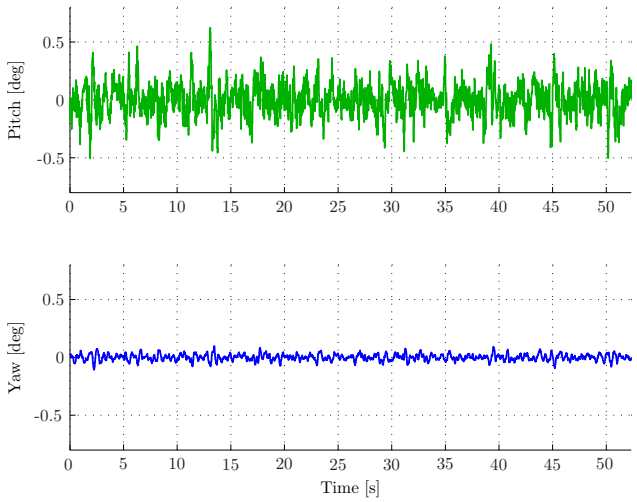


Figura 40: Andamenti del rumore di pitch e yaw ottenuti dalla scheda IMU montata sulla videocamera nel caso di controllo per la stabilizzazione attivato mantenendo fermo il brandeggio. I valori di deviazioni standard dei segnali sono, rispettivamente, 0.41° per pitch e 0.03° per lo yaw.

Un ulteriore osservazione che è necessario fare, riguarda la presenza di una cross-correlazione tra le misure di orientazione dei vari assi

della IMU. Si può vedere come, in fig. 39 nel caso del controllo non attivo, le oscillazioni di roll vanno ad influenzare pure il segnale di yaw, come pure esiste una correlazione tra i segnali di roll e pitch. Ciò può trovare spiegazione nella non perfetta elaborazione dei dati da parte dell'algoritmo di sensor fusion. In ogni caso, al fine di ottenere prestazioni migliori, è stato necessario ricercare il migliore compromesso per quanto riguarda la banda passante in modo da rendere il sistema più pronto possibile senza però renderlo vulnerabile ad eventuali rumori.
Risultati analoghi (riportati in fig. 41 e fig. 42 e nelle rispettive tab. 3 e tab. 4) si sono ottenuti eseguendo le prove con i riferimenti ondosi realistici precedentemente descritti.

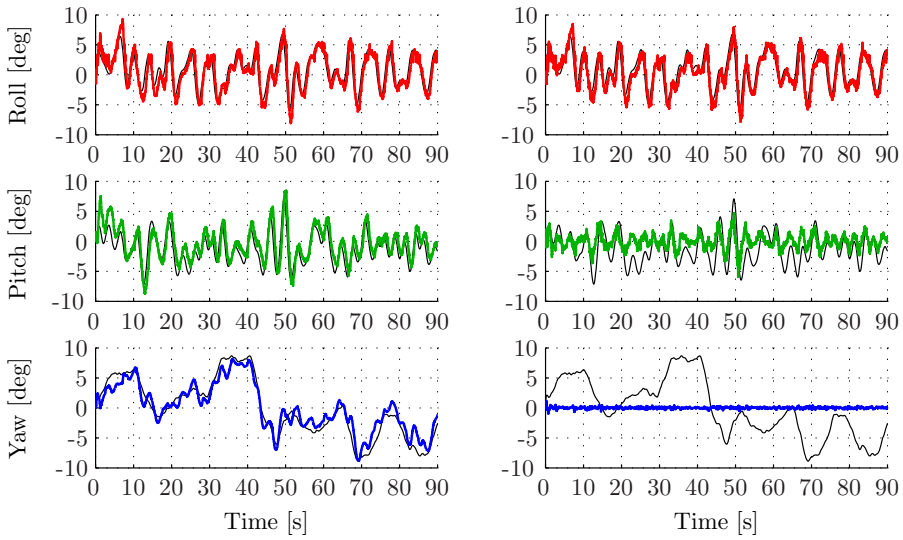


Figura 41: Andamenti di roll, pitch, yaw ottenuti dalla scheda IMU montata sulla videocamera nel caso di controllo per la stabilizzazione disattivato (a sinistra) e attivato (a destra). I riferimenti di orientazione utilizzati dal simulatore per la generazione del moto (linee nere) riguardano la nave di pattugliamento costiera costiera(primo riferimento ondoso).

Tabella 3: Tabella delle deviazioni standard dei segnali di roll, pitch, yaw ottenute con il primo riferimento di moto ondoso.

Deviazione standard

Angolo	Simbolo	No comp. [deg]	Comp. [deg]
Roll	σ_R	3.58	3.40
Pitch	σ_P	2.93	1.13
Yaw	σ_Y	3.77	0.16

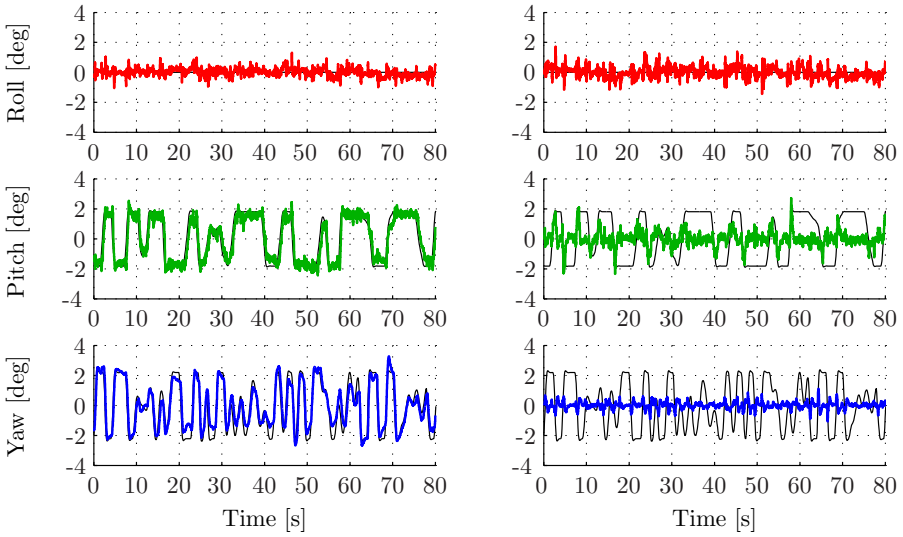


Figura 42: Andamenti di roll, pitch, yaw ottenuti dalla scheda IMU montata sulla videocamera nel caso di controllo per la stabilizzazione disattivato (a sinistra) e attivato (a destra). I riferimenti di orientazione utilizzati dal simulatore per la generazione del moto (linee nere) riguardano la boa costiera(secondo riferimento ondoso).

Tabella 4: Tabella delle deviazioni standard dei segnali di roll, pitch, yaw ottenute con il secondo riferimento di moto ondoso.

		Deviazione standard	
Angolo	Simbolo	No comp. [deg]	Comp. [deg]
Roll	σ_R	0.27	0.34
Pitch	σ_P	1.45	0.51
Yaw	σ_Y	1.48	0.25

5 ELABORAZIONE D'IMMAGINE E VIDEO

5.1 INTRODUZIONE

L'obbiettivo di questo capitolo è di verificare la bontà del controllo sviluppato attraverso l'uso della videocamera incorporata nell'Ulisse Compact. A tale scopo verranno inizialmente descritte, e successivamente applicate, alcune tecniche di elaborazione digitale di immagini e video. In particolare verranno approfondite le metodologie che permettono di riconoscere particolari configurazioni di punti presenti nell'immagine, come segmenti, curve o altre forme prefissate.
Si cercherà poi di sfruttare i risultati ottenuti per fornire un'analisi non solo qualitativa mediante video, ma anche quantitativa, fornendo al lettore indici di prestazione del controllo in termini di varianze e deviazioni standard.
Nell'ultima parte verrà proposta una compensazione off-line del moto di roll, ossia quel moto non compensabile dal brandeggio, utilizzando i dati forniti dal sensore IMU. Anche di tale compensazione verranno presentati risultati in termini di indice di prestazione.
Questo tipo di stabilizzazione del moto di roll assume un significato molto importante se si pensa all'applicazione in oggetto. Infatti utilizzando un controllo anche sul movimento di roll si ottiene un video sempre allineato orizzontalmente qualunque sia l'ondulazione imposta dal moto ondoso sulla boa su cui l'Ulisse Compact è installato.

5.2 L'IMMAGINE DIGITALE

Un'immagine digitale è la rappresentazione numerica di una immagine reale bidimensionale e può essere di tipo vettoriale oppure raster (altrimenti detta bitmap).
Nel primo caso sono descritti degli elementi primitivi, quali linee o poligoni, che vanno a comporre l'immagine mentre nel secondo l'immagine è composta da una matrice di punti, detti pixel, la cui colorazione è definita tramite uno o più valori numerici.
Nel seguito verranno approfonditi alcuni aspetti legati a quest'ultima tipologia di immagini. Con riferimento a fig. 43, in questo tipo di immagini, i valori memorizzati nelle celle indicano le caratteristiche di ogni punto dell'immagine da rappresentare (pixel):

- nelle immagini a colori, viene memorizzato solitamente il livello di intensità dei colori fondamentali (nel modello di colore RGB, uno dei più usati, sono tre: rosso, verde e blu. Un altro

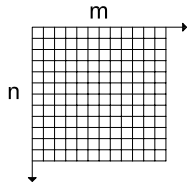


Figura 43: Rappresentazione in $m \times n$ pixel di un immagine digitale raster

esempio è CMYK, usato per la stampa, basato su quattro colori fondamentali: ciano, magenta, giallo e nero);

- nelle immagini monocromatiche in scala di grigio (dette impropriamente in bianco e nero) il valore indica l'intensità del grigio, che varia dal nero al bianco.

Il numero (detto anche profondità) di colori o di livelli di grigio possibili dipende dal massimo numero di combinazioni permesse dalla quantità di bit utilizzata per ognuno di questi dati: un'immagine con 1 bit per pixel avrà al massimo due combinazioni possibili (0 e 1) e quindi potrà rappresentare solo due colori o solo bianco e nero; nelle immagini a 4 bit per pixel, si possono rappresentare al massimo 16 colori o 16 livelli di grigio; un'immagine a 8 bit per pixel, 256 e così via.
Le immagini bitmap possono essere memorizzate in diversi formati, spesso basati su un algoritmo di compressione, che può essere lossy (in cui c'è perdita di informazione), come nelle immagini JPEG, oppure lossless (senza perdita), come nel caso dei file d'immagine GIF o PNG. Questo tipo di immagini può essere generato da una grande varietà di dispositivi d'acquisizione: scanner e fotocamere digitali (contenenti dei sensori CCD o CMOS), ma anche da radar e microscopi elettronici. Inoltre possono venire sintetizzate anche a partire da dati arbitrari, come funzioni matematiche bidimensionali o modelli geometrici tridimensionali. Il campo dell'elaborazione digitale delle immagini studia gli algoritmi per modificare tali immagini a seconda dell'applicazione da sviluppare. Di seguito verranno descritte due principali funzioni che stanno alla base dell'elaborazione video sviluppata.

5.3 LA BINARIZZAZIONE DELL'IMMAGINE

In molti casi, le scene di interesse conducono ad immagini considerate binarie, cioè contenenti nel caso ideale solo due livelli di grigio (bianco, nero). Alcuni esempi possono essere il testo stampato, un manoscritto, parti meccaniche piatte o sagome. In altri casi, le immagini in analisi sono intrinsecamente a più livelli di grigio o a colori, ma l'unico contenuto rilevante è dato dalla forma degli oggetti, delle regioni o delle linee. Ad esempio, dopo l'acquisizione di una scena da parte di un sensore ottico (telecamera o scanner), l'immagine che

si ottiene è formata da numerosi livelli (tipicamente 256 se codificata ad 8 bit).
Sono quindi necessari degli algoritmi per la trasformazione di un'immagine a livelli di grigio o a colori, in immagine binaria, detti algoritmi di binarizzazione, in modo da conservare, quanto più possibile, solo il contenuto rilevante ai fini dell'applicazione che si sta svolgendo. La soluzione più semplice è far uso di una soglia fissa, definita come S, per cui la binarizzazione si realizza tramite la trasformazione:

$$y(x) = \begin{cases} 0 & \text{Se } x < S \\ 255 & \text{Se } x \geqslant S \end{cases} \tag{24}$$

É quindi necessario individuare il valore di tale soglia in modo che renda efficace l'operazione di binarizzazione. Esistono diversi metodi per valutare la soglia a partire dall'istogramma dei livelli dell'immagine originale [18]. In alcuni casi favorevoli, l'istogramma dell'immagine da binarizzare presenta un andamento nettamente bimodale, cioè sono facilmente individuabili due picchi che rappresentano distintamente lo sfondo e gli oggetti presenti nella scena. In questo caso, la soglia viene fissata in corrispondenza del punto di minimo tra i due picchi. Un esempio di istogramma bimodale viene mostrato in fig. 44 in cui si può facilmente individuare i due picchi in questione e il valore della soglia può essere posto a circa 125.

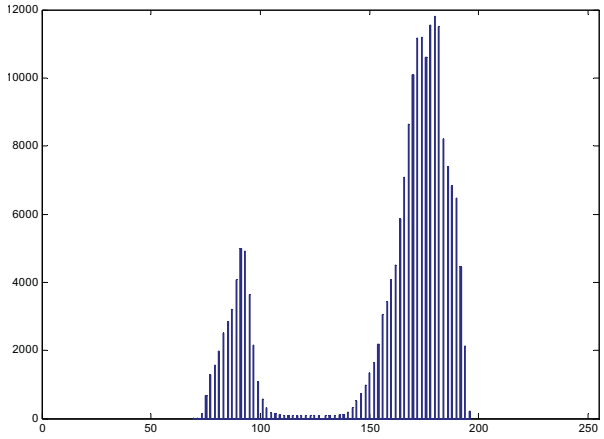


Figura 44: Esempio di istogramma bimodale

Più spesso però l'istogramma è molto più uniforme, anche nel caso di immagini di partenza formate da oggetto e sfondo, dove dovrebbe apparire di forma bimodale. Ciò può essere dovuto alle transizioni sui bordi che generano una gamma di livelli di grigio intermedi tra l'oggetto e lo sfondo e/o da effetti fotometrici che creano delle gradazioni di grigio anche all'interno dell'area dell'oggetto che dovrebbe essere uniforme. Tali problemi possono essere ridotti attraverso un ap-

propriato aggiustamento del contrasto e/o della luminosità dell'immagine prima della binarizzazione in modo da rendere l'istogramma più simile ad uno bimodale, rendendo di conseguenza più facile la stima della giusta soglia.

5.4 LA TRASFORMATA DI HOUGH

Nel campo dell'elaborazione delle immagini, e più specificatamente nel riconoscimento delle forme, è importante considerare il contesto del problema. In un'applicazione OCR[1] i due simboli in fig. 45 devono risultare ``diversi´´(ed essere riconosciuti come le due distinte lettere, ossia *Z* ed *N*) mentre, per un'applicazione industriale che può essere utilizzata per esempio su un robot d'assemblaggio, le due forme devono essere riconosciute come un unico oggetto ma disposto in due modi differenti.

Figura 45: Simboli con molteplici interpretazioni

E' quindi possibile dividere il problema in due parti principali sottoproblemi:

- riconoscimento e l'analisi delle immagini indipendentemente dal contesto di applicazione;
- implementazione di metodi decisionali dedicati per la singola applicazione.

La trasformata di Hough è una tecnica che appartiene al primo dei due sottoproblemi. Permette il riconoscimento di configurazioni globali presenti in una immagine (segmenti, curve, forme prestabilite), e si basa sulla trasformazione di tutti i punti costituenti una immagine, in punti di un nuovo piano detto piano dei parametri, fig. 46. Nel caso in esame tale trasformata è stata utilizzata per identificare le linee presenti sullo scenario riportato in fig. 52. Ciò è essenziale ai fini della verifica del controllo sviluppato e per l'analisi dei dati ottenuti dalle prove sperimentali effettuale.

Nella sua versione tradizionale, la trasformata di Hough si applica ad immagini binarie, ovvero immagini in cui sono presenti due soli livelli, bianco e nero, ed in cui l'informazione associata ad un punto è rappresentata unicamente dalla sua posizione.
Questa tecnica, ideata da Hough nel 1962, è basata sulla ``validazione delle ipotesi´´in cui, definita la curva che si intende cercare nella scena,

1 OCR (*optical character recognition*) sono programmi dedicati alla conversione di un'immagine contenente testo in testo digitale modificabile con un normale editor

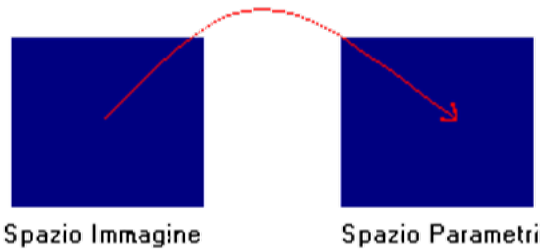


Figura 46: Trasformazione dei punti dal piano immagine al piano dei parametri

per ogni punto dell'immagine si calcolano i parametri di tutte le curve che potrebbero passare per quel punto e si incrementano le celle di un piano n-dimensionale[2]. Si ottiene così una funzione di accumulazione definita nel piano dei parametri. Alla fine saranno i massimi di questa funzione, ovvero i punti nel piano dei parametri che hanno il valore dell'accumulatore più alto, a rappresentare le curve che hanno maggior probabilità di essere presenti nell'immagine, come se si trattasse di una ipotesi avvalorata dal maggior numero di conferme sperimentali [3].
Una delle caratteristiche più interessanti della Trasformata di Hough è quella di non risentire del rumore presente in una immagine. Infatti, l'altezza dei picchi nella matrice di accumulazione, dipende in maniera del tutto trascurabile da eventuali lacune presenti nella retta di partenza o dall'esistenza di punti spuri presenti nel piano immagine.

5.4.1 *Riconoscimento di segmenti*

Assumendo come rappresentazione della retta la forma cartesiana

$$y = mx + q \tag{25}$$

qualunque retta è completamente specificata dal valore dei parametri (m,q). Se si assume un tipo di rappresentazione diversa, quale la forma polare,

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{26}$$

l'insieme di parametri varia di conseguenza. In questo caso la retta è completamente specificata dalla coppia (ρ,θ) dove ρ è la distanza tra la retta e il punto (0,0) e θ è l'angolo che forma tale segmento distanza (ortogonale alla retta) con l'asse delle ascisse, fig. 47.

Sebbene la due rappresentazioni possono sembrare equivalenti per identificare una retta nel piano immagine, la (25) presenta un proble-

2 piano dei parametri

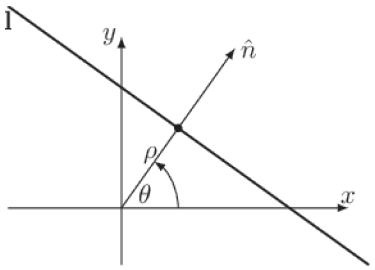


Figura 47: Rappresentazione di una retta in forma polare

ma legato alla trasformata di Hough perché al variare di m e q si possono avere infinite rette. Si pone dunque il problema della discretizzazione del piano (m,q) in un numero finito di celle. Mentre per q si potrebbe fissare un valore massimo e uno minimo, il parametro m, invece, cresce indefinitamente quando la retta diventa verticale, per cui limitare tale parametro significherebbe limitare le rette riconoscibili. A tal fine, è utile utilizzare la formula alternativa in (26) poiché, quest'ultima, permette una discretizzazione finita di tutti i parametri presenti anche in caso di rette verticali. Per questo motivo nel seguito della trattazione si farà riferimento all'equazione (26) e non alla (25) in modo da aver sempre un piano dei parametri di dimensioni finite. Per quanto scritto, una particolare istanza della forma cercata nel piano immagine è completamente precisata dal valore assunto dai parametri nel piano dei parametri. Una retta, per esempio, viene

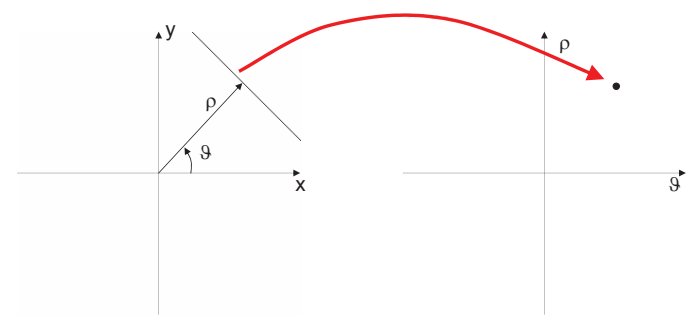


Figura 48: Trasformazione nel piano dei parametri di una retta

rappresentata da un punto nel piano dei parametri come mostrato in fig. 48.
Resta dunque da chiarire in che modo è possibile sfruttare questa trasformata ai fini dell'individuazione di segmenti in un immagine. Nell'immagine, l'unica informazione disponibile è costituita dall'insieme di punti che la costituiscono, e pertanto è utile chiedersi quale sia la trasformata di un punto sul piano dei parametri. Nel piano dell'immagine, con riferimento a fig. 49, un punto è identificato dall'intersezione di più rette e quindi, ad ogni punto P corrisponde, nel piano dei parametri, la curva formata dai parametri delle rette passanti per P. Se ora nell'immagine compaiono due o più punti allineati sulla

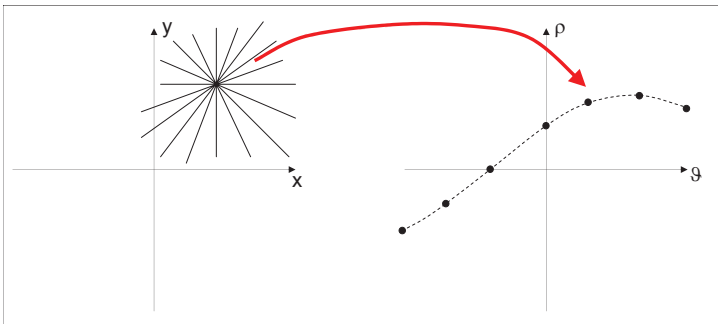


Figura 49: Trasformazione nel piano dei parametri di un punto

stessa retta, sul piano dei parametri vengono a formarsi più curve che corrispondono alle trasformazioni dei vari punti. Con riferimento a fig. 50, si osserva che queste curve si intersecano in un punto del piano trasformato che corrisponde ai parametri identificatori della retta su cui giacciono tutti i punti del piano immagine.

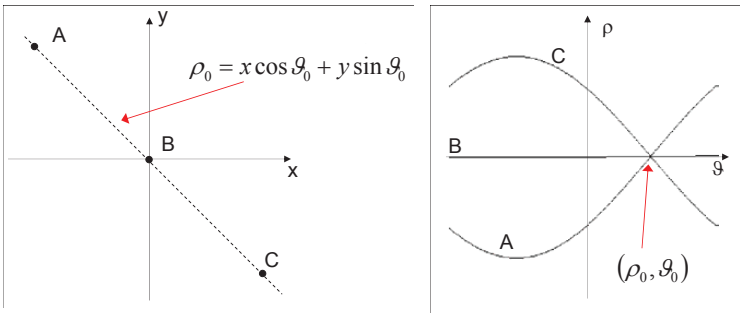


Figura 50: Trasformazione nel piano dei parametri di due o più punti allineati su una stessa retta

In questo modo, è possibile individuare i segmenti di retta presenti sull'immagine originale [17].

5.4.2 *Implementazione della trasformata di Hough*

Se si considera una discretizzazione del piano dei parametri (ρ,θ) è possibile rappresentarlo su una matrice $H(m, n)$ i cui indici di riga m, e di colonna n, corrispondono ai valori quantizzati di ρ e θ. Gli intervalli di variazione di ρ e θ sono fissati sulla base delle caratteristiche dell'immagine originale. Tipicamente

$$-\rho_{max} \leqslant \rho \leqslant \rho_{max} \quad \text{e} \quad -\pi/2 \leqslant \theta \leqslant \pi/2 \tag{27}$$

Il numero dei livelli di quantizzazione va poi scelto in base all'accuratezza desiderata sia per ρ che per θ. In verità esistono dei vincoli dati dalla velocità computazionale del PC utilizzato. Infatti più la risoluzione aumenta più il processo di identificazione delle linee diventa lungo e oneroso. Operativamente i passi da seguire sono i seguenti:

1. si azzera la matrice $H(m, n)$
2. per ogni punto del piano immagine si definisce $P = (x, y)$
   a) per ogni θ_n del piano trasformato che varia tra $-\pi/2$ e $\pi/2$
      i. si valuta $\rho(n) = x \cdot \cos(\theta_n) + y \cdot \sin(\theta_n)$
      ii. si ricava l'indice m corrispondente a $\rho(n)$
      iii. si incrementa la cella $H(m, n)$
3. Si individuino i massimi locali su H corrispondenti ai parametri dei segmenti individuati

5.5 IL FILMATO DIGITALE

Un filmato, o un video, è una sequenza di immagini digitali riprodotte ad una velocità sufficientemente alta da fornire, all'occhio umano, l'illusione del movimento. La frequenza di riproduzione dei fotogrammi (in inglese frame rate) viene misurata in hertz (Hz), nei monitor a scansione progressiva, oppure viene semplicemente espressa in termini di fotogrammi per secondo (*fps*).
Il frame rate minimo di un filmato, affinché l'apparato visivo umano non percepisca sfarfallii o alterazioni del movimento, si attesta sui 30 fps (o 30 Hz), ma può anche essere superiore per gli oggetti che si muovono molto veloci sullo schermo.
Detto ciò, è facile intuire, che l'operazione di video-processing equivale di fatto ad elaborazioni su immagini digitali (o di image-processing) dove ogni immagine è costituita da un preciso frame del video in esame. Pertanto quando si acquisisce un video con lo scopo di rielaborarlo è importante evitare qualunque algoritmo di compressione dell'immagine che comporterebbe una perdita di informazioni video.

5.6 L'ACQUISIZIONE VIDEO

Le registrazioni video sono state effettuate utilizzando la videocamera di serie interna al brandeggio mostrata in fig. 51.

Figura 51: Videocamera dell'Ulisse Compact: Sony FCB-EX480CP

Questa videocamera a colori ha uno zoom ottico fino a 18× e un sensore di visione CCD a 380.000 pixel in formato PAL. Il controllo dello zoom avviene manualmente tramite la tastiera associata al brandeggio, mentre per quanto riguarda l'iris e il focus esiste la possibilità di una gestione automatica svolta dalla scheda body dell'Ulisse Compact. L'alimentazione alla videocamera viene fornita attraverso le schede elettroniche interne al brandeggio, le quali gestiscono le immagini acquisite e le rendono disponibili in uscita. Il segnale video ottenuto è di tipo composito in formato PAL con frequenza video di 25 frame al secondo.
Per acquisire i video è stato necessario utilizzare un PC aggiuntivo con installata la scheda PCI/TV della TerraTec (modello TerraTValue PCI-TV Card). Tale scheda presenta, tra gli altri, un ingresso video-composito a cui è possibile collegare dispositivi di acquisizione video. Il software utilizzato per la registrazione delle tracce video è VLC prodotto dalla VIDEOLAN organization. Quest'ultimo permette, grazie alla funzionalità *converti/salva* presente nel menù *Media* sulla barra delle applicazioni e settando opportunamente alcuni parametri, l'acquisizione di immagini o video da un qualsiasi device precedentemente installato nel PC. I parametri da settare per una corretta acquisizione variano in base al tipo di telecamera utilizzata e all'applicazione da sviluppare. Nel caso in esame si è impostato:

- Modalità di acquisizione: *DirectShow*;
- Formato del file in uscita: *AVI non compresso*[3];
- Frequenza fotogrammi: 25 fps;
- Spazio colore: *YUY2*[4];
- Formato video: *PAL-B*;
- Dimensione output: 320 × 240.

Con questo set di parametri si riesce quindi ad acquisire un video in formato AVI con codifica PAL-YUV, a 25 fps e risoluzione 320 × 240 pixel. Quest'ultima potrebbe essere anche aumentata ma ciò comporterebbe un maggior ingombro del file ottenuto. Ciò, oltre ad essere inutile per gli obbiettivi proposti, rende più onerosa, dal punto di vista computazionale, l'operazione di video-elaborazione.

3 è necessario acquisire i video senza compressione per i motivi specificati in 5.5

4 Il formato YUY2 rispetta lo standard YUV eseguendo un sub-campionamento orizzontale con fattore 2. Lo standard YUV, prima battezzato YCrCb (Y Cr Cb), è un modello di rappresentazione del colore dedicato al video analogico. Si basa su una modalità di trasmissione video a componenti separate che usa tre cavi diversi per far transitare le informazioni di luminosità (luminance) e le due componenti di colore (crominance) chiamante U (blue projection) e V (red projection). Si tratta del formato usato negli standard PAL (Phase Alternation Line) e SECAM (Séquentiel Couleur avec Mémoire).

5.6.1 *Lo scenario di ripresa*

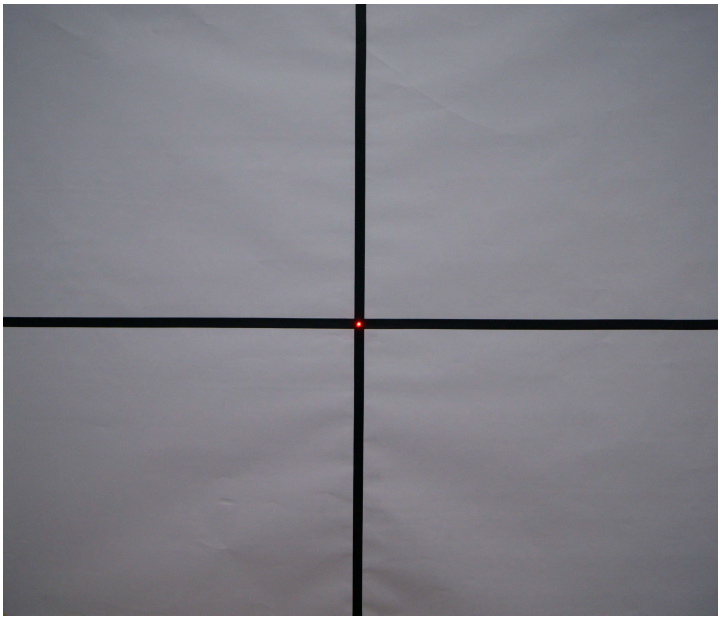


Figura 52: Scenario di ripresa per l'acquisizione video

Al fine di acquisire dei video confrontabili tra loro è stato creato uno scenario di ripresa comune visibile in fig. 52. Questo scenario consiste in una croce formata da due linee nere su sfondo bianco. Tale scelta è motivata dal fatto che tutte le informazioni rilevanti riguardo l'orientazione della videocamera e la posizione del suo centro possono essere determinate dalla misura della posizione e orientazione della croce nel piano immagine.
Lo scenario è stato posizionato ad un metro della lente della videocamera ed in modo tale che il suo centro sia coincidente con il centro del campo visivo della videocamera a meno di piccole movimentazioni della base causate dalle perdite nel circuito pneumatico. Così facendo è possibile valutare l'efficacia della stabilizzazione rilevando in ogni frame video la differenza tra la posizione del centro della croce e quella del centro dell'immagine. Inoltre osservando l'inclinazione della linea orizzontale è possibile eseguire una compensazione del movimento di roll.
In aggiunta alle linee sopra descritte, per motivi che verranno approfonditi nel paragrafo successivo, è stato posizionato un led al centro dello scenario. La trasformata di Hough di tale scenario viene riportata in figura 53. Le due frecce nel grafico indicano i punti di accumulazione che identificano i parametri delle due rette. Si osserva che il picco centrale ha come una angolo θ pari a $0°$ e quindi identifica i parametri della retta verticale mentre quello a sinistra ha θ pari a $90°$ e identifica quindi la retta orizzontale.

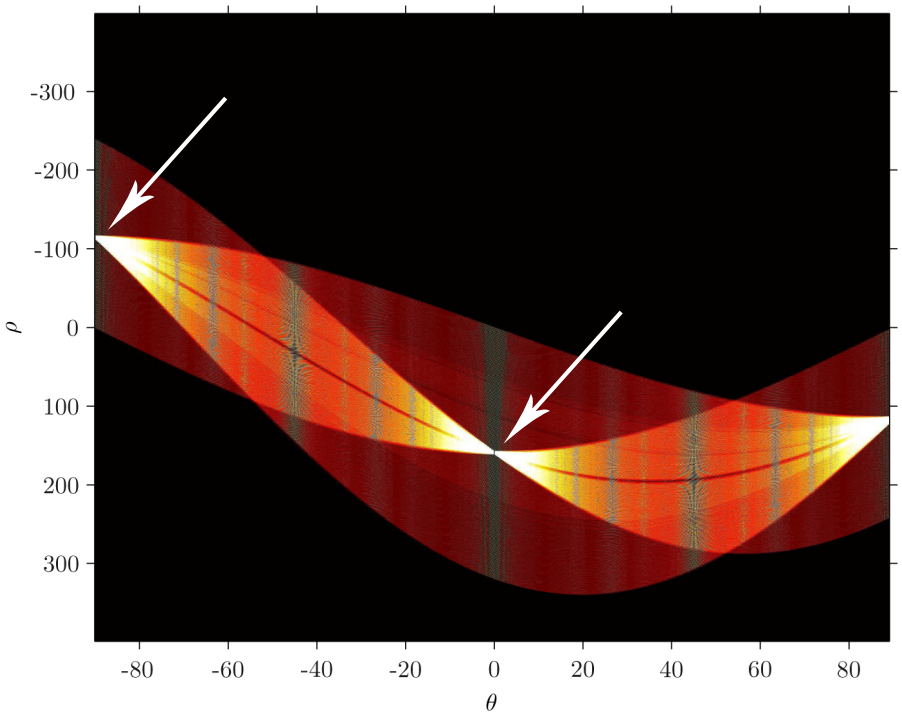


Figura 53: Trasformata di Hough dello scenario

5.6.2 *La sincronizzazione dei PC*

L'esperimento consiste nell'acquisizione dei segnali della IMU e dei video della videocamera al fine di ottenre una valutazione qualitativa e quantitativa del controllo svolto. Per far ciò si sono effettuate, per ogni tipo di movimentazione, due registrazioni; una con il controllo attivo, e una con il controllo disattivo, ossia con i motori di pan e tilt bloccati nella posizione di Home. L'obbiettivo è quello di eseguire un confronto tra i video e i dati acquisiti. A tal fine è essenziale riuscire a definire precisamente l'istante di inizio del movimento[5], ovvero bisogna creare un segnale di sincronizzazione tra il PC che comanda la piattaforma mobile, il PC che controlla il brandeggio e acquisisce i dati dalla IMU e il PC esegue la registrazione video. All'istante di inizio del movimento che si vuole effettuare viene mandato, per mezzo delle schede di acquisizione dati, un segnale al PC che controlla il brandeggio. Questo, attraverso i blocchi visibili in alto a destra in fig. 38 genera un enable per l'inizio dell'acquisizione e salvataggio dei dati provenienti dalla IMU. Per quanto riguarda il PC che esegue la registrazione video, essendo sprovvisto di scheda di acquisizione dati, si è pensato di introdurre un led sullo scenario che, spegnendosi, fornisce un segnale visivo dell'inizio della prova. Il led quindi è stato collegato al PC della piattaforma che lo spegne precisamente all'inizio della movimentazione selezionata. Successivamente attraverso un

5 Il frame rate è 25 fps e quindi è possibile identificare una variazione nel video minimo ogni 0.04 secondi

programma di visualizzazione frame by frame è stato possibile selezionare il preciso frame di spegnimento del led e da questo iniziare ad eseguire l'operazione di video-elaboration.
Riassumendo qunidi i passi da seguire per l'acquisizione dei video sono:

1. preparare la piattaforma ed il brandeggio in posizione di Home (il led sullo scenario si accederà automaticamente);
2. attivare o disattivare il controllo a seconda di che video e dati si vogliono acquisire;
3. iniziare l'acquisizione video;
4. iniziare la movimentazione voluta, e contemporaneamente invio del segnale di inizio acquisizione al PC del brandeggio e spegnimento del led sullo scenario;
5. al termine della movimentazione chiudere l'acquisizione video;
6. trovare il frame di inizio movimentazione con un programma di visualizzazione video frame by frame;
7. iniziare la video-elaboration con Matlab a partire dal frame trovato.

5.7 MATLAB E I PROGRAMMI DI ACQUISIZIONE VIDEO

Il programma utilizzato per l'elaborazione video è Matlab. Tale software permette, tra le tante funzioni, di acquisire video in vari formati ed elaborarli con diverse funzioni preimpostate. Si sono utilizzate:

- *VideoReader(filename)*: questa funzione crea un oggetto che permette di leggere un file video AVI (*.avi) di nome *filename*;
- *VideoWriter(filename)*: costruisce un oggetto per scrivere un video in un file AVI chiamato *filename*;
- *rgb2gray(I)*: converte un'immagine *I* con codifica RGB in un immagine in scala di grigi eliminando i colori. In altre parole esegue una binarizzazione dell'immagine I con sogliatura automatica;
- *imadjust(I)*: incrementa il contrasto dell'immagine *I*;
- *getframe(I)*: restituisce in formato di frame la figura aperta in quel preciso istante;
- *frame2im(fr)*: converte un frame *fr* in un immagine;
- *imagerotate(I,angle)*: ruota un immagine rispetto il suo centro di misura specificata in *angle*;

- *writeVideo(writerObj,fr)*: scrive un frame *fr* nel file video associato all'oggetto *writerObj*;
- *hough(I_BW)*: calcola la Standard Hough Trasform (SHT) di un immagine binaria *I_BW*;
- *houghpeaks(H,numpeacks)*: individua i picchi nella matrice di accumulazione della trasformata di hough *H*.

I listati dei programmi sviluppati sono riportati in appendice A.1.4, A.1.5, A.1.6 e A.1.7. Tali programmi consistono in un codice principale di *main* che parametrizza e invoca la funzione di *video elaboration* all'interno della quale vengono richiamate le funzioni di *line_detection_Hough* e *find_center*. Ciò permette, mantenendo uguali tutte le funzioni sviluppate, una facile elaborazione di tutti i movimenti parametrizzando opportunamente il main come descritto nei commenti di riga ad inizio file.
Come risultato dell'elaborazione si ottiene, oltre a una serie di strutture di dati utili per un analisi quantitativa di cui si discuterà in seguito, anche un video di output in formato AVI in cui è possibile osservare la bontà dell'operazione svolta. Si nota infatti che le linee identificate mediante la trasformata di Hough sono perfettamente sovrapposte a quelle reali dello scenario durante tutta la durata del video.

5.7.1 *L'identificazione del centro dello scenario*

Una volta acquisiti i video con la procedura in 5.6.2, è essenziale, al fine di trovare la posizione del centro della croce, riuscir ad identificare le linee sullo scenario di ripresa. Per far ciò è stata utilizzata la trasformata di Hough ampiamente descritta nei paragrafi precedenti. Tale funzione, nella versione implementata in Matlab, ritorna tre matrici, dette di accumulazione (H), di parametro theta (T) e di parametro rho (R). La prima di queste ha per elementi dei contatori che vengono incrementati secondo l'algoritmo espresso in pseudocodice in 5.4.2. Quelli che raggiungono il valore massimo identificheranno gli indici utili per trovare nelle matrici di R e T il valore dei parametri della retta individuata.
Identificate per ogni frame del video le rette nello spazio dei parametri è quindi sufficiente, al fine dell'individuazione del centro nel piano immagine, calcolarne i coefficienti m e q in forma cartesiana e metterle a sistema come segue:

$$\begin{cases} y = m_1 \cdot x + q_1 \\ y = m_2 \cdot x + q_2 \end{cases} \tag{28}$$

Risolvendo tale sistema si trovano le coordinate x e y del centro della croce espresse in pixel per ogni frame del video in esame. Tale operazione viene eseguita nella funzione chiamata *find_center*. Le due

coordinate x e y vengono poi salvate in un vettore bidimensionale chiamato xy per le successive analisi.

5.8 ANALISI DEI DATI OTTENUTI DALLA POST-ELABORAZIONE VIDEO

Verranno di seguito analizzate le strutture dati salvate durante l'elaborazione video effettuata sulle medesime prove proposte precedentemente nel paragrafo 4.6. Si vuole quindi effettuare una valutazione della bontà del controllo in termini di deviazione standard sul movimento del centro della croce nel piano immagine.

5.8.1 *Prove con riferimento sinusoidale*

In fig. 54 è possibile osservare, in forma qualitativa, il movimento del centro della croce sul piano immagine acquisito frame dopo frame durante la movimentazione della piattaforma con riferimenti sinusoidali. Tali riferimenti sono i medesimi che si sono utilizzati nel paragrafo 4.6 e hanno un andamento nel tempo visibile in nero in figura 39. Si

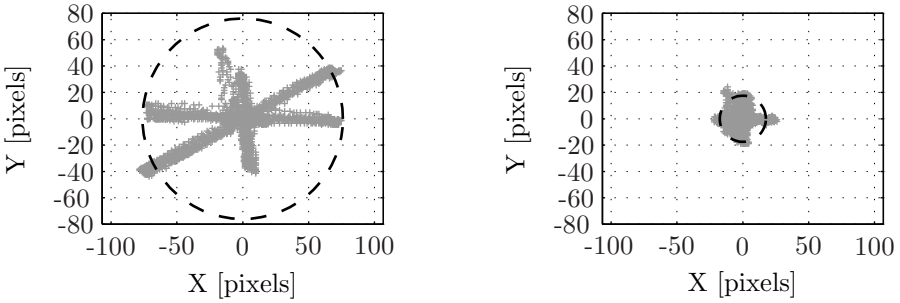


Figura 54: Movimento del centro della croce sul piano immagine con riferimento sinusoidale.

può notare come nel grafico di sinistra, corrispondente alla prova con controllo disattivato, il centro della croce sullo scenario si muova, come è giusto che sia, a destrasinistra e in sugiu. Si osserva anche una traccia obliqua data dalla combinazione di tali movimenti. Nel grafico di destra, che corrisponde invece alla medesima prova con controllo attivo, tali movimenti non sono più distinguibili proprio per l'effetto stabilizzante del controllo sviluppato. Il cerchio in entrambe i grafici rappresenta il limite $3\sigma_r$ della distribuzione del centro della croce nel piano immagine. Un analisi quantitativa può essere sviluppata osservando tab. 5, dove sono riportate le deviazioni standard espresse in pixel del movimento del centro della croce sullo scenario. Da tali dati si può facilmente notare come le deviazioni standard ottenute con controllo attivo risultino più che dimezzate rispetto a quelle ottenute con controllo disattivo. Si osserva inoltre che la compensazione del movimento di yaw è molto migliore di quella sul movimento di pitch.

Ciò risulta coerente con i risultati ottenuti attraverso le misure inerziali nel capitolo precedente.

Tabella 5: Tabella delle deviazioni standard ottenute con riferimento sinusoidale.

Deviazione standard			
Direzione	Simbolo	No comp. [pixels]	Comp. [pixels]
X–axis	σ_x	33.64	7.93
Y–axis	σ_y	17.60	6.91
radial	σ_r	25.37	5.84

5.8.2 *Prove con riferimento di moto ondoso acquisito su una nave per il pattugliamento costiero*

In fig. 55 è possibile osservare il movimento del centro della croce sul piano immagine acquisito frame dopo frame durante la movimentazione che simula l'oscillazione, causata dal moto ondoso, di una nave da pattugliamento ancorata. Il grafico di sinistra illustra tale movimento nel caso di controllo della videocamera PTZ disattivato mentre quello di destra lo mostra con il controllo attivato. Il cerchio rappresenta, come nella prova precedente, il limite $3\sigma_r$ della distribuzione del centro della croce nel piano immagine.

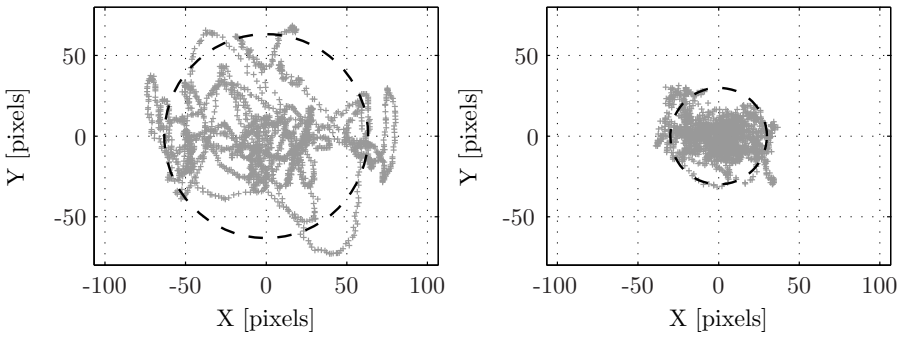


Figura 55: Movimento del centro della croce sul piano immagine con il primo riferimento di moto ondoso.

Per valutazione quantitativa vengono proposte in tab. 6 le deviazioni standard di tale movimento espresse in pixel. Dalla tabella si può notare come le deviazione standard calcolate nel caso di controllo attivo risultino ridotte di un fattore circa pari a 2.2 rispetto a quelle calcolate nel caso di controllo non attivo.

Tabella 6: Tabella delle deviazioni standard ottenute con il primo riferimento di moto ondoso.

Deviazione standard			
Direzione	Simbolo	No comp. [pixels]	Comp. [pixels]
X–axis	σ_x	38.79	16.94
Y–axis	σ_y	23.65	10.45
radial	σ_r	21.09	9.98

5.8.3 *Prove con riferimento di moto ondoso acquisito su una boa da sorveglianza costiera*

Una valutazione del tutto simile alle precedenti può essere effettuata osservando fig. 56. Tale figura mostra lo stesso movimento del centro della croce sul piano immagine generato durante la movimentazione che simula l'oscillazione di una boa da sorveglianza costiera.

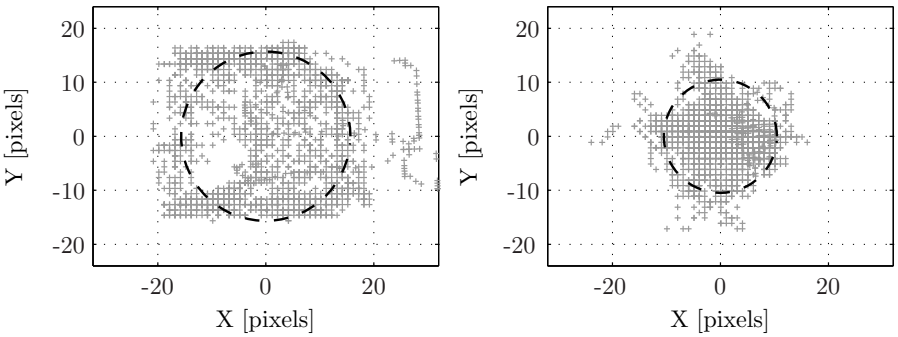


Figura 56: Movimento del centro della croce sul piano immagine con il secondo riferimento di moto ondoso.

Sempre con lo scopo di fornire un analisi dei risultati ottenuti anche in forma quantitativa vengono proposte in tabella 7 le deviazioni standard calcolate come nel caso precedente in pixel. Anche in questo caso le deviazioni standard calcolate con controllo attivo risultano ridotte di circa un fattore 2.2 rispetto alla prova con controllo disattivo.

Confrontando ora i risultati ottenuti con le due prove relative al moto ondoso, si riesce ad intuire come il movimento di roll della base influenzi in modo negativo la compensazione del moto ondoso. Tale movimento infatti, presente solo nel primo riferimento di moto, non è compensabile dal brandeggio. Questo fatto non sarebbe minimamente apprezzabile con l'analisi svolta se il centro di rotazione di roll fosse il centro del piano immagine in quanto tale movimento non altererebbe la posizione in X e in Y del centro della croce sullo scenario. Tuttavia, in questo caso, come anche nei casi reali di videosorveglian-

Tabella 7: Tabella delle deviazioni standard ottenute con il secondo riferimento di moto ondoso.

Deviazione standard			
Direzione	Simbolo	No comp. [pixels]	Comp. [pixels]
X–axis	σ_x	12.06	5.31
Y–axis	σ_y	11.28	4.21
radial	σ_r	5.22	3.50

za in mare, il centro di rotazione non è coincidente con il centro del piano immagine ma è identificato dal centro del sistema di riferimento dell'end-effector della piattaforma che ospita la videocamera PTZ. Tale diversità porta ad ottenere una peggior compensazione del movimento di pan della videocamera in quanto le rotazioni di roll tendono a spostare il centro del piano immagine a destra e sinistra (vedi fig. 57) . Si ottiene così una differenza maggiore in termini di deviazione standard tra asse X e asse Y, osservabile in tabella 6, e una distribuzione dei centri di forma ellittica con asse maggiore coincidente con l'orizzontale, osservabile in fig. 55. Ciò non accade per il secondo riferimento di moto ondoso in quanto non contempla nessun movimento di roll. In questo caso quindi, le deviazioni standard dell'asse di X e di Y sono molto simili e i centri della croce sono distribuiti nel piano immagine con forma all'incirca circolare.

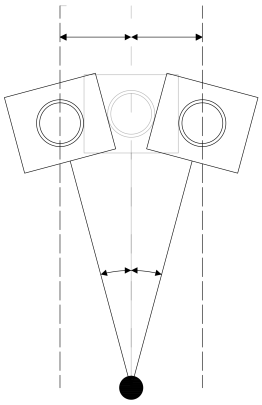


Figura 57: Schema qualitativo di una videocamera ruotata rispetto un centro di rotazione differente dal centro del piano immagine.

Questa problematica non è così grave in quanto la rotazione di roll induce solo una traslazione laterale della videocamera. Tale traslazione, come scritto in 1.5, nel caso più comune di puntamento di un target sufficientemente lontano, può essere considerata trascurabile perché non rischia di farlo uscire dall'inquadratura della videocamera.

5.9 COMPENSAZIONE VIDEO DEL MOVIMENTO DI ROLL OFF-LINE

Di seguito si proporrà un metodo per la compensazione del movimento di roll attraverso l'elaborazione del video acquisito durante la prova precedente con riferimento sinusoidale. Tale metodo si basa sulla rotazione del singolo frame video rispetto al centro del piano immagine. Per far ciò si è quindi ipotizzato che il centro di rotazione del movimento di roll sia proprio il centro del piano immagine. Nel caso in esame, tale ipotesi è veritiera solo nel caso di osservazione di target sufficientemente lontani per cui tale centro del piano immagine può essere approssimato con il vero centro di rotazione di roll.
Un primo metodo che si vuole descrivere riguarda la compensazione della rotazione di roll attraverso la sola elaborazione video. Tale prova consiste nel ruotare il video sulla base delle informazioni estrapolate dal singolo frame in fase di elaborazione video. Nello specifico, per ogni frame, identificate le rette sullo scenario con la trasformata di Hough, si calcola il coefficiente angolare di quella orizzontale ed in base a tale valore si ruota il frame corrispondente. Uno schema a blocchi di tale procedura è riportato in fig. 58.

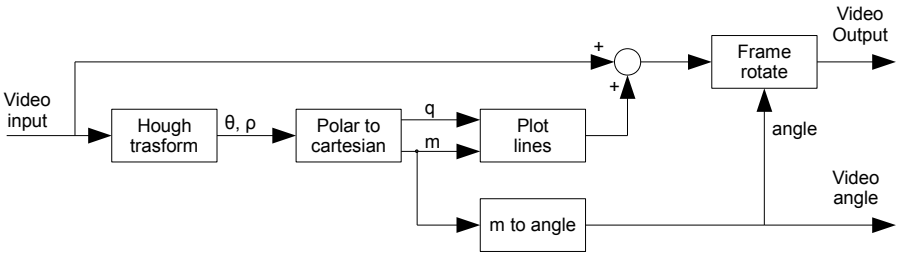


Figura 58: Schema a blocchi della video elaborazione per la valutazione della compensazione delle oscillazioni di roll

E facilmente intuibile come tale compensazione risulti idealmente perfetta, a meno di errori nell'identificazione delle rette. Tali errori possono derivare dalla discretizzazione dei parametri attuata nella trasformata di Hough o dalla mancanza di una linea orizzontale ben distinguibile come nel caso di sorveglianza costiera in cui l'orizzonte è spesso rappresentato da un profilo montuoso e quindi non lineare. Proprio in quest'ultimo caso infatti l'elaborazione video non riuscirebbe ad identificare la linea orizzontale e quindi a calcolarne il coefficiente angolare.

Per questi motivi, si è pensato ad un secondo metodo che sfrutta, come riferimento per la rotazione del frame video, le misure di roll acquisite dal sensore inerziale IMU posizionato sulla videocamera. Ciò necessita ovviamente di una precisa sincronizzazione tra i tre

PC che si occupano del controllo e delle registrazione dei dati. La sincronizzazione tra PC sviluppata è stata ampiamente descritta nel paragrafo 5.6.2. I dati acquisiti dal sensore IMU vengono salvati in una struttura e successivamente utilizzati per la compensazione video off-line della rotazione di roll.
Uno schema a blocchi di tale operazione è visibile in fig. 59.

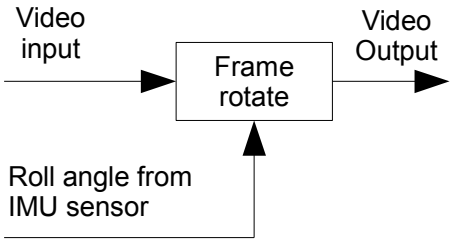


Figura 59: Schema a blocchi della video elaborazione per la compensazione delle oscillazioni di roll off-line o real-time.

Per verificare la fattibilità di questa compensazione si è acquisita la rotazione del video, sfruttando la retta orizzontale identificata dalla trasformata di Hough, e la si è sottratta alla rotazione acquisita dalla IMU durante la medesima prova. In questo modo è possibile ottenere un'indicazione sull'efficacia dell'eventuale compensazione di roll a meno dell'errore di quantizzazione introdotto dalla trasformata di Hough che nel caso in esame è pari a $0.5°$. I risultati ottenuti sono mostrati in figura 60.

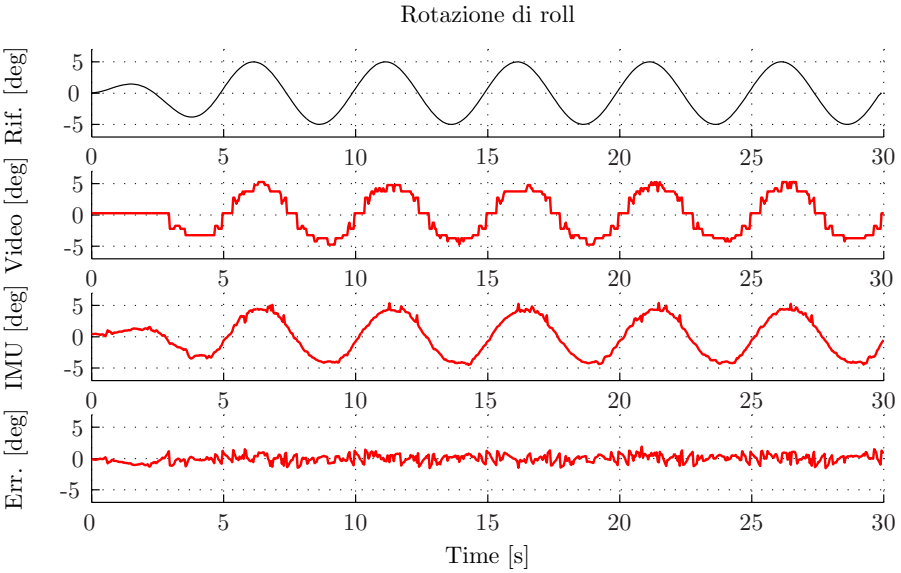


Figura 60: Nel primo grafico in nero viene riportato il riferimento di roll imposto alla base mobile. La traccia rossa nel secondo grafico mostra la rotazione di roll estrapolata dal video. Nel terzo grafico la traccia rossa mostra la rotazione di roll misurata dal sensore IMU e nel quarto mostra la differenza tra la rotazione di roll estrapolata dal video e quella misurata dal sensore IMU al fine di ottenere una stima dell'efficacia di un eventuale compensazione di roll off-line o real-time.

Dalla figura si può osservare come la misura del sensore IMU, anche se rumorosa ha andamento simile al riferimento dato alla piattaforma mobile. In particolare risulta essere leggermente attenuata e in ritardo, il che può essere dovuto ad un non preciso tracking del riferimento da parte della piattaforma stessa. La conferma di tale affermazione può essere trovata osservando i dati sulla rotazione estrapolati dal video, visibili nel secondo grafico della stessa figura. Anche in questo grafico infatti la traccia risulta essere leggermente attenuata e in ritardo rispetto al riferimento imposto. Si osserva inoltre che il segnale è maggiormente segmentato per via della minor frequenza di campionamento del video stesso e per l'errore di quantizzazione introdotto dalla trasformata di Hough. Infatti, mentre i dati dal sensore IMU sono stati salvati con un tempo di campionamento pari a 10 ms, per quanto riguarda il video, il massimo tempo di campionamento possibile è di 40 ms essendo il frame rate pari a 25 fps. Il terzo grafico invece rappresenta la differenza tra la rotazione del roll estrapolata dal video e la stessa rotazione misurata del sensore IMU. Si nota che il segnale derivante è notevolmente ridotto rispetto quello estrapolato dal video. Ciò significa che ruotando il video frame-by-frame con le misure provenienti dalla IMU è possibile compensare off-line il movimento di roll. Ciò sarebbe possibile anche in real-time avendo a disposizione potenze di calcolo maggiori. La differenza tra i due segnali non è perfettamente nulla per via di errori come la quantizza-

zione della trasformata di Hough, tempo di campionamento diverso e rumore sovrapposto.

Tabella 8: Tabella delle ampiezze di movimento di roll. Tali valori sono stati ricavati a partire dai segnali riportati nel precedente grafico, fittati attraverso un algoritmo ai minimi quadrati per determinare la sinusoide che meglio li aprrossima.

Ampiezze picco-picco

Angolo	Simbolo	Riferimento [deg]	video [deg]	IMU [deg]	Errore residuo [deg]
Roll	V_{pp}	9.99	8.21	8.20	0.47

Per una valutazione quantitativa dell'operazione svolta si riportano in tabella 8 le ampiezze picco-picco dei segnali di roll espresse in gradi. Tali valori sono stati calcolati sui segnali ottenuti dal fitting ai minimi quadrati dei segnali reali in modo da evitare errori dovuti alla presenza di rumore o discretizzazioni. Si nota, come mostrava il grafico precedente, che l'ampiezza dei due segnali acquisiti dal video e dalla IMU è praticamente uguale e inferiore a quella del segnale di riferimento alla piattaforma. Inoltre si osserva come l'ampiezza del movimento di roll risultante dalla sottrazione dei due segnali precedenti è pari a 0.47 $gradi$ il che testimonia l'efficacia di un'eventuale compensazione di roll mediante le sole misure inerziali. Tale compensazione è stata successivamente svolta off-line e il video ruotato risultante viene allegato alla tesi.

CONCLUSIONI

La stabilizzazione dell'immagine per un brandeggio per videosorveglianza può essere realizzata in differenti modi, ad esempio, agendo sugli equipaggiamenti ottici a bordo (lenti o sensori) o attraverso un'elaborazione software delle immagini acquisite. In questo elaborato è stata proposta una soluzione alternativa che può essere applicata come adattamento o aggiornamento di sistemi già esistenti sul mercato, sfruttando esclusivamente le misure inerziali ottenute da una scheda IMU aggiuntiva. Vale la pena notare che tale tipo di sensore può essere già trovato all'interno di sistemi di visione avanzati dove viene spesso utilizzato per la soppressione delle vibrazioni e ripple di coppia.

Il lavoro svolto ha permesso dimostrare sperimentalmente l'elevato miglioramento nel mantenimento del target all'interno del campo visivo della videocamera, sotto realistiche condizioni ottenute attraverso opportuni riferimenti al simulatore. In particolare, l'effettivo utilizzo di questo sistema in applicazioni off-shore, ad esempio per brandeggio montati su navi o boe in mare.

L'elaborazione video sviluppata ha permesso di riconfermare i buoni risultati ottenuti dalla stabilizzazione elettromeccanica eseguita evidenziando in modo qualitativo la bontà del controllo sviluppato. Buoni risultati sono stati ottenuti utilizzando le misure inerziali della IMU per compensare off-line i movimenti di roll impossibili da controllare elettro-meccanicamente.

I risultati ottenuti hanno fatto emergere possibili altri sviluppi e miglioramenti che si potrebbero apportare a tale sistema. Infatti il controllo di pitch risulta essere meno performante rispetto a quello di yaw. Risultati migliori potranno essere ottenuti ottimizzando maggiormente l'algoritmo di sensor fusion implementato nella scheda IMU e utilizzando tecniche di filtraggio più avanzate per la riduzione del rumore misurato e delle vibrazioni.

Ulteriori aspetti che potranno essere sviluppati in futuro riguardano la realizzazione di algoritmi per la stabilizzazione a video, algoritmi di riconoscimento ed inseguimento di target, il tutto in real-time.

APPENDIX

A

FILE MATLAB, ARDUINO E C

A.1 FILE MATLAB

A.1.1 *Inverse kinematics*

```
function [CYL_DISP, CYL_LEN, ERR] = World2Joint(XYZ, RPY, ...
    CYL_LEN_home, CYL_LEN_min, CYL_LEN_max, ...
    CYL_XYZb, CYL_XYZp)



%%  Temporary state to be used in case of kinematics inversion
    errors
persistent CYL_LEN_prev;

if isempty(CYL_LEN_prev),
    CYL_LEN_prev = zeros(6,1);
    CYL_LEN_prev = CYL_LEN_home;
end;

%%  Compute cylinder elongations
a = zeros(3,6);
CYL_LEN = zeros(6,1);
ERR = zeros(6,1);
Rbp = get_DCM(RPY);
for k=1:6,
    a(:,k) = Rbp*CYL_XYZp(:,k) + XYZ - CYL_XYZb(:,k);
    CYL_LEN(k) = norm(a(:,k));
    if CYL_LEN(k) < CYL_LEN_min(k),
        ERR(k) = -k;
    elseif CYL_LEN(k) > CYL_LEN_max(k),
        ERR(k) = k;
    end;
end;

%%  In case of errors, keep the same pose
if any(ERR),
    CYL_LEN = CYL_LEN_prev;
end;

%%  Output cylinders displacements
CYL_DISP = zeros(6,1);
CYL_DISP = CYL_LEN - CYL_LEN_home;

%%  Update previous value of L
CYL_LEN_prev = CYL_LEN;
```

```
end     %   World2Joint

% .....................................................%
function DCM = get_DCM(RPY)

cr = cos(RPY(1));     sr = sin(RPY(1));
cp = cos(RPY(2));     sp = sin(RPY(2));
cy = cos(RPY(3));     sy = sin(RPY(3));

DCM = zeros(3,3);
DCM(1,1) = cy*cp;
DCM(2,1) = sy*cp;
DCM(3,1) = -sp;
DCM(1,2) = cy*sp*sr - sy*cr;
DCM(2,2) = sy*sp*sr + cy*cr;
DCM(3,2) = cp*sr;
DCM(1,3) = cy*sp*cr + sy*sr;
DCM(2,3) = sy*sp*cr - cy*sr;
DCM(3,3) = cp*cr;

end     %   get_DCM
% .....................................................%
```

A.1.2 *Identificazione PAN*

```
%%  Prepare data for sys id

load expdata/pan_input_noise_power0_0001_24V_A2V_1_4.mat;
load expdata/pan_output_noise_power0_0001_24V_A2V_1_4.mat;

%   select I/O data
u = sign_input.signals.values(:,1); %HSM electrical angle [rad]
y = sign_output.signals.values(:,1); %accelerometer output [V]

Ts = 0.001;  % sampling time [s]

%   remove non-zero operating point
uf = detrend(u);   %remove DC value (const vel case)
yf = detrend(y);   %remove DC value (const vel case)

%   define data for sys id
z = iddata(yf, uf, Ts);

%%  Spectral analysis

%   freq resp id using ETFE
g = etfe(z, 10000, 10000);

w = g.Frequency;
f = w/2/pi;
```

```
frGi = squeeze(g.ResponseData); %in: electrical angle, out: lin
    acc

%   add a double integrator to get position out
sysI = zpk([], 1, Ts, Ts);%discrete-time integrator
frI = freqresp(sysI, w);
frI = squeeze(frI);

frPi = frGi.*frI.^2;%in: electrical angle, out: load mech angle

[coh, f_coh] = mscohere(uf, yf, [], [], 10000, 1/Ts);

%   evaluate the DC gain "gc" of "frPi"
f1 = 3;
f2 = 10;
[m1,k1] = min(abs(f-f1));
[m2,k2] = min(abs(f-f2));
gc = mean(abs(frPi(k1:k2)));

%   adjust "frPi" to get the expected DC gain "ge"
ge = 1/50/7;
frPi = (ge/gc)*frPi;

%%  Parametric id (LS fitting on frequency response data)

f1 = 10;
f2 = 40;
[m1,k1] = min(abs(f-f1));
[m2,k2] = min(abs(f-f2));

[numPi, denPi] = invfreqz(frPi(k1:k2), f(k1:k2)/(0.5/Ts)*pi, 1,
    2);
sysPi = tf(numPi, denPi, Ts, 'ioDelay', 0);

%%  Save results
save idres_PAN  sysPi  frPi  f  coh  f_coh  Ts;

%% Plot results

% Plot etfe results
figure;
subplot(3,1,1)
semilogx(f, 20*log10(abs(frPi)), ...
    'LineWidth', 0.5, 'LineStyle', '-', 'Color', 0.6*[1 1 1]);
hold on
ylabel('Magnitude [dB]');
grid on;
%title('Bode diagram')
xlim([1 100])
ylim([-100 -20])
set(gca, 'Box', 'off');

```

```
subplot(3,1,2)
semilogx(f, unwrap(angle(frPi))*180/pi-360, ...
    'LineWidth', 0.5, 'LineStyle', '-', 'Color', 0.6*[1 1 1]);
hold on
ylabel('Phase [deg]');
grid on;
xlim([1 100])
ylim([-400 90])
set(gca, 'Box', 'off');

subplot(3,1,3)
semilogx(f_coh, coh, ...
    'LineWidth', 0.5, 'LineStyle', '-', 'Color', 0.6*[1 1 1]);
hold on
xlabel('Frequency [Hz]');
ylabel('Coherence fcn');
grid on;
xlim([1 100])
set(gca, 'Box', 'off');

% Plot invfreqz results
[magPi, phaPi] = bode(sysPi, 2*pi*f);
magPi = squeeze(magPi);
phaPi = squeeze(phaPi);

subplot(3,1,1)
semilogx(f, 20*log10(magPi), ...
    'LineWidth', 1, 'LineStyle', '-', 'Color', 'k');

subplot(3,1,2)
semilogx(f, phaPi-360, ...
    'LineWidth', 1, 'LineStyle', '-', 'Color', 'k');
```

A.1.3 *Identificazione TILT*

```
%%  Prepare data for sys id

load expdata/tilt_input_noise_power0_00002_24V_A2V_1_4.mat;
load expdata/tilt_output_noise_power0_00002_24V_A2V_1_4.mat;

%   select I/O data
u = sign_input.signals.values(:,1); %HSM electrical angle [rad]
y = sign_output.signals.values(:,1); %accelerometer output [V]

Ts = 0.001;  % sampling time [s]

%   remove non-zero operating point
uf = detrend(u);   %   remove DC value (const vel case)
yf = detrend(y);   %   remove DC value (const vel case)

%   define data for sys id
```

```
z = iddata(yf, uf, Ts);

%%  Spectral analysis

%   freq resp id using ETFE
g = etfe(z, 10000, 10000);

w = g.Frequency;
f = w/2/pi;
frGi = squeeze(g.ResponseData); %in: electrical angle, out: lin
    acc

%   add a double integrator to get position out
sysI = zpk([], 1, Ts, Ts); %discrete-time integrator
frI = freqresp(sysI, w);
frI = squeeze(frI);

frPi = frGi.*frI.^2; %in: electrical angle, out: load mech angle

[coh, f_coh] = mscohere(uf, yf, [], [], 10000, 1/Ts);

%   evaluate the DC gain "gc" of "frPi"
f1 = 10;
f2 = 30;
[m1,k1] = min(abs(f-f1));
[m2,k2] = min(abs(f-f2));
gc = mean(abs(frPi(k1:k2)));

%   adjust "frPi" to get the expected DC gain "ge"
ge = 1/50/7;
frPi = (ge/gc)*frPi;

%%  Parametric id (LS fitting on frequency response data)

f1 = 40;
f2 = 130;
[m1,k1] = min(abs(f-f1));
[m2,k2] = min(abs(f-f2));

[numPi, denPi] = invfreqz(frPi(k1:k2), f(k1:k2)/(0.5/Ts)*pi, 4,
    5);
sysPi = tf(numPi, denPi, Ts, 'ioDelay', 0)

%%  Save results
save idres_PAN  sysPi  frPi  f  coh  f_coh  Ts;

%% Plot results

% Plot etfe results
%figure;
subplot(3,1,1)
semilogx(f, 20*log10(abs(frPi)), ...
```

```
    'LineWidth', 0.5, 'LineStyle', '--', 'Color', 0.6*[1 1 1]);
hold on
%ylabel('Magnitude [dB]');
grid on;
%title('Bode diagram')
xlim([10 200])
ylim([-100 -20])
set(gca, 'Box', 'off');

subplot(3,1,2)
semilogx(f, unwrap(angle(frPi))*180/pi, ...
    'LineWidth', 0.5, 'LineStyle', '--', 'Color', 0.6*[1 1 1]);
hold on
%ylabel('Phase [deg]');
grid on;
xlim([10 200])
ylim([-400 90])
set(gca, 'Box', 'off');

subplot(3,1,3)
semilogx(f_coh, coh, ...
    'LineWidth', 0.5, 'LineStyle', '--', 'Color', 0.6*[1 1 1]);
hold on
xlabel('Frequency [Hz]');
%ylabel('Coherence function');
grid on;
xlim([10 200])
set(gca, 'Box', 'off');

% Plot invfreqz results
[magPi, phaPi] = bode(sysPi, 2*pi*f);
magPi = squeeze(magPi);
phaPi = squeeze(phaPi);

subplot(3,1,1)
semilogx(f, 20*log10(magPi), ...
    'LineWidth', 1, 'LineStyle', '-', 'Color', 'k');

subplot(3,1,2)
semilogx(f, phaPi-360, ...
    'LineWidth', 1, 'LineStyle', '-', 'Color', 'k');
```

A.1.4 *Video Elaboration*

```
clc
clear all
close all

%% define video parameters

x_resolution = 320;
```

```
y_resolution = 240;
fps=25;

%% selection video input

%chose_video_input = 1 for video_onda1_compensato
%chose_video_input = 2 for video_onda1_non_compensato
%chose_video_input = 3 for video_onda2_compensato
%chose_video_input = 4 for video_onda2_non_compensato
%chose_video_input = 5 for video_sinusoidi_compensato
%chose_video_input = 6 for video_sinusoidi_non_compensato

chose_video_input = 5;

%% selection part of video input for sine curve signal (ignore if
    you use another signal)

%chose motion_for_sin = 1 to yaw motion
%chose_motion_for_sin = 2 to roll motion
%chose_motion_for_sin = 3 to pitch motion
%chose_motion_for_sin = 4 to pitch and yaw motion
%chose_motion_for_sin = 5 to all motion

chose_motion_for_sin = 5;

%% selection the type of roll compensation

%imag_rotate = 0 to not_compensated
%image_rotate = 1 to compensated with IMU data
%image_rotate = 2 to compensated with image data

img_rotate = 1;

%% define the video elaboration parameters

switch chose_video_input
    case 1
        Vobj_input=VideoReader('./video_input/si_comp_onda1.avi')
            ;
        time_movement=97; % 1' e 37''
        frame_start = 80;
        frame_stop = time_movement*fps+frame_start;
        Vobj_output = VideoWriter('./video_output/
            onda1_compensate_post_elaboration');

    case 2
        Vobj_input=VideoReader('./video_input/no_comp_onda1.avi')
            ;
        time_movement=97; % 1' e 37''
        frame_start = 1630;
        frame_stop = time_movement*fps+frame_start;
        img_rotate=0;
```

```
        Vobj_output = VideoWriter('./video_output/
            onda1_no_compensate_post_elaboration');

    case 3
        Vobj_input=VideoReader('./video_input/si_comp_onda2.avi')
            ;
        time_video=147.2393; % 2' e 27''
        frame_start = 2500;
        frame_stop = time_video*fps+frame_start;
        Vobj_output = VideoWriter('./video_output/
            onda2_compensate_post_elaboration');

    case 4
        Vobj_input=VideoReader('./video_input/no_comp_onda2.avi')
            ;
        time_video=147.2393; % 2' e 27''
        frame_start = 77;
        frame_stop = time_video*fps+frame_start;
        img_rotate=0;
        Vobj_output = VideoWriter('./video_output/
            onda2_no_compensate_post_elaboration');

    case 5
        Vobj_input=VideoReader('./video_input/si_comp_sin.avi');
        frame_start_movement=95;
        time_total_movement=140; % 1' e 20''
        time_single_movement=30; % in sec
        time_pause=5; % in sec

        frame_start_yaw = frame_start_movement+time_pause*fps;
        frame_stop_yaw = frame_start_yaw+time_single_movement*fps
            ;
        frame_start_roll = frame_stop_yaw+time_pause*fps;
        frame_stop_roll = frame_start_roll+time_single_movement*
            fps;
        frame_start_pitch = frame_stop_roll+time_pause*fps;
        frame_stop_pitch = frame_start_pitch+time_single_movement
            *fps;
        frame_start_yaw_pitch = frame_stop_pitch+time_pause*fps;
        frame_stop_yaw_pitch = frame_start_yaw_pitch+
            time_single_movement*fps

        switch chose_motion_for_sin

            case 1
                frame_start = frame_start_yaw;
                frame_stop = frame_stop_yaw;
                Vobj_output = VideoWriter('./video_output/
                    yaw_compensate_post_elaboration');
            case 2
                frame_start = frame_start_roll;
                frame_stop = frame_stop_roll;
```

```
                Vobj_output = VideoWriter('./video_output/
                    roll_compensate_post_elaboration');
            case 3
                frame_start = frame_start_pitch;
                frame_stop = frame_stop_pitch;
                Vobj_output = VideoWriter('./video_output/
                    pitch_compensate_post_elaboration');
            case 4
                frame_start = frame_start_yaw_pitch;
                frame_stop = frame_stop_yaw_pitch;
                Vobj_output = VideoWriter('./video_output/
                    yaw_pitch_compensate_post_elaboration');
            case 5
                frame_start = frame_start_yaw;
                frame_stop = time_total_movement*fps;
                Vobj_output = VideoWriter('./video_output/
                    all_movenent_compensate_post_elaboration');
        end

    case 6
        Vobj_input=VideoReader('./video_input/no_comp_sin.avi');
        img_rotate=0;
        frame_start_movement=77;
        time_total_movement=140; %in sec
        time_single_movement=30;
        time_pause=5;

        frame_start_yaw = frame_start_movement+time_pause*fps;
        frame_stop_yaw = frame_start_yaw+time_single_movement*fps
            ;
        frame_start_roll = frame_stop_yaw+time_pause*fps;
        frame_stop_roll = frame_start_roll+time_single_movement*
            fps;
        frame_start_pitch = frame_stop_roll+time_pause*fps;
        frame_stop_pitch = frame_start_pitch+time_single_movement
            *fps;
        frame_start_yaw_pitch = frame_stop_pitch+time_pause*fps;
        frame_stop_yaw_pitch = frame_start_yaw_pitch+
            time_single_movement*fps;

        switch chose_motion_for_sin
            case 1
                frame_start = frame_start_yaw;
                frame_stop = frame_stop_yaw;
                Vobj_output = VideoWriter('./video_output/
                    yaw_not_compensate_post_elaboration');
            case 2
                frame_start = frame_start_roll;
                frame_stop = frame_stop_roll;
                Vobj_output = VideoWriter('./video_output/
                    roll_not_compensate_post_elaboration');
            case 3
```

```
                frame_start = frame_start_pitch;
                frame_stop = frame_stop_pitch;
                Vobj_output = VideoWriter('./video_output/
                    pitch_not_compensate_post_elaboration');
            case 4
                frame_start = frame_start_yaw_pitch;
                frame_stop = frame_stop_yaw_pitch;
                Vobj_output = VideoWriter('./video_output/
                    yaw_pitch_not_compensate_post_elaboration');
            case 5
                frame_start = frame_start_yaw;
                frame_stop = time_total_movement*fps;
                Vobj_output = VideoWriter('./video_output/
                    all_movenent_not_compensate_post_elaboration'
                    );
        end
end

%% video elaboration function

[xy roll_angles]= video_elaboration(Vobj_input,Vobj_output,
    x_resolution,y_resolution,fps,frame_start,frame_stop,
    img_rotate);

%% close the video output

close(Vobj_output);
```

A.1.5 *Function Video Elaboration*

```
function [ xy, roll_angles ] = video_elaboration(Vobj_input,
    Vobj_output, x_resolution,y_resolution,fps,frame_start,
    frame_stop,img_rotate)

% IMU roll signal pre-processing
if (img_rotate == 1)
    load ./misure/comp_sin_cam_roll.mat
    roll = roll_signal_elaboration(roll);
    for i = 1 : length(roll.time)%define the index of value at 5
        sec
        if (roll.time(i) < (5))
            k=i;
        end
    end
 end

Vobj_output.FrameRate = fps; %define the video output frame rate
open(Vobj_output);
xy = zeros(2,frame_stop-frame_start);
% Read one frame at a time.
i=1;
```

```
for f=frame_start:frame_stop
    Im=read(Vobj_input,f); %create one immage of each frame
    Im_BW = rgb2gray(Im); %Convert RGB image or colormap to
        grayscale
    Im_BW = imadjust(Im_BW);%Adjust image intensity values or
        colormap
    BW = im2bw(Im_BW, 0.4);
    figure(1)        %convert image to binary image
    imshow(Im_BW);   %Display image
    hold on

    %hough transform and line detection
    [m m1 q1 m2 q2] = line_detection_hough(BW,x_resolution,
        y_resolution);

    %find center
    [x y] = find_center(m1,m2,q1,q2);
    xy(1,f-frame_start+1)=x;
    xy(2,f-frame_start+1)=y;

    current_frame=getframe; %capture the current frame
    %rotate the output image
    switch img_rotate
        case 0
            writeVideo(Vobj_output,current_frame);
        case 1
            X = frame2im(current_frame);%Convert image data to
                movie frame
            B = imrotate(X,-roll.signals.values(k),'crop'); %
                Rotate image
            writeVideo(Vobj_output,B);
            roll_angles.signals.values(i,1)=180/pi*atan(m);
            roll_angles.signals.values(i,2)=roll.signals.values(k
                );
            roll_angles.time(i)= roll.time(k);
            k=k+4;
            i=i+1;
        case 2
            X = frame2im(current_frame); %Convert image data to
                movie frame
            B = imrotate(X,180/pi*atan(m),'crop'); %Rotate image
            writeVideo(Vobj_output,B);
    end
    close % close the figure in order to speed the video-
        elaboration process
end
```

A.1.6 *Function Line Detection Hough*

```
function [m , m1 , q1 , m2 , q2]=line_detection_hough(BW,
    x_resolution,y_resolution)
```

```
%% compute the Hough transform

ThetaResolution=0.5;
[H,T,R] = hough(~(BW),'RhoResolution',0.5,'Theta',-90:
    ThetaResolution:90-ThetaResolution);
P  = houghpeaks(H,6);   %Identify peaks in Hough transform for
    line detection

%% create the RT matrix in according to the variable size of P
    matrix

ERROR=0;
[m n]=size(P);
switch m
   case 2
        rho1    = R(P(1,1));
        theta1  = T(P(1,2));
        rho2    = R(P(2,1));
        theta2  = T(P(2,2));

        RT=[rho1 theta1;
            rho2 theta2];

    case 3
        rho1    = R(P(1,1));
        theta1  = T(P(1,2));
        rho2    = R(P(2,1));
        theta2  = T(P(2,2));
        rho3    = R(P(3,1));
        theta3  = T(P(3,2));

        RT=[rho1 theta1;
            rho2 theta2;
            rho3 theta3];

    case 4
        rho1    = R(P(1,1));
        theta1  = T(P(1,2));
        rho2    = R(P(2,1));
        theta2  = T(P(2,2));
        rho3    = R(P(3,1));
        theta3  = T(P(3,2));
        rho4    = R(P(4,1));
        theta4  = T(P(4,2));

        RT=[rho1 theta1;
            rho2 theta2;
            rho3 theta3;
            rho4 theta4];

    case 5
        rho1    = R(P(1,1));
```

```
        theta1  = T(P(1,2));
        rho2    = R(P(2,1));
        theta2  = T(P(2,2));
        rho3    = R(P(3,1));
        theta3  = T(P(3,2));
        rho4    = R(P(4,1));
        theta4  = T(P(4,2));
        rho5    = R(P(5,1));
        theta5  = T(P(5,2));

        RT=[rho1 theta1;
            rho2 theta2;
            rho3 theta3;
            rho4 theta4;
            rho5 theta5];

     case 6
        rho1    = R(P(1,1));
        theta1  = T(P(1,2));
        rho2    = R(P(2,1));
        theta2  = T(P(2,2));
        rho3    = R(P(3,1));
        theta3  = T(P(3,2));
        rho4    = R(P(4,1));
        theta4  = T(P(4,2));
        rho5    = R(P(5,1));
        theta5  = T(P(5,2));
        rho6    = R(P(6,1));
        theta6  = T(P(6,2));

        RT=[rho1 theta1;
            rho2 theta2;
            rho3 theta3;
            rho4 theta4;
            rho5 theta5;
            rho6 theta6];

    otherwise
        ERROR=1
        return
end
%% identify the correct peacks

k=0;
j=0;
t=0;

for k = 1 : m
    if (RT(k,2)>45 || RT(k,2)<-45)
        if (RT(k,2) == 90 || RT(k,2) == -90)
            t=k;
        else
```

```
            break;
        end
    end
end
if RT(k,2)>45 || RT(k,2)<-45
    k=k;
else
    k=t;
end

for j = 1 : m
    if (RT(j,2)>=-45 && RT(j,2)<=45)
        if RT(j,2)==0
            t=j;
        else
            break;
        end
    end
end
if (RT(j,2)>=-45 && RT(j,2)<=45)
    j=j;
else
    j=t;
end

rho1=RT(k,1);
theta1=RT(k,2);
rho2=RT(j,1);
theta2=RT(j,2);

%% color lines

if theta1>-45 && theta1<45
    color_string_theta1='g';
    color_string_theta2='r';
else
    color_string_theta1='r';
    color_string_theta2='g';
end

%% plot lines

theta1=theta1*pi/180;%deg2rad
theta2=theta2*pi/180;%deg2rad

if theta1~=0
    plot(line([1 x_resolution],[((rho1-cos(theta1))/sin(theta1))
        ((rho1-x_resolution*cos(theta1))/sin(theta1))],'Color',
        color_string_theta1,'LineWidth',2)) % inclinata
    m = -cos(theta1)/sin(theta1); %for image_rotate
    m1 = -cos(theta1)/sin(theta1); %for find_center_function
    q1 = rho1/sin(theta1);  %for find_center_function
```

```
else
    plot(line([abs(rho1) abs(rho1)],[1 y_resolution],'Color',
        color_string_theta1,'LineWidth',2)) %verticale
    m = -cos(theta2)/sin(theta2); %for image_rotate
    m1 = 1.5; %for find_center_function
    q1 = rho1; %for find_center_function
end

if theta2~=0
    plot(line([1 x_resolution],[((rho2-cos(theta2))/sin(theta2))
        ((rho2-x_resolution*cos(theta2))/sin(theta2))],'Color',
        color_string_theta2,'LineWidth',2))% inclinata
    m2=-cos(theta2)/sin(theta2); %for find_center_function
    q2=rho2/sin(theta2); %for find_center_function
else
    plot(line([abs(rho2) abs(rho2)],[1 y_resolution],'Color',
        color_string_theta2,'LineWidth',2))%verticale
    m2=1.5; %for find_center_function
    q2=rho2; %for find_center_function
end

end
```

A.1.7 *Function Find Center*

```
function [x , y] = find_center(m1, m2, q1, q2);
% y = m1*x+q1
% y = m2*x+q2
% m2 * x + q2 = m1 * x + q1
% x(m2 - m1) = q1 - q2

x = (q1-q2)/(m2-m1);
y = m1*((q1-q2)/(m2-m1))+q1;

if m1 == 1.5 % theta1=0
    x=q1;
    if m2 == 1.5
        y=q2;
    else
        y=m2*x+q2;
    end
end

if m2 == 1.5 % theta2=0
    x=q2;
    if m1 == 1.5
        y=q1;
    else
        y=m1*x+q1;
    end
end
```

```
end
```

A.2 FILE ARDUINO

A.2.1 *Compass*

```
/* This file is part of the Razor AHRS Firmware */

void Compass_Heading()
{
  float mag_x;
  float mag_y;
  float cos_roll;
  float sin_roll;
  float cos_pitch;
  float sin_pitch;

  cos_roll = cos(roll);
  sin_roll = sin(roll);
  cos_pitch = cos(pitch);
  sin_pitch = sin(pitch);

  // Tilt compensated magnetic field X
  mag_x = magnetom[0]*cos_pitch + magnetom[1]*sin_roll*sin_pitch
      + magnetom[2]*cos_roll*sin_pitch;
  // Tilt compensated magnetic field Y
  mag_y = magnetom[1]*cos_roll - magnetom[2]*sin_roll;
  // Magnetic Heading
  MAG_Heading = atan2(-mag_y, mag_x);
}
```

A.2.2 *DCM*

```
/* This file is part of the Razor AHRS Firmware */

// DCM algorithm

/**************************************************/
void Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])
      *.5; //eq.19

```

```
  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq
      .19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq
      .19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix
      [0][0]);//eq.19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix
      [1][0]);//eq.19

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&
      temporary[1][0]); // c= a x b //eq.20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary
      [0][0])); //eq.21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary
      [1][0])); //eq.21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary
      [2][0])); //eq.21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/**************************************************/
void Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
  //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


  //*****Roll and Pitch***************

  // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] +
      Accel_Vector[1]*Accel_Vector[1] + Accel_Vector[2]*
      Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to
      gravity.
  // Dynamic weighting of accelerometer info (reliability filter)
  // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G
       = 0.0)
  Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);
      //
```

```
  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&
      DCM_Matrix[2][0]); //adjust the ground of reference
  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*
      Accel_weight);

  Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH
      *Accel_weight);
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

  //*****YAW***************
  // We make the gyro YAW drift correction based on compass
      magnetic heading

  mag_heading_x = cos(MAG_Heading);
  mag_heading_y = sin(MAG_Heading);
  errorCourse=(DCM_Matrix[0][0]*mag_heading_y) - (DCM_Matrix
      [1][0]*mag_heading_x);  //Calculating YAW error
  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys
      the yaw correction to the XYZ rotation of the aircraft,
      depeding the position.

  Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01
      proportional of YAW.
  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding
      Proportional.

  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001
      Integrator
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator
      to the Omega_I
}

void Matrix_update(void)
{
  Gyro_Vector[0]=GYRO_SCALED_RAD(gyro[0]); //gyro x roll
  Gyro_Vector[1]=GYRO_SCALED_RAD(gyro[1]); //gyro y pitch
  Gyro_Vector[2]=GYRO_SCALED_RAD(gyro[2]); //gyro z yaw

  Accel_Vector[0]=accel[0];
  Accel_Vector[1]=accel[1];
  Accel_Vector[2]=accel[2];

  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding
      proportional term
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding
      Integrator term

#if DEBUG__NO_DRIFT_CORRECTION == true // Do not use drift
    correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
```

```
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
#else // Use drift correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
#endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a
      *b=c

  for(int x=0; x<3; x++) //Matrix Addition (update)
  {
    for(int y=0; y<3; y++)
    {
      DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
    }
  }
}

void Euler_angles(void)
{
  pitch = -asin(DCM_Matrix[2][0]);
  roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
  yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}
```

A.2.3 *Output*

```
/* This file is part of the Razor AHRS Firmware */

// Output angles: yaw, pitch, roll
void output_angles()
{
  if (output_mode == OUTPUT__MODE_ANGLES_BINARY)
  {
    /* New output format: 0xFF00FF00,<yaw>,<pitch>,<roll> */
    /* Modified by: Riccardo A. on July 17th, 2012 */
    float ypr[3];
    ypr[0] = TO_DEG(yaw);
```

```
    ypr[1] = TO_DEG(pitch);
    ypr[2] = TO_DEG(roll);

    Serial.write((byte) 0xFF);
    Serial.write((byte) 0x00);
    Serial.write((byte) 0xFF);
    Serial.write((byte) 0x00);
    Serial.write((byte*) ypr, 12);  // No new-line

    /* Original code */
    /* float ypr[3];
    ypr[0] = TO_DEG(yaw);
    ypr[1] = TO_DEG(pitch);
    ypr[2] = TO_DEG(roll);
    Serial.write((byte*) ypr, 12);  // No new-line */
  }
  else if (output_mode == OUTPUT__MODE_ANGLES_TEXT)
  {
      /* New output format: $<yaw>,<pitch>,<roll>#\r\n */
      /* Modified by: Riccardo A. on July 04th, 2012 */
      Serial.print("$");
      Serial.print(TO_DEG(yaw)); Serial.print(",");
      Serial.print(TO_DEG(pitch)); Serial.print(",");
      Serial.print(TO_DEG(roll));
      Serial.println("#"); Serial.println();

      /* Original code */
      /*Serial.print("#YPR=");
      Serial.print(TO_DEG(yaw)); Serial.print(",");
      Serial.print(TO_DEG(pitch)); Serial.print(",");
      Serial.print(TO_DEG(roll)); Serial.println();*/
  }
}

void output_calibration(int calibration_sensor)
{
  if (calibration_sensor == 0)  // Accelerometer
  {
    // Output MIN/MAX values
    Serial.print("accel x,y,z (min/max) = ");
    for (int i = 0; i < 3; i++) {
      if (accel[i] < accel_min[i]) accel_min[i] = accel[i];
      if (accel[i] > accel_max[i]) accel_max[i] = accel[i];
      Serial.print(accel_min[i]);
      Serial.print("/");
      Serial.print(accel_max[i]);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
  else if (calibration_sensor == 1)  // Magnetometer
```

```
  {
    // Output MIN/MAX values
    Serial.print("magn x,y,z (min/max) = ");
    for (int i = 0; i < 3; i++) {
      if (magnetom[i] < magnetom_min[i]) magnetom_min[i] =
          magnetom[i];
      if (magnetom[i] > magnetom_max[i]) magnetom_max[i] =
          magnetom[i];
      Serial.print(magnetom_min[i]);
      Serial.print("/");
      Serial.print(magnetom_max[i]);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
  else if (calibration_sensor == 2)  // Gyroscope
  {
    // Average gyro values
    for (int i = 0; i < 3; i++)
      gyro_average[i] += gyro[i];
    gyro_num_samples++;

    // Output current and averaged gyroscope values
    Serial.print("gyro x,y,z (current/average) = ");
    for (int i = 0; i < 3; i++) {
      Serial.print(gyro[i]);
      Serial.print("/");
      Serial.print(gyro_average[i] / (float) gyro_num_samples);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
}

void output_sensors()
{
  Serial.print("#ACC=");
  Serial.print(accel[0]); Serial.print(",");
  Serial.print(accel[1]); Serial.print(",");
  Serial.print(accel[2]); Serial.println();

  Serial.print("#MAG=");
  Serial.print(magnetom[0]); Serial.print(",");
  Serial.print(magnetom[1]); Serial.print(",");
  Serial.print(magnetom[2]); Serial.println();

  Serial.print("#GYR=");
  Serial.print(gyro[0]); Serial.print(",");
  Serial.print(gyro[1]); Serial.print(",");
  Serial.print(gyro[2]); Serial.println();
}
```

A.2.4 *Math*

```
/* This file is part of the Razor AHRS Firmware */

// Computes the dot product of two vectors
float Vector_Dot_Product(float vector1[3], float vector2[3])
{
  float op=0;

  for(int c=0; c<3; c++)
  {
    op+=vector1[c]*vector2[c];
  }

  return op;
}

// Computes the cross product of two vectors
void Vector_Cross_Product(float vectorOut[3], float v1[3], float
    v2[3])
{
  vectorOut[0]= (v1[1]*v2[2]) - (v1[2]*v2[1]);
  vectorOut[1]= (v1[2]*v2[0]) - (v1[0]*v2[2]);
  vectorOut[2]= (v1[0]*v2[1]) - (v1[1]*v2[0]);
}

// Multiply the vector by a scalar.
void Vector_Scale(float vectorOut[3], float vectorIn[3], float
    scale2)
{
  for(int c=0; c<3; c++)
  {
    vectorOut[c]=vectorIn[c]*scale2;
  }
}

// Adds two vectors
void Vector_Add(float vectorOut[3], float vectorIn1[3], float
    vectorIn2[3])
{
  for(int c=0; c<3; c++)
  {
    vectorOut[c]=vectorIn1[c]+vectorIn2[c];
  }
}

//Multiply two 3x3 matrixs. This function developed by Jordi can
    be easily adapted to multiple n*n matrix's. (Pero me da
    flojera!).
void Matrix_Multiply(float a[3][3], float b[3][3],float mat
    [3][3])
{
```

```
  float op[3];
  for(int x=0; x<3; x++)
  {
    for(int y=0; y<3; y++)
    {
      for(int w=0; w<3; w++)
      {
       op[w]=a[x][w]*b[w][y];
      }
      mat[x][y]=0;
      mat[x][y]=op[0]+op[1]+op[2];

      float test=mat[x][y];
    }
  }
}

// Init rotation matrix using euler angles
void init_rotation_matrix(float m[3][3], float yaw, float pitch,
    float roll)
{
  float c1 = cos(roll);
  float s1 = sin(roll);
  float c2 = cos(pitch);
  float s2 = sin(pitch);
  float c3 = cos(yaw);
  float s3 = sin(yaw);

  // Euler angles, right-handed, intrinsic, XYZ convention
  // (which means: rotate around body axes Z, Y', X'')
  m[0][0] = c2 * c3;
  m[0][1] = c3 * s1 * s2 - c1 * s3;
  m[0][2] = s1 * s3 + c1 * c3 * s2;

  m[1][0] = c2 * s3;
  m[1][1] = c1 * c3 + s1 * s2 * s3;
  m[1][2] = c1 * s2 * s3 - c3 * s1;

  m[2][0] = -s2;
  m[2][1] = c2 * s1;
  m[2][2] = c1 * c2;
}
```

A.2.5 *Razor AHRS*

```
* Razor AHRS Firmware v1.4.0
* 9 Degree of Measurement Attitude and Heading Reference System
* for Sparkfun "9DOF Razor IMU" (SEN-10125 and SEN-10736)
* and "9DOF Sensor Stick" (SEN-10183, 10321 and SEN-10724)
*
* Released under GNU GPL (General Public License) v3.0
```

```
* Copyright (C) 2011 Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin
*
* Infos, updates, bug reports and feedback:
*     http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs
*
*
* History:
*   * Original code (http://code.google.com/p/sf9domahrs/) by Doug Weibel and Jose Julio,
*     based on ArduIMU v1.5 by Jordi Munoz and William Premerlani, Jose Julio and Doug Weibel. Thank you!
*
*   * Updated code (http://groups.google.com/group/sf_9dof_ahrs_update) by David Malik (david.zsolt.malik@gmail.com)
*     for new Sparkfun 9DOF Razor hardware (SEN-10125).
*
*   * Updated and extended by Peter Bartz (peter-bartz@gmx.de):
*     * v1.3.0
*       * Cleaned up, streamlined and restructured most of the code to make it more comprehensible.
*       * Added sensor calibration (improves precision and responsiveness a lot!).
*       * Added binary yaw/pitch/roll output.
*       * Added basic serial command interface to set output modes/calibrate sensors/synch stream/etc.
*       * Added support to synch automatically when using Rovering Networks Bluetooth modules (and compatible).
*       * Wrote new easier to use test program (using Processing).
*       * Added support for new version of "9DOF Razor IMU": SEN-10736.
*       --> The output of this code is not compatible with the older versions!
*       --> A Processing sketch to test the tracker is available.
*     * v1.3.1
*       * Initializing rotation matrix based on start-up sensor readings -> orientation OK right away.
*       * Adjusted gyro low-pass filter and output rate settings.
*     * v1.3.2
*       * Adapted code to work with new Arduino 1.0 (and older versions still).
*     * v1.3.3
*       * Improved synching.
*     * v1.4.0
*       * Added support for SparkFun "9DOF Sensor Stick" (versions SEN-10183, SEN-10321 and SEN-10724).
*
* TODOs:
*   * Allow optional use of EEPROM for storing and reading calibration values.
```

```
*   * Use self-test and temperature-compensation features of the
    sensors.
*   * Add binary output of unfused sensor data for all 9 axes.
********/

/*
  "9DOF Razor IMU" hardware versions: SEN-10125 and SEN-10736

  ATMega328@3.3V, 8MHz

  ADXL345  : Accelerometer
  HMC5843  : Magnetometer on SEN-10125
  HMC5883L : Magnetometer on SEN-10736
  ITG-3200 : Gyro

  Arduino IDE : Select board "Arduino Pro or Pro Mini (3.3v, 8Mhz
      ) w/ATmega328"
*/

/*
  "9DOF Sensor Stick" hardware versions: SEN-10183, SEN-10321 and
       SEN-10724

  ADXL345  : Accelerometer
  HMC5843  : Magnetometer on SEN-10183 and SEN-10321
  HMC5883L : Magnetometer on SEN-10724
  ITG-3200 : Gyro
*/

/*
  Axis definition (differs from definition printed on the board!)
      :
    X axis pointing forward (towards the short edge with the
        connector holes)
    Y axis pointing to the right
    and Z axis pointing down.

  Positive yaw   : clockwise
  Positive roll  : right wing down
  Positive pitch : nose up

  Transformation order: first yaw then pitch then roll.
*/

/*
  Commands that the firmware understands:

  "#o<param>" - Set output parameter. The available options are:
      "#o0" - Disable continuous streaming output.
      "#o1" - Enable continuous streaming output.
      "#ob" - Output angles in binary format (yaw/pitch/roll as
          binary float, so one output frame
```

```
89              is 3x4 = 12 bytes long).
      "#ot" - Output angles in text format (Output frames have
          form like "#YPR=-142.28,-5.38,33.52",
              followed by carriage return and line feed [\r\n]).
92    "#os" - Output (calibrated) sensor data of all 9 axes in
          text format. One frame consist of
              three lines - one for each sensor.
      "#oc" - Go to calibration output mode.
95    "#on" - When in calibration mode, go on to calibrate next
          sensor.
      "#oe0" - Disable error message output.
      "#oe1" - Enable error message output.
98
  "#f" - Request one output frame - useful when continuous output
       is disabled and updates are
        required in larger intervals only.
101 "#s<xy>" - Request synch token - useful to find out where the
      frame boundaries are in a continuous
         binary stream or to see if tracker is present and
             answering. The tracker will send
         "#SYNCH<xy>\r\n" in response (so it's possible to read
             using a readLine() function).
104      x and y are two mandatory but arbitrary bytes that can
             be used to find out which request
         the answer belongs to.

107 ("#C" and "#D" - Reserved for communication with optional
      Bluetooth module.)

  Newline characters are not required. So you could send "#ob#o1#
      s", which
110 would set binary output mode, enable continuous streaming
      output and request
  a synch token all at once.

113 The status LED will be on if streaming output is enabled and
      off otherwise.

  Byte order of binary output is little-endian: least significant
       byte comes first.
116 */



119

/*****************************************************************
    */
/*********** USER SETUP AREA! Set your options here! ************
    */
122 /*****************************************************************
    */

// HARDWARE OPTIONS
```

```
125 /*****************************************************************
    */
    // Select your hardware here by uncommenting one line!
    //#define HW__VERSION_CODE 10125 // SparkFun "9DOF Razor IMU"
    version "SEN-10125" (HMC5843 magnetometer)
128 #define HW__VERSION_CODE 10736 // SparkFun "9DOF Razor IMU"
    version "SEN-10736" (HMC5883L magnetometer)
    //#define HW__VERSION_CODE 10183 // SparkFun "9DOF Sensor Stick"
    version "SEN-10183" (HMC5843 magnetometer)
    //#define HW__VERSION_CODE 10321 // SparkFun "9DOF Sensor Stick"
    version "SEN-10321" (HMC5843 magnetometer)
131 //#define HW__VERSION_CODE 10724 // SparkFun "9DOF Sensor Stick"
    version "SEN-10724" (HMC5883L magnetometer)


134 // OUTPUT OPTIONS
    /*****************************************************************
    */
    // Set your serial port baud rate used to send out data here!
137 #define OUTPUT__BAUD_RATE 57600

    // Sensor data output interval in milliseconds
140 // This may not work, if faster than 20ms (=50Hz)
    // Code is tuned for 20ms, so better leave it like that
    #define OUTPUT__DATA_INTERVAL 20  // in milliseconds
143
    // Output mode
    #define OUTPUT__MODE_CALIBRATE_SENSORS 0 // Outputs sensor min/
    max values as text for manual calibration
146 #define OUTPUT__MODE_ANGLES_TEXT 1 // Outputs yaw/pitch/roll in
    degrees as text
    #define OUTPUT__MODE_ANGLES_BINARY 2 // Outputs yaw/pitch/roll in
     degrees as binary float
    #define OUTPUT__MODE_SENSORS_TEXT 3 // Outputs (calibrated)
    sensor values for all 9 axes as text
149 // Select your startup output mode here!
    int output_mode = OUTPUT__MODE_ANGLES_BINARY; //
    OUTPUT__MODE_ANGLES_TEXT;

152 // Select if serial continuous streaming output is enabled per
    default on startup.
    #define OUTPUT__STARTUP_STREAM_ON true  // true or false

155 // If set true, an error message will be output if we fail to
    read sensor data.
    // Message format: "!ERR: reading <sensor>", followed by "\r\n".
    boolean output_errors = false;  // true or false
158
    // Bluetooth
    // You can set this to true, if you have a Rovering Networks
    Bluetooth Module attached.
```

```
// The connect/disconnect message prefix of the module has to be set to "#".
// (Refer to manual, it can be set like this: SO,#)
// When using this, streaming output will only be enabled as long as we're connected. That way
// receiver and sender are synchronzed easily just by connecting/disconnecting.
// It is not necessary to set this! It just makes life easier when writing code for
// the receiving side. The Processing test sketch also works without setting this.
// NOTE: When using this, OUTPUT__STARTUP_STREAM_ON has no effect!
#define OUTPUT__HAS_RN_BLUETOOTH false  // true or false


// SENSOR CALIBRATION
/*****************************/
// How to calibrate? Read the tutorial at http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs
// Put MIN/MAX and OFFSET readings for your board here!
// Accelerometer
// "accel x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define ACCEL_X_MIN ((float) -250)
#define ACCEL_X_MAX ((float) 250)
#define ACCEL_Y_MIN ((float) -250)
#define ACCEL_Y_MAX ((float) 250)
#define ACCEL_Z_MIN ((float) -250)
#define ACCEL_Z_MAX ((float) 250)

// Magnetometer
// "magn x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define MAGN_X_MIN ((float) -600)
#define MAGN_X_MAX ((float) 600)
#define MAGN_Y_MIN ((float) -600)
#define MAGN_Y_MAX ((float) 600)
#define MAGN_Z_MIN ((float) -600)
#define MAGN_Z_MAX ((float) 600)

// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X  .../OFFSET_Y  .../OFFSET_Z
#define GYRO_AVERAGE_OFFSET_X ((float) 0.0)
#define GYRO_AVERAGE_OFFSET_Y ((float) 0.0)
#define GYRO_AVERAGE_OFFSET_Z ((float) 0.0)

/*
// Calibration example:
// "accel x,y,z (min/max) = -278.00/270.00  -254.00/284.00  -294.00/235.00"
#define ACCEL_X_MIN ((float) -278)
```

```
203 #define ACCEL_X_MAX ((float) 270)
    #define ACCEL_Y_MIN ((float) -254)
    #define ACCEL_Y_MAX ((float) 284)
206 #define ACCEL_Z_MIN ((float) -294)
    #define ACCEL_Z_MAX ((float) 235)

209 // "magn x,y,z (min/max) = -511.00/581.00  -516.00/568.00
        -489.00/486.00"
    #define MAGN_X_MIN ((float) -511)
    #define MAGN_X_MAX ((float) 581)
212 #define MAGN_Y_MIN ((float) -516)
    #define MAGN_Y_MAX ((float) 568)
    #define MAGN_Z_MIN ((float) -489)
215 #define MAGN_Z_MAX ((float) 486)

    //"gyro x,y,z (current/average) = -32.00/-34.82  102.00/100.41
        -16.00/-16.38"
218 #define GYRO_AVERAGE_OFFSET_X ((float) -34.82)
    #define GYRO_AVERAGE_OFFSET_Y ((float) 100.41)
    #define GYRO_AVERAGE_OFFSET_Z ((float) -16.38)
221 */


224 // DEBUG OPTIONS
    /*****************************************************************
        */
    // When set to true, gyro drift correction will not be applied
227 #define DEBUG__NO_DRIFT_CORRECTION false
    // Print elapsed time after each I/O loop
    #define DEBUG__PRINT_LOOP_TIME false
230


    /*****************************************************************
        */
233 /****************** END OF USER SETUP AREA!  *********************
        */
    /*****************************************************************
        */


236
    // Check if hardware version code is defined
    #ifndef HW__VERSION_CODE
239   // Generate compile error
      #error YOU HAVE TO SELECT THE HARDWARE YOU ARE USING! See "
          HARDWARE OPTIONS" in "USER SETUP AREA" at top of Razor_AHRS
           .pde!
    #endif

242
    #include <Wire.h>

245 // Sensor calibration scale and offset values
    #define ACCEL_X_OFFSET ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f)
```

```
#define ACCEL_Y_OFFSET ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f)
#define ACCEL_Z_OFFSET ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f)
#define ACCEL_X_SCALE (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define MAGN_X_OFFSET ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f)
#define MAGN_Y_OFFSET ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f)
#define MAGN_Z_OFFSET ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f)
#define MAGN_X_SCALE (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET))
#define MAGN_Y_SCALE (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET))
#define MAGN_Z_SCALE (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET))


// Gain for gyroscope (ITG-3200)
#define GYRO_GAIN 0.06957 // Same gain on all axes
#define GYRO_SCALED_RAD(x) (x * TO_RAD(GYRO_GAIN)) // Calculate
    the scaled gyro readings in radians per second

// DCM parameters
#define Kp_ROLLPITCH 0.02f
#define Ki_ROLLPITCH 0.00002f
#define Kp_YAW 1.2f
#define Ki_YAW 0.00002f

// Stuff
#define STATUS_LED_PIN 13  // Pin number of status LED
#define GRAVITY 256.0f // "1G reference" used for DCM filter and
    accelerometer calibration
#define TO_RAD(x) (x * 0.01745329252)  // *pi/180
#define TO_DEG(x) (x * 57.2957795131)  // *180/pi

// Sensor variables
float accel[3];  // Actually stores the NEGATED acceleration (
    equals gravity, if board not moving).
float accel_min[3];
float accel_max[3];

float magnetom[3];
float magnetom_min[3];
float magnetom_max[3];

float gyro[3];
float gyro_average[3];
int gyro_num_samples = 0;

// DCM variables
float MAG_Heading;
float Accel_Vector[3]= {0, 0, 0}; // Store the acceleration in a
    vector
float Gyro_Vector[3]= {0, 0, 0}; // Store the gyros turn rate in
    a vector
```

```
float Omega_Vector[3]= {0, 0, 0}; // Corrected Gyro_Vector data
float Omega_P[3]= {0, 0, 0}; // Omega Proportional correction
float Omega_I[3]= {0, 0, 0}; // Omega Integrator
float Omega[3]= {0, 0, 0};
float errorRollPitch[3] = {0, 0, 0};
float errorYaw[3] = {0, 0, 0};
float DCM_Matrix[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
float Update_Matrix[3][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
float Temporary_Matrix[3][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}};

// Euler angles
float yaw;
float pitch;
float roll;

// DCM timing in the main loop
long timestamp;
long timestamp_old;
float G_Dt; // Integration time for DCM algorithm

// More output-state variables
boolean output_stream_on;
boolean output_single_on;
int curr_calibration_sensor = 0;
boolean reset_calibration_session_flag = true;
int num_accel_errors = 0;
int num_magn_errors = 0;
int num_gyro_errors = 0;

void read_sensors() {
  Read_Gyro(); // Read gyroscope
  Read_Accel(); // Read accelerometer
  Read_Magn(); // Read magnetometer
}

// Read every sensor and record a time stamp
// Init DCM with unfiltered orientation
// TODO re-init global vars?
void reset_sensor_fusion() {
  float temp1[3];
  float temp2[3];
  float xAxis[] = {1.0f, 0.0f, 0.0f};

  read_sensors();
  timestamp = millis();

  // GET PITCH
  // Using y-z-plane-component/x-component of gravity vector
  pitch = -atan2(accel[0], sqrt(accel[1] * accel[1] + accel[2] *
      accel[2]));

  // GET ROLL
```

```
  // Compensate pitch of gravity vector
  Vector_Cross_Product(temp1, accel, xAxis);
  Vector_Cross_Product(temp2, xAxis, temp1);
  // Normally using x-z-plane-component/y-component of
      compensated gravity vector
  // roll = atan2(temp2[1], sqrt(temp2[0] * temp2[0] + temp2[2] *
       temp2[2]));
  // Since we compensated for pitch, x-z-plane-component equals z
      -component:
  roll = atan2(temp2[1], temp2[2]);

  // GET YAW
  Compass_Heading();
  yaw = MAG_Heading;

  // Init rotation matrix
  init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);
}

// Apply calibration to raw sensor readings
void compensate_sensor_errors() {
    // Compensate accelerometer error
    accel[0] = (accel[0] - ACCEL_X_OFFSET) * ACCEL_X_SCALE;
    accel[1] = (accel[1] - ACCEL_Y_OFFSET) * ACCEL_Y_SCALE;
    accel[2] = (accel[2] - ACCEL_Z_OFFSET) * ACCEL_Z_SCALE;

    // Compensate magnetometer error
    magnetom[0] = (magnetom[0] - MAGN_X_OFFSET) * MAGN_X_SCALE;
    magnetom[1] = (magnetom[1] - MAGN_Y_OFFSET) * MAGN_Y_SCALE;
    magnetom[2] = (magnetom[2] - MAGN_Z_OFFSET) * MAGN_Z_SCALE;

    // Compensate gyroscope error
    gyro[0] -= GYRO_AVERAGE_OFFSET_X;
    gyro[1] -= GYRO_AVERAGE_OFFSET_Y;
    gyro[2] -= GYRO_AVERAGE_OFFSET_Z;
}

// Reset calibration session if reset_calibration_session_flag is
     set
void check_reset_calibration_session()
{
  // Raw sensor values have to be read already, but no error
      compensation applied

  // Reset this calibration session?
  if (!reset_calibration_session_flag) return;

  // Reset acc and mag calibration variables
  for (int i = 0; i < 3; i++) {
    accel_min[i] = accel_max[i] = accel[i];
    magnetom_min[i] = magnetom_max[i] = magnetom[i];
  }
```

```
392
  // Reset gyro calibration variables
  gyro_num_samples = 0;  // Reset gyro calibration averaging
395 gyro_average[0] = gyro_average[1] = gyro_average[2] = 0.0f;

  reset_calibration_session_flag = false;
398 }

void turn_output_stream_on()
401 {
  output_stream_on = true;
  digitalWrite(STATUS_LED_PIN, HIGH);
404 }

void turn_output_stream_off()
407 {
  output_stream_on = false;
  digitalWrite(STATUS_LED_PIN, LOW);
410 }

// Blocks until another byte is available on serial port
413 char readChar()
{
  while (Serial.available() < 1) { } // Block
416 return Serial.read();
}

419 void setup()
{
  // Init serial output
422 Serial.begin(OUTPUT__BAUD_RATE);

  // Init status LED
425 pinMode (STATUS_LED_PIN, OUTPUT);
  digitalWrite(STATUS_LED_PIN, LOW);

428 // Init sensors
  delay(50);  // Give sensors enough time to start
  I2C_Init();
431 Accel_Init();
  Magn_Init();
  Gyro_Init();
434
  // Read sensors, init DCM algorithm
  delay(20);  // Give sensors enough time to collect data
437 reset_sensor_fusion();

  // Init output
440 #if (OUTPUT__HAS_RN_BLUETOOTH == true) || (
    OUTPUT__STARTUP_STREAM_ON == false)
  turn_output_stream_off();
#else
```

```
  turn_output_stream_on();
#endif
}

// Main loop
void loop()
{
  // Read incoming control messages
  if (Serial.available() >= 2)
  {
    if (Serial.read() == '#') // Start of new control message
    {
      int command = Serial.read(); // Commands
      if (command == 'f') // request one output _f_rame
        output_single_on = true;
      else if (command == 's') // _s_ynch request
      {
        // Read ID
        byte id[2];
        id[0] = readChar();
        id[1] = readChar();

        // Reply with synch message
        Serial.print("#SYNCH");
        Serial.write(id, 2);
        Serial.println();
      }
      else if (command == 'o') // Set _o_utput mode
      {
        char output_param = readChar();
        if (output_param == 'n')  // Calibrate _n_ext sensor
        {
          curr_calibration_sensor = (curr_calibration_sensor + 1)
              % 3;
          reset_calibration_session_flag = true;
        }
        else if (output_param == 't') // Output angles as _t_ext
          output_mode = OUTPUT__MODE_ANGLES_TEXT;
        else if (output_param == 'b') // Output angles in
            _b_inary form
          output_mode = OUTPUT__MODE_ANGLES_BINARY;
        else if (output_param == 'c') // Go to _c_alibration mode
        {
          output_mode = OUTPUT__MODE_CALIBRATE_SENSORS;
          reset_calibration_session_flag = true;
        }
        else if (output_param == 's') // Output _s_ensor values
            as text
          output_mode = OUTPUT__MODE_SENSORS_TEXT;
        else if (output_param == 'o') // Disable continuous
            streaming output
        {
```

```
491           turn_output_stream_off();
              reset_calibration_session_flag = true;
            }
494         else if (output_param == '1') // Enable continuous
                streaming output
            {
              reset_calibration_session_flag = true;
497           turn_output_stream_on();
            }
            else if (output_param == 'e') // _e_rror output settings
500         {
              char error_param = readChar();
              if (error_param == '0') output_errors = false;
503           else if (error_param == '1') output_errors = true;
              else if (error_param == 'c') // get error count
              {
506             Serial.print("#AMG-ERR:");
                Serial.print(num_accel_errors); Serial.print(",");
                Serial.print(num_magn_errors); Serial.print(",");
509             Serial.println(num_gyro_errors);
              }
            }
512       }
#if OUTPUT__HAS_RN_BLUETOOTH == true
          // Read messages from bluetooth module
515       // For this to work, the connect/disconnect message prefix
              of the module has to be set to "#".
          else if (command == 'C') // Bluetooth "#CONNECT" message (
              does the same as "#o1")
            turn_output_stream_on();
518       else if (command == 'D') // Bluetooth "#DISCONNECT" message
               (does the same as "#o0")
            turn_output_stream_off();
#endif // OUTPUT__HAS_RN_BLUETOOTH == true
521     }
        else
        { } // Skip character
524   }

      // Time to read the sensors again?
527   if((millis() - timestamp) >= OUTPUT__DATA_INTERVAL)
      {
        timestamp_old = timestamp;
530     timestamp = millis();
        if (timestamp > timestamp_old)
          G_Dt = (float) (timestamp - timestamp_old) / 1000.0f; //
              Real time of loop run. We use this on the DCM algorithm
               (gyro integration time)
533     else G_Dt = 0;

        // Update sensor readings
536     read_sensors();
```

```
    if (output_mode == OUTPUT__MODE_CALIBRATE_SENSORS)  // We're
        in calibration mode
    {
      check_reset_calibration_session();  // Check if this
          session needs a reset
      if (output_stream_on || output_single_on)
          output_calibration(curr_calibration_sensor);
    }
    else if (output_mode == OUTPUT__MODE_SENSORS_TEXT)
    {
      // Apply sensor calibration
      compensate_sensor_errors();

      if (output_stream_on || output_single_on) output_sensors();
    }
    else
    {
      // Apply sensor calibration
      compensate_sensor_errors();

      // Run DCM algorithm
      Compass_Heading(); // Calculate magnetic heading
      Matrix_update();
      Normalize();
      Drift_correction();
      Euler_angles();

      if (output_stream_on || output_single_on) output_angles();
    }

    output_single_on = false;

#if DEBUG__PRINT_LOOP_TIME == true
    Serial.print("loop time (ms) = ");
    Serial.println(millis() - timestamp);
#endif
  }
#if DEBUG__PRINT_LOOP_TIME == true
  else
  {
    Serial.println("waiting...");
  }
#endif
}
```

A.2.6 *Sensors*

```
/* This file is part of the Razor AHRS Firmware */

// I2C code to read the sensors
```

```
4
// Sensor I2C addresses
#define ACCEL_ADDRESS ((int) 0x53) // 0x53 = 0xA6 / 2
7 #define MAGN_ADDRESS  ((int) 0x1E) // 0x1E = 0x3C / 2
#define GYRO_ADDRESS  ((int) 0x68) // 0x68 = 0xD0 / 2

10 // Arduino backward compatibility macros
#if ARDUINO >= 100
  #define WIRE_SEND(b) Wire.write((byte) b)
13   #define WIRE_RECEIVE() Wire.read()
#else
  #define WIRE_SEND(b) Wire.send(b)
16   #define WIRE_RECEIVE() Wire.receive()
#endif


19
void I2C_Init()
{
22   Wire.begin();
}

25 void Accel_Init()
{
  Wire.beginTransmission(ACCEL_ADDRESS);
28   WIRE_SEND(0x2D);  // Power register
  WIRE_SEND(0x08);  // Measurement mode
  Wire.endTransmission();
31   delay(5);
  Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x31);  // Data format register
34   WIRE_SEND(0x08);  // Set to full resolution
  Wire.endTransmission();
  delay(5);
37
  // Because our main loop runs at 50Hz we adjust the output data
       rate to 50Hz (25Hz bandwidth)
  Wire.beginTransmission(ACCEL_ADDRESS);
40   WIRE_SEND(0x2C);  // Rate
  WIRE_SEND(0x09);  // Set to 50Hz, normal operation
  Wire.endTransmission();
43   delay(5);
}

46 // Reads x, y and z accelerometer registers
void Read_Accel()
{
49   int i = 0;
  byte buff[6];

52   Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x32);  // Send address to read from
  Wire.endTransmission();
```

```
55
  Wire.beginTransmission(ACCEL_ADDRESS);
  Wire.requestFrom(ACCEL_ADDRESS, 6);  // Request 6 bytes
58  while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
61    i++;
  }
  Wire.endTransmission();
64
  if (i == 6)  // All bytes received?
  {
67    // No multiply by -1 for coordinate system transformation
        here, because of double negation:
    // We want the gravity vector, which is negated acceleration
        vector.
    accel[0] = (((int) buff[3]) << 8) | buff[2];  // X axis (
        internal sensor y axis)
70    accel[1] = (((int) buff[1]) << 8) | buff[0];  // Y axis (
        internal sensor x axis)
    accel[2] = (((int) buff[5]) << 8) | buff[4];  // Z axis (
        internal sensor z axis)
  }
73  else
  {
    num_accel_errors++;
76    if (output_errors) Serial.println("!ERR: reading
        accelerometer");
  }
}
79
void Magn_Init()
{
82  Wire.beginTransmission(MAGN_ADDRESS);
  WIRE_SEND(0x02);
  WIRE_SEND(0x00);  // Set continuous mode (default 10Hz)
85  Wire.endTransmission();
  delay(5);

88  Wire.beginTransmission(MAGN_ADDRESS);
  WIRE_SEND(0x00);
  WIRE_SEND(0b00011000);  // Set 50Hz
91  Wire.endTransmission();
  delay(5);
}
94
void Read_Magn()
{
97  int i = 0;
  byte buff[6];

100  Wire.beginTransmission(MAGN_ADDRESS);
```

```
  WIRE_SEND(0x03);  // Send address to read from
  Wire.endTransmission();
103
  Wire.beginTransmission(MAGN_ADDRESS);
  Wire.requestFrom(MAGN_ADDRESS, 6);  // Request 6 bytes
106 while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
109 i++;
  }
  Wire.endTransmission();
112
  if (i == 6)  // All bytes received?
  {
115 // 9DOF Razor IMU SEN-10125 using HMC5843 magnetometer
#if HW__VERSION_CODE == 10125
    // MSB byte first, then LSB; X, Y, Z
118 magnetom[0] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // X
        axis (internal sensor -y axis)
    magnetom[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);  // Y
        axis (internal sensor -x axis)
    magnetom[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Z
        axis (internal sensor -z axis)
121 // 9DOF Razor IMU SEN-10736 using HMC5883L magnetometer
#elif HW__VERSION_CODE == 10736
    // MSB byte first, then LSB; Y and Z reversed: X, Z, Y
124 magnetom[0] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // X
        axis (internal sensor -y axis)
    magnetom[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);  // Y
        axis (internal sensor -x axis)
    magnetom[2] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Z
        axis (internal sensor -z axis)
127 // 9DOF Sensor Stick SEN-10183 and SEN-10321 using HMC5843
    magnetometer
#elif (HW__VERSION_CODE == 10183) || (HW__VERSION_CODE == 10321)
    // MSB byte first, then LSB; X, Y, Z
130 magnetom[0] = (((int) buff[0]) << 8) | buff[1];         // X
        axis (internal sensor x axis)
    magnetom[1] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Y
        axis (internal sensor -y axis)
    magnetom[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Z
        axis (internal sensor -z axis)
133 // 9DOF Sensor Stick SEN-10724 using HMC5883L magnetometer
#elif HW__VERSION_CODE == 10724
    // MSB byte first, then LSB; Y and Z reversed: X, Z, Y
136 magnetom[0] = (((int) buff[0]) << 8) | buff[1];         // X
        axis (internal sensor x axis)
    magnetom[1] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Y
        axis (internal sensor -y axis)
    magnetom[2] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Z
        axis (internal sensor -z axis)
139 #endif
```

```
  }
  else
  {
    num_magn_errors++;
    if (output_errors) Serial.println("!ERR: reading magnetometer
        ");
  }
}

void Gyro_Init()
{
  // Power up reset defaults
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x3E);
  WIRE_SEND(0x80);
  Wire.endTransmission();
  delay(5);

  // Select full-scale range of the gyro sensors
  // Set LP filter bandwidth to 42Hz
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x16);
  WIRE_SEND(0x1B);  // DLPF_CFG = 3, FS_SEL = 3
  Wire.endTransmission();
  delay(5);

  // Set sample rato to 50Hz
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x15);
  WIRE_SEND(0x0A);  //  SMPLRT_DIV = 10 (50Hz)
  Wire.endTransmission();
  delay(5);

  // Set clock to PLL with z gyro reference
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x3E);
  WIRE_SEND(0x00);
  Wire.endTransmission();
  delay(5);
}

// Reads x, y and z gyroscope registers
void Read_Gyro()
{
  int i = 0;
  byte buff[6];

  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x1D);  // Sends address to read from
  Wire.endTransmission();

```

```
  Wire.beginTransmission(GYRO_ADDRESS);
  Wire.requestFrom(GYRO_ADDRESS, 6);  // Request 6 bytes
  while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
    i++;
  }
  Wire.endTransmission();

  if (i == 6)  // All bytes received?
  {
    gyro[0] = -1 * ((((int) buff[2]) << 8) | buff[3]);    // X
        axis (internal sensor -y axis)
    gyro[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);    // Y
        axis (internal sensor -x axis)
    gyro[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);    // Z
        axis (internal sensor -z axis)
  }
  else
  {
    num_gyro_errors++;
    if (output_errors) Serial.println("!ERR: reading gyroscope");
  }
}
```

A.3 FILE C

A.3.1 *S-Function DAC*

```
#define S_FUNCTION_NAME  sfun_MultiQ3_DAC
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "WinIo.h"

/* MultiQ3 board defs */
#define NUM_INPUTS           6
#define NUM_OUTPUTS          0
#define NPARAMS              0
#define SAMPLE_TIME_0        INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES      0
#define DISC_STATES_IC       [0]
#define NUM_CONT_STATES      0
#define CONT_STATES_IC       [0]
#define INPUT_0_WIDTH       1
#define INPUT_0_FEEDTHROUGH 1
#define INPUT_1_WIDTH       1
#define INPUT_1_FEEDTHROUGH 1
```

```
#define INPUT_2_WIDTH       1
#define INPUT_2_FEEDTHROUGH 1
#define INPUT_3_WIDTH       1
#define INPUT_3_FEEDTHROUGH 1
#define INPUT_4_WIDTH       1
#define INPUT_4_FEEDTHROUGH 1
#define INPUT_5_WIDTH       1
#define INPUT_5_FEEDTHROUGH 1

/* Definition of Quanser Register Adresses for DA conversion */
#define BASE_PORT       0x330
#define DIGIN_PORT      BASE_PORT + 0x00
#define DIGOUT_PORT     BASE_PORT + 0x00
#define DA_DATA         BASE_PORT + 0x02
#define AD_DATA         BASE_PORT + 0x04
#define STATUS_REG      BASE_PORT + 0x06
#define CONTROL_REG     BASE_PORT + 0x06
#define CLK_REG         BASE_PORT + 0x08
#define ENC_REG1        BASE_PORT + 0x0c
#define ENC_REG2        BASE_PORT + 0x0e

/* Definition of 16 bit CONTROL REGISTER values */
/* (S/H) = disable sample and hold on the A/D*/
#define AD_SH               0x200
/*LD0 and LD1 on the CONTROL REGISTER to be latched before DAC*/
#define LATCH_ON            0x1800
/* (CLK) = choose clock frequency of the A/D */
#define AD_CLOCK_4M         0x400
/* bit wise OR, so we keep both S/H and CLK high all the time */
#define CONTROL_MUST        (AD_SH | AD_CLOCK_4M)

static int open_dll = 0;

/*====================*
 * S-function methods *
 *====================*/

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);/* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected */
        /* != number of actual parameters */
        return;
    }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);


    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
    /*Input Port 0*/
```

```
    ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 0, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1);
    /*Input Port 1*/
    ssSetInputPortWidth(S, 1, INPUT_1_WIDTH);
    ssSetInputPortDataType(S, 1, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 1, INPUT_1_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 1, 1);
    /*Input Port 2*/
    ssSetInputPortWidth(S, 2, INPUT_2_WIDTH);
    ssSetInputPortDataType(S, 2, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 2, INPUT_2_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 2, 1);
    /*Input Port 3*/
    ssSetInputPortWidth(S, 3, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 3, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 3, INPUT_3_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 3, 1);
    /*Input Port 4*/
    ssSetInputPortWidth(S, 4, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 4, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 4, INPUT_4_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 4, 1);
    /*Input Port 5*/
    ssSetInputPortWidth(S, 5, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 5, SS_INT16);
    ssSetInputPortDirectFeedThrough(S, 5, INPUT_5_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 5, 1);/

    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);

}

#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S)
  {
```

```
  }
#endif

#define MDL_START
#if defined(MDL_START)
  static void mdlStart(SimStruct *S)
  {
  }
#endif

static void mdlOutputs(SimStruct *S, int_T tid)
{
    const int16_T   *ivalue0  = (const int16_T*)
        ssGetInputPortSignal(S,0);
    const int16_T   *ivalue1  = (const int16_T*)
        ssGetInputPortSignal(S,1);
    const int16_T   *ivalue2  = (const int16_T*)
        ssGetInputPortSignal(S,2);
    const int16_T   *ivalue3  = (const int16_T*)
        ssGetInputPortSignal(S,3);
    const int16_T   *ivalue4  = (const int16_T*)
        ssGetInputPortSignal(S,4);
    const int16_T   *ivalue5  = (const int16_T*)
        ssGetInputPortSignal(S,5);

    if ( open_dll == 0 )
    {
        InitializeWinIo();
        open_dll = 1;
    }
    SetPortVal(CONTROL_REG, LATCH_ON | 0 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue0, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
    SetPortVal(CONTROL_REG, LATCH_ON | 1 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue1, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
    SetPortVal(CONTROL_REG, LATCH_ON | 2 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue2, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
    SetPortVal(CONTROL_REG, LATCH_ON | 3 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue3, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
    SetPortVal(CONTROL_REG, LATCH_ON | 4 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue4, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
    SetPortVal(CONTROL_REG, LATCH_ON | 5 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue5, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
```

```
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif

#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif

static void mdlTerminate(SimStruct *S)
{
    if ( open_dll == 1)
    {
        SetPortVal(CONTROL_REG, LATCH_ON | 0 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
        SetPortVal(CONTROL_REG, LATCH_ON | 1 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
        SetPortVal(CONTROL_REG, LATCH_ON | 2 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
        SetPortVal(CONTROL_REG, LATCH_ON | 3 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
        SetPortVal(CONTROL_REG, LATCH_ON | 4 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
        SetPortVal(CONTROL_REG, LATCH_ON | 5 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        ShutdownWinIo();
        open_dll = 0;
    }
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

A.3.2 *S-Function ADC*

```
#define S_FUNCTION_NAME  sfun_MultiQ3_ADC
#define S_FUNCTION_LEVEL 2
```

```
#include "simstruc.h"
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "WinIo.h"

/* MultiQ3 board defs */
#define NUM_INPUTS           0
#define NUM_OUTPUTS          6
#define NPARAMS              0
#define SAMPLE_TIME_0        INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES      0
#define DISC_STATES_IC       [0]
#define NUM_CONT_STATES      0
#define CONT_STATES_IC       [0]
#define INPUT_WIDTH     0
#define OUTPUT_0_WIDTH       1
#define OUTPUT_1_WIDTH       1
#define OUTPUT_2_WIDTH       1
#define OUTPUT_3_WIDTH       1
#define OUTPUT_4_WIDTH       1
#define OUTPUT_5_WIDTH       1

/* Definition of Quanser Register Adresses for ADC */
#define BASE_PORT       0x330
#define DIGIN_PORT      BASE_PORT + 0x00
#define DIGOUT_PORT     BASE_PORT + 0x00
#define DA_DATA         BASE_PORT + 0x02
#define AD_DATA         BASE_PORT + 0x04
#define STATUS_REG      BASE_PORT + 0x06
#define CONTROL_REG     BASE_PORT + 0x06
#define CLK_REG         BASE_PORT + 0x08
#define ENC_REG1        BASE_PORT + 0x0c
#define ENC_REG2        BASE_PORT + 0x0e

/* Definition of 16 bit CONTROL REGISTER values */
#define AD_SH           0x200 /* active low */
#define AD_AUTOCAL      0x100 /* active high */
#define AD_AUTOZ        0x80 /* active high */
#define AD_MUX_EN       0x40 /* active high */
#define AD_CLOCK_4M     0x400 /* high = 4 MHz */

static int open_dll = 0;

/* IMPORTANT */
/* sample and hold disabled to prevent exrtaneous sampling */
/* and fix the clock speed to 4 MHz */
#define CONTROL_MUST (AD_SH | AD_CLOCK_4M)

/*====================*
 * S-function methods *
 *====================*/
```

```
56
static void mdlInitializeSizes(SimStruct *S)
{
59
    ssSetNumSFcnParams(S, 0);/* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
62      /* Return if number of expected */
        /* != number of actual parameters */
        return;
65  }

    ssSetNumContStates(S, NUM_CONT_STATES);
68  ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
71
    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;
    /*Output Port 0*/
74  ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    /*Output Port 1*/
77  ssSetOutputPortWidth(S, 1, OUTPUT_1_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    /*Output Port 2*/
80  ssSetOutputPortWidth(S, 2, OUTPUT_2_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    /*Output Port 3*/
83  ssSetOutputPortWidth(S, 3, OUTPUT_3_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    /*Output Port 4*/
86  ssSetOutputPortWidth(S, 4, OUTPUT_4_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
    /*Output Port 5*/
89  ssSetOutputPortWidth(S, 5, OUTPUT_5_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);

92  ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
95  ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
98  ssSetOptions(S, 0);
}

101 static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
104 ssSetOffsetTime(S, 0, 0.0);
}

107 #define MDL_INITIALIZE_CONDITIONS
```

```
#if defined(MDL_INITIALIZE_CONDITIONS)

static void mdlInitializeConditions(SimStruct *S)
  {
  }
#endif /* MDL_INITIALIZE_CONDITIONS */

#define MDL_START
#if defined(MDL_START)
    static void mdlStart(SimStruct *S)
    {
    }
#endif /*  MDL_START */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  *y[6];
    int ch, toolong, maxcnt;
    DWORD lbw, lb, hb;
    PDWORD lbw_p, lb_p, hb_p;
    int16_T adw;
    unsigned int control_word;
    for(ch=0; ch<6; ch++)
    {
        y[ch] = ssGetOutputPortRealSignal(S,ch);
    }
    if ( open_dll == 0 )
    {
        InitializeWinIo();
        open_dll = 1;
    }
    lbw_p = &lbw;
    lb_p = &lb;
    hb_p = &hb;
    for(ch=0; ch<6; ch++)
    {
        control_word = CONTROL_MUST | AD_MUX_EN | (ch<<3);
        SetPortVal(CONTROL_REG, control_word, 2);
        toolong=0; maxcnt=301;
        do{
            GetPortVal(STATUS_REG, lbw_p, 2);
            toolong++;
        }while( ((lbw & 0x0008) == 0x0000)  && (toolong < maxcnt)
             );
        if(toolong >= maxcnt) Beep(350,300);
        SetPortVal(AD_DATA, 0, 1);
        do{
            GetPortVal(STATUS_REG, lbw_p, 2);
        }while( (lbw & 0x0010) == 0x0000 );
        GetPortVal(AD_DATA, hb_p, 1);
        GetPortVal(AD_DATA, lb_p, 1);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);
```

```
        adw = ( ((int16_T)hb) & 0x00ff ) << 8;
        adw = adw | ( ((int16_T)lb) & 0x00ff);
        *y[ch] = ((real_T)adw) * 5/4095;
    }
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */

#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */

static void mdlTerminate(SimStruct *S)
{
    if ( open_dll == 1)
    {
        ShutdownWinIo();
        open_dll = 0;
    }
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

A.3.3 *S-Function Parser*

```
#define S_FUNCTION_NAME  sfun_parser_bin
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include <malloc.h>
#include <string.h>

/* S-function defs */
#define NUM_INPUTS          1
#define NUM_OUTPUTS         3
#define NUM_PARAMS          0
#define SAMPLE_TIME_0       INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES     1
#define DISC_STATES_IC      [0]
#define NUM_CONT_STATES     0
```

```
#define CONT_STATES_IC      [0]
#define INPUT_0_WIDTH           1
#define INPUT_0_FEEDTHROUGH     1
#define OUTPUT_0_WIDTH      3
#define OUTPUT_1_WIDTH      1
#define OUTPUT_2_WIDTH      1

/*====================*
 * S-function methods *
 *====================*/

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUM_PARAMS);  /* Number of expected
        parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual
            parameters */
        return;
    }
    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;

    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;

    /*Input Port 0*/
    ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 0, SS_UINT8);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1); /*direct input
        signal access*/

    /*Output Port 0*/
    ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);

    /*Output Port 1*/
    ssSetOutputPortWidth(S, 1, OUTPUT_1_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);

    /*Output Port 2-debug*/
    ssSetOutputPortWidth(S, 2, OUTPUT_2_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 3);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
```

```
    ssSetOptions(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
#if defined(MDL_INITIALIZE_CONDITIONS)

  static void mdlInitializeConditions(SimStruct *S)
  {
      real_T   *x = ssGetRealDiscStates(S);

      x[0] = 0;
  }
#endif /* MDL_INITIALIZE_CONDITIONS */

#define MDL_START
#if defined(MDL_START)

  static void mdlStart(SimStruct *S)
  {
    void **PWork = ssGetPWork(S);
    //uint8_T bcnt = *(uint8_T *)ssGetPWorkValue(S,0);

    PWork[0] = (void *) calloc(1, sizeof(uint8_T));
    PWork[1] = (void *) calloc(3, sizeof(real_T));
    PWork[2] = (void *) calloc(12, sizeof(uint8_T));

    //bcnt = 0;
  }
#endif /*  MDL_START */

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T   *y = ssGetOutputPortSignal(S,0);
    real_T   *data_ready = ssGetOutputPortSignal(S,1);
    real_T   *output_debug = ssGetOutputPortSignal(S,2);
    real_T   *x = ssGetRealDiscStates(S);

    uint8_T *bcnt = (uint8_T *)ssGetPWorkValue(S,0);
    real_T  *yprev = (real_T *)ssGetPWorkValue(S,1);
    uint8_T *buff = (uint8_T *)ssGetPWorkValue(S,2);

    if(x[0] == 4)
    {
        if(*bcnt == 11)
        {
```

```
            y[0] = (real_T) (*(real32_T *)buff);
            y[1] = (real_T) (*(real32_T *)(buff+4));
            y[2] = (real_T) (*(real32_T *)(buff+8));
            memcpy(yprev, y,  3*sizeof(real_T));
            *data_ready = 1;
        }
    }
    else
    {
        memcpy(y, yprev, 3*sizeof(real_T));
        *data_ready = 0;
    }
    //*output_debug = *x;
    *output_debug = (real_T) *bcnt;
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)

  static void mdlUpdate(SimStruct *S, int_T tid)
  {
      real_T   *x = ssGetRealDiscStates(S);
      const uint8_T *u = (const uint8_T*) ssGetInputPortSignal(S
          ,0);

      uint8_T * bcnt = (uint8_T *)ssGetPWorkValue(S,0);
      uint8_T *buff = (uint8_T *)ssGetPWorkValue(S,2);

      switch((uint8_T)x[0])
      {
        case 0:
            if(*u == 0xFF)
                x[0] = 1;
            else
                x[0] = 0;
            break;

        case 1:
            if(*u == 0x00)
                x[0] = 2;
            else
                x[0] = 0;
            break;

        case 2:
            if(*u == 0xFF)
                x[0] = 3;
            else
                x[0] = 0;
            break;

         case 3:
```

```
             if(*u == 0x00)
168             x[0] = 4;
            else
                x[0] = 0;
171         break;

         case 4:
174          buff[*bcnt] = *u;
             if(*bcnt < 11)
             {
177             *bcnt= (*bcnt)+1;
                x[0] = 4;
             }
180          else
             {
                 *bcnt = 0;
183              x[0] = 0;
             }
            break;
186     }
  }
#endif /* MDL_UPDATE */
189
#undef MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
192
  static void mdlDerivatives(SimStruct *S)
  {
195   }
#endif /* MDL_DERIVATIVES */

198 static void mdlTerminate(SimStruct *S)
{
    if (ssGetPWork(S) != NULL)
201 {
        free(ssGetPWorkValue(S,0));
        ssSetPWorkValue(S,0,NULL);
204
        free(ssGetPWorkValue(S,1));
        ssSetPWorkValue(S,1,NULL);
207
        free(ssGetPWorkValue(S,2));
        ssSetPWorkValue(S,2,NULL);
210 }
}

213 #ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
216 #include "cg_sfun.h"
#endif
```

B

SCHEMI SCHEDE PCB

B.1 SCHEDA DI CONDIZIONAMENTO DEI SEGNALI

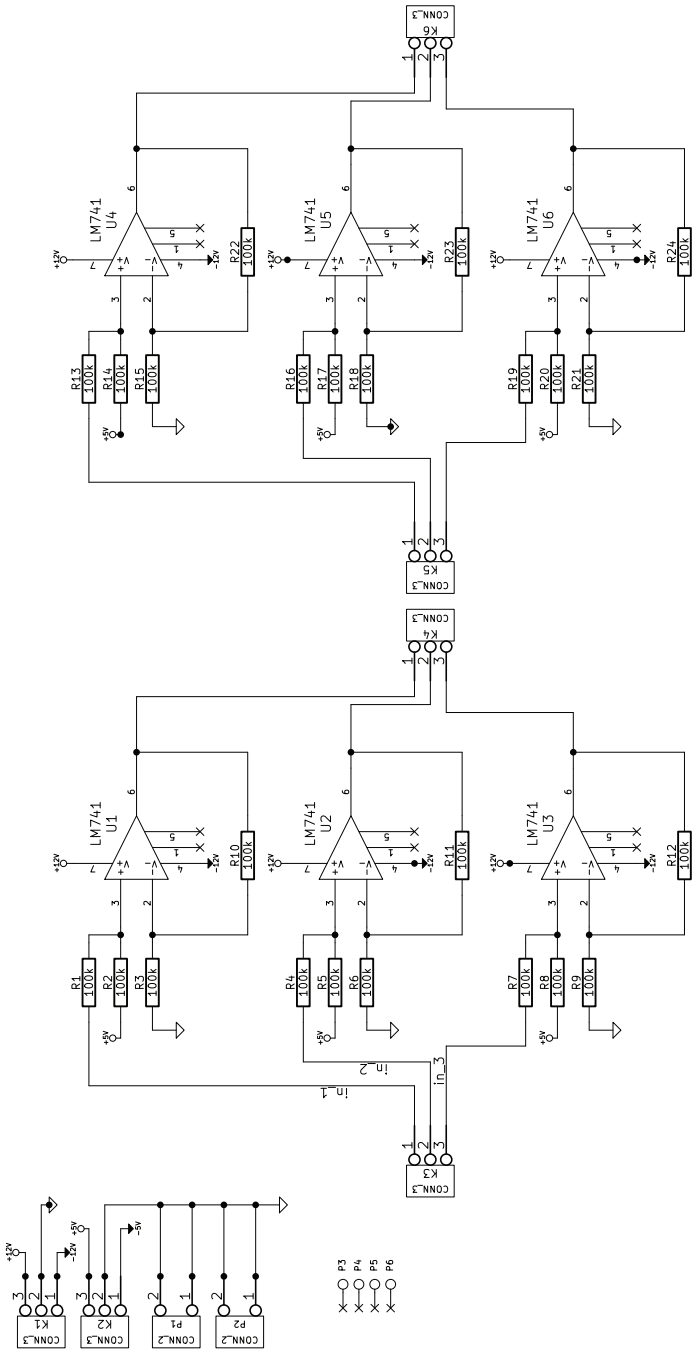


Figura 61: Schema elettrico della scheda di condizionamento dei segnali

B.2 SCHEDA PER LA GESTIONE DELLA TENSIONE

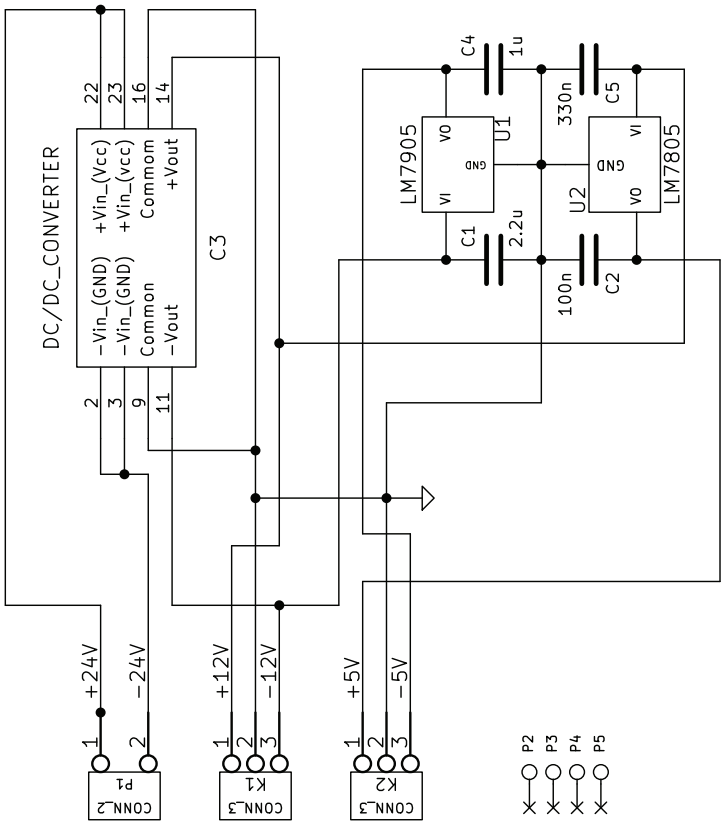


Figura 62: Schema elettrico della scheda di gestione delle tensioni

B.3 SCHEDA DI INTERFACCIA PER IL SENSORE IMU

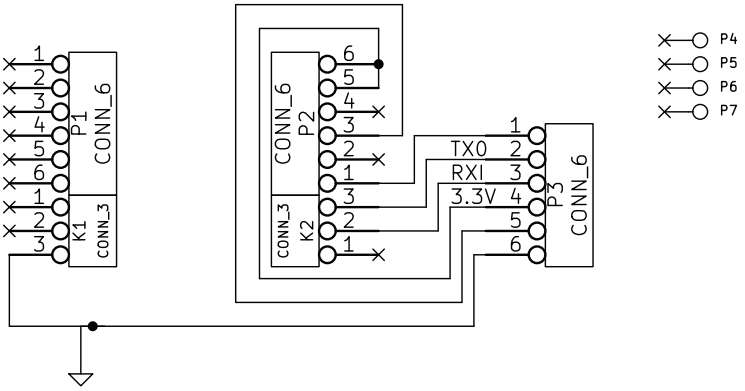


Figura 63: Schema elettrico di interfaccia tra la morsettiera USB e il sensore IMU

BIBLIOGRAFIA

[1] Jyh-Yeong Chang, Wen-Feng Hu, Mu-Huo Cheng, and Bo-Sen Chang. Digital image translational and rotational motion stabilization using optical flow technique. *Consumer Electronics, IEEE Transactions on*, 48(1):108 –115, feb 2002. ISSN 0098-3063. doi: 10.1109/TCE.2002.1010098.

[2] COAST GUARD RESEARCH AND DEVELOPMENT CENTER GROTON CT. Side-By-Side Seakeeping Tests of the USCGC PADRE (WPB 1328) and the USCGC SHEARWATER (WSES 3). Report, 1996.

[3] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2003.

[4] J.M. Hilkert. Inertially stabilized platform technology concepts and principles. *Control Systems, IEEE*, 28(1):26 –46, feb. 2008. ISSN 1066-033X. doi: 10.1109/MCS.2007.910256.

[5] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(3):334 –352, aug. 2004. ISSN 1094-6977. doi: 10.1109/TSMCC.2004.829274.

[6] M.K. Masten. Inertially stabilized platforms for optical imaging systems. *Control Systems, IEEE*, 28(1):47 –64, feb. 2008. ISSN 1066-033X. doi: 10.1109/MCS.2007.910201.

[7] Andrea Parisotto. Soppressione delle vibrazioni nei brandeggi per telecamere: minimizzazione del ripple di coppia degli attuatori, 2010.

[8] Daniele Pellizzer. Soppressione delle vibrazioni nei brandeggi per telecamere: analisi prestazionale mediante elaborazione delle immagini, 2010.

[9] Marco Peruzzo. Soppressione delle vibrazioni nei brandeggi per telecamere: tecniche di smorzamento attivo, 2010.

[10] Davide Pilastro. Stabilizzazione dell'immagine in brandeggi per videosorveglianza: applicazione di tecniche basate sull'uso di sensori inerziali, 2012.

[11] Ing. Pini. *Manuale di servizio: MicroStar B Brushless & dc Motor Drive SMB*. URL http://www.speedermotion.com/.

[12] Mattia Schiesaro. Analisi di prestazione di un brandeggio per applicazioni di videosorveglianza, 2010.

[13] Co. Inc. Sensoray. *Instruction Manual of the PCI Multifunction I/O Board Sensoray Model 626*. URL http://www.sensoray.com/.

[14] M.H. Shakoor and M. Moattari. A new fast method for digital image stabilization, aug. 2010. ISSN 2154-7491.

[15] M.J. Smith, A. Boxerbaum, G.L. Peterson, and R.D. Quinn. Electronic image stabilization using optical flow with inertial fusion. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1146 –1153, oct. 2010. doi: 10.1109/IROS.2010.5651113.

[16] Thompson, J. L and R.A.N. Research Laboratory. Ship motion measurements made on an Attack Class patrol boat (HMAS Bombard). Report, 1980.

[17] Prof. Francesco Tortella. Dispense del corso di teoria e tecniche di interpretazione delle immagini. In *La trasformata di Hough*. .

[18] Prof. Francesco Tortella. Dispense del corso di teoria e tecniche di interpretazione delle immagini. In *Immagini binarie. Algoritmi di binarizzazione*. .

[19] Videotec S.p.A. *ULISSE COMPACT TECHNICAL DATA SHEET*. URL http://www.videotec.com/.

[20] Enrico Zambon. Compensazione delle vibrazioni in brandeggi per videocamere mediante misure da acceleromentro e giroscopio, 2010.

[21] Prof. Mauro Zigliotto. Dispense del corso di macchine ed azionamenti elettrici. In *part 10.0-Azionamenti con motore a passo*.