Alma Mater Studiorum - Università di Bologna

Scuola di Scienze

Corso di Laurea Magistrale in Informatica

Acquisizione, ricostruzione ed analisi di dati rotazionali da dispositivo mobile

Tesi di Laurea in Fisica dei Sistemi Complessi

Relatore :

Chiar.mo Prof.

Sandro Rambaldi

Presentata da :

Andrea Ferracin

Sessione III

Anno Accademico 2012/2013

Indice

Capitolo 1

Introduzione

In questa dissertazione si affronta il problema della ricostruzione delle rotazioni compiute da un veicolo in movimento durante un determinato lasso di tempo.
La situazione presentata è già risolta approssimativamente da strumenti di uso comune dotati di GPS ed accelerometro, i quali possono determinare il percorso fatto da un mezzo di trasporto utilizzando la sua posizione geolocalizzata.
Il nostro scopo è più complicato perché non dobbiamo ricostruire un percorso tra un punto di origine ed una destinazione, ma vogliamo ottenere in tempo reale la sequenza di rotazioni di un veicolo attorno ai suoi assi principali, per poi poterli analizzare in un secondo momento. Il vincolo del tempo reale richiede degli algoritmi molto efficienti che siano in grado di aggiornare in modo continuo la rotazione attuale.

Un esempio di applicazione

Per capire l'utilità di risolvere questo tipo di problema, basti pensare alla polizia stradale, che sarebbe molto avvantaggiata nel ricostruire la dinamica di un incidente con diversi mezzi coinvolti, oppure alle compagnie assicurative, che potrebbero variare il premio di assicurazione a seconda dello stile di guida del conducente.
Di seguito verrà presentata la dinamica di un incidente tra due automobili dotate di dispositivi con GPS e accelerometro.
Nella figura 1.1a viene mostrata la traiettoria di due automobili ricostruita con segnali GPS, mentre nelle due immagini a lato sono evidenziati i possibili punti di impatto interpretabili dai valori forniti dagli accelerometri dei due veicoli coinvolti. Dall'immagine si può capire la

ricostruzione dell'incidente fatta per mezzo delle informazioni raccolte.

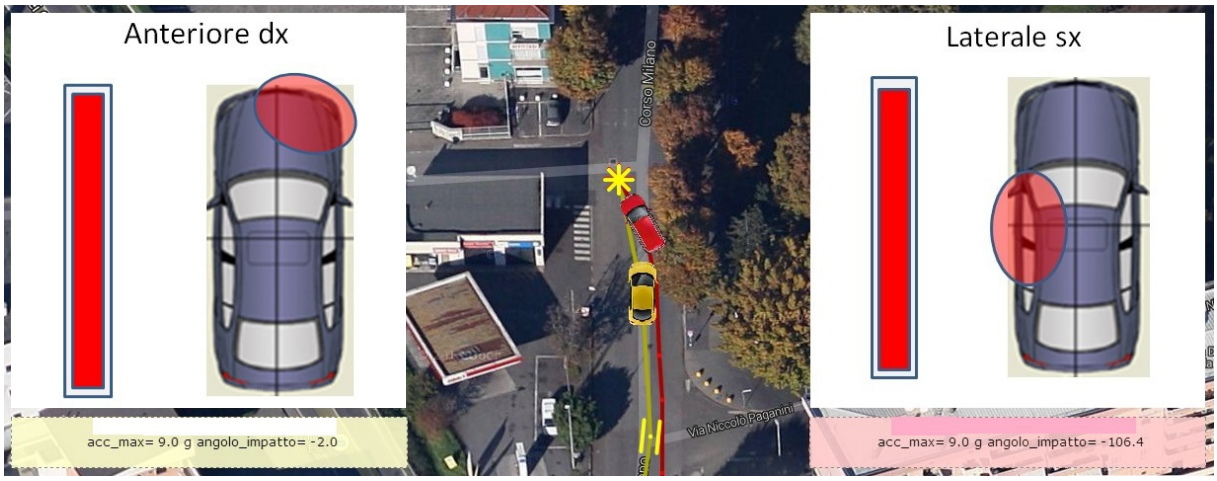

Figura 1.1a

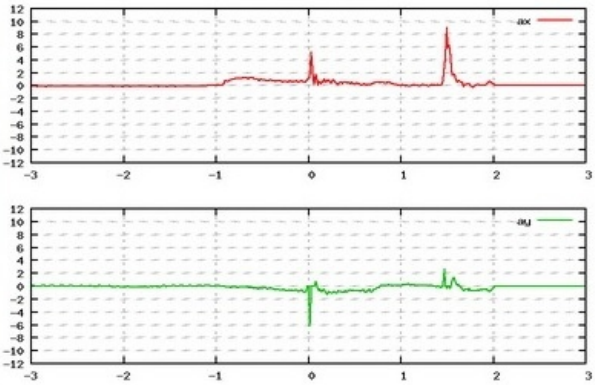

Figura 1.1b

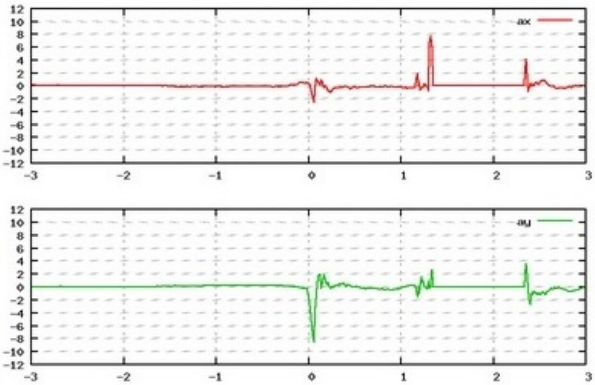

Figura 1.1c

Dall'analisi delle accelerazioni, si può capire che l'auto gialla tampona l'auto rossa con la parte destra del paraurti, con più precisione con un angolo di 53 gradi rispetto all'asse longitudinale del veicolo, mentre l'auto rossa subisce l'urto nella sua fiancata sinistra, più precisamente con un angolo di 102 gradi sempre rispetto all'asse longitudinale.

La dinamica dell'incidente appena descritta è ricostruita nell'immagine 1.1a.

Nelle figure 1.1b e 1.1c vengono mostrati due grafici rappresentanti le variazioni in secondi delle accelerazioni rispettivamente dell'auto gialla e rossa misurate in g (accelerazione di gravità); le accelerazioni riguardano l'asse longitudinale (grafico di colore rosso) e trasversale (colore verde) alla carrozzeria.

Si può vedere da entrambi i grafici che si ottiene un primo impatto all'istante 0 attraverso l'osservazione di un picco di accelerazione; da questi valori si può ricostruire la dinamica del primo impatto tra le due auto, ma non quello che succede dopo il primo urto.
Dopo l'impatto, con i dati a nostra disposizione, non è possibile ricostruire le posizioni delle auto per determinare se queste portano ad una ulteriore evoluzione dell'incidente.
Da quanto si può vedere dai grafici 1.1b e 1.1c, si hanno ulteriori variazioni di accelerazione in entrambi gli assi per le due automobili, solo che è impossibile da questi dati poter ricostruire la dinamica.
Anche il GPS non è di aiuto poiché presenta una precisione troppo bassa per determinare l'evolversi dell'incidente, infatti i GPS migliori montati sulle auto hanno una precisione di 3-4 metri ed una frequenza di 1 Hz (un dato al secondo), mentre un urto si sviluppa in pochi decimi di secondo.
Per risolvere questo problema si potrebbe utilizzare un giroscopio interno ai veicoli per poter comprendere le rotazioni durante gli spostamenti fatti, ed in particolare durante gli incidenti.
In questa maniera, complementando le accelerazioni con le rotazioni, si potrà ricostruire la dinamica dell'incidente.

Scopo dell'elaborato

Abbiamo appena visto come sarebbe vantaggioso l'uso di un giroscopio nella ricostruzione degli incidenti e quindi la dotazione nelle auto di dispositivi contenenti componenti atti a recuperare oltre alle accelerazioni, anche i valori di rotazione.
Allo scopo di affrontare questo argomento si è astratto il problema in modo da concentrarsi solo sulle rotazioni di un dispositivo mobile.
Oggigiorno esistono numerosi componenti hardware dotati di accelerometri e giroscopi che possono rilevare gli spostamenti compiuti da un veicolo; inoltre possono aiutare nel computare gli stessi dati di movimento per ottenere entità matematiche utili a rappresentare le rotazioni eseguite.
Scopo di questa tesi è quello di creare un'architettura in grado di acquisire, ricostruire ed analizzare dei dati rotazionali acquisiti da un dispositivo mobile Arduino al fine di confrontare la verosimiglianza dei risultati ottenuti con la posizione effettiva assunta dal

dispositivo utilizzato, ed in particolare di stimare quantitativamente gli errori che ci si può aspettare.

Nel capitolo 2 verrà presentata la piattaforma Arduino con tutte le potenzialità che essa può offrire sia per quanto riguarda i componenti hardware che la parte software.

Nel capitolo 3 verranno esposti i diversi sistemi di riferimento utilizzati elencando le somiglianze e differenze, per poi spiegar il sistema adottato da Arduino.

Nel capitolo 4 verrà esposta l'entità matematica utilizzata per rappresentare la rotazione ottenuta, ovvero il quaternione; di conseguenza saranno mostrate le formule classiche utilizzate per la gestione dei quaternioni in questo specifico ambito.

Nel capitolo 5 verrà mostrata l'architettura software creata su misura per gestire la piattaforma Arduino e controllare i quaternioni in modo da ricostruire graficamente le posizioni rotazionali ottenute.

Nel capitolo 6 saranno mostrati i risultati ricavati da algoritmi basati sulle formule classiche dei quaternioni in contrapposizione ad hardware specializzato per il rilevamento di rotazioni di cui è dotata la piattaforma Arduino considerata. In questa parte dell'elaborato vedremo che la precisione ottenibile è adeguata alla ricostruzione della cinematica in un incidente stradale e questo giustifica l'aumento dei costi dovuti ala presenza del giroscopio.

Infine nel capitolo 7 verranno indicati gli sviluppi futuri che le idee presentate possono avere e le migliorie da apportare.

Capitolo 2

Piattaforma Arduino

In questo capitolo verrà descritta dapprima la piattaforma Arduino utilizzata nel progetto presentato, poi si parlerà brevemente dell'hardware a disposizione (in particolare del componente per percepire i movimenti), del sistema di sviluppo e delle peculiarità del linguaggio utilizzato per programmare; infine si approfondirà sulle caratteristiche delle librerie specifiche utilizzate.

Cos'è Arduino

Arduino è una piattaforma elettronica open-source che si basa su un circuito stampato che integra un microcontrollore con pin connessi alle porte di I/O ed una interfaccia (USB o altro) che permette la comunicazione col computer.
A questo hardware viene affiancato un ambiente di sviluppo integrato (IDE) multipiattaforma (quindi per Linux, Apple e Windows). Questo software permette di scrivere con un linguaggio derivato dal C e C++ chiamato Wiring, liberamente scaricabile e modificabile.
Arduino è un componente hardware utilizzato per creare prototipi, per scopi didattici, hobbistici e professionali, e si è diffuso rapidamente per la sua semplicità e praticità.

Hardware

L'hardware Arduino viene presentato in diverse versioni, a seconda del tipo di attuatori e sensori montati sul dispositivo. I vari modelli sono differenziati anche dalla potenza di calcolo, dalla memoria o dalle dimensioni. Sono tutti accomunati dal fatto che supportano lo

stesso software, quindi, a meno di utilizzare componenti peculiari di uno specifico tipo di piattaforma, il programma creato può essere utilizzato su altro hardware Arduino.
Per implementare il comportamento interattivo, la piattaforma è fornita di funzionalità di input ed output (I/O), grazie alle quali riceve i segnali raccolti dai sensori esterni.
Il comportamento della scheda è gestito dal microcontroller che governa la piattaforma in base alle scelte determinate dal particolare programma in esecuzione sulla scheda. L'interazione con l'esterno avviene attraverso attuatori e canali di output.
In questo progetto specifico, viene utilizzato una circuiteria che percepisce il movimento del dispositivo per ottenere accelerazioni e rotazioni da esportare verso un pc esterno attraverso una porta seriale virtuale.

Componenti MEMS

MEMS, ovvero Micro Electo-Mechanical Systems, è un acronimo per indicare un sistema elettro-meccanico, nel nostro caso utilizzato per catturare i movimenti (rotazioni ed accelerazioni).
I microsistemi elettromeccanici non sono altro che un insieme di dispositivi di varia natura (meccanici, elettrici ed elettronici) integrati in forma altamente miniaturizzata su uno stesso substrato di silicio, e comprendono anche la tecnologia per ottenere sensori ed attuatori.
Si ha quindi un microsistema in grado di captare informazioni dall'ambiente traducendo le grandezze fisiche in impulsi elettrici. Le quantità osservate sono convertite in numeri che possono essere elaborati dalla stessa piattaforma e memorizzati per una fase di post-processing.
Di seguito verrà indicata la struttura del sensore che percepisce le rotazioni e le accelerazioni.

Sensore MEMS

In figura 2.1 è presente l'immagine della struttura del sensore MEMS addetto alla percezione delle accelerazioni in 2 dimensioni. Si presenta come una struttura a croce dotata di 4 molle ai lati (rettangoli colorati di verde), queste permettono alla struttura di scorrere e incastrarsi con le sbarre laterali fisse (di colore rosso e blu).

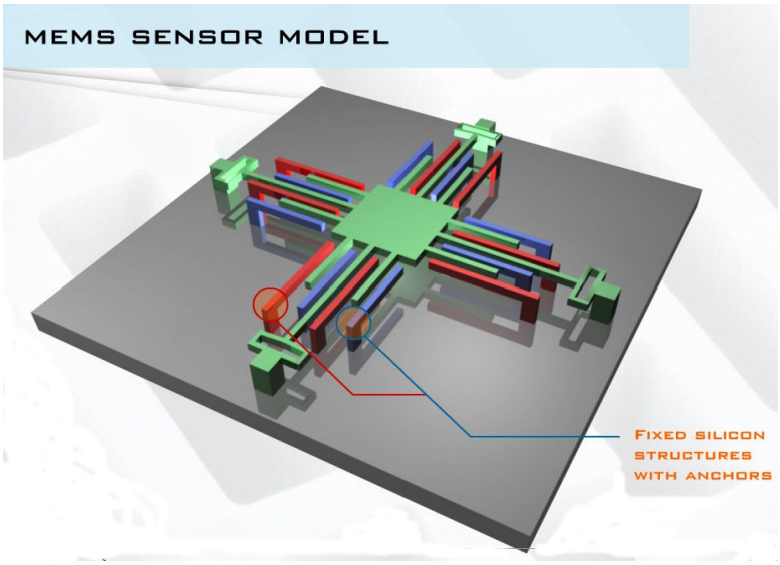

Figura 2.1

Tutto il meccanismo è fatto in silicone su cui sono poggiati sottili strati di materiale ed è in grado di funzionare come un condensatore elettrico a capacità variabile che traduce oscillazioni meccaniche in oscillazioni elettriche. In questa maniera attraverso i movimenti sulle due dimensioni della parte dotata di molle rispetto a quella fissa, si percepiscono diverse differenze di potenziale, che poi sono convertite con un ADC (analog digital converter) interno al MEMS in valori numerici.

Dalla forma del componente possiamo intuire che le differenze di potenziale saranno idealmente mappate su un sistema di assi cartesiano solidale al circuito (nello specifico solidale alla croce dotata di 4 molle ai lati), quindi si otterranno dati che rappresentano le accelerazioni di un sistema di assi cartesiano solidale al circuito. Un oscillatore indipendente risolve la terza dimensione.

Nella piattaforma Arduino utilizzata abbiamo montato un MEMS MotionTracking MPU6050 (mpu è un acronimo di motion processing unit), ovvero un componente a basso costo e ad alte prestazioni utilizzato anche in smartphone e tablet che includono un accelerometro ed un giroscopio.

Siccome le rotazioni e le accelerazioni si possono influenzare tra di loro, bisogna considerare entrambi per ottenere una ricostruzione del movimento il più fedele possibile. A questo scopo è stato inserito nell'hardware un processore accessorio, il Digital Motion Processor (DMP).

Processore DMP

Un sensore di Motion Tracking fornisce delle informazioni sui movimenti percepiti allo scopo di poter ricostruire gli spostamenti dell'apparecchio, questo fonde i valori di accelerazione e

rotazione per minimizzare gli errori che scaturiscono da ogni sensore.
Il DMP calcola i risultati in termini di quaternioni (strutture dati utilizzate per mantenere i movimenti del dispositivo, questi saranno presi in esame più avanti) dai quali si possono estrarre informazioni riguardanti le rotazioni. L'uso del DMP è facoltativo, ovvero bisogna attivarlo quando le accelerazioni in gioco sono superiori all'accelerazione di gravità, introducendo errori.

Considerazioni sui sensori MEMS

In questo paragrafo è stata mostrata la struttura di uno strumento elettro-meccanico di misurazione dei movimenti. Questo apparato, per quanto preciso, introduce degli errori di misurazione, infatti sperimentalmente un sensore completamente fermo andrà a percepire del rumore di fondo, ossia delle piccole accelerazioni che, con l'avanzare del tempo, si sommano alla velocità angolare indicando un movimento del sensore.
Una possibile soluzione a questo inconveniente è quello di impostare una funzione di reset direttamente sulla piattaforma Arduino attivato con un altro sensore per la GPS (global positioning system), acronimo per indicare il sistema di posizionamento globale.

Linguaggio Wiring

L'architettura Arduino viene utilizzata in combinazione con un IDE che permette di caricare il programma su piattaforma, nel nostro caso specifico attraverso la porta seriale. Con questo strumento è possibile scrivere dei programmi in un linguaggio imperativo (chiamato Wiring) simile al C e C++ come accennato nell'introduzione al capitolo. Questo linguaggio è stato pensato per essere più intuitivo possibile, in modo da permettere a chiunque disponga di rudimenti di programmazione a interagire con la piattaforma.
Una piattaforma Arduino è dotata di un quantitativo di RAM che varia da 1 Kbyte fino a 4 Kbyte, da qui si ha la necessità di creare programmi molto semplici.
Un programma in gergo viene chiamato **sketch,** ovvero schema, abbozzo.
Per semplificare la fase di programmazione e per rendere il codice prodotto portabile anche oltre la piattaforma stessa, è possibile importare librerie in C o C++, quindi vengono

supportati file con estensione c, cpp oppure h per gli header dei file. L'unica limitazione è che le librerie importate devono soddisfare la grammatica del compilatore di Arduino.
Per differenziare il codice per la piattaforma rispetto a quello tipico della programmazione in C, lo sketch ha una diversa estensione (ino o anche pde nelle precedenti versioni).
Lo sketch è considerato l'unità di codice che è caricata e che gira nella piattaforma ed è l'unico programma che governa il dispositivo considerato, quindi non esiste uno strato software sottostante da utilizzare come macchina astratta, ad esempio un sistema operativo specifico; infatti esiste solo un microcontrollore a livello hardware. Non vengono quindi supportati semafori oppure thread poiché non esiste un sistema operativo che possa gestire queste operazioni; inoltre non possono girare due programmi contemporaneamente sulla piattaforma, da ciò si può capire che il linguaggio è fortemente semplificato poiché non deve curare aspetti di concorrenza tra programmi.
Nel C il processo di compilazione trasforma le istruzioni scritte in “codice oggetto” da far utilizzare alla macchina sottostante, mentre con gli sketch la fase di compilazione si completa col caricamento del codice oggetto nella piattaforma, sostituendo eventualmente il codice che risiedeva prima.
Di seguito verrà presentata una breve descrizione del linguaggio Wiring con caratteristiche tipiche di Arduino.

Struttura del linguaggio

Nel C e C++ l'inizio del programma avviene per convenzione con la funzione main(), mentre lo sketch prevede due fasi principali:

- setup: una fase di inizializzazione che viene chiamata una volta sola quando il programma parte, serve per attivare i sensori, gli attuatori, le porte di comunicazione con il mondo esterno, le librerie e variabili da utilizzare nel corpo del programma.

- loop: questa fase segue quella di inizializzazione, ed indica che il programma cicla di continuo, di solito per percepire le informazioni dei sensori per poi analizzarle e agire di conseguenza. Nel nostro caso specifico in questa fase vengono letti ed interpretati i valori dati dal giroscopio ed accelerometro in modo ciclico.

Istruzioni e costrutti del linguaggio

Le istruzioni sono simili a quelle viste per C e C++ ed è previsto anche l'uso del preprocessore nello stesso modo.
Si hanno variabili, costanti e tipi di dato uguali al C, con aggiunta di costanti predefinite e accessibili a qualsiasi punto del programma:

- HIGH e LOW si usano quando si vuole accendere o spegnere un pin di Arduino, ovvero i piedini di collegamento dei circuiti stampati. In pratica indicano una quantità di +5 Volt o 0 Volt passante per i piedini.

- INPUT e OUTPUT si usano per impostare un determinato pin come ingresso oppure uscita, quindi se l'informazione entra o esce dal particolare pin.

Si possono scrivere funzioni come nel C e i controlli di flusso (il costrutto if-then, i cicli for e while,…) sono pressoché uguali.

Ingressi ed uscite dei pin

Siccome uno sketch gestisce direttamente l'hardware attraverso il microcontrollore, sono presenti delle funzionalità per controllare i vari pin che compongono i circuiti della piattaforma; questi si differenziano in pin di ingresso ed uscita, sia digitale (segnali 0-1 che indicano uno stato ben definito), sia analogico (restituisce valori che possono variare entro un dato intervallo).
Per gli ingressi digitali abbiamo le funzioni:

-pinMode(pin, mode): utilizzata in fase di setup, imposta se un dato pin è di input o di output.

-sigitalRead(pin): serve per leggere lo stato di un pin (restituisce HIGH e LOW visti prima).

-digitalWrite(pin, value): attiva o disattiva un pin.

Simili sono le funzioni analogiche analogRead e analogWrite, che rispettivamente leggono ed impostano la tensione passante per un pin con valori variabili entro un dato range.

Gestione dell'orologio interno

Nella piattaforma Arduino sono presenti specifiche funzioni per analizzare il tempo, fermare e regolare l'analisi del programma:

- millis(): restituisce il numero di millisecondi da quando è azionato il programma corrente, va in overflow (cioè ritorna a zero) dopo circa 50 giorni.

- micros(): restituisce i milionesimi di secondo da quando è azionato il programma corrente, va in overflow dopo circa 70 minuti.

- delay(msec): provoca un ritardo in millisecondi.

- delayMicroseconds(microsec): provoca un ritardo in microsecondi.

Gestione della porta seriale

Anche se ci sono vari modi per accedere ad una piattaforma Arduino (via wireless, bluetooth, ethernet, …) la modalità di accesso ai dati analizzati durante l'esecuzione dello sketch è stata fatta tramite porta seriale.
Wiring supporta diverse funzioni built-in per gestire la connessione con le porte, tutti i vari metodi appartengono all'oggetto statico Serial.
Di seguito citiamo le funzioni basilari che permettono una veloce comunicazione:

- begin(baud rate): questa funzione apre la comunicazione con la porta seriale, è richiesto inoltre di indicare il baud rate, ovvero la velocità di trasmissione di un

numero prefissato di simboli al secondo. Questo parametro può variare da un minimo di 300 baud/sec fino a 115200 baud/sec. Da sottolineare che anche il programma risiedente nel computer deve supportare la comunicazione via porta seriale con lo stesso baud rate utilizzato dallo sketch, oltre ad indicare il tipo di porta scelta.

- available(): indica se ci sono dati in lettura; questo metodo non è bloccante, ma avverte solo se ci sono dati disponibili o meno. Per avere una comunicazione efficace con il pc collegato con la porta seriale, bisogna che sia messo all'interno del ciclo loop dello sketch; in questa maniera il buffer della porta seriale di Arduino sarà periodicamente scansionato per controllare eventuali dati ricevuti.

- readBytes(buffer, length): funzione di lettura dei dati in input, vengono inseriti nel buffer dato fino alla lunghezza prestabilita. Anche questo metodo non è bloccante e deve essere inserito in loop con la stessa logica proposta per available(). Questa funzione prevede ulteriori varianti per leggere e formattare dati in entrata, ma che alla fine si possono riportare tutti alla lettura non bloccante di byte in input.

- print(ascii_text): scrive in output una stringa di testo. Ci sono altre funzioni simili a questa per inviare dati a seconda della loro formattazione o della loro codifica.

- end(): chiude la comunicazione con la porta seriale, per riaprirla bisogna richiamare il metodo begin presentato prima.

Attraverso queste semplici funzioni si può comunicare tranquillamente con una piattaforma Arduino, per operazioni più sofisticate sulla porta sono fornite altre funzioni più specifiche.

Librerie i2cdev

Oltre alle funzioni di base del linguaggio Wiring, è possibile sfruttare librerie aggiuntive.
Nel nostro caso sono state utilizzate le i2cdev, ovvero delle librerie in C fornite da Arduino

che permettono di utilizzare il sistema elettro-meccanico descritto in precedenza per catturare i movimenti.
Le librerie prendono il nome dall'acronimo di Inter Integrated Circuit (I^2C), ovvero un sistema di comunicazione seriale utilizzato tra i circuiti integrati.
Queste librerie servono ad utilizzare svariati tipi di attuatori e sensori, ci si soffermerà solo alle funzioni analizzate per i nostri scopi.
Come è stato detto prima, lo sketch ha una struttura molto semplice e le azioni sono in loop, quindi ripetitive. Senza entrare nella spiegazione del codice prodotto, ad ogni ciclo viene analizzato il MEMS e attraverso delle funzioni specifiche vengono riportati i dati di accelerazione e di velocità angolare nell'attimo della scansione.
Le librerie possono restituire i valori numerici ottenuti direttamente dal MEMS senza aggiungere interpretazioni (dati raw), quindi relativi alla differenza di potenziale analizzato;
inoltre possono riportare una interpretazione più complessa delle informazioni percepite, come angoli, vettori di accelerazioni e quaternioni attraverso il controllo del DMP (il Digital Motion Processor presentato prima).
Senza analizzare tutte le i2cdev, verranno mostrate le funzioni di base utilizzate durante la stesura del progetto e specifiche per l'utilizzo del MEMS MPU6050.

- initialize(): attiva ed inizializza il MEMS, prepara il giroscopio e l'accelerometro alla scansione.

- reset(): resetta lo stato di avanzamento del MEMS, quindi qualsiasi rotazione tenuta in memoria indicante l'attuale posizione del sensore viene cancellata per riportare il MEMS allo stato iniziale. Questa funzione è utile per resettare l'accelerometro ed il giroscopio, quindi per diminuire gli errori di misurazione che alla fine andrebbero a sommarsi con l'avanzare del tempo (fenomeno descritto in precedenza nel paragrafo dedicato all'hardware). Questa funzione però non è stata utilizzata poiché lo scopo dell'elaborato consiste nel fare una piena interpretazione dei dati ottenuti. Infatti un resettaggio da parte della piattaforma Arduino potrebbe implicare una incorretta interpretazione dei dati raccolti: si potrebbe imporre al sensore di eseguire il reset con una cadenza stabilita a priori oppure analizzare condizioni specifiche come l'assenza di

stimoli esterni (ovvero la piattaforma decide di essere ferma), solo che queste azioni potrebbero avvenire in situazioni critiche per cui l'analisi dei singoli movimenti potrebbe essere falsata. La soluzione adottata è quindi di non adoperare questa funzione nello sketch e studiare i dati dopo il loro salvataggio su computer.

- getMotion6(<puntatori a variabili per accelerazione e rotazione>): questa funzione restituisce a parametro di input i valori senza alcun filtro (non viene utilizzato il processore DMP), perciò compresi tra una scala di valori da interpretare. Ci sono funzioni simili a queste, come ad esempio getAcceleration e getRotation, che alla fine restituiscono le stesse informazioni di questo metodo.

- dmpInitialize(): questa funzione inizializza il processore DMP allo scopo di interpretare i dati ottenuti come descritto prima.

- setDMPEnabled(bool): questa funzione indica di utilizzare il DMP per i dati da restituire, va utilizzata dopo l'inizializzazione.

- getFIFOCount(): se si decide di utilizzare il DMP, allora le librerie provvedono a salvare i dati filtrati in pacchetti di informazioni in una coda FIFO (first in first out). Questa funzione indica se ci sono dati da analizzare.

- dmpGetFIFOPacketSize(): restituisce la dimensione di un pacchetti di dati filtrati dal DMP; questa informazione è utile in fase di estrazione di dati dalla coda FIFO.

- getFIFOBytes(buffer, numbyte): preleva il numero di byte richiesti dalla coda e li inserisce nel buffer dato in input, di solito il numero di byte è un multiplo della dimensione del pacchetto restituita dalla funzione precedente. In questa maniera si possono estrarre i dati e quindi fare spazio nella coda.

- resetFIFO(): toglie i pacchetti registrati fino ad adesso.

- dmpGetQuaternion(quaternione, buffer): questo metodo preleva i dati inseriti nel buffer dalla funzione getFIFOBytes vista in precedenza e li trasforma in un quaternione (anche in questo caso è a parametro di input allo scopo di essere modificato). Nelle librerie sono comprese altre entità matematiche che possono essere rese per indicare la rotazione del dispositivo (angoli e vettori di rotazioni), però si rifanno sempre alla struttura del quaternione estratta con queste funzioni.

E' stata presentata solo una piccola parte della libreria dedicata all'utilizzo del processore MPU, infatti ci si è limitati a presentare metodi in grado di estrarre una rotazione e di renderla con una entità matematica (quaternione); sono presenti altre funzioni più complesse adatte ad una maggiore interattività con la piattaforma, che però non sono state inserite perché non utilizzate al fine del progetto svolto.

Considerazioni

In questo capitolo è stata fatta una panoramica sia sull'hardware sia sul software che comprendono il progetto Arduino.
Si è visto come è fatto il componente hardware per l'estrazione delle rotazioni e accelerazioni (MPU6050) ed anche come gli sia stato affiancato un processore (DMP) per filtrare i dati in modo da diminuire gli errori derivanti dalle misurazioni fatte; inoltre è stata presentata la libreria utilizzata per estrarre i dati dall'hardware e trasformarli in entità matematiche, i quaternioni.
Per quanto questo meccanismo sia preciso, andrà sempre incontro a delle imperfezioni, seppur lievi, nel calcolo della posizione di rotazione della piattaforma utilizzata; quindi il compito dell'analisi dei valori raccolti fatta da computer comprenderà anche la decisione di una tolleranza per indicare quando il valore delle rotazioni estratte è da considerarsi attendibile oppure quando questo valore deve essere modificato e filtrato ulteriormente per ricostruire la successione di rotazioni volute.

Capitolo 3

Sistemi di rappresentazione tridimensionale

In questo capitolo si descriveranno i vari tipi di sistemi di rappresentazione tridimensionale analizzati per mappare il movimento di un solido, nel nostro caso la piattaforma Arduino utilizzata per estrarre i dati di rotazione. Verranno dapprima indicate le caratteristiche generali dei sistemi di rappresentazione prendendo spunto dai due più usati sistemi ovvero quello degli angoli di Euclide e di Tait-Bryan, utilizzati in questo ambito per mostrare le loro somiglianze e differenze; infine si descriverà il sistema adottato per la piattaforma Arduino.

Definizione

Un sistema di rappresentazione tridimensionale è una convenzione per descrivere la posizione di un sistema di riferimento XYZ (contrassegnato col colore blu nell'immagine 3.1) solidale con un corpo rigido attraverso una serie di rotazioni a partire da un sistema di riferimento fisso xyz (nell'immagine 3.1 è contrassegnato dal colore rosso). I due sistemi di riferimento coincidono nell'origine.
Quindi si può intuire che in un sistema di riferimento si ha la rotazione di un sistema di assi cartesiani per indicare il movimento di un corpo rigido.
Per un ulteriore movimento, allo scopo di calcolare il nuovo sistema di riferimento solidale all'oggetto ruotato, si considera adesso come sistema di riferimento fisso quello che precedentemente ha ruotato insieme al corpo, e così via.
Sempre dalla figura 3.1, si può vedere che un sistema di riferimento di rotazione descrive la rotazione dall'asse x verso l'asse X, e lo stesso si può dire delle coppie degli assi y - Y e z - Z.

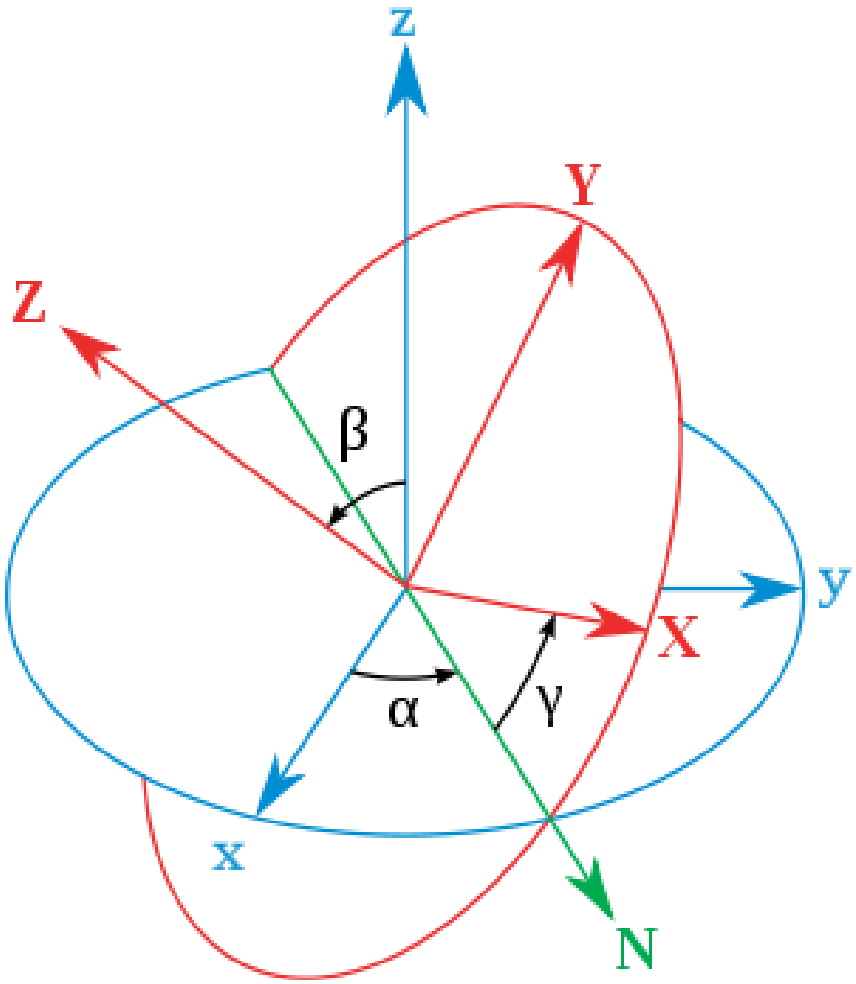

Figura 3.1

Queste rotazioni possono essere indicate con angoli, quindi avremo l'angolo alfa che indica la rotazione dall'asse x verso l'asse X (la posizione dell'asse X prima che agissero le rotazioni degli altri angoli è rappresentato dalla freccia verde in figura), l'angolo gamma indica la rotazione verticale dall'asse y verso l'asse Y e beta indica l'angolo tra z e Z.

Quindi per descrivere una rotazione del sistema di riferimento cartesiano bisogna indicare gli angoli di rotazione degli assi, più altre convenzioni che nel seguito verranno analizzate.

Per mappare le rotazioni in una entità matematica si possono utilizzare sia matrici che quaternioni, questi ultimi saranno ripresi in un capitolo a parte; per adesso infatti ci soffermeremo sulle convenzioni utilizzate senza addentrarci nell'ambito della matematica.

Convenzioni possibili

Possono essere utilizzati diversi tipi di standard per descrivere una rotazione, ad esempio si deve definire quali angoli hanno la precedenza rispetto ad altri nell'inserimento nella struttura matematica che li deve memorizzare, oppure se la rotazione è sinistrorsa oppure destrorsa, addirittura si deve stabilire la nomenclatura delle rotazioni e la loro associazione rispetto al sistema di assi cartesiano fisso come descritto precedentemente.

Nomenclatura

Gli angoli di rotazione possono essere chiamati in diversi modi a seconda del loro campo di utilizzo, ad esempio possono avere nomi di lettere greche in campi legati alla matematica o alla fisica (questo si ha nel sistema di rotazione di Eulero) oppure possono avere nomi legati a precisi movimenti di un velivolo in ambito aeronautico (sistema di Tait-Bryan).
Negli angoli euleriani si indicano con le lettere greche phi, theta e psi (oppure alfa, beta e gamma come in figura 3.1) rispettivamente le rotazioni degli assi x, y e z; per la codifica di Tait-Bryan gli angoli di rotazione sono chiamati con bank, heading ed attitude (o anche roll, pitch e yaw). Questi termini derivano dall'aeronautica per indicare la posizione di un velivolo rispetto ai movimenti impartiti dai comandi del pilota.
Si identificano le rotazioni di yaw – attitude (imbarcata) come la rotazione attorno all'asse verticale, pitch – heading (beccheggio) come la rotazione intorno all'asse trasversale, mentre con roll – bank (rollìo) si indica la rotazione intorno all'asse longitudinale (figura 3.2).
La relazione tra Tait-Bryan ed Eulero è rappresentata in figura 3.3.
Una ulteriore differenza tra questi due sistemi è che in Eulero non viene associata la disposizione degli assi x, y e z del sistema al corpo ruotato, senza particolari distinzioni per la sua forma (figura 3.1), mentre in Tait-Bryan il sistema di assi cartesiano è sempre solidale alla forma del velivolo considerato, ovvero si ha un asse che passa per la fusoliera, un asse verticale al velivolo ed uno trasversale che passa per le ali.

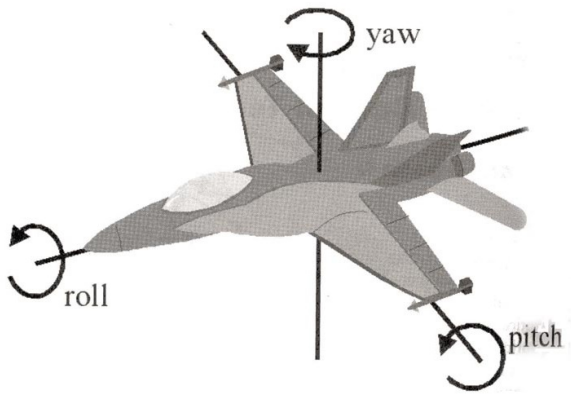

Figura 3.2

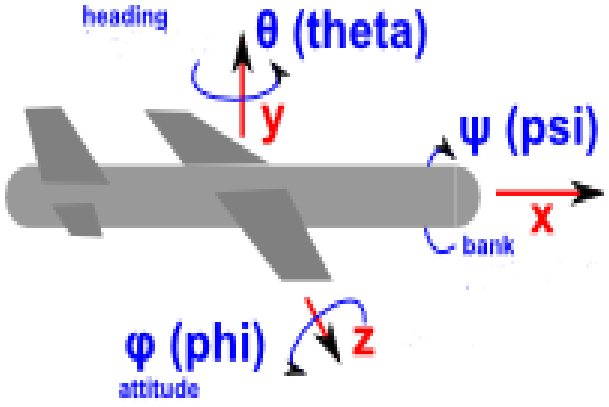

Figura 3.3

Ordine di combinazione degli angoli

Oltre alla nomenclatura, ci sono altre sostanziali differenze che determinano il sistema di rotazione, come ad esempio la precedenza nel considerare gli angoli.

Per spiegare con un esempio questa differenza, consideriamo un veicolo terrestre a cui vengono applicate delle rotazioni di heading (theta) ed attitude (phi) pari a 90 gradi.

Se questi valori sono dati contemporaneamente, si deve scegliere se applicare prima l'heading oppure l'attitude, poiché si otterrebbero altrimenti delle posizioni differenti del veicolo a seconda della diversa applicazione.

Per quanto riguarda Eulero, senza entrare nell'ambito di formule matematiche, viene data precedenza prima alla rotazione verticale (equivalente all'attitude, come in figura 3.3) e poi a quella orizzontale (heading), per poi considerare alla fine il bank.

In Tait-Bryan invece prima viene considerato l'heading e poi l'attitude ed infine il bank. Questa differenza viene mostrata nelle figure 3.4.

Le immagini fanno riferimento come posizione degli assi alla figura 3.3, infatti l'aereo nella figura 3.3 ha la stessa posizione dell'auto dell'immagine 3.4a con ancora nessuna rotazione applicata, mentre nelle figure 3.4b e 3.4c vengono mostrate le trasformazioni di 90 gradi di attitude ed heading rispetto agli angoli di Euclide (figura 3.4b) e di Tait-Bryan (figura 3.4c).

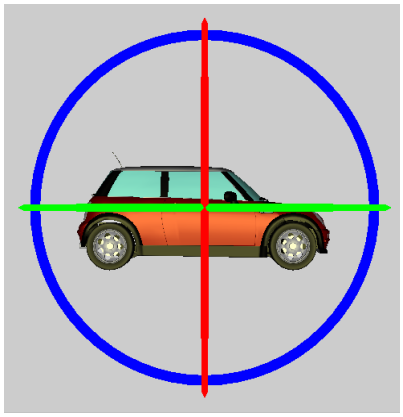

Figura 3.4a

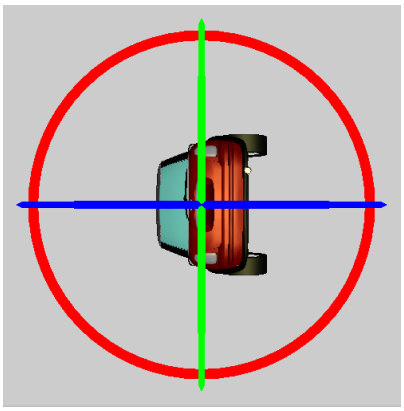

Figura 3.4b

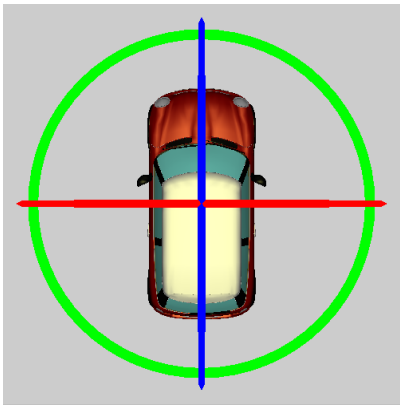

Figura 3.4c

Da ciò possiamo dire che, date delle rotazioni rispetto ai tre assi di uno spazio cartesiano, bisogna rispettare un certo ordine di inserimento degli angoli per essere coerenti con le misurazioni fatte, altrimenti non si potrebbe ricreare fedelmente la rotazione eseguita dal corpo studiato.
L'ordine di inserimento alla fine viene reso con formule matematiche, quindi gli angoli saranno inseriti ed estratti in matrici o quaternioni dalle formule specifiche che mantengono al loro interno un ordine ben preciso.

Rotazione destrorsa e sinistrorsa

Un'altra convenzione da stabilire è quando si considera positivo il verso dell'asse di rotazione. Si hanno le due convenzioni standard, ovvero la rotazione destrorsa e sinistrorsa, che seguono rispettivamente la ben nota regola della mano destra o sinistra.
Sia i sistemi di Euclide e Tait-Bryan sono destrorsi. Siccome il sistema della mano destra è utilizzato in ambito vettoriale, questa risulta essere la convenzione più usata.

Associazione con il sistema di assi cartesiani

Fino ad ora abbiamo descritto le possibili nomenclature degli angoli, il verso del loro asse di rotazione e la priorità che possono avere gli uni rispetto agli altri.
Una ulteriore convenzione da stabilire risiede nella mappatura degli angoli presentati con gli assi x, y e z del sistema cartesiano. Nel sistema di Euclide questo problema non sussiste,

infatti gli angoli phi, theta e psi sono associati per definizione agli assi x, y e z come in figura 3.1.

Gli angoli di Tait-Bryan invece descrivono un movimento di un velivolo rispetto alla sua forma, quindi si tiene conto se ruota sul suo asse longitudinale (bank - roll), se varia la sua quota (attitude - pitch) oppure se cambia direzione (heading - yaw); questo senza indicare esplicitamente la connessione con gli assi x, y e z. Di solito nelle formule classiche viene associato il bank all'asse x, l'attitude all'asse z e l'heading all'asse y, come mostrato in figura 3.3.

Siccome la nomenclatura di Tait-Bryan è più espressiva di quella di Eulero, viene di solito utilizzata anche in altri ambiti (come appunto nella rotazione del sensore Arduino) con una diversa mappatura degli assi cartesiani.

Sistema utilizzato per Arduino

Dopo aver presentato le diverse caratteristiche che contraddistinguono i vari sistemi di rotazione, si può adesso descrivere la convenzione utilizzata dalle librerie per la piattaforma Arduino che trattano rotazioni.

Il sistema preso in esame ha elementi di entrambi i metodi di riferimento, ovvero si ha un sistema di riferimento solidale alla forma del dispositivo Arduino (nelle stesse librerie si utilizzano le indicazioni di yaw, pitch e roll per indicare gli angoli di rotazione) e l'interpretazione viene pensata come un sistema euleriano, quindi si tiene in considerazione in ordine di inserimento l'angolo verticale, quello orizzontale e la rotazione sull'asse longitudinale, quindi il pitch, il yaw ed il roll in questa sequenza. Una ulteriore peculiarità è la mappatura degli assi di rotazione con gli assi dello spazio cartesiano, infatti se nella figura 3.3 per convenzione al pitch viene collegato l'asse Z, al roll viene collegato l'asse X e al yaw l'asse Y, nelle librerie considerate per Arduino si utilizzano convenzioni differenti a quelle riportate usualmente, ovvero si identifica con roll la rotazione rispetto all'asse Y, con pitch quella rispetto all'asse X e con yaw la rotazione rispetto all'asse Z, sempre considerando un sistema di assi cartesiano solidale al sensore.

Per quanto riguarda il verso dell'asse di rotazione, questo è stabilito come destrorso.

Queste convenzioni non sono state scelte come le più vantaggiose rispetto a quelle disponibili,

ma sono le caratteristiche proprie dell'hardware Arduino viste nel capitolo precedente. Infatti i valori restituiti dal MEMS Motion Tracking rappresentano un sistema di riferimento solidale al circuito ed inoltre indicano gli spostamenti rispetto ad un altro sistema di riferimento spaziale fisso. Questo sistema di riferimento deve essere percepito dal circuito per poter restituire lo spostamento fatto, e dipende dalla posizione iniziale del sensore: se ad esempio la piattaforma Arduino è situata in un piano orizzontale senza essere soggetta a movimenti, si ottiene il sistema di riferimento fisso, quindi se ruoto il dispositivo questo percepirà delle rotazioni rispetto alla posizione di inizializzazione del circuito.

Una caratteristica aggiuntiva che merita di essere menzionata è che il sistema di riferimento fisso deve avere una componente perpendicolare al suolo, quindi parallela alla forza di gravità, percepita dall'accelerometro del sensore. Per struttura dell'hardware, l'asse fisso perpendicolare al suolo è indicato con z, mentre gli assi x e y sono paralleli al suolo e dipendono dalla posizione di inizializzazione del Motion Tracking.

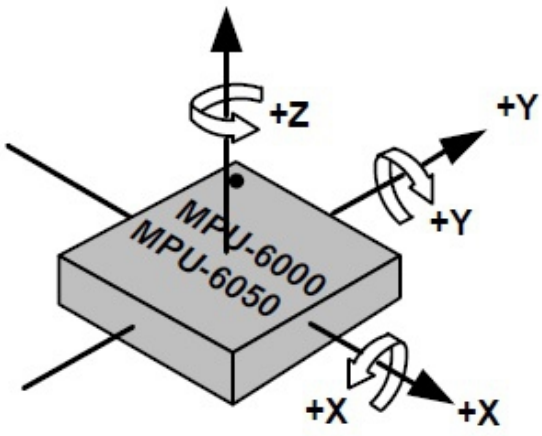

Figura 3.5

Nell'immagine 3.5 viene mostrato il sistema di assi solidale alla MPU ed anche il verso di rotazione destrorso utilizzato dal sensore.

Quando l'apparecchio si muove, questo restituisce valori che riportano il movimento del sistema di riferimento solidale al circuito MEMS (la posizione assi x,y e z sono stabiliti dalla forma a croce del circuito come detto nel capitolo precedente) e che indicano la posizione rispetto al sistema di assi fissi appena detto.

Capitolo 4

Quaternioni

In questo capitolo si descrivono i quaternioni, ovvero le entità matematiche da noi utilizzate per rappresentare le rotazioni dello spazio tridimensionale.
I quaternioni furono formalizzati dal matematico William Hamilton, il quale era alla ricerca di un metodo per estendere i numeri complessi su un numero maggiore di dimensioni spaziali.
Vengono utilizzati principalmente nella rappresentazione di rotazioni e direzioni nello spazio tridimensionale ed hanno applicazioni nella computer grafica 3D, nella fisica e nel controllo dell'assetto di veicoli in generale; nel nostro caso specifico sono usati per calcolare le rotazioni della piattaforma Arduino utilizzata, in modo da poter ricostruire ed analizzare i movimenti compiuti.
Di seguito saranno mostrate le caratteristiche basilari di queste entità, per poi approfondire gli aspetti relativi alle proprietà e formule utilizzate in collegamento col sistema di riferimento per Arduino.

Definizione

Come detto nell'introduzione, un quaternione è una estensione dei numeri complessi.
Come è noto, un numero complesso è formato da due componenti, ovvero da una parte reale ed una parte immaginaria; può essere mappato in un piano cartesiano e quindi essere rappresentato come un vettore. Se si ha un numero complesso del tipo $x+i\cdot y$ in un piano si otterrà come coordinata dell'ascissa il valore x, come ordinata il valore y. Le operazioni tra questi numeri si eseguono con la nota regola $i^2=-1$.
I quaternioni, come è stato detto in precedenza, sono una estensione dei numeri complessi allo

scopo di mappare lo spazio, solo che a differenza del campo complesso le dimensioni utilizzate sono quattro, una reale e tre immaginarie. Questa peculiarità li rendono difficili da visualizzare nella loro interezza poiché consistono in un'astrazione quadridimensionale che soddisfa delle proprietà simili a quelle legate ai numeri complessi.
Un quaternione può essere descritto dalla formula:

$$t + x \cdot i + y \cdot j + z \cdot k \tag{4.1}$$

ove t, x, y e z sono numeri reali e i, j e k sono componenti immaginarie. Se y e z sono pari a 0, si ottiene banalmente un numero immaginario, mentre se x, y e z sono pari a 0 si ha un numero reale.
Somma e prodotto di due quaternioni sono definiti tenendo conto delle relazioni:

$$i^2 = j^2 = k^2 = -1 \tag{4.2}$$

$$\begin{aligned} i \cdot j &= k \\ j \cdot k &= i \\ k \cdot i &= j \\ j \cdot i &= -k \\ k \cdot j &= -i \\ i \cdot k &= -j \end{aligned} \tag{4.3}$$

Si può vedere da (4.2) le analogie con la regola $i^2 = -1$ che governa i numeri complessi. Diversamente da questi, il prodotto di quaternioni non è, in generale, commutativo, appunto per le regole (4.3) che indicano un ben preciso ordine nelle operazioni da compiere.

Proprietà basilari

I quaternioni hanno molte caratteristiche analoghe ai numeri complessi, come la norma ed il coniugato, però si differenziano per non avere un prodotto commutativo come accennato in precedenza.
Di seguito vengono indicate le proprietà principali dei quaternioni.

Prodotto non commutativo

Si può applicare il prodotto a due quaternioni considerandoli al pari di polinomi, per poi applicare le regole (4.3) per una eventuale semplificazione.
In generale, considerando due quaternioni q1 e q2, il prodotto di q1 per q2 di solito differisce dal prodotto di q2 per q1 ($q_1 \cdot q_2 \neq q_2 \cdot q_1$).
Siccome il prodotto dei quaternioni prima delle semplificazione delle componenti immaginarie ha sempre la stessa struttura, si può direttamente ottenere il quaternione risultante operando da subito sui parametri a, b, c e d della rappresentazione (4.1), quindi se si considerano ad esempio i quaternioni:

$q_1 = t_1 + x_1 \cdot i + y_1 \cdot j + z_1 \cdot k$, $q_2 = t_2 + x_2 \cdot i + y_2 \cdot j + z_2 \cdot k$

il prodotto $q_1 \cdot q_2$ sarà pari a $q_{ris} = t_{ris} + x_{ris} \cdot i + y_{ris} \cdot j + z_{ris} \cdot k$

ove:

$$t_{ris} = t_1 \cdot t_2 - x_1 \cdot x_2 - y_1 \cdot y_2 - z_1 \cdot z_2 \qquad (4.4)$$
$$x_{ris} = t_1 \cdot x_2 + x_1 \cdot t_2 + y_1 \cdot z_2 - z_1 \cdot y_2$$
$$y_{ris} = t_1 \cdot y_2 + y_1 \cdot t_2 + z_1 \cdot x_2 - x_1 \cdot z_2$$
$$z_{ris} = t_1 \cdot z_2 + z_1 \cdot t_2 + x_1 \cdot y_2 - y_1 \cdot x_2$$

Questa definizione di prodotto è oltremodo conveniente dal punto di vista computazionale poiché permette di ottenere un risultato diretto senza ulteriori raffinamenti, come ad esempio procedere con semplificazioni utilizzando le regole (4.3).
Dalla (4.4) si può facilmente vedere che il prodotto di due quaternioni può essere risolto con 16 moltiplicazioni e 12 somme, quindi 28 operazioni.
Come vedremo più avanti, i quaternioni rappresentano un sistema di riferimento nello spazio, ed un loro prodotto implica una trasformazione del sistema di riferimento. Se gli stessi calcoli fossero affidati a delle matrici 3 x 3, avremmo quindi come prodotto una matrice risultante dalla moltiplicazione di altre due in cui ogni elemento si otterrebbe con 3 moltiplicazioni e 2 somme, quindi si avrebbero in totale 27 moltiplicazioni e 18 somme, ovvero 45 operazioni.
Quindi da questo esempio è lampante vedere come l'uso dei quaternioni nelle trasformazioni porti a dimezzare il costo computazionale.

Associatività

Questa caratteristica non riguarda solo il prodotto dei quaternioni, ma può essere estesa anche alla somma, infatti i quaternioni possono essere sommati e moltiplicati tra di loro come i numeri complessi (a patto di mantenere nel prodotto l'ordine dei moltiplicandi poiché non commutativo) e godono della proprietà associativa rispetto a queste due operazioni, per cui dati tre quaternioni q_1, q_2, q_3 possiamo dire che $q_1+(q_2+q_3)=(q_1+q_2)+q_3$ e che $q_1 \cdot (q_2 \cdot q_3)=(q_1 \cdot q_2) \cdot q_3$.

Identità

Si consideri un particolare tipo di quaternione utile nelle moltiplicazioni, ovvero il quaternione con parte reale pari ad 1 e parti immaginarie pari a 0 (paragonabile quindi al numero reale 1).
Un quaternione cosi formato viene indicato come **identità** ed è formulato attraverso la seguente scrittura:

$$q_{id}=1+0 \cdot i+0 \cdot j+0 \cdot k \tag{4.5}$$

Questo quaternione ha la caratteristica di essere l'elemento neutro per la moltiplicazione, ovvero per qualsiasi quaternione vale che:

$$q \cdot q_{id}=q_{id} \cdot q=q$$

Questa proprietà è utile per impostare valori iniziali per misurazioni successive di quaternioni, infatti non modifica il valore del quaternione ad esso moltiplicato.

Quaternione nullo

Un altro quaternione notevole è quello con tutti i parametro posti a 0, quindi ha la forma $0+0 \cdot i+0 \cdot j+0 \cdot k$. Questo tipo di quaternione annulla qualsiasi prodotto ed è l'elemento neutro rispetto alla somma.

Coniugato

Rispetto a (4.1), il coniugato di un quaternione q è definito come:

$$q^* = t - x \cdot i - y \cdot j - z \cdot k \tag{4.6}$$

ovvero si inverte il segno dei parametri delle componenti immaginarie. Nelle parti successive di questo elaborato verrà mostrato come i quaternioni possono rappresentare una rotazione nello spazio tridimensionale; da ciò si può intuire che il coniugato produce una rotazione contraria del quaternione da cui deriva.

Norma

Rispetto a (4.1), la norma (detta anche modulo) di un quaternione q è definita come:

$$|q| = \sqrt{(t^2 + x^2 + y^2 + z^2)} \tag{4.7}$$

Quindi un quaternione normalizzato q è definito come $\frac{q}{|q|}$ ed ha modulo unitario.

Da notare che un quaternione ha la stessa norma del suo coniugato, quindi $|q| = |q^*|$.

Una proprietà aggiuntiva degna di nota è la norma del prodotto di quaternioni: dati due quaternioni q_1 e q_2 si può dimostrare con le (4.4) che

$$|q_1| \cdot |q_2| = |q_1 \cdot q_2| \tag{4.8}$$

ovvero la norma del prodotto di due quaternioni è pari al prodotto delle loro norme. Da notare che sulla natura di q1 e q2 non si fa alcuna supposizione, infatti possono essere quaternioni qualsiasi.

La norma è sempre positiva ed è uguale a 0 solo quando q è il quaternione nullo.

Inverso

Un quaternione q non nullo ha un inverso rispetto alla moltiplicazione, ottenibile con la formula:

$$q^{-1}=\frac{q^*}{|q|^2} \tag{4.9}$$

In questa formula si inseriscono due concetti visti prima, ovvero il coniugato e la norma; ne consegue che il reciproco di un quaternione non è altro che il coniugato diviso per il quadrato del suo modulo.

Quaternioni unitari

Consideriamo adesso un sottoinsieme di quaternioni con delle proprietà particolari, ovvero i quaternioni unitari, cioè dotati di norma pari ad 1.
In questo insieme non è compreso il quaternione nullo, quindi è sempre definito l'inverso. Ne consegue dalla (4.9) che i quaternioni unitari hanno il coniugato uguale all'inverso.
Dalla (4.8) possiamo invece vedere che il prodotto di due quaternioni unitari è a sua volta un quaternione unitario, avendo il modulo pari al prodotto dei moduli dei due moltiplicandi, ossia 1.
Da queste premesse, possiamo dire che i quaternioni unitari formano un gruppoide, ovvero un insieme di entità matematiche ove è definita una operazione binaria (in questo caso la moltiplicazione) il cui risultato è un altro elemento dell'insieme (proprietà di chiusura).
Siccome le proprietà elencate per i normali quaternioni valgono anche per i quaternioni unitari, possiamo indicare che questo insieme è un gruppo, ovvero un gruppoide ove sono definite le proprietà di associazione, di inverso e la presenza di elemento neutro rispetto alla moltiplicazione.
Il gruppo non è abeliano, ovvero non vale la proprietà commutativa del prodotto.
Anche l'insieme più vasto dei quaternioni è un insieme chiuso con la proprietà associativa e l'elemento neutro, solo che non si ha l'inverso per ogni quaternione, poiché per definizione il

quaternione nullo non ha inverso. Un insieme con queste proprietà viene indicato come monoide non abeliano.

Siccome i quaternioni sono una rappresentazione quadridimensionale, si possono idealmente mappare in $\mathbb{R}^4$ come dei vettori, quindi rispetto a (4.1), un quaternione q può essere visto come un vettore della forma (t,x,y,z). Con questo particolare punto di vista, i quaternioni unitari formano una ipersfera, ovvero un insieme definito come

$$\{(t,x,y,z) \in \mathbb{R}^4 \mid t^2+x^2+y^2+z^2=1\} \quad .$$

In questa interpretazione, il prodotto di due quaternioni unitari, intesi come due punti sulla sfera, non è altro che un altro punto sulla sfera stessa.

Bisogna sottolineare che per il sottoinsieme dei quaternioni unitari ci sono ulteriori proprietà rispetto a quelle viste in precedenza, ad esempio possiamo introdurre il prodotto di un quaternione unitario con il suo coniugato, ottenendo un risultato pari ad 1, cioè l'identità.

$$q \cdot q^* = q^* \cdot q = 1 \tag{4.10}$$

Si mostrerà più avanti come il coniugato sarà utile per funzioni di reset per l'analisi di rotazioni da un certo punto in poi.

Interpretazione tridimensionale

Fino ad ora abbiamo visto le proprietà dei quaternioni e le analogie con i numeri complessi senza addentrarci nella loro interpretazione, questo allo scopo di mostrare il loro funzionamento dal punto di vista matematico.

Qualsiasi rotazione in tre dimensioni può essere raffigurata come una combinazione di un asse e di un angolo di rotazione; i quaternioni rappresentano un modo semplice per codificare questa rotazione asse – angolo in quattro numeri.

Prima di passare a descrivere le relazioni tra quaternioni e rotazioni, analizziamo prima le analogie con i numeri complessi. Come noto, possono essere rappresentati come vettori in un piano, ed il prodotto di due numeri complessi risulta un nuovo numero avente come modulo il prodotto dei moduli e come angolo (compreso tra l'asse delle ascisse ed il vettore stesso) la

somma degli angoli dei numeri moltiplicati.

Come si evince dalla figura 4.1, si considerano i numeri complessi Z e W aventi angoli differenti rispetto all'asse delle ascisse. Il loro prodotto ZW risulta un numero complesso che ha come modulo il prodotto dei moduli di Z e W e come angolo la somma degli angoli dei moltiplicandi.

Con questo punto di vista, consideriamo di avere rotazioni successive attorno ad un asse e prendiamo in esame i numeri complessi indicanti le diverse fasi successive della trasformazione. In questo caso, moltiplicando in ordine temporale i vari numeri complessi, si ottiene quindi un prodotto che ha come angolo la somma degli angoli di tutte le varie rotazioni compiute per la trasformazione; grazie all'angolo trovato si può passare direttamente dalla situazione iniziale del vettore fino alla rotazione finale; ottenendo quindi la trasformazione complessiva generata da tutte le piccole variazioni fatte.

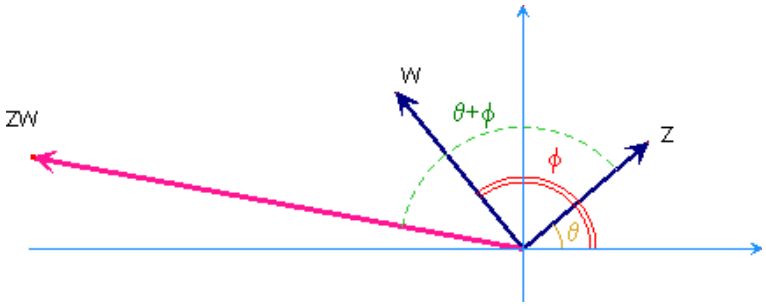

Figura 4.1

Quanto appena detto si può rendere con una rappresentazione in coordinate polari, infatti se indichiamo il modulo del vettore pari a ρ e l'angolo pari a θ, otteniamo che un numero complesso $z=x+i\cdot y$. Questo si può esprime con una coppia ordinata (ρ,θ) attraverso la formula:

$$z=\rho\cdot(\cos(\theta)+i\cdot\sin(\theta)) \tag{4.11}$$

ove $\rho=\sqrt{(x^2+y^2)}$ è il modulo del numero complesso, e θ è l'angolo in senso antiorario compreso tra il vettore e l'asse delle ascisse, quest'ultimo viene indicato come fase o argomento.

Usando la formula di Eulero (nota anche come formula di esponenziale complesso) il numero z presentato in (4.11) può essere scritto come:

$$z=\rho\cdot e^{(i\cdot\theta)} \tag{4.12}$$

utilizzando sempre i valori di ρ e θ visti per la (4.11).

La notazione (4.12) riesce perfettamente a rendere l'idea del prodotto di numeri complessi, infatti considerando i due numeri $z_1=\rho\cdot e^{(i\cdot\theta_1)}$ e $z_2=\rho\cdot e^{(i\cdot\theta_2)}$, il loro prodotto sarà il numero complesso $z_1\cdot z_2=\rho_1\cdot\rho_2\cdot e^{(i\cdot(\theta_1+\theta_2))}$, ovvero un numero che ha come modulo il prodotto dei moduli e come angolo la somma degli angoli.

Dopo aver visto l'interpretazione bidimensionale, alla stessa maniera si possono immaginare il funzionamento dei quaternioni: data una rotazione intorno ad un vettore tridimensionale, questa può essere scomposta in tre rotazioni intorno agli assi x, y e z di R^3. Considerando il sistema di riferimento cartesiano sempre solidale alla rotazione eseguita, si possono prendere in esame le trasformazioni successive come quaternioni, che moltiplicati in ordine formeranno il quaternione che rappresenterà la rotazione complessiva.

Rotazioni in un quaternione

Nel capitolo precedente si sono visti diversi sistemi di riferimento, in particolare quello adottato dal sistema Arduino. Adesso si deve mappare il sistema visto in quaternioni, allo scopo di estrarre informazioni per ricostruire i movimenti eseguiti dal dispositivo dotato di giroscopio. In seguito si prenderanno in considerazione i quaternioni unitari, quindi si farà riferimento a questo sottoinsieme. Esistono formule anche per utilizzare i quaternioni normali per le rotazioni, solo che non saranno prese in esame poiché il loro funzionamento è simile a quanto riportato per il sottoinsieme unitario.

Come abbiamo detto in precedenza, il sistema di riferimento fisso dipende dalla posizione iniziale del sensore, e che il sistema solidale alla piattaforma ha gli assi x, y e z definiti direttamente dall'hardware. Le librerie utilizzate per Arduino restituiscono angoli di rotazione e quaternioni, e sono governate da precise regole matematiche.

In questo paragrafo si mostrano le formule per inserire una rotazione in un quaternione, per poi vedere come estrarre informazioni di rotazione.

Da rotazione a quaternione

Il giroscopio della piattaforma Arduino, quando scansionato, fornisce contemporaneamente delle misurazioni di rotazioni dei 3 assi. Questi dati possono essere trasformati in angoli di rotazione yaw, pitch e roll. L'ordine di inserimento delle rotazioni è un punto cruciale per l'ottenimento del corretto quaternione, infatti se si inserisce prima una rotazione in un asse (esempio una rotazione di yaw) e poi una rotazione in un altro asse (di roll) si ottiene un quaternione diverso rispetto a quello che si otterrebbe se si invertisse l'ordine.
A tal proposito è necessario avere una formula di inserimento (e poi, come vedremo, di relativa estrazione) che possa gestire contemporaneamente questi valori. La formula per la piattaforma Arduino è simile a quella per gli angoli euleriani, differisce solo dalla posizione degli assi X,Y,Z di riferimento dello spazio cartesiano come detto in precedenza.
Dati degli angoli di rotazione yaw, pitch e roll, definiamo i seguenti angoli intermedi:

$$p=\frac{pitch}{2} \quad , \quad y=\frac{yaw}{2} \quad , \quad r=\frac{roll}{2} \quad .$$

definiamo inoltre i seguenti risultati intermedi, utili per ottenere le formule di rotazione:

$$a=\cos(p)\cdot\cos(y)$$
$$b=\sin(p)\cdot\cos(y)$$
$$c=\sin(p)\cdot\sin(y)$$
$$d=\cos(p)\cdot\sin(y)$$

Le formule che trasformano una rotazione in angoli in un quaternione sono definite come:

$$t=a\cdot\cos(r)+c\cdot\sin(r)$$
$$x=a\cdot\sin(r)-c\cdot\cos(r)$$
$$y=b\cdot\sin(r)+d\cdot\cos(r)$$
$$z=b\cdot\cos(r)-d\cdot\sin(r)$$

Sono stati così inseriti i tre angoli dati in una entità che quindi può essere moltiplicata successivamente con altri quaternioni ottenuti allo stesso modo, questo allo scopo di aggiornare i cambiamenti di rotazione del nostro apparecchio Arduino. Questo passaggio è

cruciale poiché permette di inserire una rotazione spaziale in una entità matematica a cui si possono applicare tutte le proprietà elencate precedentemente, in primis il prodotto di quaternioni. Questo infatti porterà all'ottenimento della somma degli angoli che si susseguiranno dalla scansione del giroscopio del sensore. Questa operazione non può essere fatta direttamente sugli angoli, poiché anche una singola rotazione su di un asse può modificare interamente il valore di tutti gli altri angoli, mentre i quaternioni mantengono il valore degli angoli considerati in un'unica formula. Perciò moltiplicare un quaternione con uno successivo porta ad un comodo e facile aggiornamento della posizione spaziale del corpo ruotato.
Bisogna sottolineare che questo processo è implementato direttamente dall'hardware Arduino visto in precedenza attraverso il processore DMP, il quale fornisce il quaternione della rotazione attuale rispetto al punto di inizializzazione del MEMS Motion Tracking.

Reset di una rotazione

Per capire come può funzionare una possibile funzione di reset, si consideri una sequenza di rotazioni come delle piccole variazioni che indicano il passaggio da uno stato iniziale ad uno stato finale. In questo caso si prenda in considerazione per esempio i quaternioni q_i con $0 \leq i \leq 5$, ove quelli con indice più basso simboleggiano una rotazione eseguita per prima in ordine cronologico rispetto a quelli con indice maggiore. Indichiamo queste come rotazioni di variazione. Detto ciò, la rotazione complessiva risulta essere $q_{ris} = q_0 \cdot q_1 \cdot q_2 \cdot q_3 \cdot q_4 \cdot q_5$.
Si può definire una rotazione parziale come un quaternione risultante dal prodotto di una sequenza contigua di quaternioni rappresentanti le rotazioni di variazione.
Da quanto detto si ricava che una rotazione parziale può essere il quaternione $q_0 \cdot q_1 \cdot q_2$ oppure anche il quaternione $q_3 \cdot q_4 \cdot q_5$. Indichiamo con la dicitura $q_{a,b}$ un prodotto di quaternioni di variazioni dal momento A fino al momento B; quindi $q_{0,2}$ è uguale a $q_0 \cdot q_1 \cdot q_2$ oppure $q_{3,5} = q_3 \cdot q_4 \cdot q_5$.
Se vengono fornite le rotazioni di variazione sotto forma di quaternioni, questi si possono moltiplicare ogni volta con il prodotto precedente, così da ottenere una sequenza di rotazioni parziali. In definitiva i quaternioni $q_{0,0}$, $q_{0,1}$, $q_{0,2}$, $q_{0,3}$, $q_{0,4}$, $q_{0,5}$

indicano tutte le rotazioni compiute dal nostro oggetto a partire dal momento 0 fino al momento 5 rispetto alla posizione iniziale assunta.
Analizzarli quindi risulta più agevole, infatti i quaternioni parziali indicano la rotazione eseguita a partire dal sistema di riferimento fisso iniziale fino al momento preso in considerazione; in questa maniera si può studiare la rotazione complessiva senza dover ogni volta ricalcolare tutti i prodotti dall'inizio.
Ipotizziamo adesso di avere solo quaternioni parziali per descrivere la rotazione dell'oggetto e desideriamo studiarne il movimento da un dato momento in poi, quindi, ad esempio, se si arriva fino al momento 5 e si vuole analizzare la rotazione parziale a partire dal 3, alla fine si vuole analizzare il quaternione $q_{3,5}$. Per ottenere questo consideriamo i quaternioni $q_{0,5}$ e $q_{0,2}$, ovvero il quaternione della trasformazione complessiva ed il quaternione parziale formato dalle variazioni che succedono prima del momento 3.
Per ottenere $q_{3,5}$ si può utilizzare la definizione di coniugato, infatti se si trova il coniugato di $q_{0,2}$, ovvero $q_{0,2}^*$, si può eseguire il prodotto $q_{0,2}^* \cdot q_{0,5}$ che sarà pari a $q_{3,5}$, infatti $q_{0,2}^* = (q_0 \cdot q_1 \cdot q_2)^*$ e $q_{0,5} = q_0 \cdot q_1 \cdot q_2 \cdot q_3 \cdot q_4 \cdot q_5$ per definizione di rotazione parziale, e in seguito $(q_0 \cdot q_1 \cdot q_2)^* = q_2^* \cdot q_1^* \cdot q_0^*$ per definizione di coniugato. Eseguiamo adesso il prodotto $q_{0,2}^* \cdot q_{0,5}$ che quindi è pari a $q_2^* \cdot q_1^* \cdot q_0^* \cdot q_0 \cdot q_1 \cdot q_2 \cdot q_3 \cdot q_4 \cdot q_5$. Per la proprietà associativa possiamo applicare le seguenti parentesi, ottenendo $(q_2^* \cdot (q_1^* \cdot (q_0^* \cdot q_0) \cdot q_1) \cdot q_2) \cdot q_3 \cdot q_4 \cdot q_5$. Da qui il nostro quaternione si potrà semplificare notevolmente per la (4.10), ottenendo così che $q_{0,2}^* \cdot q_{0,5} = q_3 \cdot q_4 \cdot q_5 = q_{3,5}$.
Quanto appena detto si può utilizzare per l'analisi dei quaternioni parziali salvati durante la misurazione con la piattaforma Arduino, infatti se vogliamo studiare solo le rotazioni da un dato punto in poi, basta solo prendere il quaternione parziale precedente al punto indicato, trovare il suo coniugato e moltiplicarlo per le trasformazioni parziali volute salvate in precedenza.

Da quaternione a rotazione

Si presume che dopo diverse moltiplicazioni di quaternioni successivi, e quindi di diverse trasformazioni che si sono susseguite nel tempo, si decida di analizzare la rotazione

complessiva. Possiamo quindi estrarre gli angoli yaw, pitch e roll espressi sempre in radianti, questo viene fatto con la seguenti formule:

$$roll = arcotan(2\cdot(t\cdot x + y\cdot z), 1 - 2\cdot(x\cdot x + y\cdot y))$$

$$pitch = arcosin(2\cdot(t\cdot y - z\cdot x))$$

$$yaw = arcotan(2\cdot(t\cdot z + x\cdot y), 1 - 2\cdot(y\cdot y + z\cdot z))$$

ove t, x, y e z sono i parametri della (4.1), mentre arcotan e arcosin sono rispettivamente le funzioni di arcotangente e di arcoseno.
Oltre a questi tipi di angoli, è possibile ottenere la matrice di rotazione derivante dal quaternione preso in esame,
questo principalmente per gestire grafica 3D, come nell'architettura presentata in questo elaborato per usare le trasformazioni in OpenGL. La matrice che si ottiene è una matrice di parametri 3 x 3 utilizzata per le trasformazioni di vettori, come mostrato di seguito:

$$\begin{vmatrix} m0 & m3 & m6 \\ m1 & m4 & m7 \\ m2 & m5 & m8 \end{vmatrix} \cdot \begin{vmatrix} x \\ y \\ z \end{vmatrix} = \begin{vmatrix} m0\cdot x & m3\cdot y & m6\cdot z \\ m1\cdot x & m4\cdot y & m7\cdot z \\ m2\cdot x & m5\cdot y & m8\cdot z \end{vmatrix}$$

ove la prima matrice opera la trasformazione, la seconda è il vettore da trasformare e la terza è il vettore trasformato.
I parametri contrassegnati da m0 fino ad m8 si possono ricavare in questo modo da quelli di (4.1):

$$m0 = 1 - 2\cdot y\cdot y - 2\cdot z\cdot z$$

$$m3 = 2\cdot x\cdot y - 2\cdot z\cdot t$$

$$m6 = 2\cdot x\cdot z + 2\cdot y\cdot t$$

$$m1 = 2\cdot x\cdot y + 2\cdot z\cdot t$$

$$m4 = 1 - 2\cdot x\cdot x - 2\cdot z\cdot z$$

$$m7 = 2\cdot y\cdot z - 2\cdot x\cdot t$$

$$m2 = 2 \cdot x \cdot z - 2 \cdot y \cdot t$$

$$m5 = 2 \cdot y \cdot z + 2 \cdot x \cdot t$$

$$m8 = 1 - 2 \cdot x \cdot x - 2 \cdot y \cdot y$$

Da notare che questa trasformazione è molto conveniente dal punto di vista computazionale, poiché non utilizza funzioni come seno e coseno, ma sfrutta direttamente i parametri dei quaternioni. Questa matrice, così come è presentata, non è utilizzabile da OpenGL poiché la libreria grafica utilizza matrici 4 x 4, infatti richiede l'inserimento di una dimensione aggiuntiva per la traslazione.

Rotazione di un vettore

Fino ad ora i quaternioni sono stati mostrati come rotazioni di sistemi di riferimento, però possono anche andare a rappresentare anche altre entità matematiche, come i vettori tridimensionali. Facendo sempre riferimento a (4.1), questi possono essere resi come quaternioni con la parte reale uguale a 0, quindi:

$$0 + x \cdot i + y \cdot j + z \cdot k \tag{4.13}$$

Siccome è stato accennato in precedenza che un quaternione può essere visto come un vettore in $\mathbb{R}^4$ attraverso la quadrupla (t, x, y, z), similmente un vettore in $\mathbb{R}^3$ può essere mappato in $\mathbb{R}^4$ con la tupla $(0, x, y, z)$.
Attraverso i quaternioni rappresentanti una rotazione in un sistema di riferimento, è possibile far ruotare anche i vettori espressi nella forma suddetta.
E' dimostrabile che, dati due quaternioni che rappresentano rispettivamente una rotazione q_{rot} ed un vettore v, quest'ultimo viene ruotato nel vettore v_{rot} attraverso la formula

$$v_{rot} = q_{rot} \cdot v \cdot q_{rot}^{*} \tag{4.14}$$

ovvero si moltiplica il sistema di riferimento per il vettore dato e poi si moltiplica ancora per il coniugato del sistema di riferimento. Da questo prodotto si ottiene un quaternione della forma di un vettore come in (4.13) ruotato nel nuovo sistema di riferimento.

Considerazioni

Per memorizzare le rotazioni spaziali si potrebbero comunemente utilizzare delle matrici, poiché sono più semplici e intuitive dei quaternioni; questi ultimi infatti introducono la quarta dimensione (1 reale e 3 immaginarie) per mappare lo spazio tridimensionale, e sono governati da regole come (4.2) e (4.3) che risultano poco intuitive.
Il vantaggio di questa entità matematica risiede nel fatto che possiede meno variabili per indicare una posizione spaziale, infatti può essere rappresentata come un vettore quadridimensionale, una matrice invece deve mantenere 9 variabili al suo interno, con un maggior dispendio computazionale per le operazioni.
Altri aspetti non secondari sono legati alla piattaforma Arduino, infatti questa può calcolare direttamente via hardware i quaternioni come descritto in precedenza, quindi sarebbe inutile una trasformazione del quaternione in matrice.
Inoltre spedire, via porta seriale o via wireless, un vettore di 4 elementi rappresentante un quaternione risulta molto più vantaggioso rispetto ad una matrice di 9 parametri, quindi si ha un consistente risparmio di dati trasmessi.

Capitolo 5

Architettura del sistema proposto

In questo capitolo si fornisce una descrizione del software sviluppato per interagire con la piattaforma Arduino utilizzata, questo viene fatto per visualizzare le rotazioni eseguite con il giroscopio in dotazione. Dapprima verrà mostrata l'architettura in generale, per poi scendere nei dettagli dei singoli componenti, spiegando le scelte eseguite per la realizzazione del software presentato.
Siccome lo scopo di questo documento è di mostrare un software adatto allo studio di dati rotazionali, si vuole prediligere un approccio architetturale ed organizzativo del progetto presentato rispetto ad uno implementativo inerente alla codifica degli algoritmi e meccanismi visti in precedenza; quest'ultimo punto di vista infatti è più utile al fine di aggiornare o espandere il software presentato, rischiando però di perdere la visione d'insieme che si vuole dare in questo elaborato.

Requisiti

Requisiti funzionali

Il software prodotto deve acquisire i dati di rotazione da un apparecchio Arduino per poterli interpretare graficamente, poi deve salvare i dati prelevati allo scopo di poterli di nuovo ricostruire ed analizzare in un secondo momento senza l'ausilio del sensore.
Il dispositivo deve comunicare con il programma risiedente nel computer attraverso una porta seriale via USB.
Tutto il sistema deve essere comandato graficamente in modo da rendere più agevole il suo

utilizzo, inoltre non deve avere rallentamenti nella resa grafica degli spostamenti dell'apparecchio analizzato al fine di poter controllare anche le minime variazioni di rotazione.

E' necessario inoltre che il software sia in grado di supportare delle modifiche per quanto riguarda gli algoritmi utilizzati sia dagli sketch di Arduino che dal programma nel computer collegato al sensore; l'ipotetico utente quindi deve disporre di diversi componenti che si possono scambiare tra di loro allo scopo di utilizzare svariate combinazioni.

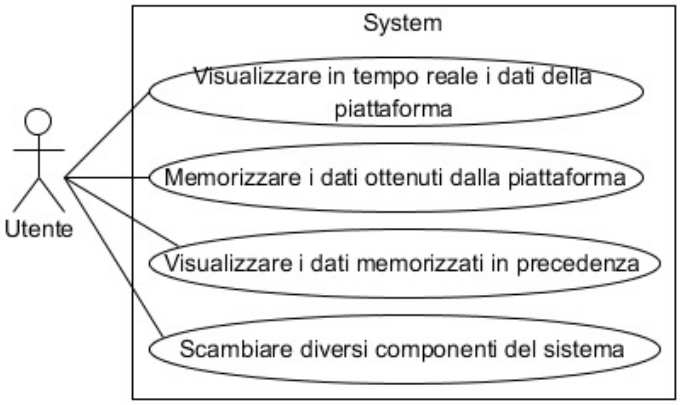

Figura 5.1

Questi requisiti possono essere resi graficamente con un sintetico diagramma di casi d'uso, rappresentato dalla figura 5.1.

Requisiti non funzionali

Un requisito implicito per il software creato è la flessibilità, ovvero la capacità di adattarsi a diverse variazioni a seconda dell'evoluzione dei risultati ottenuti (ad esempio se un prototipo non soddisfa le aspettative sperate, bisogna poterlo variare con pochi cambiamenti senza dover ricominciare tutto) e la riusabilità, ovvero più componenti possono sfruttare sottoparti già implementate per altri programmi con scopi simili.

Un'altra caratteristica che bisogna tenere in considerazione è la facilità di utilizzo non solo dal punto di vista grafico. Questo software infatti contiene al suo interno la logica dei quaternioni e dei sistemi di riferimento visti in precedenza, quindi per facilitare la loro corretta interpretazione bisogna fornire una interfaccia non solo facile da utilizzare, ma anche esaustiva nella comprensione dei movimenti esaminati.

Inoltre questo software deve gestire due diversi dispositivi hardware che devono collaborare, ovvero la piattaforma Arduino e il computer ad essa collegata, quindi bisogna stabilire una sequenza di utilizzo del software presente nei due diversi elaboratori in modo da poter

utilizzare in modo corretto l'architettura proposta. Naturalmente se questa sequenza non dovesse essere rispettata, i vari componenti non devono andare in errore ma gestire questa eventualità segnalando l'errato utilizzo oppure andando in stato di attesa per un uso corretto.
Altre caratteristiche e particolari tecnici sono stati visti durante la stesura del framework citato in precedenza, quindi molte altre caratteristiche sono state decise durante l'evoluzione dell'elaborato, e saranno discusse in seguito durante l'analisi nel capitolo.

Struttura generale del framework

Per la creazione di questo elaborato sono state fatte numerose prove e programmi intermedi allo scopo di ricavare informazioni utili principalmente per analizzare il comportamento del dispositivo Arduino, sia per quanto riguarda l'analisi delle formule viste nei capitoli precedenti, sia per ottenere una resa grafica dei dati raccolti dalla piattaforma.
Ci si vuole sbilanciare sull'esposizione dell'architettura indicando il C++ come linguaggio utilizzato per l'implementazione; questa scelta è stata determinata oltretutto dal fatto che i vari file sketch per la piattaforma Arduino sono scritti in un linguaggio simile al C++, quindi si è cercato di riutilizzare la logica dei moduli creati.
Data quindi l'eterogeneità del software da creare e considerando anche gli sviluppi futuri di questo progetto, è stata presa la decisione di creare un framework di elementi generici da cui attingere per poter creare velocemente delle demo da analizzare e per permettere una gestione più flessibile del codice.
Il framework si può dividere in due parti principali, ovvero la parte relativa alla piattaforma Arduino e la parte più corposa riferita alle applicazioni lato computer.
Siccome le applicazioni create sono per la maggior parte grafiche o prevedono un intenso uso di widget per comandare varie fasi o parametri delle demo sviluppate, si è deciso di creare il framework in modo che spaziasse su un pattern architetturale tipico delle applicazioni grafiche, ovvero il Model-View-Presenter (semplificato con MVP).

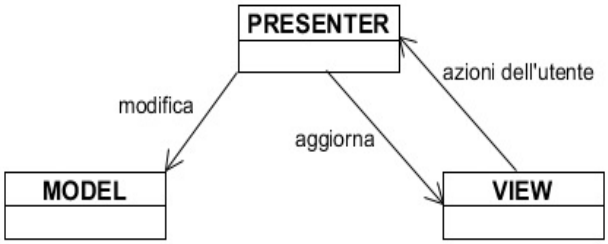

Figura 5.2

Ricordiamo brevemente che il Model rappresenta la parte che contiene la logica in un programma, ovvero le parti contenenti i dati che poi saranno interpretati visivamente; la View è l'interfaccia grafica utilizzata dall'utente ed il Presenter è il componente che collega Model e View. Questa rappresentazione è resa dall'immagine sovrastante.

In seguito verranno indicati con “model”, “view” e “presenter” i moduli che corrispondono ad una delle relative categorie del pattern architetturale appena introdotto.

La suddivisione nell'architettura presentata non è solo logica, ma anche fisica, ovvero il codice e tutti gli artefatti creati sono stati divisi in sottocartelle come mostrato nell'immagine 5.3.

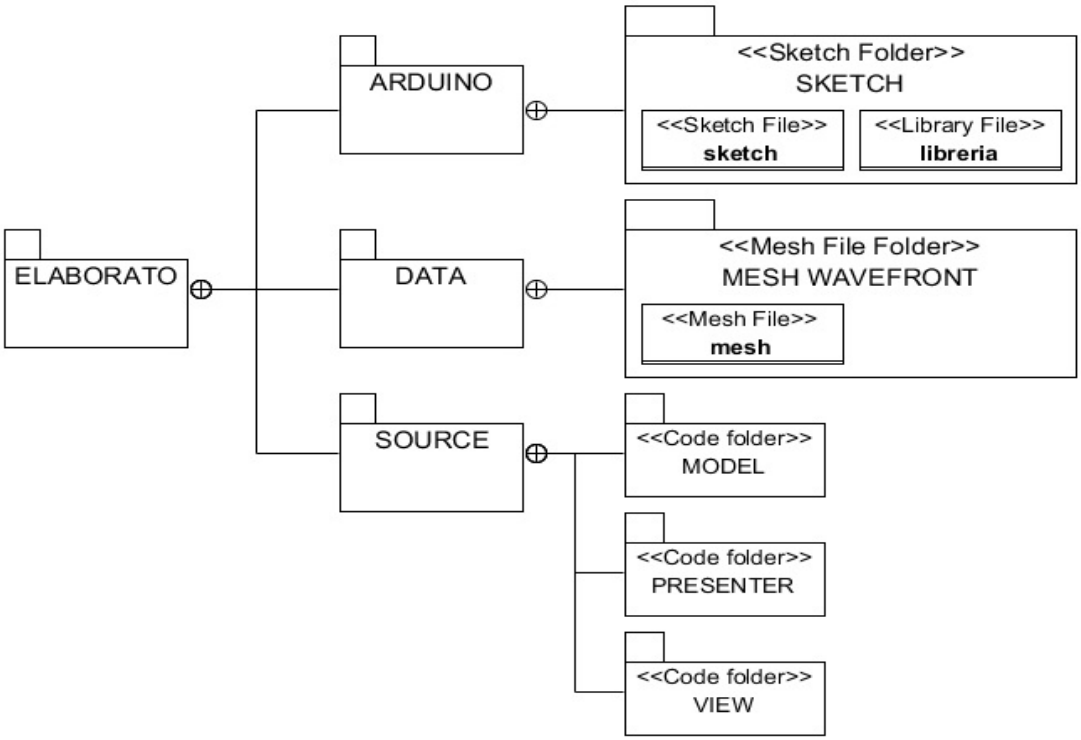

Figura 5.3

La disposizione delle cartelle dedicate al MVP è a sua volta suddivisa in un oggetti con la medesima struttura, come evidenziato dall'immagine 5.4; questo è stato fatto per rendere il codice più facile da gestire. Si può vedere che sono presenti per ogni oggetto delle cartelle contenenti il codice di implementazione, l'interfaccia utilizzata, le funzioni di callback e le variabili pubbliche note agli altri oggetti al fine di impostare metodi e opzioni particolari. Sono presenti inoltre le variabili private non modificabili dagli oggetti esterni, come ad esempio le impostazioni dell'interfaccia grafica, ed infine è compresa la struttura dei campi interni manipolata dai file di implementazione; questo viene fatto con lo scopo di alleggerire il carico di complessità inserito nel file di implementazione e per nascondere i campi dell'oggetto dalla loro dichiarazione nell'interfaccia.
Oltre a queste cartelle, ne è presente un'altra riportante i test eseguiti per l'oggetto, utili per determinare e testare le funzionalità implementate.

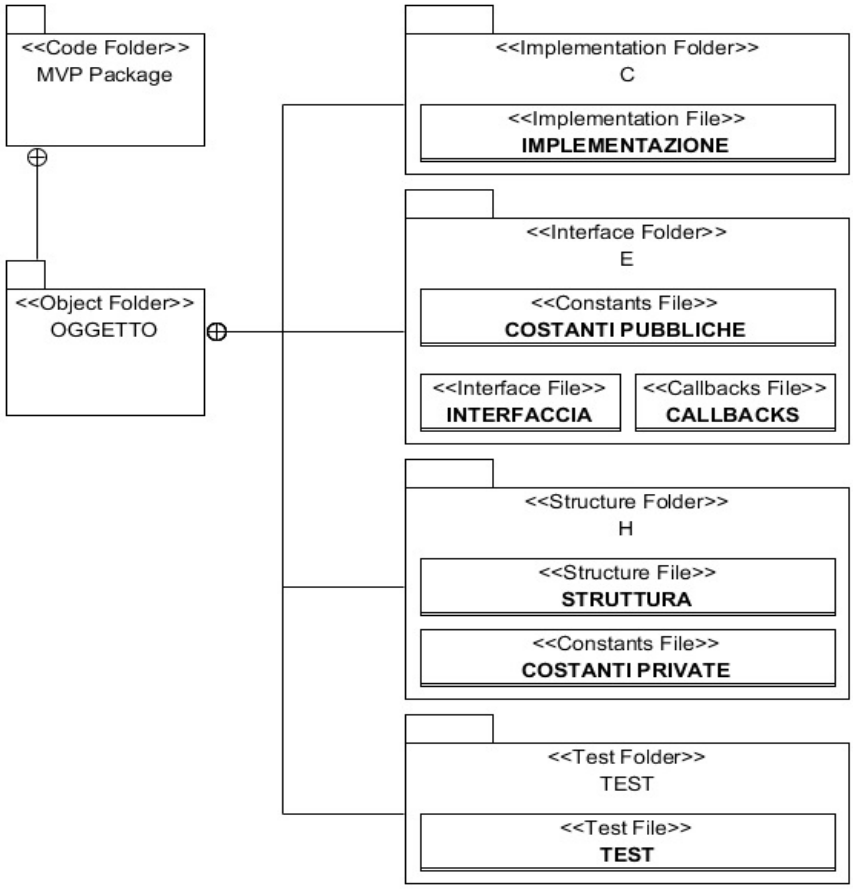

Figura 5.4

Tutti i file rappresentati in questo diagramma sono a basso indice di accoppiamento, questo per facilitare la loro sostituzione oppure il loro aggiornamento senza modificare il resto del codice prodotto; la loro dipendenza può essere resa con il diagramma seguente.

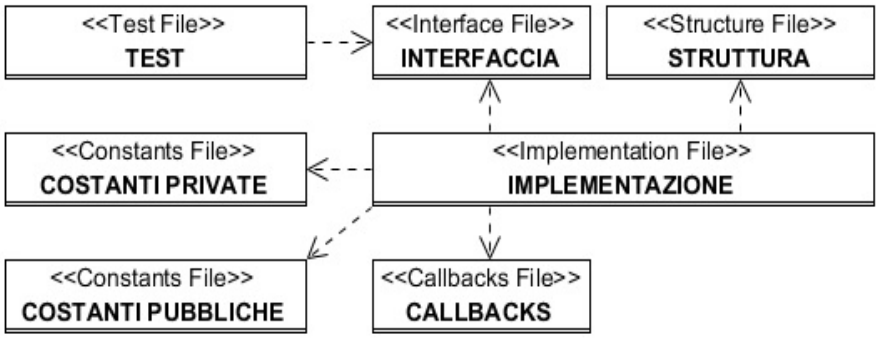

Figura 5.5

In questa immagine si può vedere che se un elemento viene modificato, il peso della modifica ricade per la maggior parte sul modulo di implementazione che quindi si deve adattare al cambiamento fatto; gli altri moduli al massimo hanno leggere modifiche di peso trascurabile. Il file di test è l'unico che dipende solo dall'interfaccia.
Fino ad adesso è stata presentata la struttura generale e la disposizione fisica di quanto prodotto, in seguito verranno approfondite le parti relative al software sviluppato per Arduino e per computer per poi approfondire la collaborazione tra queste due parti.

Struttura degli sketch

Nella parte dedicata ad Arduino sono inseriti gli sketch utilizzati e le relative librerie di supporto per il calcolo dei quaternioni. Queste, pur condividendone la logica, non sono condivise con le librerie corrispettive dei programmi creati per computer, infatti queste ultime non prevedono le limitazioni richieste per poter essere collegate con il linguaggio Wiring, come accennato nei capitoli precedenti.
Tutti gli sketch proposti hanno il compito di analizzare le velocità angolari del sensore Arduino per ricavare dei quaternioni da inviare attraverso la connessione seriale al programma collegato. La struttura di questi sketch è molto semplice e si può riassumere con il

seguente diagramma a stati.

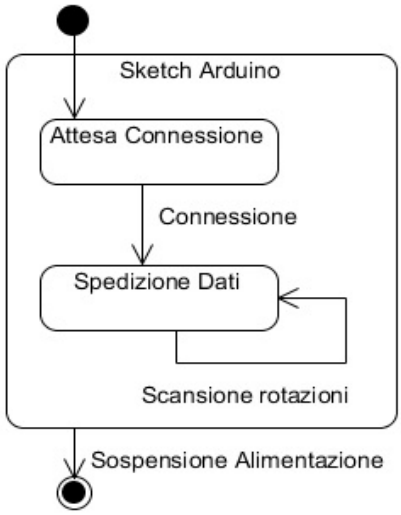

Figura 5.6

Nel diagramma si può vedere che lo sketch, una volta caricato sulla piattaforma, attende un segnale di connessione spedito dal programma collegato. Una volta ricevuto il segnale, incomincia a scansionare il giroscopio in modo da ottenere rotazioni da spedire. Siccome non è previsto che il dispositivo venga comandato dall'esterno ma deve fornire solo dati da analizzare, la terminazione avviene solo con la sospensione dell'alimentazione.
Questo tipo di comportamento cerca di emulare il suo reale utilizzo, ovvero imita un apparecchio montato su auto che una volta acceso, scansiona di continuo la posizione attuale fino a che il veicolo non si ferma, provocando così anche lo spegnimento del dispositivo.
Nel software presentato sono presenti due tipi di sketch, uno che utilizza i dati del DMP per ottenere i quaternioni di rotazione, mentre un altro che non utilizza questo componente, ma solo le velocità angolari ricavate dalla MPU. Entrambi questi algoritmi si comportano in maniera differente; l'analisi dei risultati sarà fatta nel capitolo dedicato.

Struttura MVP

Quasi tutte le applicazioni intermedie costruite hanno una struttura MVP, quindi possono attingere dal framework creato con widget specifici (comprendenti visualizzatori parametrici di quaternioni o pulsantiere che riportano la connessione tipica con una porta seriale) oppure

modelli di dati riutilizzabili presenti in quasi tutti i programmi (ad esempio i quaternioni ed i vari sistemi di riferimento descritti nei capitoli precedenti).
Sono inseriti inoltre diversi presenter che sfruttano model e view per applicazioni che andranno a comporre l'architettura utilizzata.
Nella categoria Model sono inseriti tutti i moduli che riguardano la logica di business dei vari programmi. Data la loro natura, questi tipi di moduli devono essere più astratti possibile per permettere un loro facile riuso in diverse applicazioni; non troviamo nessuna gestione di grafica al loro interno.
Nella categoria View sono inserite tutte le componenti grafiche che sono utilizzate nei vari progetti, quindi si possono accomunare in questa categoria dei widget specifici per particolari strutture dati (come ad esempio un visualizzatore dei parametri dei quaternioni) oppure oggetti della libreria grafica OpenGL. Lo scopo di questa categoria è fornire un'astrazione delle librerie grafiche utilizzate, in modo da relegare i loro dettagli e peculiarità a questi componenti software. Nel corso della stesura sono state analizzate diverse librerie tra cui ricordiamo le Qt, le Glui e le WxWidgets; le librerie prese in esame sono accomunate dal fatto di poter supportare o meno le OpenGL, necessarie per poter gestire la grafica tridimensionale, utilizzata per riportare a video il movimento nello spazio della piattaforma Arduino. Si ricordi infatti che le OpenGL sono sprovviste di widget per il loro utilizzo su schermo, quindi è necessario appoggiarsi a librerie esterne che le supportino.
La scelta è ricaduta sulle librerie FLTK (fast light tool kit) per la loro semplicità di utilizzo e per la loro facile integrazione con i più noti IDE sul mercato (questi ultimi infatti gestiscono un ruolo importante per la stesura ed organizzazione di un progetto complesso ed articolato come quello presentato).
Come mostrato nella figura 5.2, i componenti software che fanno parte rispettivamente della categoria Model e View sono slegati tra di loro, quindi le loro interfacce software non sono direttamente compatibili e quindi non utilizzabili a meno che non si usi un modulo di raccordo. Detto questo è intuibile che le categorie Model e View hanno bisogno di moduli software che siano in grado di convertire le informazioni date in input dalle interfacce grafiche in variazioni di stato per gli oggetti forniti dalla categoria Model; inoltre questi moduli di raccordo devono essere in grado di interpretare le variazioni generate per i model in informazioni da visualizzare con i componenti view forniti dal framework. A questo scopo

sono inseriti i presenter, ovvero dei componenti della categoria Presenter del pattern MVP, questi sono in grado di far cooperare i model e view e di scambiare le informazioni tra di loro. Si può passare adesso ad una descrizione più dettagliata dei componenti software creati e le loro peculiarità. Dato il basso accoppiamento che hanno gli oggetti delle categorie Model e View, si è deciso a descrivere discorsivamente questi elementi e le caratteristiche che li contraddistinguono, per poi affrontare con più dettaglio parti più complesse come i presenter.

Componenti model

I moduli che fanno parte di questa categoria trattano calcoli matematici o configurazioni del sistema operativo su cui si basano i programmi che li utilizzano. Di seguito vengono elencati i moduli principali (i sottomoduli si considerano accorpati nella loro descrizione).

Quaternion: questo modulo contiene l'oggetto quaternione ed i metodi per poterlo gestire e modificare come descritto nel capitolo dedicato; inoltre sono presenti anche le varie versioni dei sistemi di riferimento presentanti precedentemente (angoli di Eulero, di Tait-Bryan e naturalmente la versione utilizzata dalla piattaforma Arduino) e le funzioni per passare da angoli di rotazioni a quaternioni e viceversa. Sono aggiunte inoltre le conversioni da quaternione a matrice a 4 dimensioni secondo lo standard OpenGL.
In questa sezione è compresa anche una struttura dati affine al quaternione, denominata quaternion_struct, che racchiude le informazioni basilari comunicate dalla piattaforma Arduino verso i componenti software. In questa struttura sono compresi i dati del quaternione di rotazione complessiva da quando il dispositivo è utilizzato e l'informazione in millisecondi di quando questa rotazione è stata generata.

SerialPortConnection: questo modulo gestisce la connessione con la porta seriale tramite la quale il programma su computer comunica con la piattaforma Arduino. Siccome per utilizzare la porta seriale bisogna comunicare con il sistema operativo (in questo caso è stata utilizzata una versione del SO Windows) per richiedere i tipi di porta disponibili ed allocare le risorse necessarie, si è vista la necessità di utilizzare una variante del linguaggio C/C++ utilizzata dalla Microsoft nei suoi sistemi operativi, ovvero il Common Language Infrastructure

(infrastruttura del linguaggio comune). Il CLI non è altro che un linguaggio specifico nato per utilizzare l'ambiente di esecuzione Windows.
Attraverso un modulo in C/CLI è possibile accedere in lettura e scrittura ad una porta seriale. Naturalmente, essendo un linguaggio specializzato, è stata creata una interfaccia in C/C++ utilizzabile da altri componenti in modo da racchiudere tutta la complessità del CLI all'interno del modulo di implementazione dell'oggetto SerialPortConnection.

SharedMem: in questo modulo sono racchiuse alcune delle funzionalità delle librerie Boost, in particolare i metodi per poter chiedere al sistema operativo di allocare una parte di memoria in RAM da condividere poi con diversi programmi sia in lettura che in scrittura.
Le librerie Boost sono delle librerie multi-piattaforma open-source che servono ad espandere le funzionalità del linguaggio C++. Queste librerie spaziano su diversi campi di applicazione, riguardanti la gestione del file system, del multithreading, della comunicazione con il sistema operativo; inoltre si occupa della manipolazione di immagini e cosi via.
In questo caso specifico, le librerie sono utilizzate per creare una memoria condivisa tra i vari programmi e gestire la concorrenza all'accesso della risorsa allocata; vengono quindi forniti dei ruoli di scrittura o di lettura per i processi che richiedono l'utilizzo.
Questa memoria condivisa è utilizzata per mantenere i dati prelevati da piattaforma Arduino dal programma addetto alla connessione con la porta seriale.
Questi dati sono rappresentati da un vettore di quaternion_struct visti precedentemente, i quali vengono letti da un altro programma che li visualizza con modelli OpenGL che si muovono su schermo. Possiamo accennare che la distinzione di due programmi (uno per la scrittura dei dati in memoria ed uno per l'interpretazione grafica) è necessaria a causa della differenza di velocità di lettura e di scrittura, infatti il programma che scrive i dati sulla memoria condivisa (quello addetto alla porta seriale) è più veloce rispetto al programma che gestisce la grafica, infatti deve gestire il calcolo della posizione delle mesh poligonali rispetto alle nuove rotazioni date dai quaternioni. Inevitabilmente quindi non vengono visualizzati tutti i dati, ma solo una parte. Questo fenomeno, alla fine dell'analisi di movimento della piattaforma Arduino, non provoca alcun problema, infatti la successione di immagini ottenuta risulta fluida e continua, indice che le rotazioni ottenute da porta seriale sono così vicine tra di loro da risultare impercettibili se qualcuna di esse non viene resa graficamente.

Timer: questa libreria viene utilizzata per richiedere il tempo corrente al sistema operativo; serve per creare un cronometro utile per le misurazioni degli esperimenti svolti con un dispositivo in movimento su di un'auto. Questo tipo di esperimento è stato utile per vedere il comportamento della piattaforma Arduino con delle sollecitazioni che differiscono da gesti o movimento che una persona può direttamente imprimere al dispositivo.

Componenti view

Come detto in precedenza, i moduli view racchiudono tutta la complessità derivante dall'utilizzo delle librerie grafiche FLTK e OpenGL, fornendo quindi interfacce in C/C++ libere da qualsiasi contesto o dipendenza. E' possibile quindi utilizzare widget disposti diversamente oppure totalmente differenti all'interno dei moduli, basta che rispettino le convenzioni imposte dalle interfacce create. E' possibile quindi, con lo stesso principio, sostituire anche le librerie grafiche per provare il comportamento e le prestazioni di altre.
Questi tipi di moduli sono i meno complessi da costruire, ma non meno importanti poiché sono lo strumento che permette all'utente di interagire col programma.
Ecco di seguito una breve presentazione degli elementi creati per questa categoria:

QuaternionDisplay: widget creato allo scopo di visualizzare i parametri di un quaternione come descritto nel capitolo dedicato. Serve unicamente come output grafico.

SerialPortWidget: questo componente rappresenta l'interfaccia utente per gestire la porta seriale per il collegamento con la piattaforma Arduino. Oltre ad una funzione di connessione e disconnessione espletata attraverso un bottone, a questo oggetto viene affidato anche il compito di gestire tutti i parametri per impostare una comunicazione. Quindi tra le tante opzioni per iniziare un collegamento con la piattaforma Arduino si prendono in considerazione solo alcuni fattori, quali il nome della porta seriale (indicata con la sigla COM e seguita dal numero della porta) ed il baud rate (ovvero la velocità di trasmissione dei caratteri). Altri parametri come il tipo di hand-shake o il controllo degli errori di trasmissione non sono presi in considerazione dall'interfaccia grafica.

Slider3D: questo widget fornisce tre slider che rappresentano il variare dei valori di 3 angoli in un sistema di riferimento tridimensionale. Rispetto a quanto detto nei capitoli precedenti, possono rappresentare rotazioni di qualsiasi sistema di riferimento (Yaw, Pitch e Roll o Psi, Theta e Phi, oppure rotazioni intorno ad assi X, Y e Z) poiché il loro utilizzo non è legato a nessun sistema di riferimento specifico. Questo componente è utilizzato per le simulazioni di rotazione senza piattaforma Arduino per controllare il diverso comportamento dei sistemi di riferimento analizzati.

ChoronometerDisplay: interfaccia per l'utilizzo di un cronometro utilizzato per ottenere le tempistiche di percorrenza sui percorsi stradali eseguiti in auto per testare la piattaforma Arduino. Questa interfaccia viene utilizzata per visualizzare lo stato di attività del cronometro, il tempo di percorrenza dalla sua accensione ed il tempo dell'ultimo percorso registrato. Attraverso questo modulo si può indicare oltretutto il file ove salvare i tempi memorizzati durante gli esperimenti. La descrizione del programma che utilizza questa interfaccia grafica sarà ripresa nella sezione dei presenter.

OpenGLFrame: componente grafico per la gestione della libreria OpenGL. Come detto in precedenza non tutte le librerie grafiche supportano l'utilizzo di OpenGL; questa infatti permette di comunicare direttamente con la scheda grafica per poter allocare e riutilizzare sia mesh poligonali, sia intere procedure e manipolazioni di dati utilizzate durante lo svolgimento del programma che le utilizza. Dato che si analizza il comportamento rotazionale di un dispositivo Arduino, è necessario emularne il comportamento su schermo attraverso un programma in resa tridimensionale, quindi bisogna impostare tutte le caratteristiche dell'ambiente necessarie al rendering 3D (luci, posizione dell'osservatore, colore di background, piramide di vista, …). Tutti questi parametri sono inseriti in questo componente, quindi tutta la complessità delle impostazioni iniziali di OpenGL sono concentrate in questo modulo, il quale necessita solo di definire una mesh da visualizzare. Le sole azioni non definite ma lasciate all'utente riguardano la rotazione attraverso la manipolazione della mesh resa.

MeshReader: questo componente serve ad estrarre una mesh poligonale da tipi di file di modellazione grafica.
OpenGL, al contrario di altre librerie che accedono alla scheda grafica come DirectX, non ha funzioni per leggere e convertire le mesh da file esterni; quindi si è vista la necessità di creare un convertitore di file generati da programmi di grafica vettoriale. Il formato di file utilizzato per racchiudere la mesh è il wavefront, noto per avere l'estensione obj per la mesh e mtl per indicare i materiali per le varie parti che compongono il modello grafico.
Il componente software preso in esame quindi legge uno o più file del formato wavefront e restituisce una rappresentazione in OpenGL della mesh desiderata; questo output non è altro che un programma di gestione di grafica 3D che può essere salvato e riutilizzato nella scheda grafica come accennato in precedenza. Questo modulo inoltre definisce l'accuratezza della resa della mesh, questa infatti può essere renderizzata come un insieme di poligoni (flat shading) oppure, aggiungendo le normali per ogni vertice si possono ottenere forme più armoniche (Gouraud shading).

Material: in OpenGL si possono definire i materiali utilizzabili in OpenGL per la resa tridimensionale, questi differiscono dai colori RGB (sigla del sistema di codici di colori internazionali ed acronimo di “red, green, blue”) poiché forniscono informazioni non solo sul colore che la superficie di poligoni trattata deve assumere, ma anche sulla riflessione diffusa e speculare che il materiale emulato deve avere (in pratica descrivono come deve comportarsi un raggio di luce sulla superficie colpita, ovvero se viene riflesso in una direzione specifica oppure viene diffuso nell'ambiente a causa dell'opacità del materiale illuminato).
In questo componente sono presenti diversi tipi di materiale predefiniti, infatti OpenGL non fornisce materiali pronti per l'uso ma solo le funzioni per poterli utilizzare.

ICar: interfaccia C++ che definisce i metodi che deve avere un oggetto al fine di rendere una mesh di un'automobile in OpenGL. Questo non è un componente grafico, infatti è una classe astratta che serve per definire le caratteristiche di base di una classe implementante.
In questa interfaccia quindi vengono specificate informazioni per quanto riguarda la resa finale dell'auto da renderizzare; questo è utile poiché serve a nascondere le implementazioni peculiari di ogni veicolo presentato, come ad esempio i percorsi dei file wavefront da cui

attingere per le mesh o i materiali utilizzati per le varie parti dell'auto.

Honda e MiniCooper: questi due sono modelli di auto implementanti l'interfaccia ICar, quindi forniscono una resa di due tipi diverse di auto, rispettivamente una Honda Civic ed una Mini Cooper. Possono essere utilizzate indifferentemente poiché possiedono la stessa interfaccia di utilizzo. Anche se non sono presenti programmi che utilizzano due modelli differenti di auto, questi sono stati creati in modo da fornire delle primitive per eventuali applicazioni che prevedano l'uso di più veicoli, come ad esempio un analizzatore della dinamica degli incidenti.

Gyroscope: questo oggetto fornisce una mesh aggiuntiva da aggiungere alla scena, infatti rappresenta il sistema di riferimento solidale all'oggetto ruotato (in questo caso all'auto posizionata al suo interno) come descritto nei capitoli precedenti. La mesh di questo oggetto è formata da 3 corone circolari disposte perpendicolarmente le une con le altre in modo da rappresentare le rotazioni di yaw, pitch e roll del veicolo inserito all'interno. Questo modulo è oltremodo utile poiché si fornisce immediatamente una visione intuitiva di come si muove il sistema di riferimento dell'auto rispetto a quello fisso (quest'ultimo paragonabile al sistema di riferimento di OpenGL) attraverso l'uso diretto del dispositivo Arduino.

Nella categoria View sono presenti altri componenti che non sono stati inseriti nella lista appena elencata poiché riguardano delle sottocategorie degli elementi appena indicati (ovvero widget intermedi creati per semplificare la progettazione del componente complessivo), oppure perché non utilizzati direttamente per analizzare i dati della piattaforma Arduino. Questi infatti sono stati usati al fine di ottenere particolari impostazioni dei moduli appena descritti; in particolare ci si riferisce al modulo MeshViewer, utilizzato per impostare le mesh dei due veicoli in modo che questi abbiano entrambi le stesse dimensioni e gli stessi requisiti richiesti dall'interfaccia ICar.

Componenti presenter

Come detto prima, i presenter sono componenti che utilizzano elementi della categoria View e

Model per dar vita a programmi autonomi da utilizzare. Questi presenter sono utilizzati in modo autonomo oppure in maniera collaborativa tra di loro, nello specifico collaborano per leggere e scrivere dati da una stessa porzione di memoria. Quest'ultima categoria è formata da componenti che non sono accomunati in un singolo thread, appunto per venire incontro ad un fenomeno emerso durante la costruzione dei prototipi ed anticipato nel sottoparagrafo riguardante il modulo SharedMem: la velocità con cui arrivano i dati dalla piattaforma è superiore alla velocità di resa dei dati stessi in grafica tridimensionale. Questo comporta degli inconvenienti che sono evitabili considerando i presenter come componenti eseguibili in parallelo e che agiscono su una zona comune di memoria come esemplificato nell'immagine sottostante.

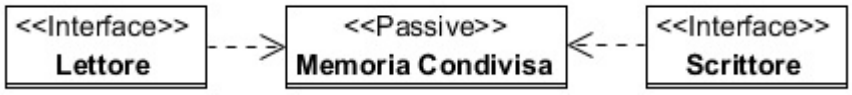

Figura 5.7

Nel nostro caso, i presenter si posso differenziare come lettori o scrittori a seconda se il loro scopo è ottenere dati da una fonte esterna (piattaforma Arduino o file di salvataggio) per scriverli sulla memoria condivisa oppure leggere i dati al fine di interpretarli.

La memoria condivisa perciò viene utilizzata come zona in cui leggere e scrivere in modalità atomica, in questo modo un dato nella memoria può essere sovrascritto con un altro appena arrivato senza che il precedente venga letto. In questa maniera tutte le applicazioni possono leggere o scrivere a diverse velocità; l'unico inconveniente è che il lettore alla fine legge solo una parte dei dati. Nel programma presentato il processo lettore non è altro che un visualizzatore di modelli 3D che interpreta il movimento della piattaforma Arduino, sperimentalmente si è visto che l'immagine risulta fluida e continua anche senza rappresentare tutti i dati, appunto perché le variazioni di rotazione sono talmente piccole da risultare impercettibili alla vista.

I presenter che sono considerati lettori o scrittori sono gestiti da più thread in un singolo processo; tuttavia la memoria condivisa non è interna allo spazio adibito al processo stesso, ma è allocata esternamente attraverso la libreria SharedMem, la quale richiede al sistema operativo sottostante uno spazio di memoria condiviso tra i vari processi. Questa caratteristica

soddisfa uno dei requisiti funzionali esplicitati all'inizio, ovvero la sostituzione di diversi componenti con degli altri durante l'uso del software, questo per permettere di provare altri programmi in lettura e scrittura; questi infatti possono accedere alla memoria condivisa senza dover creare una applicazione ex novo.
Da quanto appena detto si può capire che la memoria condivisa può essere utilizzata non solo dai singoli thread di un processo, ma anche da più processi contemporaneamente.
Nel seguito del paragrafo vengono mostrati i presenter creati per questo progetto con l'ausilio di diagrammi che indicano la loro composizione rispetto agli elementi già presentati ed il loro meccanismo interno.

Chronometer: questo programma unisce i moduli Timer e ChoronometerDisplay visti in precedenza per creare un cronometro. La figura seguente mostra le relazioni tra questi moduli.

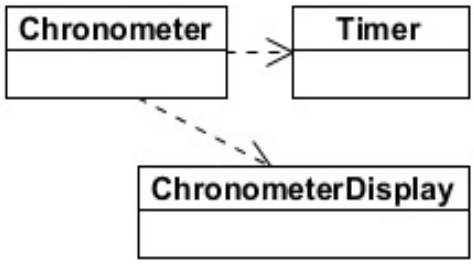

Figura 5.8

Come già accennato, il cronometro viene utilizzato per misurare il tempo di particolari percorsi eseguiti in auto, questo allo scopo di controllare il comportamento della piattaforma Arduino in diverse circostanze, non solo basate su semplici movimenti manuali di prova, ma anche su percorsi stradali. I dati ottenuti da queste prove saranno discusse nel capitolo dei risultati ottenuti.
In figura 5.9 viene descritto l'automa che governa il cronometro in un diagramma rappresentante i vari stadi del programma in esecuzione; come si può intuire, il cronometro incomincia da fermo per poi passare in stato di attività facendo partire il timer interno. Una volta attivato, si passa in fase di registrazione quando si desidera memorizzare uno specifico tempo ottenuto, per poi ritornare allo stato di attività. Infine, prima di fermare il cronometro, vengono memorizzati su file i tempi registrati per poterli analizzare in seguito.

Questo componente non si combina con altri presenter e non è utilizzato direttamente con la piattaforma Arduino.

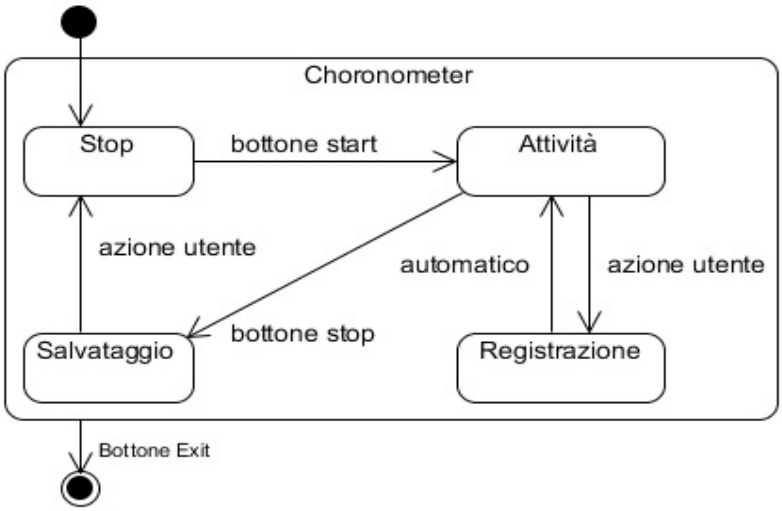

Figura 5.9

RotationDemo: questo programma è stato creato con lo scopo di mostrare le differenze tra sistema di riferimento degli angoli di Eulero e quelli di Tait-Bryan.

Per quanto riguarda la resa grafica OpenGL, questa demo compone l'OpenGLFrame, la mesh del giroscopio ed un'auto; sono presenti poi un visualizzatore di quaternioni ed uno slider3D per i vari widget descritti in precedenza; per quanto riguarda la categoria del Model, vengono utilizzati i quaternioni e le funzioni per gestire le rotazioni secondo i vari sistemi di riferimento. Le dipendenze appena descritte sono raffigurate nel diagramma della figura 5.10, mentre nel gruppo di immagini 5.11 viene rappresentato il comportamento di questa demo come un automa.

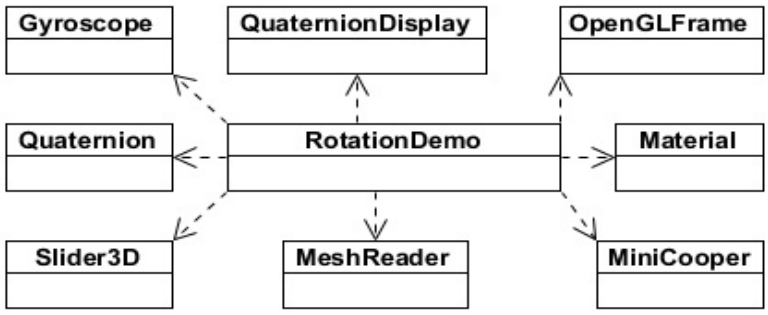

Figura 5.10

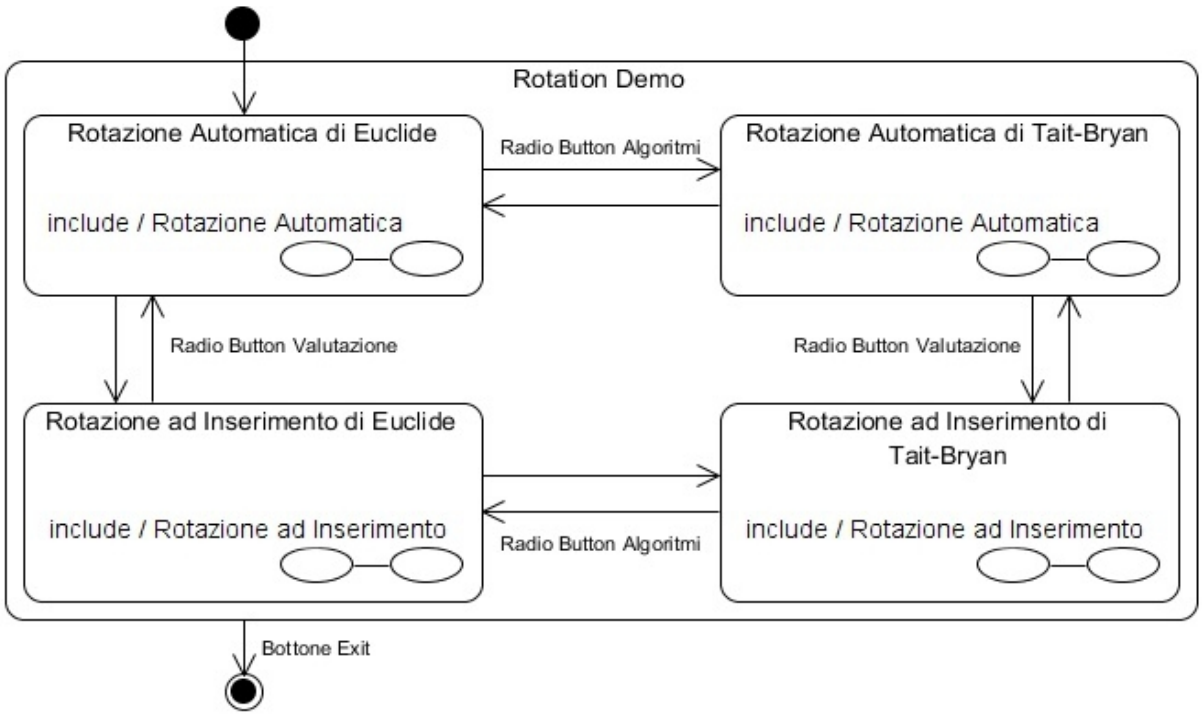

Figura 5.11a

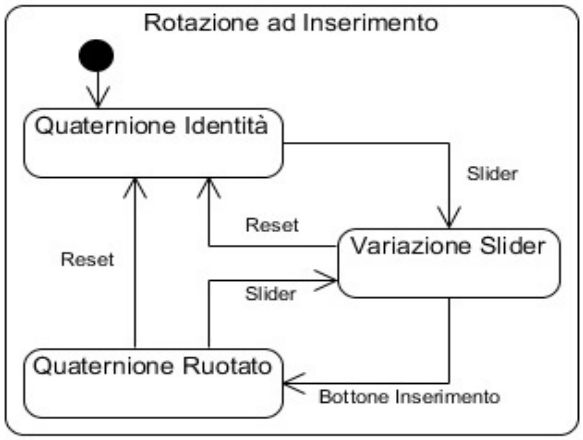

Figura 5.11b

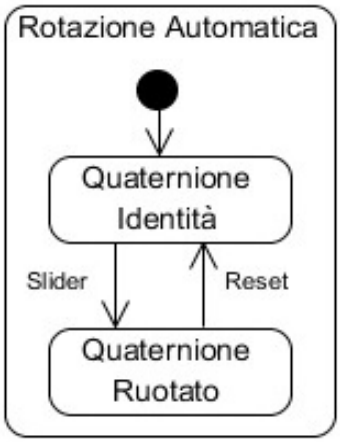

Figura 5.11c

Nello schema proposto dalle figure 5.11 si può vedere il comportamento ripetitivo che ha questo automa; la figura 5.11a infatti rappresenta le quattro possibili combinazioni tra angoli di Euclide e di Tait-Bryan e la scelta di interpretare le rotazioni quando viene subito utilizzato lo slider3D oppure dopo essersi posizionati su una rotazione precisa. Dall'immagine si può vedere che la scelta è stata fatta con dei radio button, il cui azionamento fa transitare da un algoritmo all'altro.

All'interno di ognuna di queste combinazioni, viene richiamato un automa specifico a seconda se si vuole interpretare istantaneamente il movimento dello slider (figura 5.11c) oppure aspettare la sua valutazione con la pressione di un pulsante apposito (figura 5.11b). In ogni caso è possibile ritornare ad un punto di partenza di uno degli automi secondari premendo un pulsante di reset che sostituisce il quaternione di rotazione fino ad adesso rappresentato con un quaternione d'identità.

Si rimane in uno degli automi secondari fino a quando l'automa primario non modifica il suo stato; è da notare che gli automi secondari indicano solo come si comportano i vari algoritmi ma non specificano nessun sistema di riferimento in particolare. Questo viene fatto dagli stati dell'automa principale, i quali specificano se utilizzare il sistema di riferimento di Tait-Bryan oppure di Euclide.

Nella figura 5.12 viene mostrata, per una maggior comprensione dell'applicazione finale, l'interfaccia grafica creata per questa demo.

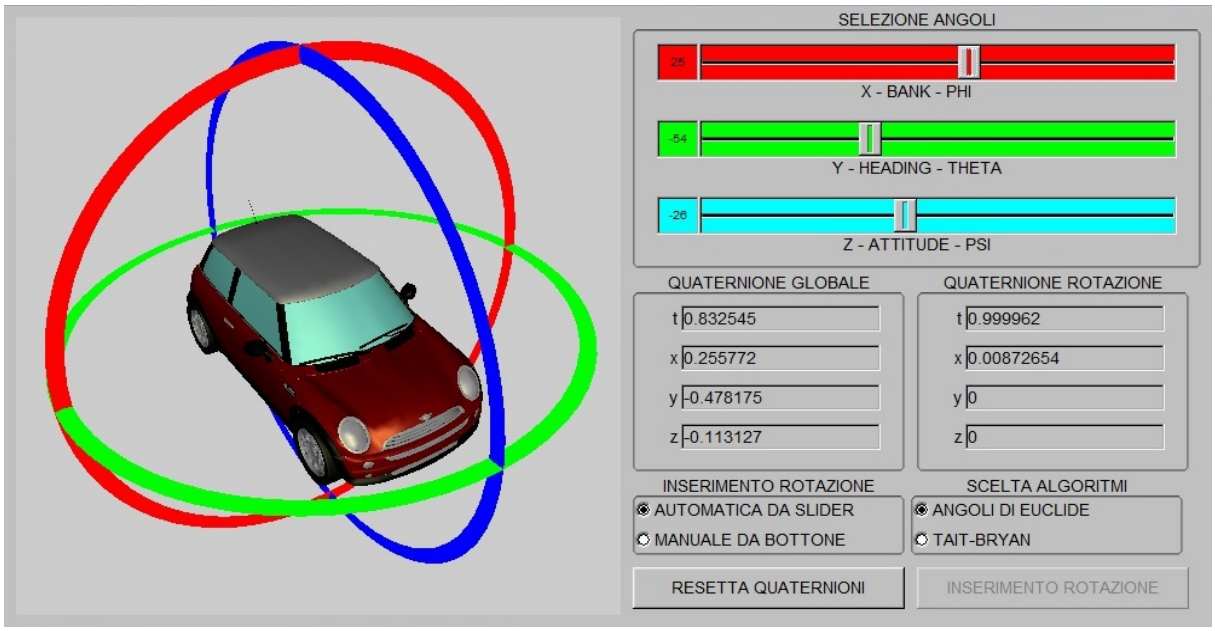

Figura 5.12

In questa immagine si possono scorgere gli elementi grafici presenti nel diagramma dell'immagine 5.10, ovvero l'auto rappresentante la posizione del sensore Arduino ed il giroscopio che la racchiude, poi lo Slider3D per lo spostamento del sistema di riferimento

solidale al dispositivo; si può intuire che ogni slider presente modifica la rotazione di uno degli anelli del giroscopio. Infine si possono notare i QuaternionDisplay utilizzati per mostrare il quaternione di rotazione rispetto al sistema di riferimento fisso e la rotazione di spostamento eseguita per modificare il sistema di riferimento solidale all'auto.
Sono presenti inoltre i radio button che danno vita all'automa presentato nell'immagine 5.11a ed i bottoni per resettare il quaternione di rotazione e per inserire il valore degli slider; questi elementi sono raffigurati negli automi secondari.
Questa è una demo puramente di analisi dei sistemi di riferimento e non è collegabile con la piattaforma Arduino, infatti serve per analizzare i comportamenti delle formule delle interpretazioni degli angoli come descritto nel capitolo dedicato. In questo software è stata inserita la possibilità di interpretare le rotazioni subito quando si utilizzano gli slider degli angoli, oppure dopo aver posizionato gli slider in una particolare posizione: nel primo caso avremo solo una rotazione in un asse, appunto perché non è possibile utilizzare più di uno slider per volta; da qui noteremo che i sistemi di Eulero e di Tait-Bryan non comportano differenze. Se si inseriscono invece i valori delle rotazioni prima dell'interpretazione, si può notare che i due sistemi appena citati si comportano in modo differente sulla precedenza degli angoli di yaw e pitch, come descritto nel capitolo dedicato. Ne consegue che le formule per l'inserimento ed estrazione di rotazioni in un quaternione devono essere utilizzate in modo da rendere coerente la trasformazione fatta.
Anche questo programma, puramente dimostrativo, non si combina con altri presenter mostrati.

SerialPortPresenter: questo presenter ha il compito di connettersi al dispositivo Arduino attraverso la porta seriale. Questo componente inoltre legge i dati spediti dallo sketch in funzione sulla piattaforma; nella versione definitiva vengono letti i quaternioni di rotazione calcolati dall'apparecchio. Nell'immagine seguente è presente il diagramma che descrive la sua interazione con i componenti della categoria Model e View.

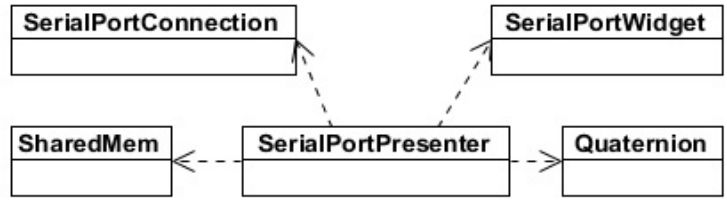

Figura 5.13

Questo presenter utilizza come interfaccia grafica il SerialPortWidget presentato prima, attraverso il quale vengono scelti dei parametri di connessione con la porta seriale; inoltre è l'unico presenter ad utilizzare la logica del SerialPortConnection per interagire con il sistema operativo per utilizzare la porta seriale via USB.

Questo componente comprende il modulo SharedMem, utilizzato per allocare memoria allo scopo di scrivere i quaternioni ottenuti dai dati spediti dalla piattaforma Arduino.

Il funzionamento del SerialPortController è descritto nell'automa seguente:

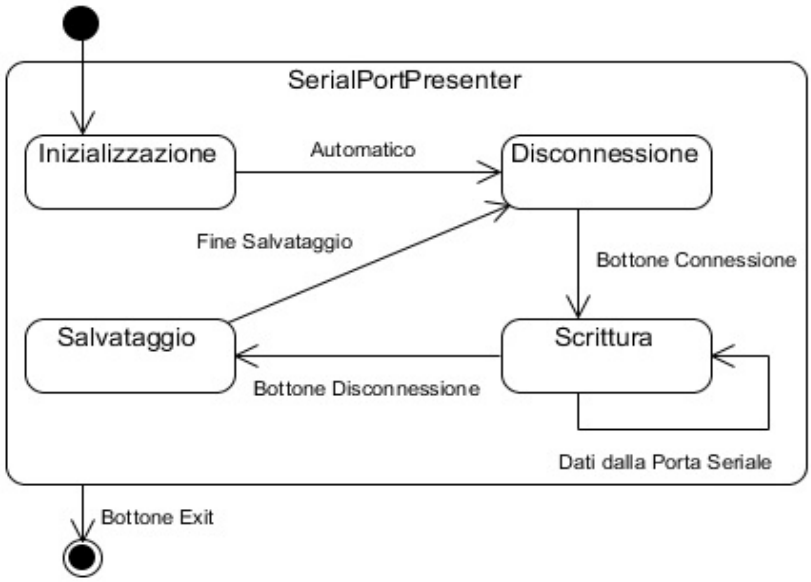

Figura 5.14

Nella fase di inizializzazione, il presenter alloca la memoria condivisa ove scrivere i dati ottenuti in quaternioni in modo da poterli condividere con altre applicazioni di lettura. In questa fase inoltre si prepara la porta seriale per una possibile connessione. Fatte queste operazioni, automaticamente si passa alla fase in cui l'automa attende un evento da parte

dell'utente che porti alla connessione con la piattaforma Arduino. Avvenuta la connessione, il presenter aspetta i dati spediti sulla porta seriale, ed una volta arrivati li scrive nella memoria condivisa per poi attendere quelli successivi.
Quando l'utente decide si interrompere la scansione dei dati, il presenter salva in un file indicato i quaternioni memorizzati fino a quel momento per poi ritornare in fase di disconnessione.

ExternalQuaternionViewer: questo è un presenter di lettura, ovvero legge i dati dalla memoria condivisa per rappresentare le rotazioni del sistema di riferimento solidale al dispositivo Arduino, rappresentato con un'auto come nel RotationDemo.
Le somiglianze con il presenter appena menzionato sono varie, infatti la dipendenza delle varie classi è simile come mostrato nel diagramma 5.10.

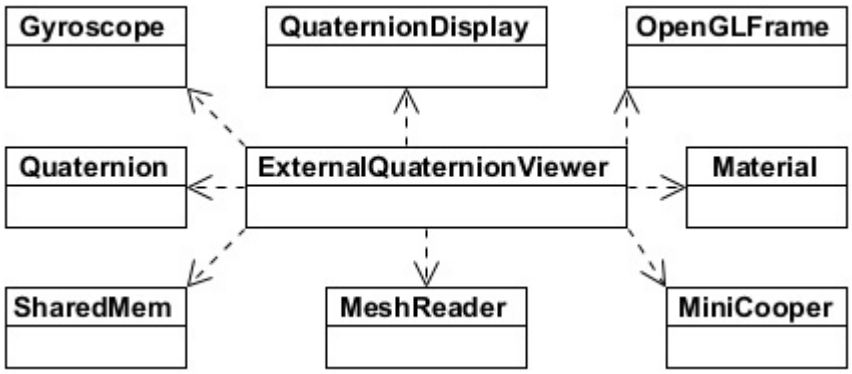

Figura 5.15

A differenza del RotationDemo, si ha la comparsa della memoria condivisa e non si utilizzano gli slider poiché i dati non sono inseriti manualmente ma letti direttamente dalla memoria.
Nella figura 5.16 viene indicato l'automa che governa questo presenter.
Si può notare come all'inizio il presenter viene posizionato in una fase di attesa della memoria condivisa, ovvero aspetta che un presenter scrittore crei la memoria dalla quale attingere i dati da leggere. Si passa in fase di lettura solo quando la memoria è stata creata e l'utente collega manualmente questo presenter. Nella fase successiva il presenter incomincia a prelevare i dati dalla memoria condivisa.

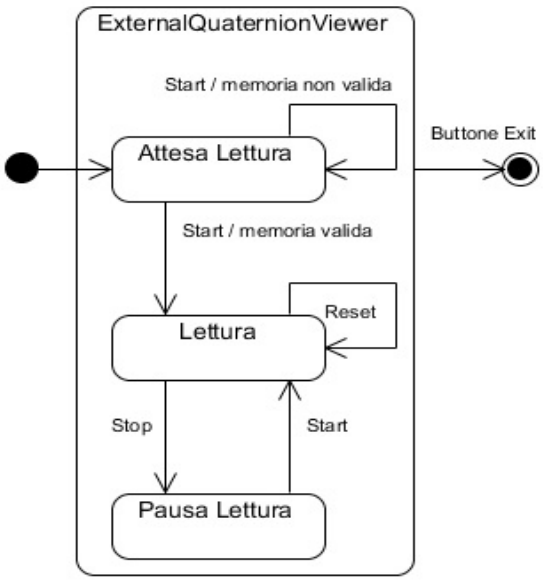

Figura 5.16

Naturalmente se i dati non sono corretti, ovvero non corrispondono ad un quaternione valido a causa di una mancata inizializzazione della memoria condivisa, viene rappresentato un quaternione identità. In questa fase si può solo resettare il quaternione rappresentato, appunto come descritto nel capitolo inerente; questo viene eseguito allo scopo di togliere tutte le modifiche compiute fino a quel momento dal sistema di riferimento solidale alla piattaforma. Da questa fase si può mettere in pausa la lettura dei dati, ovvero si può interrompere temporaneamente la resa grafica dalla memoria condivisa per riprenderla in un secondo momento. In questo caso, se la memoria non viene letta, allora viene visualizzata l'ultima rotazione scansionata.

QuaternionFileSlider: Questo presenter utilizza i salvataggi dei movimenti dell'apparecchio Arduino eseguiti dal SerialPortPresenter, infatti i dati salvati vengono dapprima memorizzati e poi scritti nella memoria condivisa a seconda del movimento dello slider fatto.
Lo slider in questo caso rappresenta la linea temporale dall'inizio fino alla fine della registrazione compiuta dal SerialPortPresenter.
Le informazioni raccolte poi sono passate all'ExternalQuaternionViewer con lo scopo di visualizzare graficamente le rotazioni salvate. Anche in questo caso viene applicato il

paradigma del presenter lettore e scrittore, infatti in questo modo si possono utilizzare programmi differenti per poter leggere ed interpretare i dati salvati e riproposti dallo slider.
Come si può vedere dall'immagine sottostante, non si fa uso dei componenti della categoria View ma viene utilizzata direttamente la libreria grafica, dato che la sua struttura è talmente semplice da poter essere incorporata nel codice del presenter stesso. Per il resto vengono utilizzate le librerie per convertire i dati letti in quaternioni e per creare la memoria condivisa ove scrivere.

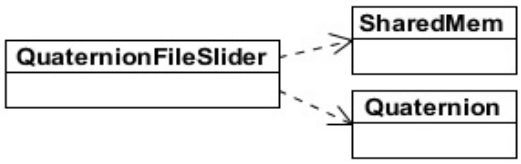

Figura 5.17

Nell'immagine successiva è rappresentato l'automa che riassume il comportamento dello slider.

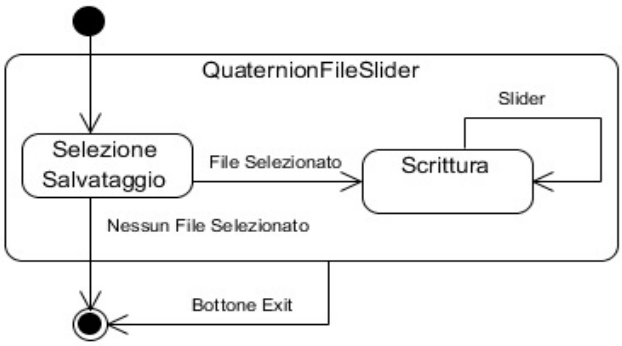

Figura 5.18

Si può vedere nella fase di selezione che all'utente viene lasciata la libertà di scegliere il file di salvataggio da rappresentare, o addirittura si lascia la possibilità di uscire dall'applicazione.
Una volta scelto il salvataggio desiderato, si entra nella fase di scrittura, ove il presenter scrive nella memoria condivisa la rotazione richiesta dall'utente. Si ricordi che lo slider rappresenta tutta la successione temporale delle rotazioni memorizzate nel file di salvataggio dall'inizio alla fine, quindi spostamenti dello slider implicando una visione in avanti o indietro nel tempo delle rotazioni fatte con il giroscopio al momento della registrazione.

PresenterChooser: fino ad ora abbiamo visto i vari presenter di lettura e scrittura in modo autonomo, ovvero non legati da un processo unico. Questo presenter accomuna tutti i lettori e gli scrittori in modo da affidare ad ognuno un thread indipendente. Nell'immagine sottostante sono indicati con un diagramma di classe tutti i vari presenter accorpati.

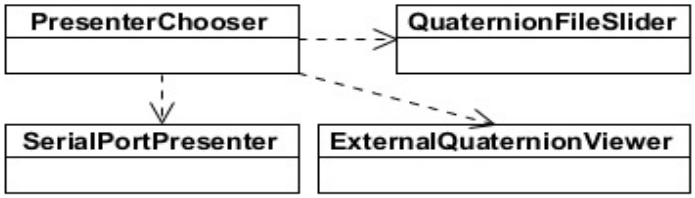

Figura 5.19

Da quanto detto, questo componente risulta come un contenitore. Il suo unico scopo è quello di fornire una pulsantiera per far partire a richiesta un lettore ed uno scrittore, a seconda dell'utilizzo che si vuole fare dei presenter collegati. Con questo componente perciò si possono interpretare graficamente i dati del sensore Arduino per poi eventualmente salvarli, oppure si possono rendere in grafica 3D i dati di un salvataggio precedentemente eseguito.
Nel diagramma seguente viene mostrato il comportamento del PresenterChooser.
Dal diagramma della figura 5.20 si può vedere che il compito del PresenterChooser è quello di fornire una posizione di attesa per i presenter lettori e scrittori. La loro attivazione ed il relativo funzionamento è da considerarsi parallelo poiché sono posizionati in thread differenti. Dal disegno si capisce che può essere scelto solo uno scrittore, quindi si possono visualizzare dei quaternioni derivanti dalla piattaforma Arduino oppure dal file di salvataggio, non da entrambi. Una volta selezionato il presenter, questo si comporterà come nei diagrammi visti in precedenza.
É da notare che i vari presenter possono essere attivati in qualsiasi momento ed in qualsiasi ordine; questo argomento verrà preso in analisi nei paragrafi successivi.

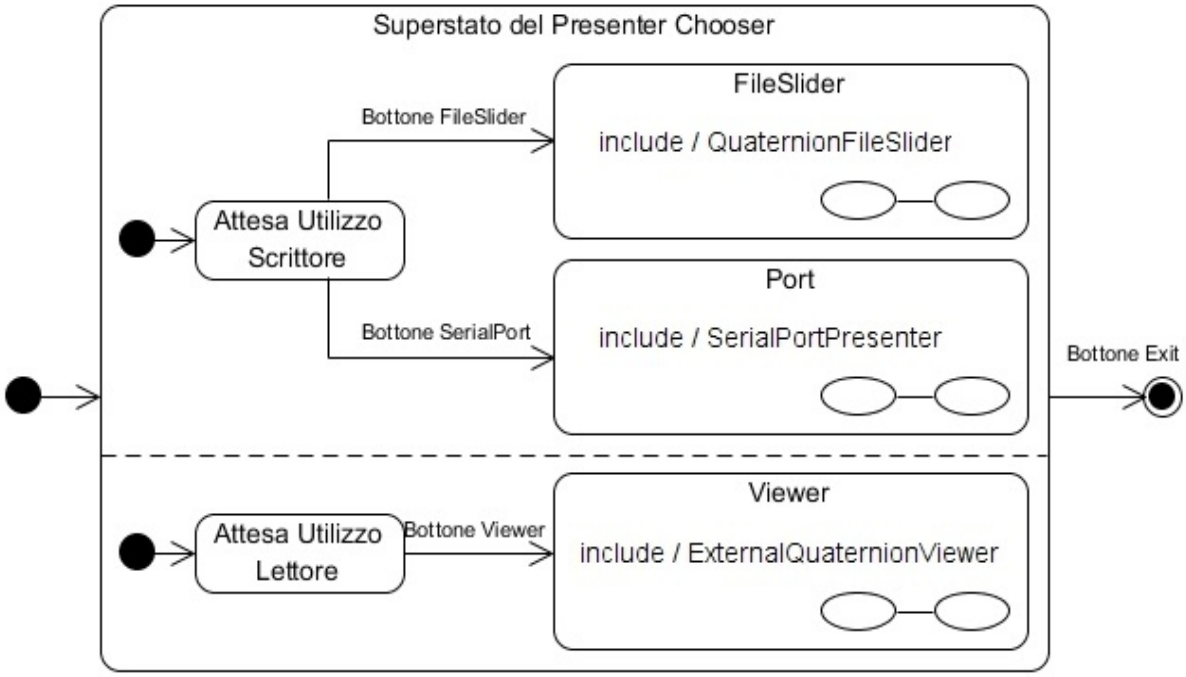

Figura 5.20

Nell'immagine 5.21 viene rappresentata l'interfaccia grafica attribuita al PresenterChooser e quindi a tutti i vari componenti che ne fanno parte.

Naturalmente i componenti non sono in una singola interfaccia grafica ma ognuno dispone di una propria finestra indipendente attivabile attraverso la pulsantiera del PresenterChooser.

Nell'immagine non viene reso un possibile utilizzo dell'applicazione (infatti compaiono sia il QuaternionFileSlider che il SerialPortPresenter), ma vengono rappresentati singolarmente tutti i possibili componenti richiamabili nella posizione stabilita a schermo dal PresenterChooser.

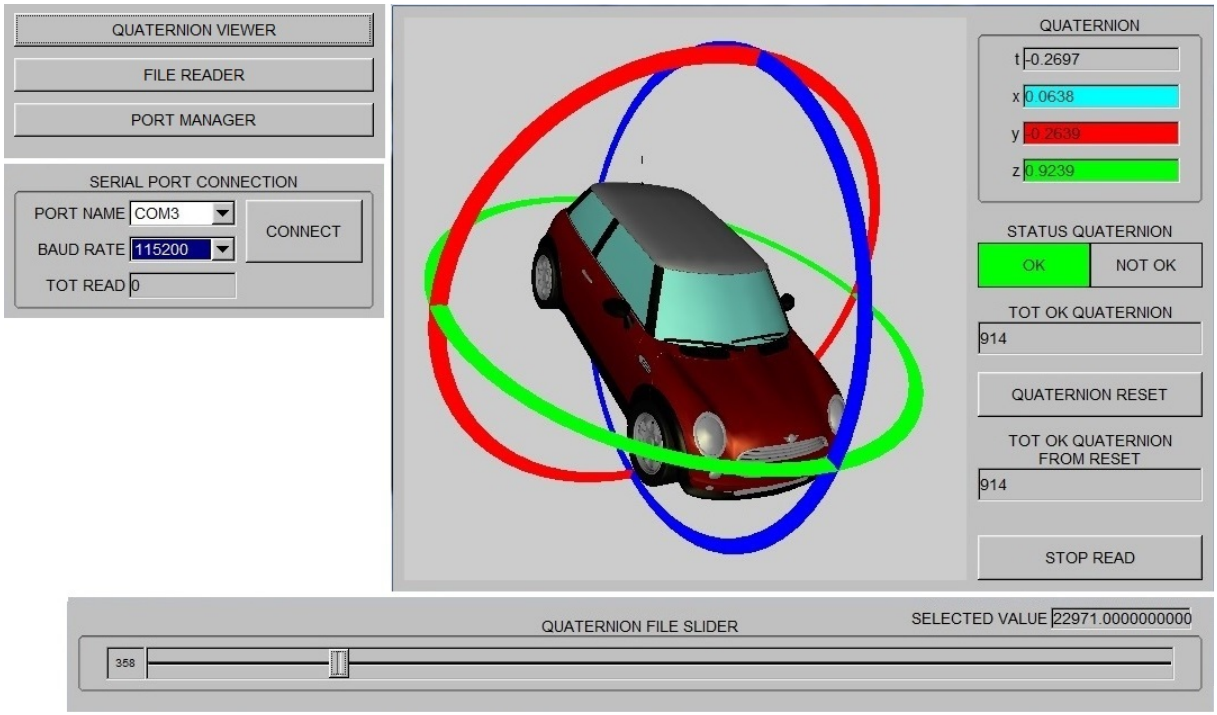

Figura 5.21

Interazione tra i processi

Fino ad ora sono stati considerati singolarmente sia gli sketch per Arduino sia i vari presenter che compongono il PresenterChooser, senza mai considerare la loro interazione.

Da quanto visto dagli automi mostrati nei vari diagrammi di stato, questi componenti si possono arrestare in diversi stadi per aspettare o un segnale di interazione con l'utente oppure un cambiamento di stato di un altro presenter.

Di seguito vengono mostrati dei diagrammi di attività che rappresentano due possibili scenari in cui i vari componenti interagiscono tra di loro e si identifica la giusta sequenza in cui si possono richiamare. Gli stessi stati inseriti nelle immagini precedenti si possono ritrovare nei diagrammi di attività, con l'aggiunta di indicazioni grafiche di spedizione e ricezione di segnali da parte dei diversi componenti. I segnali di spedizione rappresentano una conseguenza dal passaggio di uno stato all'altro, mentre la ricezione di un segnale indica un requisito per passare da uno stato all'altro.

Nel diagramma 5.23 viene rappresenta la sequenza in cui i vari componenti si richiamano per

poter leggere i dati spediti dallo sketch Arduino in esecuzione, per poi interpretarli graficamente ed alla fine salvarli.

Dall'immagine si può vedere che all'inizio la piattaforma Arduino aspetta un segnale di connessione dal PresenterChooser; attraverso quest'ultimo componente si può indifferentemente far partire prima l'interfaccia grafica oppure il presenter della porta seriale. In ogni caso il visualizzatore grafico si ferma se la memoria condivisa non è stata creata dal gestore della porta. Una volta creata la memoria, è possibile far avanzare il visualizzatore. Quando si impostano le caratteristiche della porta seriale e si esegue la connessione, lo sketch Arduino termina il suo stato di attesa per incominciare a spedire i quaternioni di trasformazione. A questo punto i presenter si troveranno nello stato di lettura e scrittura; da sottolineare il fatto che in queste ultime fasi non sono indicati segnali di attivazione o disattivazione per indicare al lettore che sono arrivati nuovi dati, infatti all'inizio la memoria viene inizializzata con un quaternione identità e successivamente aggiornato con dati della piattaforma sempre più recenti; in ogni caso il visualizzatore ha sempre dati da leggere e quindi al massimo renderà graficamente l'identità oppure uno stato non ancora aggiornato.

Il flusso di dati che intercorre tra sketch, controllore della porta seriale e visualizzatore tridimensionale può essere reso con il diagramma di sequenza rappresentato nell'immagine 5.22. Da questo si può notare che il flusso di informazioni non è bidirezionato, quindi lo sketch non comunica con nessun presenter dopo essere entrato nella fase di spedizione dati, ed il presenter di lettura non comunica con quello di scrittura.

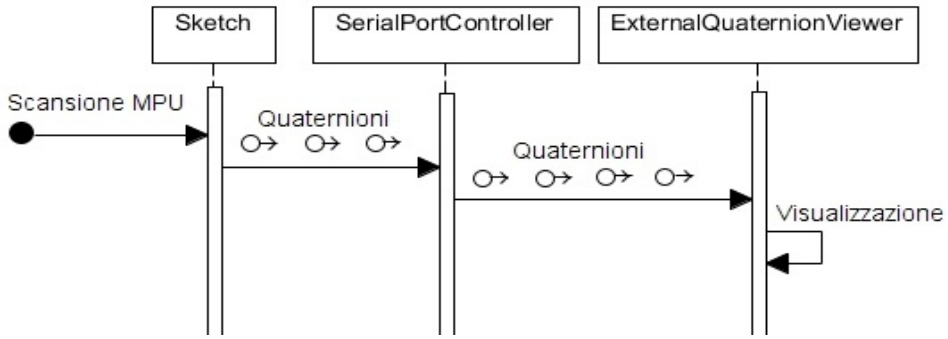

Figura 5.22

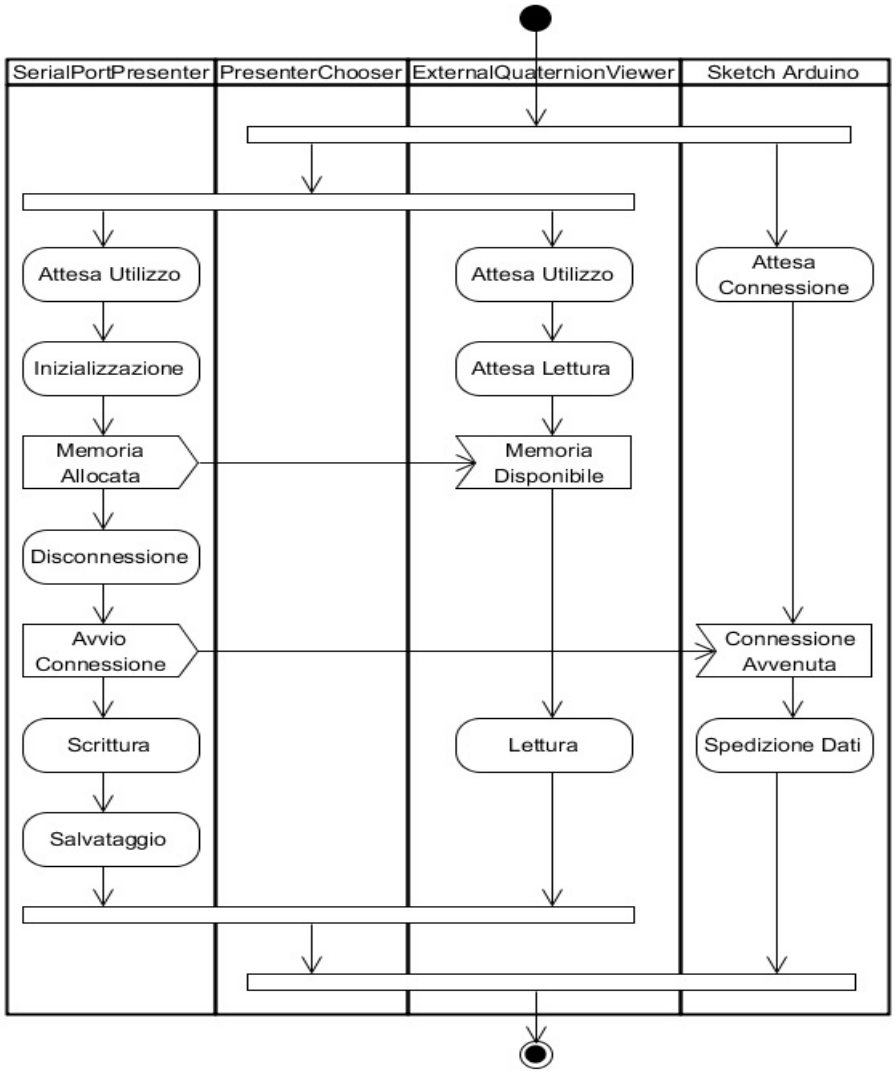

Figura 5.23

Si può notare che per ottenere soltanto dei dati da salvare non è necessario attivare anche il QuaternionViewer, infatti il flusso di informazioni è unidirezionale ed il componente addetto al salvataggio delle rotazioni eseguite è il SerialPortPresenter.

Una più semplice interazione avviene quando si vogliono visualizzare i dati salvati, quindi si utilizza solo il PresenterChooser senza l'ausilio del sensore Arduino; questo comportamento viene mostrato nel diagramma delle attività dell'immagine 5.24.

Similmente a quanto abbiamo già visto precedentemente, sia il QuaternionFileSlider che il visualizzatore di quaternioni possono essere richiamati indifferentemente in qualsiasi ordine, al massimo il visualizzatore andrà ad arrestarsi fino a che il QuaternionFileSlider non creerà

la memoria condivisa.

Considerazioni simili a quelle già viste nel caso precedente si possono fare per il flusso di informazioni unidirezionale che caratterizza anche questo scenario, il flusso infatti inizia dal QuaternionFileSlider grazie all'attivazione da parte dell'utente, fino ad arrivare alla visualizzazione dei dati scritti nella memoria condivisa.

Questo si può vedere nel diagramma di sequenza rappresentato nell'immagine 5.25.

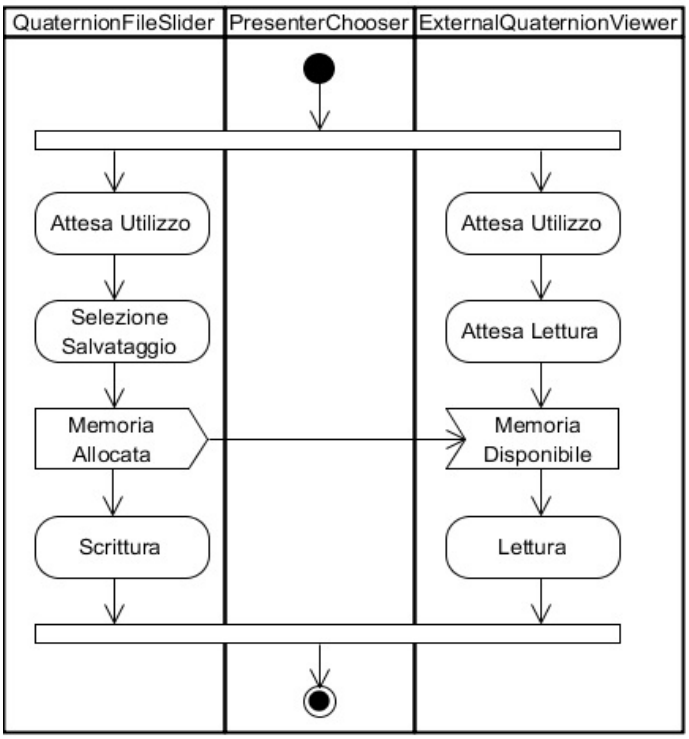

Figura 5.24

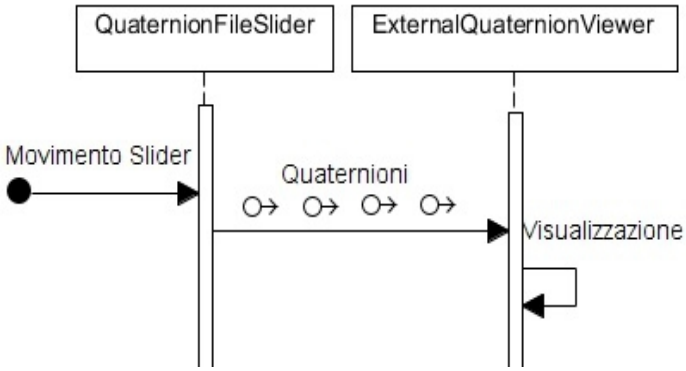

Figura 5.25

Visione generale dei componenti

Uno dei requisiti per la creazione di questo software riguarda la scomposizione in componenti del programma per una possibile sostituzione immediata, questo allo scopo di testare nuove soluzioni per analizzare la piattaforma Arduino.
Oltre a sostituire gli sketch, è possibile cambiare altri componenti, come il visualizzatore grafico in caso si volesse interpretare i quaternioni in modo differente. Questo può portare a creare altri componenti in lettura o scrittura sempre mantenendo inalterati i componenti già sviluppati.
Il comportamento appena descritto può essere riassunto nel seguente diagramma, in cui sono messi in evidenza le varie astrazioni dei componenti da implementare ed anche quelli fino ad ora creati.

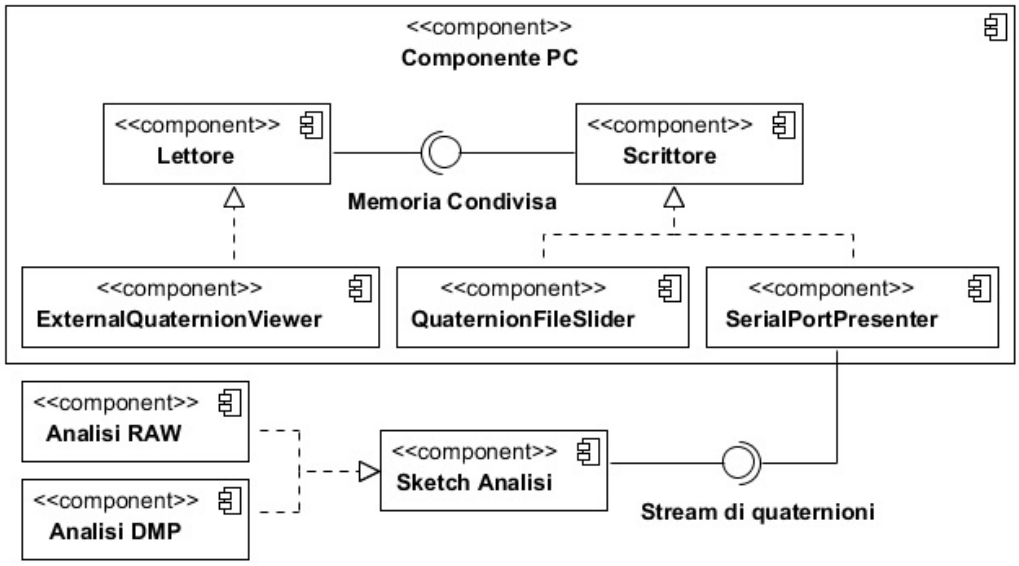

Figura 5.26

Si può vedere da questo schema le varie interfacce che legano i componenti, ovvero la memoria condivisa per i lettori e scrittori, ed il flusso di quaternioni che collega il presenter della porta seriale con lo sketch Arduino. Questi quaternioni sono spediti con una formattazione particolare che è comune ad entrambi i componenti in gioco.
Questi connettori software devono essere inclusi nei componenti che implementano le

astrazioni appena mostrate per poter far funzionare tutta l'architettura.
Da notare inoltre che i componenti lettori e scrittori possono essere considerati dei sottocomponenti, infatti alla fine si ottiene un unico programma (come il presenterChooser che comprende diversi presenter al suo interno) oppure un insieme di programmi collaborativi (questo in caso di utilizzo di componenti esterni diversi da quelli mostrati fino ad ora). In ogni caso questi due possibili scenari sono accomunati dal fatto di potersi considerare come una unica parte di una architettura software risiedente in una singola macchina.
Questo concetto viene ribadito anche nel diagramma di deployment degli artefatti prodotti riportato nel paragrafo successivo.

Deployment del sistema

L'architettura proposta in questo capitolo fornisce del software che, per essere utilizzato, deve essere inserito in strumenti collegati tra di loro per interagire e dare vita al sistema richiesto.
Nel nostro caso la disposizione fisica dei componenti è abbastanza semplice, ovvero si ottiene alla fine la creazione di artefatti che interagiscono grazie ad una piattaforma Arduino collegata ad un computer, tutto questo attraverso una connessione seriale emulata su una porta USB. I due sistemi hardware non devono necessariamente essere collegati con un cavo, a meno che non sia strettamente necessario (ad esempio per l'alimentazione del dispositivo Arduino). La connessione infatti può anche essere wireless, purché venga rispettata l'astrazione della porta seriale.

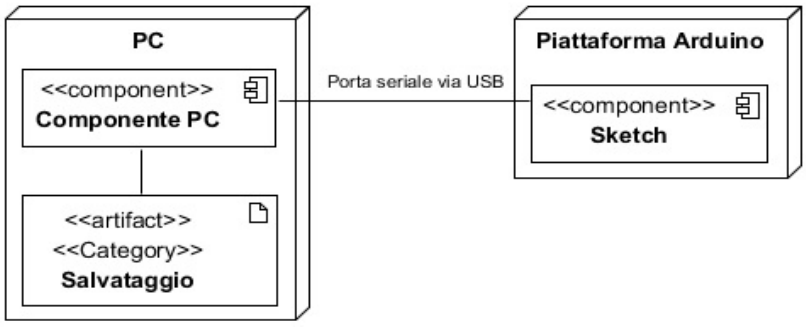

Figura 5.27

Come da figura, si può vedere come la collaborazione dei componenti può portare alla

produzione dei file di salvataggio, che si possono quindi considerare come una parte fondamentale dell'architettura proposta. Questi artefatti sono una risorsa essenziale per lo studio dei movimenti del giroscopio, infatti è attraverso la produzione di questi file che si possono osservare i diversi comportamenti degli sketch testati in laboratorio e su strada.
L'architettura proposta gestisce solo la scrittura su questi file allo scopo di memorizzare i movimenti fatti, quindi si lascia all'utente la loro gestione, disposizione e cancellazione.

Considerazioni

In questo capitolo è stata mostrata l'architettura proposta per l'analisi di movimento della piattaforma Arduino con l'ausilio di tutte le informazioni viste nei capitoli in precedenza. Attraverso un lavoro prototipale, è stato possibile studiare i vari sistemi di riferimento e testare gli algoritmi di rotazione dei quaternioni.
Dopo aver raggiunto una fase ottimale del software, è stato possibile eseguire dei test per ottenere un'analisi degli algoritmi presenti negli sketch allo scopo di valutare l'uso della piattaforma in situazioni reali e non solo prettamente teoriche.
Il codice scritto per l'architettura ottenuta è formato all'incirca da 10.000 righe comprese tra file per C++ e sketch Arduino.
Questa struttura inoltre è stata costruita con lo scopo di essere riutilizzata in altre applicazioni inerenti lo studio delle rotazioni ed eventualmente espansa per altri esperimenti con la piattaforma Arduino.
Nel prossimo capitolo si affronteranno i risultati ottenuti dalla fase di sperimentazione.

Capitolo 6

Risultati

In questo capitolo si valutano i risultati dei test fatti sugli algoritmi implementati nell'architettura fino ad adesso presentata. Con queste prove si vuole controllare se le soluzioni usate per catturare le rotazioni del dispositivo ricostruiscono più o meno fedelmente il movimento eseguito.

Sono stati presi in esame due tipi di soluzioni implementate negli sketch Arduino, uno utilizza il componente hardware DMP della piattaforma e l'altro usa i dati grezzi restituiti dal giroscopio del sensore MPU. Entrambi restituiscono un quaternione indicante la posizione del sistema di riferimento solidale con il dispositivo. Nel primo caso si utilizza una soluzione hardware creata per avere quaternioni di rotazione, impiegando ottimizzazioni interne al componente MEMS e ed in particolare utilizzando anche le misure dell'accelerometro. Nel secondo caso invece si utilizzano le velocità angolari restituite dal giroscopio della piattaforma (dati RAW) senza alcuna elaborazione da parte di altri componenti hardware come il DMP o l'accelerometro, e si ricostruiscono le rotazioni con le formule indicate nei capitoli precedenti.

In entrambi gli sketch vengono utilizzate le librerie i2cdev descritte precedentemente, sia per estrarre i valori grezzi dal microprocessore MPU, sia per controllare l'uso del DMP. Sono state utilizzate ove possibile le stesse impostazioni di sensibilità di rilevazione e le stesse velocità di scansione dei dati.

Naturalmente sono utilizzati dei componenti di lettura al lato computer adatti per ogni tipo di sketch per una corretta interpretazione tridimensionale.

Di seguito verranno sottoposte le diverse prove eseguite utilizzando il dispositivo con movimenti manuali oppure montato su auto.

Siccome queste prove alla fine portano a dei cambiamenti visibili in tre dimensioni, si è

deciso di mostrare le rotazioni rispetto ad una posizione standard della piattaforma, ovvero si paragonano gli spostamenti a partire dalla rappresentazione del quaternione identità (figura 6.1). Quindi quando si avranno movimenti significativi, si farà riferimento alla posizione iniziale dell'identità, ottenuta analizzando i salvataggi e portandoli ad un punto di reset come descritto nei capitoli precedenti. Questo procedimento viene fatto per poter studiare dei particolari fenomeni, isolandoli dalle rotazioni precedenti della piattaforma.

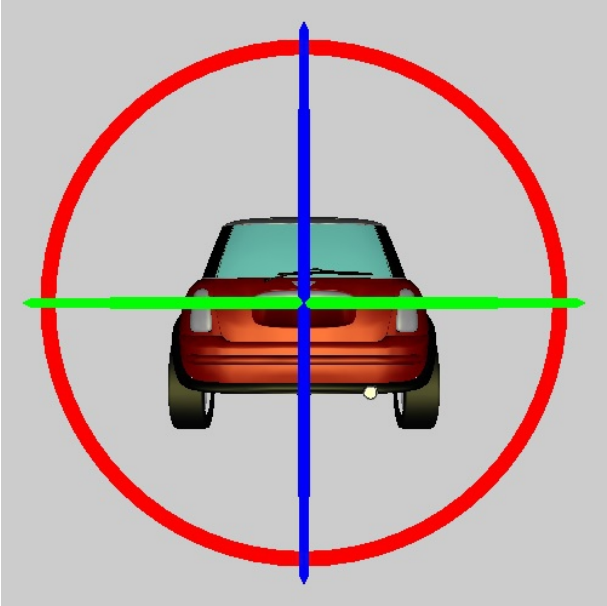

Figura 6.1

Temperatura del processore MPU

Il processore MPU può surriscaldarsi durante il suo utilizzo e di riflesso i valori di velocità angolare che restituisce possono variare a causa dell'aumento di temperatura.
Di conseguenza, abbiamo monitorato la temperatura del processore dall'inizio del suo utilizzo fino al raggiungimento di un valore stazionario per visualizzare se le varie velocità ritornate sono influenzate.
Attraverso uno sketch creato per estrarre la temperatura del processore MPU, si sono ottenute delle informazioni utili per la calibrazione dei valori estratti dal microprocessore. Di seguito è mostrato un grafico che rappresenta un esempio di variazione di temperatura ottenuta durante

queste prove.

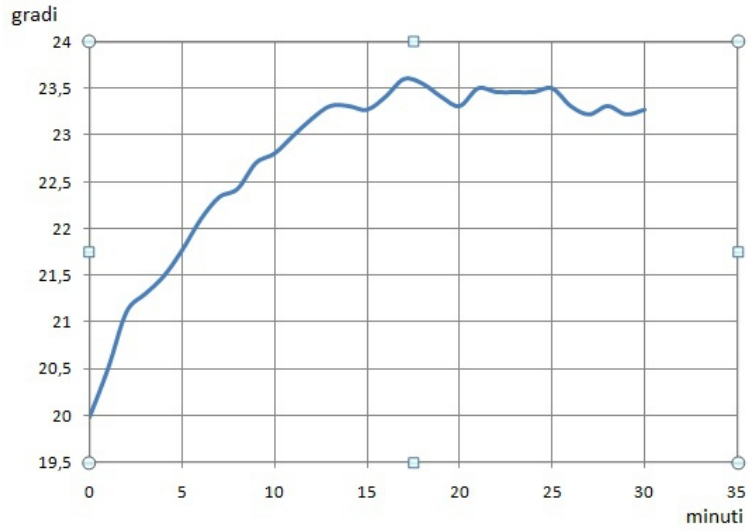

Figura 6.2

Questo grafico fa riferimento ad una analisi compiuta in laboratorio con una piattaforma non racchiusa in un contenitore, il quale avrebbe favorito il surriscaldamento. Inoltre la piattaforma non è dotata di batteria ma alimentata attraverso un cavo USB collegato ad un computer, quindi non ha risentito di sbalzi aggiuntivi di temperatura causati dal surriscaldamento di una eventuale batteria collegata.

Da quanto si può notare, la piattaforma parte con una temperatura di poco meno di 20 gradi celsius per poi attestarsi sui 23,25 gradi circa, attraversando una fase transiente di circa 15 minuti. In totale si è notato una sbalzo di poco più di 3 gradi celsius.

Bisogna notare che questi esperimenti sono stati fatti in condizioni particolari, infatti se si dovesse considerare un dispositivo racchiuso in un contenitore e soggetto a forti sbalzi climatici (da -20 a +40 gradi), il grafico della sua temperatura risulterebbe differente.

Sono state eseguite prove anche in ambienti con diverse temperature mantenute in modo costante e si è osservato sempre un leggero surriscaldamento, però tale da non comportare il raggiungimento di una temperatura media di 23,25 gradi celsius come nelle particolari condizioni elencate prima. Quindi si può dire che la media trovata in laboratorio non è un valore che il processore MPU raggiunge costantemente.

Questo sta ad indicare che la temperatura del processore a regime dipende più dall'ambiente in cui si trova che dal surriscaldamento derivante dall'analisi ed elaborazione continua delle

rotazioni. Le variazioni di temperatura dell'ambiente in cui si trova il dispositivo portano inevitabilmente a variazioni di temperatura del processore con possibili conseguenze sui risultati ottenuti. Questa ultima osservazione sarà presa in esame nel prossimo paragrafo.

Calibrazione del dispositivo Arduino

Le stesse case produttrici dei componenti utilizzati indicano nelle specifiche di utilizzo il grado di tolleranza delle misurazioni, che per quanto siano basse possono portare alla lunga a delle alterazioni delle rotazioni scansionate. La media dei valori ottenuti, per quanto riguarda accelerazioni e velocità angolari, non si attesta sullo 0 come si potrebbe pensare, ma si concentra su un valore numerico che discosta poco dalla media teorica. I dati quindi vanno corretti con la media di un campionamento di valori detto offset, in modo da poter calibrare i valori restituiti dalla MPU al fine di essere più vicini allo 0 teorico.
In questa parte si analizzano dei potenziali offset calcolati allo scopo di calibrare un dispositivo e di vedere le loro variazioni in dipendenza delle temperatura nelle condizioni elencate nel paragrafo precedente.
Si ricordi che una condizione necessaria per il calcolo dell'offset di calibrazione è che il dispositivo deve rimanere fermo.
Nel gruppo di immagini 6.3 vengono mostrate delle medie aritmetiche di 1000 valori continui di velocità angolare scansionati in laboratorio. Questi dati sono ottenuti nelle stesse condizioni del test della temperatura. Per ottenere una media sono necessari 2 secondi; sono state eseguite analisi ogni minuto per circa 1 ora.
Bisogna ricordare che nelle immagini è rappresentato uno dei vari risultati ottenuti preso a campione; i dati analizzati presentano tutti medesime caratteristiche.
Questi test sono stati fatti per poter individuare il momento più favorevole per stabilire un offset da utilizzare per la calibrazione del dispositivo; infatti i valori presentati nei grafici rappresentano tutti potenziali offset che possono essere applicati al fine di ottenere delle rotazioni verosimiglianti ai movimenti compiuti.
Come si può notare le medie ottenute variano nel tempo con scarti l'una dall'altra di 0,18 gradi al secondo, più precisamente per gli assi X ed Y i valori si attestano sui -3,42 e -3,24 gradi al secondo, per l'asse Z invece variano tra i -3,6 e -3,42 gradi al secondo.

Queste variazioni sono ovviamente entro le tolleranze stabilite dalle case produttrici, quindi sono da considerarsi normali.

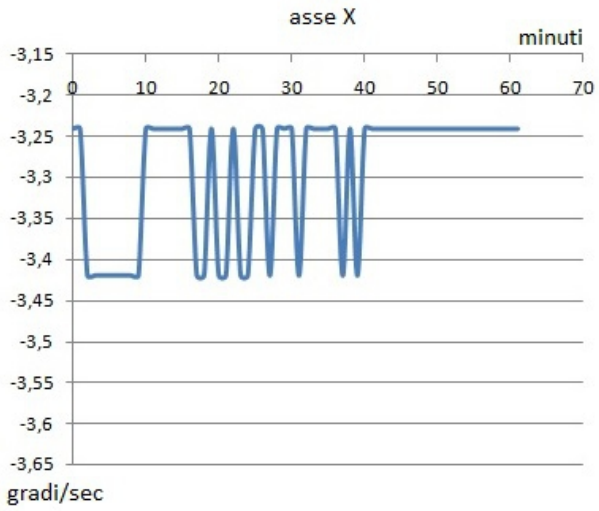

Figura 6.3a

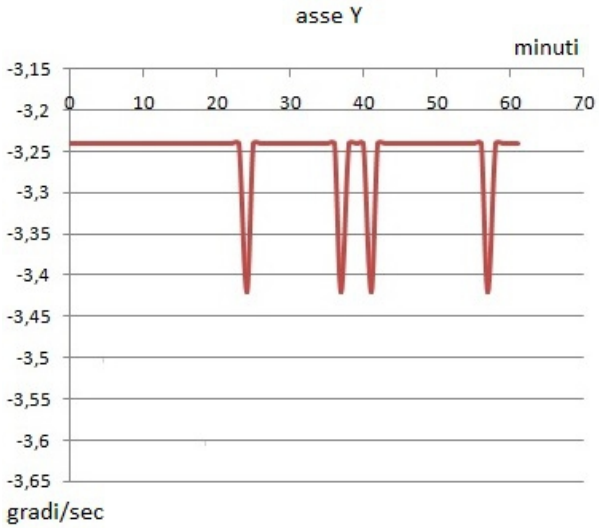

Figura 6.3b

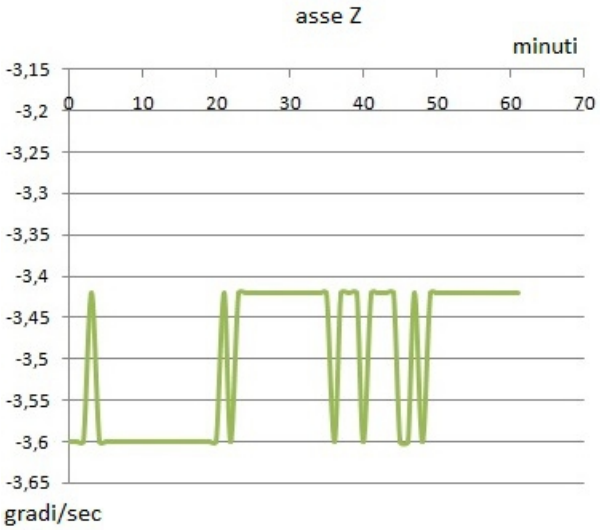

Figura 6.3c

Si può vedere che i valori possono cambiare repentinamente da un minuto all'altro, oppure rimanere stabili anche per una ventina di minuti per poi variare. Questo comportamento rende il compito di stabilire un offset per un utilizzo continuo estremamente difficile. Alla lunga infatti si andrebbe a perdere inevitabilmente di precisione.

Possiamo vedere che questi valori per gli assi X, Y e Z sono lievemente influenzati dalla variazione di temperatura fino a che non viene superato lo stato transiente, dopo circa 15

minuti dall'accensione del dispositivo. Questo si può vedere nei grafici per l'asse X e Z, ove il valore si attesta alla temperatura più elevata verso i 15 – 20 minuti, mentre per l'asse Y sembra essere ininfluente.

Quanto appena presentato è solo uno dei molti casi analizzati, e in tutti si può notare che il lieve surriscaldamento influenza leggermente l'offset finale, portando a dei valori che oscillano tra un valore minimo e massimo la cui distanza è pari a 0,18 gradi al secondo come visto nell'esempio proposto, ovvero la distanza minima percepibile dal sensore.

Superata la fase transiente del surriscaldamento, si può notare che si ha sempre una oscillazione di 0,18 gradi al secondo dei valori di calibrazione.

Anche se la differenza è minima, questa può portare in breve tempo a delle rotazioni che possono non corrispondere alla rotazione del dispositivo, anche se fermo.

Un esempio di questo è mostrato nell'immagine seguente, in cui si può vedere il comportamento di uno sketch con un offset impostato nella fase iniziale.

Lo stato di partenza del dispositivo è rappresentato nell'immagine 6.1, mentre nell'immagine 6.4 viene mostrata la posizione del sensore dopo 30 minuti, senza mai essere mosso. Si può notare come la posizione non rispecchi minimamente lo stato di inattività del dispositivo.

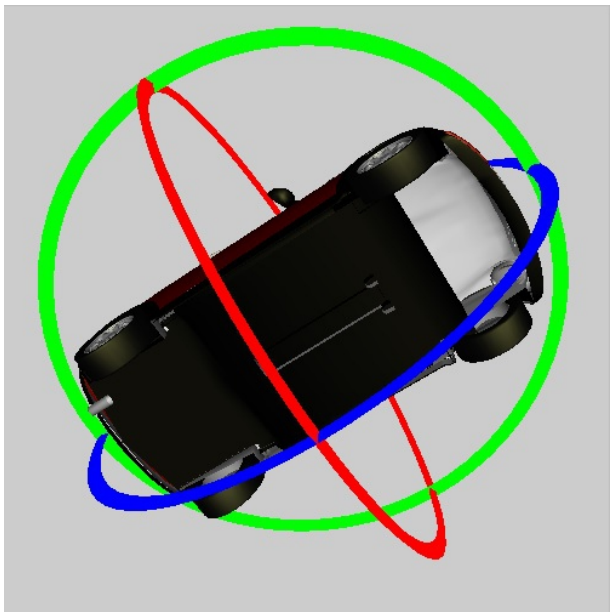

Figura 6.4

Con una analisi più accurata, si è visto che l'errore di calibrazione coinvolge tutti e tre gli assi

di rotazione, ed il loro senso di rotazione non è predicibile a priori.

Per i nostri scopi questo non è molto rilevante, infatti la nostra analisi si concentra sugli incidenti, che si sviluppano di solito entro i due decimi di secondo, quindi l'importante è che gli spostamenti derivanti dalle calibrazioni siano trascurabili in questo breve lasso di tempo.

Nel caso dello sketch che sfrutta il DMP sono utilizzate funzioni interne per la sua ottimizzazione, comprensive di un meccanismo di calibrazione comune a molti microcontrollori di rilevamento di spostamenti. Questo meccanismo agisce quando il microprocessore decide di essere in una posizione stabile o invariata, quindi effettua una nuova calibrazione al fine di diminuire l'errore di misurazione, dipendente da fattori come i cambiamenti di temperatura e di umidità.

Nell'immagine sottostante si può vedere l'utilizzo del dispositivo in una posizione di quiete per 30 minuti.

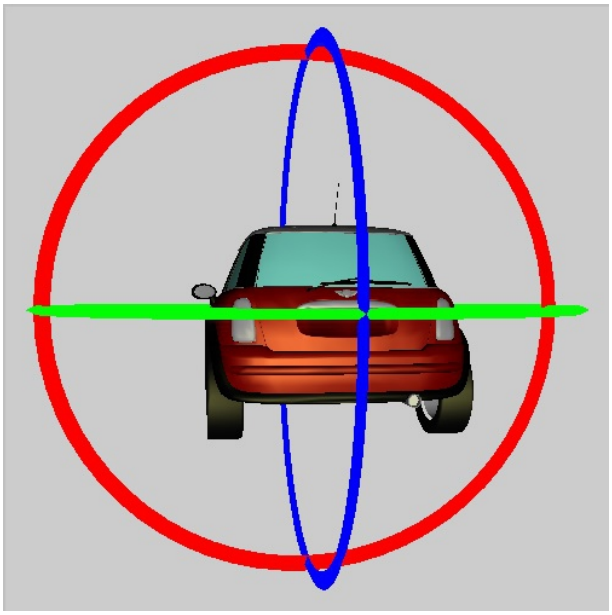

Figura 6.5

Lo sketch che utilizza il DMP ha una leggera rotazione di yaw pari a 0,144 radianti (8,25 gradi) rispetto alla posizione iniziale. Si può considerare irrilevante questa variazione per la ricostruzione anche di percorsi a lunghi periodi oltre alle rotazioni sviluppate in un incidente di pochi decimi di secondo.

Risultati con movimenti manuali

Questo è il test principale eseguito sull'apparecchio per vedere se con movimenti manuali il sistema di riferimento rappresentato rispecchia quello effettivo del sensore. In questo test sono stati eseguiti movimenti di rotazione sul piano orizzontale in modo da osservare la verosimiglianza della posizione ricostruita.
Le rotazioni eseguite riguardano angoli di 180 e 360 gradi a diverse velocità al fine di vedere come si comportano gli sketch che utilizzano sia il DMP che la rotazione elaborata dai dati grezzi.
Di seguito sono riportati i risultati degli esperimenti eseguiti sul piano, quindi riguardanti solo una dimensione ruotata.
Sono stati compiuti test più o meno veloci, così da avere diverse categorie a seconda del tempo impiegato a ruotare. Nella tabella sono state inserite le approssimazioni in gradi dei valori ottenuti con rotazioni di 180 e 360 gradi degli sketch utilizzanti il DMP.

Sketch DMP	Rotazioni di 180°	Rotazioni di 360°
< 3 secondi	179,5	359
10 secondi	179,5	359
30 secondi	179,5	359
1 minuto	179,5	358

Si può vedere che i valori non riescono a superare l'angolo di 180 o 360 gradi compiuto per la rotazione. Questo indica che, per quanto preciso sia l'algoritmo, la differenza dipende sempre dai dati scansionati dal processore MPU, quindi è possibile che i valori di rotazione vengano discretizzati portando ad una perdita di precisione per difetto.
Eseguendo gli stessi test con gli sketch che utilizzano i dati RAW, si possono vedere valori discordanti tra di loro, ovvero molto prossimi alla rotazione teorica desiderata oppure molto distanti da questa (anche 4-5 gradi nei test che durano un minuto). Con una analisi più approfondita si è individuato, come causa di queste disparità di dati, l'errore di calibrazione mostrato nel paragrafo precedente in combinazione con il senso di rotazione del dispositivo

(orario oppure antiorario).

Come è stato detto in precedenza, sussiste un errore di calibrazione nell'algoritmo che utilizza i dati grezzi; questo errore influenza il risultato della rotazione, inserendo un movimento aggiuntivo che influenza tutti gli assi del sistema di riferimento.

Dai vari test eseguiti a dispositivo fermo, si sono riscontrate rotazioni aggiuntive che, proiettate sul piano perpendicolare all'asse ruotato, portano ad una variazione che va da 1 fino a 3 gradi al minuto. Queste variazioni, nel lasso di tempo in cui sono state scansionate, risultano concordi tra di loro, ovvero provocano una rotazione oraria o antioraria all'angolo di yaw analizzato.

Quindi, se ruotiamo il dispositivo in senso orario oppure antiorario, otterremo un errore di misurazione paragonabile all'ampiezza dell'angolo che noi vogliamo ottenere (ad esempio 180 o 360 gradi come nei test); a questo risultato poi dobbiamo aggiungere l'errore di calibrazione a seconda della durata del test (abbiamo eseguito prove della durata di pochi secondi fino ad un minuto), che quindi andrà sommato o sottratto a seconda se il verso di rotazione del dispositivo è concorde o discorde con il senso di rotazione dell'errore.

Gli errori di misurazione dello sketch che utilizza dati RAW sono simili a quelli presentati nella tabella mostrata per lo sketch che utilizza il DMP, mentre l'errore di calibrazione varia di volta in volta.

Quindi, se ad esempio eseguiamo una rotazione di 360 gradi in senso orario della durata di un minuto, bisogna tenere conto dell'errore di misurazione di una rotazione di 360 gradi, per poi controllare il senso di rotazione dell'errore di calibrazione e trovare il suo ammontare per la durata di un minuto; se il senso di rotazione è orario allora verrà sommato l'errore di calibrazione, altrimenti verrà sottratto.

Da ciò possiamo capire che l'errore di calibrazione è irrilevante in tempi molto brevi, infatti in caso di analisi di rotazioni in un incidente, si dovrebbe tenere conto dell'errore di pochi decimi di secondo, quindi ininfluente.

Questo risultato indica inoltre che l'algoritmo utilizzato via software deve essere adeguato al funzionamento del sensore e non riferirsi semplicemente ad un modello teorico come quello presentato nei capitoli precedenti. Questa mancanza di precisione non è presente nello sketch che utilizza il DMP, il quale evidentemente utilizza al suo interno un algoritmo che tiene già conto del funzionamento del sensore MPU.

Risultati su percorso stradale

Dobbiamo ricordare che si sta analizzando il comportamento della piattaforma Arduino per vedere se è possibile impiegarla per le ricostruzioni dei movimenti di un automobile o di un veicolo a motore in generale, anche senza incidenti.
Per questi motivi è stato installato un sensore sul cruscotto di un'auto e testato su percorsi stradali di diversa natura e con tempi di percorrenza differenti.
In generale nei percorsi stradali non si verificano spostamenti in tre dimensioni, al massimo in due se ci si sposta in salita o discesa con un'auto che sta svoltando; perciò, in percorsi pianeggianti come quelli eseguiti, gli spostamenti si verificano solo in una dimensione.
Da questi esperimenti si è notato che gli sketch si comportano in modo differente, ovvero il sistema di riferimento ottenuto con le sole velocità angolari risulta più verosimile alla posizione dell'auto rispetto allo sketch che fa uso del DMP.
Dopo ulteriori prove si è vista la necessità di analizzare un percorso più semplice come una rotatoria per vedere di isolare questo fenomeno.
E' stata percorsa quindi, per diverse volte consecutive, una rotatoria di 270 metri impiegando al massimo 25 secondi per un giro completo. Si è andati ad una velocità costante compresa tra 38.5 e 40 Km/h, percorrendo all'incirca 10.8 metri al secondo, pari ad una velocità angolare media di circa 14.5 gradi al secondo.
Eseguendo quindi traiettorie curve a velocità angolare costante, si è notato che il DMP le interpreta come una condizione invariante in cui calibrare il sensore, similmente a quanto visto nei test con piattaforma ferma. Questo non risulta nei test nei movimenti manuali anche se le rotazioni in gioco sono molto più lente, infatti con gesti manuali non si riesce a ricreare una velocità costante, ma un susseguirsi di movimenti generati da accelerazioni e decelerazioni percepiti come stato di moto dal DMP.
Considerando quindi questo fenomeno riportato su un percorso stradale misto, si può notare che una traiettoria curva come una svolta può falsare la rotazione di tutto il tragitto compiuto.
Lo sketch che calcola la rotazione via software non risente di questo problema, poiché in questo caso il microprocessore MPU restituisce dei valori che non vengono filtrati e ottimizzati come nel caso del DMP.
L'impostazione del DMP per adesso non è modificabile attraverso le i2cdev. Queste librerie

infatti, utilizzate per governare le tecnologie MEMS della piattaforma Arduino, non forniscono allo stato dell'arte delle primitive adeguate o una documentazione esaustiva del comportamento del DMP in modo da poterlo personalizzare a seconda delle circostanze in cui bisogna scansionare le rotazioni.
Si può dire quindi che l'algoritmo che utilizza il DMP non può essere utilizzato in modo da registrare un intero percorso di un'auto, però può essere utilizzato per quanto riguarda l'elaborazione di rotazioni in un incidente. In questo caso il sistema di calibrazione del DMP non dovrebbe entrare in azione poiché banalmente in un urto non si dovrebbe verificare uno stato di quiete tale da permettere un ricalcolo degli offset

Considerazioni

In questo capitolo abbiamo visto il comportamento degli sketch utilizzati per ottenere la rotazione di una piattaforma.
Entrambi non sono definitivi, ma vanno migliorati come indicato nelle varie parti di questo capitolo.
E' da notare che se entrambi gli sketch visti calcolassero le rotazioni in ogni tipo di scenario possibile, la soluzione che utilizza hardware dedicato sarebbe senza dubbio preferibile. Le limitazioni di questa risorsa risiedono nelle librerie che ne permettono l'uso, evidentemente non ancora in stato definitivo. Attualmente è permesso l'uso del DMP senza concedere personalizzazioni o variazioni sulla gestione dei dati ottenuti dal microprocessore MPU.
Un esperimento che sarebbe stato interessante da provare riguarda la reazione degli algoritmi degli sketch in caso di un urto o incidente stradale. In questo caso si avrebbero dei repentini cambiamenti di velocità angolare e accelerazioni improvvise, che potrebbero coinvolgere quindi più di un asse di rotazione.
Con questo prototipo si è stabilito che il giroscopio è utile per poter determinare le rotazioni di un veicolo in un incidente, infatti permette di percepire movimenti che si evolvono in pochi decimi di secondo con un margine di errore più che accettabile.

Capitolo 7

Conclusioni e sviluppi futuri

In questa tesi è stata presentata un'architettura adatta alla ricostruzione ed analisi di rotazioni ottenute da piattaforma Arduino. Inoltre si è evidenziato come i risultati ricavati possono variare a seconda del tipo di soluzioni adottate, e si è mostrato come queste possono essere migliorate in modo da ottenere ricostruzioni più verosimili al modello analizzato.
In attesa di aspettare nuove sviluppi delle librerie i2cdev utilizzate per il controllo della piattaforma Arduino, si può pensare di aggiungere altri sviluppi collegati all'architettura proposta, sempre nell'ambito della ricostruzione di movimenti compiuti da un'auto e ottenuti tramite sensori MEMS. Come è stato detto nelle caratteristiche architetturali, il progetto è stato costruito per essere flessibile e adeguarsi a nuovi esperimenti ed espansioni.
Fino ad ora si è pensato alla ricostruzione ed analisi delle rotazioni di un giroscopio, analizzando quindi la bontà degli algoritmi sviluppati, senza specificare come impiegare i risultati ottenuti.
Una idea che non abbiamo avuto tempo di implementare è la ricostruzione di un movimento di un'auto integrando quando possibile le posizioni assunte con un segnale GPS, in modo da poter visualizzare su una mappa anche la locazione dell'auto oltre alla sua rotazione.
Ci sono state diverse idee considerate all'inizio dello sviluppo dell'architettura che potrebbero essere riprese in mano per essere integrate nel progetto; una di queste riguarda l'utilizzo di un dispositivo Android collegato all'apparecchio Arduino. Questo tipo di soluzione porterebbe all'utilizzo del dispositivo da remoto, senza dover controllare i risultati con un computer nelle vicinanze. Oltretutto questa configurazione sarebbe vantaggiosa per l'invio dei dati di spostamento, magari attraverso un sistema di comunicazione GPRS (General Packet Radio Service) della rete cellulare per poi agganciarsi a internet; in questa maniera si potrebbero far pervenire i dati di movimento in modo da poterli analizzare in contemporanea con altri

apparecchi simili. Quanto esposto sarebbe utile per la ricostruzione della dinamica di un incidente stradale coinvolgendo i dati ottenuti da più auto, questo con lo scopo di poter individuare le cause scatenanti.

Una ulteriore evoluzione dell'architettura proposta potrebbe riguardare l'analisi dei movimenti di un auto in modo da determinare il modo di guidare di un conducente. Questo sarebbe utile per determinare diversi stili di guida più o meno sicuri e propensi a provocare incidenti.

Un'altra possibile idea consisterebbe nel non soffermarsi ad utilizzare una piattaforma Arduino, ma addirittura di poter analizzare altri tipi di dispositivi od addirittura veicoli dotati di giroscopio. Questo è possibile poiché la piattaforma Arduino è solo una parte dell'architettura creata, quindi può essere sostituita con altro hardware.

Figura 7.1a

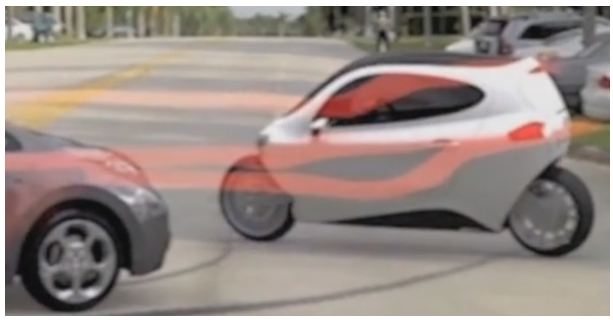

Figura 7.1b

Figura 7.2

Esistono già diversi mezzi di trasporto che montano giroscopi che gestiscono la loro tenuta ed il loro equilibrio insieme al conducente. Per questo basti pensare a scooter in grado di stabilizzare la loro posizione e quindi capaci di non ribaltarsi (figure 7.1), oppure ad un ormai comune monopattino a motore come mostrato in figura 7.2.

Con una visione più ampia, si potrebbero quindi pensare anche a delle applicazioni che fuoriescono dalla ricostruzione di dinamiche di incidente, ed in grado di poter diminuire i danni e gli infortuni stabilizzando il mezzo di trasporto nel caso in cui il conducente ne perda il controllo.

BIBLIOGRAFIA

[Fow04]	Fowler M., *UML Distilled - Terza edizione,* Milano, Pearson Education Italia, 2004
[Inv1]	InvenSense, *MPU-6000 and MPU-6050 Product Specification Revision 3.4,* 19 agosto 2013
[Inv2]	InvenSense, *MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2,* 19 agosto 2013
[Ard13]	Arduino, Language Reference, http://arduino.cc/en/Reference, 20.11.2013
[Ban09]	Banzi M., *Getting Started with Arduino,* O'Reilly Media*,* 2009
[Sma11]	Schmidt, Maik*, Il manuale di Arduino*, Apogeo, 2011
[MMi11]	Margolis, Michael, *Arduino progetti e soluzioni,* Tecniche Nuove, 2011
[Ban12]	Banzi M., *Arduino La guida ufficiale,* Tecniche Nuove, 2012
[I2C13]	I2Cdevlib, MPU6050 Class Reference, http://www.i2cdevlib.com/docs/html/class_m_p_u6050.html, 20.12.2013
[Kui02]	Kuipers J. B., *Quaternions and Rotation Sequences,* Princeton, Princeton University Press, 2002
[Han06]	Hanson Andrew J., *Visualizing Quaternions*, San Francisco, Morgan Kaufmann, 2006

[CSm03]	Conway J.H., Smith D., *On Quaternions and Octonions*, A K Peters, 2003
[DLa03]	Dorian C., Lasenby A., *Geometric Algebra for Physicists*, Cambridge, Cambridge University Press, 2003
[DFM07]	Dorst L., Fontijne D., Mann S., *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry,* San Francisco, Morgan Kaufmann, 2007
[Gaz13]	Gaztanaga I., Sharing memory between processes, http://www.boost.org/doc/libs/1_55_0/doc/html/interprocess/sharedmemorybetweenprocesses.html, 27.12.2013
[Mic13]	Microsoft, SerialPort Class, http://msdn.microsoft.com/en-us/library/system.io.ports.serialport, 27.11.2013
[Kra03]	Kravchenko V., *Applied Quaternionic Analysis*, Heldermann Verlag, 2003
[Vin08]	Vince J. A., *Geometric Algebra for Computer Graphics*, Springer, 2008
[MLS98]	Mac Lane, Saunders, *Categories for the Working Mathematician*, New York, Springer-Verlag, 1998
[Asi86]	Altmann, Simon L., *Rotations, quaternions, and double groups*, New York, Oxford University Press, 1986