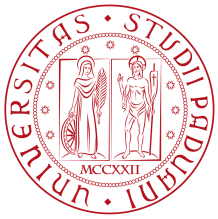
Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei Sistemi Industriali



Università
degli Studi
di Padova

Sede Amministrativa: Universitá degli Studi di Padova

—

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

—

Scuola di Dottorato di Ricerca
in
Ingegneria Meccatronica e dell'Innovazione del Prodotto

ciclo xxvii

STUDIO ED OTTIMIZZAZIONE DI SISTEMI DI AUTOMAZIONE FLESSIBILE

Direttore della Scuola: Ch.mo Prof. Ing. Alessandro Persona

Supervisore: Ch.mo Prof. Ing. Aldo Rossi

Dottorando: Simone Minto

alla mia famiglia...

"One equal temper of heroic hearts,
Made weak by time and fate, but strong in will
To strive, to seek, to find, and not to yield"

Alfred Tennyson, Ulysses, 1833

Indice

Sommario

Al giorno d'oggi il mercato è caratterizzato da prodotti sempre più vari e con cicli di vita sempre più brevi. Per adattarsi a questo contesto e per restare competitive a livello internazionale, molte aziende stanno adottando dei sistemi di assemblaggio flessibili (FAS). Tali sistemi devono garantire anche un'elevata produttività ed un basso costo unitario diretto, il che richiede lo sviluppo di nuove tecnologie per i moderni sistemi di assemblaggio. Oltre alla flessibilità di prodotto, ossia la possibilità di poter gestire diversi componenti con la stessa automazione, vi è una forte esigenza in ambito industriale di disporre di strumenti hardware e software al fine di poter aumentare la flessibilità di layout di una cella robotizzata, ovvero la caratteristica di poter riconfigurare l'ambiente di lavoro in tempi rapidi. Infatti, ad ogni modifica della planimetria di una cella di lavoro, ad esempio per la sostituzione di un componente con ingombri differenti, l'integrazione di ulteriori moduli o sottosistemi, è necessario compiere determinate operazioni come la ridefinizione dei punti di via del manipolare, la ridefinizione del ciclo di lavoro e la ricalibrazione della sensoristica utilizzata. Inoltre, è necessario porre molta attenzione al problema delle collisioni del robot, data la presenza di eventuali nuovi ostacoli. Tutte queste operazioni richiedono personale specializzato, tempo, con un conseguente fermo della produzione. In questa tesi di dottorato, focalizzata nella flessibilità di layout, sono state studiate ed implementate alcune soluzioni per rendere più rapide ed intuitive, senza la necessità di ricorrere a personale altamente specializzato, le operazioni di aggiornamento della configurazione di una cella robotizzata. Le soluzioni sviluppate riguardano procedure rapide per la calibrazione di telecamere industriali, strumenti per l'acquisizione tridimensionale dell'ambiente di lavoro e l'importazione dello stesso in un ambiente CAD commerciale. Un

simulatore robotico permette la navigazione off-line di un manipolatore nella geometria acquisita, la memorizzazione off-line delle traiettorie robot e dei punti di via, oltre che alla gestione dei dati nel ciclo di lavoro in tempo reale all'interno dell'ambiente CAD. A fronte di alcuni studi condotti nel laboratorio Rehabilitation And RObotics (ROAR) presso la Columbia University in the city of New York, NY, USA, sotto la supervisione del Prof. S. K. Agrawal si descrive un dispositivo di scarpa strumentata in grado di fornire un feedback audio-tattile e misurare parametri spazio-temporali della camminata, che potrebbero essere utilizzati per stimare la posizione di un operatore all'interno di una cella robotica a fini di sicurezza. Infine, per la movimentazione rapida ed intuitiva di robot industriali si presenta la progettazione e realizzazione di un dispositivo impugnabile, basato su sensoristica inerziale MEMS.

Abstract

Nowadays, the market is characterized by products which are increasingly varied and with short life cycles. To adapt to this environment and to remain competitive on an international level, many companies are adopting the use of flexible assembly systems (FAS). These systems must also guarantee a high productivity at a low unit cost, which requires the development of new technology for modern assembly systems. In addition to flexibility of the product, that is the possibility to manage different components with the same automation, there is a strong need on an industrial level to arrange hardware and software in order to increase the flexibility of layout of robotized cells, with features of being able to reconfigure the workplace in a short period of times. Infact, with each amendment to the layout of the work-cell, for example to substitute components with different obstacles, the integration of further modules and subsystems, it is necessary to complete certain operations like the redefinition of the via point, the redefinition of the work cycle and the recalibration of the sensors utilized. Furthermore, it is necessary to pay a great deal of attention to the problem of collisions of the robot, given the presence of potentially new obstacles. All of these operations require specialized personnel, time, and subsequently an interruption in production. In this PhD Thesis, focusing on the flexibility of layout, some solutions have been studied and implemented in order to achieve more rapid and more intuitive operations of updating of the configuration of a robotized cell, without needing to use highly specialized personnel. The solutions developed regard the rapid procedures for the calibration of industrial television-cameras, tools for a three-dimensional overview of the workplace and the importation of such in the environment of commercial CAD. A robotic simulator allows off-line navigation of the manipulator in the

acquired geometry, the off-line memorization of the robot trajectories and via points, in addition to handling of data of the work cycle in real time inside of the CAD space. In the face of some studies conducted in the Rehabilitation And Robotics (ROAR) laboratory at Columbia University in the city of New York, NY, USA, under the supervision of Prof. S. K. Agrawal, a shoe device is described which is able to provide feedback of audio and tactile sensory information nature giving spacial-time measurements of the walk which could be utilized to estimate the position of the operator inside of a robotic cell for safety applications. In conclusion, for rapid and intuitive movement of industrial robots, the design and making of a impugnable device based on MEMS sensors is presented.

Introduzione

L'idea di proporre architetture e metodologie di sviluppo software ed hardware innovativi per il controllo di celle robotiche, ovvero che abbiano la possibilità di riconfigurarsi in modo semplice, rapido e senza la necessità di personale specializzato, si accompagna alla crescente necessità da parte delle aziende manifatturiere di commissionare ad enti di ricerca studi per la progettazione di impianti di automazione industriale flessibili.

Il **primo capitolo** è introduttivo agli impianti automatici flessibili per l'automazione industriale. Vengono illustrate le differenze principali che contraddistinguono il concetto di flessibilità di prodotto e di layout oltre che ad una breve descrizione dei sottosistemi presenti in una cella robotica.

Nel **secondo capitolo** viene illustrato un framework software sviluppato, descrivendo rapidamente la soluzione implementata per la calibrazione rapida di telecamere industriali e per la scansione tridimensionale di una cella robotica.

Il **terzo capitolo** descrive le procedure di importazione della cella acquisita e l'interfacciamento con un CAD commerciale per la movimentazione di un robot industriale.

Nel **quarto capitolo** è descritta la principale attività di ricerca effettuata presso la Columbia University in the city of New York, descrivendo lo sviluppo di una scarpa strumentata in grado anche di misurare alcuni parametri spazio-temporali della camminata, che potrebbero essere utilizzati per stimare la posizione di un

operatore all'interno dell'ambiente di lavoro, per fini di sicurezza.

Il **quinto capitolo** presenta la realizzazione ed il controllo di un dispositivo I/O per la rapida ed intuitiva movimentazione di robot antropomorfi.

Capitolo 1

Gli impianti automatici flessibili per l'automazione industriale

1.1 Robotica Industriale

Affinché un'azienda industriale possa avere successo nel mondo attuale, altamente competitivo, un aspetto essenziale è rappresentato dall'aumento della produttività. Un modo per fare ciò è quello di velocizzare il ciclo di lavoro tramite l'automazione delle catene produttive, la quale consente di gestire i processi e i macchinari tramite appositi sistemi di controllo. Tale approccio permette inoltre di ridurre l'apporto di manodopera e migliorare la qualità del prodotto finale. Per tali motivi l'automazione è attualmente implementata nella maggior parte dei sistemi produttivi. Un sistema automatizzato di produzione consiste in una serie di macchinari che gestiscono la movimentazione del prodotto e le sue successive lavorazioni fino all'ottenimento del prodotto finito. Questo consente una lavorazione accurata e ripetibile assicurando al prodotto finale caratteristiche standard riducendo, inoltre, il numero di scarti. La progettazione e realizzazione di una linea industriale apposita, o macchina dedicata, per un determinato ciclo di produzione richiede un investimento notevole. Un elevato indice di ritorno dell'investimento (ROI) e la durata pluridecennale del sistema ne giustificano tuttavia il costo iniziale.

La progettazione di un sistema automatizzato dedicato consente di produrre un determinato prodotto finito sempre uguale a sé stesso per l'intero tempo di utilizzo

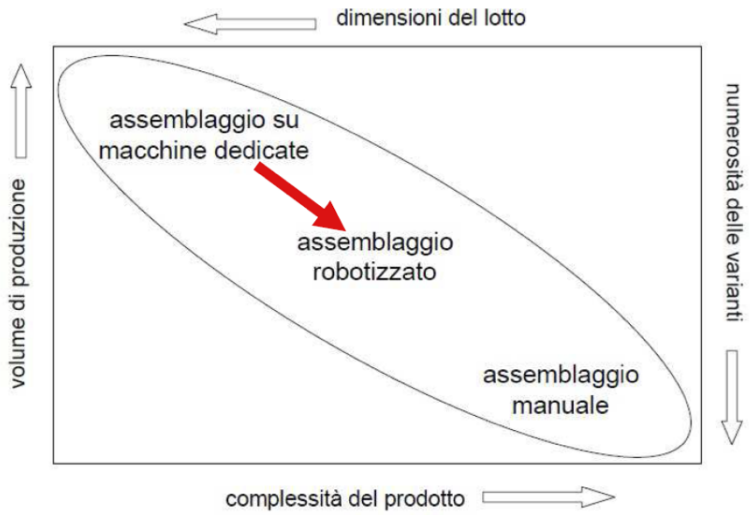


Figura 1.1: Tecnologie di assemblaggio in funzione dei parametri di produzione. La freccia scura rappresenta il trend attuale verso lotti di dimensione ridotta e numerose varianti del prodotto finale.

del sistema di produzione. Un sistema siffatto ottiene una resa elevata a scapito della flessibilità e risulta perciò sconveniente in una situazione di produzione limitata nel tempo o con necessità di frequenti modifiche al processo. Il trend attuale in ambito industriale consiste nella produzione di lotti di dimensioni ridotte e con notevole varietà nel prodotto: in tale scenario la realizzazione di macchinari dedicati non rappresenta la scelta ottimale.

La flessibilità in un sistema industriale automatizzato viene ottenuta tramite l'utilizzo di celle di lavoro robotizzate anziché macchinari dedicati. La gestione dell'automazione in celle robotizzata è nota come robotica flessibile.

1.2 Robotica Flessibile

La tendenza attuale in ambito industriale è determinata dalla realizzazione di sistemi di produzione formati da una successione di singole unità di base robotizzate estremamente flessibili. Una cella robotizzata è un sistema che contiene uno o più robot ed un numero di dispositivi ausiliari dipendente dalle necessità produttive. I robot utilizzati nelle celle sono manipolatori industriali, macchine programmabili

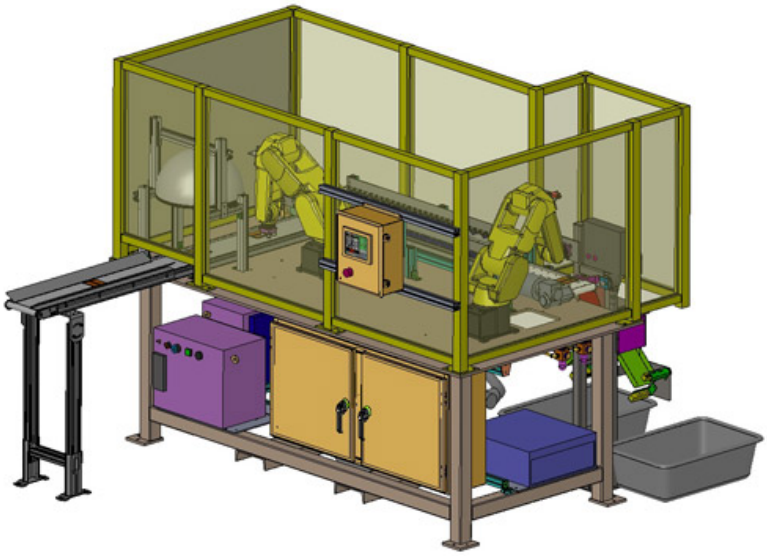


Figura 1.2: Esempio di cella robotizzata. Design per una cella di ispezionamento in una catena di produzione alimentare.

e pertanto in grado di effettuare operazioni molto differenti fra di loro. Per la maggior parte delle applicazioni è utilizzato un robot singolo come semplice *handling device*, ovvero per gestire la movimentazione dei pezzi nella cella: al robot è assegnato il solo compito di prelevare e depositare i componenti sui macchinari disposti all'interno della cella. Una cella robotizzata è un ambiente estremamente flessibile: è sufficiente riprogrammare il manipolatore e variare la disposizione delle postazioni all'interno della cella per soddisfare eventuali nuove esigenze del ciclo produttivo. I componenti della cella possono essere in gran parte riutilizzati, riducendo così il problema della progettazione ad hoc dell'attrezzatura che si presenta nel caso di macchine dedicate. In particolare il manipolatore, il relativo controller e il quadro elettrico sono riutilizzabili. Essendo questi i componenti di costo maggiore è comprensibile la tendenza verso l'utilizzo della robotica flessibile.

1.2.1 Flessibilità di prodotto

La crescente necessità di far fronte a produzioni industriali caratterizzate da numerose personalizzazioni e cicli produttivi estremamente brevi richiede elevata flessibilità dei sistemi di produzione e assemblaggio. Molte aziende cercano di

progettare e controllare i propri sistemi produttivi in modo da potersi adattare velocemente alle richieste del mercato in termini di volumi produttivi e di costi operativi. Diversi ricercatori hanno sviluppato sistemi di assemblaggio flessibile (FAS) in grado di soddisfare le richieste di flessibilità pur mantenendo elevate produttività ([1, 2, 3]). Di recente, alcuni autori hanno proposto innovativi sistemi di alimentazione flessibili basati su sistemi di visione per i quali la flessibilità è garantita dalla possibilità di riconoscere e manipolare oggetti di diverse forme e dimensioni [4, 5]. Il sistema di visione svolge un ruolo fondamentale nel riconoscere gli oggetti presenti sul piano di alimentazione e nel guidare il robot verso l'esatta posizione di quest'ultimi. Successivamente, altri autori svilupparono sistemi flessibili analoghi per applicazioni di ispezione [6, 7], controllo dimensionale [8] o di controllo qualità[9].

Recentemente, alcuni ricercatori hanno introdotto un innovativo sistema di assemblaggio flessibile chiamato "Fully-flexible assembly system" (F-FAS)[10, 11, 12]. Tale sistema consiste in una stazione robotizzata di assemblaggio "mixed-model" che offre la possibilità di produrre una famiglia di prodotti su una stessa linea. La stazione è caratterizzata da un alimentatore flessibile che è responsabile della distribuzione di tutti i componenti necessari all'assemblaggio garantendo così un livello di flessibilità di gran lunga superiore rispetto al tradizionale FAS. Il sistema di alimentazione dei componenti è composto da un sistema di carico, un piano vibrante ed una videocamera in grado di alimentare componenti molto diversi fra loro per forma e dimensione senza bisogno di riconfigurazione. Tuttavia, il tempo necessario per acquisire ed elaborare le immagini, pur essendo parzialmente mascherato, limita la produttività. Un tipico ciclo di lavoro di un sistema F-FAS inizia con il carico dei componenti sul piano e la successiva vibrazione per distribuirli su di esso. A questo punto viene acquisita un'immagine ed elaborata al fine di identificare i componenti che possono essere successivamente assemblati. Dal momento in cui il processo di alimentazione è stocastico, l'insieme di componenti disposti sul piano vibrante possiede una composizione del tutto casuale. Inoltre, può accadere che non tutti i componenti disposti sul piano siano riconosciuti e afferrabili a causa dei limiti del sistema di visione dovuti alla sovrapposizione di componenti oppure all'orientazione dei componenti che rende la presa irrealizzabile.

Queste tipologie di problematiche limitano la produttività del sistema aumentando così il costo unitario di produzione. Quando i componenti da manipolare non possono essere disposti su di un piano vibrante, è necessario disporre di un sistema in grado di ricostruire la geometria tridimensionale dei pezzi in modo tale da poter elaborare una strategia di riconoscimento e presa. Diversi ricercatori hanno sviluppato sistemi di questo tipo caratterizzati da una o più telecamere e sensori laser per la ricostruzione tridimensionale delle superfici [13].

1.2.2 Flessibilità di layout

Oltre alla flessibilità di prodotto, ossia la possibilità di poter gestire diversi componenti con la stessa automazione, vi è la crescente necessità di poter modificare in tempi estremamente ridotti la configurazione della cella oppure il ciclo operativo che la contraddistingue. Questa innovativa caratteristica nell'ambito dell'automazione viene definita flessibilità di layout e ricercatori ed aziende stanno sviluppando sistemi hardware e software sempre più avanzati in grado di aumentarne l'efficacia [14]. L'obiettivo principale della flessibilità di layout consiste nella possibilità di riconfigurare l'ambiente di lavoro e il ciclo operativo in tempi rapidi e senza dover interrompere la produzione. Inoltre, ad ogni modifica della configurazione della cella di lavoro o del ciclo operativo potrebbe essere necessario ridefinire le locazioni relative ai movimenti del manipolatore oppure ricalibrare i sensori utilizzati all'interno della cella robotizzata. La programmazione di un sistema robotico industriale, per esempio, è una procedura complessa ed onerosa in termini di tempo e denaro. Al giorno d'oggi, nella pratica industriale, ci sono due principali categorie di programmazione di sistemi robotici: programmazione in linea e programmazione fuori linea. Nella programmazione in linea si utilizza un dispositivo chiamato "teach pendant" per muovere manualmente il robot industriale fino alla posizione e all'orientazione desiderate. La locazione viene quindi registrata ed utilizzata nel ciclo operativo. Il procedimento risulta molto semplice tuttavia richiede di fermare la produzione per eseguire le operazioni di programmazione. Al contrario, la programmazione fuori linea permette di definire le locazioni ed il ciclo operativo grazie ad un software di simulazione del robot e dell'ambiente che lo circonda. Rispetto alla programmazione in linea, permette di non interrompere la

produzione e di avere indicazioni sul tempo ciclo in fase progettuale [15, 16]. Tale metodologia risulta estremamente flessibile tuttavia presenta alcune problematiche come il costo elevato, la ricostruzione tridimensionale dell'ambiente operativo e la difficoltà di simulare il comportamento di altri sistemi integrati all'interno della cella di lavoro.

Oltre alle tecniche per riconfigurare rapidamente una cella, una delle soluzioni più interessanti per incrementare la flessibilità in ambiente industriale consiste nell'idea di combinare le capacità manuali di un operatore con le potenzialità tipiche di sistemi robotici per consentire una collaborazione efficace. Al giorno d'oggi lo spazio operativo in cui operano sistemi ad elevata automazione è marcatamente separato dallo spazio operativo in cui può muoversi un operatore umano. Recentemente, la normativa che prescrive i requisiti di sicurezza per robot industriali e per l'integrazione di sistemi robotici [17, 18] ha introdotto la possibilità di collaborazione uomo-robot. In particolare, alcuni autori stanno lavorando a sistemi robotici in grado di percepire la presenza umana e attuare delle procedure per evitare eventuali collisioni [19, 20]. Lo sviluppo di sistemi di visione e di sensori inerziali a basso costo permette di implementare diverse tecniche per identificare la posizione ed il movimento di un operatore. Dal punto di vista del mercato, le aziende produttrici di sistemi robotici hanno iniziato a sviluppare nuovi prodotti caratterizzati da ridotte masse e inerzie ed equipaggiati con sensori di coppia e di pressione estremamente sensibili per individuare eventuali collisioni impreviste. Lo stato dell'arte rivela che l'interazione uomo-robot è un campo di ricerca potenzialmente ricco di applicazioni e in rapida evoluzione. Lo sviluppo di sistemi e algoritmi che permettano la cooperazione uomo-robot in sicurezza è un aspetto fondamentale che richiede ancora molto sviluppo.

1.3 Sottosistemi di una cella robotica flessibile

Il termine *cella robotica* viene utilizzato per identificare un'unità base in un processo complesso. Più celle indipendenti possono essere disposte in successione per realizzare il ciclo produttivo e ottenere il prodotto finito. Nelle successive sottosezioni verranno illustrati i principali componenti di una cella robotica

flessibile.

1.3.1 Struttura portante

Il primo e fondamentale sottosistema che costituisce una cella robotica flessibile è la struttura di supporto dell'intera cella robotizzata. Essa permette di sorreggere e fissare saldamente gli altri sottosistemi, limitando la trasmissione delle vibrazioni. L'utilizzo di profili di alluminio come sostegno ad alcuni sottosistemi è largamente diffuso nel mondo industriale proprio per le caratteristiche di leggerezza, facilità di montaggio e possibilità di modificare velocemente la loro locazione durante la fase di prototipazione della cella di lavoro.

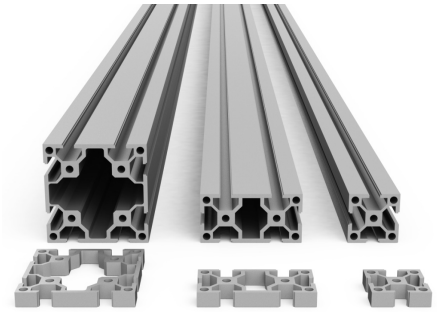


Figura 1.3: Profili in alluminio comunemente utilizzati

1.3.2 Manipolatori ed organi di presa

I manipolatori più frequentemente utilizzati in celle robotizzate sono i robot antropomorfi a 6 assi e i robot di tipo SCARA, *Selective Compliance Assembly Robot Arm*, caratterizzati da 4 gradi di libertà. Il numero di assi, o gradi di libertà, indica la molteplicità di giunti indipendentemente controllabili. Il manipolatore SCARA è tipicamente utilizzato per le semplici operazioni di pick-and-place che prevedono l'afferraggio ed il prelievo di pezzi con movimenti perpendicolari al piano di lavoro. I robot antropomorfi consentono una più ampia gamma di operazioni. Sulla flangia robot è solitamente previsto l'utilizzo di una pinza singola, ovvero un end-effector in grado di effettuare presa e rilascio di un singolo componente alla volta. Le pinze industriali sono costituite da una flangia con annesse valvole

Figura 1.4: Esempi di manipolatori industriali. Robot a 6 assi Adept Viper s650 (sinistra) e robot a 4 assi Adept Cobra s800 di tipologia SCARA (destra).

pneumatiche o oleodinamiche controllate tramite segnali output del controller. Il gruppo pinza viene in seguito reso solidale alla flangia robot tramite appositi fori disposti sulla flangia robot dal produttore di manipolatori.

1.3.3 Sistemi di alimentazione dei componenti

Una postazione di prelievo pezzi, o feeder, permette di rendere disponibili i componenti al manipolatore. I componenti vengono forniti in numero limitato e, in seguito al prelievo dell'ultimo di essi, è necessario attendere un tempo di rinnovo prima di disporre di ulteriori componenti. Il feeder flessibile si contrappone ai tradizionali *bowl-feeder*, dispositivi in grado di fornire costantemente un pezzo al manipolatore, garantendo per di più posizione e orientamento desiderati. Questi ultimi sono dispositivi che consistono in tazze vibranti riempite con componenti alla rinfusa da un operatore o da sistemi ausiliari automatizzati. Tramite un apposito moto sussultorio i componenti risalgono lungo un percorso a spirale presente all'interno della tazza. Lungo tale percorso sono disposte delle trappole appositamente progettate. Tramite esse i componenti che presentano un'orientazione non conforme a quella desiderata vengono riconosciuti e obbligati a cadere nuovamente al centro della tazza. I bowl-feeder sono dispositivi che richiedono una progettazione apposita a seconda del componente movimentato, di conseguenza costosi e per nulla flessibili. L'utilizzo di feeder flessibili consente di ridurre il costo di realizzazione della cella robotizzata e utilizzare la medesima strumen-

tazione nell'eventuale riconfigurazione del ciclo di lavoro. Un esempio di feeder flessibile è un semplice nastro trasportatore: a monte di esso un operatore carica i componenti. Solo un'estremità raggiunge l'interno della cella, consentendo un agevole caricamento esterno senza richiedere la presenza dell'operatore nella cella. Un sistema di visione solitamente inquadra la porzione di nastro interna alla cella, individua i componenti e fornisce al controller le relative coordinate per consentirne la presa. Ciclicamente viene richiesto l'avanzamento del nastro di una quantità fissata. Il numero di componenti presenti ad ogni rinnovo del feeder, o in questo caso avanzamento del nastro, può variare a seconda della tipologia di caricamento a monte e a seconda dell'effettivo riconoscimento da parte del sistema di visione. Un ulteriore esempio di feeder flessibile è dato dall' alimentazione di componenti tramite pallet distinti.

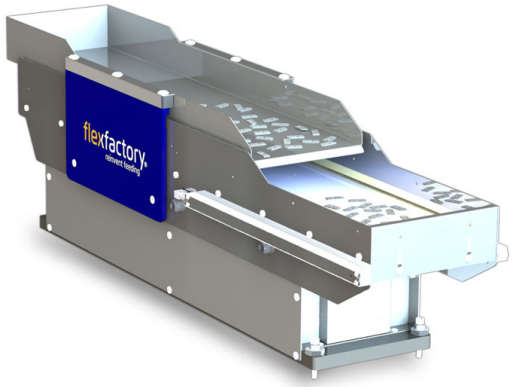


Figura 1.5: Feeder flessibile prodotto dall'azienda Flexfactory

1.3.4 I sistemi di sicurezza

Precise norme [21] regolano e stabiliscono i principi generali affinché i costruttori mettano in commercio prodotti o sistemi che non siano pericolosi per gli operatori. Ciascuna cella è fisicamente caratterizzata da una struttura chiusa solitamente inagibile per gli operatori. Per motivi di sicurezza, infatti, l'ambiente attorno al manipolatore è chiuso da strutture di profilati e pannelli di plexiglass. L'apertura di una qualsiasi porta della cella impone l'arresto delle attività del robot per garantire la sicurezza degli operatori. Dispositivi di arresto meccanico devono essere

in grado di fermare il movimento del robot con il carico nominale, alla massima velocità ed alla minima e massima estensione. Dispositivi di arresto non meccanico, fermi posizionati elettricamente, pneumaticamente o idraulicamente, barriere fotoelettriche, laser scanner, possono essere utilizzati solamente se garantiscono lo stesso livello di sicurezza dei dispositivi di arresto meccanico.

1.3.5 Il sistema di controllo

Il sistema di controllo di una cella robotica deve essere in grado di gestire, coordinare e monitorare il manipolatore industriale e tutti i componenti inclusi quali, ad esempio, i sistemi di visione, i PLC, gli alimentatori di componenti oltre che a svolgere i calcoli necessari al riconoscimento ed alla manipolazione dei componenti. Strategie implementative del controllo possono ricadere sulla realizzazione di sistemi sequenziali o basati su macchine a stati finiti; quando, invece, non vi è un ciclo di lavoro stabilito a priori ci si può riferire a sistemi di sequenziamento dinamico. In questa implementazione, i comandi impartiti al robot vengono stabiliti al termine di ciascuna operazione a seconda della situazione nella cella. Nel codice che gestisce la cella viene assegnata una priorità per ciascuna operazione e, in base alla disponibilità istantanea delle postazioni, si determina la seguente operazione del robot. Tale gestione viene ottenuta analizzando ciclicamente la situazione dei segnali forniti dai vari sistemi implementati nella cella. A livello pratico le postazioni presenti all'interno di una cella dialogano con il controllore tramite PLC o collegamento diretto al controller. Una postazione può ad esempio assumere gli stati occupata/disponibile, caso tipico di un macchinario di lavorazione che stia lavorando o meno un componente.

1.3.6 I sistemi di visione

Per sistema di visione si intende un insieme di componenti hardware e software capaci di acquisire dall'ambiente circostante delle informazioni sfruttando fenomeni di tipo ottico. L'hardware è generalmente costituito da un sistema di illuminazione dell'ambiente di lavoro, da una o più telecamere per la ripresa delle immagini, da un'elettronica per la loro acquisizione e da un elaboratore per il loro post-

Figura 1.6: Telecamera Allied Vision Pike F-505

processing. Il software è formato dai programmi che elaborano l'immagine per ricavarne le informazioni desiderate che possono avere varie finalità: di controllo, classificazione, selezione, ecc.

Capitolo 2

Moduli software sviluppati

In questo capitolo si riporta il framework software sviluppato per poter utilizzare un sistema di sviluppo prettamente prototipale con successo in ambito industriale, al fine di sopperire ad alcune limitatezze dell'ambiente di sviluppo stesso. Si riportano brevemente le descrizioni dei moduli implementati per la calibrazione di telecamere e per la scansione tridimensionale di una cella di lavoro robotica, in quanto ampiamente trattati in letteratura. Tali componenti software hanno richiesto un'intensa parte implementativa e di ottimizzazione al fine di rendere il loro utilizzo rapido e flessibile nell'ambiente dell'automazione industriale.

2.1 Framework software flessibile

Matlab, sviluppato dalla società MathWorks, è un linguaggio ad alto livello, dinamico, interpretato, non multithread, dotato di una vastissima libreria di funzioni e toolbox, con caratteristiche di programmazione orientata agli oggetti; esso è utilizzato in moltissimi ambiti dell'ingegneria. Oltre che ad essere un versatile ambiente per il calcolo numerico e l'analisi statistica permette la creazione di interfacce grafiche (GUI) in maniera rapida e versatile. A differenza dei linguaggi interpretati, i compilatori di linguaggi come il C ed il Foltran traducono il codice in istruzioni macchina, solitamente con prestazioni decisamente superiori. Matlab offre la possibilità di utilizzare codice scritto in C/C++ o Fortran scrivendo degli opportuni MEX-files (MATLAB Executable) che successivamente dovranno essere compilati, ottenendo quindi le prestazioni di tali linguaggi. Un'esaustiva documentazione al riguardo si può trovare in [22]. In ambito industriale vengono utilizzati,usualmente, hardware dedicati con sistemi operativi real-time, contraddistinti da alte prestazioni ma anche dal difficile sviluppo. L'idea di usare l'ambiente di sviluppo Matlab, prototipale, in ambiente industriale permette il rapido sviluppo di applicazioni di controllo per celle robotiche automatizzate, fornendo quindi un'elevata flessibilità. Per far fronte ad alcune limitatezze prestazionali, ad esempio, dei toolbox *Instrument Control Toolbox* (per la gestione della comunicazione seriale, TCP/IP ed UDP) e *Image Acquisition Toolbox* (dedicato all'acquisizione di immagini da telecamere) e per rendere maggiormente stabili e performanti componenti software a basso livello, sono stati sviluppati diversi moduli, che permettono:

- rapida acquisizione di immagini da telecamere industriali con relativa gestione dei trigger hardware al fine di poter acquisire fotogrammi anche in background;
- comunicazione efficiente e stabile con manipolatori industriali e PLC;
- interfacciamento del dispositivo aptico, con force feedback, Geomagic Phantom per il controllo di robot paralleli,
- connessione a database SQL, mediante protocollo ODBC, per la gestione e monitoraggio del ciclo produttivo;

- interfacciamento con schede di acquisizione dati utilizzate in ambito industriale.

In molte situazioni, il ciclo di controllo potrebbe richiedere il processamento di routine complesse ed onerose dal punto di vista computazionale, dovendo attendere la loro terminazione. Poiché molte operazioni non sono vincolanti per il proseguo del ciclo di controllo principale, possono essere delegate ad un processo parallelo che si occupa di processarle, in tempo mascherato, restituendo successivamente i risultati, quando disponibili. Di conseguenza, per evitare che il ciclo di controllo interrompa la sua attività di scheduling e gestione dei vari sottosistemi, è stata sviluppata un'architettura software basata su due sessioni Matlab in grado di comunicare fra loro attraverso una regione di memoria condivisa. In tal modo, la sessione principale può scambiare strutture dati ed istruzioni con una sessione di supporto in grado di avviare a sua volta processi e funzioni, attuando quindi un supporto per il multitasking.

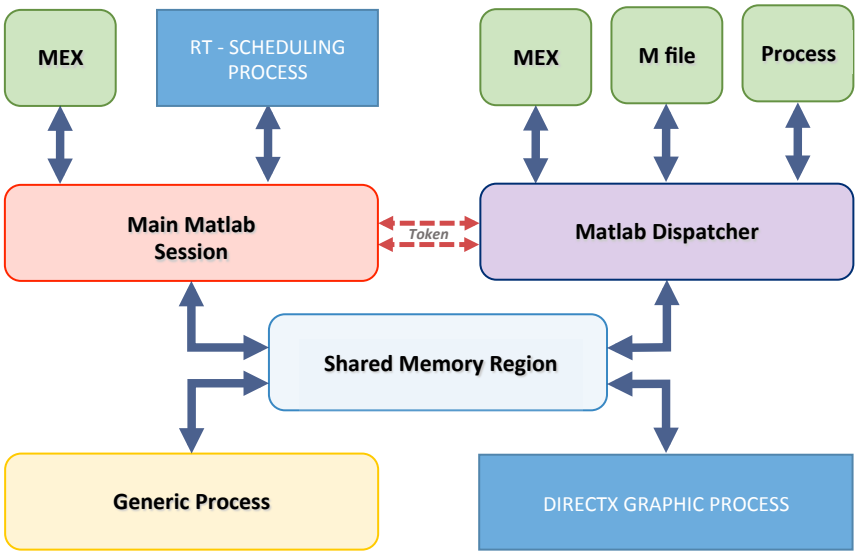


Figura 2.1: Architettura software sviluppata

In Fig. 2.1 è rappresentata una schematizzazione semplificata di quanto appena esposto. Tale framework, utilizzato in un simulatore per un sistema di misura 3D (Fig. 2.2) per il settore della calzatura, sviluppato in precedenza nel laboratorio Mechatronics dell'Università degli Studi di Padova, ha permesso di ridurre i tempi di elaborazione di una scansione di una forma da 180 minuti a 3,5 minuti.

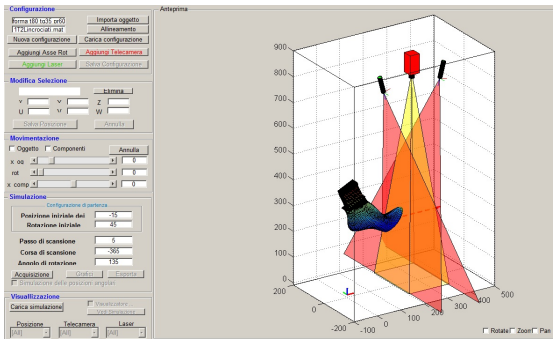


Figura 2.2: Interfaccia utente del simulatore

2.2 Calibrazione rapida di telecamere

Nell'utilizzo di una telecamera sono da tenere presenti gli effetti di distorsione introdotti dall'ottica stessa, da cui nessun obiettivo è realmente immune. Tali distorsioni derivano dall'utilizzo di lenti sferiche che provocano una focalizzazione diversa da quella teorica per i raggi incidenti più distanti dall'asse ottico. Le principali distorsioni ottiche riscontrabili sono l'effetto a barilotto e a cuscino (Fig. 2.3).

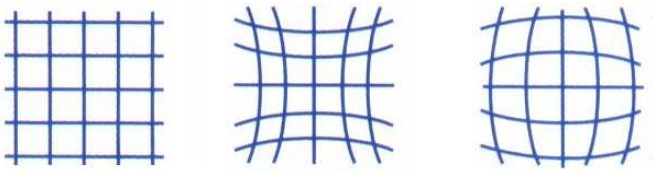


Figura 2.3: Distorsioni ottiche presenti in un obiettivo: nessuna distorsione (a sinistra), distorsione a cuscino (centrale) e distorsione a barilotto (destra).

Molto importante, soprattutto quando si necessita di una buona precisione, è conoscere l'entità di tali distorsioni (radiale e tangenziale) per poter utilizzare un modello correttivo al fine di poter elaborare le immagini distorte ottenendo delle immagini rettificate. La stima dei parametri necessari a correggere l'errore di distorsione si ottiene per mezzo di un procedimento detto *calibrazione* della telecamera.

La tecnica comunemente utilizzata per calibrare la telecamera richiede l'acquisizione di diverse immagini (circa 20-25 [23]) che inquadrino una trama, di dimensioni note, senza aver conoscenza delle posizioni o dell'orientazione della

stessa. La trama utilizzata consiste, solitamente, in una scacchiera di colorazione bianco e nero stampata su carta e posizionata in un piano rigido per evitare la deformazione. Durante la fase di sviluppo di una cella robotica può spesso capitare di dover modificare i piani di lavoro, spostare fisicamente la locazione della telecamera o cambiare obiettivo a quest'ultima: in queste situazioni è necessario effettuare una nuova calibrazione. Un tool largamente utilizzato per la calibrazione è il *Matlab Calibration Toolbox* [23]. Con tale software, dopo aver acquisito diverse immagini del pattern di calibrazione, è necessario estrarre gli angoli della scacchiera in maniera manuale per tutti i frame acquisiti. Al fine di velocizzare tale operazione è stata sviluppata una GUI (Fig. 2.4) per poter effettuare la calibrazione di camere industriali anche ad operatori non esperti. La GUI è composta essenzialmente da un pannello in cui inserire le caratteristiche geometriche della scacchiera e da due pannelli di visualizzazione. Nel pannello di anteprima (di dimensioni inferiori) vi è la rappresentazione in tempo reale (con un subsampling) di ciò che la telecamera sta acquisendo. Nel momento in cui il pattern di calibrazione viene riconosciuto, vengono plottati i corner dei punti di controllo (punto di contatto fra quattro caselle) identificati. A questo punto l'operatore può cliccare il bottone *Find Corners* per effettuare l'operazione di riconoscimento della trama di calibrazione (a piena risoluzione) sull'ultima immagine acquisita dal sistema. Successivamente è possibile inserire il frame al database di immagini che sarà utilizzato per la calibrazione. Dopo aver acquisito un numero sufficiente di immagini, con la pressione del tasto *Calibration* verrà eseguita l'operazione di calibrazione che restituirà i parametri intrinseci, estrinseci e di distorsione della camera. L'algoritmo implementato è dettagliatamente riportato in [24]. L'utilizzo di tale modulo software, in ambiente di produzione, ha permesso di dimezzare il tempo di un operatore nelle operazioni di calibrazioni di telecamere industriali.

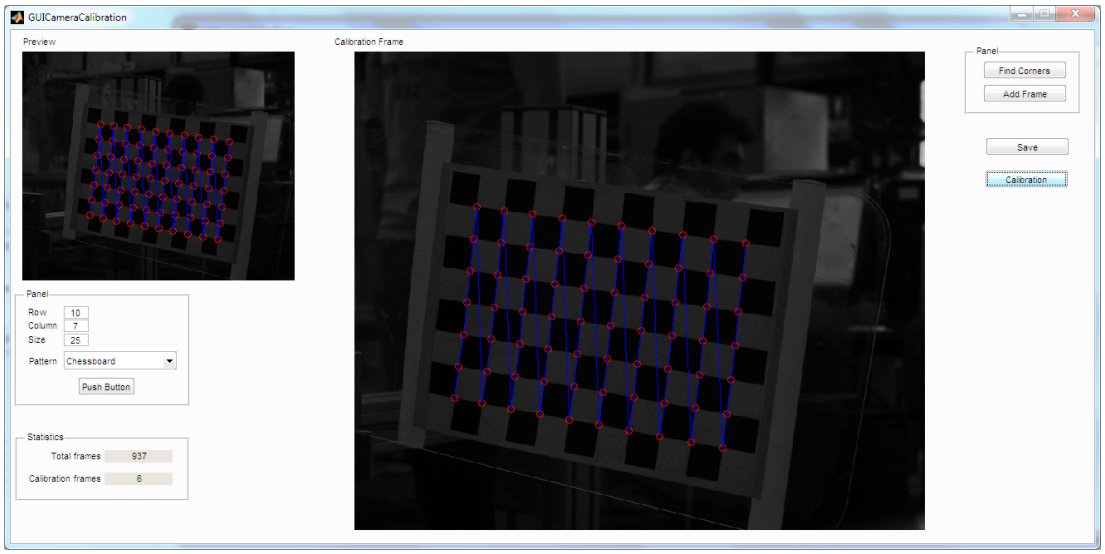


Figura 2.4: GUI per la calibrazione di telecamere 2D

2.3 Scansione tridimensionale di una cella di lavoro

In questa sezione, viene descritto il modulo sviluppato per l'acquisizione della geometria tridimensionale di una cella robotizzata al fine di poterla importare in un ambiente CAD commerciale (Cap.3), mediante l'utilizzo del sensore Microsoft Kinect. La scelta di utilizzo di questo specifico sensore è dettata dal suo sempre più diffuso uso in ambito di ricerca, in settori quali la robotica, sia per il controllo gestuale che per la realtà virtuale. Oltretutto, dispone di peculiari caratteristiche quali: velocità di acquisizione, adeguata precisione nella ricostruzione, facilità di utilizzo e costo ridotto.

Il Microsoft Kinect, raffigurato in Fig. 2.5, è un dispositivo dotato di una telecamera RGB con risoluzione di 640 x 480 pixel, un proiettore IR a luce strutturata da 1.2W, una telecamera IR composta da una matrice di 320 x 240 pixel, un array di microfoni ed una base motorizzata. Attraverso un hardware proprietario prodotto da PrimeSense, il Kinect è in grado di generare immagini depth di risoluzione pari a 640 x 480 pixel (11 bit, 2048 livelli di sensitività) ad una frequenza di 30 Hz, con un range di misura da 40 cm a 4 m. Lo strumento è in grado di generare un'immagine depth andando a rilevare le variazioni indotte

dagli oggetti presenti nella scena nel pattern pseudo-causale proiettato [25].

Figura 2.5: Microsoft Kinect per PC

In Fig. 2.6 sono presenti alcune immagini depth acquisite con il sensore. È possibile osservare la presenza di numerosi buchi dove non è stato possibile rilevare correttamente il pattern IR emesso. Questa situazione può essere causata, come dettagliatamente spiegato in [25], da materiali a bassa o alta riflettività, particolare illuminazione di fondo, superfici eccessivamente inclinate, occlusioni e discontinuità degli oggetti.

Per il processo di ricostruzione tridimensionale è stato utilizzato, ottimizzandolo al particolare ambiente di utilizzo, l'SDK Kinect Fusion studiato e sviluppato presso i laboratori Microsoft Research [26, 27]. La più importante classe degli algoritmi utilizzati sono basati sul concetto di Iterative Closest Point (ICP) introdotto in [28] che pone l'allineamento delle nuvole di punti come un problema di ottimizzazione non lineare in cui le corrispondenze tra le scansioni sono approssimate utilizzando le coppie più vicine dei punti trovati nella scansione avvenuta all'iterazione precedente, tenendo traccia costantemente della posa a 6 gdl della camera. In Fig. 2.7 viene schematicamente rappresentato il workflow del processo di acquisizione.

La procedura di acquisizione prevede che l'operatore impugni il Kinect, come raffigurato in Fig. 2.8 , e si sposti lentamente all'interno dell'ambiente che vuole acquisire. Attraverso una porzione della GUI, visibile in Fig. 2.10, può verificare il risultato dell'acquisizione in tempo reale. Nel caso in cui si verificassero eventi o condizioni che impedissero il proseguimento della ricostruzione, l'operatore viene avvertito da un allarme sonoro e a video sarà raffigurato l'ultimo frame correttamente elaborato: in questo caso l'utente dovrà rimanere per qualche secondo fermo cercando di inquadrare l'ultima scena processata ed attendere che il processo di acquisizione continui.

Figura 2.6: Immagini depth RAW acquisite con il sensore Microsoft Kinect.

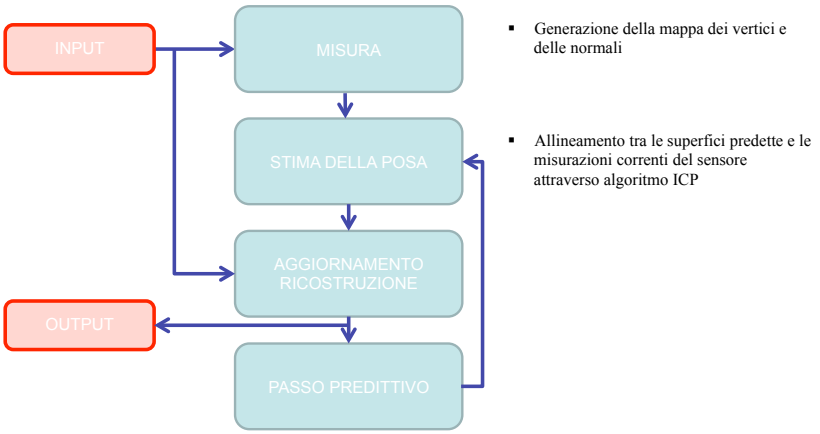


Figura 2.7: Workflow del processo di ricostruzione tridimensionale

La ricostruzione acquista dettagli con l'acquisizione di nuove misure di profondità. Nel caso in cui si desiderasse acquisire maggiori particolari di un determinato componente è necessario semplicemente riprenderlo con la telecamera per maggior tempo e con maggiori viste. I buchi vengono colmati ed il modello diventa sempre più completo e definito nel tempo. Le sottofigure (*h*-*l*) di Fig. 2.10 raffigurano quanto in precedenza esposto: è possibile notare come la spalla del robot acquisti definizione, particolari e volume.

A procedura terminata è possibile esportare il risultato ottenuto in STL binario (STereo Lithography interface format). Con tale formato è possibile descrivere la superficie geometrica di un oggetto tridimensionale discretizzandola in triangoli, eliminando i vertici duplicati non necessari.

In Fig. 2.9 è presente il risultato della scansione ottenuto. La risoluzione spaziale dell'acquisizione è circa 10 mm, mentre il matching in tempo-reale avviene a 22 frame al secondo.

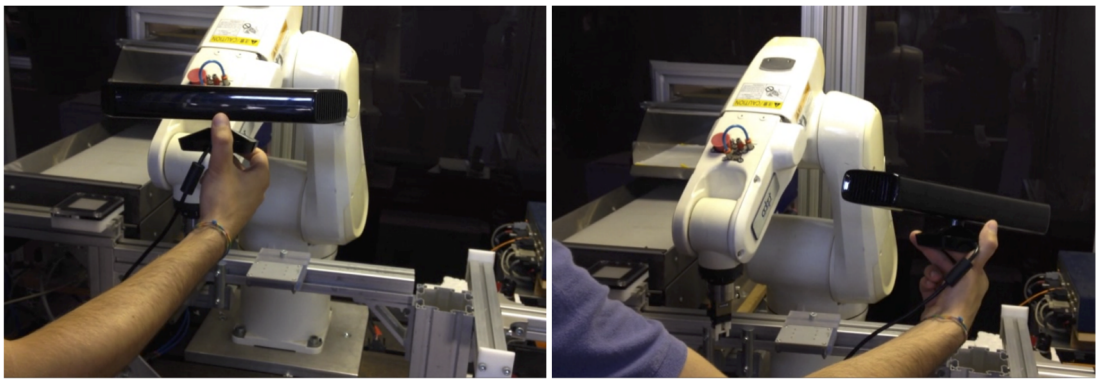
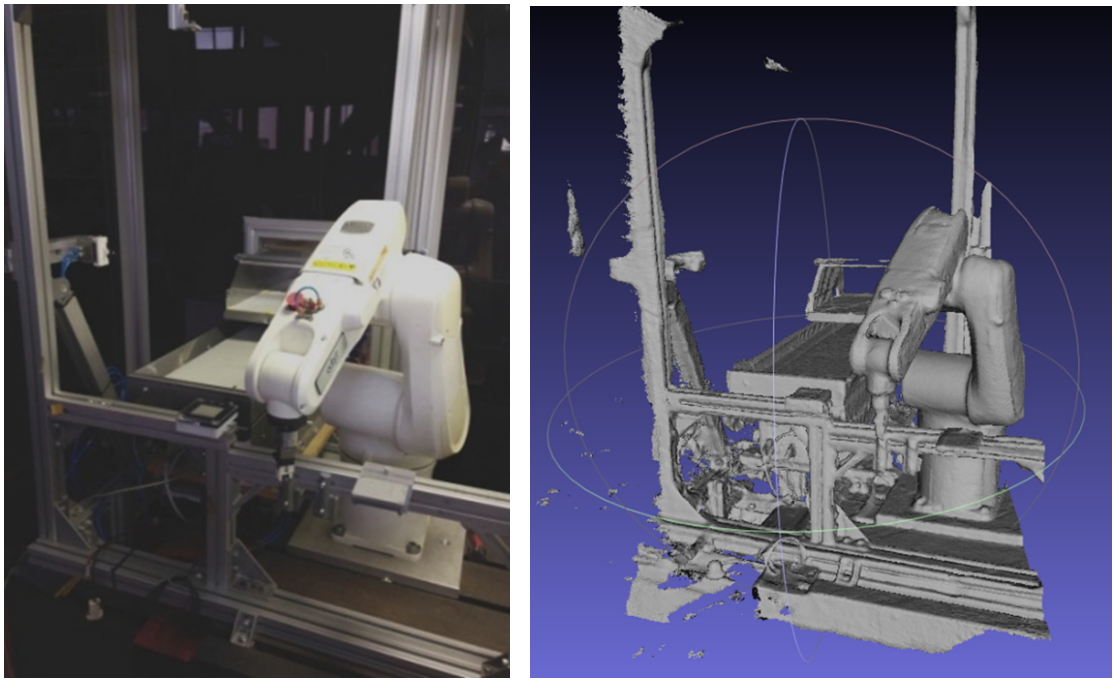


Figura 2.8: Processo di acquisizione della geometria della cella robotizzata



(a) (b)

Figura 2.9: Risultato della scansione del volume di una cella robotizzata

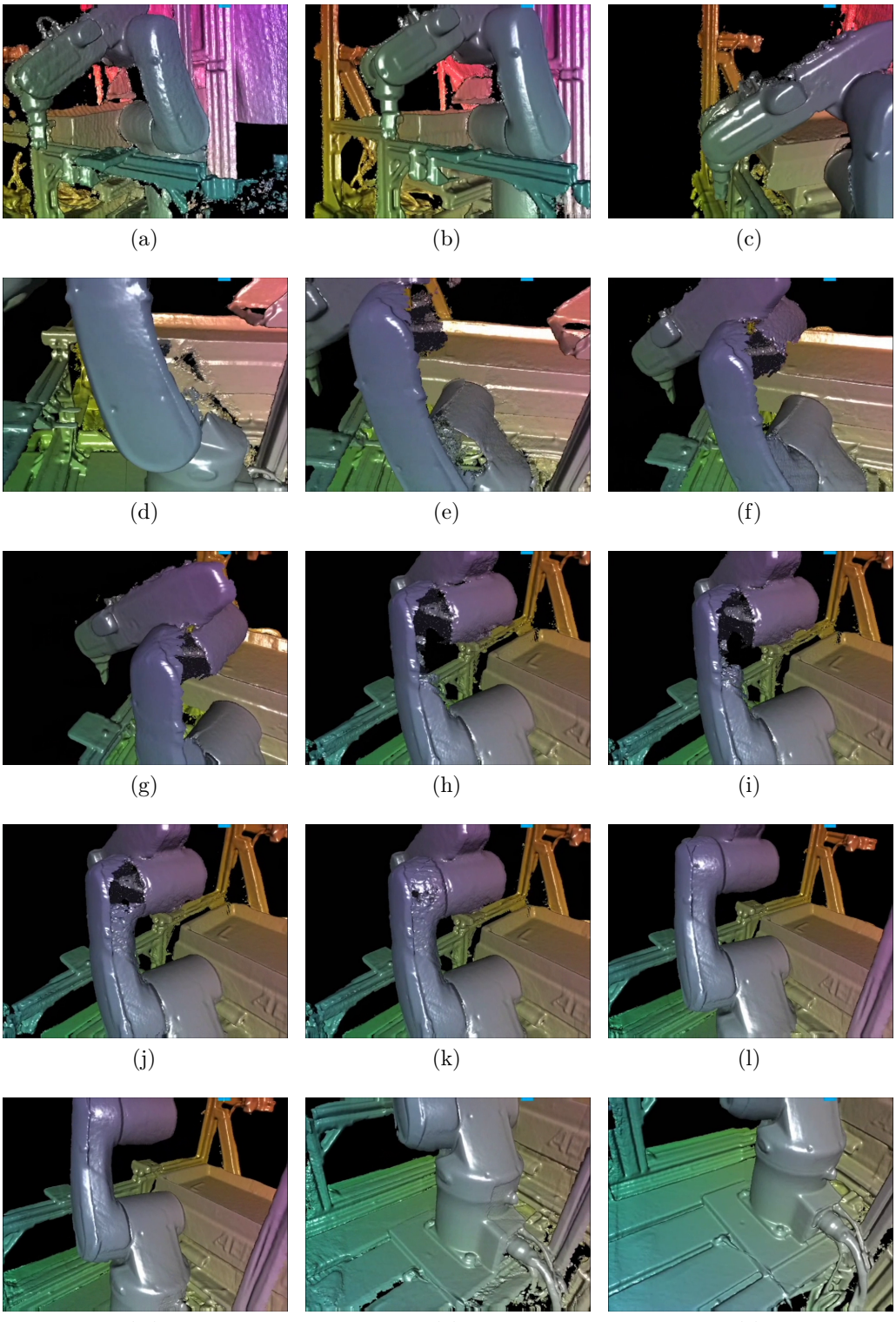


Figura 2.10: Frames in tempo reale durante il processo di acquisizione

Capitolo 3

Movimentazione assemblato SolidWorks tramite MatLab

In questo capitolo verrà descritto l'interfacciamento di un simulatore robotico con un software CAD commerciale per la simulazione di movimenti di un robot antropomorfo, la memorizzazione off-line delle traiettorie e dei punti di via.

3.1 Importazione della cella acquisita

Con l'acquisizione della geometria della cella robotica, descritta nel Cap. 2, si ottiene, come visto, un file di estensione STL che contiene l'intera scansione tridimensionale della cella acquisita. I due più diretti impieghi di tale file sono:

- la progettazione al CAD 3D nell'ambiente scansionato;
- la simulazione in tempo reale di cicli robot all'interno dell'ambiente scansionato.

Il primo impiego è possibile e naturale dal momento che il file è compatibile con tutti i software CAD 3D utilizzati nell'ambiente industriale. Ciò significa che è possibile sfruttare l'acquisizione per testare l'inserimento nello spazio di accessori della cella, di nuovi componenti, di alimentatori differenti, etc.

La movimentazione del robot secondo cicli robot reali in tempo reale all'interno dell'ambiente scansionato ha necessitato invece la progettazione e l'implementazione di un framework specifico, descritto nelle sezioni successive.

3.2 Framework per simulazioni robot in ambiente scansionato

Scopo del framework è muovere i link di un robot, secondo l'effettivo ciclo desiderato, all'interno dell'ambiente della cella scansionata, caricato in un software CAD 3D. Requisito necessario è quindi il file CAD dell'assemblato robot (estensioni interscambiabili STEP o IGES) costituito dai componenti link vincolati in corrispondenza dei giunti. Il file è tipicamente reperibile dal sito del produttore. Il software CAD che si è impiegato e preso in esame è SolidWorks, mentre il framework è stato sviluppato in MatLab. Si suppone quindi l'implementazione dei cicli robot e la gestione robot tramite tale ambiente software di sviluppo.

La procedura può essere suddivisa in alcune fasi principali:

1. importazione file CAD con estensione STL (cella acquisita) e STEP/IGES (robot) in SolidWorks;

2. confronto tra i file per eliminare il robot dal file STL della cella, sostituito dall'assemblato STEP/IGES;

3. inserimento delle terne di Denavit-Hartenberg [];

4. esecuzione delle funzioni del framework per la movimentazione.

Il framework sviluppato in MatLab si può a sua volta suddividere in 3 moduli:

- connessione alle API di SolidWorks;
- scansione componenti di SolidWorks;
- calcolo coordinate libere;
- movimentazione dei link in SolidWorks.

Il framework è stato sviluppato seguendo due possibili percorsi permessi dalle API. Un metodo consiste nell'utilizzare il comando *DRAG* e nello sfruttare così l'equivalente del trascinamento manuale. Il secondo tramite la modifica di *vincoli* negli accoppiamenti tra il link del robot. Nel seguito si farà riferimento solamente alla prima modalità che è l'unica che ha permesso il raggiungimento di performance in termini di frame video al secondo (fps) dell'ordine delle decine, requisito necessario per una buona visualizzazione.

3.3 Importazione file, confronto e sostituzione solido robot

La prima operazione necessaria per permettere poi la movimentazione del robot è importare il CAD dell'assemblato robot in SolidWorks per effettuare un confronto con la cella scansionata in modo da rimuovere il robot (statico) sostituendolo con l'assieme composto da parti accoppiate (dinamico), come mostrato in Fig. **??**. Per far questo si possono utilizzare le funzioni di SolidWorks che permettono di individuare dei punti desiderati, come il fissaggio della base robot al supporto, selezionare tutta la parte relativa al robot e sostituirla con l'assemblato dinamico.

Questa procedura potrebbe essere automatizzata importando i due file in MatLab ed eseguendo un algoritmo di pattern matching con calcolo baricentro per individuare e sostituire il robot. Sebbene siano stato costruite e implementate delle funzioni per l'importazione di file IGES e STL in MatLab, tuttavia la procedura non è stata resa automatica in questa versione del framework.

3.4 Inserimento delle terne di Denavit-Hartenberg

Per determinare e cambiare la configurazione di un robot nello spazio, vale a dire la posizione di tutti i suoi punti rispetto ad un sistema di riferimento assoluto, è stato necessario definire una procedura manuale semplice per l'inserimento di terne di riferimento in ogni corpo rigido di cui è costituito. Tali terne devono descrivere la posizione di ciascun link e permetterne il movimento da MatLab. Per questa ragione, si è optato per utilizzare la notazione di Denavit-Hartenberg, nata per semplificare l'analisi cinematica diretta di un meccanismo in catena aperta []. Le terne, indicate con $T\#$, sono state di conseguenza numerate seguendo tale notazione (una terna per ogni g.d.l.) e distinte tra traslazione $T\#Ti$ e rotazione $T\#Ri$ ($i = 1, ..., n$ con n numero di g.d.l.), coerentemente con i g.d.l. del robot. Il simbolo # serve per contraddistinguere l'elemento grafico di SolidWorks con una stringa speciale.

In Fig. 3.1 si riporta un esempio di assemblato CAD a tre gradi di libertà (due rotazione e una traslazione), composto da elementi semplici. La seconda barra presenta una sezione atta a realizzare un accoppiamento prismatico, questo per poter testare un caso completo di entrambe le due tipologie di g.d.l.

Per non dover ricostruire l'assemblato ad ogni spostamento di un componente, le terne sono state aggiunte in SolidWorks nel disegno CAD di ogni singola parte. In particolare $T\#R0$ è la terna fissa a telaio, $T\#R1$ la terna per la rotazione del primo link, $T\#R2$ la terna per la rotazione del secondo link e $T\#T3$ la terna del secondo link.

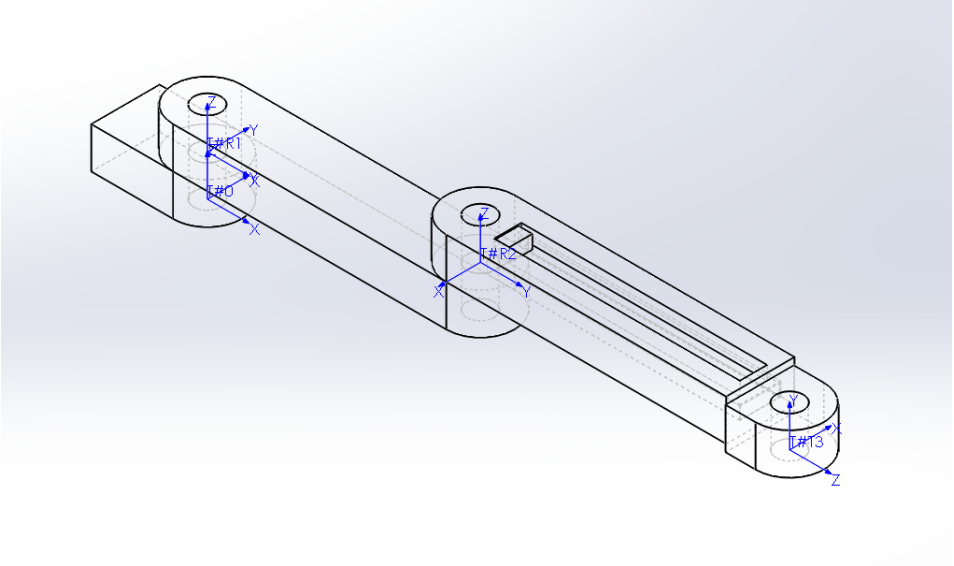


Figura 3.1: Assemblato CAD con avambraccio telescopico.

3.5 Connessione alle API di SolidWorks

Il metodo scelto per muovere un componente meccanico sfrutta il comando *Drag* (o *trascinamento*) presente nella UI (interfaccia utente) di SolidWorks. Per poter invocare questo comando da MatLab è necessario collegarsi alle API di SolidWorks ed individuare il componente da muovere. Per eseguire la connessione è stata implementata la funzione `connection`:

```
function [swApp, swModel, swRootComp, swDragOp, swSelMgr, swMathUtil,
    swModelView] = connection()

swApp = actxserver('SldWorks.Application');
% openDoc(swApp);
swModel = invoke(swApp, 'ActiveDoc');%invoke(swApp, 'ActiveDoc3',);
swDragOp = invoke(swModel, 'GetDragOperator');
swSelMgr = invoke(swModel, 'SelectionManager');
swMathUtil = invoke(swApp, 'GetMathUtility');

swConfMgr   = invoke(swModel,'ConfigurationManager');
```

```
swActiveConf = invoke(swConfMgr,'ActiveConfiguration');
swRootComp  = invoke(swActiveConf,'GetRootComponent');
swModelView = invoke(swModel,'ActiveView');
```

In questa funzione di MatLab si richiamano tutte le classi e le proprietà principali che verranno utilizzate all'interno dello script principale per richiamare i metodi essenziali al lavoro.

ActiveDoc è una delle diverse proprietà che ha la classe principale *ISldWorks* e permette l'attivazione del documento corrente; se nessun documento è attivato viene restituito *NULL*. Si tratta di una proprietà di sola lettura.

Il metodo *GetDragOperator*, appartenente alla classe *IAssemblyDoc*, restituisce l'operatore *drag* per le operazioni di trascinamento dell'assemblato.

SelectionManager restituisce l'oggetto ISelectionMgr per il documento corrente, rendendo l'oggetto attualmente selezionato disponibile. Gli oggetti ISelectionMgr sono transitori perché non sono più validi appena è fatta un'altra selezione, quindi è consigliato non trattenere questi puntatori per un lungo periodo di tempo.

GetMathUtility è un metodo di ISldWorks e restituisce l'interfaccia *IMathUtility* o *NULL* se l'operazione fallisce. *IMathUtility* fornisce l'accesso agli oggetti matematici di SolidWorks, i quali possono semplificare i calcoli matematici di uso comune utilizzati con molte funzioni API.

ConfigurationManager è una proprietà di IModelDoc e restituisce l'oggetto IConfigurationManager, il quale permette l'accesso alle configurazioni in un modello. *ActiveConfiguration* è la proprietà di IConfigurationManager che attiva le configurazioni del documento e, se questo documento è un assemblato, si potrà utilizzare questa proprietà per iniziare la scansione delle componenti di un assemblato: è ciò che sarà illustrato nella funzione *scanner()*.

GetRootComponent, metodo di IConfiguration, restituisce il componente radice della configurazione dell'assemblato in questione.

La chiamata ad *ActiveView* è invece necessaria per permettere l'attivazione e la disattivazione di barre, comandi, oggetti, etc. di SolidWorks durante il movimento. Ciò permette di incrementare le prestazioni in termini di fps.

3.6 Scansione componenti di SolidWorks

La prima operazione, a seguito della connessione, è eseguire la scansione del *FeatureManager* (albero features di SolidWorks) alla ricerca di tutte le terne solidali ai membri costruite precedentemente (funzione `scanner`). Tuttavia, per realizzare lo spostamento dei membri di un meccanismo in catena aperta ci si trova a dover risolvere il *problema cinematico diretto di posizione*, ovvero si vuole calcolare la posizione di ogni suo punto a partire dai valori delle coordinate libere. La risoluzione di tale problema sta nell'utilizzare le terne di D.-H. disposte in precedenza per calcolare le matrici di trasformazione tra tali sistemi e il sistema alla base del robot. In questa fase è necessario quindi che la funzione `scanner` individui la posizione e l'orientazione delle terne di D.H. e calcoli le matrici in modo da salvare la configurazione iniziale del robot. Ciò è necessario per permettere poi la movimentazione a partire da questa configurazione iniziale.

La funzione è riportata per sommi capi:

```
[nameLink, nameLink0, swComp, swComp0, swLink, swLink0, Tname, T0a,
    Tia_in] = scanner(swRootComp)

% Ricerca dei componenti dell'assieme (link del robot)
swComp = invoke(swRootComp,'GetChildren');

...

% Ricerca delle terne all'interno dei componenti dell'assieme
for j=1:length(swComp)

        swFeature = invoke(swComp{j}, 'FirstFeature');
    swLink{j} = invoke(swComp{j},'GetModelDoc2'); %ModelDoc2
    swDocExt = invoke(swLink{j}, 'Extension');
    swZform = invoke(swComp{j},'Transform2');
    swTd = invoke(swZform,'ArrayData');
    Tia_in(:,:,j) = sldvect2mat(swTd);
```

```
    while swFeature ~= isnan(nan)
        swFeatName = invoke(swFeature, 'Name');
        if strncmp(swFeatName, 'T#', 2) == 1 % Confronta i primi due
            caratteri
            Tname{j} = swFeatName;
            swZform = invoke(swDocExt, '
                GetCoordinateSystemTransformByName', swFeatName);%
                swFeatName;
            swTd = invoke(swZform,'ArrayData');
            Tia_in(:,:,j) = Tia_in(:,:,j) * sldvect2mat(swTd);
        end
        swFeature = invoke(swFeature, 'GetNextFeature');
    end

end

% Ri-denominazione e salvataggio
...
```

Questa funzione inizia con il metodo *GetChildren*, il quale restituisce un array di componenti, detti *Children* del *FeatureManager and Component Design Tree*. È necessario poi procedere alla ricerca delle terne all'interno di quest'albero e per farlo si dovranno analizzare anche le sottocartelle dei singoli componenti. Quest'ultima operazione si esegue perché le terne sono introdotte all'interno delle singole parti dell'assemblato. Per raccogliere gli elementi utili per lavorare con le terne solidali al membro sono create delle celle e una matrice. In questo modo si possono immagazzinare tutte le informazioni che pervengono nella scansione dell'albero FeatureManager.

FirstFeature è il metodo che fornisce il primo elemento del array in cui sono contenuti tutti i componenti del FeatureManager. Per procedere all'interno delle sottocartelle si invoca il metodo *GetModelDoc2*, il quale restituisce il nome della prima parte che costituisce l'assemblato e viene salvato nella cella swLink. Il metodo *Extension* espande le sottocartelle, restituendo l'oggetto *IModelDocExten-*

sion che permette l'accesso al documento del modello. *Transform2*, proprietà di *IComponent2*, restituisce un componente *IMathTransform*. *ArrayData*, proprietà di *IMathTransform*, permette di settare o di leggere un array di 16 elementi double che costituiscono la matrice di trasformazione, disposti in un array nell'ordine seguente:

$$T_{ia} = [a, b, c, d, e, f, g, h, i, l, m, n, o, p]$$

Questo vettore contiene tutti gli elementi della matrice di trasformazione T_{ia} che consente di trasformare punti e vettori del sistema i-esimo in punti e vettori del sistema ambiente. La matrice è scritta in forma di vettore, è quindi necessario trasformarla in forma matriciale, per questo è stata scritta la funzione *sldvect2mat*. La matrice di trasformazione in forma matriciale contiene nelle colonne i versori della terna i-esima e la posizione dell'origine della terna i-esima, espressi nella terna ambiente in coordinate omogenee.

$$\mathcal{T}_{ia} = \begin{bmatrix} a & d & g & j \\ b & e & h & k \\ c & f & i & l \\ n & o & p & m \end{bmatrix}$$

La funzione *sldvect2mat* ordina gli elementi del array in una matrice di trasformazione.

Il ciclo che scompone tutti i componenti del sotto-albero di ogni parte alla ricerca delle terne contiene il metodo *Name*, proprietà di *IFeature*, il quale restituisce o imposta il nome della corrente caratteristica.

Infine *GetNextFeature*, metodo di *IFeature*, fornisce l'elemento successivo nel FeatureManager.

Questa funzione viene eseguita iterativamente e alla fine restituisce un insieme di celle e matrici ridimensionate:

- *nameLink*, cella con i nomi di tutti i membri del meccanismo tranne il membro che funge da telaio;
- *nameLink0*, nome del componente che funge da telaio per il meccanismo meccanico;

- *swComp*, nomi delle parti che raccolgono tutti i dettagli CAD per la costruzione di ogni singolo membro che compone l'assemblato;

- *swComp0*, nome della parte che costituisce il telaio su cui è vincolato il primo membro del componente meccanico;

- *swLink*, i riferimenti oggetto dei membri tranne il telaio;

- *swLink0*, riferimento all'oggetto telaio;

- *Tname*, la cella che contiene tutti i nomi delle matrici di trasformazione ($T\#R0$, $T\#R1$, ecc.);

- T_{0a}, la matrice di trasformazione T_{0a};

- T_{ia_in}, contiene tutte le matrici delle trasformazioni relative a ogni singolo membro rispetto alla terna ambiente T_{ia}.

3.7 Calcolo valori coordinate libere

Una volta note tutte le matrici di trasformazione delle terne i-esime rispetto alla terna ambiente, si è sviluppata la funzione `calcDHmatrixAll` che consente di ricavare tutte le terne $T_{i,i-1}$ e le coordinate libere $q_1, q_2, ..., q_n$. Trovare le coordinate libere secondo la notazione di D.-H. risulta essenziale dal momento che la movimentazione sarà ottenuta variando proprio tali valori q_i di Δq_i per ottenere i valori finali desiderati.

Le matrici di trasformazione relative, che descrivono la trasformazione dal sistema (i-1)-esimo al sistema i-esimo, $T_{i,i-1}$ si calcolano da:

$$T_{ia} = T_{i-1,a} \cdot T_{i,i-1} \quad \Longrightarrow \quad T_{i,i-1} = [T_{i-1,a}]^{-1} \cdot T_{ia} \tag{3.1}$$

La coordinata libera i-esima può essere una rotazione di angolo θ_i intorno all'asse z_i, che porta l'asse x_{i-1} lungo la direzione dell'asse x_i, oppure una traslazione

di distanza d_i lungo l'asse z_i. Data la matrice di D.-H.:

$$T_{i,i-1} = T_{Rx}(\alpha_{i-1}) \cdot T_{Tz}(a_{i-1}) \cdot T_{Rz}(\theta_i) \cdot T_{Tz}(d_i) \tag{3.2}$$

$$= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

nel caso di rotazione la coordinata libera q_i si ricava tramite:

$$q_i = \theta_i = \arctan(-T_{i,i-1}(1,2)/T_{i,i-1}(1,1)) \tag{3.4}$$

mentre nel caso di traslazione:

$$q_i = d_i = T_{i,i-1}(3,4) \tag{3.5}$$

3.8 Movimentazione dei link in SolidWorks da MatLab

Ultimo passo è definire il valore desiderato delle coordinate libere e muovere i link solidali alle terne. La funzione implementata e richiamata è `moveAll`.

La seguente funzione di movimento prende in ingresso parametri noti e si presta ad invocare il comando di movimento distinguendo i casi di rotazione e traslazione. Segue un estratto della funzione:

```
function [Tia,Ti,q] = moveAll(swModel,q_fin,q_in,swComp,swLink,
    swMathUtil,swDragOp,T0a,Tia,Ti,Tname,rbool)

...

% Ciclo di aggiornamento di ogni link
for i=1:n
```

```
    % Aggiornamento Tia(:,:,i) = trasformazione assoluta
    if i==1
        Tia(:,:,i) = T0a * Ti(:,:,i);
    else
        Tia(:,:,i) = Tia(:,:,i-1) * Ti(:,:,i);
    end

    % Variazione coordinata relativa (DH)
    deltaq = q_fin(i)-q_in(i);

    % Movimento
    Tia(:,:,i) = move(swComp{i},rbool(i),deltaq,swMathUtil,swDragOp,Tia
        (:,:,i),Tia_in(:,:,i));

end

...
```

Come si vede dal codice, la funzione ha lo scopo di richiamare, per ogni coordinata libera, la funzione `move`, la quale passa a SolidWorks la matrice di trasformazione per l'aggiornamento richiesto e fornisce in uscita la matrice $T_{i,a}$ i-esima.

La funzione `move` calcola la matrice di rotazione/traslazione rispetto all'asse z per ottenere la nuova matrice di trasformazione T_{ia} aggiornata. Inoltre si ricalcola (matrice $TTRa$) e si riscrive (matrice $vZform$) la trasformazione nella forma necessaria a SolidWorks. Si passa il parametro al metodo *CreateTransform* di SolidWorks.

Infine si richiamano i metodi *Drag* per il trascinamento che permette la movimentazione richiesta.

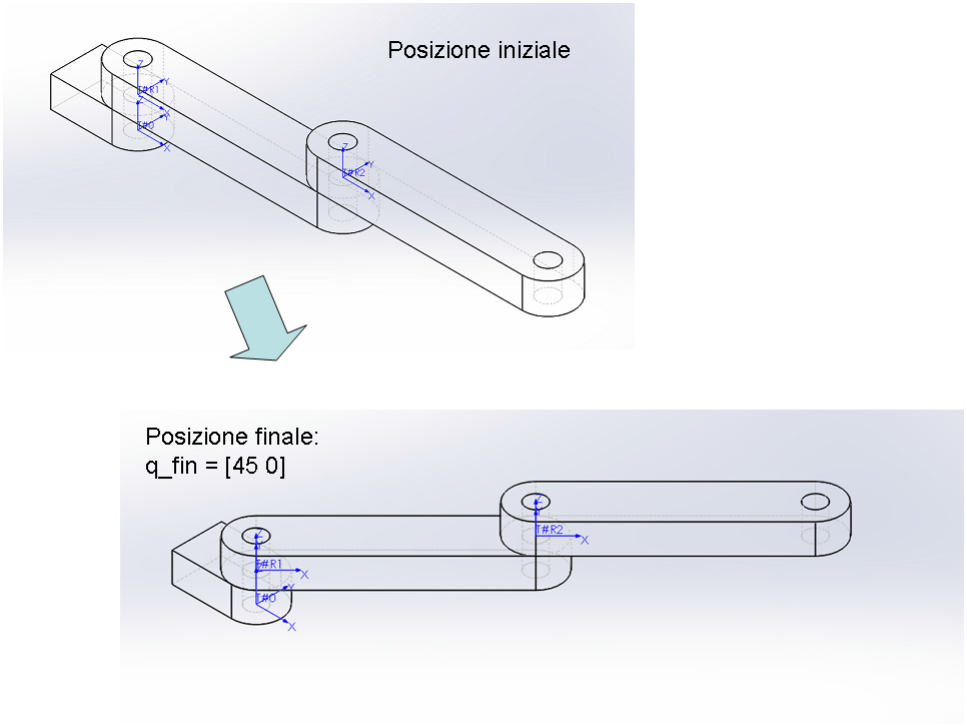
```
function Tia = move(swComp,rbool,deltaq,swMathUtil,swDragOp,Tia,Tia_in)

% Rotazione DH relativa rispetto al valore precedente
% rbool = 1 ROTAZIONE; 0 TRASLAZIONE
```

```
Tr_deltaq = [cosd(deltaq*rbool) -sind(deltaq*rbool) 0 0; ...
            sind(deltaq*rbool) cosd(deltaq*rbool) 0 0; ...
            0                  0                  1 deltaq*(rbool==0)/1e3;
                 ... %trasformazione in [m]
            0                  0                  0 1];

TTRa = Tia * Tr_deltaq * Tia_in^(-1);
Tia = Tia * Tr_deltaq;

% Creazione matrice
vZform = [TTRa(1:3,1)', TTRa(1:3,2)', TTRa(1:3,3)', TTRa(1:3,4)', 1, 0
    0 0];
swXform = invoke(swMathUtil, 'CreateTransform', vZform);

bRet = invoke(swDragOp, 'AddComponent', swComp, 0);
bRet = invoke(swDragOp, 'BeginDrag');
bRet = invoke(swDragOp, 'DragAsUI', swXform);
bRet = invoke(swDragOp, 'EndDrag');
```

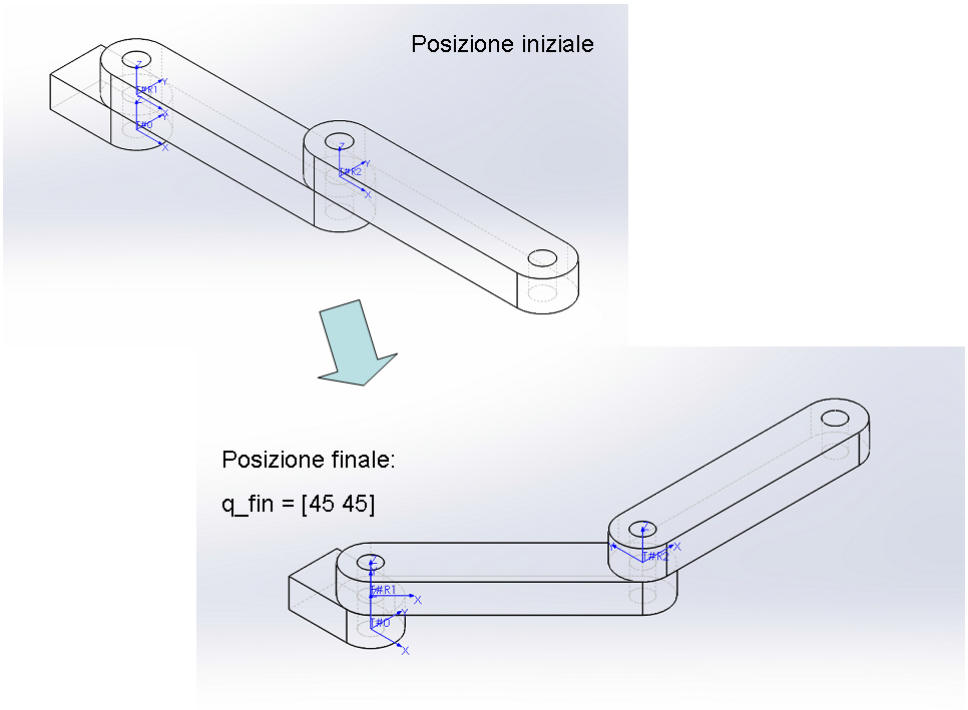
3.9 Test del framework

Il primo assemblato testato è un robot a due gradi di libertà disegnato in SolidWorks in maniera semplificata con vincoli di tipo rotoidale. L'applicazione di tale metodo a questo componente soddisfa tutte le specifiche e non presenta complicazioni. Nella Figura 3.2(a) e nella Figura 3.2(b) sono visualizzate rispettivamente due immagini che riportano la rotazione di 45° del primo membro prima, poi vengono mossi entrambi della stessa quantità.

Si è testato inoltre un robot commerciale presente nei laboratori del dipartimento il cui CAD è stato scaricato dal sito del produttore. Il robot *Adept Viper s650* a sei assi di rotazione è un robot articolato ad alte prestazioni progettato appositamente per le applicazioni di assemblaggio. La velocità e la precisione ne consentono l'utilizzo per la movimentazione dei materiali, l'imballaggio, l'asservi-



(a) `q_fin = [45; 0]`



(b) `q_fin = [45; 45]`

Figura 3.2: Movimentazione manipolatore a due gradi di libertà, coordinate espresse in [gradi].

mento macchine e molte altre operazioni che richiedono un'automazione rapida e precisa [29]. Il disegno CAD del robot Viper s650 è riportato in Fig. 3.4.

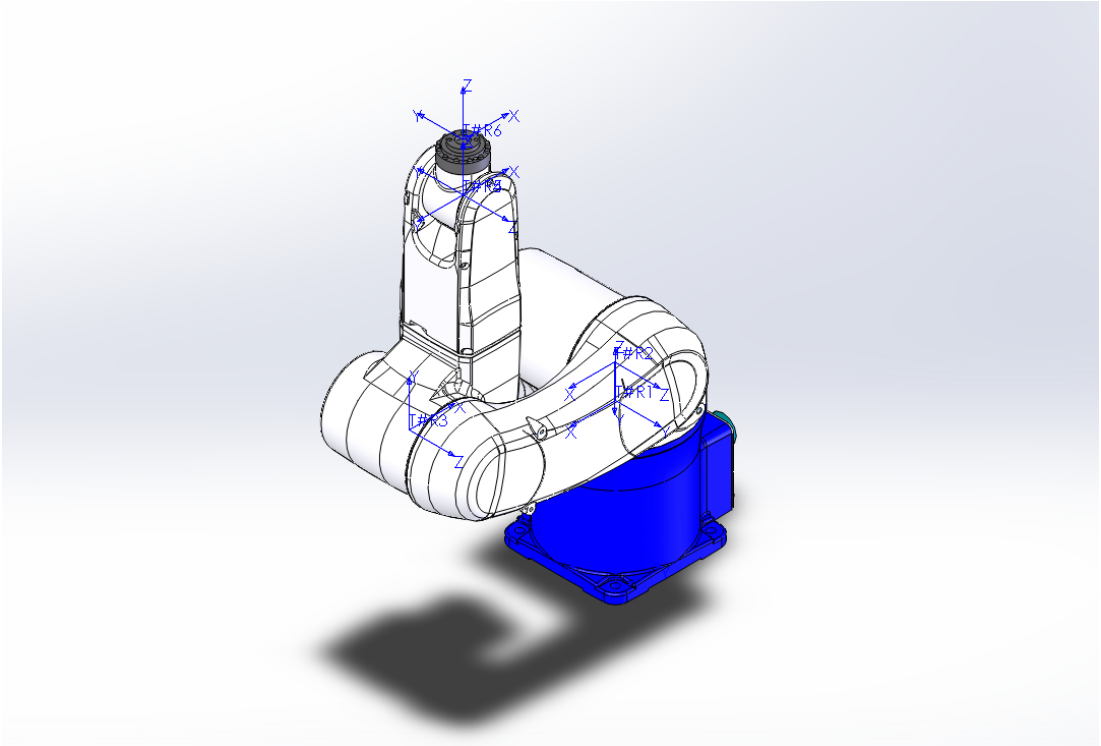


Figura 3.3: Adept Viper s650.

Il manipolatore, composto da membri vincolati con giunti di tipo solo rotoidale, risponde correttamente al metodo *DRAG*. Un'analisi semplice mostra come il refresh, ottenuto temporizzando il movimento dell'immagine per mezzo di una funzione di attesa sviluppata e compilata `mexwait`, sia di circa $23 fps$ per portare il robot dalla configurazione `q_in=[0 0 0 0 0 0];` alla configurazione `q_fin=[60 -45 45 -45 -45 -45]`.

3.10 Utilizzo del framework

Attraverso l'interfacciamento con un simulatore robotico in Matlab, in precedenza sviluppato presso i laboratori Mechatronics dell'Università degli Studi di Padova, e del framework software in grado di interagire con l'ambiente di disegno tecnico SolidWorks, è possibile compiere operazioni come:

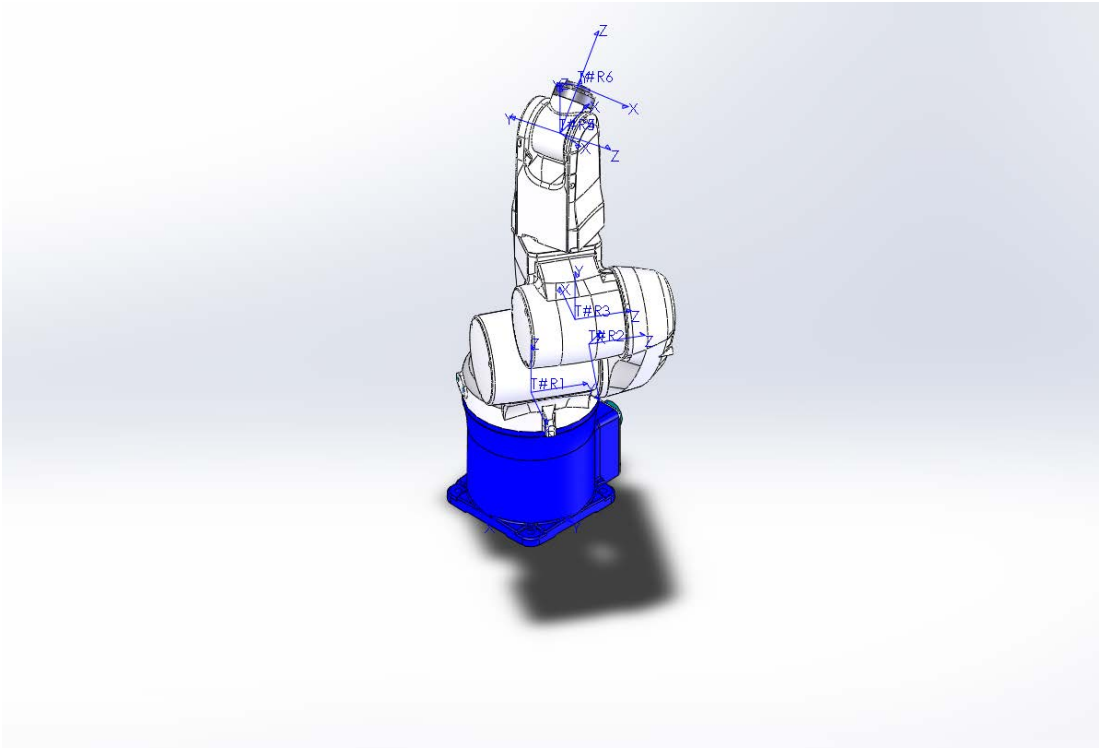


Figura 3.4: Movimentazione tramite Drag con `q_fin=[60 -45 45 -45 -45 -45]`.

- la navigazione off-line nella geometria acquisita simulando i movimenti del robot attraverso la GUI;
- la memorizzazione off-line delle traiettorie e dei via-point;
- la possibilità di progettazione di parti o specifici componenti che andranno ad integrare la cella di lavoro, potendo simulare le traiettorie direttamente nell'ambiente CAD commerciale e memorizzare successivamente i punti nella GUI;
- la possibilità, sfruttando la geometria acquisita della cella di lavoro, di verificare il rischio di collisione del robot in tempo reale; problema importante in quelle fasi del ciclo di lavoro in cui il movimento del robot non sia pre-registrato ma venga comandato da coordinate calcolate, ad esempio, da sensori, attraverso il motore di collision detection del software CAD commerciale.

Capitolo 4

SoleSound: calzatura instrumentata

In questo capitolo verrà descritta l'attività di ricerca svolta con il gruppo Rehabilitation And Robotics (ROAR) presso la Columbia University in the city of New York, descrivendo lo sviluppo di una scarpa strumentata, completamente portabile, in grado anche di misurare alcuni parametri spazio-temporali della camminata. Usando un sistema camera-based le performance del dispositivo sono state valutate con due strategie di calibrazione: soggetto-specifico e non soggetto specifico.

4.1 Introduzione

L'Analisi del Cammino (Gait Analysis) - un metodo con il quale la locomozione umana è quantitativamente analizzata utilizzando strumenti che catturano parametri cinematici e cinetici - è uno strumento molto importante per il trattamento clinico di pazienti con disturbi alla camminata. Il Gait Analysis permette una diagnosi più accurata rispetto a quelle effettuate attraverso osservazioni o attraverso questionari e permette di stabilire la terapia maggiormente adeguata [30, 31]. Alcuni studi hanno anche suggerito che i risultati funzionali sono superiori quando i trattamenti seguono le raccomandazioni dell'analisi del cammino [31]. Ancora, carenze che impediscono l'uso diffuso dell'analisi del cammino come strumento clinico comprendono costose attrezzature da laboratorio, elevata richiesta di tempo per l'analisi dei dati, con costi elevati che molti assicurazioni mediche non coprono.

Per superare queste limitazioni, negli ultimi anni diversi ricercatori hanno sviluppato dispositivi di analisi della camminata indossabili, che consistono in leggeri sensori alimentati a batteria indossati sulla parte bassa del corpo comprendendo anche una unità logica che permette di elaborare e memorizzare i dati acquisiti. Questi dispositivi possono essere realizzati per una frazione del costo dei sistemi di analisi della camminata attuali e possono essere utilizzati in qualsiasi posizione. Inoltre, essi possono essere utilizzati per valutare la deambulazione del paziente in remoto, su qualsiasi distanza o percorso (ad esempio, salire le scale), per periodi di tempo prolungati, consentendo a medici di valutare la variabilità di deambulazione o quantitativamente monitorare episodici fenomeni che sono difficili da suscitare in contesti clinici, come il congelamento di deambulazione [32]. Infine, questi dispositivi indossabili possono agire come strumenti per la riqualificazione della camminata, fornendo un feedback uditivo o vibratorio per aiutare coloro che indossano questi dispositivi a correggere i loro movimenti in caso di bisogno. D'altro canto, questi sistemi sono limitati nel numero di parametri della camminata che riescono a misurare e nella loro precisione [33]. Solitamente, richiedono una connessione cablata [34, 35], o senza fili [36, 37, 38, 39] ad un computer host per l'elaborazione dei dati on-line o per la registrazione dei dati, limitando la loro mobilità effettiva.

I dispositivi di analisi della camminata possono essere classificati in base alla tipologia di variabili che permettono di monitorare: *cinematici* (spaziali o temporali) o *cinetici*. Parametri spaziali come *stride lenght*, *base of walking*, *foot clearance*, *foot trajectory* ed *ankle range of motion* sono indicatori importanti per i disturbi dell'andatura. Per esempio, allargare la *base of walking* e compiere piccoli passi potrebbero essere associati a disturbi dell'equilibrio [40], un ridotto movimento angolare della caviglia è associato ad un rischio più elevato di caduta [41, 42, 43]; un controllo alterato della traiettoria del piede durante la fase di *swing* potrebbe risultare in una insufficiente *foot clearance* aumentando le possibilità di scivolamento od inciampo [44]. In seguito alla miniaturizzazione e alla discesa di costo dei sensori inerziali, l'ultimo decennio ha visto una rapida crescita di prototipi di ricerca e nella disponibilità di prodotti commerciali per il *motion tracking*, la localizzazione pedonale [45, 46] e per l'identificazione della postura e delle attività motorie [47, 48]. I dispositivi di motion tracking sono utilizzati nell' analisi del cammino od in applicazioni di realtà virtuale. Possono essere ulteriormente classificati in *inertial orientation trackers* - per esempio reti di sensori inerziali che sono indossate negli arti dei soggetti, comprese le estremità inferiori [49, 50, 51, 52, 53, 39], opzionalmente nel tronco [54] e nelle braccia [55] - ed in *shoe-based system* che sono indossati nel piede [34, 38, 56] ed in maniera opzionale sulla tibia [35, 37].

I parametri spaziali sono derivati dai dati di posizione e di orientazione misurati dalle unità inerziali. In linea di principio l'orientazione relativa ad un sistema inerziale può essere calcolata integrando le letture di un giroscopio tri-assiale o attraverso i dati di inclinazione provenienti da un accelerometro a tre-assi e da un magnetometro per ottenere l'azimut. I primi sistemi indossabili stimavano gli angoli sul piano sagittale integrando le letture di giroscopi monoassiali situati su ciascun segmento dell'arto [49]. Questo approccio, tuttavia, soffriva di derive in bassa frequenza a causa del bias del sensore. D'altronde, le misurazioni fornite dagli accelerometri sono affette da rumore in alta frequenza mentre i dati forniti dal magnetometro soffrono da distorsioni causate da campi magnetici locali. Recenti ricerche si sono concentrate su possibili metodi per migliorare l'accuratezza della stima dell'orientazione e gli approcci suggeriti si possono dividere in tre principali

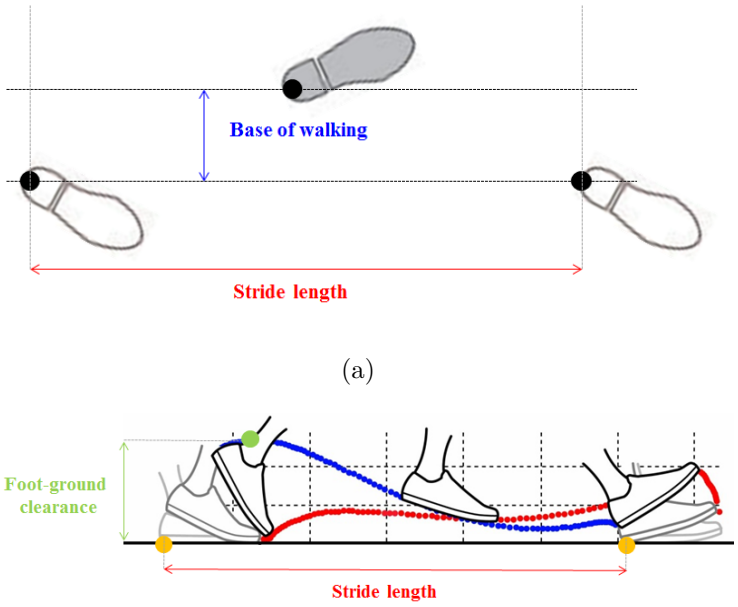


Figura 4.1: Rappresentazione grafica di alcuni parametri della camminata

categorie. I sistemi *shoe-based* possono sfruttare i vantaggi della dinamica della camminata, cioè la posa relativamente stazionaria del piede e della tibia durante il *foot flat period of stance* (FF), al fine di compensare il drift dei sensori. Sotto ipotesi pseudo-stazionarie i contributi dinamici alle letture degli accelerometri possono essere trascurate e la direzione del vettore gravità può essere stimato con sufficiente accuratezza da questi segnali. Le fasi FF possono essere rilevate dai giroscopi a bordo [49, 34, 37, 50], da sensori di forza resistivi (FRS) [37] o da sensori di forza multi-assiali [35, 57, 56]. Nei modelli planari, l'integrazione dei giroscopi è condotta tra successive fasi FF, l'angolo di beccheggio del piede viene azzerato all'inizio di ogni fase ed una compensazione del drift lineare viene applicata imponendo a zero l'angolo di beccheggio alla fine dell'intervallo di integrazione [34, 37]. Nei modelli spaziali, l'orientazione 3D è derivata dall'integrazione dei giroscopi e successivamente aggiustata con la stima dell'inclinazione proveniente dalle letture di un accelerometro ad ogni fase FF [56] o, in alternativa, ad ogni volta che la norma dell'accelerazione è sufficientemente vicina all'accelerazione di gravità [58, 38, 59, 60, 61].

Il secondo approccio per migliorare l'accuratezza di stima delle orientazioni

si basano su tecniche di sensor fusion. Il filtro di Kalman Esteso (EKF) è stato ampiamente utilizzato in applicazioni di navigazione. Nel gait analysis, diversi autori hanno proposto tecniche per prevenire spurie correzioni dall'EKF quando le condizioni di pseudo-statica non sono soddisfatte. Queste includono filtraggi con filtri passa-basso per le accelerazioni [52], correzioni condizionali in base al modulo dell'accelerazione misurata [62] e attraverso una stima adattativa della matrice di covarianza . Nonostante l'EKF sia uno strumento molto potente, richiede anche una notevole capacità di calcolo e delle conoscenze a priori delle dinamiche di processo. Per ovviare a tali inconvenienti, i ricercatori hanno proposto filtri model-free, tra cui il filtro di Wiener ed i filtri complementari [63].

Il terzo approccio mira a migliorare ulteriormente la precisione di stima dell'orientazione sfruttando sia tecniche di sensor-fusion che vincoli biomeccanici inerenti. Cooper [51] ha stimato l'angolo di flessione/estensione del ginocchio con una coppia di 6-DOF IMU situati sulla coscia e sulla tibia, modellando il ginocchio come una semplice cerniera e supponendo noto l'orientamento relativo iniziale tra i due sensori inerziali. Seel [53] ha sviluppato un metodo similare che non richiede una calibrazione esterna. Precisione ed accuratezza riportate dai due precedenti lavori risultato essere superiori a quelle degli approcci precedenti. Slajpah [39] ha modellato le basse estremità come una catena cinematica seriale assumendo il piede come un punto di contatto stazionario a terra ed applicando successivamente un filtro EKF in maniera iterativa. L'inconveniente principale del loro approccio risiede nella necessità di calibrare il modello cinematica con un sistema basato su telecamera.

Molti sistemi sono anche in grado di misurare lo spostamento del piede all'interno di un ciclo del passo [35, 57, 38, 59, 60, 61], o la sua proiezione sul piano sagittale [34, 37, 59]. Queste stime sono utilizzate per derivare parametri come *stride length* e *foot clearance*. Sebbene approcci basati su vincoli biomeccanici siano stati proposti [49], la tecnica maggiormente utilizzata prevede l'eliminazione della gravità dalle letture degli accelerometri nel piede (attraverso la stima dell'orientazione) seguita da una doppia integrazione [37]. L'integrazione viene comunemente eseguita tra successive fasi FF, con tecniche ZUPT e di compensazione del drift in velocità [34, 35, 38, 57, 59, 60, 61].

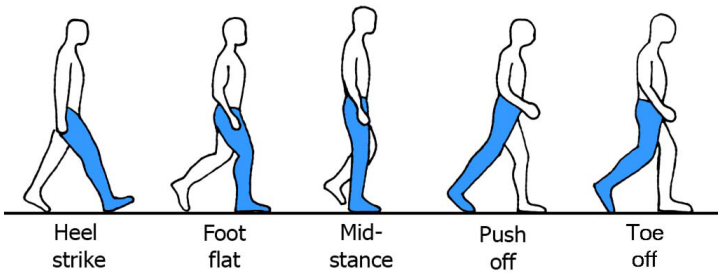


Figura 4.2: Fasi della camminata [66]

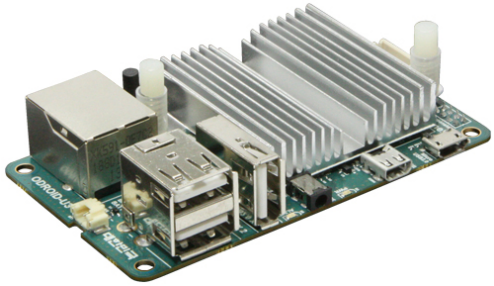
Parametri temporali come il tempo di un passo, la durata del doppio supporto e parametri temporali di simmetria sono usualmente stimati tramite sensori inerziali inseriti all'interno della scarpa [49, 34, 50]. Un approccio maggiormente diretto, invece, si basa sul rilevamento di forze sotto al piede con l'ausilio di sensori di forza resistivi (FSR) in grado di rilevare temporalmente le fasi di *heel-strike* e *toe-off*. Più complessi sistemi basati su array di sensori posizionati sotto al piede sono stati proposti per una accurata segmentazione del cammino e per stimare parametri cinetici come le forze di reazione normali (GRF) e la traiettoria del centro di pressione (CoP) [35, 64, 62, 65].

4.2 Descrizione del prototipo

SoleSound è un sistema indossabile modulare che consiste di un'unità calzatura e di un marsupio. In Fig. 4.8(a) è raffigurato il sistema completo indossato da un soggetto. Ogni calzatura permette di misurare dati di pressione del piede oltre che i dati cinematici del piede e della tibia. Un motore di sintetizzazione real-time, in esecuzione su un computer portatile, processa questi dati generando segnali analogici, i quali sono amplificati. Essi permettono di alimentare dei motori vibrotattili posizionati nella suola della calzatura, degli speaker in miniatura portatili o delle cuffie.

Il modulo del piede è costituito da un paio di sandali (Teva Jetter, Deckers Outdoor Corp., Goleta, CA) la cui schiuma è stata intagliata per ospitare cinque motori vibro-tattili Haptuators Mark II, Tactile Labs, Montreal, Canada) ed un sensore inerziale a 9 gradi di libertà (3-Space Embedded, Yost Engineering Inc.,

Tabella 4.1: Caratteristiche tecniche del computer portatile



CPU	1.7GHz Exynos4412 Prime Cortex-A9 Quad-core processor
RAM	2Gbyte LPDDR2
PMIC	MAX77686 Power Management IC
HSIC USB 2.0 Hub	USB3503A Integrated USB 2.0-compatible
HSIC Ethernet controller	LAN9730HSIC USB 2.0 to 10/100 Ethernet controller
Audio CODEC	MAX98090
HDMI connector	Standard Micro-HDMI, supports up to 1920 x 1080 resolution
Connectivity	USB Host x 3, Device x 1, Ethernet RJ-45, Headphone Jack
IO Ports	GPIO, UART, I2C, SPI(Board Revision 0.5 or higher)?
Storage Slot	Micro-SD slot, eMMC module connector
DC Input	5V / 2A input

Portsmouth, OH). Due di questi trasduttori sono stati posizionati anteriormente (sotto l'alluce ed il primo metatarso), due posteriormente (sotto il calcagno) ed uno sotto l'arco laterale del piede, seguendo approssimativamente la distribuzione cutanea dei meccanocettori nella pianta del piede [67]. Il sensore inerziale è stato posto lungo la linea mediana del piede, al di sotto dell'articolazione tarso-metatarsale. Quattro sensori di forza piezoelettrici resistivi (A-401, FlexiForce, Tekscan, Inc., South Boston, MA) sono stati incorporati in una suola di gomma di silicone per misurare la pressione sotto il calcagno, del quarto e del primo metatarso e dell'alluce. In Fig. 4.3(b) è raffigurata la disposizione spaziale dei sensori e degli attuatori utilizzati.

Una seconda unità inerziale è racchiusa in una piccola scatola di plastica per essere fissata alla tibia del soggetto per mezzo di uno strap in Velcro. Un sensore ad ultrasuoni (PING, Parallax Inc., Rocklin, CA), visibile in Fig. 4.5(a), è stato montato sul lato mediale-posteriore della calzatura per la stima della base del

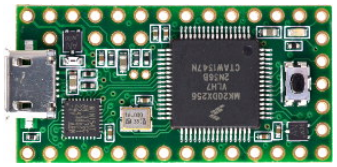
Tabella 4.2: Caratteristiche tecniche IMU

Accelerometer scale	±2g, ±4g, ±8g
Accelerometer resolution	14 bit
Accelerometer noise density	99 $\mu g\sqrt{Hz}$
Accelerometer sensitivity	0.00024 g/digit - 0.00096 g/digit
Gyro scale	±250, ±500, ±1000, ±2000 deg/sec
Gyro resolution	16 bit
Gyro noise density	0.009 deg/sec/$\sqrt{Hz}$
Gyro bias stability (25 °C)	2.5 deg/hr
Gyro sensitivity	0.06667 deg/sec/digit for ± 2000 deg/sec
Gyro non-linearity	0.2% full-scale
Compass scale	±0.88 Ga to ±8.1 Ga
Compass resolution	12 bit
Compass sensitivity	0.73 mGa/digit
Compass non-linearity	0.1% full-scale

cammino. Ciascun sandalo dispone di un driver custom-made (Fig. 4.5(b)) fissato nel lato laterale della calzatura, contenente tre amplificatori a due canali per poter comandare i motori vibro-tattili. Quando le cuffie non vengono utilizzate, uno speaker in miniatura può essere collegato direttamente al driver box. La logica di controllo responsabile dell'acquisizione e dell'elaborazione dei segnali dai sensori viene eseguita su un microcontrollore ARM a 32 bit (TEENSY 3.1, PJRC, LLC, Sherwood, OR) (Tab. 4.3).

Questa unità è stata racchiusa su una scatola in plastica posta sotto il tallone con un accelerometro a tre assi ed un'antenna Wi-Fi (XBee WiFi Module, Digi International Inc., Minnetonka, MN)(Tab. 4.4). I dati acquisiti vengono inviati attraverso protocollo UDP su WLAN ad un computer single-board (Odroid-U3, Hardkernel co., Ltd.) (Tab. 4.1) che sincronizza i dati, gestisce il motore di feedback e esegue il data logging in una scheda micro SD alla frequenza di 500 Hz. Oltre

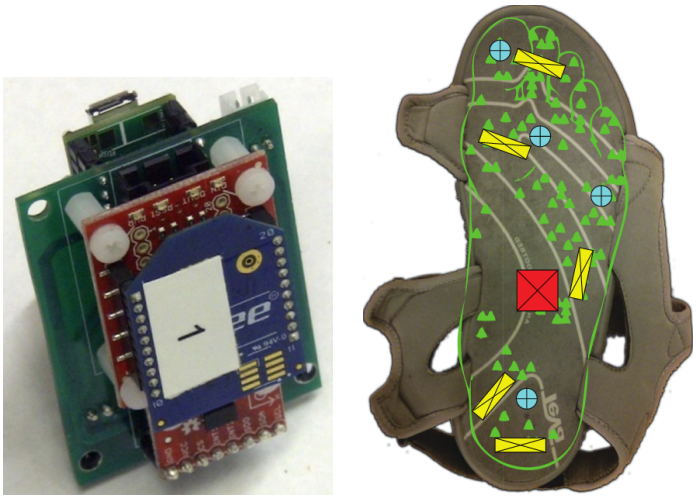
Tabella 4.3: Caratteristiche tecniche del microcontrollore



Core	ARM Cortex-M4
Rated Speed	96 MHz
Flash Memory	256 kbytes
RAM	64 kbytes
EEPROM	2 kbytes
Digital I/O	34 Pins
Analog Input	21 (@ 16 bit) Pins
Converters	2
Analog Output	1 Pin
DAC Resolution	12 Bits
Communication	1 USB, 3 serial, 1 SPI, 2 I2C, 1 CAN Bus

che al micro computer, il modulo marsupio alloggia anche una scheda audio esterna a 24 bit (Xonar U7, ASUSTeK Computer Inc.), un piccolo router Wi-Fi (TL-WR702N, TP-LINK Technologies Co., Ltd., Shenzhen, China) e due batterie Li-Po a tre celle da 2000 mAh.

È possibile generare in real-time un feedback sonoro grazie al motore di sintesi descritto in [68, 69], successivamente convertito in otto indipendenti segnali analogici - quattro per gamba - dalla scheda audio a bordo. Una coppia di sottili cavi audio in PET trasportano i segnali audio dalla vita al piede del soggetto. L'interfaccia grafica implementata in Matlab (The MathWorks, Natick, MA, USA), visibile in Fig. 4.9, consente allo sperimentatore di connettersi senza fili al controllore e permette di monitorare in tempo reale i dati misurati dai sensori e ottimizzare i parametri del motore di rendering audio. Il sistema completo può essere indossato in circa 5 minuti senza assistenza, il peso totale dei componenti collegati a ciascun sandalo è di circa 0.19 Kg mentre quello del modulo a marsupio è di circa 1.14 kg.



(a) (b)

Figura 4.3: (a) Scheda elettronica con microcontrollore, accelerometro e antenna. (b) Disposizione dei componenti nella suola della scarpa.

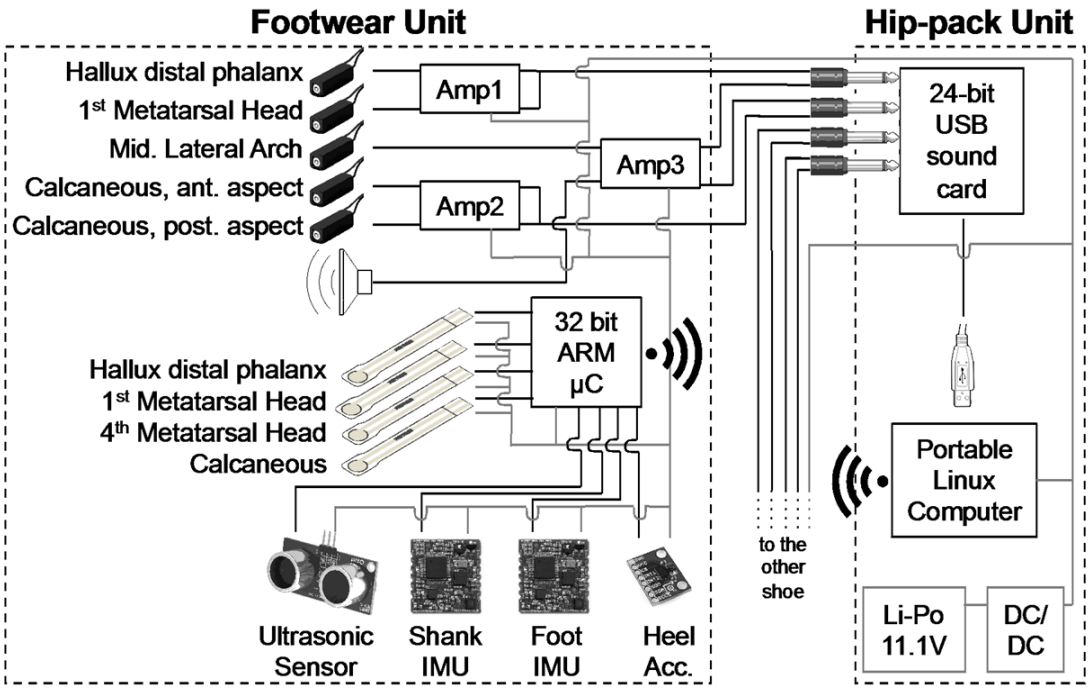
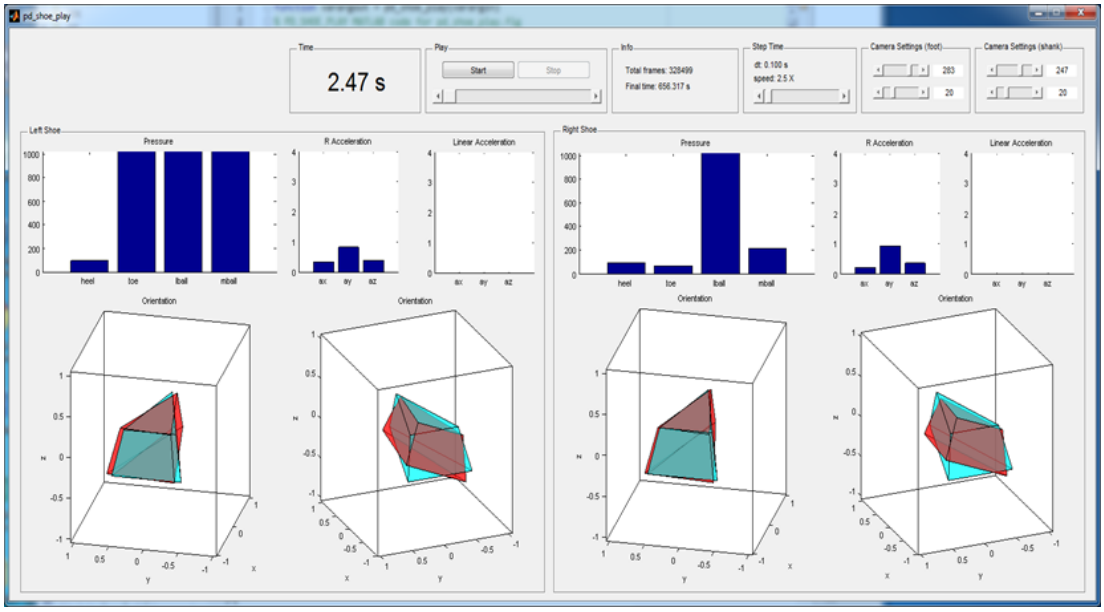


Figura 4.4: Schema generale

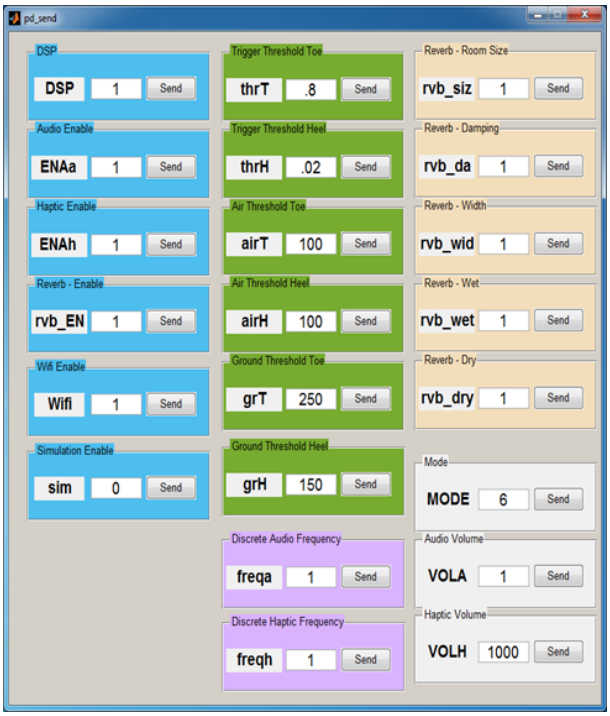
(a)

(b)

Figura 4.5: Particolari del sandalo instrumentato: in (a) è visibile il sonar utilizzato, mentre in (b) è presente un dettaglio del driver e del box contenente la componentistica elettronica



(a)



(b)

Figura 4.6: (a) GUI per il monitoraggio e confronto dei paramatri SoleSound e VICON ,(b) GUI per il settaggio dei parametri

Tabella 4.4: Caratteristiche tecniche del dispositivo wireless

Serial Data Interface	UART up to 1 Mbps, SPI up to 6 Mbps
Frequency Band	ISM 2.4 GHz
ADC Inputs	4 (12-bit)
Digital I/O	10
Security	WPA-PSK, WPA2-PSK and WEP
Channels	13 channels
Standard	802.11b/g/n
Data Rates	1 Mbps to 72 Mbps
Transmit Power	Up to +16 dBm (+13 dBm for Europe/Australia/Brazil)

4.3 Metodi

Ciascun IMU (Inertial Measurement Unit) del piede e della tibia fornisce la stima dell'orientazione relativa ad un sistema di riferimento (*tare*) attraverso un EKF. L'IMU del piede è incorporato all'interno della suola del sandalo, con l'asse locale $\hat{\mathbf{z}}_{\mathrm{F}}$ ortogonale alla suola con puntamento verso il basso e l'asse locale $\hat{\mathbf{x}}_{\mathrm{F}}$ allineato con l'asse longitudinale del sandalo. All'avvio, si chiede al soggetto di rimare fermo ed in posizione diritta per un periodo di 5 secondi e l'orientazione di riferimento tra i sensori inerziali del piede e della tibia vengono settati come in seguito descritto. I lavori medi di accelerazione misurati nella fase di startup definiscono la direzione del vettore gravità $\mathbf{g}$ relativamente al frame locale degli IMU del piede e della tibia. Il sistema di riferimento del piede $\{\mathrm{F0}\}$ viene definito come:

$$\mathbf{z}_{\mathrm{F0}} = \mathbf{g}, \quad \mathbf{x}_{\mathrm{F0}} = \frac{\hat{\mathbf{x}}_{\mathrm{F0}} - (\hat{\mathbf{x}}_{\mathrm{F0}} \cdot \mathbf{z}_{\mathrm{F0}})\, \mathbf{z}_{\mathrm{F0}}}{\left\| \hat{\mathbf{x}}_{\mathrm{F0}} - (\hat{\mathbf{x}}_{\mathrm{F0}} \cdot \mathbf{z}_{\mathrm{F0}})\, \mathbf{z}_{\mathrm{F0}} \right\|}, \quad \mathbf{y}_{\mathrm{F0}} = \mathbf{z}_{\mathrm{F0}} \times \mathbf{x}_{\mathrm{F0}}, \tag{4.1}$$

dove $\hat{\mathbf{x}}_{\mathrm{F0}}$ è l'asse locale $\hat{\mathbf{x}}_{\mathrm{F}}$ al tempo $t = 0$. L'IMU posizionato nella tibia

è attaccato nella tibia prossimale del soggetto per mezzo di un Velcro strap. L'asse locale $\hat{\mathbf{x}}_{\mathrm{S}}$ è assunto essere allineato con l'asse longitudinale della tibia, con puntamento verso l'alto, l'asse locale $\hat{\mathbf{z}}_{\mathrm{F}}$ è diretto posteriormente. In maniera similare al piede, il sistema di riferimento della tibia {S0} è definito come:

$$\mathbf{x}_{\mathrm{S0}} = -\mathbf{g}, \quad \mathbf{z}_{\mathrm{S0}} = \frac{\hat{\mathbf{z}}_{\mathrm{S0}} - (\hat{\mathbf{z}}_{\mathrm{S0}} \cdot \mathbf{x}_{\mathrm{S0}})\,\mathbf{x}_{\mathrm{S0}}}{\|\hat{\mathbf{z}}_{\mathrm{S0}} - (\hat{\mathbf{z}}_{\mathrm{S0}} \cdot \mathbf{x}_{\mathrm{S0}})\,\mathbf{x}_{\mathrm{S0}}\|}, \quad \mathbf{y}_{\mathrm{S0}} = \mathbf{z}_{\mathrm{S0}} \times \mathbf{x}_{\mathrm{S0}}, \tag{4.2}$$

con $\hat{\mathbf{z}}_{\mathrm{S0}}$ l'asse locale $\hat{\mathbf{z}}_{\mathrm{S}}$ al tempo $t = 0$. Assumendo una neutra posizione subtalare ed un neutrale allineamento del ginocchio durante il processo di taratura, la mappatura tra {F0} e {S0} è data dalla seguente matrice anti-diagonale:

$$\overset{\mathrm{S0}}{\mathrm{F0}}\mathrm{R} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{4.3}$$

Per $t > 0$, le stime di orientazione del piede e della tibia nei rispettivi sistemi di riferimento sono fornite in termini di angoli di Eulero (Yaw - Pitch - Roll). Insieme a (4.3), questi dati sono sufficienti a ricavare i tre angoli della caviglia, ovvero: abduzione/adduzione, inversione/eversione e plantare/dorsiflessione. Successivamente verrà considerato l'angolo plantare/dorsiflessione γ_{PD}, dal momento che è il maggiormente critico per la propulsione della camminata e per il supporto contro la forza di gravità [70]. γ_{PD} è definito come il relativo angolo di pitch tra il piede e la tibia, con un offset pari a $\pi/2$ [41]. Come visto in (4.3), gli assi $\mathbf{y}_{\mathrm{S0}}$ e $\mathbf{y}_{\mathrm{F0}}$ sono antiparalleli, fornendo:

$$\gamma_{\mathrm{PD}} = \theta_{\mathrm{F}} + \theta_{\mathrm{S}}, \tag{4.4}$$

dove θ_{F} e θ_{S} sono gli angoli di pitch per il piede e la tibia, rispettivamente. Per ciascuna gamba, l'angolo della caviglia (4.4) è suddiviso in gait cycle (GC) utilizzando le letture del sensore di pressione posizionato nel tallone per determinare il contato iniziale (IC). Per l'i-esimo passo di ciascuna gamba, l'angolo della caviglia è normalizzato rispetto al tempo di GC al fine di ottenere una traiettoria dell'angolo della caviglia normalizzata, $\bar{\gamma}_{\mathrm{PF\,i}}$. Il range di movimento dell'angolo della caviglia $\mathrm{ROM_i}$ è definito come la differenza assoluta tra il massimo ed il minimo valore di $\bar{\gamma}_{\mathrm{PF\,i}}$. La metrica di simmetria $\mathrm{SYM_i}$ è derivata come la

deviazione RMS tra due normalizzate traiettorie dell'angolo della caviglia della gamba destra e sinistra, corrispondenti a due passi successivi:

$$\mathrm{SYM_i} = \sqrt{\frac{\sum_{j=1}^{N} \left(\bar{\gamma}_{\mathrm{PF\_LEFT\ i,\ j}} - \bar{\gamma}_{\mathrm{PF\_RIGHT\ i,\ j}}\right)^2}{N}}, \tag{4.5}$$

con N che rappresenta il numero di campioni $\bar{\gamma}_{\mathrm{PF\ i}}$.

L'IMU posizionato nel sandalo fornisce i componenti del vettore accelerazione $\mathbf{a}$ (privato della componente gravità) nel sistema di riferimento {F0}. Un algoritmo basato su un valore di soglia permette di rilevare i periodi di foot-flat (FF) come frazione della fase di appoggio in cui la norma euclideo di $\mathbf{a}$ è minore di una soglia predefinita. Innanzitutto, la velocità del piede nell' i-esimo passo $\mathbf{v}_{\mathrm{i}}$ è ottenuta per integrazione di $\mathbf{a}$ dove le mediane dell' i-esimo e $(i+1)$-esimo periodo di FF definiscono l'i-esimo intervallo di integrazione [38]:

$$\mathbf{v}_{\mathrm{i,\ j}} = \mathbf{v}_{0\mathrm{i}} + \frac{1}{f_s} \sum_{k=\mathrm{ff}_i}^{\mathrm{ff}_i+j-1} \mathbf{a}_k, \quad j \in [1,\ \mathrm{ff_{i\ +\ 1}} \text{ - } \mathrm{ff_i} + 1], \tag{4.6}$$

$\mathbf{v}_{\mathrm{i,\ j}}$ è la velocità lineare del piede nel j-esimo campione dell i-esimo passo, e $[\mathrm{ff_i}, \mathrm{ff_{i\ +\ 1}}]$ è l'intervallo di integrazione per l'i-esimo passo. La costante di integrazione $\mathbf{v}_{0\mathrm{i}}$ è stata posta a zero (ZUPT technique) e le stima raw della velocità (4.6) sono corrette per compensare un modello lineare di drift della velocità:

$$\bar{\mathbf{v}}_{\mathrm{i,\ j}} = \mathbf{v}_{\mathrm{i,\ j}} - \frac{j-1}{\mathrm{ff_{i\ +\ 1}} - \mathrm{ff_i}} \mathbf{v}_{\mathrm{i,\ ff_{i\ +\ 1}} \text{ - } \mathrm{ff_i} + 1} \tag{4.7}$$

Lo spostamento del piede $\mathbf{d}_{\mathrm{i}}$ viene calcolato mediante integrazione di $\bar{\mathbf{v}}_{\mathrm{i}}$:

$$\mathbf{d}_{\mathrm{i,\ j}} = \frac{1}{f_s} \sum_{k=1}^{j} \bar{\mathbf{v}}_{\mathrm{i,\ j}}, \quad j \in [1,\ \mathrm{ff_{i\ +\ 1}} \text{ - } \mathrm{ff_i} + 1], \tag{4.8}$$

dove $\mathbf{d}_{\mathrm{i,\ j}}$ è lo spostamento del piede nel j-esimo campione dell' i-esimo passo. $\mathbf{d}_{\mathrm{i}}$ è noto nel sistema di riferimento {F0}, tuttavia, ai fini dell'analisi del cammino, un sistema di riferimento allineato con la direzione di progressione del cammino, definito come {Di}, è maggiormente desiderabile:

$$\mathbf{x}_{\mathrm{Di}} = \frac{\mathbf{d}_{\mathrm{i,\ ff_{i+1}\text{ - }ff_i+1}} - \mathbf{d}_{\mathrm{i,\ 1}}}{\left\|\mathbf{d}_{\mathrm{i,\ ff_{i+1}\text{ - }ff_i+1}} - \mathbf{d}_{\mathrm{i,\ 1}}\right\|}, \quad \mathbf{z}_{\mathrm{Di}} = -\frac{\mathbf{z}_{\mathrm{F0}} - (\mathbf{z}_{\mathrm{F0}} \cdot \mathbf{x}_{\mathrm{Di}})\,\mathbf{x}_{\mathrm{Di}}}{\left\|\mathbf{z}_{\mathrm{F0}} - (\mathbf{z}_{\mathrm{F0}} \cdot \mathbf{x}_{\mathrm{Di}})\,\mathbf{x}_{\mathrm{Di}}\right\|}, \tag{4.9}$$
$$\mathbf{y}_{\mathrm{Di}} = \mathbf{z}_{\mathrm{Di}} \times \mathbf{x}_{\mathrm{Di}}$$

Il piano sagittale, normalizzando la traiettoria per l'i-esimo passo $\bar{\mathbf{d}}_{\mathrm{i}}$, è ottenuto proiettando $\mathbf{d}_{\mathrm{i}}$ nel piano $\mathbf{x}_{\mathrm{Di}}\mathbf{z}_{\mathrm{Di}}$ e normalizzando rispetto alla durata del passo stesso nell'intervallo $[1,\ \mathrm{ff_{i+1}}\text{ - }\mathrm{ff_i}+1]$. Infine, lo stride-length $\mathrm{SL_i}$ ed il foot ground clearance $\mathrm{SH_i}$ sono definiti come:

$$\mathrm{SL_i} = \left|\bar{\mathbf{d}}_{\mathrm{i,\ N}}(x) - \bar{\mathbf{d}}_{\mathrm{i,\ 1}}(x)\right|, \quad \mathrm{SH_i} = \max_{\mathrm{j}\,\in\,[1,\ \mathrm{N}]}\left(\bar{\mathbf{d}}_{\mathrm{i,\ j}}(z)\right), \tag{4.10}$$

con $\bar{\mathbf{d}}_{\mathrm{i,\ j}}(x)$ e $\bar{\mathbf{d}}_{\mathrm{i,\ j}}(z)$ le proiezioni di $\bar{\mathbf{d}}_{\mathrm{i,\ j}}$ in $\mathbf{x}_{\mathrm{Di}}$ e $\mathbf{z}_{\mathrm{Di}}$, rispettivamente, e N il numero di campioni in $\bar{\mathbf{d}}_{\mathrm{i}}$.

The step width of the i-th stride SWi is therefore estimated by the absolute minimum of the ultrasonic sensor readings during the swing phase of the i-th left stride.

4.4 Protocollo sperimentale

Le metriche descritte nella sezione precedente sono influenzate da errori sistemici e casuali. Non solo questi errori possono essere quantificati sperimentalmente attraverso la comparazione dei dati raccolti rispetto ad un sistema da laboratorio di motion-capture, ma gli stessi dati possono essere utilizzati per calibrare il sistema , compensando ampiamente la sistematicità degli errori e quindi migliorando il livello di accordo tra i due sistemi di misura. Quattordici soggetti adulti sani, con nessuna anormalità nel cammino, sono stati inclusi nello studio (9 maschi, 5 femmine, età 26.6±4.2 anni, altezza 1.70±0.10 m, peso 64.9±10.6 kg), consistendo di una singola visita al laboratorio. La Columbia University Institution Review Board (IRB) ha approvato il protocollo per questo studio e tutti i soggetti hanno dato il proprio consenso informato prima di partecipare alla sessione sperimentale. La tabella 4.5 mostra i dati demografici dei soggetti.

I soggetti, dopo aver scelto un'appropriata misura della calzatura, hanno indossato l'attrezzatura come visibile in figura 4.8(a). Dei marker riflettenti (Fig. 4.7)

Figura 4.7: Posizionamento dei marker durante un esperimento di validazione

sono stati collocati su entrambe le gambe che su punti di riferimento anatomici (malleolo mediale e laterale, condili femorali, tibia distale e prossimale). In figura 4.8(b) e 4.7 è possibile osservare la loro disposizione. Prima di effettuare il test, ai soggetti è stato chiesto di stare in posizione verticale per 5 secondi. In questo frangente temporale i sensori inerziali sono stati azzerati e mai reimpostati durante la sessione di test. I partecipanti hanno completato 30 giri selezionando in modo autonomo un passo della camminata confortabile. Ad ogni giro, i soggetti camminavano lungo un percorso rettilineo di 14 m segnato sul pavimento, effettuavano un giro orario al termine del percorso e ritornavano al punto di partenza. La durata di ogni sessione è stata di circa 15 minuti ciascuna. I movimenti dei soggetti sono stati registrati sia da SoleSound sia da un sistema di motion-capture basato su 10 telecamere (VICON Bonita B10). Le frequenze di campionamento sono state fissate a 500 Hz e 100 Hz rispettivamente ed un LED all'infrarosso, comandato da SoleSound, è stato utilizzato per poter sincronizzare i due sistemi. In figura 4.8(b) è rappresentato l'ambiente di test ed il LED utilizzato per il sincronismo tra i due sistemi.

(a) (b)

Figura 4.8: (a) Soggetto mentre indossa il prototipo SoleSound. (b) Soggetto durante un esperimento di validazione: la linea verde rappresenta il percorso effettuato, il cerchio rosso il dispositivo di sincronismo tra SoleSound e VICON

Tabella 4.5: Caratteristiche antropometriche dei soggetti

Height [cm]	Weight [kg]	Shoe Size [US]	Age [y]	Gender
172	58	8.5	30	0
163	55	5.5	23	F
168	60	8	24	M
180	68	10	31	M
190	85	13	30	M
165	58	6.5	22	F
164	56	5.5	29	F
150	54	4	28	F
180	74	11	23	M
175	72	9.5	22	M
170	85	7	36	M
160	57	5	24	F
168	59	9.5	23	M
173	68	9.5	27	M

4.5 Analisi dei dati

Il dispositivo è in grado di misurare diverse metriche della camminata, nello specifico: *stride length*, *foot-ground clearance*, *base of walking*, *ankle ROM*, *ankle RMS Error*, *foot trajectory (x)*, *foot trajectory (y)* ed *ankle angle*. Mentre i primi parametri sono scalari, le traiettorie del piede (in x e y) e gli angoli della caviglia sono vettoriali; quest ultimi sono normalizzati rispetto alla percentuale del Gait Cycle ($0 - 100\%$), composti quindi da 101 elementi.

L'analisi delle metriche raw di SoleSound e dei dati collezionati dal sistema di motion capture, come reference system, sono stati processati attraverso appositi script Matlab (The Mathworks). Al fine di stabilire l'ordine e la frequenza di taglio opportuni del filtro alla Butterworth utilizzato per filtrare i dati di accelerazione forniti dai sensori, per determinare la soglia della norma euclidea dell'accelerazione con cui individuare il periodo di foot-flat (FF) ed il relativo fronte di considerazione e' stato sviluppato un apposito script di ottimizzazione. L'ottimizzazione è stata effettuata esclusivamente nei dati utilizzati nel training.

In questa sottosezione verranno descritte le strategie di calibrazione utilizzate:

subject-specific e generic. La prima permette di validare i dati ottenuti da SoleSound avendo a disposizione i dati di calibrazione dello specifico soggetto, la seconda invece consente di simulare la validazione del dispositivo con un soggetto non presente nella fase di calibrazione.

Subject specific calibration

Le componenti di training-set e testing-set, per il parametro considerato, sono state create a partire dai dati contenuti nel data-set di ciascun soggetto. In pratica, gli elementi del data-set riguardanti lo specifico parametro e soggetto sono stati assegnati in modo alternato rispettivamente al training-set ed al testing-set. Considerando i dati grezzi ottenuti da SoleSound e dal reference system, per ogni parametro analizzato e per ogni soggetto sono stati ottenuti i seguenti "vettori": p^S_{tr}, p^S_{ts}, p^V_{tr} e p^V_{ts}, che rappresentano i data-set di training e testing per SoleSound e reference system, rispettivamente.

Per ciascun parametro in analisi e per ciascun soggetto sono stati computati N distinti linear regression model, di sintassi:

$$p^V_{tr}(j) \sim p^S_{tr}(j), \quad j \in [1, N] \tag{4.11}$$

ottenendo, quindi, gli incogniti parametri di regressione $\beta_{0,j}$ e $\beta_{1,j}$, dove j rappresenta il j-esimo degli eventuali N campioni della variabile p. La nuova stima della misura del parametro ottenuto da SoleSound viene computata, al fine di compensare gli errori sistematici e di aumentare il livello di adesione tra i due sistemi di misura, attraverso la seguente formula:

$$\hat{p}^S_i(j) = \beta_{0,j} + \beta_{1,j}\ p^S_{ts,i}(j), \quad j \in [1, N] \tag{4.12}$$

dove p^S_{ts} indica la misura del parametro ottenuto da SoleSound, dal dataset di testing, i rappresenta l'i-esimo passo analizzato di uno specifico soggetto ed j il j-esimo degli N campioni della variabile in esame p.

L'errore di misura tra SoleSound ed il reference system (VICON), di un dato parametro per un determinato soggetto, viene computato come:

$$e_i(j) = \hat{p}^S_i(j) - p^V_{ts,i}(j), \quad j \in [1, N] \tag{4.13}$$

dove i e' l'i-esimo passo analizzato di uno specifico soggetto ed j il j-esimo degli N campioni della variabile p.

Generic calibration

L'intero processo è stato sviluppato in conformita' con le specifiche di leaveone-out cross validation (LOOCV). Ad ogni iterazione sono stati selezionati 13 su 14 soggetti per poter effettuare il training del modello, mentre i dati relativi al soggetto escluso sono stati utilizzati per effettuare la validazione tra SoleSound ed il reference system con il modello appena elaborato. Il modello lineare, per lo specifico parametro, e' stato costruito a partire dai dati dei 13 soggetti in esame, incorporando, in questa strategia di validazione, anche le caratteristiche antropomorfiche dei soggetti. (Tab. 4.5). Per ogni parametro preso in considerazione, tale procedura e' stata iterata per 14 volte, creando ogni volta due dataset dei dati p^V_{tr}, p^S_{tr}, contenente i dati dei 13 soggetti presi in esame per il training del modello ed altri due dataset p^V_{ts} e p^S_{ts}, con i dati del singolo soggetto preso in considerazione per il testing. L'apice V indica i dati ottenuti dal reference system mentre l'apice S le metriche grezze di Solesound. In ogni iterazione, quindi, viene calcolata la discrepanza di misura tra il reference system e SoleSound per il parametro da valutare.

Per ciascun parametro in analisi e' stato computato un distinto modello di regressione lineare, di sintassi:

$$p^V_{tr}(j) \sim p^S_{tr}(j)_{+\text{Height + Weight + Shoe Size + Age + Gender}},\ \ j \in [1, N] \tag{4.14}$$

dove $p^V_{tr}(j)$ rappresenta la variabile dipendente (dati collezionati dal reference system) e $p^S_{tr}(j)$ (dati ottenuti da SoleSound) con le caratteristiche antropometriche dei soggetti, i predittori del modello lineare. j e' il j-esimo degli N, eventuali, campioni della variabile p presa in esame. La valutazione del modello permette di ottenere gli $m + 2$, con m il numero delle caratteristiche antropometriche dei soggetti inserite nel modello, coefficienti di regressione $(\beta_0 \dots \beta_{m+1})$, specifici per il parametro considerato. Dati i coefficienti di regressione β, la nuova stima del parametro $\hat{p}^S$ del soggetto preso in esame e' computata come:

$$\hat{p}^S_i(j) = \beta_{0,j} + \beta_{1,j}\ p^S_{ts,i}(j) + \sum_{k=1}^{m} \beta_{k+1,j}\ x_k, \quad j \in [1, N] \tag{4.15}$$

dove p_{ts}^{S} indica la misura raw del parametro ottenuto da SoleSound, dal dataset di testing, i rappresenta l'i-esimo passo analizzato dello specifico soggetto ed j il j-esimo degli N campioni della variabile in esame p. Il vettore x rappresenta le covariate relative ai dati antropometriche del soggetto.

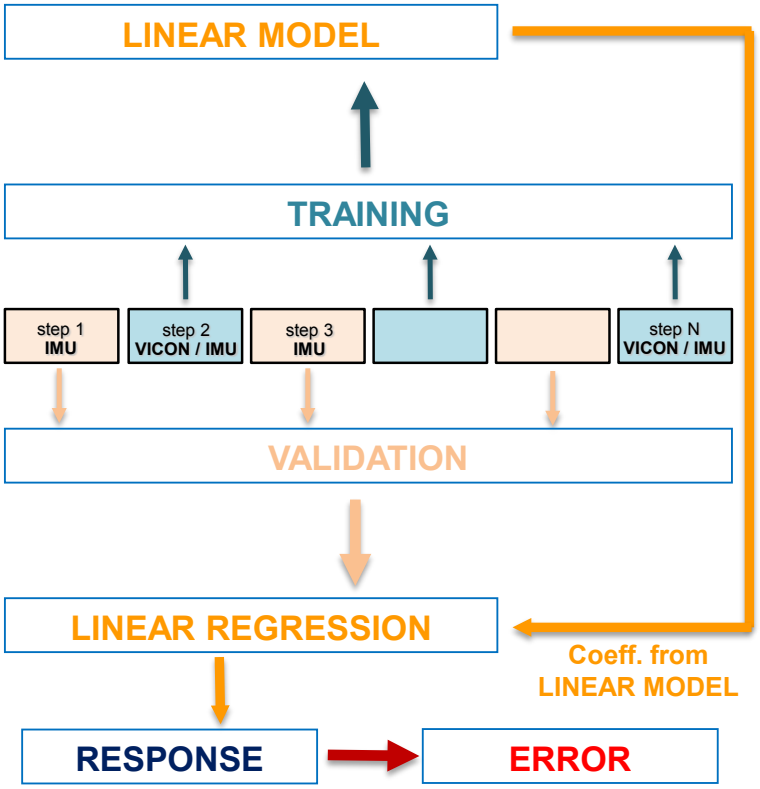
L'errore di misura tra SoleSound ed il reference system, di un dato parametro per il soggetto utilizzato nel testing, viene calcolato come in (4.13).

4.6 Risultati e conclusioni

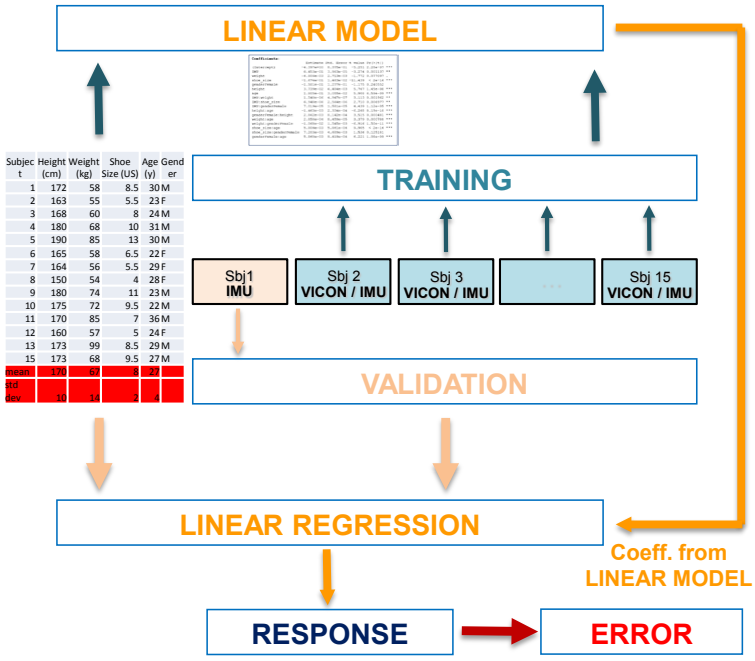
In Tab. 4.6 sono riportati i risultati ottenuti per le diverse metriche analizzate, nelle due strategie di calibrazione effettuate. In Fig. 4.10 e 4.11 sono riportati, per alcuni parametri scalari, i grafici di correlazione tra SoleSound ed il reference system, la distribuzione di frequenza dell'errore di misura ed i plot di Bland-Altman[1]. Sono inoltre raffigurate la traiettoria del piede, media per uno specifico soggetto, ed l'angolo di dorsiflessione della caviglia, medio fra tutti i soggetti, computate dal dispositivo e dal reference system utilizzato.

Dati gli incoraggianti risultati ottenuti, nonostante SoleSound sia stata progettata con l'obiettivo di essere utilizzata in ambito riabilitativo, si potrebbe pensare di continuare lo sviluppo di un dispositivo similare per poter determinare la posizione ed alcuni movimenti del soggetto che la indossa. Tale caratteristica potrebbe trovare applicazione anche in un contesto industriale per applicazioni di interazione uomo-robot. Se utilizzato come sensore indossabile, permetterebbe di ricavare con accuratezza la posizione ed i movimenti dall'operatore all'interno di una cella robotizzata permettendo così di attuare le procedure necessarie affinché quest'ultimo operi con la massima sicurezza. Un approccio basato su dispositivi indossabili garantisce di determinare la posizione dell'operatore senza i problemi di occlusione di cui soffrono i sistemi di visione. Inoltre i sistemi di videosorveglianza potrebbero non essere conformi alla normativa sulla tutela della libertà e dignità

[1]Il Bland-Altman plot serve per confrontare due misure della stessa natura; è un diagramma di dispersione in cui sulle ordinate viene riportata la differenza delle due misure e sulle ascisse la misura di riferimento, ottenuta come media aritmetica delle due misure. Le linee orizzontali rappresentano la media delle differenze, e la media delle differenze ± 1.96 x SD.



(a) Schematizzazione della calibrazione *subject-specific*



(b) Schematizzazione della calibrazione *generic*

Figura 4.9: Strategie di calibrazione

dei lavoratori (Legge 20 Maggio 1970, n. 300) mentre un sistema indossabile ne garantirebbe il rispetto.

Tabella 4.6: Risultati (mean RMSE ± SD)

	Subject specific	Generic
Stride Length [cm]	2.30 ± 0.90	2.93 ± 1.32
Foot-Ground Clearance [cm]	0.38 ± 0.10	0.70 ± 0.37
Base of Walking [cm]	0.82 ± 0.19	1.54 ± 0.70
Ankle ROM [deg]	2.12 ± 0.63	4.76 ± 1.91
Ankle RMS Error [deg]	1.95 ± 0.38	2.72 ± 1.53
Foot Trajectory [cm]	3.38 ± 2.39	4.70 ± 3.57
Ankle Angle [deg]	2.70 ± 0.39	4.33 ± 1.01

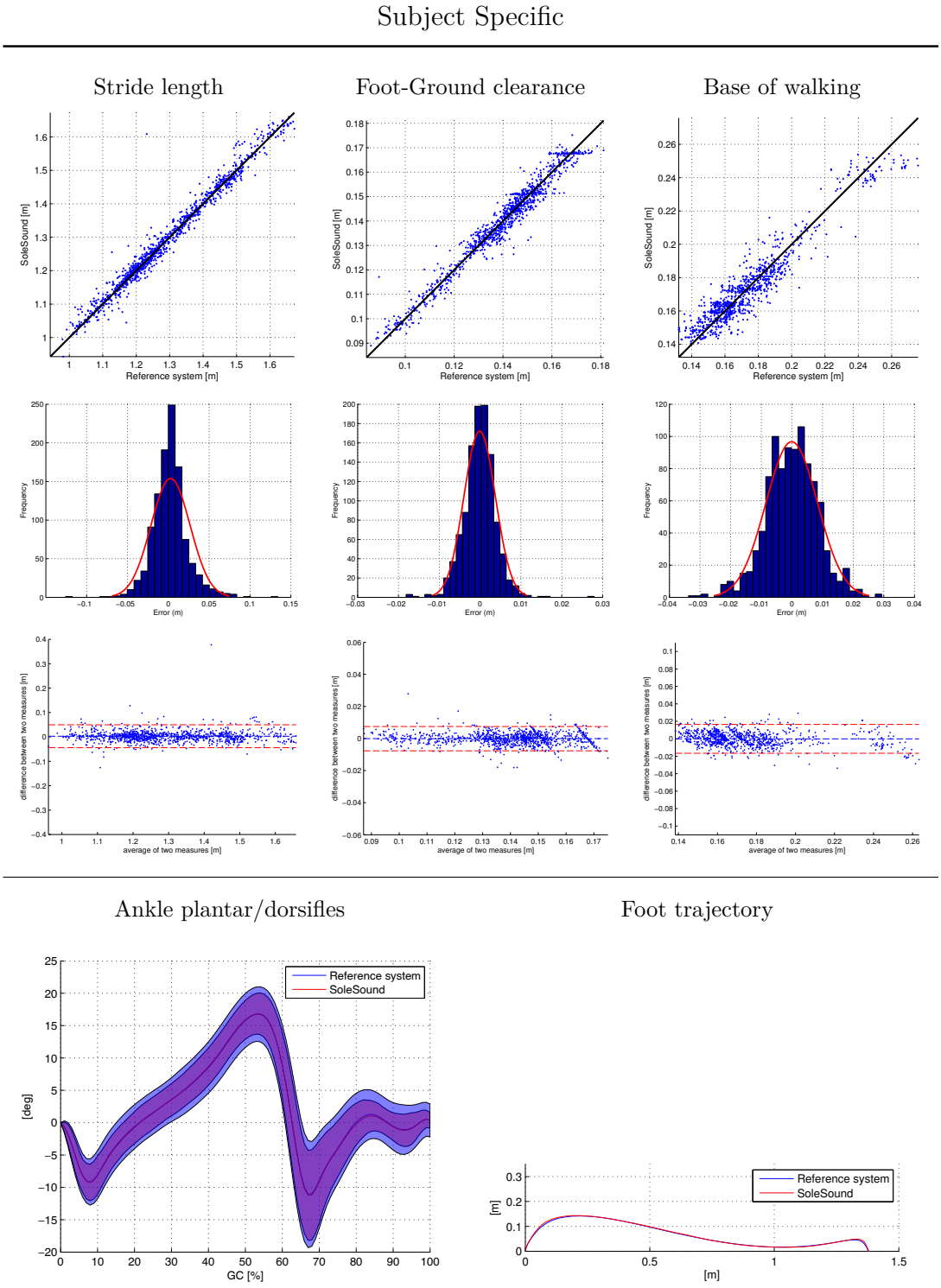
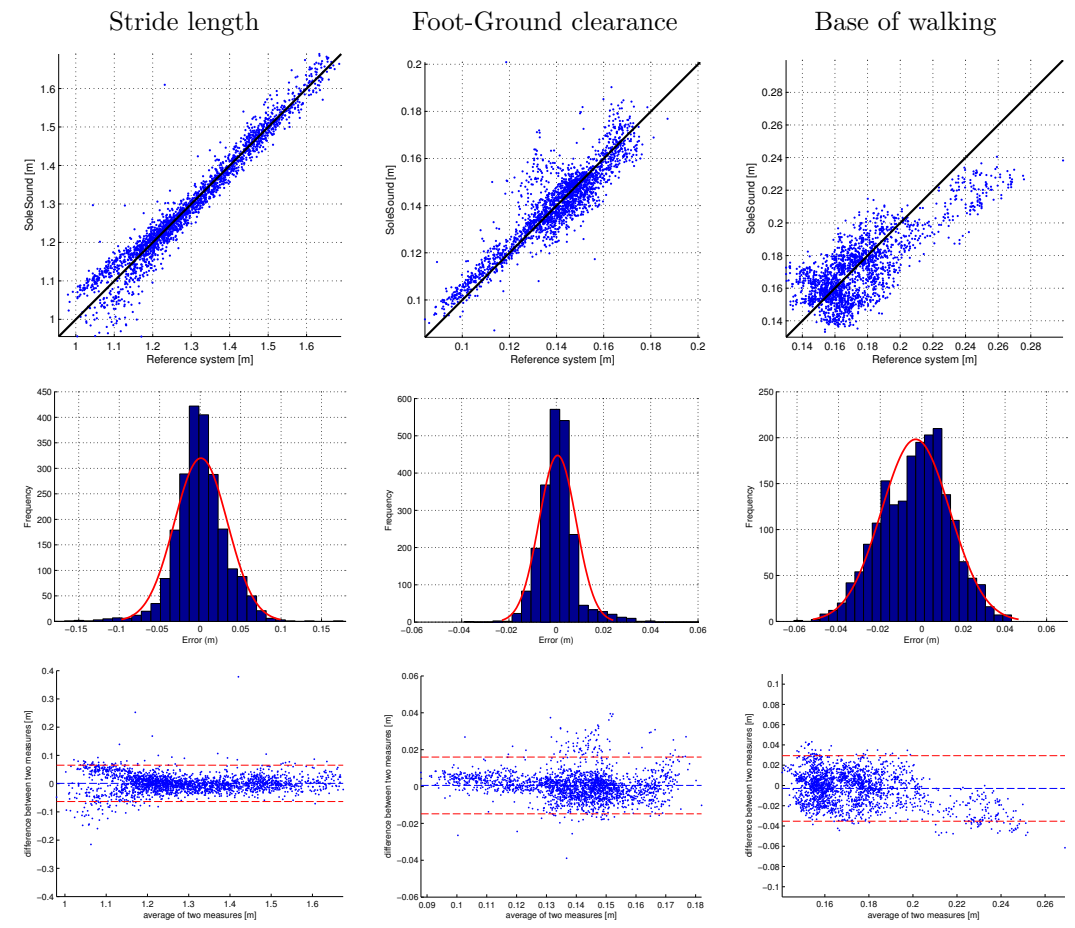
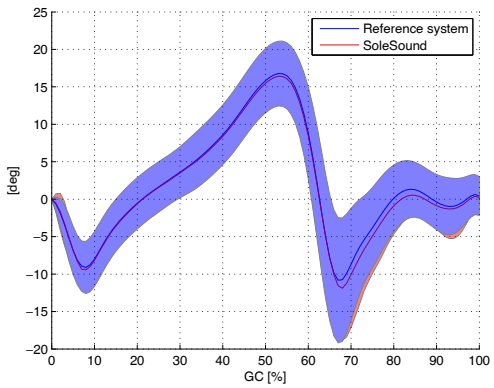


Figura 4.10: Risultati soggetto-specifici

Generic



Ankle plantar/dorsifles



Foot trajectory

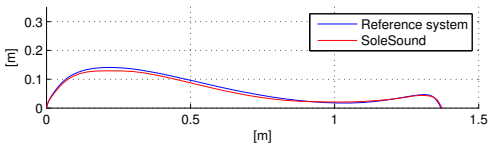


Figura 4.11: Risultati non soggetto-specifici

Capitolo 5

Controllo robot con dispositivo basato su sensori MEMS

In questo capitolo verrà descritta la progettazione di un nuovo dispositivo di input impugnabile basato su sensori inerziali MEMS per la movimentazione rapida ed intuitiva di robot industriali al posto del classico utilizzo di teach-pendant. Verrà fornita una rapida descrizione dell'hardware e del sistema di controllo a basso livello del dispositivo, successivamente verranno descritte le tecniche di controllo utilizzate per l'interfacciamento del joystick ad un controllore robot industriale.

5.1 Introduzione

Solitamente i robot industriali sono programmati per eseguire azioni precise e ripetitive, in maniera da eseguire un dato compito (posizionamento, lavorazione, verniciatura, etc.) nel minor tempo possibile. In queste situazioni vi è la necessità di acquisire dei punti dello spazio di lavoro per la realizzazione della movimentazione desiderata. A tal scopo si possono utilizzare le *teach pendant* (Fig. 5.1), ossia dei dispositivi di controllo direttamente connessi al controller del robot e che ne permettono il moto tramite la pressione di opportuni pulsanti. Questi strumenti hanno la capacità di muovere il manipolatore agendo direttamente nello spazio operativo oppure nello spazio dei giunti, dando la possibilità di variare la configurazione di ogni asse in maniera indipendente. Questa operazione, all'apparenza semplice, richiede tuttavia molta dimestichezza e generalmente personale specializzato, non permettendo comunque il raggiungimento di elevate prestazioni data la complessità e la ridotta velocità raggiungibile. Si tratta quindi di una tecnica macchinosa, per nulla efficace e poco intuitiva, non *user-friendly* e inadatta a controllare il manipolatore in tempo reale.
Per rendere il sistema più orientato all'utente, si è scelto quindi di studiare e sviluppare un dispositivo basato su sensori elettromeccanici (MEMS), in maniera da tradurre le operazioni di orientazione e movimentazione del joystick tramite l'uso combinato di un sensore di accelerazione, un giroscopio ed un magnetometro. Attraverso l'utilizzo dei dati forniti dal joystick e con un'architettura di controllo specifica è possibile ottenere un sistema sensibile e pratico per la movimentazione di un manipolatore industriale.

La richiesta di eseguire movimenti in tempo reale richiede delle specifiche molto stringenti: tutte le elaborazioni e i ritardi di propagazione devono essere ridotte al minimo, in maniera tale da garantire una latenza minima. La bassa latenza è richiesta per avere una percezione del movimento del manipolatore il più possibile legata all'attuazione da parte dell'utente, senza che il ritardo introdotto tra generazione del movimento e sua attuazione comprometta l'impressione di fluidità.

Figura 5.1: *Teach pendant* TP1 per robot EPSON.

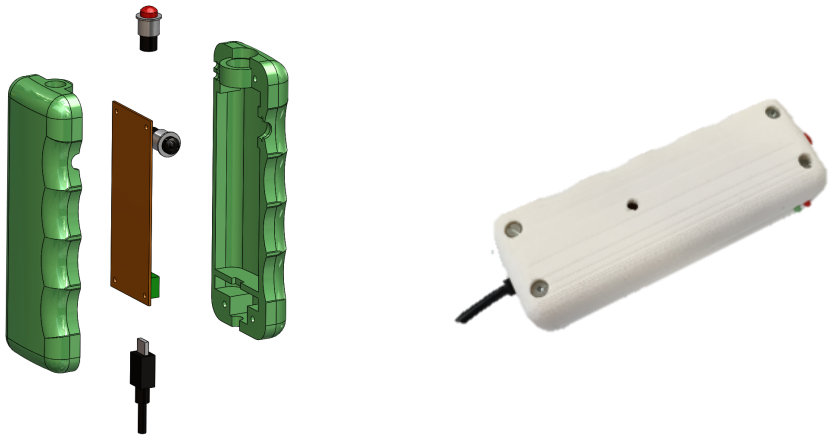
5.2 Descrizione del prototipo di I/O

5.2.1 Hardware

Il dispositivo, il cui involucro esterno è stato realizzato mediante tecnica di prototipazione rapida, contiene al suo interno una scheda elettronica composta da un microcontrollore ARM, un accelerometro, un giroscopio ed un magnetometro. Sono stati utilizzati gli stessi componenti impiegati nella costruzione del prototipo SoleSound, descritto nel Capitolo 4; si rimanda alle Tab. 4.3 e 4.2 per una approfondita descrizione dei componenti. Il device dispone anche di due led e due pulsanti che possono essere utilizzati come dispositivi di output ed input del device. In Fig. 5.2 è presente la raffigurazione dell'esploso e la realizzazione fisica del dispositivo.

5.2.2 Software

Date le notevoli capacità computazionali del microcontrollore impiegato, il sistema di controllo a basso livello sviluppato si basa su una architettura multiprocesso su un sistema operativo hard-real time, nello specifico FreeRTOS. I sistemi operativi hard-real time vengono infatti tipicamente utilizzati in ambito industriale, in quanto è necessario ottenere una risposta dal sistema in un tempo massimo



(a) (b)

Figura 5.2: Esploso del dispositivo (a) e realizzazione finale (b)

prefissato. Le API messe a disposizione dal sistema operativo consentono, in modo agevole di:

- creare e gestire i task, potendone assegnare e variarne la priorità;
- gestire le code di comunicazione fra task;
- gestire semafori binari e semafori con counter;
- creare mutex per la sincronizzazione tra processi.

In Fig. 5.3 è possibile osservare l'organigramma dei task implementati per il sistema di controllo del dispositivo. Il thread principale implementa una macchina a stati finiti per la gestione del controllo, mentre altri thread si occupano dell'acquisizione dei dati (attraverso protocollo SPI) ed al loro processamento. Un thread è dedicato alla comunicazione bidirezionale del device, la quale avviene attraverso interfaccia USB 2.0. La comunicazione tra processi è stata implementata attraverso code di messaggi con politica FIFO.

5.2.3 Comunicazione

Il protocollo di comunicazione viene gestito mediante invio e ricezione di pacchetti di dimensione fissa di 64 bytes. Si è scelto questo approccio in quanto la trama di

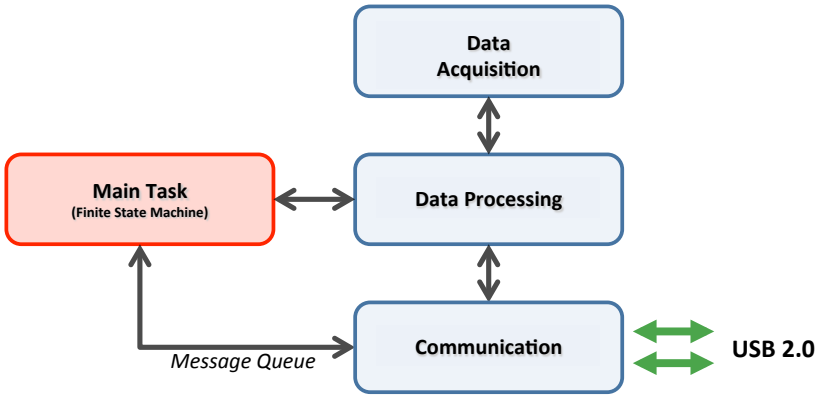


Figura 5.3: Task

comunicazione del protocollo USB 2.0 permette di inviare o ricevere al minimo tale mole di dati.

In Tab. 5.2.3 è rappresentato un tipico pacchetto che il joystick spedisce. I campi `START` e `STOP` rappresentano i bytes che delimitano il pacchetto, il campo `CMD` e `PARAM` rappresentano rispettivamente il comando ed il parametro a cui il device, in precedenza interrogato, sta rispondendo. Il byte `HW` fornisce informazioni a riguardo dei pulsanti premuti e dei led accesi, nel campo `TIMESTAMP` è presente il timestamp, in microsecondi, del microcontrollore quando effettua la lettura dei sensori. Nella word `CRC` è presente il valore di un semplice controllo di ridondanza ciclico. I campi appena descritti, ed in grassetto nella rappresentazione grafica del protocollo, sono sempre fissi per tutti i comandi e relativi parametri. Nell'esempio vengono anche restituiti i quaternioni e le accelerazioni lineari in float a singola precisione. Gli altri parametri, come i dati raw dei sensori, vengono rappresentati con 2 bytes; ovvero i valori rilevati vengono divisi per una quantità nota e successivamente castati ad interi con segno.

5.3 Descrizione del robot

Il robot utilizzato è il modello C4-A601S (Fig. 5.4) prodotto dalla Epson Seiko, il più recente manipolatore a 6 assi prodotto dall'azienda, caratterizzato da uno sbraccio di 600 mm e una ripetibilità di ± 0.02 mm.Una panoramica delle principali specifiche della serie C4 è riportata in Fig. 5.5. Il controller associato al robot è

START		
1	2	3

CMD	PARAM
4	5

q_0				q_1				q_2				q_3			
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

la_x				la_y				la_z			
22	23	24	25	26	27	28	29	30	31	32	33

a_x		a_y		a_z		g_x		g_y		g_z		m_x		m_y		m_z		–		
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54

HW	TIMESTAMP				CRC	
55	56	57	58	59	60	61

STOP		
62	63	64

Tabella 5.1: Protocollo di comunicazione

l'Epson RC700.

Figura 5.4: Modello Epson C4 e controller Epson RC700

SPEL+: comandi di movimento

In questa sottosezione verrà fornita una breve panoramica sui comandi di movimento principalmente utilizzati,al fine di capire quali siano migliori per il tipo di applicazione in uso.

All'interno dell'ambiente di sviluppo EPSON SPEL+ esistono vari comandi utilizzati per eseguire un movimento verso un punto specificato, e ognuno ha le sue peculiarità che lo rendono più adatto ad un determinato tipo di operazione:

- *Go*: esegue una movimentazione point-to-point (PTP) dalla posizione corrente fino al punto specificato. La traiettoria generata non è predicibile, se non

EPSON ROBOT MODEL		C4-A60***	C4L-A90***
Configuration		Articulated 6 Axis	
Mounting Configurations		Table Top, Ceiling	
Payload		Rated 1 kg / Max. 4 kg (5 kg*1)	
Repeatability		±0.02 mm	±0.03 mm
Axis Rotation	J1 (Turning)	±170 deg	
	J2 (Lower Arm)	-160 ~ +65 deg	
	J3 (Upper Arm)	+225 ~ -51 deg	
	J4 (Wrist Roll)	±200 deg	
	J5 (Wrist Bed)	±135 deg	
	J6 (Wrist Twist)	±360 deg	
Horizontal Reach	to mounting face	665	965
	to wrist center	600	900
Vertical Reach	to mounting face	885	1185
	to wrist center	820	1120
Speed	J1	450 deg/sec	275 deg/sec
	J2	450 deg/sec	275 deg/sec
	J3	514 deg/sec	289 deg/sec
	J4	555 deg/sec	
	J5	555 deg/sec	
	J6	720 deg/sec	
Cycle Time	1 kg workload	0.37 sec	0.47 sec
Allowable Moment	J4	4.41 N*m	
	J5	4.41 N*m	
	J6	2.94 N*m	
Moment of Inertia	J4	0.15 kg*m^2	
	J5	0.15 kg*m^2	
	J6	0.1 kg*m^2	
Motor Ratings	Axis 1	400 watts	
	Axis 2	400 watts	
	Axis 3	150 watts	
	Axis 4	50 watts	
	Axis 5	50 watts	
	Axis 6	50 watts	
Environment	Temperature	0 ~ 40 deg C	
	Humidity	20 ~ 80% RH (no condensation)	
	Other:	Keep away from: *Flammable/corrosive gases *Flammable/corrosive solvents *Water or oil and powder dust *Sources of electric noise	
Weight		27 kg	29 kg

Figura 5.5: Principali specifiche dei manipolatori industriali della serie C4 Epson Seiko.

con tecniche di reverse engineering, per cui non è possibile studiare a priori con esattezza il percorso effettuato. Nonostante ciò, si tratta di una delle più comuni istruzioni per il posizionamento: il comando necessita, nella sua forma più banale, del solo set di coordinate del punto di arrivo, e permette al robot di raggiungere il punto desiderato nella maniera più semplice. Il difetto principale di questa istruzione, però, è quella di non poter raggiungere elevate velocità nel caso in cui vengano richieste movimentazioni successive i cui punti di destinazioni non siano presenti nel programma compilato.

- *Move*: esegue una movimentazione lineare dalla posizione corrente fino al

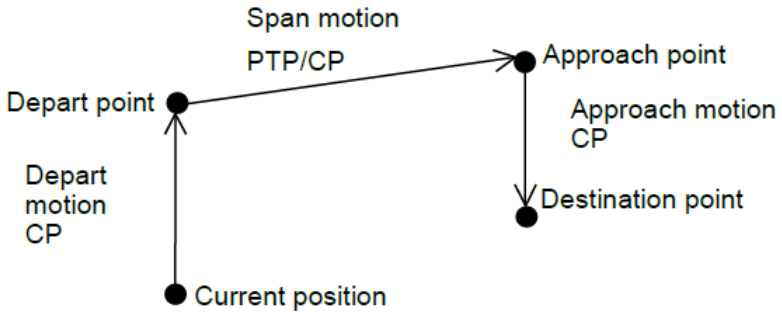


Figura 5.6: Caratteristiche del comando di movimento *Jump*.

punto specificato. Si differenzia dal comando *Go* per il fatto che, in questo caso, il movimento viene eseguito lungo la retta congiungente il punto corrente a quello di destinazione.

- *Jump*: esegue un successione di tre movimenti: *depart*, *span* e *approach*. Il comando necessita di tre punti in ingresso: il primo identifica gli estremi per la prima parte di movimento (dalla posizione corrente a quella di *depart*), il secondo identifica il movimento di *span* point-to-point oppure lineare, a seconda dell'impostazione scelta, dal punto di *depart* a quello di *approach*, mentre l'ultimo punto è quello di destinazione che conclude il movimento dal punto di *approach* precedentemente specificato (Fig. 5.6). Si tratta di un'istruzione largamente utilizzata per processi di *pick and place*, e dal punto di vista costitutivo non si discosta molto da una sequenza di tre *Go* per il movimento PTP, oppure da una sequenza di *Go*, *Move*, *Go* per il movimento lineare.

- *Curve-CVMove*: diversamente dagli altri comandi sopra descritti, questo tipo di movimento richiede due passaggi: la creazione di un file contenente tutte le informazioni necessarie e l'esecuzione del moto secondo le specifiche contenute in tale file. In particolare, il comando *Curve* richiede in ingresso una serie di punti (minimo 3, massimo 200), con cui verrà creata una spline che verrà salvata su un file .CVT, pronta per essere utilizzata dal successivo comando *CVMove* (Fig. 5.7). In questo caso la procedura utilizzata per creare il moto è quella di una interpolazione con spline cubica, per cui il

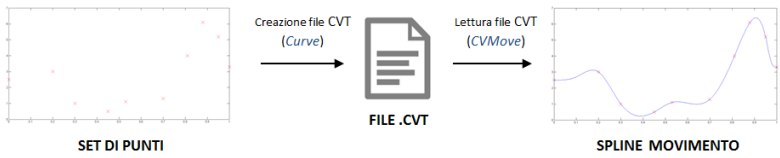


Figura 5.7: Procedura di utilizzo dei comandi *Curve-CVMove*

moto è predicibile e ricostruibile. Lo svantaggio principale di questo comando è il (leggermente) maggiore tempo richiesto per l'elaborazione e la necessità di operare su file, ma ha il grande vantaggio di permettere il raggiungimento di elevate velocità e di garantire il passaggio per tutti i punti con estrema precisione. Il comando *Curve* non può essere utilizzato quando la distanza tra due punti successivi è inferiore a 0.001 *mm*.

- *CP (Continuous Path)*: non si tratta di un vero e proprio comando di movimento quanto di un parametro aggiuntivo da abbinare a uno qualunque delle istruzioni precedenti. Questa opzione permette di eseguire due istruzioni di movimento in sequenza impedendo che il robot si fermi al punto di arrivo per poi ripartire. In sostanza l'abilitazione del CP fa sì che, in prossimità del punto di arrivo, la decelerazione non sia completa ma permetta al manipolatore di continuare il movimento verso il punto successivo senza fermare completamente il moto, come mostrato in Fig. 5.8. Com'è ovvio, questa opzione non garantisce il passaggio verso il punto specificato, in quanto solitamente la pianificazione viene effettuata imponendo velocità finale nulla, e l'attivazione del CP rende imprevedibile il movimento in prossimità del punto di passaggio (nel caso *Curve*, invece, i punti intermedi vengono tutti raggiunti, mentre ciò non può dirsi per il punto finale).

5.4 Strategia di controllo

In questo paragrafo viene presentato l'algoritmo utilizzato per il controllo in tempo reale del robot tramite la piattaforma di sensori precedentemente definita. La piattaforma comunica con il manipolatore per mezzo di due interfacce separate:

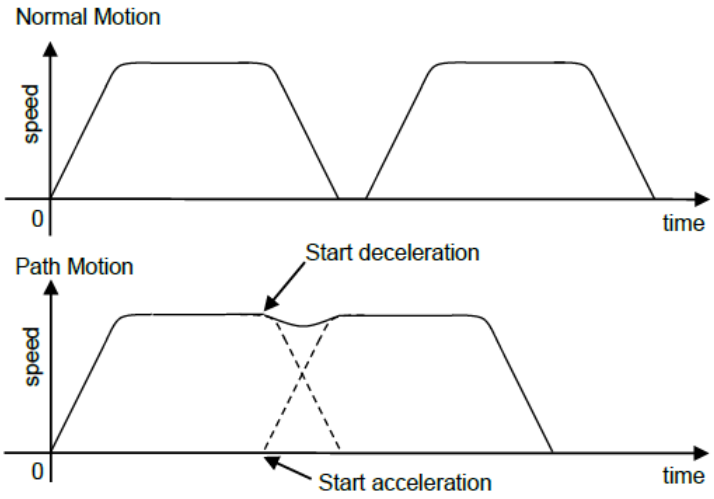


Figura 5.8: Effetti del parametro *CP*.

una lato Matlab e l'altra lato EPSON. In particolare, una GUI Matlab si occupa di raccogliere, processare e inviare i dati relativi al movimento del robot con un'interfaccia grafica che agevola la gestione del dispositivo, mentre un programma SPEL+ risiedente nel lato EPSON raccoglie i pacchetti inviati dalla GUI con protocollo TCP/IP e attua le operazioni richieste per muovere il robot secondo le specifiche desiderate.

Fissato il periodo di campionamento con il quale vengono richiesti i dati dal joystick, questi possono essere processati e inviati direttamente al robot sotto forma di coordinate spaziali (sestuple, 3 coordinate per il posizionamento e 3 per l'orientazione). Il programma EPSON si preoccuperà poi di analizzare i punti inviati e di attuare il movimento per raggiungerli secondo le tempistiche previste. Una prima idea potrebbe essere quella di inviare al controller del robot un punto alla volta, e attendere che la movimentazione corrente sia esaurita prima di inviare il punto successivo; in pratica, si tratta di inviare un set di coordinate ad ogni passo di campionamento. Se da un lato questa tecnica minimizzerebbe la latenza tra generazione del movimento e sua attuazione, in quanto la distanza temporale sarebbe di un solo periodo di campionamento, dall'altro introdurrebbe notevoli complicazioni, alcune di carattere teorico e altre di carattere pratico.
Dal punto di vista teorico, un approccio di questo tipo richiedere la conoscenza esatta della velocità di percorrenza da impostare per garantire il passaggio at-

traverso i punti indicati nel tempo di campionamento imposto. Sebbene questo sia un problema generale, dovendo trattare un solo punto alla volta c'è maggior rischio di incorrere, nel caso di stima errata, in un movimento "stop and go" in prossimità di ogni punto di passaggio piuttosto che in un moto continuo e senza interruzioni.
Dal punto di vista pratico, invece, si riscontra una difficoltà insormontabile che è forse la vera causa dell'impraticità di questo approccio: tra i comandi disponibili, si riconoscono il *Go*, il *Move* e il *CVMove*. Il comando *Go* è sicuramente l'alternativa più semplice, tuttavia presenta delle notevoli vibrazioni durante il movimento e, soprattutto, non riesce a raggiungere delle velocità consistenti. Il comando *Move*, invece, permette il raggiungimento di elevate velocità, ma i percorsi in linea retta rendono il moto molto irregolare e discontinuo, nonché pieno di oscillazioni. Il *CVMove* si basa direttamente sul file creato con il comando *Curve*, il quale per creare la spline corrispondente necessita di almeno 3 punti di passaggio. Questo ovviamente è un requisito impossibile da ottenere con l'approccio in esame, che prevede di inviare un punto alla volta, a meno che non si esegua una qualche forma di interpolazione per introdurre artificialmente dei punti intermedi.

Occorre quindi fare una prima distinzione fondamentale: per poter fare in modo che il manipolatore riesca a seguire i movimenti imposti dall'utente è necessario utilizzare come comando di movimento l'istruzione *CVMove*. Questo pone però dei limiti all'uso dell'approccio precedentemente descritto: non è possibile inviare al programma EPSON un solo punto alla volta, poiché rappresenta l'unico limite del comando *Curve*.
Ne consegue quindi che l'alternativa più efficace risulta l'utilizzo del comando *Curve* a cui vengono inviati il minimo dei punti occorrenti, ossia 3. Poiché il punto finale della spline n-esima coincide con il punti iniziale della spline $(n+1)$-esima, sono necessari solo due nuovi punti per ogni spline, in sostanza ciò significa introdurre una latenza di due intervalli di campionamento.

I dati campionati dalla piattaforma vengono utilizzati dal software Matlab per creare il set di punti corrispondente, e verranno successivamente inviati al programma EPSON mediante protocollo TCP/IP, il quale li utilizza per creare i file .CVT necessari a permettere il movimento del robot in "real-time fashion".

Il modo e le tempistiche con cui questi file vengono creati rappresenta l'aspetto peculiare del programma EPSON: poiché il set di punti inviati non può essere utilizzato per muovere direttamente il manipolatore (il comando *CVMove* infatti legge solo i file .CVT creati con il comando *Curve*), è necessaria una politica di ottimizzazione per impedire che questa limitazione si ripercuota sul movimento del robot, causando rallentamenti o, peggio ancora, movimenti irregolari e discontinui. Dovendo quindi lavorare con i file *Curve* (i file .CVT), si nota innanzitutto come non sia possibile creare ogni volta un nuovo file da accodare ai successivi: di questa strada infatti si verrebbero ad accumulare continuamente nuovi file che, dovendo essere letti ed eseguiti una sola volta ciascuno, sarebbero inutili e superflui dopo il primo utilizzo. D'altro canto lavorare su un solo file *Curve* è impossibile, dal momento che durante l'esecuzione del movimento corrispondente, ossia durante la lettura del file, non è possibile accedervi per modificarlo con operazioni di scrittura.
È necessario dunque definire fin da subito quale strategia utilizzare per la lettura/scrittura delle spline, in particolare quanti debbano essere i file totali e quali siano i meccanismi di creazione/esecuzione.

5.4.1 Gestione dei file *Curve*

Come già detto in precedenza, occorre trovare innanzitutto un limite massimo per il numero dei file presenti durante l'esecuzione. La scelta più semplice porta a considerare un massimo di 2 spline, che però rappresenta solo il punto di partenza verso la realizzazione di una infrastruttura solida e robusta per la gestione e il processamento di ogni tipo di dato. L'utilizzo di due sole spline infatti, nonostante rappresenti la scelta più semplice, implica anche numerosi cicli di controllo necessari a rendere l'algoritmo più completo ed affidabile per ogni tipo di dato.
Utilizzando questo metodo, si destina una spline al moto corrente e l'altra alla pianificazione del successivo. La difficoltà maggiore sta nell'adattare il moto corrente in maniera che sia esaurito esattamente quando sta per iniziare il successivo, in questo modo si ha una dinamica precisa e quasi immediata. Si potrebbe pensare quindi di predisporre un ciclo di controllo lato EPSON che si occupi di informare Matlab qualora si stia per passare da una spline all'altra, in modo tale da garantire

l'invio di punti solamente in queste condizioni. Questo implica l'invio di pacchetti con un numero variabile di punti e la necessità di predisporre inoltre di un *timeout* per evitare che la richiesta di invio avvenga dopo un intervallo temporale massimo. Questo fornisce uno spunto per un controllo automatico della velocità: l'occorrenza del timeout implica che tale parametro è stato sottostimato, e occorre dunque un suo aumento per impedire che si verifichino nuovi casi. Viceversa, i casi in cui la spline viene terminata ancora prima che siano disponibili i punti per la successiva implica una sovrastima del parametro velocità, per cui occorre ridurlo. Questo approccio risulta eccessivamente appesantito da tutti gli accorgimenti necessari per permettere una corretta esecuzione, e nonostante ciò presenta alcuni problemi che ne intaccano il funzionamento:

- Ci sono numerosi casi limite che portano ad un malfunzionamento della struttura. Uno fra questi è l'occorrenza del timeout un istante prima del termine della spline: in questo modo verranno richiesti nuovi dati, che il software interpreterà come punti da accodare alla spline successiva. Purtroppo però in questo caso si tratta di un *false timeout*, in quanto il programma EPSON occuperà la spline successiva per accodare i punti, paralizzando di fatto il movimento che richiederebbe la lettura immediata del file *Curve* in uso per appending. Ciò si traduce in un'ulteriore processo di sofisticazione dell'algoritmo, che deve anche considerare la validità di ogni timeout e diventerebbe così ancora più pesante.

- Una regolazione a $+\Delta v$, $-\Delta v$ occasionali come quella proposta non sarebbe sufficiente a far convergere in un tempo ragionevole la velocità reale verso il valore ottimo: l'incremento/decremento di velocità avviene solo durante i timeout o le chiusure anticipate di set di punti che arrivano *in ritardo* rispetto all'azione attuale dell'utente, ma durante i set di punti successivi la velocità ideale da imporre potrebbe comunque subire delle ulteriori variazioni.

- La gestione "fisica" dei punti, intesa come uso delle locazioni di memoria, risulta molto sconveniente utilizzando questa pratica: utilizzando due buffer separati, nel caso di false timeout, ogni predizione sbagliata richiederebbe di spostare i nuovi punti nell'altro buffer. Si potrebbe allora pensare di

usare un buffer circolare, nonostante sia più difficile da gestire, ma poiché il software prevede di inviare al comando *Curve* gli estremi della serie di punti da considerare (punto iniziale e punto finale), l'interfaccia tra la fine e l'inizio del buffer rappresenta un problema di coerenza sintattica, che per essere risolto richiede un ulteriore ciclo di controllo.

L'approccio più semplice è quello dell'applicazione della definizione di velocità nel dominio discreto: dato un set di n punti, rappresentante il pacchetto da inviare, di cui sono riconoscibili le tre componenti di posizionamento spaziale x, y e z, si può impostare la velocità come:

$$v = A_f \sum_{k=1}^{n-1} \frac{\sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2}}{nT_c} \tag{5.1}$$

dove A_f è un parametro di fitting, da regolare per ottenere una dinamica il più possibile simile a quella ottimale, x_k è il valore della coordinata x del k-esimo set di punti e T_c è il periodo di campionamento. Se il parametro A_f è unitario la formula corrisponde al rapporto tra la distanza euclidea del set di punti e tra il tempo impiegato per percorrerli. Risulta subito ovvio che la formula che ne consegue per il calcolo della velocità risulta valida se e solo se il percorso effettuato lungo i vari punti corrisponde ad una spezzata (comando *Move*). Ciò è ovviamente in contrasto con il moto imposto dal comando *Curve*, il quale presuppone un movimento tramite spline cubiche, per cui il risultato finale sarà un errore che porta generalmente ad una sottostima del parametro di velocità necessario, in quanto l'uso di traiettorie basate su polinomi cubici aumenta il percorso totale (Fig. 5.9).

Occorre quindi settare manualmente il valore del parametro di fitting per ottenere il risultato più consono.

Questa tecnica risulta più efficace della precedente con invio asincrono, tuttavia presenta un problema molto simile: se in caso di lieve sottostima l'invio di punti viene effettuato quando la spline è in fase conclusiva, l'altro file viene utilizzato per scrittura e rimane inutilizzabile da altri finché è in atto la procedura di pianificazione. Purtroppo se la spline corrente sta per terminare, il moto trova occupato il file successivo e dovrà fermarsi in attesa che in tale file sia terminata l'operazione di

Figura 5.9: Differenza tra percorso spezzato (distanza euclidea, a) e percorso a spline (distanza generalizzata, b), per lo stesso set di punti.

scrittura. L'operazione di sottostima avviene con una certa frequenza, considerati i problemi espressi in precedenza, e questa situazione degrada purtroppo il moto del manipolatore. Poiché il problema deriva dall'attuale configurazione scelta per i file *Curve*, la sola strada percorribile per mantenere la funzionalità dell'algoritmo senza appesantirlo eccessivamente è aumentare il numero di spline a 3: l'uso di 3 spline elimina infatti il fastidioso problema degli errori di salvataggio all'interfaccia, rendendo l'algoritmo complessivo più fluido e, soprattutto, simmetrico.
Il principio di funzionamento è simile ai casi precedenti, in cui in un dato momento una sola spline può essere occupata per lettura (cioè per movimento), tuttavia ognuna di loro possiede un bit di validità che indica se il corrispondente file contiene dati validi oppure no: dopo ogni lettura (movimento) il corrispondente bit di validità viene posto a 0, mentre dopo ogni salvataggio tale bit viene portato a 1. Inoltre, per poter mantenere una traccia del movimento, si utilizza una struttura a token, il quale può essere posseduto da un solo file per volta e indica la spline attualmente in uso per lettura. In questa maniera, ad ogni arrivo di nuovi dati, si creano i rispettivi file *Curve* secondo la seguente gerarchia:

1. *Spline corrente*: si tratta della spline momentaneamente utilizza per il moto e corrisponde al file possessore del token. Se il suo bit di validità è uguale a 0, significa che si è interessati al movimento ma non ci sono ancora dati disponibili nella spline corrispondente; in tal caso i dati ricevuti saranno subito utilizzati per creare il file in esame. Questa situazione, vista come situazione di emergenza e che non dovrebbe mai verificarsi (tranne alla prima occorrenza), è rappresentativa di due casi particolari: errore nei

parametri di velocità (manipolatore troppo veloce) oppure una ripartenza da stop imposto dall'utente. La seconda situazione accade almeno una volta durante il movimento (all'inizio il robot parte da fermo) e in questi casi non rappresenta un problema. Viceversa, il caso di sovrastima del parametro di velocità porta il robot ad essere troppo veloce e a presentare questa occorrenza spesso, e ciò significa che il parametro di fitting non è stato impostato nel modo giusto. Quando invece il bit di validità è posto a 1, significa che la spline corrente è già pronta, come dovrebbe essere nel movimento normale, per cui il salvataggio può proseguire nelle successive spline.

2. *Spline successiva*: si tratta della spline successiva a quella momentaneamente in uso, ossia quella che verrà letta immediatamente dopo il termine del movimento corrente. Se il suo bit di validità è uguale a 0, ciò implica che il movimento successivo non è ancora stato predisposto, e occorre utilizzare i set di punti ricevuti per organizzarlo il prima possibile. Se, viceversa, tale bit vale 1, allora anche il movimento successivo è pronto e l'unica alternativa rimane il file a sua volta successivo.

3. *Spline posteriore*: è la spline "successiva della successiva", ossia che è avanti di due rispetto alla posseditrice del token (cioè quella in uso). Quando il bit di validità corrispondente è posto a 0, ciò significa che sono state organizzate la spline corrente e quella seguente, mentre quella ancora successiva è libera per il salvataggio. Questo comportamento dovrebbe essere quello comune durante l'esecuzione, rappresentativo della strategia ottimale per ottenere un movimento fluido e senza alcun tipo di interruzioni per l'elaborazione o il salvataggio. Quando invece il bit di validità è uguale a 1, ciò implica che tutte le spline fino a quella posteriore sono state salvate, e che i nuovi punti non hanno apparentemente un posto in cui essere memorizzati. In realtà, analogamente a quanto già visto, in questo caso i punti posso essere aggiunti alla spline posteriore (append), senza intaccare il funzionamento o la coerenza della struttura globale. C'è da dire che questo caso non voluto è

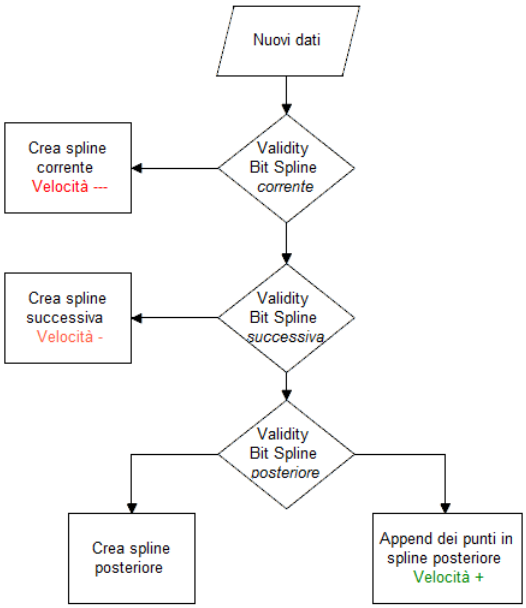


Figura 5.10: Schematizzazione dell'algoritmo proposto.

legato al problema della sottostima del parametro di velocità, solo in questo caso infatti il robot può ritrovarsi in arretrato con la lettura dei file.

In Fig. 5.10 è possibile vedere una schematizzazione dell'algoritmo appena presentato.

Ecco quindi come, con questo meccanismo, l'utilizzo di tre spline rimuova tutte le complicazioni derivate dall'imprevedibilità della ricezione dei dati, rendendo la procedura valida in qualunque situazione.
Il parametro di fitting va di conseguenza regolato per avere il funzionamento ottimale (corrente occupata, successiva occupata, posteriore libera), e in questo modo è possibile verificare che non ci potranno mai essere interruzioni.
Si può pensare di aggiungere un controllo *feedback* per regolare il parametro di fitting in maniera automatica. Per fare ciò, si possono sfruttare gli effetti dei casi particolari sopra descritti utilizzando una variabile specifica:

- In caso di spline corrente occupata (a parte la partenza da fermo, in cui questa occorrenza è del tutto necessaria), ossia in caso di grave sovrastima del parametro velocità, la variabile assume il valore -5δ.

- In caso di spline successiva occupata (a parte la partenza da fermo, in cui questa occorrenza è del tutto necessaria), ossia in caso di sovrastima del parametro velocità, la variabile assume il valore $-\delta$.

- In caso di append nella spline posteriore, ossia in caso di sottostima del parametro velocità, la variabile assume il valore $+\delta$ per ogni occorrenza.

Ad ogni ciclo, il valore della variabile viene utilizzato come "regolazione fine" per impostare il valore del parametro di fitting A_f incrementandolo o decrementandolo della quantità opportuna.

Utilizzando tutti questi accorgimenti, si perviene all'organizzazione finale dell'algoritmo, con 3 spline a gestione gerarchica e ad invio sincrono di dati. Questa struttura si è dimostrata robusta, affidabile e precisa, e rappresenta la modalità di controllo definitiva per il sistema complessivo.

5.5 Implementazione lato EPSON

Verrà ora descritta più in dettaglio l'implementazione effettiva della versione finale dell'algoritmo precedentemente enunciato. Il software EPSON SPEL+ permette, tra le altre cose, l'utilizzo di un massimo di 16 task paralleli, sicuramente utili per eseguire computazioni contemporanee (e separate), nonostante occorra prestare più attenzione durante la sincronizzazione dei vari task (accesso simultaneo alla stessa porzione di memoria).

Il programma principale del lato EPSON (*main*) si occupa interamente della gestione e del controllo di tutti i sub-task, agendo quindi da *supervisor*. Si tratta del codice più importante in quanto controlla istante per istante, in un ciclo infinito, se si sono verificati errori o situazioni di emergenza e, in tal caso, blocca il flusso del programma e restituisce il codice e il messaggio di errore verificato. Si tratta inoltre del programma di comunicazione principale verso l'esterno dal momento che, sempre all'interno di ogni ciclo, controlla il buffer di rete per verificare se sono stati inviati nuovi pacchetti dall'esterno, e si preoccupa inoltre di gestire l'eventuale invio di pacchetti da parte del software EPSON. I pacchetti, sempre all'interno del codice supervisore, verranno analizzati e scomposti nelle istruzioni

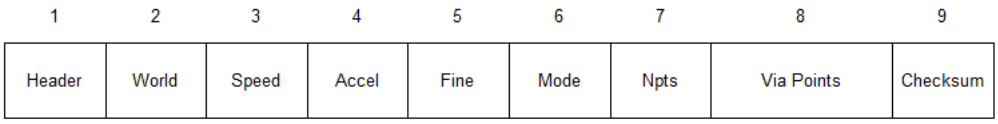


Figura 5.11: Struttura del pacchetto *Curve*.

fondamentali che lo costituiscono e, una volta riconosciuta tale istruzione sulla base del loro *header*, viene chiamata la funzione corrispondente in forma di task parallelo oppure subroutine. Ovviamente tutto ciò presuppone anche dei controlli di validità sulle azioni da eseguire: ad esempio occorre tenere traccia dell'eventuale movimento in atto, per far sì che il task *main* riconosca immediatamente se il robot è disponibile a ricevere un comando di movimento o, in caso contrario, sarà costretto a ignorare (oppure inserire in un apposito buffer) un successivo comando di moto.
Nel caso in esame, i pacchetti che permettono al robot di muoversi con le metodologie finora descritte prendono il nome di pacchetti *Curve*. Un pacchetto di questo tipo è rappresentato in Fig. 5.11 ed è così composto:

1. *Header* (4 byte): l'header identificativo del pacchetto. Serve ad attivare la modalità *Curve* sul programma EPSON per l'utilizzo del robot in real-time mode;

2. *World* (4 byte): selettore dello spazio operativo. Se posto a 1, specifica che le coordinate che seguono sono espresse nello spazio operativo (*world mode*), se invece vale 0, le coordinate seguenti dovranno essere intese nello spazio dei giunti (*joint mode*);

3. *Speed* (4 byte): selettore di velocità. Rappresenta il valore di velocità da impostare nel successivo ciclo, espresso come percentuale della velocità massima;

4. *Accel* (4 byte): selettore di accelerazione. Rappresenta il valore di accelerazione da impostare nel successivo ciclo, espresso come percentuale dell'accelerazione massima;

5. *Fine* (4 byte): selettore di accuratezza. Rappresenta il valore dell'accuratezza da impostare nel successivo ciclo con il comando *FineDist*. Si tratta di un parametro espresso in *mm*, da un minimo di 0.001 a un massimo di 10, ed esprime l'errore massimo che può essere tollerato durante il posizionamento;

6. *Mode* (4 byte): selettore della modalità di movimento. Se posto a 1 esegue il moto lungo i punti con il comando *Curve*, quindi utilizzando un'interpolazione di spline cubiche. Se posto a 2, invece, esegue il movimento utilizzando il comando *Go*, utile quando sono previsti movimenti lenti oppure di sola orientazione;

7. *Npts* (4 byte): indicatore del numero di punti. Esprime il numero di punti n contenuti nel frame successivo;

8. *Via Points* ($4n$ byte): set di punti di passaggio. In questo frame sono contenute le coordinate degli n punti inviati, utilizzabili per creare il file *Curve* o, eventualmente, per eseguire direttamente il movimento con il comando *Go*;

9. *Checksum* (4 byte): codice di controllo per verificare l'integrità del messaggio.

Questa è la struttura base di un comune pacchetto utilizzato per far muovere il robot secondo le tecniche precedentemente descritte. In esso sono contenuti tutti i dati necessari per adattare il movimento del manipolatore a quello imposto dall'utente. In particolare, c'è la possibilità di utilizzare l'istruzione *Go* al posto del *Curve*, questo è necessario qualora si voglia variare l'orientazione mantenendo fissa la posizione. In tal caso, infatti, il comando *Curve* non è in grado di generare alcun file, in quanto i punti successivi hanno posizione coincidente e non superano il requisito della minima distanza (0.001 *mm*).

Dal punto di vista del controllo, una volta riconosciuto il pacchetto da parte del task *main*, vengono avviate le procedure per il trattamento dei dati e la traduzione nel movimento corrispondente. Ciò avviene per mezzo di due task paralleli, chiamati *PlanCurve3* e *moveCurve*. Tali task rimangono attivi finché sopraggiungono nuovi pacchetti e, in caso di interruzione, vengono soppressi dopo un timeout di 5 secondi. Finché questi task rimangono in background, nessun altro

tipo di movimento è ammesso, e per interromperli immediatamente senza aspettare il timeout è sufficiente inviare un apposito pacchetto chiamato *StopCurve*.
Di questi due task, il primo (*PlanCurve3*) serve per la corretta pianificazione delle spline negli opportuni file, mentre il secondo (*moveCurve*) si occupa della giusta movimentazione utilizzando i file disponibili. La cooperazione tra di essi è favorita dall'uso di opportune variabili condivise, adibite a semafori e mutex per l'accesso esclusivo a determinate porzioni di codice.
Nei sottoparagrafi successivi verranno descritti più in dettaglio entrambi i task e le tecniche utilizzate per la sincronizzazione.

5.5.1 Task *PlanCurve3*

Viene attivato in automatico alla ricezione del primo pacchetto *Curve*. Rimane attivo finché non viene disattivato con il comando *stopCurve* oppure se sono passati più di 5 secondi dalla ricezione dell'ultimo pacchetto. Viene abilitato all'ingresso nel suo loop principale dopo che il task *main* ha ricevuto un nuovo pacchetto e ne ha verificato l'idoneità, informando il *PlanCurve3* con un segnale software di attivazione.
Lo scopo principale di questo task è pianificare nell'opportuna spline i punti ricevuti, dipendente dalla posizione del moto corrente e dallo status dei vari bit di validità. Altro compito del programma è quello di inserire nei buffer circolari di velocità e accelerazione i valori opportunamente letti dal corrente pacchetto. Questo accorgimento è necessario poiché l'impostazione di questi parametri deve avvenire coerentemente con il movimento, non può quindi essere impostato alla ricezione del pacchetto che solitamente avviene con qualche spline di anticipo. Tuttavia questa non può essere prerogativa del task *moveCurve*, il quale opera solamente sui file *Curve* ed è distaccato dalla ricezione dei pacchetti.
Dopo aver completato questa operazione, il task procede con il salvataggio dei punti negli appositi slot e con la creazione dei file di movimento, utilizzando l'algoritmo descritto nella Sez. 5.4. In questo caso il token è rappresentato per comodità da una variabile globale, che può assumere i valori da 1 a 3 a seconda del numero della spline momentaneamente in lettura. La creazione di un file *Curve* con 3 punti richiede tipicamente meno di 1 *ms*, per cui questo processo non occupa una

significativa quantità di tempo. Se invece si è preferito utilizzare il comando *Go*, allora questa operazione non sarà necessaria, ed è sufficiente il semplice salvataggio dei punti. Per informare il task di movimento se una determinata parte prevede il *CVMove* o il *Go*, al bit di validità viene sostituita un'ulteriore variabile globale che può assumere il valore 0 nel caso la movimentazione non sia valida (nessuna traiettoria oppure traiettoria già eseguita), 1 se la traiettoria in esame è un file *Curve* oppure 2 se si tratta di un set di *Go*.
In questa maniera il task *PlanCurve3* predispone il programma ad eseguire il moto necessario, sia esso di traslazione-rotazione oppure solamente di rotazione. In particolare, questa azione viene eseguita dal task *moveCurve*, descritto successivamente.

5.5.2 Task *moveCurve*

Anche questo task viene attivato in maniera automatica alla ricezione del primo pacchetto *Curve*. Rimane attivo finché è attiva la sua controparte *PlanCurve3*, ossia quando quest'ultimo termina per i motivi già citati, allora si conclude anche il *moveCurve*.
Il task comincia ponendo a 1 il bit di movimento, che rimane a tale valore finché il programma è in esecuzione, per segnalare che nessun altro comando di movimento oltre a quello del pacchetto *Curve* è ammesso. Successivamente, il task entra nel loop principale, dove parte dalla Spline1 e controllo il bit di validità ad esso corrispondente. Se vale 0, rimane in attesa di una sua variazione controllandone periodicamente il valore, in attesa che il task *PlanCurve3* prepari le specifiche necessarie. In caso contrario, procede innanzitutto impostando velocità e accelerazione opportunamente lette dai buffer opportuni, e successivamente esegue la movimentazione appropriata. Una volta terminato il movimento lungo la prima spline, il task passa alla seconda e aggiorna la variabile globale che funge da token, procedendo come nel caso precedente e così via in maniera circolare.
In questo modo ogni traiettoria può essere eseguita nella maniera opportuna senza generare conflitti e ognuna con le proprie caratteristiche dinamiche.

Il problema della scrittura simultanea nelle stesse variabili globali da parte di due task diversi non si verifica mai, in quanto ogni scrittura da parte di un

task funge da acknowledgement per l'altro, ed è necessario rilevare tale acknowledgement per poter proseguire con un'ulteriore scrittura. Questo garantisce al programma complessivo robustezza e stabilità senza complicarlo ulteriormente, ma soprattutto conferisce all'algoritmo la snellezza necessaria per poter trascurare le computazioni e il ritardo da esse introdotto.

5.6 Implementazione lato Matlab

Il programma Matlab si occupa della relazione che intercorre tra utente e punti, ossia la parte di generazione. Si tratta in sostanza di una GUI Matlab, dotata di periferiche di interfaccia software che permettono una migliore interazione con l'ambiente circostante.
Fondamentalmente la GUI, dopo aver inizializzato tutti i parametri, rimane in attesa della connessione con la piattaforma. Finché permane in questo stato, non sarà possibile eseguire nessuna funzionalità, né variare i parametri né tantomeno comunicare con il robot. La situazione si sblocca solamente quando viene riconosciuta una connessione seriale con la piattaforma e viene premuto l'apposito pulsante "*Connetti*".
A questo punto la GUI aggiorna la sua interfaccia e inizia a comunicare con la piattaforma, richiedendo a intervalli regolari un pacchetto con le informazioni richieste. Tutto ciò che concerne poi il calcolo dei corrispondenti angoli nei sistemi di riferimento arbitrari è prerogativa del codice Matlab, il quale utilizza tutte queste informazioni ricevute dal dispositivo per convertire gli angoli nella matrice di rotazione relativa, che verrà utilizzata per tutte le operazioni successive. Sarà l'utente stesso a scegliere poi quale sistema di riferimento utilizzare attraverso un opportuno comando della suddetta GUI, per facilitare la comprensione e l'utilizzo della piattaforma. Ulteriore supporto visivo verrà dato da grafici e *plot* tridimensionali, che serviranno a capire l'orientazione imposta per la configurazione corrente.

Il cuore dell'elaborazione è dato però da una funzione di callback attivata a intervalli regolari da parte di un timer che scandisce il campionamento dei dati.

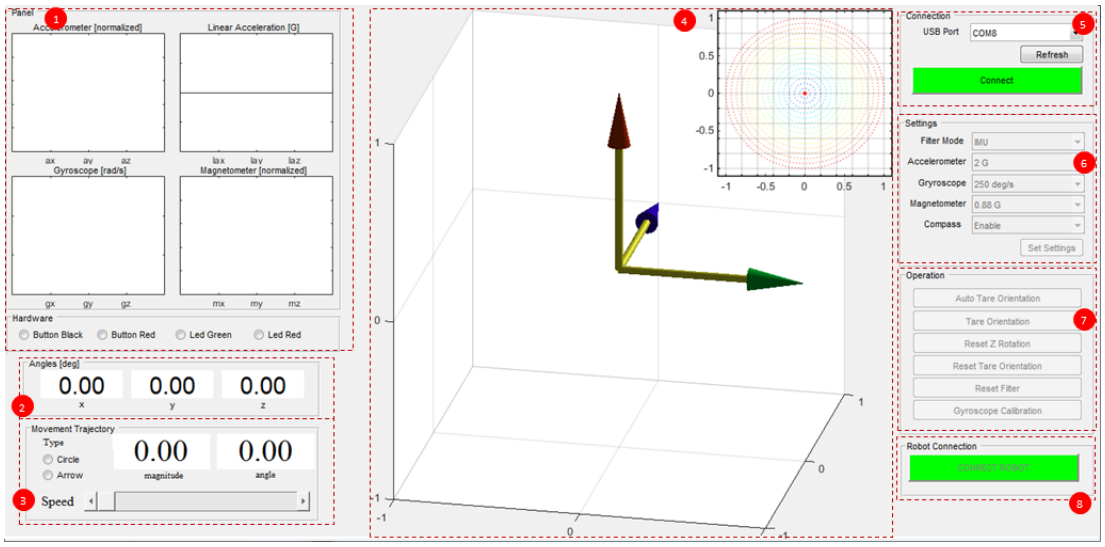


Figura 5.12: Aspetto della GUI.

Tale timer viene attivato non appena risulta connesso il dispositivo, e richiama ogni $25ms$ circa la routine principale che si occupa di prelevare i dati dalla piattaforma, eseguire le dovute elaborazioni e aggiornare la GUI con le nuove informazioni ottenute. Il passo successivo sarà quello di creare i punti da inviare al robot sulla base dei dati raccolti e, in caso di verificata connessione del robot, procedere alla spedizione del pacchetto.

Prima di passare alla descrizione dell'algoritmo di creazione vero e proprio, però, è opportuno analizzare la GUI e le metodologie per gestire la sua comunicazione con piattaforma e manipolatore. In Fig. 5.12 è visibile la GUI e le varie parti cui è suddivisa, descritte più nel dettaglio nelle sezioni seguenti.

5.6.1 Panel-Hardware

In questo pannello vengono mostrate tutte le informazioni raccolte dalla piattaforma nel ciclo corrente. In particolare, vengono rappresentati i valori normalizzati al loro fondo scala di: accelerazione (totale e senza componente gravitazionale), velocità angolare e campo magnetico. Tutti questi valori rappresentano, per semplicità, i valori rilevati dai sensori nel sistema di riferimento di base, ossia quello del sensore stesso, per cui non dipendono dal sistema di riferimento correntemente in uso.

Il pannello hardware sottostante, invece, tiene traccia dell'attivazione delle varie periferiche I/O del dispositivo: i due pulsanti (nero/rosso) e i due LED (rosso/verde).

5.6.2 Angles (RPY)

Questo pannello raccoglie le informazioni sull'orientamento corrente, espresso in funzione del sistema di riferimento scelto. Si tratta in sostanza dell'espressione in termini di angoli di Cardano dell'orientazione corrente {C}.
Data la matrice di rotazione corrente nel sistema di riferimento imposto {O}, definita come:

$$\mathbf{R}_c^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{5.2}$$

e ricordando l'espressione degli angoli di Cardano

$$\mathbf{R}_{\mathrm{RPY}}(\varphi, \vartheta, \psi) = \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix} \tag{5.3}$$

Si ricava la soluzione nella forma (per $-\pi < \vartheta < \pi$):

$$\begin{cases} \varphi = atan2(r_{21}, r_{11}) \\ \vartheta = atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \psi = atan2(r_{32}, r_{33}) \end{cases} \tag{5.4}$$

Queste variabili rappresentano rispettivamente gli angoli di *roll* (φ), *pitch* (ϑ) e *yaw* (ψ).

Poiché questi valori sono anche quelli utilizzati dal software EPSON per impostare l'orientazione, risulta molto utile avere una loro espressione in formato testuale per avere un'idea del valore che andrà spedito.

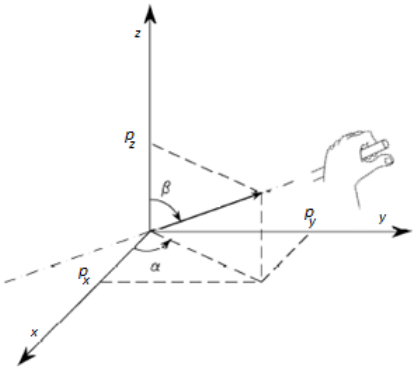


Figura 5.13: Calcolo dei parametri nel caso di traiettoria planare.

5.6.3 Movement trajectory

In questo pannello vengono raccolte le informazioni riguardanti il moto planare e la traiettoria generata. Può capitare infatti di voler muovere il robot su un piano mantenendo le stesse impostazioni di orientazione, e in tal caso la via migliore è quella di trattare la piattaforma come fosse un *joystick*. Fissato per comodità utilizzativa il suo sistema di riferimento in maniera che l'asse longitudinale del dispositivo coincida con l'asse z del sistema di riferimento del robot (in sostanza che il joystick sia rivolto verso l'alto), si considerano in questo caso le componenti dell'asse z nel riferimento in esame. In base al sistema espresso in Fig. 5.13, si considera, anziché l'intera terna, solamente il vettore orientato lungo l'asse z del dispositivo (asse longitudinale), in maniera da ricavare la sua orientazione nella base indicata. In questo modo si possono ricavare le tre componenti p_x, p_y e p_z a partire dall'espressione della matrice di rotazione corrente semplicemente risolvendo un problema di cambio di coordinate: conoscendo l'espressione del vettore nel sistema di riferimento solidale al joystick (che è coincidente con l'asse z per definizione), il calcolo dei tre parametri nel riferimento scelto è

$$\mathbf{p}^0 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \mathbf{R}_c^0 \mathbf{z} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.5}$$

Per cui a partire dalla matrice di trasformazione si ottengono i parametri semplicemente come:

$$\begin{cases} p_x = r_{13} \\ p_y = r_{23} \\ p_z = r_{33} \end{cases} \tag{5.6}$$

L'angolo β, definito come l'angolo tra l'asse z di riferimento e quello attuale, rappresenta una misura di modulo, ossia, in termini comparativi, quanto il joystick è distante dalla sua posizione di riposo. Viceversa, l'angolo α rappresenta l'orientazione nel piano, ossia la direzione che la proiezione assume. Tali angoli sono ricavabili come

$$\begin{cases} \beta = atan2(r_{33}, \sqrt{r_{13}^2 + r_{23}^2}) \\ \alpha = atan2(r_{13}, r_{23}) \end{cases} \tag{5.7}$$

Questi parametri identificano direzione, modulo e verso di un vettore nel piano, che viene utilizzato come vettore velocità effettivo: tale sarà infatti la direzione di movimento del manipolatore istante per istante. Portando il joystick in posizione verticale, ossia allineandolo verso la posizione di riposo, il modulo del vettore decrescerà e il manipolatore rimarrà fermo nella posizione precedente. Viceversa, allontanando il joystick da questa posizione, aumenterà il modulo e con esso la velocità del robot, che si muoverà lungo la direzione imposta dall'utente.

Nel pannello sono presenti due caselle di testo in cui è indicato, istante per istante, il valore del modulo (normalizzato a 1) e dell'angolo α, nonché un selettore per la rappresentazione in 2D sotto forma di punto (*Circle*) oppure di freccia (*Arrow*) nell'apposito grafico, descritto successivamente.

5.6.4 Plots

Com'è intuibile, nella sezione centrale vengono rappresentati, istante per istante, le informazioni sull'orientazione e sul movimento planare. Partendo dagli angoli RPY calcolati nell'apposita sezione, il software non fa altro che aggiornare un plot tridimensionale creando una nuova terna basata sulle coordinate correnti. Come metro di paragone, sul grafico viene sempre rappresentata una terna fissa rappresentante l'ultimo sistema di riferimento memorizzato, che risulta particolarmente

utile per avere un'idea dell'orientazione corrente.
In alto a destra è invece fissato un plot in 2D, che viene anch'esso aggiornato con le coordinate planari correnti, ossia quelle descritte in 5.8.1. Sulla base della rappresentazione scelta, viene riprodotto un punto oppure una freccia, localizzati convertendo le coordinate polari (modulo e angolo) in cartesiane.

5.6.5 Connection

La parte a destra riguarda l'interazione e la comunicazione con piattaforma e robot. In particolare questo riquadro contiene le informazioni sulla connessione/disconnessione del dispositivo. Permette di poter scegliere dalla lista le porte seriali attualmente disponibili e di effettuare una connessione tramite l'apposito pulsante. Se in una di esse viene rilevato un dispositivo, la GUI viene aggiornata a "connessa" e inizia la comunicazione tra essa e il dispositivo. Se a questo punto viene premuto nuovamente il pulsante (che stavolta mostrerà la scritta "*Disconnect*"), la GUI terminerà le sue elaborazione e lo scambio di dati con la piattaforma, rimanendo in attesa di una nuova connessione.

5.6.6 Settings

In questo pannello si trovano le impostazioni di lavoro del dispositivo, ossia quei parametri con cui normalmente i sensori e il microcontrollore processano i segnali e le istruzioni. Tramite i menu a tendina è possibile selezionare ogni valore per l'opportuno parametro, e il pulsante "*Set Settings*" invia al dispositivo le caratteristiche impostate. In totale ci sono 5 opzioni da poter scorrere:

- *Filter Mode*: permette di selezionare il tipo di filtro utilizzato dal microcontrollore per l'aggiornamento dello stato. Le opzioni selezionabili sono: *No filter*, *Kalman Filter*, *Complementary filter*, *Gradient descent*.
- *Accelerometer Range*: permette di scegliere il fondo scala da utilizzare per l'accelerometro. È possibile scegliere tra *2g*, *4g* e *8g*.
- *Gyroscope Range*: permette di scegliere il fondo scala da utilizzare per il giroscopio. È possibile scegliere tra *250deg/s*, *500deg/s* e *2000deg/s*.

- *Magnetometer Range*: permette di scegliere il fondo scala da utilizzare per il magnetometro. È possibile scegliere tra *0.88G*, *1.3G*, *1.9G*, *2.5G*, *4G*, *4.7G*, *5.6G* e *8.1G*.

- *Compass*: consente di attivare o disattivare il magnetometro. La sua attivazione migliora la precisione di calcolo dell'orientamento, e consente inoltre di definire una terna univoca di partenza.

5.6.7 Operation

Questo pannello contiene le informazioni essenziali per la calibrazione e la corretta impostazione del dispositivo. Permette di agire su 6 pulsanti:

- *Auto Tare Orientation*: consente di trovare in maniera automatica l'impostazione di taratura ottimale del dispositivo. A partire dal sistema di riferimento solidale al dispositivo stesso, questa opzione consente di riconoscere automaticamente se il suo asse longitudinale è perpendicolare al suolo, ossia se la piattaforma è posizionata "in piedi". Tale condizione viene verificata dall'analisi delle tre componenti dell'accelerazione misurate dallo strumento, e non appena viene riconosciuta la configurazione corrispondente, con una certa tolleranza, tale configurazione diventa il sistema di riferimento corrente e viene memorizzato sotto forma di matrice di rotazione. Più nello specifico, questo comando equivale ad un *Tare Orientation* e ad un *Reset Z rotation* eseguiti quando il dispositivo viene rilevato "in piedi".

- *Tare Orientation*: consente di tarare il dispositivo nella configurazione corrente. Questo comando permette di memorizzare la matrice di rotazione della piattaforma nella posizione corrente, ossia quella prodotta dal sensore secondo il suo riferimento. Tale matrice viene utilizzata per fissare il nuovo sistema di riferimento nella posizione assunta dal dispositivo al momento della pressione del pulsante, permettendo di fissare in maniera arbitraria la terna d'origine a seconda della posizione più comoda all'utilizzatore.

- *Reset Z rotation*: permette di tarare la rotazione attorno all'asse z secondo la corrente configurazione. Questo comando rileva l'angolo di *yaw* della

corrente posizione e lo memorizza sotto forma di matrice di rotazione (come rotazione attorno all'asse x). Tale matrice verrà utilizzata per togliere dalla configurazione attuale l'angolo di *yaw*, in maniera che la taratura di un diverso sistema di riferimento non influisca sulla gestualità e sull'utilizzo della piattaforma. È noto infatti che un generico riferimento, espresso in coordinate RPY, risulta dato da

$$\mathbf{R}(\varphi, \vartheta, \psi) = \mathbf{R}_{\mathrm{z}}(\varphi)\mathbf{R}_{\mathrm{y}}(\vartheta)\mathbf{R}_{\mathrm{x}}(\psi) \tag{5.8}$$

Può capitare di volere tarare il dispositivo in maniera da considerare la configurazione corrente a meno dell'angolo di imbardata (*yaw*). Per fare ciò, è sufficiente memorizzare tale angolo e utilizzarlo per generare la matrice $\mathbf{R}_{\mathrm{x}}(-\psi) = \mathbf{R}_{\mathrm{x}}{}^{T}(\psi)$. La postmoltiplicazione di $\mathbf{R}(\varphi, \vartheta, \psi)$ per $\mathbf{R}_{\mathrm{x}}(-\psi)$ contribuirà ad eliminare il termine di rotazione attorno ad x, in quanto l'ultima moltiplicazione è il prodotto della matrice di *yaw* per la sua inversa.

- *Reset Tare Orientation*: consente di cancellare le correnti impostazioni di taratura, riportando lo stato del dispositivo a quello assunto appena dopo l'avvenuta connessione.

- *Reset Filter*: permette di cancellare le correnti impostazioni di stato del filtro utilizzato dal microcontrollore.

- *Gyroscope Calibration*: permette di calibrare il giroscopio al fine di limitare eventuali offset di misura.

5.6.8 Robot Connection

Questo riquadro contiene le informazioni per la connessione/disconnessione del robot. In particolare presenta un solo pulsante, "*CONNECT ROBOT*", alla cui pressione predispone la GUI a caricare la classe robot corrispondente e ad effettuare la connessione TCP/IP con il controller robot. Perché questa operazione vada a buon fine, è necessario che il programma EPSON sia già stato avviato e che rimanga in attesa di connessioni, altrimenti verrà visualizzato un messaggio di errore. Una volta stabilita la connessione, la GUI comincerà ad inviare al

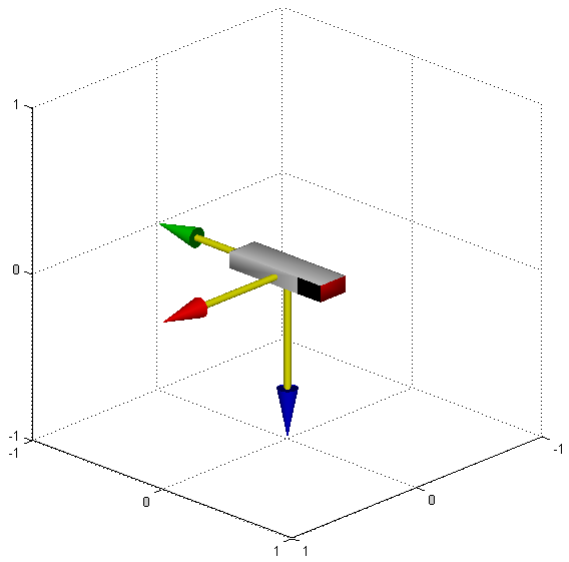


Figura 5.14: Rappresentazione del dispositivo nel sistema IMU iniziale.

robot tutti i dati elaborati, fintanto che non viene premuto nuovamente il pulsante di connessione, che questa volta riporterà la dicitura "*DISCONNECT ROBOT*".

5.7 Calibrazione del joystick

Essendo che la movimentazione del robot avviene tramite specifica dei punti di passaggio, per cui l'oggetto fondamentale della comunicazione Matlab-EPSON sono dei set di sestuple che identificano tali punti, il programma Matlab dovrà quindi procedere alla loro creazione a partire dai dati ricevuti dal joystick.

I dati ricevuti dai sensori del joystick (IMU) sono espressi sotto forma di matrice di rotazione rispetto ad un opportuno sistema di riferimento assoluto, detto *ambiente*, corrispondente al joystick in posizione orizzontale. Tale sistema è mostrato in Fig. 5.14, dove le aree rosse e nere corrispondono rispettivamente al pulsante rosso e nero del joystick. Si è deciso di mantenere fisso dal lato IMU tale riferimento dal momento che il codice per la gestione dei sensori è stato efficacemente utilizzato e testato nel progetto descritto nel Cap. **??**.

Tuttavia, l'ausilio di altre terne di riferimento permette un utilizzo più intuitivo

del joystick quando utilizzato per la movimentazione di un robot. Per questo motivo si è implementata una procedura di taratura per la scelta di una o più terne di riferimento, eventualmente arbitrarie. Tali funzioni sono richiamabili da pulsanti presenti nel panel *Operation* della GUI in Matlab. In questo senso si possono definire complessivamente i seguenti sistemi di riferimento:

$\{a\}$ sistema di riferimento *ambiente* fisso rispetto al quale si ricevono tutte le coordinate da parte dei sensori;

$\{j\}$ sistema di riferimento corrente del *joystick* di cui l'IMU misura la rotazione operata rispetto ad $\{a\}$ (se la rotazione $\mathbf{R_j^a} \equiv \mathbf{I}$, le due terne coincidono);

$\{0\}$ sistema di riferimento fisso per l'*azzeramento* nell'orientamento calibrato con il pulsante *Tare Orientation* (es. a joystick verticale);

$\{0'\}$ sistema di riferimento fisso per l'*azzeramento* della rotazione attorno a z nell'orientamento calibrato con il pulsante *Reset Z Rotation* (es. a joystick verticale leggermente ruotato in avanti rispetto alla presa);

$\{b\}$ sistema di riferimento disposto come la terna della *base* del robot a joystick in azzeramento;

$\{i\}$ sistema di riferimento corrente del *joystick* ma con assi disposti secondo ordinamento del sistema $\{b\}$ (se la rotazione $\mathbf{R_i^b} \equiv \mathbf{I}$, le due terne coincidono);

$\{e\}$ sistema di riferimento disposto come la terna dell'*end-effector* del robot.

L'idea è quindi definire un sistema da utilizzare come riferimento in generale per tutti i panel della GUI in MatLab ed in particolar modo per la creazione dei punti di movimento da inviare al controllore del robot. Con riferimento alla Fig. 5.15 si definisce in giallo il sistema di riferimento ambiente $\{a\}$ del joystick, mostrato in figura come corrispondente al dispositivo nell'orientamento orizzontale, e in nero una terna corrispondente al corrente orientamento del joystick di tipo $\{j\}$ oppure $\{i\}$. Per i singoli assi x, y, z si utilizza la convenzione RGB (red, green, blue), diversa dalla convenzione inusuale GBR del software EPSON.

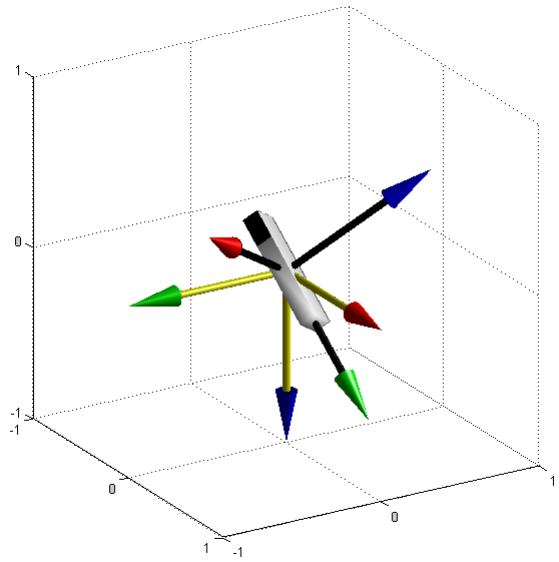


Figura 5.15: Rotazione generica del dispositivo rispetto al sistema IMU iniziale ($\psi = 135°$, $\vartheta = 30°$, $\varphi = 60°$).

5.7.1 Taratura della terna di azzeramento

Data quindi una variazione di postura del dispositivo, come quella mostrata in Fig. 5.15 definibile tramite tre rotazioni attorno ad assi correnti RPY (roll φ in z, pitch ϑ in y, yaw ψ in x) di angoli rispettivamente pari a $\varphi = 60°$, $\vartheta = 30°$, $\psi = 135°$, chiamiamo $\mathbf{R}_j^a$ la trasformazione corrispondente data da:

$$\mathbf{R}_j^a = \mathbf{R_z}(\varphi)\mathbf{R_y}(\vartheta)\mathbf{R_x}(\psi) \tag{5.9}$$

Scopo della taratura è definire una terna $\{0\}$, memorizzarla ed utilizzarla come riferimento. Per far ciò si posiziona il joystick ad una determinata orientazione e si preme sul pulsante *Tare Orientation*. In questo modo si memorizza la matrice $\mathbf{R}_0^a = \mathbf{R}_j^a$ per utilizzarla come riferimento per le trasformazioni successive che possiamo chiamare $\mathbf{R}_j^0$. Tuttavia il sensore fornirà la trasformazione sempre in termini di $\mathbf{R}_j^a$, per cui:

$$\mathbf{R}_j^0 = (\mathbf{R}_0^a)^T \mathbf{R}_j^a \tag{5.10}$$

Una volta impostata tale matrice, questa operazione viene quindi eseguita per ogni dato ricevuto.

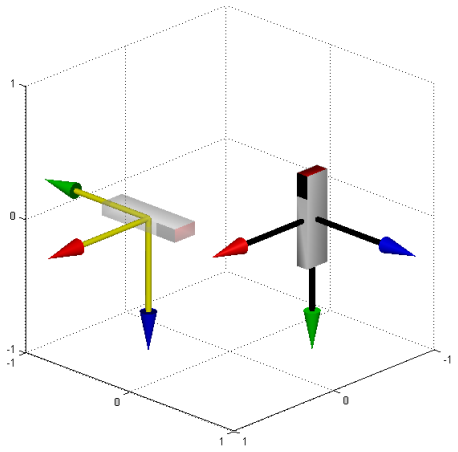


Figura 5.16: Sistema IMU iniziale $\{a\}$ e sistema IMU $\{j\} \equiv \{0\}$ nella posizione verticale del joystick.

Tra le diverse scelte arbitrarie una postura è in particolare conveniente per il controllo del manipolatore, cioè disporre il joystick, per l'azzeramento, con l'asse longitudinale verticale e l'impugnatura delle dita in corrispondenza del pulsante nero. In Fig. 5.16 è mostrata la postura del joystick.

5.7.2 Taratura della rotazione z

La taratura della rotazione z tramite la pressione del pulsante *Reset Z Rotation* è stata implementata per correggere la rotazione attorno all'asse z della terna $\{0\}$. Ciò può essere utile dal momento che qui si può concentrare l'attenzione solo su questa rotazione perché solo essa verrà effettivamente memorizzata. Si può pensare quindi di trovare la posizione più confortevole, come ad es.a polso a riposo che con joystick verticale significa leggera rotazione in avanti (positiva in z).

La nuova terna $\{0'\}$ sarà utilizzata come riferimento per tutte le rotazioni successive. Analogamente all'Eq. 5.10, si può scrivere allora che:

$$\mathbf{R}_j^{0'} = (\mathbf{R}_0^a \mathbf{R}_{0'}^0)^T \mathbf{R}_j^a \tag{5.11}$$

La matrice $\mathbf{R}_{0'}^0$ si ricava come:

$$\mathbf{R}_{0'}^0 = (\mathbf{R}_0^a)^T \mathbf{R}_{0'}^{a^*} \tag{5.12}$$

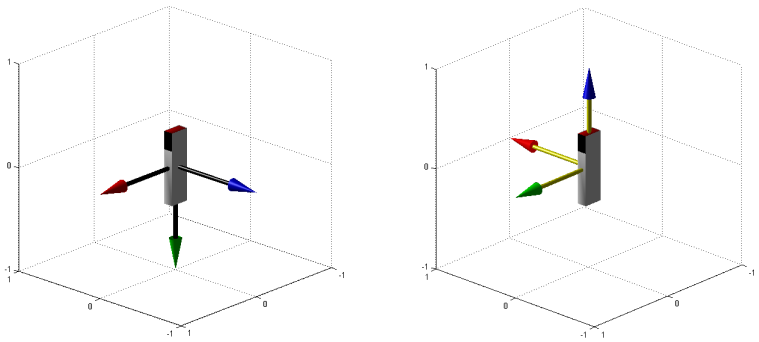


Figura 5.17: (a) Terna di azzeramento che utilizza il sistema di riferimento IMU solidale al joystick ; (b) sistema di riferimento desiderato (corrispondente a quello della base robot) alla postura attuale.

dove

$$\mathbf{R}_{0'}^{a^*} = \mathbf{R_z}(\varphi)\mathbf{R_y}(\vartheta)\mathbf{R_x}(\psi) = \mathbf{R_z}(\varphi) \tag{5.13}$$

L'angolo φ si ricava dalla cinematica inversa della matrice $\mathbf{R}_{0'}^{a^*}$ vista come matrice di Cardano, nello specifico:

$$\varphi = \text{atan2}(-r_{1,2}, r_{1,1}) \tag{5.14}$$

5.7.3 Impostazione della terna base

L'utilizzo degli assi del sistema $\{j\}$, in termini di orientamento rispetto al dispositivo stesso, non è detto che siano convenienti nel nostro caso. Per questo motivo è stata implementata la possibilità di operare una trasformazione di assi mantenendo inalterata la postura del joystick.

Tra tutte le arbitrarie possibilità, conveniente è la scelta di utilizzare una terna identica a quella della base robot quando il joystick si trova in configurazione di azzeramento verticale. La Fig. 5.17 mostra la terna solidale al joystick secondo IMU e quella desiderata. La scelta è ottimale nell'ottica di inviare punti al robot da raggiungere con il centro utensile (TCP), infatti punti espressi rispetto ad una terna joystick disposta come la base potrebbero essere inviati come punti robot espressi nel sistema base del robot.

La rotazione che porta gli assi del sistema $\{0'\}$ a sovrapporsi a quelli della terna $\{b\}$ è allora data da:

$$\mathbf{R}_b^{0'} = \mathbf{R_z}(-90°)\mathbf{R_y}(90°)\mathbf{R_x}(0°) \tag{5.15}$$

Una rotazione $\mathbf{R}_i^b$ può essere allora calcolata considerando che tale rotazione avviene rispetto alla terna $\{b\}$, definita a sua volta dalla rotazione $\mathbf{R}_b^{0'}$. Tuttavia quest'ultima rotazione non coincide con una rotazione del joystick e di conseguenza va sottratta a $\mathbf{R}_i^b$. Si può scrivere perciò:

$$\mathbf{R}_0^a\mathbf{R}_{0'}^0\mathbf{R}_b^{0'}\mathbf{R}_i^b\mathbf{R}_{0'}^b = \mathbf{R}_j^a \tag{5.16}$$

da cui si ottiene

$$\mathbf{R}_i^b = (\mathbf{R}_0^a\mathbf{R}_{0'}^0\mathbf{R}_b^{0'})^T\mathbf{R}_j^a\mathbf{R}_b^{0'} \tag{5.17}$$

La terna $\{b\}$ viene quindi presa come riferimento per tutti i dati successivi. Ciò significa che punti ricavati dal movimento del joystick saranno valutati rispetto a questa terna e potranno essere spediti al robot ad es. come punti di centro utensile da raggiungere definiti nel sistema base del robot.

Si fa notare che si è utilizzata una matrice costante $\mathbf{R}_b^{0'}$ definita rispetto al sistema $\{0'\}$ in luogo ad es. di una matrice $\mathbf{R}_b^a$. Ciò è dovuto al fatto che, quando il magnetometro è disabilitato come nel caso di utilizzo in ambito industriale, la rotazione intorno a z letta all'accensione del dispositivo dalla matrice $\mathbf{R}_j^a$ non è assoluta (come con il magnetometro, es. x che punta al nord magnetico). Di conseguenza servirebbe un riposizionamento manuale riferito a qualcosa di assoluto per poter impiegare tale matrice.

5.7.4 Impostazione della terna end-effector

Un'altra terna che è comoda durante l'utilizzo del joystick in alcune modalità implementate è la terna utensile del robot. Si è quindi implementata la possibilità di inviare dati che il robot deve interpretare come terne a cui sovrapporre la terna tool.

Si considera il robot in posizione di riposo. come mostrato in Fig. 5.18. La rotazione che permette di passare dalla terna base del robot alla sua terna utensile

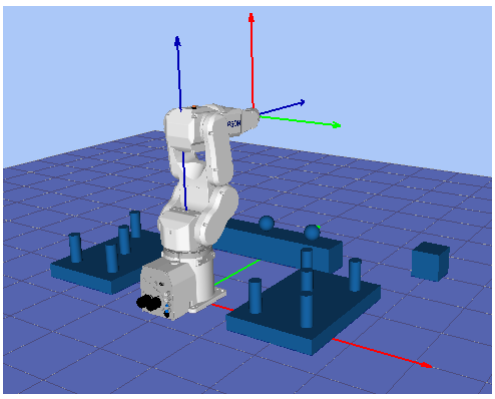


Figura 5.18: Relazione tra end-effector del robot in posizione di riposo e terna base del robot.

è data da:

$$\mathbf{R}_e^b = \mathbf{R}_z(180°)\mathbf{R}_y(-90°)\mathbf{R}_x(90°) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.18}$$

Siccome per un robot più tipicamente la definizione della terna da raggiungere con la sua terna utensile è definita rispetto alla terna della base robot e non rispetto ad una terna utensile di riferimento, ne consegue la necessità di definire la rotazione da inviare rispetto alla terna base del joystick, seguita dalla rotazione costante $\mathbf{R}_e^b$. Ciò equivale a dire che quando il joystick si trova in posizione di azzeramento la matrice che si desidera inviare non è l'identità bensì proprio $\mathbf{R}_e^b$. La matrice da inviare vale quindi:

$$\mathbf{R}_i^b\mathbf{R}_e^b = (\mathbf{R}_0^a\mathbf{R}_{0'}^0\mathbf{R}_b^{0'})^T\mathbf{R}_j^a\mathbf{R}_b^{0'}\mathbf{R}_e^b \tag{5.19}$$

5.8 Movimento robot tramite joystick

L'ultima operazione dell'interfaccia è la procedura di creazione dei punti da inviare al robot. Sfruttando la GUI per calibrare e gestire correttamente il dispositivo, è possibile tramite la pressione di opportuni pulsanti eseguire delle movimentazioni di traslazione o di rotazione. Il timer di base della GUI richiama a intervalli regolari la funzione principale, che si occupa di interrogare il dispositivo sul suo stato corrente e riceve da esso la matrice di rotazione $\mathbf{R}_j^a$ e i dati inerenti

ad accelerometro, giroscopio e magnetometro (se abilitato). Sulla base delle caratteristiche di configurazione corrente, viene generata la matrice $\mathbf{R}_{fin}$, che verrà utilizzata per creare i punti da inviare poi al programma EPSON. Essa vale $\mathbf{R}_i^b$ o $\mathbf{R}_i^b\mathbf{R}_e^b$ a seconda della modalità di utilizzo desiderata. In particolare, la pressione del pulsante *nero* abilita la funzione di movimento planare legata alla prima matrice, mentre la pressione del pulsante *rosso* abilita il controllo di orientazione dell'end-effector, basato sulla seconda matrice. I successivi paragrafi illustreranno più nel dettaglio ciascuno dei due casi, soffermandosi sulle tecniche di utilizzo e di interfacciamento diretto con il programma EPSON.

5.8.1 Movimento planare

Come già anticipato durante la descrizione della GUI (5.6), c'è la possibilità di utilizzare il dispositivo come fosse un joystick, al fine di poter muovere il robot su larga scala creando una traiettoria sul piano $x-y$. La modalità di utilizzo è molto semplice: partendo dal dispositivo tarato in posizione di riposo, ossia con asse longitudinale (considerato come asse z) perpendicolare al suolo, viene valutata la proiezione dell'asse z sul piano parallelo al suolo, per ogni rotazione assunta.
Il meccanismo di funzionamento è quello descritto nel paragrafo "*Movement Trajectory*", brevemente ripreso nelle righe successive. La matrice utilizzata è quella ricavata dall'elaborazione, ossia:

$$\mathbf{R}_{fin} = (\mathbf{R}_0^a\mathbf{R}_{0'}^0\mathbf{R}_b^{0'})^T\mathbf{R}_j^a\mathbf{R}_b^{0'} \tag{5.20}$$

In questo sistema di coordinate, dunque, la proiezione dell'asse z sul piano è effettuabile, analogamente a quanto già detto, come:

$$\mathbf{p}^0 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \mathbf{R}_{fin} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.21}$$

a cui si perviene allo stesso risultato espresso in 5.8.1, ossia

$$\begin{cases} p_x = r_{fin_{13}} \\ p_y = r_{fin_{23}} \\ p_z = r_{fin_{33}} \end{cases} \tag{5.22}$$

I punti che giacciono sul piano $x - y$ sono dunque p_x e p_y, e rappresentano la direzione del movimento.
Una cosa importante da sottolineare è che tali punti non indicano le coordinate in cui si dovrà spostare direttamente il robot: un approccio del genere significherebbe dover mappare l'intero piano di lavoro del manipolatore nel piano di rotazione del dispositivo. A rigor di logica ciò non rappresenterebbe un problema, dal momento che ad ogni punto del *workspace* del robot viene fatto corrispondere un punto sul piano di utilizzo del dispositivo, tuttavia ciò significherebbe mappare un'area di lavoro grande (robot) in una molto più piccola (joystick). È questa la limitazione maggiore a questo tipo di approccio: una piccola variazione, dovuta ad esempio a tremolii della mano, produrrebbe infatti movimenti macroscopici del manipolatore, che si ritroverebbe a oscillare fastidiosamente attorno ad un punto.
Ecco perché il criterio adottato non è quello della rappresentazione di posizione, quanto di velocità: la proiezione dell'asse z del dispositivo sul piano rappresenta una misura di direzione, modulo e verso della velocità puntuale assunta in quel momento. Questo consente un controllo migliore, più semplice ed intuitivo: portare il joystick nella posizione di riposo equivale, infatti, a mantenere il manipolatore fisso nella posizione corrente.

L'identificazione del vettore velocità avviene dunque per semplice estrapolazione dei valori di $r_{fin_{13}}$ e $r_{fin_{23}}$: tali punti vengono utilizzati per creare una rappresentazione sul piano in un grafico separato, come mostrato in Fig. 5.19.

Ricordando quanto già espresso in 5.8.1, sono possibili due raffigurazioni tipiche per la rappresentazione 2D: tramite punto e tramite freccia. Entrambi questi metodi sono equivalenti, dal momento che identificano la stessa grandezza, tuttavia la freccia permette un'associazione migliore con il concetto di velocità puntuale, inoltre essa varia il suo spessore a seconda dell'aumento del suo modulo, come visibile in Fig. 5.20.

Le coordinate così ricavate vengono poi utilizzate per generare i set di punti da inviare al robot. Tramite l'opportuno slider è possibile eseguire una regolazione della velocità effettiva da imporre: la proiezione nel piano, infatti, genera un vettore che però è normalizzato, occorre dunque moltiplicarlo per un opportuno scalare al fine di regolarne l'effettivo valore. Questo perché può capitare di volere

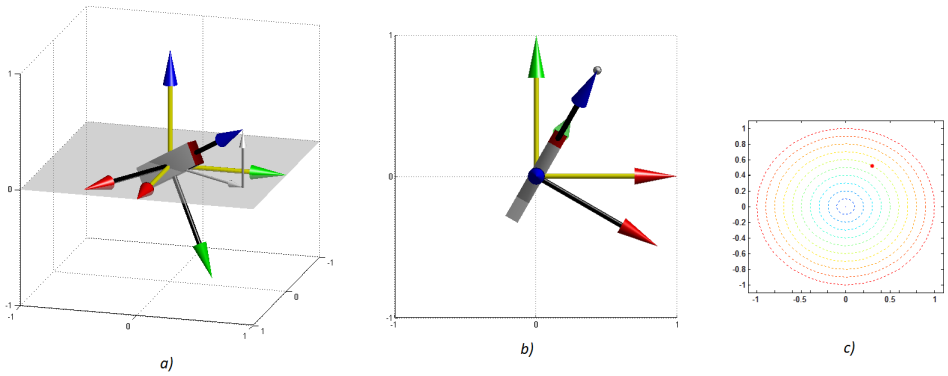


Figura 5.19: Esempio di utilizzo della movimentazione planare: a) orientazione generica con rappresentazione delle proiezioni, b) vista del dispositivo dall'alto e c) rappresentazione sul piano 2D.

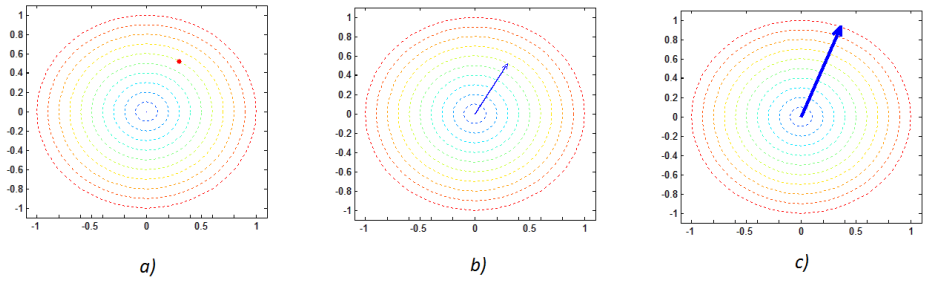


Figura 5.20: Rappresentazione delle due diverse icone per il piano 2D: a) punto, b) freccia con le stesse coordinate, c) freccia con modulo maggiore.

controllare il robot con estrema precisione in un intorno molto ristretto, ed è perciò più conveniente avere un'area di movimento ampia a cui però corrisponde una velocità massima molto bassa. Viceversa, quando si vuole spostare il robot in un'altra locazione molto lontana, è impraticabile dover lavorare con velocità massime così ridotte, ed in quei casi occorre poter associare alla stessa area di lavoro una velocità massimo più alta, così da poter raggiungere un estremo opposto in poco tempo, dove magari si potrà reimpostare lo slider al valore minimo per garantire una regolazione fine.

Il modulo del vettore proiettato nel piano $x - y$, con modulo unitario, andrà

quindi moltiplicato per il valore imposto dallo slider, completando così il vettore velocità finale che sarà utilizzato come dato di ingresso per la funzione di creazione punti. Tale funzione, denominata *CreatePathTrajectory*, viene chiamata qualora sia rilevata la pressione del pulsante *nero*, e si occupa sia della creazione dei dati, sia del controllo della loro correttezza.

Ad ogni movimento successivo, pertanto, l'ultimo punto del percorso precedente diventerà il primo del percorso attuale, e così via aggiornando di volta in volta. Il primo set di coordinate, ossia quello da cui parte il manipolatore una volta connesso alla GUI, viene ottenuto tramite interrogazione diretta, con l'apposito pacchetto dall'header "*getpos*", il quale richiama il comando *CurPos* del software EPSON che restituisce la posizione corrente nelle coordinate dello spazio operativo.

A partire da tale punto, dunque, la funzione *CreatePathTrajectory* genera il successivo tramite semplice somma vettoriale di coordinate: chiamato $\mathbf{p_0}$ il punto di partenza, e dato il vettore velocità $\mathbf{v}$ che rappresenta il vettore proiezione nel piano scalato per il valore dato dallo slider, il punto successivo $\mathbf{p_1}$ viene ricavato tramite semplice applicazione della legge oraria nel dominio discreto:

$$\mathbf{p_1} = \mathbf{p_0} + T_c\mathbf{v} \tag{5.23}$$

dove T_c è il tempo di campionamento. Generalmente tale costante viene inclusa nella moltiplicazione per lo scalare ottenuto dallo slider, in maniera da inglobare tutto in un unico parametro. Chiamando quindi $\mathbf{d} = T_c\mathbf{v}$ il valore finale ottenuto, chiamato *displacement*, i nuovi punti si ricavano come:

$$\begin{cases} p_{1x} = p_{0x} + d_x \\ p_{1y} = p_{0y} + d_y \\ p_{1z} = p_{0z} \end{cases} \tag{5.24}$$

Da notare che, siccome il movimento viene eseguito sul piano $x - y$, le coordinate z non vengono modificate.

Dopo questo passaggio, si possiedono dunque due punti: quello di partenza e il primo creato. Per poter costruire una spline è necessario un minimo di 3 punti, per cui le strade percorribili sono due:

1. Si attende il successivo campionamento di un nuovo valore di velocità, con cui, dopo l'elaborazione necessaria, si genera un nuovo punto. A questo

punto si dispone di 3 set totali, che vengono inviati al programma EPSON per eseguire la movimentazione relativa. Il terzo punto viene salvato nella locazione di partenza e verrà usato come punto di inizio $\mathbf{p_0}$ per il ciclo successivo.

2. Si interpola in maniera lineare per ottenere un terzo punto "fittizio" a metà strada tra i punti $\mathbf{p_0}$ e $\mathbf{p_1}$. Con 3 set è ora possibile inviare le coordinate al robot per eseguire la movimentazione. Il punto $\mathbf{p_1}$ viene salvato nella locazione di partenza e verrà usato come punto di inizio $\mathbf{p_0}$ per il ciclo successivo.

Questi due approcci rappresentano due estremi diversi che minimizzano ciascuno due parametri, uno a discapito dell'altro: il primo approccio è volto a salvaguardare la "veridicità" dei dati, fornendo dei valori ottenuti campionando posizioni reali; di contro, con questo meccanismo aumenta la latenza tra movimento utente e movimento robot, in quanto sono richiesti due cicli di campionamento prima dell'invio dei dati. Il secondo approccio, di contro, riduce al minimo la latenza utente-robot, inviando dati ad ogni periodo di campionamento; vi è però l'introduzione di punti interpolati, quindi non misurati ma imposti.

(1) Se si vuole procedere con il primo approccio, al ciclo successivo si otterrà un nuovo valore di velocità $\mathbf{v}$, da cui discenderà la corrispondente spaziatura $\mathbf{d}$, e il terzo punto verrà creato come

$$\begin{cases} p_{2x} = p_{1x} + d_x \\ p_{2y} = p_{1y} + d_y \\ p_{2z} = p_{1z} \end{cases} \tag{5.25}$$

Dopo aver inviato i tre set $\mathbf{p_0}$, $\mathbf{p_1}$ e $\mathbf{p_2}$, si sovrascrive la locazione $\mathbf{p_0}$ (punto di partenza) con il valore $\mathbf{p_2}$ e si ripete il ciclo al successivo campionamento.

(2) Se invece si intende proseguire con il secondo metodo, occorre fare una distinzione di notazioni: poiché in questo caso il nuovo punto non viene accodato dopo $\mathbf{p_1}$ ma rappresenta il punto medio del segmento $\overline{\mathbf{p_0 p_1}}$, il terzo punto verrà chiamato $\mathbf{p_m}$ in vece di $\mathbf{p_2}$ per non generare confusione nelle notazioni.

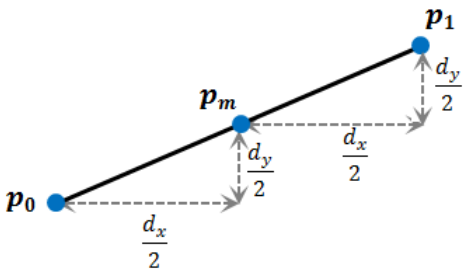


Figura 5.21: Rapporto delle distanze tra i punti nel caso di interpolazione del punto medio.

Pertanto le sue coordinate risulteranno:

$$\begin{cases} p_{mx} = p_{0x} + \frac{d_x}{2} \\ p_{my} = p_{0y} + \frac{d_y}{2} \\ p_{mz} = p_{0z} \end{cases} \tag{5.26}$$

Dopo l'invio seguirà anche in questo caso la sovrascrittura del punto $\mathbf{p_0}$ con il valore contenuto in $\mathbf{p_1}$ e il processo ripartirà da capo al successivo innesco del timer.

La scelta di quale delle due strategie adottare è legata principalmente all'ambito di utilizzo: quando i punti sono molto distanti tra loro, la procedura di interpolazione introduce delle variazioni significative. Quando invece, come nel caso in esame, i punti vengono campionati a frequenza elevata e sono pertanto molto vicini l'uno all'altro, l'interpolazione introdotta non causa variazioni visibili. Pertanto in questi casi è preferibile adottare tale strategia al fine di minimizzare il più possibile la latenza di controllo.

Per quanto riguarda l'impostazione della velocità effettiva da usare per il programma EPSON, la procedura per il calcolo è quella espressa in eq. 5.27. Nel caso la tecnica utilizzata sia basata su interpolazione del punto medio, è possibile semplificare la formula come segue:

$$v = A_f \frac{\sqrt{d_x^2 + d_y^2}}{T_c} \tag{5.27}$$

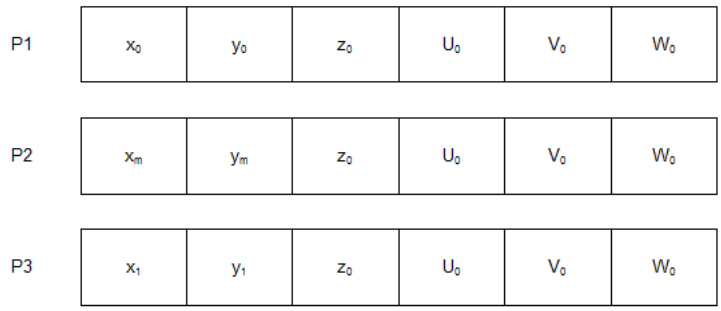


Figura 5.22: Struttura delle coordinate dei punti nel pacchetto inviato al robot in modalità movimento planare, nel caso di interpolazione del punto medio.

Questo si può vedere anche per via grafica in Fig. 5.21: poiché il punto $\mathbf{p_m}$ si trova esattamente a metà tra $\mathbf{p_0}$ e $\mathbf{p_1}$, la formula calcolo della velocità (che in 2 dimensioni impone il rapporto incrementale tra ascissa e ordinata) si traduce nel calcolo della lunghezza dei segmenti $\overline{\mathbf{p_0 p_m}}$ $\overline{\mathbf{p_m p_1}}$, che equivale alla lunghezza del segmento $\overline{\mathbf{p_0 p_1}}$, ricavabile come

$$|\overline{\mathbf{p_0 p_1}}| = \sqrt{d_x^2 + d_y^2} \tag{5.28}$$

Il parametro A_f, come già detto, va settato e regolato in maniera da ottenere un movimento il più fluido possibile.

Per quanto riguarda le orientazioni, in entrambi i casi, non è stato specificato nulla in quanto la movimentazione planare riguarda solo variazioni di spazio in due coordinate, ossia x e y, e tanto la terza coordinata spaziale z quante le tre coordinate di orientamento φ, ϑ e ψ non vengono modificate: il procedimento di creazione è dunque un processo ad incremento relativo, non assoluto, che interessa solo i valori di x ed y. Il set di punti da inviare, nel caso di interpolazione del punto medio, sarà quindi composto nella maniera indicata in Fig. 5.22. Mentre il pacchetto complessivo da inviare, comprendendo tutte le informazioni raccolte, risulta costituito come in Fig. 5.29.

A titolo di esempio, in Fig. 5.24 viene riportato un possibile utilizzo del dispositivo per movimentazione planare: nella riga *a)*, il robot viene mantenuto fisso nell'ultima posizione raggiunta. In *b)* viene imposta una velocità inclinando

1	2	3	4	5	6	7	8			9
Header	**World**	**Speed**	**Accel**	**Fine**	**Mode**	**Npts**	**Via Points**			**Checksum**
240	1	v	*variable*	0.1	1	3	P_1	P_2	P_3	*variable*

Figura 5.23: Struttura del pacchetto inviato al robot in modalità movimento planare, nel caso di interpolazione del punto medio.

il joystick in alto a sinistra rispetto al riferimento corrente: il risultato è un movimento del robot in tale direzione (45° rispetto all'asse y) che continuerà il suo moto fintanto che il dispositivo verrà mantenuto in tale direzione. In *c)*, invece, l'orientamento del dispositivo viene bruscamente variato e portato quasi completamente verso destra. Il risultato sarà una improvvisa variazione del movimento del robot, che ora tenderà a muoversi lungo il verso positivo delle x secondo il suo riferimento. Da notare che le componenti di orientazione non vengono modificate durante l'intera operazione, e nemmeno l'asse z dal momento che il moto del robot si mantiene lungo il piano di partenza. È interessante poi notare che, mantenendo la stessa direzione del joystick ma diminuendo la pendenza di inclinazione, il robot si muoverà sempre nella stessa direzione, ma in maniera più lenta.

5.8.2 Orientamento dell'organo terminale

Il movimento planare permette movimenti molti rapidi e intuitivi utilizzando la piattaforma come fosse un joystick. Tuttavia permette traiettorie che giacciono solo sul piano, e per di più non consente di modificare l'orientazione dell'end-effector. La modalità di *traiettoria in end-effector*, invece, consente di ruotare la terna utensile in maniera solidale all'orientazione assunta dal dispositivo, dato un punto fisso. Il punto di partenza è la matrice di rotazione finale ricavata in Eq. 5.19, disponibile ad ogni istante di campionamento:

$$\mathbf{R}_{fin} = \mathbf{R}_i^b \mathbf{R}_e^b = (\mathbf{R}_0^a \mathbf{R}_{0'}^0 \mathbf{R}_b^{0'})^T \mathbf{R}_j^a \mathbf{R}_b^{0'} \mathbf{R}_e^b \tag{5.29}$$

Da notare che in questo caso tale matrice, a differenza di quanto espresso precedentemente in 5.8.1, contiene anche il termine $\mathbf{R}_e^b$ che permette di passare dalle coordinate di base del robot a quelle solidali all'end-effector in posizione di riposo.

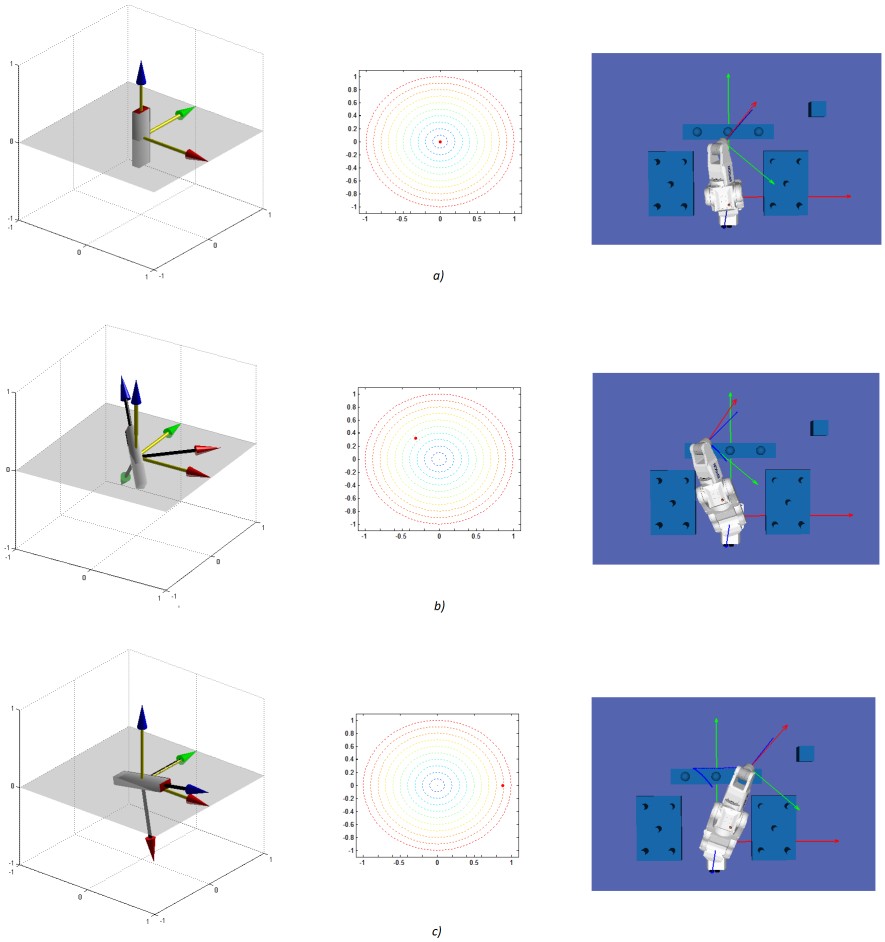


Figura 5.24: Esempio di movimento planare: a) posizione di stazionamento, b) imposizione del moto iniziale, c) variazione di direzione.

In questa modalità, la matrice $\mathbf{R}_{fin}$ contiene già tutte le informazioni necessarie: è sufficiente eseguire l'inversione di Cardano per ricavare i valori degli angoli RPY da inviare direttamente al software EPSON:

$$\begin{cases} \varphi = atan2(r_{fin_{21}}, r_{fin_{11}}) \\ \vartheta = atan2(-r_{fin_{31}}, \sqrt{r_{fin_{32}}^2 + r_{fin_{33}}^2}) \\ \psi = atan2(r_{fin_{32}}, r_{fin_{33}}) \end{cases} \tag{5.30}$$

Tali valori verranno utilizzati per formare direttamente le coordinate del punto da spedire. Poiché varia solo l'orientazione, la posizione spaziale rimane invariata e uguale all'ultima memorizzata, mentre le coordinate di orientazione vengono sostituite con quelle ricavate dalla eq. 5.30. Gli angoli così ricavati vengono passati

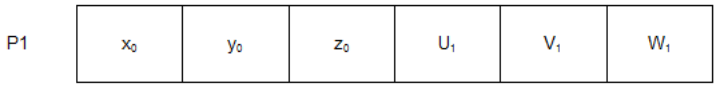


Figura 5.25: Struttura delle coordinate dei punti nel pacchetto inviato al robot in modalità end-effector orientation.

alla funzione *CreateOrientationPath*, la quale genera il pacchetto da inviare con i valori appena ricavati. Questa funzione setta il valore del parametro *Mode* del pacchetto a 2, impostando cioè come comando di movimento EPSON l'istruzione *Go*. Questo è giustificato dal fatto che, laddove non ci siano variazione spaziali ma solo di orientazione, il comando *Curve* non è utilizzabile, ed è necessario utilizzare in sua vece il *Go*.
Il vantaggio principale di tale istruzione riguarda la possibilità di poter inviare un solo punto alla volta senza che il programma vada in fault (come accadrebbe nel caso del *Curve*), per cui anche in questo caso la latenza viene ridotta al minimo dall'invio di un punto per volta ad ogni periodo di campionamento.

Per quanto riguarda il calcolo della velocità, in questo caso non ci sono variazioni spaziali ma solo di orientazione. È possibile dunque ricorrere ad una formula per ricavare una misura della velocità angolare: memorizzato in $\mathbf{p_0}$ l'ultimo punto inviato, e chiamato $\mathbf{p_1}$ il punto da inviare, partendo dagli angoli RPY relativi si può impostare la velocità come:

$$\omega = A_f \frac{\sqrt{(\psi_1 - \psi_0)^2 + (\vartheta_1 - \vartheta_0)^2 + (\varphi_1 - \varphi_0)^2}}{T_c} \tag{5.31}$$

dove, come di consueto, φ, ϑ e ψ identificano rispettivamente gli angoli di *roll*, *pitch* e *yaw*. L'Eq. 5.31 rappresenta una formulazione alternativa che tiene conto della velocità angolare anziché di quella lineare.

Considerando quanto detto, nel caso di movimenti di sola orientazione, il pacchetto comprenderà un solo punto da inviare ad ogni ciclo, e sarà composto come indicato in Fig. 5.25. Di conseguenza, il pacchetto complessivo da inviare al software EPSON risulta costituito come in Fig. 5.26.

1	2	3	4	5	6	7	8	9
Header	**World**	**Speed**	**Accel**	**Fine**	**Mode**	**Npts**	**Via Points**	**Checksum**
240	1	w	*variable*	0.1	2	1	P_1	*variable*

Figura 5.26: Struttura del pacchetto inviato al robot in modalità end-effector orientation.

5.8.3 Movimentazione dell'organo terminale nello spazio tridimensionale

Gestire solo l'orientazione in un punto fisso, però, può risultare molto macchinoso: può capitare infatti di volersi anche spostare mentre si effettua una variazione di orientamento. Basti pensare ad esempio al caso in cui la terna utensile del manipolatore debba avvicinarsi ad un oggetto per eseguire operazioni di lavorazione o raccoglimento, principalmente muovendosi lungo l'asse di approccio del polso.
Poiché la terna del dispositivo serve per gestire l'orientazione, una possibile soluzione per aggiungere anche il controllo del movimento potrebbe essere quella di utilizzare il valore di accelerazione lineare per risalire al movimento indotto dall'utente sul dispositivo: muovendo la piattaforma nella direzione voluta, infatti, è possibile risalire dalla misura del valore di accelerazione alla direzione del moto imposta con tale gesto.
Ovviamente, in linea teorica, la conoscenza dei valori di accelerazione lineare permetterebbe di risalire alla posizione in ogni istante tramite doppia integrazione; in questo ambito, rispetto a quanto svolto nel capitolo precedente, risulta rischioso l'utilizzo di questa tecnica. Ciò è dovuto al fatto che l'intervallo di integrazione potrebbe essere di qualche secondo e le tecniche di eliminazione del drift in velocità non efficaci.
È dunque necessario cercare un approccio più valido, che sia allo stesso tempo funzionale e comodo. Valutando il problema più nel dettaglio si nota come, in realtà, non sia necessario garantire il movimento in ogni direzione spaziale: dato un certo orientamento dell'end-effector, le movimentazioni più comuni sono quelle lungo gli assi della terna utensile, ossia avvicinamento-allontanamento, spostamento verso destra-sinistra e verso l'alto-basso. Eseguendo separatamente queste traslazioni,

si riesce a garantire comunque un buon controllo della terna utensile del robot, sfruttando a pieno tutte le potenzialità offerte dalla piattaforma.
In particolare, quindi, la movimentazione in modalità *traiettoria in end-effector* sarà composta da una variazione di orientazione e, nell'eventualità, anche da un movimento spaziale lungo uno dei tre assi della terna utensile, innescata da una traslazione della piattaforma lunga la direzione desiderata.

L'implementazione del processo appena descritto richiede la conoscenza dei valori di accelerazione lineare $\mathbf{a}$ misurati dal dispositivo. Tali segnali vengono aggiornati e resi disponibili assieme alla matrice di rotazione $\mathbf{R}_j^a$, ossia ad ogni ciclo di campionamento. I valori ottenuti, però, rappresentano i dati misurato nel sistema di riferimento dell'IMU, ossia $\mathbf{a}^a$. Ciò che invece sarebbe auspicabile avere per risalire alla traslazione effettuata nelle coordinate utensile è il valore misurato nel sistema $\{e\}$. Poiché con le notazioni impiegate

$$\mathbf{a}^a = \mathbf{R}_j^a \mathbf{R}_i^j \mathbf{R}_e^i \mathbf{a}^e = \mathbf{R}_j^a \mathbf{R}_b^{0'} \mathbf{R}_e^b \mathbf{a}^e \tag{5.32}$$

invertendo l'espressione per ricavare $\mathbf{a}^e$ si perviene alla formula risultante:

$$\mathbf{a}^e = (\mathbf{R}_j^a \mathbf{R}_b^{0'} \mathbf{R}_e^b)^T \mathbf{a}^a \tag{5.33}$$

Dopo aver ricavato l'espressione dell'accelerazione lineare nel riferimento opportuno, quello che rimane da fare è un controllo della traslazione effettuata: una (doppia) integrazione parziale su ogni componente porta, come già detto, a risultati poco soddisfacenti. L'alternativa più semplice è un metodo ad isteresi, in cui al superamento di un certo valore di accelerazione, definito come *limite positivo*, viene innescato il comando di movimento nella direzione voluta, che si protrae fino a quando tale valore non scenda al di sotto di un'ulteriore soglia, chiamata *limite negativo*.
Questo tipo di controllo, nella sua semplicità, è sufficiente a consentire una serie di traslazioni molto efficaci quando si è intenzionati a muovere il robot in determinate direzioni, mantenendo anche il controllo dell'orientazione. Per consentire movimenti a diverse velocità, si possono introdurre più soglie limite positive, in maniera da riconoscere se la movimentazione effettuata con il dispositivo sia stata più brusca oppure più lieve.

In particolare, il processo di creazione punti per la traslazione è riconducibile a quanto già spiegato in 5.8.1 per la movimentazione planare: in questo caso la velocità puntuale con cui vengono generati i vari punti a partire dall'ultimo memorizzato, anziché essere generata dall'inclinazione del joystick, viene scelta sulla base dell'ultima soglia superata dal valore di accelerazione. Non appena viene superato il primo *limite positivo*, e finché l'accelerometro restituisce un valore superiore al *limite negativo*, il programma Matlab continua a chiamare la funzione che si occupa di generare i punti da inviare al robot. Tale funzione deve essere molto simile a quella che genera la traiettoria planare, *CreatePathTrajectory*, e dovrebbe differire da essa in pochi aspetti:

- Deve generare i punti successivi lungo la direzione indicata dai tre assi nel sistema di riferimento dell'end-effector, mentre *CreatePathTrajectory* genera i punti solamente sul piano z.
- Deve permetterere, a differenza della controparte *CreatePathTrajectory*, di poter variare l'orientazione di ogni punto.

A parte queste differenze, il procedimento di creazione è pressoché identico: il comando di movimento utilizzato in questi casi è l'istruzione *Curve*, e vengono generati tre punti di cui quello mediano è ottenuto per interpolazione del punto medio.
Queste riflessioni portano alla considerazione che, visto che la funzione cercata è una generalizzazione della *CreatePathTrajectory*, forse è più conveniente utilizzare un'unica funzione, chiamata sempre *CreatePathTrajectory*, che viene estesa per funzionare anche nel caso in esame: ad essa viene aggiunto come parametro di input una variabile booleana che funge da selettore di aggiornamento dei parametri di rotazione (vera per la *traiettoria in end-effector*, falsa per il movimento planare) e il *displacement* d_z, che varrà sempre 0 nel caso di movimento planare. In questo modo la funzione viene estesa a entrambe le tipologie di funzionamento con poche variazioni del suo codice, facilitando il funzionamento e la comprensione dell'algoritmo anche per il caso di *traiettoria in end-effector*.
In sostanza, dati il punto di inizio $\mathbf{p_0}$ e il punto di fine $\mathbf{p_1}$, il punto medio $\mathbf{p_m}$ viene ottenuto come:

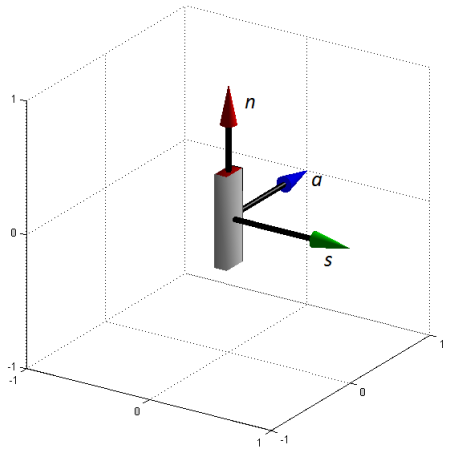


Figura 5.27: Convenzioni della terna del dispositivo: n (normal), s (sliding) e a (approach).

$$\begin{cases} p_{mx} = \frac{p_{0x}+p_{1x}}{2} \\ p_{my} = \frac{p_{0y}+p_{1y}}{2} \\ p_{mz} = \frac{p_{0z}+p_{1z}}{2} \\ p_{m\varphi} = \frac{p_{0\varphi}+p_{1\varphi}}{2} \\ p_{m\vartheta} = \frac{p_{0\vartheta}+p_{1\vartheta}}{2} \\ p_{m\psi} = \frac{p_{0\psi}+p_{1\psi}}{2} \end{cases} \tag{5.34}$$

La creazione del punto $\mathbf{p_1}$, ossia del punto successivo, a partire da $\mathbf{p_0}$, ossia da quello precedente, viene effettuata valutando la direzione intenzionale di moto, ossia controllando quale sia la coordinata di accelerazione che è stata innescata dal movimento del dispositivo.

Con le convenzioni assunte, al dispositivo sono associati gli assi *normal* (n), *sliding* (s) e *approach* (a), che coincidono, come mostrato in Fig. 5.27, rispettivamente con gli assi x, y e z. Il moto viene effettuato lungo uno di questi tre assi, e su tali rette devono giacere i punti che andranno generati.

A seconda di quale traslazione si sia deciso di operare (il controllo viene effettuato a monte valutando separatamente quale componente abbia superato la soglia), è necessario trasportare le coordinate del versore opportuno (x, y o z) nel sistema

di riferimento calibrato. A partire dalla matrice di rotazione finale, espressa dall'eq. 5.29, i versori sono calcolabili come:

$$\mathbf{v} = \mathbf{R}_{fin}\mathbf{e} \tag{5.35}$$

dove $\mathbf{e}$ è un vettore di selezione della componente che identifica lungo quale asse dovrà essere eseguito il moto: a seconda dei casi, vale:

$$\begin{cases} \text{direzione } n \\ \mathbf{e} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{cases} \quad \begin{cases} \text{direzione } s \\ \mathbf{e} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{cases} \quad \begin{cases} \text{direzione } a \\ \mathbf{e} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{cases} \tag{5.36}$$

L'eq. 5.35 permette quindi di ricavare le coordinate del versore scelto espresse secondo il sistema di riferimento calibrato, ossia (in caso di taratura corretta) solidale a quello di base del robot.
La scelta di identificare tale versore con la lettera $\mathbf{c}$ non è casuale: similmente a quanto fatto per il calcolo della direzione nel caso di movimento planare (eq. 5.23), anche in questo caso il procedimento è molto simile, con la differenza che ora il moto avviene in tre dimensioni anziché due. Anche in questo caso, è possibile ricavare il vettore di *displacement* $\mathbf{d}$ come $\mathbf{d} = \alpha\mathbf{v}$, dove α è un opportuno fattore correttivo che tiene conto dell'ultimo *limite superiore* attraversato e, dunque, dell'intensità della traslazione impressa.
Una volta ricavato il valore di $\mathbf{d}$, è possibile scrivere le coordinate del punto di arrivo $\mathbf{p_1}$ come

$$\begin{cases} p_{1x} = p_{0x} + d_x \\ p_{1y} = p_{0y} + d_y \\ p_{1z} = p_{0z} + d_z \end{cases} \tag{5.37}$$

I valori di $p_{1\varphi}$ $p_{1\vartheta}$ e $p_{1\psi}$ vengono invece imposti semplicemente dai rispettivi angoli ottenuti applicando l'inversione di Cardano alla matrice $\mathbf{R}_{fin}$. Conoscendo $\mathbf{p_1}$ e $\mathbf{p_0}$, è possibile ricavarsi il punto medio $\mathbf{p_m}$ utilizzando L'eq. 5.34.

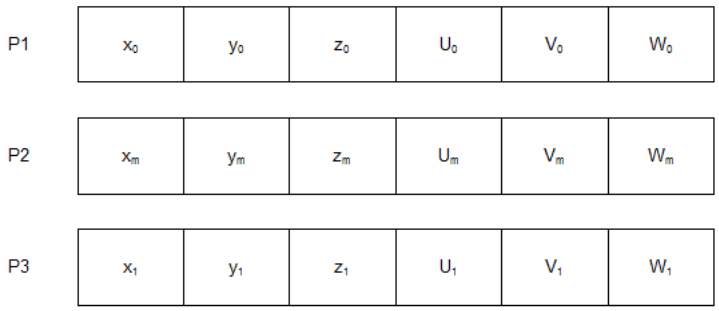


Figura 5.28: Struttura delle coordinate dei punti nel pacchetto inviato al robot in modalità end-effector path (movimento + orientazione).

1	2	3	4	5	6	7	8			9
Header	**World**	**Speed**	**Accel**	**Fine**	**Mode**	**Npts**	**Via Points**			**Checksum**
240	1	v	*variable*	0.1	1	3	P_1	P_2	P_3	*variable*

Figura 5.29: Pacchetto inviato al robot in modalità end-effector path (movimento + orientazione).

Ora che sono disponibili tutti i punti necessari, non resta far altro che assemblare il pacchetto e inviare i dati al robot. Analogamente a quanto fatto per il movimento planare, le coordinate dei punti da inviare nel pacchetto saranno costituite come mostrato in Fig. 5.28:

La struttura del pacchetto per intero, invece, è visibile in Fig. 5.29 ed è identica a quella già descritta durante il movimento planare:

In Fig. 5.30 è mostrato un esempio del funzionamento in modalità *traiettoria in end-effector*. Nella striscia *a)* viene evidenziata la posizione di partenza del joystick, il valore dell'accelerazione lineare misurata per ciascuno dei tre assi e la postura corrispondente del robot. In *b)*, mantenendo la stessa orientazione, viene traslato in dispositivo lungo la direzione di *approach*, ossia lungo l'asse z del riferimeno end-effector: si vede come il corrispondente valore di accelerazione lungo l'asse z subisca un incremento, e di conseguenza anche il robot si muoverà in tale direzione. Terminato questo movimento, in *c)* viene infine imposta una traslazione della piattaforma lungo la direzione x (*normal*) di intensità inferiore: questo viene evidenziato dal corrispondente valore di accelerazione lungo x, che

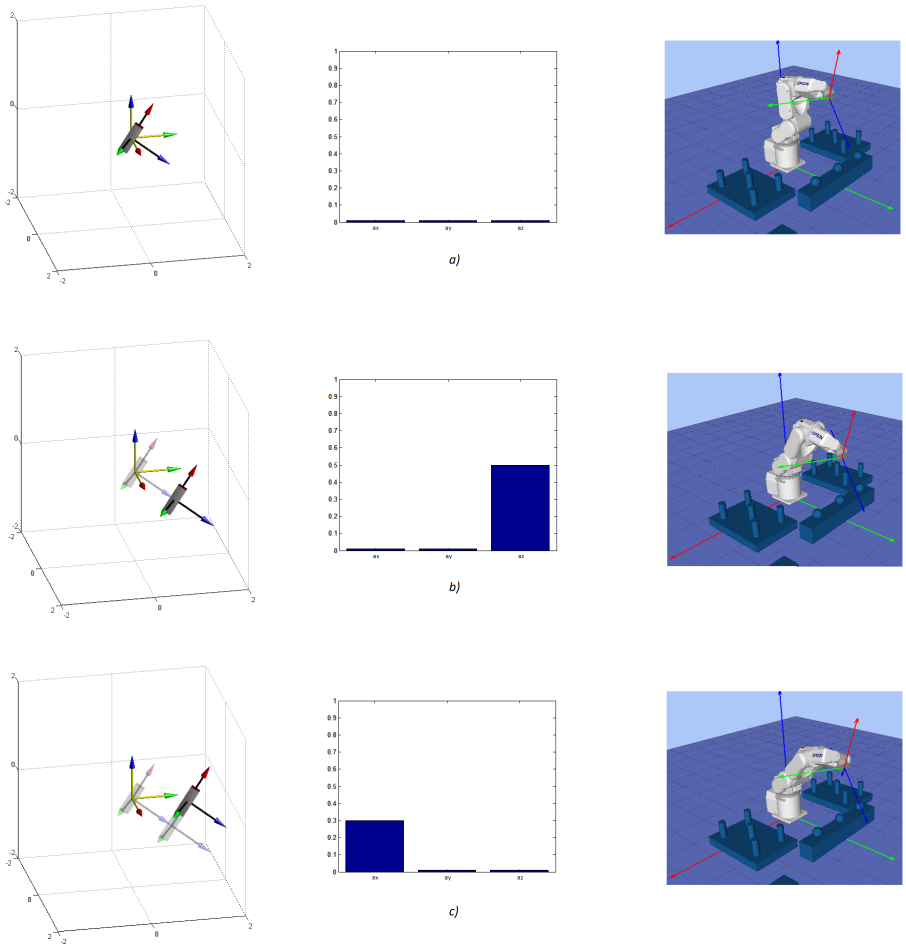


Figura 5.30: Esempio di funzionamento della *traiettoria in end-effector*: a) posizione di partenza, b) avvicinamento (movimento in *approach*, c) spostamento in alto (movimento in *normal*).

comporta di conseguenza un movimento del manipolatore lungo tale asse, anche se il percorso coperto è inferiore rispetto al caso *b)*.

Alla luce di tutte le modalità viste finora, si può procedere all'unificazione degli algoritmi di gestione in maniera da creare una infrastruttura solida e semplice da utilizzare. In particolare, è necessario operare una revisione per l'attivazione della parte appena descritta, in quanto il suo impiego con un solo pulsante risulta molto difficoltoso.

Il problema principale della *traiettoria in end-effector*, infatti, è la necessità di effettuare una traslazione spaziale della piattaforma per poter permettere il

movimento del manipolatore lungo i tre assi previsti. Questo gesto, all'atto pratico, modifica in maniera più o meno marcata anche l'orientazione del dispositivo: il polso dell'utente difficilmente rimarrà nella stessa orientazione durante il movimento lungo l'asse scelto. Tale problema è molto rilevante qualora si cerchi di avvicinare l'end effector lungo l'asse di *approach* mantenendo il più possibile tale direzione. Una soluzione potrebbe essere quella di destinare la pressione simultanea dei pulsanti *rosso* e *nero* per eseguire un movimento di sola traslazione, mentre la variazione di orientazione viene effettuata con la pressione del pulsante *rosso*. In tal modo, si sceglie l'orientamento voluto per l'end-effector premendo il pulsante rosso e muovendo la piattaforma nella posizione cercata, dopodiché, mantenendo sempre la pressione sul rosso, si attiva anche il nero e in questo modo si blocca l'aggiornamento dell'orientamento. Ora l'unico dato su cui è possibile agire è quello dell'accelerazione lineare, che permette una traslazione nella direzione imposta secondo il metodo già descritto. Se ora si rilascia la pressione del rosso e si mantiene solo il nero, il dispositivo continuerà a non aggiornare le informazioni di orientazione, ma non sarà possibile agire nemmeno su quella del posizionamento. Questo accorgimento è necessario per dare la possibilità all'utente di riposizionare il dispositivo nel luogo di partenza senza modificare lo stato del robot: può accadere infatti che dopo una serie di traslazioni l'utente si sia allontanato dal punto in cui era inizialmente partito, e in tal caso la necessità più impellente è quella di tornare in tale luogo senza però intaccare lo stato del robot. Questo meccanismo garantisce la possibilità di effettuare tale operazione, favorendo così più operazioni di traslazione in sequenza e offrendo anche lo spunto di poter gestire in maniera più precisa la velocità di spostamento. Ovviamente, rilasciando tutti i pulsanti e premendo solo in nero si attiva, come di consueto, la modalità di movimento planare.

Per permettere il riconoscimento dell'azione di *traiettoria in end-effector* piuttosto che quella di movimento planare, è prevista una variabile aggiuntiva chiamata *ee-path_flag*, la quale assume valore *true* alla pressione del pulsante rosso, mentre ritorna allo stato *false* al rilascio di tutti i pulsanti.

Una miglior visione d'insieme è resa disponibile e visibile in Fig. 5.31, dove è schematizzato l'intero algoritmo in forma di *flow chart*.

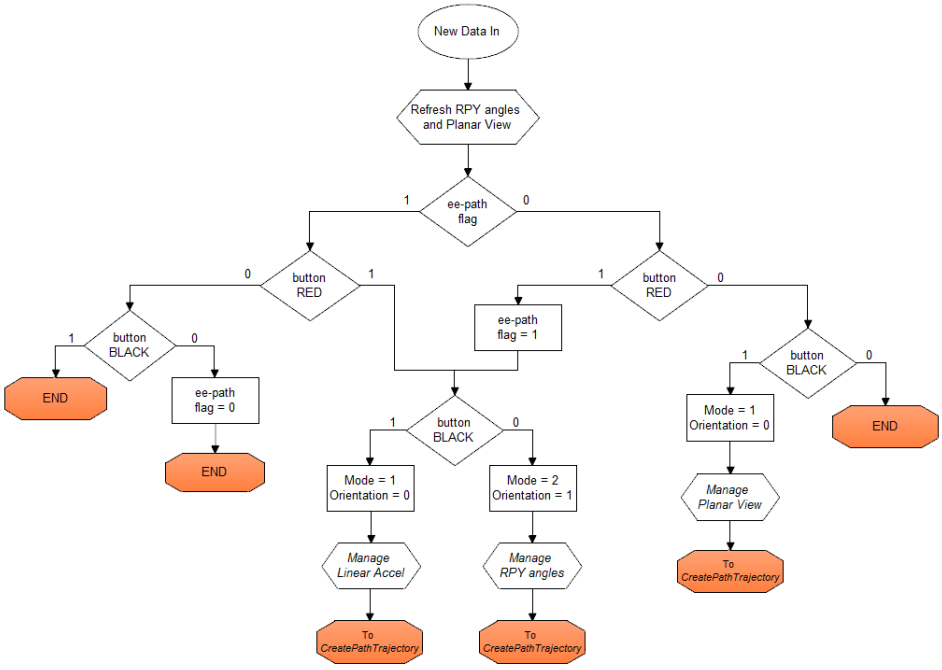


Figura 5.31: Diagramma dell'algoritmo per l'utilizzo del dispositivo.

Conclusioni

La crescente necessità di far fronte a produzioni industriali caratterizzate da elevata personalizzazione e cicli di vita estremamente brevi richiede elevata flessibilità dei sistemi di produzione e assemblaggio. Molte aziende cercano di progettare e controllare i propri sistemi produttivi in modo da potersi adattare velocemente alle richieste del mercato in termini di volumi produttivi e di costi operativi. Lo sviluppo di nuove tecnologie per i moderni sistemi di assemblaggio risulta fondamentale affinché le aziende riescano a mantenere elevate produttività e basso costo unitario. La flessibilità può essere suddivisa in flessibilità di prodotto e flessibilità di layout. La flessibilità di prodotto consiste nella possibilità di manipolare componenti diversi fra loro per forma e dimensione utilizzando la stessa automazione. D'altro canto, la flessibilità di layout rappresenta la possibilità di poter riconfigurare l'ambiente di lavoro oppure il ciclo operativo in tempi rapidi e senza fermare la produzione. Infatti, ad ogni modifica della configurazione della cella di lavoro o del ciclo operativo potrebbe essere necessario ridefinire le locazioni relative ai movimenti del manipolatore oppure ricalibrare i sensori utilizzati all'interno della cella robotizzata. Tutte queste operazioni sono complesse ed onerose in termini di tempo e denaro. In questa tesi di dottorato, sono state studiate ed implementate delle tecniche innovative per incrementare la flessibilità di layout rendendo più rapide ed intuitive le operazioni di aggiornamento della configurazione di una cella robotizzata. In particolare, sono state sviluppate procedure rapide per la calibrazione di telecamere industriali e strumenti per l'acquisizione tridimensionale dell'ambiente di lavoro e l'importazione dello stesso in un ambiente CAD commerciale. Un simulatore robotico permette la navigazione off-line di un manipolatore nella geometria acquisita, la memorizzazione off-line

delle traiettorie robot e dei punti di via, oltre che alla gestione dei dati nel ciclo di lavoro in tempo reale all'interno dell'ambiente CAD. Tali procedure si sono dimostrate rapide ed implementabili anche da personale non specializzato riducendo così il tempo di configurazione della cella ed il conseguente fermo della produzione. Otre alle procedure rapide di configurazione dell'ambiente di lavoro, una soluzione interessante per incrementare la flessibilità in ambiente industriale consiste nella possibilità di interazione uomo-robot in sicurezza: un dispositivo di scarpa strumentata in grado di misurare parametri spazio-temporali della camminata, potrebbe essere utilizzata per stimare la posizione di un operatore all'interno di una cella robotizzata. Inoltre è stato ideato, progettato e costruito un innovativo sistema di guida rapida per robot industriali basato su sensori inerziali. Tale sistema è potenzialmente ricco di sviluppi in quanto, oltre alla possibilità di inserire numerose funzionalità per rendere ancora più rapida ed intuitiva la guida robot, potrebbe prevedere delle procedure di interazioni fra robot e dispositivo in modo da renderlo sempre più facilmente utilizzabile.

Bibliografia

[1] G. Reinhart and M. Loy, "Design of a modular feeder for optimal operating performance," *CIRP Journal of Manufacturing Science and Technology*, vol. 3, no. 3, pp. 191–195, 2010.

[2] J. Heilala and P. Voho, "Modular reconfigurable flexible final assembly systems," *Assembly Automation*, vol. 21, no. 1, pp. 20–30, 2001.

[3] T. Frädrich, J. Pachow-Frauenhofer, F. Torsten, and P. Nyhuis, "Aerodynamic feeding systems: an example for changeable technology," *Assembly Automation*, vol. 31, no. 1, pp. 47–52, 2011.

[4] A. Perks, "Advanced vision guided robotics provide future-proof flexible automation," *Assembly Automation*, vol. 26, no. 3, pp. 216–220, 2006.

[5] D. Gudmundsson and K. Goldberg, "Optimizing robotic part feeder throughput with queueing theory," *Assembly Automation*, vol. 27, no. 2, pp. 134–140, 2007.

[6] F. Al-Ghathian, M. Tarawneh, M. Nawafleh, and N. Al-Kloub, "A mechanical device used to improve the vision inspection in small scale industry," *International Journal of Mechanical and Materials Engineering*, vol. 5, no. 2, pp. 191–8, 2010.

[7] S. Ravikumar, K. Ramachandran, and V. Sugumaran, "Machine learning approach for automated visual inspection of machine components," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3260–3266, 2011.

[8] G. Rosati, G. Boschetti, A. Biondi, and A. Rossi, "On-line dimensional measurement of small components on the eyeglasses assembly line," *Optics and Lasers in Engineering*, vol. 47, no. 3, pp. 320–328, 2009.

[9] S. Bi and J. Liang, "Robotic drilling system for titanium structures," *The International Journal of Advanced Manufacturing Technology*, vol. 54, no. 5-8, pp. 767–774, 2011.

[10] G. Rosati, M. Faccio, A. Carli, and A. Rossi, "Fully flexible assembly systems (f-fas): a new concept in flexible automation," *Assembly Automation*, vol. 33, no. 1, pp. 8–21, 2013.

[11] G. Rosati, M. Faccio, C. Finetto, and A. Carli, "Modelling and optimization of fully flexible assembly systems (f-fas)," *Assembly Automation*, vol. 33, no. 2, pp. 165–174, 2013.

[12] C. Finetto, M. Faccio, G. Rosati, and A. Rossi, "Mixed-model sequencing optimization for an automated single-station fully flexible assembly system (f-fas)," *The International Journal of Advanced Manufacturing Technology*, vol. 70, no. 5-8, pp. 797–812, 2014.

[13] F. Boughorbel, Y. Zhang, S. Kang, U. Chidambaram, B. Abidi, A. Koschan, and M. Abidi, "Laser ranging and video imaging for bin picking," *Assembly Automation*, vol. 23, no. 1, pp. 53–59, 2003.

[14] Z. Pan, J. Polden, N. Larkin, S. Van Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 87–94, 2012.

[15] M. Bruccoleri, C. D'Onofrio, and U. La Commare, "Off-line programming and simulation for automatic robot control software generation," in *Industrial Informatics, 2007 5th IEEE International Conference on*, vol. 1. IEEE, 2007, pp. 491–496.

[16] S. Mitsi, K.-D. Bouzakis, G. Mansour, D. Sagris, and G. Maliaris, "Off-line programming of an industrial robot for manufacturing," *The International*

Journal of Advanced Manufacturing Technology, vol. 26, no. 3, pp. 262–267, 2005.

[17] R. EN-ISO 10218-1:2011 and robotic devices Safety requirements for industrial robots Part 1: Robots.

[18] R. EN-ISO 10218-2:2011, robotic devices Safety requirements for industrial robots Part 2: Robot Systems, and Integration.

[19] D. M. Ebert and D. D. Henrich, "Safe human-robot-cooperation: Image-based collision detection for industrial robots," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 2. IEEE, 2002, pp. 1826–1831.

[20] N. Pedrocchi, F. Vicentini, M. Matteo, and L. M. Tosatti, "Safe human-robot cooperation in an industrial environment," *Int J Adv Robotic Sy*, vol. 10, no. 27, 2013.

[21] R. EN-ISO2018-2, robotic devices Safety requirements for industrial robots Part 2. Robot systems, and integration.

[22] Mathworks, "Introducing mex-files," 2015.

[23] J.-Y. Bouguet, "Matlab calibration toolbox."

[24] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[25] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo, *Time-of-flight cameras and Microsoft Kinect*. Springer, 2012.

[26] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606.

[27] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: real-time 3d

reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 559–568.

[28] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

[29] A. TECHNOLOGY, "Adept viper 6-axis robots," 1996-2013. [Online]. Available: http://www.adept.com/products/robots/6-axis/viper-s650/general

[30] S. R. Simon, "Quantification of human motion: gait analysis−benefits and limitations to its application to clinical problems," *Journal of Biomechanics*, vol. 37, no. 12, pp. 1869 – 1880, 2004.

[31] T. A. Wren, G. E. G. III, S. Ounpuu, and C. A. Tucker, "Efficacy of clinical gait analysis: A systematic review," *Gait & Posture*, vol. 34, no. 2, pp. 149 – 153, 2011.

[32] J. G. Nutt, B. R. Bloem, N. Giladi, M. Hallett, F. B. Horak, and A. Nieuwboer, "Freezing of gait: moving forward on a mysterious clinical phenomenon," *The Lancet Neurology*, vol. 10, no. 8, pp. 734–744, 2011.

[33] K. Lebel, P. Boissy, M. Hamel, and C. Duval, "Inertial measures of motion for clinical biomechanics: Comparative assessment of accuracy under controlled conditions-effect of velocity," *PloS one*, vol. 8, no. 11, p. e79945, 2013.

[34] A. M. Sabatini, C. Martelloni, S. Scapellato, and F. Cavallo, "Assessment of walking features from foot inertial sensing," *Biomedical Engineering, IEEE Transactions on*, vol. 52, no. 3, pp. 486–494, 2005.

[35] H. M. Schepers, H. Koopman, and P. H. Veltink, "Ambulatory assessment of ankle and foot dynamics," *Biomedical Engineering, IEEE Transactions on*, vol. 54, no. 5, pp. 895–902, 2007.

[36] J.-A. Lee, S.-H. Cho, J.-W. Lee, K.-H. Lee, and H.-K. Yang, "Wearable accelerometer system for measuring the temporal parameters of gait," in *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*. IEEE, 2007, pp. 483–486.

[37] S. J. M. Bamberg, A. Y. Benbasat, D. M. Scarborough, D. E. Krebs, and J. A. Paradiso, "Gait analysis using a shoe-integrated wireless sensor system," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 12, no. 4, pp. 413–423, 2008.

[38] B. Mariani, C. Hoskovec, S. Rochat, C. Büla, J. Penders, and K. Aminian, "3d gait assessment in young and elderly subjects using foot-worn inertial sensors," *Journal of biomechanics*, vol. 43, no. 15, pp. 2999–3006, 2010.

[39] S. Slajpah, R. Kamnik, and M. Munih, "Human motion kinematics assessment using wearable sensors," in *Advances in Robot Kinematics*. Springer, 2014, pp. 171–180.

[40] J. Nutt, C. Marsden, and P. Thompson, "Human walking and higher-level gait disorders, particularly in the elderly," *Neurology*, vol. 43, no. 2, pp. 268–268, 1993.

[41] D. A. Winter, *Biomechanics and motor control of human gait: normal, elderly and pathological*, 1991.

[42] F. Prince, H. Corriveau, R. Hébert, and D. A. Winter, "Gait in the elderly," *Gait & Posture*, vol. 5, no. 2, pp. 128–135, 1997.

[43] J. Rose, J. G. Gamble, and J. M. Adams, *Human walking*. Lippincott Williams & Wilkins Philadelphia, 2006.

[44] J. Perry, J. R. Davids, *et al.*, "Gait analysis: normal and pathological function," *Journal of Pediatric Orthopaedics*, vol. 12, no. 6, p. 815, 1992.

[45] E. Foxlin, "Pedestrian tracking with shoe-mounted inertial sensors," *Computer Graphics and Applications, IEEE*, vol. 25, no. 6, pp. 38–46, 2005.

[46] A. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu," in *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on.* IEEE, 2009, pp. 37–42.

[47] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S.-M. Makela, and H. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, vol. 2. IEEE, 2005, pp. ii–973.

[48] E. S. Sazonov, G. Fulk, J. Hill, Y. Schutz, and R. Browning, "Monitoring of posture allocations and activities by a shoe-based wearable sensor," *Biomedical Engineering, IEEE Transactions on*, vol. 58, no. 4, pp. 983–990, 2011.

[49] K. Aminian, B. Najafi, C. Büla, P.-F. Leyvraz, and P. Robert, "Spatio-temporal parameters of gait measured by an ambulatory system using miniature gyroscopes," *Journal of biomechanics*, vol. 35, no. 5, pp. 689–699, 2002.

[50] T. Liu, Y. Inoue, and K. Shibata, "Development of a wearable sensor system for quantitative gait analysis," *Measurement*, vol. 42, no. 7, pp. 978–988, 2009.

[51] G. Cooper, I. Sheret, L. McMillian, K. Siliverdis, N. Sha, D. Hodgins, L. Kenney, and D. Howard, "Inertial sensor-based knee flexion/extension angle estimation," *Journal of biomechanics*, vol. 42, no. 16, pp. 2678–2685, 2009.

[52] T. Watanabe, H. Saito, E. Koike, and K. Nitta, "A preliminary test of measurement of joint angles and stride length with wireless inertial sensors for wearable gait evaluation system," *Computational intelligence and neuroscience*, vol. 2011, p. 6, 2011.

[53] T. Seel, J. Raisch, and T. Schauer, "Imu-based joint angle measurement for gait analysis," *Sensors*, vol. 14, no. 4, pp. 6891–6909, 2014.

[54] S. Simcox, S. Parker, G. M. Davis, R. W. Smith, and J. W. Middleton, "Performance of orientation sensors for use with a functional electrical stimulation mobility system," *Journal of biomechanics*, vol. 38, no. 5, pp. 1185–1190, 2005.

[55] C. Strohrmann, H. Harms, C. Kappeler-Setz, and G. Troster, "Monitoring kinematic changes with fatigue in running using body-worn sensors," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 16, no. 5, pp. 983–990, 2012.

[56] T. Liu, Y. Inoue, and K. Shibata, "A wearable force plate system for the continuous measurement of triaxial ground reaction force in biomechanical applications," *Measurement Science and Technology*, vol. 21, no. 8, p. 085804, 2010.

[57] H. Martin Schepers, E. H. Van Asseldonk, C. Baten, and P. H. Veltink, "Ambulatory estimation of foot placement during walking using inertial sensors," *Journal of biomechanics*, vol. 43, no. 16, pp. 3138–3143, 2010.

[58] J. Favre, B. Jolles, O. Siegrist, and K. Aminian, "Quaternion-based fusion of gyroscopes and accelerometers to improve 3d angle measurement," *Electronics Letters*, vol. 42, no. 11, pp. 612–614, 2006.

[59] B. Mariani, S. Rochat, C. Bula, and K. Aminian, "Heel and toe clearance estimation for gait analysis using wireless inertial sensors," *Biomedical Engineering, IEEE Transactions on*, vol. 59, no. 11, pp. 3162–3168, 2012.

[60] A. Bregou Bourgeois, B. Mariani, K. Aminian, P. Zambelli, and C. Newman, "Spatio-temporal gait analysis in children with cerebral palsy using, foot-worn inertial sensors," *Gait & posture*, vol. 39, no. 1, pp. 436–442, 2014.

[61] B. Mariani, M. C. Jimenez, F. J. Vingerhoets, and K. Aminian, "On-shoe wearable sensors for gait and turning assessment of patients with parkinson's disease," *Biomedical Engineering, IEEE Transactions on*, vol. 60, no. 1, pp. 155–158, 2013.

[62] T. Liu, Y. Inoue, K. Shibata, and K. Shiojima, "A mobile force plate and three-dimensional motion analysis system for three-dimensional gait assessment," *Sensors Journal, IEEE*, vol. 12, no. 5, pp. 1461–1467, 2012.

[63] A. Young, "Comparison of orientation filter algorithms for realtime wireless inertial posture tracking," in *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on.* IEEE, 2009, pp. 59–64.

[64] L. Shu, T. Hua, Y. Wang, Q. Li, D. D. Feng, and X. Tao, "In-shoe plantar pressure measurement and analysis system based on fabric pressure sensing array," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 14, no. 3, pp. 767–775, 2010.

[65] S. Crea, M. Donati, S. M. M. De Rossi, C. M. Oddo, and N. Vitiello, "A wireless flexible sensorized insole for gait analysis," *Sensors*, vol. 14, no. 1, pp. 1073–1093, 2014.

[66] O. Bebek, M. A. Suster, S. Rajgopal, M. J. Fu, X. Huang, M. C. Cavusoglu, D. J. Young, M. Mehregany, A. J. van den Bogert, and C. H. Mastrangelo, "Personal navigation via high-resolution gait-corrected inertial measurement units," *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 11, pp. 3018–3027, 2010.

[67] P. M. Kennedy and J. T. Inglis, "Distribution and behaviour of glabrous cutaneous receptors in the human foot sole," *The Journal of Physiology*, vol. 538, no. 3, pp. 995–1002, 2002.

[68] D. Zanotto, L. Turchet, E. M. Boggs, and S. K. Agrawal, "Solesound: Towards a novel portable system for audio-tactile underfoot feedback," in *Biomedical Robotics and Biomechatronics (2014 5th IEEE RAS & EMBS International Conference on.* IEEE, 2014, pp. 193–198.

[69] L. Turchet, S. Serafin, S. Dimitrov, and R. Nordahl, "Physically based sound synthesis and control of footsteps sounds," in *13th International Conference on Digital Audio Effects (DAFx-10)*, 2010, pp. 161–168.

[70] H. Sadeghi, S. Sadeghi, F. Prince, P. Allard, H. Labelle, and C. L. Vaughan, "Functional roles of ankle and hip sagittal muscle moments in able-bodied gait," *Clinical Biomechanics*, vol. 16, no. 8, pp. 688–695, 2001.

Ringraziamenti

Un sincero e sentito ringraziamento al mio supervisore Prof. Aldo Rossi ed al Prof. Giulio Rosati per la fiducia accordatami.
Ringrazio il Prof. S. K. Agrawal per la sua ospitalità presso il laboratorio Rehabilitation And Robotics (ROAR) presso la Columbia University.
Un immane grazie a Damiano Zanotto per il suo imprescindibile ed enorme aiuto, supporto, guida ed insegnamento nel periodo di ricerca effettuato all'estero.
Un ringraziamento sentito anche ad Emily Marie Boggs e Stephan Stansfield per il lavoro svolto insieme nel progetto SoleSound.
Un enorme grazie anche ai miei attuali o precedenti colleghi, dai quali ho potuto imparare moltissimo: Luca Barbazza, Andrea Carli, Christian Finetto e Fabio Oscari.