Università degli Studi di Padova

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI

Corso di Laurea Magistrale in Ingegneria Meccatronica

Utilizzo del controllo di impedenza nei robot collaborativi

Candidato:
Nicola Mosele
Matricola 1156705

Relatore:
Paolo Boscariol

Correlatore:
Giovanni Boschetti

Anno Accademico 2018–2019

Voglio ringraziare tutti coloro che mi sono stati vicini in questo percorso di studi che si conclude oggi. Dalla mia famiglia ai miei amici, dai professori ai colleghi e a tutte le persone che mi hanno aiutato e arricchito di conoscenza e spirito. Un ringraziamento speciale alla mia famiglia, in particolare a mia madre e mio padre: è grazie al loro sostegno e al loro incoraggiamento se oggi sono riuscito a raggiungere questo traguardo. Un ringraziamento a mia sorella, ai nonni, zii, cugini perchè hanno sempre creduto in me e mi hanno sempre incitato a non mollare. Un ringraziamento ai miei amici più cari, che durante questo percorso mi hanno aiutato mescolando il dovere, lo studio, con il piacere, con l'amicizia e il divertimento. Un ringrazimento particolare al relatore e correlatore, che mi hanno aiutato nella stesura della tesi e mi hanno dato la possibilità di appassionarmi ad un ambito per me poco conosciuto. Per concludere un grazie a tutti, perchè ogni persona che ho incontrato, in qualche modo, mi ha influenzato e ha partecipato alla mia crescita.

Indice

Sommario

Lo scopo di questa tesi è la realizzazione di task robotici avanzati che sfruttino le capacità collaborative del robot KUKA LBR iiwa. Per fare ciò si andranno ad analizzare diversi aspetti, dall'analisi cinematica diretta e inversa, al software di programmazione e all'implementazione di codice che viene eseguito dal robot. In una prima parte del lavoro verranno trattate l'analisi cinematica diretta e inversa del robot, successivamente si analizzeranno le potenzialità offerte dal software di programmazione del robot. Tale fase di analisi si è resa necessaria data la novità imposta dall'utilizzo del Sunrise OS, ovvero il software di gestione del robot LBR iiwa, che si differenzia notevolmente dal tradizionale ambiente di sviluppo utilizzato per tutti gli altri robot prodotti dalla stessa azienda.

Il task definito durante l'attività di tesi realizza il rilevamento della posizione di una superficie all'interno dello spazio di lavoro del robot, che può essere realizzato sia in maniera autonoma da parte del robot, sia in modalità collaborativa. L'applicativo sfrutta la modalità di impedance control, la quale permette di impostare la cedevolezza del robot durante l'esecuzione di un movimento. La rilevazione di una superficie per punti è realizzata monitorando le forze scambiate tra robot e ambiente, nonché tra robot ed operatore. I risultati dell'acquisizione sono poi trasferiti ad un computer esterno tramite una un collegamento TCP/IP, dove i dati acquisiti vengono processati e rappresentati graficamente in uno script Matlab.

Capitolo 1

Introduzione

Con questo elaborato si va a studiare e applicare il controllo di impedenza nei robot collaborativi. L'obiettivo è riuscire a far copiare al robot l'andamento di una superficie grazie ad una traiettoria impostata lungo la quale avviene il movimento. Si sono realizzati diversi passaggi prima di arrivare all'obiettivo principale, per prendere confidenza con il software di programmazione e analizzare la cinematica del robot.

E' nato l'interesse per questo tipo di applicazione perchè in questo periodo di continua evoluzione nell'industria, con l'avvento dell'Industria 4.0, si cerca di far collaborare sempre di più uomo e macchina e pertanto stanno prendendo quota in maniera importante i robot collaborativi. In più l'inserimento di questi dispositivi può portare alla realizzazione di sistemi integrati di robot autonomi non solamente pensati per l'industria ma anche per la domotica e l'aiuto in casa. Si è analizzato un possibile ambito applicativo con lo scopo di studiare, analizzare e capire quali possono essere i vantaggi rispetto a robot industriali.

Vista la recente diffusione dei cobots, non si trovano molte documentazioni su di essi perchè, essendo nati come attività di Ricerca e Sviluppo per le case costruttrici, è difficile trovare articoli o paper che riportino usi diffusi di questi robot. Ad oggi la diffusione di elaborati è legata alla volontà delle aziende produttrici di liberalizzare informazioni su di essi.

In questa prima parte introduttiva si espone un focus sui robot collaborativi, la storia, differenze rispetto ai robot industriali, tipi di applicazioni e controlli e un assaggio di visione futura.

Per quanto riguarda l'organizzazione di questo elaborato, esso si dividerà nelle seguenti parti:

- Setup sperimentale

- Analisi cinematica
- Descrizione ambiente di programmazione Kuka Sunrise.OS
- Attività sperimentale
- Commenti conclusivi e considerazioni finali

1.1 Storico sui robot collaborativi

Un cobot o co-robot (derivante da "collaborative robot") è un robot concepito per interagire fisicamente con l'uomo in uno spazio di lavoro. Ciò trasmette un contrasto della maggior parte dei robot industriali adottati fino al 2008, i quali erano progettati per operare in maniera autonoma o con una guida limitata e protetti da barriere.

I cobot vennero inventati nel 1996 da J. Edward Colgate e Michael Peshkin, professori alla Northwestern University. Un brevetto statunitense, depositato nel 1997, descrive i cobot come "un apparato e un metodo per l'interazione fisica diretta tra una persona e un manipolatore controllato da un computer".

Anche aziende come la General Motors, guidata da Prasad Akella del GM Robotics Center, cercarono un modo per rendere i robot o le attrezzature simili ai robot sicure al punto da poter collaborare con le persone. I primi cobot erano sicuri per l'uomo dato che non avevano alcuna fonte interna di forza motrice. Al contrario, la forza motrice veniva fornita direttamente dall'assistente umano. La funzione del cobot era quindi quella di consentire il controllo del movimento da parte del computer, reindirizzando o guidando il carico, in modo cooperativo con il lavoratore umano. Negli anni a seguire anche i cobot erano così dotati di piccole quantità di forza motrice.

Il team di General Motors ha utilizzato il termine Intelligent Assistant Device (IAD) come alternativa al cobot, specialmente nell'ambito del trasporto di materiali industriali e delle operazioni di assemblaggio nel settore automobilistico. Una bozza di standard di sicurezza per l'Intelligent Assist Devices è stata pubblicata nel 2002. Una versione aggiornata dello standard di sicurezza è stata pubblicata nel 2016. La Cobotics ha rilasciato diversi modelli di cobot nel 2002.

KUKA, azienda tedesca pioniera nel campo della robotica industriale, ha lanciato il suo primo cobot, LBR 3, nel 2004. Questo robot leggero controllato da computer è stato il risultato di una lunga collaborazione con l'Istituto Aerospaziale tedesco. KUKA inoltre ha perfezionato ulteriormente la propria tecnologia, distribuendo il KUKA LBR 4 nel 2008 e il KUKA LBR iiwa nel 2013.

Universal Robots ha rilasciato il suo primo cobot, l'UR5, nel 2008. Nel 2012 è stato distribuito il cobot UR10, e successivamente un robot collaborativo di piccole dimensioni, l'UR3, nel 2015. Rethink Robotics ha distribuito sul mercato un nuovo cobot industriale, denominato Baxter nel 2012 e insieme ad esso il robot più piccolo, più veloce e più collaborativo chiamato Sawyer nel 2015, progettato per compiti ad alta precisione.

FANUC - il più importante produttore al mondo di robot industriali, ha rilasciato il suo primo robot collaborativo nel 2015, il FANUC CR-35iA con un carico fino a 35 kg. Da allora FANUC ha distribuito una linea più piccola di robot collaborativi includendo il FANUC CR-4iA, il CR-7iA e la versione con il braccio lungo CR-7/L.

Figura 1.1: Il primo cobot del mondo venne installato nel dicembre 2008 presso Linatex, un fornitore danese di gomma e plastiche industriali

Oggi i cobots sono specializzati nello svolgimento di compiti specifici che "imparano" direttamente sul campo. Possono essere più o meno autonomi e stanno rivoluzionando i settori della logistica e dell'automazione di fabbrica. Si parla in questo caso di robotica al servizio dell'industria. I robot collaborativi rappresentano una grande opportunità di avanzamento tecnologico in molti settori in cui la robotica è quasi del tutto estranea. Se ne sente parlare sempre più spesso come uno degli elementi fondamentali di Industria 4.0, come tecnologia abilitante di sistemi adattivi, di flessibilità della produzione, di riconfigurabilità, di efficienza. In misura ancora più importante, i robot collaborativi sono il simbolo di un cambiamento graduale ma costante della sensibilità alla creazione di condizioni di lavoro in grado di combinare produttività e salute, ovvero di fabbriche per le persone. In più, con l'avvento della Industria 4.0 i robot collaborativi potranno essere in grado di comunicare con gli altri macchinari o anche tra loro per aumentare l'efficienza e la produttività, riuscendo a gestire situazioni anche complicate "parlandosi".

1.2 Differenze tra robots e cobots

Oggi molti non riescono ancora a capire veramente come i cobot siano diversi dai robot, per cui si analizzano 5 punti chiave che svelano con esattezza le differenze tra le due tipologie. In primis va sottolineato che i robot collaborativi sono decisamente più piccoli e leggeri rispetto ai robot tradizionali.

Collaborazione uomo-macchina

I classici robot industriali sono molto potenti e in genere svolgono il loro lavoro seguendo un programma fisso, senza riguardo per le persone che lavorano intorno a loro. Gli incidenti sono quindi evitabili soltanto usando recinti e gabbie. I cobot invece sono progettati specificamente per lavorare insieme alle persone, senza quindi la necessità di essere ingabbiati, oltre che adatti per svolgere compiti complessi. Ad esempio, possono consegnare componenti vari ai colleghi umani per l'assemblaggio oppure eseguire un controllo di qualità del prodotto finito.

Minore pericolosità

I cobot soddisfano appieno i compiti che potrebbero essere rischiosi per le persone, come ad esempio il trasporto sicuro di pezzi taglienti, appuntiti o caldi, così come eseguire lavori di bullonatura piuttosto pericolosi. Tutte queste funzionalità si traducono in un minor numero di incidenti e lasciano ai tecnici l'opportunità di concentrarsi su aspetti meno ardui della produzione, con conseguenti vantaggi in termini di velocità di determinati lavori e con un sostanziale ritorno sull'investimento che il cobot ha richiesto per essere implementato nella catena di produzione.

Comportamento intelligente

I cobots sono progettati per funzionare perfettamente insieme ai loro colleghi umani che non vanno incontro ad alcun rischio, infatti si immobilizzano al minimo contatto grazie a dei sensori tecnologicamente sofisticati ed appositamente ideati per prevenire qualsiasi pericolo alle persone vicine. Aree chiuse e recinzioni di sicurezza non sono quindi più necessarie in un'azienda che intende far coesistere cobots ed operatori umani.

Facilità di programmazione e utilizzo

Oltre ad essere funzionali grazie alle loro caratteristiche tecniche e strutturali, i cobot sono anche molto facili da programmare, a differenza dei tradizionali robot

che invece ancora oggi richiedono competenze specifiche nel settore. Ad esempio, da un tecnico che esegue un movimento con il braccio del robot UR3 della Universal Robots, il cobot stesso è in grado di riprodurlo automaticamente. Ad altri sistemi possono tra l'altro essere fornite istruzioni di lavoro senza alcuna codifica, utilizzando un'interfaccia utente grafica. I dipendenti possono quindi riprogrammare flessibilmente i cobots e usarli poi per svariati compiti.

Utilizzabili ovunque

I cobot sono anche relativamente facili da spostare e da utilizzare in altri punti della catena di produzione di un'azienda. La maggior parte di questi bracci può tra l'altro essere montata su qualsiasi superficie sia essa orizzontale che verticale così come appeso al soffitto, proprio perché si tratta di strutture molto leggere e che si possono spostare con l'intervento di una sola persona.

1.3 Tipi di applicazioni

I robot collaborativi possono esserre utilizzati per diversi scopi, qui di seguito ne vengono elencati alcuni.

Pick&Place

Le operazioni di Pick&Place, raccolta e posizionamento manuale, sono tra le più ripetitive e banali oggi eseguite dagli operatori umani. La ripetitività comporta spesso un alto tasso di errore e scarso rispetto delle tempistiche. Il Pick&Place è forse la prima applicazione, in termini di facilità e immediatezza, per un cobot. Prendere un oggetto, spostarlo e depositarlo in un altro punto è generalmente la prima funzione per la quale pensare a un cobot in grado di operare in tutta sicurezza nello stesso spazio di un operatore umano.

Machine tending

Il Machine Tending, o asservimento a una macchina, richiede la presenza per lunghe ore di un operatore umano davanti a un macchinario industriale. È un processo lungo, faticoso e impegnativo che riguarda non solo il carico e lo scarico della macchina ma anche altre operazioni del sistema automatizzato di produzione, dall'ispezione e controllo al soffiaggio e lavaggio, fino al confezionamento. I cobot possono liberare da molte di queste operazioni l'operatore umano, anche interagendo con due o più macchine, garantendo maggior produttività.

Imballaggio e pallettizzazione

Imballaggio e pallettizzazione sono funzioni che conseguono dal Pick&Place: al termine del processo produttivo e prima di lasciare la fabbrica, i prodotti devono essere imballati e stipati su pallet. Si tratta di un'operazione ripetitiva, da eseguire con velocità e precisione, che comporta lo spostamento e l'imballaggio di oggetti dalle dimensioni, dalle forme e dai peso più diversi. Nella fase di imballaggio e pallettizzazione i cobot possono migliorare la sincronizzazione con l'intero processo produttivo, la standardizzazione della qualità di imballaggio, la velocizzazione delle operazioni, liberando l'operatore umano per altri compiti non ripetitivi.

Compiti di processo produttivo

Un Process Task è qualsiasi compito di un processo produttivo che richiede l'interazione di un operatore con un oggetto per il tramite di uno strumento. Esempi concreti ne sono la verniciatura, la finitura, la saldatura o l'incollatura. Si tratta di compiti per i quali l'operatore umano non è fisiologicamente in grado di garantire una qualità standard e uniforme. Il cobot può affiancare l'operatore umano svolgendo i compiti più ripetitivi e con elevate richieste di precisione liberando il tempo dell'operatore umano per altre operazioni. È bene sottolineare come per molti di questi compiti di processo produttivo – come la saldatura o la verniciatura – rimane necessaria l'esperienza specifica dell'operatore umano in fase di programmazione robotica.

Compiti di finitura

I compiti di finitura di un prodotto industriale – come levigatura, sbavatura o rettifica – sono meno ripetitivi di quelli esposti precedentemente, tuttavia richiedono di maneggiare uno strumento per lunghe ore, doti di forza e comportano conseguenti rischi di infortuni o patologie per l'operatore umano. Un cobot è in grado di assicurare standardizzazione della qualità, velocità, precisione, dosaggio della forza, efficienza e versatilità per ogni compito di finitura.

Controllo qualità

I cobot possono anche montare videocamere ad altissima risoluzione per effettuare ispezioni e controllo qualità di parti e prodotti industriali: un cobot dotato di videocamere di ispezione può accelerare e standardizzare il controllo e il confronto delle parti rispetto al modello originale CAD, aumentare il livello di qualità finale dei lotti di produzione e ridurre i costi per i pezzi non conformi agli standard.

1.4 La transizione dai robot collaborativi alle applicazioni collaborative

Fin dallo sviluppo e dalla commercializzazione dei primi bracci robotici collaborativi, il mercato della robotica industriale ha conosciuto una crescita quasi esponenziale di anno in anno. Gli esperti prevedono che il trend continuerà anche in futuro, soprattutto perché piccole e medie imprese, per cui i robot industriali tradizionali sono troppo grandi e costosi da installare, si stanno rendendo conto che i cobot – più piccoli, più flessibili e meno costosi – rappresentano una possibilità per automatizzare i loro processi. Con la maturazione del mercato cobot e la crescente diffusione dei robot leggeri collaborativi, si sta entrando in una nuova fase della rivoluzione dell'automazione industriale: l'attenzione si sta spostando verso le tecnologie di fine braccio EOAT, acronimo di End-of-Arm Tooling, come mezzo per creare valore non solo per gli utenti finali, ma anche per i distributori e gli integratori di sistemi. Con i recenti progressi nelle tecnologie EOAT, si sta passando dai robot collaborativi come componenti chiave nel processo di automazione ad applicazioni collaborative come base per la creazione di valore.

Nell'economia globalizzata di oggi, l'unica strada per competere con i concorrenti di tutto il mondo è automatizzare i processi di più e più velocemente. Che si tratti di una grande azienda manifatturiera o di una piccola realtà con risorse umane ed economiche limitate, i robot leggeri collaborativi possono rappresentare la soluzione, consentendo di automatizzare quei processi che con i robot industriali tradizionali, molto più grandi e costosi, non era possibile automatizzare. Semplici da installare e programmare, i cobot consentono di risparmiare rapidamente sui costi e di aumentare la produttività, con un conseguente vantaggio competitivo per l'azienda.

Tuttavia un braccio robotico è inutile senza uno strumento finale – magari una pinza o un aspiratore – che gli permetta di svolgere il compito per cui è stato acquistato. Ne consegue che anche il mercato globale degli utensili EOAT, vale a dire il mercato delle pinze, dei sensori, dei dispositivi di visione e dei pacchetti software applicativi, sperimenterà una curva di crescita simile nello stesso periodo. L'IFR (International Federation of Robotics) prevede che entro il 2023 saranno in uso circa 1,6 milioni di applicazioni EOAT in tutto il mondo, distribuite in modo approssimativamente uniforme tra applicazioni collaborative e applicazioni robotiche industriali di peso inferiore ai 20 kg.

La crescita spettacolare del mercato EOAT, anche se trainata in parte dall'aumento delle vendite di bracci robotici, sarà ulteriormente stimolata dai progressi

delle tecnologie di fine braccio che hanno contribuito a spostare l'attenzione dai robot stessi verso applicazioni collaborative nuove e innovative. Ed è per questo che si cercherà di introdurre sensori di coppia/forza sempre più sensibili e raffinati per avvicinare la precisione e la sensibilità del robot a quella, per esempio, della mano umana in applicazioni come pick&place evoluto o lavori di controllo qualità e dispositivi di visione artificiale. I sensori di forza sono progettati specificamente per conferire ai bracci robotici il senso del tatto e quando combinati con il giusto strumento EOAT possono essere utilizzati per una serie di applicazioni di automazione industriale finora impensabili, come lucidatura, fresatura, pallettizzazione e così via. Allo stesso modo, i dispositivi di visione artificiale consentono ai bracci di "vedere", aprendo così un nuovo panorama di applicazioni programmabili, in particolare nei settori del controllo qualità, del confezionamento, del pick&place e simili. Le nuove tecnologie EOAT non solo creano l'opportunità di automatizzare di più, ma consentono anche di farlo più velocemente. Anche se usate con le applicazioni già esistenti, eliminano le complesse procedure di programmazione e riducono il tempo necessario per rendere il braccio pienamente operativo ed efficiente. Prendiamo come esempio l'inserimento di connettori. Una semplice pinza attaccata al braccio robotico non è in grado di sentire la posizione del connettore e quindi richiede molto più tempo di programmazione per realizzare un inserimento preciso, con poco o nessun margine di errore e senza il minimo cambiamento nella posizione del connettore. Invece, una pinza che collabora con un sensore di forza può percepire la posizione del connettore e realizzare ogni volta un inserimento rapido, facile e preciso. La combinazione di pinza e sensore, che a sua volta viene fornito con software applicativo preprogrammato, accelera quindi l'automazione e crea valore aggiunto non solo per l'utente finale, ma anche per i distributori e gli integratori di sistemi.

1.5 Tipi di controllo

Scopo degli algoritmi e delle architetture di controllo dei manipolatori è quello di fornire il segnale di comando agli attuatori dei giunti per ottenere che l'organo terminale del manipolatore esegua il compito assegnato nello spazio cartesiano. Schematicamente si possono definire due tipi di compiti:

- Compiti senza interazione con l'ambiente circostante: si tratta cioè di muovere l'organo terminale da un punto ad un altro dello spazio cartesiano o lungo traiettorie assegnate, rispettando vincoli di posizione, velocità ed accelerazione;

- Compiti che richiedono interazione con l'ambiente: si tratta solitamente di muovere nello spazio cartesiano l'organo terminale lungo traiettorie assegnate e contemporaneamente esercitare una forza e/o una coppia su oggetti o superfici definiti nello stesso spazio;

Tipiche tecniche di controllo sono le seguenti:

- Controllo a giunti indipendenti;
- Controllo di coppia calcolata e a dinamica inversa;
- Controllo di forza e controllo ibrido (forza–posizione);
- Controllo adattativo.

1.5.1 Controllo a giunti indipendenti

Il controllo di un robot può essere eseguito attraverso la determinazione delle coppie con cui attuare i giunti per fare eseguire all'organo terminale un compito desiderato. Le caratteristiche meccaniche di un manipolatore hanno un forte impatto sul modo di controllare un robot. Per esempio, i problemi che si incontrano nel controllo di robot cartesiani sono fondamentalmente diversi da quelli che si incontrano nel controllo di robot antropomorfi.

Il controllo di posizione impone di portare il robot in una posizione target (regolazione) o fargli seguire una certa traiettoria. Può essere effettuato sia nello spazio dei giunti che nello spazio cartesiano.

Solitamente si cerca di utilizzare un controllo in spazio dei giunti perchè ha il vantaggio di semplificare la costruzione del controllore. Per contro è necessario portare i punti di riferimento nello spazio dei giunti. Esistono due principali filosofie di controllo nello spazio dei giunti:

- Controllo decentralizzato (o a giunto indipendente): ogni singolo giunto è visto come un sistema SISO e viene controllato indipendentemente dal moto degli altri giunti. L'accoppiamento tra i link viene considerato un disturbo.
- Controllo centralizzato: Si tiene esplicitamente conto dell'accoppiamento tra i giunti e della dinamica non lineare del robot.

L'idea che sta alla base del controllo a giunto indipendente è quella di considerare il robot come formato da n sistemi indipendenti (gli n giunti) e di controllare ciascun giunto come un sistema SISO utilizzando tecniche di controllo lineare. Il

contributo derivante dalla dinamica non lineare del robot è trattato come un disturbo e il controllo deve essere disegnato in modo da attenuarne l'effetto sulla dinamica del sistema. Questa strategia funziona bene quando non c'è accoppiamento diretto tra motore e giunto.

Nel controllo a giunto indipendente ciascun giunto è considerato separatamente. In realtà un manipolatore non è un insieme di n sistemi disaccoppiati ma è un unico sistema multivariabile con n ingressi (le coppie ai giunti). Gli algoritmi di controllo centralizzato considerano il robot nel suo insieme e sfruttano le caratteristiche, anche non lineari, del sistema per ottenere gli obiettivi desiderati. L'approccio centralizzato è più rigoroso di quello decentralizzato e permette di disegnare leggi di controllo non lineari che consentono di ottenere la stabilità globale e l'inseguimento di punti di riferimento desiderati. Le performance che si ottengono con algoritmi di controllo centralizzato sono migliori di quelle che si ottengono con il controllo decentralizzato. Pertanto, il controllo centralizzato diventa indispensabile quando si ha a che fare con applicazioni avanzate.

Un tipo di controllo centralizzato molto semplice è il controllo PD + compensazione di gravità. Questo combina l'azione lineare del controllore PD con un termine di compensazione non lineare e risolve il problema della regolazione, cioè di portare il manipolatore in una configurazione che sia globalmente che asintoticamente stabile. E' utile per compiti in cui è richiesto il posizionamento dell'organo terminale in una ben precisa posizione (es. pick&place).

1.5.2 Controllo di forza e impedenza

Il controllo di forza è legato al fatto che il robot possa interagire con l'ambiente e appunto scambiare forze con esso. I manipolatori interagiscono con l'ambiente modificandone lo stato (pick&place, saldatura, verniciatura...) o applicando forze (assemblaggio, rifinitura, ...). Per svolgere tali compiti è necessario affiancare al controllo della posizione del manipolatore anche il controllo delle forze di interazione che nascono fra la punta e le superfici esterne di contatto. In alcuni casi è sufficiente mantenere tali forze entro limiti che garantiscano il contatto senza ledere la superficie esterna, mentre in altre circostanze l'assegnazione di uno specifico valore alla forza applicata rappresenta una delle specifiche fondamentali del controllo.

L'interazione del manipolatore con l'ambiente implica l'insorgere di:

- Vincoli cinematici, che limitano i gradi di libertà di movimento del robot;
- Forze di contatto;

Esistono pertanto tre famiglie di controlli legati al monitoraggio delle forze: 1) controllo di pura forza, avente come obiettivo l'assegnazione di un valore desiderato di forza, applicata dal manipolatore su una superficie esterna in condizioni stazionarie, tra cui controllo di rigidezza (stiffness control) realizzato mediante retroazione di posizione e velocità; 2) controllo di forza realizzato mediante retroazione di forza, con l'ausilio di sensori di forza; 3) controllo di impedenza, aventi come obiettivo l'assegnazione di un comportamento dinamico desiderato al sistema costituito dal manipolatore e dall'ambiente esterno con cui interagisce; tale comportamento dinamico viene descritto da una opportuna funzione di "impedenza meccanica" desiderata; 4) controllo ibrido, avente come obiettivo l'inseguimento di traiettorie di posizione e l'assegnazione di valori desiderati alla forza applicata lungo differenti gradi di libertà, compatibilmente con i vincoli cinematici imposti dall'ambiente esterno.

Controllo di forza

Il controllo di forza viene applicato quando il manipolatore è in contatto con superfici esterne rigide, che possono essere solitamente ben rappresentate mediante un opportuno coefficiente di rigidezza. L'obiettivo è l'applicazione sulla superficie esterna di un desiderato valore di forza in condizioni stazionarie; esso può essere raggiunto: 1) mediante retroazione di posizione e velocità, 2) mediante retroazione di forza.

Nel caso 1 la forza può essere ottenuta:

- Rappresentando il sistema complessivo, formato dal manipolatore e dall'ambiente esterno, con una molla equivalente e fissandone opportunamente il coefficiente di rigidezza mediante il guadagno di posizione (stiffness control);
- Rappresentando il sistema complessivo con uno smorzatore equivalente e fissandone opportunamente il coefficiente di smorzamento mediante il guadagno di velocità (damping control).

Nel caso 2 vengono utilizzati sensori di forza per realizzare il controllo mediante una retroazione di forza. Tale tipo di controllo soffre però intrinsecamente di problemi di robustezza.

Controllo di impedenza

Il controllo di impedenza è basato sull'assegnazione di un particolare comportamento dinamico al manipolatore in contatto con l'ambiente esterno; tale comportamento viene descritto mediante una impedenza dinamica desiderata, solitamente

corrispondente all'insieme di equazioni differenziali del secondo ordine che rappresentano un sistema massa–molla–smorzatore. L'utilizzo del controllo di impedenza è particolarmente indicato nei casi in cui non sia richiesta una regolazione accurata delle forze applicate, ma sia sufficiente garantire la loro limitatezza. Facendo riferimento alle comuni analogie elettro-meccaniche, che associano la forza meccanica alla tensione elettrica e la velocità alla corrente, il rapporto fra la forza e la velocità di un sistema meccanico ad un grado di libertà (ovvero il rapporto fra la coppia e la velocità angolare) può essere visto come una sorta di impedenza meccanica del sistema stesso, esprimibile nel dominio della frequenza come:

$$Z(s) = \frac{F(s)}{v(s)}$$

dove F(s) e v(s) sono le trasformate di Laplace rispettivamente della forza e della velocità.

L'impedance control utilizzato nei cobots permette di utilizzare il robot come se all'organo terminale fossero applicate delle molle virtuali per permettere al robot si assecondare superfici, attutire urti e, in generale, guadagnare un comportamento collaborativo e interattivo con l'operatore che lo utilizza.

Esistono diversi tipi di controllo di impedenza:

- Controllo di impedenza cartesiano: è modellato su un sistema di ammortizzatori a molla virtuale con valori configurabili per rigidità e smorzamento. Questa molla viene estesa tra le posizioni di riferimento e le posizioni effettive del TCP (Tool Center Point). Ciò consente al robot di reagire in modo conforme alle influenze esterne.

- Controllo di impedenza cartesiano con sovraforza: forma speciale del controllore di impedenza cartesiana. Oltre al comportamento conforme, è possibile sovrapporre punti di riferimento di forza costanti e oscillazioni di forza sinusoidali. Questo controller può essere utilizzato, ad esempio, per implementare percorsi di ricerca dipendenti dalla forza e movimenti di vibrazione per i processi di accoppiamento.

- Controllore di impedenza specifico per asse: è modellato su uno smorzatore a molla virtuale. I valori di rigidità e smorzamento possono essere configurati per ciascun asse.

Per studiare il controllo di impedenza si utilizzano, a livello matematico, le equazioni di Lagrange e il modello dinamico del robot può essere scritto nel se-

guente modo:

$$M(q)\ddot{q} + S(q,\dot{q})\dot{q} + g(q) = u + J^T(q)F$$

dove q = coordinata generalizzata, M = matrice di inerzia, S = matrice di smorzamento, g = matrice di rigidezza, u = ingresso di controllo, J = Jacobiano geometrico, F = vettore delle forze lineari e coppie angolari esterne applicate.

Questa equazione del modello dinamico può essere riscritta sia in coordinate cartesiane che in coordinate dei giunti. Per questa attività si utilizza il controllo su coordinate cartesiane.

In coordinate cartesiane si ha:

$$M_x(q)\ddot{x} + S_x(q,\dot{q})\dot{x} + g_x(q) = J_a^{-T}(q)u + F_a$$

dove $J_a(q)$ è lo Jacobiano analitico calcolato per la coordinata cartesiana lungo la quale si vuole analizzare il sistema, per esempio x.

$$M_x(q) = J_a^{-T}(q)M(q)J_a^{-1}(q)$$

$$S_x(q,\dot{q}) = J_a^{-T}(q)S(q,\dot{q})J_a^{-1}(q) - M_x(q)\dot{J}_a(q)J_a^{-1}(q)$$

$$g_x(q) = J_a^{-T}(q)g(q)$$

Il modello dinamico cartesiano del robot è parametrizzato linearmente in termini di un insieme di coefficienti dinamici progettati in due fasi:

1. linearizzazione del feedback nello spazio cartesiano (con misura della forza)

$$u = J_a^T(q)[M_x(q)a + S_x(q,\dot{q})\dot{x} + g_x(q) - F_a]$$

   $\ddot{x}$ = a sistema in catena chiusa;

2. imposizione di un modello di impedenza dinamica

$$M_m(\ddot{x} - \ddot{x}_d) + D_m(\dot{x} - \dot{x}_d) + K_m(x - x_d) = F_a$$

   dove M_m matrice di inerzia desiderata, D_m matrice di smorzamento desiderata, K_m matrcie di rigidezza desiderata, F_a vettore forze esterne applicate, che può essere realizzato scegliendo

$$a = \ddot{x}_d + M_m^{-1}[D_m(\dot{x}_d - \dot{x}) + K_m(x_d - x) + F_a]$$

x_d è il movimento desiderato.

Il controllo di impedenza può essere utilizzato per: 1) evitare grandi forze di impatto dovute a incerte caratteristiche geometriche (posizione, orientamento) dell'ambiente, 2) adattarsi alle caratteristiche dinamiche (in particolare, rigidità) dell'ambiente, in modo complementare, 3) imitare il comportamento di un braccio umano (veloce e rigido in movimento libero, lento e conforme in movimento "protetto").

Per ognuno di queste utilizzi vengono impostati i parametri del modello dinamico in modo diverso: 1) $M_{m,i}$ grande e $K_{m,i}$ piccola in direzioni cartesiane in cui il contatto è previsto (forze di contatto basse), 2) $M_{m,i}$ piccola e $K_{m,i}$ grande in direzioni cartesiane che dovrebbero essere libere (buon tracciamento della traiettoria di movimento desiderata), 3) i coefficienti di smorzamento $D_{m,i}$ vengono utilizzati per modellare i comportamenti transitori.

1.5.3 Controllo ibrido

Con il tempo si è passati dall'utilizzo dei singoli controlli a sistemi più evoluti basati sulla creazione di ibridi, come il controllo combinato posizione-forza. Questi controlli ibridi vengono utilizzati in molte applicazioni industriali per avere maggior controllo e informazioni sullo stato del robot; esempi di lavorazioni industriali sono assemblaggio, rifinitura, lucidatura. Quindi la richiesta di movimenti controllati con l'applicazione di forze specifiche porta alle soluzioni ibride.

Capitolo 2

Setup sperimentale

In questa sezione si andrà a descrivere il sistema di Kuka, composto da robot, controllore e software. Un sistema robotizzato (Figura 2.1) comprende tutti gli assiemi di un robot industriale, inclusi il manipolatore (sistema meccanico e installazioni elettriche), il controller, i cavi di collegamento, l'utensile e altre apparecchiature.

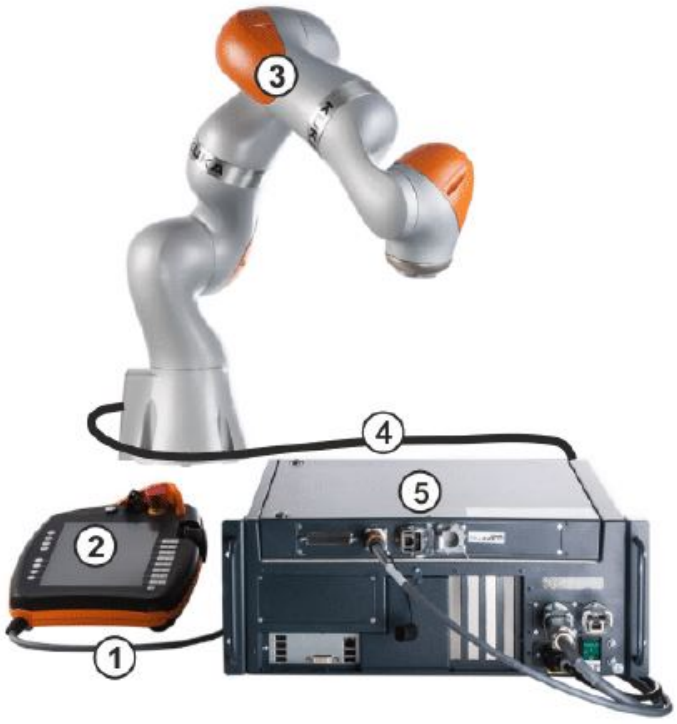


Figura 2.1: Panoramica del sistema robot

Si possono notare:

1. Cavo di collegamento allo smartPAD
2. Pannello di controllo Kuka smartpad
3. Manipolatore
4. Cavo di collegamento al controller robot KUKA Sunrise Cabinet
5. Controllore Kuka Sunrise Cabinet

2.1 Analisi robot

Il Kuka lbr iiwa 14 820R è un braccio robotico industriale leggero a sette assi. In paticolare LBR è l'acronimo di "robot leggero", mentre iiwa sta per "intelligent industrial work assistant". Ciascuno dei suoi giunti è dotato di sensori di coppia e di un sensore di posizione. I dati dei sensori consentono l'uso del controllo di impedenza oltre del controllo di posizione, rendendo così possibile implementare comportamenti collaborativi. Misurazioni estremamente accurate, con intervalli di aggiornamento inferiori al millisecondo, consentono al robot di reagire molto rapidamente alle forze di processo e lo rendono particolarmente adatto all'interazione con l'uomo. Il KUKA iiwa può essere programmato per una varietà di compiti tramite la "tecnologia di controllo Sunrise KUKA". Questo comprende il software di controllo "KUKA Sunrise OS" che può eseguire programmi in JAVA come linguaggio di programmazione sull'hardware di controllo "KUKA Sunrise Cabinet". Sebbene Java sia un linguaggio flessibile e comune, è necessaria una conoscenza approfondita del sistema Sunrise per programmare il robot e utilizzarne le funzionalità. Le attività che verrano presentate all'interno di questo elaborato sono svariate e con l'obiettivo di analizzare vari settori del robot Kuka.

L'area di lavoro è rappresentata dalla seguente Figura 2.2.

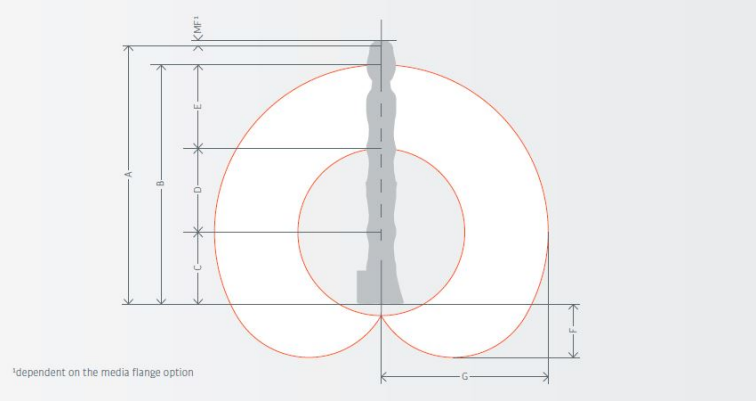


Figura 2.2: Spazio di lavoro

Le dimensioni del robot e dello spazio di lavoro vengono qui di seguito riportate:

Tabella 2.1: Struttura fisica del robot

Dim. A	Dim. B	Dim. C	Dim. D	Dim. E	Dim. F	Dim. G
1306 *mm*	1180 *mm*	360 *mm*	420 *mm*	400 *mm*	255 *mm*	820 *mm*

Un aspetto importante, che serve a definire il limite dell'area di lavoro, è ovviamente il limite fisico-strutturale che il robot ha legato alla sua costruzione. Ogni giunto ha dei limiti che determinano, tutti insieme, l'area di lavoro.

Tabella 2.2: Parametri di movimento

N. asse	Range di movimento	Coppia massima	Velocità massima
1	$\pm 170\ [gradi]$	$320\ [Nm]$	$85\ [gradi/s]$
2	$\pm 120\ [gradi]$	$320\ [Nm]$	$85\ [gradi/s]$
3	$\pm 170\ [gradi]$	$176\ [Nm]$	$100\ [gradi/s]$
4	$\pm 120\ [gradi]$	$176\ [Nm]$	$75\ [gradi/s]$
5	$\pm 170\ [gradi]$	$110\ [Nm]$	$130\ [gradi/s]$
6	$\pm 120\ [gradi]$	$40\ [Nm]$	$135\ [gradi/s]$
7	$\pm 175\ [gradi]$	$40\ [Nm]$	$135\ [gradi/s]$

Degna di particolare attenzione è la capacità del cobot, avendo 7 gradi di libertà, di poter tenere fisso l'organo terminale e variare la sua configurazione, quello che viene definito come movimento in uno spazio nullo (Figura 2.3). Ciò significa che teoricamente può spostarsi su ogni punto dell'area di lavoro con un numero infinito di configurazioni dell'asse. A causa della ridondanza cinematica, durante il movimento cartesiano si può effettuare un cosiddetto movimento nullo dello spazio. Nel

movimento dello spazio nullo, gli assi vengono ruotati in modo tale che la posizione e l'orientamento del set TCP (Tool Center Point) vengano mantenuti durante il movimento.

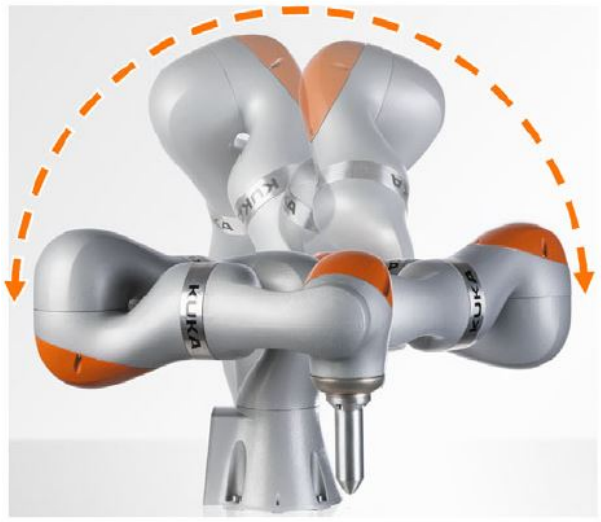


Figura 2.3: Movimento spazio vuoto

Il movimento dello spazio nullo viene eseguito tramite il "gomito" del braccio del robot. La posizione del gomito è definita dall'angolo del gomito (R). La posizione dell'angolo del gomito (R) può essere modificata usando i tasti jog durante il movimento cartesiano.

La configurazione ottimale dell'asse può essere impostata per una data posizione e orientamento del TCP. Ciò è particolarmente utile in uno spazio di lavoro limitato. Quando viene raggiunto un finecorsa software, è possibile tentare di spostare il robot fuori dalla gamma dei finecorsa modificando l'angolo del gomito.

2.1.1 Sistemi di coordinate

Sistemi di coordinate o frame determinano la posizione e l'orientamento di un oggetto nello spazio. I seguenti sistemi di coordinate sono rilevanti per il controller del robot: world, robot base, base, flangia e tool.

Sistema di coordinate world

Il sistema di coordinate world è un sistema di coordinate cartesiane definito in modo permanente. È il sistema di coordinate radice per tutti gli altri sistemi di coordinate, in particolare per i sistemi di coordinate di base e il sistema di coordinate di base del robot. Per impostazione predefinita, il sistema di coordinate world si trova nella base del robot.

Sistema di coordinate robot base

Il sistema di coordinate di base del robot è un sistema di coordinate cartesiane, che si trova sempre alla base del robot. Definisce la posizione del robot rispetto al sistema di coordinate world. Per impostazione predefinita, il sistema di coordinate di base del robot è identico al sistema di coordinate world. È possibile definire una rotazione del robot relativa al sistema di coordinate world modificando l'orientamento di montaggio in Sunrise.Workbench. Di default, l'orientamento di montaggio del robot a pavimento è impostato (A = 0 gradi, B = 0 gradi, C = 0 gradi).

Sistema di coordinate base

Per definire i movimenti nello spazio cartesiano, è necessario specificare un sistema di coordinate di riferimento (base). Come standard, il sistema di coordinate world viene utilizzato come sistema di coordinate di base per un movimento. Ulteriori sistemi di coordinate di base possono essere definiti in relazione al sistema di coordinate world.

Sistema di coordinate flangia

Il sistema di coordinate della flangia descrive la posizione corrente e l'orientamento del punto centrale della flangia del robot. Non ha una posizione fissa e viene spostato con il robot. Il sistema di coordinate della flangia viene utilizzato come origine per i sistemi di coordinate che descrivono gli utensili montati sulla flangia.

Sistema di coordinate utensile

Il sistema di coordinate dell'utensile è un sistema di coordinate cartesiane che si trova nel punto di lavoro dello strumento montato. Questo è chiamato TCP (Tool Center Point). Qualsiasi numero di frame può essere definito per uno strumento e può essere selezionato come TCP. L'origine del sistema di coordinate dell'utensile è generalmente identica al sistema di coordinate della flangia. Il sistema di coordinate dell'utensile è sfalsato dal punto centrale dell'utensile dall'utente.

2.2 Analisi software

2.2.1 Panoramica Kuka Sunrise.OS

KUKA Sunrise.OS è un pacchetto software di sistema per robot industriali in cui le attività di programmazione e controllo dell'operatore sono strettamente separate le une dalle altre. Le applicazioni per il robot sono programmate con Kuka Sunrise.Workbench; una stazione robot opera utilizzando il pannello di controllo Kuka Smartpad, è composta da un controller robot, un manipolatore e altri dispositivi e può eseguire più applicazioni (task) (Figura 2.4).

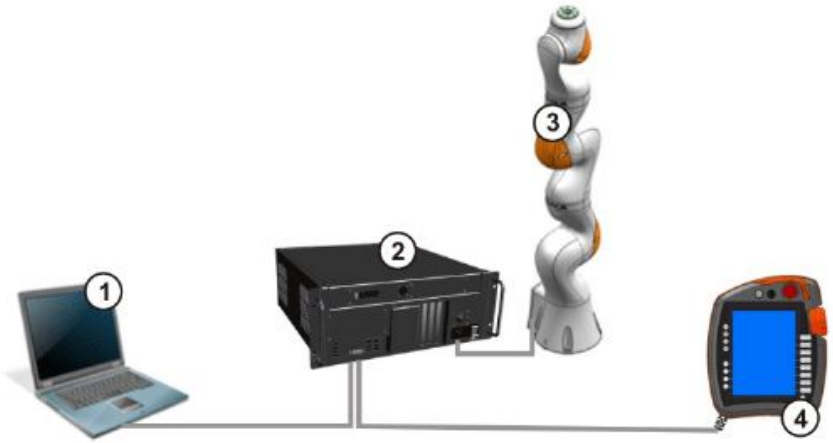


Figura 2.4: Separazione tra controllo operatore (Smartpad) e programmazione

1. Computer con KUKA Sunrise.Workbench (connessione tramite il KLI (KUKA Line Interface) del controller del robot)
2. Controller robot KUKA Sunrise Cabinet
3. Manipolatore
4. Smartpad

KUKA Sunrise.Workbench è lo strumento per l'avvio di una stazione e lo sviluppo di applicazioni robotiche. WorkVisual viene utilizzato per la configurazione e la mappatura del bus. Lo SmartPAD è richiesto solo nella fase di avvio dell'attività, che per motivi pratici o di sicurezza non possono essere eseguite utilizzando KUKA Sunrise.Workbench. Lo smartpad viene utilizzato ad esempio per padroneggiare assi, strumenti di calibrazione e punti di insegnamento. Dopo l'avvio e lo sviluppo

di applicazioni, l'operatore può eseguire semplici operazioni di manutenzione e sequenze operative utilizzando smartPAD. L'operatore non può modificare la stazione e la configurazione di sicurezza durante la programmazione.

2.2.2 Panoramica Kuka Sunrise.Workbench

KUKA Sunrise.Workbench è l'ambiente di sviluppo per la cella robotizzata (stazione). Offre le seguenti funzionalità per l'avvio e lo sviluppo di applicazioni. Per l'avvio si ha: installazione software di sistema, configurazione della cella robotizzata, modifica della configurazione di sicurezza, creazione configurazione I/O, trasferimento del progetto sul controller del robot. Per lo sviluppo delle applicazioni si ha: programmazione di applicazioni robot in Java, gestione di progetti e programmi, modifica e gestione dei dati di runtime, sincronizzazione di progetti e debugging remoto.

2.3 Analisi Smartpad

Lo smartpad è il pannello di controllo portatile per il robot industriale, raffigurato in Figura 2.5 e 2.6. Esso ha tutte le funzioni di controllo e visualizzazione dell'operatore necessarie per il funzionamento. E' provvisto di un display touchscreen: la smartHMI può essere azionata con un dito o una stilo. Non è necessario un mouse esterno o una tastiera esterna.

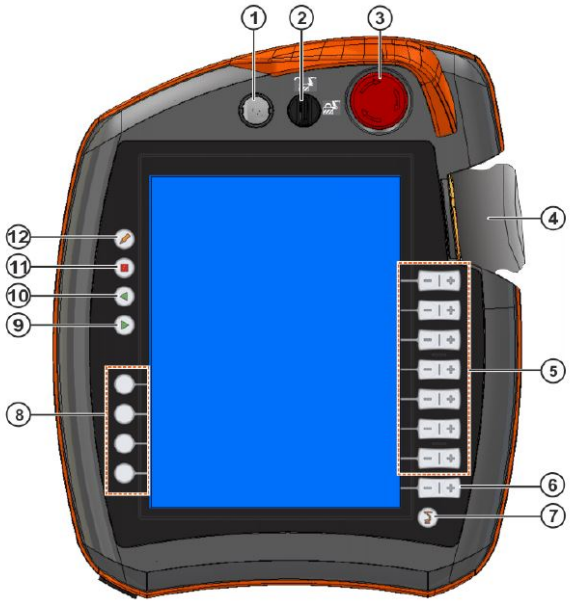


Figura 2.5: Kuka smartpad, vista frontale

1. Pulsante per disconnettere lo smartpad;
2. Interruttore a chiave: la gestione della connessione viene richiamata mediante l'interruttore a chiave. L'interruttore può essere ruotato solo se la chiave è inserita. La gestione della connessione serve per cambiare modalità operativa;
3. Pulsante Arresto di emergenza: il robot può essere fermato in situazioni pericolose utilizzando il dispositivo di arresto di emergenza. Il dispositivo di arresto di emergenza si blocca in posizione quando viene premuto;
4. Space mouse: non ha funzioni rilevanti;
5. Tasti jog: utilizzati per spostare manualmente il robot;
6. Tasto per impostare l'override;
7. Tasto del menu principale: il tasto del menu principale mostra e nasconde il menu principale su smartHMI;
8. Tasti utente: la funzione dei tasti utente è liberamente programmabile. L'uso delle chiavi utente include il controllo dei dispositivi periferici o l'attivazione di azioni specifiche dell'applicazione;
9. Tasto Start: il tasto Start viene utilizzato per avviare un programma. Il tasto Start è anche usato per indirizzare manualmente i frame e per riportare il robot sul percorso;
10. Tasto Restart: non ha funzioni rilevanti;
11. Tasto Stop: mette in pausa un programma in esecuzione;
12. Tasto della tastiera: non ha funzioni rilevanti.

Figura 2.6: Kuka smartpad, vista posteriore

1. Interruttore di abilitazione: ha 3 posizioni: a) non premuto, b) posizione centrale, c) completamente premuto (posizione di panico). L'interruttore di abilitazione deve essere tenuto in posizione centrale nelle modalità operative T1, T2 e CRR per poter fare jogging sul manipolatore. Di default, l'interruttore non ha funzione nella modalità automatica;

2. Tasto start (verde): il tasto Start viene utilizzato per avviare un programma. Il tasto Start è anche usato per indirizzare manualmente i frame e per riportare il robot sul percorso;

3. Interruttore di abilitazione;

4. Connessione USB: viene utilizzata per esempio per l'archiviazione dei dati;

5. Interruttore di abilitazione;

6. Targhetta di identificazione.

Capitolo 3

Analisi cinematica

3.1 Cinematica diretta

I manipolatori antropomorfi a sette gradi di libertà hanno una cinematica S-R-S (sferica-rotazionale-sferica). Per cominciare si considera la disposizione dei sistemi di riferimento: il sistema di riferimento assoluto è posizionato alla base del robot e presenta una configurazione, rispetto al foglio, con x positiva verso destra, y positiva verso l'interno del foglio, z positiva verso l'alto; il sistema di riferimento dell'organo terminale ha la stessa notazione anche se dipende da come è posizionato quest'ultimo, ovvero se è posto in alto allineato con la base del robot o è rovesciato in posizione di utilizzo. Per la classificazione delle rotazioni il robot presenta una notazione in angoli di Cardano (quindi prima rotazione attorno ad asse z, poi attorno al nuovo asse y, infine attorno al nuovo asse x). Un possibile insieme di parametri di Denavit-Hartenberg (DH) che descrivono la catena cinematica di un manipolatore seriale di questo tipo sono elencati nella tabella 3.1.

Tabella 3.1: Parametri Denavit-Hartenberg di un manipolatore antropomorfo a 7 gradi di libertà

i	a_i	α_i	d_i	θ_i
1	0	-$\pi/2$	d_{bs}	θ_1
2	0	$\pi/2$	0	θ_2
3	0	$\pi/2$	d_{se}	θ_3
4	0	-$\pi/2$	0	θ_4
5	0	-$\pi/2$	d_{ew}	θ_5
6	0	$\pi/2$	0	θ_6
7	0	0	d_{wf}	θ_7

Ogni i rappresenta la variabile di giunto i-esimo, che è fisicamente limitata dal limite superiore meccanico $\theta_{i,s}$ e dal limite inferiore $\theta_{i,i}$, con $i \in [1,7]$. La colonna d_i descrive le lunghezze del collegamento cinematico tra: (b)base, (e)gomito, (w)polso e (f)flangia (Figura 3.1).

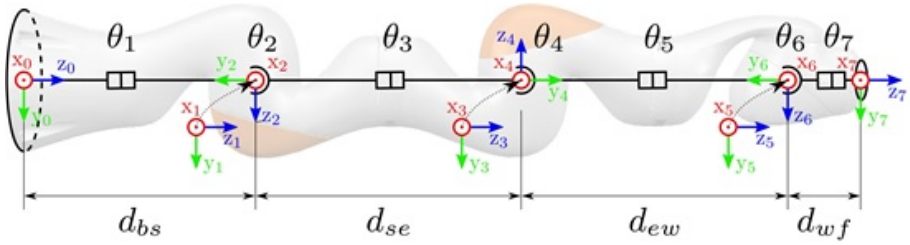


Figura 3.1: Struttura generica manipolatore

Il problema della cinematica diretta è facilmente risolvibile una volta determinati i parametri DH. A ciascun giunto sono assegnati quattro parametri, che sono convertiti in una matrice di trasformazione che stabilisce la relazione tra uno assegnato riferimento (i-1) e il suo successivo (i).

$$T_{i,i-1} = \begin{bmatrix} cos(\theta_i) & -sen(\theta_i)cos(\alpha_i) & sen(\theta_i)sen(\alpha_i) & a_i cos(\theta_i) \\ sen(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sen(\alpha_i) & a_i sen(\theta_i) \\ 0 & sen(\alpha_i) & cos(\alpha_i) & d\_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Dal prodotto di queste matrici dalla base alla flangia, ritorna il posizionamento del manipolatore nello spazio di lavoro.

A livello di implementazione software, tramite Matlab si è riusciti a riprodurre la struttura semplificata del robot, come si può vedere dalla seguente Figura 3.2. La terna di vettori colorati, asse x rosso, asse y verde e asse z blu indicano come variano i sistemi di riferimento in ogni giunto, data dalla soluzione della matrice DH.

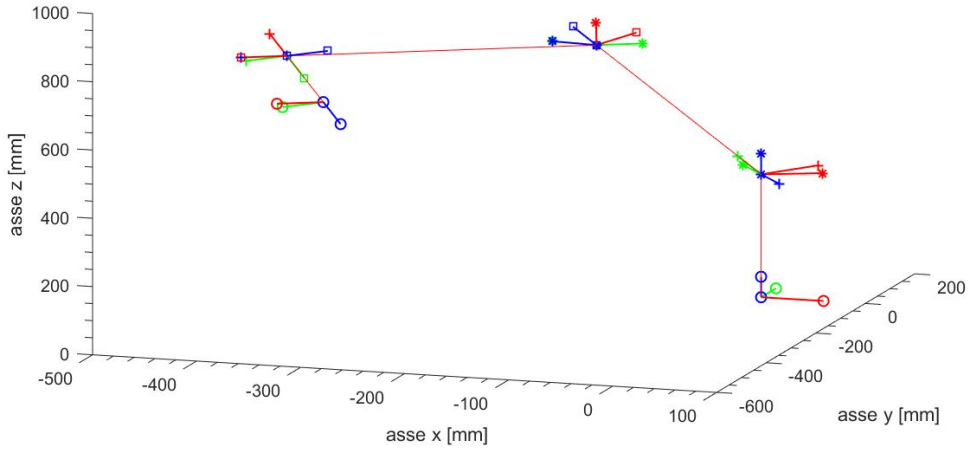


Figura 3.2: Modello della soluzione della cinemtica diretta del robot

3.2 Parametri di auto-movimento

Per affrontare i numerosi casi di auto-movimento globali e locali, nel calcolo della cinematica inversa vengono introdotti due parametri aggiuntivi. Il primo parametro è definito Configurazione Globale (GC) e specifica in modo univoco il ramo di soluzioni cinematiche per una data configurazione, il secondo è Angolo del Braccio (ψ) che indica la posizione del gomito nel cerchio di ridondanza, ovvero la circonferernza che il robot, muovendo il gomito, può eseguire durante lo NullSpace motion. Entrambi GC e ψ sono determinati direttamente nel problema della cinematica diretta dalle variabili di giunto e sono passati come parametri all'algoritmo di cinematica inversa. Il parametro GC_k della Configurazione globale è suddiviso in 3 variabili che esprimono il segno delle coordinate di giunto a livello della spalla (GC_2), del gomito (GC_4) e del polso (GC_6). Queste variabili esercitano successivamente il controllo sulla disposizione del manipolatore nello spazio. Il GC_k è dato da

$$GC_k = \begin{cases} 1 & \text{se} \quad \theta_k \geq 0 \\ -1 & \text{se} \quad \theta_k < 0 \end{cases} \tag{3.2}$$

$\forall k \in [2,4,6]$.

La disposizione del manipolatore nello spazio è direttamente gestita dal GC_k sulla spalla, sul gomito e sul polso. L'angolo del braccio rappresenta l'angolo formato dal piano spalla-gomito-polso (SEW) e il piano di riferimento (SE_vW), come

mostrato nella Figura 3.3. La ridondanza del gomito dipende esclusivamente dalla struttura del manipolatore e attiva il vettore del polso-spalla. Il centro del cerchio si trova a metà della distanza lungo la linea retta che collega la spalla al polso, e la sua curva è inscritta nel piano definito dal vettore spalla-polso.

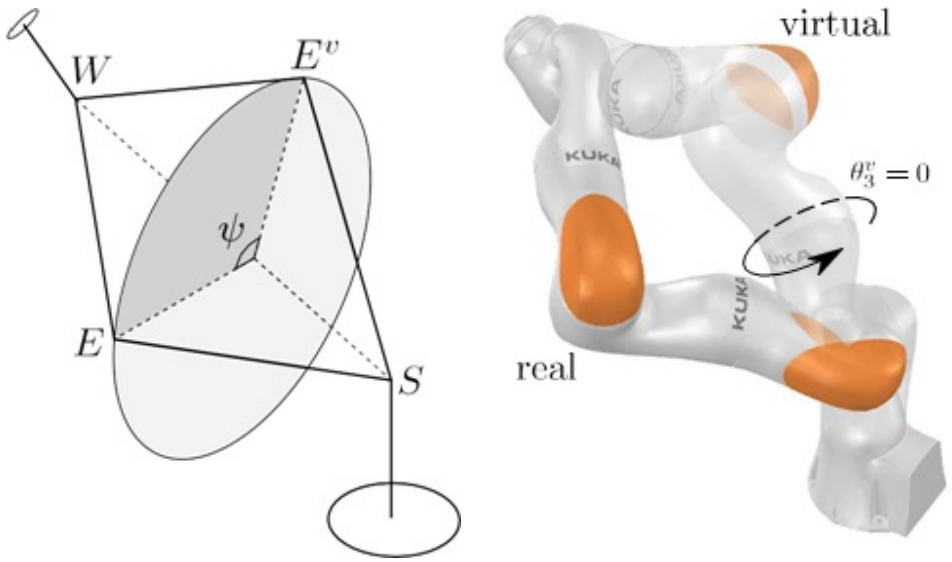


Figura 3.3: (a) Rappresentazione dell'angolo del braccio (ψ), come l'angolo tra il reale (E) e il gomito virtuale (E_v) nel cerchio di ridondanza, (b) Configurazione del manipolatore reale e virtuale nella stessa posa

La selezione del piano di riferimento non è semplice ed è stata oggetto di discussione in diversi lavori. Shimizu e altri (vedi [2]) hanno introdotto una soluzione alternativa alla singolarità algoritmica definendo il piano di riferimento basato su un manipolatore virtuale, che è un'immagine non ridondante del manipolatore 7-DoF originale, con $\theta_3 = 0$. Se non vengono imposti limiti comuni al manipolatore virtuale, a condizione che un bersaglio si trovi nel suo spazio di lavoro, è possibile trovare un insieme di soluzioni per il problema della cinematica inversa. Queste soluzioni dipendono dalla configurazione del manipolatore virtuale, che non è stato affrontato nel documento di riferimento. Per sviluppare un algoritmo che risolva l'auto-movimento della configurazione globale insieme alla ridondanza del gomito bisogna considerare che qualsiasi funzione nello spazio di lavoro dovrebbe essere continuamente deformata in una funzione del parametro per uno specifico ramo di configurazione globale.

3.3 Ridondanza gomito

L'angolo del braccio, ψ, è l'angolo tra i vettori normali al piano reale *SEW* del manipolatore e quello virtuale SE_vW del manipolatore (non ridondante). C'è tuttavia un piccolo avvertimento alla dichiarazione, da allora lo stesso piano può essere definito sia dal suo vettore che dal suo vettore negativo. Questa indeterminazione è rilevante per il nostro problema perché la definizione del vettore del piano *SEW* di riferimento dipende dalla configurazione globale del manipolatore virtuale, indipendentemente che il gomito sia verso l'alto o verso il basso. Per determinare univocamente un vettore di piano di riferimento, si associa la variabile di configurazione del robot GC al calcolo del piano di riferimento. È facile dimostrare che le posizioni della spalla e del polso sono comuni al robot reale e virtuale per la stessa posa target (vedi Figura 3.3). Pertanto, si ha solo bisogno della posizione del gomito del manipolatore virtuale per trovare il piano di riferimento e di conseguenza calcolare ψ. Le espressioni cinematiche inverse per il manipolatore virtuale 6-DoF vengono applicate per scoprire la sua posizione del gomito. Le posizioni dei giunti del manipolatore virtuale saranno rappresentate da un superindice $\theta_{i,v}$. La matrice target sarà rappresentata dalla matrice di trasformazione $T_{7,0}$, che combina la posizione $p_{7,0} \in \Re_3$ e la rotazione $R_{7,0} \in SO(3)$. Per semplificare la notazione delle seguenti equazioni, si elencano i vettori da: base a spalla ($p_{2,0}$), spalla a gomito ($p_{4,2}$), gomito a polso ($p_{6,4}$) e polso a flangia ($p_{7,6}$) in base ai parametri DH:

$$p_{2,0} = \begin{bmatrix} 0 & 0 & d_{bs} \end{bmatrix}^T$$

$$p_{4,2} = \begin{bmatrix} 0 & 0 & d_{se} \end{bmatrix}^T$$

$$p_{6,4} = \begin{bmatrix} 0 & 0 & d_{ew} \end{bmatrix}^T$$

$$p_{7,6} = \begin{bmatrix} 0 & 0 & d_{wf} \end{bmatrix}^T$$

L'articolazione del gomito virtuale ($\theta_{4,v}$) è il primo da calcolare, poiché dipende solo dalla struttura cinematica del manipolatore e dal vettore del polso e della spalla. Il vettore spalla-polso è calcolato da:

$$p_{6,2} = p_{7,0} - p_{2,0} - (R_{7,0} * p_{7,6}) \tag{3.3}$$

e $\theta_{4,v}$ è calcolato usando la legge dei coseni:

$$\theta_{4,v} = GC_4 arcos\left(\frac{(p_{6,2})^2 - (d_{se})^2 - (d_{ew})^2}{2 * d_{se} * d_{ew}}\right) \tag{3.4}$$

La variabile di configurazione del gomito (GC_4) - il segnale del quarto giunto - è necessaria per definire univocamente il piano di riferimento. Poiché $\theta_{3,v}$ è 0, il vettore spalla-gomito ($p_{4,2}$), così come il vettore del gomito-polso ($p_{6,4}$) sono allineati nel piano xy. L'articolazione $\theta_{1,v}$ è quindi responsabile del trasferimento del braccio virtuale in coordinate x-y della posizione del polso.

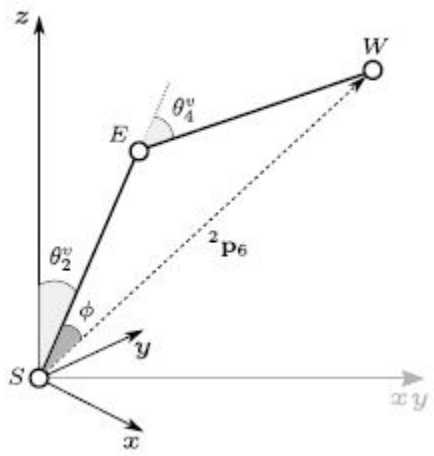


Figura 3.4: Rappresentazione del manipolatore virtuale e delle variabili richieste per il calcolo della $\theta_{2,v}$

Tuttavia, se il vettore spalla-polso ($p_{6,2}$) è allineato all'asse z del giunto 1 ($R_{1,0,z}$), l'articolazione 1 non viene più definita e si verifica una singolarità algoritmica. Quindi, il calcolo di $\theta_{1,v}$ è:

$$\theta_{1,v} = \begin{cases} atan2(p_{6,2,y}, p_{6,2,x}) & \text{se} \quad (p_{6,2})^2 \times R_{1,0,z} > 0 \\ 0 & \text{se} \quad (p_{6,2})^2 \times R_{1,0,z} = 0 \end{cases} \tag{3.5}$$

L'articolazione della spalla virtuale $\theta_{2,v}$ è l'ultima variabile mancante per la posizione del gomito virtuale. Come possiamo vedere in Figura 3.4 ϕ può essere calcolato con il teorema del coseno:

$$\phi = arcos\left(\frac{(p_{6,2})^2 + (d_{se})^2 - (d_{ew})^2}{2 * d_{se} * p_{6,2}}\right) \tag{3.6}$$

e il valore di $\theta_{2,v}$, che dipende dalla configurazione del gomito, è:

$$\theta_{2,v} = atan2\left(\sqrt{(p_{6,2,y})^2 + (p_{6,2,x})^2}, p_{6,2,z}\right) + GC_4\phi \tag{3.7}$$

Con $\theta_{1,v}$, $\theta_{2,v}$, $\theta_{3,v}$ e $\theta_{4,v}$ si può calcolare la matrice di trasformazione da base a gomito $T_{7,4,v}$ usando la cinematica diretta. Il vettore normale al piano $(v_{sew})^v$ viene ora calcolato come prodotto incrociato dei vettori unitari che collegano spalla-gomito e polso-spalla:

$$(v_{sew})^v = \left(\frac{p_{4,0,v} - p_{2,0,v}}{\| p_{4,0,v} - p_{2,0,v} \|} \right) \times \left(\frac{p_{6,0,v} - p_{2,0,v}}{\| p_{6,0,v} - p_{2,0,v} \|} \right) \tag{3.8}$$

Il vettore normale al piano *SEW* del braccio reale (v_{sew}) viene calcolato utilizzando la stessa formula con la posizione del gomito reale:

$$v_{sew} = \left(\frac{p_{4,0} - p_{2,0}}{\| p_{4,0} - p_{2,0} \|} \right) \times \left(\frac{p_{6,0} - p_{2,0}}{\| p_{6,0} - p_{2,0} \|} \right) \tag{3.9}$$

Sia $\psi \in [-\pi, \pi]$ e in particolare ψ sia 0 sul piano di riferimento. Il segno del parametro dell'angolo del braccio è determinato come:

$$sg_{\psi} = sgn\left[(v_{sew} \times (v_{sew})^v) * p_{6,2} \right] \tag{3.10}$$

e ψ come:

$$\psi = sg_{\psi} arcos(v_{sew} \times (v_{sew})^v) \tag{3.11}$$

Dato che l'angolo del braccio, ψ, è ora determinato, insieme alla configurazione globle (*GC*) e alla posa del robot finale ($T_{7,0}$) esiste una descrizione completa della configurazione del robot nello spazio di lavoro, che si associa a una configurazione del manipolatore univoca nello spazio dei giunti.

3.4 Cinematica inversa

L'algoritmo cinematico inverso descritto si basa sull'approccio standard di divisione del manipolatore nel braccio superiore e inferiore, rispettivamente responsabili del posizionamento e dell'orientamento. Allo stesso modo in cui la cinematica diretta calcola la posa finale $T_{7,0}$ e i parametri di auto-movimento (ψ, *GC*) dalle posizioni dei giunti, reciprocamente la cinematica inversa richiede che queste tre variabili generino un insieme di posizioni dei giunti. Il primo passo consiste nel determinare il vettore spalla-polso $p_{6,2}$. La stessa equazione utilizzata per il manipolatore virtuale (3.3) può essere applicata per il robot reale, poiché le posizioni delle spalle e del polso sono coincidenti. Inoltre, poiché il vettore da spalla a polso è lo stesso, anche la variabile di giunto del gomito θ_4 può essere calcolata da (3.4). θ_4 è l'unico angolo che non dipende da ψ. Per determinare gli altri angoli di giunto bisogna trovare la posizione del vero gomito del manipolatore. Per fare ciò, per

prima cosa si calcola la posa virtuale del gomito sul piano di riferimento ($T_{4,0,v}$), seguendo l'algoritmo descritto nella sezione 3.3. La vera posa del gomito non è altro che la posa virtuale del gomito, ruotata attorno all'asse della spalla-polso ($p_{6,2}$) di ψ:

$$R_{4,0} = R_{\psi,0} R_{4,0,v} \tag{3.12}$$

che è equivalente a:

$$R_{3,0} = R_{\psi,0} R_{3,0,v} \tag{3.13}$$

perchè θ_4 è lo stesso per la configurazione reale e virtuale, quindi $R_{4,3,v} = R_{4,3}$. La matrice di rotazione di ridondanza del gomito ($R_{\psi,0}$) codifica la rotazione dell'angolo attorno al vettore di spalla-polso $p_{6,2}$, Figura 3.3a. Viene calcolata utilizzando la formula di rotazione di Rodrigues in notazione matriciale,

$$R_{\psi,0} = I_3 + sen(\psi) \left[p_{6,2}\times\right] + (1 - cos(\psi)) \left[p_{6,2}\times\right]^2 \tag{3.14}$$

dove $p_{6,2}\times$ è il prodotto vettoriale vettore per versore $p_{6,2}\wedge$.

Sostituendo (3.14) in (3.13) e seguendo la notazione di Shimizu, $R_{3,0}$ può essere ottenuta in funzione di 3 matrici ausiliarie A_s, B_s, C_s,

$$R_{3,0} = A_s sen(\psi) + B_s cos(\psi) + C_s \tag{3.15}$$

dove

$$A_s = \left[p_{6,2}\times\right] R_{3,0,v} \tag{3.16}$$

$$B_s = -\left[p_{6,2}\times\right]^2 R_{3,0,v} \tag{3.17}$$

$$C_s = \left[p_{6,2} \wedge (p_{6,2}\wedge)^T\right]^2 R_{3,0,v} \tag{3.18}$$

I valori reali di θ_1, θ_2 e θ_3 sono ora determinati analiticamente combinando gli elementi della matrice $R_{3,0}$ (che contiene θ_1, θ_2 e θ_3), derivata dai parametri della tabella DH,

$$R_{3,0} = \begin{bmatrix} * & cos(\theta_1)sen(\theta_2) & * \\ * & sen(\theta_1)sen(\theta_2) & * \\ -sen(\theta_2)cos(\theta_3) & cos(\theta_2) & -sen(\theta_2)sen(\theta_3) \end{bmatrix} \tag{3.19}$$

Il simbolo $*$ indica elementi omessi della matrice, che non sono richiesti per i calcoli di posizione dei giunti. E' importante notare come questi angoli di giunto sono soggetti al parametro di configurazione globale GC:

$$\theta_1 = atan2[GC_2(a_{s,2,2}sen(\psi) + b_{s,2,2}cos(\psi) + c_{s,2,2}), \\ GC_2(a_{s,1,2}sen(\psi) + b_{s,1,2}cos(\psi) + c_{s,1,2})] \quad (3.20)$$

$$\theta_2 = GC_2 arcos(a_{s,3,2}sen(\psi) + b_{s,3,2}cos(\psi) + c_{s,3,2}) \quad (3.21)$$

$$\theta_3 = atan2[GC_2(-a_{s,3,3}sen(\psi) - b_{s,3,3}cos(\psi) - c_{s,3,3}), \\ GC_2(-a_{s,3,1}sen(\psi) - b_{s,3,1}cos(\psi) - c_{s,3,1})] \quad (3.22)$$

Una volta che si conosce la matrice di rotazione relativa alle articolazioni della spalla ($R_{3,0}$), è semplice calcolare la matrice di rotazione delle articolazioni del polso ($R_{7,4}$).

$$R_{7,4} = A_w sen(\psi) + B_w cos(\psi) + C_w \quad (3.23)$$

dove

$$A_w = R_{4,3}^T A_s^T R_{7,0} \quad (3.24)$$

$$B_w = R_{4,3}^T B_s^T R_{7,0} \quad (3.25)$$

$$C_w = R_{4,3}^T C_s^T R_{7,0} \quad (3.26)$$

e ora si possono estrapolare analiticamente le posizioni angolari delle articolazioni del polso dagli elementi della matrice $R_{7,4}$ (che contiene θ_5, θ_6 e θ_7),

$$R_{7,4} = \begin{bmatrix} * & * & cos(\theta_5)sen(\theta_6) \\ * & * & sen(\theta_5)sen(\theta_6) \\ -sen(\theta_6)cos(\theta_7) & sen(\theta_6)sen(\theta_7) & cos(\theta_6) \end{bmatrix} \quad (3.27)$$

Rispettando il parametro di configurazione globale, si calcolano le variabili di giunto rimanenti:

$$\theta_5 = atan2[GC_6(a_{w,2,3}sen(\psi) + b_{w,2,3}cos(\psi) + c_{w,2,3}), \\ GC_6(a_{w,1,3}sen(\psi) + b_{w,1,3}cos(\psi) + c_{w,1,3})] \quad (3.28)$$

$$\theta_6 = GC_6 arcos(a_{w,3,3}sen(\psi) + b_{w,3,3}cos(\psi) + c_{w,3,3}) \quad (3.29)$$

$$\theta_7 = atan2[GC_6(a_{w,3,2}sen(\psi) + b_{w,3,2}cos(\psi) + c_{w,3,2}), \\ GC_6(-a_{w,3,1}sen(\psi) - b_{w,3,1}cos(\psi) - c_{w,3,1})] \quad (3.30)$$

In conclusione, le variabili di giunto sono determinate in modo univoco in base a una posa target ($T_{7,0}$) e due parametri ausiliari, l'angolo del braccio (ψ) e la configurazione articolare (GC).

Capitolo 4

Descrizione ambiente di programmazione Kuka Sunrise.OS

Vengono utilizzati i seguenti componenti software:

- KUKA Sunrise.OS 1.13
- KUKA Sunrise.Workbench 1.13
- WorkVisual 4.0

KUKA Sunrise.OS è un pacchetto software di sistema per robot industriali in cui le attività di programmazione e controllo operatore sono strettamente separate l'una dall'altra. Le applicazioni robot sono programmate con KUKA Sunrise.Workbench, una cella robot (stazione) viene azionata tramite il pannello di controllo KUKA smartPAD, una stazione è composta da un controller di robot, un manipolatore e altri dispositivi e infine una stazione può eseguire più applicazioni (attività).

KUKA Sunrise.Workbench è lo strumento per l'avvio di una stazione e lo sviluppo di applicazioni robot ed è basato su linguaggio Java. WorkVisual viene utilizzato per la configurazione del bus e la mappatura di esso. Lo smartPAD è richiesto solo in fase di avvio per attività che per motivi pratici o di sicurezza non possono essere eseguite utilizzando KUKA Sunrise.Workbench. Lo smartPAD è utilizzato per esempio per padroneggiare assi, strumenti di calibrazione e punti di insegnamento. Dopo lo start-up e lo sviluppo dell'applicazione, l'operatore può eseguire semplici operazioni di manutenzione e attività operative utilizzando lo smartPAD. L'operatore non può modificare la stazione e la configurazione di sicurezza o la programmazione.

Si considera ora l'interfaccia di programmazione del software e si cercano di spiegare gli aspetti principali di fronte a cui si trova il programmatore.

L'interfaccia utente di KUKA Sunrise.Workbench è composta da diverse viste (Figura 4.1). La combinazione di più viste è detta prospettiva. KUKA Sunrise.Workbench offre varie prospettive preconfigurate. La prospettiva di programmazione è aperta di default ma possono poi essere visualizzate ulteriori prospettive.

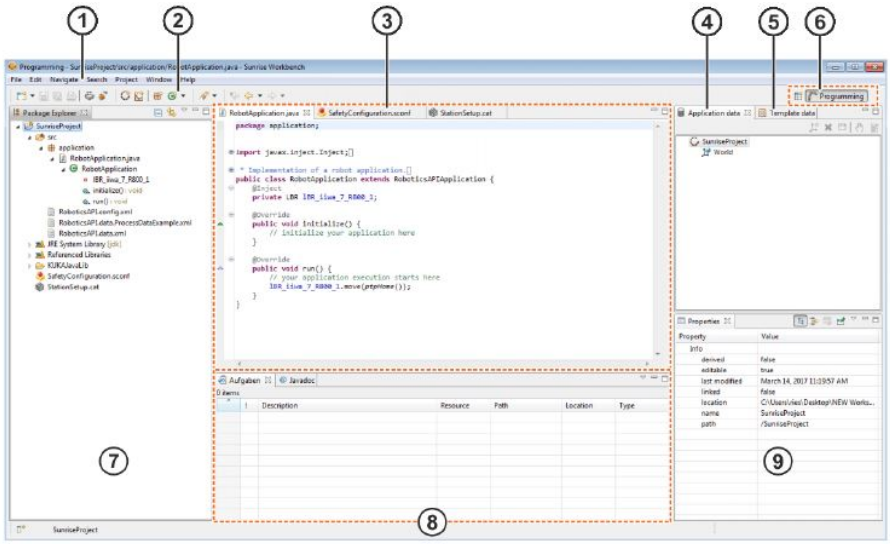


Figura 4.1: Panoramica dell'interfaccia utente - Prospettiva "Programmazione"

Tabella 4.1: Descrizione interfaccia utente

Voce	Descrizione
1	Barra menù
2	Barra degli strumenti
3	Area editor: file aperti, ad es. applicazioni robotizzate, possono essere visualizzate e modificate nell'area dell'editor
4	Vista dati dell'applicazione: questa vista mostra i frame creati per un progetto in una struttura ad albero
5	Vista modelli oggetto: questa vista mostra gli oggetti geometrici, gli strumenti e i pezzi creati per un progetto in una struttura ad albero
6	Selezione prospettica: è possibile passare da una prospettiva all'altra già in uso facendo clic sul nome della prospettiva desiderata o selezionandolo tramite l'icona Apri prospettiva
7	Vista Esplora pacchetto: questa vista contiene i progetti creati e i loro corrispondenti file
8.1	Visualizzazione delle attività: le attività che un utente ha creato sono visualizzate in questa vista
8.2	Vista Javadoc: i commenti Javadoc sugli elementi selezionati di un'applicazione Java vengono visualizzati in questa vista
9	Vista Proprietà: le proprietà dell'oggetto, ad es. progetto, cornice o strumento, selezionato in una vista diversa, sono visualizzati in questa vista

4.1 Strumenti utilizzati

In una prima fase di studio si è cercato di prendere confidenza con più strumenti software possibili. Solo successivamente, avendo definito l'attività specifica, se ne sono utilizzati alcuni.

4.1.1 Gestione dei Frame

I frame sono trasformazioni di coordinate che descrivono la posizione dei punti nello spazio o gli oggetti in una stazione. Le trasformazioni di coordinate sono disposte gerarchicamente in una struttura ad albero. In questa gerarchia, ogni frame ha un frame padre di livello superiore con cui è collegato tramite la trasformazione. L'elemento principale o l'origine della trasformazione è il sistema di coordinate world che si trova di serie nella base del robot. Ciò significa che tutti i frame sono direttamente o indirettamente correlati al sistema di coordinate world. Una trasformazione descrive la posizione relativa di 2 sistemi di coordinate tra loro, cioè come un frame è sfalsato e orientato rispetto al suo frame principale.

La posizione di un frame rispetto al frame principale è definita dai seguenti dati di trasformazione: X, Y, Z: offset dell'origine lungo gli assi del frame principale, A, B, C: offset di rotazione degli angoli degli assi del frame principale (A = rotazione attorno all'asse Z, B = rotazione attorno all'asse Y, C = rotazione attorno all'asse X).

4.1.2 Panoramica tipi di movimento

I seguenti tipi di movimento possono essere programmati come un singolo movimento: movimento punto a punto (PTP), movimento lineare (LIN), movimento circolare (CIRC), movimento di guida manuale con dispositivo di guida manuale. I seguenti tipi di movimento possono essere programmati come segmenti di un blocco CP spline (Continuos Path): movimento lineare (LIN), movimento circolare (CIRC), movimento polinomiale (SPL). I seguenti tipi di movimento possono essere programmati come segmenti di un blocco JP spline (Joint Path): movimento punto a punto (PTP).

Movimento PTP

Il robot guida il TCP (Tool Center Point) lungo il percorso più veloce fino al punto finale. Il percorso più veloce non è generalmente il percorso più breve nello spazio e quindi non è una linea retta. Poiché i movimenti degli assi del robot sono

simultanei e rotatori, i percorsi curvi possono essere eseguiti più velocemente dei percorsi rettilinei. PTP è un movimento di posizionamento veloce. Il percorso esatto del movimento non è prevedibile, ma è sempre lo stesso, purché le condizioni generali non vengano modificate.

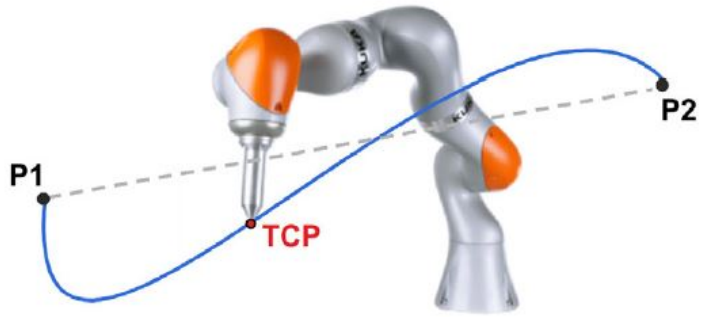


Figura 4.2: Movimento PTP

Movimento LIN

Il robot guida il TCP alla velocità definita lungo un percorso rettilineo nello spazio fino al punto finale. In un movimento LIN, la configurazione del robot della posa finale non viene presa in considerazione.

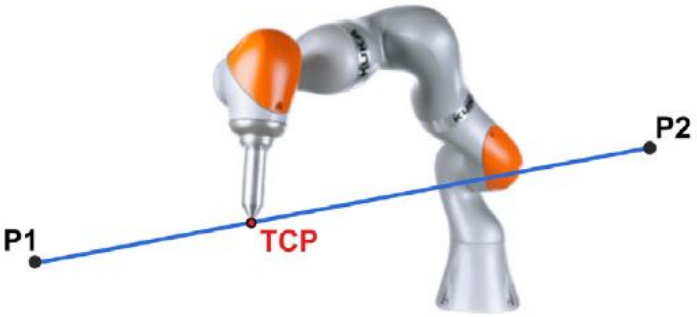


Figura 4.3: Movimento LIN

Movimento CIRC

Il robot guida il TCP alla velocità definita lungo un percorso circolare fino al punto finale. Il percorso circolare è definito da un punto iniziale, un punto ausiliario e un punto finale. In un movimento CIRC, la configurazione del robot della posa finale non viene presa in considerazione.

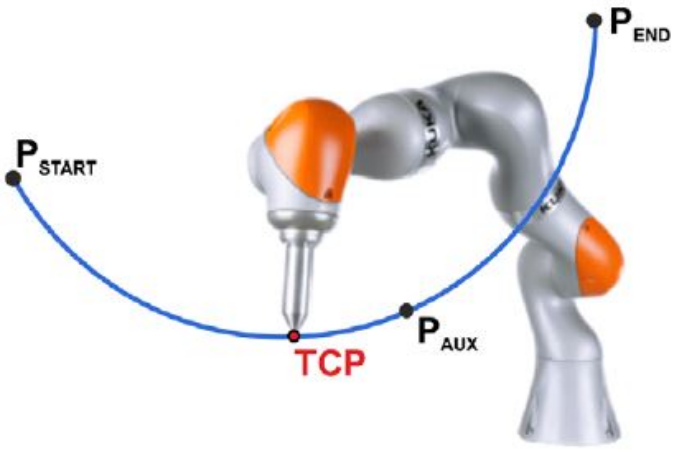


Figura 4.4: Movimento CIRC

Movimento Spline SPL

Il tipo di movimento SPL consente la generazione di percorsi curvi. I movimenti SPL sono sempre raggruppati in blocchi spline. I percorsi risultanti passano in maniera morbida attraverso i punti finali del movimento SPL. In un movimento SPL, la configurazione del robot della posa terminale non viene presa in considerazione.

La spline è un tipo di movimento particolarmente adatto per percorsi complessi e curvi. Con un movimento spline, il robot può eseguire questi percorsi complessi in un movimento continuo. Tali percorsi possono anche essere generati usando movimenti approssimati di LIN e CIRC, ma le spline hanno comunque dei vantaggi. Le spline sono programmate in blocchi; un blocco spline viene utilizzato per raggruppare più movimenti individuali come movimento generale. Il blocco spline è pianificato ed eseguito dal controller del robot come un singolo blocco di movimento. I movimenti contenuti in un blocco spline sono detti segmenti spline. Un blocco CP spline può contenere segmenti SPL, LIN e CIRC. Un blocco JP spline può contenere segmenti PTP. In un movimento spline cartesiano, la configurazione del robot della posa terminale non viene presa in considerazione. La configurazione della posa terminale di un segmento spline dipende dalla configurazione del robot all'inizio del segmento spline.

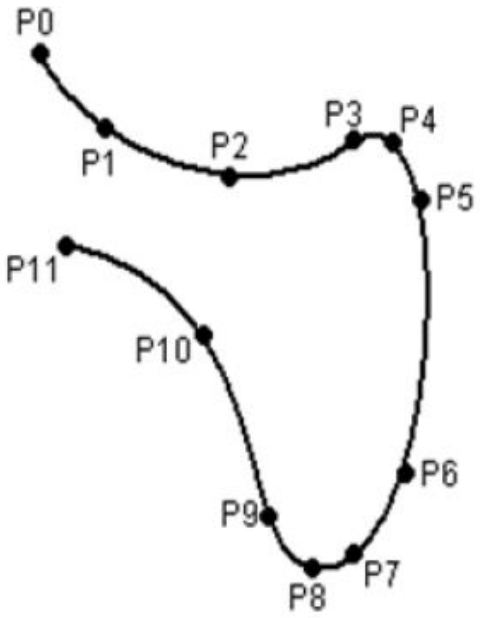


Figura 4.5: Percorso curvo con blocchi spline

4.1.3 Panoramica Impedance Control

Il controllore di impedenza cartesiana è rappresentato dalla classe Cartesian-Impedance Control Mode. Di norma, il controllore di impedenza fa riferimento al sistema di coordinate con cui viene eseguito il comando di movimento (sia esso riferito alla flangia o all'organo terminale accoppiato). Sotto il controllo impedenza, il comportamento del robot è collaborativo, è sensibile e può reagire a influenze esterne come ostacoli o forze di processo. L'applicazione di forze esterne può far sì che il robot lasci il percorso pianificato. Il modello sottostante è basato su molle virtuali e smorzatori, che sono allungati a causa della differenza tra la posizione attualmente misurata e quella specificata del TCP (punto centrale dell'utensile, ovvero il suo punto di lavoro). Le caratteristiche delle molle sono descritte da valori di rigidità e quelli degli smorzatori sono descritti da valori di smorzamento. Questi parametri possono essere impostati individualmente per ogni dimensione traslazionale e rotazionale. Se le posizioni del robot misurate e specificate corrispondono, le molle virtuali sono allentate. Poiché il comportamento del robot è collaborativo, una forza esterna o un comando di movimento provoca uno scostamento tra il punto di regolazione e le posizioni effettive del robot. Ciò si traduce in una deflessione delle molle virtuali, che porta a una forza in conformità con la legge di Hooke. La forza risultante F può essere calcolata sulla base della legge di Hooke utilizzando la

rigidità della molla C impostata e la freccia Δx:

$$F = C\Delta x$$

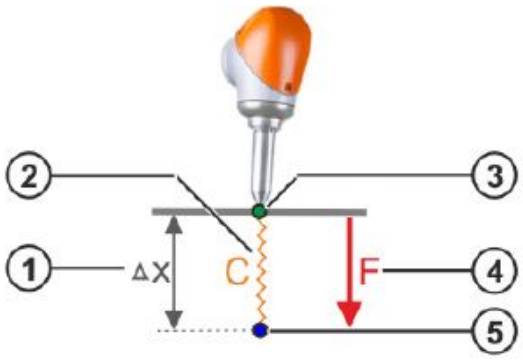


Figura 4.6: Molla virtuale con rigidità C

dove: 1) Deformazione Δx, 2) Molla virtuale, 3) Posizione attuale, 4) Forza risultante, 5) Posizione del punto di riferimento.

Se il robot è in un tiro, esercita la forza calcolata. Se è posizionato nello spazio libero, si sposta verso la posizione del setpoint. A causa delle forze di attrito interne nei giunti, le deviazioni del percorso si verificano sulla strada verso la posizione di riferimento, la cui ampiezza dipende dalla rigidità della molla impostata. Valori di rigidità più elevati portano a deviazioni più piccole. Se il robot si trova già nella posizione di setpoint e viene applicata una forza esterna al sistema, il robot applica questa forza fino a quando le forze risultanti non si annullano. Una grande differenza di posizione e una bassa rigidità possono provocare la stessa forza di una differenza di posizione più piccola e una maggiore rigidità. Se la forza viene aumentata da un movimento in una situazione di contatto, il tempo richiesto per raggiungere questa forza differisce se la velocità cartesiana è identica.

Se si utilizzano valori di rigidezza più elevati, è possibile raggiungere prima una forza desiderata, poiché è richiesta solo una piccola differenza di posizione. Poiché la posizione di riferimento viene raggiunta rapidamente, in questo modo si può produrre un sobbalzo. Nel caso di una grande differenza di posizione e di una bassa rigidità, la forza si accumula più lentamente. Questo può essere utilizzato, ad esempio, se il robot si sposta nel punto di contatto e i carichi di impatto devono essere ridotti.

La deflessione nella direzione X di Δx e nella direzione Y di Δy risulta in forza F_x nella direzione X e F_y nella direzione Y. L'aggiunta del vettore determina la forza complessiva F_{res}.

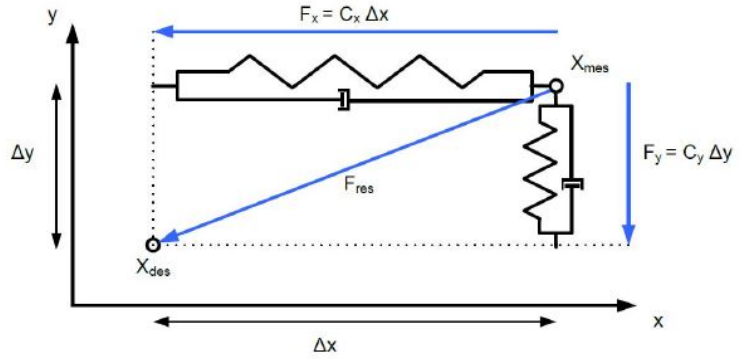


Figura 4.7: Forza complessiva nel caso di deflessione in 2 direzioni

I parametri che si possono andare a settare all'interno del codice quando si ha intenzione di adottare l'Impedance Control sono diversi. Le seguenti proprietà del controllo possono essere definite singolarmente per ciascun grado di libertà cartesiano, ovvero sia per le tre dimensioni X,Y,Z sia per le rispettive rotazioni: rigidezza, smorzamento, forza da applicare in aggiunta alla molla. Le seguenti proprietà del controller possono essere definite indipendentemente dal grado di libertà: rigidità del grado di libertà di ridondanza, smorzamento del grado di libertà di ridondanza, limitazione della forza massima sul TCP, massima velocità cartesiana, deviazione massima del percorso cartesiano.

L'Impedance Control può essere utilizzato sia per percorsi lineari e rettilinei ma anche per percorsi di varie forme, sinusoidali per esempio o di combinazioni, nel piano cartesiano, di traiettorie diverse per direzione X,Y e Z.

Controllo di impedenza cartesiana con oscillazione forzata sovrapposta

Il controllore di impedenza cartesiana con oscillazione forzata sovrapposta è una forma speciale del controllore di impedenza cartesiana. La forza può essere sovrapposta separatamente per ogni grado di libertà cartesiano. Oscillazioni di forza su un asse generano oscillazioni di coppia. Le oscillazioni di coppia sovrapposte possono determinare la generazione di oscillazioni rotazionali. La sovrapposizione di forze costanti o sinusoidali provoca il movimento del robot. Combinazioni di oscillazioni adatte nei singoli gradi di libertà possono essere utilizzate per generare diversi modelli di movimento. Utilizzando le oscillazioni sovrapposte, è possibile implementare movimenti pendolari conformi per le corse di ricerca e le vibrazioni nello strumento per i processi di unione. Il controllore di impedenza cartesiana con oscillazione forzata sovrapposta è rappresentato dalla classe CartesianSineImpedanceControlMode. In questa forma di controllo dell'impedenza, la forza sovrapposta induce il robot a lasciare il percorso pianificato in modo mirato. Il nuovo percorso

è quindi determinato da una vasta gamma di parametri diversi: oltre alla rigidità e allo smorzamento, è possibile definire ulteriori parametri, ad esempio frequenza e ampiezza. Anche la velocità programmata del robot gioca un ruolo significativo per il percorso reale.

Sovrapponendo una semplice oscillazione di forza, il punto di lavoro viene deviato dal percorso pianificato (= percorso senza oscillazioni sovrapposte) e viene invece spostato dal punto iniziale al punto finale del movimento in un percorso sinusoidale.

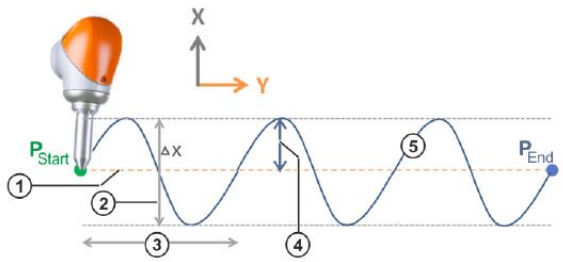


Figura 4.8: Sovrapposizione di una semplice oscillazione di forza

dove: 1) Percorso originale, 2) Deformazione Δx, 3) Lunghezza d'onda, 4) Ampiezza, 5) Nuovo percorso.

La deformazione massima Δx è la deviazione dal percorso originale nelle direzioni X positive e negative. La deformazione massima è determinata dalla rigidità e ampiezza che sono definite per il controllore di impedenza nella direzione X cartesiana. La lunghezza d'onda può essere utilizzata per determinare quante oscillazioni il robot deve eseguire tra il punto iniziale e il punto finale del movimento. La lunghezza d'onda è determinata dalla frequenza definita per il controllore di impedenza con oscillazione forzata sovrapposta, nonché dalla velocità programmata del robot.

Una alternativa al percorso rettilineo e con oscillazione di forza in una dimensione sola è la combinazione di oscillazioni forzate sovrapposte (curve di Lissajous).

Combinazione di oscillazioni forzate sovrapposte (curve di Lissajous)

Le curve di Lissajous risultano quando un'oscillazione della forza sinusoidale viene sovrapposta in 2 direzioni cartesiane diverse. La sovrapposizione delle due oscillazioni consente di creare forme molto diverse per il percorso. Il percorso esatto dipende da un numero di parametri; ad esempio se si considerano due oscillazioni di forza sinusoidale di diverse frequenze, queste possono essere sovrapposte per generare vibrazioni al TCP.Tali vibrazioni possono rimuovere tensioni e inceppamenti che si verificano durante un processo di assemblaggio.

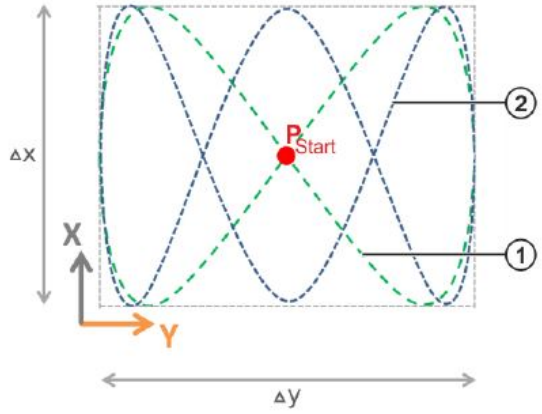


Figura 4.9: Percorso di una curva di Lissajous

dove: 1) Percorso senza sfasamento (rapporto di frequenza X:Y = 2:1), 2) Percorso con sfasamento (rapporto di frequenza X:Y = 3:1).

La forma del percorso è determinata principalmente dal rapporto tra le due frequenze e lo sfasamento tra le due oscillazioni. La curva risultante è sempre asimmetrica. L'ampiezza e la rigidezza impostata per una direzione di oscillazione determinano l'ampiezza della sua posizione. Il rapporto tra le due ampiezze di posizione determina il rapporto tra la larghezza e l'altezza della curva.

Con un controllo di impedenza cartesiano con oscillazione forzata sovrapposta, le forze possono essere sovrapposte per tutti i gradi di libertà cartesiani. Le forze che agiscono attorno a un asse generano una coppia. Per questo motivo, la coppia sovrapposta, e non la forza sovrapposta, è specificata per i gradi di libertà di rotazione. A fini di semplificazione, i termini "forza" e "oscillazione forzata" sono intesi a includere i termini "coppia" e "oscillazione di coppia" per i gradi di libertà di rotazione nel testo seguente. Il controllo di impedenza cartesiana con oscillazione forzata sovrapposta è parametrizzato allo stesso modo del controllo di impedenza standard. I parametri del controllo specifici per i gradi di libertà e i parametri del controllo indipendenti dai gradi di libertà descritti per il controllo di impedenza standard possono essere utilizzati allo stesso modo per il controllo di impedenza con oscillazione forzata sovrapposta.

Le seguenti proprietà aggiuntive del controller possono essere definite singolarmente per ciascun grado di libertà cartesiano: 1) Ampiezza dell'oscillazione della forza, 2) Frequenza dell'oscillazione della forza, 3) Offset di fase dell'oscillazione della forza, 4) Forza costante sovrapposta, 5) Limitazione della forza, 6) Limitazione della deformazione dovuta all'oscillazione della forza.

Le seguenti proprietà aggiuntive del controller possono essere definite indipendentemente dal grado di libertà: 1) Tempo di salita dell'oscillazione della forza, 2)

Tempo di attesa dell'oscillazione della forza, 3) Tempo di caduta dell'oscillazione della forza, 4) Durata complessiva dell'oscillazione della forza.

Il controllo di impedenza cartesiana con oscillazione forzata sovrapposta può anche essere configurato tramite metodi statici della classe CartesianSineImpedanceControlMode. Questo semplifica la programmazione, in particolare delle curve di Lissajous, poiché l'utente deve solo specificare alcuni parametri. I restanti parametri importanti per l'implementazione vengono calcolati e impostati automaticamente. I valori predefiniti sono usati per tutti gli altri parametri. Ulteriori impostazioni vengono eseguite come descritto utilizzando la funzione parametrize() e i metodi set() di CartesianSineImpedanceControlMode.

4.1.4 Panoramica Break Condition per comandi di movimento

Per determinati processi, un movimento pianificato non deve essere completamente eseguito, ma piuttosto terminato quando si verificano eventi definibili. Ad esempio, nel processo di unione, il robot deve fermarsi se viene raggiunta una soglia di forza.

Le condizioni di pausa sono condizioni che causano la chiusura di un movimento. Una condizione di interruzione viene soddisfatta se ha già lo stato TRUE prima dell'inizio del movimento o se passa allo stato TRUE durante il movimento. Le condizioni sono definite come oggetti di tipo ICondition. Per definire una condizione di fermata per un movimento, un oggetto del tipo di condizione desiderato viene trasferito al comando di movimento tramite il metodo breakWhen(). breakWhen() può essere chiamato più volte quando si programma un comando di movimento per definire condizioni di fermata differenti per un movimento. Le singole condizioni di interruzione sono quindi collegate da un'operazione logica OR. I seguenti punti devono essere presi in considerazione durante la programmazione delle condizioni di interruzione: 1) per un blocco spline, le condizioni di interruzione possono essere programmate solo per l'intero blocco spline; le condizioni di rottura per i singoli segmenti di spline non sono consentite, 2) se viene attivata una condizione di interruzione definita per un movimento all'interno di un MotionBatch, questa viene interrotta e quindi viene eseguito il comando di movimento successivo nel batch. Se si verifica una condizione di interruzione definita per l'intero MotionBatch, l'intero MotionBatch viene terminato, 3) una condizione di pausa causa la chiusura del movimento attualmente in esecuzione; se nell'applicazione non è programmata alcuna strategia di reazione appropriata, i movimenti successivi vengono eseguiti immediatamente dopo il movimento terminato, 4) nel caso di movimenti approssimati, l'arco di posizionamento approssimativo fa parte del percorso del movimento successivo.

Per questo motivo, solo le condizioni di interruzione per il movimento successivo influiscono sull'arco di posizionamento approssimativo, 5) se la condizione di rottura in un movimento approssimato si verifica appena prima del raggiungimento del punto di posizionamento approssimativo, e se ciò non causa il blocco del robot fino a quando si trova sull'arco di posizionamento approssimativo, il robot viene nuovamente accelerato quando l'arco di posizionamento approssimativo è raggiunto al fine di eseguire il movimento successivo.

Se sono state definite condizioni di interruzione per un comando di movimento, è possibile visualizzare varie informazioni sulla conclusione di un movimento: la condizione che ha causato la cessazione di un movimento, posizione del robot valida al momento in cui la condizione di fermata è stato attivata, il segmento di un blocco spline o il movimento di a MotionBatch che è stato chiuso. Più nello specifico, per quanto riguarda la posizione del robot possiamo sapere: posizione attuale e specifica degli assi, posizione attuale cartesiana, posizione di riferimento specifica per gli assi, posizione di riferimento cartesiana, differenza valore di riferimento / valore attuale (traslazionale).

4.1.5 Panoramica Condizioni

Spesso i valori delle grandezze di interesse devono essere monitorati nelle applicazioni e se i limiti definibili vengono superati o non raggiunti, devono essere attivate reazioni specifiche. Le possibili fonti per questi valori includono i sensori del robot o gli ingressi configurati. Il progresso di un comando di movimentazione può anche essere monitorato. Le possibili reazioni sono la cessazione di un movimento in esecuzione o l'esecuzione di una routine di gestione. Una condizione può avere 2 stati: soddisfatta (stato = VERO) o non soddisfatta (stato = FALSO). Per definire una condizione viene formulata un'espressione. In questa espressione, i dati, come le misure fornite dal sistema, vengono confrontati con un valore limite consentito. Il risultato della valutazione dell'espressione definisce lo stato della condizione. Alcuni dati di sistema, ad esempio coppie di assi o forze e coppie cartesiane sulla flangia del robot, sono disponibili solo per tipi di robot sensibili dotati di sistemi di sensori corrispondenti. Questi tipi di robot sensibili includono LBR iiwa. I tipi di condizione che utilizzano forze o coppie sono supportati solo da questi tipi di robot sensibili. Se questi tipi di condizione vengono applicati a robot che non forniscono informazioni su forze o coppie, ciò si traduce in un errore di runtime (eccezione).

I vari tipi di condizione possono essere suddivisi nelle seguenti categorie: 1) condizioni relative al sensore, 2) condizioni relative al percorso, 3) condizioni rela-

tive alla distanza, 4) condizioni relative all'input/output.

Condizioni relative al sensore sono: 1) JointTorqueCondition, ovvero la condizione di coppia dell'asse è soddisfatta se la coppia misurata in un asse si trova al di fuori di un intervallo definito di valori, 2) ForceCondition, ovvero la condizione di forza è soddisfatta se la forza cartesiana esercitata su un frame al di sotto della flangia del robot (ad esempio sul TCP) supera un valore definito, 3) ForceComponentCondition, ovvero la condizione del componente di forza viene soddisfatta se la forza cartesiana esercitata lungo un asse di un frame al di sotto della flangia del robot (ad esempio lungo un asse del TCP) supera un intervallo definito, 4) CartesianTorqueCondition, ovvero le condizioni per la coppia cartesiana sono soddisfatte se la coppia cartesiana agente attorno all'asse di un frame al di sotto della flangia del robot (ad esempio attorno all'asse del TCP) supera un valore definito, 5) TorqueComponentCondition, ovvero la condizione del componente di coppia è soddisfatta se la coppia cartesiana esercitata attorno ad un asse di un frame al di sotto della flangia del robot (ad esempio attorno ad un asse del TCP) è al di fuori di un intervallo definito.

Condizione relativa al percorso è MotionPathCondition, ovvero la condizione relativa al percorso viene soddisfatta se viene raggiunta una distanza definita sul percorso pianificato, dal punto iniziale o finale del movimento. Inoltre, è possibile definire un ritardo che deve essere rispettato.

Condizioni relative alla distanza sono: 1) FrameDistanceCondition, ovvero la condizione di distanza viene soddisfatta se la distanza cartesiana tra 2 frame è inferiore ad una distanza definita, ad esempio la distanza tra un pezzo in lavorazione, riferito al TCP o afferrato, e un punto di riferimento fisso definito, 2) FrameDistanceComponentCondition, ovvero la condizione del componente di distanza viene soddisfatta se la distanza cartesiana tra 2 frame rispetto agli assi specificati di un frame di orientamento è inferiore a una distanza definita.

Condizioni relative all'input/output sono: 1) BooleanIOCondition, ovvero la condizione per i segnali booleani è soddisfatta se un ingresso o uscita digitale booleana ha uno stato specifico, 2) IORangeCondition, ovvero la condizione per il campo di valori di un segnale viene soddisfatta se il valore di un ingresso o uscita analogici o digitali si trova all'interno di un intervallo definito.

4.1.6 Panoramica Thread

Per registrare i dati sulla posizione del robot durante il suo movimento si sono utilizzati i Thread, ovvero programmi che possono essere eseguiti in parallelo. Ve-

diamo velocemente cosa sono i Thread come si programma con essi. Un thread è simile a un programma sequenziale.

Un singolo thread ha: una sequenza di esecuzione, una fine e, ad ogni istante, durante l'esecuzione del thread c'è un singolo punto in esecuzione. Comunque, non è esso stesso un programma in quanto non può essere eseguito in modo indipendente ma viene invece eseguito all'interno di un programma. Un thread è un singolo flusso sequenziale di istruzioni all'interno di un programma. Si possono utilizzare più threads all'interno di un singolo programma; essi andranno in esecuzione allo stesso tempo ma eseguendo tasks differenti. Alcuni si riferiscono ai threads come "processo leggero", eseguito all'interno di un contesto di un programma completo, che utilizza le risorse allocate per il programma e l'ambiente del programma. Tuttavia, come un flusso di controllo sequenziale, un thread deve ritagliarsi alcune delle sue risorse all'interno del programma in esecuzione. Esso deve quindi avere il suo stack di esecuzione e un program counter.

L'uso dei thread può comportare svantaggi: essi sono difficili da usare, rendono i programmi difficili da debuggare, rischio di deadlock, bisogna avere cura che tutti i threads creati non invochino componenti di Swing (i componenti di Swing non sono "thread safe"). L'uso dei thread comporta però anche vantaggi: miglioramento delle performance dei programmi, specie in applicazioni grafiche; semplificazione del codice: ad esempio per eventi temporizzati, piuttosto che eseguire cicli o polling è possibile utilizzare una classe Timer che notifica una certa quantità di tempo trascorsa.

Un programma diventa multithreaded se il singolo thread costruisce e avvia un secondo thread di esecuzione; ogni thread può attivare (start) un qualsiasi numero di threads. In Java i threads sono oggetti, e vi sono 2 modi per crearli: implementare l'interfaccia Runnable (e passare questo nel costruttore di un Thread), estendere java.lang.Thread (che implementa Runnable). Quando viene implementata l'interfaccia Runnable, si deve fornire il metodo run() (l'unico) per indicare il lavoro da svolgere; tale metodo solitamente ha natura privata, ma essendo parte di un'interfaccia è di fatto pubblico, e questo potrebbe non essere accettabile (chiunque potrebbe invocarlo). L'uso di oggetti Runnable permette maggiore flessibilità, perché ognuno di essi costituisce una unità lavorativa indipendente, anche rispetto al resto del programma.

Stati di un Thread

Quando un thread è creato (New Thread) nel suo stato iniziale è un empty Thread object, ovvero è vuoto; nessuna risorsa di sistema è allocata per esso. Quando

un thread è in questo stato, su di esso può essere invocato il metodo start; se start è invocato su un oggetto thread che non si trova in tale stato iniziale si causa l'eccezione IllegalThreadStateException. Il metodo start crea le risorse di sistema necessarie per eseguire il thread, schedula il thread per eseguirlo, e chiama il metodo run della classe thread; dopo il return di start il thread è Runnable.

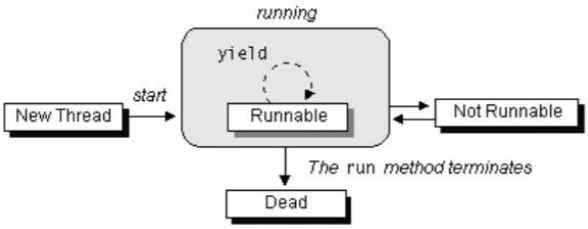


Figura 4.10: Stati di un thread

Un thread che viene inizializzato con start() è nello stato Runnable. Molti computer hanno un singolo processore, quindi è impossibile eseguire tutti i "running" threads allo stesso tempo. Il Java runtime system deve implementare uno schema di scheduling che permette la condivisione del processore fra tutti i "running" threads, così, ad un dato istante un solo "running" thread è realmente in esecuzione, gli altri sono in waiting per il loro turno di CPU. Un thread diventa Not Runnable (o blocked) quando: il thread chiama il metodo wait per aspettare che una specifica condizione sia soddisfatta oppure il thread è bloccato su un I/O. Per ogni entrata in un Not Runnable state, c'è una specifica e distinta azione che permette il ritorno nello stato di Runnable. Un programma non termina un thread attraverso la chiamata di un metodo; piuttosto, un thread termina naturalmente (dead) quando il metodo run termina (exit).

Schedulazione dei Thread

La concorrenza dell'esecuzione dei thread è vera, di norma, solo concettualmente; la maggior parte delle configurazioni di computer prevedono singole CPU, così i threads realmente sono eseguiti uno alla volta in modo da fornire una illusione di concorrenza. L'esecuzione di thread multipli su una singola CPU, in qualche ordine, è detto scheduling. Il Java runtime supporta un semplice algoritmo di scheduling deterministico conosciuto come fixed priority scheduling, che schedula i threads sulla base delle loro priorità relative agli altri runnable threads. Quando un thread viene creato esso eredita la sua priorita dal thread che lo ha creato. È possibile modificare la priorità di un thread in qualsiasi momento dopo la sua creazione utilizzando il metodo setPriority. Le priorità dei thread sono interi compresi fra *MINPRIORITY*

e *MAXPRIORITY* (costanti definite nelle classe Thread); il valore intero più grande corrisponde alla più alta priorità.

Ad un dato istante, quando più thread sono pronti per essere eseguiti, il runtime system sceglie il thread runnable con la più alta priorità per l'esecuzione. Solo quando il thread finisce o diventa not runnable per qualche ragione il thread di priorità minore viene eseguito. Se due threads della stessa priorità sono in attesa per la CPU, lo scheduler ne sceglie uno di loro seguedo un modello round-robin. Il thread scelto sarà eseguito fino a quando: o un thread con più alta priorità diventa runnable, o il suo metodo run esiste. Così i thread hanno una possibilità di essere eseguiti. Lo scheduler del Java runtime system è preemptive: se in qualsiasi momento un thread con una più alta priorità di tutti gli altri diventa runnable, allora il runtime system sceglie il nuovo thread di più alta priorità per l'esecuzione. Ad ogni istante il thread di massima priorità dovrebbe essere in esecuzione. Comunque, ciò non è garantito.

4.1.7 Panoramica Socket

Un socket è un oggetto software che permette l'invio e la ricezione di dati, tra host remoti (tramite una rete) o tra processi locali (Inter-Process Communication). Più precisamente, il concetto di socket si basa sul modello Input/Output su file di Unix, quindi sulle operazioni di open, read, write e close; l'utilizzo, infatti, avviene secondo le stesse modalità, aggiungendo i parametri utili alla comunicazione, quali indirizzi, numeri di porta e protocolli.

Socket locali e remoti in comunicazione formano una coppia (pair), composta da indirizzo e porta di client e server; tra di loro c'è una connessione logica. Si possono vedere i socket come degli intermediari tra il livello applicazione e di trasporto nello stack TCP/IP. Infatti la funzione dei socket è quella di indirizzamento dei processi.

Dato che sui sistemi interlocutori possono esserci molti processi, bisogna avere un modo per indirizzare precisamente il processo con cui si sta dialogando. Per questo si usano le porte: dei numeri che identificano i processi in esecuzione. Gli interlocutori, quindi, memorizzano indirizzo e porta della controparte, in un indirizzo socket, formato così: 1) Indirizzo IP: 32 bit, 2) Numero di porta: 16 bit. A loro volta i numeri di porta si dividono in: 1) Riservate per processi well-known: 1-255, 2) Riservate per altri processi: 256-1023, 3) Altre applicazioni: 1024-65535.

I tipi di protocolli utilizzati dal socket, ne definiscono la famiglia (o dominio). Si possono distinguere, ad esempio, due importanti famiglie: 1) AF-INET: comunicazione tra host remoti, tramite Internet, 2) AF-UNIX: comunicazione tra processi locali, su macchine Unix.

All'interno della famiglia si distingue il tipo di socket, a seconda della modalità di connessione. Si hanno:

- Stream socket: orientati alla connessione (connection-oriented), basati su protocolli affidabili come TCP o SCTP;
- Datagram socket: non orientati alla connessione (connectionless), basati sul protocollo veloce ma inaffidabile UDP;
- Raw socket (raw IP): il livello di trasporto viene bypassato, e l'header è accessibile al livello applicativo.

Nella attività presentata si utilizza la Stream socket.

Stream socket

Essendo basati su protocolli a livello di trasporto come TCP, garantiscono una comunicazione affidabile, full-duplex, orientata alla connessione, e con un flusso di byte di lunghezza variabile. La comunicazione avviene in diverse fasi, come riportato anche nella Figura 4.11.

1 – Creazione dei socket

Client e server creano i loro rispettivi socket, e il server lo pone in ascolto su una porta. Dato che il server può creare più connessioni con client diversi (ma anche con lo stesso), ha bisogno di una coda per gestire le varie richieste.

2 – Richiesta di connessione

Il client effettua una richiesta di connessione verso il server. Da notare che si possono avere due numeri di porta diversi, perchè una potrebbe essere dedicata solo al traffico in uscita, l'altra solo in entrata; questo dipende dalla configurazione dell'host. In sostanza, non è detto che la porta locale del client coincida con quella remota del server. Il server riceve la richiesta e, nel caso in cui sia accettata, viene creata una nuova connessione.

3 – Comunicazione

Ora client e server comunicano attraverso un canale virtuale, tra il socket del primo, ed uno nuovo del server, creato appositamente per il flusso dei dati di questa connessione: data socket.

Coerentemente a quanto accennato nella prima fase, il server crea il data socket perchè il primo serve esclusivamente alla gestione delle richieste. È possibile, quindi, che ci siano molti client a comunicare con il server, ciascuno verso il data socket creato dal server per loro.

4 – Chiusura della connessione

Essendo il TCP un protocollo orientato alla connessione, quando non si ha più la necessità di comunicare, il client lo comunica al server, che ne deistanzia il data socket. La connessione viene così chiusa.

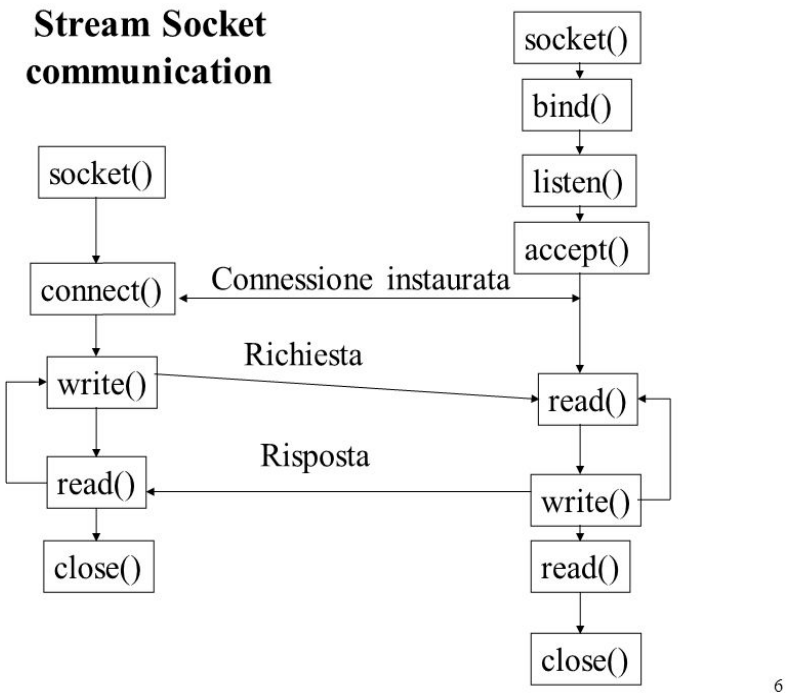


Figura 4.11: Diagramma di flusso di istruzioni per comunicazione

Capitolo 5

Attività sperimentale

L'obiettivo è quello di far muovere il robot in modo da seguire il profilo di una superficie qualsiasi e immagazzinare i dati sulle posizioni per poi rielaborarli e riprodurre la superficie attraverso un altro software (es. Matlab).

In laboratorio si è cercato di implementare un codice che potesse essere il più semplice possibile ma che racchiudesse tutte le nozioni viste nel capitolo precedente. In questa prima introduzione si descrive l'idea di base e si spiega il motivo delle varie scelte, mentre la descrizione del codice verrà fatta tramite la mostra di alcune funzioni utilizzate nel software nel paragrafo successivo. A conclusione l'analisi dei risultati e considerazioni.

L'idea base è far acquisire al robot dei punti di riferimento sulla superficie da analizzare, salvare questi punti e impostarli come vertici del percorso (un rettangolo) che il robot dovrà eseguire scansionando la superficie. Questo percorso potrà rimanere un rettangolo al momento in cui non siano presenti forze esterne che agiscono sull'organo terminale del robot, al contrario il percorso verrà deviato in funzione della forza applicata.

Per prima cosa si sono creati tutti i Frame di cui si aveva bisogno. Naturalmente tutti i Frame devono essere all'interno dello spazio di lavoro del robot, un Frame definito oltre i limiti fisici non sarà raggiungibile e il sistema darà un errore di violazione dei limiti degli assi. Ogni Frame è definito tramite 3 parametri di traslazione e 3 di rotazione.

Le prime righe di codice sono dedicate all'inclusione di tutti i pacchetti di java che si vogliono utilizzare. Ogni pacchetto definisce quale funzionalità delle librerie software si vogliono utilizzare e raccolgono al loro interno diverse funzioni preimpostate da poter chiamare all'interno del codice.

Per riuscire a far muovere il robot seguendo la superficie data si sono utilizzate le istruzioni di break e le varie condizioni di fermata impostate con una componente

di forza. Si è scelto questo approccio perchè permette la gestione di fermate dovute al contatto/urto del robot con altri oggetti, implementando così una soluzione che il robot può gestire in autonomia senza che sia l'operatore, dall'esterno, a farlo fermare. Quindi definita la forza superiore alla quale il robot deve fermarsi e la direzione di applicazione di essa, il robot, dati i comandi di movimento, comicia a muoversi verso il primo frame definito. Frame che, per le condizioni dettate dalla condizione di fermata, dovrà essere definito sotto il piano della superficie di studio, in modo da avere un punto non avvicinabile sempre indipendentemente dal tipo di superficie.

All'avvio il robot si porta nel frame definito come casa settato via software. Per rendere più interattiva la collaborazione con l'operatore, il robot viene programmato per attedere, in Home, che l'operatore gli mandi un segnale di partenza. In questo caso quando l'operatore va ad applicare una coppia positiva al giunto 6, il robot si approccerà al punto di partenza per la discesa verso il primo frame.

Da qui il robot scende lungo la sua verticale fino a cercare di raggiungere il frame impostato attraverso il rispettivo comando. Tuttavia il robot andrà ad incontrare la superficie di interesse prima di riuscire a raggiungere il suddetto frame, interrompendo la sua corsa. La condizione di fermata sarà dovuta alla forza che il robot sente sull'organo terminale dovuta alla superficie. A questo punto il robot dovrà aquisire la posizione in cui si è fermato. Questa procedura verrà ripetuta per gli altri tre punti di riferimento che si dovrà acquisire, in modo da definire in una prima fase quali sono i frame vertice che delimiteranno il percorso che il robot andrà ad eseguire.

Attraverso l'impostazione di costante elastica e coefficiente di smorzamento delle molle si imposta il robot ad essere più o meno sensibile alle asperità che trova lungo il suo percorso. Con una costante elastica elevata, la molla sarà più dura e avrà difficoltà a comprimersi, più è piccola la costante elastica e più si comprime. Analogamente per lo smorzamento, più si ha un coefficiente elevato (vicino a 1) e più si ha dissipazione sulle oscillazioni della struttura del robot, più è basso e meno sono filtrate le oscillazioni sul robot.

Una volta memorizzati i frame vertice del percorso il robot segue il profilo della superficie mantenendo sempre il contatto con essa, muovendosi all'interno dell'area definita dai vertici, i frame definiti appunto, non andando oltre il loro limite. Per rendere maggiormente interattiva l'azione del robot con il suo operatore si è introdotta una condizione di fermata imposta al robot se esso viene toccato dall'operatore e quest'ultimo esercita una forza opposta in direzione al moto del robot in quella specifica fase (se per es. il robot sta compiendo un movimento lungo l'asse

Y la forza per poterlo fermare dovrà essere applicata lungo Y). Questo impone al robot di cambiare la sua traiettoria una volta che avrà acquisito la posizione in cui si è fermato a causa dell'operatore. Così facendo il robot percorre una traiettoria da noi definita all'interno dei limiti imposti dai frame, tornando alla fine al punto di partenza.

Durante ogni tratto che il robot compie seguendo la superficie, si cercano di salvare le informazioni riguardo alla posizione in coordiante cartesiane dell'organo terminale in coordinate world. Per registrare i dati sulla posizione del robot durante il suo movimento si sono utilizzati i Thread, ovvero programmi che possono essere eseguiti in parallelo. Il Thread secondario viene creato come nuova classe di oggetto con tutte le sue funzioni specifiche che andranno fatte mentre il controllore esegue il programma principale nel quale verrà creata una istanza del Thread in modo da legare i due programmi. All'interno del Thread è presente una funzione che registra la posizione e la salva all'interno di un ArrayList; una funzione che monitora quando viene dato l'enable al Thread di iniziare la registrazione, in quanto si vogliono registrare solo i movimenti che il robot compie quando è sulla superficie; una funzione che permette di mettere in pausa la registrazione; una funzione che fa ripartire la registrazione; una funzione che termina il Thread; una funzione che mostra a video cosa il Thread sta registrando. Una volta che la registrazione si è conclusa, viene instaurata una connessione tramite stream socket, tramite tcp-ip, per consentire di passare l'ArrayList dal controllore del robot al pc esterno di elaborazione con Matlab. Così facendo si vanno ad analizzare i dati raccolti per cercare di riprodurre a schermo una piccola traccia di profilo della superficie rilevato.

Quindi se al robot non sono applicate forze esterne, esso si muoverà in autonomia percorrendo il percorso dato e passando per tutti i frame vertice. Se sono applicate forze esterne, il robot cambierà il suo percorso evitando di raggiungere il/i frame vertice per cui è stata soddisfatta la condizione di forza.

5.1 Struttura codice

Il software è diviso in due programmi, il programma principale nel quale vengono eseguite tutte le istruzioni di movimento e il Thread secondario legato al principale che dovrà registrare la posizione dell'organo terminale durante i movimenti desiderati senza contare i transitori.

L'intento del codice è fornire dati utili a studiare la ripetibilità e accuratezza del manipolatore durante la fase di movimento.

5.1.1 Programma principale

Il programma principale viene definito come SunriseApplication e in particolare come una classe, che prende il nome del programma, che è una estensione della classe RoboticsAPIApplication. Appena fatto questo il programma produce codice di default che è possibile integrare con le istruzioni desiderate.

Le prime righe di codice sono dedicate all'inclusione di tutti i pacchetti di Java che si vogliono utilizzare. Ogni pacchetto definisce quale funzionalità delle librerie software si vogliono utilizzare e raccolgono al loro interno diverse funzioni preimpostate da poter chiamare all'interno del codice. I pacchetti inclusi riguardano:

- Il movimento (es. com.kuka.roboticsAPI.motionModel.BasicMotions.ptp);
- La definizione del tipo di controllore (com.kuka.roboticsAPI.controllerModel.Controller);
- L'implementazione del controllo di impedenza (com.kuka.roboticsAPI.motionModel.controlModeModel. CartesianImpedanceControlMode);
- La definizione del tipo di robot (com.kuka.roboticsAPI.deviceModel.LBR);
- La definizione delle condizioni di fermata (com.kuka.roboticsAPI.executionModel.IFiredConditionInfo, com.kuka.roboticsAPI.conditionModel.ForceComponentCondition, com.kuka.roboticsAPI.conditionModel.ForceCondition);
- La definizione del tipo di informazione che si desidera ottenere dai sensori (com.kuka.roboticsAPI.deviceModel.PositionInformation, com.kuka.roboticsAPI.deviceModel.JointEnum);
- Il monitoraggio dei dati da un Frame, ovvero ottenere coordinate x,y,z (com.kuka.roboticsAPI.geometricModel.math.CoordinateAxis);
- Annotazione per l'integrazione delle risorse mediante dependency injection (javax.inject.Inject);
- L'implementazione dei Thread (com.kuka.common.ThreadUtil);
- L'implementazione delle liste di array (java.util.ArrayList);

- L'implementazione della classe Java DataInputStream che consente a un'applicazione di leggere i dati primitivi dal flusso di input in modo indipendente dalla macchina (java.io.DataInputStream);
- L'implementazione della classe Java DataOutputStream che consente a un'applicazione di scrivere tipi di dati primitivi nel flusso di output in modo indipendente dalla macchina (java.io.DataOutputStream);
- L'implementazione del socket del server. Un socket del server attende che le richieste arrivino attraverso la rete. Esegue alcune operazioni in base a tale richiesta e quindi restituisce eventualmente un risultato al richiedente (java.net.ServerSocket);
- L'implementazione dei socket client (chiamati anche solo "socket"). Un socket è un endpoint per la comunicazione tra due macchine (java.net.Socket);
- L'implementazione della classe che gestisce i Buffer di byte per la raccolta dei dati (java.nio.ByteBuffer).

Succesivamente, all'interno della definizione della classe, si definiscono tutte le variabile che verranno utilizzate nel programma. Attraverso il comando @Inject vengono richiamati gli oggetti che si vogliono utilizzare, che sono già definiti nella loro classe principale. Quindi nel caso d'interesse si ha:

- private LBR lBR-iiwa-14-R820-1 che definisce il tipo di robot;
- private JointPosition newHome che definisce la nuova casa del robot alla partenza del codice;
- private public Controller kukaController che definisce il controllore;
- private IMotionContainer motionCmd che definisce il comando di movimento;
- private IFiredConditionInfo firedCondInfo che definisce il tipo di informazione che si vuole ottenere dalla fermata dovuta alla condizione di break del movimento;
- private PositionInformation firedPosInfo che definisce la posizione alla quale si è interessati;
- private Frame firedCurrPos che definisce il tipo di posizione che si vuole ottenere;

- private NicolaTrackRec rec che inizializza l'istanza con la quale si richiama il Thread creato, che verrà spiegato in seguito;
- private ArrayList<byte[]> trackPoints che definisce la lista nella quale si andranno a salvare i dati;

A seguire si trova il metodo di inizializzazione:

- kukaController = (Controller) getContext().getControllers().toArray()[0] che inizializza il controllore;
- newHome = new JointPosition(Math.toRadians(1.41),Math.toRadians(23.58), Math.toRadians(-0.93),Math.toRadians(-87.08),Math.toRadians(0.46), Math.toRadians(69.35),Math.toRadians(0.40)) [1] che imposta l'home del robot;
- trackPoints = new ArrayList<byte[]>() che crea l'ArrayList;
- rec = new NicolaTrackRec(lBR-iiwa-14-R820-1, trackPoints, getLogger()) che crea l'istanza con i suoi parametri;

A seguire del metodo di inizializzazione si è voluto implementare un metodo aggiuntivo che rendesse più interattiva l'azione dell'operatore con il robot. Il metodo si chiama ManualMotion. Per questo l'idea concepita è quella di far posizionare il robot in "home" e farlo attendere finchè non sarà l'operatore, attraverso l'applicazione di una forza esterna, a dare il via al movimento del robot. Per implementare ciò si sono utilizzate le seguenti istruzioni:

- Si imposta la posizione di home del robot con le istruzioni setHomePosition(newHome),
  move(ptpHome());
- Attraverso un ciclo for attende la presenza di coppie esterne, una volta che la condizione viene verificata allora il robot si sposta e si porta nella posizione di partenza per il ciclo di lavoro;
- getLogger().info("Waiting for torque") consente di far vedere a display quello che è l'argomento, in formato stringa;
- getExternalTorque().getSingleTorqueValue(JointEnum.J6) consente di ottenere dal sensore del giunto 6 se è applicata coppia su di esso;

[1] NOTA BENE: il comando Math.toRadians() permette di convertire in radianti l'angolo, in gradi, preso come argomento

- con la condizione if si decide cosa il robot deve fare se la coppia esterna è positiva o negativa e quindi la rispettiva istruzione di move;

Dopo il metodo ManualMotion si passa al metodo run(), ovvero il metodo in cui è racchiusa tutta la parte di movimentazione e di fatto il programma del robot.

Per cominciare si sono definite le condizioni di fermata, ovvero quelle condizioni che verranno soddisfatte quando l'operatore dovrà interrompere il percorso del robot. Le condizioni fanno riferimento alla forza in direzione opposta a quella del moto del robot in quel momento, quindi varieranno per ogni tratto considerato visto che il percorso, a piacimento, è un rettangolo e quindi si hanno forze in x e in y. In più è presente la condizione di forza spaziale utilizzata nella prima parte per la definizione dei frame vertice raccolti in autonomia dal robot. La forma delle condizioni è la seguente:

- ForceComponentCondition(argomento1, argomento2, lim.inf., lim.sup) dove il primo argomento fa riferimento al punto di applicazione della forza, il secondo la direzione di applicazione, il terzo il limite inferiore e il quarto il limite superiore, in Newton. La condizione sarà soddisfatta se la forza applicata sarà inferiore al limite inferiore o maggiore al limite superiore;

- ForceCondition.createSpatialForceCondition(argomento1, argomento2) dove il primo argomento fa riferimento al punto di applicazione della forza, il secondo al valore di essa in Newton. La condizione sarà soddisfatta se la forza applicata sarà maggiore al valore impostato;

Successivamente si passa alla definizione dei parametri dell'Impedance Control, che all'interno della istruzione di motion si imposta attraverso la funzione setMode(nome dell'oggetto ImpedanceControl). Come visto nel capitolo precedente ci sono molti parametri impostabili; quelli utili all'applicazione sono stati:

- parametrize(CartDOF.X,CartDOF.Y,CartDOF.Z).setStiffness(val) che permette di settare al valore desiderato la costante elastica della molla nei tre assi cartesiani; ovviamente la costante elastica può differire per ciascun asse;

- parametrize(CartDOF.ROT).setStiffness(val) che permette sempre di settare al valore desiderato la costante elastica della molla ma riguardo alle rotazioni sul piano cartesiano;

- parametrize(CartDOF.ALL).setDamping(val) che permette di impostare lo smorzamento su ciascuno grado di libertà, sia di traslazione che di rotazione sul piano;

Dopo la definizione del metodo parte il movimento: viene richiamato il metodo ManualMotion e successivamente viene data al robot l'istruzione di muoversi verso il primo Frame di partenza, o vertice del rettangolo ideale che dovrà essere seguito sulla superficie; al soddisfacimento della condizione di forza il robot si fermerà e il programma memorizzerà la posizione in cui esso ha toccato la superficie. Una volta che il robot memorizza la posizione, si alza e si sposta verso il successivo punto interessato. Questa procedura di identificazione dei frame vertice verrà eseguita anche negli altri tre punti. La sintassi è la seguente:

- motionCmd = lBR-iiwa-14-R820-1.move(lin(getApplicationData().getFrame()) .setCartVelocity(30).breakWhen(cond1)) memorizza in motionCmd l'istruzione di movimento;
- firedCondInfo = motionCmd.getFiredBreakConditionInfo() restituisce informazioni riguardo la fermata;
- if(firedCondInfo != null) condizione di verifica, se realizzata memorizza informazioni sul punto in cui il robot si è fermato;
- firedPosInfo = firedCondInfo.getPositionInfo() permette di ottenere informazioni riguardo la posizione;
- firedCurrPos = firedPosInfo.getCurrentCartesianPosition() permette di definire se interessa il piano cartesiano o lo spazio dei giunti, in questo caso il piano cartesiano;
- getLogger().info() permette di stamapare a video l'attaule posizione in cui si è fermato il robot;
- Ritorno in partenza per togliere la condizione di forza una volta che ho salvato il frame vertice (move(ptp(getApplicationData().getFrame("/Step1")).setJointVelocityRel(0.2)));
- Movimento verso il successivo punto di acquisizione (move(ptp(getApplicationData().getFrame("/Step2")).setJointVelocityRel(0.2)));

Dopo la prima fase di approccio e fissaggio dei frame vertice cominicia la fase di rilevazione tramite Impedance Control. I movimenti sulla superficie saranno lineari e lungo direzioni x e y del piano. La procedura che sarà considerata per il primo tratto sarà ripetuta come un ciclo su tutti gli altri Frame.

In più, dato che il percorso che si vuole registrare parte da qui, si ha bisogno di far partire la registrazione attraverso il Thread secondario, che verrà spiegato

successivamente. Ogni metodo o funzione esposta non è presente all'interno della guida software fornita pertanto sono state implementate da zero.

- Per far partire l'esecuzione del Thread si è utilizzato il metodo start();
- Per far partire la registrazione si è utilizzato il metodo startRec();
- Per mettere in pausa il Thread si è utilizzato ThreadUtil.milliSleep();
- Per mettere in pausa la registrazione si è utilizzato rec.pauseRec(); questa funzone è servita in quanto ad ogni fermata imposta dall'operatore con la mano sul robot si interrompe la registrazione delle posizioni per non avere dati ridondanti o errati dovuti alla fermata;
- Per concludere la registrazione dei dati e concludere il Thread si è utilizzato rec.stopRec();

Successivamente all'inizio della registrazione, il robot si sposta per andare verso il Frame di riferimento. Qui, come spiegato precedentemente, si sono implementate due possibilità, per rendere più interattiva e visiva la collaborazione uomo-robot. Se non si applicano forze esterne, il robot procede in direzione lineare da un vertice al successivo. L'atra possbilità consiste nel porre una condizione di fermata determinata dall'applicazione di una forza esercitata dall'operatore lungo la direzione del moto ma contraria ad esso. Così facendo, toccando il robot, esso si fermerà a comando. A seguire verrà registrata la posizione di fermata come precedentemente avvenuto e a seguire il robot non procede verso il vertice che stava raggiungendo ma devierà il suo percorso verso quello immediatamente successivo.

Questa sequenza sarà poi ripetuta lungo tutti gli altri spostamenti.

- Comando di motion verso il frame vertice
  move(lin(frame vertice).setCartVelocity().breakWhen(cond).setMode(impcontr));
- firedCondInfo = motionCmd.getFiredBreakConditionInfo() restituisce informazioni riguardo la fermata;
- if(firedCondInfo != null) condizione di verifica, se realizzata memorizza informazioni sul punto in cui il robot si è fermato;
- Se la condizione è soddisfatta, salva la posizione corrente; se la condizione è falsa (vuol dire che non è stata rilevata forza esterna) procedi verso frame successivo;

Alla fine del percorso stabilito si accede all'array che il Thread secondario ha creato per stampare a display sullo smartpad il suo contenuto. Successivamente l'ArrayList di dati verrà inviato, tramite una comunicazione instaurata tramite socket, a Matlab per essere poi elaborato.

- Per accedere all'array si è utilizzato il metodo getList() legato al nome dell'array creato;
- Si verifica quale sia lo stato dell'array: vuoto o meno attraverso la condizione if;
- Se lo stato dell'array è pieno si procede alla visualizzazione a display tramite ciclo for con il quale si accede all'array e, dato che ogni elemento è salvato come byte, si utilizza la funzione get() per estrarre le varie componenti in x, y, z;

Per la comunicazione tra controllore e pc esterno il socket è stato realizzato con le seguenti istruzioni:

- Si imposta la variabile di tipo ServerSocket nulla per eliminare qualunque tipo di connessione precedente(ServerSocket server = null);
- Tramite il costrutto try si cerca di instaurare la connessione;
- Si crea una socket per l'utenza server (server= new ServerSocket(numero porta));
- Si attende che il server parta (server.setSoTimeout(50000));
- Creazione della connessione (Socket serverClient=server.accept());
- Creazione flusso in uscita di dati (DataOutputStream(serverClient.getOutputStream()));
- Creazione flusso in entrata (DataInputStream(serverClient.getInputStream()));
- Verifica connessione ok (outStream.write(ok), inStream.read(input));
- Creazione vettore di byte che verrà passato tramite la socket a Matlab (byte[] messageb = new byte[24]);
- Salvataggio, mediante ciclo for, di ogni riga dell'ArrayList trackPoints all'interno del nuovo buffer di byte (messageb = trackPoints.get(i));
- Invio del buffer al pc esterno (outStream.write(messageb));

- Chiusura flusso di lettura e scrittura (inStream.close(), outStream.close());
- Chiusura socket client e server (serverClient.close(), server.close());

5.1.2 Thread secondario

Il Thread a servizio del programma principale è concepito come una nuova classe, in particolare una estensione della classe Thread di Java. Infatti si vedrà poi come viene implementata. Nella prima parte di codice, come visto per il programma principale, si importano i vari pacchetti. I pacchetti inclusi riguardano:

- La definizione per l'utilizzo degli ArrayList (import java.util.ArrayList); un ArrayList è una lista rappresentata mediante array. Ogni ArrayList ha una capacità. La capacità è la lunghezza dell'array usato per memorizzare gli elementi della lista, e vale almeno la lunghezza della lista. Quando vengono aggiunti elementi alla lista, la sua capacità varia automaticamente;
- L'implementazione dei Timer (import java.util.Timer); la classe Timer fornisce una chiamata di metodo utilizzata da un thread per pianificare un'attività, ad esempio l'esecuzione di un blocco di codice dopo qualche istante di tempo regolare. Ogni attività può essere programmata per essere eseguita una volta o per un numero ripetuto di esecuzioni. Ogni oggetto timer è associato a un thread in background responsabile dell'esecuzione di tutte le attività di un oggetto timer;
- import java.util.TimerTask; TimerTask è una classe astratta definita nel pacchetto java.util. La classe TimerTask definisce un'attività che può essere pianificata per l'esecuzione per una sola volta o per un numero ripetuto di volte. Per definire un oggetto TimerTask, questa classe deve essere implementata e il metodo di esecuzione deve essere sostituito. Il metodo run viene implicitamente richiamato quando un oggetto timer lo schdedules per farlo;
- L'implementazione della classe per la gestione dei Thread (import com.kuka.common.ThreadUtil);
- La definizione del tipo di robot (com.kuka.roboticsAPI.deviceModel.LBR);
- Il monitoraggio dei dati da un Frame (com.kuka.roboticsAPI.geometricModel.Frame);
- L'implementazione dell'oggetto che stampa a display (com.kuka.task.ITaskLogger);

La definizione della classe viene descritta dalla seguente sintassi: public class NicolaTrackRec extends Thread all'interno della quale si definiscono i metodi e le variabili che servono per effettuare la registrazione delle posizioni in tempo reale.

Le variabili utilizzate sono le seguenti:

- private LBR lBR-iiwa-14-R820-1 che definisce il tipo di robot;
- private ITaskLogger logger che definisce l'oggetto di stampa a schermo;
- private ArrayList<Frame> theList che definisce il nome dell'ArrayList;
- private Boolean enable = true variabile booleana che viene utilizzata per dare il consenso o meno allo start per la registrazione;
- private int state che viene utilizzata come variabile di controllo per i vari stati che può assumere il Thread;
- private Timer timer che viene utilizzata come variabile di impostazione per il timer;

Successivamente si crea una funzione che definisce un task periodico schedulato dalla classe Timer, che può mettersi in pausa e ripartire quando state cambia di valore.

- private TimerTask task = new TimerTask() che definisce l'istanza task all'interno del quale la variabile di stato state cambia valore, è composto da un suo metodo di run(). All'interno del metodo run di TimerTask avvengono i diversi cambi di valore della variabile state;
- switch(state) permette di costruire condizioni annidate; nel caso1 avviene la registrazione della posizione cartesiana corrente attraverso l'istruzione theList.add(lBR-iiwa-14-R820-1.getCurrentCartesianPosition(lBR-iiwa-14-R820-1.getFlange()));
- Nel caso2 avviene la pausa del Tread e quindi della registrazione;
- Nel caso3 si procede al restart della registrazione, sempre attraverso l'istruzione precedente;
- Nel caso4 si procede all'interruzione della registrazione, la fine, con conseguente arresto del Thread;

Insieme alla definizione del task si definisce il costruttore della classe NicolaTrackRec, utilizzata per raccogliere in background i dati di posizione del robot. La sintassi è la seguente: public NicolaTrackRec (LBR -lbriiwa, ArrayList<Frame> -trackPoints,ITaskLogger visual), dove i parametri passati come argomenti al costruttore sono i seguenti:

- lBR-iiwa-14-R820-1 = -lbriiwa è la variabile LBR che definisce il robot di cui si vuole raccogliere i dati;
- theList = -trackPoints è l'ArrayList di Frame dal thread principale;
- logger = visual è l'oggetto Logger dal Thread principale per la visualizzazione a schermo i dati raccolti;
- timer = new Timer() permette di inizializzare il timer, ovvero definisce l'intervallo di tempo tra un richiamo e il successivo per la registrazione;

Dopo la definizione del costruttore vengono specificati i vari metodi, tra cui il metodo di avvio del Thread, dell'avvio della registrazione, della pausa di essa, del suo riavvio e della fine del Thread, oltre del run() dell'intera classe.

- All'interno del metodo run() si trova l'istruzione: timer.scheduleAtFixedRate(task, 0, dt) che definisce che il task debba essere eseguito ogni qualvolta trascorra la quantità di tempo definita dalla variabile dt, in millisecondi;
- Il metodo startRec() inizializza e avvia il Thread; la varibile state viene posta a 1 ed enable è TRUE;
- Il metodo pauseRec() mette in pausa la registrazione; la varibile state viene posta a 2 ed enable è TRUE;
- Il metodo restartRec() permette di ripartire con la registrazione; la varibile state viene posta a 3 ed enable è TRUE;
- Il metodo stopRec() conclude il Thread; la varibile state viene posta a 4 ed enable è FALSE;
- Il metodo getList() che ritorna l'ArrayList in cui sono state salvate le informazioni;

5.2 Analisi dei risultati

In questa sezione si elaborano i dati raccolti attraverso il software Matlab. Si è cercato di far vedere come il robot riesca a riprodurre la superficie d'interesse.

5.2.1 Studio ripetibilità nei quattro vertici

Come primo step si analizza la raccolta dei punti tramite una semplice discesa del robot sulla superficie, più precisamente una tavoletta di legno. Al contatto con essa il robot memorizza la posizione e la passa al software Matlab. Con questa prima prova si vuole utilizzare il robot come tastatore per studiare la precisione del robot. Precisione che racchiude concetti di accuratezza, ripetibilità e risoluzione. L'accuratezza è la capacità del manipolatore di raggiungere un punto per la prima volta. La ripetibilità è la capacità di tornare in un punto già raggiunto precedentemente. Infine la risoluzione che esprime il movimento più piccolo che il robot può fare senza che esso se ne renda conto, in quanto dipende dal tipo di encoder montato nei motori. L'analisi viene sviluppata impostando da computer i Frame che il robot deve raggiungere, che però, per via della superficie presente come ostacolo, non raggiungerà. Si confrontano i diversi set di posizioni presi da una prova all'altra. Il grafico che si ottiene è il seguente:

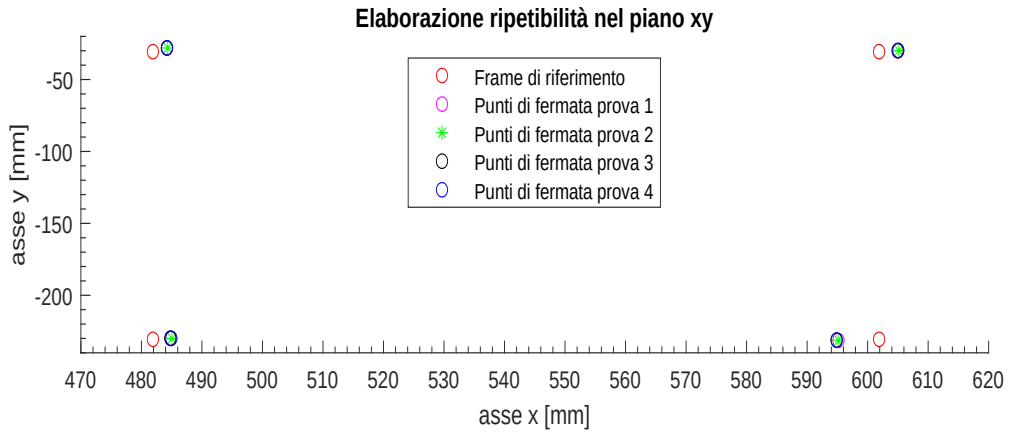


Figura 5.1: Analisi ripetibilità nel piano xy

I punti rappresentati corrispondono ai quattro vertici del percorso rettangolare che è stato impostato per il robot. Essi possono essere punti all'interno di una superficie più ampia, come nel caso studiato, ma anche limiti che il robot non deve superare (ad es. area di lavoro predefinita, limite fisico della superficie da analizzare). Si nota come il robot non si posiziona esattamente sulle coordinate imposte da software. Questa ultima affermazione viene approfondita e confemata dai grafici seguenti, dove si nota che, effettuando quattro prove ripetute, il manipolatore non posiziona mai l'organo terminale nella posizione della prova precedente.

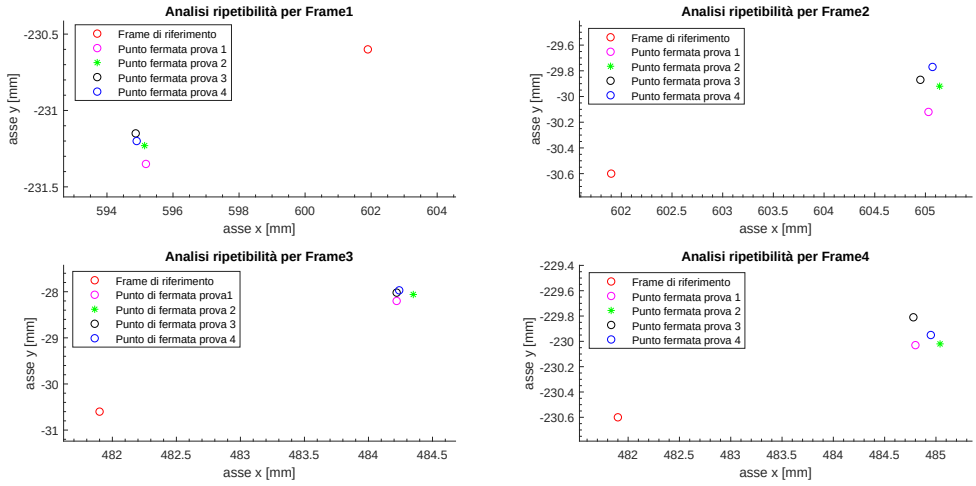


Figura 5.2: Analisi ripetibilità per i vari Frame

Si nota come ci sia un errore nell'accuratezza della misura piuttosto evidente mentre la ripetibilità è piuttosto buona. L'errore di accuratezza, il maggiore tra i due rilevati, può essere dovuto ad una mancata calibrazione cinematica che non è stata eseguita durante la configurazione del robot, quindi il risultato di un errore rilevante era atteso. L'errore di ripetibilità è nell'ordine di decimi di millimetro e anch'esso è coerente con le aspettative in quanto i sensori di forza sono si precisi ma non tali da consentire al robot una ripetibilità più ottimale. Tuttavia i risultati, soprattutto in termini di ripetibilità, sono soddisfacenti. Questo potrebbe anche essere dovuto alle tolleranze che sono presenti dovute alla fisica e meccanica del manipolatore e all'azione dell'impedance control.

In aggiunta a questa analisi si è andati a studiare come sono i piani interpolanti che si generano dai punti ottenuti (vedi Figura 5.3), per verificare se la superficie analizzata sia effettivamente piana o meno. Dalle prove ottenute è emerso che i piani generati dai quattro set di punti diversi risultano sovrapponibili, quindi a conferma che anche in errore di ripetibilità comunque i risultati sono accettabili. Il

piano generato risulta leggermente inclinato rispetto all'orizzontale, a denotare una leggera flessione della superficie analizzata.

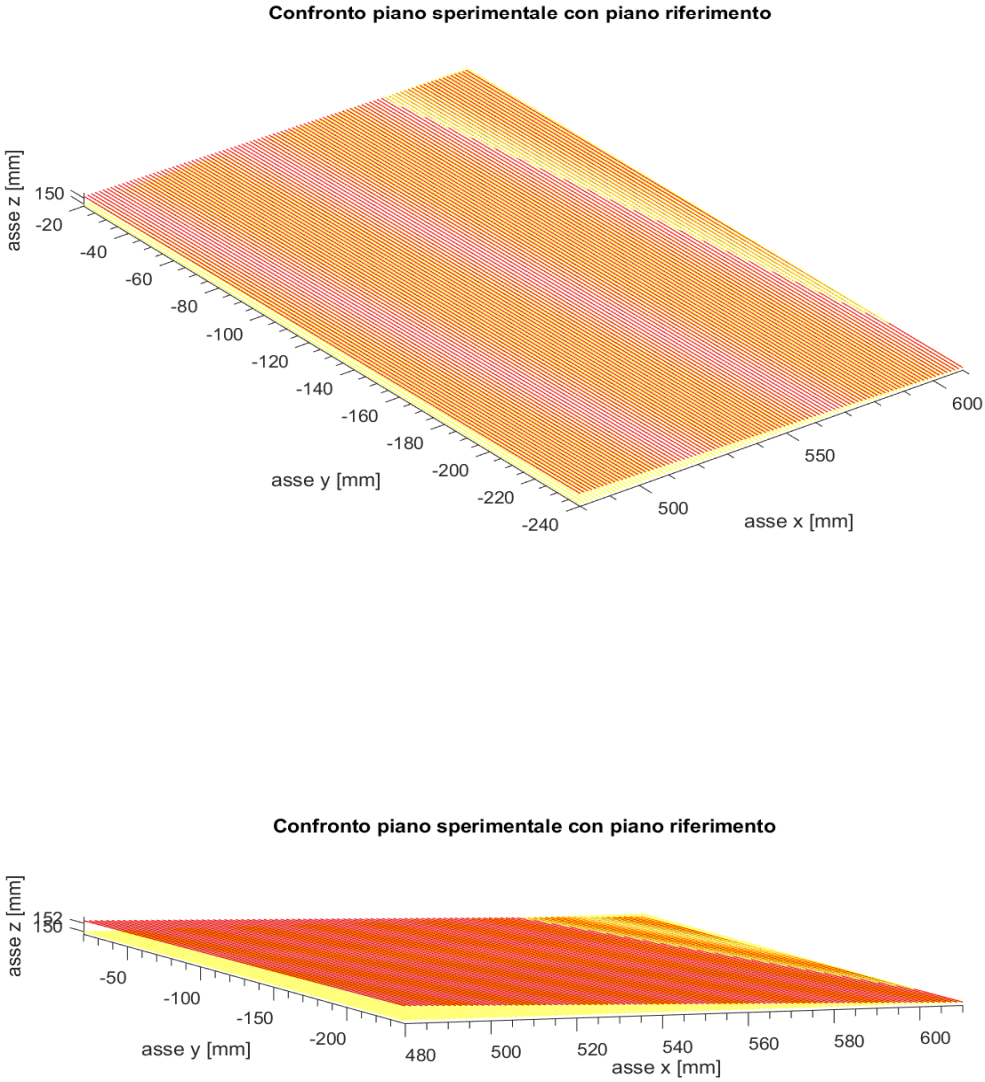


Figura 5.3: Confronto piano ottenuto dai set di punti sperimentali delle prove (rosso) con il piano di riferimento (giallo)

5.2.2 Analisi percorso impostato su piano orizzontale

Si analizza ora la ripetibilità lungo il percorso definito. In particolare si vuole studiare la ripetibilità con due punti di vista differenti: il primo tramite analisi dei grafici Matlab, il secondo attraverso concetti di media e varianza. Come detto durante la descrizione del codice si studia il caso senza operatore e variando i parametri di costante elastica delle molle virtuali con l'Impedance Control.

Sono state eseguite diverse prove di tracciamento del percorso e per ogni prova si sono plottati i punti raccolti dal controllore.

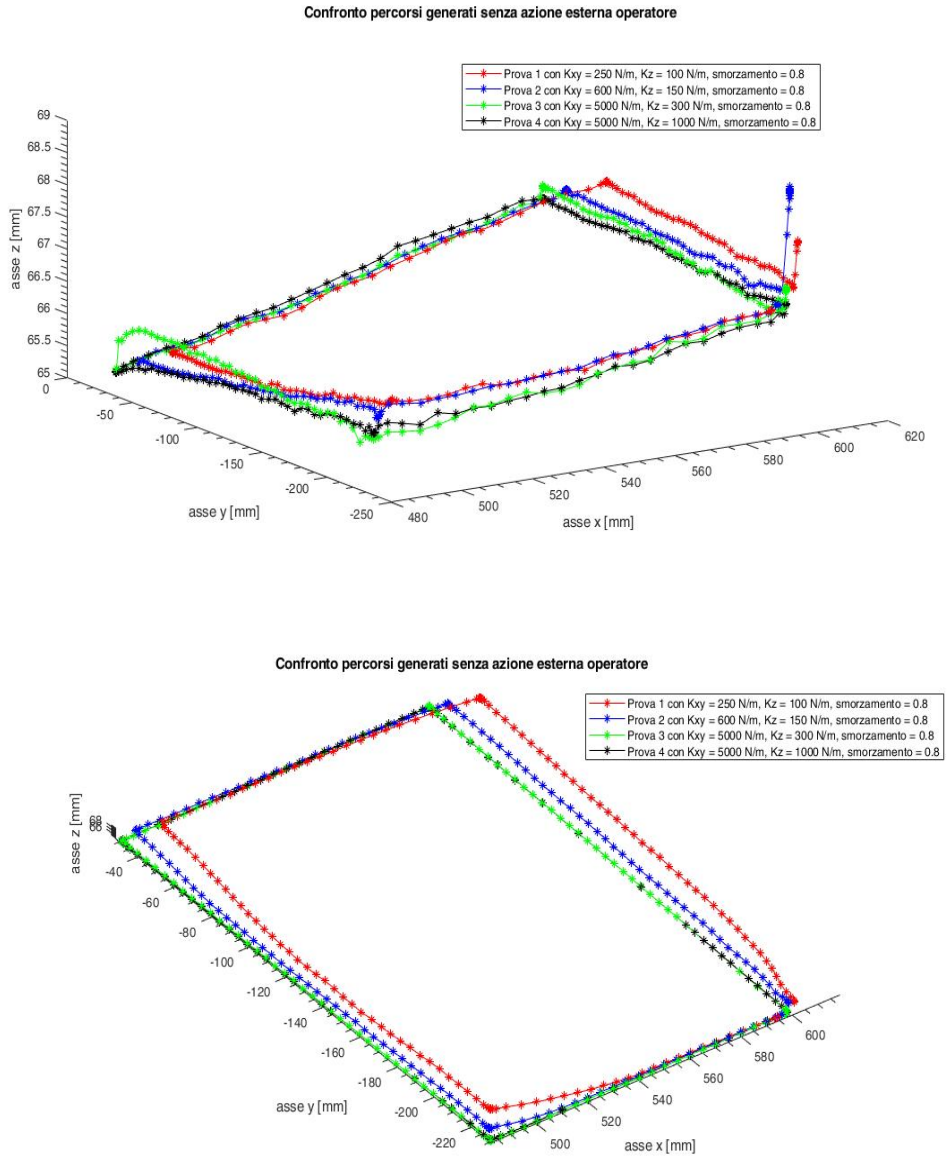
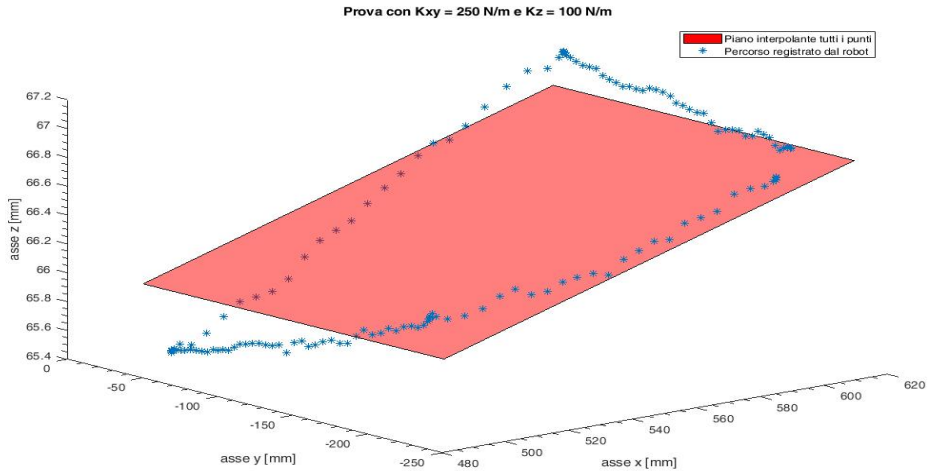


Figura 5.4: Confronto percorsi senza azione dell'operatore

Come si nota dai grafici, nelle quattro prove analizzate variando la costante elastica delle molle virtuali, il robot percorre dei percorsi quasi uguali nella forma ma differenti per punti tracciati. Si vede molto bene come varia la ripetibilità, dovuta oltre che alla sensisbiltà dei sensori di forza, anche dalla sensibilità che viene data al robot tramite la variazione dei parametri dell'Impedance Control. Con delle costanti elastiche elevate il percorso tracciato è rigoroso e segue maggiormente le specifiche di attraversamento dei Frame vertice date dal software. Al diminuire del valore delle costanti elastiche il passaggio per i Frame vertice viene meno e il robot è molto più sensibile e il movimento è più lento e ondeggiante rispetto a costanti elastiche elevate. Si nota che devia il suo percorso pur non essendoci fattori esterni agenti, ci mette più tempo a controllare il soddisfacimento della condizione di forza dovuto al fatto che, avendo K minore, ha bisogno di pigiare di più sulla superficie e questo allunga il tempo ciclo.

Un altro punto di vista potrebbe essere il piano interpolante passante per i punti, diverso da prova a prova. Questo aspetto è interessante perchè permette di capire la differenza tra il piano ipotetico reale ricostruito attraverso i punti raccolti, quindi il piano teorico della superficie, e la superficie reale analizzata dal robot.



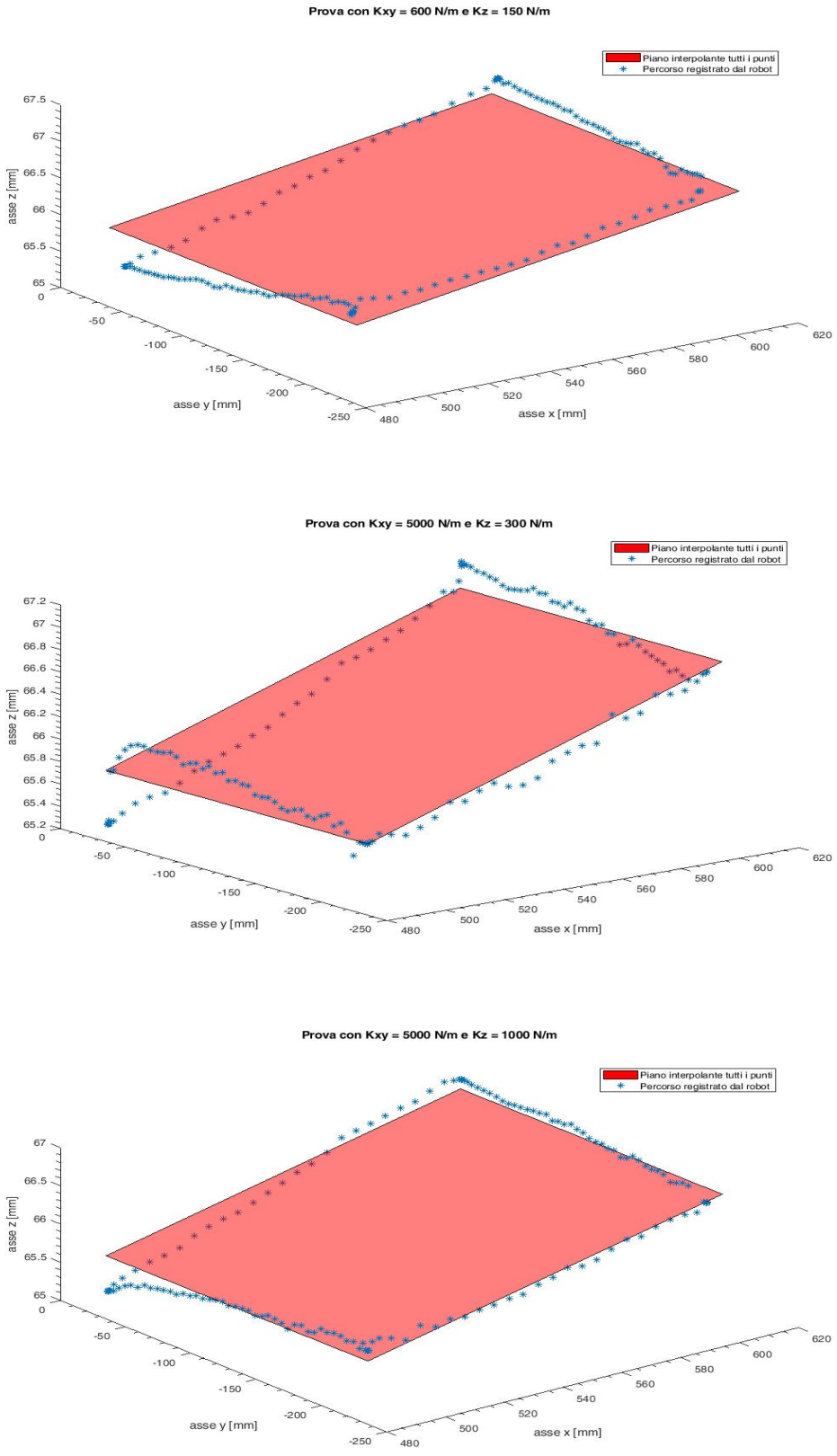


Figura 5.5: Analisi dei piani che interpolano i vari punti per le quattro prove effettuate e confronto con il percorso generato dai dati che il robot ha raccolto

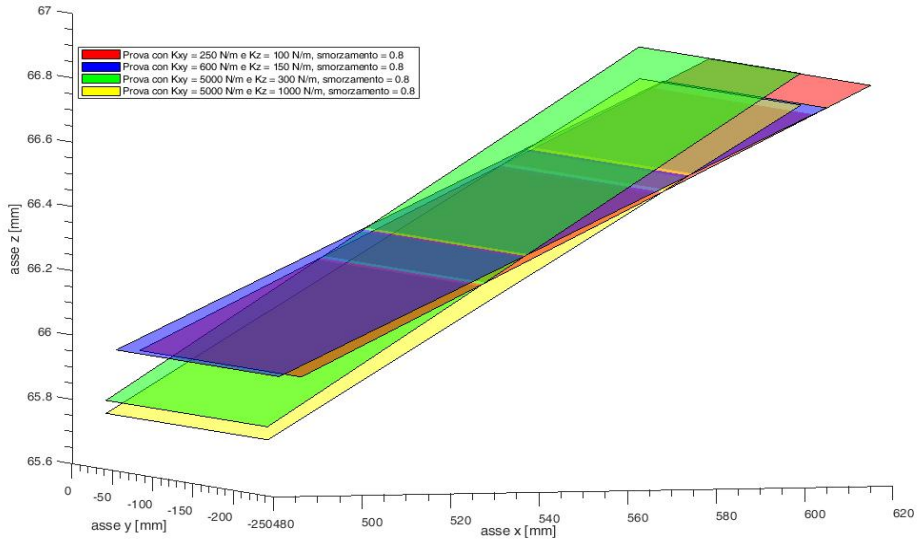


Figura 5.6: Confronto tra i piani interpolanti delle quattro prove svolte

Come si nota dai grafici c'è un po' di differenza nella ricostruzione dei piani. In particolare tra due le prove con costanti elastiche basse e le due prove con costanti elevate. C'è un errore minore tra il piano ricostruito e il percorso tracciato nei casi con costanti elevate ad evidenziare una possibile accuratezza migliore in quanto i valori registrati discostano meno dal piano. L'aspetto negativo però lo si riscontra nel fatto che il robot, con queste impostazioni, è meno sensibile alle variazioni della superficie. Infatti, con costanti minori, il percorso registrato ha un errore maggiore con il piano interpolante, ma potrebbe essere dovuto al fatto che il robot segue meglio la superficie privileggiando la sensibilità. Il percorso inoltre, come visto anche nei grafici precedenti, varia la sua forma in base alla taratura delle molle: tende a essere più rettangolare e rigoroso con costanti elastiche elevate mentre più sinuoso con costanti piccole.

Questo aspetto può essere visto da punti di vista differenti e in base alle necessità di cui si ha bisogno in una determinata applicazione: da un lato, se si vuole una migliore ripetibilità e accuratezza allora si tendono ad aumentare le costanti elastiche e quindi a rendere più rigido il manipolatore (e questo può servire in applicazioni in cui è richiesta una precisione accurata ad esempio); dall'altro, con costanti elastiche minori si ha una maggiore sensibilità del manipolatore a copiare e seguire le imperfezioni della superficie che gli viene presentata davanti, con una fedeltà di riproduzione maggiore (e questo è servito nel caso di questa tesi, ma oltre a ciò in diverse applicazioni collaborative in cui non ci devono essere contatti bruschi tra robot e operatore ma anche tra robot e attrezzature).

5.2.3 Analisi percorso impostato su piano inclinato

Tutto quello visto fino ad ora vale per superfici su un piano orizzontale, tuttavia la stessa analisi può essere fatta anche su piani inclinati, di diversa forma e geometria, impostando correttamente i parametri di Impedance Control. Questo in base al tipo di applicazione che si vuole studiare e ai punti di interesse. Come per il caso precedente, vengono analizzati prima i percorsi registrati dal robot e successivamente si generano i piani interpolanti passanti per mettere in luce il piano della superficie.

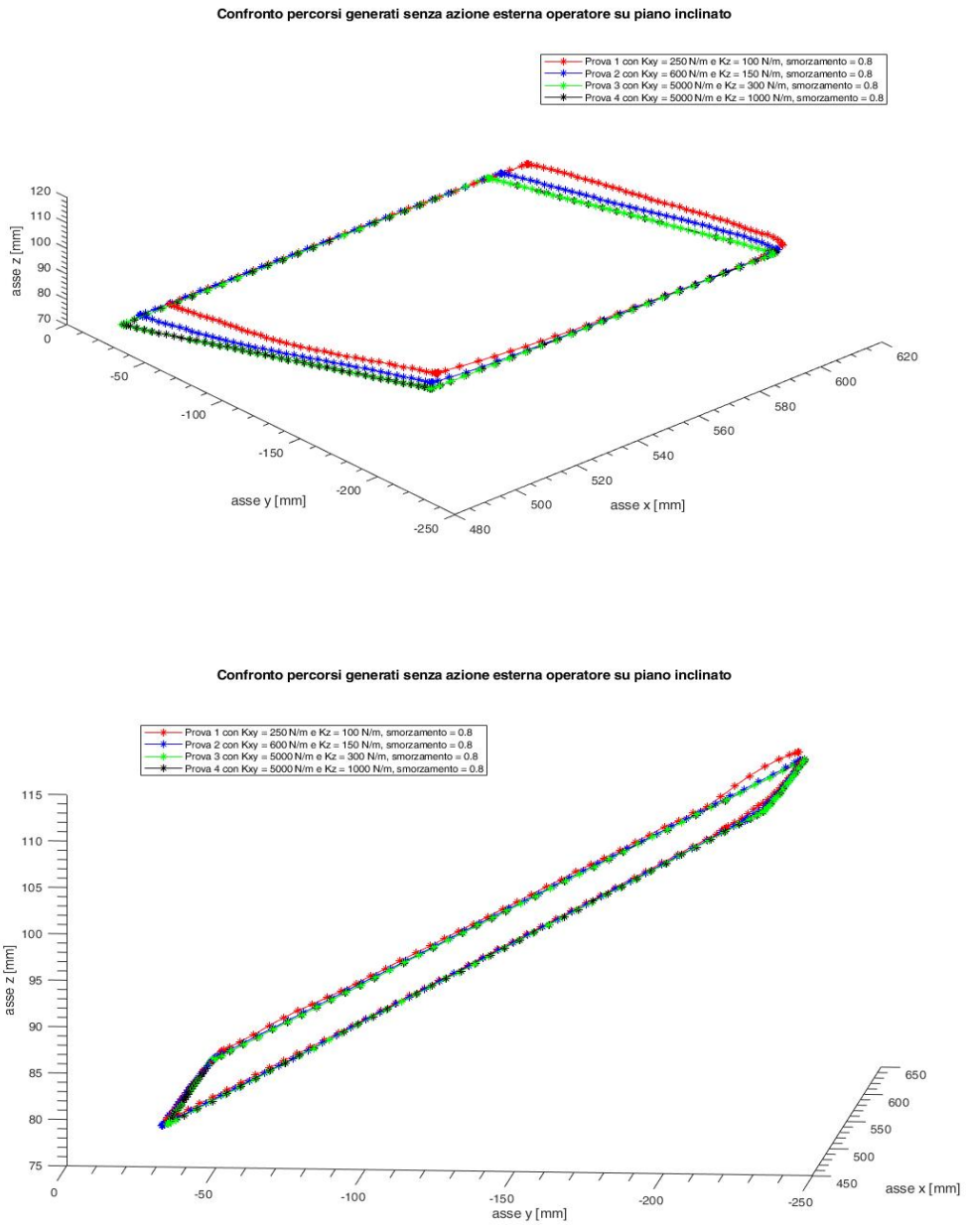
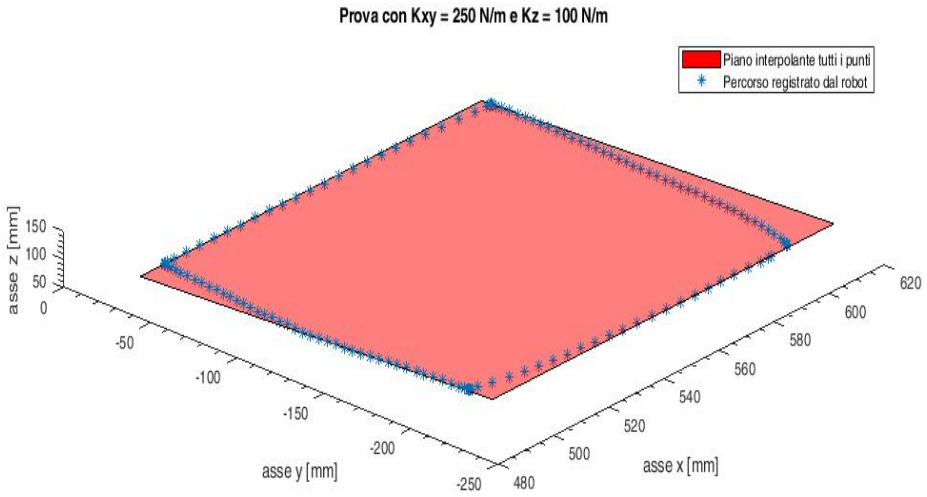
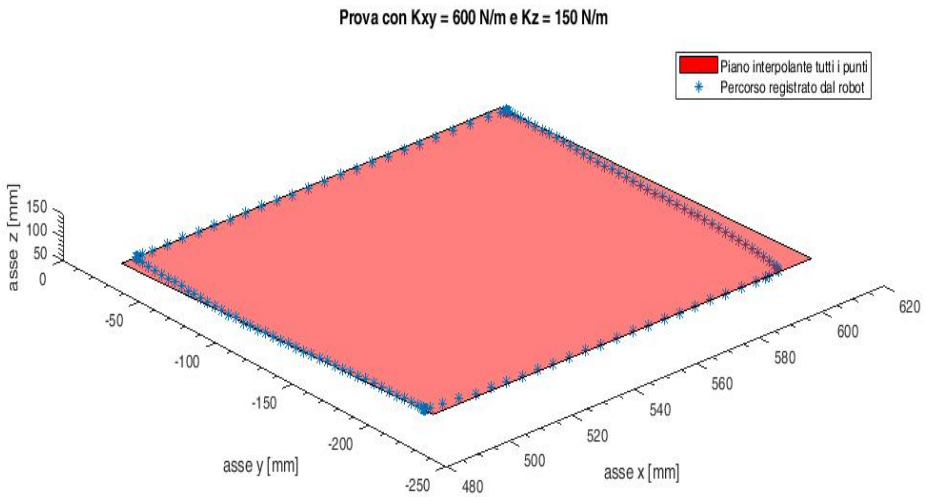
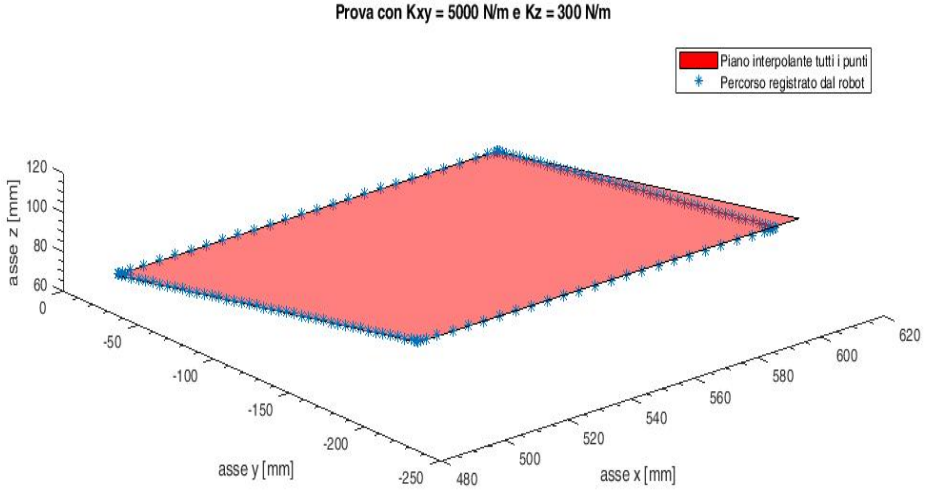


Figura 5.7: Confronto percorsi senza azione dell'operatore con piano inclinato


Prova con Kxy = 250 N/m e Kz = 100 N/m
Piano interpolante tutti i punti
Percorso registrato dal robot
asse z [mm]
150
100
50
0
-50
-100
-150
-200
-250
asse y [mm]
480
500
520
540
560
580
600
620
asse x [mm]


Prova con Kxy = 600 N/m e Kz = 150 N/m
Piano interpolante tutti i punti
Percorso registrato dal robot
asse z [mm]
150
100
50
0
-50
-100
-150
-200
-250
asse y [mm]
480
500
520
540
560
580
600
620
asse x [mm]


Prova con Kxy = 5000 N/m e Kz = 300 N/m
Piano interpolante tutti i punti
Percorso registrato dal robot
asse z [mm]
120
100
80
60
0
-50
-100
-150
-200
-250
asse y [mm]
480
500
520
540
560
580
600
620
asse x [mm]

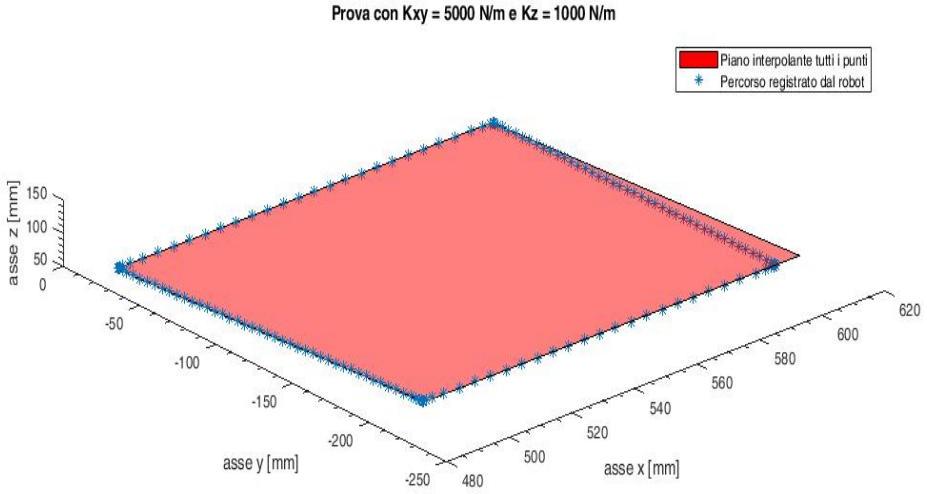


Figura 5.8: Analisi dei piani che interpolano i vari punti per le quattro prove effettuate e confronto con il percorso generato dai dati che il robot ha raccolto

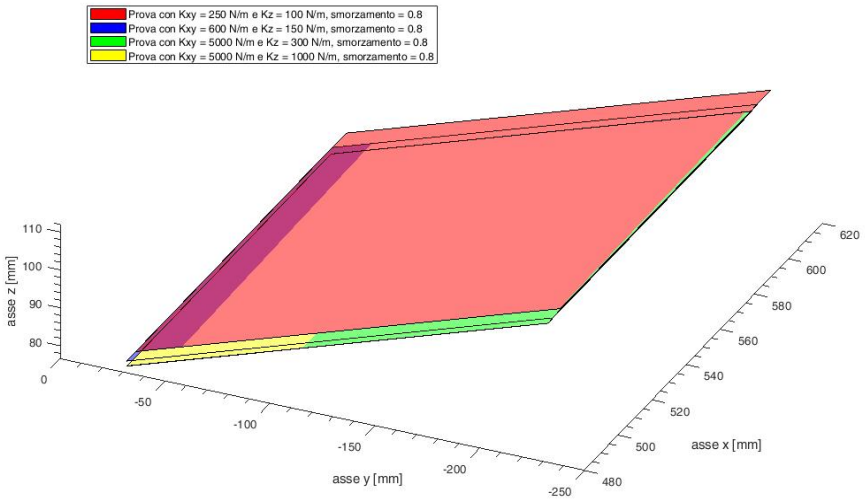


Figura 5.9: Confronto tra i piani interpolanti delle quattro prove svolte

Come si nota dai grafici, i dati raccolti prova dopo prova sono molto simili tra loro. Non si riescono ad apprezzare le differenze e gli errori di accuratezza e ripetibilità come nel piano orizzontale in quanto la scala delle z è maggiore e non permette di apprezzare la ripetibilità. Tuttavia si sono ottenuti risultati soddisfacenti anche a fronte della mancata calibrazione cinematica del robot e dalla superficie analizzata non perfettamente orizzontale ma leggermente fuori asse.

5.2.4 Analisi percorso impostato orizzontale con azione esterna

In questa sezione infine si va ad analizzare come varia il percorso del robot se vengono applicate forze esterne all'organo terminale. Viene mostrato un esempio, senza focus su ripetibilità o accuratezza ma appunto per far vedere cosa provoca l'interazione con l'operatore che tocca il robot.

In base alle specifiche date dal codice e dalla quantità di volte che si va a contatto con il robot il percorso cambia come mostrato nei seguenti grafici.

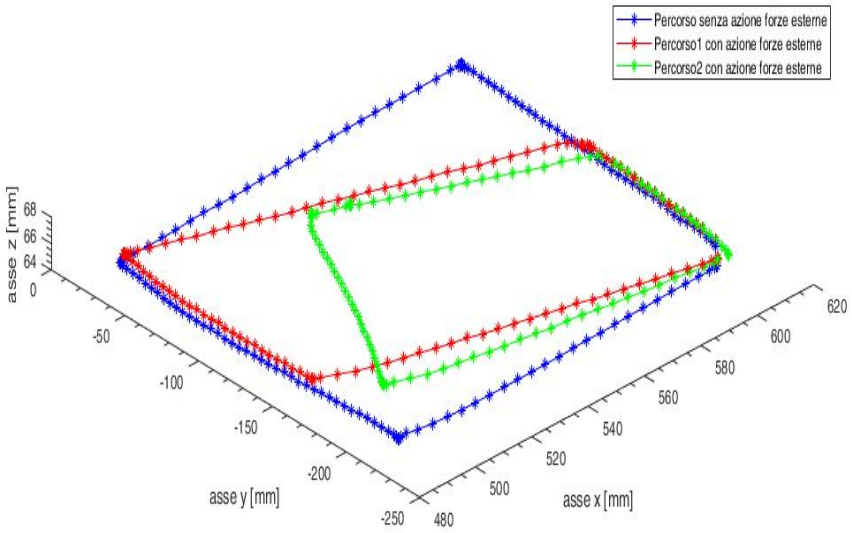


Figura 5.10: Confronto del percorso con Kxy = 600 N/m e Kz = 150 N/m

Come si può notare, più i contatti sono frequenti e più cambia il percorso disegnato dal robot. In particolare si nota come, al punto di applicazione della forza, corrisponda un deviazione verso il frame vertice successivo. Il percorso non sarà più un rettangolo ma una figura di quattro lati. Da notare come, visto la presenza dell'Impedance Control, il robot cambia percorso solo se vengono realizzate le condizioni di forza imposte via software. Infatti, se le forze esterne fossero presenti ma non sufficientemente elevate, il robot devierebbe solamente il suo tratto raggiungen-

do comunque il punto prestabilito, a conferma della bontà dell'Impedance Control, come si vede nel grafico a seguire.

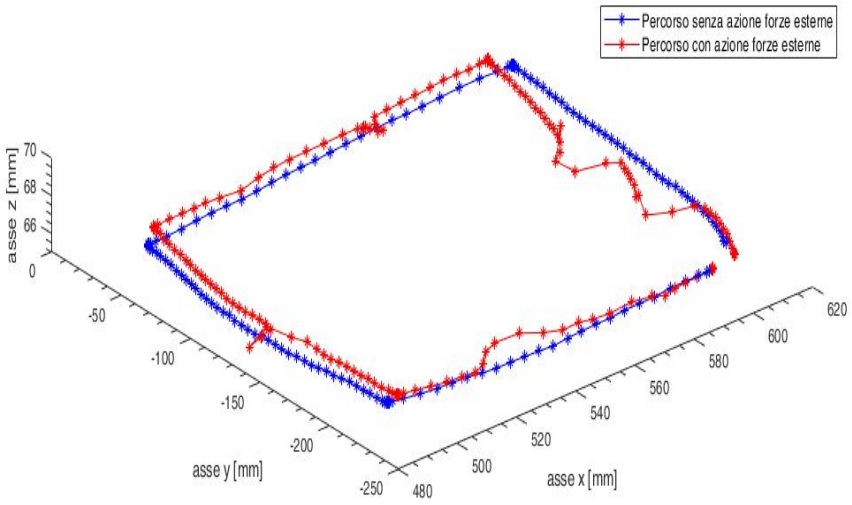


Figura 5.11: Confronto del percorso con Kxy = 250 N/m e Kz = 100 N/m

Capitolo 6

Commenti conclusivi

Con questo elaborato si è cercato di riprodurre uno dei possibili ambiti applicativi dei cobots. In particolare si è implementato un task che, tramite il controllo d impedenza, permette al robot di copiare il profilo di una superficie e, tramite un Thread secondario, la registrazione delle posizioni elaborate successivamente tramite Matlab, instaurando una comunicazione tramite socket tra il controllore del robot e Matlab. L'applicativo sfrutta la modalità di impedance control, la quale permette di impostare la cedevolezza del robot durante l'esecuzione di un movimento. La rilevazione di una superficie per punti è realizzata monitorando le forze scambiate tra robot e ambiente, nonché tra robot ed operatore. I risultati si sono rivelati soddisfacenti a fronte della risoluzione di misura dei sensori del robot e di eventuali disturbi esterni che ne hanno influenzato il movimento. Possibili ambiti applicativi possono essere: la robotica collaborativa in generale o comunque applicazioni che richiedono un supporto umano all'azione del robot, come per esempio l'assemblaggio di macchine o pezzi, un pick&place evoluto con l'interazione uomo macchina sulla stessa postazione o linea di assemblaggio, riconoscimento e riproduzione di oggetti particolari.

Capitolo 7

Appendice

Programma principale

```
package application;


import static com.kuka.roboticsAPI.motionModel.BasicMotions.ptp;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.ptpHome;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.lin;

import javax.inject.Inject;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import com.kuka.roboticsAPI.motionModel.IMotionContainer;
import com.kuka.roboticsAPI.motionModel.controlModeModel.CartesianImpedanceControlMode;
import com.kuka.roboticsAPI.deviceModel.JointEnum;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.deviceModel.PositionInformation;
import com.kuka.roboticsAPI.geometricModel.*;
import com.kuka.roboticsAPI.geometricModel.math.CoordinateAxis;
import com.kuka.roboticsAPI.deviceModel.JointPosition;
import com.kuka.roboticsAPI.executionModel.IFiredConditionInfo;
import com.kuka.roboticsAPI.conditionModel.ForceComponentCondition;
import com.kuka.roboticsAPI.conditionModel.ForceCondition;
import com.kuka.roboticsAPI.controllerModel.Controller;

import com.kuka.common.ThreadUtil;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.List;


public class ImpedanceControl extends RoboticsAPIApplication {

// Definizione delle variabili di utilizzo principali

@Inject
private LBR lBR_iiwa_14_R820_1;
private JointPosition newHome;
public Controller kukaController;

private IMotionContainer motionCmd0, motionCmd1, motionCmd2, motionCmd3, motionCmd1a, motionCmd2a, motionCmd3a;
private Frame firedCurrPos0, firedCurrPos1, firedCurrPos2, firedCurrPos3, firedCurrPos1a, firedCurrPos2a, firedCurrPos3a;
private IFiredConditionInfo firedCondInfo0, firedCondInfo1, firedCondInfo2, firedCondInfo3, firedCondInfo1a, firedCondInfo2a, firedCondInfo3a;
private PositionInformation firedPosInfo0, firedPosInfo1, firedPosInfo2, firedPosInfo3, firedPosInfo1a, firedPosInfo2a, firedPosInfo3a;

private NicolaTrackRec  rec;
private ArrayList<byte[]> _trackPoints;


@Override
public void initialize()
{
// Inizializzazione variabili

kukaController = (Controller) getContext().getControllers().toArray()[0];

newHome = new JointPosition(Math.toRadians(1.41),Math.toRadians(23.58),Math.toRadians(-0.93),Math.toRadians(-87.08),
Math.toRadians(0.46),Math.toRadians(69.35),Math.toRadians(0.40));

// Definizione della lista dove contenere le info sulle posizioni
_trackPoints = new ArrayList<byte[]>();

// Definizione della nuova istanza di registrazione per inizializzare il Thread
rec = new NicolaTrackRec(lBR_iiwa_14_R820_1,_trackPoints,getLogger());
}

// Serve per rendere interattiva l'azione con il robot, per farlo partire bisogna che l'operatore eserciti una coppia esterna
```

```
public void ManualMotion()
{
// Posizionamento robt in home
lBR_iiwa_14_R820_1.setHomePosition(newHome);
lBR_iiwa_14_R820_1.move(ptpHome().setJointVelocityRel(0.3));
lBR_iiwa_14_R820_1.move(ptp(getApplicationData().getFrame("/HomePos")).setJointVelocityRel(0.3));

for (int j = 0; j < 1; j++)
{
// Inizio movimentazione
getLogger().info("Starting motion...");

getLogger().info("Waiting for torque");
// Attesa di coppia per lo spostamento
double tq = lBR_iiwa_14_R820_1.getExternalTorque().getSingleTorqueValue(JointEnum.J6);
getLogger().info(tq+"");
while (Math.abs(tq) < 1)
{
tq = lBR_iiwa_14_R820_1.getExternalTorque().getSingleTorqueValue(JointEnum.J6);
}

if (tq>0)
{
getLogger().info("Partenza");
lBR_iiwa_14_R820_1.move(ptp(getApplicationData().getFrame("/Partenza")).setJointVelocityRel(0.2));
}
else
{
getLogger().info("Home");
lBR_iiwa_14_R820_1.move(ptp(getApplicationData().getFrame("/HomePos")).setJointVelocityRel(0.2));
}
}
}

@Override
public void run()
{
// Metodo di esecuzione
// Definizione condizione di stop per quando si tocca la superficie
ForceCondition cond = ForceCondition.createSpatialForceCondition(lBR_iiwa_14_R820_1.getFlange(),4.0);

ForceComponentCondition cond1 = new ForceComponentCondition(lBR_iiwa_14_R820_1.getFlange(),CoordinateAxis.Y,-5.0, 20.0);

ForceComponentCondition cond2 = new ForceComponentCondition(lBR_iiwa_14_R820_1.getFlange(),CoordinateAxis.X,-5.0, 200.0);

ForceComponentCondition cond3 = new ForceComponentCondition(lBR_iiwa_14_R820_1.getFlange(),CoordinateAxis.Y,-200.0, 5.0);

//ForceComponentCondition cond5 = new ForceComponentCondition(lBR_iiwa_14_R820_1.getFlange(),CoordinateAxis.X,-4.0, 5.0);

// Definizione parametri impedance control
CartesianImpedanceControlMode cartImpCtrlMode = new CartesianImpedanceControlMode();
cartImpCtrlMode.parametrize(CartDOF.X,CartDOF.Y).setStiffness(250.0);
cartImpCtrlMode.parametrize(CartDOF.Z).setStiffness(100.0);
cartImpCtrlMode.parametrize(CartDOF.ALL).setDamping(0.8);  // ex 0.6

ManualMotion();


/*UTILIZZO PRIMO APPROCCIO PER SALVARE I PUNTI CHE DIVENTERANNO I VERTICI DEL RETTANGOLO CHE IL ROBOT PERCORRERA' SUCCESSIVAMENTE*/

lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P3")).setCartVelocity(20).breakWhen(cond));

// Settaggio per capire dove si ferma il robot
motionCmd0 = lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P3")).setCartVelocity(20).breakWhen(cond));
firedCondInfo0 = motionCmd0.getFiredBreakConditionInfo();
if(firedCondInfo0 != null)
{
firedPosInfo0 = firedCondInfo0.getPositionInfo();
firedCurrPos0 = firedPosInfo0.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos0);
}

// Movimento verso Step1, definito sulla verticale di P3
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step1")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Movimento verso Step2 lungo Y, definito sulla verticale di P4
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step2")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Approccio verso P4
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P4")).setCartVelocity(20).breakWhen(cond).setMode(cartImpCtrlMode));
motionCmd1 = lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P4")).setCartVelocity(20).breakWhen(cond).setMod(cartImpCtrlMode));
firedCondInfo1 = motionCmd1.getFiredBreakConditionInfo();
if(firedCondInfo1 != null)
{
firedPosInfo1 = firedCondInfo1.getPositionInfo();
firedCurrPos1 = firedPosInfo1.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos1);
}

// Movimento verso Step2, definito sulla verticale di P4
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step2")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Movimento verso Step3 lungo X, definito sulla verticale di P5
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step3")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Approccio verso P5
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P5")).setCartVelocity(20).breakWhen(cond).setMode(cartImpCtrlMode));
motionCmd2 = lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P5")).setCartVelocity(20).breakWhen(cond).setMod(cartImpCtrlMode));
firedCondInfo2 = motionCmd2.getFiredBreakConditionInfo();
if(firedCondInfo2 != null)
{
firedPosInfo2 = firedCondInfo2.getPositionInfo();
firedCurrPos2 = firedPosInfo2.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos2);
}

// Movimento verso Step3, definito sulla verticale di P5
```

```
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step3")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Movimento verso Step4 lungo Y, definito sulla verticale di P6
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step4")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Approccio verso P6
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P6")).setCartVelocity(20).breakWhen(cond).setMode(cartImpCtrlMode));
motionCmd3 = lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/P6")).setCartVelocity(20).breakWhen(cond).setMode(cartImpCtrlMode));
firedCondInfo3 = motionCmd3.getFiredBreakConditionInfo();
if(firedCondInfo3 != null)
{
firedPosInfo3 = firedCondInfo3.getPositionInfo();
firedCurrPos3 = firedPosInfo3.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos3);
}

// Movimento verso Step4, definito sulla verticale di P6
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step4")).setCartVelocity(60).setMode(cartImpCtrlMode));

// Ritorno in Step1
lBR_iiwa_14_R820_1.move(lin(getApplicationData().getFrame("/Step1")).setJointVelocityRel(0.3));



/*RITORNO NEL PRIMO PUNTO E INIZIO A PERCORRERE LA SUPERFICIE*/

// Movimento verso prima posizione rilevata
lBR_iiwa_14_R820_1.move(lin(firedCurrPos0).setCartVelocity(20).setMode(cartImpCtrlMode));


rec.start(); //fa partire l'esecuzione del Thread
rec.startRec(); //fa partire la registrazione dei dati
ThreadUtil.milliSleep(1000); //serve per mettere in pausa il Thread

// Movimento verso seconda posizione rilevata
lBR_iiwa_14_R820_1.move(lin(firedCurrPos1).setCartVelocity(20).breakWhen(cond1).setMode(cartImpCtrlMode));

// Se durante il movimento il robot sente una forza esterna che agisce allora si ferma e cambia direzione,
altrimenti raggiunge la posizione prestabilita
motionCmd1a = lBR_iiwa_14_R820_1.move(lin(firedCurrPos1).setCartVelocity(20).breakWhen(cond1).setMode(cartImpCtrlMode));
firedCondInfo1a = motionCmd1a.getFiredBreakConditionInfo();
if (firedCondInfo1a != null)
{
firedPosInfo1a = firedCondInfo1a.getPositionInfo();
firedCurrPos1a = firedPosInfo1a.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos1a);
}
else
lBR_iiwa_14_R820_1.move(lin(firedCurrPos1).setCartVelocity(20).setMode(cartImpCtrlMode));

// Movimento verso terza posizione rilevata
lBR_iiwa_14_R820_1.move(lin(firedCurrPos2).setCartVelocity(20).breakWhen(cond2).setMode(cartImpCtrlMode));
motionCmd2a = lBR_iiwa_14_R820_1.move(lin(firedCurrPos2).setCartVelocity(20).breakWhen(cond2).setMode(cartImpCtrlMode));
firedCondInfo2a = motionCmd2a.getFiredBreakConditionInfo();
if(firedCondInfo2a != null)
{
firedPosInfo2a = firedCondInfo2a.getPositionInfo();
firedCurrPos2a = firedPosInfo2a.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos2a);
}
else
lBR_iiwa_14_R820_1.move(lin(firedCurrPos2).setCartVelocity(20).setMode(cartImpCtrlMode));

// Movimento verso quarta posizione rilevata
lBR_iiwa_14_R820_1.move(lin(firedCurrPos3).setCartVelocity(20).breakWhen(cond3).setMode(cartImpCtrlMode));
motionCmd3a = lBR_iiwa_14_R820_1.move(lin(firedCurrPos3).setCartVelocity(20).breakWhen(cond3).setMode(cartImpCtrlMode));
firedCondInfo3a = motionCmd3a.getFiredBreakConditionInfo();
if(firedCondInfo3a != null)
{
firedPosInfo3a = firedCondInfo3a.getPositionInfo();
firedCurrPos3a = firedPosInfo3a.getCurrentCartesianPosition();
getLogger().info("Posizione attuale cartesiana:" + firedCurrPos3a);
}
else
lBR_iiwa_14_R820_1.move(lin(firedCurrPos3).setCartVelocity(20).setMode(cartImpCtrlMode));

// Movimento verso prima posizione rilevata e conclusione del percorso
lBR_iiwa_14_R820_1.move(lin(firedCurrPos0).setCartVelocity(20).setMode(cartImpCtrlMode));

// Conclusione della registrazione
rec.stopRec();

// Ritorno in partenza
lBR_iiwa_14_R820_1.move(ptp(getApplicationData().getFrame("/Step1")).setJointVelocityRel(0.2));

// Ritorno in home
lBR_iiwa_14_R820_1.move(ptp(getApplicationData().getFrame("/HomePos")).setJointVelocityRel(0.3));


// Ritorno dell'ArrayList
_trackPoints = rec.getList();

getLogger().info("dimensione" + _trackPoints.size());

if(_trackPoints.isEmpty()==true)
{
getLogger().info("array vuoto");
}
else
{
// Comunicazione tramite socket

double limit = 1000;
long date1 = 1;

ServerSocket server = null;
try{
```

```
// response
byte[] ok = {1};
byte[] input = new byte[8];

server= new ServerSocket(30003);
System.out.println("Server started. Listening to the port 30003");
server.setSoTimeout(50000); // attesa avvio matlab
Socket serverClient=server.accept();
System.out.println("Matlab connected");
DataOutputStream outStream=new DataOutputStream(serverClient.getOutputStream());
DataInputStream inStream=new DataInputStream(serverClient.getInputStream());
outStream.write(ok);

inStream.read(input);
System.out.println(_trackPoints);
byte[] messageb = new byte[24];
for(int i=0;i<_trackPoints.size();i++){
inStream.read(ok);
messageb = _trackPoints.get(i);
outStream.write(messageb);
System.out.println("inviato: " + messageb);
}

byte[] exit = combineArray(10000,10000,10000);
outStream.write(exit);
Thread.sleep(800);
inStream.close();
outStream.close();
serverClient.close();
server.close();
System.out.println("Closed socket");
}
catch(Exception e){
System.out.println(e);
if (!server.isClosed()){
try {
server.close();
System.out.println("Closing socket");
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}

}
}

}
}
public static byte[] combineArray(double x, double y, double z) {
byte[] a = new byte[8];
ByteBuffer.wrap(a).putDouble(x);
byte[] b = new byte[8];
ByteBuffer.wrap(b).putDouble(y);
byte[] c = new byte[8];
ByteBuffer.wrap(c).putDouble(z);
int length = a.length+b.length+c.length;
byte[] combined = new byte[length];
System.arraycopy(a, 0, combined, 0, a.length);
System.arraycopy(b, 0, combined, a.length, b.length);
System.arraycopy(c, 0, combined, a.length+b.length, c.length);
return combined;
}


public static void main(String[] args) {
ImpedanceControl app = new ImpedanceControl();
app.runApplication();
}
}
```

Thread secondario

```
package application;

import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;

import com.kuka.common.ThreadUtil;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.task.ITaskLogger;


public class NicolaTrackRec extends Thread{

// Definizione variabili
private ITaskLogger logger;
private LBR lBR_iiwa_14_R820_1;
private Boolean enable = true;
private long dt = 250; //intervallo di tempo di pausa che fa il Thread tra una acquisizione e l'altra
private int state;
private Timer timer;
byte[] messagei = new byte[24];
private ArrayList<byte[]> theList;

private TimerTask task = new TimerTask(){
@Override
public void run() {

switch(state) {
case 1: // Registrazione della posizione attuale cartesiana del robot e salvataggio dell'infirmazione nell'array theList
Frame pos = lBR_iiwa_14_R820_1.getCurrentCartesianPosition(lBR_iiwa_14_R820_1.getFlange());
```

```
double x = pos.getX();
double y = pos.getY();
double z = pos.getZ();
messagei = combineArray(x, y, z);
theList.add(messagei);
break;
case 2: // Pausa
logger.info("Recording Paused");

break;
case 3: // Ripresa registrazione dopo pausa e salvataggio
logger.info("Recording Restart");
pos = lBR_iiwa_14_R820_1.getCurrentCartesianPosition(lBR_iiwa_14_R820_1.getFlange());
x = pos.getX();
y = pos.getY();
z = pos.getZ();
messagei = combineArray(x, y, z);
theList.add(messagei);
state = 1;
break;
case 4: // Stop
timer.cancel();
logger.info("Timer Cancelled");
break;

default:
// code block
}
}
};

/**
* Costruttore della classe NicolaTrackRec, utilizzata per raccogliere in background i dati di posizione del robot.
* @param _lbriiwa Variabile LBR che definisce il robot di cui si vuole raccogliere i dati.
* @param _trackPoints   ArrayList di Frame dal thread principale.
* @param visual Logger dal Thread principale per la visualizzazione a schermo dei dati raccolti.
*/
public NicolaTrackRec (LBR _lbriiwa, ArrayList<byte[]> _trackPoints,ITaskLogger visual) {
lBR_iiwa_14_R820_1 = _lbriiwa;
theList = _trackPoints;
logger = visual;
timer = new Timer();
}

@Override
public void run()
{
// Esecuzione del Thread
timer.scheduleAtFixedRate(task, 0, dt);
}

// Metodo di inizio
public void startRec(){
logger.info("Recording Started");
state = 1;
enable = true;
}

// Metodo di pausa
public void pauseRec(){
state = 2;
//logger.info("Recording Paused");
enable = true;
}

// Metodo di ripresa
public void restartRec(){
//logger.info("Recording Restarted");
state = 3;
enable = true;
}

// Metodo fermare la registrazione dei dati
public void stopRec(){
state = 4;
enable = false;
}


// Metodo che ritorna l'ArrayList in cui sono state salvate le informazioni
public ArrayList<byte[]> getList() {
return theList;
}

public static byte[] DoutoByte(double x) {
byte[] b = new byte[8];
ByteBuffer.wrap(b).putDouble(x);
//Byte[] B = new Byte[b.length];
//for (int i = 0; i < b.length; i++){
//B[i] = Byte.valueOf(b[i]);
//}
return b;
}

// Metodo che permette di combinare l'array di byte date le informazioni ricavate dalla lettura della posizione del robot
public static byte[] combineArray(double x, double y, double z) {
byte[] a = new byte[8];
ByteBuffer.wrap(a).putDouble(x);
byte[] b = new byte[8];
ByteBuffer.wrap(b).putDouble(y);
byte[] c = new byte[8];
ByteBuffer.wrap(c).putDouble(z);
int length = a.length+b.length+c.length;
byte[] combined = new byte[length];
System.arraycopy(a, 0, combined, 0, a.length);
System.arraycopy(b, 0, combined, a.length, b.length);
System.arraycopy(c, 0, combined, a.length+b.length, c.length);
```

```
return combined;
}


// Metodo che ottiene le informazioni sulla posizione del robot e visualizza le informazioni sul display dello smartpad
public void displayInfo(){
Frame ActualCartesianPosition = lBR_iiwa_14_R820_1.getCurrentCartesianPosition(lBR_iiwa_14_R820_1.getFlange());
double x = ActualCartesianPosition.getX();
double y = ActualCartesianPosition.getY();
double z = ActualCartesianPosition.getZ();

logger.info(

"␣␣␣␣␣␣␣x=" + String.valueOf(x) +
"␣␣␣␣␣␣␣y=" + String.valueOf(y) +
"␣␣␣␣␣␣␣z=" + String.valueOf(z));
}
}
```

Bibliografia

[1] Guida "KUKA Sunrise.OS 1.13, KUKA Sunrise.Workbench 1.13 Operating and Programming Instructions for System Integrators"

[2] Dispensa "Position-Based Kinematics for 7-DoF Serial Manipulators with Global Configuration Control, Joint Limit and Singularity Avoidance" di Carlos Fariaa, Flora Ferreirab, Wolfram Erlhagenc, Sergio Monteiroa, Estela Bichoa, https://en.wikipedia.org/wiki/Rodrigues

[3] https://www.kuka.com

[4] ttps://it.wikipedia.org

[5] ttp://deformazione.it/2018/12/11/la-transizione-dai-robot-collaborativi-alle-applicazioni-collaborative

[6] ttps://www.universal-robots.com

[7] ttps://webthesis.biblio.polito.it

[8] ttp://www.tecnofocus.it/robot-e-cobot-le-differenze-sostanziali

[9] ttp://www.automazione.ingre.unimore.it/pages/corsi/materialedidattico/Controllo

[10] ttps://en.wikipedia.org/wiki/RemoteCenterCompliance

[11] ttp://www.lar.deis.unibo.it/people/cmelchiorri/FilesRobotica/ControlloForza.pdf

[12] ttp://www.ladispe.polito.it/corsi/meccatronica/download/ControlloManipolatoriIndustriali

[13] ttp://www.diag.uniroma1.it/ deluca/rob2en/ImpedanceControl.pdf

[14] Appunti di robotica industriale

[15] https://www.github.com

[16] ttps://www.robotforum.com

[17] http://www.diit.unict.it/users/alongheu/linguaggi/lezione14thread.pdf

[18] https://informaticabrutta.it/socket