UNIVERSITÀ DEGLI STUDI DI PADOVA

—

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

—

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCANICA

OTTIMIZZAZIONE DI TRAIETTORIE DI UN MANIPOLATORE INDUSTRIALE RIDONDANTE

RELATORE: CH.MO PROF. ING. GIULIO ROSATI
CORRELATORE: CH.MO ING. MATTEO BOTTIN

LAUREANDO: MARTINI NICOLA

ANNO ACCADEMICO 2019-2020

Alla mia famiglia,

con affetto e gratitudine.

Indice

Sommario

La pianificazione di traiettorie nello spazio operativo è un'operazione richiesta molto spesso per applicazioni industriali che implichino un'elevata precisione nel movimento dell'*end-effector* di un manipolatore robotizzato, come ad esempio la scrittura o la saldatura. Il prezzo da pagare per questo tipo di pianificazione è il tempo di movimento, che, in condizioni non ottimizzate, può essere elevato.
Presso i laboratori di Robotica e Automazione del Dipartimento di Ingegneria Industriale (DII) dell'Università degli Studi di Padova è stato utilizzato un *end-effector* a sbalzo montato su un robot antropomorfo a sei assi a polso sferico, al fine di sfruttare la ridondanza del manipolatore - rispetto al *task* di scrittura su un piano orientato a piacere - come opportunità di ottimizzazione del movimento.
In questo lavoro di tesi sono stati sviluppati diversi algoritmi di ottimizzazione con l'obiettivo di minimizzare il tempo di movimento lungo una traiettoria cartesiana precedentemente definita.

Introduzione

La tesi qui presentata si inserisce nel contesto del progetto "Chocobot - Decorazione personalizzata di torte celebrative ad alta efficienza energetica e prototipazione rapida di grandi strutture di cioccolato", sviluppato dal gruppo di Robotica dell'Università di Padova in collaborazione con Epson Italia.

L'aspetto specifico affrontato in questa tesi è la massimizzazione delle prestazioni del robot in termini di tempo durante le operazioni di scrittura, ossia la minimizzazione del tempo ciclo e l'aumento della produttività. Ciò può essere ottenuto sfruttando la ridondanza cinematica del robot antropomorfo nell'operazione di decorazione e scrittura. La ridondanza può essere ottenuta sia con l'impiego di un end-effector a sbalzo che regge il dosatore del fluido da depositare, sia servendosi di un equipaggiamento esterno come un nastro trasportatore o una tavola rotante. Il lavoro svolto consiste nell'implementazione di un modello di ottimizzazione, in ambiente MatLAB, per il movimento dei giunti robot, grazie al quale si possa valutare l'orientazione ottimale della flangia durante il movimento e l'effetto sulle velocità dei motori del robot. I metodi di ottimizzazione, già parzialmente sviluppati da alcuni tesisti precedenti ed applicati ad un robot Scara, qui sono stati estesi ad un robot di tipo antropomorfo.

Gli obiettivi del lavoro di tesi svolto sono stati i seguenti:

- Impiegare la classe KINpro per la gestione del movimento e la visualizzazione del robot Adept Viper s650.

- Implementare diversi criteri di ottimizzazione dei movimenti cartesiani da eseguire col robot antropomorfo.
- Eseguire simulazioni su geometrie di varia complessità in ambiente MatLAB, verificando i vantaggi derivanti dai diversi criteri di ottimizzazione.
- Creare una connessione TCP/IP tra MatLAB ed ACE V+ per la movimentazione del robot reale.
- Confrontare i risultati sperimentali in termini di qualità del movimento e di tempo effettivo di esecuzione rispetto alle stime ottenute con le simulazioni.

Il capitolo 1 è introduttivo al lavoro svolto e vuole dare una panoramica sul mondo della robotica industriale. Inizialmente sono riassunte le diverse tipologie di manipolatori industriali, mentre in un successivo paragrafo viene descritto brevemente lo stato attuale dell'automazione nel mondo. Nel capitolo 2 viene presentato il robot Adept Viper s650, utilizzato nelle prove simulative e sperimentali di questa tesi. Inoltre, sono esposte le nozioni teoriche che stanno alla base della robotica industriale. Poiché il progetto Chocobot è già stato in parte approfondito nella tesi di [1], vengono solamente riportate alcune informazioni essenziali sul suo stato dell'arte, senza scendere in dettaglio sui codici già sviluppati negli ultimi anni. Nel capitolo 3 è illustrato il processo di definizione su MatLAB della traiettoria cartesiana che il robot dovrà eseguire, mentre nel capitolo 4 vengono spiegati i codici utilizzati per la simulazione cinematica e gli algoritmi di ottimizzazione. Il capitolo 5, infine, descrive la coomunicazione tra MatLAB ed ACE V+, il programma di Omron addetto al controllo del robot.

Capitolo 1

Robotica industriale

1.1 Cenni storici

Il termine robot deriva da un pezzo teatrale di Carel Čapek (1090-1938) intitolato *R.U.R. Rossumovi univerzální roboti (I robot universali di Roussom)*. In questo spettacolo, scritto nel 1920, piccoli esseri meccanici e antropomorfi, denominati *robota*, eseguivano alla perfezione gli ordini del loro padrone. Il nome *robota* in lingua ceca significa *lavoro pesante*. Il termine *robotica* venne coniato successivamente dallo scrittore fantascientifico I. Asimov (1920-1992).

Oggigiorno la robotica è ampiamente diffusa nei vari settori industriali, ma anche in applicazioni di domotica e medicali, militari e di ricerca spaziale, ad esempio. In ambito industriale, tipici impieghi di normali robot sono operazioni del tipo: *manipolazione, montaggio, pallettizzazione, verniciatura, saldatura, taglio, lavorazione o misura di oggetti singoli o a gruppi*. Un vantaggio è sicuramente quello di scaricare gli operatori umani da compiti troppo pesanti, ripetitivi o nocivi per la salute.
L'integrazione delle cosiddette "intelligenze artificiali" e dei sistemi di visione ai robot sta via via allargando le operazioni da affidare agli stessi, con crescenti precisione ed affidabilità. Inoltre, la comparsa e diffusione dei robot collaborativi sta consentendo in questi ultimi anni la presenza dei robot a fianco degli operatori,

in una vera e propria collaborazione, che migliori produttività ed ergonomia del luogo di lavoro.

Sebbene i primi *automi* siano comparsi già ai tempi degli antichi Greci e poi ripresi nel Medioevo e Rinascimento, i robot hanno conosciuto una grande diffusione solo quando la tecnologia ha potuto realizzare dei calcolatori che li comandassero. Questo ha permesso di poter fare svolgere ai robot operazioni anche molto diverse tra di loro, semplicemente riprogrammandoli.
La robotica industriale, come viene ora intesa, muove i suoi primi passi nell'immediato dopoguerra. Nel 1946 l'americano G. C. Devol sviluppa un dispositivo che permette di registrare magneticamente dei segnali elettrici che possono essere successivamente inviati a un dispositivo elettromeccanico per comandarlo opportunamente. Negli anni '50 del Novecento vengono sviluppati e brevettati i primi telemanipolatori. Nel 1950 due viene realizzato al MIT (Massachusetts Institute of Technology) un prototipo di macchina a controllo numerico computerizzato. Nel 1960, in base al brevetto di Devol, la Unimation (UNIversal AutoMATION) produce il primo robot denominato UNIMATE: esso viene installato con successo nel 1961 in una linea di produzione della General Motor con funzioni di asservimento a una macchina per la pressocolata.
Negli anni settanta vengono installati diversi tipi di robot commerciali ad azionamento elettrico o idraulico. Ma la vera rivoluzione avviene nel 1979, quando viene sviluppato presso l'Università di Yamanashi in Giappone il primo robot SCARA (Selective Compliance Assembly Robot Arm). Questa tipologia diventerà molto comune in applicazioni di montaggio sul piano orizzontale, come nel caso delle schede elettroniche. Tutti i robot industriali fin qui citati sono a *cinematica seriale*.
Si elencano solo alcune altre date significative: nel 1975 il robot cartesiano SIGMA, nel 1978 il primo antropomorfo a 6 assi Puma, nel 1992 il primo robot *parallelo* (catena cinematica chiusa) chiamato Delta. Quest'ultima famiglia di robot è ad oggi primariamente utilizzata in applicazioni di "presa e rilascio" (*pick*

and place) e confezionamento (*packaging*).
L'ultima "conquista" della robotica è stata l'uscita del primo robot collaborativo, l'UR5, nel 2008.

In ogni caso, la nascita dei moderni robot industriali deriva dalla confluenza di alcune caratteristiche dei telemanipolatori (struttura cinematica), dalle macchine a controllo numerico computerizzato (CNC) (riprogrammabilità), nonché dagli elaboratori elettronici che li hanno facilmente resi programmabili. È inoltre evidente che tutti gli sviluppi della robotica sono legati a quelli di parecchie scienze e principalmente della meccanica, dell'informatica, dell'elettronica, della sensoristica e, in prospettiva, dell'intelligenza artificiale [2].

1.2 Principali manipolatori industriali

È opportuno riportare alcune definizioni tratte dalle normative vigenti, per comprendere cosa sia un robot industriale.
Ecco la definizione di *macchina* secondo la "Direttiva Macchine" (art. 2 della 2006/42/CE):

> *un insieme equipaggiato o destinato ad essere equipaggiato di un sistema di azionamento diverso dalla forza umana o animale diretta, composto di parti o di componenti, di cui almeno uno mobile, collegati tra loro solidamente per un'applicazione ben determinata.*

La definizione di *industrial robot*, invece, viene data dalla norma UNI EN ISO 10218-1: 2012, come segue:

> *automatically controlled reprogrammable multipurpose manipulator programmable in three or more axes which can be either fixed in place or mobile for use in industrial automation applications.*

Le principali caratteristiche funzionali di un robot sono:

- versatilità;
- possibilità di adattamento a situazioni differenti;
- autonomia e cioè una qualche capacità di prendere decisioni.

Per garantire queste caratteristiche, il robot industriale non può lavorare da solo, bensì deve integrarsi in un complesso sistema di feeder, sensori, sistemi di visione, di movimentazione e di controllo, all'interno della propria cella robotizzata.

1.2.1 Architettura dei manipolatori

Come già anticipato nella sezione 1.1, esistono diverse tipologie di manipolatori industriali. Esse variano a seconda del tipo di catena cinematica spaziale, ossia della sequenza di membri rigidi e di coppie cinematiche: se questa è aperta, parleremo di *robot seriali*; se, invece, è chiusa, parleremo di *robot paralleli*. Altra caratteristica fondamentali sono gli accoppiamenti o coppie cinematiche presenti nel manipolatore, che possono essere riassunte in figura 1.1. Il numero di gradi di libertà del manipolatore, infatti, dipende dal numero di membri di cui si compone, dal numero di gradi di libertà di ciascun membro e dal numero e dalla tipologia di vincoli realizzati tramite gli accoppiamenti cinematici.

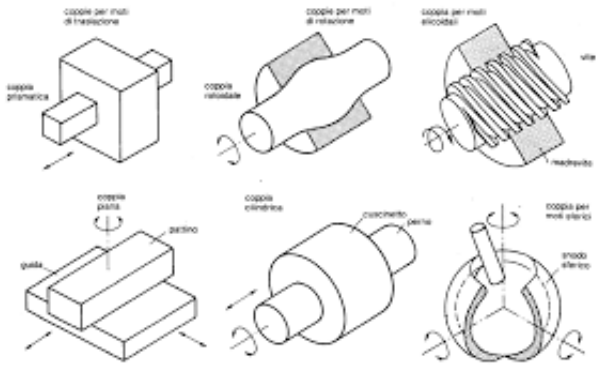


Figura 1.1: Principali accoppiamenti cinematici presenti nei meccanismi spaziali

Esistono differenti tipologie di robot e differenti criteri di classificazione Quello di gran lunga più diffuso si basa sull'identificazione della successione dei tipi di giunto - prismatici (P) o rotazionali (R) - che articolano il braccio del manipolatore.
Si possono raggruppare in cinque categorie fondamentali:

- configurazione cartesiana (3P) [Fig. 1.2]
- configurazione cilindrica (1R, 2P) [Fig. 1.3]
- configurazione sferica o polare (2R, 1P) [Fig. 1.4]
- configurazione SCARA (2R, 1P) [Fig. 1.5]
- configurazione articolata o antropomorfa (3P) [Fig. 1.6]

Figura 1.2: Esempio di robot cartesiano a 3 assi. Fonte: www.robotcartesiani.com

1.3 Mercato dei robot

I robot industriali iniziano a diffondersi in maniera significativa soltanto verso la metà degli anni settanta, quando il loro costo orario, a parità di prestazioni, risulta concorrenziale a quello della manodopera umana. Un altro fattore determinante è stato la sostituzione dell'uomo in operazioni pericolose, gravose o eccessivamente ripetitive.
Il punto di trade-off viene raggiunto grazie a due fattori concomitanti: da una parte il costo dell'elettronica di controllo del robot diminuisce drasticamente (negli anni sessanta, esso incideva sul costo totale del robot per il 75%, mentre solo il 20% nel 2000) riducendone il costo orario, dall'altra le rivendicazioni sindacali fanno significativamente aumentare gli stipendi della manodopera specializzata.
A titolo esemplificativo, il grafico di Figura 1.7 riporta la diminuzione del costo

Figura 1.3: Esempio di robot cilindrico. Fonte: www.superdroidrobots.com

Figura 1.4: Esempio di robot sferico. Fonte: Fanuc

relativo medio dei robot e l'aumento del costo del lavoro a partire dal 1990 al 2014.

Si riportano ora, per completezza, alcune informazioni in merito al mercato dei robot industriali tratte dall'"Executive Summary World Robotics 2019 - In-

Figura 1.5: Esempio di robot SCARA. Fonte: Epson

Figura 1.6: Esempio di robot antropomorfo a 6 assi. Fonte: Mitsubishi

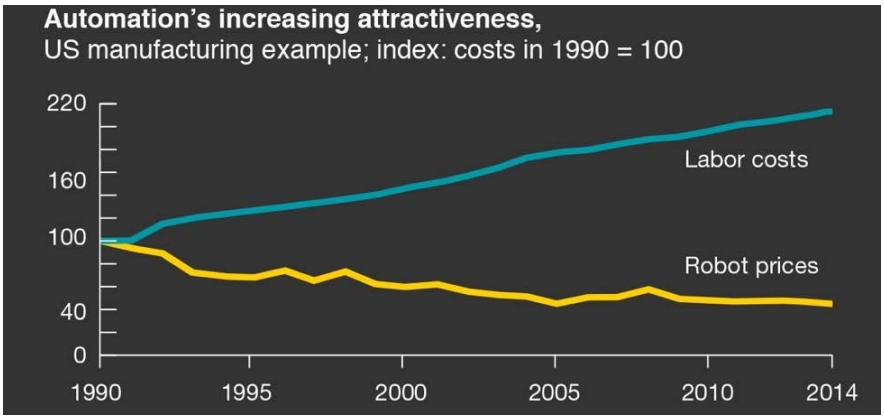


Figura 1.7: Costo del lavoro rispetto al costo dei robot dal 1990 al 2014. Fonte: McKinsey

dustrial Robots" stilato dall'IFR (*International Federation of Robotics*) [3].
Nel 2018, le installazioni globali di robot sono aumentate del 6% a 422.271 unità, per un valore 16,5 miliardi di dollari (senza software e periferiche). Lo stock operativo di robot era calcolato a 2.439.543 unità (+ 15%). Questo risultato è stato una sorpresa perché le principali industrie dei clienti, automobilistica ed elettrica/elettronica, hanno avuto un anno difficile e le due principali destinazioni, Cina e Nord America, sono state protagoniste di un conflitto commerciale, diffondendo incertezza nell'economia globale. Tuttavia, l'industria automobilistica rimane il più grande settore cliente con il 30% delle installazioni totali, davanti a elettrico / elettronico (25%), metallo e macchinari (10%), plastica e prodotti chimici (5%) e cibo e bevande (3%).
Si noti che per il 19% dei robot non esistono informazioni sul settore del cliente. Questa cifra è superiore di cinque punti percentuali all'anno precedente.
Dal 2010, la domanda di robot industriali è aumentata notevolmente a causa della tendenza in atto verso l'automazione e le continue innovazioni tecniche nei robot industriali. Dal 2013 al 2018, le installazioni annuali sono aumentate del 19% in media all'anno (CAGR). Fra il 2005 e il 2008, il numero medio annuo di robot venduti è stato di circa 115.000 unità, prima che la crisi economica e finanziaria globale facesse diminuire le installazioni di robot 60.000 unità nel 2009 con molti investimenti rinviati. Nel 2010, investimenti ha lasciato spazio e ha portato le installazioni di robot a 120.000 unità. Fino al 2015, annuale le installazioni erano più che raddoppiate fino a quasi 254.000 unità. Nel 2016 il marchio di Sono state

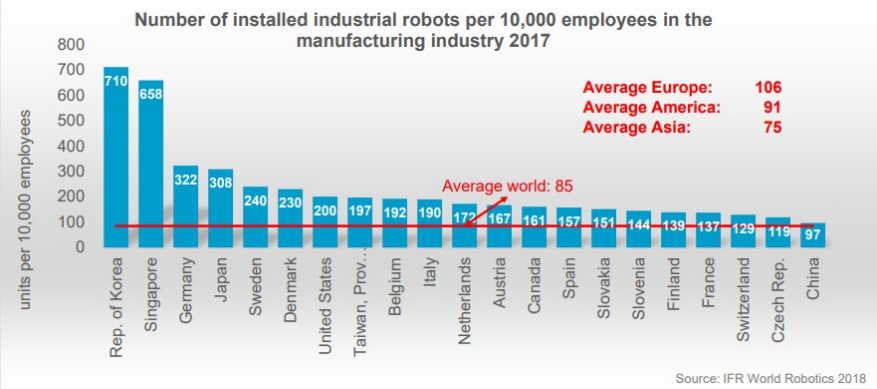


Figura 1.8: Numero di robot industriali installati ogni 10,000 operai nel 2017. Fonte: IFR

superate 300.000 installazioni all'anno e nel 2017 le installazioni sono aumentate a quasi 400.000 unità.

L'Asia è il più grande mercato mondiale di robot industriali, anche se la crescita è rallentata sostanzialmente nel 2018. Un totale di 283.080 unità sono state installate nel 2018 solo l'1% in più rispetto all'anno prima, ma ancora un nuovo picco per il sesto anno consecutivo. Due robot su tre (67%) recentemente schierati nel 2018 sono stati installati in Asia. Dal 2013 al 2018, robot annuale le installazioni sono aumentate in media del 23% all'anno. Il 2018 rivela un quadro differenziato per i tre maggiori mercati asiatici: le installazioni in Cina (154.032 unità; -1%) e nella Repubblica di Corea (37.807 unità; -5%) sono diminuite, mentre le installazioni in Giappone (55.240 unità; $+21\%$) sono notevolmente aumentate. L'Europa è aumentata del 14% a 75.560 unità. Il il tasso di crescita medio annuo dal 2013 al 2018 è del 12%. Il tasso di crescita è stato ancora più alto nelle Americhe: nel 2018 sono stati installati circa 55.212 robot, il 20% in più rispetto all'anno prima e - come in Asia e in Europa - rappresenta un nuovo picco per il sesto anno di fila. Il tasso di crescita medio annuo dal 2013 è del 13%.

Esistono cinque mercati principali per i robot industriali: Cina, Giappone, Stati Uniti, Repubblica di Corea e Germania. Questi paesi rappresentano il 74% delle installazioni globali di robot.

La Cina è il più grande mercato mondiale di robot industriali dal 2013 e ne rappresenta 36% delle installazioni totali nel 2017 e 2018. Qui, nel 2018 sono state installate 154.032 unità.

Nel 2018, le installazioni di robot in Giappone sono aumentate del 21% a 55.240

unità. Il il tasso di crescita medio annuo del 17% dal 2013 è notevole per un paese che già ha un alto livello di automazione nella produzione industriale.
Per l'ottavo anno consecutivo, le installazioni di robot negli Stati Uniti hanno raggiunto un nuovo picco (40.373 unità; +22%). Per quanto riguarda il numero annuo di installazioni, gli Stati Uniti hanno preso la terza posizione dalla Repubblica di Corea nel 2018.
Nella Repubblica di Corea, le installazioni annuali di robot sono diminuite da quando hanno raggiunto un livello massimo di 41.373 unità nel 2016. Nel 2018 sono state installate 37.807 unità (−5%). Le installazioni sono aumentate in media del 12% per anno dal 2013.
La Germania è il quinto mercato di robot al mondo. Nel 2018, il numero di robot installati è aumentato del 26% raggiungendo un nuovo picco di 26.723 unità. Le installazioni in questo paese sono guidate principalmente dall'industria automobilistica.

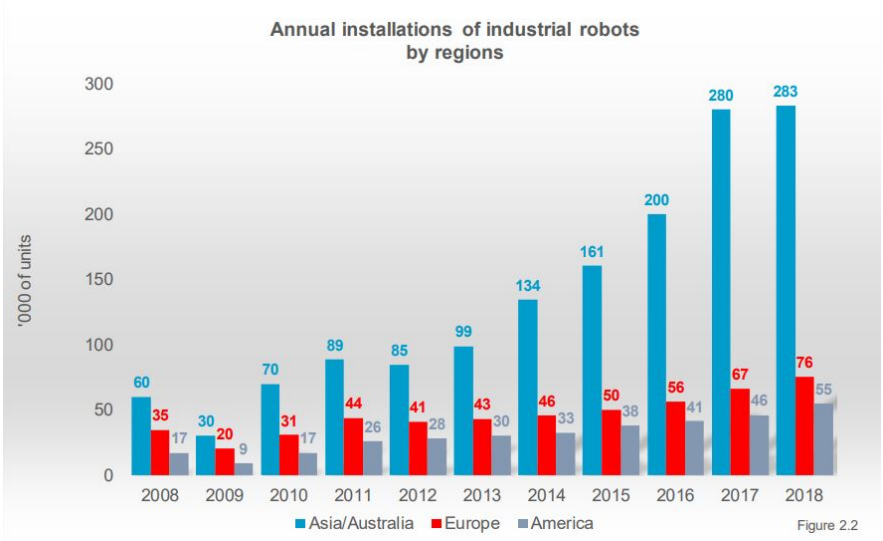


Figura 1.9: Installazioni annuali di robot industriali dal 2008 al 2018. Fonte: IFR

L'industria automobilistica è il cliente più importante dei robot industriali. Quasi il 30% di tutte le installazioni di robot industriali avviene in questo settore. Dopo un 2017 molto forte che ha visto un aumento del 21% delle installazioni a 123.439 unità, questo livello è stato mantenuto nel 2018.
Dal 2013 al 2018, le installazioni annuali nell'industria automobilistica sono aumentate mediamente del 13% ogni anno (CAGR). Dopo la crisi economica del

2008/2009, le case automobilistiche hanno iniziato a ristrutturare le loro attività. Dal 2010, gli investimenti in nuove capacità produttive nei mercati emergenti e gli investimenti nella modernizzazione della produzione nei principali paesi produttori di automobili hanno guidato la domanda di robot. L'utilizzo di nuovi materiali, lo sviluppo di sistemi di trasmissione ad alta efficienza energetica e l'elevata concorrenza in tutti i principali mercati automobilistici hanno spinto la domanda di investimenti nonostante le capacità produttive eccedenti. I fornitori di componenti per autoveicoli sono stati pesantemente colpiti dalla ristrutturazione dell'industria automobilistica dopo la crisi economica del 2009. Hanno dovuto seguire l'esempio dopo che i fornitori di autoveicoli hanno iniziato a realizzare i propri piani di investimento. Pertanto, la fornitura di robot ai fornitori di componenti per autoveicoli ha acquisito slancio solo nel 2011.
Le installazioni di robot nell'industria elettrica/elettronica (inclusi computer e apparecchiature, radio, TV e dispositivi di comunicazione, apparecchiature mediche, strumenti di precisione e ottici) sono aumentate in media del 24% ogni anno dal 2013. Nel 2017, rappresentavano il 31% delle installazioni totali e stavano per sorpassare l'industria automobilistica. Tuttavia, nel 2018, la domanda globale di dispositivi e componenti elettronici è notevolmente diminuita. Questo settore cliente è probabilmente quello più colpito dalla crisi commerciale Cina-USA in quanto i paesi asiatici sono leader nella produzione di prodotti e componenti elettronici. Le installazioni di robot in questo settore sono diminuite del 14% dal loro livello massimo di 121.955 unità nel 2017 a 105.153 unità nel 2018.

Nel 2018, la densità media dei robot nell'industria manifatturiera era di 99 robot ogni 10.000 dipendenti. Si noti che questa media globale include solo quei paesi che hanno uno stock operativo rilevante. È quindi sovrastimato poiché i paesi con una bassa densità di robot vengono sistematicamente esclusi. Stesso discorso per i seguenti dati: l'Europa è la regione con la più alta densità di robot, vantando un valore medio di 114 unità. Nelle Americhe il valore è di 99 unità e in Asia/Australia è di 91 unità. Spinta dall'elevato volume di installazioni di robot negli ultimi anni, l'Asia ha il più alto tasso di crescita della densità di robot e sta per raggiungere le Americhe. Tra il 2013 e il 2018, il tasso di crescita medio annuo della densità dei robot in Asia è stato del 16%, nelle Americhe del 9% e in Europa del 6% [3].

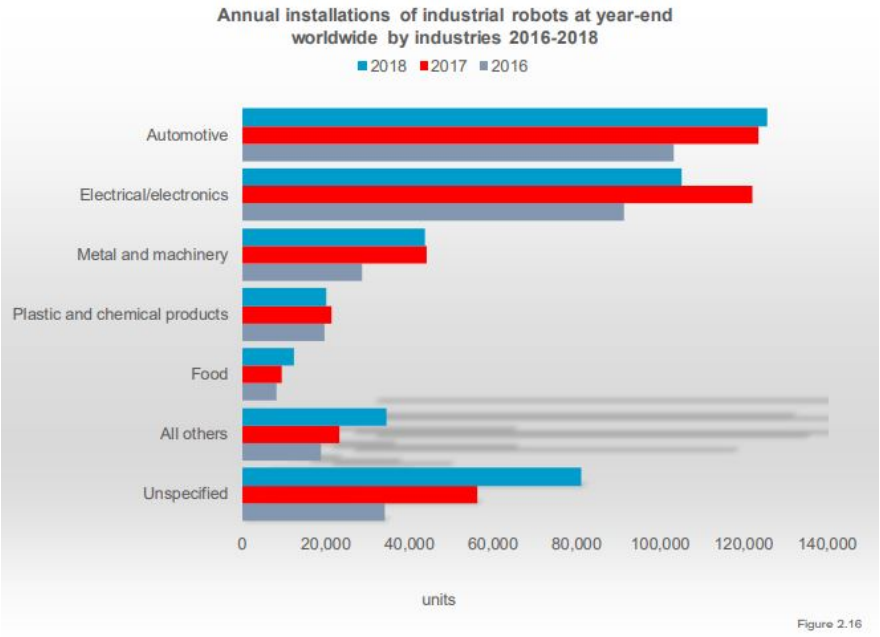


Figura 1.10: Installazioni annuali di robot industriali nel periodo 2016-2018 per i diversi settori industriali nel panorama mondiale. Fonte: IFR

1.4 Progetto Chocobot

Il progetto "Chocobot" è stato uno dei sei vincitori del primo concorso di Epson Europe "Win-A-Robot", creato per individuare e promuovere nuovi utilizzi nel campo della robotica e delle tecnologie di automazione.
L'iniziativa si poneva come obiettivo lo sviluppo di metodologie e algoritmi innovativi e di sicuro interesse industriale non solo per il tema specifico del progetto, ovvero la decorazione di torte, ma soprattutto per altri ambiti industriali dove questo tipo di soluzioni possano essere traslate e impiegate.

Capitolo 2

Robot e modello cinematico

2.1 Il robot Adept Viper s650

L'Adept Viper s650 appartiene ad una famiglia di robot antropomorfi a 6 assi ad alte prestazioni della compagnia Omron Adept Technology. Si tratta di un manipolatore a polso sferico, ossia in cui gli assi 4, 5 e 6 sono concorrenti in uno stesso punto: come vedremo nel paragrafo 2.2.3, questa caratteristica costruttiva porta notevoli semplificazioni nei calcoli della cinematica inversa. La sigla "s650" indica lo sbraccio massimo del robot, che per questo modello è proprio pari a 650 mm.
Il Viper è stato immesso nel mercato nel 2006 ed è particolarmente impiegato per assemblaggi. La velocità e precisione dei robot Adept Viper li rende particolarmente indicati anche per lavori di manipolazione e packaging e anche per qualunque lavoro dove si necessita velocità e precisione.
Nelle Figure 2.1 e 2.2 sono riportati rispettivamente un'immagine del robot e i suoi disegni costruttivi. Nel data-sheet di Figura 2.3 sono riportate tutte le informazioni caratteristiche sul robot, come rotazioni massime, velocità di giunto, carico massimo (*payload*) [4].

2.2 Analisi cinematica di posizione

La posizione relativa tra due corpi può essere espressa mediante la matrice di posizione T_{ij}: questa è la matrice di trasformazione delle coordinate tra due terne

Figura 2.1: Adept Viper s650. Fonte: Omron/Adept

solidali ai corpi stessi:

$$T_{ij} = \left[\begin{array}{ccc|c} X_x & Y_x & Z_x & X_P \\ X_y & Y_y & Z_y & Y_P \\ X_z & Y_z & Z_z & Z_P \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R_{ij} & & O_{ij} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.1}$$

Come è noto, la sottomatrice R_{ij} descrive l'orientamento della terna (j) rispetto alla terna (i), mentre O_{ij} è il vettore della posizione di j rispetto a i. In generale, R potrà essere espressa in funzione di 3 coordinate angolari (angoli di Eulero, per esempio), mentre O sarà generalmente funzione dei 3 coordinate lineari (coordinate cartesiane x, y, z):

$$R = f(\alpha, \beta, \gamma)$$

$$O = f(x, y, z)$$

In totale si avranno 6 parametri che descrivono la posizione relativa tra le due terne. Ora, schematizzando un robot con una catena aperta di membri rigidi

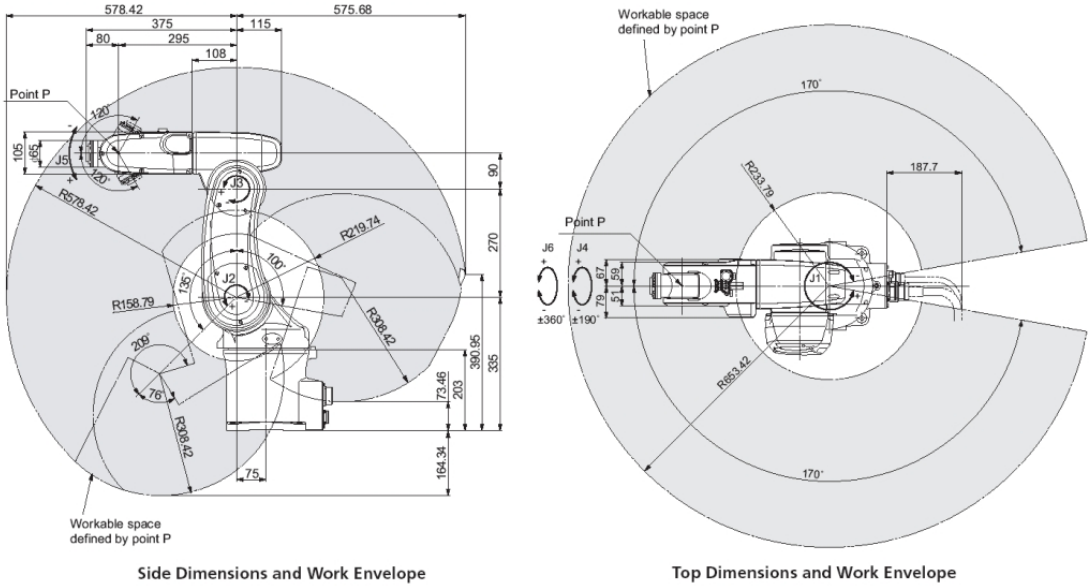


Figura 2.2: Schemi costruttivi dell'Adept Viper s650 e quote principali

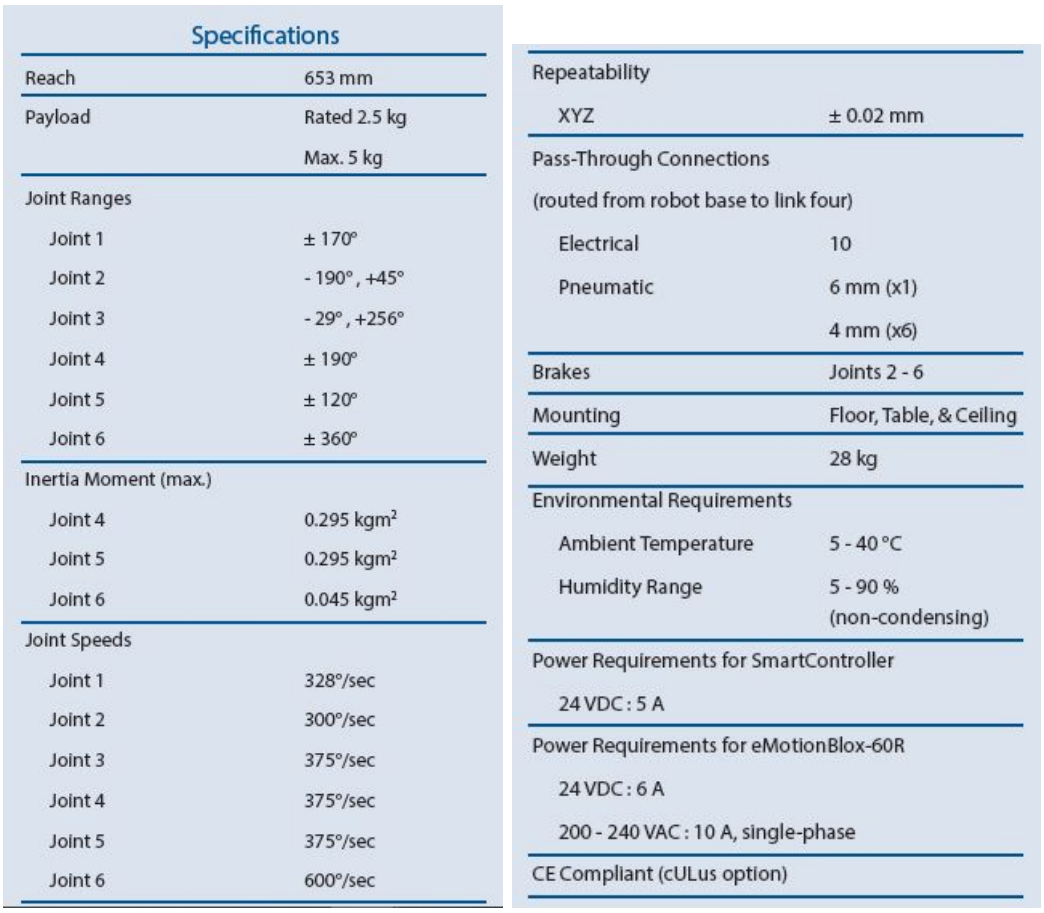
numerati da 0 a n, si può pensare di descrivere la posizione relativa tra 2 qualsiasi membri i e j del robot per mezzo di una matrice T_{ij}. Per fare questo è necessario fissare su ogni membro del robot (base e pinza compresi) una terna cartesiana. Un criterio di posizionamento "intelligente" delle terne è spiegato nel paragrafo 2.2.1.
Le proprietà delle matrici di trasformazione assicurano che, note la matrice di posizione tra due terne i e j e quella tra le terne j e k, è immediato calcolare la matrice di trasformazione tra le terne i e k con una semplice moltiplicazione

$$T_{ik} = T_{ij}T_{jk}$$

È talvolta necessario calcolare la derivata della matrice di posizione di un membro del robot rispetto a una coordinata ai giunti. Si definisce come segue:

$$\frac{\partial T_{0k}}{\partial q_i} = \begin{cases} L_{i-1,i(0)}T_{0k} & \text{se } i \leq k \\ 0 & \text{se } i > k \end{cases} \tag{2.2}$$

ove la matrice $L_{i-1,i(0)}$ è la matrice che descrive nel sistema di riferimento (0) l'asse elicoidale tra i membri i ed $i-1$. La relazione appena scritta evidenzia il fatto che T_{0k} dipende dalla coordinata q_i solo se questa è compresa tra la base (0) e il giunto (k). In altre parole, la posizione del membro k dipende solo dalle



Specifications

Reach	653 mm
Payload	Rated 2.5 kg
	Max. 5 kg
Joint Ranges	
Joint 1	± 170°
Joint 2	- 190° , +45°
Joint 3	- 29° , +256°
Joint 4	± 190°
Joint 5	± 120°
Joint 6	± 360°
Inertia Moment (max.)	
Joint 4	0.295 kgm²
Joint 5	0.295 kgm²
Joint 6	0.045 kgm²
Joint Speeds	
Joint 1	328°/sec
Joint 2	300°/sec
Joint 3	375°/sec
Joint 4	375°/sec
Joint 5	375°/sec
Joint 6	600°/sec

Repeatability	
XYZ	± 0.02 mm
Pass-Through Connections	
(routed from robot base to link four)	
Electrical	10
Pneumatic	6 mm (x1)
	4 mm (x6)
Brakes	Joints 2 - 6
Mounting	Floor, Table, & Ceiling
Weight	28 kg
Environmental Requirements	
Ambient Temperature	5 - 40 °C
Humidity Range	5 - 90 % (non-condensing)
Power Requirements for SmartController	
24 VDC : 5 A	
Power Requirements for eMotionBlox-60R	
24 VDC : 6 A	
200 - 240 VAC : 10 A, single-phase	
CE Compliant (cULus option)	

Figura 2.3: Datasheet dell'Adept Viper s650

coordinate ai giunti da i a k.
La matrice L, in generale, è così definita:

$$L = \left[\begin{array}{ccc|c} & \underline{\mathrm{u}} & & t \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{2.3}$$

$$dove \;\; t = -\underline{u}P^* + up$$

dove $\underline{\mathrm{u}}$ rappresenta il versore dell'asse, mentre t rappresenta tanto la posizione dell'asse (P^*) quanto il passo dell'elica (p) [2].

2.2.1 Notazione di Denavit-Hartenberg

Noto il modello di robot da utilizzare, vogliamo risolvere il problema cinematico diretto di posizione per un meccanismo spaziale in catena aperta, ovvero vogliamo calcolare la posizione di ogni suo punto a partire dai valori delle coordinate libere.
L'idea è quella di associare ad ogni membro un sistema locale di coordinate (solidale al membro) e calcoliamo le matrici di trasformazione tra tali sistemi e il sistema alla base robot.
È conveniente definire innanzitutto le trasformazioni relative tra le terne solidali a membri adiacenti ($T_{i,i-1}$). In generale, ciascuna trasformazione relativa è funzione di 6 parametri: una coordinata libera (quella associata al moto relativo tra i membri i ed $i-1$) e cinque costanti. Le trasformazioni rispetto alla terna assoluta (T_{i0}) si ottengono dal prodotto di più trasformazioni relative:

$$T_{i0} = T_{10}T_{21}\dots T_{i,i-1}$$

$$T_{n0} = T_{10}(q_1)T_{21}(q_2)\dots T_{n,n-1}(q_n)$$

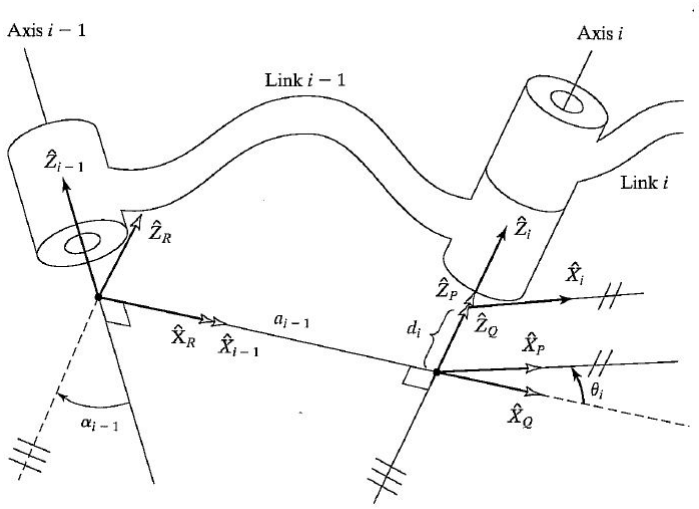


Figura 2.4: Posizionamento delle terne tra due generici assi $i-1$ e i secondo la notazione DH. Fonte: Craig

La notazione di *Denavit-Hartenberg* è un metodo per sistematizzare e semplificare la scelta delle terne di riferimento ed il calcolo dei parametri che definiscono le matrici di trasformazione relative. Questi parametri risulteranno 4 per ciascuna

trasformazione: due rotazioni e due traslazioni, di cui una coordinata libera e tre costanti [5].
La procedura si riassume così:

- si individuano e si numerano gli assi di movimento;
- si individuano le normali comuni alle coppie di assi successivi;
- si disegnano le terne di riferimento (prima l'origine, poi l'asse z e l'asse x, poi l'asse y secondo la regola della mano destra);
- si compila una tabella (la *tabella di DH*) contenente in ogni riga i 4 parametri di una trasformazione relativa; le righe saranno in numero pari al numero di gradi di libertà del manipolatore; ciascuna riga conterrà una e una sola coordinata libera (un angolo se il giunto è rotazionale, una distanza se è traslazionale).

Seguendo la procedura indicata sopra, si dovrebbe arrivare a posizionare le terne come illustrato in Figura 2.4. In particolare, si riconoscono i 4 parametri di DH come segue:

- **angolo** α_{i-1}: è l'angolo intorno all'asse x_{i-1} che porta l'asse z_{i-1} lungo la direzione dell'asse z_i (positivo secondo la regola della mano destra);
- **distanza** a_{i-1}: è la distanza di O_i da O_{i-1} misurata lungo l'asse x_{i-1} (positiva secondo il verso dell'asse x_{i-1});
- **angolo** ϑ_i: è l'angolo intorno all'asse z_i che porta l'asse x_{i-1} lungo la direzione dell'asse x_i (positivo secondo la regola della mano destra);
- **distanza** d_i: è la distanza di O_i da O_{i-1} misurata lungo l'asse z_{i-1} (positiva secondo il verso dell'asse z_{i-1}).

La generica matrice di trasformazione relativa $T_{i,i-1}$ ottenuta con l'approccio di Denavit-Hartenberg, dunque, si calcola con la seguente equazione:

$$
\begin{aligned}
T_{i,i-1} &= T_{Rx}(\alpha_{i-1})T_{Tx}(a_{i-1})T_{Rz}(\vartheta_i)T_{Tz}(d_i) \\
&= \begin{bmatrix}
\cos\vartheta_i & -\sin\vartheta_i & 0 & a_{i-1} \\
\sin\vartheta_i\cos\alpha_{i-1} & \cos\vartheta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\
\sin\vartheta_i\sin\alpha_{i-1} & \cos\vartheta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\
0 & 0 & 0 & 1
\end{bmatrix}
\end{aligned}
$$

dove i pedici R e T indicano una rotazione intorno all'asse specificato e una traslazione lungo la direzione indicata, rispettivamente. I parametri tra parentesi rappresentano l'entità delle trasformazioni stesse.
In Tabella 2.1 sono riportati i valori dei parametri di Denavit-Hartenberg necessari alla definizione delle terne di riferimento nel caso particolare del Viper s650 (vedi Figura 2.5), dove $a_1 = 75$ mm, $a_2 = 270$ mm, $a_3 = 90$ mm, $d_1 = 335$ mm, $d_4 = 295$ mm, $d_6 = 80$ mm.

T_{ij}	α_{i-1}	a_{i-1}	θ_i	d_i
10	0	0	θ_1	d_1
21	-90	a_1	-90+θ_2	0
32	0	a_2	θ_3	0
43	-90	a_3	θ_4	d_4
54	90	0	θ_5	0
65	-90	0	θ_6	d_6

Tabella 2.1: Tabella DH dell'Adept Viper s650

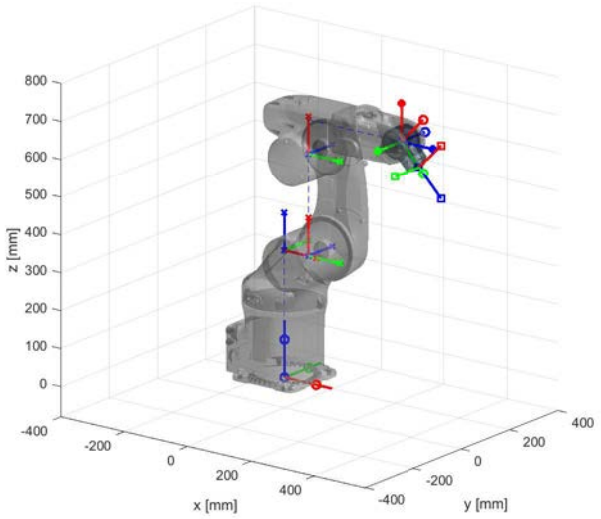


Figura 2.5: Applicazione della notazione di DH al Viper s650

$$T_{10} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{21} = \begin{bmatrix} c_2 & -s_2 & 0 & a_1 \\ 0 & 0 & 1 & 0 \\ -s_2 & -c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{32} = \begin{bmatrix} c_3 & -s_3 & 0 & a_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{43} = \begin{bmatrix} c_4 & -s_4 & 0 & a_3 \\ 0 & 0 & 1 & d_4 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{54} = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{65} = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{t6} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{20} = T_{10}T_{21}$$

$$T_{30} = T_{10}T_{21}T_{32}$$

$$T_{40} = T_{10}T_{21}T_{32}T_{43}$$

$$T_{50} = T_{10}T_{21}T_{32}T_{43}T_{54}$$

$$T_{60} = T_{10}T_{21}T_{32}T_{43}T_{54}T_{65}$$

$$T_{t0} = T_{10}T_{21}T_{32}T_{43}T_{54}T_{65}T_{t6}$$

2.2.2 Cinematica diretta

La descrizione della configurazione di un manipolatore richiede la definizione di una terna solidale alla base (terna 0) e di una terna solidale alla flangia terminale (terna n), dove n è il numero di gradi di mobilità del manipolatore.
Considerato che un manipolatore, per essere impiegato, deve essere inserito in un ambiente e dotato di un organo terminale che consenta la presa o lavorazione

di oggetti, definiamo la terna ambiente (*world*) e la terna utensile (*tool*), e le trasformazioni relative $T_{t,n}$ e $T_{0,w}$.
La posizione della terna *tool* dipende dalla conformazione dell'utensile e dalla sua funzionalità e viene sempre definita rispetto a quella della terna n.

Il problema cinematico diretto di posizione di un manipolatore in catena aperta viene formulato, in forma matriciale, come segue:

$$T_{t,w} = T_{0,w} T_{n,0}(\mathbf{q}) T_{t,n} \tag{2.4}$$

Note le coordinate di giunto $\mathbf{q}$, si ricava la posizione incognita $T_{t,w}$ dell'utensile rispetto alla terna *world*. Le matrici $T_{t,n}$ e $T_{t,n}$ sono note e costanti, mentre la matrice $T_{t,n}$ è funzione delle coordinate di giunto e si ricava in forma esplicita applicando il metodo di Denavit-Hartenberg.
Di conseguenza, la soluzione del problema cinematico diretto di posizione di un meccanismo spaziale in catena aperta esiste ed è unica [5].

Eseguendo l'analisi diretta di posizione al variare di tutti i valori possibili delle variabili di giunto, si ottengono:

- lo *spazio raggiungibile* del manipolatore, ovvero il luogo dei punti raggiungibili dall'utensile (senza specifica di orientamento);
- lo *spazio di lavoro* del manipolatore, ovvero il luogo dei punti raggiungibili con qualunque orientazione dell'utensile.

Lo spazio di lavoro è un sottoinsieme dello spazio raggiungibile. Entrambi sono limitati dal range ammissibile per le variabili di giunto, ovvero dai range di rotazione/traslazione degli assi del manipolatore.

2.2.3 Cinematica inversa

Il problema cinematico inverso di posizione di un manipolatore in catena aperta viene formulato, in forma matriciale, come segue:

$$T_{n,0}(\mathbf{q}) = T_{0,w}^{-1} T_{t,w} T_{t,n}^{-1} = A \tag{2.5}$$

dove la matrice A è nota e dipende dalla posizione dell'utensile $T_{t,w}$ (assegnata), dalla posizione del robot $T_{0,w}$ rispetto alla terna ambiente (costante) e dalla geometria dell'utensile definita dalla trasformazione $T_{t,n}$ (costante).

Le incognite sono le variabili di giunto contenute nella matrice $T_{n,0}$.

$$\left[\begin{array}{ccc|c} & R(q) & & \begin{matrix} x(q) \\ y(q) \\ z(q) \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.6}$$

Il problema cinematico inverso si presenta nella forma di un sistema di 12 equazioni algebriche non lineari (vedi Equazione 2.6), delle quali solamente 6 indipendenti.

Le soluzioni del problema cinematico inverso di posizione possono:

- **non esistere**: in tal caso la configurazione obiettivo $T_{t,w}$ si trova al di fuori dello spazio raggiungibile dal manipolatore;
- essere **unica** o **multipla**: in quest'ultimo caso è possibile raggiungere la configurazione utensile richiesta con un numero finito di configurazioni, tra loro differenti, del braccio robotico;
- essere **infinite**: si parla allora di manipolatore *ridondante*. Quest'ultimo punto verrà ripreso nel Capitolo 4.

In generale, per calcolare le soluzioni del problema cinematico inverso di posizione è necessario ricorrere a metodi numerici.
Al fine di ottenere una soluzione in forma chiusa (analitica) è necessario disaccoppiare il sistema di equazioni:

$$T_{n,0}(\mathbf{q}) = A \tag{2.7}$$

Questo è possibile in presenza di un polso sferico o di almeno tre assi consecutivi paralleli.
I manipolatori industriali sono costruiti tenendo conto di questa esigenza di calcolo, infatti il computo delle soluzioni della cinematica inversa deve essere eseguito in *real-time* dal controller, in particolare, durante l'esecuzione di movimenti definiti nello spazio cartesiano [5].
Qualora il manipolatore non avesse il polso sferico, si dovrà ricorrere a metodi numerici, come l'algoritmo di Newton-Raphson, per la risoluzione approssimata

del sistema di equazioni nonlineari.

Nel caso di un robot antropomorfo, qual'è il Viper s650, il sistema è costituito da 6 equazioni non lineari nelle 6 incognite di giunto (rotazioni).
Essendo il polso sferico, l'origine della terna 5 (*centro polso*) dipende esclusivamente dalle rotazioni dei primi tre giunti. Ciò consente, perciò, di disaccoppiare il sistema, moltiplicando entrambi i membri della 2.7 per il vettore $\begin{Bmatrix} 0 & 0 & -d_6 & 1 \end{Bmatrix}^T$.

$$A \begin{Bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{Bmatrix}_6 = T_{60}(\mathbf{q}) \begin{Bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{Bmatrix}_6 \tag{2.8}$$

Da questo si ricava

$$\begin{Bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{Bmatrix}_0 = T_{30}(\vartheta_1, \vartheta_2, \vartheta_3) \begin{Bmatrix} a_3 \\ d_4 \\ 0 \\ 1 \end{Bmatrix}_3 \tag{2.9}$$

Pre-moltiplicando entrambi i membri per l'inversa della T_{10} otteniamo la rappresentazione del centro polso nel sistema 1, con un conseguente ulteriore disaccoppiamento di variabili.

$$T_{10}^{-1}(\vartheta_1) \begin{Bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{Bmatrix}_0 = T_{31}(\vartheta_2, \vartheta_3) \begin{Bmatrix} a_3 \\ d_4 \\ 0 \\ 1 \end{Bmatrix}_3 \tag{2.10}$$

Si determinano, proseguendo con metodi geometrici, quattro possibili soluzioni $(\vartheta_1, \vartheta_2, \vartheta_3)$ in funzione della posizione x, y, z dell'origine del sistema 6 rispetto alla base robot. In particolare, si avranno due valori possibili di ϑ_1 (braccio sinistro (*lefty*)/braccio destro (*righty*)) e due valori differenti di ϑ_3 (gomito alto (*above*)/gomito basso (*below*)) per ciascuna valore di ϑ_1. L'angolo ϑ_2, invece, è univocamente determinato come conseguenza delle scelte precedenti.

Al fine di determinare gli angoli del polso, riconsideriamo il sistema iniziale della 2.7:

$$A = T_{30}(\vartheta_1, \vartheta_2, \vartheta_3)T_{63}(\vartheta_4, \vartheta_5, \vartheta_6) \tag{2.11}$$

Essendo a questo punto nota la matrice T_{30}, pre-moltiplichiamo entrambi i membri per la sua inversa ottenendo

$$B = T_{30}^{-1}A = T_{63}(\vartheta_4, \vartheta_5, \vartheta_6) \tag{2.12}$$

dove B è una matrice, da ritenersi nota, la cui sotto-matrice di rotazione R_{63} contiene tutte le informazioni necessarie a ricavare gli angoli del polso. Anche qui, risolvendo il sistema algebrico nonlineare, si trovano altre due soluzioni $(\vartheta_1, \vartheta_2, \vartheta_3)$. In particolare, la soluzione con ϑ_5 positivo viene denominata "*no flip*", quella con ϑ_5 negativo "*flip*".

Per vedere le espressioni analitiche degli angoli soluzioni al problema cinematico inverso si rimanda ai testi [5] e [2].

Figura 2.6: Le 8 possibili configurazioni per un robot antropomorfo a polso sferico

In definitiva, si ottengono otto possibili soluzioni (*configurazioni*), che sono riassunte in Figura 2.6.

2.3 Analisi cinematica di velocità

La risoluzione del problema cinematico inverso consiste nell'ottenere i valori di q, $\dot{q}$, $\ddot{q}$ per ogni giunto, partendo dalla conoscenza delle matrici di posizione T_{0n},

di velocità $W_{0n(0)}$ e di accelerazione $H_{0n(0)}$ assolute della flangia robot, che sono funzione delle matrici di posizione, velocità e accelerazione relative ai link:

$$T_{0n} = T_{01}T_{12}\dots T_{n-1,n} = \prod_{i=1}^{n} T_{i-1,i}(q_i) \tag{2.13}$$

$$W_{0n} = \sum_{i=1}^{n} W_{i-1,i} = \sum_{i=1}^{n} L_{i-1,i}\dot{q}_i \tag{2.14}$$

$$H_{0n} = \sum_{i=1}^{n} H_{i-1,i} + \sum_{j=2}^{n}\sum_{k=1}^{j-1} 2W_{k-1,k}W_{j-1,j} \tag{2.15}$$

$$= \sum_{i=1}^{n}(L_{i-1,i}\ddot{q}_i + L_{i-1,i}^2\dot{q}_i^2) + 2\sum_{j=2}^{n}\sum_{k=1}^{j-1} L_{k-1,k}L_{j-1,j}\dot{q}_k\dot{q}_j \tag{2.16}$$

Le singole matrici W, H ed L hanno la forma:

$$W = \left[\begin{array}{ccc|c} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{2.17}$$

$$H = \left[\begin{array}{ccc|c} & & & a_x \\ & \underline{\omega^2} + \underline{\dot{\omega}} & & a_y \\ & & & a_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{2.18}$$

$$L = \left[\begin{array}{ccc|c} 0 & -u_z & u_y & t_x \\ u_z & 0 & -u_x & t_y \\ -u_y & u_x & 0 & t_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{2.19}$$

Immaginiamo di avere già risolto il problema cinematico di posizione e pertanto consideriamo noto il valore di tutte le variabili di giunto $\mathbf{q}$.
Scelto un sistema di riferimento qualsiasi (k), scrivendo per esteso la relazione

dell'Equazione 2.14 si ottiene:

$$\left[\begin{array}{ccc|c} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] = \left[\begin{array}{ccc|c} 0 & -u_z & u_y & t_{x1} \\ u_z & 0 & -u_x & t_{y1} \\ -u_y & u_x & 0 & t_{z1} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \dot{q}_1 + \cdots +$$

$$+ \left[\begin{array}{ccc|c} 0 & -u_z & u_y & t_{xi} \\ u_z & 0 & -u_x & t_{yi} \\ -u_y & u_x & 0 & t_{zi} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \dot{q}_i + \cdots + \left[\begin{array}{ccc|c} 0 & -u_z & u_y & t_{xn} \\ u_z & 0 & -u_x & t_{yn} \\ -u_y & u_x & 0 & t_{zn} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \dot{q}_n$$

Uguagliando gli elementi corrispondenti delle matrici di sinistra e di destra si ottiene il seguente sistema lineare:

$$\left[\begin{array}{cccccc} t_{x1} & t_{x2} & \dots & t_{xi} & \dots & t_{xn} \\ t_{y1} & t_{y2} & \dots & t_{yi} & \dots & t_{yn} \\ t_{z1} & t_{z2} & \dots & t_{zi} & \dots & t_{zn} \\ u_{x1} & u_{x2} & \dots & u_{xi} & \dots & u_{xn} \\ u_{y1} & u_{y2} & \dots & u_{yi} & \dots & u_{yn} \\ u_{z1} & u_{z2} & \dots & u_{zi} & \dots & u_{zn} \end{array}\right] \left[\begin{array}{c} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_i \\ \vdots \\ \dot{q}_n \end{array}\right] = \left[\begin{array}{c} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{array}\right] \tag{2.20}$$

cioè un sistema di sei equazioni in n incognite. Questo può essere scritto in forma compatta con ovvio significato dei termini come:

$$J\dot{\mathbf{q}} = \mathbf{v} \tag{2.21}$$

dove la matrice J è chiamata *matrice jacobiana* geometrica, poiché non implica nessuna derivata.

In alternativa la jacobiana può essere ottenuta in modo analitico, per differenziazione. Se chiamiamo $\mathbf{s}$ il vettore che contiene le coordinate della mano e con $\mathbf{q}$ il vettore contenente le coordinate ai giunti, indicheremo con la funzione $\mathbf{s} = f(\mathbf{q})$ la relazione tra esse. L'analisi di velocità e accelerazione può essere eseguita derivando rispetto al tempo l'equazione $\mathbf{s} = f(\mathbf{q})$:

$$\dot{\mathbf{s}} = \frac{d\mathbf{s}}{dt} = \frac{d\mathbf{f}}{d\mathbf{q}} \frac{d\mathbf{q}}{dt} \tag{2.22}$$

ove il termine $\dfrac{\partial \mathbf{f}}{\partial \mathbf{q}}$ è proprio la matrice jacobiana analitica.

Il significato fisico delle due formulazioni è lo stesso, ma la seconda può risultare più onerosa da calcolare dal punto di vista computazionale. Perciò, se sono note le caratteristiche geometriche del robot, è conveniente ricorrere alla jacobiana geometrica, che coinvolge solo le rotazioni dei giunti e non le loro derivate rispetto al tempo.

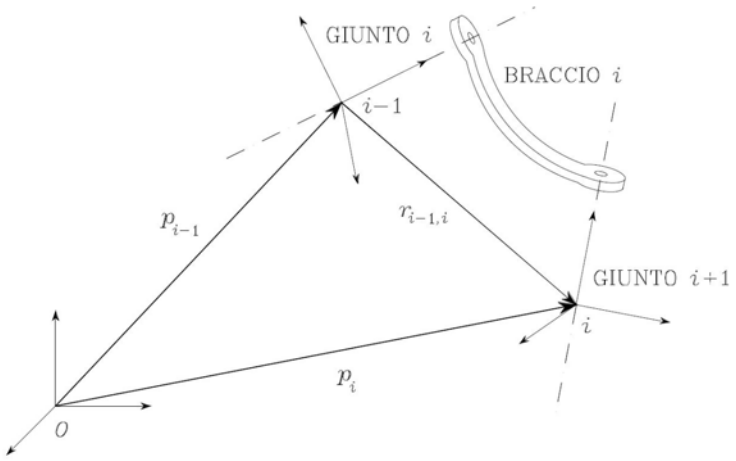


Figura 2.7: Notazione utilizzata nel calcolo delle velocità dei giunti

Per costruire la jacobiana geometrica è necessario definire le velocità lineari ed angolari di ciascun giunto rispetto alla base del robot. Si prenda a riferimento la Figura **??**.

$$p_i = p_{i-1} + R_{i-1} r_{i-1,i}^{i-1} \tag{2.23}$$

$$\dot{p}_i = \dot{p}_{i-1} + R_{i-1} \dot{r}_{i-1,i}^{i-1} + \omega_{i-1} \times R_{i-1} r_{i-1,i}^{i-1} = \dot{p}_{i-1} + v_{i-1,i} + \omega_{i-1} \times r_{i-1,i} \tag{2.24}$$

per la velocità lineare,

$$\omega_i = \omega_{i-1} + R_{i-1} \omega_{i-1,i}^{i-1} = \omega_{i-1} + \omega_{i-1,i} \tag{2.25}$$

per la velocità angolare, dove R_i è la matrice di rotazione dell'asse i-esimo rispetto alla base robot.
Le velocità lineari ed angolari dipendono dal tipo di giunti (coppie) presenti nel

	Giunto prismatico	**Giunto rotoidale**
$\omega_{i-1,1} =$	0	$\dot{\vartheta}_i z_{i-1}$
$v_{i-1,1} =$	$\dot{d}_i z_{i-1}$	$\omega_{i-1,i} \times r_{i-1,i}$
$\omega_i =$	ω_{i-1}	$\omega_{i-1} + \dot{\vartheta}_i z_{i-1}$
$v_i =$	$\dot{p}_{i-1} + \dot{d}_i z_{i-1} + \omega_i \times r_{i-1,i}$	$\dot{p}_{i-1} + \omega_i \times r_{i-1,i}$

Tabella 2.2: Velocità angolari e lineari per giunti prismatici e rotoidali

manipolatore e possono essere riassunte come in Tabella 2.2.

Ora si può definire la jacobiana come una matrice di dimensioni $6 \times n$, con n numero di gradi di libertà del manipolatore, avente per entrate dei termini che, combinati linearmente con le velocità generalizzate dei giunti, danno come risultato le velocità cartesiane dell'end-effector.

$$J = \left[\begin{array}{c|c|c} j_{P1} & \cdots & j_{Pn} \\ j_{O1} & \cdots & j_{On} \end{array}\right] \tag{2.26}$$

- Velocità lineare
  - giunto i prismatico: $\dot{q}_i j_{Pi} = \dot{d}_i z_{i-1} \;\Rightarrow\; j_{Pi} = z_{i-1}$
  - giunto i rotoidale: $\dot{q}_i j_{Pi} = \omega_{i-1,i} \times r_{i-1,n} = \dot{\vartheta}_i z_{i-1} \times (p - p_{i-1}) \;\Rightarrow\; j_{Pi} = z_{i-1} \times (p - p_{i-1})$
- Velocità angolare
  - giunto i prismatico: $\dot{q}_i j_{Oi} = 0 \;\Rightarrow\; j_{Oi} = 0$
  - giunto i rotoidale: $\dot{q}_i j_{Oi} = \dot{\vartheta}_i z_{i-1} \;\Rightarrow\; j_{Oi} = z_{i-1}$

Riassumendo, la i-esima colonna della jacobiana geometrica risulta essere:

$$\begin{bmatrix} j_{Pi} \\ j_{Oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

per un giunto *prismatico*,

$$\begin{bmatrix} j_{Pi} \\ j_{Oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \times (p - p_{i-1}) \\ z_{i-1} \end{bmatrix}$$

per un giunto *rotoidale*.
In generale, la i-esima colonna della jacobiana riferita alla flangia robot può anche essere calcolata dal prodotto $L_i T_{0n}$, secondo l'equazione 2.2.
Nel caso di un robot antropomorfo a 6 assi quale il Viper s650, è possibile sia calcolare una jacobiana "parziale" di dimensione 6×3, riferito al centro polso, se il task da eseguire coinvolge principalmente i primi tre giunti, sia lo jacobiana completa 6×6 riferito al centro della flangia robot.
La jacobiana riferita al centro polso si calcola come segue:

$$J_p = \begin{bmatrix} z_1 \times (p_4 - p_1) & z_2 \times (p_4 - p_2) & z_3 \times (p_4 - p_3) \\ z_1 & z_2 & z_3 \end{bmatrix}$$

con $s_i = sin(q_i)$, $c_i = cos(q_i)$, $s_{ij} = sin(q_i + q_j)$, $c_{ij} = cos(q_i + q_j)$, mentre la jacobiana completa si ottiene aggiungendo le ultime tre colonne e considerando come punto di riferimento p il centro della flangia p_6 o la punta dell'utensile p_t, anziché il centro polso p_4:

$$J = \begin{bmatrix} z_1 \times (p - p_1) & z_2 \times (p - p_2) & z_3 \times (p - p_3) & z_4 \times (p - p_4) & z_5 \times (p - p_5) & z_6 \times (p - p_6) \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 \end{bmatrix}$$

dove si ha che

$$p_i = T_{i0}(1:3, 4)$$
$$z_i = T_{i0}(1:3, 3)$$

con $i = 1, \ldots, 6, t$ e le T_{i0} sono le matrici di trasformazione ottenute alla fine della Sezione 2.2.1.
In Appendice si riporta l'espressione dettagliata della jacobiana completo.

Spesso può essere utile rappresentare la jacobiana in una terna differente rispetto a quella associata alla flangia del robot. Per fare ciò basta pre-moltiplicarla per una matrice 6×6 contenente la matrice di rotazione R^* [6][7]:

$$J^* = \left[\begin{array}{c|c} R^* & O \\ \hline O & R^* \end{array}\right] J \tag{2.27}$$

Durante il loro movimento, i manipolatori possono assumere particolari configurazioni dette "singolari" in cui lo jacobiano, cioè il determinante della matrice

J, è nullo; in tal caso, la matrice jacobiana non è invertibile e, avvicinandosi alla situazione di singolarità, alcuni elementi della matrice jacobiana inversa tendono a infinito. Nell'intorno di queste configurazioni potranno essere necessari grossi movimenti degli attuatori per effettuare piccoli movimenti della pinza [6] [2].
Nel caso di un robot antropomorfo, un'esempio di configurazione singolare si ha quando gli assi 4 e 6 risultano allineati.

2.3.1 Cinematica diretta

Note le velocità di ciascun giunto del manipolatore, si calcolano le velocità cartesiane dell'end-effector $\mathbf{v}$ moltiplicando la jacobiana J per il vettore delle velocità di giunto $\dot{\mathbf{q}}$ con la seguente equazione:

$$\mathbf{v} = J\dot{\mathbf{q}} \tag{2.28}$$

2.3.2 Cinematica inversa

Note le velocità cartesiane $\mathbf{v}$ da assegnare all'end-effector, si calcolano le velocità di giunto $\dot{\mathbf{q}}$ grazie all'inversa della matrice jacobiana J: la seguente equazione:

$$\dot{\mathbf{q}} = J^{-1}\mathbf{v} \tag{2.29}$$

La risoluzione del problema cinematico inverso di velocità richiede, quindi, la risoluzione di un sistema lineare avente per matrice dei coefficienti proprio la jacobiana J.
Come già detto, alcuni problemi possono insorgere in prossimità delle configurazioni singolari [2].

Nel caso in cui la jacobiana fosse rettangolare, ossia con un numero di righe diverso dal numero di colonne, sarà possibile invertirne solo una sotto-matrice quadrata. Questo è, ad esempio, il caso della jacobiana calcolata rispetto al centro polso di un robot antropomorfo, avente dimensioni 6×3; si può invertire solo una sotto-matrice 3×3 ottenendo le velocità di soli tre giunti.

Capitolo 3

Fase simulativa

3.1 MatLAB

MatLAB (abbreviazione di *Matrix Laboratory*) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C, che comprende anche l'omonimo linguaggio di programmazione creato dalla *MathWorks*. MatLAB consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, e interfacciarsi con altri programmi.
MatLAB diventa conveniente da usare anche in applicazioni di analisi cinematica di manipolatori industriali e ottimizzazione di traiettorie, proprio per la sua capacità di definire e manipolare matrici e vettori in maniera semplice. Come già visto, infatti, la trattazione matriciale è la più conveniente nell'analisi cinematica di sistemi spaziali in catena aperta.

3.2 La classe KINpro

La gestione dei calcoli per il movimento e della visualizzazione del robot può richiedere molte funzioni che, considerate separatamente, andrebbero ad appesantire il codice e ad obbligare l'integratore ad accompagnare il programma principale con decine di altri script, anche per la più semplice pianificazione.
Ciò rende vantaggioso l'utilizzo delle *classi*, ossia un insieme di *oggetti* con caratteristiche comuni. Gli oggetti sono, dunque, specifiche istanze di una classe. I valori contenuti nelle *proprietà* di un oggetto sono ciò che rende ciascun oggetto diverso dagli altri oggetti della stessa classe. Le proprietà memorizzano le

informazioni restituite negli oggetti. Le funzioni definite dalla classe, chiamate *metodi*, sono ciò che implementa i comportamenti dell'oggetto comuni a tutti gli oggetti di una classe. Il linguaggio MatLAB definisce oggetti progettati per l'uso in qualsiasi codice MatLAB.
La *programmazione orientata agli oggetti* ha come scopo la creazione di classi, a seguito dell'identificazione degli oggetti desiderati, della loro applicazione e della classificazione di questi in base a somiglianze e differenze. Ciò che ne risulta è una sorta di libreria da cui poter "pescare" gli oggetti desiderati e i relativi metodi.
L'opposto della programmazione orientata agli oggetti è la *programmazione procedurale*, dove ci si concentra sui passaggi che devono essere eseguiti per raggiungere lo stato desiderato [8].

Nel caso specifico di questa tesi, si è utilizzata una particolare classe, chiamata `@KINpro`, sviluppata dal gruppo di ricerca del Laboratorio di Robotica e Automazione dell'Università di Padova, con l'aiuto di alcuni tesisti. Tale classe rappresenta uno strumento estremamente vantaggioso perché gestisce in modo del tutto generale la creazione di un oggetto "robot", della sua rappresentazione grafica e della completa gestione del movimento (cinematica diretta e inversa, movimenti di *jump*, restituzione di informazioni sullo stato del robot, fine corsa e limiti di velocità dei giunti), sulla base del modello geometrico del robot di interesse e delle informazioni correlate, derivate dal datasheet. Per fare un esempio: se si vuole studiare un robot antropomorfo a 6-assi, è sufficiente importare da un file `.mat` la geometria del robot, le lunghezze dei link, i limiti di rotazione e le velocità massime dei singoli giunti, per poter creare l'oggetto "Antropomorfo" in grado di svolgere tutte le funzioni desiderate necessarie alla pianificazione del movimento. La classe KINpro riconosce il tipo di robot importato - SCARA, 6 assi a polso sferico, 6 assi UR, 7 assi - ed applica di conseguenza i metodi corrispondenti.
Un'altra classe utilizzata per la pianificazione è la classe `@trans` responsabile della creazione delle matrici di trasformazione necessarie per l'analisi cinematica.

3.3 Pianificazione cartesiana

La pianificazione del movimento effettuata in questo lavoro di tesi avviene principalmente nello spazio operativo. Questo perché vogliamo che il manipolatore utilizzato esegua con la punta del proprio *end-effector* un preciso percorso carte-

siano assegnato, entro un certo tempo T e secondo un'adeguata legge di moto. Infatti l'operazione di scrittura, così come la saldatura o l'incollaggio, richiede un'elevata precisione durante tutto il movimento e non può essere eseguita con una semplice pianificazione ai giunti con pochi *via-point*.
All'interno di questa stessa tesi, come vedremo nel Capitolo 4, verrà effettuata anche una pianificazione ai giunti, che, però, coinvolgerà solo l'ultimo giunto del robot, ossia quello controllabile ai fini dell'ottimizzazione.
Consideriamo innanzitutto il caso in cui una traiettoria da seguire sia facilmente esprimibile in forma analitica, come un segmento di retta o un arco di cerchio. Traiettorie più complesse si possono comporre assemblando più tratti semplici o approssimando i percorsi con un numero sufficiente di punti intermedi (sarà il caso, appunto, delle traiettorie qui chiamate *free-shape*).

Sia $\mathbf{P} = \mathbf{g}(\sigma)$ una generica curva nello spazio, funzione dell'unico parametro σ e passante per il punto iniziale $\mathbf{P_i} = \mathbf{g}(\sigma_i)$ e il punto finale $\mathbf{P_f} = \mathbf{g}(\sigma_f)$. Essendo di interesse il tratto di curva compreso tra i due punti, risulta utile riscrivere l'equazione della curva in funzione dell'ascissa curvilinea s, che misura la distanza percorsa lungo la curva:

$$\mathbf{P} = \mathbf{f}(s) = \begin{pmatrix} x(s) \\ y(s) \\ z(s) \end{pmatrix} \tag{3.1}$$

$$s = \int_{\sigma_i}^{\sigma} \sqrt{dx^2 + dy^2 + dz^2} = \int_{\sigma_i}^{\sigma} \sqrt{\frac{dx}{d\sigma}^2 + \frac{dy}{d\sigma}^2 + \frac{dz}{d\sigma}^2}\, d\sigma \tag{3.2}$$

Tranne casi particolari, in ogni punto $\mathbf{P}$ della curva sono definiti tre versori. Il primo è il versore tangente $\mathbf{t}$, il quale giace sulla retta tangente in $\mathbf{P}$ alla curva ed è orientato nel verso positivo indotto sulla curva da s. Il secondo è il versore normale $\mathbf{n}$. che insieme al versore tangente genera il *piano osculatore*. Il terzo è il versore binormale $\mathbf{b}$=$\mathbf{t} \times \mathbf{n}$, normale sia a $\mathbf{t}$ che a $\mathbf{n}$ e tale da rendere la terna ($\mathbf{t}$,$\mathbf{n}$,$\mathbf{b}$) levogira.

Per una particolare scelta del parametro s risulta:

$$\mathbf{t}(s) = \frac{d\mathbf{P}}{ds} = \begin{pmatrix} \frac{dx}{ds} \\ \frac{dy}{ds} \\ \frac{dz}{ds} \end{pmatrix} \tag{3.3}$$

$$\mathbf{n}(s) = \frac{1}{||\frac{d^2\mathbf{P}}{ds^2}||} \frac{d^2\mathbf{P}}{ds^2} \tag{3.4}$$

Preso un percorso $\mathbf{P}(s)$ si può generare una traiettoria assegnando una legge oraria $s(t)$. Tale legge del moto può essere determinata in maniera del tutto analoga al caso del moto punto-punto, tenendo conto del compito che deve svolgere il robot, del tempo di esecuzione e dei limiti meccanici del sistema.

Tra le tante *primitive del moto* oggi diffuse nel settore della robotica e automazione, qui si è impiegata la legge a profilo di velocità trapezoidale, per la sua semplicità di implementazione. In luogo di questa, si sarebbe potuta ugualmente impiegare una legge polinomiale di 3° o di 5° grado o una legge armonica (sinusoidale). Il requisito fondamentale di una legge oraria del moto è che sia una funzione continua e derivabile nell'intervallo considerato e che almeno la sua derivata prima (velocità) sia continua.
L'implementazione della legge a profilo di velocità trapezoidale è stata svolta col seguente codice:

```
1  %% Function che implementa la legge oraria del moto s(t)
2  % e le sue derivate nel tempo: ds,dds
3  % LEGGE CON PROFILO DI VELOCITA' TRAPEZOIDALE
4
5  function [s,ds,dds] = TrapVel(s_max,v_max,Ta,Td,t)
6
7      % Estraggo il fine movimento
8      T = t(end);
9
10     % Inizializzo i vettori di s, ds, dds
11     s = zeros(1,length(t)); % ascissa curvilinea
12     ds = zeros(1,length(t)); % derivata prima rispetto al tempo
```

```
13      dds = zeros(1,length(t)); % derivata seconda rispetto al ...
            tempo
14
15      for i = 1:length(t)
16          if t(i) <= Ta % Primo tratto
17              s(i) = 1/2*v_max*t(i)^2/Ta;
18              ds(i) = v_max*t(i)/Ta;
19              dds(i) = v_max/Ta;
20          elseif t(i) > Ta && t(i) <= (T-Td) % Tratto intermedio
21              s(i) = v_max*(t(i)-Ta/2);
22              ds(i) = v_max;
23              dds(i) = 0;
24          elseif t(i) > (T-Td) && t(i) <= T % Tratto finale
25              s(i) = s_max-1/2*v_max*((T-t(i))^2)/Td;
26              ds(i) = v_max*(T-t(i))/Td;
27              dds(i) = -v_max/Td;
28          end
29      end
30  end
```

La legge oraria del moto $s(t)$ fornisce l'andamento nel tempo di s, della sua derivata prima $\dot{s}$ e della sua derivata seconda $\ddot{s}$, ma è di maggior importanza conoscere le caratteristiche temporali del moto del punto $\mathbf{P}$.

$$\mathbf{P}(t) = \mathbf{P}(s(t)) \tag{3.5}$$

$$\dot{\mathbf{P}}(t) = \frac{d\mathbf{P}}{ds}\frac{ds}{dt} = \mathbf{t}\dot{s} \tag{3.6}$$

$$\ddot{\mathbf{P}}(t) = \frac{d\mathbf{P}}{ds}\frac{d^2s}{dt^2} + \frac{d^2\mathbf{P}}{ds^2}\left(\frac{ds}{dt}\right)^2 = \mathbf{t}\ddot{s} + ||\frac{d^2\mathbf{P}}{ds^2}||\mathbf{n}\dot{s}^2 \tag{3.7}$$

Si può notare che $\dot{\mathbf{P}}(t)$ ha una direzione uguale al vettore tangente e un modulo uguale a $\dot{s}$. L'accelerazione $\ddot{P}(t)$ ha invece una componente lungo $\mathbf{t}$ uguale a $\ddot{s}$ e una componente centripeta proporzionale a $\dot{s}^2||\frac{d^2\mathbf{P}}{ds^2}||$.

Ricapitolando, il problema della pianificazione cartesiana è stato scomposto nei seguenti sotto-problemi:

- scelta della forma della traiettoria $\mathbf{P}(s)$, parametrizzata rispetto all'ascissa curvilinea s;

- scelta della legge di moto di percorrenza della traiettoria $s(t)$, $\dot{s}(t)$, $\ddot{s}(t)$;
- scelta dei punti di campionamento della legge (vettore t dei tempi);
- calcolo del movimento della pinza del robot negli istanti di campionamento scelti $\mathbf{S}(t) = \mathbf{P}(s(t))$, $\dot{\mathbf{S}}(t)$, $\ddot{\mathbf{S}}(t)$;
- calcolo del corrispondente movimento dei motori $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$, $\ddot{\mathbf{q}}(t)$ tramite la cinematica inversa: $\mathbf{q} = G(\mathbf{S})$, $\dot{\mathbf{q}} = J^{-1}\dot{\mathbf{S}}$, $\ddot{\mathbf{q}} = J^{-1}\left(\ddot{\mathbf{S}} - \dot{J}\mathbf{q}\right)$.

3.3.1 Traiettorie rettilinee

Quando la curva $\mathbf{P}(s)$ si riduce a un segmento congiungente i punti $\mathbf{P_i}$ e $\mathbf{P_f}$ allora la sua rappresentazione parametrica diventa (per $0 \leq \mathrm{s} \leq ||\mathbf{P_f} - \mathbf{P_i}||$)

$$\mathbf{P}(s) = \mathbf{P_i} + \frac{\mathbf{P_f} - \mathbf{P_i}}{||\mathbf{P_f} - \mathbf{P_i}||} s \tag{3.8}$$

Derivando, si ottengono velocità, accelerazione e i versori caratteristici della curva:

$$\mathbf{t} = \frac{d\mathbf{P}}{ds} = \frac{\mathbf{P_f} - \mathbf{P_i}}{||\mathbf{P_f} - \mathbf{P_i}||} \tag{3.9}$$

$$\dot{\mathbf{P}}(t) = \frac{\mathbf{P_f} - \mathbf{P_i}}{||\mathbf{P_f} - \mathbf{P_i}||} \dot{s} = \mathbf{t}\dot{s} \tag{3.10}$$

$$\ddot{\mathbf{P}}(t) = \mathbf{t}\ddot{s} \tag{3.11}$$

3.3.2 Traiettorie circolari

Si consideri ora una circonferenza nel piano, centrata nel punto $\mathbf{C}$ e avente raggio R e lungo la quale viaggia il punto $\mathbf{P}$. Il percorso seguito da $\mathbf{P}$ è quindi un arco di cerchio di lunghezza s e può essere descritto completamente tramite l'angolo $\theta = \frac{s}{R}$:

$$\mathbf{P} = \mathbf{C} + R \begin{pmatrix} \cos\left(\frac{s}{R}\right) \\ \sin\left(\frac{s}{R}\right) \end{pmatrix} \tag{3.12}$$

dalla quale si ottengono rapidamente la velocità e l'accelerazione del punto $\mathbf{P}$:

$$\mathbf{t} = \frac{d\mathbf{P}}{ds} = \begin{pmatrix} -\sin\left(\frac{s}{R}\right) \\ \cos\left(\frac{s}{R}\right) \end{pmatrix} \tag{3.13}$$

$$\dot{\mathbf{P}}(t) = \begin{pmatrix} -\sin\left(\frac{s}{R}\right) \\ \cos\left(\frac{s}{R}\right) \end{pmatrix} \dot{s} = \mathbf{t}\dot{s} \tag{3.14}$$

$$\ddot{\mathbf{P}}(t) = \begin{pmatrix} -\sin\left(\frac{s}{R}\right) \\ \cos\left(\frac{s}{R}\right) \end{pmatrix} \ddot{s} + \frac{1}{R} \begin{pmatrix} -\cos\left(\frac{s}{R}\right) \\ -\sin\left(\frac{s}{R}\right) \end{pmatrix} \dot{s}^2 = \mathbf{t}\ddot{s} + \frac{1}{R}\mathbf{n}\dot{s}^2 \tag{3.15}$$

3.3.3 Traiettorie free-shape

Nel caso di traiettorie complesse *free-shape*, definite a mano libera tramite una periferica di input quale un mouse o una tavoletta grafica, chiaramente non esiste un'espressione analitica della curva da percorrere. La forma della curva viene costruita come una sequenza di punti, corrispondenti alle posizioni del mouse all'interno di una finestra grafica. Tali posizioni vengono acquisite con un apposito contatore richiamando la funzione `get(0, 'PointerLocation')`:

```
1  %% TEMPO
2  % Parametri di controllo di una legge oraria s(t) con profilo ...
       di velocità
3  % trapezoidale simmetrico
4
5  T = 5; %[s] TEMPO max di movimento
6  dt = 0.05; % step temporale di movimento
7  t = 0:dt:T; %[s] VETTORE TEMPO
8
9  %% Definizione di un PIANO DI SCRITTURA
10
11 % Origine della terna Tsw del piano rispetto alla terna world
12 Ps = [300 0 -5.5]; %per piano orizzontale (theta = 0°)
13
14 % Terna associata al piano di lavoro (surface) rispetto alla ...
       terna world
15 theta = 0; % [°] angolo di inclinazione del piano
```

```
16 Tsw = trans.Eulero(Ps(1),Ps(2),Ps(3),0,180,180)*trans.Y(theta);
17
18 % Definizione della base robot
19 T0w = trans.P(0,0,0); % trasformazione terna da base 0 a world
20
21 %%  Definizione della TRAIETTORIA CARTESIANA
22 % Caratteristiche del percorso
23
24 % CONTATORE CON CICLO WHILE
25 % Acquisizione della posizione del pointer del mouse
26 LOC = [];
27 time = 0;
28
29 hf = figure(1);
30 clf
31 hf.Position = [20 20 1700 800]; hold on;
32
33 w = waitforbuttonpress;
34 while length(LOC) < length(t)
35     loc = get(0, 'PointerLocation');
36     LOC = [LOC; loc];
37     time = time + dt;
38     disp(time)
39     pause(dt);
40 end
41
42 % fattori di scala (mm/pixel) basati su dimensioni e ...
       risoluzione del monitor
43 scala1=567/1920; %direzione x
44 scala2=421/1080; %direzione y
45
46 % Visualizzo in un grafico il tratto disegnato
47 plot(LOC(:,1),LOC(:,2));
48 hold off;
49
50 start = [10 30];
```

```
51 LOC = [scala1*LOC(:,1),scala2*LOC(:,2)]+start;
```

Successivamente, vengono differenziate numericamente rispetto al tempo le coordinate $(x\ y)$ delle posizioni memorizzate, viene calcolata l'ascissa curvilinea e le sue derivate (in questo caso non viene assegnata una legge oraria, bensì viene seguito l'andamento temporale col quale si è disegnata la curva). Inoltre, le derivate dell'ascissa curvilinea vengono filtrate con un filtro passa-basso, per ridurre il rumore dovuto ai tremolii del mouse o della penna sulla tavoletta grafica.

```
1  dPdt = [[0;diff(LOC(:,1))/dt] [0;diff(LOC(:,2))/dt]]; ...
       %differenzio rispetto a t
2  s_max = sum(sqrt(dPdt(:,1).^2+dPdt(:,2).^2)*dt); %integro da ...
       0 a T
3
4  %% ASCISSA CURVILINEA
5
6  % Calcolo il vettore s(t)
7  s = cumsum(sqrt(dPdt(:,1).^2+dPdt(:,2).^2)*dt);
8  ds = [0; diff(s)./dt];
9  dds = [0; diff(ds)./dt];
10
11 % Filtro passa-basso per ridurre il rumore in ingresso
12 ds = lowpass(ds,100,300,'Steepness',0.5);
13 dds = lowpass(dds,100,300,'Steepness',0.5);
```

A questo punto, si costruiscono i vettori velocità e accelerazione per ogni istante e si calcolano la curvatura locale k ed il raggio di curvatura locale R, secondo le seguenti equazioni:

$$k = \frac{|\dot{x}\ddot{y} - \dot{y}\ddot{x}|}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}} \tag{3.16}$$

$$R = \frac{1}{R} \tag{3.17}$$

e si "lisciano" le rispettive curve con la function `smooth`, che esegue una media mobile.

```
1  dP = zeros(length(t),2);
```

```
ddP = zeros(length(t),2);
dx = dPdt(:,1); %velocità istantanea lungo x
dy = dPdt(:,2); %velocità istantanea lungo y
ddx = [0;diff(dx/dt]; %accelerazione istantanea lungo x
ddy = [0;diff(dy/dt]; %accelerazione istantanea lungo y
R = zeros(length(t),1); %raggio di curvatura locale
for i = 2:length(t)
    dP(i,:) = [dx(i) dy(i)];
    ddP(i,:) = [ddx(i) ddy(i)];
    % Calcolo del raggio di curvatura locale
    k = ...
        abs(dx(i)*ddy(i)-dy(i)*ddx(i))/(dx(i)^2+dy(i)^2)^(3/2); ...
        %curvatura
    if isnan(k) == 1
        R(i,:) = 0;
    elseif k < 0.001
        R(i,:) = 1000;
    else
        R(i,:) = 1/k; %raggio di curvatura
    end

end

% Filtro posizioni e velocità cartesiane
P = [smooth(LOC(:,1),7) smooth(LOC(:,2),7)];
dP = [smooth(dP(:,1),7) smooth(dP(:,2),7)];
R = smooth(R,7);
```

Infine, si costruiscono le matrici delle posizioni e delle velocità in coordinate omogenee nel sistema di riferimento assoluto *world* e si salvano per poterle utilizzare negli altri script.

```
%% TRAIETTORIA LIBERA IN COORDINATE WORLD
% Calcolo P_s (4 x length(t))
% Contiene le coordinate del punto P 4x1, per ogni istante t

```

```
5  % TRAIETTORIA LIBERA
6  P_s = zeros(4,length(t));
7  P_s_dot = zeros(4,length(t));
8  for i = 1:length(t)
9      P_s(1:4,i) = Tsw*[P(i,2) P(i,1) 0 1]';
10     if norm(dP(i,:)) == 0
11         P_s_dot(1:4,i) = [0 0 0 0]';
12     else
13         P_s_dot(1:4,i) = [dP(i,1) dP(i,2) 0 ...
               0]'/norm(dP(i,:)).*ds(i);
14     end
15 end
```

Capitolo 4

Criteri di ottimizzazione

4.1 Robot ridondanti

4.1.1 Spazio dei giunti e spazio operativo

Come già visto nei Capitoli 2 e 3, lo scopo della cinematica di posizione è identificare le relazioni $\mathbf{S} = G(\mathbf{q})$ e $\mathbf{q} = G^{-1}(\mathbf{S})$ tra posizione dei giunti $\mathbf{q} = \{\theta_1\ \theta_2\ \ldots\ \theta_n\}$ e posa dell'end-effector $\mathbf{S} = \{x\ y\ z\ \alpha\ \beta\ \gamma\}$.

$$\mathbf{q} \in \Re^n$$

$$\mathbf{S} \in \Re^{m=6}$$

dove n è il numero di gradi di libertà del manipolatore, mentre m è la dimensione dello spazio operativo. Inoltre, definiamo r il numero di gradi di libertà del compito (task) da svolgere.
La relazione tra **spazio dei giunti** R^n e **spazio operativo** R^m può essere rappresentata graficamente come in Figura 4.1.

4.1.2 Ridondanza cinematica

La ridondanza cinematica si verifica quando un manipolatore robotico ha più gradi di libertà di quelli strettamente richiesti per eseguire una determinata attività. Ciò significa che, in linea di principio, nessun manipolatore è intrinsecamente ridondante; piuttosto, ci sono alcuni compiti rispetto ai quali esso può diventare

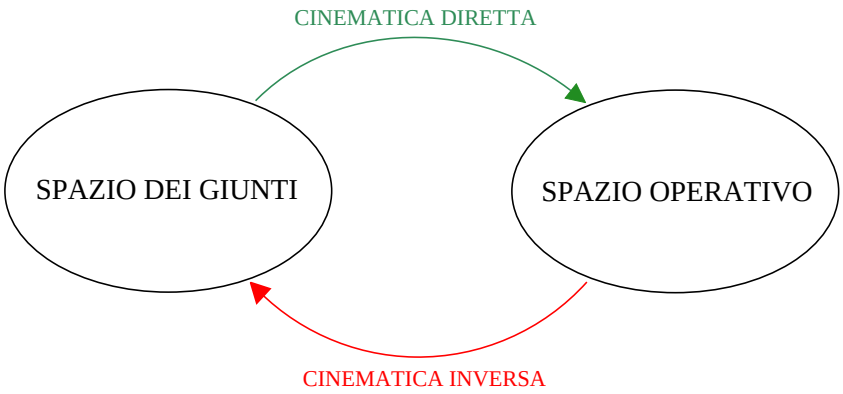


Figura 4.1: Spazio dei giunti e spazio operativo

ridondante. Poiché è ampiamente riconosciuto che un compito spaziale completamente generale consiste nel seguire una traiettoria di movimento dell'effettore finale che richiede sei gradi di libertà, un braccio robotico con sette o più articolazioni è spesso considerato come un tipico esempio di un manipolatore intrinsecamente ridondante. Tuttavia, anche i bracci del robot con meno gradi di libertà, come i tradizionali manipolatori industriali a sei giunti, possono diventare cinematicamente ridondanti per compiti specifici, come il semplice posizionamento dell'end-effector senza vincoli sull'orientamento.
La motivazione per l'introduzione della ridondanza cinematica nella struttura meccanica di un manipolatore va oltre quella per l'utilizzo della ridondanza nella progettazione ingegneristica tradizionale, vale a dire aumentare la robustezza ai guasti in modo da migliorare l'affidabilità (ad esempio, processori o sensori ridondanti). Infatti, dotare i manipolatori robotici di ridondanza cinematica è finalizzato principalmente ad aumentare la destrezza, come nel caso di un braccio umano.
L'approccio a complessità minima che caratterizzava i primi progetti di manipolatori aveva l'obiettivo di ridurre al minimo i costi e la manutenzione. Ad esempio, ciò ha portato allo sviluppo di robot SCARA per operazioni di pick-and-place in cui i prodotti erano stati progettati per l'assemblaggio, cioè, utilizzando un unico asse di inserimento. Tuttavia, fornire a un manipolatore il numero minimo di giunti necessari per eseguire il suo compito si traduce in una grave limitazione nelle applicazioni del mondo reale in cui, oltre al problema di singolarità, sono presenti limiti di giunti o ostacoli nell'area di lavoro. Tutti questi danno luogo

a zone proibite nello spazio dei giunti che devono essere evitate durante il funzionamento, richiedendo quindi uno spazio di lavoro attentamente strutturato, dove il movimento del manipolatore può essere pianificato in anticipo; questa è la situazione tipica delle celle di lavoro nelle applicazioni industriali tradizionali. D'altra parte, la presenza di gradi di libertà aggiuntivi oltre a quelli strettamente necessari per eseguire il compito consente movimenti del manipolatore che non spostano l'end-effector; ciò implica che lo stesso compito a livello di end-effector può essere eseguito in diversi modi a livello di giunto, dando la possibilità di evitare le regioni proibite e, in definitiva, risultando in un meccanismo più versatile. Questa caratteristica è fondamentale per consentire il funzionamento in ambienti non strutturati o dinamicamente variabili che caratterizzano applicazioni industriali avanzate e scenari di robotica di servizio.
In pratica, se adeguatamente gestita, la maggiore destrezza che caratterizza i robot ridondanti cinematicamente può consentire loro non solo di evitare singolarità, limiti articolari e ostacoli nell'area di lavoro, ma anche di ridurre al minimo il tempo o l'energia spesi per un determinato compito.
L'archetipo biologico di un manipolatore ridondante cinematicamente è il braccio umano, che, non a caso, ispira anche la terminologia utilizzata per caratterizzare la struttura dei manipolatori a catena seriale. Il braccio umano ha infatti tre gradi di libertà alla spalla, un grado di libertà al gomito e tre gradi di libertà al polso. La ridondanza disponibile può essere facilmente verificata bloccando il polso, ad esempio, su un tavolo e muovendo il gomito mentre si tiene ferma la spalla. La disposizione cinematica del braccio umano è stata replicata in un certo numero di robot spesso definiti come manipolatori antropomorfi [9].

task per l'organo terminale	**gdl del task r**
posizionamento nel piano	2
posizionamento nello spazio	3
orientamento nel piano	1
puntamento nello spazio	2
posizione e orientamento nello spazio	6
scrittura nello spazio	5

Tabella 4.1: Alcuni compiti (*tasks*) e la loro dimensione

Mentre la cinematica diretta di posizione è sempre possibile, quella inversa può portare a diverse soluzioni:

- nessuna soluzione, se il punto è fuori dallo spazio raggiungibile
- una soluzione unica, se il punto è raggiungibile con un'unica configurazione (ad esempio, perché raggiunto a braccio completamente esteso)
- soluzioni multiple, se il punto può essere raggiunto con più configurazioni (ad esempio, 2 per lo SCARA, 8 per l'antropomorfo)
- infinite soluzioni, se il punto può essere raggiunto con infinite configurazioni. Diremo, quindi, che il robot è ridondante:
  - intrinsecamente, se $n > m$
  - funzionalmente, se $n > r$

La ridondanza può essere usata per evitare ostacoli (nello spazio cartesiano), evitare singolarità cinematiche (nello spazio dei giunti), aumentare la manipolabilità in determinate direzioni, distribuire uniformemente/limitare le velocità di giunto, rimanere entro i limiti di fondo corsa dei giunti, minimizzare il consumo di energia, ottimizzare il tempo di percorrenza o le coppie richieste, aumentare l'affidabilità rispetto a guasti.

Il robot Viper s650, essendo un antropomorfo a 6 assi, risulta essere funzionalmente ridondante rispetto al compito della scrittura su una superficie qualsiasi nello spazio ($r = 5$), poiché la rotazione attorno all'asse dell'utensile è ininfluente sul risultato.
L'idea, dunque, è quella di sfruttare la ridondanza a proprio favore, rendendo l'asse del tool eccentrico rispetto all'asse 6 del manipolatore. In questo modo, per ogni posizione del centro polso avremo infinite possibilità di posizionamento della punta del tool.

4.1.3 End-effector a sbalzo

Per le operazioni prese in considerazione nel progetto Chocobot, ad esempio la deposizione di un adesivo lungo un particolare percorso sul pezzo oppure in ambito alimentare la decorazione con un fluido viscoso della superficie di una torta, viene preso in considerazione un organo terminale a sbalzo che possa reggere il

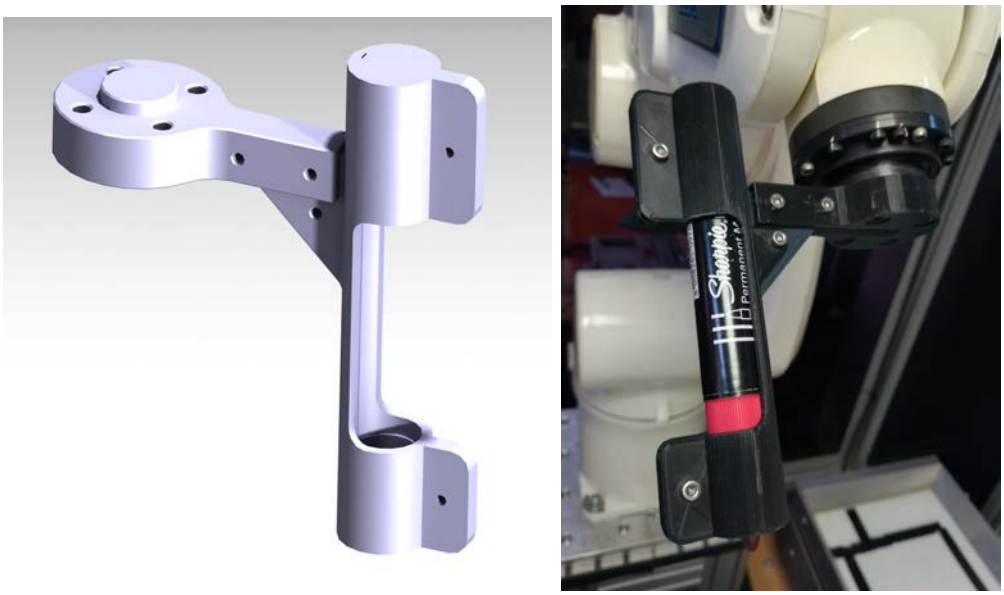


Figura 4.2: Utensile a sbalzo utilizzato nelle prove sperimentali: modello CAD 3D realizzato con CATIA V5 (a sinistra) e prototipo stampato con tecnologia FDM (a destra)

dosatore.
Nella presente tesi, non essendo ancora disponibile un sistema di dosaggio, è stato progettato e prodotto un end-effector fuori asse rispetto all'asse 6, che potesse essere un'alternativa alla testa di stampa con il materiale decorativo.
L'end-effector è stato creato con uno sbalzo regolabile tra 70 mm e 100 mm per sfruttare la ridondanza cinematica data dall'ultimo giunto e poter minimizzare l'energia utilizzata (vedi Figure 4.2 e 4.3).
Non disponendo del materiale decorativo, si è utilizzato un pennarello che potesse rappresentare la deposizione del materiale stesso. L'end-effector è stato stampato in tre parti con prototipazione rapida (tecnologia FDM) e garantisce sia la presa che la stabilità del pennarello. Inoltre, è stato progettato un piccolo vano nella parte superiore del cilindro contenente il pennarello (in cui è stata inserita una molla), per consentire uno scorrimento dello stesso ed evitare la rottura della punta in caso di un errore della quota z.
L'accoppiamento con la flangia robot è stato fatto tramite quattro viti M5 a testa cilindrica come suggerito dal manuale del robot Omron.

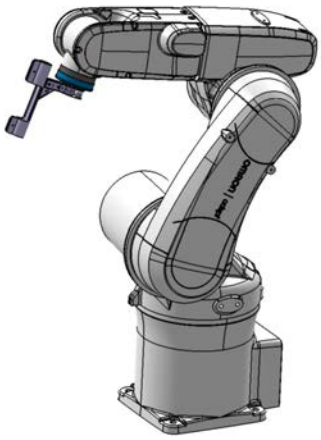


Figura 4.3: Utensile a sbalzo montato sul manipolatore

4.2 Struttura dell'ottimizzazione

In questa sezione vengono spiegati passo passo il contenuto dei codici sviluppati su MatLAB ed il loro utilizzo.
Il codice sviluppato per la presente tesi comprende quattro script fondamentali:

1. Definizione della traiettoria cartesiana con relativa legge oraria, delle terne della base robot, del piano di scrittura (*surface*) e dell'offset del tool. La legge oraria qui utilizzata è del tipo a profilo di velocità trapezoidale.

2. Simulazione di primo tentativo, con cinematica inversa sulla traiettoria cartesiana definita al punto (1). Definizione di una legge "conveniente" di primo tentativo per la rotazione γ della flangia del robot rispetto all'asse x del piano di scrittura.

3. Ottimizzazione della velocità: a partire dai valori di γ ottenuti al punto (2), calcolo iterativo di una costante ottimale γ_0 da sommare al vettore delle γ, tale da minimizzare l'indice $I = max(\lambda_i)$. Applicazione di un metodo numerico di ottimizzazione, attraverso la funzione (`fmincon`), per ottenere una legge oraria ottima, costituita da n polinomi cubici raccordati, per l'angolo γ.

4. Simulazione ottima, con l'applicazione della legge ottima di γ ricavata al punto (3), con aggiunta di movimenti di *appro* e *depart*.

I quattro script sono da eseguire nell'ordine con cui sono elencati e comunicano attraverso variabili salvate in apposite strutture dati del tipo `.mat`.
Ora, andiamo ad illustrare in maggior dettaglio il contenuto di ciascuno script.

Il primo script, ossia quello dedicato alla definizione della traiettoria cartesiana, implementa le formule e i metodi descritti nel paragrafo 3.3.
Innanzitutto, si definiscono la durata massima del movimento T, la risoluzione temporale dt, il vettore del tempo $t = 0 : dt : T$ ed i tempi di accelerazione T_a e decelerazione T_d per il profilo di velocità trapezoidale da assegnare alla punta dell'utensile lungo il percorso cartesiano da definire. Vengono dichiarate anche la terna della base robot T_{w0} e la terna del piano di scrittura T_{sw} nel sistema *world*.
A questo punto, si definiscono le caratteristiche geometriche del percorso cartesiano, che può essere un segmento di retta, un arco di circonferenza o una curva free-shape, ciascuno con la propria lunghezza complessiva. A partire da quest'ultima, vengono definite l'ascissa curvilinea s e le relative velocità ds ed accelerazioni dds, secondo il profilo di velocità trapezoidale. Quindi, vengono calcolate le coordinate spaziali P_s di tutti i punti della traiettoria e i vettori delle velocità tangenziali $\dot{P}_s = \frac{dP_s}{dt}$ in ogni istante del movimento nel sistema *world*.
Infine, si inserisce l'offset d_y dell'utensile rispetto all'asse del giunto 6 del manipolatore. Tutte le variabili vengono salvate nel file `dati.mat`.

Nel secondo script, si caricano le variabili relative alla traiettoria dal file `dati.mat`, quindi, grazie ai metodi della classe `KINpro`, si definiscono l'oggetto robot, la terna utensile $T_{t6} = T_p(0, d_y, d_z)$ e si impostano le proprietà di visualizzazione. Si disegna il piano di lavoro utilizzando la funzione *patch*.
A questo punto, si dichiara la configurazione del robot antropomorfo, necessaria al calcolo della cinematica inversa. Nelle simulazioni eseguite, il robot è stato configurato in braccio sinistro (*lefty*), gomito alto (*above*), polso non flippato (*noflip*).
Le velocità cartesiane v del punto P nella propria terna mobile sono memorizzate in un array di dimensioni $6 \times 1 \times length(t)$. Gli unici elementi non nulli di questo array sono quelli in posizione $(1, 1)$ di ciascun livello, in particolare sono posti uguali agli elementi del vettore ds delle derivate temporali dell'ascissa curvilinea definite nel primo script: ciò significa che la velocità v è tangente alla traiettoria

in ciascun punto di essa.

$$v = \begin{bmatrix} ds \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

A questo punto, con un ciclo `for` avente tante iterazioni quante sono le colonne della matrice delle coordinate cartesiane P_s, ad ogni iterazione i definisco la terna mobile lungo la traiettoria, avente asse x tangente alla traiettoria in quel punto, asse z entrante nel piano di scrittura e asse y fissato di conseguenza (la terna deve risultare levogira). A tale terna è, dunque, associata la matrice T_{pw} così costruita:

$$T_{pw} = \left[\begin{array}{c|c|c|c} \dot{P}_s(i) & \dot{P}_s(i) \times P_s(i) & T_{sw}(:,3)(i) & P_s(i) \\ 0 & 0 & 0 & 1 \end{array}\right]$$

La terna del punto P rispetto alla base robot, si ottiene quindi col seguente calcolo:

$$T_{p0} = T_{pw} T_{0w}^{-1}$$

La matrice di trasformazione tra la terna del tool e la base robot è quindi:

$$T_{t0} = \left[\begin{array}{c|c} T_{sw}(1:3,1:3) \cdot \begin{bmatrix} \cos(\gamma(i-1)) & -\sin(\gamma(i-1)) & 0 \\ \sin(\gamma(i-1)) & \cos(\gamma(i-1)) & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{matrix} T_{p0}(1,4) \\ T_{p0}(2,4) \\ T_{p0}(3,4) \end{matrix} \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right]$$

Infine, la matrice di trasformazione tra la terna del tool e la terna *world* si ottiene come:

$$T_{tw} = T_{t0} T_{0w}$$

Ora si hanno tutte le informazioni necessarie al calcolo della cinematica inversa di posizione, che viene effettuata invocando l'apposita function della classe *KINpro*, la quale prende in input la T_{tw} dell'iterazione i corrente e la configurazione impostata e restituisce come output il vettore delle rotazioni dei giunti, denominato q_0. Se la posizione è raggiungibile, il robot viene mandato in tale posizione ai giunti.

Oltre alla cinematica inversa di posizione, è necessario calcolare la cinematica inversa di velocità, visto che vogliamo che la punta dell'utensile si muova lungo la traiettoria secondo la legge di moto pianificata nel primo script. Per fare ciò, si richiama la function `jacobiana` per il calcolo della jacobiana completa J secondo il criterio illustrato al paragrafo 2.3.
La jacobiana va però cambiata di riferimento, trasformandola dalla flangia alla base robot, tramite la matrice di rotazione R_{06}:

$$J' = \left[\begin{array}{c|c} R_{06} & O \\ \hline O & R_{06} \end{array}\right] J$$

Anche la velocità cartesiana v espressa nella terna mobile va trasformata nella base robot, tramite la matrice di rotazione R_z:

$$R_z = \left[T_{p0}(1:3, 1:3)\right]$$

$$v_0 = \left[\begin{array}{c|c} R_z & O \\ \hline O & R_z \end{array}\right] v = \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Possiamo, quindi, calcolare le velocità dei primi cinque giunti del robot applicando la cinematica inversa di velocità:

$$\dot{q}_0 = \begin{bmatrix} J'_{11} & \dots & J'_{15} \\ \vdots & \ddots & \vdots \\ J'_{51} & \dots & J'_{55} \end{bmatrix}^{-1} \begin{bmatrix} v_{01} \\ v_{02} \\ v_{03} \\ v_{04} \\ v_{05} \end{bmatrix} \cdot \frac{180}{\pi}$$

moltiplicato per $180/\pi$ per ottenere delle velocità di rotazione in $[/s]$. La velocità del giunto 6 non è calcolabile, poiché il manipolatore è ridondante e, di conseguenza, esistono infinite soluzioni.
Tra le possibili scelte, si impone che il giunto 6 abbia una velocità pari alla media

delle velocità degli altri giunti, calcolate con la cinematica inversa, moltiplicata per il rapporto tra i limiti di velocità del giunto 6 e del giunto più lento:

$$\dot{q}_6 = \frac{1}{5}\sum_{i=1}^{5} \dot{q}_{0i} \cdot \frac{\dot{q}_{6lim}}{\min(\dot{q}_{1-5lim})}$$

in modo tale che abbia un rapporto di velocità prossimo a quello degli altri giunti. Si definisce, quindi, un vettore w di fattori peso per il calcolo della velocità di correzione della flangia $\dot{\gamma}$, come segue:

$$w' = \frac{\dot{q}_{6lim}}{\dot{q}_{1-5lim}}$$

$$w = \frac{w'}{||w'||} \cdot m = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix}$$

dove $\dot{q}_{6lim}$ è la velocità limite del giunto 6 e $\dot{q}_{1-5lim}$ è un vettore delle velocità limite dei giunti da 1 a 5, mentre m è un coefficiente legato al rapporto tra raggio di curvatura della traiettoria R e offset dell'utensile d_y, così definito:

$$m = \begin{cases} 1 & \text{se } R > d_y \\ 0 & \text{se } R \leq d_y \end{cases}$$

nel caso di traiettorie circolari o free-shape, mentre:

$$m = \begin{cases} 1 & \text{se } l > 3d_y \\ 0 & \text{se } l \leq 3d_y \end{cases}$$

nel caso di traiettorie rettilinee, con l la lunghezza del segmento di retta.
Tale criterio deriva dal fatto che, nel caso di raggio di curvatura R piccolo e offset d_y grande, non sia conveniente che la velocità di rotazione della flangia $\dot{\gamma}$ rispetto alla base robot sia maggiore della velocità di rotazione $\dot{q}_6$ dell'asse 6, poiché, in tal caso, il fuori asse dell'utensile giocherebbe un ruolo importante e grandi correzioni della flangia produrrebbero movimenti esagerati del manipolatore anche per piccoli spostamenti del tool, situazione certo non ottimale. Nel caso di traiettorie rettilinee, in cui non vi sono variazioni nella curvatura, il parametro di correlazione tra geometria del percorso e geometria del tool potrebbe essere proprio la lunghezza l del segmento: nel caso in cui questa sia sufficientemente maggiore dell'offset del tool - diciamo, almeno 3 volte -, il tool potrà ruotare maggiormente, compiendo una sorta di "inversione"; in caso contrario, conviene che la velocità

di correzione $\dot{\gamma}$ non superi la velocità di rotazione $\dot{q}_6$ dell'asse 6.
Perciò, la velocità di correzione del tool $\dot{\gamma}$ all'iterazione i-esima viene assegnata come segue:

$$\dot{\gamma}(i) = \dot{q}_6 + \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \begin{bmatrix} \dot{q}_{01} \\ \dot{q}_{02} \\ \dot{q}_{03} \\ \dot{q}_{04} \\ \dot{q}_{05} \end{bmatrix}$$

La rotazione γ all'iterazione i-esima si ottiene come somma cumulata della $\dot{\gamma}$ dalla iterazione 1 all'iterazione i (*integrazione numerica*):

$$\gamma(i) = \sum_{k=1}^{i} \dot{\gamma}_k \cdot dt$$

Il vettore delle γ rappresenta già il risultato di un primo step di ottimizzazione, poiché è stato ottenuto imponendo che tutti i giunti del robot abbiano all'incirca lo stesso rapporto di velocità, in modo tale che il movimento sia scalabile rispetto al tempo.
I grafici prodotti in questo secondo script sono i seguenti:

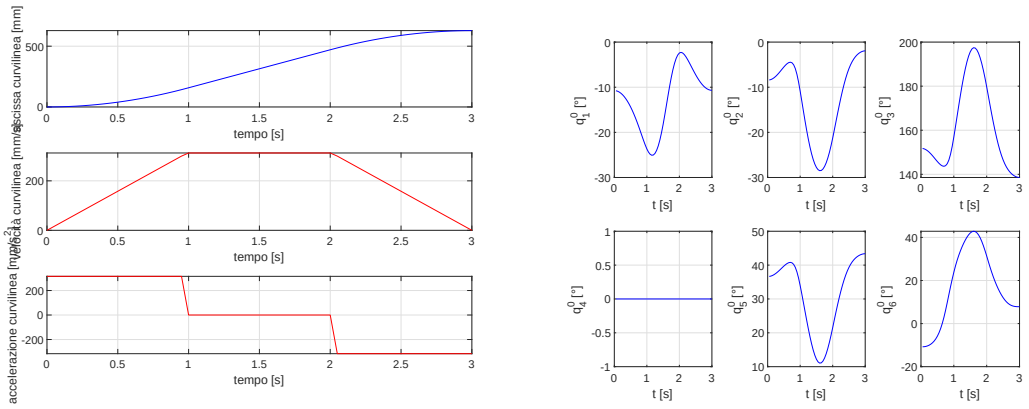


Figura 4.4: Ascissa curvilinea (a sinistra) e rotazioni di giunto di primo tentativo (a destra)

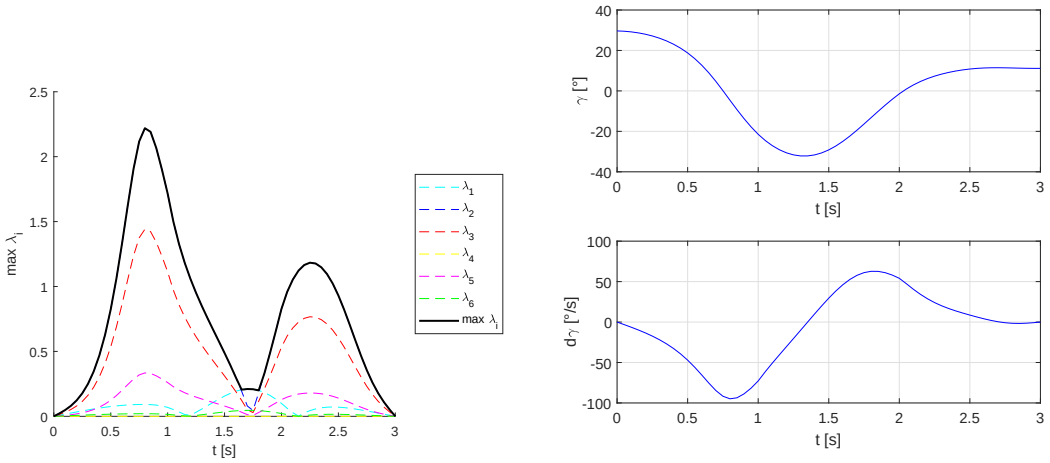


Figura 4.5: Rapporti di velocità λ_i (a sinistra) e correzione γ di primo tentativo (a destra)

4.3 Criterio analitico per γ (*gamma*)

Nel terzo script vengono implementati tre diversi algoritmi di ottimizzazione. Il primo criterio, che possiamo ritenere analitico, consiste nel riprendere il vettore delle γ ottenuto dal secondo script e nel cercare una costante γ_0 da sommarvi, tale da minimizzare l'indice I definito come il massimo dei rapporti tra la velocità massima in modulo e la velocità limite dei singoli giunti del manipolatore:

$$I = max(\lambda_i) \; con \; \lambda_i = \| \frac{\dot{q}_{i_{max}}}{\dot{q}_{i_{lim}}} \|$$

Il γ_0 viene calcolato nel range $-180 \div 180$ con una risoluzione di 1°, eseguendo ad ogni iterazione la cinematica inversa di posizione e calcolando le velocità dei giunti. Infine, viene preso come risultato ottimale quello per cui l'indice è minimo. Si riporta la sezione di codice che esegue tale algoritmo:

```
1 maxl = 1; %lambda massimo
2      c = 1;
3
4      for gamma00 = -180:1:180
5
6          gamma = gamma00 + velocita.gamma;
7          Ttw = Tool_Funz(gamma,dati);
```

```
8
9           W = Kin_Inv_6assi(Antropomorfo,Ttw,dati);
10
11
12          % Rapporti di velocità di giunto
13          lambda1= max(abs(W(:,1)))/dqlim(1);
14          lambda2= max(abs(W(:,2)))/dqlim(2);
15          lambda3= max(abs(W(:,3)))/dqlim(3);
16          lambda4= max(abs(W(:,4)))/dqlim(4);
17          lambda5= max(abs(W(:,5)))/dqlim(5);
18          lambda6= max(abs(W(:,6)))/dqlim(6);
19
20          Lambda = [lambda1 lambda2 lambda3 lambda4 lambda5 ...
                lambda6]' ;
21
22          maxLambda(c) = max(Lambda);
23
24          if maxLambda(c)< maxl
25              maxl = maxLambda(c);
26          end
27
28          c = c+1;
29
30      end
31
32      gamma0ott = find(maxLambda == maxl,1)-181;
33
34      display(gamma0ott)
35      display(maxl)
36      gamma = gamma0ott+velocita.gamma;
```

La funzione `Tool Funz` calcola la matrice di trasformazione del tool T_{tw} a partire dalle caratteristiche del percorso e dalla correzione γ della flangia. La funzione `Kin Inv 6assi`, invece, calcola la cinematica inversa di posizione e poi le velocità dei giunti, come derivata numerica delle rotazioni di giunto. Il vettore γ_{ott} ottimale

viene, dunque, calcolato sommando il γ_0 al γ di primo tentativo:

$$\gamma_{ott} = \gamma_0 + \gamma$$

4.4 Criterio semi-analitico con *fmincon*

Il secondo criterio comprende i calcoli previsti nel primo, ma con l'aggiunta della ricerca di un minimo locale per via numerica tramite la function $fmincon$ di MatLAB, che itera un vettore contenente tempi e valori di gamma in modo tale da ottenere una legge di n polinomi cubici ottimale nel senso della minimizzazione dell'indice I.
Scelto il numero n di polinomi raccordati per la legge oraria di γ, il vettore X delle variabili di ottimizzazione, viene costruito come segue [10]:

$$X = [[\gamma_0\ \gamma_1\ \dots\ \gamma_{n+1}]\ [t_1\ \dots\ t_{n-1}]]$$

Inoltre, poiché si tratta di un problema di ottimizzazione vincolata nonlineare, risolto con l'algoritmo iterativo detto "interior-point", vanno definiti un vettore iniziale X_0, costruito a partire dal γ ottenuto al primo criterio, e due vettori X_{min} e X_{max} che rappresentano i vincoli inferiore e superiore, rispettivamente. In questo caso il dominio di ottimizzazione è abbastanza ristretto attorno alla soluzione X_0, in modo tale da contenere i tempi di calcolo.
Si riporta anche in questo caso un estratto del codice relativo a questo secondo criterio di ottimizzazione:

```
1  n = 5; % numero di polinomi
2
3  T = dati.T;
4  t = dati.t;
5  dt = dati.dt;
6  par_t = linspace(1/n,1-1/n,n-1);
7  tt = linspace(T/n,T*(1-1/n),n-1);
8  par_gam = [gamma(1)];
9  for i = 1 : n-1
10     g = find(abs(t-tt(i))<=dt);
11     par_gam = [par_gam,gamma(g(1))];
12 end
```

```
13 par_gam = [par_gam,gamma(end)];
14
15 dg=ceil(max(gamma)-min(gamma))/20;
16 dt=T/20;
17
18 %condizioni Nuove
19 Xmin = [par_gam-dg, par_t-dt]; %vincolo inferiore
20 Xmax = [par_gam+dg, par_t+dt]; %vincolo superiore
21 X0=[par_gam, par_t]; %vettore iniziale
22
23 %Aumento il numero massimo di funzioni valutate e il numero ...
      massimo delle
24 %iterazioni rispetto ai valori di default
25     options = optimoptions(@fmincon,'MaxFunctionEvaluations', ...
          100000,'MaxIterations',10000);
26
27 [Xott,FVAL,EXITFLAG,OUTPUT] = fmincon(@(x) ...
      Antropomorfo.indiceFUNZ3(x,dati,dqlim), ...
      X0,[],[],[],[],Xmin,Xmax,[],options);
28
29 % Parametri ottimali
30 ParamOTT = Xott;
31 save('ParamOTT.mat','ParamOTT')
```

La funzione obiettivo da minimizzare è la funzione `indiceFUNZ3` che, dato il vettore X in input, calcola la funzione interpolante di n polinomi cubici, la cinematica inversa, quindi il relativo indice I dei rapporti di velocità dei giunti del manipolatore.

4.4.1 Interpolazione polinomiale

La funzione interpolante sopra citata è una funzione interpolante di N punti con $n = N - 1$ polinomi raccordati. In questa tesi, si sono utilizzati polinomi di 3° grado e di 5° grado. Tale funzione costituirà la legge oraria di γ.
Una pianificazione ai giunti su N con $N - 1$ polinomi di grado poco elevato (ad esmpio, 3° o 5° grado, appunto), definiti a tratti, è preferibile rispetto ad un unico

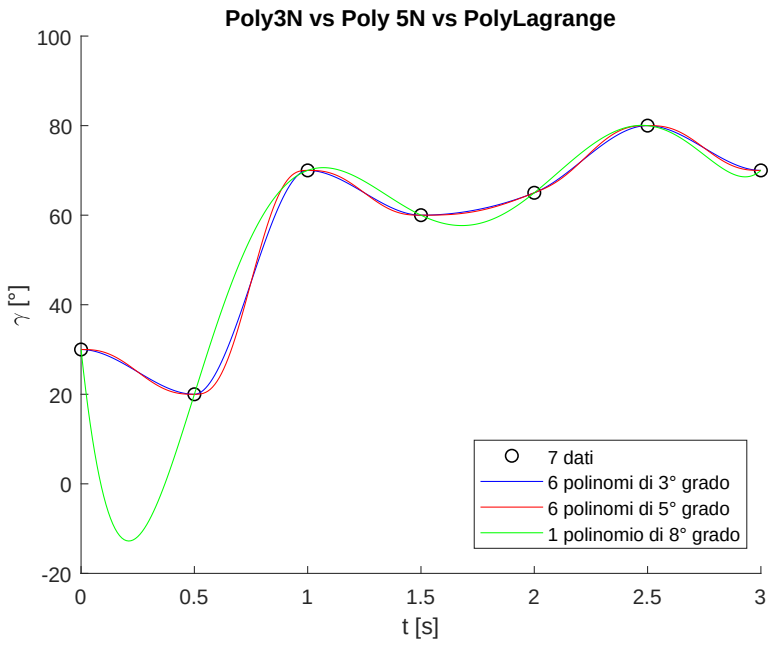


Figura 4.6: Confronto tra l'interpolazione ottenuta con polinomi di 3° grado, polinomi di 5° grado e un unico polinomio di grado $N+1$ (polinomio di Lagrange)

polinomio interpolante di grado $N+1$, poiché quest'ultimo produrrebbe oscillazioni e quindi accelerazioni troppo elevate.
In Figura 4.6 è visibile il confronto tra diversi tipi di interpolazione.

4.4.2 Polinomi di 3° grado

Per l'interpolazione con n polinomi cubici si parte dall'interpolazione punto-punto con un polinomio cubico, imponendo per ciascuna coppia di punti la continuità nelle posizioni e nelle velocità agli estremi. La legge polinomiale di terzo grado e le sue derivate hanno le seguenti espressioni:

$$\gamma(t) = a_3\left(\frac{t}{T}\right)^3 + a_2\left(\frac{t}{T}\right)^2 + a_1\left(\frac{t}{T}\right) + a_0 \tag{4.1}$$

$$\dot{\gamma}(t) = \frac{1}{T}\left[3a_3\left(\frac{t}{T}\right)^2 + 2a_2\left(\frac{t}{T}\right) + a_1\right] \tag{4.2}$$

$$\ddot{\gamma}(t) = \frac{1}{T^2}\left[6a_3\left(\frac{t}{T}\right) + 2a_2\right] \tag{4.3}$$

Si scrivono ora le quattro condizioni (vincoli), ossia la posizione iniziale γ_0 e la posizione finale γ_f , nonché le velocità iniziale $\dot{\gamma}_0$ e finale $\dot{\gamma}_f$. Così facendo si ottiene un sistema di equazioni algebriche che permette di calcolare i parametri a_i della legge oraria.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & \frac{1}{T} & 0 \\ \frac{3}{T} & \frac{2}{T} & \frac{1}{T} & 0 \end{bmatrix} \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} \gamma_0 \\ \gamma_f \\ \dot{\gamma}_0 \\ \dot{\gamma}_f \end{pmatrix} \tag{4.4}$$

$$\begin{cases} a_3 = 2(\gamma_0 - \gamma_f) + T(\dot{\gamma}_0 + \dot{\gamma}_f) \\ a_2 = 3(\gamma_f - \gamma_0) - T(2\dot{\gamma}_0 + \dot{\gamma}_f) \\ a_1 = T\dot{\gamma}_0 \\ a_0 = \gamma_0 \end{cases} \tag{4.5}$$

I coefficienti a_3, a_2, a_1, a_0 elencati sopra andranno dunque inseriti nelle espressioni di $\gamma(t)$, $\dot{\gamma}(t)$, $\ddot{\gamma}(t)$.

A questo punto, si procede a calcolare i polinomi cubici tra ogni coppia di punti, garantendo la continuità di posizione e velocità. Le posizioni sono assegnate e corrispondono a quelle prelevate dal vettore iniziale X_0 presentato all'inizio del paragrafo; le velocità, invece, sono calcolate con un certo criterio nei punti intermedi [5]. Il criterio qui utilizzato sfrutta in una prima fase una interpolazione lineare degli N punti e usa la pendenza dei vari segmenti di retta come valore iniziale per il calcolo della velocità:

$$v_{i,k} = \frac{q_{i,k} - q_{i,k-1}}{t_k - t_{k-1}} \tag{4.6}$$

Tranne che nel punto iniziale e in quello finale, in cui è nulla, la velocità del giunto viene calcolata prendendo la pendenza media dei segmenti adiacenti, a meno che le due pendenze non abbiano segni differenti:

$$\dot{q}_{i,1} = \dot{q}_{i,N} = 0 \tag{4.7}$$

$$\dot{q}_{i,k} = \frac{v_{i,k} + v_{i,k+1}}{2} \tag{4.8}$$

Il codice utilizzato è raccolto nella seguente funzione:

```
1  function X = poly3N_mod(x,dati)
2
3  % Viene assegnata all'angolo gamma una legge oraria ...
       POLINOMIALE MISTA
4  % definita a tratti attraverso N POLINOMI di GRADO 3.
5
6  % INPUT: x: vettore di (2*n) elementi:
7  %           n+1 angoli per gamma, n-1 tempi intermedi
8  %        dati: struttura dei dati relativi al percorso
9  % OUTPUT: X: vettore dei punti (posizioni) risultanti
10 %            dall'interpolazione,
11 %         delle velocità risultanti dall'interpolazione,
12 %         vettore delle accelerazioni risultanti ...
       dall'interpolazione
13 % ----------------------------------
14
15 %% Interpolazione
16 %variabili
17 Ttot = dati.T; % tempo totale in secondi
18 dt = dati.dt; % step temporale in secondi
19 t = dati.t; % vettore complessivo dei tempi
20 lx = length(x);
21 n = lx/2;  % numero di polinomi
22 N = n+1; % numero di punti
23
24 %ordino con un sort il vettore perchè per come sono state ...
       definite
25 %Xmax e Xmin potrei avere degli scambi
26 x = [x(1:(n+1)) sort(x((n+2):(2*n)))];
27
28 %estraggo il vettore delle posizioni gamma: gamma0, ..., gammaN
29  for i = 1 : (n+1)
30      gamma(i) = x(i);
31  end
32
```

```
33  %creo il vettore tj: T0,Tint1,Tint2,.....,T
34  tj(1)=0;
35  tj(n+1)=Ttot;
36  for i = (n+2): (lx)
37      tj(i-(n)) = x(i)*Ttot;
38  end
39
40 %% Interpolazione
41
42 g = []; %vettore dei punti (posizioni) risultanti ...
       dall'interpolazione
43 dg = []; %vettore delle velocità risultanti dall'interpolazione
44 ddg = []; %vettore delle accelerazioni risultanti ...
       dall'interpolazione
45
46 % Velocità calcolate nei punti intermedi
47
48 deltat = Ttot/(N-1); %durata di un singolo tratto
49 tt = 0:dt:deltat; %vettore del tempo nel tratto j-esimo
50 % plot(tj,gamma,'ok');
51
52 gamma_dot = zeros(1,N);
53 gamma_dotdot = zeros(1,N);
54 v = zeros(1,N-1);
55 a = zeros(1,N-1);
56 for k = 2:N
57     v(k) = (gamma(k)-gamma(k-1))/deltat; % vettore delle pendenze
58     a(k) = (gamma_dot(k)-gamma_dot(k-1))/deltat;
59 end
60
61 for i = 2:N-1
62     if sign(v(i))*sign(v(i+1))==1
63         gamma_dot(i) = (v(i)+v(i+1))/2; % vettore delle ...
              velocità nei vari punti
64     else
65         gamma_dot(i) = 0; % pendenze discordi
```

```
66      end
67
68      if sign(a(i))*sign(a(i+1))==1
69          gamma_dotdot(i) = (a(i)+a(i+1))/2; % vettore delle ...
                velocità nei vari punti
70      else
71          gamma_dotdot(i) = 0; % pendenze discordi
72      end
73  end
74
75  % Interpolazione con N-1 polinomi cubici
76  for j = 1:N-1
77      if j==N-1
78          [gamma_j,dgamma_j,ddgamma_j] = ...
                poly3(gamma(j),gamma(j+1), ...
                gamma_dot(j),gamma_dot(j+1),deltat,[tt tt(end)+dt]);
79      else
80          [gamma_j,dgamma_j,ddgamma_j] = ...
                poly3(gamma(j),gamma(j+1), ...
                gamma_dot(j),gamma_dot(j+1),deltat,tt);
81      end
82      g = [g gamma_j];
83      dg = [dg dgamma_j];
84      ddg = [ddg ddgamma_j];
85  end
86
87  X = [g; dg; ddg];
88
89  end
```

dove il vettore chiamato X alla fine del codice è in realtà il vettore $Xott$ risultato dell'ottimizzazione. La funzione `poly3N` esegue l'interpolazione punto-punto con un polinomio cubico secondo le formule illustrate sopra.

4.4.3 Polinomi di 5° grado

Usando un polinomio di 3° grado è possibile imporre dei vincoli sulla posizione e sulla velocità agli estremi di ciascun tratto. Tuttavia, esso produce delle discontinuità nell'accelerazione e, di conseguenza, degli impulsi di *jerk*. Un gradino di accelerazione porta inevitabilmente a dei fenomeni vibratori, i quali vanno possibilmente evitati.
Una possibile soluzione è quella di usare dei polinomi di 5° grado, in modo da poter imporre anche la continuità dell'accelerazione.

$$\gamma(t) = a_5\left(\frac{t}{T}\right)^5 + a_4\left(\frac{t}{T}\right)^4 + a_3\left(\frac{t}{T}\right)^3 + a_2\left(\frac{t}{T}\right)^2 + a_1\left(\frac{t}{T}\right) + a_0 \tag{4.9}$$

$$\dot{\gamma}(t) = \frac{1}{T}\left[5a_5\left(\frac{t}{T}\right)^4 + 4a_4\left(\frac{t}{T}\right)^3 + 3a_3\left(\frac{t}{T}\right)^2 + 2a_2\left(\frac{t}{T}\right) + a_1\right] \tag{4.10}$$

$$\ddot{\gamma}(t) = \frac{1}{T^2}\left[20a_5\left(\frac{t}{T}\right)^3 + 12a_4\left(\frac{t}{T}\right)^2 + 6a_3\left(\frac{t}{T}\right) + 2a_2\right] \tag{4.11}$$

Si scrivono ora le sei condizioni (vincoli), ossia la posizione iniziale γ_0 e la posizione finale γ_f , le velocità iniziale $\dot{\gamma}_0$ e finale $\dot{\gamma}_f$ e le accelerazioni iniziale $\ddot{\gamma}_0$ e finale $\ddot{\gamma}_f$. Così facendo si ottiene un sistema di equazioni algebriche che permette di calcolare i parametri a_i della legge oraria.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & \frac{1}{T} & 0 \\ \frac{5}{T} & \frac{4}{T} & \frac{3}{T} & \frac{2}{T} & \frac{1}{T} & 0 \\ 0 & 0 & 0 & \frac{2}{T^2} & 0 & 0 \\ \frac{20}{T^2} & \frac{12}{T^2} & \frac{6}{T^2} & \frac{2}{T^2} & 0 & 0 \end{bmatrix} \begin{pmatrix} a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} \gamma_0 \\ \gamma_f \\ \dot{\gamma}_0 \\ \dot{\gamma}_f \\ \ddot{\gamma}_0 \\ \ddot{\gamma}_f \end{pmatrix} \tag{4.12}$$

$$\begin{cases} a_5 = 6(\gamma_f - \gamma_0) - 3T(\dot{\gamma}_0 - \dot{\gamma}_f) - \frac{T^2}{2}\left(\ddot{\gamma}_0 - \ddot{\gamma}_f\right) \\ a_4 = 15(\gamma_0 - \gamma_f) + T(8\dot{\gamma}_0 + 7\dot{\gamma}_f) + \frac{T^2}{2}\left(3\ddot{\gamma}_0 - 2\ddot{\gamma}_f\right) \\ a_3 = 10(\gamma_f - \gamma_0) - T(6\dot{\gamma}_0 - 4\dot{\gamma}_f) - \frac{T^2}{2}\left(3\ddot{\gamma}_0 - \ddot{\gamma}_f\right) \\ a_2 = \frac{T^2}{2}\ddot{\gamma}_0 \\ a_1 = T\dot{\gamma}_0 \\ a_0 = \gamma_0 \end{cases} \tag{4.13}$$

I coefficienti a_5, a_4, a_3, a_2, a_1, a_0 elencati sopra andranno dunque inseriti nelle espressioni di $\gamma(t)$, $\dot{\gamma}(t)$, $\ddot{\gamma}(t)$.
A questo punto, si procede a calcolare i polinomi di 5° grado tra ogni coppia di punti, garantendo la continuità di posizione, velocità e accelerazione. Come nel caso dei polinomi cubici, le posizioni sono assegnate e corrispondono a quelle prelevate dal vettore iniziale X_0 presentato all'inizio del paragrafo; le velocità nei punti intermedi, invece, sono calcolate anche qui col criterio dell'interpolazione lineare e delle pendenze medie, così come le accelerazioni.
Per non appesantire la trattazione, non si riporta il codice utilizzato per il calcolo dei polinomi raccordati, poiché è molto simile a quello utilizzato nel caso dei polinomi cubici, ad eccezione del fatto che richiama la funzione `poly5N` al posto di `poly3N`. La funzione `poly5N`, infatti, esegue l'interpolazione punto-punto con un polinomio di 5° grado secondo le formule illustrate sopra.

4.5 Criterio numerico con *fmincon*

Ricerca di un minimo locale per via numerica tramite la function *fmincon*, che itera un vettore contenente tempi e valori di gamma in modo tale da ottenere la legge di due polinomi cubici o di 5° grado ottimale nel senso della minimizzazione dell'indice dei rapporti di velocità.
Le function di interpolazione polinomiale utilizzate in questo caso sono le medesime impiegate nel paragrafo 4.4.

Il grafico prodotto dal terzo script, dunque, è il seguente:

Nel quarto ed ultimo script, vengono caricati tutti i risultati dell'ottimizzazione e viene applicata la legge oraria di γ all'utensile del manipolatore. Viene perciò ricalcolata un'ultima volta la cinematica inversa di posizione e di velocità, nelle condizioni in cui l'angolo γ vari nel tempo, e vengono calcolati i rapporti di velocità di tutti i giunti. Il massimo di questi, chiamato L_a sarà il fattore di scalatura del tempo di movimento, così come segue:

$$T_{new} = L_a \cdot T$$

dove $L_a = max(\lambda_i)$.
Le rotazioni ottenute e le informazioni necessarie alla scalatura vengono salvate in due appositi file (`jointRot.mat` e `LambdaOttimo.mat`), in modo tale da poter

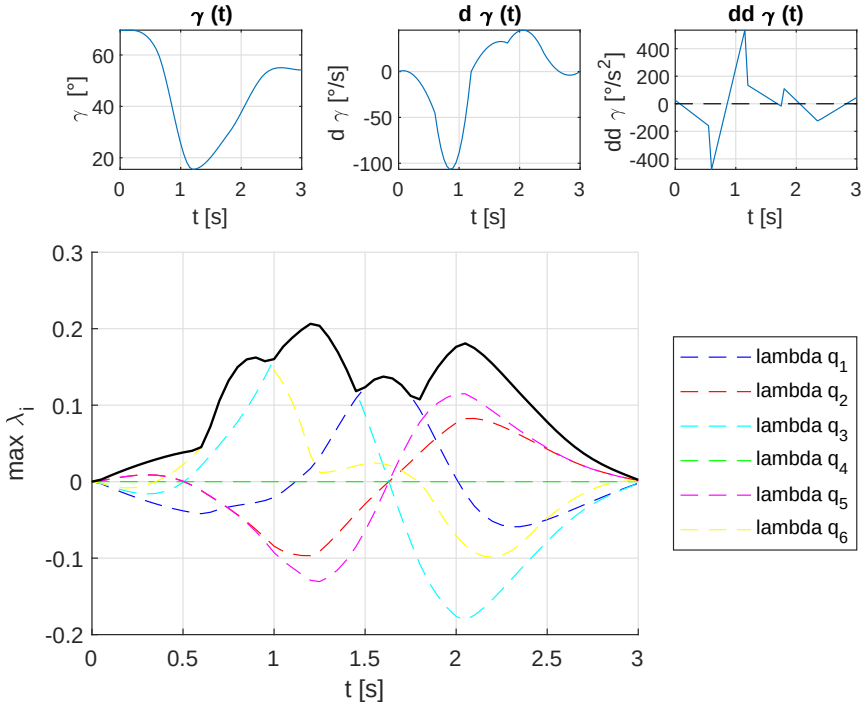


Figura 4.7: Posizione γ, velocità $\dot{\gamma}$ e accelerazione $\ddot{\gamma}$ (in alto); rapporti di velocità λ_i di tutti i giunti e valore massimo

essere inviati successivamente al robot tramite una connessione TCP. Approfondiremo quest'ultimo argomento nel Capitolo 5.
I grafici prodotti in questo quarto script sono i seguenti:

A questo punto, viene eseguita la simulazione ottima del movimento, tramite cinematica diretta, visualizzando il robot in una sequenza di fotogrammi come quello in Figura 4.11.

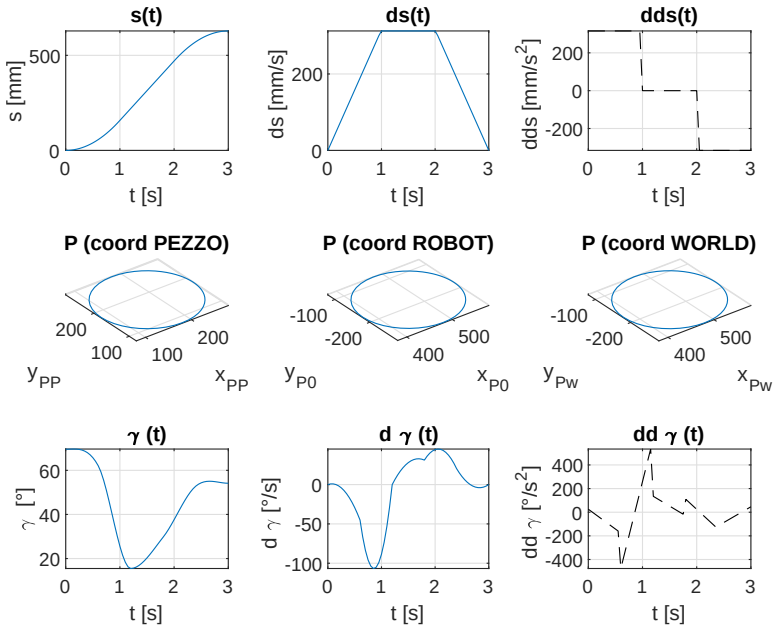


Figura 4.8: Ascissa curvilinea della traiettoria, traiettoria cartesiana nei diversi sistemi di riferimento e legge oraria per γ

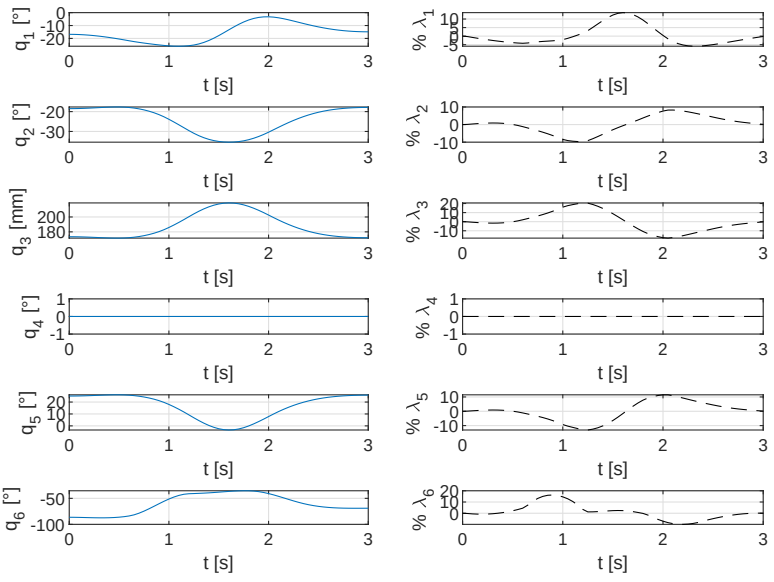


Figura 4.9: Leggi orarie e rotazioni dei giunti

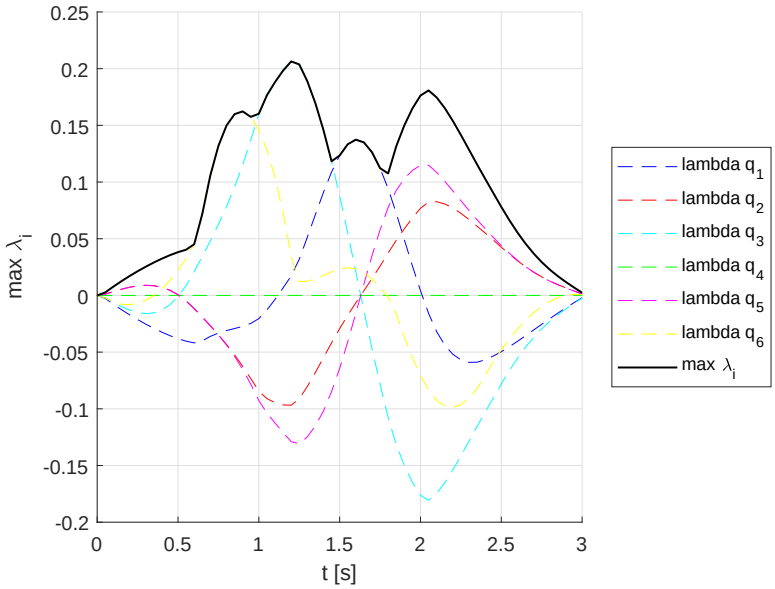


Figura 4.10: Rapporti di velocità post-ottimizzazione

4.6 Risultati delle simulazioni

Si riportano, ora, i risultati delle simulazioni effettuate su diverse geometrie, con diversi offset dell'utensile, con differenti criteri di ottimizzazione e su diversi piani di scrittura.
Per ogni piano di scrittura, si sono simulate una traiettoria circolare ed una rettilinea; inoltre, nel caso di piano orizzontale, si sono eseguite alcune traiettorie a mano libera. Nelle tabelle dei risultati, per ciascun criterio di ottimizzazione sono riportati i tempi di movimento scalati a seguito dell'ottimizzazione (t *scalato*) e i tempi impiegati per l'esecuzione dell'algoritmo di ottimizzazione (t *comput*).
I criteri di ottimizzazione, sono così denominati:

- 1° criterio = γ costante
- 2° criterio = γ_J (primo tentativo, 2° script) + γ_0 (criterio analitico, 3° script)
- 3° criterio = γ_J + γ_0 + *fmincon* (criterio semi-analitico, 3° script)
- 4° criterio = *fmincon* (criterio numerico, 3° script)

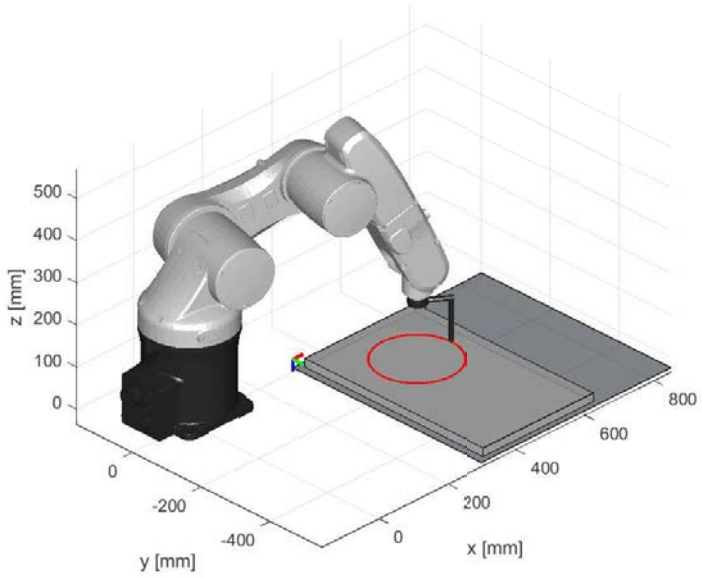


Figura 4.11: Fotogramma della simulazione ottima

4.6.1 Piano obliquo

Partendo dal **piano obliquo** (origine della terna del piano in [300 100 300]), inclinato di 30° rispetto all'orizzontale, si sono trovati i risultati riportati nelle Tabelle 4.6.1 e 4.6.1 e nelle Figure 4.12 e 4.13.

piano obliquo - cerchio							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.7323 s	0.7294 s	8.5266 s	0.6857 s	17.1888 s	0.5683 s	22.8946 s
70 mm	0.7079 s	0.7440 s	8.5774 s	0.7894 s	18.8304 s	0.5879 s	22.8369 s

Tabella 4.2: Tempi ottimi per traiettoria circolare con centro in $C = [200\ 200\ 0\ 1]$ (nel sistema di riferimento del piano di scrittura), raggio $R = 100\ mm$ e tempo massimo di movimento $T = 3.00\ s$ su piano inclinato di 30° rispetto all'orizzontale

piano obliquo - segmento							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.1561 s	0.1267 s	8.1757 s	0.1250 s	22.5931 s	0.1308 s	13.3789 s
70 mm	0.1473 s	0.2147 s	8.1579 s	0.1986 s	24.8637 s	0.1350 s	13.8876 s

Tabella 4.3: Tempi ottimi per traiettoria rettilinea tra $P_{in} = [150\ 450\ 0\ 1]'$ e $P_{fin} = [250\ 200\ 0\ 1]'$ (nel sistema di riferimento del piano di scrittura) e tempo massimo di movimento $T = 3.00\ s$ su piano inclinato di 30° rispetto all'orizzontale

4.6.2 Piano verticale

Nel caso di **piano verticale** (origine della terna del piano in [550 100 200]) si sono trovati i risultati riportati nelle Tabelle 4.6.2 e 4.6.2 e nelle Figure 4.14 e 4.15.

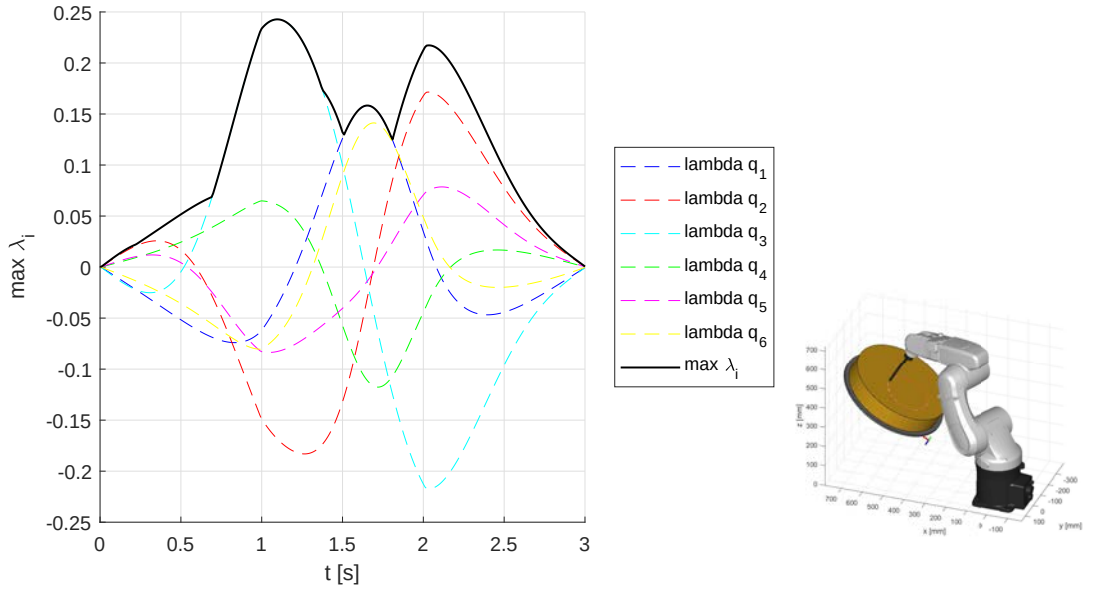


Figura 4.12: Rapporti di velocità di giunto nel caso di traiettoria circolare su piano obliquo con 4° criterio, quello più vantaggioso (22.4% con $d_y = 100mm$ e 17.0% con $d_y = 70mm$ in meno rispetto al 1° criterio)

piano verticale - cerchio							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.6200 s	0.5571 s	8.8343 s	0.5461 s	16.3458 s	-	-
70 mm	0.6030 s	0.6337 s	8.6103 s	0.6016 s	17.0902 s	-	-

Tabella 4.4: Tempi ottimi per traiettoria circolare con centro in $C = [150\ 150\ 0\ 1]$ (nel sistema di riferimento del piano di scrittura), raggio $R = 100\ mm$ e tempo massimo di movimento $T = 3.00\ s$ su piano inclinato di 30° rispetto all'orizzontale

4.6.3 Piano orizzontale

Nel caso di **piano orizzontale** (origine della terna del piano in [300 0 -5.5]) si sono trovati i risultati riportati nelle Tabelle 4.6.3 e 4.6.3 e nelle Figure 4.16 e 4.17.

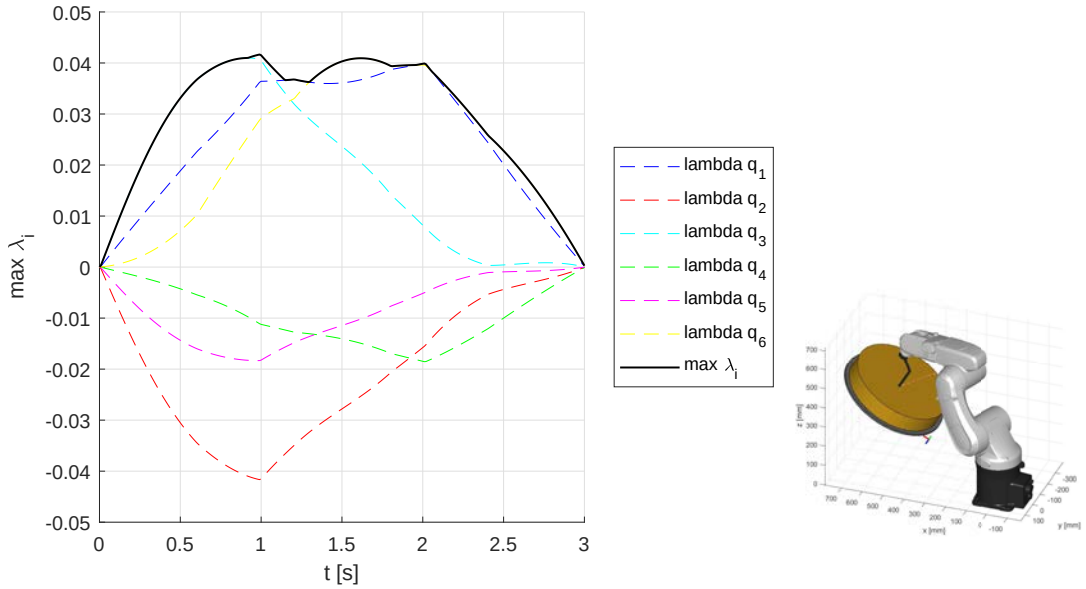


Figura 4.13: Rapporti di velocità di giunto nel caso di traiettoria rettilinea su piano obliquo con 3° e 4° criterio, quelli più vantaggiosi (19.9% con $d_y = 100mm$ e 8.4% con $d_y = 70mm$ in meno rispetto al 1° criterio, rispettivamente)

piano verticale - segmento							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.2585 s	0.2158 s	8.6495 s	0.2023 s	29.2235 s	-	-
70 mm	0.2585 s	0.2368 s	8.6701 s	0.2239 s	37.4911 s	-	-

Tabella 4.5: Tempi ottimi per traiettoria rettilinea tra $P_{in} = [50\ 400\ 0\ 1]'$ e $P_{fin} = [200\ 50\ 0\ 1]'$ (nel sistema di riferimento del piano di scrittura) e tempo massimo di movimento $T = 3.00\ s$ su piano inclinato di 30° rispetto all'orizzontale

Nel caso di **piano orizzontale**, inoltre, si sono simulati anche movimenti lungo traiettorie *free-shape*, tracciate a mano libera. In particolare sono stati disegnati una spirale ed una stella. I risultati ottenuti sono riportati nelle Tabelle 4.6.3 e 4.6.3 e nelle Figure 4.18 e 4.20.

Andando ad analizzare brevemente i risultati ottenuti, si vede che il 4° criterio

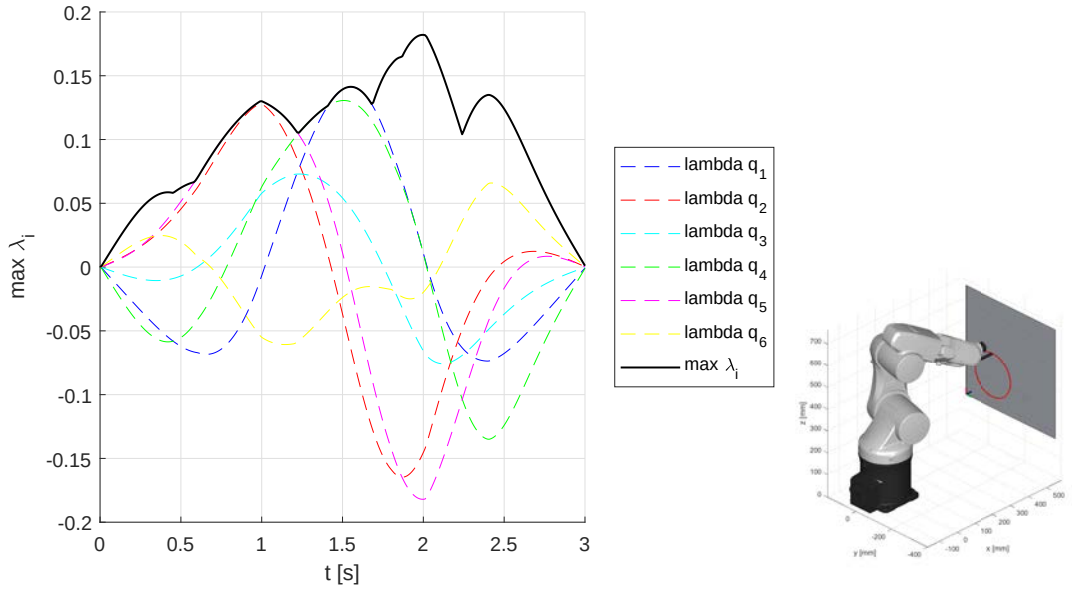


Figura 4.14: Rapporti di velocità di giunto nel caso di traiettoria circolare su piano verticale con 3° criterio, quello più vantaggioso (11.9% con $d_y = 100mm$ e 0.2% con $d_y = 70mm$ in meno rispetto al 1° criterio)

piano orizzontale - cerchio							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.6602 s	0.5217 s	12.8166 s	0.5282 s	17.1240 s	0.4602 s	19.3240 s
70 mm	0.6717 s	0.5949 s	8.7777 s	0.5845 s	18.7650 s	0.5178 s	14.0576 s

Tabella 4.6: Tempi ottimi per traiettoria circolare con centro in $C = [150\ 150\ 0\ 1]$ (nel sistema di riferimento del piano di scrittura), raggio $R = 100\ mm$ e tempo massimo di movimento $T = 3.00\ s$ su piano orizzontale

risulta spesso vincente, poiché è quello che lascia più libertà all'ottimizzatore, dal momento che i "nodi" della legge polinomiale possono essere spostati all'interno di un dominio rettangolare molto ampio: la posizione γ può variare tra -180° e 180°, l'istante t tra 0 e T. Tuttavia, nel caso in cui il piano sia verticale, il 4° criterio "impazzisce" e non porta a risultati accettabili per l'ottimizzazione (ecco

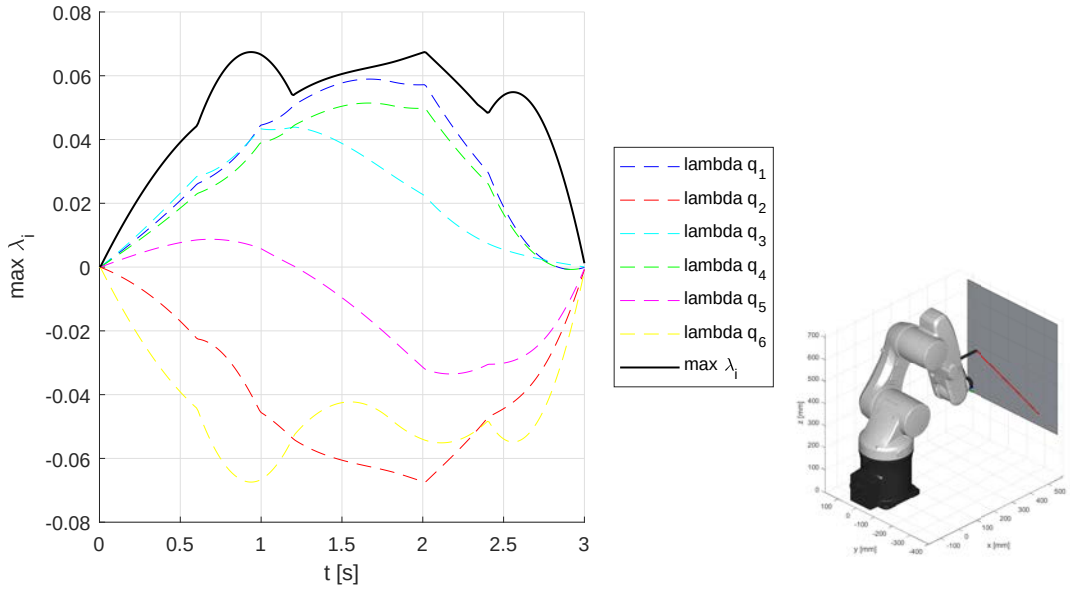


Figura 4.15: Rapporti di velocità di giunto nel caso di traiettoria rettilinea su piano verticale con 3° criterio, quello più vantaggioso (21.7% con $d_y = 100mm$ e 13.4% con $d_y = 70mm$ in meno rispetto al 1° criterio)

piano orizzontale - segmento							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.2208 s	0.1997 s	9.4510 s	0.1933 s	19.1876 s	0.1528 s	30.7240 s
70 mm	0.2151 s	0.2020 s	8.8531 s	0.2004 s	16.0869 s	0.1639 s	19.5000 s

Tabella 4.7: Tempi ottimi per traiettoria rettilinea tra $P_{in} = [150\ 450\ 0\ 1]'$ e $P_{fin} = [250\ 200\ 0\ 1]'$ (nel sistema di riferimento del piano di scrittura) e tempo massimo di movimento $T = 3.00\ s$ su piano orizzontale

perché le caselle corrispondenti in tabella sono vuote). In quel caso, dunque, risulta conveniente il 3° criterio.

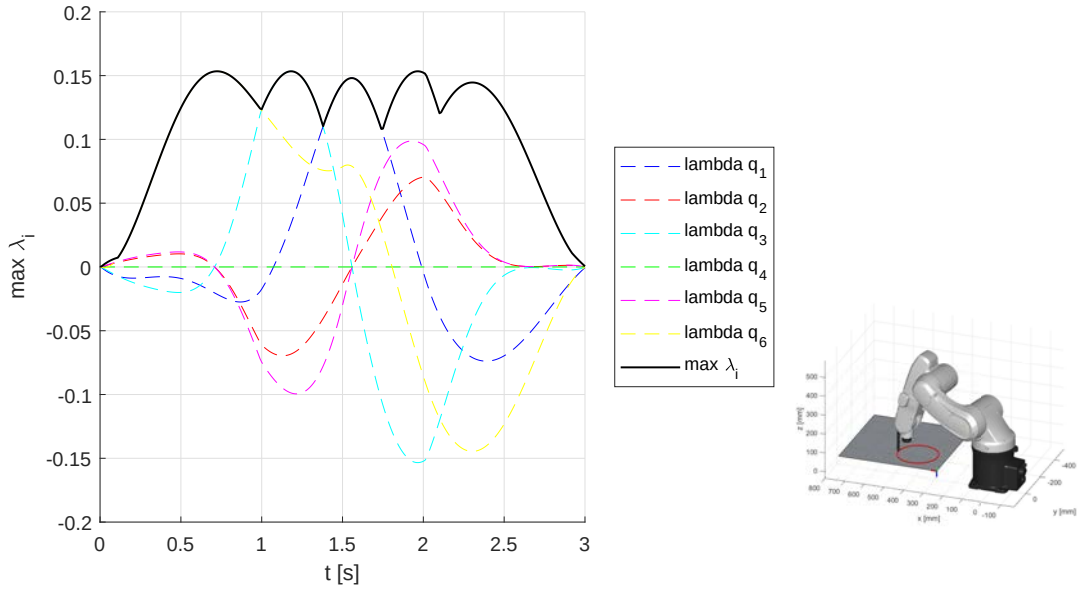


Figura 4.16: Rapporti di velocità di giunto nel caso di traiettoria circolare su piano orizzontale con 4° criterio, quello più vantaggioso (30.3% con $d_y = 100mm$ e 22.9% con $d_y = 70mm$ in meno rispetto al 1° criterio)

piano orizzontale - spirale							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	1.3463 s	0.9690 s	8.8741 s	1.0501 s	15.3367 s	1.0539 s	10.1524 s
70 mm	1.3152 s	0.8200 s	8.6930 s	0.9317 s	18.7733 s	0.9565 s	11.0364 s

Tabella 4.8: Tempi ottimi per traiettoria free-shape a forma di spirale e tempo massimo di movimento $T = 3.00\ s$ su piano orizzontale

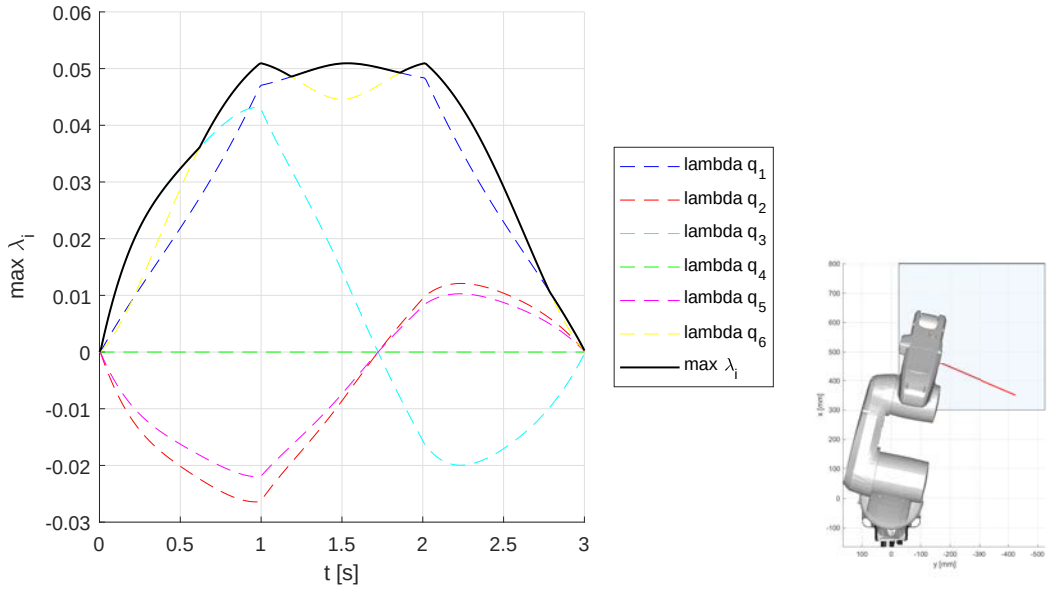


Figura 4.17: Rapporti di velocità di giunto nel caso di traiettoria rettilinea su piano orizzontale con 4° criterio, quello più vantaggioso (30.8% con $d_y = 100mm$ e 23.8% con $d_y = 70mm$ in meno rispetto al 1° criterio)

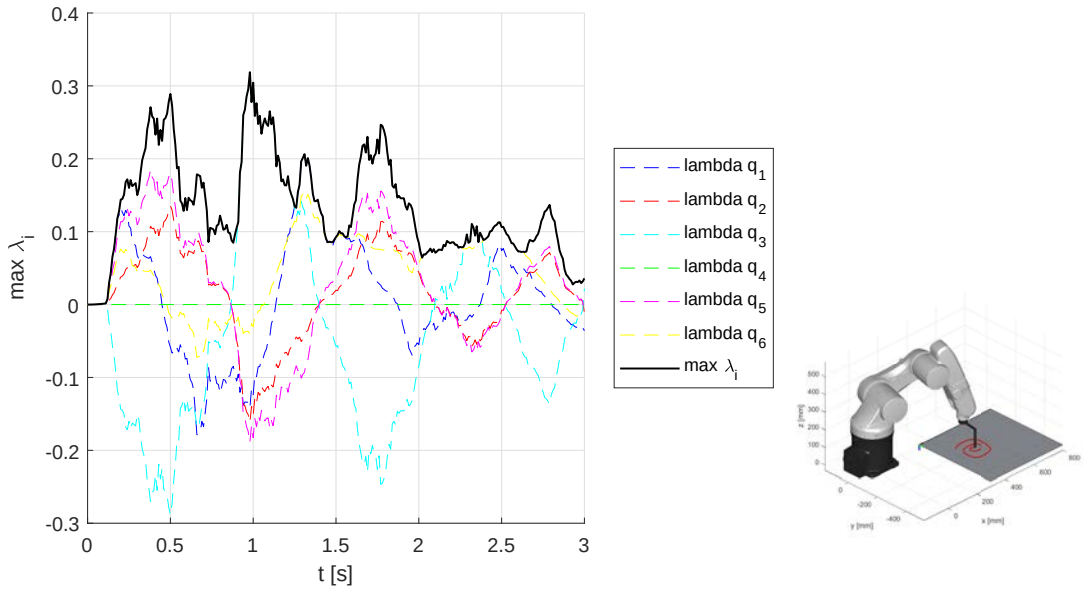


Figura 4.18: Rapporti di velocità di giunto nel caso di traiettoria free-shape a forma di spirale su piano orizzontale con 2° criterio, quello più vantaggioso (28.0% con $d_y = 100mm$ e 37.7% con $d_y = 70mm$ in meno rispetto al 1° criterio)

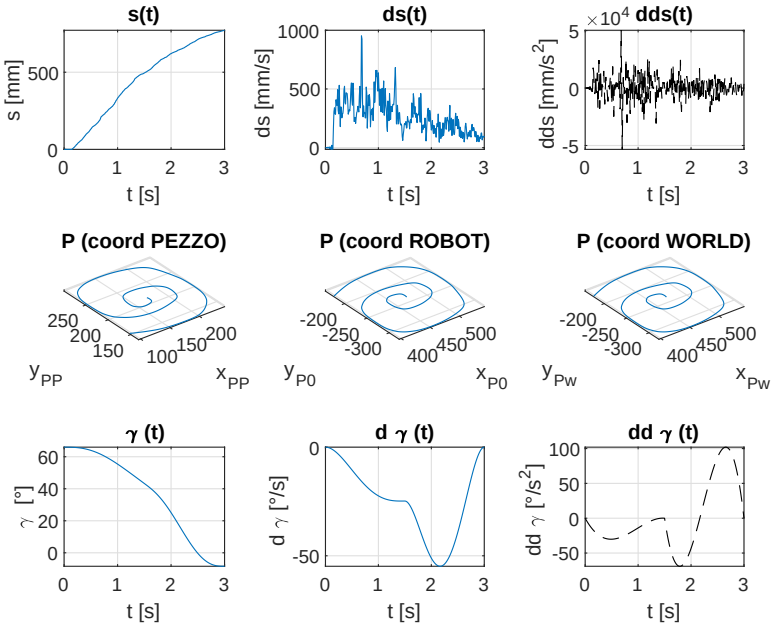


Figura 4.19: Leggi orarie ottenute a seguito dell'ottimizzazione nel caso di traiettoria free-shape a forma di spirale su piano orizzontale con 2° criterio

piano orizzontale - spirale							
	γ fisso	γ variabile					
	1° criterio	2° criterio		3° criterio		4° criterio	
d_y	t scalato	t scalato	t comput	t scalato	t comput	t scalato	t comput
100 mm	0.8265 s	0.8443 s	8.6700 s	0.9455 s	15.0403 s	0.6402 s	10.2937 s
70 mm	0.8379 s	0.7761 s	8.7585 s	0.9256 s	24.3629 s	0.6293 s	15.4459 s

Tabella 4.9: Tempi ottimi per traiettoria free-shape a forma di stella e tempo massimo di movimento $T = 3.00\ s$ su piano orizzontale

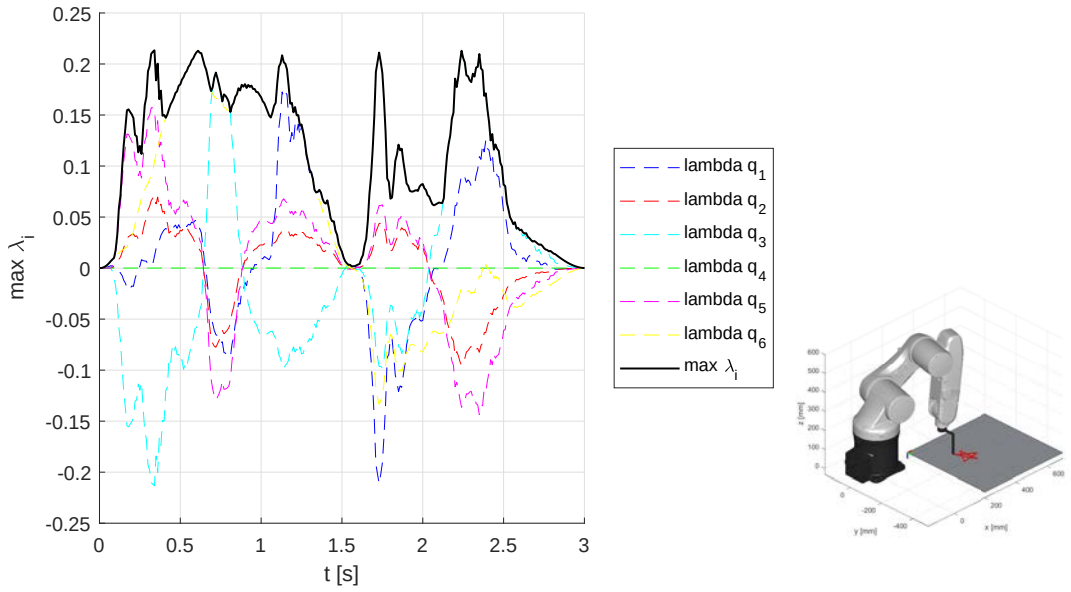


Figura 4.20: Rapporti di velocità di giunto nel caso di traiettoria free-shape a forma di stella su piano orizzontale con 4° criterio, quello più vantaggioso (22.5% con $d_y = 100mm$ e 24.9% con $d_y = 70mm$ in meno rispetto al 1° criterio)

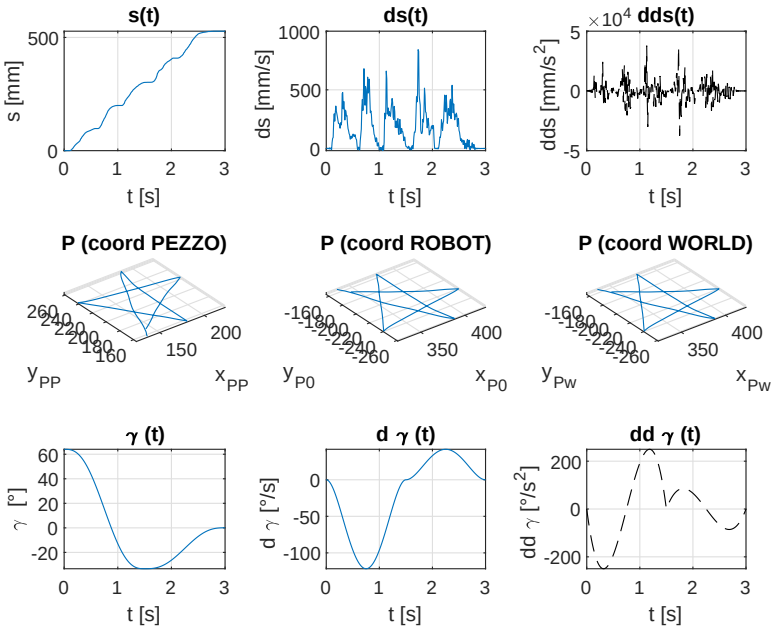


Figura 4.21: Leggi orarie ottenute a seguito dell'ottimizzazione nel caso di traiettoria free-shape a forma di stella su piano orizzontale con 4° criterio

Capitolo 5

Fase sperimentale

Nell'ultima parte di questo lavoro di tesi, dopo aver eseguito la pianificazione cartesiana (Capitolo 3) ed implementato gli algoritmi di ottimizzazione in modo tale da sfruttare a proprio favore la ridondanza del manipolatore, così da poter scalare il tempo di movimento (Capitolo 4), si è svolta l'attività sperimentale nel Laboratorio di Robotica e Automazione dell'Università di Padova. Tale attività ha compreso la creazione di una connessione tra MatLAB ed ACE V+ e la successiva movimentazione del robot, secondo le posizioni ottenute durante le simulazioni.

5.1 Omron ACE V+

Il software Adept Automation Control Environment (ACE) è un pacchetto software che contiene una collezione di strumenti per la configurazione, la programmazione, il controllo e il monitoraggio dell'equipaggiamento Adept in una cella di lavoro. Questi strumenti sono accessibili dall'interfaccia grafica utente (GUI) di ACE, la quale fornisce un ambiente semplice ed intuitivo.
In particolare, V+ è il linguaggio di programmazione utilizzato da ACE per controllare i movimenti del robot. All'interno del codice è possibile definire locazioni nello spazio, eseguire movimenti ai giunti o nello spazio operativo, definire e leggere segnali digitali I/O, definire cicli di lavoro con operazioni di *pick-and-place*, leggere dati da file o dispositivi esterni (un disco di memoria o una connessione). È possibile sia collegarsi al *controller* di un robot fisico, sia lavorare in "emulation mode", ossia collegarsi ad un *controller* virtuale, che controllerà esclusivamente

Figura 5.1: Cella robotizzata con robot Adept Viper s650 e piano di scrittura utilizzati per la fase sperimentale di questa tesi

il movimento del modello all'interno del *3D virtual display*.

5.2 Connessione TCP/IP

Si vogliono spedire al robot le rotazioni di giunto ottenute a seguito della simulazione ottima eseguita su MatLAB. Per fare ciò si sceglie di creare una connessione TCP/IP tra MatLAB ed ACE.

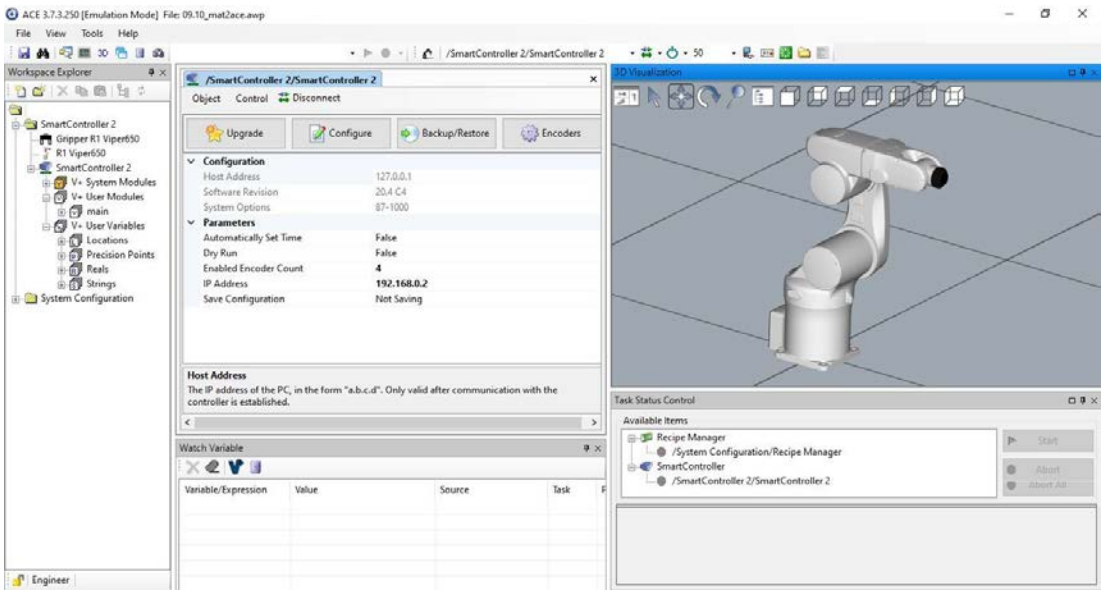


Figura 5.2: Interfaccia grafica del programma Omron ACE: Workspace Explorer (a sinistra), Editor window (al centro) e 3D virtual display (a destra)

Il *Transmission Control Protocol* (TCP) si riferisce a un importante protocollo Internet che è responsabile della trasmissione o del trasferimento di pacchetti di dati in rete e su Internet. Può essere classificato come protocollo orientato alla connessione, ovvero prima di poter trasmettere dati deve stabilire la comunicazione, negoziando una connessione tra mittente e destinatario, che rimane attiva anche in assenza di scambio di dati e viene esplicitamente chiusa quando non più necessaria. Esso quindi possiede le funzionalità per creare, mantenere e chiudere una connessione. Funziona sul livello di trasporto come protocollo nell'interconnessione di sistemi aperti e viene utilizzato per costruire una connessione stabile tra computer remoti confermando l'invio del messaggio su reti di supporto e Internet [11].

Tutti i computer connessi mediante TCP/IP, possiedono un loro indirizzo unico, detto indirizzo IP (*Internet Protocol*) che si compone di quattro cifre decimali, ognuna delle quali è compresa tra 0 e 255. Un esempio di indirizzo è 212.24.124.151.
Affinché un computer possa comunicare con un altro mediante TCP/IP è necessario che su entrambi i computer sia disponibile un *socket* utilizzabile, una sorta di *handle* TCP/IP. L'utilizzo di un socket comporta l'apertura di una porta TCP,

ovvero la messa in ascolto su un computer *server* che attende che i *client* si colleghino. Anche sul computer client ci sarà una porta TCP aperta, ma non sarà posta in ascolto come sul server: tramite questa porta il computer invierà i dati relativi al tentativo di connessione.
Finché la connessione non è stabilita il server assume una posizione passiva di ascolto. Sarà compito del client effettuare tutte le operazioni di preparazione, la scelta dell'indirizzo e della *porta* corretta e l'effettuazione della chiamata.
L'attivazione di un collegamento comporta l'allineamento del computer client al computer server. Sarà il programma server che stabilirà su quale porta si dovrà comunicare. Il client, dal lato suo, potrà aprire il suo socket su qualunque porta. Ciò che conta, infatti, è il numero di porta su cui i dati confluiscono.
Per esempio un programma server può aprire in ascolto la sua porta locale 1500. Il programma client, affinché la connessione sia stabilita, dovrà aprire un socket libero sul computer in cui il programma viene eseguito (locale) e mediante tale socket dovrà inviare una richiesta di connessione alla porta remota 1500 dell'indirizzo del computer scelto [12].

I dati inviati tramite connessioni TCP sono divisi in segmenti numerati indipendentemente. Ogni segmento contiene la destinazione di origine e la sezione dati che viene inserita in un'intestazione. Il protocollo di controllo della trasmissione è quello responsabile del riordino dei segmenti nella giusta sequenza al suo arrivo nel terminale ricevente. Il TCP è responsabile di tenere traccia di questi segmenti, mentre l'IP gestisce l'effettiva consegna dei dati. Il TCP include anche il controllo degli errori, che assicura che ogni pacchetto sia consegnato come richiesto.

Dal momento che il TCP deve controllare gli errori e verificare la ricezione dei pacchetti, si instaurerà una certa latenza, che è proprio il punto di debolezza di questo protocollo, nel caso di applicazioni in real-time.
Esistono anche altri protocolli di rete, come l'UDP (*User Datagram Protocol*), che garantisce prestazioni superiori in quanto a velocità di invio dei pacchetti, ma risulta meno affidabile, poiché non verifica l'effettivo arrivo dei segmenti, né il loro ordine in sequenza in arrivo.

In questo caso specifico, MatLAB fa da client, mentre il controller del robot fa da server. Il trasferimento di dati tra il computer su cui è aperto MatLAB ed

il controller avviene attraverso un cavo ethernet.
I codici MatLAB ed ACE utilizzati per la creazione della connessione, la spedizione e la ricezione dei pacchetti (TCPconnection_TICTOC.m, TestTimer_Read.m, TestTimer_Send.m e mat2ACE.awp, main.awp) sono riportati in Appendice, insieme alle altre funzioni richiamate al loro interno.

5.3 Temporizzazione del movimento

L'invio delle posizioni da MatLAB ad ACE non può essere realizzato a piacere, ma deve anzi rispettare una certa temporizzazione imposta, in modo che il movimento sia concluso in un tempo quanto più possibile vicino a quello utilizzato nelle simulazioni.
Per fare ciò, si sono adottate diverse strategie, utilizzando in combinazione i codici di MatLAB ed ACE nominati sopra e riportati in Appendice:

1. `mat2ACE` + `TCPconnection_TICTOC.m` (mode = 1)
2. `mat2ACE` + `TCPconnection_TICTOC.m` (mode = 2; eseguito in una prima istanza MatLAB) + `TestTimer_Read.m` (eseguito in una seconda istanza MatLAB)
3. `main` + `TestTimer_Read.m` (eseguito in una prima istanza MatLAB) + `TestTimer_Send.m` (mode = 1; eseguito in una seconda istanza MatLAB)

Il metodo (1) stabilisce una singola connessione tra MatLAB ed ACE, gestendo la temporizzazione da MatLAB con delle condizioni `if`, secondo le quali, se il manipolatore si trova in ritardo in una certa posizione, sarà autorizzato a saltare qualche posizione successiva, per poter recuperare il ritardo. Il contatore del tempo viene gestito con il costrutto `tic-toc`. La modalità *mode = 1* indica che le posizioni vengono inviate una alla volta ed il movimento del robot avviene subito dopo la ricezione di ciascuna.

Il metodo (2) stabilisce due connessioni simultanee tra MatLAB ed ACE, gestendo la temporizzazione da MatLAB con delle condizioni `if`, allo stesso modo del metodo (1). La modalità *mode = 2* indica che le posizioni vengono inviate tutte insieme ed il movimento del robot avviene solo dopo la ricezione dell'intero set. Il programma `mat2ACE` esegue in un task parallelo una routine di scrittura

delle posizioni correnti sulla seconda connessione; queste posizioni vengono lette da `TestTimer_Read.m`.

Il metodo (3) stabilisce due connessioni simultanee tra MatLAB ed ACE, gestendo la temporizzazione da MatLAB, ma questa volta con un *timer*, anziché con il costrutto `tic-toc`. Un timer è un oggetto utilizzato per programmare l'esecuzione dei comandi MatLAB una o più volte. Se si programma il timer in modo che venga eseguito più volte, è possibile definire il tempo tra le esecuzioni e come gestire i conflitti di accodamento. L'oggetto timer utilizza le *callback function* per eseguire i comandi. Le callback function eseguono il codice durante alcuni eventi. Per l'oggetto timer, è possibile specificare la callback function come handle di funzione o come vettore di caratteri. Se la callback function è un vettore di caratteri, MatLAB la valuta come codice eseguibile. L'oggetto timer supporta le funzioni di callback quando un timer si avvia (*StartFcn*), esegue (*TimerFcn*), si arresta (*StopFcn*) o incontra un errore (*ErrorFcn*) [13].

5.4 Prove sperimentali e risultati

Sono state eseguite diverse prove sperimentali ponendo l'attenzione su diversi aspetti: il confronto tra il γ fisso e il γ ottimizzato, la qualità della curva disegnata, l'effetto della frequenza di micro-interpolazione del controller.

5.4.1 Confronto tra γ variabile e γ fisso

In primo luogo si è voluto verificare il vantaggio dato dal movimento con γ variabile secondo l'ottimizzazione rispetto al caso con γ fisso. Per fare ciò, si è rinunciato alla temporizzazione, lasciando quindi lo stesso tempo T a disposizione per entrambi i casi. Si è utilizzato il metodo (2) illustrato al Paragrafo 5.3, sostituendo alla temporizzazione gestita da ACE un semplice aggiornamento dell'indice i del ciclo con l'istruzione $i = i + 1$.

La prova è stata eseguita su una traiettoria circolare avente raggio R = 110 mm e centro C = [160 200 0 1] nel sistema di riferimento del piano, offset utensile d_y = 100 mm, T = 3 s, dt = 0.05 s, piano orizzontale; il controller era settato

ad una frequenza di 500 Hz. Il caso con γ variabile, utilizzando il 4° criterio di ottimizzazione, è stato eseguito in 3.0900 secondi; il caso con γ fisso, invece, ha impiegato 3.5230 secondi. La discrepanza tra i tempi effettivi di movimento nei

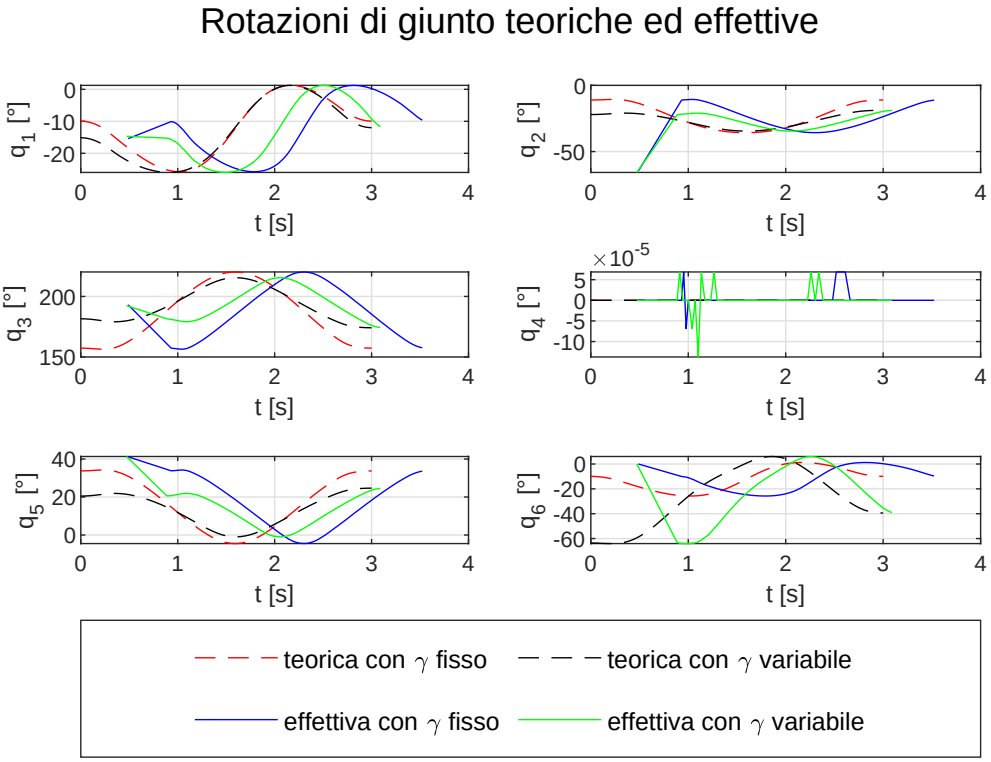


Figura 5.3: Confronto tra le posizioni teoriche ed effettive nel caso di γ variabile e γ fisso con traiettoria circolare

due casi è dovuta al fatto che nel movimento con γ variabile le velocità dei giunti sono distribuite in maniera più uniforme, con dei rapporti di velocità simili tra un giunto e l'altro. Ne consegue, perciò, un movimento più rapido, anche senza il controllo sulla temporizzazione.

Infatti, andando ad analizzare le Figure 5.3,5.4,5.5,5.6, si osserva che nel caso del γ variabile (linea verde continua), il giunto 6 viene sfruttato maggiormente, andando a ridurre contemporaneamente rotazioni e velocità degli altri giunti: ne risulta, di conseguenza, un andamento del λ massimo (linea nera continua in Figura 5.6) più costante nel'intervallo di tempo del movimento.

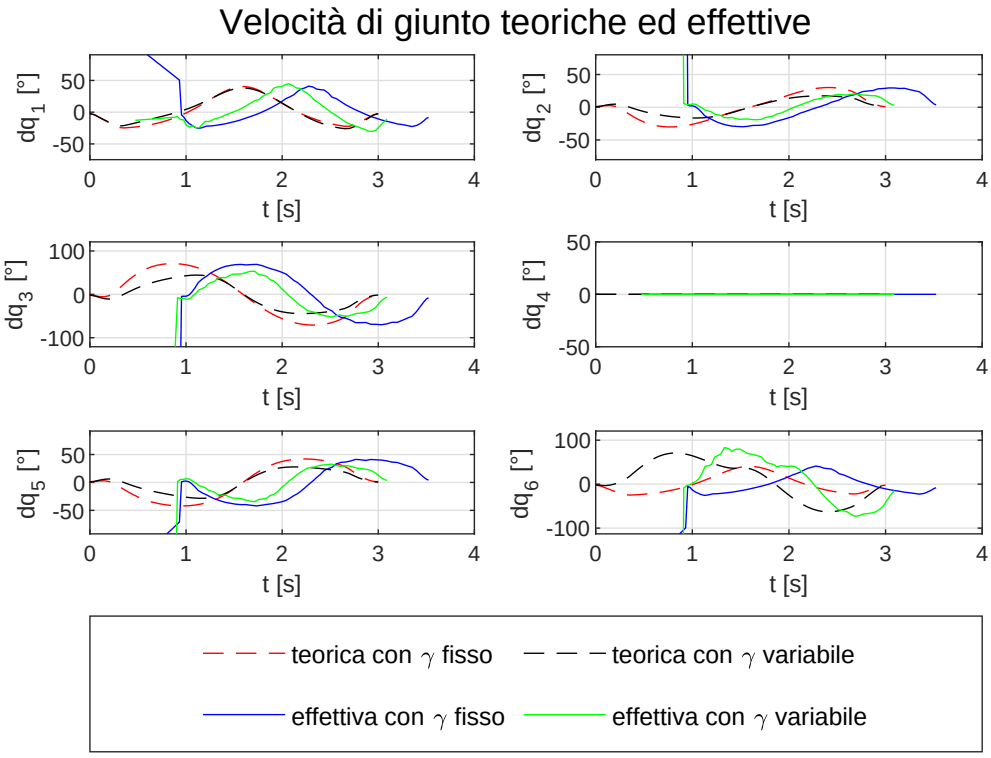


Figura 5.4: Confronto tra le velocità teoriche ed effettive nel caso di γ variabile e γ fisso con traiettoria circolare

5.4.2 Qualità della curva disegnata

In un secondo set di prove, si è posta l'attenzione sulla qualità dei tratti di curva disegnati, variando il metodo di comunicazione tra MatLAB (vedi Paragrafo 5.3) ed ACE e la scalatura del tempo di movimento.
La qualità del movimento del robot può essere valutata soggettivamente, con un controllo visivo, oppure può essere valutata oggettivamente sulla base del numero di posizioni inviate ed eseguite rispetto al numero complessivo di posizioni previste dalla pianificazione.
Queste prove sono state eseguite su una traiettoria circolare avente raggio R = 110 mm e centro C = [160 200 0 1] nel sistema di riferimento del piano, offset utensile d_y = 70 mm, T = 3 s, dt = 0.05 s; frequenza del controller pari a 500 Hz.
I risultati ottenuti sono riportati in Figura 5.7.
Il metodo (2) è quello che garantisce la miglior qualità del movimento, poiché salta meno posizioni, pur rispettando la temporizzazione.

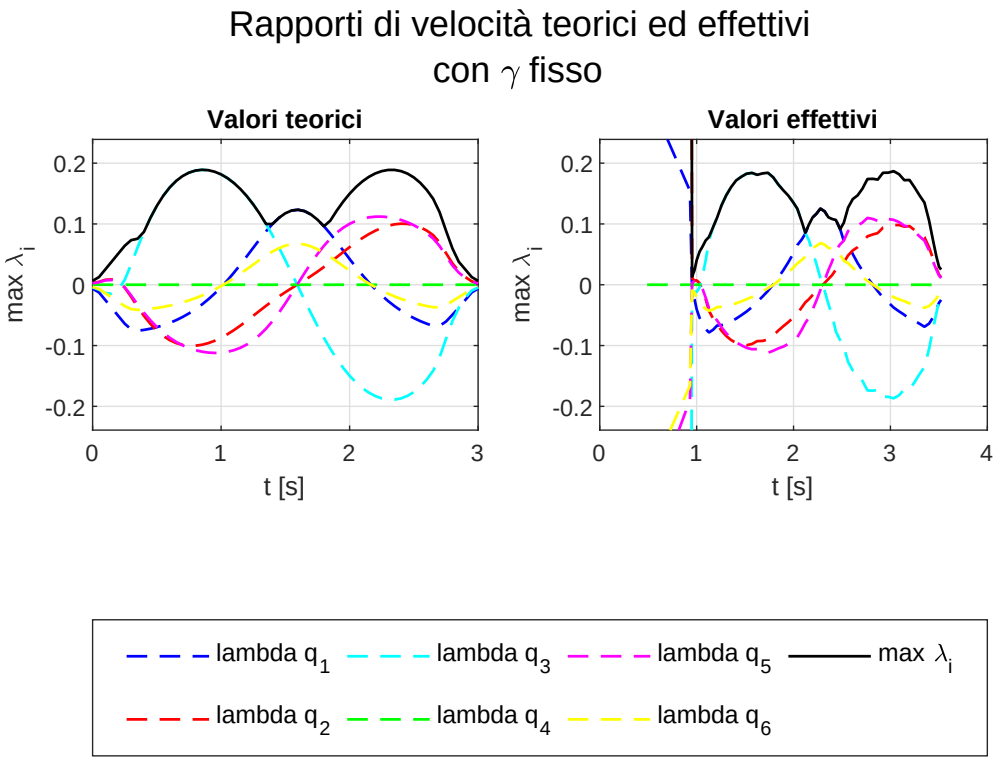


Figura 5.5: Confronto tra i rapporti di velocità teorici ed effettivi nel caso di γ fisso con traiettoria circolare

Tutti e tre i metodi riescono a rispettare sempre la temporizzazione assegnata; tuttavia, se il tempo disponibile è troppo basso (ad esempio, perché è stato scalato), possono essere tagliate parecchie posizioni, portando ad un risultato molto distante da quello desiderato.

5.4.3 Effetto della frequenza di micro-interpolazione del controller

Da ultimo, si sono effettuate diverse prove su una traiettoria circolare, variando la frequenza di micro-interpolazione del controller del robot (125 Hz - 250 Hz - 500 Hz). È stato utilizzato il metodo (1) per la comunicazione tra MatLAB ed ACE ed il tempo di movimento, pari a $T = 3$ secondi, non è stato scalato.
Non si osservano differenze significative tra i risultati ottenuti a diverse frequenze, sebbene intuitivamente si potrebbe pensare che il caso con frequenza più elevata

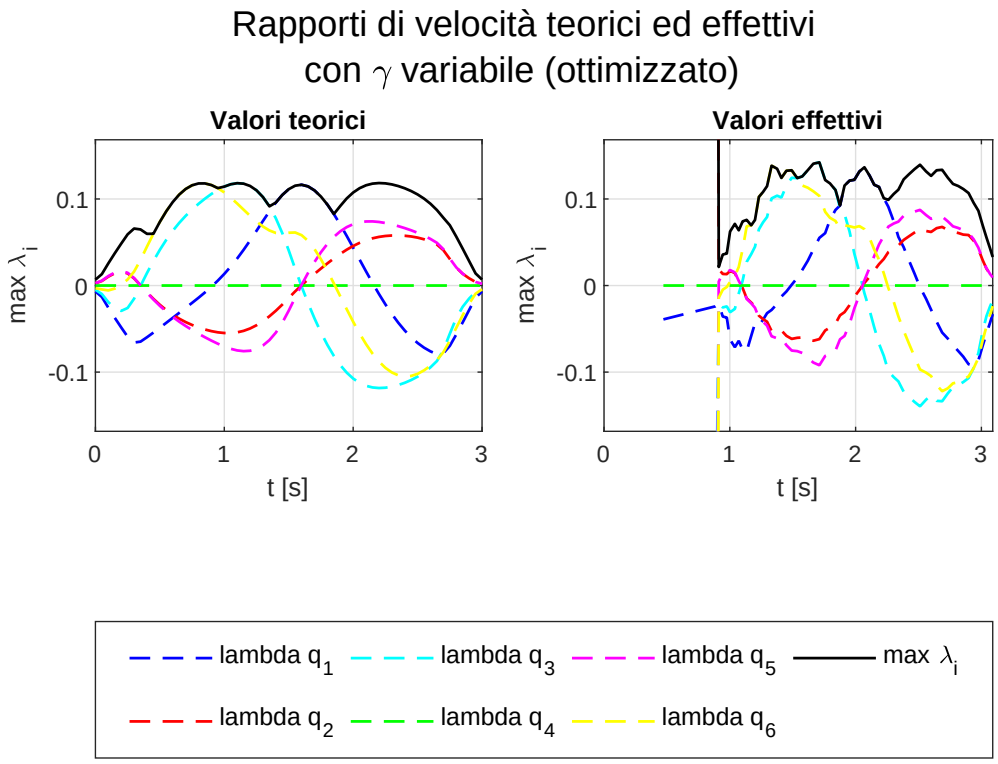


Figura 5.6: Confronto tra i rapporti di velocità teorici ed effettivi nel caso di γ variabile con traiettoria circolare

possa dare un movimento più aderente a quello teorico.

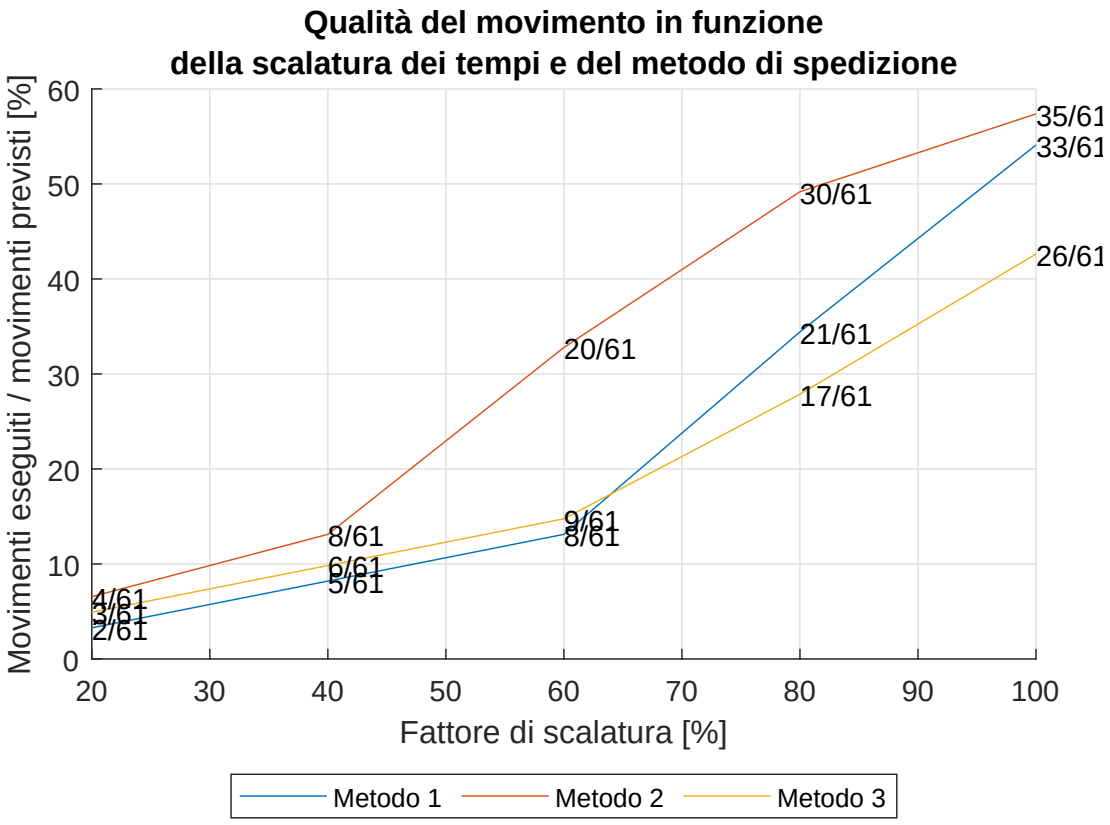


Figura 5.7: Posizioni eseguite su posizioni previste al variare del fattore di scalatura dei tempi e del metodo di comunicazione tra MatLAB ed ACE

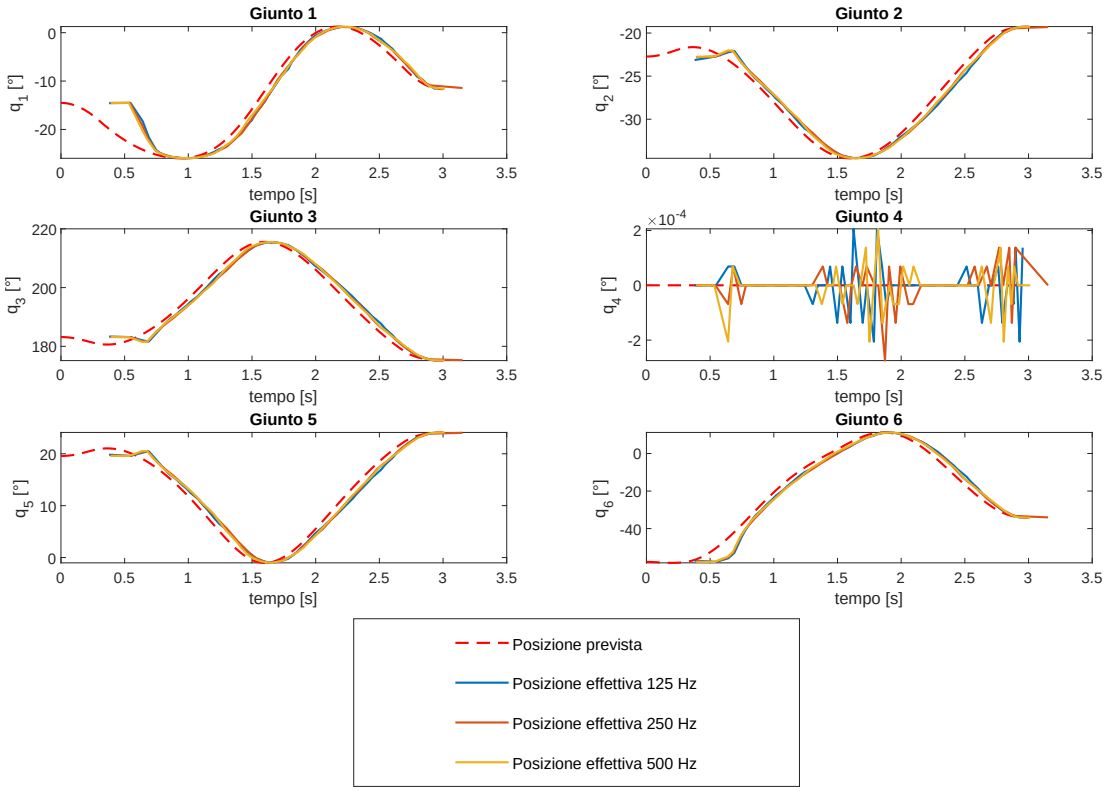


Figura 5.8: Posizioni teoriche ed effettive al variare della frequenza di micro-interpolazione del controller

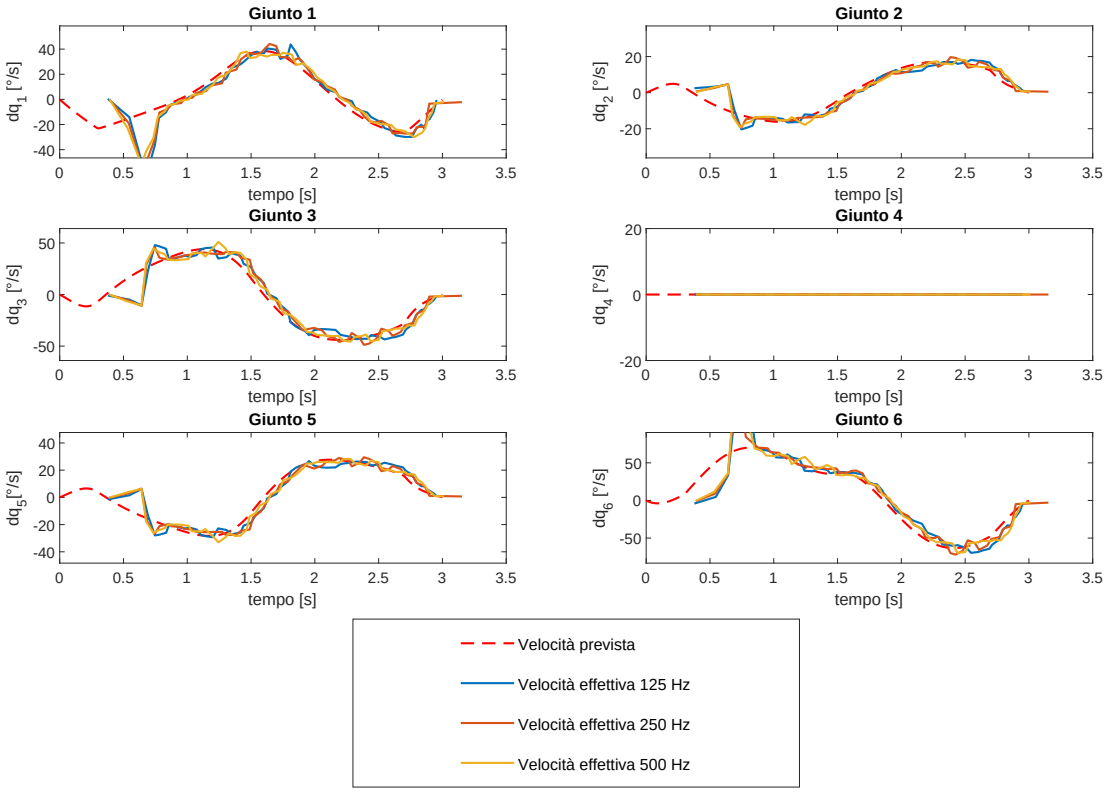


Figura 5.9: Velocità teoriche ed effettive al variare della frequenza di micro-interpolazione del controller

Conclusioni

Gli obiettivi che ci si era posti all'inizio di questa tesi sono stati raggiunti.
Dopo aver lavorato sulla pianificazione delle traiettorie nello spazio operativo, sia per geometrie regolari, che per curve disegnate a mano libera, si è eseguita una prima simulazione di movimento, avvalendosi della classe KINpro per la gestione della cinematica.
Si sono poi implementati diversi algoritmi di ottimizzazione dei movimenti da eseguire.
In una prima parte, esclusivamente simulativa, si sono eseguite diverse simulazioni di movimento su varie traiettorie tracciate su piani si scrittura a diversa inclinazione. Si sono dunque confrontati i risultati ottenuti applicando i diversi criteri di ottimizzazione, che sono risultati vantaggiosi rispetto al caso non ottimizzato.
La seconda parte del lavoro, invece, si è svolta in laboratorio con il robot Adept Viper s650. Si sono dapprima sviluppati una serie di codici per la creazione di una connessione TCP/IP tra MatLAB ed ACE V+ finalizzata alla spedizione delle posizioni ottimizzate e alla successiva movimentazione del robot. Successivamente, si sono eseguite diverse prove sperimentali per testare la qualità del movimento eseguito e la capacità di rispettare la temporizzazione imposta.
I risultati sperimentali sono stati accettabili, ma non pienamente soddisfacenti rispetto alle simulazioni in MatLAB, poiché queste non tenevano conto della dinamica del manipolatore reale, ma esclusivamente della sua caratterizzazione cinematica. Ne conseguono, dunque, dei limiti sulla scalatura dei tempi, probabilmente imputabili anche alla latenza della connessione TCP.

Uno dei possibili sviluppi di questo lavoro può sicuramente riguardare il tipo di connessione, che potrebbe essere sostituita con una connessione UDP, teoricamente più veloce rispetto al protocollo TCP.

Inoltre, si potrebbero svolgere delle prove sperimentali di scrittura su piani inclinati, che qui sono state solo simulate virtualmente, per mancanza di tempo.
In aggiunta, si può migliorare il sistema di acquisizione delle traiettorie free-shape, ad esempio impiegando una tavoletta grafica, oppure implementare un sistema di importazione delle traiettorie da un software di modellazione geometrica 2D o 3D.
Infine, si potrebbe progettare e realizzare un dosatore per la deposizione di un fluido che sostituisca l'attuale supporto del pennarello e che abbia un sistema automatico di regolazione dell'offset, magari tramite azionamento pneumatico.

Appendice A

Jacobiana di un robot antropomorfo

A.1 Jacobiana riferita al centro polso

Si riporta l'espressione completa della Jacobiana riferita al centro polso di un robot antropomorfo a 6 assi, ottenuta grazie al *Symbolic Math Toolbox* di Matlab. Si ricordi che a_i e d_i sono i parametri geometrici di DH, mentre c_i e s_i sono le funzioni coseno e seno del generico angolo ϑ_i.

$$J_p = \begin{bmatrix} -\sin(q_1)\,(a_1 + a_3\,c_{23} - d_4\,s_{23} + a_2\,\cos(q_2)) & -\cos(q_1)\,(c_{23}\,d_4 + a_3\,s_{23} + a_2\,\sin(q_2)) & -\cos(q_1)\,(c_{23}\,d_4 + a_3\,s_{23}) \\ \cos(q_1)\,(a_1 + a_3\,c_{23} - d_4\,s_{23} + a_2\,\cos(q_2)) & -\sin(q_1)\,(c_{23}\,d_4 + a_3\,s_{23} + a_2\,\sin(q_2)) & -\sin(q_1)\,(c_{23}\,d_4 + a_3\,s_{23}) \\ 0 & d_4\,s_{23} - a_3\,c_{23} - a_2\,\cos(q_2) & d_4\,s_{23} - a_3\,c_{23} \\ 0 & -\sin(q_1) & -\sin(q_1) \\ 0 & \cos(q_1) & \cos(q_1) \\ 1 & 0 & 0 \end{bmatrix}$$

A.2 Jacobiana riferita al centro flangia

Si riporta l'espressione completa della Jacobiana riferita al centro della flangia di un robot antropomorfo a 6 assi, ottenuta grazie al *Symbolic Math Toolbox* di Matlab. Questa è la Jacobiana utilizzata in questo lavoro di tesi per il calcolo della cinematica inversa di velocità. Si ricordi che a_i e d_i sono i parametri geometrici

di DH, mentre c_i e s_i sono le funzioni coseno e seno del generico angolo ϑ_i.

$$J = \begin{pmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{pmatrix} \tag{A.1}$$

$$J_{11} = d_6\ (c_5\ (c_2\, s_1\, s_3 + c_3\, s_1\, s_2) - s_5\ (c_1\, s_4 - c_4\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3))) - a_1\, s_1 - a_3\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3)$$
$$+ d_4\ (c_2\, s_1\, s_3 + c_3\, s_1\, s_2) - a_2\, c_2\, s_1$$

$$J_{12} = -c_1\,(a_2\, s_2 + a_3\ (c_2\, s_3 + c_3\, s_2) + d_4\ (c_2\, c_3 - s_2\, s_3) + d_6\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2)))$$

$$J_{13} = -c_1\ (a_3\ (c_2\, s_3 + c_3\, s_2) + d_4\ (c_2\, c_3 - s_2\, s_3) + d_6\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2)))$$

$$J_{14} = -d_6\, s_5\ \left(c_4\, s_1\, {c_2}^2\, {c_3}^2 + c_4\, s_1\, {c_2}^2\, {s_3}^2 - c_1\, s_4\, c_2\, c_3 + c_4\, s_1\, {c_3}^2\, {s_2}^2 + c_4\, s_1\, {s_2}^2\, {s_3}^2 + c_1\, s_4\, s_2\, s_3\right)$$

$$J_{15} = -d_6\ (c_1\, c_4 + s_4\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3))\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2))$$
$$- d_6\, s_4\ (c_5\ (c_2\, s_1\, s_3 + c_3\, s_1\, s_2) - s_5\ (c_1\, s_4 - c_4\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3)))\ (c_2\, s_3 + c_3\, s_2)$$

$$J_{16} = 0$$

$$J_{21} = a_1\, c_1 - d_6\ (c_5\ (c_1\, c_2\, s_3 + c_1\, c_3\, s_2) + s_5\ (s_1\, s_4 + c_4\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3)))$$
$$+ a_3\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3) - d_4\ (c_1\, c_2\, s_3 + c_1\, c_3\, s_2) + a_2\, c_1\, c_2$$

$$J_{22} = -s_1\ (a_2\, s_2 + a_3\ (c_2\, s_3 + c_3\, s_2) + d_4\ (c_2\, c_3 - s_2\, s_3) + d_6\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2)))$$

$$J_{23} = -s_1\ (a_3\ (c_2\, s_3 + c_3\, s_2) + d_4\ (c_2\, c_3 - s_2\, s_3) + d_6\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2)))$$

$$J_{24} = d_6\, s_5\ \left(c_1\, c_4\, {c_2}^2\, {c_3}^2 + c_1\, c_4\, {c_2}^2\, {s_3}^2 + s_1\, s_4\, c_2\, c_3 + c_1\, c_4\, {c_3}^2\, {s_2}^2 + c_1\, c_4\, {s_2}^2\, {s_3}^2 - s_1\, s_4\, s_2\, s_3\right)$$

$$J_{25} = d_6\, s_4\ (c_5\ (c_1\, c_2\, s_3 + c_1\, c_3\, s_2) + s_5\ (s_1\, s_4 + c_4\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3)))\ (c_2\, s_3 + c_3\, s_2)$$
$$- d_6\ (c_5\ (c_2\, c_3 - s_2\, s_3) - c_4\, s_5\ (c_2\, s_3 + c_3\, s_2))\ (c_4\, s_1 - s_4\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3))$$

$$J_{26} = 0$$

$$J_{31} = 0$$

$$J_{32} = \left({c_1}^2 + {s_1}^2\right)\ (c_2\, d_4\, s_3 - a_3\, c_2\, c_3 - a_2\, c_2 + c_3\, d_4\, s_2 + a_3\, s_2\, s_3 + c_2\, c_5\, d_6\, s_3 + c_3\, c_5\, d_6\, s_2 + c_2\, c_3\, c_4\, d_6\, s_5 - c_4\, d_6\, s_2\, s_3\, s_5)$$

$$J_{33} = \left({c_1}^2 + {s_1}^2\right)\ (c_2\, d_4\, s_3 - a_3\, c_2\, c_3 + c_3\, d_4\, s_2 + a_3\, s_2\, s_3 + c_2\, c_5\, d_6\, s_3 + c_3\, c_5\, d_6\, s_2 + c_2\, c_3\, c_4\, d_6\, s_5 - c_4\, d_6\, s_2\, s_3\, s_5)$$

$$J_{34} = -d_6\, s_4\, s_5\ (c_2\, s_3 + c_3\, s_2)\ \left({c_1}^2 + {s_1}^2\right)$$

$$J_{35} = d_6\ \left({c_1}^2 + {s_1}^2\right)\ \left(c_2\, c_3\, {c_4}^2\, s_5 + c_2\, c_3\, {s_4}^2\, s_5 - {c_4}^2\, s_2\, s_3\, s_5 - s_2\, s_3\, {s_4}^2\, s_5 + c_2\, c_4\, c_5\, s_3 + c_3\, c_4\, c_5\, s_2\right)$$

$$J_{36} = 0$$

$$J_{41} = 0$$

$$J_{42} = -s_1$$

$$J_{43} = -s_1$$

$$J_{44} = -c_1\ (c_2\, s_3 + c_3\, s_2)$$

$$J_{45} = s_4\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3) - c_4\, s_1$$

$$J_{46} = -c_5\ (c_1\, c_2\, s_3 + c_1\, c_3\, s_2) - s_5\ (s_1\, s_4 + c_4\ (c_1\, c_2\, c_3 - c_1\, s_2\, s_3))$$

$$J_{51} = 0$$

$$J_{52} = c_1$$

$$J_{53} = c_1$$

$$J_{54} = -s_1\ (c_2\, s_3 + c_3\, s_2)$$

$$J_{55} = c_1\, c_4 + s_4\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3)$$

$$J_{56} = s_5\ (c_1\, s_4 - c_4\ (c_2\, c_3\, s_1 - s_1\, s_2\, s_3)) - c_5\ (c_2\, s_1\, s_3 + c_3\, s_1\, s_2)$$

$$J_{61} = 1$$

$$J_{62} = 0$$

$$J_{63} = 0$$

$$J_{64} = s_2\, s_3 - c_2\, c_3$$

$$J_{65} = -s_4\ (c_2\, s_3 + c_3\, s_2)$$

$$J_{66} = c_4\, s_5\ (c_2\, s_3 + c_3\, s_2) - c_5\ (c_2\, c_3 - s_2\, s_3)$$

Appendice B

Ulteriori codici per la comunicazione tra MatLAB ed ACE

B.1 Programmi MatLAB

Listing B.1: TCPconnection_TICTOC.m

```
% MATLAB connection creation using TCP/IP
clc
clear all
close all

host = '192.168.0.2'; % indirizzo IP per controller Viper
% host = '127.0.0.1'; % indirizzo IP per emulation mode
portS = 4012; % porta per invio
mode = 2; % modalità di invio dei pacchetti: con 1 invio una ...
    configurazione alla volta, con 2 le invio tutte insieme e ...
    muovo solo successivamente

% Creazione connessione TCP/IP per invio posizioni (tcpip).
connectionSend = tcpip(host, portS, 'NetworkRole', 'client');
```

```
13 connectionSend.Terminator = 13;
14 set(connectionSend, 'InputBufferSize', 58);
15 fopen(connectionSend);
16 disp('ConnectionSend pronta a inviare!');
17
18 %% Caricamento delle posizioni ottenute dall'ottimizzazione
19 load('jointRot.mat');
20 load('dati.mat');
21 load('LambdaOttimo.mat');
22
23 dy = dati.dy;
24 dz = dati.dz;
25
26 data_to_convert = qtot;
27
28 % Tempi scalati
29 scalaMin = 0.1;
30 if La < scalaMin
31     scala = scalaMin;
32 else
33     scala = La;
34 end
35 dt = scala*dati.dt;
36 T = scala*dati.T;
37
38 Ttot = T;
39
40 ttot = [0,0:dt:Ttot];
41 step = 1;
42
43 % Pacchetto con informazioni iniziali
44 pacchetto1 = [mode,size(qtot,1),dt,dy,dz,qtot(1,:)];
45
46 % Inizializzo matrice delle posizioni correnti
47 loc_read = [];
48 tempi = [];
```

```
49
50 i = 1;
51 tic %avvio il contatore
52 ttime = 0;
53 while i < length(ttot)
54
55     tic
56
57     if i == 1
58
59         % Invio un pacchetto ad ACE per indicare la modalità ...
               voluta
60         disp('Inizio set modalità')
61         sendPositionACE(connectionSend,pacchetto1,0);
62         disp('Set modalità completo')
63         pause(0.7)
64         i = i+1;
65
66     else
67
68         % Mantiene aperta la connessione ed invia le ...
               posizioni ad ACE
69         if mode == 1
70             % Lettura della posizione corrente inviata da ACE
71             str = sendPositionACE(connectionSend, ...
                   data_to_convert(i,:), 1);
72             fprintf('Invio pacchetto %d completo\n',i)
73             r1 = fliplr(uint8(str(1:end-2)));
74             r = fliplr(typecast(r1,'double'));
75             loc_read = [loc_read; r(2:7)];
76             tempi = [tempi; r(1)];
77
78         else
79
80             % Se la modalità è la 2, invio tutto senza conferma
```

```
81              sendPositionACE(connectionSend, ...
                    data_to_convert(i,:), 0);
82              fprintf('Invio pacchetto %d completo\n',i)
83
84          end
85
86          % TEMPORIZZAZIONE:
87          % Da usare solo in modalità diretta "mode = 1"
88          % "toc" = Lettura del tempo trascorso per eseguire il ...
                singolo ciclo
89          % "time" = Restituisce il tempo in secondi
90          if mode == 1
91              if i < length(ttot)
92                  time = toc;
93                  ttime = ttime + time; %%
94                  idx = find(ttot > ttime,1); %%
95                  Attesa = ttot(i+step)-(ttot(i)+time);
96                  if Attesa > 0
97                      pause(Attesa);
98                  end
99                  i = idx +1;
100             else
101                 pause(Attesa);
102             end
103             %i = i+1;
104         else
105             pause(0.01)
106             i = i+1;
107         end
108
109     end
110
111 end
112
113 toc %spengo il contatore di tutta la spedizione dei pacchetti
114
```

```
115 sendPositionACE(connectionSend,1,0); % segnale di chiusura ...
       spedizione
116
117 %% VELOCITA'
118 [~,FY] = gradient(loc_read);
119 FY = FY./gradient(tempi); %velocità effettive
120 [~,FY0] = gradient(qtot,dt); %velocità teoriche
121
122 % Salvataggio delle posizioni e velocità ricevute
123 risultati.t = ttot;
124 risultati.q = qtot;
125 risultati.vel = FY0;
126 risultati.tempi = tempi;
127 risultati.locRead = loc_read;
128 risultati.velRead = FY;
129 save('risultati_','risultati')
130 end
131
132 disp(['Values received: ...
       ',num2str(connectionSend.ValuesReceived)]);
133 disp(['Values sent: ',num2str(connectionSend.ValuesSent)]);
134
135 % Disconnect and clean up the server connection.
136 fclose(connectionSend);
137 delete(connectionSend);
138 clear connectionSend
139
140 %% Function sendPositionACE
141 function str = sendPositionACE(varargin)
142
143     connectionSend = varargin{1};
144     data_to_convert = varargin{2};
145     if nargin > 2
146         wait = varargin{3}; % 0 non aspettare |1 aspetta una ...
               risposta da ACE
147     end
```

```
148
149     joints_to_convert = fliplr(data_to_convert);
150
151     % generazione dati da inviare come sequenze di byte
152     joints_to_send = fliplr(typecast(joints_to_convert, ...
            'uint8')); %get byte array
153
154     % mando le coordinate dei punti ad ACE
155     fwrite(connectionSend, joints_to_send);
156
157     % leggo la conferma di movimento inviata da ACE
158     if (nargin < 3) || (wait > 0)
159         str = fread(connectionSend); % leggo i dati inviati ...
                da ACE
160         str = str';
161     end
162
163 end
```

Listing B.2: TestTimer_Read.m

```
1  % Script per creazione timer ricezione posizioni da ACE
2  clear all
3  close all
4  clc
5
6
7  %% Connessione TCP
8  host = '192.168.0.2'; % indirizzo IP per controller Viper
9  % host = '127.0.0.1'; % indirizzo IP per emulation mode
10 portR = 4013; % porta per ricezione
11
12 % Creo la connessione per la ricezione delle posizioni
13 connectionRead = tcpip(host, portR, 'NetworkRole', 'client');
14 connectionRead.Terminator = 13;
15 set(connectionRead, 'InputBufferSize', 58); % 7*8+2 = 58
```

```
16 fopen(connectionRead); % metto Matlab in ascolto per la ricezione
17 disp('ConnectionRead pronta a ricevere!');
18
19 %% Caricamento delle posizioni e dei dati
20 load('jointRot.mat');
21 load('dati.mat');
22 load('LambdaOttimo.mat');
23
24 dy = dati.dy;
25 dz = dati.dz;
26
27 % Tempi scalati
28 scalaMin = 0.1;
29 if La < scalaMin
30     scala = scalaMin;
31 else
32     scala = La;
33 end
34 % scala = 1; %maggiore di La
35 dt = scala*dati.dt;
36 T = scala*dati.T;
37
38 q = qtot;
39
40 t = 0:dt:T;
41
42 %% Creazione del timer per la ricezione
43 timR = timer;
44 timR.Name = 'Timer Ricezione';
45 timR.Period = 0.001;
46 timR.StartFcn = @TimerStart;
47 % tim.StopFcn = @TimerCleanup;
48 timR.TimerFcn = @timerRead;
49 timR.ExecutionMode = 'fixedRate';
50 timR.UserData.tempoPassato = tic;
51 timR.UserData.connectionRead = connectionRead;
```

```
52 timR.UserData.locRead = [];
53 timR.UserData.stop = 0;
54 timR.BusyMode = 'drop';
55
56 % Avvio dei timer
57 start(timR)
58
59 while timR.UserData.stop ~= 1
60     continue
61 end
62 [tempi,locRead] = showTimerOutput(timR);
63 disp(tempi(end))
64
65 delete(timR)
66 clear timR
67
68 %% Disconnessione
69 fclose(connectionRead);
70 delete(connectionRead);
71 clear connectionRead
72
73 % Funzione di avvio del timer
74 function TimerStart(mTimer,~)
75     str = ['Avvio del timer ',mTimer.Name,'.'];
76     disp(str)
77 end
```

Listing B.3: TestTimer_Send.m

```
1 % Script per creazione timer invio e spedizione posizioni
2 clear all
3 close all
4 clc
5
6 host = '192.168.0.2'; % indirizzo IP per controller Viper
7 % host = '127.0.0.1'; % indirizzo IP per emulation mode
```

```
portS = 4012; % porta per invio

mode = 1; % modalità di invio dei pacchetti: con 1 invio una ...
    configurazione alla volta, con 2 le invio tutte insieme e ...
    muovo solo successivamente

% Creazione connessione TCP/IP per invio posizioni (tcpip).
connectionSend = tcpip(host, portS, 'NetworkRole', 'client');
connectionSend.Terminator = 13;
set(connectionSend, 'InputBufferSize', 58);
set(connectionSend, 'RecordMode','append');
fopen(connectionSend);
disp('ConnectionSend pronta a inviare!');


%% Caricamento delle posizioni e dei dati
load('jointRot.mat');
load('dati.mat');
load('LambdaOttimo.mat');

dy = dati.dy;
dz = dati.dz;

% Tempi scalati
scalaMin = 1;
if La < scalaMin
    scala = scalaMin;
else
    scala = La;
end
dt = scala*dati.dt;
T = scala*dati.T;

q = qtot;

t = 0:dt:T;
```

```
42
43 % Pacchetto informazioni iniziali
44 pacchetto1 = [size(q,1),dt,dy,dz,q(1,:)];
45 sendPositionACE(connectionSend,pacchetto1,0);
46 pause(1)
47
48 % Creazione del timer per la spedizione
49 timS = timer;
50 timS.Name = 'Timer Invio';
51 timS.Period = dt;
52 timS.StartFcn = @TimerStart;
53 timS.TimerFcn = @timerSend;
54 timS.ExecutionMode = 'fixedRate';
55 timS.TasksToExecute = length(t);
56 timS.UserData.q = q;
57 timS.UserData.t = t;
58 if mode == 1
59     timS.UserData.mode = 1;
60 else
61     timS.UserData.mode = 0;
62 end
63 timS.UserData.tempoPassato = tic;
64 timS.UserData.connectionSend = connectionSend;
65 timS.UserData.timeCycle = 0;
66 timS.UserData.id = [];
67 timS.BusyMode = 'drop';
68
69 % Avvio del timer
70 start(timS)
71
72 %% Disconnessione
73 disp(['Values received: ...
       ',num2str(connectionSend.ValuesReceived)]);
74 disp(['Values sent: ',num2str(connectionSend.ValuesSent)]);
75
76 % Disconnect and clean up the server connection.
```

```
77 fclose(connectionSend);
78 delete(connectionSend);
79 clear connectionSend
80
81 %% Function per avvio e chiusura timer
82
83 function TimerStart(mTimer,~)
84 str = ['Avvio del timer ',mTimer.Name,'.'];
85 disp(str)
86 end
```

Listing B.4: sendPositionACE.m

```
1  %% spedizione dato di posizione ad ACE
2
3  function str = sendPositionACE(varargin)
4
5      connectionSend = varargin{1};
6      data_to_convert = varargin{2};
7      if nargin > 2
8          wait = varargin{3}; % 0 non aspettare |1 aspetta una ...
               risposta da ACE
9      end
10
11     joints_to_convert = fliplr(data_to_convert);
12
13     % generazione dati da inviare come sequenze di byte
14     joints_to_send = fliplr(typecast(joints_to_convert, ...
           'uint8')); %get byte array
15
16     % mando le coordinate dei punti ad ACE
17     fwrite(connectionSend, joints_to_send);
18
19     % leggo la conferma di movimento inviata da ACE
20     if (nargin < 3) || (wait > 0)
21         str = fread(connectionSend); % leggo i dati inviati ...
```

```
              da ACE
22          str = str';
23  %         r1 = fliplr(uint8(str(1:end-2)));
24  %         r = fliplr(typecast(r1,'double'));
25      end
26
27  end
```

Listing B.5: readPositionACE.m

```
1   %% lettura dato di posizione da ACE
2
3   function r = readPositionACE(connection)
4
5       str = fread(connection); % leggo i dati inviati da ACE
6       str = str';
7       r1 = fliplr(uint8(str(1:end-2))); % memorizzo come uint8 ...
            (stringa di
8       % 8 byte), escludendo gli ultimi due elementi del vettore ...
            (13 10),
9       % poiché rappresentano il terminator. Infine flippo il ...
            vettore.
10      r = fliplr(typecast(r1,'double')); % converto in double e ...
            flippo di
11      %nuovo il vettore
12
13  end
```

Listing B.6: timerSend.m

```
function timerSend(myTimerObj, ~)

    myTimerObj.UserData.timeCycle = tic;

     fprintf('Esecuzione = %d\n',myTimerObj.TasksExecuted)
%     fprintf('\tPeriod = %.4f\n',myTimerObj.Period);
    if myTimerObj.TasksExecuted > 1
%         fprintf('\tAveragePeriod = ...
    %.4f\n',myTimerObj.AveragePeriod);
        fprintf('\tInstantPeriod = ...
            %.4f\n',myTimerObj.InstantPeriod);

%         % Non ha senso inserire questi comandi perché il ...
    timer assolve
%         % già a questa funzione
%         nSkip = ...
    ceil(myTimerObj.InstantPeriod/myTimerObj.Period);
%         myTimerObj.UserData.count = ...
    myTimerObj.UserData.count + nSkip;
%         fprintf('\tEffettivo punto = ...
    %d\n',myTimerObj.UserData.count);
    end

% idx = 1;
% if idx <= length(myTimerObj.UserData.t)
% if toc(myTimerObj.UserData.tempoPassato) < ...
    myTimerObj.UserData.t(end)

        tIst = toc(myTimerObj.UserData.tempoPassato); %tempo ...
            impiegato dall'avvio del timer
        time = toc(myTimerObj.UserData.timeCycle); %tempo ...
            impiegato dall'inizio di questo ciclo
        fprintf('\tTempo passato = %.4f\n',tIst);

        idx = find(myTimerObj.UserData.t > tIst,1);
```

```
27          myTimerObj.UserData.id = [myTimerObj.UserData.id ; idx];
28
29      if idx < length(myTimerObj.UserData.t)
30  %       if tIst < myTimerObj.UserData.t(end)
31          qDaSpedire = myTimerObj.UserData.q(idx,:);
32          % Spedizione dato posizione
33          sendPositionACE(myTimerObj.UserData.connectionSend, ...
                qDaSpedire, myTimerObj.UserData.mode);
34
35          Attesa = myTimerObj.UserData.t(idx+1) - ...
                (myTimerObj.UserData.t(idx) + time);
36          pause(Attesa)
37      else
38          sendPositionACE(myTimerObj.UserData.connectionSend,1,0); ...
                % segnale di chiusura
39          TimerCleanup(myTimerObj);
40      end
41  % end
42
43  % myTimerObj.TasksToExecute = idx;
44
45  % tToWait = myTimerObj.InstantPeriod-myTimerObj.Period;
46  % fprintf('\tAttesa = %.4f\n',tToWait)
47  % pause(tToWait)
48
49  end
```

Listing B.7: timerRead.m

```
1   function timerRead(myTimerObj, ~)
2
3       myTimerObj.UserData.timeCycle = tic;
4
5       fprintf('Esecuzione = %d\n',myTimerObj.TasksExecuted)
6   %       fprintf('\tPeriod = %.4f\n',myTimerObj.Period);
7       if myTimerObj.TasksExecuted > 1
```

```
8  %         fprintf('\tAveragePeriod = ...
       %.4f\n',myTimerObj.AveragePeriod);
9          fprintf('\tInstantPeriod = ...
               %.4f\n',myTimerObj.InstantPeriod);
10     end
11
12 %         tIst = toc(myTimerObj.UserData.tempoPassato); ...
       %tempo impiegato dall'avvio del timer
13
14         % Lettura dato posizione
15         r = readPositionACE(myTimerObj.UserData.connectionRead);
16 %         fprintf('Posizione attuale = %d\n',r);
17         if length(r) == 7
18             myTimerObj.UserData.locRead = ...
                   [myTimerObj.UserData.locRead; r];
19         else
20             myTimerObj.UserData.stop = 1;
21             stop(myTimerObj)
22             str = ['Stop del timer ',myTimerObj.Name,'.'];
23             disp(str)
24 %             myTimerObj.UserData.Output = ...
       showTimerOutput(myTimerObj);
25             myTimerObj.UserData.Output = ...
                   myTimerObj.UserData.locRead;
26         end
27
28 end
```

Listing B.8: showTimerOutput.m

```
function [tempi,locRead] = showTimerOutput(mTimer)

    tempi = mTimer.UserData.Output(:,1);
    locRead = mTimer.UserData.Output(:,2:7);

end
```

Listing B.9: TimerCleanup.m

```
function TimerCleanup(mTimer)

disp(['Il timer ',mTimer.Name,' ha eseguito ...
    ',num2str(mTimer.TasksExecuted),' operazioni su ...
    ',num2str(mTimer.TasksToExecute),' totali.'])
disp('Stop ed eliminazione del timer.')
stop(mTimer)
delete(mTimer)

end
```

B.2 Programmi ACE

Listing B.10: mat2ACE.awp

```
.PROGRAM mat2ace()

        SPEED 100 ALWAYS
        CPON ALWAYS

        ATTACH ()

        SET #home = #PPOINT(0,-90,180,0,45,0)
        SET #appro = #PPOINT(-20,-57,197,0,40,0)
```

```
10
11          MOVE #home
12
13          ; Avvio la prima connessione (lettura e movimento)
14          ATTACH (lun, 4) "TCP"
15          IF IOSTAT(lun) < 0 THEN
16              TYPE "Attach error: ", $ERROR(IOSTAT(lun))
17          END
18          FOPEN (lun, 16) "/LOCAL_PORT 4012 /CLIENTS 1 ...
                /BUFFER_SIZE 1024"
19          ; Apro il socket con matlab, attendo la connessione
20          READ (lun, 0, 0) $in.str
21
22          status = IOSTAT(lun)
23
24          DO
25              status = IOSTAT(lun)
26              RELEASE -1 ;wait
27          UNTIL status == 100
28
29          ; Inizializzazione del segnale di conferma di movimento
30          go = 1
31          $rx_mon1 = $DBLB(go)
32
33          TYPE "go"
34          SPEED 30
35          MOVE #appro
36
37          ; Modalità di movimento
38          $in.str = ""
39          READ (lun, 0, 0) $in.str
40          mode = DBLB($MID($in.str,1,8)) ;modalità di movimento
41          length = DBLB($MID($in.str,9,8)) ;numero di posizioni ...
                da ricevere
42          dt = DBLB($MID($in.str,17,8)) ;step temporale tra due ...
                movimenti consecutivi
```

```
43          dy = DBLB($MID($in.str,25,8)) ;offset tool direzione y
44          dz = DBLB($MID($in.str,33,8)) ;offset tool direzione z
45          ;COORDINATE DEL PRIMO PUNTO
46          punto_i[0] = DBLB($MID($in.str,41,8))
47          punto_i[1] = DBLB($MID($in.str,49,8))
48          punto_i[2] = DBLB($MID($in.str,57,8))
49          punto_i[3] = DBLB($MID($in.str,65,8))
50          punto_i[4] = DBLB($MID($in.str,73,8))
51          punto_i[5] = DBLB($MID($in.str,81,8))
52
53          TOOL TRANS(0,dy,dz,0,0,0)
54
55          SPEED 100 ALWAYS
56          JMOVE punto_i[0], punto_i[1], punto_i[2], punto_i[3], ...
                punto_i[4], punto_i[5]
57
58          ; ROBOT PRONTO PER IL MOVIMENTO (SEGNALE)
59
60          CASE mode OF
61            VALUE 1: ; Modalità 1
62              TYPE "Modalità 1: una posizione alla volta"
63
64              TYPE "Inizio spedizione posizioni da Matlab ad ACE"
65              TIMER 10 = 0 ; imposta a zero il timer di ...
                    movimento completo
66              SIGNAL 1
67
68              i = 0
69              WHILE SIG(1) DO
70              ; Attendi le coordinate dei punti da matlab
71                  $in.str = ""
72                  READ (lun, 0, 0) $in.str
73
74                  IF DBLB($in.str) == 1 THEN
75                      SIGNAL -1 ;stoppa l'invio delle posizioni ...
                            a Matlab
```

```
76                  ELSE
77                      SIGNAL 1
78                      TYPE "ciao"
79
80              ;ricavo le coordinate dalla stringa
81                      mat_loc[0] = DBLB($MID($in.str,1,8))
82                      mat_loc[1] = DBLB($MID($in.str,9,8))
83                      mat_loc[2] = DBLB($MID($in.str,17,8))
84                      mat_loc[3] = DBLB($MID($in.str,25,8))
85                      mat_loc[4] = DBLB($MID($in.str,33,8))
86                      mat_loc[5] = DBLB($MID($in.str,41,8))
87
88              ; esecuzione movimento
89                      JMOVE mat_loc[0], mat_loc[1], mat_loc[2], ...
                           mat_loc[3], mat_loc[4], mat_loc[5]
90
91                      IF i < length-1 THEN
92              ;incremento il contatore del ciclo while
93                          i = i+1
94                      END
95
96              ;Invio a Matlab la posizione corrente
97                      HERE #here_loc
98                      DECOMPOSE loc[] = #here_loc    ;estraggo ...
                           i valori delle singole componenti ...
                           della locazione
99
100                     loc0 = loc[0]
101                     loc1 = loc[1]
102                     loc2 = loc[2]
103                     loc3 = loc[3]
104                     loc4 = loc[4]
105                     loc5 = loc[5]
106
107                     $attuale = $DBLB(TIMER(10)) + $DBLB(loc0) ...
                           + $DBLB(loc1) + $DBLB(loc2) + ...
```

```
                        $DBLB(loc3) + $DBLB(loc4) + $DBLB(loc5)
108
109                     TYPE "Pacchetto numero ", i, " ricevuto"
110
111                     WRITE (lun) $attuale ;invio a matlab le ...
                            posizioni correnti
112                 END
113
114             END
115
116             TYPE "Tempo impiegato per il movimento = ", ...
                    TIMER(10), "secondi"
117             SIGNAL -1
118
119
120         VALUE 2: ; Modalità 2
121             TYPE "Modalità 2: tutte le configurazioni ...
                    inviate, movimento eseguito solo dopo"
122             SIGNAL 2
123
124             i = 0
125             ;ciclo per l'acquisizione di tutte le posizioni e ...
                    la memorizzazione nell'array mat_loc
126             WHILE SIG(2) DO
127
128             ; Attendi le coordinate dei punti da matlab
129                 $in.str = ""
130                 READ (lun, 0, 0) $in.str
131
132                 IF DBLB($in.str) == 1 THEN
133                     SIGNAL -2 ;stoppa l'invio delle posizioni ...
                            a Matlab
134                 ELSE
135                     SIGNAL 2
136
137             ;ricavo le coordinate dalla stringa
```

```
138                     mat_loc[i,0] = DBLB($MID($in.str,1,8))
139                     mat_loc[i,1] = DBLB($MID($in.str,9,8))
140                     mat_loc[i,2] = DBLB($MID($in.str,17,8))
141                     mat_loc[i,3] = DBLB($MID($in.str,25,8))
142                     mat_loc[i,4] = DBLB($MID($in.str,33,8))
143                     mat_loc[i,5] = DBLB($MID($in.str,41,8))
144
145                     IF i < length THEN
146             ;incremento il contatore del ciclo while
147                         i = i+1
148                     END
149                 END
150
151             END
152
153             i_end = i
154
155             ; creazione del vettore dei tempi
156             ttot[0] = 0
157             FOR i = 1 TO i_end
158                 ttot[i] = ttot[i-1]+dt
159             END
160
161             ; ciclo per l'esecuzione movimento
162             EXECUTE 1 invio_pos_2()
163             DO
164                 WAIT
165             UNTIL SIG(1)
166
167             lim = 10    ; numero massimo ammissibile di salti ...
                    consecutivi
168
169             TIMER 5 = 0
170             count = 0
171             i = 0
172             WHILE i < i_end DO
```

```
173                 TIMER 10 = 0
174
175                 JMOVE mat_loc[i,0], mat_loc[i,1], ...
                        mat_loc[i,2], mat_loc[i,3], mat_loc[i,4], ...
                        mat_loc[i,5]
176
177             ; Temporizzazione
178                 IF i < length THEN
179
180                     attesa = ttot[i+1]-(ttot[i]+TIMER(10))
181
182                     idx = INT(TIMER(5)/dt)
183                     TYPE idx
184
185                     IF attesa > 0 THEN
186                         WAIT.EVENT , attesa
187                     END
188                     i = idx+1
189                 ELSE
190                     WAIT.EVENT , attesa
191                 END
192
193                 count = count+1
194             END
195
196             SIGNAL -1
197
198             TYPE TIMER(5)
199             TYPE "Il robot ha eseguito ", count, " movimenti"
200
201         END
202         TYPE "Fine movimento"
203
204         MOVE #appro
205
```

```
206         ; Al termine del movimento torno nella posizione di ...
                #home (per settaggio tool) passando per la #home
207         MOVE #home
208
209         DETACH (lun)
210         DETACH ()
211
212 .END
```

Listing B.11: main.awp

```
1  .PROGRAM main()
2
3           SIGNAL -1
4           EXECUTE 1 invio_pos_1()
5
6           SPEED 100 ALWAYS
7           CPON ALWAYS
8
9           ATTACH ()
10
11          SET #home = #PPOINT(0,-90,180,0,45,0)
12          SET #appro = #PPOINT(-16,-30,183,0,26,0)
13
14          MOVE #home
15          BREAK
16
17          ; Avvio prima connessione
18          ATTACH (lun, 4) "TCP"
19          IF IOSTAT(lun) < 0 THEN
20              TYPE "Attach error: ", $ERROR(IOSTAT(lun))
21          END
22          FOPEN (lun, 16) "/LOCAL_PORT 4012 /CLIENTS 1 ...
                /BUFFER_SIZE 1024"
23          ;apro il socket con matlab
24          ; Attendo la connessione
```

```
25          READ (lun, 0, 0) $in.str
26
27          status = IOSTAT(lun)
28
29          ; Inizializzazione del segnale di conferma di movimento
30          go = 1
31          $rx_mon1 = $DBLB(go)
32          $risposta = $DBLB(1) + $DBLB(1) + $DBLB(1) + $DBLB(1) ...
               + $DBLB(1) + $DBLB(1) + $DBLB(1)
33
34          TYPE "go"
35          SPEED 30
36          MOVE #appro
37
38          ; Modalità di movimento
39          $in.str = ""
40          READ (lun, 0, 0) $in.str
41          length = DBLB($MID($in.str,1,8)) ;numero di posizioni ...
               da ricevere
42          dt = DBLB($MID($in.str,9,8)) ;step temporale tra due ...
               movimenti consecutivi
43          dy = DBLB($MID($in.str,17,8)) ;offset tool direzione y
44          dz = DBLB($MID($in.str,25,8)) ;offset tool direzione z
45          ;COORDINATE DEL PRIMO PUNTO
46          punto_i[0] = DBLB($MID($in.str,33,8))
47          punto_i[1] = DBLB($MID($in.str,41,8))
48          punto_i[2] = DBLB($MID($in.str,49,8))
49          punto_i[3] = DBLB($MID($in.str,57,8))
50          punto_i[4] = DBLB($MID($in.str,65,8))
51          punto_i[5] = DBLB($MID($in.str,73,8))
52
53          TOOL TRANS(0,dy,dz,0,0,0)
54
55          SPEED 100 ALWAYS
56          JMOVE punto_i[0], punto_i[1], punto_i[2], punto_i[3], ...
               punto_i[4], punto_i[5]
```

```
57
58          ; ROBOT PRONTO PER IL MOVIMENTO (SEGNALE)
59          SIGNAL 1
60
61          TYPE "Inizio spedizione posizioni da Matlab ad ACE"
62
63          DO
64              RELEASE -1
65          UNTIL SIG(1)
66          TYPE "segnale ok"
67
68          TIMER 10 = 0 ; imposta a zero il timer di movimento ...
                completo
69
70          i = 0
71          WHILE SIG(1) DO
72              TIMER 2 = 0
73              ; Attendi le coordinate dei punti da matlab
74              $in.str = ""
75              READ (lun, 0, 0) $in.str
76
77              IF DBLB($in.str) == 1 THEN
78                  SIGNAL -1 ;stoppa l'invio delle posizioni a ...
                        Matlab
79              ELSE
80
81              ;ricavo le coordinate dalla stringa
82                  mat_loc[0] = DBLB($MID($in.str,1,8))
83                  mat_loc[1] = DBLB($MID($in.str,9,8))
84                  mat_loc[2] = DBLB($MID($in.str,17,8))
85                  mat_loc[3] = DBLB($MID($in.str,25,8))
86                  mat_loc[4] = DBLB($MID($in.str,33,8))
87                  mat_loc[5] = DBLB($MID($in.str,41,8))
88
89              ; esecuzione movimento
90                  JMOVE mat_loc[0], mat_loc[1], mat_loc[2], ...
```

```
                    mat_loc[3], mat_loc[4], mat_loc[5]
91
92                  IF i < length-1 THEN
93              ;incremento il contatore del ciclo while
94                      i = i+1
95                  END
96
97                  TYPE "Il tempo ciclo è pari a ", TIMER(2)
98                  TYPE "Pacchetto numero ", i, " ricevuto"
99                  WRITE (lun) $risposta
100
101             END
102
103         END
104
105         TYPE "Tempo impiegato per tutto il movimento = ", ...
                TIMER(10), "secondi"
106         SIGNAL -1
107
108         SPEED 30
109         MOVE #appro
110
111         ; Al termine del movimento torno nella posizione di ...
                #home (per settaggio tool) passando per la #home
112         MOVE #home
113
114         DETACH (lun)
115         DETACH ()
116
117 .END
```

Listing B.12: invio_pos_1.awp

```
1  .PROGRAM invio_pos_1()
2
3          ; Avvio la seconda connessione per la spedizione ...
```

```
            delle posizioni
4           ATTACH (lun1, 4) "TCP"
5           FOPEN (lun1, 16) "/LOCAL_PORT 4013 /CLIENTS 1 ...
               /BUFFER_SIZE 1024"
6           ; Attendo la connessione
7           READ (lun1, 0, 0) $str
8
9           DO
10              status = IOSTAT(lun)
11              RELEASE -1 ;wait
12          UNTIL status == 100
13
14          DO
15              RELEASE -1 ;wait
16          UNTIL SIG(1)
17
18          TIMER 15 = 0 ; imposta a zero il timer di movimento ...
               completo
19
20          WHILE SIG(1) DO
21
22              ;Invio a Matlab la posizione corrente
23              HERE #here_loc
24              DECOMPOSE loc[0] = #here_loc ;estraggo i valori ...
                   delle singole componenti della locazione
25
26              loc0 = loc[0]
27              loc1 = loc[1]
28              loc2 = loc[2]
29              loc3 = loc[3]
30              loc4 = loc[4]
31              loc5 = loc[5]
32
33              $attuale = $DBLB(TIMER(15)) + $DBLB(loc0) + ...
                   $DBLB(loc1) + $DBLB(loc2) + $DBLB(loc3) + ...
                   $DBLB(loc4) + $DBLB(loc5)
```

```
34
35          END
36
37          WRITE (lun1) $DBLB(1)
38
39  .END
```

Listing B.13: invio_pos_2.awp

```
1   .PROGRAM invio_pos_2()
2
3           ; Avvio la seconda connessione per la spedizione ...
              delle posizioni
4           ATTACH (lun1, 4) "TCP"
5           FOPEN (lun1, 16) "/LOCAL_PORT 4013 /CLIENTS 1 ...
              /BUFFER_SIZE 1024"
6           ; Attendo la connessione
7           READ (lun1, 0, 0) $str
8
9           DO
10              status = IOSTAT(lun)
11              RELEASE -1 ;wait
12          UNTIL status == 100
13          SIGNAL 1
14
15          TIMER 15 = 0 ; imposta a zero il timer di movimento ...
              completo
16
17          WHILE SIG(1) DO
18
19              ;Invio a Matlab la posizione corrente
20              HERE #here_loc
21              DECOMPOSE loc[0] = #here_loc ;estraggo i valori ...
                  delle singole componenti della locazione
22
23              loc0 = loc[0]
```

```
24              loc1 = loc[1]
25              loc2 = loc[2]
26              loc3 = loc[3]
27              loc4 = loc[4]
28              loc5 = loc[5]
29
30              $attuale = $DBLB(TIMER(15)) + $DBLB(loc0) + ...
                    $DBLB(loc1) + $DBLB(loc2) + $DBLB(loc3) + ...
                    $DBLB(loc4) + $DBLB(loc5)
31
32              WRITE (lun1) $attuale
33
34          END
35
36          WRITE (lun1) $DBLB(1)
37
38  .END
```

Bibliografia

[1] L. Clemente, "Ottimizzazione delle traiettorie dei giunti di un manipolatore industriale," Tesi di Laurea Magistrale in Ingegneria Meccanica, Università degli Studi di Padova, 2019-2020.

[2] G. Legnani and I. Fassi, *Robotica Industriale*. CittàStudi, 2018.

[3] IFR. (2020) Executive summary world robotics 2019 - industrial robots. [Online]. Available: https://www.ifr.org

[4] Omron. (2020) Industrial robotics automation catalog - product datasheets. [Online]. Available: https://industrial.omron.eu

[5] G. Rosati, "Dispensa del corso di robotica industriale," Università degli Studi di Padova, Corso di Laurea in Ingegneria Meccanica, A.A. 2018-2019.

[6] J. J. Craig, *Introduction to Robotics Mechanism and control*, 3rd ed. Pearson Prentice Hall, 2005.

[7] B. Siciliano, "Dispense del corso di robotica avanzata," Università degli Studi di Napoli "Federico II", A.A. 2019-2020. [Online]. Available: prisma.dieti.unina.it

[8] MathWorks, *MatLAB - Object-oriented programming*, September 2018.

[9] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Springer, 2016.

[10] G. Rosati, S. Cocuzza, M. Bottin, N. Comand, and G. Cipriani, "Optimal path planning of a redundant robot in food industry," *Advances in Italian Mechanism Science. IFToMM ITALY 2020. Mechanisms and Machine Science*, vol. 91, pp. 769–780, 2020.

[11] SpeedCheck. (2018) Protocollo di controllo della trasmissione e protocollo internet (tcp/ip). [Online]. Available: www.speedcheck.org/it/wiki/tcp-ip

[12] VisualBasicSimple. (2001) Introduzione alla tecnologia client / server e tcp/ip. [Online]. Available: www.vbsimple.net/cliserv/clser_00.htm

[13] MathWorks. (2020) Documentazione online. [Online]. Available: https://it.mathworks.com/support.html

Ringraziamenti

Desidero innanzitutto ringraziare il professor Giulio Rosati per i preziosi insegnamenti da lui ricevuti durante questa laurea magistrale e per avermi consentito di svolgere questa tesi.
Ringrazio, inoltre, l'ingegner Matteo Bottin per il tempo e la disponibilità concessi durante il lavoro della tesi.
Ringrazio mia mamma Cristina, mio papà Franco, le mie sorelle Giulia e Laura per la presenza ed il sostegno che mi danno ogni giorno della mia vita. Li ringrazio per avere investito amore e fiducia nell'educarmi e per avermi consentito di costruirmi un futuro.
Ringrazio mio nonno Cesare, che è e sarà sempre per me un riferimento di saggezza e bontà. Ringrazio i nonni Isa, Elda e Antonio, che mi hanno dato tanto affetto ed ora sono i miei angeli custodi.
Ringrazio i miei zii Marta e Gianluca e i miei cugini Anna, Davide e Francesco, per i tanti bei momenti condivisi fin dall'infanzia. Siamo proprio una bella famiglia!
Un ringraziamento va sicuramente ad Unipd, che mi ha addirittura fatto spuntare un po' di capelli bianchi, per le tante ore ed energie spese nello studio, ma che è anche stata un luogo in cui maturare e formarmi, nonché una seconda casa in questi ultimi 5 anni. Qui ho trovato degli amici meravigliosi con cui alleggerire un po' la fatica dello studio, facendoci spalla a vicenda, e con cui vivere avventure, vacanze e momenti di svago anche fuori dall'università.
Ci tengo anche a ringraziare tutti i ragazzi del Team Quartodilitro UNIPD, un progetto a cui ho dedicato parecchio tempo ed energie in questi ultimi 2 anni, che mi ha fatto crescere, mettere in gioco, maturare competenze utili nel futuro, ma soprattutto lavorare di squadra. Li ringrazio per la simpatia e per le gioie condivise in mezzo ad un mare di imprevisti. Ed ora, attendiamo Aragon 2021 carichi a molla!
Ringrazio Zanna, di cui sono amico dai tempi dell'asilo, per tutte le esperienze condivise, per la stima, l'amicizia, l'ascolto.
Ed ora un grande grazie va a tutti gli amici compaesani, che sono troppi per essere nominati uno ad uno, ma che hanno condiviso con me un sacco di esperienze, dall'AC a scout, dai campi ai weekend di formazione educatori, dal Vicoretto ai momenti di festa e bagordi. Li ringrazio per la loro delicatezza e per il loro travolgimento, per i silenzi e per le risate, per gli abbracci e per le pacche.
Ringrazio la Commissione Campi diocesana, di cui faccio parte ormai da un po', ora addirittura in veste di referente. La ringrazio perché, nonostante sia un ulteriore impegno in mezzo a tanti altri, non mi ha mai portato a pensarlo; perché è sempre un'occasione di crescita, di amicizia, di risate, a cui sono affezionato; perché è un gruppo di persone che ti fanno stare bene.
Grazie a tutti, anche a quelli che non ho nominato, ma che credono in me ogni giorno.
A voi dico: "Grazie di cuore, vi voglio bene. Questo traguardo è anche un po' vostro".