POLITECNICO DI MILANO

Scuola di Ingegneria Industriale

Corso di Laurea Specialistica in Ingegneria Meccanica

COORDINAZIONE E INVERSIONE CINEMATICA PER MANIPOLATORI RIDONDANTI IN UN SISTEMA MULTI-ROBOT

Relatore: **Prof. Francesco BRAGHIN**

Correlatore: **Ing. Federico VICENTINI**

Tesi di Laurea Specialistica di:

Paolo MAGNONI Matr. **740009**

Anno Accademico 2011-2012

“I Want to break F.R.I.”

- *Queen , misread lyrics*

Ringraziamenti

Desidero innanzitutto ringraziare il mio relatore ing. Francesco Braghin e, in particolare, il mio correlatore ing. Federico Vicentini.

Desidero ringraziare, inoltre, tutto il gruppo di ricerca di ITIA-CNR, con particolare riferimento agli ing. Matteo Malosio, Nicola Pedrocchi, Loris Roveda.

Un sentito ringraziamento anche a tutte le persone che mi sono state vicine in questi anni.

Indice

Indice delle Figure

Indice delle tabelle

Abstract

Nella presente tesi, sviluppata presso l'Istituto di Tecnologie Industriali e Automazione del Consiglio Nazionale delle Ricerche (ITIA-CNR), viene descritto un approccio per la coordinazione in real-time di due manipolatori antropomorfi KUKA LWR4+ in interazione con un ambiente dinamico e cercando di ottimizzarne la configurazione assunta in funzione dei task da eseguire.

Questo renderà necessario definire un framework nel *Task Space* con cui descrivere il sistema e legarlo ai vari input possibili e operazioni eseguibili.

Grazie a tale formalizzazione sarà possibile realizzare algoritmi di coordinazione dei manipolatori sia in ambienti di simulazione Matlab, sia in ambienti embedded per il controllo real-time in remoto dei robot.

Si renderà inoltre necessaria un'operazione di inversione cinematica per il passaggio dal *Task Space* in cui si colloca il suddetto formalismo allo spazio dei giunti in cui viene controllato il robot.

Si analizzeranno quindi diversi algoritmi di inversione cinematica che siano in grado di garantire le migliori prestazioni in un contesto come quello considerato, caratterizzato da possibili saturazioni in velocità e in escursione ai giunti, e che sfruttino la ridondanza dei manipolatori per ottimizzarne la postura relativa in funzione dei task da eseguire. Tra i metodi analizzati si considererà particolarmente l'algoritmo di *Saturazione nel Null Space,* aggiungendo la possibilità di compiere task secondari come *l'obstacle avoidance,* o la massimizzazione di un indice di controllabilità in forza che indichi la bontà del posizionamento reciproco dei manipolatori per operazioni che li vedono interagire a contatto.

Ogni algoritmo di inversione cinematica sarà quindi implementato sia in ambiente Matlab che sul programma scritto in C++ agente sul sistema reale. I risultati delle simulazioni e delle applicazioni sui robot reali mostreranno l'efficacia degli algoritmi di coordinazione e di inversione cinematica sviluppati.

Abstract

In this thesis, developed in ITIA-CNR laboratories, an approach for real-time coordination of two anthropomorphic manipulators KUKA LWR4 + is discussed. These manipulators are also asked to interact with a dynamic environment and to optimize the assumed configuration, depending on the required task.

This will make necessary the definition of a framework (inspired by the *Task Space Formalism*) to describe the system and to deal with the various possible inputs and executable tasks.

Thanks to this framework it will be possible to implement algorithms for manipulators coordination both in Matlab simulation environments, both in a C++ code used to develop a robot control application.

The above framework takes place in the *Task Space,* so it needs to be transformed via the inverse kinematics into corresponding joint trajectories, which then constitute the reference inputs to the robot joint space control scheme.

Different kinematic inversion algorithms are thus analyzed. They have to guarantee the best performance in this context, characterized by possible joint saturations. They also have to exploit the manipulators redundancy in order to optimize the posture as a function of the task to perform.

The CLIK and the *Saturation in the Null Space* algorithms will be discussed, adding the possibility to perform secondary tasks such as obstacle avoidance, joints limits avoidance, or the maximization of a controllability index which is maximum if the mutual configuration of the robots is the best to perform contact between robots (max compliance).

Each inverse kinematics algorithm will be implemented in the Matlab environment and in the C + + program acting on the real system.

Capitolo 1

Introduzione

1.1 Il Sistema Multi-Robot: Contesto

Vi sono molte operazioni che possono richiedere un'azione cooperativa da parte di due o più manipolatori, anche in interazione con altri devices presenti in un sistema o in interazione con l'ambiente esterno.

Possono esservi casi in cui tutti i robot agiscono su uno stesso oggetto, ad esempio per il sollevamento di un carico pesante, voluminoso o non rigido, oppure casi in cui cooperano per effettuare delle operazioni di assemblaggio anche complesse (vedi Fig. 1.1a in cui "Justin", robot del centro di studi aerospaziali tedesco DLR apre e chiude un tappo avvitato grazie alla coordinazione delle dita che lo afferrano). Il problema di far cooperare al meglio i due o più manipolatori agenti sull'oggetto non è banale: si forma infatti tra i robot e l'oggetto una catena cinematica chiusa in cui i gradi di libertà a disposizione del sistema robot più oggetto sono maggiori di quelli in genere specificati per il task (tanto più se anche il singolo robot risulta ridondante, come nel caso che verrà considerato). Inoltre tale accoppiamento porta ad equazioni cinematiche e dinamiche non lineari e a sforzi interni sull'oggetto manipolato [1] [2] [3].

Se poi, oltre ai bracci robotici e all'oggetto manipolato, sono presenti nel sistema anche altri devices o fonti di incertezze che richiedono monitoraggio, la coordinazione si complica ulteriormente. In questi casi infatti non si può pianificare compiutamente il comportamento del sistema off-line e si richiedono quindi logiche di controllo distribuito real-time, acquisendo ad ogni istante

informazioni sull'ambiente circostante quali presenza di ostacoli, posizione corretta dell'oggetto da afferrare mediante sistemi di visione, comandi aptici. Esempi di sistemi multi-robot che interagiscono con l'ambiente possono trovarsi in ambito chirurgico (vedi in Fig. 1.1b il robot per cardiochirurgia "Da Vinci") ma anche in ambito industriale (ad esempio in Fig. 1.1c si ha un sistema multi-robot che grazie a un dispositivo di visione completa saldature su pezzi assemblati).

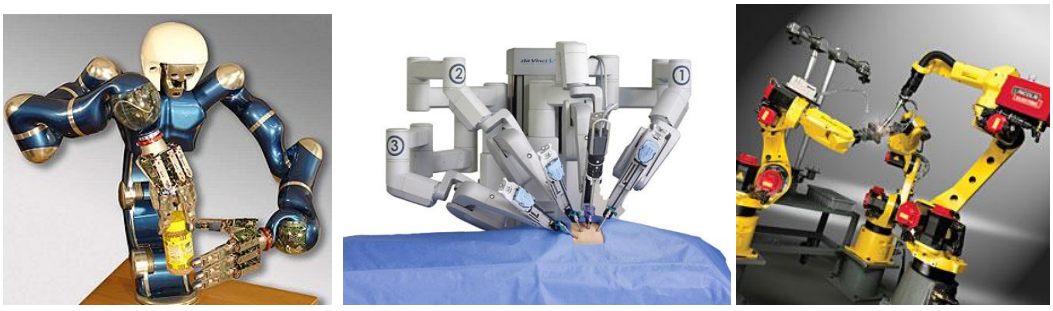


Figura 1.1 : Esempi di più manipolatori cooperanti: (a) Robot "Justin"; (b) Robot chirurgico "Da Vinci" ; (c) Robot Fanuc in sistema di saldatura Lincoln Electric

Riassumendo, le problematiche generali qui affrontate vanno da quella di definire come i manipolatori debbano agire nei confronti di un ambiente variabile o nei confronti degli altri devices, a quella di coordinarli tra loro in modo da disporli con una configurazione di giunti ottimale considerando il task da eseguire, a quella di rappresentare tale azione di coordinamento per poterla descrivere, pianificare, sintetizzare in un software. Il sistema multi-robot reale, con l'aggiunta di altri dispositivi remoti per applicazioni di controllo distribuito, è presente presso l'Istituto di Tecnologie Industriali e Automazione (ITIA) del CNR, presso il quale il presente lavoro di Tesi è stato impostato e sviluppato.

1.2 Il Sistema Multi-Robot: Obiettivi

Date queste problematiche, l'obiettivo che ci si prefigge sarà quello di coordinare il sistema multi-robot in real-time e con l'ottimizzazione delle posture relative dei manipolatori per determinati task.

I passaggi logici per conseguire l'obiettivo saranno :

- Definizione di un formalismo per poter descrivere il sistema e potergli imporre i task desiderati, considerando tutti i possibili input. Ciò sarà affrontato nel Capitolo 2, in cui richiamando il *Task Space Formalism* [1] [2] [3] [4] si presenteranno le trasformazioni di sistemi di riferimento necessarie e si analizzerà l'algoritmo di *Motion Coordination*.

- Realizzazione di un modello Matlab per validare l'algoritmo di coordinazione, simulare traiettorie e vari task possibili, come esposto nel Capitolo 3, Paragrafo 3.1. Nel Capitolo 3, Paragrafo 3.2 ci si prefiggerà di trovare un modo con cui il modello Matlab possa servire per pianificare efficacemente i moti del sistema reale, che sarà controllato in remoto da un computer real-time tramite un software in C++. Nel capitolo 3, Paragrafo 3.3 ci si prefiggerà quindi di inserire i calcoli matriciali di *Motion Coordination* nel codice C++.

- Una volta presentato il formalismo e il software per gestire nel complesso il sistema multi-robot, si deve passare a ottimizzare il posizionamento reciproco dei manipolatori, considerando anche il fatto che si tratta di robot a sette gradi di libertà, quindi ridondanti rispetto a normali task nello spazio tridimensionale. Si trovano in letteratura metodi di coordinazione multi-robot e di sfruttamento della ridondanza che passano proprio dalla definizione dei task nello spazio cartesiano come fatto con il *Task Space Formalism* e da adeguati algoritmi di inversione cinematica [1] [2] [3]. In aggiunta, la versione a disposizione della libreria che consente al software di comunicare con i manipolatori, non permette di comandarli in posizione assegnando riferimenti

nel *Task Space* cartesiano. Si ha quindi una duplice utilità nell'indagare algoritmi di inversione cinematica, ed è quello che si farà nel Capitolo 4.

- Nel Capitolo 5 si approfondirà lo studio di algoritmi di inversione cinematica per trovare i più adatti per il sistema considerato. I problemi chiave si riveleranno essere il dover invertire la cinematica in real-time, il trovarsi a volte a lavorare vicino ai limiti di giunto e, nel caso di oscillazioni rapide o traiettorie severe, anche vicino a limiti di velocità ai giunti. In 5.2 si descriverà pertanto un algoritmo presentato quest'anno in [5] che ha il suo punto di forza proprio nello sfruttare la ridondanza per compensare saturazioni di giunti o velocità. In 5.3 si estenderà questo recente algoritmo introducendo ulteriori task secondari da eseguire dove possibile. Questi potranno essere la massimizzazione della distanza da ostacoli o l'ottimizzazione della postura relativa tra due manipolatori che stanno per operare un'operazione di assemblaggio controllati in forza o impedenza. Quest'ultimo criterio di ottimizzazione verrà presentato in 5.4 e si baserà sui concetti di ellissoidi di manipolabilità in forza e velocità.

- Nel Capitolo 6 si presenterà la realizzazione degli algoritmi di inversione cinematica analizzati. Si scriveranno per il modello Matlab e si testeranno i risultati simulati. Si passerà quindi a realizzare una classe da includere nei codici C++ per eseguire calcoli di cinematica diretta e inversa sul sistema reale. Si analizzeranno i dati raccolti.

- Nel capitolo 7 si presenteranno i possibili sviluppi di questa impostazione sull'apparato a disposizione. Essi andranno da un miglioramento ulteriore degli algoritmi di inversione cinematica stessi, a criteri più complessi di ottimizzazione della postura reciproca dei robot manipolanti uno stesso oggetto, all'integrazione di inversione cinematica e controllo d'impedenza, fino alla possibilità di utilizzare il sistema per operazioni chirurgiche al cervello.

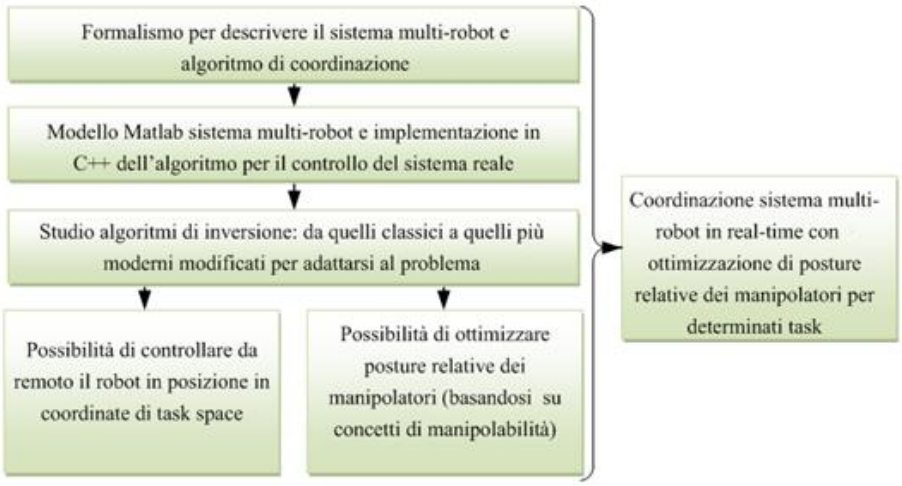


Figura 1.2 : Sinossi dei passaggi logici e degli obiettivi

Questi obiettivi sono stati raggiunti scrivendo un programma di simulazione in Matlab e un'applicazione real-time per il sistema reale. La seguente figura riassume le relazioni tra i vari elementi realizzati ed utilizzati: si distinguono i blocchi della simulazione in Matlab e dell'applicazione real-time, ciascuna con in evidenza le funzioni e le librerie che si sono sviluppate o impiegate. L'ambiente di simulazione e il sistema hardware sono interconnessi poiché una volta validata la consistenza del primo, esso sarà sfruttato per facilitare l'utilizzo del sistema reale.

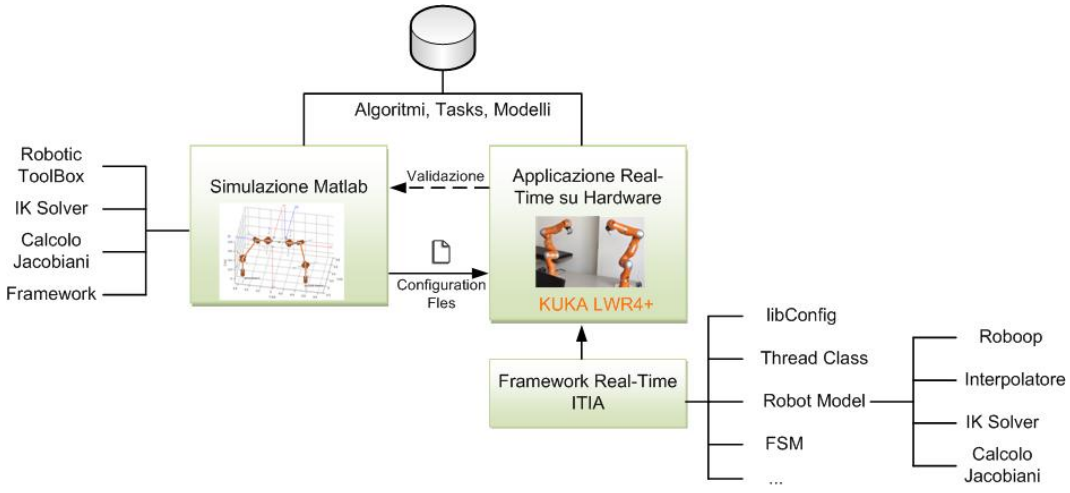


Figura 1.3 : Mappa relazionale tra gli elementi sviluppati e utilizzati

1.3 Il Sistema Multi-Robot: Descrizione

1.3.1 Il Sistema Visto come " Network Communication System "

Per arrivare all'obiettivo finale di ottimizzazione del moto dei manipolatori, il primo passo consiste nel definire un formalismo che sia adatto in generale per gestire sistemi con più devices interconnessi.

In tal senso si può inquadrare il discorso in una schematizzazione di tipo *Network Communication System* (NCS) e solo in seguito passare al caso specifico. La Figura 1.4 rappresenta un NCS caratterizzato dall'interconnessione di robot, ma anche sensori che rilevano le condizioni dell'ambiente (eventualmente organizzandole in sotto-reti), condizioni di vincolo, informazioni in ingresso provenienti da dispositivi di controllo esterni.

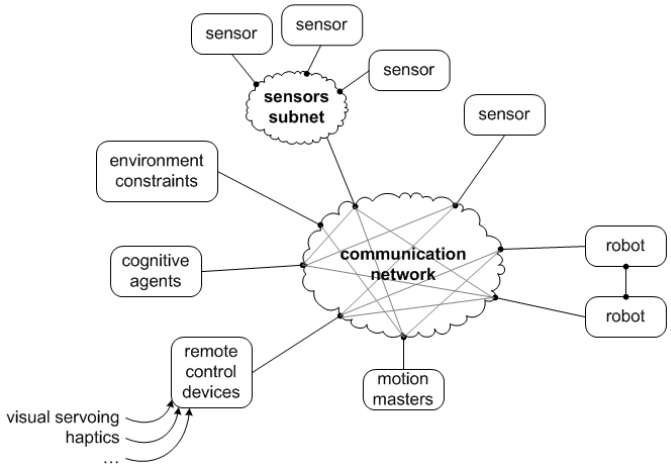


Figura 1.4 : Schema NCS rappresentante un generico sistema distribuito in cui più sensori, comandi remoti, dispositivi sono interconnessi

Una volta precisato che il formalismo per descrivere il sistema è valido in generale, si può passare ad uno schema specifico, di un particolare sistema multi robot come ad esempio un sistema per sincronizzare robot di saldatura guidati da

telecamere, oppure un sistema per la chirurgia come quello rappresentato nello schema di controllo distribuito della Figura 1.5, derivato dalla configurazione del progetto ACTIVE in corso presso il CNR-ITIA di cui si daranno alcuni cenni nel capitolo 7 (Sviluppi Futuri). In quest'ultimo caso il sistema sarà costituito da un robot di supporto per la testa di un paziente in operazioni di neurochirurgia in grado di misurarne le vibrazioni e da dei manipolatori che eseguiranno una compensazione di moto seguendo tali vibrazioni così che agli occhi del chirurgo che guida i robot mediante interfacce aptiche lo scenario appaia statico. Vi saranno poi altri dati provenienti da sensori e modelli, che però verranno accennati più approfonditamente solo nel capitolo degli Sviluppi Futuri.

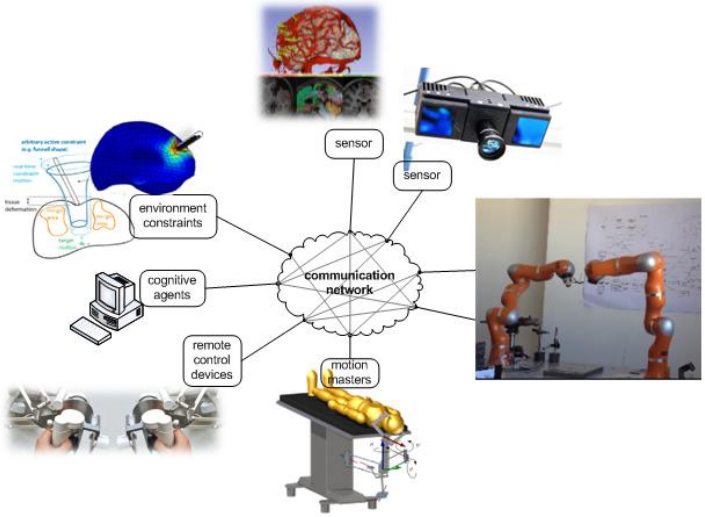


Figura 1.5 : Schema NCS di un sistema per la chirurgia cerebrale (ACTIVE frame)

Il sistema a cui si farà riferimento nello svolgimento del presente lavoro *è simile a* quello schematizzato nella Figura 1.5.

1.3.2 Architettura Considerata

Il sistema analizzato e su cui si eseguiranno simulazioni ed esperimenti sarà quindi *simile* a quello presentato in Figura 1.5 perché comunque vi saranno i due robot (KUKA LWR 4+), il concetto di *carrier* di cui si deve seguire, compensare il moto (in analogia con il robot per il sostegno della testa), il concetto di

comando aptico, la possibilità di inserimento di telecamere. Quello che si farà sarà quindi sfruttare il formalismo generale per coordinare i due bracci robot presenti in laboratorio in modo che seguano un generico *carrier* con funzioni di *motion master* (che potrebbe essere una catena di montaggio, una tavola rotante, etc.) e che combinino tale inseguimento con le informazioni che arriverebbero da comandi aptici e/o telecamere (come accadrebbe se sulla catena di montaggio i pezzi fossero disposti in modo non deterministico tale da richiedere una forma di visione artificiale). Nella seguente Figura 1.6 sono riportati i due KUKA LWR 4+ presenti in laboratorio, collegati tramite i loro controllori a un PC real-time comune che li possa coordinare.

Figura 1.6 : Setup di laboratorio: KUKA LWR4+ con relativi controllori e PC real-time

Prima di passare alla descrizione dei *componenti* principali del setup, si presenta l'architettura *component-based* con cui si andrà a controllare il sistema distribuito. L'architettura mostra ogni *componente* coinvolto. Per *componente* si può intendere un device hardware, ma anche ogni elemento omogeneo dell'applicazione, come un thread (parte di processo eseguibile in parallelo ad altri thread) che si occupa di elaborare dati o trasferirli mediante interacce. Sostanzialmente l'architettura utilizzata consiste in devices (i robot LWR ad esempio) che scambiano dati con dei *componenti* real-time dedicati a gestirne il flusso che a loro volta scambiano i dati con un *componente* (MROE) con il compito di eseguire gli algoritmi per portare alla coordinazione desiderata. Ovviamente il MROE dovrà scambiare i dati anche con ulteriori *componenti* che

conterranno le informazioni degli altri devices del sistema. Lo scambio dei dati è realizzato da *interfacce* (che possono essere memorie condivise o protocolli di rete, ad esempio protocolli UDP). Si dettaglierà maggiormente l'architettura software nel Capitolo 3.

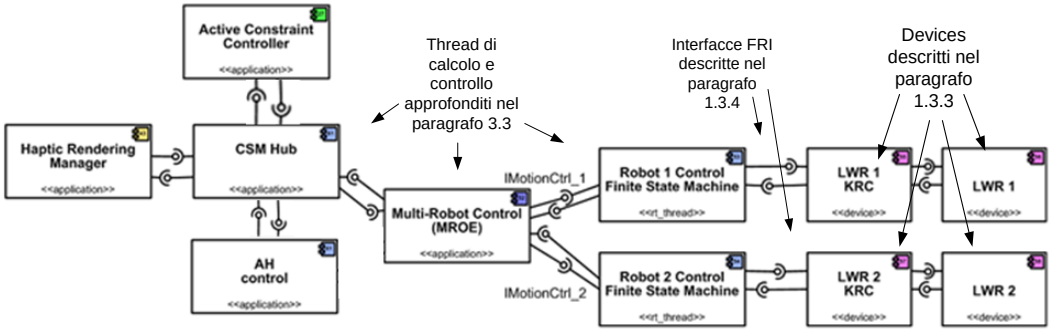


Figura 1.7 : Diagramma delle componenti e delle interfacce del programma di gestione del sistema.

1.3.3 KUKA LWR4+

Il Kuka LightWeight Robot (LWR), sviluppato presso l'Istituto di Robotica e Meccatronica del Centro Aerospaziale Tedesco (DLR) fa parte della nuova generazione di robot leggeri, con pesi ridotti al minimo (15 kg) che rendono possibile migliorare le prestazioni dinamiche con minor consumo di energia.

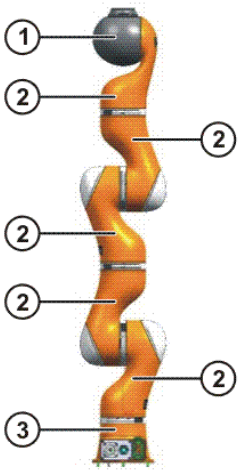


Figura 1.8 : Principali componenti del robot: (1) Polso "In Line"; (2) Moduli di giunto; (3) Base frame

Oltre a questo, le sue caratteristiche principali possono essere così classificate:

- E' dotato di sette assi e quindi ha sempre almeno un grado di ridondanza nello spazio tridimensionale. Questo permette al programmatore maggiore flessibilità di muoversi nel workspace, e in generale di adottare tecniche di gestione della ridondanza ad esempio per evitare meglio le singolarità che si hanno tipicamente su sistemi cinematici a sei assi. Nello specifico del problema considerato la ridondanza permetterà di migliorare le logiche di interazione tra i robot nel sistema.

- In ognuno dei sette assi sono posti sensori di coppia, che uniti a un dettagliato modello dinamico del robot, a servo controllori ai giunti (con banda di 3 kHz localmente al giunto, 1 kHz per il robot nel complesso, che sarà pertanto la frequenza di comunicazione tra robot e sistema remoto), a motori moderni e al già descritto peso contenuto, permettono controllo di forza e controllo attivo di vibrazioni molto efficienti [6].

- Sempre grazie agli elementi appena descritti è anche possibile ottenere una interazione programmata basata su controllo d'impedenza (cartesiano o ai giunti) tra robot e ambiente: il robot potrà comportarsi come un sistema molla-smorzatore i cui parametri si possono specificare. Una conseguenza della possibilità di interazione sarà la possibilità di compiere *manual guidance* del robot: il manipolatore potrà essere spostato manualmente e quindi in modo veloce e intuitivo alla posizione desiderata, o condotto in operazioni di assemblaggio prima assai più complesse da programmare.

- Un'altra caratteristica del robot è la possibilità di poter essere controllato da un calcolatore real-time su cui eseguire programmi in C++ scritti dall'utilizzatore. Questo è reso possibile dalla libreria F.R.I. di cui si accennerà tra poco. Questo aspetto, assieme a quello della ridondanza, saranno quelli maggiormente sfruttati in questo lavoro.

Per ulteriori dati sul LWR si rimanda al manuale [7], mentre ora si accenna anche al controllore del robot, in Figura 1.9 assieme al Kuka Control Panel (KCP).

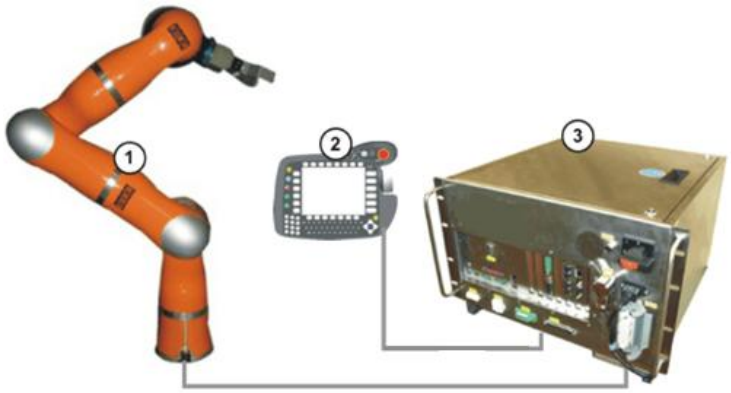


Figura 1.9 : (1) LWR; (2) KCP; (3) Controllore KR C2 lr

Parlando del controllore, si riportano le modalità secondo cui può operare:

- controllo in posizione;
- compensazione di gravità;
- controllo di impedenza nello spazio dei giunti;
- controllo di impedenza nello spazio operativo.

Tutte queste strategie di controllo sfruttano la misura delle coppie ai giunti per chiudere l'anello di controllo. Non ci si sofferma ulteriormente sull'argomento, rimandando alla Letteratura [8]. Per quanto riguarda invece altri dati sul controllore si possono trovare in [9].

1.3.4 Fast Research Interface

La Fast Research Interface (FRI [10]) è l'*interfaccia* che consente la comunicazione tra un PC remoto e il controllore del robot: grazie alla FRI è possibile inviare comandi al controllore o ricevere strutture dati contenenti

informazioni, ad esempio, sullo stato del robot, quali matrice di massa, Jacobiano, posizione del robot nello spazio di lavoro.

Per illustrare il funzionamento della FRI si fa riferimento alla seguente Figura 1.10, in cui è mostrata la possibilità di inviare comandi dal computer remoto al KRC Motion Kernel, che si occupa di gestire i comandi in arrivo dall'esterno sovrapponendoli a quelli propri del controllore ove questo sia ammissibile (ciò previene ovviamente di attuare comandi infattibili dal robot).

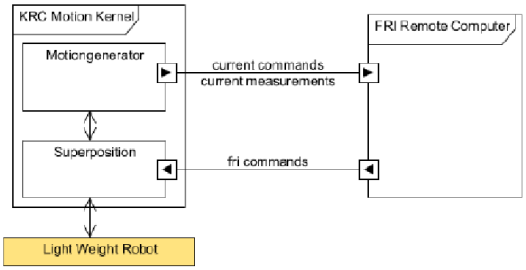


Figura 1.10 : Interazione tra FRI e Motion Kernel

Per i comandi disponibili nella libreria FRI e per i tipi di dati scambiabili si rimanda al manuale, mentre qui si accenna alla procedura con cui avviene la comunicazione tra il PC remoto il Motion Kernel del controllore del robot. Questa comunicazione avviene attraverso protocollo UDP ed è eseguibile in real-time. Se durante tale comunicazione si perdono pacchetti di dati o si ha eccessivo ritardo nella comunicazione non sarà possibile utilizzare tutte le modalità operative con cui la FRI consente di interfacciare il PC remoto al controllore. Queste modalità operative sono:

- Monitor mode: l'intera struttura dati può essere ottenuta lato PC remoto dal controllore ma non gli è consentito inviare comandi al controllore. Di conseguenza, il ciclo di lettura delle variabili non deve essere strettamente real-time.

- Command mode: al contrario dello stato operativo precedente è possibile inviare comandi dal PC remoto al controllore del robot. Questo richiede che il ciclo di lettura/scrittura sia strettamente real-time.

  Questo funzionamento dell'interfaccia FRI può essere modellato come una macchina a stati (Figura 1.11)

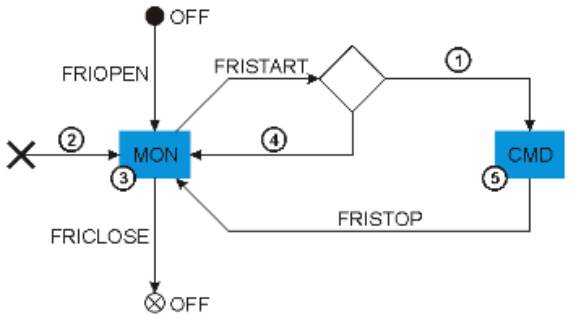


Figura 1.11 : Macchina a stati realizzata dalla FRI comprendente il Monitor mode e il Command mode

Attivata l'interfaccia, la macchina a stati realizzata dalla FRI entra in modalità Monitor mode e il controllore del robot inizia la spedizione di pacchetti di dati contenenti le variabili misurate al PC remoto. La modalità Command mode può essere attivata solo previa verifica della qualità della comunicazione tra PC remoto e controllore. Una volta attivata la modalità Command mode, se la comunicazione dovesse risultare di qualità non sufficiente, automaticamente la modalità verrà riportata a monitor mode, attivando un blocco di emergenza del robot se necessario.

Capitolo 2
Schema per la Coordinazione del Sistema Multi Robot

2.1 Motion Coordination: Matrici

Il primo obiettivo che ci si prefigge (vedi Paragrafo 1.2) è quello di realizzare un *framework* che consenta la gestione del sistema multi-robot in estrema versatilità rendendo possibili le azioni descritte nell'introduzione: coordinazione di due manipolatori in un sistema ove siano presenti anche variabili esterne e altri devices (come un *carrier* in moto che i manipolatori devono seguire per poi ad esempio prelevarne degli oggetti da manipolare).

Un *framework* adatto può essere il *Task Frame Formalism* [4], che consiste nella rappresentazione cinematica delle relazioni di posizione (estendibili all'atto di moto) tra vari punti dei devices coinvolti. Questa rappresentazione è inclusa nel modulo di controllo coordinato ed è riferita, in particolare, alla presenza tra i moduli di un generico NCS di:

- Un riferimento master (*carrier*), che come accennato è l'oggetto il cui moto deve essere seguito dai manipolatori . Su di esso è possibile prevedere un ulteriore moto relativo.
- Questo ulteriore moto relativo (*target*) viene indicato, come nella maggior parte dei task avviene, attraverso una telecamera che funge da analizzatore di moto relativo. Se questo moto è presente, le estremità dei manipolatori dovranno inseguire appunto il moto del *target.*

- Un input esterno (*ref*) diretto a ciascun singolo robot per farlo muovere indipendentemente dal *target.*

Specificato ciò, si passa a descrivere lo schema concepito e le relative catene cinematiche rappresentate nel loro complesso nella Figura 2.1.

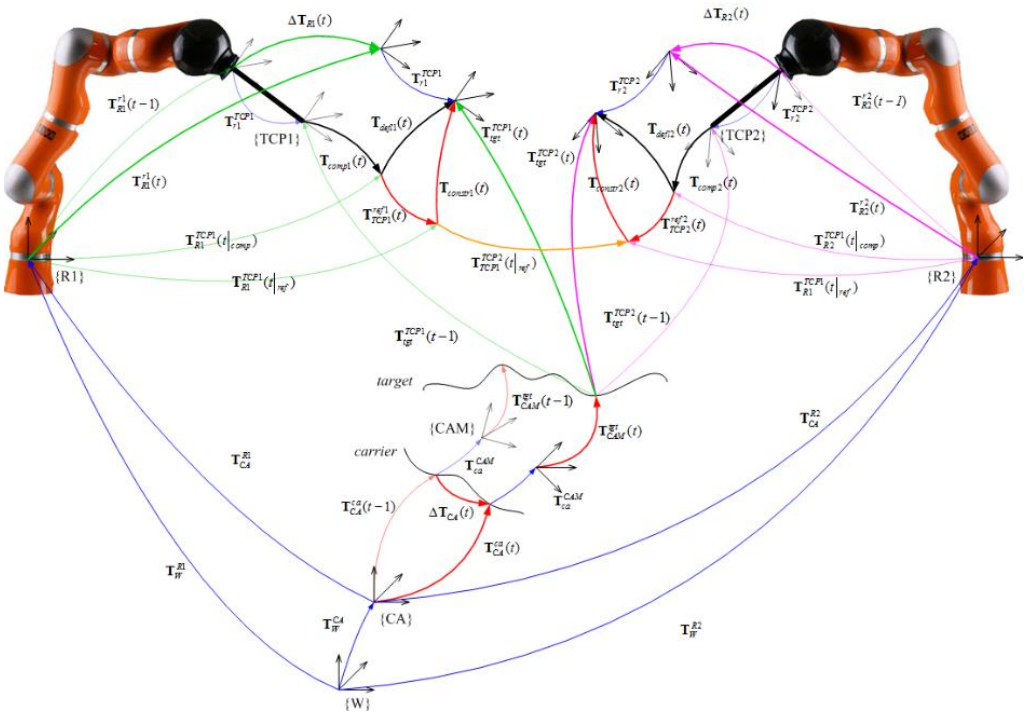


Figura 2.1 : Catene cinematiche per il framework di motion coordination del sistema

Ogni trasformazione presente nella precedente Figura 2.1 è descritta mediante una matrice omogenea $\boldsymbol{T}$, rappresentazione compatta della mutazione di coordinate tra due terne cartesiane A e B. Essa contiene nel vettore $[x, y, z]^T$ l'informazione univoca sulla distanza tra le origini degli assi e nella sottomatrice **R** l'informazione univoca sulla rotazione relativa tra le due terne. Essendo **R** una matrice $3x3$ di rotazione, infatti, non c'è dipendenza da scelte convenzionali che si avrebbero invece se si decidesse di dare una rappresentazione minima della rotazione, ossia se si assegnasse un set di tre angoli definiti secondo varie convenzioni possibili, ad esempio quella di Eulero o quella di Cardano. (Si è utilizzata una rappresentazione minima solo dove strettamente necessario, e in quei casi si è generalmente scelta quella di Cardano per allinearsi con le convenzioni del LWR: vedi Paragrafo 3.1.2).

Calcolando tali matrici di trasformazione istante per istante si caratterizzerà compiutamente il problema. Questo calcolo si potrà operare secondo molteplici catene cinematiche chiuse consentendo una flessibile gestione delle matrici rispetto alla funzione di input o di output del calcolo e ottenendo combinazioni di task differenti. Tra queste matrici si potranno distinguere alcune tipologie:

- Matrici di Calibrazione: descrivono il passaggio da una terna A ad una terna B e rimangono costanti durante l'esecuzione di un task (ad esempio definiscono il passaggio dalla terna di riferimento assoluta W a quella posta alla base dei robot).
- Matrici di Input: matrici di passaggio da una terna A ad una terna B, con B che rototrasla relativamente ad A nel tempo e che costituisce un input del problema.
- Matrici tempo-varianti "calcolate": se posizioni e/o orientamenti di A o B variano nel tempo in seguito all'intervento dell'algoritmo di coordinazione.
- Matrici di Output: se B rappresenta la terna collocata all'end effector si ha la matrice di output del sistema, che serve per inviare un comando di moto al robot.

Matrice	*Tipologia*	*Descrizione*
$\boldsymbol{T}_{W}^{CA}$	Calibrazione	Trasformazione dal *World Coordinate System* alla terna di riferimento per il moto del *carrier*
$\boldsymbol{T}_{W}^{R1}$	Calibrazione	Trasformazione dal *World Coordinate System* alla terna base del robot 1
$\boldsymbol{T}_{W}^{R2}$	Calibrazione	Trasformazione dal *World Coordinate System* alla terna base del robot 2
$\boldsymbol{T}_{r1}^{TCP1}$	Calibrazione	Trasformazione dalla terna all'end effector alla terna all'estremità dell'utensile del robot 1
$\boldsymbol{T}_{r2}^{TCP2}$	Calibrazione	Trasformazione dalla terna all'end effector alla terna all'estremità dell'utensile del robot 2
$\boldsymbol{T}_{ca}^{CAM}$	Calibrazione	Trasformazione da una terna di riferimento alla terna dove è fissata la telecamera

Tabella 2.1 : Matrici di Calibrazione

Matrice	*Tipologia*	*Descrizione*
$\boldsymbol{T}_{CA}^{ca}(t)$	Input	Trasformazione dalla terna di riferimento per il moto del *carrier* alla terna posta sul punto in movimento
$\boldsymbol{T}_{CAM}^{tgt}(t)$	Input	Trasformazione dalla terna di base della telecamera alla terna centrata sul suo *target*
$\boldsymbol{T}_{TCP1}^{ref1}(t)$	Input	Trasformazione imposta all'estremità utensile del robot 1 da un comando esterno
$\boldsymbol{T}_{TCP2}^{ref2}(t)$	Input	Trasformazione imposta all'estremità utensile del robot 2 da un comando esterno
$\boldsymbol{T}_{constr1}(t)$	Input	Trasformazione dovuta a un vincolo sulla traiettoria dell'estremità utensile del robot 1
$\boldsymbol{T}_{constr2}(t)$	Input	Trasformazione dovuta a un vincolo sulla traiettoria dell'estremità utensile del robot 2

Tabella 2.2 : Matrici in Input

Matrice	*Tipologia*	*Descrizione*
$\boldsymbol{T}_{comp1}(t)$	Calcolata	Trasformazione di compensazione di moto sul robot 1, calcolata per seguire il moto del *target*
$\boldsymbol{T}_{comp2}(t)$	Calcolata	Trasformazione di compensazione di moto sul robot 2, calcolata per seguire il moto del *target*.
$\boldsymbol{T}_{defl1}(t)$	Calcolata	Trasformazione di deflessione sul robot 1, calcolata componendo $\boldsymbol{T}_{TCP1}^{ref1}(t)$ e $\boldsymbol{T}_{constr1}(t)$
$\boldsymbol{T}_{defl2}(t)$	Calcolata	Trasformazione di deflessione sul robot 2, calcolata componendo $\boldsymbol{T}_{TCP2}^{ref2}(t)$ e $\boldsymbol{T}_{constr2}(t)$
$\boldsymbol{T}_{tgt}^{TCP1}(t)$	Calcolata	Trasformazione che descrive la posa del TCP1 rispetto alla terna del *target*
$\boldsymbol{T}_{tgt}^{TCP2}(t)$	Calcolata	Trasformazione che descrive la posa del TCP2 rispetto alla terna del *target*
$\boldsymbol{T}_{TCP1}^{TCP2}(t)$	Calcolata	Trasformazione che lega i TCP dei due robot.

Tabella 2.3 : Matrici calcolate dall'algoritmo di coordinazione

Matrice	*Tipologia*	*Descrizione*
$\boldsymbol{T}_{r1}^{R1}(t)$	Output	Trasformazione che lega la terna alla base del robot 1 con quella al suo end effector
$\boldsymbol{T}_{r2}^{R2}(t)$	Output	Trasformazione che lega la terna alla base del robot 2 con quella al suo end effector

Tabella 2.4 : Matrici di Output

Si passa ora a dettagliare maggiormente le trasformazioni qui introdotte che in ogni istante dovranno essere aggiornate nell'algoritmo di coordinazione per avere gli output desiderati.

2.1.1 Matrici di Calibrazione

Una volta definita una terna destrorsa di riferimento assoluto per l'intero sistema (terna con origine nel punto *W* di Figura 2.1, generalmente detta *World Coordinate System*), si potranno definire le posizioni delle basi dei due robot rispetto a tale sistema di riferimento assoluto mediante le matrici omogenee di trasformazione $\boldsymbol{T}_{W}^{R1}$ e $\boldsymbol{T}_{W}^{R2}$, come esemplificato nella Figura 2.2.

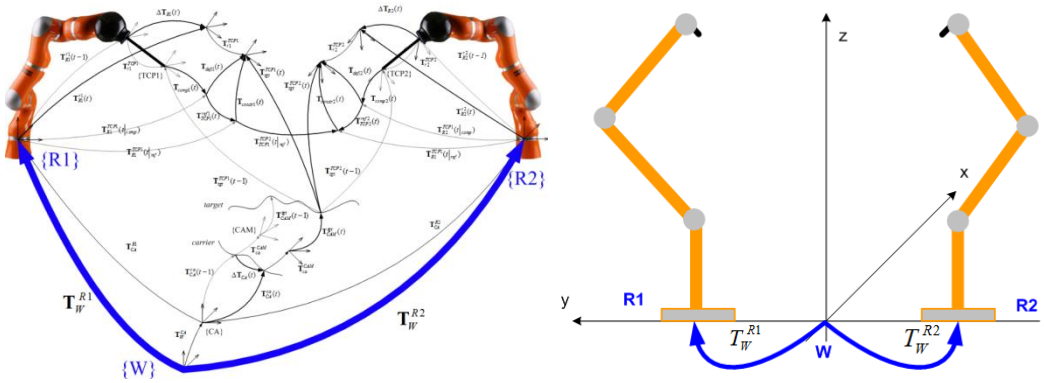


Figura 2.2 : Definizione delle basi dei robot rispetto al sistema di riferimento assoluto del sistema. La convenzione degli assi è quella assunta in ITIA, secondo il montaggio dei LWR

Una volta definita la posizione delle basi rispetto al sistema di riferimento assoluto del sistema è anche possibile memorizzare la matrice di passaggio tra una base e l'altra dei

due robot come: $\boldsymbol{T}_{R1}^{R2} = inv(\boldsymbol{T}_{W}^{R1}) \cdot \boldsymbol{T}_{W}^{R2}$ e analogamente $\boldsymbol{T}_{R2}^{R1} = inv(\boldsymbol{T}_{W}^{R2}) \cdot \boldsymbol{T}_{W}^{R1}$. Questo permetterà di risparmiarsi delle inutili inversioni quando servirà chiudere la catena cinematica a ogni istante dell'esecuzione in tempo reale del moto coordinato.

Il passo successivo consiste nel determinare il sistema di riferimento posto all'origine del moto dell'oggetto che dovrà essere seguito (*carrier*). Tale sistema di riferimento è definito rispetto al sistema di coordinate assolute mediante la matrice omogenea: $\boldsymbol{T}_{W}^{CA}$. La Figura 2.3 illustra il passaggio in questione:

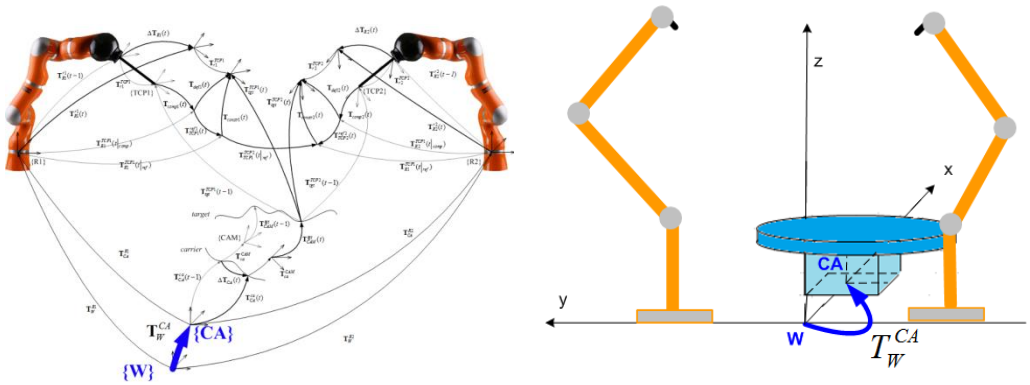


Figura 2.3 : Definizione della base del carrier (i.e. tavola rotante) rispetto al sistema di riferimento assoluto del sistema.

Da W si potrà anche calcolare: $\boldsymbol{T}_{CA}^{R1} = inv(\boldsymbol{T}_{W}^{CA}) \cdot \boldsymbol{T}_{W}^{R1}$ e $\boldsymbol{T}_{CA}^{R2} = inv(\boldsymbol{T}_{W}^{CA}) \cdot \boldsymbol{T}_{W}^{R2}$.

Si deve poi considerare che quelli che nelle figure sono chiamati TCP_i sono le terne alle estremità dei tools fissati sui robot (Tool Centre Point). Il legame tra *end effectors* (EE_i) e TCP_i è dato dalle matrici in genere costanti $\boldsymbol{T}_{r,i}^{TCP,i}$.

Infine se è presente una telecamera fissa rispetto ad un'altra terna, chiaramente anche la trasformazione che ne descriverà la posa rispetto a un sistema di riferimento su cui è fissata sarà costante (Fig. 2.4).

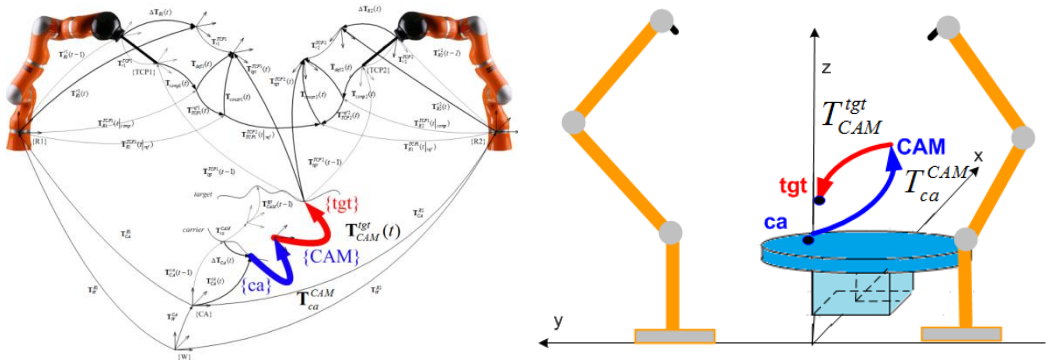


Figura 2.4 : Definizione del posizionamento di una telecamera rispetto a un punto noto e del target finale rispetto alla telecamera

2.1.2 Matrici di Input

Nella precedente Figura 2.4 si è visto che nel framework si può prevedere una possibilità che lo renda ancora maggiormente flessibile: l'introduzione di una telecamera, con i relativi vantaggi che ne possono conseguire. Esempi possono essere quelli classici della visione, come l'identificare l'esatta posizione di oggetti disposti sulla tavola rotante per effettuarne un successivo afferraggio, oppure sempre tornando all'esempio chirurgico, monitorare i moti di pulsazione del cervello che ovviamente non saranno misurabili dal supporto mobile della testa. Una volta definito il posizionamento della telecamera rispetto a un sistema di riferimento noto, la telecamera fornirà il posizionamento del *target* finale rispetto alla sua posizione mediante dati organizzabili in una matrice $\boldsymbol{T}_{CAM}^{tgt}(t)$ che verranno passati all'algoritmo di coordinazione che andrà ad utilizzarli. Si noti che se non è previsto l'uso di una telecamera, basterà porre $\boldsymbol{T}_{CAM}^{tgt}(t) = \boldsymbol{T}_{ca}^{CAM}(t) = \boldsymbol{I}$.

Oltre a quello eventualmente rilevato dalla telecamera, vi è il moto del sistema di riferimento collocato su una estremità dell'oggetto avente base in CA, ossia si considera di avere un *carrier*. Questo moto potrà essere una normale traiettoria imposta (ad esempio, rifacendosi alle figure, il moto di un punto su una tavola rotante) o comunque un moto misurato. In ogni caso quello centrato in ca sarà un sistema di riferimento rototraslante. La rappresentazione di $\boldsymbol{T}_{CA}^{ca}(t)$ si ha nella seguente Figura 2.5.

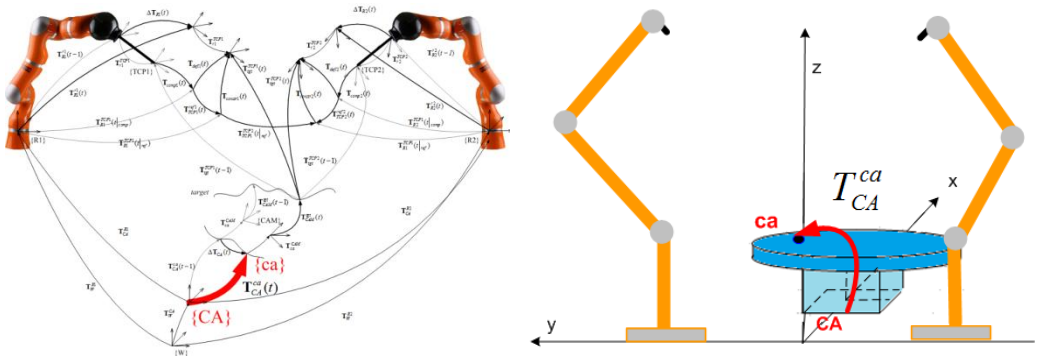


Figura 2.5 : Definizione di un punto mobile (i.e. punto su tavola rotante) rispetto al sistema di riferimento alla base del sistema mobile stesso.

Si procede ora a definire la possibilità di agire su un robot direttamente assegnando una trasformazione di riferimento $\boldsymbol{T}_{ref}(t)$ che è in generale differente per i due robot ($\boldsymbol{T}_{ref1}(t)$, $\boldsymbol{T}_{ref2}(t)$). Un esempio di tale input può essere il comando di avvicinarsi al *target* fino a quel momento seguito per magari afferrarlo, oppure un comando derivante da interfacce aptiche. Sempre agenti sui singoli robot possono essere movimenti di vincolo, imposti ad esempio dal software per evitare collisioni tra gli end effector o tra il singolo end effector e altre parti del sistema o ostacoli esterni. Si hanno così le trasformazioni omogenee $\boldsymbol{T}_{constr1}(t)$ e $\boldsymbol{T}_{constr2}(t)$. A un determinato istante, quindi, al TCP i-esimo potrà giungere un comando di riferimento che tenderà a spostarlo secondo la $\boldsymbol{T}_{ref,i}(\bar{t})$ desiderata, cui si sommerà un possibile comando di vincolo volto ad impedire scontri indesiderati all'istante successivo $\boldsymbol{T}_{constr,i}(\bar{t})$. I due contributi saranno componibili portando all'ottenimento della trasformazione risultante calcolabile come $\boldsymbol{T}_{defl,i}(t) = \boldsymbol{T}_{ref,i}(t) \cdot \boldsymbol{T}_{constr,i}(t)$.

2.1.3 Matrici Calcolate in Linea

Si riportano qui anche le altre matrici che verranno determinate chiudendo le adeguate catene cinematiche ad ogni iterazione del ciclo.

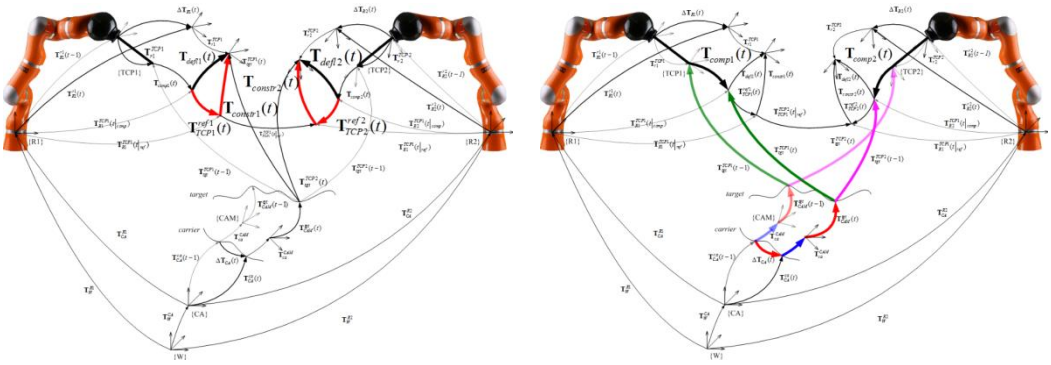


Figura 2.6 : Catene cinematiche per la definizione di $T_{defl,i}$ e $T_{compl,i}$

Per introdurre $\boldsymbol{T}_{comp,i}(t)$ si ricordi che il moto dell'estremità mobile del *carrier* $\boldsymbol{T}_{CA}^{ca}(t)$ e quello del *target* rilevato dalla telecamera $\boldsymbol{T}_{CAM}^{tgt}(t)$ sono dipendenti dal tempo: ciò significa che a due istanti $\bar{t}$ e $(\bar{t}-1)$ corrispondono due trasformazioni $\boldsymbol{T}_{CA}^{ca}(\bar{t})$ e $\boldsymbol{T}_{CA}^{ca}(\bar{t}-1)$ legate dalla relazione: $\boldsymbol{T}_{CA}^{ca}(\bar{t}) = \boldsymbol{T}_{CA}^{ca}(\bar{t}-1)\cdot\Delta\boldsymbol{T}_{ca}(\bar{t})$. Discorso analogo vale per la trasformazione $\boldsymbol{T}_{CAM}^{tgt}(t)$ per cui $\boldsymbol{T}_{CAM}^{tgt}(\bar{t}) = \boldsymbol{T}_{CAM}^{tgt}(\bar{t}-1)\cdot\Delta\boldsymbol{T}_{tgt}(\bar{t})$.

E' immediato dedurre che se si vuole che i TCP seguano tali variazioni temporali, essi dovranno compiere un moto di compensazione descritto da una trasformazione $\boldsymbol{T}_{comp,i}(t)$. Nel caso quindi si supponga di non avere moti di deflessione dovuti a riferimenti da seguire sul singolo robot o a imposizioni di vincoli ($\boldsymbol{T}_{defl,i}(t) = \boldsymbol{T}_{ref,i}(t) = \boldsymbol{T}_{constr,i}(t) = \boldsymbol{I} \;\Rightarrow\; \boldsymbol{T}_{tgt}^{TCP,i} = cost, \forall t$) e non vi fossero moti rilevati da telecamere ($\boldsymbol{T}_{CAM}^{tgt}(t) = cost, \forall t$) , la compensazione ad un dato istante sarebbe calcolabile semplicemente come:

$$\boldsymbol{T}_{compl,i}(\bar{t}) = inv\left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t}-1)\right)\cdot\left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t})\right) =$$

$$= inv\left(\boldsymbol{T}_{tgt}^{TCP,i}(\bar{t}-1)\right)\cdot inv\left(\boldsymbol{T}_{CAM}^{tgt}(\bar{t}-1)\right)\cdot\left(\Delta\boldsymbol{T}_{ca}(\bar{t})\right)\cdot\left(\boldsymbol{T}_{CAM}^{tgt}(\bar{t})\right)\cdot\left(\boldsymbol{T}_{tgt}^{TCP,i}(\bar{t})\right) =$$

$$= \Delta\boldsymbol{T}_{ca}(\bar{t})$$

Che è il risultato intuitivo che ci si aspetta: se le estremità degli utensili devono inseguire un punto che si sposta di $\Delta\boldsymbol{T}_{ca}(\bar{t})$ e tutte le altre variabili che possono

cambiare si mantengono costanti, anche tali estremità saranno caratterizzate da una trasformazione $\Delta \boldsymbol{T}_{ca}(\bar{t})$. Con questo esempio si è mostrato il significato del termine di compensazione $\boldsymbol{T}_{compl,i}(t)$ e si è introdotto il metodo con cui si possono utilizzare le catene cinematiche.

Si passa ora a definire il legame tra il *target* e le estremità degli utensili (TCP). In tal senso vi possono essere più strategie di definizione possibili, ma le principali sono generalmente quella *Leader – Follower* (o *Master – Slave* in cui si definisce il posizionamento del braccio *Leader* rispetto al *target* e quello del braccio *Follower* rispetto al *Leader*) o quella *Task Space Oriented* (in cui è data uguale importanza ai due robot che eseguono un task e ciascuno quindi assume una propria posizione rispetto al *target* indipendentemente dal comportamento dell'altro braccio) [1].

L'approccio *Leader – Follower* si può ritrovare definendo inizialmente la postura del primo braccio (ad esempio in coordinate polari) e derivandone quella del secondo braccio o mediante l'imposizione diretta di una $\boldsymbol{T}_{TCP1}^{TCP2}$ oppure sfruttando delle particolari condizioni geometriche rispetto al *target* che possono risultare assai più comode, come ad esempio imponendo che il secondo TCP si disponga rispetto al primo mediante una simmetria polare o una simmetria assiale. Questi casi sono riassunti nelle seguenti figure: nella Figura 2.7 si definisce in coordinate sferiche la posa del primo robot rispetto al *target* (trasformazione $\boldsymbol{T}_{tgt}^{TCP1}$); nella Figura 2.8 si definisce sempre in coordinate sferiche la posa del secondo robot indipendentemente dal primo e si persegue pertanto l'approccio *Task Space Oriented* (si decide $\boldsymbol{T}_{tgt}^{TCP2}$ e $\boldsymbol{T}_{TCP1}^{TCP2}$ viene calcolato di conseguenza); nella Figura 2.9 si definisce invece la posa del secondo robot direttamente rispetto al primo oppure sfruttando simmetria polare rispetto al *target* o assiale rispetto a una retta passante per esso seguendo un'impostazione *Leader – Follower*. In entrambi i casi si arriva comunque alla definizione di $\boldsymbol{T}_{tgt}^{TCP2}$ e $\boldsymbol{T}_{TCP1}^{TCP2}$ per l'istante iniziale.

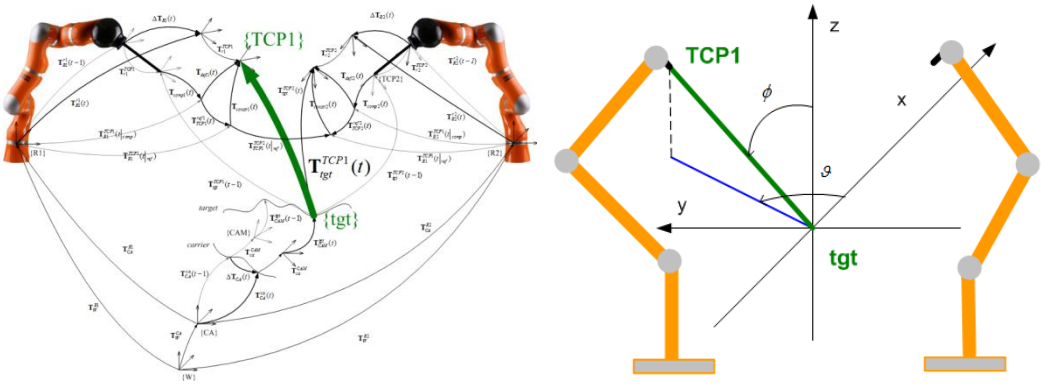


Figura 2.7 : : Definizione del posizionamento del TCP1 rispetto al target in coordinate sferiche.

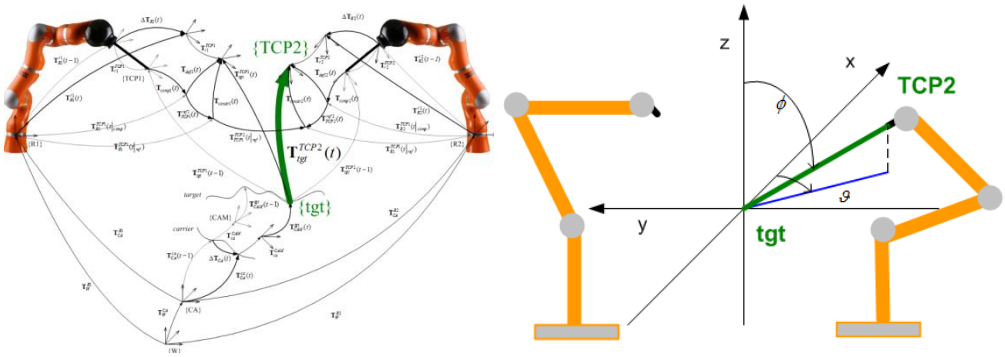


Figura 2.8 : Definizione del posizionamento del TCP2 rispetto al target in coordinate sferiche, indipendentemente dal braccio 1

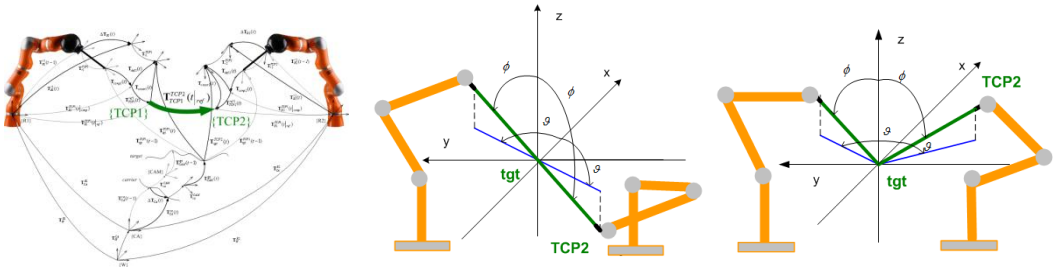


Figura 2.9 : Definizione del posizionamento del TCP2 rispetto al TCP1: esempi di simmetria polare e assiale, espresse in coordinate sferiche

Si sottolinea che l'approccio per definire il posizionamento iniziale non sarà necessariamente quello seguito durante il task, ossia si può decidere di posizionare il secondo robot con una postura iniziale in simmetria rispetto a quella del primo e poi però muoverlo indipendentemente da esso durante il task.

2.1.4 Matrici di Output

Infine si hanno le matrici $\boldsymbol{T}_{r1}^{R1}(t)$ e $\boldsymbol{T}_{r2}^{R2}(t)$ che specificano il movimento di ciascun robot rispetto alla propria terna base, ossia quello che serve per impartire comandi e controllare il robot nello spazio cartesiano. Quindi sarà questo il valore che si dovrà calcolare con l'algoritmo presentato al prossimo paragrafo.

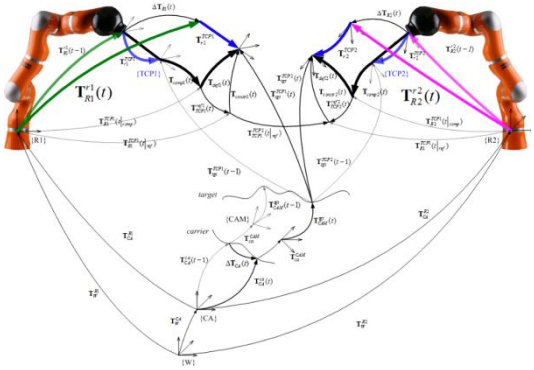


Figura 2.10 : Catene cinematiche per il calcolo delle matrici $\boldsymbol{T}_{R,i}^{r,i}$

2.2 Motion Coordination: Algoritmo

Si sono presentati quindi tutti gli elementi che permetteranno di chiudere le catene cinematiche per definire l'evoluzione temporale del legame $\boldsymbol{T}_{r,i}^{R,i}(t)$ tra l'*end effector* dell'i-esimo robot e la sua base. Tali elementi verranno pertanto utilizzati da un algoritmo di coordinazione che seguirà i seguenti passi:

- A partire dalla moto rilevato istante per istante di un *target*, esprimibile come composizione di un moto del *carrier* e di un moto rilevato da una telecamera secondo la relazione $\boldsymbol{T}_{CA}^{tgt}(t) = \boldsymbol{T}_{CA}^{ca}(t) \cdot \boldsymbol{T}_{ca}^{CAM} \cdot \boldsymbol{T}_{CAM}^{tgt}(t)$, eseguire una *Motion Compensation*, ossia la compensazione del moto del *target* mantenendo inalterato il posizionamento tra TCP e il *target* stesso. Ciò significa dire che $\boldsymbol{T}_{tgt}^{TCP,i} = cost \ \forall t$, da cui:

$$\boldsymbol{T}_{CA}^{TCP,i}(t) = \boldsymbol{T}_{CA}^{tgt}(t) \cdot \boldsymbol{T}_{tgt}^{TCP,i}$$
$$\boldsymbol{T}_{R,i}^{r,i}(t) = inv\left(\boldsymbol{T}_{CA}^{R,i}\right) \cdot \boldsymbol{T}_{CA}^{TCP,i}(t) \cdot inv(\boldsymbol{T}_{r,i}^{TCP,i})$$

Come risultato ad ogni istante avrò in output la matrice che descrive la postura del robot, che sarà mutata di una quantità $\Delta\boldsymbol{T}_{R,i}(\bar{t})$ rispetto a quella all'istante precedente $\boldsymbol{T}_{Ri}^{ri}(\bar{t}-1)$. Ovviamente $\Delta\boldsymbol{T}_{R,i}(\bar{t})$ sarà equivalente a $\boldsymbol{T}_{comp,i}(\bar{t})$ dato che :

$$\Delta\boldsymbol{T}_{R,i}(\bar{t}) = inv\left(\boldsymbol{T}_{R,i}^{r,i}(\bar{t}-1)\right) \cdot \boldsymbol{T}_{R,i}^{r,i}(\bar{t}) =$$
$$= inv\left(\left(\boldsymbol{T}_{CA}^{R,i}\right)^{-1}\left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t}-1)\right)\left(\boldsymbol{T}_{r,i}^{R,i}\right)^{-1}\right) \cdot \left(\left(\boldsymbol{T}_{CA}^{R,i}\right)^{-1}\left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t})\right)\left(\boldsymbol{T}_{r,i}^{R,i}\right)^{-1}\right) =$$
$$= inv\left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t}-1)\right) \cdot \left(\boldsymbol{T}_{CA}^{TCP,i}(\bar{t})\right) = \boldsymbol{T}_{comp,i}(\bar{t})$$

- Nel caso in cui in ci sia anche una componente di deflessione l'algoritmo deve tenerne conto e quindi ad ogni istante non solo si compenserà il moto come fatto nel caso precedente, ma a tale termine si aggiungerà la trasformazione di deflessione stessa e si procederà ad aggiornare la postura del TCP rispetto al *target* che è cambiata appunto in seguito all'azione di quel termine. In altre parole ora $\boldsymbol{T}_{tgt}^{TCP,i} \neq cost\ \forall t.$ Quindi all'effetto di compensazione (designato con $|_{comp}$)

$$\boldsymbol{T}_{R,i}^{TCP,i}\left(\bar{t}|_{comp}\right) = \boldsymbol{T}_{R,i}^{TCP,i}(\bar{t}-1) \cdot \left(\boldsymbol{T}_{comp,i}(\bar{t})\right) =$$
$$= \boldsymbol{T}_{R,i}^{TCP,i}(\bar{t}-1)\left[\boldsymbol{T}_{CA}^{TCP,i}(\bar{t}-1)\right]^{-1}\left[\boldsymbol{T}_{CA}^{TCP,i}(\bar{t})\right]$$

si dovrà aggiungere anche il contributo di deflessione e aggiornare la configurazione del TCP per gli istanti successivi. Si ha quindi:

$$\boldsymbol{T}_{TCP,i}^{ref,i}(\bar{t}) = \left[\boldsymbol{T}_{R,i}^{TCP,i}(t-1)\right]^{-1}\boldsymbol{T}_{R,i}^{ref,i}(t)$$

Da cui il termine di inseguimento (designato con $|_{\mathrm{ref}}$) risulta:

$$\boldsymbol{T}_{R,i}^{TCP,i}(t|ref) = \boldsymbol{T}_{R,i}^{TCP,i}\left(t|_{comp}\right) \cdot \boldsymbol{T}_{R,i}^{ref,i}(t)$$

Si attiva a questo punto il controllo della posa che andrebbe ad assumere il TCP in seguito a tale atto di moto e subentra il confronto con i vincoli sulle posizioni reciproche tra i due robot e gli altri oggetti del sistema generando un possibile moto di deflessione totale dato da:

$$\boldsymbol{T}_{constr,i}(t) = f\left(\boldsymbol{T}_{R,i}^{TCP,i}\left(t|_{ref}\right), \boldsymbol{T}_{R,j}^{TCP,j}\left(t|_{ref}\right), f_{AC}(t)\right)$$
$$\boldsymbol{T}_{defl,i}(t) = \boldsymbol{T}_{R,i}^{TCP,i}\left(t|_{ref}\right) \cdot \boldsymbol{T}_{constr,i}(t)$$

La posa aggiornata del TCP e la configurazione aggiornata del TCP rispetto al target si otterranno secondo le seguenti relazioni:

$$\boldsymbol{T}_{R,i}^{TCP,i}(t) = \boldsymbol{T}_{R,i}^{TCP,i}\left(t|_{ref}\right) \cdot \boldsymbol{T}_{defl,i}(t)$$
$$\boldsymbol{T}_{Ri}^{r,i}(t) = \boldsymbol{T}_{Ri}^{TCPi}(t)\left[\boldsymbol{T}_{Ri}^{TCPi}\right]^{-1}$$
$$\boldsymbol{T}_{tgt}^{TCP,i}(t) = \left[\boldsymbol{T}_{CA}^{tgt}(t)\right]^{-1}\boldsymbol{T}_{CA}^{R,i} \cdot \boldsymbol{T}_{Ri}^{TCP,i}(t)$$

- Come è stato generato il riferimento per il robot $i = 1$, si può procedere anche con il robot $i = 2$. Questo potrà avvenire secondo le due strategie già prima presentate parlando del posizionamento iniziale. Se si utilizza una strategia "*Task Space Oriented"*, entrambi i robot seguono appunto un riferimento nel *Task Space*. Quindi mantengono il posizionamento relativo inalterato solo finché non vi sono moti di deflessione agenti in modo diverso sull'uno o sull'altro robot. In tal caso per ottenere il riferimento da passare al secondo robot basta ripetere quanto fatto col primo ponendo $i = 2$. Quindi:

$$\boldsymbol{T}_{R2}^{TCP2}(t) = \boldsymbol{T}_{R2}^{TCP2}\left(t|_{ref}\right) \cdot \boldsymbol{T}_{defl2}(t)$$
$$\boldsymbol{T}_{R2}^{r2}(t) = \boldsymbol{T}_{R2}^{TCP2}(t)[\boldsymbol{T}_{R2}^{TCP2}]^{-1}$$

$$\boldsymbol{T}_{tgt}^{TCP2}(t) = \left[\boldsymbol{T}_{CA}^{tgt}(t)\right]^{-1}\boldsymbol{T}_{CA}^{R2} \cdot \boldsymbol{T}_{R2}^{TCP2}(t)$$

Un esempio è quello in cui entrambi i robot sono "agganciati" ad un *target* virtuale il cui moto periodico (ad esempio lungo l'asse x) è stato simulato con una serie di Fourier ($\boldsymbol{T}_{CA}^{ca}\ (x_i, 4) \cong \sum_{k=1}^{n} \frac{X_{0i}}{k^2}\cos(k\omega t + \varphi(t))$

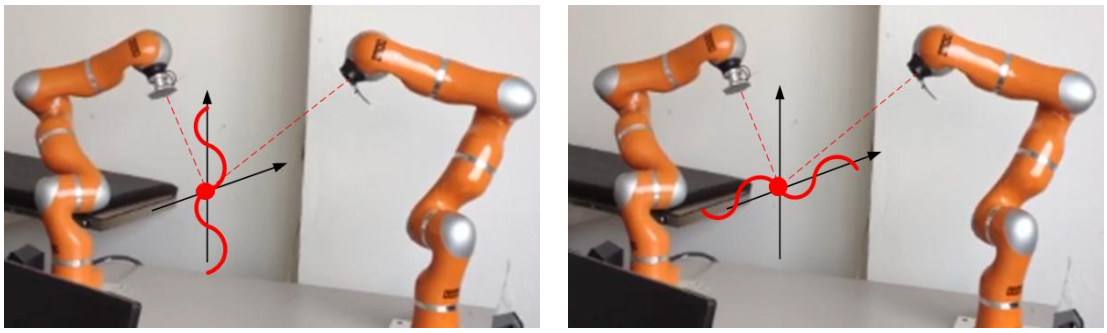


Figura 2.11 : Esempio di strategia Task Space Oriented in una applicazione di compensazione del moto periodico di un target.

Se si segue invece la strategia *Leader – Follower*, si manterrà sempre la posizione reciproca definita. $\mathbf{T}_{\mathrm{TCP1}}^{\mathrm{TCP2}} = \mathrm{cost.}$ Quindi per ottenere $\mathbf{T}_{\mathrm{R2}}^{\mathrm{r2}}(\mathrm{t})$ si potrà operare semplicemente secondo la relazione:

$\boldsymbol{T}_{R2}^{r2}(t) = \boldsymbol{T}_{R2}^{r2}(t-1) \cdot \Delta\boldsymbol{T}_{R1}$, essendo in questo caso $\Delta\boldsymbol{T}_{R1} = \Delta\boldsymbol{T}_{R2}$.

Quindi Nonostante i moti di deflessione che si possono riscontrare su un braccio, la posizione reciproca non cambia. Un esempio è mostrato in Figura 2.12, che riporta un'applicazione per cui si trascura di avere la possibilità di inseguire un *target* (di fatto si pongono tutti gli elementi che portano da W al *target* costanti) e si impone solo un moto di riferimento al primo robot (direttamente sfruttando la possibilità del KUKA LWR4+ di ammettere *manual guidance*) che di conseguenza viene a coincidere con $\Delta\boldsymbol{T}_{R1}$. Il secondo robot ha il compito di mantenere sempre la stessa postura rispetto al primo e un laser misura la bontà della riuscita di tale operazione.

Figura 2.12 : Esempio di strategia Leader (Skull) – Follower (Laser)

Infine si deve specificare che è possibile assegnare anche strategie di interazione ibride (e anche più complesse) che le due esposte finora. Una chiara possibilità è che si desideri far muovere in secondo robot esattamente come il primo *a meno che* gli sia ordinato diversamente. Questo equivale a dire di eseguire una strategia *Leader – Follower* senza imporre implicitamente $\boldsymbol{T}_{defl2} = \boldsymbol{I}$ come era stato fatto parlando della stessa, ed è una cosa ragionevole da fare per esempio perché se si deve evitare un urto dell'end effector si avrà $\boldsymbol{T}_{constr2} \neq \boldsymbol{I} \Longrightarrow \boldsymbol{T}_{defl2} \neq \boldsymbol{I}$. Per rispettare ciò, al fine di ottenere l'andamento di $\boldsymbol{T}_{R2}^{r2}(t)$ basterà scrivere: $\boldsymbol{T}_{R2}^{r2}(t) = \boldsymbol{T}_{R2}^{r2}(t-1) \cdot \Delta\boldsymbol{T}_{R1}(t) \cdot \boldsymbol{T}_{defl2}(t)$.

Altre possibilità più complesse possono essere quelle sperimentate da DLR riassunte dalla Figura 2.13: è possibile modellare i legami mutui tra robot e tra robot e ambiente come appunto sistemi molla-smorzatore, ossia passando da relazioni cinematiche a sistemi di accoppiamento dinamici.

Tale approccio, in cui il sistema multi-robot è regolato con vincoli d'impedenza, è un possibile campo di indagine sviluppabile in futuro, dal momento che anche esso si basa sul *Task Frame Formalism* che è in buona parte coincidente con quello fin qui presentato. Anche altri approcci alla coordinazione di più bracci robotici partono proprio dalla definizione di un *framework* in maniera analoga a quanto fatto [1] [11].

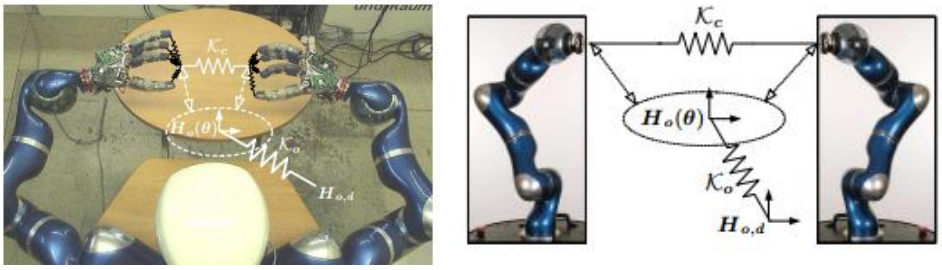


Figura 2.13 : Schema di due bracci robotici controllati in impedenza combinando una molla posta tra di loro con una a livello oggetto (fonte: DLR)

Capitolo 3

Coordinazione del Sistema: Modello Matlab e Sistema Reale

3.1 Modello Matlab

3.1.1 Calcolo Matrici e Simulazione Catene Cinematiche

Il formalismo descritto permette la scrittura sistematica di programmi per la simulazione e in seguito il controllo del sistema multi-robot. Si è quindi realizzato un simulatore in Matlab che permetterà la comoda imposizione di matrici di calibrazione, di traiettorie, di vincoli e legami reciproci, oltre che appunto la simulazione delle catene di trasformazioni e, come si vedrà in seguito, di logiche di inversione cinematica. Tale modello sarà poi validato sul sistema reale.

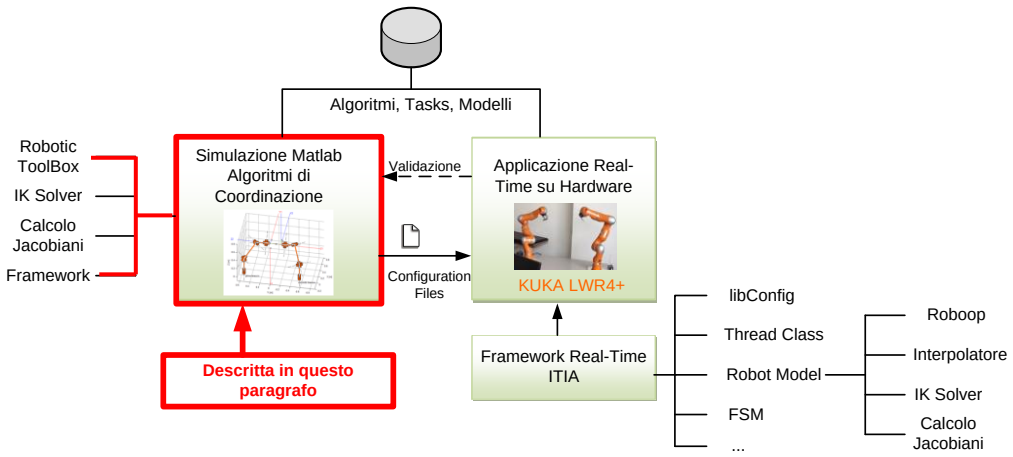


Figura 3.1 : Collocazione del Paragrafo

Si espongono ora i passi logici del programma Matlab.

Passo 0: Inizializzazione

Si richiedono le quantità per inizializzare il sistema: angoli di giunto iniziali, matrici di calibrazione. Questo viene eseguito con interfacce grafiche riassunte dalla seguente Figura :

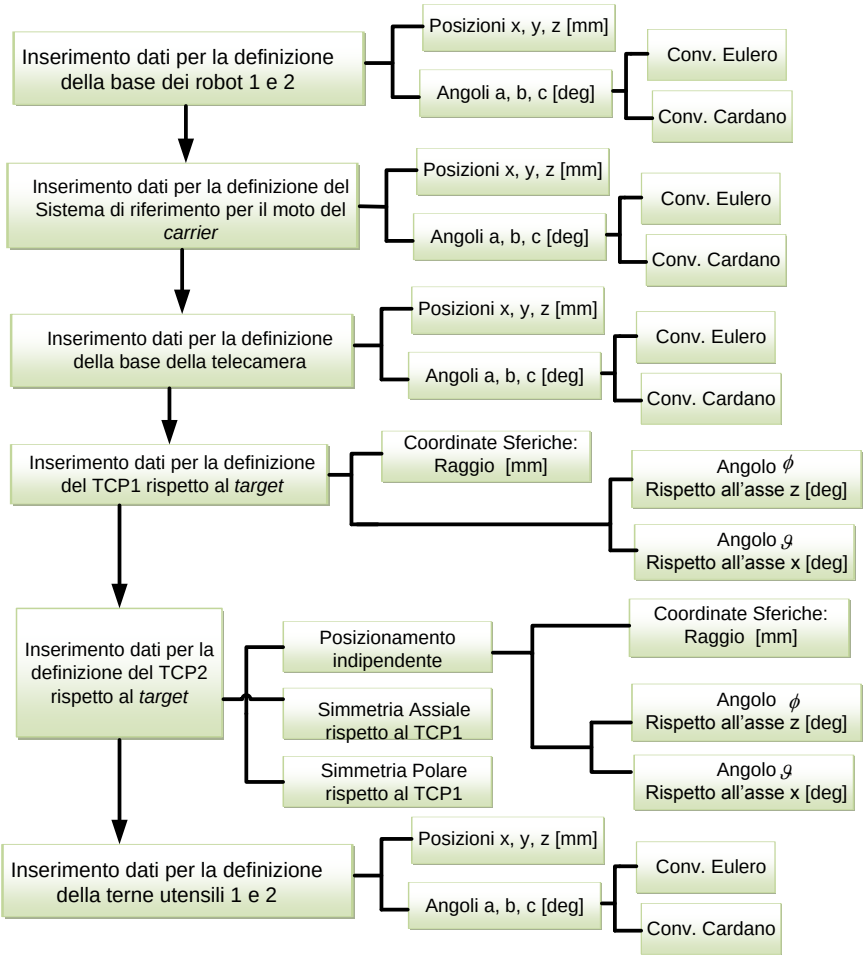


Figura 3.2 : Schema dati richiesti dalla GUI Matlab per effettuare l'inizializzazione

La precedente Figura permette di vedere come gli orientamenti siano specificabili in rappresentazione minima, il che consente una maggior intelligibilità nella designazione delle terne. In particolare ad una rappresentazione in angoli di Eulero scelta inizialmente si è affiancata una rappresentazione in angoli Cardanici r, p, y per allinearsi alle convenzioni del robot (vedere paragrafo 3.1.2). A partire da tali angoli si calcoleranno le matrici di rotazione:

$$R_{Cardano} = Rot_x(r) \cdot Rot_y(p) \cdot Rot_z(y) =$$

$$= \begin{bmatrix} c(y)c(p) & c(p)s(y) & s(p) \\ c(y)s(p)s(r) + c(r)s(y) & s(r)s(p)s(y) + c(y)c(r) & s(r)c(p) \\ c(y)s(p)c(r) + s(r)s(y) & s(r)s(p)c(y) + c(y)s(r) & c(r)c(p) \end{bmatrix}$$

Sempre dalla Figura precedente si vede che per definire $\boldsymbol{T}_{tgt}^{TCP,i}$, oltre alla normale assegnazione di tre dati per la posizione e tre angoli per l'orientamento, si è data la possibilità di usare anche angoli sferici. Ossia, dato un punto di riferimento, si può specificare il raggio r e la sua orientazione rispetto al centro della sfera mediante gli angoli ϑ e ϕ. La terna al TCP sarà ricavata usando tale segmento orientato nello spazio come vettore di *approach* $\boldsymbol{a}$ su cui si andrà ad allineare l'asse z della terna utensile. Questa convenzione porta alla possibilità di definire relazioni geometriche tra i TCP dei due robot: come già accennato in precedenza oltre a posizionamento indipendente infatti l'estremità del secondo robot potrà essere impostata simmetricamente al punto preso come *target* o simmetricamente a una retta passante per esso. Per l'utilizzo delle coordinate sferiche per la determinazione dei vettori di *approach* dei due robot ci si riferisce alla seguente Figura 3.3.

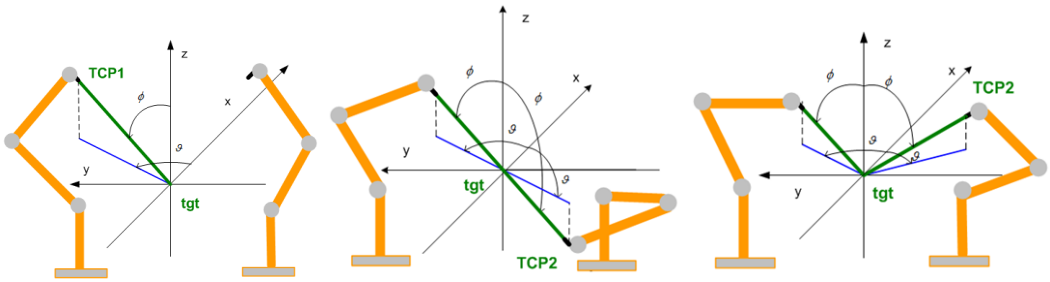


Figura 3.3 : Coordinate sferiche per definire le posizioni dei TCP

I vettori di *approach* risultano:

$$approach_i = \frac{[-r_i \cdot \sin(\phi_i)\cos(\vartheta_i), -r_1 \sin(\phi_i)\sin(\vartheta_i), r_1 \cdot cos(\phi_i)]}{norm(-r_i \cdot \sin(\phi_i)\cos(\vartheta_i), -r_1 \sin(\phi_i)\sin(\vartheta_i), r_1 \cdot cos(\phi_i))}$$

Ma in tal modo sarà ancora libera la rotazione delle terne utensili attorno a tale asse (sarà da specificare il vettore di *opening* ***o***). L'utente potrà specificare tale rotazione oppure lasciarla inespressa così che il programma possa sceglierne una particolarmente adatta. Si potrà dunque sfruttare la rotazione non specificata come grado di libertà aggiuntivo e utilizzare un criterio gestione della ridondanza per fare in modo che la posizione iniziale sia caratterizzata da angoli che, rispettando la posa imposta, siano i più lontani possibili dal fine corsa. Nella Figura 3.4 si ipotizza di non specificare il vettore di *opening*. L'impostazione di default prevede in tal caso di assumere un'orientazione iniziale preimpostata e utilizzare una soluzione della cinematica inversa (vedi Cap. 4) che, attraverso la gestione della ridondanza, trovi l'orientazione migliore in termini di distanza dai limiti di giunto. Si vede che se si lasciassero le terne orientate come nel *guess* iniziale (terne tratteggiate nella figura a sinistra) vi sarebbero ben tre angoli ai limiti di corsa. Sfruttando invece un criterio di inversione cinematica sulla posizione iniziale si sceglie una configurazione di robot con angoli complessivamente più lontani dai limiti (in particolare l'angolo del settimo giunto).

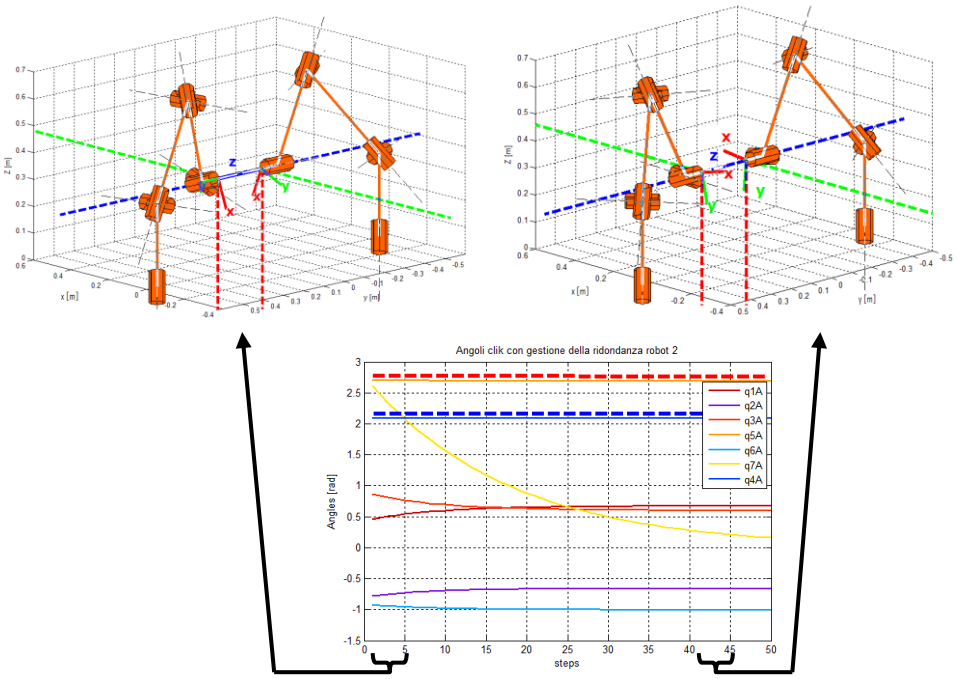


Figura 3.4 : Scelta della rotazione attorno all'asse z della terna utensile, qualora non fosse imposta, al fine di allontanare i giunti dal fine corsa ove possibile

Una volta raccolti tutti i dati di calibrazione dalla GUI si procede ricavandone le relative matrici omogenee, decidendo la frequenza di aggiornamento del sistema e il tempo totale di simulazione. Eseguita l'inizializzazione si avvierà il ciclo base di coordinazione.

Passo 1: Update *Carrier* :

Si acquisiscono istante per istante i dati del movimento del carrier $\boldsymbol{T}_{CA}^{ca}(t)$ (vedi Paragrafo 2.1). Chiaramente in simulazione tali valori andranno estratti da un array di valori precomputati. A tal scopo sono stati realizzati codici per generare traiettorie che vanno da quella che permette di disattivare la trasformazione in questione (matrice identità) a traiettorie rettilinee, a traiettorie chiuse circolari o poligonali a seconda dei parametri inseriti (esempi in Figura 3.5), a traiettorie ancora più complesse in cui si aggiungono angoli di rotazione mentre si percorre la traiettoria o si creano traiettorie elaborando immagini.

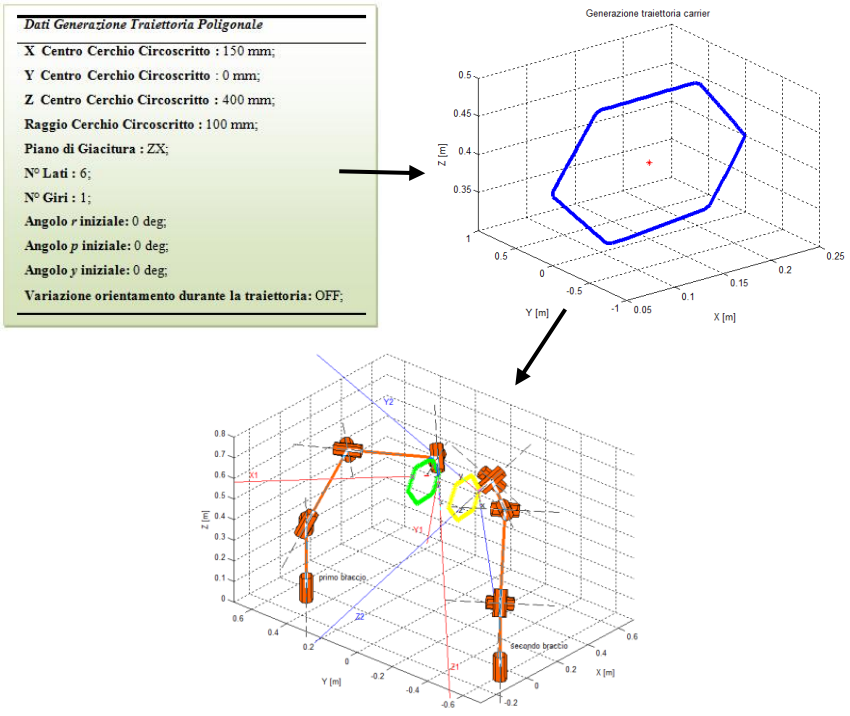


Figura 3.5 : Generazione di una traiettoria poligonale per il carrier a partire dai parametri riportati e moto dei robot agganciati ad esso

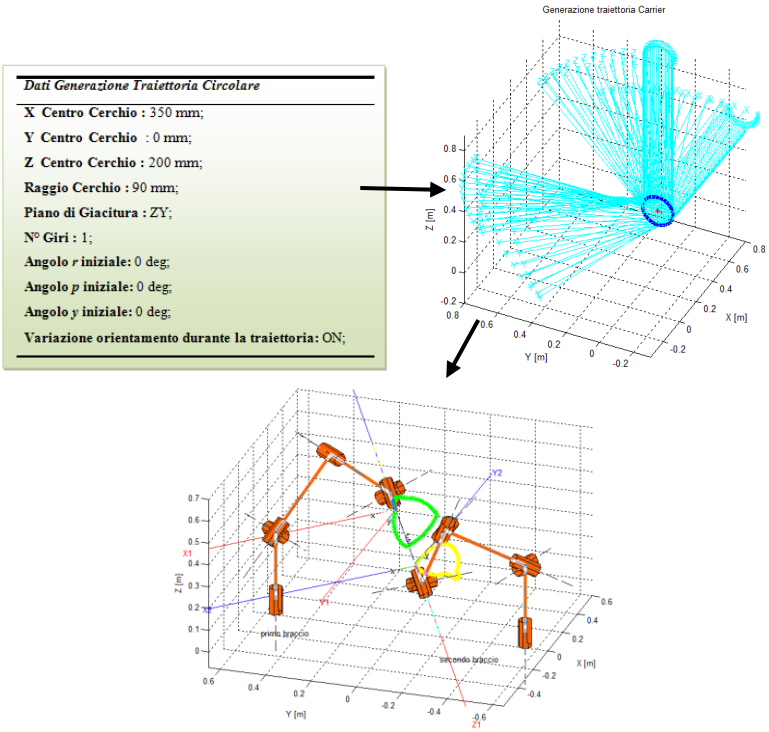


Figura 3.6 : Generazione di una traiettoria circolare per il carrier a partire dai parametri riportati. Si noti che si è impostata una legge per far variare anche l'orientazione della terna centrata su di esso

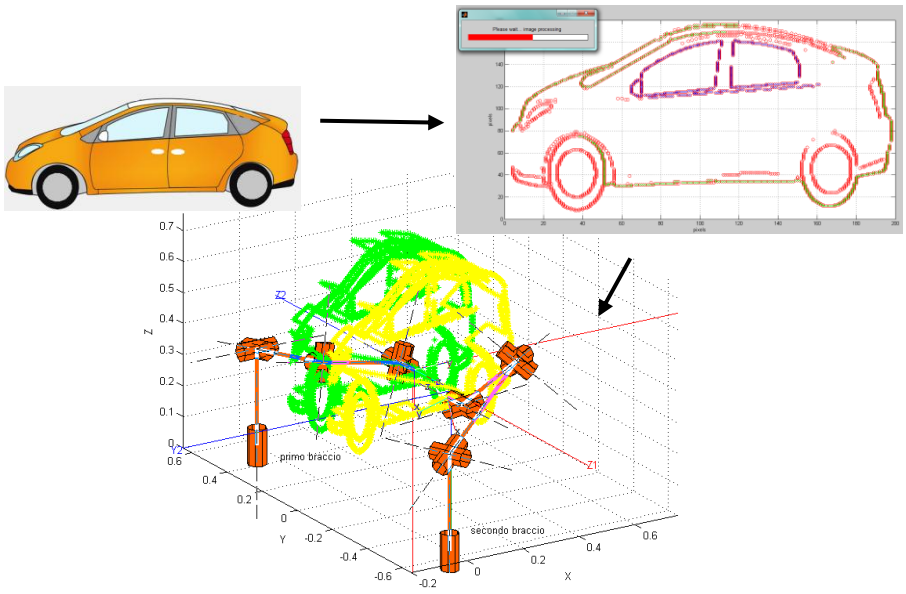


Figura 3.7 : Generazione di una traiettoria a partire dall'elaborazione di un file immagine

Passo 2: Update Input T_{CAM}^{tgt}

La traiettoria così generata andrà a costituire il moto medio del target a cui si può sovrapporre un contributo di vibrazione o di disturbo (imputabile al target stesso o alla rilevazione da parte di telecamere). Si è quindi previsto un modulo attivabile per la generazione di vibrazioni e disturbi. Per quanto riguarda le generazione di vibrazione è ottenibile inserendo i parametri di una serie di Fourier, mentre per quanto riguarda i moti random si sono assunti essere estratti da una normale di cui si possono specificare i parametri. Ciò è visibile nella seguente Figura 3.8 :

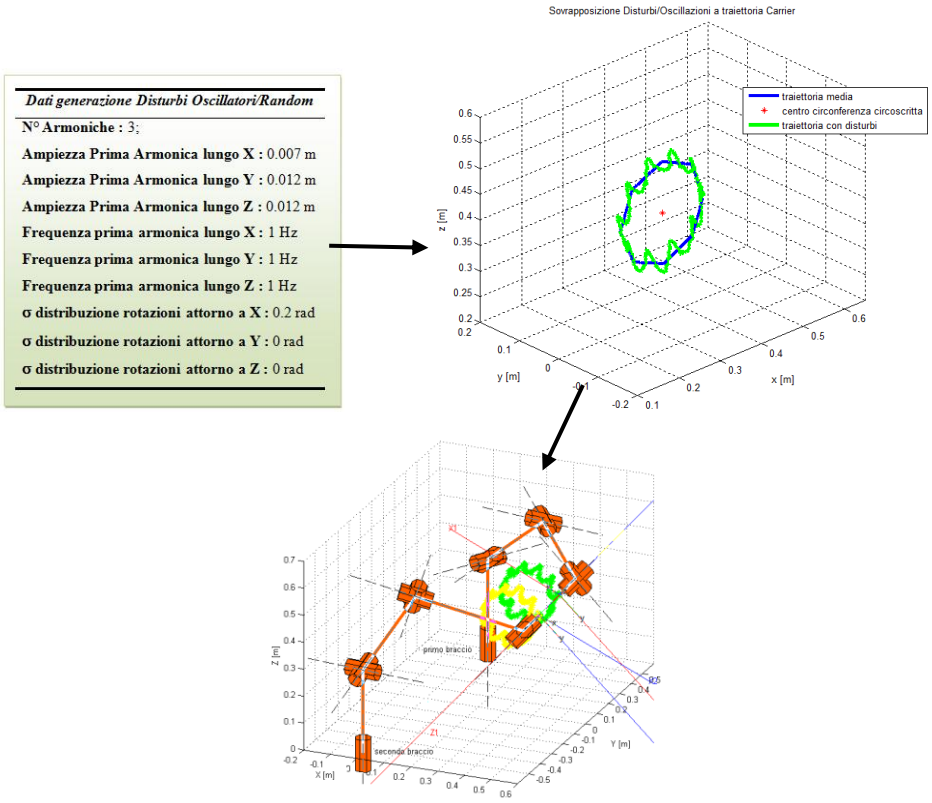


Figura 3.8 : Esempio sovrapposizione moto oscillatorio modellato come serie di Fourier nelle tre traslazioni e rotazioni random

Le formula utilizzata per generare il moto oscillatorio come serie di Fourier è stata:

$$\boldsymbol{x}(t) = \boldsymbol{x}_{medio} + \sum_{i=1}^{N} \frac{\boldsymbol{X}_1}{i^2} \cos(2\pi\, i\, fr_1\, t + \varphi(t))$$

Con ampiezze, numero di armoniche , frequenze, fasi selezionabili a piacere.
Le rotazioni di disturbo attorno agli assi sono estratte casualmente da una normale con deviazione standard desiderata. Si è utilizzata la funzione Matlab $normrnd()$.

Passo 3: Compensazione moto target

I manipolatori dovranno seguire il moto del target descritto da $\boldsymbol{T}_{CA}^{ca}$ e $\boldsymbol{T}_{CAM}^{tgt}$ e per ottenere ciò si adotta un algoritmo di compensazione del moto che verrà descritto nel prossimo paragrafo con cui si chiuderanno le catene cinematiche espresse in precedenza ottenendo $\boldsymbol{T}_{comp1}(\bar{t})$ e $\boldsymbol{T}_{comp2}(\bar{t})$.

Passo 4: Update riferimento

Si acquisiscono poi eventuali input comandati sui singoli robot (comandi aptici, *manual guidance*). Anche qui per effettuarne la simulazione sarà possibile scegliere tra varie traiettorie possibili generate con una funzione simile a quanto già visto per il moto medio del *carrier*.

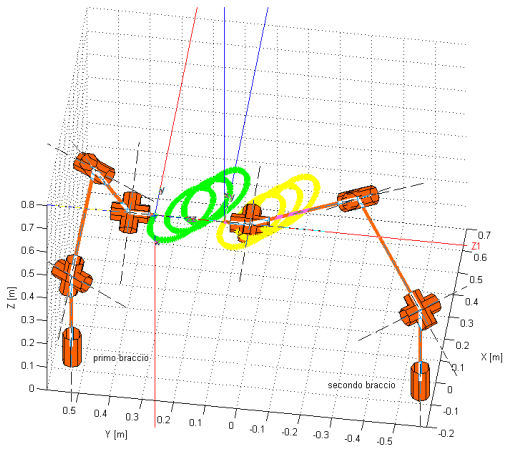


Figura 3.9: Al moto del carrier (circolare) è sovrapposto un moto comandato lineare

Passo 5: Verifica vincoli

La componente di deflessione è dovuta a due contributi: quello in input già acquisito poco sopra e quello di vincolo che , se presente, va calcolato a questo punto. Si è realizzata una funzione per generare anche questa matrice. Essa richiama $\boldsymbol{T}_{R1}^{TCP1}(\bar{t}-1)$ all'istante precedente e valuta quale sarebbe la posizione all'istante considerato se vi

fossero solo le $\boldsymbol{T}_{comp1}(\bar{t})$ e $\boldsymbol{T}_{ref1}(\bar{t})$, di fatto prevedendo quale sarebbe $\boldsymbol{T}_{R1}^{TCP1}(\bar{t})$. Analogamente per il secondo robot, ottenendo $\boldsymbol{T}_{R2}^{TCP2}(\bar{t})$. Da queste matrici estrae le coordinate cartesiane della posizione che sarebbe assunta all'istante in questione dai due TCP: $[x(\bar{t})_1, y(\bar{t})_1, z(\bar{t})_1]$ e $[x(\bar{t})_2, y(\bar{t})_2, z(\bar{t})_2]$ e calcola la distanza reciproca:

$$d(\bar{t}) = \sqrt{(x(\bar{t})_1 - x(\bar{t})_2)^2 + (y(\bar{t})_1 - y(\bar{t})_2)^2 + (z(\bar{t})_1 - z(\bar{t})_2)^2}$$

Se tale distanza diventa più piccola di una soglia minima è possibile imporre una "traslazione di repulsione" modellabile come quella di una molla compressa:
$F = K_f \cdot \Delta l_{compressione}$, dove chiaramente $\Delta l_{compressione} = d_{min} - d(\bar{t})$ ove d_{min} è la distanza sotto lo quale si attiva il vincolo (e quindi considerabile come la lunghezza di molla scarica). L'output di questa funzione sarà dunque:

$\mathbf{T}_{\text{constr},1,2} = \text{transl}([-\text{F}_\text{x}, -\text{F}_\text{y}, -\text{F}_\text{z}])$.

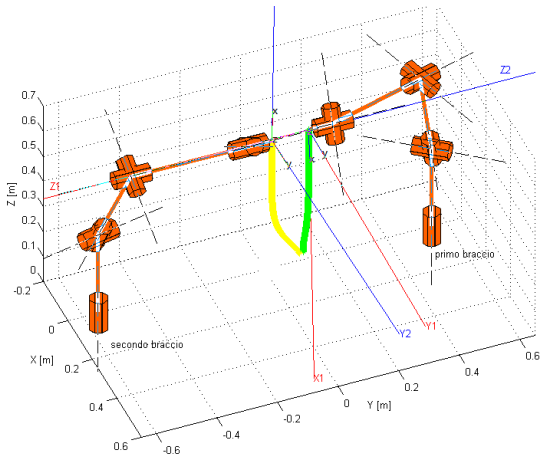


Figura 3.10 : Esempio vincolo di repulsione attivato. I due end effector si allontanano rispetto alla posizione iniziale dove erano a contatto e proseguono in verticale solo se a una debita distanza

Passo 6: Calcolo Matrici di Output

Si procede quindi a calcolare $\boldsymbol{T}_{defl1}$, $\Delta\boldsymbol{T}_{R1}$ e quindi la nuova $\boldsymbol{T}_{r1}^{R1}$. In modo analogo si agisce per il secondo braccio in base al legame mutuo scelto e in base ai passaggi esposti nel paragrafo precedente, ottenendo $\boldsymbol{T}_{r2}^{R2}$.

Passo 7: Update configurazione

Si conclude il ciclo aggiornando i legami $\boldsymbol{T}_{tgt}^{TCP1}$ e $\boldsymbol{T}_{tgt}^{TCP2}$ che in generale variano per componenti di deflessione non nulle.

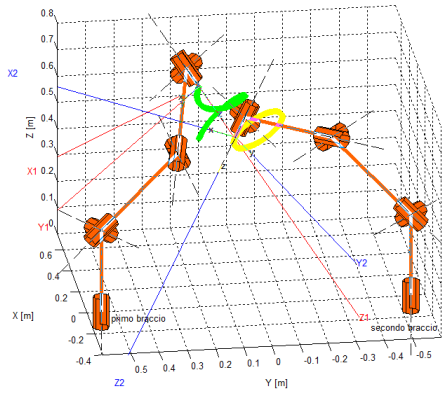


Figura 3.11 : Esempio di moto *Task Space Oriented.* Mentre il primo braccio subisce anche una deflessione verso l'alto, il secondo si limita a seguire il carrier su un tracciato circolare

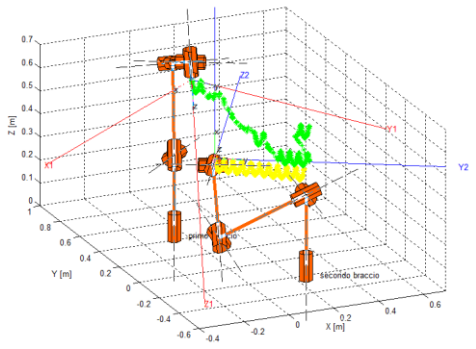


Figura 3.12 : Esempio con robot posizionati in modo generale e indipendente. Entrambi compensano un moto rettilineo con disturbi oscillatori. Il primo braccio ha anche un riferimento inserito mediante slider gains

Dalla Figura 3.12 si vede che si può simulare qualsiasi generico moto. Però poi si deve passare a considerarne la fattibilità. Infatti ci si può chiedere se i limiti di escursione degli angoli saranno rispettati, così come i limiti di velocità. Ad esempio, proprio nel caso di Figura 3.12 si ha un superamento dei limiti di giunto e di velocità, per via delle configurazioni imposte e della presenza delle vibrazioni (Fig. 3.13). Questi problemi andranno affrontati parlando dei metodi di inversione cinematica.

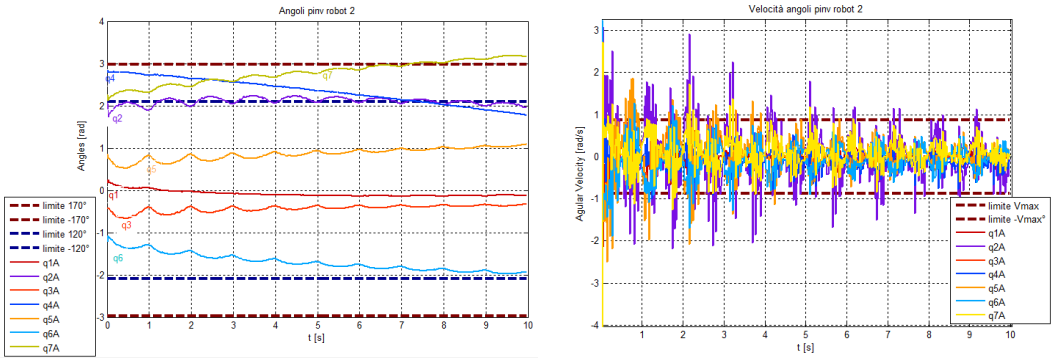


Figura 3.13 : Andamento angoli e velocità Robot 2 in esecuzione del task di Fig. 3.12

3.1.2 Modello del Robot LWR

- In questo paragrafo si discutono gli aspetti di allineamento tra il sistema reale e i modelli. Innanzitutto, per rappresentare il robot con la Robotic Toolbox e poi per l'inversione cinematica, è servito ricavare i dati geometrici del Robot. La seguente tabella riporta i Parametri di Denavit-Hartenberg per il KUKA-LWR4+:

N. Asse	$a_i\ [m]$	$d_i\ [m]$	$\alpha_i\ [rad]$	$\vartheta_i\ [rad]$
1	0	0.3105	$\pi/2$	ϑ_1
2	0	0	$-\pi/2$	ϑ_2
3	0	0.4	$\pi/2$	ϑ_3
4	0	0	$-\pi/2$	ϑ_4
5	0	0.39	$\pi/2$	ϑ_5
6	0	0	$-\pi/2$	ϑ_6
7	0	0.0785	0	ϑ_7

Tabella 3.1 : Parametri di Denavit Hartenberg

- Tuttavia si deve tenere presente che rispetto alla convenzione di Denavit-Hartenberg il robot prevede degli offset per quanto riguarda gli angoli ϑ_i: si hanno due offset diversi a seconda che si leggano gli angoli dal pannello di controllo (KCP) o dall'interfaccia di comunicazione (FRI) (vedi Paragrafo 1.3) e tali offset si sono dovuti quantificare come riportato ella seguente tabella. Oltre agli offset anche la nomenclatura degli assi è diversa a seconda che si leggano da pannello di controllo o da interfaccia di comunicazione, quindi anche questo va tenuto in considerazione per effettuare confronti. Come si vede nella seguente tabella la differenza riguarda il terzo giunto, che nel caso del KCP è visto come un asse aggiuntivo "e1" collocato alla fine della sequenza.

N. Asse	*Offset FRI* $[rad]$	*Offset KCP* $[rad]$	*Nome KCP*	*Nome FRI*
1	0	0	A1	J1
2	0	$\pi/2$	A2	J2
3	π	π	E1	J3
4	0	0	A3	J4
5	π	π	A4	J5
6	0	0	A5	J6
7	0	0	A6	J7

Tabella 3.2 : Offset rispetto alla convenzione di Denavit-Hartenberg e denominazione assi

Questo ha portato a dover inserire all'inizio del programma l'informazione sulla convenzione utilizzata, così da poter avere risultati coerenti.

- Altri dati cinematici di interesse che verranno inseriti nel modello Matlab per considerazioni successive sono i limiti di giunto e le velocità massime sempre ai giunti:

N. Asse	*Angolo Massimo [deg]*	*Angolo Minimo [deg]*	*Velocità Massima [deg/s]*
1	170	-170	100
2	120	-120	100
3	170	-170	100
4	120	-120	120
5	170	-170	120
6	120	-120	170
7	170	-170	170

Tabella 3.3 : Escursioni massime angoli e limiti di velocità

- Inoltre, per parlare di allineamento tra modello e sistema reale si devono considerare le convenzioni angolari per le rappresentazioni minime, che come

già accennato in precedenza sono gli angoli di Cardano per la FRI e gli angoli di Cardano a ordine inverso per quanto riguarda il KCP.

angoli	*FRI*	*KCP*
r	$rot_x(r)$	$rot_z(r)$
p	$rot_y(p)$	$rot_y(p)$
y	$rot_z(y)$	$rot_x(y)$

Tabella 3.4 : Convenzioni angolari di FRI e KCP per le rappresentazioni minime

- Un successivo problema riguardante l'allineamento sarà costituito dal fatto che lo Jacobiano fornito dal LWR e usato per confrontarsi punto per punto con quello calcolato sarà sempre fornito come Jacobiano geometrico e in terna utensile e si necessiteranno quindi le adeguate conversioni (vedi Paragrafo 6.1).

3.2 Dal Modello Matlab al Software per il Sistema Reale

3.2.1 I File di Configurazione

Il modello Matlab serve per descrivere il moto del sistema reale. Questo significa che dopo aver impostato su Matlab dei posizionamenti relativi e delle traiettorie desiderate è possibile simularle e quindi trasferirle al sistema reale (approccio al motion planning) salvando le relative matrici su files di configurazione che saranno passati al programma C++ che realizza l'applicazione. Quindi il modello non sarà usato solamente per simulare il comportamento del sistema, ma anche per fornire in output files di configurazione utilizzati in seguito per controllare il sistema reale.

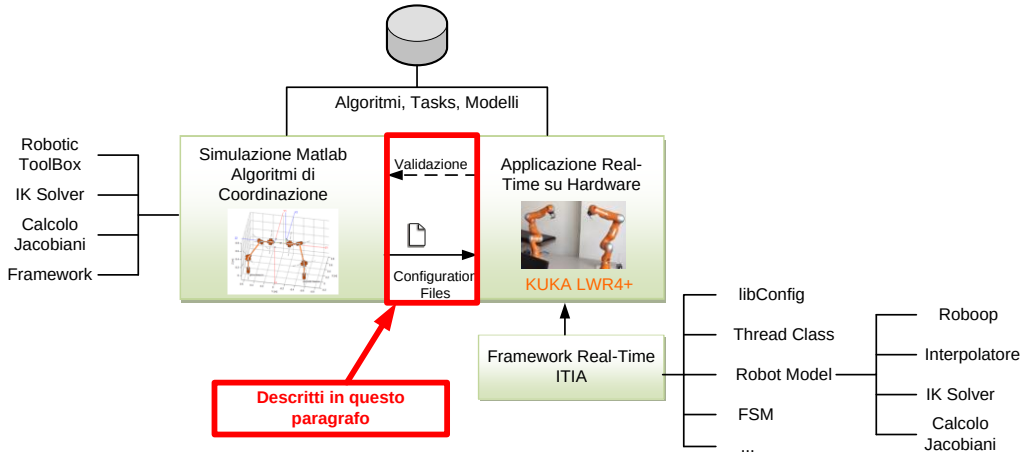


Figura 3.14 : Collocazione del paragrafo: utilizzo modello Matlab

I files di configurazione sono dei files di testo generati automaticamente dagli scripts Matlab e contenenti tipicamente i dati per la creazione delle matrici di calibrazione e di posizionamento iniziale ($\boldsymbol{T}_{W}^{R_1}$, $\boldsymbol{T}_{W}^{R_2}$, $\boldsymbol{T}_{W}^{CA}$, $\boldsymbol{T}_{ah}^{CAM}$, $\boldsymbol{T}_{tgt}^{TCP1}$, $\boldsymbol{T}_{tgt}^{TCP2}$, $\boldsymbol{T}_{r1}^{TCP1}$, $\boldsymbol{T}_{r2}^{TCP2}$) oppure i dati per la generazione di traiettorie da seguire ($\boldsymbol{T}_{CA}^{ca}$ ad esempio per imporre il moto di un *carrier* virtuale, o $\boldsymbol{T}_{CAM}^{tgt}$ per simulare vibrazioni random del target rilevabili da una telecamera). Si riportano di seguito degli esempi di file di configurazione, il cui formato è quello richiesto dalla libreria open source "*Libconfig"* per poter estrarre dati da un file di testo in un programma scritto in C++.

```
version = "1.0  distances in [mm] and angles in [deg]";

T_W_R1 = [ 0.0, 550.0, 0.0, 0.0, 0.0, 0.0 ];
T_W_R2 = [ 0.0, -550.0, 0.0, 0.0, 0.0, 0.0 ];
T_W_AH = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ];
T_ah_CAM = [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ];
T_tgt_TCP1 = [ 10.0, 45.0, 45.0, 1.0 ];
T_tgt_TCP2 = [ 10.0, 45.0, -45.0, 1.0 ];
T_r1_TCP1 = [ 0.0, 0.0, 3.0, 0.0, 0.0, 0.0 ];
T_r2_TCP2 = [ 0.0, 0.0, 3.0, 0.0, 0.0, 0.0 ];
```

Il precedente è un esempio di file di configurazione delle matrici di calibrazione e di posizionamento iniziale che sarà caricato quindi all'inizio dell'esecuzione del task di motion coordination dal programma.

```
#
CARTESIAN_SPACE
ypr
0.0           #time [s]
0.0 0.25 0.22 #xyz  [m]
0.0 0.10 0.0  #angles [rad]
2.0           #time [s]
0.0 0.25 0.23 #xyz  [m]
0.1 0.12 0.0  #angles [rad]
4.0           #time [s]
0.0 0.25 0.24 #xyz  [m]
0.2 0.14 0.0  #angles [rad]
```

Il precedente è un file di configurazione per la generazione di traiettorie. Si specifica lo spazio di interpolazione (spazio cartesiano), la convenzione angolare utilizzata (angoli di Cardano), e a ogni istante di tempo la relativa posizione e orientamento.

3.2.2 Realizzazione interpolatore in C++

Come si è mostrato è stato possibile creare dei files di configurazione in cui a istanti specificati viene assegnata una posizione in coordinate cartesiane x, y, z e angolari con angoli di Cardano r, p, y, permettendo di designare traiettorie da seguire. Tuttavia, se il riferimento deve essere passato al robot ad alta frequenza, si necessita di realizzare un interpolatore per fornire un riferimento anche per gli istanti non descritti nel file di configurazione. Ovviamente si potrebbero generare delle traiettorie già con passo temporale pari al millesimo di secondo, ma la soluzione dell'interpolatore consente maggior flessibilità permettendo di variare il tempo di campionamento, di specificare delle traiettorie non necessariamente generate con un simulatore Matlab, di alleggerire i dati da caricare. L'interpolatore è stato realizzato a partire da alcuni codici già presenti nella libreria open source "ROBOOP: A Robotics Object Oriented Package in C++" e prevede di integrare i metodi di interpolazione già presenti in tale libreria ma che agiscono separatamente per posizioni e angoli.

Per quanto riguarda l'interpolazione delle posizioni assegnate in coordinate cartesiane, il metodo impiegato dalla classe "ROBOOP::Spl_path" è un normale metodo di spline cubica per la cui trattazione teorica si rimanda ai riferimenti presenti nella documentazione della libreria o a [12].

Per quanto riguarda l'interpolazione degli orientamenti, la rappresentazione è basata sull'uso dei quaternioni, e la tecnica utilizzata dalla classe "ROBOOP::Spl_quaternion" è quella del "Quaternion Slerp". Il termine "Slerp" è acronimo di "Spherical Linear Interpolation", tecnica sviluppata in computer grafica con l'intento di animare rotazioni tridimensionali. Essa fa riferimento a un moto a velocità costante lungo un arco di circonferenza a raggio unitario, assegnati i limiti di tale arco e con un parametro di interpolazione compreso tra zero e uno.
Lo Slerp ha una formula geometrica indipendente dalla successiva formulazione con i quaternioni, e indipendente dalle dimensioni dello spazio in cui l'arco si trova. Questa formula si basa sul fatto che ogni punto della curva deve essere una combinazione lineare degli estremi. Siano infatti $\mathbf{p}_0$ e $\mathbf{p}_1$ il primo e l'ultimo punto della curva e sia t il parametro $0 \leq \mathrm{t} \leq 1$. Se Ω è l'angolo sotteso all'arco in questione, si ha che $\cos(\Omega) = \mathbf{p}_0 \cdot \mathbf{p}_1$, ove $\mathbf{p}_0 \cdot \mathbf{p}_1$ è il prodotto scalare dei versori dall'origine alle estremità. Da ciò si ottiene la formula di Slerp geometrico come:

$$Slerp(\boldsymbol{p}_0, \boldsymbol{p}_1; t) = \frac{\sin[(1-t)\Omega]}{\sin\Omega}\boldsymbol{p}_0 + \frac{\sin[t\Omega]}{sin\Omega}\boldsymbol{p}_1$$

Con lo Slerp si ottiene quindi l'equivalente in geometria sferica di un percorso lungo un segmento nel piano. Estendendo il concetto di Slerp ai quaternioni si riesce ad ottenere un'interpolazione di rotazioni nello spazio come quelle garantite dalla classe "ROBOOP::Spl_quaternion". Tuttavia essa richiede in ingresso proprio dei quaternioni agli istanti considerati, ed è stato quindi necessario modificarla per poter leggere direttamente dal file di configurazione i più comodi angoli di Cardano (Si veda l'appendice per richiami sui quaternioni).

Esiste una classe, sempre della libreria in questione, che facilita le conversioni da matrice di rotazione a quaternione e viceversa: "ROBOOP::Quaternion".

Nei file di configurazione tuttavia sono presenti, istante per istante, gli angoli di Cardano. Quindi si deve procedere a ricavare la matrice di rotazione a partire da essi, come già scritto in 3.1.1.

Quindi per realizzare l'interpolatore si sono seguiti i seguenti passi:

1) Modifica della classe "ROBOOP::Spl_path" affinché potesse leggere le coordinate spaziali dai file di configurazione strutturati come descritto in precedenza.
2) Modifica della classe "ROBOOP:: Spl_Quaternion" affinché potesse leggere gli angoli di Cardano dai file di configurazione.
3) Modifica della classe ROBOOP:: Spl_Quaternion" affinché creasse la matrice di rotazione a partire da tali angoli.
4) Modifica della classe ROBOOP:: Spl_Quaternion" affinché ricavasse i quaternioni a partire dalla matrice di rotazione.
5) Creazione di una classe "Interpolatore" per unire le interpolazioni di posizione e orientamento. Tale classe viene costruita passando il file di configurazione e restituisce le matrici omogenee agli istanti desiderati mediante la funzione "$Update\ (t)$" che se richiamata procede ad eseguire i seguenti passi:

   - Interpola all'istante considerato la spline spaziale mediante la classe "ROBOOP::Spl_path" modificata come descritto prima. Si ottiene un vettore di posizioni $\boldsymbol{s}(\bar{t}) = [x(\bar{t}), y(\bar{t}), z(\bar{t})]$.
   - Interpola all'istante considerato il quaternione mediante la classe ROBOOP::Spl_Quaternion" modificata come descritto prima. Si ottiene un quaternione $q(\bar{t})$ che verrà utilizzato per calcolare la corrispondente matrice di rotazione $\boldsymbol{R}(\bar{t})$ utilizzando le relazioni riportate sopra.
   - Unisce la componente di traslazione e quella di rotazione : $\boldsymbol{T}_{tot} = transl\big(\boldsymbol{s}(\bar{t})\big) \cdot \begin{bmatrix} \boldsymbol{R}(\bar{t}) & 0 \\ \underline{0} & 1 \end{bmatrix}$. Si è ottenuta una matrice che descrive una posa intermedia interpolata tra due istanti temporali specificati nel file di

configurazione . Negli screenshots che seguono è riportata la classe "Interpolatore":

Dichiarazione nell'header file:

```
class Interpolatore
{
  public:
Interpolatore(std::string configfile);
~Interpolatore();

void Update(double t);
NEWMAT::Matrix& GetTransform() { return this->T;}

  private:
ROBOOP::Spl_path::Spl_path      m_path;
ROBOOP::Spl_Quaternion::Spl_Quaternion  m_quaternion_path;
NEWMAT::ColumnVector            s;
NEWMAT::Matrix                  T, Trot;
ROBOOP::Quaternion              q;

};
```

Costruttore:

```
Interpolatore::Interpolatore(std::string configfile)
    : m_path (configfile)
    , m_quaternion_path(configfile)
    , s(4)
    , T(4,4)
    , Trot(4,4)
{ }
```

Funzione " $Update\ (t)$":

```
void Interpolatore::Update(double t)
{
    int nstep = 0;
      try{
          nstep = 1;
          m_path.interpolating(RBD_COMMON::Real(t),s);

          nstep = 2;
          m_quaternion_path.quat(RBD_COMMON::Real(t),q);

          nstep = 3;
          Trot = q.T();
          T = ROBOOP::transl(s)*Trot;
          }
      catch(ROBOOP::Exception& e) {
          std::string err =
          std::string("\nException in Interpolatore::Update due to")
          + e.what();
          throw ROBOOP::Exception (err);

      }
}
```

3.3 Software in C++ per il Sistema Reale

Si passa a descrivere l'applicazione sviluppata in C++ per l'utilizzo dell'algoritmo di coordinazione dei bracci, la cui collocazione è presentata nella seguente figura.

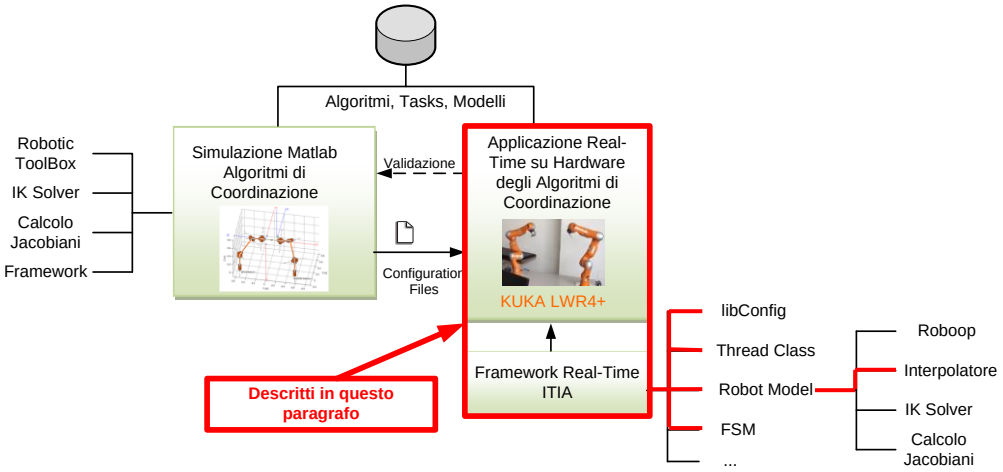


Figura 3.15 : Collocazione del Paragrafo

Si deve dapprima spiegare la struttura del *framework* real-time disponibile in ITIA per le applicazioni *multi-arm* con KUKA LWR, in cui si è sviluppata la serie di funzioni e classi per la gestione del moto combinato dei LWR.

- Il *framework* è composto da threads che, come già accennato, sono parti di processo eseguibili in parallelo ad altri threads. Ognuno di essi, una volta avviato, vive in parallelo agli altri threads e svolge il suo compito che può essere quello di eseguire calcoli o di scambiare dati o comandi. In particolare lo scambio di dati o comandi può avvenire dall'esterno (via *network sockets*) o con accesso a una memoria condivisa. In quest'ultimo caso la lettura/scrittura sulla memoria condivisa non potrà essere effettuata contemporaneamente da più threads, che quindi saranno dotati di un sistema di *Mutex* che renderanno inaccessibile quella data porzione di memoria se un thread ne starà leggendo/scrivendo i contenuti in quell'istante. Tutto questo è riassunto nella

seguente Fig. 3.16, in cui la geometria circolare richiama la periodicità di esecuzione dei threads che accedono a memoria condivisa (pallini neri) o a network sockets (quadrati). Si procede ora a dettagliare la Figura 3.16, che rappresenta l'applicazione utilizzata con in evidenza i threads che esistono.

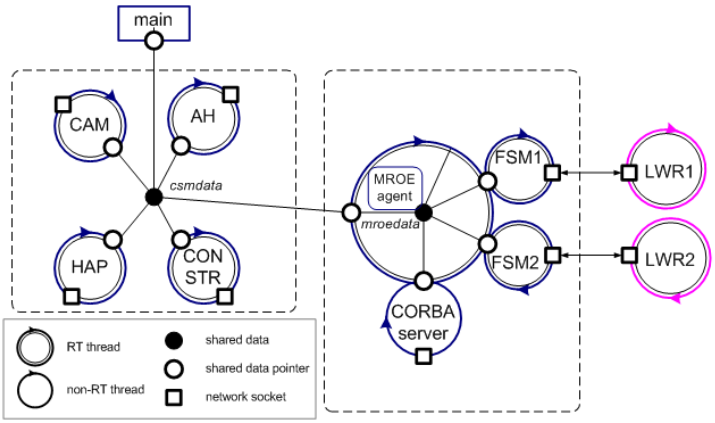


Figura 3.16 : Rappresentazione applicazione

- Si hanno i thread FSM1 e FSM2 con cui si gestisce la Fast Research Interface (vedi Paragrafo 1.3.4). Il loro ruolo è quindi quello di comunicare con i robot, passando i comandi da eseguire e leggendo le grandezze misurate. Ciascuno di questi thread è di natura aciclica in quanto realizza una risposta di scambio dati via UDP con un thread connesso chiamante (eseguito sul LWR KRC). Lo scambio dati deve essere molto rapido (vedi Paragrafo 1.3.4) pena il decadimento dell'interfaccia. La costituzione del thread come macchina a stati finiti è necessario alla gestione lato PC remoto dei corrispondenti stati del KRC-FRI (es. MON/CMD mode, sempre riferendosi al Paragrafo 1.3.4).

- Si ha il thread MROE con cui si esegue l'algoritmo di coordinazione vero e proprio. Questo thread quindi accede in lettura protetta da *Mutex* alla struttura dati *mroedata* dove sono allocate tutte le matrici che caratterizzano il sistema (vedi Cap. 2) e le aggiorna in modalità scrittura protetta in base ai dati che vengono passati dai manipolatori e dal resto del sistema o in base all'algoritmo di coordinazione. Tale algoritmo verrà implementato all'interno di una classe

mroeAgent di cui un oggetto viene creato all'interno del thread MROE. L'oggetto di classe *mroeAgent* sovrintende a tutte le operazioni di lettura/scrittura e calcolo cicliche. Il risultato sarà l'aggiornamento delle matrici di output che verranno passate ai thread FSM1 e FSM2 affinché i robot si muovano in quella posizione. Si noti che quando si introdurranno i metodi di inversione cinematica il thread MROE avrà il passaggio aggiuntivo di calcolare appunto la cinematica inversa così da passare come comando ai robot non matrici ma già vettori di angoli comandati ai giunti.

- Si ha, in riferimento alla NCS (vedi Paragrafo 1.3.2), l'accesso protetto da *Mutex* da parte del thread MROE a una struttura contenente i dati degli altri devices del sistema: *csmdata*. Questa struttura conterrà i dati tempo-varianti della posizione del carrier, i dati provenienti, ad esempio, dalla telecamera o dal sistema aptico, i dati sui vincoli da applicare per evitare impatti tra parti del sistema. Sarà quindi aggiornata (sempre basandosi su un sistema di Mutex) dai threads dedicati alla gestione di carrier, dispositivo aptico, telecamere, vincoli e sarà sfruttata dal thread MROE per avere istante per istante le matrici tempo-varianti in input. Tali matrici varranno sfruttate dalla classe *mroeAgent* per eseguire l'algoritmo di coordinazione.

- Quindi, l'applicazione consisterà in un *main* (eseguito sotto Linux Real-Time Xenomai) che concettualmente avrà il solo compito creare i vari threads, avviarli e infine terminarli con opportuna gestione di pulizia di memoria.

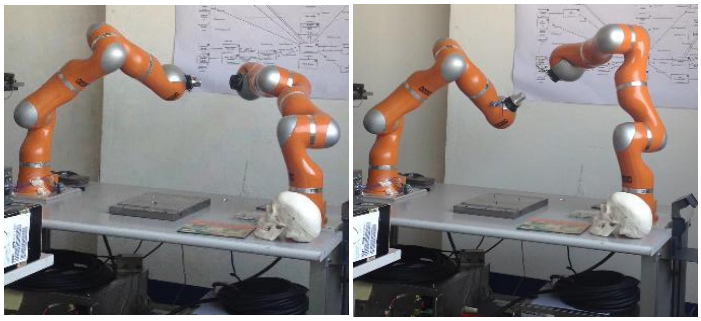


Figura 3.17 : Fotogrammi moto coordinato eseguito con il software descritto

Capitolo 4

Metodi di Inversione Cinematica

4.1 Ridondanza e Inversione Cinematica

Come accennato nell'introduzione si procederà ad indagare metodi di inversione cinematica per sfruttarla al fine di far compiere al sistema task secondari come l'evitare ostacoli, il posizionarsi meglio in funzione del compito da eseguire o in funzione della cooperazione con l'altro robot.

Inoltre grazie all'inversione cinematica si aggiungerà la possibilità di assegnare riferimenti nello spazio cartesiano in controllo di posizione, cosa che come visto in precedenza (Paragrafo 1.3.4) non è possibile fare da FRI.

Verranno qui introdotti i principali concetti per trattare l'argomento.

Definizione di ridondanza

Quando un manipolatore robotico possiede un numero di gradi di libertà superiore a quello necessario per eseguire un determinato compito, si dice che il robot è cinematicamente ridondante rispetto all'operazione assegnata. I manipolatori robotici non ridondanti soffrono maggiormente di limitazioni meccaniche (ad esempio i finecorsa dei giunti) e di configurazione del braccio (ad esempio la limitazione dello spazio di lavoro), mentre l'introduzione della ridondanza cinematica permette di migliorare la destrezza e la flessibilità nei movimenti. Ciò si riflette in riduzione di tempi di riprogrammazione e lavorazione, risparmio energetico, minore ingombro, aumento di prestazioni grazie alla possibilità di sfruttare la ridondanza per finalità

"ah hoc" passando per opportuni metodi di inversione cinematica, come si vedrà nel seguito. Quindi l'introduzione della ridondanza cinematica nei manipolatori permette di eseguire dei movimenti nella struttura senza imprimere variazioni di posizione ed orientamento dell'end effector (*self-motion*).

Ridondanza e Jacobiani

Per contro, la trattazione matematica di un manipolatore ridondante risulterà più articolata quando si vorrà passare dallo spazio operativo a quello dei giunti mediante Jacobiano. Infatti, definendo :

$$\boldsymbol{v} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\omega} \end{bmatrix}$$

il vettore delle velocità nello spazio operativo (dove $\dot{\mathbf{p}}$ è il vettore delle velocità lineari e $\boldsymbol{\omega}$ è il vettore di quelle angolari), si ha che tale $\boldsymbol{v}$ è legata alla velocità dei giunti $\dot{\mathbf{q}}$ da una relazione locale, che dipende quindi dalla configurazione assunta nel determinato istante, espressa dalla matrice Jacobiano geometrico $\boldsymbol{J}_G$:

$$\boldsymbol{v} = \boldsymbol{J}_G(\boldsymbol{q})\dot{\boldsymbol{q}}$$

Lo Jacobiano geometrico può essere determinato mediante un procedimento geometrico che individua il contributo delle velocità di ogni giunto alle componenti di velocità lineare e angolare dell'end effector. Se invece si definisce lo Jacobiano attraverso un'operazione di derivazione delle funzioni della cinematica diretta rispetto alle variabili di giunto, si ottiene lo Jacobiano analitico, indicato con $\boldsymbol{J}_A(\boldsymbol{q})$, o $\boldsymbol{J}(\boldsymbol{q})$ nel seguito. Lo Jacobiano analitico è generalmente diverso da quello geometrico a causa della derivazione rispetto al tempo di una rappresentazione minima dell'orientamento che non coincide con le componenti del vettore delle velocità angolari ω. É comunque possibile stabilire una relazione tra i due Jacobiani.
Mentre si rimanda a un momento successivo il metodo di calcolo dello Jacobiano, si fa ora notare che $\boldsymbol{J}(\boldsymbol{q})$ ha dimensione $(m \times n)$, dato che mette in relazione il vettore delle velocità dell'organo terminale $\boldsymbol{v}$ di dimensioni m (dove $m = 6$ nel caso di

posizionamento ed orientamento nello spazio), con il vettore $\dot{\mathbf{q}}$ di dimensioni n nello spazio delle velocità di giunto. Nel caso di manipolatore ridondante si ha che $m < n$, ed esistono pertanto $l = n - m$ gradi di mobilità ridondanti, con lo Jacobiano che risulterà una matrice rettangolare. Per analizzare ciò, è possibile caratterizzare lo Jacobiano in termini di spazio immagine e spazio nullo.

Lo spazio immagine di $\boldsymbol{J}$ è il sottospazio $Im(\boldsymbol{J}) \in \mathbb{R}^m$ che individua le velocità dell'organo terminale generabili dalle velocità di giunto, nella configurazione assegnata al manipolatore. Lo spazio nullo di $\boldsymbol{J}$ è il sottospazio $Ker(\boldsymbol{J})$ in $\mathbb{R}^m$ a cui appartengono le velocità di giunto che non producono alcuna velocità all'organo terminale, nella configurazione assegnata al manipolatore. Se lo Jacobiano è di rango pieno, si ha:

$$\begin{cases} dim\big(Im(\boldsymbol{J})\big) = m \\ \dim\big(Ker(\boldsymbol{J})\big) = n - m \end{cases} \Rightarrow dim\big(Im(\boldsymbol{J})\big) + \dim\big(Ker(\boldsymbol{J})\big) = n$$

Quanto detto serve a definire l'esistenza di uno spazio nullo $Ker(\boldsymbol{J}) \neq \{0\}$ per i manipolatori ridondanti, da cui si partirà per individuare procedure sistematiche di gestione dei gradi di ridondanza.

Infatti, dato il vettore $\mathbf{v}$, indicando con $\dot{\mathbf{q}}^*$ una soluzione di $\mathbf{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^*$ e con $\boldsymbol{P}$ una matrice $(n \times n)$ tale che $Im(\boldsymbol{P}) = Ker(\boldsymbol{J})$, anche il vettore di velocità ai giunti costruito come $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}^* + \boldsymbol{P}\dot{\boldsymbol{q}}_a$ sarà soluzione di $\mathbf{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$, (con $\dot{\boldsymbol{q}}_a$si indica un vettore arbitrario di velocità nello spazio dei giunti). Infatti premoltiplicando ambo i membri dell'espressione precedente per $\boldsymbol{J}$, si ottiene:

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^* + \underbrace{\boldsymbol{J}(\boldsymbol{q})\boldsymbol{P}\dot{\boldsymbol{q}}_a}_{\boldsymbol{J}(\boldsymbol{q})\boldsymbol{P}=\mathbf{0}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}^* = \mathbf{v}$$

L'effetto di $\dot{\boldsymbol{q}}_a$ sarà quello di generare dei moti interni della struttura, che non apportano modifiche alla posizione ed orientamento dell'organo terminale, ma che

possono consentire l'individuazione di posture del manipolatore per l'esecuzione di un compito assegnato. L'individuazione di una matrice di proiezione nel nullo e lo sfruttamento di $\dot{\boldsymbol{q}}_a$ sarà quindi presente negli algoritmi utilizzati per l'inversione cinematica.

Inversione cinematica

L'inversione cinematica è un processo che permette di ottenere le velocità dei giunti del manipolatore quando sono note le velocità desiderate nello spazio operativo dell'end effector. Supponendo la matrice Jacobiana quadrata si possono ottenere le velocità dei giunti semplicemente con la seguente formula:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\boldsymbol{v}$$

Se è nota la postura iniziale, le posizioni dei giunti possono essere calcolate con un integrale nel dominio del tempo:

$$\boldsymbol{q}(t) = \int_0^t \dot{\boldsymbol{q}}(s)ds + \boldsymbol{q}(0)$$

Di fatto la soluzione sarà da applicare a tempo discreto, e in tal senso la soluzione genericamente adottata prevede l'utilizzo della formulazione di Eulero in avanti:

$$\boldsymbol{q}(t_{k+1}) = \boldsymbol{q}(t_k) + \dot{\boldsymbol{q}}(t_k)\Delta t$$

Quando il manipolatore è ridondante lo Jacobiano assume la forma di matrice rettangolare bassa. Questo pone un problema significativo nella ricerca della soluzione a $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\boldsymbol{v}$, perché esisterà più di una soluzione.

Assegnata la velocità dell'organo terminale $\boldsymbol{v}$ e lo Jacobiano del manipolatore $\boldsymbol{J}(\boldsymbol{q})$, un criterio per scegliere la soluzione può essere quello di adottare le $\dot{\boldsymbol{q}}$ che soddisfano $\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$ e che minimizzano localmente il funzionale di costo quadratico della velocità: $g(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\mathbf{W}\dot{\boldsymbol{q}}$, ove $\mathbf{W}$ è un'opportuna matrice di peso $(n \times n)$ simmetrica e definita positiva.

Il problema si può risolvere con il metodo dei *moltiplicatori di Lagrange*, introducendo il funzionale di costo modificato:

$$g(\dot{\boldsymbol{q}},\lambda) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\mathbf{W}\dot{\boldsymbol{q}} + \lambda^{T}(\boldsymbol{v} - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}),$$

in cui λ è un vettore incognito $(r \times 1)$ di moltiplicatori che permette di incorporare il vincolo del funzionale da minimizzare. La soluzione cercata deve pertanto soddisfare le seguenti condizioni di ottimo:

$$\left(\frac{\partial g}{\partial \dot{\boldsymbol{q}}}\right)^{T} = \mathbf{0} \quad \text{e} \quad \left(\frac{\partial g}{\partial \lambda}\right)^{T} = \mathbf{0}$$

Dalla prima condizione si ricava: $\boldsymbol{W}\dot{\boldsymbol{q}} - \boldsymbol{J}^{T}\lambda = \mathbf{0} \Rightarrow \dot{\boldsymbol{q}} = \boldsymbol{W}^{-1}\boldsymbol{J}^{T}\lambda$, che è una soluzione di minimo in quanto $\frac{\partial^2 g}{\partial \dot{\boldsymbol{q}}^2} = \boldsymbol{W}$ con $\boldsymbol{W}$ definita positiva. La seconda condizione restituisce il vincolo simile alla cinematica diretta: $\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$. Combinando le due condizioni si ottiene:

$$\boldsymbol{v} = \boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^{T}\lambda$$

dove, nell'ipotesi che $\boldsymbol{J}$ abbia rango pieno, anche $\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^{T}$ è una matrice di rango pieno m , quindi quadrata con relativa possibilità di calcolarne l'inversa. Risolvendo per λ e sostituendo il risultato in $\dot{\boldsymbol{q}} = \boldsymbol{W}^{-1}\boldsymbol{J}^{T}\lambda$ si ottiene la soluzione completa:

$$\dot{\boldsymbol{q}} = \boldsymbol{W}^{-1}\boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^{T})^{-1}\boldsymbol{v}.$$

Questa equazione soddisfa il vincolo della cinematica differenziale, e se ci si pone nel caso particolare in cui $\boldsymbol{W} = \boldsymbol{I}$ la soluzione si semplifica: $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\boldsymbol{v}$, dove :

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{T}(\boldsymbol{J}\cdot\boldsymbol{J}^{T})^{-1}$$

prende il nome di matrice pseudo-inversa destra, ed è l'unica matrice che soddisfa le proprietà di Moore-Penrose, portando la soluzione della cinematica inversa verso quella a minore velocità e quindi minor energia cinetica tra le infinite possibili. [13]

Problematiche dei metodi di inversione cinematica:

Esponendo i vari algoritmi di inversione cinematica presi in considerazione si individueranno varie caratteristiche e problematiche: problematiche di ripetibilità, errore di posizionamento e orientamento, errore dovuto all'integrazione numerica (*joint drift*), tempo di calcolo e applicabilità al caso real-time, contrasti tra task primario e altri task, casi di limiti di giunto e di velocità che vorranno essere soddisfatti in modo deterministico.

Uno dei maggiori problemi, poi, consiste nelle singolarità. Si descrive ora di cosa si tratta. Esistono tre tipi di singolarità in cui si può incorrere: cinematiche, di rappresentazione e algoritmiche. Una configurazione dei giunti $\boldsymbol{q}$ di un robot è detta singolare se in corrispondenza di essa lo Jacobiano geometrico diminuisce di rango. Considerando il ruolo che ha lo Jacobiano nella generazione delle velocità dell'end effector, è facile osservare che in una configurazione singolare è impossibile generare delle velocità in certe direzioni con conseguente perdita di mobilità.

In presenza di singolarità cinematiche si ha: perdita di mobilità (non sarà possibile imporre leggi di moto arbitrarie), possibilità di infinite soluzioni al problema cinematico inverso, velocità elevate nello spazio dei giunti. Le singolarità cinematiche possono essere ai confini dello spazio di lavoro (cioè quando il robot è completamente esteso o raccolto su sé stesso, possono essere evitate pianificando traiettorie al di fuori di queste zone) o interne allo spazio operativo (limitano la mobilità del robot proprio dove lo si vorrebbe utilizzare e sono più difficili da trattare, anche se alcuni casi sono di facile intuizione, come quello di giunti allineati che possono causare difficoltà di movimento lungo la direzione di allineamento).

Le singolarità algoritmiche possono occorrere quando si vuole risolvere il problema dell'inversione cinematica inserendo un compito ausiliario (non si hanno se il compito ausiliario si pone già in termini di angoli ai giunti).
Le singolarità di rappresentazione sono quelle in cui diventa singolare la matrice dipendente dalla rappresentazione utilizzata per l'orientamento $\mathbf{T}(\Phi)$ che lega la velocità angolare $\boldsymbol{\omega}$ e la derivata del vettore che esprime l'orientamento:
$\boldsymbol{\omega} = \boldsymbol{T}(\boldsymbol{\Phi})\dot{\boldsymbol{\Phi}}$.

4.2 Algoritmi CLIK

Si può procedere ad elencare gli algoritmi di inversione cinematica secondo la classificazione presentata in [14], in cui dapprima si presentano i metodi che prevedono l'utilizzo della pseudo-inversa o trasposta dello Jacobiano, per poi passare ai metodi che prevedono l'attuazione deterministica di un task aggiuntivo nel caso vi sia ridondanza, per infine passare al concetto di task secondari.

Algoritmi CLIK con uso della pseudo-inversa dello Jacobiano

Gli algoritmi che prevedono l'utilizzo della pseudo-inversa partono dal risultato descritto in 4.1 per cui si può rispettare il legame tra *Task Space* e spazio dei giunti con minimizzazione dell'energia spesa secondo la relazione $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{v}$, in cui la definizione di $\boldsymbol{J}^{\dagger}(\boldsymbol{q}) = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1}$ garantisce che si minimizzi la norma $\frac{1}{2}\|\dot{\boldsymbol{q}}\|^2$ e fa in modo che nel caso di Jacobiano a rango pieno sia calcolabile numericamente usando la decomposizione a valori singolari (SVD).

Tuttavia questo tipo di inversione non garantisce la ripetibilità (il che non sarà un problema rilevante nella presente analisi), non garantisce di evitare le singolarità anche se le velocità sono mantenute al minimo, e può portare a non essere precisi se ci si vuole spostare compiendo distanze notevoli (comandi punto a punto) per via del fatto che il metodo si basa sulla definizione dello Jacobiano che è funzione della configurazione. Se la configurazione varia molto è difficile compensare errori di

posizionamento. Per lo stesso principio vi possono essere errori di approssimazione dovuti all'integrazione numerica a tempo discreto, per cui il calcolo delle velocità ai giunti è ottenuto con riferimento all'inversa dello Jacobiano valutata in corrispondenza delle posizioni dei giunti all'istante precedente. Ne consegue che le velocità di giunto calcolate non coincidono con quelle che si avrebbero a tempo continuo. Questo fenomeno di deriva è tanto più grande quanto maggiore è il Δt tra due istanti campionati [13] [15].

Per quanto riguarda le singolarità si vedranno più avanti tecniche per affrontarle. Si introduce invece ora una metodologia per rimediare agli ultimi due problemi per cui alle variabili di giunto calcolate corrisponde una postura dell'organo terminale diversa da quella desiderata nello spazio operativo: la CLIK (Closed Loop Inverse Kinematics). Essa prevede uno schema di soluzione che tenga conto dell'errore nello spazio operativo tra posizione e orientamento desiderati e posizione e orientamento calcolati. Si può definire tale errore e calcolarne la derivata temporale:

$$\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x} \quad \Rightarrow \quad \dot{\boldsymbol{e}} = \dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}} \underbrace{=}_{\dot{\boldsymbol{x}}=\boldsymbol{J}_A(\boldsymbol{q})\dot{\boldsymbol{q}}} \dot{\boldsymbol{x}}_d - \boldsymbol{J}_A(\boldsymbol{q})\dot{\boldsymbol{q}}$$

Si può usare tale errore per chiudere un anello di controllo in ogni istante del task come indicato nella seguente Figura.

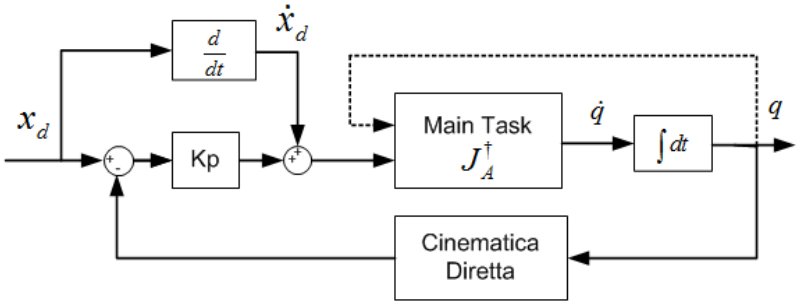


Figura 4.1 : CLIK con inversa dello Jacobiano

La traduzione formale dell'algoritmo risulterà quindi:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_A^{\dagger}(\boldsymbol{q})\left(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_p \boldsymbol{e}\right)$$

Si ha che:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\left(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x})\right) \Rightarrow \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}}_d + \boldsymbol{J}^{\dagger}(\boldsymbol{q})\left(\boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x})\right)$$
$$\Rightarrow \quad \boldsymbol{0} = (\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x}) \Rightarrow \boldsymbol{0} = \dot{\boldsymbol{e}} + \boldsymbol{K}_p \boldsymbol{e}$$

Da quest'ultima relazione si evince che la dinamica dell'errore è data da un'equazione del prim'ordine per cui, con $\boldsymbol{K}_p$ definita positiva, si ha asintotica stabilità e l'errore tende a zero lungo la traiettoria con una velocità di convergenza che dipende dagli autovalori della matrice $\boldsymbol{K}_p$. Per i sistemi del prim'ordine più $\boldsymbol{K}_p$ contiene autovalori elevati, più rapida sarà la convergenza senza problemi di oscillazione. Tuttavia si dovrà tenere conto della limitazione in banda imposta dal campionamento, essendo un sistema a tempo discreto.
Inoltre lo schema a blocchi evidenzia la presenza di un integratore nella catena di andata e quindi, per un riferimento costante, garantisce un errore a regime nullo. L'azione in avanti fornita da $\dot{\boldsymbol{x}}_d$ in caso di riferimento variabile assicura, inoltre, che l'errore si mantenga nullo (in caso di $\boldsymbol{e}(0) = 0$) lungo l'intera traiettoria.

La Figura seguente mostra l'andamento della norma dell'errore di posizione e angolare nel caso in cui la posa a inizio task non coincida con quella iniziale del robot. Si vede che in assenza di saturazioni di velocità l'algoritmo CLIK porta a una convergenza a zero dell'errore molto più rapida (fin dal primo istante) di quella della semplice pseudo-inversione e quindi può costituire una valida soluzione per problemi di deriva della soluzione.

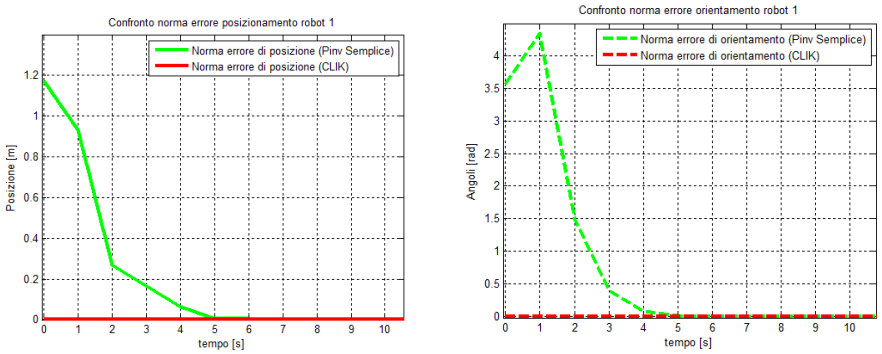


Figura 4.2 : Norma di errori di posizione e di orientamento a confronto: pseudo-inversione semplice e CLIK nel caso di posizione di partenza diversa da quella iniziale del task

Si ha infine la possibilità di dare della CLIK un'interpretazione fisica per via della dualità cineto-statica: sia $\boldsymbol{\tau}$ il vettore coppie ai giunti $(n \times 1)$ e f il vettore forza sull'end effector espresso nello spazio cartesiano $(m \times 1)$. Si ha: $\boldsymbol{\tau} = \mathbf{J}^T \cdot \mathbf{f}$. In tal caso $\mathbf{K_P} \cdot \mathbf{e}$ si può vedere come la forza elastica applicata all'end effector di una struttura ideale senza massa e con smorzamento unitario che tende a spostarlo verso la traiettoria desiderata [13] [16].

Per contro, oltre alla possibile presenza di singolarità cinematiche che si ha in questa forma di CLIK, si possono avere tempi di calcolo lunghi (per via sia della presenza dell'anello chiuso, sia per via della presenza della pseudo-inversa). Esiste un approccio, riportato ora di seguito, che permette di evitare la pseudo-inversione.

Algoritmi CLIK con uso della trasposta dello Jacobiano

Si può dimostrare che è anche possibile utilizzare un anello del tipo riportato in Figura 4.3:

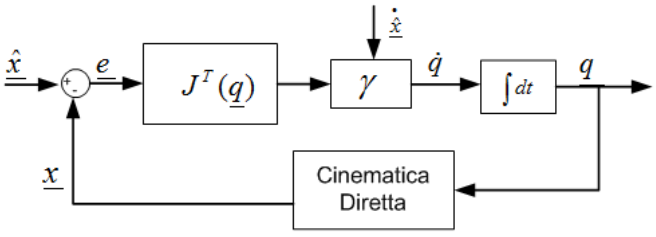


Figura 4.3 : Schema a blocchi dell'algoritmo generale con la trasposta dello Jacobiano

Da cui risulta:

$$\dot{\boldsymbol{q}} = \gamma \cdot \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{e} \quad \text{con} \quad \gamma = \boldsymbol{K} + (\boldsymbol{e}^T \cdot \dot{\boldsymbol{x}}_d)(\boldsymbol{e}^T \boldsymbol{J}\boldsymbol{J}^T \boldsymbol{e})^{-1}$$

Ne consegue:

$$\dot{\boldsymbol{q}} = \boldsymbol{K} \cdot \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{e} + \boldsymbol{e}^T \cdot \dot{\boldsymbol{x}}_d \cdot (\boldsymbol{e}^T \boldsymbol{J}\boldsymbol{J}^T \boldsymbol{e})^{-1} \cdot \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{e}$$

E da qui si può dimostrare la convergenza dell'errore a zero per $\boldsymbol{K} > 0$ considerando che la funzione dell'errore secondo Lyapunov $\upsilon = \frac{1}{2}\boldsymbol{e}^T \boldsymbol{e}$ assume derivata temporale negativa [16]. Sempre nello stesso articolo e in [13] si dimostra che l'algoritmo è semplificabile come:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{K} \cdot \boldsymbol{e}$$

Infatti se si sceglie $\upsilon = \frac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{K}\mathbf{e}$ ($\mathbf{K}$ definita positiva) e si deriva, si ottiene: $\dot{\upsilon} = \mathbf{e}^{\mathrm{T}}\mathbf{K}\dot{\mathbf{x}}_{\mathrm{d}} - \mathbf{e}^{\mathrm{T}}\mathbf{K}\dot{\mathbf{x}} = \mathbf{e}^{\mathrm{T}}\mathbf{K}\dot{\mathbf{x}}_{\mathrm{d}} - \mathbf{e}^{\mathrm{T}}\mathbf{K}\mathbf{J}\dot{\mathbf{q}}$. Se si sceglie $\dot{\boldsymbol{q}} = \boldsymbol{J}^T(\boldsymbol{q}) \cdot \boldsymbol{K} \cdot \boldsymbol{e}$, si ha:
$\dot{\upsilon} = \boldsymbol{e}^T \boldsymbol{K}\dot{\boldsymbol{x}}_d - \boldsymbol{e}^T \boldsymbol{K}\boldsymbol{J}\boldsymbol{J}^T \boldsymbol{K}\boldsymbol{e}$ che è chiaramente definita negativa nel caso di riferimento costante ($\dot{\boldsymbol{x}}_d = \mathbf{0}$). La condizione $\dot{\upsilon} < 0$ con $\upsilon > 0$ implica che le traiettorie del sistema convergano uniformemente ad $\boldsymbol{e} = \mathbf{0}$ (sistema asintoticamente stabile). Anche il caso in cui $\mathcal{N}(\boldsymbol{J}^T) \neq \mathbf{0}$ si genera stallo solo se la posizione assegnata all'organo terminale non è effettivamente raggiungibile a partire dalla configurazione corrente. Invece nel caso in cui $\dot{\boldsymbol{x}}_d \neq \mathbf{0}$, il primo termine a secondo membro della $\dot{\upsilon} = \boldsymbol{e}^T \boldsymbol{K}\dot{\boldsymbol{x}}_d - \boldsymbol{e}^T \boldsymbol{K}\boldsymbol{J}\boldsymbol{J}^T \boldsymbol{K}\boldsymbol{e}$ non viene cancellato e nulla si può dire sul segno. Ciò comporta che non è possibile ottenere l'asintotica stabilità lungo la traiettoria, pur limitando l'errore superiormente in norma: risulta tanto più piccolo quanto più grande è la norma della matrice dei guadagni $\boldsymbol{K}$ (che quindi conviene prendere il più alto possibile in rispetto del tempo di campionamento) e tanto più piccolo è $\|\dot{\boldsymbol{x}}_d(t)\|$. Si ha comunque una notevole semplificazione dell'algoritmo con la possibilità di tenere sotto controllo l'errore di inseguimento e con il vantaggio di avere un tempo di calcolo molto contenuto.

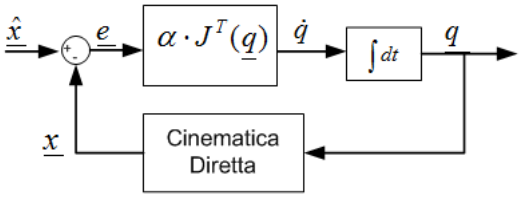


Figura 4.4 : Schema a blocchi dell'algoritmo semplificato con la trasposta dello Jacobiano

4.3 Algoritmi con Task Space Augmentation

Si tratta di metodi per cui è possibile sfruttare la ridondanza aggiungendo un task ausiliario $f_y(q) = y \in \mathbb{R}^l$ con $l \leq n - m$. Questo permette di formulare il problema come la ricerca di:

$$\boldsymbol{x}_{augm} = \begin{bmatrix} \boldsymbol{x}_d \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{q}) \\ \boldsymbol{f}_y(\boldsymbol{q}) \end{bmatrix} \Rightarrow \dot{\boldsymbol{x}}_{augm} = \begin{bmatrix} \boldsymbol{J}(\boldsymbol{q}) \\ \boldsymbol{J}_y(\boldsymbol{q}) = \dfrac{\partial \boldsymbol{f}_y}{\boldsymbol{q}} \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{J}_{augm}\, \dot{\boldsymbol{q}}$$

Le dimensioni di $\boldsymbol{J}_{augm}$ saranno pertanto $(m + l) \times n$ e quindi si potrà adottare un'inversione cinematica ad esempio del tipo: $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}_{augm}\, \dot{\boldsymbol{x}}_{augm}$, anche inseribile all'interno di una CLIK descritta dalla seguente Figura:

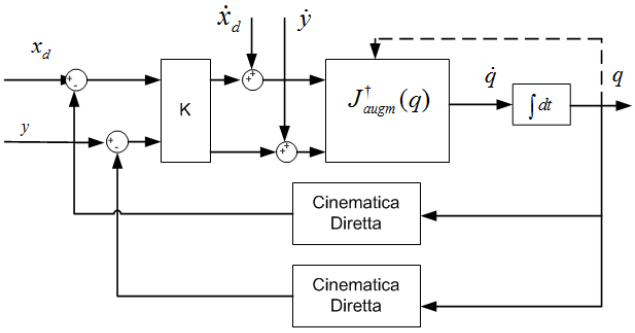


Figura 4.5 : Schema a blocchi dell'algoritmo CLIK con Jacobiano aumentato

I vantaggi di tale approccio consistono nel poter imporre delle funzioni aggiuntive definibili nel *Task Space* e che saranno rispettate. Gli svantaggi saranno però quelli di generare possibili singolarità algoritmiche e, nella formulazione appena riportata,

possibili contrasti con il task principale, che è posto "allo stesso livello". Affinché essi non insorgano, dato $\boldsymbol{J}_{augm} = \begin{bmatrix} \boldsymbol{J} \\ \boldsymbol{J}_y \end{bmatrix}$, si deve avere sempre che le righe di $\boldsymbol{J}$ e di $\boldsymbol{J}_y$ siano linearmente indipendenti, ossia: [14]

$$\mathbb{R}(\boldsymbol{J}^T) \cap \mathbb{R}(\boldsymbol{J}_y^T) = \emptyset$$

Un'applicazione è ad esempio quella di aggirare ostacoli (modellabili come volumi convessi nello spazio 3D), che possono generare problemi anche con una traiettoria all'end effector tale da evitarli. Infatti l'urto potrebbe ugualmente avvenire con uno dei link del manipolatore. Quindi la minima distanza dei link da tali ostacoli dovrà esser maggiore di un limite prestabilito, se ciò non accadrà la soluzione verrà modificata. Sia $\widehat{\boldsymbol{d}}_0$ il limite di distanza, $\boldsymbol{c}$ la posizione cartesiana dell'ostacolo, $\boldsymbol{p}_0$ il punto più vicino all'ostacolo o comunque il punto di controllo quale può essere il gomito. Se $\boldsymbol{d}_0 = (\boldsymbol{p}_0 - \boldsymbol{c}) < \widehat{\boldsymbol{d}}_0$ serve attivare il vincolo nell'algoritmo. La definizione del vincolo parte da quella dell'errore dato dalla distanza che è troppo piccola rispetto a quella desiderata:

$$\boldsymbol{e}_0 = \frac{1}{2}\left(\widehat{\boldsymbol{d}}_0{}^2 - \boldsymbol{d}_0^T\boldsymbol{d}_0\right) \Rightarrow \dot{\boldsymbol{e}}_0 = \widehat{\boldsymbol{d}}_0\dot{\widehat{\boldsymbol{d}}}_0 - \boldsymbol{d}_0^T\dot{\boldsymbol{c}} - \boldsymbol{j}_{d0}^T\dot{\boldsymbol{q}} \quad \text{con } \boldsymbol{j}_{d0}^T = \boldsymbol{d}_0^T\boldsymbol{J}_{p0} = \boldsymbol{d}_0^T\left[\frac{\partial \boldsymbol{P}_0}{\partial \boldsymbol{q}}\right]$$

Si avrà quindi un vettore aumentato degli errori di dimensioni $\big((m+1)\times 1\big)$ nello spazio Cartesiano, il cui ultimo componente è $\boldsymbol{e}_0$ e uno Jacobiano aumentato $\big((m+1)\times n\big)$ la cui ultima riga è $\boldsymbol{j}_{d0}^T$. Se si inseriscono tali quantità in un anello chiuso quale quello di Figura 4.5, il moto di quei *d.o.f.* che determinerebbero l'impatto è interrotto e di fatto il link in questione è costretto a muoversi tangenzialmente attorno alla sfera immaginaria centrata in $\boldsymbol{c}$ e con raggio $\widehat{\boldsymbol{d}}_0$.

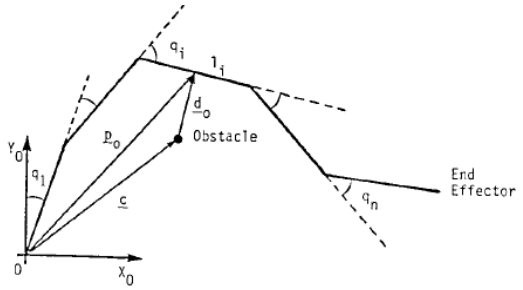


Figura 4.6 : Geometria di un manipolatore planare e relativo posizionamento di un ostacolo

Altra applicazione del tutto analoga è quella che permette di evitare il fine corsa di un i-esimo giunto. Se si ha $q_{i,min} \leq q_i \leq q_{i,max}$, definendo $d_q = q_i - q_{i,min}$ o, a seconda del caso $d_q = q_{i,max} - q_i$, si ha:

$$\boldsymbol{e}_q = \hat{d}_q - d_q \Rightarrow \dot{\boldsymbol{e}}_q = \dot{\hat{d}}_q - \boldsymbol{u}_i^T \dot{\boldsymbol{q}} \text{ con } \boldsymbol{u}_i^T = \underbrace{(0 \;\; 0 \;\; \ldots \;\; \pm 1 \;\; \ldots \;\; 0)}_{\substack{+ \text{ si applica a } q_{i,min} \\ - \text{ si applica a } q_{i,max}}}$$

Ancora una volta si arriva a un vettore degli errori aumentato che comprenderà una componente additiva $\boldsymbol{e}_q$ e uno Jacobiano aumentato che comprenderà una riga data da $\boldsymbol{u}_i^T$. Se le quantità così definite saranno usate in un algoritmo di inversione quale la CLIK, il moto di quel dof che si sta avvicinando al limite è interrotto e mantenuto pari al limite.

Alla luce di questi esempi di vincolo tramite task augmentation, si nota che è necessario un controllo di alto livello per pianificare la traiettoria dal punto di vista dei vincoli, per attivarli/disattivarli correttamente. Infatti si attiva un vincolo solo se si supera il limite impostato $\hat{d}_0$ *e* q_0 dato che a tenere attivo un vincolo a lungo si generano più facilmente problemi sovra-condizionati o contrasti con il task all'end effector. Si deve quindi predisporre un livello di switch algoritmici che a rigore potrebbero prevenire la convergenza, ma che di fatto sono assolutamente necessari.
Inoltre l'attivazione-disattivazione genera un transitorio non modellato che potrebbe portare al colpire lo stesso l'ostacolo per cui si deve sempre tenere il limite proporzionale alla velocità dell'operazione [16] [17].

4.4 Algoritmi con Task Secondari Mediante Proiezione nel Null Space

Si è visto che è possibile sfruttare la ridondanza per effettuare compiti aggiuntivi. Ma se ci si ferma alla formalizzazione del punto precedente si possono avere problemi di contrasto tra task all'end effector e task aggiuntivi, oltre al fatto che se emergono singolarità algoritmiche tutti i task, "posti allo stesso livello", ne risentono. Si introduce quindi la possibilità di specificare task aggiuntivi con una priorità inferiore, che quindi agiscano non interferendo con il task principale (generalmente proprio quello di inseguimento di traiettoria dell'organo terminale). Per fare ciò, come anticipato in 4.1, si sfrutta il null-space dello Jacobiano [18].

E' possibile proiettare nel nullo dello Jacobiano un task espresso come gradiente di una funzione obiettivo nelle variabili di giunto ($\dot{\boldsymbol{q}}_a$):

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}}_d + \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\right) \underbrace{\dot{\boldsymbol{q}}_a}_{K_a\left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)}$$

Naturalmente si può utilizzare anche chiudendo un anello di controllo cinematico:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\left(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x})\right) + \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\right) \underbrace{\dot{\boldsymbol{q}}_a}_{K_a\left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)}$$

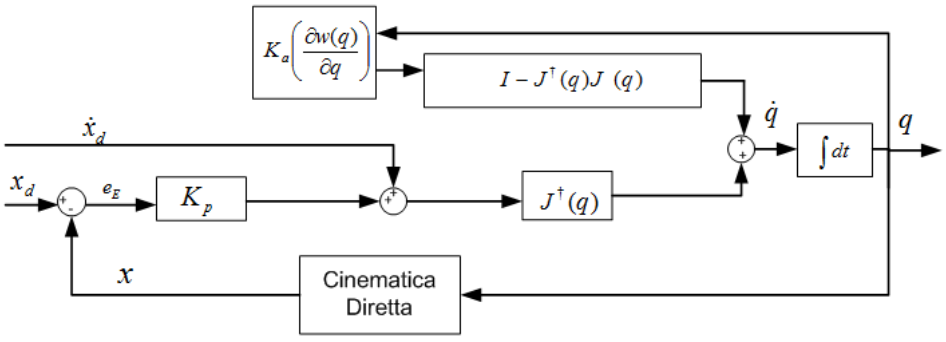


Figura 4.7 : Schema a blocchi di una CLIK con task secondario (Projected Gradient)

Se invece si vuole utilizzare una funzione obiettivo scritta nelle variabili di *Task Space* si può proiettare il task aggiuntivo presentato nel paragrafo precedente nel nullo, ottenendo:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}}_d + \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\right)\boldsymbol{J}_y^T(\boldsymbol{q})\boldsymbol{K}_y\boldsymbol{e}_y$$

SI può estendere al caso di CLIK (rappresentato in Figura 4.8) in cui si assicurerà errore nullo all'end effector se $\boldsymbol{e}_E(0) = 0$ ed errore del task di vincolo limitato:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\left(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x})\right) + \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\right)\boldsymbol{J}_y^T(\boldsymbol{q})\boldsymbol{K}_y\boldsymbol{e}_y$$

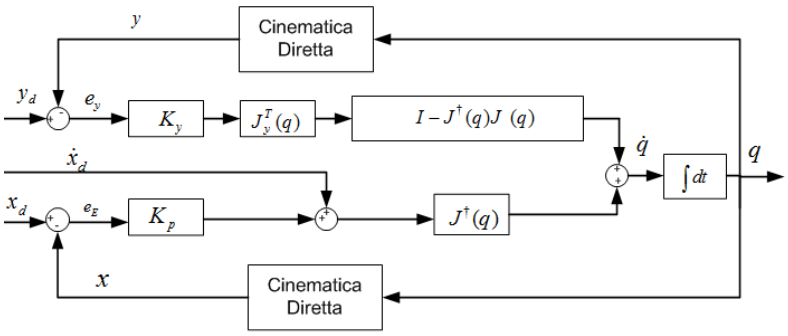


Figura 4.8 : Schema a blocchi di una CLIK con task secondario (specificato nel Task Space)

Con questo metodo si possono introdurre task a priorità inferiore. Lo svantaggio è che il calcolo della pseudo-inversa è computazionalmente rilevante (è chiaro che se si svolge questo calcolo per il task secondario e si vuole utilizzare un anello di controllo, si privilegerà un anello con pseudo-inversa e non quello con la trasposta, dato che si dispone già della pseudo-inversa calcolata per proiettare nel nullo). Per velocizzare il calcolo è possibile tuttavia, se lo Jacobiano è a rango pieno, ricorrere a metodi di gradiente ridotto che porterà ad algoritmi computazionalmente più rapidi [14].

Si noti poi che è possibile in ogni momento attivare un'inversione di importanza di due task, ad esempio se ad un certo istante diventerà più importante rispettare il vincolo dato da y rispetto ad ogni altra cosa si potrà scrivere

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_y^T(\boldsymbol{q})\boldsymbol{K}_y\boldsymbol{e}_y + [\boldsymbol{I} - \boldsymbol{J}^\dagger(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\,]\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{K}_p\boldsymbol{e}$$

E' infine possibile aggiungere altri task con un ordine decrescente di importanza secondo una struttura sistematica (task priority strategy),come descritto in [19].

Ogni task dovrà essere espresso in termini di variabili di giunto con quindi la possibilità di definire uno Jacobiano. Partendo da questo si deriverà una soluzione ricorsiva che semplifica il problema di una programmazione ad alto livello della ridondanza del sistema permettendo all'utente di specificare successivamente tutti i task desiderati.

Si consideri un generico task i-esimo, caratterizzato da: $\dot{\mathbf{x}}^{\mathrm{i}} = \mathbf{J}^{\mathrm{i}}(\mathbf{q})\dot{\mathbf{q}}$ e che può riferirsi al posizionamento dell'end effector o alla distanza dai limiti di giunto, o alla distanza da ostacoli. Si consideri poi:

$$\mathbf{P}_{\mathrm{a}}^{\mathrm{i}} = \mathbf{I} - \mathbf{J}_{\mathrm{augm}}^{\mathrm{i}\dagger}\mathbf{J}_{\mathrm{augm}}^{\mathrm{i}}$$

Che è il proiettore nel nullo dello Jacobiano aumentato $\mathbf{J}_{\mathrm{augm}}^{\mathrm{i}} = \left[\mathbf{J}^1\,\mathbf{J}^2\,\ldots\mathbf{J}^{\mathrm{i}}\right]^{\mathrm{T}}$.
Da qui si ha che la soluzione in velocità sarà:

$$\dot{\mathbf{q}}^{\mathrm{i}} = \dot{\mathbf{q}}^{\mathrm{i}-1} + \tilde{\mathbf{J}}^{\mathrm{i}\dagger}\left(\dot{\mathbf{x}}^{\mathrm{i}} - \mathbf{J}^{\mathrm{i}}\dot{\mathbf{q}}^{\mathrm{i}-1}\right)$$

In cui $\dot{\mathbf{q}}^1 = \mathbf{J}^{1\#}\dot{\mathbf{x}}^1$ e $\tilde{\mathbf{J}}^{\mathrm{i}} = \mathbf{J}^{\mathrm{i}}\mathbf{P}_{\mathrm{a}}^{\mathrm{i}-1}$ e grazie a cui tale i-esimo task sarà eseguito con priorità inferiore rispetto al precedente task $\mathrm{i}-1$, solo nelle direzioni che non disturbano l'$(\mathrm{i}-1)$-esimo con maggiore priorità.

4.5 Considerazioni sugli Algoritmi di Inversione Tradizionali

4.5.1 Metodi per Affrontare le Singolarità

Si è visto che uno dei problemi lasciati scoperti dai generici algoritmo presentati finora è la possibilità che si incorra in singolarità. Nei casi con task secondario si ha che eventuali singolarità algoritmiche non inficiano la prestazione del task principale, ma comunque restano sul task secondario stesso e restano le singolarità di rappresentazione e cinematiche per entrambi i task.

Proiezione nel nullo del gradiente della funzione di manipolabilità

Una prima soluzione al problema di singolarità cinematiche sul task principale può consistere nell'utilizzare un compito secondario proprio per far assumere al robot una configurazione che sia il più lontano possibile dalle singolarità. Questo si ottiene per esempio proiettando nel nullo dello Jacobiano del task a priorità maggiore il gradiente di una funzione del tipo:

$$\omega(\boldsymbol{q}) = \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))}$$

Questa funzione rappresenta una misura di manipolabilità: essa si annulla in corrispondenza di una configurazione singolare, per cui massimizzandola attraverso il gradiente $\mathbf{q}_a = +k_a \frac{\partial \omega}{\partial \mathbf{q}}$, si utilizza la ridondanza per allontanarsi dalle singolarità (se $\omega(\mathbf{q})$ è grande si avranno ellissoidi di manipolabilità poco schiacciati, senza quindi direzioni in cui il movimento tende a risultare difficoltoso, come appunto avviene nel caso di singolarità cinematiche).

Metodo dell'inversa ai minimi quadrati smorzata dello Jacobiano

Un altro approccio è quello riportato in [20] :

Per investigare le singolarità si considera che lo Jacobiano può essere scritto secondo la seguente decomposizione ai valori singolari:

$$\boldsymbol{J} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

Con $\mathbf{U}$ $(\mathrm{m} \times \mathrm{m})$ matrice ortonormale dei vettori singolari di output $\mathbf{u}_\mathrm{i}$, $\mathbf{V}$ $(\mathrm{n} \times \mathrm{n})$ matrice ortonormale dei vettori singolari in input $\boldsymbol{v}_\mathrm{i}$, e $\boldsymbol{\Sigma} = (\mathbf{S} \quad \mathbf{O})$ $(\mathrm{m} \times \mathrm{n})$ matrice in cui la sottomatrice diagonale $(\mathrm{m} \times \mathrm{m})$ contiene i valori singolari σ_i di $\mathbf{J}$.

Poi si considera la generica soluzione in della cinematica inversa con task secondario:

$$\dot{\boldsymbol{q}}_{PI} = \mathbf{J}^{\dagger}\dot{\boldsymbol{x}} + (\boldsymbol{I} - \mathbf{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}_a$$

Si ha che $\dot{\mathrm{q}}_\mathrm{a}$ può essere ricondotta alla specificazione di un task di vincolo $(\mathrm{n} - \mathrm{m})$-dimensionale espresso dal vettore velocità $\dot{\mathbf{y}} = \mathbf{J}_\mathrm{y}(\mathbf{q})\dot{\mathbf{q}}$ $\left(\text{con } \mathbf{J}_\mathrm{y} = \frac{\partial \mathbf{y}}{\partial \mathbf{q}}\right)$:

$$\dot{\boldsymbol{q}}_a = \left(\boldsymbol{J}_y(\boldsymbol{I} - \mathbf{J}^{\dagger}\boldsymbol{J})\right)^{\dagger}\left(\dot{\boldsymbol{x}}_C - \boldsymbol{J}_y\mathbf{J}^{\dagger}\dot{\boldsymbol{x}}\right)$$

Quindi si considera che la soluzione $\dot{\mathbf{q}}_\mathrm{PI} = \mathbf{J}^{\dagger}\dot{\mathbf{x}} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\dot{\mathbf{q}}_\mathrm{a}$ può essere riscritta secondo la scomposizione a valori singolari:

$$\dot{\mathbf{q}}_\mathrm{PI} = \sum_{\mathrm{i}=1}^{\mathrm{r}} \frac{\left(\mathbf{u}_\mathrm{i}^\mathrm{T}\dot{\mathbf{x}}\right)}{\sigma_\mathrm{i}}\mathbf{v}_\mathrm{i} + \sum_{\mathrm{i}=\mathrm{r}+1}^{\mathrm{n}} \left(\mathbf{v}_\mathrm{i}^\mathrm{T}\dot{\mathbf{q}}_\mathrm{a}\right)\mathbf{v}_\mathrm{i}$$

Da ciò si vede che vicino alle singolarità vi sono componenti di velocità che derivano a infinito $(\sigma_\mathrm{i} \to 0)$, mentre se non fosse per quello l'errore di ricostruzione sarebbe :

$$\dot{\mathbf{x}} - \mathbf{J}\dot{\mathbf{q}}_{\mathrm{PI}} = \sum_{i=r+1}^{m} (\mathbf{u}_i^T \dot{\mathbf{x}})\mathbf{u}_i$$

Ergo dovuto solamente a perdite di mobilità imputabili a singolarità cinematiche ed allineato con le velocità infattibili.

Una soluzione che invece è robusta alle singolarità si basa sull'uso della pseudo-inversa ai minimi quadrati smorzata da $\lambda \in \mathbb{R}$:

$$\mathbf{J}^{\#} = \mathbf{J}^T(\mathbf{J} \cdot \mathbf{J}^T + \lambda^2 \mathbf{I})^{-1} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \mathbf{v}_i \mathbf{u}_i^T$$

Da cui:

$$\dot{\mathbf{q}}_{\mathrm{DSL}} = \mathbf{J}^{\#}\dot{\mathrm{x}} + (\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\dot{\mathbf{q}}_a = \sum_{i=1}^{r} \frac{\sigma_i(\mathbf{u}_i^T \dot{\mathbf{x}}) + \lambda^2(\mathbf{v}_i^T \dot{\mathbf{q}}_a)}{\sigma_i^2 + \lambda^2} \mathbf{v}_i + \sum_{i=r+1}^{n} (\mathbf{v}_i^T \dot{\mathbf{q}}_a)\mathbf{v}_i$$

Il fattore di smorzamento assicura di evitare derive di velocità, ma a costo di un aumento dell'errore di ricostruzione, dato che:

$$\dot{\mathbf{x}} - \mathbf{J} \cdot \dot{\mathbf{q}}_{\mathrm{DSL}} = \sum_{i=1}^{r} \lambda^2 \frac{(\mathbf{u}_i^T \dot{\mathbf{x}}) + \sigma_i(\mathbf{v}_i^T \dot{\mathbf{q}}_a)}{\sigma_i^2 + \lambda^2} \mathbf{u}_i + \sum_{i=r+1}^{m} (\mathbf{u}_i^T \dot{\mathbf{x}})\mathbf{u}_i$$

Vi sono termini che aumentano l'errore, il cui impatto diminuisce se si riduce il fattore di smorzamento lontano dalle singolarità o se si evita del tutto lo smorzamento lungo componenti di velocità fattibili. Si può quindi raffinare il tutto con coefficienti di smorzamento variabili e operando filtraggio numerico delle componenti di velocità [20]. L'alternativa è includere l'algoritmo in una CLIK. E' possibile, con un procedimento di questo tipo, anche affrontare le singolarità algoritmiche, pesando anche lo Jacobiano del task secondario.

Utilizzo dei quaternioni per evitare singolarità di rappresentazione

Resterebbero le singolarità di rappresentazione, ad esempio quelle che si potrebbero generare dando una rappresentazione minima per un task. Un approccio per affrontare questo caso può consistere nell'uso dei quaternioni, come mostrato in [21].

In questo paragrafo si è quindi mostrata una panoramica delle varie soluzioni che si possono integrare negli algoritmi di inversione cinematica qualora si volessero affrontare problemi con singolarità cinematiche, algoritmiche, di rappresentazione.

4.5.2 Agire sull'Accelerazione con Algoritmi di Ordine Superiore

Ulteriori possibilità rispetto ai metodi presentati in precedenza è quella di estendere gli algoritmi a ordine superiore, per poter imporre anche l'accelerazione desiderata. Nella seguente Figura se ne riporta lo schema:

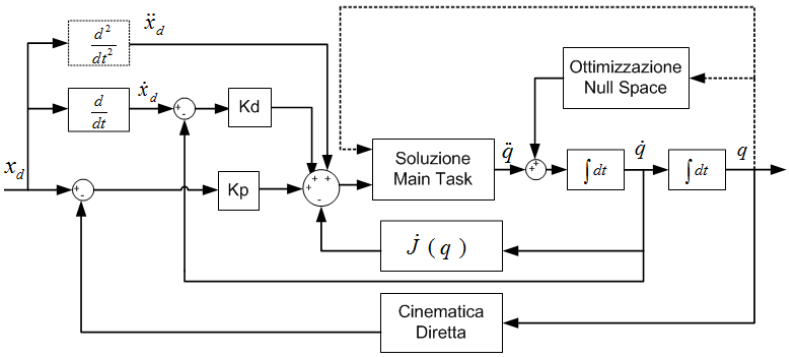


Figura 4.9 : CLIK a livello accelerazione

Se ne ricavano le relazioni : [2] [22]

$$\ddot{\boldsymbol{x}} = \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} \rightarrow \ \ddot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\left(\ddot{\boldsymbol{x}}_{\boldsymbol{d}} - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}\right)$$

da cui, chiudendo l'anello:

$$\ddot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\left(\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{K}_p\boldsymbol{e} + \boldsymbol{K}_v\dot{\boldsymbol{e}}\right)$$

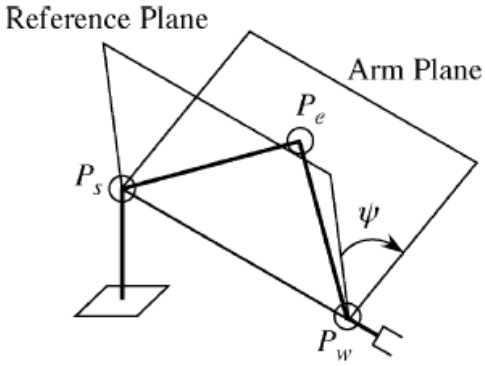
Se con $\boldsymbol{J}^{\#}$ si considera la pseudo-inversa pesata dello Jacobiano definita come visto prima parlando della pesatura $\boldsymbol{J}^{\#} = \boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{J}\cdot\boldsymbol{J}^{\boldsymbol{T}} + \lambda^2\boldsymbol{I})^{-1}$, si arriva a una dinamica dell'errore del tipo:

$$\ddot{\boldsymbol{e}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} = \Delta\left(\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}_A\dot{\boldsymbol{q}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e}\right) \text{ con } \Delta = \boldsymbol{I} - \boldsymbol{J}\cdot\boldsymbol{J}^{\#}$$

Per garantire una corretta convergenza a valori piccoli di errore servirà quindi trovare degli adeguati valori di λ, $\mathbf{K}_P$, $\mathbf{K}_D$ come mostrato, ad esempio in [20].

4.5.3 Cenni sugli Algoritmi Analitici con Parametrizzazione della Ridondanza

Esistono metodi che permettono di risolvere la cinematica inversa anche senza passare per l'inversione dello Jacobiano e stando nel dominio delle posizioni piuttosto che passando dai legami nelle velocità [23]. Naturalmente per derivare una soluzione in forma chiusa della cinematica per un manipolatore ridondante è necessario introdurre una qualche parametrizzazione. Un esempio possibile è quello di prendere come parametro l'angolo di Swivel, definito come nella seguente Figura.



4.10 : Definizione dell'angolo di Swivel ψ

In tal caso, per un robot antropomorfo quale il LWR, si dimostra che l'espressione per gli angoli di giunto in funzione dell'angolo di Swivel può essere scritta come:

$$\tan\theta_1 = \frac{-a_{s22}\sin\psi - b_{s22}\cos\psi - c_{s22}}{-a_{s12}\sin\psi - b_{s12}\cos\psi - c_{s12}}$$

$$\cos\theta_2 = -a_{s32}\sin\psi - b_{s32}\cos\psi - c_{s32}$$

$$\cos\theta_4 == \frac{\left\| {}^0\boldsymbol{x}_7^d - {}^0\boldsymbol{l}_{bs} - {}^0\boldsymbol{R}_7^d\,{}^7\boldsymbol{l}_{wt}\right\|^2 - d_{se}^2 - d_{ew}^2}{2d_{se}d_{ew}}$$

$$\tan\theta_3 = \frac{a_{s33}\sin\psi + b_{s33}\cos\psi + c_{s33}}{-a_{s31}\sin\psi - b_{s31}\cos\psi - c_{s31}}$$

$$\tan\theta_5 = \frac{a_{w23}\sin\psi + b_{w23}\cos\psi + c_{w23}}{a_{w13}\sin\psi + b_{w13}\cos\psi + c_{w13}}$$

$$\cos\theta_6 = a_{w33}\sin\psi + b_{w33}\cos\psi + c_{s32}$$

$$\tan\theta_7 = \frac{a_{w32}\sin\psi + b_{w32}\cos\psi + c_{w32}}{-a_{w31}\sin\psi - b_{w31}\cos\psi - c_{w31}}$$

In cui ogni termine è funzione dell'angolo preso come parametro, della posizione desiderata e da considerazioni geometriche, per la cui definizione si rimanda a [23].
Tuttavia è chiaro che passare per un metodo di questo tipo può essere scomodo se si desidera che il sistema compia dei task secondari, che è l'obiettivo che invece ci si prefigge.

Capitolo 5 Metodi di Inversione per il Sistema Considerato

5.1 Confronto tra gli Algoritmi Tradizionali per l'Applicazione Considerata

Avendo presenti le considerazioni fatte nel capitolo precedente, si possono analizzare criticamente i metodi di inversione cinematica e scegliere quelli da implementare alla luce delle problematiche specifiche del problema.

Metodo	*Caratteristiche*
Pseudo-inversa **IMPLEMENTATO**	- Accuratezza alta solo con $\Delta\boldsymbol{x}$ piccoli tra un istante e il successivo [-] - Tempi di calcolo medi (pseudo inversione) [-] - Nessun vincolo/gestione della ridondanza [-] - Singolarità geometriche e di rappresentazione [-]
CLIK Pseudo-inversa **IMPLEMENTATO**	- Accuratezza alta con $\Delta\boldsymbol{x}$ piccoli e medi [++] - Tempi di calcolo alti con $\Delta\boldsymbol{x}$ medi (pseudo inversione e cicli in anello), medi con $\Delta\boldsymbol{x}$ piccoli dove di fatto coinciderà con una pseudo-inversione semplice [-] - Nessun vincolo/gestione della ridondanza [-] - Singolarità geometriche e di rappresentazione [-]
CLIK Trasposta	- Accuratezza buona con $\Delta\boldsymbol{x}$ piccoli e medi, ma inferiore rispetto alla CLIK con pseudo-inversa [+] - Tempi di calcolo bassi [+] - Nessun vincolo/gestione della ridondanza [-] - Singolarità di rappresentazione [-]

Aggiunta Task Space Augmentation	- Vincoli aggiuntivi da rispettare sfruttando la ridondanza [+] - Possibili contrasti tra task originario e task aggiunto con conseguente degrado delle prestazioni [-] - Possono aggiungersi singolarità algoritmiche che andrebbero a degradare entrambi i task [-]
Aggiunta Task Secondario **IMPLEMENTATO**	- Task aggiuntivi con priorità inferiore rispetto a quella del task principale eseguiti solo se non si instaurano contrasti con esso [+] - Possono aggiungersi singolarità algoritmiche che andrebbero a degradare solo il task secondario. Facili da evitare se si usa il "Projected Gradient Method" [+]
Metodi analitici con parametrizzazione	- Metodo più veloce e più adatto a spostamenti tra punti distanti [++] - Meno versatile per l'imposizione di vincoli e task aggiuntivi [--]

Tabella 5.1 : Confronto Metodi Inversione

Si ricordi che si possono aggiungere le considerazioni fatte nel paragrafo 4.5 per rendere anche la CLIK basata su pseudo-inversione meno influenzata dalle singolarità, e si consideri che se si vuole usare un task secondario servirà comunque una pseudo-inversione tale per cui si annullerebbero i vantaggi di tempo computazionale del metodo della trasposta. Quindi non si implementerà il metodo della trasposta (peraltro usato principalmente negli scorsi decenni quando vi era meno potenza di calcolo). Inoltre, se si considera quanto era stato detto nel paragrafo 3, i maggiori problemi saranno quelli di rispettare limiti di angolo e di velocità. Questo si potrebbe affrontare con un Task Space Augmentation, ma esistono nuovi algoritmi che si prefiggono lo stesso obiettivo proponendo una via più efficace per ottenerli (vedi prossimo paragrafo). Quindi anche questo aspetto non si implementerà, mentre si realizzeranno gli algoritmi basati sulla pseudo-inversa e sulla proiezione nel nullo dello Jacobiano, oltre appunto all'algoritmo proposto nel successivo paragrafo.

5.2 Algoritmi Specifici per far Fronte ai Limiti di Giunto e Velocità in Tempo Reale: l'Algoritmo SNS

L'inversione cinematica ottenuta mediante pseudo-inversa ha quindi la caratteristica non tenere esplicitamente in considerazione limiti di giunto, limiti di velocità, limiti di accelerazione, e di fatto l'assunzione è che il task sia inserito in limiti compatibili con quelli del robot. Infatti, a meno di imporre un task aumentato con i problemi che ne derivano, generalmente si convertono i limiti "hard" in limiti "soft" risolvendo la ridondanza con task di ottimizzazione (tipicamente con l'algoritmo di proiezione del gradiente nel nullo). Dato che il task cartesiano ha sempre la priorità maggiore, il soddisfacimento dei limiti di giunto non è però garantito. Questo è scomodo in ambienti dinamici, perché non si può pianificare in anticipo se i limiti saranno raggiunti o meno. Se tali limiti vengono raggiunti si dovranno imporre saturazioni o scalatura di traiettorie, che non sono sempre soddisfacenti dal punto di vista del task Cartesiano in quanto comportano una perdita di accuratezza. Pertanto è opportuno verificare se esistano moti alternativi dei giunti che non valichino i limiti ed eseguano ugualmente bene il task [5].

Questo risultato può essere ottenuto dalla classe di algoritmi con saturazione nel nullo (SNS), presentati dapprima in [24]. In tale caso la saturazione degli $l \leq (n-m)$ giunti che eccedono il limite è compensata per quanto possibile dai rimanenti ancora non saturi.

In [5] si presenta un'evoluzione dell'algoritmo che verrà esaminata e ampliata nei prossimi paragrafi. Essa è orientata a disabilitare un giunto alla volta dal set di quelli in saturazione (il giunto più critico), e di reintrodurre il contributo di tale giunto saturato nel nullo dello Jacobiano. Alle velocità si ottiene pertanto:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q}) \cdot s \cdot \dot{\boldsymbol{x}} + \left(\boldsymbol{I} - \boldsymbol{J}^{\#}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\right)\dot{\boldsymbol{q}}_N$$

dove $s \in [0,1]$ è il coefficiente di scalatura della traiettoria, $\boldsymbol{J}^{\#}$ è la pseudo inversa eventualmente pesata dello Jacobiano, il vettore $\dot{\boldsymbol{q}}_N$ è un vettore che conterrà i giunti saturati e che verrà proiettato nel nullo. Il coefficiente di scalatura s è, se possibile, mantenuto pari a 1 per preservare il task cartesiano, altrimenti è ridotto quanto basta per rendere il task fattibile considerando i limiti. La diversità rispetto al metodo di Omrcen [24] sta nel fatto che selezionando un solo giunto critico alla volta è possibile ricavare nuove pseudo inverse con minori oneri di calcolo rendendo il tutto più adatto all'applicazione real-time. Segue un esempio illustrativo del funzionamento dell'algoritmo con saturazione di velocità nel null-space.

Esempio illustrativo (da [5] **)**

Si consideri un manipolatore planare 4R che segua una traiettoria nel piano, con quindi 2 dof ridondanti $(n = 4, m = 2)$

- Esso abbia i seguenti dati:

Velocità massime: $v1_{max} = v2_{max} = 2; \quad v3_{max} = v4_{max} = 4$

Angoli all'istante considerato: $\boldsymbol{q} = \left[\frac{pi}{2}, \ -\frac{pi}{2}, \ \frac{pi}{2}, \ -\frac{pi}{2}\right]'$

Jacobiano in tale configurazione: $\boldsymbol{J} = [\boldsymbol{J}_1 \ \boldsymbol{J}_2 \ \boldsymbol{J}_3 \ \boldsymbol{J}_4] = \begin{bmatrix} -2 & -1 & -1 & +0 \\ +2 & +2 & +1 & +1 \end{bmatrix}$

Velocità desiderata nello spazio cartesiano: $\dot{\boldsymbol{x}} = [-4, \ -1.5]'$

- La soluzione semplice sarebbe:

$\dot{\boldsymbol{q}}_{PS} = \boldsymbol{J}^{\#}\dot{\boldsymbol{x}} = [2.45, \ -2.14, \ 1.23, \ -3.36]'$;

- Correzione imponendo $\dot{q}_1 = v1_{max} = \dot{q}_2 = v2_{max} = 2$ (Omrcen) :

$\dot{\boldsymbol{x}}' = \dot{\boldsymbol{x}} - \boldsymbol{J}_1 v1_{max} + \boldsymbol{J}_2 v2_{max} = [\boldsymbol{J}_3 \ \boldsymbol{J}_4] \begin{bmatrix} \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} \Rightarrow$

$\dot{\mathbf{q}}' = [v1_{max} \ \ -v2_{max} \ [\boldsymbol{J}_3 \ \boldsymbol{J}_4]^{-1}\dot{\boldsymbol{x}}'] = [2 \ -2 \ \ 2 \ -3.5]$

- Correzione SNS imponendo solo il caso più critico ($\dot{q}_1 = v1_{max}$) :

$$\dot{\boldsymbol{x}}_{SNS} = \dot{\boldsymbol{x}} - \boldsymbol{J}_1 v1_{max} = [\boldsymbol{J}_2\ \boldsymbol{J}_3\ \boldsymbol{J}_4] \begin{bmatrix} \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} \Rightarrow$$

$$\dot{\boldsymbol{q}}_{SNS} = [\mathrm{v1}_{\max} \quad [\boldsymbol{J}_2\ \boldsymbol{J}_3\ \boldsymbol{J}_4]^{\#} \dot{\boldsymbol{x}}_{\mathrm{SNS}}] = [2, -1.83,\ 1.83,\ -3.667]$$

- Se non si riesce a sfruttare la ridondanza per eliminare l'infattibilità allora si procede a scalare il task (si scala di meno con SNS rispetto che col metodo di Omrcen).

Definizione dei limiti utilizzati nell'algoritmo

Si possono definire i limiti sui giunti includendo anche i limiti di escursione oltre che quelli di velocità . Una possibilità consiste nel porre:

$$\dot{Q}_{min,i}(t_k) \leq \dot{q} \leq \dot{Q}_{max,i}(t_k)$$

con

$$\dot{Q}_{min,i}(t_k) = \max\left\{\frac{Q_{min,i} - q_{k,i}}{T};\ -V_{max,i}\right\} \quad i = 1 \dots n$$

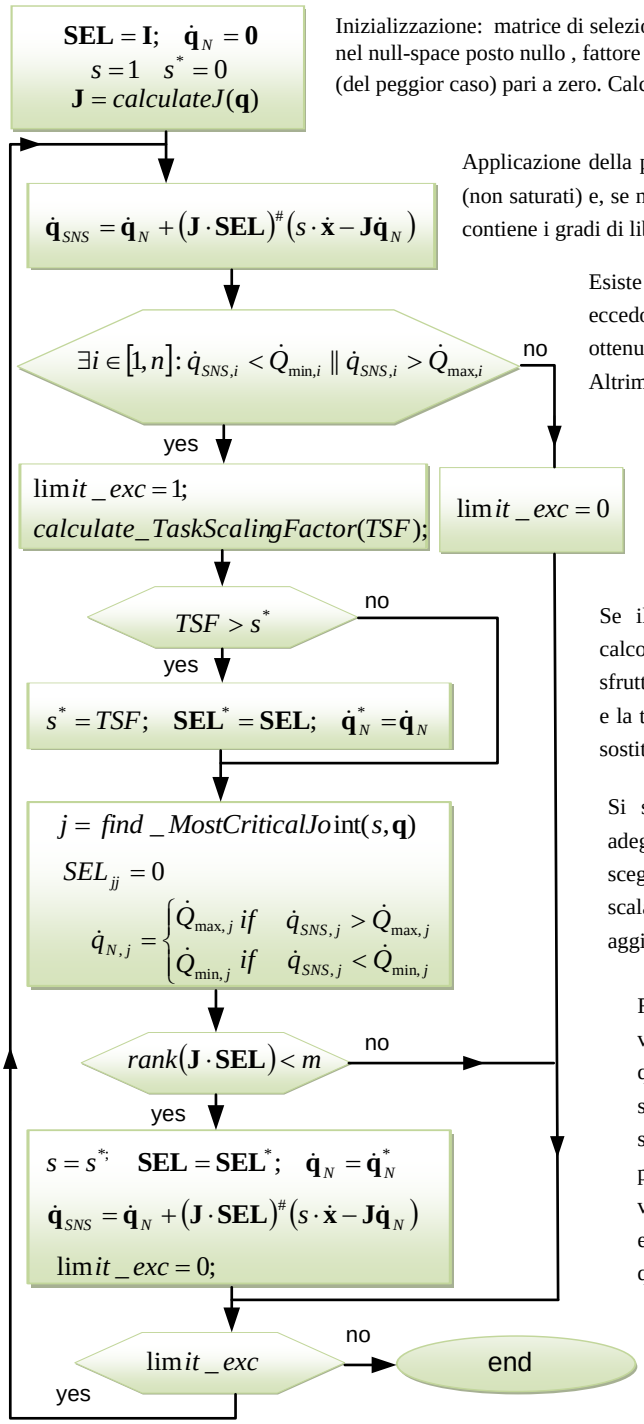
$$\dot{Q}_{max,i}(t_k) = \min\left\{\frac{Q_{max,i} - q_{k,i}}{T};\ V_{max,i}\right\} \quad i = 1 \dots n$$

Dove $Q_{min,i}$ e $Q_{max,i}$ sono i limiti dell'escursione dei giunti e $V_{max,i}$ è il limite alla velocità di giunto (si veda Tabella 3.3 per i corrispondenti dati del LWR).

Algoritmo SNS

Si presenterà ora l'algoritmo SNS, che avrà il vantaggio di utilizzare il nullo dello Jacobiano per ricalcolare le velocità avendo imposto la saturazione al giunto più critico. Inoltre si terranno presenti nello stesso tempo vincoli sui limiti di giunto e scelta di un coefficiente di scalatura del task che sia il minimo possibile. Infine si avrà il vantaggio di un algoritmo molto veloce anche dal punto di vista computazionale per il modo in cui è calcolabile la pseudo-inversione ad ogni passo [5]. Nella pagina successiva è riportato uno schema dell'algoritmo, in cui $\boldsymbol{SEL}$ è una matrice di selezione che rimuoverà dallo Jacobiano le colonne corrispondenti ai

giunti saturati che verranno inseriti in $\dot{\boldsymbol{q}}_N$ che verrà proiettato nel nullo; s è il coefficiente di scalatura, $\dot{\boldsymbol{q}}_{SNS}$ è il vettore velocità ai giunti, $\dot{\boldsymbol{x}}$ è il vettore velocità desiderato, $\dot{\boldsymbol{Q}}_{max}$ e $\dot{\boldsymbol{Q}}_{min}$ sono i limiti per i giunti.



Inizializzazione: matrice di selezione posta come matrice identità; vettore delle velocità nel null-space posto nullo , fattore di scala pari a uno; fattore di scala minimo applicabile (del peggior caso) pari a zero. Calcolo dello Jacobiano analitico.

Applicazione della pseudo inversione sui soli gradi di libertà ancora attivi (non saturati) e, se necessario, scalati. Al risultato è sommato il vettore che contiene i gradi di libertà disattivati e saturati.

Esiste almeno un giunto in cui le velocità così calcolate eccedono quelle limite ? Se non succede il risultato ottenuto con la pseudo-inversione è già accettabile. Altrimenti si procede come segue.

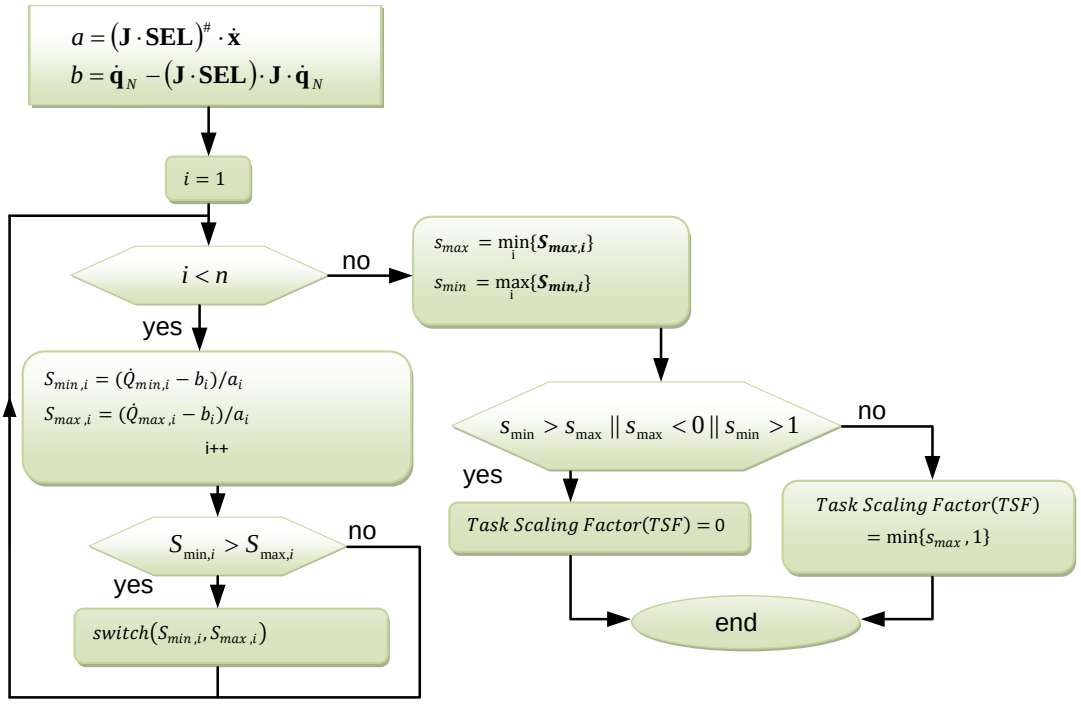
Se i limiti sono ecceduti, prima di saturare si richiama un sub-algoritmo che calcola il fattore di scala che sarebbe da applicare alla situazione corrente (vedi in seguito)

Se il Task Scaling Factor risulta maggiore di quello calcolato al passo precedente significa che grazie allo sfruttamento della ridondanza si è migliorata la situazione e la traiettoria sarà più vicina a quella imposta. In tal caso sostituisco s* e preparo le matrici e vettori di selezione

Si seleziona quindi il giunto più critico con un adeguato criterio (che può essere ad esempio quello di scegliere il giunto che richiederebbe il coefficiente di scala minore per poter rientrare tra i limiti). Si aggiornano la matrice di selezione e il vettore $\dot{q}_N$

Fino a quando si può continuare a imporre un valore di giunto pari al relativo limite ? Fino a quando il rango di è >= alla dimensione dello spazio in cui si impone il task (tipicamente nello spazio tridimensionale in cui si definiscono posizione e angoli) . Quando ciò non fosse verificato non si potrà più sfruttare la saturazione e si dovrà applicare la scalatura della traiettoria e quindi uscire

Per quanto riguarda il calcolo dello Jacobiano analitico si rimanda al prossimo capitolo. Per quanto riguarda la chiamata al sub-algoritmo per calcolare il task scaling factor da applicare alla traiettoria nel caso essa comporti superamento dei limiti non compensabili, la sua struttura è la seguente:



Per quanto riguarda la scelta del giunto più critico, un criterio è quello di scegliere il giunto la cui velocità necessita il fattore di scala s_{max} più prossimo a zero per rimanere entro i limiti.

Il principio dell'algoritmo è quindi quello di controllare se il task originario ($s = 1$) possa essere eseguito con la velocità calcolata tramite la pseudo inversa all'interno dei limiti imposti. Se ciò non avviene si richiama l'algoritmo 2 per trovare il minimo fattore positivo di scala solo tra i giunti ancora attivi. Se tale fattore di scala è maggiore di quello attuale si memorizza $\mathbf{SEL}^* = \mathbf{SEL}$, $\dot{\mathbf{q}}_N^* = \dot{\mathbf{q}}_N$, $s^* = s$. Questi valori assicurano la velocità più vicina a quella desiderata che possa al contempo essere erogata. Poi si passa a saturare il giunto più critico scelto come descritto sopra settando $SEL_{jj} = 0$. Se il task non può essere eseguito con questi nuovi parametri, se

ora il rango di J · SEL è strettamente minore di m (con m che è la dimensione del task da eseguire), l'algoritmo si stoppa con i migliori parametri fin qui trovati ($\mathbf{SEL} = \mathbf{SEL}^*$, $\dot{\mathbf{q}}_N = \dot{\mathbf{q}}_N^*$, $s = s^*$) dando in output il risultato di $\dot{\mathbf{q}}_{SNS} = \dot{\mathbf{q}}_N + (\mathbf{J} \cdot \mathbf{SEL})^{\#} \cdot (s \cdot \dot{\mathbf{x}} - \mathbf{J} \cdot \dot{\mathbf{q}}_N)$. Altrimenti, la velocità di giunto è ricalcolata con i parametri correnti ripetendo l'iterazione.

5.3 Modifica all'Algoritmo SNS con l'Aggiunta di Altri Task

Aggiunta di task di vincolo

Per imporre dei vincoli cartesiani che vengano sempre rispettati è possibile incorporarli nei vincoli già definiti $\dot{\boldsymbol{Q}}_{max,min}$. Ad esempio, considerando il problema di *obstacle avoidance*, sia $\mathbf{C} = (x_c \quad y_c \quad z_c)^T$ un punto di controllo del robot il cui spostamento cartesiano è limitato dall'ostacolo. Si può considerare il semplice limite $\mathbf{C} > \mathbf{C}_L$, dove $\mathbf{C}_L$ è un limite cartesiano fisso. Quando $\mathbf{d} = \mathbf{C}(q) - \mathbf{C}_L \geq 0$, si associa a $\|\mathbf{d}\| \geq 0$ una funzione adatta che tenda rapidamente a zero se tale distanza fosse più grande di un range γ e che invece stia nell'intorno di uno se tale distanza fosse circa nulla. Una funzione atta allo scopo è la "Funzione Logistica" sigmoidale:

$$f(\mathbf{d}) = \frac{1}{1+e^{\left(\frac{\|\mathbf{d}\|_2}{\gamma}-1\right)\alpha}}.$$

Una volta fatto ciò è sufficiente definire il vettore $\mathbf{D} = f(\mathbf{d}) \cdot \frac{\mathbf{d}}{\|\mathbf{d}\|}$ interpretabile come una forza repulsiva dal vincolo nello spazio cartesiano, e passarlo allo spazio dei giunti come: $s = \mathbf{J}^T(\mathbf{q})D$ le cui componenti s_i rappresentano un "grado di influenza" del vincolo cartesiano sul giunto i-esimo. Esso si può utilizzare per modellare i limiti in velocità come segue:

$$\text{if } s_i \geq 0 \quad \dot{Q}_{max,i} = \dot{Q}_{max,i}\left(1 - f(\mathbf{d})\right)$$
$$\text{else} \qquad \dot{Q}_{min,i} = \dot{Q}_{min,i}\left(1 - f(\mathbf{d})\right)$$

Ciò fa in modo che i giunti in contrasto con il vincolo cartesiano rallentino (anche se sempre si potrà contare sulla reiterazione dell'algoritmo finché si avranno gradi di libertà ridondanti rispetto al task). Multipli ostacoli si potranno contemplare considerando, per ogni i-esimo giunto, il fattore minimo $1 - f(\mathbf{d}_i)$ ottenuto da tutti i vincoli applicati in formulazione di $\dot{Q}_{max,i}$ e $\dot{Q}_{min,i}$.

Modifica dell'algoritmo per consentire l'esecuzione di task secondari

L'algoritmo così presentato non ha esplicitamente la possibilità di specificare task secondari. L'obiettivo è quello di aggiungere un task secondario (espresso come proiezione del gradiente di una funzione obiettivo). Per farlo si parte dal framework per l'aggiunta di task a priorità decrescente presentato in 4.4 [19] per cui

$$\dot{\mathbf{q}}^i = \dot{\mathbf{q}}^{i-1} + \tilde{\mathbf{J}}^{i\dagger}\left(\dot{\mathbf{x}}^i - \mathbf{J}^i\dot{\mathbf{q}}^{i-1}\right)$$

Se $i = 2$ è facile dimostrare che, sempre riferendosi a quanto visto in 4.4:

$$\dot{\mathbf{q}}^2 = \dot{\mathbf{q}}^1 + \left(\mathbf{J}^2\left(\mathbf{I} - \mathbf{J}^{1\#}_{augm}\mathbf{J}^1_{augm}\right)\right)^{\#}\left(\dot{\mathbf{x}}^2 - \mathbf{J}^2\dot{\mathbf{q}}^1\right)$$

Se poi il task aggiunto impone un comportamento direttamente nel *configuration space*, con $\dot{\mathbf{x}}^2 = \dot{\mathbf{q}}_{cs}$ e $\mathbf{J}^2 = \mathbf{I}$, tutto si semplifica e si trova, come ci si aspetta:

$$\dot{\mathbf{q}}^2 = \dot{\mathbf{q}}^1 + \left(\mathbf{I} - \mathbf{J}^{1\#}_{augm}\mathbf{J}^1_{augm}\right)\dot{\mathbf{q}}_{cs} = \dot{\mathbf{q}}^1 + \left(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J}\right)\dot{\mathbf{q}}_{cs}$$

Per aggiungere il task secondario basterà aggiungere al $\dot{\mathbf{q}}_{SNS}$ trovato in 5.2 la quantità $\left(\mathbf{I} - \left(\mathbf{J}\cdot\mathbf{SEL}_{CS}\right)^{\#}\mathbf{J}\cdot\dot{\mathbf{q}}_{CS}\right.$ ove $\mathbf{SEL}_{CS}$ esclude gli angoli già saturati dal provare

a soddisfare il task secondario. Inoltre si dovrà includere anche la scalatura per questo task, così che l'algoritmo per aggiungere il task secondario sarà:

$$\boldsymbol{SEL}_a = \boldsymbol{I}$$
$$for\ i: 1 \rightarrow n$$
$$if\ \left(\dot{\mathrm{q}}_{\mathrm{SNS,i}} == \dot{Q}_{max,i}\right) || \left(\dot{\mathrm{q}}_{\mathrm{SNS,i}} == \dot{Q}_{min,i}\right)$$
$$SEL_{a,i,i} = 0\ ;$$
$$endif;$$
$$endfor$$

$$\boldsymbol{P}_a = (\mathbf{I} - (\mathbf{J} \cdot \mathbf{SEL}_{\mathrm{a}})^{\#}\mathbf{J}$$
$$\mathbf{a} = \mathbf{P}_{\mathrm{a}} \cdot \dot{\mathbf{q}}_{\mathrm{a}}$$
$$\mathbf{b} = \dot{\mathbf{q}}_{\mathrm{SNS}}$$
$$\mathrm{TSF} = \mathrm{Calculate\_TaskScalingFactor}(\mathbf{a}, \mathbf{b})$$

$$\dot{\mathbf{q}}^2_{\mathrm{SNS}} = \dot{\mathbf{q}}_{\mathrm{SNS}} + \mathrm{TSF} \cdot \mathbf{P}_{\mathrm{a}} \cdot \dot{\mathbf{q}}_{\mathrm{a}}$$

5.4 Criteri di Gestione della Ridondanza Testati

5.4.1 Per Mantenersi Lontano dai Limiti Imposti ai Giunti

Per i metodi implementati (sia in Matlab che C++) si sono aggiunti task secondari che potrebbero essere utili per la coordinazione del sistema multi robot in real-time con imprevedibilità di traiettorie e velocità. Un primo esempio di task secondario testato è stato quello "semplice" del cercare di mantenere i giunti il più possibile a centro corsa per evitare che arrivino ai limiti di giunto e quindi provochino saturazione (saturazione pura per la CLIK, proiezione della velocità saturata per l'SNS). La formulazione del vettore velocità proiettato nel nullo sarà:

$$\dot{\boldsymbol{q}}_a = -k_a \frac{\partial w}{\partial \boldsymbol{q}} = -k_a \frac{\partial}{\partial \boldsymbol{q}} \left(\frac{1}{n} \sum_{i=1}^{n} \left(\frac{\boldsymbol{q}_i - \overline{\boldsymbol{q}}}{\boldsymbol{q}_{iMAX} - \boldsymbol{q}_{imin}} \right)^2 \right)$$

Nel caso dell'LWR: $\overline{\boldsymbol{q}} = 0, \quad \boldsymbol{q}_{imin} = -\boldsymbol{q}_{iMAX}, \quad n = 7,$ quindi:

$$\dot{\boldsymbol{q}}_a = -k_a \frac{\partial}{\partial \boldsymbol{q}} \left(\frac{1}{7} \sum_{i=1}^{n} \left(\frac{\boldsymbol{q}_i}{2\boldsymbol{q}_{iMAX}} \right)^2 \right) = \frac{-k_a}{7} \sum_{i=1}^{n} \frac{\boldsymbol{q}_i}{4\boldsymbol{q}_{iMAX}^2}$$

5.4.2 Per Mantenere la Distanza da Ostacoli

Di particolare interesse anche la possibilità di evitare ostacoli. Il task secondario tramite gradiente positivo potrà massimizzare la distanza tra un ostacolo (qui assunto puntiforme per semplicità) e un punto di controllo del robot (qui posto, sempre per semplicità, coincidente con il giunto di gomito, ma estendibile anche a ogni altro punto del robot così da trovare quello a minor distanza dall'ostacolo). Si ha:

$$\dot{\boldsymbol{q}}_a = +k_a \frac{\partial w}{\partial \boldsymbol{q}} = +k_a \frac{\partial}{\partial \boldsymbol{q}} \min \underbrace{(\|\mathbf{C}(\mathrm{q}) - \mathbf{C}_\mathrm{L}\|)}_{P_{robot} - xyz_{ostacolo}}$$

Nel caso semplificato in cui si considera solo il gomito come punto di controllo del robot e singolo ostacolo fermo puntiforme:

$$\dot{\boldsymbol{q}}_a = +k_a \frac{\partial}{\partial \boldsymbol{q}} (fkine(q_1, q_2, q_3, q_4) - \mathbf{C}_\mathrm{L})$$

Che è calcolabile ad esempio tramite calcolo simbolico su Matlab. Altri approcci più completi alla presenza di ostacoli nell'area di lavoro di un robot si possono trovare ad esempio in [25].

5.4.3 Per Massimizzare la Controllabilità in Forza e Ottimizzare la Postura Relativa dei Manipolatori

Richiamo del concetto di ellissoidi di manipolabilità

Per rappresentare l'attitudine di un manipolatore ad eseguire un compito nella configurazione corrente si può usare il concetto di ellissoidi di manipolabilità cinetostatica. Se si è interessati a rappresentare la capacità da parte del manipolatore di cambiare arbitrariamente posizione e orientamento dell'organo terminale si può usare l'ellissoide di manipolabilità in velocità. Esso è definito a partire dall'insieme delle velocità ai giunti a norma costante $\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} = 1$: questa equazione descrive i punti sulla superficie di una sfera nello spazio delle velocità ai giunti. Se si vogliono caratterizzare le velocità nello spazio operativo generabili con il dato insieme di velocità ai giunti si ha (ricordando il legame $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}}$) :

$$\dot{\boldsymbol{x}}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)^{-1}\dot{\boldsymbol{x}} = 1$$

Che è l'equazione dei punti sulla superficie di un ellissoide nello spazio delle velocità dell'organo terminale. In virtù della dualità cineto-statica [13] è possibile descrivere la manipolabilità di una struttura anche in relazione alle forze. Infatti se nello spazio delle coppie ai giunti si ha $\boldsymbol{\tau}^T\boldsymbol{\tau} = 1$ passando nello spazio delle forze all'organo terminale si ha:

$$\boldsymbol{\gamma}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)\boldsymbol{\gamma} = 1$$

Che è l'ellissoide di manipolabilità in forza e caratterizza le forze all'organo terminale che sono generabili con il dato insieme di coppie ai giunti, alla postura assegnata del manipolatore. Quindi ad esempio lungo l'asse maggiore a pari coppia ai giunti si avrà alta forza generabile all'end effector, e viceversa l'ungo l'asse minore.

Una interpretazione utile delle considerazioni effettuate può ricavarsi guardando il manipolatore come un *trasformatore meccanico* di velocità e forze dallo spazio dei giunti allo spazio operativo. Il rapporto di trasmissione lungo una data direzione è determinato dall'intersezione del vettore lungo tale direzione con la superficie dell'ellissoide. Una volta assegnato il versore $\boldsymbol{u}$ in una certa direzione, è possibile calcolare il rapporto di trasformazione per l'ellissoide di manipolabilità in forza come [13] [26]:

$$\alpha = [\boldsymbol{u}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)\boldsymbol{u}]^{\frac{1}{2}}$$

Questo indice può essere usato per analizzare la compatibilità della struttura ad eseguire un compito assegnato lungo una direzione in un determinato punto della traiettoria in equilibrio quasi statico[1]. A tal scopo, è utile distinguere tra compiti di attuazione e di controllo di forza. In termini del relativo ellissoide, un compito di attuazione di una forza richiede preferibilmente un elevato rapporto di trasformazione lungo la direzione del compito, poiché per un dato insieme di forze ai giunti può generare una forza elevata all'end effector. Al contrario per compiti di controllo è importante che il rapporto di trasformazione sia il più piccolo possibile per avere massima sensibilità ad eventuali errori lungo la direzione assegnata (N.B. la direzione migliore per avere massima controllabilità non coincide con quella allineata all'asse minore dell'ellissoide, dato che la minima distanza tra centro e superficie dell'ellissoide lungo una direzione può essere più piccola dell'asse minore dell'ellissoide, come mostrato nella seguente Figura).

[1] *N.B. : Qui si considerano gli ellissoidi di manipolabilità cineto-statica valutati in ogni configurazione istantanea e sfruttati per definire un indice di gestione della ridondanza. Se si volessero invece includere in un loop di controllo, bisognerebbe introdurre la manipolabilità dinamica per il calcolo della coppia (famiglia degli algoritmi di controllo a coppia pre-calcolata)* [33].

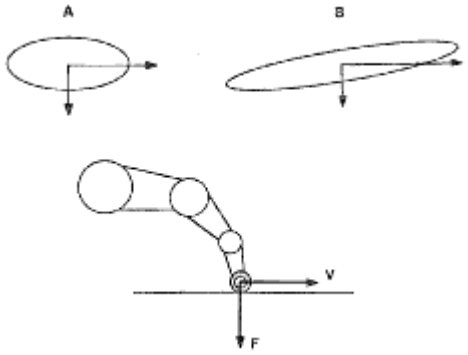


Figura 5.1 : Comparazione di ellissoidi di forza. B risulterà più compatibile per un controllo di forza lungo la direzione verticale

Un esempio si ha con il braccio umano durante la scrittura: bisogna controllare finemente una forza verticale per dare la giusta pressione alla penna. Si può vedere che quindi il braccio umano tende a porsi nella configurazione tale per cui il coefficiente di trasmissione di forza è minimo.

Applicazione al caso considerato

Uno dei criteri per gestire la ridondanza di manipolatori che, come visto nel Capitolo 1, sono particolarmente adatti ad interagire con l'ambiente mediante controllo di forza o di impedenza, può essere proprio quello di minimizzare/massimizzare i rapporti di trasmissione appena descritti lungo alcune direzioni desiderate. Si pensi ad esempio all'inserzione di utensili chirurgici: è chiaro che avere buona controllabilità di forza nella direzione di inserzione può essere molto utile.

Inoltre, anche passando al sistema multi robot, questo aspetto può essere molto interessante: si pensi ai due bracci che interagiscono ad esempio assemblando un oggetto, o secondo un modello tipo quello di Figura 2.13. In quei casi è ancora utile gestire la controllabilità in forza garantita da una determinata postura. Quindi, in base alle considerazioni fatte sopra, se si vuole minimizzare il rapporto di trasmissione di forza lungo una direzione e ottenerne quindi buona controllabilità è possibile definire una funzione in forma quadratica pari a :

$$C^2 = \left(\frac{1}{\alpha}\right)^2 = [\boldsymbol{u}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)\boldsymbol{u}]$$

Se si vorrà gestire la ridondanza affinché il robot si posizioni nella configurazione locale (che in generale è mutevole istante per istante) che dia miglior controllabilità, questa funzione sarà da massimizzare tramite proiezione del gradiente nel nullo dello Jacobiano nell'algoritmo di inversione cinematica presente nel programma di cooperazione tra robot:

$$\dot{\boldsymbol{q}}_a(\boldsymbol{q},\boldsymbol{u}) = +k_a\frac{\partial}{\partial \boldsymbol{q}}\,(\boldsymbol{u}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)\boldsymbol{u})^{-\frac{1}{2}}$$

Nel calcolo di tale funzione si considera che generalmente **u** è specificata come

$$\boldsymbol{u} = \begin{bmatrix}\boldsymbol{u}_{3\times1}\\ \boldsymbol{0}_{3\times1}\end{bmatrix} \Rightarrow \boldsymbol{u}^T(\boldsymbol{J}\cdot\boldsymbol{J}^T)\boldsymbol{u} = \boldsymbol{u}_{3\times1}^T \quad \underbrace{(\boldsymbol{J}\cdot\boldsymbol{J}^T)}_{\substack{prima\\ sottomatrice\\ 3\times3}} \quad \boldsymbol{u}_{3\times1}$$

Il che semplificherà molto i calcoli eseguiti con il simbolico di Matlab.

Capitolo 6

Inversione Cinematica: Modello Matlab e Sistema Reale

6.1 Calcolo dello Jacobiano

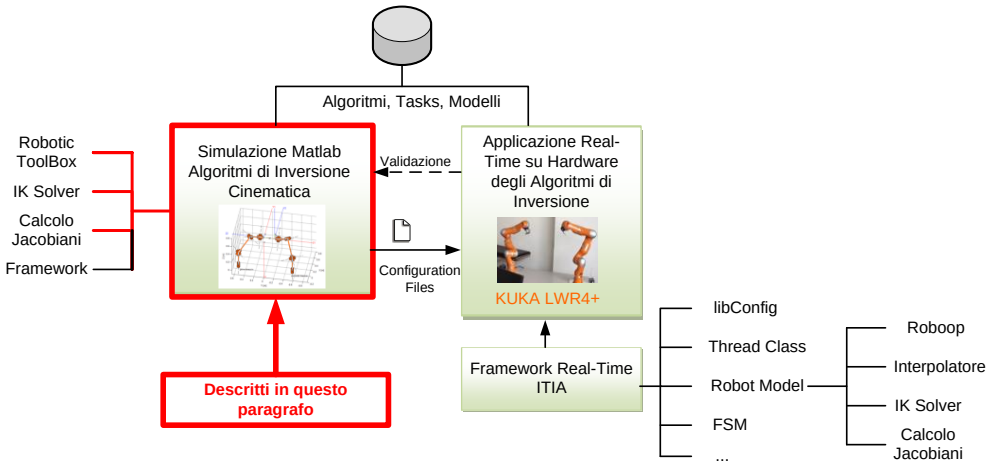


Figura 6.1 : Collocazione del Paragrafo

In ogni algoritmo implementato in Matlab e C++ si passerà dalla (pseudo)inversione dello Jacobiano analitico. Si elencano ora i passi con cui è stata creata una funzione per ottenerlo:

- Generazione della matrice (simbolica) di trasformazione tra la terna base del robot e l'end effector a partire dai parametri di Denavit-Hartenberg $\left(\boldsymbol{T}_{tot} = \prod_{i=1}^{ndof} \boldsymbol{T}_{i-1}^{i}(q_i)\right)$

- Estrazione del vettore posizione da tale matrice $\left(\mathbf{pos}(1:3) = \mathbf{T}(1:3,4)\right)$

- Estrazione del vettore degli angoli da tale matrice secondo una convenzione angolare. Si è visto in 3.1.2 che per poter effettuare confronti con le quantità fornite dal LWR conviene utilizzare gli angoli di Cardano. Quindi si utilizza:

$$r = atan2\left(-\boldsymbol{T}(2,3), \boldsymbol{T}(3,3)\right)$$
$$p = \text{asin}\left(\boldsymbol{T}(1,3)\right)$$
$$y = atan2(-\boldsymbol{T}(1,2), \boldsymbol{T}(1,1))$$

  Sarebbe stato comodo se le funzioni atan2 fossero già utilizzabili col simbolico di Matlab. Poiché ciò non è possibile, si sono riscritte sfruttando la $atan2(y,x) = 2\,\text{atan}\left(\frac{y}{\sqrt{x^2+y^2}+x}\right)$, comportandosi come descritto in [27] anche per i valori che azzererebbero il denominatore. Si avrà così che gli unici valori per cui la formula sarà indefinita saranno $x = 0,\ y = 0$, che genereranno singolarità di rappresentazione.

- Costruzione del vettore $\boldsymbol{x} = [\boldsymbol{pos};\ r; \text{p; y}]$ e calcolo dello Jacobiano $\boldsymbol{J} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}}$

- Confronto con lo Jacobiano Geometrico per verificare la correttezza del risultato. Lo Jacobiano Geometrico è ricavabile ad esempio con il comando "*Jacobian*" della *Robotic Toolbox* o robot.Jacobian() della libreria *Roboop.* Si ha:

$$\boldsymbol{J}_{verifica} = \boldsymbol{T}_{af}/\boldsymbol{JG} \ \ con \ \ \boldsymbol{T}_{af} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{w} \end{bmatrix}$$

$$con \quad \boldsymbol{w} = \begin{bmatrix} 1 & 0 & \sin(p) \\ 0 & \cos(r) & -\sin(r)\cos(p) \\ 0 & \sin(r) & \cos(r)\cos(p) \end{bmatrix}$$

- Esportazione delle funzioni così ottenute in normali funzioni Matlab per il modello e in funzioni utilizzabili in C++ mediante il comando *ccode*.

- Considerando quanto scritto in 3.1.2, si dovrà calcolare lo Jacobiano aggiungendo agli angoli passati in argomento alle funzioni gli offset della FRI rispetto alla convenzione di Denavit-Hartenberg.

- Confronto con gli Jacobiani misurabili tramite FRI. Per ottenere ciò, sempre considerando quanto scritto in 3.1.2, si dovrà passare lo Jacobiano misurato da terna utensile a terna base: $\mathbf{J}_{base} = \begin{bmatrix} \boldsymbol{R}_{tool}^{base} & 0 \\ 0 & \mathbf{R}_{\text{tool}}^{\text{base}} \end{bmatrix} \mathbf{J}_{tool}$ con $\mathbf{R}_{tool}^{base} = \mathbf{R}^{-1}$ [13]

6.2 Inserimento degli Algoritmi nel Modello Matlab e Risultati Simulati

Si integreranno gli algoritmi presentati nei Capitoli 4 e 5 nel programma Matlab descritto al Capitolo 3. Ciò permetterà di calcolare gli angoli per arrivare alle pose $\boldsymbol{T}_{R,i}^{r,i}$ desiderate. Per ogni algoritmo saranno possibili varie opzioni di utilizzo e considerazioni, che verranno presentate con esempi.

Come primo esempio si considera di avere un carrier in moto a formare una traiettoria pentagonale, con gli organi terminali dei robot che lo inseguono mantenendosi affacciati l'uno con l'altro, ovvero con i rispettivi assi *z* locali uscenti dalla flangia allineati antiversi. I metodi di inversione cinematica qui analizzati saranno quello della pseudo-inversione semplice ($\boldsymbol{J}^{\#}$ nelle didascalie) e quello della pseudo-inversione inserita in una CLIK. Le seguenti figure riassumono gli errori di tracking dati dai due metodi in questione.

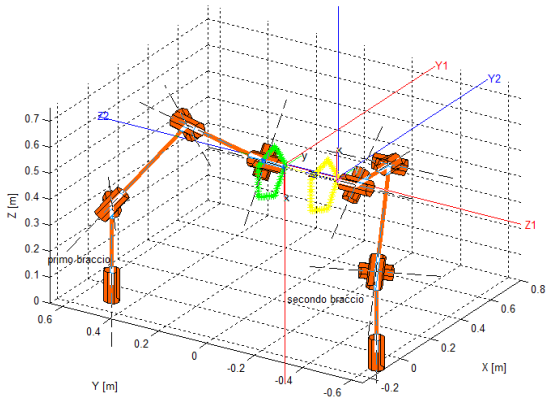


Figura 6.2 : Traiettoria pentagonale seguita con algoritmi " $J^{\#}$ " e "CLIK"

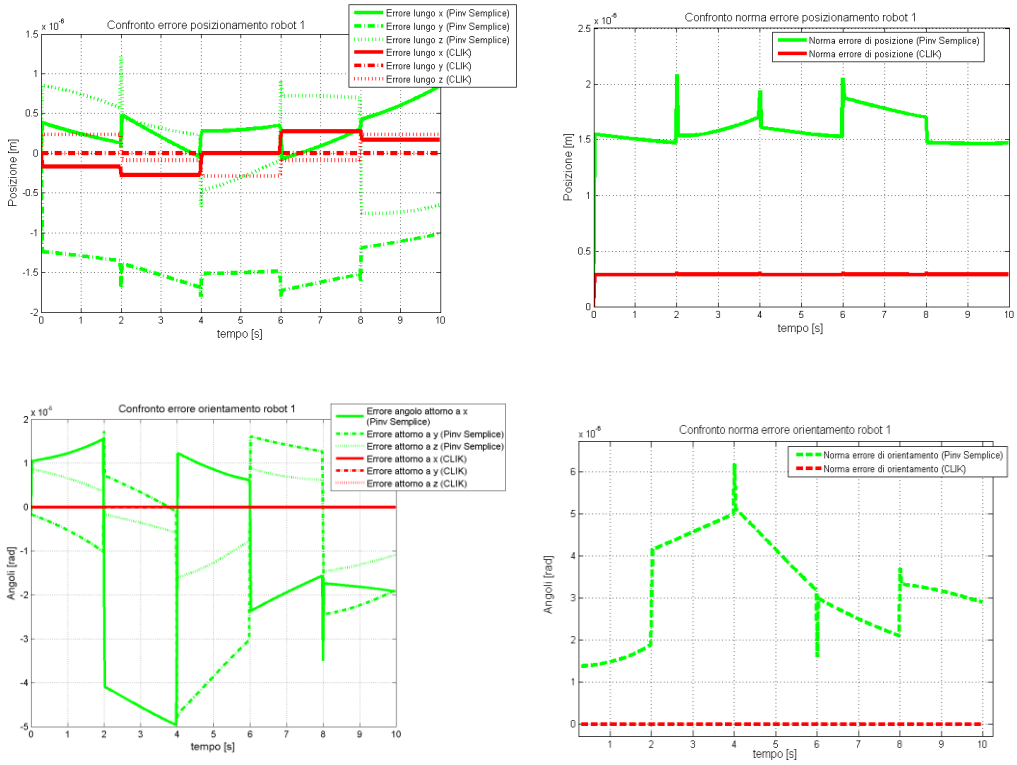


Figura 6.3 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi " $J^{\#}$ " e "CLIK"

Andamenti del tutto analoghi per gli errori del secondo robot. Considerando poi gli angoli e le velocità angolari ottenute si hanno i seguenti risultati (N.B. limite di velocità circa il 50% di quello massimo nominale di 3.1.2)

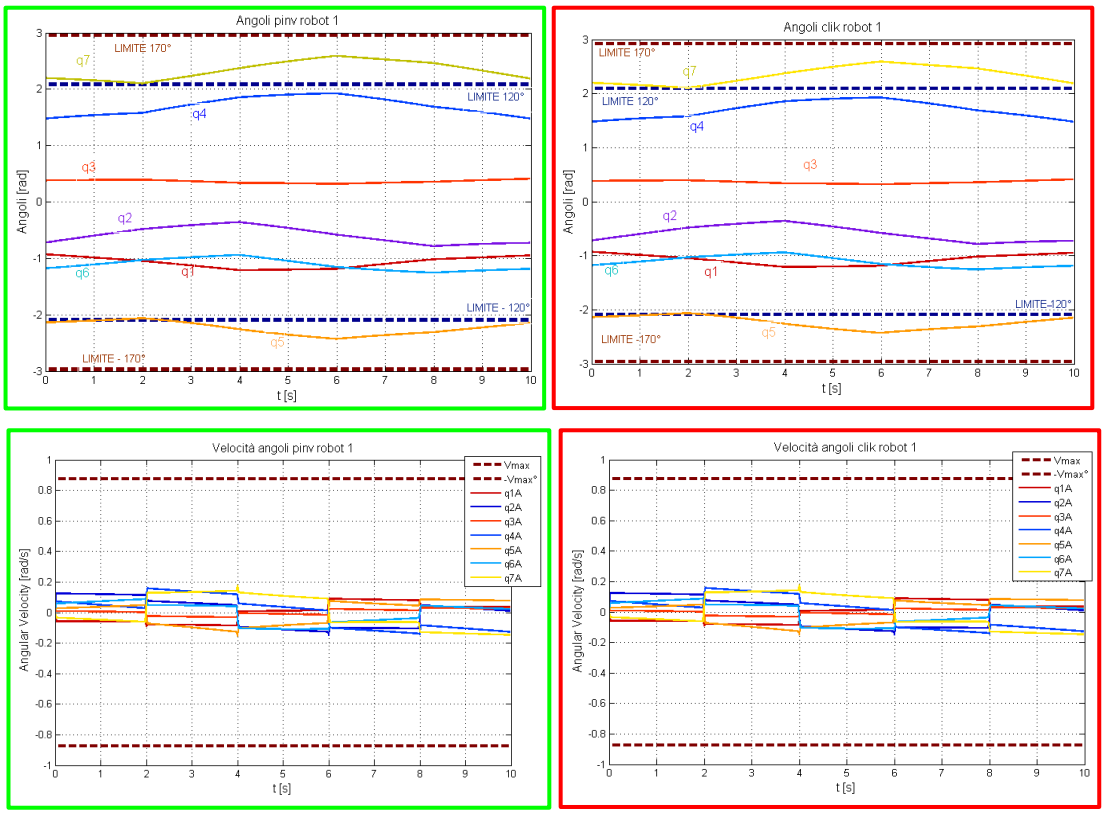


Figura 6.4 : Andamenti degli angoli e delle velocità angolari per il robot 1, con algoritmi " $J^{\#}$ " (cornice verde, a sinistra) e "CLIK" (cornice rossa, a destra)

Si vede che gli errori sono dell'ordine del millesimo di millimetro per entrambi i metodi, e che quello della CLIK è inferiore, come prevedibile in base alle considerazioni fatte in precedenza (vedi Cap. 4). Tale divario tra gli errori dei due metodi si riduce tanto più gli istanti di tempo in cui si opera l'inversione sono ravvicinati. Infatti se non vi fosse una discretizzazione temporale, la soluzione della pseudo-inversione sarebbe esatta e non servirebbe l'inserimento in un anello di retroazione per avere risultati migliori.

Nell'esempio successivo si introducono una rotazione nella parte finale della traiettoria (da $t = 6\,s$ a $t = 8\,s$) e un task secondario. In particolare si chiede al primo robot di allontanare il gomito da un ostacolo che è in sua prossimità. Il primo robot quindi cercherà sempre di allontanare il gomito dall'ostacolo, ove consentito dal task primario. Questa azione di allontanamento continuerebbe a qualsiasi

distanza, per cui è possibile anche disattivare tale task secondario qualora si giudichi che la distanza ottenuta sia sufficiente.

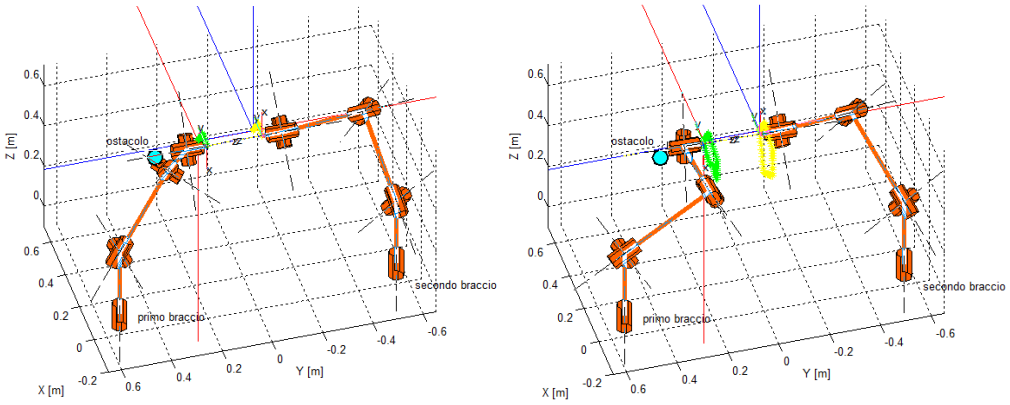


Figura 6.5 : Traiettoria pentagonale seguita con algoritmi " $J^{\#}$ " , "CLIK" e "CLIK con task secondario di allontanamento da ostacolo". Configurazione di partenza (sinistra) e di arrivo (destra)

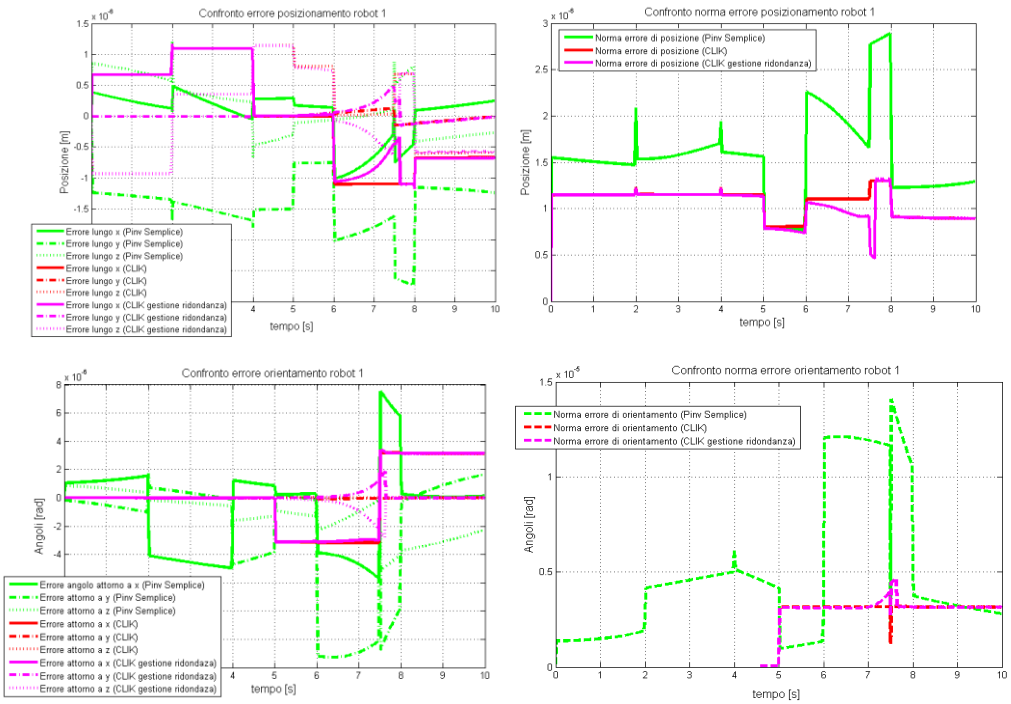


Figura 6.6 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi " $J^{\#}$ " , "CLIK" e "CLIK con task secondario di allontanamento da ostacolo"

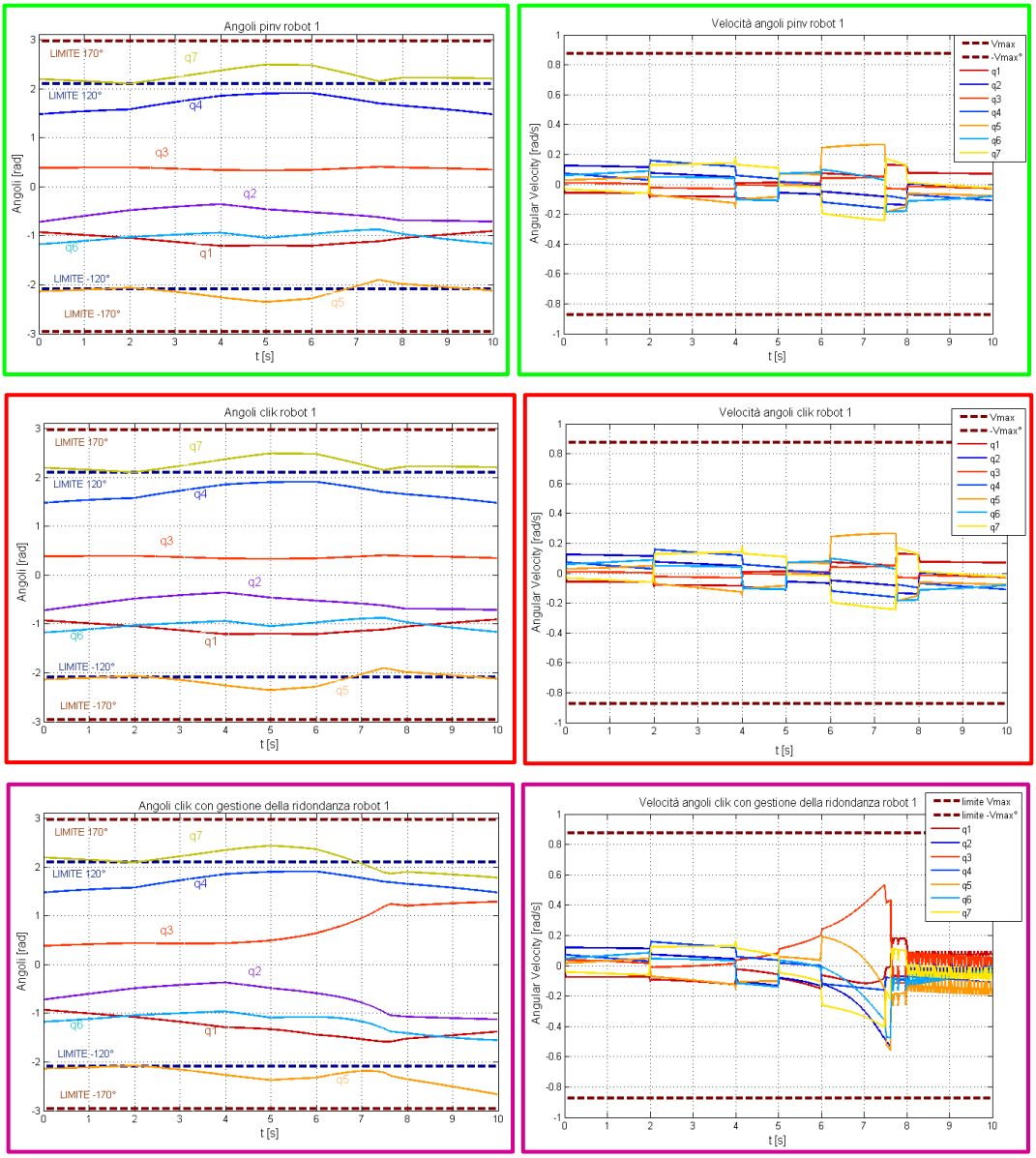


Figura 6.7 : Andamenti degli angoli e delle velocità angolari per il robot 1, con algoritmi " $J^{\#}$ " (cornice verde) , "CLIK" (cornice rossa) e "CLIK con task secondario per allontanamento da ostacolo" (cornice viola)

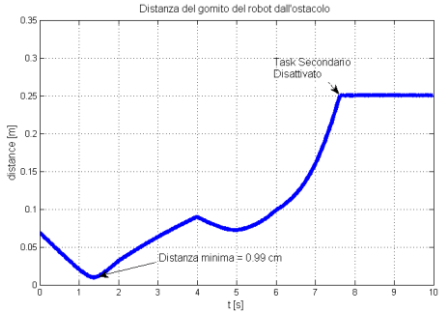


Figura 6.8 : Distanza ostacolo - gomito del manipolatore 1

Si è visto che il task secondario è stato svolto a costo praticamente nullo: essendo eseguito solo se non in contrasto con il task principale, il moto di allontanamento non peggiora l'errore, anche se si cambia in modo improvviso l'orientamento come accade in corrispondenza di $t = 6$. Questo è tuttavia vero solo se si sceglie un guadagno limitato per il task secondario. Nel caso si scelga k_a troppo grande, appena il task principale permetterà di attivare quello secondario, gli angoli si muoveranno molto velocemente ad assecondare il brusco gradiente della funzione obiettivo, provocando un cambiamento di configurazione repentino con conseguente peggioramento dell'approssimazione dello Jacobiano. Questo si tradurrà in un peggioramento dell'errore, come si può vedere nei seguenti grafici in cui si compie lo stesso moto di prima ma con un guadagno k_a maggiore.

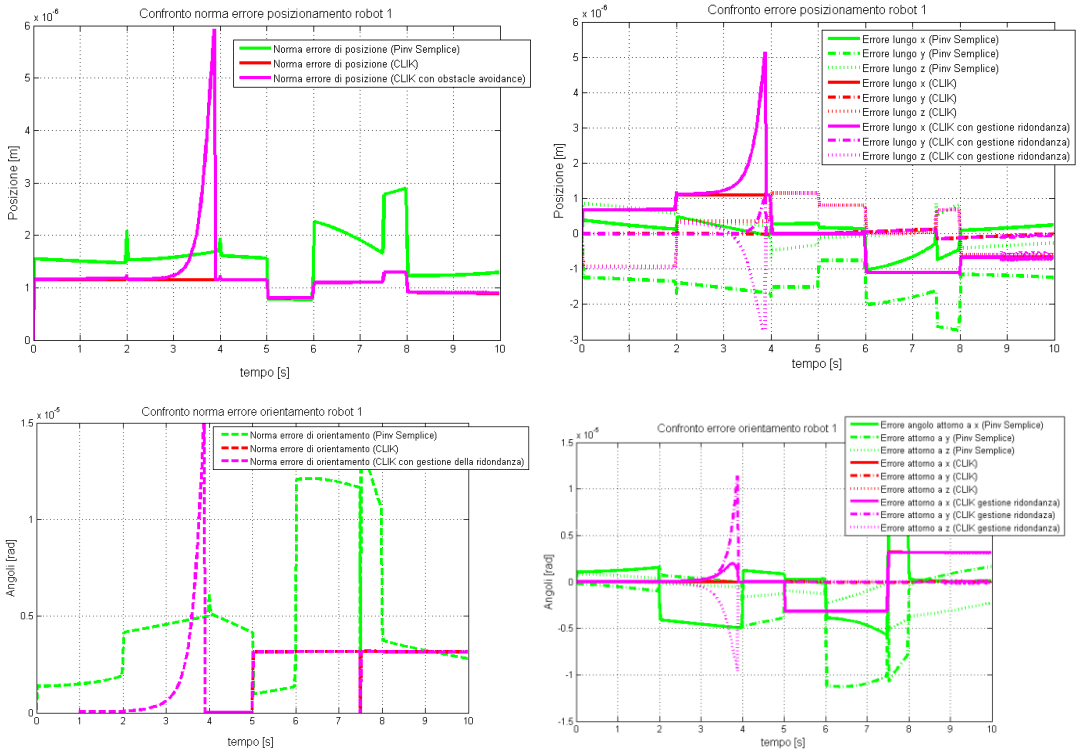


Figura 6.9 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi "$J^{\#}$ ", "CLIK" e "CLIK con task secondario per allontanamento da ostacolo". Caso con k_a alto.

Un problema dell'utilizzo dei task secondari è quindi quello di definire un guadagno opportuno per ogni task. Inoltre si dovrà ricordare che non si avrà mai la certezza che

il task secondario venga eseguito, dato che appunto ha priorità inferiore a quella del task principale. Ad esempio nella seguente Figura si vede che l'allontanamento dall'ostacolo è stato più rapido, ma nel tratto iniziale si è comunque subito un avvicinamento dovuto al task principale.

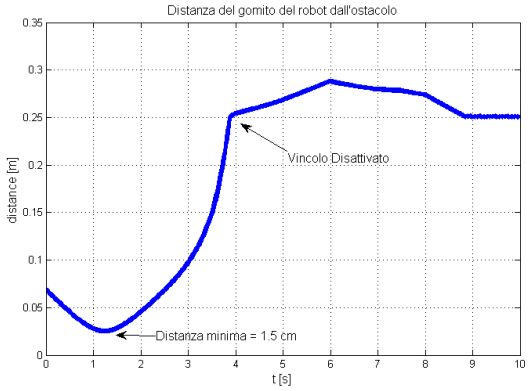


Figura 6.10 : Distanza ostacolo - gomito del manipolatore 1

Si passa poi ad analizzare il task secondario che punta a ottimizzare la configurazione di un robot in base alla posizione dell'altro (Paragrafo 5.4.3). In questo caso, mentre si seguirà la traiettoria, il secondo robot cercherà di configurarsi così da avere massima controllabilità lungo la direzione che congiunge i due *end effectors*. Questo, come già accennato, garantirà che tra tutte le possibili configurazioni si assuma la più adatta per effettuare successive operazioni in cui i due manipolatori entrino in contatto o agiscano in controllo di forza lungo tale direzione congiungente. Nella seguente Figura si riportano quattro fotogrammi successivi del movimento dei robot, in cui è facile vedere che il secondo manipolatore sta assumendo una posizione simile a quella che assumerebbe il braccio umano per "scrivere" su un piano perpendicolare alla congiungente, ossia con l'avambraccio circa allineato con la superficie del piano xz.

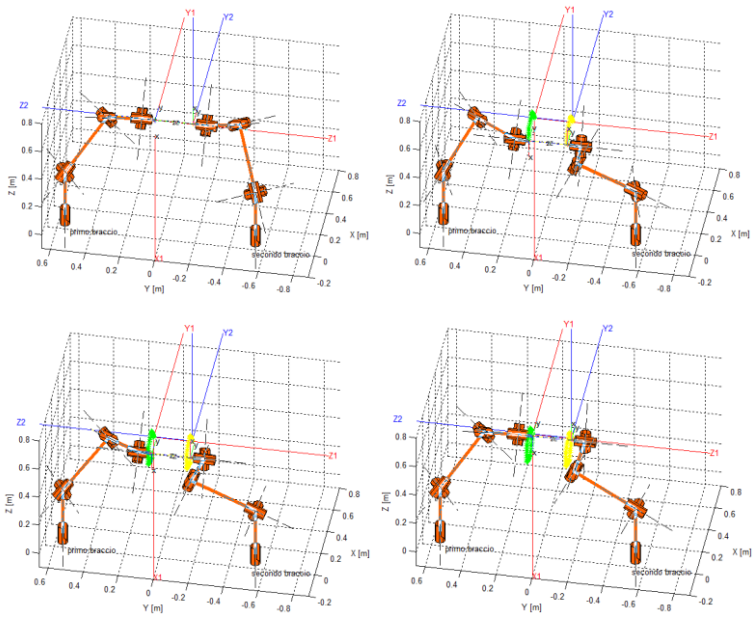


Figura 6.11 : Traiettoria pentagonale seguita con algoritmi "CLIK" e "CLIK con task secondario per avere configurazione di massima controllabilità per il robot 2 "

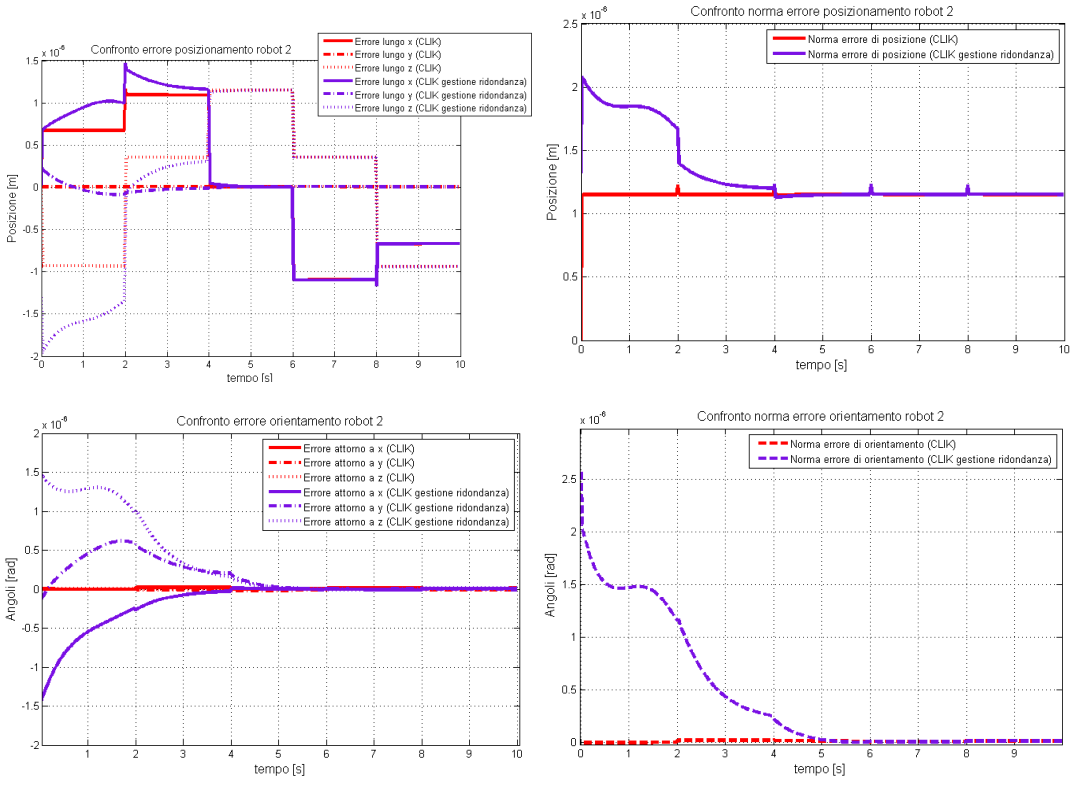


Figura 6.12 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 2, con algoritmi "CLIK" e "CLIK con task secondario per avere configurazione di massima controllabilità"

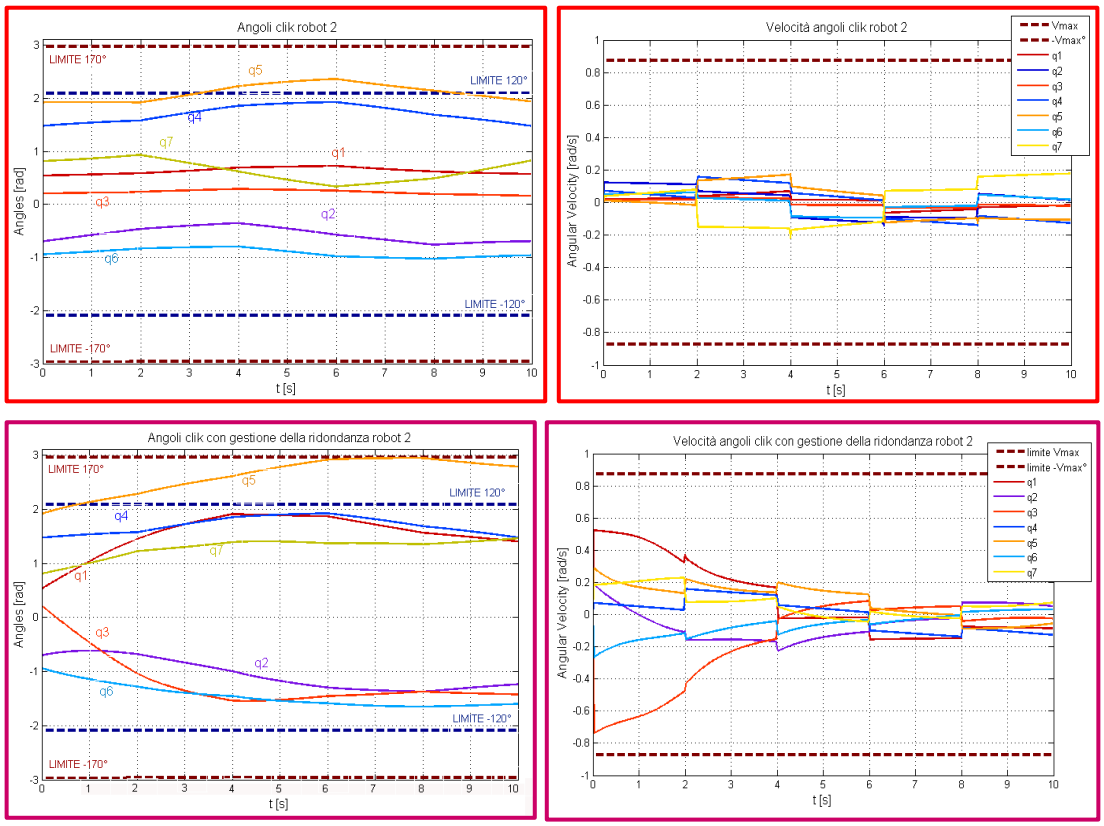


Figura 6.13 : Andamenti degli angoli e delle velocità angolari per il robot 2, con algoritmi "CLIK" (cornice rossa) e "CLIK con task secondario per avere configurazione di massima controllabilità" (cornice viola)

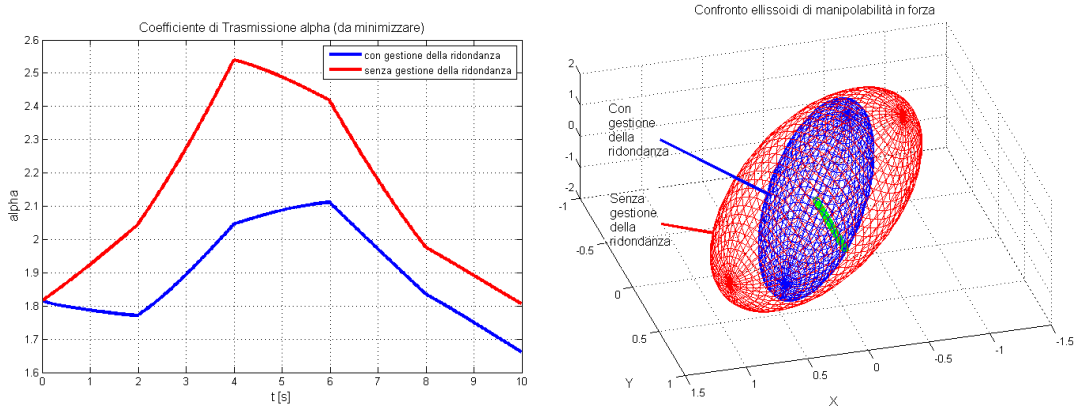


Figura 6.14 : Sinistra: Andamento del coefficiente di trasmissione di forza lungo la congiungente degli end effectors. Destra: ellissoidi di manipolabilità in forza in corrispondenza della configurazione finale nei casi in cui vi sia gestione della ridondanza o meno. In verde la distanza tra centro e superficie dell'ellissoide lungo la direzione di interesse

Dalla figura 6.14 si evince che effettivamente con la gestione della ridondanza si ha una riconfigurazione del robot a garantire un coefficiente di trasmissione di forza minore rispetto a quello che si avrebbe senza tale gestione. Questo porterà ad avere, a pari forza attuata ai giunti, minor forza esprimibile all'organo terminale come è desiderabile in situazioni ove si vuole controllare finemente la forza all'end effector. Gli svantaggi saranno un consumo di energia che si allontana dal minimo possibile garantito dalla pura pseudo-inversione e la possibilità di diminuire la manipolabilità globale, come si vede sempre nella figura 6.14. Infatti si è ottenuto di ridurre la distanza tra centro e superficie dell'ellissoide lungo la direzione desiderata ma a costo di ridurre il volume totale dell'ellissoide.

Fino a questo esempio si è limitata la velocità del task principale (si è percorsa la traiettoria pentagonale una sola volta). E' possibile testare il caso in cui tale velocità viene aumentata: nel seguente esempio si percorrerà la traiettoria pentagonale cinque volte, mentre si cercherà di eseguire sia un task secondario di allontanamento da un ostacolo posizionato nei pressi del primo braccio, sia un task secondario di allineamento per massimizzare la controllabilità del secondo braccio.

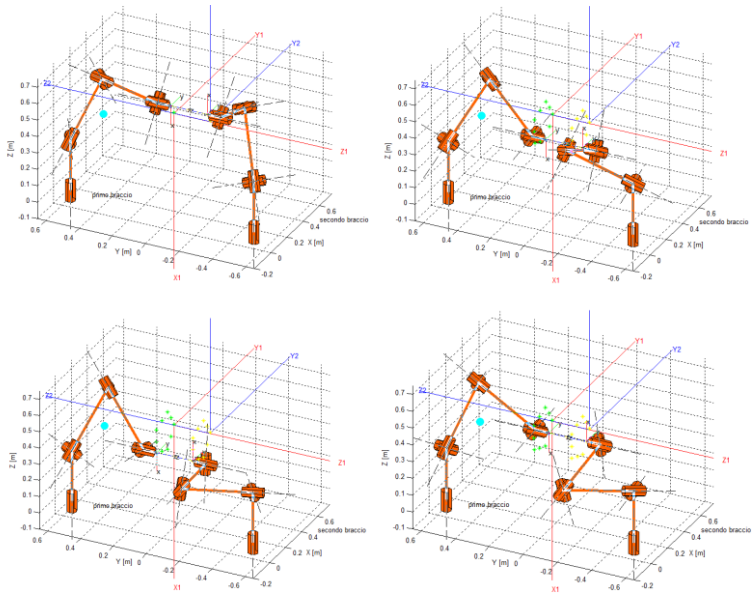


Figura 6.15 : Traiettoria pentagonale (5 giri) seguita con algoritmi "CLIK" e "CLIK con task secondario: obstacle (azzurro) avoidance per il robot 1, massima controllabilità per il robot 2"

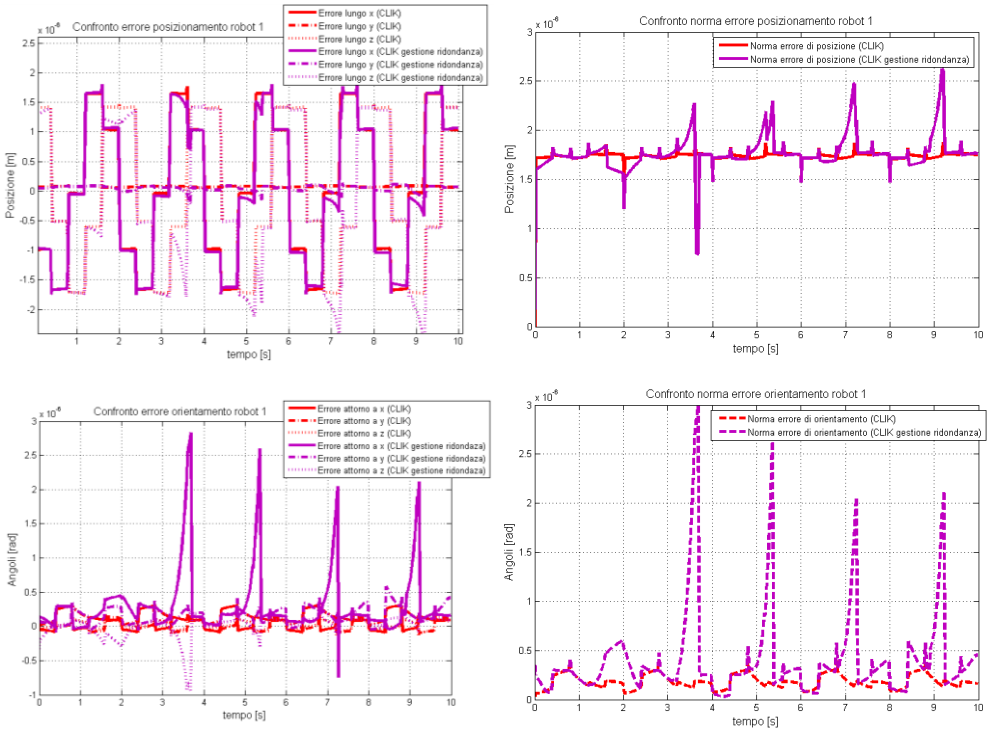


Figura 6.16 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1 con "CLIK" e "CLIK con task secondario"

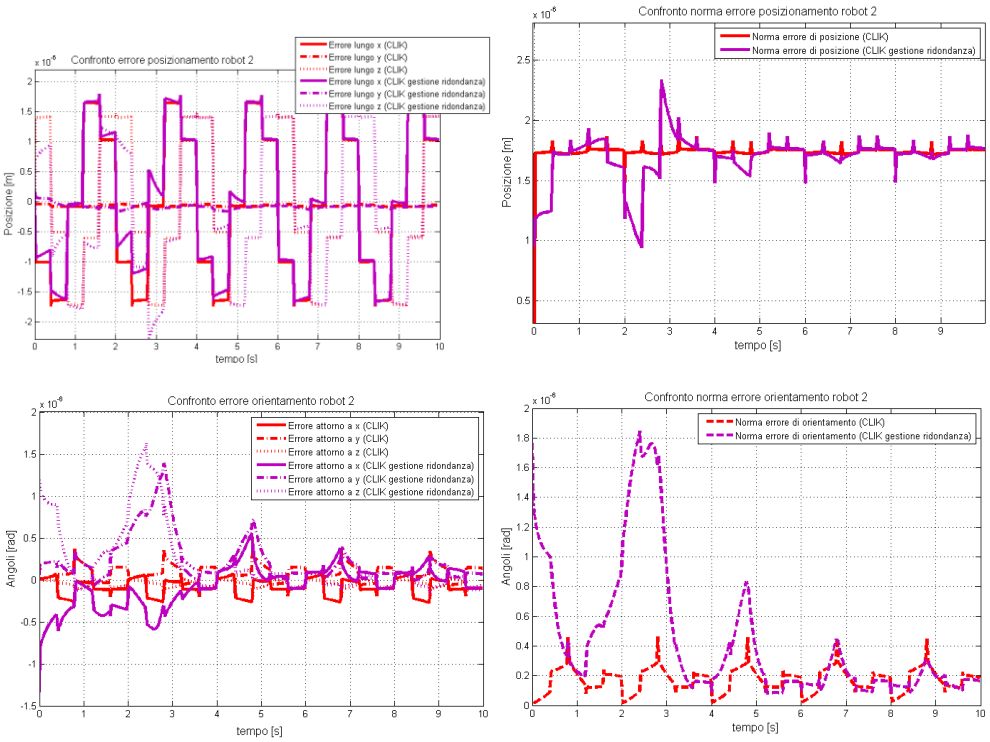


Figura 6.17 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 2 con "CLIK" e "CLIK con task secondario"

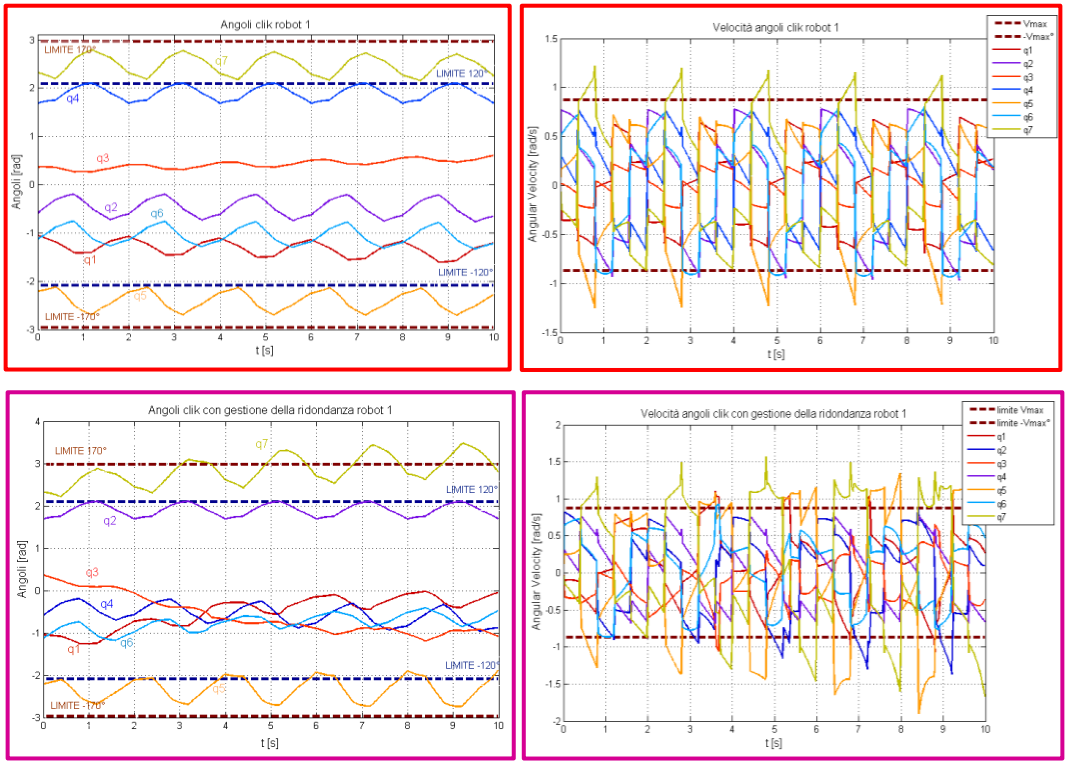


Figura 6.18 : Andamenti degli angoli e delle velocità angolari per il robot 1 con "CLIK" (cornice rossa) e "CLIK con task secondario di allontanamento da ostacolo" (cornice viola)

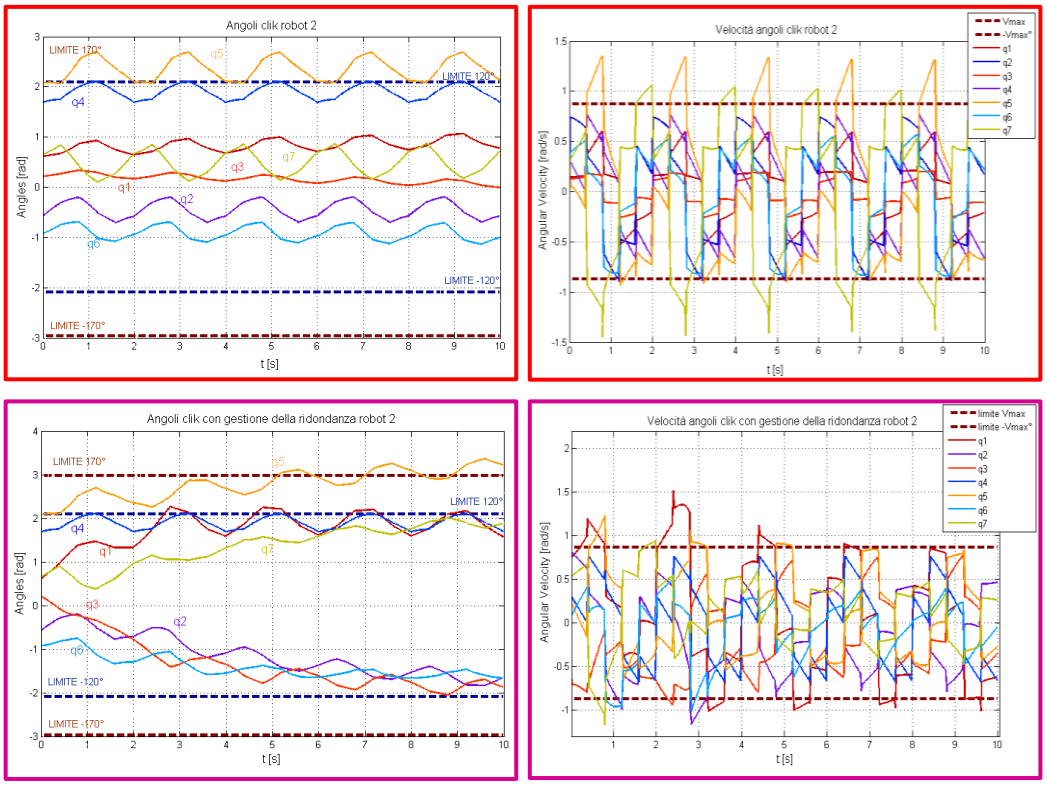


Figura 6.19 : Andamenti degli angoli e delle velocità angolari per il robot 2 con "CLIK" (cornice rossa) e "CLIK con task secondario di massima controllabilità" (cornice viola)

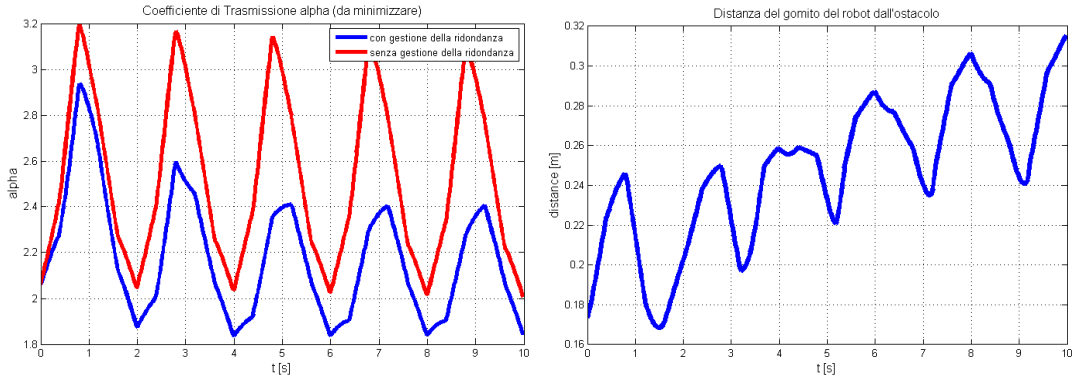


Figura 6.20 : Sinistra : Coefficiente di trasmissione di forza per il robot 2 (con e senza esecuzione task secondario). Destra : distanza ostacolo – gomito del robot 1

Quello che si vede è che si riesce ancora a soddisfare task secondari e mantenere un errore irrisorio sul posizionamento all'organo terminale, ma ora per ottenere ciò si sono superati i limiti di giunto e di velocità massima ai giunti ammissibili. Se si replicasse ciò sul sistema reale si avrebbe un blocco del robot. Si passa quindi a introdurre la saturazione sugli angoli e le velocità di giunto, e a introdurre il criterio di saturazione nel null space (SNS) che è finalizzato al superamento di questo genere di problemi. Nelle seguenti figure si confrontano i risultati ottenibili con Pseudoinversa semplice, CLIK, SNS nel caso di saturazione dei limiti di giunto.

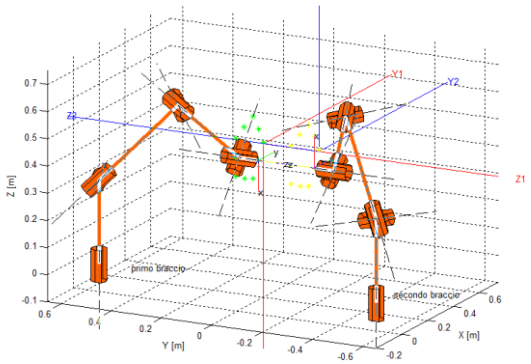


Figura 6.21 : Traiettoria pentagonale (5 giri) seguita con algoritmi "Pinv Semplice", "CLIK", "SNS"

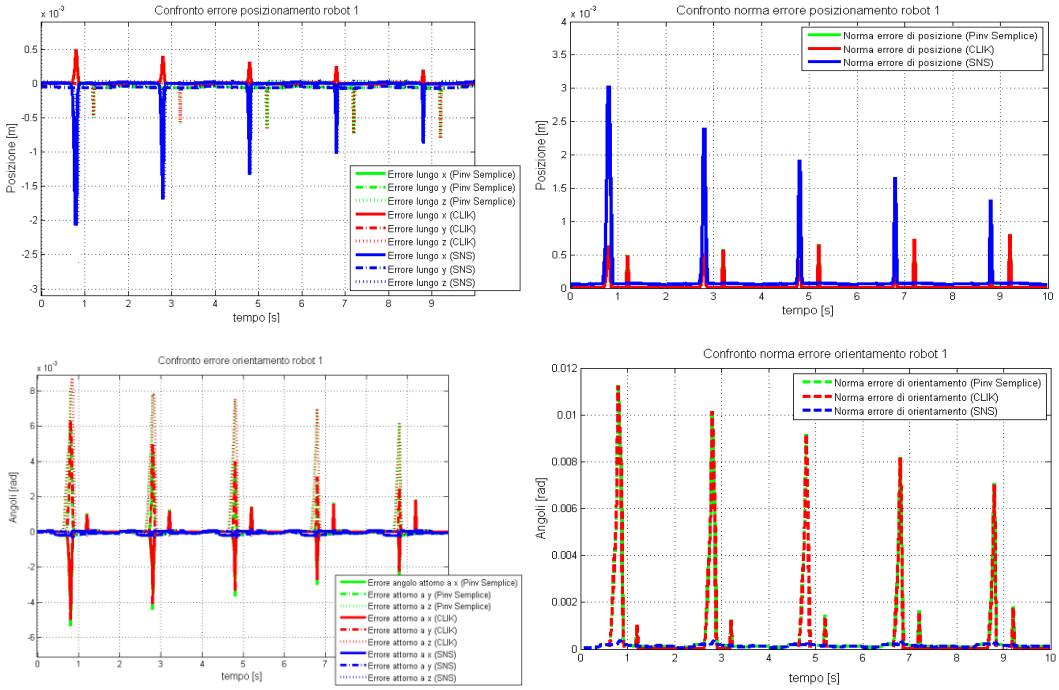


Figura 6.22 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi " $J^{\#}$ ", "CLIK", "SNS"

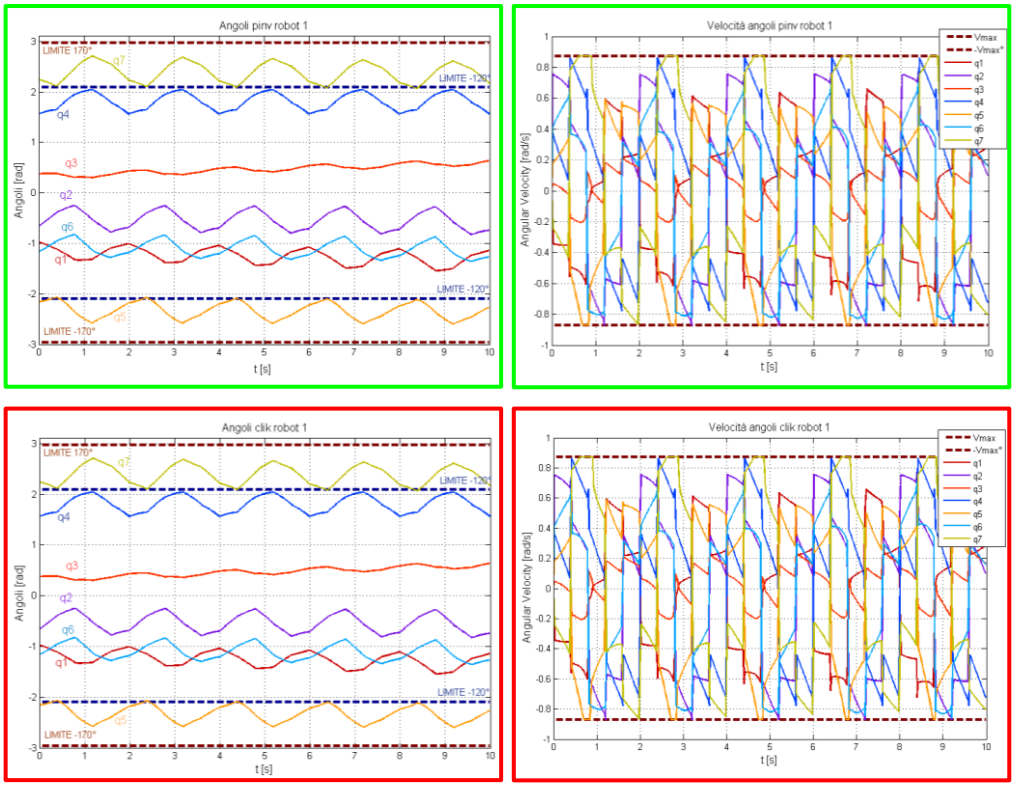


Figura 6.23 : Andamenti degli angoli e delle velocità angolari per il robot 1, con algoritmi " $J^{\#}$ " (cornice verde) e "CLIK" (cornice rossa)

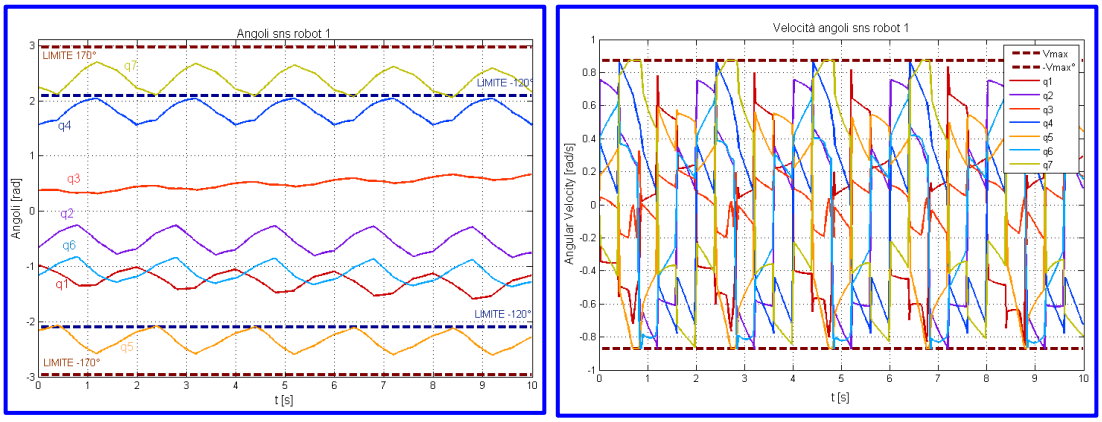


Figura 6.24 : Andamenti degli angoli e delle velocità angolari per il robot 1, con algoritmo "SNS" (cornice blu)

Si noti che l'algoritmo SNS si comporta meglio degli altri "tradizionali" per quanto riguarda l'errore di orientamento, ma è peggiore degli altri in termini di posizione. Tuttavia, per come è definito, l'algoritmo migliorerà notevolmente quanti più gradi di libertà ridondanti si avranno a disposizione come ad esempio per task assialsimmetrici al TCP. Infatti se ad esempio si imponesse solo il rispetto della posizione (4 gradi di libertà ridondanti), l'algoritmo SNS si comporterebbe decisamente meglio degli altri, come mostrato nelle seguenti figure relative allo svolgimento di sette giri del task con traiettoria pentagonale.

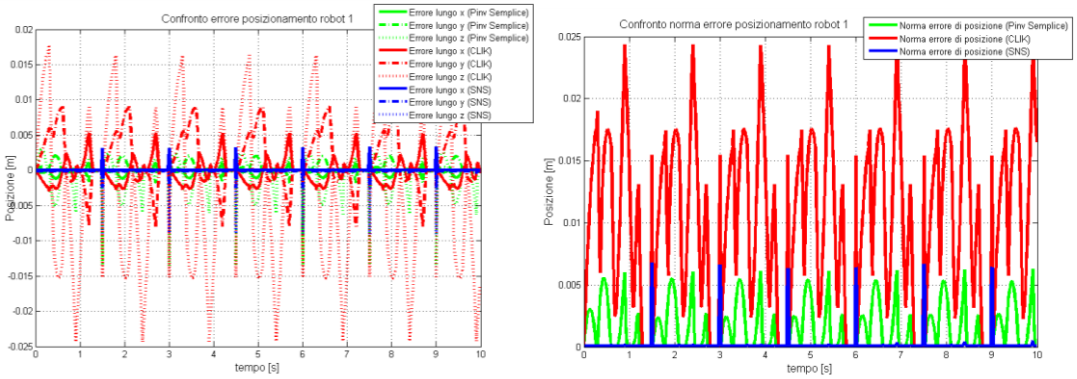


Figura 6.25: Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi " $J^{\#}$ " , "CLIK", "SNS"

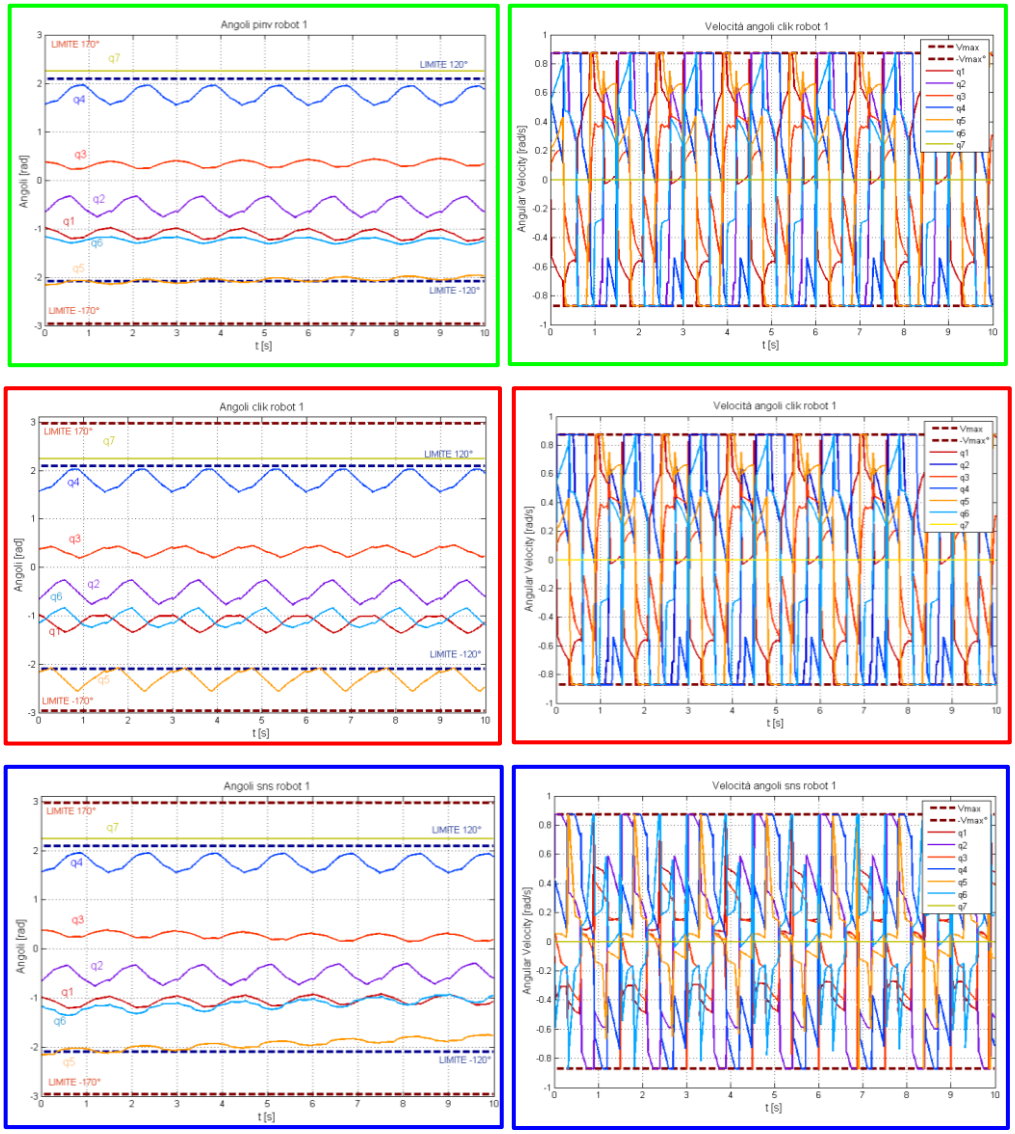


Figura 6.26 : Andamenti degli angoli e delle velocità angolari per il robot 1 con algoritmi " $J^{\#}$ " (cornice verde), "CLIK" (cornice rossa), "SNS" (cornice blu)

Si presenta quindi un esempio di algoritmo SNS completo anche di task secondari (si imporrà la massimizzazione della controllabilità per il secondo robot lungo la direzione congiungente dei due robot), e con l'incorporamento del vincolo di *obstacle avoidance* (vedi Paragrafo 5.3) per il primo braccio. Questo tipo di vincolo sarà rispettato deterministicamente, a differenza dei task secondari di allontanamento che non garantirebbero che l'ostacolo venga effettivamente evitato.

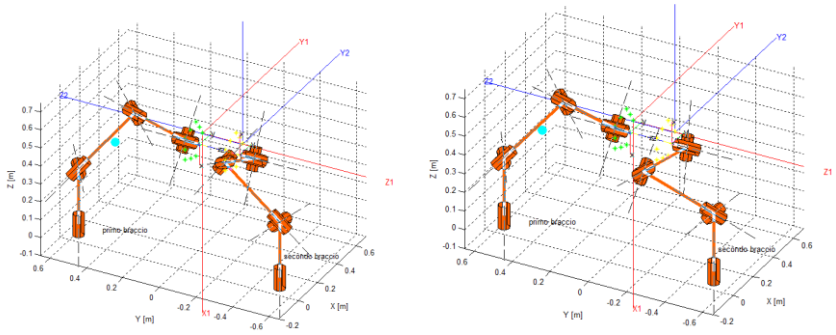


Figura 6.27 : Traiettoria pentagonale (5 giri) seguita con algoritmi "SNS"e "SNS con vincoli e task secondari"

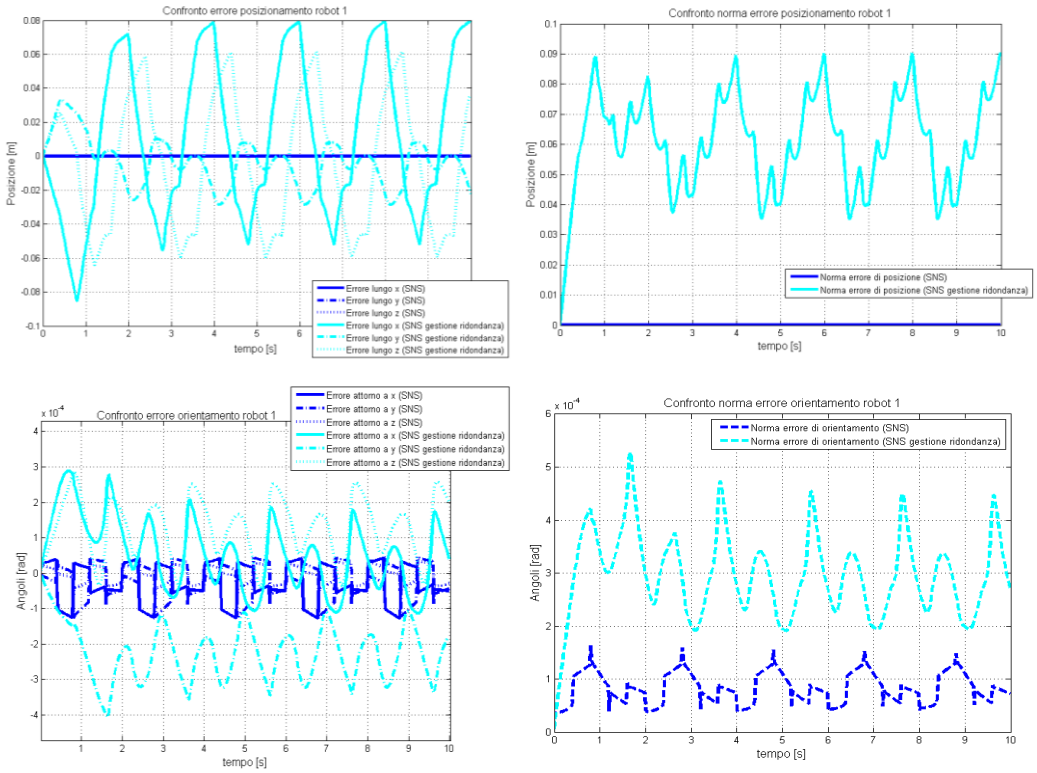


Figura 6.28 : Errori di tracking al giunto (sinistra) e in norma (destra) per il robot 1, con algoritmi "SNS"e "SNS con vincoli e task secondari"

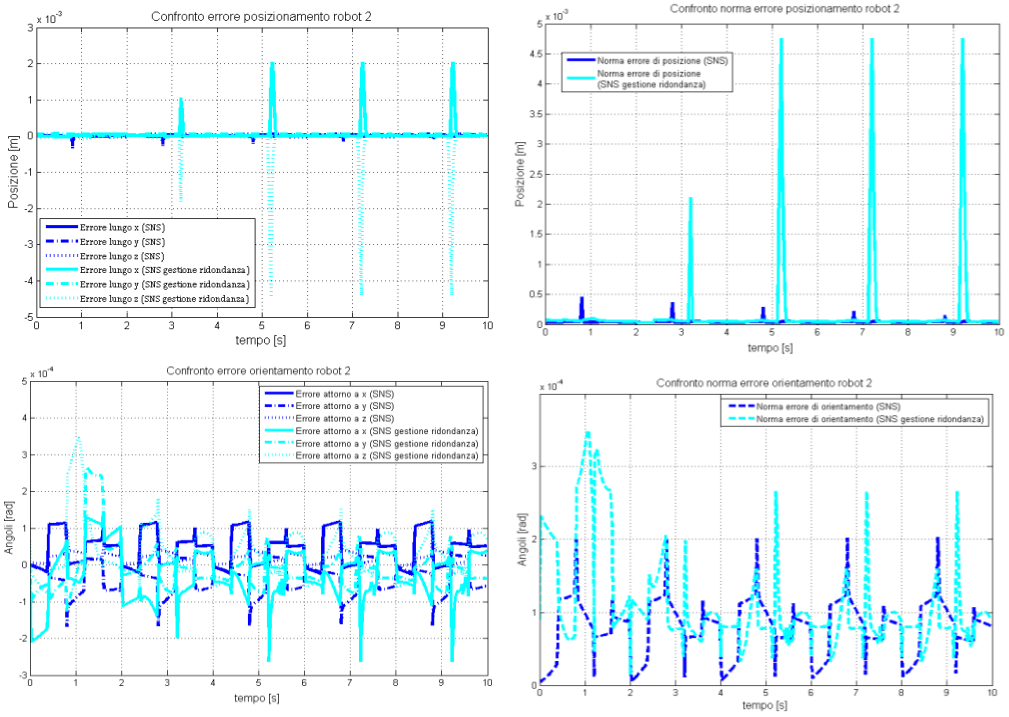


Figura 6.29 : Errori di tracking al giunto (sinistra) e in norma (destra) robot 2, con "SNS"e "SNS con vincoli e task secondari"

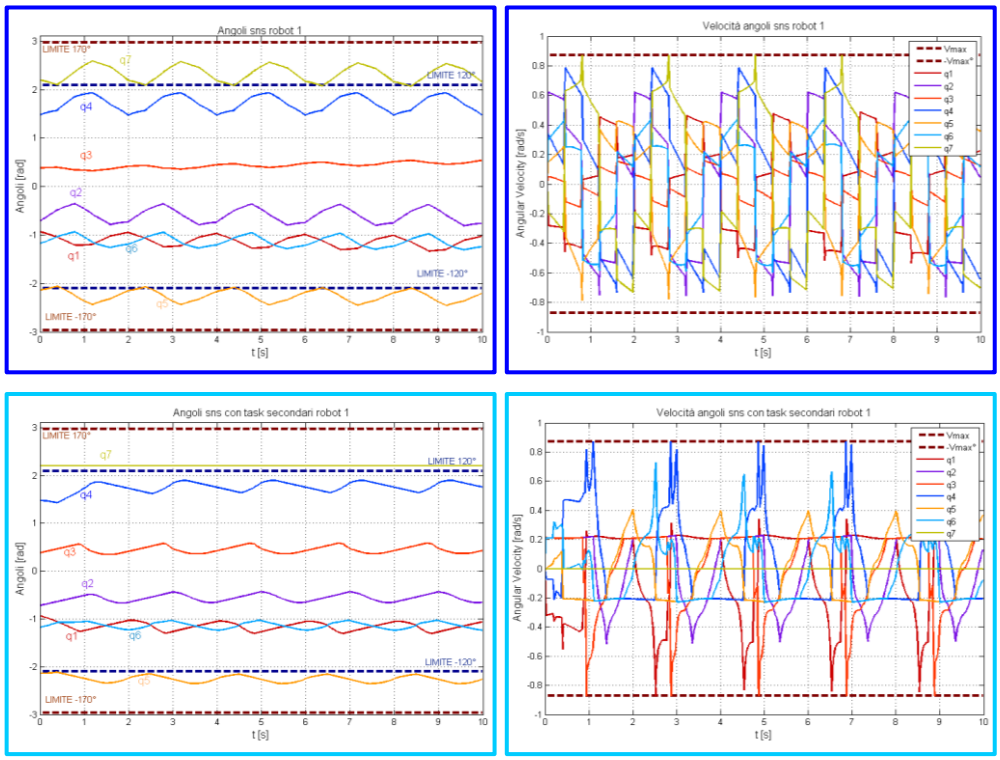


Figura 6.30 : Andamento angoli e velocità angolari per il robot 1 con algoritmi "SNS " (cornice blu) e "SNS con vincolo di distanza da ostacolo" (cornice azzurra)

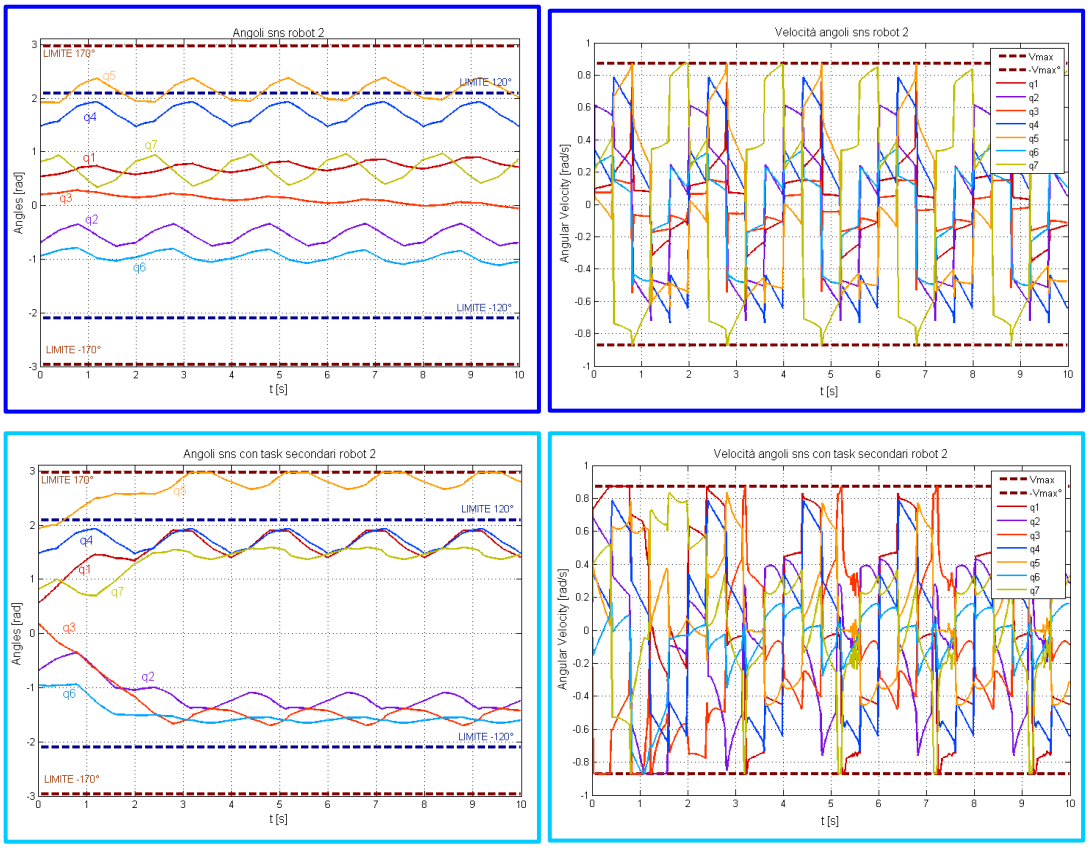


Figura 6.31 : Andamento angoli e velocità angolari per il robot 2 con algoritmi "SNS " (cornice blu) e "SNS con task secondario di massima controllabilità" (cornice azzurra)

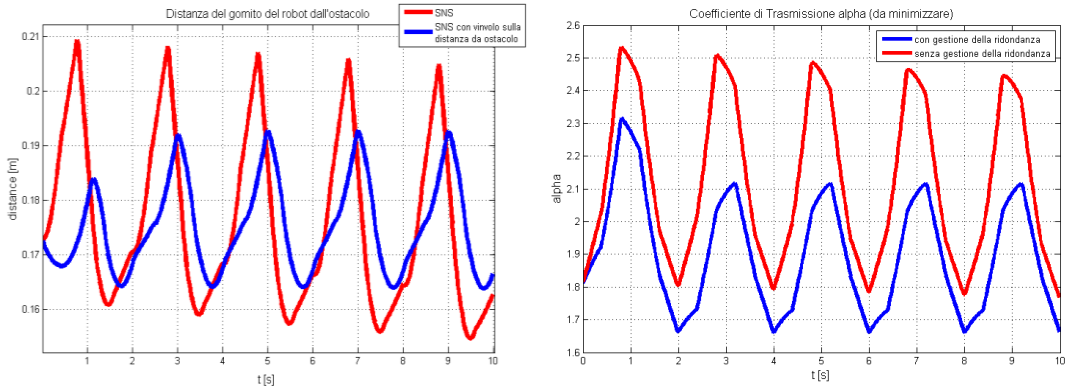


Figura 6.32 : Distanza tra ostacolo e gomito del primo robot e coefficiente di trasformazione di coppia per il secondo robot

Come aspetto notevole di questo esempio si nota che per il primo robot la distanza tra gomito e ostacolo ha un minimo che non è valicabile perché in tal caso si blocca la traiettoria lungo i gradi di libertà che porterebbero all'impatto . Questo però può

comportare un severo errore di inseguimento, se l'ostacolo è posizionato (come in questo caso) lungo il tracciato che sarebbe assunto dal gomito nello svolgimento del task.

Si concludono gli esempi mostrando la situazione che si troverebbe se si avesse una vibrazione lungo l'asse x con 3 Armoniche e fondamentale a 1 Hz ampia 3 cm. Si vede che basta questo tipo di disturbo per portare a velocità molto elevate ai giunti, per cui metodi appositi per la gestione della saturazione di velocità e limiti di giunto saranno utili. Si utilizzerà quindi un algoritmo SNS con gestione della ridondanza volta a portare gli angoli verso il centro corsa e limiti angolari progressivi come descritto nel paragrafo 5.3.

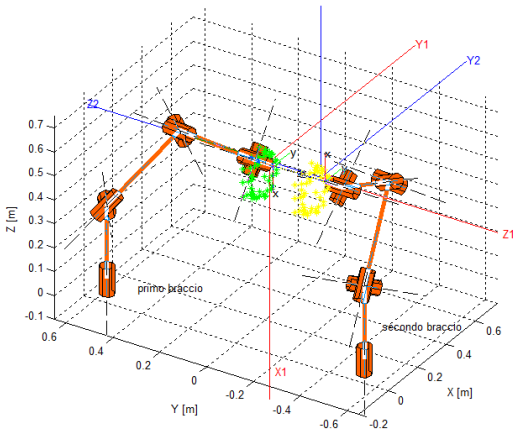


Figura 6.33 : Traiettoria pentagonale con sovrapposizione di oscillazioni periodiche di disturbo

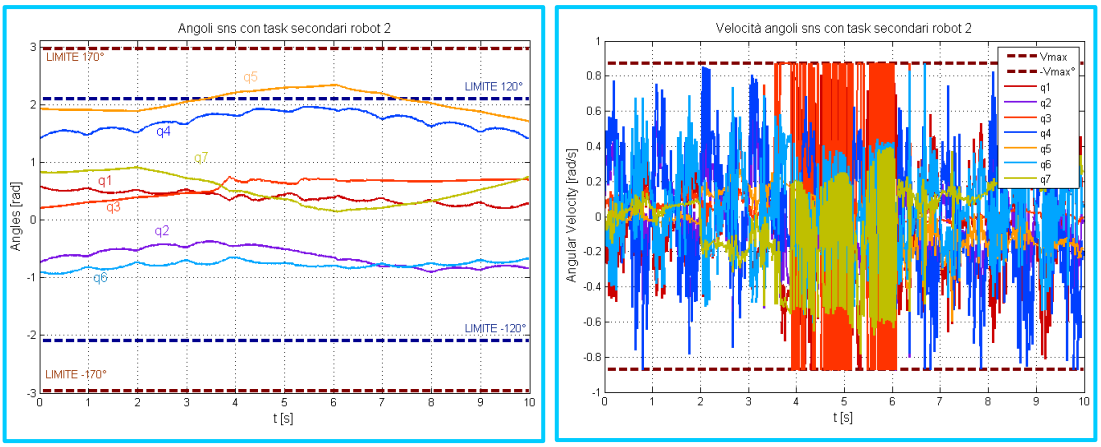


Figura 6.34 : Andamento degli angoli e delle velocità per il robot 2, caso con oscillazioni di disturbo

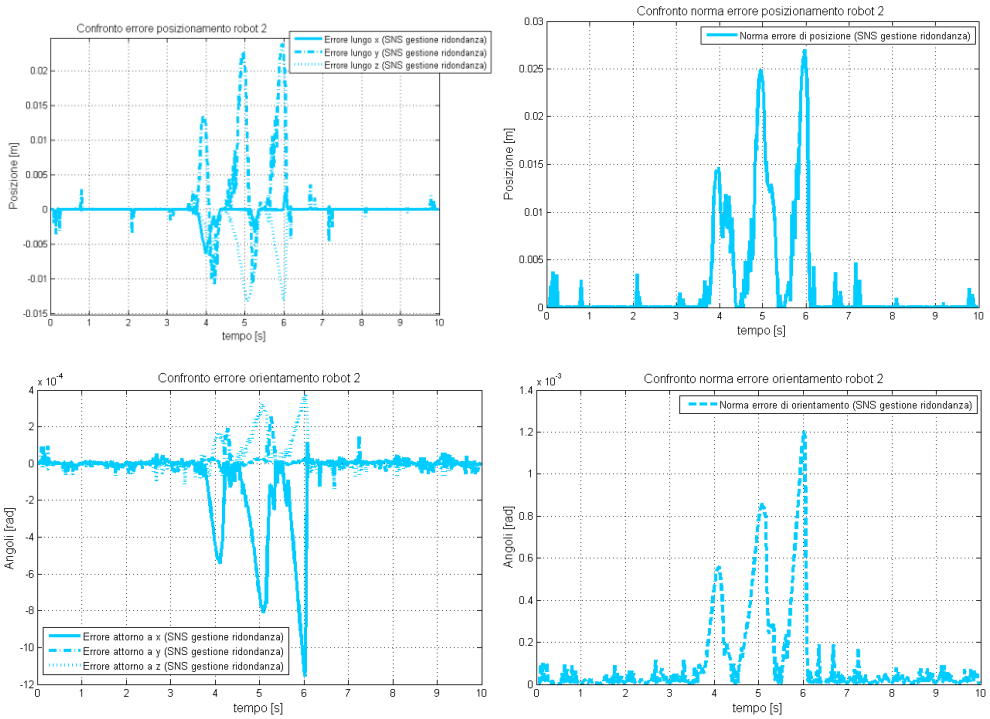


Figura 6.35 : Andamento degli errori di tracking tracking al giunto (sinistra) e in norma (destra) per il robot 2, caso con oscillazioni di disturbo

6.3 Implementazione C++ : La Classe RobotModel

Come si era fatto per gli algoritmi di coordinazione, anche per quelli di inversione cinematica si procede alla scrittura di un codice per il controllo del sistema reale.

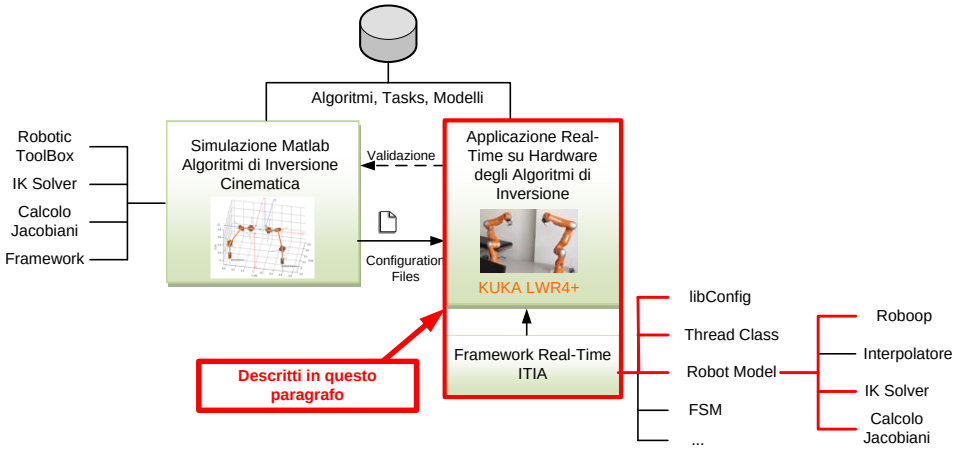


Figura 6.36 : Collocazione del Paragrafo

Riferendosi allo schema dell'applicazione sviluppata per il sistema reale (vedi Paragrafo 3.3), si ha che l'inversione cinematica trova collocazione all'interno del thread di coordinazione (MROE). Qui, dopo l'esecuzione dei calcoli matriciali da parte dell'oggetto di classe *mroeAgent* per legare i vari elementi del framework multi-robot (vedi Cap. 2), si vogliono ottenere tali posizionamenti mediante un'inversione cinematica eseguita come funzione dell'oggetto *mroeAgent*, il quale trasferirà i risultati nella struttura dati dalla quale il thread FSM effettuerà lo scambio dati fisico (via FRI con UDP) con il robot.

La cinematica inversa sarà ottenuta mediante gli algoritmi presentati e testati in precedenza, implementati all'interno di una classe scritta appositamente per gestire la cinematica dei manipolatori: la classe *RobotModel*.

La *RobotModel* è una classe dedicata alla codifica delle funzioni che appunto vanno a calcolare la cinematica inversa secondo gli algoritmi espressi, che contiene a sua volta oggetti di classe ROBOOP [28] utilizzati per la modellistica di base e le funzioni di inversione dello Jacobiano. Di seguito si riporta la dichiarazione della classe:

```
class RobotModel
{
public:
    RobotModel(std::string roboopConfigfile, std::string roboopConvention, std::string ikineConfigfile);
    ~RobotModel();

    void Init ();
    void ConfigIKine( std::string ikineConfigfile );
    void UpdateData( const NEWMAT::ColumnVector&  q0 );
    void UpdateJA ();
    void UpdateJA ( NEWMAT::ColumnVector dof_inv);
    void UpdateJG ();
    void UpdateTa ();
    double getCartesianDistance( const NEWMAT::ColumnVector& q, const NEWMAT::Matrix& T );
    NEWMAT::ReturnMatrix    getJointValues();
    NEWMAT::ReturnMatrix    getCartesianPose( const NEWMAT::ColumnVector& q);
    bool doCLIK(NEWMAT::Matrix Td, double dt, const NEWMAT::ColumnVector& qstart,
                NEWMAT::ColumnVector dof_inv, NEWMAT::ColumnVector u_dir );
    bool doSNS( NEWMAT::ColumnVector xd, double dt, const NEWMAT::ColumnVector& qstart,
                NEWMAT::ColumnVector dof_inv, NEWMAT::ColumnVector u_dir );
    const ROBOOP::Robot& getRobot()   { return robot; };

private:
    ROBOOP::Robot robot;
    NEWMAT::Matrix JA, JG, Ta, Rnum;
    NEWMAT::ColumnVector lwroffset;
    ITIA::KUKA::CLIKdata clikdata;
    ITIA::KUKA::SNSdata snsdata;
};
```

Quindi, quando si dovesse calcolare la cinematica inversa, si potrà creare un oggetto di questa classe che conterrà tutte le informazioni per eseguire tale calcolo.

Infatti mediante il costruttore si caricheranno i files di configurazione contenenti tutti i dati sui manipolatori (parametri di Denavit-Hartenberg, limiti di giunto, etc.) con cui costruire l'oggetto ROBOOP::Robot e tutti i dati da usare nei metodi di inversione (guadagni CLIK e task secondari, etc.) implementati nelle funzioni *doCLIK* e *doSNS*. Queste funzioni prenderanno dal programma a livello superiore che le richiama anche dati come il numero di gradi di libertà da tenere in considerazione nell'inversione cinematica o in generale i dati variabili istante per istante come la direzione lungo cui calcolare la controllabilità, l'insieme degli angoli ai giunti all'istante precedente, la matrice omogenea o la posa desiderata all'istante in questione.

Nella classe ci saranno poi funzioni di inizializzazione per dimensionare e riempire le variabili utili in *doCLIK* e *doSNS* e funzioni di utilità come *updateJA, update JG, updateTa*. Quindi per usare questa classe un esempio può essere quello che segue:

```
ITIA::KUKA::RobotModel rob("file_dati_robot","file_dati_ikine");
rob.Init();
rob.doSNS(xd, dt,  qstart, dof_inv, u_dir );
ColumnVector q_FRI = rob.getRobot().get_q();
```

In tal modo sarà facile usare oggetti RobotModel anche in altri programmi ove sia richiesta la cinematica inversa, con il vettore finale di angoli da passare al robot.

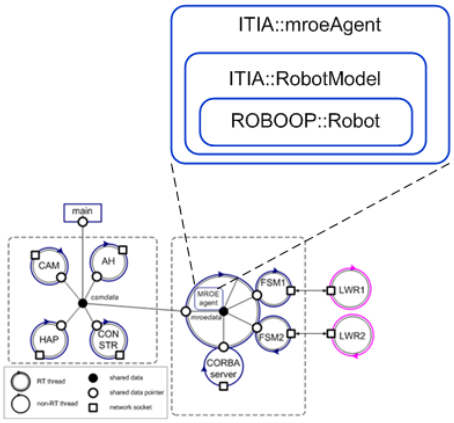


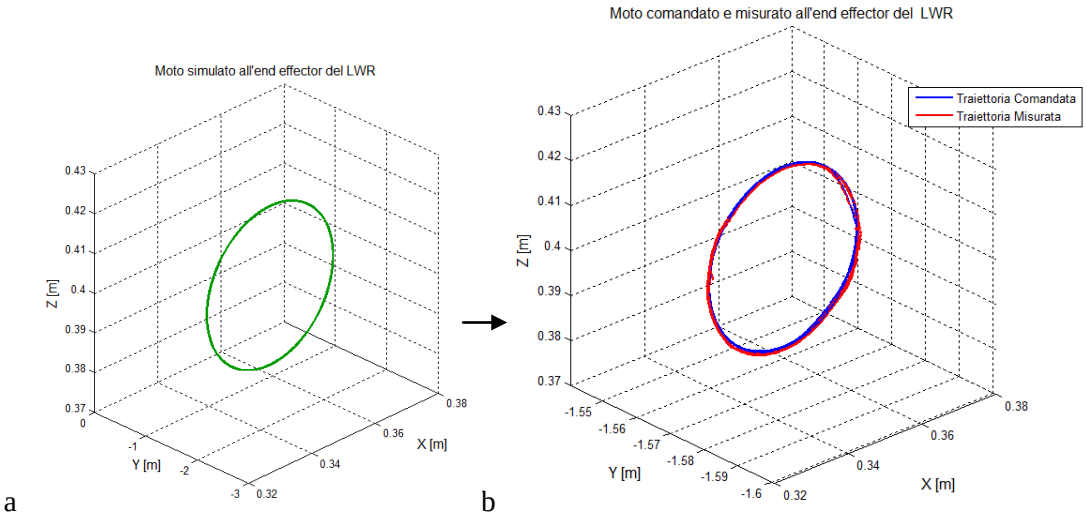
Figura 6.37 : Rappresentazione applicazione e gerarchia classi

Dal punto di vista della realizzazione delle funzioni *doCLIK* e *doSNS* il punto critico è stato ancora ricavare espressione analitica per Jacobiani e gradienti di funzioni per task secondario (soprattutto per il caso della controllabilità). Si è eseguito il procedimento esposto in 6.1 esportando però la funzione in C++ grazie al comando *ccode* di Matlab.

6.4 Risultati Sperimentali

Una volta scritta la classe per gestire l'inversione cinematica si sono validati i risultati mediante l'applicazione degli algoritmi sul sistema reale. Ci si focalizza sull'algoritmo SNS, di cui si riportano i risultati sperimentali.

Un primo esempio riporta l'algoritmo funzionante nell'inseguimento di una traiettoria circolare. Nelle seguenti figure è mostrata la traiettoria realizzata in Matlab (Fig. 6.36a) ed esportata sul sistema reale mediante file di configurazione + interpolatore (vedi Paragrafo 3.2)e quelle che hanno costituito l'input e l'output del sistema reale (Fig. 3.36b)



a b

Figura 6.38: (a) Traiettorie Simulata; (b) Traiettorie Comandata e Misurata sul sistema reale

E' possibile notare che l'inseguimento presenta errori di posizionamento dell'ordine del decimo di millimetro lungo le varie direzioni (concentrati soprattutto in

direzione ortogonale al moto) , e una norma dell'errore con massimi compresi tra 1 mm e 1.1 mm). Le seguenti figure mostrano l'andamento dell'errore di posizionamento. L'errore di orientazione è dell'ordine del decimillesimo di radiante.

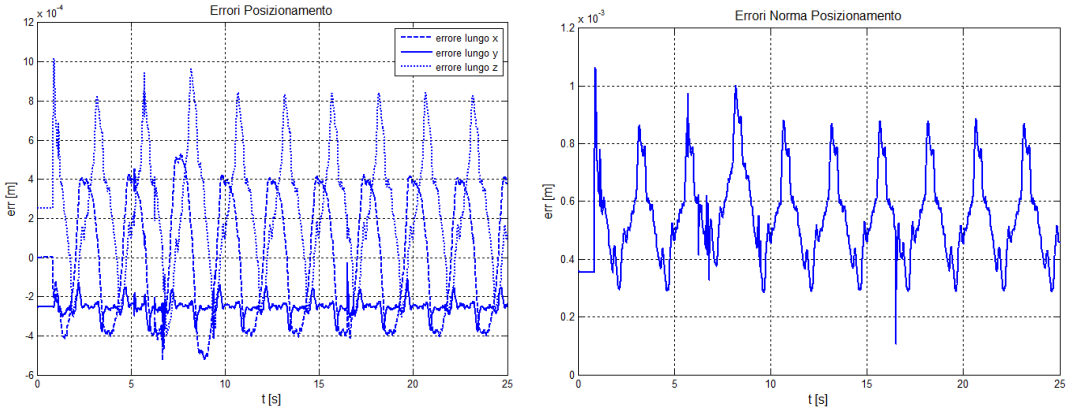


Figura 6.39 : Errori posizionamento con algoritmo SNS applicato al sistema reale

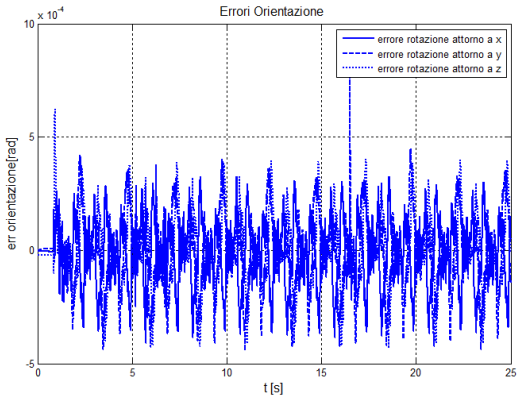


Figura 6.40 : Errori orientazione, con algoritmo SNS applicato al sistema reale

Si possono quindi confrontare gli andamenti degli angoli. Si riscontra sempre una buona corrispondenza tra risultati simulati, angoli calcolati dall'algoritmo di inversione ed angoli effettivamente assunti dal robot.

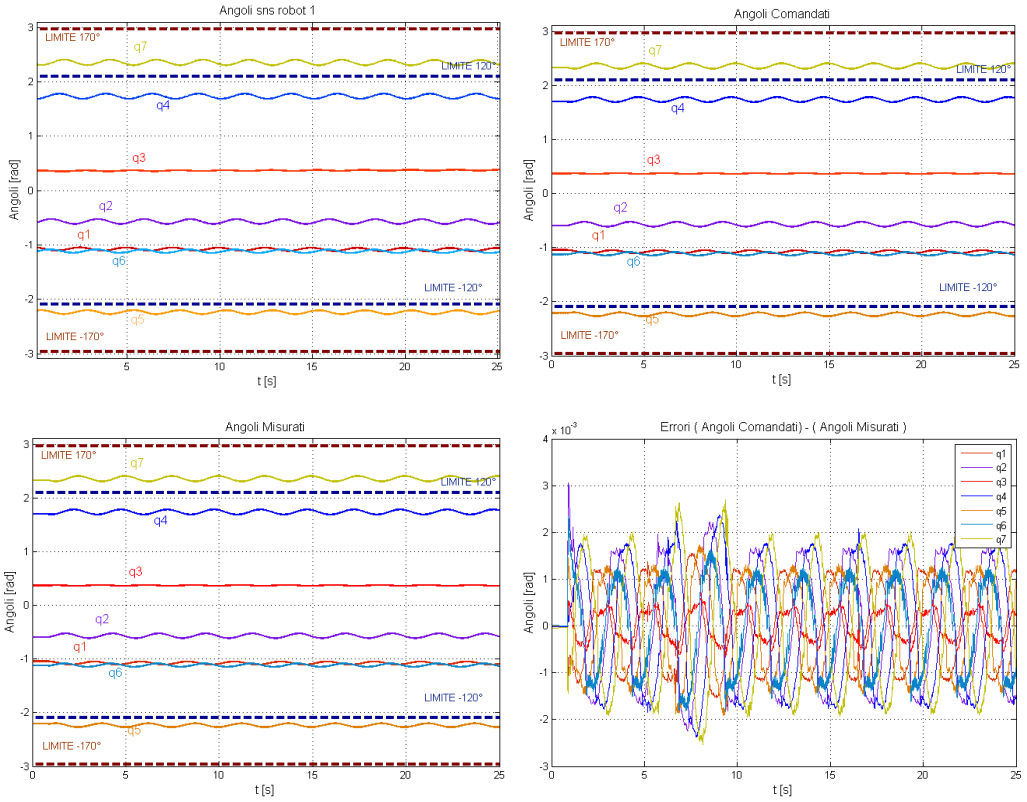


Figura 6.41 : Angoli ai giunti simulati, in input e output al sistema reale, algoritmo SNS.

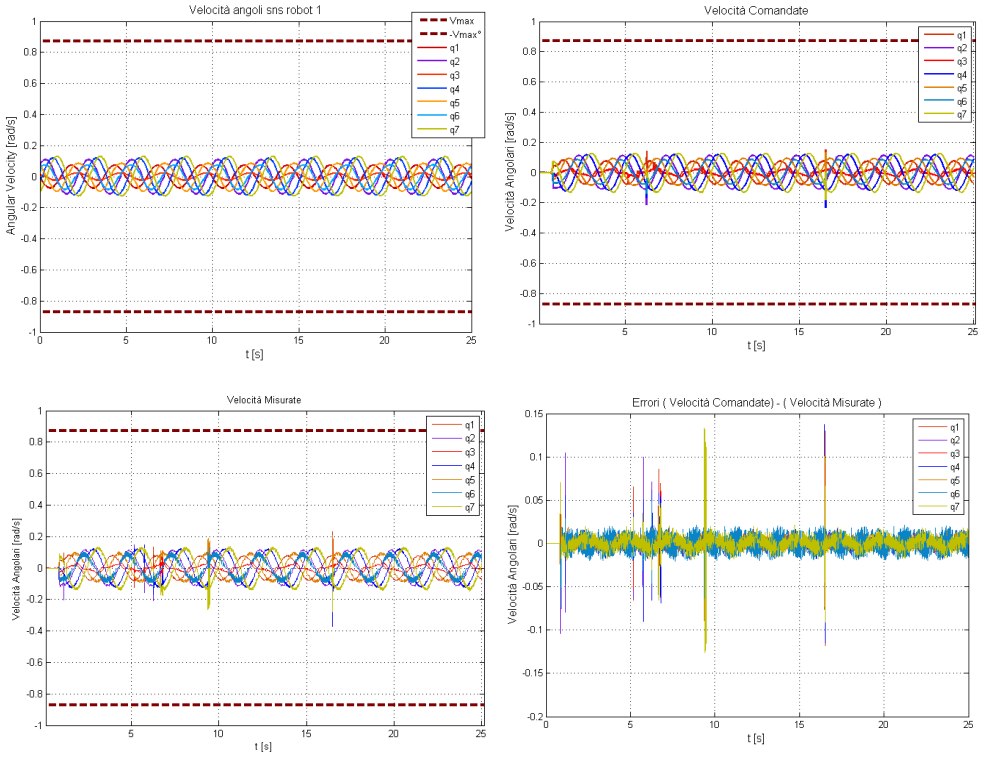


Figura 6.42 : Velocità ai giunti simulate, attese e misurate sul sistema reale, algoritmo SNS.

Si può dedurre che anche a livello di velocità esiste buona corrispondenza tra angoli simulati in Matlab, angoli ottenuti dall'algoritmo implementato sul programma di controllo del robot e angoli effettivamente misurati. La natura delle oscillazioni e delle differenze a livello di velocità si può valutare in base alla seguente figura di dettaglio che riguarda il giunto con maggiori errori, in un singolo giro di circonferenza.

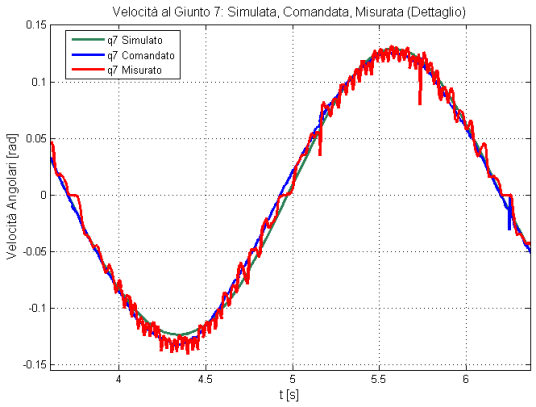


Figura 6.43 : Velocità al Giunto 7. Simulata, Desiderata, Misurata (Dettaglio)

La differenza tra andamento delle velocità angolari simulate e comandate è imputabile innanzitutto al fatto che la traiettoria per il sistema reale è stata ricostruita su un numero limitato di punti estratti dalla sequenza presente in ambiente di simulazione tramite interpolatore (vedi Paragrafo 3.2). Passando alla differenza tra velocità angolari comandate e misurate, essa è all'incirca dello stesso ordine della differenza tra andamento simulato e comandato. Questa differenza è imputabile ai controllori ai giunti (e quindi non dovuta all'algoritmo) , a errori di sincronizzazione nella comunicazione dei dati e a errori dovuti al fatto che si tratta di un sistema reale (attriti, vibrazioni del supporto, inaccuratezze del modello cinematico e dinamico).

Validata la bontà dell'algoritmo per la traduzione del task primario si è passati ad applicare un task secondario. Un caso in cui esso si esplicita in modo chiaro è quello in cui si impone che l'end effector stia fermo, così da avere solo moti interni che allineino la struttura del robot per ottenere il task secondario desiderato. Si è applicato il criterio della massima controllabilità in forza lungo una direzione (che in

generale sarebbe stata quella congiungente i due end effectors dei robot, ma che è stata scelta coincidente con l'asse y del sistema di riferimento assoluto per via di un guasto al secondo robot).

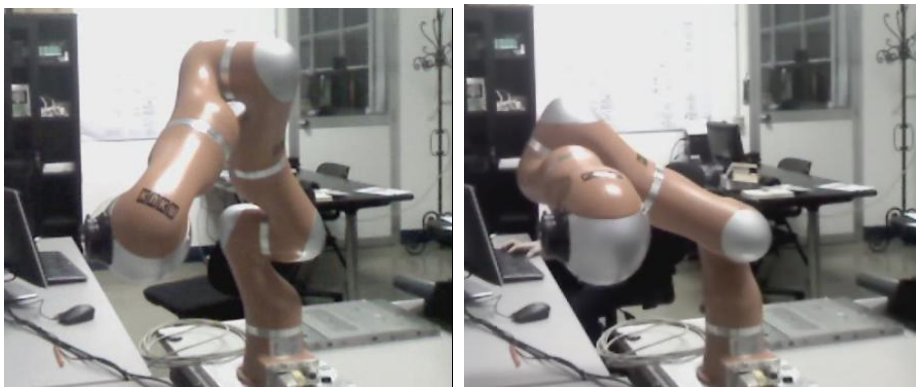


Figura 6.44 : Moti interni del robot dati da task secondario per massimizzazione della controllabilità lungo l'asse y

Ancora una volta si riscontra buona approssimazione tra gli andamenti desiderati e quelli effettivamente realizzati, e si nota la similitudine con la postura che assumerebbe un braccio umano intento a controllare la forza esercitata durante un'operazione di scrittura.

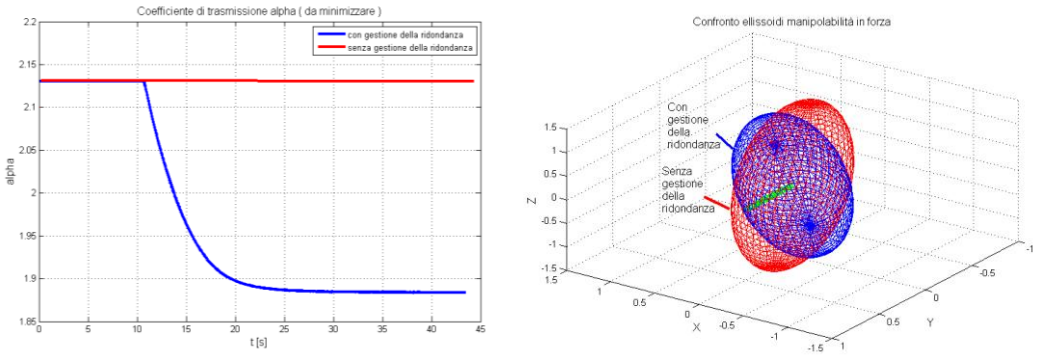


Figura 6.45: Sinistra : Andamento del coefficiente di trasmissione di forza lungo la direzione y. Destra: ellissoidi di manipolabilità in forza in configurazione finale nel caso con e senza gestione della ridondanza

Nella precedente figura 6.45 si sono ricavati i coefficienti di trasmissione di forza e gli ellissoidi di manipolabilità in forza nel caso in cui il robot si mantenesse nella

configurazione iniziale e nel caso in cui vi si applica invece il criterio di gestione della ridondanza. Si vede che l'andamento ricavato da tali dati sperimentali ricalca quello atteso, con coefficiente di trasmissione α in diminuzione ed ellissoide con proiezione lungo l'asse di interesse minore (anche se a discapito della manipolabilità globale).

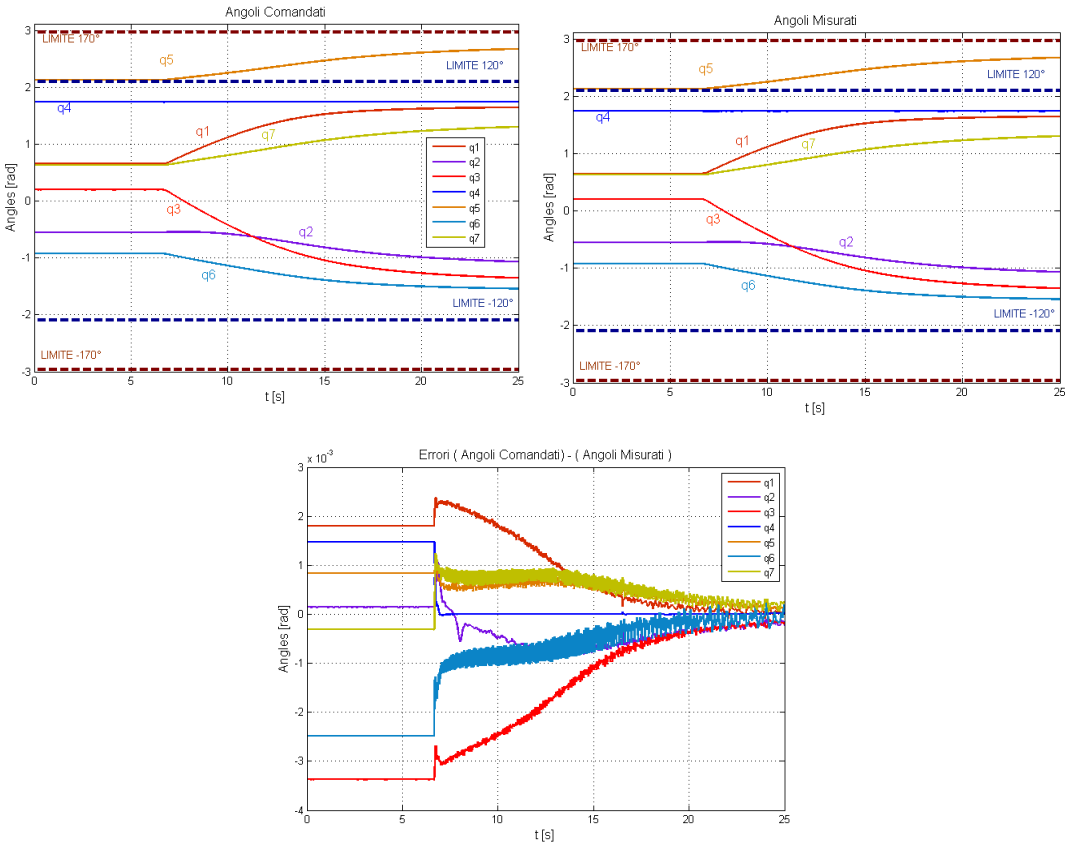


Figura 6.46 : Angoli ai giunti in input e in output al sistema reale, algoritmo SNS con gestione della ridondanza (criterio massima controllabilità)

Si nota che in questo esempio reale, si ha convergenza verso gli angoli che denotano la configurazione relativa all'ottenimento del rapporto di trasformazione di forza minimo lungo la direzione desiderata.

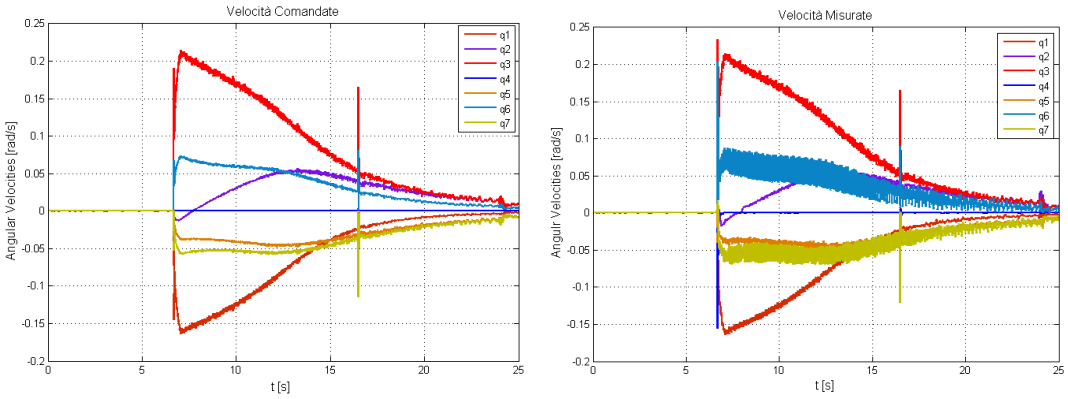


Figura 6.47 : Velocità attese e misurate sul sistema reale, algoritmo SNS con gestione della ridondanza (criterio massima controllabilità)

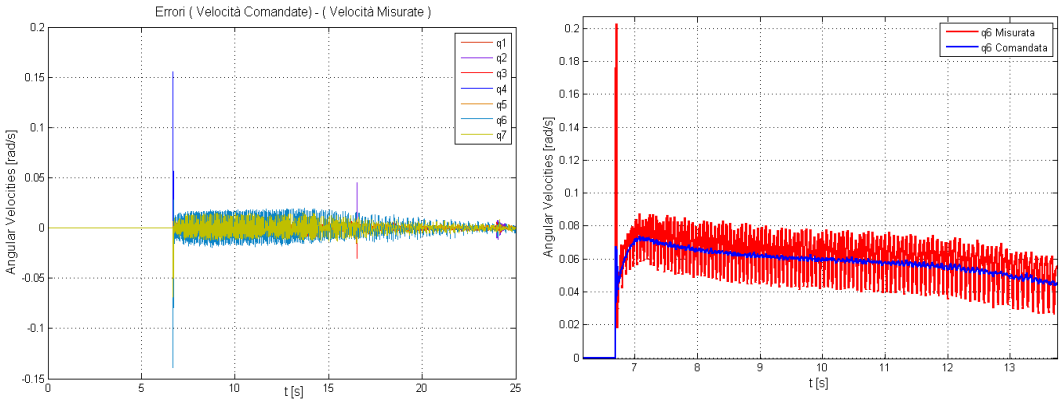


Figura 6.48 : Errori tra velocità attesa e misurata ai giunti, e particolare del sesto giunto.

Si nota che i maggiori errori si hanno in corrispondenza delle discontinuità di velocità. Esse possono essere dovute al suddetto problema di sincronizzazione dati o al fatto di non aver esplicitamente preso in considerazione l'accelerazione nella formulazione degli algoritmi e nella generazione delle traiettorie. Questo porta ad errori maggiori in corrispondenza delle discontinuità di velocità.

Capitolo 7 Conclusioni e Sviluppi Futuri

7.1 Conclusioni

Si sono realizzati i passi riassunti dalla seguente figura, già anticipata nel Capitolo 1.

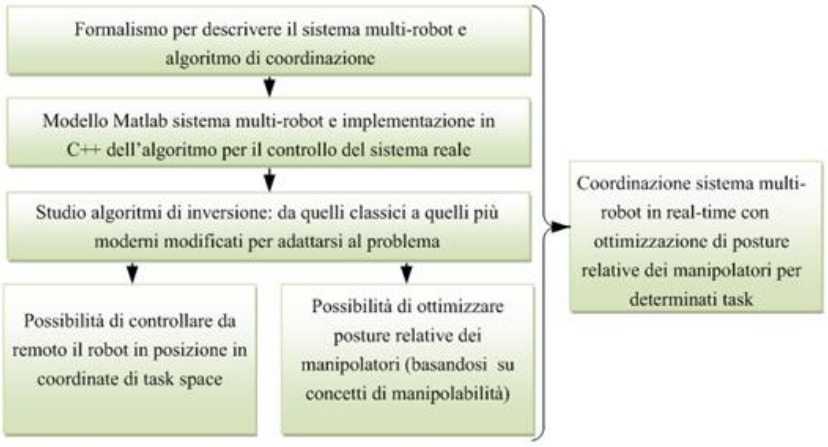


Figura 7.1 : Sinossi dei passaggi logici e degli obiettivi realizzati

Si è quindi realizzato un formalismo per descrivere il sistema multi-robot considerato e un algoritmo di coordinazione implementato sia in un modello Matlab sia in una applicativo compilato a partire da programmazione object oriented (C++) per il controllo del sistema reale. Il modello Matlab, una volta validato, è stato utile sia per simulare traiettorie, configurazioni, algoritmi, che per fornire dati utilizzati dal software di controllo del sistema reale grazie allo sviluppo di un interpolatore. Si sono poi studiati gli algoritmi di inversione cinematica più adatti al sistema real-time in questione e si sono sfruttati affinché non solo i manipolatori si muovessero in modo coordinato, ma che assumessero configurazioni funzionali a task da eseguire in cooperazione.

I principali contributi di innovazione, oltre all'applicazione degli algoritmi, dei formalismi, dei modelli e dei linguaggi di programmazione per la risoluzione dei problemi sullo specifico sistema considerato, hanno riguardato l'utilizzo di recenti algoritmi di inversione cinematica (SNS) a cui si è aggiunta la possibilità di compiere task secondari. In particolare si è utilizzato un task secondario pensato appositamente per il sistema di robot cooperanti preso in considerazione (massima controllabilità sulla direzione che congiunge i due *end effectors*) .

7.2 Sviluppi Futuri

- A partire dai risultati ottenuti, un primo possibile sviluppo può consistere nell'ulteriore miglioramento degli algoritmi includendo un microinterpolatore perfezionato (sulla base di quello realizzato nel Paragrafo 3.2), risolvendo alcuni dei problemi di comunicazione interni al NCS in cui vengono generati e distribuiti i riferimenti di posa/velocità che precludono lo svolgimento di task ad alta velocità sul manipolatore reale, un utilizzo dei coefficienti di scalatura ottenuti con l'SNS anche nel tempo per task non strettamente real-time. Quest'ultimo accorgimento garantirebbe all'algoritmo SNS errori di inseguimento ottimali anche in caso di stringenti vincoli sulla velocità o escursione di giunto a costo di un maggior tempo di esecuzione del task.

  Con tali miglioramenti è possibile pensare di rilasciare una libreria dedicata ai metodi di inversione completa e versatile così da poter essere usata su varie macchine e in diversi contesti scegliendo il metodo più adatto per ogni situazione in base alle considerazioni svolte nei Capitoli 4 e 5 (ad esempio attivando o meno un anello di controllo cinematico di tipo CLIK solo se il tempo di campionamento o la distanza da percorrere tra due punti sono grandi).

- Un altro sviluppo potrebbe prevedere di integrare nel formalismo di coordinazione non più solo delle catene cinematiche (di ordine zero), ma anche catene che

presentino elementi visco-elastici (di ordine uno). Questo porterebbe ad utilizzare il framework presentato nel Capitolo 2 per descrivere il controllo dei due manipolatori in impedenza a livello oggetto e sfruttare così le possibilità di compliant motion control. Questo implicherebbe la possibilità per i manipolatori di operare in condizioni di interazione dinamica con un oggetto afferrato e manipolato, similmente a quanto effettuato in [11].

Sempre in ottica di un sistema multi-arm per la manipolazione di oggetti si possono estendere le considerazioni sui metodi di inversione cinematica al sistema robot + oggetto manipolato così che non solo un robot si disponga in configurazione ottimale per scambiare le forze lungo una direzione data dall'altro robot, ma che entrambi i robot si dispongano in configurazione ottimale per avere una sistema di forze desiderato sull'oggetto manipolato. Anche questo approccio si può basare su concetti di manipolabilità, come esposto in [1].

- Un'ulteriore applicazione si allaccia allo sviluppo del progetto ACTIVE (FP7 270460) in cui il CNR-ITIA è partner incaricato dello sviluppo e controllo delle piattaforme robotiche per la chirurgia cerebrale. Come già espresso nel Capitolo 1 tale progetto ha molte analogie col caso considerato:

  1) Si tratta di un sistema robotico complesso e con molti gradi di libertà che andranno coordinati.
  2) La testa del paziente dovrà essere sostenuta dal supporto mobile che ne assecondi le vibrazioni (accidentali o indotte da stimolazione corticale) con i manipolatori che dovranno compensare tale moto del supporto.
  3) Vi saranno sistemi di telecamere e modelli per caratterizzare i moti di pulsazione celebrale, la sua deformazione a contatto con il tool, la pressione sanguigna (*brain shift*)
  4) L'inserimento dei tools è eseguita mediante interfaccia aptica utilizzata dal chirurgo, a partire dal posizionamento corretto dei manipolatori che stanno

eseguendo la compensazione del moto. La compensazione del moto permetterà al chirurgo di vedere il sistema come statico.

5) Il processo di inserzione può trarre giovamento da una corretta configurazione del robot, ottenuta attraverso un uso appropriato della gestione della ridondanza. Per ulteriori approfondimenti si veda [29]

Figura 7.2: Scenario ACIVE

Bibliografia

[1] S. Chiaverini, P. Chiacchio, L. Sciavicco e B. Siciliano, «Global Task Space Manipulability Ellipsoids for Multiple-Arm Systems,» *IEEE Transactions on robotics and automation,* pp. 678 - 685, 1991.

[2] X. Yun e V. Kumar, «An Approach to Simultaneous Control of Trajectory and Interaction Forces in Dual-Arm Configurations,» *IEEE Transactions on robotics and automation,* pp. 618-625, 1991.

[3] F. Caccavale, P. Chiacchio e S. Chiaverini, «Task-space regulation of cooperative manipulators,» *Automatica,* pp. 879- 886, 2000.

[4] T. Kroger, B. Finkemeyer e F. Wahl, «A Task Frame Formalism for Practical Implementations,» in *International Conference of Robotics & Automation,* New Orleans, 2004.

[5] F. Flacco e A. De Luca, «Motion Control of Redundant Robots under Joint Constraints: Saturation in the Null Space,» in *IEEE International Conference on Robotics and Automation*, RiverCentre, Saint Paul, Minnesota, USA, 2012.

[6] R. Bischoff e A. Albu-Schäffer, «The KUKA-DLR Lightweight Robot arm – a new reference platform for robotics research and manufacturing,» *ISR/ROBOTIK,* pp. 741-748, 2010.

[7] K. Roboter, *Lightweight Robot 4+, Operating Instructions.*

[8] M. Spong, S. Hutchinson e M. Vidyasagar, Robot Modeling and Control, John Wiley & Sons, 2005.

[9] K. Roboter, *KR C2 lr controller, Operating Instructions.*

[10] G. Schreiber, A. Stemmer e R. Bischoff, «Icra 2010 workshop on innovative robot control architectures for demanding applications,» in *In Innovative Robot Control Architectures for Demanding Applications*, 2010.

[11] T. Hirzinger e C. Wimbock, «Impedance Behaviour for Two-handed Manipulation: Design and Experiments,» in *2007 IEEE International Conference on Robotics and Automation*, Roma, 2007.

[12] J. Angeles, Fundamentals of Robotic Mechanical Systems: Theory, Methods and Algorithms, Springer-Verlag, 1997.

[13] L. Sciavicco e B. Siciliano, Robotica industriale: modellistica e controllo di robot manipolatori, McGraw Hill, 2000.

[14] A. De Luca, *Lecture Slides.*

[15] J. Wang, Y. Li e X. Zhao, «Inverse Kinematics and Control of a 7-DOF Redundant Manipulator Based on the Closed-Loop Algorithm,» *International Journal of Advanced Robotic Systems,* vol. 7, n. 4, 2010.

[16] L. Sciavicco e B. Siciliano, «A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators,» *IEEE JOURNAL OF ROBOTICS AND AUTOMATION,* vol. 4, n. 4, pp. 403-410, 1988.

[17] B. Siciliano, *Closed-Loop Inverse Kinematics Algorithms for Redundant Spacecraft /Manipulat or Systems,* ieee, 1993.

[18] P. Chiacchio, S. Chiaverini, L. Sciavicco e B. Siciliano, «Closed Loop Inverse Kinematics for Constrained Redundant Manipulators with Task Space Augmentation,» *The International Journal of Robotics Research,* vol. 10, n. 4, pp. 410-426, 1991.

[19] B. Siciliano e J. Slotine, «A general framework for managing multiple tasks in highly redundant robotic systems,» in *Fifth International Conference on Advanced Robotics: 'Robots in Unstructured Environments'*, 1991.

[20] S. Chiaverini, «Singularity-Robust Task-Priority Redundancy Resolution for Real-Time Kinematic Control of Robot Manipulators,» *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION,* vol. 13, n. 3, pp. 398-410, 1997.

[21] f. Caccavale e B. Siciliano, «Quaternion-Based Kinematic Control of Redundant Spacecraft/Manipulator Systems,» in *International Conference on Robotics & Automation*, Seoul, 2001.

[22] F. Caccavale, S. Chiaverini e B. Siciliano, «Second-Order Kinematic Control of Robot Manipulators with Jacobian Damped Least-Squares Inverse: Theory and Experiments,» *IEEE/ASME TRANSACTIONS ON MECHATRONICS*, vol. 2, n. 3, pp. 188-194, 1997.

[23] M. Shimizu, H. Kakuya e Y. Woo-Keun, «Analytical Inverse Kinematic Computation for 7-DOF Redundant Manipulators With Joint Limits and Its Application to Redundancy Resolution,» *IEEE TRANSACTIONS ON ROBOTICS*, vol. 24, n. 5, pp. 1131-1142, 2008.

[24] D. Omrcen, L. Zlajpah e B. Nemec, «Compensation of velocity and/or acceleration joint saturation applied to redundant manipulator,» *Robotics and Autonomous Systems*, vol. 55, p. 337–344, 2006.

[25] B. Lacevic e P. Rocco, «Safety-Oriented Control of Robotic Manipulators - a Kinematic Approach,» in *18th IFAC World Congress*, Milano, 2011.

[26] S. Chiu, «Control of Redundant Manipulators for Task Compatibility,» in *IEEE International Conference on Robotics and Automation*, 1987.

[27] G. Legnani, Robotica Industriale, Hoepli, 2003.

[28] R. Gourdeau, ROBOOP, A Robotics Object Oriented Package in C++ Documentation, 2007.

[29] «www.active-fp7.eu,» [Online].

[30] J. Soulié, C++ Language Tutorial, cplusplus.com, 2007.

[31] S. Piccardi, GaPiL - Guida alla Programmazione in Linux, 2010.

[32] K. Roboter, *KUKA Fast Research Interface For KUKA System Software 5.6 lr.*

[33] T. Yoshikawa, «Dynamic manipulability of robot manipulators,» in *IEEE International Conference on Robotics and Automation.* , Kyoto, 1985.

Appendice A: Quaternioni

Si richiama qui la definizione di quaternione come riportata in [27]. Si tratta di un insieme di coordinate costituito da quattro numeri a, b, c, d ottenuti dai coseni direttori U_x, U_y , U_z dell'asse di rotazione e dell'angolo α di rotazione attorno a tale asse mediante le seguenti relazioni matematiche:

$$a = U_x \sin\frac{\alpha}{2},\quad b = U_y \sin\frac{\alpha}{2},\quad c = U_z \sin\frac{\alpha}{2},\quad d = \cos\frac{\alpha}{2}$$

La rappresentazione delle rotazioni attraverso i quaternioni si ha con:

$$q = ai + bj + ck + d$$

I quaternioni sono quindi come dei numeri "iper complessi" che rappresentano rotazioni e che di conseguenza possono essere usati per definire matrici di rotazione secondo la relazione:

$$\boldsymbol{R} = \begin{bmatrix} 2(d^2+a^2)-1 & 2(ab-dc) & 2(ac+db) \\ 2(ab+dc) & 2(d^2+b^2)-1 & 2(bc-da) \\ 2(ac-db) & 2(bc+da) & 2(d^2+c^2)-1 \end{bmatrix} = \begin{bmatrix} X_X & X_Y & X_Z \\ Y_X & Y_Y & Y_Z \\ Z_X & Z_Y & Z_Z \end{bmatrix}$$

Per inverso, nota la matrice di rotazione **R**, si p i termini del quaternione come:

$$d = \frac{1}{2}\sqrt{t-1}\,;\quad t = X_X + Y_Y + Z_Z;$$
$$a = \frac{(Y_Z - Z_Y)}{4d};\quad b = \frac{Z_X - X_Z}{4d};\quad c = \frac{(X_Y - Y_X)}{4d};$$

Qualora si abbia $d = 0$ le relazioni diventano:

$$a = \pm\sqrt{\frac{X_X+1}{2}},\quad b = \pm\sqrt{\frac{Y_Y+1}{2}},\quad c = \pm\sqrt{\frac{Z_Z+1}{2}},\quad \begin{cases} 4bc = Y_Z + Z_Y \\ 4ac = Z_X + X_Z \\ 4ab = Y_X + X_Y \\ a^2 + b^2 + c^2 + d^2 = 1 \end{cases}$$

Appendice B: Output di calcolo Algoritmo SNS

Per verificare il funzionamento dell'algoritmo SNS si riporta l'output di tale algoritmo in corrispondenza di un passo temporale in cui dapprima un limite di velocità di giunto risulta superato, per poi venire saturato con conseguente sfruttamento della ridondanza per annullare l'errore corrispondente a tale saturazione. Se fossero rimasti altri giunti in saturazione si sarebbe applicato il coefficiente di scalatura comunque calcolato e pari a 0.9637.

inizio sns per l'istante considerato

Qp_min = -0.8727 -0.8727 -0.8727 -0.8727 -0.8727 -0.8727 -0.8727

Qp_max = 0.8727 0.8727 0.8727 0.8727 0.8727 0.8727 0.8727

qp_SNS = -0.5161 0.3334 -0.2075 0.7177 -0.7895 0.3482 0.9055

err = 0 0 0 0 0 0 0.0328

a = -0.5161 0.3334 -0.2075 0.7177 -0.7895 .3482 0.9055

b = 0 0 0 0 0 0 0

Smax = 1.6910 2.6176 4.2054 1.2160 1.1053 2.5065 0.9637

Smin = -1.6910 -2.6176 -4.2054 -1.2160 -1.1053 -2.5065 -0.9637

smax = 0.9637

smin = -0.9637

task_scaling_factor = 0.9637

SEL =

```
   1   0   0   0   0   0   0
   0   1   0   0   0   0   0
   0   0   1   0   0   0   0
   0   0   0   1   0   0   0
   0   0   0   0   1   0   0
   0   0   0   0   0   1   0
   0   0   0   0   0   0   0
```

qp_n = 0 0 0 0 0 0 0.8727

JA_x_SEL =

```
   0.4400  -0.0646   0.3718  -0.0853  -0.0381   0.0600   0
   0.2608   0.1842   0.2328   0.1689   0.0000   0.0000   0
        0   0.5015   0.0278  -0.4006  -0.0451  -0.0506   0
   0.0000  -0.9436   0.0917   0.7831   0.5748   0.6449   0
   1.0000   0.0000   0.9609   0.0925  -0.4850   0.7643   0
  -0.0000   0.3311   0.2612  -0.6150   0.6590  -0.0000   0
```

rango = 6

qp_SNS_seconda iterazione = -0.6584 0.3194 -0.0369 0.7177 -0.8003 0.3131 0.8727

err = 0 0 0 0 0 0 0

finito ciclo per il task principale