Elaborato di Fondamenti di Robotica

Ciro Arena Vito Giura

4 luglio 2021

Indice

Capitolo 1

Analisi di manipolabilità

1.1 Introduzione

Il robot SCARA è un manipolatore di semplice struttura che fu realizzato in Giappone a fine anni 70'. SCARA è un acronimo che sta per *Selective Compliance Assembly Robot Arm*, che letteralmente significa "braccio manipolatore per l'assemblaggio a cedevolezza selettiva". Tale manipolatore è pensato per la manipolazione di oggetti su un piano orizzontale, perché abbiamo una struttura braccio-avambraccio che si muove nel piano orizzontale con i suoi giunti rotoidali, e poi c'è un giunto prismatico per la traslazione verticale (cioè sopra e sotto). Tale robot per come è progettato è una struttura che ha un'elevata rigidezza a carichi verticali e per ovvi motivi è caratterizzata da una cedevolezza per carichi orizzontali. É di piccole dimensione e per questo è atto solo alla manipolazione di oggetti piccoli su un piano orizzontale. Infatti il robot SCARA è comunemente impiegato per l'assemblaggio di schede di circuiti integrati, perciò c'è necessità di muoversi rapidamente lungo il piano orizzontale e di favorire il montaggio lungo il piano verticale. Quindi ci rendiamo conto che è una struttura con un design progettato ad hoc, proprio per questo tipo di compiti. A tal proposito questo robot è attuato da azionamenti elettrici, proprio per consentire i movimenti rapidi nel piano orizzontale e ha come svantaggio che la precisione del polso si riduce al crescere della distanza del polso stesso dall'asse del primo giunto.

Lo scopo di questo elaborato risolvere tutta una serie di problematiche relativamente a tale tipologia di robot a partire ovviamente dalla cinematica diretta, e faremo tutto ciò utilizzando il Robotics Toolbox di Peter Corke, con cui abbiamo fatto un modello dello SCARA e che è raffigurato in figura 1.1.

L'obiettivo del seguente capitolo è analizzare la manipolabilità in velocità e forza per la struttura portante, rappresentando i relativi ellissoidi per un numero significativo di posizioni dell'organo terminale all'interno dello spazio di lavoro. Prima

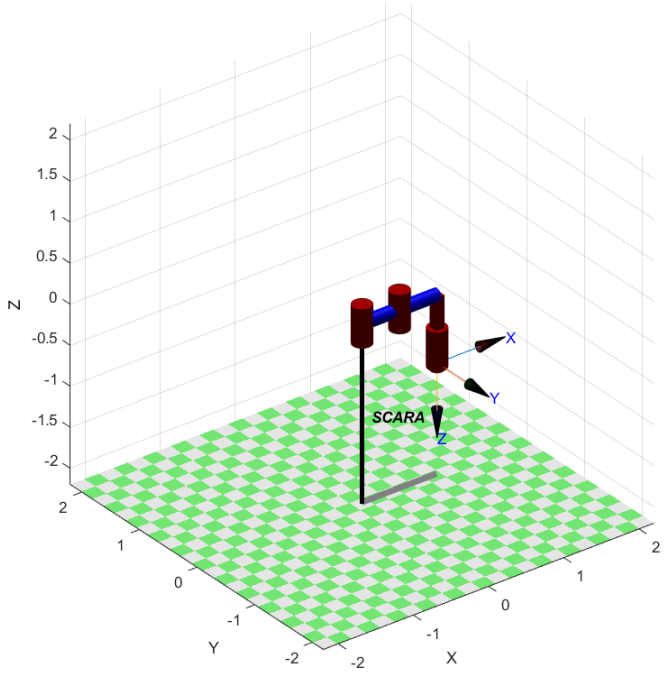

Figura 1.1: Robot SCARA

però bisogna effettuare tutta una serie si step preliminari che portino al calcolo dello Jacobiano geometrico e alle relative singolarità cinematiche.

1.2 Cinematica diretta

Per cominciare andiamo ad applicare la convenzione di Denavit-Hartenberg per ricavare le matrici di trasformazione omogenea tra 2 bracci consecutivi, e quindi l'equazione di cinematica diretta del robot SCARA:

1. Si numerino i giunti e si fissino gli assi $\boldsymbol{z}$ delle terne solidali ai bracci, in corrispondenza degli assi dei giunti rotoidali e lungo la direzione di traslazione del giunto prismatico, in particolare per i giunti rotoidali gli assi $\boldsymbol{z}$ li sceglieremo diretti verso l'alto per i giunti 1 e 2, dato che le rotazioni vengono considerate positive se fatte in senso antiorario, mentre per il giunto prismatico possiamo sceglierlo arbitrariamente. In tal caso dato che tale giunto deve traslare verso il basso, come è usuale per un robot SCARA, alla fine per esso l'asse $\boldsymbol{z}_2$ lo sceglieremo diretto invece verso il basso. I restanti assi $\boldsymbol{z}_3$ e $\boldsymbol{z}_4$ li sceglieremo in verso concorde a $\boldsymbol{z}_2$ per fare in modo che ci siano più parametri α nulli;

2. Dopodiché per la terna 0 si fissi l'origine O_0 in corrispondenza dell'intersezione tra l'asse $\boldsymbol{z}_0$ e la direzione coincidente con il braccio 1 in modo tale che questa terna sia localizzata ad un'altezza $d_0 = 1$ m dal piano di lavoro. Poi si scelga

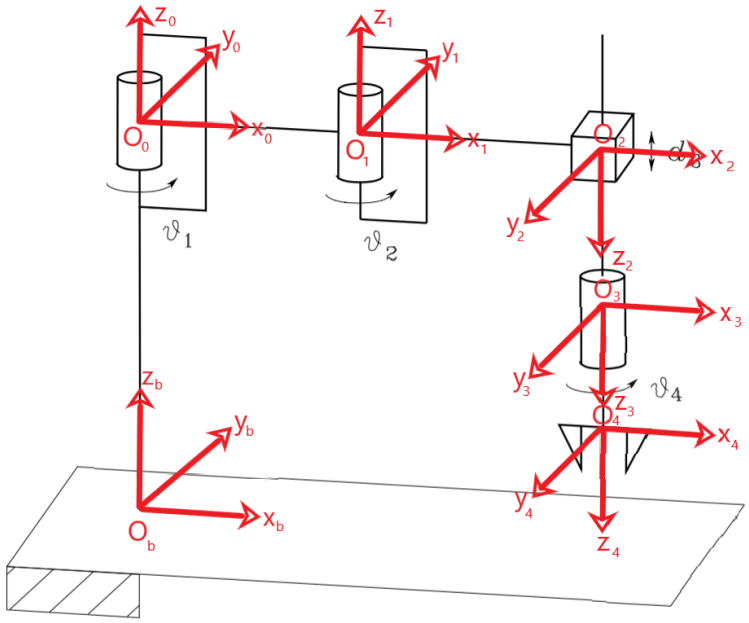

Figura 1.2: Convenzione D-H del robot SCARA

l'asse $\boldsymbol{x}_0$ in maniera tale che sia parallelo ad uno dei lati del piano di lavoro rettangolare con verso che va dal giunto 1 al giunto 2;

3. Ora poiché i primi due giunti rotoidali, così come il terzo prismatico, hanno assi $\boldsymbol{z}$ paralleli tra di loro, allora le normali comuni a 2 assi $\boldsymbol{z}$ consecutivi sono infinite e quindi noi le sceglieremo in maniera tale che i parametri d_i si annullino, cioè sceglieremo gli assi $\boldsymbol{x}_1, \boldsymbol{x}_2$ e $\boldsymbol{x}_3$ paralleli e concordi $\boldsymbol{x}_0$

4. gli assi $\boldsymbol{y}_i$ li sceglieremo infine in modo tale che ovviamente siano ortogonali agli assi $\boldsymbol{x}_i$ e $\boldsymbol{z}_i$ e che formino terne levogire

5. Infine fissiamo la terna 4 solidale all'organo terminale, con asse $\boldsymbol{z}_4$ coincidente a $\boldsymbol{z}_3$ ma nel verso opposto, in modo tale che il suo versore $\boldsymbol{a}^e$ punti nella direzione di approccio, e con origine O_4 a distanza d_4 dall'origine della terna precedente. Dato che gli assi $\boldsymbol{z}_3$ e $\boldsymbol{z}_4$ sono coincidenti, non esisterà una normale comune per cui si potrà scegliere arbitrariamente l'asse $\boldsymbol{x}_4$ e lo faremo mettendolo parallelo ai precedenti assi $\boldsymbol{x}_i$.

Quello che otterremo è raffigurato in figura 1.2.

Adesso provvediamo a calcolare i parametri di Denavit-Hartenberg:

- I parametri θ_i sono tutti variabili ad eccezione di θ_3 relativo al giunto prismatico che ovviamente sarà costante e nello specifico pari a 0 dato che gli assi $\boldsymbol{x}_1$ e $\boldsymbol{x}_2$ sono paralleli e concordi;

- i parametri d_1 e d_2 sono ovviamente nulli per la scelta che è stata fatta delle normali comuni che intersecano le origini delle terne solidali ai primi due giunti

rotoidali e al terzo giunto prismatico, d_3 ovviamente è variabile in quanto c'è un giunto prismatico, e d_4 è un parametro costante non specificato per la scelta che è stata fatta della terna 4 solidale all'organo terminale, e che noi supporremo d'ora in avanti essere pari a 0.2;

- i parametri α_i sono tutti nulli essendo gli assi $\boldsymbol{z}$ paralleli e concordi tra di loro, ad eccezione di α_2 che sarà pari a π poiché siamo andati a fissare tale terna con asse $\boldsymbol{z}_2$ diretto in verso opposto a quello precedente $\boldsymbol{z}_3$;

- per quanto riguarda i parametri a_i abbiamo che tra le terne 0 e 1, e le terne 1 e 2 le loro origini sono distanti rispettivamente a_1 e a_2 i cui valori sono assegnati a sono pari a $a_0 = a_1 = 0.5\ m$, mentre a_3 e a_4 sono nulli poiché le terne 2, 3 e 4 hanno assi z coincidenti e non c'è una normale comune definita

Tabella 1.1: parametri di D-H

Braccio	a_i	α_i	d_i	θ_i
1	0.5	0	0	θ_1
2	0.5	π	0	θ_2
3	0	0	d_3	0
4	0	0	d_4	θ_4

A questo punto ci possiamo calcolare le matrici di trasformazione omogenea $\boldsymbol{A}_i^{i-1}$ per $i = 1, 2, 3, 4$ con la seguente formula

$$\boldsymbol{A}_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ed essendo in tal caso che $\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \\ \theta_4 \end{bmatrix}$ avremo allora

$$\boldsymbol{A}_1^0(\theta_1) = \begin{bmatrix} c_1 & -s_1 & 0 & \frac{1}{2}c_1 \\ s_1 & c_1 & 0 & \frac{1}{2}s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{A}_2^1(\theta_2) = \begin{bmatrix} c_2 & s_2 & 0 & \frac{1}{2}c_2 \\ s_2 & -c_2 & 0 & \frac{1}{2}s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{A}_3^2(d_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{A}_4^3(\theta_4) = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pertanto in definitiva ci possiamo calcolare la matrice di trasformazione omogenea $\boldsymbol{A}_0^4(\boldsymbol{q})$ che restituisce la posa della terna 4 rispetto alla terna 0 come composizione delle matrici di trasformazione omogenea precedentemente calcolate, cioè

$$\boldsymbol{A}_0^4(q) = \boldsymbol{A}_1^0(\theta_1)\boldsymbol{A}_2^1(\theta_2)\boldsymbol{A}_3^2(d_3)\boldsymbol{A}_4^3(\theta_4) =$$

$$\begin{bmatrix} c_{124} & s_{124} & 0 & a_2c_{12} + a_1c_1 \\ s_{124} & -c_{124} & 0 & a_2s_{12} + a_1s_1 \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{124} & s_{124} & 0 & \frac{1}{2}(c_{12} + c_1) \\ s_{124} & -c_{124} & 0 & \frac{1}{2}(s_{12} + s_1) \\ 0 & 0 & -1 & -d_3 - 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

dove abbiamo usato le seguenti notazioni

$$c_{124} = \cos(\theta_1 + \theta_2 - \theta_4) \qquad s_{124} = \sin(\theta_1 + \theta_2 - \theta_4)$$

In conclusione fissiamo un terna base sul piano di lavoro con asse $\boldsymbol{z}_b$ coincidente con l'asse $\boldsymbol{z}_0$, con origine O_b localizzata nel punto in cui è vincolata al pavimento l'estremità del manipolatore, e con lo stesso orientamento della terna 0. Di conseguenza per allineare la terna base alla terna 0 basterà semplicemente traslare la prima verso l'alto lungo z di una quantità pari a $d_0 = 1\ m$, per cui ci possiamo scrivere una matrice di trasformazione omogenea costante $\boldsymbol{T}_0^b$ come

$$\boldsymbol{T}_0^b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

E dato che già la terna 4 è quella solidale all'organo terminale per cui non c'è bisogno anche di una matrice $\boldsymbol{T}_e^4$ allora ci possiamo calcolare l'equazione di cinematica diretta $\boldsymbol{T}_e^b$ che esprime la posa dell'end-effector rispetto alla terna di base come

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \boldsymbol{T}_0^b\boldsymbol{T}_4^0(\boldsymbol{q}) = \begin{bmatrix} c_{124} & s_{124} & 0 & a_2c_{12} + a_1c_1 \\ s_{124} & -c_{124} & 0 & a_2s_{12} + a_1s_1 \\ 0 & 0 & -1 & 1 - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{124} & s_{124} & 0 & \frac{1}{2}(c_{12} + c_1) \\ s_{124} & -c_{124} & 0 & \frac{1}{2}(s_{12} + s_1) \\ 0 & 0 & -1 & \frac{4}{5} - d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

1.3 Cinematica differenziale

1.3.1 Jacobiano geometrico

Adesso andiamo a calcolarci lo Jacobiano geometrico che, essendoci 4 giunti, sarà una matrice di dimensione 6×4. Nello specifico, essendo i giunti 1, 2 e 4 dei giunti rotoidali, e il giunto 3 un giunto prismatico, tale Jacobiano avrà la seguente struttura

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_p \\ \boldsymbol{J}_o \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_0 \times (\boldsymbol{p}_4 - \boldsymbol{p}_0) & \boldsymbol{z}_1 \times (\boldsymbol{p}_4 - \boldsymbol{p}_1) & \boldsymbol{z}_2 & \boldsymbol{z}_3 \times (\boldsymbol{p}_4 - \boldsymbol{p}_3) \\ \boldsymbol{z}_0 & \boldsymbol{z}_1 & 0 & \boldsymbol{z}_3 \end{bmatrix} \tag{1.3}$$

Pertanto in definitiva il nostro Jacobiano geometrico sarà

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} -a_1 s_1 - a_2 s_{12} & -a_2 s_{12} & 0 & 0 \\ a_1 c_1 + a_2 c_{12} & a_2 c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}(s_1 + s_{12}) & -\frac{1}{2} s_{12} & 0 & 0 \\ \frac{1}{2}(c_1 + c_{12}) & \frac{1}{2} c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} \tag{1.4}$$

ossia una matrice 4×4 poiché si possono rimuovere le righe nulle. In generale l'i-esima riga nulla della matrice Jacobiana implica, guardando la relazione della cinematica differenziale, che le variabili di giunto non contribuiscono in alcun modo all'i-esima componente della velocità dell'end-effector, cioè queste righe nulle moltiplicate per $\boldsymbol{q}$ restituiscono un risultato pari a zero. In questo caso si hanno la quarta e la quinta riga nulle, le quali contribuiscono al calcolo delle componenti della velocità angolare dell'end-effector intorno all'asse $\boldsymbol{x}$ e intorno all'asse $\boldsymbol{y}$ rispettivamente, e questo è dovuto al fatto appunto che il polso sferico nel nostro caso è costituito da un solo giunto rotoidale che provoca in generale una rotazione intorno all'asse $\boldsymbol{z}$. Infine, si tenga conto del fatto che tutti i gradi di libertà del nostro manipolatore, causano rotazioni dell'end-effector intorno lo stesso asse nello spazio, per cui possiamo anche concludere che lo Jacobiano geometrico coincide con quello analitico.

$$\boldsymbol{J} = \boldsymbol{J_A} \tag{1.5}$$

1.3.2 Singolarità cinematiche

Una volta calcolata la matrice Jacobiana, possiamo procedere all'individuazione delle singolarità cinematiche calcolando il determinante della matrice (1.4) prece-

dentemente ricavata, e vedendo in quali casi esso è nullo:

$$det(\boldsymbol{J}(\boldsymbol{q})) = a_1 a_2 s_2 = \frac{s_2}{4} = 0 \Longrightarrow \theta_2 = 0,\ \theta_2 = \pi \tag{1.6}$$

Scopriamo quindi che che le singolarità cinematiche in questo caso sono singolarità che si presentano ai *confini* dello spazio di lavoro, perché per $\theta_2 = 0$ il manipolatore è completamente disteso, mentre per $\theta_2 = \pi$ è completamente ritratto.

Si osservi che tale determinante è massimo invece quando il seno è massimo, ossia quando esso è pari a 1. Come sappiamo ciò avviene per $\theta_2 = \frac{\pi}{2}$ e quindi questa rappresenta la configurazione di massima destrezza del nostro manipolatore.

1.4 Ellissoidi di manipolabilità

Adesso possiamo occuparci dello studio della manipolabilità del nostro robot SCARA. Come prima cosa diciamo che è richiesta nello specifico l'analisi della manipolabilità in forza e velocità della *struttura portante*, ciò significa che non dobbiamo considerare per intero lo Jacobiano geometrico calcolato nella (1.4), ma solo quella porzione relativa appunto alla struttura portante, cioè in pratica solo le prime 3 righe

$$\boldsymbol{J}_p = \begin{bmatrix} -\frac{1}{2}(s_1 + s_{12}) & -\frac{1}{2}s_{12} & 0 & 0 \\ \frac{1}{2}(c_1 + c_{12}) & \frac{1}{2}c_{12} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}(s_1 + s_{12}) & -\frac{1}{2}s_{12} & 0 \\ \frac{1}{2}(c_1 + c_{12}) & \frac{1}{2}c_{12} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{1.7}$$

Ora noi sappiamo bene che in generale gli ellissoidi di manipolaibilità, sia in forza che velocità, dipendono dallo Jacobiano, nello specifico gli assi dell'ellissoide sono diretti come gli autovettori della matrice JJ^T, mentre invece le lunghezze dei semiassi sono dati dai valori singolari (o dai loro inversi nel caso degli ellissoidi in forza) dello Jacobiano. Adesso vedendo l'espressione dello Jacobiano nella (1.7) si può facilmente notare come le variabili di giunto che modificano l'indice di manipolabilità del robot siano esclusivamente θ_1 e θ_2, perché solo tali variabili compaiono nella matrice J_p. In particolare noi sappiamo che ad influenzare in maniera vera e propria la manipolabilità è la posizione reciproca dei vari bracci del manipolatore, e non la sua posizione assoluta, posizione reciproca che come possiamo facilmente intuire è data solo da θ_2. Come ulteriore conferma di ciò, andandoci a calcolare anche la misura di manipolabilità possiamo verificare come questa dipenda solo da θ_2

$$w(\boldsymbol{q}) = \sqrt{\det(J_p J_p^T)} = a_1 a_2 \sin\theta_2 = \frac{\sin\theta_2}{4} \tag{1.8}$$

Cioè otteniamo lo stesso risultato di quando abbiamo esaminato le singolarità cinematiche, ossia $\det(J)$, cosa prevedibile dato che la matrice in questione è quadrata.

Per questo motivo le configurazioni dello spazio dei giunti sono state scelte modificando solo ed unicamente la variabile di giunto θ_2, fissando le altre due variabili di giunto, θ_1 e d_3, a zero.

Qui di seguito sono riportati i grafici degli ellissoidi ottenuti col Robotics Toolbox di Peter Corke in cui siamo andati anche a proiettare le ombre sui piani xy, xz e yz per renderci meglio conto delle dimensioni di tali ellissoidi, distinguendo opportunamente gli ellissoidi fi velocità da quelli di forza.

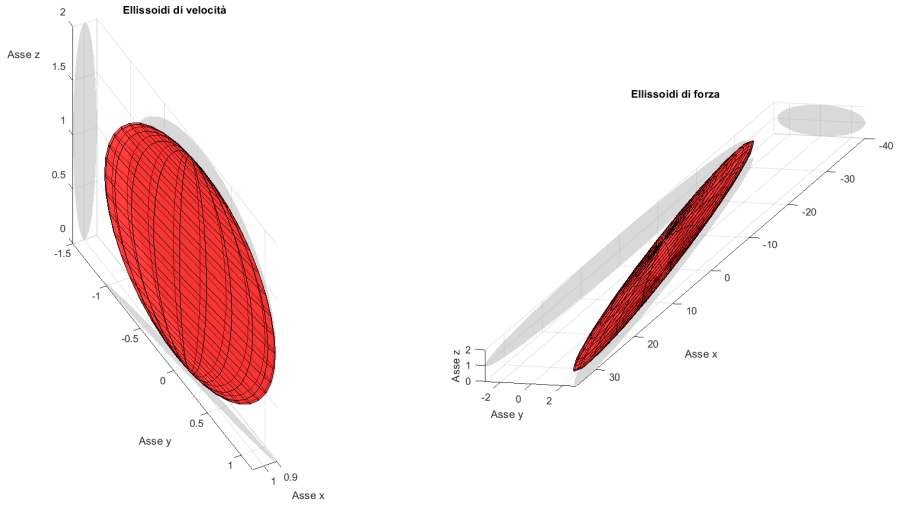

Figura 1.3: Ellissoidi di manipolabilità per $\theta_2 = \frac{\pi}{24} \approx 0.13$

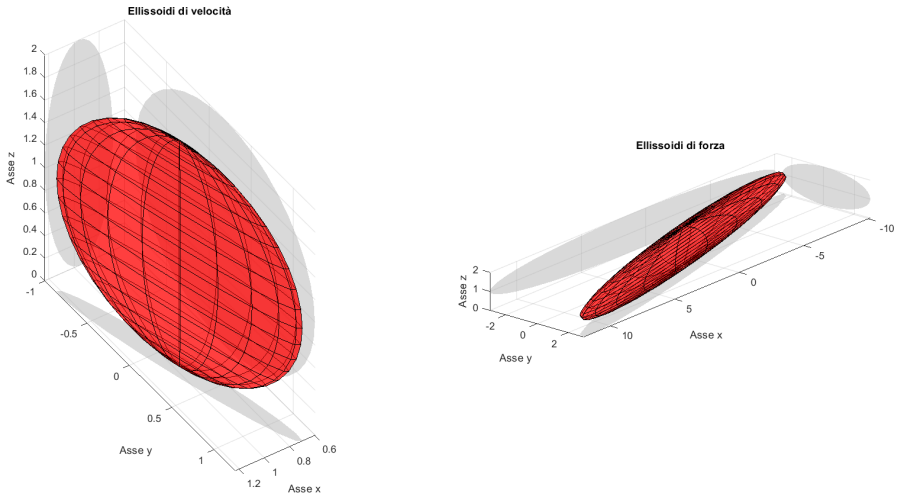

Figura 1.4: Ellissoidi di manipolabilità per $\theta_2 = \frac{\pi}{8} \approx 0.39$

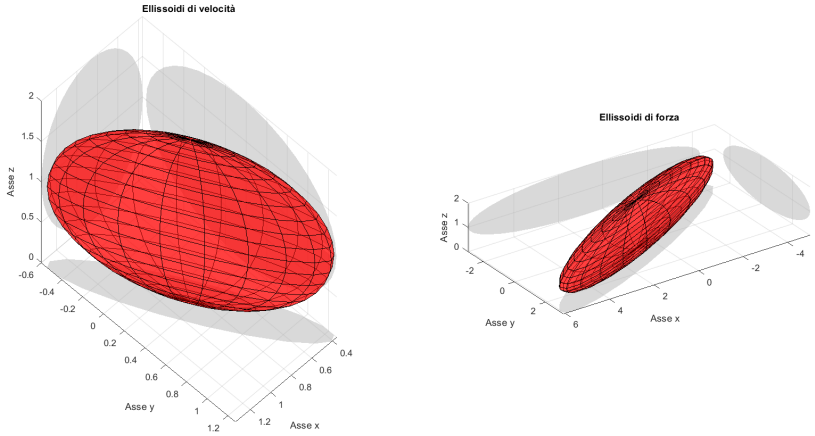

Figura 1.5: Ellissoidi di manipolabilità per $\theta_2 = \frac{\pi}{4} \approx 0.785$

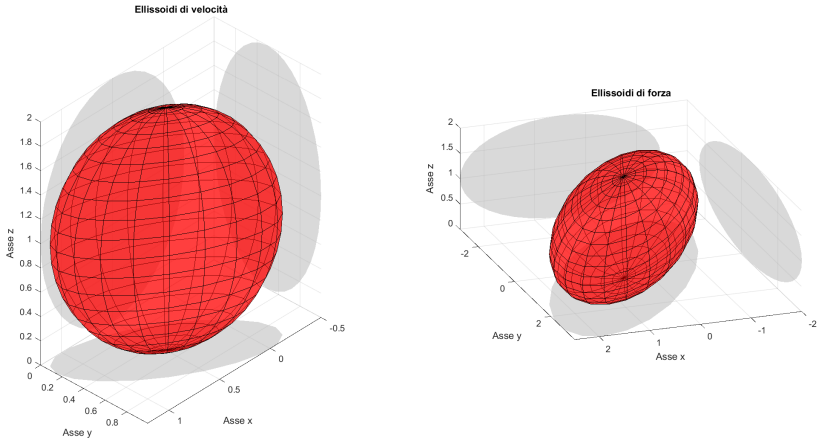

Figura 1.6: Ellissoidi di manipolabilità per $\theta_2 = \frac{\pi}{2} \approx 1.57$

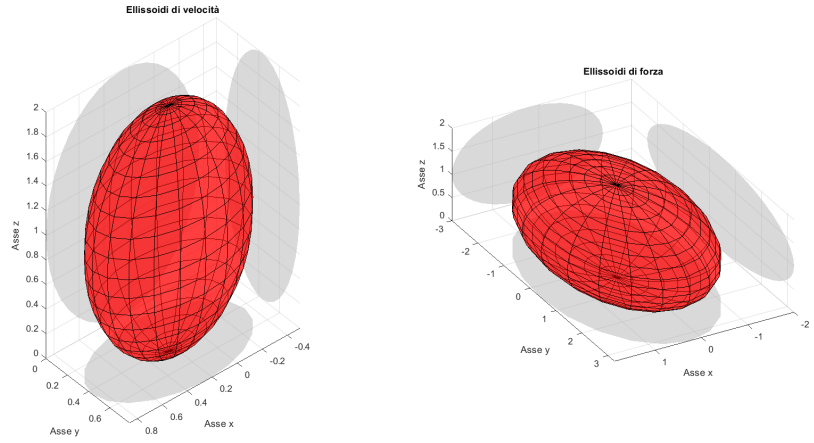

Figura 1.7: Ellissoidi di manipolabilità per $\theta_2 = \frac{2\pi}{3} \approx 2.09$

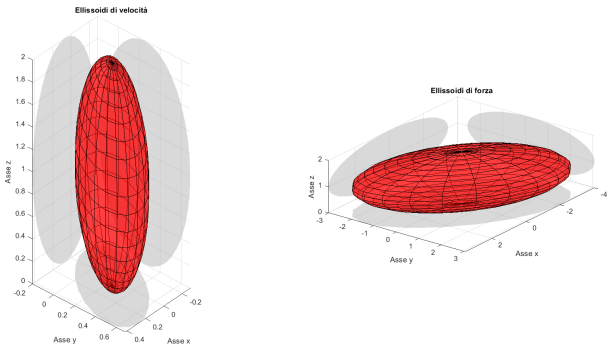

Figura 1.8: Ellissoidi di manipolabilità per $\theta_2 = \frac{5\pi}{6} \approx 2.62$

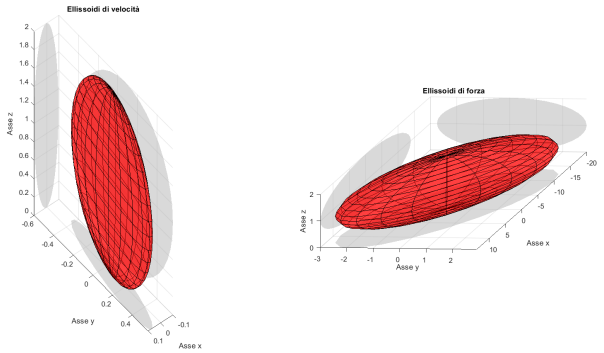

Figura 1.9: Ellissoidi di manipolabilità per $\theta_2 = 3 \approx \pi$

Come possiamo notare dalle immagini appena viste, in prossimità delle singolarità cinematiche, cioè per valori di θ_2 vicini a 0 e π i corrispondenti ellissoidi di manipolabilità che sono tridimensionali, tendono ad essere delle ellissi bidimensionali. Mentre invece man mano che ci si avvicina a $\theta_2 = \frac{\pi}{2}$ i nostri ellissoidi sono sempre più simili ad una sfera, dato che come abbiamo constatato in precedenza questa è la configurazione di massima destrezza. Infine si noti come in ognuno di queste coppie di ellissoidi di velocità e di forza, la direzione lungo cui si hanno velocità ridotte, sono le stesse in cui si può esercitare una grande forza e viceversa, in virtù della dualità cinetico-statica.

Capitolo 2

Traiettoria e inversione cinematica

Dopo aver effettuato l'analisi in manipolabilità del nostro robot SCARA, il prossimo passo, come richiesto da traccia, sarà pianificare una traiettoria lungo un percorso caratterizzato da almeno 10 punti all'interno dello spazio di lavoro, in cui vi siano almeno un tratto rettilineo e uno circolare e inoltre il passaggio per almeno 4 punti di via.
Dopodiché per fare in modo che venga seguita la traiettoria tracciata da parte dell'end-effector bisognerà tradurre queste specifiche di movimento assegnate in termini di variabili dello spazio operativo, nei corrispondenti set di variabili di giunto. Per fare ciò si andranno ad implementare in MATLAB gli algoritmi per l'inversione cinematica con inversa e trasposta dello Jacobiano lungo la traiettoria (per notare la differenza di precisione), andando ad adottare la regola di integrazione numerica di Eulero con tempo di integrazione di 1 ms.
Infine, supponendo di rilassare una componente di spazio operativo, andremo anche ad implementare in MATLAB l'algoritmo per l'inversione cinematica con pseudo-inversa dello Jacobiano lungo la traiettoria, nell'ipotesi di ottimizzare un vincolo di destrezza.

2.1 La traiettoria

La primissima cosa da fare, come già detto, è la generazione della traiettoria da far seguire allo SCARA. La traiettoria prescelta è mostrata in figura 2.1. Come possiamo vedere, essa altro non è che l'immagine di un robot stilizzato, in cui possiamo individuare negli occhi dei percorsi circolari, nella bocca e bordi delle orecchie degli archi di circonferenza, e infine nella sagoma della testa dei percorsi rettilinei e per punti di via. Ci possiamo rendere facilmente conto però che per riprodurre fedelmente il disegno mostrato ci vorrebbe una quantità eccessiva di punti, per cui ci siamo limitati a disegnare solo la sagoma della testa e gli occhi, rispettando così la traccia.

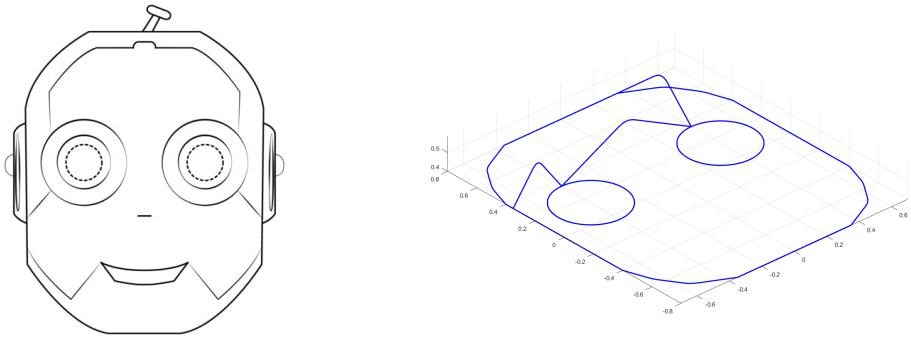

Figura 2.1: Immagine del robot stilizzato a sinistra, e la traiettoria disegnata a destra

Per la generazione di questa traiettoria, anche in tal caso abbiamo deciso di sfuttare il Robotics Toolbox di Peter Corke. In particolare per la generazione dei tratti rettilinei e dei percorsi per punti di via, abbiamo sfruttato determinate funzioni offerte dal toolbox, mentre invece per la generazione di percorsi circolari e archi di circonferenza abbiamo dovuto progettare apposite funzioni, denominate rispettivamente *circonferenza.m* e *arco.m* (si trovano in allegato). In ogni caso, per il tracciamento delle suddette porzioni di traiettoria, l'ascissa curvilinea $s(t)$ che è stata generata è una funzione analitica che segue un profilo di velocità di tipo trapezoidale per la quale è stata impiegata un'altra funzione del Robotics Toolbox.

Infine, in fase di generazione della traiettoria attraverso i punti desiderati, abbiamo deciso di programmare una traiettoria da seguire per l'orientamento nello spazio operativo, attivando così anche il giunto 4 del robot SCARA. Pertanto, a conclusione del paragrafo, andiamo ad illustrare nella seguente figura il profilo temporale desiderato per le variabili dello spazio operativo.

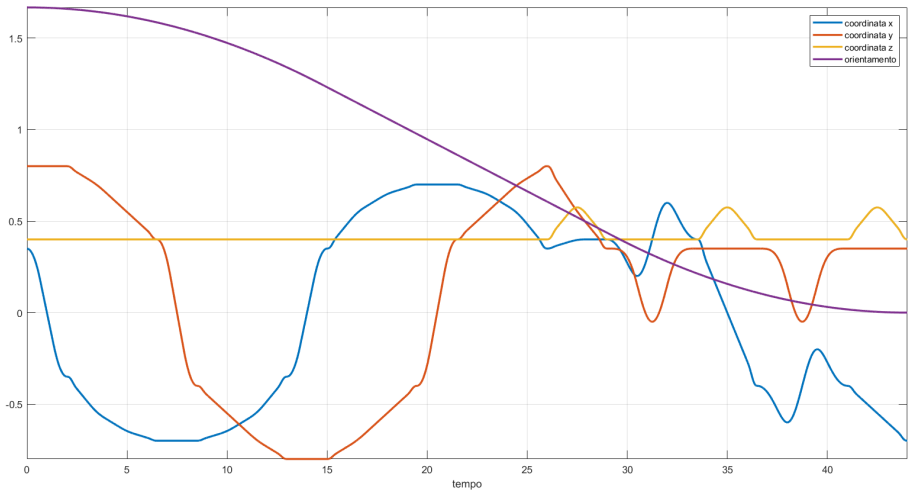

Figura 2.2: L'andamento temporale desiderato per le variabili dello spazio operativo

2.2 Algoritmi CLIK del primo ordine

Gli algoritmi CLIK, *Closed Loop Inverse Kinematic*, ci permettono, dati come riferimento la traiettoria nello spazio dei giunti, di generare le variabili di giunto corrispondenti ai punti della suddetta traiettoria, in modo tale da poter dare tali set di variabili ai giunti del robot facendogli seguire così la traiettoria desiderata.
Gli algoritmi CLIK che andremo qui ad implementare in prima battuta, saranno solo quelli del primo ordine, ossia quelli per cui andremo a dare solo ed unicamente un riferimento in velocità dello spazio operativo, mentre per quanto riguarda quelli del secondo ordine, ne discuteremo nel capitolo della dinamica.
Per ognuno di tali algoritmi CLIK che vedremo, e che sono quelli con inversa dello Jacobiano, con trasposta dello Jacobiano e con Pseudo-inversa dello Jacobiano, otterremo le velocità di giunto $\dot{\boldsymbol{q}}$ dalle quali si potranno ricavare le variabili di giunto $\boldsymbol{q}$ usando la tecnica di integrazione di Eulero.

2.2.1 CLIK con inversa della Jacobiana

Il primo algoritmo CLIK che qui presenteremo è quello che fa uso dell'inversa dello Jacobiano analitico, ossia quello che permette di calcolare le velocità dei giunti come

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_A^{-1}(\boldsymbol{q})(\dot{\boldsymbol{x}}_d + \boldsymbol{K}\boldsymbol{e}) \tag{2.1}$$

dove:

- $\boldsymbol{J}_A^{-1}(\boldsymbol{q})$ è l'inversa dello Jacobiano analitico che calcoliamo attraverso una funzione MATLAB che calcola semplicemente l'inversa della matrice (1.4);
- $\dot{\boldsymbol{x}}_d$ è il riferimento della velocità della traiettoria pianificata nel paragrafo precedente, il cui riferimento in posizione è mostrato in figura 2.2;
- $\boldsymbol{K}$ è la matrice dei pesi ed è una matrice diagonale 4×4 perché 4 è il numero dei giunti, e più sono alti questi pesi e più l'errore del ciclo di retroazione converge rapidamente a zero; noi abbiamo settato così tale matrice

$$\boldsymbol{K} = \begin{bmatrix} 1200 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

  e ci possiamo spingere per questi pesi fino a valori pari a $\frac{2}{T_c}$

Lo schema Simulink progettato dell'algoritmo CLIK ricalca alla perfezione quello visto sul libro e viene riportato nella figura 2.3:

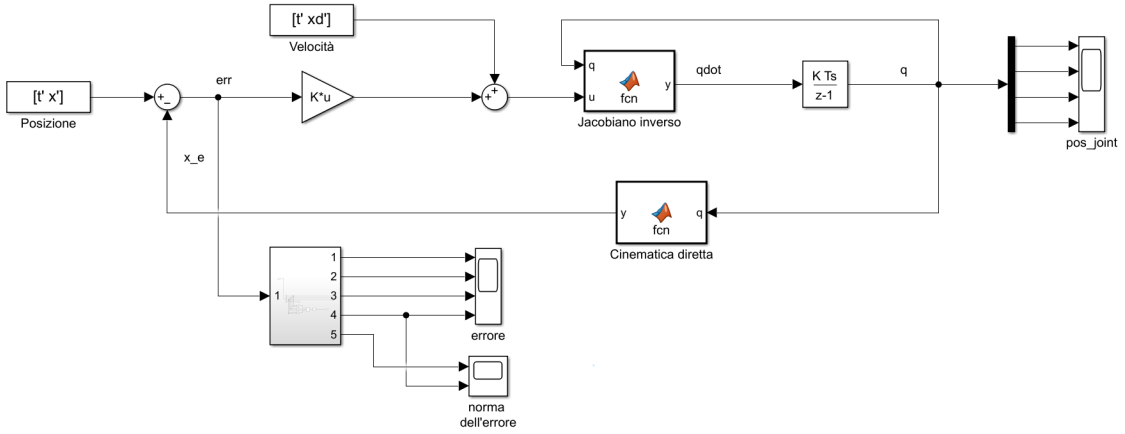

Figura 2.3: Schema Simulink dell'algoritmo CLIK del primo ordine con inversa dello Jacobiano

Come possiamo vedere, la posa effettiva $\boldsymbol{x}_e$ e quindi l'errore $e = \boldsymbol{x}_d - \boldsymbol{x}_e$, viene calcolato attraverso un'altra MATLAB function che calcola la funzione di cinematica diretta $\boldsymbol{k}(\boldsymbol{q})$. Tale funzione come sappiamo è un vettore costituito da un numero minimo di variabili dello spazio operativo che esprimono la posa dell'end-effector, e le sue componenti sono le componenti x, y, z della posizione, e l'orientamento ϕ

$$\boldsymbol{x}_e = \boldsymbol{k}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{p}_e(\boldsymbol{q}) \\ \boldsymbol{\phi}_e(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(c_{12} + c_1) \\ \frac{1}{2}(s_{12} + s_1) \\ \frac{4}{5} - d_3 \\ \theta_1 + \theta_2 - \theta_4 \end{bmatrix} \tag{2.2}$$

dove il vettore posizione è preso dall'ultima colonna della matrice di trasformazione omogenea (1.2), mentre l'orientamento in termini di angoli di Eulero viene calcolata a partire dalla matrice rotazione presente sempre nella stessa matrice di trasformazione omogenea. Nel nostro caso però è abbastanza semplice ricavare l'orientamento corrispondente perché tale matrice di rotazione ha la struttura di una matrice di rotazione elementare intorno all'asse $\boldsymbol{z}$ di un angolo pari a $\theta_1 + \theta_2 - \theta_4$.

Lanciando la simulazione possiamo visualizzare le traiettorie richieste ai giunti in figura 2.4, notando come al termine della traiettoria eseguita, si permanga nell'ultima posa raggiunta.
Dopodiché passiamo anche a graficare in figura l'errore tra la posa desiderata e quella effettivamente conseguita dall'end-effector raffigurando separatamente l'errore sull'orientamento e la *norma* dell'errore di posizione. Si noti che l'ampiezza dell'errore rimane molto piccola e si annulla a regime.

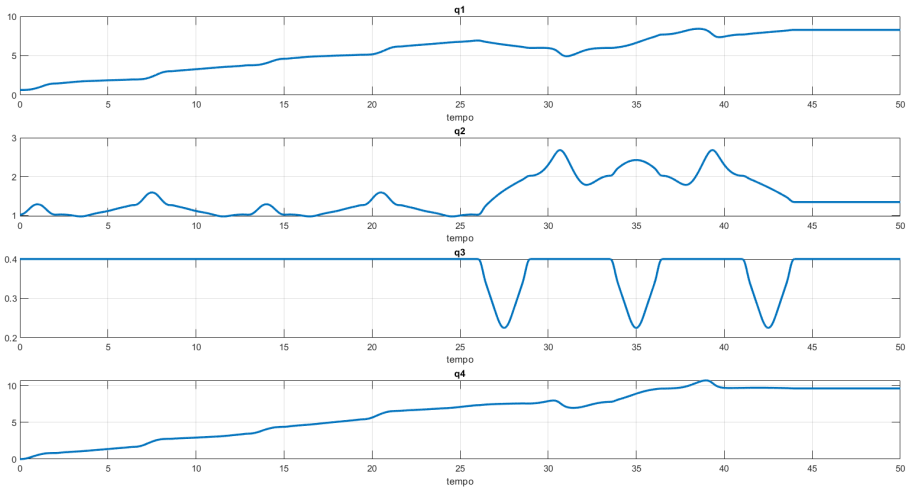

Figura 2.4: Traiettorie richieste ai giunti per l'algoritmo con inversa della Jacobiana

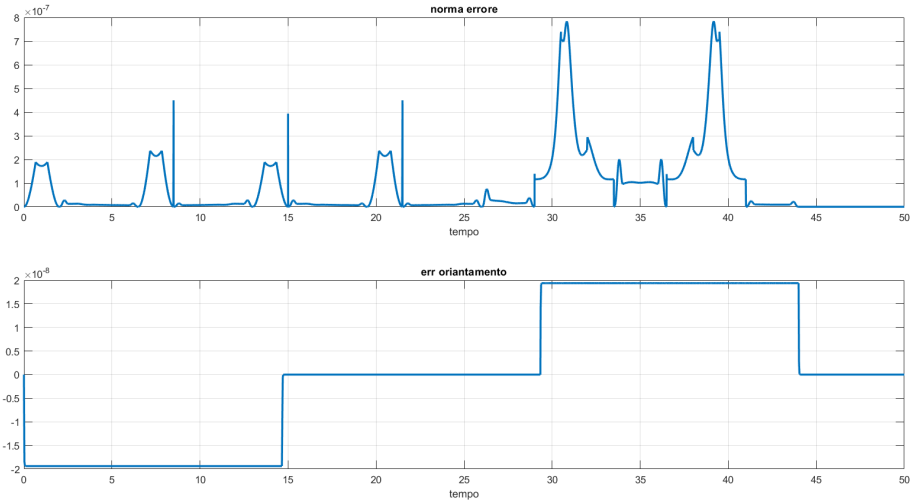

Figura 2.5: Andamento dell'errore tra posa effettiva e desiderata lungo la traiettoria

2.2.2 CLIK con trasposta della Jacobiana

L'algoritmo CLIK con trasposta della Jacobiana, come suggerisce stesso il nome, va a calcolare le velocità di giunto $\dot{\boldsymbol{q}}$ sfruttando non più l'inversa della Jacobiana, ma la sua trasposta, cioè come

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_A^T(\boldsymbol{q})\boldsymbol{K}\boldsymbol{e} \tag{2.3}$$

Cosa comporta ciò? Innanzitutto facciamo presente che stavolta, guardando quest'ultima formula, non ci sarà bisogno di alcun riferimento in velocità a differenza dell'algoritmo CLIK con inversa della Jacobiana. Inoltre noi ricordiamo dal paragra-

fo precedente che l'aver usato l'inversa della Jacobiana ha restituito una dinamica lineare dell'errore, perché usando l'inversa è stato possibile eliminare le non linearità nell'equazione differenziale che governa la dinamica dell'errore, cioè siamo andati ad effettuare di fatto una *feedback linearization*. Usare invece la trasposta, cioè calcolare le velocità di giunto come nella (2.3), chiaramente non linearizza più la dinamica dell'errore, che rimane a questo punto non lineare, ma ci restituisce un algoritmo che è più robusto numericamente perché in corrispondenza di una singolarità cinematica ci possiamo tranquillamente calcolare la trasposta dello Jacobiano analitico, cosa che purtroppo non possiamo fare con la sua inversa. Svantaggio di tutto ciò risiede nell'errore tra la posa desiderata e la posa effettiva, che in questo caso è più ampio lungo tutta la traiettoria seppur rimanga a norma limitata, per cui la nostra traiettoria risulterà essere meno precisa come possiamo vedere dalle simulazioni successive. Alla fine però si tenga conto che nel momento in cui la traiettoria si arresta e la posa desiderata è quindi costante, l'errore va precisamente a zero.

Vediamo adesso i risultati ottenuti partendo col mostrare lo schema Simulink implementato per tale algoritmo

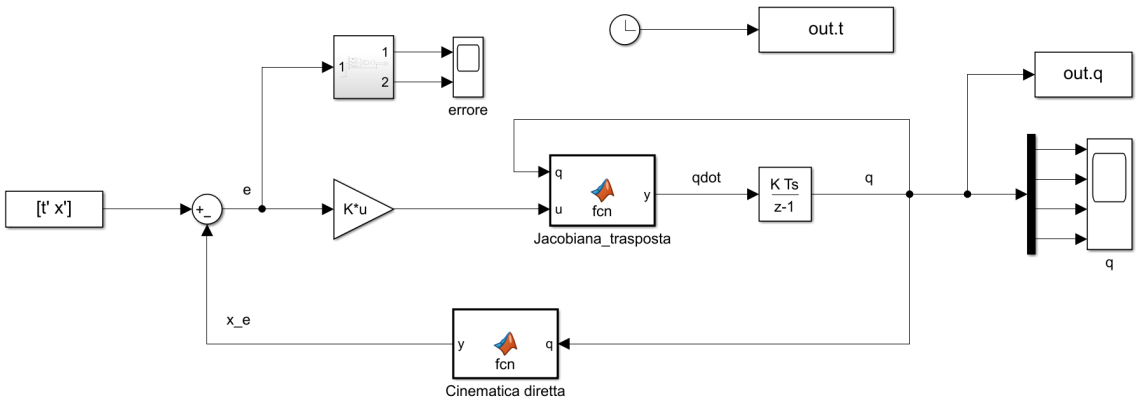

Figura 2.6: Schema Simulink dell'algoritmo CLIK con trasposta dello Jacobiano

Dove sono stati scelti i seguenti pesi per la matrice $\boldsymbol{K}$

$$\boldsymbol{K} = \begin{bmatrix} 500 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 \\ 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 500 \end{bmatrix}$$

Si noti che stavolta i pesi scelti sono diversi da prima perché il limite superiore oltre il quale non si può eccedere è cambiato, dato che è cambiata la dinamica dell'errore. Eseguendo il codice MATLAB contenuto nel file *traiettoria.m* otterremo il seguente andamento per le variabili di giunto con relativo errore

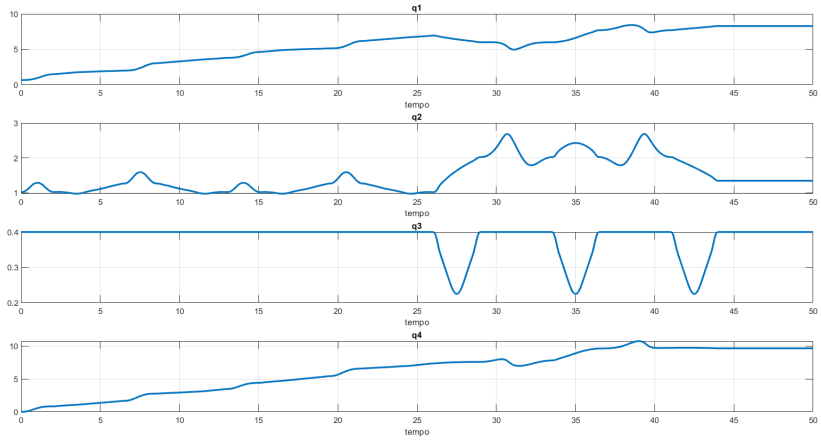

Figura 2.7: Traiettorie richieste ai giunti per l'algoritmo con trasposta della Jacobiana

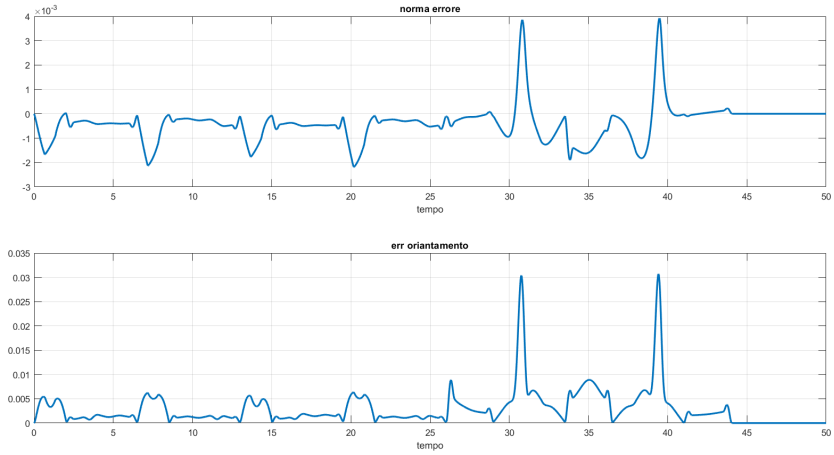

Figura 2.8: Andamento dell'errore tra posa effettiva e desiderata lungo la traiettoria

2.2.3 CLIK con pseudo-inversa della Jacobiana

L'algoritmo CLIK con pseudo-inversa dello Jacobiano viene impiegato nel momento in cui si ha a che fare con un manipolatore ridondante, ossia con un manipolatore tale che il suo Jacobiano sia una matrice, non più quadrata, ma rettangolare di dimensione $r \times n$ a rango pieno, dove $r < n$ è il numero di variabili dello spazio operativo da specificare per un determinato task. Tale tipo di algoritmo CLIK alla fine è solo una versione generalizzata di quello con inversa dello Jacobiano visto nel paragrafo 2.2.1, dove in luogo dell'inversa va considerata la pseudo-inversa destra di $\boldsymbol{J}_A$, e dove stavolta è possibile generare moti interni alla struttura per massimizzare un certo vincolo di destrezza sfruttando i gradi di libertà ridondanti. Infatti adesso

le velocità di giunto le andremo a calcolare come

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_A^{\dagger}(\boldsymbol{q})(\dot{\boldsymbol{x}}_d + \boldsymbol{K}\boldsymbol{e}) + (\boldsymbol{I}_n + \boldsymbol{J}_A^{\dagger}(\boldsymbol{q})\boldsymbol{J}_A(\boldsymbol{q}))\dot{\boldsymbol{q}}_0 \tag{2.4}$$

dove questo $\dot{\boldsymbol{q}}_0$ viene scelto in modo tale da massimizzare un certo vincolo di destrezza, precisamente come

$$\dot{\boldsymbol{q}}_0 = k_0 \frac{\partial\, \boldsymbol{w}(\boldsymbol{q})}{\partial\, \boldsymbol{q}} \tag{2.5}$$

con $\boldsymbol{w}(\boldsymbol{q})$ che è una funzione che rappresenta un vincolo di destrezza.

Adesso però ci troviamo in una situazione per cui non è possibile applicare questo tipo di algoritmo, perché il nostro robot SCARA è un manipolatore NON ridondante a 4 gradi di libertà e che per il quale viene specificato un task che necessita di 4 variabili dello spazio operativo, quindi abbiamo uno Jacobiano che è una matrice quadrata 4×4. Per questo motivo se vogliamo applicare un algoritmo del genere dobbiamo necessariamente rilassare una delle 4 componenti dello spazio operativo, ottenendo così uno Jacobiano che è una matrice rettangolare 3×4, e di cui ci si può calcolare la pseudo-inversa destra.

La riflessione che c'è da fare adesso è capire quale di queste 4 componenti rilassare. Se guardiamo la matrice Jacobiana calcolata nella (1.4) possiamo notare diverse cose. Se decidessimo di rilassare la quarta componente, ossia quella relativa all'orientamento, dovremmo eliminare la quarta riga dello Jacobiano (1.4); il problema è che ciò comporterebbe l'ottenere uno jacobiano 3×4 che ha l'ultima colonna nulla, dato che l'ultima colonna ha tutti valori nulli eccetto proprio l'ultimo che eliminiamo. Pertanto di questo Jacobiano con l'ultima colonna nulla, se ne andassimo a calcolare la pseudo-inversa destra, otterremmo lo stesso risultato che avremmo ottenuto calcolandoci l'inversa dello jacobiano ridotto eliminando quest'ultima colonna nulla. Ciò quindi significa che se andassimo ad applicare la (2.4) otterremmo lo stesso identico risultato per le prime 3 componenti di quelle ottenute con l'inversa dello jacobiano, senza così ottimizzare il vincolo di destrezza, cioè di fatto è come se NON stessimo applicando l'algoritmo con pseudo-inversa dello Jacobiano. Infatti se ci facciamo caso, il quarto giunto rotoidale contribuisce solo per la parte di orientamento, e non influenza minimamente le altre 3 componenti dello spazio operativo. Discorso analogo se decidessimo di rilassare la terza componente dello spazio operativo, ossia la coordinata z. Perché se facessimo una cosa del genere dovremmo eliminare la terza riga dello (1.4) e anche in questo caso ciò restituirebbe uno Jacobiano 3×4 con terza colonna nulla, dato che il giunto prismatico è l'unico che contribuisce a modificare la componente z dello spazio operativo.

Se ne deduce quindi infine che se vogliamo applicare tale algoritmo bisognerà rilassare o la componente x o la componente y dello spazio operativo. Noi abbiamo deciso di rilassare la componente x, e quindi di eliminare la prima riga della

Jacobiana ottenendo la seguente matrice

$$\boldsymbol{J}_A(\boldsymbol{q}) = \begin{bmatrix} \frac{1}{2}(c_1 + c_{12}) & \frac{1}{2}c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix} \tag{2.6}$$

Inoltre c'è da decidere quale deve essere il vincolo di destrezza che vogliamo ottimizzare calcolandone il gradiente come nella (2.5). La nostra scelta è ricaduta come funzione $\boldsymbol{w}(\boldsymbol{q})$ da ottimizzare sulla cosiddetta misura di manipolabilità

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

Arrivati a questo punto siamo pronti per eseguire l'algoritmo. Lo schema Simulink che viene implementato è lo stesso identico di quello visto in figura 2.3 con l'unica differenza che nell'apposita MATLAB function al posto di calcolare l'inversa della Jacobiana analitica, si va a calcolare la pseudo-inversa destra della jacobiana e $\dot{\boldsymbol{q}}_0$, ossia ci si va a calcolare le velocità di giunto come nella (2.4).

I pesi scelti per la matrice $\boldsymbol{K}$ sono gli stessi di quelli usati per il CLIK con inversa dello Jacobiano con la differenza che stavolta la matrice diagonale è 3×3

$$\boldsymbol{K} = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

Gli andamenti delle variabili di giunto lungo la traiettoria sono le seguenti

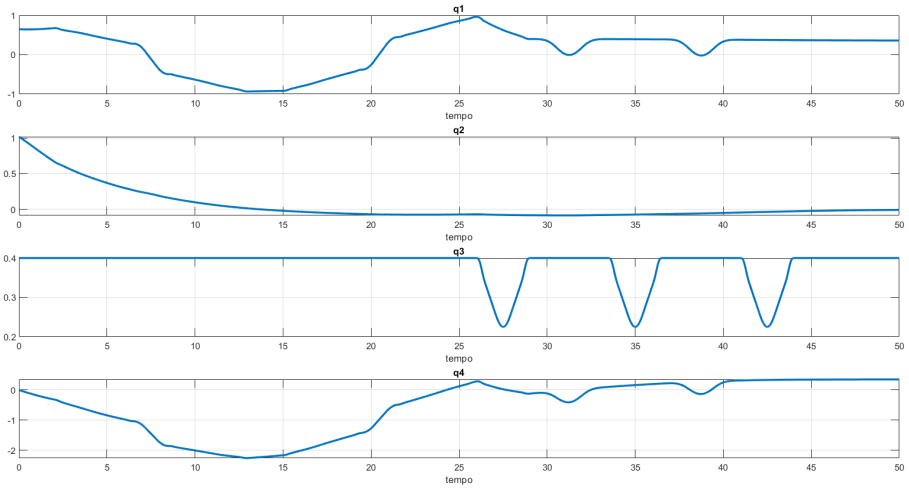

Figura 2.9: Traiettorie richieste ai giunti per l'algoritmo con pseudo-inversa della Jacobiana

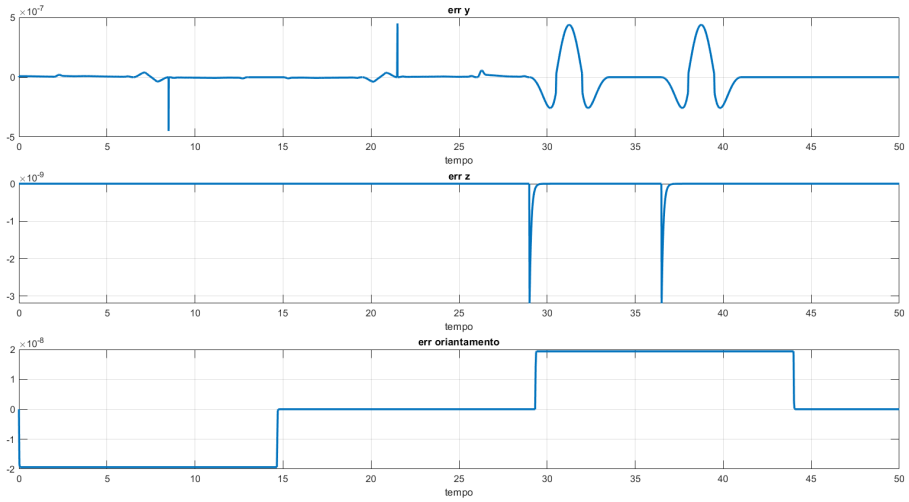

Figura 2.10: Andamento dell'errore tra posa effettiva e desiderata lungo la traiettoria

Infine per mettere in risalto quelli che sono i vantaggi di voler considerare un manipolatore ridondante, possiamo andare a graficare la misura di manipolabilità. Nello specifico andiamo prima a graficare la misura di manipolabilità senza considerare il termine del proiettore del nullo dello jacobiano, e poi grafichiamola con questo termine. Si può notare, ovviamente, come nel secondo caso tale misura di manipolabilità sia aumentata.

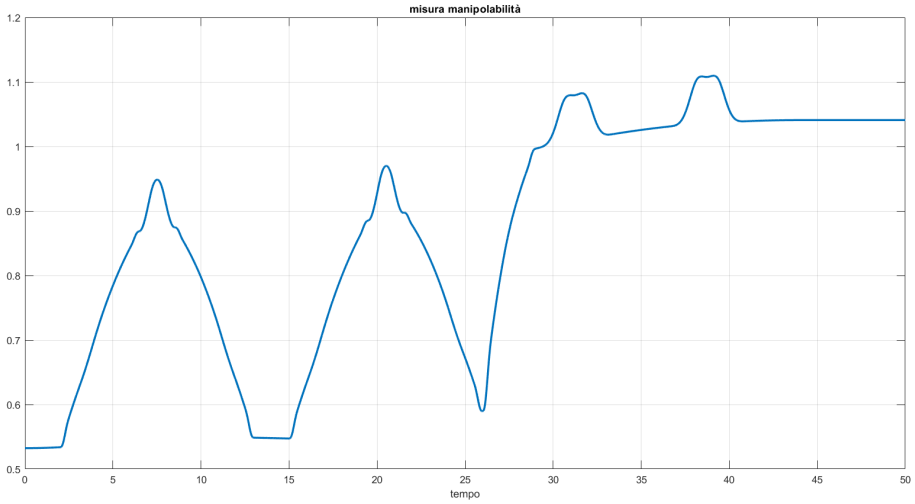

Figura 2.11: Andamento della misura di manipolabilità senza il proiettore nel nullo dello Jacobiano

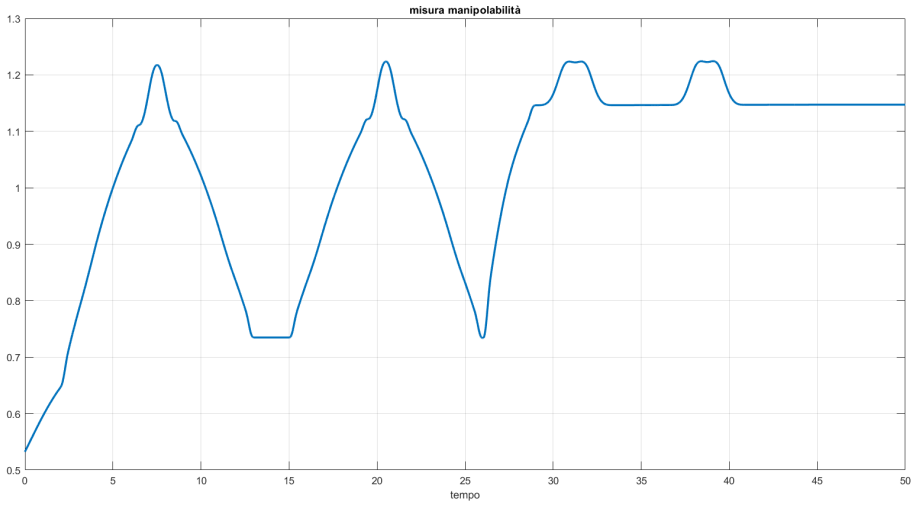

Figura 2.12: Andamento della misura di manipolabilità con il proiettore nel nullo dello Jacobiano

Capitolo 3

Dinamica e Controllo

Arrivati a questo punto dell'elaborato è giunto il momento di occuparci di effettuare l'analisi dinamica del manipolatore, ossia ricavare il suo modello dinamico, e di implementare tecniche di controllo nello spazio dei giunti per poter effettivamente controllare questo manipolatore.
In particolare come strategie di controllo da adottare, nell'ipotesi di un carico concentrato in punta di massa pari a circa 5 kg, progetteremo rispettivamente un controllo robusto e un controllo adattativo. Andremo a simulare in MATLAB il moto del manipolatore controllato, nell'ipotesi che le traiettorie desiderate ai giunti siano generate con un algoritmo per l'inversione cinematica del *secondo* ordine con inversa dello Jacobiano. Come al solito si implementeranno i controllori a tempo discreto con un periodo di campionamento di 1 ms.

3.1 CLIK del 2° ordine con inversa dello Jacobiano

Dato che per la progettazione degli schemi di controllo robusto ed adattativo servono anche le accelerazioni desiderate dei giunti, è richiesto per questo caso l'utilizzo come algoritmo per l'inversione cinematica un algoritmo CLIK del 2° ordine con inversa dello Jacobiano. Tale algoritmo è semplicemente una generalizzazione dell'algoritmo CLIK del 1° ordine con inversa dello Jacobiano visto in precedenza, dove stavolta si considera l'equazione della cinematica differenziale derivata nuovamente rispetto al tempo.

Con lo stesso identico ragionamento già visto per gli algoritmi CLIK del 1° ordine, si va a considerare la derivata seconda temporale dell'errore, e in essa si va a sostituire l'espressione della derivata seconda della posa effettiva (calcolata derivando la relazione di cinematica differenziale). Fatto questo dall'espressione che si ricava si può notare come sia conveniente calcolare le accelerazioni dei giunti come

$$\ddot{\boldsymbol{q}} = \boldsymbol{J}_A^{-1}(\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} - \dot{\boldsymbol{J}}_A(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}) \tag{3.1}$$

Procedendo in questo modo si riesce ad ottenere una dinamica dell'errore che è lineare del secondo ordine ed asintoticamente stabile. In tal caso però, rispetto a quello del 1° ordine, sarà necessario dover definire 2 matrici di pesi, anziché 1 sola, che abbiamo indicato con $\boldsymbol{K}_P$ e $\boldsymbol{K}_D$ e che abbiamo scelto in questo modo

$$\boldsymbol{K}_P = \begin{bmatrix} 1500 & 0 & 0 & 0 \\ 0 & 1500 & 0 & 0 \\ 0 & 0 & 1500 & 0 \\ 0 & 0 & 0 & 1500 \end{bmatrix} \qquad \boldsymbol{K}_D = \begin{bmatrix} 300 & 0 & 0 & 0 \\ 0 & 300 & 0 & 0 \\ 0 & 0 & 300 & 0 \\ 0 & 0 & 0 & 300 \end{bmatrix}$$

Lo schema Simulink progettato per l'implementazione di tale algoritmo CLIK è il seguente:

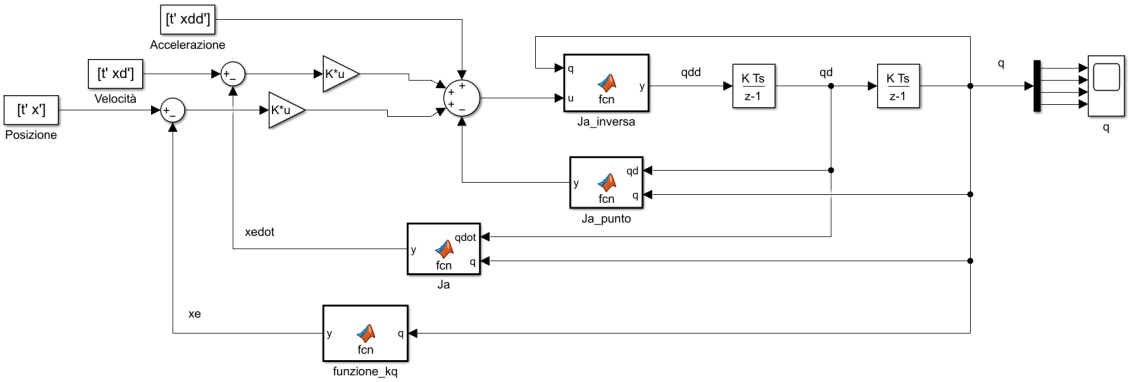

Figura 3.1: Schema Simulink dell'algoritmo CLIK del secondo ordine con inversa dello Jacobiano

Qui invece avremo le traiettorie ai giunti risultanti

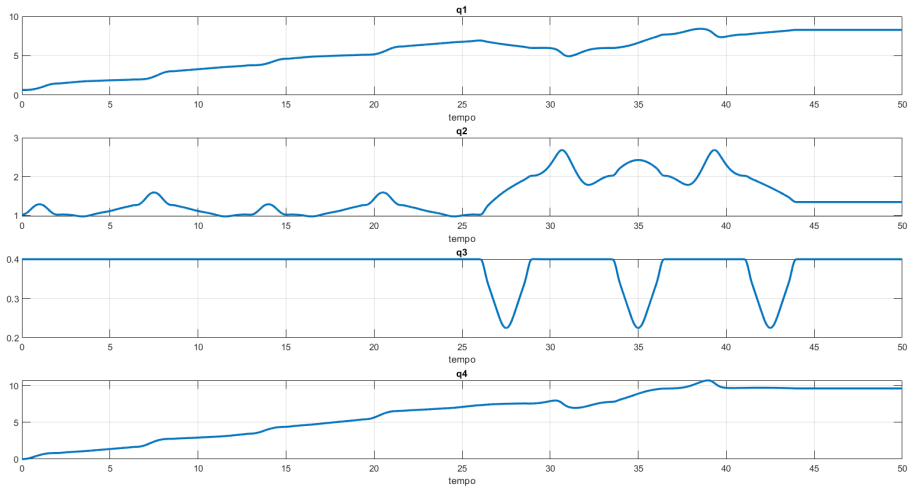

Figura 3.2: Traiettorie risultanti ai giunti per il CLIK del secondo ordine

E infine l'errore

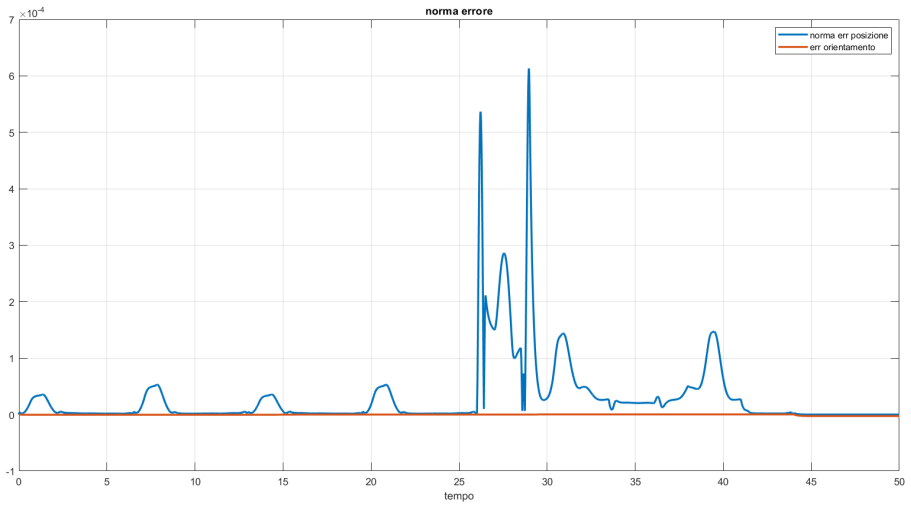

Figura 3.3: Andamento dell'errore tra posa effettiva e desiderata lungo la traiettoria

3.2 Controllo Robusto

Nel presente paragrafo vogliamo mostrare i risultati ottenuti con la tecnica di controllo robusto. Il controllo robusto viene utilizzato in conseguenza del fatto che non vengono compensati perfettamente tutti i termini non lineari del modello dinamico. Ciò non avviene o per una modellazione approssimata del modello dinamico o per compensazioni imperfette volontarie per via dell'elevato peso computazionale che ci sarebbe nel calcolare e quindi compensare tutti i termini del modello dinamico. La legge di controllo che utilizziamo si avvale dei seguenti contributi:

- $\hat{\boldsymbol{B}}\boldsymbol{y} + \hat{\boldsymbol{n}}$ con il quale abbiamo una compensazione approssimata dei termini non lineari del modello dinamico, nello specifico abbiamo scelto $\hat{\boldsymbol{B}} = \bar{\boldsymbol{B}}$ dove $\bar{\boldsymbol{B}}$ è una matrice diagonale con tutti elementi costanti ed è semplicemente la parte diagonale della matrice d'inerzia $\boldsymbol{B}$, mentre invece $\hat{\boldsymbol{n}}$ l'abbiamo scelta pari al solo termine di gravità $\boldsymbol{g}(\boldsymbol{q})$ quindi abbiamo trascurato il termine di attrito e quello delle forze di Coriolis;

- $\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_P\tilde{\boldsymbol{q}}$ dove $\ddot{\boldsymbol{q}}_d$ è un azione in feedforward e $\boldsymbol{K}_D\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_P\tilde{\boldsymbol{q}}$ che sono azioni in feedback che vengono utilizzate per stabilizzare la dinamica dell'errore, dove le matrici $\boldsymbol{K}_P$ e $\boldsymbol{K}_D$ sono state scelte così

$$\boldsymbol{K}_P = \begin{bmatrix} 400 & 0 & 0 & 0 \\ 0 & 529 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 2500 \end{bmatrix} \qquad \boldsymbol{K}_D = \begin{bmatrix} 40 & 0 & 0 & 0 \\ 0 & 46 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

- la legge di controllo a versore che rappresenta il contributo di robustezza che contrasta l'indeterminazione dovuta all'incertezza, e tale legge è espressa nella forma

$$\boldsymbol{w} = \begin{cases} \frac{\rho}{\|\boldsymbol{z}\|}\boldsymbol{z} & per\ \|\boldsymbol{z}\| \geq \varepsilon \\ \frac{\rho}{\varepsilon}\boldsymbol{z} & per\ \|\boldsymbol{z}\| < \varepsilon \end{cases} \tag{3.2}$$

  dove in quest'ultimo contributo sono stati assegnati i seguenti valori

  - $\rho = 4$;
  - $\varepsilon = 0.14$
  - $\boldsymbol{z} = \boldsymbol{D}^T\boldsymbol{Q}\boldsymbol{\xi}$ con $\boldsymbol{Q}$ che è la soluzione dell'equazione di Lyapunov

$$\boldsymbol{H}^T\boldsymbol{Q} + \boldsymbol{Q}\boldsymbol{H} = -\boldsymbol{P}$$

    dove $\boldsymbol{P}$ è stata scelta pari alla matrice identità $\boldsymbol{I}$ e dal MATLAB la matrice $\boldsymbol{Q}$ risultante sarà

$$\boldsymbol{Q} = \begin{bmatrix} 0.0625 & 0 & 0 & 0 & -0.5000 & 0 & 0 & 0 \\ 0 & 0.0544 & 0 & 0 & 0 & -0.5000 & 0 & 0 \\ 0 & 0 & 0.1253 & 0 & 0 & 0 & -0.5000 & 0 \\ 0 & 0 & 0 & 0.0250 & 0 & 0 & 0 & -0.5000 \\ -0.5000 & 0 & 0 & 0 & 5.0125 & 0 & 0 & 0 \\ 0 & -0.5000 & 0 & 0 & 0 & 5.7609 & 0 & 0 \\ 0 & 0 & -0.5000 & 0 & 0 & 0 & 2.5250 & 0 \\ 0 & 0 & 0 & -0.5000 & 0 & 0 & 0 & 12.5050 \end{bmatrix}$$

Lo schema Simulink che è stato progettato per il controllo robusto è riportato in figura 3.7. Di seguito andiamo a riportare l'andamento delle traiettorie ai giunti, l'andamento delle coppie ai giunti e quello degli errori sulle 4 variabili di giunto ottenuti in seguito alla simulazione di tale schema simulink. Tali grafici sono riportati rispettivamente in figura 3.4, 3.5 e 3.6. Adesso invece proviamo a vedere i risultati che si ottengono diminuendo ε in modo tale da ottenere un errore più basso. Proviamo a mettere un valore di ε pari a 0.04 per esempio. Quello che otterremo per l'errore è raffigurato in figura 3.8 e come possiamo notare è sicuramente più basso rispetto al caso precedente. Graficando pure le coppie però si potrà constatare come stavolta ci sia il chattering. I risultati sono riportati in 3.9

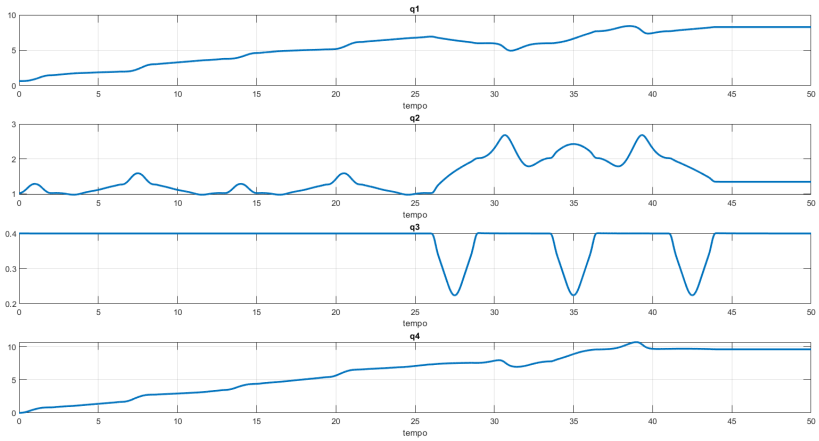

Figura 3.4: Traiettorie dei giunti nel controllo robusto

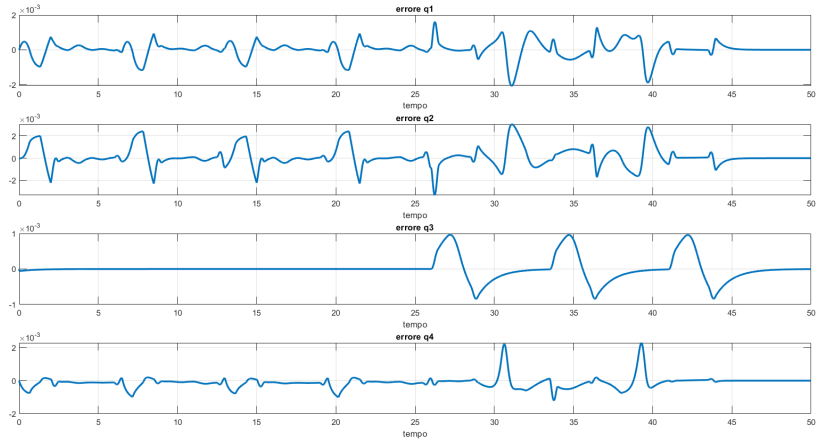

Figura 3.5: Andamento dell'errore sulle 4 variabili di giunto

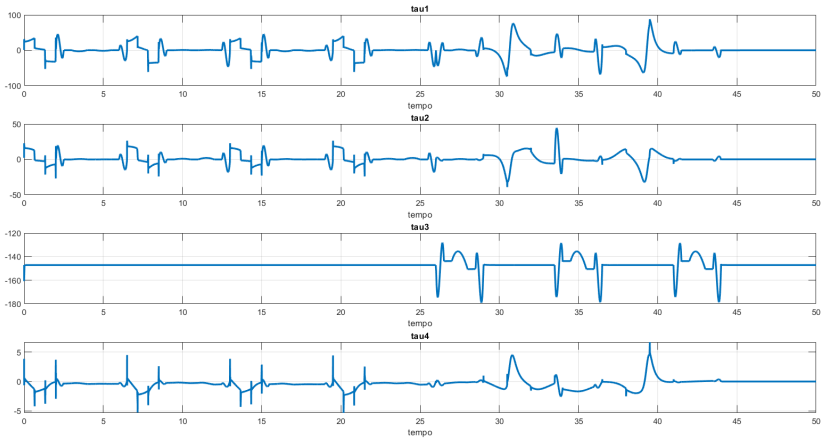

Figura 3.6: Andamento delle coppie che devono essere fornite ai giunti

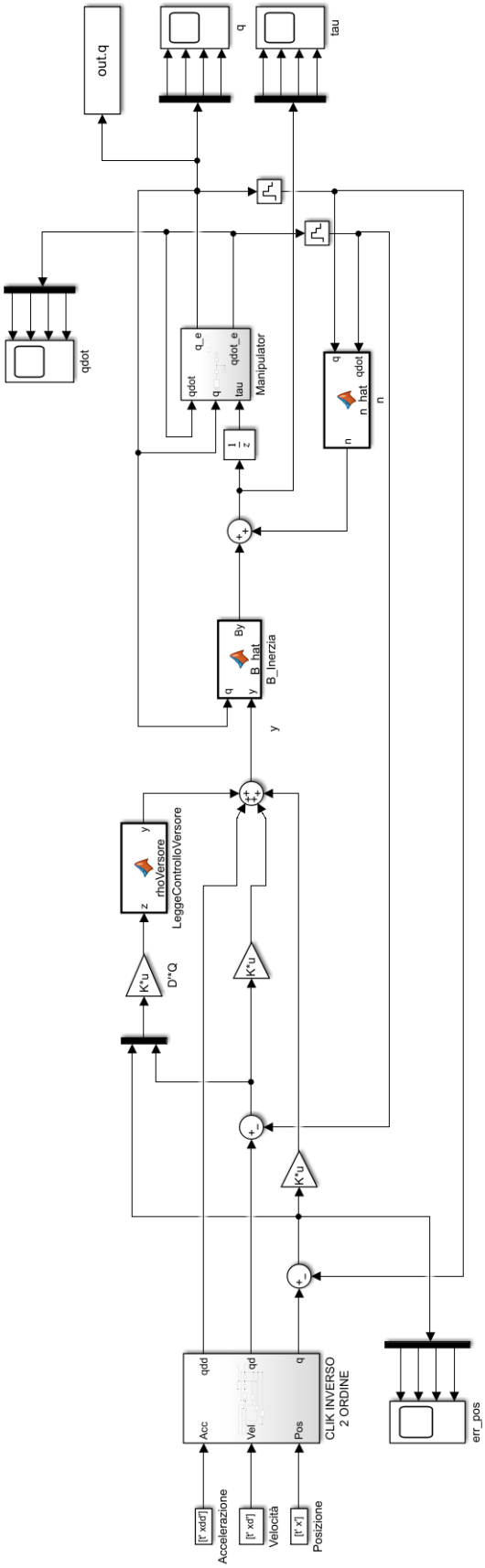

Figura 3.7: Schema Simulink del controllo robusto

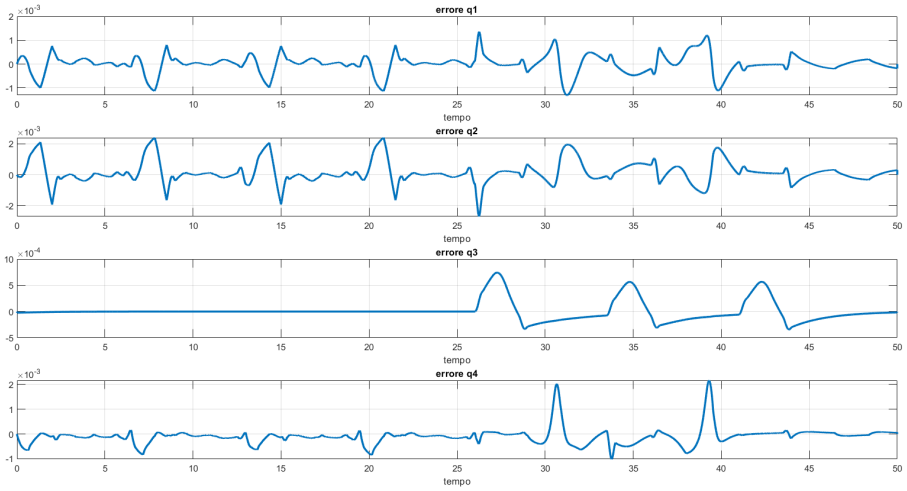

Figura 3.8: Andamento dell'errore sulle 4 variabili di giunto con ε diminuito

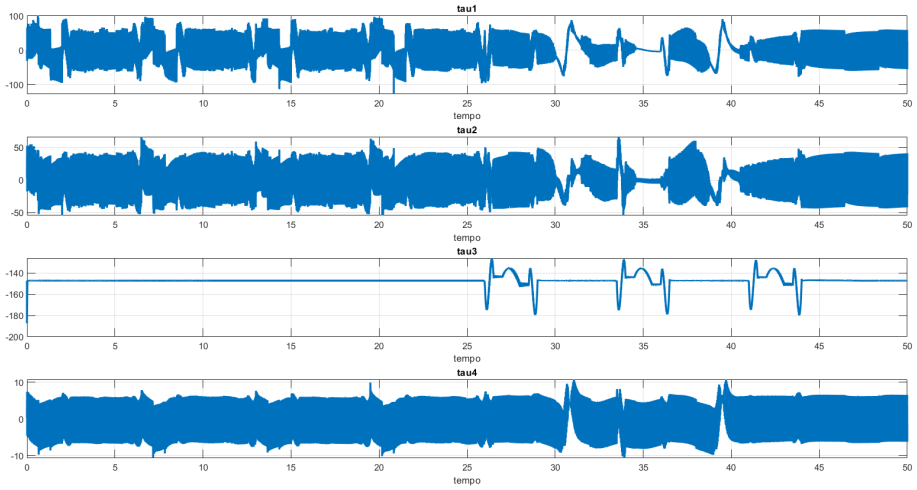

Figura 3.9: Andamento delle coppie ai giunti con chattering

controllo adattativo

3.3 Controllo Adattativo

Adesso illustriamo i risultati ottenuti mediante il controllo adattativo. Tale controllo viene utilizzato quando i parametri che caratterizzano il modello sono affetti da incertezza. Nello specifico il parametro affetto da incertezza è quello del carico.
La legge di controllo adattativo che dovremo andare a progettare consta dei seguenti termini:

- $\boldsymbol{Y}\tilde{\pi}$ per compensare approssimativamente le non linearità del modello;

- $\boldsymbol{K}_D\sigma$ introduce un'azione lineare stabilizzante di tipo PD sull'errore di inseguimento dove

  $$\boldsymbol{K}_D = \begin{bmatrix} 200 & 0 & 0 & 0 \\ 0 & 200 & 0 & 0 \\ 0 & 0 & 200 & 0 \\ 0 & 0 & 0 & 200 \end{bmatrix} \qquad \sigma = \dot{\tilde{\boldsymbol{q}}} + \boldsymbol{\Lambda}\tilde{\boldsymbol{q}} \quad dove \quad \boldsymbol{\Lambda} = \begin{bmatrix} 500 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 \\ 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 500 \end{bmatrix}$$

  con

- e poi un ultimo termine che è una legge di adattamento ai parametri di tipo a gradiente, dove la velocità di convergenza dei parametri al lor valore asintotico è determinato da una matrice $\boldsymbol{K}_\pi$ che è stata scelta nel seguente modo

  $$\boldsymbol{K}_\pi = K * diag([1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 2/K\ 1\ 1\ 1\ 1\ 1])$$

  dove K è stato posto pari a 10000 per concentrare l'incertezza solo sul parametro del carico.

Lo schema Simulink del Controllo Adattativo progettato è illustrato in figura 3.10 Lanciando tale schema simulink otterremo i seguenti risultati per le traiettorie effettive ai giunti, per gli errori sulle variabili di giunto e per le coppie ai giunti rispettivamente riportati in figura 3.11, 3.12 e 3.13.

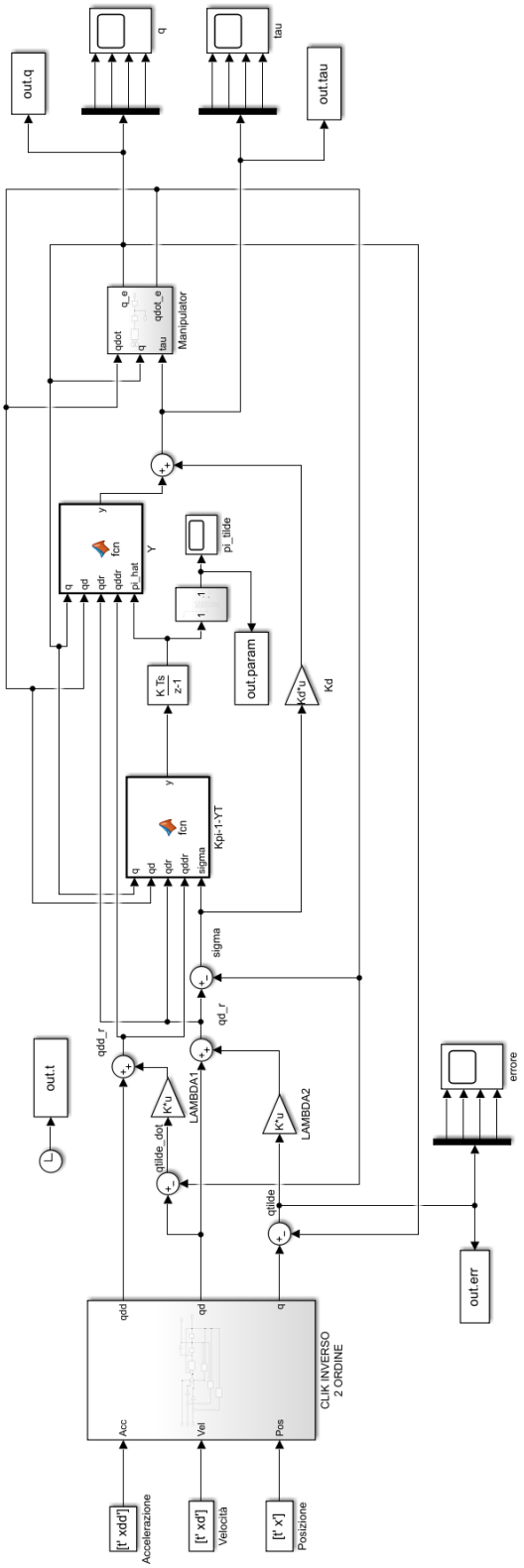

Figura 3.10: Schema Simulink del controllo adattativo

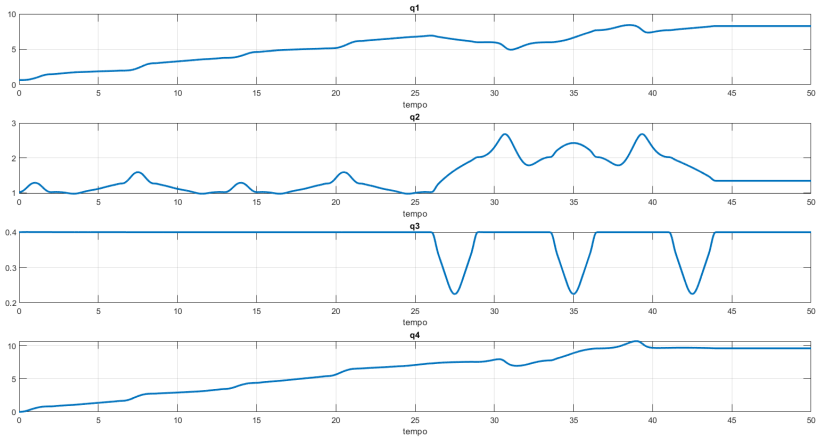

Figura 3.11: Traiettorie dei giunti nel controllo adattativo

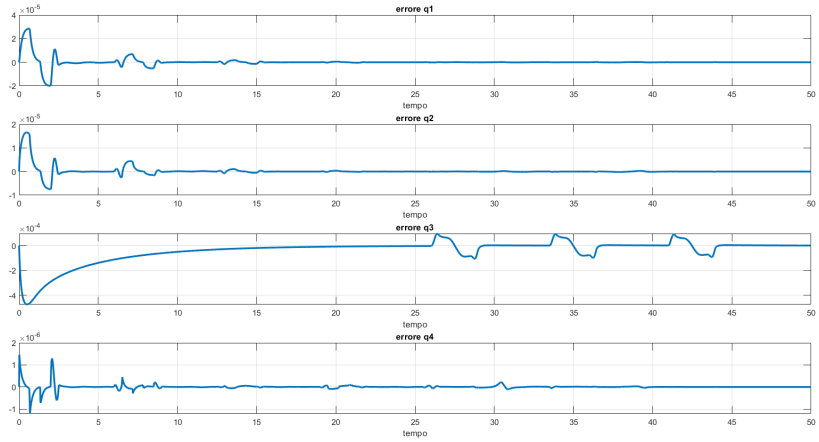

Figura 3.12: Andamento dell'errore sulle 4 variabili di giunto

Infine possiamo anche vedere come in questo particolare caso sia possibile stimare l'andamento del parametro ignoto relativo al carico.

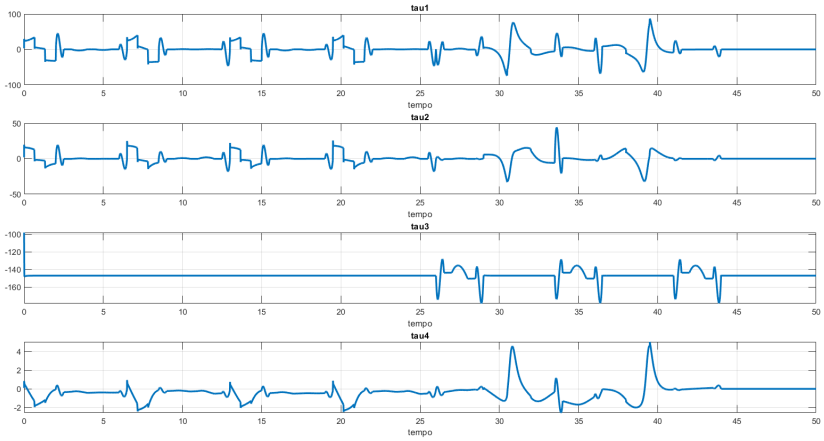

Figura 3.13: Andamento delle coppie che devono essere fornite ai giunti

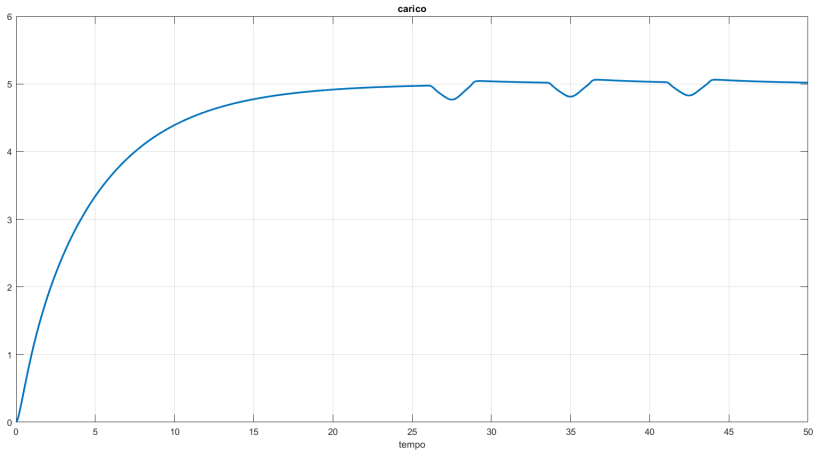

Figura 3.14: Andamento della stima del parametro

Capitolo 4

Implementazione reale del robot

In ultima analisi abbiamo pianificato una traiettoria e quindi progettato un algoritmo CLIK per l'inversione cinematica per un robot Scara stampato in 3D. Abbiamo usato degli stepper motor con micro stepping per raggiungere la risoluzione richiesta per ottenere la traiettoria desiderata. La traiettoria scelta è stata il simbolo di Yin e Yang come si può notare dalla seguente figura

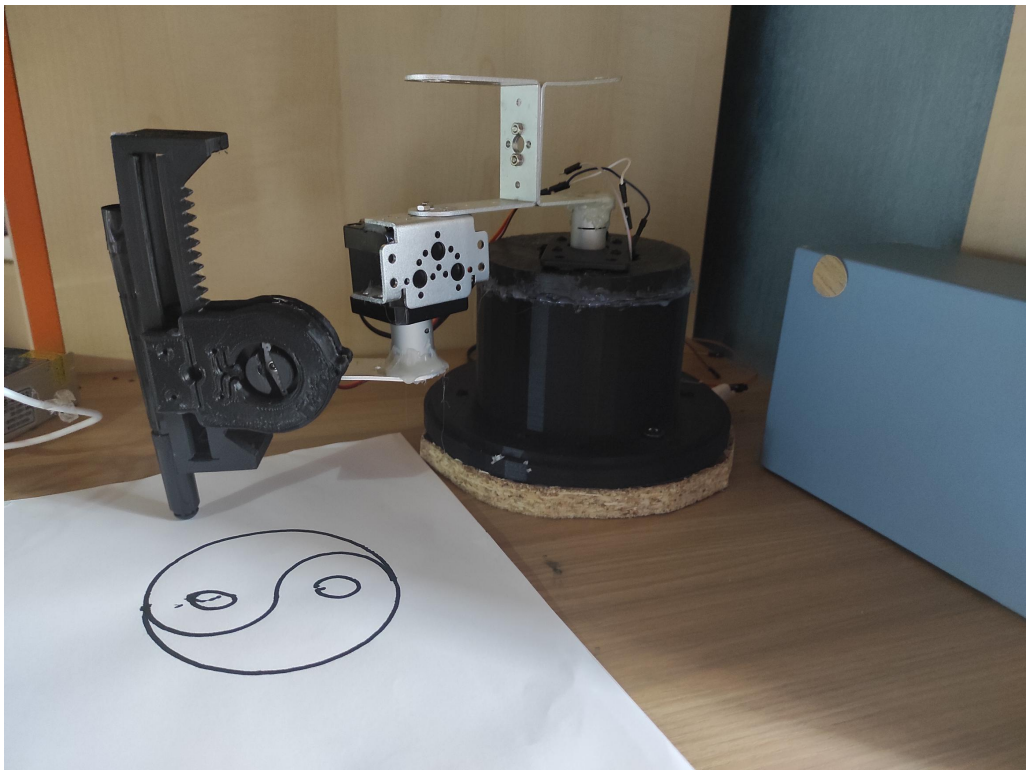

Figura 4.1: Immagine del robot reale stampato in 3D