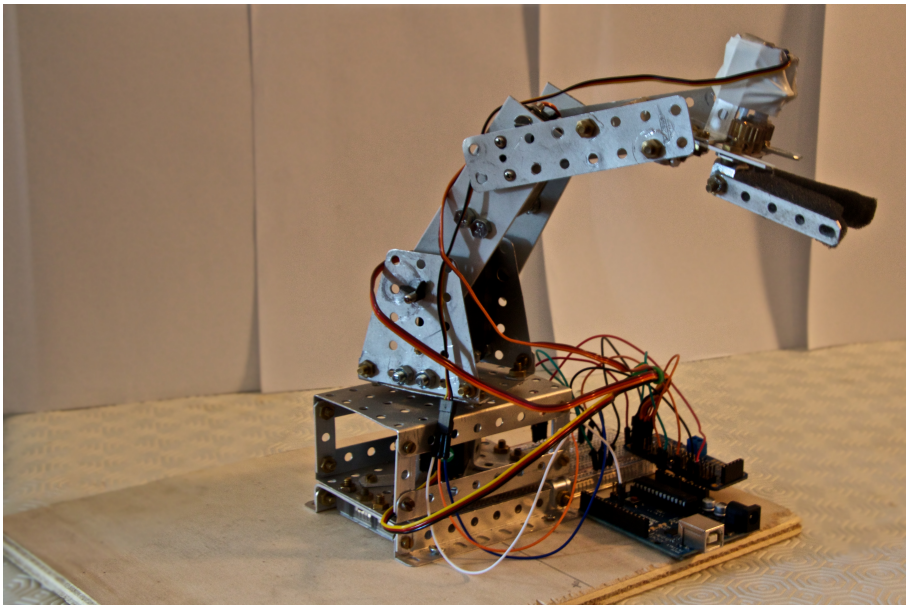
TESINA DI SISTEMI DI CONTROLLO MULTUVARIABILE



Robot Revoluto a 3 assi

Andrea Aleni, Dario Antonacci, Giulio Pisani
Febbraio 2013

RELAZIONE SCM

Robot Revoluto a 3 assi

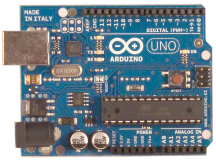
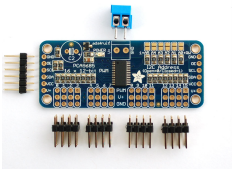
Andrea Aleni, Dario Antonacci, Giulio Pisani
Febbraio 2013

Introduzione

Il progetto è stato diviso in due parti distinte: la realizzazione fisica di un prototipo controllato in anello aperto tramite MATLAB e la realizzazione simulata di un modello matematico implementato in MATLAB/Simulink.

Il modello virtuale su MATLAB si è reso necessario per l'impossibilità di comandare in velocità i servo motori utilizzati sul prototipo fisico e si configura, quindi, come un modello puramente didattico che implementa le tecniche di controllo apprese durante il corso, nello specifico, il controllo di traiettoria.




Prototipo fisico

Il prototipo è stato realizzato partendo da pezzi provenienti dal gioco "Meccano" e da profilati di alluminio acquistati per lo scopo lavorati ad hoc. La struttura, pur risultando, costruita con pezzi "di fortuna" risulta essere molto stabile e permette di sollevare pesi fino a 200kg.

I servo motori sono di due tipi diversi, più prestanti i 3 che devono azionare i link e meno prestante, ma miniaturizzato quello che deve chiudere o aprire la pinza del polso.

Il controllo dei primi è stato eseguito tramite una scheda Adafruit 16 Channel PWM che permette di comandare simultaneamente fino a 16 servo motori inviando un impulso modulato, mentre l'ultimo motore è controllato direttamente dalla scheda Arduino. Per l'alta corrente di spunto si è reso necessario l'utilizzo di un alimentatore esterno da 2,5 A .

L'intera struttura è comandata da una scheda Arduino UNO che riceve in ingresso, tramite porta seriale, i valori degli angoli generati dal programma MATLAB e li converte nel corrispondente impulso che verrà poi inviato, mediante protocollo I2C, alla scheda che comanderà i motori.

È stato implementato, inoltre, un controllo di sicurezza tramite due fotodiodi che posizionati a destra e a sinistra del terzo link rilevano la presenza o meno di ostacoli lungo la traiettoria.

Si presenta ora la lista dei materiali utilizzati per la realizzazione fisica del prototipo.

Meccanica:

Base:

#1 base di compensato (30 x 21 cm)

#2 pezzi rettangolare in alluminio con bordi a doppia L (10 x 1 cm)

#2 pezzo rettangolare in alluminio a L (6 x 1 cm)

Link 0:

#2 base rettangolare in alluminio con bordi a L (10 x 6 cm)

#4 pezzo rettangolare in alluminio (3 x 1 cm)

#3 pezzo rettangolare in alluminio (5 x 1 cm)

#2 pezzo rettangolare in alluminio a L (3x3 cm)

Link 1:

#1 base rettangolare in allumino (6 x 4,5 cm)

#2 pezzo trapezoidale in alluminio (3x5x6 cm)

#2 vite passante in ferro (6,5 cm)

#2 vite passante in ferro (2,5 cm)

#1 albero in metallo (3,5 cm)

#3 ferma albero

Link 2:

#2 profilato in alluminio (10 x 3 cm)

#2 vite passante in ferro (6,5 cm)

#2 vite passante in ferro (2,5 cm)

#1 albero in metallo (3,5 cm)

#3 ferma albero

Link 3:

#2 pezzo rettangolare in alluminio (9,5 x 2 cm)

#2 pezzo rettangolare in alluminio a doppia L (5 x 1 cm)

Polso:

#2 pezzo rettangolare in alluminio (6 x 0.5 cm)

#2 pezzo rettangolare in alluminio a L (0,5 x 0,5 cm)

#1 pezzo ad U in alluminio (4 x 0,5 x 2 cm)

#2 pezzo rettangolare in alluminio a L (4 x 0,5 cm)

#2 ruota dentata (1,5 cm diametro)

#2 spugna

Elettronica:

#1 servo motore Hitec HS-325HB

#2 servo motore Hitec HS-422

#1 servo motore Hitec HS-55

#1 scheda a Microcontrollore Arduino UNO

#1 scheda PWM servo driver Adafruit 16-Channel

#1 piastra sperimentale

#2 fototransistore

#1 alimentatore 5V-2500mA

#30 Jumpers

Costruzione

Il modello è stato costruito partendo da dei pezzi di un meccano degli anni '60 con l'aggiunta di profilati in alluminio realizzati appositamente. La base contiene il primo motore che si occupa della movimentazione del link 0. Il primo link, realizzato con i pezzi di forma trapezoidale, contiene al suo interno il secondo motore, il quale mediante una flangia di nylon si aggancia con il secondo link. Quest'ultimo, per motivi di stabilità strutturale, è stato realizzato con i profilati succitati, e, per incrementare la rigidità della struttura, è stato bloccato mediante due viti passanti bloccate da dadi e controdadi. In maniera analoga al primo link anche qui è stato alloggiato il motore che permette di azionare il link successivo. Al termine di questo è stata montata la struttura che fa da supporto per la pinza fissa (end-effector). Tale pinza è stata realizzata mediante due ruote dentate, una delle quali è stata incollata sulla flangia del micro servo motore per consentirne apertura e chiusura. Al fine di aumentare la presa dello strumento sono state incollate due strisce di spugna.

Azionamento

L' azionamento della struttura si è reso possibile mediante l'uso di una scheda Adafruit 16 channel PWM, che non solo fornisce l'adeguata alimentazione ai servo motori (circa 2,5A), ma anche permette di comandarli in contemporanea mediante l'uso di impulsi modulati in frequenza. Questa shield comunica con il microcontrollore tramite il protocollo I2C, il quale prevede due soli canali, uno deputato alla trasmissione dei dati ed uno destinato al clock di sincronizzazione. Il motore relativo al movimento della pinza è stato collegato direttamente alla scheda Arduino poiché né necessita di controllo preciso (essendo solo apertura-chiusura) né di un'alimentazione esterna.

Controllo

Il controllo del robot è stato affidato ad una scheda a microcontrollore Arduino che gestisce: la comunicazione seriale (tramite I2C) verso la shield supplementare, il controllo di prossimità realizzato con il fototransistore, l'azionamento del micro servo motore e la comunicazione con l'ambiente di lavoro Matlab.

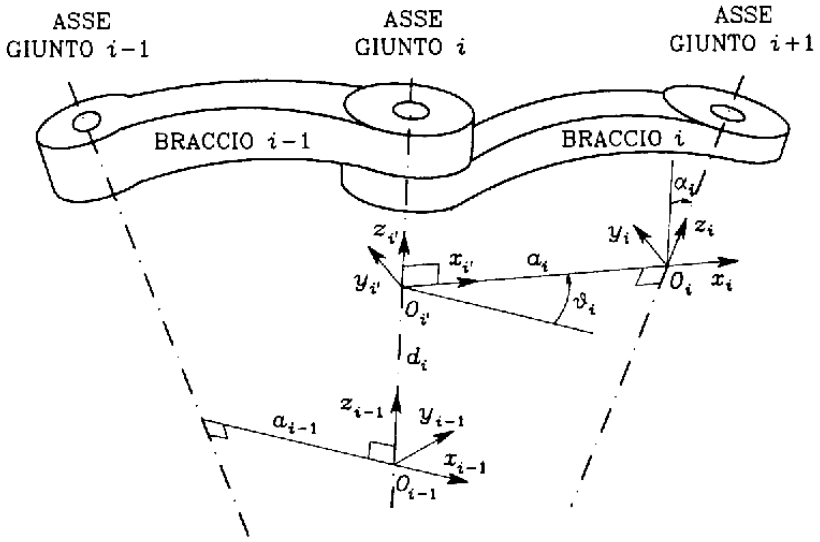
È stata realizzata una funzione Matlab che partendo dai principi di cinematica inversa prende in ingresso posizione iniziale e finale dell'end effector e genera come risultato tre angoli di giunto che verranno poi trasferiti tramite porta seriale al microcontrollore. La funzione calcola_angoli.m è allegata in Appendice A. Si è resa necessaria, inoltre, la creazione di un programma in linguaggio Sketch (proprietario di Arduino) che gestisce la comunicazione tra i segnali inviati dal Matlab e la scheda Adafruit.

Il programma riceve in ingresso, sulla porta seriale (emulata sull'USB), tre angoli generati dalla funzione Matlab e, tramite una funzione di mappatura tra angoli e frequenza degli impulsi, li trasforma in un vettore di impulsi da inviare ai motori. Per simulare un controllo di sicurezza, inoltre, è stato realizzato un blocco all'interno del programma che interrompe istantaneamente il movimento dei motori se un oggetto si trova sulla traiettoria del robot. Tale controllo è stato reso possibile mediante l'utilizzo di due fototransistori collegati in ponte ad una resistenza di $330 k\Omega$. Il sorgente dello sketch Arduino (servoultimo.ino) è disponibile in Appendice A.

Cinematica del Robot Antropomorfo

Per descrivere il robot si utilizza la convenzione di Denavit-Hartenberg la quale fornisce un metodo generale per le catene cinematiche aperte, e consente di fissare i sistemi di riferimento sui giunti (e link) per poterne determinare i parametri caratteristici. Da questa rappresentazione è possibile, tramite l'uso di matrici di roto-traslazione dei sistemi di riferimento, trovare un legame fra i parametri dei giunti e la posizione e l'orientamento dell'end-effector (nel caso presentato, una pinza).

Di seguito, con riferimento alla figura mostrata, diamo luogo ad una breve spiegazione della succitata convenzione. Si assuma come asse i l'asse del giunto che connette il braccio i-1 al braccio i e per la definizione della terna i (solidale al braccio i).



- si sceglie l'asse z_i giacente lungo l'asse del giunto i + 1 ;
- si individua O_i all'intersezione dell'asse z_i con la normale comune agli assi z_{i-1} e z_i , e con $O_{i'}$ si indica l'intersezione della normale comune con z_{i-1} ;
- si sceglie l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunto i+1;
- si sceglie l'asse y_i in modo da completare una terna levogira.

Una volta scelte le terne solidali ai bracci, la posizione e l'orientamento della terna i rispetto alla terna i -1 risultano completamente specificati dai seguenti parametri:

a_i distanza di O_i da $O_{i'}$,
d_i coordinata su z_{i-1} di $O_{i'}$

α_i angolo intorno all'asse x_i tra l'asse z_{i-1} e l'asse z_i valutato positivo in senso antiorario

θ_i angolo intorno all'asse z_i tra l'asse x_{i-1} e l'asse x_i valutato positivo in senso antiorario

I parametri a_i e α_i sono sempre costanti e dipendono soltanto dalla geometria di connessione dei giunti consecutivi dettata dalla presenza del braccio i.

Degli altri due, uno soltanto è variabile in dipendenza del tipo di giunto utilizzato per connettere il braccio i -1 al braccio i ; in particolare:

- se il giunto i è rotoidale a variabile è θ_i
- se il giunto i è prismatico la variabile è d_i

Pertanto una particolare configurazione del robot sarà individuata da un vettore $q = [\theta_1, \theta_2, \theta_3]^T$ mentre i parametri d_1, d_2, d_3 sono fissi.

Segue l'algoritmo che mette in relazione i parametri per il successivo uso nella simulazione:

1. Individuare e numerare consecutivamente gli assi dei giunti; assegnare, rispettivamente, le direzioni agli assi $z_0,\ldots,z_{n-1}$.

2. Fissare la terna 0 posizionandone l'origine sull'asse z_0; gli assi x_0 e y_0 sono scelti in maniera tale da ottenere una terna levogira.

(i passi 3-5 vanno ripetuti per i =1,...,n -1)

3. Individuare l'origine O_i all'intersezione di z_i con la normale comune agli assi z_{i-1} e z_i . Se gli assi z_{i-1} e z_i sono paralleli e il giunto i è rotoidale, posizionare O_i in modo da annullare d_i ; se il giunto i è prismatico, scegliere O_i in corrispondenza di una posizione di riferimento per la corsa del giunto.

4. Fissare l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunto i + 1 .

5. Fissare l'asse y_i in modo da ottenere una terna levogira.

6. Fissare la terna n, allineando z_n lungo la direzione di z_{n-1} se il giunto n è rotoidale, ovvero scegliendo z_n in maniera arbitraria se il giunto n è prismatico; fissare l'asse x_n in accordo al punto 4.

Cinematica diretta

Si vuole ora esprimere le relazioni che, a partire dalle configurazioni dei giunti del robot, permettono di ricavare le coordinate della terna solidale all'end-effector e quindi la sua posizione, ossia risolvere il problema della cinematica diretta. Si continua quindi con l'algoritmo di Denavit-Hartenberg focalizzando l'attenzione sui passi 5 e 6 precedentemente presentati.

7. Costruire per i=1,.....n la tabella dei parametri a_i, d_i,α_i,θ_i.

Braccio	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_2

si noti che la terna 0 è stata scelta con origine all'intersezione di z_0 e z_1 (cioè sul secondo giunto), da cui $d_1 = 0$ e che z_1 e z_2 sono paralleli e gli assi x_1 e x_2 sono scelti lungo gli assi dei relativi bracci, da cui gli altri valori. In questa assegnazione i valori a_2 e a_3 corrispondono alle lunghezze del secondo e del terzo braccio.

8. Calcolare sulla base dei parametri di cui al punto 7 le matrici di trasformazione omogenea $A^{i-1}(q)$ per $i = 1,..., n$. Queste esprimono il legame tra la terna solidale al giunto i e la terna solidale al giunto $i-1$ e contengono le operazioni da effettuare sull'una perché si sovrapponga all'altra. Per il robot presentato in base al calcolo delle operazioni si ottengono le seguenti matrici:

$$A_1^0(\theta_1)=\begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{e} \quad A_1^{i-1}(\theta_1)=\begin{bmatrix} c_i & -s_i & 0 & a_i c_i \\ s_i & c_i & 0 & a_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ per i=2,3....}$$

$$s_{i,\ldots\ldots,j} = \sin(\theta_i + \ldots\ldots + \theta_j)$$
$$c_{i,\ldots\ldots,j} = \cos(\theta_i + \ldots\ldots + \theta_j)$$

9. Calcolare la trasformazione omogenea $T_n^0(q) = A_1^0 \ldots\ldots A_n^{n-1}$ che fornisce posizione e orientamento della terna *n* rispetto alla terna 0.

Nel caso in esame si ha che $T_3^0(q) = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & s_1 & c_1(a_2c_2 + a_3c_{23}) \\ s_1c_{23} & -s_1s_{23} & -c_1 & s_1(a_2c_2 + a_3c_{23}) \\ s_{23} & c_{23} & 0 & a_2s_2 + a_3s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

10. Assegnate T^b e T^n, calcolare la funzione cinematica diretta $T_e^b(q) = T_0^b T_n^0 T_e^n$ che fornisce posizione e orientamento della terna utensile rispetto alla terna base.

La funzione che realizza la cinematica diretta in Matlab è la cindir.m di cui segue in Appendice B il codice.

Cinematica differenziale

A differenza della cinematica diretta che è un legame statico tra spazio dei giunti e spazio cartesiano, la cinematica differenziale utilizza il legame dinamico tra la velocità cartesiana dell'end-effector e le velocità dei giunti. Tali relazioni si basano su un operatore lineare detto *Jacobiano* che in generale dipende dalla configurazione in cui si trova il robot.

Esistono due tipi di Jacobiano, quello geometrico e quello analitico. Il primo descrive il legame tra le velocità dei vari giunti e quella dell'end-effector tramite una matrice di trasformazione dipendente dalla configurazione in cui si trova il manipolatore. Il secondo sfrutta una rappresentazione in forma minima della postura dell'end-effector partendo da un'operazione di differenziazione della funzione cinematica diretta rispetto alle variabili di giunto. Nel caso in esame i due tipi di Jacobiano coincidono e presentano la seguente struttura:

$$J_{P_i} = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

$$J_{o_i} = \begin{bmatrix} z_{i-1} \times (p - p_{i-1}) \\ z_{i-1} \end{bmatrix}$$

rispettivamente per un giunto prismatico e per uno rotoidale. Nel caso del robot antropomorfo si ha:

$$J = \begin{bmatrix} z_0 \times (p - p_0) & z_1 \times (p - p_1) & z_2 \times (p - p_2) \\ z_0 & z_1 & z_2 \end{bmatrix}$$

il calcolo dei vettori di posizioni dei bracci fornisce:

$$p_0 = p_1 \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad p_2 = \begin{bmatrix} a_2 c_1 c_2 \\ a_2 s_1 c_2 \\ a_2 s_2 \end{bmatrix} \quad p = \begin{bmatrix} c_1(a_2 c_2 + a_3 c_{23}) \\ s_1(a_2 c_2 + a_3 c_{23}) \\ (a_2 s_2 + a_3 s_{23}) \end{bmatrix}$$

e quello dei versori degli assi di rotazione dei giunti:

$$z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$z_1 = z_2 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix}$$

sostituendo si ottiene lo Jacobiano completo

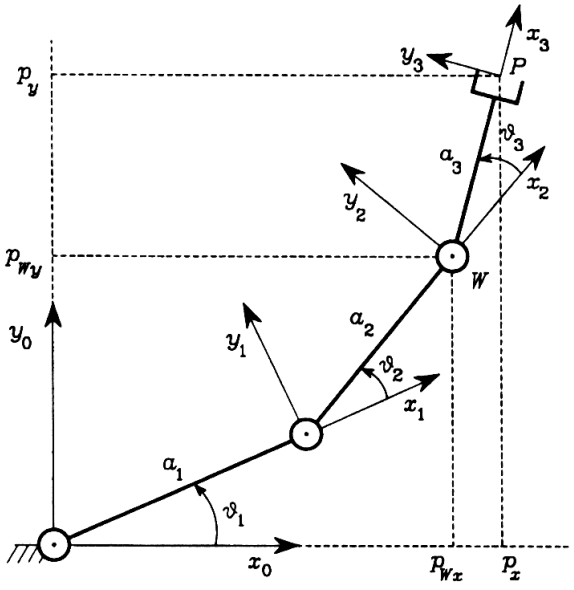
$$J = \begin{bmatrix} J_P \\ J_O \end{bmatrix} = \begin{bmatrix} -s_1(a_2c_2 + a_3c_{23}) & -c_1(a_2s_2 + a_3s_{23}) & -a_3c_1s_{23} \\ c_1(a_2c_2 + a_3c_{23}) & -s_1(a_2s_2 + a_3s_{23}) & -a_3s_1s_{23} \\ 0 & a_2c_2 + a_3c_{23} & a_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix}$$

Essendo il manipolatore mai ridondante con soli tre gradi di libertà si considerano solo le prime tre righe della matrice, quelle linearmente indipendenti. Il calcolo dello Jacobiano è implementato nel file Jac.m in Matlab. (si veda l'Appendice B)

Cinematica inversa

Il problema cinematico inverso permette di ricavare i parametri interni al robot (angoli dei giunti) a partire dalla posizione finale dell'end-effector desiderata. È di particolare interesse l'implementazione di una funzione che non solo controlli l'esistenza delle soluzioni, ma anche se essere esistono in numero finito o infinito. Data una determinata posizione dell'end-effector può esistere nessuna configurazione dei giunti tale da soddisfare il problema (la posizione è all'esterno dello spazio di lavoro), un numero finito di soluzioni (nel caso del robot antropomorfo in genere esistono 4 soluzioni, di cui di solito 2 coincidenti), infinite soluzioni (nel caso in cui ci troviamo in posizione di singolarità).

Dal caso planare del revoluto (tenendo fisso θ_1 la descrizione per θ_2 e θ_3 è analoga):



si pone

$$p_{Wx} = p_x - a_3 c_\phi = a_1 c_1 + a_2 c_{12}$$
$$p_{Wy} = p_y - a_3 s_\phi = a_1 s_1 + a_2 s_{12}$$

Nel caso planare p_{Wx} è la proiezione sull'asse x dell'origine della terna solidale al giunto

W mentre nel caso tridimensionale rappresenta, per un valore fissato dell'angolo relativo al primo giunto, la proiezione sull'asse x dell'origine della terna solidale all'end-effector; dal quale si vuole ricavare tramite inversione la configurazione dei giunti. Analoghe considerazioni valgono per p_{Wy} e p_{Wz} , dato che si sta lavorando in tre dimensioni. Per il primo giunto si ottengono così due soluzioni distinte:

$$\theta_1 = A\tan 2(p_{Wy}, p_{Wx})$$
$$\theta_1 = \pi + A\tan 2(p_{Wy}, p_{Wx})$$

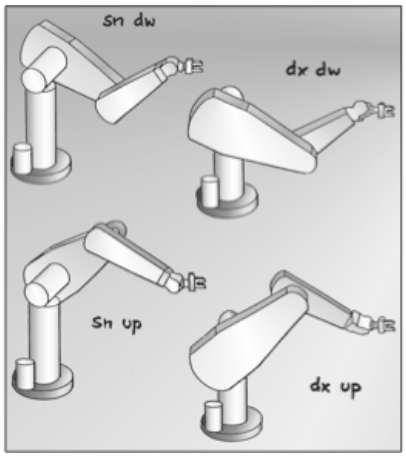
La funzione Atan2, a differenza della semplice funzione arco-tangente, considerando i segni degli argomenti, restituisce l'arco-tangente nel quadrante corretto.

Ora determinato θ_1 , considerando il manipolatore planare risulta facile calcolare θ_2 e θ_3 :

ponendo $c_3 = \dfrac{p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2 - a_2^2 - a_3^2}{2a_2 a_3}$ e $s_3 = \pm\sqrt{1 - c_3^2}$ dove con p_{Wz} si indica la proiezione del vettore posizione sull'asse z del riferimento, si ottiene $\theta_3 = A\tan 2(s_3, c_3)$. In maniera analoga si calcola θ_2 a patto di porre $c_2 = \dfrac{(a_2 + a_3 c_3)\sqrt{p_{Wx}^2 + p_{Wy}^2} + a_3 s_3 p_{Wz}}{p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2}$ e

$s_2 = \dfrac{(a_2 + a_3 c_3) p_{Wz} - a_3 s_3 \sqrt{p_{Wx}^2 + p_{Wy}^2}}{p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2}$, infatti $\theta_2 = A\tan 2(s_2, c_2)$.

Dai precedenti passaggi si evince l'esistenza di quattro soluzioni illustrate in figura al variare dei valori di θ_1 , θ_2 e θ_3 : spalla destra-gomito alto, spalla sinistra-gomito alto, spalla destra-gomito basso, spalla sinistra-gomito basso.



Il problema cinematico inverso esiste in una singola soluzione se e solo se $p_{Wx} \neq 0$ e $p_{Wy} \neq 0$, in caso contrario si può giungere in situazione di singolarità cinematica, cioè infinite soluzioni indipendentemente dal valore di θ_1 .

Nel caso in cui $p_{Wx} \neq 0$, $p_{Wy} \neq 0$ e $\theta_3 = 0$, cioè la situazione di gomito steso, si hanno due coppie di soluzioni coincidenti per cui la configurazione spalla-destra gomito alto corrisponde a spalla-destra gomito basso e spalla-sinistra gomito alto è identica a spalla-sinistra gomito basso.

Classificazione delle singolarità cinematiche

Le singolarità o configurazioni singolari si hanno quando la matrice che esprime lo Jacobiano perde di rango (determinante nullo); in tali configurazioni del manipolatore può accadere che:

- alcune direzioni di moto non siano realizzabili
- basse velocità dello spazio operativo corrispondono a velocità molto elevate ai giunti
- non si ha una ben definita soluzione del problema cinematico inverso

È chiaro dunque che tali punti (o zone) vanno evitati oppure affrontati con opportuni metodi. Esistono due classi di singolarità:

- ai confini dello spazio di lavoro: il manipolatore è completamente esteso o retratto; sono facilmente aggirabili evitando di portare il manipolatore ai confini dello spazio di lavoro. Per il manipolatore revoluto se le lunghezze degli ultimi due bracci non sono uguali si crea una zona di non raggiungibilità sferica e centrata nell'origine avente come raggio la differenza del più lungo con il più corto.

Nel modello cinematico del simulatore si è deliberatamente ignorato il limite imposto dal fine corsa dei servomotori (180°) e si è lasciato che tutti i giunti fossero liberi di ruotare nell'intervallo [0 , 2π] compiendo anche più giri.

- All'interno dello spazio di lavoro: allineamento di assi del moto o configurazioni particolari dell'organo terminale; sono in generale da evitare in fase di pianificazione della traiettoria.

Si analizzano in maniera rigorosa le singolarità del robot: non essendo dotato di polso esso può imbattersi soltanto in singolarità di struttura portante.

Il determinante dello Jacobiano da noi considerato è $\det(\mathcal{J}_P) = -a_2 a_3 s_3 (a_2 c_2 + a_3 c_{23})$. Nell'ipotesi di a_2 , $a_3 \neq 0$ il determinante si annulla per:

- $s_3 = 0$ si ha quando $\theta_3 = 0$ o $\theta_3 = \pi$, cioè quando il robot ha il gomito tutto steso o ripiegato su se stesso. Queste vengono chiamate *singolarità di gomito*.
- $(a_2 c_2 + a_3 c_{23}) = 0$ si ha quando l'origine del sistema di riferimento solidale all'end-effector si trova lungo l'asse di rotazione di base z_0 (infinite soluzioni), cioè quando $p_x = p_y = 0$. Questa viene detta singolarità di spalla. In particolare nell'origine del sistema di riferimento adottato, se essa è raggiungibile, cioè se $l_2 = l_3$, si hanno ∞^2 soluzioni. Un modo di descriverle è per esempio fissare θ_3 a π oppure a $-\pi$ e far variare indipendentemente gli altri due parametri θ_2 in $[-\pi / 2, +\pi / 2]$ e θ_1 in $[0, 2\pi]$.

Nel caso di singolarità sul confine interno dello spazio di lavoro, qualora i bracci l_2 ed l_3 fossero di lunghezza diversa, si trovano sempre quattro soluzioni distinte sebbene l'unico orientamento possibile per l'end-effector, che essendo ancorato al terzo braccio è esattamente θ_3, è comune a tutte e quattro le soluzioni.

Il file che sviluppa la cinematica inversa è invanalitica.m consultabile nell'Appendice B. Nel caso in cui ci siano infinite soluzioni possibile la funzione ricerca i valori ammissibili delle configurazioni dei giunti.

Implementazione della cinematica differenziale

Considerando la relazione $v = J(q)\dot{q}$ che coinvolge lo Jacobiano geometrico, nel caso in cui lo Jacobiano è quadrato e non singolare, l'equazione della cinematica differenziale inversa diventa $\dot{q} = J^{-1}(q)v$. Questa permette, assegnata la velocità della terna utensile e la configurazione del manipolatore, di determinare le velocità dei giunti. e consente, quindi, di risolvere il problema della cinematica inversa.

Assegnata la traiettoria della terna utensile e la velocità ad essa associata $v(t)$ è possibile determinare le variabili di giunto corrispondenti integrando la relazione $\dot{q}(t) = J^{-1}(q(t))v(t)$ a partire da una condizione iniziale nota $q(0)$; questa condizione deve essere tale che la terna utensile sia nella posizione iniziale desiderata. L'integrazione può essere eseguita per via numerica come realizzato nel programma presentato. Tale metodo si pone alternativo alla risoluzione analitica del problema cinematico inverso.

Le soluzioni della cinematica differenziale inversa possono essere calcolate solo nell'ipotesi in cui lo Jacobiano sia di rango pieno. Nelle configurazioni singolari il sistema associato contiene equazioni linearmente dipendenti. È possibile comunque determinare una soluzione di $\dot{q}$ estraendo tutte le equazioni linearmente indipendenti, quando il percorso assegnato nello spazio operativo è fisicamente eseguibile da parte del manipolatore.

Va considerato che lo Jacobiano può dar luogo a una serie di problemi che riguardano le velocità imposte ai giunti. Il determinante assume, man mano che ci si avvicina ad una singolarità valori tendenti allo zero, il che significa imporre grandi velocità ai giunti, situazione da evitare per la sua irrealizzabilità pratica. Per ovviare a questo problema allora si utilizza l'*inversa ai minimi quadrati smorzata*:

$$J^* = J^T(JJ^T + \lambda^2 I)^{-1}$$

così facendo si aggiunge un fattore di correzione che modifica il comportamento delle inversioni seguendo l'avvicinamento alle zone di singolarità. Il fattore λ , dunque, produce uno smorzamento che ammorbidisce l'inversione e migliora il comportamento.

Metodi per l'inversione della cinematica differenziale

La simulazione, implicando un processo di discretizzazione delle funzioni per l'inversione, rispetto al caso continuo, comporta fenomeni di deriva per la soluzione: non si ha una corrispondenza esatta tra configurazione di giunti calcolata e postura effettivamente raggiunta. Si può ovviare a questo problema tenendo conto dell'errore tra la posizione desiderata e la posizione calcolata:

$e = x_d - x$, da cui, derivando si ha $\dot{e} = \dot{x}_d - \dot{x}$. Sfruttando le formule della cinematica differenziale poc'anzi mostrate si ha $\dot{e} = \dot{x}_d - J_A(q)\dot{q}$. Per trovare il legame tra $\dot{q}$ ed e si sfrutta il metodo di Newton (ed inversione robusta).

Tale algoritmo si utilizza per l'approssimazione delle soluzioni di problemi numerici (non lineari) e si riassume nella seguente espressione:

$$f(q) = f(q^k) + J_r(q^k)[q - q^k] + o(\|q - q_k\|^2)$$

posta la *f(q)* come la cinematica diretta del manipolatore e ignorando l' *o* piccolo allora è possibile riscrivere tale espressione come:

$$q_{k+1} = q_k + J^{-1}(q_k)[r - f(q_k)]$$

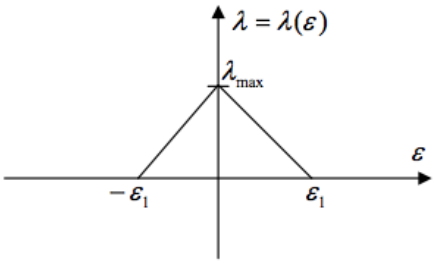
si giunge quindi a:

$$\dot{q} = J_A^{-1}(q)(\dot{x}_d + Ke)$$

Il parametro *K* è una matrice definita positiva diagonale che fa tendere l'errore a zero lungo la traiettoria desiderata con una velocità di convergenza che dipende dai suoi autovalori: più sono grandi, più è veloce la convergenza. L'inversa $\mathcal{J}^{-1}$ può essere adattata in base alle esigenze specifiche. Se il percorso passa vicino ad una zona di singolarità la normale inversione dello Jacobiano comporta un aumento esponenziale dell'algoritmo. È pertanto opportuno modificare il metodo per l'inversione definendo una nuova inversione "robusta", secondo il metodo poc'anzi visto dei minimi quadrati smorzati. La nuova espressione diventa:

$$\dot{q} = J^*(q)(\dot{x}_d + Ke)$$

Nel caso in questione, si esamina dapprima una configurazione in cui l'end-effector risulta prossimo ad una singolarità. Si considera una posizione in cui la configurazione di giunti risulta singolare per via dell'annullamento del determinante. La lontananza dalla singolarità è indicata con un ε che rappresenta la distanza del punto dal "centro" della singolarità. Se il manipolatore è prossimo ad una zona di singolarità, come l'asse z_0 , allora ε sarà la distanza dell'end-effector dall'asse. Si trovano, infatti, infiniti "centri" di singolarità vicini al manipolatore e l'ε scelto è per l'appunto quello relativo al più vicino. Per valori inferiori a ε_1 , cioè da un certo ε fissato in poi (in direzione dello zero), si un profilo triangolare al parametro λ in funzione di ε :



L'espressione di tale funzione risulta analiticamente:

$$\lambda(\varepsilon) = \lambda_{\max}\left(1 - \frac{|\varepsilon|}{\varepsilon_1}\right) \text{ con } \varepsilon \le \varepsilon_1$$

nell'implementazione di tale profilo in Matlab si è scelto di utilizzare la funzione solo nel primo quadrante (positivo) in quanto il programma non permette di gestire vettori con coordinate negative, senza peraltro alterare effettivamente il comportamento della funzione: al centro della singolarità la correzione λ è massima, mentre, man mano che l'end-effector si allontanana dal centro, λ decresce sempre più per ogni direzione in modo lineare. La funzione che implementa tale profilo è contenuta nel file lam.m (vedi Appendice B). La funzione che implementa l'inversione della cinematica differenziale è cidiffinv.m (vedi Appendice B)

Pianificazione delle traiettorie

L' obiettivo della pianificazione di traiettorie è quello di produrre i riferimenti che assicurano l'esecuzione da parte dell'end-effector delle traiettorie specificate. Questi riferimenti consistono in una sequenza temporale dei valori assunti dalla funzione scelta come traiettoria. La traiettoria può essere convenientemente scomposta nei due ingressi

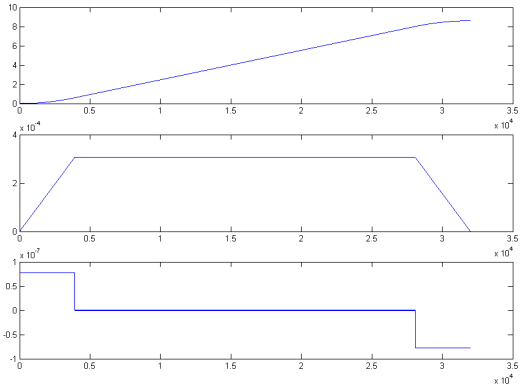
- percorso: luogo dei punti dello spazio dei giunti o dello spazio operativo che il manipolatore deve descrivere per eseguire il movimento assegnato

- legge oraria: funzione che esprime lo spazio percorso dal manipolatore al variare del tempo (legge lineare, cubica, polinomiale di grado n, spline...) a partire dal tempo iniziale.

Nella simulazione effettuata la pianificazione può essere eseguita nello spazio cartesiano. Così facendo per quanto riguarda la legge di moto possono essere specificati gli estremi, il tempo di percorrenza e l'accelerazione massima. Quello che verrà poi generato dall'algoritmo dovrà essere riportato, per il controllo del robot, nello spazio dei giunti tramite le inversioni.

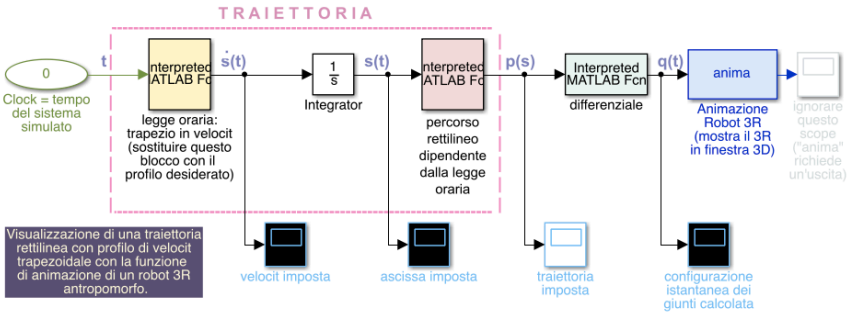
Si vuole che l'end-effector segua un percorso predefinito in linea retta portandosi da un punto iniziale p_0 ad un punto finale p_f entro un intervallo di tempo prefissato. Si considera allora il segmento che congiunge i due punti nello spazio che avrà come rappresentazione parametrica: $p(s) = p_i + \frac{s}{\|p_f - p_i\|}(p_f - p_i)$. La legge oraria utilizzata $s = s(t)$ è la legge oraria trapezoidale o *"bang-coast-bang"* (in accelerazione) composta da tre fasi di moto, insieme alla quale viene imposto che le velocità iniziali e finali devono essere nulle, i tratti di accelerazione e decelerazioni costanti e in modulo uguali, e il tratto intermedio deve avere velocità costante

$$s(t)=\begin{cases} a_{\max}\dfrac{t^2}{2} & t\in[0,T_s] \\ v_{\max}t-\dfrac{v_{\max}^2}{2a_{\max}} & t\in[T_s,T-T_s] \\ -a_{\max}\dfrac{(t-T)^2}{2}+v_{\max}T-\dfrac{v_{\max}^2}{a_{\max}} & t\in[T-T_s,T] \end{cases}$$



la funzione che implementa tale profilo di velocità è contenuta nel file trapezio.m, invece la funzione che calcola la traiettoria è contenuta nel file percorso.m.

Sviluppo del blocco Simulink



Viene generato, attraverso un clock, il tempo utilizzato dalla funzione 'trapezio.m' (giallo) che genera la legge oraria $s(t)$ la quale è elaborata dalla funzione 'percorso.m' (rosa) dando luogo ai riferimenti $p(s)$ per l'end-effector. I valori vengono invertiti con la 'cindiffinv' (verde) e passati alla s-function di animazione 'anima' che ricostruisce l'immagine del robot in base alla configurazione istantanea dei giunti. Il blocco integratore integra l'ingresso nel tempo a partire dalla condizione iniziale specificata (zero).

L'interfaccia del simulatore permette di assegnare la coppia (p_0, p_f), posizione iniziale e posizione finale. Si è visto che utilizzando l'inversa analitica ('invanalitica.m') si hanno in uscita più valori per sia per p_0 sia per p_f, dato che se esistono soluzioni sono almeno due, per si rende necessaria una funzione ulteriore che selezioni tra le varie soluzioni quella migliore secondo un criterio. Si è presa tra le possibili coppie (q_0, q_f) quella che minimizza la distanza nello spazio dei giunti $d(q_0, q_f)$ calcolata secondo la norma euclidea. Pur non essendo un metodo rigoroso

si è verificato mediante numerosi esperimenti che produce risultati di accettabile accuratezza. Il calcolo è svolto dalla funzione invsceltanormaminima.m (vedi Appendice B).

Appendice A

Sorgenti Matlab e Arduino modello reale

calcolo_angoli.m

```
function [ th1,th2,th3 ] = calcolo_angoli( Px0,Py0,Pz0,Pxf,Pyf,Pzf );

Pm=sqrt(Px0^2+Py0^2);

Pz10=Pz0-110;
th10=atan2d(Py0,Px0);
th30=acosd((Pm^2+Pz10^2-75^2-130^2)/(2*75*130));
th20=atan2d(Pz10,Pm)-atan2(130*sin(th30),75+130*cos(th10));

Pm=sqrt(Pxf^2+Pyf^2);

Pz1f=Pzf-110;
th1f=atan2d(Pyf,Pxf);
th3f=acosd((Pm^2+Pz1f^2-75^2-130^2)/(2*75*130));
th2f=atan2d(Pz1f,Pm)-atan2(130*sin(th3f),75+130*cos(th1f));

th1=th1f-th10;
th2=th2f-th20;
th3=th3f-th30;

th1=round(th1)
th2=round(th2)
th3=round(th3)

%%creo la connessione con la porta seriale
s=serial('/dev/tty.usbmodem621');
fclose(s);
s.InputBufferSize = 10000;
s.OutputBufferSize = 10000;
s.Terminator = ';';
fopen(s);

%%invio i valori del vettore k
k=[th1,th2,th3];
k1='';
for i=1:1:3
    k1=[k1, num2str(k(i)), ' '];
end
k1
fwrite(s,k1);
fwrite(s,';');
```

```
%%salvo i campioni ottenuti in data e chiudo la connessione

%%chiudo la connessione
fclose(s);
delete(s);
clear s

end
```

servoultimo.ino

```
/*************************************************
  These displays use I2C to communicate, 2 pins are required to
  interface. For Arduino UNOs, thats SCL -> Analog 5, SDA -> Analog 4
   *************************************************/
#include <Servo.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
#define SERVOMIN  150 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  577 // this is the 'maximum' pulse length count (out of 4096)

// our servo # counter
uint8_t servonum = 0;
Servo myservo;
int pos = 0;
int a,b;
int tempo, tempo0, tempo1, tempo2, tempo_max;
int i=0;
int d1,d2,d3;
int ang_max2;
int ang_max[3];
char s[10];
char s1[10];

void setup() {
   pinMode(A0, INPUT);
 myservo.attach(9);
 Serial.begin(9600);
 Serial.println("16 channel Servo test!");
 tempo=15;
```

```
 pwm.begin();

 pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates
}

void setServoPulse(uint8_t n, double pulse) {
 double pulselength;

 pulselength = 1000000;   // 1,000,000 us per second
 pulselength /= 60;   // 60 Hz
 Serial.print(pulselength); Serial.println(" us per period");
 pulselength /= 4096;  // 12 bits of resolution
 Serial.print(pulselength); Serial.println(" us per bit");
 pulse *= 1000;
 pulse /= pulselength;
 Serial.println(pulse);
 pwm.setPWM(n, 0, pulse);
}

void loop() {
while(1)
{
 for(i=0; i<3; i++) ang_max[i]=0;
 i=0;
 while(i!=3){
  if (Serial.available() > 0) {
    int p=0;
    Serial.readBytesUntil(' ', s, 8);
    while(s[p]!='\0'){
    ang_max[i]=ang_max[i]*10+s[p]-48;
    p=p+1;
    }
    ang_max[i]=map(ang_max[i],0,180,150,577);
    Serial.println(ang_max[i]);
    memcpy(s,s1,10);
    i=i+1;
  }
 }
 for(pos = 0; pos < 80; pos += 1)
 {
  myservo.write(pos);
```

```
  delay(15);
 }

 ang_max2=max(ang_max[0],ang_max[1]);
 ang_max2=max(ang_max2,ang_max[2]);
 d1=0;
 d2=0;
 d3=0;
for (uint16_t pulselen = 0; pulselen < ang_max2; pulselen++) {
  while(analogRead(A0)>=900){Serial.println(analogRead(A0));delay(100);}
  Serial.println(analogRead(A0));

  if (d1<=ang_max[0]){
   pwm.setPWM(0, 0, pulselen);
   d1++;
   }
  if (d2<=ang_max[1]){
   pwm.setPWM(1, 0, pulselen);
   d2++;
   }
  if (d3<=ang_max[2]){
   pwm.setPWM(2, 0, pulselen);
   d3++;
   }
   delay(tempo);
 }

 delay(500);

 for(pos = 80; pos>=1; pos-=1)
 {
  myservo.write(pos);
  delay(15);
   }

 for (uint16_t pulselen = ang_max2; pulselen > 0; pulselen--) {

  while(analogRead(A0)>=900){Serial.println(analogRead(A0));delay(100);}
   if (d1>=0){
   pwm.setPWM(0, 0, pulselen);
   d1--;
```

```
    }
   if (d2>=0){
     pwm.setPWM(1, 0, pulselen);
     d2--;
     }
   if (d3>=0){
     pwm.setPWM(2, 0, pulselen);
     d3--;
     }
     delay(tempo);
  }
  delay(500);

  for(pos = 0; pos < 50; pos += 1)
  {
   myservo.write(pos);
   delay(15);
  }
}
}
```

Appendice B

Sorgenti Matlab modello simulato

- **main.m**

```
%
%       Si richiedono posizione iniziale e finale dell'end-effector
% quindi il programma calcola, in base ai parametri scelti per il
% manipolatore, una traiettoria su percorso rettilineo tra i due punti,
% imponendo un profilo di velocità trapezoidale, entro
% l'intervallo di tempo specificato.
%
%       Quindi vengono rappresentati i movimenti del robot animandolo in
% 3D con grafici relativi a posizioni (e velocità) cartesiane e di giunto.
%
% funzioni richiamate durante la simulazione (digitare l'help relativo):
%
% dati3R.m              % paramdiff.m     % anima.m
% cindir.m              % percorso.m      % trapezio.m
% invanalitica.m        % dist2p.m        % Jac.m
% cindiffinv.m          % lam.m
% invsceltanormaminima  % test.m
%
% Bloccho simulink utilizzato
% - cartetrapezio


clear all
close all

% inizializzazioni varie

global p0 pf q0 qf L time amax vmax l1 l2 l3 sing ngrad nnewt sceltaconf scelta-
rad sceltaspace...
    sceltaprofilocart sing ngrad nnewt eps lmax Id K Ks alpha dTG dTN dTNS qd sp
accuracy;

l1=0;                       % lunghezza 1° braccio
l2=0;                       % lunghezza 2° braccio
l3=0;                       % lunghezza 3° braccio
time=-1;                    % tempo dedicato al compito
amax=0;                     % accelerazione massima in caso di profilo trapezoi-
dale
vmax=2;                     % velocità massima in caso di profilo trapezoidale
L=1;                        % distanza tra i due punti p0 e pf
check=1;                    % variabile di appoggio per l'utilizzo della funzio-
ne test in fondo
ngrad=0;                    % contatore relativo ai passi degli algoritmi cine-
matici differenziali lontano
% da zone di singolarità con il metodo del Gradiente
```

```
nnewt=0;                    % contatore relativo ai passi degli algoritmi cine-
matici differenziali lontano
% da zone di singolarità con il metodo di Newton
sing=0;                     % contatore relativo ai passi degli algoritmi cine-
matici differenziali in zone
% di singolarità
Id=eye(3);                  % matrice identità
sp=0;                       % inizializza la velocità imposta nello spazio car-
tesiano

warning off MATLAB:HandleGraphics:SupersededProperty:AspectRatio    % accorgi-
mento per la grafica

% presentazione

disp(' '); disp(' '); disp(' '); disp(' ');
disp('                      S   A   L   V   E !   ');
disp('                    ---------------------- ');
disp(' '); disp(' ');
disp('Benvenuto nella simulazione preparata da Andrea Aleni ,Dario Antonacci e
Giulio Pisani.');
disp(' ');
disp('Questo programma fa muovere un manipolatore antropomorfo da un');
disp('punto ad un altro secondo modalità e parametri specificati');
disp(' ');
disp('In qualsiasi momento è possibile premere <CTRL+C> o <CTRL+Pausa> per fer-
marmi!');
disp(' ');
disp(' ');
run dati3R;
for i=1:8
    disp(' ');
end
disp('Inserire stato iniziale e finale');
disp('del manipolatore per la simulazione.');
disp(' ');
% inserimento p0
check=1;
while check==1
    disp(' ');
    disp('inserire le coordinate del punto iniziale nel formato [x1 y1 z1] ');
    p0=input('p0= ');
    s13=size(p0)~=([1 3]);
    s31=size(p0)~=([3 1]);
    if (s13(1)|s13(2))&(s31(1)|s31(2))
        disp('attenzione: deve essere inserito un vettore di tre componenti');
    end
    while (s13(1)|s13(2))&(s31(1)|s31(2))
        p0=input('reinserire p0 ');
        s13=size(p0)~=([1 3]);
        s31=size(p0)~=([3 1]);
    end
    s13=size(p0)~=([1 3]);
    s31=size(p0)~=([3 1]);
    if s31(1)&s31(2)
        p0=p0';
```

```
    end
    check=test(p0);
    if check==1
        disp('attenzione: punto specificato al di fuori dello spazio di lavo-
ro!')
    end
end
% inserimento pf
pf=p0;
while pf==p0
    check=1;
    while check==1
        disp(' ');
        disp('inserire le coordinate del punto finale');
        pf=input('pf= ');
        s13=size(pf)~=([1 3]);
        s31=size(pf)~=([3 1]);
        if (s13(1)|s13(2))&(s31(1)|s31(2))
            disp('attenzione: deve essere inserito un vettore di tre componen-
ti');
        end
        while (s13(1)|s13(2))&(s31(1)|s31(2))
            pf=input('reinserire pf ');
            s13=size(pf)~=([1 3]);
            s31=size(pf)~=([3 1]);
        end
        s13=size(pf)~=([1 3]);
        s31=size(pf)~=([3 1]);
        if s31(1)&s31(2)
            pf=pf';
        end
        check=test(pf);
        if check==1
            disp('attenzione: punto specificato al di fuori dello spazio di la-
voro!')
        end
    end
    if pf==p0
        disp('attenzione: il punto specificato è uguale al punto di partenza!')
    end
end
% inserimento tempo
disp('in quanto tempo (in secondi) vuoi che il manipolatore svolga il compi-
to?');
while (time<=0) | (imag(time)~=0)
    time=input('time= ');
    if (time<=0) | (imag(time)~=0)
        disp ('inserire un valore reale positivo per il tempo');
    end
end
disp(' '); disp (' ');
L=dist2p(p0,pf);
disp('Inserire valore dell accelerazione massima');

amax=input('amax= ');
if amax==0
```

```
    disp('attenzione: amax deve essere diverso da 0');
end
while amax==0
    amax=input('amax= ');
end
vmax=(-time+(sqrt(time^2-4*(L/amax))))/(-2/amax);
if ((imag(vmax))~=0) | (L<((vmax^2)/amax))
    disp(' ');
    disp('attenzione: soluzione complessa per vmax');
    disp('oppure accelerazione troppo bassa');
    disp('con le condizioni fornite non è possibile realizzare il profilo');
    disp('trapezoidale entro il tempo desiderato: cambiare amax');
end
while ((imag(vmax))~=0) | (L<((vmax^2)/amax))
    amax=input('amax= ');
    vmax=(-time+(sqrt(time^2-4*(L/amax))))/(-2/amax);
    if ((imag(vmax))~=0) | (L<((vmax^2)/amax))
        disp('attenzione: soluzione complessa per vmax');
        disp('oppure accelerazione troppo bassa');
        disp('con le condizioni fornite non è possibile realizzare il profilo');
        disp('trapezoidale entro il tempo desiderato: cambiare amax');
    end
end
disp('Dati caricati e verificati: il Robot si muoverà da posizione iniziale a
posizione finale');
disp('secondo la legge di velocità specificata entro il tempo desiderato.');
disp('Verranno peraltro trattate le singolarità, dunque si farà utilizzo de-
gli');
disp('algoritmi di inversione cinematica differenziale (numerici) basate su una
serie');
disp('di parametri consigliati che è possibile impostare manualmente,')
disp('sebbene piccoli cambiamenti possano compromettere tutta la simulazione.');
disp(' ');
disp('I parametri sono i seguenti:');
disp('  eps = misura entro la quale si considera di stare in zona di singolari-
tà');
disp('        (con riferimento alla tesina è la distanza cartesiana da una sin-
golarità)');
disp('  lmax= valore massimo di correzione del fattore lambda in zona di singo-
larità');
disp('        (con riferimento alla tesina è il vertice del profilo triangola-
re)');
disp('  K=    valore della matrice diagonale di correzione dello Jacobiano in
inversione');
disp('        cinematica quando si utilizza il metodo di Newton;');
disp(' ');
disp('  Ks=   valore della matrice diagonale di correzione dello Jacobiano in
inversione');
disp('        cinematica quando si utilizza il metodo dell''inversa robusta;');
disp(' ');
disp('  alpha=valore della matrice diagonale di correzione dello Jacobiano in
inversione');
disp('        cinematica quando si utilizza il metodo del Gradiente');
disp(' ');
disp('  dTN=  passo di campionamento relativo al metodo di Newton');
disp(' ');
```

```
disp('  dTG=  passo di campionamento relativo al metodo del Gradiente');
disp(' ');
disp('  dTNS= passo di campionamento relativo al metodo dell''inversione ai mi-
nimi quadrati.');
disp(' ');
run paramdiff;
[message,vettorenorme,q0,qf]=invsceltanormaminima(p0,pf,accuracy);
qd=q0;
cartetrapezio;
disp(' ');
```

• cindir.m

```
% Calcola la cinematica diretta del manipolatore antropomorfo,
% con in ingresso il vettore di configurazione dei giunti e restituisce
% il vettore posizione dell'origine della terna solidale all'end-effector


function [p]=cindir(q)

global l1 l2 l3

 % Tabella dei parametri di Denavit-Hartenberg robot
 a1=0;          d1=0;          tw1=pi/3;     th1=q(1);
 a2=l2;         d2=0;          tw2=0;        th2=q(2);
 a3=l3;         d3=0;          tw3=0;        th3=q(3);

% posizioni comode per scrivere la matrice completa
    c1=cos(q(1));
    c2=cos(q(2));
    c23=cos(q(2)+q(3));
    s1=sin(q(1));
    s2=sin(q(2));
    s23=sin(q(2)+q(3));


Tdir=   [   c1*c23      -c1*s23     s1      c1*(a2*c2+a3*c23)   ;
            s1*c23      -s1*s23     -c1     s1*(a2*c2+a3*c23)   ;
            s23         c23         0       a2*s2+a3*s23        ;
            0           0           0       1                   ];

p=[Tdir(1,4) Tdir(2,4) Tdir(3,4)]';
```

• cindiffinv.m

```
% cindiffinv: è una funzione real time che calcola, secondo lo scorrere del
clock di simulink (implicito)

% la cinematica differenziale secondo le equazioni e i metodi numerici adottati
nella relazione.
```

```
% ad ogni passo di campionamento dell'ambiente Simulink avviene il seguente per-
corso:
%
% la funzione calcola lo jacobiano del manipolatore in base alla configurazione
dei giunti più recente;
% calcola poi con la cinematica diretta la posizione attuale del manipolatore in
base a q del passo precedente
% passo per passo; questo gli servirà per applicare gli algoritmi del gradiente
e di Newton.

function [qd]= cindiffinv(point)
global l1 l2 l3 pf q0 qf eps lmax sing nnewt ngrad K Ks alpha Id qd sp dTN dTG
dTNS;

Lq=dist2p(q0,qf);   % distanza nello spazio dei giunti tra configurazione ini-
ziale e configurazione finale
sceltaerr=Lq/0.5;   % questo è un parametro, funzione della suddetta distanza,
che serve a scegliere quando
                    % va usato il gradiente, e quando l'algoritmo di Newton (se
il robot è fuori da singolarità)
Jact=Jac(qd);       % calcolo dello Jacobiano nella configurazione attuale qd
p=cindir(qd);       % calcolo della posizione attuale in base all'ultima confi-
gurazione calcolata (inizia il loop)

if test(point)
    error('End effector oltre i confini dello spazio di lavoro! Rivedere la pia-
nificazione: quasi certamente perchè l2<>l3')
end

a1=l1;
a2=l2;
a3=l3;

PWx=p(1);
PWy=p(2);
PWz=p(3);

distz0=sqrt(PWx^2+PWy^2);       % distanza dall'asse delle singorità di spalla
z0
distconfest=abs((a2+a3)-sqrt(PWx^2+PWy^2+(abs(PWz)^2)));    % distanza dal con-
fine esterno dello spazio di lavoro
if l2~=l3
distconfint=sqrt(PWx^2+PWy^2+PWz^2)-(a2-a3);   % distanza dal confine interno
dello spazio di lavoro (solo se l2<>l3)
end

                                % ANALISI DELLE SINGOLARITA'
if ((sqrt(PWx^2+PWy^2)<=eps)... % di spalla: il centro del polso è giace sul-
l'asse z0 -> infinite singolarità
    | (abs((a2+a3)-sqrt(PWx^2+PWy^2+(abs(PWz)^2)))<=eps))... % gomito tutto ste-
so: polso ai confini esterni
    | (a3<a2 & (sqrt(PWx^2+PWy^2+PWz^2)-(a2-a3)<=eps)) | (a3>a2 &
(sqrt(PWx^2+PWy^2+PWz^2)-(a3-a2)<=eps))
      % gomito tutto ripiegato con l2 diverso da l3: singolarità nel confine in-
terno dello spazio di lavoro
```

```
    % viene qui scelto l'epsilon per il profilo triangolare di correzione del-
l'inversa
    if (distz0<=eps)
        distsing=distz0;
    elseif (distconfest<=eps)
        distsing=distconfest;
    elseif (distconfint<=eps)
        distsing=distconfint;
    end

     Js=Jact'*(inv((Jact*Jact')+((lam(distsing,eps,lmax)))*Id)); % INVERSA AI
MINIMI QUADRATI SMORZATI
     qd=qd+(Js*(sp+(Ks*(point-p))))*dTNS; % aggiorna la configurazione dei giun-
ti attuale (algoritmo completo)
     sing=sing+1; % conta i passi in singolarità
 elseif dist2p(p,pf)>=sceltaerr
     qd=qd+(alpha*(Jact')*(point-p))*dTG;  %METODO DEL GRADIENTE
     ngrad=ngrad+1; % conta i passi lontano da singolarità quando è usato il
gradiente
 elseif  dist2p(p,pf)<sceltaerr
     qd=qd+(inv(Jact)*(sp+(K*(point-p)))*dTN);    % METODO DI NEWTON
     nnewt=nnewt+1; % conta i passi lontano da singolarità quando è usato Newton
 end
```

• dati3R.m

```
% Questo è il file dove l'utente scrive i dati del manipolatore per
% utilizzarli nelle simulazioni;
%
% E' stata scelta la convenzione di Denavit Hartenberg%


global l1 l2 l3; % questi sono i parametri che vengono passati in comune alle
global accuracy; % varie funzioni utilizzate nella simulazione

% lunghezze dei 3 bracci:

l1=10;
l2=10;
l3=10;



accuracy=12;
             % è il valore di accuratezza con cui sono rappresentati internamen-
te gli angoli per le inversioni.

             % CONTROLLI SUL FORMATO NUMERICO

s11=size(l1)~=([1 1]);
if (s11(1)| s11(2))
```

```
    error('l1 deve essere un semplice scalare (una sola componente): correggere
''dati3R.m''');
end
s11=size(l2)~=([1 1]);
if (s11(1)| s11(2))
    error('l2 deve essere un semplice scalare (una sola componente): correggere
''dati3R.m''');
end
s11=size(l3)~=([1 1]);
if (s11(1)| s11(2))
    error('l3 deve essere un semplice scalare (una sola componente): correggere
''dati3R.m''');
end
s11=size(accuracy)~=([1 1]);
if (s11(1)| s11(2))
    error('accuracy deve essere un semplice scalare (una sola componente): cor-
reggere ''dati3R.m''');
end

% CONTROLLI SUL VALORE NUMERICO

if l1<=0
    error('è stato specificato il 1° braccio di lunghezza minore o uguale a 0:
correggere ''dati3R.m''');
end
if l2<=0
    error('è stato specificato il 2° braccio di lunghezza minore o uguale a 0:
correggere ''dati3R.m''');
end
if l3<=0
    error('è stato specificato il 3° braccio di lunghezza minore o uguale a 0:
correggere ''dati3R.m''');
end
if accuracy<1 | (accuracy-fix(accuracy)~=0)
    error('il parametro accuracy ha un valore negativo o non intero: correggere
''dati3R.m''');
end
clear s11;
```

• dist2p.m

```
% dist2p = calcola la distanza tra due punti in uno spazio a tre dimensioni
(norma euclidea)

function [d]=dist2p(p1,p2)

d=sqrt(((p1(1)-p2(1))^2)+((p1(2)-p2(2))^2)+((p1(3)-p2(3))^2));
```

• invanalitica.m

```
% Inversa analitica: data una posizione per il centro dell'end-effector del Ro-
bot
% analizza l'esistenza di soluzioni e le calcola quando esistono. Se il manipo-
latore
% si trova in singolarità viene specificato il tipo di singolarità e vengono
% calcolate, tra le infinite soluzioni, tante a seconda della specifica di pre-
cisione
% in ingresso; il motivo è che faranno comodo per la pianificazione
% di traiettorie avvicinarsi in tutte le situazioni possibili di partenza e ar-
rivo
% quelle a che rendono migliore il moto sotto vari aspetti (ad esempio minimiz-
zano
% gli spostamenti dei giunti), anche qualora il punto di partenza e/o arrivo
% fossero in singolarità.
% Ingressi: 2
%               p = posizione cartesiana del centro del polso
%               accuracy=livello di precisione con cui esplorare le possibili
soluzioni in particolari
%Uscite:   5
%               msg = è un messaggio che può essere indirizzato all'utente, in-
dica a parole la situazione
%                     analizzata dalla funzione relativamente al punto e anche
il tipo di singolarità;
%                 M = Se esistono soluzioni in forma chiusa sono 2 doppie o 4 e
sono riportate a 4 righe (matrice 4x3).
%                 insing = variabile booleana. Vale TRUE se il manipolatore si
trova in singolarità e FALSE altrimenti;
%               sit = è uno scalare che vale 1 se la singolarità è lungo l'asse
z0, 2 se è di gomito steso,
%                     3 se il punto si trova sul confine interno dello spazio di
lavoro (solo nel caso l2<>l3);
%                     in tutti gli altri casi è posto a -1.
%               nsol = numero di soluzioni trovate, può essere 0 (oltre WS), op-
pure oo (singolarità piantone/origine)
%                      2 per le singolarità ai confini esterni dello spazio di
lavoro, 4 nel WS o ai confini interni.

function [msg,M,insing,sit,nsol]=invanalitica(p,accuracy)

global l1 l2 l3;

a1=l1;          % i parametri l1 l2 l3 sono le lunghezza dei 3 bracci del robot
a2=l2;
a3=l3;

PWx=p(1);       % proiezioni del punto p sull'origine del sistema di riferimento
PWy=p(2);
PWz=p(3);

if nargin==1    % se non viene fornito in ingresso accuracy è utilizzato questo
valore di default
    accuracy=12;
end             %
check=test(p);  % richiamo la funzione test che fornisce 1 in caso il punto si
trovi fuori dello spazio di lavoro
if check==1     % 0 altrimenti
```

```
    nsol=0;
    insing=false;
    M=[];
    sit=-1;
    msg=('Attenzione: punto specificato al di fuori dello spazio di lavoro!');

else

       if ((sqrt(PWx^2+PWy^2)==0)... % di spalla: il centro del polso è giace
sull'asse z0 -> infinite singolarità
            | (abs((a2+a3)-sqrt(PWx^2+PWy^2+(abs(PWz)^2)))==0))... % gomito tut-
to steso: polso ai confini esterni
            | (a3<a2 & (sqrt(PWx^2+PWy^2+PWz^2)-(a2-a3)==0)) | (a3>a2 &
(sqrt(PWx^2+PWy^2+PWz^2)-(a3-a2)==0))
        % gomito tutto ripiegato con l2 diverso da l3: singolarità nel confine
interno dello spazio di lavoro
        insing=true;             % Viene comunicato con l'uscita "sing" che il
manipolatore è in singolarità
        msg='Il manipolatore si trova in singolarità';  % le funzioni esterne
possono utilizzare questo messaggio
        % che viene restituito come prima uscita della funzione



% COMUNICO IN USCITA IL TIPO DI SINGOLARITA' PER IL TRATTAMENTO DALL'ESTERNO

        if (sqrt(PWx^2+PWy^2)==0)     % se il punto sta sull'asse z0
            if PWx==0 & PWy==0 & PWz==0     % se inoltre sta nell'origine
                msg=[msg ' esattamente nell''origine: infinite configurazioni
ammissibili per theta1 e theta2'];
                nsol=Inf;
                M=gensolsingin0(p,accuracy);
                sit=0;
            else                          % altrimenti non è nell'origine ma sicu-
ramente su z0
                sit=1;
                msg=[msg (' lungo l''asse del piantone z0 (infinite soluzioni
per theta1)')];
                M=[];   % è inizializzata l'uscita M che sarà calcolata in base
alla sottofunzione gensolsingout0
                % saranno le funzioni esterne a scegliere come regolare i giunti
per la posa del robot sull'asse.
                nsol=Inf;                % viene comunicato all'esterno che ci
sono infinite soluzioni per theta 1
                M=gensolsingout0(p,accuracy);
                insing=true;
            end
        elseif  (abs((a2+a3)-sqrt(PWx^2+PWy^2+(abs(PWz)^2)))==0)...

                | (a3<a2 & (sqrt(PWx^2+PWy^2+PWz^2)-(a2-a3)==0)) | (a3>a2 &
(sqrt(PWx^2+PWy^2+PWz^2)-(a3-a2)==0))

            if (abs((a2+a3)-sqrt(PWx^2+PWy^2+(abs(PWz)^2)))==0)       % gomito
steso (confine esterno)
```

```
                msg=[msg ' con il gomito tutto steso ma fuori dall''asse z0: 2
coppie di soluzioni coincidenti'];
                nsol=2;
                sit=2;
            elseif (a3<a2 & (sqrt(PWx^2+PWy^2+PWz^2)-(a2-a3)==0)) |...
                    (a3>a2 & (sqrt(PWx^2+PWy^2+PWz^2)-(a3-a2)==0))    % gomito
ripiegato (non nell'origine: l2<>l3)
                msg=[msg ' sul confine interno dello spazio di lavoro, non es-
sendo l2=l3: 4 soluzioni'...
                    ' distinte in forma chiusa'];
                nsol=4;
                sit=3;
            end

            % IN ENTRAMBI I CASI DI SINGOLARITA' DI GOMITO CALCOLO 4 SOLUZIONI,
ANCHE SE NEL CASO
            % DI GOMITO STESO (p non su z0) DUE RIGHE SARANNO DIPENDENTI

            c3=(PWx^2+PWy^2+PWz^2-a2^2-a3^2)/(2*a2*a3);
            s3=[sqrt(1-c3^2); -sqrt(1-c3^2)];           % entrambe le soluzioni
per s3 che è il seno di theta3
            c2=((a2+a3*c3)*sqrt(PWx^2+PWy^2)+a3*s3*PWz)/(PWx^2+PWy^2+PWz^2); %
c2 è funzione di s3 --> darà 2 sol;
            s2=((a2+a3*c3)*PWz-a3*s3*sqrt(PWx^2+PWy^2))/(PWx^2+PWy^2+PWz^2);

            th1=[atan2(PWy,PWx); pi+atan2(PWy,PWx)];
            th2=[atan2(s2,c2)'; pi-atan2(s2,c2)'];
            th3=[atan2(s3(1),c3); atan2(s3(2),c3)];


            M=[ th1(1)  th2(1,1)  th3(1);       % spalla destra     gomito ste-
so--> gomito basso=gomito alto
                th1(1)  th2(1,2)  th3(2);       % spalla destra     gomito ste-
so--> gomito alto=gomito basso
                th1(2)  th2(2,1)  th3(2);       % spalla sinistra   gomito ste-
so--> gomito basso=gomito alto
                th1(2)  th2(2,2)  th3(1)];      % spalla sinistra   gomito ste-
so--> gomito alto=gomito basso

            insing=true; % comunica in uscita che il robot si trova in singola-
rità
        end


    else

        % RICERCO LE QUATTRO SOLUZIONI DISTINTE IN FORMA CHIUSA

        c3=(PWx^2+PWy^2+PWz^2-a2^2-a3^2)/(2*a2*a3);
        s3=[sqrt(1-c3^2); -sqrt(1-c3^2)];
        c2=((a2+a3*c3)*sqrt(PWx^2+PWy^2)+a3*s3*PWz)/(PWx^2+PWy^2+PWz^2);
        s2=((a2+a3*c3)*PWz-a3*s3*sqrt(PWx^2+PWy^2))/(PWx^2+PWy^2+PWz^2);

        th1=[atan2(PWy,PWx); pi+atan2(PWy,PWx)];
        th2=[atan2(s2,c2)'; pi-atan2(s2,c2)'];
```

```
        th3=[atan2(s3(1),c3); atan2(s3(2),c3)];


        M=[ th1(1)  th2(1,1)  th3(1);       % spalla destra     gomito basso
            th1(1)  th2(1,2)  th3(2);       % spalla destra     gomito alto
            th1(2)  th2(2,1)  th3(2);       % spalla sinistra   gomito basso
            th1(2)  th2(2,2)  th3(1)];      % spalla sinistra   gomito alto

        nsol=4;
        insing=false;                                % comunica in uscita che il
robot non si trova in singolarità
        msg=('4 soluzioni distinte in forma chiusa');
        sit=-1;

    end
end

% SEGUONO LE DUE SOTTOFUNZIONI UTILIZZATE PER GENERARE LA MATRICE DELLE SOLUZIO-
NI IN CONDIZIONI DI SINGOLARITA'
% 1a funzione: utilizzata per le singolarità di asse z0 ma non nell'origine
function M=gensolsingout0(p,accuracy)
accuracy=accuracy*2;
global l1 l2 l3
Pz=p(3);
M=zeros(accuracy,3);
if Pz>0
    p1th1init=0;
    p1th2init=pi/2-acos((l2^2+Pz^2-l3^2)/(2*l2*Pz));
    p1th3init=pi-acos((l2^2+l3^2-Pz^2)/(2*l2*l3));
elseif Pz<0
    p1th1init=0;
    p1th2init=3*pi/2-acos((Pz^2+l2^2-l3^2)/(2*abs(Pz)*l2));
    p1th3init=pi-acos((l2^2+l3^2-Pz^2)/(2*l2*l3));
end
th1=p1th1init;
th2=p1th2init;
th3=p1th3init;
for i=1:accuracy
    M(i,:)=[th1 th2 th3];
    th1=th1+2*pi/accuracy;
end

%2a funzione: utilizzata per le singolarità nell'origine qualora l2=l3
function M=gensolsingin0(p,accuracy)
global l1 l2 l3
M=zeros(2*accuracy,3*(accuracy+1));
th1=0;
th2=-pi/2;
th3=pi;
k=0;
for j=1:accuracy+1
    for i=1:2*accuracy
        sol(i,1:3)=[th1 th2 th3];
        th1=th1+pi/accuracy;
    end
    th1=0;
```

```
    M(1:2*accuracy,k+1:k+3)=sol;
    k=j*3;
    th2=th2+pi/accuracy;
end
```

invsceltanormaminima.m

```
% Funzione invsceltanormaminimaS
% Serve a migliorare la scelta per l'inizializzazione del robot
% in quanto trova tra le soluzioni (che possono essere infinite in singolarità)
% per i due punti, fornite dalla invanalitica attraverso il main, quelle miglio-
ri
% secondo il criterio che minimizza la distanza delle due configurazioni corri-
spondenti.
% Per fare ciò prende in ingresso p0 e pf, ed inoltre accuracy che è il valore
di precisione
% da passare alla funzione
%4 uscite:
%   message = è una stringa di testo che riporta a parole ciò che la funzione
analizza
%             riguardo a esistenza, numero, tipo di soluzioni. In taluni casi
esso è
%             indirizzato dal 'main'
%   vettorenorme = è un vettore di dimensione variabile a seconda del tipo e del
numero
%                  di soluzioni trovate per i due punti p0 e pf, e riporta
%                  tutte le distanze nello spazio dei giunti relative alle cop-
pie possibili.
%                  chiaramente la continuità delle infinite soluzioni in alcune
singolarità
%                  è discretizzata proprio attraverso il parametro accuracy
%   q0,qf = sono le due configurazioni "migliori" calcolate alla fine.

function [message,vettorenorme,q0,qf]=invsceltanormaminima(p0,pf,accuracy)

if nargin==2
    accuracy=12;
end;
[msg0,M0,insing0,sit0,nsol0]=invanalitica(p0,accuracy);
[msgf,Mf,insingf,sitf,nsolf]=invanalitica(pf,accuracy);

if p0==pf

    error('Attenzione! punto iniziale e finale coincidono: reimmettere i pun-
ti');
end

if nsol0==0 | nsolf==0  % se almeno un punto fuori dallo spazio di lavoro la
funzione non è definita
```

```
    message='non si possono scegliere le configurazioni relative ai due punti
perchè';
    if nsol0==0
        if nsolf==0
            message=[message ' entrambi i punti sono al di fuori dello spazio di
lavoro.'];
        else
            message=[message ' il primo punto è al di fuori dello spazio di la-
voro.'];
        end
    else
        message=[message ' il secondo punto è al di fuori dello spazio di lavo-
ro.'];
    end
    vettorenorme=[];
    q0=[];
    qf=[];
else
    if sit0==2 & sitf==2    %2,2 steso + steso
        message='entrambi i punti ammettono 2 coppie di soluzioni coincidenti in
forma chiusa.';

        % scelta della combinazione migliore (o di una tra le migliori) per le
configurazioni inziale e finale dei giunti.
        % bisogna mappare tutte le combinazioni (4) e trovare la coppia (o le
coppie) a norma minima;

        vettorenorme=zeros(4,1);

        k=0;
        for i=1:4
            for j=1:4
                if ~(i==1 | i==3 | j==1 | j==3)
                    k=k+1;
                    vettorenorme(k)=dist2p(M0(i,:),Mf(j,:));
                end
            end
        end
        k=0;
        for i=1:4
            for j=1:4
                if ~(i==1 | i==3 | j==1 | j==3)
                    k=k+1;
                    if vettorenorme(k)==min(vettorenorme)
                        q0=(M0(i,:))'; qf=(Mf(j,:))';
                    end
                end
            end
        end
    elseif (sit0==2 & nsolf==4) | (nsol0==4 & sitf==2)    %2,4      steso + nor-
male
        if sitf==2
            message='Il primo punto ammette 4 soluzioni distinte, il secondo due
coppie di soluzioni coincidenti';
            Mtmp=M0;
```

```
            M0=Mf;
            Mf=Mtmp;
        elseif sit0==2
            message='Il primo punto ammette due coppie di soluzioni coincidenti,
il secondo 4 distinte';
        end

        % scelta della combinazione migliore (o di una tra le migliori) per le
configurazioni inziale e finale dei giunti.
        % bisogna mappare tutte le combinazioni (8) e trovare la coppia (o le
coppie) a norma minima;

        vettorenorme=zeros(8,1);

        k=0;
        for i=1:4
            for j=1:4
                if ~(i==1 | i==3)
                    k=k+1;
                    vettorenorme(k)=dist2p(M0(i,:),Mf(j,:));
                end
            end
        end
        k=0;
        for i=1:4
            for j=1:4
                if ~(i==1 | i==3)
                    k=k+1;
                    if vettorenorme(k)==min(vettorenorme)
                        q0=(M0(i,:))'; qf=(Mf(j,:))';
                    end
                end
            end
        end

        if sitf==2
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end
    elseif nsol0==4 & nsolf==4 %4,4     normale + normale

        message='entrambi i punti ammettono 4 soluzioni in forma chiusa';

        % scelta della combinazione migliore (o di una tra le migliori) per le
configurazioni inziale e finale dei giunti.
        % bisogna mappare tutte le combinazioni (16) e trovare la coppia (o le
coppie) a norma minima;
        % le norme sono calcolate nello spazio dei giunti per tentare di mini-
mizzare globalmente le variazioni per i giunti;
        % si è scelto di adottare la norma euclidea.

        vettorenorme=zeros(16,1);

        k=0;
        for i=1:4
```

```
            for j=1:4
                k=k+1;
                vettorenorme(k)=dist2p(M0(i,:),Mf(j,:));
            end
        end
        k=0;
        for i=1:4
            for j=1:4
                k=k+1;
                if vettorenorme(k)==min(vettorenorme)
                    q0=(M0(i,:))'; qf=(Mf(j,:))';
                end
            end
        end
    elseif (sit0==2 & sitf==1) | (sitf==2 & sit0==1) %2,inf     steso + z0
        if sitf==2
            message='Il primo punto ammette oo^1 soluzioni, il secondo due cop-
pie di soluzioni coincidenti';
            Mtmp=M0;
            M0=Mf;
            Mf=Mtmp;
        elseif sit0==2
            message='Il primo punto ammette due coppie di soluzioni coincidenti,
il secondo oo^1 soluzioni';
        end

        vettorenorme=zeros(2*accuracy*2,1);
        k=0;
        for i=1:2
            for j=1:2*accuracy
                    k=k+1;
                    vettorenorme(k,:)=dist2p(M0(i,:),Mf(j,:));
            end
        end
        k=0;
        for i=1:2
            for j=1:2*accuracy
                    k=k+1;
                    if vettorenorme(k,:)==min(vettorenorme)
                        q0=(M0(i,:))'; qf=(Mf(j,:))';
                    end
            end
        end

        if sitf==2
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end

    elseif (nsol0==4 & sitf==1) | (nsolf==4 & sit0==1) %4,Inf       normale + z0
        if nsolf==4
            message='il primo punto ammette oo^1 soluzioni, il secondo 4 solu-
zioni distinte in forma chiusa';
            Mtmp=M0;
            M0=Mf;
```

```
            Mf=Mtmp;
        elseif nsol0==4
            message='Il primo punto ammette 4 soluzioni distinte in forma chiu-
sa, il secondo oo^1 soluzioni';
        end

        vettorenorme=zeros(4*accuracy*2,1);
        k=0;
        for i=1:4
            for j=1:2*accuracy
                    k=k+1;
                    vettorenorme(k,:)=dist2p(M0(i,:),Mf(j,:));
            end
        end
        k=0;
        for i=1:4
            for j=1:2*accuracy
                    k=k+1;
                    if vettorenorme(k,:)==min(vettorenorme)
                        q0=(M0(i,:))'; qf=(Mf(j,:))';
                    end
            end
        end

        if nsolf==4
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end
    elseif (sit0==2 & sitf==0) | (sitf==2 & sit0==0) %2,inf^2       steso + ori-
gine
        if sitf==2
            message='Il primo punto ammette oo^2 soluzioni, il secondo due cop-
pie di soluzioni coincidenti';
            Mtmp=M0;
            M0=Mf;
            Mf=Mtmp;
        elseif sit0==2
            message='Il primo punto ammette due coppie di soluzioni coincidenti,
il secondo oo^2';
        end

        % scelta della combinazione migliore (o 1 tra le migliori) per le confi-
gurazioni inziale e finale dei giunti.
        % bisogna mappare tutte le combinazioni trovare la coppia (o le coppie)
a norma minima;

        nsolapprox=(accuracy+1)*2*accuracy;
        vettorenorme=zeros(2*nsolapprox,1);

        k=0;
        m=1;
        for l=2:3
            for j=1:accuracy+1
                for i=1:2*accuracy
                    vettorenorme(m)=dist2p(M0(l,1:3),Mf(i,k+1:k+3));
```

```
                    m=m+1;
                end
                k=j*3;
            end
            k=0;
        end
        k=0;
        m=1;
        for l=2:3
            for j=1:accuracy+1
                for i=1:2*accuracy
                    if vettorenorme(m)==min(vettorenorme)
                        q0=M0(l,1:3)'; qf=Mf(i,k+1:k+3)';
                    end
                    m=m+1;
                end
                k=j*3;
            end
            k=0;
        end

        if sitf==2
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end
    elseif (nsol0==4 & sitf==0) | (sit0==0 & nsolf==4) %4,Inf^2     normale +
origine
        if sit0==0
            message='Il primo punto ammette oo^2 soluzioni, il secondo 4 solu-
zioni distinte in forma chiusa';
            Mtmp=M0;
            M0=Mf;
            Mf=Mtmp;
        elseif nsol0==4
            message='Il primo punto ammette 4 soluzioni distinte in forma chiu-
sa, il secondo oo^2 soluzioni';
        end

        % scelta della combinazione migliore (o 1 tra le migliori) per le confi-
gurazioni inziale e finale dei giunti.
        % bisogna mappare tutte le combinazioni trovare la coppia (o le coppie)
a norma minima;

        nsolapprox=(accuracy+1)*2*accuracy;
        vettorenorme=zeros(4*nsolapprox,1);

        k=0;
        m=1;
        for l=1:4
            for j=1:accuracy+1
                for i=1:2*accuracy
                    vettorenorme(m)=dist2p(M0(l,1:3),Mf(i,k+1:k+3));
                    m=m+1;
                end
                k=j*3;
```

```
            end
            k=0;
        end
        k=0;
        m=1;
        for l=1:4
            for j=1:accuracy+1
                for i=1:2*accuracy
                    if vettorenorme(m)==min(vettorenorme)
                        q0=M0(l,1:3)'; qf=Mf(i,k+1:k+3)';
                    end
                    m=m+1;
                end
                k=j*3;
            end
            k=0;
        end

        if sit0==0
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end
    elseif (sit0==1 & sitf==1) %Inf,Inf     z0 + z0
            message='entrambi i punti ammettono oo^1 soluzioni';

        vettorenorme=zeros(4*accuracy^2,1);
        k=0;
        for i=1:2*accuracy
            for j=1:2*accuracy
                    k=k+1;
                    vettorenorme(k,:)=dist2p(M0(i,:),Mf(j,:));
            end
        end
        k=0;
        for i=1:2*accuracy
            for j=1:2*accuracy
                    k=k+1;
                    if vettorenorme(k,:)==min(vettorenorme)
                        q0=(M0(i,:))'; qf=(Mf(j,:))';
                    end
            end
        end
    elseif (sit0==1 & sitf==0) | (sitf==1 & sit0==0)  %Inf,Inf^2    z0 + origine
        if sit0==0
            message='Il primo punto ammette oo^2 soluzioni, il secondo oo^1';
            Mtmp=M0;
            M0=Mf;
            Mf=Mtmp;
        elseif sitf==0
            message='Il primo punto ammette oo^1 soluzioni, il secondo oo^2';
        end

        % scelta della combinazione migliore (o 1 tra le migliori) per le confi-
gurazioni inziale e finale dei giunti.
```

```
        % bisogna mappare tutte le combinazioni trovare la coppia (o le coppie)
a norma minima;

        nsolapprox=(accuracy+1)*2*accuracy;
        vettorenorme=zeros(2*accuracy*nsolapprox,1);

        k=0;
        m=1;
        for l=1:2*accuracy
            for j=1:accuracy+1
                for i=1:2*accuracy
                    vettorenorme(m)=dist2p(M0(l,1:3),Mf(i,k+1:k+3));
                    m=m+1;
                end
                k=j*3;
            end
            k=0;
        end
        k=0;
        m=1;
        for l=1:2*accuracy
            for j=1:accuracy+1
                for i=1:2*accuracy
                    if vettorenorme(m)==min(vettorenorme)
                        q0=M0(l,1:3)'; qf=Mf(i,k+1:k+3)';
                    end
                    m=m+1;
                end
                k=j*3;
            end
            k=0;
        end

        if sit0==0
            qtmp=q0;
            q0=qf;
            qf=qtmp;
        end
    end
end
```

• Jac.m

```
%   La funzione Jac calcola lo Jacobiano del robot revoluto che
%   risulta definito per una determinata configurazione di
%   giunti in ingresso q.

function J=Jac(q)
global l2 l3;

t1=q(1);
t2=q(2);
```

```
t3=q(3);

%a1=0; l1 è indifferente in quanto il riferimento è sul piantone e a1=0
a2=l2;
a3=l3;

J=[-sin(t1)*(a2*cos(t2)+a3*cos(t2+t3)) -cos(t1)*(a2*sin(t2)+a3*sin(t2+t3))
-a3*cos(t1)*sin(t2+t3);
   cos(t1)*(a2*cos(t2)+a3*cos(t2+t3))  -sin(t1)*(a2*sin(t2)+a3*sin(t2+t3))
-a3*sin(t1)*sin(t2+t3);
   0                                   a2*cos(t2)+a3*cos(t2+t3)
a3*cos(t2+t3)];
```

• lam.m

```
% Questa funzione costruisce un profilo triangolare che viene utilizzato
% nella cinematica differenziale per ammorbidire il calcolo dello
% Jacobiano e fare subentrare gradualmente l'inversione robusta
% quando il manipolatore va verso condizioni di singolarità

function [l]=lam(err,eps,lmax);

l=lmax*(1-err/eps);

% err è l'indice delle ascisse, variabile tra 0 ed eps;
% eps è il raggio entro cui si considera il robot in configurazione di singola-
rità;
% lmax è il valore massimo di correzione che si ha in piena singolarità.
% In uscita è restituito il valore di correzione dipendente da err.
```

paramdiff.m

```
% Questo file contiene i valori dei parametri utilizzati nei calcoli della cine-
matica differenziale.

global eps lmax K Ks alpha dTG dTN dTNS;

eps=0.1;
                % misura entro la quale si considera di stare in zona di singo-
larità

lmax=0.1;
                % valore massimo di correzione del fattore lambda in zona di
singolarità
K=150*Id;
                % valore della matrice diagonale di correzione dello Jacobiano
250
                % in inversione cinematica quando si utilizza il metodo di New-
ton;
                % tale valore è moltiplicando per l'identità 3x3.
```

```
Ks=250*Id;
                % valore della matrice diagonale di correzione dello Jacobiano 8
                % in inversione cinematica quando si utilizza il metodo dell'in-
versa robusta;
                % tale valore è moltiplicando per l'identità 3x3.
alpha=3*Id;
                % valore della matrice diagonale di correzione dello Jacobiano
10
                % in inversione cinematica quando si utilizza il metodo del gra-
diente;
                % tale valore è moltiplicando per l'identità 3x3.
dTN=0.005;
                % passo di campionamento relativo al metodo di Newton
dTNS=0.005;
                % passo di campionamento relativo al metodo dell'inversione ro-
busta
dTG=0.005;
                % passo di campionamento relativo al metodo del Gradiente
```

• percorso.m

```
function [point]=percorso(s)
global p0 pf time point;

norma=dist2p(p0,pf);
point=p0+(s*(pf-p0)/norma);
```

• test.m

```
% questa funzione controlla se il punto in ingresso

% si trova o meno all'interno dello spazio di lavoro
% del robot e pone l'uscita uguale a 1 in caso affermativo
% altrimenti a 0.

function controllapunto=test(p)
global l1 l2 l3;
controllapunto=0;

PWx=p(1);
PWy=p(2);
PWz=p(3);

if sqrt(PWx^2+PWy^2+PWz^2)>l2+l3 |... se il punto è troppo lontano dall'origine
e supera la somma dei bracci l2,l3
        (l3<l2 & (sqrt(PWx^2+PWy^2+PWz^2)<l2-l3)) | (l3>l2 &
(sqrt(PWx^2+PWy^2+PWz^2)<l3-l2)) % punto oltre il

% confine interno del WS (l2<>l3)
```

```
    controllapunto=1;
else
    controllapunto=0;
end;
```

• trapezio.m

```
% Realizza il profilo di velocità trapezoidale per la

% pianificazione di traiettorie nello spazio cartesiano.
% In ingresso prende "cont"=tempo dato dal clock in Simulink;
% in uscita sp=spazio percorso in ascissa derivato al tempo t=cont

function [sp]=trapezio(cont);

global vmax amax time sp

% time=tempo totale del profilo   Ts=tempo di salita     cont è la variabile
tempo

Ts=vmax/amax;                                % Intervallo di accelerazione/dece-
lerazione=Tempo di salita

if ((cont>=0) & (cont<=Ts))                  % FASE DI ACCELERAZIONE
    sp=amax*cont;          % derivata analitica di s=(amax*(cont^2))/2;
end

if ((cont>=Ts) & (cont<=(time-Ts)))          % FASE A VELOCITA' COSTANTE
    sp=vmax;               % derivata analitica di s=(vmax*cont)-((vmax^2)/(2*-
amax));
end

if ((cont>=(time-Ts)) & (cont<=time))        % FASE DI DECELERAZIONE
    sp=-amax*(cont-time); % derivata analitica di
s=-(amax*((cont-time)^2)/2)+(vmax*time)-((vmax^2)/amax);
end
```