Politecnico di Bari

A.A. 2018/19

Cinematica del manipolatore Kuka: modello e test

Tema d'anno di
«*Modeling of Discrete Event System*»

C. Perri
D. Bevilacqua

Prof.ssa Fanti
Prof. Mangini
Ing. Parisi

Problematica e risoluzione: introduzione

Problema: il manipolatore KUKA presente nei laboratori Fablab del Politecnico di Bari effettua operazioni di inversione cinematica non accessibili dall'esterno

↓

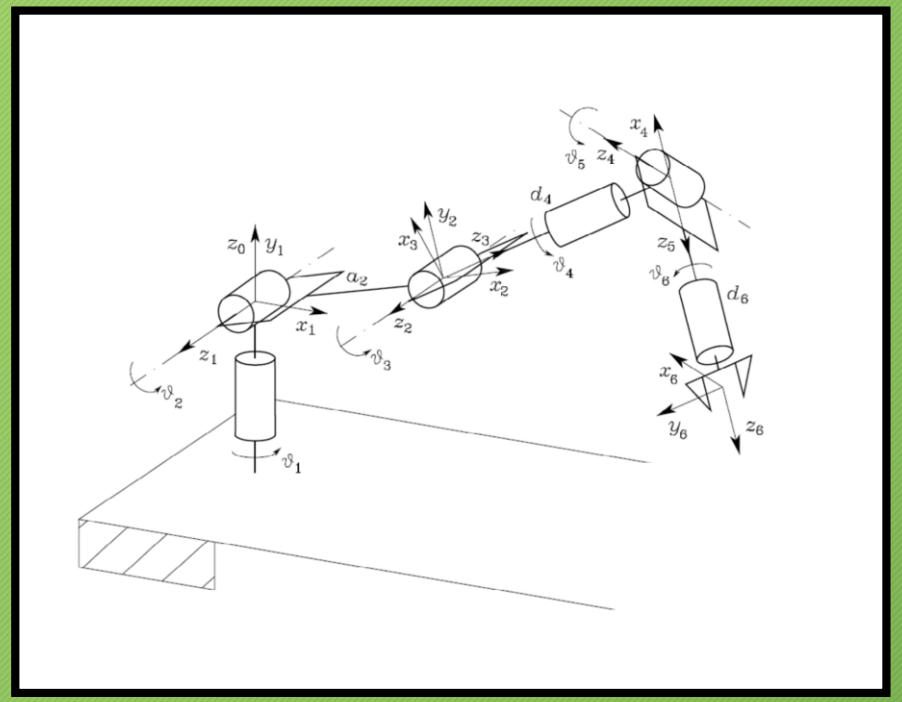
Procedimento: creare un modello matematico su Matlab, simularlo e eseguire test effettivi in laboratorio

↓

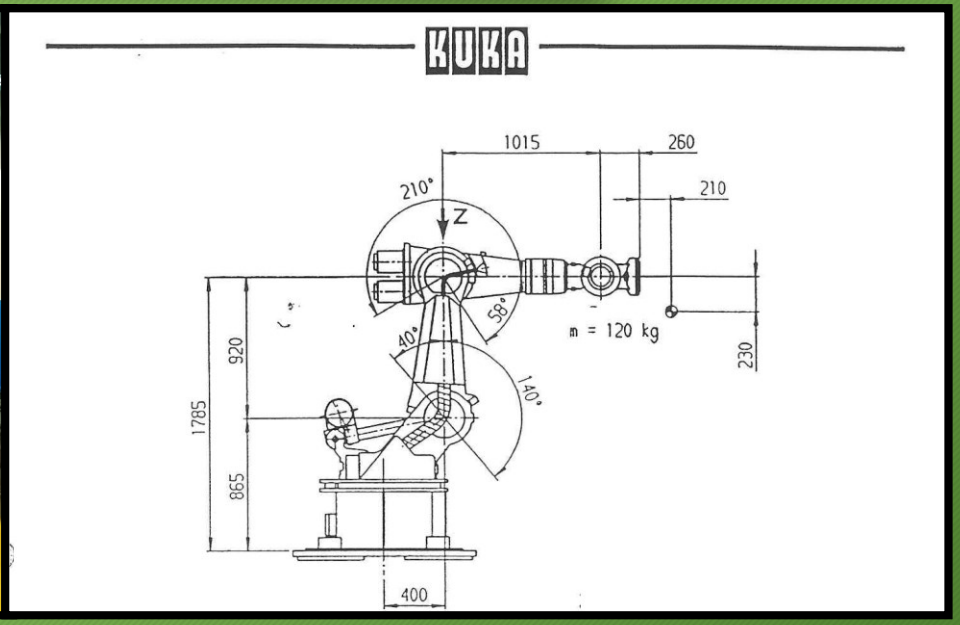
Scopo: creare una logica ben definita di calcolo di traiettoria per l'end-effector e implementarla sul manipolatore

Creazione del modello

Modello di un manipolatore antropomorfo con polso sferico



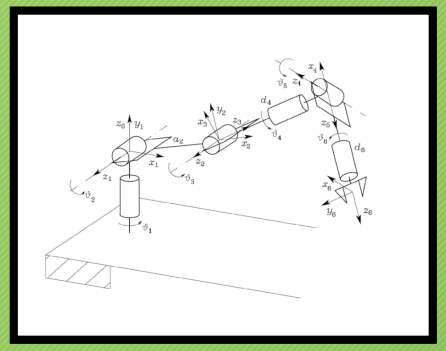
6 giunti -> 6 gradi di libertà



Dimensioni reali

Creazione del modello

Convenzione di Denavit-Hartenberg



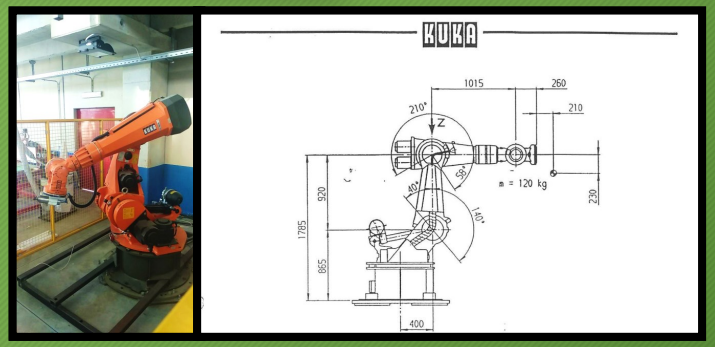
Modello di un manipolatore antropomorfo con polso sferico

Braccio	a_i	α_i	d_i	ϑ_i
1	0	$\pi/2$	0	ϑ_1
2	a_2	0	0	ϑ_2
3	0	$\pi/2$	0	ϑ_3
4	0	$-\pi/2$	d_4	ϑ_4
5	0	$\pi/2$	0	ϑ_5
6	0	0	d_6	ϑ_6

Orientamento bracci

Variabili di giunto

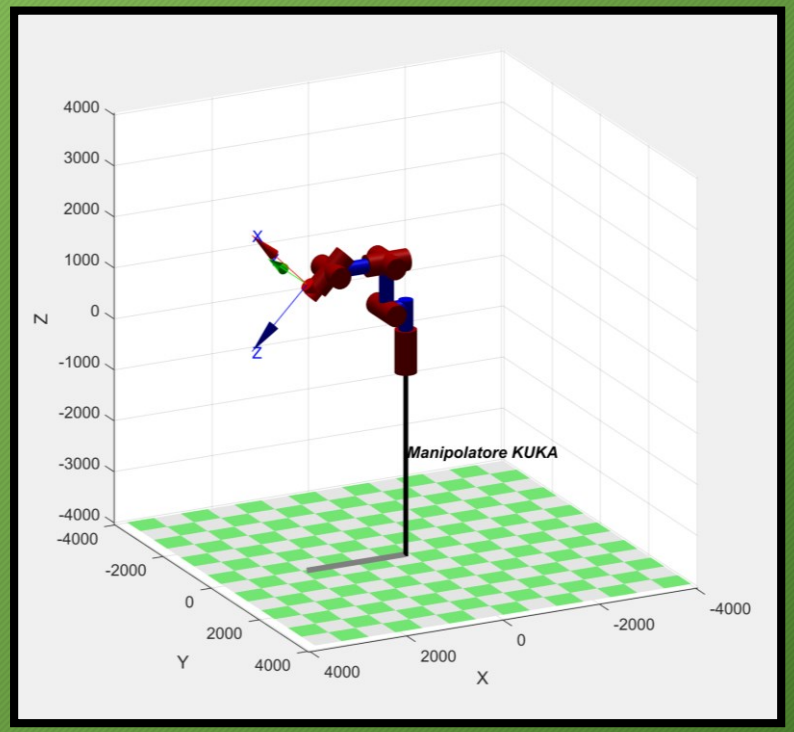
Dimensione bracci



Creazione del modello in Matlab

Robotics System Toolbox

```
%% INIZIALIZZAZIONE DEL MANIPOLATORE KUKA FABLAB

L(1) = Link([0 0.865*1e3 0.4*1e3 pi/2]);
L(2) = Link([0 0 0.92*1e3 0]);
L(3) = Link([0 0 0 pi/2]);
L(4) = Link([0 1.15*1e3 0 -pi/2]);
L(5) = Link([0 0 0 pi/2]);
L(6) = Link([0 0.470*1e3 0.23*1e3 0]);

KUKA = SerialLink(L, 'name', 'Manipolatore KUKA');

q0 = [0 90*pi/180 0 0 -45*pi/180 0];
T0 = KUKA.fkine(q0);
```



Con q0 si è riportata la condizione iniziale delle varibili di giunto

Cinematica inversa

Per il calcolo della cinematica inversa è stata creata un'opportuna funzione obbiettivo da minimizzare al fine di capire quale sia il criterio di inversione intrinseco al manipolatore.

```
function f = cinematica_inversa(q)
```

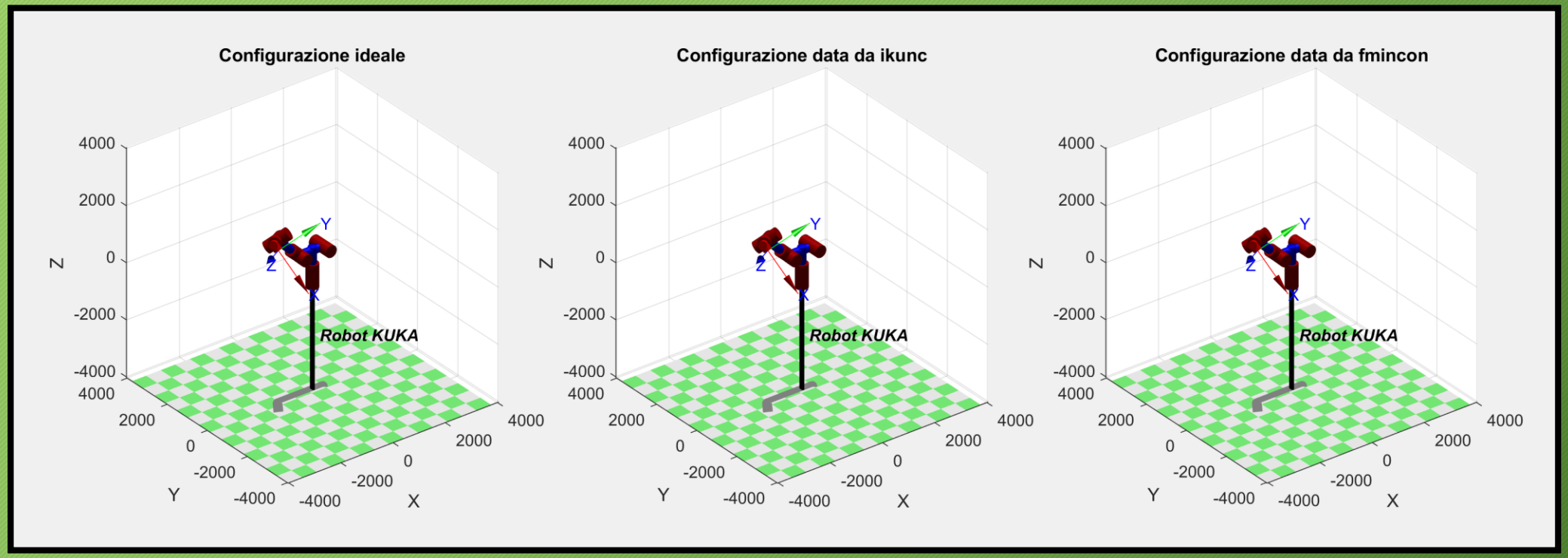
```
f = det(T_in) - det(T1);
```

```
q_inverso = fminsearch('Cinematica_inversa', q)
```

Funzione che determina il vettore di variabili di giunto ottimo fra tutti quelli che minimizzano *f*

Cinematica inversa

Confronto fra le configurazioni determinate tramite algoritmi di inversione cinematica differenti:



Generazione della traiettoria

$$p(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$ posizione

$$\dot{p}(t) = 3a_3 t^2 + 2\,a_2 t + a_1$$ velocità

$$\ddot{p}(t) = 6a_3 t + 2\,a_2$$ accelerazione

La disponibilità di 4 coefficienti consente di imporre 4 vincoli:

posizione iniziale e finale

$$\begin{cases} p_i = a_0 \\ p_f = a_3 {t_f}^3 + a_2 {t_f}^2 + a_1 t_f + a_0 \end{cases}$$

velocità iniziale e finale

$$\begin{cases} \dot{p}_i = a_1 \\ \dot{p}_f = 3a_3 {t_f}^2 + 2\,a_2 t_f + a_1 \end{cases}$$

Generazione della traiettoria su Matlab

Funzione creata ad-hoc per calcolare la traiettoria dell'end-effector:

```
function [pe,ve] = Pianifica_ee(pei,pef,pei_dot,pef_dot,tf,deltat)
```

Posizione e velocità per ogni istante di tempo

- pos.iniziale
- pos.finale
- vel.inziale
- vel.finale
- tempo finale
- intervallo campionamento

```
%% Equazioni cubiche di traiettorie e velocità lungo x y z

sx = Cx(1)*t.^3 + Cx(2)*t.^2 + Cx(3)*t + Cx(4);
sx_dot = 3*Cx(1)*t.^2 + 2*Cx(2)*t+Cx(3);

sy = Cy(1)*t.^3 + Cy(2)*t.^2 + Cy(3)*t + Cy(4);
sy_dot = 3*Cy(1)*t.^2 + 2*Cy(2)*t+Cy(3);

sz = Cz(1)*t.^3 + Cz(2)*t.^2 + Cz(3)*t + Cz(4);
sz_dot = 3*Cz(1)*t.^2 + 2*Cz(2)*t+Cz(3);

pe = [sx; sy; sz]';
ve = [sx_dot; sy_dot; sz_dot]';
end
```

Jacobiano e singolarità del manipolatore

La cinematica differenziale caratterizza i legami tra le velocità dei giunti e le corrispondenti velocità lineare e angolare dell'argano terminale. Tali legami vengono espressi tramite una matrice di trasforamzione denominata **Jacobiano geometrico**:

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

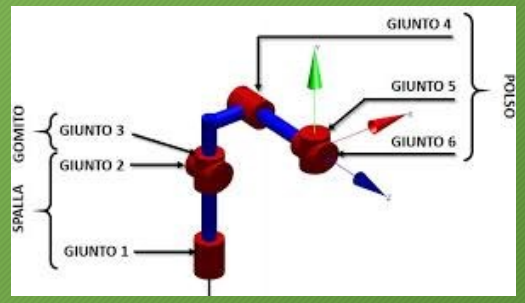
Jacobiano del manipolatore antropomorfo

Tutte quelle configurazioni per le quali J diminuisce il suo rango sono chiamate **singolarità cinematiche**,le quali comportano una perdita di mobilità della struttura:

```
jsingu(J)
```

→

```
1 linearly dependent joints:
  q6 depends on: q4
```



Singolarità del polso sferico

Cinematica differenziale inversa

Data la traiettoria dell'end-effector, si possono calcolare le velocità angolari dei singoli giunti tramite la **cinematica differenziale inversa**:

$$\dot{q} = J^{\dagger}\, v_e$$

$\dot{q}$: vel.giunti

$J^{\dagger}$: pesudo-inversa dello jacobiano

v_e: vel.end-effector

Ottimizzazione: $\dot{q} = J^{\dagger}\, v_e + (I - J^{\dagger}J)\underbrace{\dot{q}_d}_{\text{vel.giunti arbitrario}}$

→ è possibile sceglierlo per esempio per evitare situazioni di singolarità

Simulazione della traiettoria

Matlab esegue un ciclo *for* per calcolare la cinematica differenziale inversa ogni istante campionato:

```
%% Esecuzione della cinematica differenziale

for i = 1:1:n
    q = qnew;
    jacob13 = KUKA.jacob0(q);                          % Calcolo jacobiano
    q13_dot = pinv(jacob13(1:3,1:3))*Ve(i,:)';         % Inversione differenziale q_dot = J' x Ve
    q13 = q13 + q13_dot'*deltat;                       % Integrazione discreta q(t+1) = q(t) + deltat*q_dot(t)'
    qnew(:,1:3) = q13;
    mov(i,:) = qnew;
    T = KUKA.fkine(qnew);
    PeEff_kuk(i,:) = A*[T.t;1];
end

plot(KUKA,mov);                                        % Simulazione
```

Generazione del file G-Code

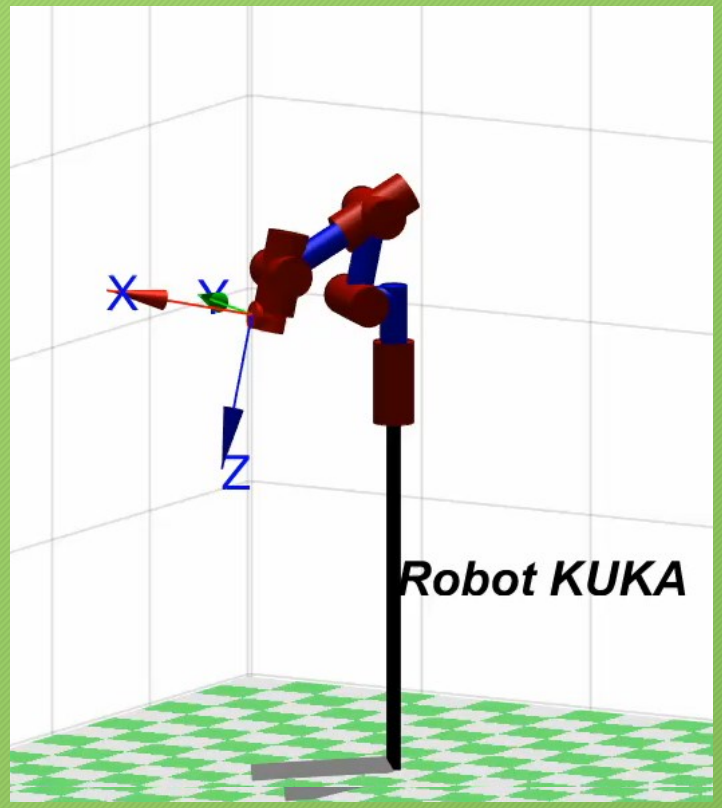
Il file G-Code permette di programmare il manipolatore:

```
function [] = G_Code(Pe)

fileID = fopen('Kuka.txt', 'w+');        % Creazione del file di testo

formatSpec1 = 'G1 X%f Y%f Z%f\n';        % Movimento punto-punto
formatSpec0 = 'G0 X%f Y%f Z%f F2000\n'; % Riposizionamento al punto iniziale

fprintf(fileID, formatSpec0, Pe(1,:));  % Generazione del file
for n = 2 : length(Pe)
    fprintf(fileID, formatSpec1, Pe(n,:));
end

fclose('all');                          % Chiusura del file di testo
end
```

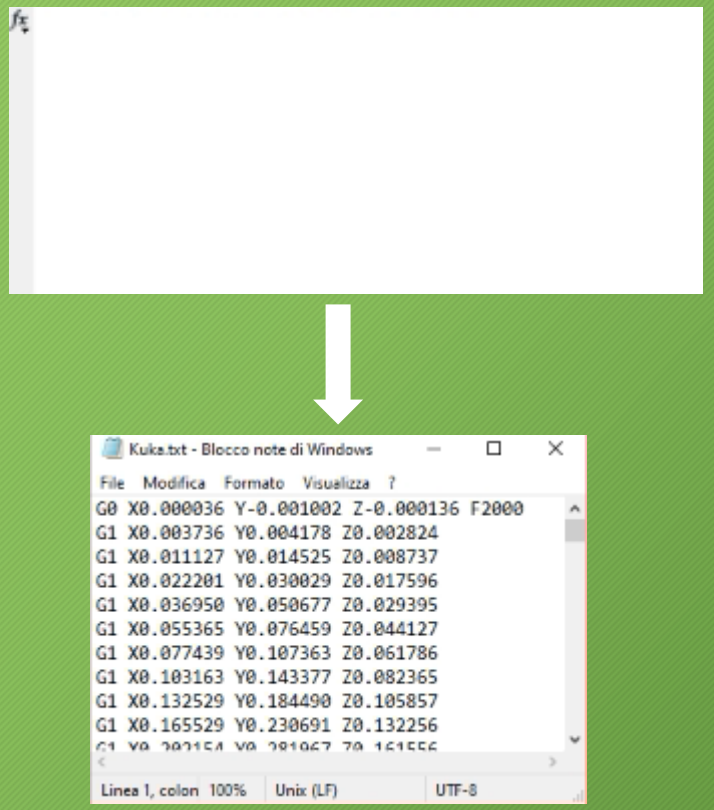
```
G_Code(PeEff_kuk);
```

```
G0 X0.000036 Y0.001002 Z-0.000136 F2000
G1 X12.282328 Y17.526416 Z9.929529
G1 X34.834117 Y50.239255 Z28.343630
G1 X65.764915 Y95.736253 Z53.849732
G1 X103.289357 Y151.562125 Z85.090577
G1 X145.710065 Y215.212637 Z120.704362
G1 X191.398964 Y284.134248 Z159.286508
G1 X238.771423 Y355.728824 Z199.376028
G1 X286.250423 Y427.367744 Z239.475387
G1 X332.228448 Y496.412361 Z278.089958
G1 X375.041482 Y560.235320 Z313.761023
G1 X412.962268 Y616.240335 Z345.074825
G1 X444.206522 Y661.880566 Z370.648377
G1 X466.939997 Y694.675386 Z389.104088
G1 X479.278362 Y712.225144 Z399.045916
G1 X479.278363 Y712.225144 Z399.045916
```

Simulazione vs test



SIMULAZIONE MATLAB



TEST DEL G-CODE GENERATO

Conclusioni

- In conclusione il test effettuato dimostra che il manipolatore segue abbastanza fedelmente la traiettoria assegnatoli
- Si potrebbe raggiungere un controllo più fine solo programmando direttamente il controllore già montato all'interno del manipolatore (o sostituirlo)
- Uno sviluppo futuro potrebbe comportare la concatenazione di traiettorie punto-punto al fine di creare percorsi che l'end-effector che evitino, ad esempio, oggetti e ostacoli all'interno dello spazio di lavoro