Corso di Controllo dei Robot
Introduzione

Paolo Lino

Dipartimento di Ing. Elettrica e dell'Informazione (DEI)

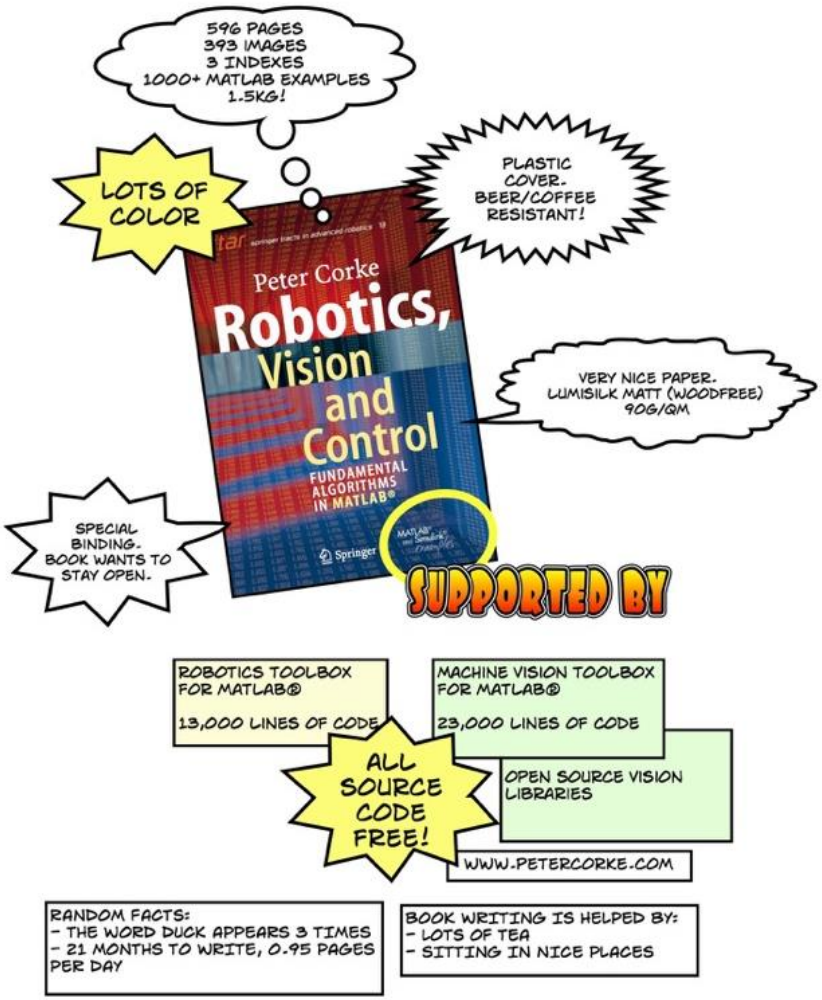
Robotics Toolbox - Introduzione

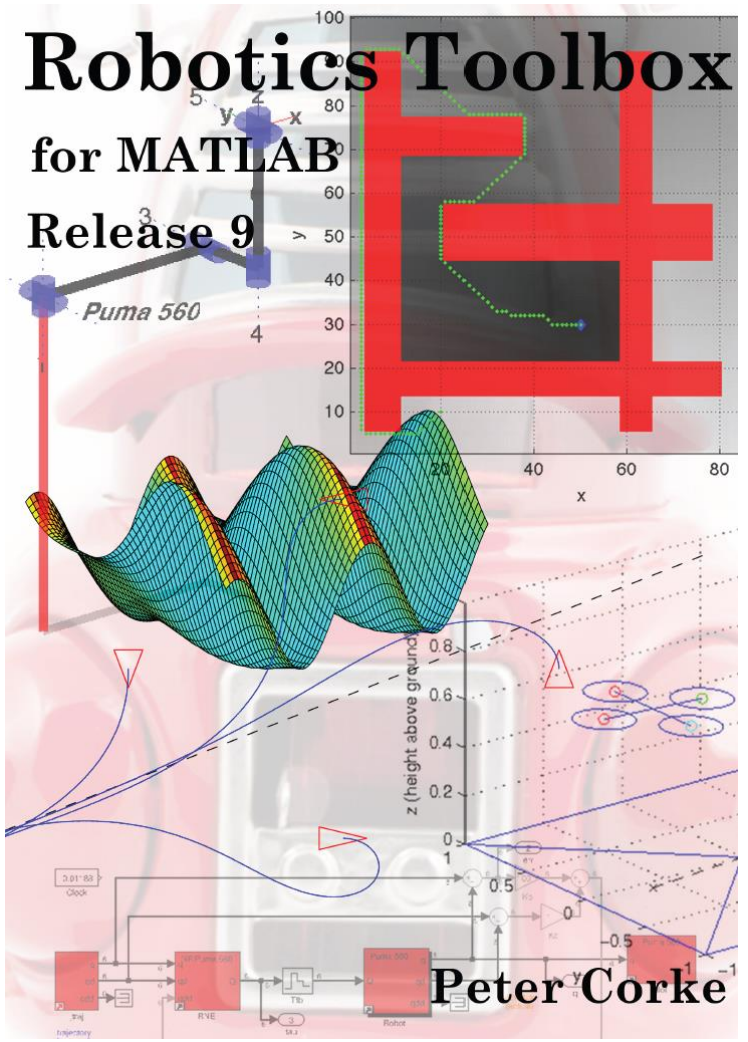
Robotics Toolbox

http://www.petercorke.com/Robotics_Toolbox.html

- **Studio e simulazione di manipolatori industriali**
  - Cinematica
  - Dinamica
  - Pianificazione delle traiettorie
- **Rappresentazione ad oggetti in ambiente MATLAB**
- **Fornisce funzioni per la manipolazione e la conversione di vettori, matrici di rotazione, trasformazioni omogenee, etc. per la rappresentazione di posizione e orientamento nello spazio**
- **Numerosi modelli presenti nel toolbox (e.g. Puma 560, Manipolatore di Stanford)**

Robotics Toolbox - Introduzione





Matrici di rotazione ortonormali

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
>> syms t

>> rotx(t)

>> roty(t)

>> rotz(t)
```

Matrici di rotazione ortonormali

$$R^T R = I$$

$$detR = 1$$

```
>> R = rotx(pi/2)
>> R'*R
>> det(R)
```

```
>> trplot(R)
>> tranimate(R)
```

Composizione di rotazioni

```
>> R = rotx(pi/2) * roty(pi/2)
>> trplot(R)
```

Matrici di rotazione ortonormali

Matrici di rotazione anticommutative

```
>> R1 = rotx(pi/2)*roty(pi/2)

>> R2 = roty(pi/2)*rotx(pi/2)

>> trplot(R1,'frame','1','color','b'), hold on

>> trplot(R2,'frame','2','color','r')

>> tranimate(R1),tranimate(R1,R1*R2)

>> tranimate(R2),tranimate(R2,R2*R1)
```

Angoli di Elulero – Trasformazione ZYZ

$$\boldsymbol{R} = \boldsymbol{R}_z(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)$$

$$\Gamma = (\phi, \theta, \psi)$$

>> syms fi theta psi

>> Reul=rotz(fi)*roty(theta)*rotz(psi)

>> R = rotz(0.1) * roty(0.2) * rotz(0.3);

>> R = eul2r(0.1, 0.2, 0.3)

Problema inverso

>> gamma = tr2eul(R)

Angoli di Elulero – Trasformazione RPY

$$\boldsymbol{R} = \boldsymbol{R}_x(\theta_r)\boldsymbol{R}_y(\theta_p)\boldsymbol{R}_z(\theta_y)$$

$$\Gamma = (\phi, \theta, \psi)$$

```
>> syms r p y
>> R = rotx(r)*roty(p)*rotz(y)
```

```
>> R = rotx(0.1)*roty(0.2)*rotz(0.3)
>> R = rpy2r(0.1, 0.2, 0.3)
```

Problema inverso

```
>> gamma = tr2rpy(R)
```

Trasformazioni omogenee

$$\tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad A_1^0 = \begin{bmatrix} R_1^0 & o_1^0 \\ 0^T & 1 \end{bmatrix}$$

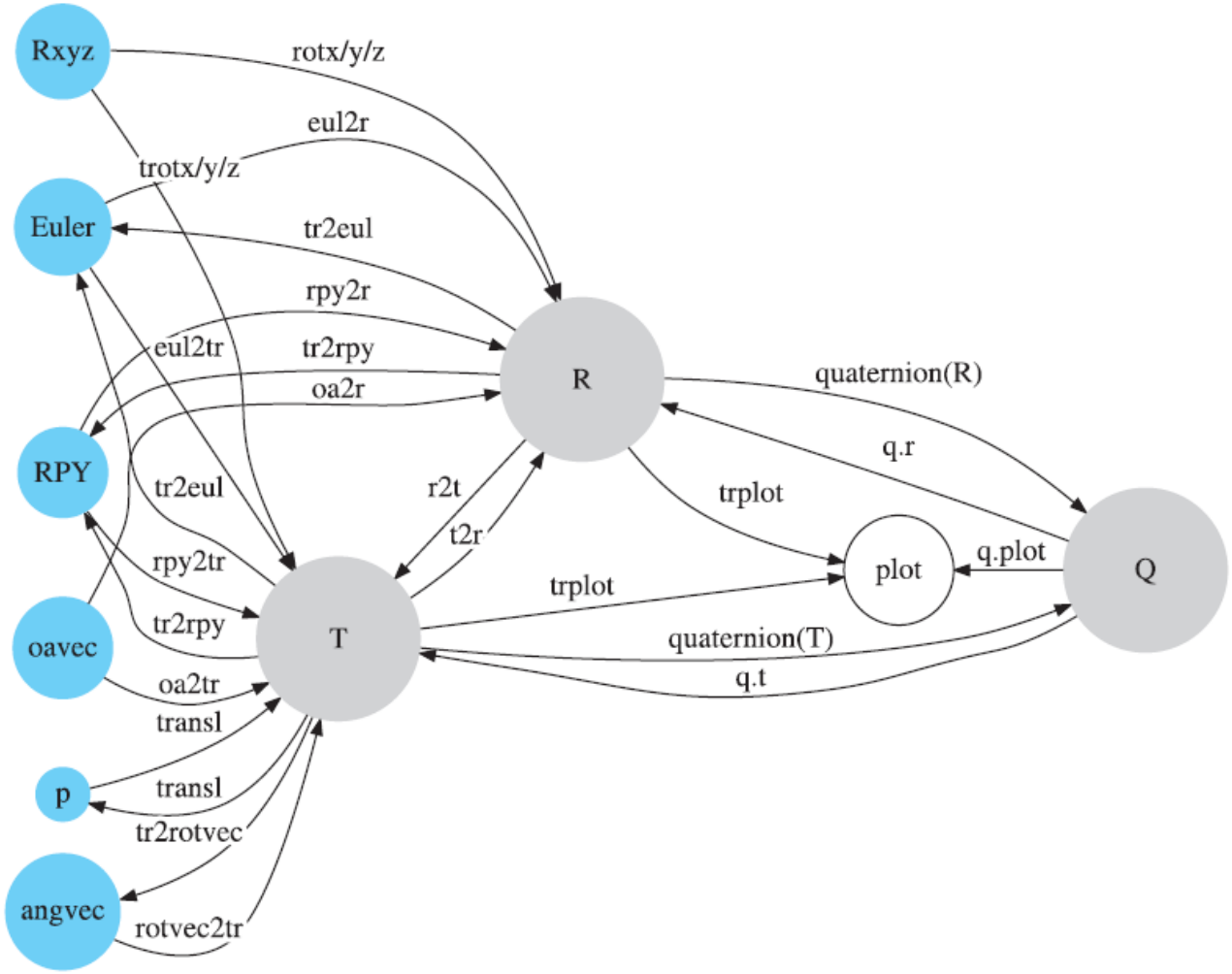
$$\tilde{p}^0 = A_1^0 \tilde{p}^1 \qquad \tilde{p}^0 = A_1^0 A_2^1 \cdots A_n^{n-1} \tilde{p}^n$$

```
>> syms t x y z
>> trotx(t)
>> troty(t)
>> trotz(t)
>> transl(x, y, z)
```

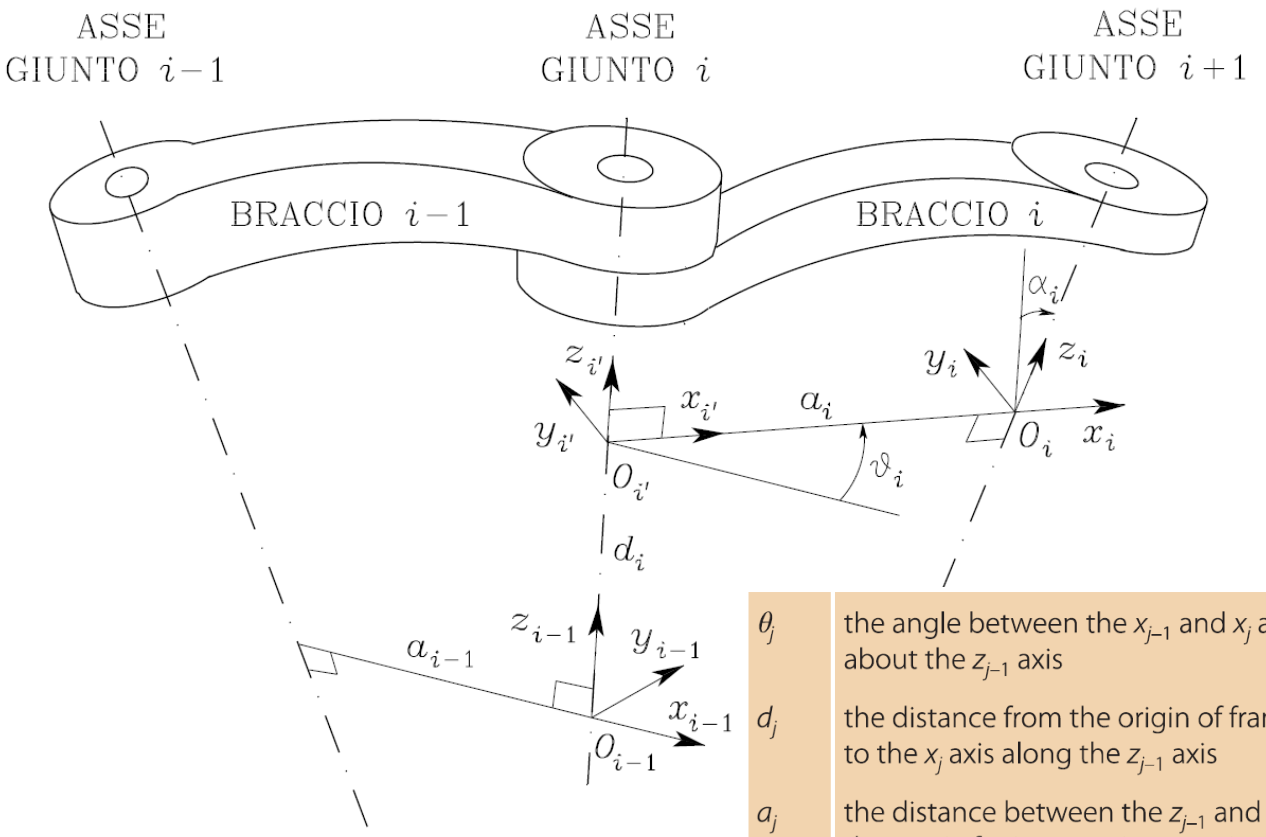
Composizione di trasformazioni

```
>> T = transl(1, 0, 0) * trotx(pi/2) *transl(0, 1, 0)
>> trplot(T)
>> tranimate(transl(1,0,0)), tranimate(transl(1,0,0), transl(1,0,0)*trotx(pi/2))
>> tranimate(transl(1,0,0)*trotx(pi/2), T)
>> t2r(T)
>> transl(T)
```

Conversione tra rappresentazioni



Convenzione di Denavit – Hartenberg



θ_j	the angle between the x_{j-1} and x_j axes about the z_{j-1} axis	revolute joint variable
d_j	the distance from the origin of frame $j-1$ to the x_j axis along the z_{j-1} axis	prismatic joint variable
a_j	the distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axes is parallel to $\hat{\boldsymbol{z}}_{j-1} \times \hat{\boldsymbol{z}}_j$	constant
α_j	the angle from the z_{j-1} axis to the z_j axis about the x_j axis	constant
σ_j	$\sigma = 0$ for a revolute joint, $\sigma = 1$ for a prismatic joint	constant

Convenzione di Denavit – Hartenberg

Definizione dei riferimenti ai giunti

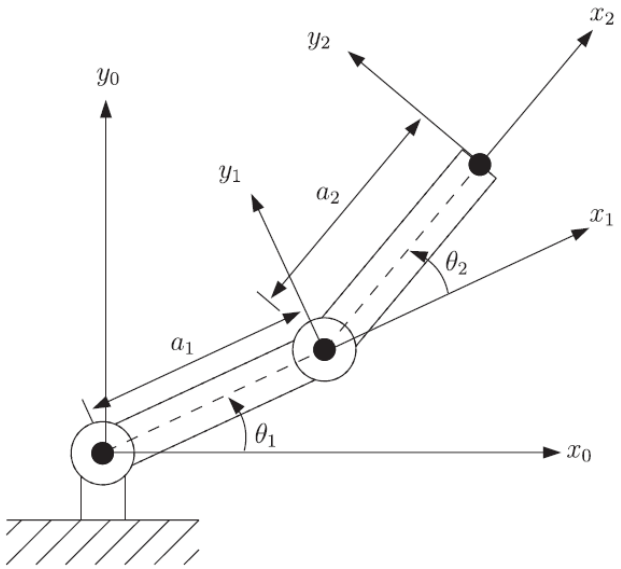
$$(\theta_j, d_j, a_j, \alpha_j, \sigma_j)$$

$$A_i^{i-1}(q_i) = A_{i'}^{i-1} A_i^{i'} = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i} c_{\alpha_i} & s_{\vartheta_i} s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i} c_{\alpha_i} & -c_{\vartheta_i} s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>> L = Link([0, 0.1, 0.2, pi/2, 0])
>> L.A(pi/4)
>> L.RP
>> L.a
>> L.d
```

Manipolatore planare a 2 bracci

$(\theta_j, d_j, a_j, \alpha_j, \sigma_j)$

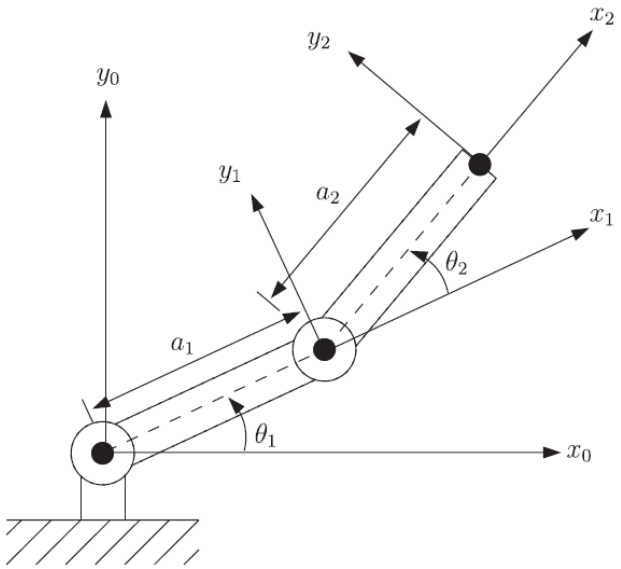



```
>> L(1) = Link([0 0 1 0]);
>> L(2) = Link([0 0 1 0]);
>> L
```

Costruttore catena cinematica

```
>> two_link = SerialLink(L, 'name', 'twolink')
>> two_link.n
>> links = two_link.links
>> clone = SerialLink(two_link, 'name', 'bob')
>> two_link.fkine([pi/6 pi/8])
>> two_link.plot([pi/6 -pi/4]),axis equal
```

Manipolatore planare a 2 bracci

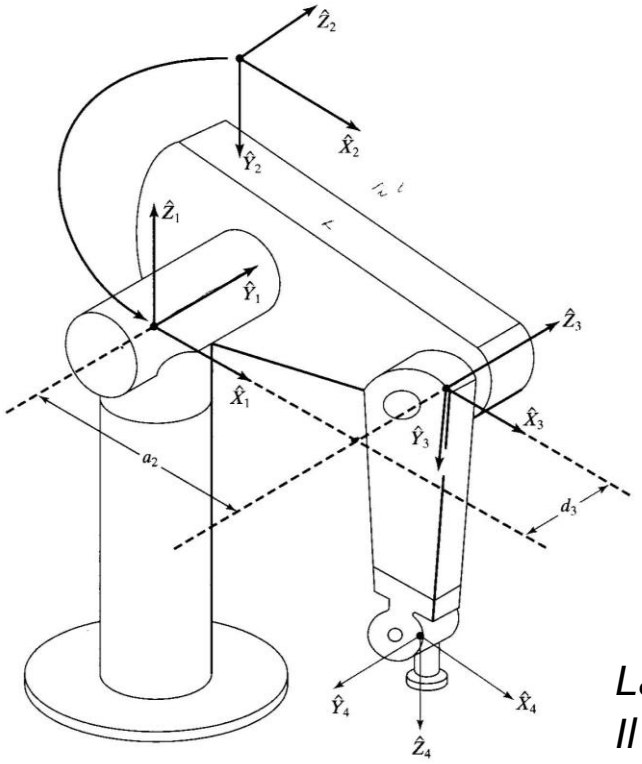



```
>> syms q1 q2 a1 a2
>> linkSym(1) = Link([0 0 a1 0 0]);
>> linkSym(2) = Link([0 0 a2 0 0]);
```

Costruttore catena cinematica

```
>> robotSym = SerialLink(linkSym, 'name', 'Symbo');
>> robotSym.fkine([q1 q2])
```

Manipolatore 6 DOF PUMA 560




```
>> mdl_puma560
>> p560
```

Configurazioni canoniche

`qz`	$(0, 0, 0, 0, 0, 0)$	*zero angle*
`qr`	$(0, \frac{\pi}{2}, -\frac{\pi}{2}, 0, 0, 0)$	*ready*, the arm is straight and vertical
`qs`	$(0, 0, -\frac{\pi}{2}, 0, 0, 0)$	*stretch*, the arm is straight and horizontal
`qn`	$(0, \frac{\pi}{4}, -\pi, 0, \frac{\pi}{4}, 0)$	*nominal*, the arm is in a dextrous working pose

```
>> p560.plot(qn)
>> p560.fkine(qz)
```

La terna 0 *è scelta con origine all'intersezione di* z_0 *e* z_1*.*
Il robot Puma560 include un piedistallo di 30 pollici. Traslando l'origine del riferimento del robot verso la base del piedistallo:

```
>> p560.base = transl(0,0,30*0.0254) * trotz(pi);
>> p560.plot(qn);
>> p560.fkine(qn)
```

Manipolatore 6 DOF PUMA 560

Cinematica inversa

```
>> qn
>> T = p560.fkine(qn)
>> qi1 = p560.ikine6s(T)
>> qi2=p560.ikine6s(T,'ru')
```

configurazione nominale

posa organo terminale

variabili di giunto

configuraz. destrorsa con gomito in sù

La soluzione al problema di cinematica inversa non è unica!

```
>> p560.plot(qi1);
>> p560.plot(qi2);
```

***Parametri ikines6s*:**

left or right handed	`'l','r'`
elbow up or down	`'u','d'`
wrist flipped or not flipped	`'f','n'`

Non tutte le configurazioni sono fisicamente raggiungibili!

```
>> T = transl(3,0,0);
>> p560.ikine6s(T)
```

Il braccio non è sufficientemente lungo per raggiungere questa posa

Manipolatore 6 DOF PUMA 560

Cinematica inversa

```
>> p1=SerialLink(p560,'name','P1');
>> p2=SerialLink(p560,'name','P2');
>> p3=SerialLink(p560,'name','P3');
>> p4=SerialLink(p560,'name','P4');
>> p1.base=transl(.5,.5,0);
>> p2.base=transl(.5,-.5,0);
>> p3.base=transl(-.5,-.5,0);
>> p4.base=transl(-.5,.5,0);
>> q1=p1.ikine6s(T,'ru'); q2=p1.ikine6s(T,'rd');
>> q3=p1.ikine6s(T,'lu'); q4=p1.ikine6s(T,'ld');
>> close all,p1.plot(q1),hold on
>> p2.plot(q2),hold on
>> p3.plot(q3),hold on
>> p4.plot(q4),hold on
>> axis equal
```

Manipolatore ridondante

Link	θ_i	d_i	a_i	α_i	σ_i
1	0	q_1	0	$-\frac{\pi}{2}$	1
2	$-\frac{\pi}{2}$	q_2	0	$\frac{\pi}{2}$	1

Creiamo una base mobile (parametri DH)

```
>> platform = SerialLink([0 0 0 -pi/2 1; -pi/2 0 0 pi/2 1], ...
        'base', troty(pi/2), 'name', 'platform' );
>> platform.fkine([1, 2])
>> p560.links(1).d = 30 * 0.0254;
>> p8 = platform * p560;
>> p8 = SerialLink( [platform, p560]);
>> p8
>> T = transl(0.5, 1.0, 0.7) * rpy2tr(0, 3*pi/4, 0);
>> qi = p8.ikine(T)
```

Connessione in serie di base e Puma

Alternativa

Cambio posa end effector

Traiettorie nello spazio dei giunti
Moto punto-punto

- Il manipolatore deve muoversi da una configurazione iniziale (delle variabili di giunto) ad una finale in un tempo t_f.
- Il percorso seguito dall'organo terminale non è importante.
- Traiettoria in grado di ottimizzare lo spostamento del giunto da una posizione all'altra.

Polinomio di 5° grado

Consente di assegnare posizioni, velocità, accelerazioni iniziali e finali:

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

$$\begin{cases} q(t_i) = a_0 \\ q(t_f) = a_5 t_f^5 + a_4 t_f^4 + a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 \\ \dot{q}(t_i) = a_1 \\ \dot{q}(t_f) = 5a_5 t_f^4 + 4a_4 t_f^3 + 3a_3 t_f^2 + 2a_2 t_f + a_1 \\ \ddot{q}(t_i) = 2a_2 \\ \ddot{q}(t_f) = 20a_5 t_f^3 + 12a_4 t_f^2 + 6a_3 t_f + 2a_2 \end{cases}$$

```
>> qi = 0; qf = 1; vi = 0.5; vf = 0;
>> s = tpoly(qi, qf, 50);
>> s = tpoly(qi, qf, 50, vi, vf);
>> [s,sd,sdd] = tpoly(qi, qf, 50);
>> figure, plot(s)
>> figure, plot(sd)
>> figure, plot(sdd)
```

Profilo di velocità trapezoidale

$$q(t)=\begin{cases} q_i+\frac{1}{2}\ddot{q}_c t^2 & 0 \le t \le t_c \\ q_i+\ddot{q}_c t_c\left(t-\frac{t_c}{2}\right) & t_c < t \le t_f - t_c \\ q_f-\frac{1}{2}\ddot{q}_c\left(t_f-t\right)^2 & t_f - t_c < t \le t_f \end{cases}$$

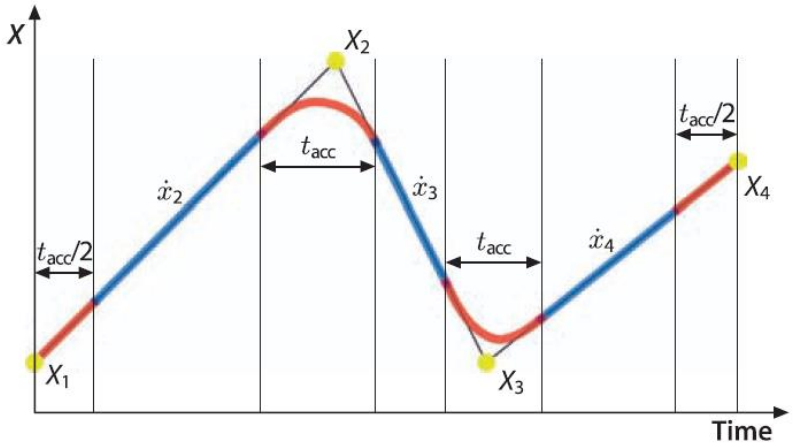
```
>> qi = 0; qf = 1; vm = 0.025;
>> s = lspb(qi, qf, 50);
>> [s,sd,sdd] = lspb(qi, qf, 50);
>> max(sd)
>> s = lspb(qi, qf, 50, vm);
>> figure, plot(s)
>> figure, plot(sd)
>> figure, plot(sdd)
```

Moto su percorso assegnato

- Sequenza di polinomi con vincoli imposti sulle velocità nei punti di percorso
- Sequenza di polinomi con velocità calcolate nei punti di percorso (impongo la primitiva e calcolo la velocità nel punto finale)
- Sequenza di polinomi con accelerazione continua nei punti di percorso (spline)
- Sequenza di polinomi parabolico-lineari con passaggio in prossimità dei punti di via

Sequenza di polinomi parabolico-lineari con passaggio in prossimità di punti di via

Sequenza di polinomi di primo e secondo grado: continuità sulla derivata prima, discontinuità sulla derivata seconda




```
>> via = [ 4,1; 4,4; 5,2; 2,5 ];
>> q = mstraj(via, [2,1], [], [4,1], 0.05, 0);
>> plot(q)
>> q = mstraj(via, [2 1], [], [4 1], 0.05, 1);
>> plot(q)
```