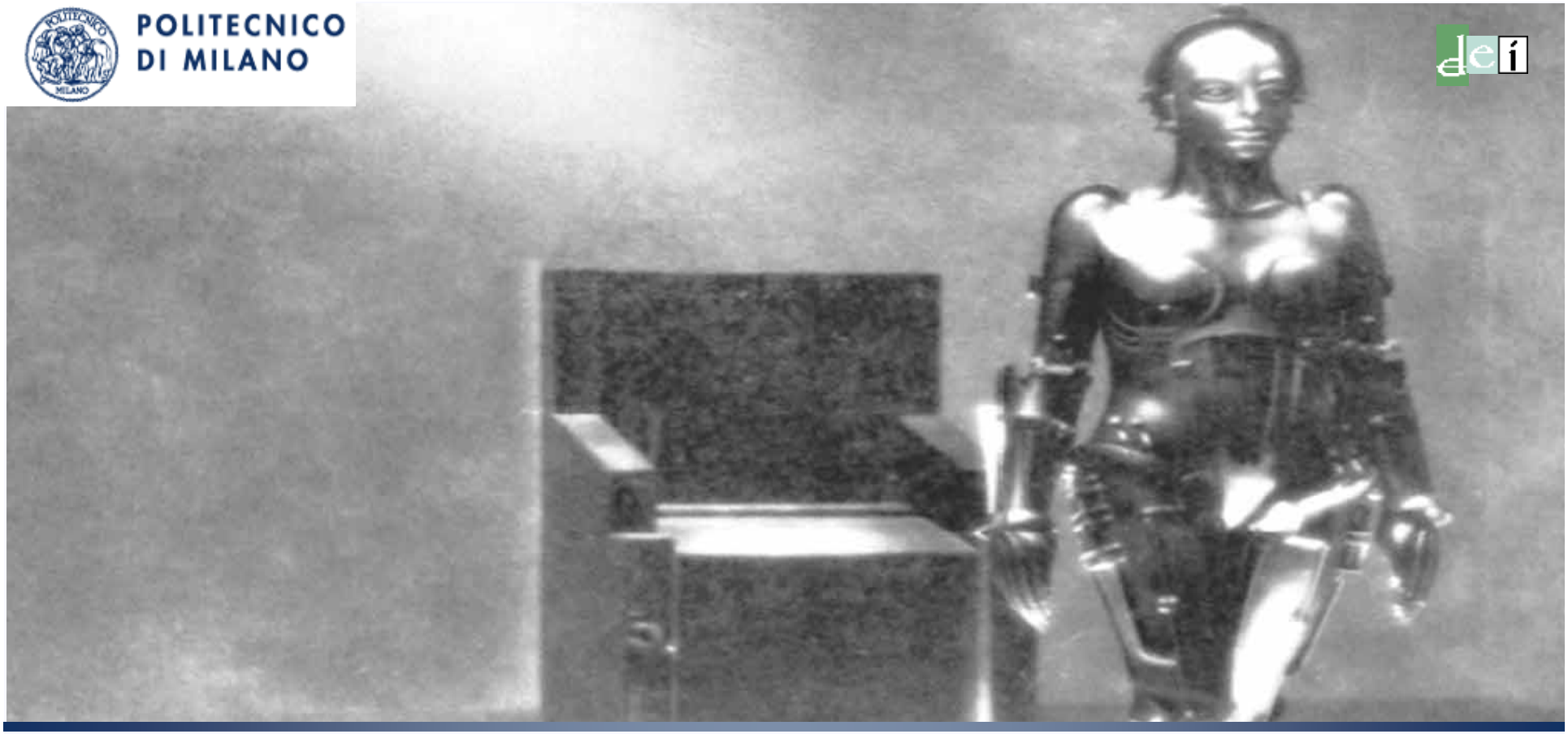
POLITECNICO DI MILANO



# 070342 – Robotica

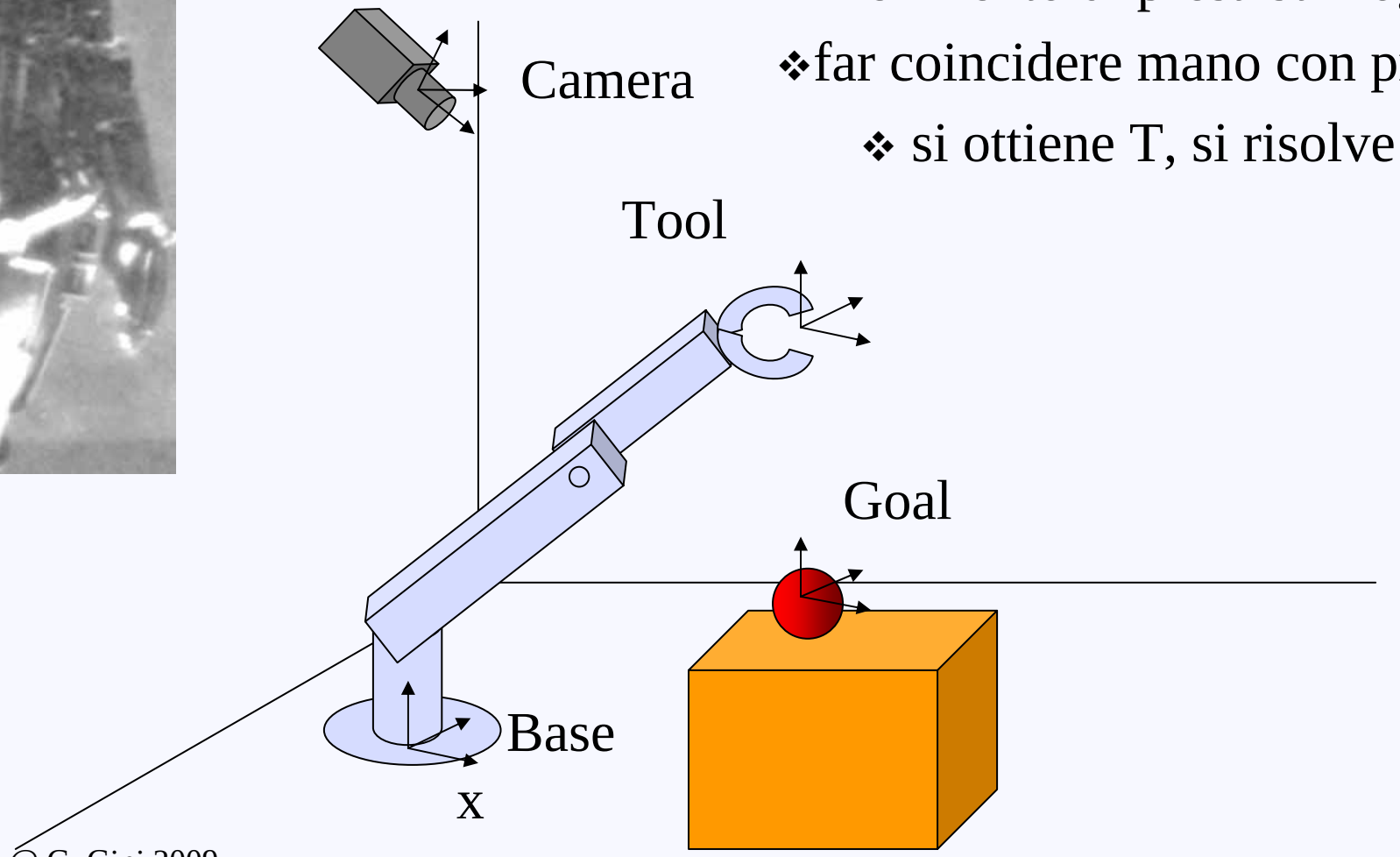
http://home.dei.polimi.it/gini/robot/

## Programmazione: dai giunti al manipolatore

© 2009 Giuseppina Gini

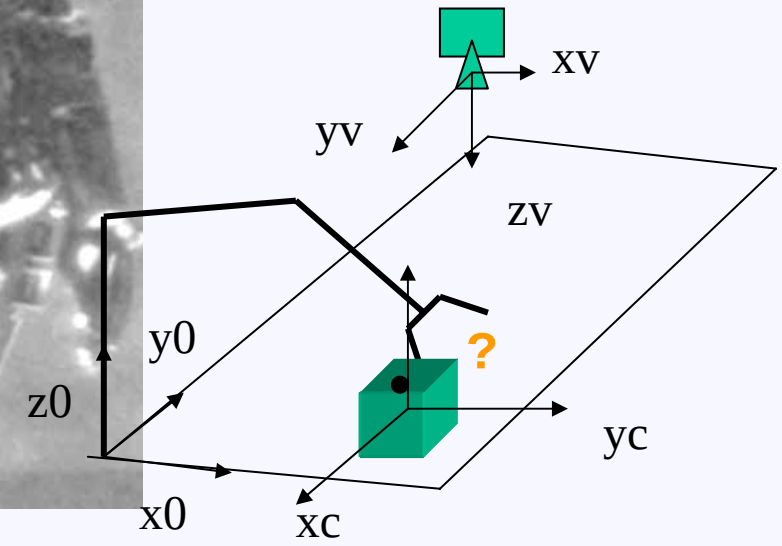
# Sistemi di riferimento e programmazione

- riferimento di presa sull'oggetto
- far coincidere mano con presa:
  - si ottiene T, si risolve CI




@ G. Gini 2009

# Posizione e presa di oggetti visti da camera




3 sistemi di riferimento:

- Sistema di riferimento *0*
- sistema telecamera
- sistema cubo (baricentro)

Date le matrici:

- **T1**:centro del cubo nel sistema di riferimento della *telecamera*
- **T2**: l'origine 0 vista dalla *telecamera*

**trovare**: il centro del cubo nel sistema assoluto *0*

@ G. Gini 2009

# Esempio numerico

$$T1 = cameraTcubo = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 10 \\ 0 & 0 & -1 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Le matrici note T1 e T2

$$T2 = cameraTorigine \begin{bmatrix} 1 & 0 & 0 & -10 \\ 0 & -1 & 0 & 20 \\ 0 & 0 & -1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

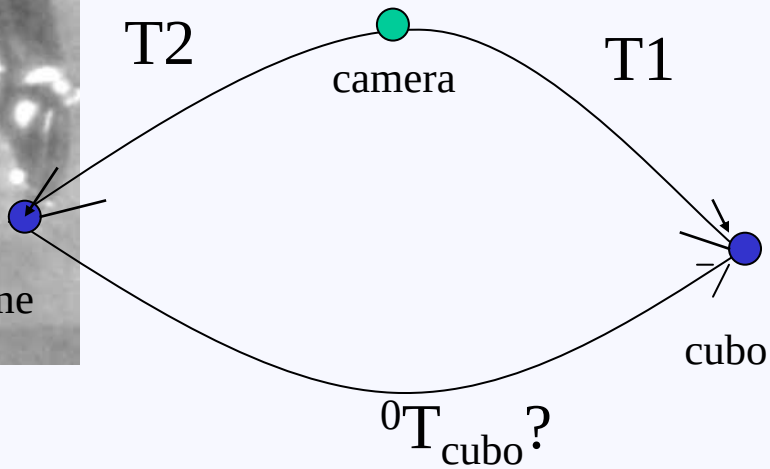
- Si inverte T2

$$invT2 = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & -1 & 0 & 20 \\ 0 & 0 & -1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

@ G. Gini 2009

# La soluzione

$$ {}^{origine}T_{cubo} = {}^{origine}T_{camera} * {}^{camera}T_{cubo} $$
$$ = (T2)^{-1} * T1 $$



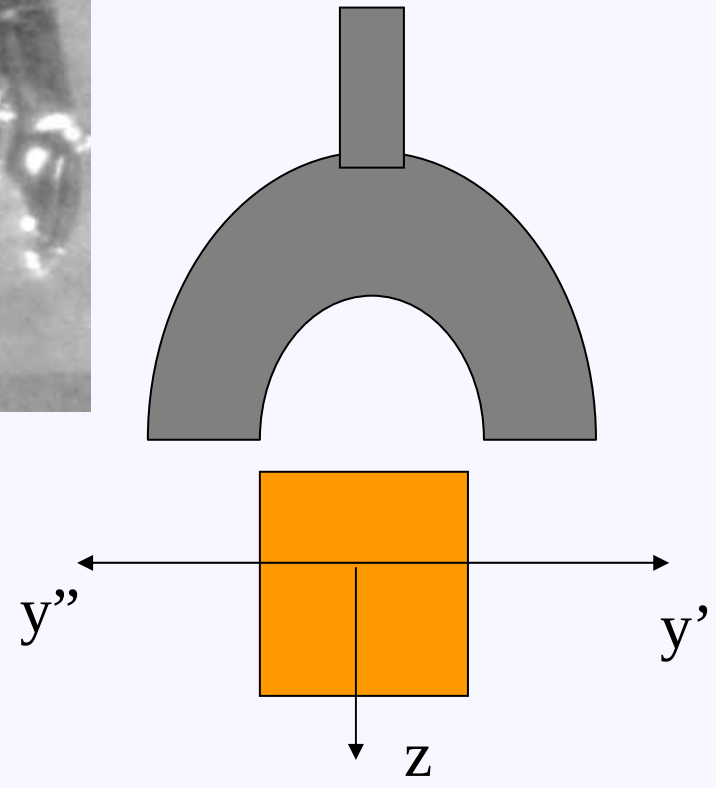

$$ invT2 * T1 = \begin{bmatrix} 0 & 1 & 0 & 11 \\ -1 & 0 & 0 & 10 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Osservazione: gli assi di *cubo* sono paralleli a -y, x, z del sistema *O*, cioè Rot(z)90°

@ G. Gini 2009

# Grasp dall'alto

- Quale è il grasp dall'alto?
- Due soluzioni per l'orientamento:
  - **a** = $(0,0,-1)^T$
  - **s** = $(\pm 1,0,0)^T$
  - **n** = **s** x **a**= $(0, \pm 1, 0)^T$




@ G. Gini 2009

# Matrice di grasp

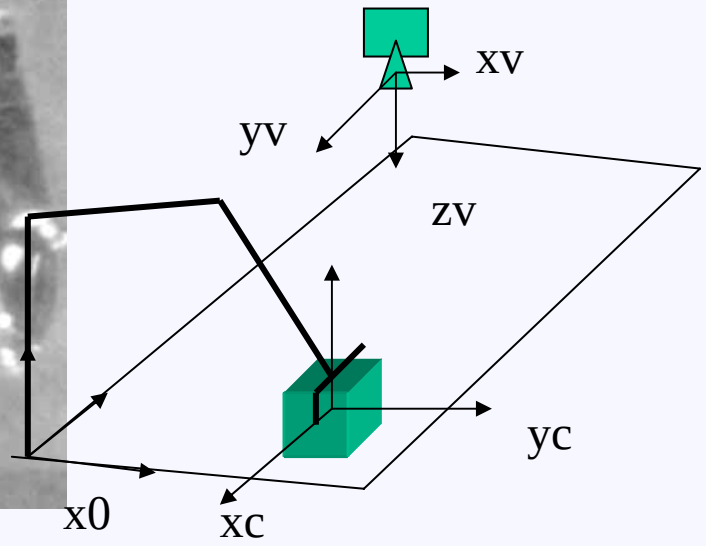
$$Tgrasp = \begin{bmatrix} 0 & 1 & 0 & 11 \\ 1 & 0 & 0 & 10 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Tgrasp' = \begin{bmatrix} 0 & -1 & 0 & 11 \\ -1 & 0 & 0 & 10 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Ci sono due possibili matrici di grasp che corrispondono alla stessa presa
  - Il programmatore ne indica una in base ai valori cartesiani che assegna

@ G. Gini 2009

# Programmazione = formulare le equazioni



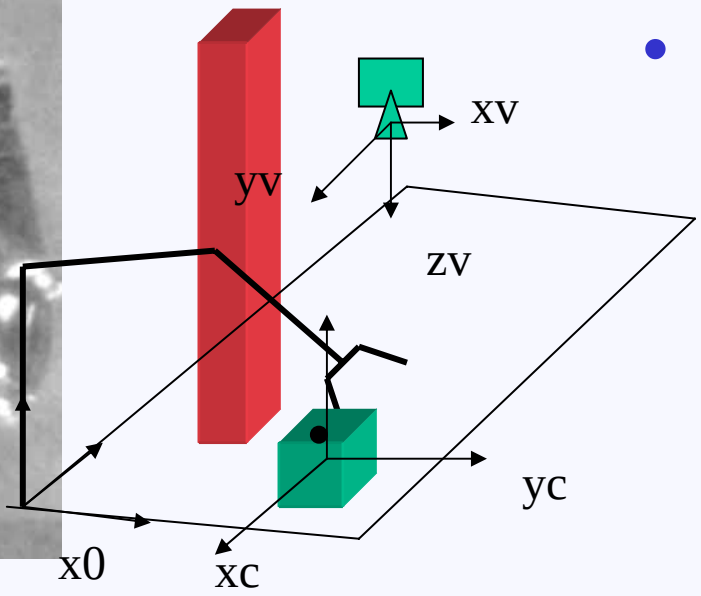

I movimenti sono descritti come equazioni del tipo:

**grasp_cubo = arm**

- Dove il secondo membro (arm) riceve i valori del primo e viene usato per estrarre, mediante cinematica inversa, i valori da attuare

@ G. Gini 2009

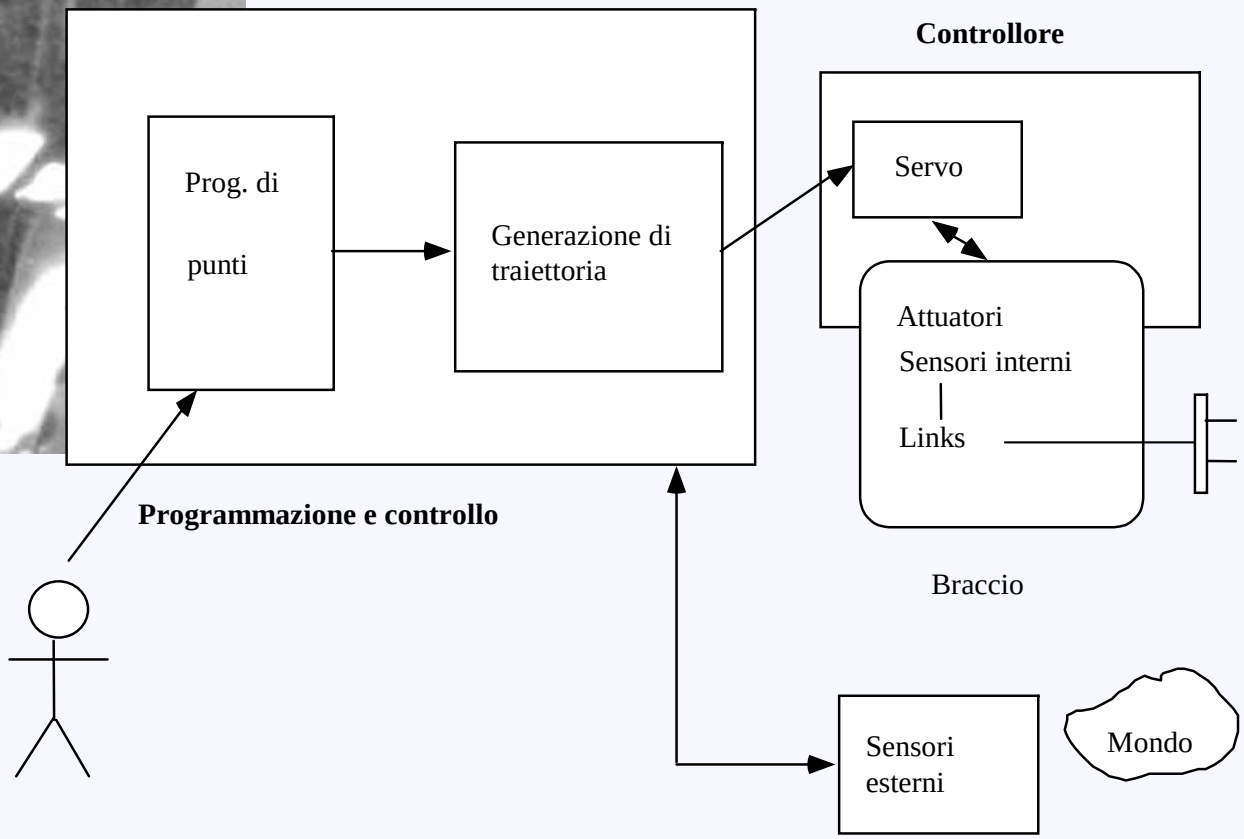
# Dalla mano alla giacitura della catena robot




- Problema del mondo cartesiano:
  - la geometria degli ostacoli e degli oggetti non basta per descrivere la giacitura del robot, quindi la catena può urtare il mondo anche quando la mano non ha collisioni
  - Occorre calcolare un cammino che evita urti con gli ostacoli per tutti i link

@ G. Gini 2009

# La programmazione: punti e traiettorie




- fornire il target (controllo in posizione)
- definire e controllare la traiettoria

@ G. Gini 2009

# Problemi

- Le azioni non sono in genere reversibili.
- Le cause degli errori sono difficilmente individuabili..
- Le situazioni sono difficilmente ripetibili.
- Il coordinamento fra le diverse attività è complicato.
- Il monitoraggio di alcune variabili o condizioni può essere complicato.
- Specificare operazioni nello spazio 3D è difficile in generale (interfaccia – grafica - VR- simulazione)
- IL FUTURO:
- ***SUPERARE LA PROGRAMMAZIONE***: mostrare l'azione (imitation learning?)

@ G. Gini 2009

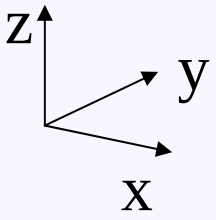
# Dal linguaggio al controllo

- Il programma è nello spazio operativo
- La cinematica inversa produce i valori dei giunti
- Il controllore li attua

cinematica inversa

Spazio cartesiano → Spazio dei giunti → Spazio degli attuatori

dinamica




@ G. Gini 2009

| Caratteristiche | Descrizione | Linguaggi che le supportano |
|---|---|---|
| ***Sintassi strutturata*** | tipica di tutti i linguaggi ad alto livello. | **derivati AL (VAL)** |
| ***Processi concorrenti*** | coordinamento | **AL (Stanford U)** |
| ***Estensibilità funzionale*** | Il linguaggio fornisce poche funzioni essenziali queste possono poi essere estese dell'utente | **AML (IBM)** |
| ***Indipendenza dal robot*** | creare programmi di validità generale | **RAPT** |
| ***Integrazione sensori*** | Amplia le possibilità di utilizzo del robot stesso. | **Sia da AL che da AML** |
| ***Uso di conoscenza geometrica*** | ricavata da modelli CAD riduce la complessità della programmazione. | **LM-Geo, RAPT** |
| ***funzionamento off-line*** | Si possono eseguire alcune fasi off-line ad esempio simulazioni.. | **tutti** |

@ G. Gini 2009

**TASK LEVEL**
(livello del task o compito)

A questo livello si definisce il compito da svolgere tramite il linguaggio naturale.
Ad esempio: "Unisci il pezzo A al pezzo B"

Comprensione del linguaggio naturale
Generare un piano delle azioni da svolgere tramite la conoscenza degli oggetti (modello geometrico) e di un modello delle azioni usando tecniche di ragionamento. Il piano generato deve essere adeguato e permettere di evitare le collisioni.

**OBJECT LEVEL**
(livello degli oggetti)

A questo livello si modellizzano in modo più accurato gli oggetti. Si hanno quindi dei modelli degli oggetti a cui si fa riferimento.
Ad esempio: "Metti il cubo A sopra la squadra B"

Modellizazione degli oggetti che saranno oggetto della manipolazione.

**MANIPULATOR LEVEL**
(livello manipolatore)

A questo livello si danno comandi precisi al manipolatore per realizzare singoli passi.
Ad esempio: "Apri le dita di 3 cm", "Muovi il braccio fino alla posizione di pick-up", ecc…

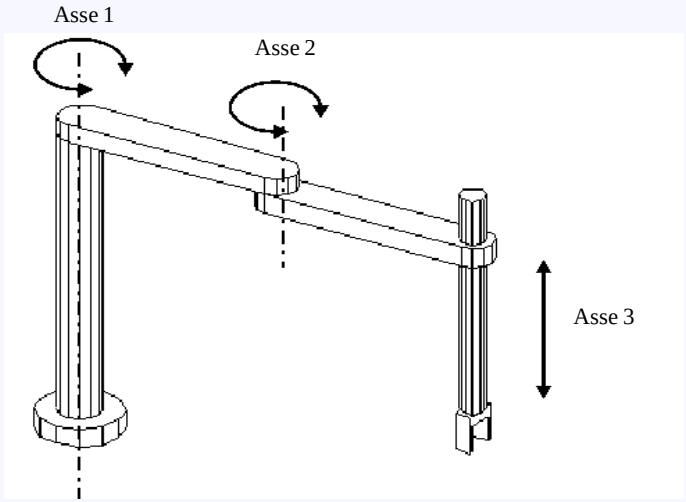
Conversione da coordinate cartesiane a variabili di giunto e viceversa.
Generazione delle traiettorie.

**JOINT LEVEL**
(livello dei giunti)

A questo livello si comandano i giunti del manipolatore.
Ad esempio: "Porta il giunto 1 a 98°", ecc…

@ G. Gini 2009

# Programmazione on line/off line

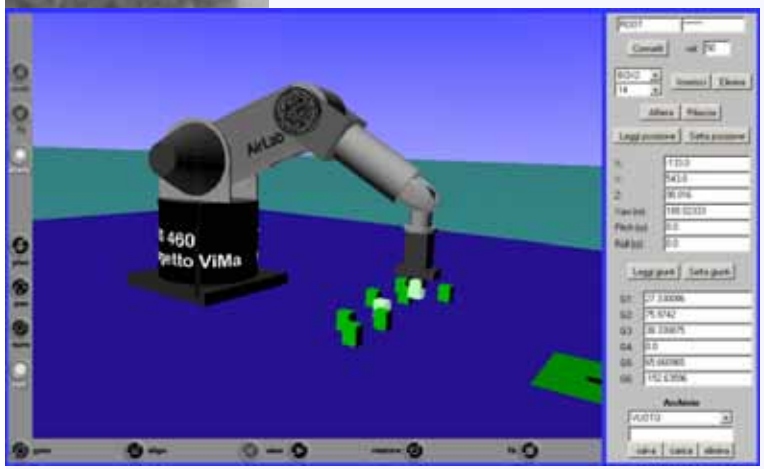



- On-line
  - Usa teach pendant e robot reale
  - Richiede abilità
  - Tempo di arresto
  - Buona accuratezza
- Off-line
  - Usa modelli
  - Sfrutta la grafica
  - Riduce tempi di arresto
  - Meno accurata

@ G. Gini 2009

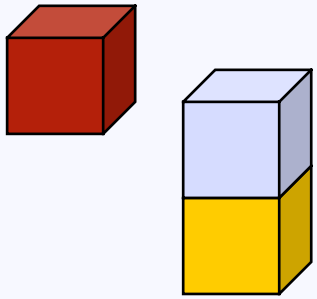
# Programmazione Off-line

Modello del robot -> posizione -> controllore -> robot reale -> mondo



x

θ



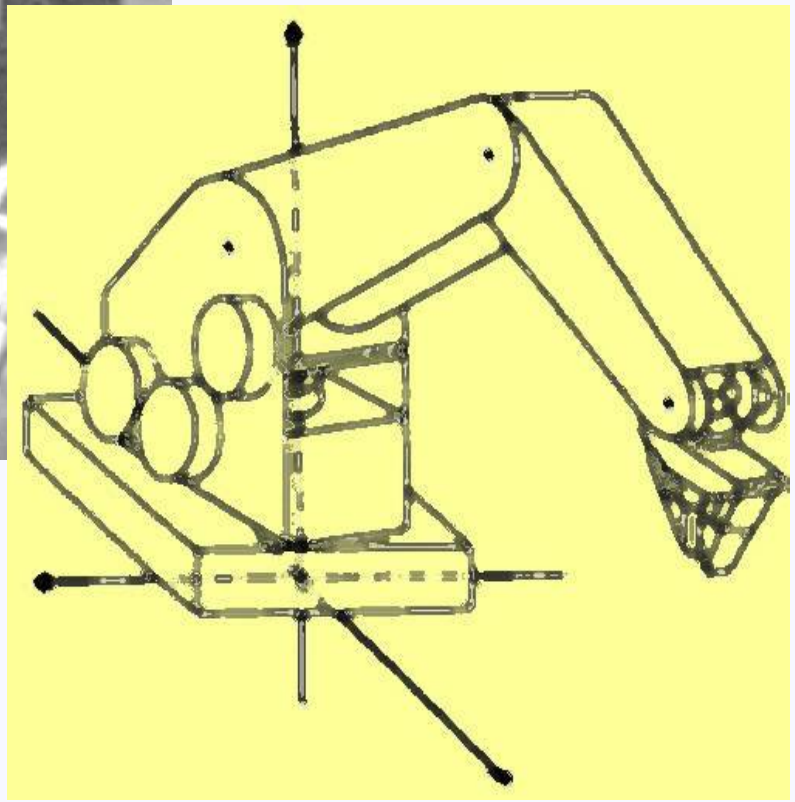
@ G. Gini 2009

# Off-line programming tools

- **RobotStudio:**
  This software is developed by ABB and bases on a VirtualController. This is a digital copy of the real ABB S4 C robotcontroller. This copy uses the real robotprograms and configfiles and realizes a real simulation.
- **RobCAD:**
  Tecnomatix' simulations system has another philosophy than RobotStudio or KR-SIM. RobCAD (in the new version it is called "eM Power") can handle robots from several manufactors. So called "RCS Module" are used, which are produced by the robotmanufactors. The hardware requirements of this software are very much higher than these of the ofter mentioned products.
- **KR-SIM:**
  This is formerly developed by the company AnySim in teamwork with the robotmanufactor KUKA. It is actually run on two PCs. One machine simulates the original robotcontroller (without manipulator). The KUKA KR is based on a standard Windows 95 PC with a so called MFC-Card (Multi function card) in it, that does the whole axis control. The secound PC does the graphical illustration of the simulation. On this PC the actual "KR-SIM" Software is installed. The two PCs are linked over the normal ethernet.

@ G. Gini 2009

# Il linguaggio attuatori:
## ARM-BASIC



- Minimover, 5gdl
- connessione con PC
- estensione di BASIC.
- aggiunte sei istruzioni per il comando del robot. Il comando avviene a livello degli attuatori cioè si comanda il numero di passi da far compiere ai motori passo-passo che muovono i giunti

@ G. Gini 2009

# istruzioni

- @ **STEP D, J1, J2, ..., J6**: Muove i giunti specificando i passi ai motori. J1..J5 sono i giunti del braccio, J6 la mano. D , delay, serve per specificare la velocità (tempo fra un passo e il successivo). La traiettoria è interpolata linearmente nello spazio dei giunti.
- @ **CLOSE D** Chiude la mano.
- @ **SET D** Impone controllo manuale
- @ **RESET D** Azzeramento. Ritorna al movimento automatico.

- @ **READ V1, V2, ..., V6** Legge le posizioni dei giunti e le memorizza nelle variabili Vi
- @ **ARM N** Seleziona la porta su cui è collegato il robot nel caso si comandino più bracci (N=(0,1)).

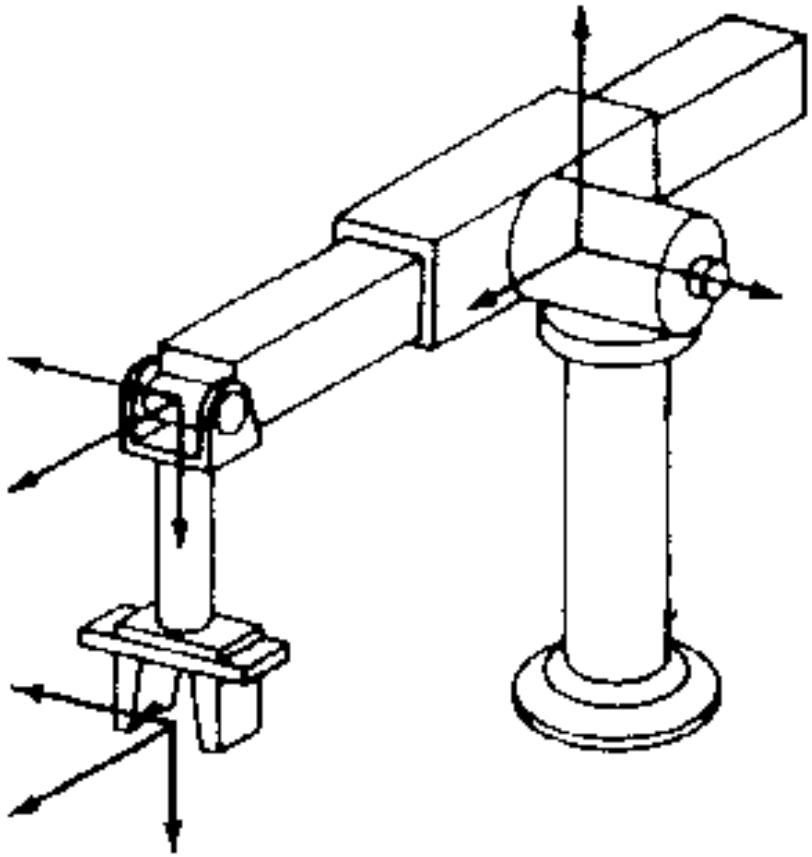
# Es: misurare oggetti nella mano

– nello spazio degli attuatori; cinematica con sottoprogrammi.

```
110 @CLOSE 20
120 @RESET
130 @STEP 20,0,0,0,0,0,500
140 PRINT “Premere INVIO per misurare”
150 INPUT D
160 @CLOSE 20
170 @READ D,D,D,D,D,G
180 T1=G/309
190 IF T1<0.015 THEN 230
200 PRINT “Spessore =”,T1
210 PRINT “---------------------------”
220 GOTO 130
230 PRINT “Spessore vicino a zero”
240 GOTO 210
```

@ G. Gini 2009

# Il linguaggio cartesiano: AL
## (Assembly Language)

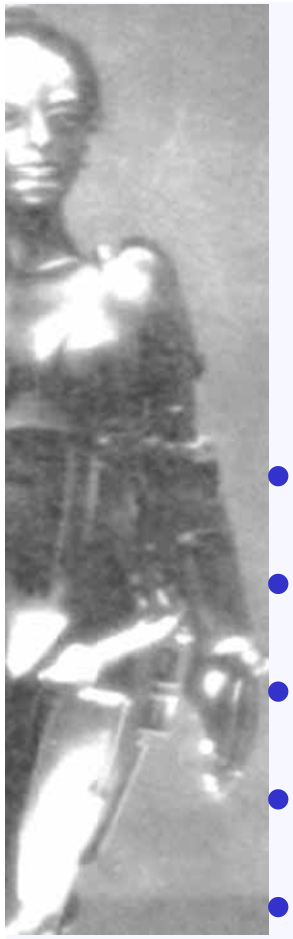


- Linguagio Algol-like
- per multirobot
- con sensori
- programmazione strutturata

© S. Cilli 2005

# STANFORD BLUE ARM, 1979


@ G. Gini 2009



# Tipi di dati supportati

- scalare
- vettore
- rotazione
- frame
- trasformazione
- evento
- array, etc…

@ G. Gini 2009

# Scalare

- *Dichiarazione: SCALAR S1;*
- *assegnamento:* S1<— 2;
- *Rappresentato come* numero in virgola mobile.
- supporta le operazioni **+ - * / ^ SQRT SIN COS TAN ASIN ACOS ATAN LOG EXP** ecc...
- Predefiniti: bhand, … apertura mano
- possibilità di definire la dimensione (distance, …)

@ G. Gini 2009

# vettore

- *Dichiarazione*: vector V1;
- *Assegnamento*: V1<-VECTOR(4, 2, 6)*inches;
- È una terna di scalar
- Operazioni: somma, sottrazione, prodotto scalare e vettoriale, modulo, o combinazione con altri tipi

$$|V| = \sqrt{X^2 + Y^2 + Z^2} \qquad V1 \bullet V2 = X1 \cdot X2 + Y1 \cdot Y2 + Z1 \cdot Z2$$

- Predefiniti xhat yhat zhat nilvect

| | |
|---|---|
| xhat←VECTOR(1, 0, 0); | Versore dell'asse X |
| yhat←VECTOR(0, 1, 0); | Versore dell'asse Y |
| zhat←VECTOR(0, 0, 1); | Versore dell'asse Z |
| nilvect←VECTOR(0,0,0); | Origine |

@ G. Gini 2009

esempio:

```
VECTOR V;
DISTANCE VECTOR DV1, DV2, DV3;
SCALAR S;
DISTANCE SCALAR DS1, DS2;
DS1←2*inches;
DV1←VECTOR(4, 2, 6)*inches;
DS2←DV1•yhat;
V←VECTOR(2, 1, 3);
V←V-zhat;
DV2←VECTOR(3, 0, 4)*inches
DS1←|DV2|;
DV3←VECTOR(4*inches, 2*inches, 6*inches);
```

Anche al tipo vettore è possibile associare una dimensione.

*Inches* è una costante predefinita. *VECTOR* è il *costruttore*.

V1•V2 è il prodotto scalare fra vettori

|V| è il modulo del vettore V.

@ G. Gini 2009

# rotazione

- *Dichiarazione:* rot R1;
- *Assegnamento:* R1<-ROT(xhat, 90*deg);
- E’ asse cartesiano, angolo
- Operazioni: somma, sottrazione, prodotto, o combinazione con altri tipi;
- Predefiniti nilrot

| | |
|---|---|
| V←AXIS (R);<br>a←\|R\|; | Restituisce in V l’asse di rotazione asserot.<br>Restituisce in a l’angolo di rotazione alfa |

@ G. Gini 2009

# frame

- *Dichiarazione*: frame F1;
- *Assegnamento*: F1<-FRAME(R, V)
- 6 variabili cartesiane memorizzate in matrice
- Operazioni: f*f, ……., prodotto scalare e vettoriale, modulo, o combinazione con altri tipi
  - ORIENT (F), Restituisce l'orientamento di F
  - POS(F), Restituisce la posizione di F
  - V1←V WRT F; costruisce un vettore V1 in station con lo stesso orientamento di V in F.
- Predefiniti barm, bpark, …

@ G. Gini 2009

# trans

- *Dichiarazione*: trans T1;
- *Assegnamento*: T1<-TRANS(R, V)
- 6 variabili cartesiane memorizzate in matrice
- Operazioni: t*f, ……., prodotto scalare e vettoriale, modulo, o combinazione con altri tipi
  - Rappresenta una trasformazione fra sistemi

| | |
|---|---|
| T2←F1→F2; | Date due frame F1 e F2 l'operatore → restituisce la trasftrasformazione che permette di passare da F1 a F2. |
| T3←INV(T2); | Restituisce la trasformazione inversa di T2. |

@ G. Gini 2009

# Struttura del programma

***Esempio:***

visualizza le posizioni di due dei bracci nella stazione di lavoro e poi ne calcola la distanza:

```
BEGIN
DISTANCE SCALAR S1;
DISTANCE VECTOR V1;
PRINT (“Il braccio BLUE è nella posizione”, barm);
PRINT (“Il braccio YELLOW è nella posizione”, yarm);
V1←POS(barm)-POS(yarm);
S1← | V1 | ;
PRINT (“La distanza fra il braccio YELLOW e quello BLUE è
”, S1, “inches”);
END
```

@ G. Gini 2009

# Movimenti del braccio

***MOVE <braccio> TO <posizione>;***

- <*braccio*> (es. *barm, yarm* ecc...) e' un sistema di coordinate la cui origine è posta fra le dita con l'asse Z nella direzione di approccio
- <*posizione*> è la frame in cui sarà spostato il riferimento posto nella mano.
  - esempio :
  - MOVE barm TO bpark;
    - il movimento avviene alla massima velocità e con la coppia massima sui motori.
  - *grinch* identifica la posizione attuale del braccio
    - MOVE barm TO Ø - 2*zhat*inches;
      - sposta il braccio dalla posizione attuale di due pollici lungo l'asse Z.

@ G. Gini 2009

# Movimento mano

- *OPEN <hand> [TO <scalar_exp>];*
- *CLOSE <hand> [TO <scalar_exp>];*
- *CENTER <arm>;*

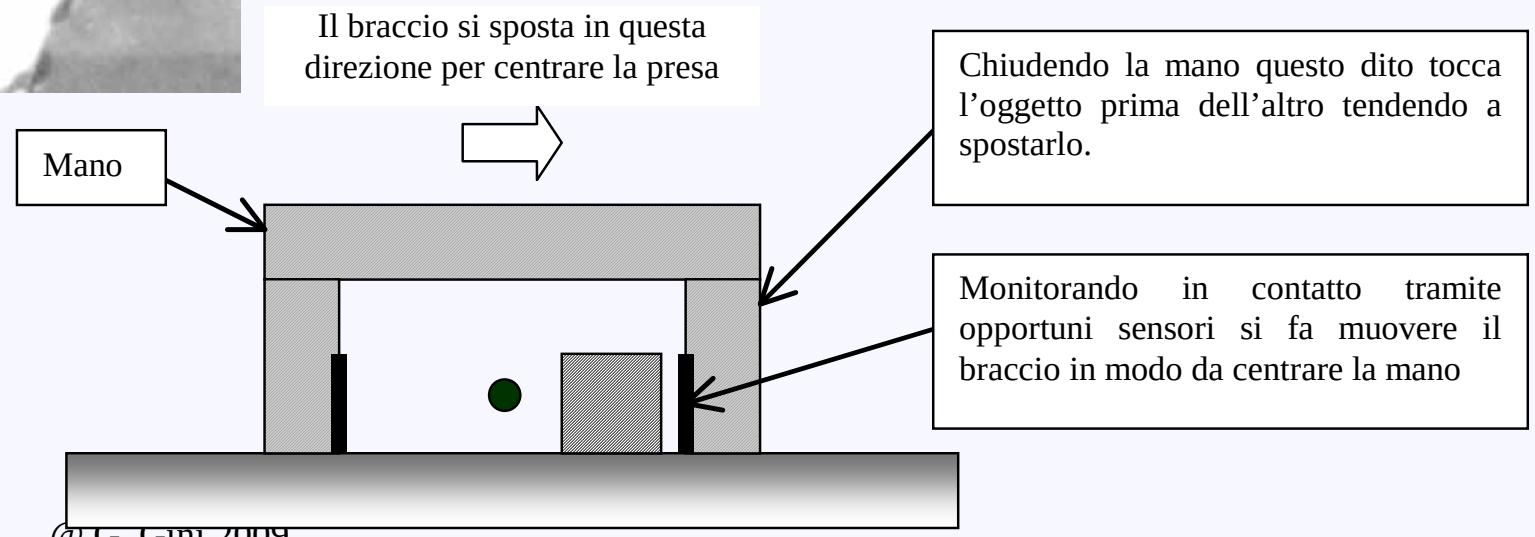
  – *Le prime aprono o chiudono le dita che si muovono contemporaneamente; usando bhand si puo' poi leggere l'apertura*

  – *La terza esegue piccoli spostamenti del braccio per ri-centrare la presa*

@ G. Gini 2009

# Ri-centrare la mano

- Il procedimento è iterativo: se un dito tocca si calcola un nuovo punto di presa con l'origine spostata su y nella direzione in cui è avvenuto il contatto di una quantità pari al tratto di chiusura eseguito




@ G. Gini 2009

# Specifica traiettorie - 1

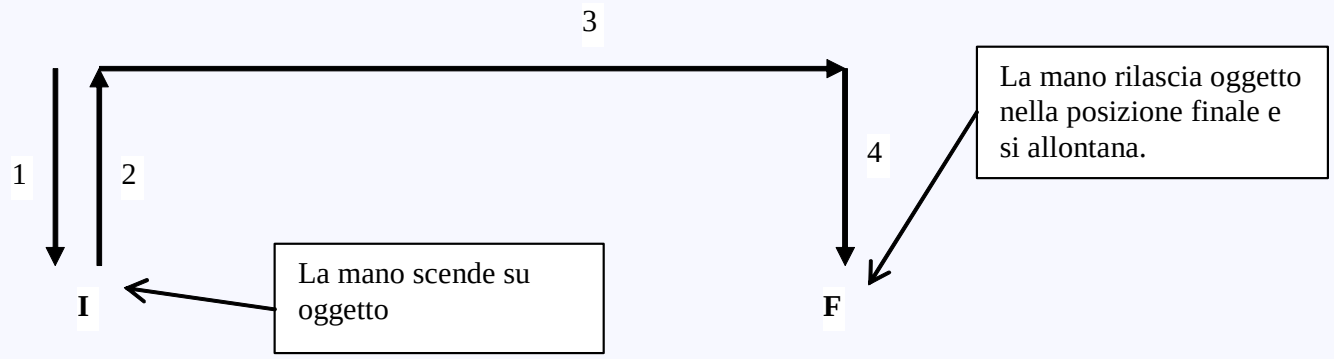
## *MOVE <braccio> TO <posizione>*

*WITH APPROACH=appr*

*WITH DEPARTURE=depr*;

*WITH APPROACH* = la mano deve passare per il punto di avvicinamento.

*WITH DEPARTURE* = la mano deve passare per il punto *di allontanamento* allontanandosi dalla posizione di start.




@ G. Gini 2009

- MOVE barm TO target
  WITH APROACH = **frame**(R1, V1)
  WITH DEPARTURE = **frame** (R2, V2);

la frame di passaggio: = target * (R1, V1) ; = start *(R2, V2)

- MOVE barm TO target
  WITH APROACH = **vector**(x1, y1, z1)
  WITH DEPARTURE = **vector**(x2, y2, z2);

la frame di passaggio ha lo stesso orientamento:

= target+vector(x1,y1,z1) WRT target; = start+vector(x2,y2,z2) WRT start

- MOVE barm TO target
  WITH APROACH = **20**
  WITH DEPARTURE = **20**;

la frame di passaggio ha stesso orientamento e origine spostata lungo z:

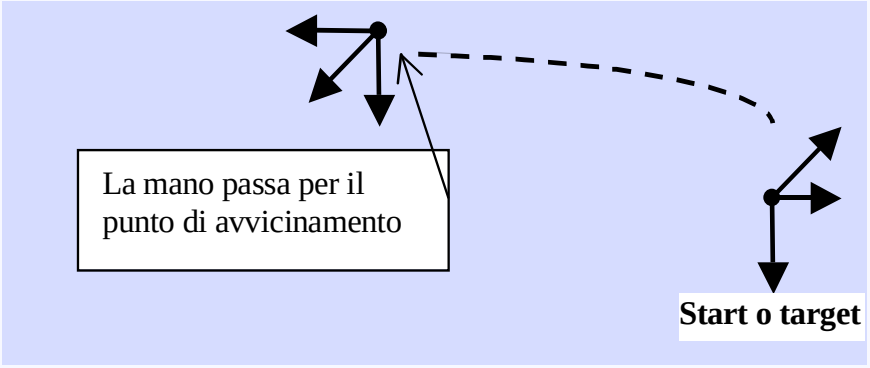
= target + 20*zhat WRT target; = start + 20*zhat WRT start

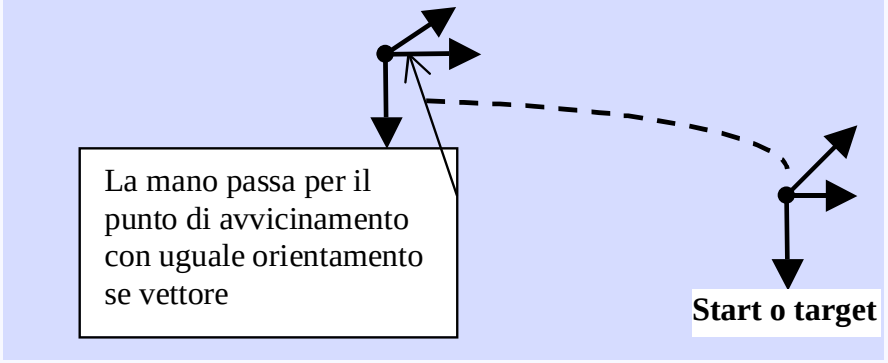
*v WRT f = ORIENT (f) * v = (f*v) - POS (f)*

@ G. Gini 2009

# Default: 3 su asse z e sua eliminazione

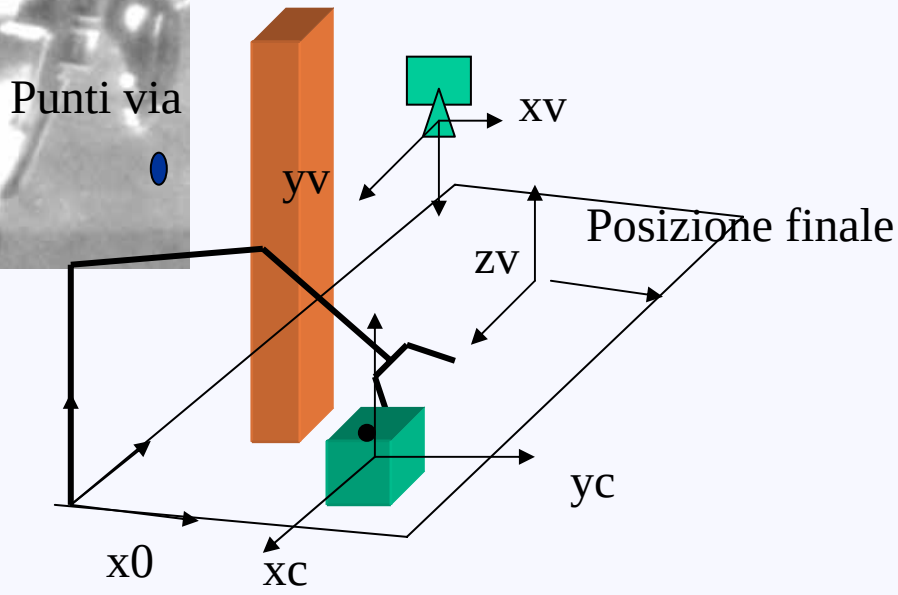
- MOVE barm to point
  - WITH APPROACH=nildeproach
  - WITH DEPARTURE=nildeproach;
- 
- ≡ MOVE barm to point DIRECTLY




@ G. Gini 2009

# Specifica traiettorie - 2




- ***Per evitare ostacoli:***

***MOVE barm to point***

VIA
punto_di_via;

Il punto di via è preso nel sistema assoluto

@ G. Gini 2009

# Specifica traiettorie - 3

**Velocità, accelerazioni, tempi?**

- Clausole da aggiungere nella istruzione:
- WITH DURATION = n*sec
  - da' il tempo totale del moto
- SLOWLY, ….
  - usano delle percentuali di default

*Non si usano specifiche complete della velocità cartesiana della mano*

@ G. Gini 2009

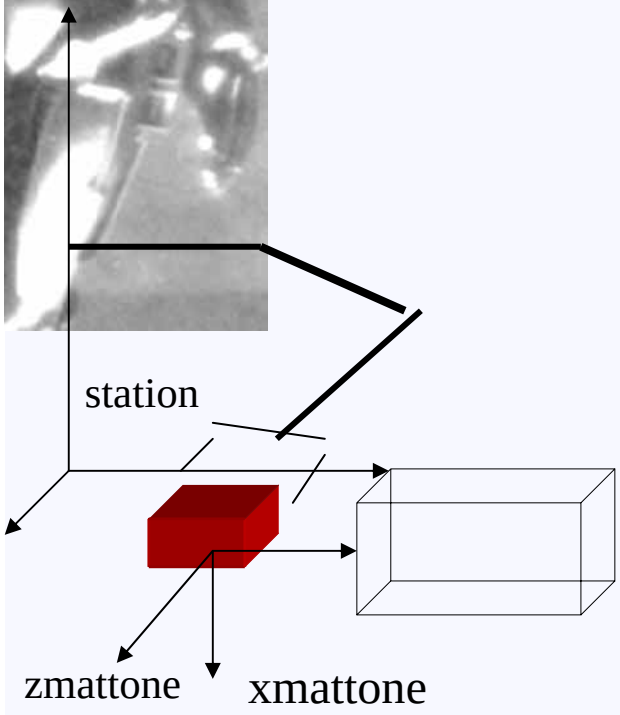
# esempio

BEGIN “Sposta cubo”

```
FRAME cubo, nuova_pos;                          {Definisce la posizione di
cubo←FRAME (ROT(xhat,180*deg),                  presa dall'alto
VECTOR(20, 30, 1)*inches);
nuova_pos←cubo + 10*xhat*inches;                {Definisce la nuova
MOVE barm TO bpark WITH                         posizione}
DURATION 3*sec;
OPEN bhand TO 3*inches;                         {Apre la mano}
MOVE barm TO cube;                              {Sposta il braccio}
CENTER barm;                                    {Afferra il cubo}
MOVE barm TO nuova_pos;                         {Sposta il braccio}
OPEN bhand TO 3*inches;                         {Apre la mano}
MOVE barm TO bpark;                             {Porta il robot in park}
END
```

@ G. Gini 2009

# Esempio – presa laterale




```
BEGIN “Metti mattone nel forno”
FRAME mattone, in_forno, porta_forno;
mattone←FRAME  (ROT(yhat,90*deg),  VECTOR(10,  30,
    3)*inches);
in_forno←FRAME  (ROT(yhat,90*deg),  VECTOR(10,  40,
    3)*inches);
porta_forno←FRAME (ROT(yhat,90*deg), VECTOR(5, 40,
    4)*inches);
MOVE barm TO bpark WITH DURATION 4*sec;
OPEN bhand TO 3.5*inches;
MOVE barm TO mattone WITH APPROACH= -3*inches;
CLOSE bhand TO 1.7*inches
MOVE barm TO in_forno VIA porta_forno
    WITH DEPARTURE = -3*xhat*inches;
OPEN bhand TO 3.5*inches;
MOVE barm TO bpark VIA porta_forno;
END
```

@ G. Gini 2009

# Estensioni: sensori esterni e modelli del mondo

- I movimenti possono essere modificati in risposta alle letture di sensori esterni;
  - le traiettorie possono essere ricalcolate
- I movimenti possono essere specificati in termini di oggetti e non di mano del robot.
  - Si può controllare la consistenza dei movimenti e dei modelli del mondo, si può cambiare facilmente il layout, si può cambiare il robot, …

@ G. Gini 2009