A Kinematic Model for the iCub

Semester project final presentation

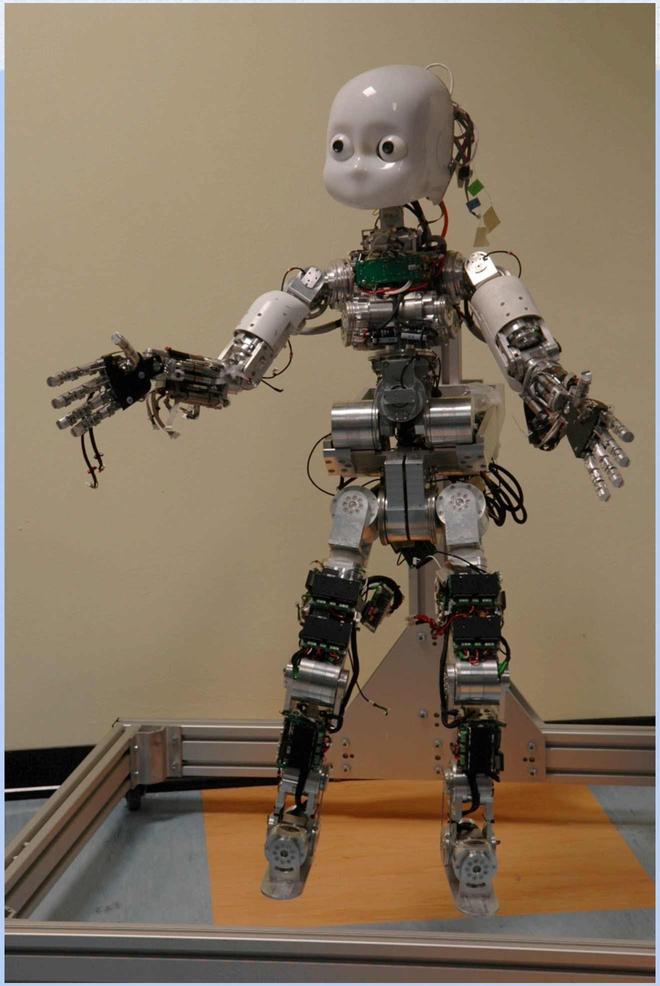
Julia Jesse

January 2009

INTRODUCTION

- **Goal of the project** :
  kinematic model for the iCub using KDL

1. Model under Webots
2. Forward position kinematics
3. Inverse position kinematics
4. Results and future work
5. Conclusion

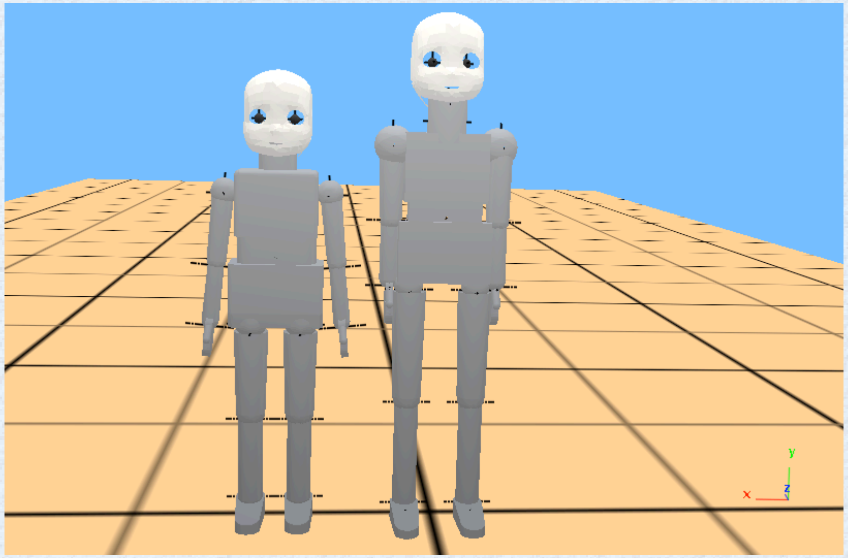


http://robotcub.org

Model under Webots

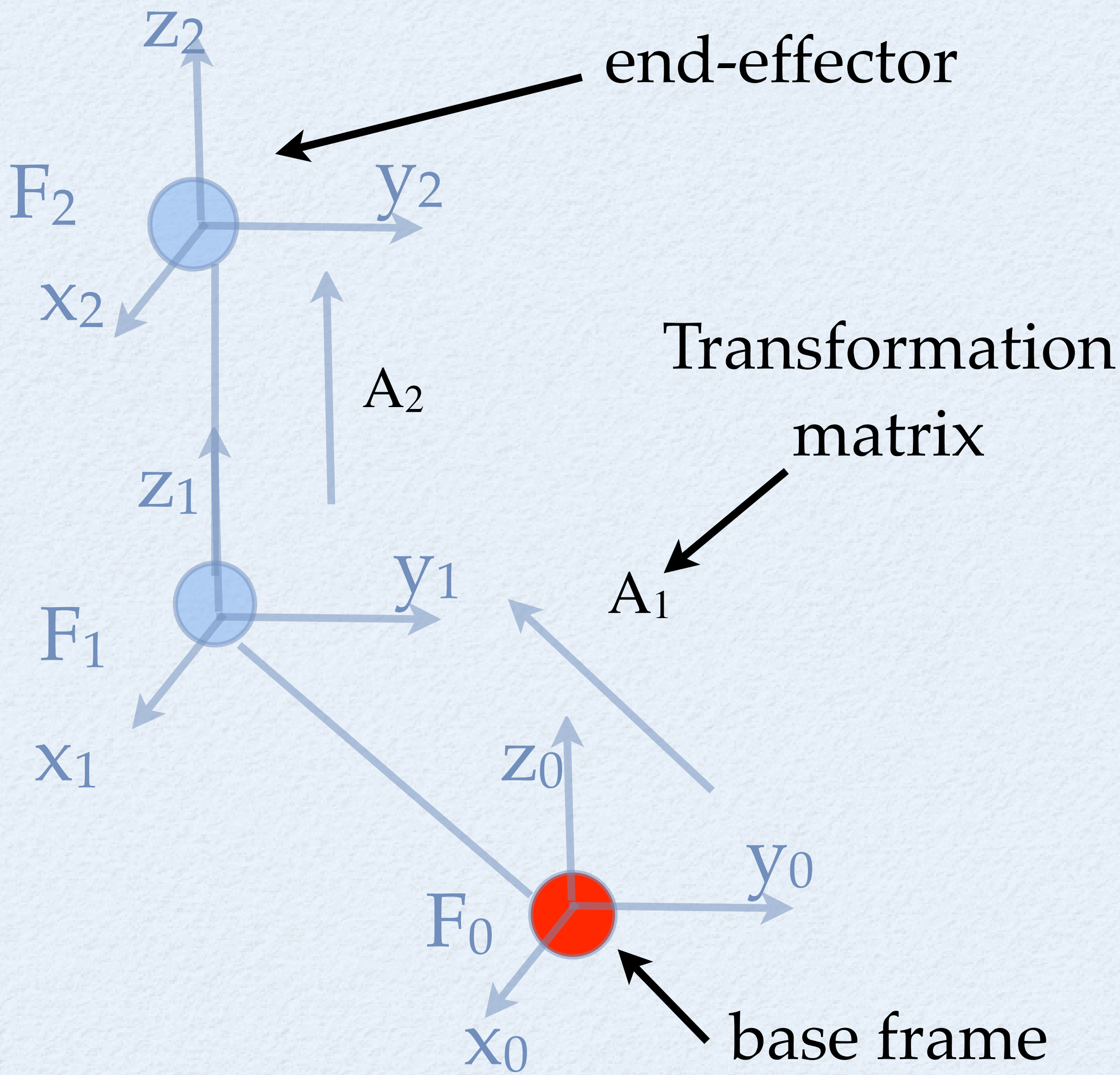
- Model not the same as the official model on http://eris.liralab.it/icubfowardkinematics

- Updates :
    - length of the limbs
    - order of the torso joints
    - eyes
    - ankle roll
    - new origin : middle of the torso



old v.s. new

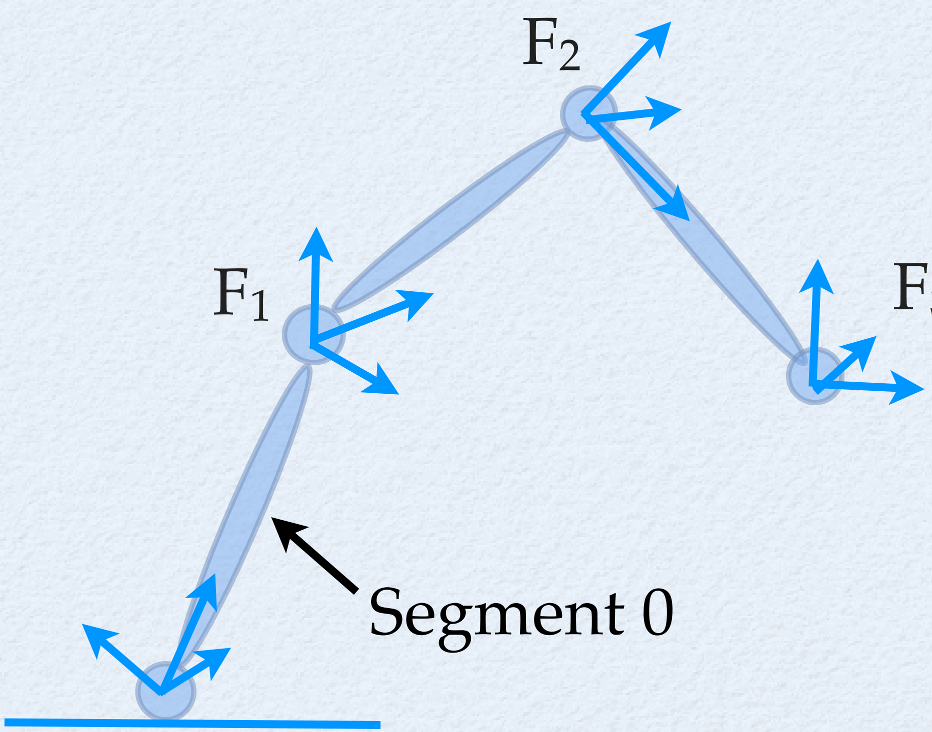
Forward Position Kinematics



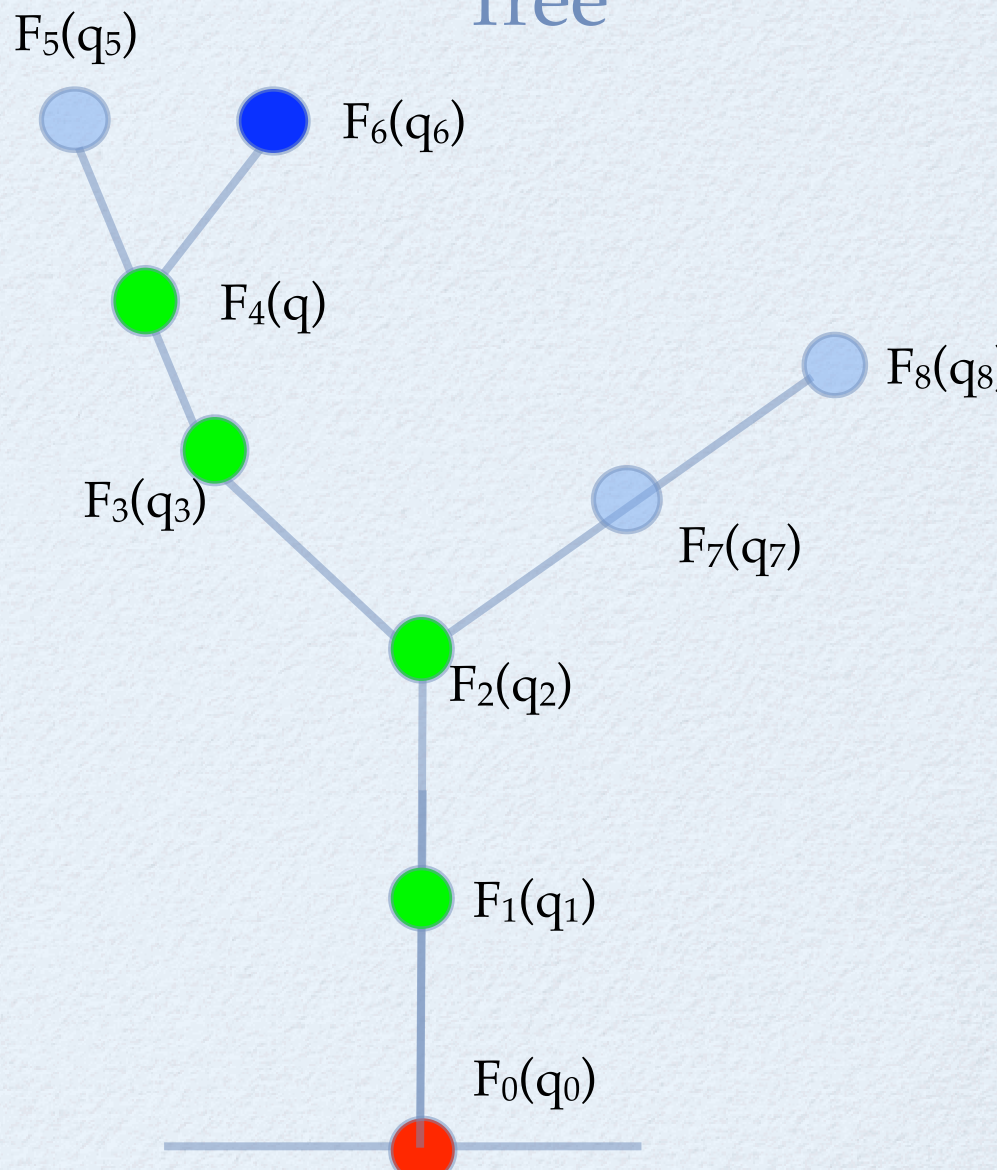
- Finding the position of the end-effector knowing the joint values
- Unique solution
- A_i = transformation matrix from F_{i-1} to F_i
- $T_{ij} = A_{i+1} A_{i+2} \dots A_j$ = transformation matrix from F_i to F_j

KDL's Forward Position Kinematics

Chain



Tree



Chain Forward Kinematics : Results

	Torso pitch	Torso roll	Torso yaw	Shoulder pitch	Shoulder roll	Shoulder yaw	Elbow	Forearm	Wrist pitch	Wrist yaw
θ	0	0	0	0	0	0	1.85	0	0	0

(a) Initial joint values for the right arm chain

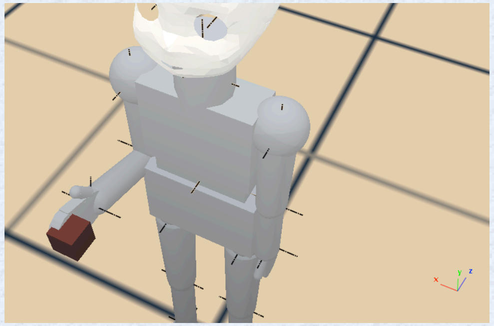
$$\begin{bmatrix} 6.71^{-17} & 2.96^{-16} & -1 & 0.0941161 \\ 0.27559 & -0.961275 & -2.4^{-16} & 0.0636638 \\ -0.961275 & -0.27559 & -1.39^{-16} & -0.201513 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) KDL's end-effector frame

$$\begin{bmatrix} 0.0 & 0.0 & -1.0 & 0.09411608 \\ 0.27559 & -0.96128 & -0.0 & 0.06366380 \\ -0.96128 & -0.27559 & -0.0 & -0.20151324 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c) Matlab's end-effector frame

http://eris.liralab.it/icubfowardkinematics



(d) Webots cube position: (0.0941161, 0.0636638, -0.201513)

Tree Forward Kinematics: Results

- Same results as with the chain
- Results for the left arm elbow joint set to 1.85

$$\begin{bmatrix} 1.25^{-16} & 6.87^{-17} & -1 & -0.0941161 \\ 0.27559 & -0.961275 & -3.17^{-17} & 0.0636638 \\ -0.961275 & -0.27559 & -1.39^{-16} & -0.201513 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KDL's chain end-effector frame

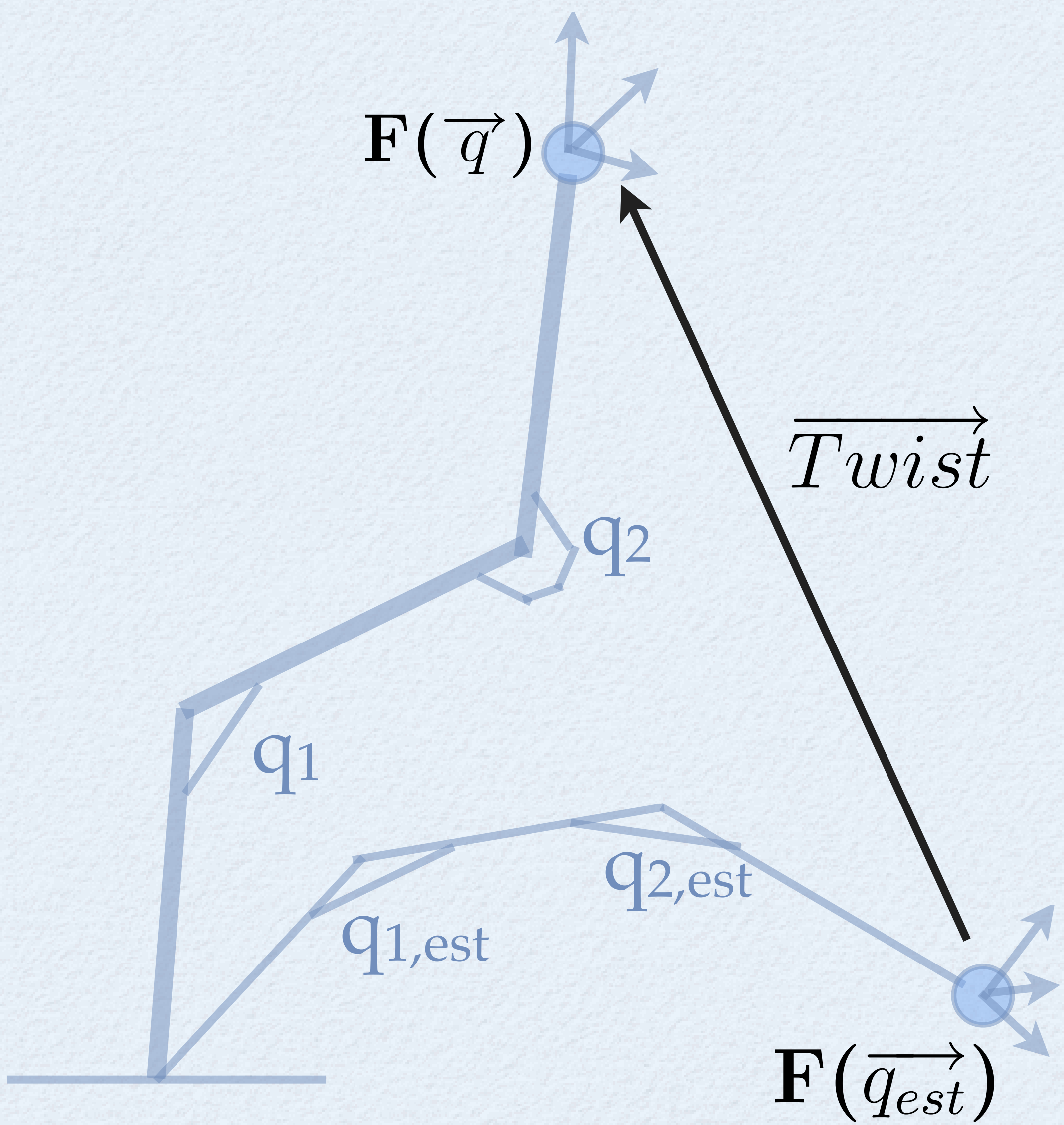
$$\begin{bmatrix} 1.25^{-16} & 6.87^{-17} & -1 & -0.0941161 \\ 0.27559 & -0.961275 & -3.17^{-17} & 0.0636638 \\ -0.961275 & -0.27559 & -1.39^{-16} & -0.201513 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KDL's tree end-effector frame

Inverse Postion Kinematics

- Finding the joint angles knowing the position and the orientation of the end-effector.
- Multiple solutions = redundant system
  - If we have more than 6 DoFs, i.e more DoF than constraints.
- Algorithm for a chain:
  - Based on the Newton-Raphson iteration
  - Takes the joint limits into account
  - Needs inverse velocity kinematics

KDL's Inverse Kinematic for a Chain



- Algorithm based on the Newton-Raphson iterations, with Joint Limits

KDL's Inverse Velocity Kinematics for a Chain

- KDL implements a “weighted damped least square” algorithm
- Need notions of :
  - Jacobian
  - Pseudo-inverse
  - Singular Value Decomposition (SVD)

Jacobian

- Twist = end-effector velocity :

$$\overrightarrow{T} = \frac{d\overrightarrow{x}}{dt} = \frac{\partial A(\overrightarrow{q})}{\partial \overrightarrow{q}} \frac{d\overrightarrow{q}}{dt}$$

  where $A(\overrightarrow{q}) = \overrightarrow{x}$ is the position of the end-effector calculated with the forward kinematics

- Jacobian : $\frac{\partial A(\overrightarrow{q})}{\partial \overrightarrow{q}}$ => $\overrightarrow{T} = \overrightarrow{\dot{x}} = J(\overrightarrow{q})\overrightarrow{\dot{q}}$

  - relation between the joint velocities and the cartesian space velocity

  - linear relationship between $\overrightarrow{\dot{q}}$ and $\overrightarrow{T}$.

Singular Value Decomposition (SVD)

- We want the inverse joint velocity, i.e

$$\overrightarrow{\dot{q}} = J^{-1}(\overrightarrow{q})\overrightarrow{T}$$

- A necessary condition for the Jacobian to be invertible :
    - square matrix
    - i.e. no redundancy

- If Jacobian not invertible => pseudo-inverse $J^*(\overrightarrow{q})$

$$\overrightarrow{\dot{q}} = J^*(\overrightarrow{q})\overrightarrow{T}$$

SVD and Pseudo-inverse

- **Singular Value Decomposition (SVD) :**
  - M is a $n_x m$ matrix
  - M has singular values $\sigma_1 \cdots \sigma_n$

    => M can be decomposed in:

$$M = U \Sigma V^T$$

where $\mathbf{U} \in \Re^{n_x n}$

$\mathbf{V} \in \Re^{m_x m}$

such that $\Sigma \in \Re^{n_x m}$ has the form

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_n & 0 & \cdots & 0 \end{bmatrix}$$

- **Pseudo-inverse of M:**

$$M^* = V \Sigma^* U^T$$

where Σ^* is the pseudo-inverse of Σ and has the form

$$\Sigma^* = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

- if $\sigma_i = 0$: can't calculate the pseudo-inverse => **Singularity problem**.

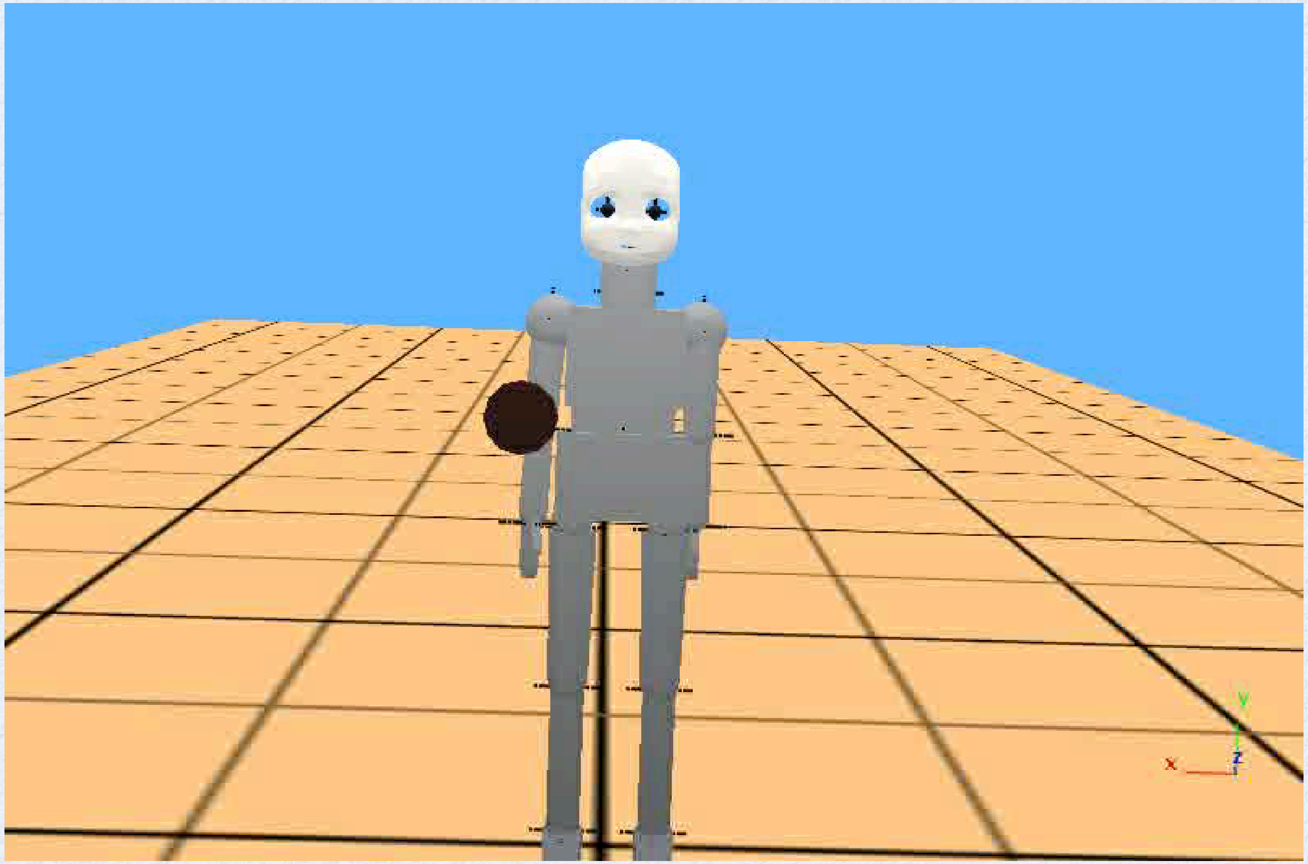
Singularity Problem

- Robotics : can not move in a given direction anymore
- Solution : damped least square
  - λ = damping parameter
  - replace $\frac{1}{\sigma_i}$ by $\frac{\sigma_i}{\sigma_i^2+\lambda}$
  - λ increases => approximation error for the pseudo-inverse increases
  - λ decreases => damping decreases and may not avoid a singular configuration

Inverse Velocity

- Weighted damped least square algorithm
  - Weighted
    - put some weight on given joints such that they don't move
  - Damped least square
    - damping parameter λ
    - least square = minimizes the joint velocities such that we get the nearest solution
- Algorithm :
  1. Weighted Jacobian
  2. SVD
  3. Joint velocities

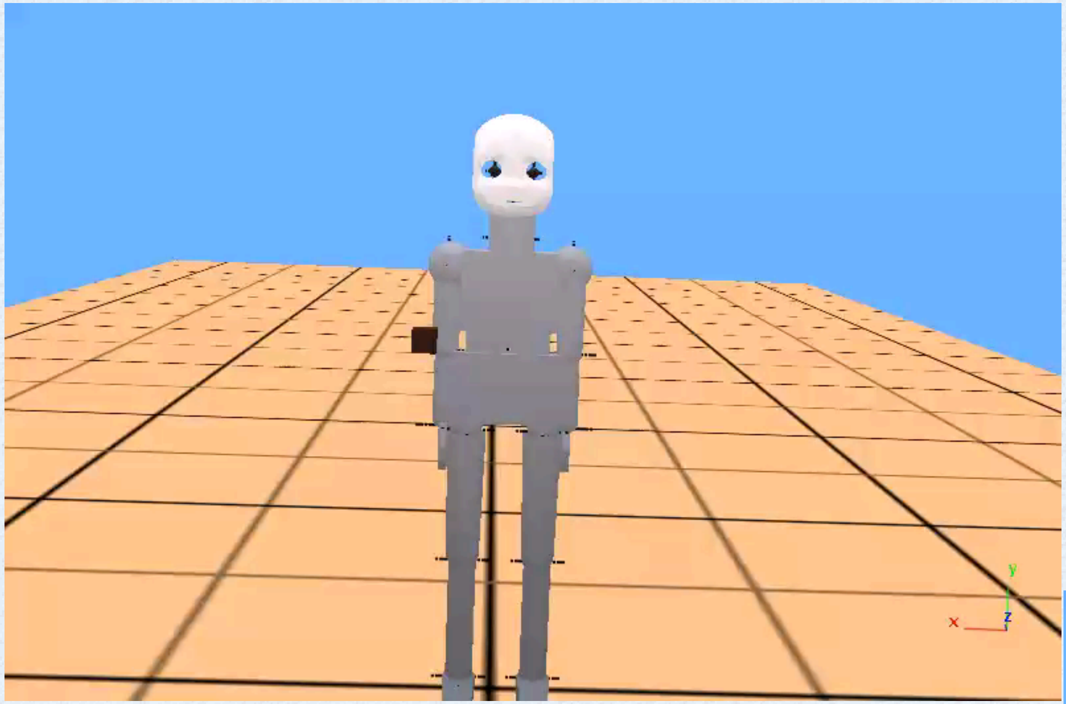
Chain Inverse Kinematics : Results



- $\lambda = 0.2$
- $\theta_{i,0} = 0$
- no torso

- problem due to :
  - local minima?
  - singularities?
  - joint limits?

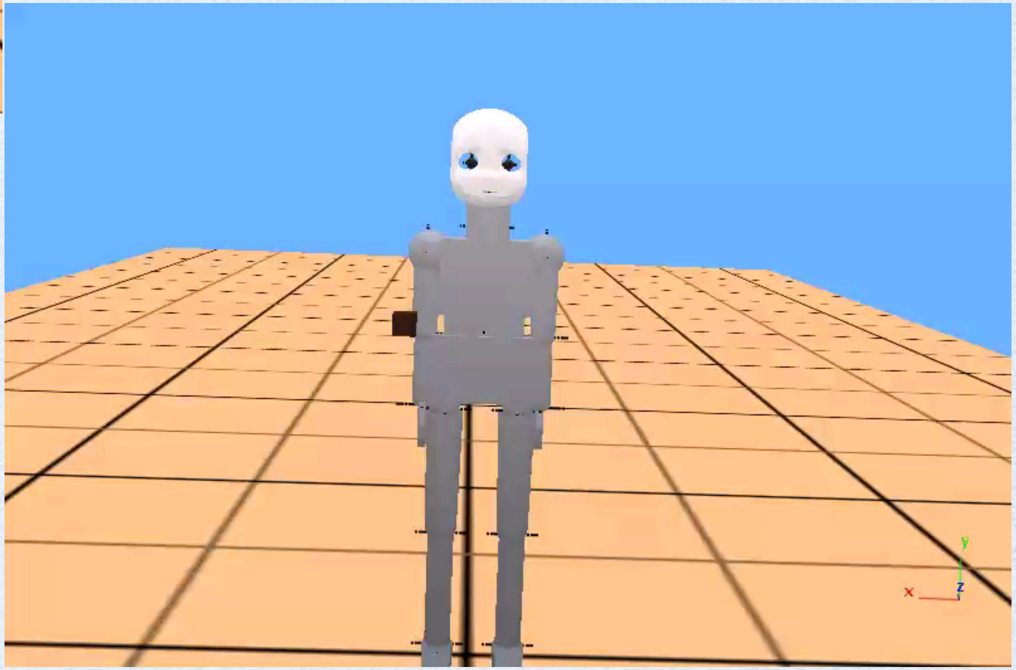
DIFFERENT DAMPING λ



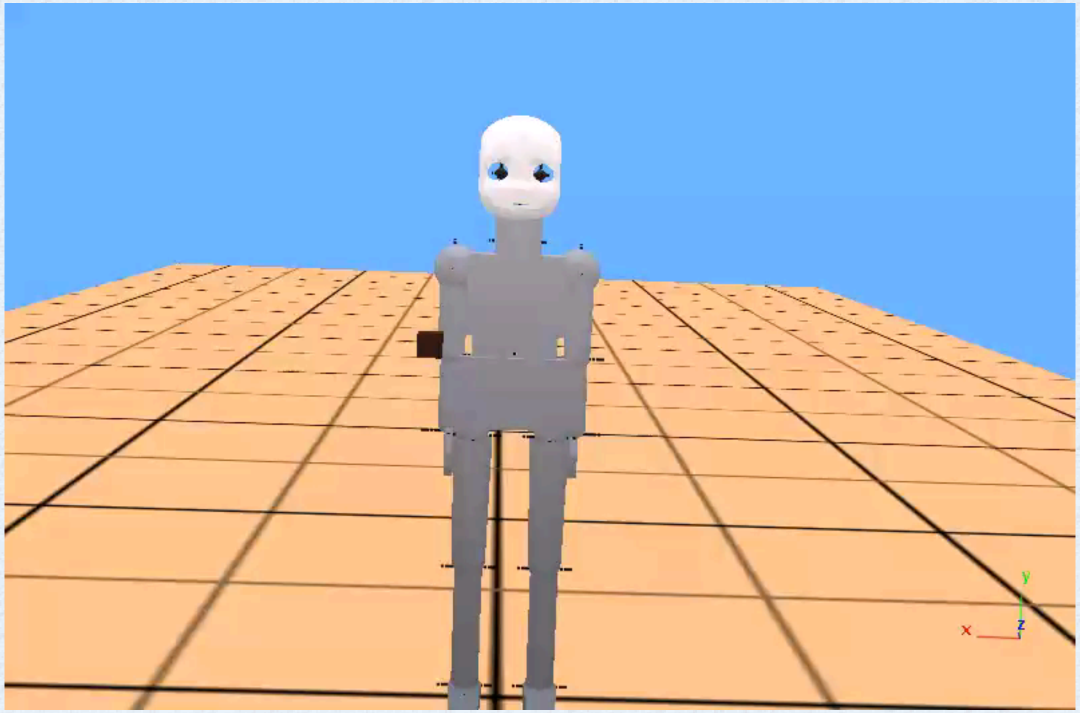
- $\lambda = 0.2$
- $\theta_{i,0} = 0$

- $\lambda = 0$
- $\theta_{i,0} = 0$

  => singular configuration

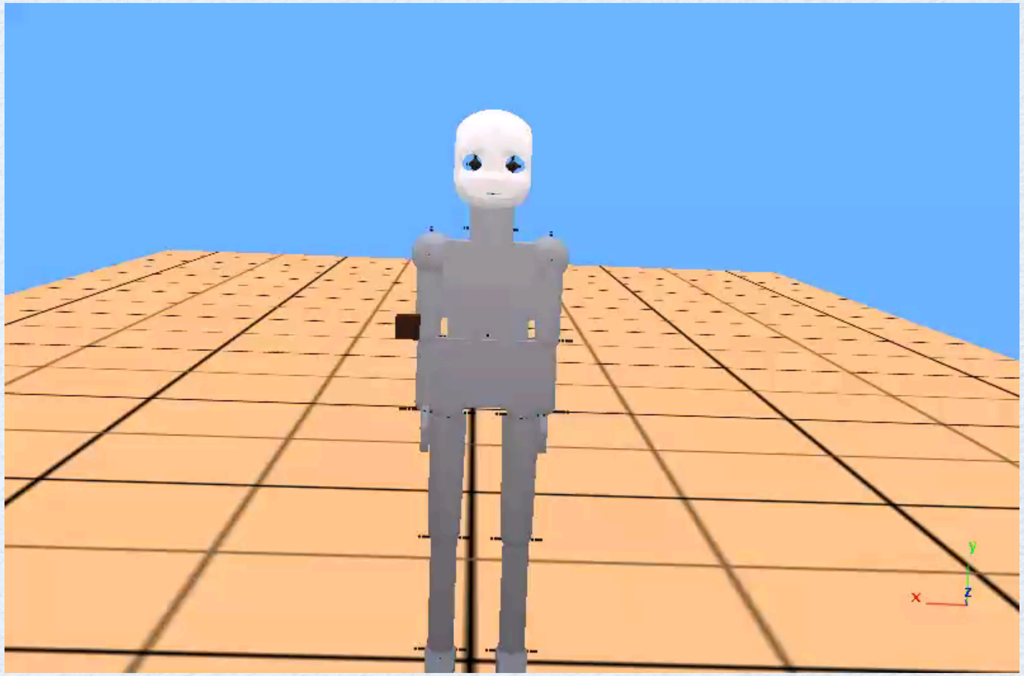


Different initial joint values

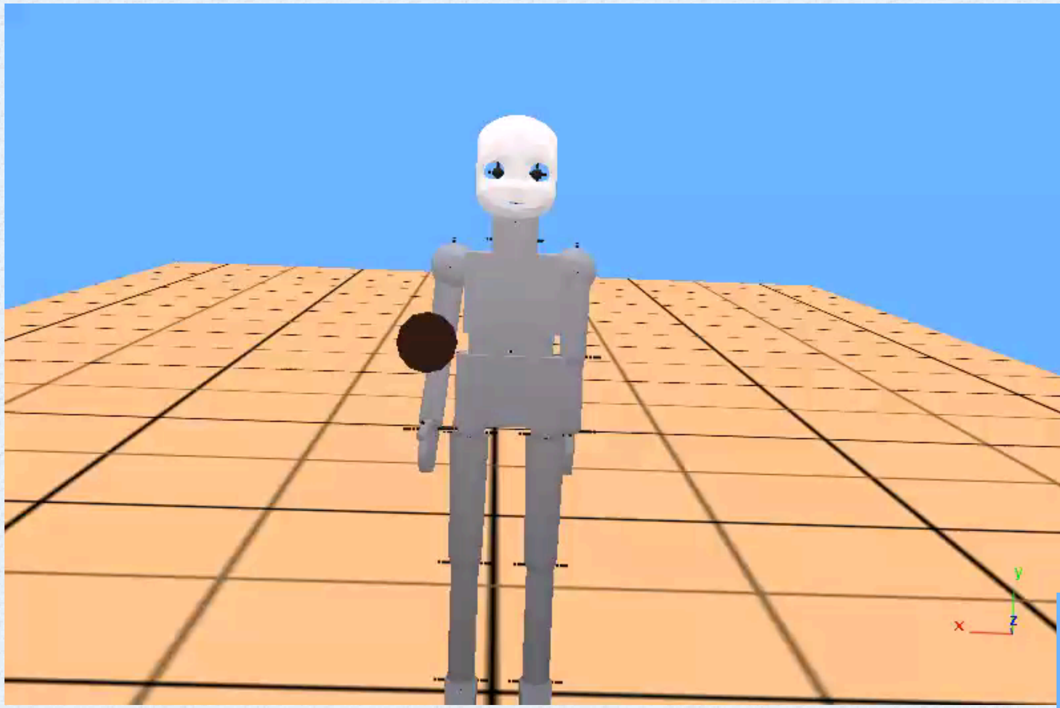


- $\lambda = 0.1$
- $\theta_{i,0} = 0$
- $\theta_{elbow,0} = 0.3$

- $\lambda = 0.1$
- $\theta_{i,0} = 0$

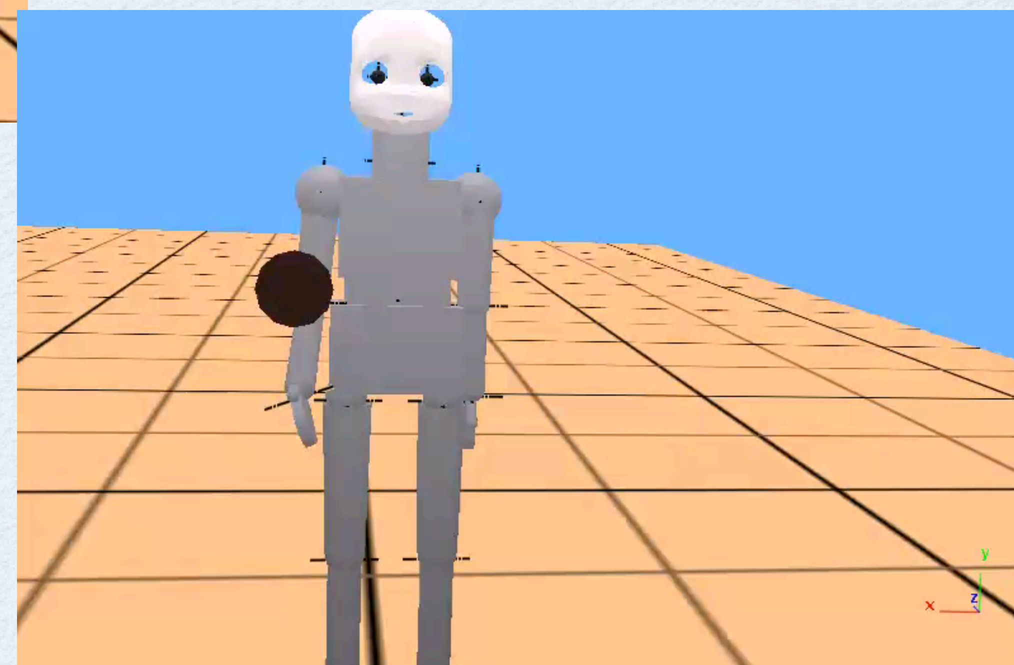


Influence of joint limits

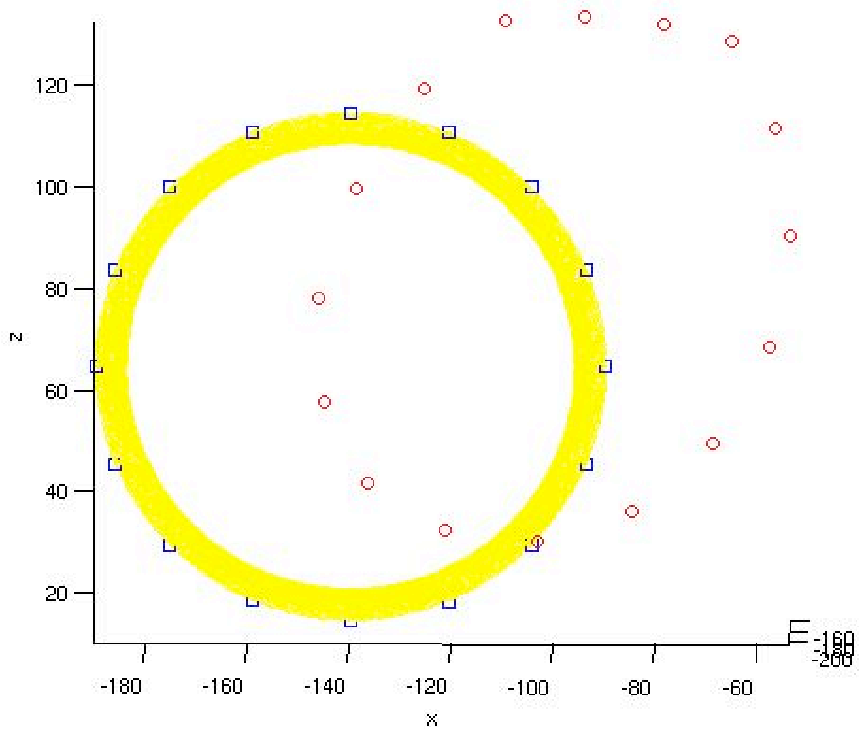


- **No joint limits**
- $\lambda = 0.1$
- $\theta_{i,0} = 0$
- $\theta_{elbow,0} = 0.3$

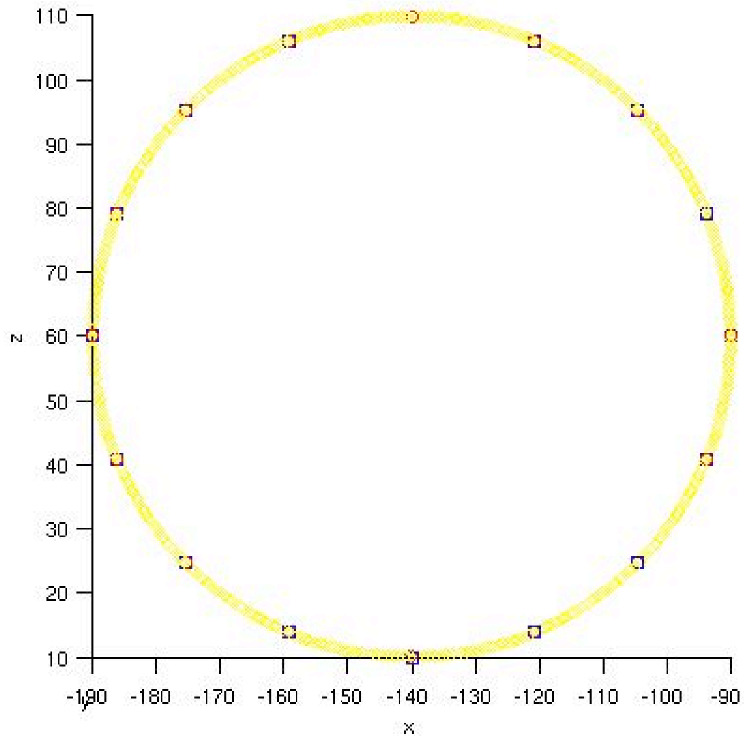
- **With joint limits**
- $\lambda = 0.1$
- $\theta_{i,0} = 0$
- $\theta_{elbow,0} = 0.3$



Chain Inverse Kinematics



with joint limits



without joint limits

yellow = original circle
❑ = calculated input circle points
❍ = circle points calculated by KDL

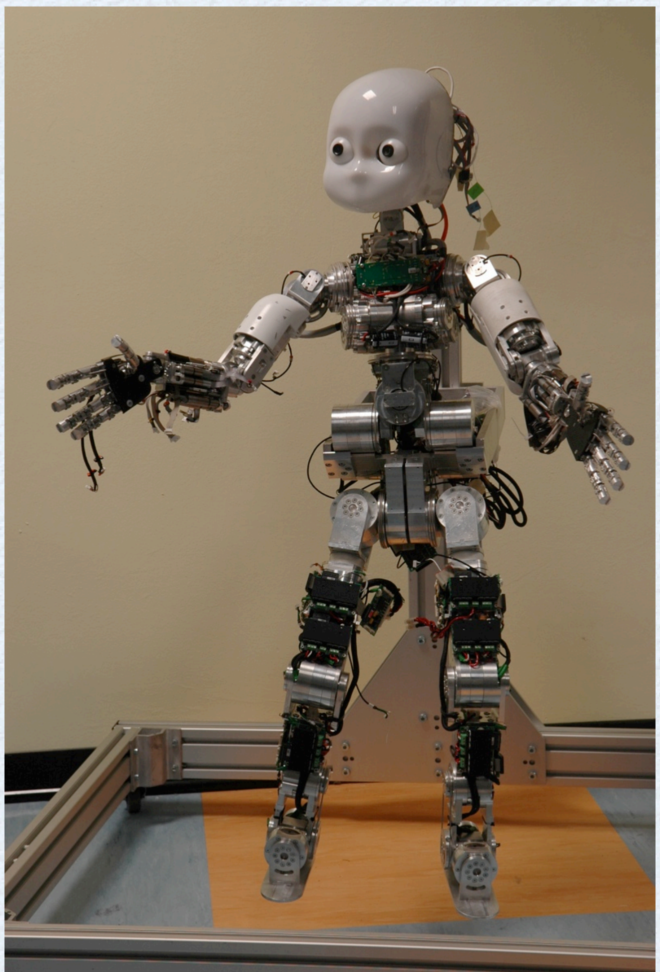
FUTURE WORK

- Improve Webots' simulation
- Inverse position with torso moving
- Other way to test the joint limits
- Other orientation for the end-effector
- Test the inverse position kinematics for trees
- Example of future applications:
    - Stability during locomotion
    - Kinematic constraints

CONCLUSION

- iCub model under Webots updated
- Forward position kinematics works well for chains and trees
- Inverse position kinematics works well for chains:
    - iCub can reach a point
    - iCub can draw a circle

QUESTIONS ?



http://robotcub.org