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 not the same as the official model on http://eris.liralab.it/icubforwardkinematics

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
- $A_i = \text{transformation matrix from } F_{i-1} \text{ to } F_i$
- $T_{ij} = A_{i+1} A_{i+2} \ldots A_j = \text{transformation matrix from } F_i \text{ to } F_j$
KDL's Forward Position Kinematics

Chain

Tree

Segment 0

F₀(q₀)

F₁(q₁)

F₂(q₂)

F₃(q₃)

F₄(q)

F₅(q₅)

F₆(q₆)

F₇(q₇)

F₈(q₈)
### Chapter 6: Results

This chapter presents the results found by testing KDL’s capabilities to model the iCub and to use it for controlling Webots’ new model presented on Chapter 5.

#### 6.1 Forward Kinematics for a Chain

The results for the chain including the torso and the right arm show that the forward kinematics for a chain works correctly. Indeed, given some initial joint values, we obtain the same end-effector for the calculations made under KDL and under Matlab using the "official" code as explained in 5.2.

When we give the joint values to the Webots model, we can see that it reaches the calculated position. Figure 6.1 shows one of the results made for the forward kinematics for a chain.

![Webots cube position](http://eris.liralab.it/icubforwardkinematics)

**Figure 6.1:** Chain forward kinematics

<table>
<thead>
<tr>
<th>θ</th>
<th>Torso pitch</th>
<th>Torso roll</th>
<th>Torso yaw</th>
<th>Shoulder pitch</th>
<th>Shoulder roll</th>
<th>Shoulder yaw</th>
<th>Elbow</th>
<th>Forearm</th>
<th>Wrist pitch</th>
<th>Wrist yaw</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(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

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

<table>
<thead>
<tr>
<th></th>
<th>1.25 \times 10^{-16}</th>
<th>6.87 \times 10^{-17}</th>
<th>-1</th>
<th>-0.0941161</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.27559</td>
<td>-0.961275</td>
<td>-3.17 \times 10^{-17}</td>
<td>0.0636638</td>
<td></td>
</tr>
<tr>
<td>-0.961275</td>
<td>-0.27559</td>
<td>-1.39 \times 10^{-16}</td>
<td>-0.201513</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1.25 \times 10^{-16}</th>
<th>6.87 \times 10^{-17}</th>
<th>-1</th>
<th>-0.0941161</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.27559</td>
<td>-0.961275</td>
<td>-3.17 \times 10^{-17}</td>
<td>0.0636638</td>
<td></td>
</tr>
<tr>
<td>-0.961275</td>
<td>-0.27559</td>
<td>-1.39 \times 10^{-16}</td>
<td>-0.201513</td>
<td></td>
</tr>
</tbody>
</table>

KDL’s chain end-effector frame
KDL’s tree end-effector frame
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
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:
  \[
  \vec{T} = \frac{d\vec{x}}{dt} = \frac{\partial A(\vec{q})}{\partial \vec{q}} \frac{d\vec{q}}{dt}
  \]
  where \( A(\vec{q}) = \vec{x} \) is the position of the end-effector calculated with the forward kinematics

- Jacobian:
  \[
  \frac{\partial A(\vec{q})}{\partial \vec{q}} \Rightarrow \vec{T} = \vec{x} = J(\vec{q}) \vec{q}
  \]
  relation between the joint velocities and the cartesian space velocity

- linear relationship between \( \vec{q} \) and \( \vec{T} \).
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 \( \Rightarrow \) pseudo-inverse \( J^*(\overrightarrow{q}) \)

\[ \overrightarrow{\dot{q}} = J^*(\overrightarrow{q}) \overrightarrow{T} \]
SVD AND PSEUDO-INVERSE

- **Singular Value Decomposition (SVD):**
  - M is a \( n \times m \) matrix
  - M has singular values \( \sigma_1 \cdots \sigma_n \)
    - \( \Rightarrow \) M can be decomposed in:
      \[
      M = U \Sigma V^T
      \]
      where \( U \in \mathbb{R}^{n \times n} \)
      \( V \in \mathbb{R}^{m \times m} \)
      such that \( \Sigma \in \mathbb{R}^{n \times m} \) has the form
      \[
      \Sigma = \begin{bmatrix}
      \sigma_1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\
      0 & \sigma_2 & 0 & \cdots & 0 & \cdots & 0 \\
      \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
      0 & 0 & \cdots & 0 & \sigma_n & \cdots & 0 \\
      \end{bmatrix}
      \]
  - Pseudo-inverse of M:
    \[
    M^* = V \Sigma^* U^T
    \]
    where \( \Sigma^* \) is the pseudo-inverse of \( \Sigma \) and has the form
    \[
    \Sigma^* = \begin{bmatrix}
    \frac{1}{\sigma_1} & 0 & \cdots & 0 \\
    0 & \frac{1}{\sigma_2} & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & \frac{1}{\sigma_n} \\
    0 & 0 & \cdots & 0 \\
    \end{bmatrix}
    \]
    - if \( \sigma_i = 0 \) : can’t calculate the pseudo-inverse \( \Rightarrow \) Singularity problem.

- Besides \( \Sigma \) has the form
  \[
  \Sigma = \begin{bmatrix}
  \sigma_1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\
  0 & \sigma_2 & 0 & \cdots & 0 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & 0 & \sigma_n & \cdots & 0 \\
  \end{bmatrix}
  \]

- \( \Rightarrow \) M can be decomposed in:
  \[
  M = U \Sigma V^T
  \]
  where \( U \in \mathbb{R}^{n \times n} \)
  \( V \in \mathbb{R}^{m \times m} \)
  such that \( \Sigma \in \mathbb{R}^{n \times m} \) has the form
  \[
  \Sigma = \begin{bmatrix}
  \sigma_1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\
  0 & \sigma_2 & 0 & \cdots & 0 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & 0 & \sigma_n & \cdots & 0 \\
  \end{bmatrix}
  \]
SINGULARITY PROBLEM

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

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

$\Rightarrow$ singular configuration

- $\lambda = 0.2$
- $\theta_{i,0} = 0$
Different initial joint values

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

- $\lambda = 0.1$
- $\theta_{i,0} = 0$
INFLUENCE OF JOINT LIMITS

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

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