CS-184: Computer Graphics

Lecture #18: Forward and Inverse Kinematics

Prof. James O'Brien
University of California, Berkeley

Today

• Forward kinematics
• Inverse kinematics
  • Pin joints
  • Ball joints
  • Prismatic joints
Forward Kinematics

• Articulated skeleton
  • Topology (what's connected to what)
  • Geometric relations from joints
  • Independent of display geometry
  • Tree structure
  • Loop joints break “tree-ness”

• Root body
  • Position set by “global” transformation
  • Root joint
    • Position
    • Rotation
  • Other bodies relative to root
    • Inboard toward the root
    • Outboard away from root
Forward Kinematics

- A joint
  - Joint's inboard body
  - Joint's outboard body

- A body
  - Body's inboard joint
  - Body's outboard joint
  - May have several outboard joints
Forward Kinematics

- A body
  - Body's inboard joint
  - Body's outboard joint
  - May have several outboard joints
  - Body's parent
  - Body's child
  - May have several children

- Interior joints
  - Typically not 6 DOF joints
  - Pin - rotate about one axis
  - Ball - arbitrary rotation
  - Prism - translation along one axis
Forward Kinematics

- Pin Joints
  - Translate inboard joint to local origin
  - Apply rotation about axis
  - Translate origin to location of joint on outboard body

Forward Kinematics

- Ball Joints
  - Translate inboard joint to local origin
  - Apply rotation about arbitrary axis
  - Translate origin to location of joint on outboard body
Forward Kinematics

• Prismatic joints
  • Translate inboard joint to local origin
  • Translate along axis
  • Translate origin to location of joint on outboard body

Forward Kinematics

• Composite transformations up the hierarchy
Forward Kinematics

- Composite transformations up the hierarchy
Forward Kinematics

- Composite transformations up the hierarchy
Inverse Kinematics

- Given
  - Root transformation
  - Initial configuration
  - Desired end point location
- Find
  - Interior parameter settings
Inverse Kinematics

- A simple two-segment arm in 2D

\[ p_x = l_1 \sin(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \]

\[ p_z = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \]

Inverse Kinematics

- Direct IK: solve for the parameters

\[ \theta_2 = \cos^{-1} \left( \frac{p_x^2 + p_z^2 - l_1^2 - l_2^2}{2l_1l_2} \right) \]

\[ \theta_1 = \frac{-p_2l_2 \sin(\theta_2) + p_x(l_1 + l_2 \cos(\theta_2))}{p_2l_2 \sin(\theta_2) + p_z(l_1 + l_2 \cos(\theta_2))} \]
Inverse Kinematics

- Why is the problem hard?
  - Multiple solutions separated in configuration space

Inverse Kinematics

- Why is the problem hard?
  - Multiple solutions connected in configuration space
Inverse Kinematics

• Why is the problem hard?
  • Solutions may not always exist

Numerical Solution

• Start in some initial configuration
• Define an error metric (e.g. goal pos - current pos)
• Compute Jacobian of error w.r.t. inputs
• Apply Newton’s method (or other procedure)
• Iterate...
Inverse Kinematics

• Recall simple two segment arm:

\[ p_z = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \]
\[ p_x = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \]

Inverse Kinematics

• We can write of the derivatives

\[ \frac{\partial p_z}{\partial \theta_1} = -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) \]
\[ \frac{\partial p_z}{\partial \theta_2} = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \]
\[ \frac{\partial p_z}{\partial \theta_1} = -l_2 \sin(\theta_1 + \theta_2) \]
\[ \frac{\partial p_z}{\partial \theta_2} = l_1 \cos(\theta_1 + \theta_2) \]
Inverse Kinematics

Direction in Config. Space

\[ \begin{align*}
\theta_1 &= c_1 \theta_s \\
\theta_2 &= c_2 \theta_s \\
\end{align*} \]

\[ \frac{\partial p_2}{\partial \theta_s} = c_1 \frac{\partial p_2}{\partial \theta_1} + c_2 \frac{\partial p_2}{\partial \theta_2} \]

---

The Jacobian (of \( p \) w.r.t. \( \theta \))

\[ J_{ij} = \frac{\partial p_i}{\partial \theta_j} \]

Example for two segment arm

\[ J = \begin{bmatrix}
\frac{\partial p_1}{\partial \theta_1} & \frac{\partial p_2}{\partial \theta_1} \\
\frac{\partial p_1}{\partial \theta_2} & \frac{\partial p_2}{\partial \theta_2}
\end{bmatrix} \]
Inverse Kinematics

The Jacobian (of \( p \) w.r.t. \( \theta \))

\[
J = \begin{bmatrix}
\frac{\partial p_x}{\partial \theta_1}, & \frac{\partial p_x}{\partial \theta_2} \\
\frac{\partial p_y}{\partial \theta_1}, & \frac{\partial p_y}{\partial \theta_2}
\end{bmatrix}
\]

\[
\frac{\partial p}{\partial \theta_x} = J \cdot \begin{bmatrix}
\frac{\partial \theta_1}{\partial \theta_x} \\
\frac{\partial \theta_2}{\partial \theta_x}
\end{bmatrix} = J \cdot \begin{bmatrix}
c_1 \\
c_2
\end{bmatrix}
\]

Inverse Kinematics

Solving for \( c_1 \) and \( c_2 \)

\[
c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \quad \delta p = \begin{bmatrix} \frac{\partial p_x}{\partial p} \\ \frac{\partial p_y}{\partial p} \end{bmatrix}
\]

\[
\delta p = J \cdot c \\
c = J^{-1} \cdot \delta p
\]
Inverse Kinematics

Solving for \( c_1 \) and \( c_2 \)

\[
p = e = dp
\]

\[
dp = J \cdot c
\]

\[
c = J^{-1} \cdot dp
\]

- Problems
  - Jacobian may not always be invertible
    - Use pseudo inverse (SVD)
    - Robust iterative method
  - Jacobian is not constant
  - Nonlinear optimization, but problem is (mostly) well behaved

\[
J = \begin{bmatrix}
\frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} \\
\frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2}
\end{bmatrix} = J(\theta)
\]
Inverse Kinematics

Jacobian is not always invertible

• Use pseudo inverse (SVD)

Computing a linear approximation

• End effector only locally moves linearly
• So iterate (choosing proper step size) and update Jacobian after each step
• Choosing step size requires line search at each step
  • Choose some step size (say 5 degrees) and compute how to update joint parameters
  • Calculate distance of end effector from goal
  • If distance decreased take step
  • If distance did not decrease set parameters to be half the current change and try again

Problems...

Jacobian may (will) not be invertible

Option #1: Use pseudo inverse (SVD)

Option #2: Use iterative method

Jacobian is not constant

Non-linear optimization...

but problem is well behaved (mostly)

Inverse Kinematics

• More complex systems
  • More complex joints (prism and ball)
  • More links
  • Other criteria (COM or height)
  • Hard constraints (joint limits)
  • Multiple criteria and multiple chains
Inverse Kinematics

• Some issues
  • How to pick from multiple solutions?
  • Robustness when no solutions
  • Contradictory solutions
  • Smooth interpolation
    • Interpolation aware of constraints

• Numerical evaluation of Jacobian

Prism Joints

\[ p_z = l_1 + d \]
\[ p_x = 0 \]

\[ p_z = l_1 \]
\[ p_x = d \]
Inverse Kinematics

Ball Joints

\[ p = \hat{p} (\hat{p} \cdot x) \]
\[ + \sin(||r||)(\hat{p} \times x) \]
\[ - \cos(||r||)(\hat{p} \times (\hat{p} \times x)) \]

Inverse Kinematics

Ball Joints (moving axis)

\[ \frac{dp}{dt} = [d\mathbf{r}]e^{[r]_x} = [d\mathbf{r}]; \mathbf{r} = [p]; d\mathbf{r} \]

That is the Jacobian for this joint.

\[ [r] = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix} \]

\[ [r] \cdot \mathbf{r} = \mathbf{r} \times \mathbf{r} \]
Inverse Kinematics

Ball Joints (fixed axis)

$$\frac{d}{dt} = \dot{\theta} \cdot \hat{r} = -\dot{x} \cdot \hat{r} \dot{\theta}$$

That is the Jacobian for this joint

• Many links / joints
  • Need a generic method for building Jacobian
Inverse Kinematics

- Can't just concatenate individual matrices

\[ J = [J_3 \ J_{2b} \ J_{2a} \ J_{1b}] \]

\[ d = \begin{bmatrix} d_3 \\ d_{2b} \\ d_{2a} \\ d_{1b} \end{bmatrix} \]

\[ \text{if } p \neq J \cdot d \]

Transformation from body to world

\[ X_{0-1} = \prod_{j=1}^{1} X_{(j-1)-j} = X_{0-1} \cdot X_{1-2} \cdots \]

Rotation from body to world

\[ R_{0-1} = \prod_{j=1}^{1} R_{(j-1)-j} = R_{0-1} \cdot R_{1-2} \cdots \]
Inverse Kinematics

Need to transform Jacobians to common coordinate system (WORLD)

\[ J_{i,\text{WORLD}} = R_{(i-1)} \cdot J_i \]

Inverse Kinematics

\[ J = \begin{bmatrix} R_{0-1} \cdot J_3(\theta_3, p_3) \\ R_{0-2} \cdot J_2(\theta_2, X_{20}) \cdot p_3 \\ R_{0-1} \cdot J_2(\theta_2, X_{21}) \cdot p_3 \\ J_1(\theta_1, X_{1-3}) \cdot p_3 \end{bmatrix} \]

\[ d = \begin{bmatrix} d_3 \\ d_{2b} \\ d_{2a} \\ d_{1b} \end{bmatrix} \]

Note: Each row in the above should be transposed...

\[ \delta p = J \cdot \delta d \]
## Suggested Reading

- Advanced Animation and Rendering Techniques by Watt and Watt
- Chapters 15 and 16