

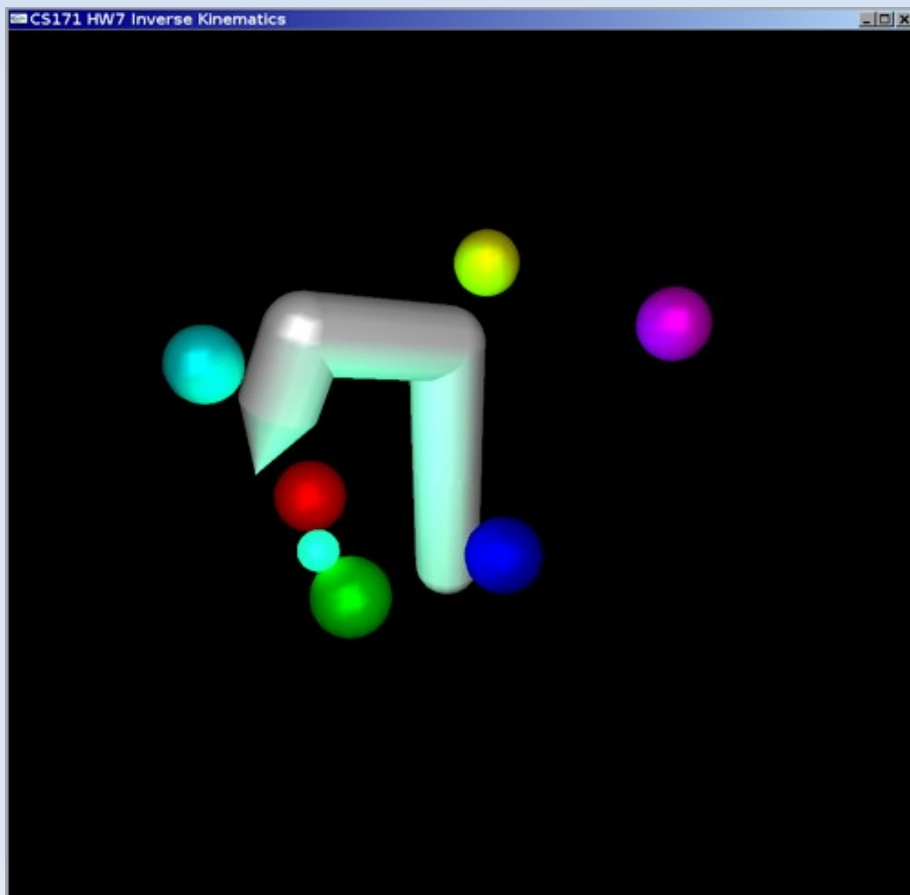
Inverse Kinematics

CS 171 Recitation, Lab 7

Isaac Chao

Formulation

- Map from state vector to a position vector
 - Robot Arm: angles \rightarrow tip in \mathbb{R}^3



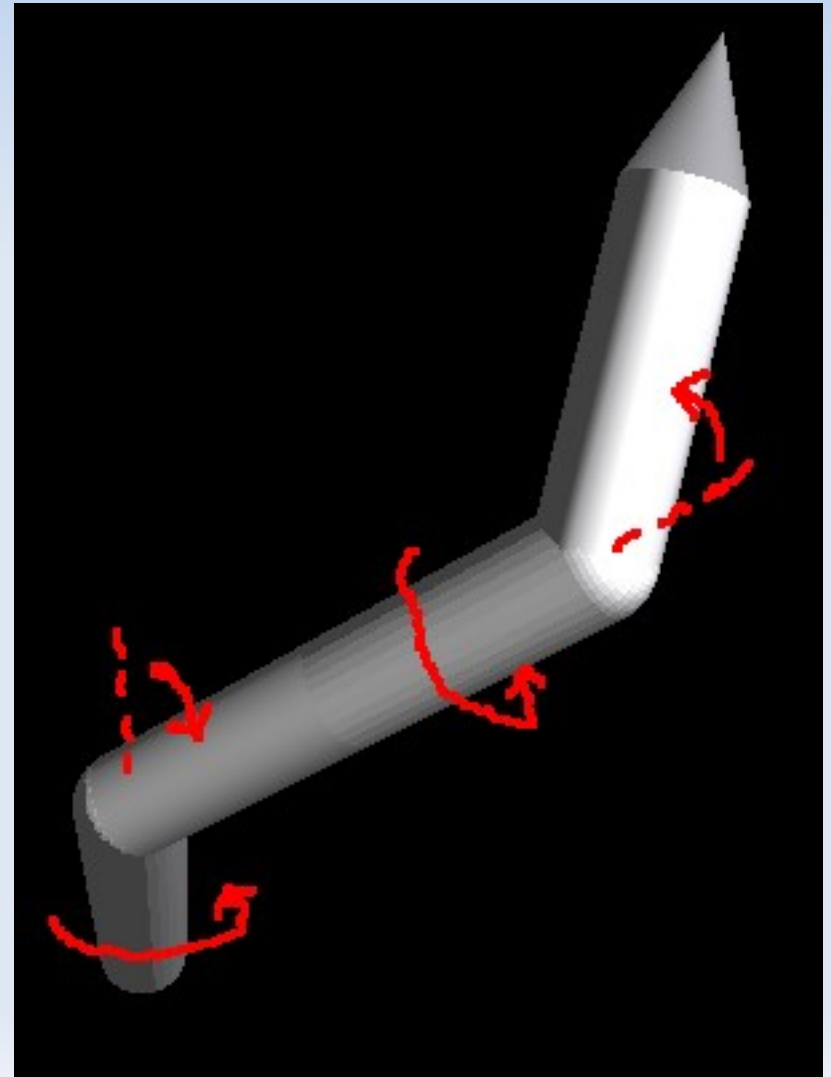
$$X = f(\Theta)$$

Know where to place tip

Corresponding state vector
not obvious

Formulation

- Two type of rotation mode: “yaw/pitch” and “roll”
- Alternatively, for each arm consider both types of rotations
- For this assignment, be sure to have at least two nonconsecutive “roll” arms.



Inversion

- Solve angle perturbation given tip movement

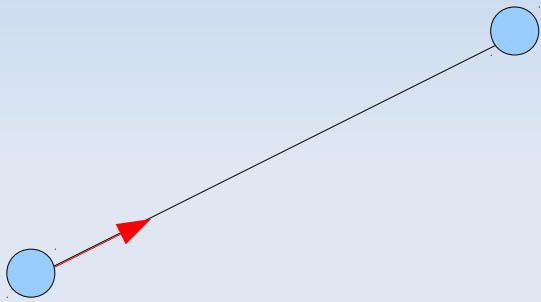
$$J(\Theta) = \frac{dX}{d\Theta} = \begin{pmatrix} \frac{\partial x}{\partial \Theta_1} & \frac{\partial x}{\partial \Theta_2} & \dots & \frac{\partial x}{\partial \Theta_k} \\ \frac{\partial y}{\partial \Theta_1} & \frac{\partial y}{\partial \Theta_2} & \dots & \frac{\partial y}{\partial \Theta_k} \\ \frac{\partial z}{\partial \Theta_1} & \frac{\partial z}{\partial \Theta_2} & \dots & \frac{\partial z}{\partial \Theta_k} \end{pmatrix}$$

- Rearranging gives

$$d\Theta = J(\Theta)^{-1} dX$$

Stepping

- Tip movement towards target



$$\Delta X = \alpha \text{ unit} (X_{\text{target}} - X)$$

- Amounts to a step in angle space

$$\Delta \Theta = J^{-1}(\Theta) \Delta X$$

And we can update $\Theta' = \Theta + \Delta \Theta$

Now the details...

- Computing tip position using state vector
 - May be difficult depending on setup
 - OpenGL can actually help!
- Computing the Jacobian
 - Finite difference – easy once we know how to compute $X = f(\text{Theta})$
- Inverting the Jacobian
 - Not necessarily square – use pseudoinverse
 - JAMA, NumPy, Boost, ...

Tip Position 1

- As we are rendering arm
 - Translating, rotating,
 - Reaches the tip eventually
- Starting from a reference position $[0,0,0,1]$
- We reach the tip via a overall transform
- $P = M * [0,0,0,1]$
 - Basically want to implement function:
vector tipPos(angles, armlength)

Tip Position 2

$$P_{tip} = M \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}^T$$

- Fetch M via glGetDoublev in OpenGL!
 - glPushMatrix()
glLoadIdentity()
glutSolidCylinder(...)
glTranslatef(...)
glutSolidSphere(...)
glRotatef(...)
glutSolidCylinder(...)
...
glTranslatef(...)
glGetDoublev(GL_MODELVIEW_MATRIX, &M)
glPopMatrix()
- Now multiply to get tip position (homogenized)

Jacobian – FD

$$J(\Theta) = \frac{dX}{d\Theta} = \begin{pmatrix} \frac{\partial x}{\partial \Theta_1} & \frac{\partial x}{\partial \Theta_2} & \dots & \frac{\partial x}{\partial \Theta_k} \\ \frac{\partial y}{\partial \Theta_1} & \frac{\partial y}{\partial \Theta_2} & \dots & \frac{\partial y}{\partial \Theta_k} \\ \frac{\partial z}{\partial \Theta_1} & \frac{\partial z}{\partial \Theta_2} & \dots & \frac{\partial z}{\partial \Theta_k} \end{pmatrix}$$

- Finite difference approximation

$$\frac{\partial x}{\partial \Theta_1} \approx \frac{f(\Theta_1 + \epsilon, \Theta_2, \dots, \Theta_k) - f(\Theta_1, \Theta_2, \dots, \Theta_k)}{\epsilon}$$

- f is tipPos function we defined earlier
 - $(\text{tipPos}([a_1 + \epsilon, a_2, \dots, a_k], l_s) - \text{tipPos}([a_1, a_2, \dots, a_k], l_s)) / \epsilon$
 - Repeat for $3 * k$ derivatives
- Now use your favorite library to compute pseudoinverse