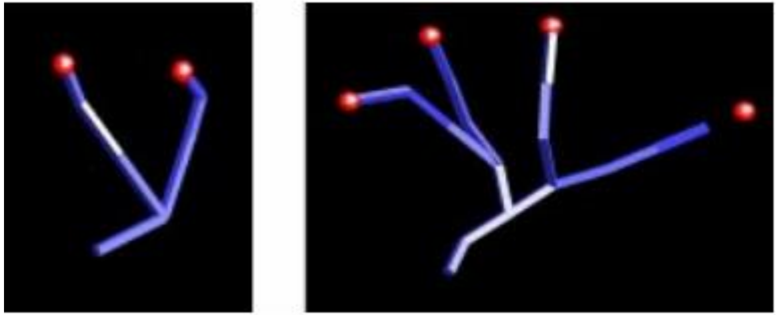
COMS W4733: Computational Aspects of Robotics

Lecture 7: Numerical Inverse Kinematics



S. R. Buss. "Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods."

Instructor: Tony Dear

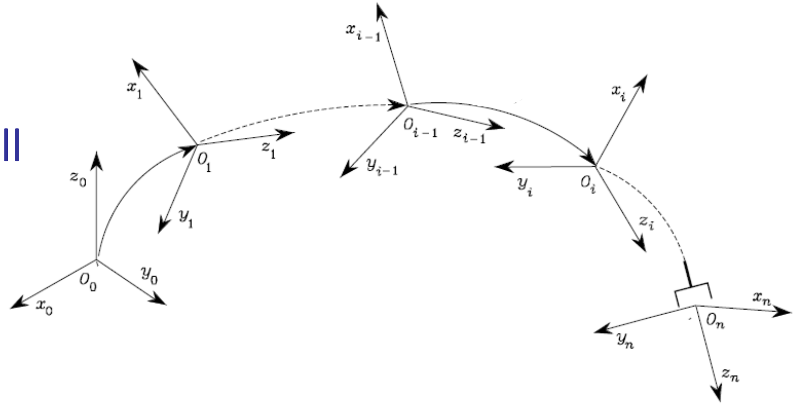
Review: Inverse Differential Kinematics

- *Linear* mapping between joint and end effector velocities
- $r \times n$ configuration-dependent Jacobian: $\boldsymbol{v}_e = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$
  - r rows for each end effector velocity
  - n columns for each joint velocity

- Inverse problem: Find joint velocities to achieve desired end effector velocities
  - $r = n$: Invert the Jacobian (assuming non-singular)
  - $r < n$: Underconstrained, **right pseudoinverse** can optimize different criteria
  - $r > n$: Overconstrained, **left pseudoinverse** can minimize resulting error
  - Singular (or near singular): **damped least-squares** to move Jacobian away from singularities

Back to Inverse Kinematics

- Moving a robot to a desired pose still a hard problem
  - Inverting nonlinear FK equations was too difficult
- Can we use the simpler IDK problem with velocities?
- Instead of finding joint angles to achieve a desired pose, we can find joint velocities that will move toward $\boldsymbol{T}_e^0$ from current configuration!



Operational Space Error

- What is the "desired" end effector velocity?
- Want end effector to move toward desired pose

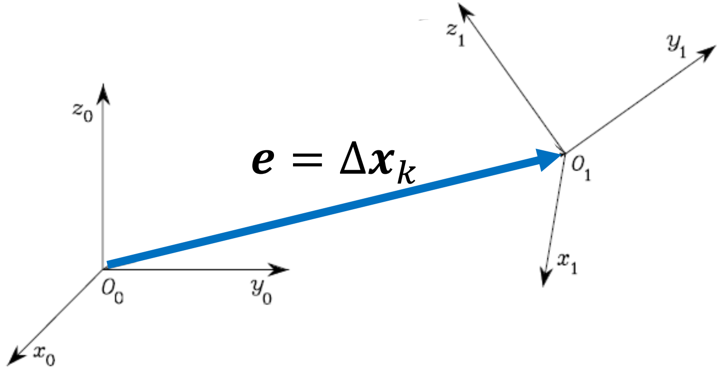
- *Operational space error* $\boldsymbol{e}$: "difference" between desired and current pose

- Solving IDK gives us a small change $\Delta\boldsymbol{q}$ to effect $\Delta\boldsymbol{x}$

- We can iteratively update joint configuration $\boldsymbol{q}_k$

$$\boldsymbol{e} = \Delta\boldsymbol{x}_k = \boldsymbol{J}(\boldsymbol{q}_k)\Delta\boldsymbol{q}_k \qquad \Delta\boldsymbol{q}_k = \boldsymbol{J}^+(\boldsymbol{q}_k)\Delta\boldsymbol{x}_k$$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \Delta\boldsymbol{q}_k$$



Computing the Error

- $\boldsymbol{e} = \Delta \boldsymbol{x}_k$ is a 6×1 workspace velocity vector
- What is the "difference" of two homogeneous transforms?

$$\boldsymbol{T}_k(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{R}_k(\boldsymbol{q}) & \boldsymbol{p}_k(\boldsymbol{q}) \\ \boldsymbol{0}^T & 1 \end{bmatrix} \qquad \boldsymbol{T}_d(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{R}_d(\boldsymbol{q}) & \boldsymbol{p}_d(\boldsymbol{q}) \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

- Linear velocity error: Difference between desired and current positions

$$\boldsymbol{e}_P = \boldsymbol{p}_d - \boldsymbol{p}_k$$

- For angular velocity, we need the "difference" between two rotation matrices; one metric is to sum the cross products of each matrix's respective columns

$$\boldsymbol{e}_O = \boldsymbol{x}_k \times \boldsymbol{x}_d + \boldsymbol{y}_k \times \boldsymbol{y}_d + \boldsymbol{z}_k \times \boldsymbol{z}_d \qquad \boldsymbol{R}_i = [\boldsymbol{x}_i \quad \boldsymbol{y}_i \quad \boldsymbol{z}_i]$$

Iterative Inverse Kinematics

- **Given**: Initial config $\boldsymbol{q}_i$, desired pose $\boldsymbol{T}_d^0(t)$, weighting matrices $\boldsymbol{K}_P$, $\boldsymbol{K}_O$
- error $\boldsymbol{e} = \infty$
- current config $\boldsymbol{q}_k = \boldsymbol{q}_i$

- while not converged($\boldsymbol{e}$)
  - $\boldsymbol{T}_k^0 = \mathrm{FK}(\boldsymbol{q}_k), \boldsymbol{J}_k = \mathrm{Jacobian}(\boldsymbol{q}_k)$
  - $\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_P \\ \boldsymbol{e}_O \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{p}}_d \\ \boldsymbol{\omega}_d \end{bmatrix} + \begin{bmatrix} \boldsymbol{p}_d - \boldsymbol{p}_k \\ \boldsymbol{x}_k \times \boldsymbol{x}_d + \boldsymbol{y}_k \times \boldsymbol{y}_d + \boldsymbol{z}_k \times \boldsymbol{z}_d \end{bmatrix}$
  - $\boldsymbol{J}_{inv} = \text{invertJacobian}(\boldsymbol{J}_k)$
  - $\Delta \boldsymbol{q}_k = \boldsymbol{J}_{inv} \begin{bmatrix} \boldsymbol{K}_P \boldsymbol{e}_P \\ \boldsymbol{K}_O \boldsymbol{e}_O \end{bmatrix}$
  - $\boldsymbol{q}_k = \boldsymbol{q}_k + \Delta \boldsymbol{q}_k$

If desired pose is a moving target, we should also include its velocity when solving IDK

Weighting matrices can help with faster convergence, at the expense of jerkier movements. Also helps to reweight different magnitudes between $\boldsymbol{e}_P$ and $\boldsymbol{e}_O$.

Choice of Jacobian Inverse

- All of the (pseudo-)inverses we looked at last time can be used!

- If J is square, simply invert it; otherwise…
  - Underconstrained: Right pseudo-inverse $\boldsymbol{J}_r^+ = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$
  - Overconstrained: Left pseudo-inverse $\boldsymbol{J}_l^+ = (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T$

- Singularities may be significant in at least some places along an entire trajectory, making pseudo-inverse solutions numerically unstable near those configurations

- Damped least-squares: $\boldsymbol{J}^* = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T + k^2\boldsymbol{I})^{-1}$

Jacobian Transpose

- What about $\boldsymbol{J}^T$? Can we use that instead of the (pseudo-)inverse?
- It has the correct dimensions $(n \times r)$, same as $\boldsymbol{J}^{-1}, \boldsymbol{J}^{+}, \boldsymbol{J}^{*}$...

- We can show that using the Jacobian transpose does indeed shrink the error:

$$\Delta \boldsymbol{q} = \alpha \boldsymbol{J}^T \boldsymbol{e}$$

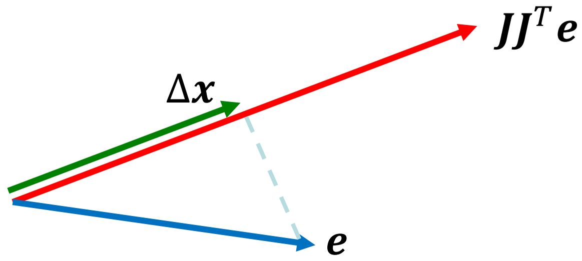
$$\Delta \boldsymbol{x} = \boldsymbol{J} \Delta \boldsymbol{q} = \alpha \boldsymbol{J} \boldsymbol{J}^T \boldsymbol{e}$$

Consider $(\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e}) \cdot \boldsymbol{e} = (\boldsymbol{J}^T\boldsymbol{e}) \cdot (\boldsymbol{J}^T\boldsymbol{e}) = \|\boldsymbol{J}^T\boldsymbol{e}\|^2 \geq 0$

The vectors $\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e}$ and $\boldsymbol{e}$ are at least somewhat aligned, so the change $\Delta\boldsymbol{x} = \alpha\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e}$ shrinks the overall error

- How about α? Maybe make $\Delta\boldsymbol{x}$ as close as possible to $\boldsymbol{e}$

$$\alpha = \frac{(\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e}) \cdot \boldsymbol{e}}{(\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e}) \cdot (\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{e})}$$



Algorithm Comparison

- **Jacobian transpose**: Fast, computationally easier (no inverses!)
- Not affected by singularities
- May produce bad trajectories or large joint motions

- **Jacobian pseudo-inverse**: Good away from singularities
- Flexibility with optimization of different criteria
- Very unstable joint motions near singularities

- **Damped least-squares inverse**: Slower but generally better overall
- Performance depends on damping parameter