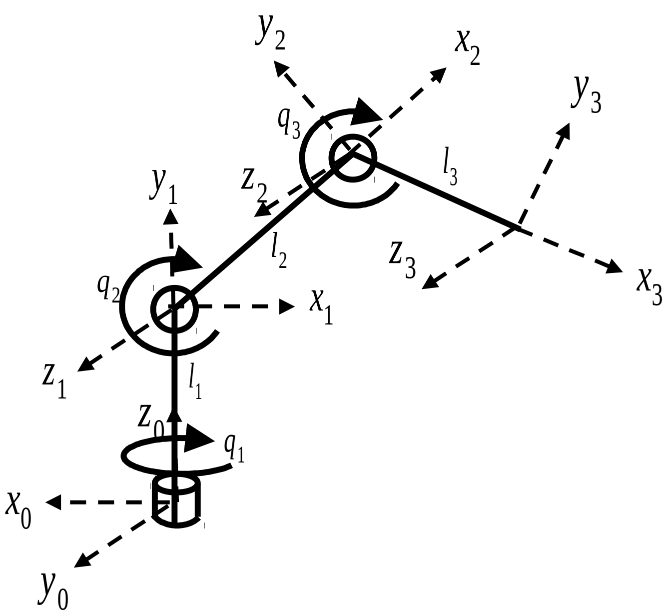
Cartesian Control (Translation)

Robert Platt
Northeastern University

Two ways of using the manipulator Jacobian

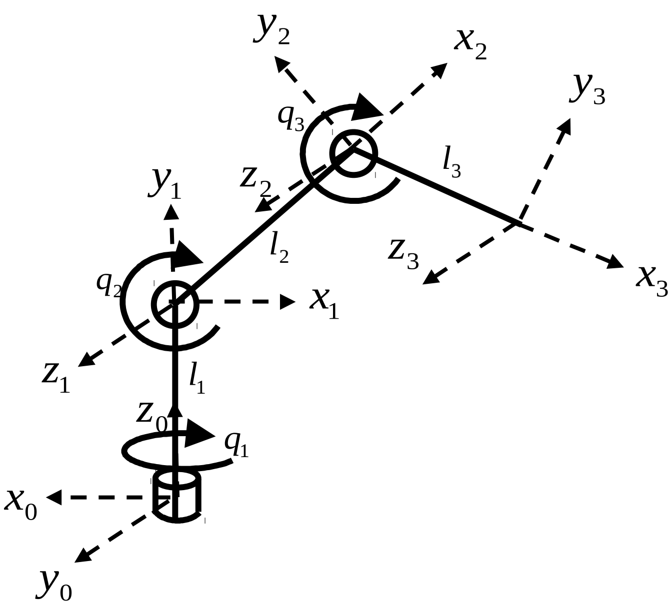


$$J=\begin{pmatrix} -s_1(l_2c_2+l_3c_{23}) & -c_1(l_2c_2+l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2+l_3c_{23}) & -s_1(l_2c_2+l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2+l_3c_{23} & l_3c_{23} \end{pmatrix}$$

1. use Jacobian to find numerical solution to IK
   – solution is just a single configuration

2. use Jacobian to find arm trajectories that achieve a desired end effector path
   – solution is a trajectory through joint space

Two ways of using the manipulator Jacobian



$$J = \begin{pmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{pmatrix}$$

1. use Jacobian to find numerical solution to IK
   – solution is just a single configuration

2. use Jacobian to find arm trajectories that achieve a desired end effector path
   – solution is a trajectory through joint space

Numerical IK Solution (method 1)

```
Input: x*
Output: q*
1. repeat until dx is small:
2.          init q to random joint configuration
3.          repeat K times:
4.                   x = FK(q)
5.                   dx = x*-x
6.                   dq = stepsize * J^{-1} dx
7.                   q = q + dq
8. return q* = q
```

Numerical IK Solution (method 1)

Input: x*
Output: q*
1. repeat until dx is small:
2. init q to random joint configuration
3. repeat K times:
4. x = FK(q)
5. dx = x*-x
6. dq = stepsize * J^{-1} dx
7. q = q + dq
8. return q* = q

Idea:

$$\dot{q} = J^{-1}\dot{x}$$

Numerical IK Solution (method 2)

Input: x*
Output: q*
1. repeat until dx is small:
2. init q to random joint configuration
3. repeat K times:
4. x = FK(q)
5. dx = x*-x
6. dq = stepsize * J^T dx
7. q = q + dq
8. return q* = q

<u>This also works</u>

$$\dot{q} = J^T \dot{x}$$

Numerical IK Solution (method 2)

Where does this $\dot{q} = J^T \dot{x}$ come from?

L2 loss

Position error

$$L = \frac{1}{2} e^T e \quad \text{where} \quad e = x^* - x$$

$$\frac{\partial L}{\partial q} = -e^T \frac{\partial x}{\partial q}$$

Do gradient descent on *L*

$$\dot{q} = -\alpha \frac{\partial L}{\partial q}^T$$

$$\dot{q} = \alpha \frac{\partial x}{\partial q}^T e = \alpha J^T e$$

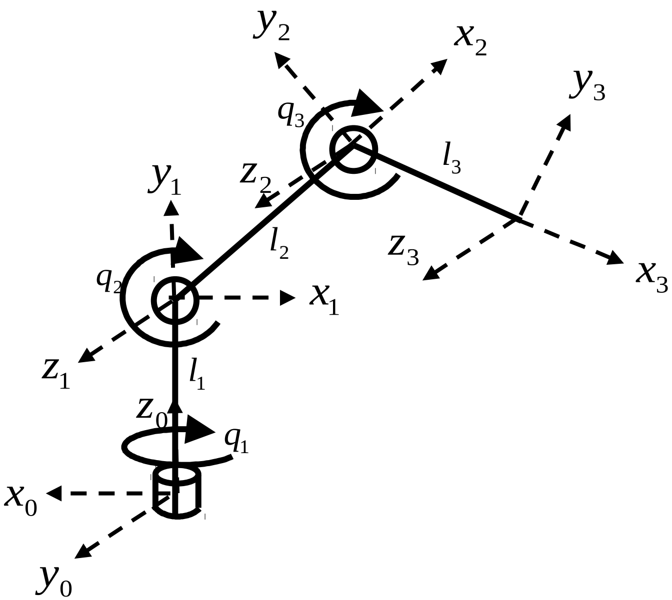
Numerical IK Solution (method 2)

```
Input: x*
Output: q*
1. repeat until dx is small:
2.          init q to random joint configuration
3.          repeat K times:
4.                      x = FK(q)
5.                      dx = x*-x
6.                      dq = stepsize * J^T dx
7.                      q = q + dq
8. return q* = q
```

So, this is just gradient descent on *L*

– people sometimes use Newton's method to get faster convergence

Two ways of using the manipulator Jacobian

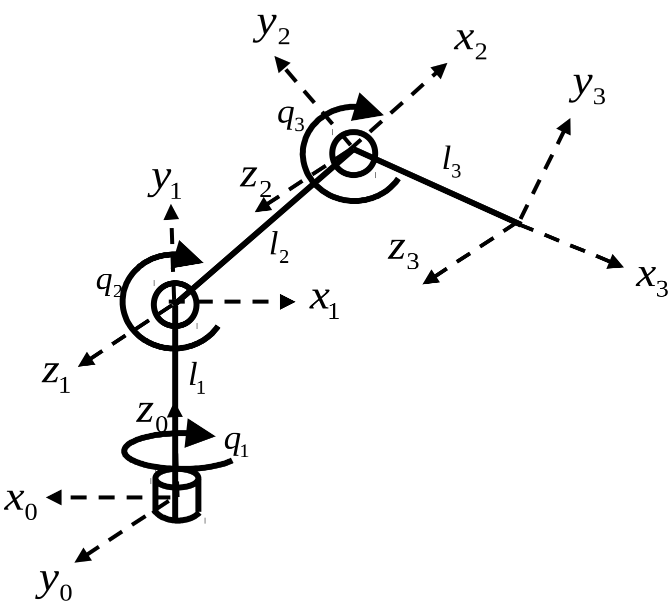


$$J = \begin{pmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{pmatrix}$$

1. use Jacobian to find numerical solution to IK
   – solution is just a single configuration

2. use Jacobian to find arm trajectories that achieve a desired end effector path
   – solution is a trajectory through joint space

Two ways of using the manipulator Jacobian



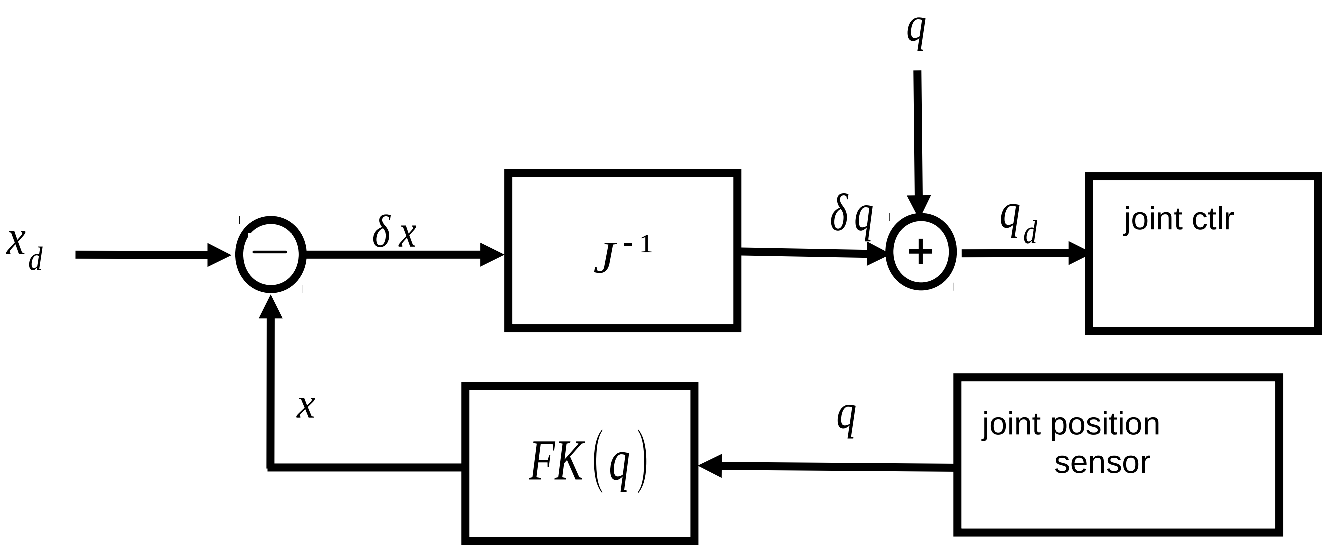
$$J = \begin{pmatrix} -s_1(l_2c_2+l_3c_{23}) & -c_1(l_2c_2+l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2+l_3c_{23}) & -s_1(l_2c_2+l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2+l_3c_{23} & l_3c_{23} \end{pmatrix}$$

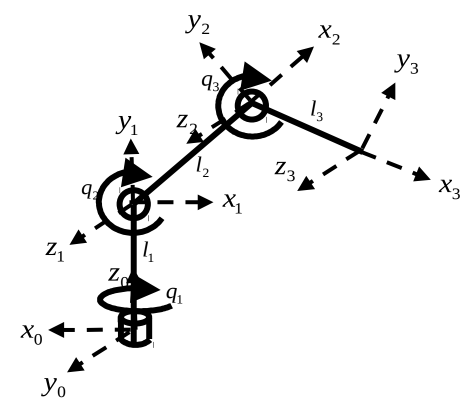
Called *Cartesian Control*

1. use Jacobian to find numerical solution to IK
   – solution is just a single configuration

2. use Jacobian to find arm trajectories that achieve a desired end effector path
   – solution is a trajectory through joint space

Cartesian control

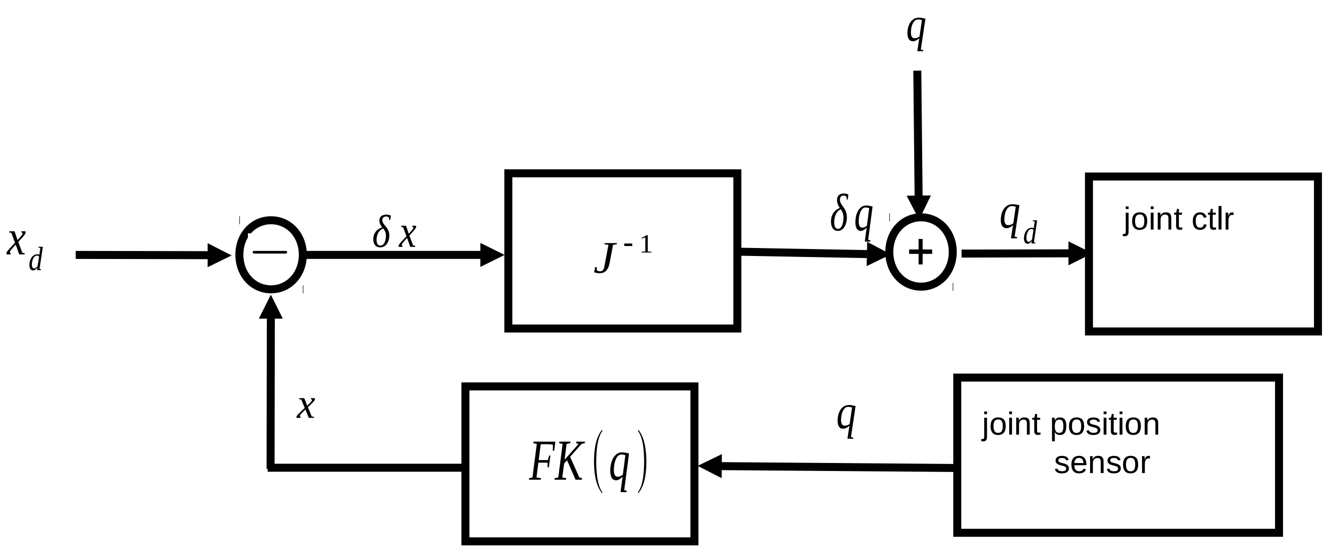


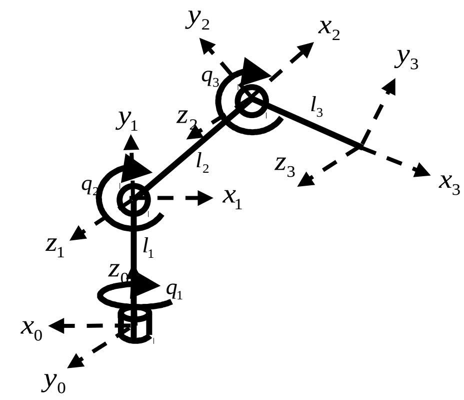


Cartesian control is almost identical to the numerical IK solution

<u>Difference</u>: Cartesian control actually moves the arm during the optimization process.

Think-pair-share



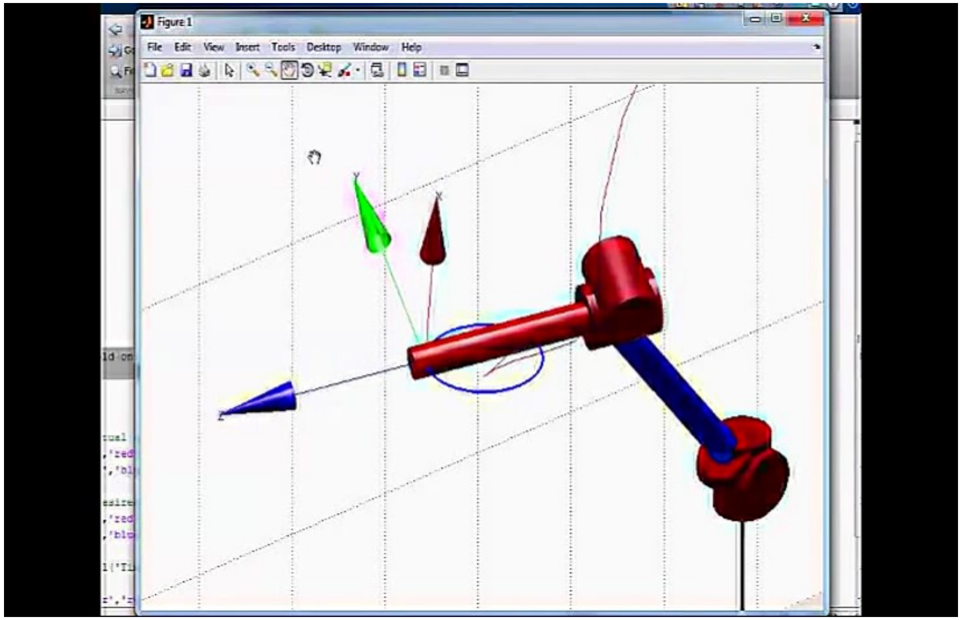


1. what does the velocity profile look like for this controller?

2. how would you modify it to move the arm at constant velocity?

3. How would you modify it to follow a trapezoidal velocity profile?

Question



This is not just IK – can use Cartesian control to get entire trajectory

– How?

Video credit: Muhammad Tufail

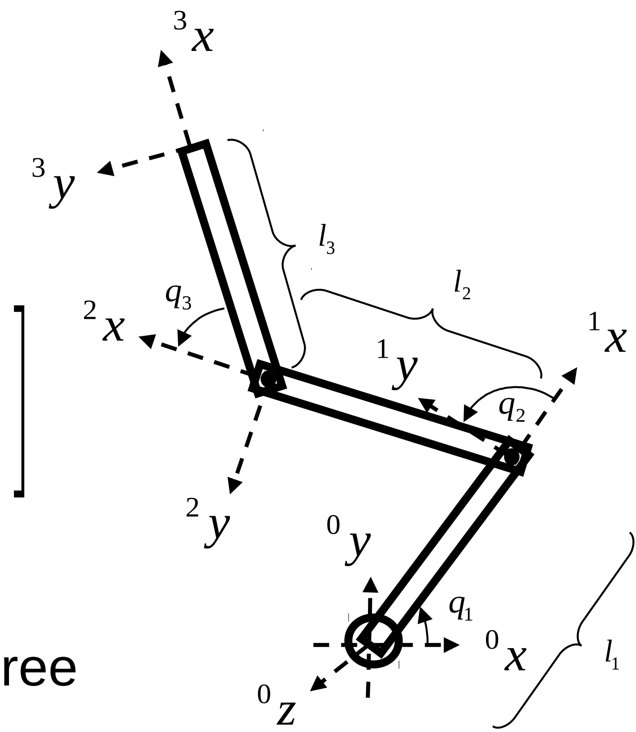
Non-square Jacobian matrix

Input: x*
Output: q*
1. repeat until dx is small:
2. init q to random joint configuration
3. repeat K times:
4. x = FK(q)
5. dx = x*-x
6. dq = stepsize * J^{-1} dx
7. q = q + dq
8. return q* = q

Can only take inverse of Jacobian it is square

Non-square Jacobian matrix

Example of a non-square Jacobian matrix:



$$J(q) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_1 s_1 - l_2 s_{12} & -l_1 s_1 \\ l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_1 c_1 + l_2 c_{12} & l_1 c_1 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = J(q) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}$$

Two equations of three variables each…

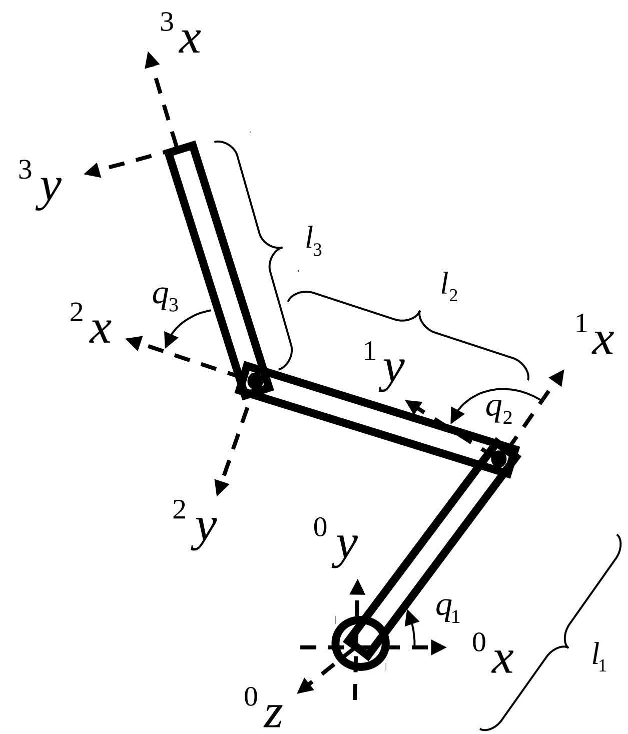
This is an *under-constrained* system of equations.

- multiple solutions
- there are multiple joint angle velocities that realize the same EFF velocity.

Non-square Jacobian matrix

If the Jacobian is not a square matrix, then you can't invert it.

- what next?



We have: $\dot{x} = J\dot{q}$

We are looking for a matrix $J^{\#}$ such that:

$$\dot{q} = J^{\#}\dot{x} \longrightarrow \dot{x} = J\dot{q}$$

Generalized inverse

Two cases:

- Underconstrained manipulator (redundant)
- Overconstrained

Generalized inverse:

- for the underconstrained manipulator: given $\dot{x}$, find a vector $\dot{q}$ that minimizes $||\dot{q}||^2$ s.t. $\dot{x} = J\dot{q}$

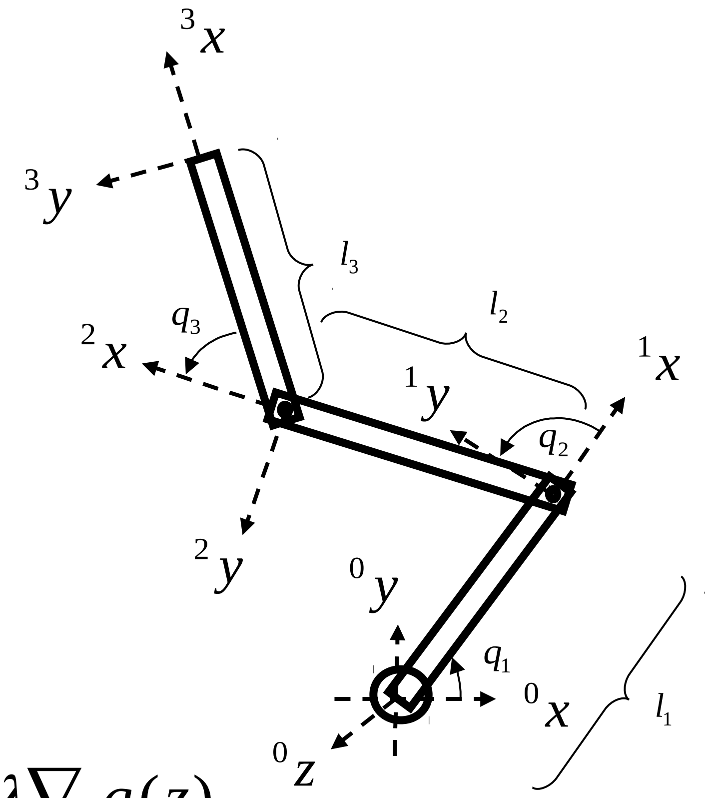
- for the overconstrained manipulator: given $\dot{x}$, find a vector $\dot{q}$
  s.t. $||\dot{x} - J\dot{q}||^2$ Is minimized

Jacobian Pseudoinverse: Redundant manipulator

Psuedoinverse definition: (underconstrained)

Given a desired twist, $\dot{x}_d$, find a vector of joint velocities, $\dot{q}$, that satisfies $\dot{x}_d = J\dot{q}$ while minimizing $f(\dot{q}) = \dot{q}^T \dot{q}$

Minimize joint velocities



Minimize $f(z)$ subject to $g(z) = 0$:

Use **lagrange multiplier method**: $\nabla_z f(z) = \lambda \nabla_z g(z)$

This condition must be met when $f(z)$ is at a minimum subject to $g(z) = 0$

Jacobian Pseudoinverse: Redundant manipulator

$$\nabla_z f(z) = \lambda \nabla_z g(z)$$

$$f(\dot{q}) = \tfrac{1}{2}\dot{q}^T \dot{q}$$ ← Minimize

$$g(\dot{q}) = J\dot{q} - \dot{x} = 0$$ ← Subject to

$$\nabla_{\dot{q}} f(\dot{q}) = \dot{q}^T$$

$$\nabla_{\dot{q}} g(\dot{q}) = J$$

$$\dot{q}^T = \lambda^T J$$

$$\dot{q} = J^T \lambda$$

Jacobian Pseudoinverse: Redundant manipulator

$$\dot{q} = J^T \lambda$$

$$J\dot{q} = \left(JJ^T\right)\lambda$$

$$\lambda = \left(JJ^T\right)^{-1} J\dot{q}$$

I won't say why, but if J is full rank, then JJ^T is invertible

$$\lambda = \left(JJ^T\right)^{-1} \dot{x}$$

$$\dot{q} = J^T \lambda$$

$$\dot{q} = J^T \left(JJ^T\right)^{-1} \dot{x}$$

$$J^{\#} = J^T \left(JJ^T\right)^{-1}$$

$$\dot{q} = J^{\#} \dot{x}$$

So, the pseudoinverse calculates the vector of joint velocities that satisfies $\dot{x}_d = J\dot{q}$ while minimizing the squared magnitude of joint velocity ($\dot{q}^T \dot{q}$).

- Therefore, the pseudoinverse calculates the *least-squares* solution.

Calculating the pseudoinverse

The pseudoinverse can be calculated using two different equations depending upon the number of rows and columns:

$$J^{\#} = J^T \left(J J^T \right)^{-1}$$

Underconstrained case (if there are more columns than rows ($m<n$))

$$J^{\#} = \left(J^T J \right)^{-1} J^T$$

Overconstrained case (if there are more rows than columns ($n<m$))

$$J^{\#} = J^{-1}$$

If there are an equal number of rows and columns ($n=m$)

These equations can only be used if the Jacobian is full rank; otherwise, use singular value decomposition (SVD):

Rank deficient Jacobian matrices

What if Jacobian is not full rank?

– rows/columns not linearly independent

– columns do not span Cartesian space

– Determinant of JJ^T is zero

Can use Singular Value Decomposition (SVD)

Calculating the pseudoinverse using SVD

Singular value decomposition decomposes a matrix as follows:

For an under-constrained matrix, Σ is a diagonal matrix of singular values:

$$J = \underbrace{U}_{m \times m}\underbrace{\Sigma}_{m \times n}\underbrace{V^T}_{n \times n}$$

$$J = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_n & 0 & 0 \end{bmatrix} V^T$$

Singular values

Calculating the pseudoinverse using SVD

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

$$J = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_n & 0 & 0 \end{bmatrix} V^T$$

$$J^{\#} = V \Sigma^{-1} U^T$$

$$J^{¿} = V \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sigma_3} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sigma_n} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} U^T$$

Calculating the pseudoinverse using SVD

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

$$J = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_n & 0 & 0 \end{bmatrix} V^T$$

$$J^{\#} = V \Sigma^{-1} U^T$$

$$J^{\text{¿}} = V \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sigma_3} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sigma_n} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} U^T$$

What if some of the singular values are zero?

Calculating the pseudoinverse using SVD

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

$$J = U \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_n & 0 & 0 \end{bmatrix} V^T$$

$$J^{\#} = V\Sigma^{-1}U^T$$

$$J^{\text{¿}} = V \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sigma_3} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sigma_n} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} U^T$$

What if some of the singular values are zero?

<u>Answer</u>: you could set them to small positive nonzero values.

Properties of the pseudoinverse

Moore-Penrose conditions:

1. $J^{\#} J J^{\#} = J^{\#}$
2. $J J^{\#} J = J$
3. $\left(J J^{\#}\right)^{T} = J J^{\#}$
4. $\left(J^{\#} J\right)^{T} = J^{\#} J$

Generalized inverse: satisfies condition 1

Reflexive generalized inverse: satisfies conditions 1 and 2

Pseudoinverse: satisfies all four conditions

Other useful properties of the pseudoinverse:

$$(J^{\#})^{\#} = J$$

$$(J^{\#})^{T} = (J^{T})^{\#}$$

Think-pair-share

Prove that one of the Moore-Penrose conditions holds for the pseudoinverse using the SVD:

1. $J^{\#} J J^{\#} = J^{\#}$
2. $J J^{\#} J = J$
3. $\left(J J^{\#}\right)^{T} = J J^{\#}$
4. $\left(J^{\#} J\right)^{T} = J^{\#} J$

Jacobian Transpose v Pseudoinverse

What gives?

- Which is more direct? Jacobian pseudoinverse or transpose?

$$\dot{q} = J^T \xi \qquad \text{or} \qquad \dot{q} = J^{\#} \xi$$

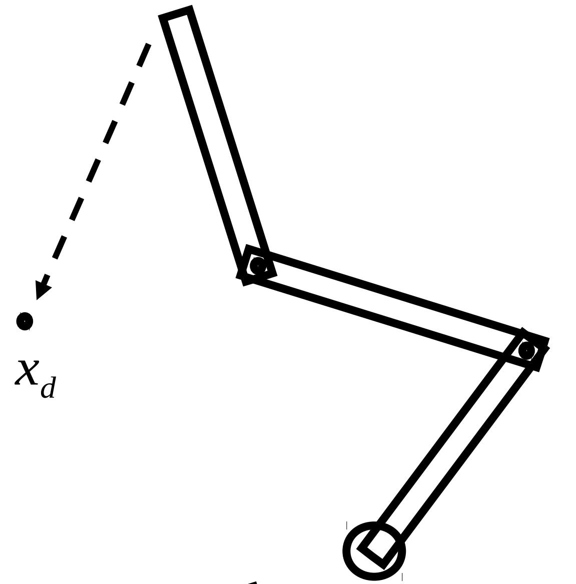
They do different things:

- Transpose: move toward a reference pose as quickly as possible
  - One dimensional goal (squared distance meteric)
- Pseudoinverse: move along a least squares reference twist trajectory
  - Six dimensional goal (or whatever the dimension of the relevant twist is)

Jacobian Transpose v Pseudoinverse

The pseudoinverse moves the end effector in a straight line path toward the goal pose using the least squared joint velocities.

- The goal is specified in terms of the reference twist
- Manipulator follows a straight line path in Cartesian space



The transpose moves the end effector toward the goal position

- In general, not a straight line path *in Cartesian space*
- Instead, the transpose follows the gradient in *joint space*