Cartesian Control

• Analytical inverse kinematics can be difficult to derive
• Inverse kinematics are not as well suited for small differential motions

• Let’s take a look at how you use the Jacobian to control Cartesian position
Cartesian control

Let’s control the position (not orientation) of the three link arm end effector:

\[
J = \begin{bmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{bmatrix}
\]

We can use the same strategy that we used before:

\[
\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = J \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}
\]
However, this only works if the Jacobian is square and full rank...

\[ \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \]

- All rows/columns are linearly independent, or
- Columns span Cartesian space, or
- Determinant is not zero
What if you want to control the two-dimensional position of a three-link manipulator?

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

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

This is an under-constrained system of equations.

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

If the Jacobian is not a square matrix (or is not full rank), then the inverse doesn’t exist...

- what next?

We have: \( \dot{x} = J\dot{q} \)

We are looking for a matrix \( J^\# \) such that:

\[
\dot{q} = J^\# \dot{x} \quad \rightarrow \quad \dot{x} = J\dot{q}
\]
Two cases:

- Underconstrained manipulator (redundant)
- Overconstrained

Generalized inverse:

- for the underconstrained manipulator: given $\dot{x}$, find any vector $\dot{q}$ s.t.

- for the overconstrained manipulator: given $\dot{x}$, find any vector $\dot{q}$
  
  \[ \dot{x} - J\dot{q} \quad \text{Is minimized} \]
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

\[ \nabla_z f(z) = \lambda \nabla_z g(z) \]

\[ f(\dot{q}) = \frac{1}{2} \dot{q}^T \dot{q} \quad \text{Minimize} \]

\[ g(\dot{q}) = J\dot{q} - \dot{x} = 0 \quad \text{Subject to} \]

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

\[ \nabla_{\dot{q}} g(\dot{q}) = J \]

\[ \dot{q}^T = \lambda^T J \]

\[ \dot{q} = J^T \lambda \]
So, the pseudoinverse calculates the vector of joint velocities that satisfies
\[ \dot{q} = J^T \lambda \]
\[ J \dot{q} = (JJ^T) \lambda \]
\[ \lambda = (JJ^T)^{-1} J \dot{q} \]
\[ \lambda = (JJ^T)^{-1} \dot{x} \]
\[ \dot{q} = J^T \lambda \]
\[ \dot{q} = J^T (JJ^T)^{-1} \dot{x} \]
\[ J^\# = J^T (JJ^T)^{-1} \]
\[ \dot{q} = J^\# \dot{x} \]

I won’t say why, but if \( J \) is full rank, then \( JJ^T \) is invertible.

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:

\[
\begin{align*}
J^\# &= J^T \left( JJ^T \right)^{-1} & \text{Underconstrained case (if there are more columns than rows (} m < n)) \\
J^\# &= \left( J^T J \right)^{-1} J^T & \text{Overconstrained case (if there are more rows than columns (} n < m)) \\
J^\# &= J^{-1} & \text{If there are an equal number of rows and columns (} n = m) \\
\end{align*}
\]

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

Singular value decomposition decomposes a matrix as follows:

\[ J = U \Sigma V^T \]

For an under-constrained matrix, \( \Sigma \) is a diagonal matrix of singular values:

\[
\begin{bmatrix}
\sigma_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \sigma_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \sigma_3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \sigma_n & 0 & 0 & 0 \\
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
\frac{1}{\sigma_1} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{\sigma_2} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sigma_3} & 0 & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{\sigma_n} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ J^\# = V \left( \begin{bmatrix} 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sigma_n} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \end{bmatrix} \right) U^T \]
Properties of the pseudoinverse

Moore-Penrose conditions:

1. \( J^\# J J^\# = J^\# \)
2. \( J J^\# J = J \)
3. \( (J J^\#)^T = J J^\# \)
4. \( (J^\# J)^T = J^\# J \)

Generalized inverse: satisfies condition 1

Reflexive generalized inverse: satisfies conditions 1 and 2

Pseudoinverse: satisfies all four conditions

Other useful properties of the pseudoinverse:

\[
(J^\#)^\# = J
\]

\[
(J^\#)^T = (J^T)^\#
\]
Controlling Cartesian Position

Procedure for controlling position:

1. Calculate position error: $x_{err}$

2. Multiply by a scaling factor: $\delta x_{err} = \alpha x_{err}$

3. Multiply by the velocity Jacobian pseudoinverse: $\dot{q} = J_v \# \alpha x_{err}$
Controlling Cartesian Orientation

How does this strategy work for orientation control?

• Suppose you want to reach an orientation of $R_d$
• Your current orientation is $R_c$
• You’ve calculated a difference: $R_{cd} = R_c^T R_d$
• How do you turn this difference into a desired angular velocity to use in $\dot{q} = J^\# \omega$?
Controlling Cartesian Orientation

You can’t do this:

• Convert the difference to ZYZ Euler angles: $r_{\phi\theta\psi}$
• Multiply the Euler angles by a scaling factor and pretend that they are an angular velocity: $\delta q = \alpha J^# r_{\phi\theta\psi}$

Remember that in general: $J_\omega \neq \frac{\partial r_{\phi\theta\psi}}{\partial q}$

![Diagram](image)
The Analytical Jacobian

If you really want to multiply the angular Jacobian by the derivative of an Euler angle, you have to convert to the “analytical” Jacobian:

$$\frac{\partial r_{\phi\theta\psi}}{\partial q} = T_A(r_{\phi\theta\psi})J_\omega \dot{q}$$

$$J_A = T_A(r_{\phi\theta\psi})J_\omega = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\theta \\ 0 & c_\phi & s_\phi s_\theta \\ 1 & 0 & c_\theta \end{bmatrix} J_\omega$$

For ZYZ Euler angles

Gimbal lock: by using an analytical Jacobian instead of the angular velocity Jacobian, you introduce the gimbal lock problems we talked about earlier into the Jacobian – this essentially adds “singularities” (we’ll talk more about that in a bit….)
The easiest way to handle this Cartesian orientation problem is to represent the error in axis-angle format

\[ \delta r_k = J_{\omega} \dot{q} \]

Axis angle delta rotation

Procedure for controlling rotation:

1. Represent the rotation error in axis angle format: \( r_{\text{err}} \)
2. Multiply by a scaling factor: \( \delta r_{\text{err}} = \alpha r_{\text{err}} \)
3. Multiply by the angular velocity Jacobian pseudoinverse: \( \dot{q} = J_{\omega} \# \alpha r_{\text{err}} \)
Controlling Cartesian Orientation

Why does axis angle work?

- Remember Rodrigues’ formula from before:

\[
R_{k\theta} = e^{S(k)\theta} = I + S(k) \sin(\theta) + S(k)^2 (1 - \cos(\theta))
\]

Compare this to the definition of angular velocity:

\[
^b\dot{\mathbf{p}} = S(^b\mathbf{\omega})^b\mathbf{p}
\]

The solution to this FO diff eqn is:

\[
^b R_{\omega t} = e^{S(^b\mathbf{\omega})t}
\]

Therefore, the angular velocity gets integrated into an axis angle representation.
Jacobian Transpose Control

The story of Cartesian control so far:

1. \( \dot{x} = J\dot{q} \)
2. \( \dot{q} = J^\# \dot{x} \)
Equation: 

\[ e = \frac{1}{2} x_{err}^T x_{err} \]

\[ \frac{\partial e}{\partial q} = -\left(x_{err}^T\right) \frac{\partial x}{\partial q} \]

\[ \dot{q} \leftarrow -\alpha \left( \frac{\partial e}{\partial q} \right)^T \]

\[ \dot{q} = \alpha \left[ \left(x_{err}^T\right) \frac{\partial x}{\partial q} \right]^T \]

\[ \dot{q} = \alpha \frac{\partial x}{\partial q} \left( x_{err} \right) \]

\[ \dot{q} = \alpha J_v^T \left( x_{err} \right) \]
Jacobian Transpose Control

The same approach can be used to control orientation:

\[ \dot{q} = \alpha J^T \omega \left[ k_{curr}^T k_{ref} \right] \]

orientation error: axis angle orientation of reference pose in the current end effector reference frame: \( k_{curr} k_{ref} \)
So, evidently, this is the gradient of that

\[ \dot{q} = J^T(x_{\text{err}}) \]

\[ e = \frac{1}{2} x_{\text{err}}^T x_{\text{err}} \]

- Jacobian transpose control descends a squared error function.
- Gradient descent always follows the steepest gradient
Jacobian Transpose v Pseudoinverse

What gives?

• Which is more direct? Jacobian pseudoinverse or transpose?

\[ \dot{q} = J^T \xi \quad \text{or} \quad \dot{q} = J^\# \xi \]

They do different things:

• Transpose: move toward a reference pose as quickly as possible
  • One dimensional goal (squared distance metric)
• 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 \textit{in} Cartesian space
- Instead, the transpose follows the gradient in \textit{joint space}
Up until now, we’ve used the Jacobian in the twist equation, \( \xi = Jq \).

Interestingly, you can also use the Jacobian in a statics equation:

\[
\tau = J^T w
\]

Joint torques \hspace{1cm} Cartesian wrench:

\[
w = \begin{pmatrix} f \\ m \end{pmatrix}
\]

force \hspace{1cm} moment (torque)
Using the Jacobian for Statics

It turns out that both wrenches and twists can be understood in terms of a representation of displacement known as a *screw*.

- Therefore, you can calculate work by integrating the dot product:

\[
W = \int (v \cdot f + \omega \cdot m) = \int \begin{bmatrix} v \\ \omega \end{bmatrix}^T \begin{bmatrix} f \\ m \end{bmatrix}
\]

Work in Cartesian space

\[
W = \int \tau^T \dot{q}
\]

Work in joint space

Conservation of energy:

\[
\int \tau^T \dot{q} = \int \begin{bmatrix} v \\ \omega \end{bmatrix}^T \begin{bmatrix} f \\ m \end{bmatrix}
\]
Using the Jacobian for Statics

\[ \tau^T \dot{q} = \begin{bmatrix} f^T \\ m \end{bmatrix} \begin{bmatrix} \nu \\ \omega \end{bmatrix} \]

Incremental work (virtual work)

\[ \tau^T \dot{q} = \begin{bmatrix} f^T \\ m \end{bmatrix} J \dot{q} \]

\[ \tau^T = \begin{bmatrix} f^T \\ m \end{bmatrix} J \]

Wrench-twist duality:

\[ \tau = J^T \begin{bmatrix} f \\ m \end{bmatrix} \]

\[ \tau = J^T w \quad \text{vs} \quad \dot{\xi} = J \dot{q} \]

\[ \tau = J^T w \]
Twist: converting between reference frames

Note that twist can be represented in different reference frames:

\[
\begin{align*}
\xi^b &= \begin{bmatrix} v^b \\ \omega^b \end{bmatrix} \\
\xi^k &= \begin{bmatrix} v^k \\ \omega^k \end{bmatrix}
\end{align*}
\]

Consider two reference frames attached to the same rigid body:

\[
\begin{align*}
\omega^b_2 &= \omega^b_1 \\
v^b_2 &= v^b_1 + \omega^b_1 \times r_{12}
\end{align*}
\]
Twist: converting between reference frames

\[ {\mathbf{b}}{\mathbf{\omega}}_2 = {\mathbf{b}}{\mathbf{\omega}}_1 \]

\[ {\mathbf{b}}{\mathbf{v}}_2 = {\mathbf{b}}{\mathbf{v}}_1 + {\mathbf{b}}{\mathbf{\omega}}_1 \times {\mathbf{r}}_{12} \]

\[
\begin{bmatrix}
{\mathbf{b}}{\mathbf{v}}_2 \\
{\mathbf{b}}{\mathbf{\omega}}_2
\end{bmatrix} =
\begin{bmatrix}
I & -{\mathbf{S}}({\mathbf{r}}_{12}) \\
0 & I
\end{bmatrix}
\begin{bmatrix}
{\mathbf{b}}{\mathbf{v}}_1 \\
{\mathbf{b}}{\mathbf{\omega}}_1
\end{bmatrix}
\]

\[
\begin{bmatrix}
{^2}{\mathbf{v}} \\
{^2}{\mathbf{\omega}}
\end{bmatrix} =
\begin{bmatrix}
{^2}{\mathbf{R}}_2^T & 0 \\
0 & {^2}{\mathbf{R}}_2^T
\end{bmatrix}
\begin{bmatrix}
I & -{\mathbf{S}}({\mathbf{r}}_{12}) \\
0 & I
\end{bmatrix}
\begin{bmatrix}
{^2}{\mathbf{R}}_1 & 0 \\
0 & {^2}{\mathbf{R}}_1
\end{bmatrix}
\begin{bmatrix}
{^1}{\mathbf{v}} \\
{^1}{\mathbf{\omega}}
\end{bmatrix}
\]

Twist in frame 2

Twist in frame 1
Wrench: converting between reference frames

Wrench can also be represented in different reference frames:

\[ b_W = \begin{bmatrix} b_f \\ b_m \end{bmatrix} \quad k_W = \begin{bmatrix} k_f \\ k_m \end{bmatrix} \]
Wrench: converting between reference frames

Use the virtual work argument to derive the relationship:

\[
\begin{bmatrix}
  2 f_2 \\ 2 m_2
\end{bmatrix}^T
\begin{bmatrix}
  2 v_2 \\ 2 \omega_2
\end{bmatrix} =
\begin{bmatrix}
  1 f_1 \\ 1 m_1
\end{bmatrix}^T
\begin{bmatrix}
  1 v_1 \\ 1 \omega_1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  2 f_2 \\ 2 m_2
\end{bmatrix}^T
\begin{bmatrix}
  2 R_1 & -2 R_1 S(1_{r_{12}}) \\
  0 & 2 R_1
\end{bmatrix}
\begin{bmatrix}
  1 v_1 \\ 1 \omega_1
\end{bmatrix} =
\begin{bmatrix}
  1 f_1 \\ 1 m_1
\end{bmatrix}^T
\begin{bmatrix}
  1 v_1 \\ 1 \omega_1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  2 f_2 \\ 2 m_2
\end{bmatrix}^T
\begin{bmatrix}
  2 R_1 & -2 R_1 S(1_{r_{12}}) \\
  0 & 2 R_1
\end{bmatrix}
\begin{bmatrix}
  1 v_1 \\ 1 \omega_1
\end{bmatrix} =
\begin{bmatrix}
  1 f_1 \\ 1 m_1
\end{bmatrix}^T
\begin{bmatrix}
  1 R_2 \\
  S(1_{r_{12}}) R_2
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 f_1 \\ 1 m_1
\end{bmatrix} =
\begin{bmatrix}
  1 R_2 & 0 \\
  S(1_{r_{12}}) R_2 & 1 R_2
\end{bmatrix}
\begin{bmatrix}
  2 f_2 \\ 2 m_2
\end{bmatrix}
\]
Use a 6-axis load cell bisecting the second link to calculate wrenches at the end effector (the tip of the last link)

\[
\begin{bmatrix}
c_3 & s_3 & 0 \\
-s_3 & c_3 & 0 \\
0 & 0 & 1
\end{bmatrix}
\text{eff } R_{\text{sensor}}
\]

\[
\begin{bmatrix}
-l_3 - \frac{l_2}{2} c_3 \\
\frac{l_2}{2} s_3 \\
0
\end{bmatrix}
\text{eff } r_{\text{sensor}}
\]
Converting wrenches: Example

\[
\begin{bmatrix}
  f_{\text{eff}} \\
  m_{\text{eff}}
\end{bmatrix} =
S \begin{bmatrix}
  R_{\text{sensor}}^{\text{eff}} \\
  r_{\text{eff}, \text{sensor}}^{\text{eff}}
\end{bmatrix} R_{\text{sensor}}^{\text{eff}}
\begin{bmatrix}
  0 \\
  f_{\text{sensor}}^{\text{eff}} \\
  m_{\text{sensor}}^{\text{eff}}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  f_{\text{eff}} \\
  m_{\text{eff}}
\end{bmatrix} =
\begin{bmatrix}
  c_3 & s_3 & 0 & 0 & 0 & 0 \\
  -s_3 & c_3 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & l_2 s_3 & c_3 & s_3 \\
  0 & 0 & 0 & l_2 c_3 - l_2 s_3 & -s_3 & c_3 \\
  l_3 s_3 & -l_3 c_3 & -l_2 c_3^2 & -l_2 s_3^2 & 0 & 0 & 0 & 1
\end{bmatrix}
\]