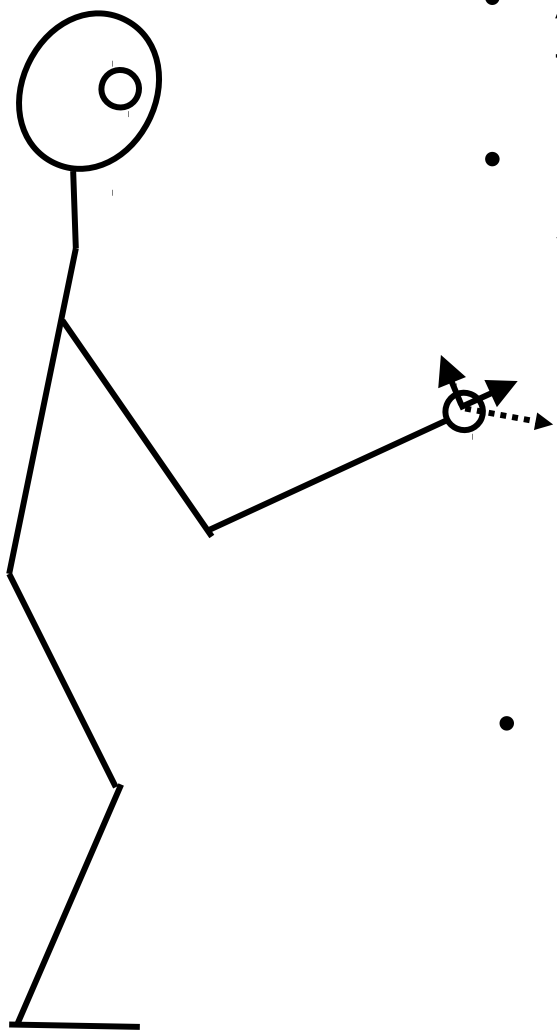
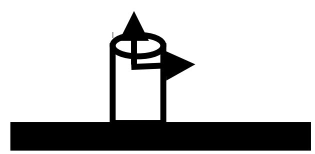
Cartesian Control



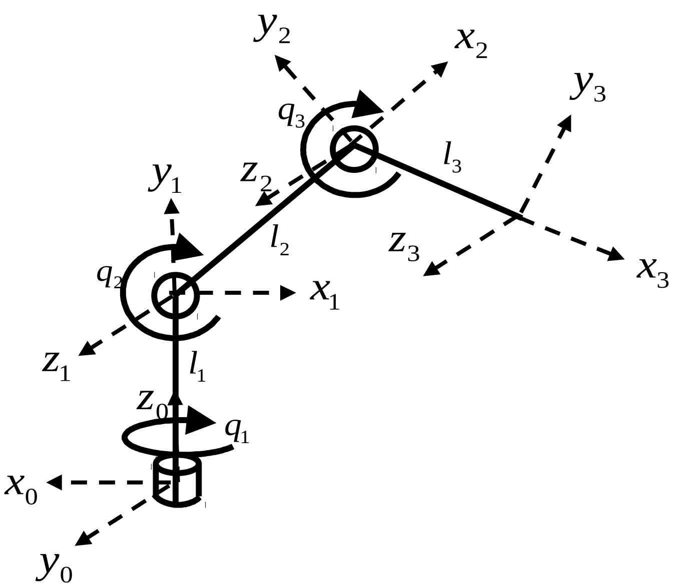
- Analytical inverse kinematics can be difficult to derive
- Inverse kinematics are not as well suited for small differential motions



- 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:

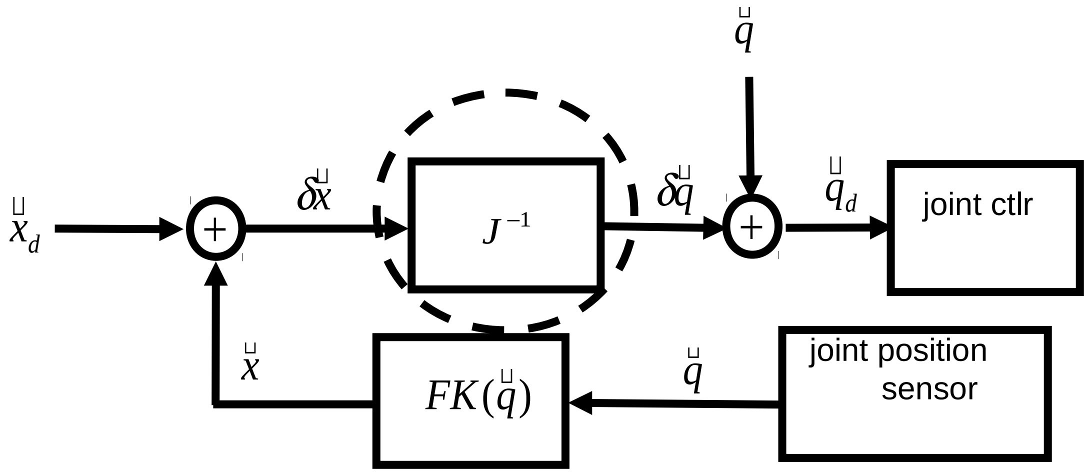


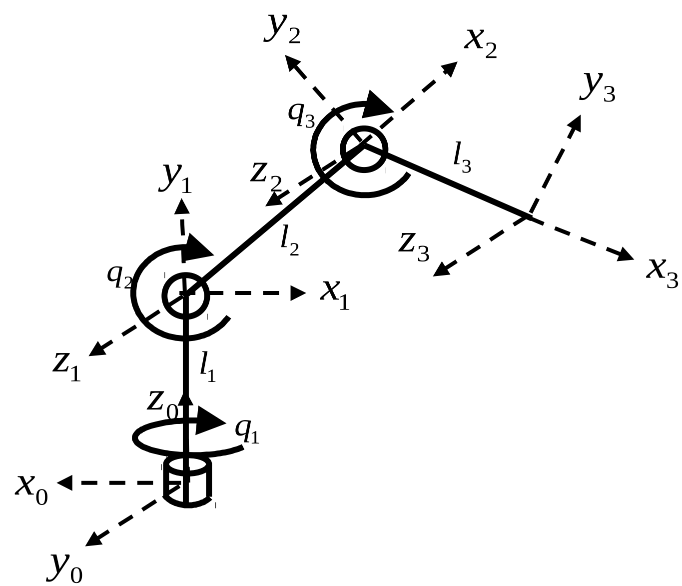
$$J = \begin{pmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2c_2 + l_3c_{23}) & -l_3c_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{pmatrix}$$

We can use the same strategy that we used before:

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

Cartesian control





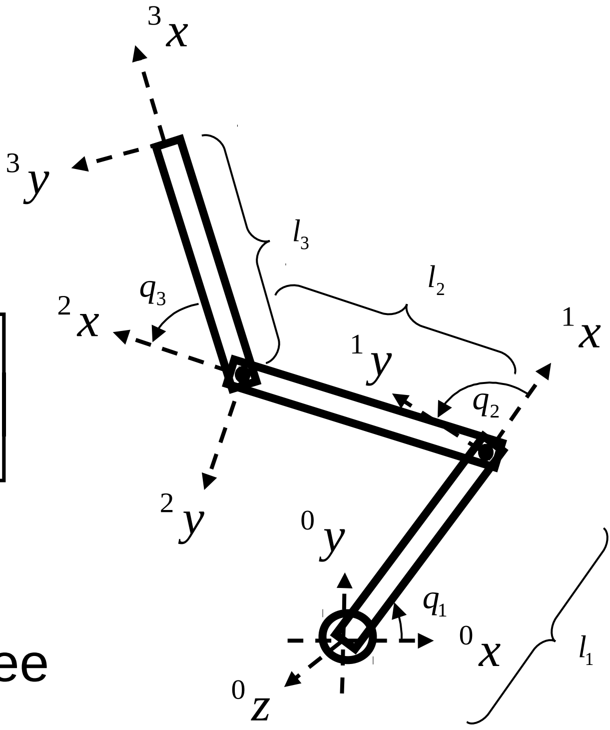
$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

However, this only works if the Jacobian is square and full rank…

- All rows/columns are linearly independent, or
- Columns span Cartesian space, or
- Determinant is not zero

Cartesian control

What if you want to control the two-dimensional position of a three-link manipulator?



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

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

← Two equations of three variables each…

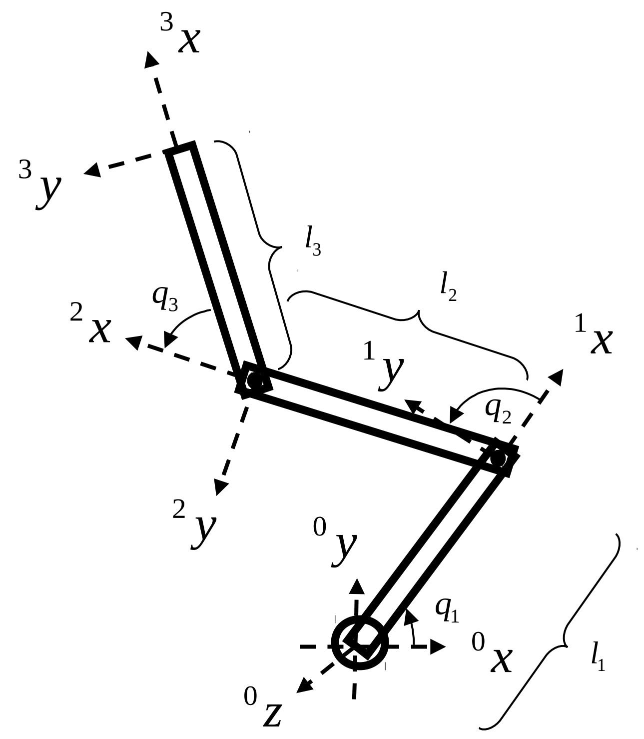
This is an *under-constrained* system of equations.

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

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:

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

Generalized inverse

Two cases:

- Underconstrained manipulator (redundant)
- Overconstrained

Generalized inverse:

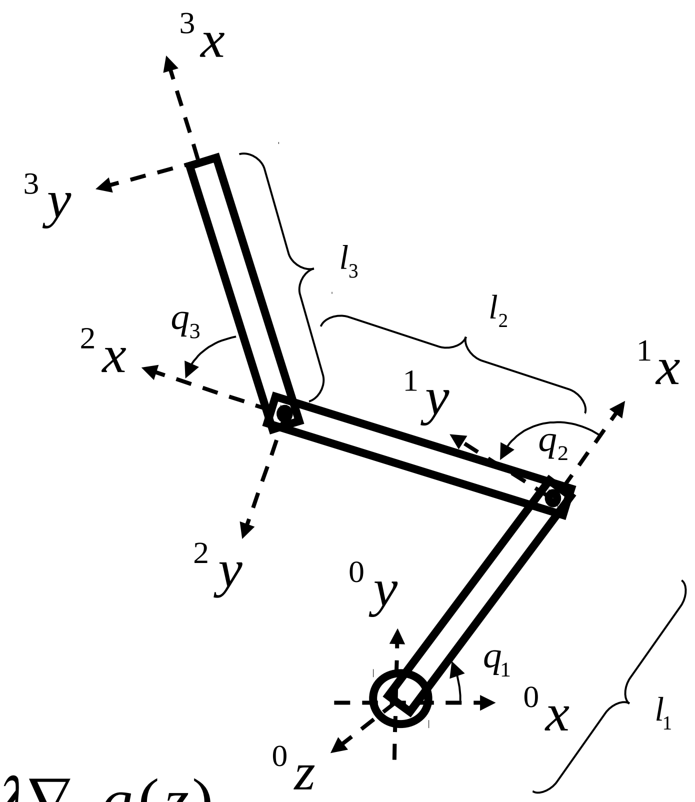
- for the underconstrained manipulator: given $\dot{x}$, find any vector $\dot{q}$ that minimizes $\dot{q}^T \dot{q}$ s.t. $\dot{x} - J\dot{q}$
- for the overconstrained manipulator: given $\dot{x}$, find any vector $\dot{q}$ s.t. $\dot{x} - J\dot{q}$ Is minimized

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):

Calculating the pseudoinverse using SVD

Singular value decomposition decomposes a matrix as follows:

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

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

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

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

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

Properties of the pseudoinverse

Moore-Penrose conditions:

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

Generalized inverse: satisfies condition 1

Reflexive generalized inverse: satisfies conditions 1 and 2

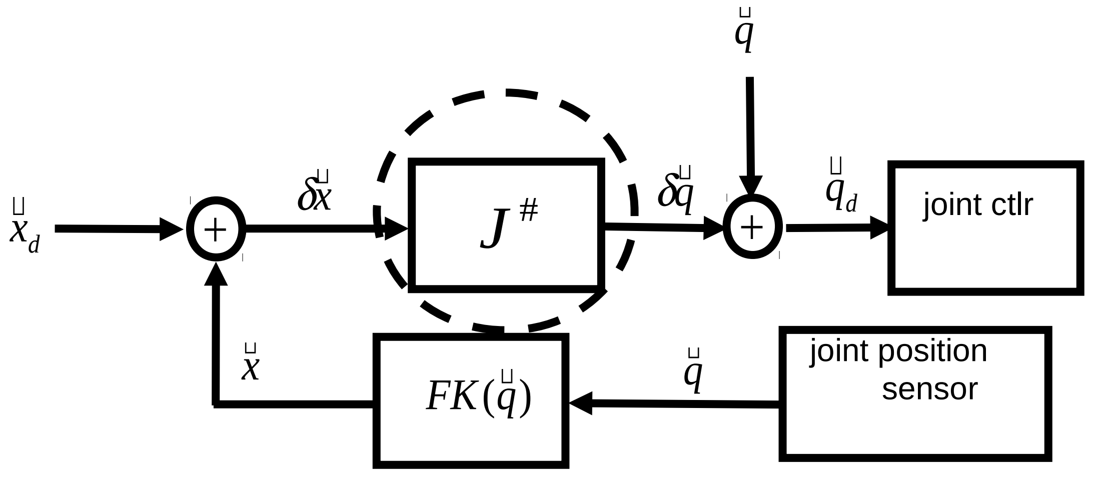
Pseudoinverse: satisfies all four conditions

Other useful properties of the pseudoinverse:

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

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

Controlling Cartesian Position



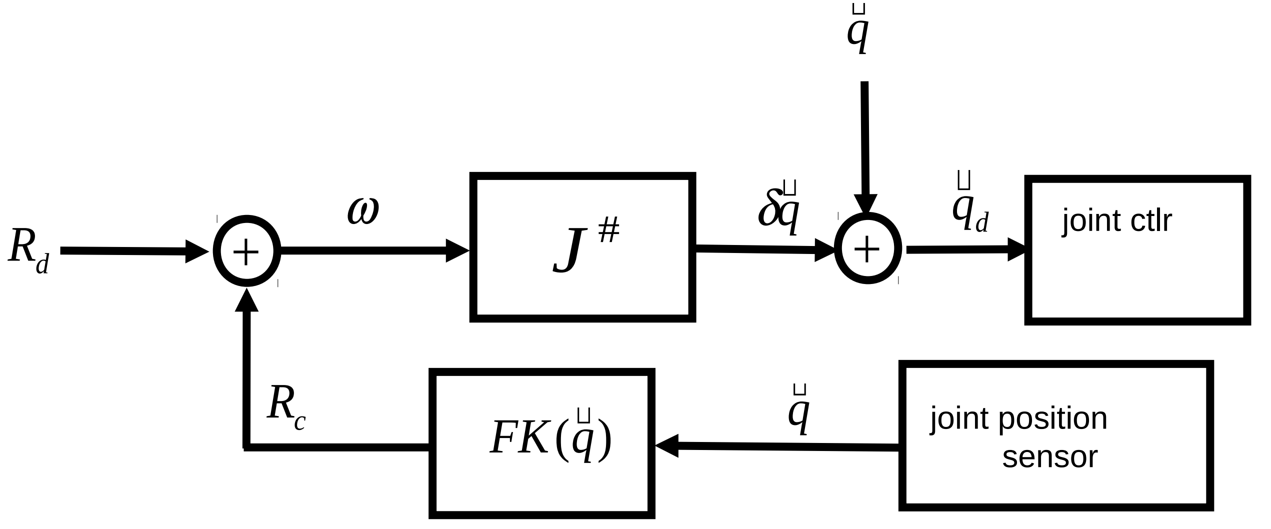
Procedure for controlling position:

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 $\dot{q} = J^{\#}\omega$?

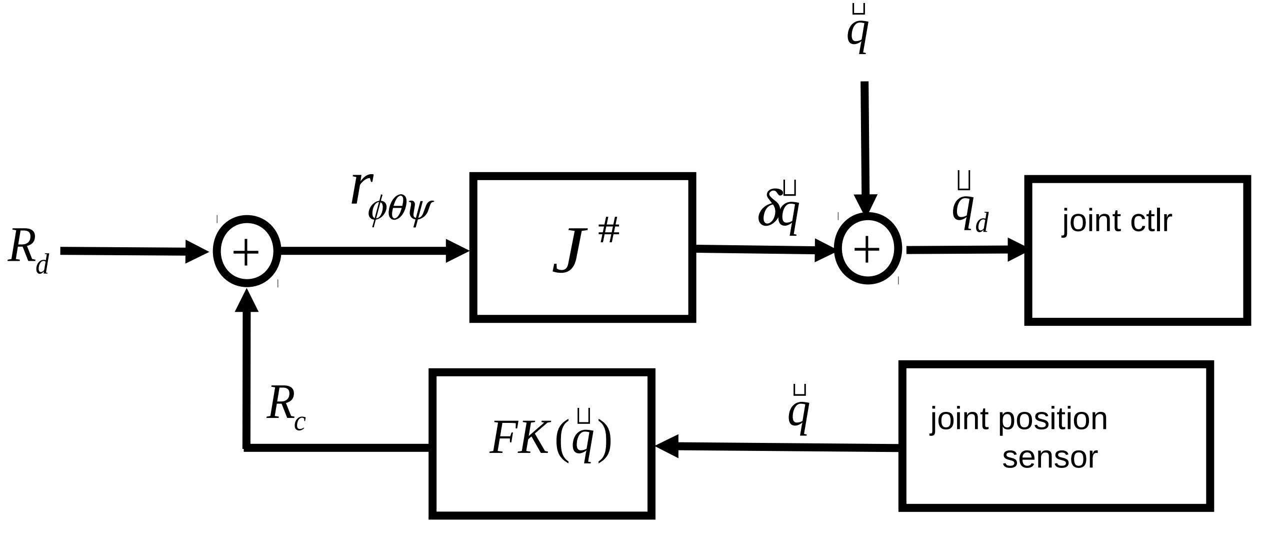


Controlling Cartesian Orientation

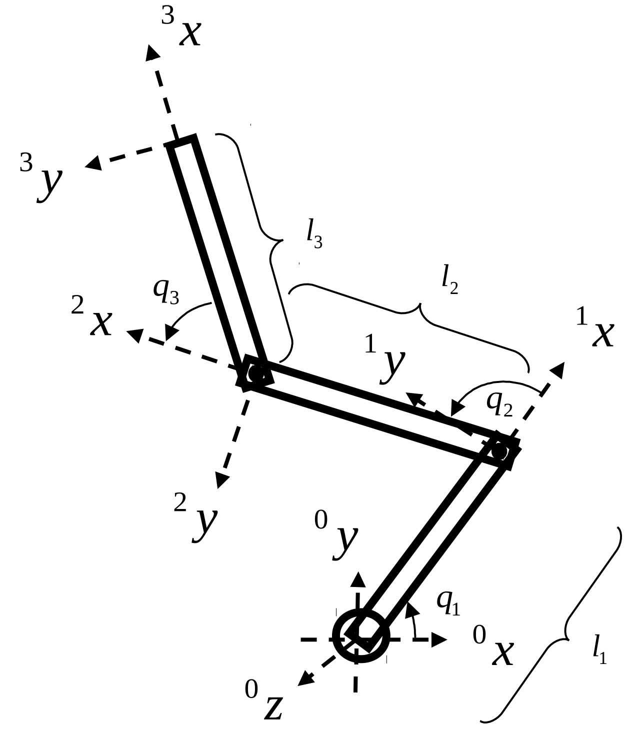
You **can't** do this:

- 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: $J_{\omega} \neq \dfrac{\partial r_{\phi\theta\psi}}{\partial q}$



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:

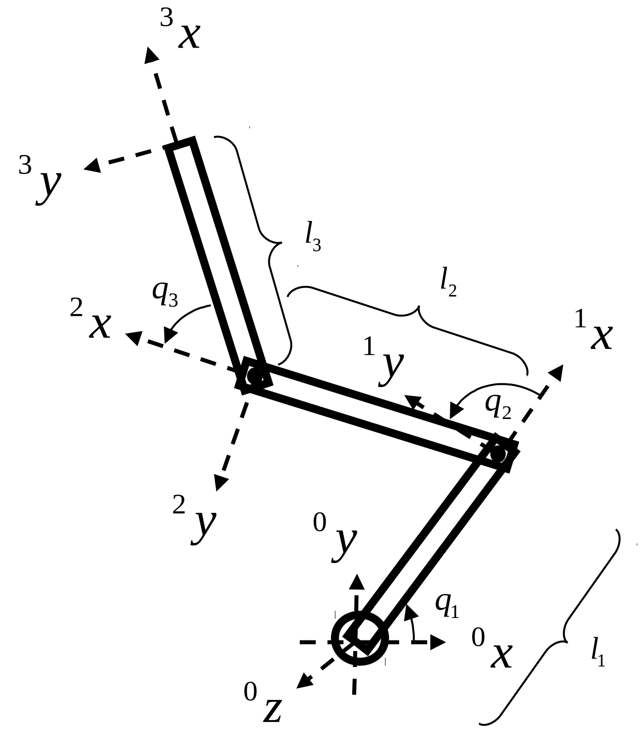
$$\frac{\partial r_{\phi\theta\psi}}{\partial q} = T_A\left(r_{\phi\theta\psi}\right) J_\omega \dot{q}$$

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

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…)

Controlling Cartesian Orientation



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_{err}
2. Multiply by a scaling factor: $\delta r_{err} = \alpha r_{err}$
3. Multiply by the angular velocity Jacobian pseudoinverse: $\dot{q} = J_\omega^{\#} \alpha r_{err}$

Controlling Cartesian Orientation

Why does axis angle work?

- Remember Rodrigues' formula from before:

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

axis angle

Compare this to the definition of angular velocity: $^{b}\dot{p} = S\left({}^{b}\omega\right){}^{b}p$

The solution to this FO diff eqn is: $^{b}R_{\omega t} = e^{S\left({}^{b}\omega\right)t}$

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

Jacobian Transpose Control

Here's another approach:

$$e = \tfrac{1}{2} x_{err}{}^T x_{err}$$

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

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

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

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

$$\dot{q} = \alpha J_v{}^T \left(x_{err}\right)$$

Start with a squared position error function (assume the poses are represented as row vectors)

Position error: $x_{err} = x_{ref} - x$

Gradient descent: take steps proportional to α in the direction of the negative gradient.

Jacobian Transpose Control

The same approach can be used to control orientation:

$$\dot{q} = \alpha J_{\omega}{}^{T} \left({}^{curr}k_{ref} \right)$$

orientation error: axis angle orientation of reference pose in the current end effector reference frame: ${}^{curr}k_{ref}$

Jacobian Transpose Control

So, evidently, this is the gradient of that

$$\dot{q} = J^T (x_{err}) \qquad\qquad e = \tfrac{1}{2} x_{err}^{\ T} x_{err}$$

- 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?

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

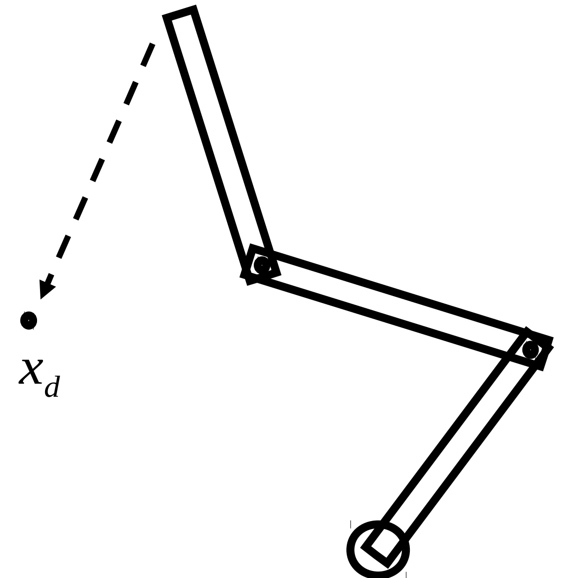
They do different things:

- 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

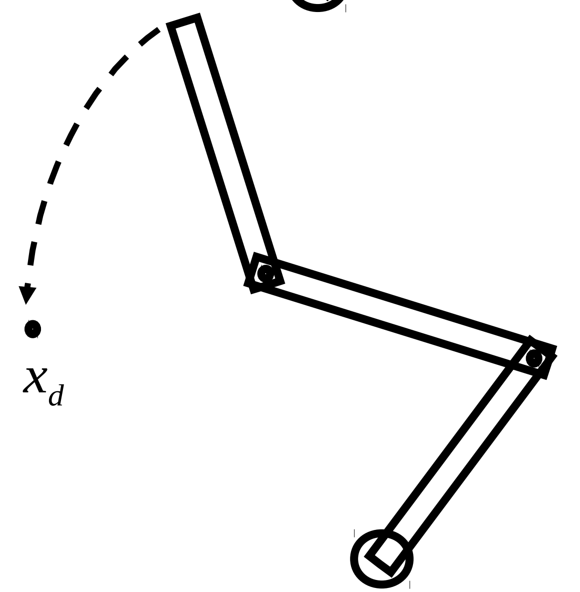
The pseudoinverse moves the end effector in a straight line path toward the goal pose using the least squared joint velocities.

- 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*



Using the Jacobian for Statics

Up until now, we've used the Jacobian in the twist equation, $\xi = J\dot{q}$

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

$$\tau = J^T w$$

Joint torques

Cartesian wrench: $w = \begin{pmatrix} f \\ m \end{pmatrix}$

force

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

$$\tau^T \dot{q} = \begin{bmatrix} f \\ m \end{bmatrix}^T \begin{bmatrix} v \\ \omega \end{bmatrix}$$

← Incremental work (virtual work)

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

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

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

$$\tau = J^T w$$

Wrench-twist duality:

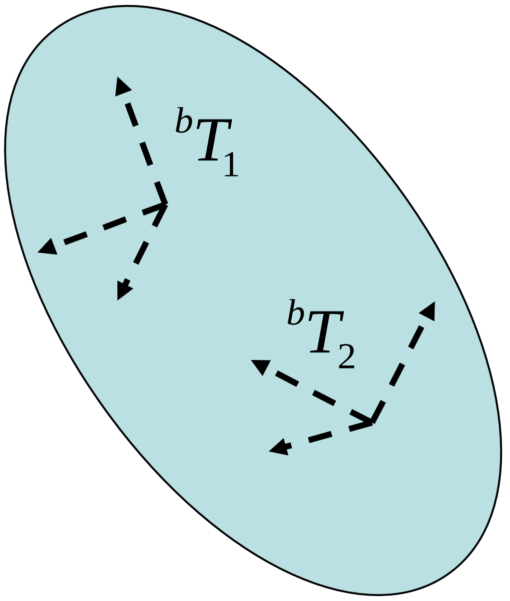
$\tau = J^T w$ vs $\xi = J\dot{q}$

Twist: converting between reference frames

Note that twist can be represented in different reference frames:

$$ {}^{b}\xi = \begin{bmatrix} {}^{b}v \\ {}^{b}\omega \end{bmatrix} \qquad {}^{k}\xi = \begin{bmatrix} {}^{k}v \\ {}^{k}\omega \end{bmatrix} $$

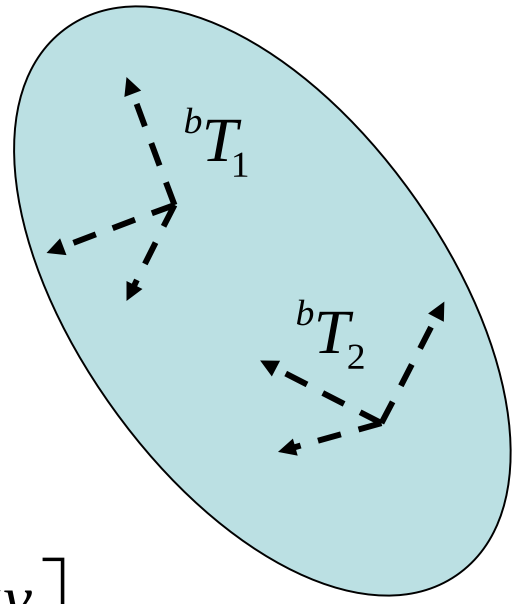
Consider two reference frames attached to the same rigid body:



$$ {}^{b}\omega_2 = {}^{b}\omega_1 $$

$$ {}^{b}v_2 = {}^{b}v_1 + {}^{b}\omega_1 \times r_{12} $$

Twist: converting between reference frames



$$^b\omega_2 = {}^b\omega_1$$

$$^bv_2 = {}^bv_1 + {}^b\omega_1 \times r_{12}$$

$$\begin{bmatrix} ^bv_2 \\ ^b\omega_2 \end{bmatrix} = \begin{bmatrix} I & -S(r_{12}) \\ 0 & I \end{bmatrix} \begin{bmatrix} ^bv_1 \\ ^b\omega_1 \end{bmatrix}$$

$$\begin{bmatrix} ^2v \\ ^2\omega \end{bmatrix} = \begin{bmatrix} ^bR_2{}^T & 0 \\ 0 & ^bR_2{}^T \end{bmatrix} \begin{bmatrix} I & -S(r_{12}) \\ 0 & I \end{bmatrix} \begin{bmatrix} ^bR_1 & 0 \\ 0 & ^bR_1 \end{bmatrix} \begin{bmatrix} ^1v \\ ^1\omega \end{bmatrix}$$

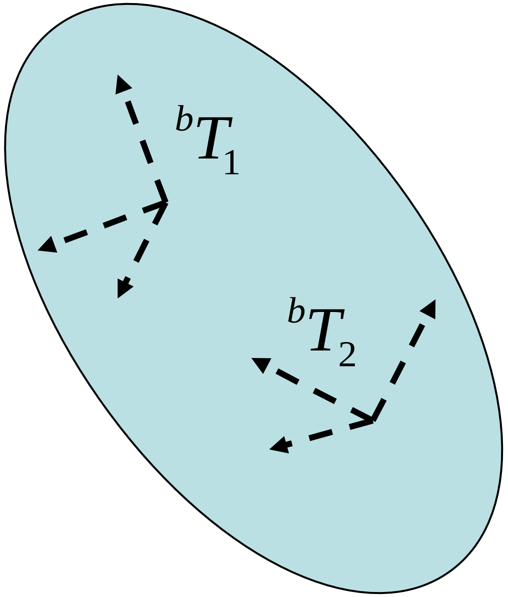
$$\begin{bmatrix} ^2v \\ ^2\omega \end{bmatrix} = \begin{bmatrix} ^2R_1 & -{}^2R_1 S({}^1r_{12}) \\ 0 & ^2R_1 \end{bmatrix} \begin{bmatrix} ^1v \\ ^1\omega \end{bmatrix}$$

Twist in frame *2* — Twist in frame *1*

Wrench: converting between reference frames

Wrench can also be represented in different reference frames:

$$ {}^{b}w = \begin{bmatrix} {}^{b}f \\ {}^{b}m \end{bmatrix} \qquad {}^{k}w = \begin{bmatrix} {}^{k}f \\ {}^{k}m \end{bmatrix} $$



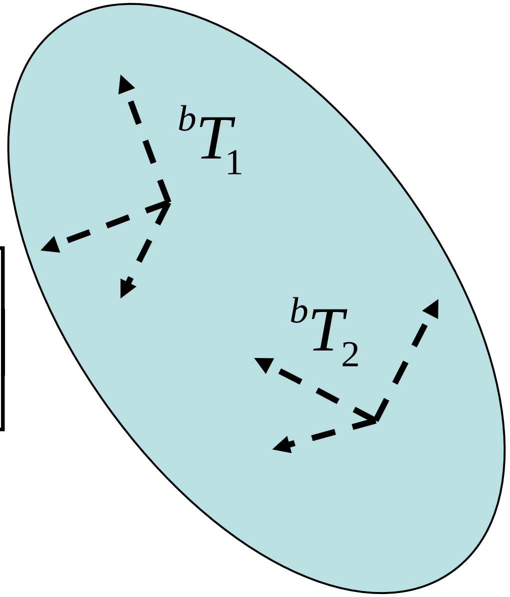
Wrench: converting between reference frames

Use the virtual work argument to derive the relationship:

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

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

$$\begin{bmatrix} {}^2f_2 \\ {}^2m_2 \end{bmatrix}^T \begin{bmatrix} {}^2R_1 & -{}^2R_1 S\left({}^1r_{12}\right) \\ 0 & {}^2R_1 \end{bmatrix} = \begin{bmatrix} {}^1f_1 \\ {}^1m_1 \end{bmatrix}^T$$



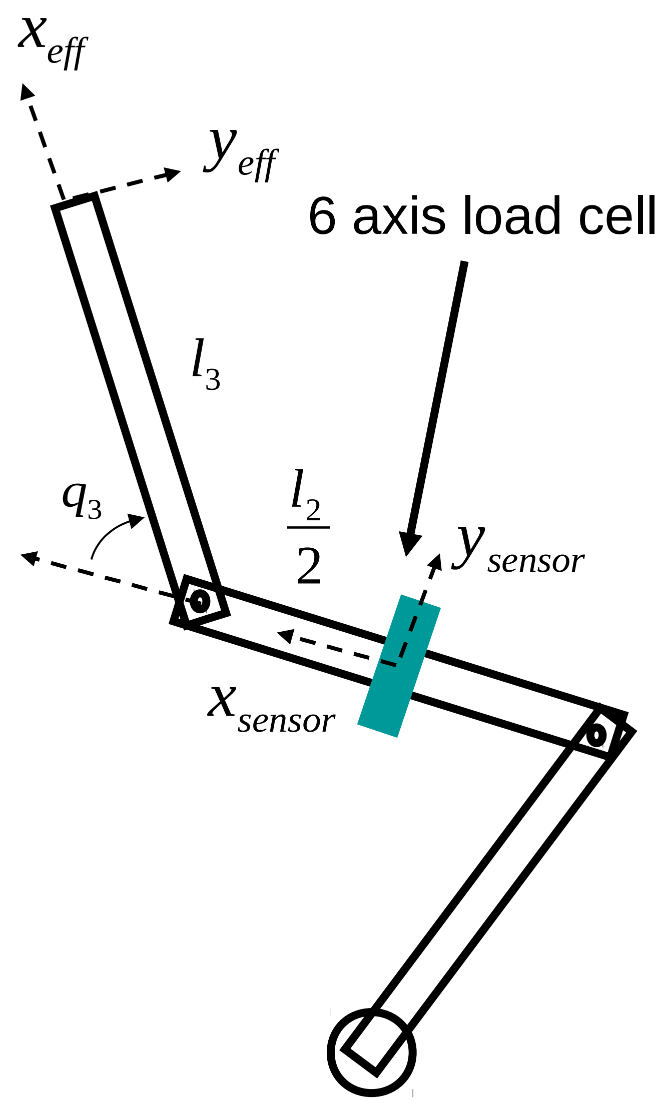
$$\begin{bmatrix} {}^1f_1 \\ {}^1m_1 \end{bmatrix} = \begin{bmatrix} {}^1R_2 & 0 \\ S\left({}^1r_{12}\right){}^1R_2 & {}^1R_2 \end{bmatrix} \begin{bmatrix} {}^2f_2 \\ {}^2m_2 \end{bmatrix}$$

Converting wrenches: Example

Use a 6-axis load cell bisecting the second link to calculate wrenches at the end effector (the tip of the last link)

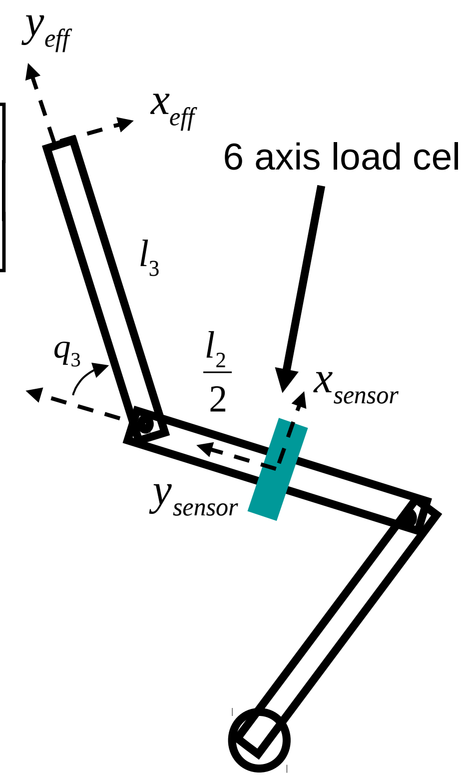
$$ {}^{eff}R_{sensor} = \begin{pmatrix} c_3 & s_3 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{eff}r_{sensor} = \begin{pmatrix} -l_3 - \frac{l_2}{2} c_3 \\ \frac{l_2}{2} s_3 \\ 0 \end{pmatrix} $$



Converting wrenches: Example

$$\begin{bmatrix} {}^{eff}f_{eff} \\ {}^{eff}m_{eff} \end{bmatrix} = \begin{bmatrix} {}^{eff}R_{sensor} & 0 \\ S\left({}^{eff}r_{eff,sensor}\right){}^{eff}R_{sensor} & {}^{eff}R_{sensor} \end{bmatrix} \begin{bmatrix} {}^{sensor}f_{sensor} \\ {}^{sensor}m_{sensor} \end{bmatrix}$$



$$\begin{bmatrix} {}^{eff}f_{eff} \\ {}^{eff}m_{eff} \end{bmatrix} = \begin{bmatrix} c_3 & s_3 & 0 & 0 & 0 & 0 \\ -s_3 & c_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{l_2}{2}s_3 & c_3 & s_3 & 0 \\ 0 & 0 & l_3 + \frac{l_2}{2}c_3 & -s_3 & c_3 & 0 \\ l_3 s_3 & -l_3 c_3 - \frac{l_2}{2}c_3{}^2 - \frac{l_2}{2}s_3{}^2 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{sensor}f_{sensor} \\ {}^{sensor}m_{sensor} \end{bmatrix}$$