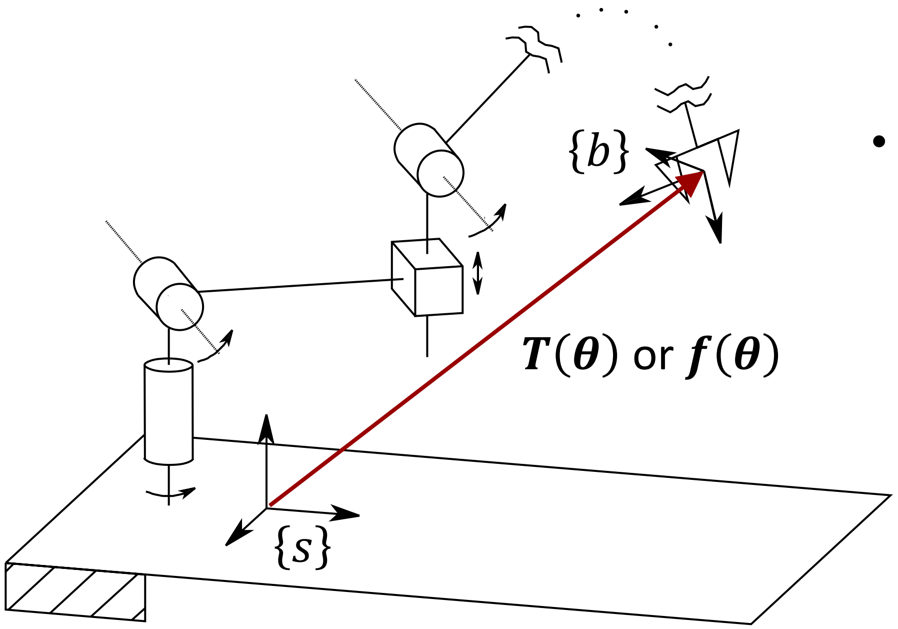
Ch7: Inverse Kinematics

Inverse Kinematics

Inverse Kinematics

The inverse kinematics of a robot refers to the calculation of the joint coordinates $\boldsymbol{\theta}$ from the position and orientation (**pose**) of its end-effector frame.


$\{b\}$
$\{s\}$
$\boldsymbol{T}(\boldsymbol{\theta})$ or $\boldsymbol{f}(\boldsymbol{\theta})$

- "Geometric" inverse kinematics:

$$\text{Given } \boldsymbol{T}_{sb} = \boldsymbol{T}(\boldsymbol{\theta}) \in SE(3)\text{, Find } \boldsymbol{\theta} \in \mathbb{R}^n$$

$$\boldsymbol{T}: \mathbb{R}^n \to SE(3)$$

- "Minimum-Coordinate" inverse kinematics:

$$\text{Given } \boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\theta}) \in \mathbb{R}^r\text{, Find } \boldsymbol{\theta} \in \mathbb{R}^n$$

$$\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^r$$

Complexities of Inverse Kinematics

- The equations to solve are in general nonlinear. Thus, it is not always possible to find a closed-form solution.
- Multiple (finite) solutions may exist.
- Infinite solutions may exist (e.g., when a manipulator is kinematically redundant for the task).
- There might be no admissible solutions (e.g., when the given EE pose does not belong to the manipulator reachable workspace).
- At singular configurations, finite (but different in number from the non-singular configurations) or infinite solutions may exist.

► **Solving Inverse Kinematics Problems**:

- **Analytic Methods** [Faster, but more difficult or impossible]
- **Iterative Numerical Methods** [Slower, but easier to set up]

Analytic Methods

Analytic Methods

Analytic Methods: Finding closed-form solutions using algebraic methods or geometric inspection. If closed-form solutions exists, these methods are preferred.

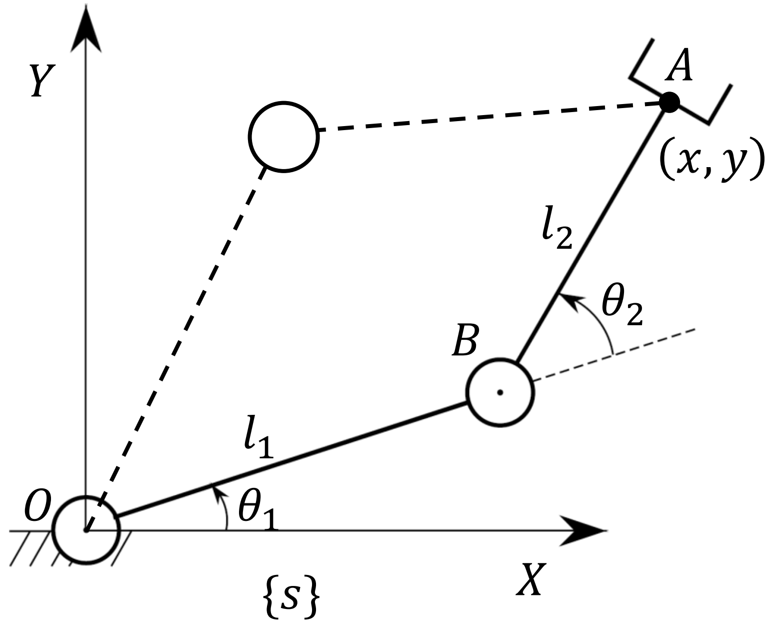
Forward Kinematics a 2R Robot:

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

Inverse Kinematics a 2R Robot :

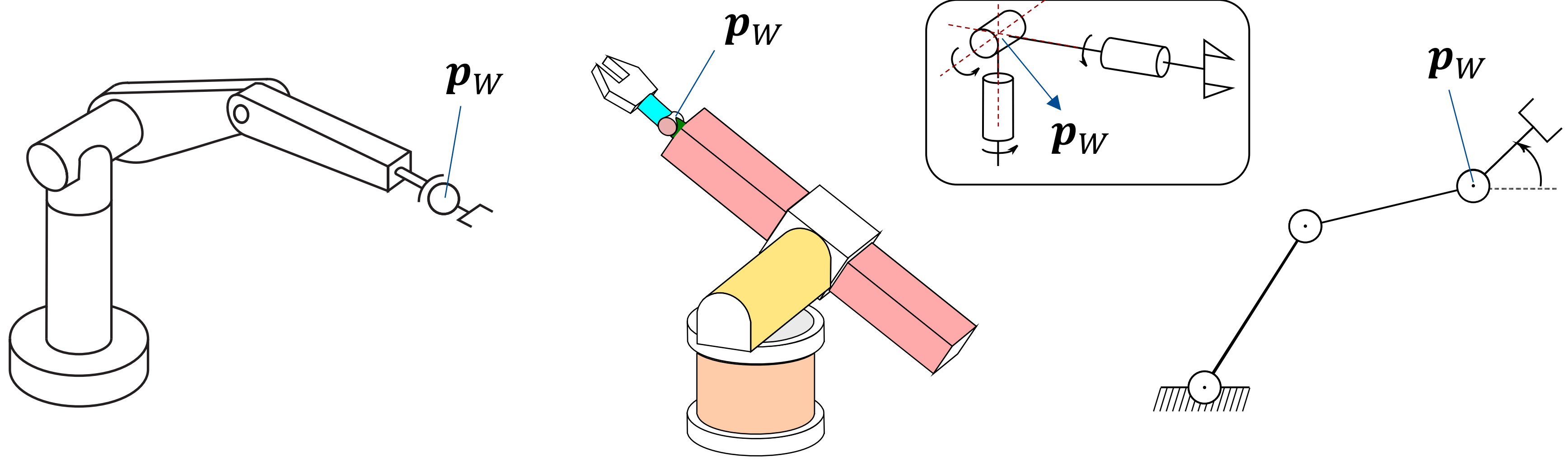
$$\theta_2 = \operatorname{atan2}\left(\pm\sqrt{1-u^2}, u\right) \in (-\pi, \pi]$$
$$\theta_1 = \operatorname{atan2}(y, x) - \operatorname{atan2}(l_2 \sin\theta_2, l_1 + l_2\cos\theta_2)$$

$$u = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$



Decoupling Arm and Wrist

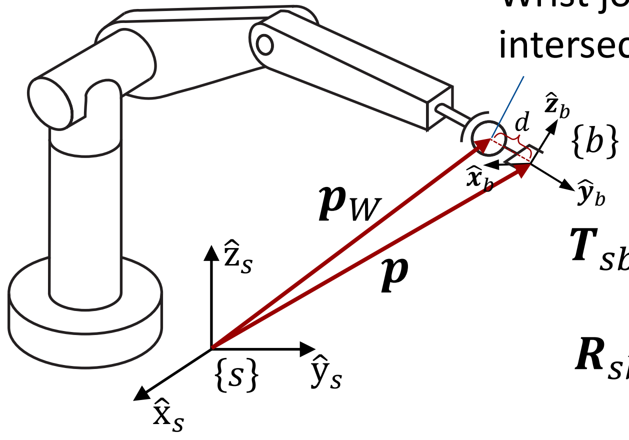
Most of the existing manipulators are typically formed by an **arm** and a **spherical wrist** (where the last three consecutive revolute joint axes intersect at a common point $\boldsymbol{p}_W$). Thus, we can <u>decouple</u> the inverse kinematic solution for the position (i.e., point $\boldsymbol{p}_W$ at the intersection of the three revolute axes) from that for the orientation.



PUMA Arm (6R) Stanford Arm (RRPRRR) 3R Planar Arm

∗ Therefore, it is possible to solve the inverse kinematics for the arm separately from the inverse kinematics for the spherical wrist.

Decoupling Arm and Wrist



Wrist joints (the last three revolute joints) intersect orthogonally at a common point $\boldsymbol{p}_W$.

$$\boldsymbol{T}_{sb} = \begin{bmatrix} \boldsymbol{R}_{sb} & \boldsymbol{p} \\ \boldsymbol{0} & 1 \end{bmatrix} = e^{[\mathcal{S}_1]\theta_1} e^{[\mathcal{S}_2]\theta_2} e^{[\mathcal{S}_3]\theta_3} e^{[\mathcal{S}_4]\theta_4} e^{[\mathcal{S}_5]\theta_5} e^{[\mathcal{S}_6]\theta_6} \boldsymbol{M}$$

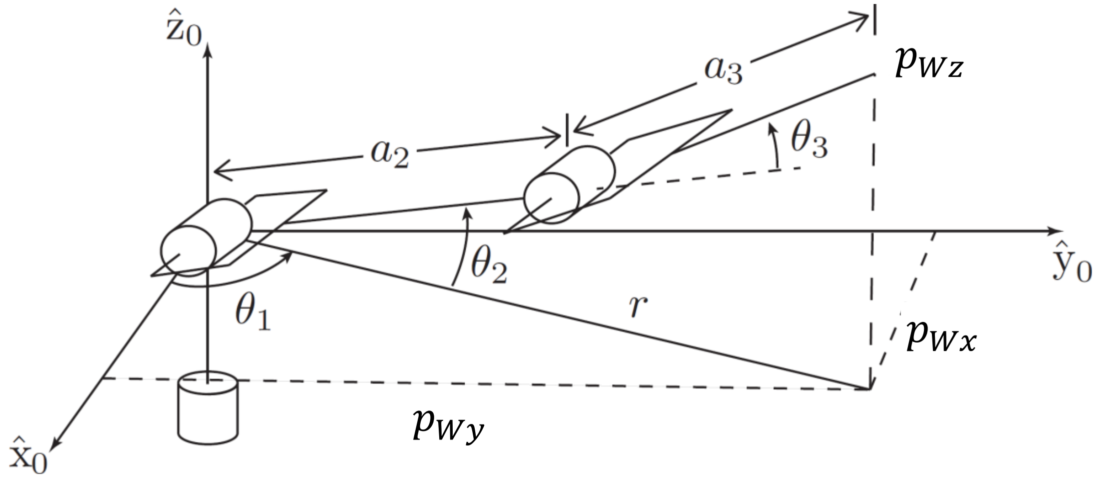
$$\boldsymbol{R}_{sb} = [\hat{\boldsymbol{x}}_b \quad \hat{\boldsymbol{y}}_b \quad \hat{\boldsymbol{z}}_b]$$

Step 1: By having $\boldsymbol{p}$, we can find $\boldsymbol{p}_W = (p_{Wx}, p_{Wy}, p_{Wz})$ (e.g., $\boldsymbol{p}_W = \boldsymbol{p} - d\hat{\boldsymbol{y}}_b$), and then, we find $(\theta_1, \theta_2, \theta_3)$ by inverse position kinematics.

Step 2: By having $\boldsymbol{T}_{sb}$, $\boldsymbol{M}$, and $(\theta_1, \theta_2, \theta_3)$, we find $(\theta_4, \theta_5, \theta_6)$ by inverse orientation kinematics (e.g., using Euler angles) for $\boldsymbol{R}'$:

$$e^{[\mathcal{S}_4]\theta_4} e^{[\mathcal{S}_5]\theta_5} e^{[\mathcal{S}_6]\theta_6} = \underbrace{e^{-[\mathcal{S}_3]\theta_3} e^{-[\mathcal{S}_2]\theta_2} e^{-[\mathcal{S}_1]\theta_1} \boldsymbol{T}_{sb} \boldsymbol{M}^{-1} = \boldsymbol{T}' = (\boldsymbol{R}', \boldsymbol{p}')}_{\text{known}}$$

Example 1: 6R PUMA-Type Arms



$$p_{Wx} = c_1 r = c_1(a_2c_2 + a_3c_{23})$$
$$p_{Wy} = c_1 r = s_1(a_2c_2 + a_3c_{23})$$
$$p_{Wz} = a_2s_2 + a_3s_{23}$$

❖ Inverse position problem of finding $(\theta_1, \theta_2, \theta_3)$ using algebraic intuition:

$$p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2 = a_2^2 + a_3^2 + 2a_2a_3c_3$$

(else, $\boldsymbol{p}$ is out of workspace!)

$$c_3 = \frac{p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2 - a_2^2 - a_3^2}{2a_2a_3} \in [-1,1] \Rightarrow \theta_3 = \operatorname{atan2}(s_3, c_3) \Rightarrow \begin{matrix} \theta_{3,\mathrm{I}} \in (-\pi, \pi] \\ \theta_{3,\mathrm{II}} = -\theta_{3,\mathrm{I}} \end{matrix}$$

$$s_3 = \pm\sqrt{1 - c_3^2}$$

If $c_3 = \pm 1$, singular configuration, on the boundary of the workspace, $\theta_{3,\mathrm{II}} = \theta_{3,\mathrm{I}} = 0$ or π)

Example 1: 6R PUMA-Type Arms (cont.)

$$p_{Wx}^2 + p_{Wy}^2 = r^2 = (a_2 c_2 + a_3 c_{23})^2 \longrightarrow \left.\begin{aligned} a_2 c_2 + a_3 c_{23} &= \pm\sqrt{p_{Wx}^2 + p_{Wy}^2} = \pm r \\ p_{Wz} &= a_2 s_2 + a_3 s_{23} \\ s_{23} &= s_2 c_3 + s_3 c_2 \\ c_{23} &= c_2 c_3 - s_2 s_3 \end{aligned}\right\} \Rightarrow$$

$$c_2 = \frac{\pm\sqrt{p_{Wx}^2 + p_{Wy}^2}(a_2 + a_3 c_3) + p_{Wz} a_3 s_3}{a_2^2 + a_3^2 + 2a_2 a_3 c_3}$$

$$s_2 = \frac{p_{Wz}(a_2 + a_3 c_3) - \left(\pm\sqrt{p_{Wx}^2 + p_{Wy}^2}\, a_3 s_3\right)}{a_2^2 + a_3^2 + 2a_2 a_3 c_3}$$

If $a_2^2 + a_3^2 + 2a_2 a_3 c_3 = 0$ (i.e., $p_{Wx}^2 + p_{Wy}^2 + p_{Wz}^2 = 0$), singular configuration, θ_2 is undefined, infinite solutions for θ_2!)

$$\Rightarrow \quad \theta_2 = \operatorname{atan2}(s_2, c_2)$$

For each θ_3, we have two solutions for θ_2:

$$\begin{cases} \theta_{3,\mathrm{I}} \begin{cases} \theta_{2,\mathrm{I}} & (+r) \\ \theta_{2,\mathrm{II}} & (-r) \end{cases} \\ \theta_{3,\mathrm{II}} \begin{cases} \theta_{2,\mathrm{III}} & (+r) \\ \theta_{2,\mathrm{IV}} & (-r) \end{cases} \end{cases}$$

$$\theta_{3,\mathrm{I}} \to (\theta_{2,\mathrm{I}}, \theta_{2,\mathrm{II}})$$
$$\theta_{3,\mathrm{II}} \to (\theta_{2,\mathrm{III}}, \theta_{2,\mathrm{IV}})$$

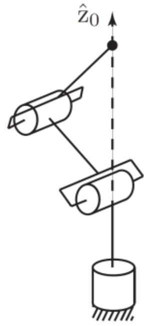
Example 1: 6R PUMA-Type Arms (cont.)

$$p_{Wx} = c_1 r = c_1(a_2c_2 + a_3c_{23})$$
$$p_{Wy} = s_1 r = s_1(a_2c_2 + a_3c_{23})$$
$$a_2c_2 + a_3c_{23} = \pm\sqrt{p_{Wx}^2 + p_{Wy}^2} = \pm r$$

$$\Rightarrow \quad p_{Wx} = \pm c_1\sqrt{p_{Wx}^2 + p_{Wy}^2}, \quad p_{Wy} = \pm s_1\sqrt{p_{Wx}^2 + p_{Wy}^2}$$

$$\Rightarrow \quad \theta_{1,\mathrm{I}} = \operatorname{atan2}(p_{Wy}, p_{Wx}), \quad \theta_{1,\mathrm{II}} = \operatorname{atan2}(-p_{Wy}, -p_{Wx})$$

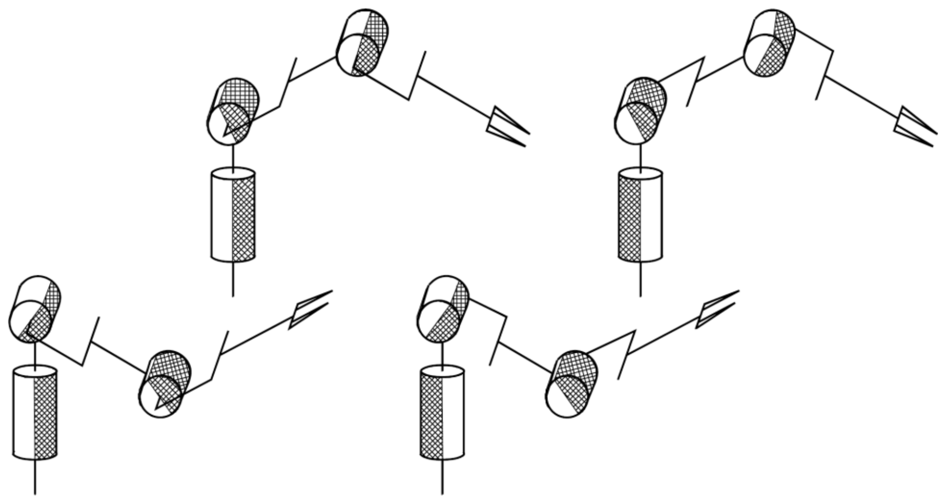
If $\sqrt{p_{Wx}^2 + p_{Wy}^2} = 0$ (i.e., $p_{Wx} = p_{Wy} = 0$), singular configuration, θ_1 is undefined, infinite solutions for θ_1!)



Thus, in total, there exist four solutions:

$$\begin{cases} \theta_{3,\mathrm{I}} \begin{cases} \theta_{2,\mathrm{I}} \ (+r) \longrightarrow \theta_{1,\mathrm{I}} \\ \theta_{2,\mathrm{II}} \ (-r) \longrightarrow \theta_{1,\mathrm{II}} \end{cases} \\ \theta_{3,\mathrm{II}} \begin{cases} \theta_{2,\mathrm{III}} \ (+r) \longrightarrow \theta_{1,\mathrm{I}} \\ \theta_{2,\mathrm{IV}} \ (-r) \longrightarrow \theta_{1,\mathrm{II}} \end{cases} \end{cases}$$

$$(\theta_{1,\mathrm{I}}, \theta_{2,\mathrm{I}}, \theta_{3,\mathrm{I}})$$
$$(\theta_{1,\mathrm{I}}, \theta_{2,\mathrm{III}}, \theta_{3,\mathrm{II}})$$
$$(\theta_{1,\mathrm{II}}, \theta_{2,\mathrm{II}}, \theta_{3,\mathrm{I}})$$
$$(\theta_{1,\mathrm{II}}, \theta_{2,\mathrm{IV}}, \theta_{3,\mathrm{II}})$$



Example 1: 6R PUMA-Type Arms (cont.)

After finding $(\theta_1, \theta_2, \theta_3)$, for finding $(\theta_4, \theta_5, \theta_6)$, if the joint axes $(\boldsymbol{S}_4, \boldsymbol{S}_5, \boldsymbol{S}_6)$ of the spherical wrist, at zero configuration, are aligned in the $(\hat{z}_s, \hat{y}_s, \hat{x}_s)$ directions, respectively, we can use the ZYX Euler angles:

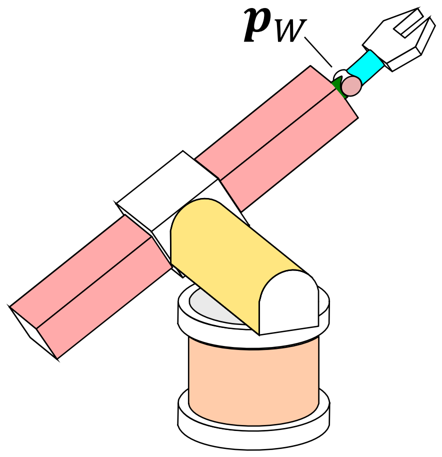
$$e^{[\boldsymbol{S}_4]\theta_4} e^{[\boldsymbol{S}_5]\theta_5} e^{[\boldsymbol{S}_6]\theta_6} = e^{-[\boldsymbol{S}_3]\theta_3} e^{-[\boldsymbol{S}_2]\theta_2} e^{-[\boldsymbol{S}_1]\theta_1} \boldsymbol{T}(\boldsymbol{\theta}) \boldsymbol{M}^{-1} = \boldsymbol{T}' = (\boldsymbol{R}', \boldsymbol{p}')$$

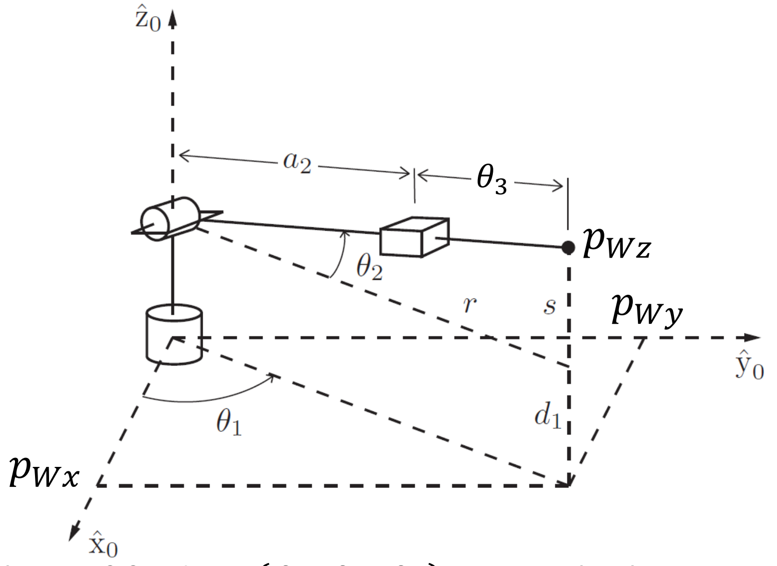
$$\boldsymbol{S}_{\omega_4} = (0,0,1)$$
$$\boldsymbol{S}_{\omega_5} = (0,1,0)$$
$$\boldsymbol{S}_{\omega_6} = (1,0,0)$$

$\Rightarrow$ $\mathrm{Rot}(\hat{\mathrm{z}}, \theta_4)\mathrm{Rot}(\hat{\mathrm{y}}, \theta_5)\mathrm{Rot}(\hat{\mathrm{x}}, \theta_6) = \boldsymbol{R}'$ $\Rightarrow$ This corresponds to the ZYX Euler angles.

$\Rightarrow (\theta_4, \theta_5, \theta_6)$

Example 2: Stanford-Type Arms (RRP)





$$p_{Wx} = (\theta_3 + a_2)\cos\theta_2\cos\theta_1$$
$$p_{Wy} = (\theta_3 + a_2)\cos\theta_2\sin\theta_1$$
$$p_{Wz} = (\theta_3 + a_2)\sin\theta_2 + d_1$$

❖ Inverse position problem of finding $(\theta_1, \theta_2, \theta_3)$ using algebraic methods:

$$p_{Wx}{}^2 + p_{Wy}{}^2 + (p_{Wz} - d_1)^2 = (\theta_3 + a_2)^2 \longrightarrow \theta_3 = +\sqrt{p_{Wx}^2 + p_{Wy}^2 + (p_{Wz} - d_1)^2} - a_2$$

$$\theta_2 = \text{atan2}\left((p_{Wz} - d_1), \pm\sqrt{p_{Wx}^2 + p_{Wy}^2}\right) \longleftarrow$$

(eliminating $(\theta_3 + a_2) > 0$ from both arguments)

If $p_{Wx}, p_{Wy} \neq 0$: $\quad \theta_1 = \text{atan2}(p_{Wy}/\cos\theta_2\,, p_{Wx}/\cos\theta_2)$

$\Rightarrow$ Thus, there are 2 solutions to the inverse kinematics problem.

If $p_{Wx}, p_{Wy} = 0$: The robot will be at the singular configuration and θ_1 will be undefined.

❖ Inverse orientation problem of finding $(\theta_4, \theta_5, \theta_6)$ is similar to PUMA.

Iterative Numerical Methods: Jacobian (Pseudo-)Inverse and Jacobian Transpose

Iterative Numerical Methods

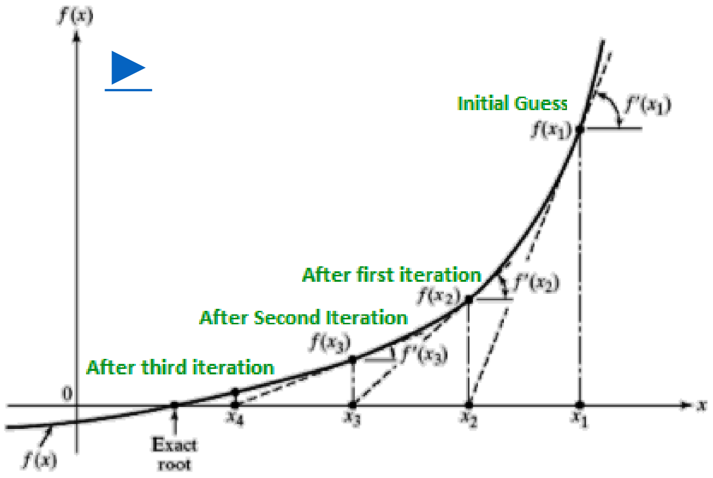
Iterative Numerical Methods: Finding a solution using iterative methods like Newton–Raphson or Gradient Descent when there are no (or difficult to find) closed-form solutions, especially for redundant manipulators or at/close to singularities.

Two main iterative numerical methods:

- **Jacobian (Pseudo-)Inverse Method** based on **Newton–Raphson Method**
- **Jacobian Transpose Method** based on **Gradient Descent Method**

Preliminary: Newton–Raphson Method

Newton–Raphson Method is an iterative method for numerically finding the roots of a nonlinear equation $f(x)=0$ where $f:\mathbb{R}\to\mathbb{R}$ is differentiable.



If x^0 is an initial guess for the solution, Taylor expansion of $f(x)$ at x^0 is

$$f(x)=f(x^0)+\left(\frac{\mathrm{d}f}{\mathrm{d}x}(x^0)\right)(x-x^0)+\overset{\cong 0}{\cancel{\text{higher–order terms}}} \xrightarrow{f(x)=0} x=x^0-\left(\frac{\mathrm{d}f}{\mathrm{d}x}(x^0)\right)^{-1}f(x^0)$$

Using x as the new guess for the solution and repeating: $$x^{k+1}=x^k-\left(\frac{\mathrm{d}f}{\mathrm{d}x}(x^k)\right)^{-1}f(x^k)$$

The iteration is repeated until some stopping criterion is satisfied, e.g., $\dfrac{|f(x^{k+1})-f(x^k)|}{|f(x^k)|}\le\epsilon$

ϵ: a given threshold value

Jacobian (Pseudo-)Inverse Method

Assume that the EE pose is represented by the minimum-coordinate representation, i.e., $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\theta}) \in \mathbb{R}^r$, $\boldsymbol{\theta} \in \mathbb{R}^n$ ($\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^r$). Thus, given a desired EE pose $\boldsymbol{x}_d$, the goal is to find joint coordinates $\boldsymbol{\theta} = \boldsymbol{\theta}_d$ such that

$$\boldsymbol{x}_d = \boldsymbol{f}(\boldsymbol{\theta})$$

(Assumption: $\boldsymbol{f}$ is differentiable)

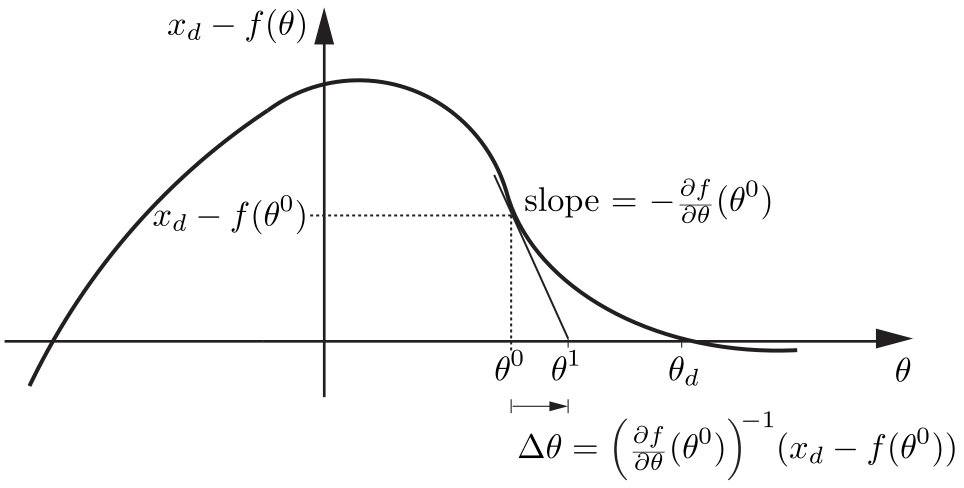
- We use a method similar to the Newton–Raphson method for nonlinear root-finding.

Given an initial guess $\boldsymbol{\theta}^0$ which is "close to" a solution $\boldsymbol{\theta}_d$, and using the Taylor expansion:

$$\boldsymbol{x}_d = \boldsymbol{f}(\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{\theta}^0) + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^0}}_{\boldsymbol{J}_a(\boldsymbol{\theta}^0) \in \mathbb{R}^{r\times n}} \underbrace{(\boldsymbol{\theta} - \boldsymbol{\theta}^0)}_{\Delta\boldsymbol{\theta}} + \overset{\cong 0}{\cancel{\text{higher–order terms}}}$$

Analytical Jacobian at $\boldsymbol{\theta}^0$

Approximately: $\longrightarrow$ $\boldsymbol{J}_a(\boldsymbol{\theta}^0)\Delta\boldsymbol{\theta} = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^0)$


$x_d - f(\theta)$
$x_d - f(\theta^0)$
slope $= -\frac{\partial f}{\partial \theta}(\theta^0)$
θ^0
θ^1
θ_d
θ
$\Delta\theta = \left(\frac{\partial f}{\partial \theta}(\theta^0)\right)^{-1}(x_d - f(\theta^0))$

Jacobian (Pseudo-)Inverse Method

$*$ If $\boldsymbol{J}_a$ is square ($r = n$) and invertible: $\Delta\boldsymbol{\theta} = \boldsymbol{J}_a^{-1}(\boldsymbol{\theta}^0)\big(\boldsymbol{x}_d - \boldsymbol{f}(\theta^0)\big)$

$$\Rightarrow \quad \boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \lambda\,\boldsymbol{J}_a^{-1}(\boldsymbol{\theta}^k)\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)\right), \qquad k = 0,1,2,\ldots \qquad \overset{\overset{\text{initial guess}}{\uparrow}}{\boldsymbol{\theta}^0},\boldsymbol{\theta}^1,\boldsymbol{\theta}^2,\ldots \to \boldsymbol{\theta}_d$$

$*$ If $\boldsymbol{J}_a$ is not square or invertible (due to redundancy or singularity): $\Delta\boldsymbol{\theta} = \boldsymbol{J}_a^{+}(\boldsymbol{\theta}^0)\big(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^0)\big)$

$$\Rightarrow \quad \boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \lambda\,\boldsymbol{J}_a^{+}\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)\right), \qquad k = 0,1,2,\ldots$$

$\boldsymbol{J}_a^{+}$: Moore–Penrose pseudoinverse

To improve stability and convergence (especially when the initial guess is far from the solution or $\lambda = 1$ would take you too far), the **step size parameter** $\lambda \in (0,1]$ is added. It can be either constant or adaptively chosen $\lambda = \lambda^k$. This parameter is also called a **damping factor** or **relaxation parameter**.

Note: If robot is redundant ($n > r$) and $\boldsymbol{J}_a$ is full rank ($\mathrm{rank}(\boldsymbol{J}_a) = \min(r, n)$), i.e., the robot is not at a singularity:

$$\boldsymbol{J}_a^{+} = \boldsymbol{J}_a^{T}(\boldsymbol{J}_a\boldsymbol{J}_a^{T})^{-1}$$

Algorithm for Minimum-Coordinate Representation

a) **Initialization**: Given $\boldsymbol{x}_d \in \mathbb{R}^r$ and an initial guess $\boldsymbol{\theta}^0 \in \mathbb{R}^n$, set $k = 0$.

b) **Iteration**: Set $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)$. While $e_p > \epsilon_p$ or $e_R > \epsilon_R$ for some small $\epsilon_p, \epsilon_R \in \mathbb{R}$, where e_p is the position error and e_R is the orientation error between two configurations $\boldsymbol{x}_d$ and $\boldsymbol{f}(\boldsymbol{\theta}^k)$:
- Set $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \lambda \boldsymbol{J}^+(\boldsymbol{\theta}^k)\boldsymbol{e}$. $0 < \lambda \le 1$: step size parameter
- Increament k by $k = k + 1$.

Example of the Algorithm in Python:

```python
max_iterations = 20
k = 0
lmbda = 1
Theta = Theta_0
e = X_d - FK(Theta)

while np.linalg.norm(e) > epsilon and k < max_iterations:
    Theta = Theta + lmbda * np.linalg.pinv(J(Theta)) @ e
    k += 1
    e = X_d - FK(Theta)
```

Position and Orientation Error for Minimum-Coordinate Representation

To find the error (distance) between two configurations in $SE(3)$, we need to define metrics in $\mathbb{R}^3$ for position and $SO(3)$ for orientation.

- For position error e_p between $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, the Euclidean distance is used: $e_p = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|_2$
- Computation of e_R depends on the representation of end-effector orientation, namely, unit quaternion, exponential coordinates (angle and axis), and Euler angles:

(1) Unit Quaternion $\boldsymbol{q}_1, \boldsymbol{q}_2$:

Method 1: Geodesic Distance

$$e_R = \theta = 2\arccos(|\boldsymbol{q}_1 \cdot \boldsymbol{q}_2|) \in [0, \pi]$$

or $$\boldsymbol{q}_{\text{rel}} = \boldsymbol{q}_1^{-1} \cdot \boldsymbol{q}_2 = (q_0, \boldsymbol{q}_v) \rightarrow \quad e_R = \theta = 2\arccos(|q_0|)$$

Method 2: Chordal Distance (Euclidean Distance in Quaternion Space)

$$e_R = \min(\|\boldsymbol{q}_1 - \boldsymbol{q}_2\|_2, \|\boldsymbol{q}_1 + \boldsymbol{q}_2\|_2) \in [0, \sqrt{2}]$$

Position and Orientation Error for Minimum-Coordinate Representation

(2) Exponential Coordinates $\boldsymbol{r}_1, \boldsymbol{r}_2$: $e_R = \|\boldsymbol{r}_1 - \boldsymbol{r}_2\|_2$

(3) Euler angles $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$: $e_R = \|\boldsymbol{\phi}_1 - \boldsymbol{\phi}_2\|_2$

(acceptable <u>only for small distances</u>, otherwise not recommended due to singularities and lack of reflecting the true rotation difference!)

For exponential coordinates and Euler angles representations, it is preferred to first compute the corresponding rotation matrices $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$, then compute $\boldsymbol{R}_{\mathrm{rel}} = \boldsymbol{R}_1^T \boldsymbol{R}_2$ and

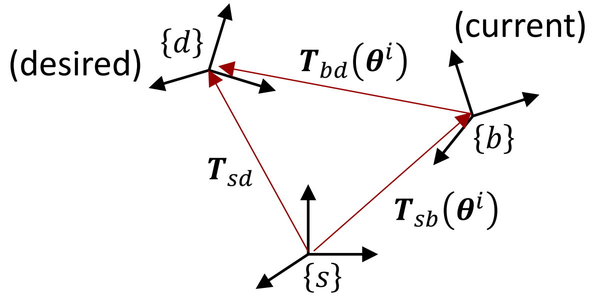
$$e_R = \theta = \arccos\left(\frac{\mathrm{trace}(\boldsymbol{R}_{\mathrm{rel}}) - 1}{2}\right)$$

or $$e_R = \theta = \|\log(\boldsymbol{R}_{\mathrm{rel}})^{\vee}\|_2$$

Algorithm for Transformation Matrix Representation

Assume that the EE pose is represented by a Transformation Matrix, i.e., $\boldsymbol{T}_{sb} = \boldsymbol{T}(\boldsymbol{\theta}) \in SE(3)$, $\boldsymbol{\theta} \in \mathbb{R}^n$. Thus, given a desired EE pose $\boldsymbol{T}_{sd}$, the goal is to find joint coordinates $\boldsymbol{\theta} = \boldsymbol{\theta}_d$ such that

$$\boldsymbol{T}_{sd} = \boldsymbol{T}(\boldsymbol{\theta}_d)$$


(desired)
$\{d\}$
$\boldsymbol{T}_{bd}(\boldsymbol{\theta}^i)$
(current)
$\{b\}$
$\boldsymbol{T}_{sd}$
$\boldsymbol{T}_{sb}(\boldsymbol{\theta}^i)$
$\{s\}$

Algorithm in Body Frame:

a) Initialization: Given $\boldsymbol{T}_{sd} \in SE(3)$ and an initial guess $\boldsymbol{\theta}^0 \in \mathbb{R}^n$, set $k = 0$.

b) Iteration: Set $\boldsymbol{\mathcal{E}}_b = (\boldsymbol{\mathcal{E}}_{b,\omega}, \boldsymbol{\mathcal{E}}_{b,v}) = \log\left(\boldsymbol{T}_{bd}(\boldsymbol{\theta}^k)\right)^\vee = \log(\boldsymbol{T}_{sb}^{-1}(\boldsymbol{\theta}^k)\boldsymbol{T}_{sd})^\vee$. While $\|\boldsymbol{\mathcal{E}}_{b,\omega}\| > \epsilon_\omega$ or $\|\boldsymbol{\mathcal{E}}_{b,v}\| > \epsilon_v$ for some small $\epsilon_\omega, \epsilon_v \in \mathbb{R}$:

- Set $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k + \lambda \boldsymbol{J}_b^+(\boldsymbol{\theta}^k)\boldsymbol{\mathcal{E}}_b$. $(0 < \lambda \leq 1)$
- Set $k = k + 1$.

Note: ϵ_ω has the unit of radian and the dimension of ϵ_v is length.

Note: $\boldsymbol{\mathcal{E}}_b$ is the twist that takes $\boldsymbol{T}_{sb}$ to $\boldsymbol{T}_{sd}$ in 1s.

Jacobian Transpose Method

Let's define the inverse kinematic problem as an optimization problem as

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{2}\|\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})\|_2^2 = \min_{\boldsymbol{\theta}} \frac{1}{2}\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})\right)^T\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})\right)$$

The gradient of the cost function $F(\boldsymbol{\theta}) \in \mathbb{R}$ is

$$\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \left(\frac{\partial F(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T = -\left(\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})\right)^T \frac{\partial \boldsymbol{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T = -\boldsymbol{J}_a^T(\boldsymbol{\theta})(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})).$$

A **Gradient Descent** algorithm to minimize $F(\boldsymbol{\theta})$ is

$$\underset{\uparrow\,\text{initial guess}}{\boldsymbol{\theta}^0}, \boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots \to \boldsymbol{\theta}_d$$

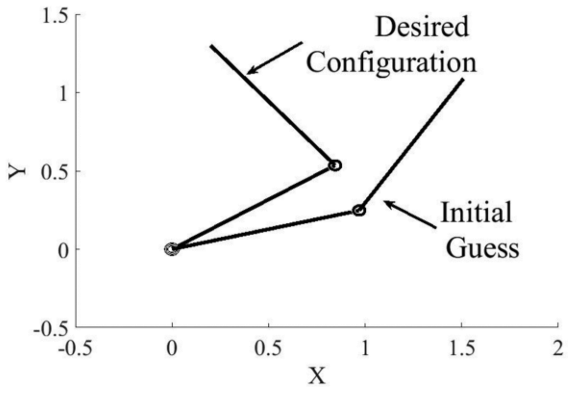
$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \lambda \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^k) = \boldsymbol{\theta}^k + \lambda \boldsymbol{J}_a^T(\boldsymbol{\theta}^k)\left(\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)\right), \qquad k = 0,1,2,\ldots$$

where $\lambda > 0$ (often $\lambda \in (0,1]$ for stability) is the **step size** (or **learning rate**) and should be chosen to guarantee a decrease of the error function at each iteration; too large λ may lead the method to miss the minimum, and too small λ may lead to extremely slow convergence.

- The step size can be constant or adaptively chosen at each iteration as λ^k (e.g., using Armijo's rule).

Case Study

Consider the following 2R robot where the desired end-effector coordinate is $\boldsymbol{x}_d = (0.2, 1.3)$, the joint variables corresponding to $\boldsymbol{x}_d$ are $\theta_1 = 0.5650$ and $\theta_2 = 0.7062$, the initial guess are $\theta_1 = 0.25$ and $\theta_2 = 0.75$, and the step size is 0.75.



Iteration	θ_1	θ_2
1	-0.33284	2.6711
2	0.80552	2.1025
3	0.46906	1.9316
4	0.53554	1.7697
5	0.55729	1.7227
6	0.56308	1.7104
7	0.56455	1.7073
8	0.56492	1.7065
9	0.56501	1.7063
10	0.56503	1.7062

IK using Jacobian inverse

Iteration	θ_1	θ_2
1	1.8362	1.3412
2	0.4667	1.1025
3	1.1215	1.6233
4	0.45264	1.415
5	0.83519	1.7273
26	0.56522	1.7063
27	0.56492	1.7061
28	0.56514	1.7063
29	0.56498	1.7062
30	0.5650	1.7062

IK using Jacobian transpose

Remarks on Jacobian (Pseudo-)Inverse and Transpose Methods

- If there are **multiple inverse kinematics solutions**, the iterative process tends to converge to the solution that is "closest" to the initial guess $\boldsymbol{\theta}^0$. Multiple initializations is required for obtaining other solutions.

- The **stopping criteria** for the iteration can be a small threshold on Cartesian error (possibly, separate for position and orientation) or joint angle increment.

$$\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)$$
$$e_p < \epsilon_{\text{pos.}}\ ,\quad e_{\text{ori.}} < \epsilon_R$$

or

$$\left\|\boldsymbol{\theta}^{k+1} - \boldsymbol{\theta}^k\right\| < \epsilon_q$$

- Joint range limits are considered only at the end by checking if the solution found is feasible, as for analytical methods.

- For the motion of a robot along **a given desired end-effector trajectory**, a good choice for the initial guess $\boldsymbol{\theta}^0$ is to use the solution to the IK at the **previous time step**. This provides continuity and requires only a few iterations to converge.

Remarks on Jacobian (Pseudo-)Inverse and Transpose Methods

- Jacobian Transpose method is **computationally more efficient** to compute than the Jacobian (Pseudo-)Inverse method (for both redundant and non-redundant robots). However, The convergence of Jacobian transpose, in terms of number of iterations, is usually **slower** than the Jacobian inverse method.

- **Lack of convergence** in Jacobian (Pseudo-)Inverse method occurs when $\boldsymbol{J}_a(\boldsymbol{\theta})$ is not full rank (i.e., the robot is in a singular configuration).

- **Lack of convergence** in Jacobian Transpose method occurs when $\boldsymbol{J}_a(\boldsymbol{\theta})$ is not full rank (i.e., the robot is in a singular configuration), **and** at the same time, the error $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta}^k)$ is in the null space of $\boldsymbol{J}_a^T$, i.e., $\mathcal{N}\left(\boldsymbol{J}_a^T(\boldsymbol{\theta})\right)$ (which, in practice, rarely happens!). Therefore, Jacobian transpose does not generally suffer from kinematic singularities.

Optimization-based Inverse Kinematics

The optimization-based inverse kinematics method can be used in situations where an infinite solutions exist (i.e., for redundant robots) and we seek a solution that is optimal with respect to some constraints.

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \phi(\boldsymbol{\theta}) \\ \text{subject to} \quad & \boldsymbol{x}_d = \boldsymbol{f}(\boldsymbol{\theta}) \end{aligned} \qquad \text{or} \qquad \min_{\boldsymbol{\theta}} \quad \|\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{\theta})\|_2^2 + \lambda\phi(\boldsymbol{\theta})$$

$\phi(\boldsymbol{\theta})$ is a cost function to resolve redundancy, e.g.:

- Joint limit avoidance, —— $\phi(\boldsymbol{\theta}) = \sum_{i=1}^{n}\left(\frac{\theta_i - \theta_{i,\mathrm{mid}}}{\theta_{i,\max} - \theta_{i,\min}}\right)^2$
- Staying close to a nominal posture $\boldsymbol{\theta}_0$, —— $\phi(\boldsymbol{\theta}) = \|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|_2^2$
- Distance from obstacles,…

❖ Since the constraint $\boldsymbol{x}_d = \boldsymbol{f}(\boldsymbol{\theta})$ is nonlinear and often non-convex, this optimization is a general nonlinear optimization, which is, in general, slow to solve (not real-time friendly), sensitive to initial guess (poor trajectory consistency with jumping across different solutions), has multiple local minima, and challenging in handling singularities.

Note: Redundancy resolution is much more naturally and efficiently handled in **Inverse Velocity Kinematics** (IVK) via null-space projection or optimization.