

Lecture 4: Inverse Kinematics

Iñigo Iturrate

Assistant Professor

SDU Robotics,

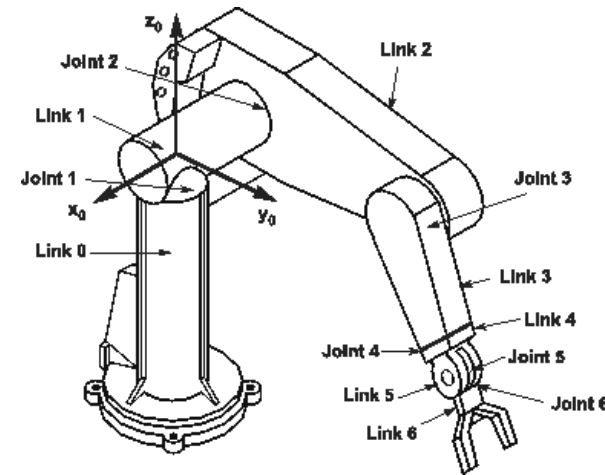
The Maersk McKinney Moller Institute,
University of Southern Denmark



[Ø27-604-3](tel:4566276043)



inju@mmmi.sdu.dk

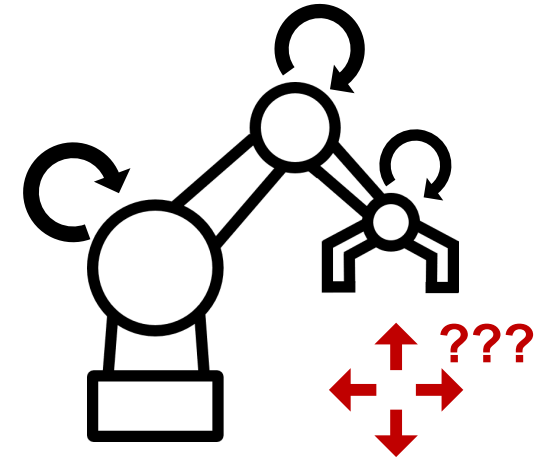


Previously you have seen/recapped...

We can **describe** a robot's **kinematic structure** using DH parameters.

Forward Kinematics describes how **motions of the joints** translate into **motion of the end-effector**:

- We can obtain the **Forward Kinematics** from the DH parameters.
- We can also derive the **forward kinematics** from transformations between frames.



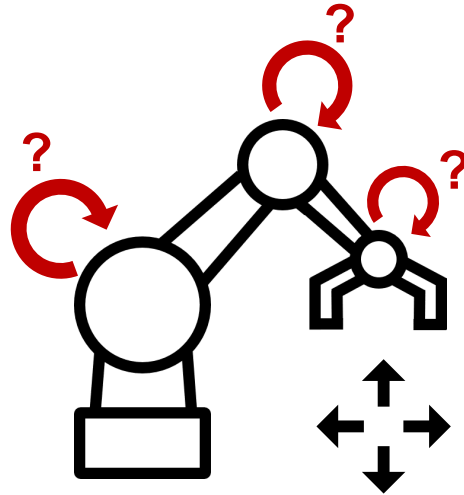
Today...

What if we want to do **the opposite**?

In most tasks, we will have a **defined task motion** that the robot needs to follow in Cartesian space, **not a defined joint space motion**.

The problem then becomes:

“Given a **target position and orientation** of the robot **end-effector**, what **joint position** values will **reach that target**?”



Topics for Today

Part I: Joint and Cartesian Spaces

Part II: Analytical Inverse Kinematics

Part III: Practical Considerations

Part IV: Numerical Inverse Kinematics

Part I: Joint and Cartesian Spaces & Mapping between Them

Joint & Cartesian Spaces

In robotics, we are constantly working with **two different spaces**:

- **Joint (or Operational) space**
 - The space where the robot actually operates.
 - Defines **how its joints** (and their associated **motors**) **actually move**.
 - It is the space **we have direct (low-level) control over**.
- **Cartesian (or Task) space**
 - “Our world”, where we want the robot to actually act and perform a task.
 - Defines **how the end-effector/tool** of the robot **should move**.
 - We **cannot control it directly** (at a low level).

Joint (Configuration) Space

The **dimensionality** of the space **will depend on the number and kind of joints (i.e. DOF)** in the robot.

For an n -DOF robot:

- **Positions** in joint space are given by $\mathbf{q} \in \mathbb{R}^n$:

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}$$

- **Velocities** are given by $\dot{\mathbf{q}} \in \mathbb{R}^n$:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix}$$

- **Accelerations** are given by $\ddot{\mathbf{q}} \in \mathbb{R}^n$:

$$\ddot{\mathbf{q}} = \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_n \end{bmatrix}$$

Where each q_i is:

- An angle θ for revolute joints
- A displacement d for prismatic joints

Note: Since we often work with robots with only revolute joints, joint positions are often denoted:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Cartesian (Task) Space

The **dimensionality** of the space **will depend on** the kind of task. Usually, we will work in a 3D environment.

For a 3D environment:

- **Poses** in Cartesian space are usually described by $\mathbf{x} \in \mathbb{R}^6$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\phi} \end{bmatrix}$$

Where:

- $\mathbf{p} \in \mathbb{R}^3$ is a position vector $[p_x, p_y, p_z]^T$
- $\boldsymbol{\phi} \in \mathbb{R}^3$ or $\boldsymbol{\phi} \in SO(3)$ is an orientation vector $[\phi_x, \phi_y, \phi_z]^T$
(which can be in many representations)

- **Velocities** are usually given by $\dot{\mathbf{x}} \in \mathbb{R}^6$:

$$\dot{\mathbf{x}} = \mathbf{v} = \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} \text{ or } \dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ \boldsymbol{\omega} \end{bmatrix}$$

Where:

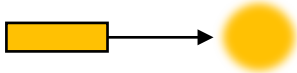
- $\dot{\boldsymbol{\phi}} \in \mathbb{R}^3$ is the time-derivative of the orientation vector
- $\boldsymbol{\omega} \in \mathbb{R}^3$ is an angular velocity

- **Accelerations** are usually given by $\ddot{\mathbf{x}} \in \mathbb{R}^6$:

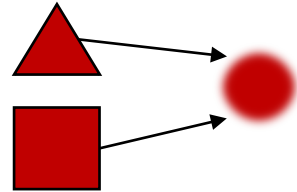
$$\ddot{\mathbf{x}} = \mathbf{a} = \begin{bmatrix} \ddot{\mathbf{p}} \\ \ddot{\boldsymbol{\phi}} \end{bmatrix} \text{ or } \ddot{\mathbf{x}} = \begin{bmatrix} \ddot{\mathbf{p}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix}$$

Different Cases with Mappings

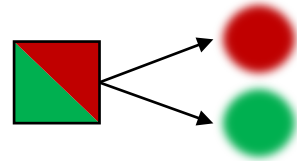
One-to-one:



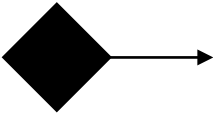
Many-to-one:



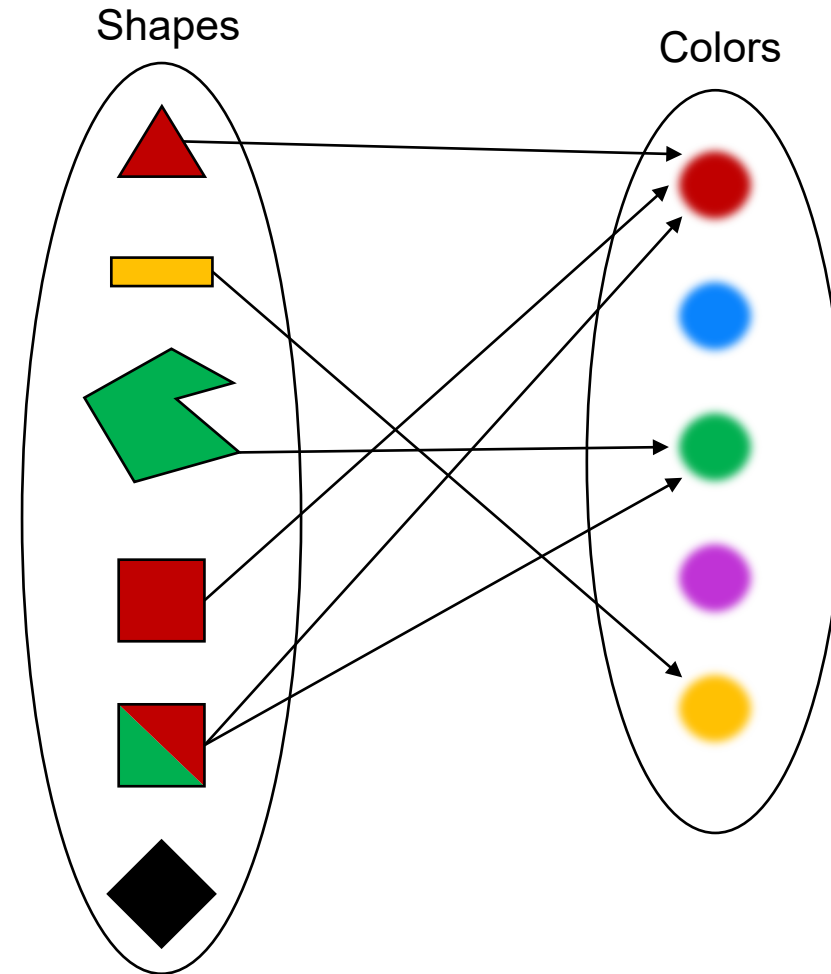
One-to-many:



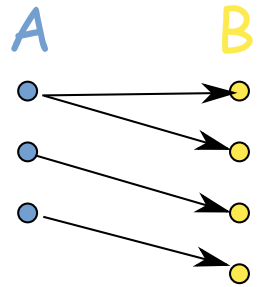
One-to-none:



None-to-one:

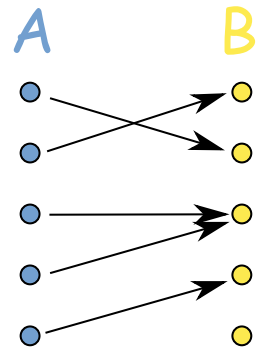


Different Cases with Mappings (More Formally)



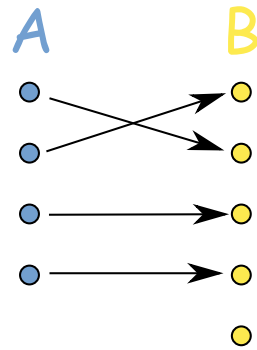
NOT a
Function

A has many B



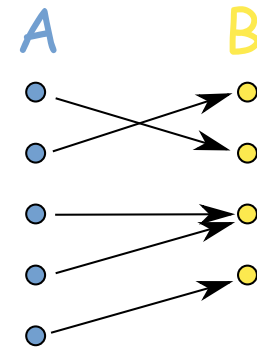
General
Function

B can have many A



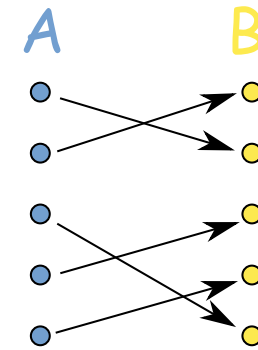
Injective
(not surjective)

B can't have many A



Surjective
(not injective)

Every B has some A



Bijjective
(injective, surjective)

A to B, perfectly

Source: <https://www.mathsisfun.com/sets/injective-surjective-bijjective.html>

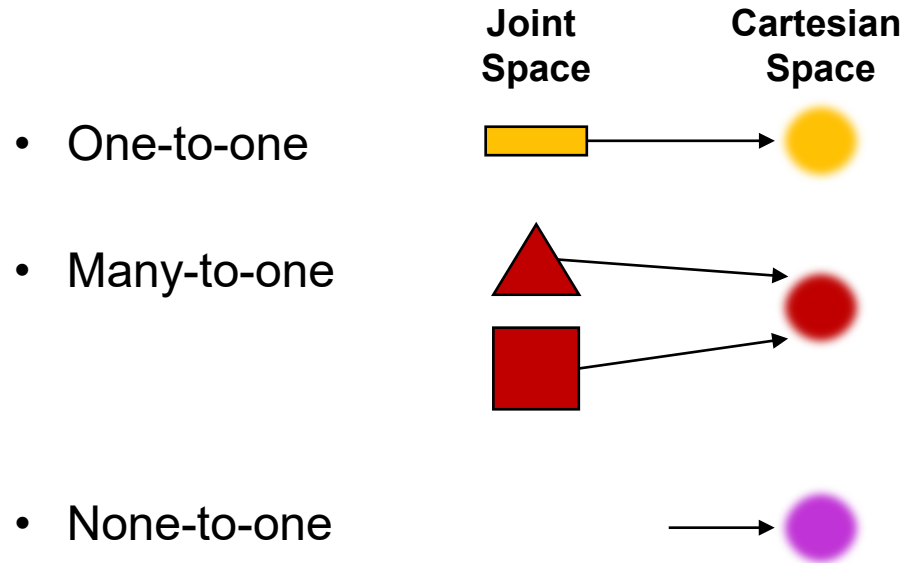
Forward Kinematics (Joint \rightarrow Cartesian)

Forward Kinematics is the name given to the mapping from joint space to Cartesian space.

In other words, **if we know the position of the joints, what is the position of the end-effector?**

This is the easy problem.

Why? Because the mappings will be:



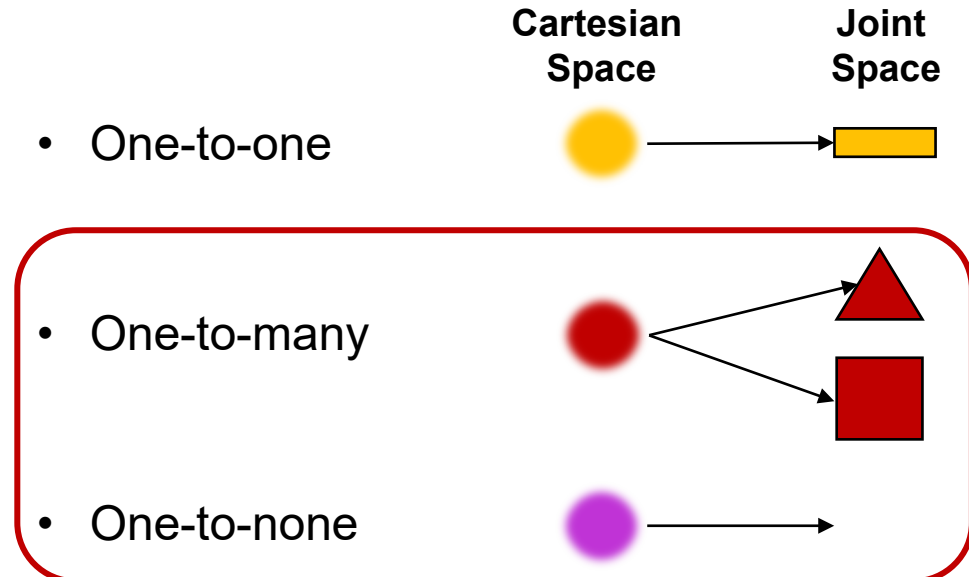
Inverse Kinematics (Cartesian \rightarrow Joint)

Inverse Kinematics is the name given to the mapping from Cartesian space to joint space.

In other words, **if we know the position of the end-effector, how should we place the joints?**

This is the HARD problem.

Why? Because the mappings will be:



And this can be nasty!

Part II: Analytical Inverse Kinematics

Inverse Kinematics as a Mathematical Problem

The forward kinematics of a robot are specified by a transformation matrix: ${}^{Base}_{Tool}\mathbf{T}(\boldsymbol{\theta}) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Let's take a Universal Robots 6R manipulator as an example.

We can write a **system of 12 equations with 6 unknowns**¹:

$$\begin{bmatrix} r_{11} \\ r_{12} \\ \vdots \\ p_z \end{bmatrix} = \begin{bmatrix} \cos(\theta_6)(\sin(\theta_1)\sin(\theta_5) + \cos(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1)\cos(\theta_5)) - \sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1)\sin(\theta_6) \\ -\sin(\theta_6) * (\sin(\theta_1) * \sin(\theta_5) + \cos(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1) * \cos(\theta_5)) - \sin(\theta_2 + \theta_3 + \theta_4)\cos(\theta_1)\cos(\theta_6) \\ \vdots \\ d_1 + d_5(\sin(\theta_2 + \theta_3)\sin(\theta_4) - \cos(\theta_2 + \theta_3)\cos(\theta_4)) + a_3\sin(\theta_2 + \theta_3) + a_2\sin(\theta_2) - d_6\sin(\theta_5)(\cos(\theta_2 + \theta_3)\sin(\theta_4) + \sin(\theta_2 + \theta_3)\cos(\theta_4)) \end{bmatrix}$$

Solving this for $[\theta_1, \theta_2, \dots, \theta_6]$ will give us the Inverse Kinematics.

Only 3 of the 9 equations for the rotation are independent → We end up with 6 equations/6 unknowns.

Closed-form vs. Numerical Solutions

Note that the equations **IK** are **non-linear and transcendental**!

There are **two main approaches** to solving the **inverse kinematics** problem:

- **Closed-form**: An analytical solution based on the forward kinematics transform equations.
 - + Fast to compute
 - + Exact
 - Does not exist for all robots
 - Can be complex to calculate
 - Needs to be calculated for each specific robot
- **Numerical**: A numerical solution based on an approximation and iterative attempts.
 - + Possible for all robots
 - + The same method can be used generally for any robot
 - Slow to compute
 - Inexact

Exercise: IK of 2R Planar Manipulator

Objective: Obtain (q_1, q_2)

Given:

- End-effector position: (x, y)
- Length of links: L_1 and L_2

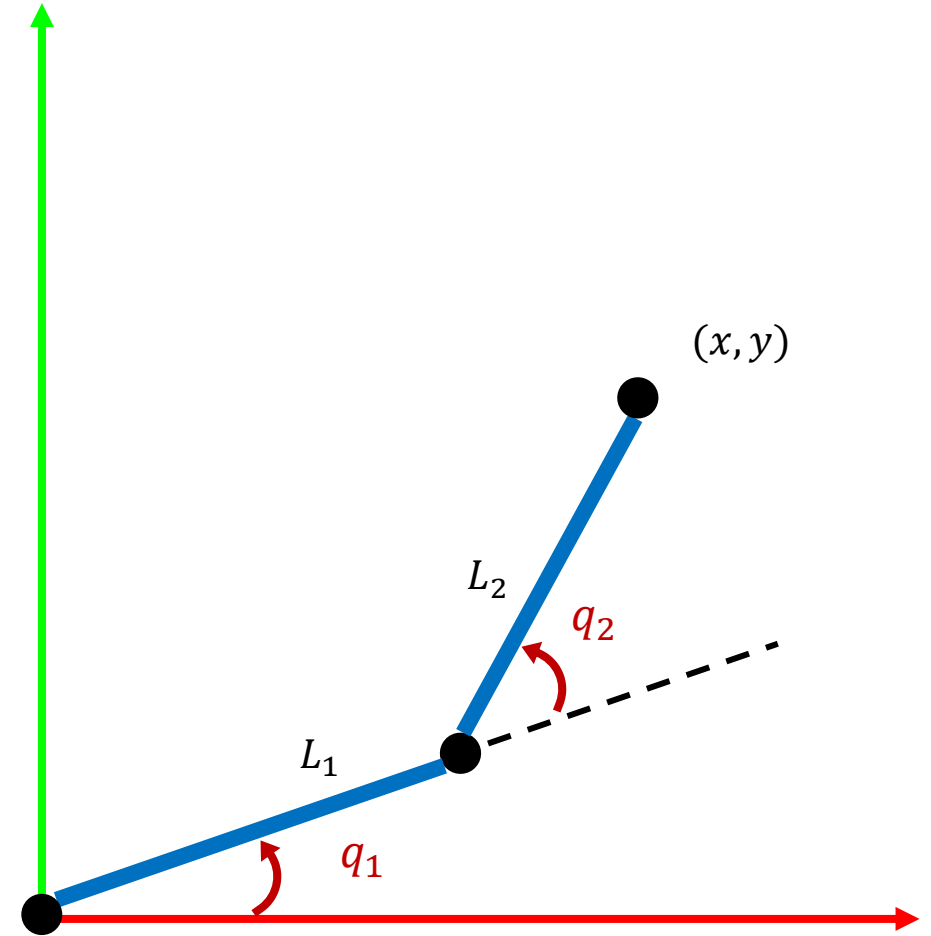
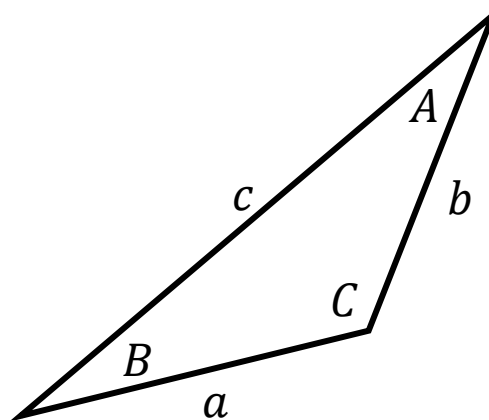
Exercise: 10-15 minutes

Hint: Use the rule of cosines:

$$a^2 = b^2 + c^2 - 2bc \cos(A)$$

$$b^2 = a^2 + c^2 - 2ac \cos(B)$$

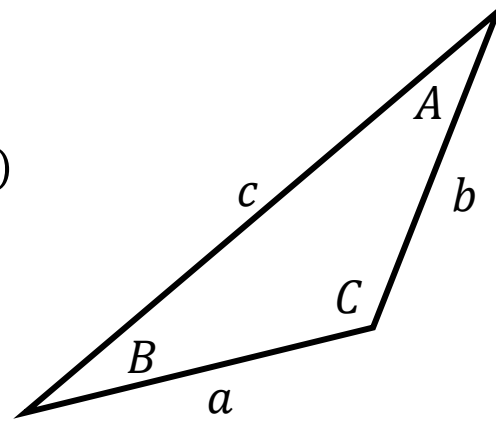
$$c^2 = a^2 + b^2 - 2ab \cos(C)$$



IK of 2R Planar Manipulator: Solution for q_2

$$c^2 = a^2 + b^2 - 2ab \cos(C)$$

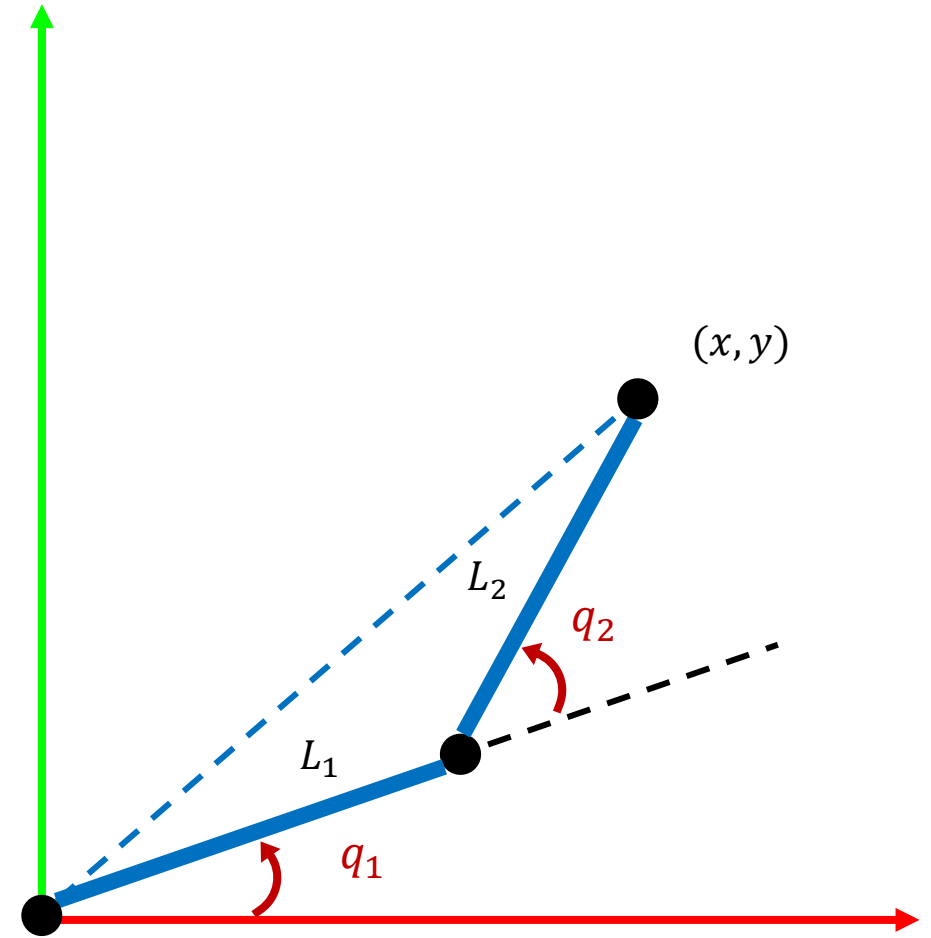
$$\cos(C) = \frac{a^2 + b^2 - c^2}{2ab}$$



$$\cos(\pi - q_2) = \frac{L_1^2 + L_2^2 - (\sqrt{x^2 + y^2})^2}{2L_1L_2}$$

$$\cos(q_2) = -\frac{L_1^2 + L_2^2 - x^2 - y^2}{2L_1L_2}$$

$$q_2 = \pm \arccos\left(\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$$



IK of 2R Planar Manipulator: Solution for q_1

$$q_1 = \varphi - B$$

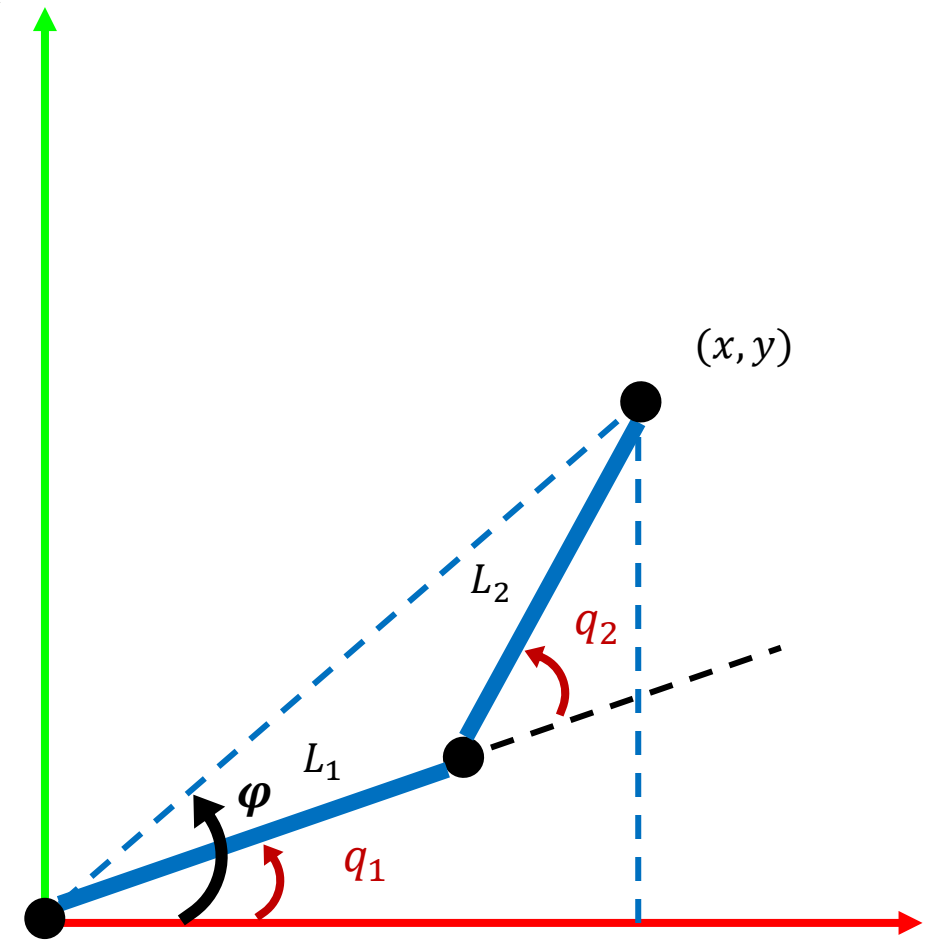
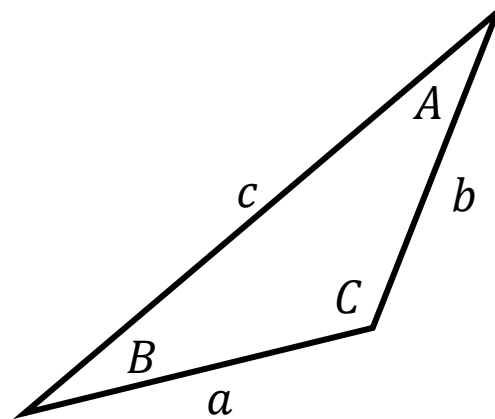
$$\varphi = \text{atan2}(y, x)$$

Using law of cosines:

$$b^2 = a^2 + c^2 - 2ac \cos(B)$$

$$B = \pm \arccos\left(\frac{L_1^2 + x^2 + y^2 - L_2^2}{2L_1\sqrt{x^2 + y^2}}\right)$$

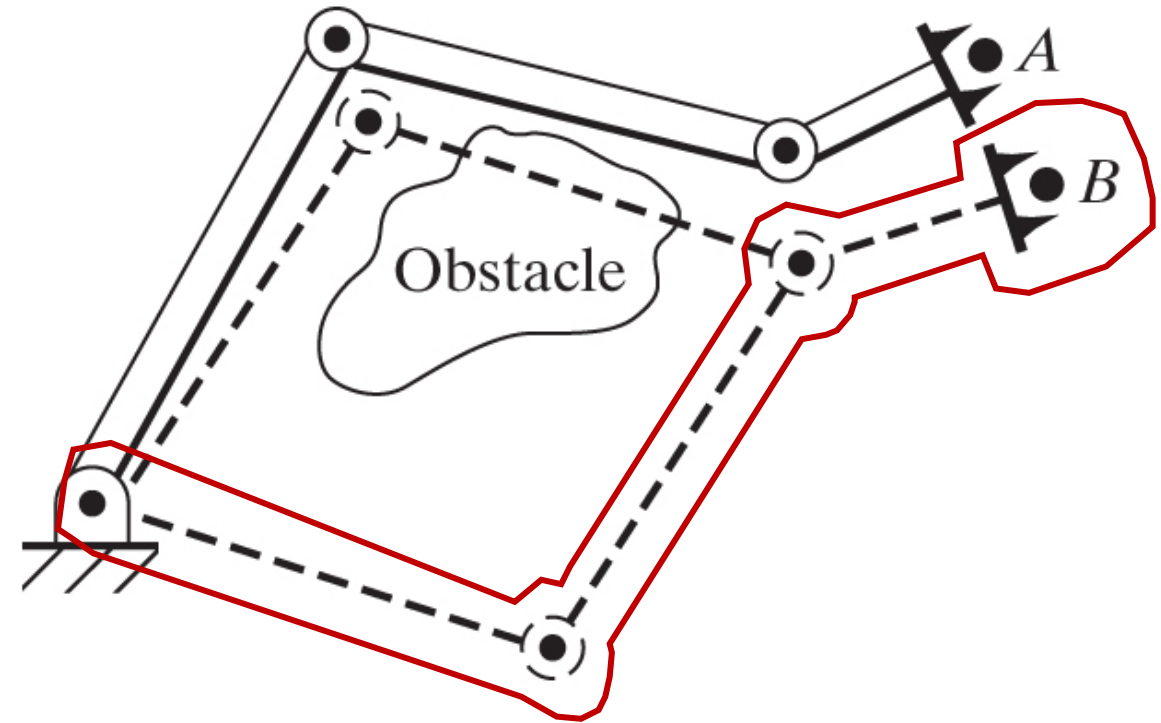
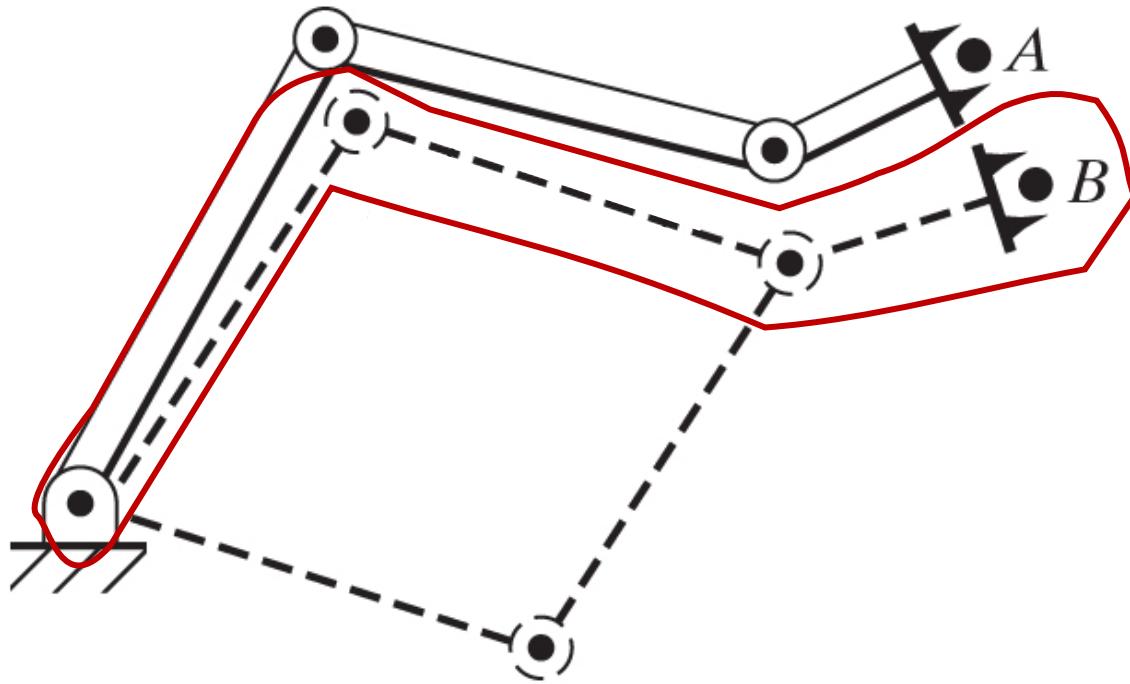
$$q_1 = \text{atan2}(y, x) \mp \arccos\left(\frac{L_1^2 + x^2 + y^2 - L_2^2}{2L_1\sqrt{x^2 + y^2}}\right)$$



Choice of IK Solutions

Notice in the previous slides that **there are multiple solutions** for the **same target pose**.

Different tasks will dictate a **different choice of IK solution**:



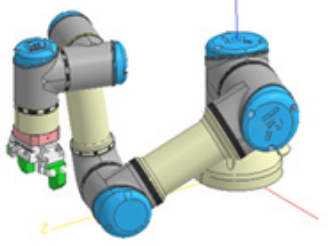
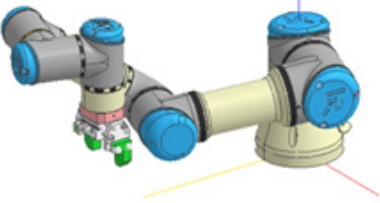
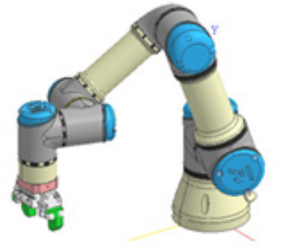
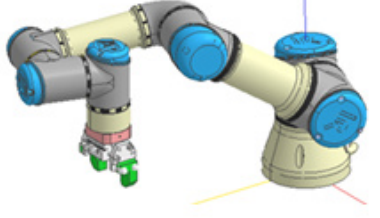
DOF and Number of IK Solutions

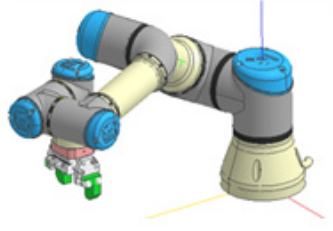
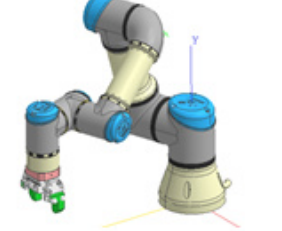
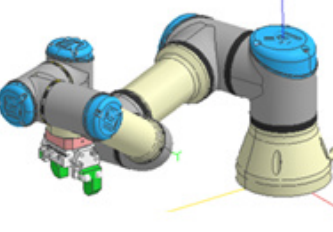
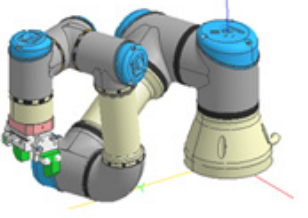
The number of **DOF** of the robot will greatly **affect the number of IK solutions**.

For a 3D Cartesian space with 3-DOF position + 3-DOF orientation and an n -DOF robot:

- $n < 6$: Will often run into the problem of not having a solution
- $n = 6$: Will (in theory) be the minimum to cover the entire space:
 - Some points will have **multiple solutions** (up to 8)
 - Some points will have a **single solution**
 - Some points will have **no solutions** (singularities)
- $n > 7$: Will be redundant (always multiple solutions).

Example – Multiple IK Solutions: UR (6-DOF)

Pattern	Shoulder	Elbow	Wrist	Figure1 (Tool Down)
1	Left Side	Down	Tool Down: Outer	
2	Left Side	Down	Tool Down: Inner	
3	Left Side	UP	Tool Down: Outer	
4	Left Side	UP	Tool Down: Inner	

Pattern	Shoulder	Elbow	Wrist	Figure1 (Tool Down)
5	Right Side	UP	Tool Down: Inner	
6	Right Side	UP	Tool Down: Outer	
7	Right Side	Down	Tool Down: Inner	
8	Right Side	Down	Tool Down: Outer	

Example: Multiple IK Solutions (7-DOF)



Part III: The Jacobian Matrix & Differential Kinematics

What do we want?

The **Forward Kinematics** allow us to **relate joint positions to end-effector poses**.

What if we want to **relate joint velocities to end-effector velocities**?

The diagram illustrates the Jacobian matrix equation: $\dot{x} = J \dot{q}$. The end-effector velocity vector \dot{x} is circled in green, with a green arrow pointing to it from the text "Get end-effector velocity here". The Jacobian matrix J is a 3x3 matrix with elements represented by question marks, enclosed in a blue rounded rectangle, with a blue arrow pointing to it from the text "Do some sorcery here". The joint velocity vector \dot{q} is circled in red, with a red arrow pointing to it from the text "Insert joint velocity vector here". Below the equation, a blue italicized text states: "(This sorcery is called the Jacobian Matrix)".

$$\dot{x} = \begin{bmatrix} ? & \dots & ? \\ \vdots & \ddots & \vdots \\ ? & \dots & ? \end{bmatrix} \dot{q}$$

Get end-effector velocity here

Do some sorcery here

Insert joint velocity vector here

(This sorcery is called the Jacobian Matrix)

The Jacobian Matrix

The **Jacobian matrix** is a multidimensional form of the derivative. It is a matrix of partial derivatives.

If we have, e.g. 6 functions of 6 variables each:

$$\left. \begin{aligned} y_1 &= f_1(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_2 &= f_2(x_1, x_2, x_3, x_4, x_5, x_6) \\ &\vdots \\ y_6 &= f_6(x_1, x_2, x_3, x_4, x_5, x_6) \end{aligned} \right\} \text{ Which we can re-write: } \mathbf{y} = \mathbf{F}(\mathbf{x})$$

And we want to **calculate the differentials** of each function based on the input:

$$\left. \begin{aligned} \delta y_1 &= \frac{\partial f_1}{\partial x_1} \delta x_1 + \frac{\partial f_1}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_1}{\partial x_6} \delta x_6 \\ \delta y_2 &= \frac{\partial f_2}{\partial x_1} \delta x_1 + \frac{\partial f_2}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_2}{\partial x_6} \delta x_6 \\ &\vdots \\ \delta y_6 &= \frac{\partial f_6}{\partial x_1} \delta x_1 + \frac{\partial f_6}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_6}{\partial x_6} \delta x_6 \end{aligned} \right\} \text{ Which we can re-write: } \delta \mathbf{y} = \mathbf{J}(\mathbf{x}) \delta \mathbf{x}$$

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_1}{\partial x_6}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_2}{\partial x_6}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial x_1}(\mathbf{x}) & \frac{\partial f_6}{\partial x_2}(\mathbf{x}) & \dots & \frac{\partial f_6}{\partial x_6}(\mathbf{x}) \end{bmatrix}$$

The Forward Velocity Kinematics Equation

Now let us put this in the **context of our robotics problem**:

From the **definition** of Jacobian:

$$\frac{\partial x}{\partial q} = J(q)$$

By applying the **chain rule**:

$$\frac{\partial x}{\partial q} \frac{dq}{dt} = J(q) \frac{dq}{dt}$$

Re-arranging:

$$\frac{dx}{dt} = J(q) \frac{dq}{dt}$$

$$\dot{x} = J(q)\dot{q}$$



Exercise: Jacobian for a 2R Planar manipulator

Remember: the forward kinematics of a 2R manipulator are given by:

$$\begin{aligned}x &= L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \\y &= L_1 \sin(q_1) + L_2 \sin(q_1 + q_2)\end{aligned}$$

Objective: Calculate $J(q_1, q_2)$

Exercise: 5 minutes

Hint: By definition, the Jacobian is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

Solution: Jacobian for a 2R Planar manipulator

Remember: the forward kinematics of a 2R manipulator are given by:

$$\begin{aligned}x &= L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \\y &= L_1 \sin(q_1) + L_2 \sin(q_1 + q_2)\end{aligned}$$

Objective: Calculate $J(q_1, q_2)$

Exercise: 5 minutes

Solution:

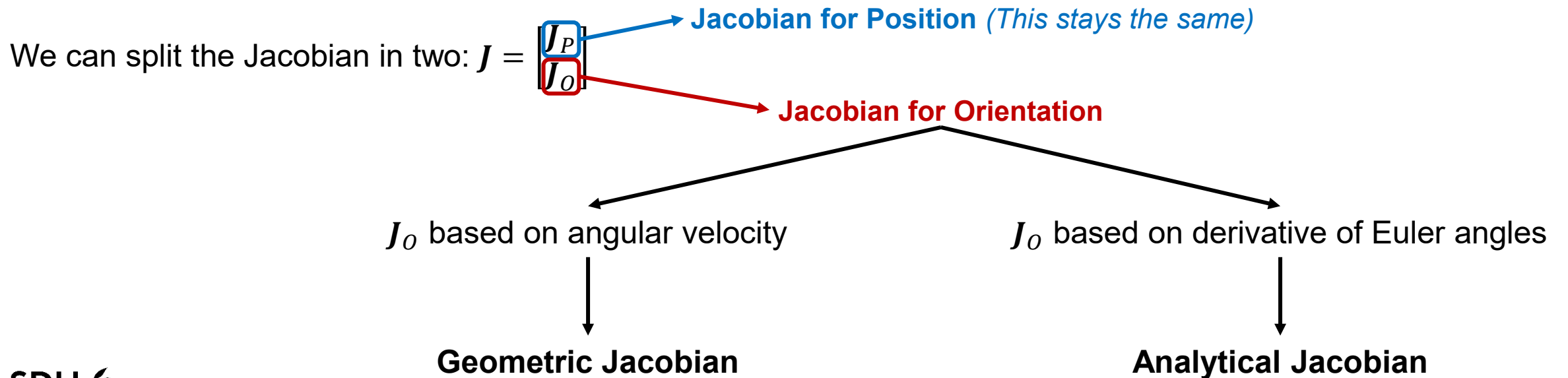
$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -L_1 \sin(q_1) - L_2 \sin(q_1 + q_2) & -L_2 \sin(q_1 + q_2) \\ L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) & L_2 \cos(q_1 + q_2) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

The Two Jacobians in Robotics

What about **orientation**?

Remember that we can **describe Cartesian-space velocity** in (at least) **two ways**, depending on how we describe **rotational velocity**:

- Using **angular** velocity: $\dot{x} = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix}$
- Using the **derivative of Euler angles**: $\dot{x} = v = \begin{bmatrix} \dot{p} \\ \dot{\phi} \end{bmatrix}$



Building a Geometric Jacobian

We first **split the matrix** into (3×1) column vectors, J_{Pi} and J_{Oi} , where each element i represents the **contribution of a single joint q_i to either the position or orientation**:

$$J = \begin{bmatrix} J_{P1} & \dots & J_{Pn} \\ J_{O1} & & J_{On} \end{bmatrix}$$

Then, for each joint i :

- **Revolute:**
$$\left. \begin{aligned} J_{Pi} &= {}^0\hat{Z}_i \times ({}^0P_e - {}^0P_i) \\ J_{Oi} &= {}^0\hat{Z}_i \end{aligned} \right\} \begin{aligned} {}^0\hat{Z}_i &= {}^0_iR \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ {}^0P_e &\text{ is the position of the end-effector} \\ {}^0P_i &\text{ is the position of the origin of frame } \{i\} \end{aligned}$$
- **Prismatic:**
$$\left. \begin{aligned} J_{Pi} &= {}^0\hat{Z}_i \\ J_{Oi} &= 0 \end{aligned} \right\}$$

Calculating the Positional Jacobian Analytically

The **positional part** of the Jacobian, J_{Pi} , can instead be **calculated analytically**.

Given the **forward kinematics** transformation matrix:

$${}^0_eT(\theta) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Take the first three rows of the last column:

$${}^0_eP(\theta) = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \rightarrow J_P = \frac{\partial P}{\partial q} = \begin{bmatrix} \frac{\partial p_x}{\partial q_1} & \dots & \frac{\partial p_x}{\partial q_1} \\ \frac{\partial p_y}{\partial q_1} & \dots & \frac{\partial p_y}{\partial q_1} \\ \frac{\partial p_z}{\partial q_1} & \dots & \frac{\partial p_z}{\partial q_1} \end{bmatrix}$$

Do NOT do this by hand!

If you have the FK in symbolic MATLAB, call:

$J = \text{jacobian}(P)$

<https://se.mathworks.com/help/symbolic/sym.jacobian.html>

Forward and Inverse Differential Kinematics

Given the **forward differential kinematics** equation, and assuming that the **inverse of the Jacobian exists**, we can obtain the inverse differential kinematics equation:

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

This is a **very important result in robotics**.

As we will see, the forward/inverse velocity kinematics equations are the **basis of numerical IK methods**.

The Problem with the Jacobian Inverse

Consider the inverse velocity kinematics equation:

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

Do you see any potential issues here?

Not all matrices have an inverse...
This does not always exist!

So... what happens then?



Singularities: Non-existence of inverse

When does the inverse of a matrix not exist?

Due to a **non-square** matrix:

- Example: **a redundant robot** (e.g. 7-DOF in a 3D space)

$$\dot{x} = J(q)\dot{q}$$

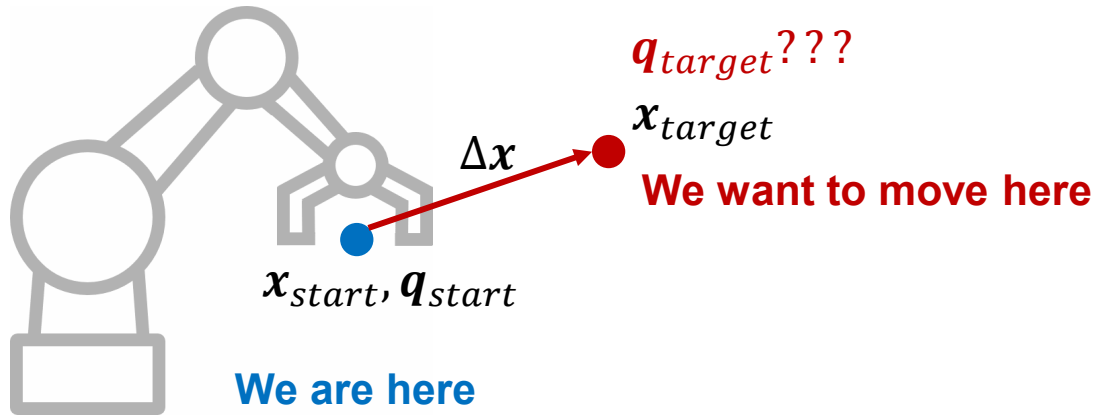
$$\begin{matrix} \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} \\ (6 \times 1) \end{matrix} = \begin{matrix} \begin{bmatrix} J_{P1} & \dots & J_{P7} \\ J_{O1} & & J_{O7} \end{bmatrix} \\ (6 \times 7) \end{matrix} \begin{matrix} \begin{bmatrix} q_1 \\ \vdots \\ q_7 \end{bmatrix} \\ (7 \times 1) \end{matrix}$$



Due to a **rank-deficient square** matrix **(singularities!)** You will study this in detail in the coming lectures.

Part IV: Numerical Inverse Kinematics

Numerical Inverse Kinematics: Conceptually



1. We know: $x_{start}, q_{start}, x_{target}$
2. We want to find: q_{target}
3. We can calculate: $\Delta x = x_{target} - x_{start}$
4. We take Δx and split it in smaller segments:

5. We make a very small step in the direction of Δx :

$$dq(t) = J^{-1}(q)dx(t)$$
$$q(t + dt) = q(t) + dq(t)dt$$

6. We re-calculate: $J(q(t + dt)), x(t + dt)$ (from FK)
7. We check if we are close enough to our target:
 - If yes: stop
 - If not: repeat 3-7.

What happens if, in step 5, the step size is not small enough?

What assumptions are we making (mathematically)?

Formally: Numerical IK using Newton-Raphson

Let us **reformulate** the IK as a **nonlinear optimization problem**. We have:

- A desired Cartesian space pose, x_d
- The desired IK solution, a set of joint angles, θ_d
- A **nonlinear** vector function that computes the FK, $f(\theta_d)$, where $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$

We want: $x_d - f(\theta_d) = 0$ (*this is our **objective function***)

We first apply Taylor expansion:

$$x_d = f(\theta_d) = f(\theta_0) + \underbrace{\frac{\partial f}{\partial x} \Big|_{\theta_0}}_{J(\theta_0)} \underbrace{(\theta_d - \theta_0)}_{\Delta\theta} + \text{higher order terms}$$

By ignoring these, we are **linearizing** around θ_0

$$J(\theta_0)\Delta\theta = x_d - f(\theta_0) \longrightarrow \Delta\theta = J^{-1}(\theta_0)(x_d - f(\theta_0))$$

If we use this to take a **new step**: $\theta_1 = \theta_0 + \Delta\theta$, then this is **guaranteed to be closer to the solution**.

We can therefore **apply this recursively** until we are "close enough", i.e. $x_d - f(\theta_d) < \epsilon$.

IK: Newton Raphson Algorithm

Algorithm 1 Newton-Rhapson method for numerical IK

Require: θ , $x^* = f(\theta)$, x_d and $x_d \approx x^*$

 Compute $\Delta x = x_d - x^*$

while $\|\Delta x\| > \epsilon$ **do**

 Compute $J(\theta)$

 Solve $J(\theta)\Delta\theta = \Delta x$ for $\Delta\theta$

$\theta := \theta + \Delta\theta$

 Compute $x^* = f(\theta)$

 Compute $\Delta x = x_d - x^*$

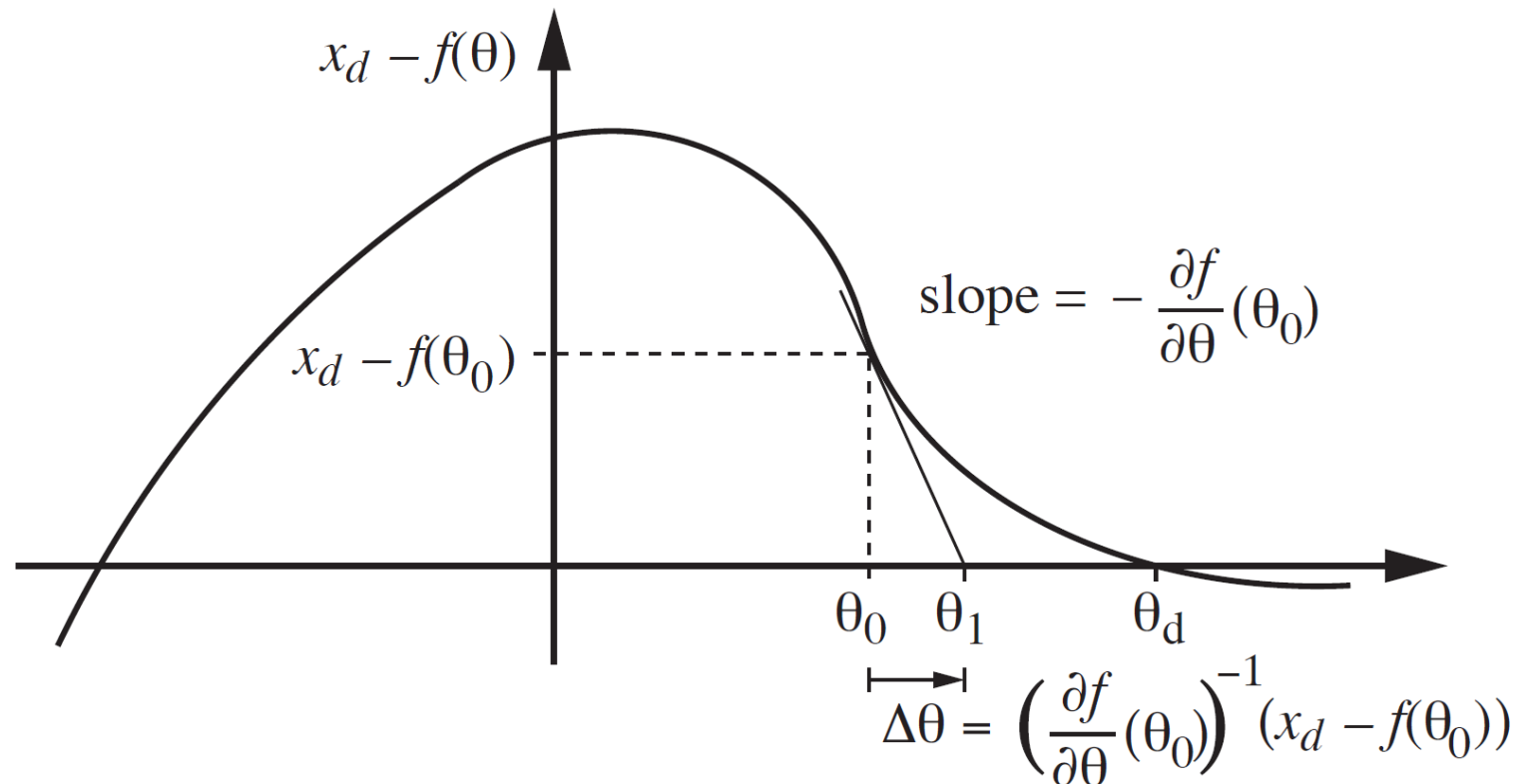
end while

Convergence to Different Solutions

Newton-Rhapson is a **locally optimal** method.


It will **converge** to the solution **closest to the initial guess**.

It is therefore **highly-dependent on the initial guess**.



What if the Jacobian is NOT square?

What do we do with robots whose Jacobians are not square, e.g. redundant robots?

- Jacobian **Transpose**: $\dot{\mathbf{q}} = \alpha \mathbf{J}^T \dot{\mathbf{x}}$ (for some small $\alpha > 0$)
 - This is can behave "close enough" to the inverse assuming a small enough α .
 We will see this in the coming slides
- Jacobian (Moore-Penrose) **Pseudoinverse**: $\dot{\mathbf{q}} = \mathbf{J}^\dagger \dot{\mathbf{x}}$ (where \dagger is the pseudoinverse)
 - This is a generalization of the concept of inverse for non-square matrices: $\mathbf{J}^\dagger = (\mathbf{J}\mathbf{J}^T)^{-1}$
 - If \mathbf{J} has full row rank, then $(\mathbf{J}\mathbf{J}^T)^{-1}$ will always exist.

In MATLAB, \mathbf{J}^\dagger can be calculated using `pinv()`:

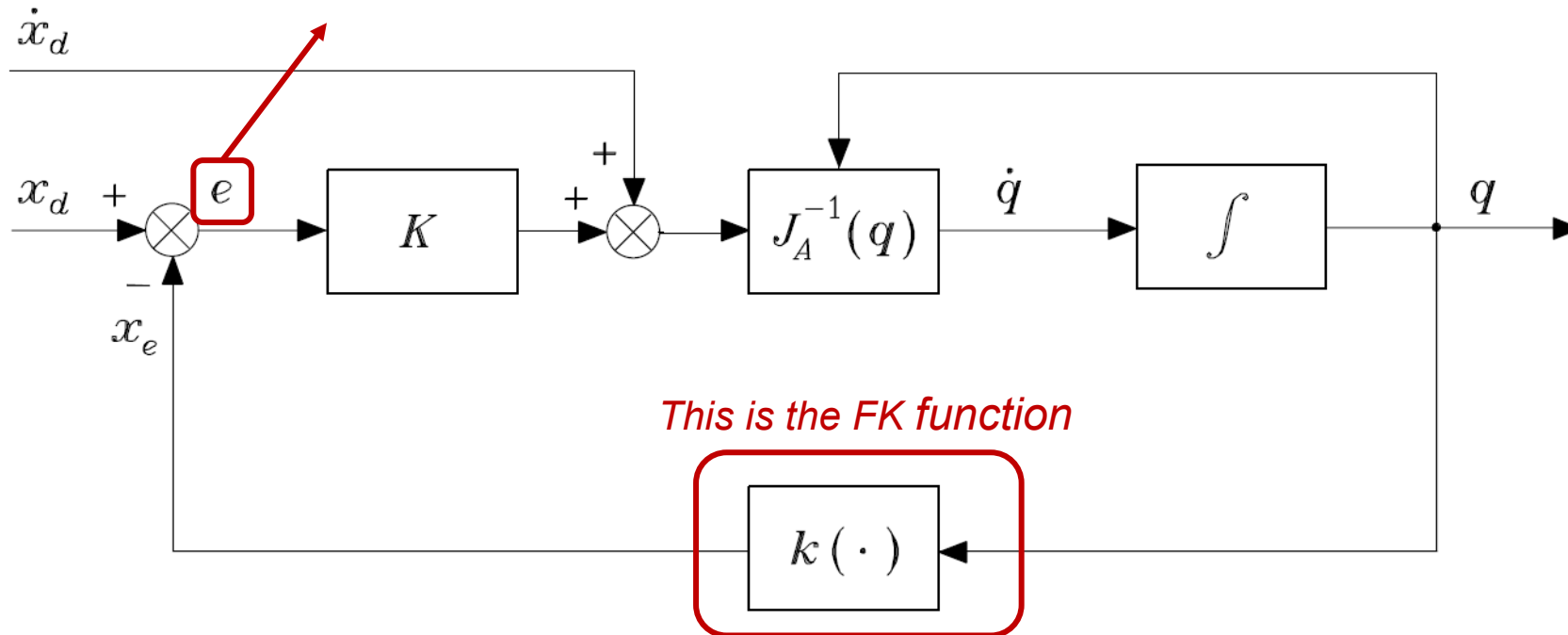
https://se.mathworks.com/help/matlab/ref/pinv.html?s_tid=doc_ta

IK as a Control Problem

We can also reformulate inverse kinematics as a **control problem**:

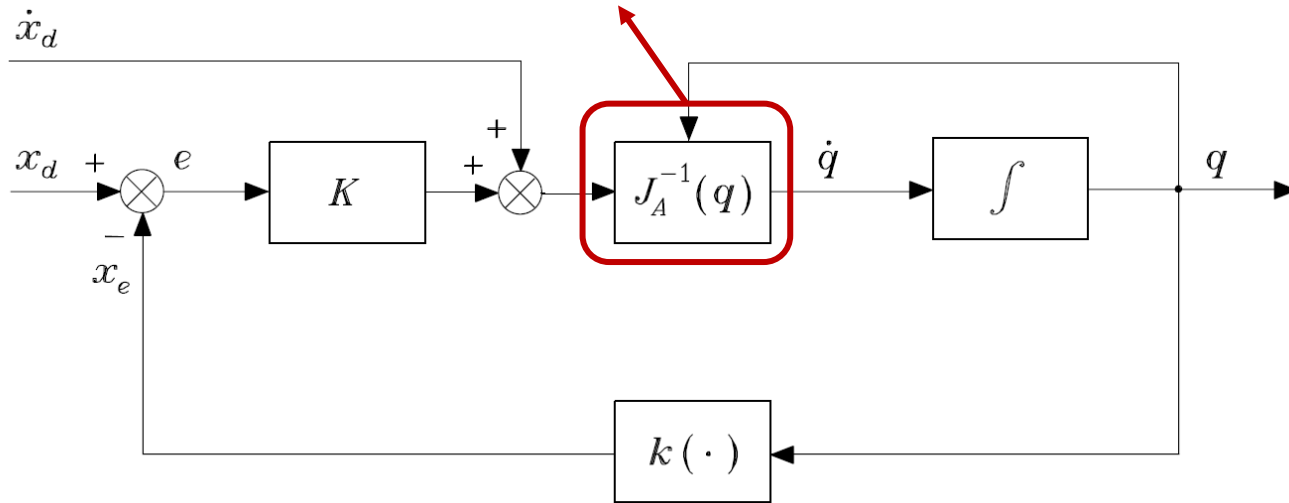
We introduce an error state between the desired and actual operational space poses:

$$e = x_d - x_e \mapsto \dot{e} = \dot{x}_d - \dot{x}_e$$



Jacobian Inverse

$J_A^{-1}(q)$ compensates for $J_A(q)$ to linearize the dynamics



Using differential kinematics, we re-write:

$$\dot{e} = \dot{x}_d - \dot{x}_e = \dot{x}_d - J_A(q)\dot{q}$$

We relate \dot{q} to the e , to obtain a differential equation that describes the evolution of the error dynamics over time.

We want to choose a relationship between \dot{q} and e that ensures the error converges to 0.

If J_A is square and non-singular:

$$\dot{q} = J_A^{-1}(q)(\dot{x}_d + Ke)$$

This leads to an equivalent linear system:

$$\dot{e} + Ke = 0 \longrightarrow \dot{e} = -Ke$$

If $\dot{x}_d = 0$, this is the same as the Newton-Raphson method

If K is positive definite, the system is asymptotically stable.

Jacobian Transpose

We may want a **computationally simpler** algorithm that **does not require linearization**.

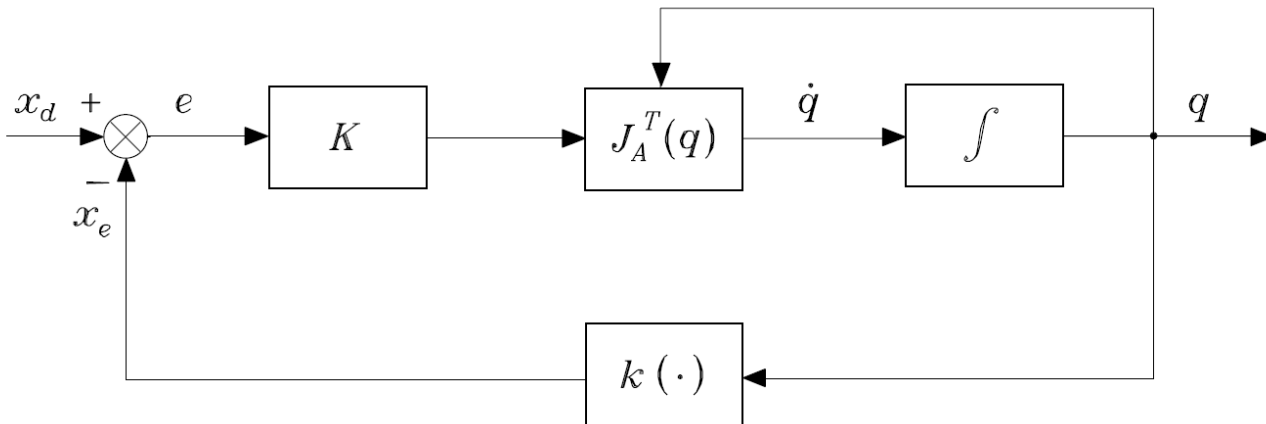
This means that the **equations for the system will be non-linear**.

The joint velocities are given by:

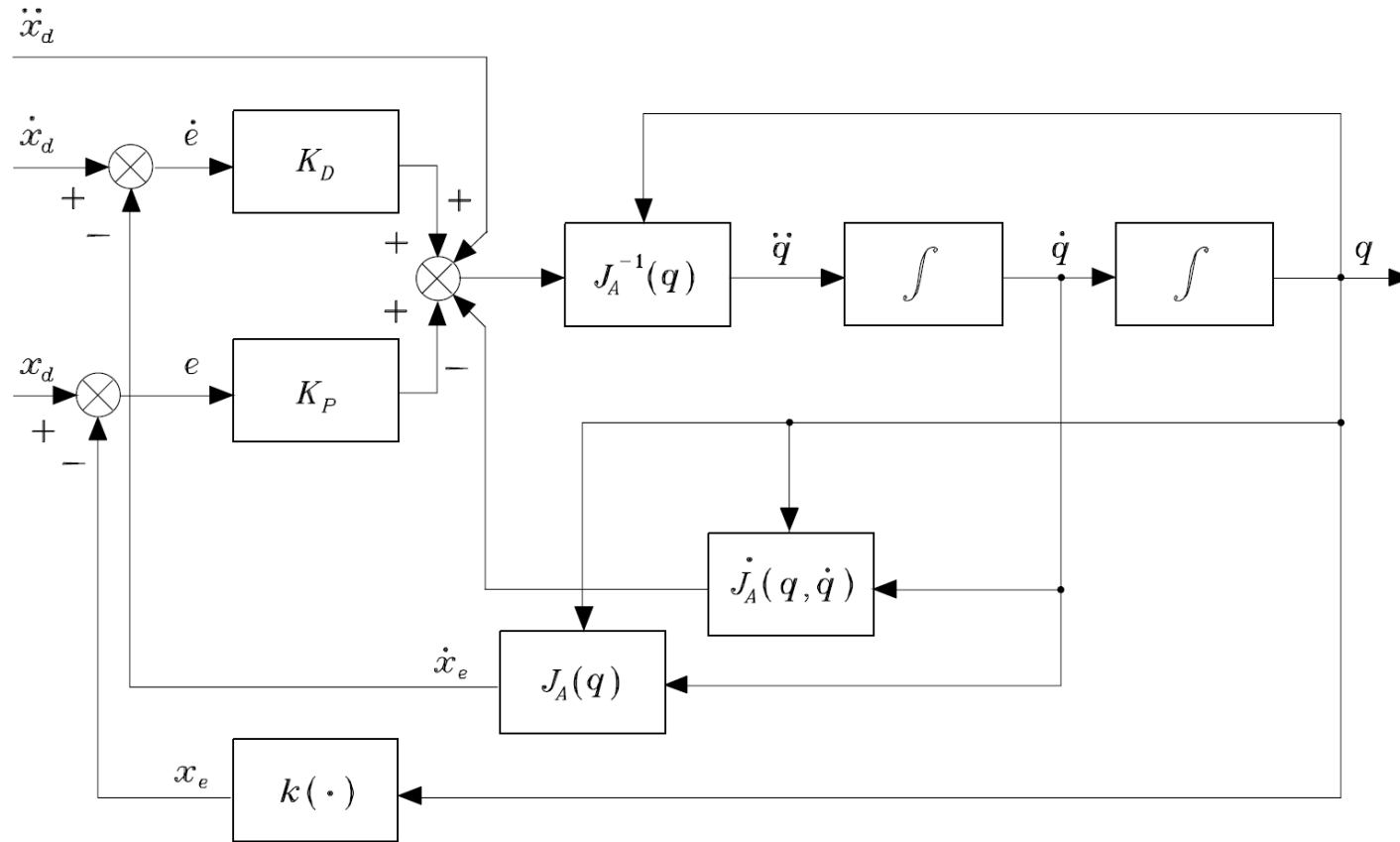
$$\dot{q} = J_A^T(q) K e,$$

Using a **Lyapunov function**¹ we can prove that, if **K** is positive definite:

- For a constant reference, $\dot{x}_d = 0$, the system converges ($e = 0$).
- There is a case where the robot can stop before having converged with $e \neq 0$, but this only happens when the desired pose is unreachable.



Second-Order Algorithm (I)



It might be necessary to **include accelerations** in the motion specification.

We can then specify the IK problem as a **second-order system**.

This is **related to the dynamical model**¹ of the manipulator which is also specified as a second-order system.

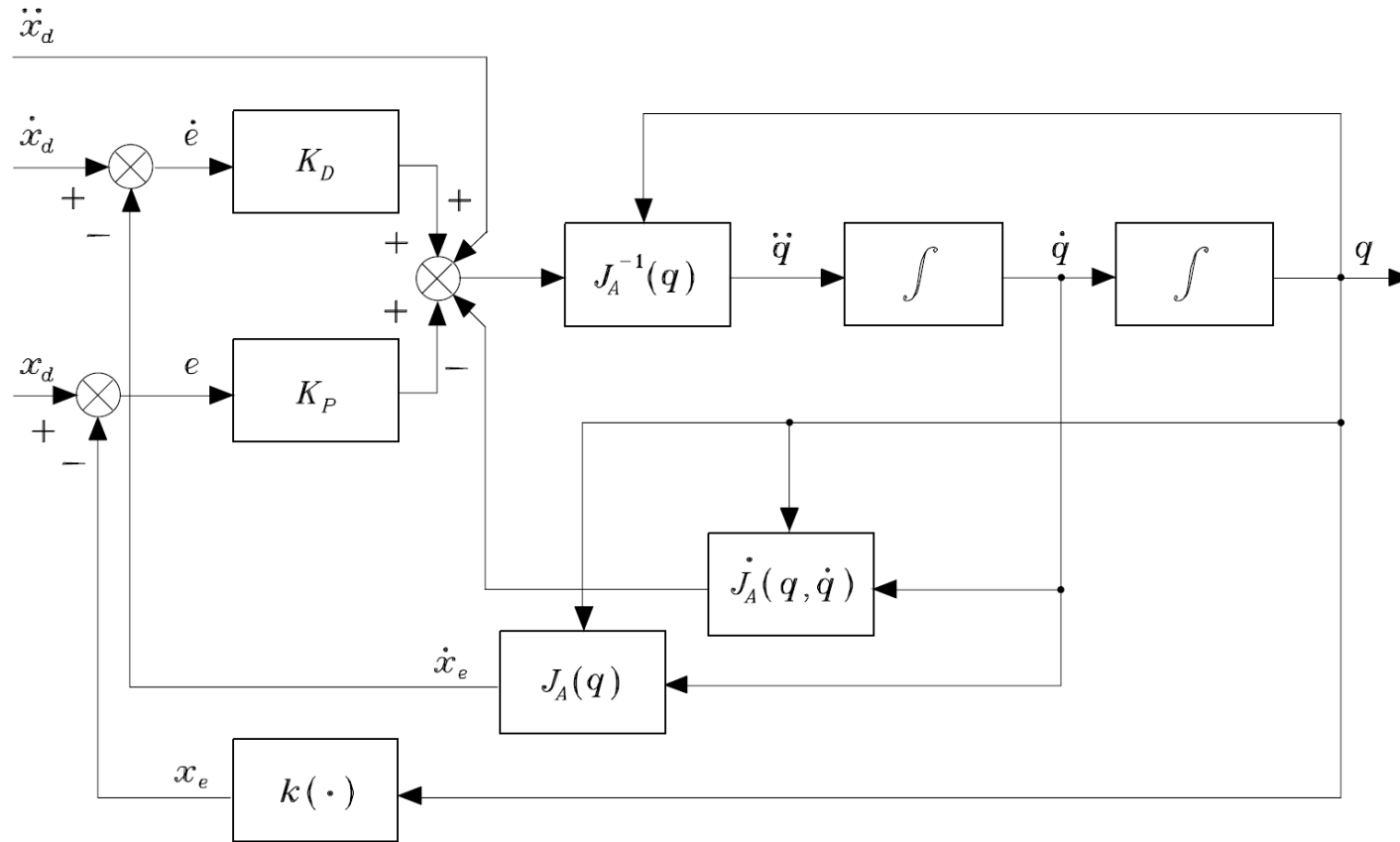
By differentiating $\dot{x}_e = J_A(q)\dot{q}$ we get:

$$\ddot{x}_e = J_A(q)\ddot{q} + \dot{J}_A(q, \dot{q})\dot{q}$$

If $J_A(q)$ is invertible, then:

$$\ddot{q} = J_A^{-1}(q)(\ddot{x}_e - \dot{J}_A(q, \dot{q})\dot{q})$$

Second-Order Algorithm (II)



Integrating \ddot{q} numerically will lead to **drift**.

We consider the **acceleration error**:

$$\ddot{e} = \ddot{x}_d - \ddot{x}_e$$

$$\ddot{e} = \ddot{x}_d - J_A(q)\ddot{q} - \dot{J}_A(q, \dot{q})\dot{q}$$

leading to the choice of **control law**:

$$\ddot{q} = J_A^{-1}(q)(\ddot{x}_d + K_D\dot{e} + K_P e - \dot{J}_A(q, \dot{q})\dot{q})$$

Substituting, the **error dynamics** become:

$$\ddot{e} + K_D\dot{e} + K_P e = 0 \longrightarrow \boxed{\ddot{e} = -K_D\dot{e} - K_P e}$$

If K_D and K_P are **positive definite**, the system is **asymptotically stable**.

Recap: What have we learned today?

1. While FK is a relatively straightforward problem, **IK is more complicated**, as the mapping:
 - May have **multiple solutions**.
 - May have **no solutions**.
2. There exist **analytical** (exact, only exist for some robots) **and numerical** (approximation, exist for all robots) **solutions to the IK problem**.
3. The **Jacobian maps joint velocities to Cartesian-space velocities**. There exist two types:
 - **Geometric**: The orientational part is expressed as **angular velocities** (i.e. not the derivative of the orientation).
 - **Analytical**: The orientational part is **directly differentiated from**, e.g. **Euler angles**.
4. There are **multiple algorithms for numerical IK**, all of which **use the Jacobian**:
 - **Newton-Raphson**
 - **Jacobian (Pseudo-)Inverse**
 - **Jacobian Transpose**
 - **Second-Order Algorithms**

Thank you for today.

Iñigo Iturrate



[Ø27-604-3](tel:027-604-3)



inju@mmmi.sdu.dk