Lecture 03:

Character Kinematics: Forward and Inverse Kinematics

Libin Liu

School of Intelligence Science and Technology
Peking University

GAMES105 课程交流

VCL @ PKU

Welcome & Course Information

- Instructor: Libin Liu (http://libliu.info)
- Website: https://games-105.github.io/
- Lecture: Monday 8:00PM to 9:00PM (12 Weeks)
- Prerequisites: linear algebra, calculus,
  programming skills (python),
  probability theory, mechanics, ML, RL…
- Exercise:
  - Codebase: https://github.com/GAMES-105/GAMES-105
  - Submission: http://cn.ces-alpha.org/course/register/GAMES-105-Animation-2022/
  - Register code: **GAMES-FCA-2022**
- BBS: https://github.com/GAMES-105/GAMES-105/discussions
- QQ Group: 533469817

群名称:GAME105课程交流群
群　号:533469817

Outline

- Character Kinematics
  - Skeleton and forward Kinematics

- Inverse Kinematics
  - IK as a optimization problem
  - Optimization approaches
    - Cyclic Coordinate Descent (CCD)
    - Jacobian and gradient descent method
    - Jacobian inverse method

Character Kinematics

kinematics /ˌkɪnɪˈmætɪks/

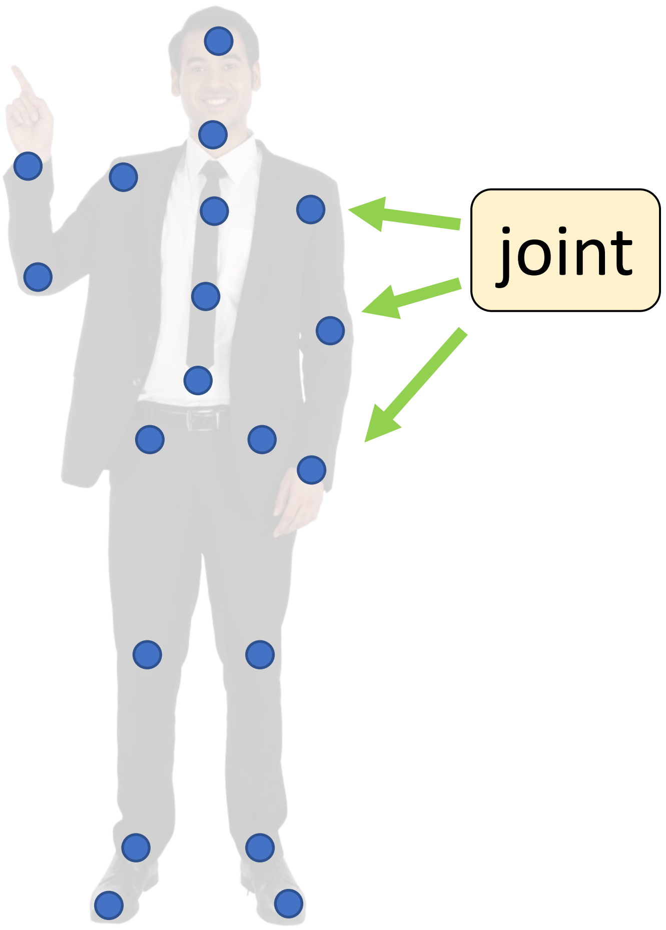
n. the study of the motion of bodies without reference to mass or force

-- Collins English Dictionary

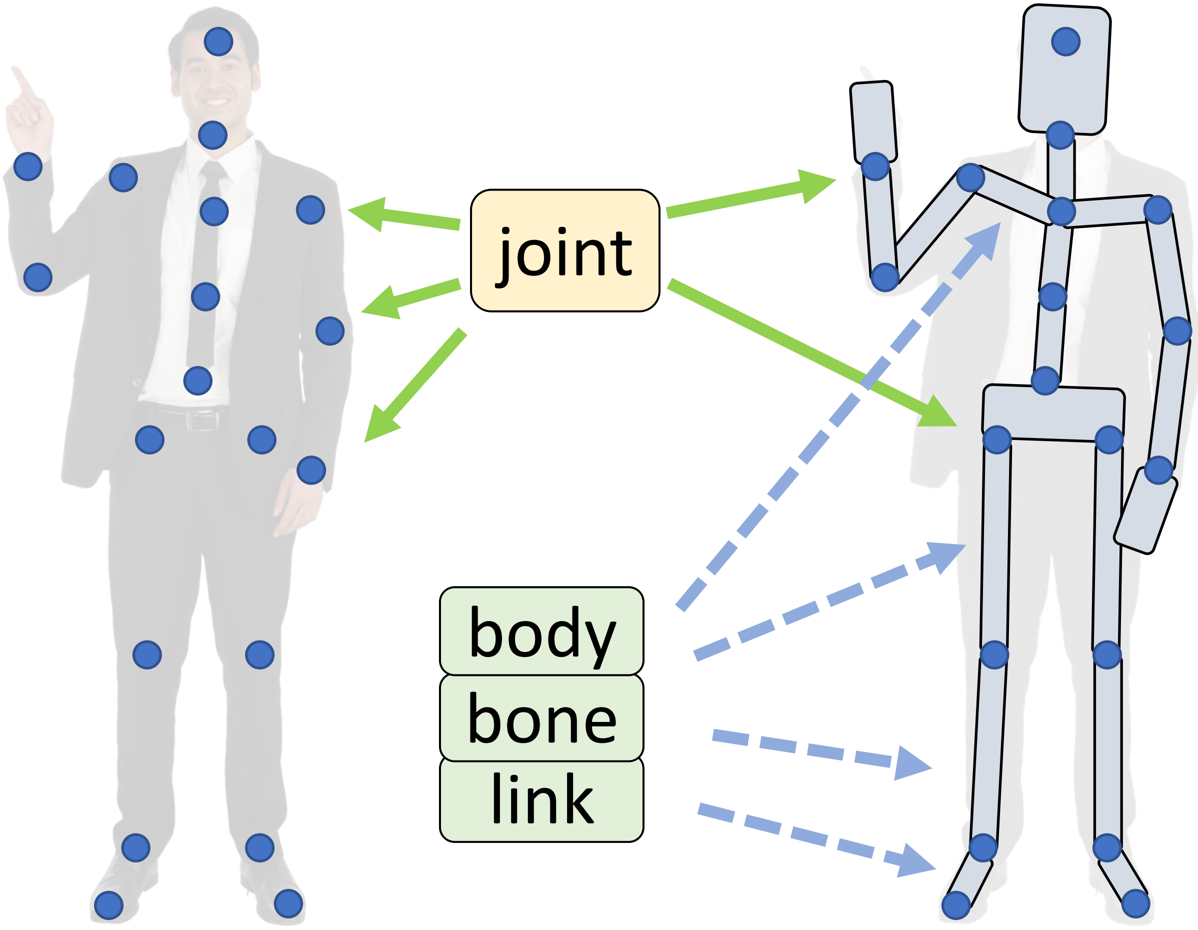
Characters

Skeleton

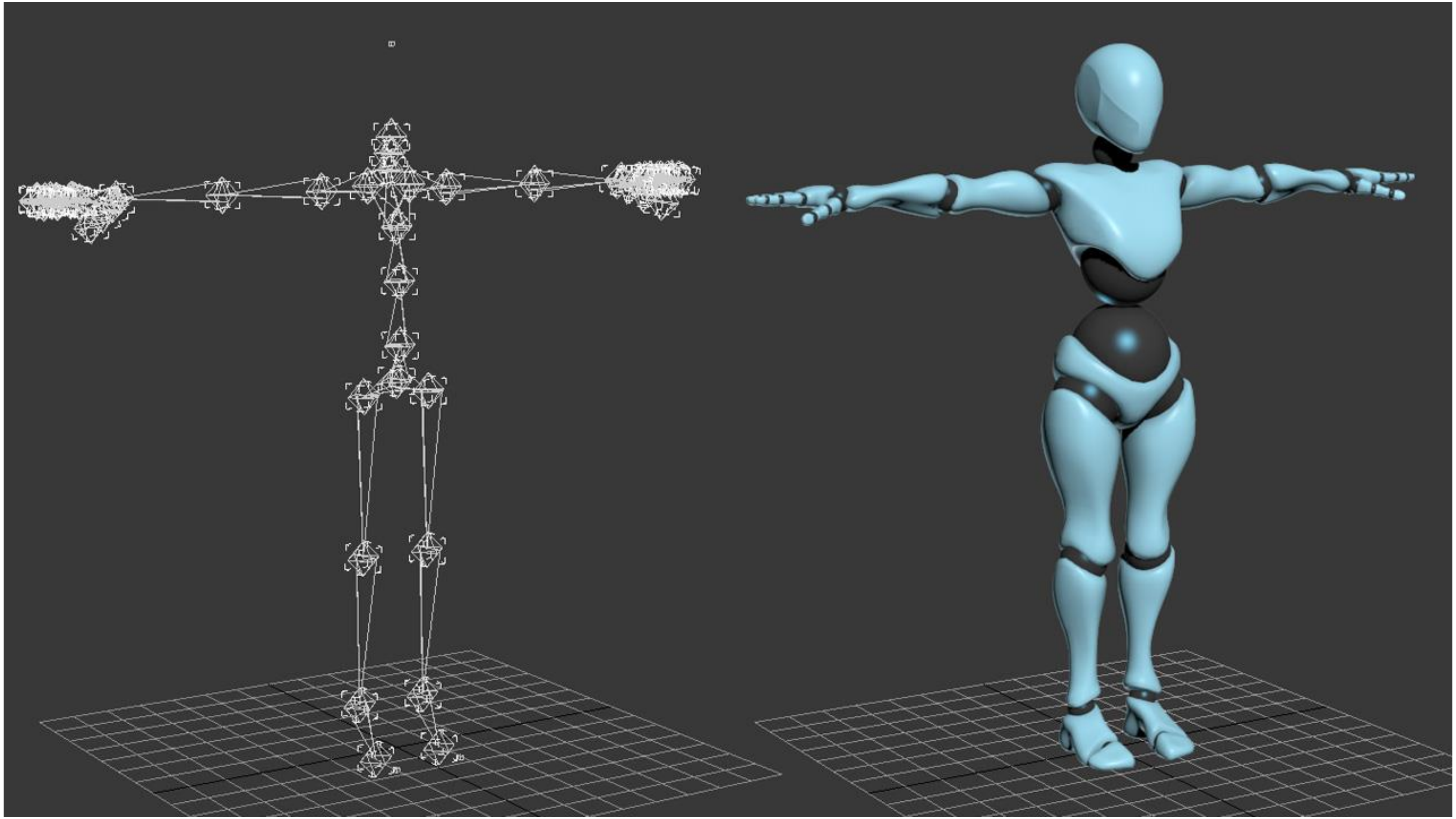
Skeleton



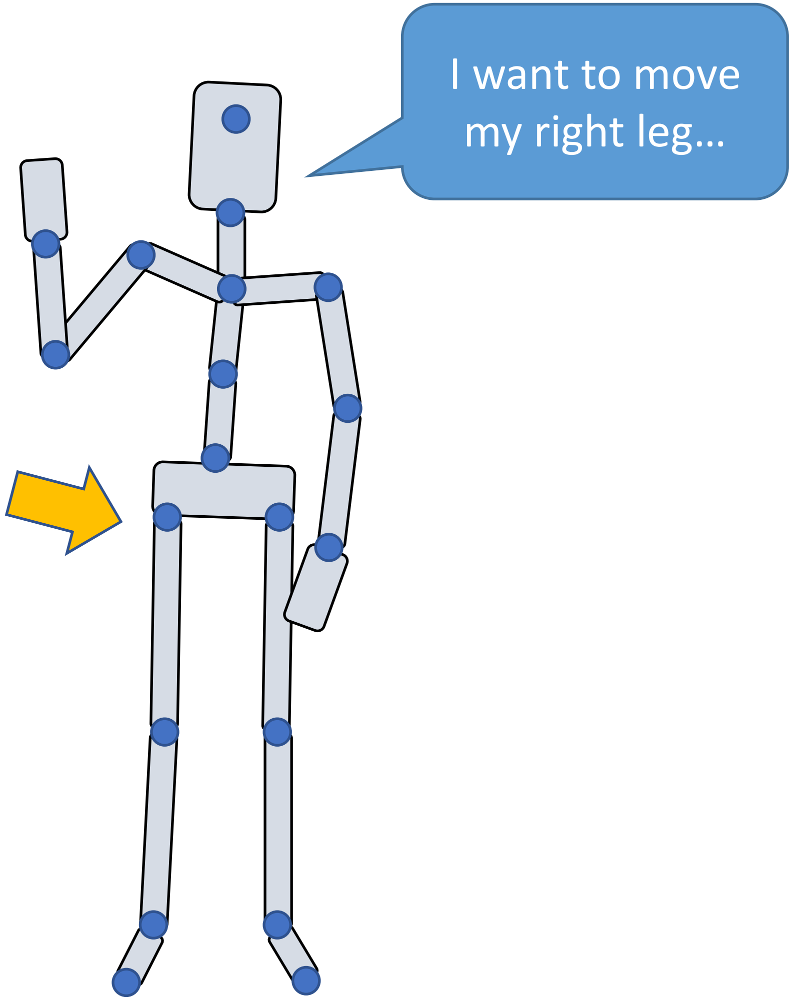
Skeleton



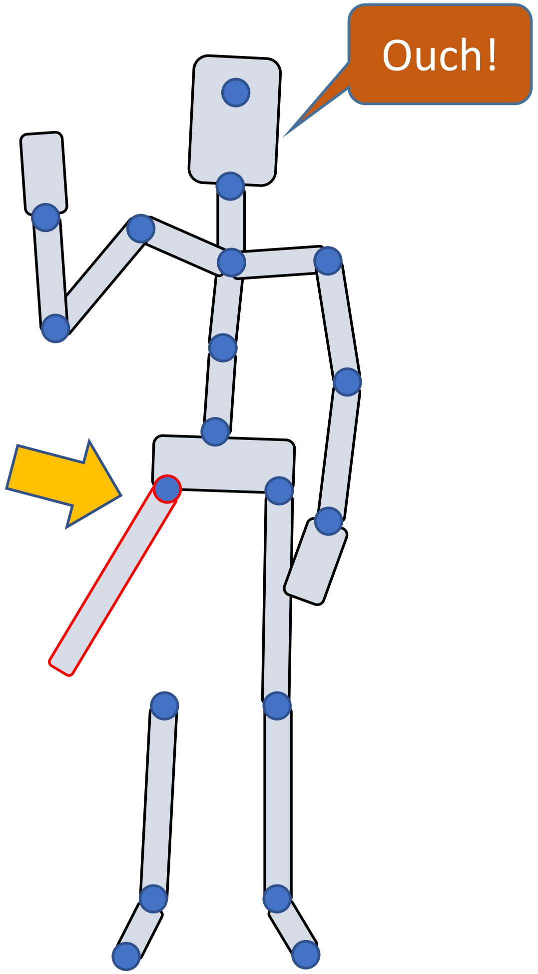
Skeleton



How to create a pose



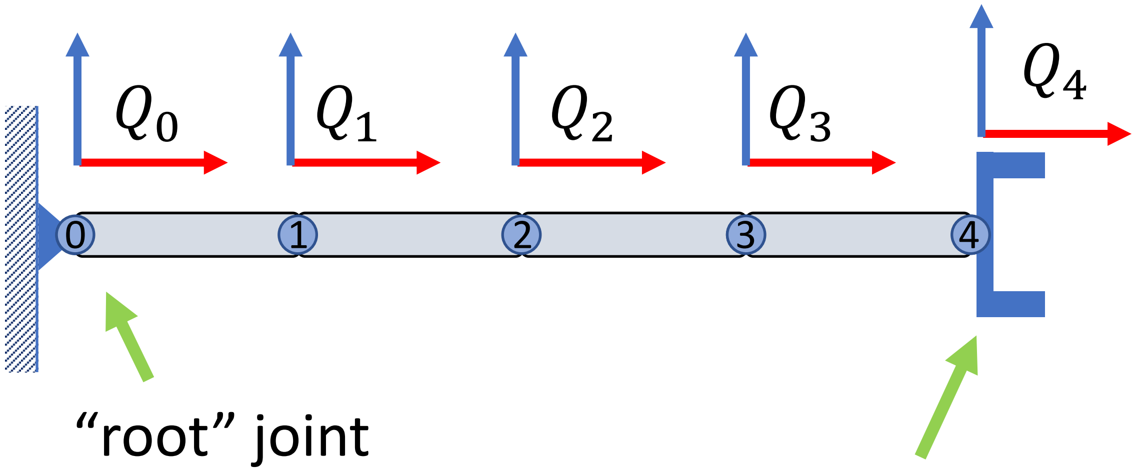
How to create a pose



- Joint will not take effect automatically…
- We need to calculate the position and orientation of each bone carefully.
- But how?

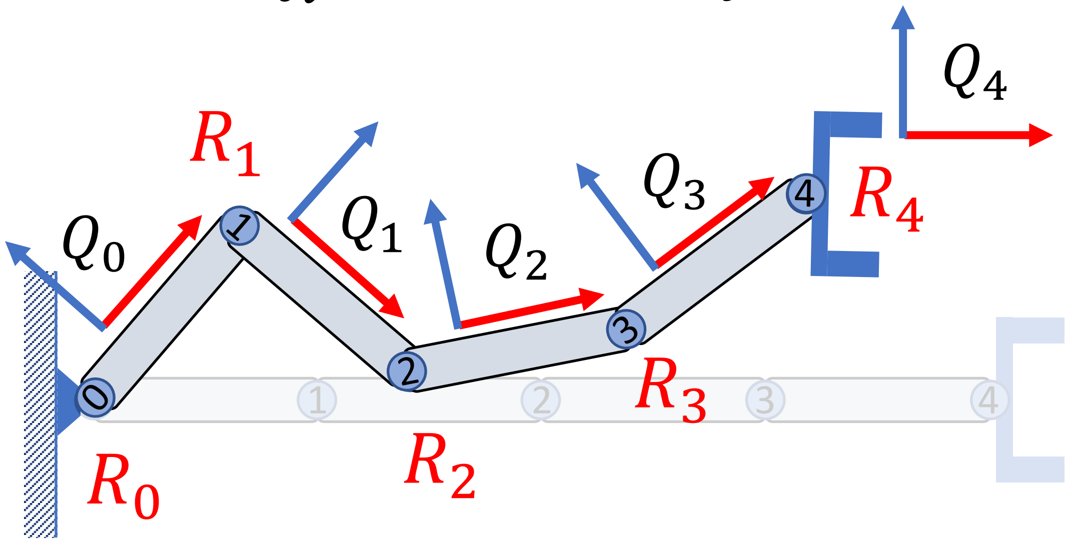
Kinematics of a Chain

Q_i: orientation of joint i



Kinematics of a Chain

Q_i: orientation of joint i



R_i: rotation of joint i

$Q_0 = ?$

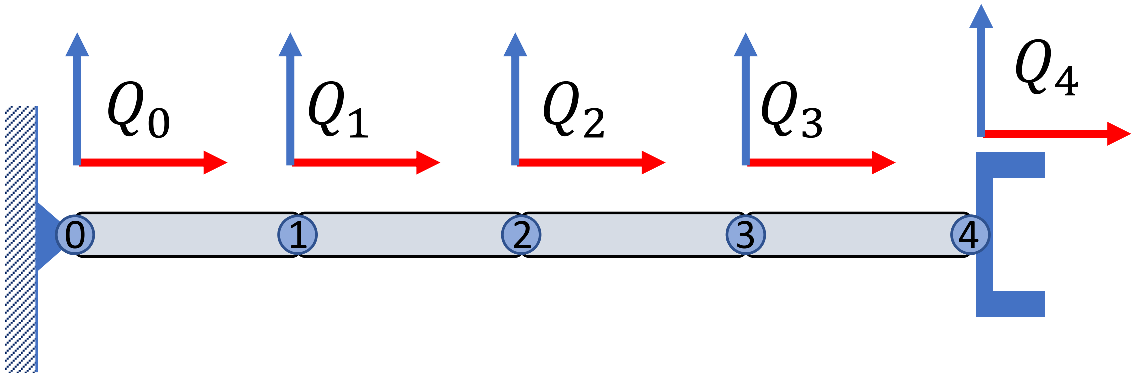
$Q_1 = ?$

$Q_2 = ?$

$Q_3 = ?$

$Q_4 = ?$

Kinematics of a Chain



$$Q_0 = I$$

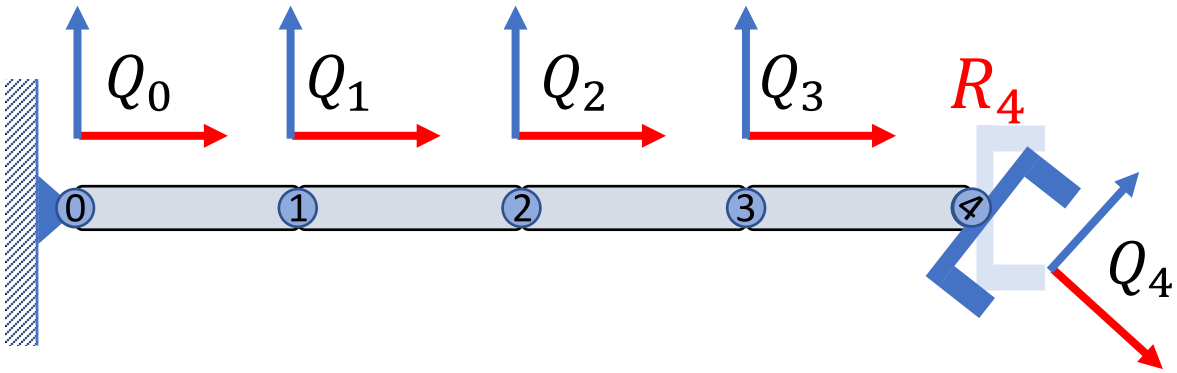
$$Q_1 = I$$

$$Q_2 = I$$

$$Q_3 = I$$

$$Q_4 = I$$

Kinematics of a Chain



$$Q_0 = I$$

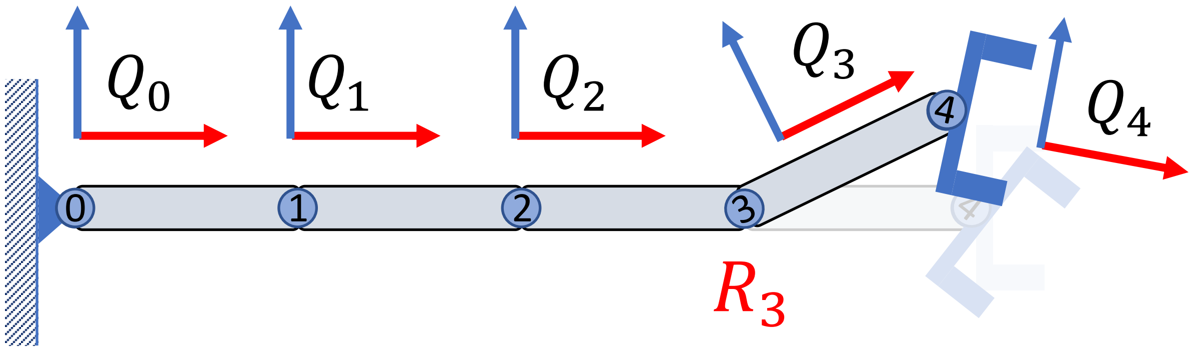
$$Q_1 = I$$

$$Q_2 = I$$

$$Q_3 = I$$

$$Q_4 = R_4$$

Kinematics of a Chain



$$Q_0 = I$$

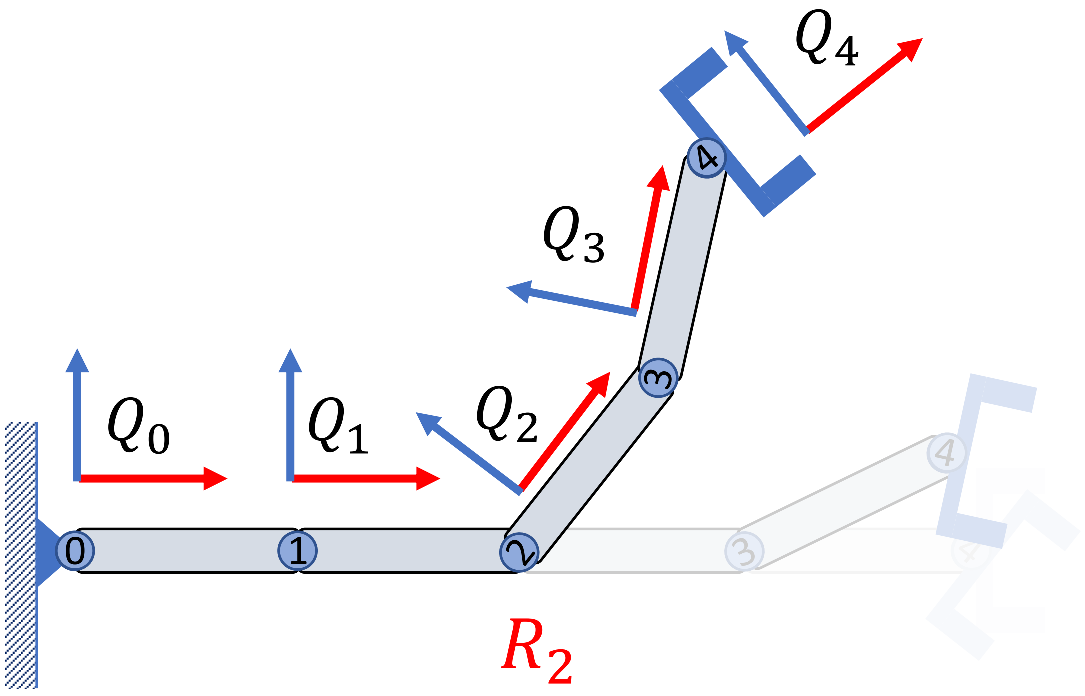
$$Q_1 = I$$

$$Q_2 = I$$

$$Q_3 = R_3$$

$$Q_4 = R_3 R_4$$

Kinematics of a Chain



$$Q_0 = I$$

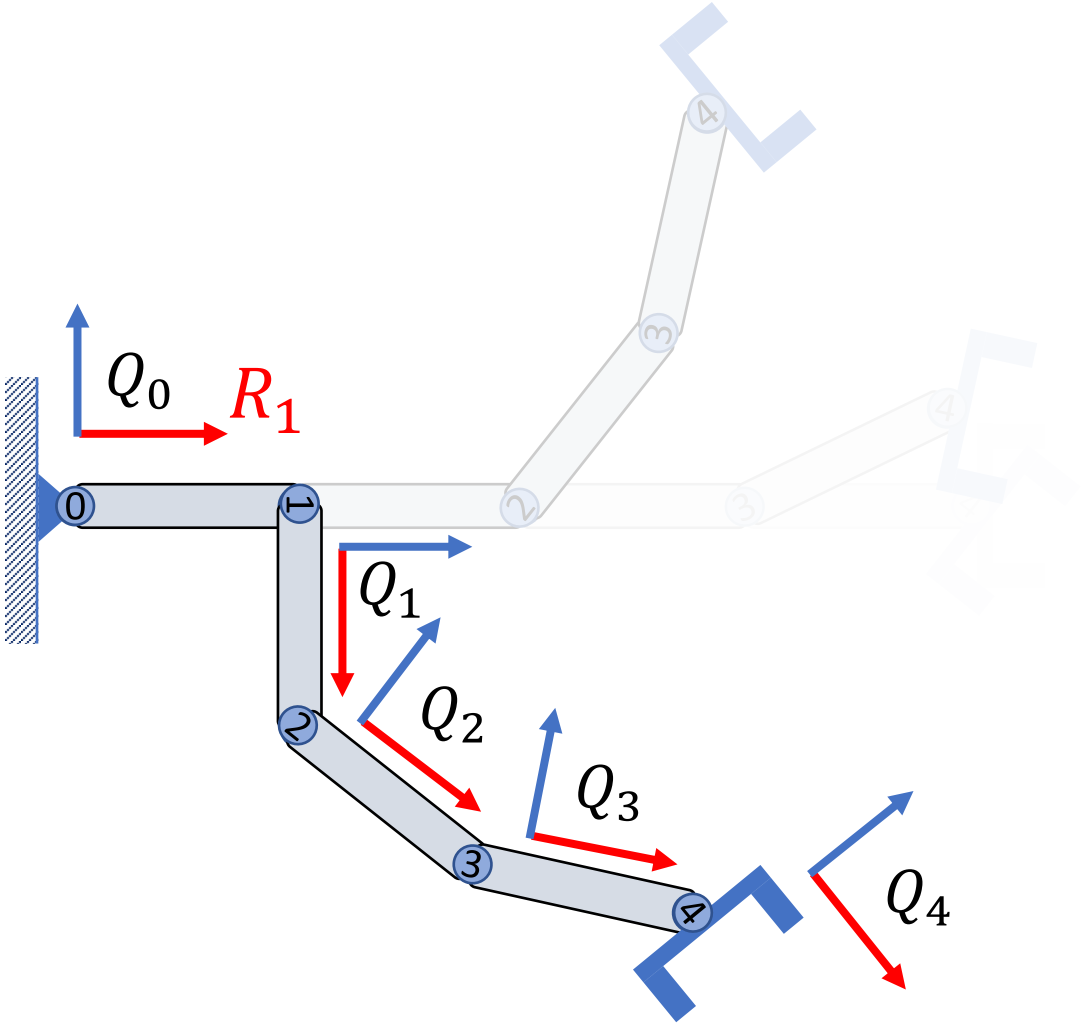
$$Q_1 = I$$

$$Q_2 = R_2$$

$$Q_3 = R_2 R_3$$

$$Q_4 = R_2 R_3 R_4$$

Kinematics of a Chain



$$Q_0 = I$$

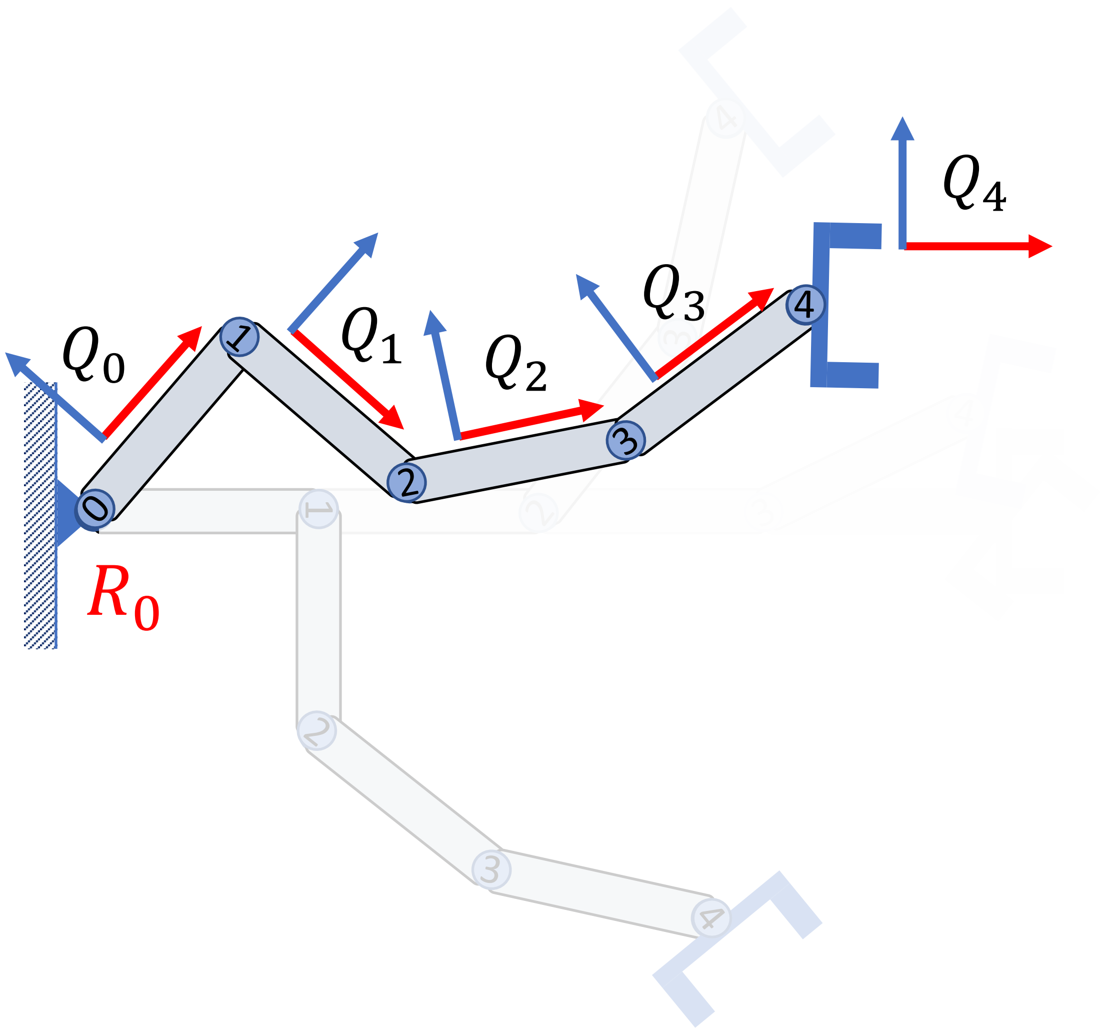
$$Q_1 = R_1$$

$$Q_2 = R_1 R_2$$

$$Q_3 = R_1 R_2 R_3$$

$$Q_4 = R_1 R_2 R_3 R_4$$

Kinematics of a Chain



$$Q_0 = R_0$$

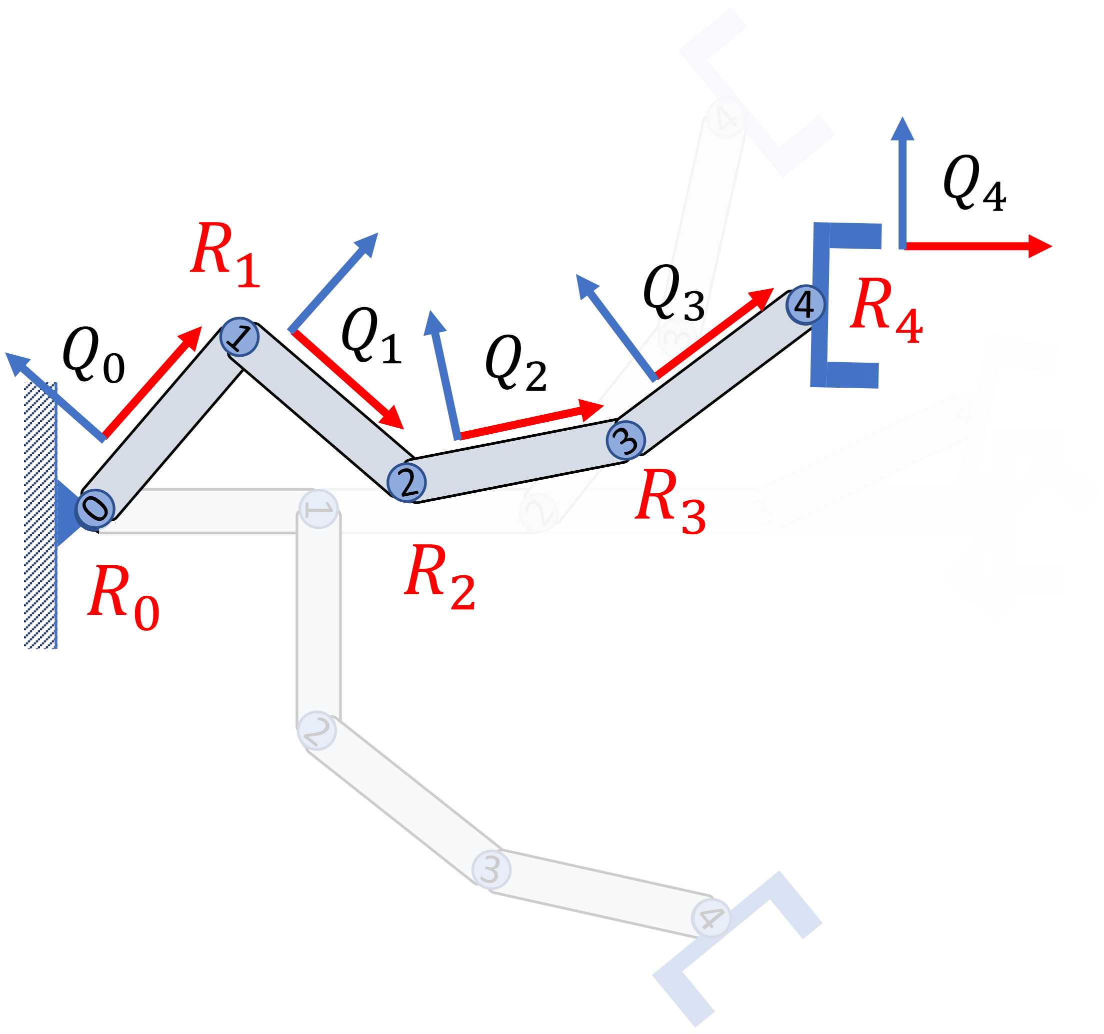
$$Q_1 = R_0 R_1$$

$$Q_2 = R_0 R_1 R_2$$

$$Q_3 = R_0 R_1 R_2 R_3$$

$$Q_4 = R_0 R_1 R_2 R_3 R_4$$

Kinematics of a Chain



$$Q_0 = R_0$$

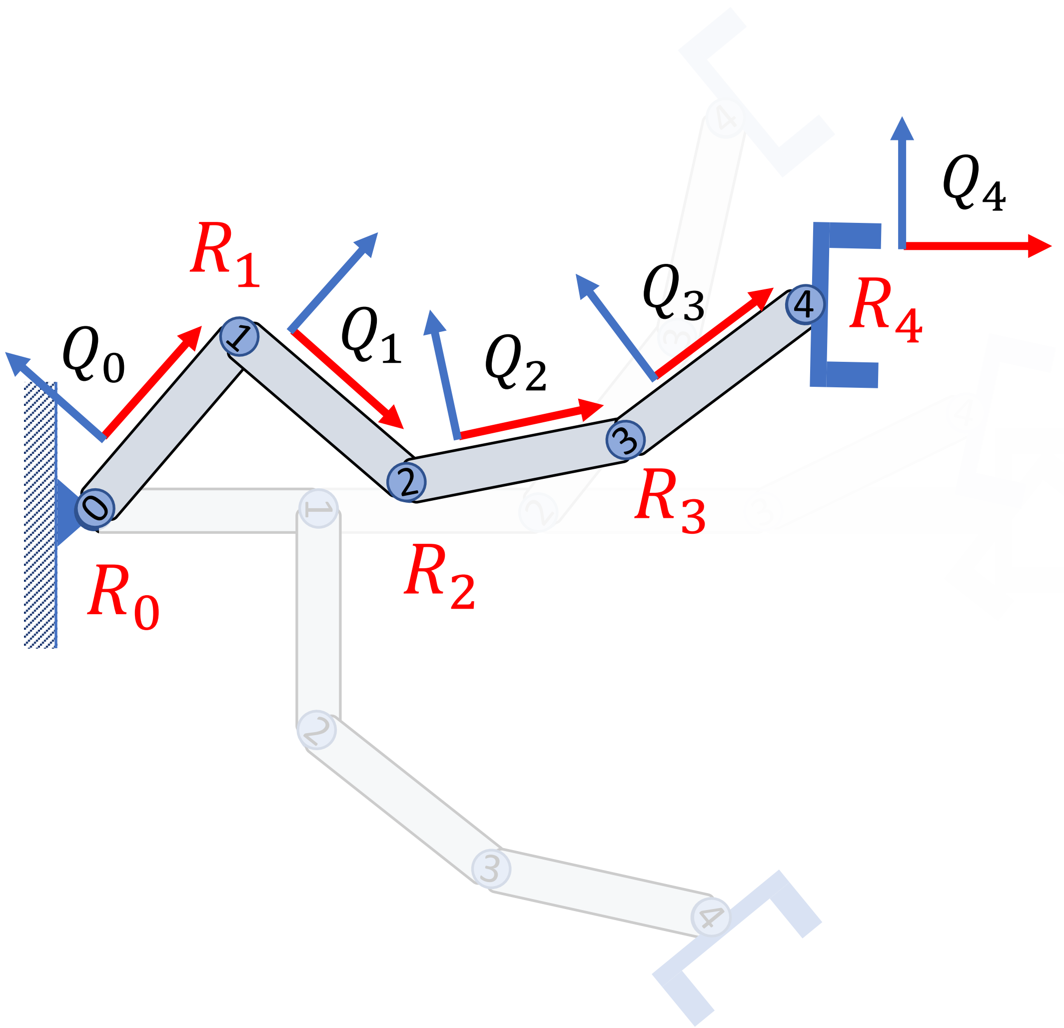
$$Q_1 = Q_0 R_1$$

$$Q_2 = Q_1 R_2$$

$$Q_3 = Q_2 R_3$$

$$Q_4 = Q_3 R_4$$

Kinematics of a Chain



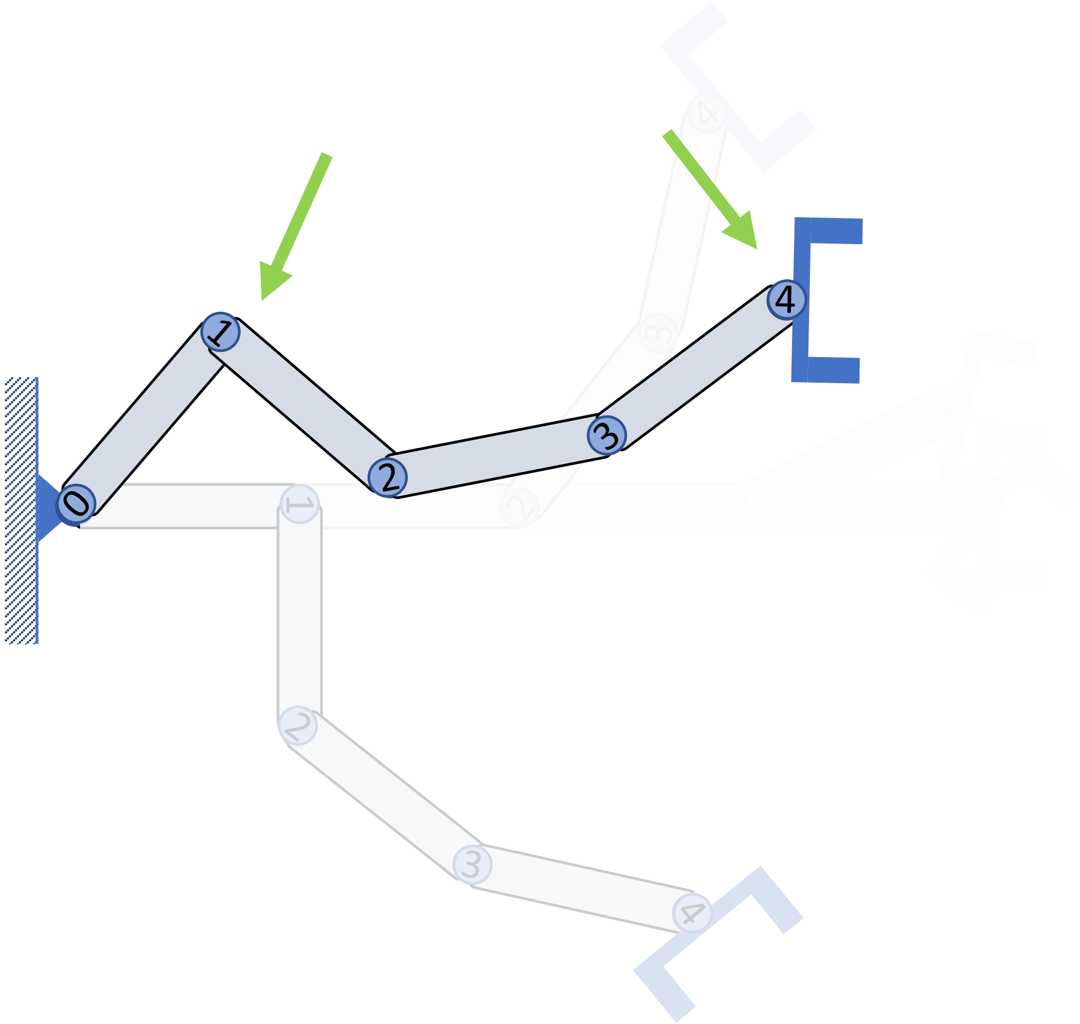
local global
From rotation to orientation

$$Q_i = Q_{i-1} R_i$$

global local
From orientation to rotation

$$R_i = Q_{i-1}^T Q_i$$

Kinematics of a Chain



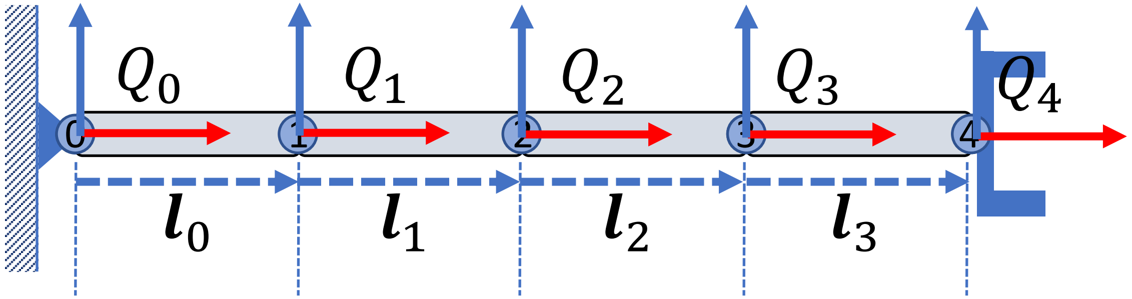
Relative rotation

$$R_4^1 = Q_1^T Q_4$$

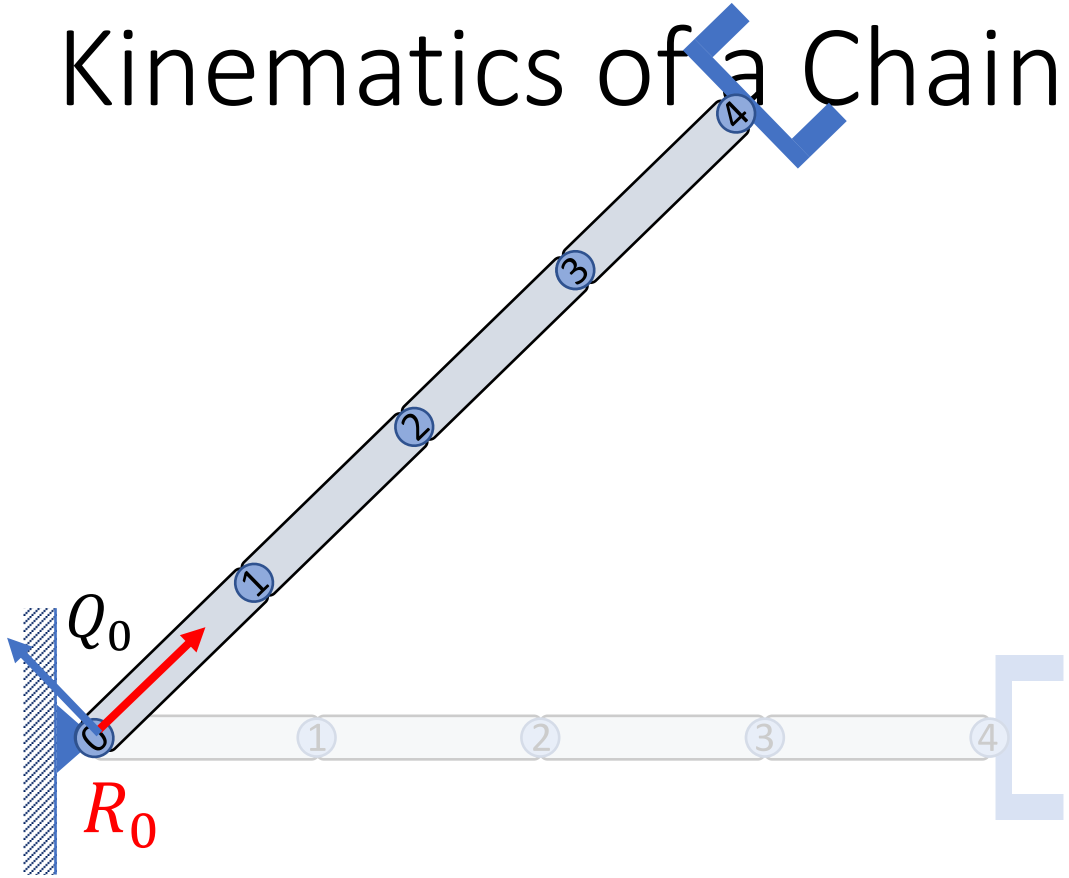
$$= (R_0 R_1)^T R_0 R_1 R_2 R_3 R_4$$

$$= R_2 R_3 R_4$$

Kinematics of a Chain

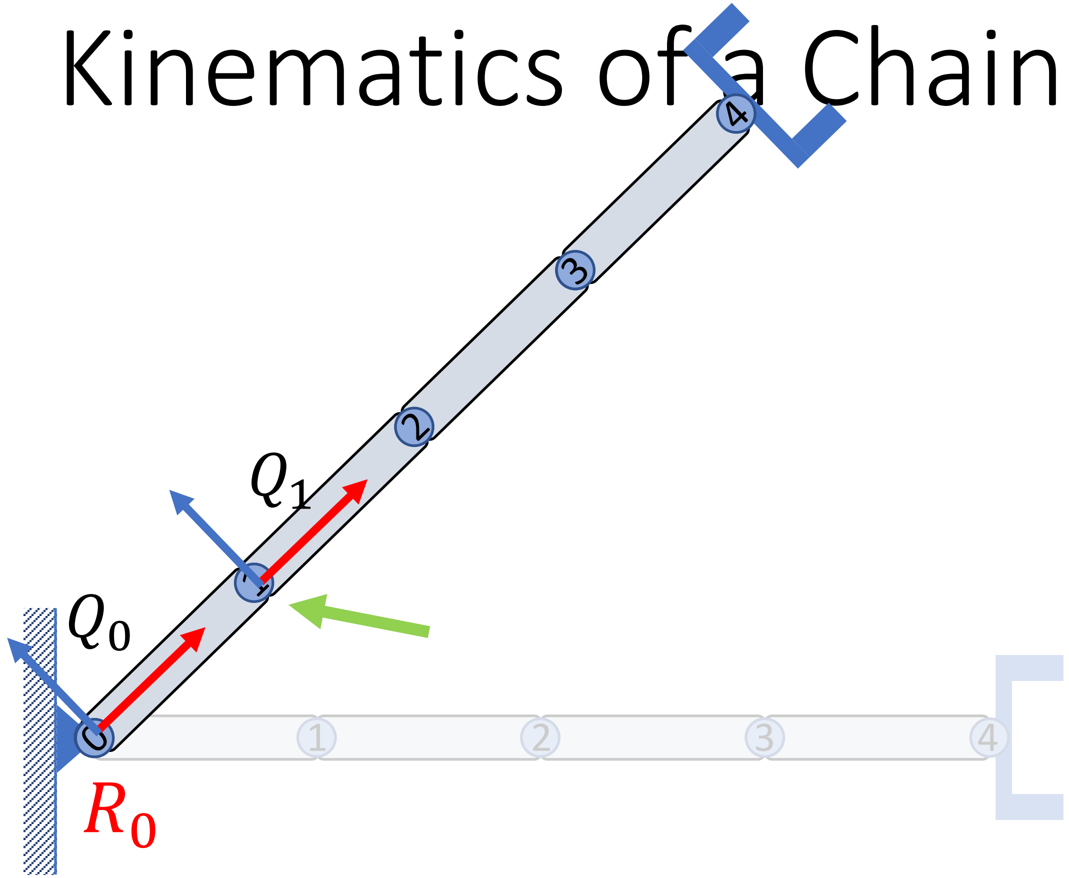


Kinematics of a Chain



$$Q_0 = R_0$$

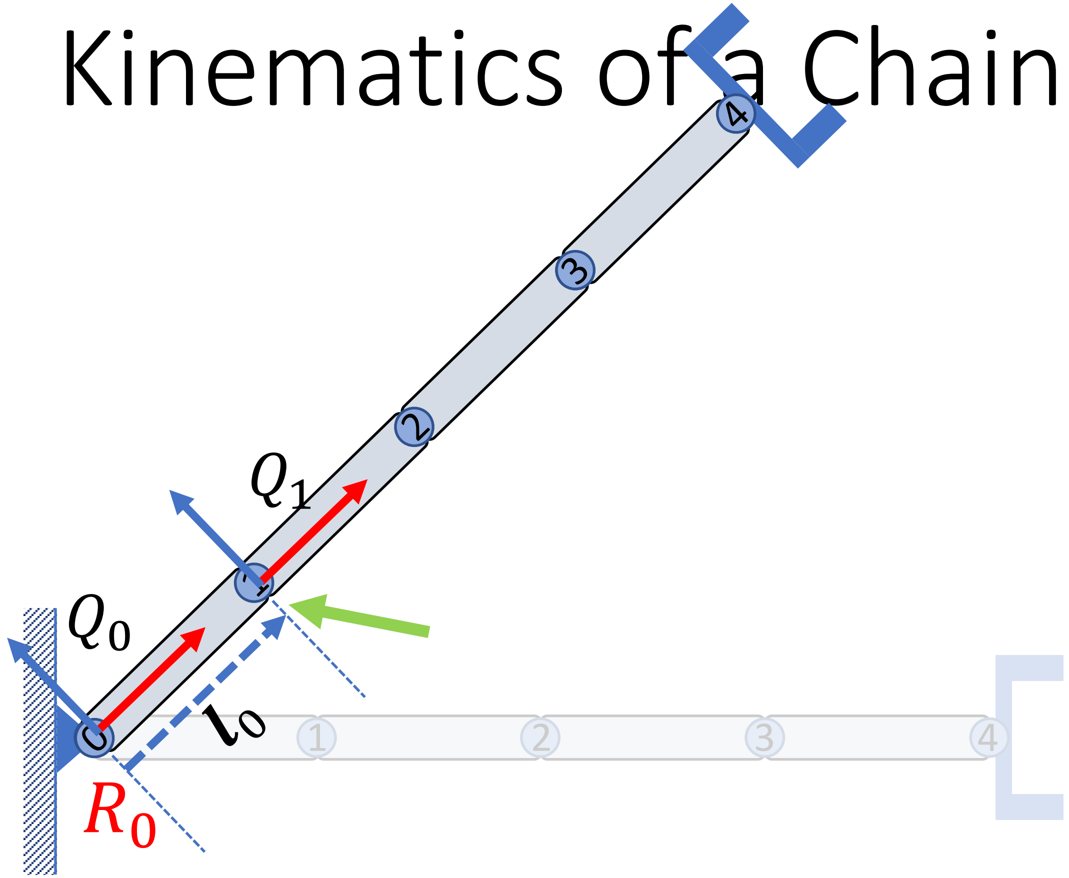
Kinematics of a Chain



$$Q_0 = R_0$$

$$\boldsymbol{p}_1 = ?$$

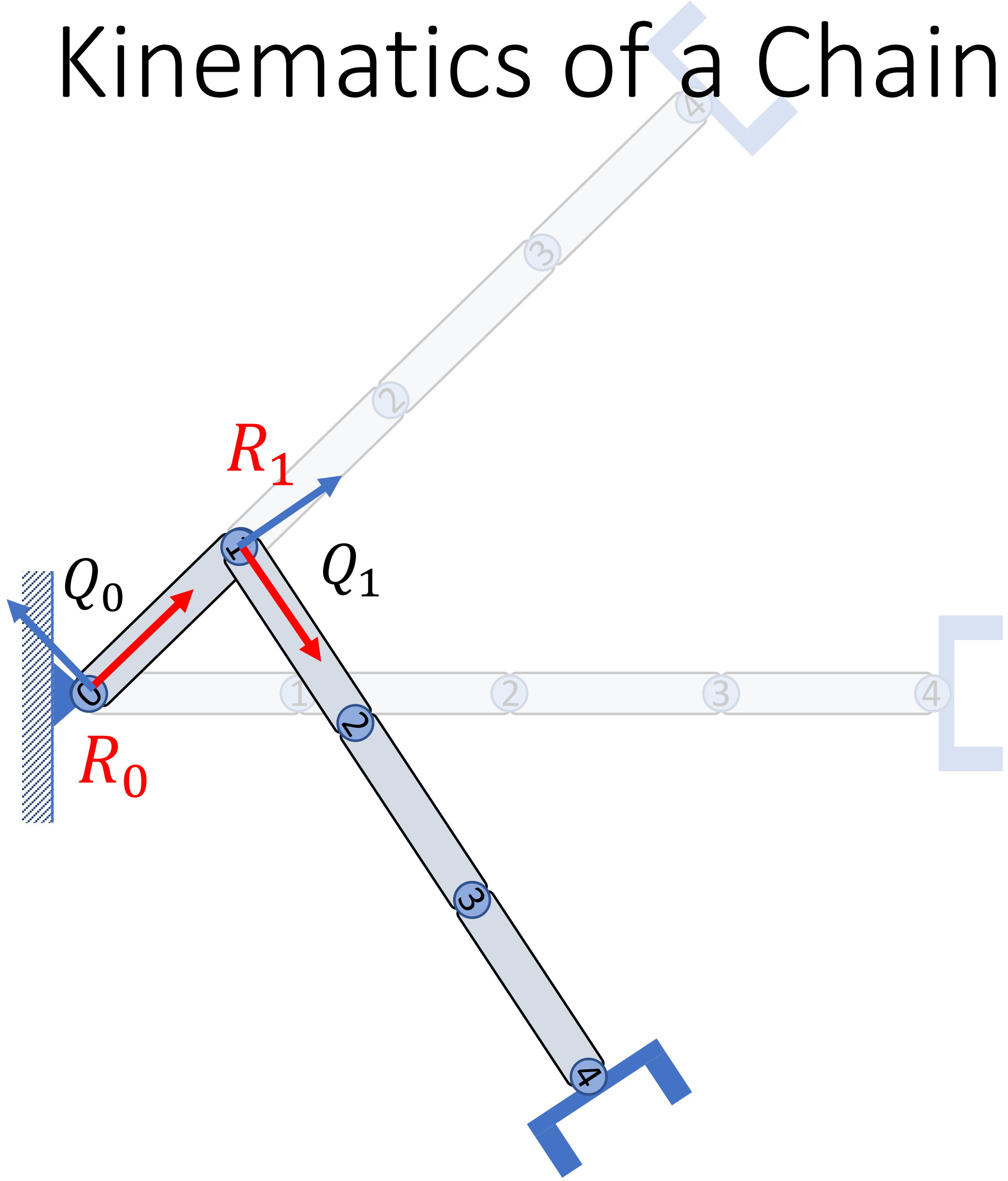
Kinematics of a Chain



$$Q_0 = R_0$$

$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_0$$

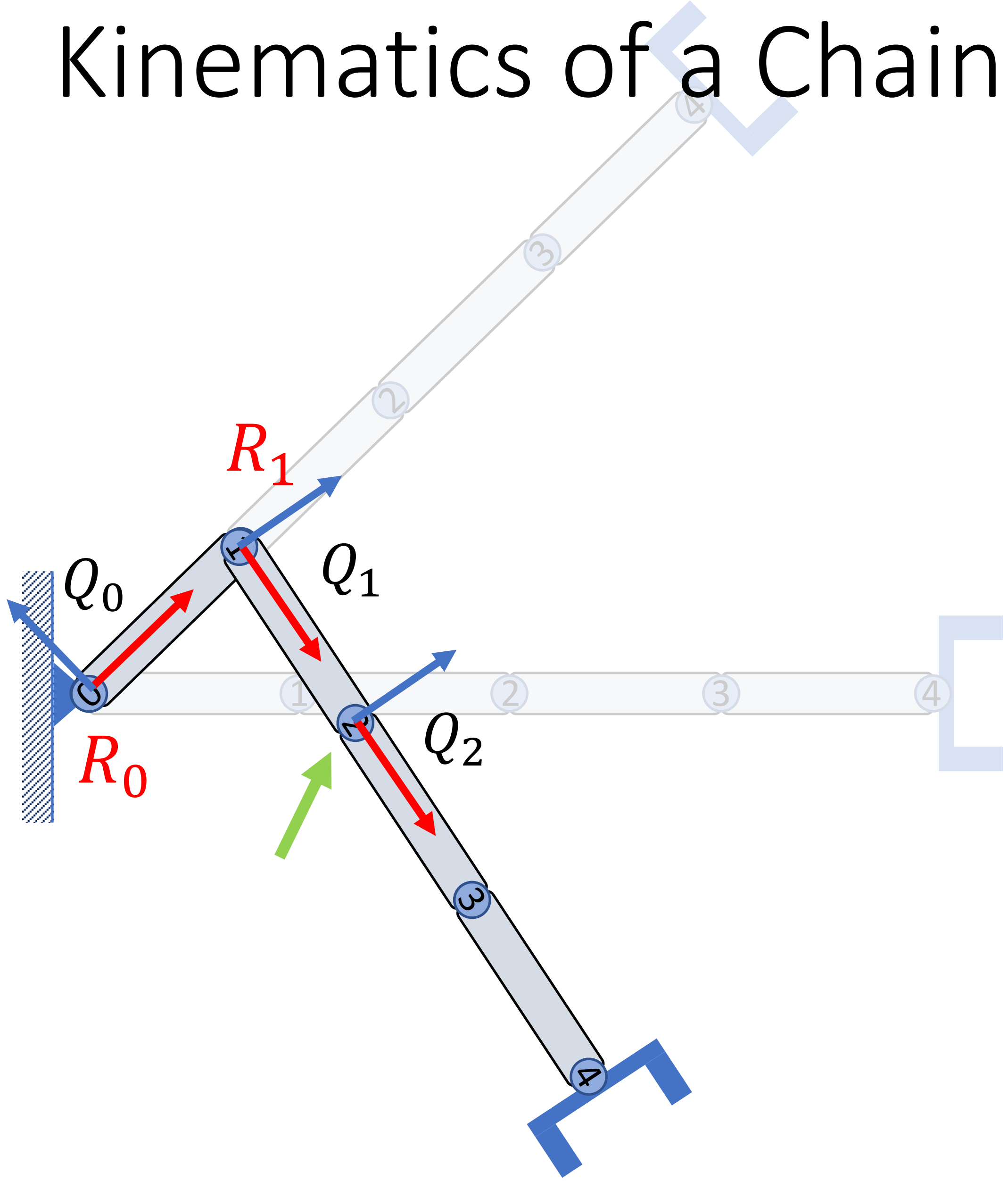
Kinematics of a Chain



$$Q_0 = R_0$$
$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0\boldsymbol{l}_0$$

$$Q_1 = Q_0 R_1$$

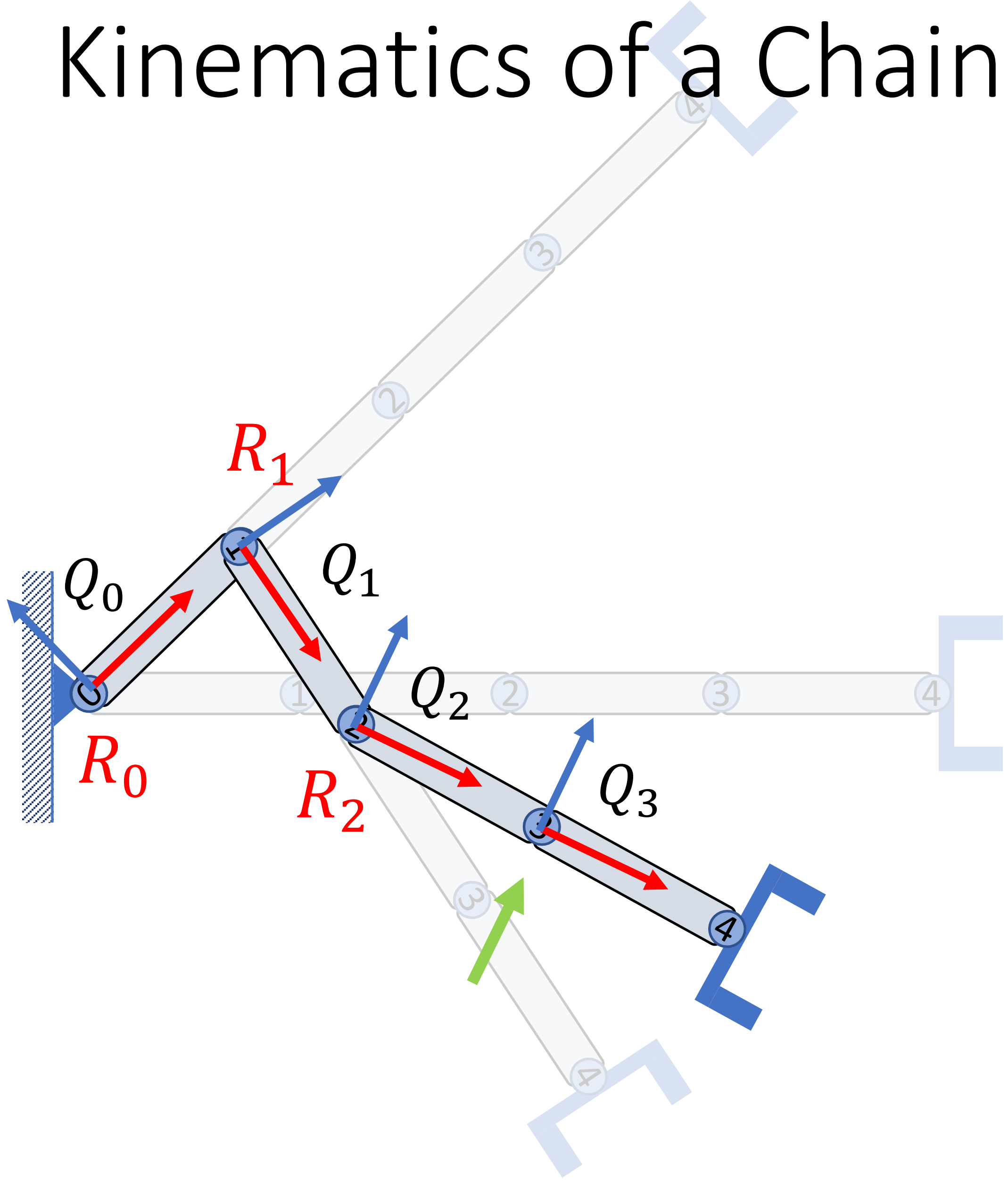
Kinematics of a Chain



$$Q_0 = R_0$$
$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_0$$

$$Q_1 = Q_0 R_1$$
$$\boldsymbol{p}_2 = \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_1$$

Kinematics of a Chain

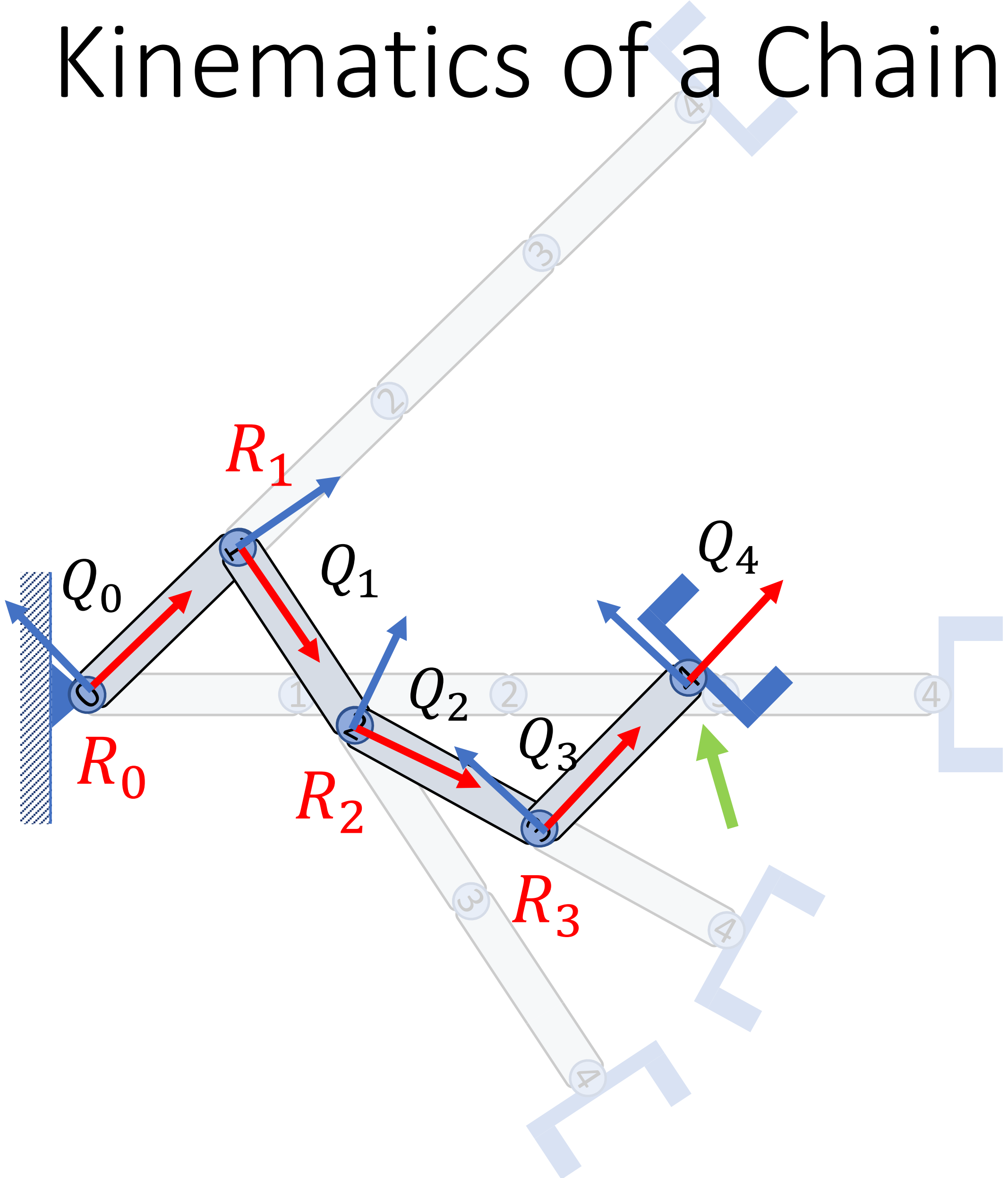


$$Q_0 = R_0$$
$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_0$$

$$Q_1 = Q_0 R_1$$
$$\boldsymbol{p}_2 = \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_1$$

$$Q_2 = Q_1 R_2$$
$$\boldsymbol{p}_3 = \boldsymbol{p}_2 + Q_2 \boldsymbol{l}_2$$

Kinematics of a Chain



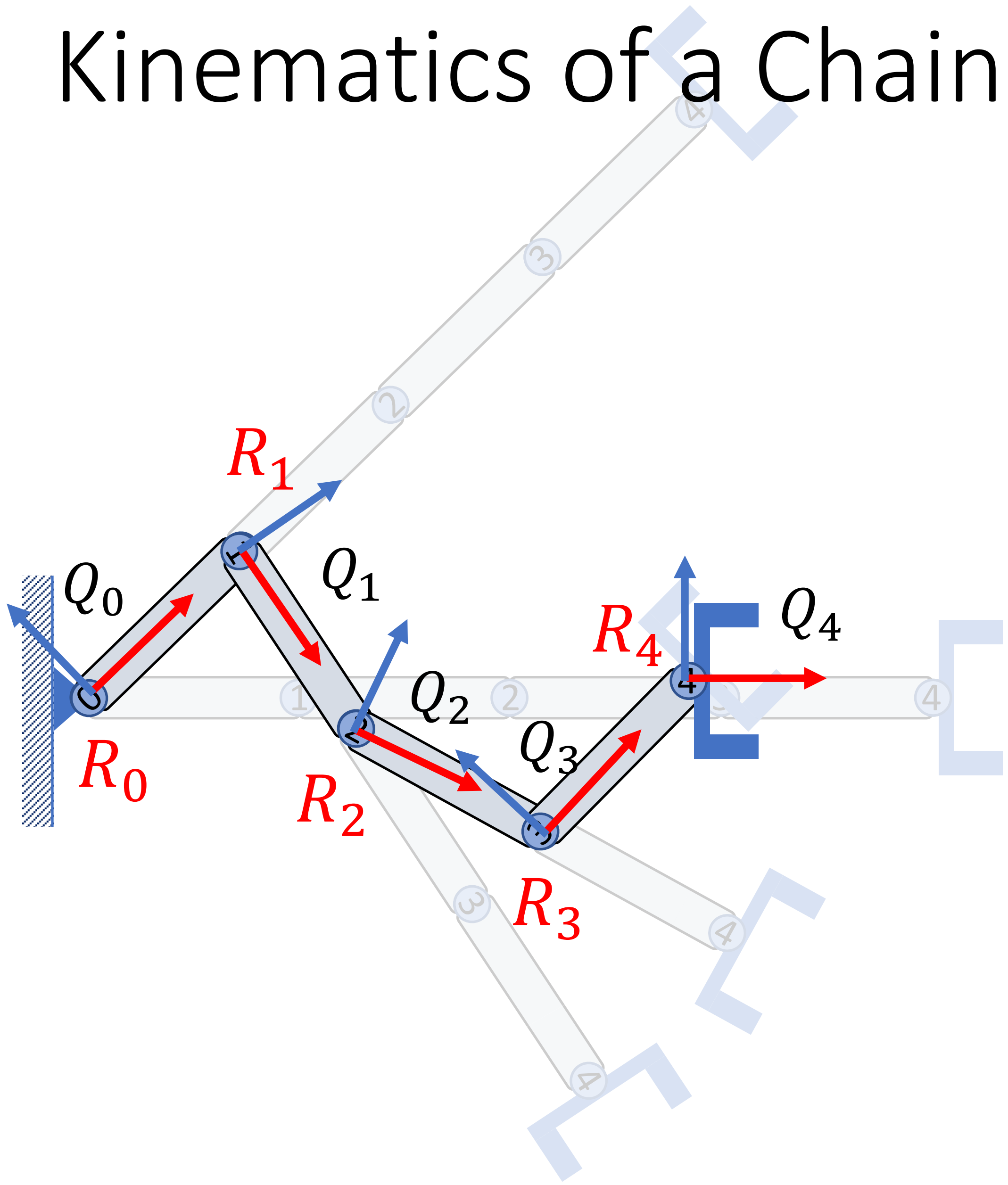
$$Q_0 = R_0$$
$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_0$$

$$Q_1 = Q_0 R_1$$
$$\boldsymbol{p}_2 = \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_1$$

$$Q_2 = Q_1 R_2$$
$$\boldsymbol{p}_3 = \boldsymbol{p}_2 + Q_2 \boldsymbol{l}_2$$

$$Q_3 = Q_2 R_3$$
$$\boldsymbol{p}_4 = \boldsymbol{p}_3 + Q_3 \boldsymbol{l}_3$$

Kinematics of a Chain



$$Q_0 = R_0$$
$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + Q_0 \boldsymbol{l}_0$$

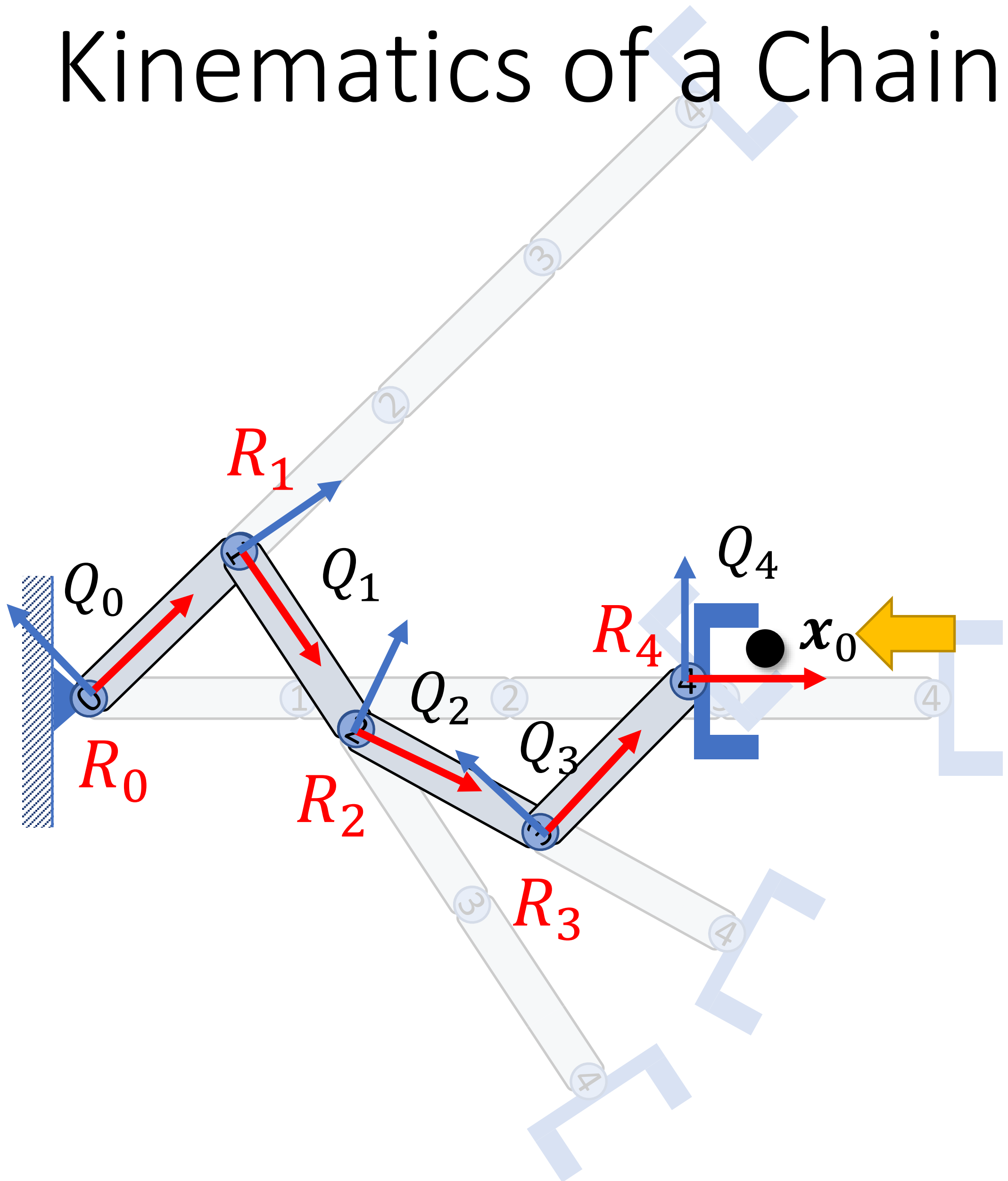
$$Q_1 = Q_0 R_1$$
$$\boldsymbol{p}_2 = \boldsymbol{p}_1 + Q_1 \boldsymbol{l}_1$$

$$Q_2 = Q_1 R_2$$
$$\boldsymbol{p}_3 = \boldsymbol{p}_2 + Q_2 \boldsymbol{l}_2$$

$$Q_3 = Q_2 R_3$$
$$\boldsymbol{p}_4 = \boldsymbol{p}_3 + Q_3 \boldsymbol{l}_3$$

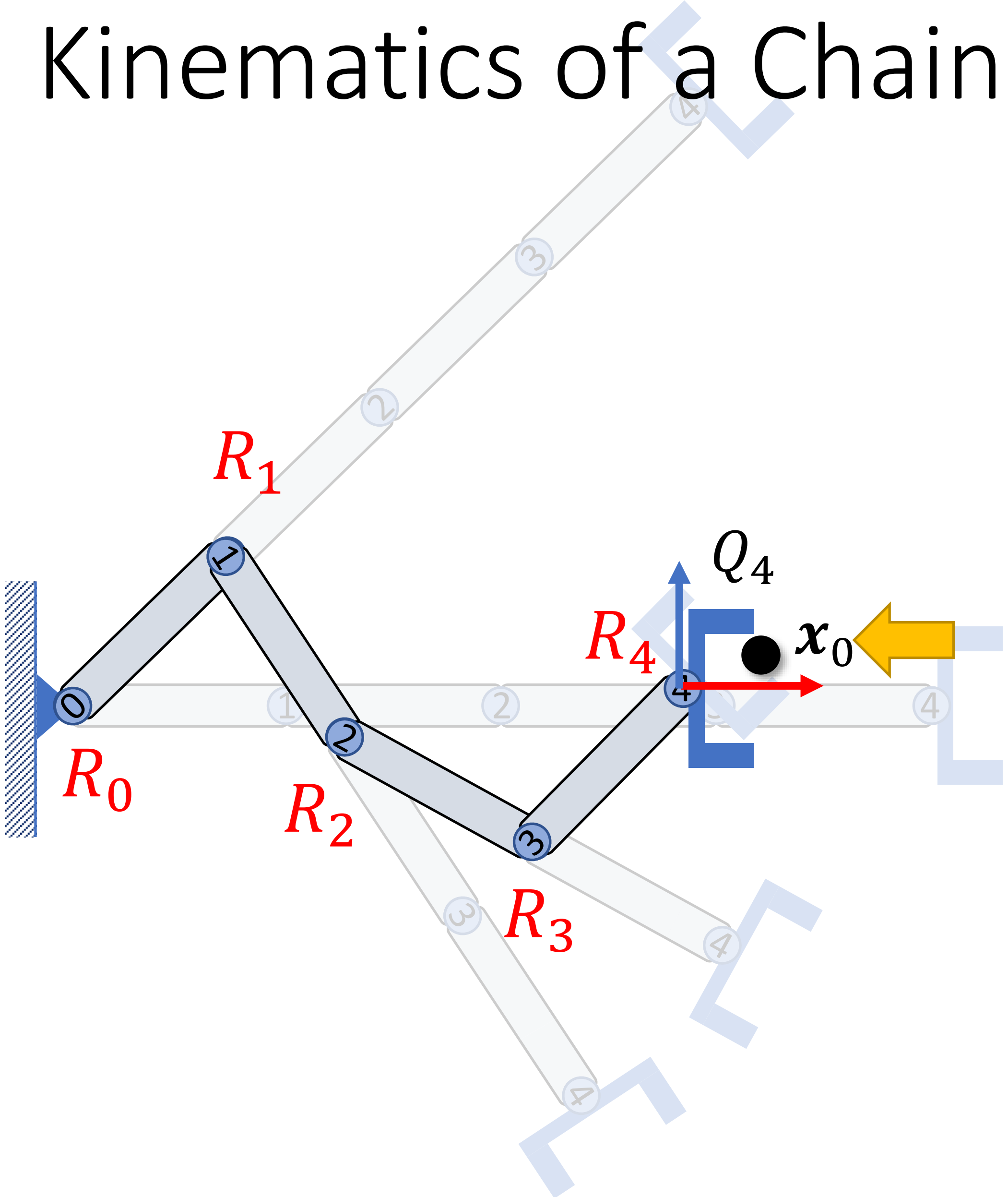
$$Q_4 = Q_3 R_4$$

Kinematics of a Chain



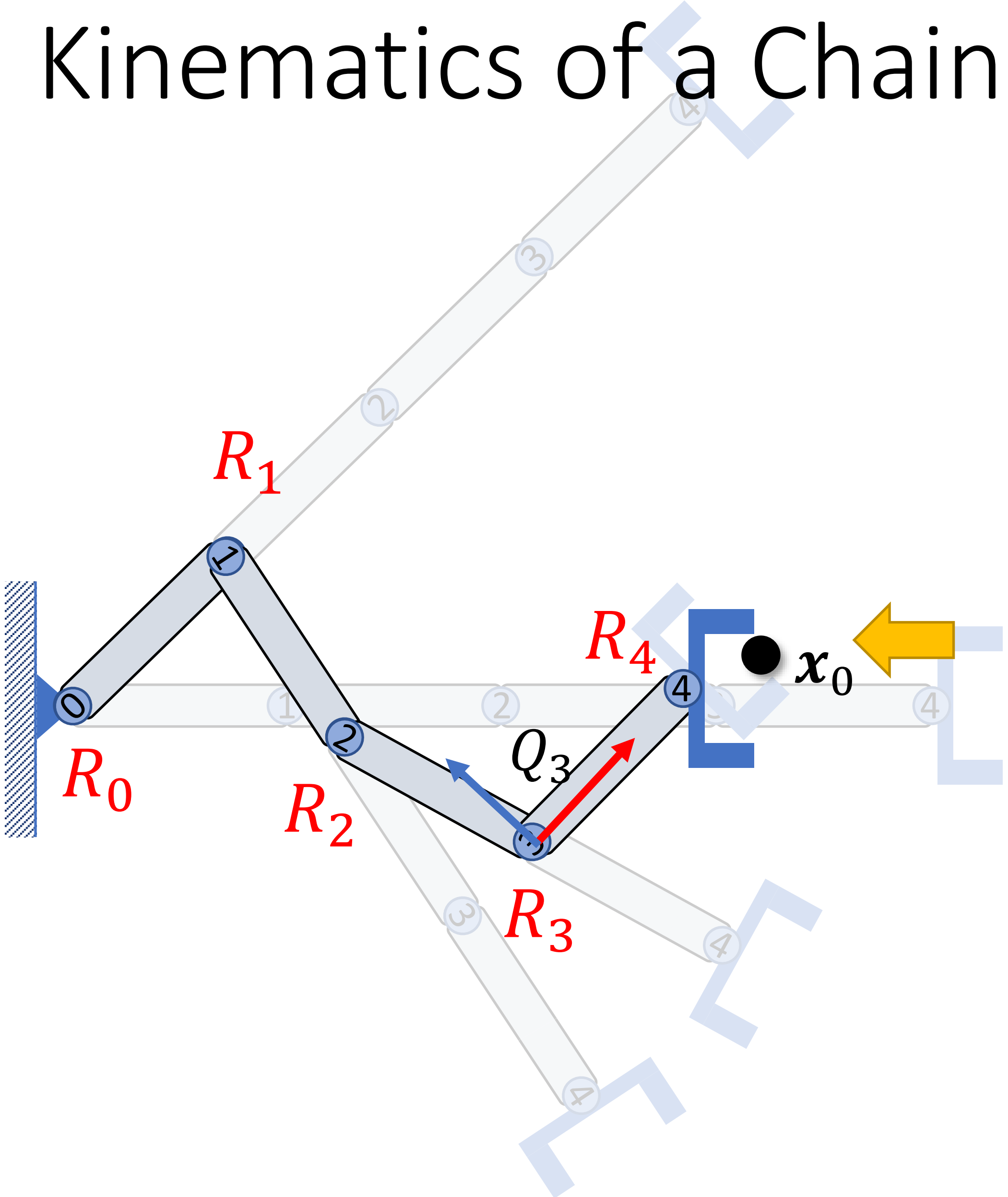
$$\boldsymbol{x} = ???\, \boldsymbol{x}_0$$

Kinematics of a Chain



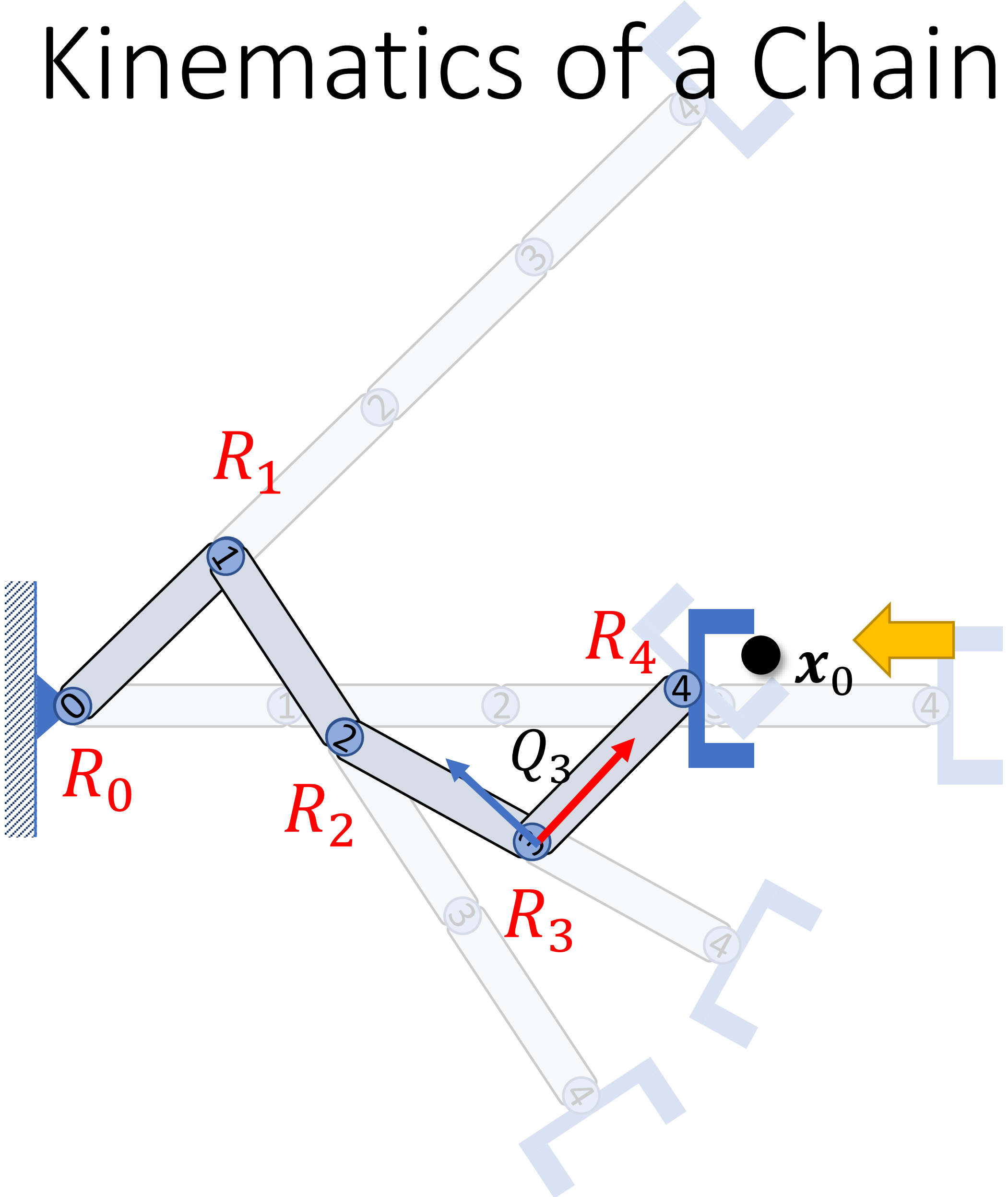
$$\boldsymbol{x} = \boldsymbol{p}_4 + Q_4 \boldsymbol{x}_0$$

Kinematics of a Chain



$$
\begin{aligned}
\boldsymbol{x} &= \boldsymbol{p}_4 + Q_4\boldsymbol{x}_0 \\
&= \boldsymbol{p}_3 + Q_3\boldsymbol{l}_3 + Q_3R_4\boldsymbol{x}_0 \\
&= \boldsymbol{p}_3 + Q_3(\boldsymbol{l}_3 + R_4\boldsymbol{x}_0)
\end{aligned}
$$

Kinematics of a Chain



$$\boldsymbol{x} = \boldsymbol{p}_4 + Q_4\boldsymbol{x}_0$$

$$= \boldsymbol{p}_3 + Q_3\boldsymbol{l}_3 + Q_3R_4\boldsymbol{x}_0$$

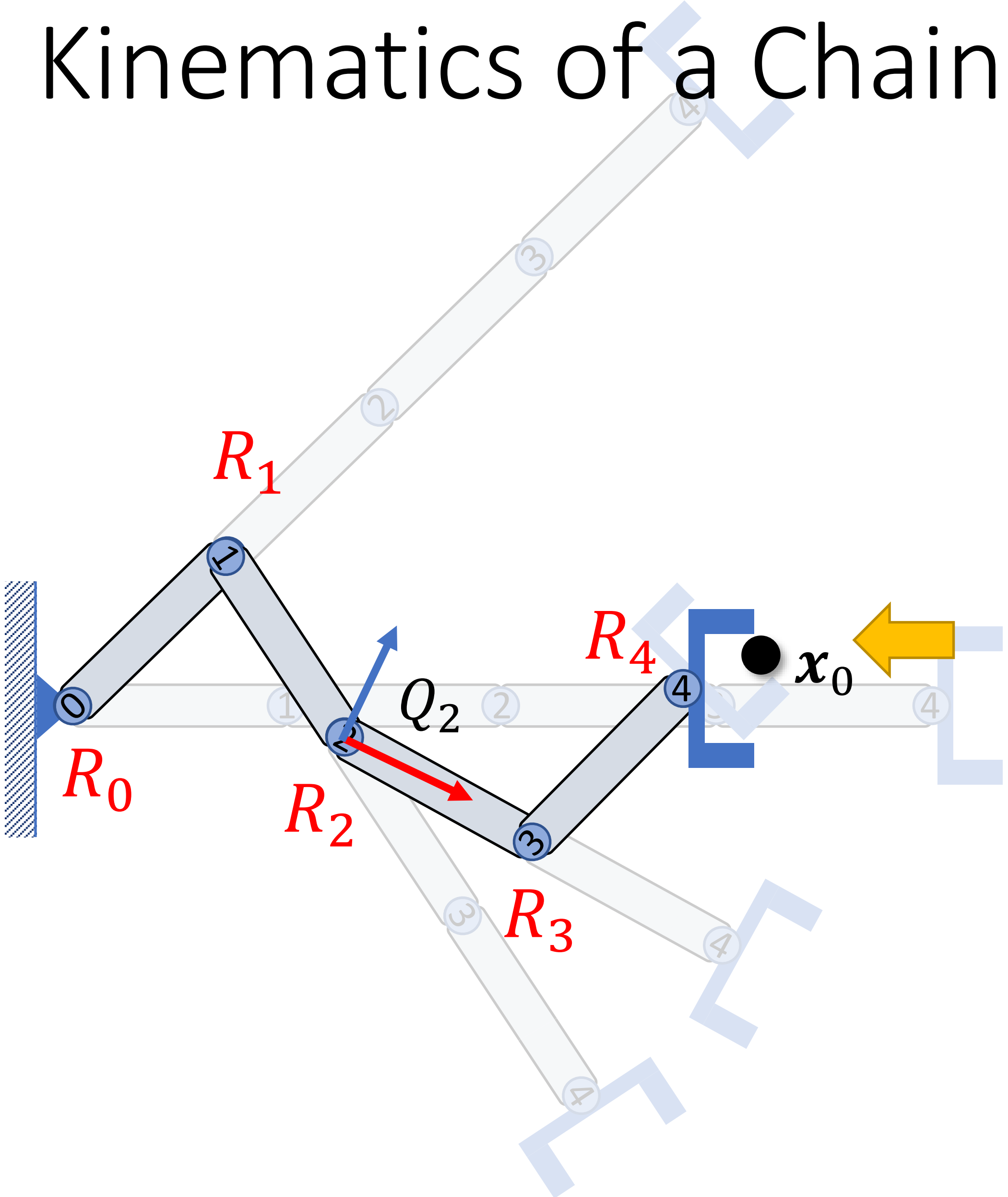
$$= \boldsymbol{p}_3 + Q_3(\boldsymbol{l}_3 + R_4\boldsymbol{x}_0)$$

Local coordinates of $\boldsymbol{x}$ in Q_3

$$\boldsymbol{x}^{Q_3} = Q_3^T(\boldsymbol{x} - \boldsymbol{p}_3)$$

$$= \boldsymbol{l}_3 + R_4\boldsymbol{x}_0$$

Kinematics of a Chain



$$\boldsymbol{x} = \boldsymbol{p}_4 + Q_4\boldsymbol{x}_0$$

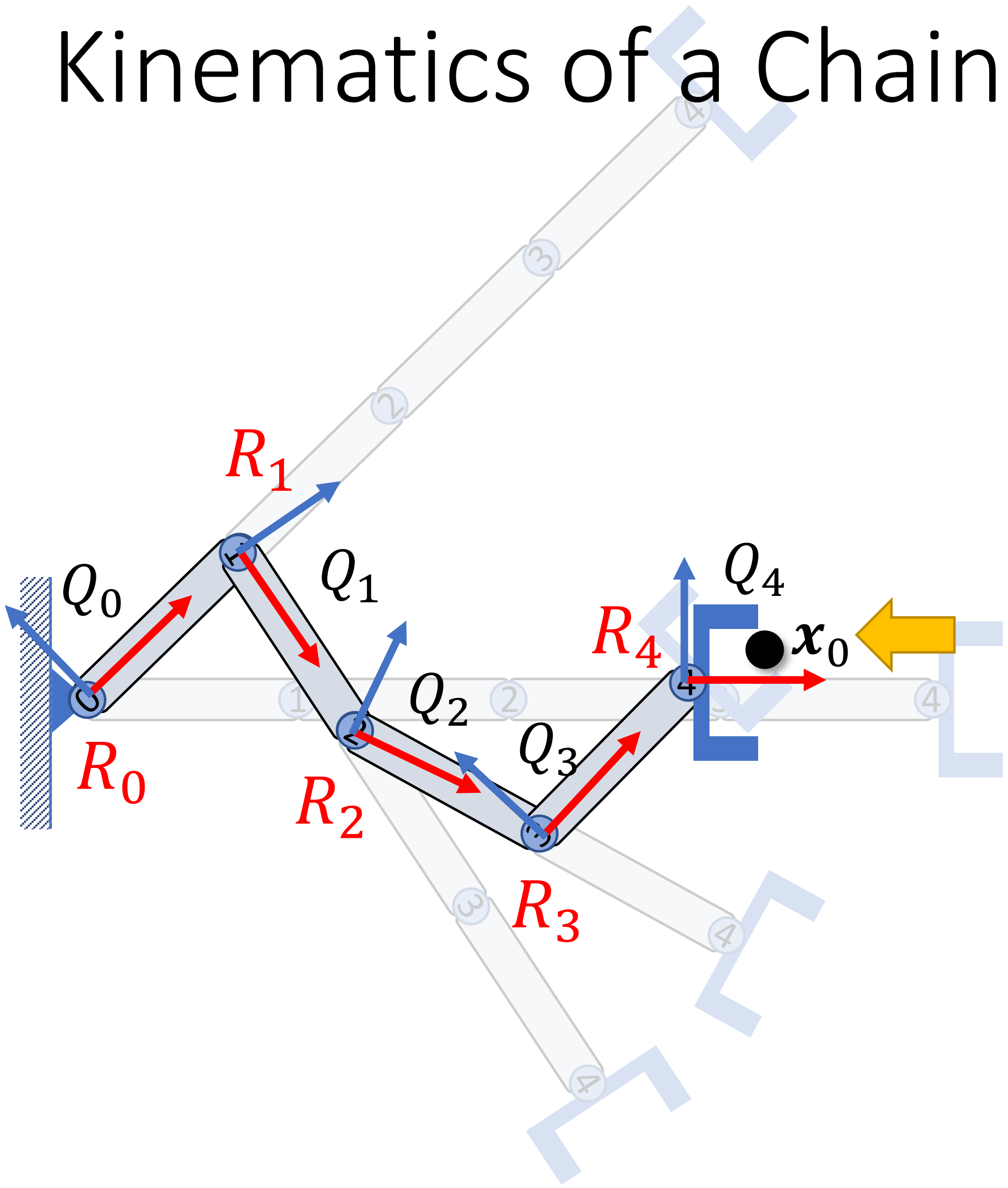
$$= \boldsymbol{p}_2 + Q_2(\boldsymbol{l}_2 + R_3\boldsymbol{l}_3 + R_3R_4\boldsymbol{x}_0)$$

Local coordinates of $\boldsymbol{x}$ in Q_2

$$\boldsymbol{x}^{Q_2} = Q_2^T(\boldsymbol{x} - \boldsymbol{p}_2)$$

$$= \boldsymbol{l}_2 + R_3\boldsymbol{l}_3 + R_3R_4\boldsymbol{x}_0$$

Kinematics of a Chain: Summary



Forward kinematics:

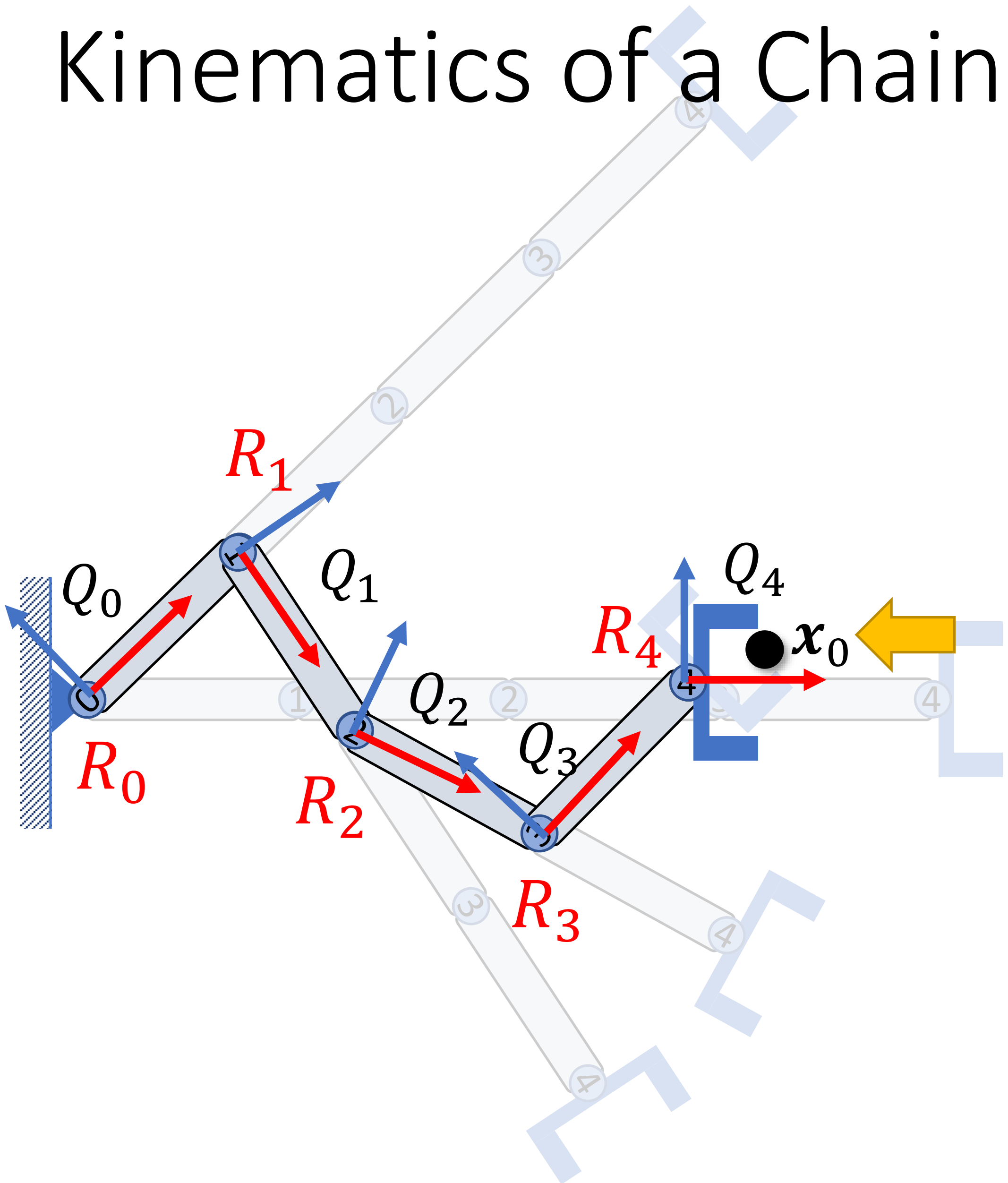
Given the rotations of all joints R_i,
find the coordinates of $\boldsymbol{x_0}$
in the global frame $\boldsymbol{x}$:

for i from the root to the end effector:

$$Q_i = Q_{i-1} R_i$$
$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i + Q_i \boldsymbol{l}_i$$

$$\boldsymbol{x} = \boldsymbol{p}_{\mathrm{E}} + Q_{\mathrm{E}} \boldsymbol{x}_0$$

Kinematics of a Chain: Summary

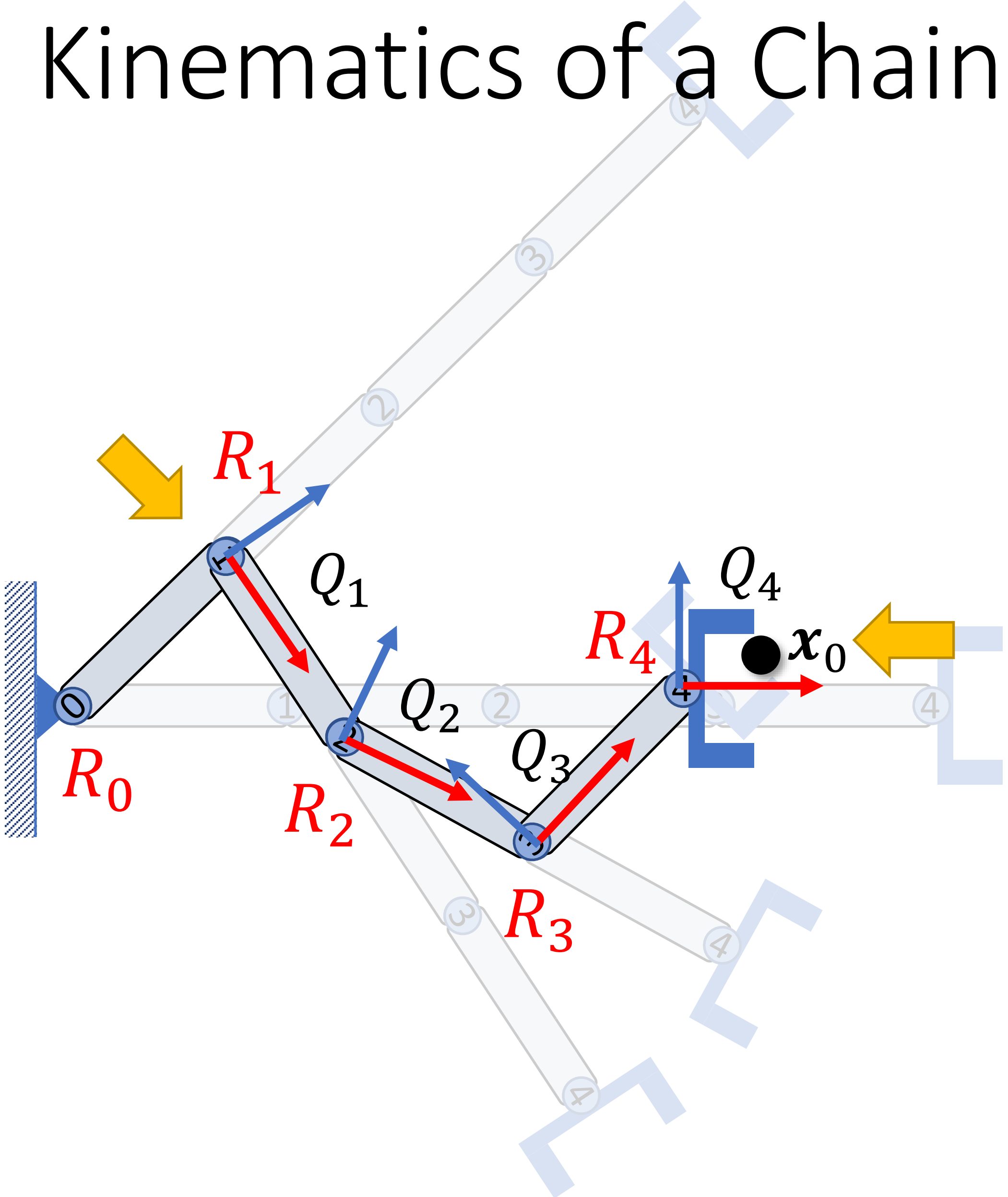


Forward kinematics:

Given the rotations of all joints R_i,
find the coordinates of $\boldsymbol{x_0}$
in the global frame $\boldsymbol{x}$:

$\boldsymbol{x} = \boldsymbol{x}_0$
for i from the end effector to the root
$\boldsymbol{x} = \boldsymbol{l}_{i-1} + R_i \boldsymbol{x}$

Kinematics of a Chain: Summary



Forward kinematics:

Given the rotations of all joints R_i, find the coordinates of $\boldsymbol{x}_0$ relative to the local frame of Q_k:

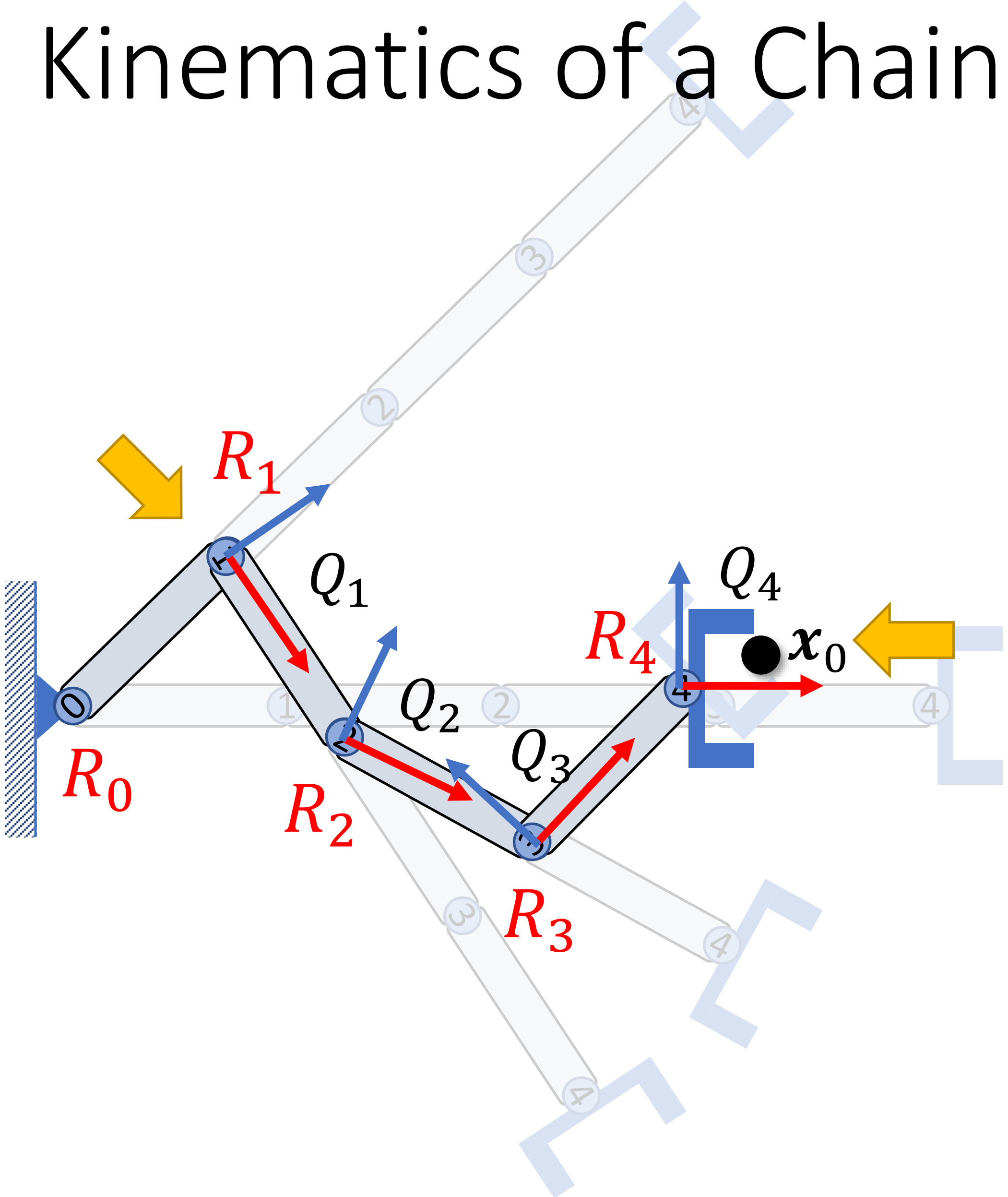
for i from joint $k+1$ to the end effector:

$$Q_i' = Q_{i-1}' R_i \quad //(Q_0' = I)$$

$$\boldsymbol{p}_{i+1}' = \boldsymbol{p}_i' + Q_i' \boldsymbol{l}_i$$

$$\boldsymbol{x} = \boldsymbol{p}_{\mathrm{E}}' + Q_{\mathrm{E}}' \boldsymbol{x}_0$$

Kinematics of a Chain: Summary



Forward kinematics:

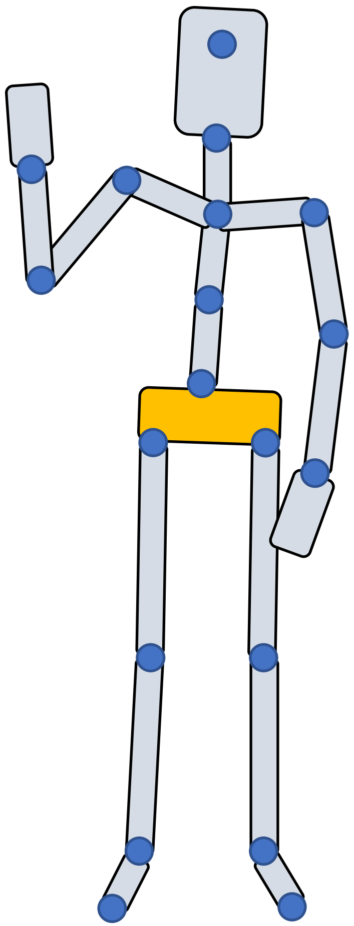
Given the rotations of all joints R_i, find the coordinates of $\boldsymbol{x_0}$ relative to the local frame of Q_k:

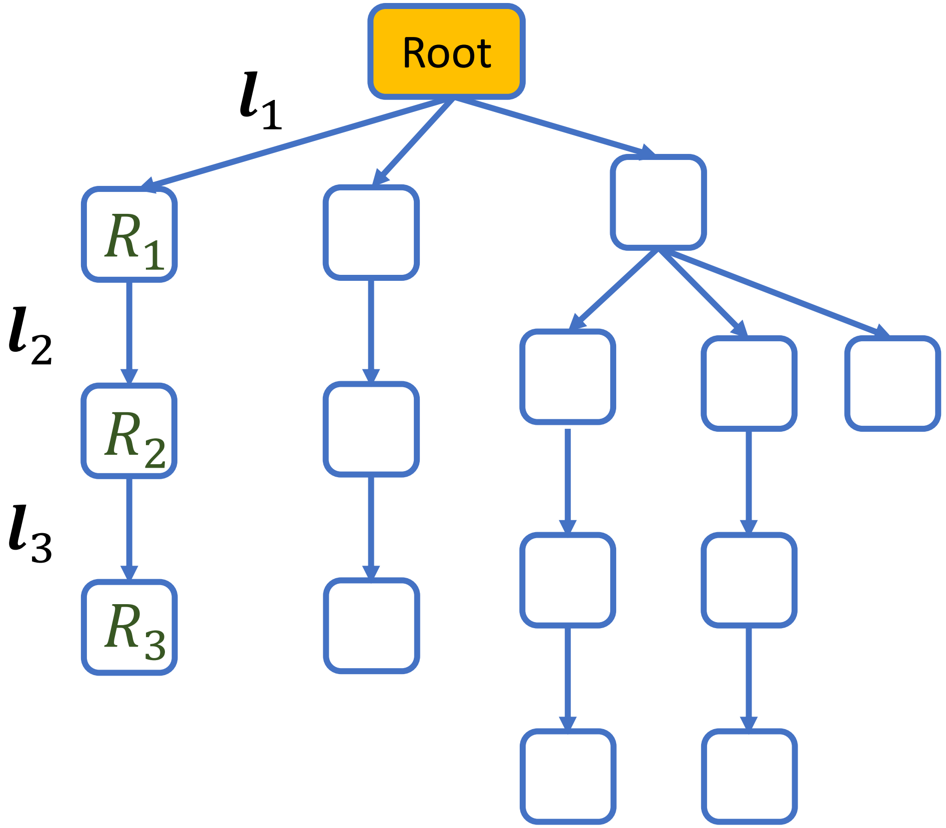
$\boldsymbol{x} = \boldsymbol{x}_0$

for i from the end effector to joint $k+1$

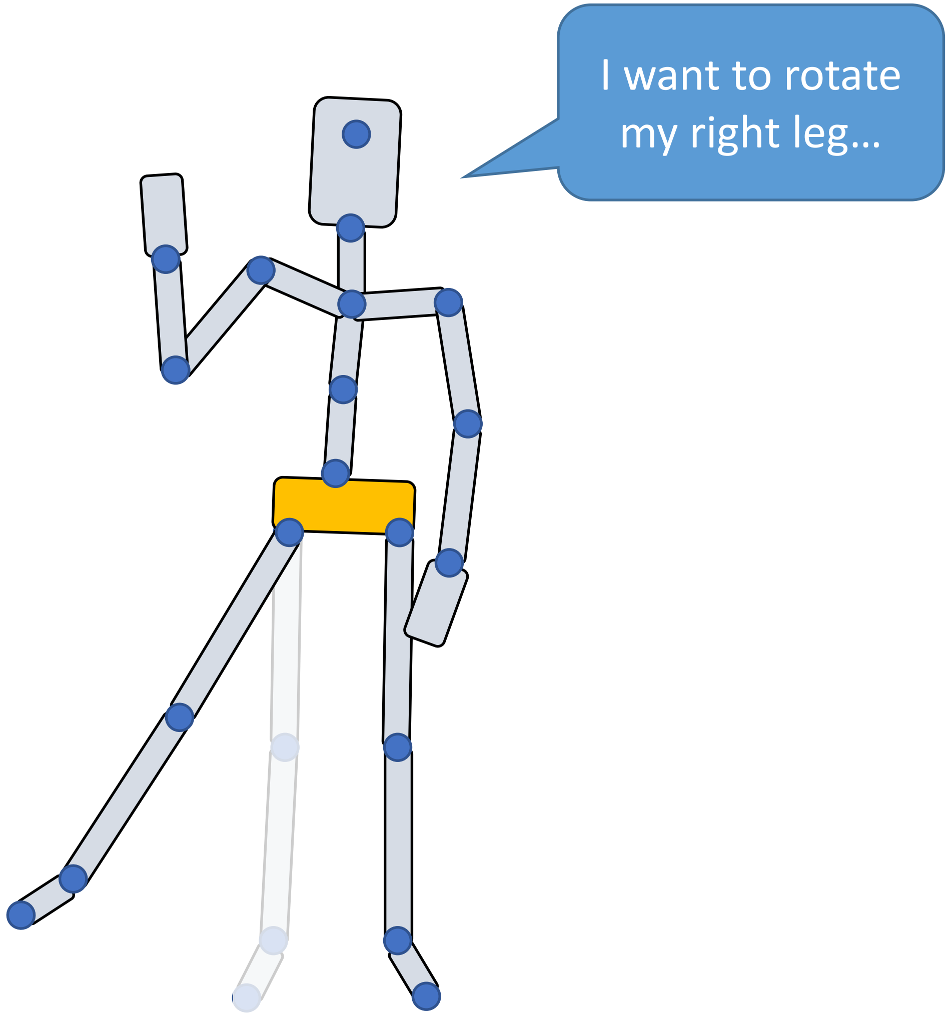
$$\boldsymbol{x} = \boldsymbol{l}_{i-1} + R_i \boldsymbol{x}$$

Kinematics of a Character

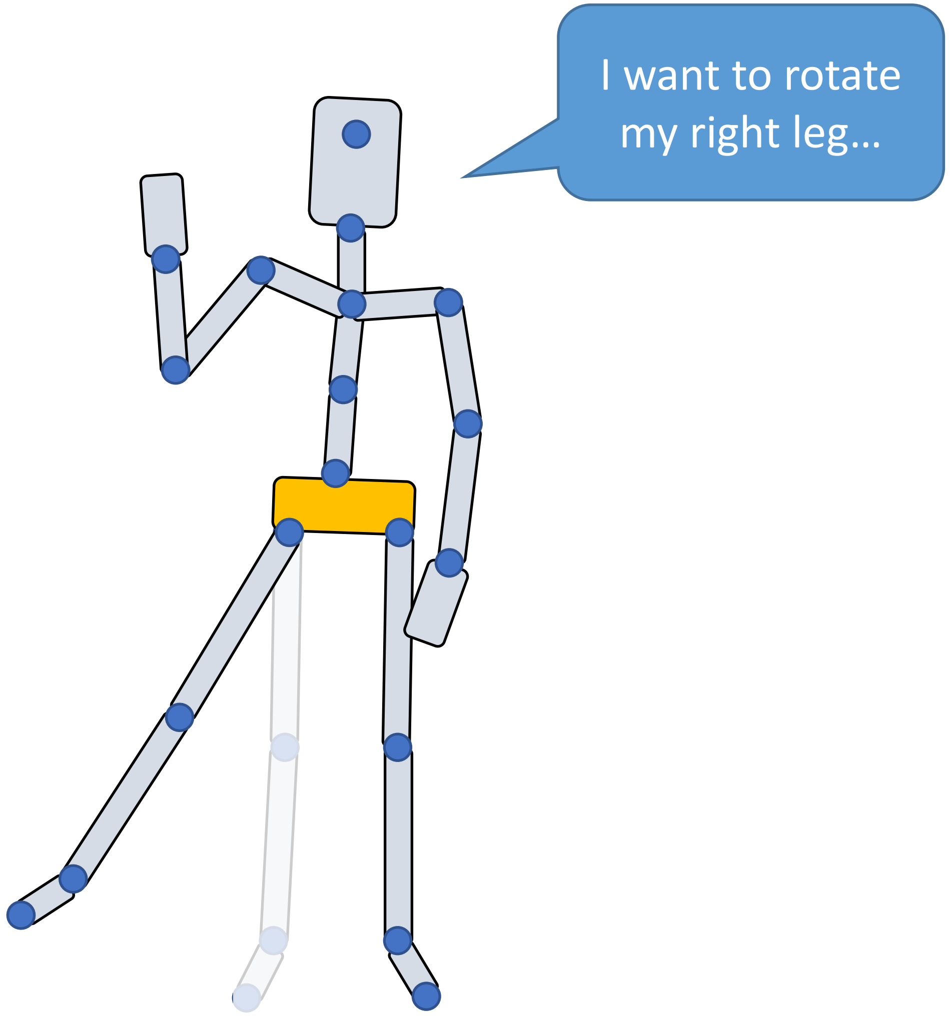


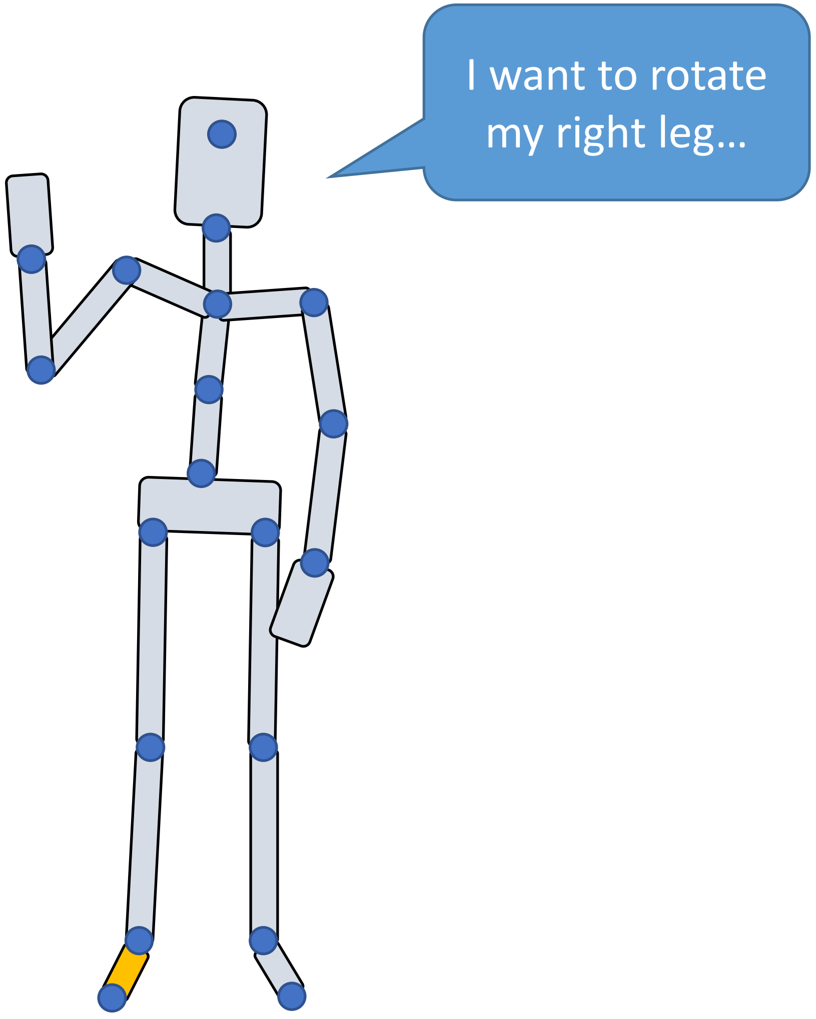


Root Location

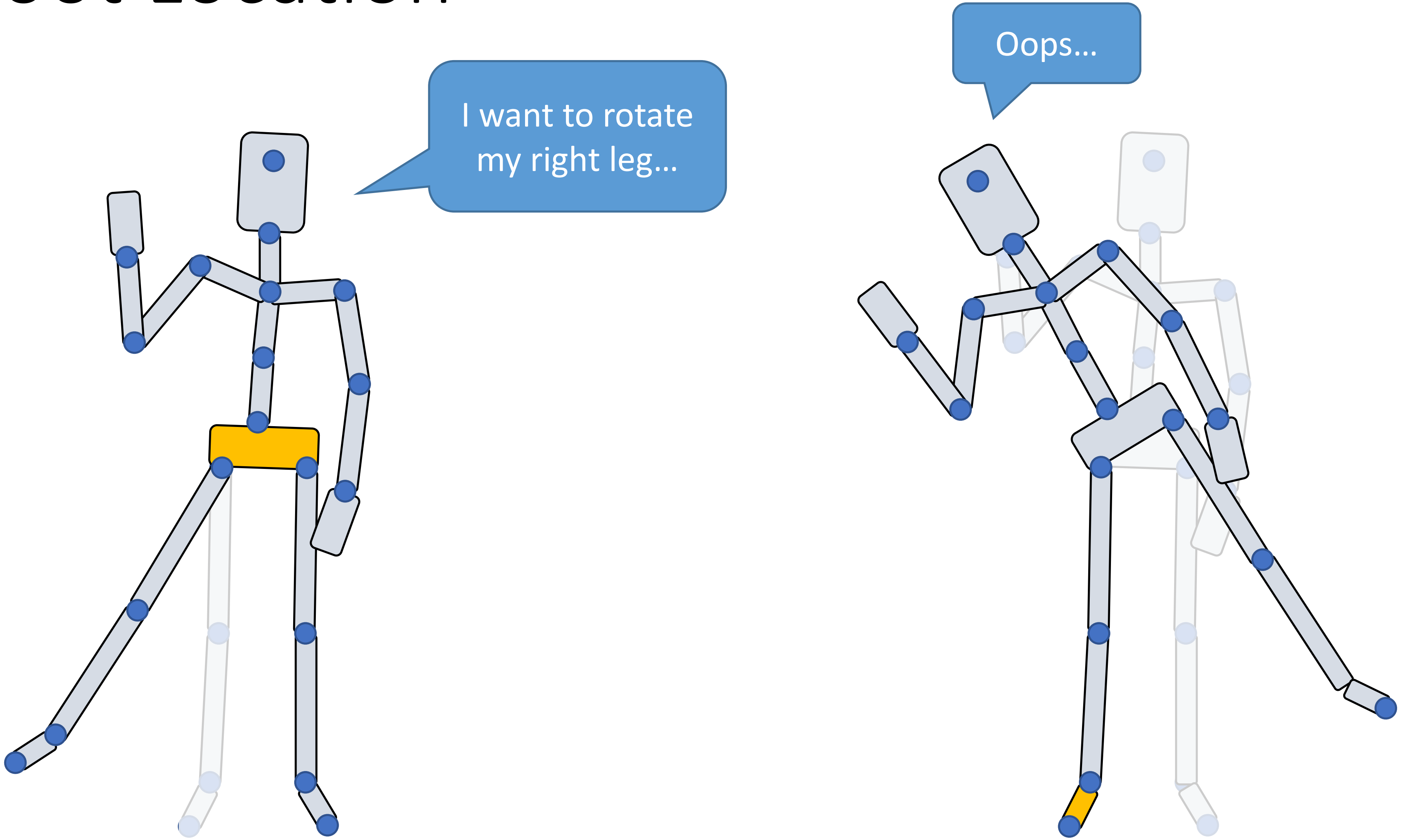


Root Location

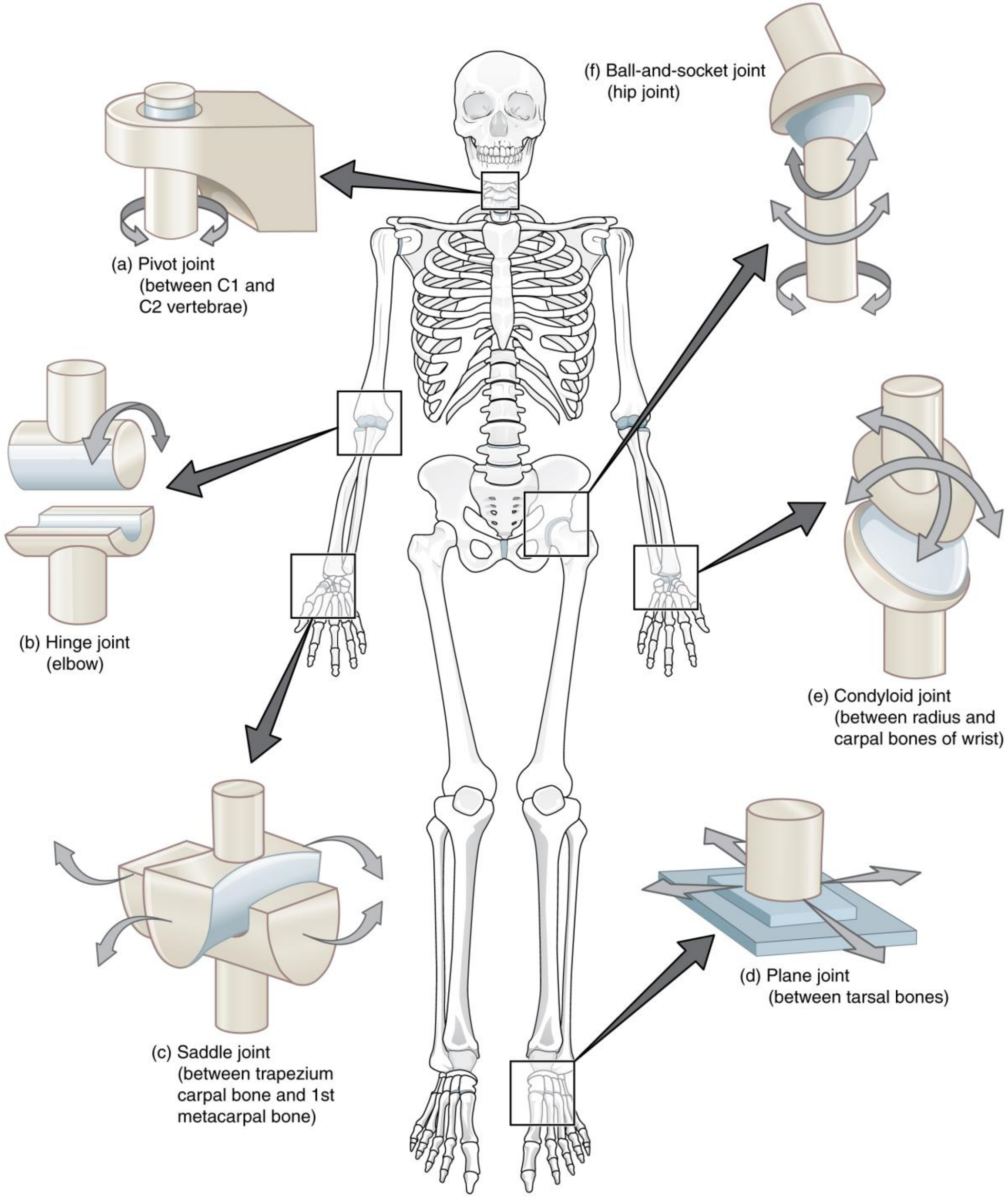




Root Location

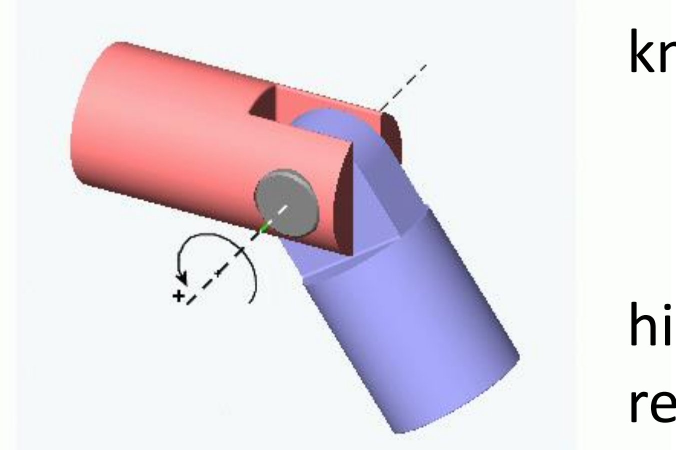


Types of Joints



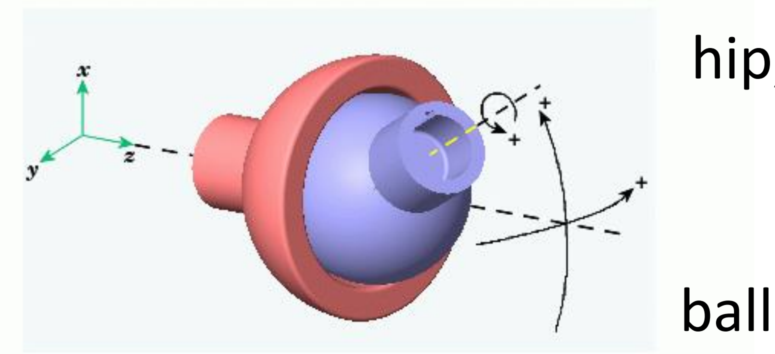
Source: Wikipedia

Types of Joints



knee, elbow

hinge joint
revolute joint

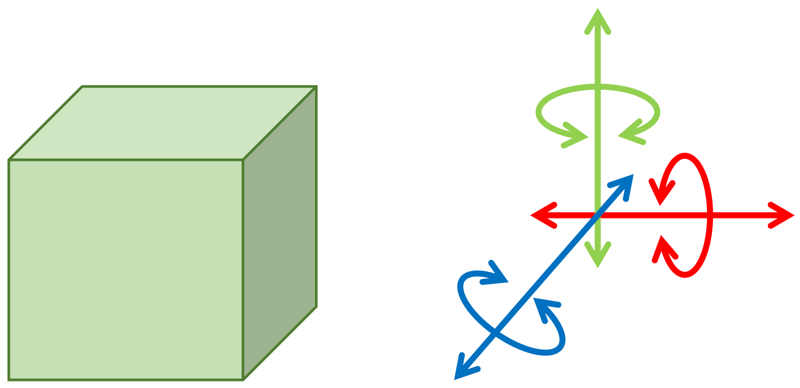


hip, shoulder

ball-and-socket joint

Degrees of Freedom (DoF)

- Number of independent parameters that define the configuration or state of a mechanical system

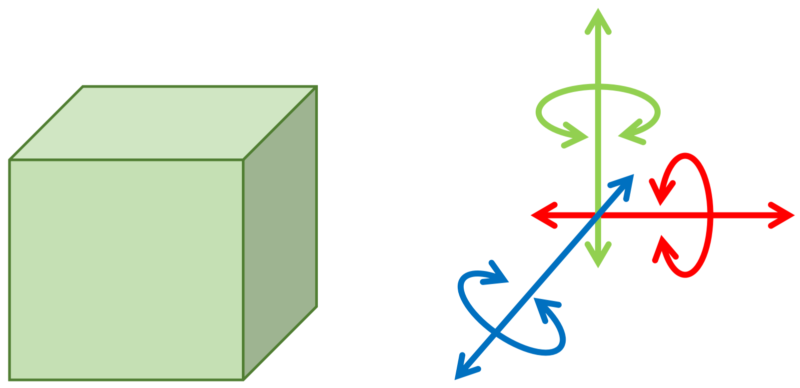


DoF = 6

$$(\boldsymbol{p}, R) \in \mathbb{R}^3 \times SO(3)$$

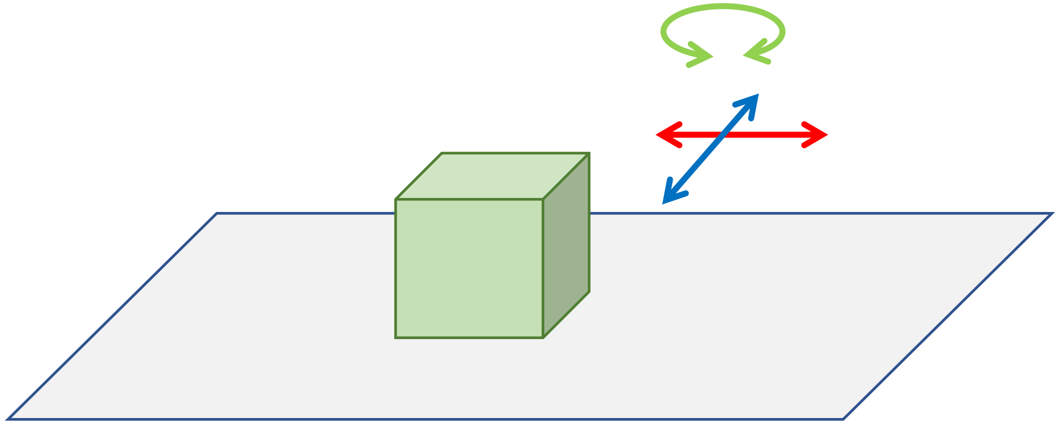
Degrees of Freedom (DoF)

- Number of independent parameters that define the configuration or state of a mechanical system



DoF = 6

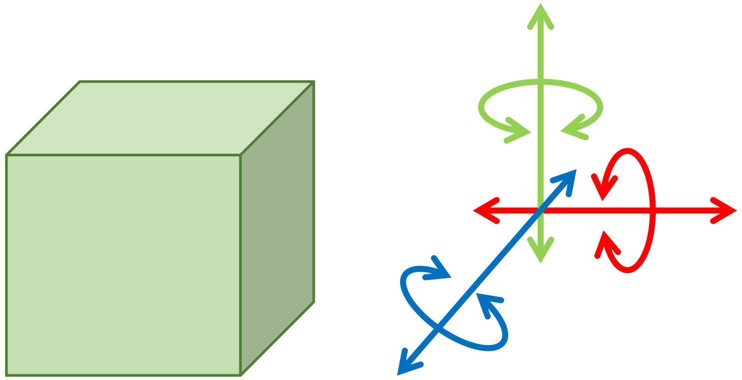
$$(\boldsymbol{p}, R) \in \mathbb{R}^3 \times SO(3)$$



DoF = 3

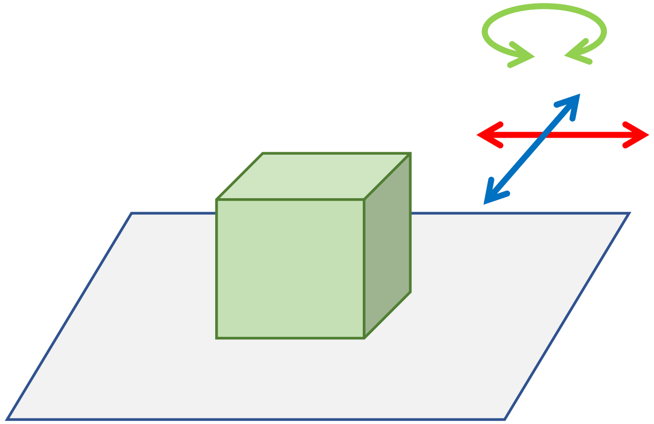
Degrees of Freedom (DoF)

- Number of independent parameters that define the configuration or state of a mechanical system

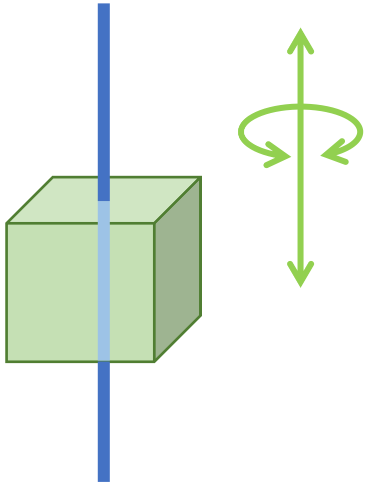


DoF = 6

$$(\boldsymbol{p}, R) \in \mathbb{R}^3 \times SO(3)$$

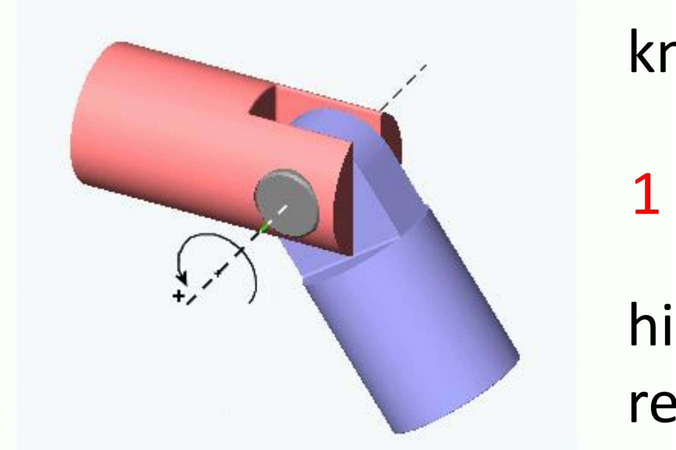


DoF = 3



DoF = 2

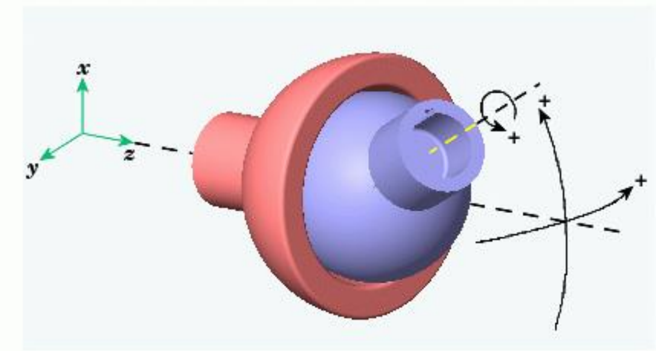
Degrees of Freedom (DoF)



knee, elbow

1 DoF

hinge joint
revolute joint

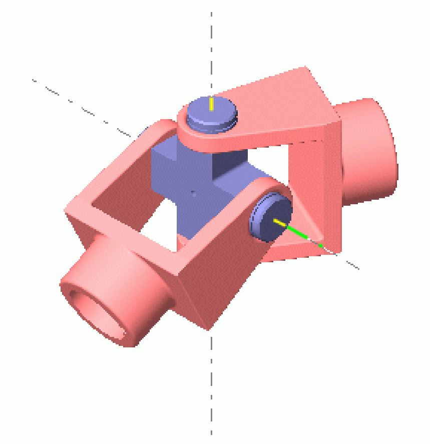


hip, shoulder

3 DoF

ball-and-socket joint

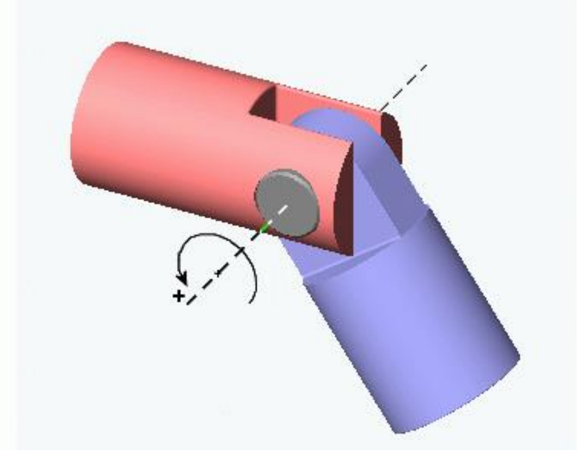
Degrees of Freedom (DoF)



2 DoF

universal joint

Joint Limits



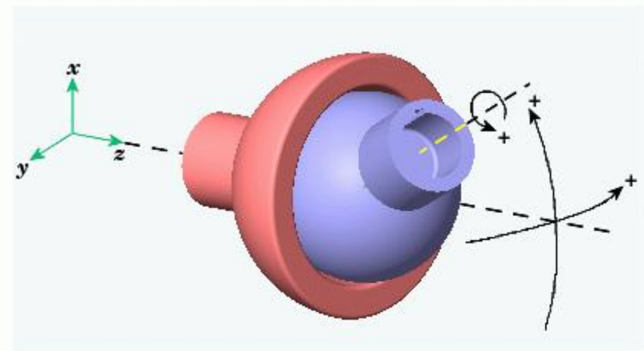
knee, elbow

1 DoF

$\theta_{\min} \leq \theta \leq \theta_{\max}$

hinge joint

revolute joint



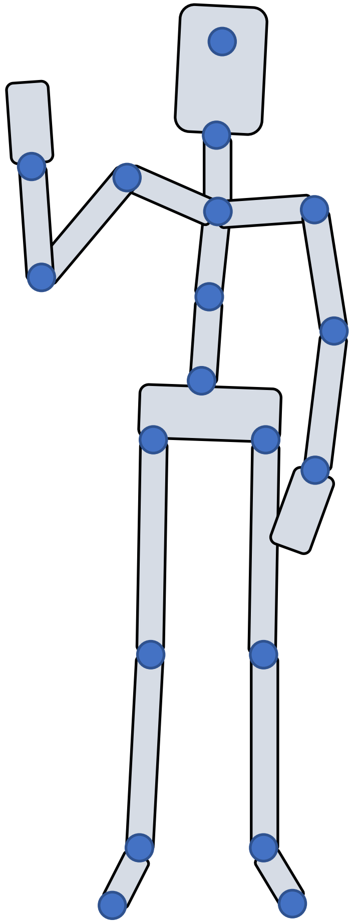
hip, shoulder

3 DoF

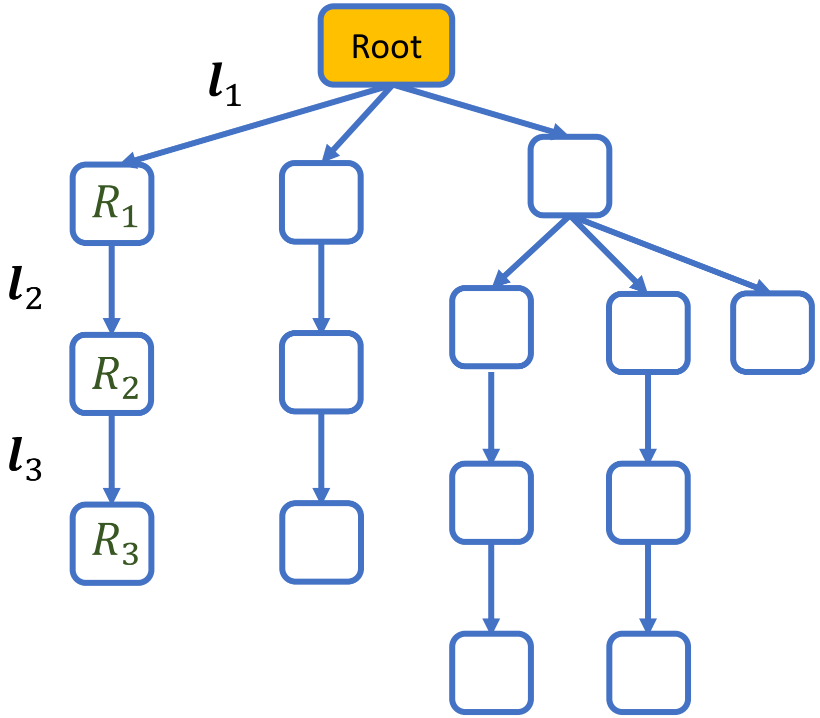
$\boldsymbol{\theta}_{\min} \preccurlyeq \boldsymbol{\theta} \preccurlyeq \boldsymbol{\theta}_{\max}$

ball-and-socket joint

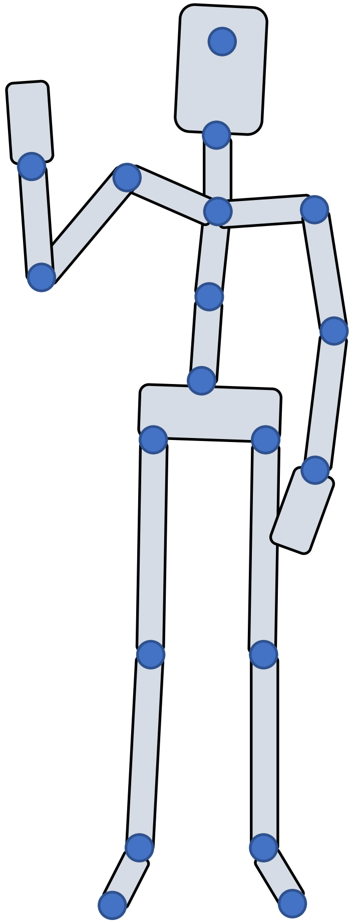
Pose Parameters



$$(\boldsymbol{t}_0, R_0, R_1, R_2, \dots \dots)$$



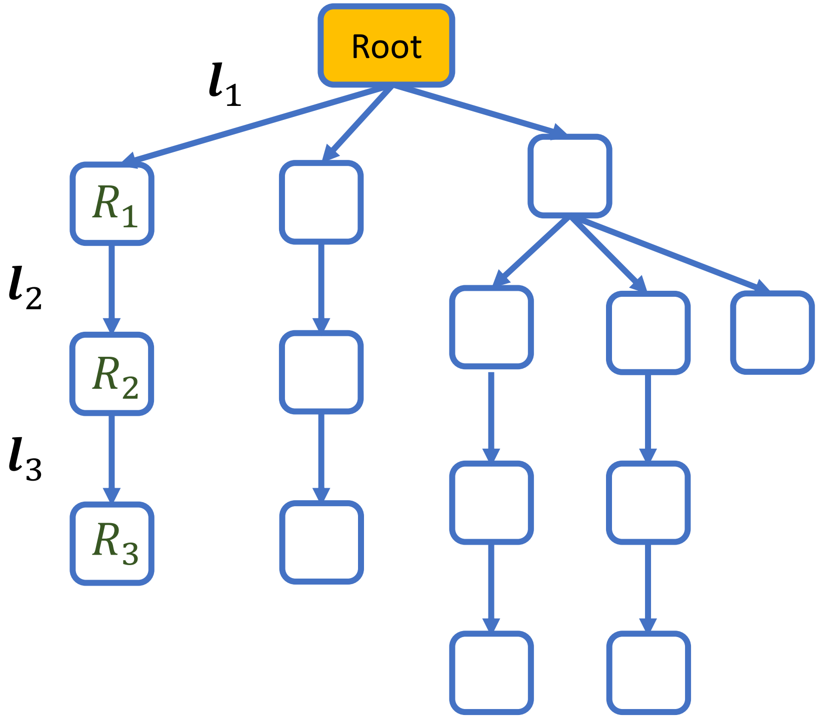
Pose Parameters



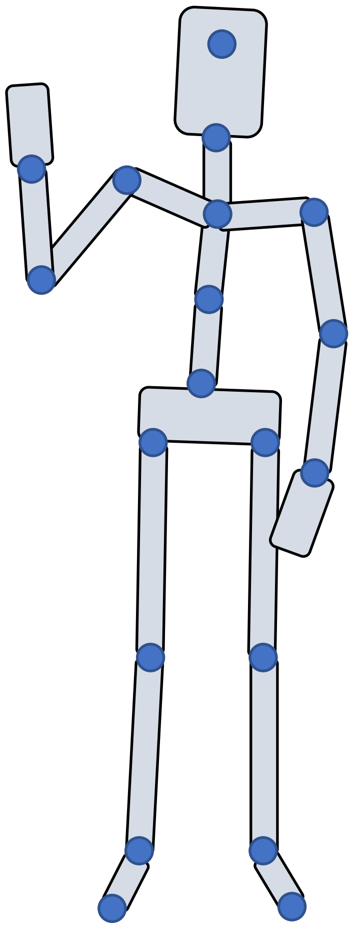
$$(\boldsymbol{t}_0, R_0, R_1, R_2, \dots \dots)$$

root | internal joints

joints are typically in the order that every joint precedes its offspring



Forward Kinematics



$$(\boldsymbol{t}_0, R_0, R_1, R_2, \dots \dots)$$

root | internal joints

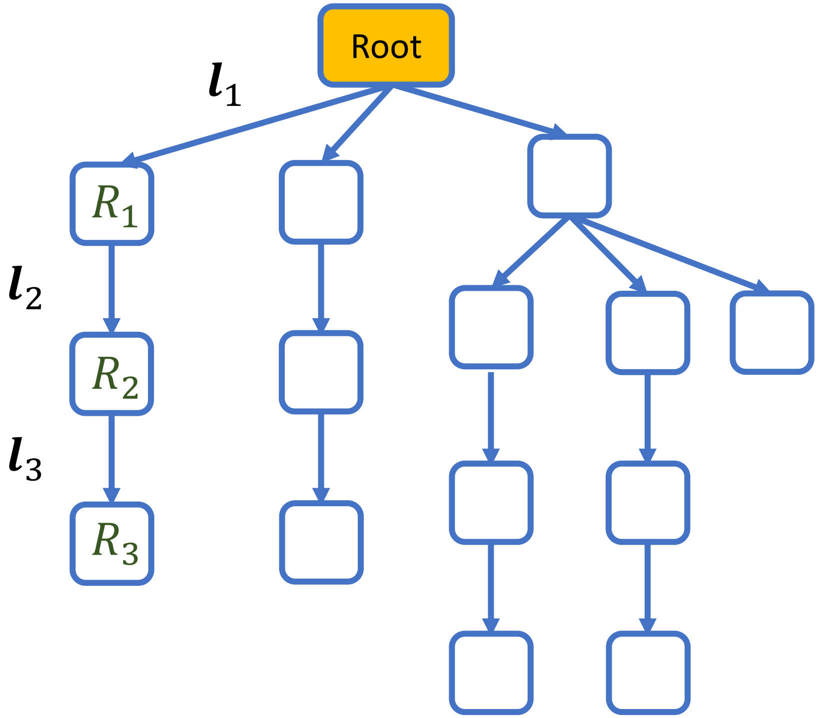
joints are typically in the order that every joint precedes its offspring

for i in joint_list:

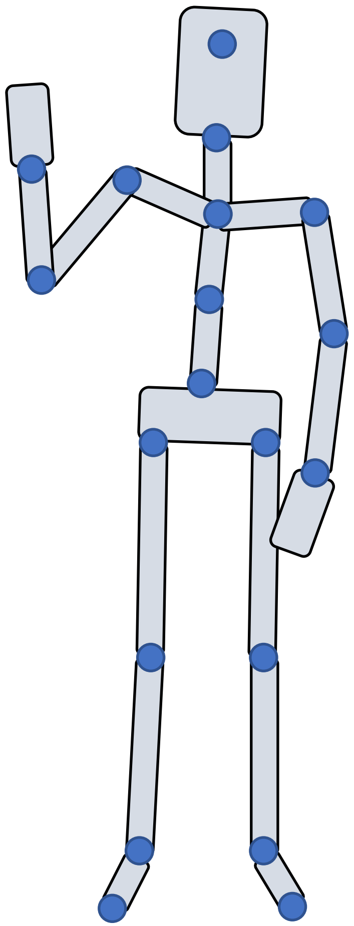
  p_i = i's parent joint

  $Q_i = Q_{p_i} R_i$

  $\boldsymbol{x}_i = \boldsymbol{x}_{p_i} + Q_{pi} \boldsymbol{l}_i$



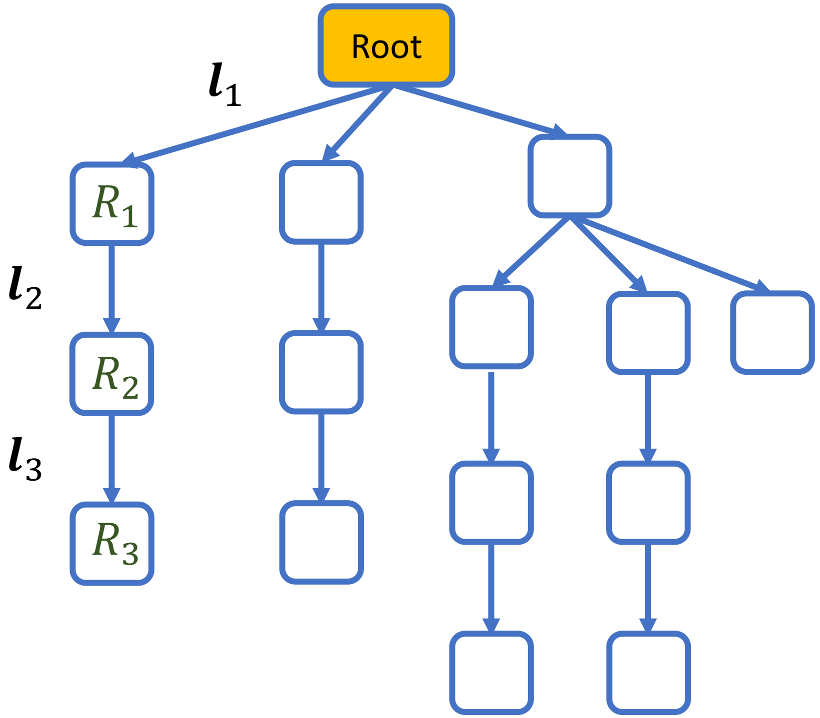
Forward Kinematics



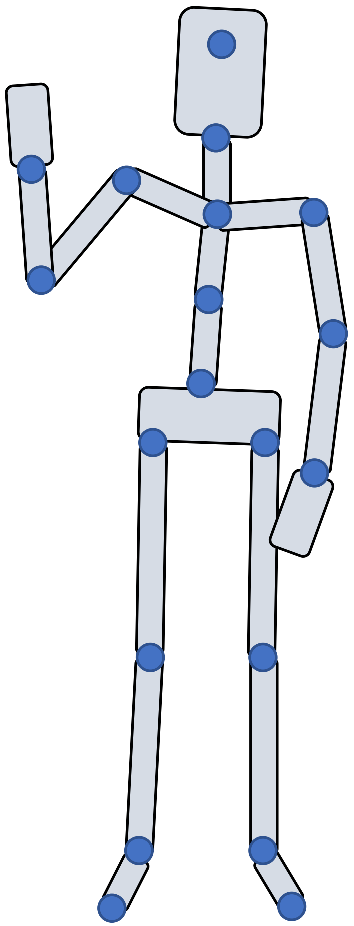
$$(\boldsymbol{t}_0, R_0, R_1, R_2, \dots \dots)$$

root | internal joints

Q1: if we know the orientations of all the joints Q_i, how to compute joint rotations?



Forward Kinematics

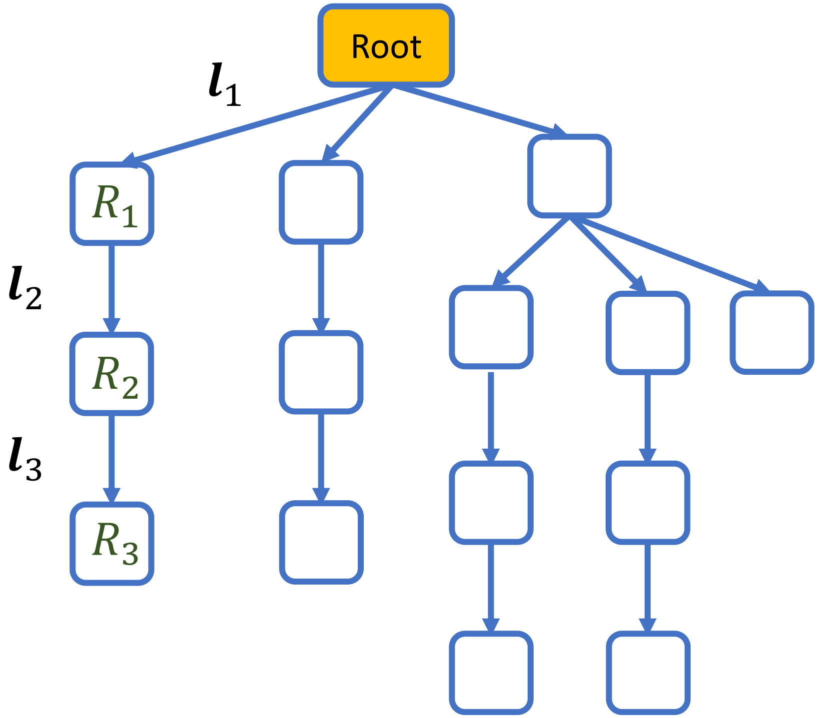


$$(\boldsymbol{t}_0, R_0, R_1, R_2, \ldots \ldots)$$

root | internal joints

Q1: if we know the orientations of all the joints Q_i, how to compute joint rotations?

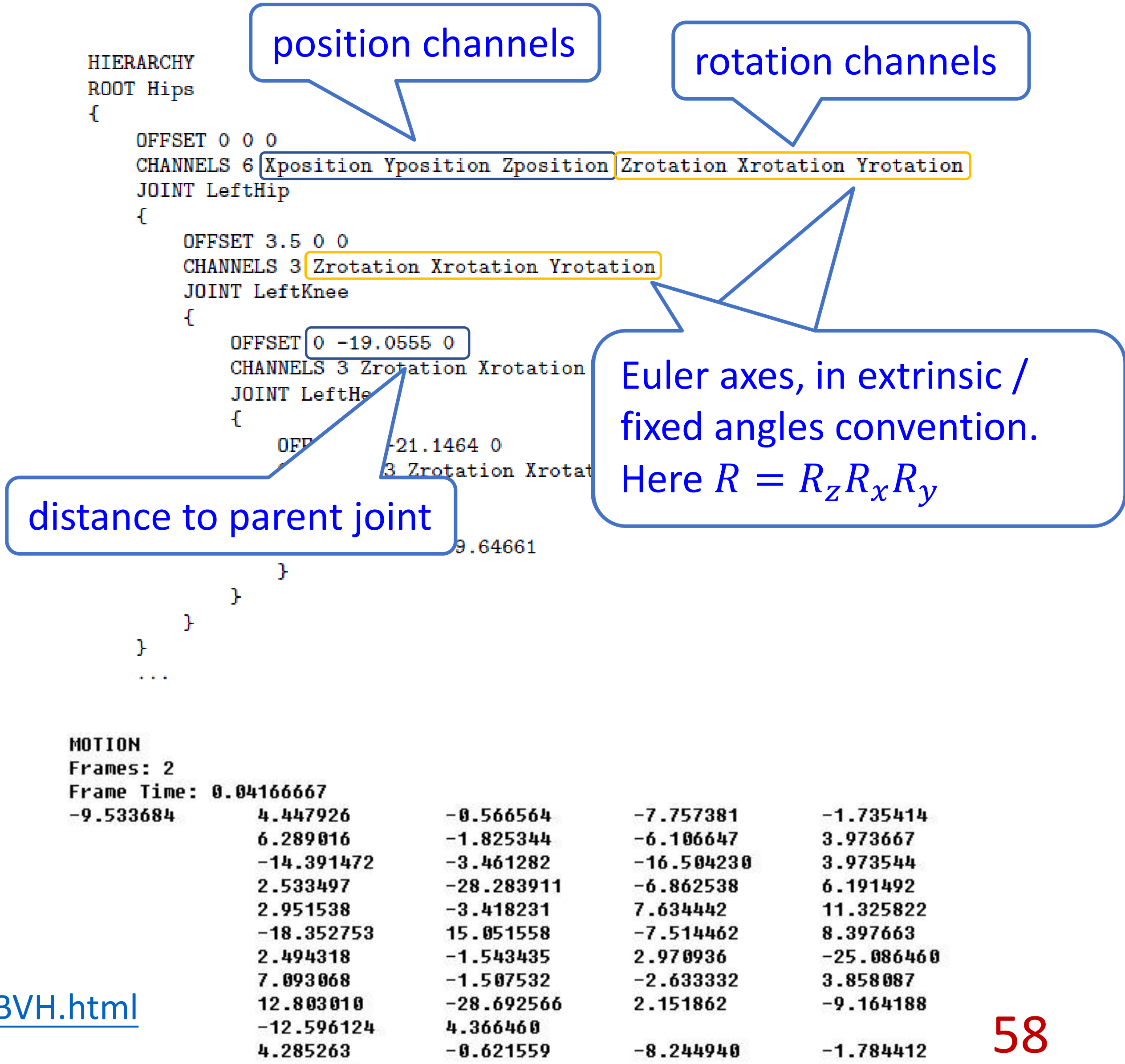
Q2: how should we allow stretchable bones?



Example: motion data in a file

- BVH files
  - One of the most-used file format for motion data
  - View in blender, FBX review, Motion Builder, etc.
  - Text-based, easy to read and edit

- Format
  - HIERARCHY: defining **T-pose** of the character
  - MOTION: root position and Euler angles of each joints



See: https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

Inverse Kinematics

DOI: 10.1111/cgf.13310

COMPUTER GRAPHICS *forum*
Volume 37 (2018), number 6 pp. 35–58

Inverse Kinematics Techniques in Computer Graphics: A Survey

A. Aristidou[1,2], J. Lasenby[3], Y. Chrysanthou[1] and A. Shamir[2]

[1]Department of Computer Science, University of Cyprus, Cyprus
a.aristidou@ieee.org, yiorgos@cs.ucy.ac.cy
[2]The Interdisciplinary Center, Herzliya, Israel
arik.shamir@gmail.com
[3]Department of Engineering, University of Cambridge, United Kingdom
jl221@cam.ac.uk

Abstract
Inverse kinematics (IK) is the use of kinematic equations to determine the joint parameters of a manipulator so that the end effector moves to a desired position: IK can be applied in many areas, including robotics, engineering, computer graphics and video games. In this survey, we present a comprehensive review of the IK problem and the solutions developed over the years from the computer graphics point of view. The paper starts with the definition of forward and IK, their mathematical formulations and explains how to distinguish the unsolvable cases, indicating when a solution is available. The IK literature in this report is divided into four main categories: the analytical, *the* numerical, *the* data-driven *and the* hybrid *methods. A timeline illustrating key methods is presented, explaining how the IK approaches have progressed over the years. The most popular IK methods are discussed with regard to their performance, computational cost and the smoothness of their resulting postures, while we suggest which IK family of solvers is best suited for particular problems. Finally, we indicate the limitations of the current IK methodologies and propose future research directions.*

Keywords: inverse kinematics, motion capture, biomechanical constraints

ACM CCS: General and reference—Surveys and overviews; Computing methodologies—Animation

1. Introduction

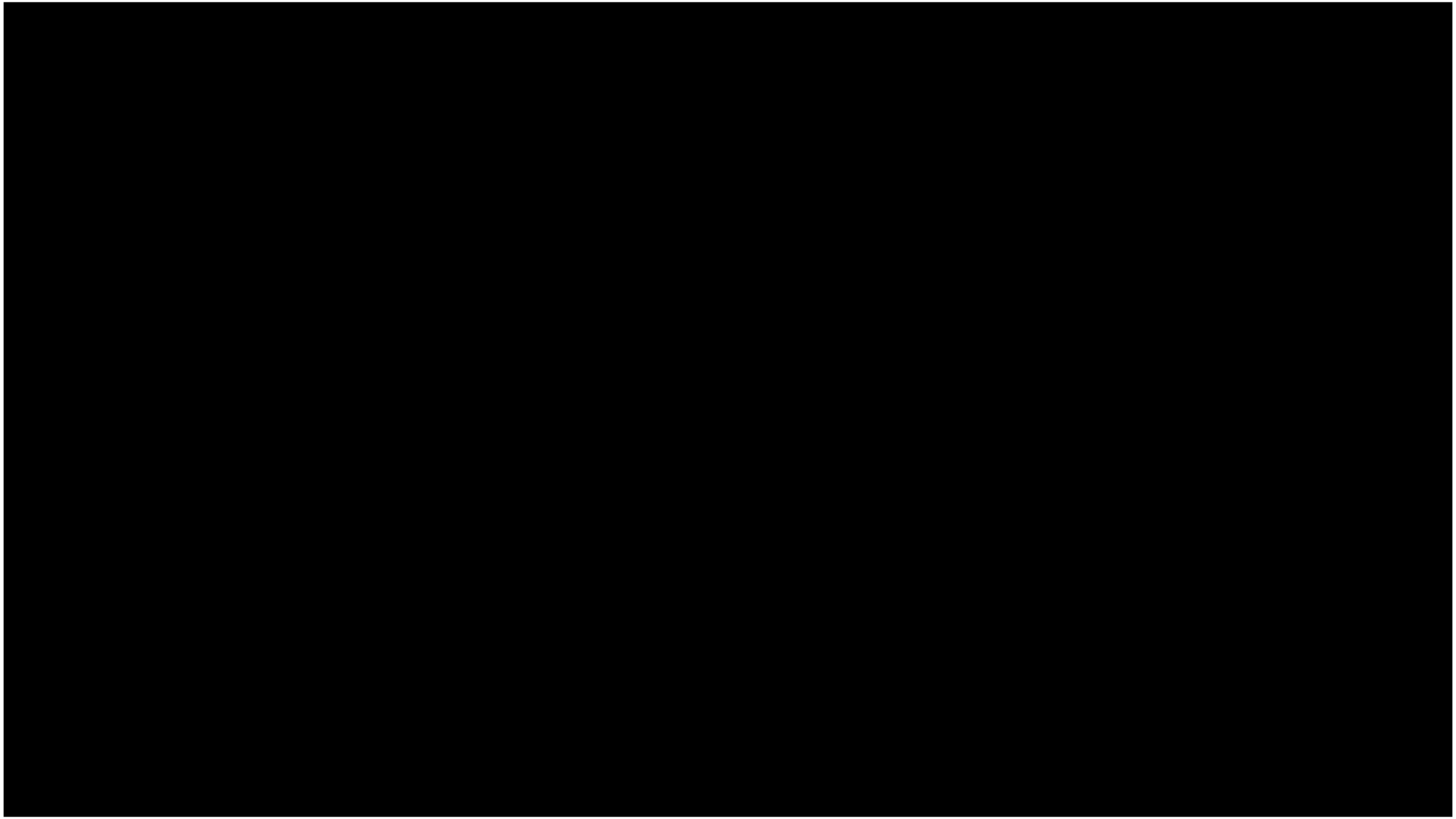
Kinematics describes the rotational and translational motion of points, bodies (objects) and systems of bodies (groups of objects) without consideration of what causes the motion or any reference to mass, force or torque. Inverse kinematics (IK) was initiated in robotics as the problem of moving a redundant kinematic arm with specific degrees of freedom (DoFs) to a pre-defined target. Beyond its use in robotics, IK has found applications in computer graphics, generating particular interest in the field of animating articulated subjects. This survey focuses on IK applications in computer graphics, aiming to provide insights about IK to young researchers by introducing the mathematical problem, and surveying the most popular techniques that tackle the problem.

Computer graphics applications usually deal with articulated figures, which are convenient models for humans, animals or other legged virtual creatures from films and video games. Animating such articulated characters is a challenging problem. Most virtual character models are complicated, made up of many joints, thus having a high number of DoFs. In addition, they are required to satisfy numerous constraints, including joint and/or contact restrictions. One way to handle this complexity is to manually adjust all the DoFs by carefully modifying the joint rotations to achieve the desired pose and ensure their temporal coherence—an extremely complex and time-consuming process.

Therefore, it was a necessity to find efficient ways to manipulate systems consisting of complex and multi-link models. IK has become one of the fundamental techniques for editing motion data. IK is commonly used for animating articulated figures using only the desired positions (and sometimes the orientations) of certain joints, commonly referred to as end effectors (e.g. usually end effectors are control points, and can be either end joints, such as feet and hands, or inner joints, such as the elbow and knee). The end effector positions are usually specified by the animator or a motion capture system, and must reach the desired positions in order to accomplish the given task. The remaining DoFs of the articulated model are

A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. 2018.
Inverse Kinematics Techniques in Computer Graphics: A Survey.
Computer Graphics Forum

Why do we need Inverse Kinematics?



Human Body Rig in Blender https://www.youtube.com/watch?v=MAM7mF2v7dE

Forward and Inverse Problems

For a system that can be described by a set of parameters $\boldsymbol{\theta}$, and a property $\boldsymbol{x}$ of the system given by

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

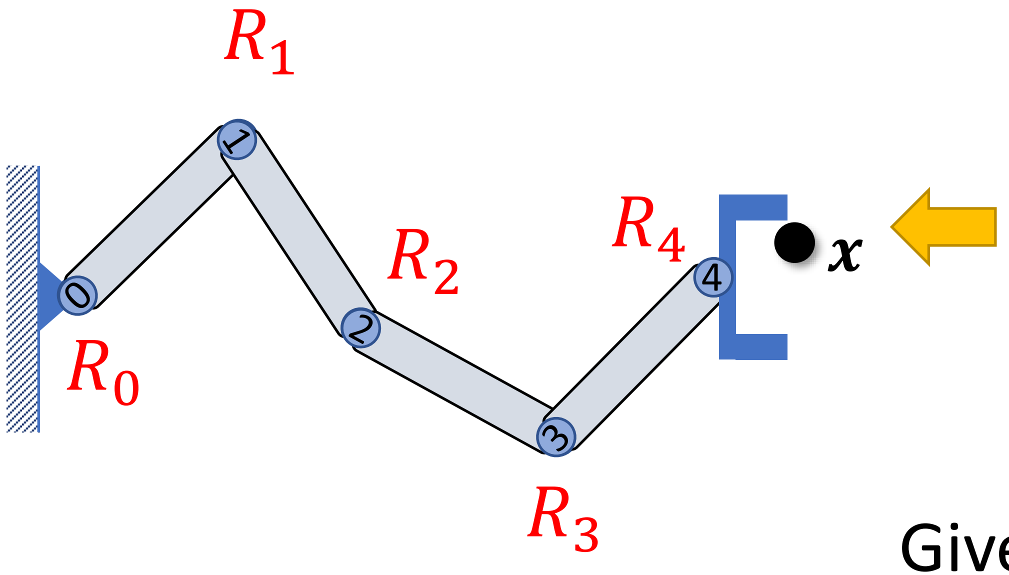
Forward problem:

- Given $\boldsymbol{\theta}$, we need to compute $\boldsymbol{x}$
- Easy to compute since f is known, the result is unique
- DoF of $\boldsymbol{\theta}$ is often much larger than that of $\boldsymbol{x}$. We cannot easily tune $\boldsymbol{\theta}$ to achieve a specific value of $\boldsymbol{x}$.

Inverse problem:

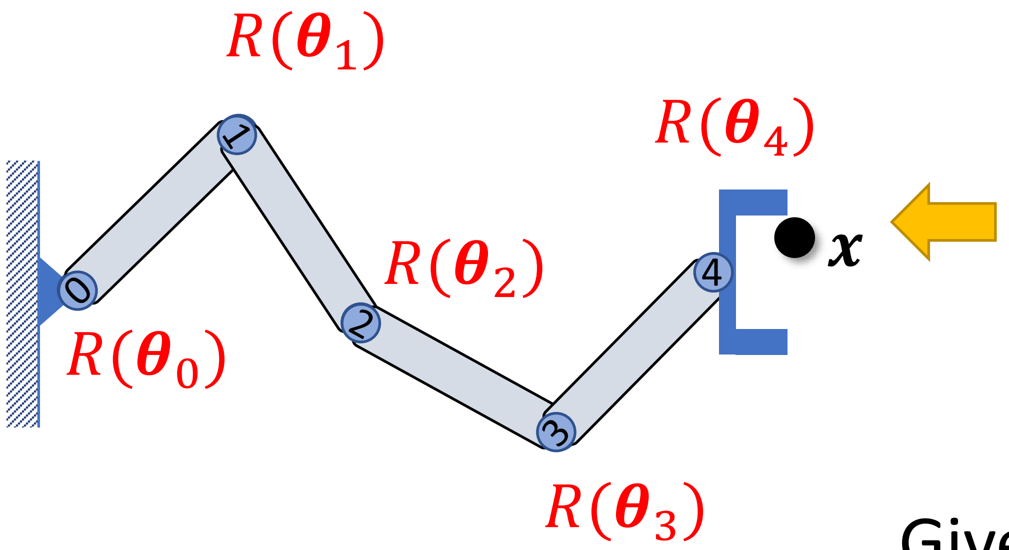
- Given $\boldsymbol{x}$, we need to find a set of valid parameters $\boldsymbol{\theta}$ such that $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\theta})$
- Often need to solve a difficult nonlinear equation, which can have multiple solutions
- $\boldsymbol{x}$ is typically meaningful and can be set in intuitive ways

Inverse Kinematics



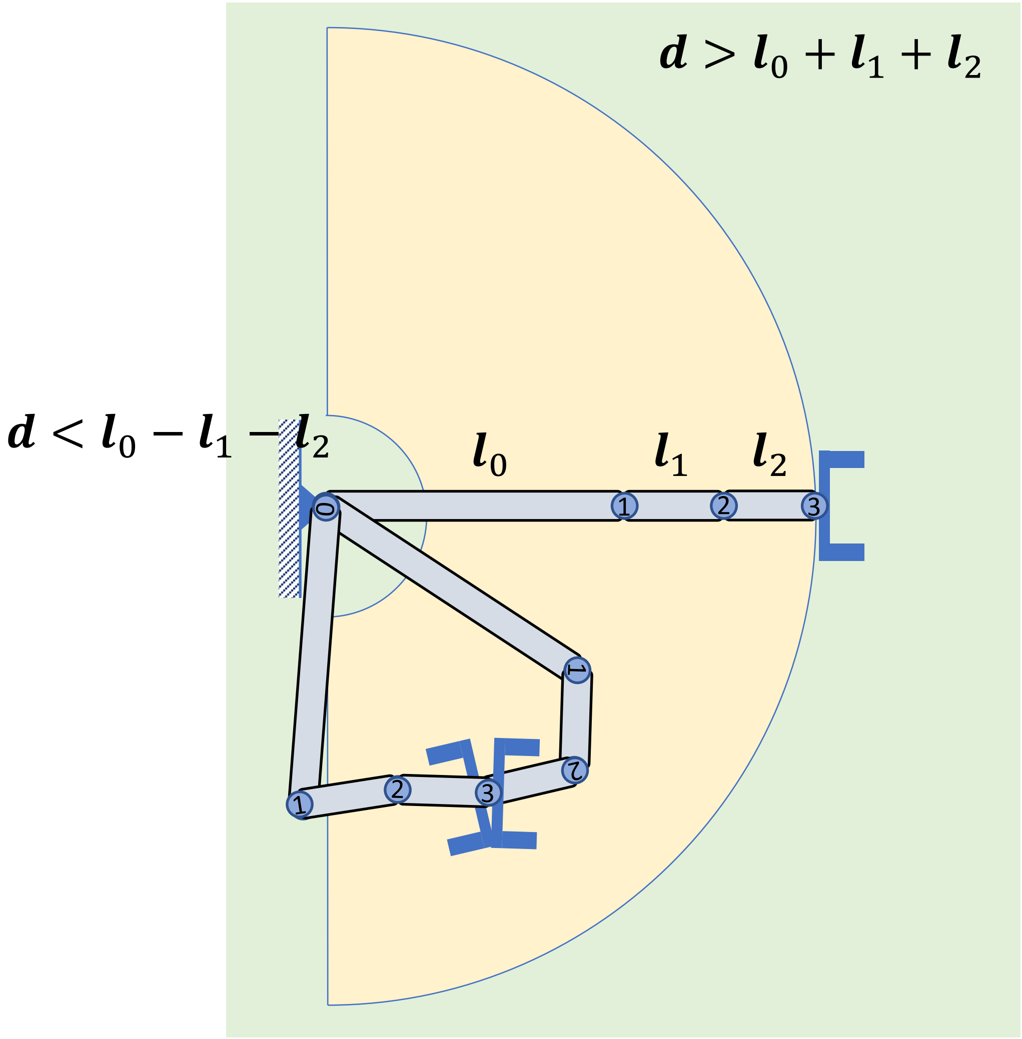
Given the position of the end-effector $\boldsymbol{x}$,
Compute the joint rotations R_i

Inverse Kinematics



Given the position of the end-effector $\boldsymbol{x}$,
Compute the joint rotation parameters $\boldsymbol{\theta}_i$

Solutions of IK Problems

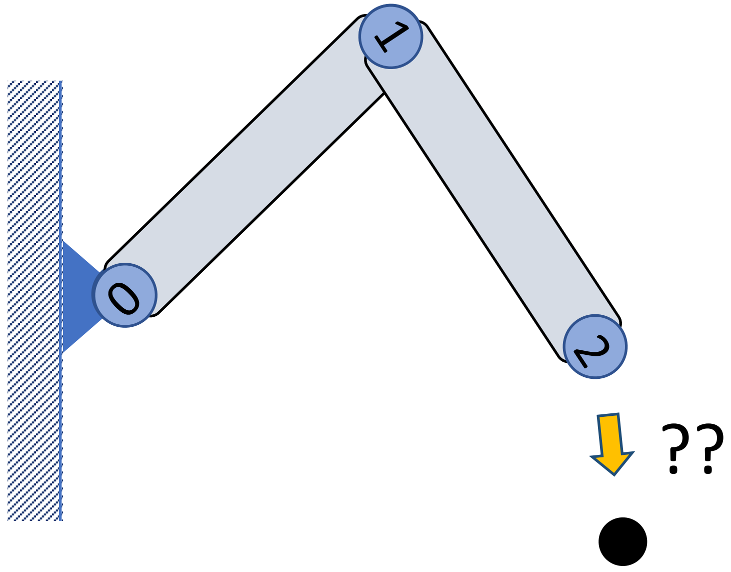


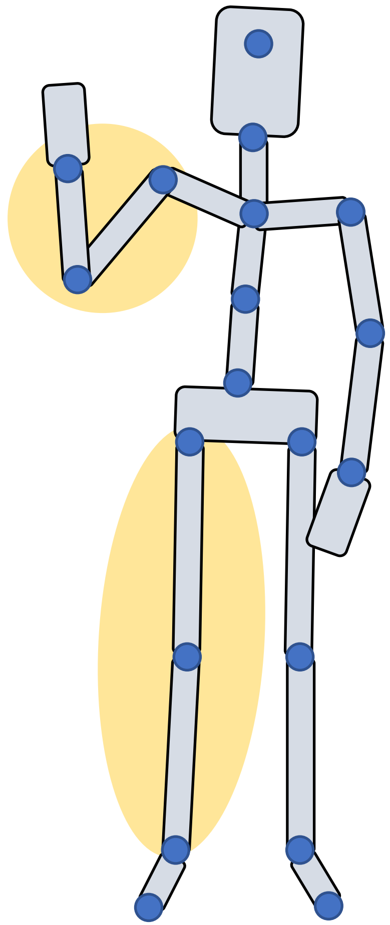
No solution

Multiple solutions

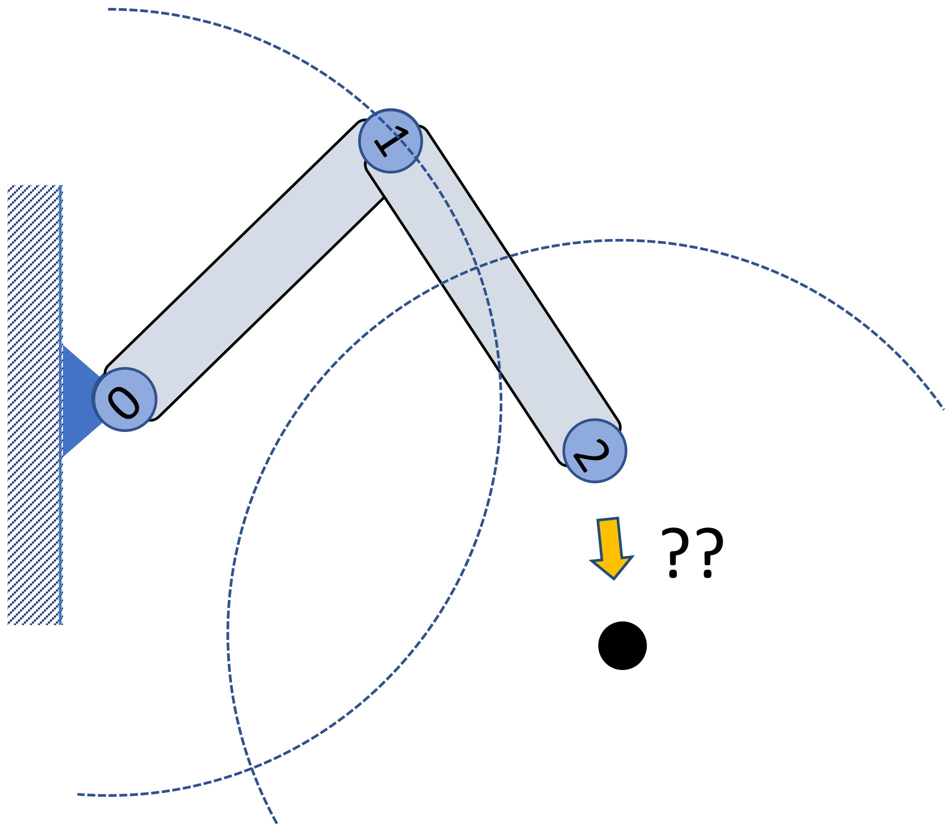
Unique solution

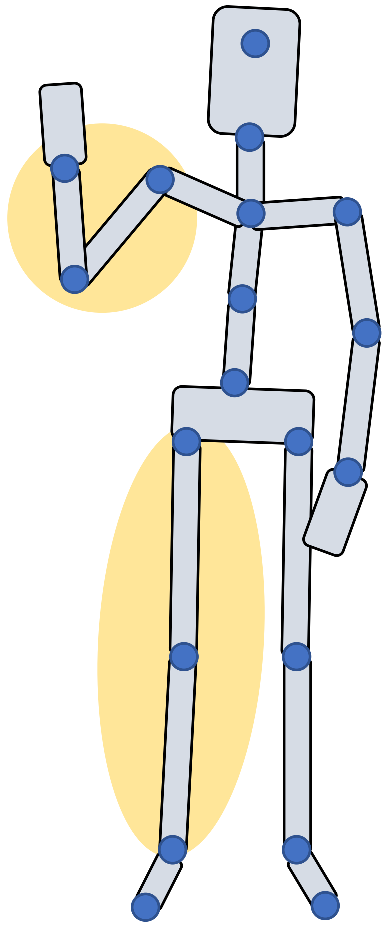
Example: Two-Joint IK



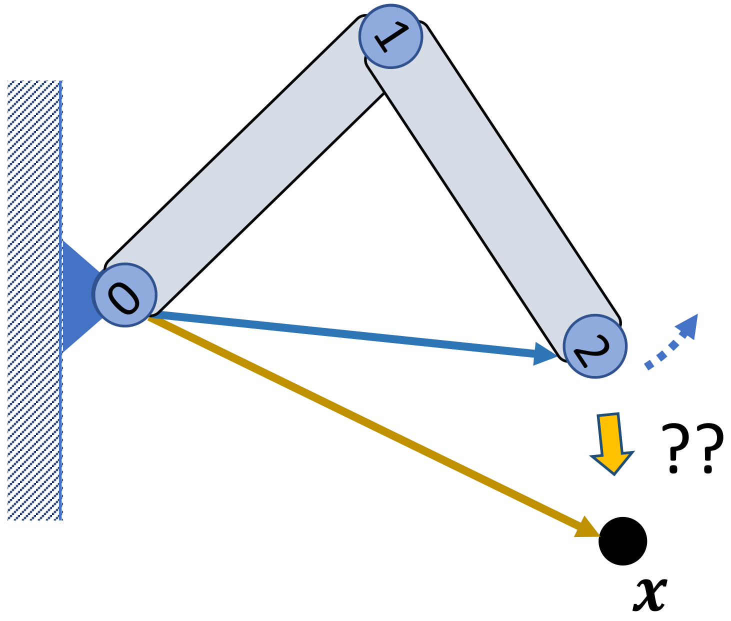


Example: Two-Joint IK



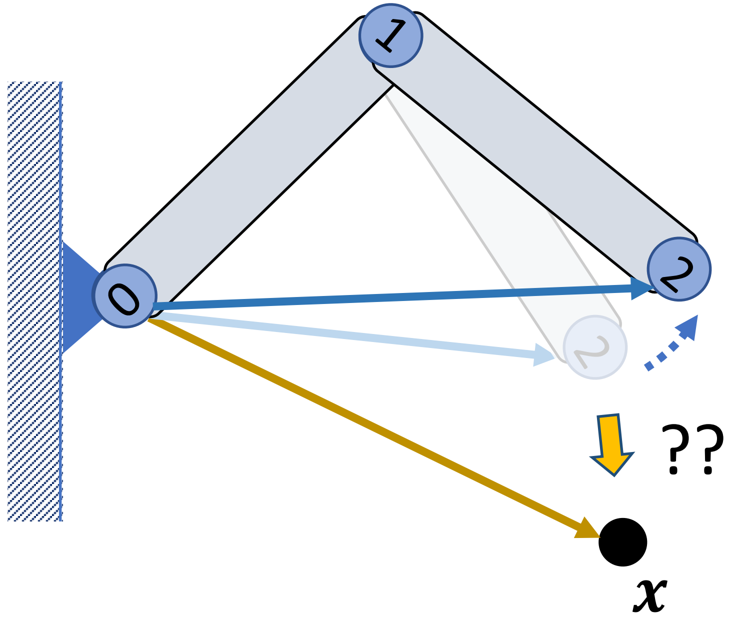


A simple solution to a two-joint IK problem



1. Rotate joint 1 such that

A simple solution to a two-joint IK problem

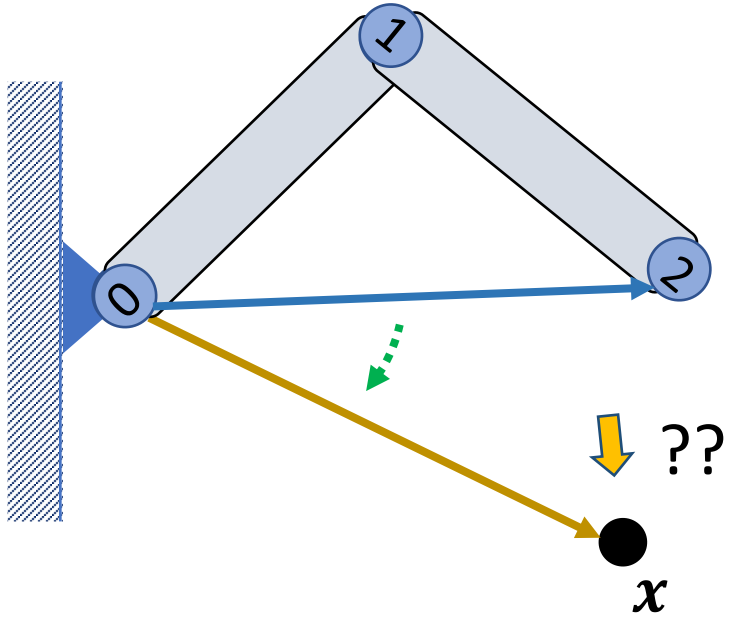


1. Rotate joint 1 such that

$$\|\boldsymbol{l}_{0x}\| = \|\boldsymbol{l}_{02}\|$$

How??

A simple solution to a two-joint IK problem



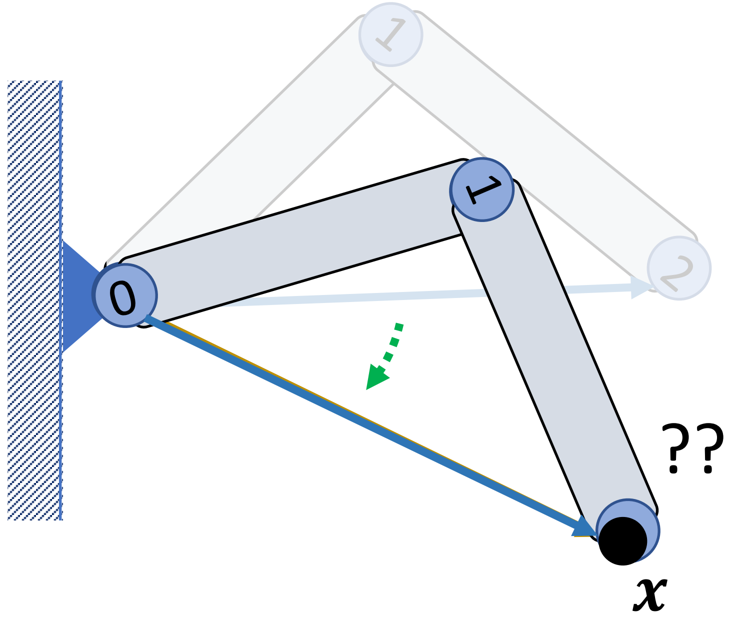
1. Rotate joint 1 such that

$$\|\boldsymbol{l}_{0x}\| = \|\boldsymbol{l}_{02}\|$$

How??

2. Rotate joint 0 such that

A simple solution to a two-joint IK problem



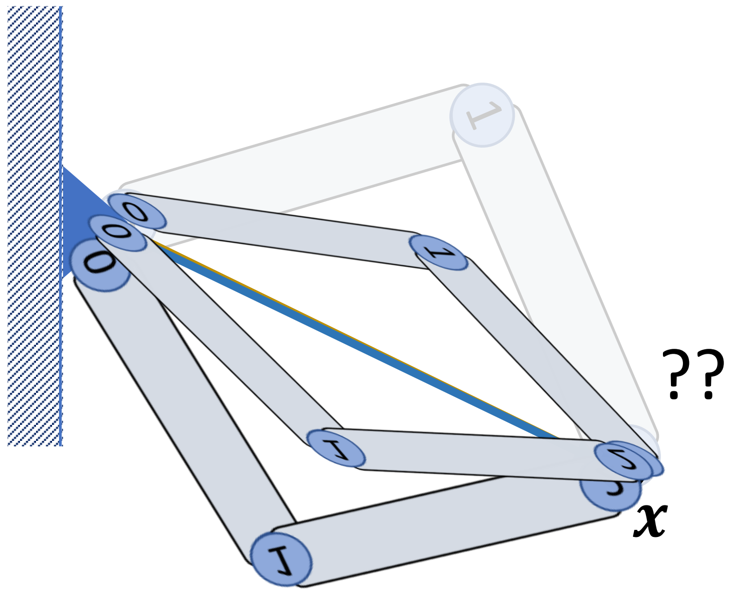
1. Rotate joint 1 such that

$$\|\boldsymbol{l}_{0x}\| = \|\boldsymbol{l}_{02}\| \qquad \text{How??}$$

2. Rotate joint 0 such that

$$\boldsymbol{l}_{0x} = \boldsymbol{l}_{02} \qquad \text{How??}$$

A simple solution to a two-joint IK problem



1. Rotate joint 1 such that

$$\|\boldsymbol{l}_{0x}\| = \|\boldsymbol{l}_{02}\|$$

How??

2. Rotate joint 0 such that

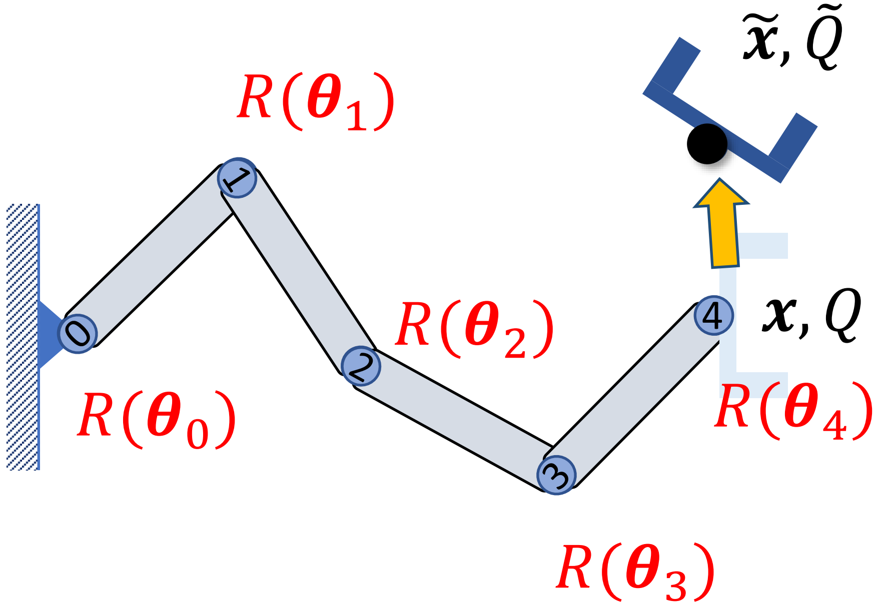
$$\boldsymbol{l}_{0x} = \boldsymbol{l}_{02}$$

How??

3. Rotate joint 0 around $\boldsymbol{l}_{0x}$ if necessary

How??

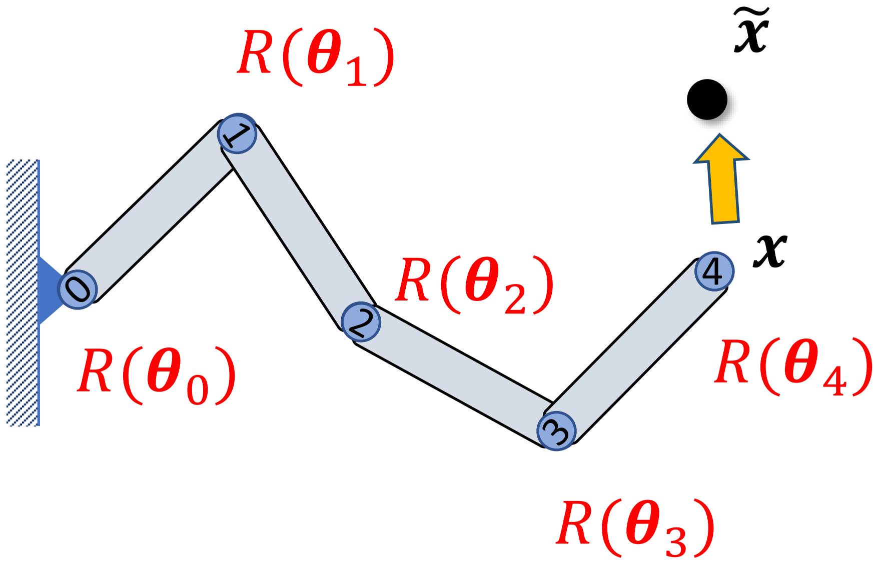
IK as an Optimization Problem



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

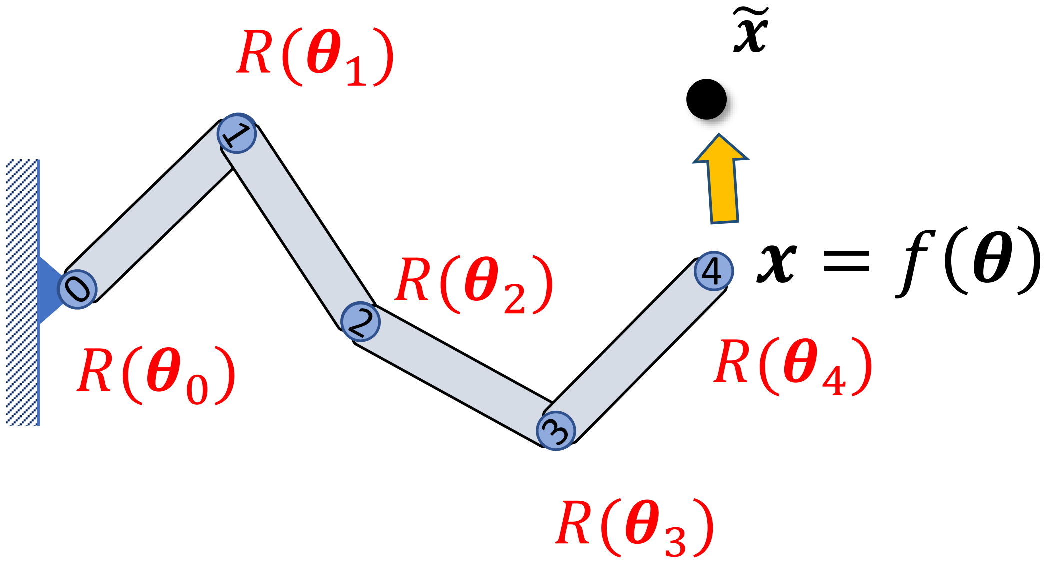
$$Q = Q(\boldsymbol{\theta})$$

IK as an Optimization Problem



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

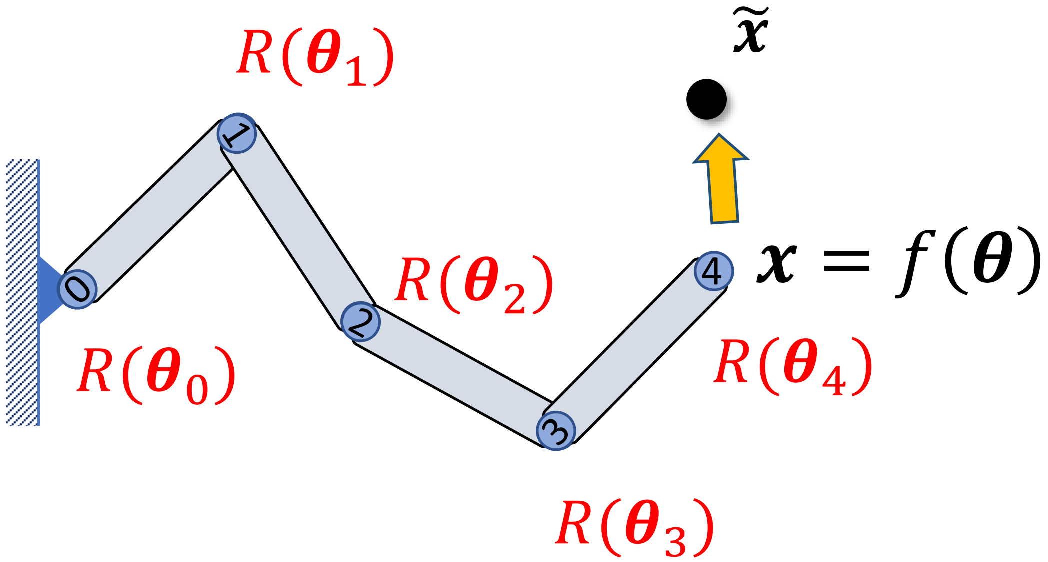
IK as an Optimization Problem



Find $\boldsymbol{\theta}$ such that

$$\tilde{\boldsymbol{x}} - f(\boldsymbol{\theta}) = 0$$

IK as an Optimization Problem



Find $\boldsymbol{\theta}$ to optimize

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$

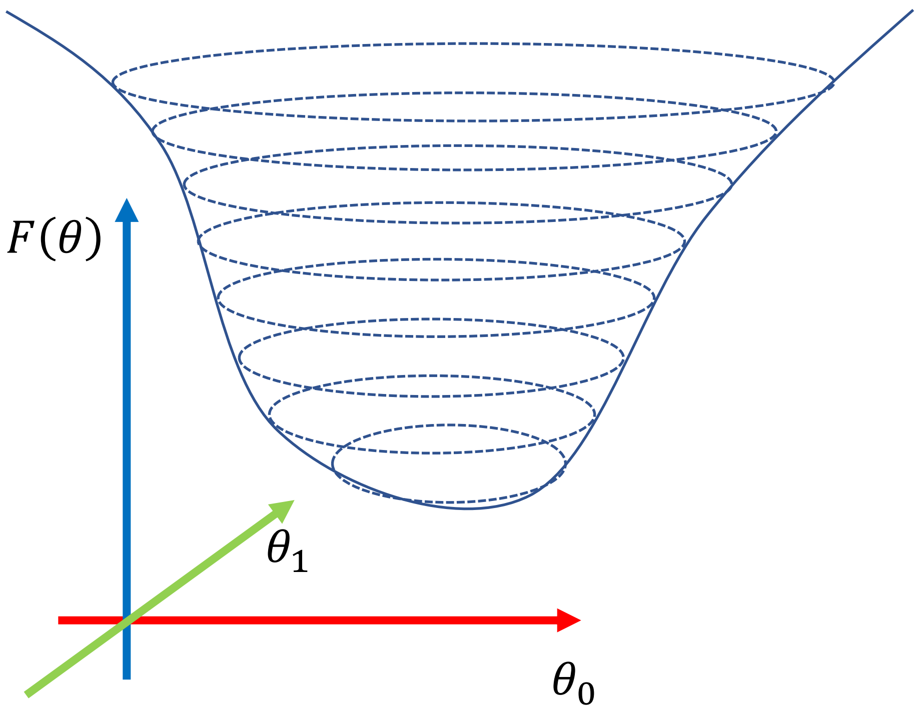
Optimization Problems

Find $\boldsymbol{\theta}$ to optimize

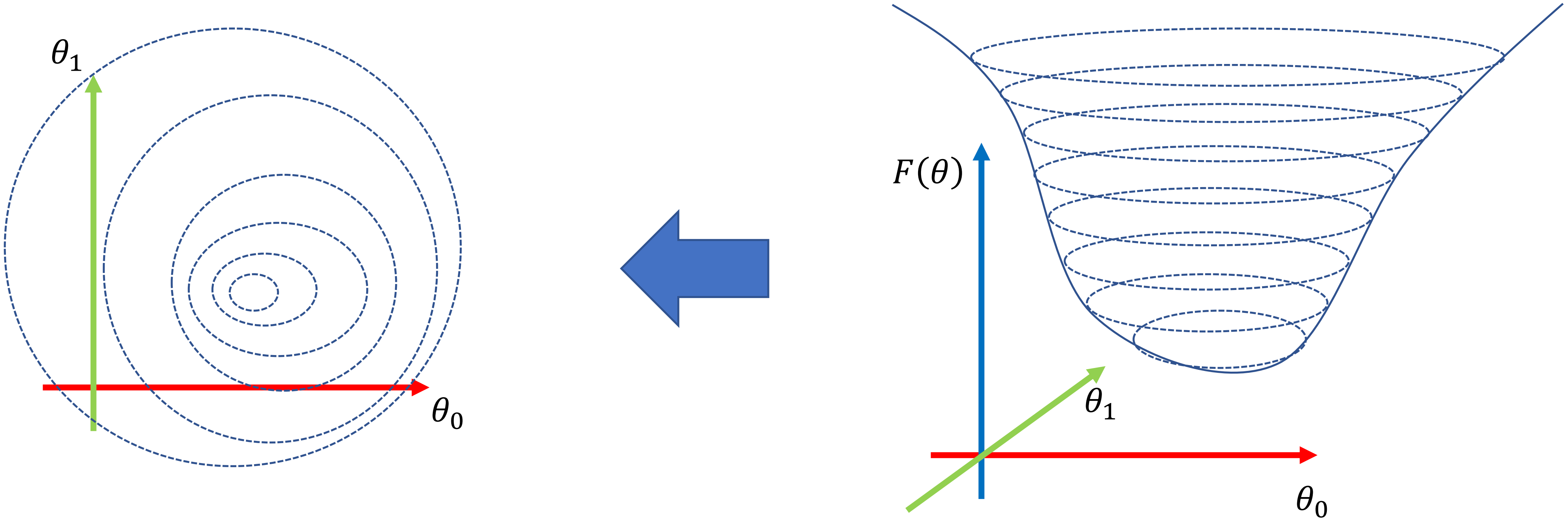
$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

For an IK problem, we can write

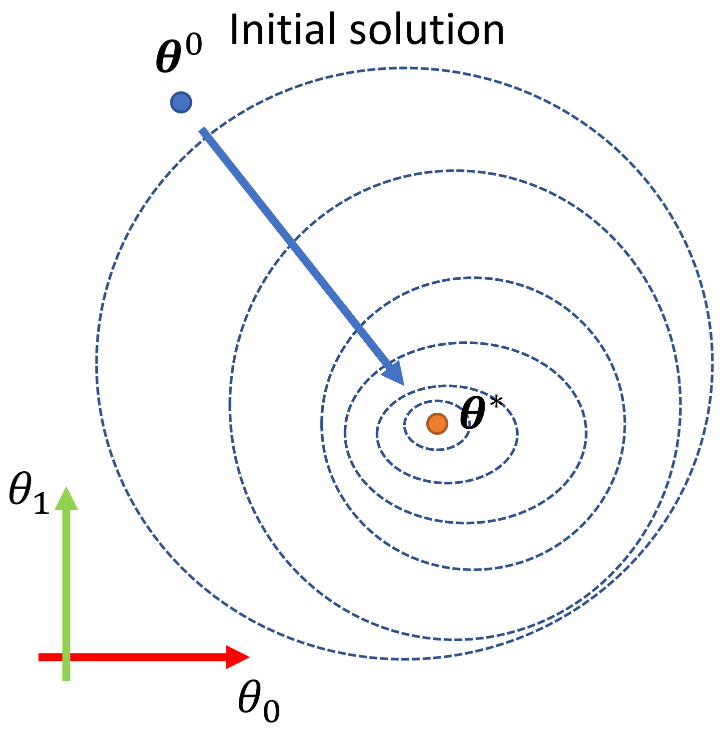
$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



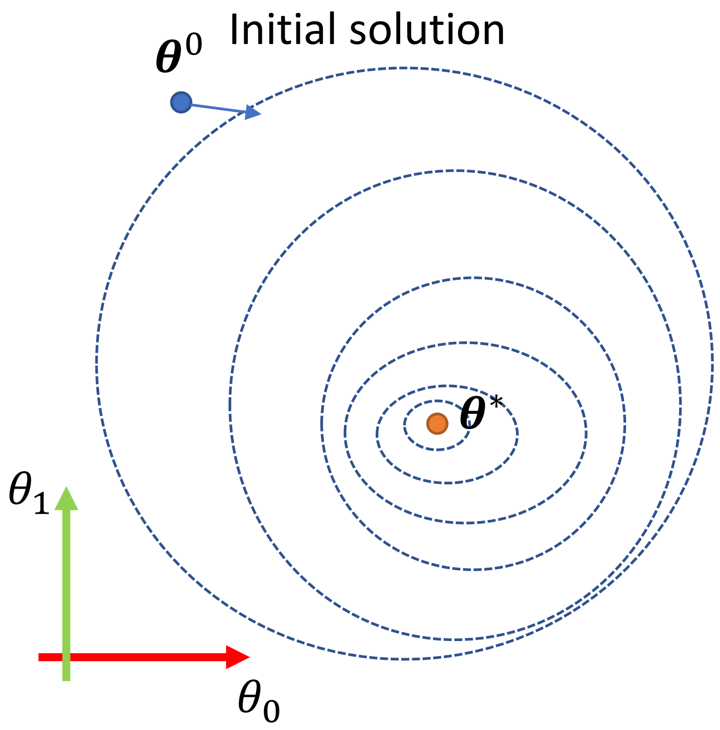
Optimization Problems



Optimization Problems

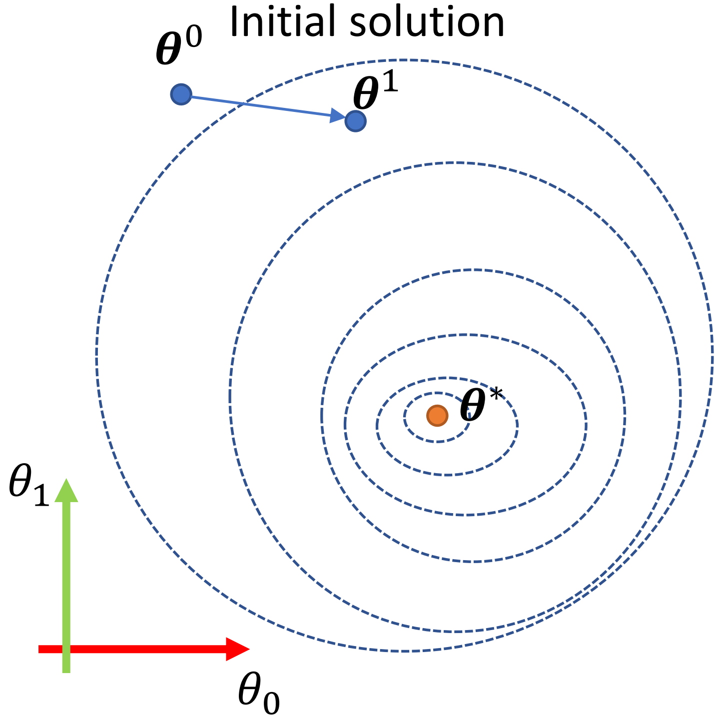


Iterative Algorithms for Optimization Problems



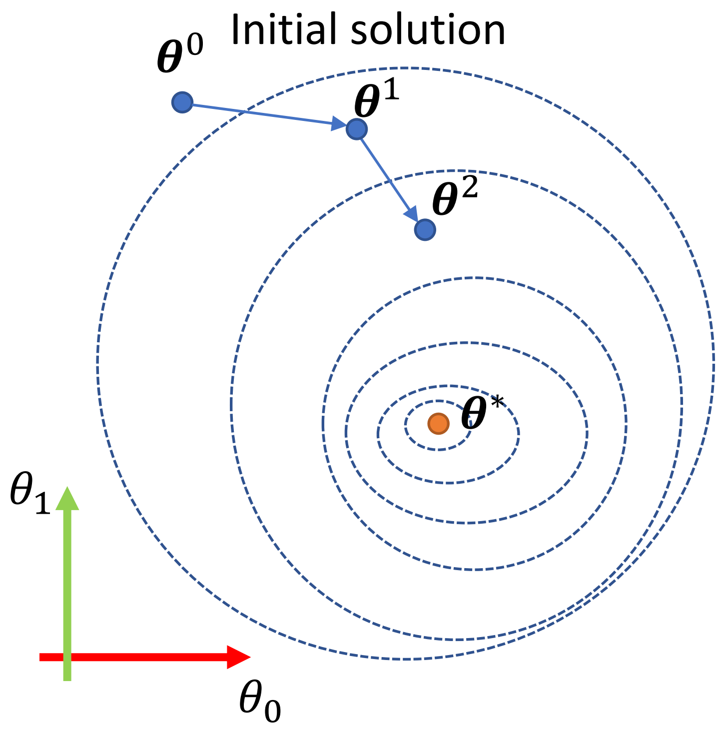
- Find a promising direction to update the parameters

Iterative Algorithms for Optimization Problems



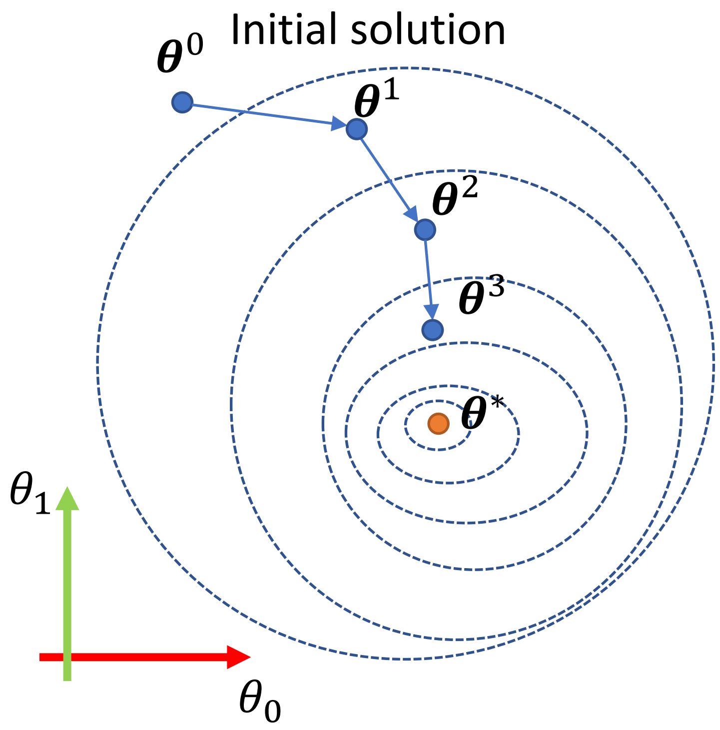
- Find a promising direction to update the parameters
- Move the parameters along that direction by a proper distance

Iterative Algorithms for Optimization Problems



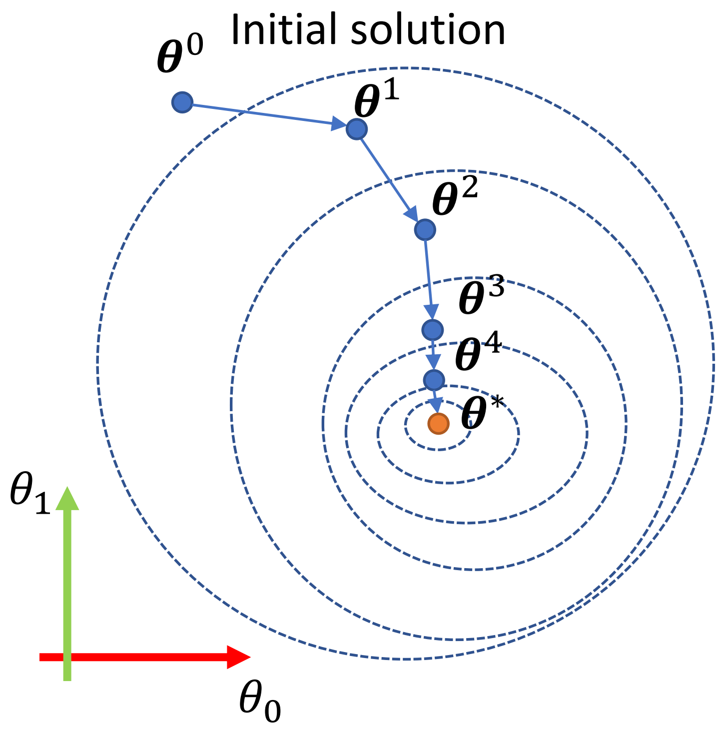
- Find a promising direction to update the parameters
- Move the parameters along that direction by a proper distance
- Repeat until reaching the optimal parameters

Iterative Algorithms for Optimization Problems



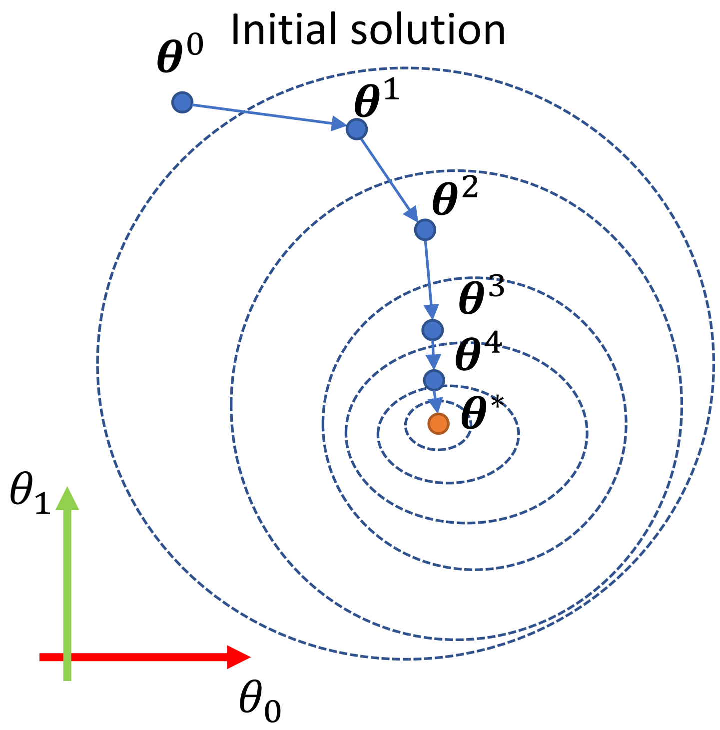
- Find a promising direction to update the parameters
- Move the parameters along that direction by a proper distance
- Repeat until reaching the optimal parameters

Iterative Algorithms for Optimization Problems



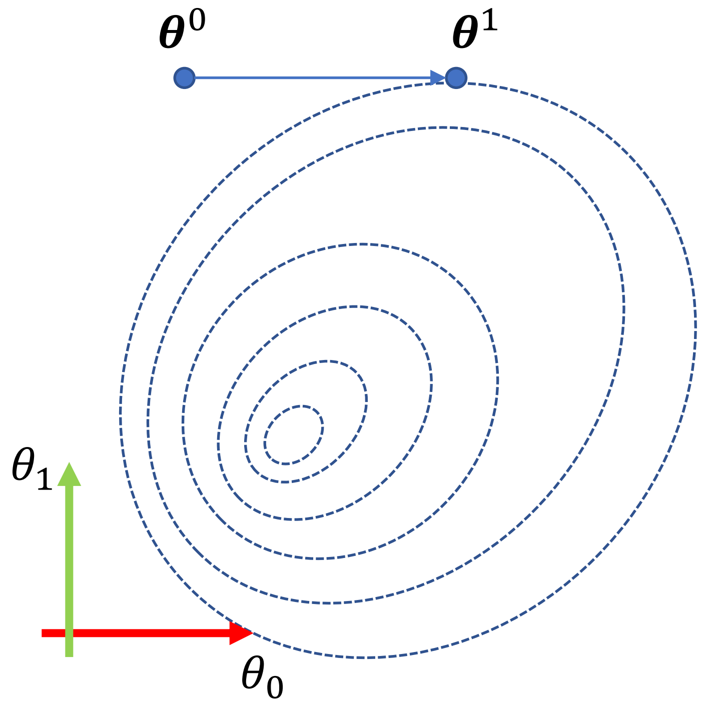
- Find a promising direction to update the parameters
- Move the parameters along that direction by a proper distance
- Repeat until reaching the optimal parameters

Iterative Algorithms for Optimization Problems



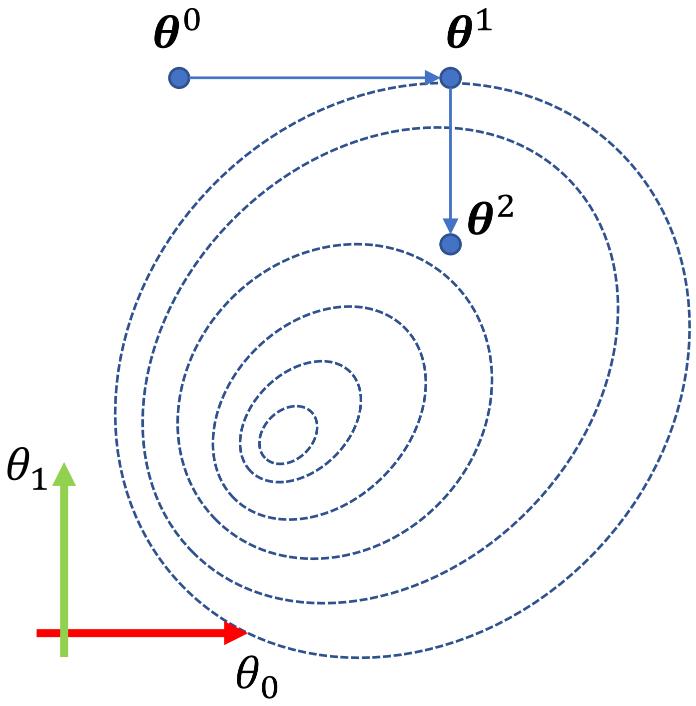
- Find a promising direction to update the parameters
- Move the parameters along that direction by a proper distance
- Repeat until reaching the optimal parameters (or stop after several iterations)

Coordinate Descent



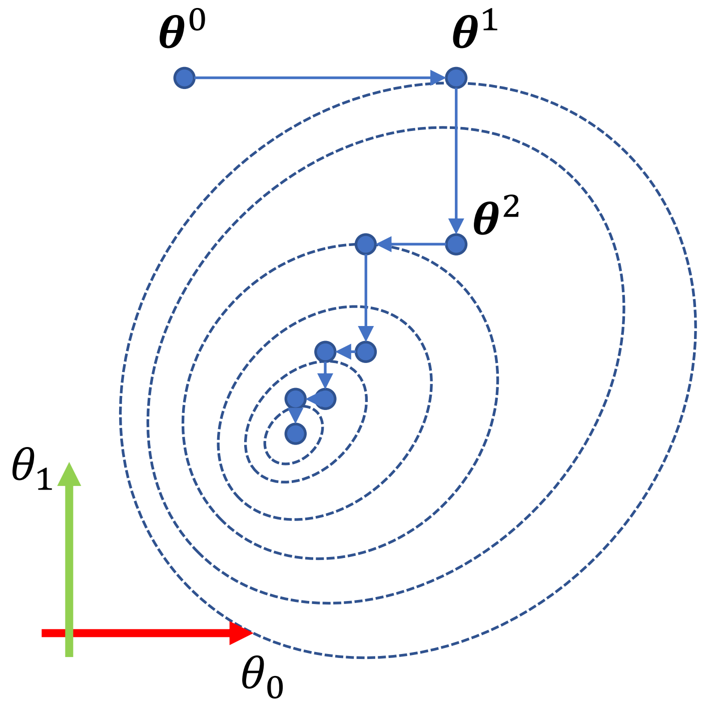
Update parameters along each axis of the coordinate system

Coordinate Descent



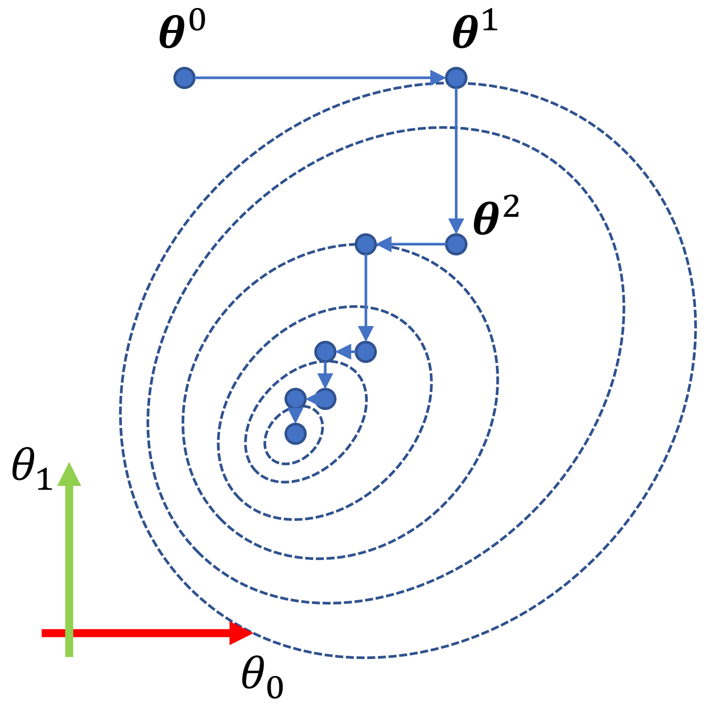
Update parameters along each axis of the coordinate system

Coordinate Descent



Update parameters along each axis of the coordinate system

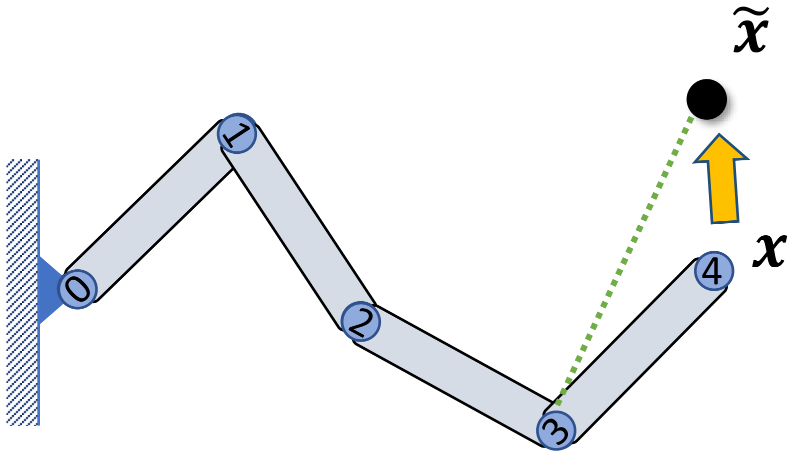
Cyclic Coordinate Descent (CCD)



Update parameters along each axis of the coordinate system

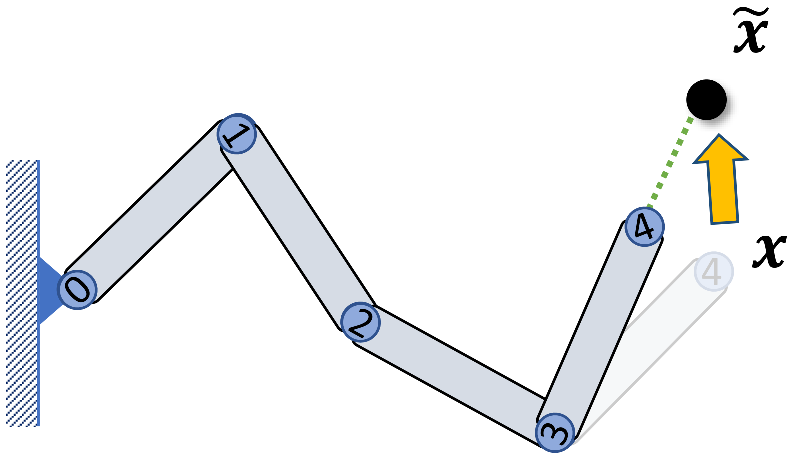
Iterate cyclically through all axes

Cyclic Coordinate Descent (CCD) IK



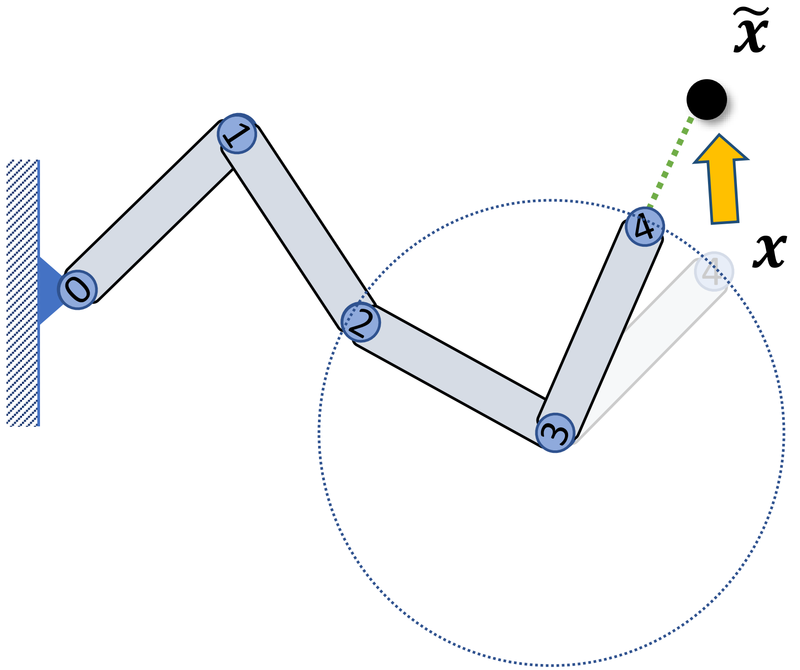
Rotate joint 3 such that

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK

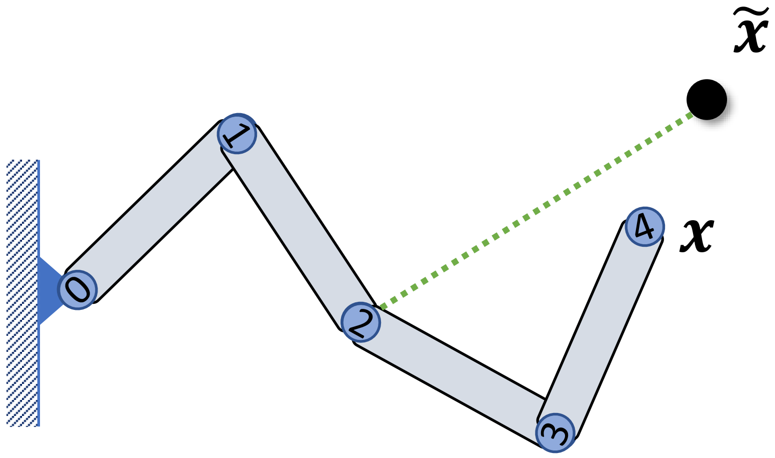


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

$$\min_{\theta_3} F(\boldsymbol{\theta})$$

$$= \min_{\theta_3} \frac{1}{2} \| f(\theta_0, \theta_1, \theta_2, \theta_3) - \tilde{\boldsymbol{x}} \|_2^2$$

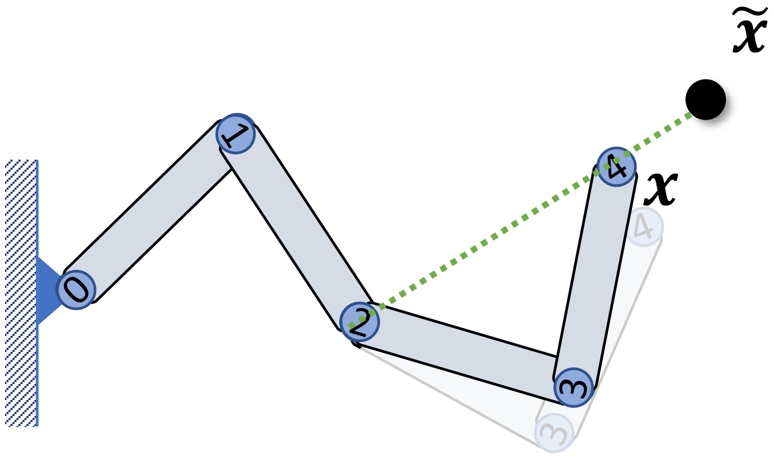
Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

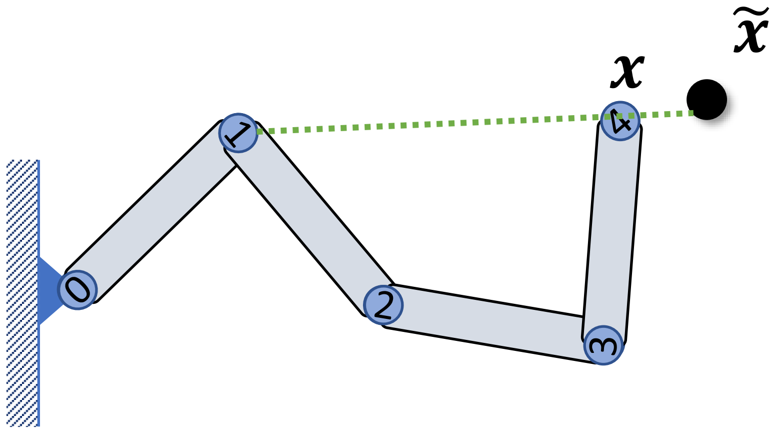
Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK

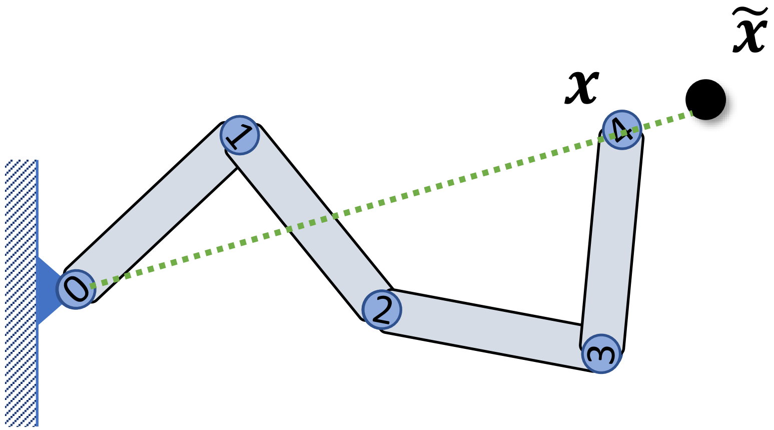


Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



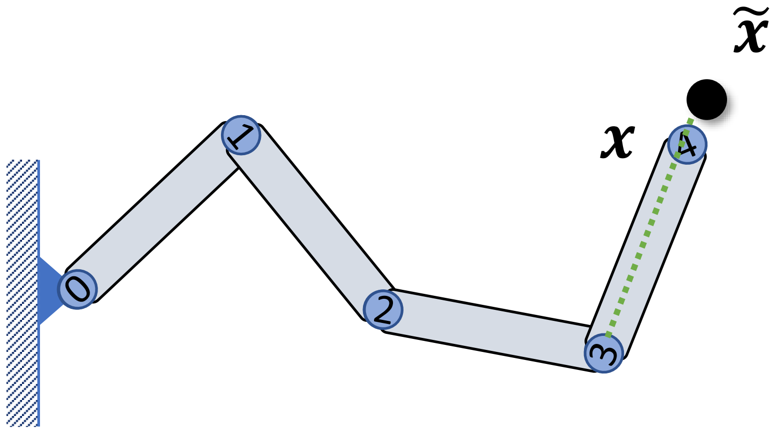
Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 0 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Cyclic Coordinate Descent (CCD) IK



Rotate joint 3 such that $\boldsymbol{l}_{34}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 2 such that $\boldsymbol{l}_{24}$ points towards $\tilde{\boldsymbol{x}}$

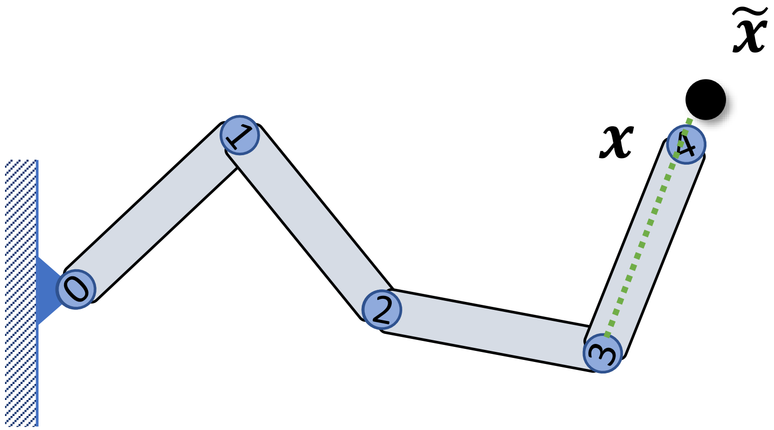
Rotate joint 1 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 0 such that $\boldsymbol{l}_{14}$ points towards $\tilde{\boldsymbol{x}}$

Rotate joint 3 such that $\boldsymbol{l}'_{34}$ points towards $\tilde{\boldsymbol{x}}$

……

Cyclic Coordinate Descent (CCD) IK



Iteratively rotation each joint to make the end-effector align with vector between the joint and the target

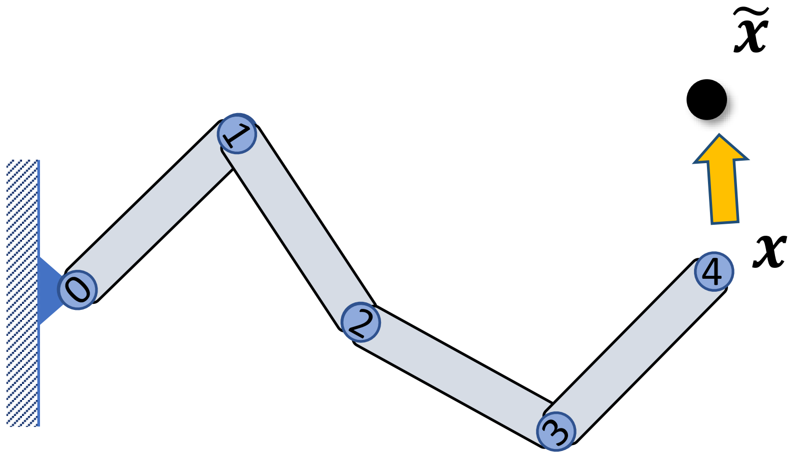
Easy to implement, very fast

The "first" joint moves more than the others

May take many iterations to converge

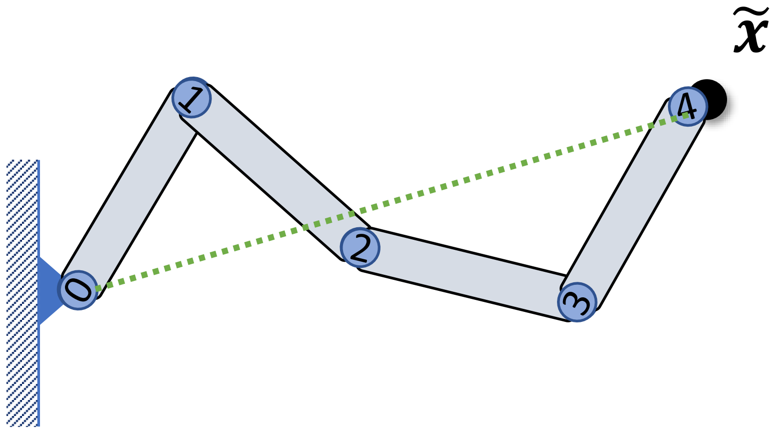
Result can be sensitive to the initial solution

Cyclic Coordinate Descent (CCD) IK



Rotate joint 0 such that $\boldsymbol{l}_{04}$ points towards $\tilde{\boldsymbol{x}}$

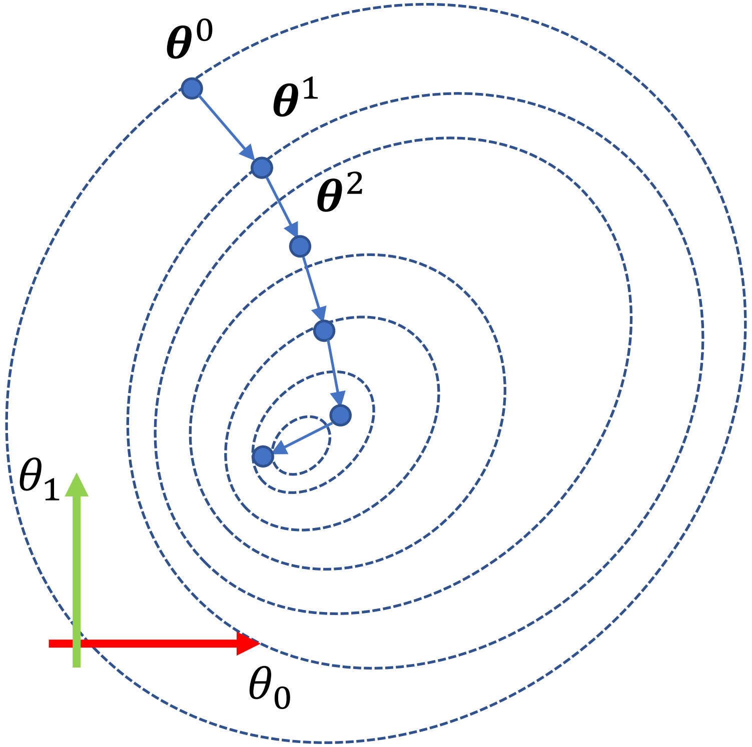
Cyclic Coordinate Descent (CCD) IK



Rotate joint 0 such that $\boldsymbol{l}_{04}$ points towards $\tilde{\boldsymbol{x}}$

Gradient Descent

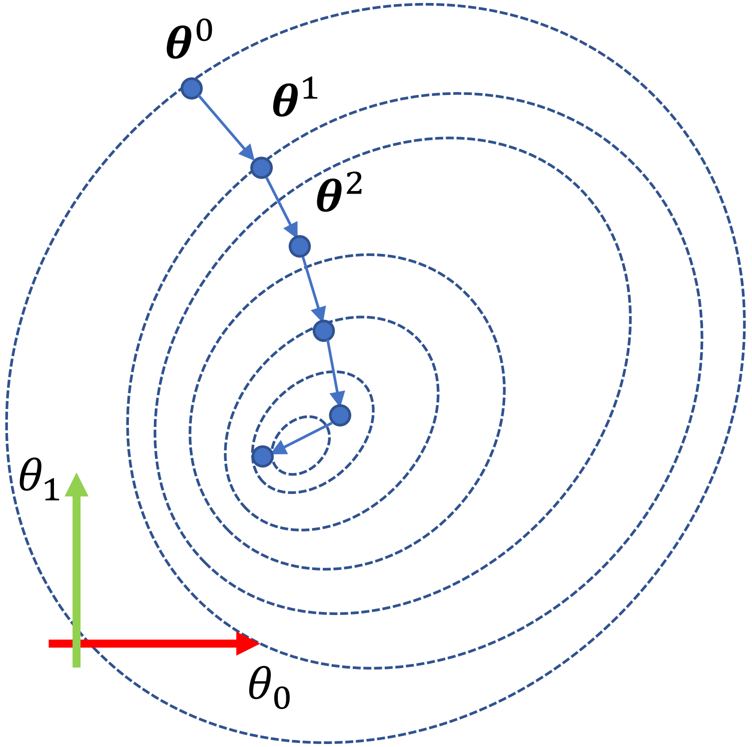
$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

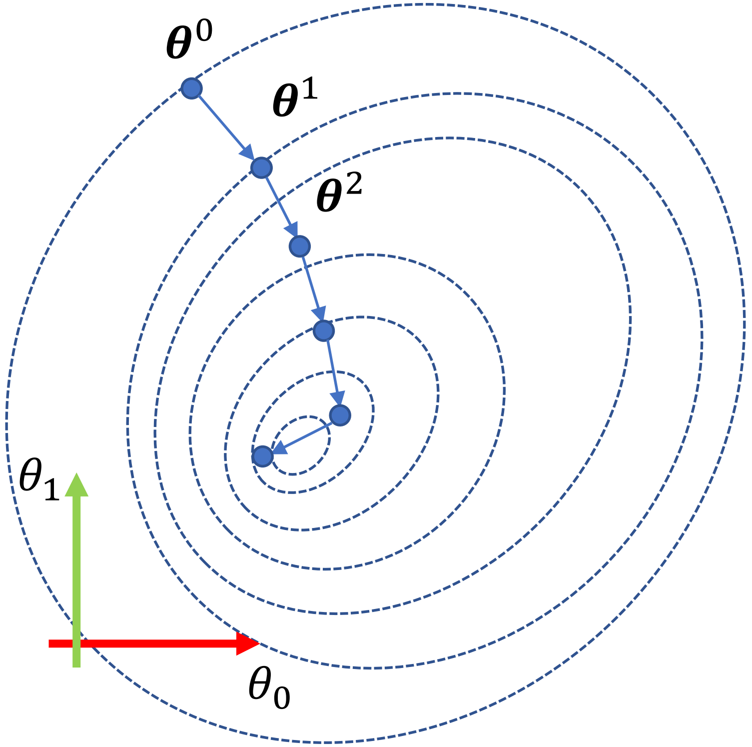
Gradient:

$$\nabla_\theta F(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial F}{\partial \theta_0}(\boldsymbol{\theta}) \\ \frac{\partial F}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial F}{\partial \theta_n}(\boldsymbol{\theta}) \end{bmatrix} = \left(\frac{\partial F}{\partial \boldsymbol{\theta}}(\theta)\right)^T$$

The direction in which $F(\theta)$ increases fastest

Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



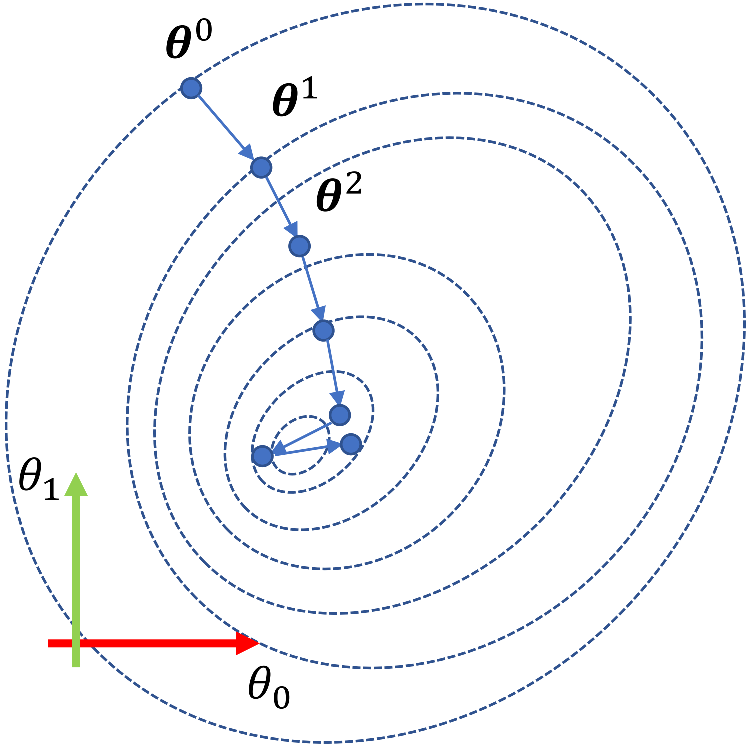
Update parameters against the direction of the gradient of the objective function

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \alpha\, \nabla_{\boldsymbol{\theta}} F\left(\boldsymbol{\theta}^i\right)$$

learning rate

Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

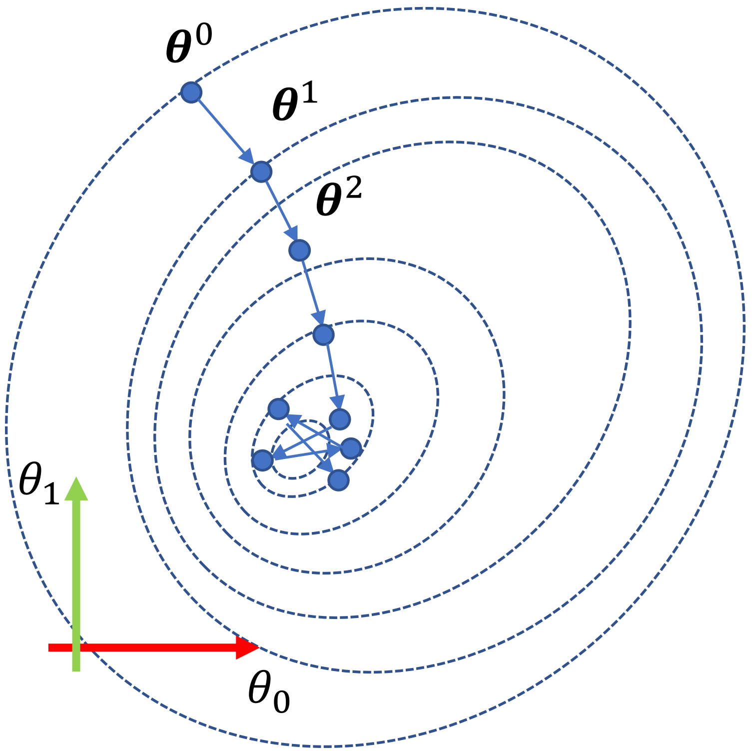
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \alpha \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^i)$$

learning rate

Large learning rate can cause problems

Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

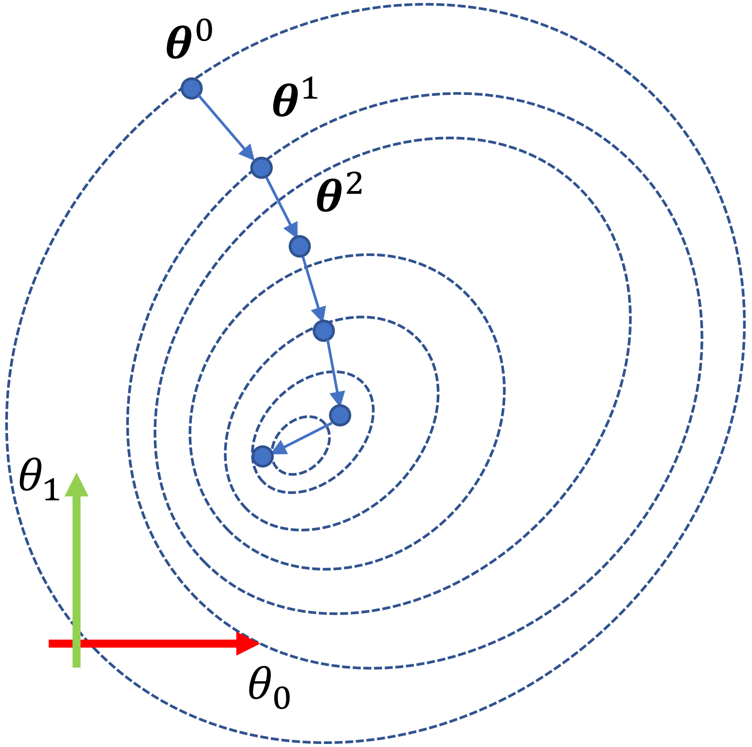
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha\, \nabla_{\boldsymbol{\theta}} F\left(\boldsymbol{\theta}^{i}\right)$$

learning rate

Large learning rate can cause problems

Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \alpha \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^i)$$

$$\nabla_\theta F(\boldsymbol{\theta}^i) = \left(\frac{\partial f}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^i)\right)^T \left(f(\boldsymbol{\theta}^i) - \tilde{\boldsymbol{x}}\right)$$

$$= J^T \Delta$$

Jacobian Transpose

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha J^T \Delta$$

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

Jacobian Transpose

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

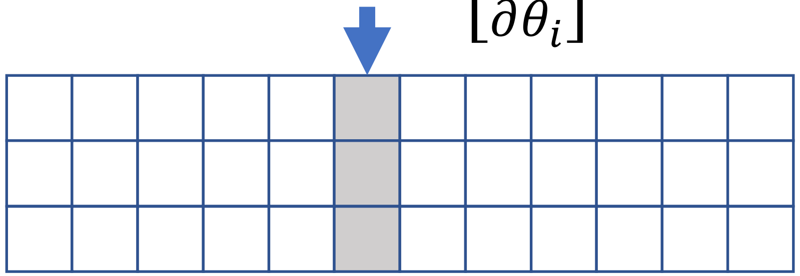
$$\frac{\partial f}{\partial \theta_i} = \begin{bmatrix} \frac{\partial f_x}{\partial \theta_i} \\ \frac{\partial f_y}{\partial \theta_i} \\ \frac{\partial z}{\partial \theta_i} \end{bmatrix}$$

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

$$f: \mathbb{R}^n \mapsto \mathbb{R}^3$$

$$\Rightarrow$$

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



How to compute the Jacobian matrix?

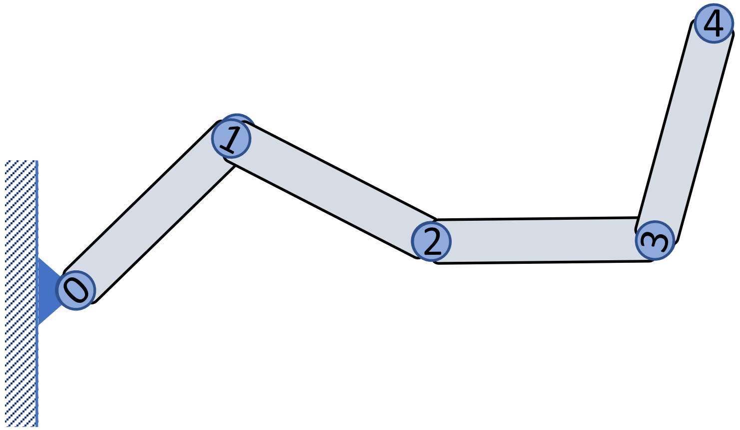
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha J^{T} \Delta \qquad J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

- Implement $f(\theta)$ using your favorite machine learning framework
  - pytorch, tensorflow, ……
- Compute gradient using its *autograd* functionality
- Enjoy!

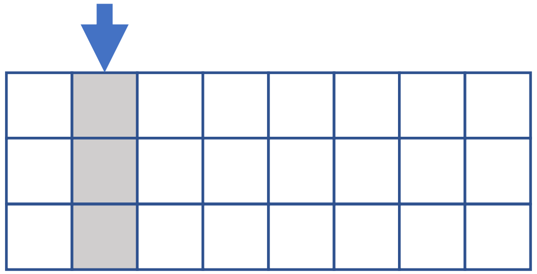
Finite Differencing

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

$$\boldsymbol{x} = f(\theta_0, \theta_1, \theta_2, \theta_3)$$



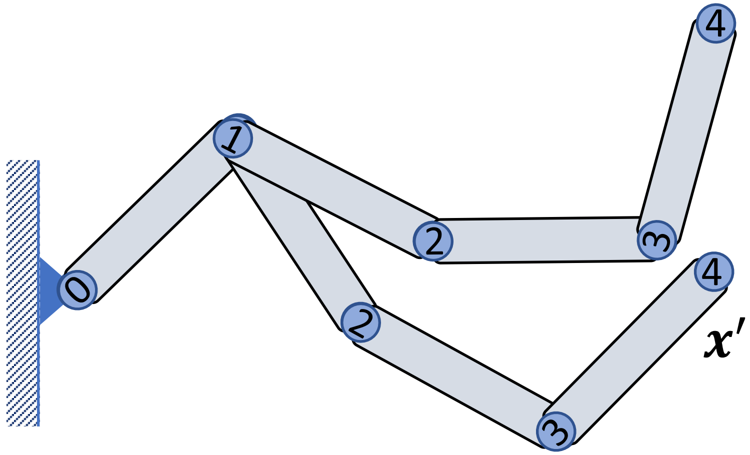
$$\frac{\partial f}{\partial \theta_1} = ??$$



Finite Differencing

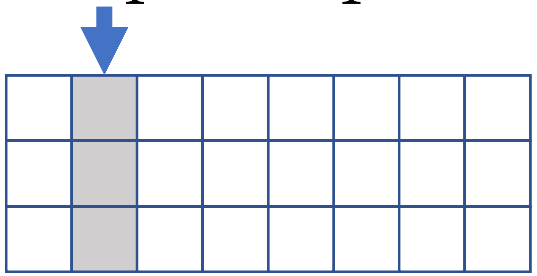
$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

$$\boldsymbol{x} = f(\theta_0, \theta_1, \theta_2, \theta_3)$$



$$\boldsymbol{x}' = f(\theta_0, \theta_1 + \delta\theta_1, \theta_2, \theta_3)$$

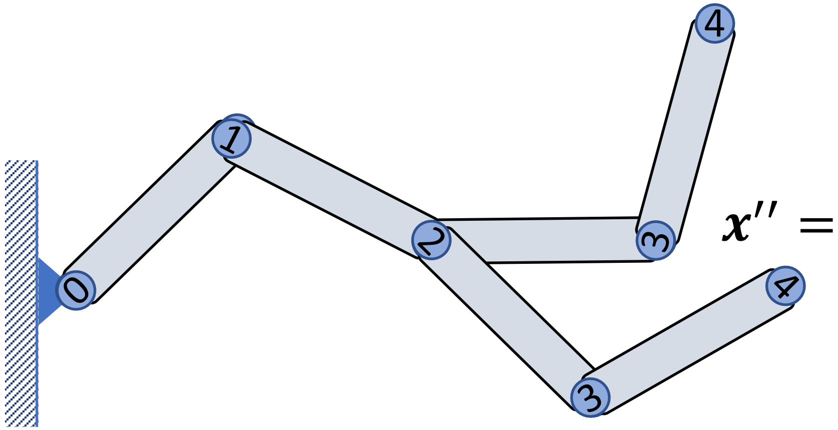
$$\frac{\partial f}{\partial \theta_1} \approx \frac{\boldsymbol{x}' - \boldsymbol{x}}{\delta \theta_1}$$



Finite Differencing

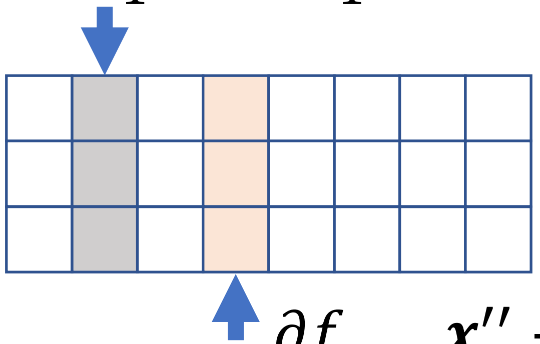
$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

$$\boldsymbol{x} = f(\theta_0, \theta_1, \theta_2, \theta_3)$$



$$\boldsymbol{x}'' = f(\theta_0, \theta_1, \theta_2 + \delta\theta_2, \theta_3)$$

$$\frac{\partial f}{\partial \theta_1} \approx \frac{\boldsymbol{x}' - \boldsymbol{x}}{\delta \theta_1}$$

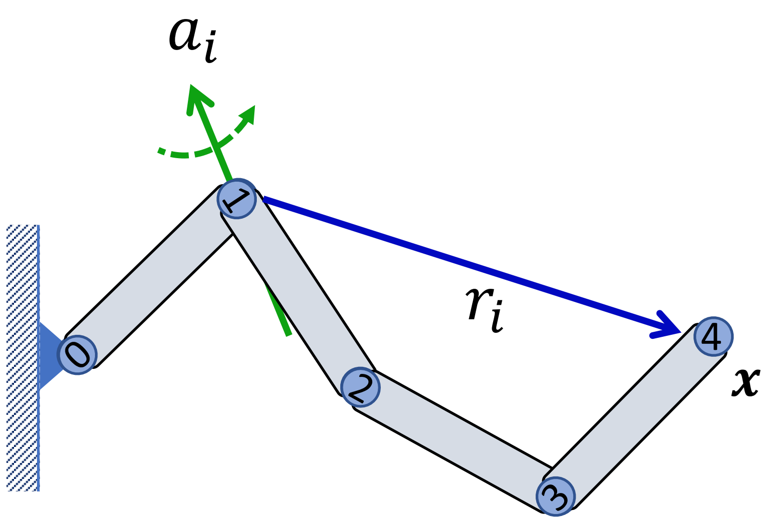


$$\frac{\partial f}{\partial \theta_2} \approx \frac{\boldsymbol{x}'' - \boldsymbol{x}}{\delta \theta_2}$$

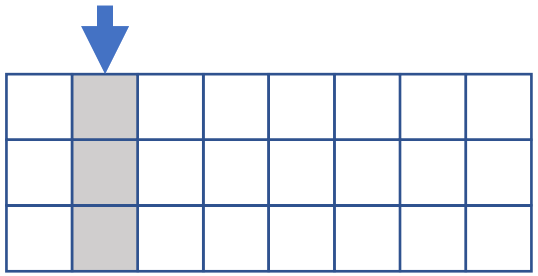
Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

Assuming all joints are hinge joint



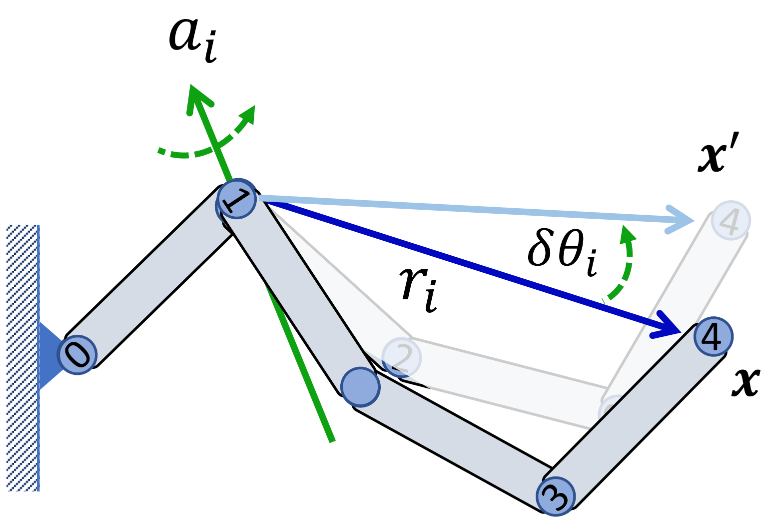
$$\frac{\partial f}{\partial \theta_i} = ??$$



Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

Assuming all joints are hinge joint



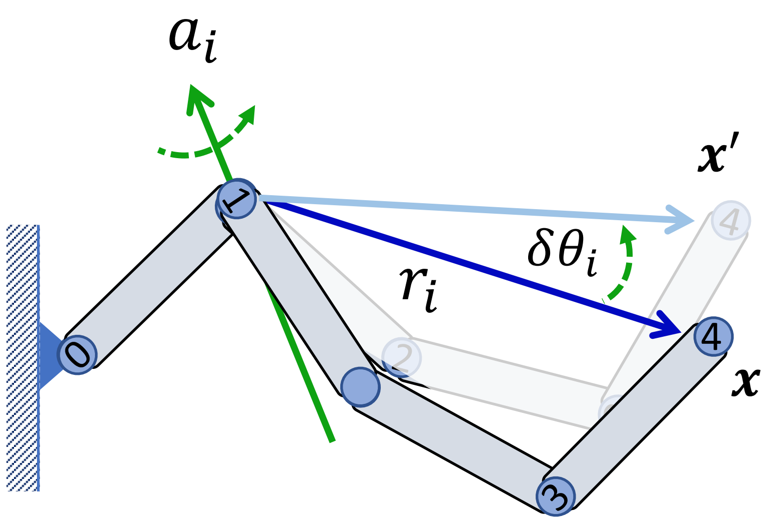
Rodrigues' rotation formula

$$\boldsymbol{x}' - \boldsymbol{x} = (\sin \delta\theta_i)\, \boldsymbol{a}_i \times \boldsymbol{r}_i + (1 - \cos \delta\theta_i)\boldsymbol{a}_i \times (\boldsymbol{a}_i \times \boldsymbol{r}_i)$$

Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

Assuming all joints are hinge joint



Rodrigues' rotation formula

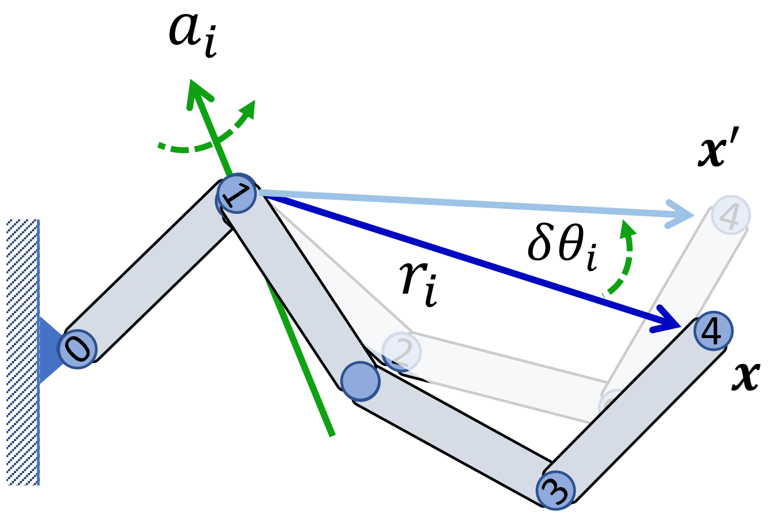
$$\boldsymbol{x}' - \boldsymbol{x} = (\sin \delta\theta_i)\, \boldsymbol{a}_i \times \boldsymbol{r}_i + (1 - \cos \delta\theta_i) \boldsymbol{a}_i \times (\boldsymbol{a}_i \times \boldsymbol{r}_i)$$

$$\frac{\partial f}{\partial \theta_i} = \lim_{\delta\theta_i \to 0} \frac{\boldsymbol{x}' - \boldsymbol{x}}{\delta\theta_i} = \boldsymbol{a}_i \times \boldsymbol{r}_i$$

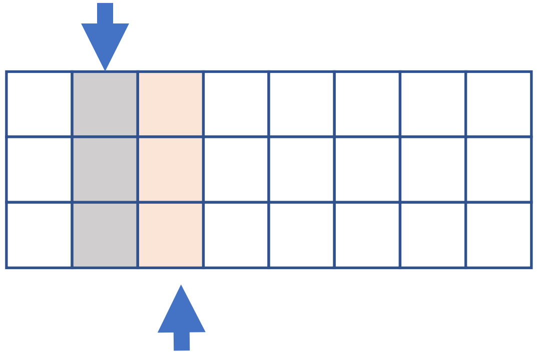
Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

Assuming all joints are hinge joint



$$\frac{\partial f}{\partial \theta_1} = \boldsymbol{a}_1 \times \boldsymbol{r}_1$$

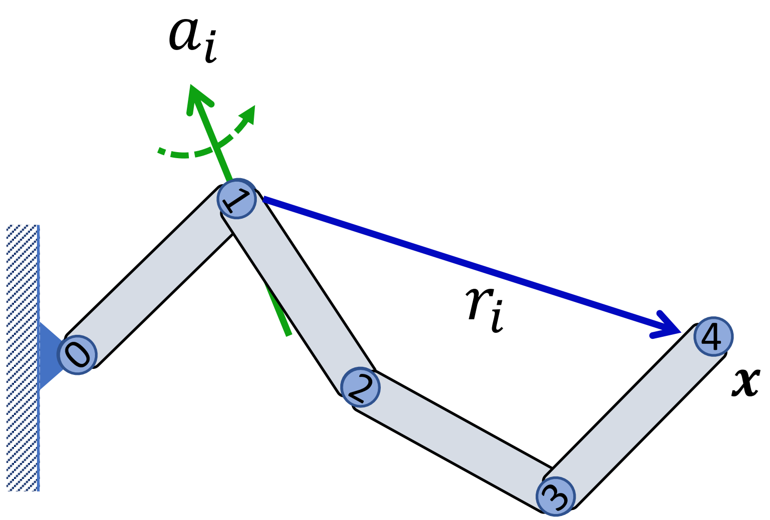


$$\frac{\partial f}{\partial \theta_2} = \boldsymbol{a}_2 \times \boldsymbol{r}_2$$

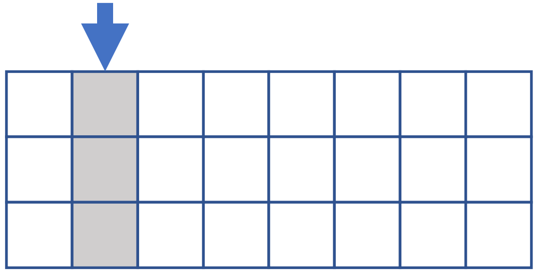
Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

How to deal with ball joints?



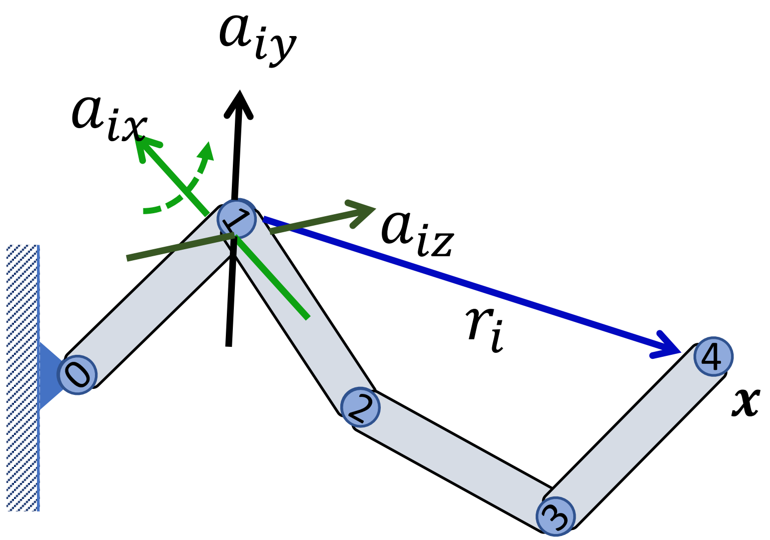
$$\frac{\partial f}{\partial \theta_i} = ???$$



Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

How to deal with ball joints?

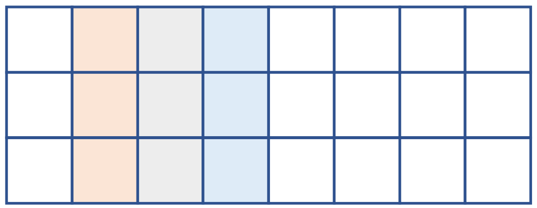


A ball joint parameterized as Euler angles:

$$R_i = R_{ix} R_{iy} R_{iz}$$

can be considered as a compound joint with three hinge joints

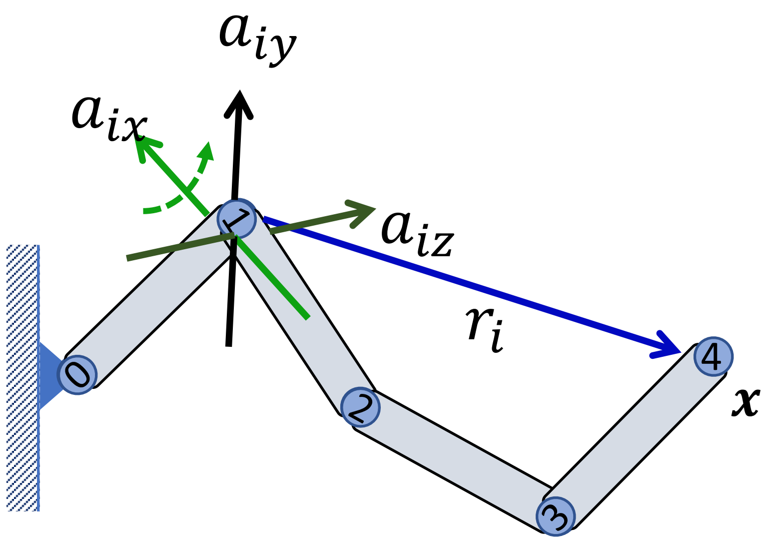
$$\frac{\partial f}{\partial \boldsymbol{\theta}_i} = \begin{pmatrix} \frac{\partial f}{\partial \theta_{ix}} & \frac{\partial f}{\partial \theta_{iy}} & \frac{\partial f}{\partial \theta_{iz}} \end{pmatrix}$$



Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

How to deal with ball joints?



A ball joint parameterized as Euler angles:

$$R_i = R_{ix}R_{iy}R_{iz}$$

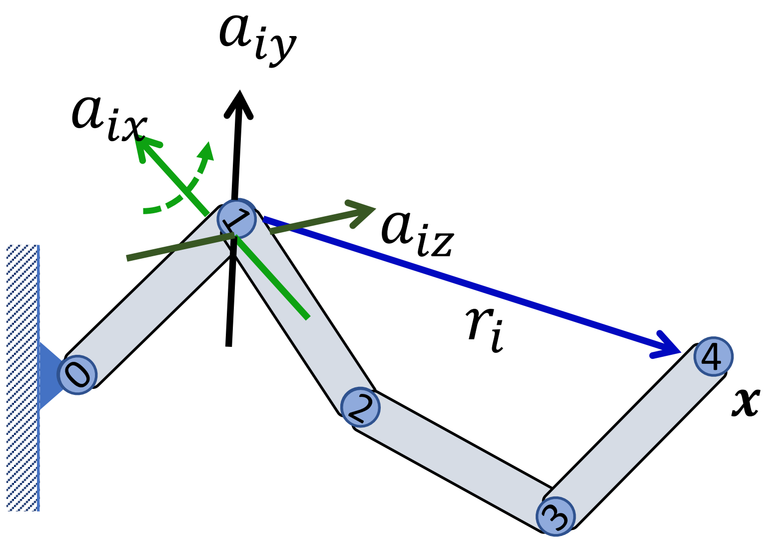
can be considered as a compound joint with three hinge joints

$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

How to deal with ball joints?



A ball joint parameterized as Euler angles:

$$R_i = R_{ix} R_{iy} R_{iz}$$

can be considered as a compound joint with three hinge joints

Note: rotation axes are

$$\boldsymbol{a}_{ix} = Q_{i-1} \boldsymbol{e}_x$$

$$\boldsymbol{a}_{iy} = Q_{i-1} R_{ix} \boldsymbol{e}_y$$

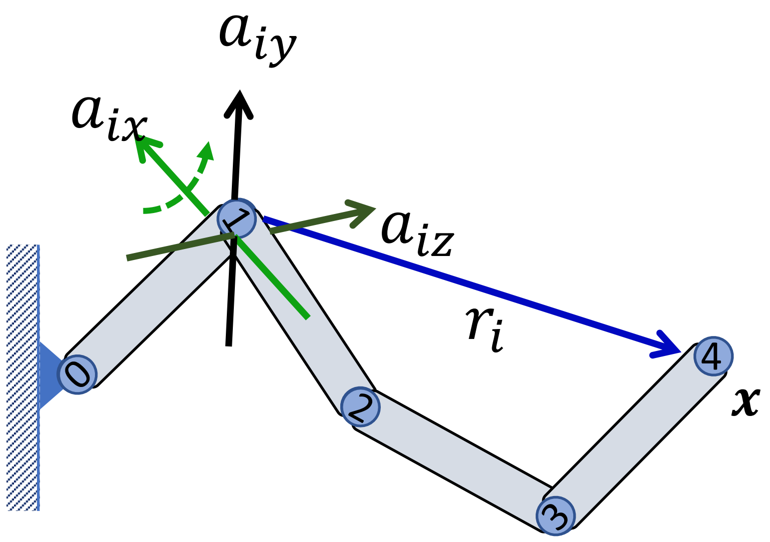
$$\boldsymbol{a}_{iz} = Q_{i-1} R_{ix} R_{iy} \boldsymbol{e}_z$$

$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Geometric Approach

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial f}{\partial \theta_0} & \frac{\partial f}{\partial \theta_1} & \cdots & \frac{\partial f}{\partial \theta_n} \end{pmatrix}$$

How to deal with ball joints?



A ball joint parameterized as Euler angles:

$$R_i = R_{ix} R_{iy} R_{ix'}$$

can be considered as a compound joint with three hinge joints

Note: rotation axes are

$$\boldsymbol{a}_{ix} = Q_{i-1} \boldsymbol{e}_x$$

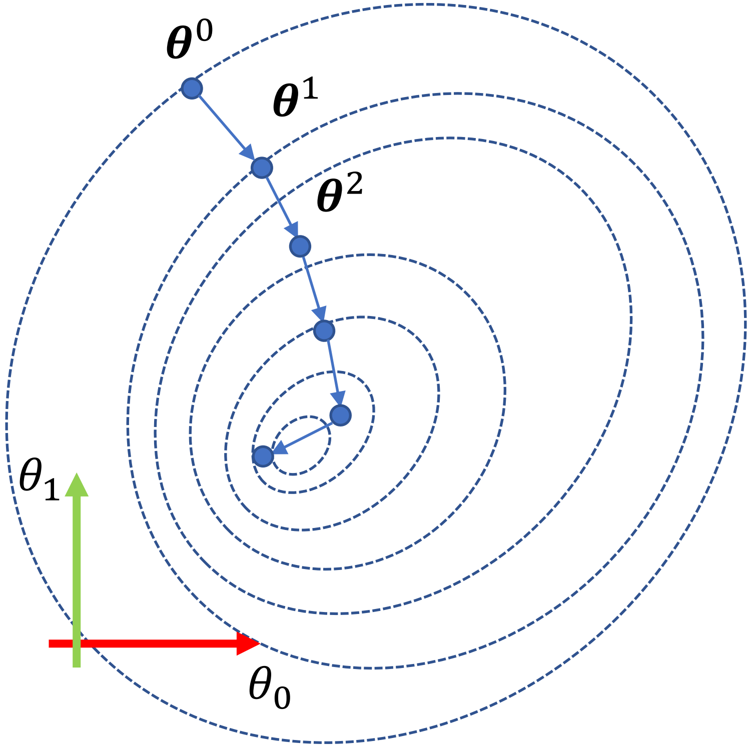
$$\boldsymbol{a}_{iy} = Q_{i-1} R_{ix} \boldsymbol{e}_y$$

$$\boldsymbol{a}_{ix'} = Q_{i-1} R_{ix} R_{iy} \boldsymbol{e}_{\boldsymbol{x}}$$

$$\frac{\partial f}{\partial \theta_{i*}} = \boldsymbol{a}_{i*} \times \boldsymbol{r}_i$$

Jacobian Transpose / Gradient Descent

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Update parameters against the direction of the gradient of the objective function

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \alpha \nabla_{\boldsymbol{\theta}} F\left(\boldsymbol{\theta}^i\right)$$

$$= \boldsymbol{\theta}^i - \alpha J^T \Delta$$

First-order approach, convergence can be slow

Need to re-compute Jacobian at each iteration

Optimality Condition

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

Gradient:

$$\nabla_\theta F(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial F}{\partial \theta_0}(\boldsymbol{\theta}) \\ \frac{\partial F}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial F}{\partial \theta_n}(\boldsymbol{\theta}) \end{bmatrix}$$

The direction in which $F(\theta)$ increases fastest

Optimality Condition

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

Gradient:

$$\nabla_\theta F(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial F}{\partial \theta_0}(\boldsymbol{\theta}) \\ \frac{\partial F}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial F}{\partial \theta_n}(\boldsymbol{\theta}) \end{bmatrix}$$

The direction in which $F(\theta)$ increases fastest

$\Rightarrow$

First-order optimality condition:

$\boldsymbol{\theta}^*$ is a local minimum of $F(\boldsymbol{\theta})$

$\Updownarrow$

$$\nabla_\theta F(\boldsymbol{\theta}^*) = 0$$

Example: Quadratic Programming

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T A\boldsymbol{\theta} + \boldsymbol{b}^T\boldsymbol{\theta}$$

where A is **positive definite**:

$$A = A^T, \boldsymbol{\theta}^T A\boldsymbol{\theta} \geq 0 \text{ for any } \boldsymbol{\theta}$$

Example: Quadratic Programming

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T A\boldsymbol{\theta} + \boldsymbol{b}^T\boldsymbol{\theta}$$

Gradient: $\nabla_\theta F(\boldsymbol{\theta}) = A\boldsymbol{\theta} + \boldsymbol{b}$

Example: Quadratic Programming

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T A\boldsymbol{\theta} + \boldsymbol{b}^T\boldsymbol{\theta}$$

Gradient: $\nabla_\theta F(\boldsymbol{\theta}) = A\boldsymbol{\theta} + \boldsymbol{b}$

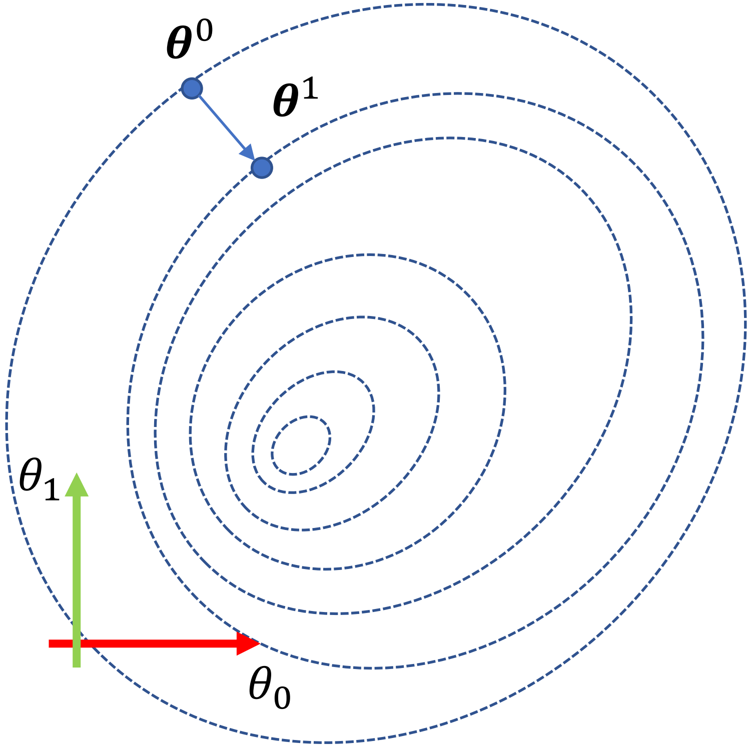
Optimality condition: $\nabla_\theta F(\boldsymbol{\theta}^*) = 0$

$$\Downarrow$$

$$\boldsymbol{\theta}^* = -A^{-1}\boldsymbol{b}$$

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



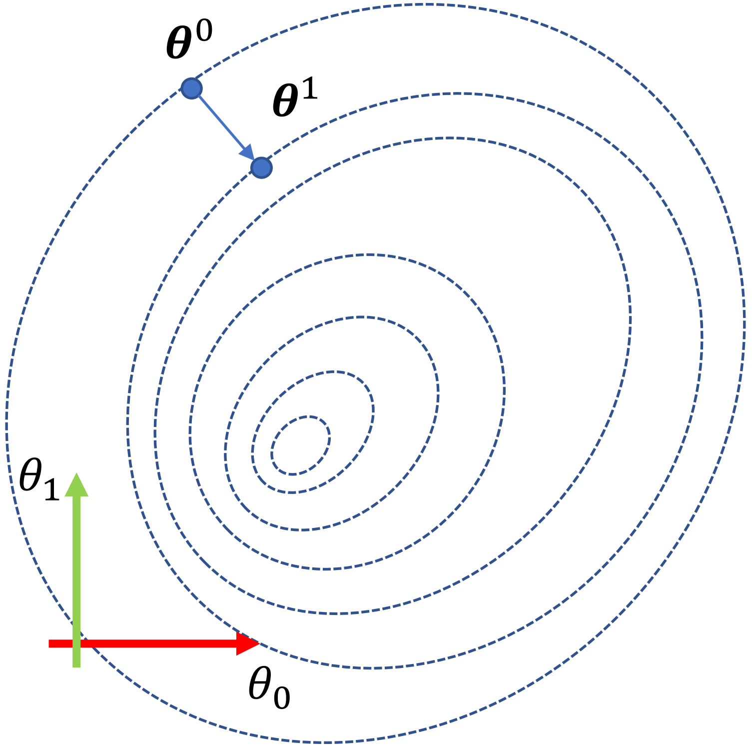
Consider the first-order approximation of $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^0$

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^0) + \frac{\partial f}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^0)(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$

$$= f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \widetilde{\boldsymbol{x}}\|_2^2$$



Consider the first-order approximation of $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^0$

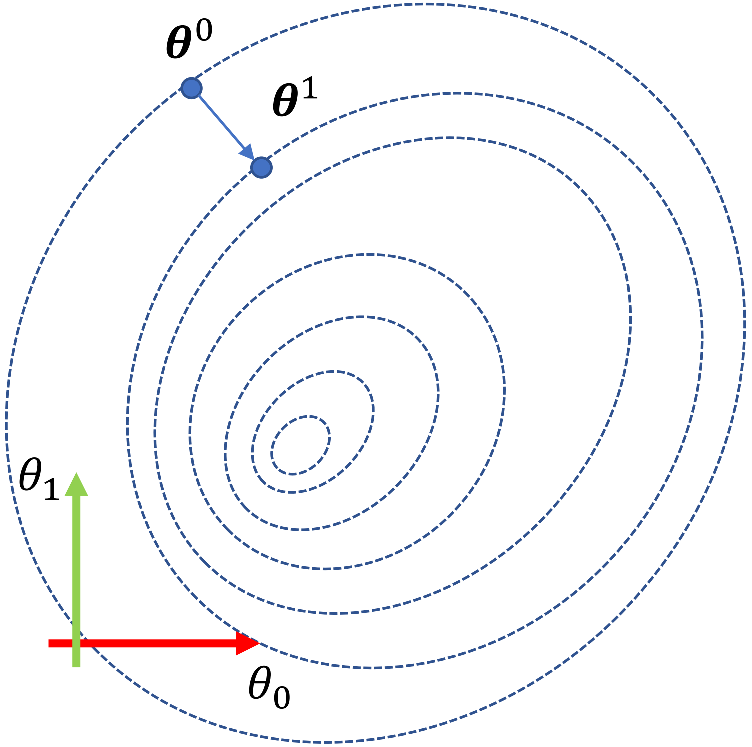
$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$

$$F(\theta) \approx \frac{1}{2}\left\|f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) - \widetilde{\boldsymbol{x}}\right\|_2^2$$

$$= \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^0)^T J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$
$$+ (\boldsymbol{\theta} - \boldsymbol{\theta}^0)^T J^T\left(f(\boldsymbol{\theta}^0) - \widetilde{\boldsymbol{x}}\right) + \boldsymbol{c}$$

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Consider the first-order approximation of $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^0$

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$

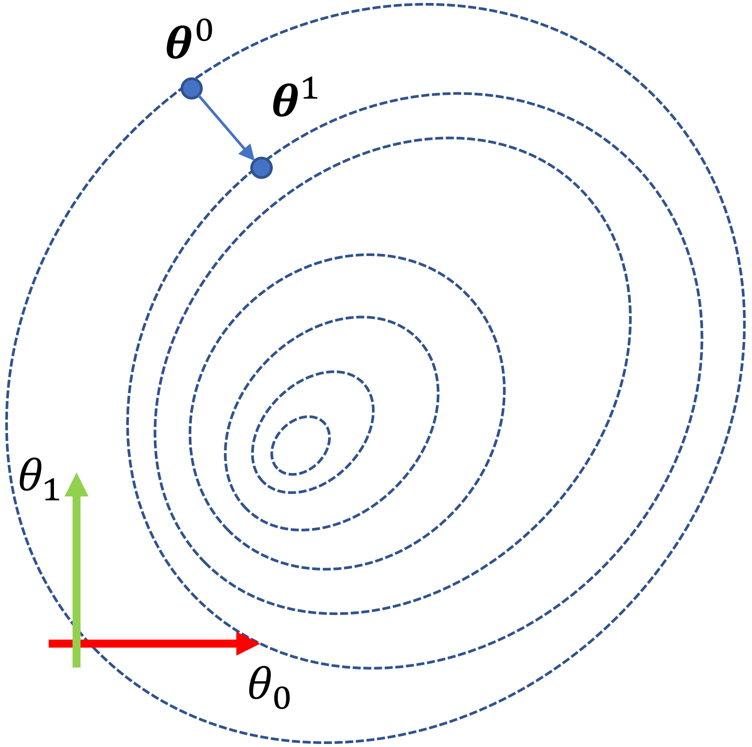
$$F(\theta) \approx \frac{1}{2}\left\|f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) - \tilde{\boldsymbol{x}}\right\|_2^2$$

$$\left(\nabla F(\theta)\right)^T = J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) + J^T\left(f(\boldsymbol{\theta}^0) - \tilde{\boldsymbol{x}}\right) = \mathbf{0}$$

first-order optimality condition

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



Consider the first-order approximation of $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}^0$

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0)$$

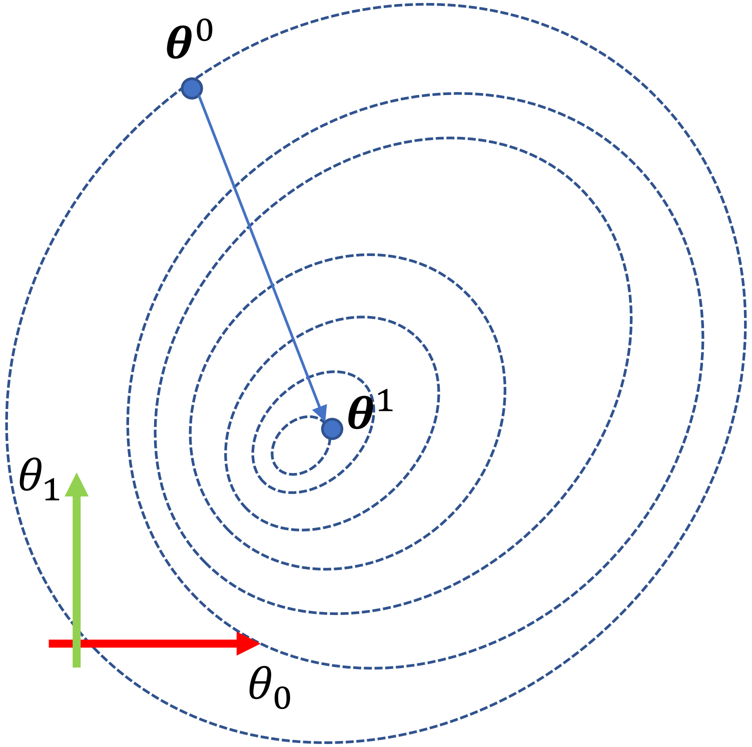
$$F(\theta) \approx \frac{1}{2}\left\|f(\boldsymbol{\theta}^0) + J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) - \tilde{\boldsymbol{x}}\right\|_2^2$$

$$J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta$$

first-order optimality condition

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \widetilde{\boldsymbol{x}}\|_2^2$$



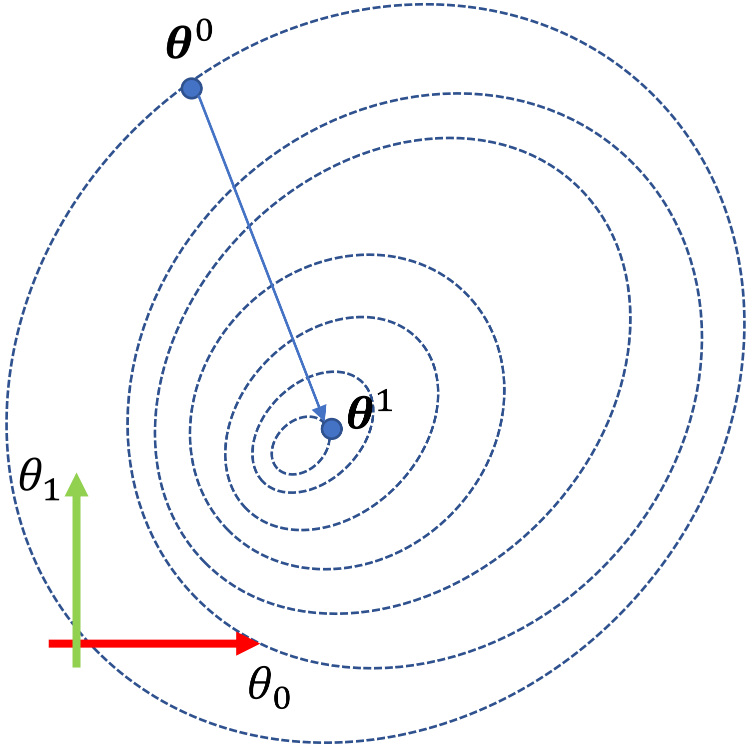
$$\boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \left(J^T J\right)^{-1} J^T \Delta$$

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



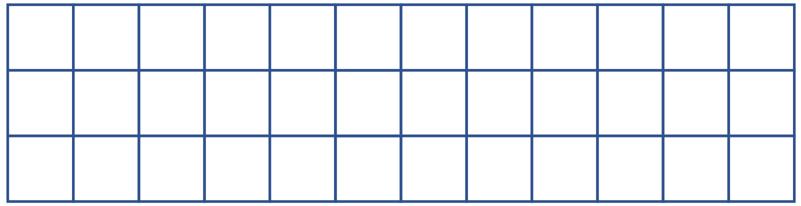
$$\boxed{J^T J\left(\boldsymbol{\theta} - \boldsymbol{\theta}^0\right) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \cancel{\boldsymbol{\theta}^0 - \left(J^T J\right)^{-1}} J^T \Delta$$

however...

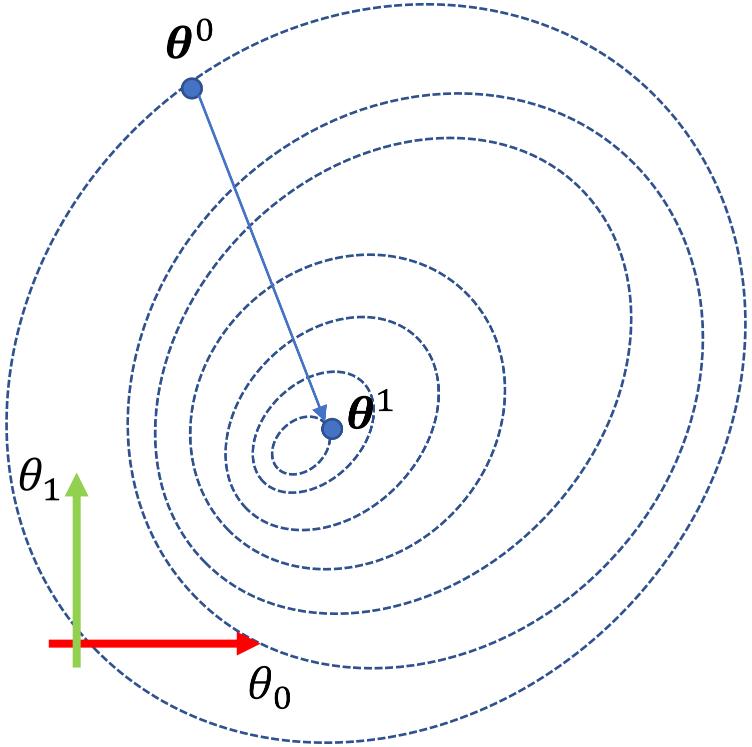
$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



$J^T J$ is **NOT** invertible

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



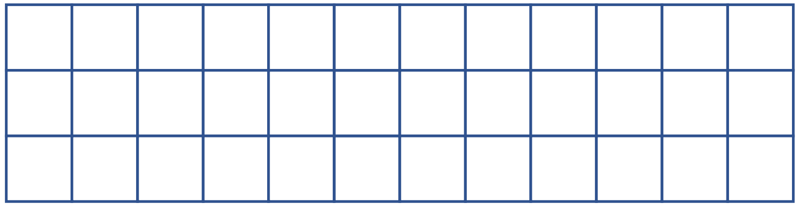
$$\boxed{J^T J\left(\boldsymbol{\theta} - \boldsymbol{\theta}^0\right) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \cancel{\boldsymbol{\theta}^0 - \left(J^T J\right)^{-1}} J^T \Delta$$

however...

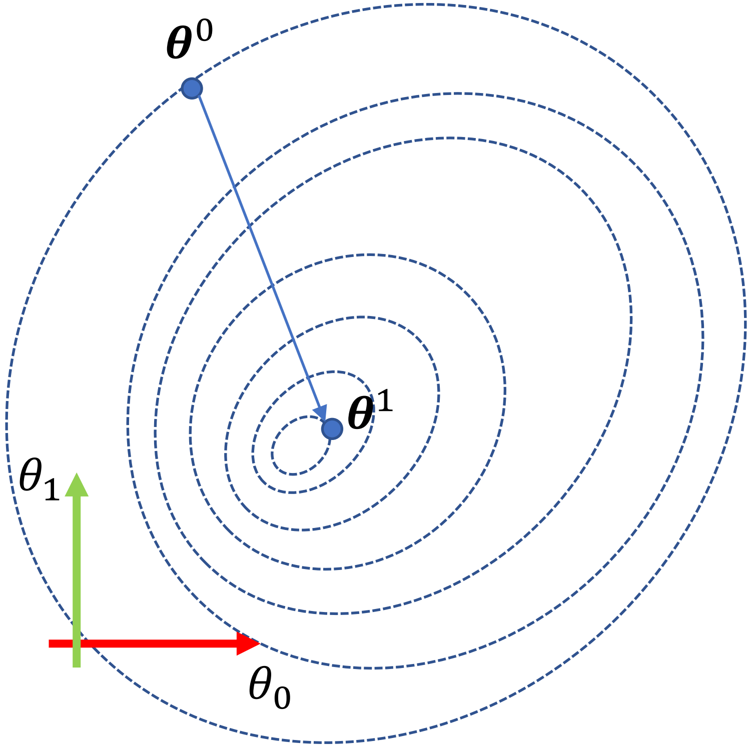
$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



$J^T J$ is **NOT** invertible, but $J J^T$ can be invertible

Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$

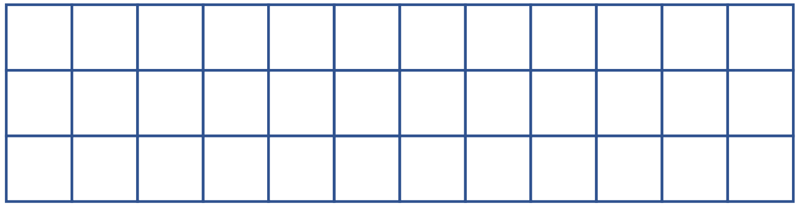


$$J \times \boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

Assume JJ^T is invertible

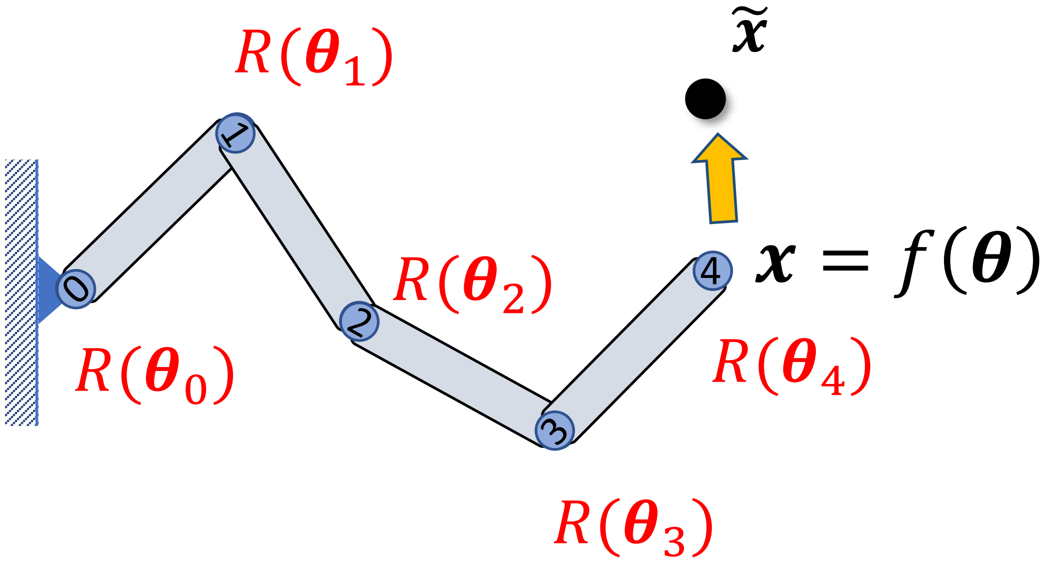
$$J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -\Delta$$

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$J \times \quad \boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

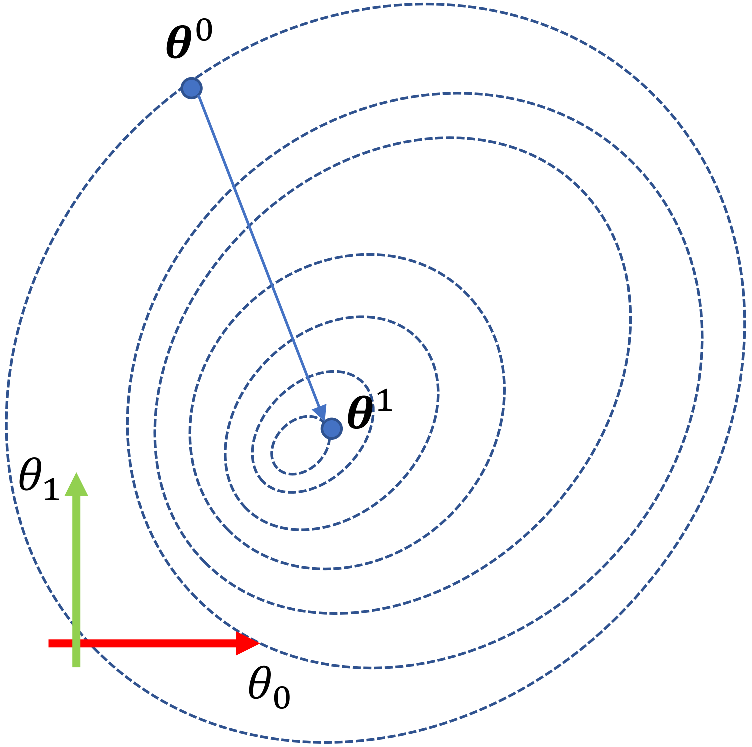
Assume JJ^T is invertible

$$J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = \tilde{\boldsymbol{x}} - f(\boldsymbol{\theta}^0)$$

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$

Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$J \times \quad \boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

Assume JJ^T is invertible

$$J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -\Delta$$

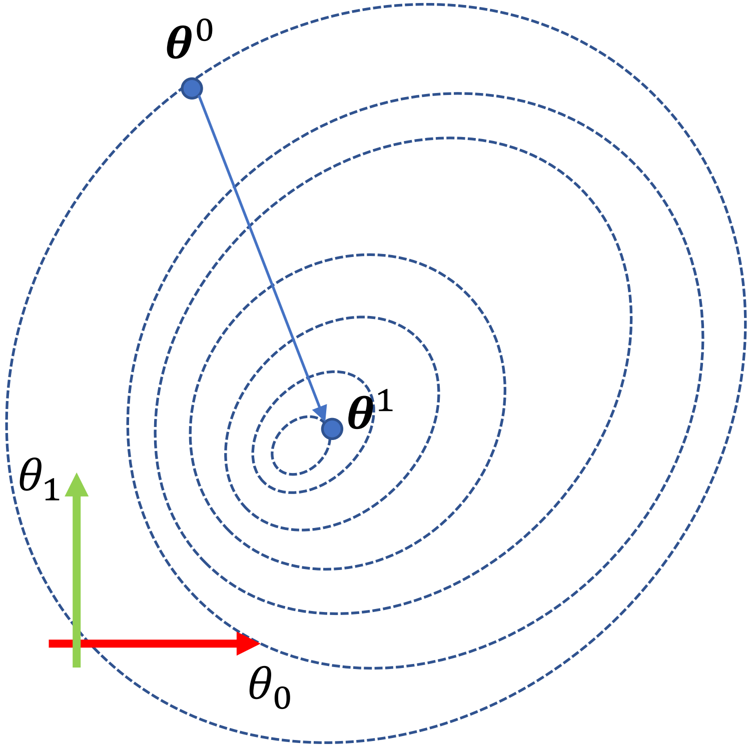
$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - J^+ \Delta$$

$$= \boldsymbol{\theta}^0 - J^T (JJ^T)^{-1} \Delta$$

(Moore-Penrose) Pseudoinverse

Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$

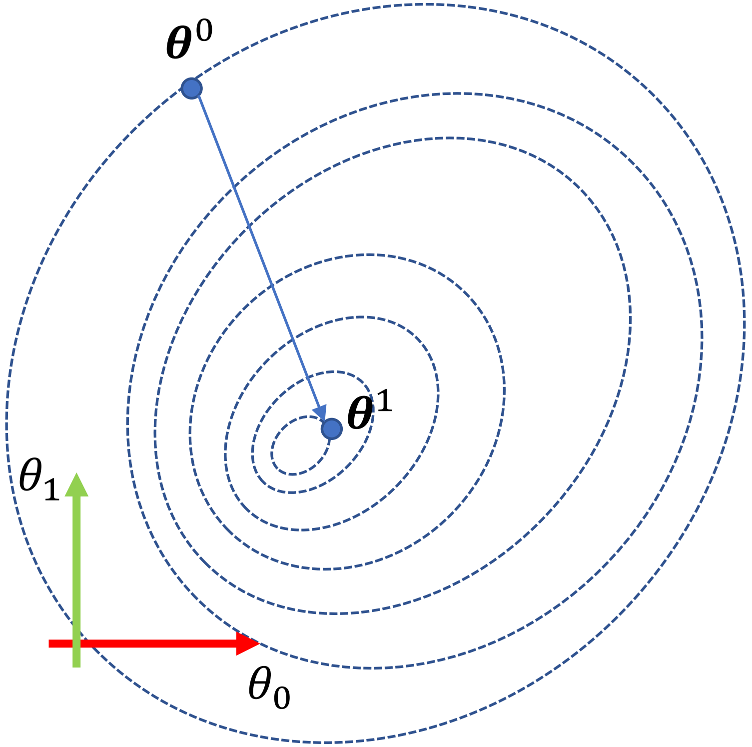


$$\begin{aligned}\boldsymbol{\theta} &= \boldsymbol{\theta}^0 - J^{+}\Delta \\ &= \boldsymbol{\theta}^0 - J^{\mathrm{T}}\left(\mathrm{JJ}^{\mathrm{T}}\right)^{-1}\Delta\end{aligned}$$

(Moore-Penrose) Pseudoinverse

Gauss-Newton Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$\boxed{J^T J\left(\boldsymbol{\theta} - \boldsymbol{\theta}^0\right) = -J^T \Delta}$$

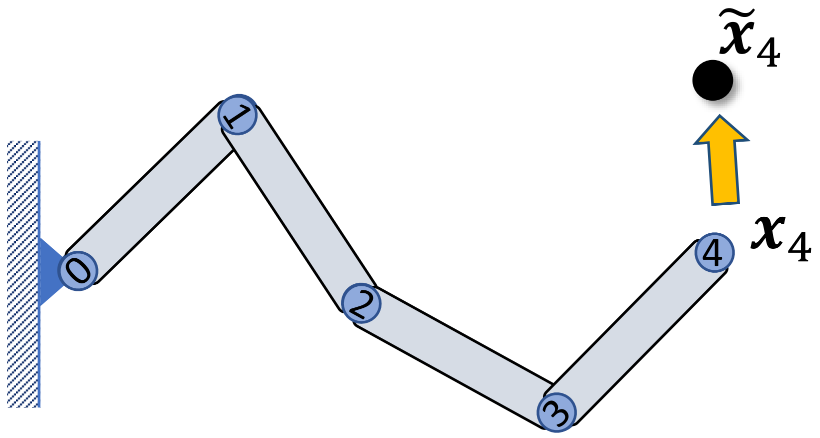
If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \left(J^T J\right)^{-1} J^T \Delta$$

but when can $J^T J$ be invertible?

Gauss-Newton Method

Assuming all joints are hinge joint



$$\boldsymbol{x}_4 = f(\boldsymbol{\theta}) \in \mathbb{R}^9$$

$$\boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

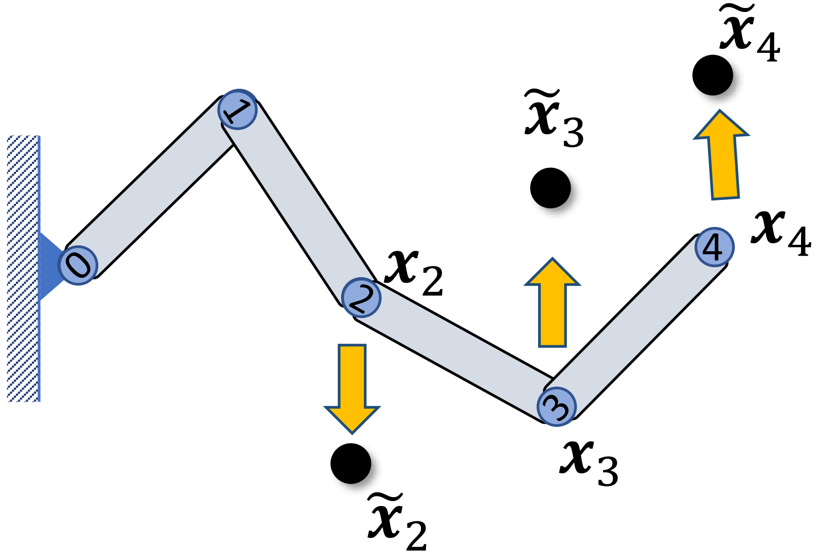
$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \left(J^T J\right)^{-1} J^T \Delta$$

but when can $J^T J$ be invertible?

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$

Gauss-Newton Method

Assuming all joints are hinge joint



$$\begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix} = f(\boldsymbol{\theta}) \in \mathbb{R}^9$$

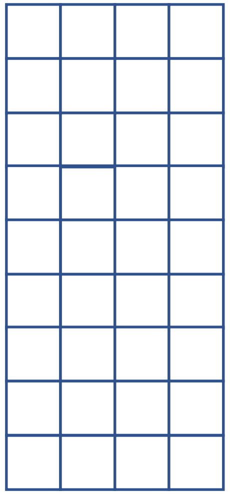
$$\boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \left(J^T J\right)^{-1} J^T \Delta$$

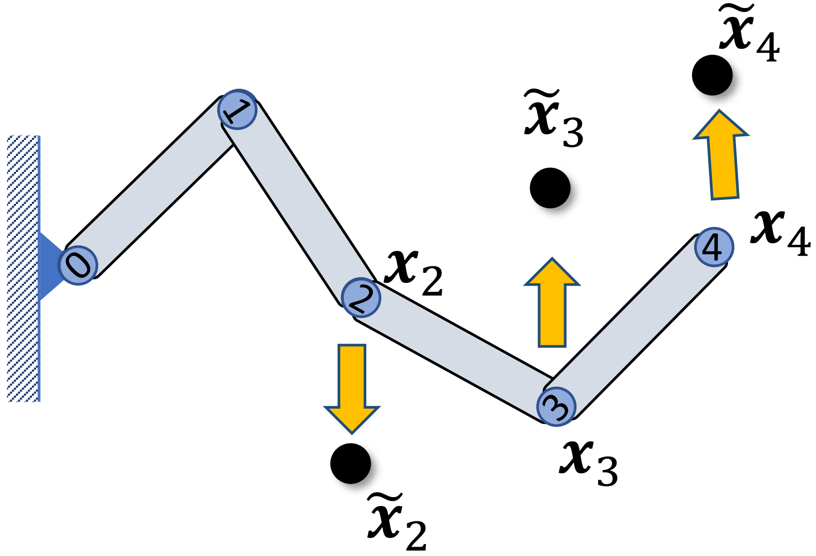
but when can $J^T J$ be invertible?

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



Gauss-Newton Method

Assuming all joints are hinge joint



$$\begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix} = f(\boldsymbol{\theta}) \in \mathbb{R}^9$$

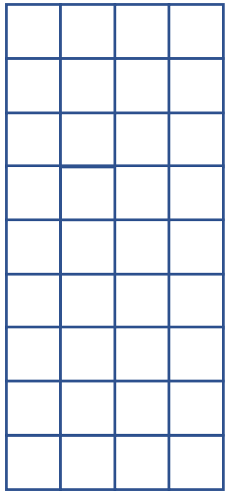
$$\boxed{J^T J(\boldsymbol{\theta} - \boldsymbol{\theta}^0) = -J^T \Delta}$$

If $J^T J$ is invertible, we have

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - (J^T J)^{-1} J^T \Delta = \boldsymbol{\theta}^0 - J^+ \Delta$$

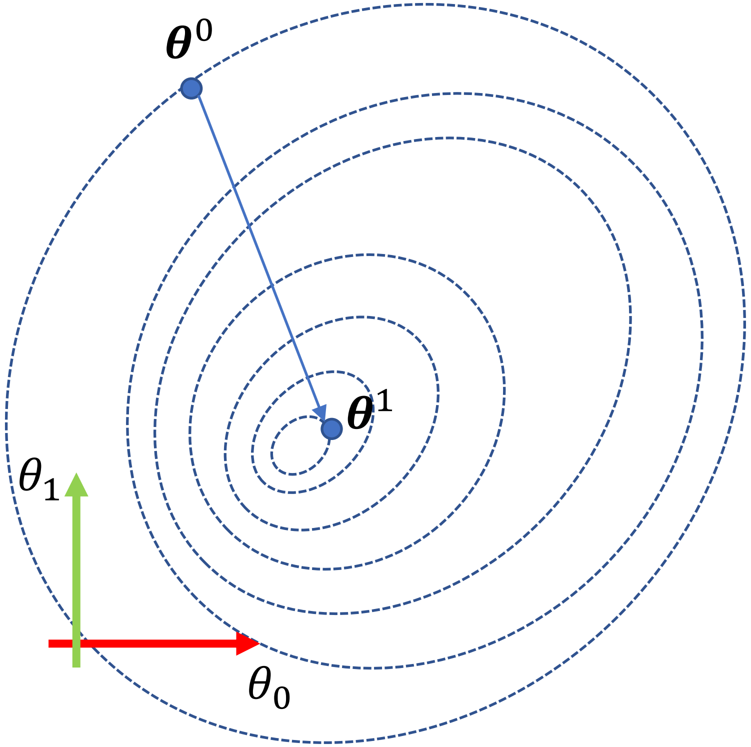
(Moore-Penrose) Pseudoinverse

$$J = \frac{\partial f}{\partial \boldsymbol{\theta}} =$$



Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$

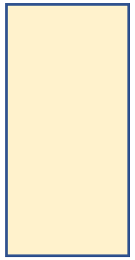


$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - J^+\Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$ 

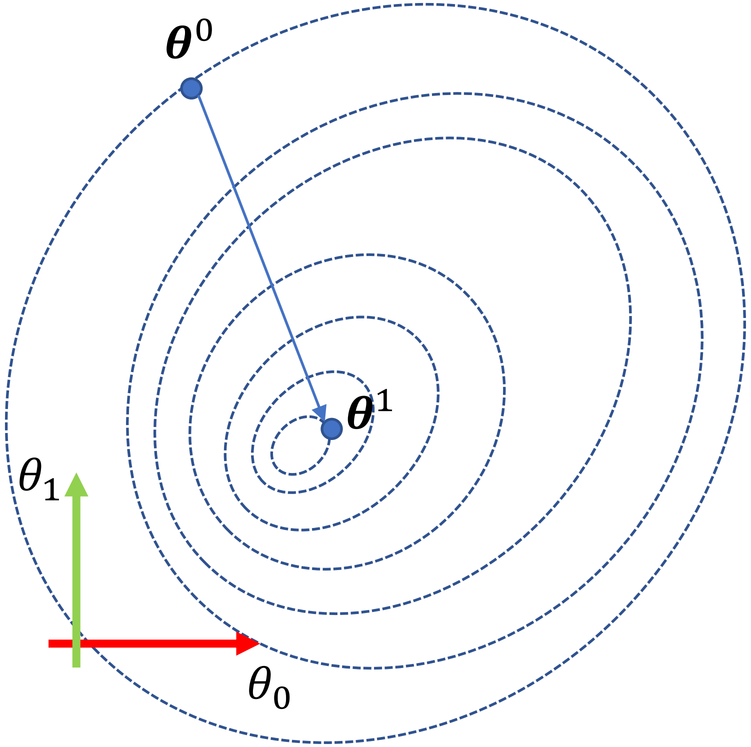
$$J^+ = J^{\mathrm{T}}\left(\mathrm{JJ}^{\mathrm{T}}\right)^{-1}$$

when $J =$ 

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

Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^+ \Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$

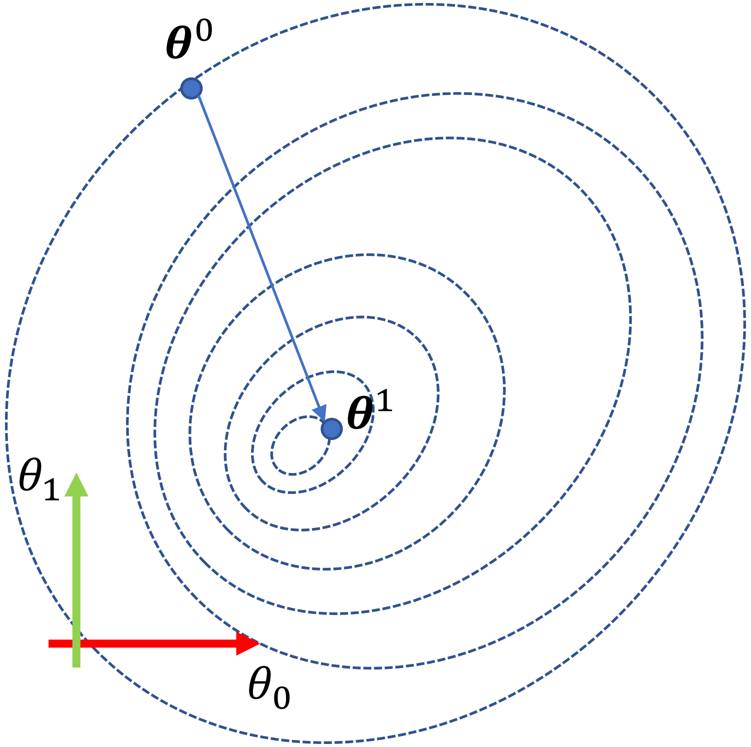
$$J^+ = J^{\mathrm{T}}(\mathrm{J}\mathrm{J}^{\mathrm{T}})^{-1}$$

when $J =$

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

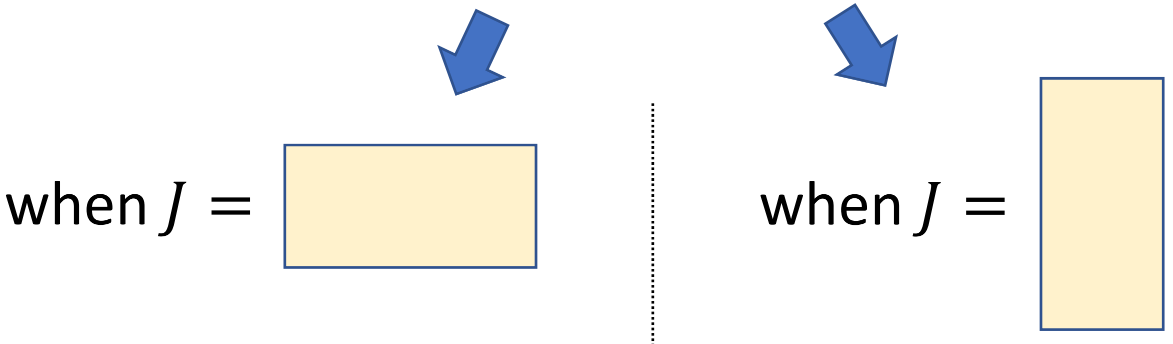
Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^+ \Delta$$

(Moore-Penrose) Pseudoinverse



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

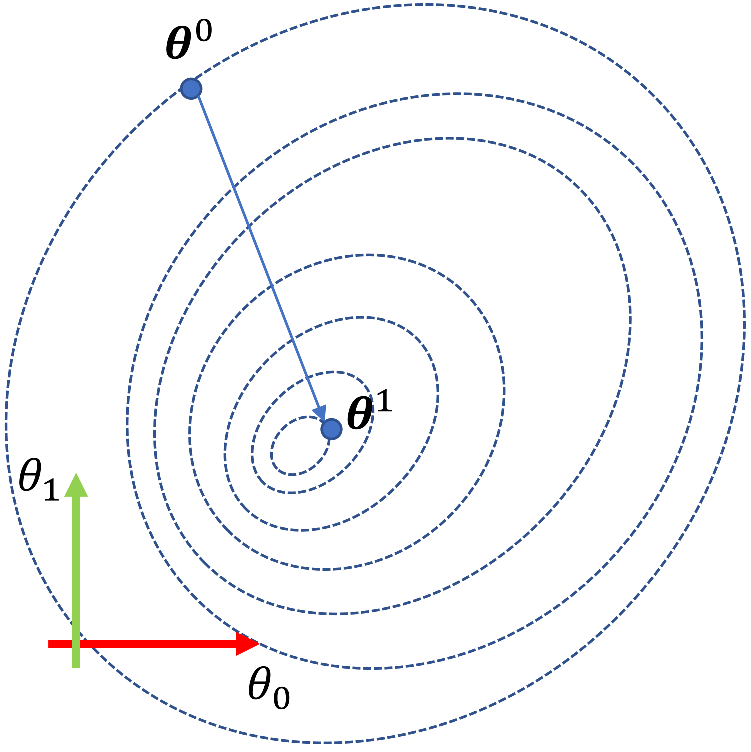
$$J^+ = \left(J^T J\right)^{-1} J^T$$

Usually faster than gradient descent/Jacobian transpose method.

Any problem?

Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2$$



$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^+ \Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$

$$J^+ = J^T(JJ^T)^{-1}$$

when $J =$

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

Usually faster than gradient descent/Jacobian transpose method.

Any problem? JJ^T / $J^T J$ can be (near) singular!

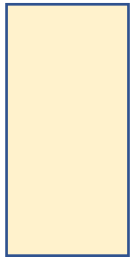
Damped Jacobian Inverse Method

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^+ \Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$ 

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

when $J =$ 

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

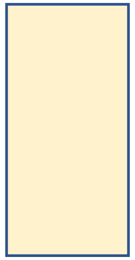
Damped Jacobian Inverse Method

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^* \Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$ 

$$J^* = J^{\mathrm{T}}\left(\mathrm{J}\mathrm{J}^{\mathrm{T}} + \lambda I\right)^{-1}$$

when $J =$ 

$$J^* = \left(J^T J + \lambda I\right)^{-1} J^T$$

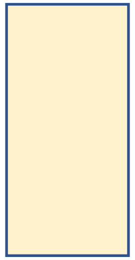
Damped Jacobian Inverse Method

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \alpha J^* \Delta$$

(Moore-Penrose) Pseudoinverse

when $J =$ 

$$J^* = J^{\mathrm{T}}\left(JJ^{\mathrm{T}} + \lambda I\right)^{-1}$$

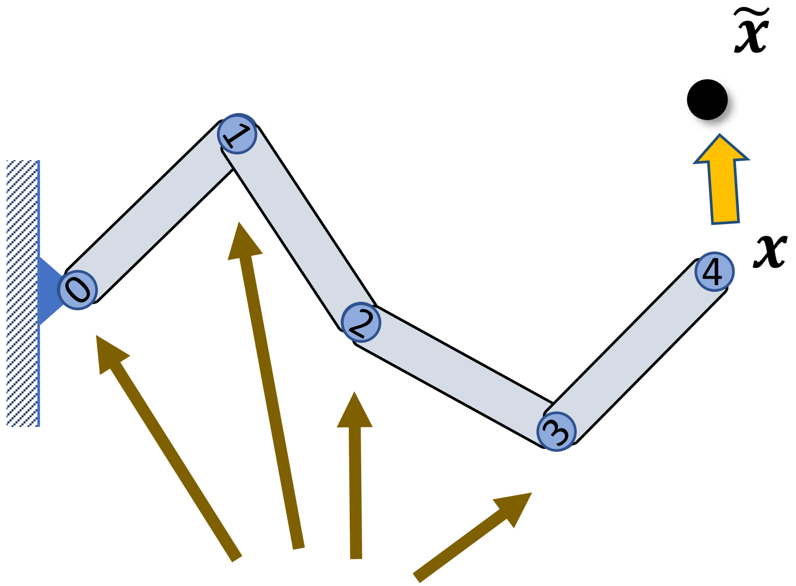
when $J =$ 

$$J^* = \left(J^T J + \lambda I\right)^{-1} J^T$$

Also called Levenberg-Marquardt algorithm

Damped Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2 + \frac{\lambda}{2}\left\|\boldsymbol{\theta} - \boldsymbol{\theta}^{\boldsymbol{i}}\right\|_2^2$$



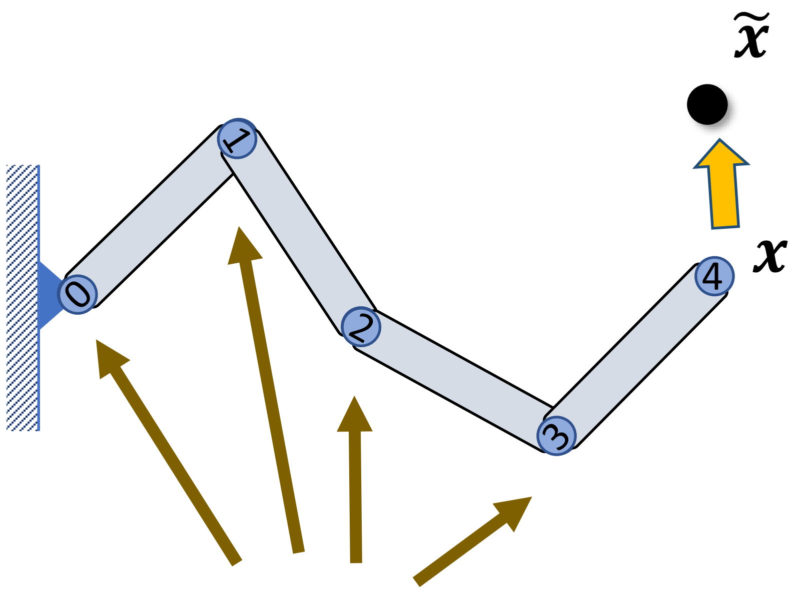
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha\left(J^T J + \lambda I\right)^{-1} J^T \Delta$$

$\boldsymbol{\lambda}$: damping parameter

Using the minimal rotations to reach the target

Damped Jacobian Inverse Method

$$F(\theta) = \frac{1}{2}\|f(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}\|_2^2 + \frac{\lambda}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\boldsymbol{i}})^{\boldsymbol{T}} W(\boldsymbol{\theta} - \boldsymbol{\theta}^{\boldsymbol{i}})$$



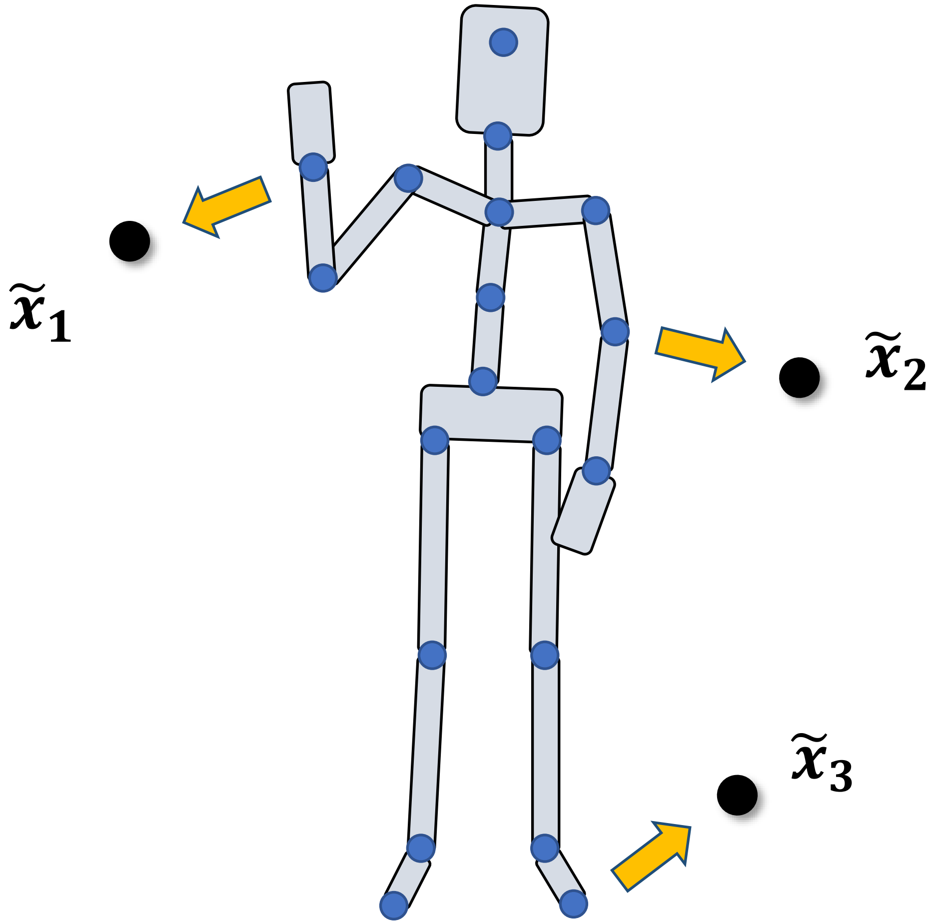
$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^{i} - \alpha\left(J^T J + \lambda W\right)^{-1} J^T \Delta$$

$\boldsymbol{\lambda}$: damping parameter

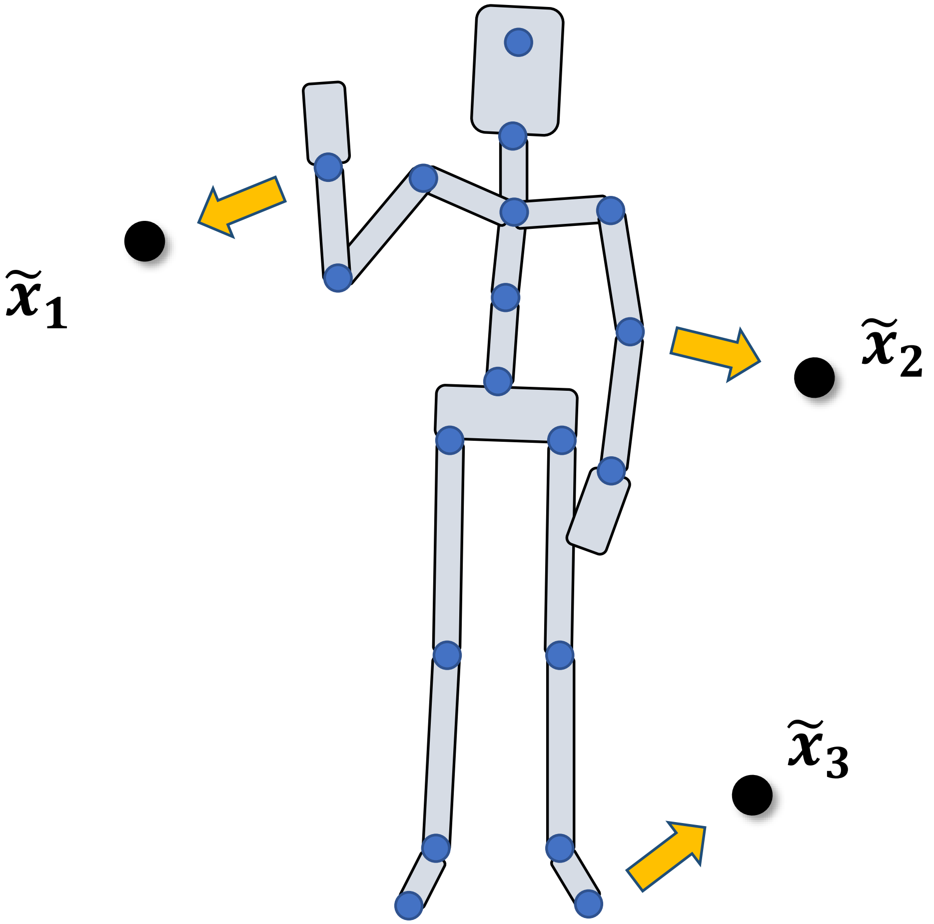
$$W = \begin{bmatrix} w_0 & & & \\ & w_1 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}$$: weight matrix

Using the minimal rotations to reach the target

Character IK



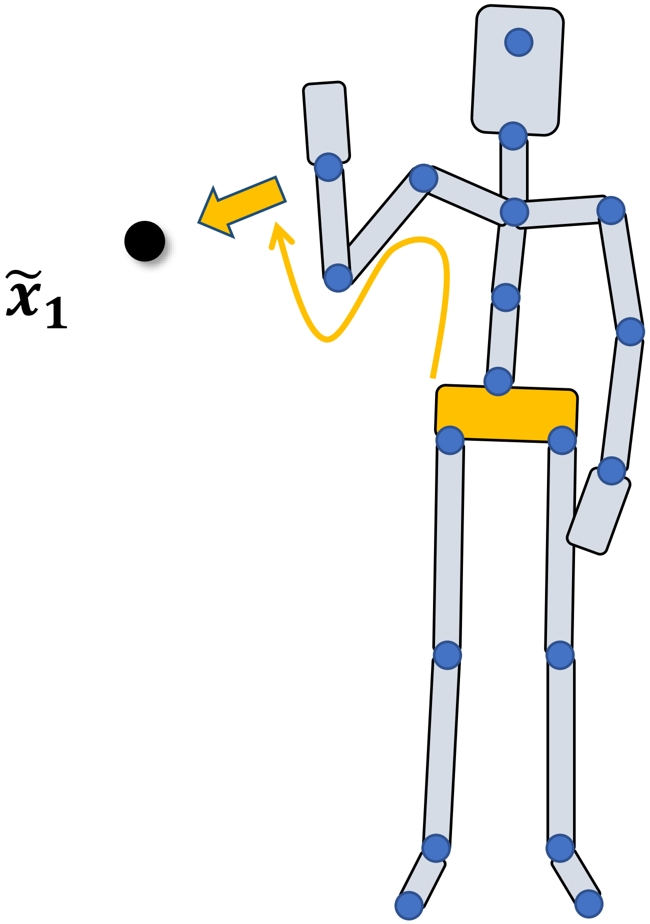
Character IK



$$F(\theta) = \frac{1}{2}\sum_i \|f_i(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

$$\boldsymbol{\theta} = (\boldsymbol{t}_0, R_0, R_1, R_2, \dots\dots)$$

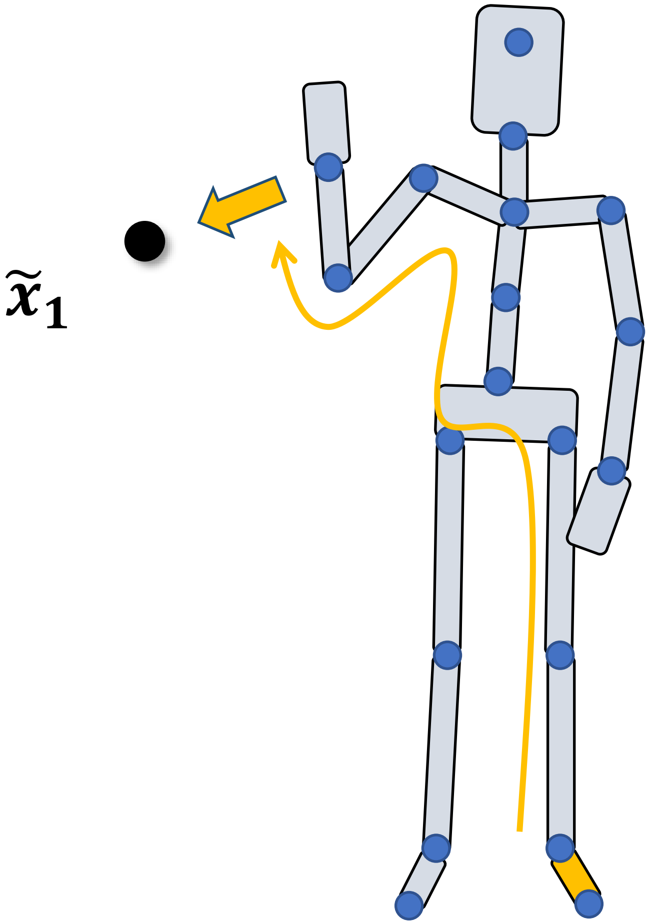
Character IK



$$F(\theta) = \frac{1}{2}\sum_i \|f_i(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

$$\boldsymbol{\theta} = (\boldsymbol{t}_0, R_0, R_1, R_2, \dots\dots)$$

Character IK



$$F(\theta) = \frac{1}{2}\sum_i \|f_i(\boldsymbol{\theta}) - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

$$\boldsymbol{\theta} = (\boldsymbol{t}_0, R_0, R_1, R_2, \dots\dots)$$

Outline

- Character Kinematics
  - Skeleton and forward Kinematics

- Inverse Kinematics
  - IK as a optimization problem
  - Optimization approaches
    - Cyclic Coordinate Descent (CCD)
    - Jacobian and gradient descent method
    - Jacobian inverse method

Outline

- Character Kinematics
  - Skeleton and forward Kinematics
- Inverse Kinematics
  - IK as a optimization problem
  - Optimization approaches
    - Cyclic Coordinate Descent (CCD)
    - Jacobian and gradient descent method
    - Jacobian inverse method

Graphical Models 73 (2011) 243–260

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

FABRIK: A fast, iterative solver for the Inverse Kinematics problem ☆

Andreas Aristidou *, Joan Lasenby

Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK

ARTICLE INFO

Article history:
Received 12 February 2010
Received in revised form 24 March 2011
Accepted 9 May 2011
Available online 15 May 2011

Keywords:
Human animation
Inverse Kinematics
Joint configuration
Motion reconstruction

ABSTRACT

Inverse Kinematics is defined as the problem of determining a set of appropriate joint configurations for which the end effectors move to desired positions as smoothly, rapidly, and as accurately as possible. However, many of the currently available methods suffer from high computational cost and production of unrealistic poses. In this paper, a novel heuristic method, called Forward And Backward Reaching Inverse Kinematics (FABRIK), is described and compared with some of the most popular existing methods regarding reliability, computational cost and conversion criteria. FABRIK avoids the use of rotational angles or matrices, and instead finds each joint position via locating a point on a line. Thus, it converges in few iterations, has low computational cost and produces visually realistic poses. Constraints can easily be incorporated within FABRIK and multiple chains with multiple end effectors are also supported.

1. Introduction

This paper addresses the problem of manipulating articulated figures in an interactive and intuitive fashion for the design and control of their posture. This problem finds its application in the areas of robotics, computer animation, ergonomics and gaming. In computer graphics, articulated figures are a convenient model for humans, animals or other virtual creatures from films and video games. Inverse Kinematics (IK) has also been used in rehabilitation medicine in order to observe asymmetries or abnormalities. The most popular method for animating such models is motion-capture; however, despite the availability of highly sophisticated techniques and expensive tools, many problems appear when dealing with complex figures. Most virtual character models are very complex; they are made up of many joints giving a system with a large number of degrees of freedom, thus, it is often difficult to produce a realistic character animation.

Inverse Kinematics is a method for computing the posture via estimating each individual degree of freedom in order to satisfy a given task that meets user constraints; it plays an important role in the computer animation and simulation of articulated figures. This paper presents a new heuristic iterative method, Forward And Backward Reaching Inverse Kinematics (FABRIK), for solving the IK problem in different scenarios. FABRIK uses a forward and backward iterative approach, finding each joint position via locating a point on line. FABRIK has been utilised in highly complex systems with single and multiple targets, with and without joint restrictions. It can easily handle end effector orientations and support, to the best of our knowledge, all chain classes. A reliable method for incorporating constraints is also presented and utilised within FABRIK. The proposed method retains all the advantages of FABRIK, producing visually smooth movements without oscillations and discontinuities. Several experiments have been implemented for comparison purposes between the most popular manipulator solvers, including multiple end effectors with multiple tasks, and highly constrained joints. The proposed algorithm is very efficient both in simple and complex problems resulting in similar or even better poses than highly sophisticated methods, requiring

☆ This paper has been recommended for acceptance by Jarek Rossignac.

* Corresponding author. Fax: +44 1223332662.
E-mail address: aa462@cam.ac.uk (A. Aristidou).

1524-0703/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2011.05.003

Andreas Aristidou and Joan Lasenby. 2011.
FABRIK: A fast, iterative solver for the Inverse Kinematics problem.
Graphical Models

Questions?