DTU

Software for Autonomous Systems

SFfAS-31391:

Robot Kinematics, Trajectory Planning Motion Planning and Execution

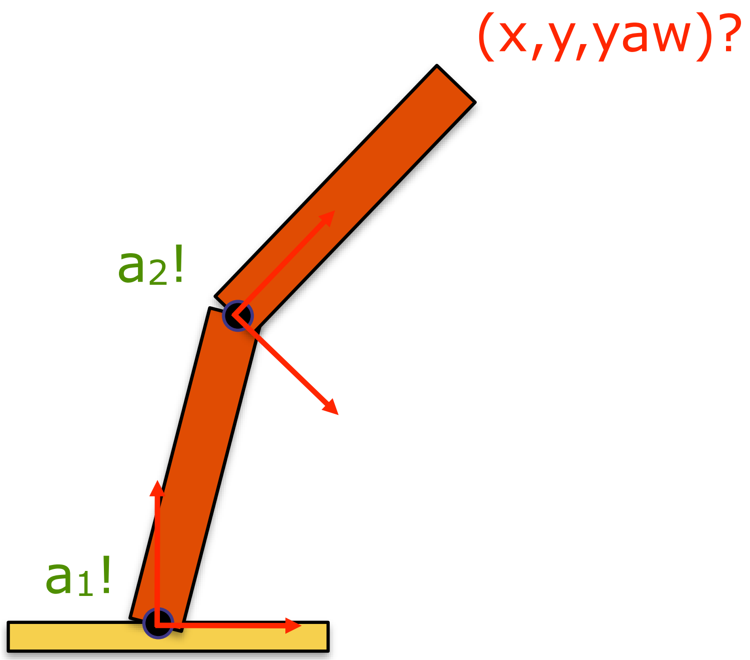
Lecturer, Course Coordinator: Evangelos Boukas—PhD

Outline of today

- Summary of the exercises from last time
- Kinematics of robot arms
    - Describing kinematics
    - Forward kinematics
    - Inverse kinematics
- Trajectories for robot arms

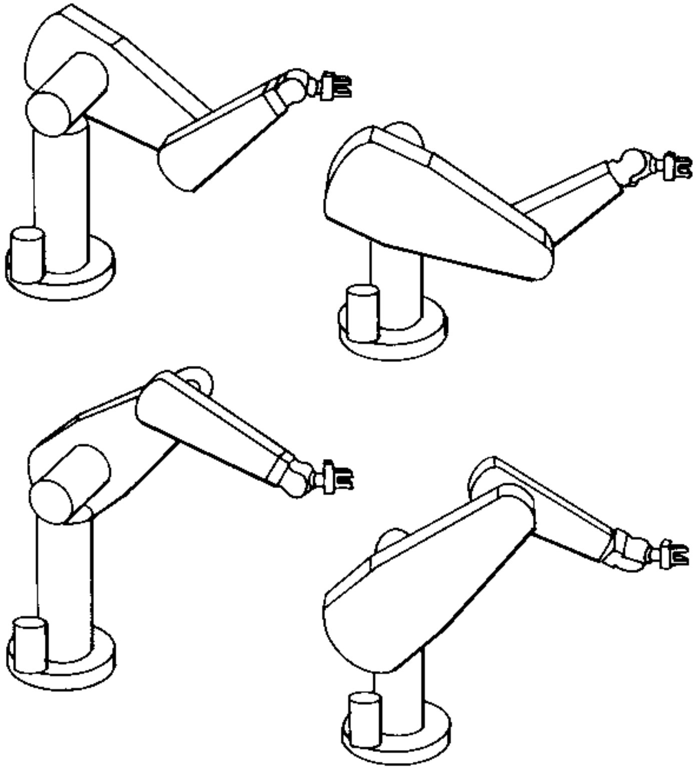
One-slide kinematics

- Kinematics is the “equations of motion”
- No regard to forces that cause the motion
- Our goal is to be able to use the robot in *Cartesian* coordinates
- Let’s consider a simple robot
  - If we know the joint angles, where is the end-effector? How is the end-effector oriented? = **Forward Kinematics**
- Forward kinematics only have 1 solution
  Why?



One-slide kinematics, slide 2

- A more complicated case is this:
  - Given a desired (x,y,yaw), what should the joint angles be?
    = **Inverse Kinematics**
- Inverse kinematics can have multiple solutions!
- How many solutions for this case?
- How about this one?
- Commonly 4 solutions for many robot arms



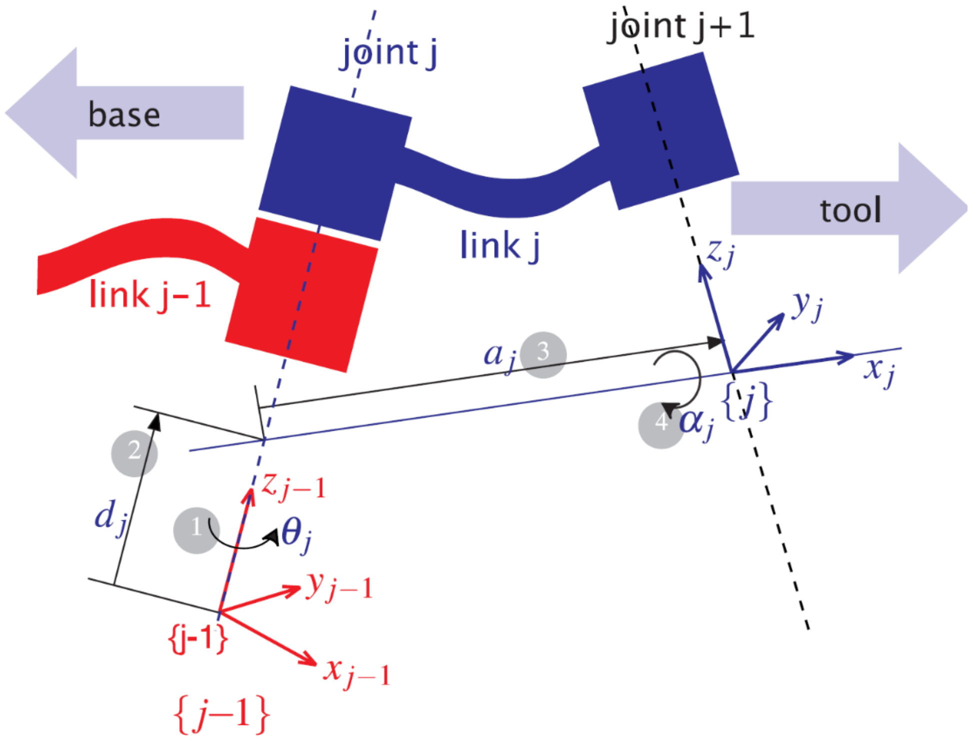
)!

Describing robot arm kinematics

- Deals with describing the chain of transformations between the links in the robot arm
- Some convention is needed
- The standard is Denavit-Hartenberg parameters (from 1955)
- Describes robot kinematics using 4 parameters for each joint
- Remember there can be different types of joints, and arms
  - “RRRRRR” - articulated robot, with only rotational joints
  - “RRPRRR” - this robot has one prismatic joint
- In the toolbox, one extra parameter defines the type of each joint
  - 0 = rotational
  - 1 = prismatic

DH parameters

- Robot has *N* joints, and *N+1* links
- Joint *j* connects link *j-1* to link *j*
  - Joint *j* moves link *j*
- Link described by two parameters:
  - Length - a_j (sometimes called r_j)
  - Twist - α_j
- Joint described by two parameters:
  - Link offset - d_j
  - Joint angle - θ_j
- Joint 1 connects Link 0 (the base of the robot) to Link 1
- Joint N connects Link N-1 to Link N (the end-effector of the robot)



DH parameters

Joint angle	θ_j	the angle between the x_{j-1} and x_j axes about the z_{j-1} axis	revolute joint variable
Link offset	d_j	the distance from the origin of frame $j-1$ to the x_j axis along the z_{j-1} axis	prismatic joint variable
Link length	a_j	the distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axes is parallel to $\hat{z}_{j-1} \times \hat{z}_j$	constant
Link twist	α_j	the angle from the z_{j-1} axis to the z_j axis about the x_j axis	constant
Joint type	σ_j	$\sigma = 0$ for a revolute joint, $\sigma = 1$ for a prismatic joint	constant

- From these parameters, we can define the transformations per link as:

$$^{j-1}\boldsymbol{A}_j(\theta_j, d_j, a_j, \alpha_j) = \boldsymbol{T}_{Rz}(\theta_j)\boldsymbol{T}_z(d_j)\boldsymbol{T}_x(a_j)\boldsymbol{T}_{Rx}(\alpha_j)$$

$$^{j-1}A_j = \begin{pmatrix} \cos\theta_j & -\sin\theta_j\cos\alpha_j & \sin\theta_j\sin\alpha_j & a_j\cos\theta_j \\ \sin\theta_j & \cos\theta_j\cos\alpha_j & -\cos\theta_j\sin\alpha_j & \alpha_j\sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A popular visual explanation: https://www.youtube.com/watch?v=rA9tm0gTln8

Common terminology

- The set of joint coordinates **q** is called the *joint space*
  - For articulated robots, usually called *joint angles*
- Also referred to as the *pose of the arm*
- Not the same as the *pose of the end-effector*!

Joint angle offset

- The DH notation does not relate directly to a real robot
- Specifically, the zero pose is not usually the same pose as the pose for joint angles of 0 in the controller
- In practice, we define an offset joint vector $\boldsymbol{q}_0$
- Whenever we perform any kinematic function, the offset vector is first added to the joint vector, i.e. $\boldsymbol{q} + \boldsymbol{q}_0$

Forward kinematics

- The goal is to find the end-effector pose, as a function of the joint angles
  - How do we do this?
- Transformation from base to end-effector ${}^{B}T_{E}$
  - How do we find this?
- By joining the transformations for each link ${}^{j-1}A_{j}$ and multiplying

$$ {}^{0}\boldsymbol{T}_{E} = {}^{0}\boldsymbol{A}_{1}\,{}^{1}\boldsymbol{A}_{2}\cdots{}^{N-1}\boldsymbol{A}_{N} $$

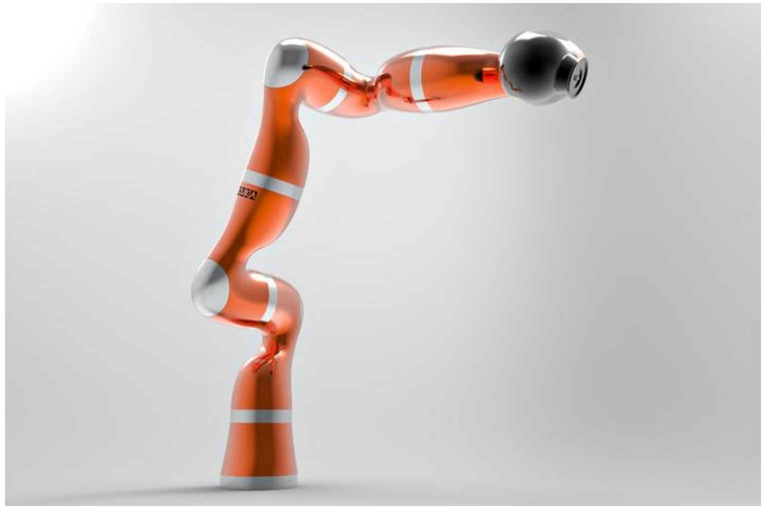
Forward kinematics

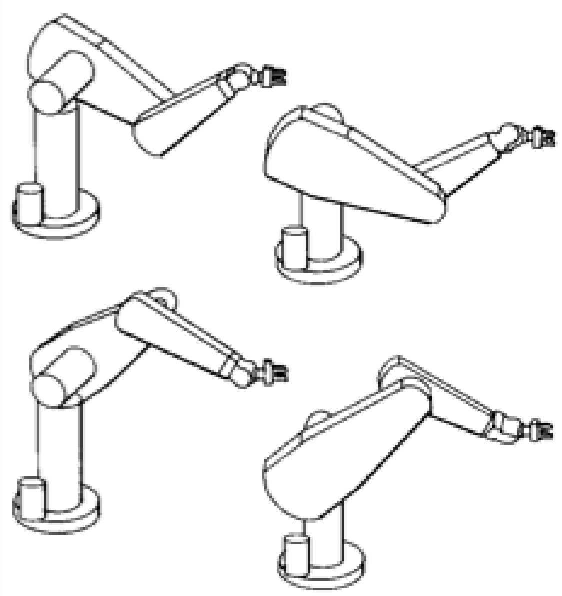
- The forward kinematics solution exists and is unique for *any* serial-link robot
- A serial-link robot is a robot where the links are connected in series

- In the toolbox, this is defined as a SerialLink object
- First, define a vector L with the Links
- Then construct the SerialLink to have the complete robot:
    - **my_robot = SerialLink(L, ’name’, ’My Cool Robot’)**

Inverse kinematics

- The goal is to find the joint angles that locate the end-effector at some desired pose
- A problem of real practical interest, since we usually know where e.g. objects are in Cartesian coordinates
- Solution is not unique, and in some cases no *closed-form* solution exists





Closed-form solution

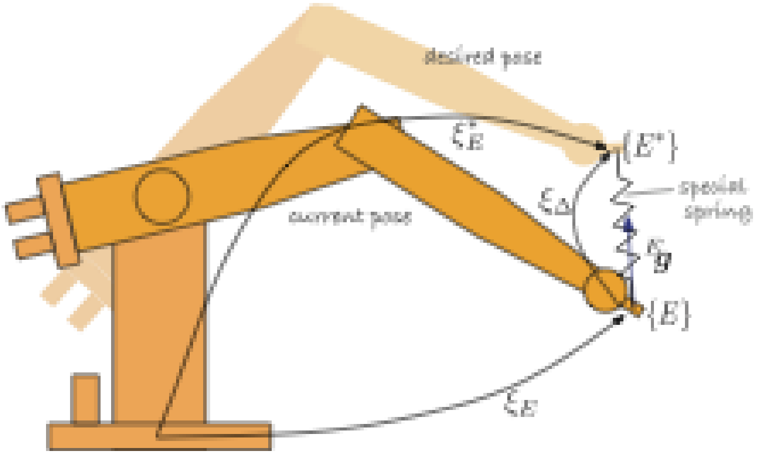
- Requires that
  - The robot has 6 axes
  - The 3 axes in the wrist intersect at a single point
- Thus, motion of the wrist joints only change the orientation of the end-effector

- What if we don’t have this type of robot?
  - We don’t for the UR robots

Numerical solution

- Can deal with any number of joints, and any type of robot
- Considerably slower than the closed-form solution

- Basic principle is to model the pose change as a *special spring*
- Spring *forces and torques* (wrench) are proportional to pose change
- Method:
  - Calculate wrench for current pose difference
  - Calculate pose for current estimate of inverse kinematics
  - Resolve wrench to joint torques
  - Calculate joint velocities due to torques
  - Calculate discrete-time update of joint angles
  - Repeat until wrench is small



Trajectories of robot arms

- The same applies as we discussed in the last lecture
  - It’s all about smooth motion!

- Two different strategies:
  - Straight lines in joint space - *joint-space motion*
  - Straight lines in Cartesian space - *Cartesian motion*

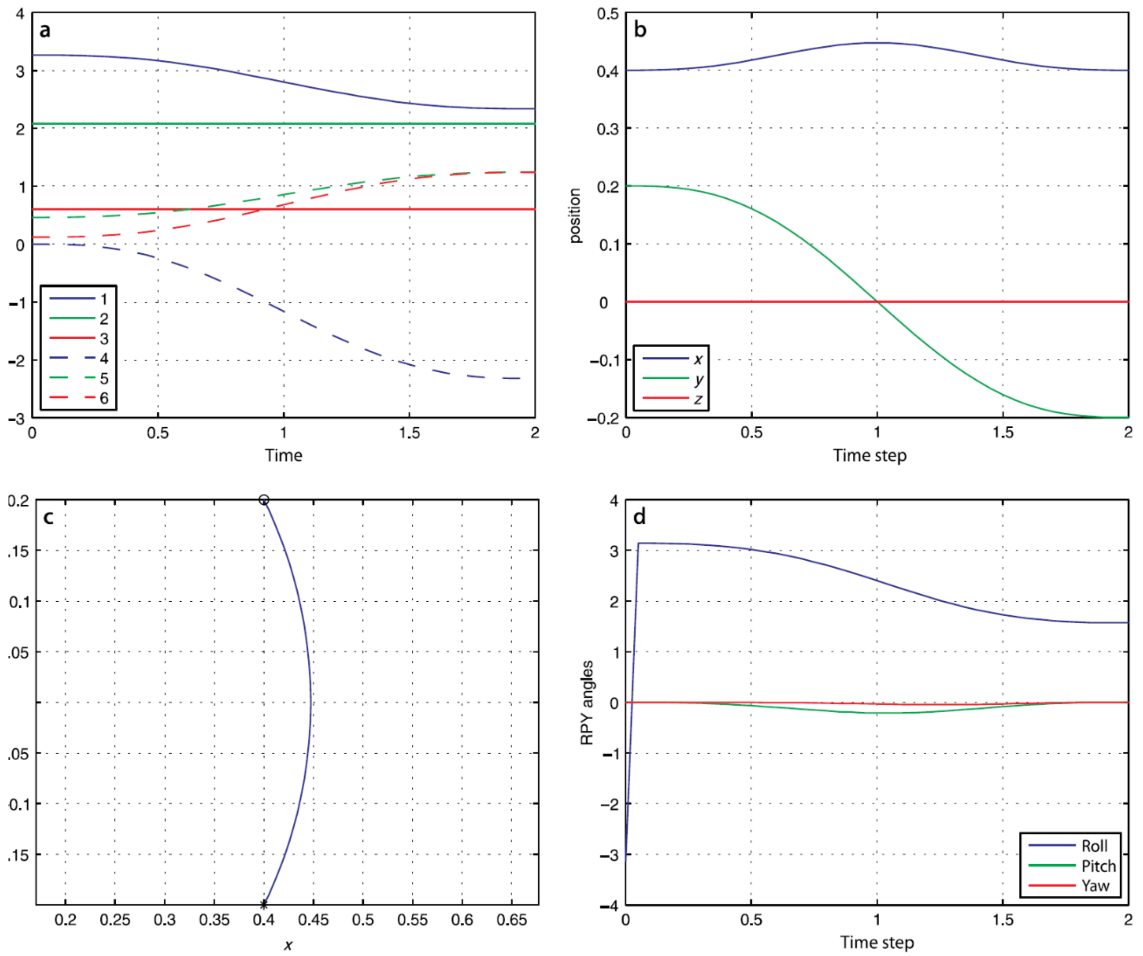
Examining the trajectory

Joint positions

Cartesian positions

Cartesian in xy-plane only

RPY angles



Cartesian trajectories

- In joint space, we cannot guarantee a specific motion of the end-effector
- Cartesian trajectories are useful for
  - Welding, painting, grinding, drawing….
  - Avoiding obstacles

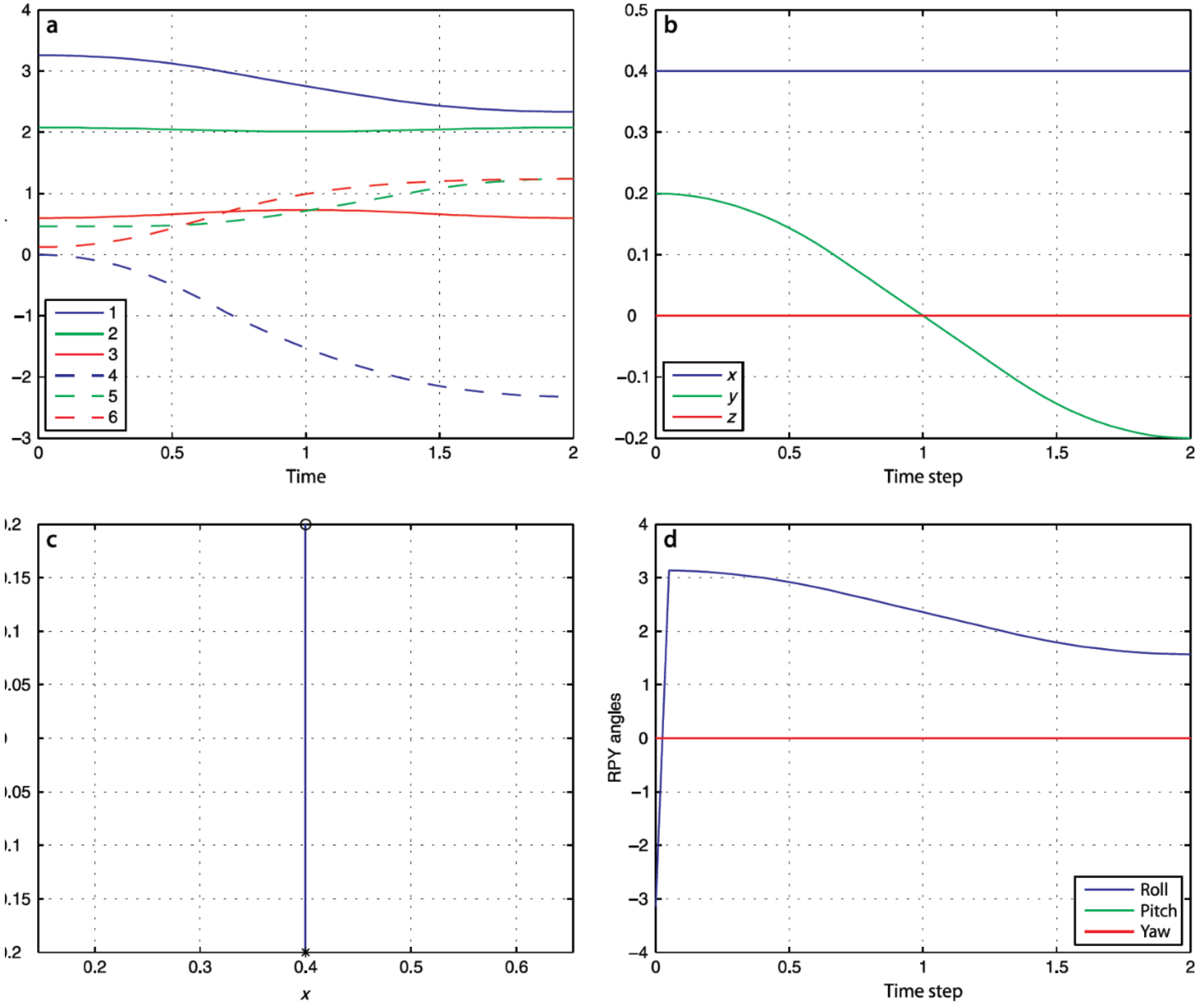
Examining the trajectory

Joint positions

Cartesian positions

Cartesian in xy-plane only

RPY angles



Singularities

- We will now have a look at Cartesian motion through a singularity

- Again, we can generate the Cartesian trajectory, and examine the corresponding joint trajectory for different cases
  - Closed-form solution
  - Numeric solution
  - Pure joint-space trajectory

Joint trajectories

Closed-form solution

Numeric solution

Joint-space trajectory

Manipulability