Lecture 11: More about Jacobians & Trajectory Generation

Iñigo Iturrate

Assistant Professor

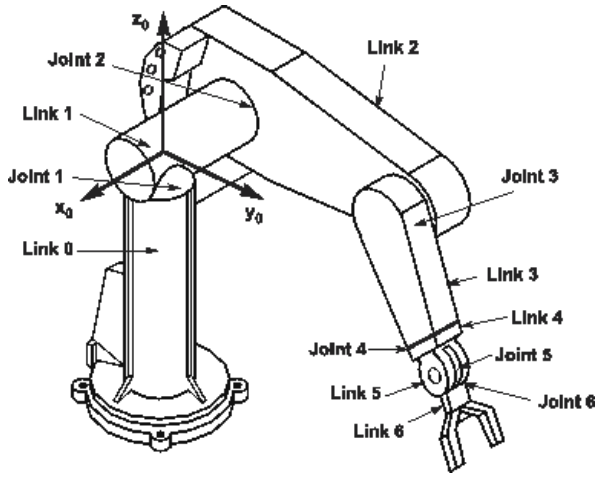
SDU Robotics,

The Maersk McKinney Moller Institute,

University of Southern Denmark

Ø27-604-3

inju@mmmi.sdu.dk



SDU ROBOTICS

Recap (Exercise, 10 min.)

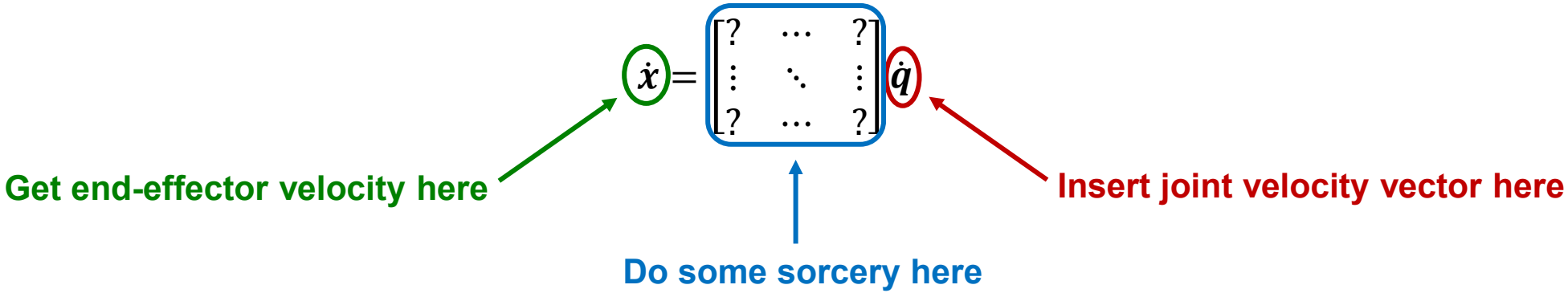
Discuss with your neighbors (5 min.):

In the context of robotics:
1. Why do we use the Jacobian?
2. How is the matrix structured?
3. What can we say about its individual elements in relation to how the robot moves in joint/Cartesian space?

Then share with me what you have discussed (5 min.).

What did we derive last time?

We wanted to find:



(This sorcery is called the Jacobian Matrix)

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

The Two Jacobians in Robotics

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**: $\dot{x} = v = \begin{bmatrix} \dot{p} \\ \dot{\phi} \end{bmatrix}$

We can split the Jacobian in two: $J = \begin{bmatrix} J_P \\ J_O \end{bmatrix}$

Jacobian for Position *(This stays the same)*

Jacobian for Orientation

J_O based on angular velocity

J_O based on derivative of Euler angles

Geometric Jacobian

We will only study this

Analytical Jacobian

SDU ROBOTICS

Building a Geometric Jacobian

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

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_{P1} & \cdots & \boldsymbol{J}_{Pn} \\ \boldsymbol{J}_{O1} & & \boldsymbol{J}_{On} \end{bmatrix} \qquad \boldsymbol{J}_{Pi} = \begin{bmatrix} J_{Pi,x} \\ J_{Pi,y} \\ J_{Pi,z} \end{bmatrix} \qquad \boldsymbol{J}_{Oi} = \begin{bmatrix} J_{Oi,\omega_x} \\ J_{Oi,\omega_y} \\ J_{Oi,\omega_z} \end{bmatrix}$$

Then, for each joint i:

- **Revolute**: $\boldsymbol{J}_{Pi} = {}^0\hat{Z}_i \times ({}^0P_e - {}^0P_i)$, $\boldsymbol{J}_{Oi} = {}^0\hat{Z}_i$
- **Prismatic:** $\boldsymbol{J}_{Pi} = {}^0\hat{Z}_i$, $\boldsymbol{J}_{Oi} = 0$

$${}^0\hat{Z}_i = {}^0_iR \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

${}^0P_e$ is the position of the end-effector

${}^0P_i$ is the position of the origin of frame {i}

What are we doing today?

1. CAD Modelling in Autodesk Inventor – *Guest Lecture by Aljaz Kramberger*
2. CAD Assemblies in Autodesk Inventor – *Guest Lecture by Aljaz Kramberger*
3. Introduction to Robotics & Recap of Linear Algebra and Mathematical Notation
4. Translations & Rotation Matrices
5. Other Representations for Orientation
6. Transformation Matrices
7. DH Parameters & Forward Kinematics
8. Inverse Kinematics
9. Kinematic Simulation
10. Velocity Kinematics & the Jacobian Matrix
11. **More about the Jacobian & Trajectory Generation** ***(Today)***
12. Manipulability, More on the Robotic Systems Toolbox

Topics for Today

Part I: The Jacobian Matrix & Singularities

- Forward & Inverse Velocity Kinematics
- Using the Jacobian for Numerical IK (conceptually)
- Singularities as seen:
  - From the Jacobian
  - From Analytical (closed-form) solutions to IK

Part II: Trajectory Generation

- Point-to-point (PTP) movements
  - In Joint space and Cartesian space (position only!)
- Trapezoidal velocity profiles
- Quadratic, Cubic and Higher-order Polynomials
- Chaining multiple PTP movements

Part I: Using the Jacobian Matrix & Singularities

Forward and Inverse Velocity Kinematics

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

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

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

The forward/inverse veocity 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)

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

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{P1} & \cdots & \boldsymbol{J}_{P7} \\ \boldsymbol{J}_{O1} & & \boldsymbol{J}_{O7} \end{bmatrix} \begin{bmatrix} q_1 \\ \vdots \\ q_7 \end{bmatrix}$$

$(6 \times 1) \qquad (6 \times 7) \qquad (7 \times 1)$

Due to a **rank-defficient square** matrix

Rank & Determinant of the Jacobian

Remember: The **rank of a matrix** is the **number of linearly independent columns**.

Full rank: The rank of the matrix is the maximum possible for a matrix of that size.

Rank-defficient/singular: The matrix is not full rank.
- The **robot cannot move in one or more directions**.
- At a **singular configuration**, the **determinant** of the Jacobian will be **zero** (and close to zero when close to a singularity).

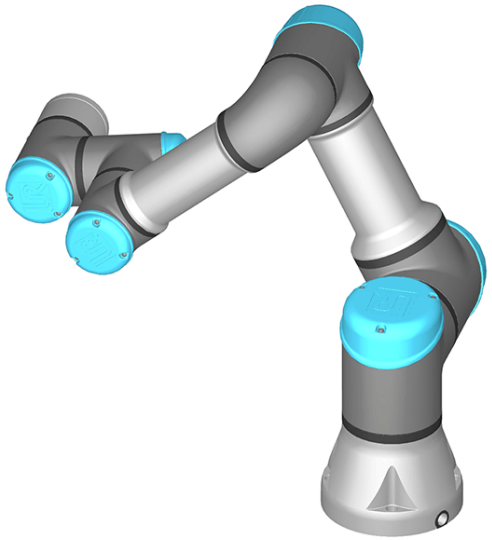
Discuss:
Think about this last statement in connection with:
- The definition of the inverse of a matrix: $\boldsymbol{J^{-1}} = \frac{1}{\det(\boldsymbol{J})}(\mathrm{adj}(\boldsymbol{J}))$
- The inverse velocity kinematics equation: $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\dot{\boldsymbol{x}}$

What can we conclude about the joint velocities of a robot when moving close to a singularity?

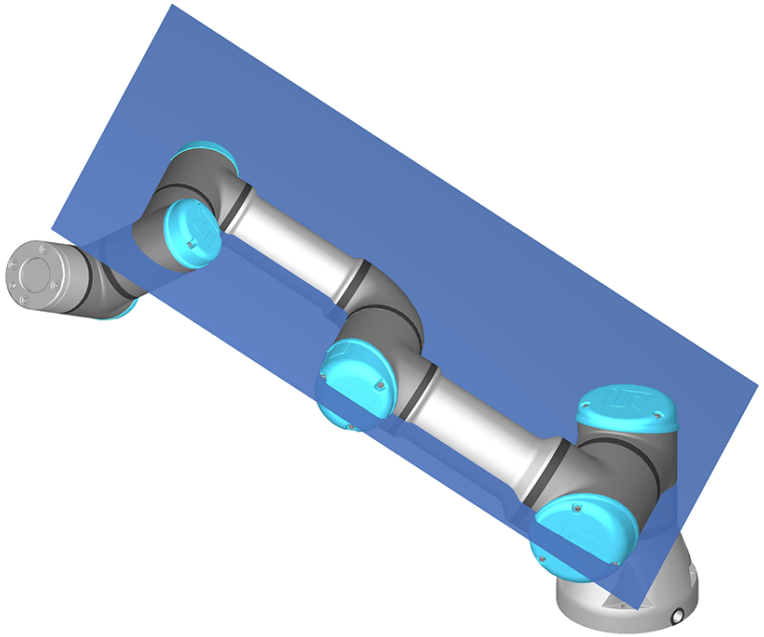
Singularities – Example: UR Robot

Singularities of UR: Relating them to IK Solutions

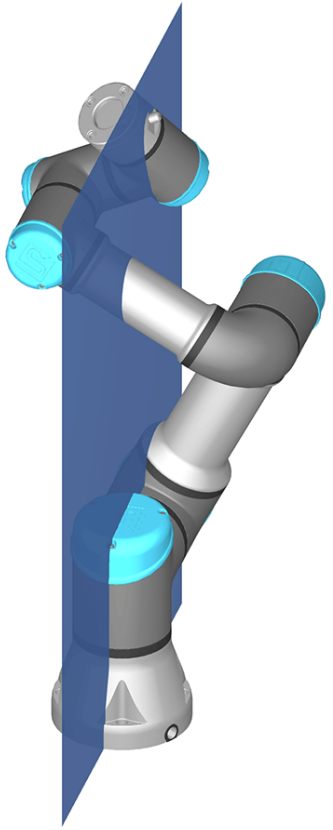
Wrist singularity
- Axes of joints 4 and 6 parallel
- $\theta_5 = 0°$, $\theta_5 = \pm 180°$ or $\theta_5 = \pm 360°$



Elbow singularity
- Axes of joints 2, 3 and 4 coplanar
- $\theta_3 = 0°$



Shoulder singularity
- Intersection of axes of joints 5 and 6 coplanar with axes of joints 1 and 2.



Practical Applications of the Jacobian

The Jacobian is **used everywhere** in robotics:

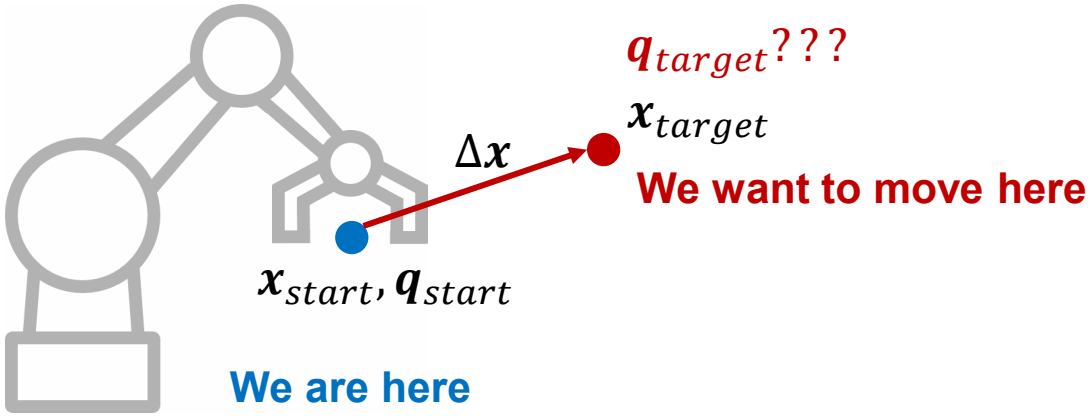
- **Numerical Inverse Kinematics**
  → **We will look at this (conceptually) now**

- **Manipulability Metrics** (checking how ”free” the robot is to move/apply a force in each direction)
  → **We will study this next lecture**

- **Robot Control**
  - Position Control
  - Velocity Control
  - Force Control

  } **You will study this in other courses**

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: $J\big(q(t + dt)\big), x(t + dt)$ *(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 if the Jacobian is NOT square?

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

- Jacobian **Transpose**: $\dot{\boldsymbol{q}} = \alpha \boldsymbol{J}^T \dot{\boldsymbol{x}}$ (for some small $\alpha > 0$)
  - This can behave ”close enough” to the inverse assuming a small enough α.

- Jacobian (Moore-Penrose) **Pseudoinverse**: $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger} \dot{\boldsymbol{x}}$ (where † is the pseudoinverse)
  - This is a generalization of the concept of inverse for non-square matrices.

…which, in MATLAB, can be calculated using `pinv()`:
https://se.mathworks.com/help/matlab/ref/pinv.html?s_tid=doc_ta

Part II: Trajectory Generation

(for Linear Paths)

Offline vs. Online Robot Trajectory Control

Internally, **a robot has a control computer** that is updating the robot parameters and **sending commands** to the motors at a **fixed updated rate/control cycle**.

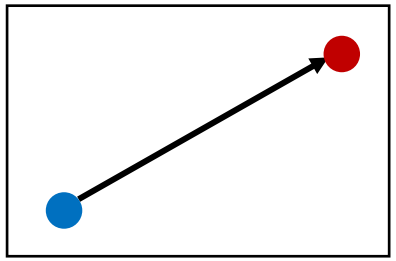
There are typically two ways for **a user** to control a robot:

- **Online control: Commands** are **sent** to the robot **at** its **update rate/control frequency**.
  - Example: UR e-series robots run at 500 Hz, so a new target point would have to be sent to the robot every 2 ms.
  - This means that **a path has to be discretized**/sampled to fit the robot's control cycle.

- **Offline control:** Specify a **start and end point**, a **type of motion**, and **constraints** on time/velocity.
  - Example: Move in a straight line in Cartesian space from $P_1 = \left[0.3, -0.2, 0.4, \frac{\pi}{2}, 0, 0\right]$ to $P_2 = \left[0.5, 0.1, 0.2, \frac{\pi}{2}, 0, 0\right]$
  - The robot internally figures out how to **translate this into an online control problem**.

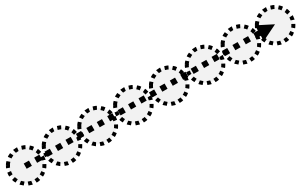
Offline vs. Online Robot Trajectory Control

Desired Path



Online Control

Discretized Path:



Offline Control

Move Type:

”Linear Cartesian Move”

or

”Linear Joint Move”

Parameters:

Start Point

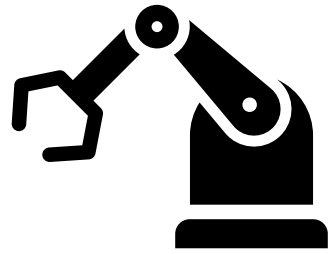
Target Point

Speed

Time

Send commands continuously

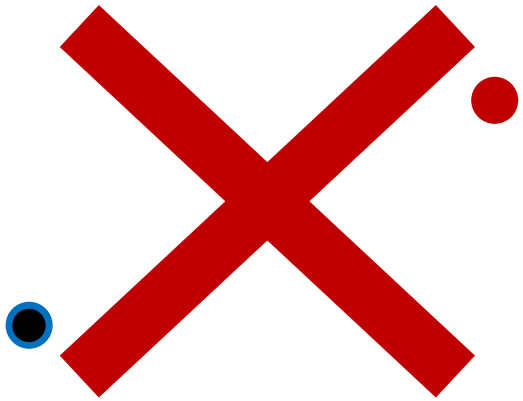
Send command once



Robot Control Computer

(updates at certain control frequency)

Getting from A to B: Multiple Paths



This is a linear path.
This is what we are discussing.

This path is not linear.
We are not considering this right now

<u>Linear paths will always take the shortest distance *in that particular space.*</u>

SDU ROBOTICS

Movement in Joint or Cartesian Space

We can **define trajectories** for a robot in **either Joint or Cartesian space**.

- **Joint space** has the advantage that we **do not have to consider singularities**.

- **Cartesian space** has the advantage that it is **easier to relate to the task** and to how we think.

Linear Movements: Cartesian ≠ Joint Space!

Linear movement in Cartesian space → curved movement in Joint Space!

Let us see it on the robot…

Getting from A to B: Multiple Velocity Profiles

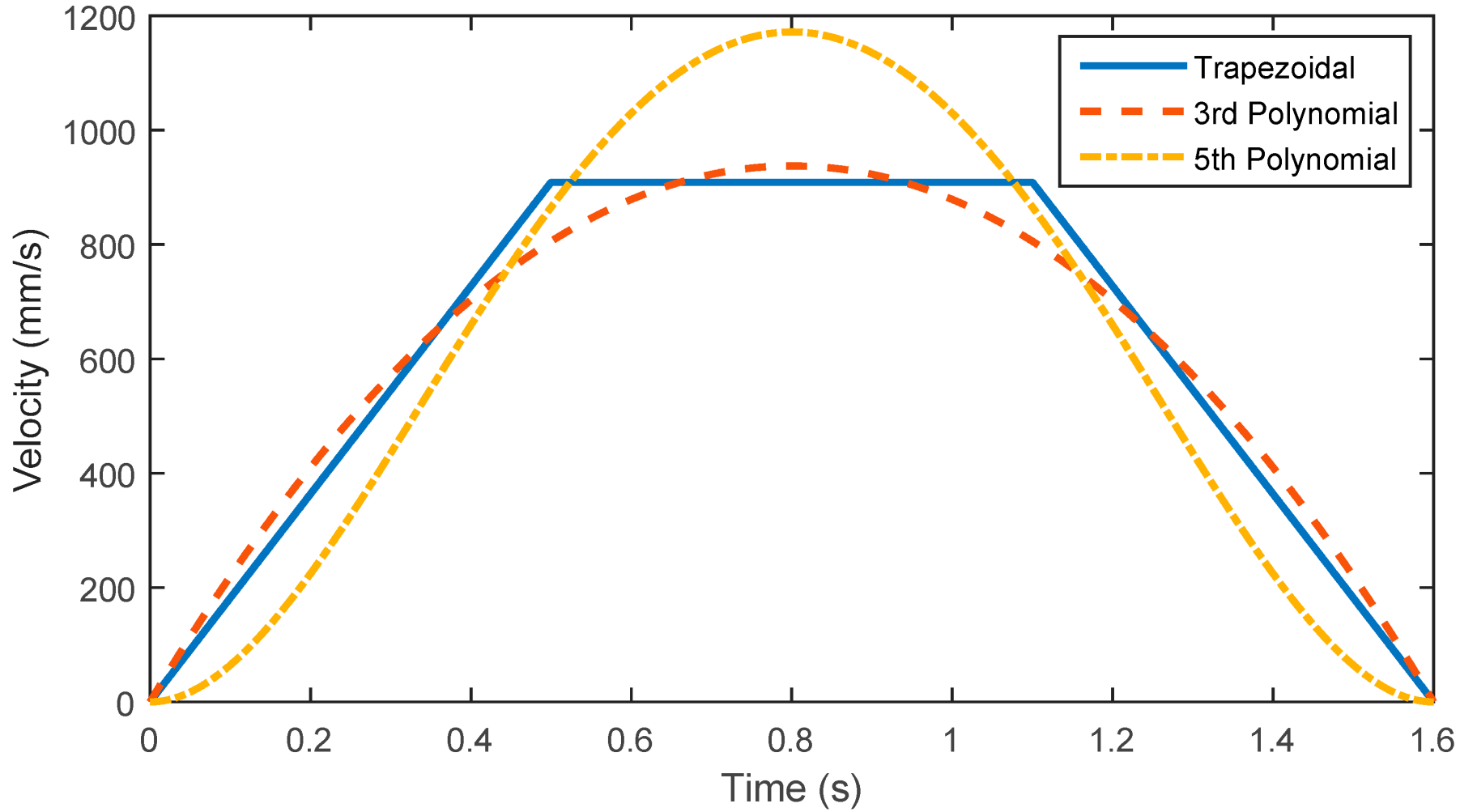
All of these paths reach the target point at the same time:

…but they do so with different velocity profiles.

This is what we will be discussing.

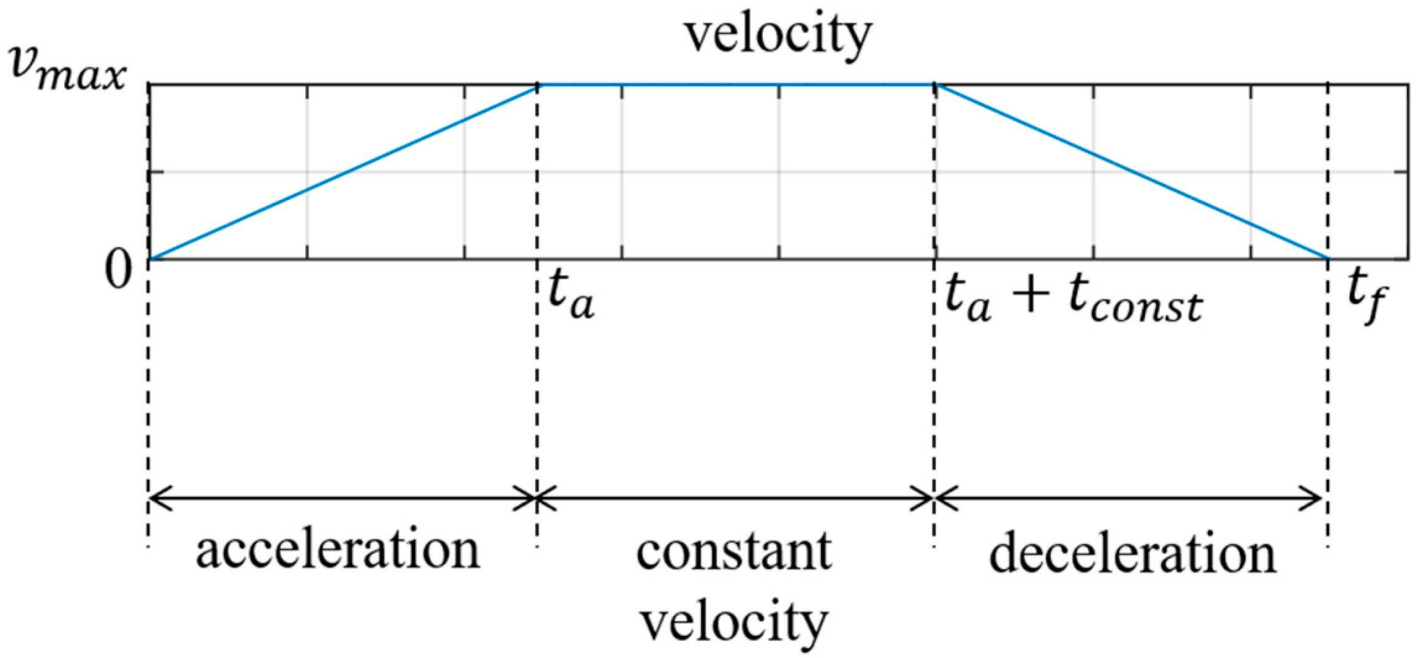
SDU ROBOTICS

Different Velocity Profiles



Source: Yoon et al. “Trapezoidal Motion Profile to Suppress Residual Vibration of Flexible Object Moved by Robot”, *Electronics*, MDPI, 2019, https://www.mdpi.com/2079-9292/8/1/30/htm

Trapezoidal Velocity Profiles



Specify either:

- Max. **velocity** and **acceleration**:

$$t_a = \frac{v_{max}}{a_{max}}$$

- Max. **velocity** and **time**:

$$a_{max} = \frac{v_{max}}{t_a}$$

- Max. **acceleration** and **time**:

$$v_{max} = a_{max} t_a$$

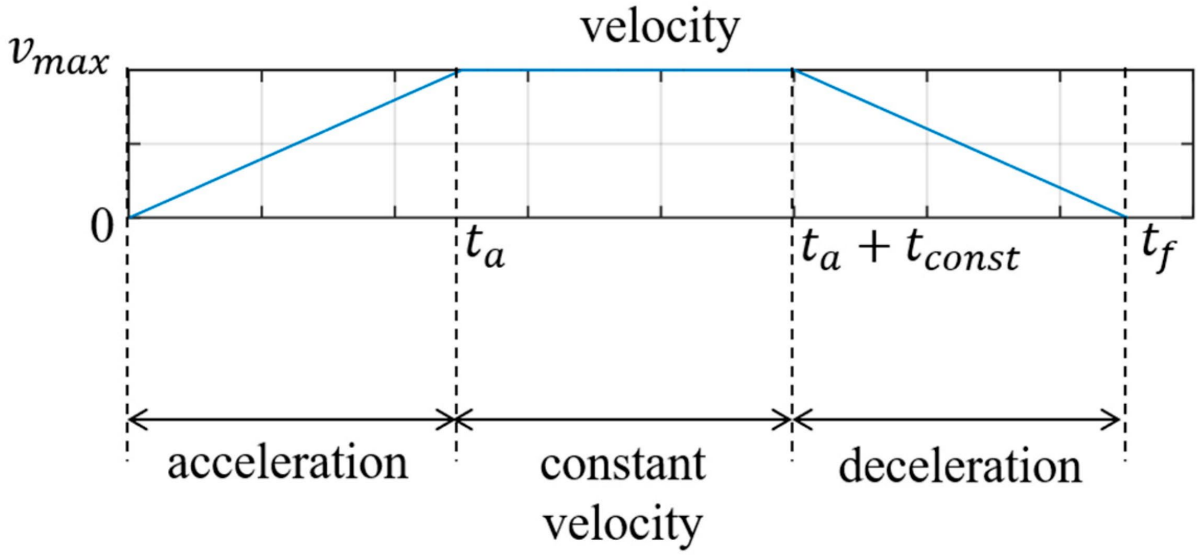
Then calculate:

$$t_f = \frac{\Delta x}{v_{max}} + t_a$$

$$t_{const} = t_f - 2t_a$$

Source: Yoon et al. “Trapezoidal Motion Profile to Suppress Residual Vibration of Flexible Object Moved by Robot”, *Electronics*, MDPI, 2019, https://www.mdpi.com/2079-9292/8/1/30/htm

Exercise (10 min.)



Given: $p_{start}, v_{max}, a_{max}, t_a, t_{const}$

1. Write an equation to calculate the velocity at any given time, t for a trapezoidal velocity profile.
2. Do the same for position.

(Hint: They should be piecewise equations)

$$v(t) = \begin{cases} a_{max}t, & t < t_a \\ v_{max}, & t_a \leq t < t_a + t_{const} \\ v_{max} - a_{max}\left(t - (t_a + t_{const})\right), & t \geq t_a + t_{const} \end{cases}$$

$$p(t) = \begin{cases} \frac{1}{2}a_{max}t^2, & t < t_a \\ \frac{1}{2}a_{max}{t_a}^2 + v_{max}(t - t_a), & t_a \leq t < t_a + t_{const} \\ \frac{1}{2}a_{max}{t_a}^2 + v_{max}(t - t_a) - a_{max}\left(t - (t_a + t_{const})\right)^2, & t \geq t_a + t_{const} \end{cases}$$

Simplification of $p(t)$ for $t \geq t_a + t_{const}$

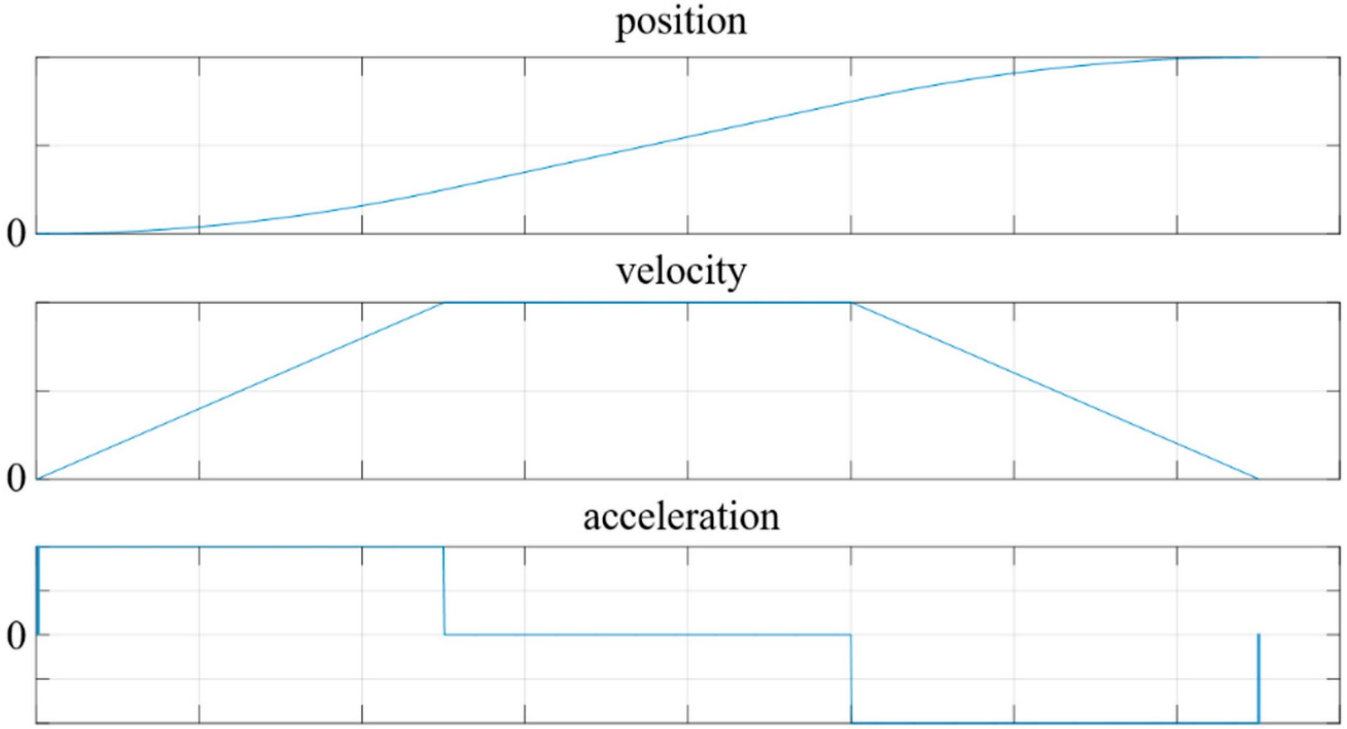
$$\frac{1}{2} a_{max} {t_a}^2 + v_{max}\left((t_a + t_{const}) - t_a\right) + \left(v_{max} - a_{max}\left(t - (t_a + t_{const})\right)\right)\left(t - (t_a + t_{const})\right)$$

$$\frac{1}{2} a_{max} {t_a}^2 + v_{max} t_{const} + v_{max}\left(t - (t_a + t_{const})\right) - a_{max}\left(t - (t_a + t_{const})\right)^2$$

$$\frac{1}{2} a_{max} {t_a}^2 + \cancel{v_{max} t_{const}} + v_{max}(t - t_a) - \cancel{v_{max} t_{const}} - a_{max}\left(t - (t_a + t_{const})\right)^2$$

$$\frac{1}{2} a_{max} {t_a}^2 + v_{max}(t - t_a) - a_{max}\left(t - (t_a + t_{const})\right)^2$$

Trapezoidal Velocity Profiles: Derivatives



There is a problem…

Jerk is the **derivative of the acceleration**,
i.e., how fast the acceleration changes over time.

Why do we care about jerk?

- High jerk causes **wear in gears, motors and mechanical components**.
- In a car accident, **jerk** is what **causes damage to your neck from whiplash**.

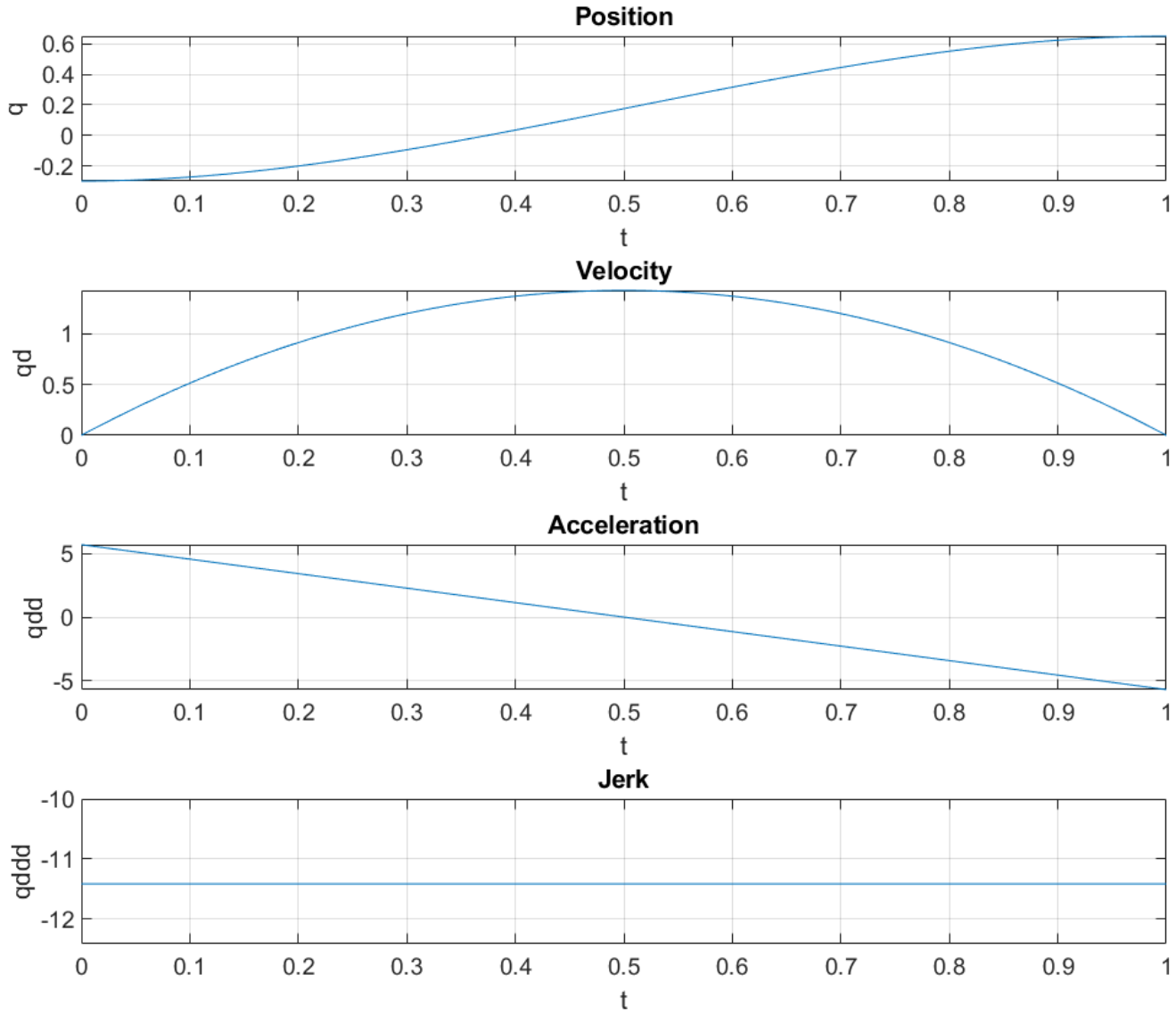
**For a trapezoidal velocity profile,
jerk is (theoretically) infinite!**

Cubic Polynomial

We can also specify a **trajectory** as a **cubic polynomial**:

$$q(t) = at^3 + bt^2 + ct + d$$
$$\dot{q}(t) = 3at^2 + 2bt + c$$
$$\ddot{q}(t) = 6at + 2b$$
$$\dddot{q}(t) = 6a$$

This will result in **smoother motion** than a trapezoidal velocity profile.



Boundary Constraints

To **calculate the parameters** of the cubic polynomial, we **set up constraints**:

- On the **starting point**:

$$q(0) = q_{start} = d$$
$$\dot{q}(0) = \dot{q}_{start} = c$$

- On the **end point**:

$$q(T) = q_{end} = aT^3 + bT^2 + cT + d$$
$$\dot{q}(T) = \dot{q}_{end} = 3aT^2 + 2bT + c$$

4 equations, 4 unknowns

Then we just **solve** for the parameters as a **system of equations**.

Higher Order Polynomials

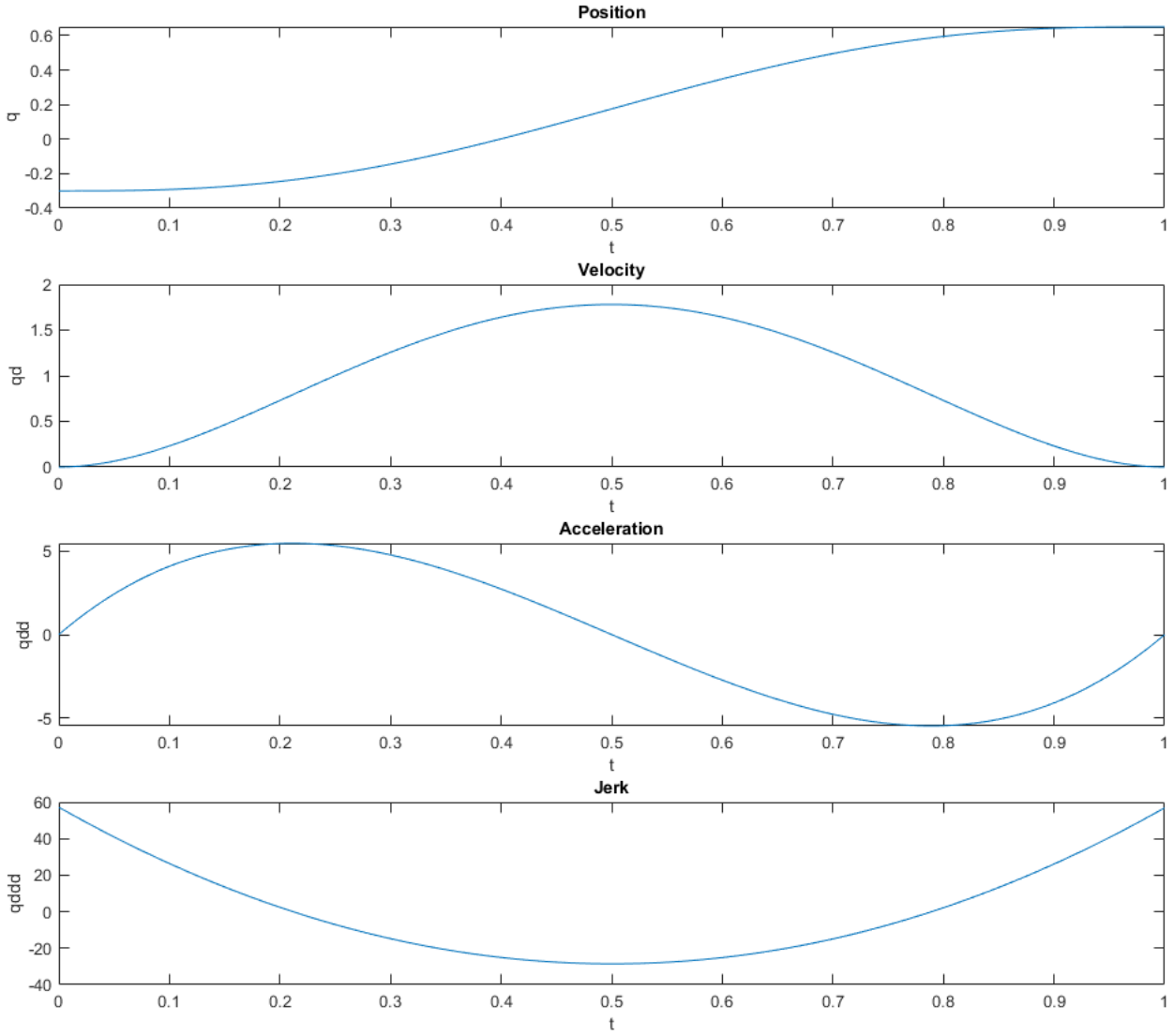
We can do the same for **higher-order polynomials.**

For example, a **fifth-order polynomial** allows us to specify **boundary constraints on the acceleration**:

$$q(t) = at^5 + bt^4 + ct^3 + dt^2 + et + f$$
$$\dot{q}(t) = 5at^4 + 4bt^3 + 3ct^2 + 2dt + e$$
$$\ddot{q}(t) = 20at^3 + 12bt^2 + 6ct + 2d$$

$$q(0) = q_{start} = f$$
$$\dot{q}(0) = \dot{q}_{start} = e$$
$$\ddot{q}(0) = \ddot{q}_{start} = 2d$$

$$q(T) = q_{end} = aT^5 + bT^4 + cT^3 + dT^2 + eT + f$$
$$\dot{q}(T) = \dot{q}_{end} = 5aT^4 + 4bT^3 + 3cT^2 + 2dT + e$$
$$\ddot{q}(T) = \ddot{q}_{end} = 20aT^3 + 12bT^2 + 6cT + 2d$$



SDU ROBOTICS

Trajectories for Multiple Joints/Dimensions

The previous examples consider only **one** joint/Cartesian space **dimension <u>independently</u>** at a time.

Generalizing to multiple dimensions is simply a matter of **making all parameters vectors**:

$$\boldsymbol{q}(t) = \boldsymbol{a}t^3 + \boldsymbol{b}t^2 + \boldsymbol{c}t + \boldsymbol{d}$$

$$\begin{bmatrix} q_1(t) \\ \vdots \\ q_n(t) \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} t^3 + \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} t^2 + \begin{bmatrix} c \\ \vdots \\ c_n \end{bmatrix} t + \begin{bmatrix} d \\ \vdots \\ d_n \end{bmatrix}$$

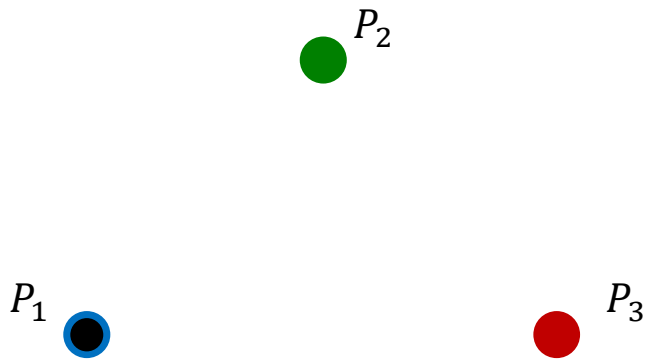
Question: The above is true for a complete joint-space position.
Will it work for a full Cartesian-space pose (position + orientation)?
Why or why not? If not, which part is it true for?
Discuss (5 minutes)

Linear Trajectories through Multiple Points

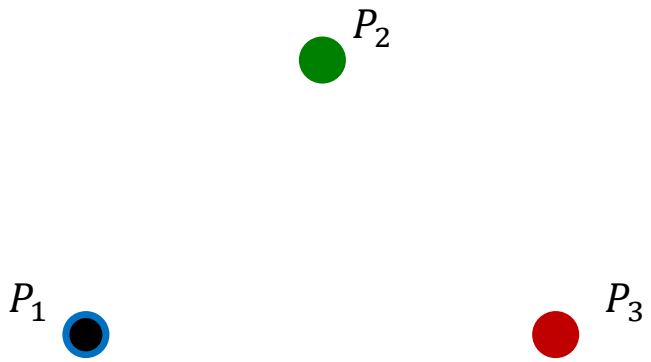
Question: with the tools so far, how do you obtain the following behavior?

(Discuss with your neighbors, 5 min., then share with the class)

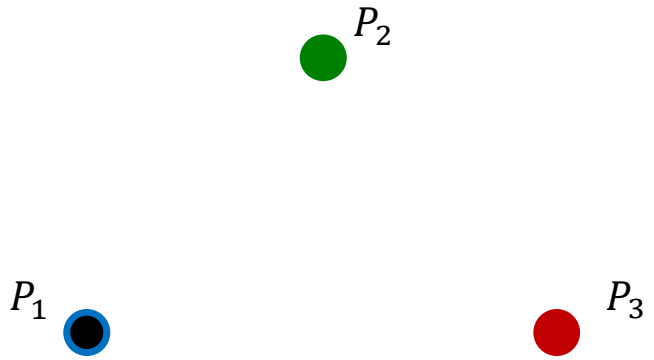
Stopping at middle point



Continuous motion through middle point

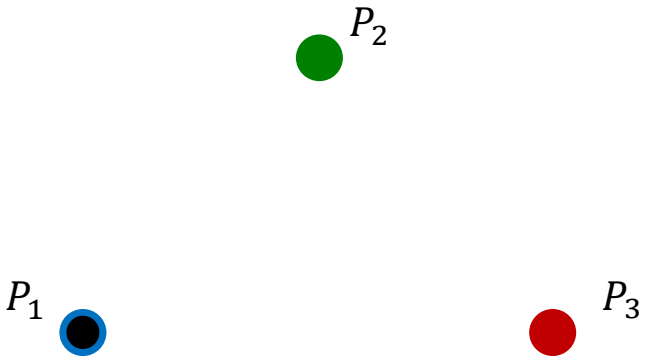


Linear Trajectories through Multiple Points



Two trapezoidal velocity profiles in a row:

- One from P_1 to P_2
- Another from P_2 to P_3



Two cubic polynomials:

$$q_1(t) = a_1t^3 + b_1t^2 + c_1t + d_1$$
$$\dot{q}_1(t) = 3a_1t^2 + 2b_1t + c_1$$
$$\ddot{q}_1(t) = 6a_1t + 2b_1$$

$$q_2(t) = a_2t^3 + b_2t^2 + c_2t + d_2$$
$$\dot{q}_2(t) = 3a_2t^2 + 2b_2t + c_2$$
$$\ddot{q}_2(t) = 6a_2t + 2b_2$$

with shared middle constraint:

$$q_1(0) = P_1 \qquad q_1(T_{end,1}) = q_2(0) = P_2 \qquad q_2(T_{end,2}) = P_3$$
$$\dot{q}_1(0) = 0 \qquad \dot{q}_1(T_{end,1}) = \dot{q}_2(0) = v \qquad \dot{q}_2(T_{end,2}) = 0$$

What about Cartesian-space Orientation?

The **methods** introduced so far **assume that each dimension is independent**.

When describing **orientation**, **all components are interdependent**.

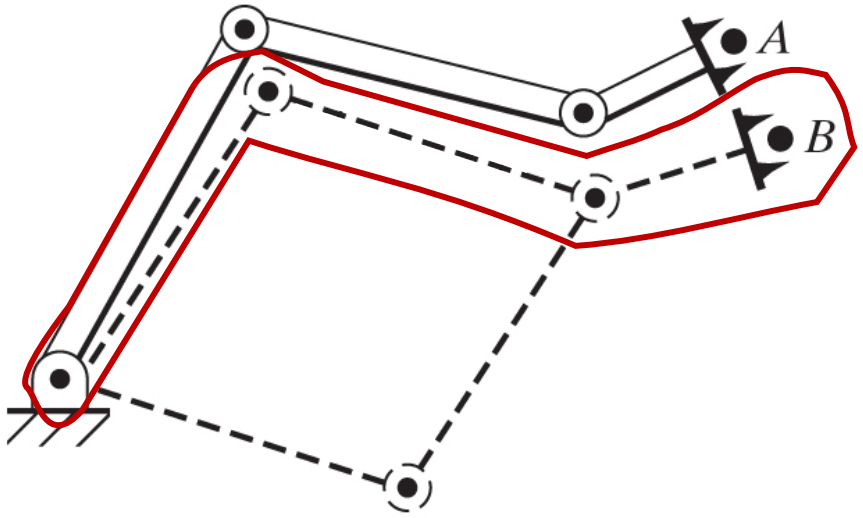
Orientations need to be treated separately using, e.g. quaternions.

This is much more complicated and beyond this course.

Moving in Cartesian Space: IK Solutions

When **moving in Cartesian space**, we also need to consider the existence of **multiple IK solutions**.

Within the same trajectory, we will usually want to pick the **closest solution to the current configuration**.



With redundant robots (and our own arm), we can **change solutions for the elbow while moving**.

Recap: What have we discussed today

- The **inverse velocity kinematics** equation has many **important applications in robotics**.
- The **inverse of the Jacobian cannot be calculated** if the robot is **in a singular configuration**.
  - In this case, the **robot cannot move in certain directions**.
  - **Joint velocities** close to singularities will be **very large**.
- **Linear movements** in **Cartesian** and **Joint space** will always follow the **shortest distance** (**in that space!**)
- We can design different linear movements by changing the **velocity profile**:
  - **Trapezoidal**
  - **Polynomial** (3rd or 5th order)
- When **moving in Cartesian space**, we must always consider the **choice of IK solution**.

Take home message(s):

1. Moving close to **singularities** (where the **Jacobian cannot be inverted**) results in **high joint velocities**, which has important practical consequences.

2. For linear motions, **different velocity profiles** (e.g. **trapezoidal** or **polynomial**) will result in different features and degrees of smoothness. The **choice of profile** will influence our ability to **set boundary conditions**.

SDU ROBOTICS

Thank you for today.

Iñigo Iturrate

Ø27-604-3

inju@mmmi.sdu.dk