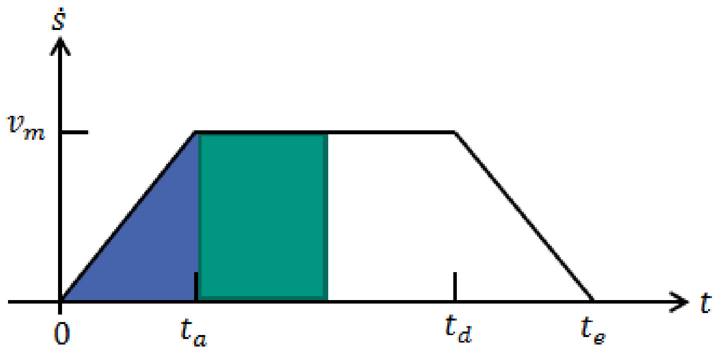
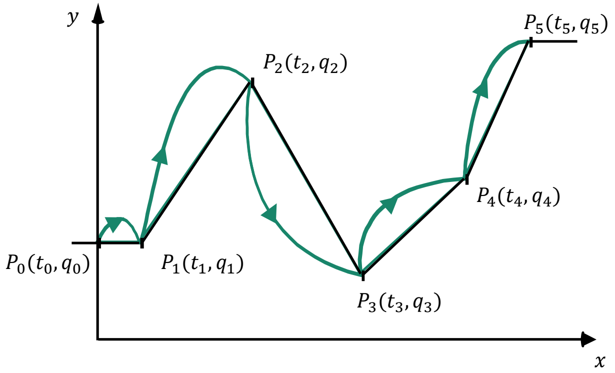
Robotics I: Introduction to Robotics

Chapter 6 – Trajectory Generation

Tamim Asfour

https://www.humanoids.kit.edu





Outline

- **Fundamentals of trajectory generation**
- Programming of key points
- Interpolation types
- Approximated trajectory generation

Fundamentals of Trajectory Generation: Trajectory

The movements of a robot are regarded as

- **State changes**
  - Over time
  - Relative to a fixed coordinate system (Workspace, Configuration space)
- with **restrictions** due to
  - Constraints
  - Quality criteria
  - Secondary and boundary conditions

Fundamentals of Trajectory Generation: Problem

- Given
  - S_{Start}:
    State at the **start time**
  - $S_{Destination}$:
    State at the **destination time**
- Desired
  - S_i:
    Intermediate states (support points), so that the trajectory is continuous.

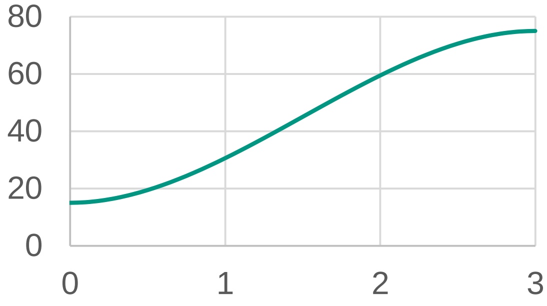
Trajectory Generation: Example for a Single Joint

- Start conditions:

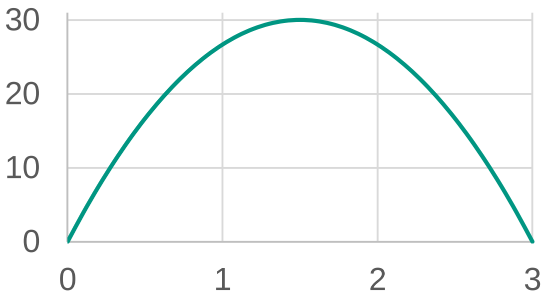
$$q(t_0) = 15°$$
$$\dot{q}(t_0) = 0\ \frac{°}{sec}$$
$$\ddot{q}(t_0) = 40\ \frac{°}{sec^2}$$

- End conditions:

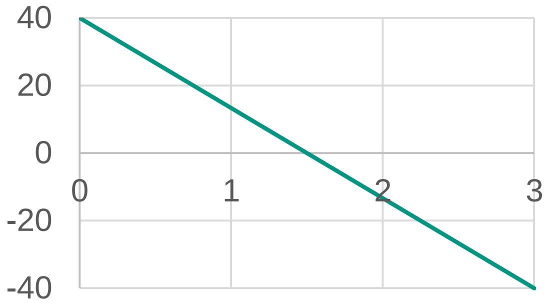
$$q(t_e) = 75°$$
$$\dot{q}(t_e) = 0\ \frac{°}{sec}$$
$$\ddot{q}(t_e) = -40\ \frac{°}{sec^2}$$



Position $q(t)$



Velocity $\dot{q}(t)$



Acceleration $\ddot{q}(t)$

Trajectory Generation: Example for a Single Joint

- Start conditions:

$$q(t_0) = 15^\circ$$
$$\dot{q}(t_0) = 0\ \frac{^\circ}{sec}$$
$$\ddot{q}(t_0) = 40\ \frac{^\circ}{sec^2}$$

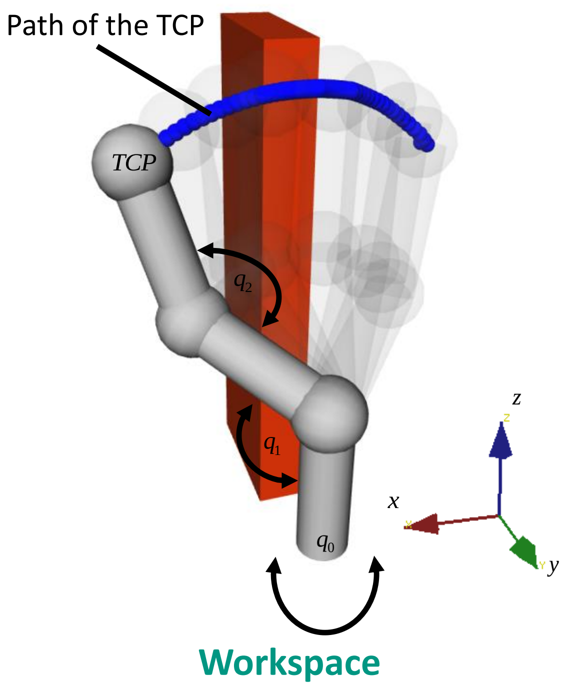
- End conditions:

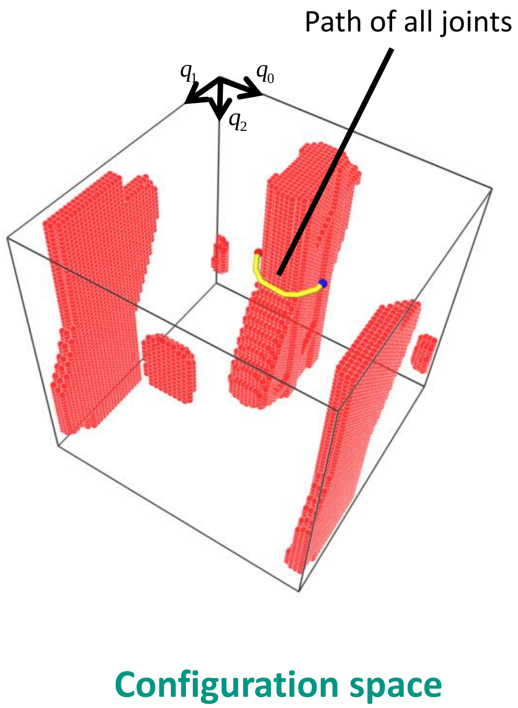
$$q(t_e) = 75^\circ$$
$$\dot{q}(t_e) = 0\ \frac{^\circ}{sec}$$
$$\ddot{q}(t_e) = -40\ \frac{^\circ}{sec^2}$$

We can determine a third-degree polynomial that fulfills the conditions:

$$q(t) = -\frac{40}{9}t^3 + 20\,t^2 + 15 \qquad \dot{q}(t) = -\frac{40}{3}t^2 + 40\,t \qquad \ddot{q}(t) = -\frac{80}{3}t + 40$$

Trajectory Generation: Representation of the States (1)





Trajectory Generation: Representation of the States (2)

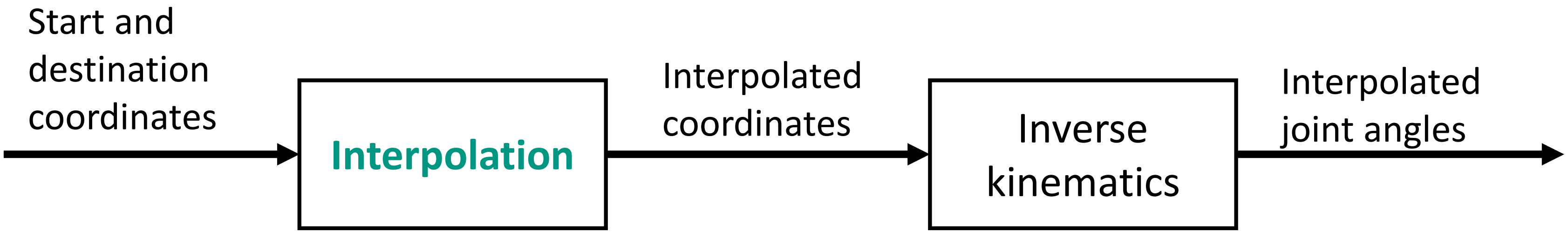
- States can be represented in
  - Configuration space: $\mathbb{R}^n$
  - Workspace: $\mathbb{R}^3$, $SE(3)$

- Trajectory generation in the configuration space is closer to the control of the robot components (joints, sensors)

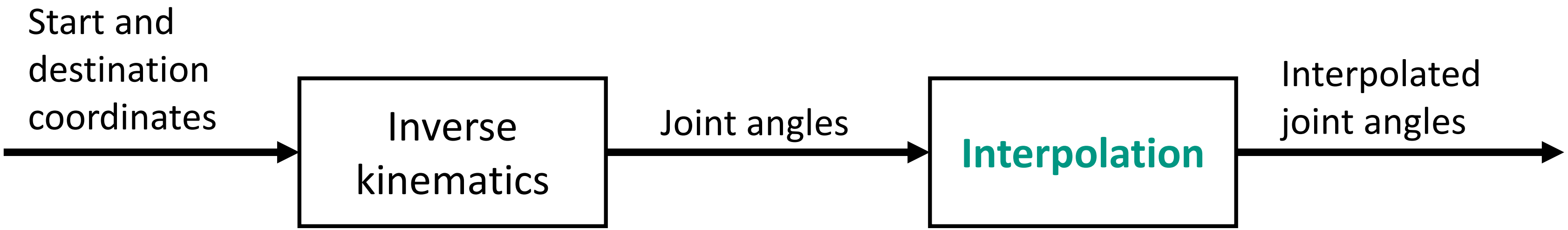
- Trajectory generation in the workspace is closer to the task to be solved
  - For control in the workspace, the **inverse kinematics** must be solved

Trajectory Generation: Interpolation

- Interpolation of **world coordinates** 内插法



- Interpolation of **joint angles**



Trajectory Generation in the Configuration Space

- Trajectory generation as a function of the **joint angle states**
  - The course of the path, which is specified point by point in joint space, does not have to be defined in the workspace.

- Traversing trajectories that are specified point by point:
  - 异步
    Asynchronous: Control of the axes independently of each other
    - Applications: Spot welding, handling tasks 点焊任务,物料搬运
  - 同步
    Synchronous: Axis-interpolated control
    - Movement of all axes starts and ends at the same time
    - Leading axis
    - Applications: Path welding, spray painting, assembly tasks

Trajectory Generation in the Workspace

- The trajectory is specified as a function of the **robot states**
  - Example: Description vector of the end effector
  - Position, Velocity, Acceleration
- **Continuous Path** (CP):
  End effector follows a **well-defined path** in terms of its position and orientation
- **Path types**
  - Linear paths 线性路径
  - Polynomial paths 多项式路径
  - Splines 样条

Trajectory Generation: Pros and Cons of the Representations

Workspace	Configuration space
+ Path easier to formulate + Interpolation is easier	+ Control of the joints is easier + Trajectory is unambiguous and respects the limits of the joint angles
– Inverse kinematics must be solved for each point of the trajectory – The planned trajectory cannot always be executed 规划的轨迹不一定总能被执行。由于关节的物理限制，某些工作空间的轨迹可能无法映射到可行的关节角度。	– Interpolation for multiple joints – Formulation of the trajectory is more complicated

Outline

- Fundamentals of trajectory generation
- **Programming of key points**
- Interpolation types
- Approximated trajectory generation

Direct Programming: Teach-In (1)

- **Manual steering** to prominent **points** along the path
  - Teach Box
  - Teach Panel
  - Spacemouse
  - Teach Ball

- **Functionality** of a **Teach Box:**
  - Individual movement of the joints
  - Movement of the end effector in 6 degrees of freedom
  - Saving and deleting waypoints
  - Setting velocities
  - Entering commands to operate the gripper
  - Starting / stopping entire programs

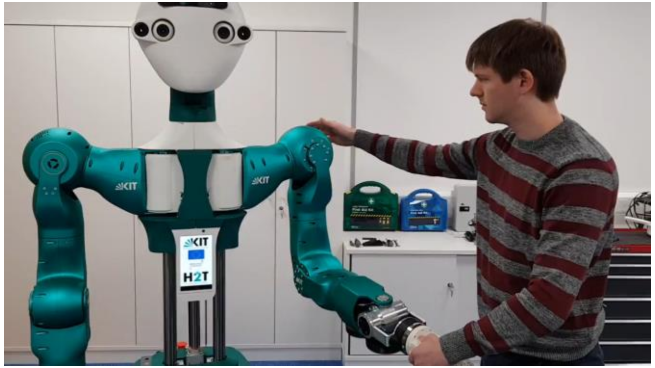
Direct Programming: Teach-In (2)

- Procedure:
  - **Move** the robot to relevant **key points** on the path
  - **Record** the **joint positions**
  - **Add parameters** such as velocities and accelerations to the stored values

- Applications:
  - Manufacturing industry
    - Spot welding
    - Riveting
  - Handling tasks
    - Taking parcels from a conveyor belt

Direct Programming: Playback (1)

- Robot in **zero-force control** mode
  - Robot can be moved **by the operator**
  - **Movement** along the desired path
  - **Recording** of the **joint values** (2 options):
    - Automatically (predefined sampling frequency)
    - Manually (by pressing a button)



- Applications:
  - Motion sequences that are difficult to describe mathematically
  - Integration of experience in craftsmanship
  - Typical application areas:
    - Spray painting
    - Gluing

Direct Programming: Playback (2)

Direct Programming: Playback (3)

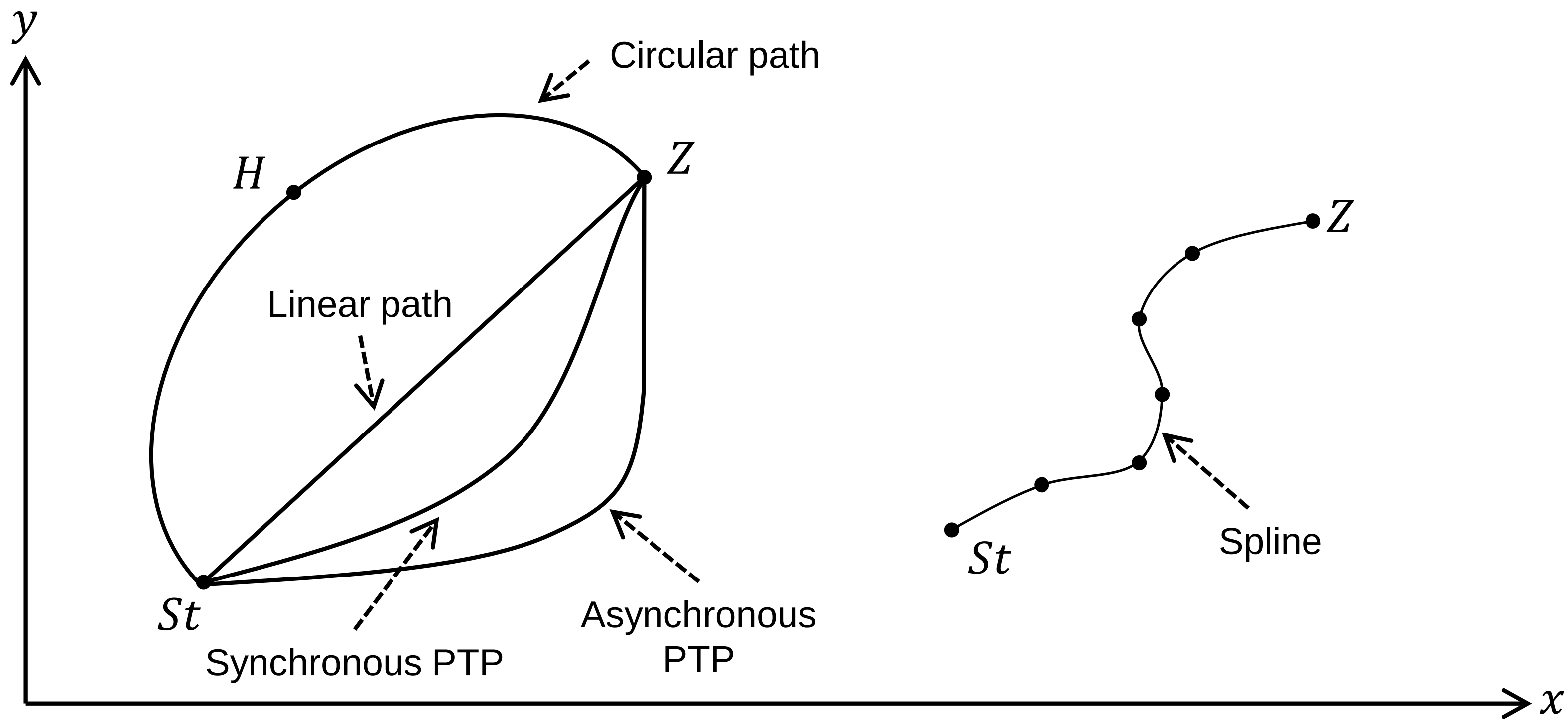
- Advantages
  - **Fast** for complex paths
  - **Intuitive**

- Disadvantages
  - **Heavy robots** are often difficult to move
  - Little **space** in narrow production cells poses a safety risk for the operator
  - **Limited correction options**
  - **Optimization** and control using interpolation methods is **difficult** (suboptimal paths)

Outline

- Fundamentals of trajectory generation
- Programming of key points
- **Interpolation types**
  - **Point-to-point (PTP)**
  - **Linear and circular interpolation**
  - **Spline interpolation**
- Approximated trajectory generation

Interpolation Types: Overview



Point-to-Point Control (PTP) (1)

- Robot performs a **point-to-point movement**
  - PTP: Point-to-Point

- Advantages:
  - Calculating the joint angle trajectory is **simple**
  - **No problems** with **singularities**

- Sequence of **joint angle vectors**

$$\boldsymbol{q}(t_j) = \left(q_1(t_j), q_2(t_j), \dots, q_n(t_j)\right)^T$$

with $q_i(t_j)$: Angle of joint i at time t_j with $j = 0, \dots, k$

Point-to-Point Control (PTP) (2)

Boundary conditions

- **Start and destination states** are known

$$\boldsymbol{q}(t_0) = \boldsymbol{q}_{Start}$$
$$\boldsymbol{q}(t_e) = \boldsymbol{q}_{Destination}$$

- Example: Velocities at the beginning and the end are zero

$$\dot{\boldsymbol{q}}(t_0) = 0$$
$$\dot{\boldsymbol{q}}(t_e) = 0$$

- The **joint positions**, the **joint velocities** and the **joint accelerations** are **limited** (e.g. fast acceleration, slow deceleration)

$$\boldsymbol{q}_{min} < \boldsymbol{q}(t_j) < \boldsymbol{q}_{max}$$
$$|\dot{\boldsymbol{q}}(t_j)| < \dot{\boldsymbol{q}}_{max}$$
$$|\ddot{\boldsymbol{q}}(t_j)| < \ddot{\boldsymbol{q}}_{max}$$

Point-to-Point Control (PTP) (3)

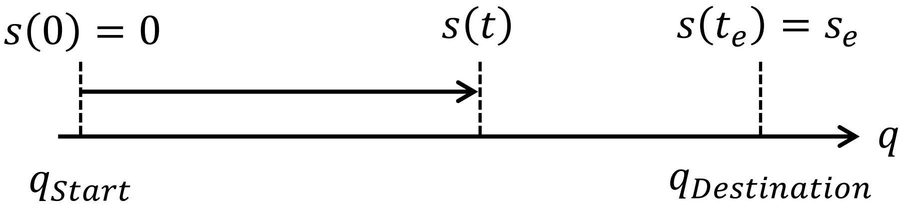
Control sequence

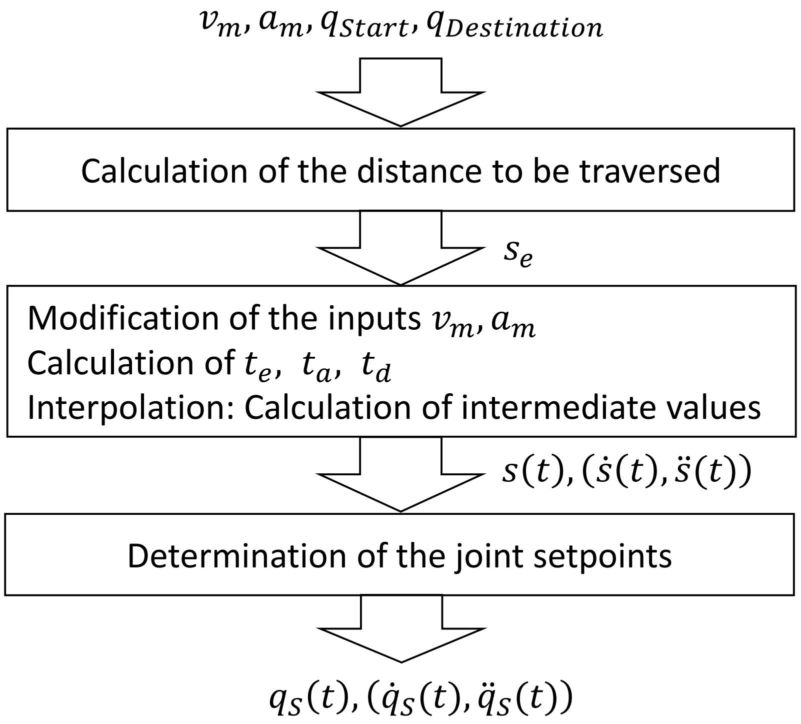
- Traversing time t_e
- Acceleration time t_a
- Start of braking time t_d
- Maximum velocity v_m
- Maximum acceleration a_m

$$s(0) = \dot{s}(0) = v(0) = 0$$

$$s(t_e) = s_e = |q_{Destination} - q_{Start}|$$

$$\dot{s}(t_e) = v(t_e) = 0$$

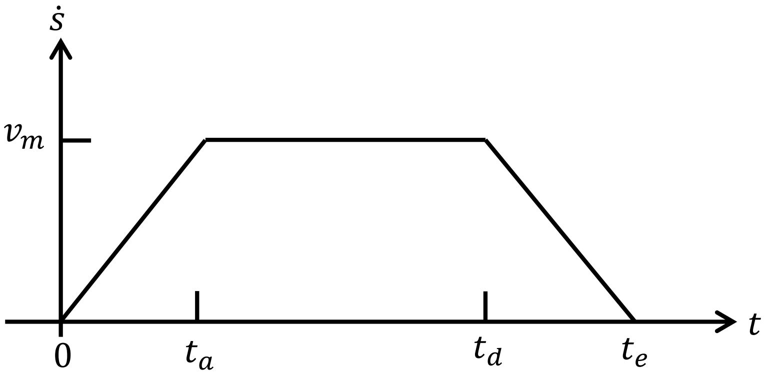




带斜坡轮廓

Interpolation for PTP with a Ramp Profile (1)

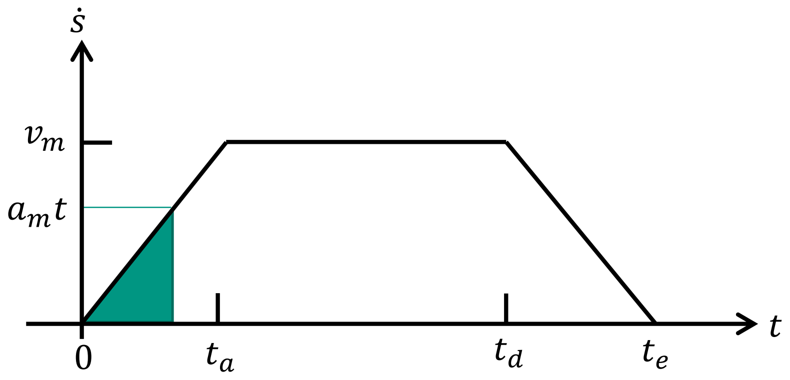
- Advantage:
  Simple way to compute the path parameters $s(t)$
- Disadvantage:
  The acceleration is **discontinuous** (unlimited jerk), which can excite natural vibrations in mechanical parts.



Interpolation for PTP with a Ramp Profile (2)

Phase I: **Acceleration**

$$0 \leq t \leq t_a$$



$$\ddot{s}(t) = a_m$$

$$\dot{s}(t) = a_m t + \dot{s}(0) \qquad with\ \dot{s}(0) = 0$$

$$= a_m t$$

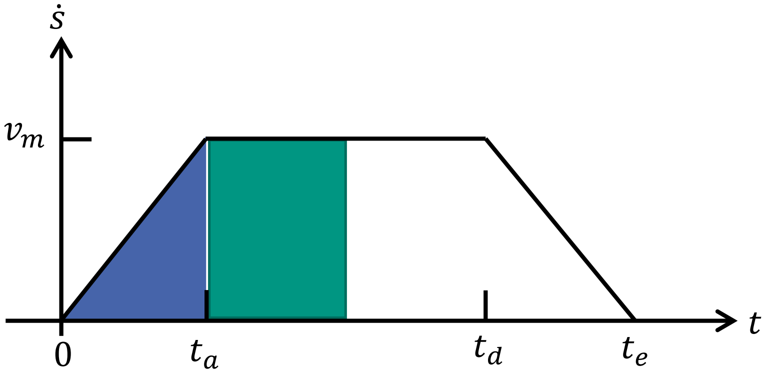
$$s(t) = \boldsymbol{\frac{1}{2} a_m t^2} + s(0) \qquad with\ s(0) = 0$$

$$= \frac{1}{2} a_m t^2$$

Interpolation for PTP with a Ramp Profile (3)

Phase II: Constant velocity

$$t_a \le t \le t_d$$



$$\ddot{s}(t) = 0$$

$$\dot{s}(t) = \dot{s}(t_a) = v_m$$

$$s(t) = \boldsymbol{v_m(t - t_a)} + \boldsymbol{s(t_a)}$$

$$= v_m\left(t - \frac{v_m}{a_m}\right) + \frac{1}{2}a_m {t_a}^2$$

$$= v_m t - \frac{1}{2}\frac{v_m^2}{a_m}$$

We know from Phase I:

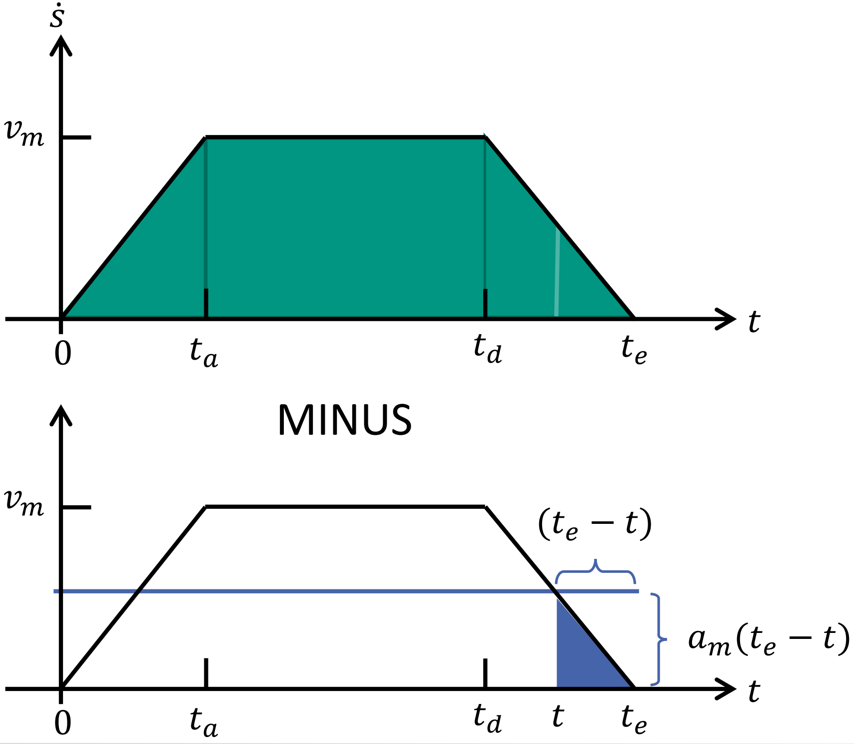
$$\dot{s}(t_a) = a_m t_a = v_m \;\rightarrow\; t_a = \frac{v_m}{a_m}$$

$$\boldsymbol{s(t_a) = \frac{1}{2} a_m {t_a}^2}$$

Interpolation for PTP with a Ramp Profile (4)

Phase III: **Braking process**

$$t_d \leq t \leq t_e \quad with \quad t_d = t_e - t_a$$



$$\ddot{s}(t) = -a_m$$

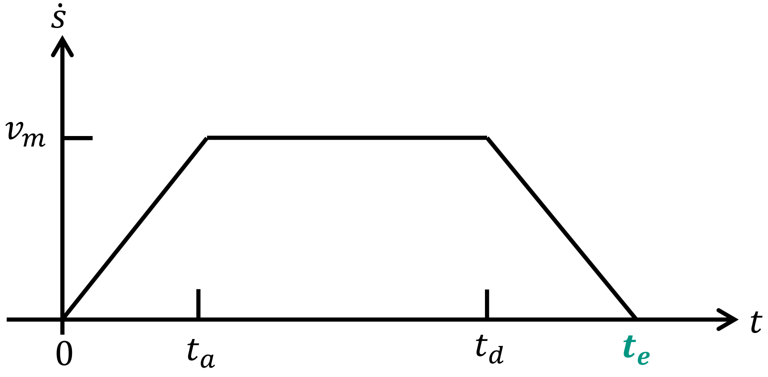
$$\dot{s}(t) = -a_m(t - t_d) + \dot{s}(t_d)$$

$$= -a_m(t - t_d) + v_m$$

$$s(t) = \boldsymbol{v_m(t_e - t_a)} - \frac{\boldsymbol{a_m}}{\boldsymbol{2}}\boldsymbol{(t_e - t)^2}$$

Interpolation for PTP with a Ramp Profile (5)

Calculation of the **traversing time**



We know from Phase III:

$$s(t_e) = s_e = v_m(t_e - t_a)$$

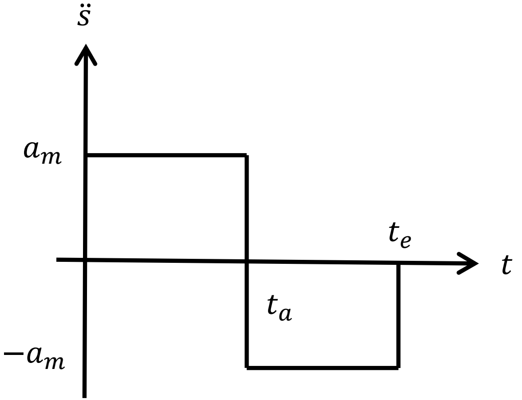
Solve for t_e, $t_a = \frac{v_m}{a_m}$

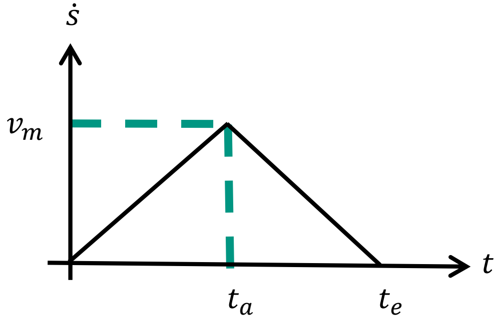
$$t_e = \frac{s_e}{v_m} + t_a = \frac{s_e}{v_m} + \frac{v_m}{a_m}$$

Time-optimal Path

If v_m is too large in relation to the acceleration and path length:
Determination of a time-optimal path according to

$$\boldsymbol{s_e = t_a \cdot v_m} = \frac{{v_m}^2}{a_m} \rightarrow v_m = \sqrt{a_m s_e}$$

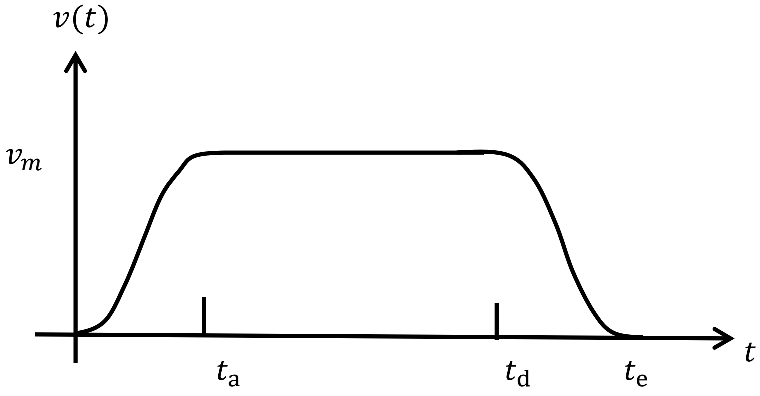




Interpolation for PTP with a Sinoid Profile (1)

- **Smoother movement** by using a sinusoidal time function
- Advantage:
  - Less strain on the robot
- Disadvantage:
  - Longer acceleration and braking phase compared to the ramp profile

- Determination of the curve parameters for the three phases:
  - Acceleration
  - Constant velocity
  - Braking process



Interpolation for PTP with a Sinoid Profile (2)

- Phase of **acceleration**

$$\ddot{s}(t) = a_m sin^2\left(\frac{\pi}{t_a}t\right) \quad 0 \le t \le t_a$$

$$\dot{s}(t) = a_m\left(\frac{1}{2}t - \frac{t_a}{4\pi}\sin\left(\frac{2\pi}{t_a}t\right)\right)$$

$$s(t) = a_m\left(\frac{1}{4}t^2 + \frac{{t_a}^2}{8\pi^2}\left(\cos\left(\frac{2\pi}{t_a}t\right) - 1\right)\right)$$

- From $\dot{s}(t_a) = a_m\frac{1}{2}t_a = v_m$ follows $t_a = \frac{2v_m}{a_m}$

- Phase of **constant velocity**

$$\ddot{s}(t) = 0 \quad t_a \le t \le t_d$$
$$\dot{s}(t) = v_m$$
$$s(t) = v_m(t - \frac{1}{2}t_a)$$

Interpolation for PTP with a Sinoid Profile (3)

- Phase of the **braking process**

$$\dot{s}(t) = v_m - \int_{t-t_d}^{t} a(\tau - t_d)d\tau = v_m - a_m(\frac{1}{2}(t - t_d) - \frac{t_a}{4\pi}\sin\left(\frac{2\pi}{t_a}(t - t_d)\right)\right) \quad t_d \leq t \leq t_e$$

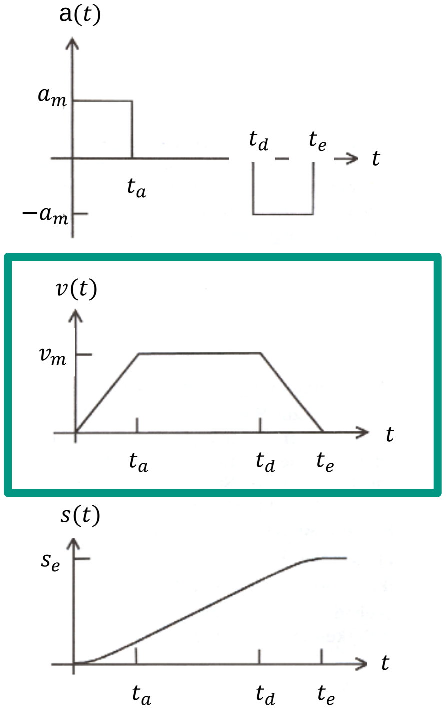
$$s(t) = s(t_d) + \int_{t-t_d}^{t} \dot{s}\,(\tau - t_d)d\tau = \frac{a_m}{2}\left(t_e(t + t_a) - \frac{t^2 + t_e^2 + 2\,t_a^2}{2} + \frac{t_a^2}{4\pi}\left(1 - \cos\left(\frac{2\pi}{t_a}(t - t_d)\right)\right)\right)$$

- Computation of the **traversing time**

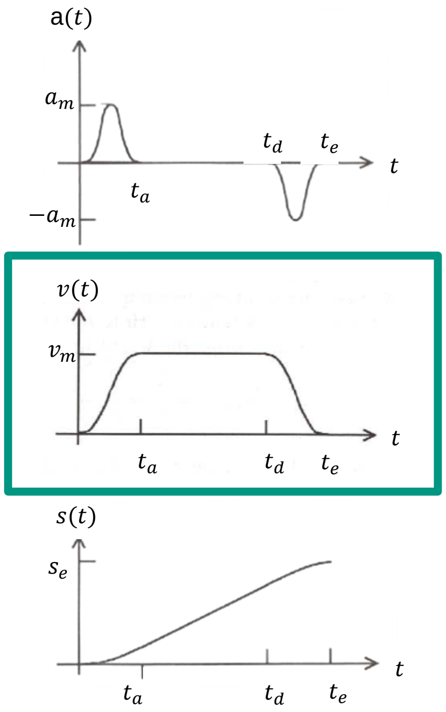
$$t_e = \frac{s_e}{v_m} + t_a = \frac{s_e}{v_m} + \frac{2v_m}{a_m}$$

Interpolation Types: Ramp vs. Sinoid Profile

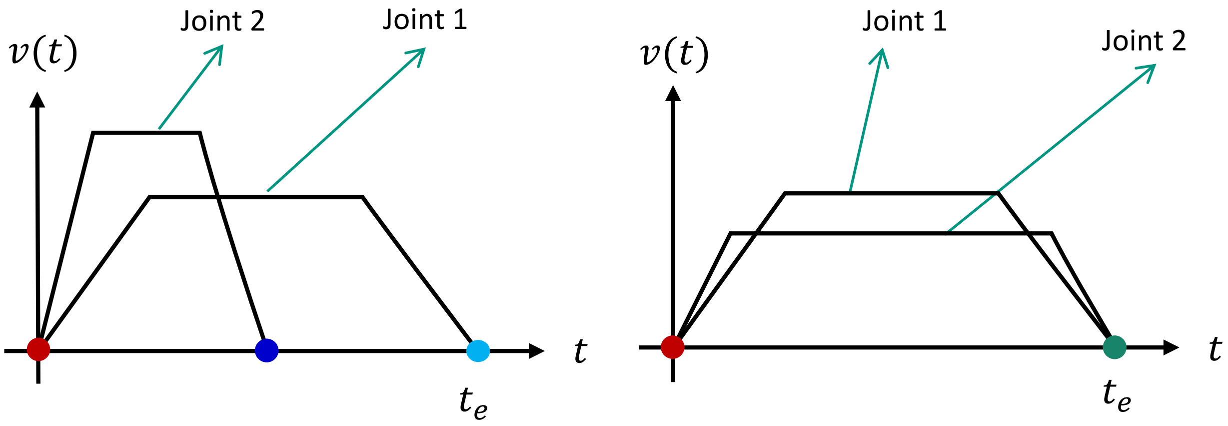
Ramp profile



Sinoid profile



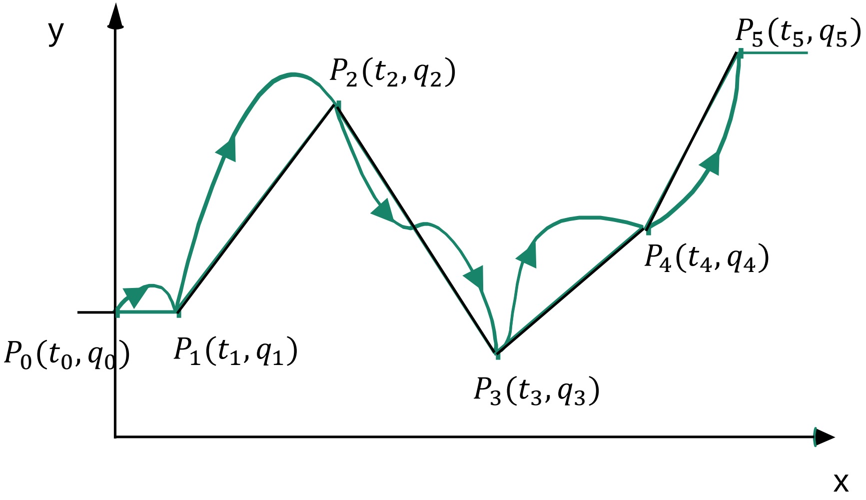
Asynchronous and Synchronous PTP Paths



a) Asynchronous

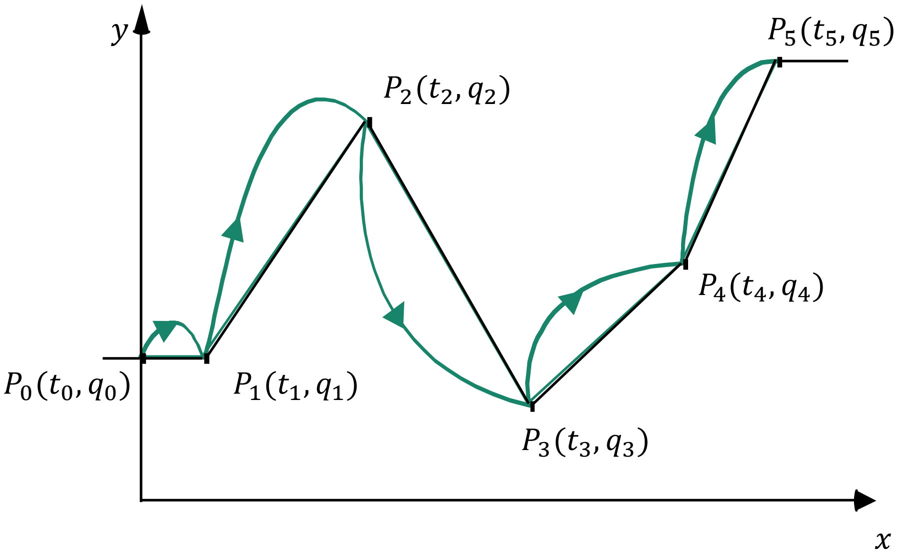
b) Synchronous

Asynchronous PTP Paths



- Each joint is **immediately** actuated with the **maximum acceleration.**
- Each joint movement **ends independently** of the others.

Synchronous PTP Paths



- All joints **start and end** their **movements at the same time** (synchronous).

Synchronous PTP Paths: Procedure (1)

- Determine the **PTP parameters** for **each joint i** (analogous to asynchronous PTP)
  - $s_{e,i}$
  - $v_{m,i}$
  - $a_{m,i}$
  - $t_{e,i}$ (traversing time)
- Determine the **maximum traversing time**
  - $t_e = t_{e,max} = max(t_{e,i})$
  - Axis with the maximum traversing time is the leading axis
- Set the **maximum traversing time** as the traversing time **for all joints.**
  - $t_{e,i} = t_e$

Synchronous PTP Paths: Procedure (2)

- Determine the **new maximum velocity** for **all joints**
  - **Conversion of the traversing time** und calculation of the new maximum velocity
    - **Ramp profile:**

$$t_e = \frac{s_{e,i}}{v_{m,i}} + \frac{v_{m,i}}{a_{m,i}} \rightarrow v_{m,i}^2 = v_{m,i} a_{m,i} t_e - s_{e,i} a_{m,i}$$

$$v_{m,i} = \frac{a_{m,i} t_e}{2} - \sqrt{\frac{a_{m,i}^2 t_e^2}{4} - s_{e,i} a_{m,i}}$$

    - Analogous calculation for a **sinoid profile:**

$$v_{m,i} = \frac{a_{m,i} t_e}{4} - \sqrt{\frac{a_{m,i}^2 t_e^2 - 8 s_{e,i} a_{m,i}}{16}}$$

Fully Synchronous PTP Paths

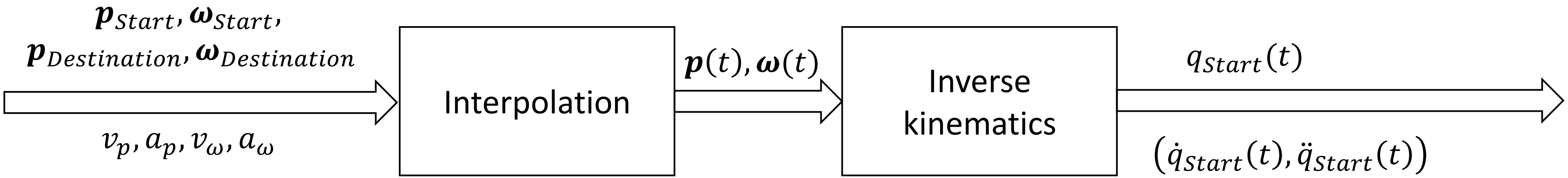
- Additional consideration of the **acceleration time and braking time**
- **Better approximation** of the start and end points in the workspace
- Determination of the leading axis with t_e and $t_a \rightarrow t_d = t_e - t_a$
- Determination of the maximum velocity and acceleration of the other axes:

$$v_{m,i} = \frac{s_{e,i}}{t_d} \qquad a_{m,i} = \frac{v_{m,i}}{t_a}$$

- Disadvantage: Acceleration of each axis is predetermined

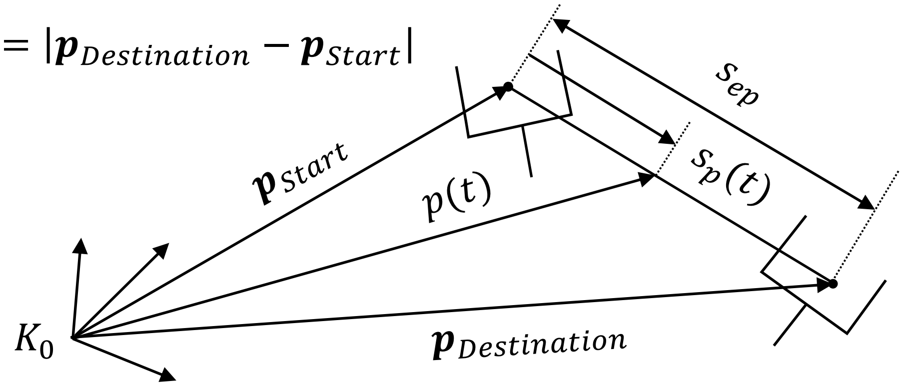
Control in the Workspace

- Continuous Path (CP)
  - End effector follows a **defined** path with regard to its position and orientation
- **Pose** of the end effector in the **workspace**
  - $\boldsymbol{p} = (x, y, z)^T \in \mathbb{R}^3$: **Position**
  - $\boldsymbol{\omega} = (\alpha, \beta, \gamma)^T \in \mathbb{R}^3$: **Orientation** (e.g. as Euler angles)
- Maximum velocities and accelerations in the work space:
  - $v_p \in \mathbb{R}$: Linear velocity
  - $a_p \in \mathbb{R}$: Linear acceleration
  - $v_\omega \in \mathbb{R}$: Angular velocity
  - $a_\omega \in \mathbb{R}$: Angular acceleration



Linear Interpolation (1)

$$s_{ep} = |\boldsymbol{p}_{Destination} - \boldsymbol{p}_{Start}|$$



$$p(t) = \boldsymbol{p}_{Start} + \frac{s_p(t)}{s_{ep}} \cdot (\boldsymbol{p}_{Destination} - \boldsymbol{p}_{Start})$$

Calculation of $s_p(t)$ with a ramp profile or a sinoid profile:

$$s_p(0) = \dot{s}_p(0) = v_p(0) = 0, \qquad \dot{s}_p(t_e) = v_p(t_e) = 0$$

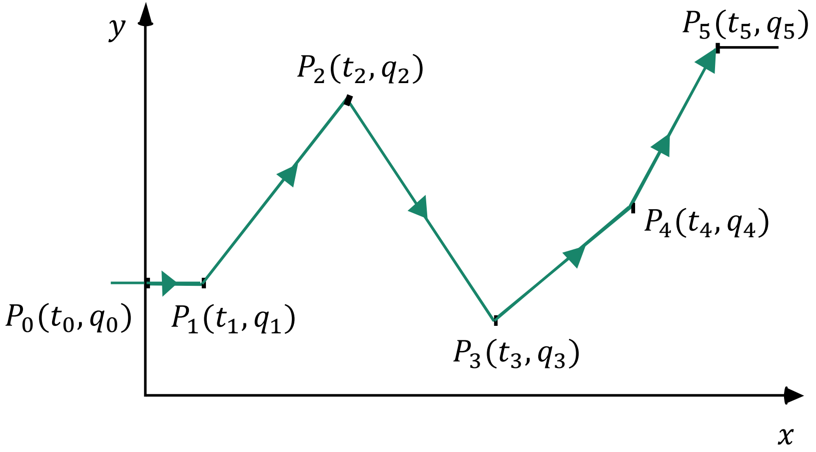
$$v_m = v_p, a_m = a_p, t_e = t_{ep}, t_a = t_{ap}, t_d = t_{dp}, s_e = s_{ep}, s = s_p$$

Linear Interpolation (2)

- Orientation in Euler angles: $\boldsymbol{\omega} = (\alpha, \beta, \gamma)^T$
  $$s_{e\omega} = |\boldsymbol{\omega}_{Destination} - \boldsymbol{\omega}_{Start}|$$
  $$= \sqrt{(\alpha_{Destination} - \alpha_{Start})^2 + (\beta_{Destination} - \beta_{Start})^2 + (\gamma_{Destination} - \gamma_{Start})^2}$$

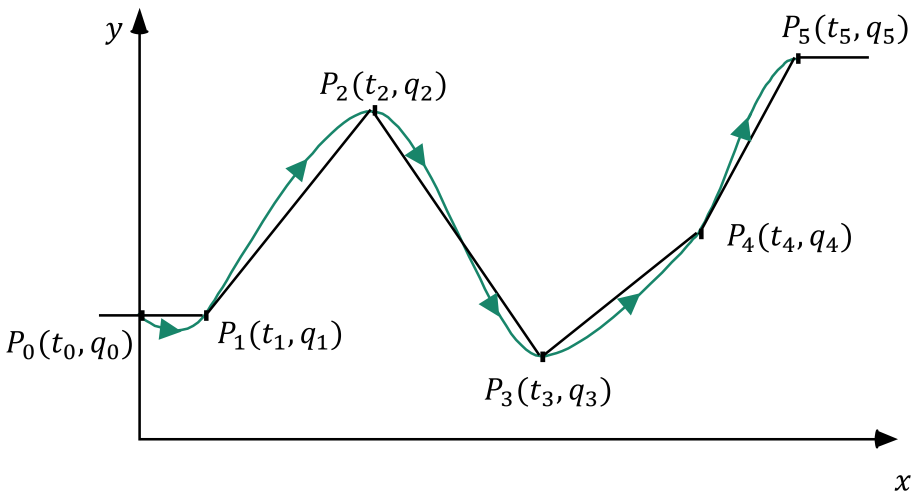
- Calculation of $s_\omega(t)$ with a ramp profile or a sinoid profile:
  $$v_m = v_\omega, \quad a_m = a_\omega, \quad t_e = t_{e\omega}, \quad t_a = t_{a\omega}, \quad t_d = t_{d\omega}, \quad s_e = s_{e\omega},$$
  $$s = s_\omega$$
- Synchronization of the traversing times t_{ep} (position) and $t_{e\omega}$ (orientation)
  $$t_e = \max(t_{ep}, t_{e\omega})$$
- Analogous to adjusting the velocities for synchronous PTP:
  - If $t_e = t_{ep}$: $$v_\omega = \frac{a_\omega t_e}{2} - \sqrt{\frac{a_\omega^2 t_e^2}{4} - s_{e\omega} a_\omega}$$
  - If $t_e = t_{e\omega}$: $$v_p = \frac{a_p t_e}{2} - \sqrt{\frac{a_p^2 t_e^2}{4} - s_{ep} a_p}$$

Linear Interpolation: Example



- The robot controller interpolates the path between 2 consecutive partial trajectories. 机器人控制器在两个连续的部分轨迹之间插补路径。

Segment-wise Path Interpolation



- The end conditions of the partial trajectory $j-1$ (direction, velocity, acceleration) and the start conditions of the partial trajectory j are adjusted to each other
- Partial trajectories are described separately (Example: Splines)

Interpolation with Cubic Splines (1)

- Polynomial

$$f(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \qquad (a_0, a_1, a_2, a_3 \in \mathbb{R})$$

- Given:
  - Starting point $f(0) = s_a$
  - End point $f(t_e) = s_e$
  - Starting velocity $\dot{f}(0) = v_a$
  - End velocity $\dot{f}(t_e) = v_e$

- Desired: $a_0, a_1, a_2, a_3 \in \mathbb{R}$

- Goal: Determine parameters for the polynomial

Cubic Splines: Determination of the Parameters (1)

- $f(0) = s_a$

- $\dot{f}(0) = v_a$

- $\dot{f}(t_e) = v_e$

Cubic Splines: Determination of the Parameters (2)

- $f(0) = s_a$

$$f(t=0) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 = a_0$$
$$\Rightarrow a_0 = s_a$$

- $\dot{f}(0) = v_a$

$$\dot{f}(t=0) = a_1 + 2a_2 t + 3a_3 t^2 = a_1$$
$$\Rightarrow a_1 = v_a$$

- $\dot{f}(t_e) = v_e$

$$a_1 + 2a_2 t_e + 3a_3 t_e^2 = v_e$$
$$v_a + 2a_2 t_e + 3a_3 t_e^2 = v_e$$
$$2a_2 t_e = v_e - v_a - 3a_3 t_e^2$$
$$a_2 = \frac{v_e - v_a}{2t_e} - \frac{3}{2} a_3 t_e$$

Cubic Splines: Determination of the Parameters (3)

- $a_0 = s_a$
- $a_1 = v_a$
- $a_2 = \frac{v_e - v_a}{2t_e} - \frac{3}{2} a_3 t_e$
- $f(t_e) = s_e$

$$a_0 + a_1 t_e + a_2 t_e^2 + a_3 t_e^3 = s_e$$

$$s_a + v_a t_e + \left(\frac{v_e - v_a}{2t_e} - \frac{3}{2} a_3 t_e \right) t_e^2 + a_3 t_e^3 = s_e$$

$$2 v_a t_e + (v_e - v_a) t_e - 3 a_3 t_e^3 + 2 a_3 t_e^3 = 2(s_e - s_a)$$

$$(v_e + v_a) t_e - a_3 t_e^3 = 2(s_e - s_a)$$

$$-a_3 t_e^3 = -(v_e + v_a) t_e$$

$$\Rightarrow \; a_3 = \frac{(v_e + v_a)}{t_e^2} - \frac{2(s_e - s_a)}{t_e^3}$$

Cubic Splines: Determination of the Parameters (4)

- $a_0 = s_a$
- $a_1 = v_a$
- $a_2 = \frac{v_e - v_a}{2t_e} - \frac{3}{2} a_3 t_e$
- $a_3 = \frac{(v_e + v_a)}{t_e^2} - \frac{2(s_e - s_a)}{t_e^3}$

$$a_2 = \frac{v_e - v_a}{2t_e} - \frac{3}{2} a_3 t_e$$

$$a_2 = \frac{v_e - v_a}{2t_e} - \frac{3}{2}\left(\frac{(v_e + v_a)}{t_e^2} - \frac{2(s_e - s_a)}{t_e^3}\right) t_e$$

$$a_2 = \frac{1}{2\mathrm{t}_e}(v_e - v_a - 3v_e - 3v_a) + \frac{3(s_e - s_a)}{t_e^2}$$

$$\Rightarrow \; a_2 = \frac{3(s_e - s_a)}{t_e^2} - \frac{v_e + 2v_a}{t_e}$$

Cubic Splines: Determination of the Parameters (5)

- Cubic polynomial

$$f(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

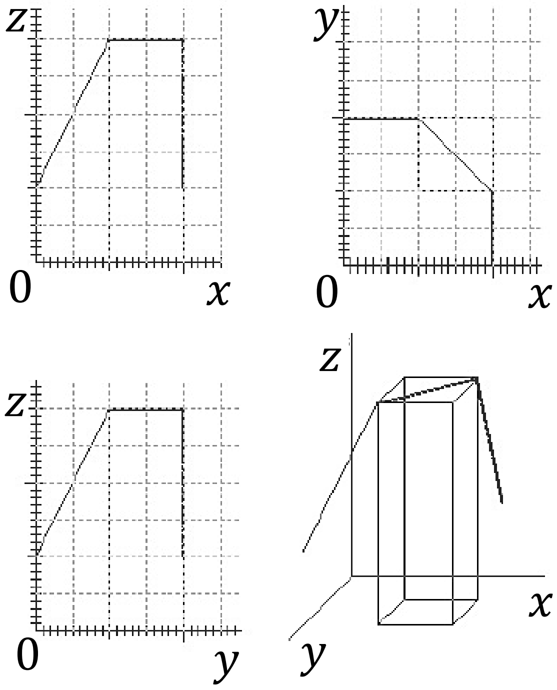
- Desired properties:
  - Starting point $\quad f(0) = s_a$
  - End point $\quad f(t_e) = s_e$
  - Starting velocity $\quad \dot{f}(0) = v_a$
  - End velocity $\quad \dot{f}(t_e) = v_e$

- Solution:

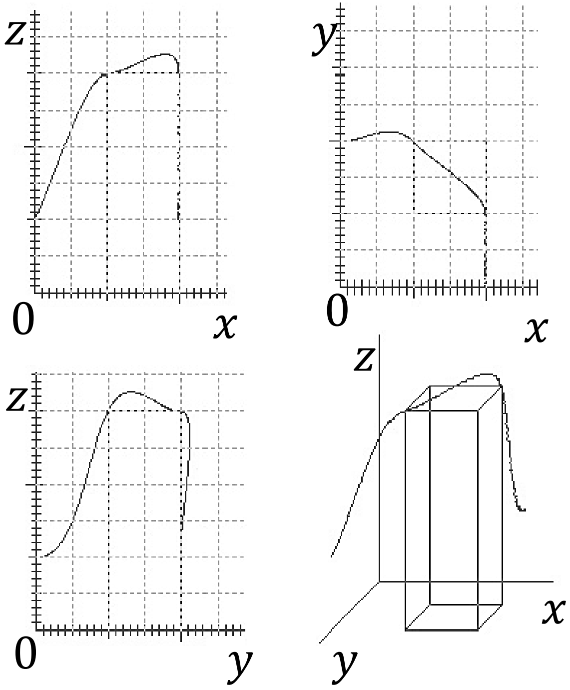
$$f(t) = s_a + v_a t + \left(\frac{3(s_e - s_a)}{t_e^2} - \frac{v_e + 2v_a}{t_e}\right) t^2 + \left(\frac{(v_e + v_a)}{t_e^2} - \frac{2(s_e - s_a)}{t_e^3}\right) \mathrm{t}^3$$

Spline Interpolation: Examples

- Path (4 support points)



- Spline interpolation

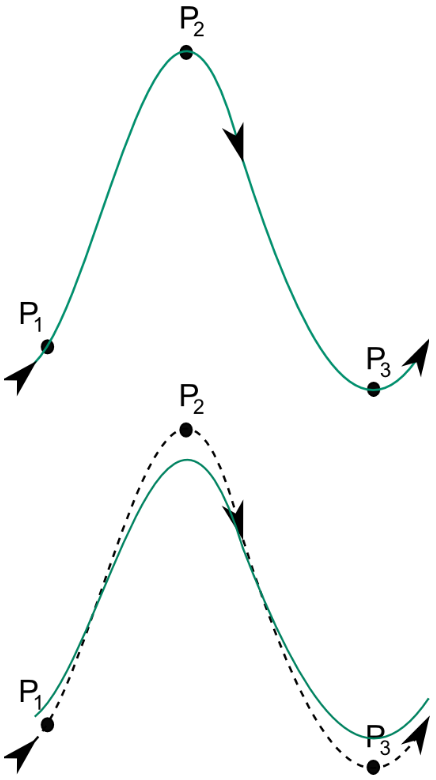


Outline

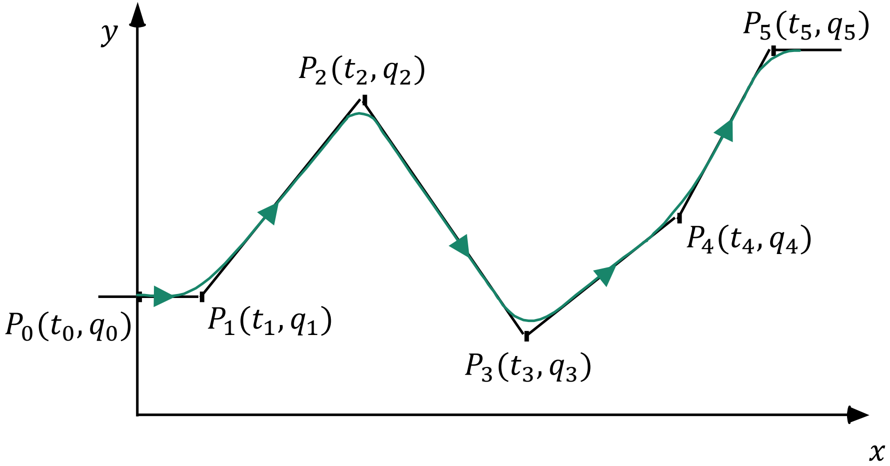
- Fundamentals of trajectory generation
- Programming of key points
- Interpolation types
- **Approximated trajectory generation**
  - **Bernstein polynomial**

Approximated Trajectory Generation: Definition

- Path interpolation:
  - The executed path traverses **all support points** of the trajectory
- Path approximation:
  - The support points influence the course of the path and are **approximated**



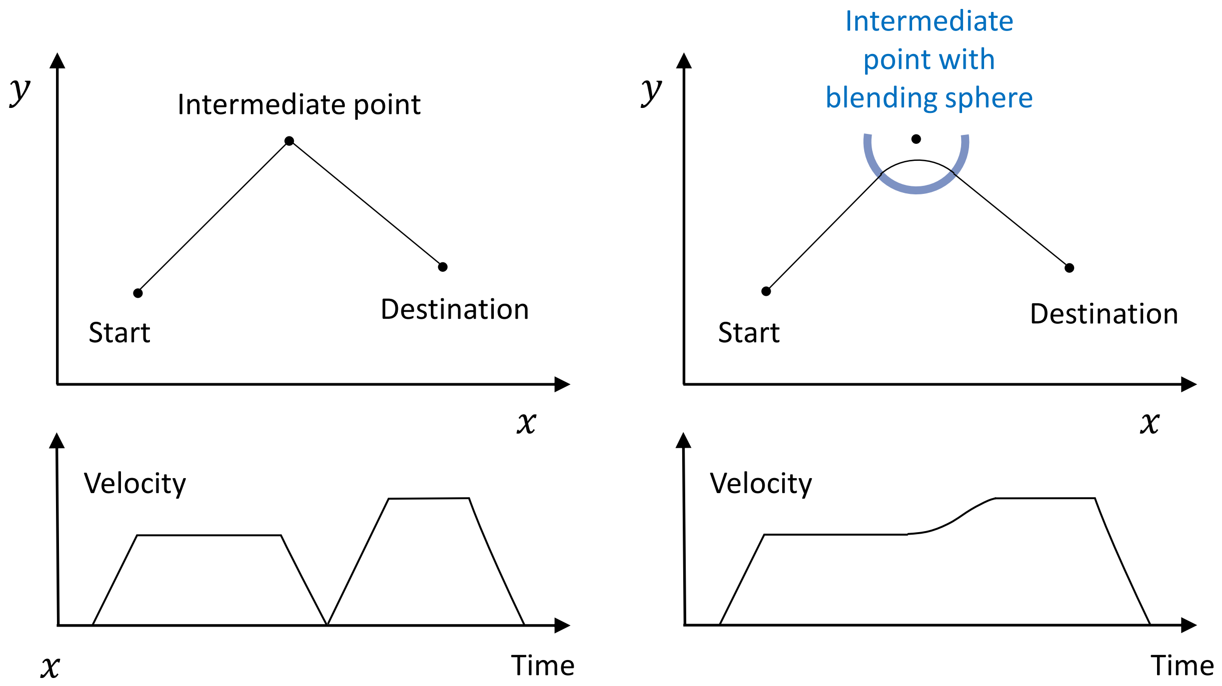
PTP and CP with Blending (1)



- At time point $t_j - \varepsilon$, start to transfer the parameters (direction and velocity) of the partial trajectory $j-1$ to the parameters of the partial trajectory j.

- Usually the **support point i is not reached**.

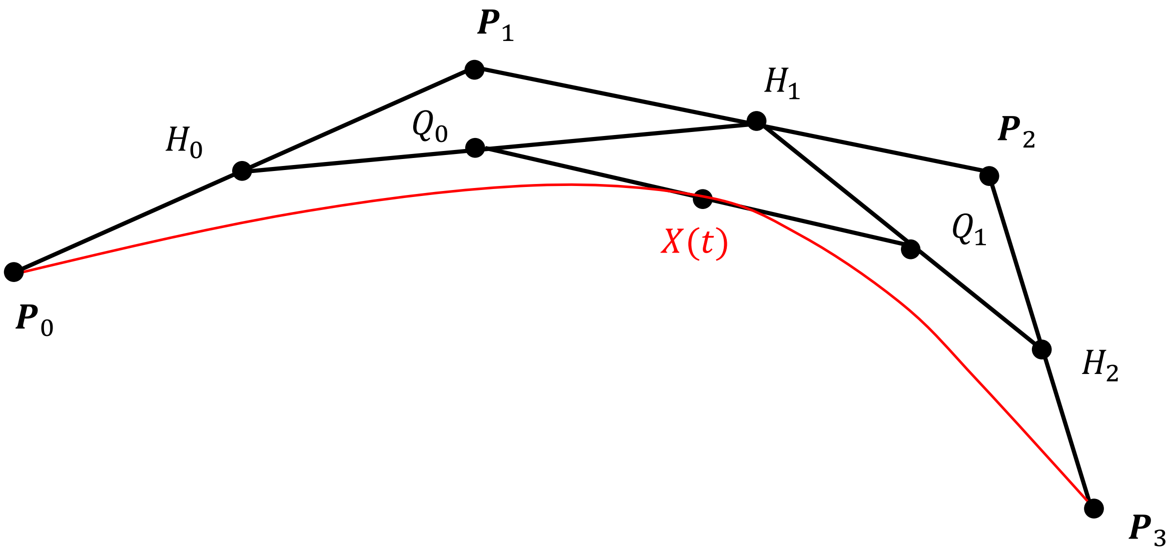
PTP and CP with Blending (2)



PTP and CP with Blending (3)

- **Velocity blending**
  - Start when the velocity falls below a specified minimum value
  - **Disadvantage**: Dependent on the velocity profile
- **Positional blending**
  - Start when the end effector enters the blending sphere
  - Outside of the blending sphere, the path is strictly adhered to.
  - **Advantage**: Easy to control

Approximation with Bernstein Polynomials



Bézier Curves (1)

- In contrast to cubic splines, **Bézier curves do not run through all support points $\boldsymbol{P}_i$**, but are only influenced by them.

- Basis function:

$$P(t) = \sum_{i=0}^{n} B_{i,n}(t)\,\boldsymbol{P}_i \quad 0 \leq t \leq 1$$

- $B_{i,n}(t)$: i–th **Bernstein polynomial** of degree n

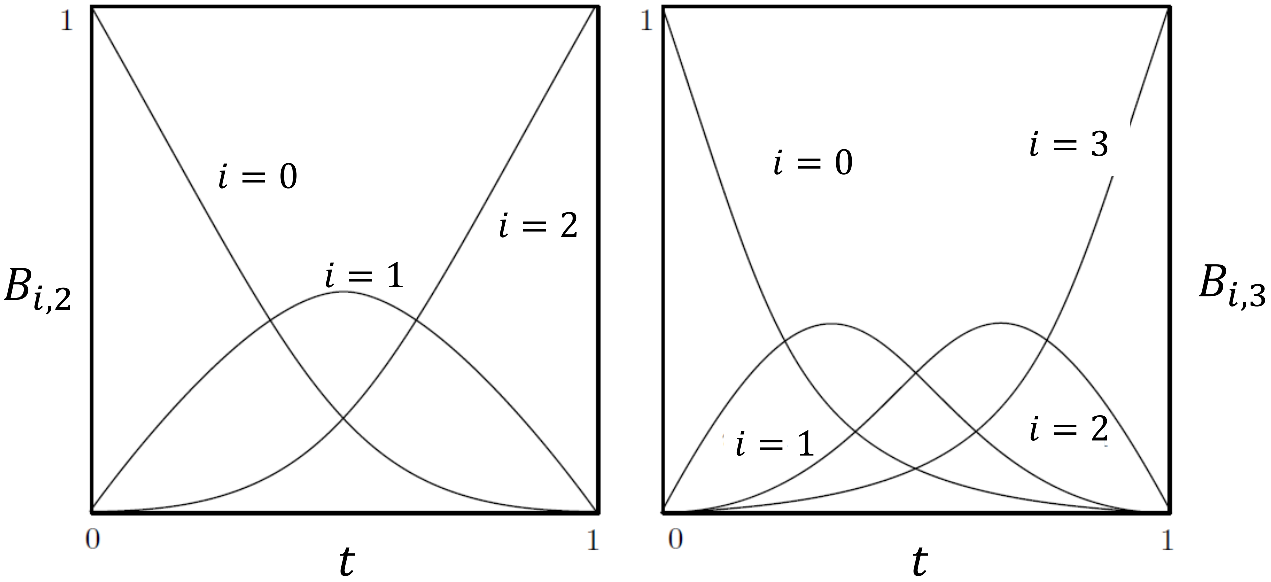
$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Bernstein Polynomials: Examples

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Quadratic polynomials $B_{i,2}$

Cubic polynomials $B_{i,3}$



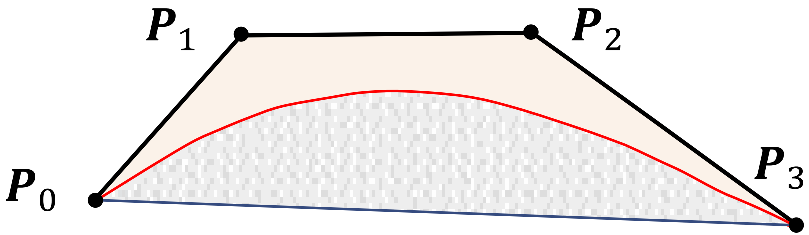
Bézier Curves (2)

- Calculation of arbitrary intermediate positions
- Example: Bernstein polynomial for the cubic case (Degree $n = 3$)

$$B_{i,3}(t) = \binom{3}{i} t^i (1-t)^{3-i}$$

$$P(t) = (1-t)^3 \boldsymbol{P}_0 + 3(1-t)^2 t \boldsymbol{P}_1 + 3(1-t)t^2 \boldsymbol{P}_2 + t^3 \boldsymbol{P}_3$$

- Approaching support points from below
- No arbitrary shape 无固定形状



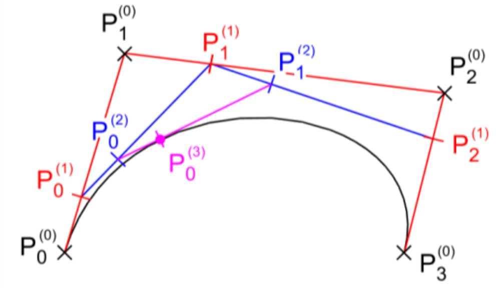
De Casteljau's Algorithm (1)

- **Approximation** of the **Bézier curve:**
- Efficient calculation of an approximate representation of Bézier curves using a polygonal chain
- **Idea:** Algorithm is based on **dividing a Bézier curve** and representing it by **two consecutive** Bézier curves
- **Iterative calculation**: Can be efficiently calculated even for large values of n

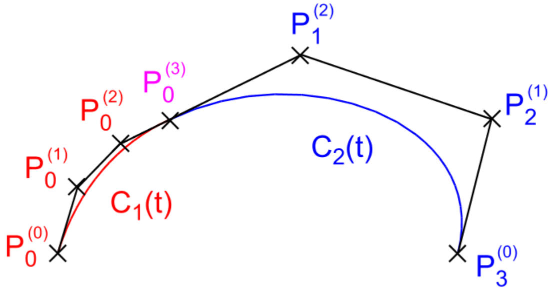
- Given: n support points $\boldsymbol{P}_0, \dots, \boldsymbol{P}_{n-1}$
- Start: $\boldsymbol{P}_i^0 = \boldsymbol{P}_i$
- Iteration k: $\boldsymbol{P}_i^{k+1} = (1 - t_0)\boldsymbol{P}_i^k + t_0\boldsymbol{P}_{i+1}^k$

De Casteljau's Algorithm (2)

- Example for $\boldsymbol{P}_0$ with $k = 3$ and $t_0 = 0{,}25$:



- **Two Bézier curves** $C_1(t)$ and $C_2(t)$
- Approximation of the Bézier curve using a polygonal chain



The End!