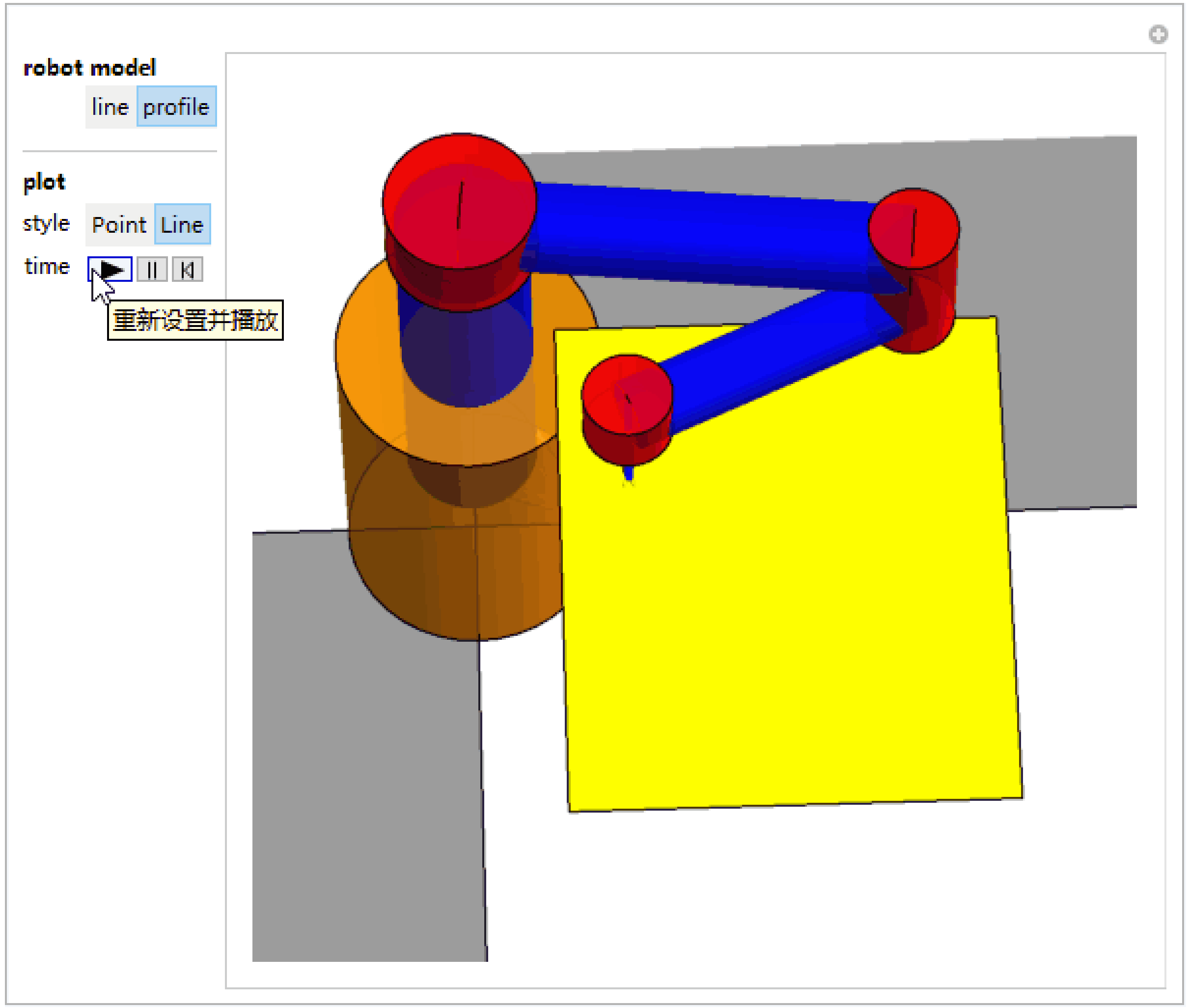
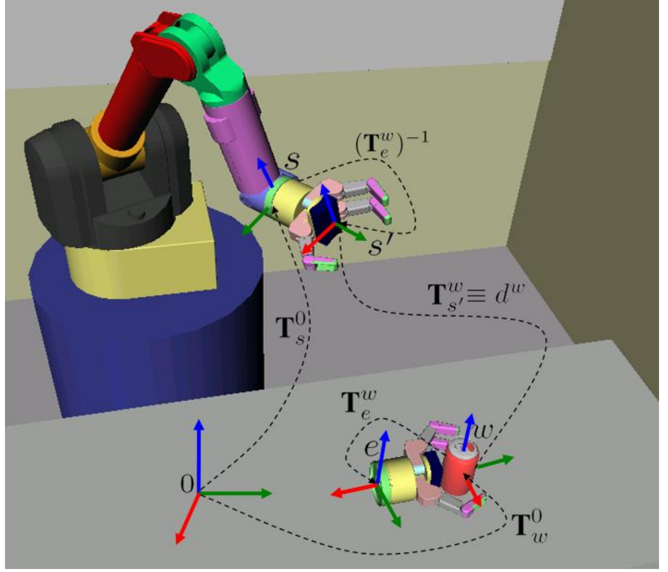
Motion-Trajectory Planning in joints space



Trajectory Planning



The goal of ***trajectory planning*** is to generate the reference inputs to the motion control system which ensures that the manipulator executes the planned trajectories.

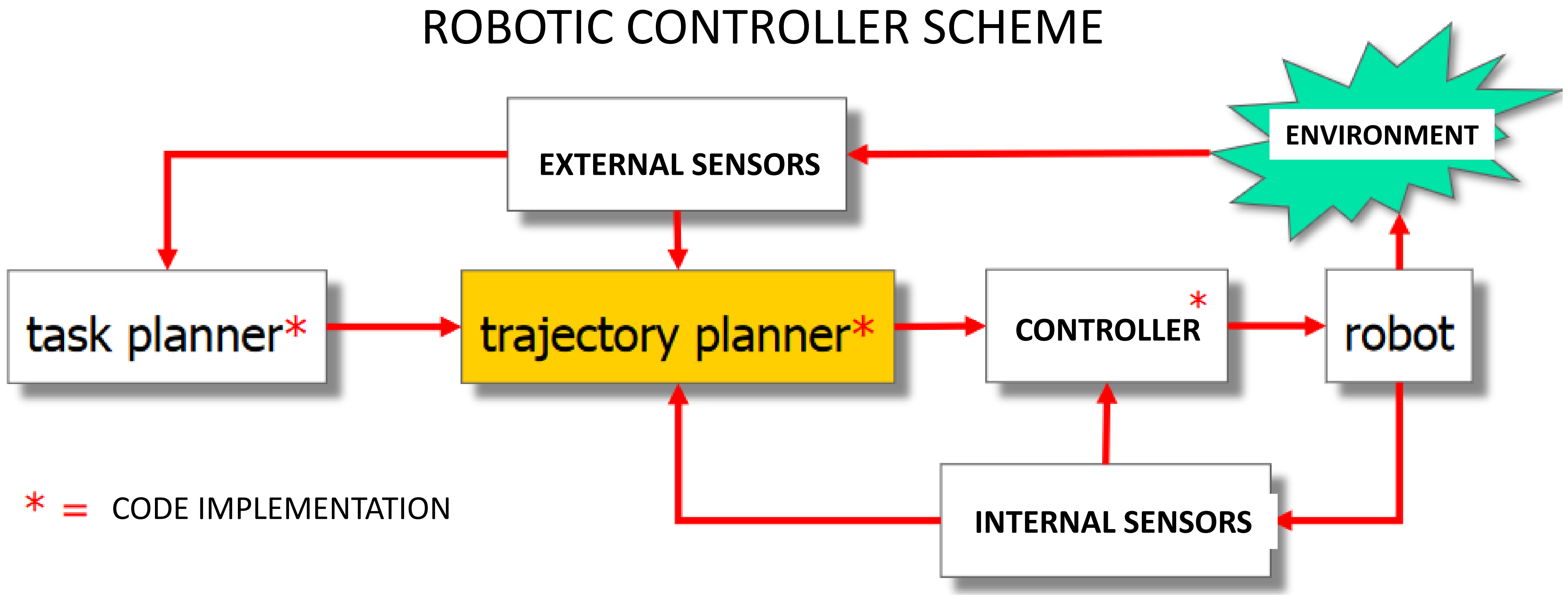
The user typically specifies a number of parameters to describe the desired trajectory.

- Planning consists of generating a time sequence of the values attained by an interpolating function (typically a polynomial) of the desired trajectory

techniques for trajectory generation,

1. in the case when the initial and final point of the path are assigned (*point-to-point motion*),
2. in the case when a finite sequence of points are assigned along the path (*motion through a sequence of points*).

Motion-Trajectory Planning



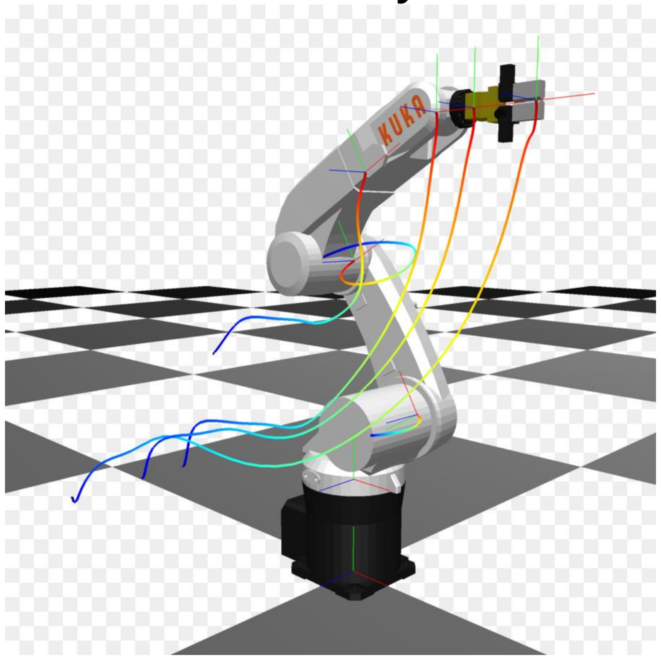
robot action described as a sequence of poses or configurations (with possible exchange of contact forces) → **TRAJECTORY PLANNER** → reference profile/values (continuous or discrete) for the robot controller

Path and Trajectory

The minimal requirement for a manipulator is the capability to move from an initial posture to a final assigned posture.

The transition should be characterized by motion laws requiring the actuators to exert joint generalized forces which do not violate the **saturation limits** and do not excite the typically modelled **resonant modes** of the structure.

It is then necessary to devise planning algorithms that generate **suitably smooth trajectories**.

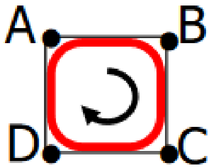


Trajectory definition

1. define Cartesian pose points (position+orientation) using the teach-box
2. program an (average) velocity between these points, as a 0-100% of a maximum system value (different for Cartesian- and joint-space motion)
3. linear interpolation in the joint space between points sampled from the built trajectory

examples of additional features

a) over-fly



b) sensor-driven STOP

c) circular path through 3 points

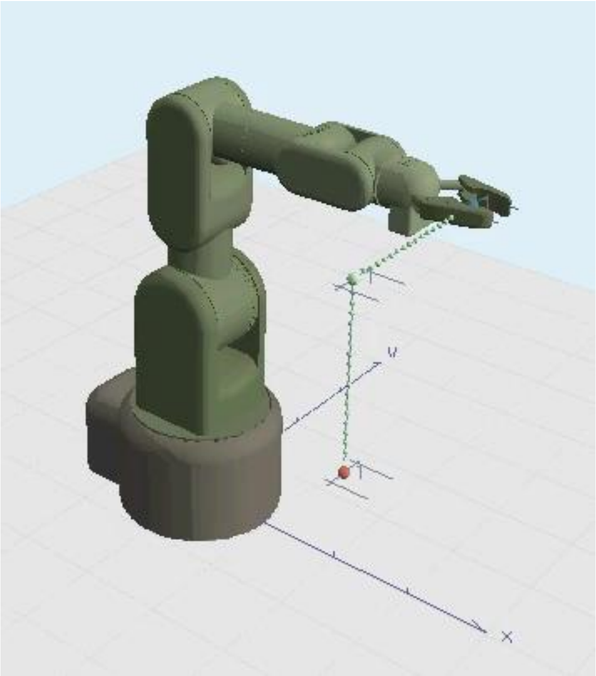
main drawbacks

- semi-manual programming (as in "first generation" robot languages)
- limited visualization of motion

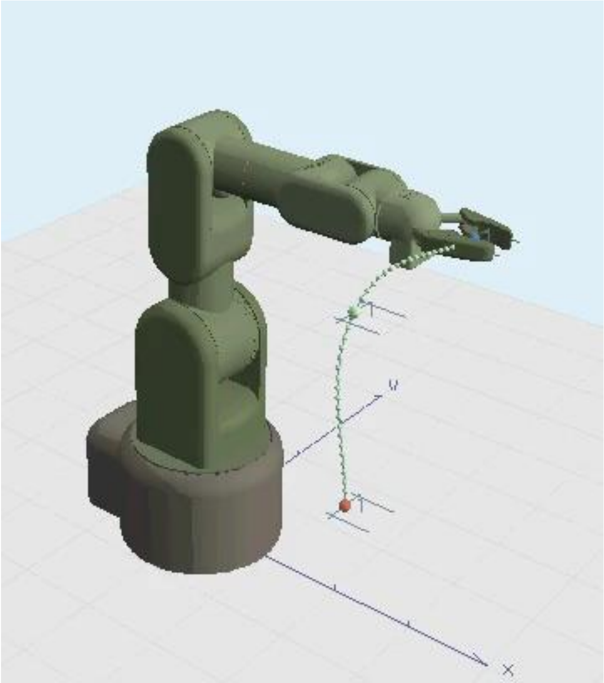
➡ a mathematical formalization of trajectories is useful/needed

Some typical trajectories

- Point-to-point Cartesian motion with an intermediate point

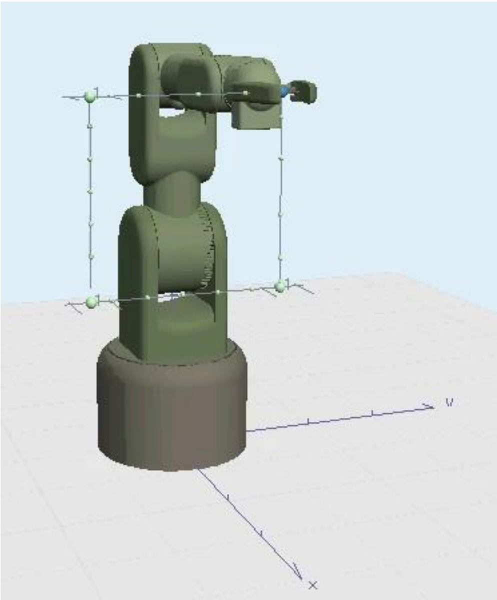


Straight lines as Cartesian path

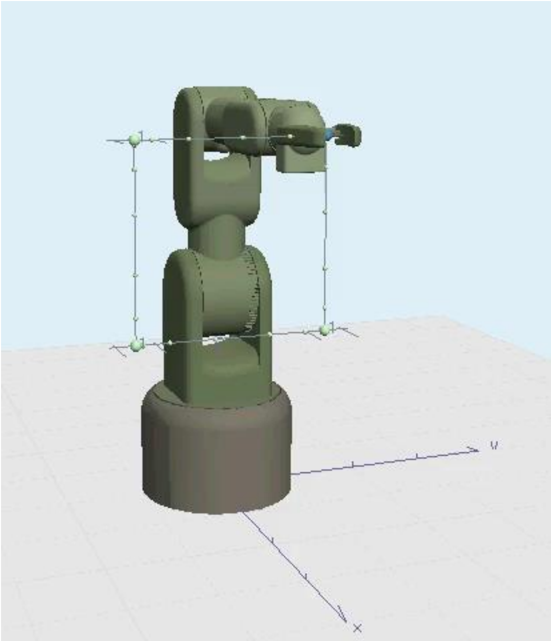


Interpolation with Bezier curves

Some typical trajectories



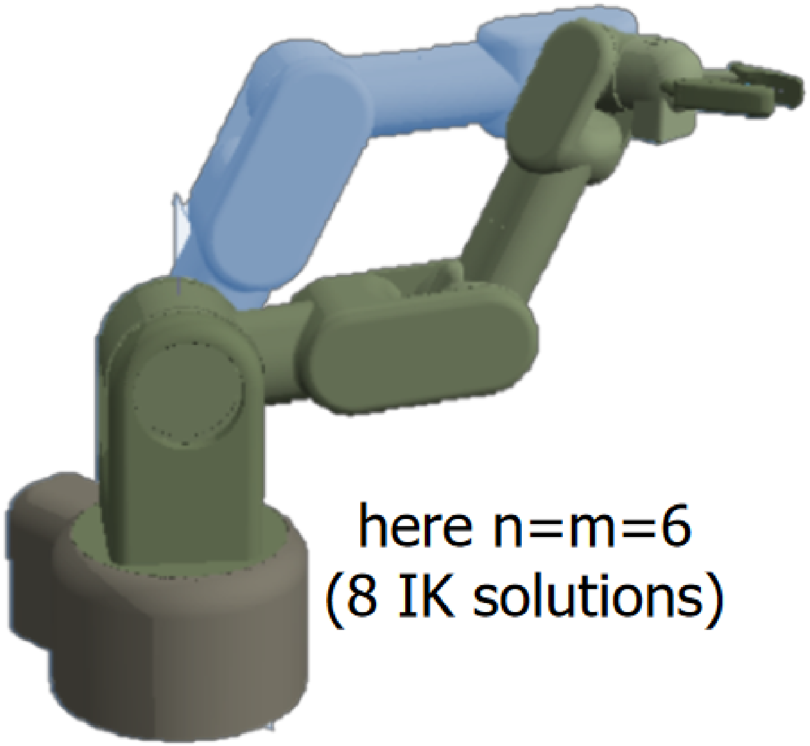
Square path at constant speed



Square path with
trapezoidal speed profile

Joint and Cartesian trajectories

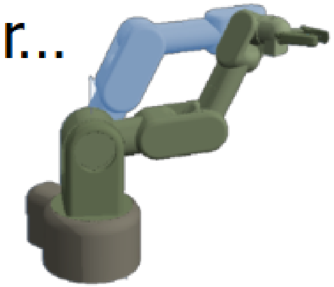
- assigned task: arm reconfiguration between two inverse kinematic solutions associated to a given end-effector pose

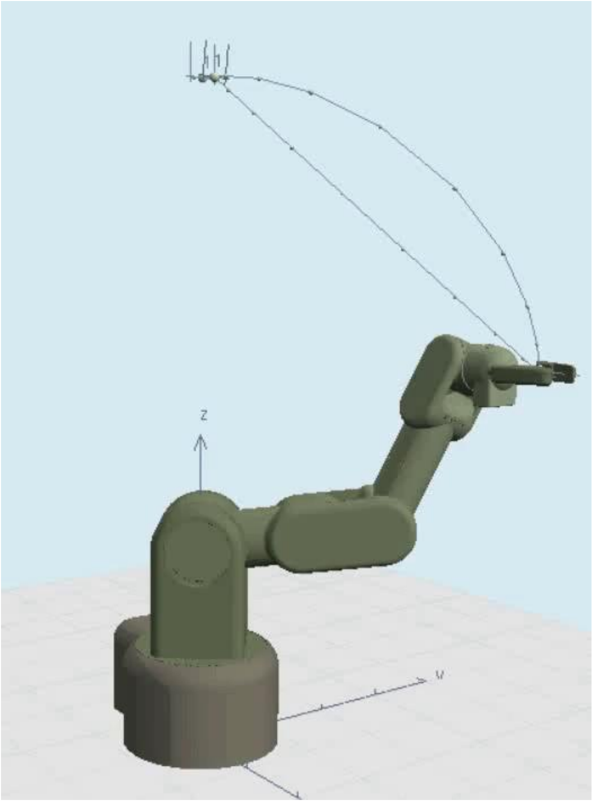


- **initial** and **final** configuration
- same Cartesian pose (no change!): the motion cannot be fully specified in the Cartesian space
- to perform this task, the robot should leave the given end-effector pose and then return to it
- a self-motion could be sufficient
  - if the robot starts in a singularity
  - if there is (task) redundancy (m<n)

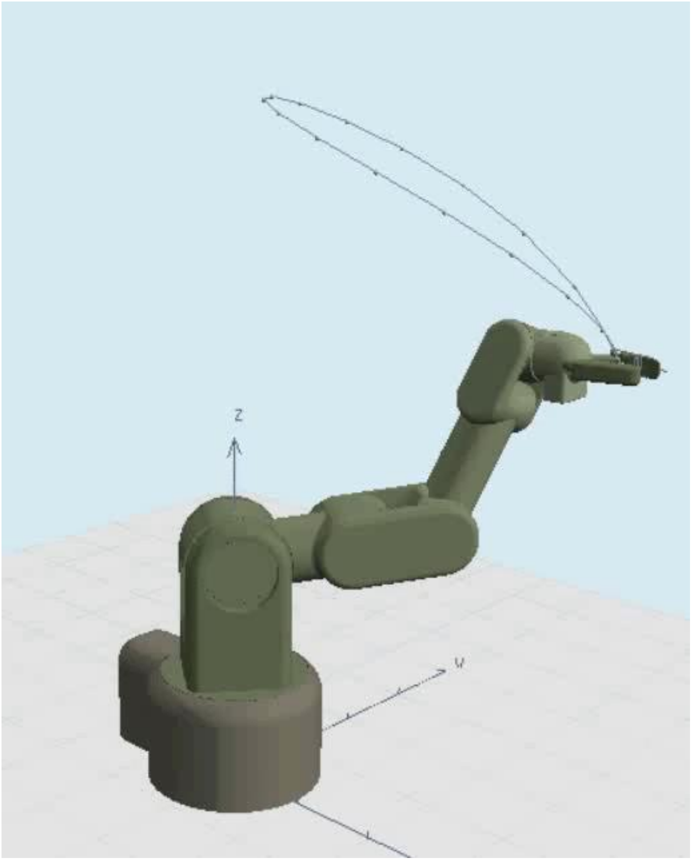
for "simple" manipulators (e.g., all industrial robots) and m=n, the execution of these tasks will require the passage through a singular configuration

Joint and Cartesian trajectories

- a reconfiguration task (or…  passing through singularity)



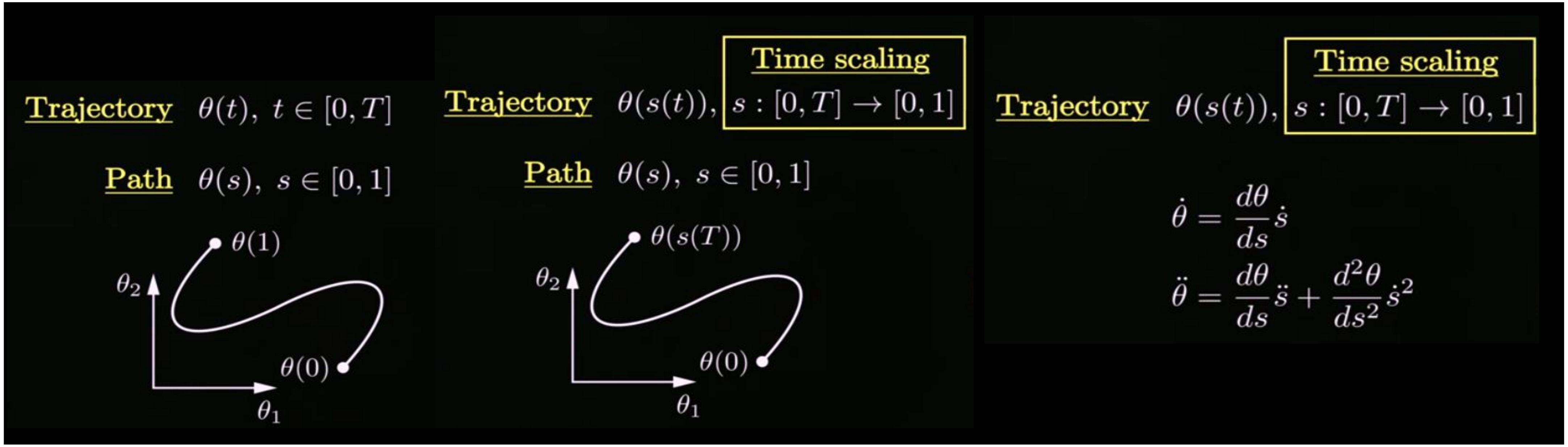
three-phase trajectory:
circular path + self-motion + linear path



single-phase trajectory
in the joint space (no stops)

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Path and Trajectory



A ***path*** denotes the locus of points in the joint space, or in the operational space, which the manipulator has to follow in the execution of the assigned motion; a path is then a pure geometric description of motion.

A ***trajectory*** is a path on which a timing law is specified, for instance in terms of velocities and/or accelerations at each point

A ***trajectory planning*** algorithm are the path description in terms of time sequence of the values attained by position, velocity and acceleration.

Path and Trajectory

TRAJECTORY $\begin{cases} \textbf{Motion} & p_d(t) \\ \textbf{Interaction} & F_d(t) \end{cases}$

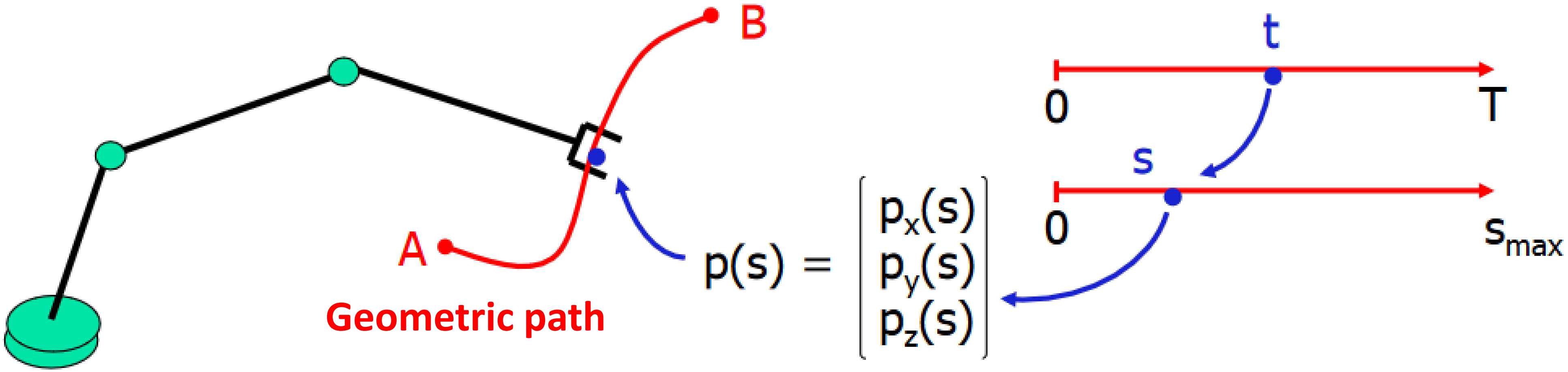
=

GEOMETRIC PATH — **It is parametric in *S*** $\quad p = p(s)$

***S* can describe a curve path** $\qquad \Big\} \; p(s(t))$

+

TIME LAW — **Describe how *S* changes** $\quad s = s(t)$



Trajectory planning

operative sequence

TASK planning

1

- sequence of pose points ("knots") in Cartesian space

interpolation in Cartesian space

- Cartesian geometric path (position + orientation): $p = p(s)$

path sampling and kinematic inversion

2

- sequence of "knots" in joint space

interpolation in joint space

- geometric path in joint space: $q = q(\lambda)$

analytic inversion

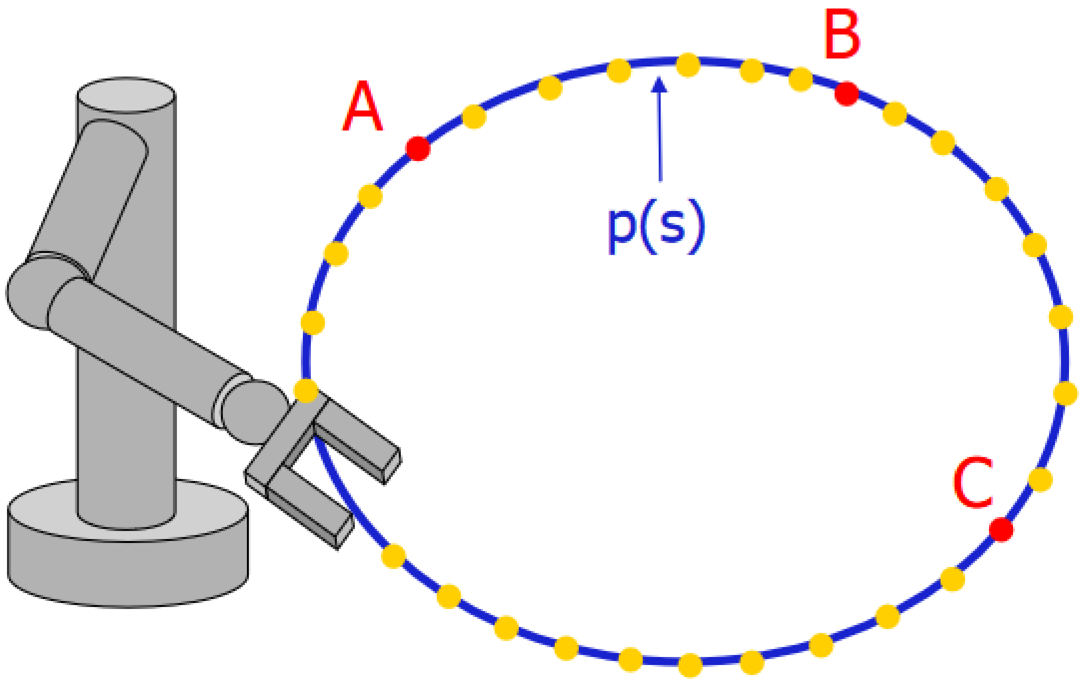
additional issues to be considered in the planning process

- obstacle avoidance
- on-line/off-line computational load
- sequence 2 is more "dense" than 1

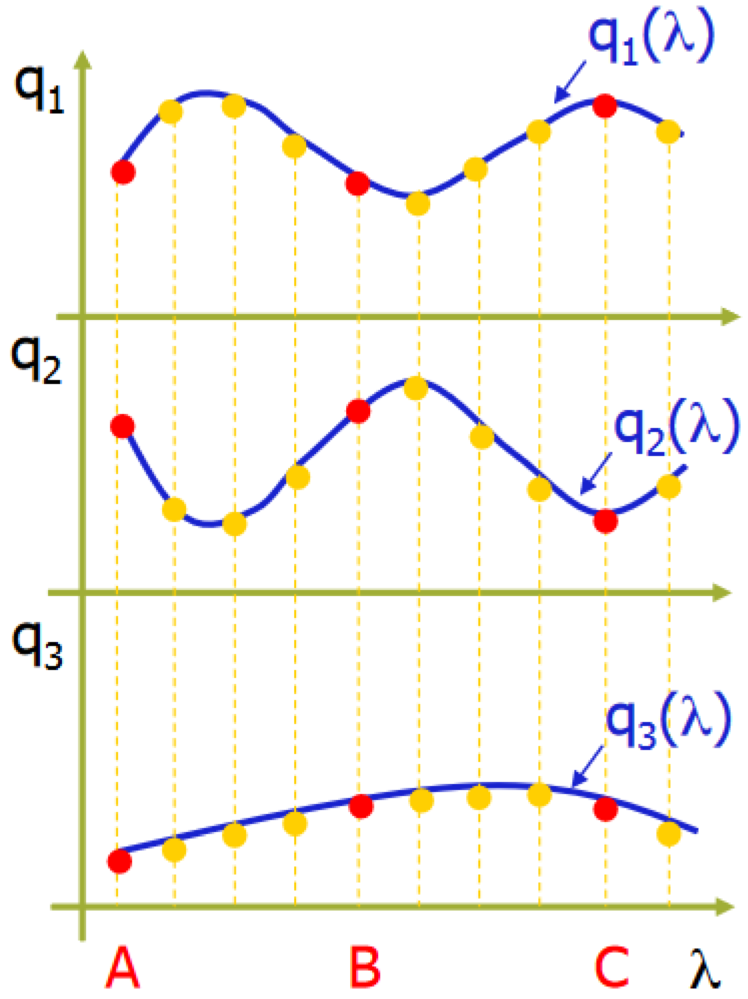
Example

$p = p(s) \quad s = s(t)$ **Cartesian space**

$q = q(\lambda) \quad \lambda = \lambda(t)$ **Joint space**



Cartesian space



Joints space

Joint space trajectories

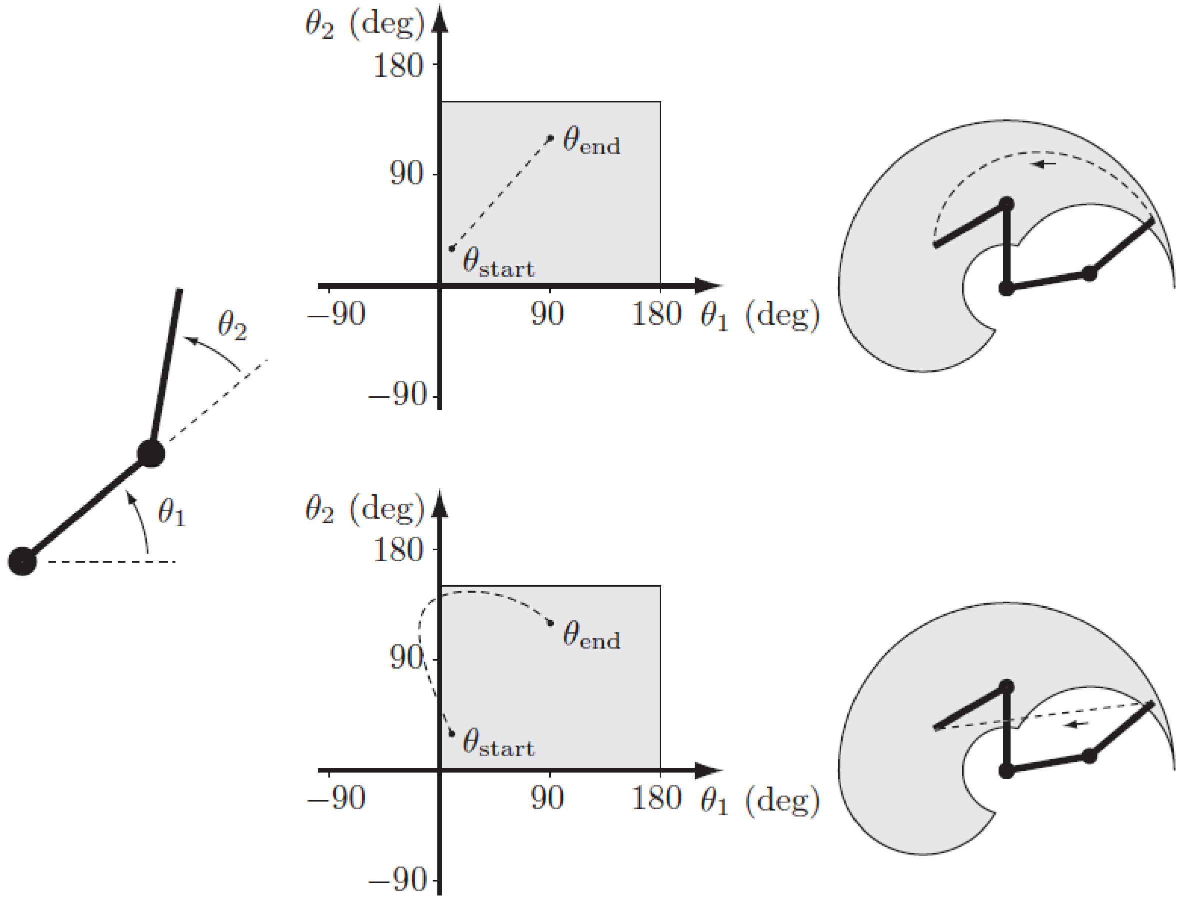
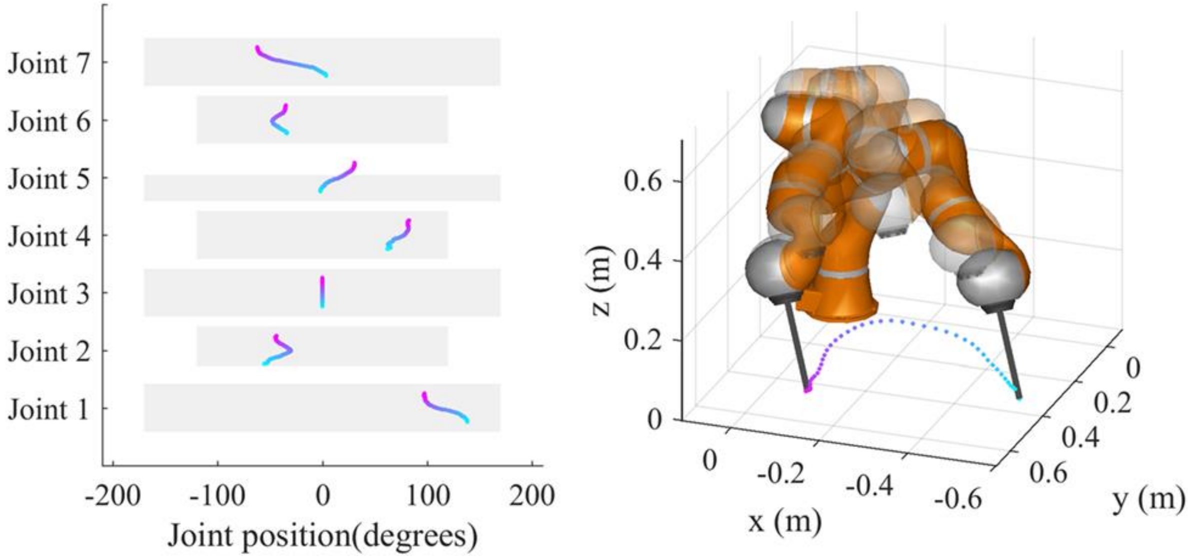


Figure (Left) A 2R robot with joint limits $0° \leq \theta_1 \leq 180°$, $0° \leq \theta_2 \leq 150°$. (Top center) A straight-line path in joint space and (top right) the corresponding motion of the end-effector in task space (dashed line). The reachable endpoint configurations, subject to joint limits, are indicated in gray. (Bottom center) This curved line in joint space and (bottom right) the corresponding straight-line path in task space (dashed line) would violate the joint limits.

Joint Space Trajectories

A manipulator motion is assigned in the operational space in terms of trajectory parameters such as:

- the initial and final end-effector pose,
- possible intermediate poses,
- and travelling time along particular geometric paths



If it is desired to plan a trajectory in the *joint space*, It is then necessary to resort to an inverse kinematics algorithm:

- if planning is **done off-line**, or to directly measure the

above variables,

- if planning is **done by the teaching-by-showing** technique

Joint Space Trajectories

Example of teaching by demonstration

https://www.youtube.com/watch?v=eJCMyrCm_V0

Joint Space Trajectories

The planning algorithm generates a function $\boldsymbol{q}(t)$ in respect of the imposed constraints.

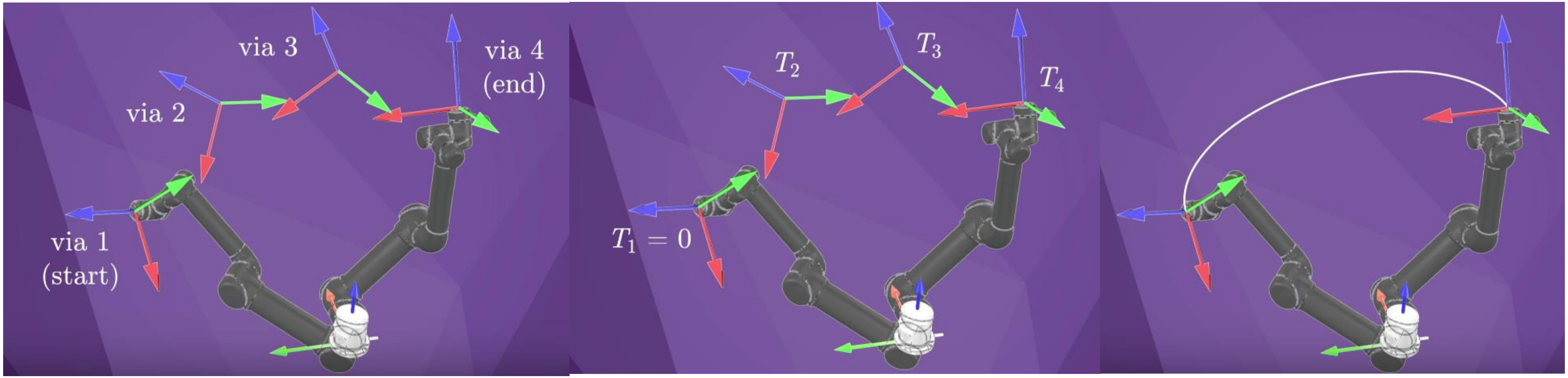
In general, a joint space trajectory planning algorithm is required to have the following features:

• the generated trajectories should be **not computationally demanding**,

• the joint positions and velocities should be continuous functions of time

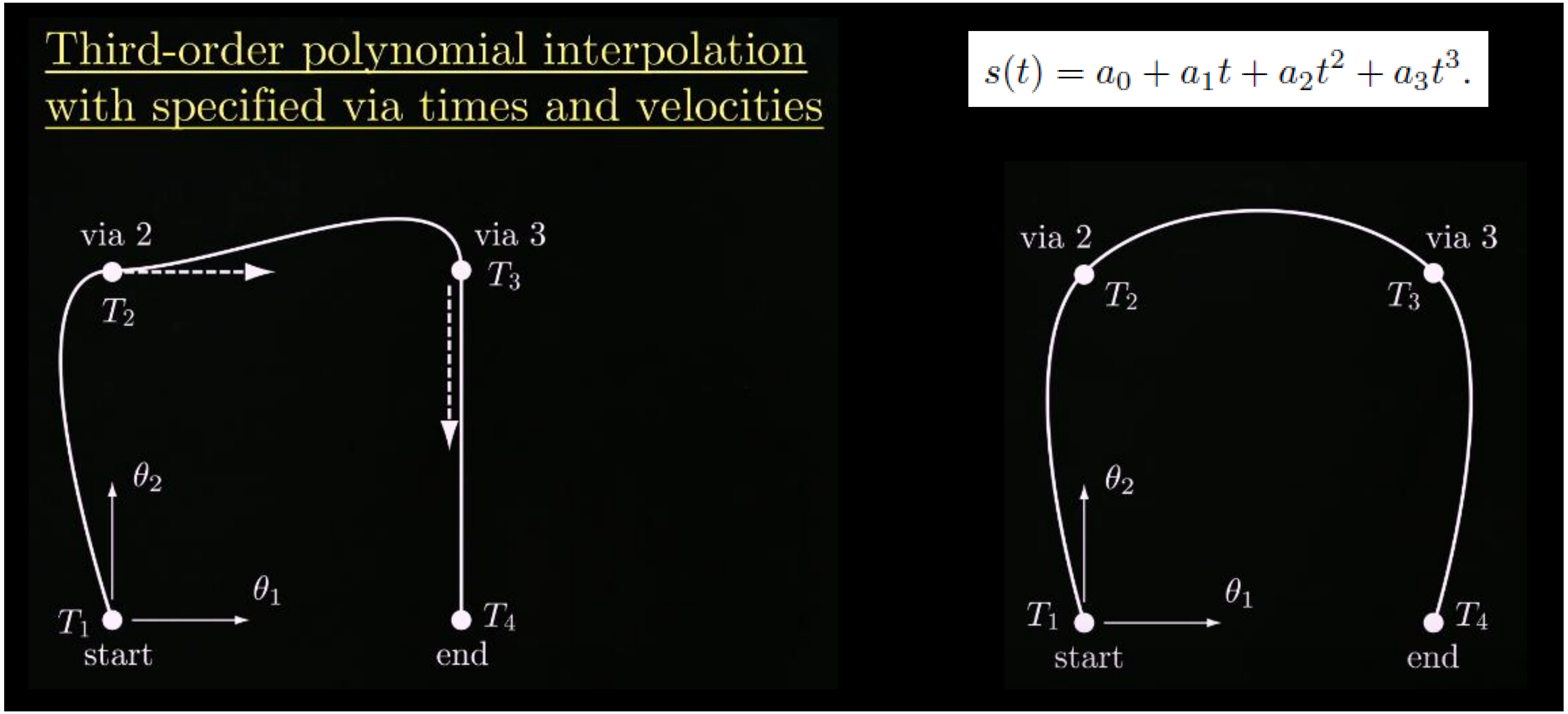
• undesirable effects should be minimized, e.g., **nonsmooth trajectories interpolating a sequence of points on a path.**

Via points-time-trajectory



In generating a trajectory we have to specify multiple constraints.
Usually are positions which are sequential **VIA POINTS** which must be reached in specific time instants **T_i**, in order to have a trajectory which responds to geometric and time specifications

Interpolating trajectories by polynimials



These are example of two trajectories generated by 4 points and using third order polynomials: to be noted that each point must be reached at a specific time and a specific velocity. These constraints will give the conditions to determine the coefficients of the polynomials.

Path and timing law

- after choosing a path, the trajectory definition is completed by the choice of a timing law

$$p = p(s) \quad \Rightarrow s = s(t) \qquad \text{(Cartesian space)}$$

$$q = q(\lambda) \quad \Rightarrow \lambda = \lambda(t) \qquad \text{(joint space)}$$

  - if $s(t) = t$, path parameterization is the natural one given by time

- the timing law
  - is chosen based on task specifications (stop in a point, move at constant velocity, and so on)
  - may consider optimality criteria (min transfer time, min energy,…)
  - constraints are imposed by actuator capabilities (max torque, max velocity,…) and/or by the task (e.g., max acceleration on payload)

note: on parameterized paths, a space-time decomposition takes place

e.g., in Cartesian space $\quad \dot{p}(t) = \frac{dp}{ds}\dot{s} \qquad \ddot{p}(t) = \frac{dp}{ds}\ddot{s} + \frac{d^2p}{ds^2}\dot{s}^2$

Cartesian vs. joint trajectory planning

- planning in Cartesian space
  - allows a more direct visualization of the generated path
  - obstacle avoidance, lack of "wandering"
- planning in joint space
  - does not need on-line kinematic inversion
- issues in kinematic inversion
  - $\dot{q}$ e $\ddot{q}$ (or higher-order derivatives) may also be needed
    - Cartesian task specifications involve the geometric path, but also bounds on the associated timing law
  - for redundant robots, choice among ∞^{n-m} inverse solutions, based on optimality criteria or additional auxiliary tasks
  - off-line planning in advance is not always feasible
    - e.g., when interaction with the environment occurs or sensor-based motion is needed

Trajectory classification

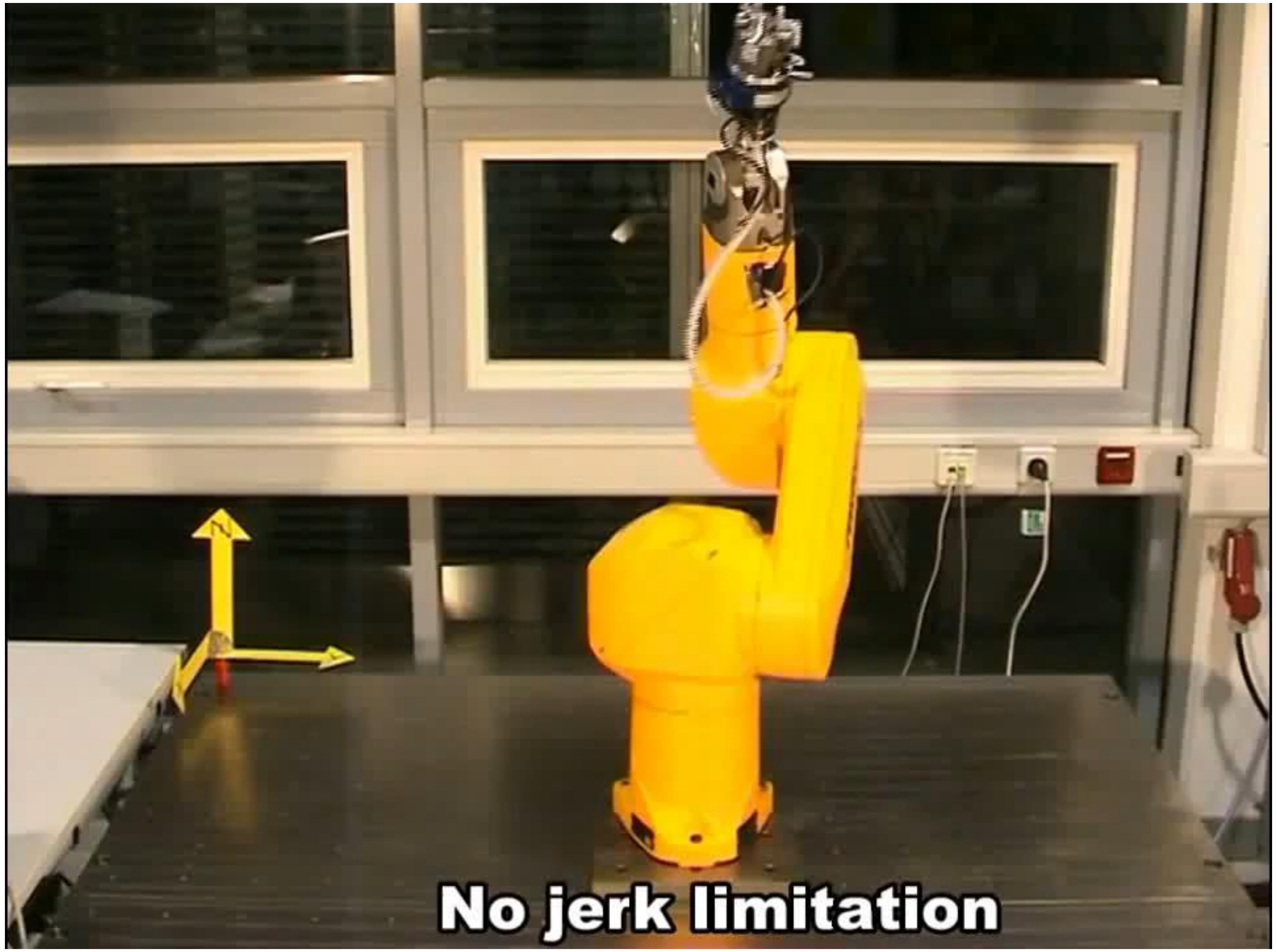
- space of definition
  - Cartesian, joint
- task type
  - point-to-point (PTP), multiple points (knots), continuous, concatenated
- path geometry
  - rectilinear, polynomial, exponential, cycloid, …
- timing law
  - bang-bang in acceleration, trapezoidal in velocity, polynomial, …
- coordinated or independent
  - motion of all joints (or of all Cartesian components) start and ends at the same instants (say, t=0 and t=T) = single timing law

  or
  - motions are timed independently (according to the requested displacement and robot capabilities) – mostly only in joint space

Relevant characteristics

- computational efficiency and memory space
  - e.g., store only the coefficients of a polynomial function
- predictability and accuracy
  - vs. "wandering" out of the knots
  - vs. "overshoot" on final position
- flexibility
  - allowing concatenation of primitive segments
  - over-fly
  - ...
- continuity
  - in space and/or in time
  - at least C^1, but also up to jerk = third derivative in time

A robot trajectory with bounded jerk



Trajectory planning in joint space

- $q = q(t)$ in time or $q = q(\lambda)$ in space (then with $\lambda = \lambda(t)$)
- it is sufficient to work component-wise (q_i in vector q)
- an implicit definition of the trajectory, by solving a problem with specified boundary conditions in a given class of functions
- typical classes: polynomials (cubic, quintic,…), (co)sinusoids, clothoids, …

- imposed conditions
  - passage through points = interpolation
  - initial, final, intermediate velocity (or geometric tangent for paths)
  - initial, final acceleration (or geometric curvature)
  - continuity up to the k-th order time (or space) derivative: class $\mathbf{C}^k$

many of the following methods and remarks can be
directly applied also to Cartesian trajectory planning (and vice versa)!

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Cubic polynomial in space

$\boxed{q(0) = q_0}$ $\boxed{q(1) = q_1}$ $\boxed{q'(0) = v_0}$ $\boxed{q'(1) = v_1}$ ← 4 conditions

$\Delta q = q_1 - q_0$

$\lambda \in [0,1]$

$$q(\lambda) = q_0 + \Delta q \, \underbrace{[a\,\lambda^3 + b\,\lambda^2 + c\,\lambda + d]}$$

4 coefficients → "doubly normalized" polynomial $q_N(\lambda)$

$q_N(0) = 0 \Leftrightarrow d = 0$

$q_N(1) = 1 \Leftrightarrow a + b + c = 1$

$q_N'(0) = dq_N/d\lambda|_{\lambda=0} = c = v_0/\Delta q$

$q_N'(1) = dq_N/d\lambda|_{\lambda=1} = 3a + 2b + c = v_1/\Delta q$

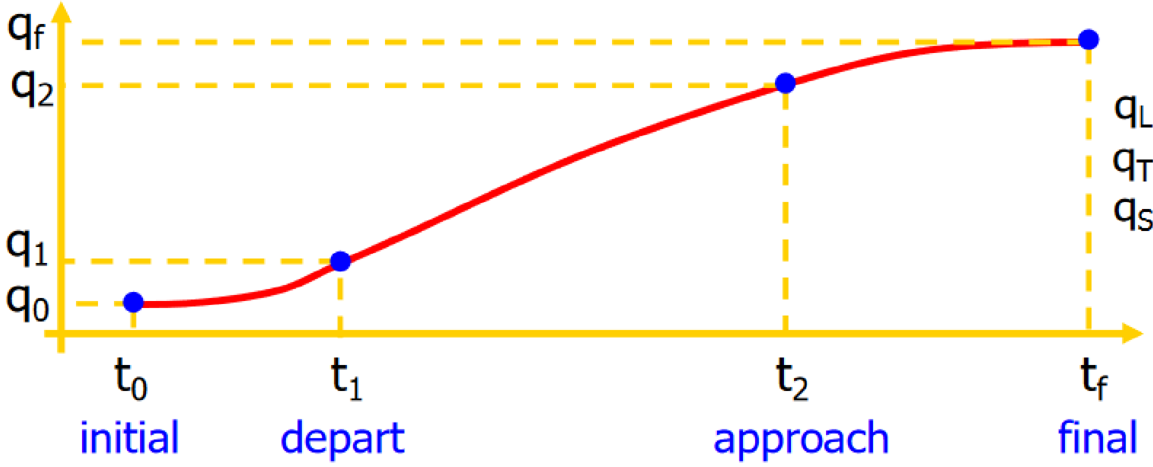
special case: $v_0 = v_1 = 0$ (zero tangent)

$q_N'(0) = 0 \Leftrightarrow c = 0$

$$\left.\begin{array}{l} q_N(1) = 1 \Leftrightarrow a + b = 1 \\ q_N'(1) = 0 \Leftrightarrow 3a + 2b = 0 \end{array}\right\} \Leftrightarrow \begin{array}{l} a = -2 \\ b = 3 \end{array}$$

4-3-4 polynomials

three phases (Lift off, Travel, Set down) in a pick-and-place operation in time



$$\left.\begin{aligned} q_L(t) &= \text{4th order polynomial} \\ q_T(t) &= \text{3rd order polynomial} \\ q_S(t) &= \text{4th order polynomial} \end{aligned}\right\}$$

14 coefficients

boundary conditions

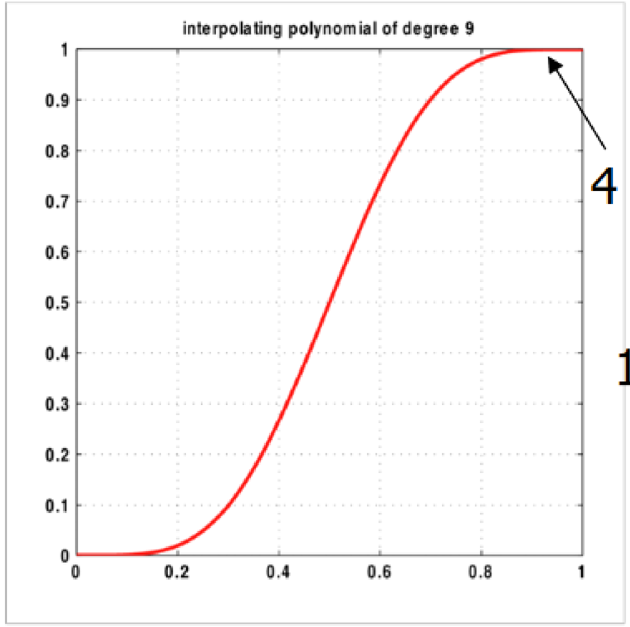
$$\left. q(t_0) = q_0 \quad q(t_1^-) = q(t_1^+) = q_1 \quad q(t_2^-) = q(t_2^+) = q_2 \quad q(t_f) = q_f \right\} \text{ 6 passages}$$

$$\left. \dot{q}(t_0) = \dot{q}(t_f) = 0 \qquad \ddot{q}(t_0) = \ddot{q}(t_f) = 0 \right\} \text{ 4 initial/final velocity/acceleration}$$

$$\left. \dot{q}(t_i^-) = \dot{q}(t_i^+) \qquad \ddot{q}(t_i^-) = \ddot{q}(t_i^+) \quad i = 1,2 \right\} \text{ 4 continuity}$$

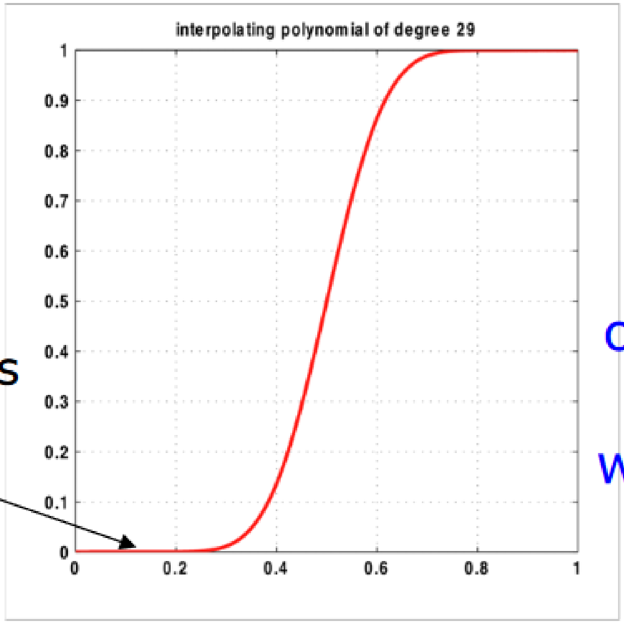
Numerical examples

9th degree



4 derivatives are zero

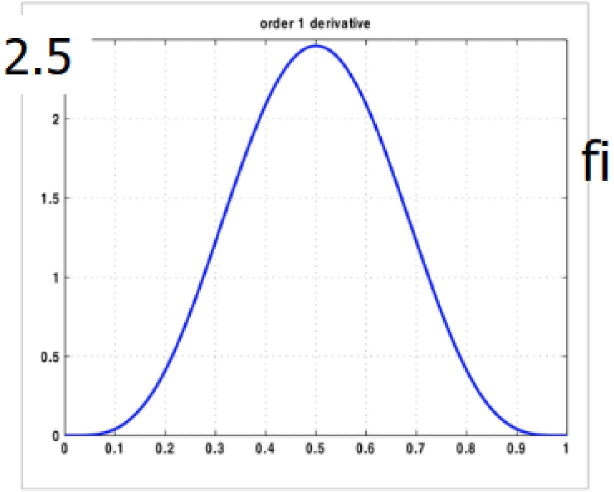
14 derivatives are zero!



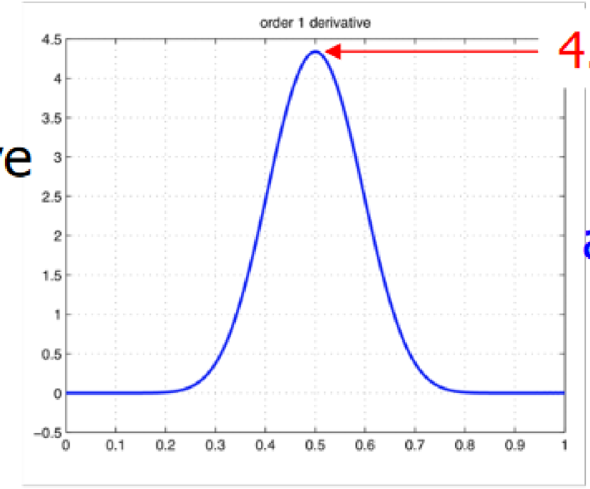
29th degree

no overshoot nor wandering

2.5



normalized first derivative (velocity in time)



4.5!!

peaking at midpoint

Interpolation using splines

- problem

  interpolate N knots, with continuity up to the second derivative

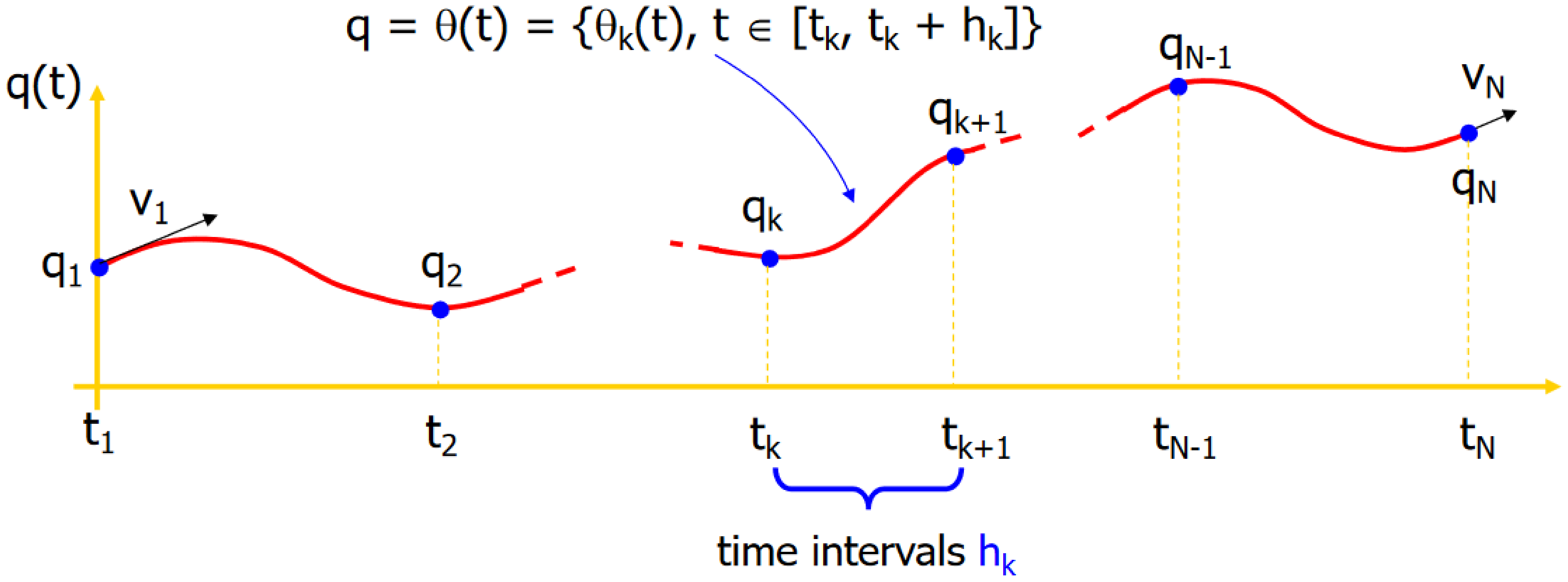
- solution

  spline: N-1 cubic polynomials, concatenated so as to pass through N knots and being continuous up to the second derivative at the N-2 internal knots

- 4(N-1) coefficients
- 4(N-1)-2 conditions, or
  - 2(N-1) of passage (for each cubic, in the two knots at its ends)
  - N-2 of continuity for first derivative (at the internal knots)
  - N-2 of continuity for second derivative (at the internal knots)

- 2 free parameters are still left over
  - can be used, e.g., to assign initial and final derivatives, v_1 and v_N
- presented next in terms of time t, but similar in terms of space λ
  - then: first derivative = velocity, second derivative = acceleration

Building a cubic spline



$$\theta_k(\tau) = a_{k0} + a_{k1}\,\tau + a_{k2}\,\tau^2 + a_{k3}\,\tau^3 \qquad \tau \in [0,\ h_k],\ \tau = t - t_k \quad (k = 1, \ldots, N-1)$$

continuity conditions for velocity and acceleration $\rightarrow$

$$\dot{\theta}_k(h_k) = \dot{\theta}_{k+1}(0)$$
$$\ddot{\theta}_k(h_k) = \ddot{\theta}_{k+1}(0) \qquad k = 1, \ldots, N-2$$

Properties of splines

- a spline (in space) is the solution with minimum curvature among all interpolating functions having continuous second derivative
- for cyclic tasks ($q_1 = q_N$), it is preferable to simply impose continuity of first and second derivatives (i.e., velocity and acceleration in time) at the first/last knot as "squaring" conditions
  - choosing $v_1 = v_N = v$ (for a given v) doesn't guarantee in general the continuity up to the second derivative (in time, of the acceleration)
  - in this way, the first = last knot will be handled as all other internal knots
- a spline is uniquely determined from the set of data $q_1,\ldots,q_N$, $h_1, \ldots, h_{N-1}, v_1, v_N$
- in time, the total motion occurs in $T = \Sigma_k\, h_k = t_N - t_1$
- the time intervals h_k can be chosen so as to minimize T (linear objective function) under (nonlinear) bounds on velocity and acceleration in $[0,T]$
- in time, the spline construction can be suitably modified when the acceleration is also assigned at the initial and final knots

A modification

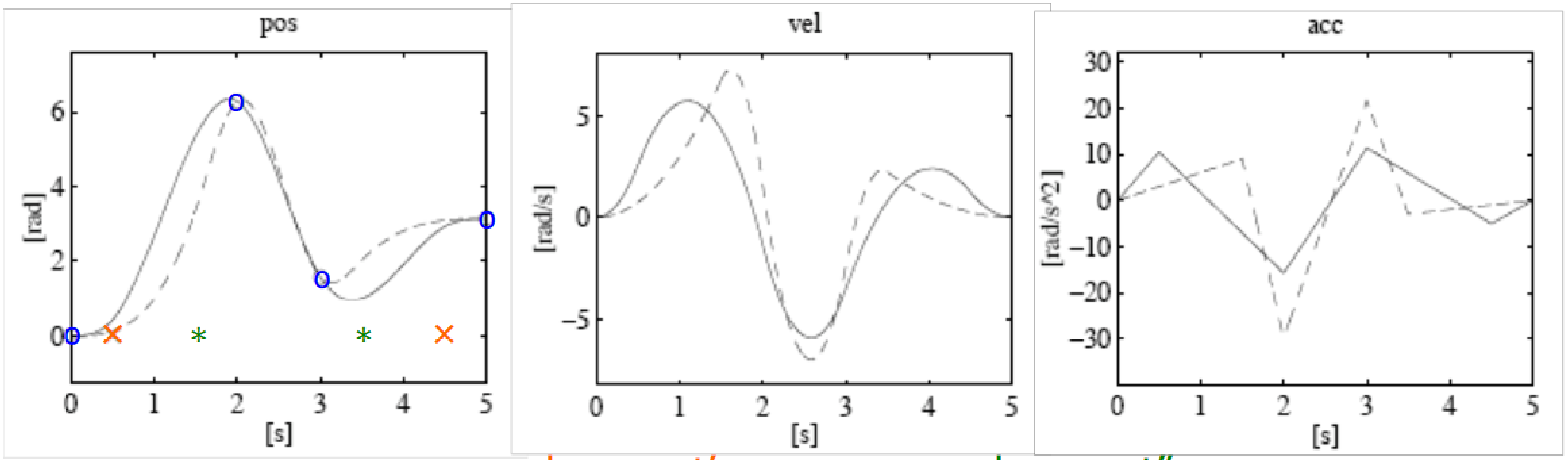
handling assigned initial and final accelerations

- two more parameters are needed in order to impose also the initial acceleration α_1 and final acceleration α_N
- two "fictitious knots" are inserted in the first and last original intervals, increasing the number of cubic polynomials from N-1 to N+1
- in these two knots only continuity conditions on position, velocity and acceleration are imposed

  $\Rightarrow$ two free parameters are left over (one in the first cubic and the other in the last cubic), which are used to satisfy the boundary conditions on acceleration
- depending on the (time) placement of the two additional knots, the resulting spline changes

A numerical example

- $N = 4$ knots (3 cubic polynomials)
  - joint values $q_1 = 0$, $q_2 = 2\pi$, $q_3 = \pi/2$, $q_4 = \pi$
  - at $t_1 = 0$, $t_2 = 2$, $t_3 = 3$, $t_4 = 5$ (thus, $h_1 = 2$, $h_2 = 1$, $h_3 = 2$)
  - boundary velocities $v_1 = v_4 = 0$
- 2 added knots to impose accelerations at both ends (5 cubic polynomials)
  - boundary accelerations $\alpha_1 = \alpha_4 = 0$
  - two placements: at $t_1' = 0.5$ and $t_4' = 4.5$ (×), or $t_1'' = 1.5$ and $t_4'' = 3.5$ (*)



——— = placement′ - - - - - = placement″

The end!