



Robotics 1

Trajectory planning

Prof. Alessandro De Luca

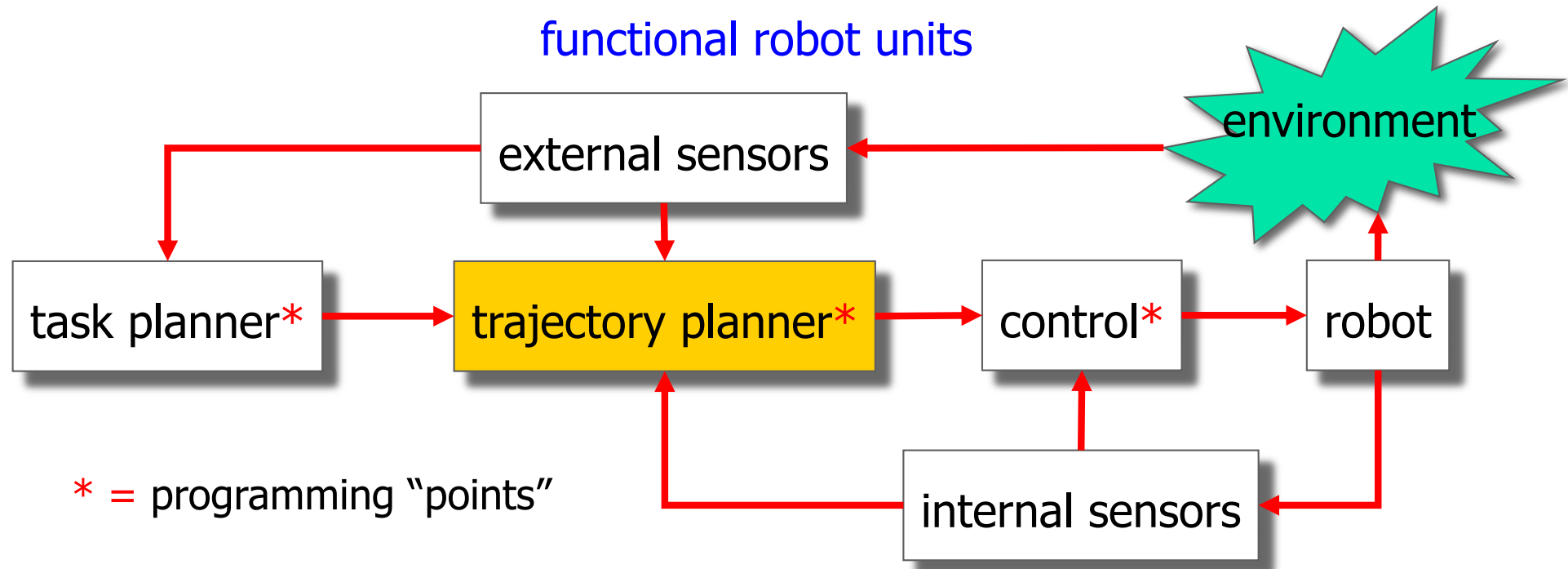
DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI



SAPIENZA
UNIVERSITÀ DI ROMA



Trajectory planner interfaces



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**

Trajectory definition

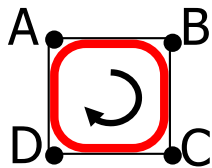
a standard procedure for industrial robots



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

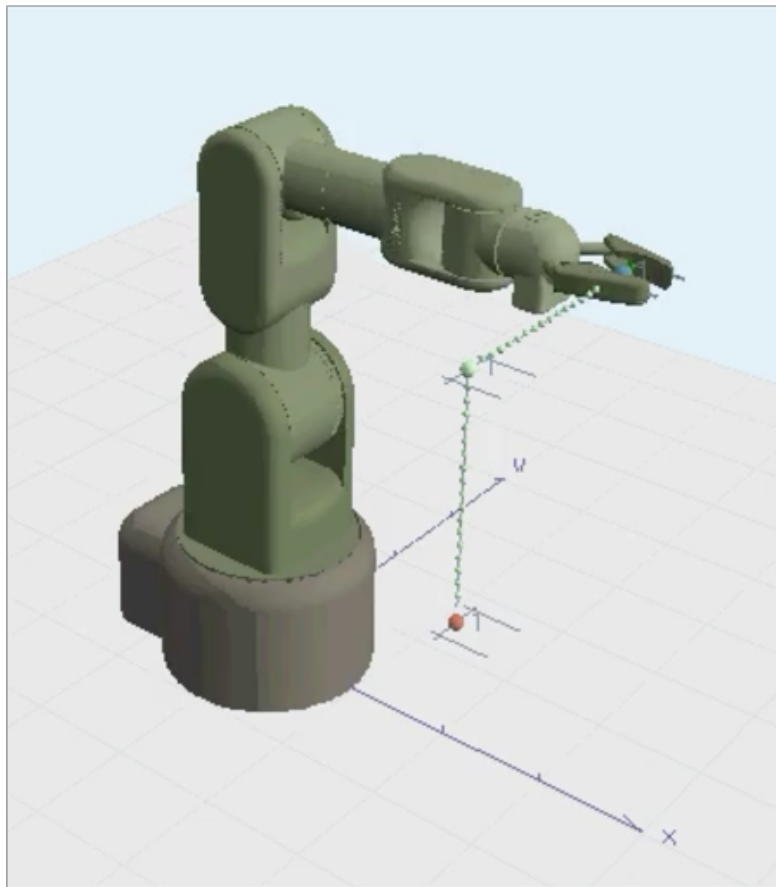
- semi-manual programming (as in “first generation” robot languages)
- limited visualization of motion



a mathematical formalization of trajectories is useful/needed

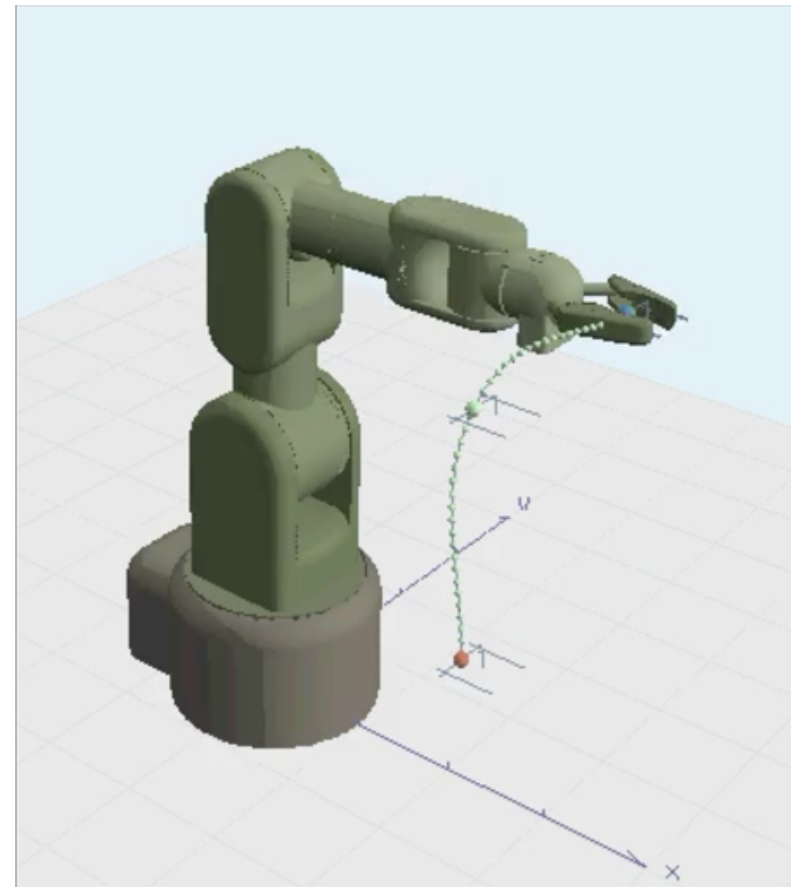
Some typical trajectories

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



video

Straight lines as Cartesian path

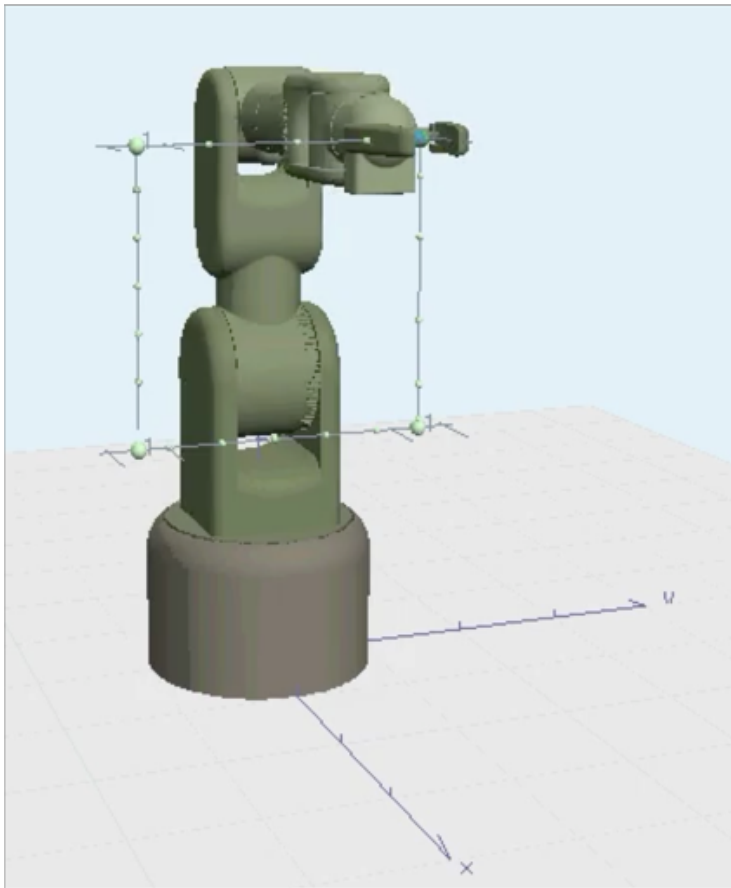


video

Interpolation with Bezier curves

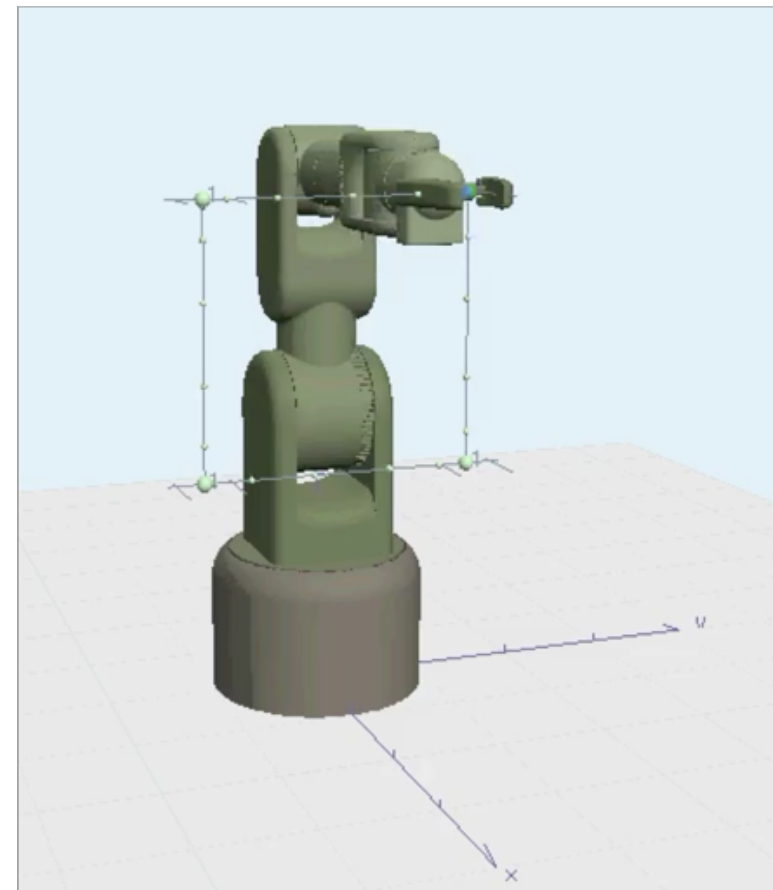
Some typical trajectories

- Timing laws: Cartesian path with (dis-)continuous tangent



video

Square path at constant speed

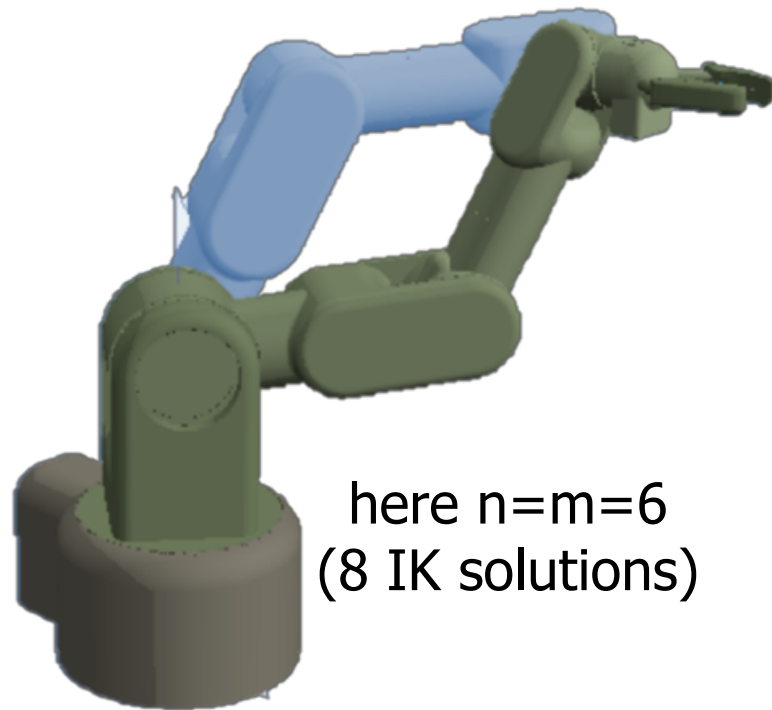


video

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**



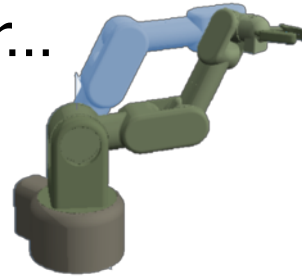
here $n=m=6$
(8 IK solutions)

- **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 ($m < n$)

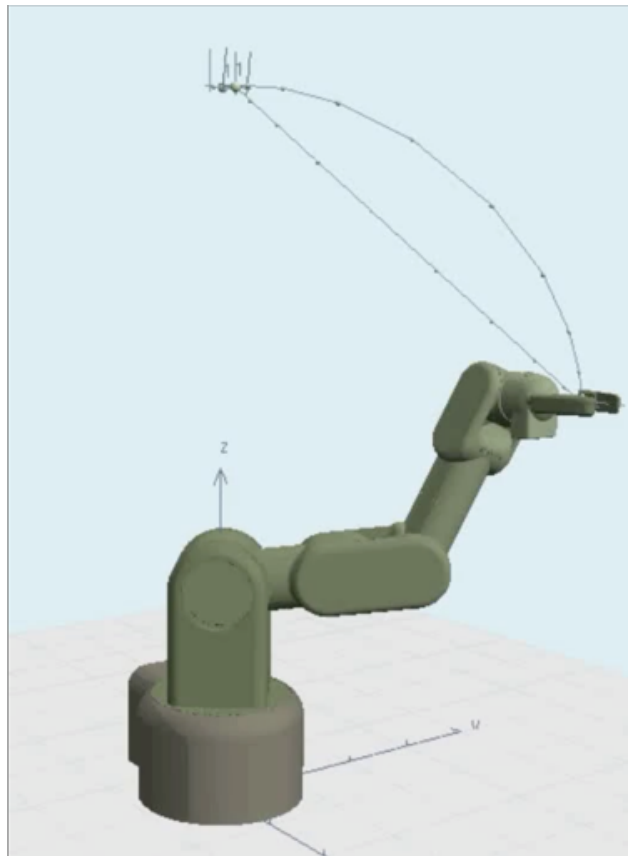
for “simple” manipulators (e.g., all industrial robots) and $m=n$, the execution of these tasks will require the **passage through a singular configuration**

Joint and Cartesian trajectories

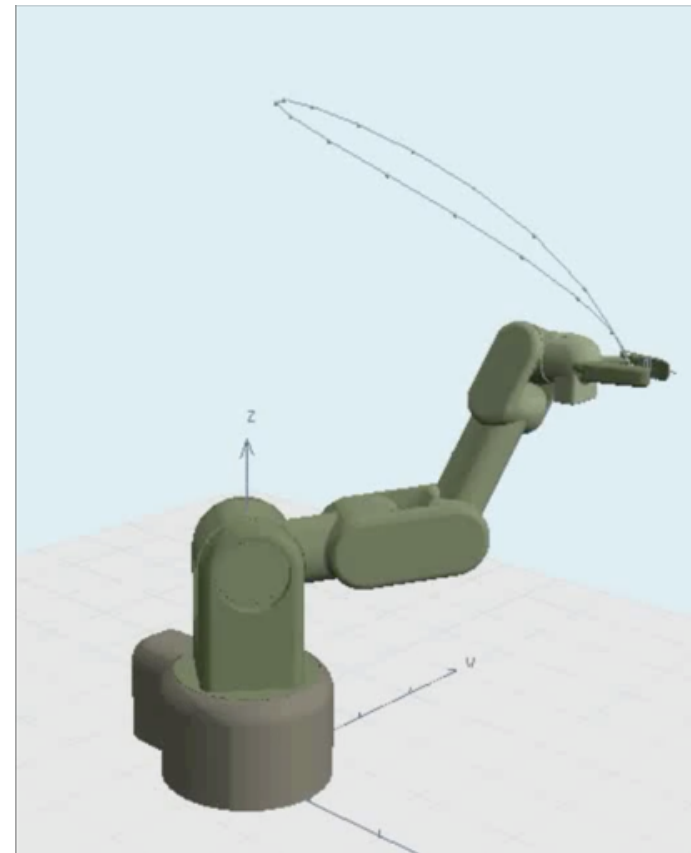
- a reconfiguration task (or... passing through singularity)



video



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



video

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



From task to trajectory

TRAJECTORY

||

GEOMETRIC PATH

+

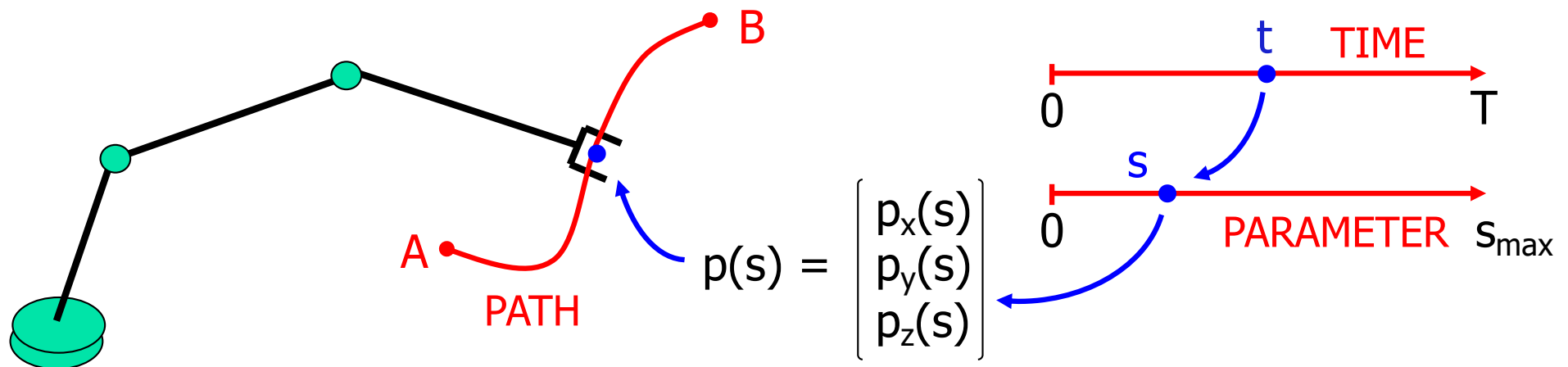
TIMING LAW

{ of motion $p_d(t)$
of interaction $F_d(t)$

parameterized by s : $p=p(s)$
(e.g., s is the arc length)

describes the time evolution of $s=s(t)$

} $p(s(t))$

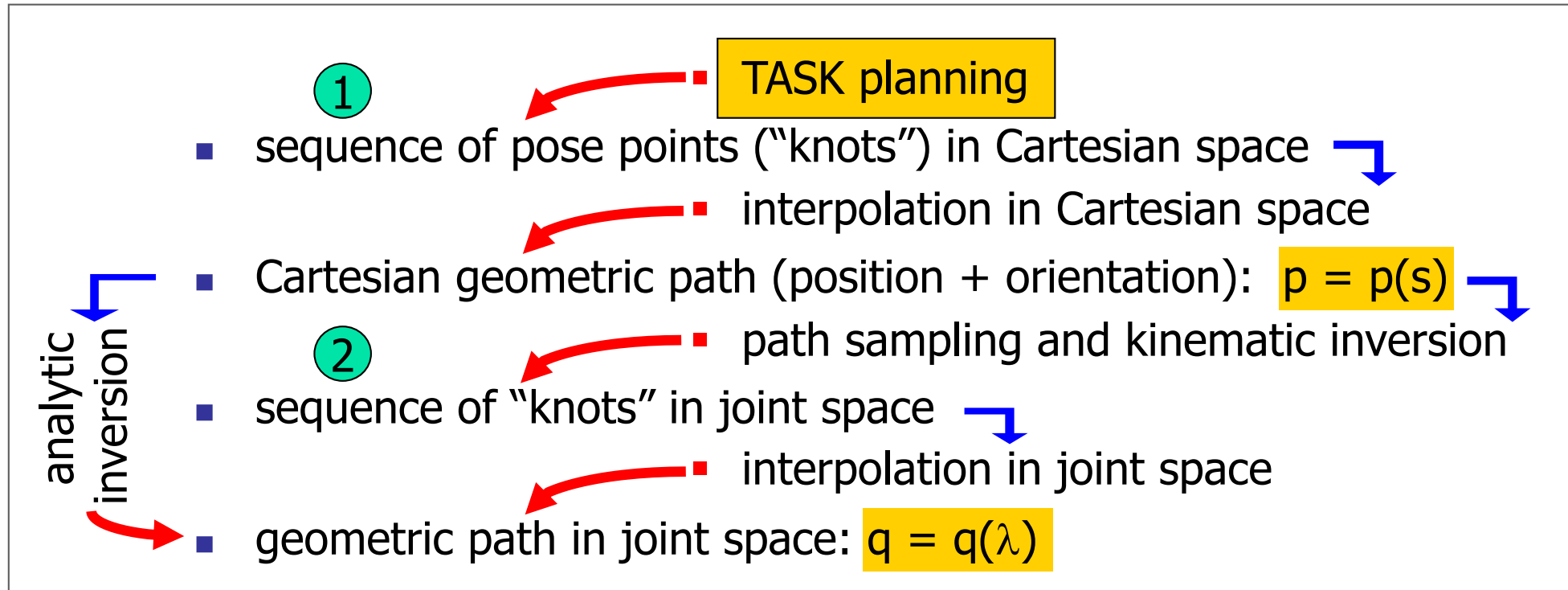


example: TASK planner provides A, B
TRAJECTORY planner generates $p(t)$



Trajectory planning

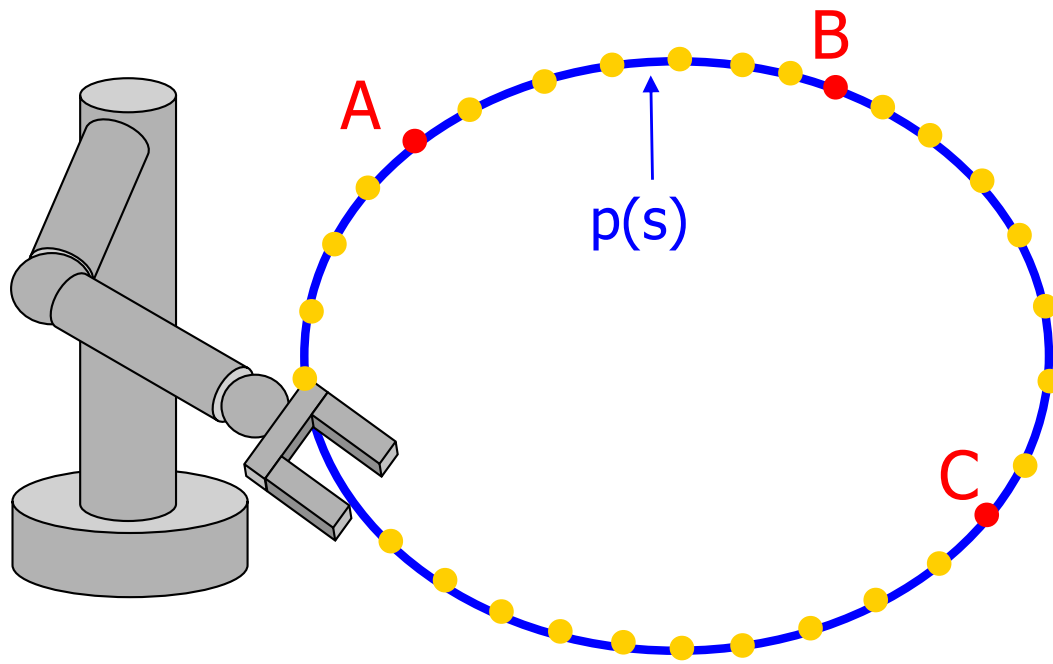
operative sequence



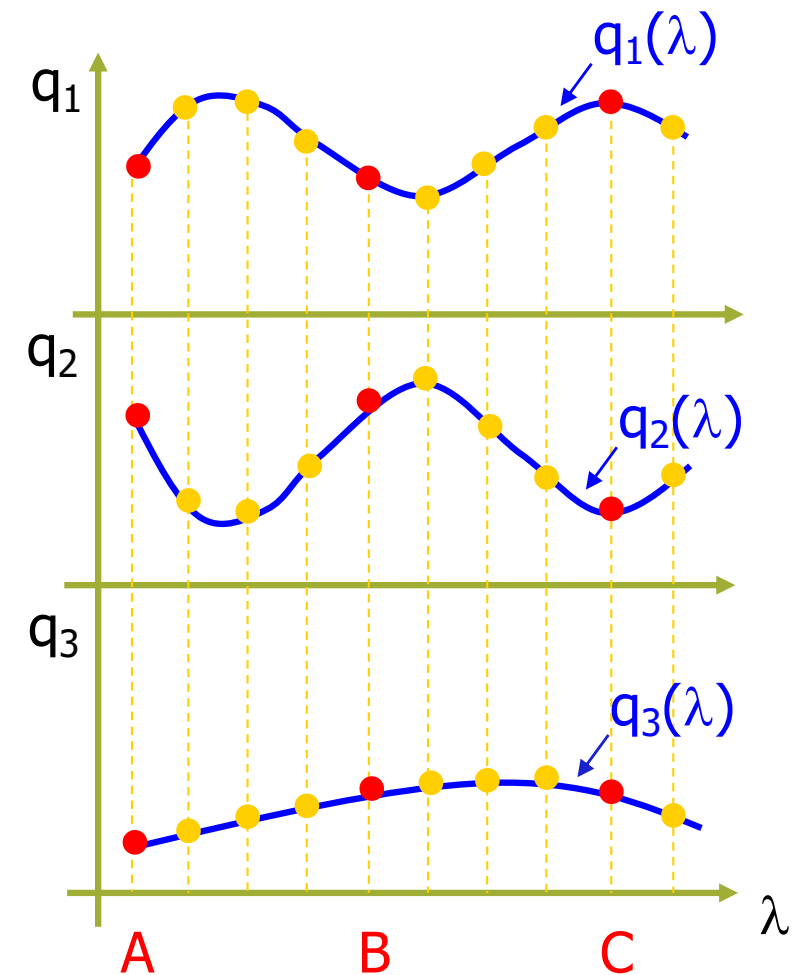
additional issues to be considered in the planning process

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

Example



Cartesian space



joint space



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**



Path and timing law

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

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

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

- 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

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



Cartesian vs. joint trajectory planning

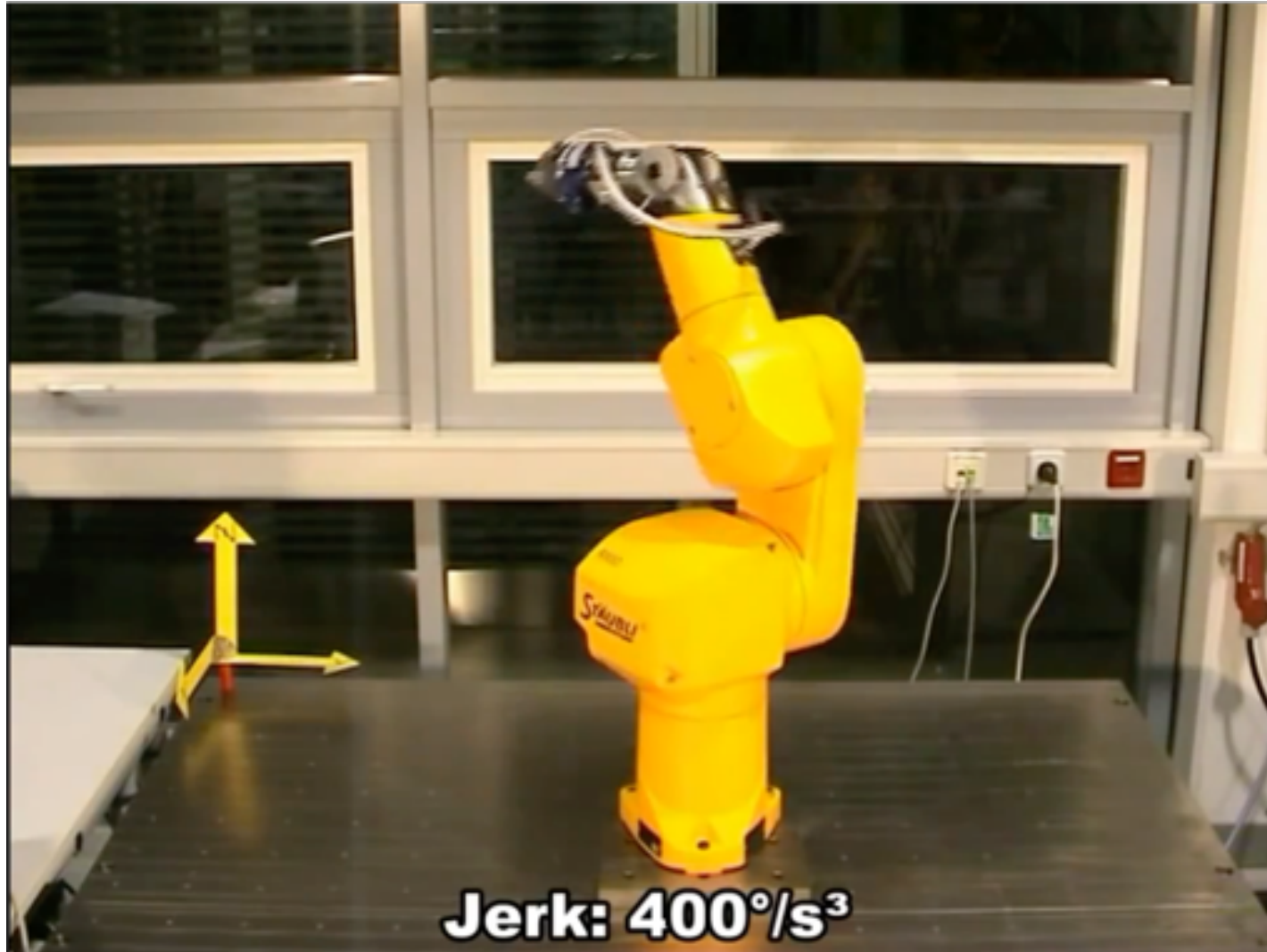
- planning in **Cartesian space**
 - allows a more direct visualization of the generated path
 - obstacle avoidance, lack of “wandering”
- planning in **joint space**
 - does not need on-line kinematic inversion
- issues in kinematic inversion
 - \dot{q} e \ddot{q} (or higher-order derivatives) may also 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



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



video



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)!



Cubic polynomial in space

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

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

$$\Delta q = q_1 - q_0$$

$$\lambda \in [0,1]$$

4 coefficients \rightarrow “doubly normalized” polynomial $q_N(\lambda)$

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

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

$$q_N'(0) = dq_N/d\lambda|_{\lambda=0} = c = v_0/\Delta q \quad q_N'(1) = dq_N/d\lambda|_{\lambda=1} = 3a + 2b + c = v_1/\Delta q$$

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

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

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



Cubic polynomial in time

$$\boxed{q(0) = q_{in}} \quad \boxed{q(T) = q_{fin}} \quad \boxed{\dot{q}(0) = v_{in}} \quad \boxed{\dot{q}(T) = v_{fin}} \quad \leftarrow 4 \text{ conditions}$$

$$q(\tau) = q_{in} + \Delta q [a \tau^3 + b \tau^2 + c \tau + d]$$

$$\Delta q = q_{fin} - q_{in}$$

$$\tau = t/T, \tau \in [0,1]$$

4 coefficients \rightarrow "doubly normalized" polynomial $q_N(\tau)$

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

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

$$q_N'(0) = dq_N/d\tau|_{\tau=0} = c = v_{in}T/\Delta q \quad q_N'(1) = dq_N/d\tau|_{\tau=1} = 3a + 2b + c = v_{fin}T/\Delta q$$

special case: $v_{in} = v_{fin} = 0$ (rest-to-rest)

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

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



Quintic polynomial

$$q(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f$$

6 coefficients

$$\tau \in [0, 1]$$

allows to satisfy 6 conditions, for example (in normalized time $\tau = t/T$)

$$q(0) = q_0$$

$$q(1) = q_1$$

$$q'(0) = v_0T$$

$$q'(1) = v_1T$$

$$q''(0) = a_0T^2$$

$$q''(1) = a_1T^2$$

$$q(\tau) = (1 - \tau)^3[q_0 + (3q_0 + v_0T)\tau + (a_0T^2 + 6v_0T + 12q_0)\tau^2/2] \\ + \tau^3[q_1 + (3q_1 - v_1T)(1 - \tau) + (a_1T^2 - 6v_1T + 12q_1)(1 - \tau)^2/2]$$

special case: $v_0 = v_1 = a_0 = a_1 = 0$

$$q(\tau) = q_0 + \Delta q[6\tau^5 - 15\tau^4 + 10\tau^3]$$

$$\Delta q = q_1 - q_0$$



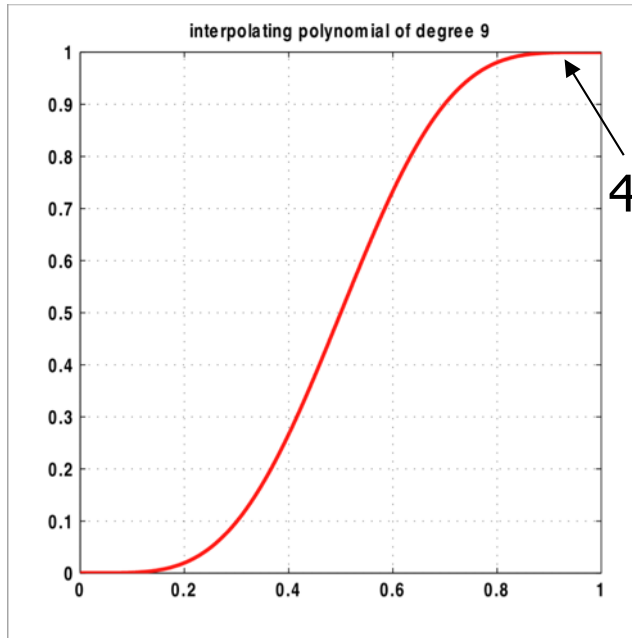
Higher-order polynomials

- a suitable solution class for satisfying **symmetric** boundary conditions (in a PTP motion) that **impose zero** values on higher-order derivatives
 - the interpolating polynomial is always of **odd** degree
 - the coefficients of such (**doubly normalized**) polynomial are always **integers, alternate in sign**, sum up to unity, and are zero for all terms up to the power = $(\text{degree}-1)/2$
- in all other cases (e.g., for interpolating a large number N of points), their use is **not** recommended
 - N -th order polynomials have $N-1$ maximum and minimum points
 - oscillations arise out of the interpolation points (**wandering**)



Numerical examples

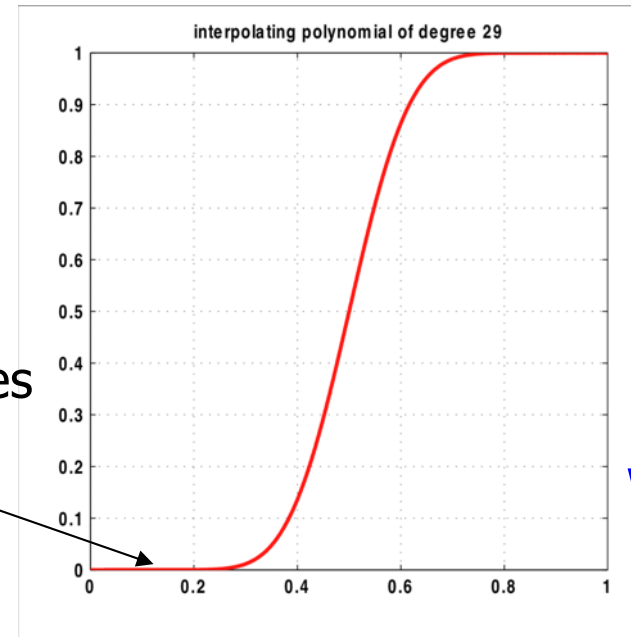
9th
degree



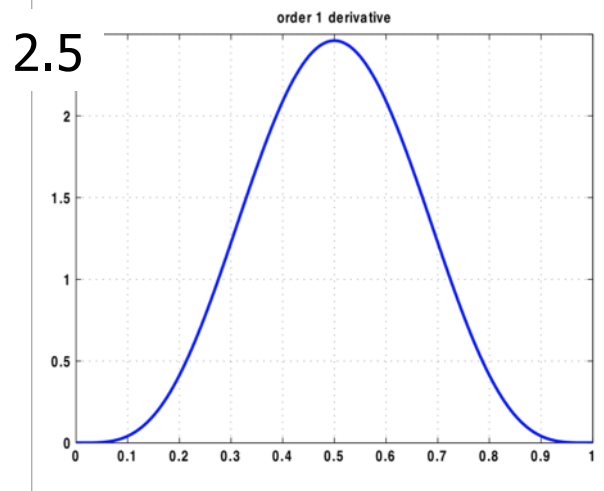
4 derivatives
are zero

14 derivatives
are zero!

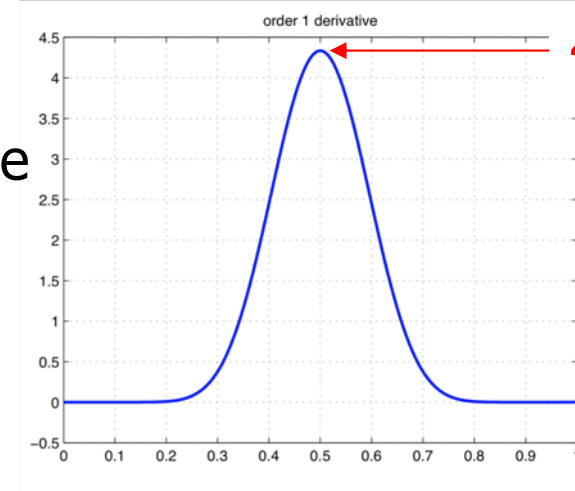
29th
degree



no
overshoot
nor
wandering



normalized
first derivative
(velocity
in time)

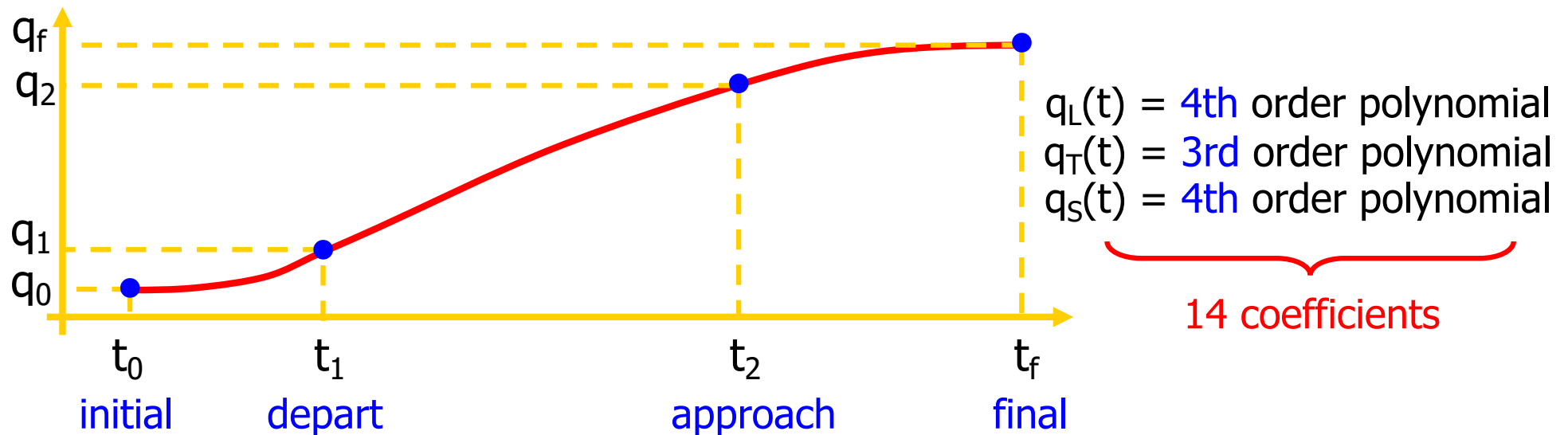


peaking
at midpoint



4-3-4 polynomials

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



boundary conditions

$$\begin{array}{l}
 q(t_0) = q_0 \quad q(t_1^-) = q(t_1^+) = q_1 \quad q(t_2^-) = q(t_2^+) = q_2 \quad q(t_f) = q_f \quad \left. \vphantom{\begin{array}{l} q(t_0) = q_0 \\ q(t_1^-) = q(t_1^+) = q_1 \\ q(t_2^-) = q(t_2^+) = q_2 \\ q(t_f) = q_f \end{array}} \right\} 6 \text{ passages} \\
 \dot{q}(t_0) = \dot{q}(t_f) = 0 \quad \ddot{q}(t_0) = \ddot{q}(t_f) = 0 \quad \left. \vphantom{\begin{array}{l} \dot{q}(t_0) = \dot{q}(t_f) = 0 \\ \ddot{q}(t_0) = \ddot{q}(t_f) = 0 \end{array}} \right\} 4 \text{ initial/final} \\
 \dot{q}(t_i^-) = \dot{q}(t_i^+) \quad \ddot{q}(t_i^-) = \ddot{q}(t_i^+) \quad i = 1, 2 \quad \left. \vphantom{\begin{array}{l} \dot{q}(t_i^-) = \dot{q}(t_i^+) \\ \ddot{q}(t_i^-) = \ddot{q}(t_i^+) \end{array}} \right\} \text{velocity/acceleration} \\
 \hspace{25em} 4 \text{ continuity}
 \end{array}$$

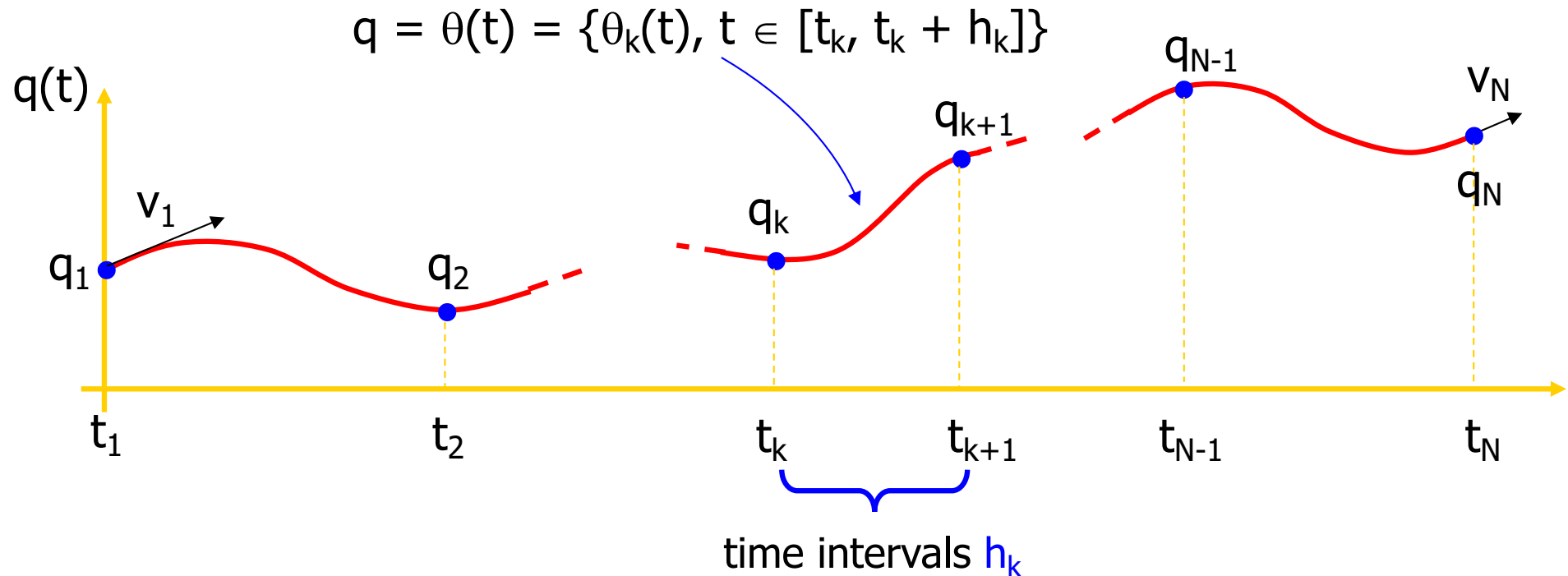


Interpolation using splines

- **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



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

continuity conditions
for velocity and acceleration



$$\begin{aligned} \dot{\theta}_k(h_k) &= \dot{\theta}_{k+1}(0) \\ \ddot{\theta}_k(h_k) &= \ddot{\theta}_{k+1}(0) \end{aligned} \quad k = 1, \dots, N-2$$



An efficient algorithm

1. if all **velocities** v_k at **internal knots** were known, then each cubic in the spline would be uniquely determined by

$$\begin{aligned} \theta_k(0) &= q_k = a_{k0} \\ \dot{\theta}_k(0) &= v_k = a_{k1} \end{aligned} \quad \begin{pmatrix} h_k^2 & h_k^3 \\ 2h_k & 3h_k^2 \end{pmatrix} \begin{pmatrix} a_{k2} \\ a_{k3} \end{pmatrix} = \begin{pmatrix} q_{k+1} - q_k - v_k h_k \\ v_{k+1} - v_k \end{pmatrix} \quad (1)$$

2. impose the **continuity for accelerations** (N-2 conditions)

$$\ddot{\theta}_k(h_k) = 2 a_{k2} + 6 a_{k3} h_k = \ddot{\theta}_{k+1}(0) = 2 a_{k+1,2}$$

1. expressing the coefficients a_{k2} , a_{k3} , $a_{k+1,2}$ in terms of the **still unknown** knot velocities (see step 1.) yields a linear system of equations that is always (easily) solvable

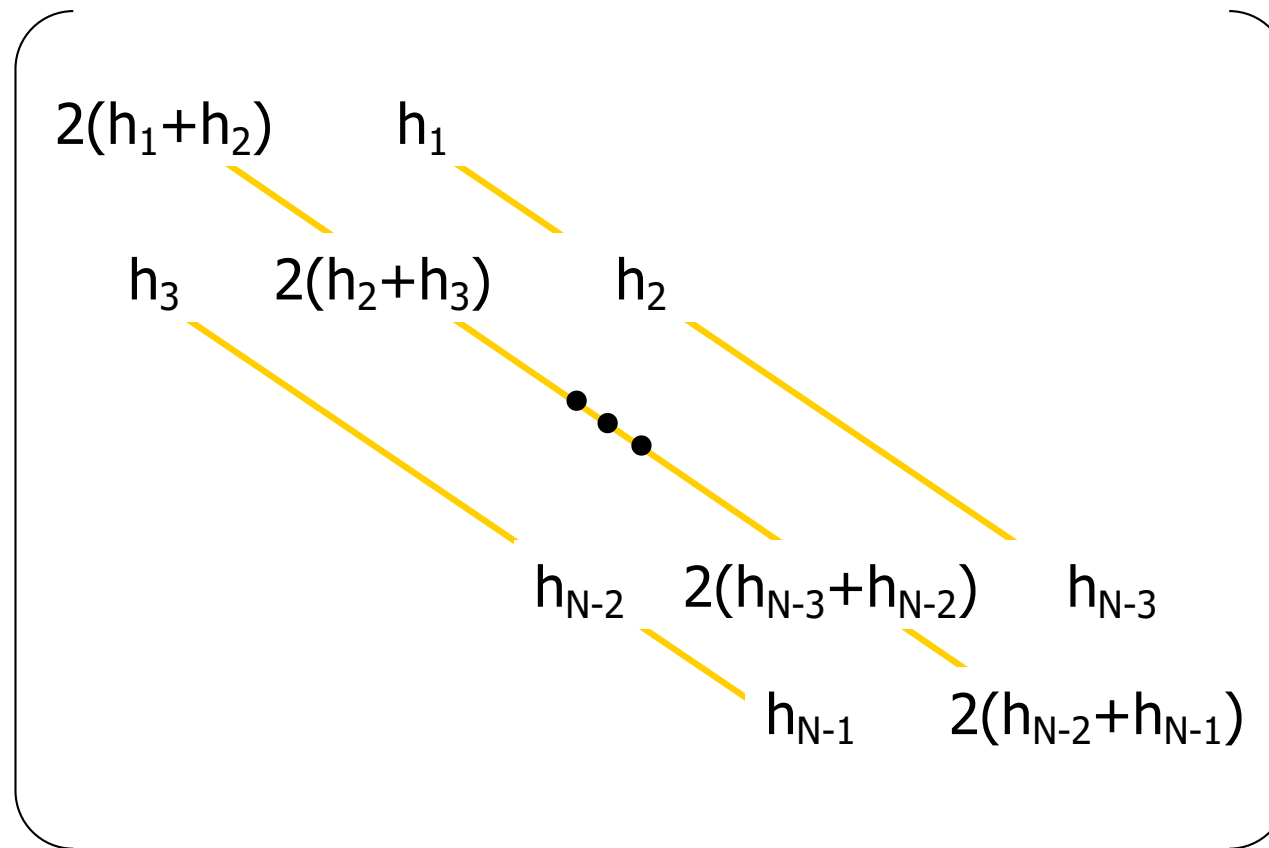
$$\begin{pmatrix} \text{tri-diagonal matrix} \\ A(h) \end{pmatrix} \begin{pmatrix} v_2 \\ v_3 \\ \vdots \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} \text{known vector} \\ b(h, q, v_1, v_N) \end{pmatrix}$$

tri-diagonal matrix always invertible unknown known vector

to be substituted then back in (1)



Structure of $A(h)$



diagonally dominant matrix (for $h_k > 0$)
[the **same** matrix for all joints]



Structure of $b(h, q, v_1, v_N)$

$$\left[\begin{array}{c} \frac{3}{h_1 h_2} [h_1^2(q_3 - q_2) + h_2^2(q_2 - q_1)] - h_2 v_1 \\ \frac{3}{h_2 h_3} [h_2^2(q_4 - q_3) + h_3^2(q_3 - q_2)] \\ \vdots \\ \frac{3}{h_{N-3} h_{N-2}} [h_{N-3}^2(q_{N-1} - q_{N-2}) + h_{N-2}^2(q_{N-2} - q_{N-3})] \\ \frac{3}{h_{N-2} h_{N-1}} [h_{N-2}^2(q_N - q_{N-1}) + h_{N-1}^2(q_{N-1} - q_{N-2})] - h_{N-2} v_N \end{array} \right]$$



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, \dots, q_N , h_1, \dots, h_{N-1} , v_1 , v_N
- in **time**, the total motion occurs in $T = \sum_k h_k = t_N - t_1$
- 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

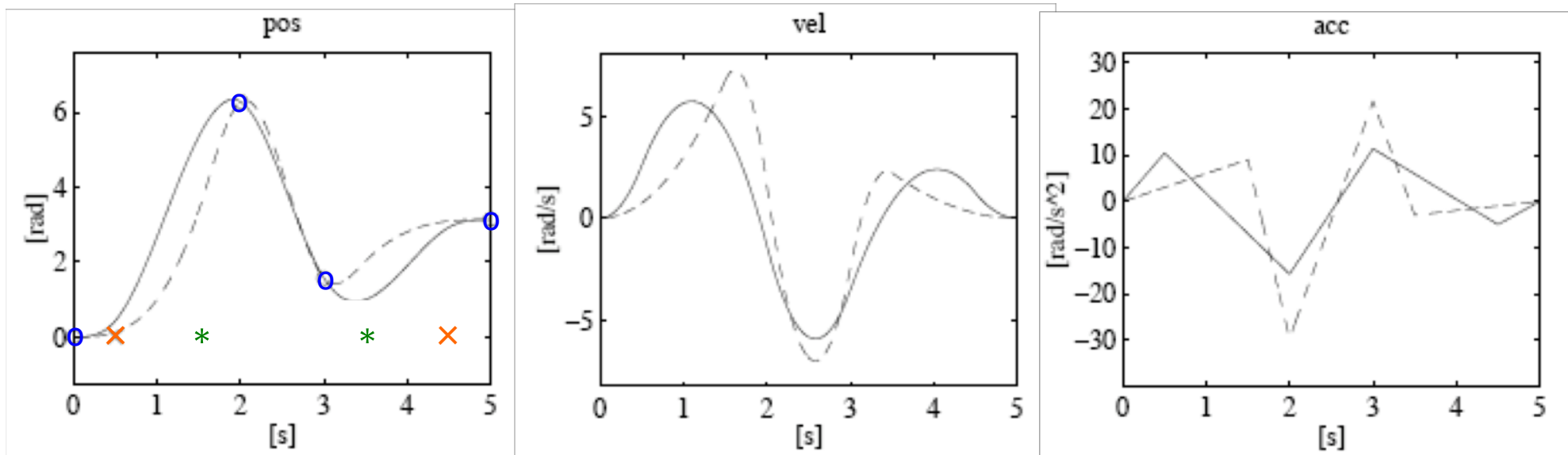
handling assigned initial and final accelerations

- 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''