Robotics 1

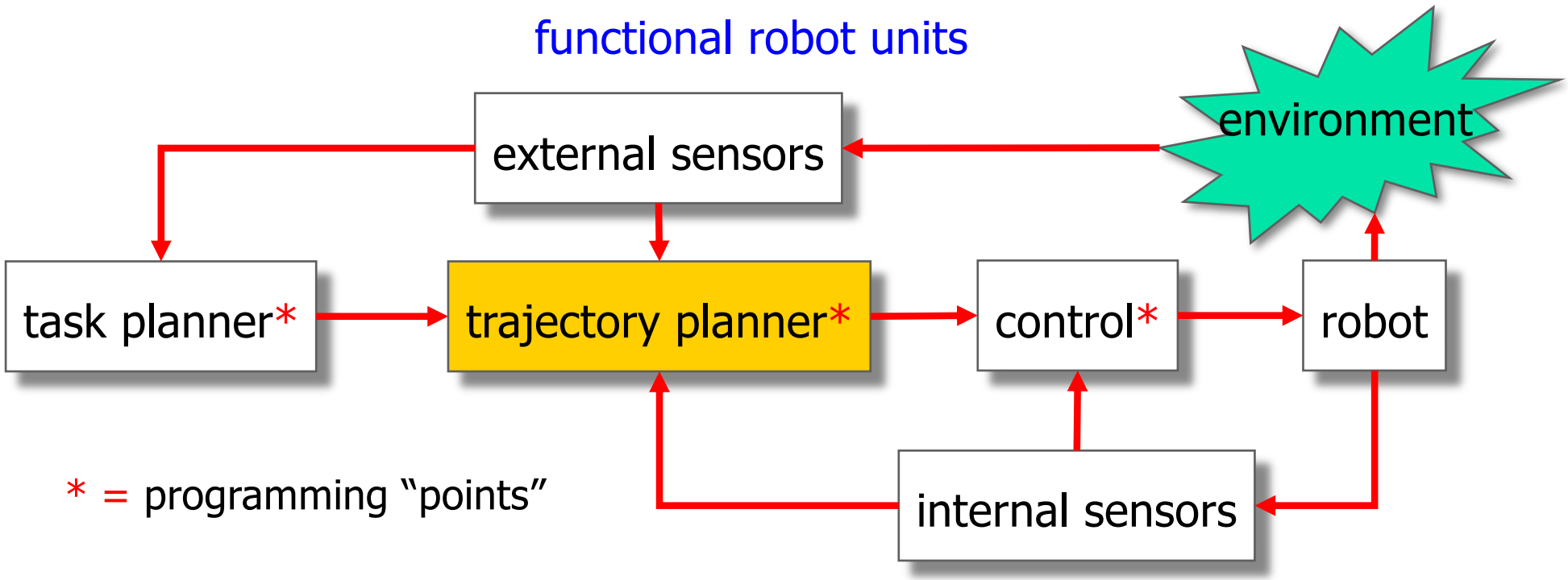
Trajectory planning

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)

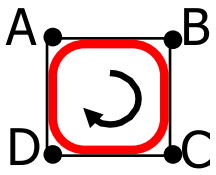
TRAJECTORY PLANNER

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

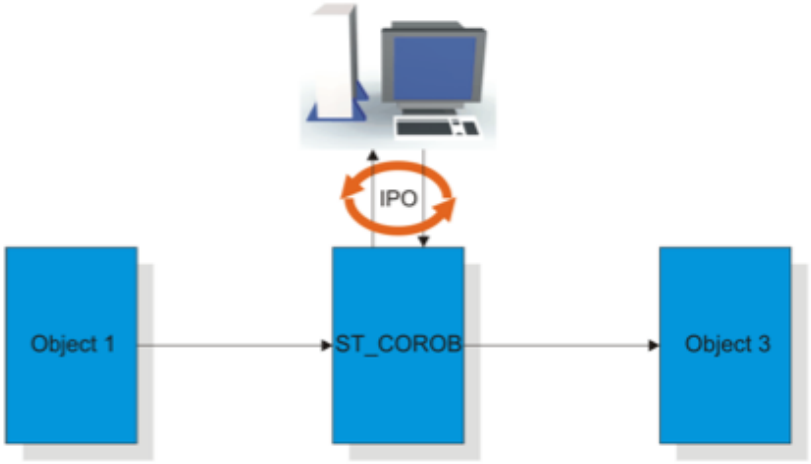
a) over-fly 

b) sensor-driven STOP

c) circular path through 3 points

KUKA user interfaces

- Teach pendant
- KRL programming
- Ethernet RSI XML (12ms)



- Fast Research Interface (1ms)

Fig. 4-1: Front view of KCP

1	Mode selector switch	10	Numeric keypad
2	Drives ON	11	Softkeys
3	Drives OFF / SSB GUI	12	Start backwards key
4	EMERGENCY STOP button	13	Start key
5	Space Mouse	14	STOP key
6	Right-hand status keys	15	Window selection key
7	Enter key	16	ESC key
8	Arrow keys	17	Left-hand status keys
9	Keypad	18	Menu keys

KRL language

- basic **instruction** set:

Variables and declarations	
DECL	(>>> 10.4.1 "DECL" page 138)
ENUM	(>>> 10.4.2 "ENUM" page 140)
IMPORT ... IS	(>>> 10.4.3 "IMPORT ... IS" page 141)
STRUC	(>>> 10.4.4 "STRUC" page 141)

Motion programming	
CIRC	(>>> 10.5.1 "CIRC" page 143)
CIRC_REL	(>>> 10.5.2 "CIRC_REL" page 144)
LIN	(>>> 10.5.3 "LIN" page 146)
LIN_REL	(>>> 10.5.4 "LIN_REL" page 146)
PTP	(>>> 10.5.5 "PTP" page 148)
PTP_REL	(>>> 10.5.6 "PTP_REL" page 148)

Program execution control	
CONTINUE	(>>> 10.6.1 "CONTINUE" page 150)
EXIT	(>>> 10.6.2 "EXIT" page 150)
FOR ... TO ... ENDFOR	(>>> 10.6.3 "FOR ... TO ... ENDFOR" page 150)
GOTO	(>>> 10.6.4 "GOTO" page 151)
HALT	(>>> 10.6.5 "HALT" page 152)
IF ... THEN ... ENDIF	(>>> 10.6.6 "IF ... THEN ... ENDIF" page 152)
LOOP ... ENDLOOP	(>>> 10.6.7 "LOOP ... ENDLOOP" page 153)
REPEAT ... UNTIL	(>>> 10.6.8 "REPEAT ... UNTIL" page 153)
SWITCH ... CASE ... ENDSWITCH	(>>> 10.6.9 "SWITCH ... CASE ... ENDSWITCH" page 154)
WAIT ... FOR	(>>> 10.6.10 "WAIT FOR" page 155)
WAIT ... SEC	(>>> 10.6.11 "WAIT SEC" page 156)
WHILE ... ENDWHILE	(>>> 10.6.12 "WHILE ... ENDWHILE" page 156)

Inputs/outputs	
ANIN	(>>> 10.7.1 "ANIN" page 157)
ANOUT	(>>> 10.7.2 "ANOUT" page 158)
DIGIN	(>>> 10.7.3 "DIGIN" page 159)
PULSE	(>>> 10.7.4 "PULSE" page 160)
SIGNAL	(>>> 10.7.5 "SIGNAL" page 164)

Subprograms and functions	
RETURN	(>>> 10.8.1 "RETURN" page 165)

Interrupt programming	
BRAKE	(>>> 10.9.1 "BRAKE" page 166)
INTERRUPT	(>>> 10.9.2 "INTERRUPT" page 166)
INTERRUPT ... DECL ... WHEN ... DO	(>>> 10.9.3 "INTERRUPT ... DECL ... WHEN ... DO" page 167)
RESUME	(>>> 10.9.4 "RESUME" page 169)

Path-related switching actions (=Trigger)	
TRIGGER WHEN DISTANCE	(>>> 10.10.1 "TRIGGER WHEN DISTANCE" page 170)
TRIGGER WHEN PATH	(>>> 10.10.2 "TRIGGER WHEN PATH" page 173)

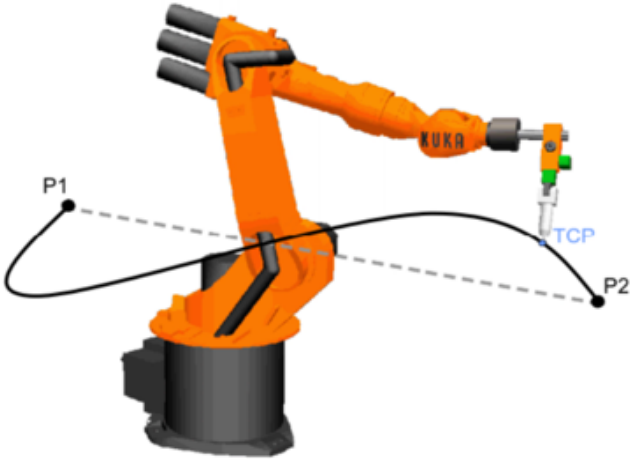
Communication
(>>> 10.11 "Communication" page 176)

System functions	
VARSTATE()	(>>> 10.12.1 "VARSTATE()" page 176)

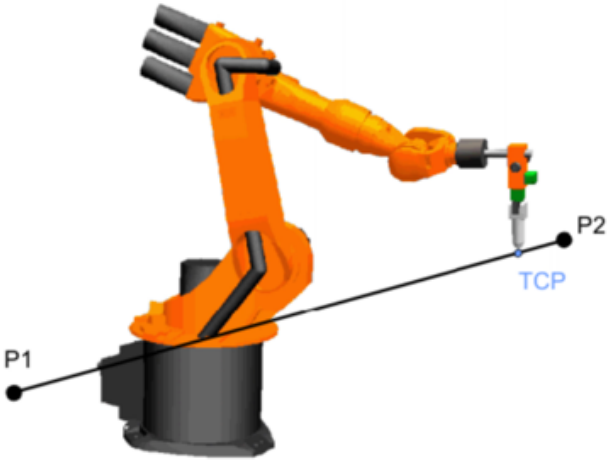
- basic **data** set: frames, vectors + DECLaration

KRL language

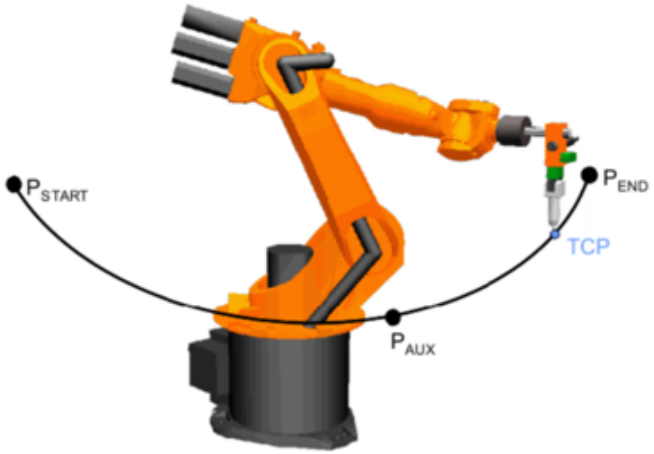
- typical motion primitives



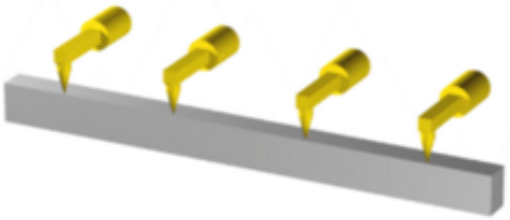
PTP motion
(point-to-point, linear
in joint space)



LIN motion
(linear in
Cartesian space)

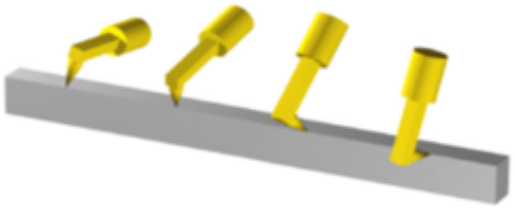


CIRC motion
(circular in
Cartesian space)



CONST orientation

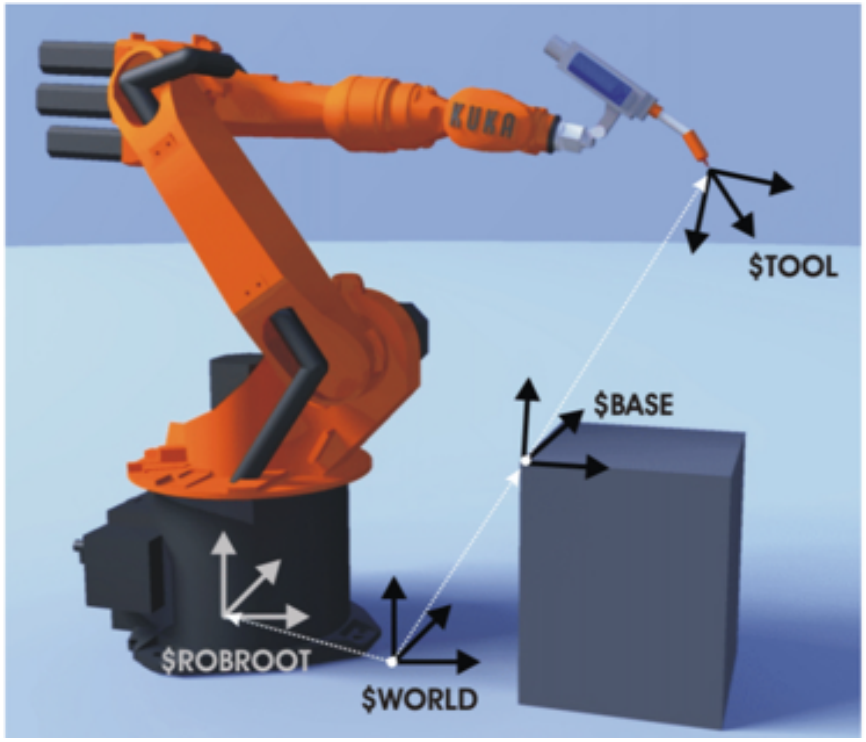
end-effector
orientation



PTP motion
(linear in RPY angles)

KRL language

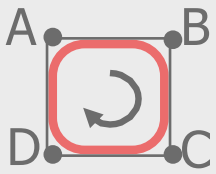
- multiple coordinate frames (in Cartesian space) and jogging of robot joints



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

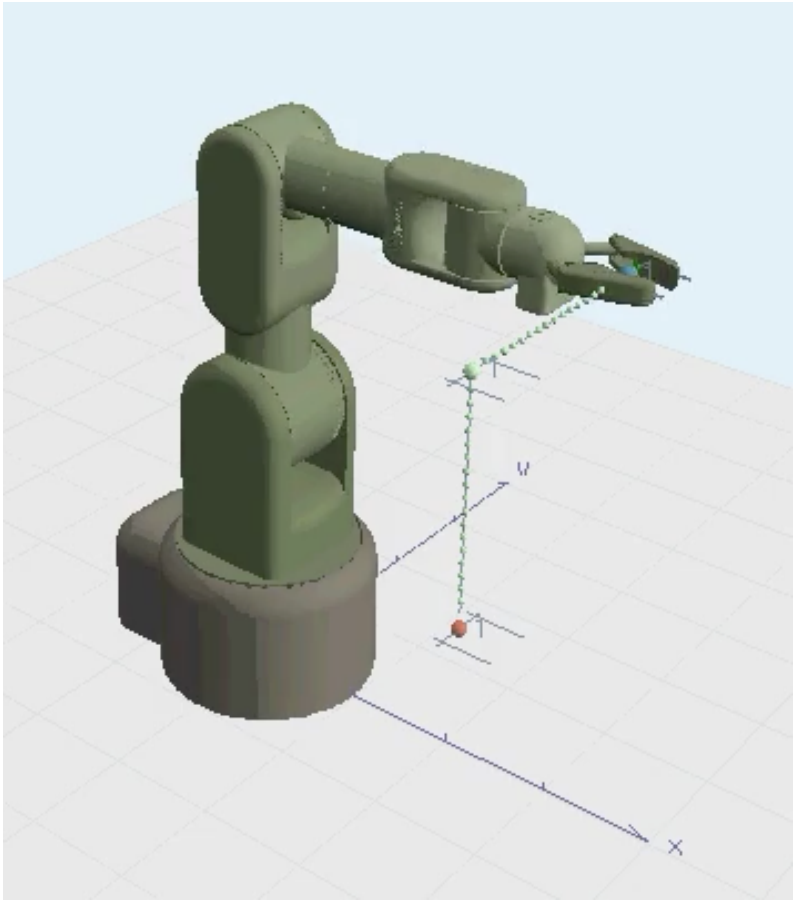
- semi-manual programming (as in "first generation" robot languages)
- limited visualization of motion

➡ a mathematical formalization of trajectories is useful/needed

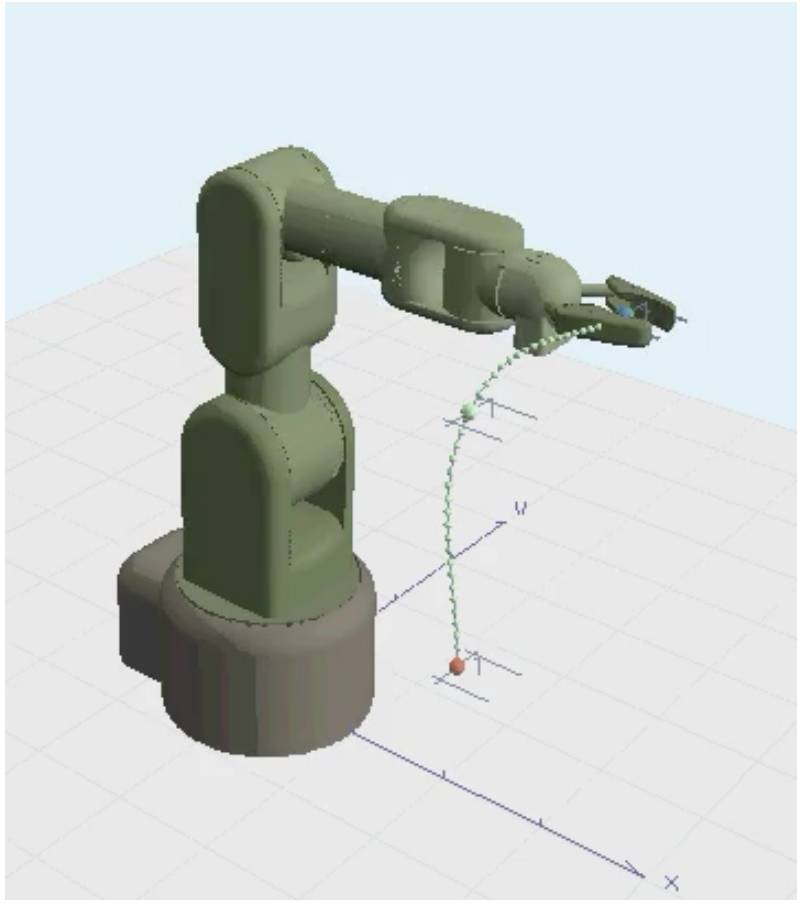
Some typical trajectories

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

video



Straight lines as Cartesian path



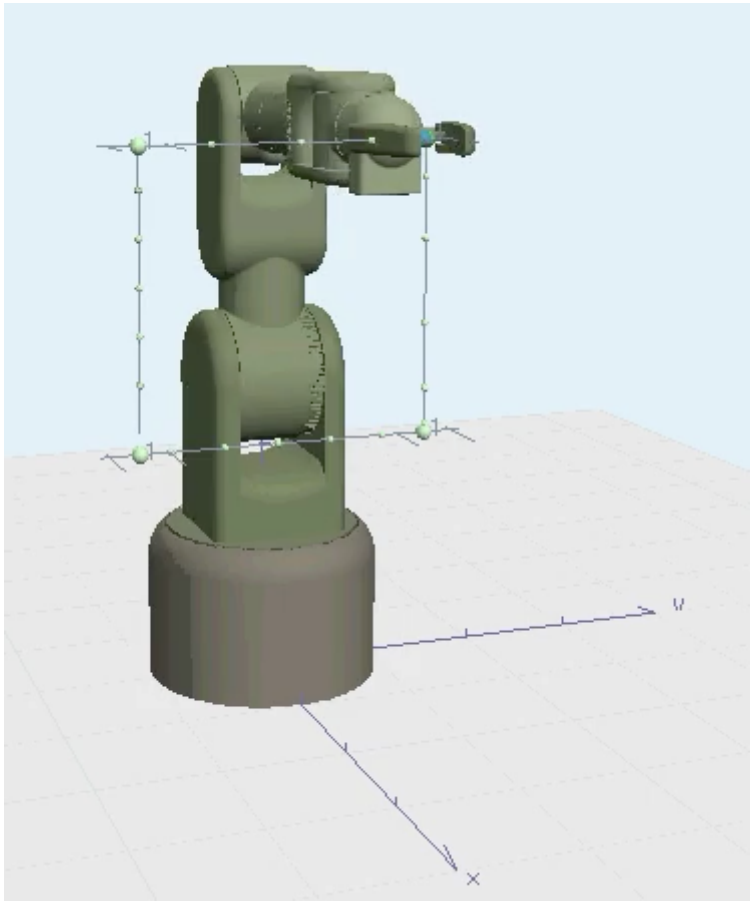
Interpolation with Bezier curves

video

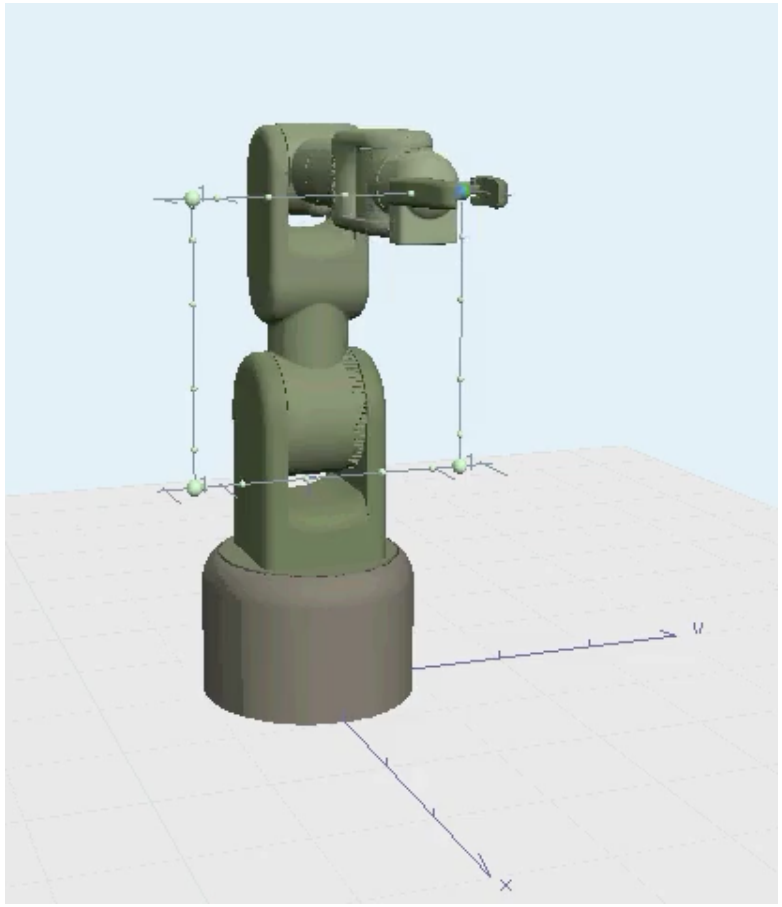
Some typical trajectories

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

video



Square path at constant speed

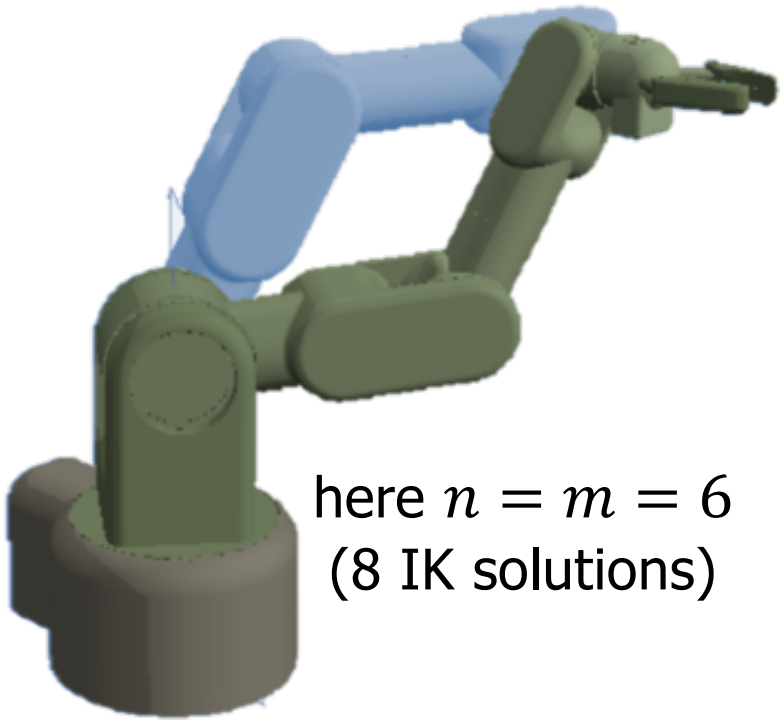


Square path with
trapezoidal speed profile

video

Joint and Cartesian trajectories

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

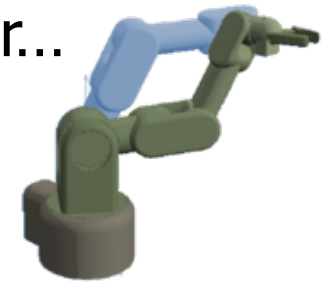


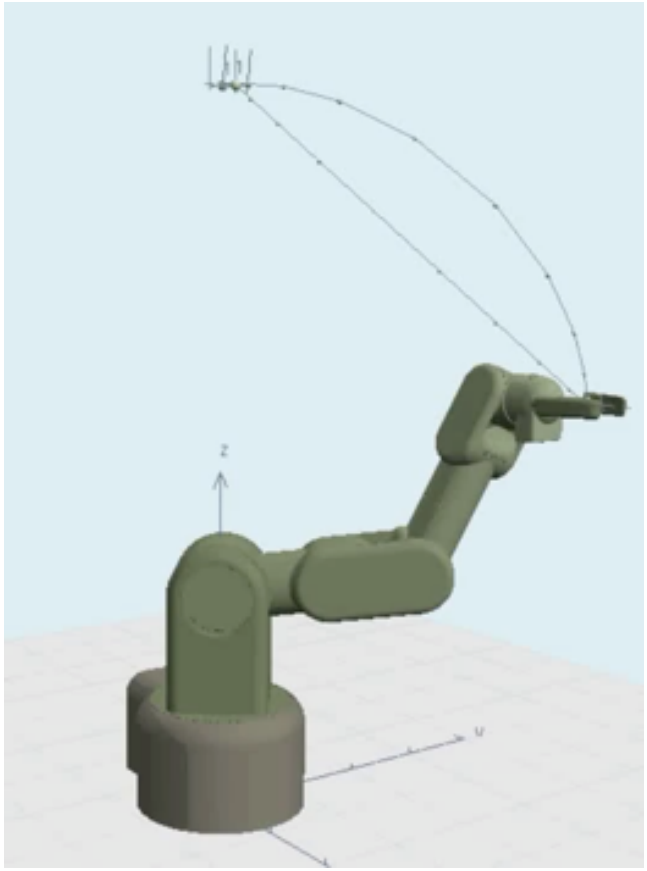
here $n = m = 6$
(8 IK solutions)

- **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 there is (task) redundancy ($m < n$)
  - if the robot starts in a singularity

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

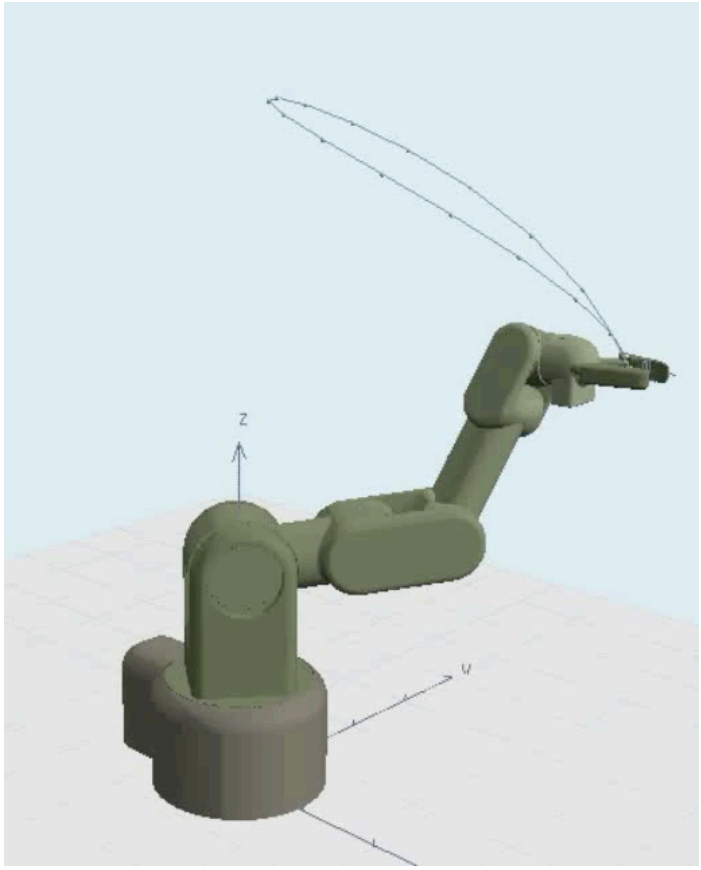
Joint and Cartesian trajectories

- 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)

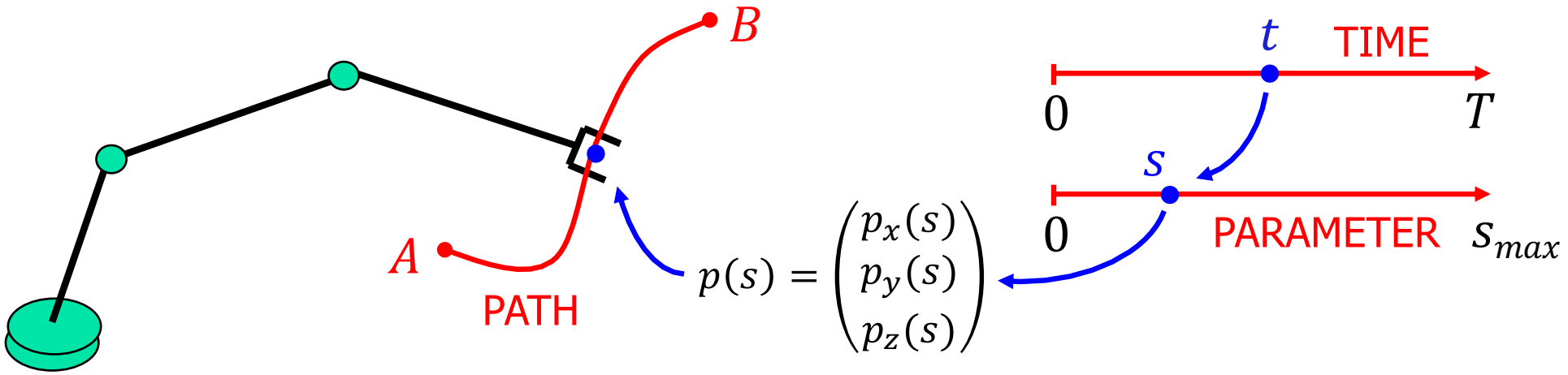
From task to trajectory

TRAJECTORY
=
GEOMETRIC PATH
+
TIMING LAW

- of motion $p_d(t)$ (or $q_d(t)$)
- of interaction $F_d(t)$

parameterized by s: $p = p(s)$
(e.g., s is the arc length)
describes the time evolution of $s = s(t)$

$\Big\}\ p(s(t))$



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

Trajectory planning
operative sequence

1

TASK planning

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

interpolation in Cartesian space

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

analytic inversion

2

path sampling and kinematic inversion

- sequence of "knots" in joint space

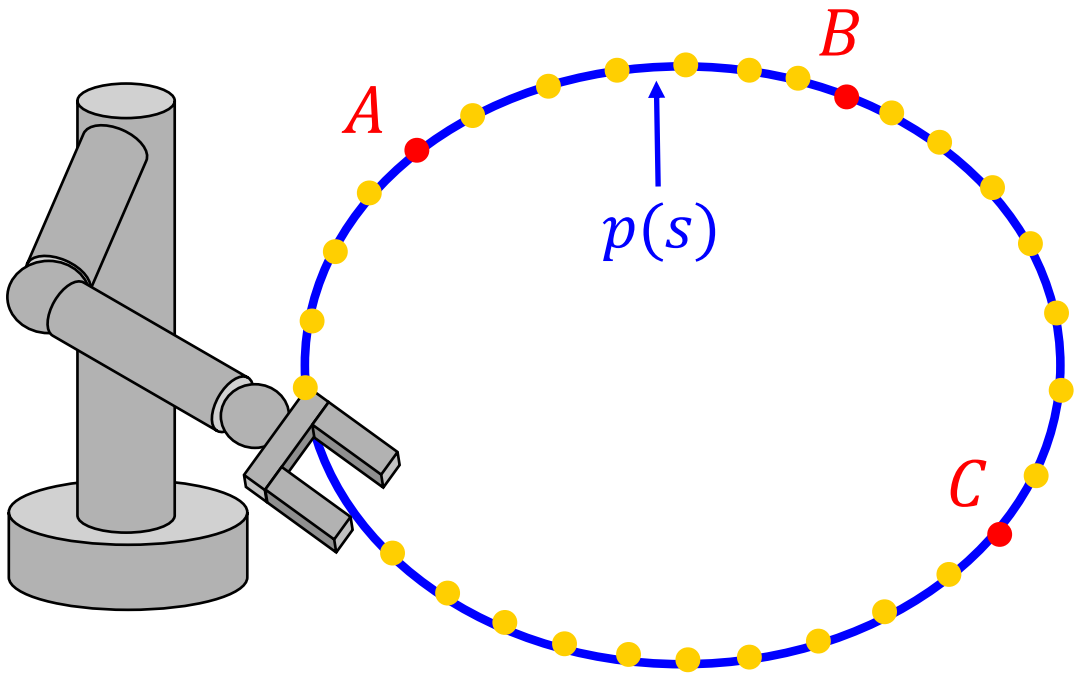
interpolation in joint space

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

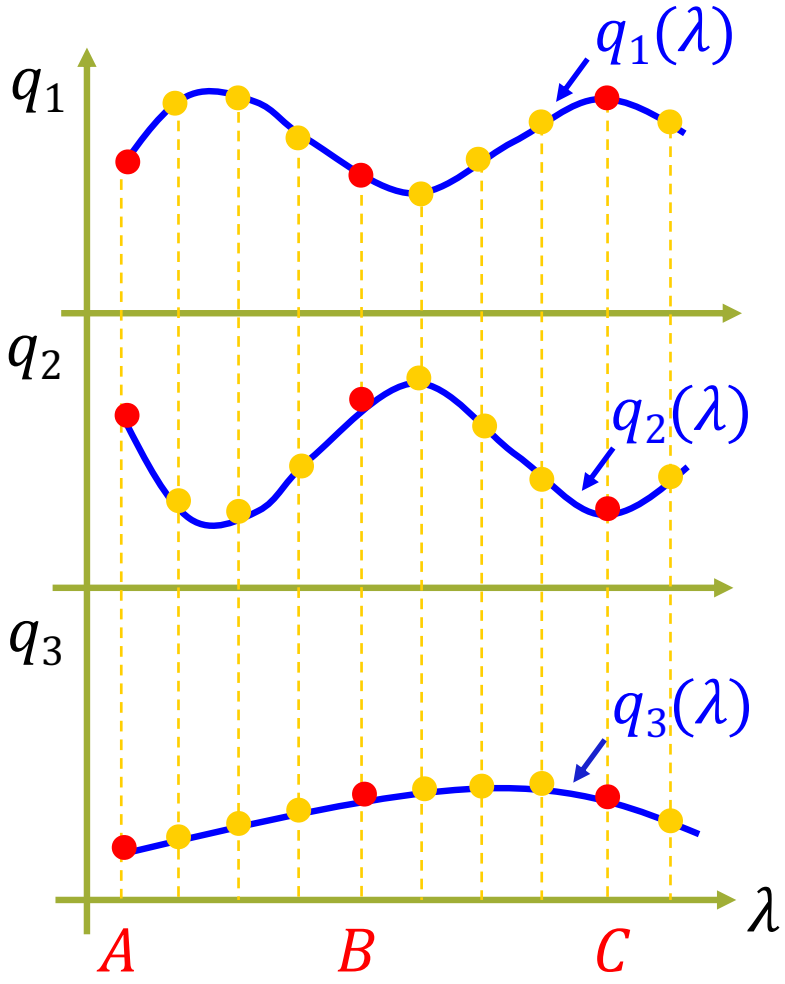
additional issues to be considered in the planning process

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

Example



Cartesian space



joint space

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

$$\dot{p}(t) = \frac{dp}{ds}\dot{s} \qquad \ddot{p}(t) = \frac{dp}{ds}\ddot{s} + \frac{d^2p}{ds^2}\dot{s}^2$$

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) starts 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 the joint space

Cartesian vs. joint trajectory planning

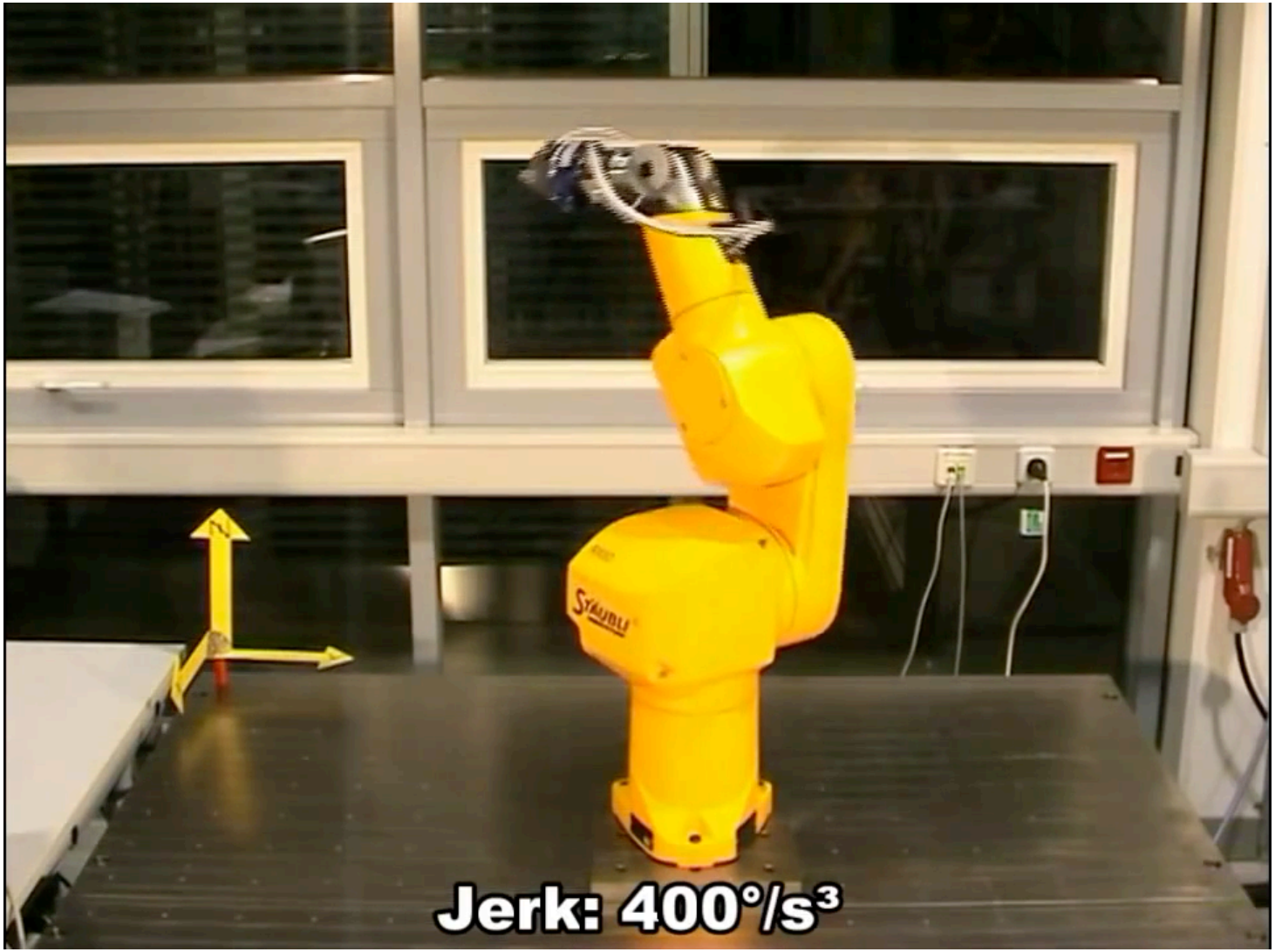
- planning in Cartesian space
  - allows a more direct visualization of the generated path
  - obstacle avoidance, lack of "wandering"
- planning in joint space
  - does not need on-line kinematic inversion
- issues in kinematic inversion
  - $\dot{q}$ and $\ddot{q}$ (or higher-order derivatives) may 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 environment interaction occurs or when 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,…), trigonometric (cosine, sines, combined, …), clothoids, …
- imposed conditions
  - passage through points = interpolation
  - initial, final, intermediate velocity (or geometric tangent for paths)
  - initial, final acceleration (or geometric curvature)
  - continuity of time-(or space-)derivative up to the k-th order: class 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

$q(0) = q_0$ $q(1) = q_1$ $q'(0) = v_0$ $q'(1) = v_1$ ← 4 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 → "doubly normalized" polynomial $q_N(\lambda)$

$q_N(0) = 0 \Leftrightarrow d = 0$ $\quad q_N(1) = 1 \Leftrightarrow a + b + c = 1$

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

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

$$\left.\begin{aligned} q'_N(0) &= 0 \Leftrightarrow c = 0 \\ q_N(1) &= 1 \Leftrightarrow a + b = 1 \\ q'_N(1) &= 0 \Leftrightarrow 3a + 2b = 0 \end{aligned}\right\} \Leftrightarrow \begin{aligned} a &= -2 \\ b &= 3 \end{aligned}$$

Cubic polynomial in time

$q(0) = q_{in}$ | $q(T) = q_{fin}$ | $\dot{q}(0) = v_{in}$ | $\dot{q}(T) = v_{fin}$ ← 4 conditions

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

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

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

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

$$q_N(0) = 0 \Leftrightarrow d = 0 \qquad q_N(1) = 1 \Leftrightarrow a + b + c = 1$$

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

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

$$q'_N(0) = 0 \Leftrightarrow c = 0$$

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

A trigonometric alternative

$q(0) = q_{in}$ $q(T) = q_{fin}$ $\dot{q}(0) = 0$ $\dot{q}(T) = 0$ ← boundary conditions (rest-to-rest)

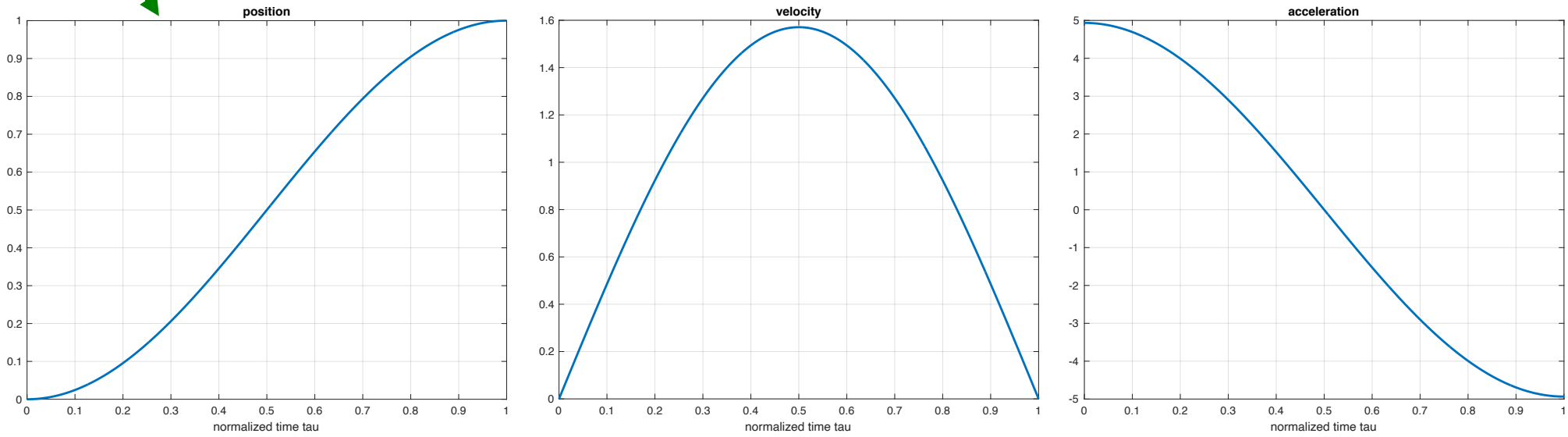
$$q(\tau) = q_{in} + \Delta q \frac{1 - \cos \pi\tau}{2}$$

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

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

doubly normalized

$$\dot{q}(\tau) = \frac{\Delta q}{T} \frac{\pi}{2} \sin \pi\tau \qquad \ddot{q}(\tau) = \frac{\Delta q}{T^2} \frac{\pi^2}{2} \cos \pi\tau$$



$$\max \dot{q}(\tau) = \dot{q}(0.5) = \frac{\Delta q}{T} \frac{\pi}{2} \qquad \max|\ddot{q}(\tau)| = \ddot{q}(0) = -\ddot{q}(1) = \frac{\Delta q}{T^2} \frac{\pi^2}{2}$$

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 $\tau = t/T$)

$q(0) = q_0$ | $q(1) = q_1$ | $q'(0) = v_0 T$ | $q'(1) = v_1 T$ | $q''(0) = a_0 T^2$ | $q''(1) = a_1 T^2$

$$q(\tau) = (1-\tau)^3(q_0 + (3q_0 + v_0 T)\tau + (a_0 T^2 + 6v_0 T + 12q_0)\tau^2/2) + \tau^3(q_1 + (3q_1 - v_1 T)(1-\tau) + (a_1 T^2 - 6v_1 T + 12q_1)(1-\tau)^2/2)$$

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

$$q(\tau) = q_0 + \Delta q(6\tau^5 - 15\tau^4 + 10\tau^3)$$ $\Delta q = q_1 - q_0$

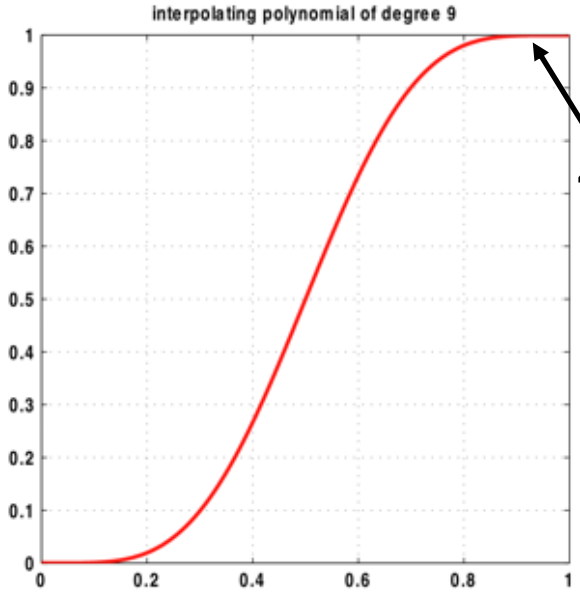
Higher-order polynomials

- 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 = (degree-1)/2
- in all other cases (e.g., for interpolating a large number N of points), their use is not recommended
  - there is a unique polynomial of degree $N-1$ interpolating N points
  - k-th degree polynomials have $k-1$ maximum and minimum points
  - oscillations arise out of the interpolation points (wandering)

Interpolating $N = 2$ knots

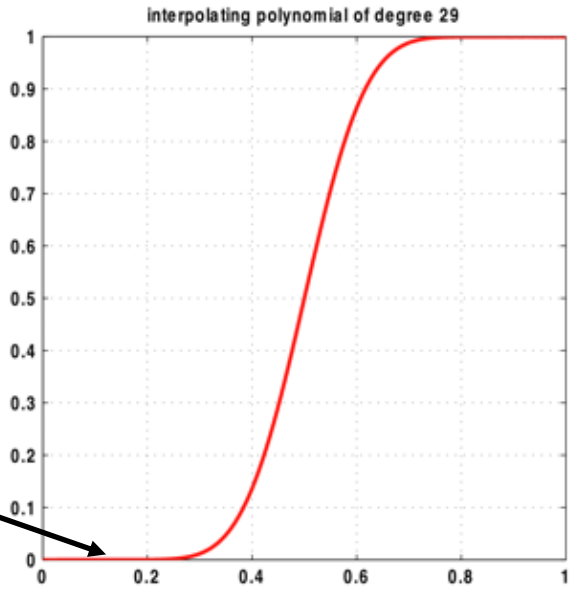
with high-order polynomials and zero boundary conditions

9th degree



4 derivatives are zero

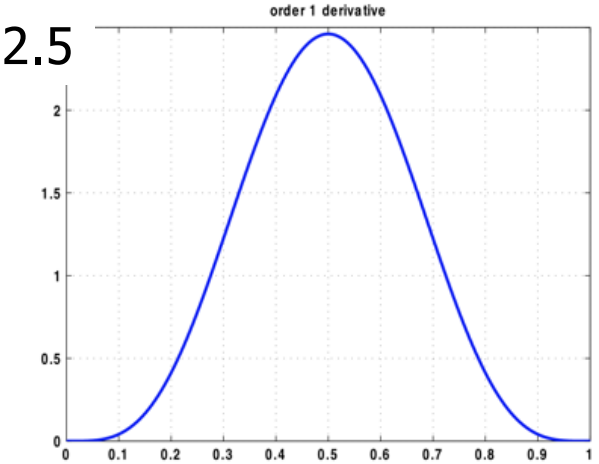
14 derivatives are zero!



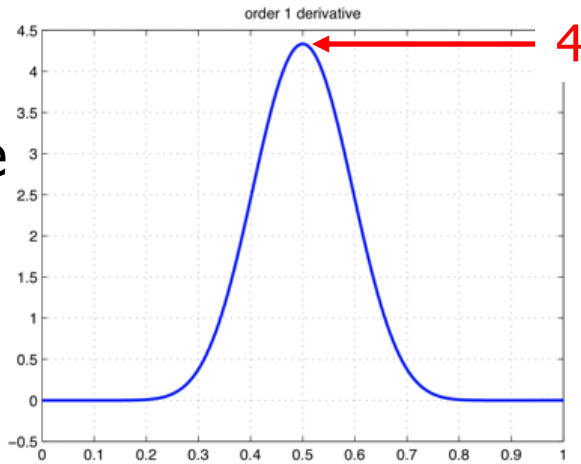
29th degree

no overshoot nor wandering

2.5



normalized first derivative (velocity in time)



4.5!!

peaking at midpoint

Interpolating N knots $q_1 \dots q_N$

with a **unique** polynomial of degree $N-1$

$N = 2 \Rightarrow$ a line

$$q(\tau) = a_0 + a_1\tau = q_1 + (q_2 - q_1)\tau$$

$N = 3 \Rightarrow$ a quadric

$$q(\tau) = a_0 + a_1\tau + a_2\tau^2$$

$$a_0 = q_1$$

$$a_1 = \frac{(q_3 - q_1)\tau_m{}^2 - (q_2 - q_1)}{\tau_m(\tau_m - 1)}$$

$$a_2 = \frac{(q_2 - q_1) - (q_3 - q_1)\tau_m}{\tau_m(\tau_m - 1)}$$

$$\tau_m \in (0,1), \qquad q(\tau_m) = q_2$$

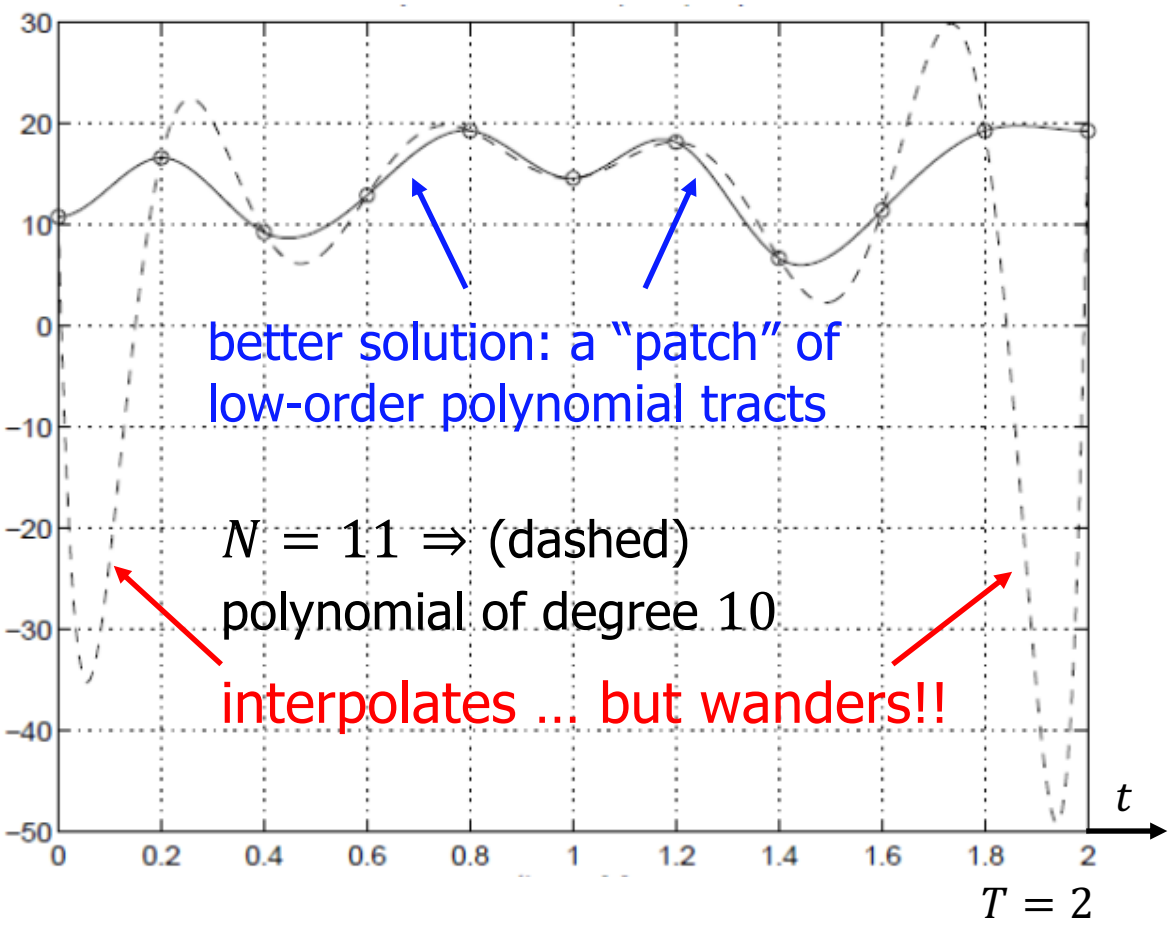
$N = 4 \Rightarrow$ a cubic

$$q(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3$$

$N \Rightarrow$ a polynomial of degree $N-1$

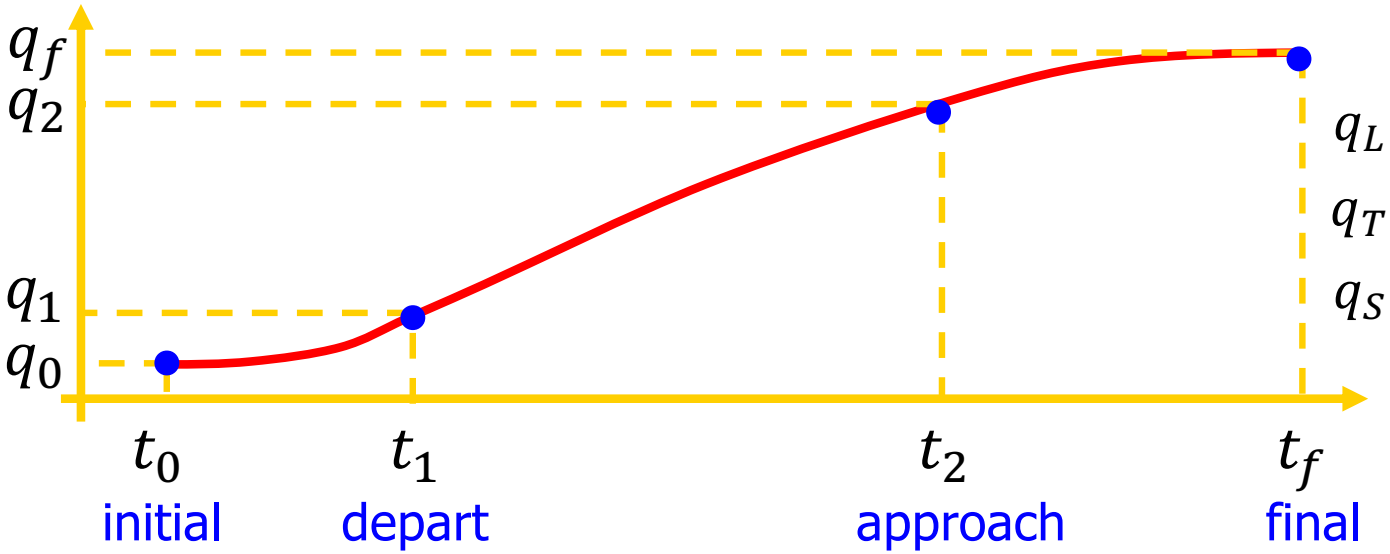
$$q(\tau) = a_0 + a_1\tau + \cdots + a_{N-1}\tau^{N-1}$$

$$\tau = \frac{t}{T} \in [0,1]$$



4-3-4 polynomials

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



$q_L(t) =$ 4th order polynomial

$q_T(t) =$ 3rd order polynomial

$q_S(t) =$ 4th order polynomial

14 coefficients

boundary conditions

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

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

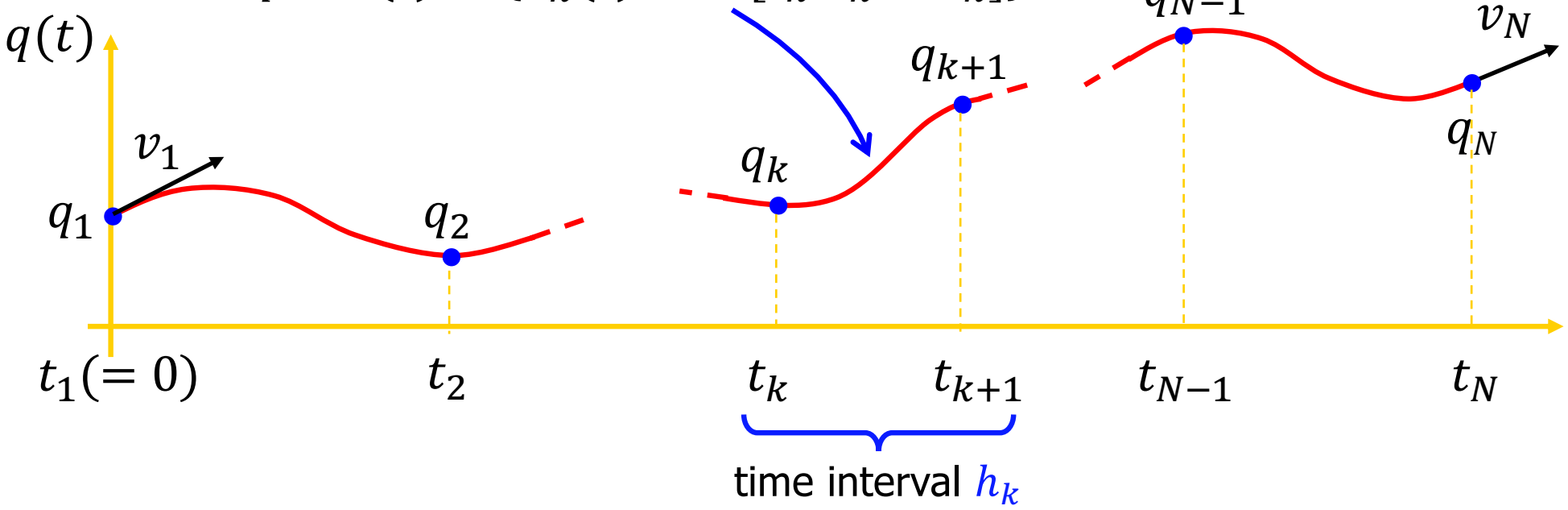
$$\dot{q}(t_i^-) = \dot{q}(t_i^+) \qquad \ddot{q}(t_i^-) = \ddot{q}(t_i^+) \qquad i = 1,2$$ } 4 continuity up to acceleration

Interpolation using splines

- problem
  interpolate N knots, with continuity up to the second derivative
- solution
  spline: $N-1$ cubic polynomials, concatenated so to pass through N knots, and 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

$$q = \theta(t) = \{\theta_k(t), t \in [t_k, t_k + h_k]\}$$



$$\theta_k(\tau) = a_{k0} + a_{k1}\tau + a_{k2}\tau^2 + a_{k3}\tau^3$$

$$\tau = t - t_k \in [0, h_k]$$

$$(k = 1, \cdots, N-1)$$

continuity conditions
for velocity and acceleration

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

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

$$\theta_k(0) = q_k = a_{k0} \qquad \dot{\theta}_k(0) = v_k = a_{k1}$$

$$\begin{pmatrix} h_k^2 & h_k^3 \\ 2h_k & 3h_k^2 \end{pmatrix} \begin{pmatrix} a_{k2} \\ a_{k3} \end{pmatrix} = \begin{pmatrix} q_{k+1} - q_k - v_k h_k \\ v_{k+1} - v_k \end{pmatrix} \quad (1)$$

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

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

3. 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 solvable

$$\begin{pmatrix} & & \\ & \boldsymbol{A}(h_1, \cdots, h_{N-1}) & \\ & & \end{pmatrix} \begin{pmatrix} v_2 \\ v_3 \\ \vdots \\ \vdots \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} \vdots \\ \boldsymbol{b}(h_1, \cdots, h_{N-1}, q_1 \cdots, q_N, v_1, v_N) \\ \vdots \end{pmatrix}$$

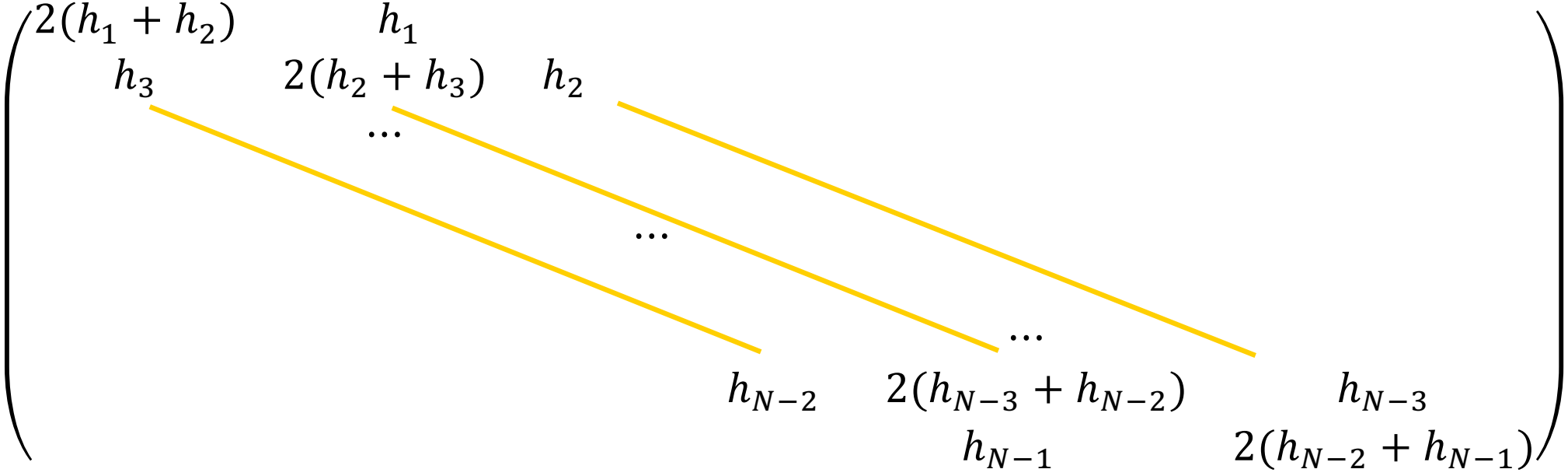
tri-diagonal matrix always invertible

unknown

known vector

to be substituted then back in (1)

Structure of $A(\boldsymbol{h})$



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

Structure of $b(\boldsymbol{h}, \boldsymbol{q}, v_1, v_N)$

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

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 (when in time, 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 $q_1, \cdots, q_N$, $h_1, \cdots, h_{N-1}, v_1, v_N$
- 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 $[0, T]$
- spline construction can be suitably modified when the second derivative (in time, 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 the 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 (o) ⇒ 3 cubic polynomials
  - joint values $q_1 = 0,\ q_2 = 2\pi,\ q_3 = \pi/2\,,\ q_4 = \pi$
  - at $t_1 = 0, t_2 = 2, t_3 = 3, t_4 = 5 \Rightarrow h_1 = 2, h_2 = 1, h_3 = 2$
  - 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 $t_3' = 4.5$ (×); or at $t_1'' = 1.5$ and $t_4'' = 3.5$ ($*$)