Robotics 1

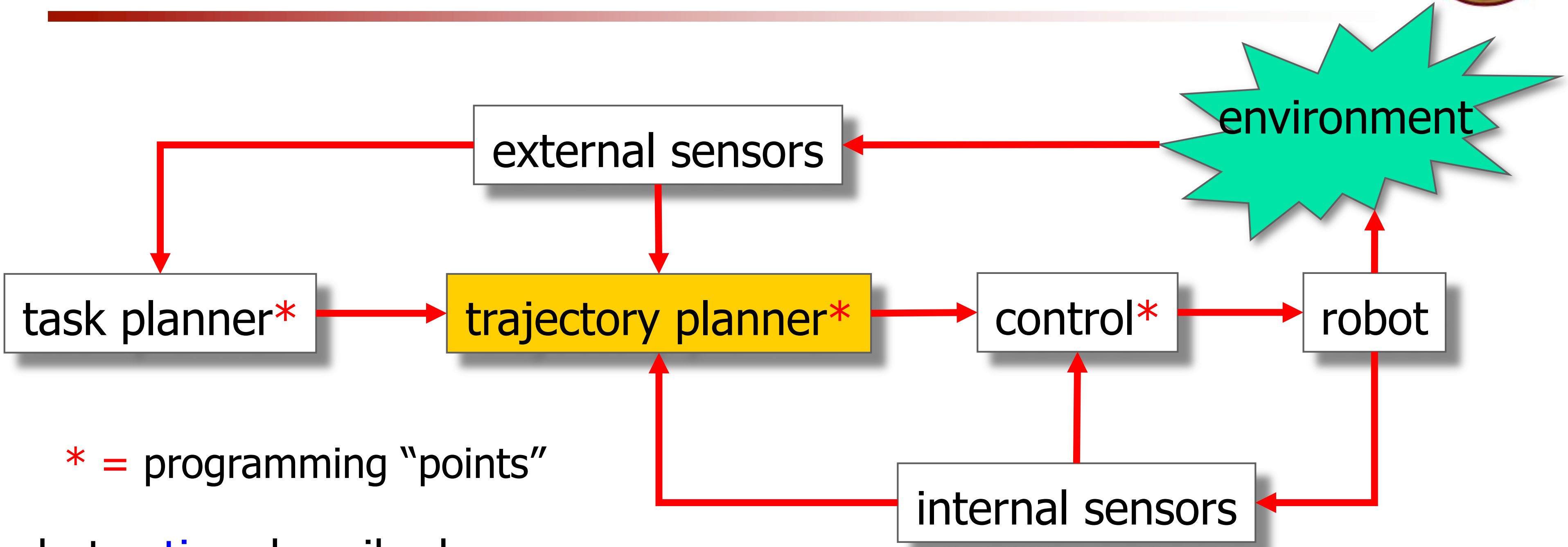
Trajectory planning

Prof. Alessandro De Luca

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Trajectory planner interfaces



* = programming "points"

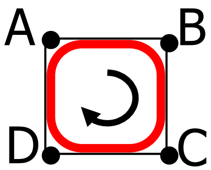
robot action described as a sequence of poses or configurations (with possible exchange of contact forces) → **trajectory planning** → reference profile/values (continuous or discrete) for the robot controller

this is **not** motion planning (i.e., find a collision-free path among obstacles): obstacles are not considered here, except for very simple situations

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

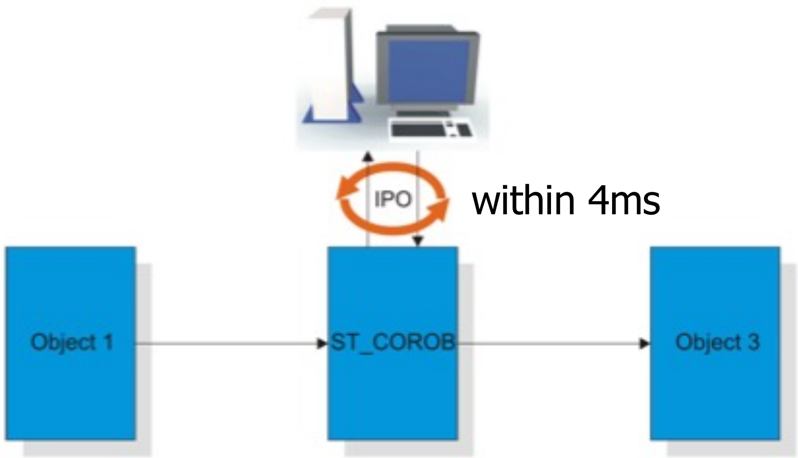
a) over-fly 

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)

communication protocols for user-defined programs, also based on external sensors

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)

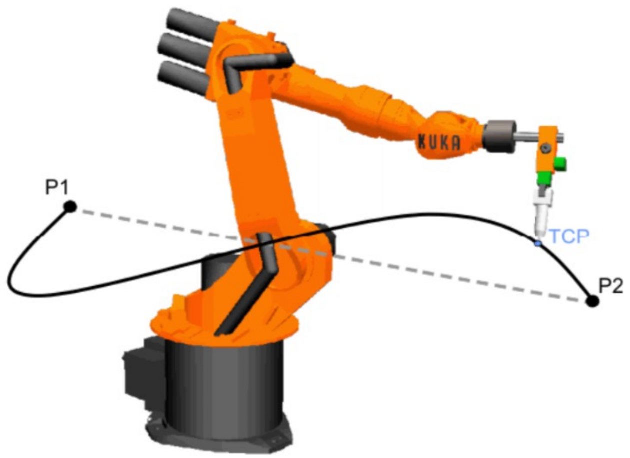
Communication
(>>> 10.11 "Communication" page 176)

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

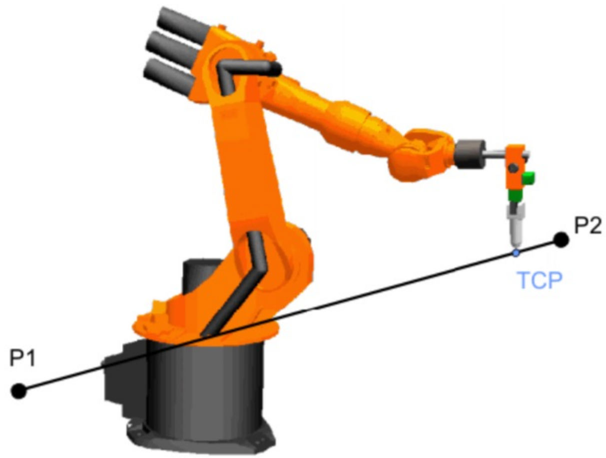
- basic data set: frames, vectors + DECLaration

KRL language

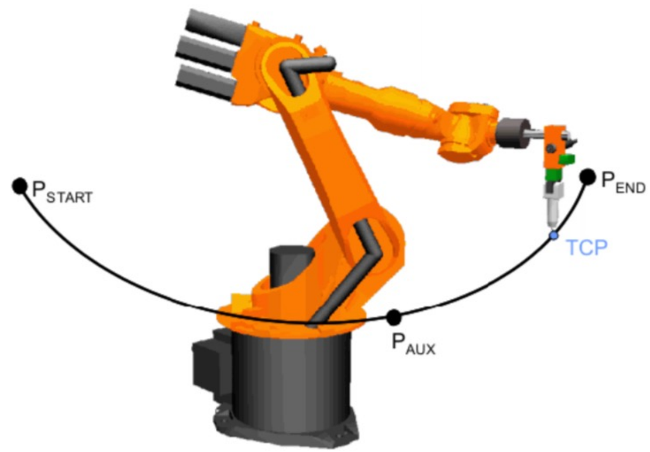
- typical motion primitives



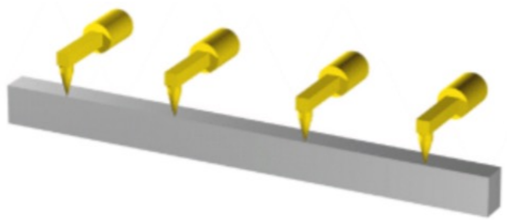
PTP motion
(point-to-point, linear
in joint space)



LIN motion
(linear in
Cartesian space)

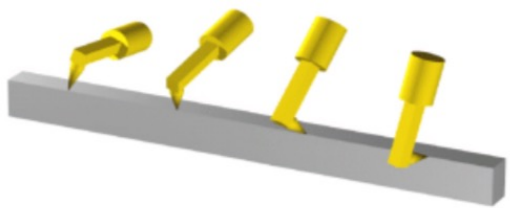


CIRC motion
(circular in
Cartesian space)



CONST orientation

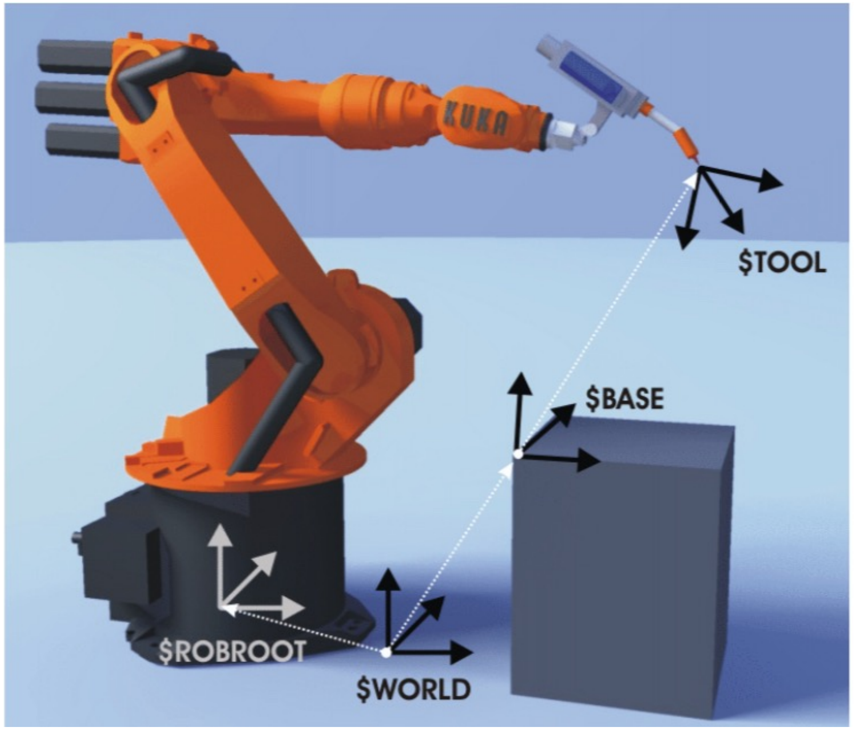
end-effector
orientation

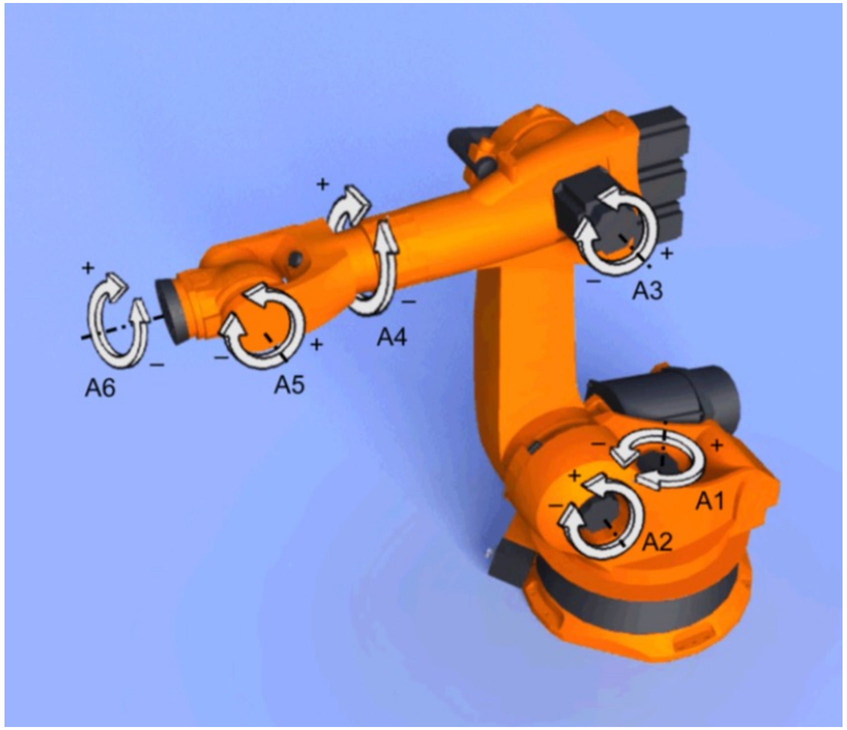


PTP motion
(linear in RPY angles)

KRL language

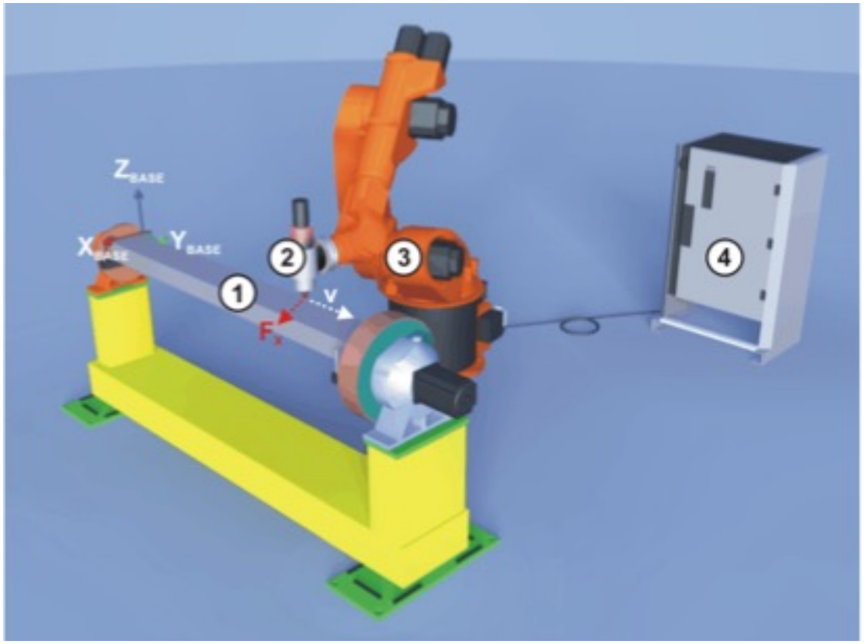
- multiple coordinate frames (in Cartesian space) and jogging of robot joints





Example of RSI use - 1

- deburring task with robot motion controlled by a force sensor



① workpiece to be deburred along the edge under force control

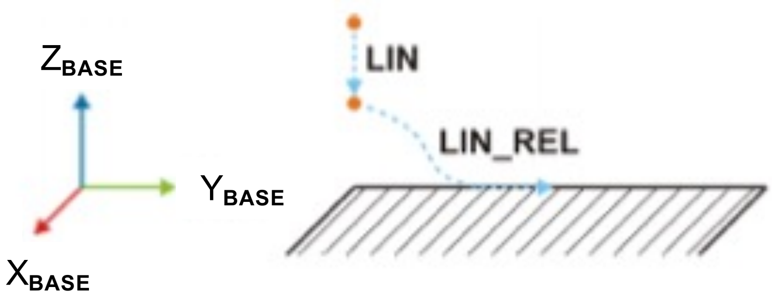
② tool with force sensor

③ robot

④ robot controller

F_X measured force in the X direction of the BASE coordinate system (perpendicular to the programmed path)

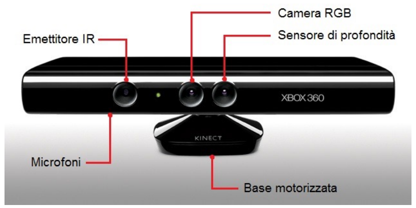
v direction of motion



LIN_REL = linear Cartesian path relative to an initial position (specified here by the force sensor signal)

Example of RSI use - 3

- human-robot interaction through vocal and gesture commands
- voice and human gestures acquired through a Kinect sensor



Kinect RGB-D sensor (with microphone)

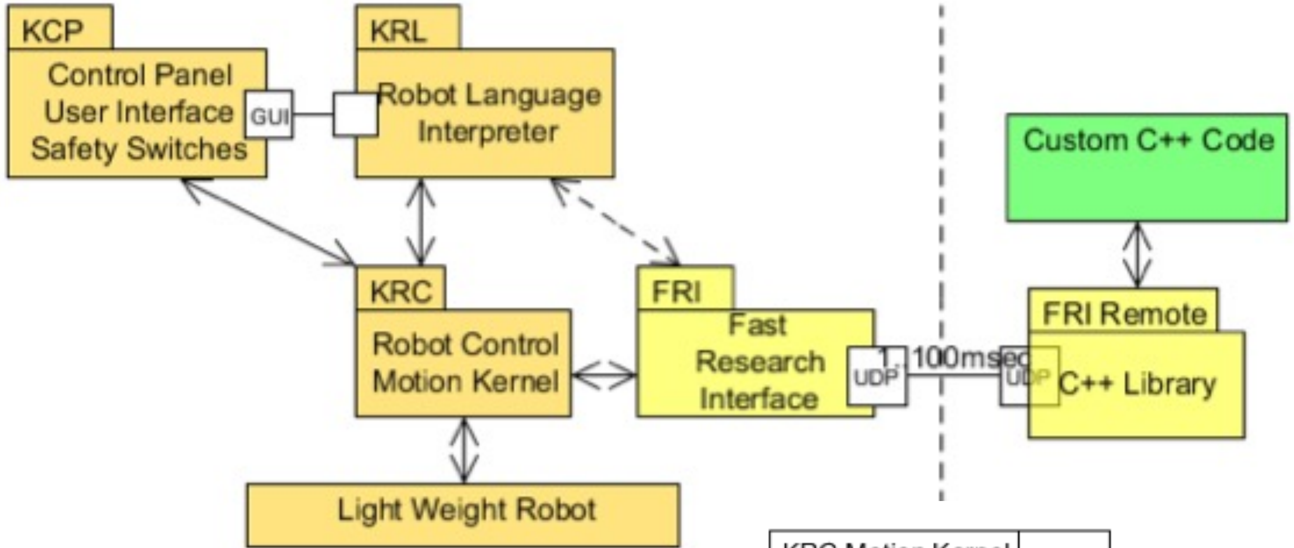
simple vocabulary, e.g.:
- listen to me
- give me
- follow
  - right/left hand
  - the nearest hand
- thank you
- stop collaboration

video

Fast Research Interface (FRI)

for KUKA Light Weight Robot (LWR-IV)

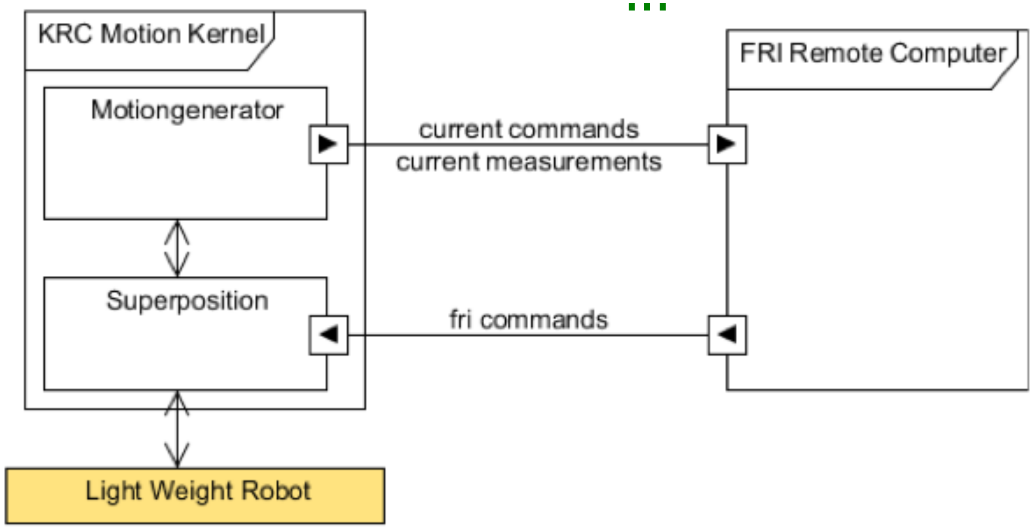
- UDP socket communication up to 1 KHz (1÷100 ms cycle time)



here, we develop
our C++/ROS code for:

- trajectory planning
- kinematic control
- redundancy resolution
- torque/dynamic control
- physical HRI
- ...

available
at DIAG
Robotics Lab
since Sep 2012



....from 06_ProgrammingArchitectures block of slides (#14)

Kinematic control using the FRI

KUKA Light Weight Robot (LWR-IV)

- joint velocity commands that mimic second-order control laws (defined in terms of acceleration or torques), exploiting task redundancy of the robot
- discrete-time implementation is simpler and still very accurate

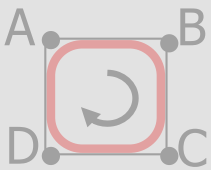
video

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

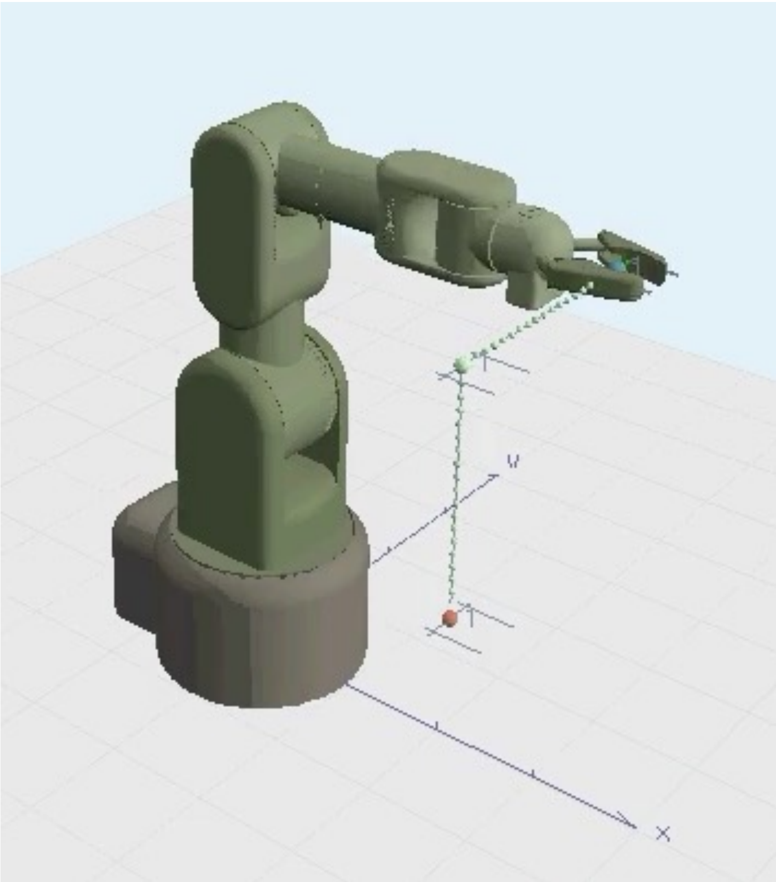
- semi-manual programming (as in "first generation" robot languages)
- limited visualization of motion

➡ a mathematical formalization of trajectories is useful/needed

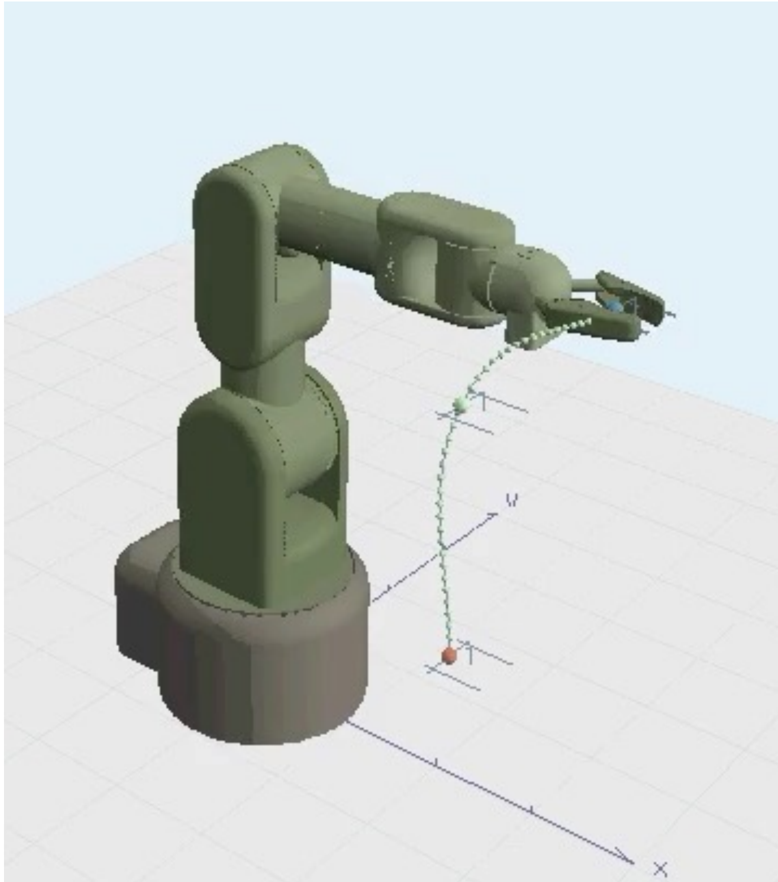
Some typical trajectories

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

video



Straight lines as Cartesian path



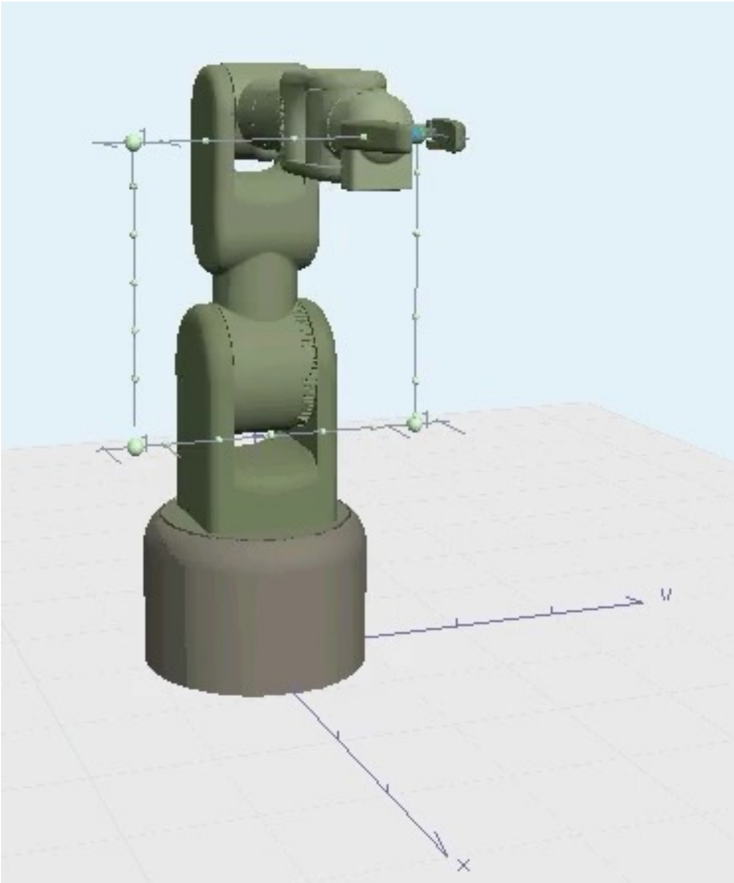
video

Interpolation with Bezier curves

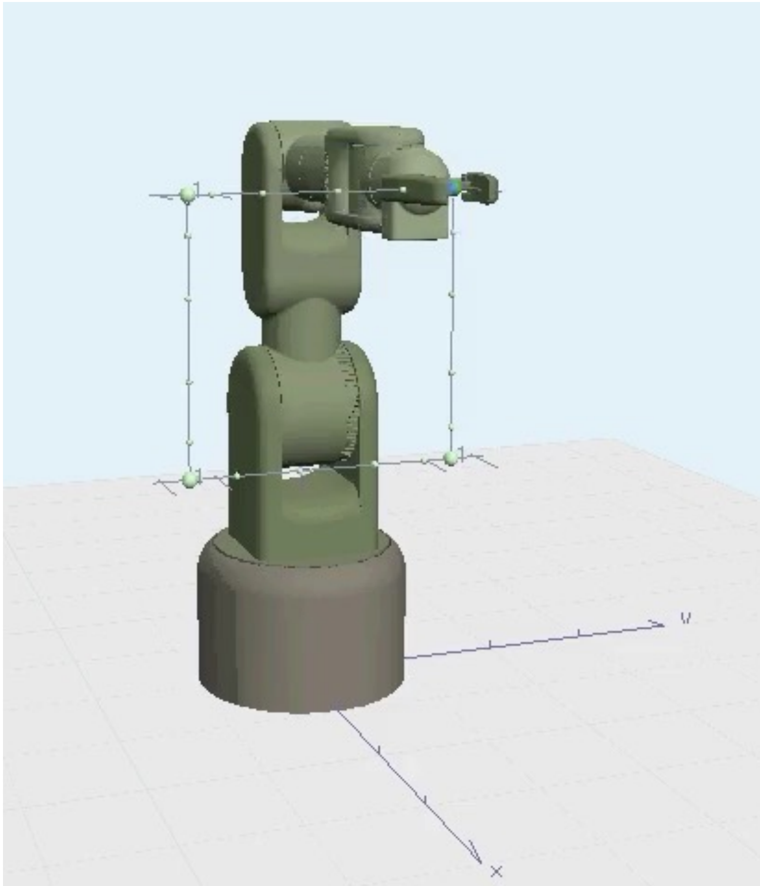
Some typical trajectories

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

video



Square path at constant speed

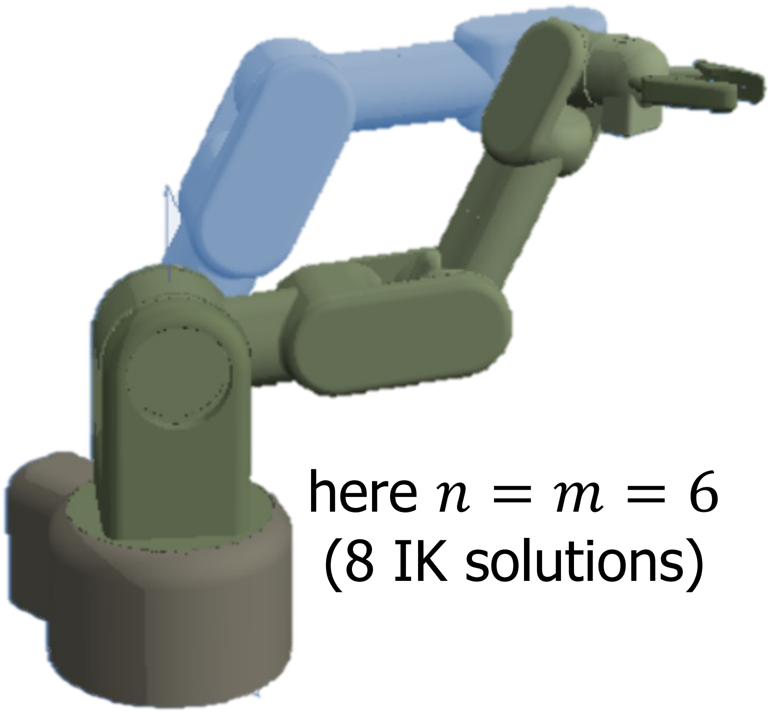


video

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

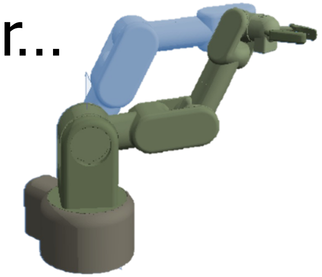


here $n = m = 6$
(8 IK solutions)

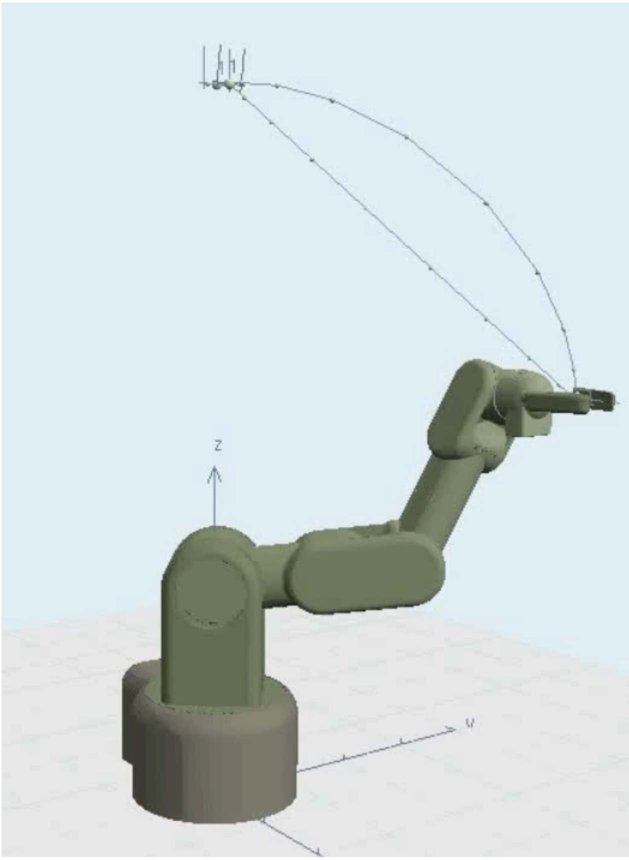
- initial and final configuration
- same Cartesian pose (no change!): the motion cannot be fully specified in the Cartesian space
- to perform this task, the robot should leave the given end-effector pose and then return to it
- a self-motion could be sufficient
  - if there is (task) redundancy ($m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

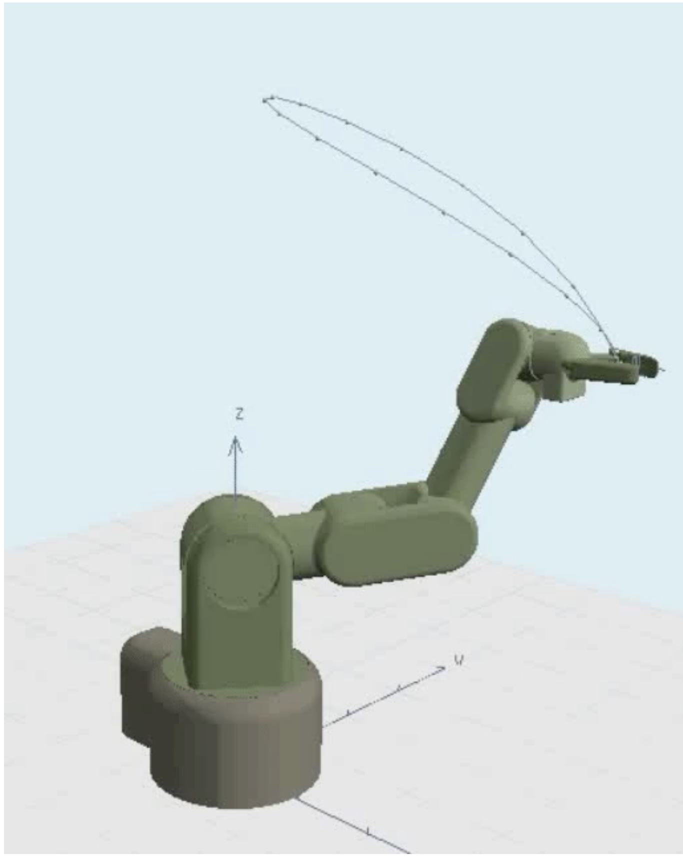
Joint and Cartesian trajectories

- a reconfiguration task (or…  passing through singularity)

video



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



video

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

From task to trajectory

TRAJECTORY
- of motion $p_d(t)$ (or $q_d(t)$)
- of interaction $F_d(t)$ – not pursued here

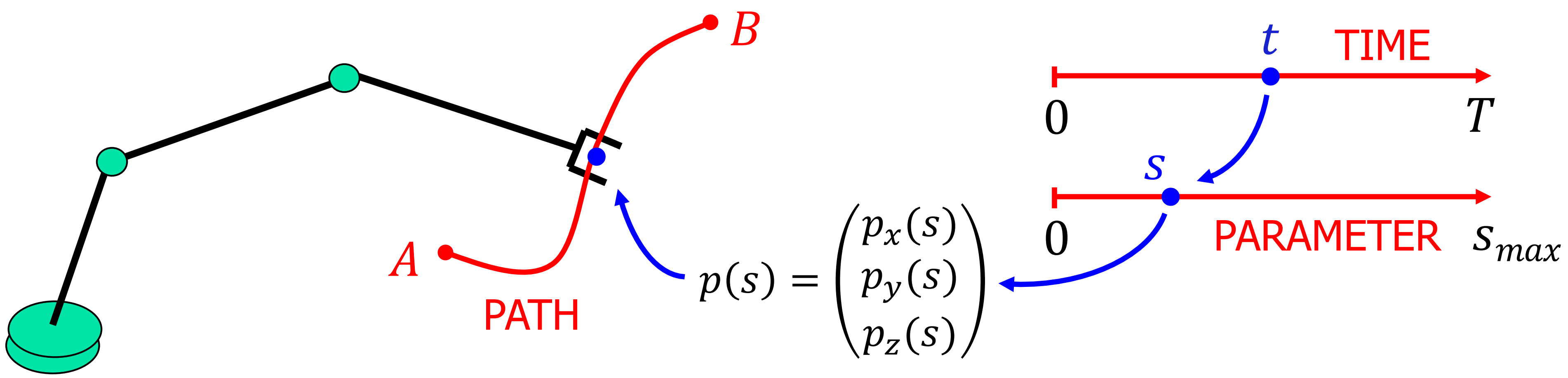
||

GEOMETRIC PATH — parameterized by s: $p = p(s)$ (any s or the arc length)

\+

TIMING LAW — describes the time evolution of $s = s(t)$

$\Rightarrow p(s(t))$



$$p(s) = \begin{pmatrix} p_x(s) \\ p_y(s) \\ p_z(s) \end{pmatrix}$$

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

Trajectory planning

operative sequence

TASK planning

1

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

  - interpolation in Cartesian space

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

2

  - path sampling and kinematic inversion

- sequence of "knots" in joint space

  - interpolation in joint space

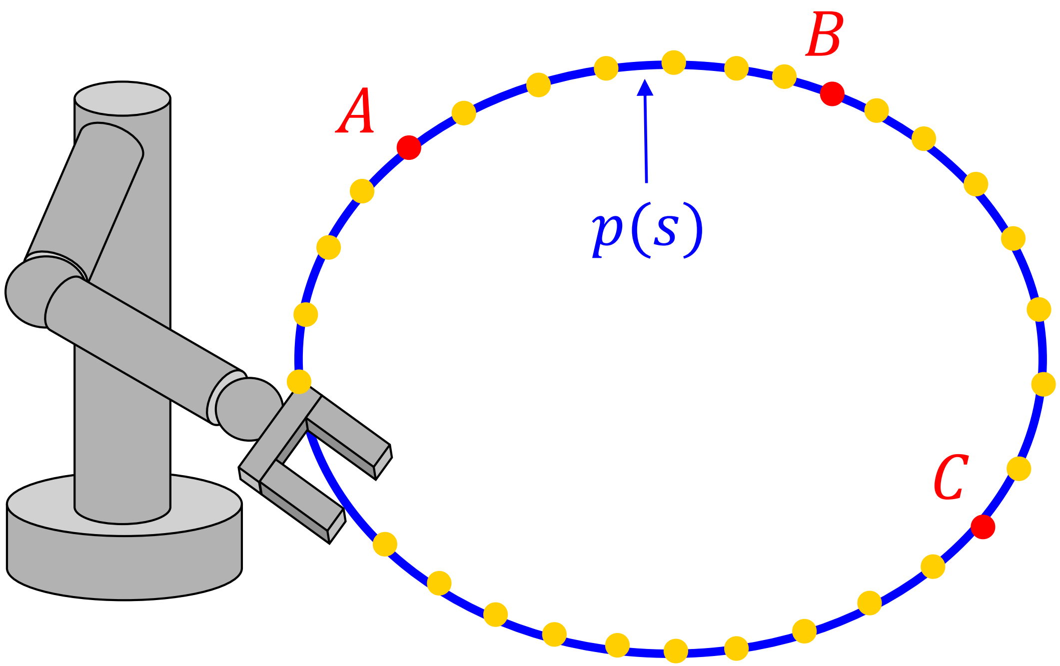
- geometric path in joint space: $q = q(s)$

analytic inversion

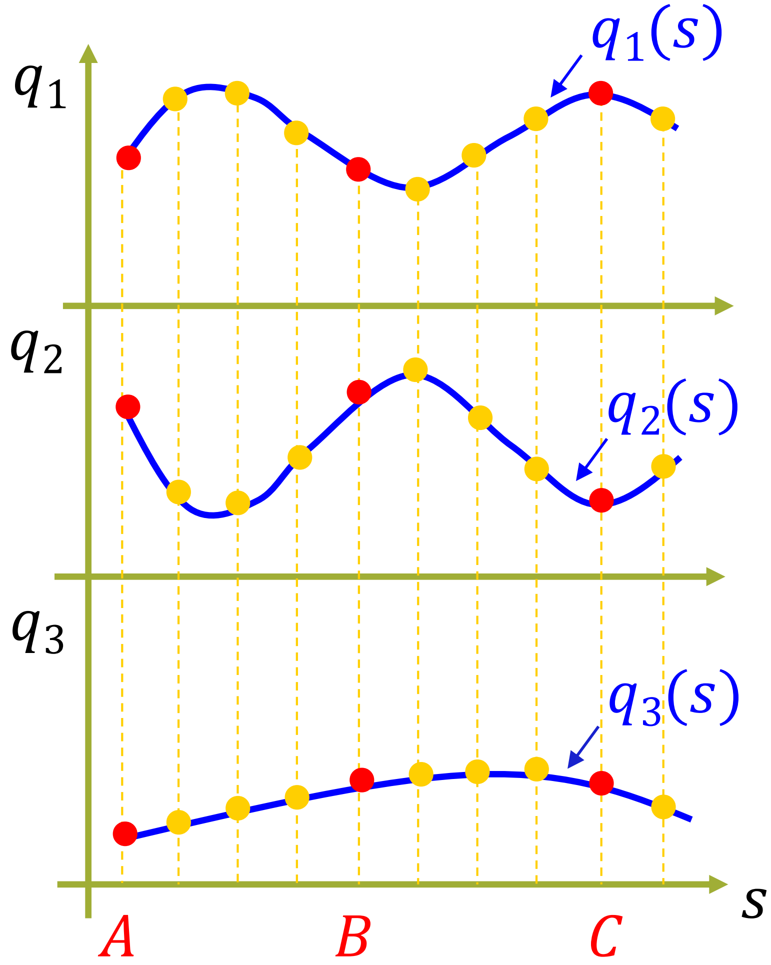
additional issues to be considered in the planning process

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

Example



Cartesian space



joint space

Path and timing law

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

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

$$\Rightarrow \quad s = s(t)$$

$$q = q(s) \qquad \text{(joint space)}$$

  - if $s(t) = t$, path parameterization is given directly by time
- the timing law
  - is chosen based on task specifications (stop in a point, move at constant speed, and so on)
  - may consider optimality criteria (min transfer time, min jerk,…)
  - constraints are imposed by actuator capabilities (max torque, max velocity,…) and/or by the task (e.g., max acceleration on payload)
  - global optimum solutions typically require also to change the path (min PTP time ≤ min time on a given path between the two points!)

Path + Timing law = Trajectory

cubic path from $q_i = 0$ to $q_f = \pi$ [rad]

with tangents $q'_i = q'_f = 1$

$$q(s) = s + 3(\pi - 1)s^2 - 2(\pi - 1)s^3$$

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

quintic timing law in $T = t_f - t_i = 4$ [s]

rest-to-rest $(\dot{s}(t_i) = \dot{s}(t_f) = 0)$

and $\ddot{s}(t_i) = \ddot{s}(t_f) = 0$

$$s(t) = 10\,\tau^3 - 15\tau^4 + 6\tau^5$$

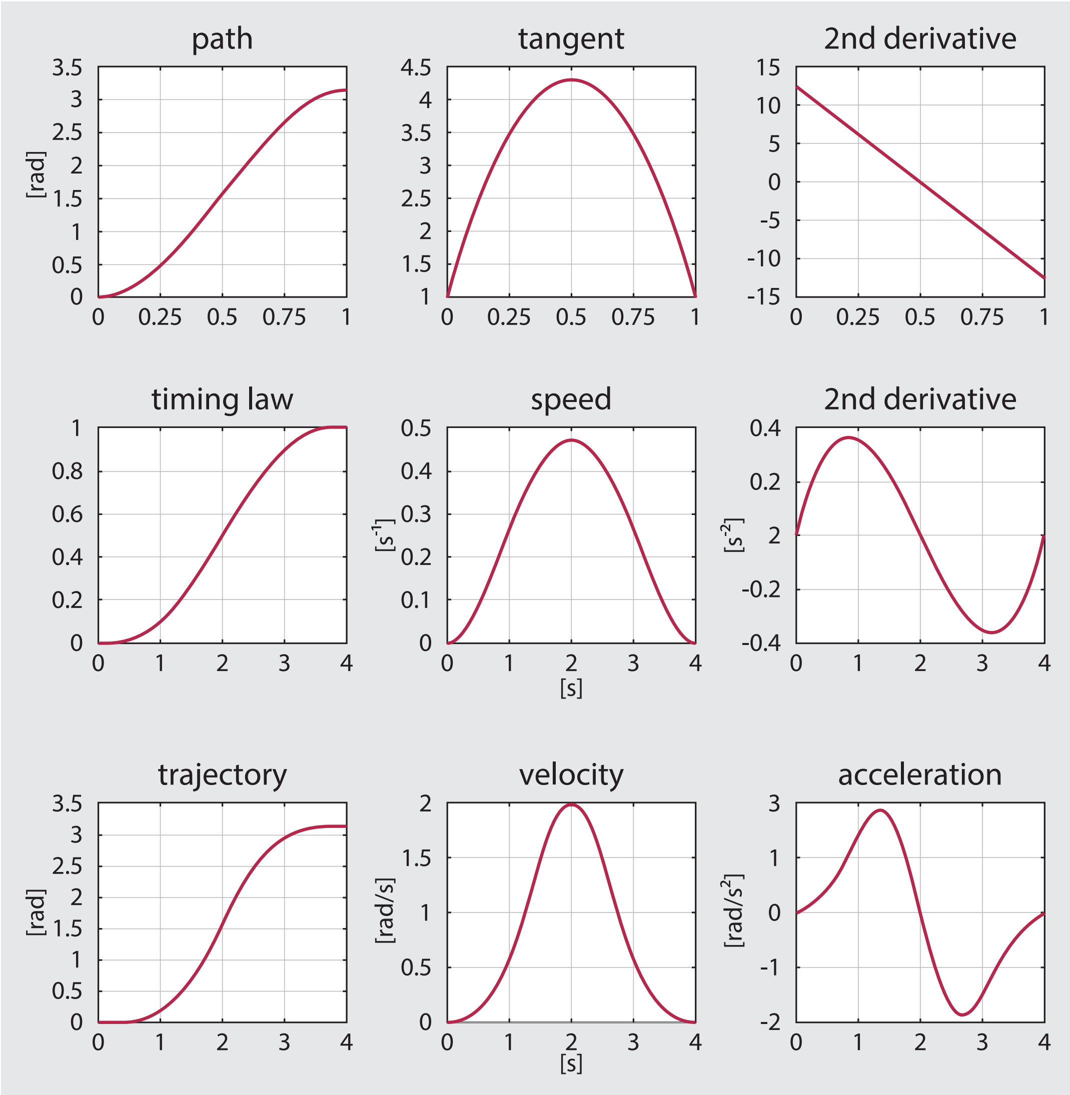
$$\tau = \frac{t}{T} \in [0,1]$$

⬇

the trajectory is a polynomial in time of degree 15…

$$q(t) = q\big(s(t)\big) \qquad t \in [0,4]$$

$$(\dot{q}(t_i) = \dot{q}\big(t_f\big) = \ddot{q}(t_i) = \ddot{q}(t_f) = 0)$$



Trajectory classification

- planning space
  - Cartesian, joint
- motion task
  - point-to-point (PTP), multi-point (MP) with knots, concatenated
- geometric path
  - rectilinear, polynomial, harmonic, 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

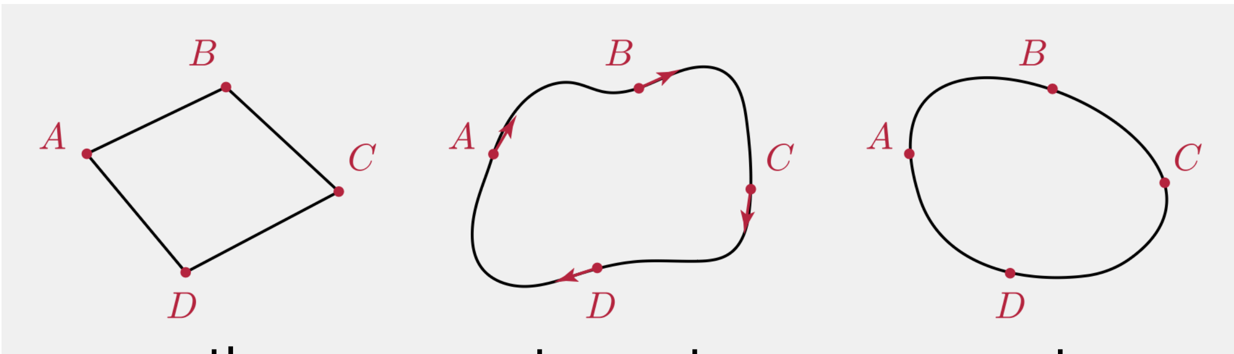
Issues in trajectory planning

- planning in Cartesian space
  - allows a more direct visualization of the generated path
  - obstacle avoidance, lack of "wandering"
- planning in joint space
  - no need of (online) kinematic inversion, may cross singularities
- offline planning is easier but not always possible
- online (re-)planning needed when environment interaction occurs or when sensor-based motion is required
- task specification may involve also
  - boundary conditions / bounds on geometric path and timing law
  - conditions / inversion of higher-order derivatives (e.g., p'' or $\ddot{q}$)
  - for redundant robots, choice among ∞^{n-m} inverse solutions, based on optimality criteria or additional auxiliary tasks

Relevant characteristics

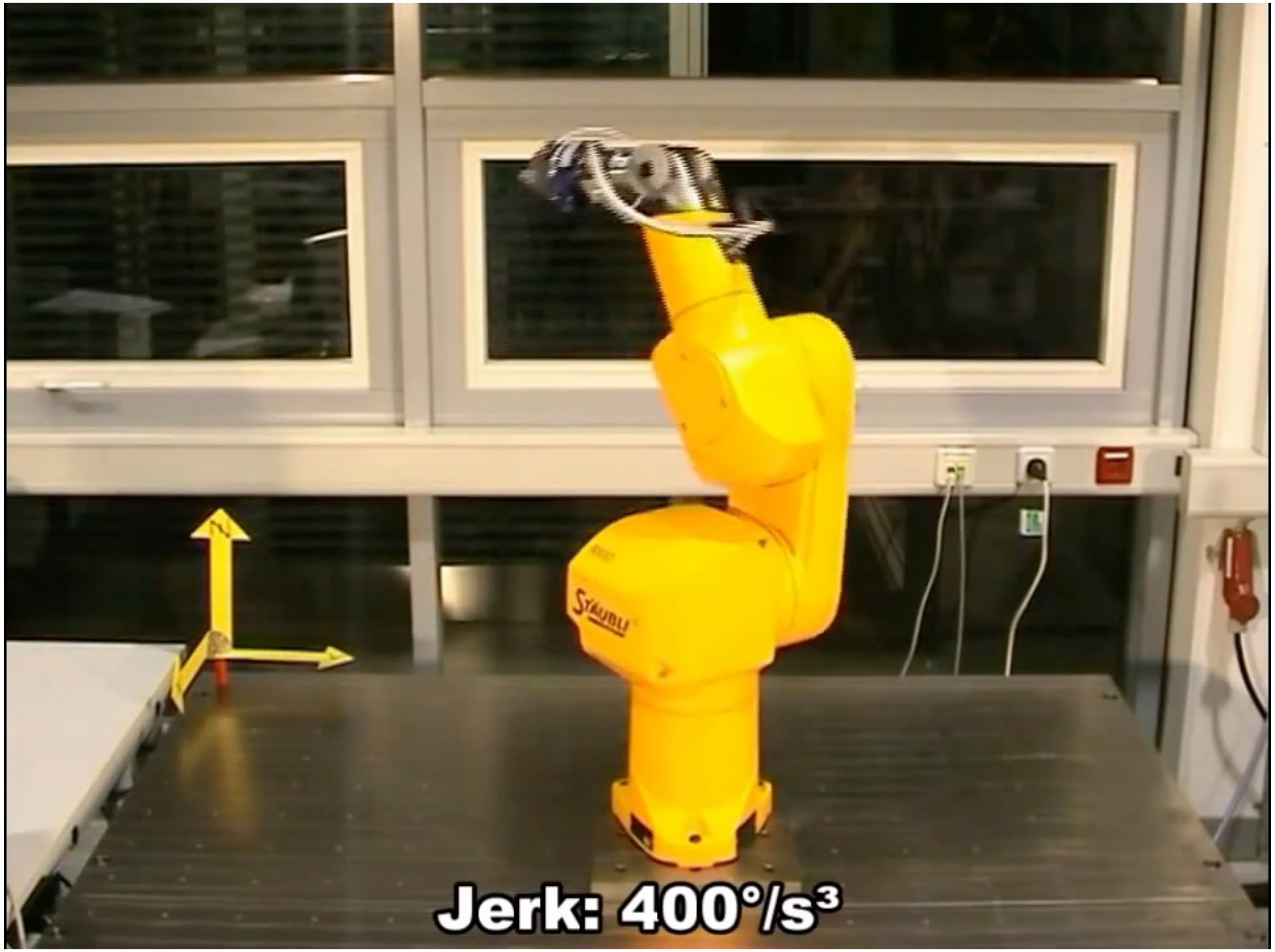
- predictability and accuracy
  - no "wandering" out of the knots or "overshoot" on final position
- flexibility
  - allow concatenation of primitive segments, over-fly of knots
- computational efficiency and memory space for MP tasks
  - e.g., store only the coefficients of polynomial functions
- smoothness
  - in space and/or in time
  - at least C^1 in time, but also continuity up to jerk ($\dddot{p}$ or $\dddot{q}$ or $\dddot{s}$)

spatial continuity of



path tangent curvature

A robot trajectory with bounded jerk



video

Path planning

- assume to work in the n-dimensional joint (configuration) space

$$q(s) = \begin{pmatrix} q_1(s) \\ q_2(s) \\ \vdots \\ q_n(s) \end{pmatrix} \qquad s \in [s_i, s_f]$$

generic parametrization

- path derivatives

tangent vector to the path

$$q'(s) = \frac{dq}{ds} \qquad q''(s) = \frac{d^2 q}{ds^2}$$

"related" to the path curvature $\kappa(s)$

- path curvature (for a generic parametrization)

$$\kappa(s) = \frac{\|q'(s) \times q''(s)\|}{\|q'(s)\|^3}$$

if $q(s) \in \mathbb{R}$ (a scalar function)

$$\kappa(s) = \frac{|q''(s)|}{\sqrt{(1+s^2)^3}}$$

- regularity condition for the path parametrization by s

$$q'(s) \neq 0 \qquad s \in [s_i, s_f]$$

no cusps
bounded curvature

Path planning

- special parametrization by the arc length $\sigma = \sigma(s)$

$$\sigma(s) = \int_{s_i}^{s} \|q'(r)\| \, dr$$

invariant w.r.t. the choice of s

- path parametrization using σ

$$q(\sigma) \qquad \sigma \in \left[\sigma(s_i), \sigma\left(s_f\right)\right] = [0, L]$$

L is the path length

- path derivatives using σ

unit tangent vector to the path $\|q'(\sigma)\| = 1$ $\qquad$ $\kappa(\sigma) = \|q''(\sigma)\|$ path curvature

- a space-time decomposition takes place on parameterized paths

$$\dot{q}(t) = \frac{dq}{ds}\dot{s}(t) = q'\dot{s} \qquad \ddot{q}(t) = \frac{dq}{ds}\ddot{s}(t) + \frac{d^2q}{ds}\dot{s}^2(t) = q'\,\ddot{s} + q''\,\dot{s}^2$$

or, equivalently, in Cartesian space $\dot{p}(t) = \frac{dp}{ds}\dot{s}(t) = p'\,\dot{s} \qquad \ddot{p}(t) = \frac{dp}{ds}\ddot{s}(t) + \frac{d^2p}{ds^2}\dot{s}^2(t) = p'\,\ddot{s} + p''\,\dot{s}^2$

Trajectory bounds

from actuation limits to bounds on timing law

- for a given robot path $q(s)$, consider velocity actuation limits

$$|\dot{q}_j(t)| \leq v_{j,\max} \qquad j = 1, \dots, n \qquad \forall t \in [0, T]$$

$$\Rightarrow \max_{s\in[s_i,s_f],\, t\in[0,T]} |q_j'(s)|\,|\dot{s}(t)| \leq v_{j,\max} \qquad j = 1, \dots, n$$

$$\Rightarrow \max_{t\in[0,T]} |\dot{s}(t)| \leq \min_{j=1,\dots,n} \frac{v_{j,\max}}{\max_{s\in[s_i,s_f]} |q_j'(s)|} \quad \Rightarrow \quad \boxed{|\dot{s}(t)| \leq V \quad t \in [0, T]}$$

the higher are the q_j', the smaller is V

- similar but conservative handling of acceleration actuation limits

$$|\ddot{q}_j(t)| \leq a_{j,\max} \qquad j = 1, \dots, n \qquad \forall t \in [0, T]$$

$$\Rightarrow \max_{s\in[s_i,s_f],\, t\in[0,T]} |q_j'(s)\ddot{s}(t) + q_j''(s)\dot{s}^2(t)| \leq a_{j,\max} \qquad j = 1, \dots, n$$

conservative inequalities

$$|\ddot{q}_j| = |q_j'\,\ddot{s} + q_j''\,\dot{s}^2| \leq |q_j'||\ddot{s}| + |q_j''|\,\dot{s}^2 \leq |q_j'||\ddot{s}| + |q_j''|\,V^2$$

$$\Rightarrow \max_{t\in[0,T]} |\ddot{s}(t)| \leq \min_{j=1,\dots,n} \frac{a_{j,\max} - V^2 \max_{s\in[s_i,s_f]} |q_j''(s)|}{\max_{s\in[s_i,s_f]} |q_j'(s)|} \quad \Rightarrow \quad \boxed{|\ddot{s}(t)| \leq A \quad t \in [0, T]}$$

Trajectory planning in joint space

- $q = q(t)$ in time or $q = q(s)$ in space (then, with timing $s = s(t)$)
- in general, it is sufficient to work component-wise (q_j in vector q)
- implicit definition of trajectory, by solving an interpolation problem with specified boundary conditions (b.c.) in a class of functions
- typical classes: polynomials (linear, cubic, quintic,...), trigonometric (cosines, sines, combined, ...), clothoids, exponentials, ...
- imposed conditions (in space and/or in time) [+ bounds/limits]
  - passage through points = interpolation (PTP or MP)
  - initial, final, intermediate velocity (or geometric tangent for paths)
  - initial, final acceleration (or curvature/second space derivative)
  - 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 component-wise also to Cartesian trajectory planning!

PTP cubic polynomial in space

$\boxed{q(0) = q_i}$ $\boxed{q(1) = q_f}$ $\boxed{q'(0) = v_i}$ $\boxed{q'(1) = v_f}$ ← 4 conditions

$$q(s) = q_i + \Delta q\underbrace{(as^3 + bs^2 + cs + d)}$$

$\Delta q = q_f - q_i$

$s \in [0,1]$

4 coefficients → "doubly normalized" polynomial $q_N(s)$

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

$q'_N(0) = dq_N/ds|_{s=0} = c = v_i/\Delta q$ $\quad$ $q'_N(1) = dq_N/ds|_{s=1} = 3a + 2b + c = v_f/\Delta q$

special case: $v_i = v_f = 0$ (zero tangent)

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

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

PTP path planning in joint space
from initial Cartesian data

- 2R planar arm with unitary link lengths
- from $p_i = (0.6, -0.4)$ to $p_f = (1, 1)$ [m]
- with $p_i' = (-2, 0)$ and $p_f' = (2,2)$
- inverse kinematics (elbow-down solution)

$$q_i = (-1.790, 2.439) \text{ and } q_f = (0, \pi/2) \text{ [rad]}$$

- inverse differential kinematics (on tangents)

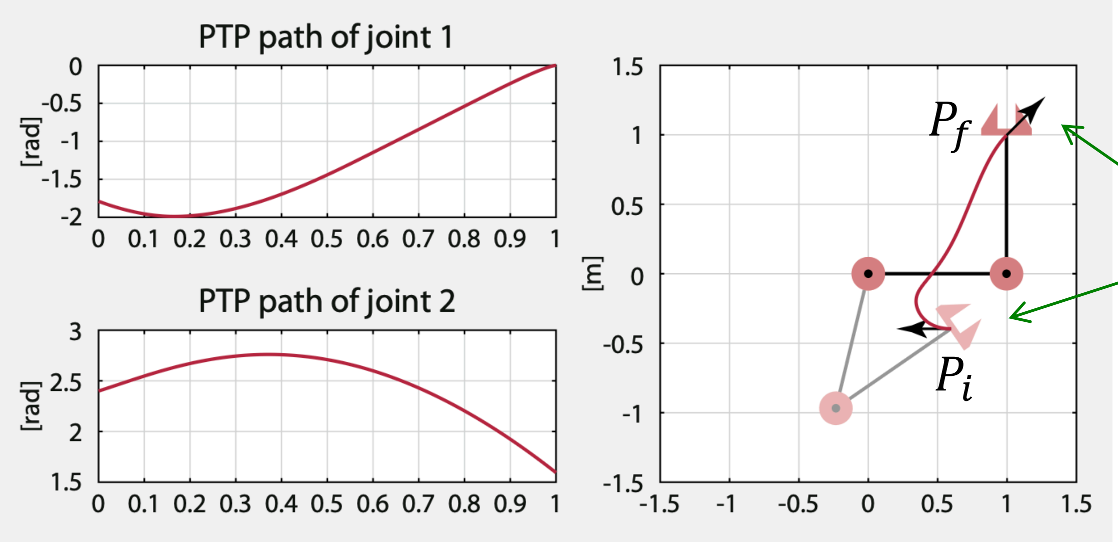
$$J(q) = \begin{pmatrix} -(s_1 + s_{12}) & -s_{12} \\ c_1 + c_{12} & c_{12} \end{pmatrix} \Rightarrow J(q_i) = \begin{pmatrix} 0.4 & -0.576 \\ 0.6 & 0.817 \end{pmatrix} \quad J(q_f) = \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix}$$

$$q_i' = J^{-1}(q_i) p_i' = \begin{pmatrix} -2.430 \\ 1.784 \end{pmatrix} \qquad q_f' = J^{-1}(q_f) p_f' = \begin{pmatrix} 2 \\ -4 \end{pmatrix}$$

- cubic interpolation

$$q(s) = \begin{pmatrix} q_1(s) \\ q_2(s) \end{pmatrix} = \begin{pmatrix} -1.790 - 2.430\, s + 8.230\, s^2 - 4.010\, s^3 \\ 2.439 + 1.784\, s - 2.067\, s^2 - 0.550\, s^3 \end{pmatrix} \qquad s \in [0,1]$$

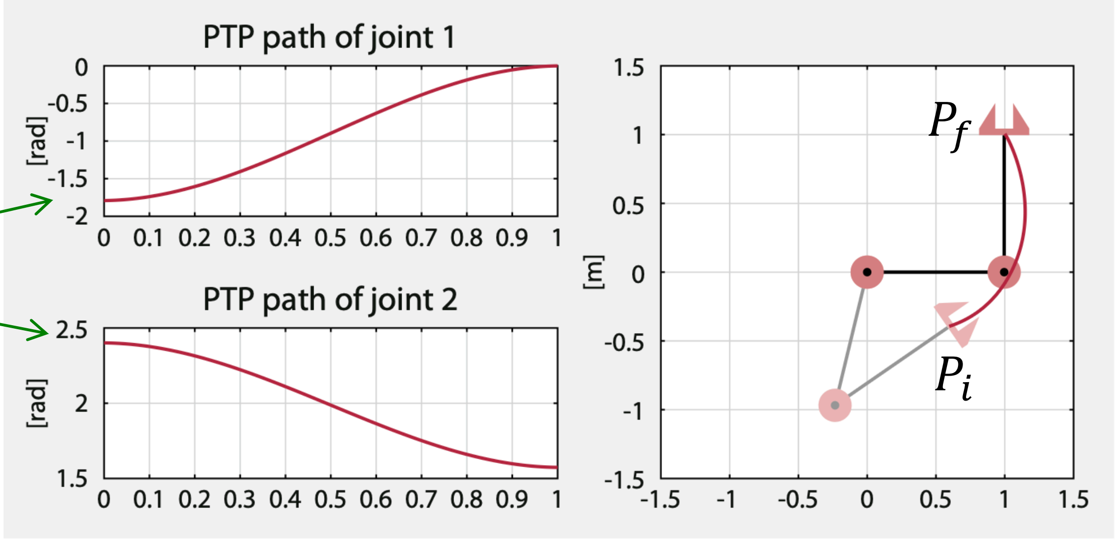
PTP path planning in joint space
from initial Cartesian data



initial/final tangents assigned to the Cartesian path

- dropping requirements on p_i' and p_f' and imposing instead $q_i' = q_f' = 0$

initial/final tangents assigned to the path in joint space



PTP cubic polynomial in time

$\boxed{q(0) = q_i}$ $\boxed{q(T) = q_f}$ $\boxed{\dot{q}(0) = v_i}$ $\boxed{\dot{q}(T) = v_f}$ ⟵ 4 conditions

$$\Delta q = q_f - q_i \qquad \tau = t/T \in [0,1]$$

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

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

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

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

special case: $v_i = v_f = 0$ (rest-to-rest)

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

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

PTP - A trigonometric alternative

$\boxed{q(0) = q_i}$ $\boxed{q(T) = q_f}$ $\boxed{\dot{q}(0) = 0}$ $\boxed{\dot{q}(T) = 0}$ ← boundary conditions (rest-to-rest)

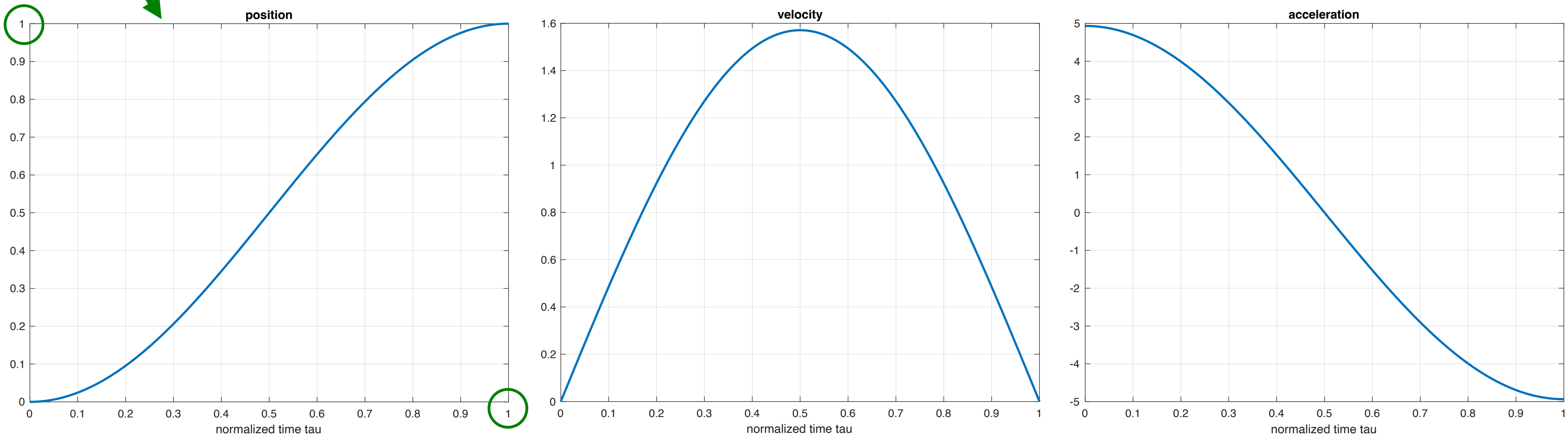
$$q(\tau) = q_i + \Delta q \, \underbrace{\frac{1 - \cos \pi\tau}{2}}$$

$\Delta q = q_f - q_i$

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

doubly normalized

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



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

PTP 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 $\tau = t/T$)

$$\boxed{q(0) = q_i} \quad \boxed{q(1) = q_f} \quad \boxed{q'(0) = v_i T} \quad \boxed{q'(1) = v_f T} \quad \boxed{q''(0) = a_i T^2} \quad \boxed{q''(1) = a_f T^2}$$

$$\begin{aligned} q(\tau) = {} & (1-\tau)^3\left(q_i + (3q_i + v_i T)\tau + (a_i T^2 + 6 v_i T + 12 q_i)\tau^2/2\right) \\ & + \tau^3\left(q_f + (3q_f - v_f T)(1-\tau) + (a_f T^2 - 6 v_f T + 12 q_f)(1-\tau)^2/2\right) \end{aligned}$$

special case: $v_i = v_f = a_i = a_f = 0$

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

$$\Delta q = q_f - q_i$$

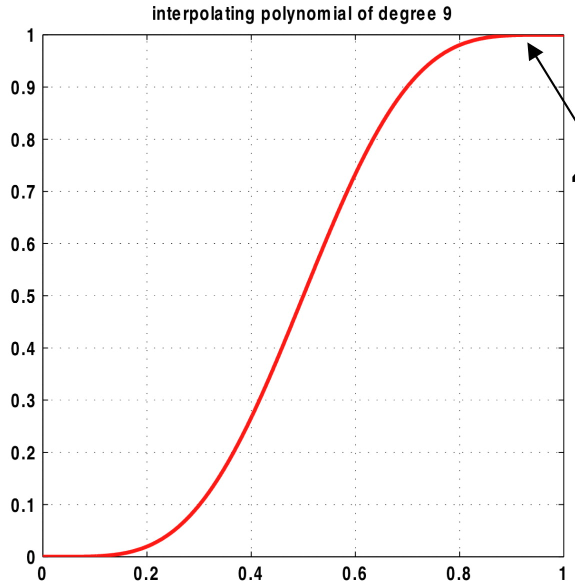
Higher-order polynomials

- 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) polynomials are always integers, alternate in sign, sum up to unity, and are zero for all terms up to the power = (degree-1)/2
- for MP tasks (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)

PTP interpolation

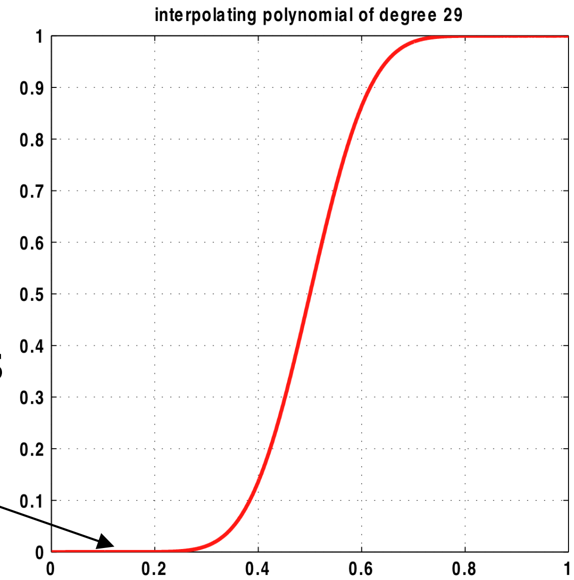
with higher-order polynomials and zero boundary conditions

9th degree



4 derivatives are zero

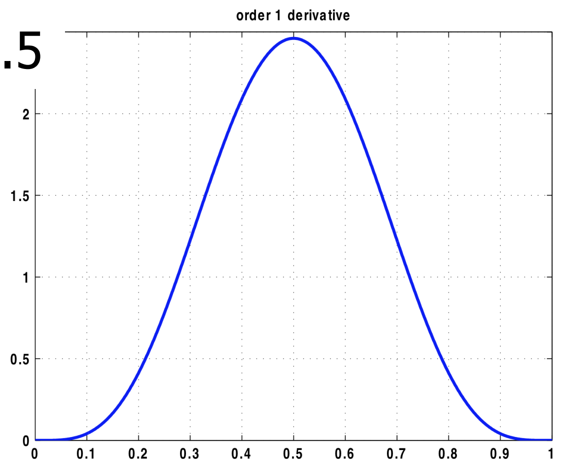
14 derivatives are zero!



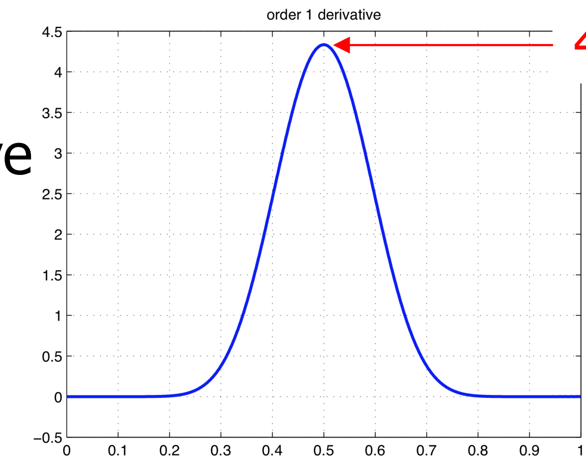
29th degree

no overshoot nor wandering

2.5



normalized first derivative (velocity in time)

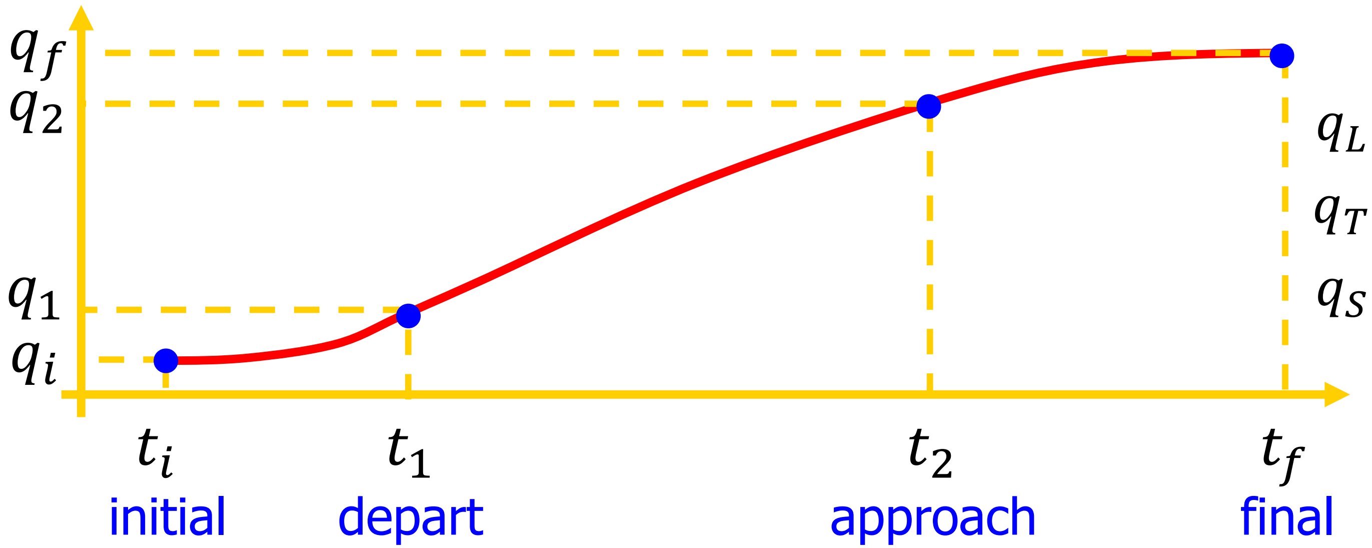


4.5!!

peaking at midpoint

4-3-4 polynomials
(special MP interpolation of $N = 4$ knots in time)

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

$$q(t_i) = q_i \quad q(t_1^-) = q(t_1^+) = q_1 \quad q(t_2^-) = q(t_2^+) = q_2 \quad q(t_f) = q_f \quad \Big\} \text{ 6 passages}$$

$$\dot{q}(t_i) = \dot{q}(t_f) = 0 \qquad \ddot{q}(t_i) = \ddot{q}(t_f) = 0 \quad \Big\} \text{ 4 initial/final velocity/acceleration}$$

$$\dot{q}(t_k^-) = \dot{q}(t_k^+) \qquad \ddot{q}(t_k^-) = \ddot{q}(t_k^+) \qquad k = 1,2 \quad \Big\} \text{ 4 continuity up to acceleration}$$

the solution to this 14-dimensional linear system can be found in symbolic form!

MP interpolation of N knots $\bar{q}_1 \dots \bar{q}_N$

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

$N = 2 \;\Rightarrow\;$ a line

$$q(\tau) = a_0 + a_1\tau$$
$$= \bar{q}_1 + (\bar{q}_2 - \bar{q}_1)\tau$$

$N = 3 \;\Rightarrow\;$ a quadratic

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

$$a_0 = \bar{q}_1$$

$$a_1 = \frac{(\bar{q}_3 - \bar{q}_1)\tau_m^2 - (\bar{q}_2 - \bar{q}_1)}{\tau_m(\tau_m - 1)}$$

$$a_2 = \frac{(\bar{q}_2 - q_1) - (\bar{q}_3 - \bar{q}_1)\tau_m}{\tau_m(\tau_m - 1)}$$

at $\tau_m \in (0,1)$, $q(\tau_m) = \bar{q}_2$

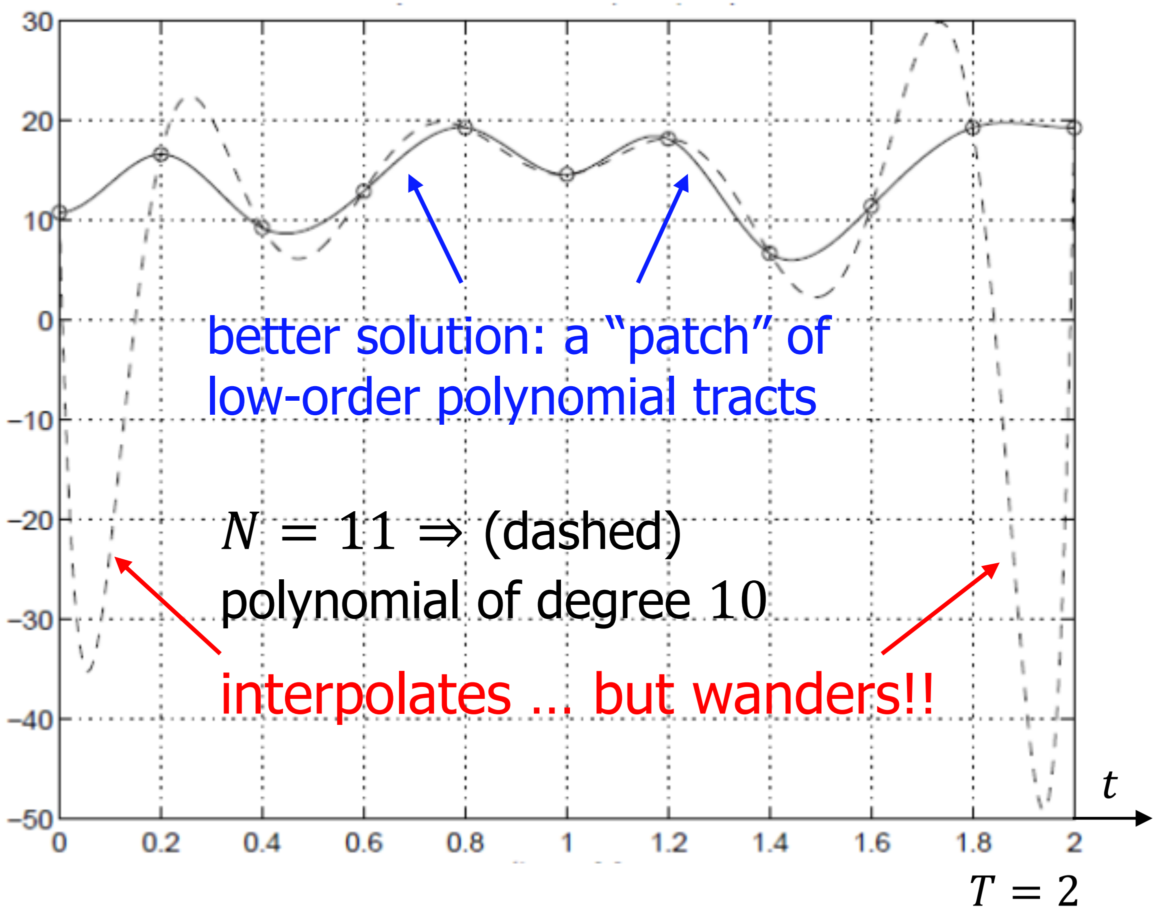
$N = 4 \;\Rightarrow\;$ a cubic

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

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

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

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



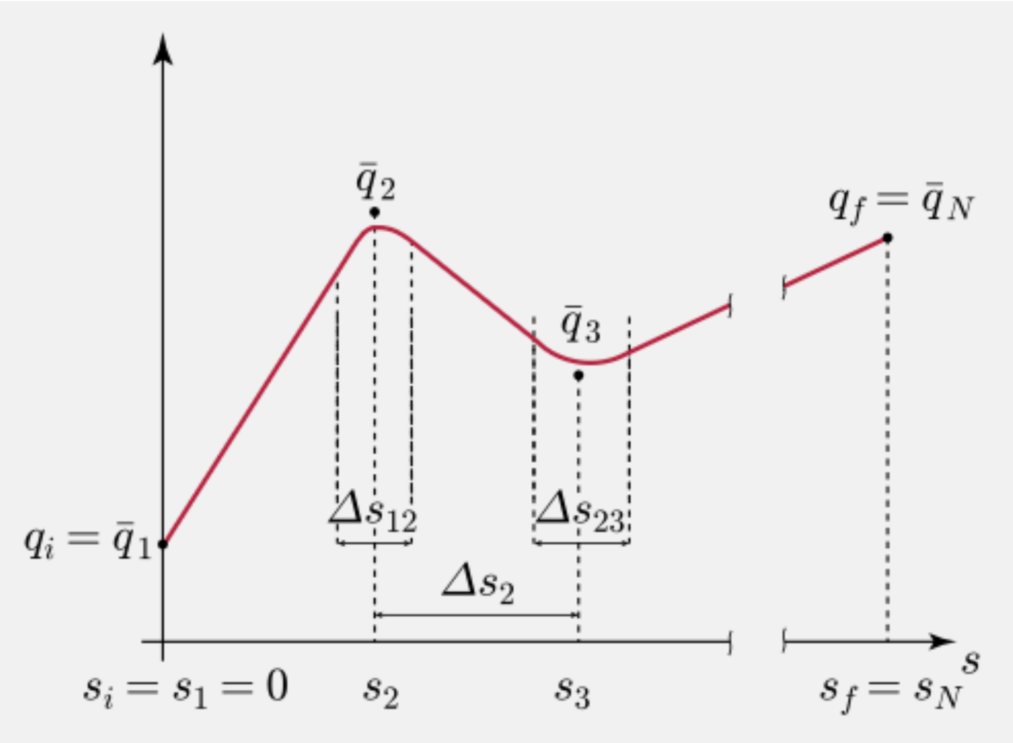
MP linear interpolation
with parabolic blends

- interpolate N knots using linear segments, with continuity of the tangent

$\{\bar{q}_1, \bar{q}_2, \dots, \bar{q}_N\}$ $\quad N$ knots

$q(s) = \{\theta_k(s), \text{for } s \in [s_k, s_{k+1}], k = 1, \dots, N-1\}$ $\quad N-1$ interpolating functions

- use quadratic polynomials that blend linear segments and overfly knots



$\Delta s_k = s_{k+1} - s_k$ $\quad$ intervals

$$\theta_k(s) = \bar{q}_k + (\bar{q}_{k+1} - \bar{q}_k)\frac{s - s_k}{\Delta s_k}$$

linear segments

$$\theta'_k = \frac{\bar{q}_{k+1} - \bar{q}_k}{\Delta s_k}$$

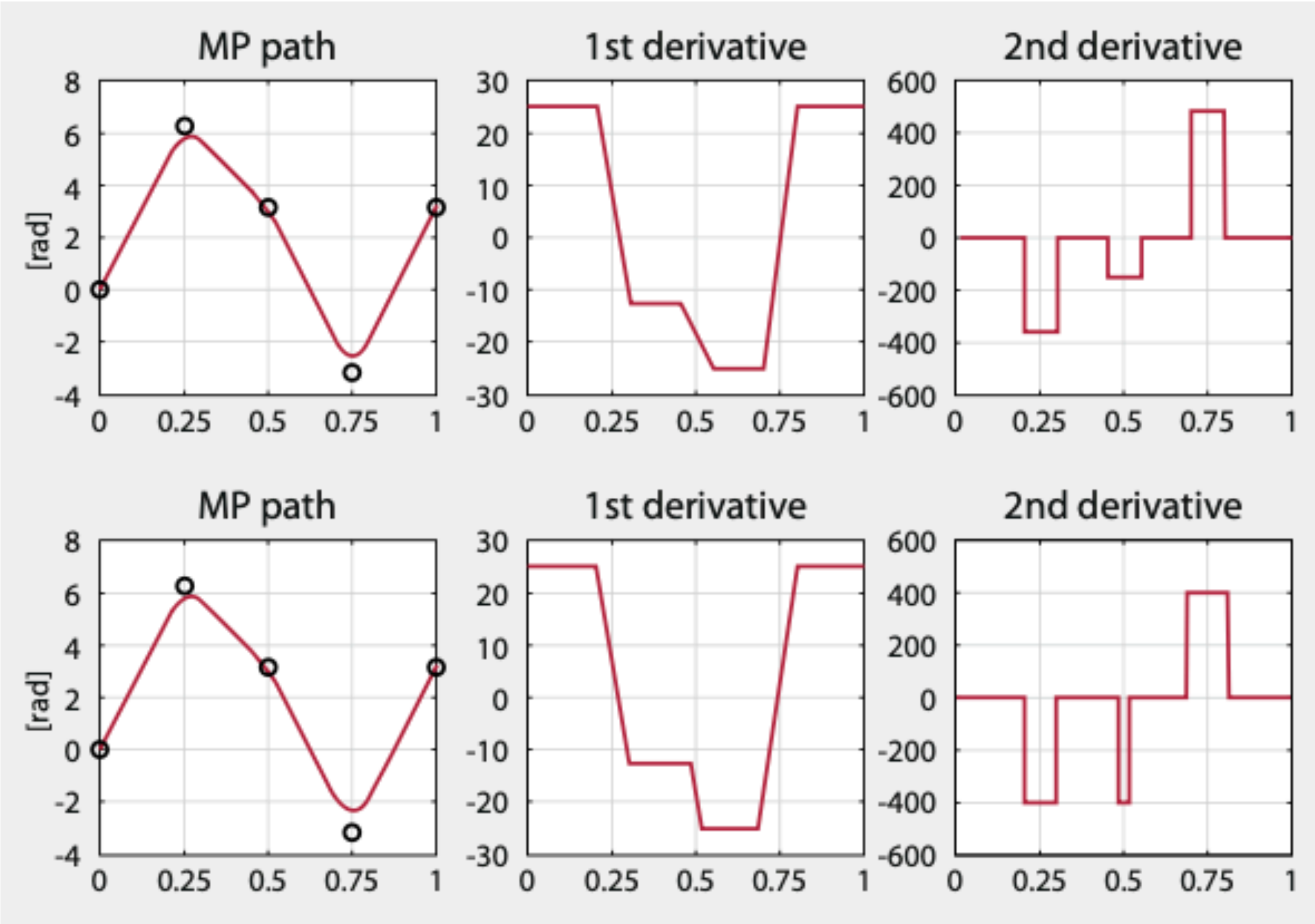
$\Delta s_{k-1,k}$ $\quad$ blending interval (at $\bar{q}_k$)

$$\theta''_k = \frac{\theta'_k - \theta'_{k-1}}{\Delta s_{k-1,k}} \Rightarrow \int \Rightarrow \int$$ quadratic function

blend with constant second derivative

MP linear interpolation
with parabolic blends

- $N = 5$ knots: $\bar{q}_1 = 0, \bar{q}_2 = 2\pi, \bar{q}_3 = \pi, \bar{q}_4 = -\pi, \bar{q}_5 = \pi$ [rad]
- at $s_1 = 0, s_2 = 0.25, s_3 = 0.5, s_4 = 0.75, s_5 = 1$ (equispaced)



$\Delta s_{k-1,k} = 0.2$

for all blending intervals

bounded

$|\theta_k''| \leq 400$ [rad]

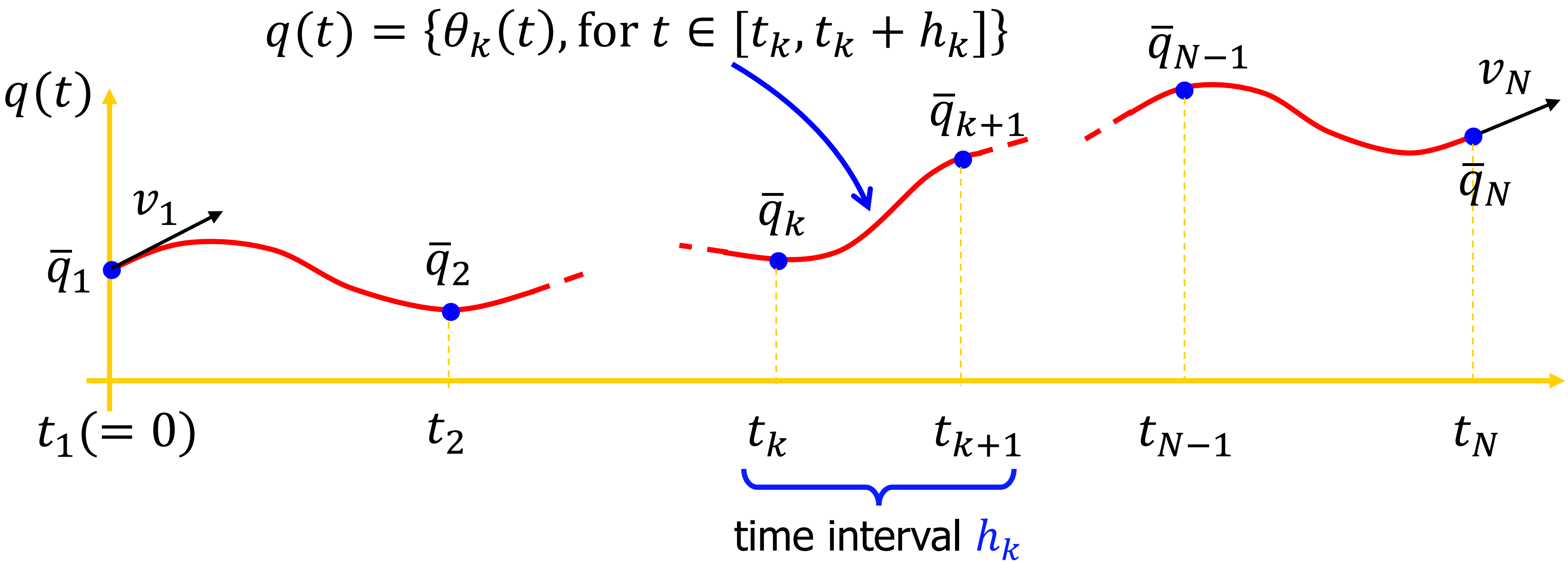
$\Downarrow$

$\Delta s_{k-1,k}$ accordingly

MP interpolation using splines

- problem
  interpolate N knots, with continuity up to the second derivative
- solution $\longleftrightarrow$ de Casteljau@Citroën, Bézier@Renault, de Boor@General Motors (late 1950) …
  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/final first derivatives $(q_1'/v_1, q_N'/v_N)$
- presented next in terms of time t, but similar in terms of space s
  - here: first derivative = velocity, second derivative = acceleration

Building a cubic spline



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

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

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

continuity conditions
for velocity and acceleration $\rightarrow$

$$\boxed{\begin{array}{ll} \dot{\theta}_k(h_k) = \dot{\theta}_{k+1}(0) & \\ & k = 1, \cdots, N-2 \\ \ddot{\theta}_k(h_k) = \ddot{\theta}_{k+1}(0) & \end{array}}$$

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

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

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

$$\begin{pmatrix} \\ \boldsymbol{A}(h_1, \cdots, h_{N-1}) \\ \\ \end{pmatrix} \begin{pmatrix} v_2 \\ v_3 \\ \vdots \\ \vdots \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} \vdots \\ \boldsymbol{b}(h_1, \cdots, h_{N-1}, \bar{q}_1 \cdots, \bar{q}_N, v_1, v_N) \\ \vdots \end{pmatrix}$$

tri-diagonal matrix always invertible

unknown

known vector

to be substituted then back in 1

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

$$\begin{pmatrix}
2(h_1+h_2) & h_1 & & & \\
h_3 & 2(h_2+h_3) & h_2 & & \\
 & \cdots & & & \\
 & & \cdots & & \\
 & & & \cdots & \\
 & & h_{N-2} & 2(h_{N-3}+h_{N-2}) & h_{N-3} \\
 & & & h_{N-1} & 2(h_{N-2}+h_{N-1})
\end{pmatrix}$$

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

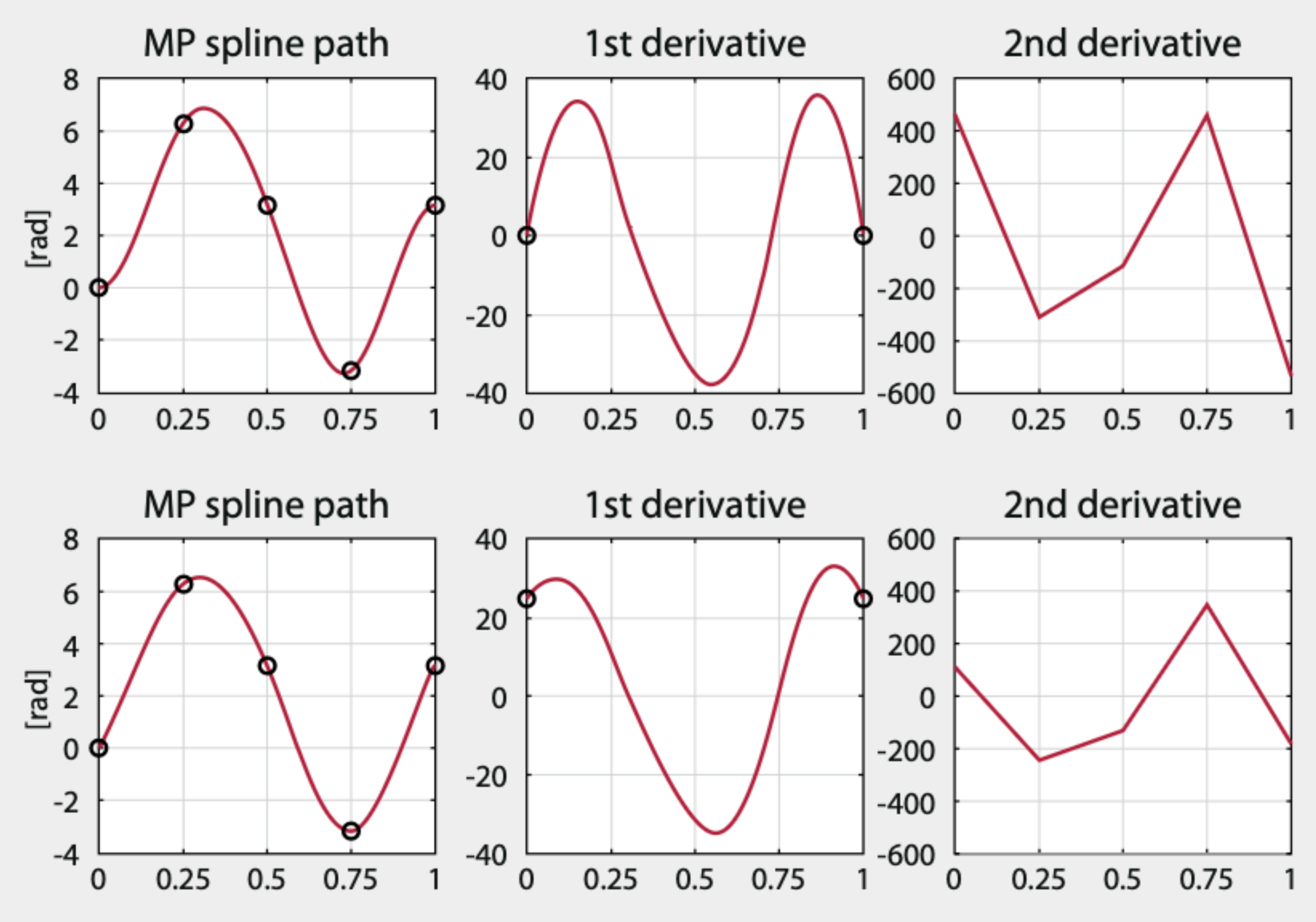
Structure of $b(\boldsymbol{h}, \boldsymbol{q}, v_1, v_N)$

$$
\begin{pmatrix}
\frac{3}{h_1 h_2}\left(h_1^2(\bar{q}_3 - \bar{q}_2) + h_2^2(\bar{q}_2 - \bar{q}_1)\right) - h_2 v_1 \\
\frac{3}{h_2 h_3}\left(h_2^2(\bar{q}_4 - \bar{q}_3) + h_3^2(\bar{q}_3 - \bar{q}_2)\right) \\
\vdots \\
\vdots \\
\frac{3}{h_{N-3} h_{N-2}}\left(h_{N-3}^2(\bar{q}_{N-1} - \bar{q}_{N-2}) + h_{N-2}^2(\bar{q}_{N-2} - \bar{q}_{N-3})\right) \\
\frac{3}{h_{N-2} h_{N-1}}\left(h_{N-2}^2(\bar{q}_N - \bar{q}_{N-1}) + h_{N-1}^2(\bar{q}_{N-1} - \bar{q}_{N-2})\right) - h_{N-2} v_N
\end{pmatrix}
$$

Spline interpolation

numerical example in space

- $N = 5$ knots: $\bar{q}_i = \bar{q}_1 = 0, \bar{q}_2 = 2\pi, \bar{q}_3 = \pi, \bar{q}_4 = -\pi, \bar{q}_f = \bar{q}_5 = \pi$ [rad]
- at $s_1 = 0, s_2 = 0.25, s_3 = 0.5, s_4 = 0.75, s_5 = 1$ (equispaced)



$$q'_i = q'_f = 0$$

as boundary conditions

$$q'_i = \frac{\bar{q}_2 - \bar{q}_1}{s_2 - s_1}$$

$$q'_f = \frac{\bar{q}_N - \bar{q}_{N-1}}{s_N - s_{N-1}}$$

as boundary conditions

Properties of splines

- a natural spline in space ($q_i'' = 0$, $q_f'' = 0$) has the minimum curvature among all interpolating functions with continuous second derivative
- for cyclic tasks ($\bar{q}_1 = \bar{q}_N$), 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 acceleration (when in space, up to the second derivative)
  - 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 $\bar{q}_1, \cdots, \bar{q}_N$, $h_1, \cdots, h_{N-1}, v_1, v_N$
- 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 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

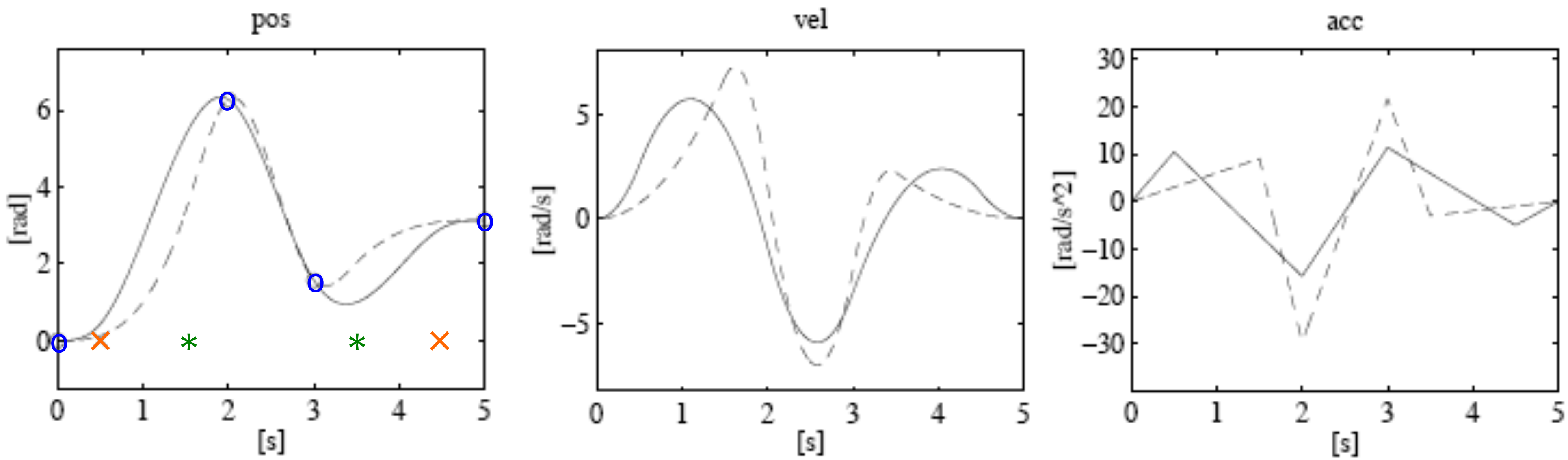
- two more parameters are needed in order to impose also the initial acceleration a_1 and final acceleration a_N
- two "virtual knots" are inserted in the first and in the last original intervals, increasing the number of cubic polynomials from $N-1$ to $N+1$
- in the two virtual knots only continuity conditions on position, velocity and acceleration are imposed (i.e., no extra values!)

  $\Rightarrow$ 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 …

see textbook Sect. 4.3.2 (pp. 210-212) and Problem 4.8

Spline interpolation
numerical example in time with b.c. on acceleration

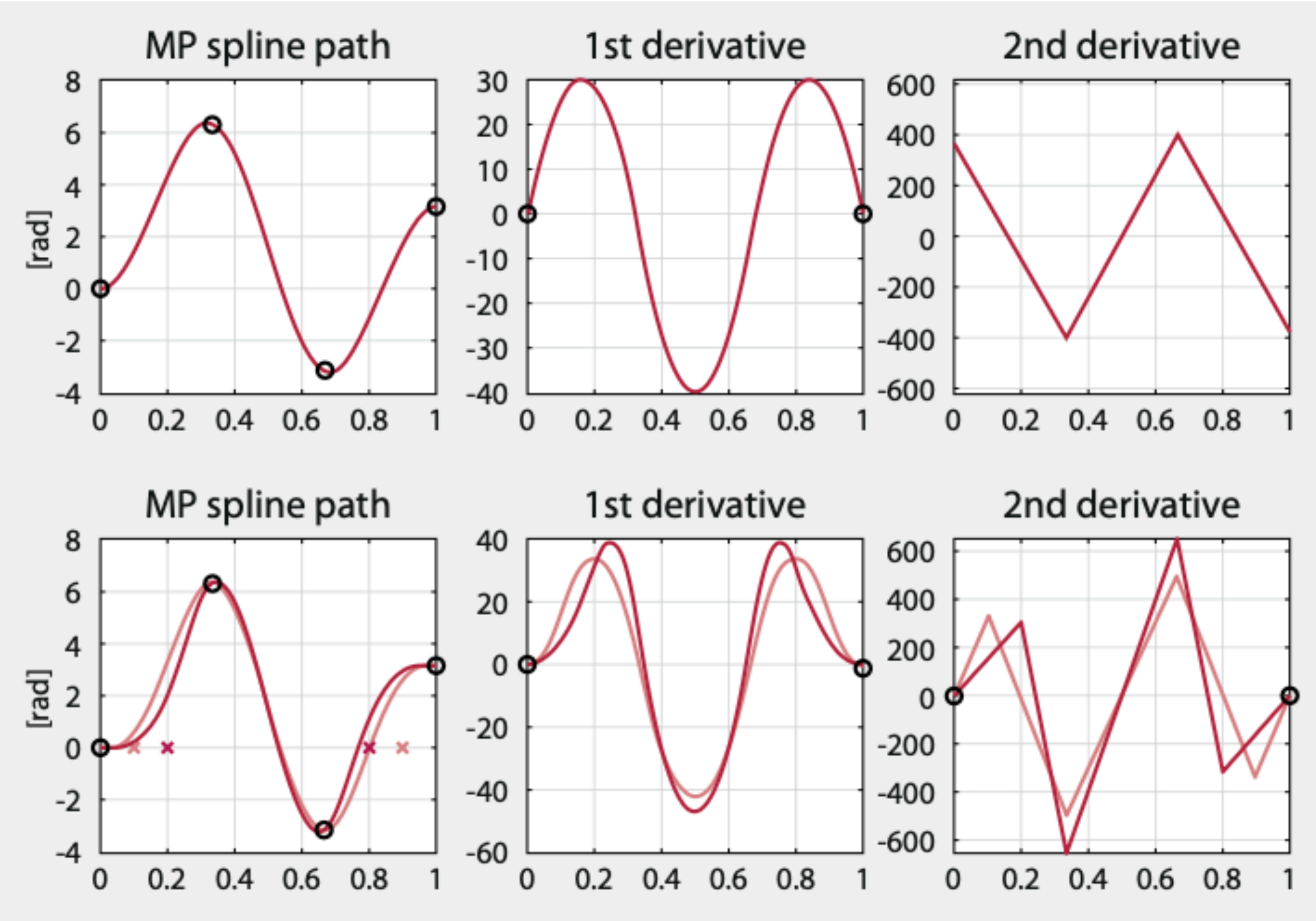
- $N = 4$ knots (o) $\Rightarrow$ 3 cubic polynomials
  - joint values $\bar{q}_1 = 0,\ \bar{q}_2 = 2\pi,\ \bar{q}_3 = \pi/2,\ \bar{q}_4 = \pi$ [rad]
  - at $t_1 = 0, t_2 = 2, t_3 = 3, t_4 = 5 \Rightarrow h_1 = 2, h_2 = 1, h_3 = 2$ [s]
  - boundary velocities $v_1 = v_4 = 0$ [rad/s]
- 2 added knots to impose accelerations at both ends (5 cubic polynomials)
  - boundary accelerations $a_1 = a_4 = 0$ [rad/s^2]
  - two placements: at $t_1' = 0.5$ and $t_3' = 4.5$ (×); or at $t_1'' = 1.5$ and $t_4'' = 3.5$ ($*$)



——— = placement (×) - - - - - - = placement ($*$)

Spline interpolation
numerical example in space with b.c. on curvature

- $N = 4$ knots: $\bar{q}_1 = 0, \bar{q}_2 = 2\pi, \bar{q}_3 = -\pi, \bar{q}_4 = \pi$ [rad]
- at $s_1 = 0, s_2 = 1/3, s_3 = 2/3, s_4 = 1$ (equispaced)



$q_i' = q_f' = 0$

NO b.c. on q_i'', q_f''

$q_i' = q_f' = 0$

$q_i'' = q_f'' = 0$

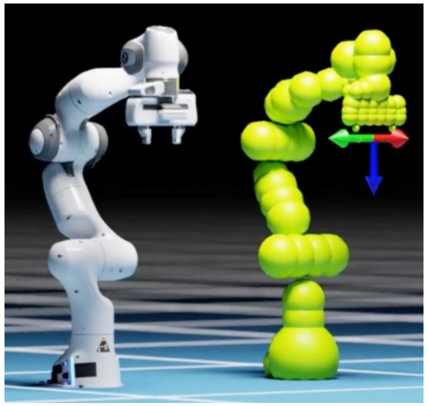
TWO choices for the virtual knots

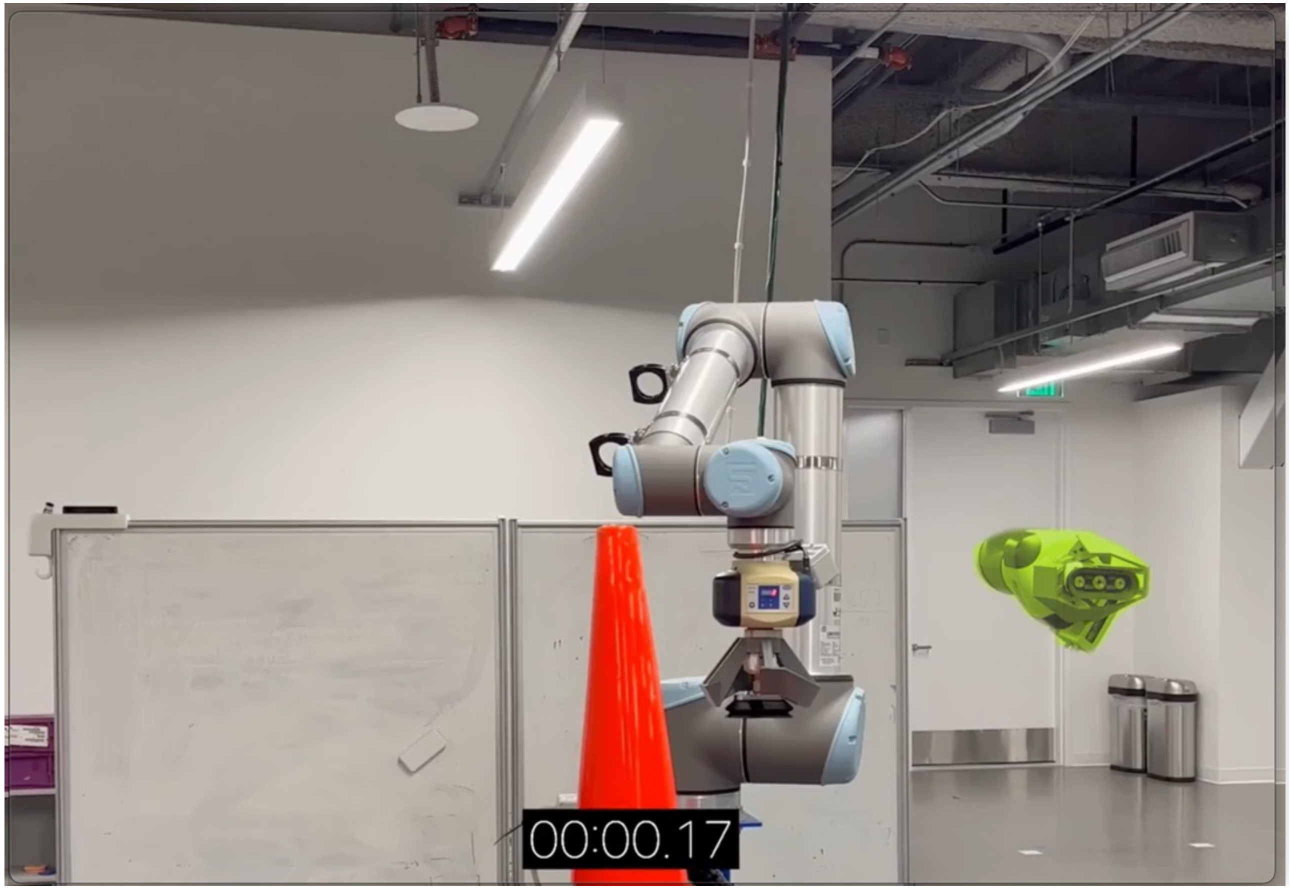
$s_a = 0.1, \; s_b = 0.9$

$s_a = 0.2, \; s_b = 0.8$

Point-to-point optimal motion
computation in real time

- point-to-point motion without prescribed interpolation path (infinite feasible trajectories …)
- in the presence of obstacles (robot modeled with "bubbles")
- optimization algorithm penalizes jerk and acceleration, leading to smooth and short trajectories
- real-time computation (100 ms) with a CUDA (Compute Unified Device Architecture) library using NVIDIA parallel GPUs





video: see https://curobo.org/index.html#overview

library content:
forward kinematics (URDF), numerical inverse kinematics (L-BFGS), collision checking, motion generation, model predictive control