Control of industrial robots

Advanced motion planning

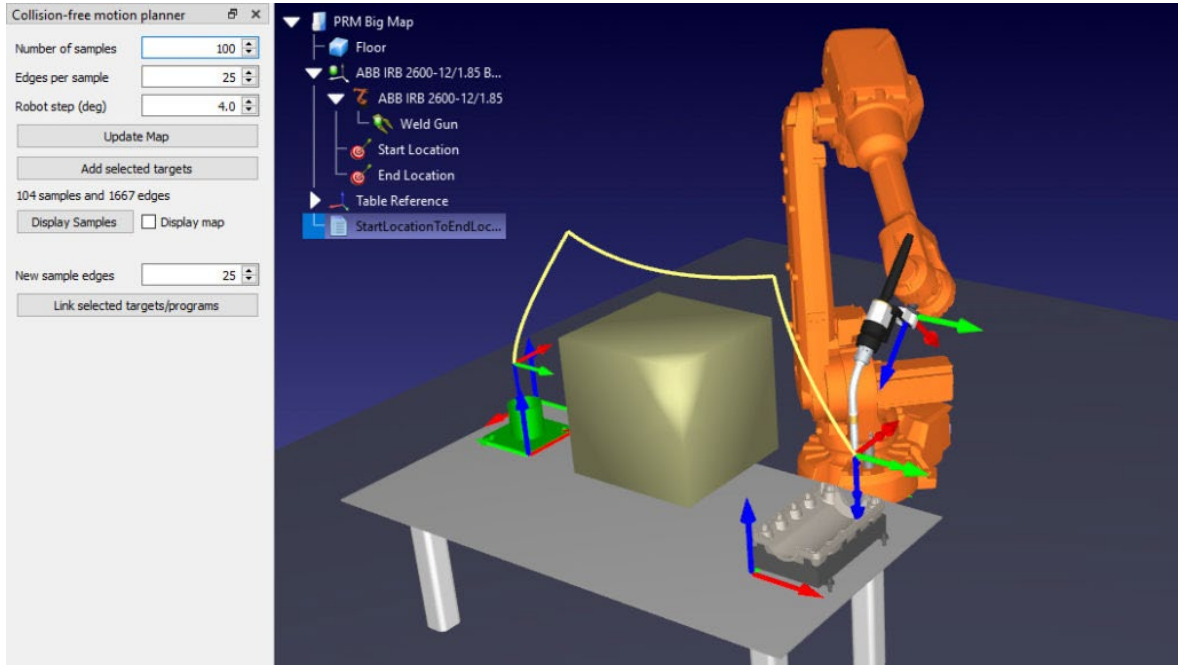
Prof. Paolo Rocco (paolo.rocco@polimi.it)

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria

Advanced motion planning

- After reviewing the basic elements of motion planning, we will now address some more advanced topics.

- Specifically, we will discuss the following problems:
  - path planning with obstacle avoidance
  - alternative time laws (other than polynomial ones)
  - kinematic and dynamic scaling of trajectories
  - interpolation of points

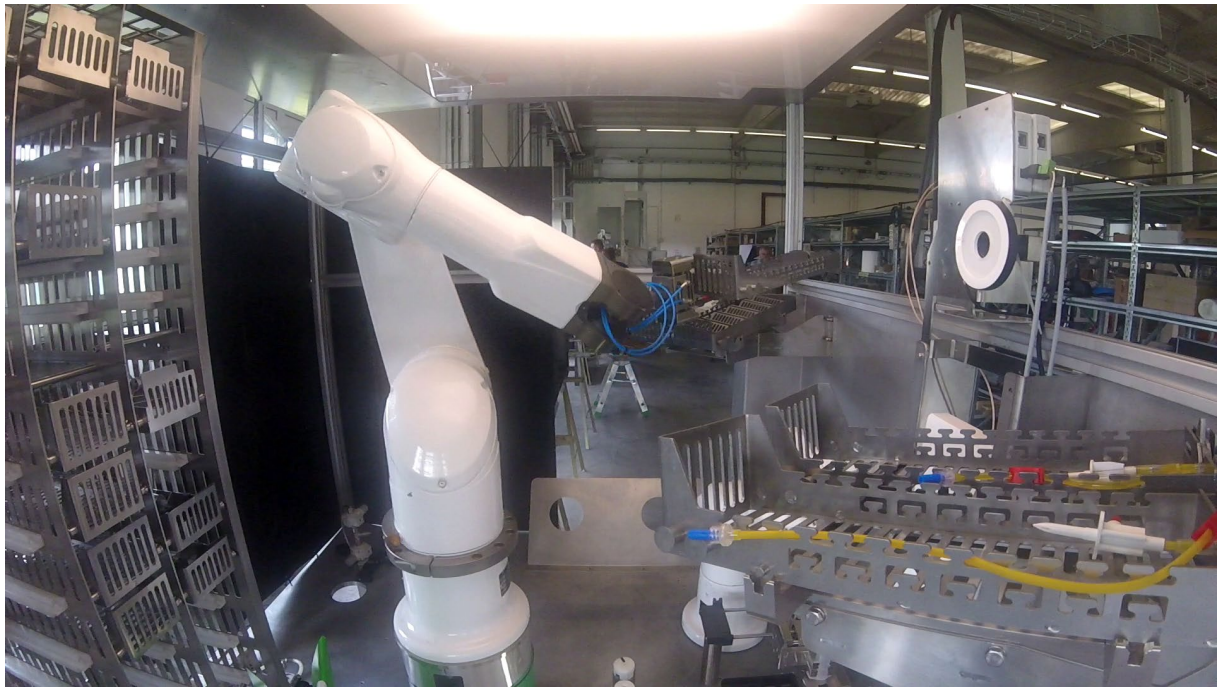


Source: RoboDK

Path planning with obstacle avoidance

- If the robot moves in a cluttered environment, the definition of the path might be troublesome
- In fact we need a **path which is collision free**: such path, at the current stage of robotics practice, is generated manually by the programmer as a sequence of motion commands

- An active research line consists in **automatic path planning**, i.e. in finding an algorithm that autonomously generate the geometric path, given the kinematics of the robot and the known positions and shapes of the obstacles

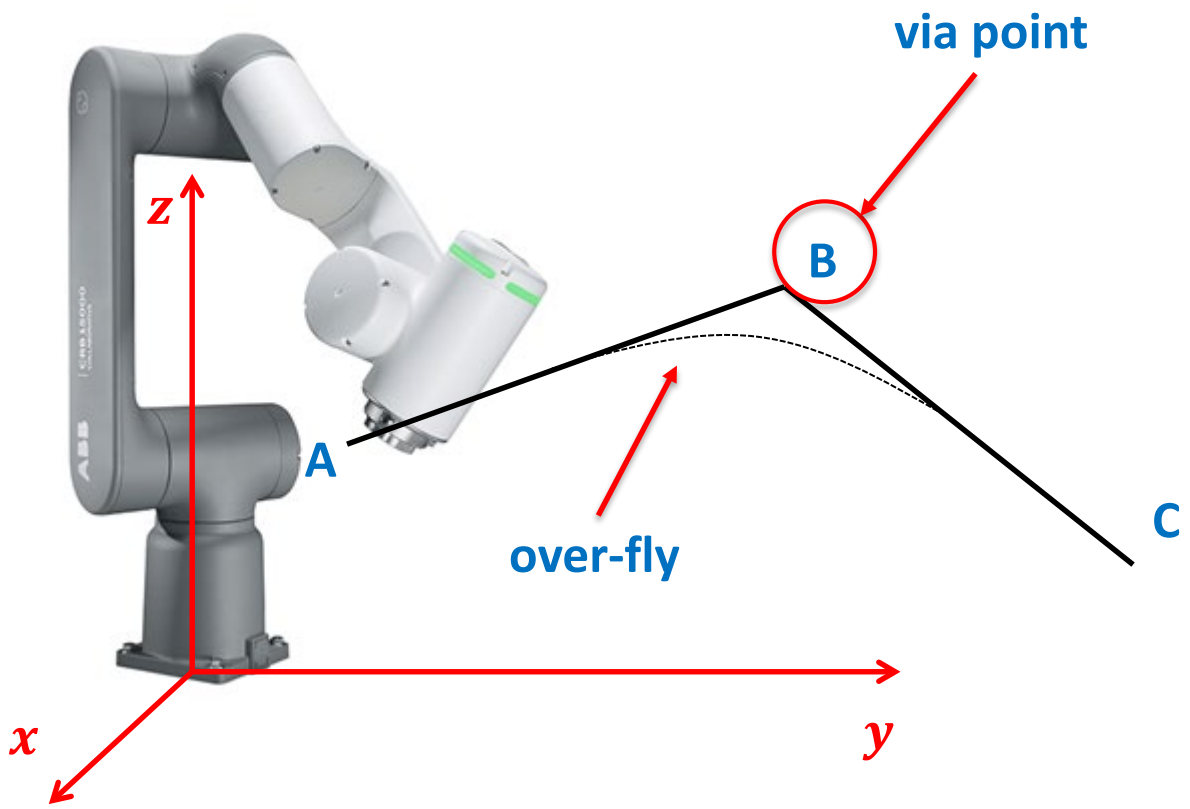


Concatenation of linear paths

A common practice in robot programming is to concatenate linear paths, and this is particularly used to avoid obstacles.

The intermediate point between two consecutive segments can be considered as a **via point**, meaning that there is no need to pass and stop there.

During the **over-fly**, i.e. the passage near a via point, the path remains always in the plane specified by the two lines intersecting in the via point. This means that the problem of planning the over-fly is planar.



Concatenation of linear paths

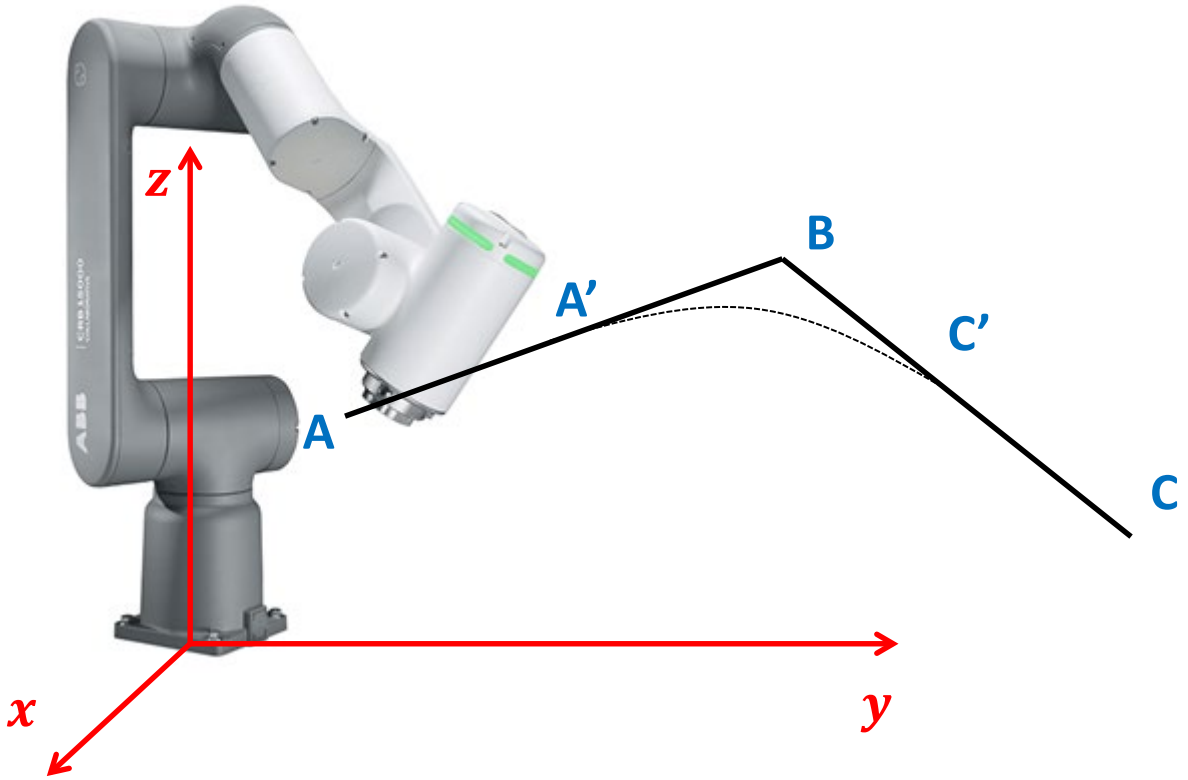
The problem of finding a good blending between two paths can be set in time domain.

Given:

- a constant velocity v_1 in the first path
- a constant velocity v_2 in the second path

find the desired transition for a time ΔT with constant acceleration.

We will call A' the point where the blending starts and C' the point where it ends.



Concatenation of linear paths

Let $\boldsymbol{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$ be the generic point along the blending,

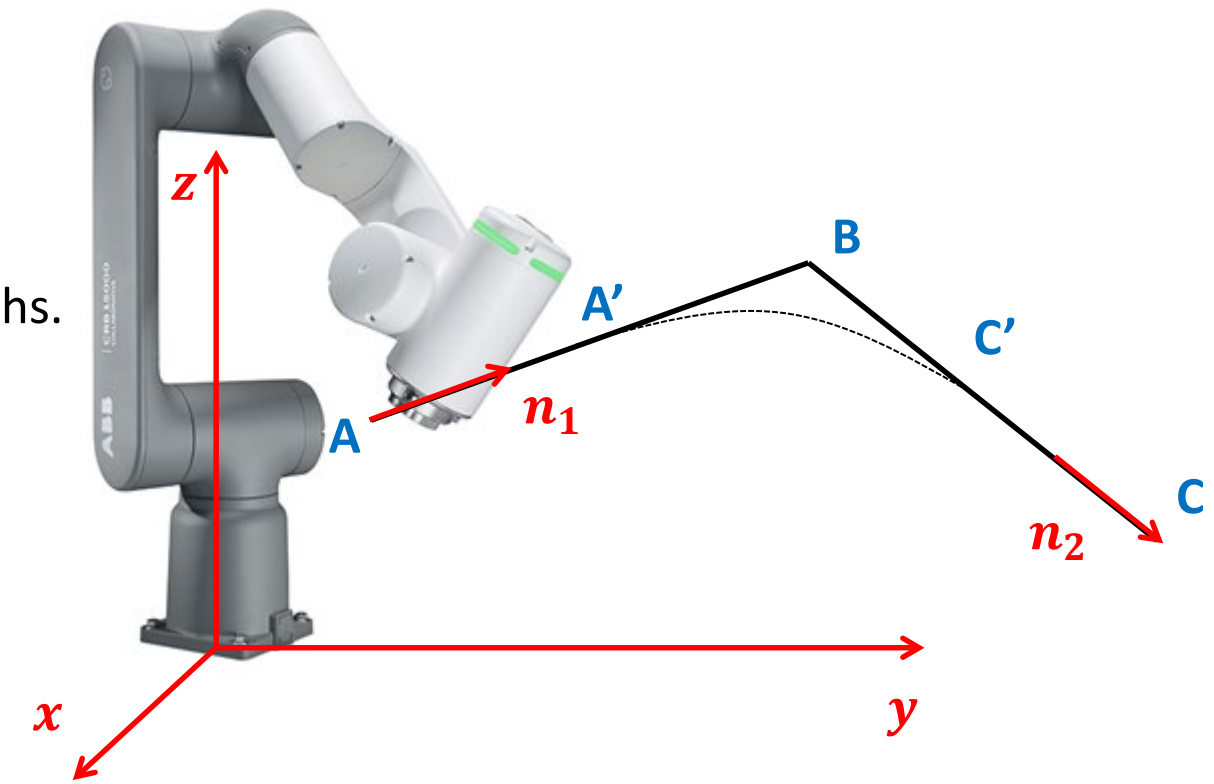
$\boldsymbol{n}_1 = \frac{B-A}{\|B-A\|}$, $\boldsymbol{n}_2 = \frac{C-B}{\|C-B\|}$ the unit vectors along the two paths.

Assuming that the transition starts at $t = 0$:

$\ddot{\boldsymbol{p}}(t) = (v_2\boldsymbol{n}_2 - v_1\boldsymbol{n}_1)/\Delta T$ → integrating →

$\dot{\boldsymbol{p}}(t) = v_1\boldsymbol{n}_1 + (v_2\boldsymbol{n}_2 - v_1\boldsymbol{n}_1)t/\Delta T$ → integrating →

$\boldsymbol{p}(t) = \boldsymbol{A}' + v_1\boldsymbol{n}_1 t + (v_2\boldsymbol{n}_2 - v_1\boldsymbol{n}_1)t^2/(2\Delta T)$ **parabolic blending**



Concatenation of linear paths

We can now find the final solution, defining the distances d_1 and d_2 such that:

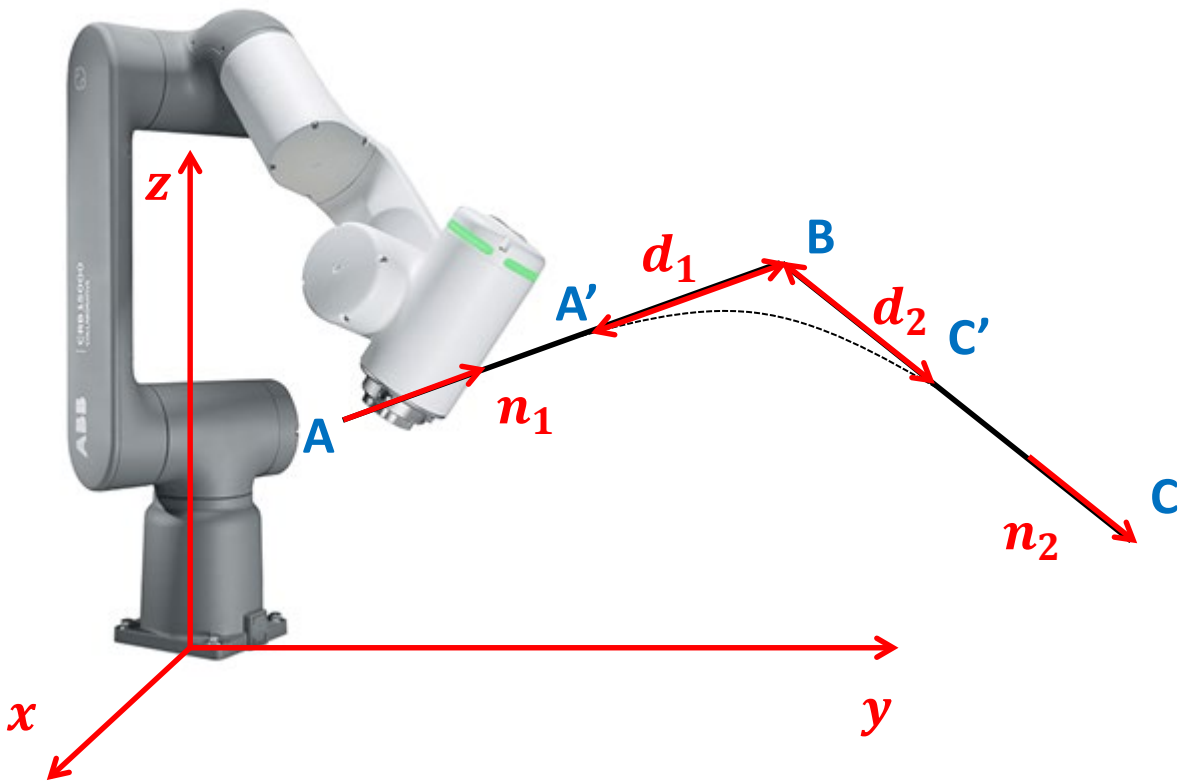
$$\boldsymbol{B} - \boldsymbol{A}' = d_1 \boldsymbol{n}_1 \qquad \boldsymbol{C}' - \boldsymbol{B} = d_2 \boldsymbol{n}_2$$

then:

$$\boldsymbol{p}(\Delta T) = \boldsymbol{C}' = \boldsymbol{A}' + (v_1 \boldsymbol{n}_1 + v_2 \boldsymbol{n}_2)\, \Delta T/2 \longrightarrow$$

$$\boldsymbol{C}' - \boldsymbol{B} = \boldsymbol{A}' - \boldsymbol{B} + (v_1 \boldsymbol{n}_1 + v_2 \boldsymbol{n}_2)\, \Delta T/2 \longrightarrow$$

$$d_1 \boldsymbol{n}_1 + d_2 \boldsymbol{n}_2 = (v_1 \boldsymbol{n}_1 + v_2 \boldsymbol{n}_2)\, \Delta T/2$$



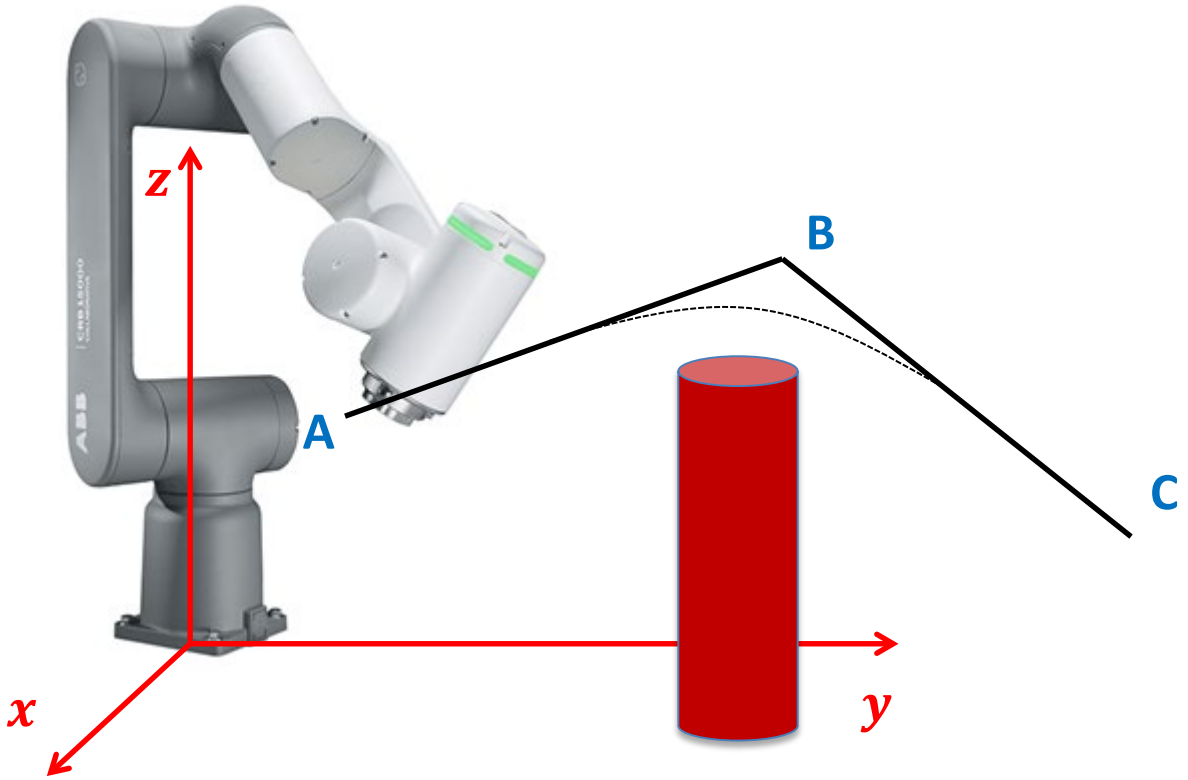
$$d_1 = v_1\, \Delta T/2 \qquad d_2 = v_2\, \Delta T/2$$

If we choose d_1 (i.e. A'): $\Delta T = 2 d_1 / v_1 \qquad d_2 = d_1\, v_2 / v_1$

Obstacle avoidance

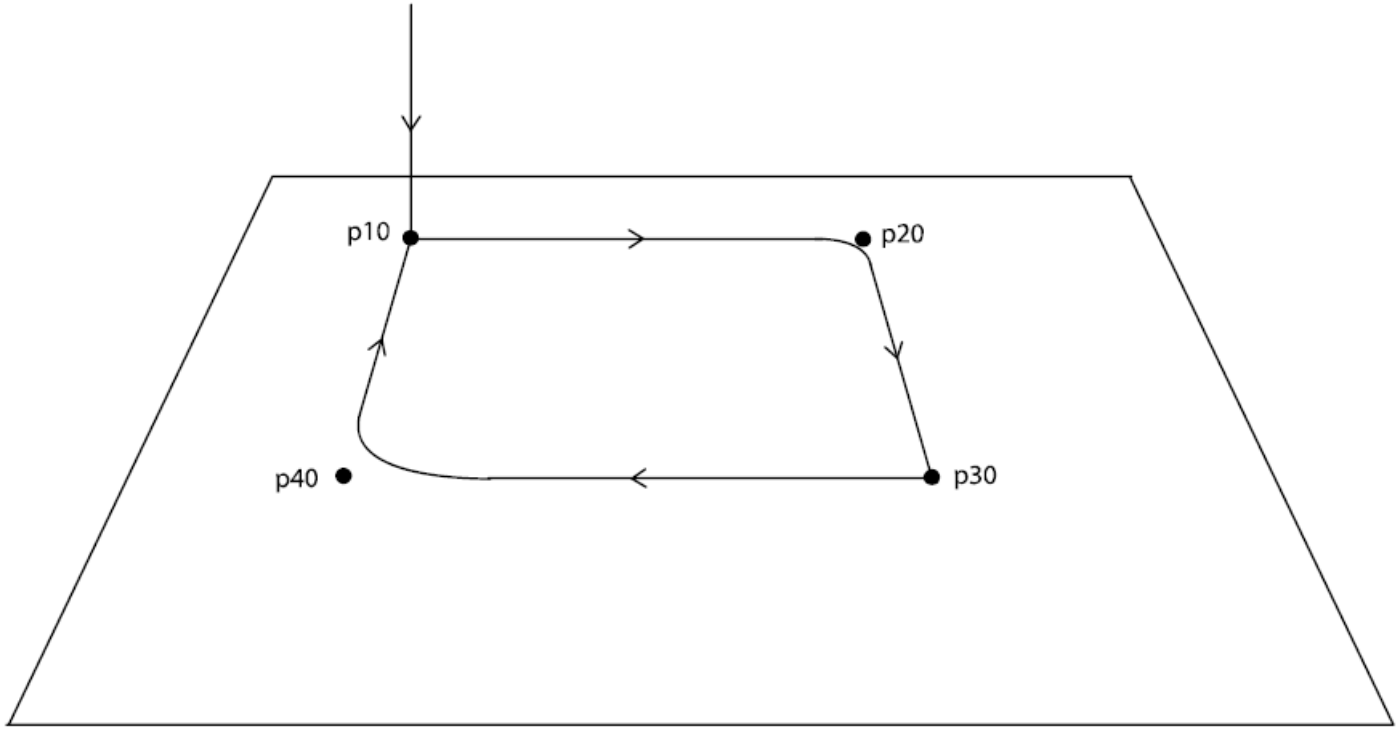
As already mentioned, via points can be used to **avoid obstacles**.

The end-effector has to move from A to C, however there is an obstacle in between. A via point B is then introduced.



Via points in robot programming languages

Robot manufacturers in some cases use **circular blending** rather than parabolic ones and have specific clauses in motion commands to define the radius of the blending.
For example with ABB:

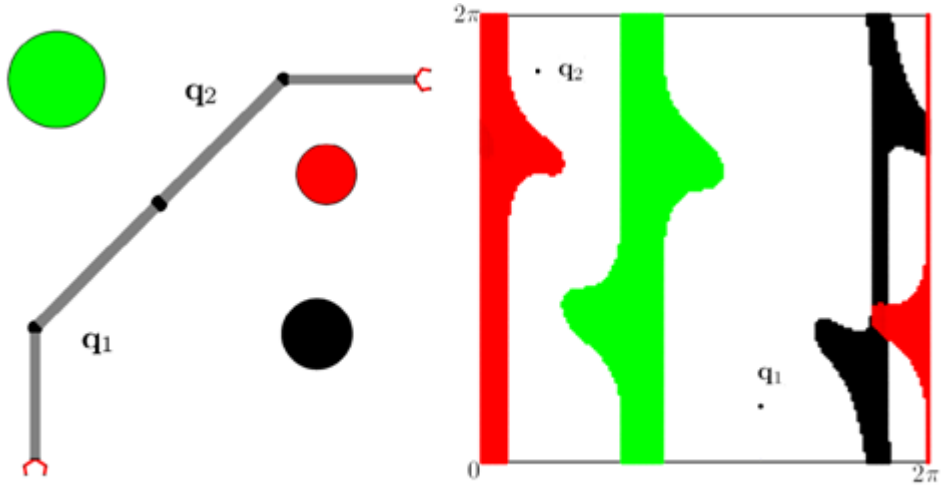



```
MoveL p10, v200, fine;
MoveL p20, v200, z20;
MoveL p30, v200, fine;
MoveL p40, v200, z50;
MoveL p10, v200, fine;
```

`fine:`	no blending
`z20:`	circular blending of 20 mm
`z50:`	circular blending of 50 mm

Configuration space

- Moving now to the automatic path planning, the problem is usually addressed in the **configuration space** (C-space), where the robot is at each time instant represented as a mobile point
- For an articulated robot, the common choice of the configuration space is the space of the joint variables (**joint space**)
- **Obstacles** are mapped from the workspace to the configuration space: in case of an articulated manipulator they take complicated shapes

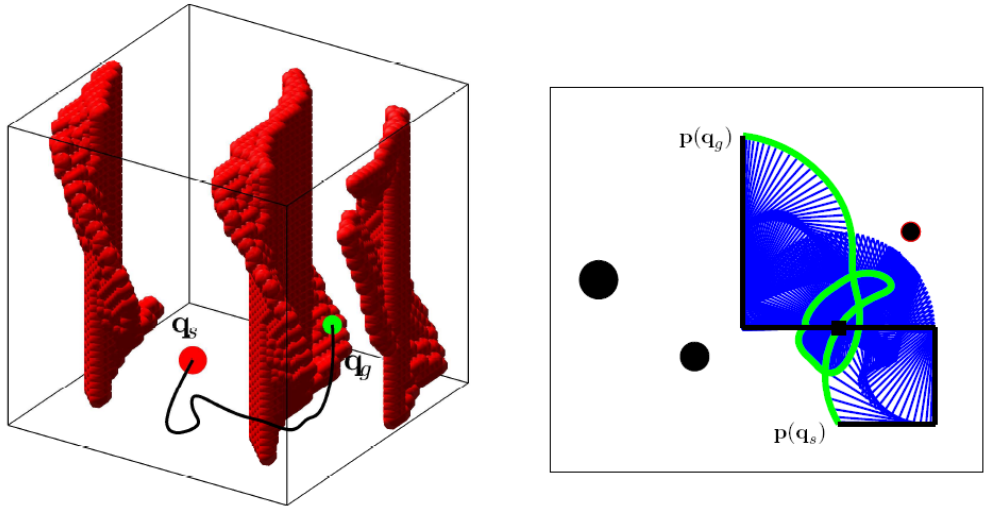


Workspace C-space

We call C_{free} the subset of C-space that does not cause collisions with the obstacles.
A path in C-space is **free** if it is entirely contained in C_{free}

Setting the problem

The path planning problem can be set as follows in general:



C-space Workspace

Assume that the initial and final posture of the robot in the Workspace are mapped into corresponding configurations in C-space, respectively a start configuration $\mathbf{q}_S$ and a goal configuration $\mathbf{q}_G$.

Planning a collision-free motion for the robot means to generate a path from $\mathbf{q}_S$ to $\mathbf{q}_G$ if they belong to the same connected component of C_{free}, and to report a failure otherwise.

Probabilistic planning

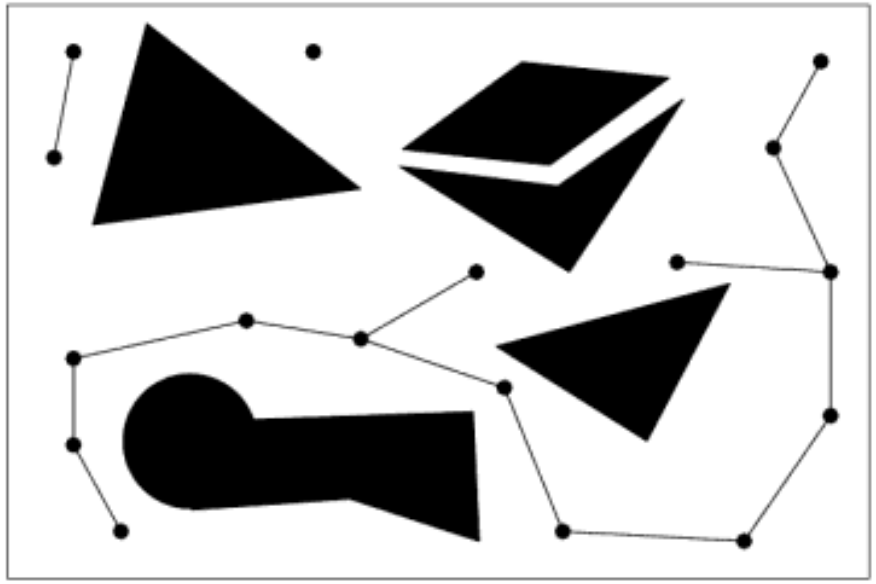
- The exact planning algorithms that are based on the a priori knowledge of the complete C-space are exponential in the dimensionality of the C-space and thus hardly applicable in practice
- **Probabilistic planners** are instead a class of methods of remarkable efficiency, especially for high-dimension configuration spaces
- They belong to the general family of **sampling-based methods**, where the basic idea is to **randomly** select a finite-set of collision-free configurations that adequately represent how C_{free} is connected, and using these configurations to build a roadmap
- At each iteration a **sample configuration** is chosen and it is checked whether it entails a collision between the robot and the obstacles: if yes, the configuration is discarded, otherwise it is added to the current roadmap
- Two versions of such randomized sampling-based approach are:
  - PRM (Probabilistic Roadmap)
  - RRT (Rapidly-exploring Random Tree)

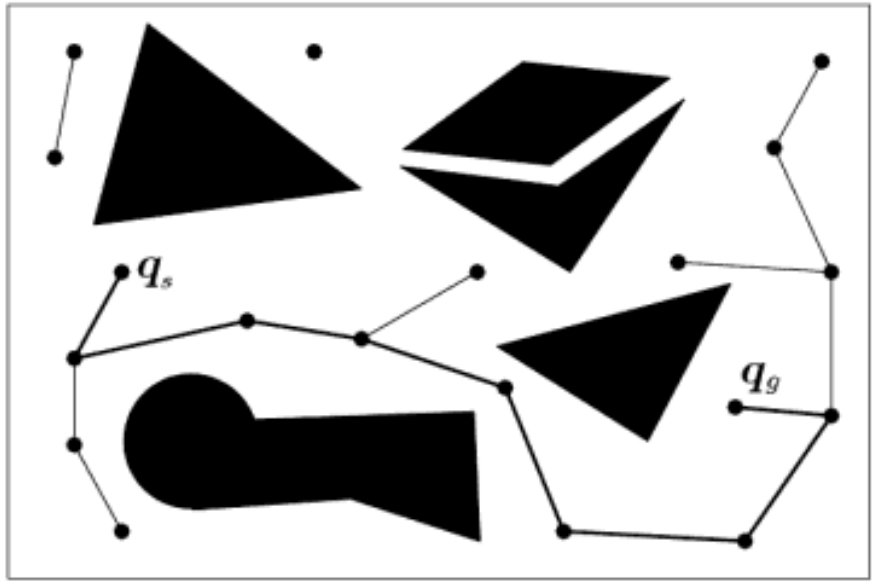
With these methods there is no need to know the shape of the obstacles in C-space!

PRM (Probabilistic roadmap)

- a random sample $\mathbf{q}_{\mathrm{rand}}$ of the C-space is selected using a uniform probability distribution and tested for collision
- if $\mathbf{q}_{\mathrm{rand}}$ does not cause collisions it is added to a roadmap which is progressively being formed and connected (if possible) through free local paths to sufficiently “near” configurations already in the roadmap
- the generation of a free local path between $\mathbf{q}_{\mathrm{rand}}$ and a near configuration $\mathbf{q}_{\mathrm{near}}$ is made by a procedure called local planner
- the iterations terminate when either a maximum number of iterations has been reached or the number of connected components in the roadmap becomes smaller than a given threshold
- then we try to verify whether the path panning problem can be satisfied by connecting $\mathbf{q}_{\mathrm{S}}$ and $\mathbf{q}_{\mathrm{G}}$ to the same connected component of the PRM by free local paths

PRM (Probabilistic roadmap)





This picture is taken from the textbook:

B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo:
Robotics: Modelling, Planning and Control, 3rd Ed.
Springer, 2009

PRM (Probabilistic roadmap)

Advantages of PRM:

- it is **remarkably fast** in finding solutions to motion planning problems
- as already mentioned, there is **no need to generate the obstacles in C-space**

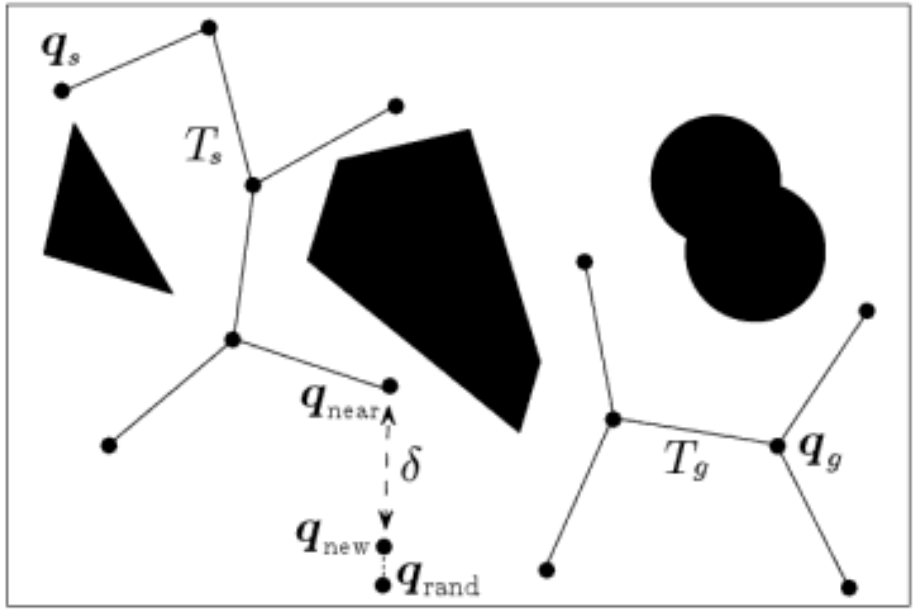
Disadvantages of PRM:

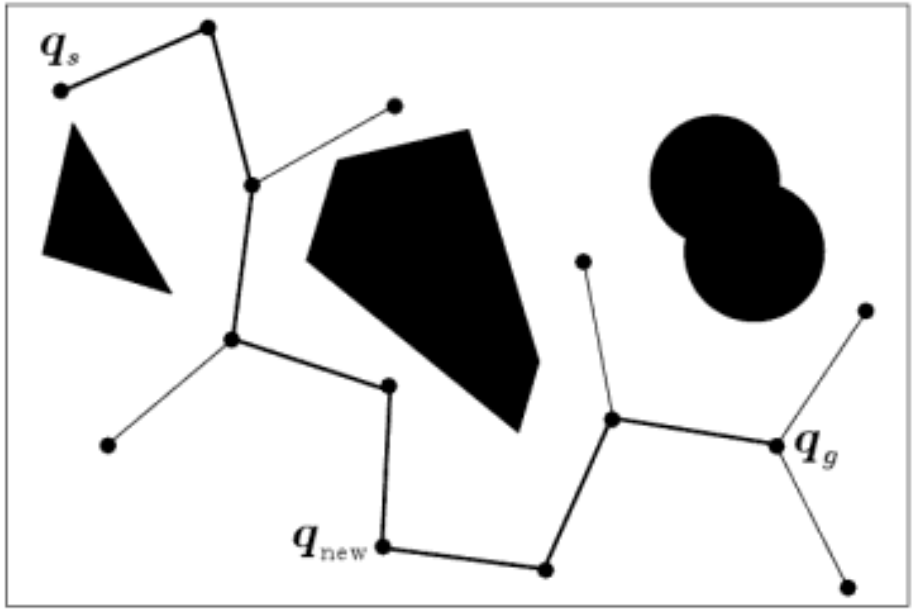
- it is **only probabilistically complete**: the probability to find a solution to the planning problem, if it exists, tends to 1 as the execution time tends to infinity

RRT (Rapidly-exploring random trees)

- the method makes use of a data structure called **tree**
- a **random sample** $\mathbf{q}_{\mathrm{rand}}$ of the C-space is selected using a uniform probability distribution
- the configuration $\mathbf{q}_{\mathrm{near}}$ in the tree *T* (which is progressively formed) that is the closest one to $\mathbf{q}_{\mathrm{rand}}$ is found and a new candidate configuration $\mathbf{q}_{\mathrm{new}}$ is produced on the segment joining $\mathbf{q}_{\mathrm{near}}$ to $\mathbf{q}_{\mathrm{rand}}$ at a predefined distance δ from $\mathbf{q}_{\mathrm{near}}$
- it is checked that both $\mathbf{q}_{\mathrm{new}}$ and the segment joining $\mathbf{q}_{\mathrm{near}}$ to $\mathbf{q}_{\mathrm{new}}$ belong to C_{free}
- if this is true, the tree *T* is expanded by incorporating $\mathbf{q}_{\mathrm{new}}$ and the said segment
- to speed up the search, **two trees** are **expanded**, rooted at $\mathbf{q}_{\mathrm{S}}$ and $\mathbf{q}_{\mathrm{G}}$, respectively
- at each iteration, both trees are expanded with the randomized procedure
- after a certain number of expansion steps, the algorithm tries to **connect** the two trees and thus to complete the path

RRT (Rapidly-exploring random trees)



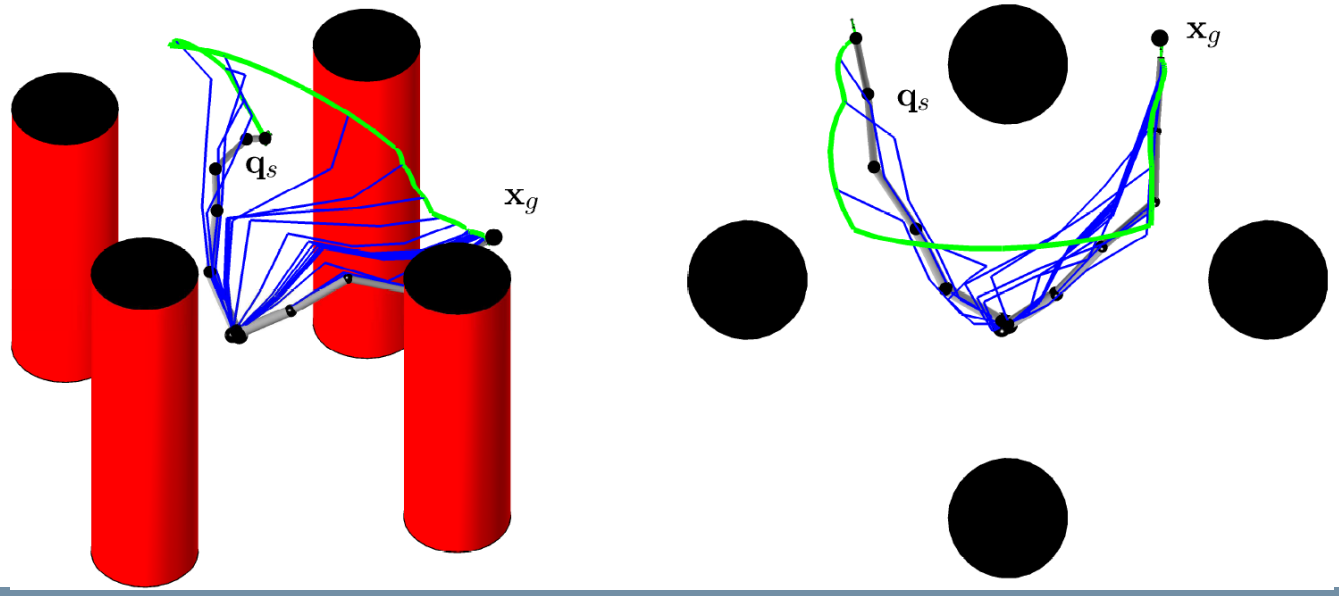


This picture is taken from the textbook:

B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo: *Robotics: Modelling, Planning and Control, 3rd Ed.*
Springer, 2009

RRT (Rapidly-exploring random trees)

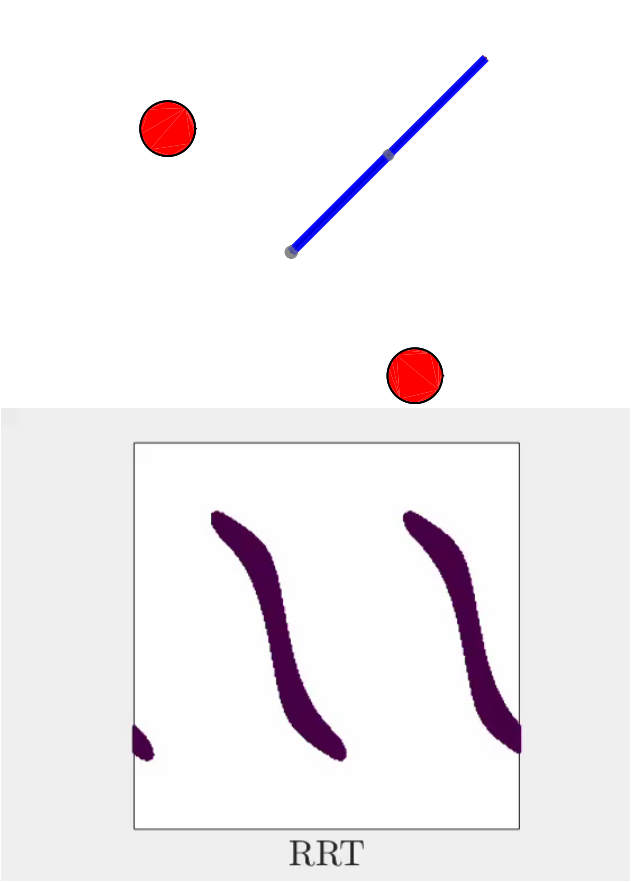
- the RRT expansion technique, though simple, results in a **very efficient** exploration of the C-space, since the procedure for generating new candidate configurations is intrinsically biased towards regions in C_{free} that have not been visited yet
- as in the PRM method, there is **no need to generate the obstacles in C-space**
- like the PRM method, it is **only probabilistically complete**



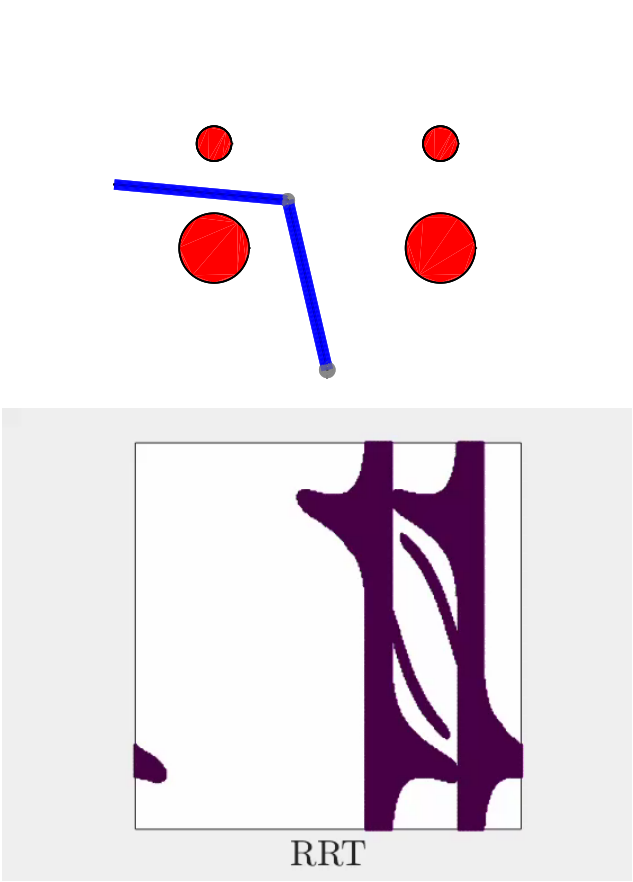
Example of path generated with a RRT algorithm

RRT (Rapidly-exploring random trees)

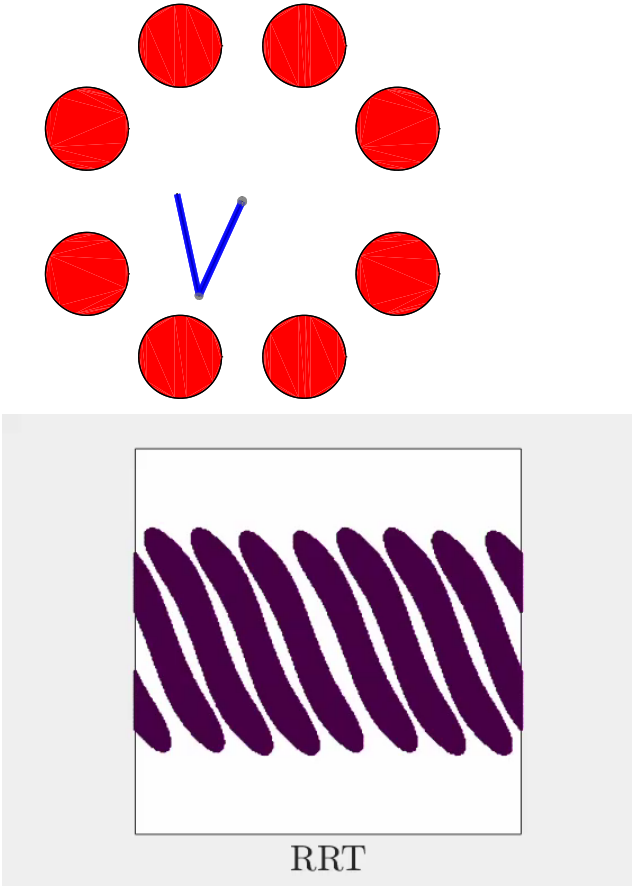
Ample free space



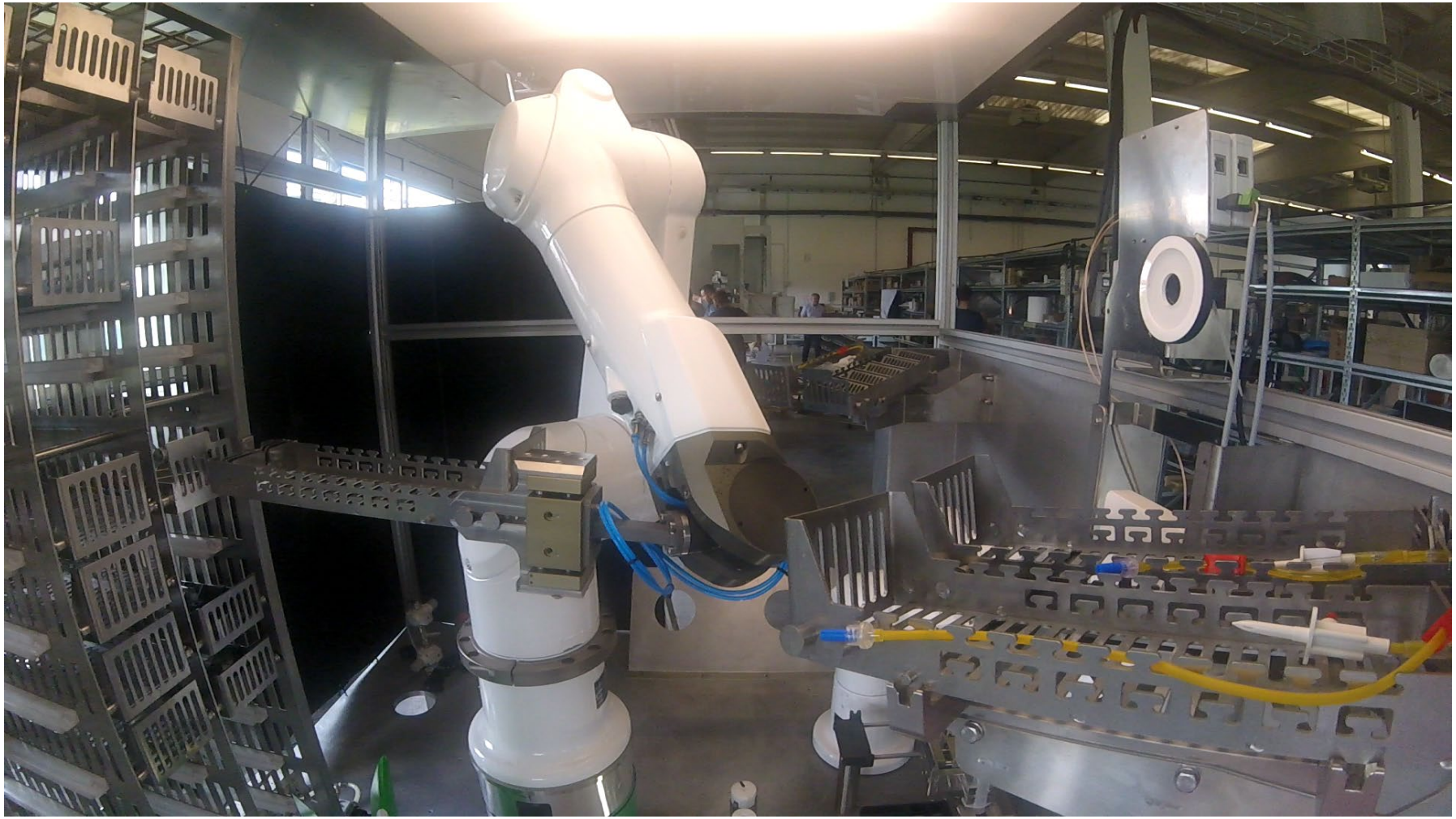
Cluttered space



Narrow passages



RRT (Rapidly-exploring random trees)



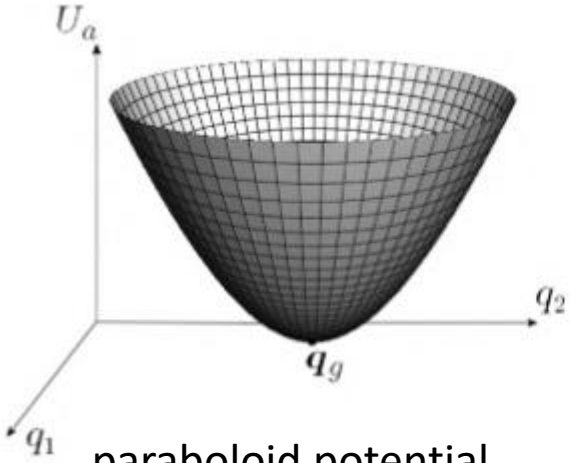
Artificial potentials

- in the artificial potential method, the motion of the point that represents the robot in C-space is influenced by a **potential field** U
- this field is obtained as the superposition of an **attractive** potential to the goal and a **repulsive** potential from the obstacles.
- at each configuration $\mathbf{q}$ the artificial force generated by the potential is defined as the negative gradient $-\nabla U(\mathbf{q})$ of the potential

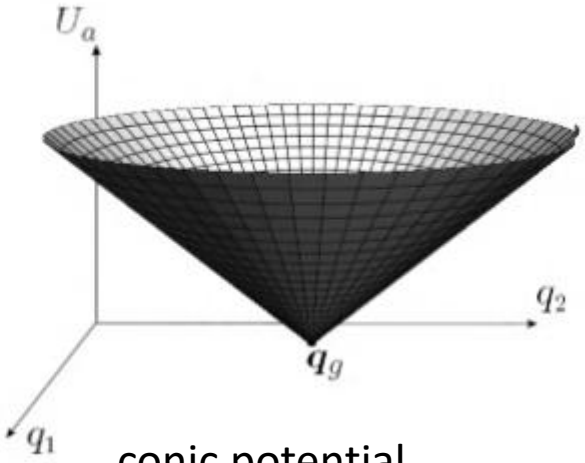
local minima!

This picture is taken from the textbook:

B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo: *Robotics: Modelling, Planning and Control, 3rd Ed.*
Springer, 2009



paraboloid potential



conic potential

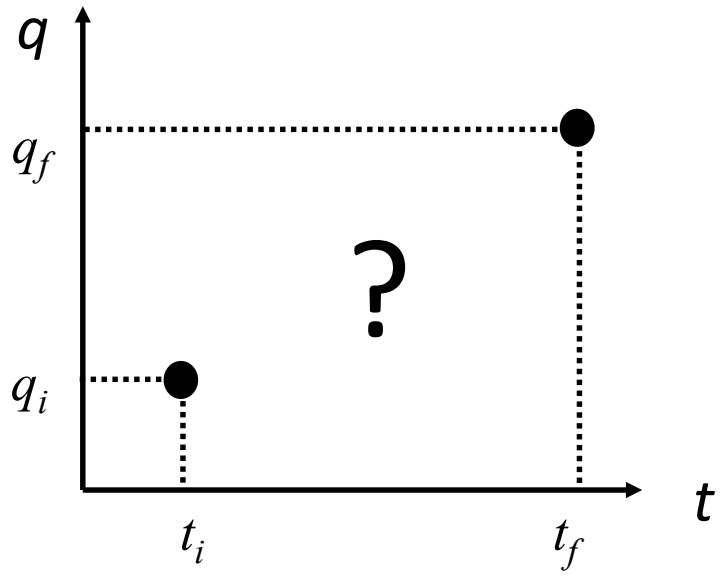
The method is particularly used in **online path planning**

Alternative definitions of time laws

- We have reviewed polynomial time laws and trapezoidal velocity profiles to solve the problem of trajectory generation

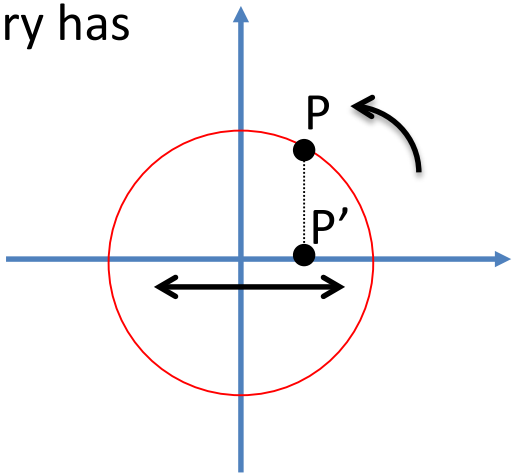
- In fact there are several alternative methods to solve the same problem

- Here we will address **harmonic** trajectories and **cycloidal** trajectories



Harmonic trajectory

The **harmonic trajectory** generalizes the equation of a harmonic motion, where the acceleration is proportional to the position, with opposite sign. A harmonic trajectory has continuous derivatives of all orders in all the internal points of the trajectory.



The harmonic trajectory corresponds to the motion of the projection P' of a point P moving along a circle with constant angular velocity

The equations are:

$$q(t) = \frac{q_f - q_i}{2}\left(1 - \cos\left(\frac{\pi(t - t_i)}{t_f - t_i}\right)\right) + q_i$$

$$q(t_i) = q_i, \quad q(t_f) = q_f$$

$$\dot{q}(t) = \frac{\pi(q_f - q_i)}{2(t_f - t_i)}\sin\left(\frac{\pi(t - t_i)}{t_f - t_i}\right)$$

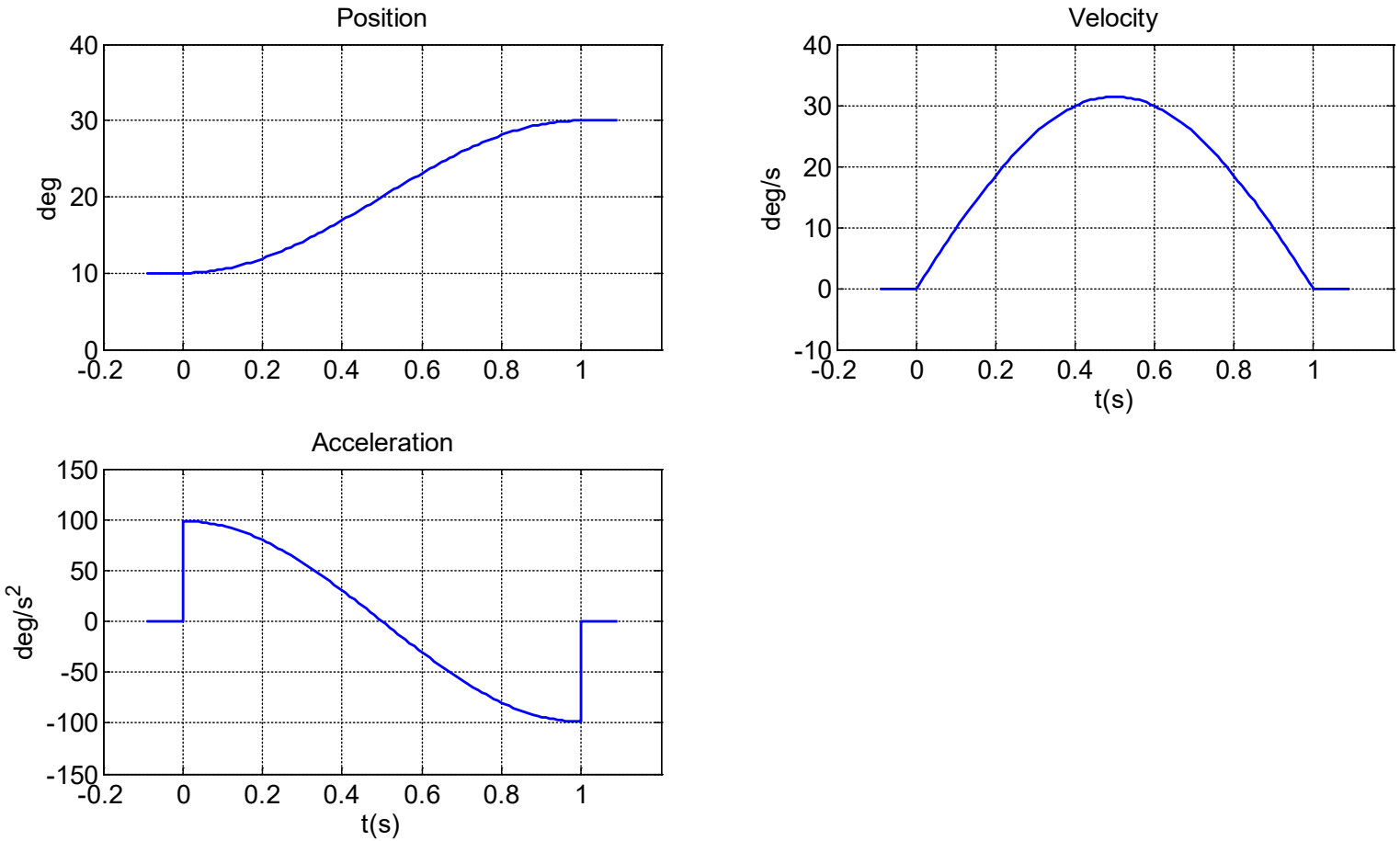
$$\dot{q}(t_i) = 0, \quad \dot{q}(t_f) = 0$$

$$\ddot{q}(t) = \frac{\pi^2(q_f - q_i)}{2(t_f - t_i)^2}\cos\left(\frac{\pi(t - t_i)}{t_f - t_i}\right)$$

Harmonic trajectory (example)

$$t_i = 0,\ t_f = 1\ s,$$

$$q_i = 10°,\ q_f = 30°$$



Cycloidal trajectory

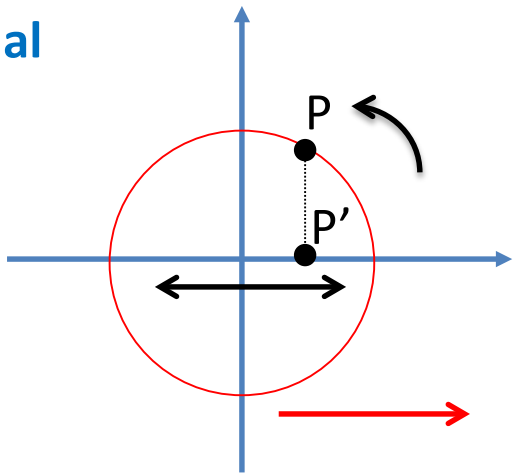
The harmonic trajectory has discontinuities in the acceleration in the initial and final instants, and then undefined (or infinite) values of jerk. An alternative is the **cycloidal trajectory**, which is continuous in the acceleration, too.

Here are the equations:

$$q(t) = (q_f - q_i)\left(\frac{t - t_i}{t_f - t_i} - \frac{1}{2\pi}\sin\left(\frac{2\pi(t - t_i)}{t_f - t_i}\right)\right) + q_i \qquad q(t_i) = q_i, \quad q(t_f) = q_f$$

$$\dot{q}(t) = \frac{q_f - q_i}{t_f - t_i}\left(1 - \cos\left(\frac{2\pi(t - t_i)}{t_f - t_i}\right)\right) \qquad \dot{q}(t_i) = 0, \quad \dot{q}(t_f) = 0$$

$$\ddot{q}(t) = \frac{2\pi(q_f - q_i)}{(t_f - t_i)^2}\sin\left(\frac{2\pi(t - t_i)}{t_f - t_i}\right) \qquad \ddot{q}(t_i) = 0, \quad \ddot{q}(t_f) = 0$$

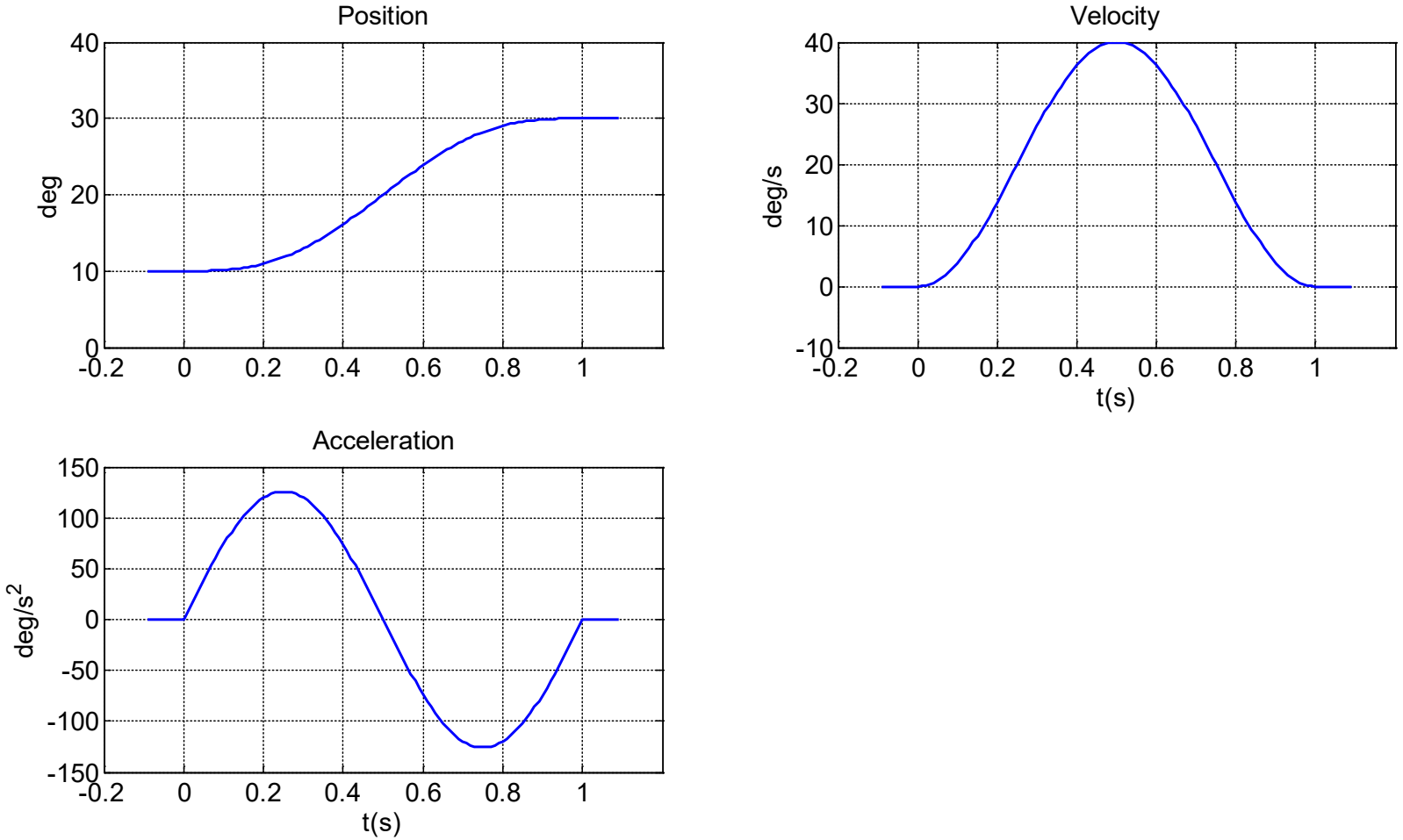


The cycloidal trajectory corresponds to the motion of the projection P' of a point P moving along a circle with constant angular velocity, when the circle advances with linear velocity

Cycloidal trajectory (example)

$$t_i = 0,\ t_f = 1\ s,$$

$$q_i = 10°,\ q_f = 30°$$



Trajectory scaling

Once a trajectory has been planned, it is often necessary to scale it in order to satisfy the constraints on the actuation system, which emerge in terms of saturations.
In particular we will consider:

1. **Kinematic scaling**: the trajectory needs satisfy the constraints on the maximum velocity and acceleration
2. **Dynamic scaling**: the trajectory needs satisfy the constraints on the maximum achievable torques/forces by the actuators

- Kinematic scaling of the trajectory is relevant for those trajectory profiles (cubic, harmonic, …) for which such values are not assigned in the planning itself.
- For kinematic scaling we can proceed joint by joint (no coupling effects)
- For dynamic scaling we will need to consider the whole coupled model of the robot

Trajectory normalization

In order to kinematically scale the trajectory it is convenient to express it in a parametric form, as a function of a suitably normalized parameter $\sigma = \sigma(t)$.

Given the trajectory $q(t)$, defined between points q_i and q_f with a travel time of $T = t_f - t_i$, its expression in **normalized form** is as follows:

$$q(t) = q_i + h\sigma(\tau) \qquad \text{with } h = q_f - q_i,\ 0 \le \sigma(\tau) \le 1,\ \tau = \frac{t - t_i}{T}, \quad 0 \le \tau \le 1 \qquad \text{(normalized time)}$$

It follows:

$$\begin{cases} \dfrac{dq(t)}{dt} = \dfrac{h}{T}\sigma'(\tau) \\ \dfrac{d^2q(t)}{dt^2} = \dfrac{h}{T^2}\sigma''(\tau) \\ \vdots \\ \dfrac{d^nq(t)}{dt^n} = \dfrac{h}{T^n}\sigma^{(n)}(\tau) \end{cases}$$

Maximum values of velocity, acceleration, etc., are obtained in correspondence to the maximum values of functions $\sigma^{(i)}(\tau)$: by **modifying the travel time T** of the trajectory it is possible to satisfy the constraints on the kinematic saturations.

Polynomial trajectory of degree 3

This trajectory can be parameterized with the polynomial:

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

Assigning the boundary conditions $\sigma'(0) = 0, \sigma'(1) = 0$ (besides $\sigma(0) = 0, \sigma(1) = 1$):

$$a_0 = 0, \quad a_1 = 0, \quad a_2 = 3, \quad a_3 = -2$$

from which:

$$\sigma(\tau) = 3\tau^2 - 2\tau^3 \quad \sigma''(\tau) = 6 - 12\tau$$
$$\sigma'(\tau) = 6\tau - 6\tau^2 \quad \sigma'''(\tau) = -12$$

Maximum values of velocity and acceleration are then:

$$\sigma'_{\max} = \sigma'(0.5) = \frac{3}{2} \quad \Rightarrow \quad \dot{q}_{\max} = \frac{3h}{2T}$$

$$\sigma''_{\max} = \sigma''(0) = 6 \quad \Rightarrow \quad \ddot{q}_{\max} = \frac{6h}{T^2}$$

Polynomial trajectory of degree 5

This trajectory can be parameterized with the polynomial:

$$\sigma(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 + a_4\tau^4 + a_5\tau^5$$

Assigning boundary conditions $\sigma(0) = 0, \sigma(1) = 1, \sigma'(0) = 0, \sigma'(1) = 0, \sigma''(0) = 0, \sigma''(1) = 0$:

$$a_0 = 0, \quad a_1 = 0, \quad a_2 = 0, \quad a_3 = 10, \quad a_4 = -15, \quad a_5 = 6$$

from which:

$$\sigma(\tau) = 10\tau^3 - 15\tau^4 + 6\tau^5 \qquad \sigma''(\tau) = 60\tau - 180\tau^2 + 120\tau^3$$
$$\sigma'(\tau) = 30\tau^2 - 60\tau^3 + 30\tau^4 \qquad \sigma'''(\tau) = 60 - 360\tau + 360\tau^2$$

Maximum values of velocity and acceleration are then:

$$\sigma'_{\max} = \sigma'(0.5) = \frac{15}{8} \quad \Rightarrow \quad \dot{q}_{\max} = \frac{15h}{8T}$$

$$\sigma''_{\max} = \sigma''(0.2123) = \frac{10\sqrt{3}}{3} \quad \Rightarrow \quad \ddot{q}_{\max} = \frac{10\sqrt{3}h}{3T^2}$$

Harmonic trajectory

This trajectory can be parameterized with the function:

$$\sigma(\tau) = \frac{1}{2}(1 - \cos(\pi\tau))$$

from which:

$$\sigma'(\tau) = \frac{\pi}{2}\sin(\pi\tau)$$

$$\sigma''(\tau) = \frac{\pi^2}{2}\cos(\pi\tau)$$

$$\sigma'''(\tau) = \frac{\pi^3}{2}\sin(\pi\tau)$$

Maximum values of velocity and acceleration are then:

$$\sigma'_{\max} = \sigma'(0.5) = \frac{\pi}{2} \quad \Rightarrow \quad \dot{q}_{\max} = \frac{\pi h}{2T}$$

$$\sigma''_{\max} = \sigma''(0) = \frac{\pi^2}{2} \quad \Rightarrow \quad \ddot{q}_{\max} = \frac{\pi^2 h}{2T^2}$$

Cycloidal trajectory

This trajectory can be parameterized with the function:

$$\sigma(\tau) = \tau - \frac{1}{2\pi}\sin(2\pi\tau)$$

from which:

$$\sigma'(\tau) = 1 - \cos(2\pi\tau)$$
$$\sigma''(\tau) = 2\pi\sin(2\pi\tau)$$
$$\sigma'''(\tau) = 4\pi^2\cos(2\pi\tau)$$

Maximum values of velocity and acceleration are then:

$$\sigma'_{\max} = \sigma'(0.5) = 2 \quad \Rightarrow \quad \dot{q}_{\max} = 2\frac{h}{T}$$

$$\sigma''_{\max} = \sigma''(0.25) = 2\pi \quad \Rightarrow \quad \ddot{q}_{\max} = 2\pi\frac{h}{T^2}$$

Kinematic scaling: an example

We want to design a trajectory with $q_i = 10°, q_f = 50°$,

for an actuator characterized by $\dot{q}_{max} = 30\,°/s\,, \ddot{q}_{max} = 80\,°/s^2$.

The results reported in the table can be obtained ($h = 40°$).

The fastest profiles are those with discontinuous acceleration at the beginning and at the end.

Trajectory	Formulas	Constraints	T_{min}
fast → Polin. 3rd degree	$\begin{cases} \dot{q}_{max} = \dfrac{3h}{2T} \\ \ddot{q}_{max} = \dfrac{6h}{T^2} \end{cases}$	$\begin{cases} T = \dfrac{3h}{60} = 2 \\ T = \sqrt{\dfrac{6h}{80}} = 1.732 \end{cases}$	2
Polin. 5th degree	$\begin{cases} \dot{q}_{max} = \dfrac{15h}{8T} \\ \ddot{q}_{max} = \dfrac{10\sqrt{3}h}{3T^2} \end{cases}$	$\begin{cases} T = \dfrac{15h}{240} = 2.5 \\ T = \sqrt{\dfrac{10\sqrt{3}h}{240}} = 1.699 \end{cases}$	2.5
fast → Harmonic	$\begin{cases} \dot{q}_{max} = \dfrac{\pi h}{2T} \\ \ddot{q}_{max} = \dfrac{\pi^2 h}{2T^2} \end{cases}$	$\begin{cases} T = \dfrac{\pi h}{60} = 2.094 \\ T = \sqrt{\dfrac{\pi^2 h}{160}} = 1.571 \end{cases}$	2.094
Cycloidal	$\begin{cases} \dot{q}_{max} = 2\dfrac{h}{T} \\ \ddot{q}_{max} = 2\pi\dfrac{h}{T^2} \end{cases}$	$\begin{cases} T = \dfrac{2h}{30} = 2.667 \\ T = \sqrt{\dfrac{2\pi h}{80}} = 1.772 \end{cases}$	2.667

Dynamic scaling

We will discuss the **dynamic scaling** technique making reference directly to the model of the robotic manipulator (neglecting friction effects):

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau$$

For each joint an equation of the following kind holds:

$$\mathbf{B}_i^T(\mathbf{q})\ddot{\mathbf{q}} + \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{C}_i(\mathbf{q})\dot{\mathbf{q}} + g_i(\mathbf{q}) = \tau_i \qquad i = 1, \dots, n$$

where $\mathbf{C}_i(\mathbf{q})$ is a suitable matrix.

Let us consider a **parameterization** of the trajectory in terms of a **scalar function**:

$$\mathbf{q} = \mathbf{q}(r), r = r(t)$$

(this means that all the joint positions depend on the time in the same way)

Then: $\dot{\mathbf{q}} = \frac{d\mathbf{q}}{dr}\dot{r}, \qquad \ddot{\mathbf{q}} = \frac{d^2\mathbf{q}}{dr^2}\dot{r}^2 + \frac{d\mathbf{q}}{dr}\ddot{r}$

Dynamic scaling

By substituting in the dynamic equation of the i_{th} joint we have:

$$\left[\mathbf{B}_i^T\big(\mathbf{q}(r)\big)\frac{d\mathbf{q}}{dr}\right]\ddot{r} + \left[\mathbf{B}_i^T\big(\mathbf{q}(r)\big)\frac{d^2\mathbf{q}}{dr^2} + \frac{1}{2}\frac{d\mathbf{q}^T}{dr}\mathbf{C}_i\big(\mathbf{q}(r)\big)\frac{d\mathbf{q}}{dr}\right]\dot{r}^2 + g_i\big(\mathbf{q}(r)\big) = \tau_i$$

i.e. an equation in the form:

$$\alpha_i(r)\ddot{r} + \beta_i(r)\dot{r}^2 + \gamma_i(r) = \tau_i$$

Observe that γ_i depends on the position only (and not on the velocity).

Torques needed to execute the motion are thus:

$$\tau_i(t) = \alpha_i\big(r(t)\big)\ddot{r}(t) + \beta_i\big(r(t)\big)\dot{r}^2(t) + \gamma_i\big(r(t)\big), \quad i = 1, \dots, n, \quad t \in [0, T]$$

Dynamic scaling

In order to obtain a different parameterization of the trajectory, consider now a **time scaling**, for instance a linear one:

$$\theta = kt \quad \theta \in [0, kT]$$

we change the time scale and consider a new time θ

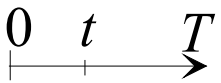
Compared with the time scaling used for kinematic scaling, here we are not using symbol τ (to avoid confusion with torques), we are assuming $t_i = 0$, and we are assuming a more general scaling ($k = 1/T$ in kinematic scaling)

We obtain:

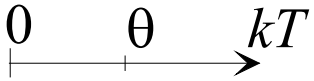
$$r(t) = \hat{r}(\theta), \quad \dot{r}(t) = k\hat{r}'(\theta), \quad \ddot{r}(t) = k^2\hat{r}''(\theta)$$

where $(\ )'$ stands for derivative with respect to θ.

- If $k > 1$ the scaled trajectory is slower



- If $k < 1$ the scaled trajectory is faster



Dynamic scaling

$$\tau_i(t) = \alpha_i\big(r(t)\big)\ddot{r}(t) + \beta_i\big(r(t)\big)\dot{r}^2(t) + \gamma_i\big(r(t)\big), \quad i = 1, \dots, n, \quad t \in [0, T]$$

With the new parameterization, the torques become:

$$\begin{aligned}
\tau_i(\theta) &= \alpha_i\big(\hat{r}(\theta)\big)\hat{r}''(\theta) + \beta_i\big(\hat{r}(\theta)\big)\hat{r}'^2(\theta) + \gamma_i\big(\hat{r}(\theta)\big) = \\
&= \alpha_i\big(r(t)\big)\frac{\ddot{r}(t)}{k^2} + \beta_i\big(r(t)\big)\frac{\dot{r}^2(t)}{k^2} + \gamma_i\big(r(t)\big) = \\
&= \frac{1}{k^2}\left[\alpha_i\big(r(t)\big)\ddot{r}(t) + \beta_i\big(r(t)\big)\dot{r}^2(t)\right] + \gamma_i\big(r(t)\big) = \\
&= \frac{1}{k^2}\left[\tau_i(t) - g_i\big(r(t)\big)\right] + g_i\big(r(t)\big)
\end{aligned}$$

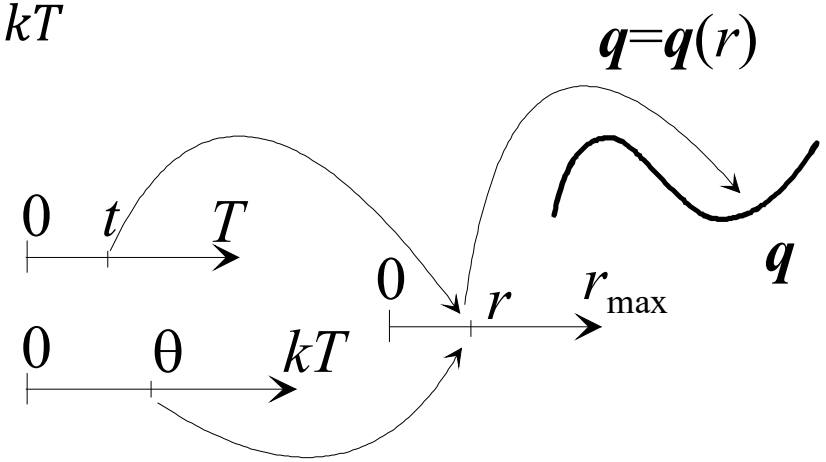
Then:

$$\tau_i(\theta) - g_i(\theta) = \frac{1}{k^2}\left[\tau_i(t) - g_i(t)\right]$$

Dynamic scaling

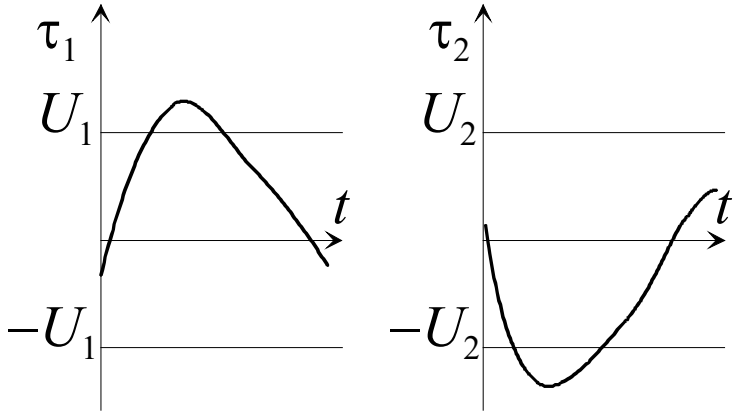
$$\tau_i(\theta) - g_i(\theta) = \frac{1}{k^2}[\tau_i(t) - g_i(t)]$$

- With a new parameterization of the trajectory it is not necessary to compute again the dynamics of the system
- New torques are obtained, apart from the gravitational term (which does not depend on the parameterization), multiplying the torques obtained with the original trajectory by the factor $1/k^2$
- The travel time of the new trajectory is kT



Dynamic scaling: example

Consider a two d.o.f. manipulator subjected to a trajectory which generates the following torques:



To scale the trajectory we compute the value:

$$k^2 = \max\left\{1, \frac{|\tau_1|}{U_1}, \frac{|\tau_2|}{U_2}\right\} \geq 1$$

New torques will be realizable ($\tau(\theta) = \tau(t)/k^2$) and at least one of them will saturate in one point.

Scaling the trajectory in order to avoid that the torque exceeds the maximum value in a given interval may excessively slow down the execution: we can resort in this case to a **variable scaling** (i.e. applied only in those intervals where there is torque saturation).

Interpolation of points

So far we have considered conditions only on the initial and final points of the trajectory.
We will now consider the more general situation where **intermediate points** have to be interpolated at given instants:

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_{n-1} \\ t_n \end{bmatrix} \Rightarrow \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{n-1} \\ q_n \end{bmatrix}$$

Interpolation with a polynomial

The problem of finding a trajectory that passes through n points can be solved adopting a **polynomial function** of degree $n-1$:

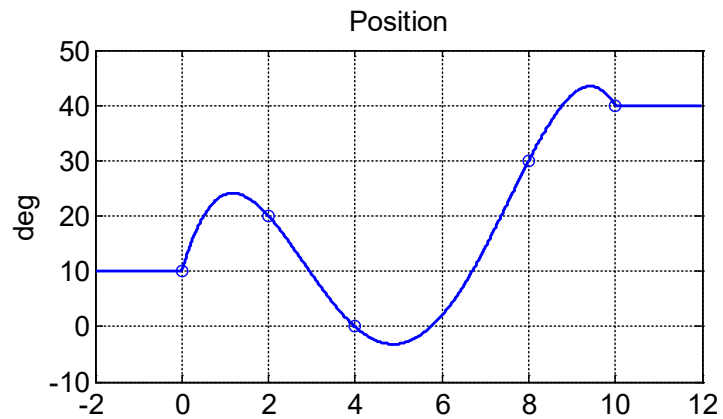
$$q(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_{n-1} t^{n-1}$$

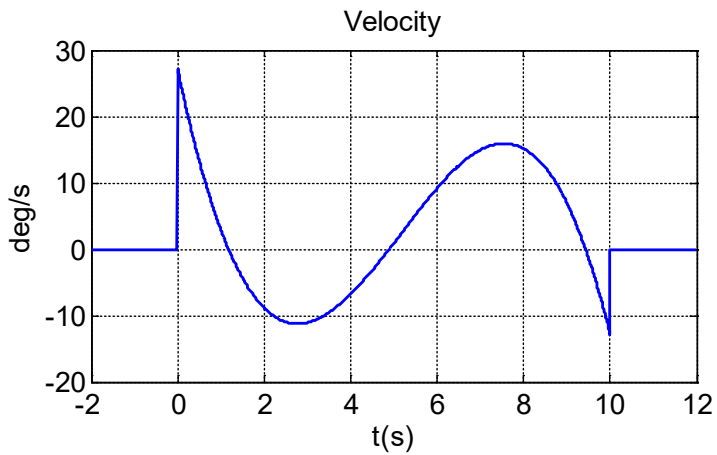
Given the values t_i, q_i, $i = 1, \cdots, n$ vectors $\mathbf{Q}$, $\mathbf{a}$ and matrix $\mathbf{T}$ (**Vandermonde matrix**) are built as:

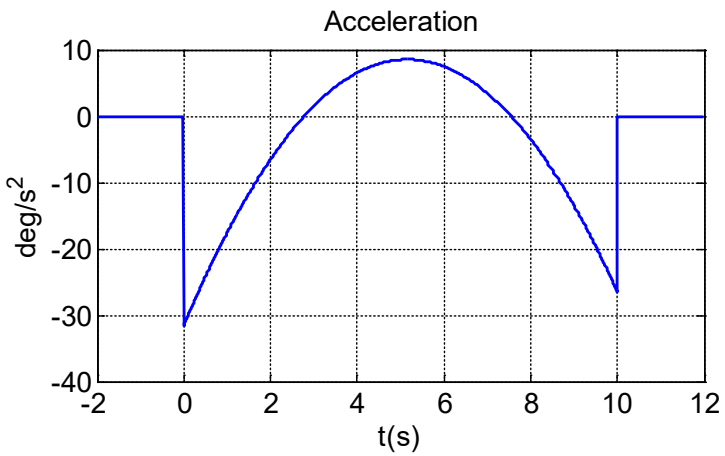
$$\mathbf{Q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{n-1} \\ q_n \end{bmatrix} = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ & & \vdots & \\ 1 & t_{n-1} & \cdots & t_{n-1}^{n-1} \\ 1 & t_n & \cdots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix} = \mathbf{T}\mathbf{a}$$

It follows: $\mathbf{a} = \mathbf{T}^{-1}\mathbf{Q}$ (matrix $\mathbf{T}$ is always invertible if $t_i > t_{i-1}$, $i = 1, \cdots, n$)

Interpolation with a polynomial







$$t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10$$
$$q_1 = 10° \quad q_2 = 20° \quad q_3 = 0° \quad q_4 = 30° \quad q_5 = 40°$$

Interpolation with a polynomial

A clear advantage of the polynomial interpolation is that function $q(t)$ has continuous derivatives of all the orders inside the interval $[t_1 \;\; t_n]$.

However the method is **not efficient from a numerical point of view**: as the number n of points increases, the condition number k (ratio between the maximum and minimum eigenvalue) of the Vandermonde matrix $\mathbf{T}$ increases too, making the inversion problem numerically ill-conditioned.

If, for example, $t_i = \frac{i}{n}, i = 1, \cdots, n)$:

n	3	4	5	6	10	15	20
k	15.1	98.87	686.43	4924.37	$1.519 \cdot 10^{7}$	$4.032 \cdot 10^{11}$	$1.139 \cdot 10^{16}$

Other, more efficient, methods exist to compute the coefficients of the polynomial, however the numerical difficulties stand for high values of n.

Interpolation with a polynomial

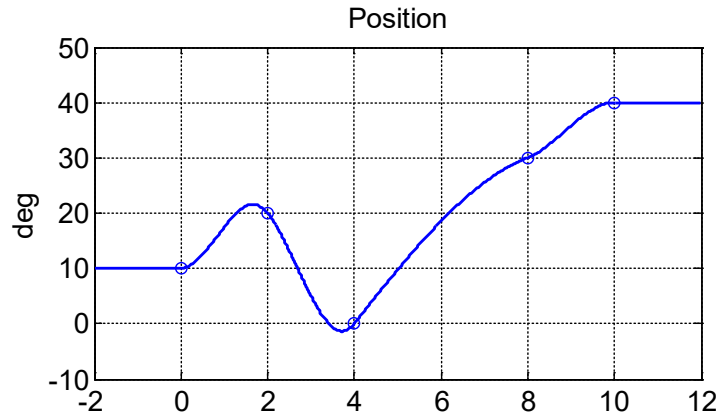
Regardless the numerical difficulties, the interpolation of n points with only one polynomial of degree $n-1$ has **other disadvantages**:

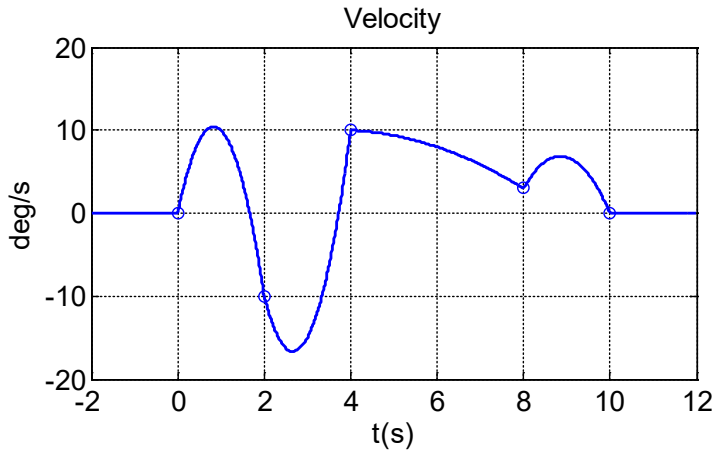
1. the degree of the polynomial depends on n and, for large values of n, the amount of computations might be remarkable;
2. changing a single point (t_i, q_i) implies to compute again the entire polynomial;
3. adding a final point (t_{n+1}, q_{n+1}) implies the use of a higher degree polynomial and thus the computation of all the coefficients;
4. the obtained solution in general presents undesired oscillations

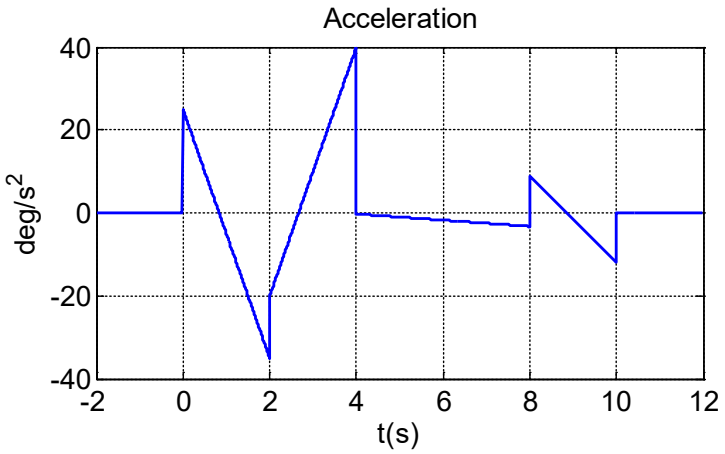
An alternative solution, instead of using a single polynomial of degree $n-1$, is to use $n-1$ polynomials of degree p (typically lower), each of which defined in an interval of the trajectory.
The degree p of the polynomials is usually equal to 3 (**pieces of cubic trajectories**).

A first, and obvious, way to proceed is to assign positions and velocities in all the points and then to compute the coefficients of the cubic polynomials between two consecutive points.

Interpolation with cubics







$t_1 = 0$	$t_2 = 2$	$t_3 = 4$	$t_4 = 8$	$t_5 = 10$
$q_1 = 10°$	$q_2 = 20°$	$q_3 = 0°$	$q_4 = 30°$	$q_5 = 40°$
$\dot{q}_1 = 0°/s$	$\dot{q}_2 = -10°/s$	$\dot{q}_3 = 10°/s$	$\dot{q}_4 = 3°/s$	$\dot{q}_5 = 0°/s$

Interpolation with cubics

If only the intermediate positions are specified, and not the intermediate velocities, these can be assigned with **rules** like:

$$\dot{q}_1 = 0$$

$$\dot{q}_k = \begin{cases} 0 & \text{sign}(R_k) \neq \text{sign}(R_{k+1}) \\ \dfrac{R_k + R_{k+1}}{2} & \text{sign}(R_k) = \text{sign}(R_{k+1}) \end{cases}$$

$$\dot{q}_n = 0$$

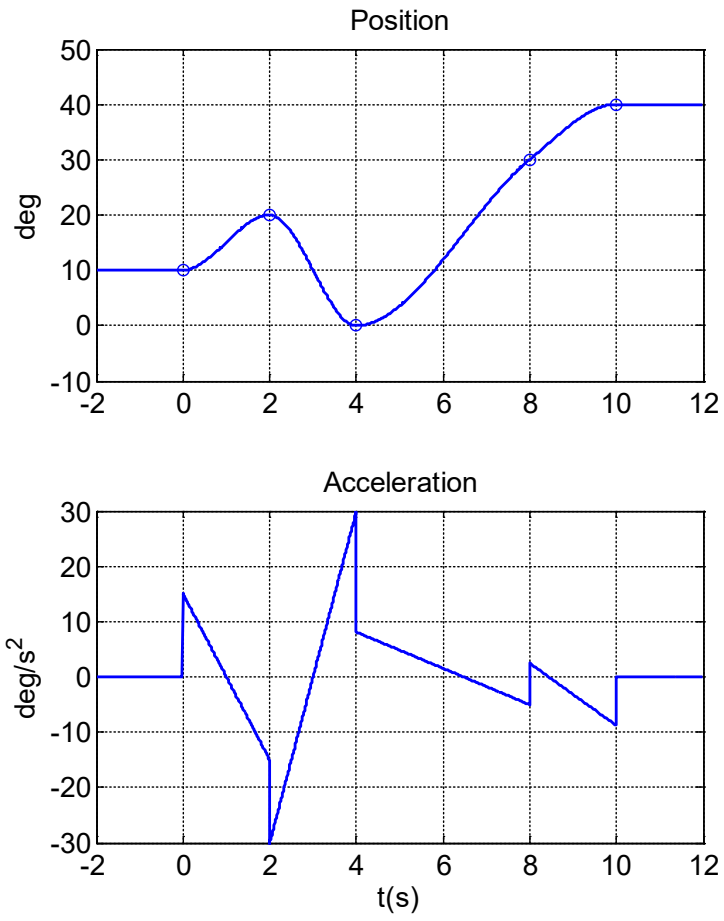
At an intermediate point, if the slopes before and after have different signs, we set zero velocity at that point, otherwise we assign a velocity which is the average of the two slopes.

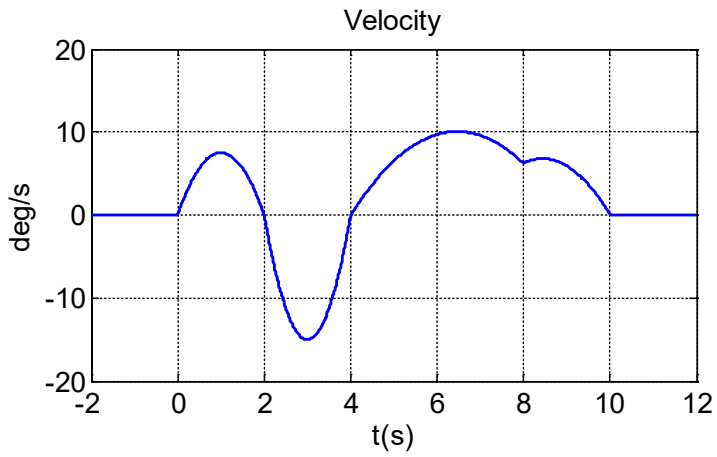
where:

$$R_k = \frac{q_k - q_{k-1}}{t_k - t_{k-1}}$$

is the slope in the interval $[t_{k-1}, t_k]$

Interpolation with cubics





$$t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10$$
$$q_1 = 10° \quad q_2 = 20° \quad q_3 = 0° \quad q_4 = 30° \quad q_5 = 40°$$

Spline

The interpolation with cubic polynomials generates a trajectory which presents a discontinuity in the acceleration in the intermediate points.

In order to avoid this problem, while keeping cubic interpolation, we must avoid to assign specific values of velocity in the intermediate points, requesting just the continuity of velocities and accelerations (and of course positions) in these points.

The trajectory which is obtained this way is called **spline** (smooth path line).

The spline is the minimum curvature interpolating function, given some conditions on continuity of derivatives.

Splines are used in a variety of domains, in particular computer graphics.

Spline: conditions

Since with n points we need $n-1$ polynomials like:

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

each of which has 4 coefficients, the total number of coefficients to be computed is $4(n-1)$. The conditions to be imposed are:

- $2(n-1)$ conditions of passage through the points (each cubic has to interpolate the points at its boundaries)
- $n-2$ conditions on continuity of velocities in the intermediate points
- $n-2$ conditions on continuity of accelerations in the intermediate points

We thus have: $4(n-1) - 2(n-1) - 2(n-2) = 2$

residual degrees of freedom.

A (not unique) way to use these 2 degrees of freedom consists in assigning suitable initial and final conditions on the velocity.

Spline: analytical position of the problem

We want to determine a function:

$$q(t) = \{q_k(t), \quad t \in [t_k, t_{k+1}], \quad k = 1, \dots, n-1\}$$
$$q_k(\tau) = a_{k0} + a_{k1}\tau + a_{k2}\tau^2 + a_{k3}\tau^3, \quad \tau \in [0, T_k] \qquad (\tau = t - t_k, \quad T_k = t_{k+1} - t_k)$$

with the conditions:

$$\begin{aligned}
q_k(0) &= q_k, \;\; q_k(T_k) = q_{k+1} && k = 1, \dots, n-1 \\
\dot{q}_k(T_k) &= \dot{q}_{k+1}(0) = v_{k+1} && k = 1, \dots, n-2 \\
\ddot{q}_k(T_k) &= \ddot{q}_{k+1}(0) && k = 1, \dots, n-2 \\
\dot{q}_1(0) &= v_1, \;\; \dot{q}_{n-1}(T_{n-1}) = v_n
\end{aligned}$$

where the quantities $v_k, k = 2, \dots, n-1$ are not specified.

The problem consists in finding the coefficients a_{ki}.

Spline: algorithm

Assume initially that the **velocities** $v_k, k = 2, \dots, n-1$ in the intermediate points are **known**. In this way, for each cubic polynomial we have four boundary conditions on position and velocity, which give rise to the system:

$$
\begin{aligned}
q_k(0) &= a_{k0} = q_k \\
\dot{q}_k(0) &= a_{k1} = v_k \\
q_k(T_k) &= a_{k0} + a_{k1} T_k + a_{k2} T_k^2 + a_{k3} T_k^3 = q_{k+1} \\
\dot{q}_k(T_k) &= a_{k1} + 2a_{k2} T_k + 3a_{k3} T_k^2 = v_{k+1}
\end{aligned}
$$

Solving the system yields:

$$
\begin{aligned}
a_{k0} &= q_k \\
a_{k1} &= v_k \\
a_{k2} &= \frac{1}{T_k}\left[\frac{3(q_{k+1} - q_k)}{T_k} - 2v_k - v_{k+1}\right] \\
a_{k3} &= \frac{1}{T_k^2}\left[\frac{2(q_k - q_{k+1})}{T_k} + v_k + v_{k+1}\right]
\end{aligned}
$$

Spline: algorithm

Obviously the velocities $v_k, k = 2, \dots, n-1$ must be computed. Let us **impose the continuity of the accelerations** in the intermediate points:

$$\ddot{q}_k(T_k) = 2a_{k2} + 6a_{k3}T_k = 2a_{k+1,2} = \ddot{q}_{k+1}(0), \quad k = 1, \dots, n-2$$

By substituting the expressions for the coefficients a_{k2}, a_{k3}, $a_{k+1,2}$ and multiplying by $(T_k T_{k+1})/2$ we obtain:

$$T_{k+1}v_k + 2(T_{k+1} + T_k)v_{k+1} + T_k v_{k+2} = \frac{3}{T_k T_{k+1}}\left[T_k^2(q_{k+2} - q_{k+1}) + T_{k+1}^2(q_{k+1} - q_k)\right]$$

Spline: algorithm

$$T_{k+1}v_k + 2(T_{k+1} + T_k)v_{k+1} + T_k v_{k+2} = \frac{3}{T_k T_{k+1}}\left[T_k^2(q_{k+2} - q_{k+1}) + T_{k+1}^2(q_{k+1} - q_k)\right]$$

In matrix form:

$$\begin{bmatrix} T_2 & 2(T_1+T_2) & T_1 & & & & & \\ 0 & T_3 & 2(T_2+T_3) & T_2 & & & & \\ & & & & \vdots & & & \\ & & & & T_{n-2} & 2(T_{n-3}+T_{n-2}) & T_{n-3} & 0 \\ & & & & & T_{n-1} & 2(T_{n-2}+T_{n-1}) & T_{n-2} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-3} \\ c_{n-2} \end{bmatrix}$$

where constants c_k depend only on the intermediate positions and on the lengths of the intervals, all known quantities.

Since the velocities v_1 and v_n are known (they are specified as initial data), by eliminating the related columns we have:

Spline: algorithm

$$
\begin{bmatrix}
2(T_1+T_2) & T_1 & & & & \\
T_3 & 2(T_2+T_3) & T_2 & & & \\
 & & \vdots & & & \\
 & & & T_{n-2} & 2(T_{n-3}+T_{n-2}) & T_{n-3} \\
 & & & & T_{n-1} & 2(T_{n-2}+T_{n-1})
\end{bmatrix}
\begin{bmatrix} v_2 \\ \vdots \\ v_{n-1} \end{bmatrix}
=
\begin{bmatrix}
\frac{3}{T_1T_2}[T_1^2(q_3-q_2)+T_2^2(q_2-q_1)]-T_2v_1 \\
\frac{3}{T_2T_3}[T_2^2(q_4-q_3)+T_3^2(q_3-q_2)] \\
\vdots \\
\frac{3}{T_{n-3}T_{n-2}}[T_{n-3}^2(q_{n-1}-q_{n-2})+T_{n-2}^2(q_{n-2}-q_{n-3})] \\
\frac{3}{T_{n-2}T_{n-1}}[T_{n-2}^2(q_n-q_{n-1})+T_{n-1}^2(q_{n-1}-q_{n-2})]-T_{n-2}v_n
\end{bmatrix}
$$

i.e an equation in the form $\mathbf{Av}=\mathbf{c}$

Spline: algorithm

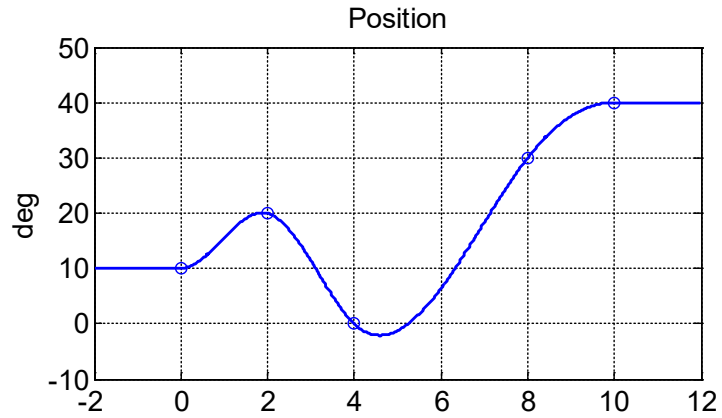
- Matrix $\mathbf{A}$ is a dominant diagonal matrix and is **always invertible** provided that $T_k > 0$
- Matrix $\mathbf{A}$ has a **tridiagonal** structure: for these matrices there exist efficient numerical techniques (Gauss-Jordan method) for its inversion
- Once the inverse of $\mathbf{A}$ is known we might compute velocities $v_2, \cdots, v_{n-1}$ as:

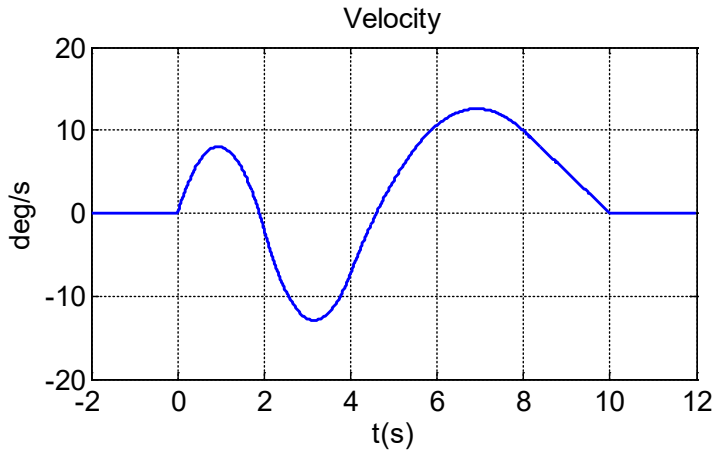
$$\mathbf{v} = \mathbf{A}^{-1}\mathbf{c}$$

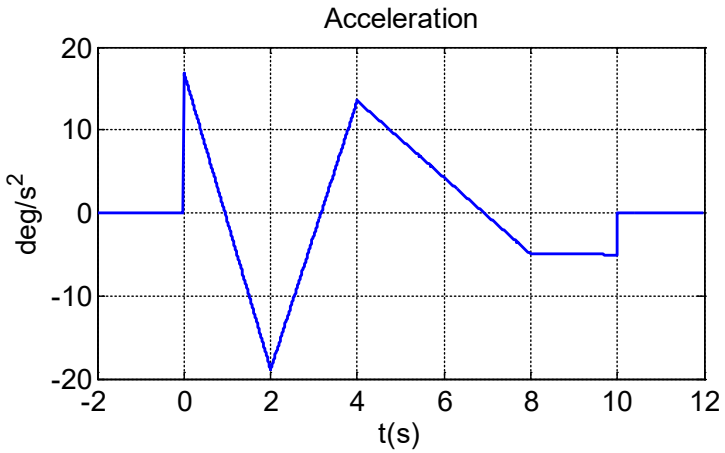
which **completely solves the problem**.

It is also possible to determine the spline with an alternative (yet completely equivalent) algorithm which computes the accelerations instead of the velocities in the intermediate points.

Spline: example







$$t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10$$
$$q_1 = 10° \quad q_2 = 20° \quad q_3 = 0° \quad q_4 = 30° \quad q_5 = 40°$$

Spline: travel time

The overall **travel time** of a spline is given by:

$$T = \sum_{k=1}^{n-1} T_k = t_n - t_1$$

It is possible to conceive an **optimization problem** which minimizes the overall travel time. The problem is to determine the values T_k so as to minimize T, with the constraints on the maximum velocities and accelerations.

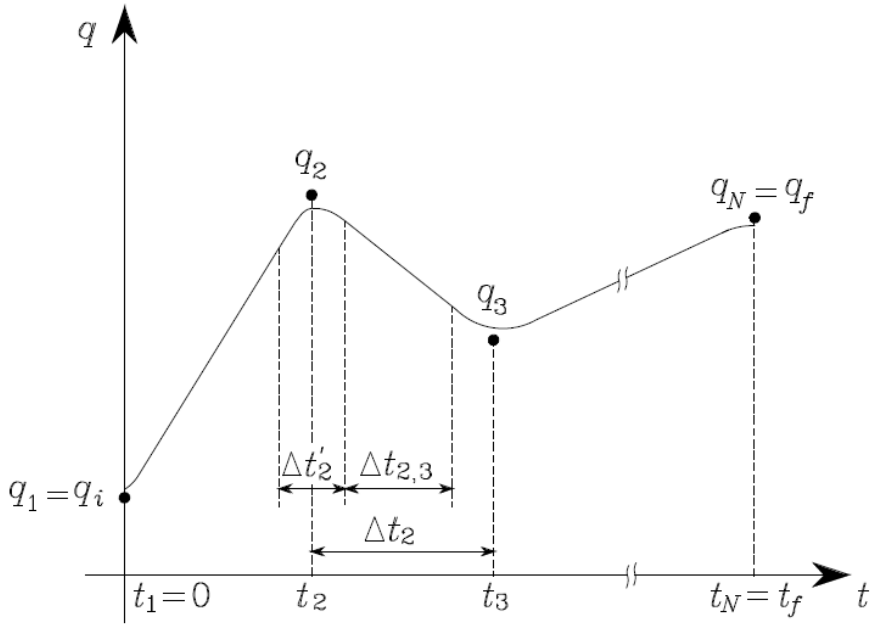
Formally:

$$\begin{cases} \min_{T_k} T = \sum_{k=1}^{n-1} T_k \\ \text{such that} \quad \begin{array}{ll} |\dot{q}(\tau, T_k)| < v_{\max} & \tau \in [0, T] \\ |\ddot{q}(\tau, T_k)| < a_{\max} & \tau \in [0, T] \end{array} \end{cases}$$

It is then a non linear optimum problem with a linear cost function, which can be solved with operations research techniques.

Interpolation with linear segments

An alternative, quite simple, way to handle the interpolation problem is to link the points with linear functions (**segments**). In order to avoid discontinuities in the velocity, the linear segments can be connected through parabolic blending (similar to the concatenation of linear paths).



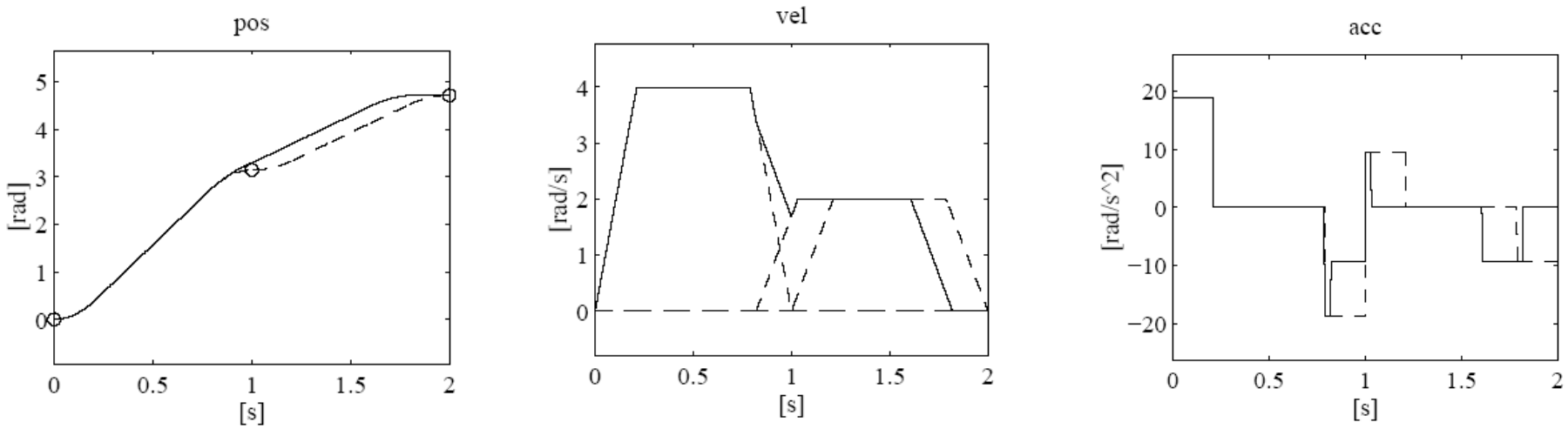
The resulting trajectory $q(t)$ does not pass through any of the intermediate points, though it is close to them. In this case the intermediate points are called **via points**

This picture is taken from the textbook:

B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo:
Robotics: Modelling, Planning and Control, 3rd Ed.
Springer, 2009

TVP and interpolation of points

If we use a sequence of trajectories with trapezoidal velocity profile to interpolate points, we would obtain a motion which passes through these points with zero velocity (i.e. stopping).
To avoid this we can start the planning of a trajectory before the end of the preceding one:



This picture is taken from the textbook:
B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo:
Robotics: Modelling, Planning and Control, 3rd Ed., Springer, 2009