Control of industrial robots

Motion planning

Prof. Paolo Rocco (paolo.rocco@polimi.it)
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Motion planning

- With the **motion planning** we want to assign the way the robot evolves from an initial posture to a final one.

- Motion planning is one of the essential problems in robotics. Most of the success on the market of a robot depends on the quality of motion planning.

- We will address the following issues related to motion planning:
  - introductory definitions (path vs. trajectory)
  - motion programming
  - path planning with obstacle avoidance
  - trajectories in joint space (point-to-point and with intermediate points)
  - kinematic and dynamic scaling of trajectories
  - trajectories in Cartesian space (position and orientation trajectories)
Definitions

We need to introduce some terminology concerning concepts which are often confused:

- **Path**: it is a geometric concept and stands for a line in a certain space (the space of Cartesian positions, the space of the orientations, the joint space,..) to be followed by the object whose motion has to be planned

- **Timing law**: it is the time dependence with which we want the robot to travel along the assigned path

- **Trajectory**: it is a path over which a timing law has been assigned

The final result of a motion planning problem is thus a trajectory that will then serve as an input to the real-time position/velocity controllers.
Trajectories in the operational space: the path (position and orientation) of the robot end effector is specified in the common Cartesian space.

- task description is natural
- constraints on the path can be accounted for
- singular points or redundant degrees of freedom generate problems
- online kinematic inversion is needed
Trajectories in joint space: the desired joint positions are directly specified

- problems related to kinematic singularities and redundant degrees of freedom are solved directly
- it is a mode of interest when we just want that the axes move from an initial to a final pose (and we are not interested in the resulting motion of the end effector)
- online kinematic inversion is not needed
Elements of a motion planning and control system

- **Instruction stack**: list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation**: converts an instruction into a trajectory to be executed
- **Inverse kinematics**: maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives**: closes the control loop ensuring tracking performance
Elements of a motion planning and control system

- **Instruction stack**: list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation**: converts an instruction into a trajectory to be executed
- **Inverse kinematics**: maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives**: closes the control loop ensuring tracking performance
A first mode for motion programming is the so called teaching-by-showing (also known as lead-through programming).

Using the teach-pendant, the operator moves the manipulator along the desired path. Position transducers memorize the positions the robot has to reach, which will be then jointed by a software for trajectory generation, possibly using some of the intermediate points as via points. The robot will be then able to autonomously repeat the motion.

No particular programming skills are required to the operator. On the other hand the method has limitations, since making a program requires that the programmer has the robot at his/her disposal (and then the robot is not productive), there is no possibility to execute logical functions or wait cycles and in general it is not possible to program complex activities.
The generation of the robot program can also be done in a robot programming environment. The robot programmer can move the robot in a virtual environment with high fidelity rendering of the robot motion in the robotic cell.

There are tools to record positions and to make the robot move along a path formed by such positions.

At the end the robot programming environment produces the code ready to be downloaded into the robot controller.
The robot programmer can also write the robot program directly using a robotic programming language.

With a **robotic programming language** the operator can program the motion of the robot as well as complex operations where the robot, inside a work-cell, interacts with other machines and devices. With respect to a general purpose programming language, the language provides specific robot oriented functionalities.

In the following, we will discuss some elements of the **PDL2** programming language by **COMAU Robotics**.
Program example

PROGRAM pack
VAR
home, feeder, table, discard : POSITION
BEGIN CYCLE
MOVE TO home
OPEN HAND 1
WAIT FOR $DIN[1] = ON
-- signals feeder ready
MOVE TO feeder
CLOSE HAND 1
IF $DIN[2] = OFF THEN
-- determines if good part
MOVE TO table
ELSE
MOVE TO discard
ENDIF
OPEN HAND 1
-- drop part on table or in bin
END pack

The program moves pieces from a feeder to a table or to a discard bin, depending on digital input signals:
Data types

Besides data types typical of any programming language (integer, real, boolean, string, array), some types specific to robotic applications are defined in PDL2. Among them:

**VECTOR:** representation of a vector through three components

**POSITION:** three components of Cartesian position, three orientation components (Euler angles) and a configuration string (which indicates whether it is a shoulder/elbow/wrist upper/lower configuration)

**JOINTPOS:** positions of the joints, measured in degrees
Reference frames

In PDL2 for each manipulator a world reference frame is defined. The operator might redefine the base frame ($BASE$) relative to the world frame. This is useful when the robot has to be repositioned in the work area, since it avoids recomputation of all the positions of objects in the cell.

Similarly the programmer can define a frame ($TOOL$) relative to the tool frame of the manipulator, which is useful whenever the tool mounted on the end effector is changed.
The instruction MOVE

With the instruction **MOVE**, **motion commands** of the arms are given. The format of the instruction is as follows:

```
MOVE <ARM[n]> <trajectory> dest_clause <opt_clauses> <sync_clause>
```

(note that a single controller can manage several arms).

The **trajectory clause** can take one of the following values:

- **LINEAR** (linear motion in Cartesian space)
- **CIRCULAR** (circular motion in Cartesian space)
- **JOINT** (motion in joint space)

The default is a motion in joint space.
MOVE: destination clauses

There are several destination clauses for the instruction MOVE. Main ones are:

MOVE TO

It moves the arm towards the specified destination, which might be a variable of type POSITION or JOINTPOS. For example:

MOVE LINEAR TO POS(x, y, z, e1, e2, e3, config)
MOVE TO home

The optional VIA clause can be used with the MOVE TO destination clause to specify a position through which the arm passes between the initial position and the destination. The VIA clause is used most commonly to define an arc for circular moves. For example:

MOVE TO initial
MOVE CIRCULAR TO destination VIA arc
MOVE NEAR

With this clause it is possible to specify a destination which is located along the approach vector of the tool, within a certain distance (expressed in mm) from a certain position. Example:

MOVE NEAR destination BY 250.0

MOVE AWAY

A destination can be specified along the approach vector of the tool, at a specified distance from the current position.

MOVE AWAY 250.0
MOVE: destination clauses

MOVE RELATIVE

A destination relative to the current position of the frame can be specified. Example:

MOVE RELATIVE VEC(100, 0, 100) IN frame

*frame* can be either **TOOL** or **BASE**

MOVE ABOUT

It defines the destination that the tool has to reach after a rotation around the specified vector, with respect to the current position. Example:

MOVE ABOUT VEC(0, 100, 0) BY 90 IN frame
MOVE: destination clauses

MOVE BY

Allows the programmer to specify a destination as a list of REAL expressions, with each item corresponding to an incremental move for the joint of an arm. Example:

MOVE BY {alpha, beta, gamma, delta, omega}

MOVE FOR

Allows the programmer to specify a partial move along the trajectory towards a theoretical destination. The orientation of the tool changes in proportion to the distance. Example:
Setting the speed and the acceleration

Using the optional clause `WITH` it is possible to assign values to some predefined temporary variables. In particular we can operate over the following variables:

$\text{PROG\_SPD\_OVR}$

It is a percentage by which we can modify the default speed value used by the controller for joint space motions.

$\text{PROG\_ACC\_OVR}, \text{PROG\_DEC\_OVR}$

They are percentages by which we can modify the default values of acceleration and deceleration used by the controller for joint space motions.

$\text{LIN\_SPD}$

It is the value of linear speed for a Cartesian motion, expressed in m/s:

**Examples:**

```plaintext
MOVE TO p1 WITH $\text{PROG\_SPD\_OVR}=50$
MOVE LINEAR TO p2 WITH $\text{LIN\_SPD}=0.6$
```
Continuous motion (MOVEFLY)

If we use the instruction \texttt{MOVEFLY} and the motion is followed by another motion, the arm will not stop at the first destination, but will move from the starting point of the first motion to the end point of the second motion, without stopping in the common point of the two motions. Example:

\begin{verbatim}
MOVE TO a
MOVEFLY TO b ADVANCE
MOVE TO c
\end{verbatim}

(the \texttt{ADVANCE} clause allows the interpretation of the next MOVE instruction as soon as the first motion begins)
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 MOVE 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.
Usually the path planning problem is 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.

We call $C_{\text{free}}$ 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 $C_{\text{free}}$. 
Setting the problem

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

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 $q_s$ and a goal configuration $q_g$.
Planning a collision-free motion for the robot means to generate a path from $q_s$ to $q_g$ if they belong to the same connected component of $C_{\text{free}}$, 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 $C_{\text{free}}$ 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)
PRM (Probabilistic roadmap)

- a random sample $q_{\text{rand}}$ of the C-space is selected using a uniform probability distribution and tested for collision
- if $q_{\text{rand}}$ 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 $q_{\text{rand}}$ and a near configuration $q_{\text{near}}$ 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 $q_s$ and $q_g$ 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: 
Roboticics: Modelling, Planning and Control, 3rd Ed.
Springer, 2009
Advantages of PRM:
- it is remarkably fast in finding solutions to motion planning problems
- unlike other methods, 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 $q_{\text{rand}}$ of the C-space is selected using a uniform probability distribution
- the configuration $q_{\text{near}}$ in the tree $T$ (which is progressively formed) that is the closest one to $q_{\text{rand}}$ is found and a new candidate configuration $q_{\text{new}}$ is produced on the segment joining $q_{\text{near}}$ to $q_{\text{rand}}$ at a predefined distance $\delta$ from $q_{\text{near}}$
- it is checked that both $q_{\text{new}}$ and the segment joining $q_{\text{near}}$ to $q_{\text{new}}$ belong to $C_{\text{free}}$
- if this is true, the tree $T$ is expanded by incorporating $q_{\text{new}}$ and the said segment
- to speed up the search, two trees are expanded, rooted at $q_s$ and $q_g$, 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
- 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 $C_{\text{free}}$ 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
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 $q$ the artificial force generated by the potential is defined as the negative gradient $-\nabla U(q)$ of the potential

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

The method is particularly used in online path planning
Elements of a motion planning and control system

- **Instruction stack**: list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation**: converts an instruction into a trajectory to be executed
- **Inverse kinematics**: maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives**: closes the control loop ensuring tracking performance
Inputs and outputs of a trajectory planner

As we have seen, in general the user, when assigning a motion command, specifies a restricted number of parameters as inputs:

- Space of definition: joint space or Cartesian space

- For the path: endpoints, possible intermediate points, path geometry (segment, circular arc, etc.)

- For the timing law: overall travelling time, maximum velocity and/or acceleration (or percentages thereof)

Based on this information the trajectory planner generates a dense sequence of intermediate points in the relevant space (joint space or Cartesian space) at a fixed rate (e.g. 10 ms).

In case of Cartesian space trajectories these points are additionally converted into points in joint space through kinematic inversion.

These values can be interpolated in order to match the rate of the motion controller (e.g. 1 ms or 500 µs): this is also called micro-interpolation.
Criteria for trajectory selection

Some criteria for the selection of the trajectories might be:

- computational efficiency and memory space
- continuity of positions, velocities (and possibly of accelerations and jerks)
- minimization of undesired effects (oscillations, irregular curvature)
- accuracy (no overshoot on final position)
Trajectories in joint space

When we plan the trajectory in joint space we want to generate a function $q(t)$ which interpolates the values assigned for the joint variables at the initial and final points.

It is sufficient to work component-wise (i.e. we consider a single joint variable $q_i(t)$): in the following we will then consider planning of a scalar variable.

Two situations are of interest:

- **Point-to-point** motion:
  - we just specify the endpoints

- **Interpolation** of points:
  - we also specify some intermediate points

When planning in joint space, the definition of the path as a geometric entity is not an issue, since we are not interested in a coordinated motion of the joints (apart from having all the joints complete their motion at the same instant).
The simplest case of trajectory planning for point-to-point motion is when some initial and final conditions are assigned on positions, velocities and possibly on acceleration and jerk and the travel time.

**Polynomial functions** of the following kind can be considered:

\[ q(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n \]

The higher the degree \( n \) of the polynomial, the larger the number of boundary conditions that can be satisfied and the smoother the trajectory will be.
Cubic polynomials

Suppose that the following boundary conditions are assigned:

- an initial and a final instants \( t_i \) and \( t_f \)
- initial position and velocity \( q_i \) and \( \dot{q}_i \)
- final position and velocity \( q_f \) and \( \dot{q}_f \)

We then have four boundary conditions. In order to satisfy them we need a polynomial of order at least equal to three (cubic polynomial):

\[
q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3
\]

If we impose the boundary conditions:

\[
\begin{align*}
q(t_i) &= q_i \\
\dot{q}(t_i) &= \dot{q}_i \\
q(t_f) &= q_f \\
\dot{q}(t_f) &= \dot{q}_f
\end{align*}
\]

we obtain:

\[
\begin{align*}
a_0 &= q_i \\
a_1 &= \dot{q}_i \\
a_2 &= \frac{-3(q_i - q_f) - (2\dot{q}_i + \dot{q}_f)T}{T^2} \\
a_3 &= \frac{2(q_i - q_f) + (\dot{q}_i + \dot{q}_f)T}{T^3}
\end{align*}
\]

\( T = t_f - t_i \)
Cubic polynomials: example

\( t_i = 0, \ t_f = 1 \text{ s}, q_i = 10^\circ, \ q_f = 30^\circ, \ \dot{q}_i = \dot{q}_f = 0^\circ / \text{s} \)
Polynomials of degree five

In order to assign conditions also on the accelerations, we need to consider polynomials of degree 5:

$$q(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 + a_3(t - t_i)^3 + a_4(t - t_i)^4 + a_5(t - t_i)^5$$

Imposing boundary conditions:

- $q(t_i) = q_i$, $q(t_f) = q_f$
- $\dot{q}(t_i) = \dot{q}_i$, $\dot{q}(t_f) = \dot{q}_f$
- $\ddot{q}(t_i) = \ddot{q}_i$, $\ddot{q}(t_f) = \ddot{q}_f$

we obtain:

- $a_0 = q_i$
- $a_1 = \dot{q}_i$
- $a_2 = \frac{1}{2}\ddot{q}_i$

Then:

- $a_3 = \frac{20(q_f - q_i) - (8\dot{q}_f + 12\ddot{q}_i)T - (3\dddot{q}_f - \dddot{q}_i)T^2}{2T^3}$
- $a_4 = \frac{30(q_i - q_f) + (14\dot{q}_f + 16\ddot{q}_i)T + (3\dddot{q}_f - 2\dddot{q}_i)T^2}{2T^4}$
- $a_5 = \frac{12(q_f - q_i) - 6(\dot{q}_f + \dot{q}_i)T - (\dddot{q}_f - \dddot{q}_i)T^2}{2T^5}$

with $T = t_f - t_i$.
Polynomials of degree five: example

\[ t_i = 0, \ t_f = 1 \text{ s}, \ q_i = 10^\circ, \ q_f = 30^\circ, \ \dot{q}_i = \dot{q}_f = 0^\circ / \text{s}, \ \ddot{q}_i = \ddot{q}_f = 0^\circ / \text{s}^2 \]
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 equations are:

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

\[
\dot{q}(t) = \frac{\pi(q_f - q_i)}{2(t_f - t_i)} \sin\left(\frac{\pi(t - t_i)}{t_f - t_i}\right) \quad \dot{q}(t_i) = 0, \quad \dot{q}(t_f) = 0
\]

\[
\ddot{q}(t) = \frac{\pi^2(q_f - q_i)}{2(t_f - t_i)^2} \cos\left(\frac{\pi(t - t_i)}{t_f - t_i}\right)
\]
Harmonic trajectory (example)

\[ t_i = 0, \ t_f = 1 \text{ s}, \ q_i = 10^\circ, \ q_f = 30^\circ \]
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:

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

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

\[
\ddot{q}(t) = \frac{2\pi(q_f - q_i)}{(t_f - t_i)^2} \sin \left( \frac{2\pi(t - t_i)}{t_f - t_i} \right) \\
\ddot{q}(t_i) = 0, \quad \ddot{q}(t_f) = 0
\]
Cycloidal trajectory (example)

\[ t_i = 0, \ t_f = 1 \text{ s}, \ q_i = 10^\circ, \ q_f = 30^\circ \]
Trapezoidal velocity profile (TVP)

A quite common industrial practice to generate the trajectory consists in planning a linear position profile adjusted at the beginning and at the end of the trajectory with parabolic bends. The resulting velocity profile has the typical trapezoidal shape.

The trajectory is then composed of three parts:

1. Constant accel., linear velocity, parabolic position;
2. Zero acceleration, constant velocity, linear position;
3. Constant deceleration, linear velocity, parabolic position.

Often the duration $T_a$ of the acceleration phase (phase 1) is set equal to the duration of the deceleration phase (phase 3): this way a trajectory is obtained, which is symmetric with respect to the central time instant. Of course it has to be $T_a \leq (t_f - t_i)/2$. 
### TVP: equations

**Acceleration:** \( t \in [t_i, t_i + T_a] \)

\[
\begin{align*}
\ddot{q}(t) &= \frac{\dot{q}_v}{T_a} \\
\dot{q}(t) &= \frac{\dot{q}_v}{T_a} (t - t_i) \\
q(t) &= q_i + \frac{\dot{q}_v}{2T_a} (t - t_i)^2
\end{align*}
\]

**Deceleration:** \( t \in [t_f - T_a, t_f] \)

\[
\begin{align*}
\ddot{q}(t) &= -\frac{\dot{q}_v}{T_a} \\
\dot{q}(t) &= \frac{\dot{q}_v}{T_a} (t_f - t) \\
q(t) &= q_f - \frac{\dot{q}_v}{2T_a} (t_f - t)^2
\end{align*}
\]

**Constant velocity:** \( t \in [t_i + T_a, t_f - T_a] \)

\[
\begin{align*}
\ddot{q}(t) &= 0 \\
\dot{q}(t) &= \dot{q}_v \\
q(t) &= q_i + \dot{q}_v \left( t - t_i - \frac{T_a}{2} \right)
\end{align*}
\]
TVP: example

\[ t_i = 0, \ t_f = 4s, \ T_a = 1s, \ q_i = 0^\circ, \ q_f = 30^\circ, \ \dot{q}_v = 10^\circ / s \]
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:

$$q(t) = q_i + h\sigma(\tau)$$

with $h = q_f - q_i$ and $0 \leq \sigma(\tau) \leq 1$,

$$\tau = \frac{t - t_i}{T}, \quad 0 \leq \tau \leq 1 \quad \text{(normalized time)}$$

It follows:

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

Maximum values of velocity, acceleration, etc., are obtained in correspondence to the maximum values of functions $\sigma^{(i)}(t)$: 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:

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

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

\[ \sigma(\tau) = 10\tau^3 - 15\tau^4 + 6\tau^5 \quad \sigma''(\tau) = 60\tau - 180\tau^2 + 120\tau^3 \]

\[ \sigma'(\tau) = 30\tau^2 - 60\tau^3 + 30\tau^4 \quad \sigma'''(\tau) = 60 - 360\tau + 360\tau^2 \]

Maximum values of velocity, acceleration and jerk are then:

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

\[ \sigma''_{\text{max}} = \sigma''(0.2123) = \frac{10\sqrt{3}}{3} \quad \Rightarrow \quad \ddot{q}_{\text{max}} = \frac{10\sqrt{3}h}{3T^2} \]
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, acceleration and jerk are then:

\[ \sigma'_{\max} = \sigma'(0.5) = \frac{\pi}{2} \implies \dot{\sigma}_{\max} = \frac{\pi h}{2T} \]

\[ \sigma''_{\max} = \sigma''(0) = \frac{\pi^2}{2} \implies \ddot{\sigma}_{\max} = \frac{\pi^2 h}{2T^2} \]
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, acceleration and jerk are then:

\[ \sigma'_{\text{max}} = \sigma'(0.5) = 2 \quad \Rightarrow \quad \dot{q}_{\text{max}} = 2 \frac{h}{T} \]
\[ \sigma''_{\text{max}} = \sigma''(0.25) = 2\pi \quad \Rightarrow \quad \ddot{q}_{\text{max}} = 2\pi \frac{h}{T^2} \]
We want to design a trajectory with $q_i = 10^\circ$, $q_f = 50^\circ$, for an actuator characterized by $\dot{q}_{\text{max}} = 30^\circ/s$, $\ddot{q}_{\text{max}} = 80^\circ/s^2$.

The following results can be obtained ($h=40^\circ$):

<table>
<thead>
<tr>
<th>Trajectory</th>
<th>Formulas</th>
<th>Constraints</th>
<th>$T_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polin. 3rd degree</td>
<td>$\dot{q}<em>{\text{max}} = \frac{3h}{2T}$, $\ddot{q}</em>{\text{max}} = \frac{6h}{T^2}$</td>
<td>$T = \frac{3h}{60} = 0.5$, $T = \frac{6h}{80} = 0.75$</td>
<td>2</td>
</tr>
<tr>
<td>Polin. 5th degree</td>
<td>$\dot{q}<em>{\text{max}} = \frac{15h}{8T}$, $\ddot{q}</em>{\text{max}} = \frac{10\sqrt{3}h}{3T^2}$</td>
<td>$T = \frac{15h}{240} = 0.625$, $T = \sqrt{\frac{10\sqrt{3}h}{240}} = 1.699$</td>
<td>2.5</td>
</tr>
<tr>
<td>Harmonic</td>
<td>$\dot{q}<em>{\text{max}} = \frac{\pi h}{2T}$, $\ddot{q}</em>{\text{max}} = \frac{\pi^2 h}{2T^2}$</td>
<td>$T = \frac{\pi h}{60} = 0.987$, $T = \sqrt{\frac{\pi^2 h}{160}} = 1.571$</td>
<td>2.094</td>
</tr>
<tr>
<td>Cycloidal</td>
<td>$\dot{q}<em>{\text{max}} = \frac{2h}{T}$, $\ddot{q}</em>{\text{max}} = 2\pi \frac{h}{T^2}$</td>
<td>$T = \frac{2h}{30} = 0.667$, $T = \sqrt{\frac{2\pi h}{80}} = 1.772$</td>
<td>2.667</td>
</tr>
</tbody>
</table>
Dynamic scaling

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

\[ B(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau \]

For each joint an equation of the following kind holds:

\[ B_i^T(q)\ddot{q} + \frac{1}{2} \dot{q}^T C_i(q)\dot{q} + g_i(q) = \tau_i \quad i = 1, \ldots, n \]

where \( C_i(q) \) is a suitable matrix.

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

\[ q = q(\sigma), \quad \sigma = \sigma(t) \quad \text{(this means that all the joint positions depend on the time in the same way)} \]

Then:

\[ \dot{q} = \frac{dq}{d\sigma} \dot{\sigma}, \quad \ddot{q} = \frac{d^2q}{d\sigma^2} \dot{\sigma}^2 + \frac{dq}{d\sigma} \ddot{\sigma} \]
By substituting in the dynamic equation of the $i_{th}$ joint we have:

$$
\begin{bmatrix}
B_i^T(q(\sigma)) \frac{dq}{d\sigma}
dq
ddq
dqq
\end{bmatrix} \ddot{\sigma} + \begin{bmatrix}
B_i^T(q(\sigma)) \frac{d^2q}{d\sigma^2} + \frac{1}{2} \frac{dq^T}{d\sigma} C_i(q(\sigma)) \frac{dq}{d\sigma}
\end{bmatrix} \dot{\sigma}^2 + g_i(q(\sigma)) = \tau_i
$$

i.e. an equation in the form:

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

Observe that $\gamma_i$ depends on the position only (and not on the velocity).

Torques needed to execute the motion are thus:

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

We obtain:

\[ \sigma(t) = \hat{\sigma}(\theta), \quad \dot{\sigma}(t) = k\hat{\sigma}'(\theta), \quad \ddot{\sigma}(t) = k^2\hat{\sigma}''(\theta) \]

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

- If \( k>1 \) the scaled trajectory is slower.
- If \( k<1 \) the scaled trajectory is faster.
$$\tau_i(t) = \alpha_i(\sigma(t))\ddot{\theta}(t) + \beta_i(\sigma(t))\dot{\sigma}^2(t) + \gamma_i(\sigma(t)), \quad i = 1, \ldots, n, \quad t \in [0, T]$$

With the new parameterization, the torques become:

$$\tau_i(\theta) = \alpha_i(\hat{\sigma}(\theta))\hat{\sigma}''(\theta) + \beta_i(\hat{\sigma}(\theta))\hat{\sigma}'^2(\theta) + \gamma_i(\hat{\sigma}(\theta)) =$$

$$= \alpha_i(\sigma(t))\frac{\ddot{\sigma}(t)}{k^2} + \beta_i(\sigma(t))\frac{\dot{\sigma}^2(t)}{k^2} + \gamma_i(\sigma(t)) =$$

$$= \frac{1}{k^2} [\alpha_i(\sigma(t))\ddot{\sigma}(t) + \beta_i(\sigma(t))\dot{\sigma}^2(t)] + \gamma_i(\sigma(t)) =$$

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

Then: $$\tau_i(\theta) - g_i(\theta) = \frac{1}{k^2} [\tau_i(t) - g_i(t)]$$
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 \(kT\).
Consider a two d.o.f. manipulator subjected to a trajectory which generates the following torques:

\[
\tau_1 = U_1 t - U_1 \\
\tau_2 = U_2 t - U_2
\]

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 \( n-1 \):

\[
q(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_{n-1} t^{n-1}
\]

Given the values \( t_i, q_i, i=1,\ldots,n \) vectors \( q, a \) and matrix \( T \) (Vandermonde matrix) are built as:

\[
\begin{bmatrix}
q_1 \\
q_2 \\
\vdots \\
q_{n-1} \\
q_n
\end{bmatrix} =
\begin{bmatrix}
1 & t_1 & \cdots & t_1^{n-1} \\
1 & t_2 & \cdots & t_2^{n-1} \\
\vdots \\
1 & t_{n-1} & \cdots & t_{n-1}^{n-1} \\
1 & t_n & \cdots & t_n^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-2} \\
a_{n-1}
\end{bmatrix}
= Ta
\]

It follows: \( a = T^{-1}q \) (matrix \( T \) is always invertible if \( t_i > t_{i-1}, i =1,\ldots,n \))
Interpolation with a polynomial

\[ t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10 \]
\[ q_1 = 10^\circ \quad q_2 = 20^\circ \quad q_3 = 0^\circ \quad q_4 = 30^\circ \quad q_5 = 40^\circ \]
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 $T$ increases too, making the inversion problem numerically ill-conditioned.

If, for example, $t_i = i/n, \ i = 1,...,n$:

<table>
<thead>
<tr>
<th>$n$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>15.1</td>
<td>98.87</td>
<td>686.43</td>
<td>4924.37</td>
<td>1.519 \cdot 10^7</td>
<td>4.032 \cdot 10^{11}</td>
<td>1.139 \cdot 10^{16}</td>
</tr>
</tbody>
</table>

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

\[
t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10 \\
q_1 = 10^\circ \quad q_2 = 20^\circ \quad q_3 = 0^\circ \quad q_4 = 30^\circ \quad q_5 = 40^\circ \\
\dot{q}_1 = 0^\circ / s \quad \dot{q}_2 = -10^\circ / s \quad \dot{q}_3 = 10^\circ / s \quad \dot{q}_4 = 3^\circ / s \quad \dot{q}_5 = 0^\circ / s
\]
Interpolation with cubics

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

\[
\begin{align*}
\dot{q}_1 &= 0 \\
\dot{q}_k &= \begin{cases} 
0 & \text{sign}(R_k) \neq \text{sign}(R_{k+1}) \\
\frac{R_k + R_{k+1}}{2} & \text{sign}(R_k) = \text{sign}(R_{k+1})
\end{cases} \\
\dot{q}_n &= 0
\end{align*}
\]

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

\[ t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10 \]
\[ q_1 = 10^\circ \quad q_2 = 20^\circ \quad q_3 = 0^\circ \quad q_4 = 30^\circ \quad q_5 = 40^\circ \]
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, assigning 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.
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:

\[ q(t) = \{q_k(t), \quad t \in [t_k, t_{k+1}], \quad k = 1, \ldots, n - 1 \} \]

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

with the conditions:

\[ q_k(0) = q_k, \quad q_k(T_k) = q_{k+1} \quad k = 1, \ldots, n - 1 \]

\[ \dot{q}_k(T_k) = \dot{q}_{k+1}(0) = v_{k+1} \quad k = 1, \ldots, n - 2 \]

\[ \ddot{q}_k(T_k) = \ddot{q}_{k+1}(0) \quad k = 1, \ldots, n - 2 \]

\[ \dot{q}_1(0) = v_1, \quad \dot{q}_{n-1}(T_{n-1}) = v_n \]

where the quantities \( v_k, \ k = 2, \ldots, n - 1 \) are not specified.

The problem consists in finding the coefficients \( a_{ki} \).
Spline: algorithm

Assume initially that the velocities $v_k$, $k=2,...,n-1$ 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:

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

Solving the system yields:

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

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

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

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:

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

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

In matrix form:

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

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

\[
\begin{bmatrix}
2(T_1 + T_2) & T_1 \\
T_3 & 2(T_2 + T_3) & T_2 \\
& & \ddots \\
T_{n-2} & 2(T_{n-3} + T_{n-2}) & T_{n-3} \\
T_{n-1} & & 2(T_{n-2} + T_{n-1})
\end{bmatrix}
\begin{bmatrix}
v_2 \\
& \ddots \\
v_{n-1}
\end{bmatrix}
= \\
\begin{bmatrix}
3 \frac{T_1^2}{T_1 T_2} \left[ (q_3 - q_2) + T_2^2 (q_2 - q_1) \right] - T_2 v_1 \\
3 \frac{T_2^2}{T_2 T_3} \left[ (q_4 - q_3) + T_3^2 (q_3 - q_2) \right] \\
& \ddots \\
3 \frac{T_{n-3}^2}{T_{n-3} T_{n-2}} \left[ (q_{n-1} - q_{n-2}) + T_{n-2}^2 (q_{n-2} - q_{n-3}) \right] \\
3 \frac{T_{n-2}^2}{T_{n-2} T_{n-1}} \left[ (q_{n} - q_{n-1}) + T_{n-1}^2 (q_{n-1} - q_{n-2}) \right] - T_{n-2} v_n
\end{bmatrix}
\]

i.e an equation in the form \( Av = c \)
Spline: algorithm

- Matrix $A$ is a dominant diagonal matrix and is always invertible provided that $T_k > 0$;

- Matrix $A$ has a tridiagonal structure: for these matrices there exist efficient numerical techniques (Gauss-Jordan method) for its inversion;

- Once the inverse of $A$ is known we might compute velocities $v_2,\ldots,v_{n-1}$ as:

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

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

Position

Velocity

Acceleration

\[ t_1 = 0 \quad t_2 = 2 \quad t_3 = 4 \quad t_4 = 8 \quad t_5 = 10 \]

\[ q_1 = 10^\circ \quad q_2 = 20^\circ \quad q_3 = 0^\circ \quad q_4 = 30^\circ \quad q_5 = 40^\circ \]
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:

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

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 bends.

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:

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
Trajectories in the operational space

Trajectory planning in the joint space yields unpredictable motions of the end-effector. When we want the motion to evolve along a predefined path in the operational space, it is necessary to plan the trajectory directly in this space.

Trajectory planning in the operational space entails both a path planning problem and a timing law planning problem: both the path and the timing law can be expressed analytically, as it will be shown in the following.

We will first address the trajectory planning for the position, and then we will concentrate on the orientation.
Path parameterization

Let us consider a parametric representation of a curve in space. The parameterization can be performed with respect to the **natural coordinate** (length of the arc of trajectory) :  \( p = p(s) \)

We can define the tangent, normal and binormal unit vectors:

\[
\begin{align*}
    t &= \frac{dp(s)}{ds} \quad \text{(unit tangent vector)} \\
    n &= \frac{d^2 p(s)/ds^2}{\left| d^2 p(s)/ds^2 \right|} \quad \text{(belongs to the osculating plane)} \\
    b &= t \times n \quad \text{(unit binormal vector)}
\end{align*}
\]
Linear path

As an example of path parameterization we can consider a segment in space (linear Cartesian path):

\[ p(s) = p_i + \frac{s}{\|p_f - p_i\|} (p_f - p_i) \]

\[ \frac{dp}{ds} = \frac{1}{\|p_f - p_i\|} (p_f - p_i) \]

\[ \frac{d^2p}{ds^2} = 0 \]

In this case it is not possible to define the frame \((t, n, b)\) uniquely.
Linear path

A linear path is completely characterized once two points in Cartesian space are given.
Concatenation of linear paths

Linear paths can be concatenated in order to obtain more elaborated paths.

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.

Formulas can be derived to define the blending (typically a parabolic one)
Circular path

A parametric representation of a circumference of radius ρ laying in a plane x‘ y’ and having the centre in the origin of such plane is:

\[ p'(s) = \begin{bmatrix} \rho \cos(s/\rho) \\ \rho \sin(s/\rho) \\ 0 \end{bmatrix} \]

Defining:

- \( c \) the vector that identifies the centre of the circumference in the base frame
- \( R \) the rotation matrix from base frame to frame x’ y’ z’

the general parametric representation of a circumference in space is:

\[ p(s) = c + Rp'(s) \]
Circular path

The circular path can also be defined assigning three points in space belonging to the same plane:
Position trajectories

For the position trajectories, taking into account the parameterization of the path with respect to the natural coordinate $p = p(s)$, we will assign the timing law through the function $s(t)$.

In order to determine function $s(t)$ we can use any of the timing laws (polynomials, harmonic, trapezoidal velocity profile, spline, etc.) already studied.

Also we notice that:

$$
\dot{p} = \dot{s} \frac{dp}{ds} = \dot{s} t \quad |\dot{s}| \quad \text{is then the norm of the velocity}
$$

For the segment:

$$
\dot{p} = \frac{\dot{s}}{\|p_f - p_i\|} (p_f - p_i) = \dot{s} t
$$

the time law $s(t)$ takes then an immediate meaning!
Orientation trajectories

For the **orientation planning** we might interpolate (e.g. linearly) the components of the unit vectors \( n(t), s(t) \) e \( a(t) \).

This procedure is however not advisable as the orthonormality of unit vectors cannot be guaranteed at all times.

An alternative way is to **interpolate three Euler angles**, using the following relations:

\[
\phi(s) = \phi_i + \frac{s}{\|\phi_f - \phi_i\|}(\phi_f - \phi_i)
\]

\[
\dot{\phi} = \frac{\dot{s}}{\|\phi_f - \phi_i\|}(\phi_f - \phi_i)
\]

\[
\ddot{\phi} = \frac{\ddot{s}}{\|\phi_f - \phi_i\|}(\phi_f - \phi_i)
\]

We can use any timing law \( s(t) \) on the natural parameter.

The angular velocity \( \omega \), which is linearly related to \( \phi \), has a continuous variation of the amplitude.

Poor predictability and understanding of the intermediate orientation!
Orientation trajectories

Orientation can also be planned by resorting to the axis/angle representation: if we assign two frames with the same origin but different orientation, it is always possible to determine a unit vector $r$ such that the second frame is obtained from the first one through a rotation of an angle $\vartheta_f$ around the axis of such unit vector.

Let $R_i$ and $R_f$ the rotation matrices, with respect to the base frame, of the initial frame and of the final frame. The rotation matrix between the two frames, with related axis/angle representation is thus:

$$
R_f^i = R_i^T R_f = \begin{bmatrix}
    r_{11} & r_{12} & r_{13} \\
    r_{21} & r_{22} & r_{23} \\
    r_{31} & r_{32} & r_{33}
\end{bmatrix}
$$

$$
\vartheta_f = \cos^{-1}\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right)
$$

$$
r = \frac{1}{2\sin(\vartheta_f)}\begin{bmatrix}
    r_{32} - r_{23} \\
    r_{13} - r_{31} \\
    r_{21} - r_{12}
\end{bmatrix}
$$
Define with $R^i(t)$ the matrix that describes the transition from $R_i$ to $R_f$. Then:

$$R^i(0) = I, \quad R^i(t_f) = R_f^i$$

Matrix $R^i(t)$ can be interpreted as $R^i(\vartheta(t), r)$, where:

- $r$ is constant and can be computed from the elements of $R_f^i$
- $\vartheta(t)$ can be made variable with time, through a timing law, with $\vartheta(0)=0$, $\vartheta(t_f)=\vartheta_f$

In order to characterize the orientation in the base frame it is then enough to compute:

$$R(t) = R_i R^i(t)$$
Orientation trajectories

Alternatively we can use \textit{unit quaternions} to specify the orientation. They are defined as:

\[ Q = \{ \eta, \, \vartheta \} \]

with:

\[ \eta = \cos \frac{\vartheta}{2} \]
\[ \omega = \sin \frac{\vartheta}{2} r \]

where \( \vartheta \) and \( r \) have the same meaning as with the axis/angle representation, but the representation singularity has been removed:

\[ \eta^2 + \varepsilon_x^2 + \varepsilon_y^2 + \varepsilon_z^2 = 1 \]
\[ \eta = \frac{1}{2} \sqrt{r_{11} + r_{22} + r_{33} + 1} \]
\[ \omega = \frac{1}{2} \begin{bmatrix} \text{sgn}(r_{32} - r_{23}) \sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \text{sgn}(r_{13} - r_{31}) \sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \text{sgn}(r_{21} - r_{12}) \sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \]
Elements of a motion planning and control system

- **Instruction stack**: list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation**: converts an instruction into a trajectory to be executed
- **Inverse kinematics**: maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives**: closes the control loop ensuring tracking performance
Inverse kinematics has already been discussed with reference to the study of the kinematics of the (possibly redundant) manipulator.

Options are:

- **Closed form** analytic solution (whenever possible)
- **Numerical solution** through (pseudo) inverse of the Jacobian
**Elements of a motion planning and control system**

- **Instruction stack**: list of instructions to be executed, specified using the proprietary programming language
- **Trajectory generation**: converts an instruction into a trajectory to be executed
- **Inverse kinematics**: maps the trajectory from the Cartesian space to the joint space (if needed)
- **Axis controllers & drives**: closes the control loop ensuring tracking performance (see next set of slides)