Robotics, vision and control

Notes from the a.y. 2021/2022 course held by Professors Riccardo Muradore, Francesco Visentin and Umberto Castellani

Author: Lorenzo Busellato

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Contents

List of Assignments

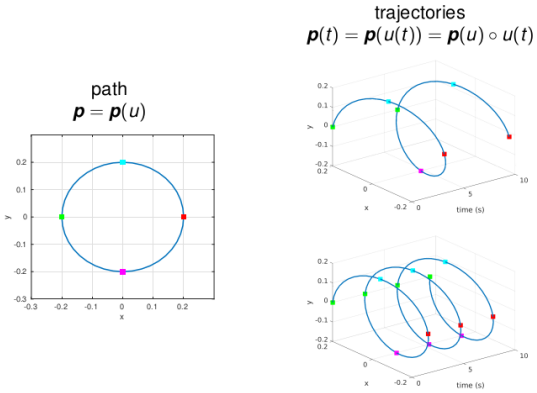
1 Trajectory planning

The goal of trajectory planning is the computation of a set of reference inputs to give a control system in order to make the end effector of a robotic manipulator accomplish some task.

Definition 1.1 (Geometric path)
*A **geometric path** is the locus of points either in the joint space (q) or the operative space (x).*

Definition 1.2 (Trajectory)
*A **trajectory** is a path on which a **motion law** is defined, e.g. in terms of velocity and/or accelerations at each point.*

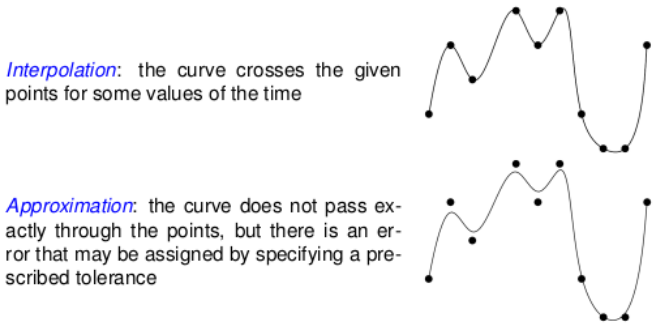
The difference between a path and a trajectory is then that a trajectory comes with a motion law attached to it, that allows us to describe not only position, but also velocity, acceleration and so on.



In general the inputs of the problem are the path description and constraints, the trajectory duration, the presence of obstacles etc. More often we only know the extreme points, maybe some intermediate points, and we can make some guesses about the geometrical primitives that can interpolate well the given points. Furthermore we have constraints on velocity and accelerations, that should either assume some value at some point or be continuous.

The outputs of the problem are the time sequences of the positions, velocities and accelerations that satisfy the constraint, if such a trajectory exists.

Another concept to keep in mind is the difference between **approximation** and **interpolation**:



1.1 Point-to-point trajectories

In the beginning we assume that we are in the scalar case, i.e. we consider only one actuator, and that no dynamic model is to be taken into account.

Starting from the initial joint configuration $q_i \in \mathbb{R}$ at time t_i and the final joint configuration $q_f \in \mathbb{R}$ at time t_f, we can try to find a polynomial function:

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

with $n + 1$ coefficients such that the initial and final constraints are respected. Note that the degree n of the polynomial is closely related to the constraints that need to be satisfied, as well as to the generic smoothness of the resulting trajectory.

1.1.1 Linear trajectory

In the **linear trajectory** interpolation we use a first order polynomial:

$$q(t) = a_0 + a_1(t - t_i) \quad t \in [t_i, t_f]$$

Given the conditions on the initial and final point of the trajectory, we can easily determine the coefficients of the polynomial:

$$\begin{cases} q(t_i) = q_i = a_0 \\ q(t_f) = q_f = a_0 + a_1(t_f - t) \end{cases} \implies \begin{cases} a_0 = q_i \\ a_1 = \frac{q_f - q_i}{t_f - t_i} = \frac{\Delta q}{\Delta T} \end{cases}$$

Note that while the acceleration profile is null during the trajectory, at the extremities it has an impulsive behavior that could not be acceptable.

1.1.2 Parabolic trajectory

The **parabolic trajectory** interpolation requires the composition of two second degree polynomials, one from time t_i to time t_m and the second from t_m to t_f.

$$t_m = \frac{t_i + t_f}{2} \qquad q(t_m) = q_m = \frac{q_i + q_f}{2}$$

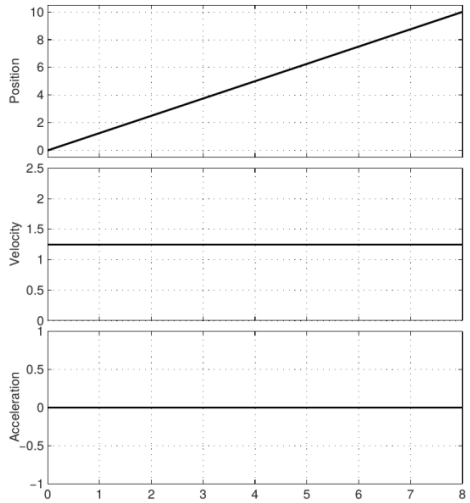
The trajectory is then divided into an acceleration period and a deceleration period:

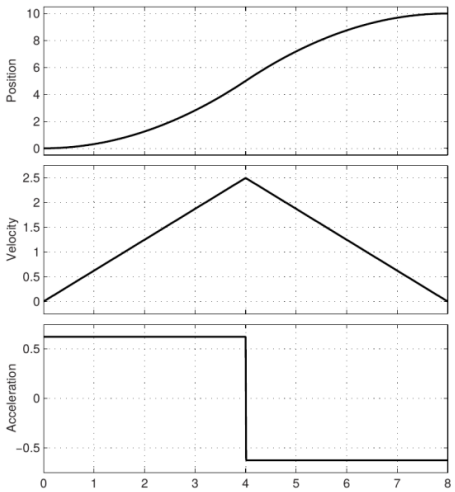
$$q(t) = \begin{cases} q_a(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 & t \in [t_i, t_m] \\ q_d(t) = a_3 + a_4(t - t_i) + a_5(t - t_i)^2 & t \in [t_m, t_f] \end{cases}$$

Having increased the degree of the polynomial we have an additional constraint available, i.e. the velocities at the initial and final points. The coefficients are then computed as follows:

$$\begin{cases} q_a(t_i) = q_i = a_0 \\ q_a(t_m) = q_m = a_0 + a_1(t_m - t_i) + a_2(t_m - t_i)^2 \\ \dot{q}_a(t_i) = \dot{q}_i = a_1 \\ q_d(t_m) = q_m = a_3 \\ q_d(t_f) = q_f = a_3 + a_4(t_m - t_i) + a_5(t_m - t_i)^2 \\ \dot{q}_d(t_f) = \dot{q}_f = a_4 + 2a_5(t_f - t_m) \end{cases} \implies \begin{cases} a_0 = q_i \\ a_1 = \dot{q}_i \\ a_2 = \frac{2 + (\Delta q - \dot{q}_i \Delta T}{2} \\ a_3 = q_m \\ a_4 = 2\frac{\Delta q}{\Delta T} - \dot{q}_f \\ a_5 = \frac{2(\dot{q}_f \Delta T - \Delta q}{\Delta T^2} \end{cases}$$

The resulting trajectories in the linear and parabolic cases are the following:





We can immediately see that increasing the polynomial degree results in a smoother trajectory. Note that in the parabolic profile we have a discontinuity in t_m if $\dot{q}_i \neq \dot{q}_f$. In any case the maximum velocity is obtained at the flex point. Acceleration is piecewise constant, while the jerk (i.e. the first derivative of the acceleration) is always null except at the flex point, where it impulsively reaches infinity.

1.1.3 Cubic polynomials

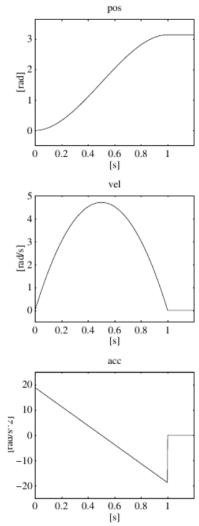
Increasing the degree once again we reach **cubic polynomials**:

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

Since we have 4 unknowns we need 4 constraints: $q_i, \dot{q}_i$, at t_i and $q_f, \dot{q}_f$ at t_f. If we apply them as before we obtain:

$$\begin{cases} a_0 = q_i \\ a_1 = \dot{q}_i \\ a_2 = \frac{3\Delta q - (2\dot{q}_i + \dot{q}_f)\Delta T}{\Delta T^2} \\ a_3 = \frac{-2\Delta q + (\dot{q}_i + \dot{q}_f)\Delta T}{\Delta T^3} \end{cases}$$

The resulting trajectory looks like the following:

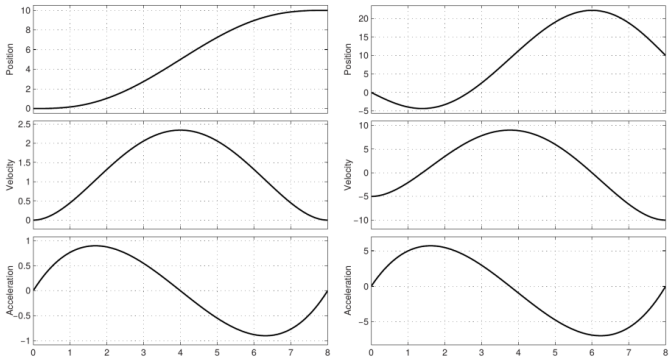


1.1.4 5th-order polynomials

We observe that discontinuities in the trajectory may generate vibrations in the robotic manipulator, due to the discontinuities being induced in the forces and due to elastic effects of the mechanical system. To solve this problem we need to be able to impose the initial and final accelerations, or in other words we need two additional constraints and therefore a polynomial with two more degrees, i.e. a **5th-order polynomial**:

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

By applying the constraints we can determine all coefficients and compute the trajectory, which is evidently much more smooth than the 3rd-order case:

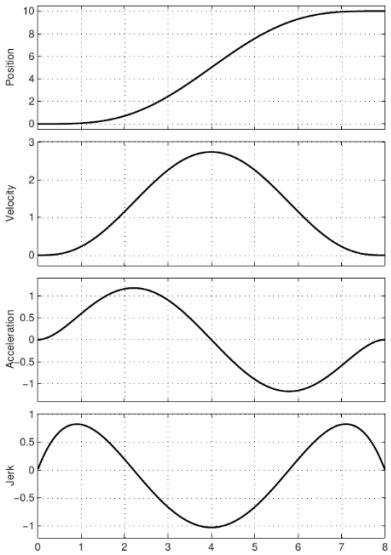


1.1.5 7th-order polynomials

It might be needed to also control the jerk, which in the 5th-order case would have impulsive discontinuities at the initial and final times, as well as every time the acceleration derivative changes in sign. As seen before, we need two additional constraints, that are given by a **7th-order polynomial**:

$$q(t) = a_7(t-t_i)^7 + a_6(t-t_i)^6 + a_5(t-t_i)^5 + a_4(t-t_i)^4 + a_3(t-t_i)^3 + a_2(t-t_i)^2 + a_1(t-t_i) + a_0$$

As we have done before, by setting the constraints and computing the coefficients we candraw the resulting trajectory:



As we can see the jerk is no longer discontinuous.

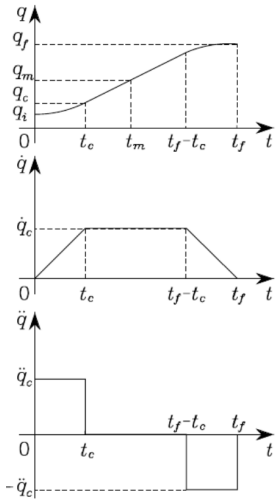
Assignment 1: Implement in Matlab 3rd- 5th- and 7th-order polynomials for $q_i > q_f$ and $q_i < q_f$[1].

1.2 Trapezoidal velocity profile

A popular solution in industrial robotics is the use of **trapezoidal velocity profiles**. Such profiles are made up of three phases:

1. Constant acceleration phase, with duration t_a.
2. Constant velocity phase
3. Constant deceleration phase, with duration t_d.

In the first and last phase a parabolic trajectory is computed, while in the middle phase a linear trajectory is used.



[1] https://github.com/lbusellato/univr/tree/master/Robotics,visionandcontrol/Assignments/Assignment1

As a first assumption, that we'll drop later, we'll set $t_a = t_d = t_c$, i.e. the acceleration and deceleration phases will last for the same amount of time. As such the trajectory will be feasible only if:

$$t_c \leq \frac{t_f - t_i}{2}$$

Furthermore we assume that $\dot{q}_i = \dot{q}_f = 0$.

By doing the computations for the three phases we obtain the position profile:

$$q(t) = \begin{cases} q_i + \frac{\dot{q}_c}{2t_c}(t - t_i)^2 & t_i \leq t \leq t_c \\ q_i + \dot{q}_c(t - \frac{t_c}{2})^2 & t_c \leq t \leq t_f - t_c \\ q_f - \frac{\dot{q}_c}{2t_c}(t_f - t)^2 & t_f - t_c \leq t \leq t_f \end{cases}$$

The unknowns are $t_c, \ddot{q}_c$ and $\dot{q}_c$. To determine them we can impose one of them as a constraint and derive the other two:

- Given t_c:

$$\ddot{q}_c = \frac{q_f - q_i}{t_c t_f - t_c^2} \qquad \dot{q}_c = \ddot{q}_c t_c$$

- Given $\ddot{q}_c$:

$$t_c = \frac{t_f}{2} - \frac{1}{2}\sqrt{\frac{t_2\ddot{q}_c - 4(q_f - q_i)}{\ddot{q}_c}} \qquad \dot{q}_c = \ddot{q}_c t_c$$

  Note that the argument of the square root must be positive for the trajectory to be feasible.

- Given $\dot{q}_c$:

$$t_c = \frac{q_i - q_f + \dot{q}_c t_f}{\dot{q}_c} \qquad \ddot{q}_c = \frac{\dot{q}_c^2}{q_i - q_f + \dot{q}_c t_f}$$

- Given $\ddot{q}_c$ and $\dot{q}_c$:

$$t_c = \frac{\dot{q}_c}{\ddot{q}_c} \qquad t_f = \frac{\dot{q}_c^2 + \ddot{q}_c(q_f - q_i)}{\dot{q}_c \ddot{q}_c}$$

  Note that in this case the final time t_f becomes an unknown.

Until now we assumed the final and initial velocities to be null. Relaxing this assumption changes the trajectory:

$$q(t) = \begin{cases} q_i + \dot{q}_i(t - t_i) + \frac{\dot{q}_c - \dot{q}_i}{2t_a}(t - t_i)^2 & t_i \leq t \leq t_a + t_i \\ q_i + \dot{q}_i\frac{t_a}{2} + \dot{q}_c(t - t_i - \frac{t_a}{2})^2 & t_a + t_i \leq t \leq t_f - t_d \\ q_f - \dot{q}_f(t_f - t) - \frac{\dot{q}_c - \dot{q}_f}{2t_d}(t_f - t)^2 & t_f - t_d \leq t \leq t_f \end{cases}$$

Note that now the acceleration phase duration t_a and deceleration phase duration t_d may differ.

Besides the initial and final velocities, we supply as a constraint the maximum acceleration we want the trajectory to reach. Before going on we need to establish whether or not such acceleration is feasible:

$$\ddot{q}_c^{max}\Delta q > \frac{\left|\dot{q}_i^2 - \dot{q}_f^2\right|}{2}$$

If the above holds, the trajectory is feasible, and we can compute the constant velocity:

$$\dot{q}_c = \frac{1}{2}(\dot{q}_i + \dot{q}_f + \ddot{q}_c^{max}\Delta T - \sqrt{(\ddot{q}_c^{max^2}\Delta T^2 - 4\ddot{q}_c^{max}\Delta q + 2\ddot{q}_c^{max}(\dot{q}_i + \dot{q}_f)\Delta T - (\dot{q}_i - \dot{q}_f)^2})$$

Where the argument of the square root must be positive for the trajectory to be feasible.

We must also perform a final check on the acceleration, that must be greater or equal than a limit value:

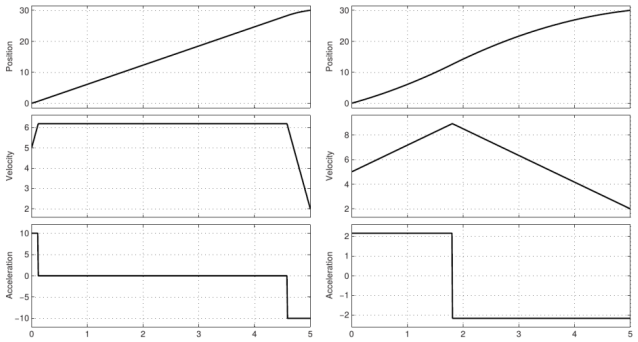
$$\ddot{q}_c^{max} \geq \ddot{q}_c^{lim} = \frac{2\Delta q - (\dot{q}_i + \dot{q}_f)\Delta T + \sqrt{4\Delta q^2 - 4\Delta q(\dot{q}_i + \dot{q}_f)\Delta T + 2(\dot{q}_i^2 + \dot{q}_f^2)\Delta T^2}}{\Delta T^2}$$

If $\ddot{q}_c^{max} = \ddot{q}_c^{lim}$ there is no constant velocity phase.

Finally we can compute the acceleration/deceleration periods:

$$t_a = \frac{\dot{q}_c - \dot{q}_i}{\ddot{q}_c^{max}} \qquad t_d = \frac{\dot{q}_c - \dot{q}_f}{\ddot{q}_c^{max}}$$

The resulting trajectory is:



On the left we see the case $\ddot{q}_c^{max} > \ddot{q}_c^{lim}$, while on the right the case $\ddot{q}_c^{max} = \ddot{q}_c^{lim}$.

If the maximum acceleration is feasible, we can also force the maximum velocity. We first check the following condition:

$$\ddot{q}_c^{max} \Delta q \lessgtr (\dot{q}_c^{max})^2 - \frac{\dot{q}_i^2 + \dot{q}_f^2}{2}$$

If the above expression is ">", then the maximum velocity is reached, and we compute the acceleration/deceleration periods as well as the trajectory duration:

$$\dot{q}_c = \dot{q}_c^{max} \quad t_a = \frac{\dot{q}_c^{max} - \dot{q}_i}{\ddot{q}_c^{max}} \quad t_d = \frac{\dot{q}_c^{max} - \dot{q}_f}{\ddot{q}_c^{max}} \quad \Delta T = \frac{\Delta q \ddot{q}_c^{max} + {\dot{q}_c^{max}}^2}{\ddot{q}_c^{max} \dot{q}_c^{max}}$$

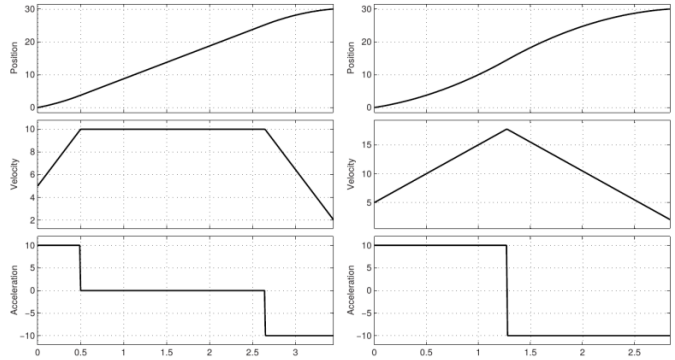
If the maximum velocity is not reached, i.e. if the previous expression is "<", then the maximum velocity of the trajectory is:

$$\dot{q}_c = \dot{q}_c^{lim} = \sqrt{\ddot{q}_c^{max} \Delta q + \frac{\dot{q}_i^2 + \dot{q}_f^2}{2}} < \dot{q}_c^{max}$$

And then the acceleration/deceleration periods:

$$t_a = \frac{\dot{q}_c^{lim} - \dot{q}_i}{\ddot{q}_c^{max}} \quad t_d = \frac{\dot{q}_c^{lim} - \dot{q}_f}{\ddot{q}_c^{max}}$$

For example:

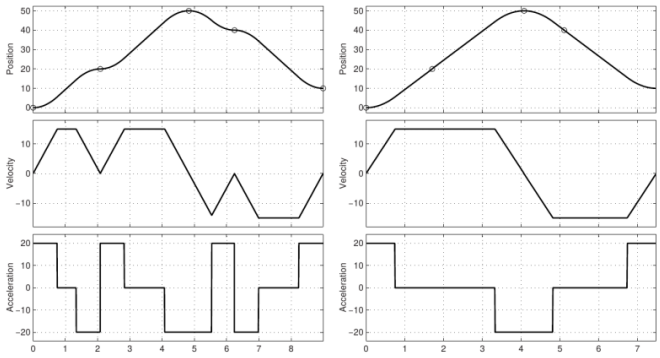


On the left we have $\dot{q}_c^{max} = 10$, while on the right we have $\dot{q}_c^{max} = 20$. Both trajectories have the same $\ddot{q}_c^{max} = 20$ and the same waypoints, although they are reached at different times.

If the goal is to design a trajectory that crosses a sequence of points, the velocity at each point can be either set to zero, or a value can be assigned to it with the following heuristic:

$$\dot{q}(t_i) = \dot{q}_i$$
$$\dot{q}(t_k) = \begin{cases} 0 & \text{if } sign(\Delta Q_k) \neq sign(\Delta Q_{k+1}) \\ sign(\Delta Q_k)\dot{q}^{max} & \text{if } sign(\Delta Q_k) = sign(\Delta Q_{k+1}) \end{cases}$$
$$\dot{q}(t_f) = \dot{q}_f$$

where $\Delta Q_k = q_k - q_{k-1}$. The difference between the two approaches is evident when we plot trajectories obtained with the two approaches:

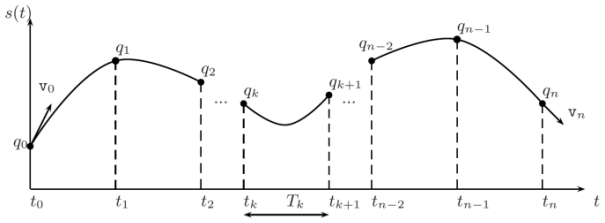


By applying the heuristic we stitch together the trapezoidal trajectories in a smarter way, resulting in a much smoother overall trajectory.

Assignment 2: Implement in Matlab the trapezoidal velocity profile generation taking into account all different constraints[2].

1.3 Multipoint trajectories

We will now define functions suitable for interpolating or approximating a set of given points $(t_k, q_k), k = 0, \ldots, n$.



Given $n+1$ pairs (t_k, q_k) we define the interpolating trajectory $s(t)$ as:

$$s(t) \coloneqq \{\Pi_k(t), t \in [t_k, t_{k+1}], k = 0, \ldots, n-1\}$$

where:

$$\Pi_k(t) = a_3^k(t-t_k)^3 + a_2^k(t-t_k)^2 + a_1^k(t-t_k) + a_0^k$$

Since n polynomials are needed to define a trajectory across $n+1$ points, the number of coefficients we need is $4n$. $2n$ conditions come from the pairs (t_k, q_k), since each polynomial crosses two of them at the extremities. $n-1$ conditions are given by the continuity of the velocities at the transition points, and $n-1$ conditions are given by the continuity of the accelerations.

All in all we have $2n + 2(n-1) = 4n - 2$ conditions, therefore we need to impose two additional constraints, that can be:

- The initial and final velocities $\dot{q}_0, \dot{q}_n$
- The initial and final velocities $\ddot{q}_0, \ddot{q}_n$
- The periodicity of the trajectory, i.e. $\dot{q}_0 = \dot{q}_n$, $\ddot{q}_0 = \ddot{q}_n$.

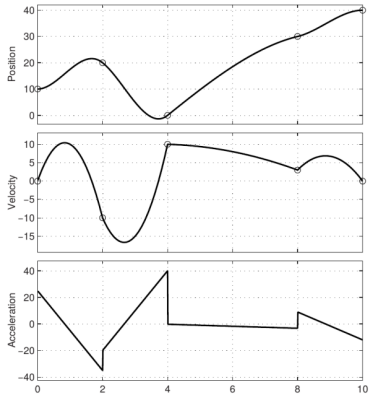
We start with imposing velocities on all trajectory points. Imposing such conditions results in:

$$\begin{cases} \Pi_k(t_k) & = a_0^k = q_k \\ \dot{\Pi}_k(t_k) & = a_1^k = \dot{q}_k \\ \Pi_k(t_{k+1}) & = a_3^k T_k^3 + a_2^k T_k^2 + a_1^k T_k + a_0^k = q_{k+1} \\ \dot{\Pi}_k(t_{k+1}) & = 3a_3^k T_k^2 + 2a_2^k T_k + a_1^k = \dot{q}_{k+1} \end{cases} \implies \begin{cases} a_0^k & = q_k \\ a_1^k & = \dot{q}_k \\ a_2^k & = \frac{1}{T_k}\left(\frac{3(q_{k+1}-q_k)}{T_k} - 2\dot{q}_k - \dot{q}_{k+1}\right) \\ a_3^k & = \frac{1}{T_k^2}\left(\frac{2(q_k-q_{k+1})}{T_k} + \dot{q}_k + \dot{q}_{k+1}\right) \end{cases}$$

where $T_k = t_{k+1} - t_k$.

[2] https://github.com/lbusellato/univr/tree/master/Robotics,visionandcontrol/Assignments/Assignment2

The resulting trajectory is:



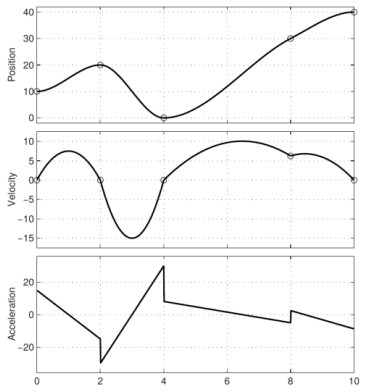
This approach is limited though, because usually the velocities at the path points are unknown. If we only know the initial and final velocities we can compute the intermediate velocities using **Euler's approximation**:

$$v_k = \frac{q_k - q_{k-1}}{t_k - t_{k-1}} \quad k = 1, \dots, n-1$$

Then:

$$\begin{aligned}
&\dot{q}(t_0) = \dot{q}_0 \\
&\dot{q}(t_k) = \begin{cases} 0 & \text{if } sign(\Delta Q_k) \neq sign(v_{k+1}) \\ \frac{v_k + v_{k+1}}{2} & \text{if } sign(v_k) = sign(v_{k+1}) \end{cases} \\
&\dot{q}(t_n) = \dot{q}_n
\end{aligned}$$

The resulting trajectory is very similar to what we obtained in the previous case:



The acceleration profile remains discontinuous. To fix this we need to impose the continuity of accelerations at the path points. Doing so results in the definition of a linear system $A\dot{q} = c$ where:

$$A = \begin{bmatrix} 2(T_0 + T_1) & T_0 & 0 & \dots & & \\ T_2 & 2(T_1 + T_2) & T_1 & \dots & & \\ & \ddots & \ddots & \ddots & & \\ & & T_{k+1} & 2(T_k + T_{k+1}) & T_k & \\ & & & \ddots & \ddots & \ddots \\ & & & & T_{n-1} & 2(T_{n-2} + T_{n-1}) \end{bmatrix} \in \mathbb{R}^{(n-1)\times(n-1)} \quad c = \begin{bmatrix} c_0 - T_1\dot{q}_0 \\ \vdots \\ c_k \\ \vdots \\ c_{n-2} - T_{n-2}\dot{q}_n \end{bmatrix} \in \mathbb{R}^{(n-1)}$$

and:

$$c_k = 2\left(\frac{T_{k+1}}{T_k}(q_{k+1} - q_k) + \frac{T_k}{T_k + 1}(q_{k+2} - q_{k+1})\right)$$

The system can be solved using linear algebra methods, i.e. by computing $\dot{q} = A^{-1}c$. A faster way comes from noticing that A is a tridiagonal matrix, i.e. it has elements only along the diagonal and immediately above and below it. Therefore $\dot{q}$ can be computed by using **Thomas' algorithm**.

Given:

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & & & \\ a_2 & b_2 & c_2 & 0 & \dots & & \\ 0 & \ddots & \ddots & \ddots & 0 & \dots & \\ & 0 & a_k & b_k & c_k & 0 & \dots \\ & & 0 & \ddots & \ddots & \ddots & 0 \\ & & & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_k \\ \vdots \\ d_n \end{bmatrix}$$

Thomas' algorithm is as follows:

for k=2:1:n **do** ▷ Forward elimination
  $m = \frac{a_k}{b_{k-1}}$
  $b_k = b_k - mc_{k-1}$
  $d_k = d_k - md_{k-1}$
end for
$x_n = \frac{d_n}{b_n}$
for k=2:1:n **do** ▷ Backwards substitution
  $x_k = \frac{d_k - c_k x_{k+1}}{b_k}$
end for

Assignment 3: Implement in Matlab the interpolating polynomials for the case in which velocity is imposed in all points of the sequence and in the case in which only initial and final velocities are imposed, using Thomas' algorithm to solve the linear system[3].

The computation can also be done with respect to the accelerations. Given:

$$s(t) = \{\Pi_k(t), t \in [t_k, t_{k+1}], k = 0, \dots, n-1\}$$

where $T_k = t_{k+1} - t_k$ and:

$$\Pi_k(t) = a_3^k(t - t_k)^3 + a_2^k(t - t_k)^2 + a_1^k(t - t_k) + a_0^k$$

The coefficients a_i^k of the polynomials can be expressed, as we saw, in terms of positions and velocities, or in terms of positions and accelerations:

$$\begin{cases} \Pi_k(t_k) & = a_0^k = q_k \\ \ddot{\Pi}_k(t_k) & = 2a_2^k = \ddot{q}_k \\ \Pi_k(t_{k+1}) & = a_3^k T_k^3 + a_2^k T_k^2 + a_1^k T_k + a_0^k = q_{k+1} \\ \ddot{\Pi}_k(t_{k+1}) & = 6a_3^k T_k + 2a_2^k = \ddot{q}_{k+1} \end{cases} \implies \begin{cases} a_0^k & = q_k \\ a_1^k & = \frac{q_{k+1} - q_k}{T_k} - \frac{\ddot{q}_{k+1} + 2\ddot{q}_k}{6} T_k \\ a_2^k & = \frac{\ddot{q}_k}{2} \\ a_3^k & = \frac{\ddot{q}_{k+1} - \ddot{q}_k}{6T_k} \end{cases}$$

And so we can express the k-th cubic polynomial $\Pi_k(t)$ of the spline as a function of the accelerations at the trajectory endpoints:

$$\Pi_k(t) = \left(\frac{q_{k+1}}{T_k} - \frac{\ddot{q}_{k+1}}{6} T_k\right)(t - t_k) + \left(\frac{q_k}{T_k} - \frac{\ddot{q}_k}{6} T_k\right)(t_{k+1} - t) + \frac{\ddot{q}_{k+1}}{6T_k}(t - t_k)^3 + \frac{\ddot{q}_k}{6T_k}(t_{k+1} - t)^3 \quad t \in [t_k, t_{k+1}]$$

By computing the velocity and acceleration from here and applying the given constraints, i.e. the continuity of accelerations and velocities:

$$\ddot{\Pi}_{k-1}(t_k) = \ddot{\Pi}_k(t_k) = \ddot{q}_k \qquad \dot{\Pi}_{k-1}(t_k) = \dot{\Pi}_k(t_k)$$

and the given initial and final velocities:

$$\dot{\Pi}_0(t_0) = \dot{q}_0 \quad \dot{\Pi}_{n-1}(t_n) = \dot{q}_n$$

[3] https://github.com/lbusellato/univr/tree/master/Robotics,visionandcontrol/Assignments/Assignment3

we arrive after some passages to the definition of the linear system:

$$A\ddot{q} = c$$

$$\begin{bmatrix} 2T_0 & T_0 & 0 & & & & \\ T_0 & 2(T_0+T_1) & T_0 & 0 & \dots & & \\ & \ddots & \ddots & \ddots & & & \\ & & T_k & 2(T_k+T_{k+1}) & T_{k+1} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & T_{n-2} & 2(T_{n-2}+T_{n-1}) & T_{n-1} \\ & & & & & T_{n-1} & 2T_{n-1} \end{bmatrix} \begin{bmatrix} \ddot{q}_0 \\ \ddot{q}_1 \\ \vdots \\ \ddot{q}_k \\ \vdots \\ \ddot{q}_{n-1} \\ \ddot{q}_n \end{bmatrix} = \begin{bmatrix} 6\left(\frac{q_1-q_0}{T_0} - \dot{q}_0\right) \\ 6\left(\frac{q_2-q_1}{T_2} - \frac{q_1-q_0}{T_0}\right) \\ \vdots \\ 6\left(\frac{q_n-q_{n-1}}{T_{n-1}} - \frac{q_{n-1}-q_{n-2}}{T_{n-2}}\right) \\ 6\left(\dot{q}_n - \frac{q_n-q_{n-1}}{T_{n-1}}\right) \end{bmatrix}$$

where $A \in \mathbb{R}^{(n+1)\times(n+1)}$ and $c, \ddot{q} \in \mathbb{R}^{(n+1)}$.

The system can be solved using Thomas' algorithm, and then the trajectory is computed with the computation based on the accelerations discussed above.

Cubic splines are continuous up to the second derivative, but it is not possible in general to assign both the initial and final velocities and the accelerations, therefore at the extremities the trajectory is characterized by discontinuities either on the velocities or on the accelerations.

To bypass this limitation we can either increase the degree of the polynomial primitive Π_k for the first and last tract using 5th-order polynomials, which have the drawback of increasing computational burden. Another approach is to add two extra points before the first and after the last point of the trajectory, computed appropriately to impose the desired constraints of velocity and acceleration in the extremities of the trajectory.

1.3.1 Smoothing cubic splines

Smoothing cubic splines are designed to approximate instead of interpolate a set of given data points:

$$q = \begin{bmatrix} q_0 & q_1 & \dots & q_{n-1} & q_n \end{bmatrix}^T \qquad t = \begin{bmatrix} t_0 & t_1 & \dots & t_{n-1} & t_n \end{bmatrix}^T$$

The approximated trajectory $s(t)$ is computed as the solution of a minimization problem, where the quantity to be minimized is a cost function L:

$$L = \mu \sum_{k=0}^{n} w_k (s(t_k) - q_k)^2 + (1-\mu) \int_{t_0}^{t_n} \ddot{s}(t)^2 dt \quad \mu \in [0,1]$$

The metric expresses the trade-off between two conflicting goals, the fitting of the given points (the part proportional to μ) and the smoothness of the trajectory, i.e. the minimization of its curvature and acceleration (the part proportional to $1-\mu$). μ is the parameter that weights this trade-off, while w_k is an arbitrary parameter that weights the fitting error, which can therefore be adjusted independently for each point. The closer w_k is to zero, the closer the resulting trajectory is to the actual interpolation in (t_k, q_k).

We can rewrite the cost function in matricial form as:

$$L = (q-s)^T W (q-s) + \lambda \ddot{s}^T A \ddot{s}$$

where $\lambda = \frac{1-\mu}{6\mu}, W = diag(w_0, w_1, \dots, w_k, \dots, w_{n-1}, w_n)$ and:

$$\begin{bmatrix} 2T_0 & T_0 & 0 & & & & \\ T_0 & 2(T_0+T_1) & T_0 & 0 & \dots & & \\ & \ddots & \ddots & \ddots & & & \\ & & T_k & 2(T_k+T_{k+1}) & T_{k+1} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & T_{n-2} & 2(T_{n-2}+T_{n-1}) & T_{n-1} \\ & & & & & T_{n-1} & 2T_{n-1} \end{bmatrix}$$

Furthermore we already know that $A\ddot{q} = c$, so substituting s and $\ddot{s}$ we obtain:

$$\begin{bmatrix} 2T_0 & T_0 & 0 & & & & \\ T_0 & 2(T_0+T_1) & T_0 & 0 & \dots & & \\ & \ddots & \ddots & \ddots & & & \\ & & T_k & 2(T_k+T_{k+1}) & T_{k+1} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & T_{n-2} & 2(T_{n-2}+T_{n-1}) & T_{n-1} \\ & & & & & T_{n-1} & 2T_{n-1} \end{bmatrix} \begin{bmatrix} \ddot{s}_0 \\ \ddot{s}_1 \\ \vdots \\ \ddot{s}_k \\ \vdots \\ \ddot{s}_{n-1} \\ \ddot{s}_n \end{bmatrix} = \begin{bmatrix} 6\left(\frac{s_1-s_0}{T_0} - \dot{s}_0\right) \\ 6\left(\frac{s_2-s_1}{T_2} - \frac{s_1-s_0}{T_0}\right) \\ \vdots \\ 6\left(\frac{s_n-s_{n-1}}{T_{n-1}} - \frac{s_{n-1}-s_{n-2}}{T_{n-2}}\right) \\ 6\left(\dot{s}_n - \frac{s_n-s_{n-1}}{T_{n-1}}\right) \end{bmatrix}$$

We can rewrite the right side as:

$$\begin{bmatrix} -\frac{6}{T_0} & \frac{6}{T_0} & 0 & & & & \\ \frac{6}{T_0} & -\left(\frac{6}{T_0}+\frac{6}{T_1}\right) & \frac{6}{T_1} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \frac{6}{T_k} & -\left(\frac{6}{T_k}+\frac{6}{T_{k+1}}\right) & \frac{6}{T_{k+1}} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \frac{6}{T_{n-2}} & -\left(\frac{6}{T_{n-2}+\frac{6}{T_{n-1}}}\right) & \frac{6}{T_{n-1}} \\ & & & & & \frac{6}{T_{n-1}} & -\frac{6}{T_{n-1}} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_k \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = Cs$$

We can then substitute into the cost function:

$$L = (q-s)^T W(q-s) + \lambda \ddot{s}^T A \ddot{s} = (q-s)^T W(q-s) + \lambda s^T C^T A^{-1} C s$$

The optimal solution is given by:

$$\nabla_s L(s) = -W(q-s) + \lambda C^T A^{-1} C s = 0 \implies s = (W + \lambda C^T A^{-1} C)^{-1} W q$$

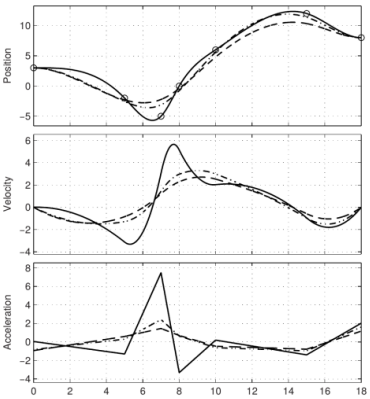
To actually prove that the solution is a minimum it is enough to compute the second derivative of L_s and check if it is positive.

Since W is diagonal it is non necessary to compute its inverse, as it is sufficient to invert each element along the diagonal. Furthermore we can simplify the formula we obtained by applying the matrix inversion lemma:

$$s = q - \lambda W^{-1} C^T (A + \lambda C W^{-1} C^T)^{-1} C q = q - \lambda W^{-1} C^T \ddot{s}$$

In this way not only we reduce the number of matrix inversions to be performed, but we also get the accelerations as an intermediate step.

The resulting trajectory will depend then on the value of μ, resulting closer to the interpolation as μ gets closer to 1, and on the weight matrix W, with the approximation error decreasing the smaller w_k is:



$\mu \in \{0.3, 0.6, 1\}$ $\quad$ $\boldsymbol{W}^{-1} = \text{diag}\{0, 1, \dots, 1, 0\}$

Assignment 4: Implement in Matlab the computation of cubic splines based on the accelerations with assigned initial and final velocities. Also compute the smoothing cubic splines[4].

[4] https://github.com/lbusellato/univr/tree/master/Robotics,visionandcontrol/Assignments/Assignment4

1.4 Complements on multipoint trajectories

We will now introduce some concepts that apply to all trajectory generation methods discussed so far.

1.4.1 Choice of time instants

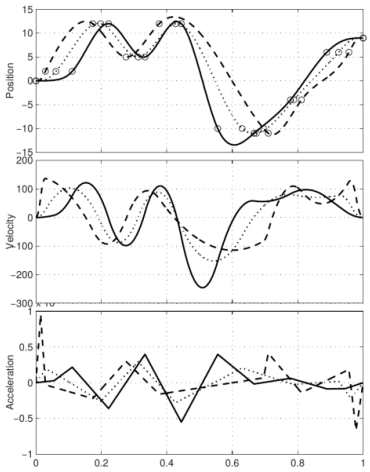
In several cases only the interpolation points $q_k, k = 0, \ldots, n$ and the initial and final times t_0, t_n are known. To choose the intermediate time instants we define a generic distribution for them over a normalized interval (i.e. $t_0 = 0$ and $t_n = 1$):

$$t_k = t_{k-1} + \frac{d_k}{d} \qquad d = \sum_{k=0}^{n-1} d_k$$

The main choices for d_k are:

- **Equally spaced points**:
$$d_k = \frac{1}{n-1}$$
- **Cord length distribution**:
$$d_k = |q_{k+1} - q_k|$$
- **Centripetal distribution**:
$$d_k = \sqrt{|q_{k+1} - q_k|}$$
- **Generic centripetal distribution**:
$$d_k = |q_{k+1} - q_k|^{\mu}$$

The choice of distribution obviously influences the resulting trajectory:



uniformly spaced (solid),
cord length (dashed),
centripetal (dotted)

1.4.2 Optimization of cubic trajectories

The total duration of a spline trajectory $s(t)$ interpolating the pairs $(t_k, q_k), k = 0, \ldots, n$ is:

$$T = \sum_{k=0}^{n-1} T_k = t_n - t_0 \qquad T_k = t_{k+1} - t_k$$

The goal is to minimize T such that the constraints on minimum/maximum velocity and acceleration are satisfied:

$$\{T_k^{\circ}\} = \underset{T_k}{argmin} T = \underset{T_k}{argmin} \sum_{k=0}^{n-1} T_k \qquad \begin{cases} |\dot{s}(t, T_0, T_1, \ldots, T_{n-1})| \leq \dot{q}^{max} \\ |\ddot{s}(t, T_0, T_1, \ldots, T_{n-1})| \leq \ddot{q}^{max} \end{cases}$$

Which is a non-linear optimum problem with a linear objective function.

Since the coefficients that determine the spline are functions of T_k, the optimization problem can be solved iteratively by scaling in time the segments that compose the spline.

If we replace T_k with λT_k then the splines become:

$$\dot{\Pi}_k(t) \to \frac{1}{\lambda}\dot{\Pi}_k(t) \quad \ddot{\Pi}_k(t) \to \frac{1}{\lambda^2}\ddot{\Pi}_k(t) \quad \dddot{\Pi}_k(t) \to \frac{1}{\lambda^3}\dddot{\Pi}_k(t)$$

Therefore the optimal λ is:

$$\lambda^\circ = max\{\lambda_v, \lambda_a, \lambda_j\}$$

where:

$$\lambda_v = \max_k \lambda_{v,k} \qquad \lambda_{v,k} = \max_{t\in[t_k,t_{k+1})} \frac{\left|\dot{\Pi}_k(t)\right|}{\dot{q}^{max}}$$

$$\lambda_a = \max_k \lambda_{a,k} \qquad \lambda_{a,k} = \left(\max_{t\in[t_k,t_{k+1})} \frac{\left|\ddot{\Pi}_k(t)\right|}{\ddot{q}^{max}}\right)^{\frac{1}{2}}$$

$$\lambda_j = \max_k \lambda_{j,k} \qquad \lambda_{j,k} = \left(\max_{t\in[t_k,t_{k+1})} \frac{|\dddot{\Pi}_k(t)|}{\dddot{q}^{max}}\right)^{\frac{1}{2}}$$

Depending on λ° the spline will then reach the maximum speed or the maximum acceleration or the maximum jerk in at least one point of the interval $[t_0, t_n]$.

1.4.3 Geometric modifications

It is useful to be able to manipulate geometrically a trajectory. The possible manipulations are:

- **Space translation** of a trajectory $q(t)$ from $(0,0)$ to (t_1, q_1):

$$\overline{q}(t) = q(t) + q_0$$
$$\overline{q}(t) : (0, q_0) \mapsto (t_1, q_0 + q_1)$$

- **Time translation** of a trajectory $q(t)$ from $(0,0)$ to (t_1, q_1):

$$\overline{q}(t) = q(t - t_0)$$
$$\overline{q}(t) : (0, q_0) \mapsto (t_0 + t_1, q_1)$$

- **Space reflection** of a trajectory $q(t)$ from $(0,0)$ to (t_1, q_1):

$$\overline{q}(t) = -q(t)$$
$$\overline{q}(t) : (0, q_0) \mapsto (t_1, -q_1)$$

- **Scaling in space** of a unitary trajectory $q(t)$ from $(0,0)$ to $(1,1)$:

$$\overline{q}(t) = q_0 + hq(t) \quad h = q_1 - q_0$$
$$\overline{q}(t) : (0, q_0) \mapsto (1, q_1)$$

- **Scaling in time** of a unitary trajectory $q(t)$ from $(0,0)$ to $(1,1)$:

$$\overline{q}(t) = q\left(\frac{t}{t_1}\right)$$
$$\overline{q}(t) : (0, q_0) \mapsto (t_1, 1)$$

1.4.4 Scaling in time

Scaling in time is particularly useful in order to satisfy the following constraints:

- **Kinematic saturation**, i.e. limits on velocity and acceleration:

$$|\dot{q}(t)| < \dot{q}^{max} \qquad |\ddot{q}(t)| < \ddot{q}^{max}$$

- **Dynamic saturation**, i.e. limits on the torques applied to the motors:

$$|\tau(t)| < \tau^{max}$$

Given the original trajectory $q(t)$ define a strictly increasing function σ:

$$t = \sigma(\bar{t})$$

such that the scaled trajectory:

$$\overline{q}(\bar{t}) = (q \circ \sigma)(\bar{t}) = q(\sigma(\bar{t})$$

has velocity and acceleration profiles:

$$\dot{\overline{q}}(\bar{t}) = \frac{dq(\sigma)}{d\sigma}\frac{d\sigma(\bar{t})}{d\bar{t}}$$
$$\ddot{\overline{q}}(\bar{t}) = \frac{dq(\sigma)}{d\sigma}\frac{d^2\sigma(\bar{t})}{d\bar{t}^2} + \frac{d^2q(\sigma)}{d\sigma^2}\left(\frac{d\sigma(\bar{t})}{d\bar{t}}\right)^2$$

that satisfy the kinematic constraints.
A common choice for σ is a linear one: $t = \lambda\bar{t}$ with $\lambda > 0$. Then we have:

$$\dot{\overline{q}}(\bar{t}) = \frac{dq(\sigma)}{d\sigma}\lambda = \lambda\dot{q}(t)$$
$$\ddot{\overline{q}}(\bar{t}) = \frac{d^2q(\sigma)}{d\sigma^2}\lambda^2 = \lambda^2\ddot{q}(t)$$
$$\dddot{\overline{q}}(\bar{t}) = \frac{d^3q(\sigma)}{d\sigma^3}\lambda^3 = \lambda^3\dddot{q}(t)$$

The optimal value for λ, i.e. the one that guarantees that the maximum values of speed, acceleration and jerk are not exceeded, is:

$$\lambda^\circ = min\left\{\frac{\dot{q}^{max}}{max_t|\dot{q}(t)|}, \sqrt{\frac{\ddot{q}^{max}}{max_t|\ddot{q}(t)|}}, \sqrt[3]{\frac{\dddot{q}^{max}}{max_t|\dddot{q}(t)|}}\right\}$$

Now let $\tilde{q}(\tau)$ be the normalized trajectory:

$$0 \le \tilde{q}(\tau) \le 1 \qquad 0 \le \tau \le 1$$

A generic trajectory from (t_0, q_0) to (t_1, q_1) can be written as:

$$q(t) = q_0 + (q_1 - q_0)\tilde{q}\left(\frac{t - t_0}{t_1 - t_0}\right) = q_0 + \Delta q\tilde{q}\left(\frac{t - t_0}{\Delta T}\right)$$

We then compute the derivatives of $q(t)$:

$$\dot{q}(t) = \frac{\Delta q}{\Delta T}\dot{\tilde{q}}(\tau) \qquad \ddot{q}(t) = \frac{\Delta q}{\Delta T^2}\ddot{\tilde{q}}(\tau) \qquad \dddot{q}(t) = \frac{\Delta q}{\Delta T^3}\dddot{\tilde{q}}(\tau)$$

Since the maximum values of velocity, acceleration and jerk of $q(t)$ are obtained in correspondence of the maximum values of velocity, acceleration and jerk of $\tilde{q}(t)$, it is easy to compute these values and the corresponding time instants τ from a given parametrization of $\tilde{q}$. By scaling ΔT we then obtain the motion profiles with maximum velocity or acceleration.

Example 1.1
Consider the normalized trajectory with null initial and final velocities:

$$\tilde{q}(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 = 3\tau^2 - 2\tau^3$$

Then:

$$\dot{\tilde{q}}(\tau) = 6\tau - 6\tau^2 \quad \ddot{\tilde{q}}(\tau) = 6 - 12\tau \quad \dddot{\tilde{q}}(\tau) = -12$$

and the maximum velocity and acceleration are:

$$\dot{\tilde{q}}^{max}(\tau) = \max_\tau \dot{\tilde{q}}(\tau) = \dot{\tilde{q}}(0.5) = \frac{3}{2} \qquad \ddot{\tilde{q}}^{max}(\tau) = \max_\tau \ddot{\tilde{q}}(\tau) = \ddot{\tilde{q}}(0) = 6$$

Therefore the maximums for a given trajectory is:

$$\dot{q}^{max} = \frac{3}{2}\frac{\Delta q}{\Delta T} \qquad \ddot{q}^{max} = 6\frac{\Delta q}{\Delta T^2}$$

We will now describe the dynamic constraints. The equations of motion of a robotic manipulator with n degrees of freedom are:

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

The i-th row is:

$$b_i^T(q)\ddot{q} + \frac{1}{2}\dot{q}^T L_i(q)\dot{q} + g_i(q(t)) = \tau_i \implies \tau_{s,i}(t) + \tau_{p,i}(t) = \tau_i(t)$$

where $\tau_{p,i}(t)$ depends only on the position.

We now consider a scaled version $\overline{q}(\bar{t}), \bar{t} \in [0, \overline{T}]$ of $q(t), t \in [0, T]$ with $t = \sigma(\bar{t})$. The corresponding torques $\overline{\tau}_i(t), i = 1, \dots n$ are:

$$\overline{\tau}_i(\bar{t}) = b_i^T(\overline{q}(\bar{t}))\ddot{\overline{q}}(\bar{t}) + \frac{1}{2}\dot{\bar{q}}(\bar{t})^T L_i(\overline{q}(\bar{t}))\dot{q} + g_i(\overline{q}(\bar{t}))$$

By exploiting the relationships:

$$\begin{aligned}
\overline{q}(\bar{t}) &= (\overline{q} \circ \sigma)(\bar{t}) \\
\dot{\bar{q}}(\bar{t}) &= \dot{q}(t)\dot{\sigma} = \dot{q}(t)\frac{d\sigma}{dt} \\
\ddot{\bar{q}}(\bar{t}) &= \ddot{q}(t)\dot{\sigma}^2 + \dot{q}(t)\ddot{\sigma} = \ddot{q}(t)\left(\frac{d\sigma}{dt}\right)^2 + \dot{q}(t)\frac{d^2\sigma}{dt^2}
\end{aligned}$$

We have:

$$\overline{\tau}_i(\bar{t}) = b_i^T(q(t))\dot{q}(t)\ddot{\sigma} + \left[b_i^T(q(t))\ddot{q}(t) + \frac{1}{2}\dot{q}^T(t)L_i(q(t))\dot{q}(t)\right]\dot{\sigma}^2 + g_i(q(t))$$

Since the term related to gravity is independent on time scaling, we can focus on the other terms:

$$\overline{\tau}_{s,i}(\bar{t}) = b_i^T(q(t))\dot{q}(t)\ddot{\sigma} + \left[b_i^T(q(t))\ddot{q}(t) + \frac{1}{2}\dot{q}^T(t)L_i(q(t))\dot{q}(t)\right]\dot{\sigma}^2 = b_i^T(q(t))\dot{q}(t)\ddot{\sigma} + \tau_{s,i}(t)\dot{\sigma}^2$$

In the linear scaling case, i.e. $t = \sigma(\bar{t}) = \lambda\bar{t}$:

$$\dot{\sigma}(\bar{t}) = \lambda \qquad \ddot{\sigma}(\bar{t}) = 0$$

Then:

$$\overline{\tau}_{s,i}(\bar{t}) = \tau_{s,i}(t)\lambda^2$$

And finally:

$$\overline{\tau}(\bar{t}) - g_i(\overline{q}(\bar{t})) = \lambda^2[\tau_i(t) - g_i(q(t))]$$

The scaling factor can be used to increase or decrease the duration of the trajectory in order to saturate at least one torque. Then:

$$\lambda^2 = min\left\{\frac{\tau_1^{max}}{|max_t\tau_1(t)|}, \frac{\tau_2^{max}}{|max_t\tau_2(t)|}, \dots, \frac{\tau_n^{max}}{|max_t\tau_n(t)|}\right\}$$

and:

$$\bar{t} = \frac{t}{\lambda} \qquad \overline{T} = \frac{T}{\lambda}$$

Note that the scaling factor λ is unique for the whole trajectory. A more efficient approach would be to scale down the time only in the intervals where one torque is larger than the corresponding maximum (**variable scaling**). With this formulation it is guaranteed that the kinematic constraints are saturated in at least a point of each segment of the trajectory.

1.5 Operational space trajectories

Operational space trajectories are the application of trajectory planning in the 3D Cartesian space.
To define a trajectory we need to define the **path**:

$$x_e(u) = \begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix}^T \in \mathbb{R}^3 \quad \Phi(u) = \begin{bmatrix} \varphi(u) & \theta(u) & \psi(u) \end{bmatrix}^T \in \mathbb{R}^3 \quad u \in [u_i, u_f]$$

and the **motion law**:

$$t = u(t) \quad t \in [t_i, t_f]$$

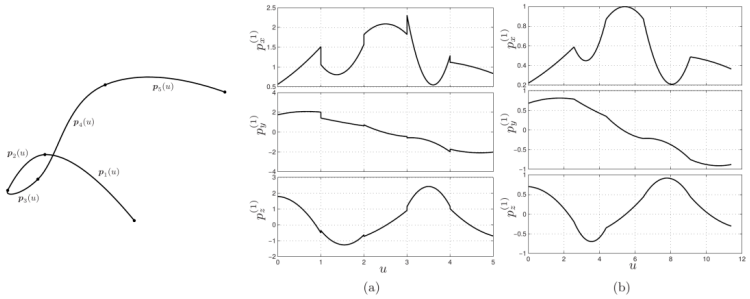
The combination of a path and a motion law defined on it makes up the trajectory. Given the trajectory, we need to use inverse kinematics to compute the corresponding joint space trajectory $q(t)$.
To define the two components of the trajectory we will use **motion primitives** for the geometric features of the path and **time primitives** for the motion law.

1.5.1 Differential geometry

Let $p \in \mathbb{R}^3$ be a Cartesian point given by $p = f(u)$ where $u \in [u_i, u_f]$. $p(u)$ is the parametric representation of the path Γ, with:

$$p_i \coloneqq p(u_i) = f(u_i) \quad \text{initial point}$$
$$p_f \coloneqq p(u_f) = f(u_f) \quad \text{final point}$$

Note that the geometric continuity of the path does not imply the parametric continuity of its components.



(a) (b)

Let $u \in [0, 1]$ and let:

$$\dot{p}(u) = \frac{dp(u)}{du}, \ddot{p}(u) = \frac{d^2p(u)}{du^2}, \ldots, p^{(i)}(u) = \frac{d^ip(u)}{du^2}$$

be the derivatives of the parametrization of f. Two infinitely differentiable segments $p_k(u), u \in [0, 1]$ and $p_{k+1}(u), u \in [0, 1]$ meeting at a common point:

$$p_k(1) = p_{k+1}(0)$$

satisfy the **n-order parametric continuity $\mathbf{C^n}$** if the first n parametric derivatives are continuous:

$$\begin{aligned} \dot{p}_k(1) &= \dot{p}_{k+1}(0) \\ \ddot{p}_k(1) &= \ddot{p}_{k+1}(0) \\ &\vdots \\ p_k^{(n)}(1) &= p_{k+1}^{(n)}(0) \end{aligned}$$

To define geometric continuity we need to consider the intrinsic properties of the curve, that are the **tangent unit vector**:

$$t = \frac{\frac{dp}{du}}{\left\| \frac{dp}{du} \right\|}$$

which is the vector of unitary norm tangent to the curve, and the **curvature unit vector**:

$$n = \frac{\frac{d^2p}{du^2}}{\left\| \frac{d^2p}{du^2} \right\|}$$

which is the vector of unitary norm normal to the curve.

Two parametric curves meet with a **first order geometric continuity $\mathbf{G^1}$** if and only if they have a common tangent unit vector. They meet with **$\mathbf{G^2}$ continuity** if they have common unit tangent and curvature vectors. They meet with **$\mathbf{G^n}$ continuity** if there exists a parametrization $\hat{u}$ equivalent to u such that $\hat{p}_k(\hat{u})$ and $\hat{p}_{k+1}(\hat{u})$ meet with C^n continuity.

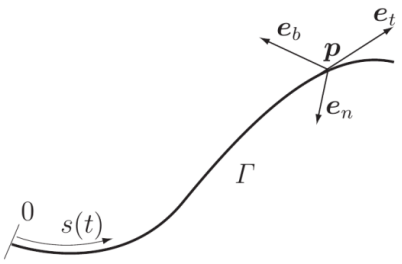
Two parametrizations u and $\hat{u}$ are **equivalent** if there exists a regular C^n function $f : [\hat{u}_{min}, \hat{u}_{max}] \mapsto [u_{min}, u_{max}]$ such that:

$$\begin{aligned} &\hat{p}(\hat{u}) = p(f(\hat{u})) = p(u) \\ &f([\hat{u}_{min}, \hat{u}_{max}]) = [u_{min}, u_{max}] \\ &\frac{df}{du} > 0 \end{aligned}$$

The **arc length** s of a generic point $p \in \Gamma$ is the length of the arc of Γ with extremes p and $p_i \in \Gamma$. The point p can be defined in terms of arc length:

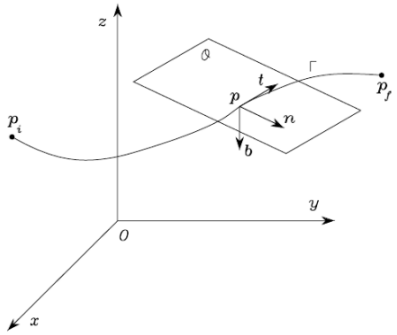
$$\Gamma : p = p(s) \qquad s \in [0, L]$$

where L is the total length of the curve.



To each point along the curve is associated a reference frame, called **Frenet frame**, which is defined by the tangent and curvature unit vector as well as by the **binormal unit vector**:

$$b = t \times n$$



The three vectors that make up the frame also identify the **osculating plane** $\mathcal{O}$, which is the limit plane containing t and a point $p' \in \Gamma$ when p' tends to p along the path.

1.5.2 Motion primitives

A **rectilinear path** is a linear segment connecting a point p_i and a point p_f. A possible parametrization is:

$$p(u) = p_i + (p_f - p_i)u \qquad u \in [0, 1]$$

while the arc length is:

$$p(s) = p_i + s\frac{p_f - p_i}{\|p_f - p_i\|} \qquad s \in [0, \|p_f - p_i\|]$$

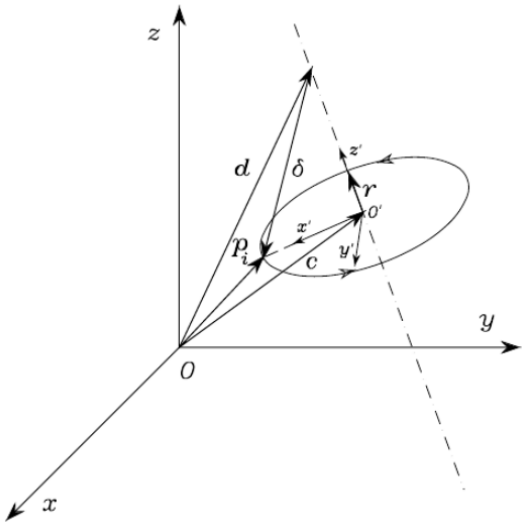
The Frenet frame is defined by:

$$t = \frac{dp}{ds} = \frac{p_f - p_i}{\|p_f - p_i\|}$$

The vectors b and n cannot be uniquely defined since the second derivative of the parametrization is null.

The resulting trajectory is composed by a set of linear segments, and is then continuous but with derivatives that are discontinuous in the intermediate points. To correct this we use **blending functions** to guarantee a smooth transition between consecutive segments.

For instance we can define a **circular path**:



The circle Γ is specified by assigning the unit vector of the circle axis r, the position vector d of a point along the circle axis and the position vector p_i of a point of the circle. The position vector c of the center of the circle is then given by:

$$c = d + (\gamma^T r)r$$

where $\gamma = p_i - d$. The radius is $\|p_i - c\|$.

To define a parametric representation of the circle as a function of the arc length s we consider the reference frame $\Sigma' = \{O', x', y', z'\}$ centered on the circle center and with x' oriented in the direction of the vector $p_i - c$, z' along r and y' oriented as to have a right handed frame (i.e. as a result of $z \times x$).

The arc length representation is then:

$$p'(s) = \begin{bmatrix} \rho\cos\left(\frac{s}{\rho}\right) \\ \rho\sin\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix} \qquad \rho \coloneqq \|p_i - c\|$$

On the base frame Σ the representation becomes:

$$p(s) = c + Rp'(s)$$

where $R = \begin{bmatrix} x' & y' & z' \end{bmatrix}$ is the rotation matrix that brings Σ' to Σ.

Since we are interested of moving along the path, we also need a parametric representation of the circle:

$$p'(u) = \begin{bmatrix} \rho\cos(u+\varphi) \\ \rho\sin(u+\varphi) \\ 0 \end{bmatrix} \qquad \rho := \|p_i - c\|, u \in [0, \theta]$$

and:

$$p(u) = c + Rp'(u) \quad u \in [0, \theta]$$

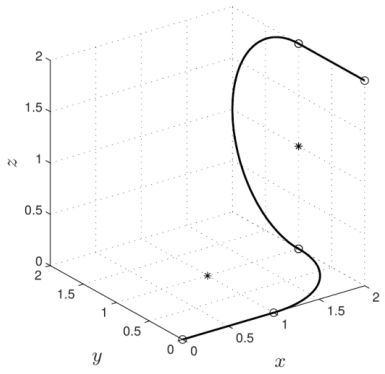
where:

$$p_0 = p(0) = c + Rp'(0)$$
$$p_1 = p(\theta) = c + Rp'(\theta)$$

In this case we can define the Frenet frame as:

$$t = \frac{dp}{ds} = R\begin{bmatrix} -\sin\left(\frac{s}{\rho}\right) \\ \cos\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix} \quad n = \frac{\frac{d^2p}{du^2}}{\left\|\frac{d^2p}{du^2}\right\|} = \frac{R}{\rho}\begin{bmatrix} -\cos\left(\frac{s}{\rho}\right) \\ -\sin\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix} \quad |b| = \frac{R^2}{\rho}$$

Assignment 5: Compute the 3D trajectory (also velocity, acceleration and jerk) in the picture as a combination of linear and circular motion primitives and compare it with the trajectory obtained using one of the multi-point methods[5].



So we can represent geometric paths with a parametric curve:

$$p(u) : [u_{min}, u_{max}] \mapsto \mathbb{R}^3$$
$$u(t) : [t_{min}, t_{max}] \mapsto [u_{min}, u_{max}]$$
$$\hat{p}(t) = (p \circ u)(t) : [t_{min}, t_{max}] \mapsto \mathbb{R}^3$$

We can then derive the velocity by applying the chain rule:

$$\dot{\hat{p}}(t) = \frac{dp}{du}\dot{u}(t)$$

The velocity vector is modulated by the velocity of motion law and its direction is the same as the tangent vector t.

In the same way we get the acceleration:

$$\ddot{\hat{p}}(t) = \frac{dp}{du}\ddot{u}(t) + \frac{d^2p}{du^2}\dot{u}^2(t)$$

which depends on both the acceleration and squared speed of the motion law. The acceleration vector has two components, one directed along the direction of t (**tangential acceleration**) and one directed along the direction of the normal vector n (**centripetal acceleration**).

[5] https://github.com/lbusellato/univr/tree/master/Robotics,visionandcontrol/Assignments/Assignment5

For instance when the motion law is a linear scaling of time:

$$u(t) = \lambda t \quad \lambda > 0$$

The k-th derivatives of the parametric equation are scaled by the factor λ^k:

$$\begin{aligned}
\dot{\hat{p}}(t) &= \frac{dp}{du}\lambda \\
\ddot{\hat{p}}(t) &= \frac{d^2p}{du^2}\lambda^2 \\
\dddot{\hat{p}}(t) &= \frac{d^3p}{du^3}\lambda^3 \\
&\vdots
\end{aligned}$$

As we have seen before we can enforce constraints on velocity, acceleration an jerk by choosing:

$$\lambda = min\left\{ \frac{\dot{\hat{p}}^{max}}{max_u\left\|\frac{dp}{du}\right\|}, \left(\frac{\ddot{\hat{p}}^{max}}{max_u\left\|\frac{d^2p}{du^2}\right\|}\right)^{\frac{1}{2}}, \left(\frac{\dddot{\hat{p}}^{max}}{max_u\left\|\frac{d^3p}{du^3}\right\|}\right)^{\frac{1}{3}} \right\}$$

Constant scaling cannot guarantee a smooth starting or ending, i.e. we cannot have both initial and final null velocities and accelerations. It is also not easy to find a motion law that satisfies constraints on acceleration and jerk since the derivatives of $u(t)$ appear mixed in the expressions for $\ddot{\hat{p}}(t)$ and $\dddot{\hat{p}}(t)$.

In a multipoint trajectory we can consider a single scaling for the whole trajectory:

$$u(t) = \lambda t \qquad t \in \left[\frac{u_k}{\lambda}, \frac{u_{k+1}}{\lambda}\right]$$

where $[u_k, u_{k+1}]$ is the parametric interval for the k-th segment.

Considering a linear segment connecting points p_i and p_f:

$$p(u) = p_i + (pf - p_i)u \quad u \in [0,1]$$

We can compute the derivatives of the parametric equation:

$$\begin{aligned}
\frac{dp}{du} = p_f - p_i &\implies \dot{\hat{p}}(t) = (p_f - p_i)\dot{u}(t) \\
\frac{d^2p}{du^2} = 0 &\implies \ddot{\hat{p}}(t) = (p_f - p_i)\ddot{u}(t) \\
\frac{d^3p}{du^3} = 0 &\implies \dddot{\hat{p}}(t) = (p_f - p_i)\dddot{u}(t)
\end{aligned}$$

So we can apply what we have seen for λ to the derivatives of the motion law, obtaining the following constraints:

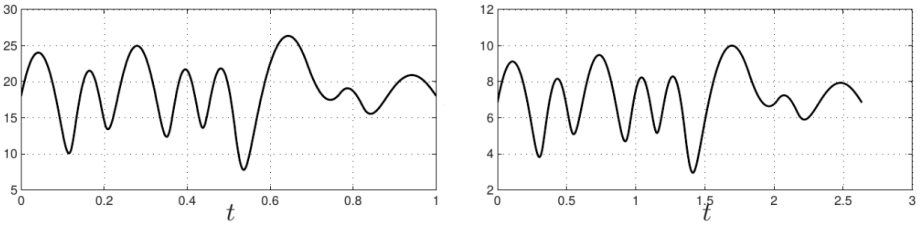
$$\begin{aligned}
|\dot{u}(t)| &\leq \frac{\dot{\hat{p}}^{max}}{\|p_f - p_i\|} \\
|\ddot{u}(t)| &\leq \frac{\ddot{\hat{p}}^{max}}{\|p_f - p_i\|} \\
|\dddot{u}(t)| &\leq \frac{\dddot{\hat{p}}^{max}}{\|p_f - p_i\|}
\end{aligned}$$

In several applications it is necessary to have a constant speed:

$$u(t) \quad \text{such that} \quad \left\|\dot{\hat{p}}(t)\right\| = v = constant, \forall t$$

For instance we could have a trajectory with $u(t) = t$, or more generically a scalable $u(t) = \lambda t$:

$\|\dot{\tilde{\boldsymbol{p}}}(t)\|$ with $u(t) = t$ $\|\dot{\tilde{\boldsymbol{p}}}(t)\|$ with $u(t) = \lambda t$



What we can do is to numerically compute $u(t)$ in real time at each time $t_k = kT_s$ using Taylor expansion:

$$u_{k+1} = u(t_{k+1}) = u_k + T_S \dot{u}_k + \frac{T_s^2}{2}\ddot{u}_k + \mathcal{O}(T_s^3 \ddot{u}_k)$$

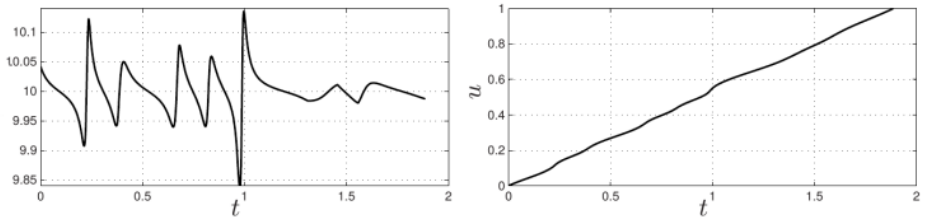
Since:

$$\dot{\tilde{p}}(t) = \frac{dp}{du}\dot{u}(t) \implies \dot{u}(t) = \frac{v}{\left\|\frac{dp}{du}\right\|}, \forall t$$

We obtain:

$$u_{k+1} = u_k + T_s \frac{v}{\left\|\frac{dp}{du}\,|_{u=u_k}\right\|}, \forall t = t_k$$

$\|\dot{\tilde{\boldsymbol{p}}}(t)\|$ with $u_{k+1} = u_k + T_s \frac{v}{\left\|\frac{dp}{du}|_{u=u_k}\right\|}$, $\forall t = t_k$

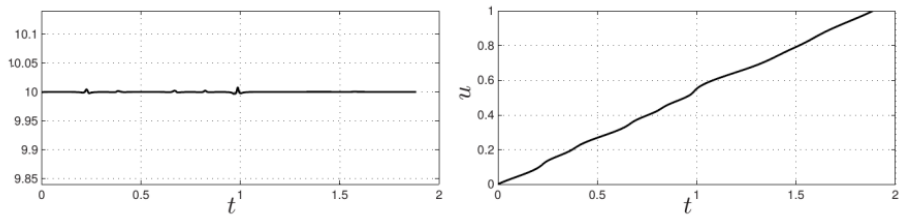


In other words the velocity is still not constant, but it varies little around the mean value.

A further improvement is obtained by using a second order approximation that takes into account the second derivative:

$$\ddot{u}(t) = -v^2 \frac{\frac{dp}{du}^T \frac{d^2p}{du^2}}{\left\|\frac{dp}{du}\right\|^4} \implies u_{k+1} = u_k + T_s \frac{v}{\left\|\frac{dp}{du}\,|_{u=u_k}\right\|} - \frac{(T_s v)^2}{2} \frac{\frac{dp}{du}^T \frac{d^2p}{du^2}}{\left\|\frac{dp}{du}\right\|^4}\,|_{u=u_k}, \forall t = t_k$$

$\|\dot{\tilde{\boldsymbol{p}}}(t)\|$ with $u_{k+1} = u_k + \frac{T_s v}{\left\|\frac{dp}{du}|_{u=u_k}\right\|} - \frac{(T_s v)^2}{2} \left.\frac{\frac{dp}{du}^T \frac{d^2p}{du^2}}{\|\frac{dp}{du}\|^4}\right|_{u=u_k}$, $\forall t = t_k$



The final result is a velocity that is not actually constant, but has negligible variations from the mean. While this approach is the best overall one, it is computationally expensive.

We can introduce an **uniform parametrization** based on the arc-length:

$$\|t(s)\| = \left\|\frac{dp}{ds}\right\| \rightarrow u(t) = vt$$

In the discrete case we have:

$$s_{k+1} = s_k + T_s v, \forall t = t_k$$

Suppose now that the velocity profile $v(t)$ and the acceleration profile $a(t)$ are known for the whole duration of the trajectory. The derivatives of u will be:

$$\dot{u}(t) = \frac{v(t)}{\left\|\frac{dp}{du}\right\|} \qquad \ddot{u}(t) = \frac{a(t)}{\left\|\frac{dp}{du}\right\|} - v^2(t)\frac{\frac{dp}{du}^T\frac{d^2p}{du^2}}{\left\|\frac{dp}{du}\right\|^4}$$

Let $v_k = v(t_k)$ and $a_k = a(t_k)$. Then:

$$u_{k+1} = u_k + \frac{T_s v_k}{\left\|\frac{dp}{du}\right\| |_{u=u_k}} + \frac{T_s^2}{2}\left(\frac{a_k}{\left\|\frac{dp}{du}\right\| |_{u=u_k}} - v_k^2\frac{\frac{dp}{du}^T\frac{d^2p}{du^2}}{\left\|\frac{dp}{du}\right\|^4}|_{u=u_k}\right), \forall t = t_k$$

We can simplify the expression with the arc-length parametrization:

$$s_{k+1} = s_k + T_s v_k + \frac{T_s^2}{2}\left(a_k - v_k^2\frac{dp}{du}^T\frac{d^2p}{du^2}\right), \forall t = t_k$$

1.6 End effector position

Let $x_e(t) = \begin{bmatrix} p_e \\ \Phi_e \end{bmatrix}$ be the pose of the manipulator's end effector at time t. Let $p_e(s) = f(s) \in \mathbb{R}^2$ be the position vector of the parametric representation of the path Γ, as a function of the arc length s with initial position p_i at $s, t = 0$ and final position p_f at $s = s_f$, $t = t_f$.

The evolution of p_e on Γ as a function of the timing law $s(t)$ is:

$$\dot{p}_e = \frac{dp_e}{dt} = \frac{dp_e}{ds} = \dot{s}\frac{dp_e}{ds} = \dot{s}t$$

The magnitude of p_e starts from zero at $t = 0$ and it varies with a generic profile (parabolic, trapezoidal, etc..) and finally returns to zero at $t = t_f$.

For a linear trajectory with $s \in [0, \|p_f - p_i\|]$ we have:

$$p_e(s) = p_i + s\frac{p_f - p_i}{\|p_f - p_i\|} \qquad \dot{p}_e(s) = \dot{s}\frac{p_f - p_i}{\|p_f - p_i\|} = \dot{s}t \qquad \ddot{p}_e(s) = \ddot{s}\frac{p_f - p_i}{\|p_f - p_i\|} = \ddot{s}t$$

For a circular trajectory we have:

$$p_e(s) = c + R\begin{bmatrix} \rho\cos\left(\frac{s}{\rho}\right) \\ \rho\cos\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix}$$

$$\dot{p}_e(s) = R\begin{bmatrix} -\dot{s}\sin\left(\frac{s}{\rho}\right) \\ \dot{s}\cos\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix} = Rt$$

$$\ddot{p}_e(s) = R\begin{bmatrix} -\dot{s}^2\frac{1}{\rho}\cos\left(\frac{s}{\rho}\right) - \ddot{s}\sin\left(\frac{s}{\rho}\right) \\ -\dot{s}^2\frac{1}{\rho}\sin\left(\frac{s}{\rho}\right) + \ddot{s}\cos\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix} = R\dot{s}\left\|\begin{bmatrix} -\frac{1}{\rho}\cos\left(\frac{s}{\rho}\right) \\ -\frac{1}{\rho}\sin\left(\frac{s}{\rho}\right) \\ 0 \end{bmatrix}\right\| n + R\dot{s}t$$

1.7 End effector orientation

The end effector orientation is specified in terms of the rotation matrix $R = \begin{bmatrix} n_e & s_e & a_e \end{bmatrix}$ of the end effector frame Σ_e with respect to the base frame Σ_b. A linear interpolation of the unit vectors n_e, s_e and a_e describing the initial and final orientations does not guarantee the orthonormality of the vectors at each time instant.

A possible solution is defining the end effector orientation Φ_e with respect to the Euler angles (φ, θ, ψ). The segment connecting the initial value Φ_i and the final value Φ_f with some timing law $s(t)$:

$$\Phi_e = \Phi_i + s\frac{\Phi_f - \Phi_i}{\|\Phi_f - \Phi_i\|} \quad \dot{\Phi}_e = \dot{s}\frac{\Phi_f - \Phi_i}{\|\Phi_f - \Phi_i\|} \quad \ddot{\Phi}_e = \ddot{s}\frac{\Phi_f - \Phi_i}{\|\Phi_f - \Phi_i\|}$$

Another approach is finding an unit vector such that the second frame can be obtained from the first frame with a rotation of a proper angle around the axis of such unit vector.

Let R_i be the rotation matrix of the initial frame $\Sigma_i = \{O_i, x_i, y_i, z_i\}$ with respect to the base frame and let R_f be the rotation matrix of the final frame $\Sigma_f = \{O_f, x_f, y_f, z_f\}$ with respect to the base frame. The rotation from Σ_f to Σ_i is then:

$$R_f^i = R_i^T R_f = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \theta_f = \cos^{-1}\left(\frac{r_{11} + r_{22} + r33 - 1}{2}\right) \qquad r = \frac{1}{2\sin\theta_f}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

$R^i(t) = R^i(\theta(t), r)$ describes the transition from R_i to R_f:

$$R^i(0) = R^i(\theta(0), r) = I \qquad R^i(t_f) = R^i(\theta(t_f), r) = R_f^i$$

where $\theta(t)$ is a generic timing law with $\theta(0) = 0$ and $\theta(t_f) = \theta_f$.

Since the rotation unit vector $r = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^T$ is constant, the only element that varies with time is $\theta(t)$. Then:

$$R^i(t) = R^i(\theta(t), r) = \begin{bmatrix} r_x^2(1-\cos\theta) + \cos\theta & r_x r_y(1-\cos\theta) - r_z\sin\theta & r_x r_z(1-\cos\theta) + r_y\sin\theta \\ r_x r_y(1-\cos\theta) + r_z\sin\theta & r_y^2(1-\cos\theta) + \cos\theta & r_y r_z(1-\cos\theta) - r_x\sin\theta \\ r_x r_z(1-\cos\theta) - r_y\sin\theta & r_y r_z(1-\cos\theta) + r_z\sin\theta & r_z^2(1-\cos\theta) + \cos\theta \end{bmatrix}$$

Moreover:

$$\omega^i(t) = \dot{\theta}(t)r \qquad \dot{\omega}^i(t) = \ddot{\theta}(t)r$$

And finally:

$$\begin{aligned} R_e(t) &= R_i R^i(t) \rightarrow \Phi_e(t) \\ \omega_e(t) &= R_i \omega^i(t) \rightarrow \dot{\Phi}_e(t) \\ \dot{\omega}_e(t) &= R_i \dot{\omega}^i(t) \rightarrow \ddot{\Phi}_e(t) \end{aligned}$$