Master's degree in Computer Engineering for Robotics and Smart Industry

Robotics Vision and Control
Final Report

Luca Ponti

June 2024

Contents

1 Project 1: Point to Point Polynomials 3

2 Project 2: 6

2.1 Point to Point with Trapezoidal velocity 6

2.2 Multi-point with Trapezoidal velocity 8

3 Multi point Trajectories 9

3.1 n-order polynomial interpolation 9

3.2 Cubic Spline interpolation with continuous velocity 9

3.3 Cubic Spline interpolation with continuous acceleration 11

4 Smoothing cubic splines 12

5 Multidimensional Trajectories and Geometric Path Planning 15

5.1 Motion Primitives 15

6 Geometry in 3D space 17

1 Project 1: Point to Point Polynomials

In this first part of the course we want to study the trajectory planning. The trajectory planning problem consists in finding a relationship between two elements belonging to different domains: time and space. Accordingly, the trajectory is usually expressed as a parametric function of the time,which provides at each instant the corresponding desired position.
From this definition we can exploit two very important concepts:

- **Geometric Path**: the locus of points in the joint space or in the operational space
- **Trajectory**: is a path on which a timing law is specified. In other words, is the modality by means of which the geometric path must be tracked, the motion law u = u(t).

In our study we will consider the trajectory described by polynomial functions:

$$q(t) = a_n t^n + a_{n-1}t^{n-1} + \cdots + a_2 t^2 + a_1 t + a_0, \quad t \in [t_i, t_f] \tag{1}$$

In order to find all the coefficients in the above equation we will use the initial and final constrains. Let us now dive into the core of the first project:

- **Linear Trajectory**: in this case we have only two unknown values, that are a_0 and a_1, so we will have only two equations to solve and it's sufficient to impose the constraint on the initial and final position, without touching the first or the second derivatives.

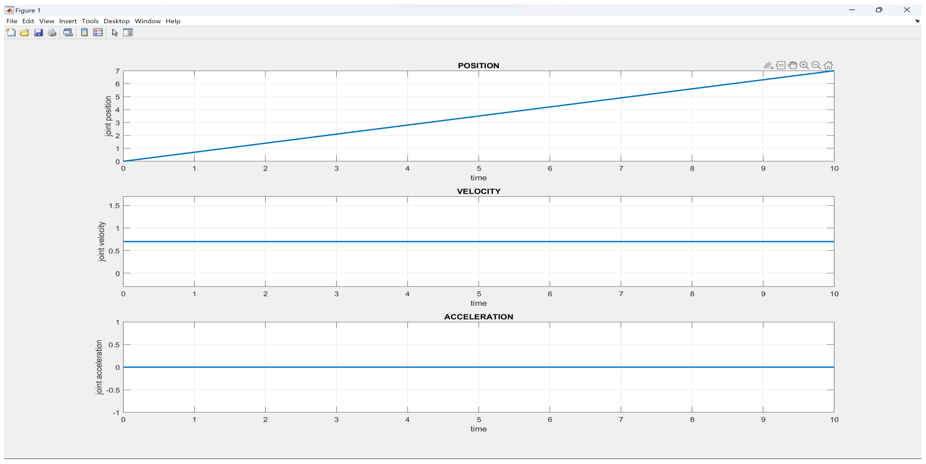

Figure 1: Linear Trajectory

- **Parabolic Trajectory**: In this case we have 3 unknowns for the acceleration period and 3 for the deceleration phase. So we need 3 equation for each phase. The first two equations are, as in the case of linear trajectory, the constrains on the initial and final position. The last one is given by the velocity at the initial point.

$$q_a(t) = a_0 + a_1(t - t_i) + a_2(t - t_i)^2 \quad t \in [t_i, t_m] \tag{2}$$

$$q_d(t) = a_3 + a_4(t - t_m) + a_5(t - t_m)^2 \quad t \in [t_m, t_f] \tag{3}$$

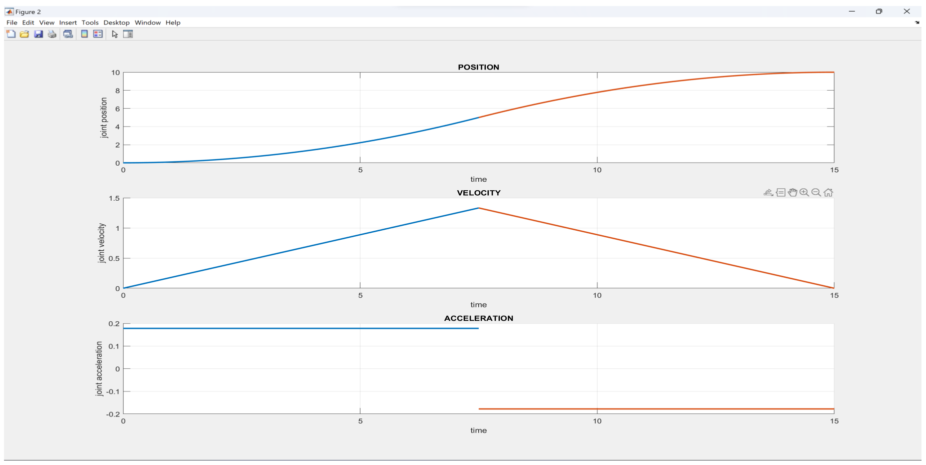

Figure 2: Parabolic Trajectory

- **Cubic Trajectory**: as before we note that here the number of unknowns is 4, therefore, we add an other equation given by the constrain on the velocity at the final point.

$$q(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \tag{4}$$

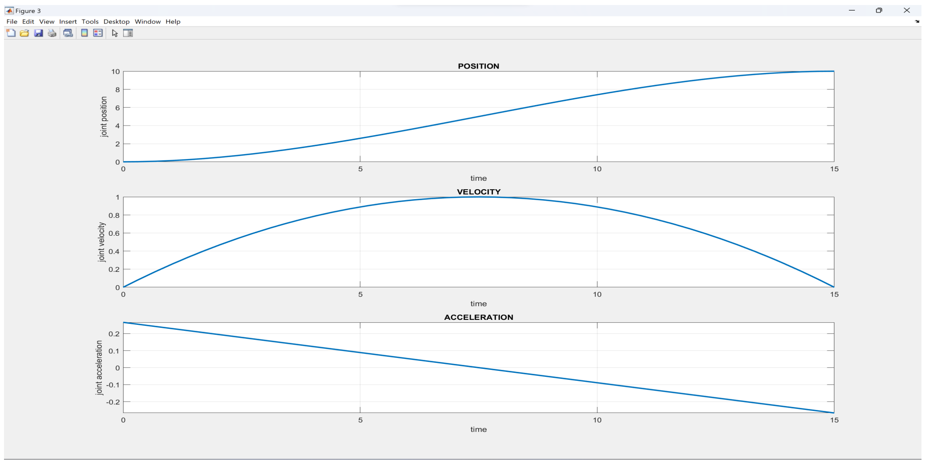

Figure 3: Cubic Trajectory

- **Fifth Order Polynomials**: in the same way of before we increase the number of equation, adding constrain on the derivative of the position and acceleration, in order to find every unknowns.

$$q(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0, \tag{5}$$

- **Seventh order Polynomials**: here we have seven unknowns, so we impose also the constrain on the derivative of the acceleration (Jerk). In this case, as we can see in the image, also the jerk is continuous.

$$q(t) = a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0, \tag{6}$$

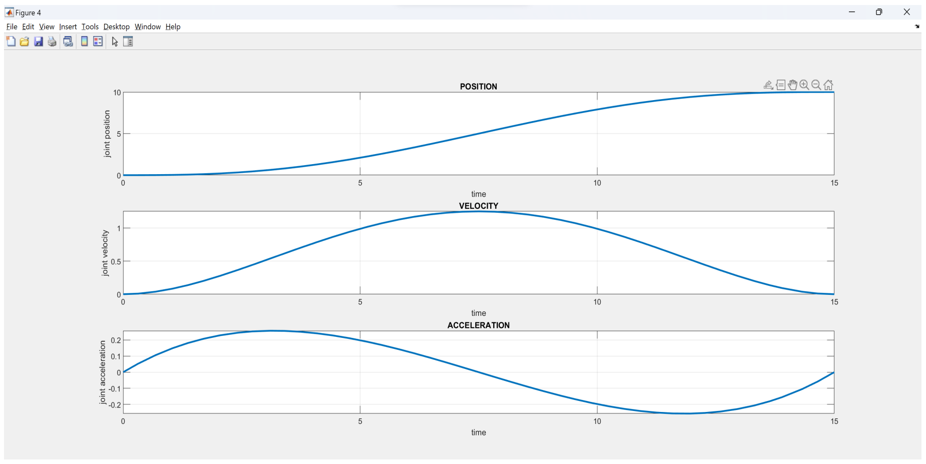

Figure 4: 5th order Trajectory

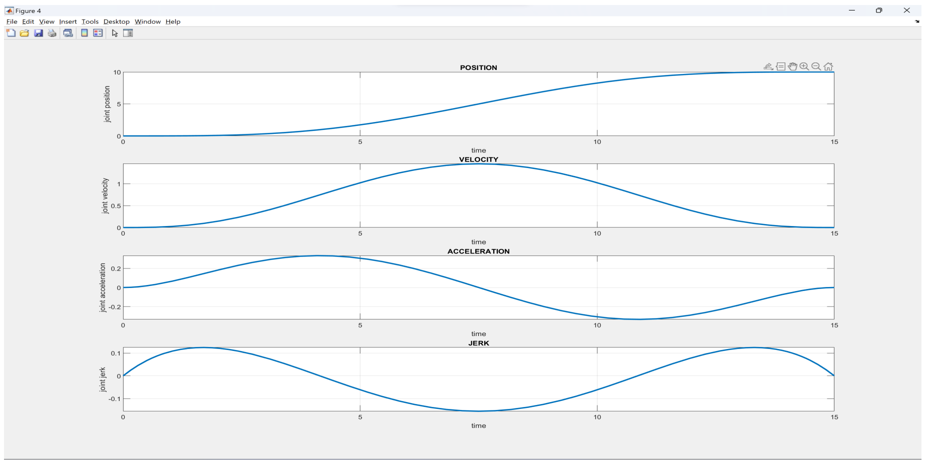

Figure 5: 7th order Trajectory

2 Project 2:

2.1 Point to Point with Trapezoidal velocity

As we saw in the first project, with a parabolic trajectory we have a continuous velocity, that is very good for us, but now we want to control in a better way the velocity part: in particular we want to use the trapezoidal velocity profiles. This solution is popular in real-world applications, such as in robots. The trajectory is divided into 3 parts:

- constant acceleration in the start phase: the acceleration is positive and constant, and therefore the velocity is a linear function of time and the position is a parabolic curve

- constant cruise velocity: the acceleration is null, the velocity is constant and the position is a linear function of time
- constant deceleration in the arrival phase: the acceleration is negative and constant, and therefore the velocity is a linear function of time and the position is a parabolic curve with negative curvature

Constrains are:

- initial and final time interval
- initial and final position
- acceleration and deceleration duration

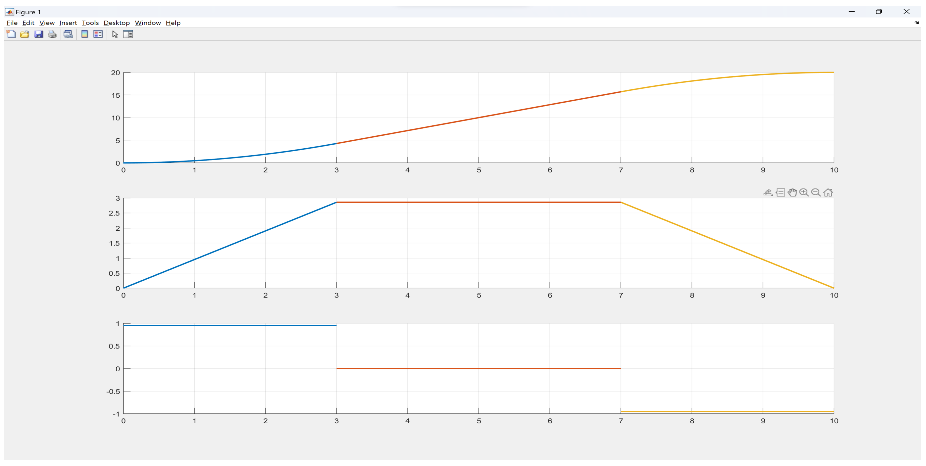

Figure 6: Trapezoidal velocity profile with constant acceleration and deceleration

Interestingly, if we set the acceleration time exactly equal to the deceleration time and both equal to half the total time of the trajectory we return to the case of the first design, this is because the constant speed phase is hidden and its duration is zero.

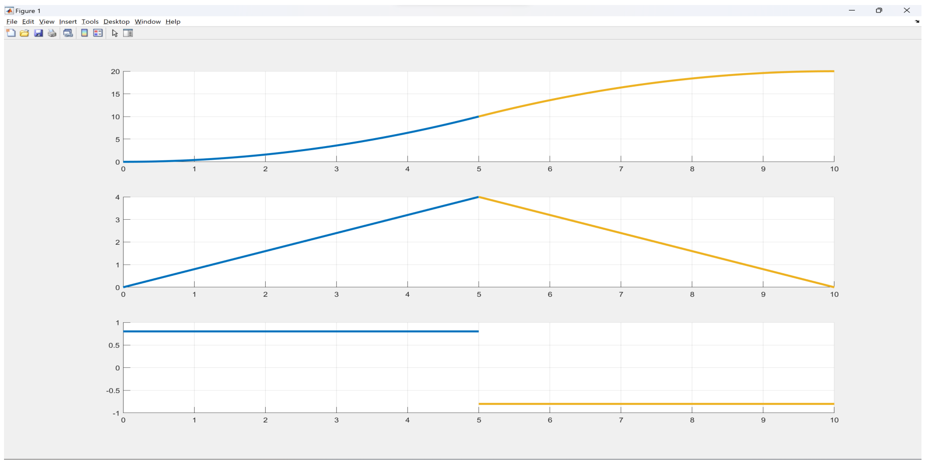

Figure 7: Trapezoidal velocity profile without constant velocity phase

2.2 Multi-point with Trapezoidal velocity

So far we have only studied point-to-point trajectories, but in the real case we have more complex trajectories with more than two points. In this part we study multi-point trajectories with trapezoidal velocity profile. An important observation in this part is that all motions must be defined according to the slowest, or the one with the largest displacement, because all pieces of the trajectory must end at the same time, otherwise the final goal is not reached. In other words, we slow down all trajectories according to the slowest. In the particular case of the image below, we set the constrain on the maximum velocity.

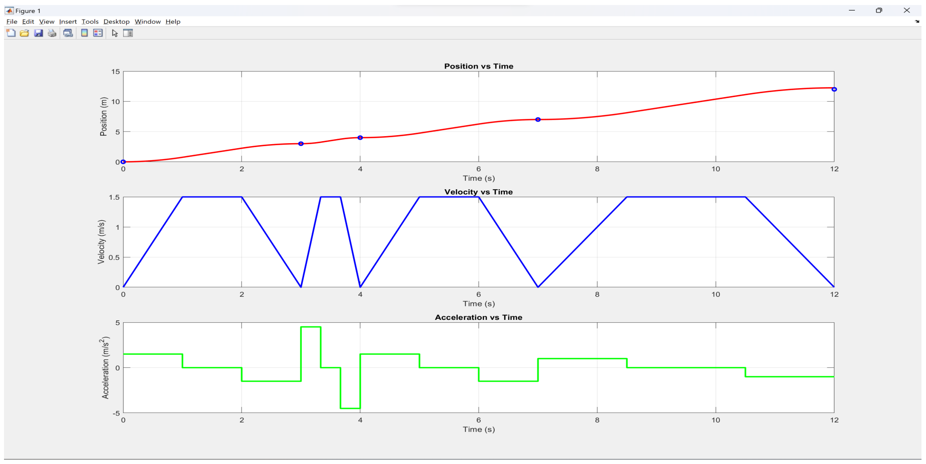

Figure 8: multi-point Trapezoidal velocity profile

3 Multi point Trajectories

In robotics, planning the movement of a robot to pass through multiple predefined points, or waypoints, is a critical task. This process, known as trajectory planning, is crucial to ensure that robots move smoothly and accurately to perform their tasks. Two common approaches that we will use to generating these trajectories are n-order polynomial interpolation and cubic spline interpolation.

3.1 n-order polynomial interpolation

Polynomial interpolation fits a single polynomial of degree n to the set of waypoints. Where n is the number of waypoints.
This method is very simple but it has significant disadvantages. Higher order polynomials may have undesirable oscillations between waypoints. This can result jerky movements for the robot, which are undesirable in most applications. In addition, ensuring smooth transitions in both velocity and acceleration is difficult with a single high-order polynomial. In the particular case of the exercise we have 6 waypoints and in fact the degree of the polynomial is 6. As can be seen, between the penultimate and the last point we have a strange behavior, due to the fact that the more we increase the oscillation and the uncontrollable motion. As can be seen, between the penultimate and the last point we have a strange behavior, due to the fact that the more we increase the order of the polynomial the more we will have oscillation and uncontrollable movement between the points. For this reason this strategy is never used.

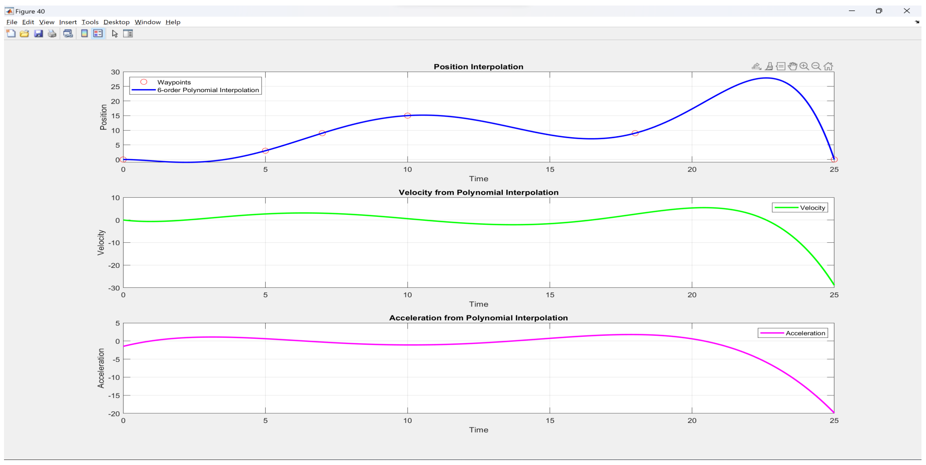

Figure 9: n-order polynomial interpolation

3.2 Cubic Spline interpolation with continuous velocity

Cubic spline interpolation overcomes many limitations of n-order polynomial interpolation by using piece wise cubic polynomials. This method ensures that the trajectory passes smoothly through all waypoints, with better control of continuity. In this particular problem, we have 4 unknowns because we are dealing with a third-order polynomial and the equation is:

$$q(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \tag{7}$$

So we need 4 constraints to find the desired trajectory:

- position at time t_k of each segment equal to the position of the k-th waypoint
- velocity at time t_{k+1} of the following segment with respect to the current one is equal to the velocity of the k-th +1 waypoint (calculated with the Euler approximation)
- position at time t_{k+1} of the current segment is equal to the position of the k-th +1 waypoint (the following)
- velocity at time t_{k+1} of the current segment is equal to the velocity of the k-th +1 waypoint (the following)

In other, simpler words, we impose that the position of two adjacent segments is equal and the same for the velocity. in this way we ensure the continuity of the derivative before the position.

Applying these 4 constraints says nothing about acceleration and in fact, as we shall see, it will be discontinuous.
At this point it is natural to ask how the velocity between the different segments is calculated.
One possible way, which is the one we will follow, is to use the Euler approximation in which the velocity is calculated in this way:

$$v_k = \frac{q_{k+1} - q_k}{t_{k+1} - t_k} \tag{8}$$

Then, we decide the velocity for each segment with this rule:

$$\text{Velocity} = \begin{cases} 0 & \text{if } \operatorname{sign}(v_k) \neq \operatorname{sign}(v_{k+1}) \\ \frac{v_k + v_{k+1}}{2} & \text{if } \operatorname{sign}(v_k) = \operatorname{sign}(v_{k+1}) \end{cases} \tag{9}$$

and the results is this one, where the velocity is continuous:

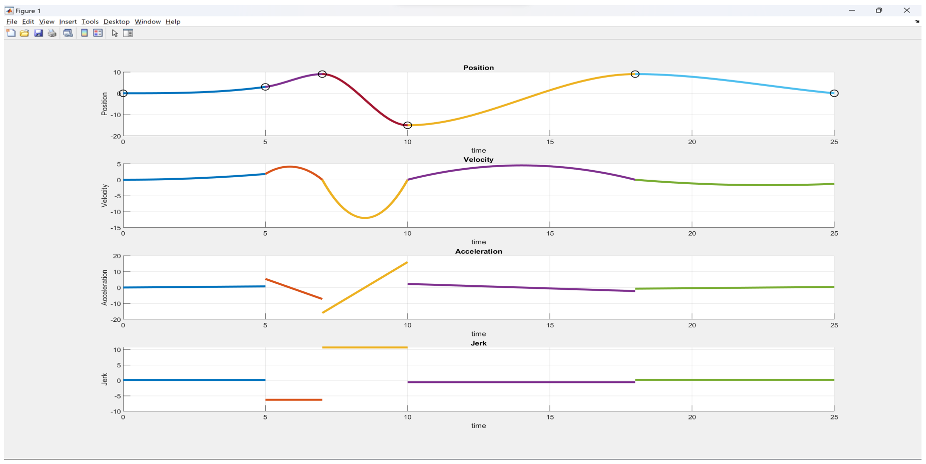

Figure 10: cubic spline multi point interpolation with continuous velocity

In the figure above we have 5 segments of different color, this is because we implemented the spline manually, from this point on we will use a MATLAB function for easier and faster calculation.

3.3 Cubic Spline interpolation with continuous acceleration

These splines also ensure that the second derivative of the trajectory (representing acceleration) is continuous. This results in even smoother trajectories, minimizing jerks and providing a more refined motion profile. Continuous acceleration is especially important in applications that require high accuracy.

Here the constraints are again 4:

- position of the previous segment with respect to the current one equal to the position of the k-th waypoint
- initial position of each segment corresponding to the final position of the previous segment
- initial velocity of each segment corresponding to the final velocity of the previous segment
- initial acceleration of each segment corresponding to the final acceleration of the previous segment

Unlike before, here we do not have enough equations, in fact we have only $4n-2$ equations in $4(n-1)$ unknowns. In fact, the conditions for the initial position of the first segment at time t_1 and the final position of the final segment at time t_n cannot be included because they are linearly dependent on the previous one.

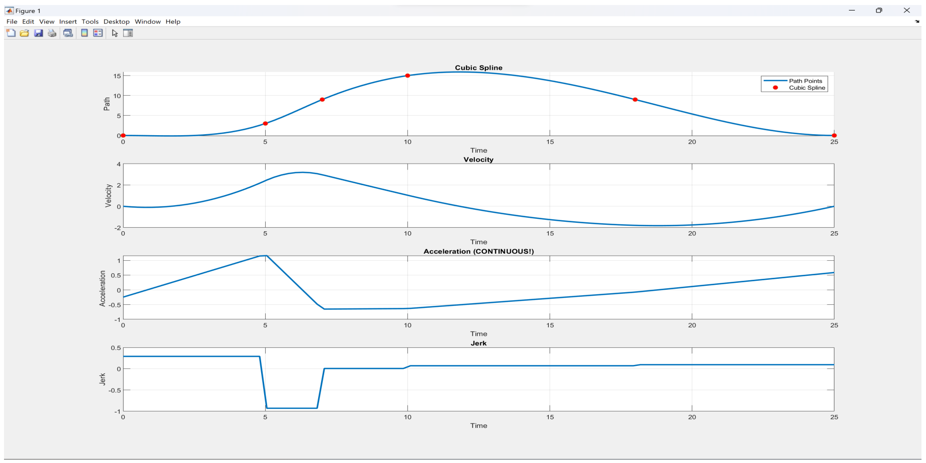

Figure 11: cubic spline multi point interpolation with continuous acceleration

4 Smoothing cubic splines

Smoothing cubic splines are defined in order to approximate, and not to interpolate, a set of given data points.
In particular, this kind of trajectory is adopted to find a tradeoff between two apposite goals:

- a good fit of the given via-points
- a trajectory as smooth as possible, so with acceleration as small as possible

So the purpose of this part is to minimize the following function:

$$L := \mu \sum_{k=0}^{n} w_k \left(s(t_k) - q_k\right)^2 + (1-\mu) \int_{t_0}^{t_n} \ddot{s}(t)^2 \, dt \tag{10}$$

where the parameter $\mu \in [0,1]$ reflects the different importance given to the two conflicting objectives that we explained earlier. On the other hand, the parameter w_k can be chosen arbitrarily to change the weight given to the fit of intermediate points and the smoothness term.

So, after some mathematical steps we end up with a matrix expression:

$$\mathbf{s} = \mathbf{q} - \lambda \mathbf{W}^{-1}\mathbf{C}^T \left(\mathbf{A} + \lambda \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^T\right)^{-1} \mathbf{C}\mathbf{q} \tag{11}$$

where the bold terms are matrices. In particular:

$$A = \begin{bmatrix} 2T_0 & T_0 & 0 & \cdots & 0 & 0 \\ T_0 & 2(T_0+T_1) & T_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & T_{n-2} & \cdots & 2(T_{n-2}+T_{n-1}) & T_{n-1} \\ 0 & 0 & 0 & \cdots & T_{n-1} & 2T_{n-1} \end{bmatrix} \tag{12}$$

$$C = \begin{bmatrix} -\frac{6}{T_0} & \frac{6}{T_0} & 0 & \cdots & 0 & 0 & 0 \\ \frac{6}{T_0} & -\left(\frac{6}{T_0}+\frac{6}{T_1}\right) & \frac{6}{T_1} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \frac{6}{T_k} & -\left(\frac{6}{T_k}+\frac{6}{T_{k+1}}\right) & \frac{6}{T_{k+1}} & 0 & \\ 0 & 0 & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & \frac{6}{T_{n-2}} & -\left(\frac{6}{T_{n-2}}+\frac{6}{T_{n-1}}\right) & \frac{6}{T_{n-1}} \\ 0 & 0 & 0 & \cdots & 0 & \frac{6}{T_{n-1}} & -\frac{6}{T_{n-1}} \end{bmatrix}$$

$$\mathbf{W} = \mathrm{diag}\{w_0, w_1, \ldots, w_{n-1}, w_n\} \tag{13}$$

The matrix **A** and **C** are tri-diagonal and for create them, in the code we used the ”diag” MATLAB function. The results of all these calculations are not as expected and I don’t understand why. So, after trying for a long time, we used the MATLAB function “csaps”, and the results are showed in the pictures below.

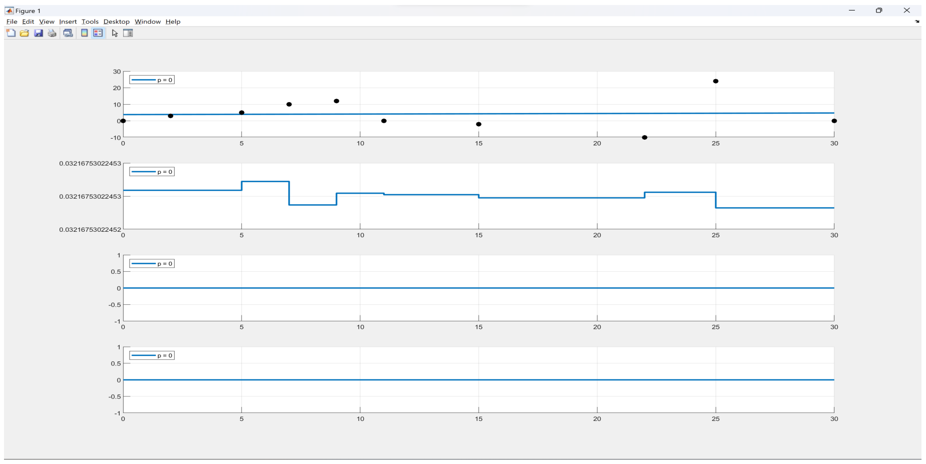

Figure 12: Smooth spline with $\mu = 0$

As can be seen, in this case the μ coefficient of the previous formula is equal to zero, which means that all the importance is given to the smoothness term and in fact the spline does not interpolate the waypoints well, also the acceleration is equal to zero and this produces a very smooth motion with fast changes.

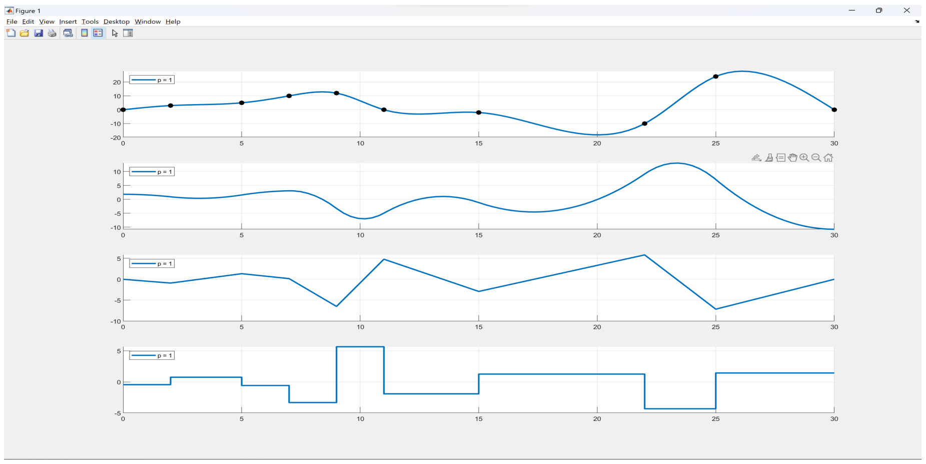

Figure 13: Smooth spline with $\mu = 1$

Unlike before, now the μ coefficient is equal to 1 and all importance is given to the fitting terms. In fact, the spline interpolates all waypoints perfectly, but the acceleration is very rough.

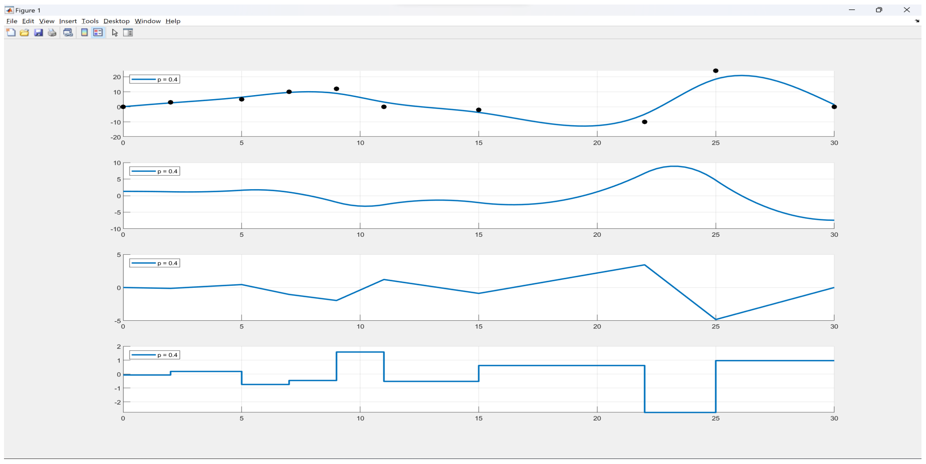

Figure 14: Smooth spline with $\mu = 0.4$

Finding the correct value for μ is an art, but with $\mu = 0.4$ we seem to have found a good balance between the two terms.

5 Multidimensional Trajectories and Geometric Path Planning

The definition of a trajectory in the Cartesian space implies the determination of a geometric path to be tracked with a prescribed motion law. For this purpose, it is convenient to consider a parametric representation of a curve in the space:

$$\mathbf{p} = \mathbf{p}(u) \tag{14}$$

where **p** is a vectorial function, which describes the curve when the independent variable u ranges over some interval of the domain space.

In particular, we have 2 main component to take into consideration,

- The geometry: which indicates the position where the end-effector will pass through
- The motion law: which indicates how we will follow the geometry path

Another important aspect is the synchronization between the different components of a robot, which is performed by imposing interpolation conditions at the same instants of time, as we did with multi-point trajectories. Obviously, imposing this condition requires a correct coordinate system. For this reason it is possible to define a coordinate frame directly related to the curve, the Frenet frame, represented by three unit vectors:

- The *tangent unit vector* e_t, which lies on the line tangent to the curve and is oriented according to the positive direction induced on the curve by s.
- The *normal unit vector* e_n, lying on the line passing through the point p, and orthogonal to e_t; the orientation of e_n is such that in a neighborhood of p the curve is completely on the side of e_n with respect to the plane passing through e_t and normal to e_n.
- The *binormal unit vector* e_b, defined in a such way that the three vectors (e_t, e_n, e_b) form a right handed frame.

5.1 Motion Primitives

A simple way to define the geometric path in the three-dimensional space is to use a sequence of basic motion primitives, such as straight lines, circles, and so on. Obviously, the simplest interpolation is the **straight line**.

$$p(u) = p_0 + (p_1 - p_0)u, \quad \text{with} \quad 0 \leq u \leq 1 \tag{15}$$

Another typical motion primitive is the circular arc, starting from a given point p0 and with the center located on a desired axis, determined by a unitary vector z1 and a generic point d.

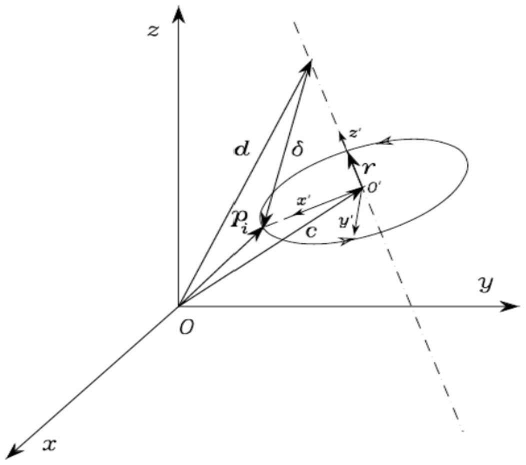

Figure 15: circular path

Finally, with these 2 motion primitives we made the trajectory in the figure below:

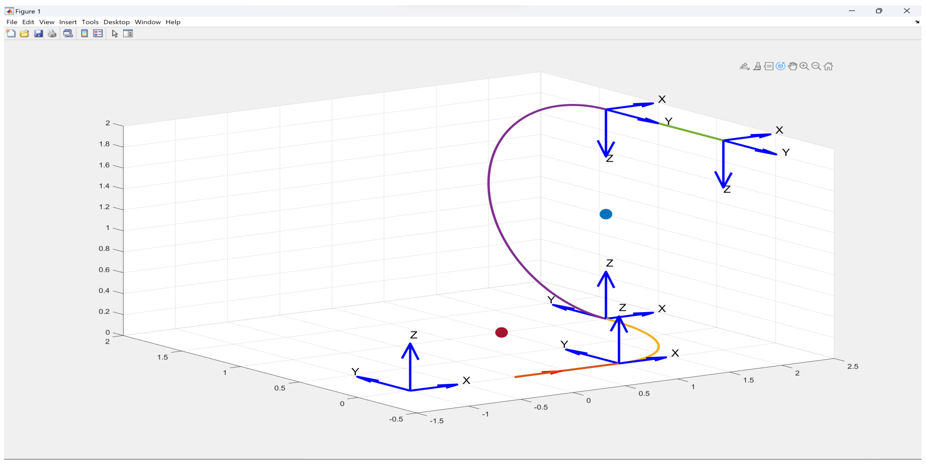

Figure 16: Trajectory

Here we plot only the trajectory with the Frenet frame to get a less confusing graph. Instead, below, we plot the trajectory with velocity and acceleration. As expected, the velocity (in red) is tangent to the trajectory and the acceleration (in light blue) points to the center of the circular trajectory; in fact, it is a centripetal acceleration.

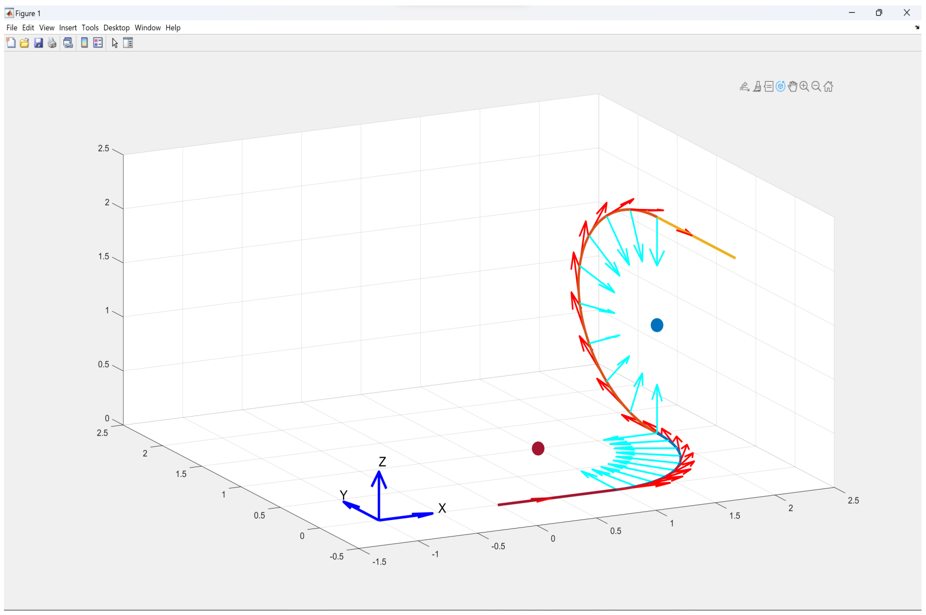

Figure 17: Trajectory with velocity and acceleration

6 Geometry in 3D space

In this chapter we will see how to plot a sphere passing through 3 distinct points. We manually implement the equation of the sphere and then using the great circle arc (geodetic) we connect the 3 points with the shortest path.

The ”great circle arc” function calculates the points along the geodesic, or arc of a maximum circle, connecting two points on a sphere.

The ”great circle arc” function returns a ”arc points‘ array of dimension num_points $\times$ 3, where each row represents the Cartesian coordinates of a point along the arc of maximum circle between ”pA and ”pB”. This set of points can be used to visualize geodesics on a graphical representation of the sphere.

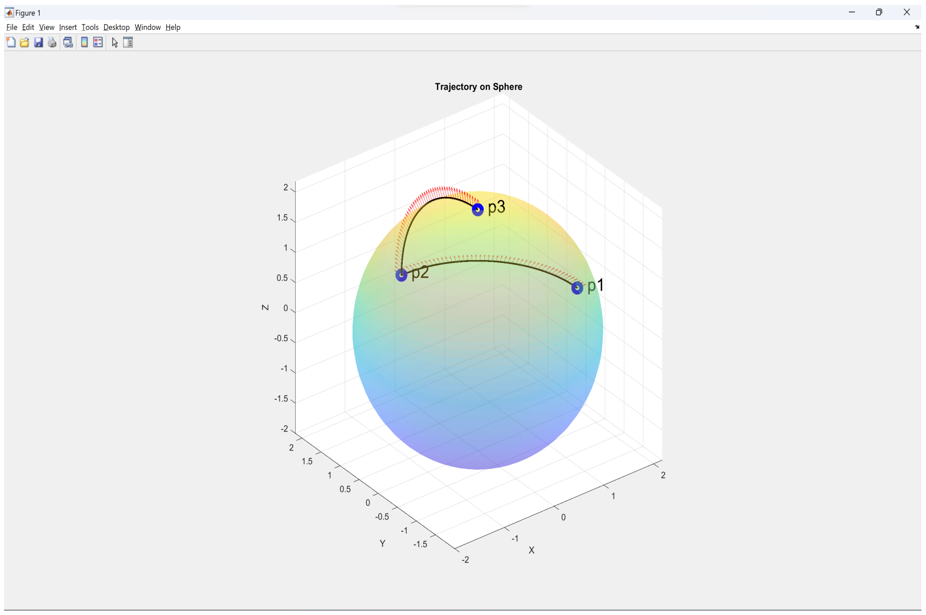

Figure 18: Sphere

In particular, in red is plotted the z axis that is always perpendicular to the path along the sphere.