POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master's Degree in Computer Engeneering

Master's Thesis

Smooth trajectory planning for anthropomorphic industrial robots employed in continuous processes

Supervisor:
Prof.ssa Marina Indri

Candidate:
Leonardo Anderlucci

Tutor:
Ing. Eliana Giovannitti
COMAU S.p.A. - RoboLAB

April 2019

At bottom, robotics is about us. It is the discipline of emulating our lives, of wondering how we work.

— ROD GRUPEN, *Discover Magazine*

Abstract

The first robotic arm was employed in an automotive assembly line in 1961. Since then, the industrial robots became progressively more widespread and now they can be commonly found even in small manufacturing enterprises.

Trajectory planning is an essential aspect of the motion programming for a robotic arm. In collaboration with COMAU, it was decided to focus on the study of a particular type of process, namely *continuous process*.

Continuous processes are tasks that require a continuous motion of the tool in terms of positions and velocity profile and are very common in industrial manufacturing. Some examples are arc welding, spray painting and adhesive sealing. A smooth execution and a precise control of the Cartesian speed is essential to guarantee a good quality of the final product.

The objective of the thesis is to propose a novel trajectory planner for this kind of processes. After a brief overview of the techniques currently employed in the trajectory planning for some common industrial processes, part of the efforts are dedicated to the geometrical analysis of the trajectory itself. This is important to find the critical parts of a path that may be difficult for the machine to follow with high precision. In these parts a lower speed can be imposed in order to obtain smoother results. This kind of work is generally done manually by trial and error, but it requires a lot of work and it can not achieve high precisions. Instead a method is proposed to simulate and automatize this process.

After that, the actual trajectory planner and its implementation is described. It is based on the spline interpolation and different techniques are analysed in order to study their performance in terms of quality of path obtained and velocity profile. The planner is proposed in two versions, one that considers all the points of the trajectory at once, and another that works only on a small subset at a time. The second version is proposed having in mind a possible on-line implementation, which limits the number of points that can be used to plan the trajectory.

The proposed methods are implemented in MatLab and have been tested on real processes proposed by COMAU. The trajectories computed with the novel planner have been properly simulated and then fed to a real robot bypassing the on-board planner. The results obtained have been analysed and verified in order to validate the approach.

The work done until now does not fully resolve the complex problem of trajectory planning for continuous processes, but the results are encouraging and they can be used as a basis for a future development and implementation.

Contents

Chapter 1

Introduction

This thesis has been developed in collaboration with COMAU S.p.A., an Italian company based in Grugliasco (Turin) that works mainly in the industrial automation field. COMAU has a well established history of cooperation with Politecnico di Torino. From this cooperation the RoboLAB was born in 2013, with the purpose of housing many research and formative projects shared between the company and the university.

This thesis was carried out thanks to the support of the COMAU engineers and to the possibility of performing tests in RoboLAB.

1.1 Problem overview

Anthropomorphic robot arms are widely used in manufacturing industry. They are employed for many different tasks such as: spot welding, continuous wire welding, spray painting, materials handling, palletizing and adhesive sealing.

Each task has different requirements in terms of trajectory planning. In this thesis we will focus on the specific sub-set of continuous processes. We call continuous processes the tasks that require a continuous motion of the robot end effector, which needs the planning of a smooth Cartesian path and a constant velocity profile.

Among the examples mentioned above, continuous processes are wire welding, spray painting and adhesive sealing. Another interesting continuous process is 3D printing with robotic arms. It is a recent application that is having a growing attention due to its potential in many different fields, spanning from the printing of design objects to the printing of bridges or houses. Regardless of the area, the quality of the results depends heavily on the planning of a trajectory that allows the tools to have a constant velocity, as needed for a uniform material deposition.

Currently the COMAU on-board planner uses just 2 points at each step to plan the programmed trajectory. It means that the motion is composed of a series of simple movements with a trapezoidal speed profile. The only exception it the fly-by

command: it allows to join two consecutive motions in a unique continuous one with a parabolic mash. In a fly-by motion these steps are followed: the first two points are selected and a first trajectory is planned and executed; during the execution the next point is fed to the planner and the next trajectory is planned; before the first motion end, the second is started after the execution of a parabolic mash. This kind of motion is used mainly to reposition the robot between different phases of a production cycle, but it is not very suitable for a continuous process. The main reasons are two: there is no guarantee of constant velocity, and the trajectory does not pass through the second point, as by definition it is just a via-point.

Given this situation and the increasing demand from the industries of manufacturing processes with continuous properties, the COMAU researchers have decided to explore different ways to improve their planner. This thesis takes place in this context and is focused on the development of a novel planner to be integrated to the existing one. The next section overviews the topics covered by this thesis.

1.2 Objective of the thesis

The objective of this thesis is firstly to analyse and describe some of the existing planning strategies in order to outline the current state of the art, focusing on the continuous processes.

Then a part of the thesis is devoted to the analysis of the geometrical characteristics of the path defined by the user. Here we propose the possible results of different interpolation techniques with a preliminary "feasibility" analysis, where the radius of the curves and the accelerations involved are computed. This is useful to find out critical points of the trajectory where we can expect to have speed drops, or to tune the target constant velocity to balance the total execution time and the maximum expected accelerations.

In the last part of the thesis, we focus on the building of a new primitive for the on-board COMAU planner. We want to discover if by enlarging the planning window by few points we could obtain a satisfying interpolation with respect to the results of a global planning. We limit the planning at each step to five or six points, because the on-board planner has to make all the computations in real time and realistically it is not possible to do that for all the points of a user-specified path.

The constraints imposed by the user are expressed in the Cartesian space and are limited to the positions (specified with a series of points), the target velocity and the maximum acceleration. For now, other constraints such as jerk limitation, actuators' limit or singularity points are not considered.

1.3 Thesis outline

The thesis is developed in seven chapters:

- Chapter 1: In the first chapter a brief introduction of the thesis and its research topics are presented.
- Chapter 2: In the second chapter the theory behind trajectory planning is reviewed with the description of some methods commonly found in literature.
- Chapter 3: In the third chapter three examples of relevant industrial continuous processes are analysed. For each industrial process an example of existing planning technique is described.
- Chapter 4: The fourth chapter is about a mathematical description of three spline interpolation techniques that are extensively used in Chapter 5.
- Chapter 5: In this chapter a novel method for trajectory planning is described. The chapter has two main parts. Firstly a global planning technique is presented, then its local implementation is described. Each approach is provided with MatLab simulations on different datasets.
- Chapter 6: The experimental set-up and the results of the testing done on a real COMAU robot are shown in this chapter.
- Chapter 7: The last chapter summarizes the achievements of this work and presents possible future developments.

Chapter 2

Trajectory planning

2.1 Introduction

Industrial robots have very specific tasks inside the production line. It is generally useful to describe in a clear and concise way the series of operations needed to complete their tasks.

The planning can be defined by a series of actions organized in a hierarchical way as follow [2]:

1. Objective: it is the highest level action. It usually refers to the main goal of the production line and not to a single robot. For example an objective is the assembly of an engine head.

2. Task: it defines a subset of actions needed to accomplish an objective. Considering the example of the engine, assembly of pistons and assembly of valves are two different tasks.

3. Operation: a task is decomposed in a sequence of operations. For example the assembly of a piston can be decomposed in the following operations: grasp a piston from the stock, move it near the cylinder, insert it.

4. Move: it defines a single elementary motion. For example, we can decompose the operation "grasp a piston" in: move the gripper toward the stock, open the gripper, move close to the first available piston, close the gripper and grab the piece, move the gripper in the right pose.

5. Path/Trajectory: each move is decomposed in one or more paths (only geometrical description without time law) or trajectories (considering also the time law or kinematic constraints). We can decompose the move "move the gripper toward the stock" in a linear path followed by an arc section and a pose change of the gripper.

6. Reference: it is the lowest level action. It usually consists in a sampling of the trajectory. It is the data signal supplied to the controller of the actuators.

The objective of trajectory planning is to generate the reference input to the motion control system. The desired trajectory is usually specified by the user with a limited number of parameters that describe the geometry of the path and possibly the time law to obtain. From this description the planner generates an interpolating function used to obtain the reference input. There are many techniques for trajectory planning depending also on how the points are assigned [9]. Here we will present some of them. In Figure 2.1 we see a block representation of the trajectory planning activity.

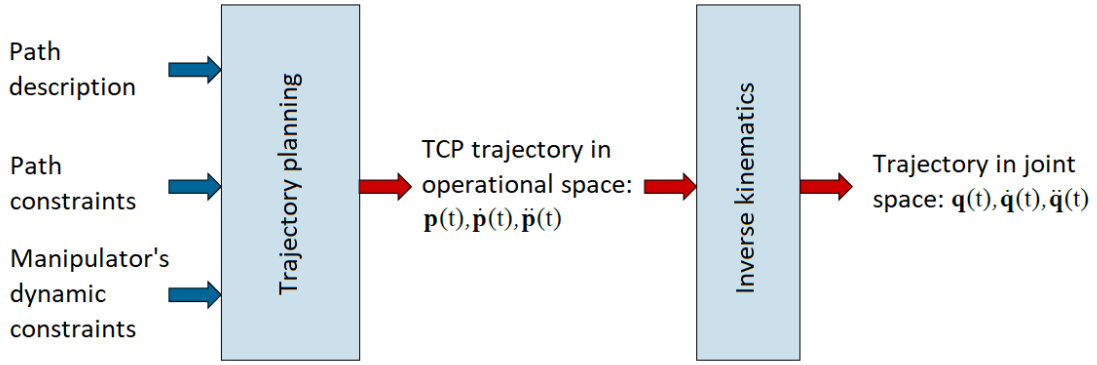

Figure 2.1: Block representation of trajectory planning [7]

2.2 Path and trajectory

Path and trajectory are often used as synonyms, but in this context they have a different meaning. With path we refer to a pure geometrical description of motion. More precisely, it is the locus of points (in the joint or operational space) that the manipulator has to follow in order to execute its task. On the other hand, a trajectory is a path with a description of the temporal law, usually specified in terms of velocity and acceleration. Figure 2.2 highlights the difference between the two concepts.

2.3 Planning constraints

Neither a geometric path nor a time law can be specified by the user in their full extent, i.e. for each point. Based on the joint or task space, the following constraints are usually specified [9]:

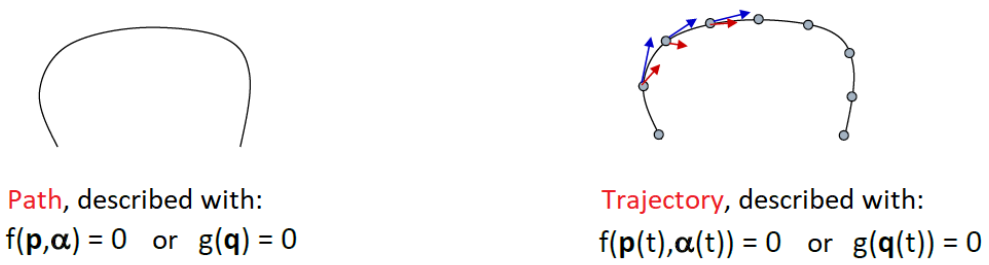

Figure 2.2: Example of differences between path and trajectory [7]

- The path is described with few parameters, like the starting and ending points, eventually some intermediate ones and an interpolating function (linear, circular, ...).
- The temporal low is imposed with constraints on maximum velocities and accelerations. The user can also assign a specific velocity or acceleration at points of particular interest or describe a global property, like a constant speed trajectory.
- Some constraints are machine-dependent and they refer to the maximum angular velocity and acceleration that can be supplied by the actuators.

2.4 Joint space and task space planning

Planning a trajectory in the joint space is generally faster and less computationally demanding, because the generated function $\mathbf{q}(t)$ can be fed directly to the controller. It is also useful to avoid kinematic singularities and to exploit redundancies, if the manipulator has one or more. The constraints on the actuators are also easily checked computing the functions $\dot{\mathbf{q}}(t)$ and $\ddot{\mathbf{q}}(t)$. The positions to generate the path can be derived by recording manually the target positions of the manipulator or by computing the inverse kinematics in simulation [9].

Joint space planning is not recommended when the TCP needs to follow a specific path or there are obstacles to avoid in the work space. As we see in Figure 2.3 and Figure 2.4, the conversion from one space to the other one is not linear due to the kinematics. This makes difficult to understand how a path planned in joint space translates in the task space. In these situations we plan the trajectory in the space where the constraints are defined. Once we have the Cartesian trajectory $\mathbf{p}(t)$ planned, we still need to translate each sampling in the joint space using the inverse kinematics of the manipulator. This operation is done on-line and it sets a limit to the maximum sampling rate obtainable [9].

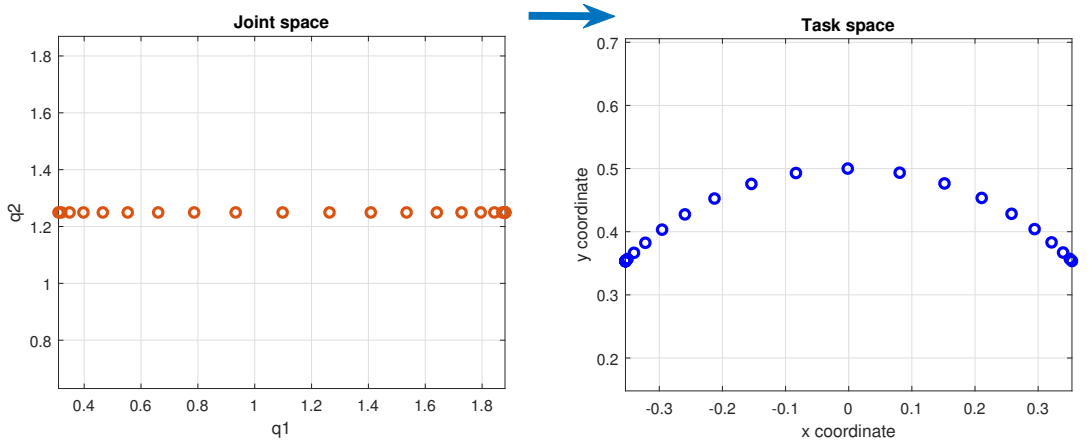

Figure 2.3: Conversion of a variable from joint space to task space

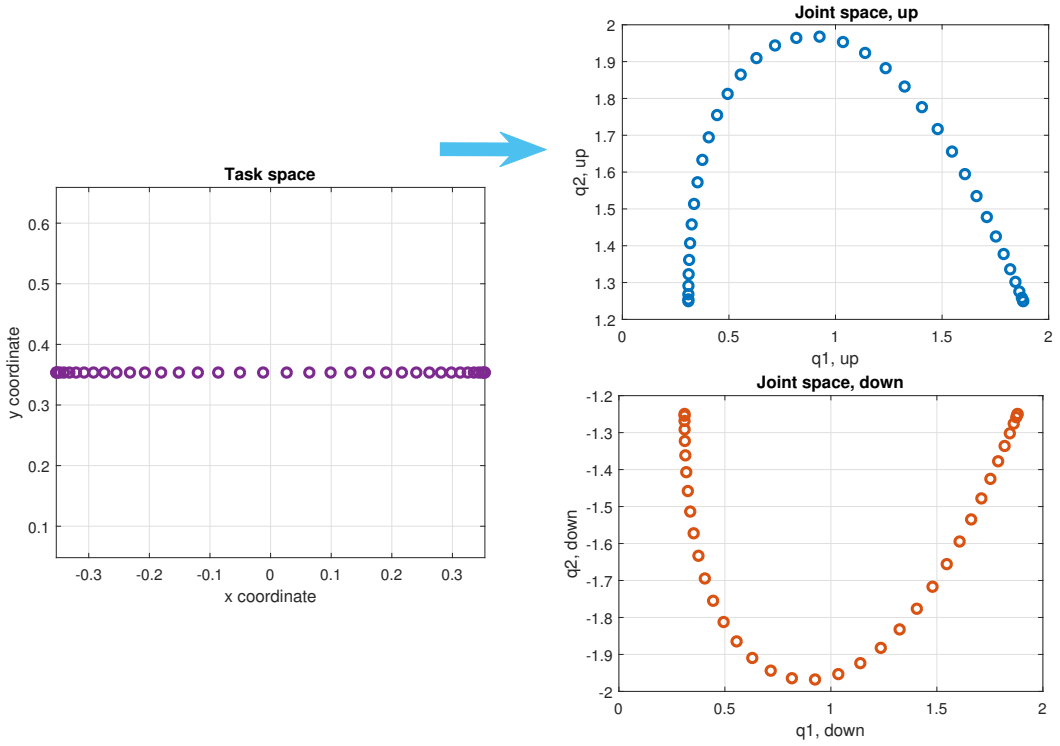

Figure 2.4: Conversion of a variable from task space to joint space

2.5 Point-to-point planning in joint space

Point-to-point trajectories (PTP) are the simplest ones and they correspond to a movement from one point to another. Firstly just a single joint variable $q(\mathrm{t})$ is considered, then the results are generalized to the case of multiple joints (planning of $\mathbf{q}(\mathrm{t})$).

The starting configuration is $q(t_0) = q_0$ and the target configuration is $q(t_f) = q_f$. The movement has to respect the constraints on velocity, acceleration and possibly jerk.

2.5.1 Third-order polynomial trajectory

A *third-order (cubic) polynomial function* is generally a good solution that minimizes the energy dissipation in the actuator [9].

The *cubic joint motion* to be determined is:

$$q(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{2.1}$$

The velocity has a parabolic profile:

$$\dot{q}(t) = 3a_3 t^2 + 2a_2 t + a_1 \tag{2.2}$$

The acceleration has a linear profile:

$$\ddot{q}(t) = 6a_3 t + 2a_2 \tag{2.3}$$

With four coefficients available, we can impose four conditions, that are: initial and final positions and initial and final velocities (generally set to zero). The trajectory is therefore given by the solution of the following system of equations:

$$\begin{cases} a_0 = q_i \\ a_1 = \dot{q}_i \\ a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 = q_f \\ 3a_3 t_f^2 + 2a_2 t_f + a_1 = \dot{q}_f \end{cases} \tag{2.4}$$

In Figure 2.5 there is an example of the application of this planning method.

Higher order polynomials can be considered to impose additional constraints on the trajectory. For example a *fifth order polynomial* in the form:

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{2.5}$$

can be used to impose the previous conditions and initial and final acceleration. In this case we give up the property of minimum energy dissipation.

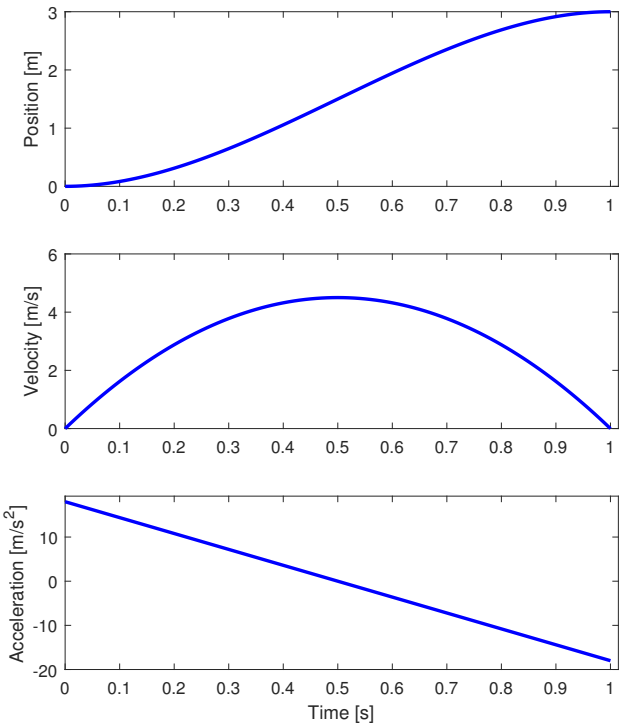

Figure 2.5: Time history of position, velocity and acceleration with a cubic polynomial timing law

2.5.2 2-1-2 trajectory

A frequently adopted approach in industrial practice is the so called *trapezoidal velocity profile*, or *2-1-2 position profile*. This kind of trajectory is very common because it allows a direct verification of the resulting velocities and accelerations. It imposes a constant acceleration starting phase, a cruise velocity phase and a constant deceleration ending phase, as in Figure 2.6. If the accelerations and the cruise velocity are the maximum possible, then this trajectory minimizes the travel time. The continuity can be guaranteed for position and velocity, not for acceleration.

This trajectory is described by the following equations[7]:

- Acceleration:

$$\ddot{q}(t) = \begin{cases} \ddot{q}^{+}_{max} & t \in I_1 \\ 0 & t \in I_2 \\ -\ddot{q}^{-}_{max} & t \in I_3 \end{cases} \tag{2.6}$$

- Velocity:

$$\dot{q}(t) = \begin{cases} \ddot{q}^{+}_{max}(t - t_0) + \dot{q}_0 & t \in I_1 \\ \dot{q}_{max} & t \in I_2 \\ \dot{q}_{max} - \ddot{q}^{-}_{max}(t - t_2) & t \in I_3 \end{cases} \tag{2.7}$$

- Position:

$$q(t) = \begin{cases} \frac{1}{2}\ddot{q}^{+}_{max}(t - t_0)^2 + \dot{q}_0(t - t_0) + q_0 & t \in I_1 \\ \dot{q}_{max}(t - t_1) + q_1 & t \in I_2 \\ -\frac{1}{2}\ddot{q}^{-}_{max}(t - t_2)^2 + \dot{q}_{max}(t - t_2) + q_2 & t \in I_3 \end{cases} \tag{2.8}$$

$I_1 = [t_0, t_1)$, $I_2 = [t_1, t_2)$ and $I_3 = [t_2, t_f]$ specify the duration of the three phases, while t_1 and t_2 are the switching instants.

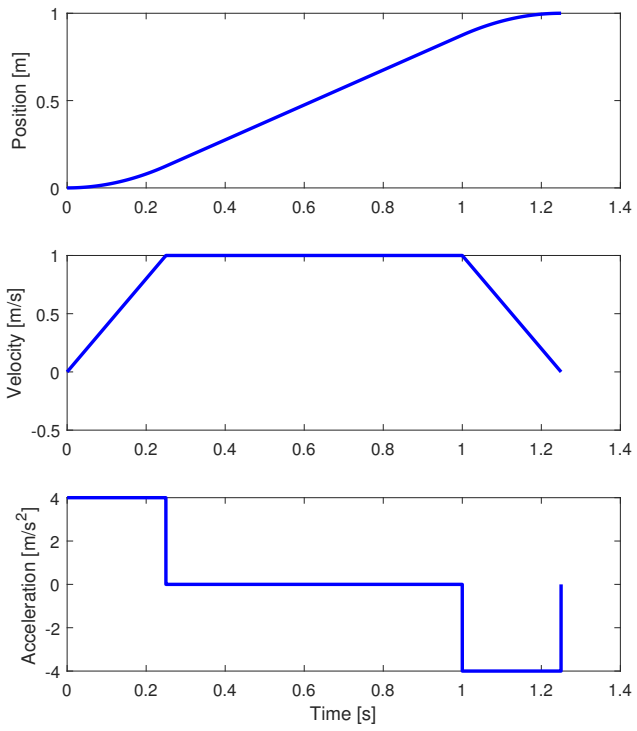

Figure 2.6: Characterization of a timing law with trapezoidal velocity profile in terms of position, velocity and acceleration

We also need to add other constraints on:

- Initial conditions:

$$\begin{aligned} q(t_0) &= 0 \\ \dot{q}(t_0) &= 0 \end{aligned} \tag{2.9}$$

- Continuity on commutation time instants:

$$\begin{aligned} q(t_1) &= \frac{1}{2}\ddot{q}^{+}_{max}(t_1 - t_0)^2 \\ q(t_2) &= \dot{s}_{max}(t_2 - t_1) + q_1 \\ \dot{q}(t_1) &= \dot{q}(t_2) = \dot{q}_{max} = \ddot{q}^{+}_{max}(t_1 - t_0) \end{aligned} \tag{2.10}$$

- Final conditions:

$$\begin{aligned} q(t_f) &= \frac{1}{2}\ddot{q}^{+}_{max}(t_1 - t_0)^2 + \dot{q}_{max}(t_f - t_1) - \frac{1}{2}\ddot{q}^{+}_{max}(t_f - t_1)^2 = 1 \\ \dot{q}(t_f) &= \dot{q}_{max} - \ddot{q}^{-}_{max}(t_f - t_2) = 0 \end{aligned} \tag{2.11}$$

The constant speed interval exists only if:

$$t_2 - t_1 > 0 \rightarrow \frac{1}{\dot{q}_{max}} > \frac{1}{2}\left(\frac{\dot{q}_{max}}{\ddot{q}^{-}_{max}}\frac{\dot{q}_{max}}{\ddot{q}^{+}_{max}}\right) \tag{2.12}$$

If this condition is not satisfied, then the trajectory is composed only by a maximum acceleration followed by a maximum deceleration interval. This trajectory has a *"bang-bang" acceleration profile* and a *triangular velocity profile*, as we can see in Figure 2.7.

2.5.3 Multiple-joints generalization

Generalizing the trajectory planning in presence of multiple joints, we need also to take into consideration the synchronization of all the joints. We have a synchronized motion when the the beginning and ending time instants coincide for every joint.

To achieve this, we can express the trajectory as a *convex combination* of $\mathbf{q}_0$ and $\mathbf{q}_f$ with a curvilinear abscissa $\mathbf{s}$(t) of the path:

$$\mathbf{q}(t) = (1 - s(t))\mathbf{q}_0 + s(t)\mathbf{q}_f = \mathbf{q}_0 + s(t)(\mathbf{q}_f - \mathbf{q}_0) = \mathbf{q}_0 + s(t)\Delta\mathbf{q} \tag{2.13}$$

$s(t)$ can be planned for example with one of the method exposed in the previous subsections and then scaled to respect the following condition:

$$0 = s(t_0) \leq s(t) \geq s(t_f) = 1 \tag{2.14}$$

In this way we obtain $\mathbf{q}(t_0) = \mathbf{q}_0$ and $\mathbf{q}(t_f) = \mathbf{q}_f$.

Since the motion is coordinated, we need to take into account the velocity and acceleration constraints of each joint. The maximum velocities are scaled for the relative distances covered and the lowest one is chosen. In this way each constraints is guaranteed to be respected:

$$\dot{s}_{i,max} = \frac{\dot{q}_{i,max}}{\Delta q_i} \rightarrow \dot{s}_{max} = \min_{i=1,\dots,n}\{\dot{s}_{i,max}\} \tag{2.15}$$

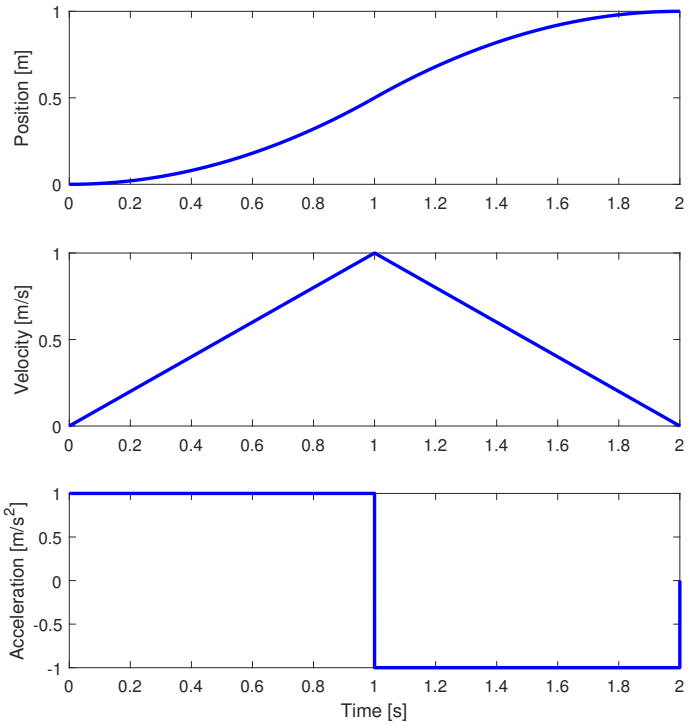

Figure 2.7: Time history of position, velocity and acceleration with a triangular velocity profile timing law

The same holds for the limits of maximum acceleration and deceleration:

$$\begin{aligned} \ddot{s}^{+}_{i,max} &= \frac{\ddot{q}^{+}_{i,max}}{\Delta q_i} \rightarrow \ddot{s}^{+}_{max} = \min_{i=1,\dots,n} \left\{ \ddot{s}^{+}_{i,max} \right\} \\ \ddot{s}^{-}_{i,max} &= \frac{\ddot{q}^{-}_{i,max}}{\Delta q_i} \rightarrow \ddot{s}^{-}_{max} = \min_{i=1,\dots,n} \left\{ \ddot{s}^{-}_{i,max} \right\} \end{aligned} \tag{2.16}$$

2.6 Motion through a sequence of points

In some application a path can be described with a number of points grater than two. This can be useful in general to guarantee a better monitoring on the executed trajectories: more points are specified in the presence of obstacles or when a high curvature path is expected. Hence the problem is to generate a trajectory interpolating N given points, called *path points*. If the exact reach is not required, than the points specified are called *via points* and other techniques can be adopted.

2.6.1 Polynomial interpolation

Given N constraints, we can use an $(N\text{-}1)$-order polynomial:

$$q(t) = \sum_{k=0}^{N-1} a_k t^k \tag{2.17}$$

This choice, however, has several drawbacks [9]:

- It is not possible to assign the initial and final velocities
- As the order of the polynomial increases, also its oscillatory behaviour increases (Runge's phenomenon, see Figure 2.8)
- The computational cost can be very heavy for high order polynomial
- The values of the coefficients depend on all the assigned points. If just one point is changed, all of them must be recomputed.

In practice this technique is not used in favour of piecewise functions of lower order.

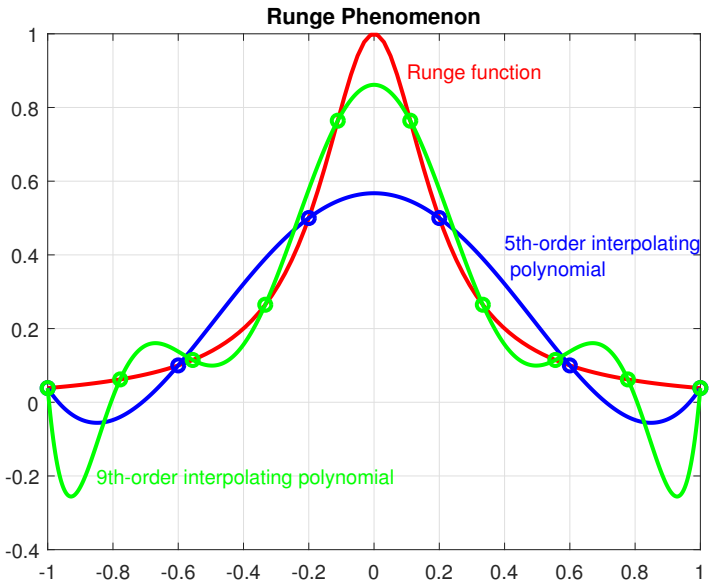

Figure 2.8: Example of polynomial interpolation and Runge's phenomenon

2.6.2 Spline interpolation

Instead of using one high order polynomial for the interpolation of N points, it is possible to consider $N\text{-}1$ low-degree interpolating polynomials, one for each interval.

Here the path points are called *knots* and, with the proper conditions, we can impose continuity up to the desired derivative. The resulting piecewise function is called *spline*.

Depending on the conditions and on the order of the basic functions, we have different kinds of spline interpolation. The interpolating polynomials are usually chosen of third order, since it allows to impose the continuity up to acceleration. *Quintic polynomials* can be chosen to further increase the smoothness at the expense of increased computational cost and higher oscillations. To simplify the problem it is also possible to use polynomials of the first or second order, but in this case we talk about *linear* or *quadratic interpolation*.

In the present subsection a brief introduction is proposed, while a deeper analysis of the methods of interest is carried out in Chapter 4.

Interpolating polynomials with imposed velocities at path points

The objective is to find the coefficient of *N-1* cubic polynomials $\Pi_k(t)$ (see equation (2.1)) interpolating the points q_k and q_{k+1} , for k = 1,..., N-1. If the user is able to specify the desired velocity at each knot, we have the following N-1 systems of equations to solve:

$$\begin{aligned} \Pi_k(t_k) &= q_k \\ \Pi_k(t_{k+1}) &= q_{k+1} \\ \dot{\Pi}_k(t_k) &= \dot{q}_k \\ \dot{\Pi}_k(t_{k+1}) &= \dot{q}_{k+1} \end{aligned} \tag{2.18}$$

The initial and final velocities are usually set to zero. The continuity of velocity at the knots is imposed by:

$$\dot{\Pi}_k(t_{k+1}) = \dot{\Pi}_{k+1}(t_{k+1}) \tag{2.19}$$

The continuity of acceleration is not guaranteed.

Interpolating polynomials with computed velocities at path points

This case is similar to the previous one, but the knot velocities are not imposed. Instead the following rule is used:

$$\begin{aligned} &\dot{q}_1 = 0 \\ &\dot{q}_k = \begin{cases} 0 & \operatorname{sgn}(v_k) \neq \operatorname{sgn}(v_{k+1}) \\ \frac{1}{2}(v_k + v_{k+1}) & \operatorname{sgn}(v_k) \neq \operatorname{sgn}(v_{k+1}) \end{cases} \\ &\dot{q}_N = 0 \end{aligned} \tag{2.20}$$

$v_k = (q_k - q_{k-1})/(t_k - t_{k-1})$ is the slope of the segment between q_{k-1} q_k in the time interval $[t_{k-1}, t_k]$.

Once the velocities are computed, the system of equations to be solved is the same of the previous case (2.18). In Figure 2.9 there is a comparison between this method and the previous one.

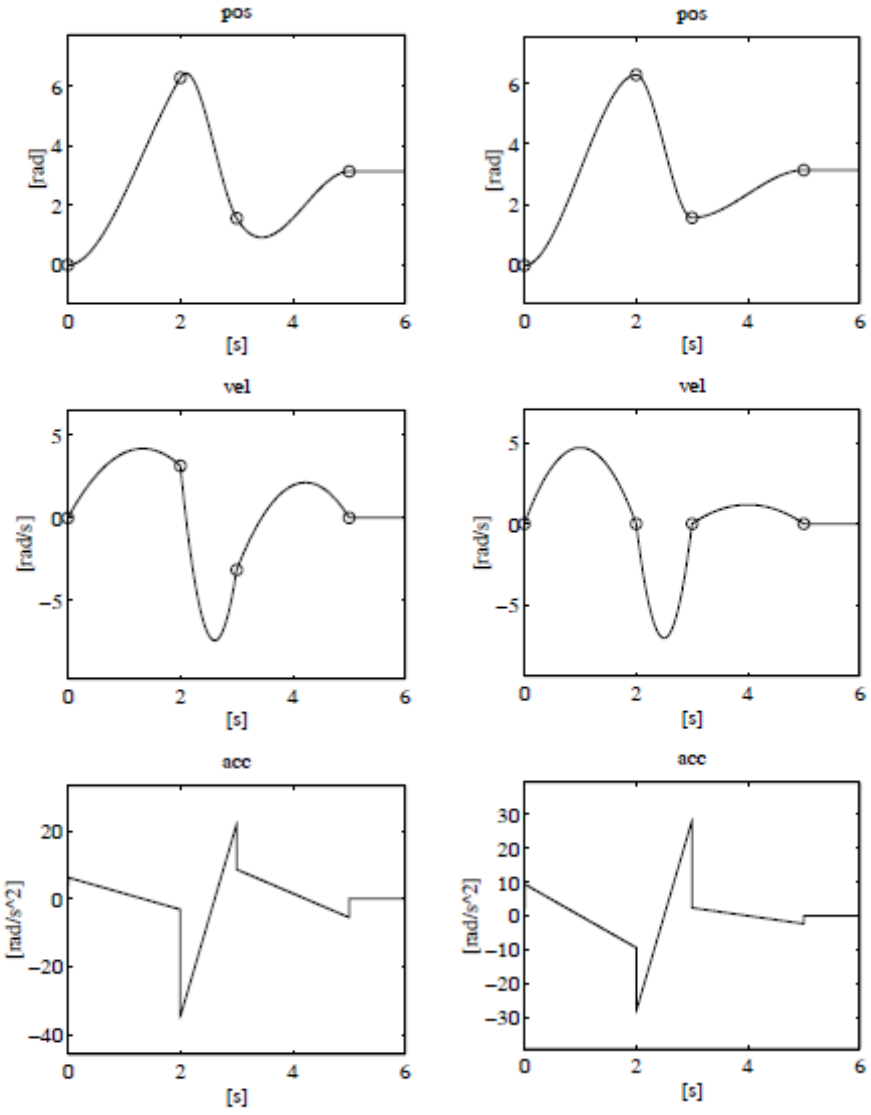

Figure 2.9: Time history of position, velocity and acceleration with with a timing law of interpolating polynomials with imposed velocities on the left and computed velocities on the right [9]

Interpolating polynomials with continuous accelerations at path points

As already stated, the previous methods do not guarantee the continuity of acceleration. If we want that, we need to impose the following conditions:

- Initial and final position:

$$\Pi_k(t_k) = q_k,\ \Pi_k(t_{k+1}) = q_{k+1} \qquad k = 1, ..., N-1 \tag{2.21}$$

- Continuity of velocity on internal knots:

$$\dot{\Pi}_k(t_{k+1}) = \dot{\Pi}_{k+1}(t_k) \qquad k = 1, ..., N-2 \tag{2.22}$$

- Continuity of acceleration on internal knots:

$$\ddot{\Pi}_k(t_{k+1}) = \ddot{\Pi}_{k+1}(t_k) \qquad k = 1, ..., N-2 \tag{2.23}$$

There are a total of $4(N-1)$ unknown coefficients and we have $2(N-1)+2(N-2) = 4(N-1)-2$ equations. To determine uniquely the spline we need two other constraints. Typical choices are [8]:

1. $\ddot{\Pi}_1(t_1) = 0,\ \ddot{\Pi}_{N-1}(t_N) = 0$ (natural cubic spline)
2. $\dddot{\Pi}_1(t_2) = \dddot{\Pi}_2(t_2),\ \dddot{\Pi}_{N-2}(t_{N-1}) = \dddot{\Pi}_{N-1}(t_{N-1})$ ("not-a-knot" conditions)
3. $\dot{\Pi}_1(t_1) = \dot{q}_1,\ \dot{\Pi}_{N-1}(t_N) = \dot{q}_N$

For our purposes we will always use the choice 3 since it is very useful to freely impose the initial and final velocities.

In Chapter 4 we will see how to find the solution of this problem in an efficient way.

2.6.3 Interpolating linear polynomials with parabolic blends

If the exact reaching of the path points is not required, than this technique can be taken in consideration due to its simplicity. The N path points are interpolated with linear segments at time instants $t_1, ..., t_N$. Around t_k a *parabolic blend* is generated to avoid discontinuity. The final trajectory is composed of a sequence of linear and quadratic polynomials (discontinuity on acceleration is tolerated). Referencing to Figure 2.10, we define:

- $\Delta t_k = t_{k+1} - t_k$, time distance between q_k and q_{k+1}
- $\Delta t_{k,k+1}$, linear trajectory time interval

- $\Delta t'_k$ quadratic trajectory time interval

Given the above data, the velocity of linear traits are computed as [9]:

$$\dot{q}_{k-1,k} = \frac{q_k - q_{k-1}}{\Delta t_{k-1}} \tag{2.24}$$

and the accelerations as:

$$\ddot{q}_k = \frac{\dot{q}_{k,k+1} - \dot{q}_{k-1,k}}{\Delta t'_{k-1}} \tag{2.25}$$

The trajectory does not reach any of the intermediate points. For this reason here they are called *way points*. To decrease the error, $\Delta t'_k$ can be chosen smaller, but the relative acceleration will be higher.

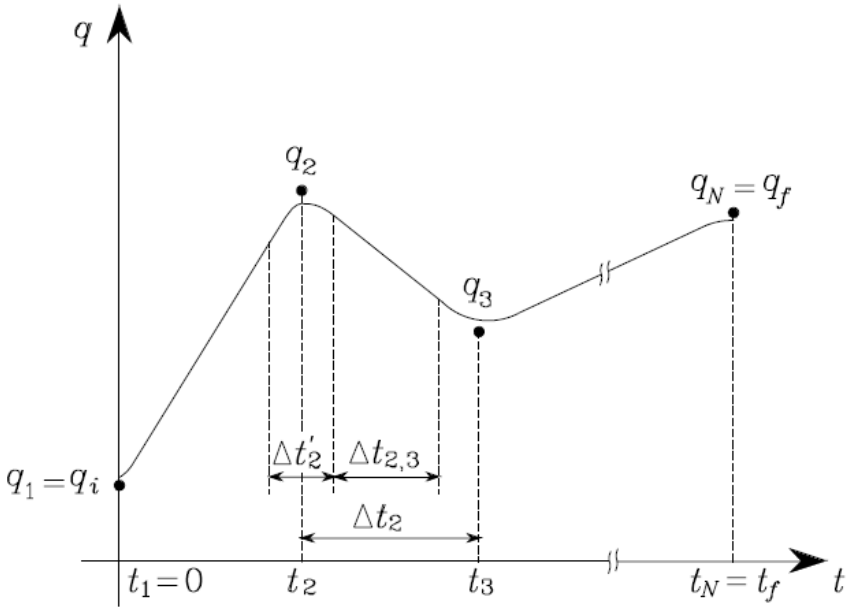

Figure 2.10: Characterization of a trajectory with interpolating linear polynomials with parabolic blends [9]

We now consider a particular case when there is only one intermediate point. The trajectory can be planned with two separate 2-1-2 motions. The intermediate points is reached, but the manipulator stops there. Instead, if we start the second segment ahead of time and we use the sum of velocities as reference, we obtain a continuous motion. The manipulator does not stop in the intermediate point, but its crossing is not guaranteed, similarly to the previous case. We can see an example in Figure 2.11

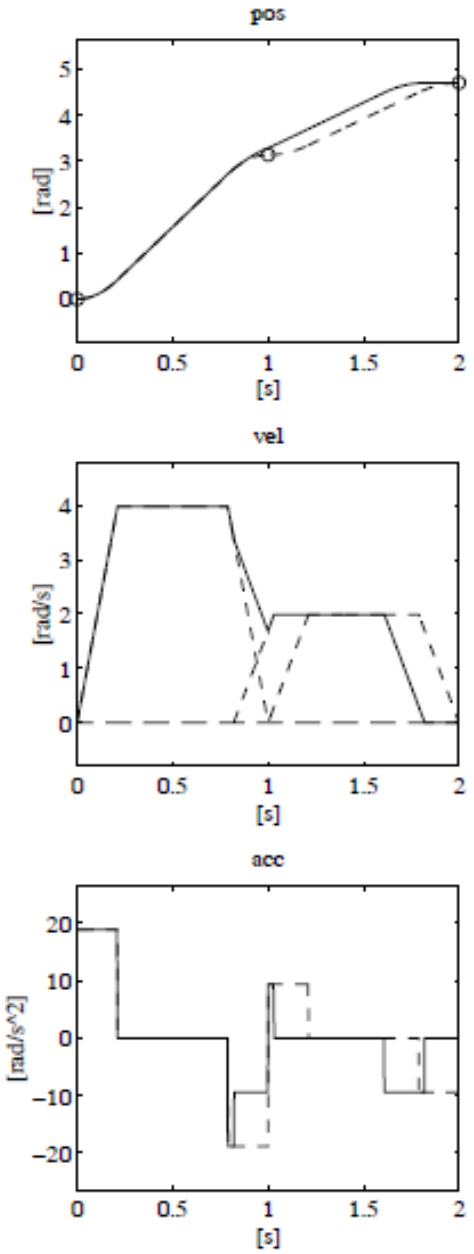

Figure 2.11: Time history of position, velocity and acceleration with a timing law of interpolating linear polynomials with parabolic blends obtained by anticipating the generation of the second segment of trajectory [9]

Chapter 3

Examples of continuous processes in industrial manufacturing

Globally, it is estimated that 3.1 million industrial robots will arrive in factories by 2020. The international market value for robotic systems is estimated around 42 billion Euro with the automotive and electrical/electronic industries being the most important customers. The automotive sector bought the 33% of the total industrial robots supply in 2017 and the electrical/electronic sector the 32% of the total in the same year [1].

Advantages of robotic manufacturing

Even though industrial robots are employed in many different processes, there are some common motivations and advantages that draw the attention to this technology. Here we summarize some of them [2].

1. **Quality / accuracy / precision.** Due to its mechanical nature and computerized control, an industrial robot can carry out a repetitive task with great precision and accuracy, thus providing improved, consistent product quality. This includes less material wasting and less post-working cleanup, which helps in lowering the overall production costs.

[1] https://ifr.org/downloads/press2018/Executive_Summary_WR_2018_Industrial_Robots.pdf

[2] https://www.brighthubengineering.com/robotics/76606-advantages-of-robotics-in-engineering/

2. **Efficiency / speed / production rate.** The same features of industrial robotics technology mentioned above, the mechanical nature of the equipment and the computerized control, make industrial robotics technology more efficient and fast, leading to higher production rates than with human labour. Another aspect of efficiency is that robots can be mounted from the ceiling and have no problem with working upside down. This can lead to a savings in floor space.

3. **Ability to work in environments that are inhospitable to humans.** This is an interesting set of advantages of robotics. There are a number of tasks that are too dangerous or too exposed to toxins for humans to conveniently do them. These are ideal robotic tasks. This includes tasks as simple as spray painting, where the fumes can be cancerous or explosive. It also includes tasks such as defusing bombs or cleaning sewers.

4. **Freedom from human limitations like boredom.** Human characteristics like boredom from doing a repetitive task do not interfere with the functioning of a robot. There is some overlap with the first two categories of advantages of robotics, because the lack of interference from boredom leads to greater accuracy, quality, and rate of production over time. There is more to this set of advantages of robotics, however. A robotic arm can work 24/7, with downtime limited only to the scheduled maintenance.

Later in this chapter we introduce three cases of continuous process with examples of existing planning algorithms.

3.1 Robotic arc welding

Robotic welding is one of the most popular industrial applications of robotics worldwide. It is estimated that approximately 25% of all industrial robots are employed for welding operations, with automotive manufacturing and electrical/electronic industry representing the most active sectors in terms of robotic welding adoption. Apart from resistance spot welding, the two most common robotised welding processes for production purposes are metal inert gas (MIG) welding and tungsten inert gas (TIG) welding, respectively [3].

Definition

In the industrial sector *welding* is a process that is used to join materials, usually metals or thermoplastics. Two parts are heated at high temperature and melt

[3] https://www.twi-global.com/technical-knowledge/job-knowledge/robotic-arc-welding-135

together, then cooled down to cause their fusion. Pressure may also be used in conjunction with heat, or by itself, to produce a weld. In addition to the base metal, a filler material is usually added to the joint to form a *weld pool*. This new joint, when cooled down, can be stronger than the base material. Different energy sources are used for welding. The most common are: gas flame, electric arc, laser, electron beam, friction or ultrasound.

Arc welding is a welding technique where the intense heat needed to melt metal is produced by an electric arc. The arc is formed between the metal part and an electrode that is moved along the joint. This electrode can be inert, simply carrying the current, or it may consumed to provide the filling metal to the joint. The metal base and the electrode are connected to an AC or DC power source and an electric arc is formed when the tip of the electrode is close to the work piece. The arc produces temperatures above 3000° necessary to induce the melting of the base and the electrode [4].

The typical components of an integrated robotic arc welding cell are [5]:

- Arc welding robot
- Power source
- Welding torch
- Wire feeder
- Welding fixtures and workpiece positioners
- Torch cleaner
- TCP calibration unit

In Figure 3.1 we can see a typical robotised welding station with a welding robot and a workpiece positioner.

3.1.1 Example of trajectory planning for robotic arc welding

As an example of trajectory planning for robotic arc welding, we refer to the work «Cartesian path planning for arc welding robots: Evaluation of the descartes algorithm» by De Maeyer, Moyaers, and Demeester [4].

[4]https://www.lincolnelectric.com/en-us/support/process-and-theory/pages/arc-welding-detail.aspx

[5]http://www.robot-welding.com/robot_arc_welding.htm

[6]http://www.smerobotics.org/demonstrations/d4.html

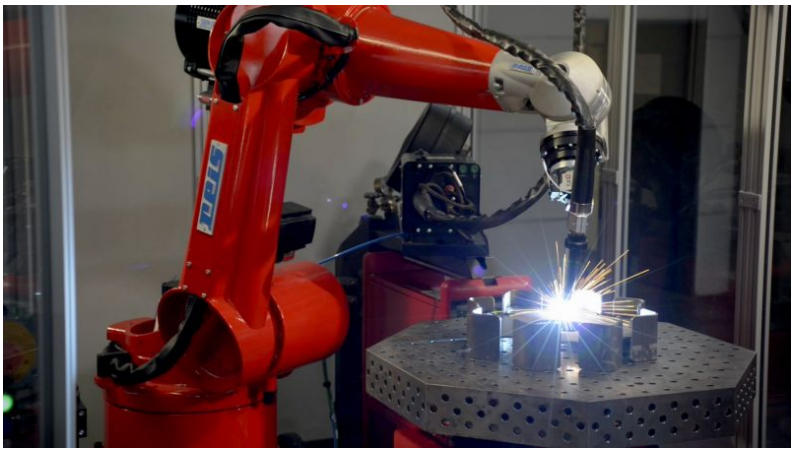

Figure 3.1: Typical arc welding station [6].

The purpose of that article is to study and test the performance of a software package for Cartesian path planning. The software name is Descartes and is released by the ROS-Industrial community specifically for applications such as robot welding.

Planning constraints

In a typical welding process the rotation of the torch (z-axis) is not specified. There also may be tolerance on the orientation of the other two angles (x and y axis). When considering a typical 6 DOF robot, this means that there are redundancies that can be exploited to find a feasible path. This path, in order to be acceptable, should: be collision-free, respect the maximum velocities and accelerations of the robot, avoid the singularity points.

Planning algorithm

The planning algorithm is divided in three phases.

Phase 1. A list of trajectory points is specified, which can be expressed as Cartesian or joint positions. For each point, the parameters of the pose can be assigned a tolerance range and than sampled inside that range. This leads to multiple joint or Cartesian points. Every Cartesian point is then converted in joint positions according to the inverse kinematics of the robot.

Phase 2. The computed joint positions are organised in a graph (Figure 3.2). Multiple nodes can belong to the same trajectory point. A joint position is discarded from the graph if the collision detection check is not passed. Then the edges between nodes of two successive trajectory points are added and the cost is assigned:

$$edge \;\; cost = ||\theta_{n+1} - \theta_n||_1 \tag{3.1}$$

where θ_n is a generic joint position. If the angle that a joint has to travel is greater than the velocity limit, than that edge is discarded.

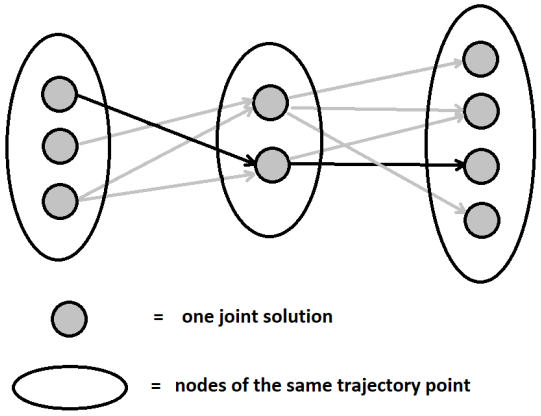

Figure 3.2: Schematic representation of the graph used in Descartes' algorithm.

Phase 3. A standard search algorithm is applied on the resulting directed graph to find the shortest path. In Descartes' case, Dijkstra's algorithm is used.

The final result is a sequence of joint positions that, when followed by the robot, result in the desired Cartesian path within the given tolerances, if such a path exists and is found by the inverse kinematic solver. The execution of these points is left to the industrial robot controller. In some cases the path may not be found or may be infeasible. The main reasons are that it may contain large accelerations or collisions in the given points or in between them, where the algorithms do not perform a check. In these cases re-planning or post processing is necessary.

3.2 Spray painting

Spray painting is a painting technique where a device sprays a coating (paint, ink, varnish, etc.) through the air onto a surface.

It is an important process in the manufacturing of many products, such as automobiles, furniture, airplanes and it was one of the first uses for industrial robots. The volatile and hazardous nature of solvent based paint means that it is better to minimise human contact, and painting robots have been developed to safely work with flammable compounds or in explosive atmospheres. Explosion proof painting robots are sealed units and the arms are pressurised with air to prevent the ingress of explosive solvents. Pressure sensors are used to monitor the integrity of the system.

A painting robot generally has quite thin arms, as access is very important and the tools do not weight much. Their controllers usually have to be specifically designed for the job, because the movements involved may be very different from

the other applications. Also teaching by moving the arm directly is a very common practice [7].

A standard painting process can be divided in five steps [8]:

- **Part Presentation**. Presenting parts to robots varies widely from application to application. Flat conveyors, overhead conveyors or racks are commonly used at this stage.
- **Coating Process**. The actual coating process generally requires the deposition of multiple layers of different materials, for example: water base, electrostatic spray, powder coat, wax spray, adhesives and sealants.
- **Curing**. After the coating is applied, it must be cured. This is normally done over time by applying heat or using ultraviolet lamps.
- **Part Handling**. After the coating is cured, parts can be removed from the rack or conveyor by a human operator or using handling robots.
- **Vision Inspection**. Before shipping, it is necessary to inspect the quality of the coating to ensure that there are no defects.

In Figure 3.3 we can see a typical robotised painting station for automotive production. Other robots may be involved to move the doors or the engine hood.

Figure 3.3: Typical painting station in automotive industry [9].

[7] http://www.robotsltd.co.uk/applications.aspx?app=8

[8] https://www.motoman.com/robotic-painting

[9] https://www.ecvv.com/product/4774992.html

3.2.1 Example of trajectory planning for spray painting

As an example of trajectory planning for robotic spray painting, we refer to the work «Novel integrated offline trajectory generation approach for robot assisted spray painting operation» by Andulkar, Chiddarwar, and Marathe [1]. The algorithm proposed in the article generates the desired trajectory by dividing the surface iteratively into different *sections*. For each generated *section*, a *section point* and corresponding *section normal* is computed. By connecting the *section points* using linear segments, different paint passes are generated, which eventually form the spray gun trajectory. The methodology assumes to work in a structured environment where the necessary parameters are known. A schematic representation of this algorithm is shown in Figure 3.4.

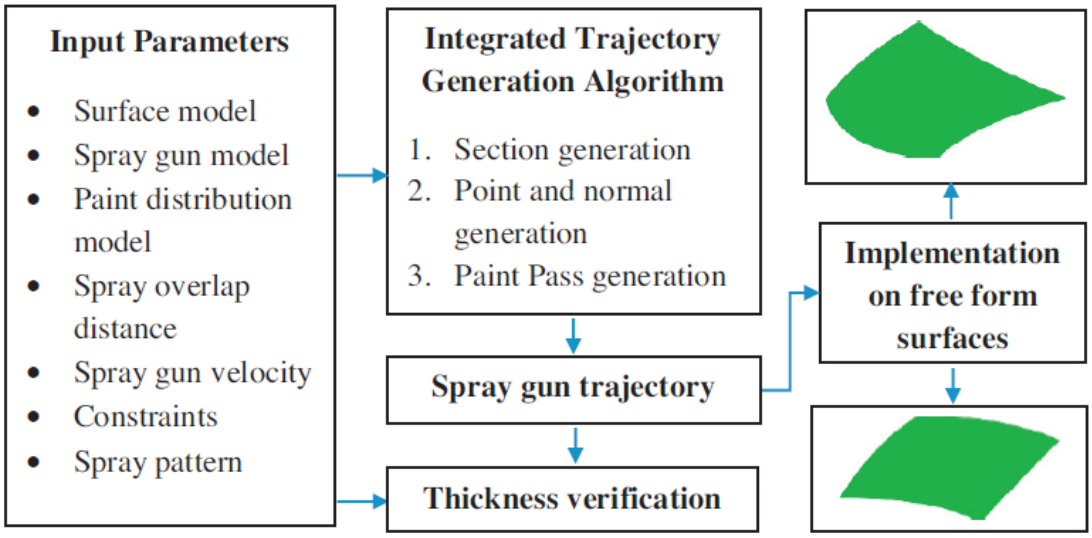

Figure 3.4: Methodology for trajectory generation.

Input models

Firstly the model of the surface is needed. The representation adopted here is the triangular approximation of a free-surface (see Figure 3.5). Each triangle has three nodes and is denoted by:

$$T_{n|i,j,k} = T_n(N_i, N_j, N_k) \qquad n = 1, ..., R \quad i, j, k \in 1, ..., V \tag{3.2}$$

where R is the total number of triangles and V the total number of vertices.

The model of the spray gun adopted is cone shaped and the deposition pattern on a flat surface is circular. The radius of the spray circle is R at the surface. See Figure 3.6 for reference.

The desired paint thickness is given by the user and it is used to compute the optimal overlap distance d_{opt} and the optimal velocity v_{opt}, given a model of the paint distribution.

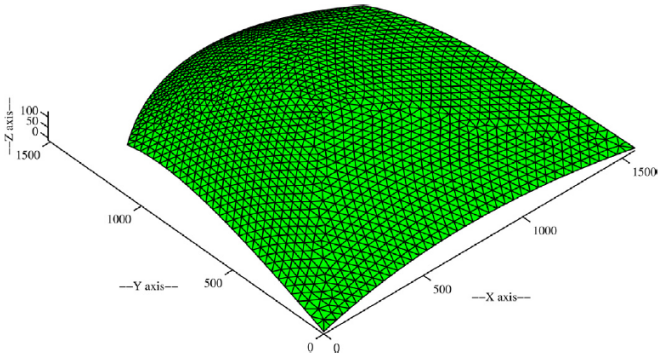

Figure 3.5: Triangular approximation of car hood.

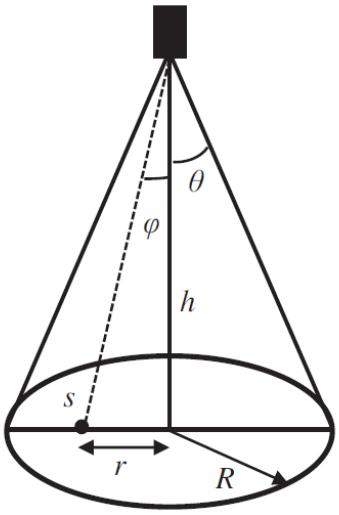

Figure 3.6: Spray gun model.

Planning algorithm

The planning algorithm is divided in three phases.

Phase 1. Based on the paint distribution model and input parameters, optimal overlap distance d_{opt} and optimal velocity v_{opt} are computed. From the global pool of triangles $T_{n|i,j,k}$ we select a subset $S_{s|i,j,k}$ in the range of $0 \leq x, y \leq 2R - d_{opt}$ where the origin is at a corner of the surface. Since the surface has an arbitrary form, the selected triangles may not be all covered by the spray circle. Hence the

triangles are sorted again based on summation of their areas. The spray circle area is taken as $\Pi(R-d_{opt}/2)^2$, assuming that the surface curvature does not vary highly within the range considered. If it is not true, then a lower spray radius R should be considered. Starting from the triangle with lowest y of the selected subset $S_{s|i,j,k}$, we sort the closest triangle until their cumulative area (each triangle as an area of A_s) is lower than the area of the spray circle:

$$\sum A_s \leq \Pi(R - d_{opt}/2)^2 \tag{3.3}$$

The other one are rejected. The remaining triangles $S'_{s|i,j,k}$ forms a *section*. In Figure 3.7 the blue area denotes the triangles $S_{s|i,j,k}$, while the $S'_{s|i,j,k}$ triangles are delimited by a thick border.

Phase 2. A *section point* is generated by taking average of x, y and z coordinates of all the centres of the triangles $S'_{s|i,j,k}$ from the previously generated *section*:

$$x_{avg} = \frac{\sum S'_{s|x}}{m} \qquad y_{avg} = \frac{\sum S'_{s|y}}{m} \qquad z_{avg} = \frac{\sum S'_{s|z}}{m} \tag{3.4}$$

where $S'_{s|x},\ S'_{s|y},\ S'_{s|z}$ are the centres and m are the total number of triangles in a *section*. Since the point $(x_{avg},\ y_{avg},\ z_{avg})$ may not lie on the surface, as a *section point* is selected the closest point to this average on the surface.

The generation of the *section normal* is similar, but since the meshing of the surface may not be uniform, also the area of the triangles is considered:

$$\vec{n}_a = \frac{\sum_{s=1}^{m} A_s \vec{n}_s / \sum_{s=1}^{m} A_s}{|| \sum_{s=1}^{m} A_s \vec{n}_s / \sum_{s=1}^{m} A_s | \, |} \tag{3.5}$$

where $\vec{n}_a$ is the *section normal* and $\vec{n}_s$ is the normal of a generic triangle. In Figure 3.7 the black arrow denotes a *section point* with its *section normal*. The spray gun is oriented along the normal with opposite direction.

Phase 3. In the last phase the raster pattern is computed.

The first paint pass is generated along the x-axis. After the first, a new *section* is generated by increasing the selection range for the x coordinates by a value x_{incr}. This value is selected based on the mesh size of surfaces, surface curvature and the optimal velocity v_{opt} (it is generally around a fifth of the spray radius R). Then *Phase 1* and *Phase 2* are repeated until the boundary of the surface is reached.

The second paint pass proceeds along the y-axis by one step. A value y_{incr} is computed based on the local surface curvature. If the surface is flat, this value is $2R - d_{opt}$. Again the new section is computed based on the new range.

The third pass is computed in the same way as the first, but the x_{incr} is subtracted until the boundary of the surface is reached.

The next steps proceed in the same way as the previous one, alternating a pass on the x-axis and an increment on the y-axis until all the surface is covered.

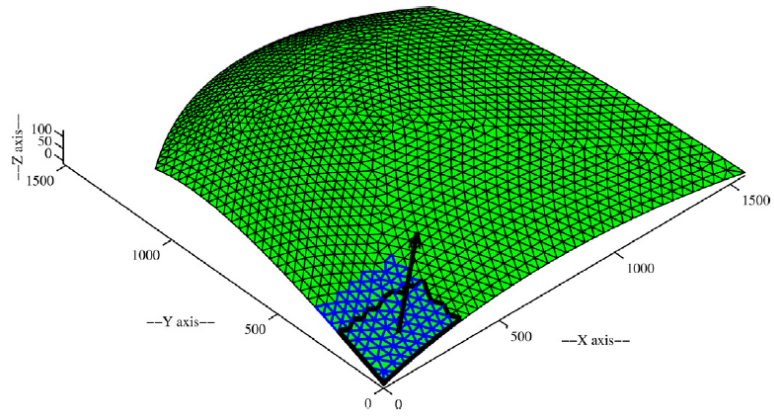

Figure 3.7: Average point and corresponding normal vector for a *section*.

3.3 Additive manufacturing

The term *additive manufacturing* references technologies that grow three-dimensional objects one superfine layer at a time. Each successive layer bonds to the preceding layer of melted or partially melted material. It is possible to use different substances to layer material, including metal powder, thermoplastics, ceramics, composites and glass.

Objects are digitally defined by computer-aided-design (CAD) software that is used to create files that essentially "slice" the object into ultra-thin layers. This information guides the path of a nozzle or print head as it precisely deposits material upon the preceding layer. Alternatively, a laser or electron beam selectively melts each layer in a bed of powdered material. Cooling down, the layers fuse together to form a three-dimensional object [10].

3.3.1 Applications and advantages

Some of the applications where additive manufacturing is used are [11]:

- Prototypes
- Small batch production runs
- Replacement parts
- Rebuilt surfaces

[10] https://www.ge.com/additive/additive-manufacturing

[11] https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Building-the-Future-with-Robotic-Additive-Manufacturing/content_id/6860

- Cladding

The primary elements of a system dedicated to metal additive manufacturing include a high-precision industrial robot, the laser system, an integrated MIG wire and laser head, and the controls system (see Figure 3.8).

The advantages obtainable with respect to a classical manufacturing system are:

- Rapid development of new parts
- Quick design changes without adding tooling costs
- Less base material wasting
- The ability of cladding a less expensive material with a more exotic one for particular properties like high wear resistance

Figure 3.8: Example of metal additive manufacturing robot [12].

3.3.2 Additive manufacturing along curved path

As an example of trajectory planning for robotic additive manufacturing, we refer to the work «Robotic additive manufacturing along curved surface — A step towards free-form fabrication» by Zhang et al. [13].

[12]https://www.compositesworld.com/articles/automation-robots-taking-off-in-commercial-aircraft

Path generation and verification. Starting from a 3D CAD model of the piece, the Arevo Kepler Engine software generates an additive manufacturing (AM) path based on the process parameters such as the diameter of the extruder and the thickness of the building layer. The example of an hollow wing structure (shown in Figure 3.9) demonstrates the "free-form" fabrication concept. A base layer is firstly created; then a frame structure with curved surfaces is built on top of the base at both ends of the wing-shaped object; then, material is built up to form a curvature starting from one of the corners between the base and the frame. The surface is then finished by weaving paths along the curved upper surface of the hollow wing. In this kind of processes a proper cooling device is needed to solidarize the material quickly to form the hollow structure;

Path conversion, simulation and optimization. The generated AM path program is converted to robot path program using one of the existing software such as RAPID. Along with the positions, the extrusion related specific is also converted into robot commands. During the conversion, the robot reachability is checked. If the positions on the path are not reachable, the building plate location needs to be relocated with respect to the robot frame. After the conversion, the robotic additive fabrication process can be simulated and optimized for TCP speed and extrusion rate.

Execution. The generated AM programs have usually a rather large size and they may not fit totally in the execution memory. A method to ensure continuously execution of the program is using of background data loading. Position and process data modules can be loaded by a parallel background task while the robot is executing the building program module in "front" task. If designed properly, this method can perform continuous building process without interruption.

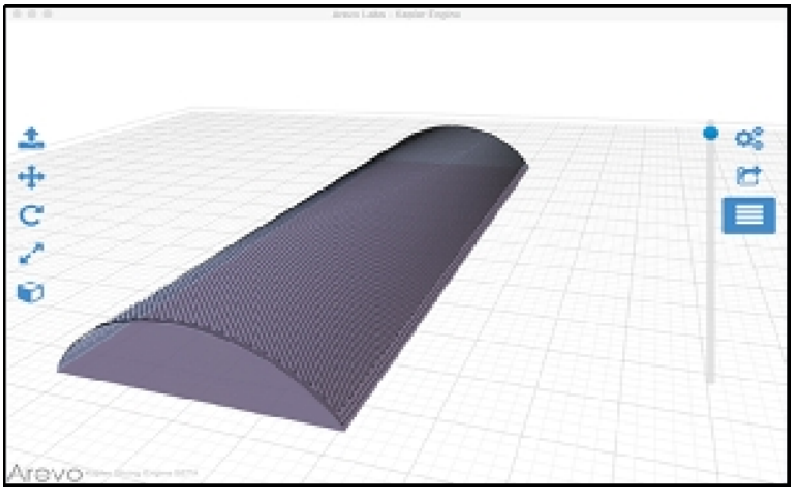

Figure 3.9: A sample of building path for hollow wing structure.

Chapter 4

Spline interpolation

In literature there are many ways to interpolate a dataset with splines, namely cubic spline, Hermite spline, NURBS, smoothing spline and many others. In this chapter three methods are analysed:

- Section 4.1: Cubic Spline
- Section 4.2: Piecewise Cubic Hermite Interpolating Polynomial
- Section 4.3: Smoothing Spline

These methods are at the basis of the new planner that will be developed in Chapter 5, therefore we dedicate this chapter to an in-depth mathematical description of these.

4.1 Cubic Spline

A cubic spline is a spline constructed with third-order polynomials that pass through a set of N control points as we can see in Figure 4.1.

As already seen in section 2.6.2, the spline function has the following form [8]:

$$\begin{cases} \Pi(t) = \Pi_k(t) & t \in [t_k, t_{k+1}], \quad k = 1, ..., N-1 \\ \Pi_k(\tau) = a_{k3}\tau^3 + a_{k2}\tau^2 + a_{k1}\tau + a_{k0} & \tau \in [0, T_k], \quad (\tau = t - t_k,\ T_k = t_{k+1} - t_k) \end{cases} \tag{4.1}$$

with the following constraints:

$$\begin{aligned} \Pi_k(0) = q_k,\ \Pi_k(T_k) = q_{k+1} \qquad & k = 1, ..., N-1 \\ \dot{\Pi}_k(T_k) = \dot{\Pi}_{k+1}(0) = v_k \qquad & k = 1, ..., N-2 \\ \ddot{\Pi}_k(T_k) = \ddot{\Pi}_{k+1}(0) \qquad & k = 1, ..., N-2 \end{aligned} \tag{4.2}$$

We suppose that q_k, t_k, v_1 and v_N are specified by the user, while $v_k,\ k = 2, ..., N-1$ are unknown. We impose for each polynomial the boundary condition on position and velocity :

$$\begin{aligned} \Pi_k(0) &= a_{k0} &&= q_k \\ \dot{\Pi}_k(0) &= a_{k1} &&= v_k \\ \Pi_k(T_k) &= a_{k3}\tau^3 + a_{k2}\tau^2 + a_{k1}\tau + a_{k0} &&= q_{k+1} \\ \dot{\Pi}_k(T_k) &= 3a_{k3}\tau^2 + 2a_{k2}\tau + a_{k1} &&= v_{k+1} \end{aligned} \tag{4.3}$$

Solving for the coefficients we obtain:

$$\begin{aligned} a_{k0} &= q_k \\ a_{k1} &= v_k \\ a_{k2} &= \frac{1}{T_k}\left[\frac{3(q_{k+1}-q_k)}{T_k} - 2v_k - v_{k+1}\right] \\ a_{k3} &= \frac{1}{T_k^2}\left[\frac{2(q_k-q_{k+1})}{T_k} + v_k + v_{k+1}\right] \end{aligned} \tag{4.4}$$

We use the condition on continuity of the accelerations:

$$\ddot{\Pi}_k(T_k) = \ddot{\Pi}_{k+1}(0) \rightarrow 2a_{k2} + 6a_{k3}T_k = 2a_{(k+1)2} \tag{4.5}$$

and substitute the values of the coefficient obtained in (4.4):

$$T_{k+1}v_k + 2(T_{k+1}+T_k)v_{k+1} + T_k v_{k+2} = \frac{3}{T_k T_{k+1}}\left[T_k^2(q_{k+2}-q_{k+1}) + T_{k+1}^2(q_{k+1}-q_k)\right] \tag{4.6}$$

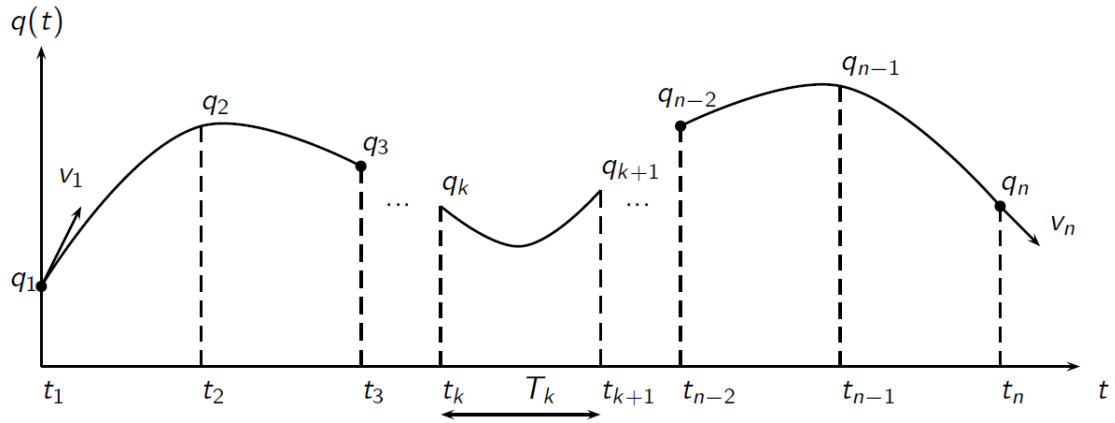

Figure 4.1: Example of a cubic spline

These equations can be expressed in matrix form as:

$$\begin{bmatrix} 2(T_1+T_2) & T_1 & & & & \\ T_3 & 2(T_2+T_3) & T_2 & & & \\ & & \vdots & & & \\ & & T_{N-2} & 2(T_{N-3}+T_{N-2}) & T_{N-3} & \\ & & & T_{N-1} & 2(T_{N-2}+T_{N-1}) \end{bmatrix} \begin{bmatrix} v_2 \\ \vdots \\ v_{N-1} \end{bmatrix} = \begin{bmatrix} \frac{3}{T_1T_2}\left[T_1^2(q_3-q_2)+T_2^2(q_2-q_1)\right]-T_2v_1 \\ \frac{3}{T_2T_3}\left[T_2^2(q_4-q_3)+T_3^2(q_3-q_2)\right] \\ \vdots \\ \frac{3}{T_{N-3}T_{N-2}}\left[T_{N-3}^2(q_{N-1}-q_{N-2})+T_{N-2}^2(q_{N-2}-q_{N-3})\right] \\ \frac{3}{T_{N-2}T_{N-1}}\left[T_{N-2}^2(q_N-q_{N-1})+T_{N-1}^2(q_{N-1}-q_{N-2})\right]-T_{N-2}v_N \end{bmatrix} \tag{4.7}$$

Since the equation is in the form

$$\mathbf{Av} = \mathbf{c}$$

the problem is solvable if $\mathbf{A}^{-1}$ exists. If $T_k > 0$ then $\mathbf{A}$ is diagonally dominant ($|a_{kk}| > \sum_{j\neq k}|a_{kj}|$) and for the Levy-Desplanques theorem it is always non-singular.

Being $\mathbf{A}$ tridiagonal, we give here an efficient method to compute its inverse (based on the Gauss-Jordan method)[3]:

$$v_N = \bar{v}_N, \quad v_i = \bar{v}_i - K_i v_{i+1}, \qquad i = N-1, ...,1 \tag{4.8}$$

with

$$\bar{v}_1 = \frac{c_1}{a_{11}}, \quad \bar{v}_i = \frac{c_i - \bar{v}_{i-1}a_{i,i-1}}{a_{ii} - K_{i-1}a_{i,i-1}}, \qquad i = 2, ..., N \tag{4.9}$$

$$K_1 = \frac{a_{12}}{a_{11}}, \quad K_i = \frac{a_{i,i+1}}{a_{ii} - K_{i-1}a_{i,i-1}}, \qquad i = 2, ..., N \tag{4.10}$$

where a_{ij} are the elements of $\mathbf{A}$ and c_{ij} are the elements of $\mathbf{c}$.

Once the velocities are computed, we can compute the coefficients (4.4) to find the interpolating cubic spline.

4.2 Piecewise Cubic Hermite Interpolating Polynomial

A *cubic Hermite spline* is a spline where each polynomial is specified in Hermite form [12]:

$$\Pi(t) = h_{00}(t)q_0 + h_{10}(t)d_0 + h_{01}(t)q_1 + h_{11}(t)d_1 \qquad t \in [0,1] \tag{4.11}$$

where q_0, q_1 are the values of the spline at the end points, d_0, d_1 are the values of the derivatives in the same points and h_{ii} are the Hermite basis functions.

In Figure 4.2 there is a visual representation of the basis functions, while their analytical expressions are:

- $h_{00}(t) = (1 + 2t)(1 - t)^2$
- $h_{10}(t) = t(1 - t)^2$
- $h_{01}(t) = t^2(3 - 2t)$
- $h_{11}(t) = t^2(t - 1)$

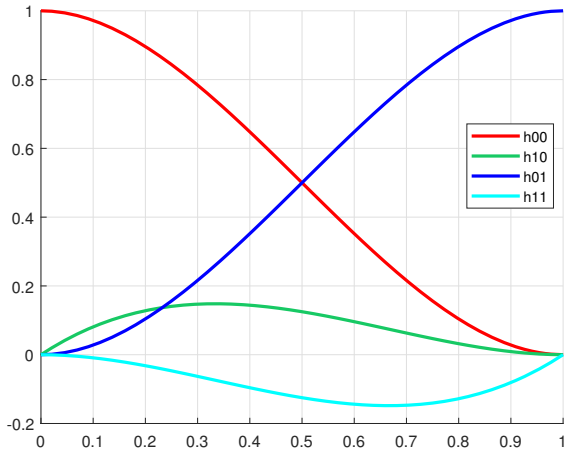

Figure 4.2: The four Hermite basis functions on the unitary interval.

Through an affine change of variables, we obtain the expression of the interpolating polynomial on an arbitrary interval $[t_k, t_{k+1}]$:

$$\Pi_k(t) = h_{00}(T)q_k + h_{10}(T)(t_{k+1} - t_k)d_k + h_{01}(T)q_{k+1} + h_{11}(T)(t_{k+1} - t_k)d_{k+1}$$
$$t \in [t_k, t_{k+1}], \quad T = \frac{t - t_k}{t_{k+1} - t_k} \tag{4.12}$$

To interpolate a set of path points, we need to properly choose the tangents at every point and then apply the equation (4.12) to each interval. The choice of tangents is not unique and there are several options available. Here we propose the one implemented in MatLab [10]:

- Define $\delta_k = (q_{k+1} - q_k)/T_k, \qquad T_k = t_{k+1} - t_k$ as the slope of the segment between the points k and k+1

- If $\delta_k = 0$ or $\delta_{k-1} = 0$, then $d_k = 0$
- If $\text{sgn}(\delta_k) \neq \text{sgn}(\delta_{k-1})$ then $d_k = 0$
- In the other cases, use the weighted harmonic mean:

$$d_k = \frac{w_1 + w_2}{\frac{w_1}{\delta_{k-1}} + \frac{w_2}{\delta_k}} \qquad (4.13)$$
$$\text{with} \quad w_1 = 2T_k + T_{k+1}, \;\; w_2 = T_k + 2T_{k-1}$$

The resulting polynomial is called *Piecewise cubic Hermite interpolating polynomial* (PCHIP), and it is shape preserving, maintaining the monotonicity of the path if possible, and avoiding overshoots. In Figure 4.3, we can see the difference between a cubic spline interpolation and a PCHIP interpolation. The appropriate interpolator should be chosen accordingly to the necessity of the user.

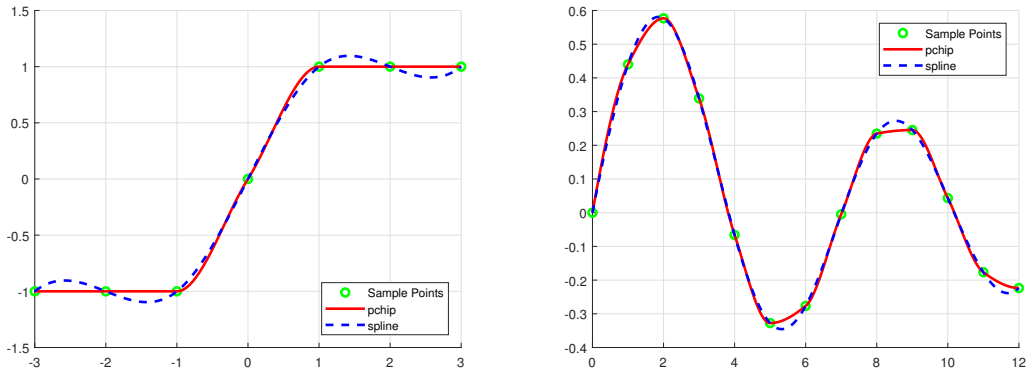

Figure 4.3: Interpolation of two different dataset with cubic spline and PCHIP

4.3 Smoothing Spline

Smoothing splines are functions that perform a regularized regression over the natural spline basis. This means that the N path points (defined as (t_j, y_j), $j = 1,...,N$) are not exactly interpolated, but a certain amount of error is tolerated in order to obtain a smoother path.

First of all we define the set of basis functions. For the sake of simplicity, here we choose the *truncated power basis*, but a more efficient set in computational term is the *B-spline basis*. Considering a set of N points defined in $t_1, ..., t_N$ and a kth order spline interpolation ($k = 3$ for cubic splines), the truncated power basis are

defined as:

$$\begin{aligned} g_1(t) = 1, g_2(t) = t, ..., g_{k+1}(t) = t^k, \\ g_{k+1+j}(t) = (t - t_j)_+^k \qquad j = 1, ..., N \end{aligned} \tag{4.14}$$

where $t_+ = \max\{t,0\}$, that is the positive part of t.

A cubic spline with n basis, is written as:

$$\Pi(t) = \sum_{j=1}^{n} \beta_j g_j(t) \tag{4.15}$$

for $n = N + 4$; β is a vector of coefficients that is chosen to minimize the following cost function[11]:

$$\sum_{j=1}^{n} (y_j - \Pi(t_j))^2 + \lambda \int (\Pi''(t))^2 dt \tag{4.16}$$

This criterion trades off the least square error of Π over the path points with a *regularization term* that penalizes the oscillations of the function. $\lambda \geq 0$ is called *smoothing parameter* and can be tuned to increase or decrease the weight of the regularization term.

We can write (4.16) in matrix form as:

$$||y - G\beta||_2^2 + \lambda \beta^T \Omega \beta \tag{4.17}$$

where

- $G \in \mathbb{R}^{nxn}$ is the basis matrix defined as:

$$G_{ij} = g_j(t_i) \qquad i, j = 1, ..., n$$

- $\Omega \in \mathbb{R}^{nxn}$ is the penalty matrix defined as:

$$\Omega_{ij} = \int g_i''(t) g_j''(t) dt \qquad i, j = 1, ..., n$$

Now we have to find the optimal β that minimizes (4.17):

$$\hat{\beta} = \arg\min_{\beta} [||y - G\beta||_2^2 + \lambda \beta^T \Omega \beta] \tag{4.18}$$

The solution is similar to the solution of a least square problem:

$$\hat{\beta} = (G^T G + \lambda \Omega)^{-1} G^T y \tag{4.19}$$

Substituting (4.19) in (4.15) we obtain the *smoothing spline*:

$$\hat{\Pi}(t) = \sum_{j=1}^{n} \hat{\beta}_j g_j(t) \tag{4.20}$$

In Figure 4.4, we can see the comparison between a cubic spline interpolation and smoothing spline interpolation of a "noisy" dataset.

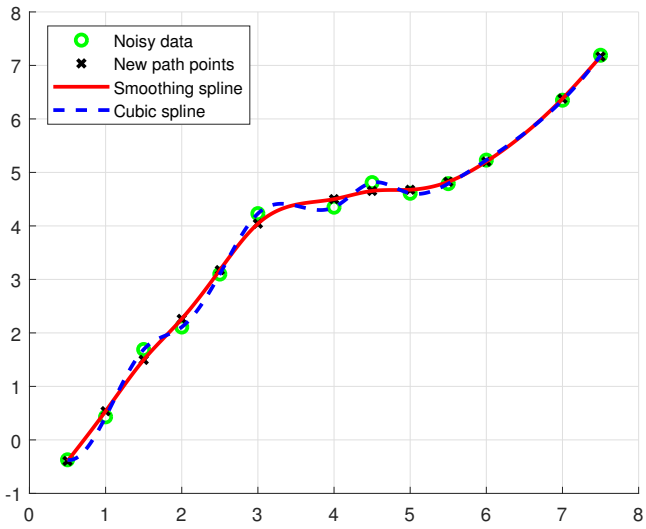

Figure 4.4: Interpolation of a "noisy" dataset with cubic spline and smoothing spline

Chapter 5

Implemented methods

In this chapter the methods chosen for code implementation and testing are presented.

Section 5.1 briefly explains the theory behind curve reparametrization. Section 5.2 is about global analysis and planning of the trajectory using different interpolation techniques. Section 5.3 explains the local version of the same planners, intended for a possible real-time implementation. Finally, in Section 5.5 there are some considerations about a different approach that was discarded after some testing.

5.1 Moving along a curve with specified speed

The topics on this section refers to the work *Moving along a curve with specified speed* by Eberly [5].

A parametric curve $\mathbf{Y}(u)$ can be seen as the path travelled by a particle whose position is $\mathbf{Y}(u)$ at time u. The velocity of the particle is the rate of change of position with respect to the variable u:

$$\mathbf{V}(u) = \frac{d\mathbf{Y}}{du} \tag{5.1}$$

The velocity vector is tangent to the position vector for each u. The speed $\sigma(u)$ of the particle is given by the length of the velocity vector:

$$\sigma(u) = |\mathbf{V}(u)| = \left|\frac{d\mathbf{Y}}{du}\right| \tag{5.2}$$

In continuous processes it is required that the speed is constant at every time, which means that $\sigma(u) = c$. We also need to take into account the acceleration/deceleration phases and occasional braking on critical path points. Therefore we will consider $\sigma(u)$ as a generic speed function.

5.1.1 Reparametrization for specified speed

Let $\mathbf{Y}(u), u \in [u_{min}, u_{max}]$ be a curve with a generic parametrization and $\mathbf{X}(t), t \in [t_{min}, t_{max}]$ be a time parametrization of the same curve, so that the speed is a specified function $\sigma(t)$.

The total arc-length of the curve is:

$$L = \int_{u_{min}}^{u_{max}} \left| \frac{d\mathbf{Y}}{du} \right| du \tag{5.3}$$

The speed $\sigma(t)$ is imposed and it has to respect the following constraint coming from calculus and physics:

$$\int_{t_{min}}^{t_{max}} \sigma(t) dt = L \tag{5.4}$$

A simple way to meet constraint (5.4) is to plan a generic function $\rho(t)$ with the desired shape, and then apply a scaling factor. The actual speed used is:

$$\sigma(t) = \frac{L\rho(t)}{\int_{t_{min}}^{t_{max}} \rho(t) dt} \tag{5.5}$$

5.1.2 Numerical solution

Since $\mathbf{Y}(u)$ and $\mathbf{X}(t)$ are different parametrizations of the same curve, we have that $\mathbf{Y}(u) = \mathbf{X}(t)$. This implies a relationship between u and t. We can apply the chain rule from calculus:

$$\frac{d\mathbf{X}}{dt} = \frac{d\mathbf{Y}}{du}\frac{du}{dt} \tag{5.6}$$

and compute the speed:

$$\sigma(t) = \left| \frac{d\mathbf{X}}{dt} \right| = \left| \frac{d\mathbf{Y}}{du} \right| \frac{du}{dt} \tag{5.7}$$

Here we assume that u and t increase jointly, which means that we can skip the absolute value on du/dt since it is always greater than or equal to zero.

Equation (5.7) can be rewritten as:

$$\int_{u_{min}}^{u} \left| \frac{d\mathbf{Y}(\mu)}{du} \right| d\mu = \int_{t_{min}}^{t} \sigma(\tau) d\tau = l \tag{5.8}$$

where l is the length of the path travelled during the time interval $(tmin, t)$.

Now we define $F(u) = \int_{u_{min}}^{u} \left| \frac{d\mathbf{Y}(\mu)}{du} \right| d\mu - l$. Given the value of l, the problem is to find the value of u so that $F(u) = 0$. This is a root find problem that can be solved with any suitable method.

The procedure to obtain the reparametrization is here summarised:

1. Define the time sequence of the output samples $T = t_i, \quad i = 1, .., n$

2. For each t_i, compute $l_i = \int_{t_{min}}^{t_i} \sigma(\tau)d\tau$

3. For each l_i, obtain the corresponding u_i solving $F(u_i) = 0$

4. Set $\mathbf{X}(t_i) = \mathbf{Y}(u_i)$ for each i.

Some pseudocode for computing u given t:

```
%% parameters definition:

umin, umax      % curve parameter interval, [umin, umax]
Y(u)            % gives the position for u in [umin, umax]
DY(u)           % derivative of Y: dY/du
tmin, tmax  % user-specified time interval, [tmin, tmax]
Sigma(t)    % user-specified velocity profile

%% functions

function v = speed(u) % length of the speed vector in u
        v = ||DY(u)||;
end

function L = ArcLength(u) % length of the path from umin to u
        L = integral(umin, u, @(x) speed(x));
end

function u = GetCurveParameter(s) % finds the u that gives an arc
   length equal to s
        u = root_find([umin, umax], @(x) ArcLength(x) - s);  %solves:
            ArcLength(x) - s = 0
end

function u = GetU(t) % coputes the u given the time t
        el = integral(tmin, t, @(x) Sigma(x));
        u = GetCurveParameter(el);
end
```

5.2 Trajectory analysis and global planning

Firstly, a method to perform an off-line analysis of the input data is proposed. The objective is to give to the user a preliminary simulation comparing the results obtained with different interpolation techniques.

There are two methods to interpolate the user-given data: *global interpolation* and *local interpolation*. The former considers all the available points, the latter just a small subset at each time. In this stage the computation is done globally. Since a global interpolation gives better results with respect to a local interpolation, we developed this analysis also to have reference values for the testing of the local trajectory planners.

In the following subsections a step-by-step explanation of how the trajectories are computed is reported. The steps covered are: inputs definition, path generation, path curvature computation, temporal law inclusion, generation of the velocity profile, performance indices and MatLab simulation.

5.2.1 User-defined inputs

The input data requested for the trajectory analysis are the following ones:

- Path points set: it is the set of points used to generate the path. These points are intended to be crossed with high precision.
- Target constant velocity: it is the Cartesian constant speed required in the continuous process.
- Maximum acceleration: it is the maximum Cartesian acceleration allowed. It is used to compute brakes in case of curvature with very small radius.
- Starting and ending velocities: they are generally set to zero, but they can be specified differently if needed.
- Sampling time: it is the time distance requested for the output samples.

For the PCHIP interpolation there is an extra input to enable or disable the critical points correction as detailed in Subsection 5.2.3. For the interpolation with re-sampling is requested to specify the number of points of the re-sampled trajectory, $\#samples$, or the approximate distance between two samples, l_{step} (the two quantities are related by the total path length: $L_{path} = \#samples \cdot l_{step}$)

5.2.2 Interpolation techniques and path generation

The trajectory analysis is performed with four different interpolation techniques:

- Cubic splines (Section 4.1): a good compromise between smoothness and giggling. Continuity is guaranteed up to acceleration. It works well with more rounded path and no straight lines.
- PCHIP splines (Section 4.2): the interpolation has more sharp edges and the acceleration is not continuous. This interpolator is shape preserving and

introduces no giggling, thus it is suitable to interpolate paths that have straight lines.

- Smoothing splines (Section 4.3): its properties are similar to the cubic spline interpolator. The path points are traversed with an error in favour of a smoother interpolation.
- Cubic spline on re-sampled PCHIP path: it combines the continuity property of the cubic spline planner with the shape preserving path of the PCHIP planner. Due to the path points being re-sampled, the exact crossing of the user-defined points is not guaranteed.

Independently from the computation technique, the path is described as a three-dimensional parametric curve: $\Pi(u) = (\Pi_x(u), \Pi_y(u), \Pi_z(u))$. This means that three interpolations are required, one for each Cartesian axis. Firstly, we define the common parameter u. Since here we are not interested in the time law, we can just use the normalized cumulative distance between the path points (p_i, $i = 1, ..., N$):

$$\begin{aligned} L &= \sum_{i=2}^{N} ||p_i - p_{i-1}|| \\ u(1) &= 0 \\ u(i) &= u(i-1) + ||p_i - p_{i-1}||/L, \quad i = 2, ..., N \end{aligned} \tag{5.9}$$

After that we can compute the spline functions for each axis:

$$\begin{aligned} \Pi_x(u) &= spline(u, p_x) \\ \Pi_y(u) &= spline(u, p_y) \\ \Pi_z(u) &= spline(u, p_z) \end{aligned} \tag{5.10}$$

where p_x, p_y, p_z are the x, y, z components of the path points and $spline(u, p)$ is a generic interpolating function.

Summarizing, cubic and smoothing spline paths are generated directly using the in-built MatLab functions; PCHIP and cubic spline with re-sampling require an additional step before applying the MatLab functions. In particular the PCHIP path can be smoothed, on user request, with the algorithm described in Subsection 5.2.3, while the cubic spline with re-sampling, as the name suggests, requires a re-sampling (see Subsection 5.2.4) of the path points before the standard cubic interpolation.

5.2.3 Smoothing narrow angles in PCHIP interpolation

In many cases, the path given by a PCHIP interpolation tends to be better than the other interpolations. The main reasons are that the straight lines are preserved

and no overshoots are introduced. However this benefit comes with some drawbacks that the users should be aware of: right or acute angles introduce very small curve radii, or even stationary points in the path that can not be traversed with constant speed even if high accelerations are permitted. In particular, stationary points require the manipulator to stop and restart the motion.

For the aforementioned reasons, an algorithm to correct and avoid those critical issues is proposed.

Ideally we would like to substitute the sharp corners in the path with circumference arcs to obtain smoother transitions between two segments. The idea is similar to the trajectory planning technique called *linear polynomials with parabolic blends* (see Subsection 2.6.3), where linear segments are blended with arcs of parabola to obtain a continuous motion between path points. In our case we are not reprogramming the planner to execute parabolic blends in specific points, but we just substitute the critical points with couples of points that the planner can connect more easily with arcs.

A point is critical if it meets at least one of the two following conditions:

- It is a stationary point: the slope computed as described in (4.2), is equal to zero

  or

- The angle between the vectors that connect it to the previous and following points is included between 75° and 180°. The convention used to compute that angle is shown in Figure 5.1.

The case of a stationary point with an angle of 180° is not considered here. This case occurs when a trajectory backtracks and can not be corrected with the algorithm proposed. The user should consider to employ other interpolators or programming the backtrack in a different way.

Once that the critical points are located, we compute the couples of points that substitute each of them. Referring to Figure 5.1, the points t_1, t_2 are selected as the tangent points of an inscribed circumference with radius R. d can be directly specified by the user, or can be computed as: $d = v_{target}^2/a_{max} \cdot tan(\theta/2)$, where v_{target}, a_{max} are parameters imposed by the user (as described in Subsection 5.2.1).

In Figure 5.2 an application of the algorithm is shown.

5.2.4 Path re-sampling

When the distances between couples of consecutive path points are very different, the cubic interpolation may not give acceptable results. In these cases a re-sampling of the original points may be needed in order to have better final interpolations.

The algorithm here proposed performs a fixed step re-sampling of the path interpolated with PCHIP spline. The information on the original path points is lost

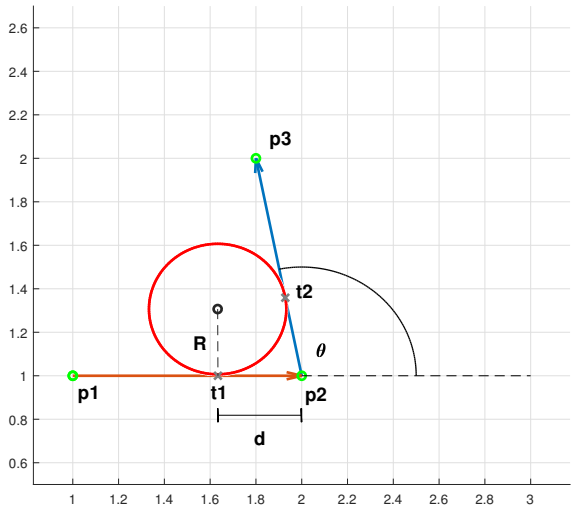

Figure 5.1: Conventions used for computing the new points.

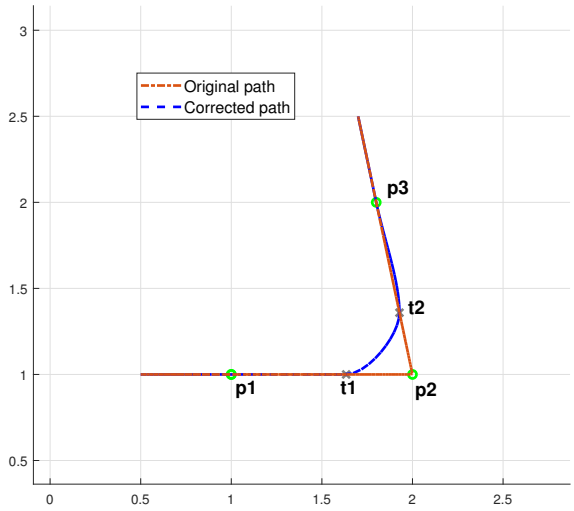

Figure 5.2: Comparison between a PCHIP planned path with and without correction.

after the first interpolation, so there is no guarantee of exact crossing. The straight lines are interpolated with precision and overshoots are present only in the intervals near points with high curvature. The user can choose the length of the steps or the total number of samples to decrease the crossing error and the amplitude of the overshoots.

The relevant steps needed to obtain a re-sampling are explained in Subsection 5.1.2. In this specific case the fixed distance between two samples is obtained imposing a constant speed profile.

In Figure 5.3 there is a comparison between two paths both interpolated with

cubic splines. One dataset has points with an unbalanced spacing, the other one is more evenly spaced. In Figure 5.4 we can see an application of the re-sampling algorithm.

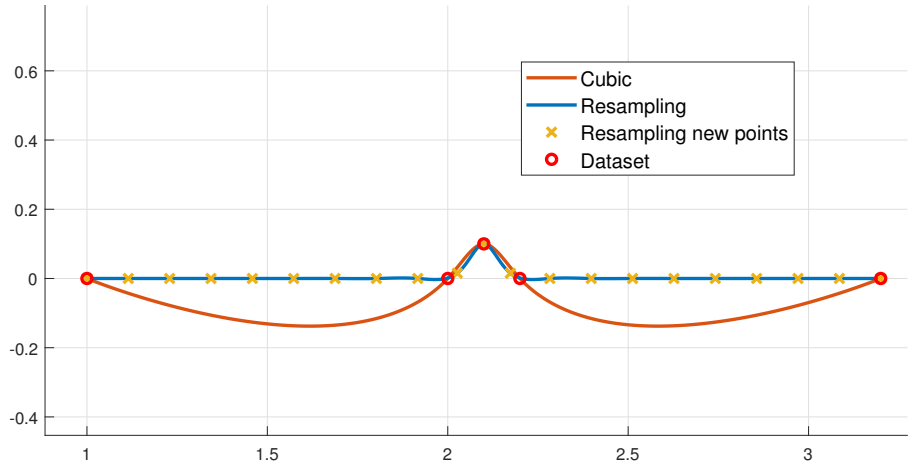

Figure 5.3: Cubic interpolation of an even-spaced dataset vs. an uneven one.

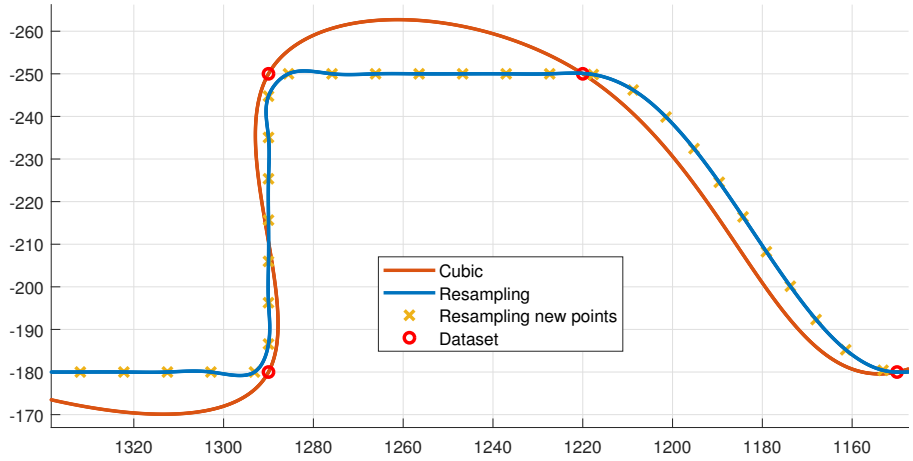

Figure 5.4: Application of the re-sampling algorithm.

5.2.5 Path curvature

The curvature is another important aspect for the geometry analysis of a path. The correlation between velocity and centripetal acceleration is given indeed by the curvature in each point of the path. For our purposes, we use the concepts of *osculating circle* and *radius of curvature*, which is just the inverse of the curvature.

Firstly we introduce the *Frenet Frame* [6]. Given a 3D curve $f(u)$, the *Frenet*

frame (or *Frenet trihedron*) at u is the triple ($\mathbf{t}$,$\mathbf{n}$,$\mathbf{b}$) consisting of three orthogonal unit vectors such that:

- $\mathbf{t} = \frac{\dot{f}(u)}{||\dot{f}(u)||}$ is the *unit tangent vector* at u
- $\mathbf{n} = \frac{-(\dot{f}(u)\cdot\ddot{f}(u))\dot{f}(u)+||\dot{f}(u)||^2\ddot{f}(u)}{||-(\dot{f}(u)\cdot\ddot{f}(u))\dot{f}(u)+||\dot{f}(u)||^2\ddot{f}(u)||}$ is the *principal normal vector* to t at u
- $\mathbf{b} = \mathbf{t} \times \mathbf{n}$ is the *binormal vector* at u

Focusing on a small neighbourhood of u, the curvature can be seen as the rate at which the normal $\mathbf{n}$ turns. The point around which the normal vector turns is the centre of curvature C at u and the osculating circle is the circle centred in C, and tangent to the curve at $f(u)$.

The radius of curvature at u is defined as:

$$\mathcal{R}(u) = \frac{||\dot{f}(u)||^3}{||\dot{f}(u) \times \ddot{f}(u)||} \tag{5.11}$$

The curvature at u is:

$$k(u) = \frac{||\dot{f}(u) \times \ddot{f}(u)||}{||\dot{f}(u)||^3} \tag{5.12}$$

Knowing $\mathcal{R}$, we can compute the centripetal component of the acceleration at u as:

$$a_c(u) = \frac{v(u)^2}{\mathcal{R}(u)} \tag{5.13}$$

Figure 5.5 shows two Frenet frames on consecutive points of a 3D elix path.

5.2.6 Velocity profile and temporal law inclusion

After the path generation, it is necessary to impose a proper temporal law in order to obtain the desired trajectory. This step can be done with the technique described in Section 5.1: the parameter used for the interpolation is changed with a time parameter that follows the desired temporal law.

Now we have to determine a speed profile that meets the constraints imposed by the user. As a starting point, the trapezoidal velocity profile (Subsection 2.5.2) is chosen. Given the target constant velocity v_t, the maximum acceleration A_{max}, the starting and final velocities v_0, v_f and the total length of the path L, we compute the commutation instants and the total execution time (t_f):

$$\begin{aligned} t_1 &= \frac{v_t - v_0}{A_{max}} \\ t_2 &= t_1 + \frac{L}{v_t} + \frac{(v_t - v_f)^2 - (v_t - v_0)^2}{2A_{max}v_t} - \frac{v_0(v_t - v_0)}{A_{max}v_t} - \frac{v_t - v_f}{A_{max}} \\ t_f &= t_2 + \frac{v_t - v_f}{A_{max}} \end{aligned} \tag{5.14}$$

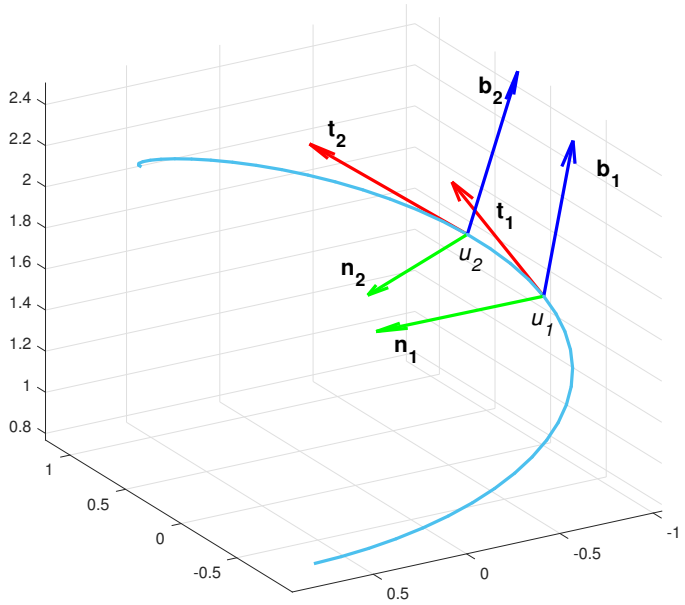

Figure 5.5: Example of Frenet frames at u_1 and u_2

Then the speed profile $v(t)$ is just a piecewise function:

$$\begin{cases} v(t) = A_{max}t + v_0 & t < t_1 \\ v(t) = v_t & t_1 \leq t \leq t_2 \\ v(t) = -A_{max}(t - t_2) + v_t & t > t_2 \end{cases} \tag{5.15}$$

Now we can compute a first re-parametrization imposing the trapezoidal speed profile. The re-parametrization is numerical and the sampling time used is h (the same given by the user). After that, we have a constant speed trajectory, but the accelerations may not be respected, since the centripetal accelerations introduced by the path curvature have to be taken into account. Therefore the curve radii are computed (Subsection 5.2.5) and the maximum velocity allowed at each point is found:

$$v_{max,i} = min\left(v(t_i), \sqrt{A_{max}\mathcal{R}(t_i)}\right) \quad i = 1, ..., n = t_f/h \tag{5.16}$$

The next step is to synchronize the decelerations with the path. This means that the cumulative area at a time t_k of the re-computed speed profile v_{max} needs to be equal to the distance covered with speed $v(t)$:

$$\sum_{i=1}^{k} \frac{v_{max,i} + v_{max,i+1}}{2} dt_i = \int_0^{t_k} v(\tau)d\tau \tag{5.17}$$

At each time instant we can compute the time variation dt_i to make sure that the

area of the interval is the same of the corresponding interval of the function $v(t)$:

$$\frac{v_{max,i} + v_{max,i+1}}{2} dt_i = \frac{v(t_i) + v(t_i + h)}{2} h := S \quad i = 1, ..., n-1 \tag{5.18}$$

The speed variation must also be programmed not to surpass the acceleration constraint. We choose to recompute $v_{max,i+1}$ using the constraint on the acceleration:

$$\left(\frac{v_{max,i+1} - v_{max,i}}{dt_i}\right)^2 + \left(\frac{v_{max,i}^2}{\mathcal{R}(t_i)}\right)^2 \leq A_{max}^2 \tag{5.19}$$

where $\frac{v_{max,i+1} - v_{max,i}}{dt_i}$ is the tangential acceleration and $\frac{v_{max,i}^2}{\mathcal{R}(t_i)}$ is the centripetal acceleration.

In (5.19) $v_{max,i+1}$ and dt_i are the unknown variables. Equation (5.18) is solved by dt_i:

$$dt_i = \frac{2S}{v_{max,i} + v_{max,i+1}} \quad i = 1, ..., n-1 \tag{5.20}$$

and substituted in (5.19), where the inequality sign is replaced by an equality sign, since we want the velocity holes to be as shortest as possible:

$$((v_{max,i+1})^2 - (v_{max,i})^2)^2 = 4S^2(A_{max}^2 - (v_{max,i})^2/\mathcal{R}(t_i)) := \Delta \quad i = 1, ..., n-1 \tag{5.21}$$

Now we have to consider two cases: $v_{max,i+1} > v_{max,i}$ and $v_{max,i+1} < v_{max,i}$. Since the minimum velocities must be respected in order to keep the centripetal accelerations inside the upper bound, we can apply the algorithm twice: once for $i = 1, ..., n-1$ considering only the case of $v_{max,i+1} > v_{max,i}$ and once for $i = n, ...,2$ for $v_{max,i-1} > v_{max,i}$. In this second case we can use the same procedure substituting $v_{max,i+1}$ with $v_{max,i-1}$ in the formulas. With this trick both the minimum velocities and acceleration constraints are respected and Δ, defined in (5.21), is guaranteed to be always greater then or equal to zero.

Given that $v_{max,i+1} > v_{max,i}$, we can take the square root of (5.21) and consider only the positive solution:

$$(v_{max,i+1})^2 = (v_{max,i})^2 + \sqrt{\Delta} \quad i = 1, ..., n-1 \tag{5.22}$$

Once again we take the square root and choose the positive solution, because negative velocities (backtrack) are not allowed:

$$v_{max,i+1} = \sqrt{(v_{max,i})^2 + \sqrt{\Delta}} \quad i = 1, ..., n-1 \tag{5.23}$$

Now that both $v_{max,i}$ and $v_{max,i+1}$ are known, we can substitute them in (5.20) to find dt_i. The new time instants are simply the sum of the time variations: $t_i = \sum_{k=1}^{i} dt_k$. In Figure 5.6 we can see an application of this procedure. The velocity are plotted with respect to their indices and not to time.

The continuous velocity profile $v_{max}(t)$ is obtained with a linear interpolation of the couples $(v_{max,i}\ ;\ t_i)$.

Applying again the re-parametrization procedure with sampling time h and velocity profile $v_{max}(t)$, we obtain a trajectory that meets all the constraints defined by the user.

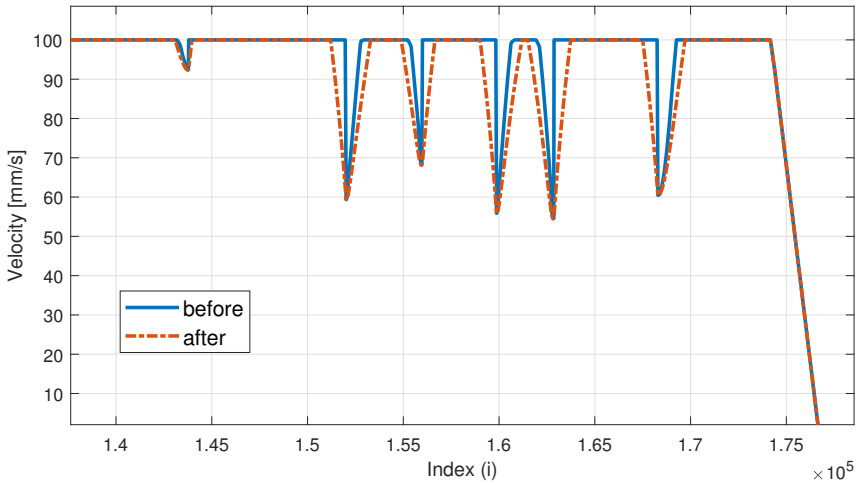

Figure 5.6: v_{max} before and after the acceleration correction, plotted with respect to the index i.

5.2.7 Performance indices and MatLab simulations

To compare the results obtained with different techniques, four metrics are proposed:

- Maximum crossing error (L_{cross} [mm]): it measures the maximum error between the path points defined by the user and the interpolated path.
- Number of velocity holes (N_{holes}): it counts the number of decelerations programmed to meet the acceleration constraint.
- Average hole length (T_{hole} [s]): it measures the average duration of a velocity hole.
- Motion duration (T_{motion} [s]): it measures the overall duration of the motion.

The global planning interpolation is tested on three different datasets. Two datasets are from real processes used by COMAU clients, while the third one is made ad hoc to test the performance of the interpolators. All the plots shown in

Figures 5.8, 5.9, 5.10, 5.11 and 5.12 are obtained with a target speed of 100 mm/s and a maximum acceleration of 2000 mm/s^2.

The first process requires to deposit a homogeneous amount of sealing on a box containing electronic boards. On Figure 5.7 we can see a photo of a piece. The constraints on this specific process are very strict because once closed, the boxes need to be waterproof. Small errors on the path or non homogeneous sealing deposit can compromise the functionality of a box. In figure Figure 5.8 the plots of the path obtained with the four described techniques are shown. In Figure 5.11 and Figure 5.12 there are the velocity profile obtained with the cubic and PCHIP interpolations and the relative accelerations. Table 5.1 summarizes the results of this simulation.

Table 5.1: Box dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]
Cubic	$9.65 \cdot 10^{-2}$	6	$2.57 \cdot 10^{-2}$	3.55
PCHIP	$9.92 \cdot 10^{-2}$	14	$5.07 \cdot 10^{-2}$	3.69
Smoothing	$9.77 \cdot 10^{-1}$	0	0	3.48
Re-sampling (50 samp.)	$6.67 \cdot 10^{-1}$	5	$3.76 \cdot 10^{-2}$	3.53

The second one is also a sealant deposit process applied on an engine hood. The original path points are represented with a scale of 1:2.5 to fit on the operational space of the robot used for the tests (details will be provided in Chapter 6). Figure 5.9 shows a plot of the path interpolated with cubic spline. The results are summarized in Table 5.2.

Table 5.2: Engine hood dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]
Cubic	$9.91 \cdot 10^{-2}$	3	$1.07 \cdot 10^{-1}$	20.1
PCHIP	$9.97 \cdot 10^{-2}$	5	$8.88 \cdot 10^{-2}$	20.1
Smoothing	1.22	2	$8.10 \cdot 10^{-1}$	20.0
Re-sampling (250 samp.)	4.70	2	$7.20 \cdot 10^{-2}$	19.9

The third dataset is called *Greek fret*. It contains many 90° turns and a circular movement, and it is used by COMAU as a testing path. The PCHIP interpolation

of this path is shown in Figure 5.10, while in Table 5.3 the results of the simulations are summarized.

Table 5.3: Greek fret dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]
Cubic	$9.64 \cdot 10^{-2}$	7	$6.40 \cdot 10^{-2}$	21.3
PCHIP	$9.73 \cdot 10^{-2}$	13	$1.27 \cdot 10^{-1}$	21.4
Smoothing	4.76	1	$9.00 \cdot 10^{-2}$	20.5
Re-sampling (200 samp.)	3.10	8	$6.88 \cdot 10^{-2}$	20.3

Table 5.4 reports the result of the application of the correction algorithm explained in Subsection 5.2.3.

Table 5.4: Greek fret dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]
PCHIP (no corr.)	$9.73 \cdot 10^{-2}$	13	$1.27 \cdot 10^{-1}$	21.4
PCHIP (auto corr.)	7.60	14	$6.80 \cdot 10^{-2}$	20.3
PCHIP (corr. d=2.5 mm)	2.17	13	$1.24 \cdot 10^{-1}$	20.9

In general we observe that, except for the cubic interpolation with re-sampling, the smoothing spline interpolation has the lowest number of velocity holes and the lowest total hole time length ($N_{holes} \cdot T_{holes}$). The PCHIP interpolation has, instead, the highest number of velocity holes and the highest total hole time length. This is coherent with the geometrical properties of the paths obtained, which is more rounded with the smoothing spline with respect to the cubic spline and the PCHIP spline. The interpolation with re-sampling gives different results depending on the number of samples considered, but in general it places between the cubic and the smoothing interpolations.

The maximum crossing errors of the cubic and PCHIP interpolations is less then a tenth of a millimeter for each simulation, while for the other two techniques this error ranges between 0.7 mm and 4.8 mm. If the narrow angle correction is applied, the exact crossing with the PCHIP interpolation is lost and the crossing error is increased up to 7.6 mm, depending on the values of the correction parameters.

Figure 5.7: A box with and application of sealant material on the edges.

The motion durations obtained are similar, with the smoothing and re-sampling planners being slightly faster. The differences between the fastest and the lowest trajectories are less then 5% of the total time length.

5.3 Local trajectory planning

The on-board planner must provide the results in real time, therefore it has strict time requirement. In general it is not possible to implement an on-line global planning strategy due to the high computational cost required. The COMAU planner, for example, evaluates the trajectory on two points at a time, but for continuous processes this is not enough to ensure a good speed profile.

The algorithm proposed in this chapter tries to find a compromise between the opposite necessities of computing ahead the whole trajectory for a good speed profile and of analysing as few points as possible to make the procedure fast enough to run in real time. For this purpose it has been decided to compute the trajectory on five points at a time, one more than the minimum number required for a PCHIP

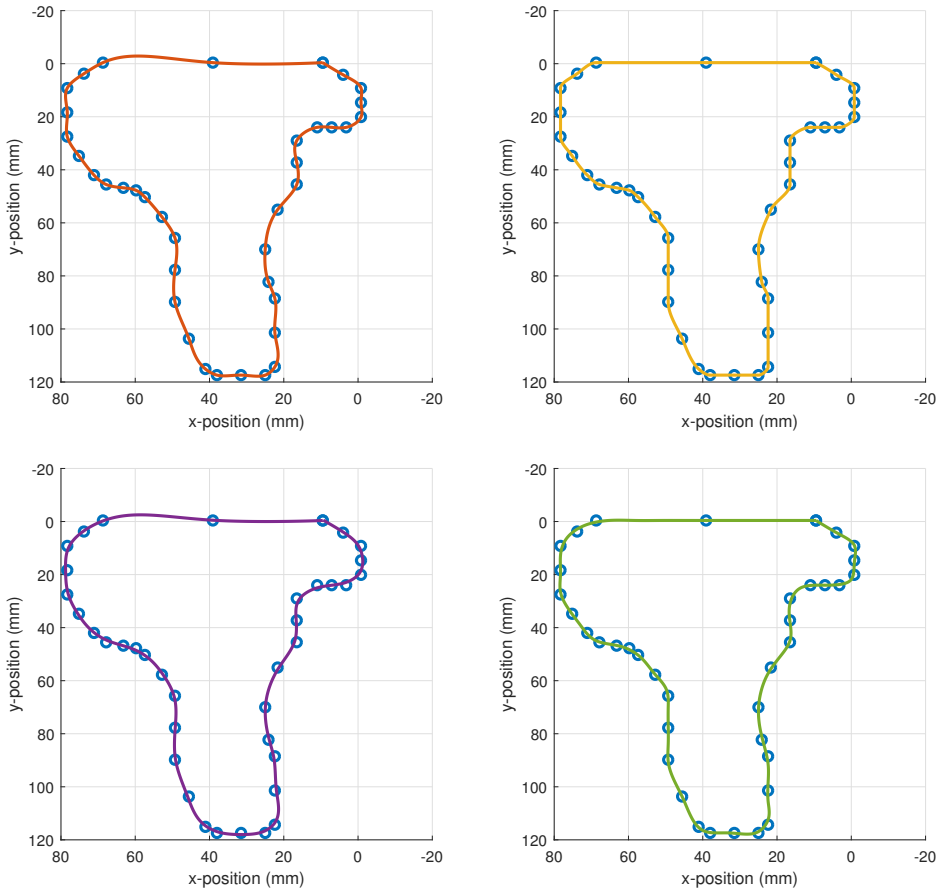

Figure 5.8: Path interpolations for the box sealing task. Interpolations: top-left cubic; top-right PCHIP; bottom-left smoothing; bottom-right cubic with re-sampling.

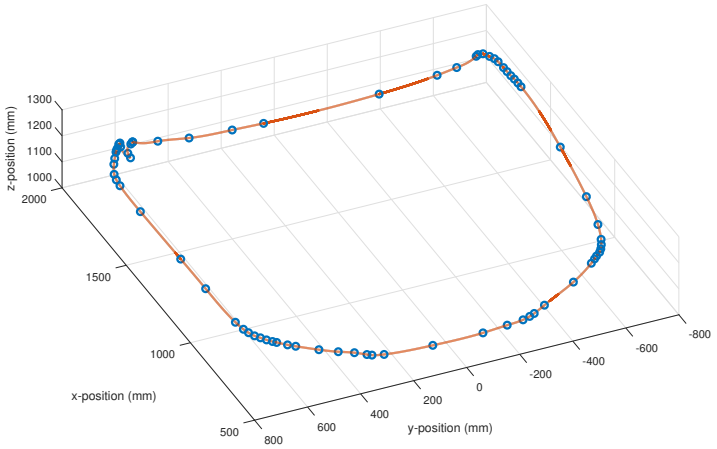

Figure 5.9: Cubic path interpolation for engine hood sealing task.

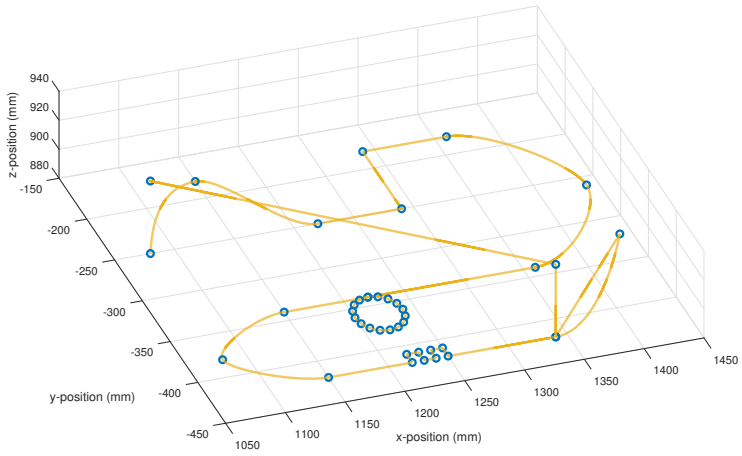

Figure 5.10: PCHIP path interpolation for testing dataset.

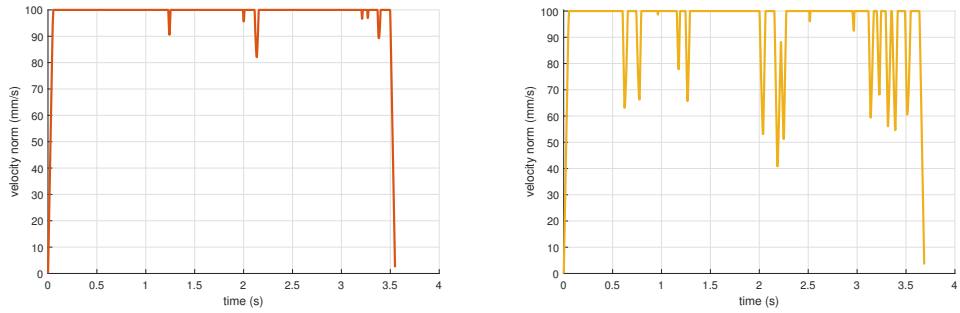

Figure 5.11: Velocity profile for the box sealing task. Interpolations: left cubic; right PCHIP.

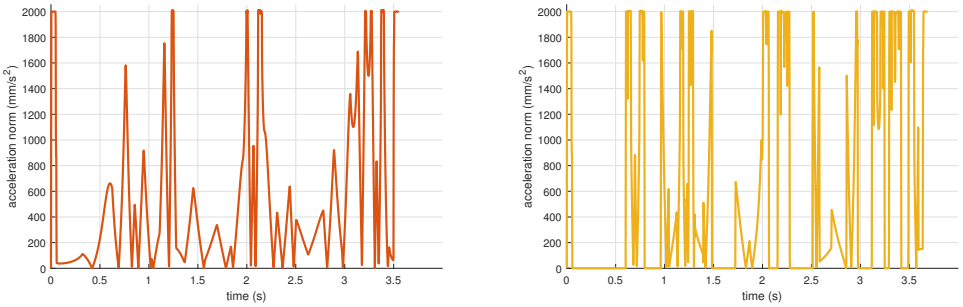

Figure 5.12: Acceleration profile for the box sealing task. Interpolations: left cubic; right PCHIP.

interpolation.

For the most part, the planning algorithm works in the same way as the one described in Section 5.2. This means that given the input data, the algorithm computes the path and the speed profile as already seen. The main difference is that multiple splines need to be computed to plan the whole trajectory and particular care is needed in the junction of consecutive splines. The next subsection is dedicated to this topic.

5.3.1 Planning and joining consecutive splines

The planner is fed with five points at a time. These points are called *visibility window* of the planner that does not have any knowledge of the rest of the path. After the planning of a spline, the window slides by one point, receiving a new point and discarding the oldest one. In this way two consecutive splines are always computed with four points in common, and the transition between them is smoother. The output of a single planning is a portion of the spline that corresponds to the interval between the first and the second point plus the final velocity of that interval. By joining every interval we cover the entire trajectory.

Another reason to use a sliding window is to have always some points of visibility ahead (at least three points with a window of five). This is essential when an emergency stop has to be planned. For safety reasons, the manipulator has to stop within the last point of the current programmed motion. In our case, if the stop signal comes at the end of the first interval and the machine is moving at target velocity, then it can not arrest before the second point, but there are three other points loaded that can be used to program a safe arrest.

At this level the interpolation can only be done with cubic splines or PCHIP splines. The smoothing splines and re-sampling interpolations described in Section 5.2 are applied globally and the re-computed points are fed to the planner as a different path.

Since the planning using cubic or PCHIP splines are slightly different, we start with the cubic interpolation and the adjustment for the PCHIP interpolation are explained later.

Trajectory planning with cubic splines

Recalling the steps explained in Section 5.2 for the global analysis, firstly we need to compute the path on the points selected by the current visibility window. While for the global interpolation the initial and final velocities are generally set to zero, here it is very important to impose the right ones to obtain smooth transitions. In particular the initial velocity is given by the previous planning and the final velocity can be computed as the unit tangent vector in the last point (see Fernet frame, Subsection 5.2.5). The tangent vector should be computed on the globally interpolated path.

The next step is the definition of the velocity profile. Also in this case the initial velocity is given by the norm of the final velocity of the previous planning. The final velocity is imposed equal to the target constant speed. After the corrections done to respect the maximum acceleration constraint, the initial and final velocities may be different from the imposed one. While a different final velocity is acceptable, to have a global continuous speed profile is important to respect the initial velocity. In this case, after having computed the corrected speed profile with the respective timing, we set the speed on the first time instant to be equal to the correct one. Then we have to check once again if the accelerations are respected and correct the speed profile accordingly.

The last phase is the re-parametrization with step h of the first interval. As output the velocity computed on the last point of the re-parametrization is also provided. It is important to observe that this last point in general does not coincide with the second point of the current window due to the discretization introduced by the re-sampling. This is compensated by replacing the original point with this one during the next planning.

Applying these tweaks, the final trajectory computed is continuous for position and velocity.

Trajectory planning with PCHIP splines

The PCHIP spline interpolation does not require the initial and final slopes. To guarantee at least the $\mathcal{C}^1$ continuity of the trajectory it is necessary to select the second interval (between the second and third point of the window) of each spline: the derivative in a point is computed using the current point, the previous one and the next one (see Section 4.2). This means that the slope in the third point of the current window is the same of the second point on the successive window, which guarantees the continuity of the first derivative.

The speed profile is computed similarly to the cubic spline case, but the initial velocity is imposed on the second point instead of the first.

5.3.2 Performance indices and MatLab simulations

For this approach we use the same four metrics used before in Subsection 5.2.7 plus two others:

- Maximum acceleration ($A_{err,max}$ [mm/s^2]): it measures the maximum acceleration actually reached. Due to numerical approximations, some junctions between consecutive intervals may have acceleration spikes.

- Maximum reference error (L_{ref} [mm]): it measures the maximum distance between the computed path and the reference path (the global computed one).

The same testing procedure illustrated in Subsection 5.2.7 is followed: for each dataset the target speed is set to 100 mm/s and the maximum acceleration to 2000 mm/s^2. Tables 5.5, 5.6 and 5.7 show the results of the simulations for the box dataset, the engine hood dataset and the greek fret dataset, respectively.

Table 5.5: Box dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]	$A_{err,max}$ [mm/s^2]	L_{ref} [mm]
Cubic	$9.91 \cdot 10^{-2}$	6	$4.13 \cdot 10^{-2}$	3.57	$2.19 \cdot 10^{3}$	$2.43 \cdot 10^{-1}$
PCHIP	$9.13 \cdot 10^{-2}$	14	$4.99 \cdot 10^{-2}$	3.68	$2.85 \cdot 10^{3}$	$1.00 \cdot 10^{-1}$
Smoothing	$9.72 \cdot 10^{-1}$	2	$2.00 \cdot 10^{-2}$	3.49	$2.00 \cdot 10^{3}$	$3.66 \cdot 10^{-1}$
Re-sampling (50 samp.)	$7.07 \cdot 10^{-1}$	6	$3.53 \cdot 10^{-2}$	3.54	$2.14 \cdot 10^{3}$	$2.53 \cdot 10^{-1}$

Table 5.6: Engine hood dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]	$A_{err,max}$ [mm/s^2]	L_{ref} [mm]
Cubic	$9.48 \cdot 10^{-2}$	3	$1.29 \cdot 10^{-1}$	20.2	$2.11 \cdot 10^{3}$	3.44
PCHIP	$9.97 \cdot 10^{-2}$	5	$8.68 \cdot 10^{-2}$	20.1	$2.64 \cdot 10^{3}$	$9.78 \cdot 10^{-2}$
Smoothing	1.21	2	$1.12 \cdot 10^{-1}$	20.0	$2.12 \cdot 10^{3}$	3.03
Re-sampling (150 samp.)	4.68	2	$8.60 \cdot 10^{-2}$	1.99	$2.03 \cdot 10^{3}$	1.87

Overall the performances of the local trajectory planners are not significantly worse than the global counterpart. For example, the number of velocity holes are in the worst case increased by 2, but for the majority of the simulations the number has not increased.

About the differences between the global and local interpolated path, we can observe that the PCHIP is the most accurate, with an error of about 0.1 mm in our simulations. The cubic and smoothing trajectory planners are the least accurate,

Table 5.7: Greek fret dataset, $V_{target} = 100$ mm/s, $A_{max} = 2000$ mm/s^2

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]	$A_{err,max}$ [mm/s^2]	L_{ref} [mm]
Cubic	$9.58 \cdot 10^{-2}$	7	$5.51 \cdot 10^{-2}$	21.7	$2.08 \cdot 10^{3}$	$1.02 \cdot 10^{1}$
PCHIP (no corr.)	$9.19 \cdot 10^{-2}$	13	$1.18 \cdot 10^{-1}$	21.3	$4.22 \cdot 10^{3}$	$1.09 \cdot 10^{-1}$
PCHIP (corr. d=2.5 mm)	2.17	13	$1.23 \cdot 10^{-1}$	20.9	$2.33 \cdot 10^{3}$	$1.04 \cdot 10^{-1}$
Smoothing	4.78	3	$4.33 \cdot 10^{-2}$	20.8	$2.07 \cdot 10^{3}$	$1.00 \cdot 10^{1}$
Re-sampling (200 samp.)	2.88	9	$7.40 \cdot 10^{-2}$	20.3	$2.50 \cdot 10^{3}$	1.32

with a maximum error of 10 mm in the Greek fret path. Figure 5.13 shows the differences between the global and local path interpolation for this simulation. We also see that this error is very dependent on the geometry of the path, as it is only 0.24 mm on the box path. Is is also significantly lower for the re-sampling interpolator, which has a maximum error of 1.8 mm.

Analysing the norm of the accelerations in Figure 5.14, we observe that some acceleration spikes are introduced. In each simulation the maximum spike is less than 125% of the maximum acceleration imposed by the user except for the PCHIP planner, which has a spike of 210% of the maximum acceleration on the Greek fret simulation. In this case the algorithm for smoothing the acute angles gives significantly better results, since it lowers that spike by 45%, which means that the maximum acceleration reached in this case is 115% of the constraint.

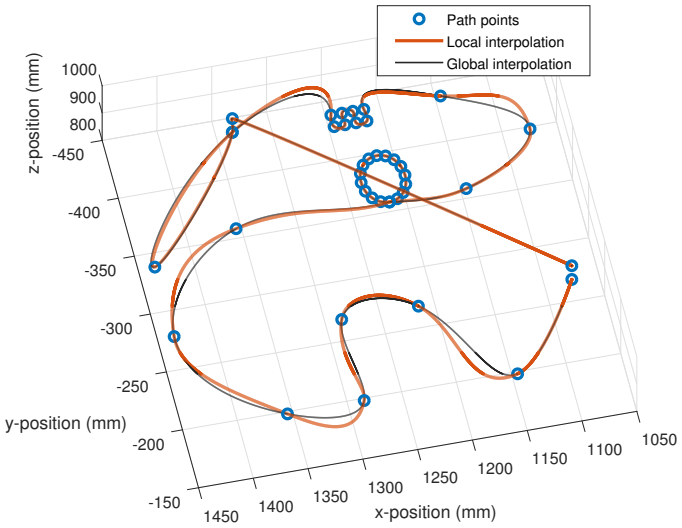

Figure 5.13: Comparison between global and local cubic spline interpolation on the Greek fret path.

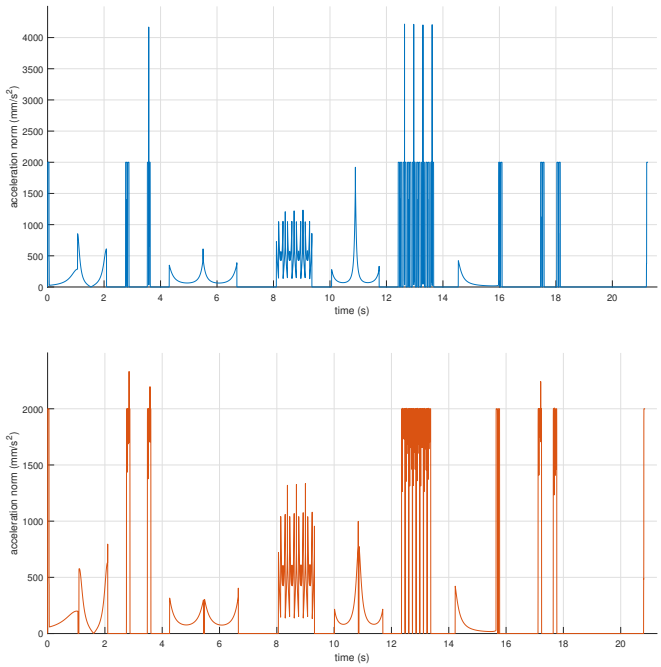

Figure 5.14: Accelerations profile for the Greek fret path. Interpolations: top PCHIP with no path correction; bottom PCHIP with correction.

5.3.3 Trajectories tested on robot

In this section the trajectory tested on the robot arm are collected. Here only the MatLab results are shown, while the complete testing set-up is presented in Chapter 6. All the trajectories tested use the local planning technique described in Section 5.3.

The trajectory tested for the box sealing task is programmed with PCHIP interpolation, target constant speed of 36 mm/s and maximum acceleration of 600 mm/s^2. The path points used are sampled from the target path of the trajectory programmed with the standard COMAU planner. The path and the velocity profile obtained are shown respectively in Figures 5.15 and 5.16.

For the engine hood sealing task, three trajectories are tested. The first one is programmed with cubic interpolation, target constant speed of 400 mm/s and maximum acceleration of 3000 mm/s^2. Its results are shown in Figures 5.17 and 5.18. Another trajectory is planned with cubic interpolation, target constant speed of 100 mm/s and maximum acceleration of 2000 mm/s^2. The last one is programmed with PCHIP interpolation, target constant speed of 100 mm/s and maximum acceleration of 2000 mm/s^2. The paths and the velocities of these last two trajectories are shown in Figures 5.19 and 5.20.

The Greek fret trajectory used for the testing on robot is programmed with the PCHIP interpolation, 400 mm/s target speed and 3000 mm/s^2 maximum acceleration. The narrow angles smoothing is also applied with d = 2.5 mm. Figure 5.21 and Figure 5.22 show the obtained results.

Table 5.8 summarizes the performance indices computed on the trajectories listed above. The path points and the target velocity of the trajectory here exposed, except for the hood trajectories with velocity target of 100 mm/s, are chosen in accordance of processes that are already used and planned with the COMAU standard planner. Since the acceleration limits are not specified, we can freely choose suitable values. For the Box trajectory, since the target velocity is low, we can impose a low acceleration that should favour the tracking capabilities of the robot. With 600 mm/s^2, we obtain a good speed profile with few holes, as we can see in Figure 5.16. The target velocities of the other two trajectories are, instead, higher. For both cases, we choose an acceleration of 3000 mm/s^2 that is high enough to avoid too many speed drops, but still within the limits of the machine. An exact measure of the maximum acceleration does not exist, because it depends on many factor, for example on the joints involved on the specific motion, but values under 4000-5000 mm/s^2 are generally considered acceptable for the robot that we use for our testing.

Table 5.8: Summary of the trajectories tested on the robot arm.

Indices	L_{cross} [mm]	N_{holes}	T_{holes} [s]	T_{motion} [s]	$A_{err,max}$ [mm/s^2]	L_{ref} [mm]
Box (PCHIP, 36 mm/s, 600 mm/s^2)	$3.60 \cdot 10^{-2}$	11	$6.82 \cdot 10^{-2}$	29.6	$8.10 \cdot 10^{2}$	$1.14 \cdot 10^{-1}$
Hood (cubic, 400 mm/s, 3000 mm/s^2)	$3.89 \cdot 10^{-1}$	7	$1.88 \cdot 10^{-1}$	5.90	$4.40 \cdot 10^{3}$	3.45
Hood (cubic, 100 mm/s, 2000 mm/s^2)	$9.48 \cdot 10^{-2}$	3	$1.29 \cdot 10^{-1}$	20.2	$2.11 \cdot 10^{3}$	3.44
Hood (PCHIP, 100 mm/s, 2000 mm/s^2)	$9.97 \cdot 10^{-2}$	5	$8.68 \cdot 10^{-2}$	20.1	$2.64 \cdot 10^{3}$	$9.78 \cdot 10^{-2}$
Greek (PCHIP, 400 mm/s, 3000 mm/s^2)	2.17	15	$2.95 \cdot 10^{-1}$	7.34	$5.06 \cdot 10^{3}$	1.60

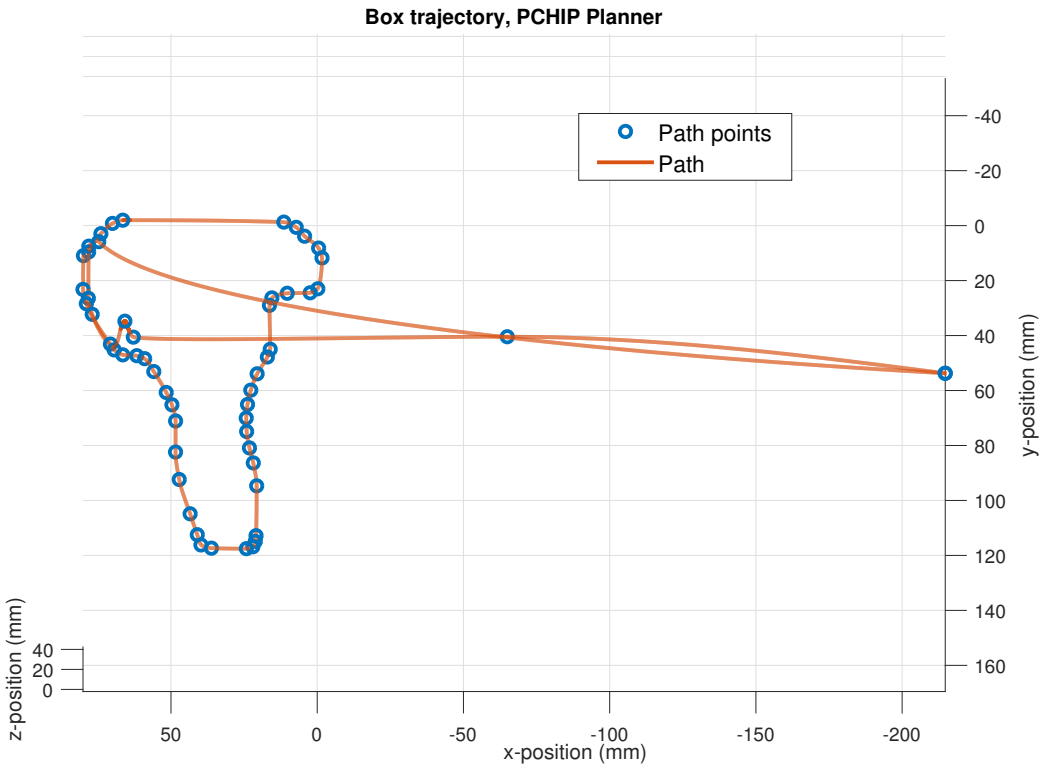

Figure 5.15: PCHIP path interpolation for the box sealing task.

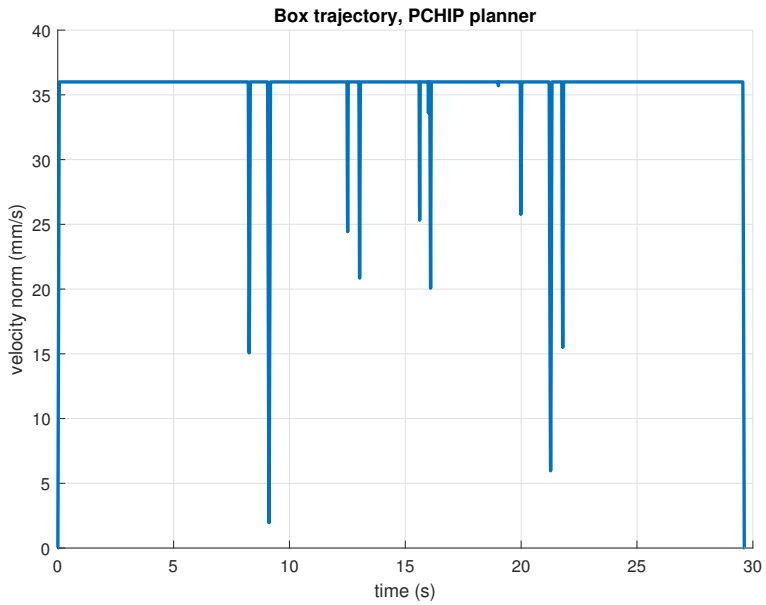

Figure 5.16: Velocity profile for the box sealing task obtained with PCHIP interpolation with 400 mm/s target speed.

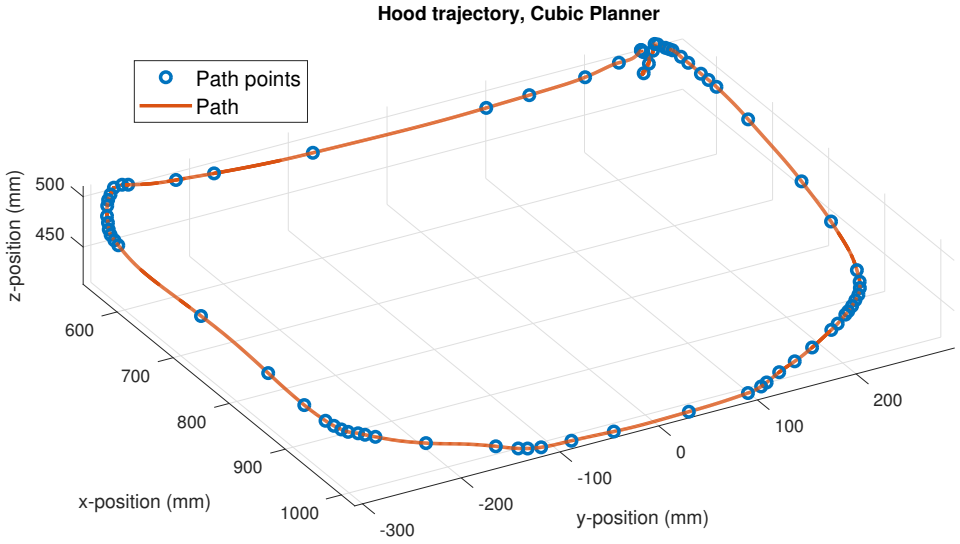

Figure 5.17: Cubic path interpolation for the engine hood sealing task.

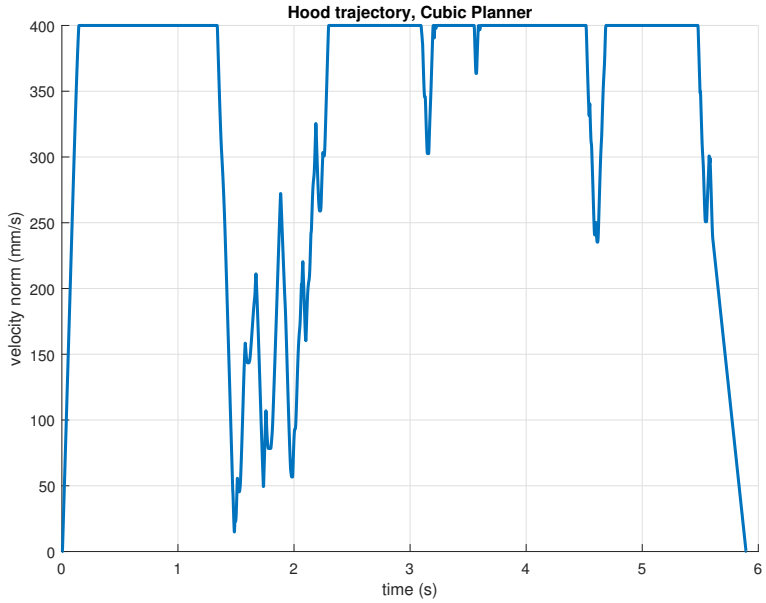

Figure 5.18: Velocity profile for the engine hood sealing task obtained with cubic interpolation.

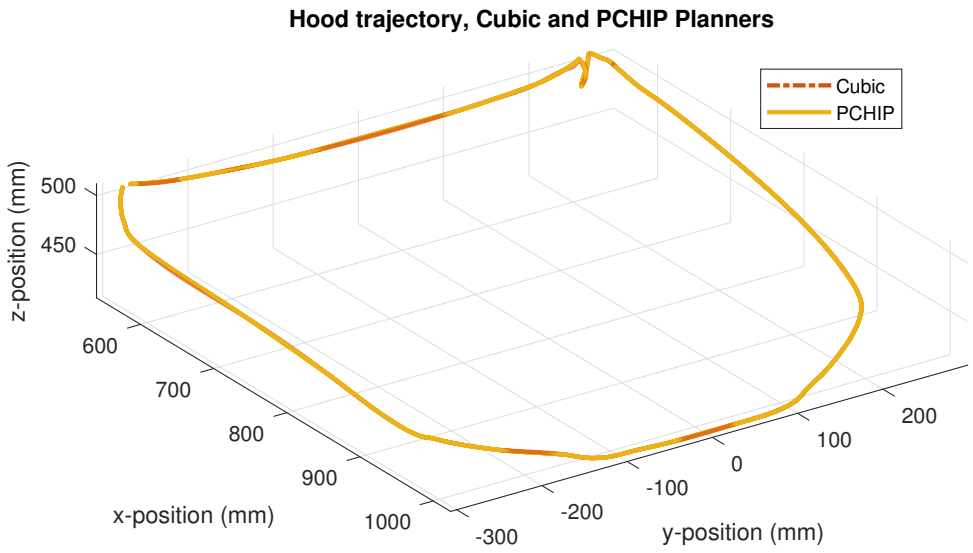

Figure 5.19: Cubic and PCHIP path interpolations for the hood sealing task.

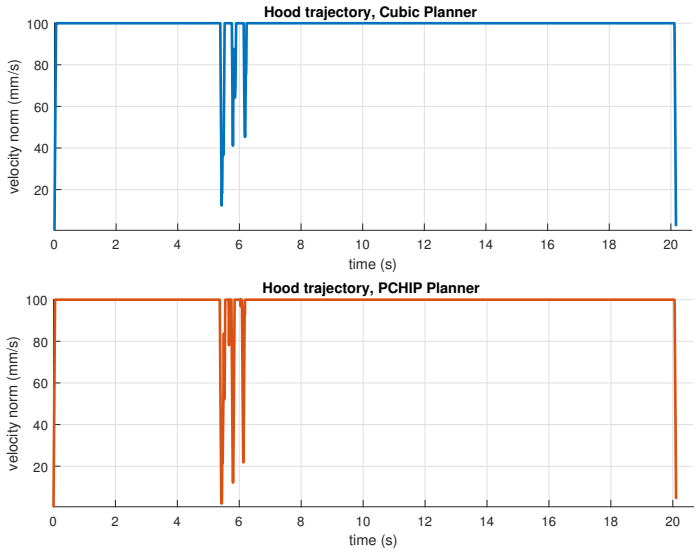

Figure 5.20: Velocity profiles for the hood sealing task obtained with cubic and PCHIP interpolation with 100 mm/s target speed.

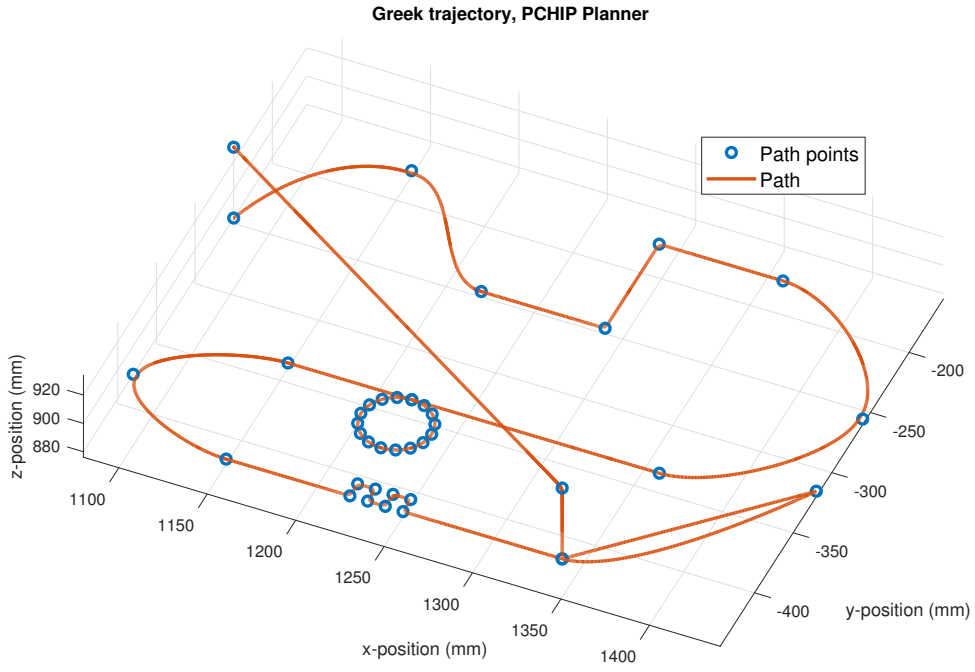

Figure 5.21: PCHIP path interpolation for the Greek fret trajectory.

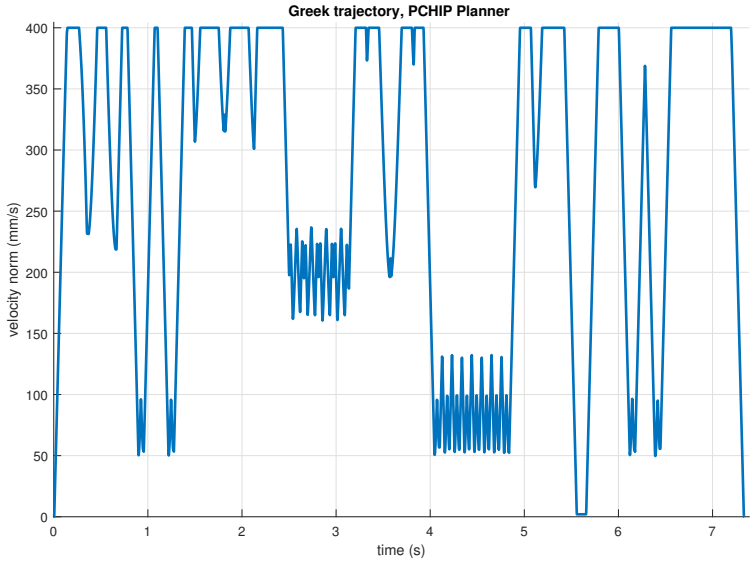

Figure 5.22: Velocity profile for the Greek fret trajectory obtained with PCHIP interpolation.

5.4 Graphical user interface

A MatLab application have been developed in order to simplify the input selection for the trajectory planning. Referencing to Figure 5.23, the GUI is composed of the following elements:

1. Input data: here the target velocity, the maximum acceleration and the sampling time can be selected. If the correction check-box is selected, than the velocity profile is programmed in order to respect the acceleration constraint, otherwise a trapezoidal speed profile is planned.
2. Select dataset: the points that define the desired path can be chosen here.
3. Path preview: a preview of the selected path is shown.
4. Sharp angle correction: this feature is enabled only for the PCHIP planner. Here different behaviours of the sharp angle correction algorithm can be selected.
5. Resampling: here the number of samples or the approximative step distance can be selected for the re-sampling planner.
6. Select planners: one or more planning techniques can be selected.
7. Preview: it shows only the path obtained with the selected interpolation technique on the desired dataset.
8. Analysis: it performs the global trajectory planning on the selected inputs plus the analysis of the path radii.
9. Planning: it performs the local trajectory planing.

The output trajectory is saved in a different folder in *MATLAB Data* format.

5.5 Cubic spline with optimized time intervals

Before developing the approach illustrated in Section 5.3, another method has been tested out.

Referring to Section 4.1, to interpolate a cubic spline we need a set of points q_k and their timing t_k. Since in continuous processes applications the user is interested in the velocity profile, we can use t_k as a set of optimization variables. The idea is to search if it is possible to obtain a sufficiently constant velocity profile by designing a proper optimization problem.

Also this algorithm adopts the sliding window technique with the selection of the first interval for the reasons explained in Section 5.3.

In the next section the details of the optimization problem are given.

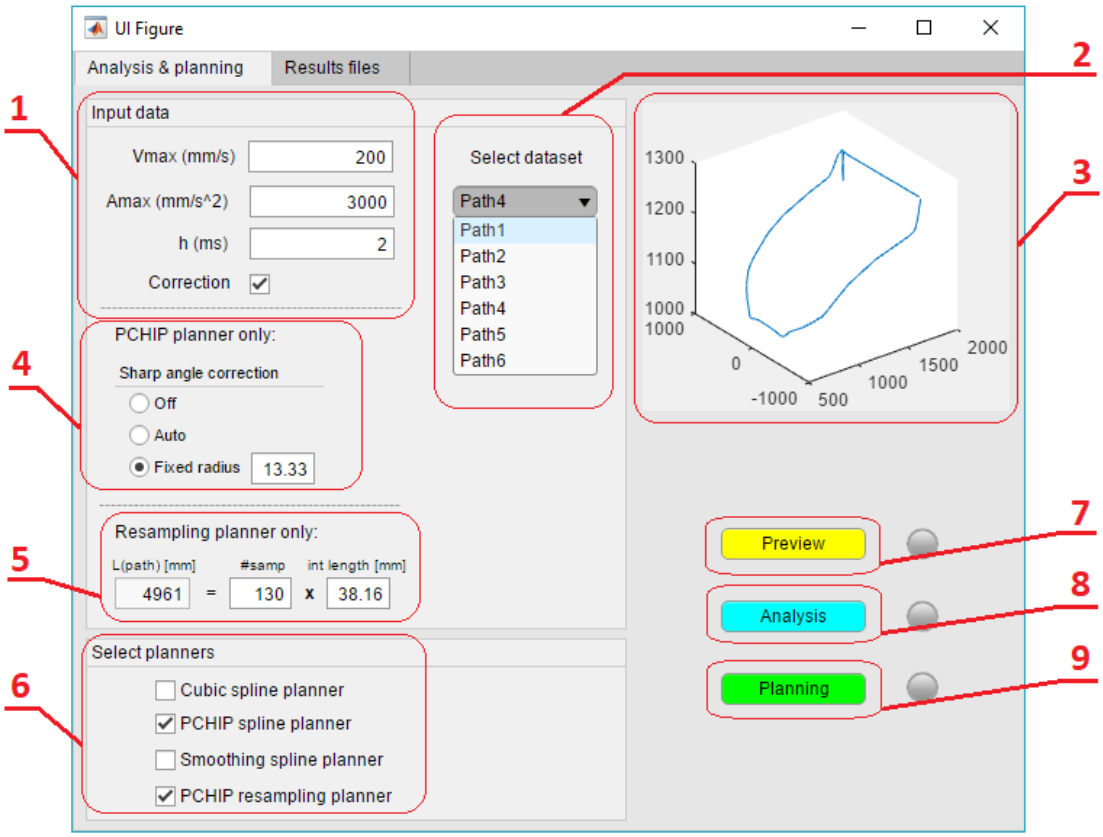

Figure 5.23: Graphical user interface for trajectory planning.

5.5.1 Optimization problem setup

Our goal is to obtain a constant velocity profile. This means that we have to minimize the deviations of the speed from a certain constant value. In a spline we have to check the velocities at the end of each interval and in the intermediate points. In particular for the intermediate points we are interested in the stationary ones (maxima and minima). This is done by finding the zero of the derivative of the speed function. First of all we compute the interpolation as explained in Section 4.1. We obtain the values of the velocities $v_{k,x}(t), v_{k,y}(t), v_{k,z}(t)$ for each interval (t_k, t_{k+1}), with $k = 1, ..., N-1$ and N equal to the number of points interpolated. The values $T = [t_1, ..., t_N]$ are the optimization variables.

In a three-dimensional spline the speed is:

$$v_k(t) = \sqrt{v_{k,x}(t)^2 + v_{k,y}(t)^2 + v_{k,z}(t)^2} \tag{5.24}$$

To obtain the stationary points more easily, we can derive the square of the speed and find the zeroes of this function:

$$(v_k(t)^2)' = 2v_{k,x}(t) + 2v_{k,y}(t) + 2v_{k,z}(t) \tag{5.25}$$

We denote the times in which the points are stationary as t_k^*. The zeroes outside the interval (t_k, t_{k+1}) are discarded.

The cost function of the optimization problem is the mean square error of the deviation of the velocities from the target speed v_{target}:

$$f(T) = \frac{1}{2N-1}\sqrt{\sum_{k=1}^{N}(v_k(t_k) - v_{target})^2 + \sum_{k=1}^{N-1}(v_k(t_k^*) - v_{target})^2} \tag{5.26}$$

The constraints are imposed on the maximum velocity and acceleration norms. Since the acceleration of a spline function is linear, it reaches its maximum at the beginning or at the end of each interval, so there is no need to check the intermediate values:

$$\begin{aligned} &v_k(t_k) < V_{max} \quad k = 1, ..., N \\ &v_k(t_k^*) < V_{max} \quad k = 1, ..., N-1 \\ &a_k(t_k) < A_{max} \quad k = 1, ..., N \end{aligned} \tag{5.27}$$

The cost function and the constraints are non linear, therefore an appropriate tool is needed to solve this optimization problem. In out case we used the MatLab nonlinear optimization toolbox.

5.5.2 MatLab simulation and considerations

The algorithm is tested on the trajectory for the box cosmetic sealing process. A first test is carried out without any pre-processing of the input data, while for a second test the path has been re-sampled with constant step distance. The setup for both tests is the same:

- Target speed: 50 mm/s
- Maximum speed: 55 mm/s
- Maximum acceleration: 2000 mm/s^2
- Initial and final velocity: 50 mm/s

As we can see from the velocity profiles obtained in Figure 5.24, the results are not satisfying enough to justify further developing. There are also many difficulties intrinsic in a non linear optimization problem: first of all, the solvers can not guarantee that the solution found is an absolute minimum, thus a slight change in the inputs can give very different results; then a solution may not be even found if the problem converges to an infeasible solution, and this is not acceptable for an on-line application; finally the solution of a non linear optimization problem is very computational demanding. For these reasons, it has been decided to discard this approach in favour of the re-parametrization approach.

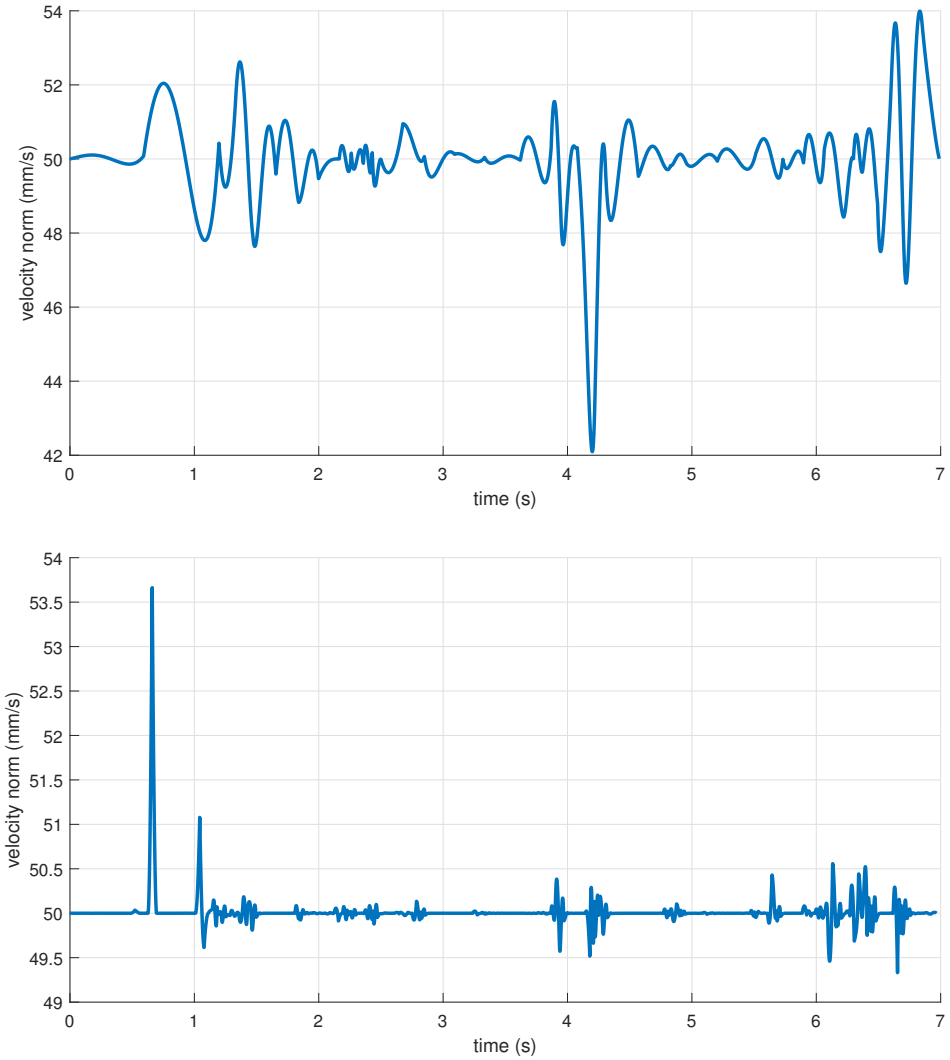

Figure 5.24: Velocity profile obtained with time interval optimization. Interpolations: top cubic; bottom cubic with re-sampling.

Chapter 6

Experimental tests and results

The testing activities were primarily carried out in the RoboLAB. Born as a joint collaboration between Politecnico di Torino and COMAU S.p.A., this laboratory, directly inside the Grugliasco plant, houses several robots that are used by students and researchers of Politecnico or by COMAU's R&D employees.

6.1 Configuration of robot cell for testing

For our tests we used the robot *Racer7-1.4*. It is a general purpose anthropomorphic robot with 6 axis and 7 Kg of payload capability. Some typical applications in which it can be used are: assembly, cosmetic sealing, handling and packaging, polishing and deburring, measurement testing and machine tending.

The control unit of the robot is the *C5 Compact*. It uses the industrial PC APC820 with a CPU Intel Core2 Duo @ 1.50 GHz. Each control unit can be connected up to two robots. It also contains the drive modules that generate the current for the motors of the robot. Figure 6.1 shows both the robot and the control unit used for testing.

The control unit can be interfaced either using a *teach pendant* TP-5, shown in Figure 6.2, or with a specific software installed on a computer called *WinC5G*, shown in Figure 6.3. A teach pendant is a portable console used to program and control the robot. It has a keyboard that allows to move the robot both in Cartesian and joint space in an intuitive way, plus a touch screen used to move between the offered functionalities. All the functionalities are also available using the WinC5G for PC, which has a command-line interface. A virtual TP can also be installed to have an exact replica of a real teach pendant. The main advantage of using WinC5G is to use its editor for programming the motion of the robot in the COMAU proprietary language PDL2. This can also be done with a TP, but it is not as

comfortable as using a PC.

An APC not connected to a robot has also been used during the testing phase. This module (later referred as *rack*) contains only the planning and control logic, without the drivers. It is interfaced with a PC and its status can be visualized with Visual3D, that shows a 3D simulation of the robot. This set-up is very useful to test a programmed motion in a safe and controlled environment before loading it on the real robot.

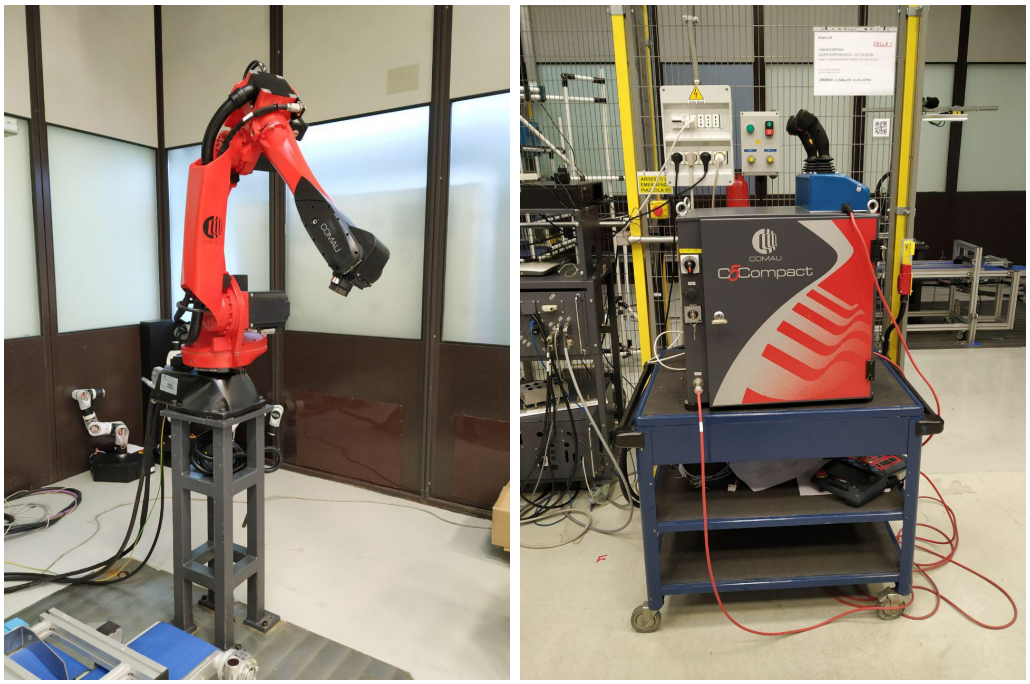

Figure 6.1: The robot Racer7-1.4 and the control unit C5 Compact used for testing.

6.2 PDL2 program, MOVE REPLAY and moni.log

There are two ways to move the robot implemented in the COMAU architecture: manually by using a teach pendant or by running a PDL2 program.

A PDL2 program contains a set of positions (Cartesian or joint) and motion primitives that describes how to move between the input points. When a PDL2 program is started, the on board planner interprets the commands and plans the desired trajectory, checking in real time the status of the robot to avoid dangerous situations, like singularity points or actuators limits.

One motion primitive, called *MOVE REPLY*, works in a different way with

Figure 6.2: The COMAU teach pendant TP-5

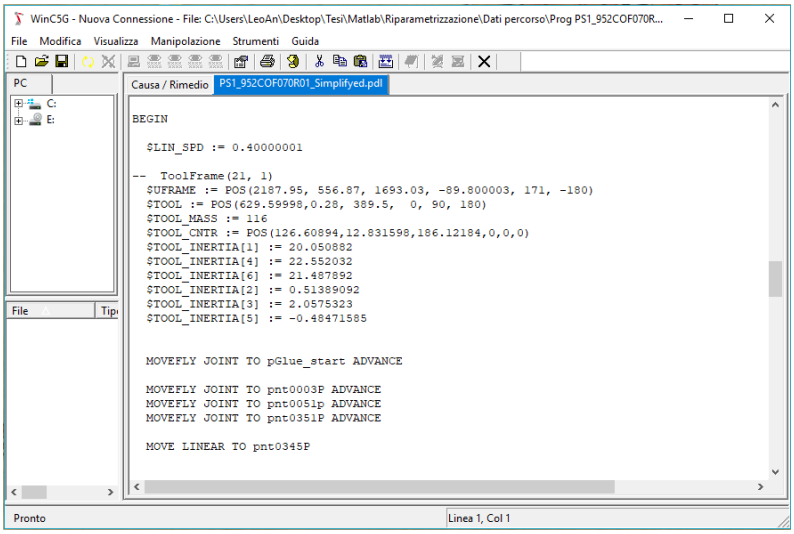

Figure 6.3: The interface of WinC5G.

respect to the other ones. This function is used to read a *moniSLJ.log* file that contains a sequence of points sampled at 2 milliseconds and reproduce it. During the execution of this motion the internal planner is bypassed and no checks are performed. This operation is generally safe because the *moni SLJ* file is produced by recording a motion of interest of the robot. The MOVE REPLY is commonly used to reproduce a manually programmed movement.

This specific feature was exploited to bypass the on board planner and test

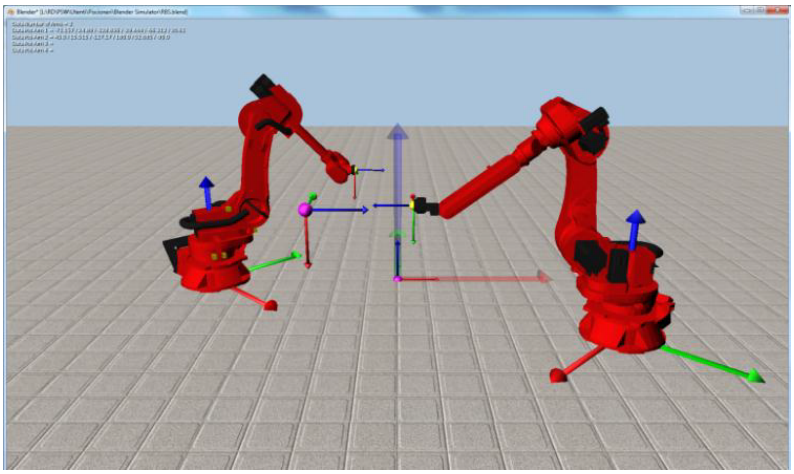

Figure 6.4: Two robots simulated in Visual3D

our planner. In particular, instead of recording a moni SLJ file, we can create a custom one inserting the points computed in MatLab and executing it with a MOVE REPLY. The details on how to build a custom moni SLJ are explained in Subsection 6.2.1.

A customized moni SLJ file can be executed with a standard MOVE REPLY. Since the on board planner does not perform any error checking on the trajectory in execution, it is very important to perform a preliminary simulation on the rack. If the simulation goes well and no critical situation emerges from that, then the trajectory can be tested on the real robot.

To verify the results, the measurements of the sensors can be extracted during the execution of the motion. This is done by registering a file called *moni.log*. It is different from a moni SLJ file, since the latter contains only the information on position and velocity sampled at 2 millisecond, while the former contains also many other measurements like accelerations, trajectory error and current drained by each actuator. The moni.log file can be read end exported on MatLab to visualize the results.

Figure 6.5 summarises the execution flow from the trajectory planning in MatLab to the recording of a moni.log file.

6.2.1 Building a custom moni SLJ file

A moni SLJ file is a binary file that contains a header and a sequence of points sampled at 2 milliseconds. The velocity information is embedded inside the sequence and can be extracted performing its discrete derivative.

The header of the file contains the information on the process at the time of recording. Some relevant information are:

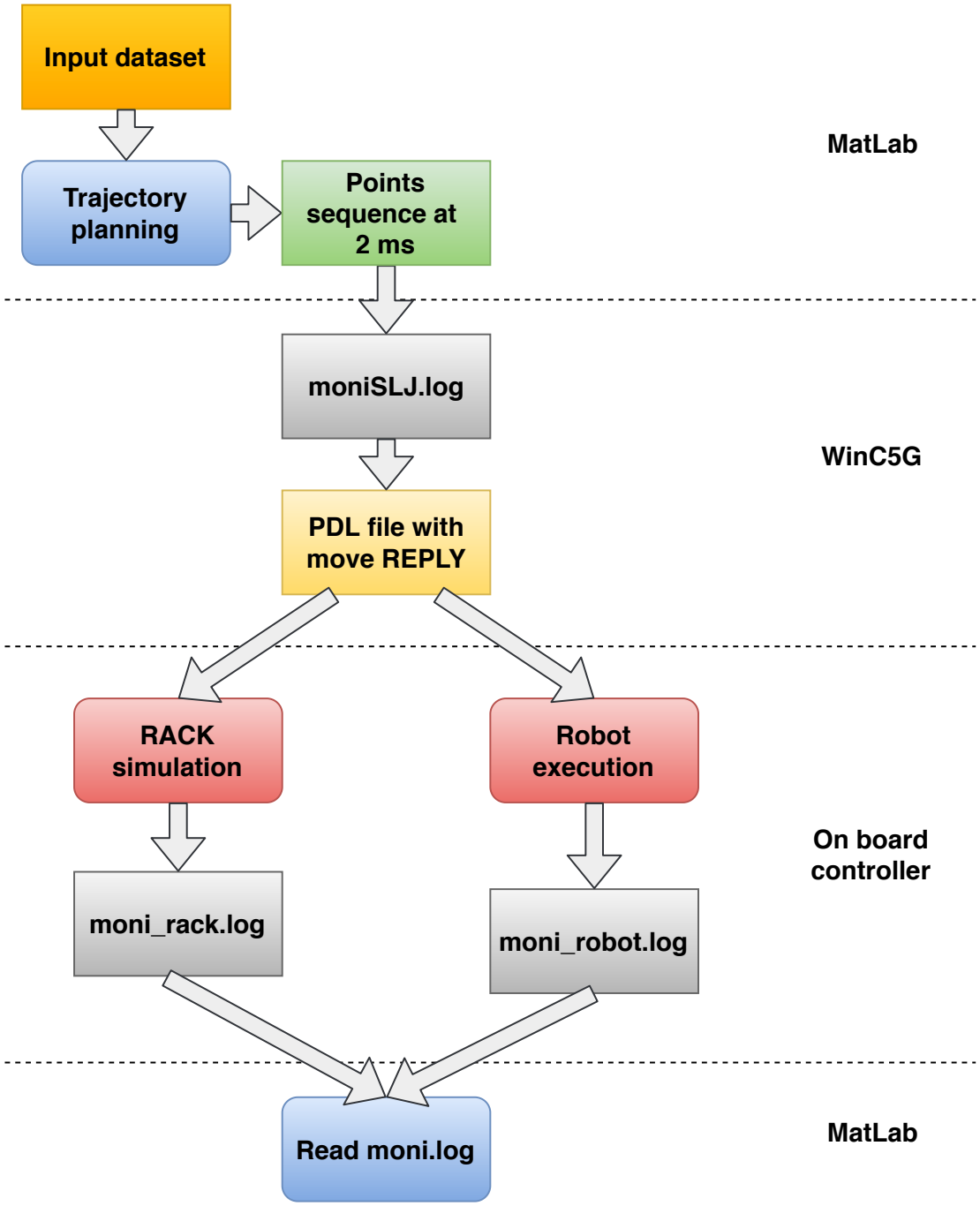

Figure 6.5: Block diagram of the operations needed to perform the simulation on the robot.

- Model of the robot
- Positions of the frames of reference (user frame and tool frame)
- Initial and final positions
- Number of records
- Checksum computed on the recorded points

When executing a moni SLJ with a MOVE REPLY command, a coherence check on the header is performed. If any part of the file is not consistent, then the MOVE is not executed for safety reasons. For example, if the tool frame at the time of execution is different than the one set at the time of recording, it is not safe to execute the motion again, since the robot would move in an unexpected way.

In order to create a moni SLJ that can be run with a MOVE REPLY, a file with a coherent header must be written. A MatLab script has been developed by COMAU in order to do that. Given the points sequence and the values of the reference frames, two files are given as outputs: a moni SLJ file with a coherent header and a PDL2 file ready to execute the MOVE REPLY.

Figure 6.6 compares the steps needed to generate and execute a moni SLJ file with the standard and the custom procedures.

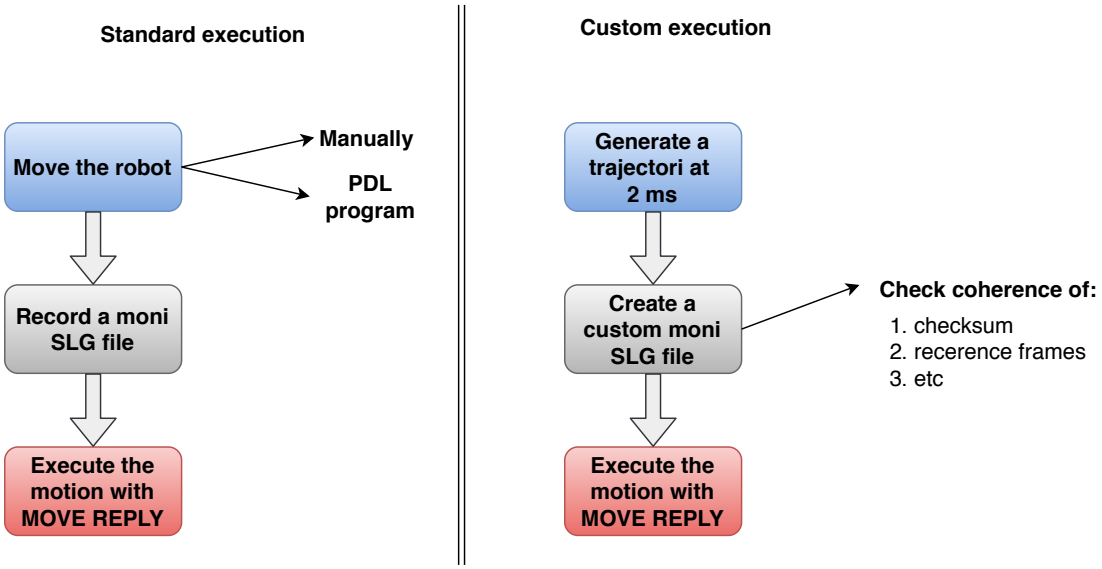

Figure 6.6: Standard and custom generation and execution of a moni SLJ file.

6.3 Test trajectories

In this section the results of the tests performed with the set-up described in Section 6.1 are shown. The trajectories used are the same used for the MatLab simulation, which are described in Subsection 5.3.3. In particular the local trajectory planner is used.

Each tested trajectory is compared with the results of the planning performed with the standard COMAU planner.

6.3.1 Box trajectory

The box trajectory is required for a sealant deposit process. The requirements of this task are very strict and the most important ones are:

- Order of magnitude of the path tracking error around a tenth of millimeter
- Constant Cartesian speed norm. The value of the target speed is not relevant, since the extruder can be regulated.

The total execution time is not critical for this process, so the execution is carried out at low speed, namely 36 mm/s.

The PDL program of the user contains a sequence of linear and circular motion with the *fly* option enabled. This option prevents the planner from stopping at the end of each instruction; instead, it tries to execute two consecutive commands with constant speed. With this kind of motion, the instructed points are not exactly reached, so they have to be chosen outside the desired trajectory by trial and error. Figure 6.7 shows how the points were chosen to obtain the desired path.

In order to have comparable results, our planner has to work on the same path imposed by the user in his process. To obtain this, the PCHIP interpolation is chosen, then some samples on the user target path are selected. With the PCHIP interpolation, just two or three points are needed to properly describe a straight segment, while a curve may need three or more points depending on its radius. In Figure 6.8 the user target path and the PCHIP interpolated target path are shown. The approach and disengage motions are not exactly the same, but this is not relevant for our purposes, since we are interested just on the portion that correspond to the piece manufacturing.

After having defined the path points and the interpolation method, the planning is performed with target constant speed of 36 mm/s and maximum acceleration of 600 mm/s^2.

Once that the data about the motion of the robot are collected with a standard moni.log file, we can plot in MatLab the results. Figures 6.9, 6.10 and 6.11 show, respectively, the Cartesian positions, the velocity norm and the tracking error for both the user planning and the PCHIP planning. The approach and disengage

intervals are excluded from the plots. The interval where the piece manufacturing occurs is delimited by two vertical black lines.

In Figure 6.10 we can see that the velocity has a certain amount of noise. Computing the standard deviation only on the interval of interest we obtain $\sigma_{user} = 3.390$ mm/s for the PDL execution, and $\sigma_{pchip} = 1.829$ mm/s with our trajectory. Estimating this index on other constant speed tasks, we found that it spans from 0.783 mm/s to 1.216 mm/s independently from the value of velocity reached, so we can assume that the mechanical system introduce a random noise with a standard deviation around 1 mm/s.

In the case of the PCHIP planner, some velocity holes are introduced by the planner itself depending on the geometry of the path. This can be seen in Figure 6.13 and in Figure 6.14, where the colours represent the velocity at each point. The robot arm operates also as a low-pass filter, so the speed profile planned is not exactly followed.

Now we analyse the path tracking error. Referencing to Figure 6.11, no meaningful differences are found between the two approaches, as a matter of fact the mean values of the errors are: $\mu_{user} = 1.369$ mm and $\mu_{pchip} = 1.364$ mm. This values are just for reference, because the executed path has a slight delay with respect to the reference path. This delay is introduced by the system during the computation of the trajectory and during its actuation. If we look at a zoomed portion of the path in Figure 6.12, we see that the actual error is less then 1 mm. The behaviour of the two planners in this case is similar, with areas where the COMAU planner is better, like in Section 1 of Figure 6.12, and other where the PCHIP planner is more precise, like in Section 2 of Figure 6.12.

In conclusion with this test we can say that we obtained a reduction of almost 50% on the standard deviation of the velocity profile, while no improvement is observed on the path tracking error. An important aspect to consider is the user-friendliness of the two approaches: both have achieved comparable results, but the PCHIP planner do not require any knowledge of the system, and the input points are directly sampled on the target path. The COMAU approach instead has required a lot of effort by the user to obtain the described results. In particular many attempts were done to select the correct path points and a proper target constant speed.

6.3.2 Engine hood trajectory

Also the engine hood trajectory is used for a sealant deposit process. In this case the workpieces are larger than the boxes considered in Subsection 6.3.1, and the requested velocity is 400 mm/s. Unfortunately the PDL program available is intended to use on another robot with a larger reach with respect to the one used for testing.

In order to execute the program on the robot Racer7-1.4, the path points have

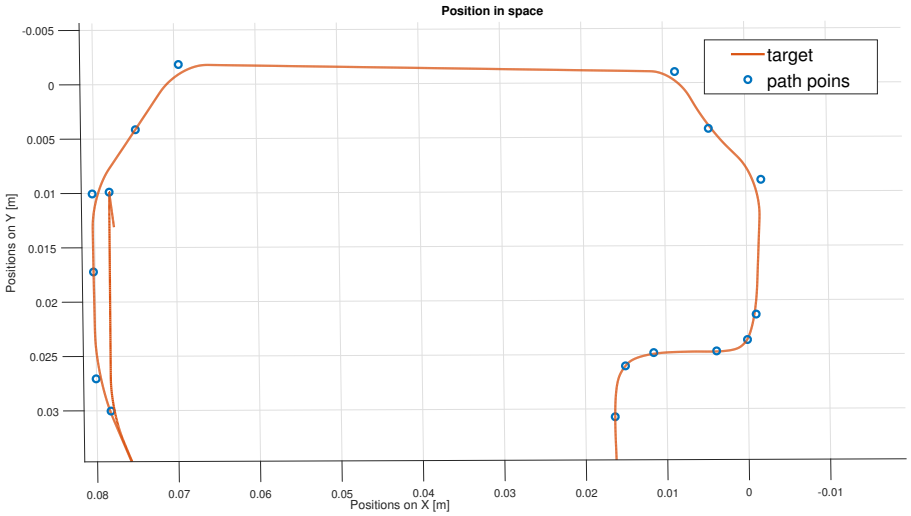

Figure 6.7: Box trajectory. Path obtained with *movefly* motions through the indicated points.

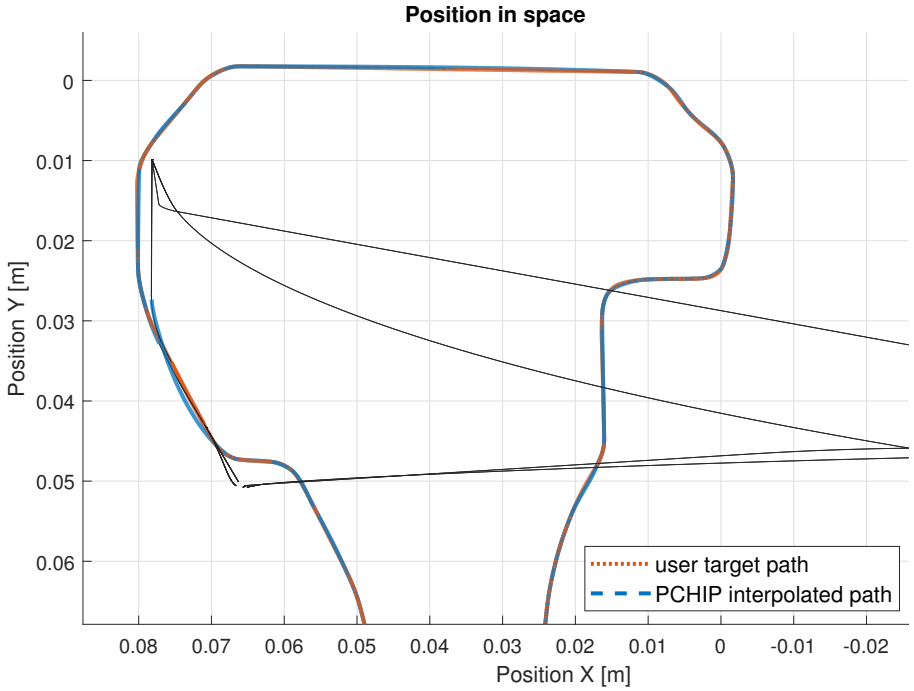

Figure 6.8: Box trajectory. Comparison between the user target path and the PCHIP interpolated path. The black lines represent the approach and disengage motions.

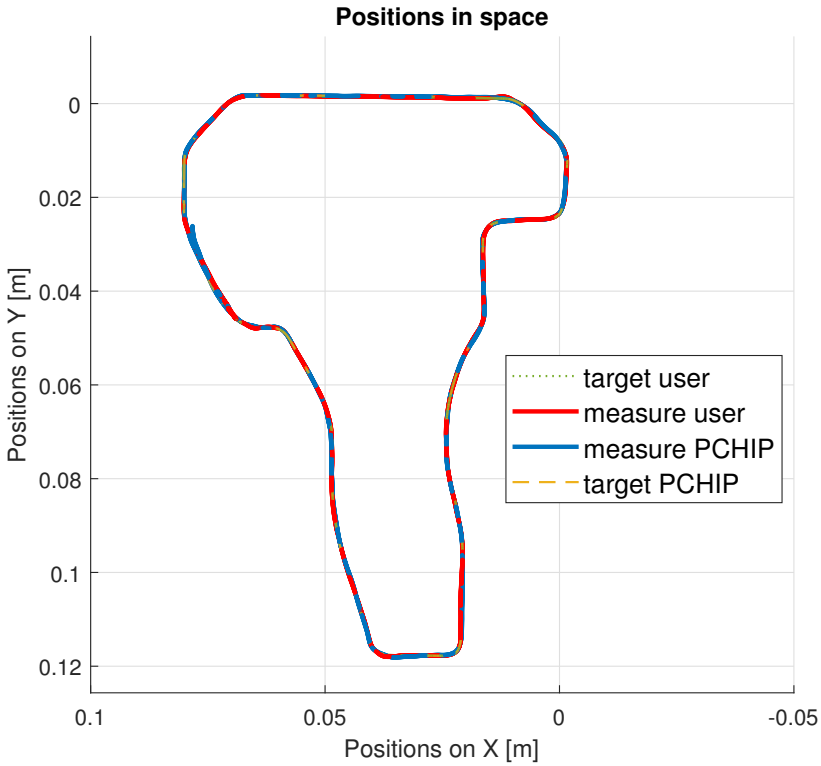

Figure 6.9: Box trajectory. Comparison between the measured Cartesian positions.

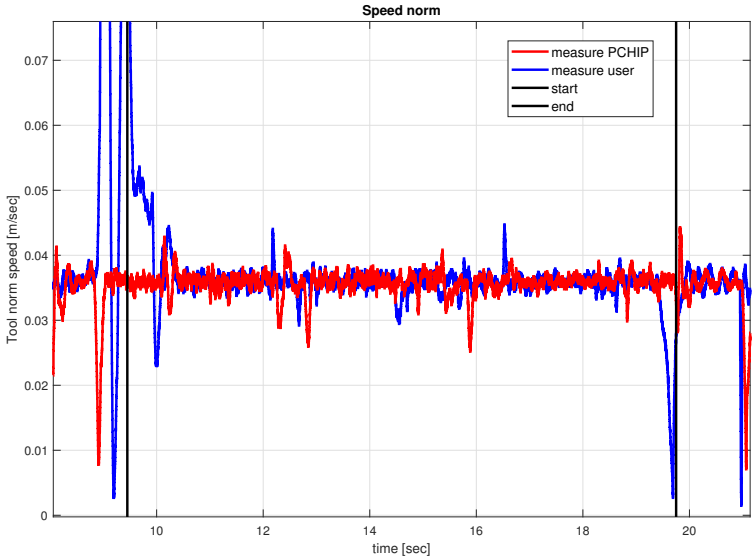

Figure 6.10: Box trajectory. Comparison between the measured velocities norm.

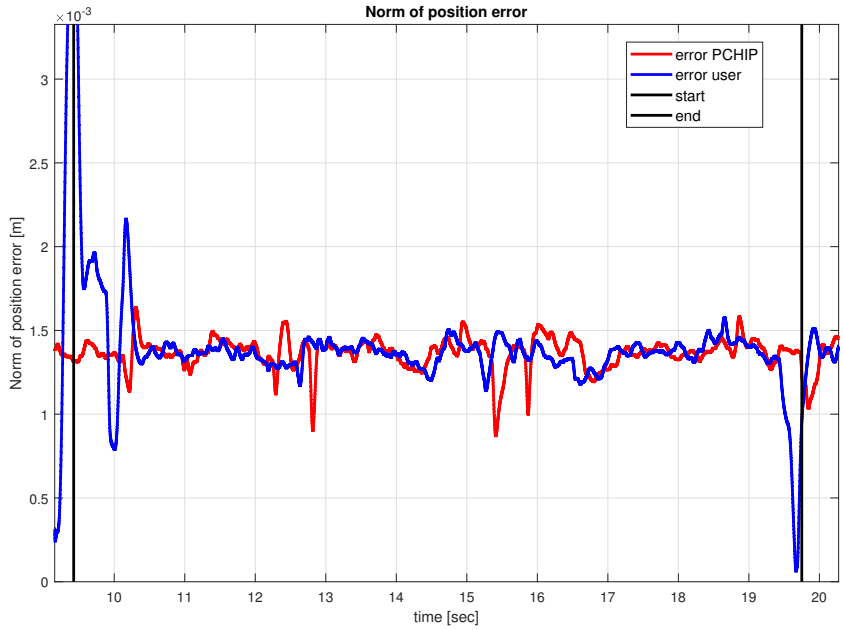

Figure 6.11: Box trajectory. Comparison between the measured tracking errors.

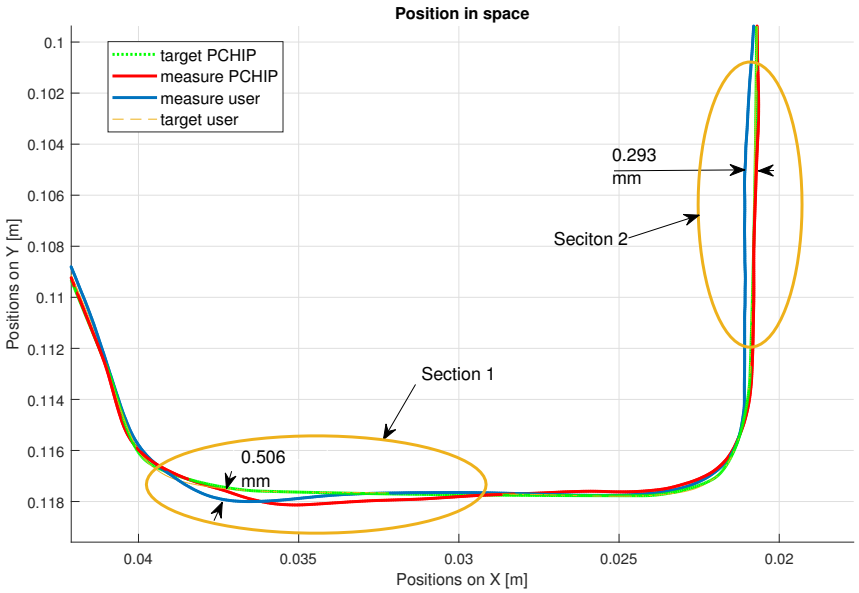

Figure 6.12: Box trajectory. Zoom of a portion of the path. Sections 1 and 2 indicate two zones where the tracking error is more relevant.

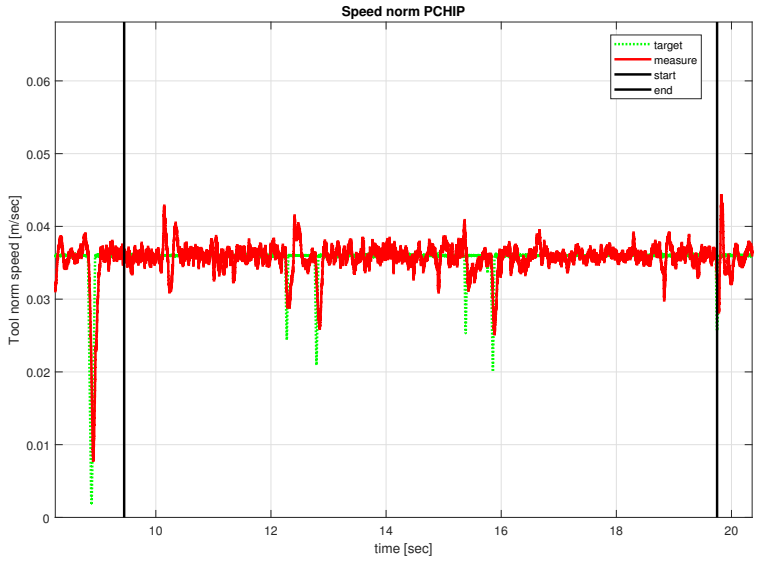

Figure 6.13: Box trajectory. Comparison between measured velocity and the target velocity planned in MatLab.

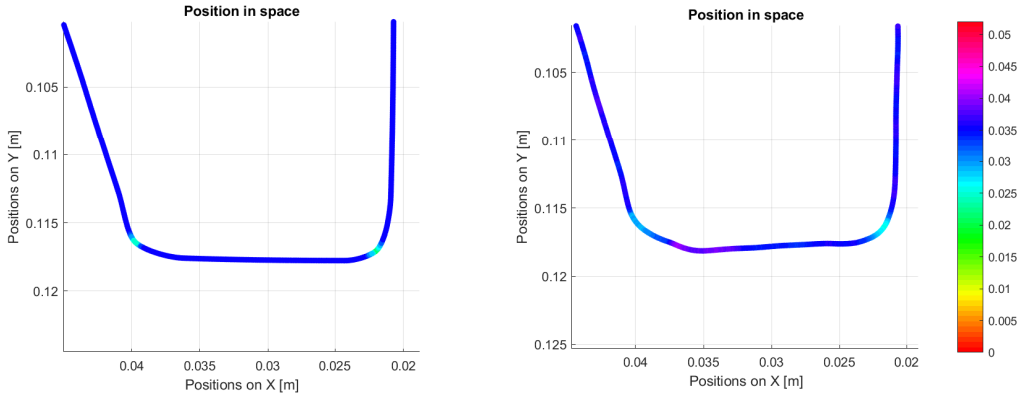

Figure 6.14: Box trajectory. On the left: target position and velocity; on the right: measured position and velocity. The colour scale is in m/s.

been scaled down with a factor 2.5. Also the orientation have been imposed constant, since our planning algorithm works only for the Cartesian position. Due to these changes, the final results may be different with respect to the original motion programmed by the user, nevertheless it is very useful to have a reference to compare with.

The input parameters for our planner are: cubic interpolation, same path points of the PDL program, target velocity of 400 mm/s, maximum acceleration of 3000 mm/s^2. The resulting interpolated path and the user target path are not exactly the

same, but they are similar enough to be comparable, as we can see in Figure 6.15.

The results of the tests are shown in Figures 6.15, 6.16 and 6.17 that report, respectively, the Cartesian positions, the velocity norm and the following error for both the PDL planning and the cubic planning.

The process is composed of two sections, in which the constant speed is required, connected by a reposition motion. This is highlighted in Figure 6.15. The measured tracking errors in Figure 6.17 are similar for the two planners. In Figure 6.18 there is a zoom of the reposition motion, where we expect to obtain the worst results due to the curve having a small radius. The path obtained with the COMAU planner has a maximum error of 2.89 mm, while the cubic planner, thanks to the programmed decelerations, achieves an almost constant error lower than 0.66 mm.

In Figure 6.19 we see the achieved speed profile compared to the one programmed in MatLab. The big gap in the velocity corresponds to the reposition motion. The filtering effect introduced by the physical system is very clear in this section. In this case it has a positive effect on the final results, since the resulting velocity is smoother than the programmed one.

The comparison between the speed profiles obtained with the two planners is shown in Figure 6.16. In this case the results obtained with the cubic planner show a clear improvement over the traditional planner. In particular, with the cubic planner, a constant speed is achieved during the whole first section, while the second section presents just two major decelerations plus a smaller one. On the other hand, the COMAU planner holds the constant speed only for small traits while having big speed changes in the rest of the trajectory.

Another test is done using the engine hood trajectory. This time we compare the planning using cubic and PCHIP interpolations. The target velocities are set at 100 mm/s with a maximum acceleration of 2000 mm/s^2. As expected from the MatLab simulations, the results shown in Figures 6.20 and 6.21 are similar. We can note a small giggling on the top right corner of the cubic Cartesian path in Figure 6.20, and smaller decelerations in its speed profile in Figure 6.21. Both of these behaviours meet our expectations, since the cubic interpolation produces a more rounded path that requires lower decelerations. As we can see in Figure 6.22, the target path is tracked with good precision with both planning techniques.

6.3.3 Greek fret trajectory

The last trajectory tested is the greek fret trajectory, which does not correspond to any real process. The PDL program is a sequence of linear and circular motion with fast direction changes useful to test the robot in different stress conditions. This program is executed at the maximum speed allowed.

The PCHIP interpolation is chosen for our planner, because the other techniques would distort the original path too much. The narrow angles smoothing is also

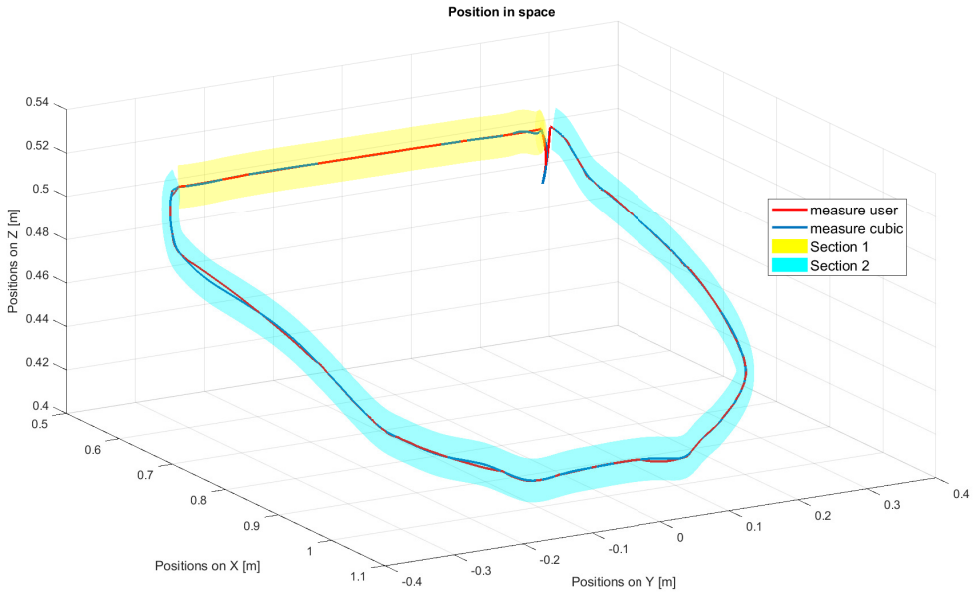

Figure 6.15: Engine hood trajectory. Comparison between the measured Cartesian positions.

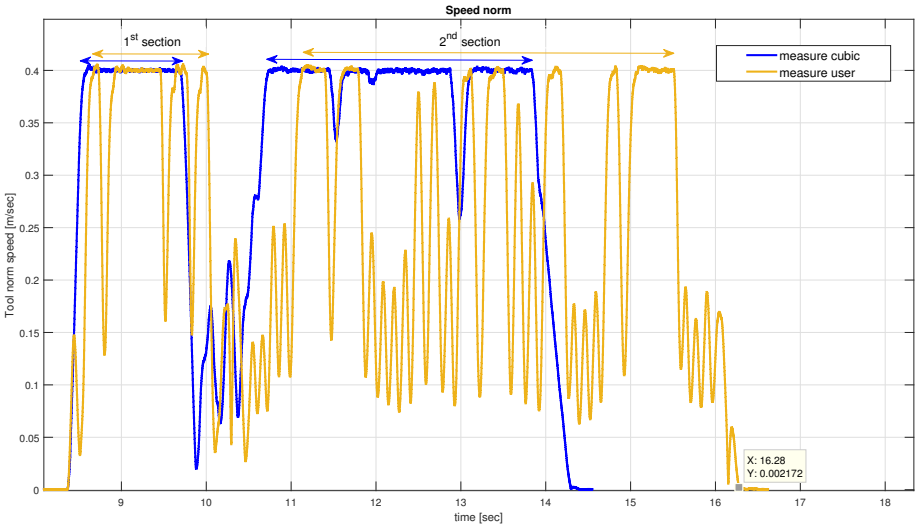

Figure 6.16: Engine hood trajectory. Comparison between the measured velocities norm.

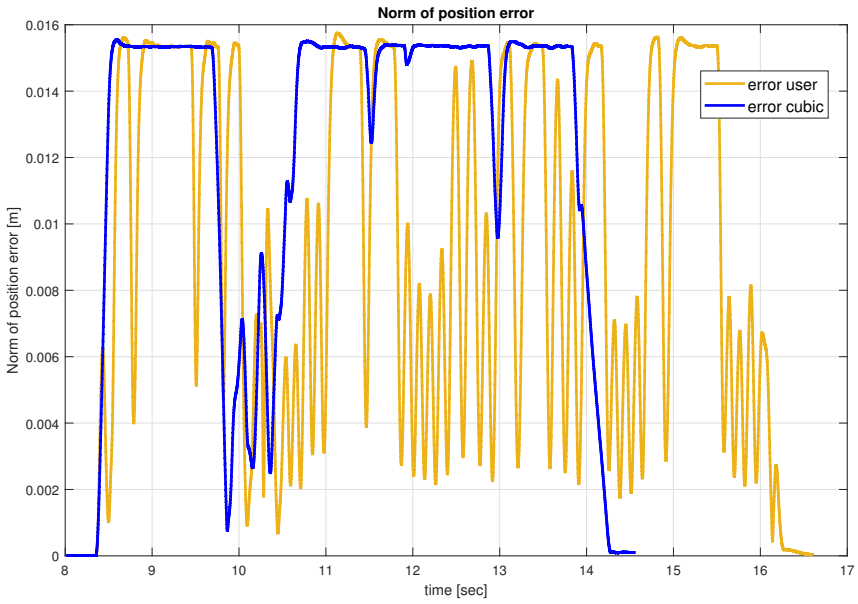

Figure 6.17: Engine hood trajectory. Comparison between the measured tracking errors.

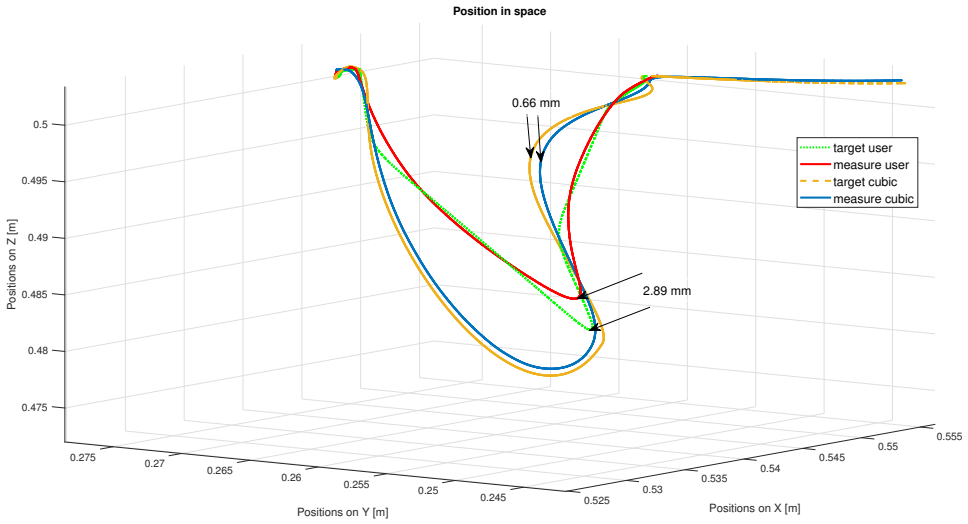

Figure 6.18: Engine hood trajectory. Zoom on the reposition motion between the two section of the path. Two distances between the target and the measured path are reported.

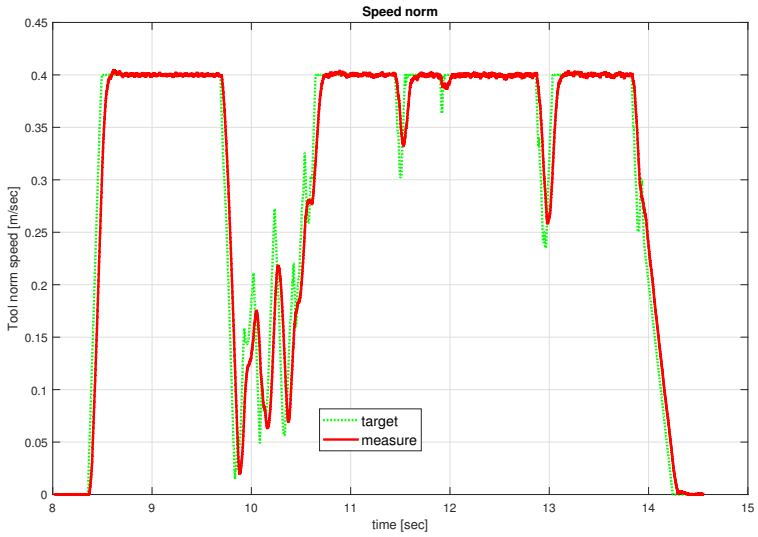

Figure 6.19: Engine hood trajectory. Comparison between measured velocity and the target velocity planned in MatLab.

applied with d = 2.5 mm. The target velocity is set to 400 mm/s and the maximum acceleration at 3000 mm/s^2.

Figures 6.23, 6.24 and 6.25 show, respectively, the Cartesian positions, the velocity norm and the tracking error for both the PDL planning and the PCHIP planning. In Figure 6.27 and Figure 6.28 two zooms of the critical sections of the path are shown to compare the tracking error. An interesting observation can be done on the speed reached during the execution of the section shown in Figure 6.27: both planners programmed its execution with similar speed, 60 mm/s for the PDL planning and 70 mm/s for the PCHIP planning, as we can see in Figure 6.24.

In this case the PCHIP planner achieves the target constant velocity only for small intervals, mainly due to the geometry of the path that imposes frequent changes of direction.

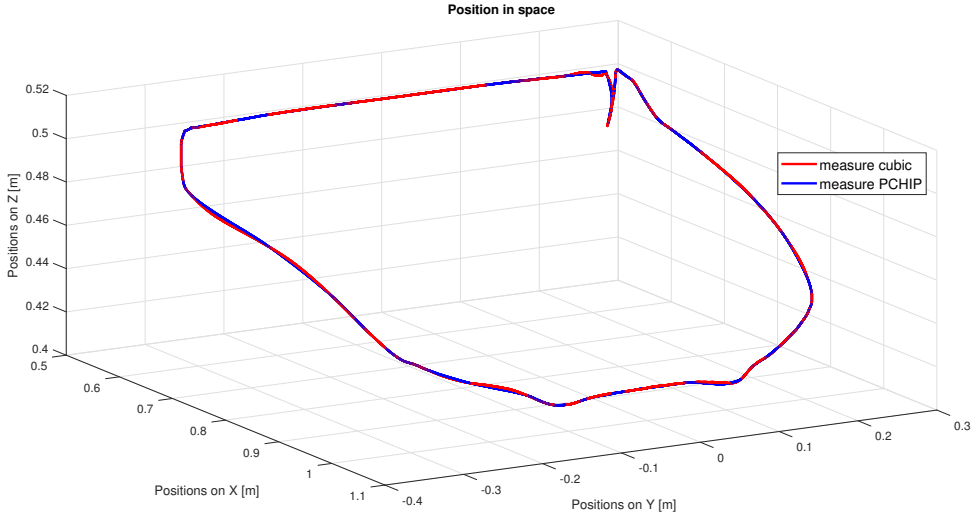

Figure 6.20: Engine hood trajectory. Comparison between the cubic and PCHIP measured Cartesian positions with velocity target at 100 mm/s.

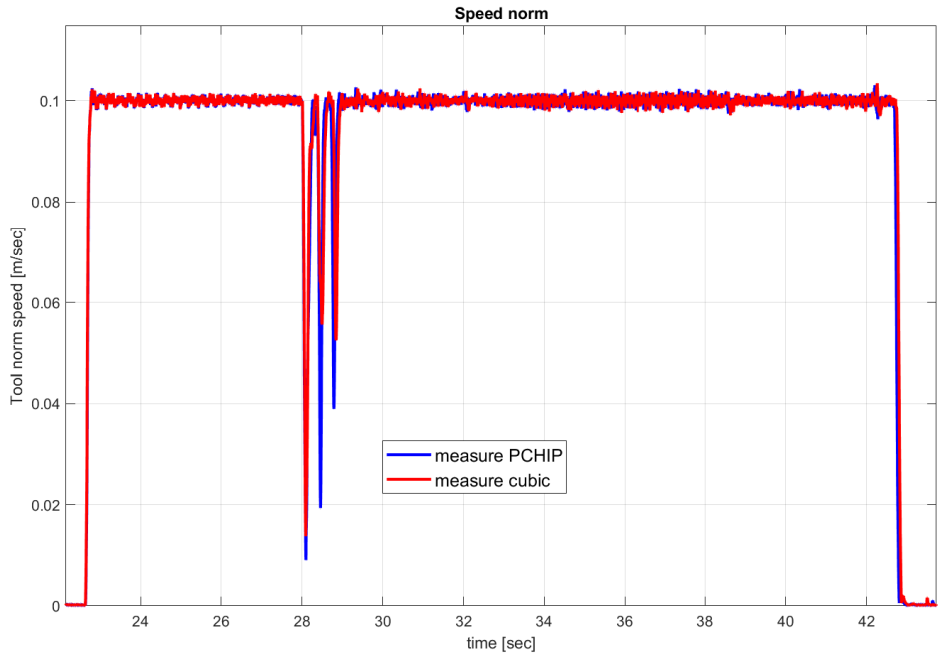

Figure 6.21: Engine hood trajectory. Comparison between the cubic and PCHIP measured velocities norm with velocity target at 100 mm/s.

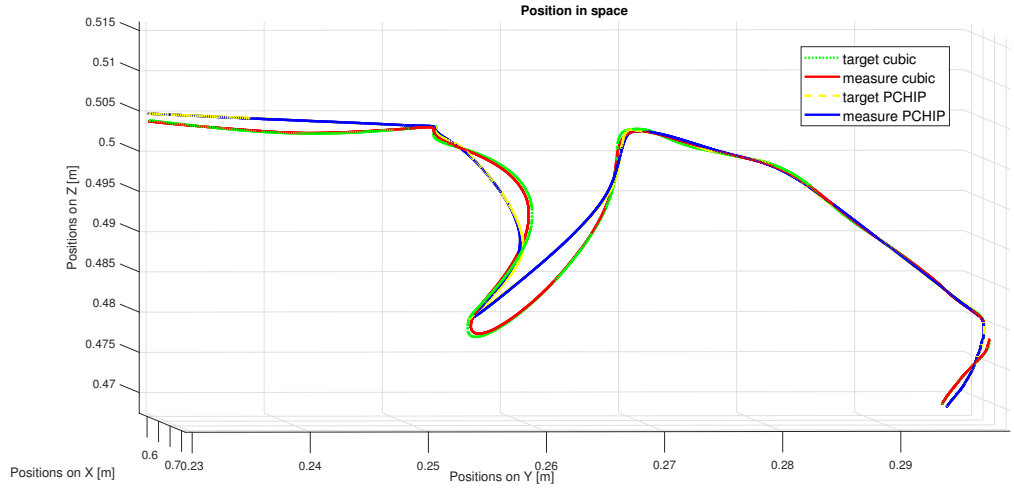

Figure 6.22: Engine hood trajectory. Zoom on the reposition motion between the two section of the path. Comparison between the cubic and PCHIP measured Cartesian positions with velocity target at 100 mm/s.

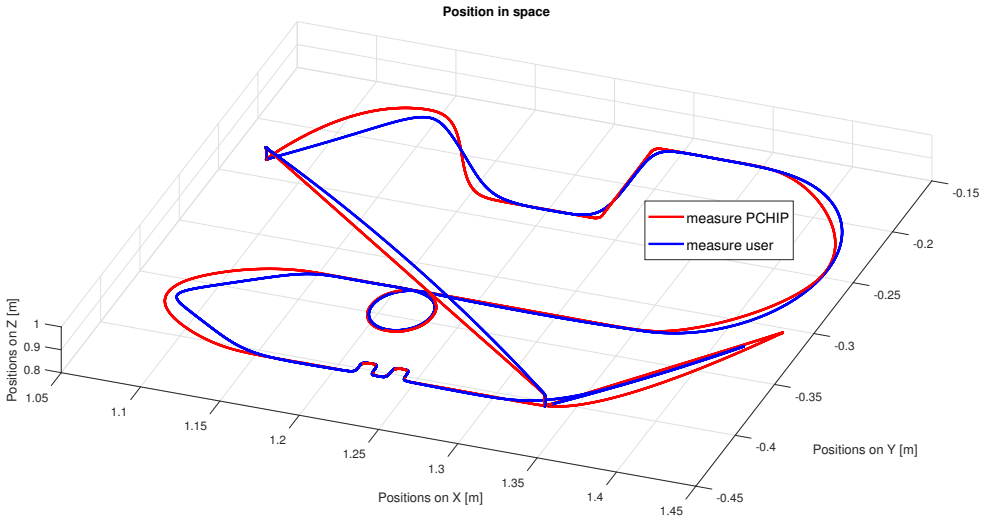

Figure 6.23: Greek fret trajectory. Comparison between the measured Cartesian positions.

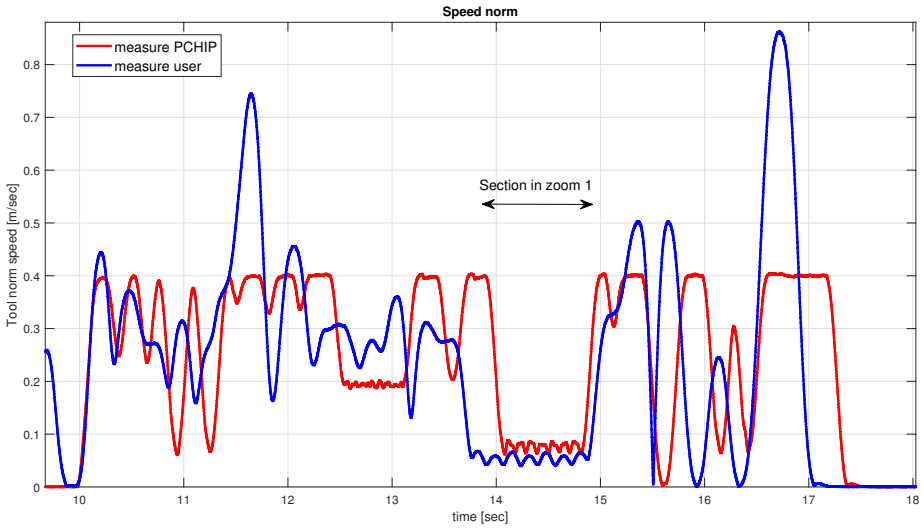

Figure 6.24: Greek fret trajectory. Comparison between the measured velocities norm.

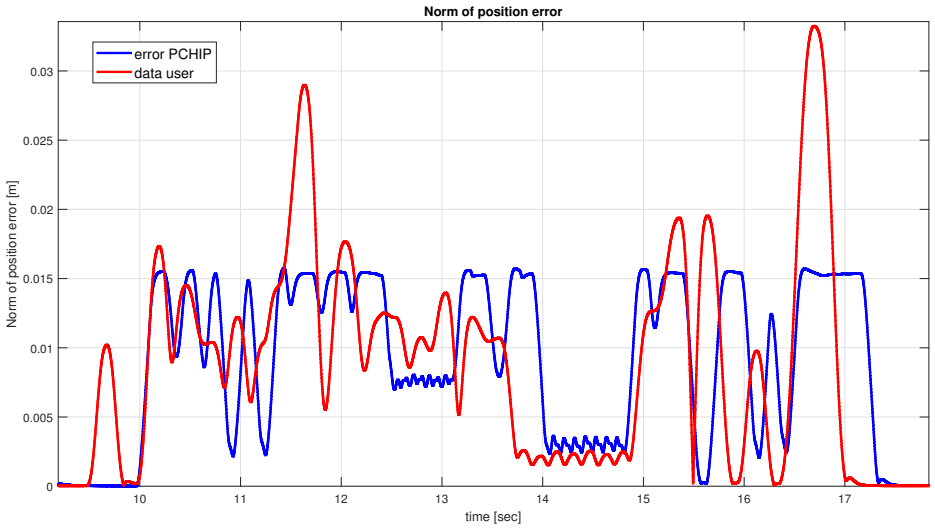

Figure 6.25: Greek fret trajectory. Comparison between the measured tracking errors.

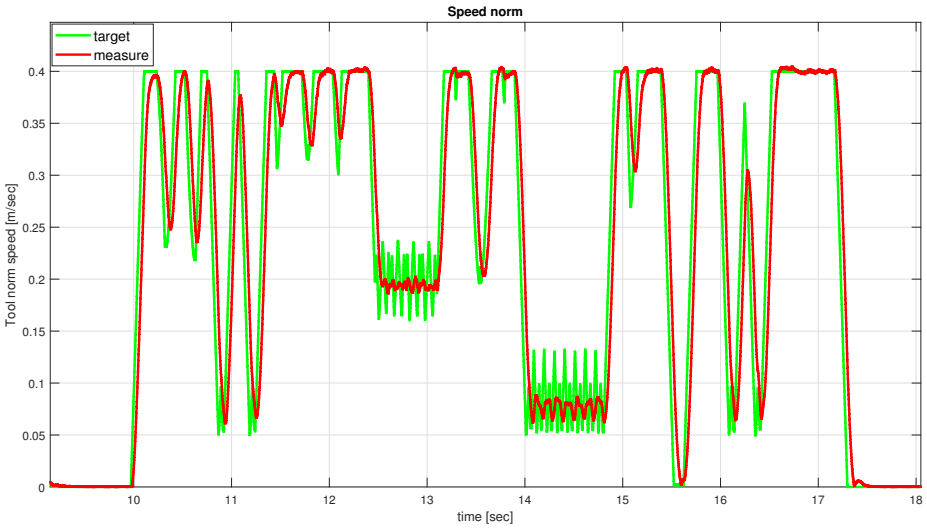

Figure 6.26: Greek fret trajectory. Comparison between measured velocity and the target velocity planned in MatLab.

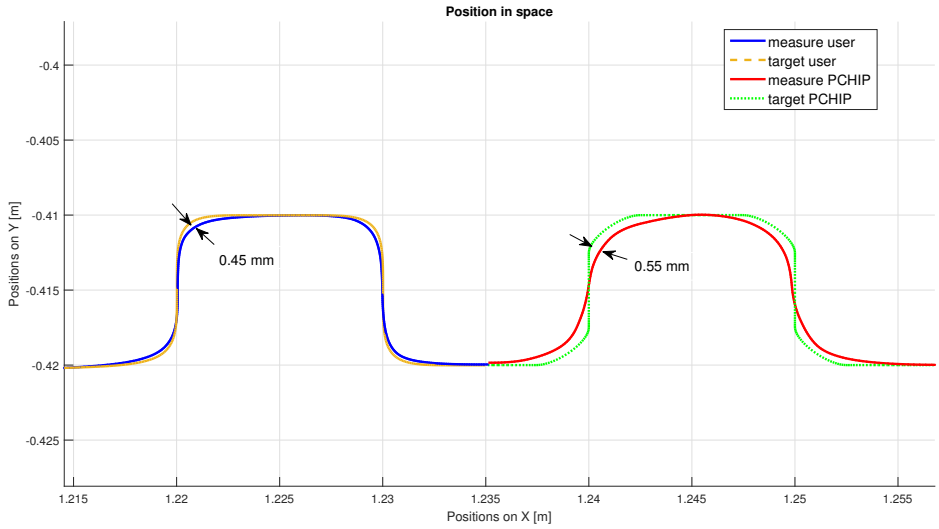

Figure 6.27: Greek fret trajectory. Zoom 1: zoom on the fast direction change section of the path.

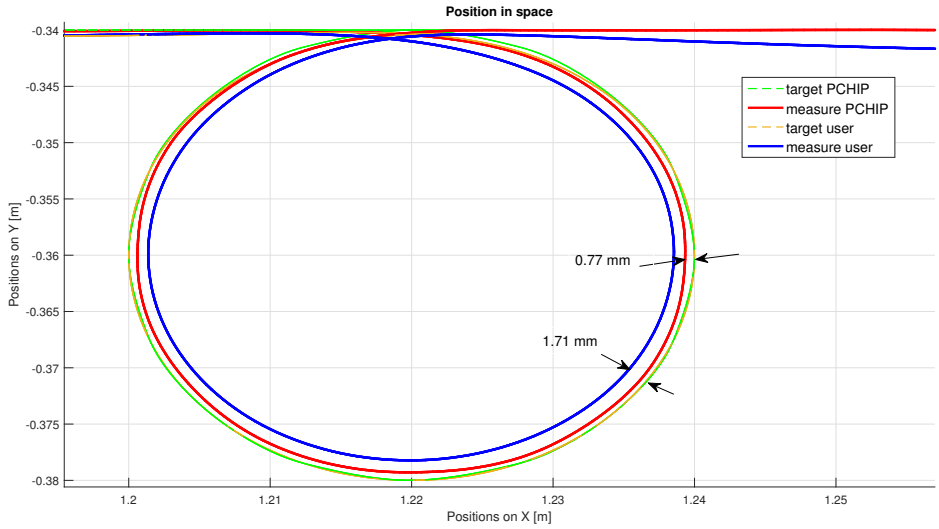

Figure 6.28: Greek fret trajectory. Zoom 2: zoom on the circular motion of the path.

Chapter 7

Conclusions

In this thesis the problem of the analysis and planning of continuous trajectories has been examined. Firstly a review of the theory behind the trajectory planning and of some common techniques is proposed. Then some examples of continuous processes commonly found in the manufacturing industry are examined. At last a novel trajectory planning technique is developed.

The trajectory planning is divided in two phases. In the first phase the input path points are interpolated with a spline function. Different interpolating techniques have been proposed in order to adapt to the necessity of the user. In particular the *Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)* seems to be the most useful in many practical cases.

Once the path function is computed, the dynamical properties of the trajectory are defined. Particular care is devoted to the analysis of the geometrical properties of the path and on the definition of a speed profile that adapts to it.

The algorithm is implemented in MatLab in order to test its performance on real cases. In particular a local trajectory planner is developed in order to simulate the condition of a real control unit, where the computational resources are limited and only a small subset of points can be computed at a time. A global version of the same algorithm is also provided in order to have a useful comparison with the proposed approach.

The planner is then validated with experimental tests run on a real COMAU robot. The measurements obtained show that the local trajectory planner provides a good solution for the continuous processes analysed. We can also observe a general improvement with respect to the standard COMAU planner applied on the same processes.

7.1 Future works

The proposed algorithm is only on its early stages of development. Before implementing it on a real controller, it is necessary to develop other features. The most important ones are:

- Interpolation of the orientation component of the path. The planner provides only the Cartesian positions of the trajectory, but in real cases it is also necessary to plan the orientation of the tool.
- Integrate an inverse kinematic module. Having the information in the joint space can be very useful to improve the quality of the planning. In particular the acceleration constraint could be directly taken from the maximum acceleration of the actuators, and the velocity profile could be adapted accordingly. It is also necessary to check violations of kinematic constraints and eventually reprogram the motion if this situations occur.
- Convert the MatLab code in C and optimize the computation time for real-time execution.

Bibliography

[1] Mayur V. Andulkar, Shital S. Chiddarwar, and Akshay S. Marathe. «Novel integrated offline trajectory generation approach for robot assisted spray painting operation». In: *Journal of Manufacturing Systems*. Elsevier, Oct. 2015, pp. 201–216.

[2] Basilio Bona. *Modellistica dei robot industriali*. Torino: Celid, Mar. 2014.

[3] Alessandro De Luca, Leonardo Lanari, and Giuseppe Oriolo. «A Sensitivity Approach to Optimal Spline Robot Trajectories». In: *Automatica* 27.3 (May 1991), pp. 535–539. Elsevier Ltd.

[4] J. De Maeyer, B. Moyaers, and E. Demeester. «Cartesian path planning for arc welding robots: Evaluation of the descartes algorithm». In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2017, pp. 1–8.

[5] David Eberly. *Moving along a curve with specified speed*. Redmond WA 98052: Geometric Tools, Apr. 2007. URL: https://www.geometrictools.com/.

[6] Jean Gallier. «Basics of the Differential Geometry of Curves». In: *Geometric Methods and Applications*. New York: Springer, May 2011. Chap. 19.

[7] Marina Indri. *Slides from "Robotica" course*. DAUIN - Politecnico di Torino, A.Y. 2016-2017.

[8] Claudio Melchiorri. *Slides from "Foundation of Industrial Robotics" course*. DEIS - University of Bologna, A.Y. 20010-2011.

[9] Bruno Siciliano et al. *Robotics - Modelling, Planning and Control*. Springer, 2009. Chap. 4.

[10] Marc Spiegelman. *Slides from "Numerical Methods" course*. Dept. of Applied Physics and Applied Mathematics - Columbia University, A.Y. 2007-2008.

[11] Ryan Tibshirani. *Slides from "Advanced Methods for Data Analysis" course*. CMU Statistics - Carneigie Mellon University, A.Y. 2013-2014.

[12] Adrien Treuille. *Slides from "Computer Graphics" course*. CMU Graphics - Carneigie Mellon University, A.Y. 2010-2011.

[13] G. Q. Zhang et al. «Robotic additive manufacturing along curved surface — A step towards free-form fabrication». In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dec. 2015, pp. 721–726.

Acknowledgement

I would like to express my deep gratitude to Professor Marina Indri, my thesis advisor, for her patient guidance and invaluable support during the development of this research. She allowed this paper to be my own work, but steered me in the right direction whenever it was needed.

I would also like to thank the COMAU R&D group, who hosted me and gave me the opportunity to use their laboratory. I am particularly grateful for the assistance given by Engineer Eliana Giovannitti throughout the entire course of the thesis. Her patience and precious knowledge have been fundamental for carrying out successfully the last testing phases.

Thank you to all my friends, with whom I shared these fantastic years and helped me to make it this far. To my colleagues from Politecnico, for the good moments we had both inside and outside the university. To my friends from "collegio", who have become like a second family to me. And to my friends from hometown, for making the little time I spent with them feel like I had never been away.

A special thanks goes to my granddad, that has always helped me in every way he could.

Then, I must express my profound gratitude to family. You have always supported me, and encouraged me to pursue whatever I most desire. This accomplishment would not have been possible without you.

Finally, thank you Francesca, for the love that you give me every day.