POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
Master of Science in Mechanical Engineering

Velocity planning of a robotic sealing task using Dynamical Movement Primitives

Thesis advisor:
Prof. Andrea Maria ZANCHETTIN

Research supervisor:
Ph.D. Loris ROVEDA

Candidates:
Beatrice MAGGIONI
ID. 920303

Elia MARESCOTTI
ID. 920571

Academic Year 2019 – 2020

Acknowledgements

First of all, we want to thank our advisor, Prof. Andrea Maria Zanchettin, for his precious suggestions and his guidance during these months. A big thank also to Ph.D. Loris Roveda, co-advisor of this work, which spent many days helping us, especially in the last period when we faced the experimental stage of our thesis. We also want to thank Prof. Mario Covarrubias for providing us a space to work in his virtual prototyping laboratory in the Lecco Campus during this global emergency situation.

We want to thank the friends and colleagues that have shared this University journey and made it more pleasant even in the difficult situations. Special thanks to the families that have supported us from the very beginning, providing good advices and believing in us. We want to send the dearest gratitude to the grandparents that have grown us wiser and stronger, with the biggest hope to still be your proud.

Milano, December 2020

Beatrice Maggioni

Elia Marescotti

The most dangerous phrase in the language is:
"We have always done it this way"

Grace Hopper

Abstract

Within the Industry 4.0 paradigm, robotic programming of industrial tasks is developed with increasing automation. This thesis work is centered on the optimization of a sealing task, which requires an advanced velocity planning to achieve the quality standard of the final products. The research focus is the trajectory planning using feature recognition and the analysis of the experimental task.

The developed algorithm enhances learning and reproduction of a taught path, by generating a trajectory planner for the end-effector motion that automatically characterizes the execution through a punctual velocity reference defined accordingly with the path geometry. A Fuzzy Logic controller has been exploited to assign the right execution velocity to each path features (such as curves or sharp edges). The computed velocity reference is then used as input for a Dynamical Movement Primitives framework, which computes the position for the robot end-effector.

The proposed work aims to create a robust controller which is able to recognize the path geometry and execute it with an optimized velocity, while it also provides a smoothing effect on the reference, taking into account the acceleration limits of the physical setup.

This method has been implemented to work also in a collaborative environment, by generating an online trajectory planner which allows work-sharing between the robot and the human operator such that the operator can interact on the manipulator without stopping the task execution.

Keywords: Autonomous Robotics, Collaborative Robotics, Dynamical Movement Primitives, Fuzzy Logic, Trajectory Planning

Sommario

Nel paradigma dell'industria 4.0, la programmazione di robot per operazioni industriali viene sviluppata con metodi sempre più automatizzati. Questo lavoro di tesi si focalizza sull'ottimizzazione di una applicazione di sigillatura, per la quale è necessario descrivere un pianificatore avanzato di velocità al fine di raggiungere gli standard qualitativi richiesti. L'obiettivo della ricerca è una programmazione di traiettoria che utilizzi un algoritmo di riconoscimento e analisi della geometria del percorso di riferimento.

L'algoritmo sviluppato permette il riconoscimento e la riproduzione di un percorso insegnato, attraverso la generazione della traiettoria che descrive il movimento dell'utensile montato all'estremità del braccio robotico. Il percorso viene automaticamente caratterizzato in maniera puntuale da una velocità di riferimento definita sulla base della geometria del tratto. Si è scelto di utilizzare un controllore con logica Fuzzy per assegnare una appropriata velocità di esecuzione ad ogni geometria del percorso (curve, angoli retti, ecc). La velocità calcolata viene poi utilizzata come variabile di ingresso per le Dynamical Movement Primitives, le quali calcolano la posizione di riferimento del robot nel suo spazio operativo.

Il lavoro proposto genera un controllore robusto che è in grado di riconoscere le geometrie del percorso ed eseguirle con una velocità ottimizzata, introducendo un filtraggio del segnale di riferimento sulla base dei limiti di accelerazione del setup sperimentale utilizzato.

Il metodo è stato implementato con il fine di lavorare anche in ambienti collaborativi, attraverso la generazione in tempo reale della traiettoria da eseguire. Questo approccio permette una ripartizione del lavoro tra il robot e l'operatore, in modo tale che il tecnico possa intervenire sull'operazione automatizzata senza bloccarne l'esecuzione.

Parole chiave: Dynamical Movement Primitives, Logica Fuzzy, Pianificazione di Traiettoria, Robotica Autonoma, Robotica Collaborativa

Contents

List of Figures

List of Tables

Acronyms

BOM	Bill of Materials A bill of materials is a list of the raw materials, sub-assemblies, intermediate assemblies, sub-components, parts, and the quantities of each needed to manufacture an end product. https://en.wikipedia.org/wiki/Bill_of_materials
CAD	Computer Aided Design CAD is the use of computers (or workstations) to aid in the creation, modification, analysis, or optimization of a design. CAD software is used to increase the productivity of the designer, improve the quality of design, improve communications through documentation, and to create a database for manufacturing. https://en.wikipedia.org/wiki/Computer-aided_design
CS	Canonical System In DMP context, the canonical system is a simple dynamical system needed to make the DMP system autonomous (the time-dependency is expressed inside CS, leaving only an implicit time dependency inside DMP).
DK	Direct Kinematic The aim of direct kinematic is to compute the pose of the end-effector as function of the joint variables (i.e. we can compute the task space variables form the joint ones) [1, 2].
DMP	Dynamical Movement Primitives DMP are a formulation of movement primitives with autonomous nonlinear differential equations, whose time evolution creates smooth kinematic control policies. They guarantee stability and convergence properties of learned trajectories, and scale well to high dimensional data [3].
DOF	Degrees of Freedom In physics, the DOF of a mechanical system are the number of independent parameters that define its configuration or state. A rigid body in space is defined by 3 translational and 3 rotational DOF, for a total of 6 DOF. https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics)

EE	**End-Effector** The end-effector is a generic term that includes all the devices that can be installed at a robot wrist, the most common one is the gripper. https://blog.robotiq.com/bid/53266/Robot-End-Effector-Definition-and-Examples
FCI	**Franka Controller Interface** The Franka Controller Interface allows a fast and direct low-level bidirectional connection to the Arm and Hand. It provides the current status of the robot and enables its direct control with an external workstation connected via Ethernet. https://frankaemika.github.io/docs/
FDM	**Fused Deposition Modeling** 3D printers that run on FDM technology build parts layer-by-layer from the bottom up by heating and extruding thermoplastic filaments. This 3D printing technology can achieve the best accuracy and repeatability, while having durable and dimensionally stable parts. https://www.stratasys.com/fdm-technology
FJR	**Flexible-Joint Robot** Manipulators with rigid links and deflection only in correspondence of joints are defined as FJR. They present an elasticity concentrated only at the joint positions.
FL	**Fuzzy Logic** In Fuzzy mathematics, FL is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 both inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and false. By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1. https://en.wikipedia.org/wiki/Fuzzy_logic
IK	**Inverse Kinematic** The inverse kinematics problem consists of the determination of the joint variables corresponding to a given end-effector position and orientation. This problem has typically multiple possible solutions and their number rises with the number of DOF of the robotic arm [1, 2].
LWR	**Locally Weighted Regression** LWR is a memory-based method that performs a regression around a point of interest using only training data that are ”local” to that point. LWR is suitable for real-time control by constructing an LWR-based system. https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/cohn96a-html/node7.html

PWM	Pulse-Width Modulation PWM is a method that reduces the average value of a signal. The average value is controlled by switching between ON and OFF states at a high frequency. The mean value of the resulting square wave is the signal output. https://en.wikipedia.org/wiki/Pulse-width_modulation
ROS	Robot Operating System The ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. https://www.ros.org/about-ros/
SCARA	Selective Compliance Assembly Robot Arm A SCARA robot is a multi-DOF robotic manipulator which is compliant in the X-Y axis, and rigid in the Z-axis. The SCARA configuration is unique and designed to handle a variety of material handling operations. https://www.fanuc.eu/de/en/robots/robot-filter-page/scara-series/selection-support
STL	STereoLitography STL has several backronyms such as "Standard Triangle Language" and "Standard Tessellation Language". STL files describe only the surface geometry of a three-dimensional object without any representation of color, texture, or other common CAD model attributes. It is widely used for rapid prototyping, 3D printing and computer-aided manufacturing. https://en.wikipedia.org/wiki/STL_(file_format)
TCP	Tool Center Point The TCP determines the exact working point of the tool in robot tool measurement. https://www.leoni-factory-automation.com/en/products-and-services/glossary/tool-center-point-tcp/

Chapter 1

Introduction

The development of Industry 4.0 is requiring that an increasing number of manual tasks have to be executed in a more efficient way, firstly by time-optimization, but also through higher quality standards. To achieve that, automatization is becoming the preferential solution in several different manufacturing environments, from automotive and aerospace to food industry. The use of automatized machines is taking over human force, thanks to their larger potentialities. In this background, the study and implementation in factories of industrial manipulators and collaborative ones is continuously rising.

1.1 Thesis Objective

This thesis focuses on the optimization of an automatized sealing task, which is executed by a collaborative robotic arm in a human-robot collaborative workspace. The automatization applied to the production process enhances the repeatability of the results if compared to the one achieved by a human operator, while also allowing higher production rate and related cost-cutting.

The proposed work introduces a novel approach to the core of task planning, which is the trajectory definition. A general input path must be characterized by an execution velocity, which in general can be a continuous varying reference. This analysis is crucial in the background of sealing applications, where a generic path needs to be defined not only by its geometrical dimensions, but also by a time characterization. In these kinds of tasks, both the End-Effector (EE) velocity and the deposition flow are parameters that must be precisely controlled in order to achieve a proper sealant distribution.

The here-presented analysis permits to set the deposition velocity reference through an automatic control algorithm instead of requiring a manual definition by the operator. In this way, the operator just needs to define the path to be executed, then the algorithm analyzes it identifying the geometry features and relating a reference velocity to each point.

Even if the work has been developed on a reference sealing task, the code provides a very general framework, that allows the use of it also for other applications outside the thesis topics. Moreover, once the algorithm is tuned on the productive setup, it can approach any path, without the need to manually tune the controller every time the reference path is slightly changed.

The trajectory planning characterization combined with the automatic velocity reference can be described as a general purpose planning algorithm, since it can be implemented in numerous applications, but it is then applied on a quite un-traditional setup. Indeed, larger studies and analysis for trajectory planning in sealing, welding and cutting application have been made for industrial robots and it is quite uncommon to perform the sealing task in a collaborative environment. This characteristic of the provided testbed also required the implementation of a model that could adapt and benefit from the interaction with the human operator.

All the thesis materials, which includes the developed Python algorithms, Arduino sketches and 3D CAD drawings, are available at:

https://github.com/emarescotti/VelocityPlanning_DMP_FL

1.2 Thesis Structure

This introduction aims to give a brief description of the presented thesis work, explaining the topics of each chapter and the workflow from State of the Art trajectory planning strategies to the experimental validation of the proposed novel approach.

In the following chapter [chap. 2], some basic concepts are remarked. Starting from the background related to the manipulation definition and its constitutive equations, are then presented the already existing general trajectory planning algorithms and industrial applications similar to the reference task.

This thesis analysis selects an innovative Dynamical Movement Primitives (DMP) framework, which modifies the standard DMP approach introducing a variable velocity reference along the path. To clarify the classical DMP theory, in [chap. 3] all the formulations provided by Ijspeert and Schaal [4] and further studies are provided.

In [chap. 4] the used experimental setup is described. It is reported a brief analysis of the provided manipulator, a Franka Emika Panda robot, and a study of the commercial caulking gun used to perform sealant deposition. Additional constructive tips necessary for the implementation of the test-bed are provided in [app. A].

The novelty introduced by this thesis is the path analysis and the automatic velocity reference generation with acceleration constraint validation, which are widely described in [chap. 5]. The Fuzzy Logic (FL) controller theory is introduced and it is presented the specific tuning of the membership functions related to the actual work. To facilitate the comprehension of the complete algorithm, some parts of the Python code are provided in [app. B], i.e. the acceleration limits analysis during the velocity definition.

The experimental validation of this thesis work is carried out in [chap. 6]. This chapter aims to compare the results obtained with a classical DMP formulation with respect to the complete approach developed in this thesis work. Relevance focus has been made both on the tuning of the model, both on the validation experiments performed.

At last, [chap. 7] depicts all the considerations about the experiments and the pros and cons of the developed method are discussed. Indeed, it is also reported a brief parley over the open possibilities of the method and on further applications that could be implemented.

Chapter 2

State of the Art

The increasing need of automatization for traditional manual processes requires the use of machines and robots that are precisely trained and programmed to execute the planned task. For standard series productions automatized industrial machines were preferred over robots, but in a framework of high variability of applications and tasks, robots are gaining space over the market.

The International Federation of Robotics (standard ISO/TR 8373), defines a manipulator as follows [5]:

> “A manipulating industrial robot is an automatically controlled, re-programmable, multipurpose manipulator, programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.”

In this framework, this study is focused on the fixed place robots, and more specifically on collaborative ones, which are conceived as "mechanical colleagues" of the human, where a relaxed human-robot interaction can be created, and the machine can be an assistant for reaching the final goal [6]. This thesis analyses the use of a collaborative robot for an industrial process, which is the sealing of an aerospace panel. The selection of this type of machine is made with the aim of accomplishing the task while being close to a human collaborator which can enter in the robot operating space and act on the environment [fig. 2.1].

Trajectory execution is a quite common task performed by robots, thanks to their ability to firmly reproduce the provided route. Manipulators must at least convey a movement form the starting to goal position, then the selected motion control algorithm will characterize the transition.

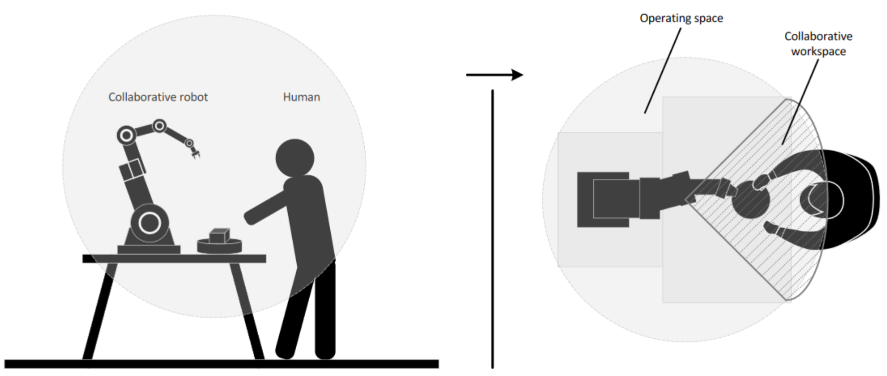


Figure 2.1: Human-Robot collaborative workspace. Source: [6]

Several algorithms have been developed, and they are always referred to as *trajectory planning.*

Before analysing possible methods, it must be clarified the difference between path and trajectory [2]:

> “A *path* denotes the locus of points in the joint space, or in the operational space, the manipulator has to follow in the execution of the assigned motion; a path is then a pure geometric description of motion. On the other hand, a *trajectory* is a path on which a time law is specified, for instance in terms of velocities and/or acceleration.”

2.1 Robots Mechanical Structures

Robots are complex electro-mechanical structures composed of multiple sub-systems. The mechanical skeleton is typically made of a locomotion apparatus, for mobile robots, and of a manipulation one, typically mechanical arms and EE. The motion of this structure is conveyed through actuators, like servomotors, drives and transmission, that are controlled by a power electronic which imposes the motion control strategy. Robots are also equipped with sensors that are necessary to perform measures and acquire data both related to the executed movement, both to the external environment, like force sensors or cameras.

The manipulator structure is composed of rigid bodies (*links*) connected by hinges or articulations, the *joints.* The arm is the sequence of links that ensures mobility, while the EE is the component that performs the task. All movements are conveyed through the joints, which can perform:

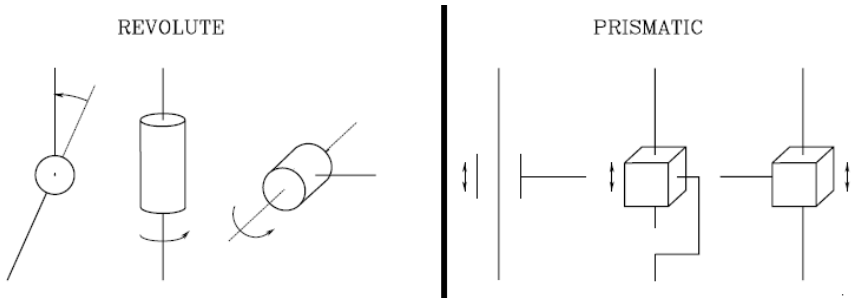
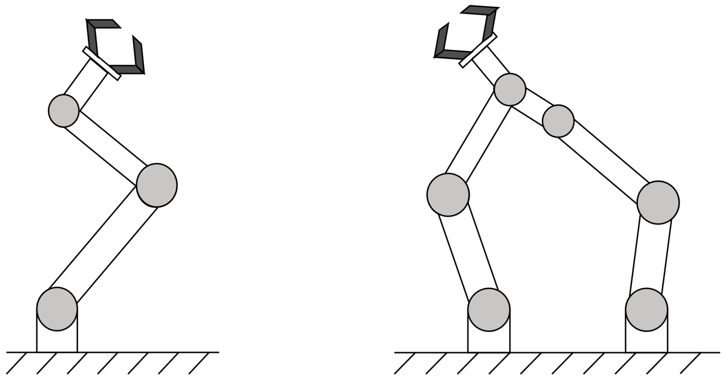


Figure 2.2: Conventional representation of joints. Source: [2]

- Revolute motion, a relative rotation between the two links;
- Prismatic motion, a relative translation between the two links.

those two types of joints are graphically schematized as in [fig. 2.2].

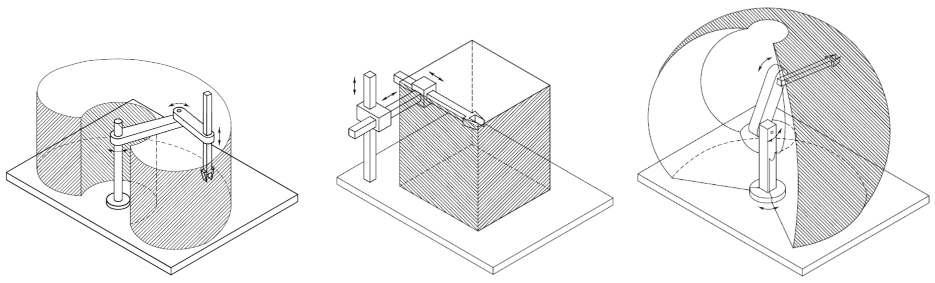
The structure of the robot influences the complexity of the system. For open-kinematic-chain robots the number of joints corresponds to the number of Degrees of Freedom (DOF), while for closed-kinematic-chain the number of DOF differs from the joints one, due to closed-loop physical constraints [2]. In the scope of this thesis, the former structure will be selected, because it is in accordance with the given setup [chap. 4]. Two example configurations are reported in [fig. 2.3].



(a) Open-kinematic-chain **(b)** Closed-kinematic-chain

Figure 2.3: Comparison of the two kinematic chain structures of manipulators.

The selection of the manipulator must be related to the task that has to be computed. If a positioning and orientation of the EE is required in three-dimensional space (3D), then 6 DOF are required, since 3 of



(a) SCARA Robot **(b)** Cartesian Robot **(c)** Antropomorphic Robot

Figure 2.4: Comparison of different workspaces of manipulators. Source: [2]

them characterize the position (x, y, z) while the other 3 characterize the orientation with respect to the reference frame (ϕ, ϑ, ψ).

Common applications in 2D space can be achieved either with simple robots as Cartesian or Selective Compliance Assembly Robot Arm (SCARA) ones, that perform movements on parallel planes, or with more complex manipulators, as anthropomorphic ones, that typically have more DOF, and sometimes are even *redundant*, as they dispose a number of DOF higher than the one needed to characterize a position in space (i.e. manipulator of 7 DOF for a 3D movement). Thanks to these possible multiple rotations of arms, these latter robots can execute more complex movements and can cover larger task areas. In this framework the concept of *workspace* has to be introduced as the locus of points that can be reached by the manipulator Tool Center Point (TCP) for executing a task. A graphical representation of the different workspaces related to different robots is reported in [fig. 2.4].

Very simple robots as the Cartesian one reach univocally each point, while others with more complex structure (SCARA, Anthropomorphic, etc.) can display multiple possible configurations. In case of anthropomorphic robots this extended motion is directly related to the mechanical structure, which is typically a sequence of links and pure revolution joints, that occupy a reduced space while permitting extensive motion in 3D. A scheme of a 3 DOF anthropomorphic manipulator is reported in [fig. 2.5], where each joint is characterized by its reference frame (x_i, y_i, z_i) and the admitted movement which in this case is a pure rotation ϑ. The EE frame is: (x_3, y_3, z_3). Knowing the position of the TCP is fundamental to describe its motion in the space, and so to define a proper trajectory planning algorithm. To evaluate this position, there are two possible approaches: *direct kinematics* and *inverse kinematics*.

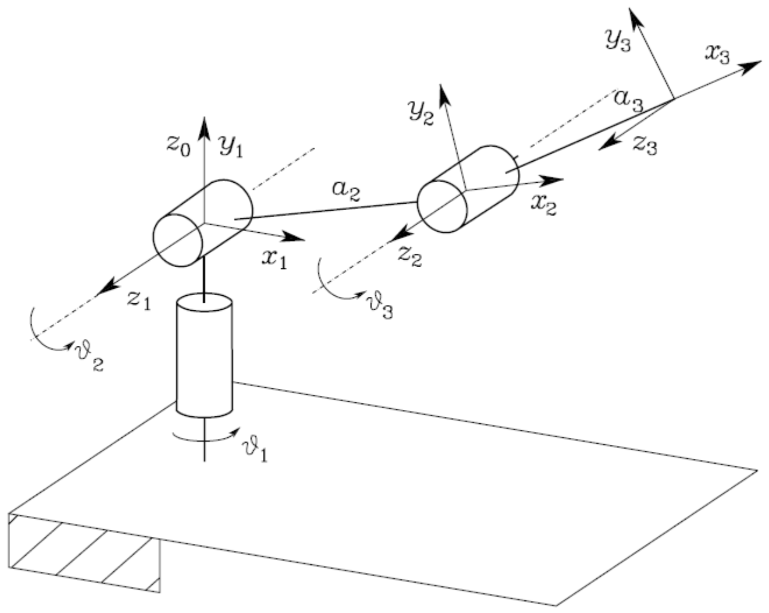


Figure 2.5: Anthropomorphic arm representation. Source: [2]

2.1.1 Direct and Inverse Kinematics

Finding the pose of the EE is fundamental to describe the desired tasks that must be reproduced by the manipulator. In Direct Kinematic (DK), the EE position is computed as a function of joint variables.

The posture of the EE can be described through a vector $\boldsymbol{x} \in \Re^m$, which collects the positions $\boldsymbol{p}_e$ and orientations $\boldsymbol{\phi}_e$ of the robot hand:

$$\boldsymbol{x}_e = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix} \tag{2.1}$$

while joint positions, $\boldsymbol{q} \in \Re^n$, which are represented with rotations and translations according to the type of joint, are:

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{2.2}$$

where $m \leq n$ so that all points are reachable. DK equations describe the position of the hand gripper as function of the joint ones:

$$\boldsymbol{x} = \boldsymbol{k}(\boldsymbol{q}) \tag{2.3}$$

In Inverse Kinematic (IK) instead, the target is commonly to define the joint positions while knowing the EE ones. This permits to define the joints movements, and then compute the torques and force actions to drive motion. Mathematically, these variables $\boldsymbol{q}$ are evaluated through an inverse relationship with respect to DK:

$$\boldsymbol{q} = \boldsymbol{k}^{-1}(\boldsymbol{x}) \tag{2.4}$$

The definition of these equations is not trivial when the number of DOF increases, because the relationships are mostly non-linear and the evaluation of the EE orientations $\boldsymbol{\phi}_e$ may require the use of rotation matrices.

2.1.2 Rotation Matrices, Euler Angles and Quaternions

A *rotation matrix* is a compact notation used to describe the orientation of a point with respect to a selected reference frame [fig. 2.6].

The *base frame* is (x, y, z), while the object frame is (x', y', z'). To describe the orientation of the body, the versors describing its position must be calculated referring to the base frame:

$$\begin{cases} \hat{\boldsymbol{x}}' = x'_x\hat{\boldsymbol{x}} + x'_y\hat{\boldsymbol{y}} + x'_z\hat{\boldsymbol{z}} \\ \hat{\boldsymbol{y}}' = y'_x\hat{\boldsymbol{x}} + y'_y\hat{\boldsymbol{y}} + y'_z\hat{\boldsymbol{z}} \\ \hat{\boldsymbol{z}}' = z'_x\hat{\boldsymbol{x}} + z'_y\hat{\boldsymbol{y}} + z'_z\hat{\boldsymbol{z}} \end{cases} \tag{2.5}$$

where a'_a terms represent the direction cosines with respect to each axis.

By switching to a matrix compact formulation, it is possible to obtain the *rotation matrix*:

$$R = \begin{bmatrix} \hat{\boldsymbol{x}}' & \hat{\boldsymbol{y}}' & \hat{\boldsymbol{z}}' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & x'_x \\ x'_y & y'_y & y'_z \\ x'_z & y'_z & z'_z \end{bmatrix} \tag{2.6}$$

The terms of the matrix represent the direction cosines needed to define the orientation of the new reference frame with respect to the base one. When multiple rotations are done, it is better to evaluate a rotation matrix for each rotation angle and then compute the overall one.

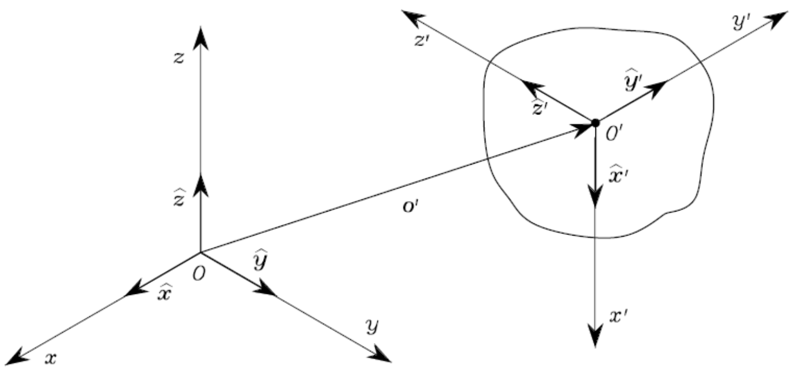


Figure 2.6: Orientation and position of a rigid body in space. Source: [2]

The definition of the 3 rotational coordinates $\boldsymbol{\phi}_e$ that describe the orientation of a body in space is done though *Euler Angles*. Euler Angles are a set of three angles:

$$\boldsymbol{\phi}_e = [\phi, \vartheta, \psi]^T \tag{2.7}$$

which represent the rotation of a reference frame with respect to a base one. They are commonly used to compute the direction cosines of the rotation matrices.

The definition of the Euler Angles is not trivial, because 12 different *sets of angles* can be used to describe an orientation. These *sets* depend on the sequence of rotations performed to reach the final position. To better explain this concept, refer to [fig. 2.7] and to the following calculus [7].

Considering to perform a rotation from the reference frame $\boldsymbol{x} = (x, y, z)$ to the final reference one $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y}, \bar{z})$, it is possible to perform three separate rotations using three rotational matrices. Considering the case showed in [fig. 2.7], the procedure is:

- Perform a rotation ϕ around x-axis, $\boldsymbol{x} = X\bar{\boldsymbol{x}}^I$:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \bar{x}^I \\ \bar{y}^I \\ \bar{z}^I \end{bmatrix} \tag{2.8}$$

- Perform a rotation ϑ around y-axis, $\bar{\boldsymbol{x}}^I = Y\bar{\boldsymbol{x}}^{II}$:

$$\begin{bmatrix} \bar{x}^I \\ \bar{y}^I \\ \bar{z}^I \end{bmatrix} = \begin{bmatrix} \cos(\vartheta) & 0 & \sin(\vartheta) \\ 0 & 1 & 0 \\ -\sin(\vartheta) & 0 & \cos(\vartheta) \end{bmatrix} \begin{bmatrix} \bar{x}^{II} \\ \bar{y}^{II} \\ \bar{z}^{II} \end{bmatrix} \tag{2.9}$$

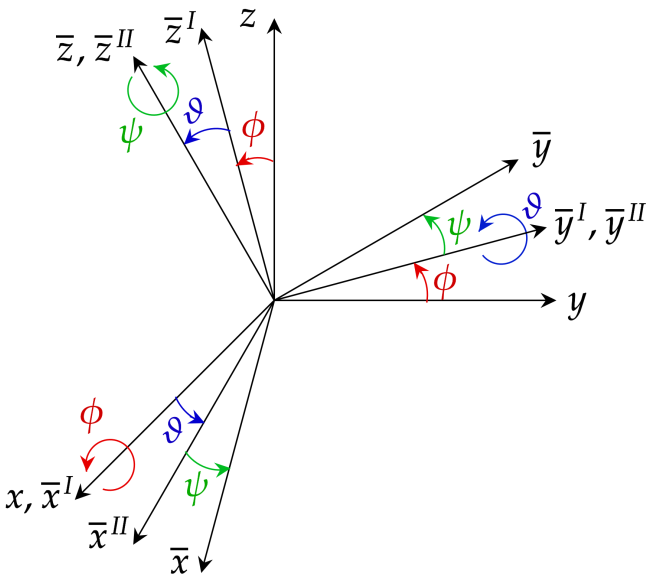


Figure 2.7: Reference Frame rotation through Euler Angles

- Perform a rotation ψ around z-axis, $\bar{\boldsymbol{x}}^{II} = Z\bar{\boldsymbol{x}}$:

$$\begin{bmatrix} \bar{x}^{II} \\ \bar{y}^{II} \\ \bar{z}^{II} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} \tag{2.10}$$

By combining the three equations, it is possible to obtain the final comp-lessive matrix required to perform the framework rotation:

$$\boldsymbol{x} = (X\ Y\ Z)\ \bar{\boldsymbol{x}} = M\bar{\boldsymbol{x}} \tag{2.11}$$

where $M(\phi, \vartheta, \psi)$ is the rotation matrix function of the three Euler Angles.

The selection of the sequence of rotations defines the M matrix. Considering each possible combination, there will be 12 possible sets of Euler angles. In aeronautical applications, the typical sequence is ZYX which means to perform a first rotation around z-axis, then a rotation around y-axis and the last rotation around x-axis, the angles are typically called: *Roll*, *Pitch* and *Yaw*.

Quaternion formulation can be selected alternatively. This representation uses four parameters to univocally select an orientation for a point in space. They are formally defined as the vector $[w, x, y, z]$ [8]:

$$q = w + xi + yj + zk \tag{2.12}$$

where $w, x, y, z \in \Re$, and the norm of the quaternion is 1:

$$w^2 + x^2 + y^2 + z^2 = 1 \tag{2.13}$$

According to the definition, it is possible to call $q_w = w$ the scalar part of the quaternion, and $q_v = (x, y, z)$ the vectorial one.

Robot controllers often require the use of quaternions to define orientations. Considering the Euler sequence ZYX, where the angles in sequence are ψ for z-axis, ϑ for y-axis and ϕ for x-axis, the quaternion is obtained as:

$$\begin{cases} w = \cos(\frac{\phi}{2})\cos(\frac{\vartheta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\vartheta}{2})\sin(\frac{\psi}{2}) \\ x = \sin(\frac{\phi}{2})\cos(\frac{\vartheta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\vartheta}{2})\sin(\frac{\psi}{2}) \\ y = \cos(\frac{\phi}{2})\sin(\frac{\vartheta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\vartheta}{2})\sin(\frac{\psi}{2}) \\ z = \cos(\frac{\phi}{2})\cos(\frac{\vartheta}{2})\sin\frac{(\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\vartheta}{2})\cos(\frac{\psi}{2}) \end{cases} \tag{2.14}$$

Mind that for each sequence of Euler Angles, the law that links them to quaternions is different. All the 12 equations are reported in [7].

2.2 Trajectory Planning

Task execution for manipulators is commonly controlled through motion planning algorithms, called *trajectory planning*. This topic is widely studied in robotics because the selection of the proper planning strategy can substantially improve the execution of the task [9]. Planning consists in generating a time-sequence of points or positions that must be executed either by the EE or by the joints.

According to the provided path description, these algorithms can set a:

- *Point-to-point motion* or position control, where only starting and goal positions are provided, and the objective is simply to reach the end position regardless of the followed trajectory;

- *Path motion* if the robot EE motion is specified with a defined sequence of points parametrized in time.

In [fig. 2.8] the two different approaches are reported with a graphical representation of the target motion. In the first case, the control strategy simply imposes the three points to be reached and then the robot controller chooses the proper path that must be performed; in the latter case the input is exactly the red 2D trajectory that must be followed.

The goal of trajectory planning control strategies is to generate a reference trajectory for the robot controller, which is a time-sequence of

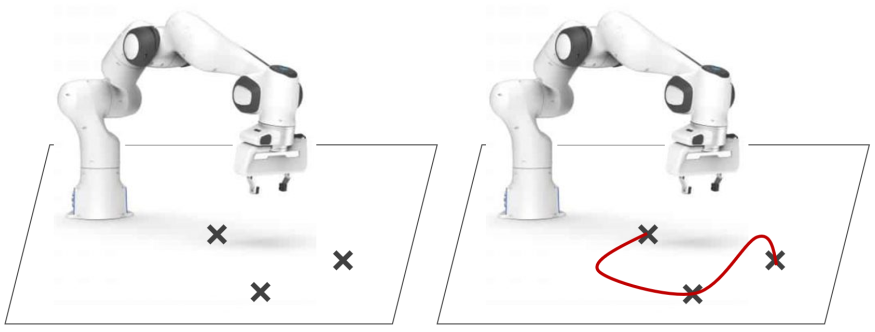


Figure 2.8: Position Control on the left, where only three intermediate points are given VS Path Control on right, where all the path is provided (red)

positions, velocities and accelerations that map the desired trajectory. Planning can be done either in joint space or in task one (Cartesian space), according to the movement that must be executed. For applications like welding, sealing or laser cutting, the joints of the robot can move almost freely in the workspace, with limitations only for singularity positions and physical spatial constraints, while proper planning for the TCP is imposed in cartesian space.

Trajectory planning algorithms can be schematized considering the planning steps and input parameters of the activity. In case of task space tracking, the operational space trajectory must be converted into a joint motion signal through the use of IK such that the robot motion control system will generate the torques and forces needed to activate motors [fig. 2.9].

Instead, joint space planning results in a faster motion because directly creates the signal $q_{out}(t)$ that must be fed into motion controller but it is not an optimal solution for precise path tracking since it describes a joint motion.

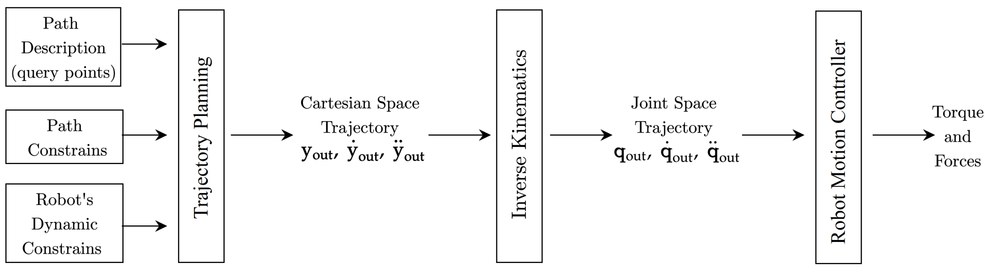

Path Description (query points)
Path Constrains
Robot's Dynamic Constrains
Trajectory Planning
Cartesian Space Trajectory
$y_{out}, \dot{y}_{out}, \ddot{y}_{out}$
Inverse Kinematics
Joint Space Trajectory
$q_{out}, \dot{q}_{out}, \ddot{q}_{out}$
Robot Motion Controller
Torque and Forces

Figure 2.9: Steps of trajectory (planning activity)

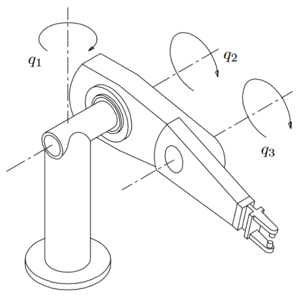


Figure 2.10: Joints on a manipulator. Source [5]

2.2.1 Joint Space Control

Joint space planning maps a set of positions for the TCP by controlling joint ones. This solution is preferrable when dealing with motion near possible manipulator singularities, or when a certain joint movement must be described. Example applications can be found in point-to-point motion or in motion through a sequence of points, where joints starting, intermediate and final positions are specified and the planning algorithm controls the interpolation and time execution in joint coordinates between one point and another.

Joint space motion is characterized by computation of joint positions of the robot, traditionally referred to with the variable q. A simple representation is proposed in [fig. 2.10] where it is possible to observe a 3 DOF manipulator with pure rotational joints.

Considering a more general open-kinematic-chain manipulator with n-DOF, characterized by n joint coordinates: $\boldsymbol{q} \in \Re^n$.

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{2.15}$$

where each coordinate is related to the movement generated by the physical link, revolute joints q will be represented by a rotation, while prismatic joints by translations.

Joint space trajectory planning algorithms interpolate the $\boldsymbol{q}(t)$ positions while respecting the imposed constraints, which can be physical limits of the manipulator (velocities, accelerations) or objective constraints imposed by the chosen interpolating function (i.e. Jerk minimization [10, 11], Time minimization etc.).

Point-to-Point Motion

The simplest planning algorithm is the *point-to-point motion.* The joints have to move from a starting configuration $\boldsymbol{q}(t_0) = \boldsymbol{q}_0$ to a final one $\boldsymbol{q}(t_f) = \boldsymbol{q}_f$ in a certain time interval t_f while respecting the imposed constraints.

Cubic polynomial interpolation algorithm is used to minimize the energy dissipated on each motor. The joint movement is obtained through a third-order polynomial function, such that the velocity profile is represented by a parabola:

$$\begin{cases} q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \end{cases} \tag{2.16}$$

to characterize the motion law, 4 constraints must be imposed: initial and final joint position and velocity. Higher order polynomials, such as *fifth order polynomial* are commonly used to impose also the acceleration limits.

Another common algorithm for interpolation is the *trapezoidal velocity profile.* This method is preferred when the motion must be executed at an imposed cruise velocity level, with a specific acceleration value during only starting and final stages [fig. 2.11].

Trapezoidal velocity profile is defined as:

- Acceleration:

$$\ddot{q}(t) = \begin{cases} \ddot{q}^+_{max}, & t \in [t_0, t_1] \\ 0, & t \in [t_1, t_2] \\ -\ddot{q}^-_{max}, & t \in [t_2, t_f] \end{cases} \tag{2.17}$$

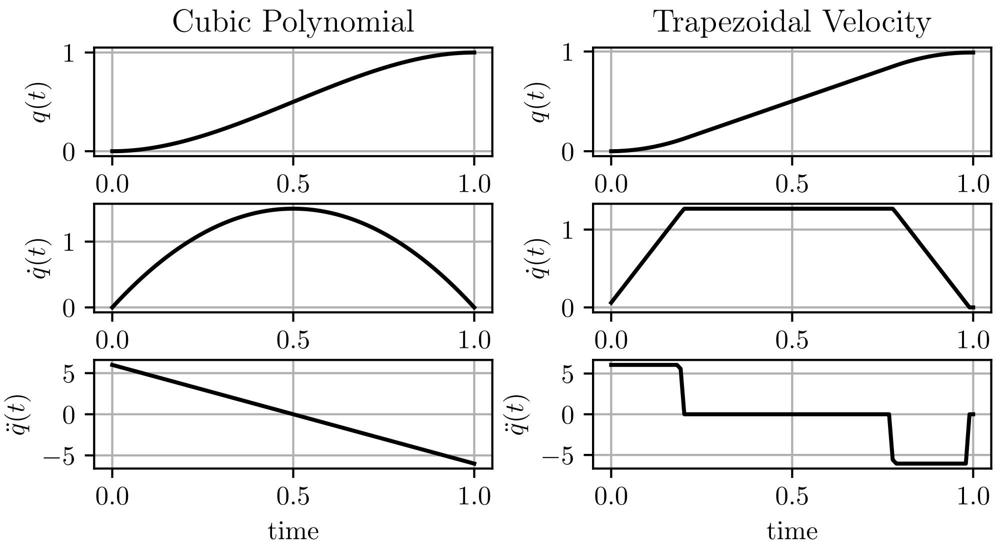
- Velocity:

$$\dot{q}(t) = \begin{cases} \ddot{q}^+_{max}(t - t_0) + \dot{q}_0, & t \in [t_0, t_1] \\ \dot{q}_{max}, & t \in [t_1, t_2] \\ -\ddot{q}^-_{max}(t - t_2) + \dot{q}_{max}, & t \in [t_2, t_f] \end{cases} \tag{2.18}$$

- Position:

$$q(t) = \begin{cases} \frac{1}{2}\ddot{q}^+_{max}(t - t_0)^2 + \dot{q}_0(t - t_0) + q_0, & t \in [t_0, t_1] \\ \dot{q}_{max}(t - t_1) + q_1, & t \in [t_1, t_2] \\ -\frac{1}{2}\ddot{q}^-_{max}(t - t_2) + \dot{q}_{max}(t - t_2) + q_2, & t \in [t_2, t_f] \end{cases} \tag{2.19}$$

where at t_0 the motion is starting, at t_1 the acceleration step has ended, at t_2 starts deceleration and at t_f motion ends. The velocity of execution is identified by $\dot{q}_{max}$.



(a) Cubic Velocity Profile **(b)** Cartesian Robot

Figure 2.11: Comparison between cubic polynomial interpolation and trapezoidal velocity profile with null velocity at starting and ending conditions, $a_{max} = 6$, $t = 1$ and $q_f = 1$

Motion Through a Sequence of Points

More complex tasks require a path characterization with more than just two points. Intermediate points or sequences are necessary to describe with higher precision the motion in space, indeed this planning approach is used whenever there are obstacles to be avoided or when a particular trajectory must be performed. The intermediate points between the starting and goal position are called *via-points*. The higher the number of *via-points*, the higher the definition of the motion (i.e. curve characterization requires more points with respect to a straight line motion).

Provided a sequence of N points that must be reached by the manipulator at certain time instants, it is necessary to define an interpolating function Π, to provide a continuous motion between the via-points. Mathematically it must be used a $(N-1)$-order polynomial, but while N increases, several problems may arise:

- Initial and final velocities cannot be assigned;
- High order polynomials show unnatural oscillatory behaviours (*Runge's phenomenon*);

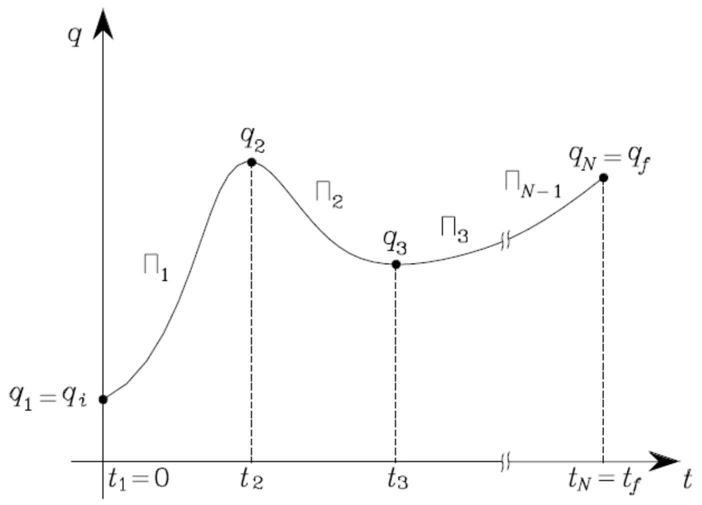


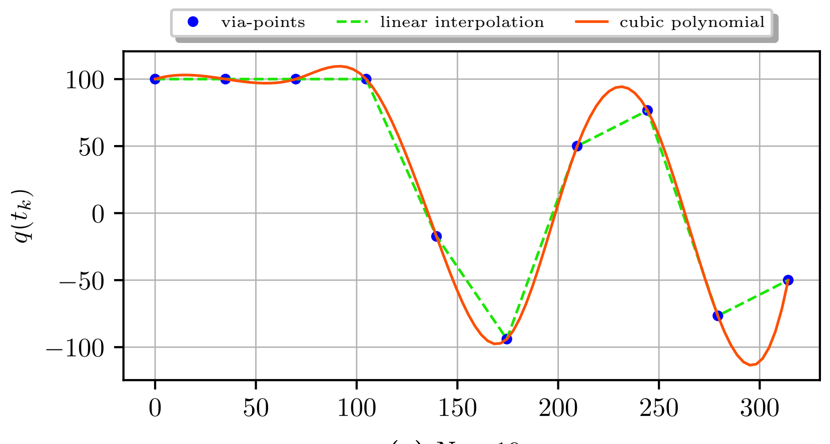
Figure 2.12: Via-points interpolation with $(N-1)$ polynomials. Source: [2]

- The solution of the system for defining polynomials coefficients results computational expensive;
- The polynomial definition strictly depends on the *via-points*, if one should be changed, all the coefficients must be re-computed.

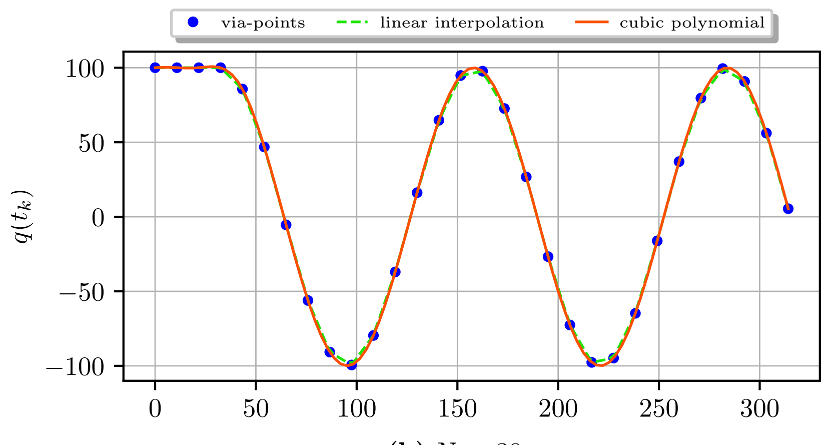
Moreover, whenever a dense sequence of points is provided, such as to describe a proper path definition, the *via-points* are very close one to another, resulting in a quasi-continuous trajectory. Especially in this situation, selecting a $(N-1)$-order polynomial is not advised. The best solution is to use $(N-1)$ polynomials of lower order to describe a motion between one point and the following one: Π_i, $i \in [1, N-1]$, [fig. 2.12].

Cubic polynomials are advised for maintaining a velocity continuity between one point and the following one, but they require a velocity characterization at each *via-point*. Also, lower order polynomials can be used, but they generate a non-smooth transition between one interval movement and another, and they lose the continuity in velocity.

Whenever *via-points* are very dense, the choice of the interpolating polynomial results less critical from the displacement point of view, because the movement to be executed in between is not large enough to show a characteristic difference in results [fig. 2.13], but the velocity profile will still show a non-smooth behaviour that will result in a non-optimal movement of the joints. Non continuous transitions can be the cause of vibrations during the execution of the task, especially if it requires a good precision. In [12] it is studied a trajectory generation algorithm that permits to achieve smooth transitions while controlling also joint positions.



(a) N = 10



(b) N = 30

Figure 2.13: Comparison between different interpolation methods when the number of *via-points* increases [$N = 10$ and $N = 30$]

2.2.2 Task Space Control

In *task space planning* the motion of the manipulator is mapped by controlling the motion of the EE. This control strategy is preferred whenever a precise movement of the TCP is requested, i.e. in precise geometry following applications, like welding and sealing.

Planning in operational space can be done with two main approaches:

- By performing interpolations between numerous *via-points* with the same method presented for the joint-space motion, typically using linear *micro-interpolations* with a point density equal to the robot frequency;

- Through generation of analytical motion primitives.

Interpolation has been exploited in [sec. 2.2.1]. In the same way it is possible to describe several *via-points* in the cartesian space, such that the movement of the EE is directly controlled.

Path-Velocity Parameterization

Path primitives are analytical functions used to precisely and analytically describe the motion of the EE. These functions characterize features of the path while providing a time law to determine the motion execution. According to the formal definition of path primitives, it is possible to refer to them through the equation:

$$\boldsymbol{p} = \boldsymbol{f}(s) \tag{2.20}$$

where $\boldsymbol{p}$ is a three-dimensional vector that describes the space position, $\boldsymbol{f}$ is a continuous vector function parametrized over the variable s, the natural coordinate, which represents the percentage of completion over all the path. If s increases, the execution along the cartesian path described by the function $\boldsymbol{f}$ proceeds [fig. 2.14].

The characterization in time of the motion execution is provided by the mathematical law that links the natural coordinate with time: $s = s(t)$. Typically, when $t = 0 \rightarrow s = 0$, while when $s = s_f = \textit{path length}$ the overall time requested to perform all the path is reached, $t = t_f$.

The execution velocity along the path depends on the time parametrization of the natural coordinate. Indeed:

$$\dot{\boldsymbol{p}} = \frac{d\boldsymbol{p}}{dt} = \frac{d\boldsymbol{f}(s(t))}{dt} = \frac{d\boldsymbol{f}}{ds}\frac{ds}{dt} \tag{2.21}$$

where $\frac{ds}{dt}$ represents the magnitude of the velocity vector in correspondence of the selected time instant, while $\frac{d\boldsymbol{f}}{ds}$ is the tangent vector to the path.

Traditionally a motion primitive provides a mathematical description of a precise motion, i.e. they are commonly used on humanoid robots to describe basic movements, such as walking or catching; they are then joined together to obtain a complex outcome [13]. A motion primitive describes the execution of a proper movement with related constraints, both in space and velocity. In [14] motion primitives are exploited to analyse the possible outcomes while combining three basic primitives. A proper analytical parametrization of each path is requested, and the overall motion is obtained as a sequence of these sub-paths.

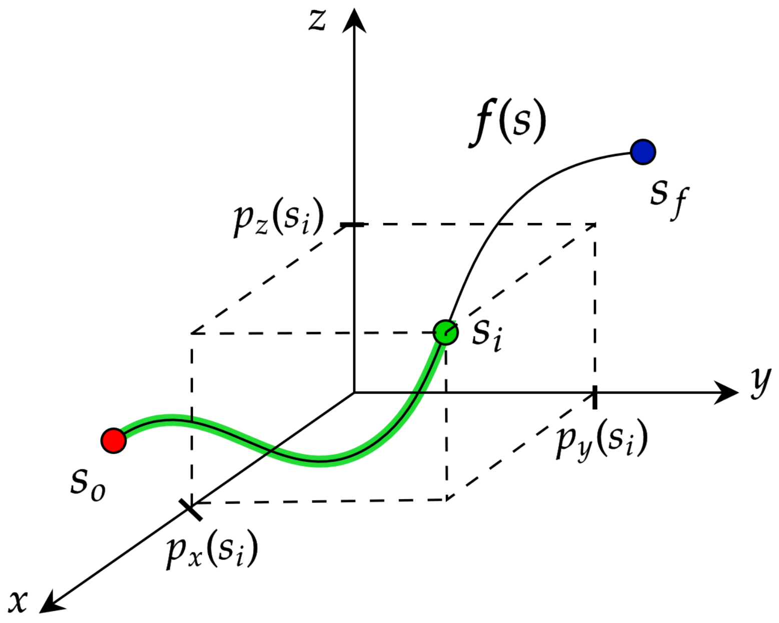


Figure 2.14: Completion of the analytical function $f(s)$, in *green* the percentage of path already executed, s_i is the actual position along the path

With the proper formulation, even one path primitive can be used to obtain a complex motion, without the need to subdivide the path in smaller intervals. Obviously, this approach can be selected only in certain situations, not for all the numerous cases of task planning. This specific analysis will be deepened in the next chapters of the thesis, since it recalls the selected method for planning the task execution.

2.3 Robotic Industrial Processes

Now that the basic concepts of trajectory planning have been explained, it is possible to analyse several methods already implemented in industrial applications that require a precise movement of the EE.

Sealing procedures include numerous types of applications, from silicone sealing to welding and gluing. Traditionally those activity are performed by a human operator, but automatization is requiring novel approaches with the use of robot manipulators. Welding procedure can be the main reference application for the sealing tasks, as it has already several automatized processes, but it is also possible to analyse other similar applications, like Fused Deposition Modeling (FDM) 3D printing or laser cutting.

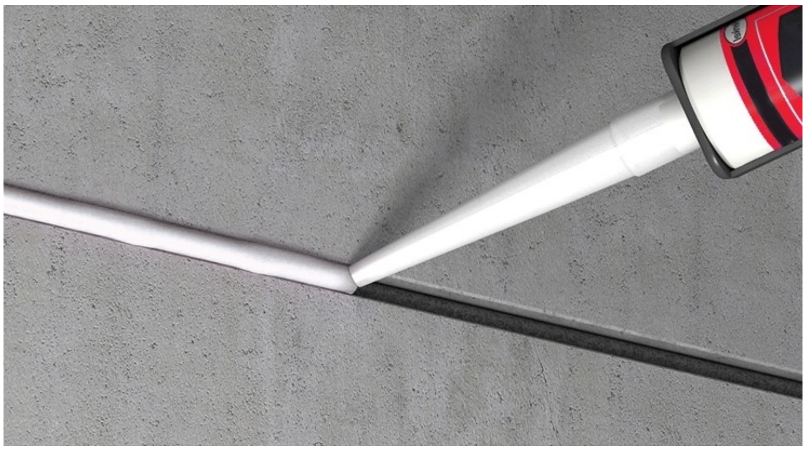


Figure 2.15: Human performing a silicone deposition

2.3.1 Sealing Applications

Sealing is a traditional manual operation that requires the deposition of a ribbon of sealant material that will permanently join separate pieces [fig. 2.15]. This operation requires a precise path tracking to deposit the material in the correct zones, and a proper characterization of the motion to generate a correct deposition.

Skillful manual operators can achieve a good quality level by slowly performing the task, robots can achieve a higher quality deposition in a faster way.

The main problems of silicone deposition are related to:

- Velocity of deposition;
- Caulking gun orientation during deposition.

The first problem for human operators is partially solved through a trial and error movement, for robots it is the object of the planning algorithm, that must characterize and optimize the execution velocity.

The gun orientation changes for the human and robotic case, because the two cases have different features and control strategies. A human cannot have the precision in position, orientation and velocity of a manipulator, so it must perform a deposition in a way that both the path and the sealant material ribbon can be semi-optimal. To achieve that, a proper deposition of the sealant material can be obtained sliding the gun nozzle on the path to be executed while keeping the gun with a certain inclination, such that the nozzle cannot be obstructed and the material can properly flow.

Instead, robots do not need to slide on the path since an accurate positional controller permits to perform a sealing application moving above the deposition plane with a fixed offset. Since the offset prevents the nozzle to be clogged, the caulking gun can be maintained vertical during all the execution. This way, the deposition can be done similarly to a FDM 3D printing process, with a constant and precise vertical distance from the plane and a completely perpendicular orientation.

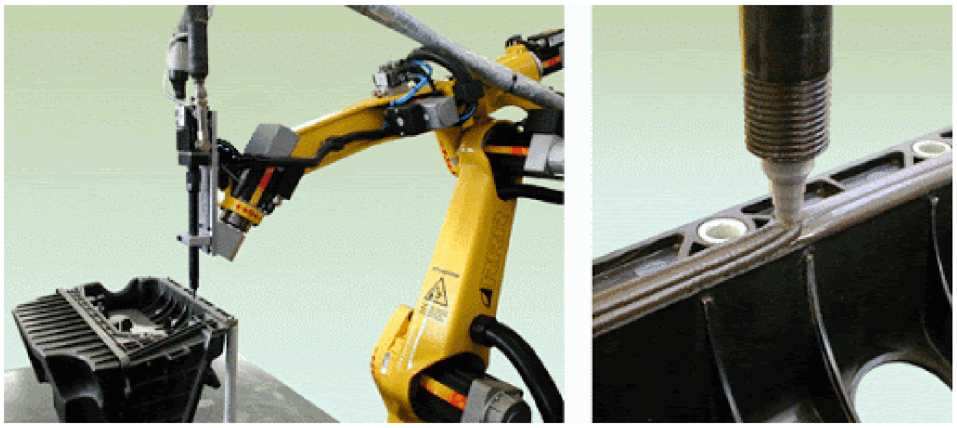


Figure 2.16: Gluing application. Source: www.rrrobotica.it/incolla.htm

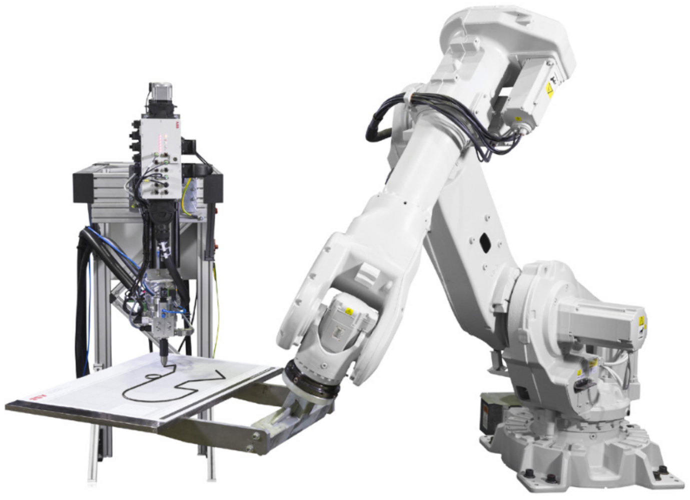


Figure 2.17: Innovative sealant deposition. Source: abb.com/robotics

A gluing industrial application is reported in [fig. 2.16], while an alternative version which uses an improved sealing and gluing process is reported in [fig. 2.17]. In the latter example, the manufacturer employed

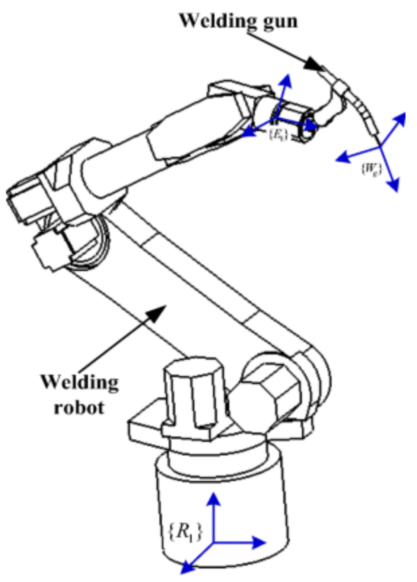


Figure 2.18: Welding Robot scheme. Source: [15]

an industrial robot to set the plane height and cartesian position with respect to the fixed sealant extruder.

2.3.2 Welding Applications

Welding is a traditional mechanical procedure used to join two or more metallic or thermoplastic parts. The pieces are heated up at the edges that must be joined, and then the connection is sealed with the cooldown of the material. The execution of this activity requires the use of arc welding torches that are fixed at the EE and that must be precisely moved along the joining edge. The cartesian movement must be described for the coordinate framework of the tip of the torch, the TCP and not for the robot flange reference one [fig. 2.18].

A known robotic algorithm for welding applications is the *Descartes Algorithm* [16]. This algorithm finds the proper movement for the manipulator by checking a timing constraint for the EE and joints positions. Through this algorithm it is possible to avoid motion redundancy by imposing a cost function on the joint positions, even if the motion is controlled in cartesian space. The planning strategy provides a sequence of TCP poses without any velocity or acceleration information and then characterizes the motion in time by introducing a *timing constraint*.

This approach is proper for large displacements in the workspace that need a time optimization for the joints movement, but it does not assess

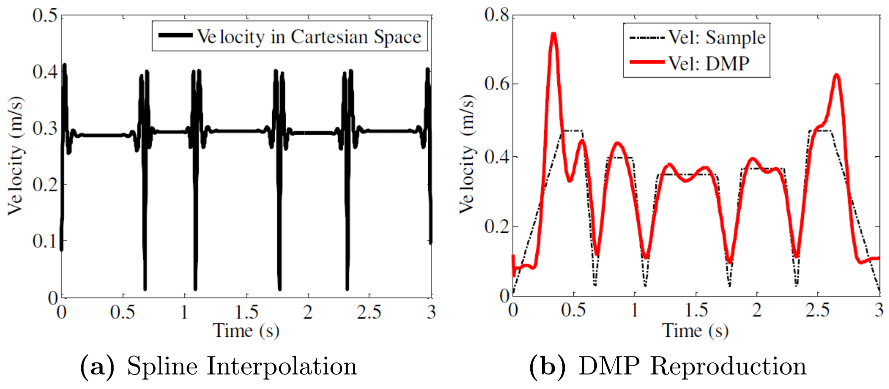


Figure 2.19: Comparison between velocity outputs obtained for Spline interpolation and DMP motion with a high number of *via-points*. Source: [19]

the problem related to the point velocity execution. In the scope of this thesis, this method does not provide any advantage because the sealing task will be executed on a collaborative robot, so a collaborative environment must be considered, in correspondence of a reduced workspace area and on a predominant 2D trajectory such that the robot internal controller will automatically optimize joints movements.

In [17] it has been studied an optimal planning for welding applications, through an analysis on the task execution that minimizes the time and interpolates cartesian *via-points* with B-spline method. This approach imposed a welding torch velocity fixed along the path and did not include a proper feature characterization, as this thesis aims to do. Another general approach which uses B-spline interpolation is studied in [18], this analysis is not specifically made on welding problems but can provide an interesting online planning strategy.

Considering a more general robotic background, the interpolation method has been proved to not always be the best choice for path planning. A comparison between a motion primitive algorithm (in this case DMP) and a traditional spline interpolation is reported in [19]. The result shows that while the cartesian tracking execution is similar, the velocity behaviour differs and results less smooth and acceptable for a robotic manipulator [fig. 2.19]. According to these outcomes, it is in the interest of this thesis to focus on the more robotic friendly method, that would compute smooth transitions during execution.

Chapter 3

Dynamical Movement Primitives

This thesis is built over a very general purpose framework for trajectory planning, which is called DMP [20]. This method has been introduced by Ijspeert, Nakanishi and Schaal in 2002 [21], with the aim of approximating a movement of a humanoid robot.

The algorithm has been developed in order to solve some disadvantages of other techniques that try to learn and reproduce a motion, such as the increasing difficulty in learning when the number of DOF rises and the large time needed for tuning the model parameters [4]. The main reason behind the choice of this control structure is related to the large potential of this framework in collaborative environments, which allows to easily implement features like obstacle avoidance and disturbance rejection [22, 23], while performing a precise tracking of the taught path.

DMP framework approximates a path, or a simple movement, in a physical manner (i.e. with velocity continuity and intrinsically smoothing sharp edges). The intrinsic physical behavior is provided by the DMP definition itself, in a nutshell, a taught motion is performed by a punctual mass-spring-damper critical system, which is carried from an initial state to a goal position by means of a non-linear forcing term. This mathematical approximation provides high tracking potential, while filtering path features that in reality a robotic arm would not be able to perform (i.e. noise or small discontinuities). This definition can be easily applied to very simple single DOF systems to complex multi-DOF robotic arms.

There are two main types of DMP which are characterized by the definition of the attractor system. We can distinguish between point attractor systems, that bring the state of the robot from a position to another one, and limit cycle attractors when one wants to perform a periodic movement (as in the oscillators) [4]. Due to the nature of the problem at hands, only on the point attractor systems are considered in this thesis. This model is used to directly map the EE positions and velocities in cartesian space, without the need to convert them into motor behaviours in joint space. The computed trajectory from the DMP can be transformed through IK into joint motions or directly fed into the robot controller, using its built-it positional tracker.

Mathematics and researchers have been performing further studies over DMP to examine in depth other applications in robotic field and also to increase the power of the method. Several studies that have been used into our thesis will be reported later and also the novel contribution to the method.

3.1 DMP Dynamical System

DMP are modelled through a "non-linear attractor system" [4], where several fundamental features are pursued through the model:

1. Learnable point attractor;
2. Autonomous system;
3. Reduced number of open parameters, with simple learning;
4. Spatial and temporal scaling;
5. Error coupling.

All of this points will be explained when recalling the mathematical formulation provided by Jispeert, Nakanishi and Shaal [4, 21], where there will be showed also their powerfulness.

3.1.1 Mathematical Formulation

As previously said, DMP are based on the simple dynamical system of a unitary mass in a spring-damper configuration moved through an external forcing action:

$$\tau \ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f \tag{3.1}$$

which can be written in a state-space notation:

$$\begin{cases} \tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f \\ \tau \dot{y} = z \end{cases} \tag{3.2}$$

where position, velocity and acceleration are expressed through y, $\dot{y}$, $\ddot{y}$, then z is the state-space term, τ is the time-constant and α_z, β_z are constants values tuned to provide a stable critically-damped response of the dynamical system.

The parameters α_z, β_z were selected in accordance with the analysis performed in [4, 21]. This structure is currently exploited through a mono-dimensional path, but it can be easily extended to 2D or 3D simply by describing the dynamical system for each spatial dimension of interest.

The second order differential equation (3.2) is characterized by two external actions that need to be clarified: the forcing term, f, and the goal position, g, which acts as a point attractor.

The goal position, g, is a constant fixed value, that represents the final state that must be reached at the end of the simulation. The definition of the target point is necessary to describe a system dynamics that converges to the requested position even when the acting force, f, is null. The forcing term is instead a nonlinear function:

$$f(t) = \frac{\sum_{i=1}^{N} \Psi_i(t)\ \omega_i}{\sum_{i=1}^{N} \Psi_i(t)} \tag{3.3}$$

which represents a weighted sum of N Gaussian basis functions, $\Psi(t)$, where ω are their weights.

At this point the DMP are still characterized by a non-autonomous function, since there is an explicit time-dependance. This causes problems in the coordination of multiple DOF, which is not allowed by the time dependency. A Canonical System (CS) is then introduced (3.4), needed to decouple the forcing function from time:

$$\tau \dot{x}(t) = -\alpha_x x(t) \tag{3.4}$$

which represents a decreasing exponential that tends to zero as the state y gets closer to the goal position, so for $y = y_0$, the starting position, we have $x(t_0) = 1$ and when $y = g$, then $x(t_{sim}) \simeq 0$. The parameter α_x must be tuned to define the decay rate of the exponential. The standard value is here considered: $\alpha_x = 1$ [fig. 3.1].

Thanks to the CS, it is possible to define a time-independent forcing function, rearranging (3.3):

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x)\ \omega_i}{\sum_{i=1}^{N} \Psi_i(x)} \cdot x \cdot (g - y_0) \tag{3.5}$$

This definition of f exploits the scalability in time and space of the DMP, because the dependence on the x variable introduces the modulation in

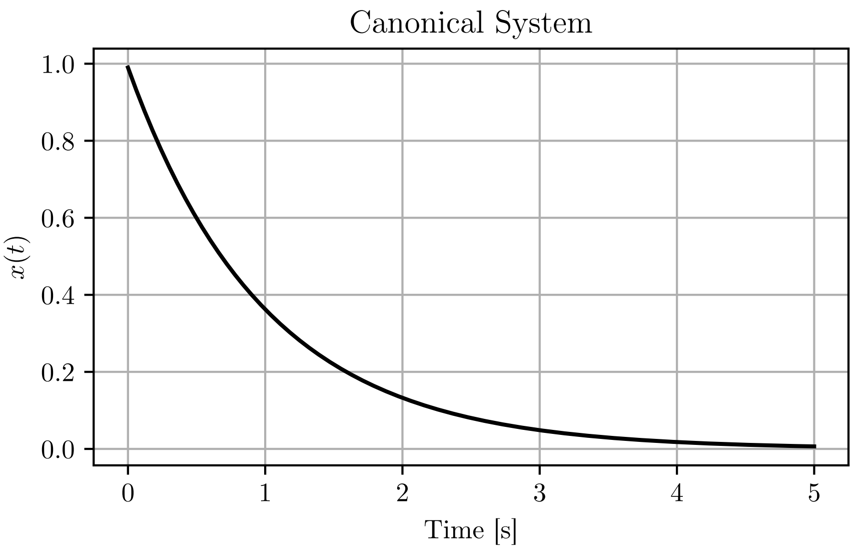


Figure 3.1: Exponential decay of the CS [$\alpha_x = 1, \tau = 1$]

time of the force as we get closer to the goal position, while the space scalability is provided by $(g - y_0)$.

The N Gaussian basis functions, Ψ, are defined as functions of the x coordinate:

$$\Psi_i(x) = exp\left(-\frac{1}{2\sigma_i^2}\cdot (x - c_i)^2\right) \tag{3.6}$$

where the shape of the Gaussian kernels depends on the center, c_i, and on the width, σ_i. It should be noticed that the centers are equally spaced in time, so according to the CS there will be fewer kernels at the beginning of the movement and denser ones closer to the end. This behaviour is shown in an example in [fig. 3.2].

The Gaussian centers, c_i, should be equally spaced in time, so their characterization along the x parametrization is obtained through the CS. Instead, the variance of the Gaussian basis functions, σ_i, is defined accordingly with the equation (3.7), using a trial and error approach.

The distribution selected in this thesis is slightly different from the standard one, because it has been chosen to use numerous Gaussian functions, with a narrower shape.

$$\sigma_i = 2\sqrt{\frac{c_i^{1.8}\cdot a_x}{N^{1.5}}} \tag{3.7}$$

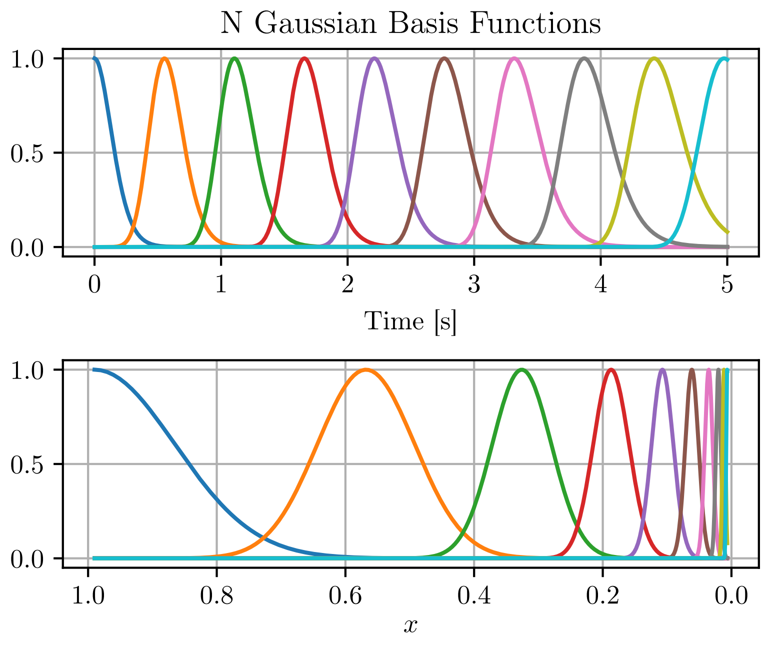


Figure 3.2: Example of 10 kernels in time and x domains, for a trajectory that lasts 5 s (in the CS are visible denser Gaussian functions when $x = 0$ is approached)

The DMP framework requires tuning of other parameters, which are α_z, β_z, α_x, σ_i and τ. For the purpose of this research some values already studied and optimized has been selected [4, 21, 24], that have been collected in [tab. 3.1], while others have been adapted to the proposed study.

Parameter	**Value**
α_z	25
β_z	$\alpha_z/4$
α_x	1
τ	1

Table 3.1: DMP and CS system coefficients (in the standard formulation)

3.2 Trajectory Learning: Locally Weighted Regression

The general framework of the DMP has been described, but the forcing term still needs to be defined (3.5), because its shape is characterized by the weights, ω_i, of each Gaussian kernel, whose values strictly depend on the trajectory that must be reproduced.

Using the traditional definition of DMP, f is linearly dependent on the Gaussian weights, ω_i, as shown in equation (3.5); this permits to find ω_i using several possible learning algorithms. Calinon and Lee tested and compared several methods which can be used to learn the weights [25].

The original framework uses supervised learning to define the weights that fit the imposed target trajectory that must be reproduced. The most commonly selected method is Locally Weighted Regression (LWR) [4, 26], but for humanoid robots has also been used a Global Weighted Regression [27]. The LWR has the advantage of less computational burden since each kernel acts only closed to its center and so it is computed locally [fig. 3.3]. Instead, global regression considers all the motion together to compute each kernel weight. This last method requires less basis functions, at the cost of increased computational resources. In this study the originally proposed method with LWR is followed because it was mainly adapted to collaborative robot environments and it was proven to be very fast and with independent learning for each kernel [28]. Moreover, due to the algorithm definition, having sharp kernels which act locally is preferable.

To perform trajectory learning, the target path that must be reproduced is defined: $y_{demo}(t)$, $\dot{y}_{demo}(t)$, $\ddot{y}_{demo}(t)$, where $t \in [0, T_{demo}]$. This path can be either mono-dimensional or with higher-order dimensions because the DMP will evaluate a forcing action for each coordinate, and so will define characteristic Gaussian weights for each dimension. For the purpose of this research a two-dimensional trajectory has to be described, which can simply be done by describing y_{demo} on a XY cartesian plane.

Reversing the equation (3.3), the forcing term f_{target} is computed:

$$f_{target} = \tau^2 \ddot{y}_{demo} - \alpha_z(\beta_z(g - y_{demo}) - \tau \dot{y}_{demo}) \tag{3.8}$$

Now, the learning problem has to generate an approximated forcing term, f, as similar as possible to the target force, f_{target}.

LWR minimizes the following cost functions, which describe locally weighted quadratic errors [29], which in this case is needed to evaluate the

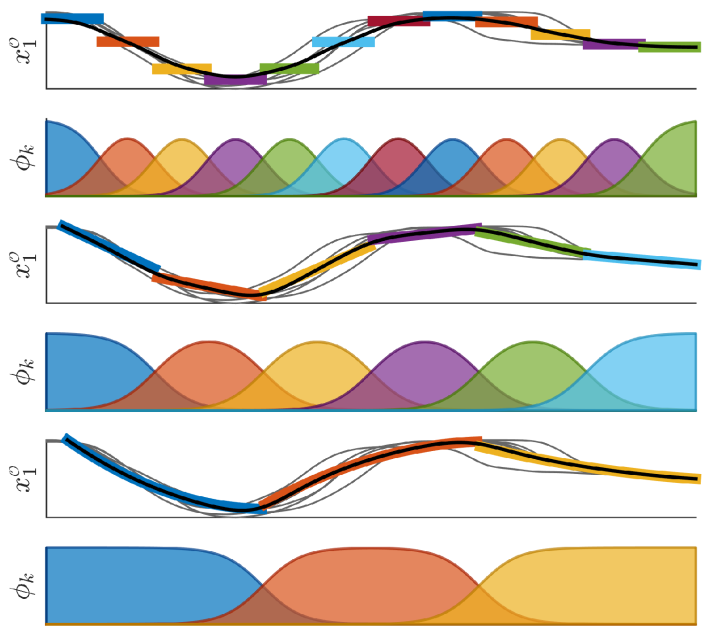


Figure 3.3: Local effect of the basis functions ϕ_k using LWR for a path x_1^0. Source: [25]

correct ω_i for each Gaussian kernel:

$$J_i = \sum_{t=1}^{N_{steps}} \Psi_i(t) \left(f_{target}(t) - \omega_i x(t)(g - y_0) \right)^2 \tag{3.9}$$

As a LWR, the solution can be computed as:

$$\omega_i = \frac{\boldsymbol{s^T} \Gamma_i \boldsymbol{f_{target}}}{\boldsymbol{s^T} \Gamma_i \boldsymbol{s}} \tag{3.10}$$

where:

$$\boldsymbol{s} = \begin{bmatrix} x(1)(g - y_0) \\ x(2)(g - y_0) \\ \vdots \\ x(N_{steps})(g - y_0) \end{bmatrix} \quad ; \quad \Gamma_i = \begin{bmatrix} \Psi_i(1) & & & 0 \\ & \Psi_i(2) & & \\ & & \cdots & \\ 0 & & & \Psi_i(N_{steps}) \end{bmatrix} \tag{3.11}$$

and the force vector is:

$$\boldsymbol{f_{target}} = \begin{bmatrix} f_{target}(1) \\ f_{target}(2) \\ \vdots \\ f_{target}(N_{steps}) \end{bmatrix} \tag{3.12}$$

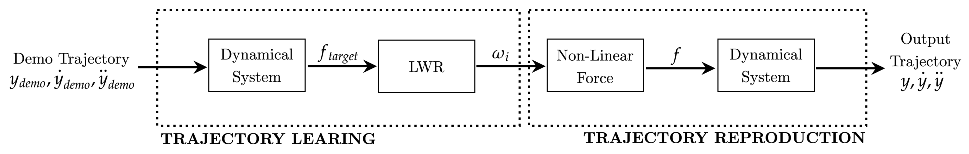


Figure 3.4: Scheme of the process for learning and evaluating the desired trajectory, y, starting from the demonstrated one, y_{demo}

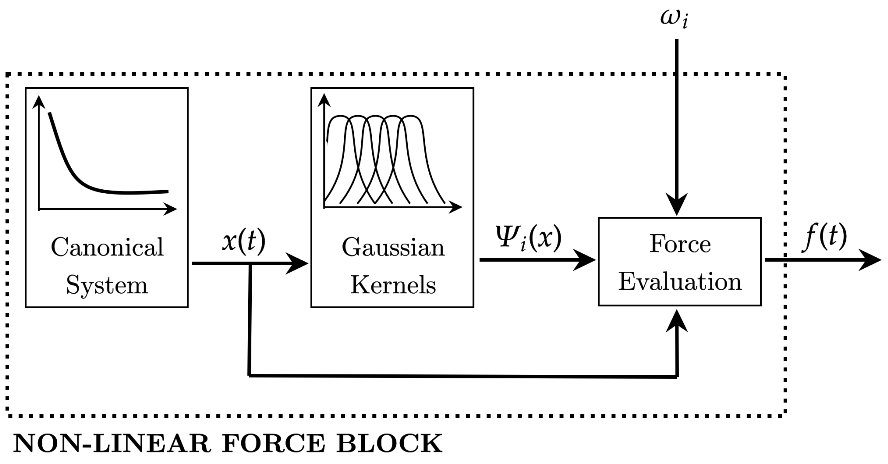


Figure 3.5: Insight on the "Non-Linear Force" block

Once the kernel weights are evaluated, the actual forcing term can be computed. This will drive the artificial dynamical system using equation (3.5).

A graphical recap of the steps needed to get the output trajectory that imposes robot movement is reported in [fig. 3.4] and the zoom on the non-linear force block is reported in [fig. 3.5]. In the presented system, only one demonstrated path is fed into the trajectory learning algorithm; this solution can be used in cases in which the desired trajectory is well known (i.e. without errors). Whenever the demonstrated reference movement is provided by human motion and interaction, it is very likely that smoothing of noises is performed; this can be simply done through averaging during LWR, without needing another complex algorithm for filtering. This process is carried out in [28], where many acquisitions are averaged to compute the final output [fig. 3.6]. In the represented case, learning is performed through Receptive Field Weighted Regression, which is another possible learning algorithm.

[fig. 3.5] reports the steps needed to evaluate the actual forcing action

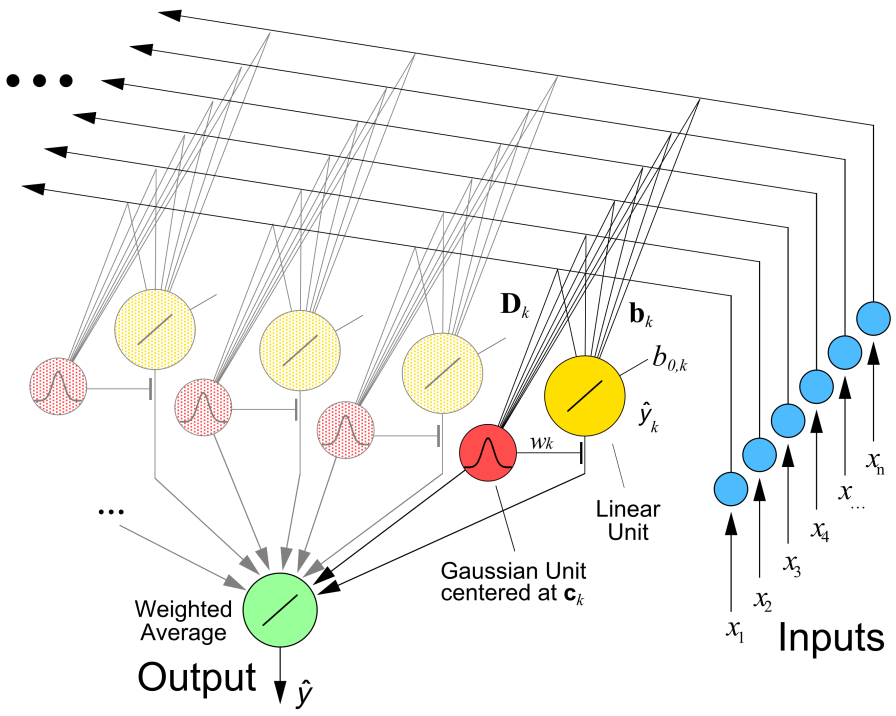


Figure 3.6: Example of averaging procedure applied to multiple input acquisitions: the output is a weighted average. Source: [28]

that will drive the system. All the fundamental passages that characterize DMP are shown, such as the evaluation of the CS, needed to decouple the kernels from time, and the Gaussian basis functions computation. It is fundamental to recall that all the coupling terms and parameters must be defined before starting the learning procedure, as reported in [tab. 3.1].

The τ parameter represents the temporal scaling factor, because as it changes from the standard value, $\tau = 1$, the execution of the trajectory is performed at a different new velocity threshold. From experimental analysis it is shown that using an lower value of τ means increasing the mean velocity threshold, and so performing the path in a faster way; simultaneously, if τ increases, then the path is performed in a longer time.

3.3 DMP Coupling

The powerfulness of DMP in collaborative robotics environments is associated to the ability to couple the output DMP trajectory $(y, \dot{y}, \ddot{y})$ with the actual robotic movement $(y_a, \dot{y}_a, \ddot{y}_a)$. A block scheme that implements this strategy is proposed in [fig. 3.7], while the mathematical formulation

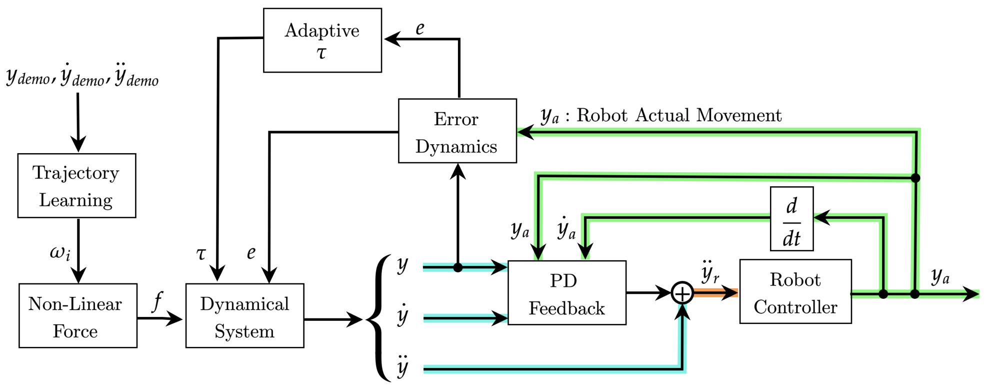


Figure 3.7: Scheme of the DMP framework with coupling system

is shown below.

Basically, an error coupling term is modelled using the measurable difference between the actual movement and the numerically computed one. The error dynamics is evaluated through a decreasing exponential, to represent a low-pass filter on the tracking error:

$$\dot{e} = \alpha_e(y_a - y - e) \tag{3.13}$$

The knowledge of the coupling error is used to introduce an external forcing action on the dynamical system (3.2):

$$\begin{cases} \tau\dot{z} = \alpha_z(\beta_z(g - y_0) - z) + f + C_t \\ \tau\dot{y} = z \\ C_t = k_t e \end{cases} \tag{3.14}$$

The standard formulation expresses the time scaling parameter as a simple value to modify the execution time. In this case instead, τ is function of the error dynamics:

$$\tau = 1 + k_c e^2 \tag{3.15}$$

Related to coupling, several studies have been made to define the proper formulation to permit a path execution in presence of physical disturbances on the EE. The original formulation studied by Ijspeert, Nakanishi and Schaal [4] considers to correct the actual robot movement by evaluating the reference robot acceleration through a feedback system:

$$\ddot{y}_r = k_p(y - y_a) + k_v(\dot{y} - \dot{y}_a) \tag{3.16}$$

The analysis performed on a Yumi collaborative robot [22], showed that a better evaluation of the reference acceleration of the EE permits to perform the movement with lower gains, and so with reduced motor torques. To achieve all that, while maintaining the coupling properties, they expressed $\ddot{y}_r$ in a PD feedback and feed-forward regulator:

$$\ddot{y}_r = k_p(y - y_a) + k_v(\dot{y} - \dot{y}_a) + \ddot{y} \tag{3.17}$$

where the acceleration computed by the DMP, $\ddot{y}$, is the feed-forward contribution.

3.4 Reference Velocity

The main limitation found in DMP framework, concerns the selection of the velocity reference. The learning algorithm tries to replicate in a smooth way the taught path at constant speed, except in correspondence of sharp edges and discontinuities, at which the velocity drops to match the physical limitations of the dynamical system. This behaviour does not show problems for the classical pick and place tasks, where a precise path execution is not the priority, but it cannot be applied in trajectories that need a precise velocity tracking of all the path.

There are several approaches in literature [24, 30] that deal with velocity discontinuities, but they only introduce a control action on the starting and ending velocities, to avoid velocity discontinuities in correspondence of the joining point between one motion primitive and another. These approaches are a proper solution for the cartesian planning with multiple path primitives that are combined to obtain a complex motion. Basically, the complex path is subdivided into sub-motions that will be performed in sequence. In this way, the system can learn separately the numerous trajectories, and then they can be joined together.

The solution proposed in [24] uses the standard DMP framework with a smoothing feature to join the different trajectories, but also introduces a third-order dynamics; while [30] keeps the second-order dynamics but introduces overlapping kernels at the joining points.

In this thesis, the problem has been approached in a different way, not by changing the DMP structure itself, but by modifying the demonstrated path. Basically it is considered to have a complex path that contains several features and that planning is done in one shot (i.e. without dividing it in smaller basic-feature parts), such that just one movement primitive framework is requested. This approach permits to avoid a motion library and separate learning of each path feature, that would require multiple

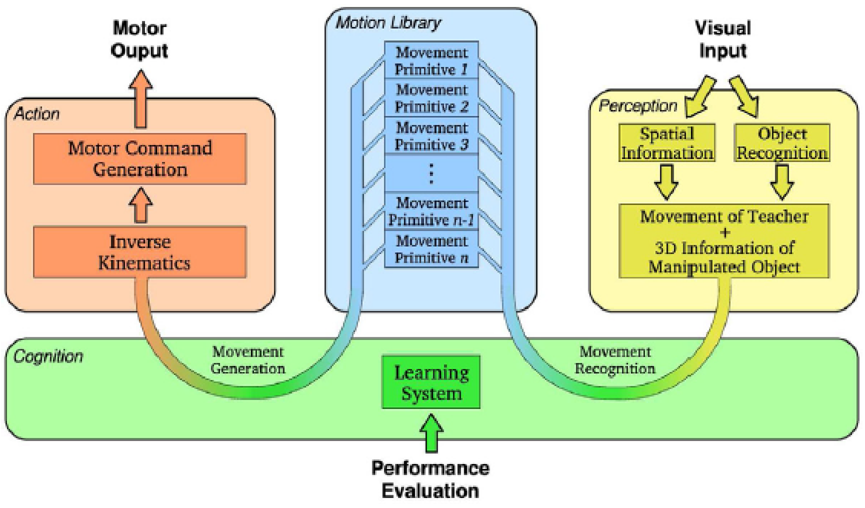


Figure 3.8: Complete scheme from path recognition to generation of the movement signal using a motion library of possible path features. Source: [31]

training activities. This more traditional approach in a complete framework is reported in [fig. 3.8], while the selected framework with single motion primitive and velocity reference is reported in [fig. 3.9]. Obviously, the more complex the trajectory, the higher the effort requested when using only one single DMP, but a proper tuning of the parameters permits to execute every path with no tracking problems.

The proposed approach does not sub-divide the path in smaller elements but analyses it completely while imposing a variable reference velocity along all the trajectory. An example of this procedure is shown in [fig. 3.10], where it is possible to see that the velocity output from the DMP framework is shaped to be not constant but it varies accordingly to the path features, while the standard DMP framework generates an almost constant velocity profile.

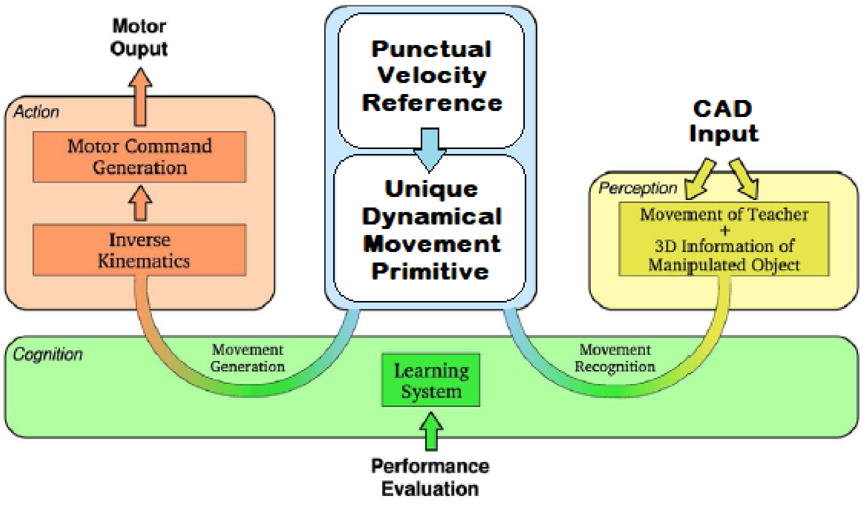
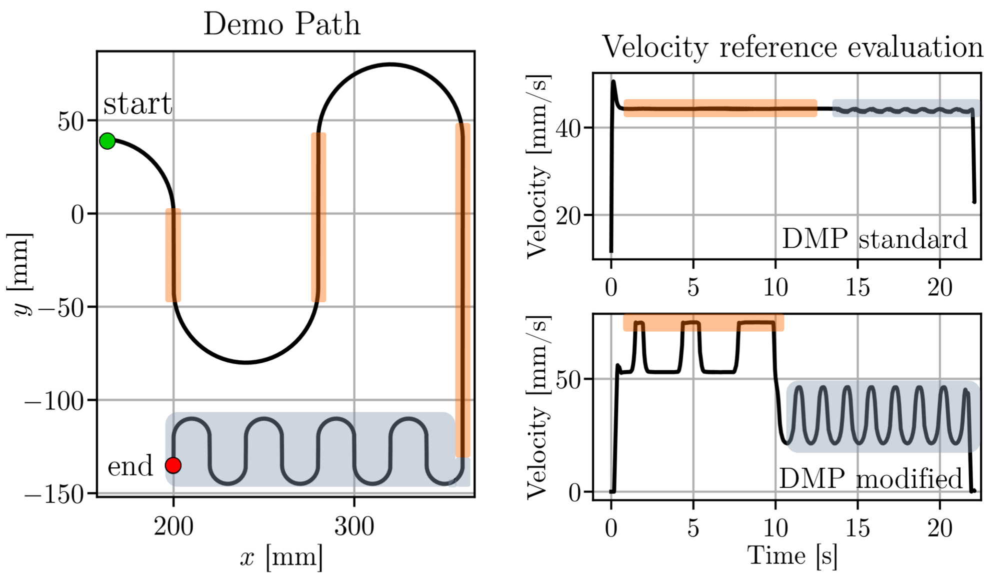


Figure 3.9: Complete scheme from path recognition to generation of the movement signal in this thesis framework



(a) Path to be performed

(b) Velocity profiles

Figure 3.10: Comparison of possible velocity executions of the same path (a): in (b)-upper graph standard DMP velocity output,
in (b)-bottom graph the proposed DMP approach, with variable reference velocity

Chapter 4

Experimental Setup

This chapter explains and analyses the test-bed used in all the physical experiments. The task specifications required the development of a trajectory planning algorithm for a collaborative robot, so the research focuses more on the algorithm and on its implementation, while only minor modifications to the provided experimental setup are made.

The test-bed can be sub-divided into four main blocks [fig. 4.1] that interact with each other through communication signals managed by the workstation. All the algorithms that generate the trajectory and control the robot run into a Robot Operating System (ROS) environment. The robot used for the tests is a ***Franka Emika Panda***, a collaborative robot with 7 DOF, which mounts at the EE a commercial caulking gun, a ***Makita DCG180***, which is connected at the manipulator through a custom flange created ad-hoc [fig. 4.2].

All the physical arrangements and some useful building instructions for the development of the 3D printed flange and the motorized system that controls the sealant deposition, are available in [app. A].

4.1 Franka Emika Panda Robot

The robot provided for physical experiments is a Franka Emika Panda [fig. 4.3]. This collaborative redundant manipulator is often selected in numerous human-robot applications thanks to some peculiar features:

- Versatility: it can work in environments that require precision, force application and sensitive handling;

- Compact design and redundancy: its small dimension is not a limit since it can reach large workspaces [fig. 4.4];

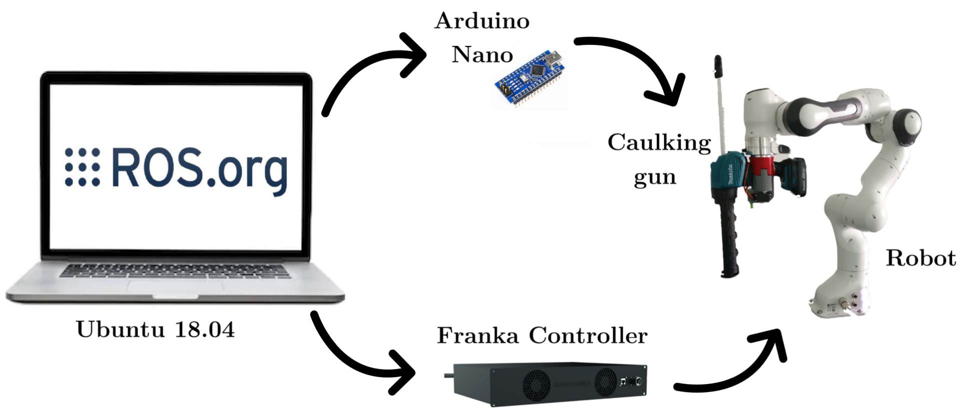


Figure 4.1: Scheme of the physical setup: from left to right there is an Ubuntu workstation which commands both the Arduino Nano and the Franka Controller that manage the robotic manipulator motion and sealant deposition flow

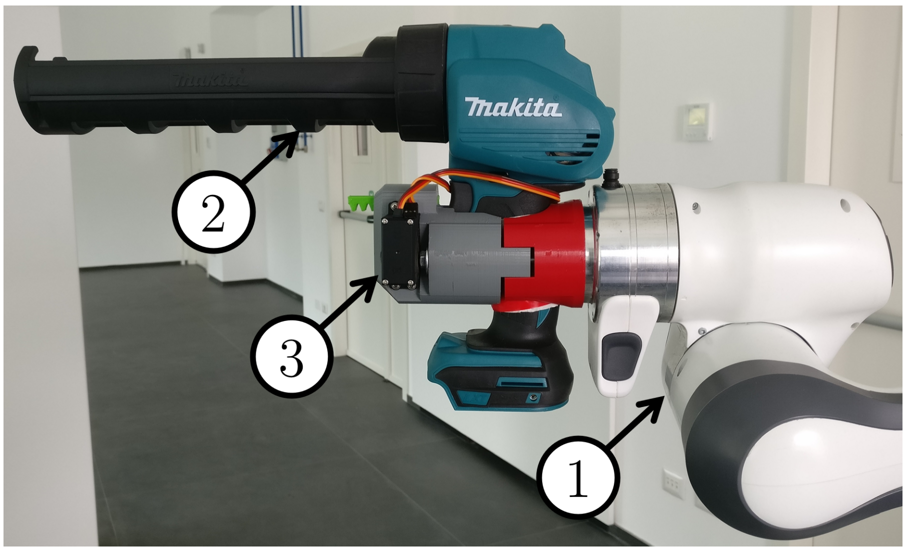


Figure 4.2: Physical setup for the experiments: (1) Franka robotic arm, (2) Makita caulking gun, (3) 3D printed custom flange

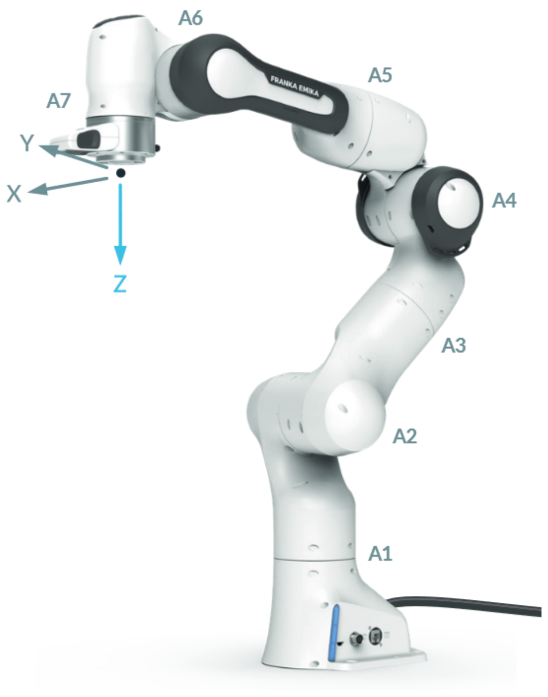


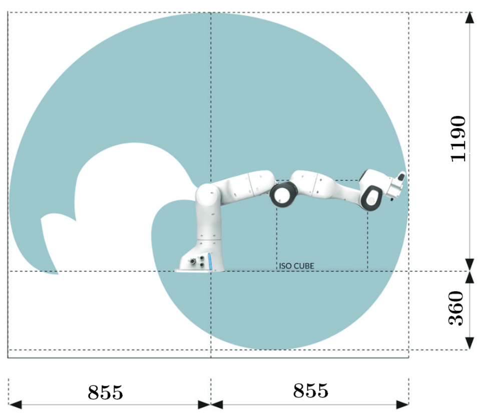
Figure 4.3: Franka Emika Panda

- Immediate control: it is provided with an easy block programming environment for simple task execution, but it also has a complete library of control commands useful to program it for more complex activities.

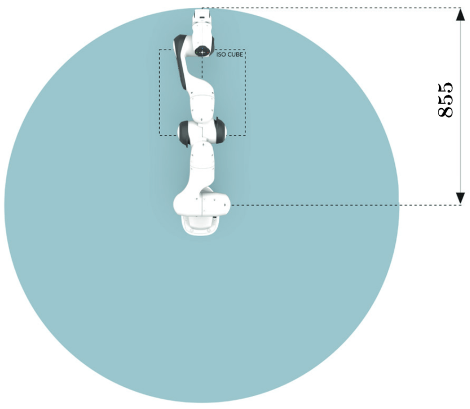
Several mechanical characteristics are interesting for various applications, from tracking motion to pick and place tasks. Franka Emika Panda manipulator is provided of torque sensors on all the joints, which are useful both for impedance control activities, both for precise motion control, moreover, it has a high frequency controller (1 kHz) and high repeatability in position.

Thanks to the additional DOF, the same EE position can be reached with multiple configurations. Moreover, it is also possible to keep fixed in space the EE, while moving the arm, which is important in a human-robot environment because it permits to change configuration and adapt to the background (this characteristic is possible since the dimension of the operational space is smaller with respect to the one of joint space). The extra DOF allows also to avoid singularity configurations and to better distribute the motion above all the axes.

Generally speaking, working in a collaborative environment means that



(a) Workspace side view



(b) Workspace top view

Figure 4.4: Franka Emika Panda Workspace [mm]

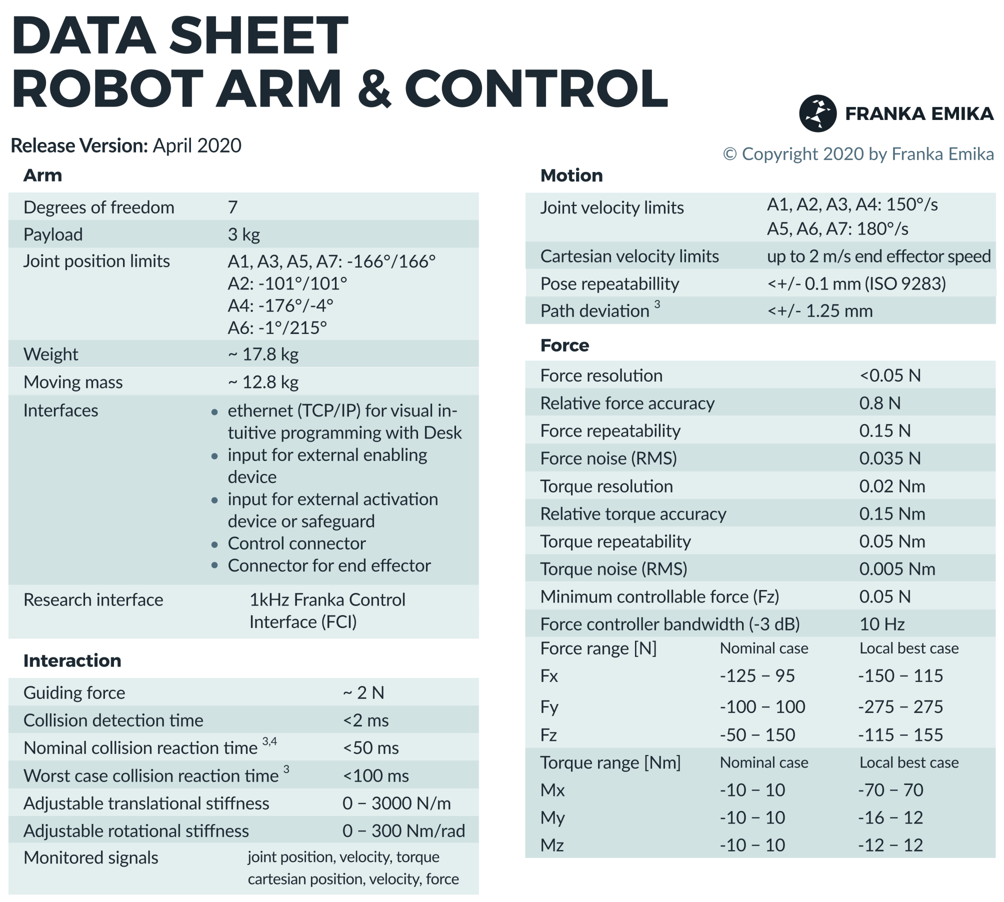
not only the robot must be collaborative, but also the task to be executed. Indeed, selecting the robot before knowing the task can be problematic; consider the welding application, even if a collaborative robot is chosen to perform the activity, the safety regulations of the mechanical procedure impose the use of fences around the robot workspace, and so it is impossible to work with human cooperation.

The sealing task does not present relevant risks for human operators, which means that a collaborative robot can be selected to accomplish and improve the reference task.

The collaborative nature of the robot refers to the manipulator capability to be inserted in an industrial environment where human operators can enter the robot workspace during the working activity. Typically this robotic characteristic is achieved only when a safety degree can be assured, which means implementing collision-detection sensors, with low velocities and low detection times, that can stop the execution when an external force is sensed on the joints.

In the case of Franka robot, the safety condition is provided by very fast detection time (smaller than 2 ms) and high-resolution torque sensors, which are able to achieve [32]:

- Resolution < 0.05 N
- Accuracy < 0.8 N
- Repeatability < 0.05 N



DATA SHEET
ROBOT ARM & CONTROL

FRANKA EMIKA

Release Version: April 2020

© Copyright 2020 by Franka Emika

Arm

Degrees of freedom	7
Payload	3 kg
Joint position limits	A1, A3, A5, A7: -166°/166° A2: -101°/101° A4: -176°/-4° A6: -1°/215°
Weight	~ 17.8 kg
Moving mass	~ 12.8 kg
Interfaces	• ethernet (TCP/IP) for visual intuitive programming with Desk • input for external enabling device • input for external activation device or safeguard • Control connector • Connector for end effector
Research interface	1kHz Franka Control Interface (FCI)

Interaction

Guiding force	~ 2 N
Collision detection time	<2 ms
Nominal collision reaction time [3,4]	<50 ms
Worst case collision reaction time [3]	<100 ms
Adjustable translational stiffness	0 – 3000 N/m
Adjustable rotational stiffness	0 – 300 Nm/rad
Monitored signals	joint position, velocity, torque cartesian position, velocity, force

Motion

Joint velocity limits	A1, A2, A3, A4: 150°/s A5, A6, A7: 180°/s
Cartesian velocity limits	up to 2 m/s end effector speed
Pose repeatabillity	<+/- 0.1 mm (ISO 9283)
Path deviation [3]	<+/- 1.25 mm

Force

Force resolution		<0.05 N
Relative force accuracy		0.8 N
Force repeatability		0.15 N
Force noise (RMS)		0.035 N
Torque resolution		0.02 Nm
Relative torque accuracy		0.15 Nm
Torque repeatability		0.05 Nm
Torque noise (RMS)		0.005 Nm
Minimum controllable force (Fz)		0.05 N
Force controller bandwidth (-3 dB)		10 Hz
Force range [N]	Nominal case	Local best case
Fx	-125 – 95	-150 – 115
Fy	-100 – 100	-275 – 275
Fz	-50 – 150	-115 – 155
Torque range [Nm]	Nominal case	Local best case
Mx	-10 – 10	-70 – 70
My	-10 – 10	-16 – 12
Mz	-10 – 10	-12 – 12

Figure 4.5: Franka Emika Panda datasheet relevant information. Source: www.franka.de/technology

The total reaction time is less than 50 ms according to datasheet information [fig. 4.5]. Moreover, rubber bumpers are placed on the robot arm to allow a reduced impact force if an unexpected collision happens.

4.1.1 Franka Control Interface

The Franka Controller Interface (FCI) [fig. 4.6] provides the low-level control of the robot, both for the hand and for the arm. The connection is bidirectional, so it is possible to either provide commands to the robot, or to read measured data, such as joint positions and torque values with 1 kHz acquisition frequency [33].

Through the FCI it is possible to compensate gravity and friction, to command the robot cartesian pose (or velocity) or command joint position (or velocity). All those different interfaces are available through ***libfranka***,

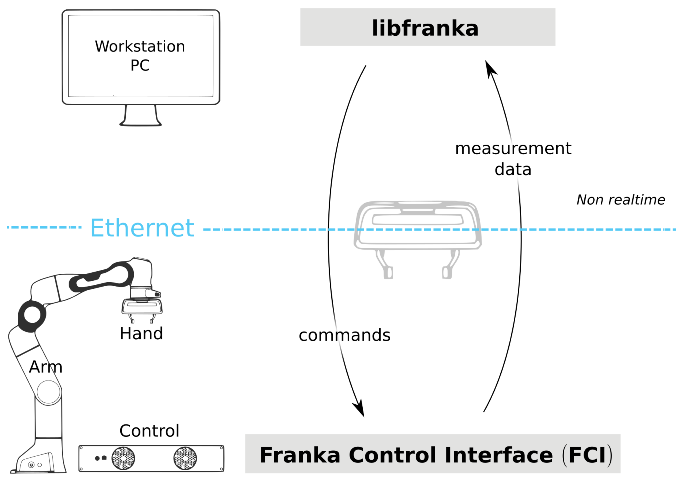


Figure 4.6: Schematic of non real-time Franka Control Interface. Source: [33]

which is the robot open source library, written in C++. A list of the robot controllers is shown in [fig. 4.7].

The library provides also measures of the joints actual positions, of the applied torques and force sensing for collision detection from the external environment. Those data can be useful for closed-loop applications, where a feedback signal with the knowledge of robot states is necessary.

Controlling the robot and providing a proper execution signal, can be done in two ways:

- Using the web interface, useful to control simple basic movements, but not recommended for providing a complete motion execution;

- Using the ROS environment, which is an open-source operating system commonly used for robotic control activities.

This thesis uses the more complete environment, ROS, which permits to implement a complex motion and to analyse the output signals of the robot itself.

Franka robots come with a build-in ROS package, called ***franka_ ros***, which provides the commands needed to interface the Python or C++ code with robot. The *franka_ ros* library will be briefly described in [sec: 4.2].

The use of ROS framework gives the possibility to plan robot movements and simulate them in a software simulator environment (*Rviz*).

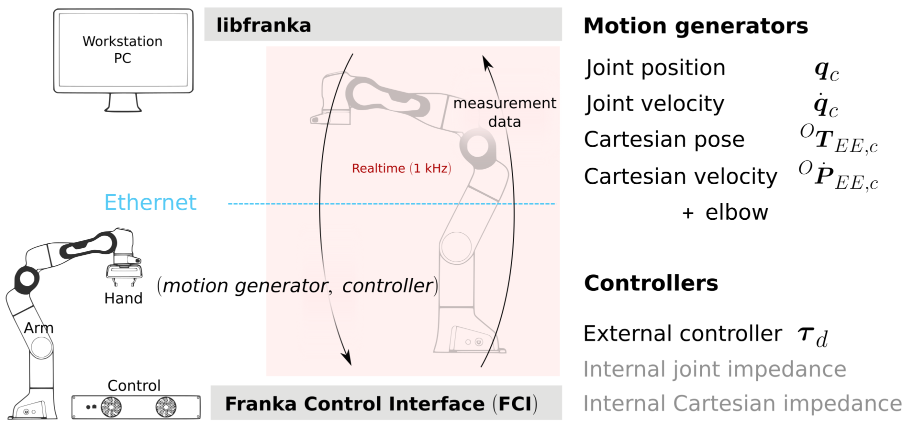

Workstation PC
libfranka
measurement data
Realtime (1 kHz)
Ethernet
Hand
(motion generator, controller)
Arm
Control
Franka Control Interface (FCI)
Motion generators
Joint position q_c
Joint velocity $\dot{q}_c$
Cartesian pose ${}^O T_{EE,c}$
Cartesian velocity ${}^O \dot{P}_{EE,c}$
+ elbow
Controllers
External controller τ_d
Internal joint impedance
Internal Cartesian impedance

Figure 4.7: List of the available real-time controllers for the robot. Source: [33]

4.1.2 Panda Dynamic Model

The dynamical structure of the *Franka Emika Panda* robot is in accordance with its description of collaborative manipulator. This means that some flexibility is admitted in the structure and in the specific case it is concentrated at the joints. This structure is in accordance with Flexible-Joint Robot (FJR), and so the joints must be modelled as elastic.

To derive the dynamics equations of the robot several standard assumptions must be made, according to [34]:

- Rotors are homogeneous bodies with the center of mass on the rotation axis;
- In a N joints manipulator, each motor is mounted on the $i-1$ link and conveys a motion on the i-link, where $i \in [1, N]$;
- Rotors have an angular velocity generated only by spinning action.

Joints can be modelled with two inertial contributions, rotor and link [fig. 4.8]. The rotation generated by the i-motor, ϑ_i, is converted in a joint rotation q_i through reduction ratios and the connection between the two masses is considered elastic, such that a mass-spring-damper system can be modelled on every joint.

The i-motor mass is defined as m_{motor}, while the link mass is m_{link}. The spring stiffness is modelled with the parameter k and the damping term related to friction forces is d.

The complete dynamic equation studies the motion of the robot dynamical system considering external torques and forces, damping actions and

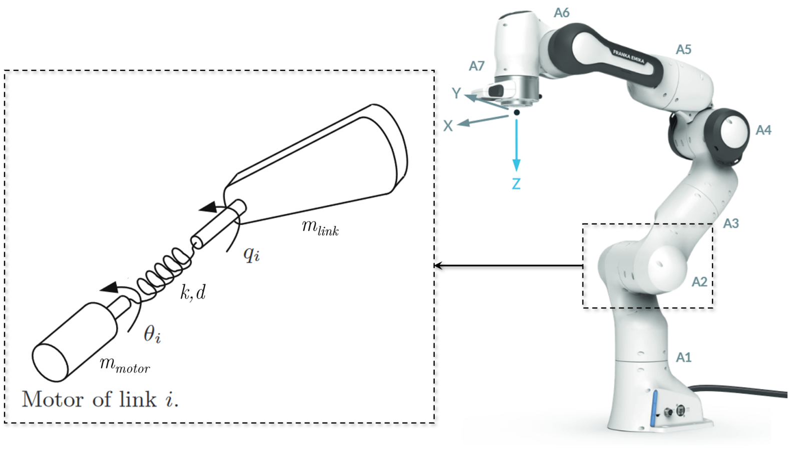


Figure 4.8: Flexible joint kinematic model

stiffness terms. For a N-DOF system, the joint coordinates are $\boldsymbol{q} \in \Re^{N\times 1}$, and the dynamical model is a set of N equations [35–37]:

$$\begin{cases} B(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + K\cdot(\boldsymbol{q} - \boldsymbol{\vartheta}) = 0 \\ J\ddot{\boldsymbol{\vartheta}} + K\cdot(\boldsymbol{\vartheta} - \boldsymbol{q}) = \boldsymbol{\tau} \end{cases} \tag{4.1}$$

where:

- $B(\boldsymbol{q}) \in \Re^{N\times N}$ is the manipulator inertial matrix;
- $C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \in \Re^{N\times 1}$ is the Coriolis vector and centrifugal forces term;
- $\boldsymbol{g}(\boldsymbol{q}) \in \Re^{N\times 1}$ is the vector of gravity terms.
- $K \in \Re^{N\times N}$ is the joint stiffness matrix.

In the second equation is reported the rotor side:

- $J \in \Re^{N\times N}$ is the rotor inertia;
- $\boldsymbol{\tau} \in \Re^{N\times 1}$ is the vector of motor torques.

For a deeper analysis on the dynamics of a FJR it is advised to look at [38]. This thesis does not focus on the impedance control or force analysis due to the nature of the task to be executed.

4.2 ROS Environment

The numerical code that controls the motions of the robot communicates with the manipulator itself and with the caulking gun using the ROS framework. This environment permits to command signals both to robot motion control system and to the Arduino board.

The communication between scripts written in traditional programming languages (Python, C++ etc.) and the robot is permitted by the *franka_ros* package, a build-in library that permits to receive numerical data and to send them to the robot. Instead, the Arduino is connected with the workstation through a Serial Port, that sends messages at fixed time rate to control the extrusion (this section is in open-loop, assuming that the servo motor does not saturates its torque).

4.2.1 ROS Nodes

The communication between the parts involved in the motion of the robot is permitted using *nodes*, which are sub-groups of the robot application that communicate with each other to retrieve data. Basically, *nodes* are executable programs that run inside the application. According to the ROS definition:

> "A *node* is a process that performs computations. *Nodes* are combined together into a graph and communicate each others using streaming *topics*, remote procedure call services, and the Parameter Server. These *nodes* are meant to operate at a fine-grained scale; a robot control system will usually comprise many *nodes*."

In order to transmit a message from a *node* to another, ROS platform uses buses called *topics*. Whenever a *node* is sending a signal, the procedure is referred to as *node publishing a topic*, instead when a node is receiving a message coming from a *topic*, then the *node is subscribing to a topic*. A graphical representation of a single *topic* communication is reported in [fig. 4.9], where it is also schematized the registration to the ROS master environment. In complex systems, it is common to have multiple publishers and multiple subscribers which exchange different message types over many *topics*.

The ROS implementation performed in this thesis for the off-line motion planning required the generation of one publishing *node* and one subscribing *node* needed to send position messages from the Python code to the

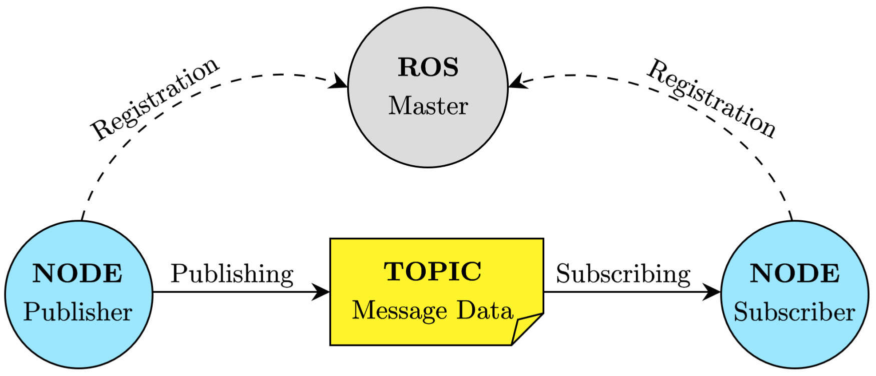


Figure 4.9: ROS standard communication

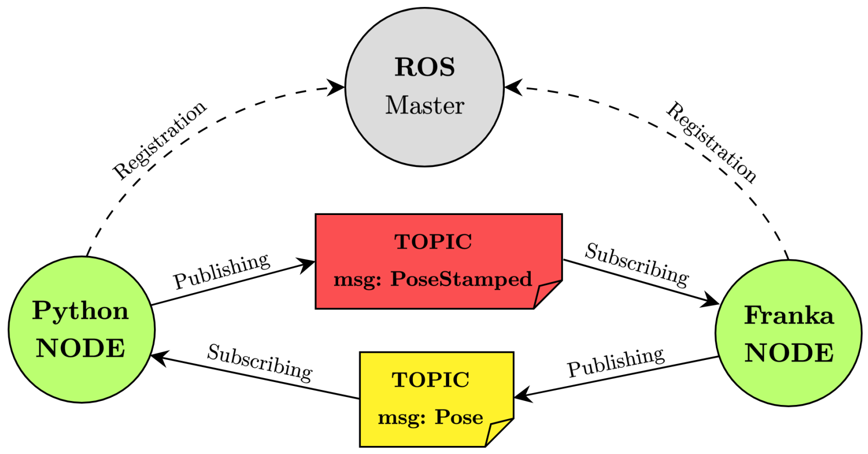


Figure 4.10: ROS structure: the Python *node* commands the robot positions (PoseStamped) to the Franka *node* and it receives back the actual robot position (Pose)

robot manipulator. The ROS *node-topic* framework related to the control structure analysed in this work is reported in [fig. 4.10].

4.3 Flange for Caulking Gun

The provided testbed requires the use of a commercial caulking gun, the *Makita DCG180*, for the implementation of the silicone deposition.

A physical limitation of the testbed was related to the mechanical connection between the robot EE and the caulking gun handle. The standard EE with the grasping hand [fig. 4.11] could not provide a rigid

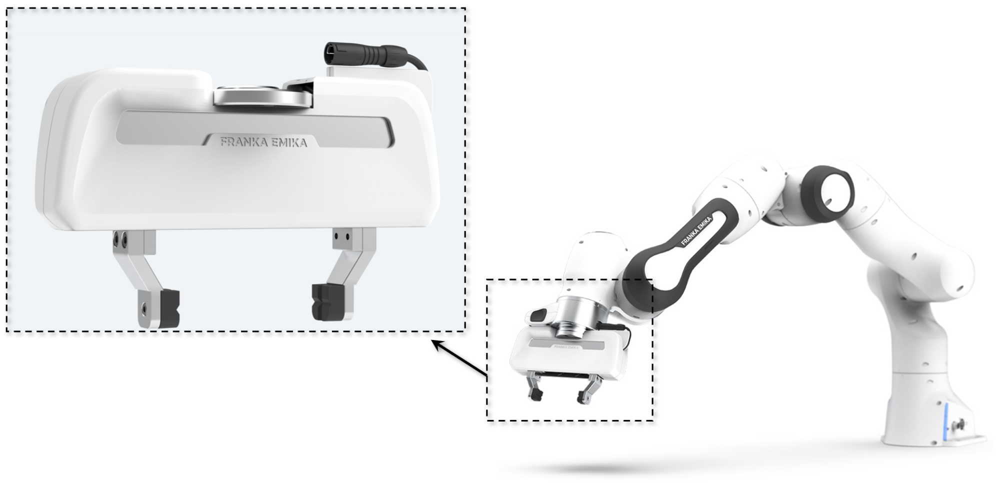


Figure 4.11: Franka Emika Panda standard gripper

connection between the two parts, so it has been studied and realized through FDM 3D printing a custom flange which permitted a stable and rigid application of the calking gun to the EE [fig. 4.12].

The designed flange firmly connects the gun to the robot EE, while also providing a control of the silicone extrusion using a servomotor. During the development of the flange, the following problems have been considered:

- **Boundary matching**: the design had to match the profile of the robot flange (DIN ISO 9409-1-A50) and the shape of the gun handle;
- **Motor action on the trigger**: the motorized rotation-to-translation mechanism generated by the servomotor;
- **Stiff connection**, to prevent unwanted vibrations on the gun tip (which would result in poor quality of the deposition);
- **Compactness**, which intrinsically permitted to avoid collisions with the robot arm.

The flange has been fully produced through FDM 3D printing, while some commercial M6 screws have been inserted to connect the piece to the robot EE with a clamped configuration.

During the design stage it has been defined the optimal system to perform the silicone extrusion: according to the datasheet of the caulking gun, the flow rate should have been proportional to the push on the trigger, while the experiments proved that the gun works with a mainly on-off system. Since the task requires a modulated extrusion rate of material, a

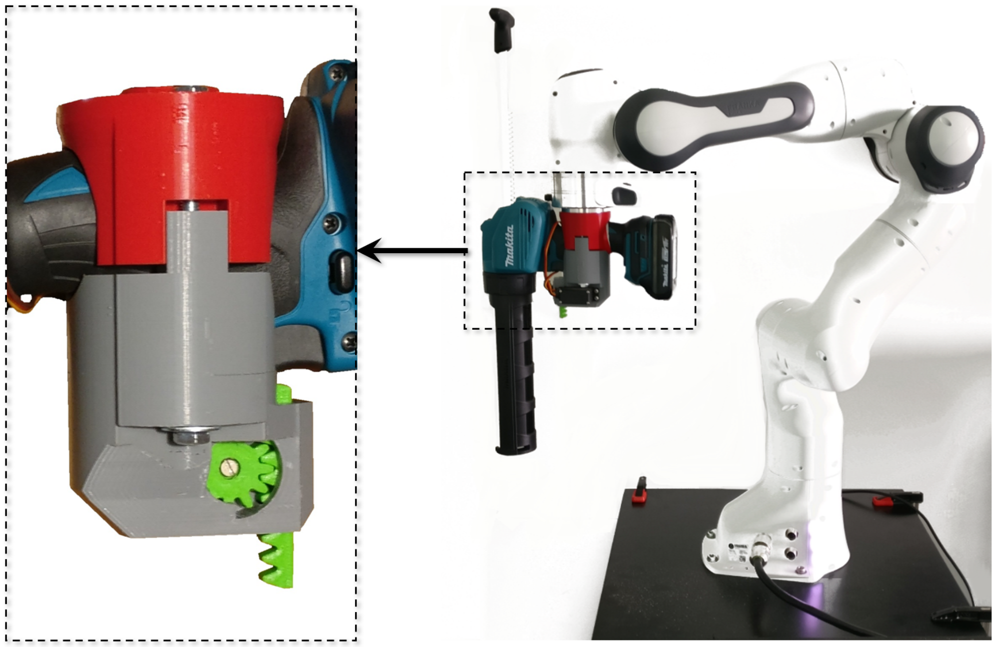


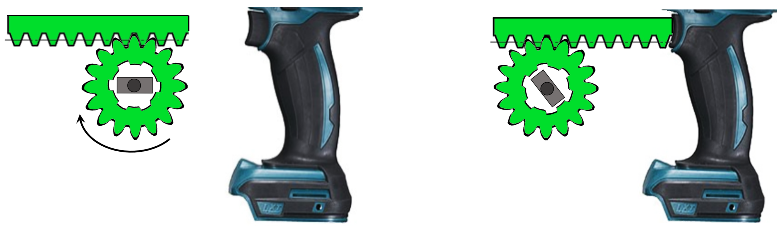
Figure 4.12: Close-up of the custom flange on the rack-pinion mechanism side. The red part is connected to the EE, while the grey part contains the motorized mechanism

Pulse-Width Modulation (PWM) strategy has been used to control the silicone flow.

To activate the extrusion, a rack-pinion mechanism motorized by a servomotor has been selected. The pinion is driven by an Arduino-controlled servomotor and the rack moves back and forth pressing the trigger [fig. 4.13].

For a complete implementation of the push mechanism it has been discussed the proper control strategy, with manual or automatic activation. In the former case it has been used a potentiometer which worked as a knob, while in the latter one the command signal is directly fed into the Arduino by means of a serial communication with the computer.

The automatic activation through a command signal is preferrable to the manual one, because it is possible to numerically generate a proper control signal that precisely presses the trigger, while in the manual case, the operator controls the execution, causing a reduction in the automatization of the task, and in reproducibility of the results. However, in both cases, it has been verified that the caulking gun never saturates its pushing force (5 kN from datasheet) since the overload warning light has never switched on.



(a) OFF state **(b)** ON state

Figure 4.13: Trigger activation, rack-pinion mechanism

4.3.1 Motor Controller

The trigger activation mechanism has been controlled with a servomotor that has an angular excursion of $\simeq 120°$ and a rating voltage of 5 V. Since the current absorption is very low, it can be directly fed by the Arduino board, which is connected to a 5 V power supply. The board also needs an USB connection with the workstation, which is necessary for the serial communication.

The angular position of the motor must be experimentally evaluated in order to increase precision of the rotation-to-translation mechanism. Knowing that the motor receives as an input a pulse of $\simeq 500 \div 2500$ μs, where the two limit values correspond to the angular limits in both directions, the tuning of the motor consists in relating the pulse signal with the angular position of the motor itself. In the analysed case, the servomotor is connected with the rack-pinion mechanism, that controls the push on the trigger [fig. 4.13], so it is possible to directly correlate the pulse duration with the gun trigger pression.

The Arduino library ***Servo.h*** has been used to control the motor. This environment has built-in functions that permit to directly generate a motion by specifying the pulse duration in μs. The Arduino code used to manage the motor is provided in [app. A].

PWM Flow Controller

The deposition flow rate has been set to be proportional to the cartesian velocity of the TCP with an experimental correlation (i.e. max extrusion rate at max EE speed and vice versa).

Since the extrusion control has a much lower bandwidth with respect to the ROS *node* which manages the robotic arm (which works at 1 kHz), the silicone flow is set with an update frequency of 5 Hz.

In order to modulate the silicone extrusion, the PWM control strategy

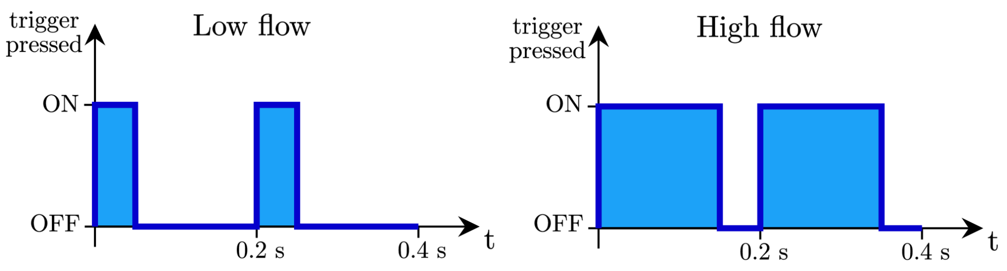


Figure 4.14: Example of PWM with a total duration of 0.2 s: the two cases show different duty cycles for different flow rates

has been used. Practically, each fifth of second (i.e. with a 5 Hz frequency) the gun trigger is pushed with a pulse of variable duration [fig. 4.14]. This duration is close to 0.2 s at maximum flow rate and conversely tends to zero if the flow rate is minimum. This strategy permits to have a mean flow rate that varies accordingly with the wished extrusion rate. The fluctuations of the square wave are negligible if the frequency is high enough with respect to the system dynamics.

Chapter 5

Algorithm Development

This chapter reports all the data processing steps needed to obtain the final target trajectory that must be fed into the robot controller. This procedure permits to switch from a generic reference path (input), to an output trajectory which is approximated through DMP with an appropriate punctual velocity reference.

Before analysing the effective planning strategy, the sealing process must be analyzed. Typically, when an expert operator performs a manual sealing task, the caulking gun is rotated of a certain angle α in the direction of movement, with respect to the orthogonal axis of the deposition plane [fig. 5.1]. This manual configuration permits to deposit the sealant in a proper way, while also avoiding nozzle clogging.

In the proposed study-case, the collaborative robot will not have any problems in maintaining a precise vertical offset form the deposition plane, so proper results can be achieved by working vertically (i.e. $\alpha = 0$), as for a typical material extrusion of a FDM 3D printer [chap 2.3]. Keeping fixed the EE orientation allows to have a wider bi-dimensional operational space because the piston of the gun will not interfere with manipulator arm movements. Moreover, fixing the orientation permits an easier planning of the robot pose, since the Euler angles remain constant all over the path.

The task to be executed requires a TCP coordinate definition in 3D space, but since the trajectory of interest is just bi-dimensional, the problem can be analyzed in a simpler way. Indeed, the general EE pose (2.1) in 3D is characterized by 6 parameters for each point of the path, but considering the caulking gun perpendicular to the plane of the path, the EE pose (2.1) has fixed orientation ($\boldsymbol{\phi}_e$) and fixed vertical offset (z) along all the path.

In order to optimize in time and quality the sealant deposition process, the proposed approach analyses the execution velocity as a critical parameter that must be tuned accordingly with the path features. This trajectory

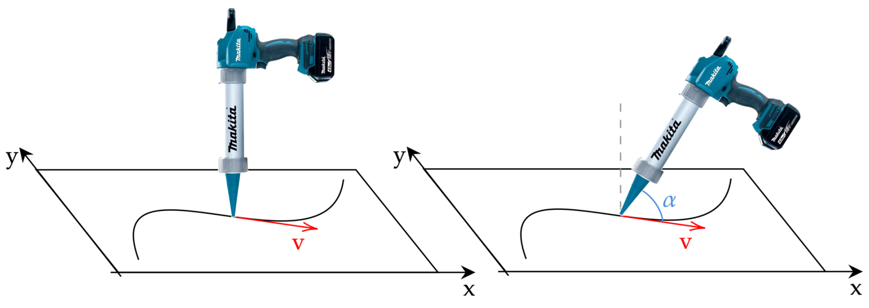


Figure 5.1: Different approaches for the gun pose definition: on the left there is a perpendicular deposition, while on the right the gun is inclined in the forward direction of an angle α

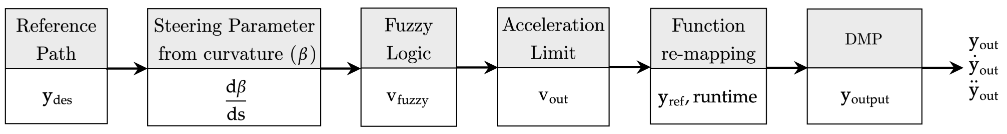


Figure 5.2: Scheme of the data-processing steps

characterization is strictly related with the background of the sealing task.

To perform a time optimization of a silicone deposition, the process mean speed must be increased, but completely different path features (straight line VS very tight curve) cannot be approached in the same way otherwise the final extrusion quality would be highly penalized. To solve this issue, it has been developed a code which automatically selects the right velocity accordingly to the feature that the robot is performing.

The complete algorithm is composed of several pre-processing steps before the motion planning one (DMP), which are necessary to generate a correct signal, in terms of limit accelerations and smoothing analysis. These sub-blocks are reported in sequence in [fig. 5.2], while a clearer description of each part is provided in the following sections:

- **Path re-sampling**
- **Curvature analysis**
- **Spatial velocity reference**
- **Acceleration and deceleration limits**
- **DMP execution**

The complete algorithm is developed to be modular. This means that each block can be independently modified and adapted to the task setup. In this thesis each block is tuned to work with the provided robot and with the given caulking gun.

In [fig. 5.2], the input ($\boldsymbol{y}_{des}$) is an array of points in cartesian coordinates which can have any spatial discretization as the first step of the process is a spatial re-sampling. This first passage is required to provide a general framework, able to acquire any kind of *via-points* that characterize a path: i.e. a straight line can be encoded at least with the two endpoints, but also with an array of whatever point density. Also, curves can be mapped with a low number of points, but obviously it would cause a poor resolution.

5.1 Path Re-Sampling

Since the core of the trajectory generation is the velocity reference, it is important to consider not only the path geometry, but also the time instant in which each point is reached. So, the first thing to do is to have a general formulation for the path to be reproduced.

It has been assumed that a curve will have a different optimal velocity with respect to a straight line. This means that the code must be able to automatically distinguish between different path features.

Analysing the path geometry for sealing deposition depends on the characteristic dimension of the caulking gun nozzle. The same curve can be defined as *large* or *tight* according to the nozzle diameter itself. Whenever the dimension of the tip is comparable with the feature, then the curve is *tight*, instead if the feature is way bigger, the curve can be called *large*.

In [fig. 5.3] is represented a nozzle performing a hairpin turn: the different subfigures show a qualitative representation of the behaviour using different point density with respect to the length of a single nozzle diameter.

As the number of *points-per-diameter* increases, the path definition rises, but in physical applications too many points are useless because the path would be over-characterized, causing at first larger computational effort, but also inability of the nozzle to reproduce such small features. A good trade-off between proper representation and low amount of data to be managed is selected at: ***4 pts per diameter***. This parameter is very important for the first re-discretization of the path, that rejects features that are too small to be represented. This behaviour intrinsically smooths the path, applying a sort of filtering effect to the reference input.

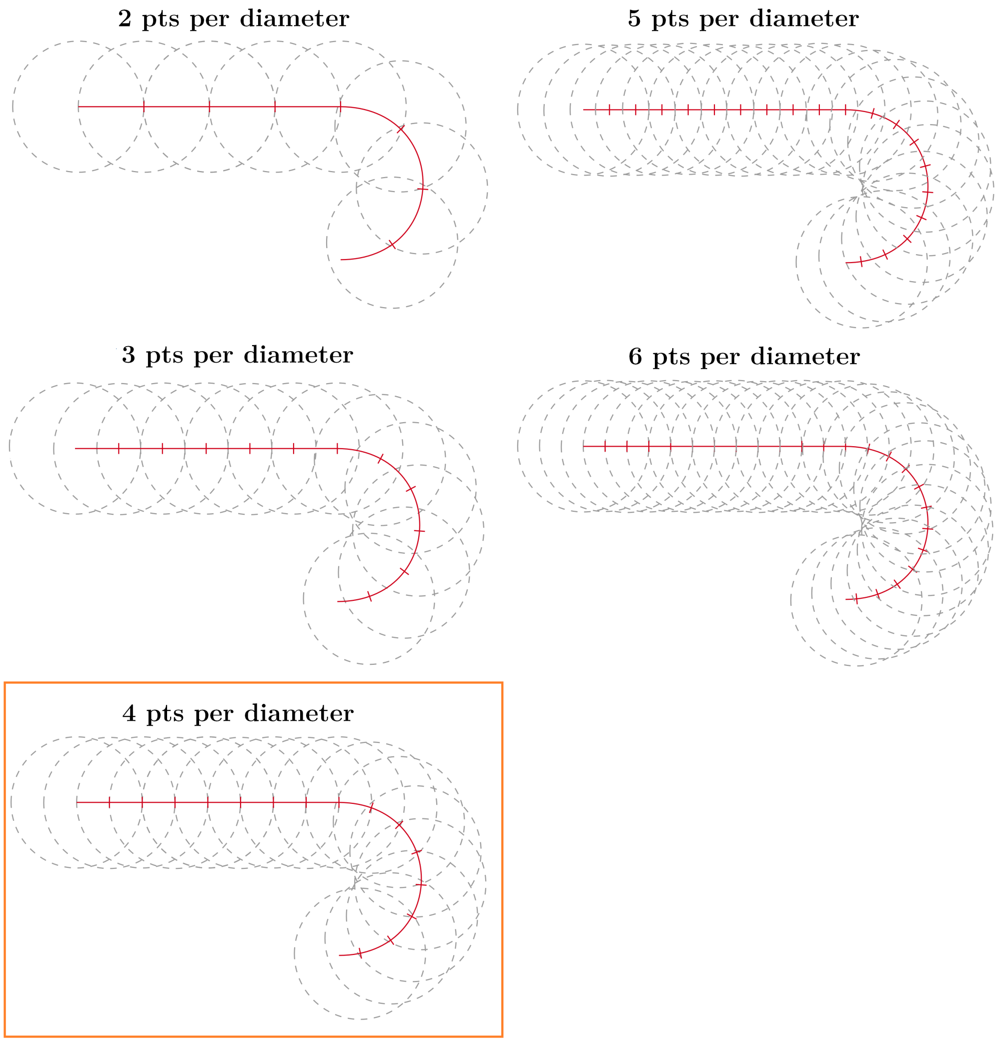


Figure 5.3: Analysis of the *point density* along the path: the gray circle represents the nozzle at each point along the natural coordinate during a hairpin turn (red curve). The path geometry and the nozzle diameter are kept constant, while the point density increases. It can be seen that a high density better approximates the curve.

Given a series of N_{pts} points expressed as XY coordinates:

$$\boldsymbol{y}_{des,i} = [x_i, y_i] \ , \ i \in [0, N_{pts}] \tag{5.1}$$

they are re-sampled to be equally spaced, accordingly with the nozzle diameter re-parametrization (4 points per diameter). It can be defined the *point density* as:

$$pt_{density} = \frac{4}{d_{nozzle}} \quad \frac{[pts]}{[mm]} \tag{5.2}$$

and the spatial step ds as:

$$ds = \frac{1}{pt_{density}} \quad \frac{[mm]}{[pts]} \tag{5.3}$$

At last, it is obtained the re-sampled path, $\boldsymbol{y}_{eq-space}$, with a number of points N_{new} equally spaced along the natural coordinate s:

$$N_{new} = l_{path} \cdot pt_{density} \tag{5.4}$$

where l_{path} is the total path length.

5.2 Curvature Analysis

From the re-sampled path, it is possible to analyze the curvature along the natural coordinate. The aim of this process is to create a single parameter, which defines punctually the characteristic of the path, i.e. it must be identified if the EE is approaching a straight line or a curve.

Without loss of generality, straight lines can be considered as curves of infinite radius, which permits to analyse the path by checking the curvature of each feature. The tighter the curve, the higher the value of the characteristic parameter defined as: *steering*.

5.2.1 Steering Parameter

The evaluation of the *steering* parameter requires the analysis of the path: $\boldsymbol{y}_{eq-space}(s)$. Basically, the slope is evaluated as the derivative of the y-coordinate with respect to the x-coordinate:

$$slope = \frac{dy}{dx} \tag{5.5}$$

then, the punctual *steering* value is described as the difference between the inclination of the path:

$$steering = d\beta \tag{5.6}$$

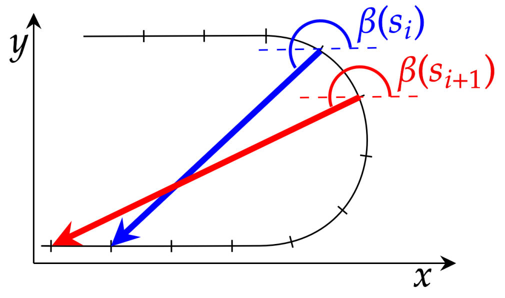


Figure 5.4: Example of vector of length 7 spaces, that approaches a curve

where $\beta = \arctan 2(dy, dx)$ is the inclination angle with respect to the x axis.

The numerical analysis defines the *steering* value in the i^{th} point with the use of finite differences and a vector of length n; the procedure is here reported with also a graphical representation [fig. 5.4]:

- the vector $\boldsymbol{v}_{i,i+n}$ is defined, connecting the actual i^{th} point under analysis with a following one n points after;

- the vector $\boldsymbol{v}_{i+1,i+n+1}$ is defined, connecting the subsequent point with respect to the one under analysis ($i^{th} + 1$) with another point of n spatial steps after;

- the angles between those two vectors and the horizontal axis are computed: $\beta(s_i)$ is the angle between $\boldsymbol{v}_{i,i+n}$ and the horizontal axis, and $\beta(s_{i+1})$ is the corresponding for $\boldsymbol{v}_{i+1,i+n+1}$;

$$\begin{cases} \beta(s_i) = \arctan 2\left(y_{i+n} - y_i,\ x_{i+n} - x_i\right) \\ \beta(s_{i+1}) = \arctan 2\left(y_{i+1+n} - y_{i+1},\ x_{i+1+n} - x_{i+1}\right) \end{cases} \tag{5.7}$$

- the *steering* in the i^{th} point is defined as the difference between $\beta(s_{i+1})$ and $\beta(s_i)$:

$$steering_i = \beta(s_{i+1}) - \beta(s_i) \tag{5.8}$$

This analysis uses a moving vector to characterize the path; to better explain this concept it can be imagined to have a vehicle moving on a track; when it has to approach a curve its direction changes and the higher the change of direction, the tighter the curve. Also, the size of the vehicle

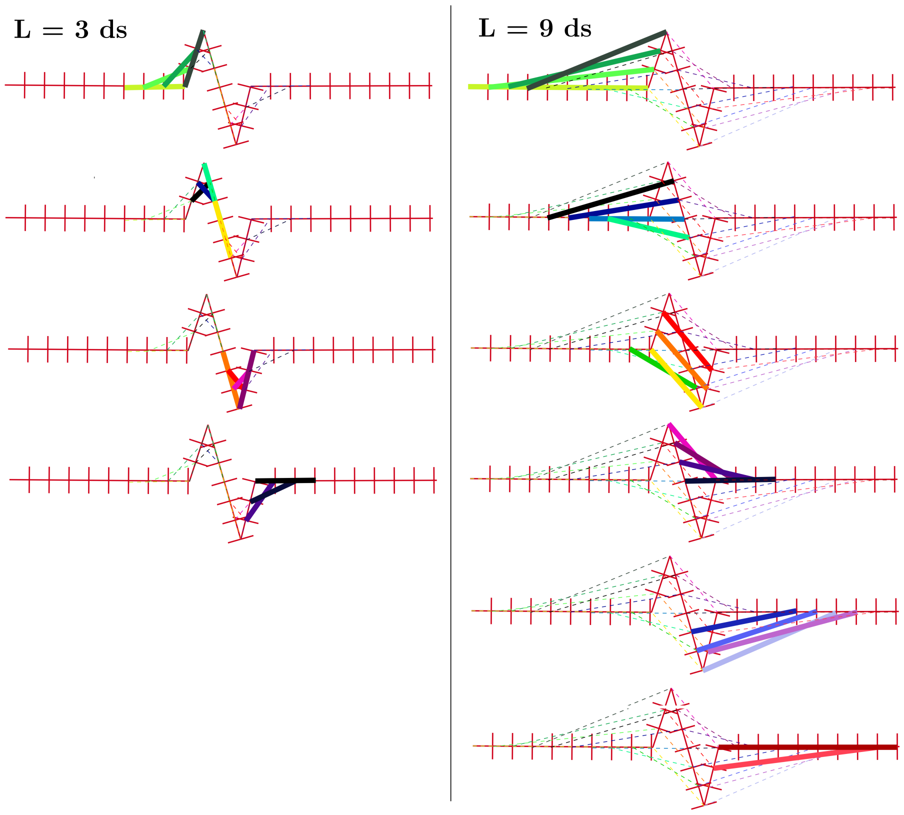


Figure 5.5: Example of different feature executions for two bodies which have a very different length

affects the perception of the curve, indeed the shorter the vector ($n_{min} = 1$) the more localized and sudden is the curvature effect. This parameter must be precisely tuned to define the smoothing effect during the execution of the track.

The difference between two consecutive values of β gives an idea of the geometrical feature in that point of the path. It is easy to notice that in the middle of a straight line, the angle computed with respect to two subsequent steps will be the same, and consequently *steering* will be null.

5.2.2 Steering Vector Length

The choice of the vector length (in terms of points n) plays a fundamental role in the path analysis. A brief analysis is reported in [fig. 5.5], where two different vector lengths are compared. Firstly, if considered a very long body, it will not be able to characterize properly small features (in terms of *steering* value), indeed it will act as a filter on very sharp directional changes.

Conversely, if the body is too short, it will properly characterize the curve, but it will not recognize it enough steps in advance, causing a sudden change of direction that will not smooth the execution velocity before the curve itself. Moreover, a short vector may result in high fluctuations of the *steering* value, which will affect the velocity reference.

This behaviour is not proper for good material deposition, which would be negatively affected by discontinuous and sudden velocity variations. A proper task execution is achieved only with smooth velocity variations, so it makes no sense to have sharp changes in the *steering* (even if the velocity variation is then processed taking into account the acceleration limit).

In [fig. 5.5] are reported two opposite situations when facing a sudden phenomenon: with a vector length of 3 ds, the geometry is well characterized in terms of direction change, but the feature is noticed too late. The opposite behavior refers to the case of 9 ds length: the small path discontinuity is recognized many steps in advance, but the sudden directional change is not properly defined.

The numerical result of the analysis on the profile shown in [fig. 5.5] is reported in [fig. 5.6] for the case of $n = 3$, $n = 5$ and $n = 9$. The different *steering* profiles are:

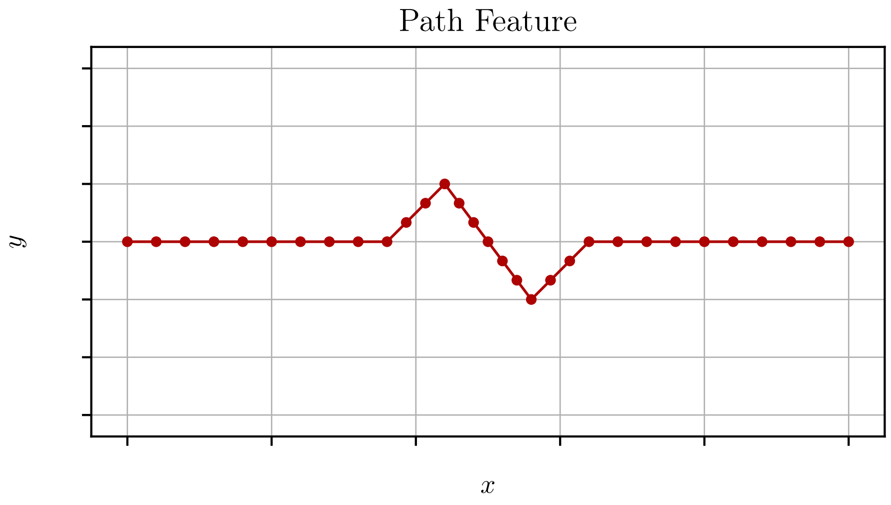
- $n = 3$: slope perception only in correspondence of the slope, this causes a large punctual *steering* value variation;

- $n = 5$: intermediate feature perception, this permits to achieve a trade-off between smoothing and preparation to face the curve;

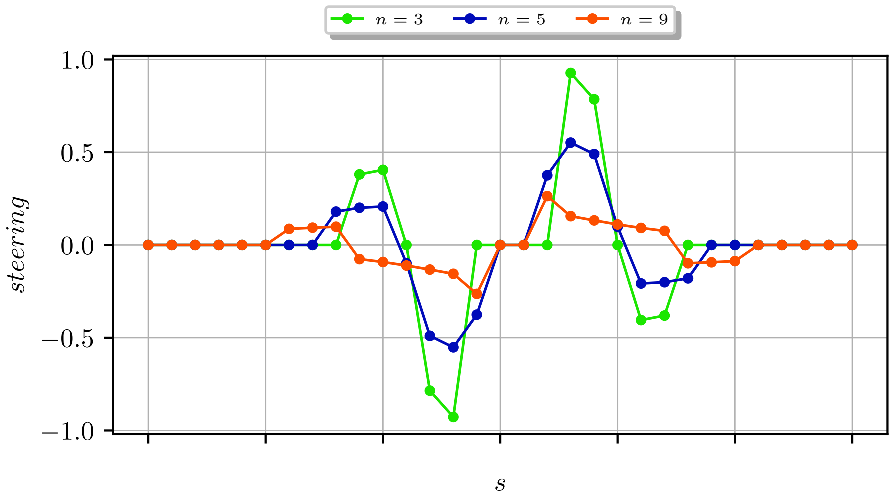
- $n = 9$: earliest perception of the change of slope, smoother final profile with lower peaks.

Another example is reported in [fig. 5.7], where multiple vector lengths are compared in correspondence of a hairpin turn. The longer the vector, the sooner the curve is approached.

Considering the dimension of the geometrical features of interest for the thesis topic, the best trade-off has been shown to be a ***body of length*** $\boldsymbol{n=5}$. This dimension is a bit more than two times the nozzle diameter and permits to well characterize small curves, while also generating a smoothing effect on the punctual variations that are not advised for the velocity reference generation.



(a) Reference Path Feature



(b) Steering Evaluation with different windows

Figure 5.6: *Steering* value for different n lengths: the higher n, the smoother the *steering* profile. For $n = 3$ the steering variation is sudden and with higher variation, while for $n = 9$ the profile has lower peaks

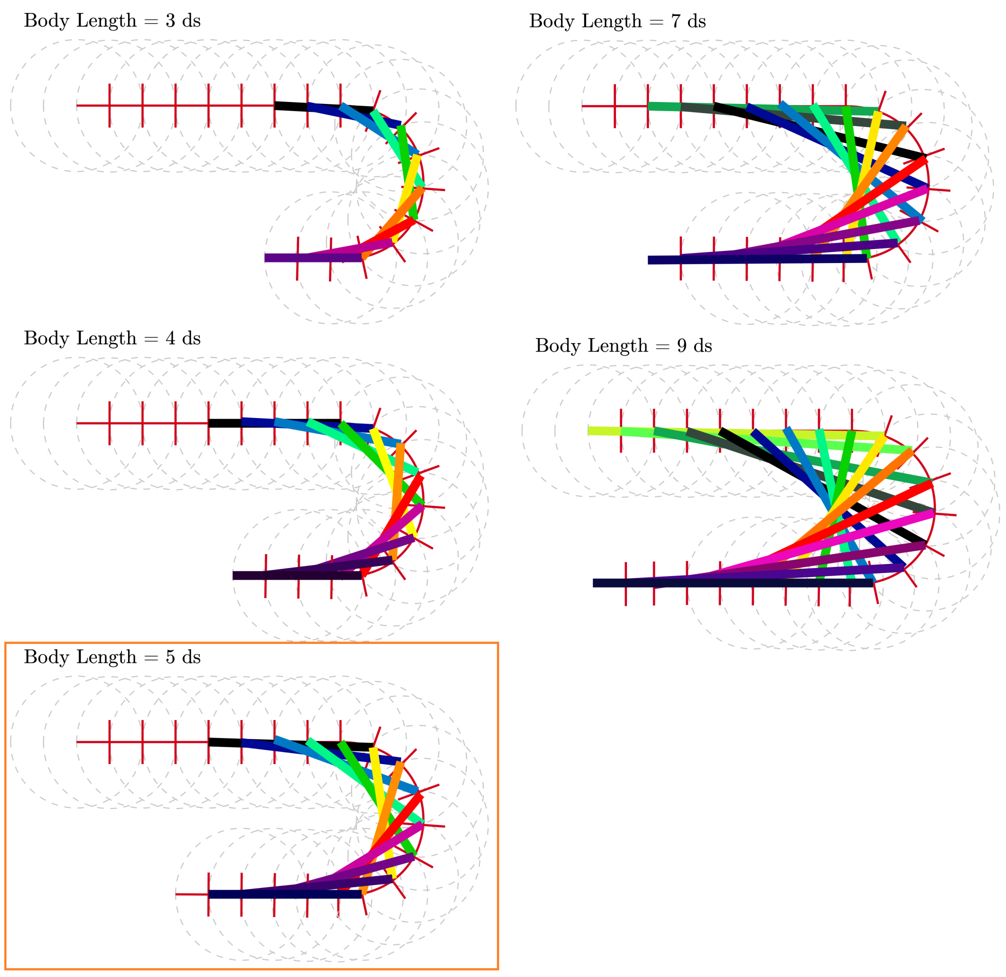


Figure 5.7: Hairpin turn performed by different body lengths

5.2.3 Steering Post-Process

Before proceeding with the velocity analysis, it is important to mention some considerations on the steering parameter:

- It is not relevant if the curve is left-wise or right-wise, so the ***absolute value*** of *steering* is considered;

- Very sharp edges cannot be performed at near zero speed, because it would mean to stop the process, so the *steering* is ***clipped*** at a maximum value (which will be correlated to the minimum velocity in the definition of the FL controller);

- Smooth velocity profiles are preferred for the task execution since they reduce vibrations, so a ***moving window mean*** is applied to the punctual value of *steering*. This passage permits to achieve an extra smoothing (i.e. if a small straight line is placed between two curves, the algorithm should not produce a high velocity reference between the features, because it would cause a big acceleration/deceleration. It is then preferred to have a slower velocity increase/decrease).

A sketch of code in which the *steering* value is computed, is reported in [lst. B.1].

Once all those data-processing techniques have been applied, the velocity reference can be defined.

5.3 Spatial Velocity Reference

The post-processed *steering* parameter can now be related with the velocity reference. Since a mathematical formulation of the velocity is not obtainable in closed form due to the highly experimental setup, a FL controller has been chosen for the definition of the velocity reference.

5.3.1 Fuzzy Logic Controller

The FL is a generalization of the classical Boolean algebra [39]. In binary logic, a sentence can assume only two values, *True* or *False* (or alternatively $\{0, 1\}$), while in FL each concept is expressed through a degree of truth belonging to the full interval $[0, 1]$.

This framework is widely adopted in decision making processes, since it well replicates the way in which the human brain takes decisions. The selection of FL is suitable whenever a mathematical model of the system does not exist or when the system is difficult to be modeled, as in the case of the presented work. Moreover, due to its capabilities of smoothly passing from one output to another one and of allowing for an input to belong to more than one class, FL is able to work with vague information [40].

Fuzzy Logic Model

The FL controller transforms a crisp input value into a crisp output, using Fuzzy sets to characterize the input and outputs and an inference system as decision making process [41]. The basic steps of the procedure are schematized in [fig. 5.8].

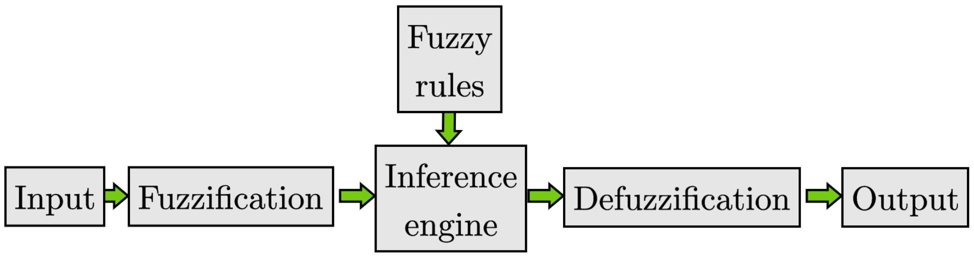


Figure 5.8: Fuzzy Logic block scheme

Class name	Steering Values	Class name	Velocity Values [mm/s]
Very tight curve	[0.22, 0.5]	Very slow	[0, 30]
Tight curve	[0.1, 0.45]	Slow	[15, 48]
Large curve	[0.02, 0.22]	Medium	[30, 75]
Straight line	[0, 0.1]	Fast	[66, 80]

Table 5.1: Input-Output Fuzzy Rules in [fig. 5.9]

The main characteristic of FL is that it allows partial belonging to more than a single class during the fuzzification procedure. In this way, an input value can be described by its degree of belonging to each class, instead of defining it with only a single class (as in the case of classical logic).

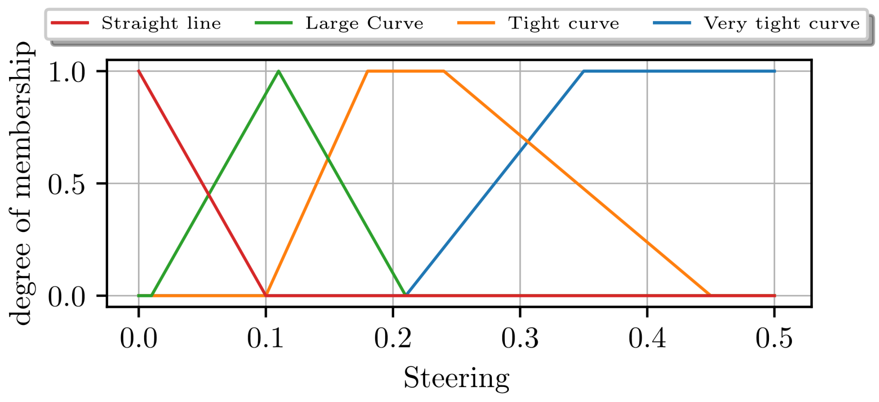
In [fig. 5.9] are reported the Fuzzy sets used in this thesis. Accordingly with the experimental setup, there are been defined four classes, both for the input *steering* parameter, both for the output velocity reference. The selected values are reported in [tab. 5.1].

For example, considering the [fig. 5.9a], a *steering* value of 0.13 is related to: 80% of a *tight curve* and 37.5% of a *large curve*. The fuzzy process will define the output velocity value by considering these degrees of truth. The fuzzified input is processed by an inference engine, which considers a set of fuzzy rules to relate the input action to the output one. Typically, the rules are *if-then* clauses. In reference to the previous example, they can be:

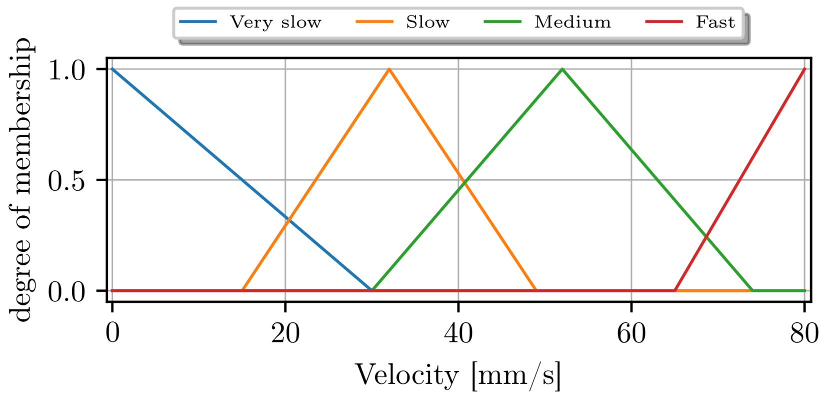
"***If*** the path feature is a *straight line*, ***then*** go *fast*"

"***If*** the path feature is a *small curve*, ***then*** go *slow*"

In this thesis a *Mamdani inference model* has been used [fig. 5.10],



(a) Fuzzy input membership functions



(b) Fuzzy output membership functions

Figure 5.9: FL membership functions

which basically states that the degree of truth of the consequent of a Fuzzy rule is equal to the one of the antecedent [42]. In case there is more than just one antecedent rule, these are merged together with a *min t-norm operator*, while in cases in which there are more than just one consequent, they are combined using a *max s-norm operator*. The *min t-norm* assigns to the consequent (green in [fig. 5.10]) the lower degree of truth between the ones of the antecedents (red and blue in [fig. 5.10]), while the *max s-norm* combines the consequents into the output evaluating for each point of the discretization the maximum values of the consequents (orange in [fig. 5.10]). The crisp output is typically computed evaluating the center of mass of the *max s-norm* output.

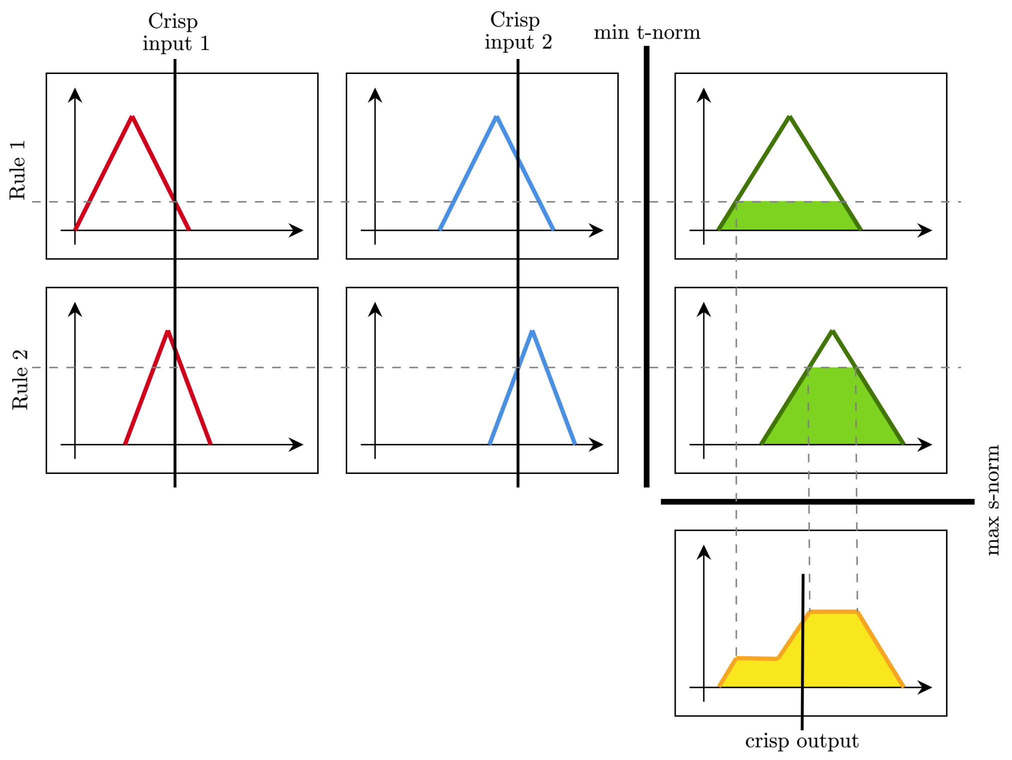


Figure 5.10: Example of Mamdani Fuzzy inference engines which uses min t-norm and max s-norm

5.3.2 Fuzzy Logic Tuning

An experimental campaign is needed to tune the velocity references: both the membership functions of the input and the output must be identified.

Some standard features, like straight lines and different-radius curves, are chosen as references, then some experiments are performed to establish the best suited velocity for performing the feature. Once the matching velocities have been evaluated, they are related with the corresponding *steering* value set computed for that reference feature.

The standard geometries should be chosen to well-represent the full-scale values of the steering parameter, then all the other intermediate curves will be characterized by a velocity reference which is computed by the FL controller. The full-scale of the fuzzy input must match the steering value (i.e. from zero to the clipping value), while the output scale must be from a near-zero velocity, to the maximum velocity reachable in the physical setup (i.e. it is selected the lower value between the maximum

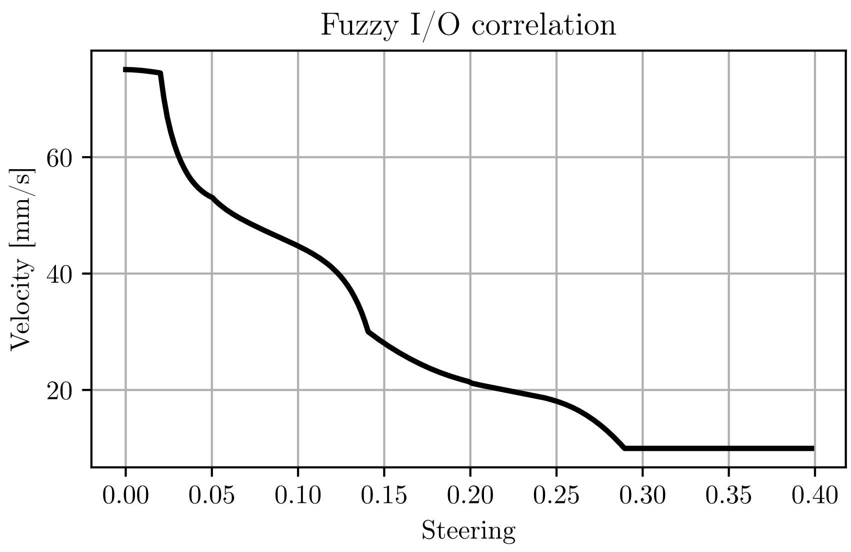


Figure 5.11: Fuzzy I/O relation

deposition speed and the maximum robot velocity). The obtained FL membership functions are reported in [fig. 5.9], as well as the input/output relation [fig. 5.11].

5.3.3 Considerations on the Velocity Reference

It must be remembered that up to now the steering and velocity are computed as function of the spatial coordinate s (instead of using time). This means that the "classical" formulation of velocity as function of time, is here expressed as function of space. This approach is necessary to define the punctual velocity along the natural coordinate, while time characterization is not addressed yet.

Since the DMP formulation requires an input expressed as an array of points function of time, the FL output is manipulated to match the DMP requirement. Using the definition of mean velocity:

$$v_{mean} = \frac{\Delta s}{\Delta t} \tag{5.9}$$

it is possible to compute the time interval between each step. Such that:

$$dt_{i,i+1} = \frac{s_{i+1} - s_i}{v_{mean,i}} \tag{5.10}$$

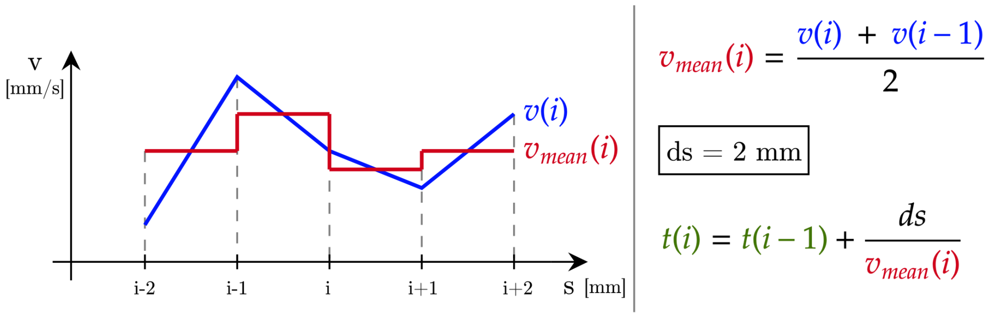

v
[mm/s]
$v(i)$
$v_{mean}(i)$
i-2
i-1
i
i+1
i+2
s [mm]
$v_{mean}(i) = \frac{v(i) + v(i-1)}{2}$
ds = 2 mm
$t(i) = t(i-1) + \frac{ds}{v_{mean}(i)}$

Figure 5.12: Mean velocity along the natural coordinate

where the spatial step $ds = s_{i+1} - s_i$ is fixed according to the first resampling where $\boldsymbol{y}_{eq-space}$ has been defined. The mean velocity of the space step is simply:

$$v_{mean,i} = \frac{v_{i+1} + v_i}{2} \tag{5.11}$$

Since the spatial step is fixed along all the path, the time steps will be variable, depending on the instantaneous velocity value. The procedure is reported in [fig. 5.12]. The desired output trajectory will be described as cartesian points function of time. The cartesian axes need to be reversed to pass from $t(s)$ to $s(t)$. This procedure is executed together with the acceleration limit evaluation, which is explained in [sec. 5.4].

A last consideration on the velocity profile has to be made: in order for the path to be feasible, both starting and ending velocity have been set to be null, otherwise the signal sent to the robot would require a sudden acceleration in starting and ending conditions.

5.4 Acceleration and Deceleration Limits

Some considerations similar to the ones proposed during the velocity analysis, have to be done also for the acceleration: if the *steering* parameter suddenly changes, it will result in a velocity gradient that cannot be achieved by the system dynamics. In this case the proposed solution sets the maximum acceleration (and deceleration) by relaxing the time at which each point is reached; the algorithm changes when dealing with an acceleration or a deceleration case. This passage of the process works with the mean velocity for computing the time step, so directly with the data manipulation procedure explained in [sec. 5.3.3].

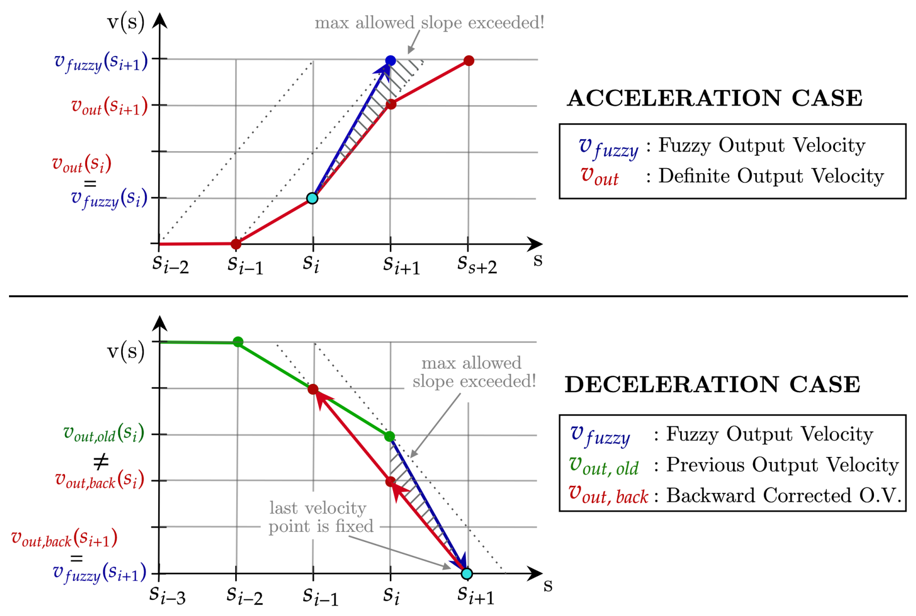

v(s)
max allowed slope exceeded!
$v_{fuzzy}(s_{i+1})$
$v_{out}(s_{i+1})$
$v_{out}(s_i) = v_{fuzzy}(s_i)$
s_{i-2} s_{i-1} s_i s_{i+1} s_{s+2} s
ACCELERATION CASE
v_{fuzzy} : Fuzzy Output Velocity
v_{out} : Definite Output Velocity
v(s)
max allowed slope exceeded!
$v_{out,old}(s_i) \neq v_{out,back}(s_i)$
$v_{out,back}(s_{i+1}) = v_{fuzzy}(s_{i+1})$
last velocity point is fixed
s_{i-3} s_{i-2} s_{i-1} s_i s_{i+1} s
DECELERATION CASE
v_{fuzzy} : Fuzzy Output Velocity
$v_{out,\,old}$: Previous Output Velocity
$v_{out,\,back}$: Backward Corrected O.V.

Figure 5.13: Graphical explanation of the procedure used to limit the acceleration. The fuzzy output, v_{fuzzy}, is modified to match the acceleration constraint. In the deceleration case the backward modification of the fuzzy velocity, $v_{fuzzy}(s_i)$, may propagate to multiple previous steps: $v_{out,back}(s_i), v_{out,back}(s_{i-1})$ etc.

It has been developed a code which examinates all the velocity references at each s point:

- compute the mean velocity $v_{mean,i}$ for the i^{th} step (5.11);
- compute the time step dt_i (5.10);
- compute the acceleration a_i:

$$a_i = \frac{v_{mean,i}}{dt_i} \tag{5.12}$$

- check the acceleration limit: $a_i < a_{lim}$.

If this value, a_i, matches the imposed limits of the algorithm it is possible to proceed with the time characterization of the path, otherwise the time required to compute a step from s_i and s_{i+1} is locally increased to reduce the acceleration. The procedure is reported in [fig. 5.13] with the acceleration and deceleration cases.

The *acceleration limit* is firstly explained since it is simpler with respect to the deceleration one. If the new point s_{i+1} has a time step that does not match the acceleration limit, the time step is simply increased up to the convergence of the acceleration value $a_{i+1} \Rightarrow a_{lim}$. This increment in time requires a consequent reduction of the mean velocity and so of the velocity reference at the point considered (the previous point is considered as fixed).

The *deceleration limit* is a bit more complex, since the computed velocity $v_{fuzzy}(s_{i+1})$ value is fixed and cannot be increased to match the deceleration limit, otherwise it would exceed the limit imposed by the FL (if the FL assigns a velocity, it is assumed that this is the maximum one at which that path feature has to be executed). Starting from the last computed velocity value $v_{fuzzy}(s_{i+1})$, the code proceeds backward step-by-step reducing the previously computed velocities $v_{fuzzy}(s_i)$ till an acceleration convergence. In this way, the right velocity at the last computed point s_{i+1} is guaranteed, and by decreasing the previous velocity values also the mean velocity for each step reduces, till matching of the deceleration limit. This procedure is repeated backward step-by-step, while keeping the previously computed velocities still below the FL limit.

The code with the procedure is shown in [lst. B.2].

5.5 DMP Execution

After the velocity characterization of the FL controller, the original taught path, $\boldsymbol{y}_{des}$, becomes a trajectory with a proper time law, $\boldsymbol{y}_{ref}$, that can now be executed with the DMP framework [chap. 3]. The DMP process can be subdivided in 3 steps:

- DMP *initialization*, which generates the framework with the CS and the basis functions;
- Path *imitation*, in which the weights of the Gaussian basis functions and the forcing action f are defined;
- DMP *rollout*, which generates the final EE reference.

The first two steps are executed offline, while the generation of the TCP reference position to be sent to the robot controller can be computed either offline or online. The former case is suitable when there is no positional feedback from the robot (i.e. the trajectory is computed *a priori* and no collaborative behaviour is implemented), while the latter computes at each time step the reference for the next step, allowing to consider a feedback from the robot.

5.5.1 DMP Initialization

The DMP framework is initialized using 4 parameters:

- Number of DMP;
- Number of basis functions, N_{bfs};
- Time step, dt;
- Runtime, T_{tot}.

The DMP number and the runtime are trivial, since they are equal to the number of DOF (in the presented case 2 DOF: x and y), and to the total duration of the execution, respectively. Instead, the other 2 parameters need some further considerations.

In general, the higher N_{bfs}, the more accurate is the path reproduction, but the computational costs rises with it. The number of basis functions needs to take into account both the most critical feature and the path length; a sharp edge requires a high density of basis functions to be properly defined and conversely, the longer the path, the less dense the basis functions. It must be underlined that the selection of the N_{bfs} must be related also to their variance; if numerous functions are required, their shape can be very narrow, generating a very local action, indeed if a low amount of functions is preferred, they cannot be as narrow as the previous case, otherwise the basis function distribution would leave big gaps that would not characterize all the path.

In this thesis work, N_{bfs} = ***2000*** has been chosen as trade-off between reasonable computational time and accuracy of the reproduction (referred to a path shorter that 1 meter with some straight edges).

The time step has instead to be chosen accordingly with the wished time resolution. Together with the runtime, this parameter defines the number of cycles that the algorithm must perform (i.e. the computational costs). A good choice related to the system dynamic has shown to be ***dt* = *0.01 s***. If the time step is lower and the trajectory is not trivial, it is possible to face problems of small oscillations on the output velocity profile.

5.5.2 DMP Imitation

The imitation procedure refers to the computation of the non-linear forcing actions that are defined through the values of the Gaussian weights. The LWR algorithm has been chosen for the weights evaluation [sec. 3.2], since it is a widely adopted method that allows each kernel to act only close to its center.

In the proposed work, the taught path has been analytically defined, so only one path is fed into the learning algorithm. Indeed, there is no need to average many demonstrations to smooth out path errors, as for human teaching situations.

5.5.3 DMP Rollout

The rollout step refers to the discrete evaluation of the DMP system evolution. In cases of no error dynamic (i.e. there is no feedback with the robot actual state), the trajectory output of the DMP framework, $\boldsymbol{y}_{out}$ can be computed offline in one passage. Otherwise, if it is needed a feedback action on the robot positions, the rollout must be executed in *real-time* step-by-step.

In the online computation, the workstation needs enough computational power to perform the calculus of a discrete step in a time which must be lower than dt.

A simplified version of the Python code which perform these two processes is presented in [app. B]. The DMP discrete step is reported in [lst. B.3], while the error dynamic is executed in the same branch of code which manages the communication with the Franka controller [lst. B.4].

Chapter 6

Experiments and Results

The experimental part of this thesis is mainly subdivided in two steps: first of all the physical setup must be calibrated, then some validation tests are performed in order to show the improved results of the proposed method with respect to the standard DMP approach.

All the calibration procedure is strictly related to the specific test-bed provided [fig. 6.1], but the main approach can be generalized to other applications. One main limitation is related to the dynamics of the robot that cannot be exploited at its nominal values since it would create unacceptable vibrations of the gun nozzle. The maximum allowed acceleration must be addressed before starting the deposition experiments, based on the EE dynamic response. In these tests, the robot is moved back and forth with increasing values of acceleration up to the point in which the caulking gun tip starts vibrating consistently.

This preliminary analysis is firstly performed on a straight line path using a very basic Trapezoidal Velocity Profile [sec. 2.2.1], and then it is verified on a more complex trajectory [fig. 6.2], especially in correspondence of big changes of direction. It has been shown that moving the gun with an acceleration higher than ***30 mm/s***^{2} produces undesirable vibration of the tip, so even if the robot can easily achieve higher linear accelerations, this value has been fixed as limit threshold.

Once the acceleration has been fixed, the analysis can proceed with the tuning of the FL shape functions and of the silicone extrusion flow. As widely explained in [chap. 5], the FL is tuned on standard path features, while the material deposition has been set to be proportional to the EE velocity.

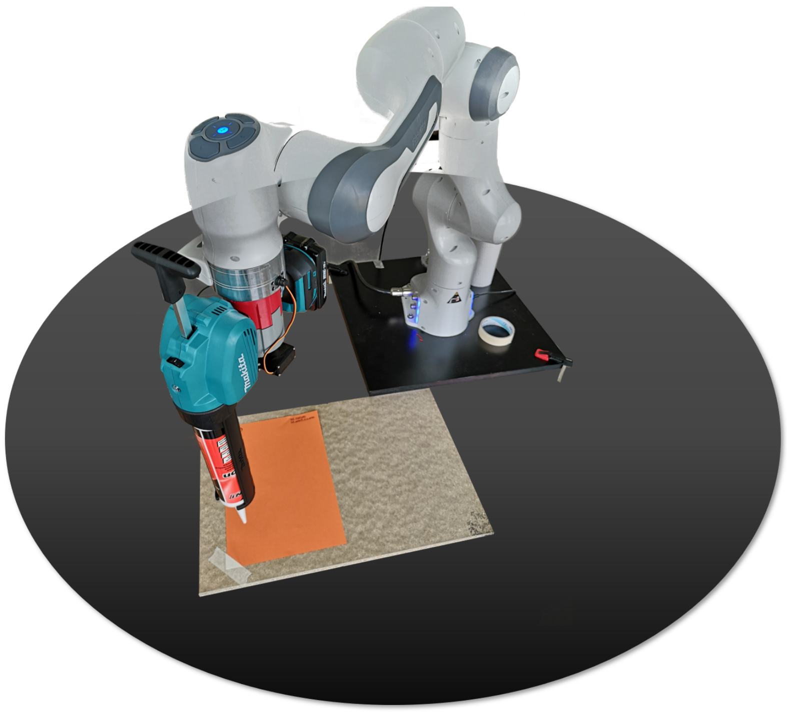


Figure 6.1: Physical test-bed with used workspace

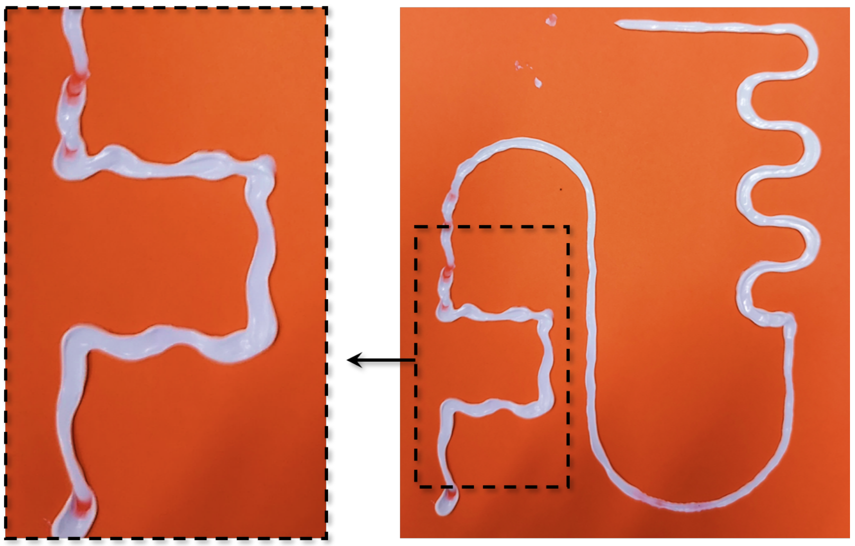


Figure 6.2: Validation of the limit acceleration value. Complete path analysis with $a_{lim} = 80\text{mm}/\text{s}^2$. Clear unwanted vibrations in correspondence of big directional changes

6.1 Tuning of the Fuzzy Logic Controller

The FL must be tuned to assign a proper execution velocity to each path feature. This requires to select some reference geometries (straight lines and different radius curves) and to relate them with an EE velocity.

Four standard features have been selected:

- a straight line;
- a very wide radius curve (radius $\simeq$ 40 mm);
- a middle-radius curve (radius $\simeq$ 25 mm);
- a hairpin curve (radius $\simeq$ 10 mm).

The main problem faced during the FL tuning concerns the steady state flow condition: to have a stable material deposition, it should be created a long path with constant *steering* value. This behaviour is wished since the caulking gun extrusion needs some transitory time to stabilize. While this phenomenon is trivial in case of a straight line, which can be as long as wished, a curve cannot be longer than a complete revolution and steady state condition is not necessarily reachable in one turn. Even with many concatenated curves, in the junction point the *steering* parameter has a variation which leads to a variation of the flow.

Since the steady-state condition cannot be reached in narrow curves, FL tuning is preferred on a complete path, such that it is possible to study not only the single feature execution, but also the deposition in transitory junction points. This approach permitted to tune individually many geometrical features while performing just one trajectory.

6.1.1 Fuzzy Logic Results

The FL membership functions which are optimized for the reference task are shown in [fig. 5.9]. With those shape functions, it is possible to have the experimental input/output correlation which is reported in [fig. 5.11].

Those relations have been obtained with many rollouts of some example paths. For what concerns the straight lines analysis, it has been used the trajectory in [fig. 6.3], where the time-law is a Trapezoidal Velocity Profile [sec. 2.2.1] with v_{max} as tuning parameter. Only the center part of the longer straight line is considered, since the junction lines are executed at a lower speed, resulting in material stack. Lack of material in [fig. 6.3] has

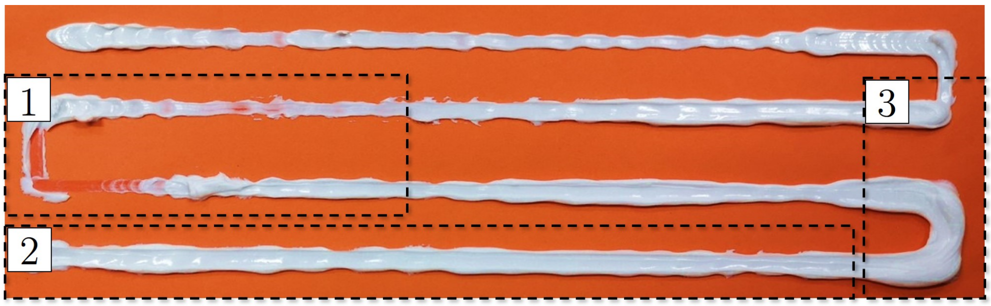


Figure 6.3: Reference path for the characterization of the membership functions related to the straight line. This example shows a lack of deposition in (1) due to nozzle contact with the plane, proper deposition in (2) and excess material in (3).

been addressed to misalignment of the deposition plane, indeed when the nozzle touches the plane the extrusion is blocked.

For the curve analysis, two complete paths have been used, [fig. 6.7d] and [fig. 6.7e] respectively. Those two trails have been studied with the complete algorithm, using the modified DMP framework. The curves have been tuned individually, considering their *steering* value and optimizing the execution velocity.

6.2 Calibration of the Deposition Flow

The silicone extrusion rate must be modulated with respect to the EE velocity in order to avoid material stockpiles when the EE moves at low speed and conversely material lacks when the EE is at full speed. An example is reported in [fig. 6.4].

As explained in [sec. 4.3.1], the caulking gun control is performed through a PWM technique. This is the selected strategy since the caulking gun trigger pression is not able to modulate the silicone extrusion (as was initially expected according to the caulking gun datasheet). The material deposition rate is maximum when the trigger is continuously pushed, while it is minimum if the trigger is pressed intermittently with a very short push. Accordingly with the PWM control strategy, the maximum flow will be referred to *100% duty cycle*, while the minimum flow to *0% duty cycle*.

The gun is designed to be used by a human operator, so it implements a drip catching feature that quickly cuts-off the extrusion when the trigger is fully released. This characteristic is very useful when performing a manual sealing application, but it is cumbersome in this automatic execution, since

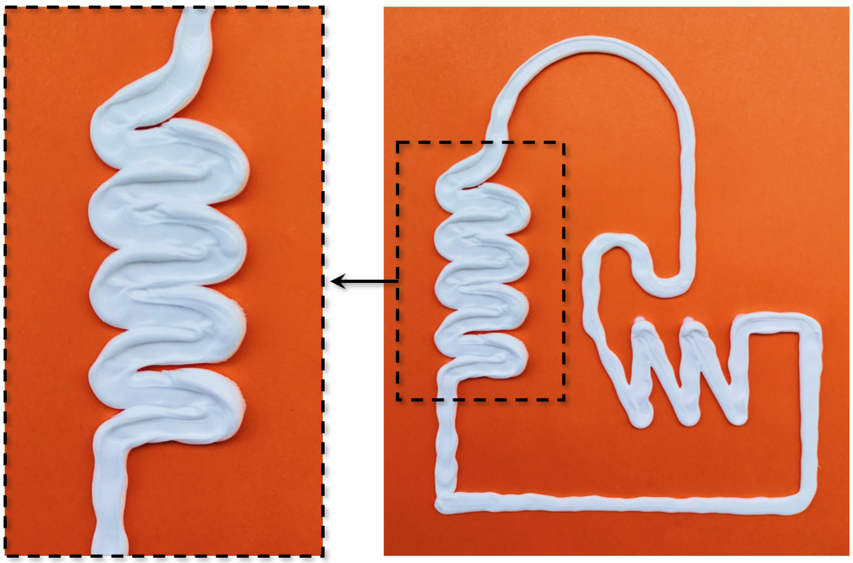


Figure 6.4: Excess deposition in hairpin curves due to extremely reduced speed and unproper material flow. The two parameters must be correlated to find a good extrusion quality.

it instantaneously nullifies the pression inside the cartridge. Experimentally it has been shown that if the trigger is not fully released, the drip catching procedure is not executed, so the *low* level of the PWM square wave has been set maintaining the trigger slightly pressed. Conversely, the *high* level is set with the trigger pressed with an intermediate stroke such that the servomotor has only a limit angular distance to perform (increasing the bandwidth of the system).

It can be assumed that the flow rate is proportional to the transversal movement of the nozzle, but the relation between the trigger pression and the extrusion is highly non-linear. To characterize the EE velocity with a proper material flow, a simple test has been designed: the EE has been moved linearly with a Trapezoidal Velocity Profile [sec. 2.2.1] with a constant PWM *duty cycle*. Only the middle part of the line is considered, neglecting boundary effects related to the acceleration stages.

This experimental analysis required a lot of trials, so only a small part is presented in [fig. 6.5] with the related test data shown in [tab. 6.1].

The executed procedure is always the same, but due to the poor repeatability of the caulking gun, the same parameters were tested many times to average the results and smooth out the wrong depositions.

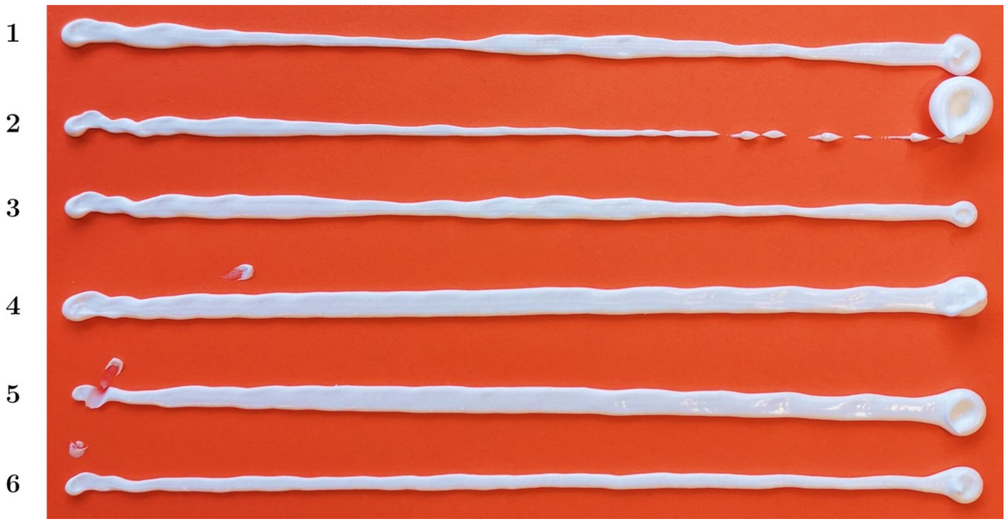


Figure 6.5: Deposition flow rate tuning procedure, depending on EE velocity and duty cycle. Testing parameters are shown in [tab. 6.1]

Test	Duty cycle [%]	v_{cruise} [mm/s]	Notes
1	80	50	Almost finished cartridge
2	80	50	Almost finished cartridge
3	90	50	Unstable flow
4	80	50	Target quality reached - flow set
5	90	65	Target quality reached - flow set
6	100	80	Correct flow but wrong vertical offset

Table 6.1: Tuning parameters and notes of experiments shown in [fig. 6.5]

6.2.1 Flow Results

The experimental correlation between the EE velocity and the silicone flow is obtained with a linear interpolation of the test points. Five different velocity values have been evaluated, with the addition of the null flow condition (achieved with near-zero duty) at null speed:

$$\boldsymbol{v}_{vect} = [0, 20, 35, 50, 65, 80] \quad \frac{mm}{s} \tag{6.1}$$

The results have been collected in [tab. 6.2], while a graphical representation of the correlation is shown in [fig. 6.6].

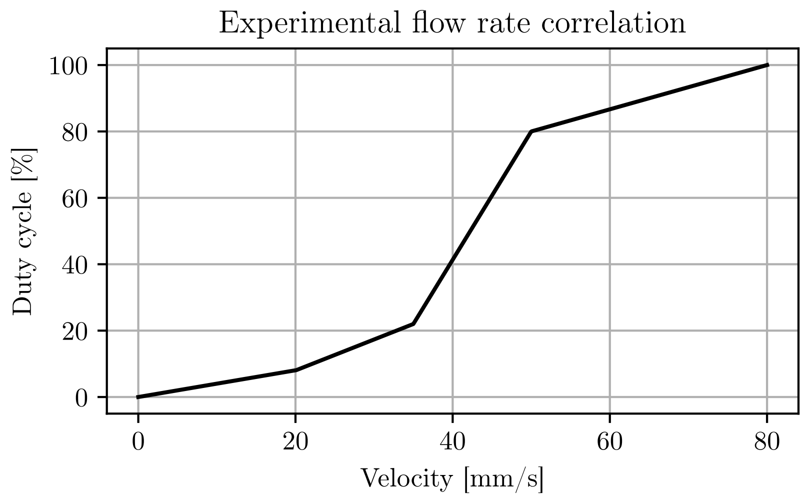


Figure 6.6: Linear interpolation procedure used to obtain a continuous relation between velocity and flow duty cycle

EE Velocity [mm/s]	0	20	35	50	65	80
Duty Cycle [%]	2%	8%	22%	80%	90%	100%

Table 6.2: Duty cycle of the PWM modulation that controls the silicone extrusion, as function of the EE velocity

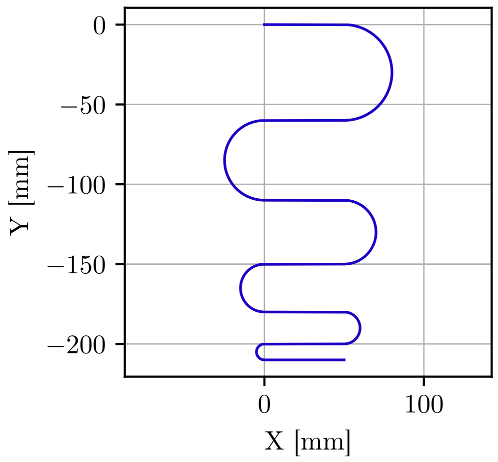
6.3 Testing Paths

Once both the feature velocities and the related deposition flow have been addressed, the algorithm must be experimentally validated on complete trajectories. The validation procedure has been carried out on different paths [fig. 6.7], comparing the standard DMP execution proposed by Ijspeert and Schaal [4] with the modified approach introduced by this thesis work.

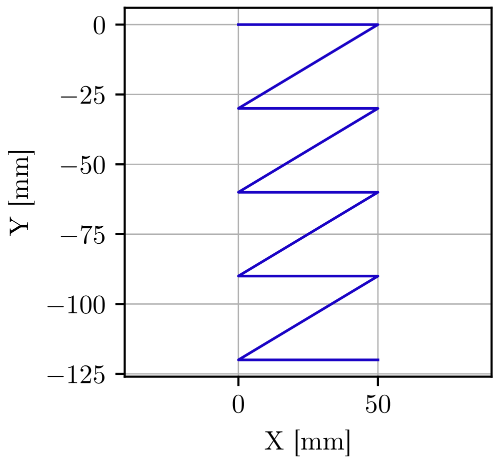
Each path has been generated to create some challenges for the planning algorithm:

1. **Different radius curves** [fig. 6.7a]: the curve radius is progressively reduced, so that the last curves (small radius) must be executed with a slower reference velocity with respect to the initial ones (larger);

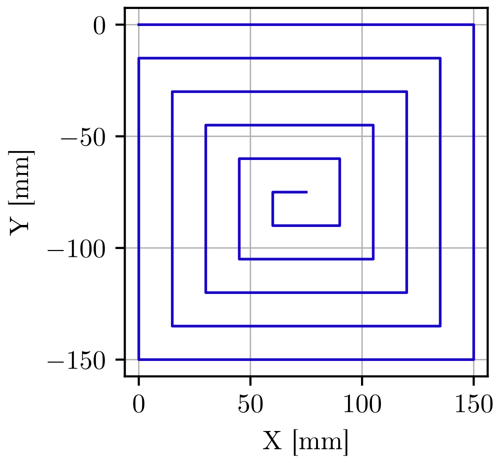
2. **Saw tooth** [fig. 6.7b]: the corners have a sudden directional change, which allows to test the algorithm against this sort of discontinuity;



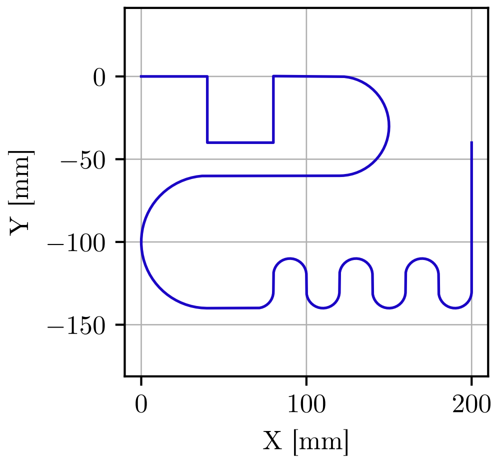
(a) Different radius curves



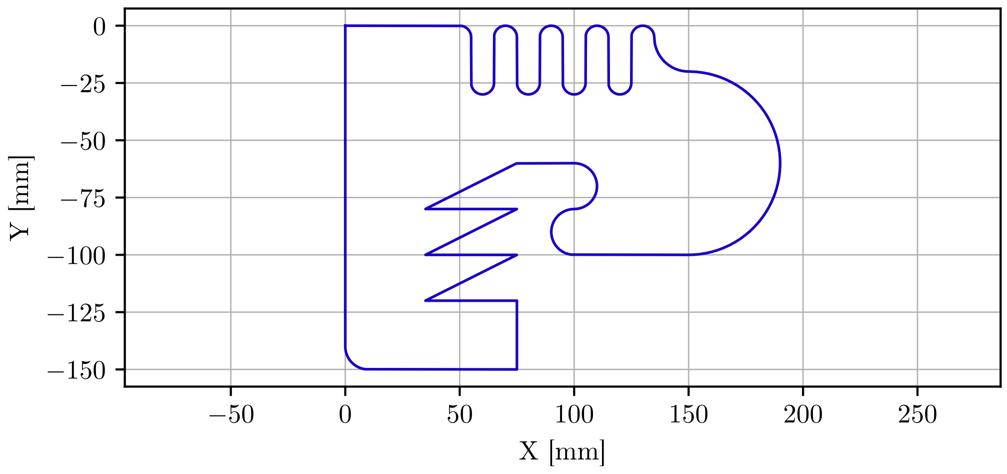
(b) Saw tooth



(c) Square spiral



(d) Generic path 1



(e) Generic path 2

Figure 6.7: Different proposed paths

3. **Square spiral** [fig. 6.7c]: the main feature of the square spiral is that the straight edges get closer while the spiral converges. This results in shorter straight lines and consequently a lower acceleration time;

4. **Generic path 1 and 2** [fig. 6.7d, 6.7e]: complete trajectories that contain a general sequence of standard features.

6.4 Results

During the experimental session, some issues related to the setup were found. Here a list of the major problems and corresponding solution is reported:

- The collaborative nature of the robot implies soft position tracking: each joint, even with a good tracking control, results to be a little movable if some external force (as example, a manual push of the operator on the robotic arm) acts on it, with the consequence of reduced precision compared to an industrial robot;

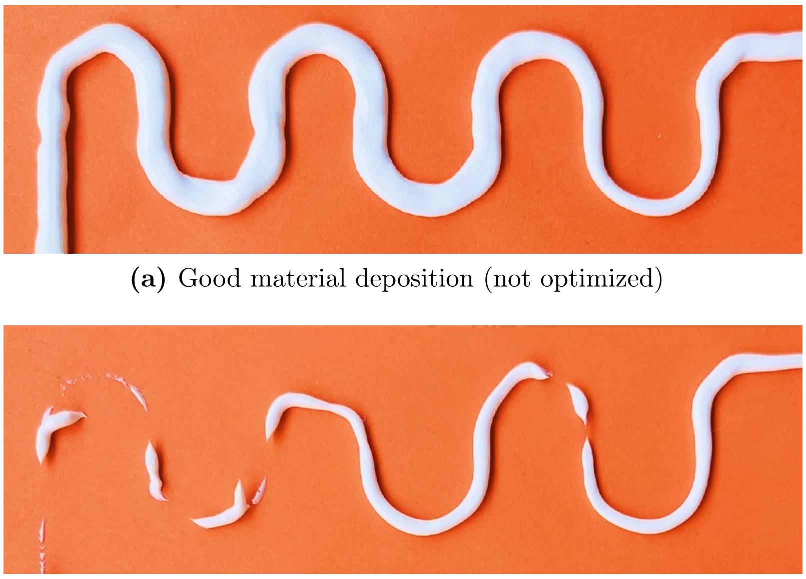
- The workspace was not stiffly connected with the robot base, resulting in some small misalignments between different runs;

- The manual caulking gun has very poor repeatability: with the same code and parameters, the results may significantly change: [fig. 6.8] shows two subsequent runs of the same algorithm with the same tuning parameters; the big difference in result is probably due to the commercial caulking gun commanded with a force on the piston instead of an extrusion velocity and to the cartridges;

- The silicone cartridge flow depends on the residual amount of material inside the tube: when a cartridge is new, the required force is different with respect to the case in which the cartridge is almost finished.

Improved deposition performances can be achieved with an ad-hoc extrusion system that deposits a homogeneous material with a flow velocity control. A general algorithm has been studied in this thesis, such that it can be implemented either on more economic test-beds, as the used one, or on expensive task-designed setups.

Despite the previously explained issues, some interesting results were achieved. The proposed approach permits to execute the path with variable velocity, allowing to speed-up the EE in straight lines while reducing



(a) Good material deposition (not optimized)

(b) Unexpected flow reduction

Figure 6.8: Comparison of two material depositions with the same parameters: unexpectedly the flow is completely different

the velocity accordingly with curves. This modulated behaviour is not achievable with standard DMP, where it is possible to choose to perform the path with a more or less constant mean velocity function of the imposed runtime parameter. If one chooses a mean velocity suitable for straight lines, it will result in vibrations when performing a curve; vice versa, if the reduced velocity referred to curves is chosen, the total time required for the path execution rises a lot, with a subsequent higher production cost due to non-optimal time managing.

The main difference between the modified DMP approach and the standard one lies in the movement definition. For standard DMP, trajectory planning is defined accordingly with the target execution time and then a mean velocity for performing the path is naturally computed. In the proposed approach time is a consequence of the planned velocity along the path, such that it is possible to optimize the execution accordingly with the reference target velocity instead than fixing the runtime.

It must be noticed that both for the DMP standard approach and the novel algorithm, the extrusion rate of material is set as proportional to the EE velocity, as explained in [sec. 6.2.1]. This would be a further improvement compared to the standard DMP approach executed with constant flow rate, but the performed experiments are aimed to compare the velocity planner instead than the extrusion process, so the material flow deposition has been performed in the same way for both the experiments.

6.4.1 Experimental Validation

The validation experiments have been carried out on the paths shown in [sec: 6.3]. Each path has its own characteristics, but the generality of the proposed algorithm permits to execute all those tests without changing any model parameters.

Different Radius Curves

This path has been proposed to show the velocity planning against different radius curves [fig. 6.9]. While the standard approach executes all the path at constant speed, the proposed algorithm starts with high velocity (in the large curves) and progressively slows while approaching the end of the path (small radius curves) [fig. 6.10].

In [fig. 6.11], the PWM wave is shown. It is visible that the higher the velocity, the higher the *duty cycle*. Between $1 \div 2$ s there is a *duty cycle* transition related to the velocity increment.

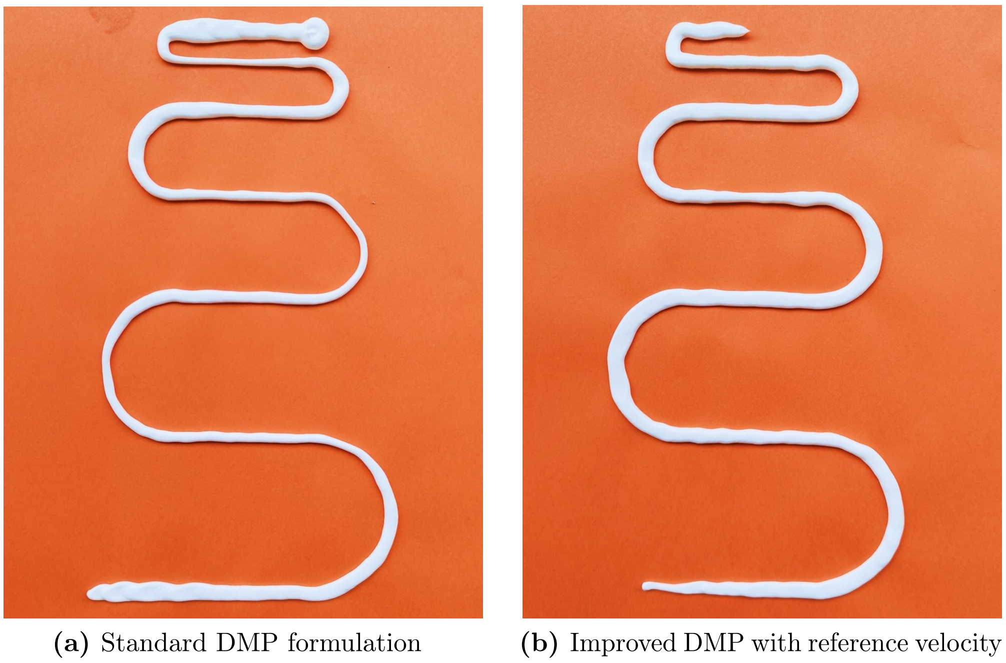

(a) Standard DMP formulation **(b)** Improved DMP with reference velocity

Figure 6.9: Experimental test on Path 1: different radius curves

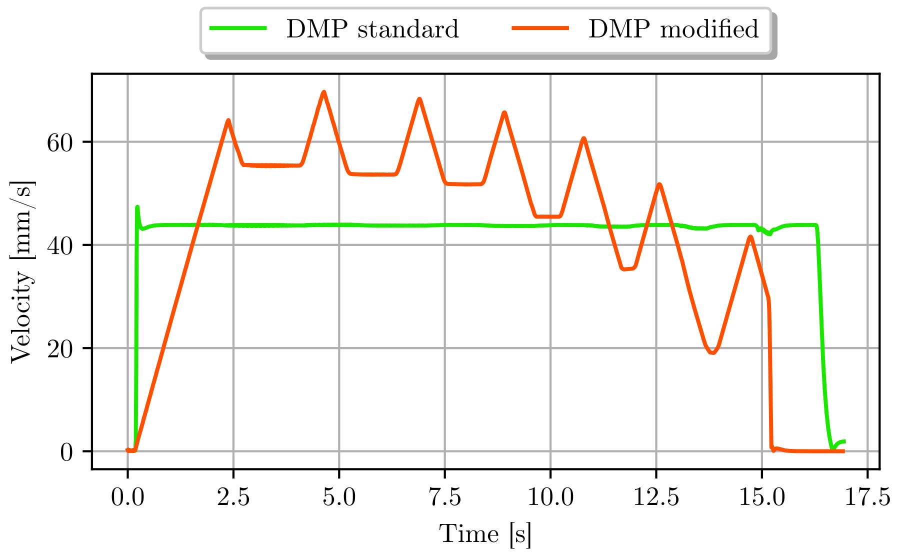
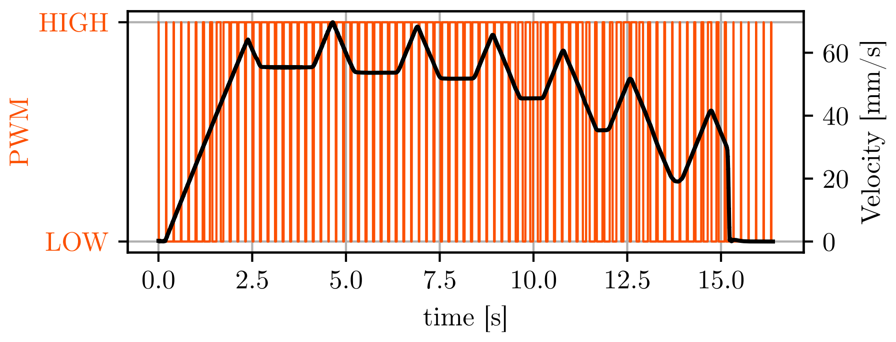
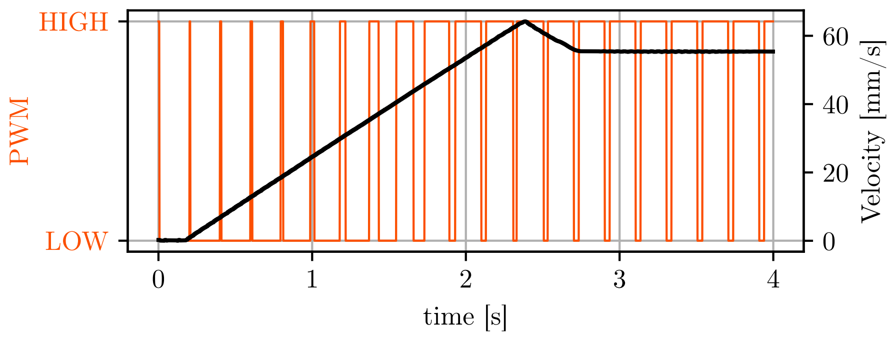


Figure 6.10: Velocity profile comparison for Path 1



(a) Example of PWM duty cycle on Path 1



(b) Magnification of the first 4 s of **(a)**

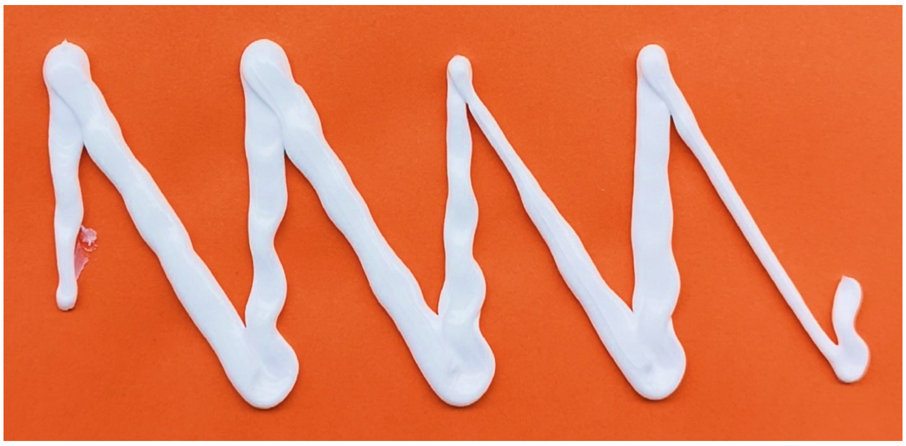
Figure 6.11: PWM Duty cycle, related to execution of Path 1

It has been experienced that for this specific application the final result is almost the same since there are no big discontinuities and directional changes, but the modified approach shows a more uniform deposition flow.

Saw Tooth

This path is more critical to be executed since it presents many sudden directional changes [fig. 6.12]. Path discontinuities are the most difficult features to be executed, since they generate consistent vibrations on the EE tip and they must be faced at reduced speed [fig. 6.13].

The proposed algorithm manages the reduced velocity at the edges, presenting many improvements with respect to the basic DMP, where no speed reduction is assessed and it is generated a poor deposition.


(a) Standard DMP formulation

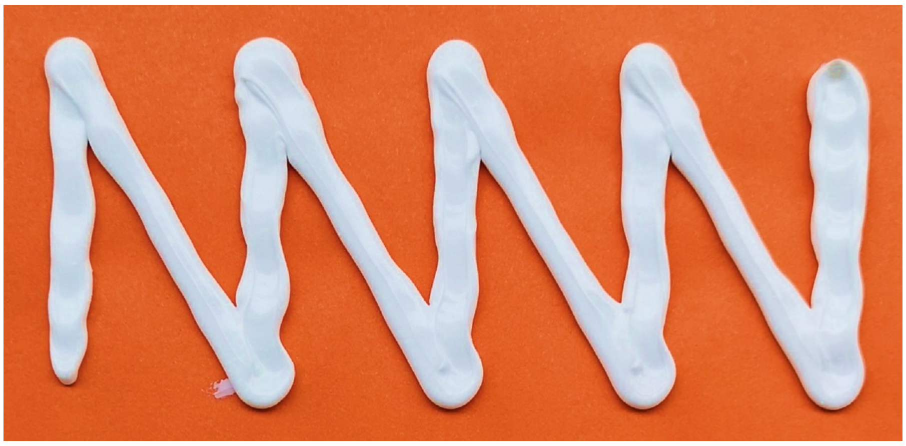

(b) Improved DMP with reference velocity

Figure 6.12: Experimental test on Path 2: saw tooth

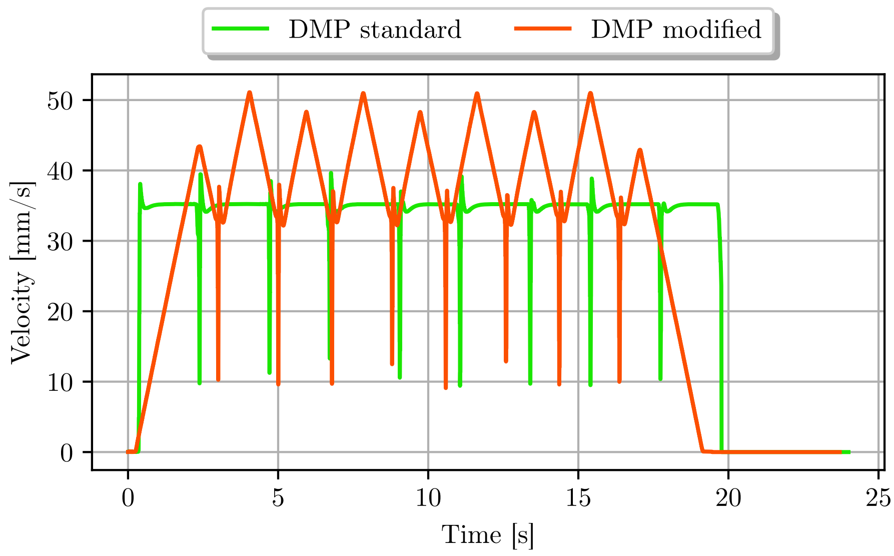
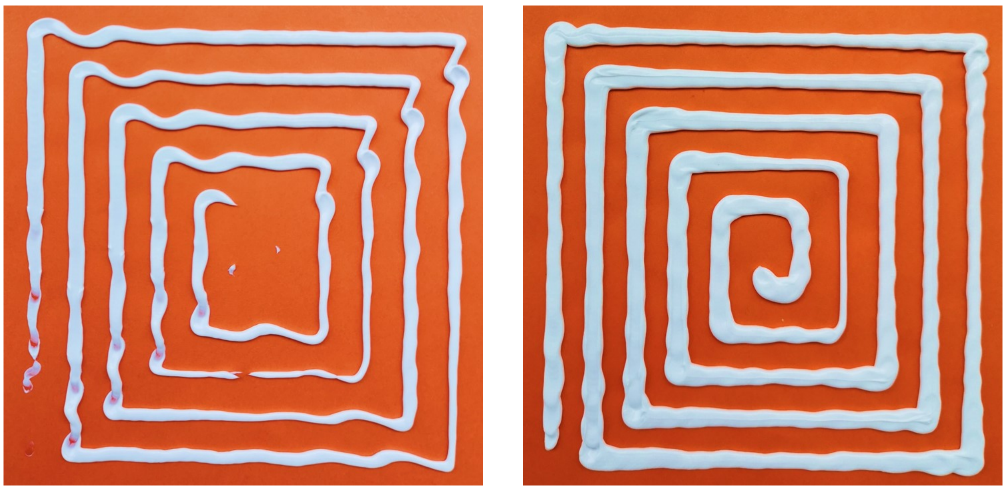


Figure 6.13: Velocity profile comparison for Path 2

Square spiral

Path 3 has been selected to analyse the flow management over a quite long path with repeated sudden directional changes [fig. 6.14].



(a) Standard DMP formulation **(b)** Improved DMP with reference velocity

Figure 6.14: Experimental test on Path 3: square spiral

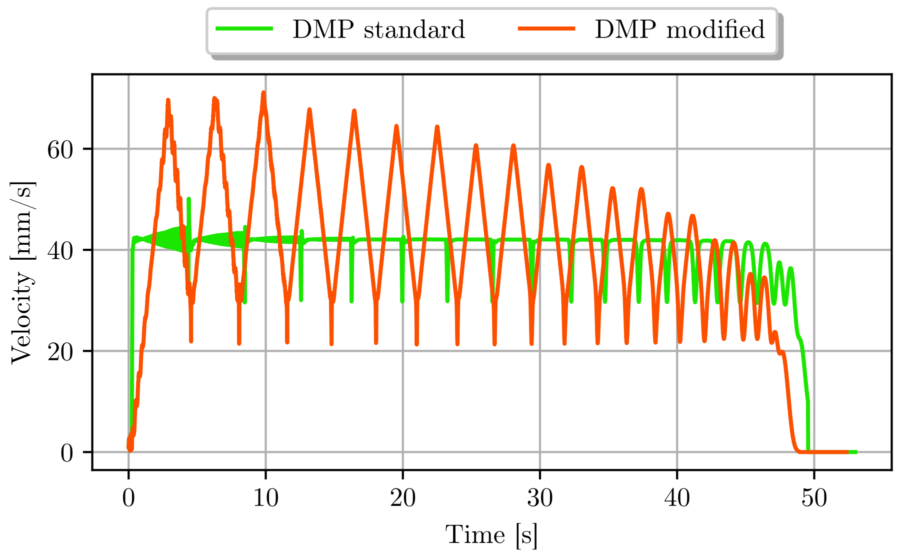


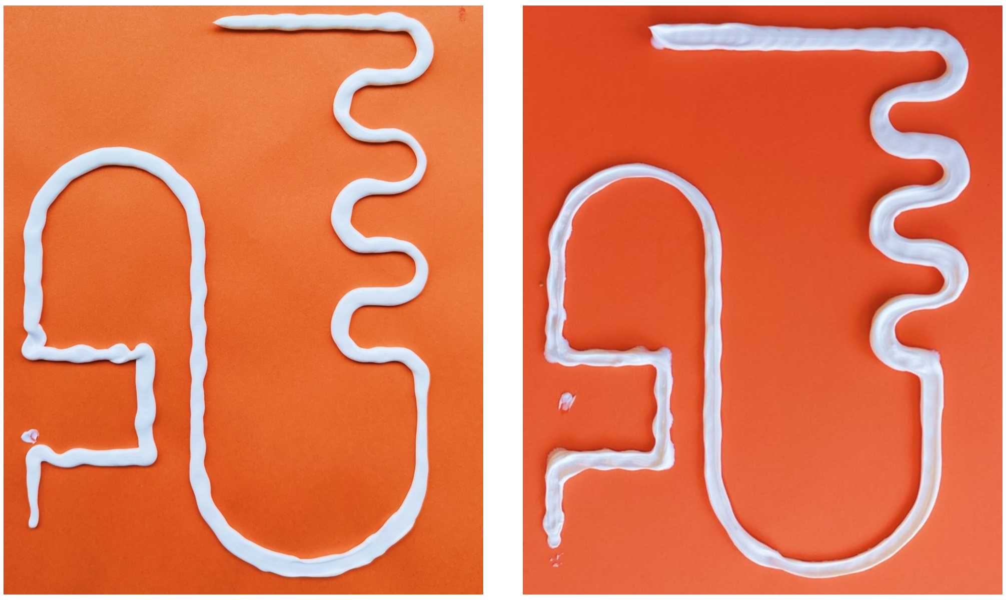
Figure 6.15: Velocity profile comparison for Path 3

The standard DMP execution is forced to work with the same runtime of the modified DMP framework. It is visible that without the slowing down procedure, major vibrations happen in correspondence of the square edges and there is also a non-uniform sealant deposition between longer and shorter sides.

Generic Path 1

This path was selected to analyse the ability of the algorithm to execute different features in series and to control the flow deposition [fig. 6.16]. The standard DMP executed at a properly tuned runtime manages to perform a good deposition but not optimal in the hairpin curves. Instead, the proposed approach reaches a more homogeneous distribution along all the path.

A major flaw related to standard DMP is that an unproper selection of the runtime would lead to too high or too low speed, causing a low-quality result. Defining the correct execution time is more difficult than tuning the velocity reference profiles, since there is no modelled relationship between time and path complexity.



(a) Standard DMP formulation (b) Improved DMP with reference velocity

Figure 6.16: Experimental test on the generic Path 4.1

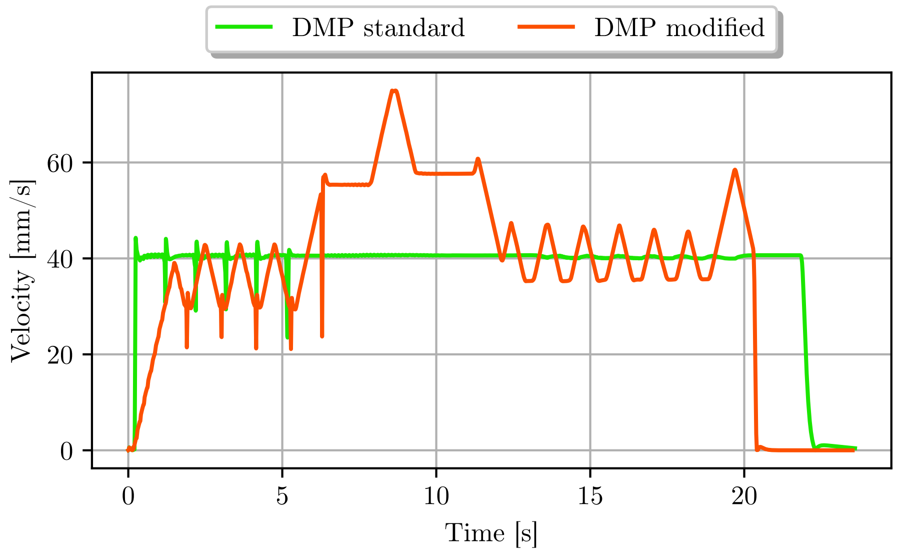
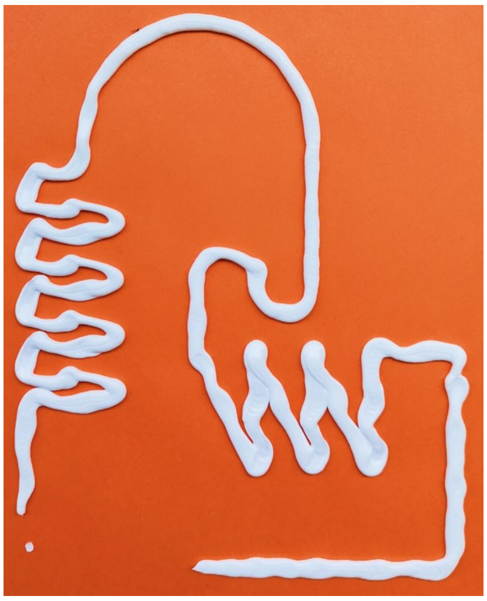


Figure 6.17: Velocity profile comparison for Path 4.1

Generic Path 2

This last example tests the algorithm on a trajectory that includes all the previously examined features [fig. 6.18]. The final execution of the standard DMP shows big vibrational problems on the junction points.


(a) Standard DMP formulation

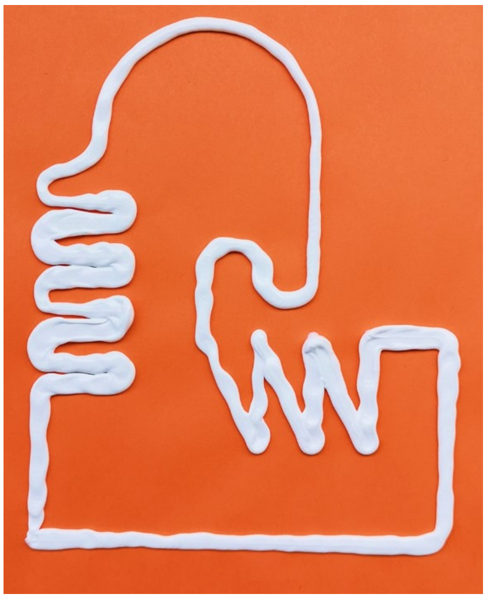

(b) Improved DMP with reference velocity

Figure 6.18: Experimental test on the generic Path 4.2

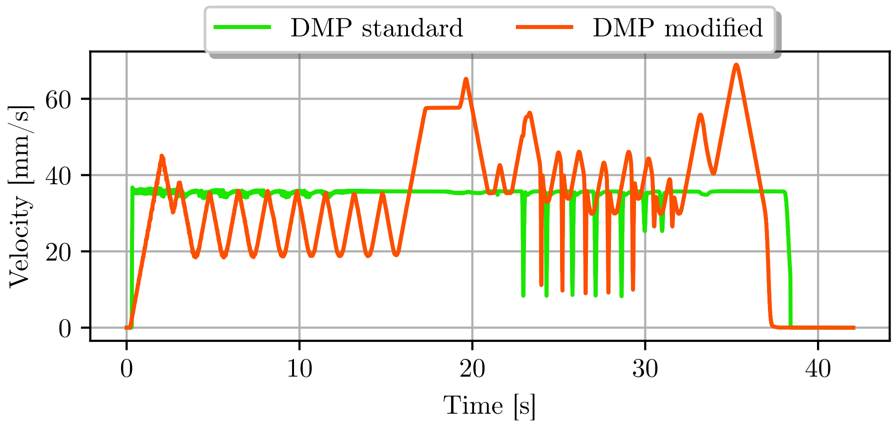


Figure 6.19: Velocity profile comparison for Path 4.2

Chapter 7

Conclusions

This thesis work proposes a novel approach to trajectory planning, using an automatic velocity characterization that showed an enhanced task execution, permitting to reduce the total task time without compromises about the sealing quality.

The developed algorithm generates a punctual velocity reference using a FL controller, which requires a one-time tuning based on the physical setup before starting the effective production. As presented in [chap. 6], some preliminary experiments are needed to well define both the velocity reference as function of the path geometry and the extrusion flow rate. Once this procedure is accomplished, the human operator can directly feed whatever path to the planning algorithm, without caring about the velocity characterization that is instead automatically computed by the proposed controller.

While the standard DMP approach requires a demonstration both in terms of space and time characterization, the proposed modified planner deals with only the geometrical path automatically assigning a time-law optimized on the execution velocity. If well-tuned, this approach permits to execute the task in the fastest possible way that guarantees the best achievable quality, without needing to manually tune the total execution time. Instead, using a standard DMP framework requires to manually tune the runtime and consequently the execution is performed with a more or less constant velocity value, which does not exploit the full velocity in straight lines and conversely may cause too high velocities across tight curves.

The experimental testing procedure validates the presented thesis work: the algorithm defines a velocity reference as function of the path geometry, allowing a good execution of tight curves and a fast travelling time when performing straight lines. Even with the complications of the used test-bed, the results showed an improvement with respect to the standard DMP approach (which executes the taught path at constant speed). Examples of the enhanced quality are less vibrations when performing sudden directional changes, more homogeneous sealant deposition and smoother acceleration.

The choice of FL as velocity planner has been experienced to be successful in the reference test-bed, since the high non-reproducibility of the setup and the uncertainties of the model do not allow to use more refined optimization algorithms. Indeed, the velocity correlation required a fully experimental definition.

The selection of DMP as motion planning algorithm is related to the large amount of possible further implementations of the method, starting from collaborative adaptation with obstacle avoidance and disturbance rejection, which can be obtained with on-line implementation of the algorithm, to learning of multiple trajectories. Moreover, the DMP works as a filter on the taught trajectory and gives continuity to the velocity reference generated by the FL controller.

This method allows future improvements of the planning procedure, for example with the introduction of vision feedback systems either to automatically acquire the path, or to improve the quality through an online feedback signal. Moreover, it can be implemented a direct path definition through Computer Aided Design (CAD) drawings, which would result in further time saving.

Finally, even if all the procedure has been optimized on a sealing task, the framework is general and can be adapted to any other industrial procedure. The here-presented algorithm works in 2D space, but it is trivial to extend it to more DOF considering the vertical displacement and the EE rotation.

Appendix A

Flange Construction

In this appendix there is a material list and some construction tips that may be useful for anyone who wants to build up the same setup used in this thesis, which is exhaustively explained in [chap. 4]. A Bill of Materials (BOM) of the components needed to build the flange [fig. A.1] is provided, together with some drafts of electrical circuit and the code which runs on Arduino.

A.1 List of Used Materials

The BOM includes both commercial components **[C]** and 3D printed parts **[3D]**:

- **[3D] Base Flange**, that is directly connected to the robot EE;
- **[3D] Top Flange**, which closes on the gun handle and by means of long screws, it clamps the gun. Both the motor and the mechanism have been inserted inside this component;
- **[3D] Mechanism**: it is composed by a rack and a pinion. The pinion is mounted on the motor standard flange (which needs to be cut to adapt it to the pinion);
- **[C] Arduino Nano**: electronic controller of the system. It receives as input the potentiometer and controls in output the servomotor;
- **[C] MG995 (high speed)**: servomotor with a stall torque of 0.8 Nm. It can be directly fed by an Arduino board;

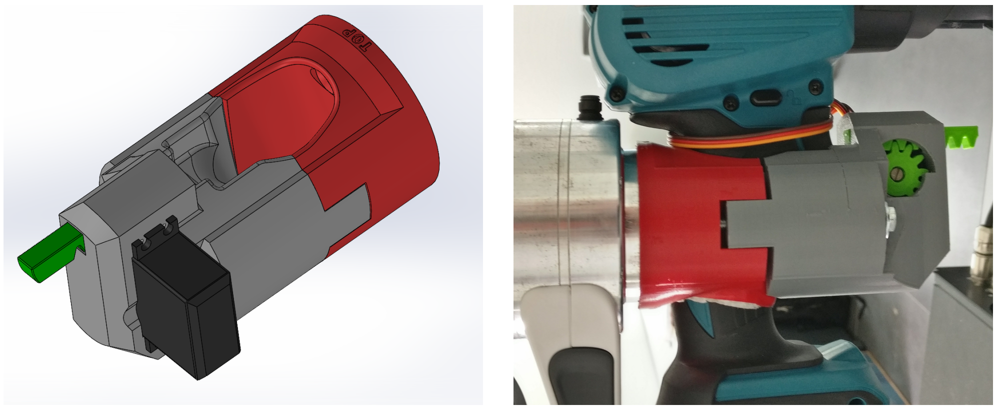

(a) Developed flange (CAD) (b) Real component

Figure A.1: Sketches of the custom flange

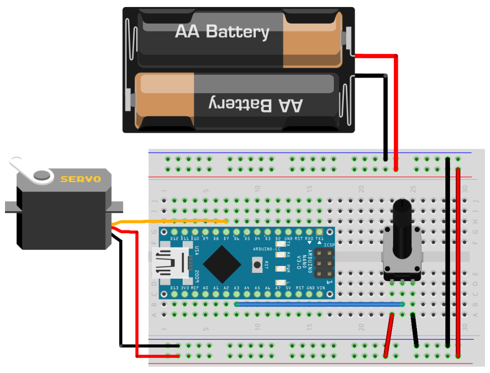
- **[C] Potentiometer**: any kind can be used, preferrable a linear one instead of logarithmic. The full scale of the potentiometer must be set in order to match the Arduino A/D converter;
- **[C] M6x100**, 2 hex head screws that keep all the flange together;
- **[C] M6x20**, countersunk head screw, used to fix the base flange to the robot EE;
- **[C] M6x15**, countersunk head screw, used to fix the base flange to the robot EE;
- **[C] M3x8**, screwdriver slotted screw, that fix the pinion to the motor. The pinion is directly sustained by the motor shaft.

All the CAD drawings that are needed to print the custom flange are provided as STereoLitography (STL) files, available at:

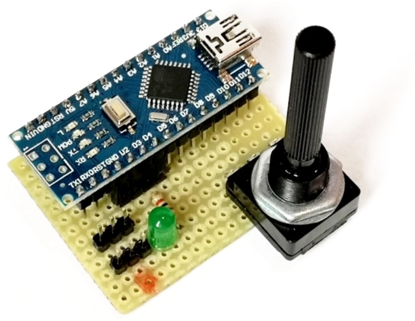
https://github.com/emarescotti/VelocityPlanning_DMP_FL/tree/main/Flange_CAD_drawings

A.2 Trigger Controller

The gun trigger can be controlled in two ways: manually through a potentiometer, or automatically feeding the Arduino serial port directly with the Python ROS node.



(a) Scheme of the circuit on the breadboard



(b) Real circuit

Figure A.2: Electrical circuit

In the first case a knob is used to choose the extrusion rate between no-pressed trigger and fully pressed trigger, while in the second case an integer number is written in the Arduino serial port (with a fixed publication rate): the input range $[0 \div 99]$ is correlated with the two limit positions of the trigger.

The potentiometer is always connected, so that it is possible to choose between the two possibilities at the beginning of the Arduino code. The electrical circuit is shown in [fig. A.2].

The Arduino code is finally reported:

Listing A.1: (Arduino) Caulking gun controller code

```
#include <Servo.h>
Servo motor;

const int pinServo = 4;
const int pinPot = A1;
const int pinLed = 3;    // just for debug

int pot = 512;      // potentiometer reading
String text = ""; // string value from the serial
    port
int alpha = 0;      // gun trigger pression [0-9]
int muS = 1400;     // microseconds used as motor
    input

bool MODE = 1;      // CHOOSE: 0 = potentiometer
```

```
                         //          1 = serial input
void setup(){
  Serial.begin(115200);
  motor.attach(pinServo);
  motor.writeMicroseconds(muS);
  }

void loop(){
  if(MODE==0) { // POTENTIOMETER
    pot = analogRead(pinPot);
    muS = map(pot,20,1000,1400,1930); //
       interpolation
    motor.writeMicroseconds(muS);
    delay(10);                          // pause
    }
  else {          // SERIAL READING
    while (Serial.available() > 0) {
      char inChar = Serial.read();
      text += inChar;

      // after reading the data, command the motor
      if (inChar == '\n') {
        alpha = text.toInt();
        muS = map(alpha,0,100,1400,1930);
        motor.writeMicroseconds(muS);
        text = ""; // clear for next input
        }
      }
    }
  }
```

Appendix B

Developed Listings

It has been decided not to report here all the developed listing, since it would have been very long and not so meaningful for the thesis purposes. Anyway, for a full view of the code used, a repository in which there is all the developed listing is provided:

https://github.com/emarescotti/VelocityPlanning_DMP_FL/tree/main/Python_code

The most significant parts, which may help explaining some mechanisms of the algorithm, are instead reported here. Each part is widely explained in the chapter of interest, while here is reported only an idea of the related context.

B.1 Code: Steering Computation

The following code refers to the computation of the *steering* parameter [sec. 5.2], which characterize the curvature of each path features:

Listing B.1: (Python) Function for *steering* computation

```python
def steeringAngle(path,body_length):
    # check that "path" is a column array
    if np.size(path,1) > np.size(path,0):
        path = path.T
    # intialization of vectors
    angle = np.zeros(int(body_length/2))
    steering = np.array([0])
    last = 0 # for unwrapping the angle profile

    for i in range(int(body_length/2),int(
    np.size(path,0)-body_length/2)+1):
      a = path[i+int(body_length/2),1] -
        path[i-int(body_length/2),1]
      b = path[i+int(body_length/2),0] -
        path[i-int(body_length/2),0]
      c = np.arctan2(a,b) # angle computation

      # unwrapping procedure
      while c < last - np.pi: c += 2*math.pi
      while c > last + np.pi: c -= 2*math.pi
      last = c

      angle = np.append(angle,c) # data saving

  # fix first values
  angle[0:int(body_length/2)] = np.ones(int(
    body_length/2))*angle[int(body_length/2)]

  # restore correct length of the 2 vectors
  angle = np.append(angle,np.ones(
    int(body_length/2))*angle[-1])

  for i in range(1,np.size(path,0)):
      steering = np.append(steering, angle[i] -
        angle[i-1]) # finite differences

  return angle, steering
```

B.2 Code: Acceleration Limit

The following listing explains the implementation of the acceleration limit [sec. 5.4]. The code recursively changes the velocity (lowering it), up to convergence of the actual acceleration to the limit value. A lower velocity value generates a longer time needed for the spatial step, which consequently also lowers the acceleration in that step.

Listing B.2: (Python) Procedure to limit the acceleration

```
# THIS CODE IS INSERTED INSIDE THE VELOCITY OUTPUT
   FROM FUZZY, SO "v_ref[-1]" REFERS TO THE LAST
   COMPUTED VELOCITY VALUE
```

```

# evaluation of the instantaneous acceleration
t_inst = 2 * ds / abs(v_ref[-1] + v_out)
acc_inst = (v_out - v_ref[-1]) / t_inst
if abs(acc_inst) > acc_lim:
    if acc_inst > 0: # ACCELERATION
        while acc_inst > acc_lim:
            v_out -= 0.1
            t_inst = 2 * ds / abs(v_ref[-1] + v_out)
            acc_inst = (v_out - v_ref[-1]) / t_inst

        v_ref = np.append(v_ref,v_out)
        t_ref = np.append(t_ref,t_ref[-1]+t_inst)
        acc_ref = np.append(acc_ref,acc_inst)

    else: # DECELERATION
        v_ref = np.append(v_ref,v_out) # last value
        t_ref = np.append(t_ref,t_ref[-1] + t_inst)
        acc_ref = np.append(acc_ref,acc_inst)
        back = 0 # counter of back instants
        while back >= 0:
            t_inst = 2 * ds / abs(v_ref[-2-back] + v_ref[-1-back])
            acc_inst = (v_ref[-1-back] - v_ref[-2-back]) / t_inst
            while abs(acc_inst) > acc_lim:
                v_ref[-2-back] -= 0.1
                t_inst = 2 * ds / abs(v_ref[-2-back] + v_ref[-1-back])
                acc_inst = (v_ref[-1-back] - v_ref[-2-back]) / t_inst
# changing the 2nd last velocity, the last 2 time instants will be longer (UPDATE)
            t_back = 2 * ds / abs(v_ref[-3-back] + v_ref[2-back]) # dt of 2nd last step
            shift_back = t_back - (t_ref[-2-back] - t_ref[-3-back]) # time shift due to change in velocity, of the previous time step
            shift = t_inst - (t_ref[-1-back] - t_ref[-2-back]) + shift_back # time shift
```

```
               due to change in velocity, of the
               actual time step

            t_ref[-1] += shift
            t_ref[-1-back:-1] += shift
            t_ref[-2-back] += shift_back

            acc_ref[-1-back] = acc_inst
            acc_ref[-2-back] = (v_ref[-2-back]-v_ref
               [-3-back]) / t_back
# check on acc at prev time instant
            if abs(acc_ref[-2-back]) < acc_lim:
                back = -1
            else: back += 1
else: # BASE CASE (LIMIT IS NOT REACHED)
    v_ref = np.append(v_ref,v_out)
    t_ref = np.append(t_ref,t_ref[-1] + t_inst)
    acc_ref = np.append(acc_ref,acc_inst)
```

B.3 Code: DMP Step

A single step of the DMP *rollout* [sec. 5.5.3] is here presented. This function is recalled each time-step during the online execution of the task. This version has already implemented an error dynamic, while without it the equations simplifies a lot.

Listing B.3: (Python) Single DMP step

```
def step(self,ya,error=0.0,tau=1.0):
    # Run the DMP system for a single timestep
    self.ae = 5.0
    kp = 35.0
    kv = 8.0
    kc = 0.5
    tau = 1.0

    # adaptive time constant (considering the error)
    tau_adapt = tau*(1+(kc*error**2))
    # generate basis function activation
    psi = self.gen_psi(self.cs.x)

    for d in range(self.n_dmps):
        self.dya[d] = (ya[d] - self.ya_old[d]) / self.dt
        self.ya_old[d] = ya[d]

        # Feed Forward Control on ACTUAL ROBOT MOVEMENT
        self.ddyr[d] = kp*(self.yc[d]-ya[d]) + kv*(self.dyc[d]-self.dya[d]) + self.ddyc[d];

        # generate the forcing term
        f = self.cs.x*(self.goal[d]-self.y0[d])*(np.dot(psi,self.w[d]))/np.sum(psi)

        # z coupling
        self.dz[d] = 1/tau_adapt * (self.ay[d] * (self.by[d] * (self.goal[d]-self.yc[d])-self.z[d]) + f)

        self.z[d] += self.dz[d] * self.dt

        self.dyc[d] = self.z[d]

        self.ddyc[d]= (self.dz[d] * tau_adapt - tau*self.z[d]*2*kc*error*(self.ae*(ya[d]-self.yc[d]-error)))/tau_adapt**2

        self.yc[d] += self.dyc[d] * self.dt

    _ = self.cs.step(tau=tau_adapt)

    return self.yc, self.dyc, self.ddyc, self.ddyr
```

B.4 Code: ROS Node

The Python ROS node which publishes the commanded position to the robot controller and receives the robot state (actual position), is here reported.

Listing B.4: (Python) ROS node which communicates with the robot controller

```python
p = PoseStamped()

def callback(Pose):
    global s
    s = Pose

pub = rospy.Publisher('/DMP_pose', PoseStamped, queue_size=1)
sub = rospy.Subscriber('/franka_ee_pose', Pose, callback)

rospy.init_node('DMP_planner', anonymous=True)
rate = rospy.Rate(1/dt) # 100 Hz
task_ended = False

while not rospy.is_shutdown():
    if not task_ended:
        for t in range(timesteps):

            ytrack[0] = x_off - s.position.x * 1000.0 # [mm]
            ytrack[1] = y_off - s.position.y * 1000.0 # [mm]

            # run and record timestep
            yc, dyc, ddyc, ddyr, tau_adapt = step(ya=ytrack,error=error_now)

            for i in range(n_dmps):
                dyr[i] += ddyr[i] * dt
                yr[i] += dyr[i] * dt

            error_now += dmp.ae * (np.sqrt((ytrack[0]-yc[0])**2 + (ytrack[1]-yc[1])**2) - error_now) * dt
            error = np.append(error,error_now)

            x = (-yr[0] + x_off) / 1000.0 # [m]
            y = (-yr[1] + y_off) / 1000.0 # [m]

            p.pose.position.x = round(x,5);
            p.pose.position.y = round(y,5);

            pub.publish(p)
            rate.sleep()

        task_ended=True
    pub.publish(p)
    rate.sleep()
```

Bibliography

References quoted in the text

[1] L. Sciavicco and B. Siciliano. *Modelling and control of robot manipulators.* Springer Science & Business Media, 2012.

[2] B. Siciliano et al. *Robotics: modelling, planning and control.* Springer Science & Business Media, 2010.

[3] S. Schaal et al. 'Learning movement primitives'. In: *Robotics research. the eleventh international symposium.* Springer. 2005, pp. 561–572.

[4] A. J. Ijspeert et al. 'Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors'. In: *Neural Comput.* 25.2 (2013), pp. 328–373.

[5] R. Kelly, V. S. Davila, and J. A. L. Perez. *Control of robot manipulators in joint space.* Springer Science & Business Media, 2006.

[6] R. Galin and R. Meshcheryakov. 'Automation and robotics in the context of Industry 4.0: the shift to collaborative robots'. In: *IOP Conference Series: Materials Science and Engineering.* Vol. 537. 3. IOP Publishing. 2019, p. 032073.

[7] D. M. Henderson. 'Euler angles, quaternions, and transformation matrices for Space Shuttle analysis'. In: (1977).

[8] M. Ben-Ari. 'A tutorial on Euler Angles and quaternions'. In: *Weizmann Institute of Science, Israel* (2014).

[9] L. Biagiotti and C. Melchiorri. *Trajectory planning for automatic machines and robots.* Springer Science & Business Media, 2008.

[10] P. Besset and R. Béarée. 'FIR filter-based online jerk-constrained trajectory generation'. In: *Control Engineering Practice* 66 (2017), pp. 169–180.

[11] S. Macfarlane and E. A. Croft. 'Jerk-bounded manipulator trajectory planning: design for real-time applications'. In: *IEEE Transactions on robotics and automation* 19.1 (2003), pp. 42–52.

[12] A. Valente, S. Baraldo, and E. Carpanzano. 'Smooth trajectory generation for industrial robots performing high precision assembly processes'. In: *CIRP Annals* 66.1 (2017), pp. 17–20.

[13] M. Kallmann, R. Bargmann, and M. Mataric. 'Planning the sequencing of movement primitives'. In: *proceedings of the international conference on simulation of adaptive behavior (SAB)*. 2004, pp. 193–200.

[14] A. J. Pachikara, J. J. Kehoe, and R. Lind. 'Path-Parameterization Approach Using Trajectory Primitives for Three-Dimensional Motion Planning'. In: *Journal of Aerospace Engineering* 26.3 (2013), pp. 571–585.

[15] J. Xiong et al. 'Simulation and trajectory generation of dual-robot collaborative welding for intersecting pipes'. In: *The International Journal of Advanced Manufacturing Technology* (2020), pp. 1–11.

[16] 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)*. IEEE. 2017, pp. 1–8.

[17] Z. Chen et al. 'An Optimized Trajectory Planning for Welding Robot'. In: *Materials Science and Engineering* 324.1 (2018), p. 012009.

[18] L. Biagiotti and C. Melchiorri. 'Online trajectory planning and filtering for robotic applications via b-spline smoothing filters'. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 5668–5673.

[19] K. Ning et al. 'A novel trajectory generation method for robot control'. In: *Journal of Intelligent & Robotic Systems* 68.2 (2012), pp. 165–184.

[20] T. DeWolf. *Dynamic Movement Primitives part 1: the Basics*. 2013. URL: https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/.

[21] A. J. Ijspeert, J. Nakanishi, and S. Schaal. 'Learning Attractor Landscapes for Learning Motor Primitives'. In: *Advances in Neural Information Processing Systems 15 [NIPS, 2002]*. MIT Press, 2002, pp. 1523–1530.

[22] M. Karlsson et al. 'Two-degree-of-freedom control for trajectory tracking and perturbation recovery during execution of dynamical movement primitives'. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 1923–1930.

[23] T. Kulvicius et al. 'Interaction learning for dynamic movement primitives used in cooperative robotic tasks'. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1450–1459.

[24] T. Kulvicius et al. 'Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting'. In: *IEEE Transactions on Robotics* 28.1 (2011), pp. 145–157.

[25] S. Calinon and D. Lee. *Learning control.* 2017.

[26] A. J. Ijspeert, J. Nakanishi, and S. Schaal. 'Movement imitation with nonlinear dynamical systems in humanoid robots'. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292).* Vol. 2. IEEE. 2002, pp. 1398–1403.

[27] B. Nemec et al. 'Task adaptation through exploration and action sequencing'. In: *2009 9th IEEE-RAS International Conference on Humanoid Robots.* IEEE. 2009, pp. 610–616.

[28] S. Schaal and C. G. Atkeson. 'Constructive incremental learning from only local information'. In: *Neural computation* 10.8 (1998), pp. 2047–2084.

[29] P. Englert. 'Locally weighted learning'. In: *Seminar Class on Autonomous Learning Systems.* Citeseer. 2012.

[30] B. Nemec and A. Ude. 'Action sequencing using dynamic movement primitives'. In: *Robotica* 30.5 (2012), p. 837.

[31] P. Pastor et al. 'Learning and generalization of motor skills by learning from demonstration'. In: *2009 IEEE International Conference on Robotics and Automation.* IEEE. 2009, pp. 763–768.

[32] *Franka Emika official website.* URL: https://www.franka.de/technology.

[33] *Franka Control Interface documentation.* URL: https://frankaemika.github.io/docs/overview.html.

[34] E. Madsen et al. 'Dynamics Parametrization and Calibration of Flexible-Joint Collaborative Industrial Robot Manipulators'. In: *Mathematical Problems in Engineering* 2020 (2020).

[35] C. Gaz et al. 'Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization'. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4147–4154.

[36] E. M. Hoffman et al. 'Robot dynamics constraint for inverse kinematics'. In: *Advances in Robot Kinematics 2016*. Springer, 2018, pp. 275–283.

[37] M. Magni and M. Cantoni. *Human-robot collaboration in assembly task learning enhanced by uncertainties adaptation via Bayesian optimization*. 2019. URL: https://www.politesi.polimi.it/handle/10589/150736.

[38] C. Ott. *Cartesian impedance control of redundant and flexible-joint robots*. Springer, 2008.

[39] T. J. Ross et al. *Fuzzy logic with engineering applications*. Vol. 2. Wiley Online Library, 2004.

[40] E. Treccani. *Introduction to Fuzzy Logic*. URL: https://www.treccani.it/enciclopedia/logica-fuzzy/.

[41] C. Gafà. *A very brief introduction to Fuzzy Logic and Fuzzy Systems*. 2020. URL: https://towardsdatascience.com/a-very-brief-introduction-to-fuzzy-logic-and-fuzzy-systems-d68d14b3a3b8.

[42] D. Sioson. *Introduction to Fuzzy Logic*. 2019. URL: https://medium.com/cafe24-ph-blog/introduction-to-fuzzy-logic-3664c610d98c.

Additional consulted material

[43] L. Anderlucci. 'Smooth trajectory planning for anthropomorphic industrial robots employed in continuous processes'. In: *Politecnico di Torino* (2019).

[44] A. Ankarali. 'Fuzzy Logic velocity control of a biped robot locomotion and simulation'. In: *International Journal of Advanced Robotic Systems* 9.4 (2012), p. 124.

[45] V. M. Aparanji, U. V. Wali, and R. Aparna. 'Robotic motion control using machine learning techniques'. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. 2017, pp. 1241–1245.

[46] C. M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[47] R. Castain and R. Paul. 'An On-Line Dynamic Trajectory Generator'. In: *International Journal of Robotic Research - IJRR* 3 (Mar. 1984), pp. 68–72.

[48] F. Chollet. *Deep Learning with Python.* Manning, 2018.

[49] Y. Jiang et al. 'Integrating task-motion planning with reinforcement learning for robust decision making in mobile robots'. In: *arXiv preprint arXiv:1811.08955* (2018).

[50] A. Mosavi, S. Ardabili, and A. R. Varkonyi-Koczy. 'List of deep learning models'. In: *International Conference on Global Research and Education.* Springer. 2019, pp. 202–214.

[51] J. Ogbemhe and K. Mpofu. 'Towards achieving a fully intelligent robotic arc welding: a review'. In: *Industrial Robot: An International Journal* (2015).

[52] L. Ren, W. Wang, and Z. Du. 'A new fuzzy intelligent obstacle avoidance control strategy for wheeled mobile robot'. In: Aug. 2012, pp. 1732–1737. ISBN: 978-1-4673-1275-2.

[53] C. Z. Resende, R. Carelli, and M. Sarcinelli-Filho. 'A nonlinear trajectory tracking controller for mobile robots with velocity limitation via fuzzy gains'. In: *Control Engineering Practice* 21.10 (2013), pp. 1302–1309.

[54] C. R. Severance. *Python for everybody: Exploring Data Using Python 3.* Charles Severance, 2009.

[55] W. Zhang et al. '3 Points Calibration Method of Part Coordinates for Arc Welding Robot'. In: Oct. 2008, pp. 216–224.