António Pedro Ascenso Gil

Licenciado em Ciências de Engenharia Mecânica

Methodological contribution to generic trajectory generation for additive manufacturing with a robotic manipulator

Dissertação para obtenção do Grau de Mestre em Engenharia Mecânica

Orientador: Doutora Carla Maria Moreira Machado, Professora Auxiliar, FCT NOVA
Co-orientador: Doutor André Rui Dantas Carvalho, Professor Auxiliar, ISEL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro 2020

Methodological contribution to generic trajectory generation for additive manufacturing with a robotic manipulator

Copyright © 2020 António Pedro Ascenso Gil

Faculdade de Ciências e Tecnologia e Universidade Nova de Lisboa

A Faculdade de Ciências e Tecnologia e a Universidade de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Para a minha família e amigos

Acknowledgments

I would like to begin by appreciating Professor Carla Machado for accepting me as her supervised student and guiding me though the chain of events ever since. Even though the initial project was cancelled due to the world pandemic situation that lead me to cancel my internship in Austria, Professor Carla was always professional and kind, guiding me through and introducing me to a new project my works inserts in.

At the same time, I would deeply like to appreciate Professor André Carvalho for welcoming me to his project and answering all my questions and requests, for countless video conferences over the past months. Professor André always kept suggesting solutions to the problems I would present him and giving me new insights on original approaches, and without his help this project would not have been possible. For all those reasons I am truly thankful.

To the Zeman Bauelemente team in Austria, from which I outline Dr. Andreas Hofer, Eng. Daniel Egger and Eng. Gregor Uher, for making me feel welcomed to Austria and facilitating the process for the almost two months spent there. I am genuinely sad our partnership had to cease to exist, however the balance made from the time spent in Zeman could not be better, as I take lessons of what I experienced for the rest of my life.

To my parents Benjamim and Rosário and my siblings João and Manuel, that endure to give me the best possible conditions to develop my work in peace. Their help along the returning to Portugal as well as multiple advices regarding the new project I was placed on were essential to make me insist on getting the best results I was capable of.

To my closest friends, who helped me deal with the pressure of developing this project and stood by whenever a break was needed in order to clear the ideas and prepare to get back to work.

Resumo

A tecnologia de fabrico aditivo apresentou melhorias substanciais na última década. No entanto, os princípios da tecnologia FFF (Fused Filament Fabrication) são essencialmente iguais. Com a fixação de uma unidade de extrusão de material a um manipulador robótico, os limites, uma vez definidos como *slicing* planar, deixam de ser impostos. Este tema é um dos principais focos da investigação "Optimal Non-Planar Trajectory Generation for Additive Manufacturing". Esta investigação em curso visa desenvolver um processador de objetos 3D (também conhecido como *slicer*) acoplado a um manipulador robótico. O programa proposto irá realizar o fatiamento em superfícies 3D, reduzindo a necessidade de estruturas de suporte e permitindo uma deposição superficial 3D que melhor se adapta às propriedades mecânicas de materiais em causa, conferindo uma maior fiabilidade estrutural às peças projetadas.

Como parte do desenvolvimento do projeto, é realizado um estudo onde, ao usar um *slicer* planar convencional, as geometrias são reduzidas à nuvem de pontos dados pelos comandos do *Código G* e as trajetórias são processadas para simular o movimento real do dispositivo FFF, usando polinómios parametrizados de 5º grau em ambiente 2D. Cenários que exigem maior atenção e posteriormente definidos como "arestas vivas" (onde o processo FFF causaria complicações aquando a impressão) são introduzidos, e uma metodologia é proposta esclarecendo o método para a interpolação polinomial nestes casos. Os instantes de tempo são normalizados, as velocidades são descritas e comparadas com as velocidades de extrusão e os seus limites são calculados.

Dois casos de estudo são apresentados e estudados com base em duas geometrias distintas: um cilindro e um cubo. Os objetivos conducentes ao trabalho da presente dissertação foram globalmente alcançados com sucesso. A aplicabilidade da metodologia proposta demonstrou-se bem-sucedida nos casos em que há extrusão contínua que não é interrompida por comandos G0 nos arquivos *G-Code* originais.

Palavras-chave: Fabrico Aditivo, FFF, Geração de Trajetórias, Manipulador Robótico, Polinómios de Quinta Ordem, Otimização

Abstract

Additive manufacturing as shown substantial improvements on the last decade, however the principles of FFF (Fused Filament Fabrication) technology are essentially unchanged. With the attachment of a material extruding unit to a robotic manipulator, boundaries once set as planar slicing are no longer imposed. A solution for this matter is one of the main focuses of the “Optimal Non-Planar Trajectory Generation for Additive Manufacturing” ongoing investigation. This investigation aims to develop 3D object processor (also known as a slicer) coupled with a robotic manipulator. The new slicing program will perform slicing in 3D surfaces, reducing the need for support structures and allowing a 3D surface deposition that best fits mechanical properties of specific materials, increasing structural reliability of the parts.

As a part of development to the project, a study is here carried out where, when using a conventional planar slicer, geometries are stripped down to the cloud of points given by G-Code commands, and trajectories are processed to simulate the real movement of the FFF device, using parameterized 5th degree polynomial equations working within a 2D environment. Scenarios demanding closer attention and later described as “sharp edges” (where the FFF process would cause complications) are introduced and a method is proposed on how to perform the polynomial interpolation on these cases. Time instants are normalized, velocities are described and compared with extrusion speeds. Limits for these are calculated.

Two case studies are introduced and studied basing on two distinct geometries: a cylinder and a cube. The objectives undertaken that lead to the creation of the present dissertation were overall successfully achieved with the proposed methodology. The applicability of the method has shown to be successful on the cases where there is continuous extrusion that is uninterrupted by G0 commands withing the original G-Code files.

Keywords: Additive Manufacturing, FFF, Trajectory Generation, Robotic Manipulator, Fifth Order Polynomials, Optimization

Table of Contents

Acknowledgments i

Resumo iii

Abstract v

Table of Contents vii

List of Figures ix

List of Tables xi

Symbols and Abbreviations xiii

1 Background, Motivation and Objectives 1

1.1 Introduction 1
1.2 Motivation and Objectives 2
1.3 Structure of the Dissertation 2

2 Literature Review 3

2.1 Fused Filament Fabrication as an Additive Manufacturing Process 3
2.1.1 Introduction: The technology 3
2.1.2 Components of an FFF machine 4
2.1.3 Path Control Fundamentals 5
2.1.4 Principles of G-Code 6
2.2 Robot Manipulators 7
2.2.1 History and basic functioning 7
2.2.2 Kinematics of serial-link manipulators 8
2.2.3 Approaches to describe rotation 9
2.2.4 Geometric and Analytical Jacobians 11
2.2.5 Kinematic Singularities 13
2.3 Trajectory Generation 15
2.3.1 Tool Path Generation in Non-Planar Layers with FFF 15
2.3.2 Splines: definition and properties 16
2.3.3 Well known types of splines 18
2.3.4 Quintic polynomial splines 18
2.4 Concluding Remarks 19

3 Methodology Development 21

3.1 Degrees of freedom definition 21
3.2 G Code points coordinates extraction 22
3.3 Linear interpolation of G code 24

3.4 Extrusion effect: G0 and G1 commands 29
3.5 Outside Interpolation 30
3.5.1 XY derivatives for outside interpolation 31
3.5.2 Polynomial interpolation 33
3.6 Inside Interpolation 35
3.6.1 Data points analysis 35
3.6.2 XY Derivatives for inside interpolation 40
3.6.3 Polynomial Interpolation 41
3.7 Time instants and time normalization 43
3.8 Extrusion speed and linear velocity 44
3.9 Maximum linear velocity module optimiziation 46

4 Analysis and Discussion 47

4.1 Introduction 47
4.2 Case Study Test Results: Cylinder 49
4.3 Case Study Test Results: Cube 54
4.4 Concluding Remarks 58

5 Conclusions and Future Work 59

5.1 Conclusions and contributions 59
5.2 Suggestions for Future Work 59

References 61

Appendix A 63

Appendix B 65

Appendix C 67

Appendix D 69

Appendix E 71

Appendix F 73

Appendix G 77

Appendix H 83

Appendix I 89

Appendix J 91

Appendix K 95

Appendix L 101

List of Figures

Figure 2.1 Maximization of different properties on material deposition 5
Figure 2.2 Typical industrial robot functioning 7
Figure 2.3 Book rotation by ZYX Euler angles applied on its corner referential 10
Figure 2.4 Manipulator with a spherical wrist at a wrist singularity 14
Figure 2.5 Function f(x)=1/(1+x^2) and the sixteenth degree polynomial approximation at the nodes x_k=-5+(10/16)k ,k=0,1,…,16 17
Figure 2.6 B-Spline consisting of seven segments 18
Figure 3.1 3D moduled cube with 10 mm edges 22
Figure 3.2 Fragment of G Code instructions for the 10 mm edge cube 23
Figure 3.3 Sliced 10 mm edge cube 25
Figure 3.4 Cube section made visible in a single layer 25
Figure 3.5 Sliced cylinder with 10 mm diameter and 10 mm height 26
Figure 3.6 Cylinder section made visible in a single layer 27
Figure 3.7 Section of a traditionally sliced cylinder with low resolution G-Code using inside and outside interpolation 28
Figure 3.8 Angle calculation between two lines on the linear interpolation of G-Code. 28
Figure 3.9 Extrusion on corners, where dashed lines represent *G0* commands and continuous lines define *G1* commands 29
Figure 3.10 Polynomial following the direction of the xy derivative on outside interpolation 31
Figure 3.11 Method for calculating the direction of the xy derivative on outside interpolation 32
Figure 3.12 Conventional tool path compared with the theorical path when encountering a sharp corner 35
Figure 3.13 Points to be generated in the event of one single sharp edge 37
Figure 3.14 Points to be generated in the event of two followed sharp edges 39
Figure 3.15 Representation of a generic polynomial following the xy derivative direction on points $ni1$ and $ni2$ 40
Figure 3.16 Generic time normalization for two different polynomials 43
Figure 4.1 Methodological programe sequence 48
Figure 4.2 Linear interpolation of the results from the algorithm in "layer_extraction.m" on the cylinder case study 49
Figure 4.3 Linear polynomial velocity module variation before optimization 51
Figure 4.4 Linear polynomial velocity module variation after optimization 52
Figure 4.5 Polynomial interpolation representation for the cylinder layer 53
Figure 4.6 Polynomial interpolation sharp edge detail for the cylinder layer 53
Figure 4.7 Linear interpolation of the results from the algorithm in "layer_extraction.m" on the cylinder case study 54
Figure 4.8 Cube linear polynomial velocity module variation before optimization 56
Figure 4.9 Polynomial interpolation representation for the cube layer 57
Figure 4.10 Cube linear polynomial velocity module variation after optimization 57

List of Tables

Table 2.1 FFF machine components 4
Table 2.2 Principle G-Code commands and correspondent meaning 6
Table 2.3 Linear and revolute joints entries 11
Table 2.4 Wrist and Arm Singularities 14
Table 3.1 Constant vectors and their components 30
Table 3.2 Direction of the xy derivative for the generic point on outside interpolation 32
Table 3.3 Definition of border conditions for outside interpolation 34
Table 3.4 Direction of the xy derivative for the generic point on inside interpolation 41
Table 3.5 Definition of border conditions for inside interpolation 42
Table 4.1 Defined parameters for the cylinder case study 50
Table 4.2 Defined parameters for the cube case study 55

Symbols and Abbreviations

3D	Three dimensional
2D	Two dimensional
FFF	Fused Filament Fabrication
FDM	Fused Deposition Modelling
AM	Additive Manufacturing
TCP	Tool Centre Point
DoF	Degree of Freedom
STL	Stereolithography
CAD	Computer Aided Design
XYZ	Axes Referential Convention
TLP	True Length Percentage
LSL	Linear Segment Length

es	Extrusion Speed [mm/s]
$angle_{in/out}$	Angle differentiating between inside and outside interpolation [°]
v	Linear polynomial velocity module [mm/s]
t	Time instant [s]
v_L	Linear polynomial velocity module limit [mm/s]

1 Background, Motivation and Objectives

1.1 Introduction

The following study was developed to integrate the progress of the ongoing investigation “Optimal Non-Planar Trajectory Generation for Additive Manufacturing”, which aims to link the developing field of Additive Manufacturing (AM) with the industrial robot manipulators technology.

Although it is now worldly recognized for several applications in tooling production, aerospace industry, radio frequency modules amongst others, Additive Manufacturing emerged a few decades ago mainly as a prototypal technology. Although it is still used for this goal, AM is now recognized as a way of fabricating fully developed products that are present on today’s market. The deposition principle of the technique has not suffered changes throughout the years, as material is still deposited layer by layer in one direction, which causes orthotropy in finished parts and creates requirements like support structures (meaning waste after production).

The investigation aims to develop 3D object processor (also known as a slicer) coupled with a robotic manipulator. This new slicing program (G3DSS) will perform slicing in 3D surfaces, diminishing the need for support structures and allowing a 3D surface deposition that best fits mechanical properties of specific materials, increasing structural reliability of the parts.

This study focuses on best fitting trajectories in 2D while using a conventional slicer, as the geometry is analysed through the result slices. The trajectory is a parametric curve that can take different forms. For the desired goal, splines using 5th order polynomials are investigated and tested. The specific requirement for these equations is for them to show continuity on both the function itself as well for first and second derivates, while displaying a smooth behaviour along their path. Scenarios demanding closer attention and later described as “sharp edges” (where the FFF process would cause difficulties) are introduced and a method is proposed on how to perform the polynomial interpolation on these cases.

As part of the trajectory definition proposed methodology, velocities along the movement are described and optimized in terms of parameters optimization. The result is the set of 5th degree polynomials that describe the movement of the extruder and nozzle when depositing material, along with instantaneously velocity and acceleration description.

Two case studies are introduced exemplifying the results on two different geometries: a cylinder and a cube. The differences are highlighted, and the validity of the method is verified.

1.2 Motivation and Objectives

The motivation that lead to the development of the following work is the result of the integration on an ambitious research that shows potential to revolutionize additive manufacturing as it changes perspectives as seen today.

From the literature review, concepts regarding additive manufacturing, robotic systems and trajectory generation are introduced and used on following chapters as a starting point for additional conclusions.

Consequently, the main objectives of the following investigation include:

- Extraction of coordinates from *G-Code* files.
- Introduction and solution to the sharp edges situation after linear interpolation.
- Differentiation between outside and inside interpolation scenarios.
- Time instants normalization.
- Velocity calculation and optimization.

1.3 Structure of the Dissertation

The following document is divided into 5 chapters and different appendixes:

Chapter 1: Background, Motivation and Objectives.

Chapter 2: Literature Review.

Chapter 3: Methodology Development.

Chapter 4: Analysis and Discussion

Chapter 5: Conclusion and Future Work.

2 Literature Review

2.1 Fused Filament Fabrication as an Additive Manufacturing Process

2.1.1 Introduction: The technology

Since the used manufacturing process, Fused Filament Fabrication (FFF) is one of the additive manufacturing processes, a contextualization is carried out in order to place FFF in the Additive Manufacturing (AM) field, presenting some concepts related to this manufacturing process and the various technologies integrating it.

Additive Manufacturing is the group of methods and technologies for production of three-dimensional (3D) objects directly from a virtual 3D model by means of addition of material [1]. It has gained ground in recent years, allowing substantial advances in the field of rapid prototyping to quickly create something similar to the final product, as a way of testing ideas and obtaining product feedback. This development has made the process more reliable and capable of creating final products.

The process of obtaining a part by additive manufacturing can be divided into several steps. On a typical FFF machine, the products to be developed are firstly created using CAD (Computer-Aided Design) programs and exported as STL files. Secondly these are imported into an interface of the printer itself, where the slicing of the part is made. The printing parameters (e.g. layer thickness, material constants, etc.) are obtained according to the desired material and geometry, as they can be adjusted. Following the choice of parameters, additive manufacturing is generally carried out autonomously by the equipment. When it is fully printed, the operator pursues to remove the part from the machine and proceeds to cleaning up excess material that is not intended in the final part, however it is necessary to use when printing specific parts (e.g. support structures).

Nowadays the applications of additive manufacturing are focused mainly on prototyping for several sectors such as aerospace, automotive, biomedical applications. However, with recent developments, it is already possible to manufacture parts with a sufficient level of reliability to be used as a final product.

Due to the high level of automation of this manufacturing method, development time and development costs are drastically reduced when compared to other common processes. The rapid development of prototypes allows to make small adjustments and to understand possible problems that are only possible by testing and observing a physical example of the part [1,2].

FFF, sometimes referred to as FDM (Fused Deposition Modelling) is a technique of additive manufacturing, in which a wire of fused material is deposited according to a certain standard. On conventional machines, this pattern is achieved through the relative movement between the extrusion head and the base plane on which the different layers are overlaid (the build plate) until

the final part is obtained. This technology is the most common among all those that make up additive manufacturing [1].

2.1.2 Components of an FFF machine

An FFF machine may have different configurations. On the investigation this work is taking part in, the XYZ movement as well as orientation of deposition will be related to the robot manipulator. However, main basic elements that enable FFF manufacturing are still present, these being represented in Table 2.1.

Table 2.1 FFF machine components

EXTRUDER	The filament that will eventually become the final product is usually wrapped in a feed roller. This is pulled by the extruder mechanism, transporting the filament from the roller to the extrusion head, which deepens the material and allows it to be deposited in the desired shape. The most common extruder mechanisms are of two types: Direct Extruder or Bowden Extruder [1].
FEEDER	Although it can be installed in different locations, the feeder operation is essentially the same for both Direct and Bowden Extruder. The feeder relies on the friction between the motor gear wheel and the filament, in order to transform the rotation of the motor into the linear movement of the wire. For this purpose, two jaws connected to another driving wheel are used to compress the base material and ensure its contact with the motor gear of the mechanism. The extrusion speed is controlled by the motor in order to guarantee an effective deposition on the build plate [1].
HOT END	Consists of a heated metal tube, usually by resistance, and a temperature sensor that allows controlling the temperature inside the hot end to be able to adapt it and maintain it according to the need of the material to be processed. As the polymer is melted inside the chamber, it is extruded through the nozzle, due to the pressure exerted by the forced filament through the feeder (still in the solid state, coming from the top of the hot end).
BUILD PLATE	It is the initial surface on which the intended part is iteratively built layer by layer. It can be fixed or mobile depending on the equipment. Being used as a base for all other layers, it is essential to ensure its stability. For this reason, the adhesion of the first layer to the build plate and the levelling of the table must be ensured, so the axis perpendicular to the build plate is parallel with the Z axis of the equipment. When the build plate is heated, the material adheres and promotes the risk of deformations in the part and its detachment. The chosen pre-heating temperature is specific to the material that is to be printed.

2.1.3 Path Control Fundamentals

After the slicing is made, the traditional style control software attributes an outline to each slice and determines how to fill within the outline. This outline is determined by extracting intersections between the STL file triangles (files with the 3D CAD parts) and a plane that represents the current cross section of the build. It is recommended to have randomly generated start/stop regions evenly distributed to avoid having a clear seam where the outline overlaps itself.

Clear access to deposit material within the outline is critical, as this is facilitated by the extra degrees of freedom and the rotation of the extrusion head (linked to the robot manipulator 6th axis), when compared to the traditional FFF machines. Another important aspect is how close the material being deposited is to the adjacent material, as a larger distance may cause the material not bonding correctly.

Regarding the fill pattern, there must be an offset inside the outline, as the extrusion nozzle will be placed inside the outline with minimal overlap. Afterwards, the filling trajectory is determined according to the predefined fill pattern. In most cases, the chosen pattern is the so called “weave pattern”, as it gives the part greater strength when the weave is rightly crossed, this being when the angle between weaves follows the directionality in the fibres. As more weave patterns are used in a specific layer, greater the weakness will be on the part. For this reason, the number of different fill patterns must be minimized throughout a single layer. However it is not possible to ensure one fill pattern will properly fill one layer. On Figure 2.1, different properties are maximized upon defining the tool path, these being precision and material strength. One solution is to control the flow rate, extruding more or less material in specific zones, avoiding voids in one case and swell in the others. By other words, the material flow from the extrusion head should not be directly proportional to the instantaneous velocity of the nozzle (same as the robot manipulator Tool Center Point) when this is low, as it should be controlled depending on the toolpath. On this study, the tool is to be considered the FFF mechanism itself, and the tip of the nozzle to represent the Tool Centre Point (TCP). If the velocity of the TCP is zero but there is a directional change in a weave path, the small amount of flow should be secured.

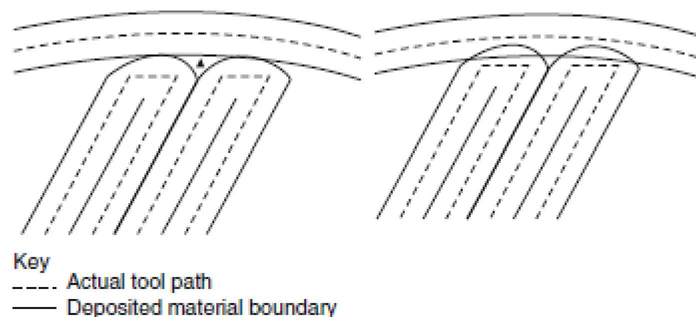

Figure 2.1 Maximization of different properties on material deposition: precision (left) and material strength (right) [1]

2.1.4 Principles of G-Code

The trajectory building process, as it is going to be further explained along this document, is defined trough information gathered from the code generated after slicing a certain geometry. That code is commonly treated as ***G-Code***. The information collected from the code that is to serve as data for the trajectory itself is gathered from the code by the form of point coordinates in x, y, z. Note that when relying on a conventional slicer, the z coordinate is a function of the layer number

The process of preparing a piece for an FFF process evolves several steps. From the design of the geometry, to exporting it as an *STL* file and later import to the conventional slicer software. At this point, the software exports a *G-Code* file containing all the information formatted so that the FFF device can read them. G-Code can be written by the user itself, however this implies simpler lines of code as for example when calibrating the FFF device.

G-Code stands for "geometric code" and it is used for all types of CNC machining other than additive manufacturing. It instructs the machines where and how to move, and it depends on the specific machine being utilized, as the code varies with different characteristics devices.

Typical *G-Code* follows the format visible in Table 2.2. Available commands are adapted from the model. All command letters as "G#" are followed by one hash representing number combinations that correspond to different commands themselves. For example, when *G-Code* is used within FFF, G0 stands for rapid move and G1 for linear move. Usually G0 commands are used to move rapidly from one point to another while not extruding material, as G1 commands generally extrude in a linear move.

M commands represent miscellaneous as mentioned. These can represent actions like "Program stop" for M0, "Spindle on, clockwise" for M3 or other commands for tool changing, flood coolant, end of program, among others.

Table 2.2 Principle G-Code commands and correspondent meaning. Adapted from [3]

Code	Function	Code	Function
N#	Line number	**T#**	Tool selection
G#	Move to a point or other sort of movement	**F#**	Feed rate in millimetres per minute
X#	X coordinate, usually to move to	**S#**	Command parameter, such as time in seconds, temperature, and others
Y#	Y coordinate, usually to move to		
Z#	Z coordinate, usually to move to		
M#	Miscellaneous functions		

2.2 Robot Manipulators

2.2.1 History and basic functioning

Corke [4] defines a robot as "a goal-oriented machine that can sense, plan and act". Having a goal in mind, a robot senses its environment and, according to the information that it gathers, plans some sort of action, and immediately performs it. On the case of this study, the action is to move following a given trajectory, as it deposits material in order to create a new part. A manufacturing robot can be one of many types, however it is generally represented by an arm-type manipulator with several degrees of freedom, fixed on a base.

The following chart in Figure 2.2 presents a brief resume of the general robot process upon faced with a certain task. The controller is the brain of the robot and it runs the code written instruction (the program). The actuator, or drive, is the engine that transmits movement to the links. It can be either hydraulic, electric, or pneumatic. The end effector functions as a hand, as it comes in contact with the material that is to be handled. It can assume different forms, as a gripper or welding torch. The sensors give feedback to the computer about the environment, as they collect information through pulses and send it back to the computer, preventing the robots from bumping into each other. Visions sensors allow the pick and place robots to differentiate between items [5].

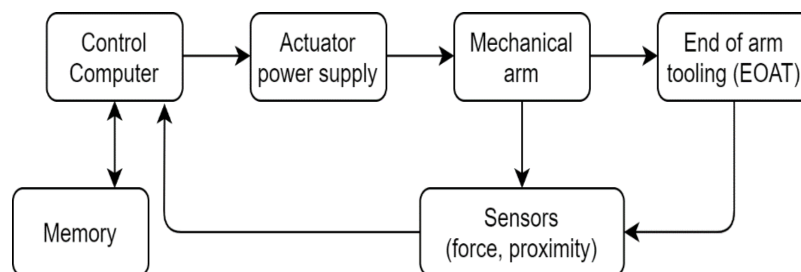

Figure 2.2 Typical industrial robot functioning (adapted from [5])

For the *ABB* robots for example, the controller is the *IRC5* industrial controller. As all the *ABB* controllers, it uses RAPID programming language to create specific solutions.

The end effector (also called as end of the arm tooling) is normally purchased separately to the robot itself. The end effector of an articulated robotic arm can be of either a gripper or a tool. The grippers can be categorized as multiple, single, internal, or external, depending on how they perform. The tools can be compliant, with contact or without contact.

Robots work within their working envelope, as every point the robot can access is defined by a cloud of points. Adding external axes to manipulators is a way of increasing its working envelope (e.g. mounting the robot on a rail).

2.2.2 Kinematics of serial-link manipulators

Kinematics are responsible for studying the motion of objects without considering the forces and moments that cause a certain motion. When associated with robots, it refers to the analytical study of the motion of a robot manipulator [6]. Each joint of the robot corresponds to one degree of freedom, and it can be either one of two kinds: translational (in the case of prismatic joints) or rotation (in the case of revolute joints). Each joint is connected to another by links, as the last joint is typically connected with the end effector [4]. For the purpose of this investigation, the study will be carried on regarding revolute joints.

It is useful to define certain concepts that relate to the position of the robotic arm when discussing kinematics.

- Resolution is the limit of the number of points the robot will reach;
- Spatial resolution is the smallest increment of movement the robot performs;
- Repeatability is a measure of error on the natural variance during a repetitive task, defined as the addition or subtraction of 3 times the standard deviation value;
- Accuracy defines how close a robot gets to a desired position and it is important when performing off-line programming.

To study the trajectory of the robotic arm, there must be a separation between forward and inverse transformation. **Forward kinematics** (result of a forward transformation) is the mapping from joint coordinates, or robot configuration, to end effector [4]. **Inverse kinematics** works somehow as the opposite of the previously studied case: if the Cartesian pose of an object is known, inverse kinematics determine the joint coordinates the robot needed to reach it. However, inverse kinematics of a redundant kinematic chain have infinite solutions (as the inverse is not a true function). The learning algorithm must acquire a particular inverse and make sure it is a valid solution for the faced scenario.

Upon planning a path, the robotic arm can follow one of three kinds of motion: slew motion, joint interpolated motion, or straight-line motion. In the **slew motion**, the robot moves from a generic point A to a generic point B as each axis of the manipulator travels as quickly as possible. Each axis starts its movement at the same time, but they can stop at different times, depending on the distance each one moves to get to the final position (considering acceleration and deceleration). **Joint interpolated motion** is like slew motion, but all the joints start and stop at the same time. It demands only speeds needed to accomplish any movement in the least amount of time. Finally, in **straight-line motion** the tool travels in a straight line from start to stop points. This method leads to non-regular motions when the boundaries of the workspace are approached. It is considered the least desired motion of the three [5].

2.2.3 Approaches to describe rotation

While describing the trajectory of the nozzle (represented by the Tool Centre Point), it is crucial to define the orientation of the end effector as it will vary in time. There are two methods to describe it, Euler angles and quaternions, and both are following described.

Euler Angles were introduced by the Swiss mathematician Leonard Euler, who said that any orientation of a rigid body can be parametrized by three independent coordinates [7]. All rotations can be represented as a combination of three rotations around the object's local axes, and always to the object axes themselves. The order on which these rotations are made is crucial, as all rotations about one particular axis are called Eulerian (e.g. XYX rotation) and all rotations about all three axes are called Cardanian (e.g. XYZ rotation). However, it is usual to refer to all different 3 angle representations as Euler angles [4].

To understand the functioning of this system, the concept of **Rotation Matrix** is introduced, starting with the following example: imagine it is needed to change from a hypothetical referential frame A to a hypothetical frame B. To describe this transformation, a rotation matrix or more completely a transformation matrix comes in hand. In the case the new B frame origin is located at the same point as frame A and there are different orientation on the axes, there will only exist a rotation between the two frames. If the origin of both is not coincident, then we also witness a translation associated to the vector between the origins of frame A and B. Here, the focus is purely on the rotation.

The **Rotation Matrix** is the result of a set of three rotations, one around each one of X,Y and Z axes. Rotation matrixes are used to represent an orientation, change the reference frame in which a vector or a frame is represented or to rotate a vector or a frame [7].

It is possible to represent the orientation of a coordinate frame by its unit vectors expressed on the reference coordinate frame, following (Eq. 2.1 (where ${}^{A}R_{B}$ denotes the rotation matrix). Note that this equation is orthogonal, and its determinant is always equal to +1. Besides this, its inverse is equal to its transpose [4].

$$\begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix} = {}^{A}R_{B} \begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \end{pmatrix} \qquad \text{(Eq. 2.1)}$$

$$\boldsymbol{R} \in \mathbf{SO}(3) \subset \mathrm{R}^{3\times 3}$$

Now that the concept of rotation matrix is introduced, it is simpler to define Euler Angles. ZYX and ZYZ sets of Euler angles will be presented next.

1. **ZYX Euler Angles**

The following example denotes the ZYX Euler angles defining the rotation of a hypothetical book displayed in Figure 2.3 [7]. The rotation is applied on the corner's referential as seen bellow.

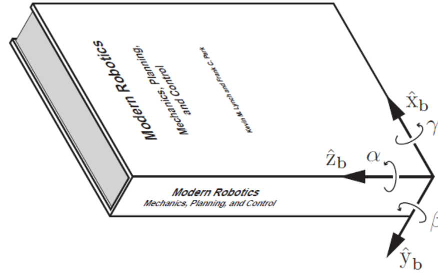

Figure 2.3 Book rotation by ZYX Euler angles applied on its corner referential [7]

The corresponding ZYX sequence is the multiplication of the three rotation matrixes (Eq. 2.2).

$$\boldsymbol{R} = \boldsymbol{R}_z(\alpha)\boldsymbol{R}_y(\beta)\boldsymbol{R}_x(\gamma) \quad \text{(Eq. 2.2)}$$

It is also possible to do the opposite, this meaning, given an arbitrary rotation matrix **R**, determining the (α,β,γ) satisfying Eq. 2.2. This shows ZYX Euler angles represent all orientations [7].

2. XYZ Euler Angles

The XYZ angles are also called the RPY (Roll-Pitch-Yaw angles). The process for obtaining a rotation described by XYZ angles follows a sequence. Firstly, there is a rotation around the original x axis (x), then a rotation about the new y axis (y') and finally a rotation around the new z (z'') axis (Eq. 2.3). Each rotation is represented by a rotation matrix. Adapted from [8].

$$\boldsymbol{R} = \boldsymbol{R}_{\boldsymbol{x}}(\phi)\boldsymbol{R}_{\boldsymbol{y}'}(\vartheta)\boldsymbol{R}_{\boldsymbol{z}''}(\psi) \quad \text{(Eq. 2.3)}$$

The best-known problem that exists when using Euler angles is known as the **Gimbal Lock**, formally called singularity. This occurs when the rotational axis of the middle term in the sequence becomes parallel to the rotation axis of the first or third term [4].

The other widely used way to define orientation is using unit quaternions. A **quaternion** consists of a normalized vector of four scalars. They work as an extension of complex numbers and can be represented in 4D spaces, as a typical quaternion is written corresponding to (Eq. 2.4).

$$q = w + xi + yj + zk \quad \text{(Eq. 2.4)}$$

Where w, x, y and z are real numbers and $i^2 = j^2 = z^2 = -1$. One quaternion can represent either a combination of several rotations or a single rotation from one orientation to another, these rotations meaning the multiplication of quaternions [9].

Quaternions overcome the drawback of angle/axis representation by the adding of the fourth parameter [8].

2.2.4 Geometric and Analytical Jacobians

The Jacobian is named after Carl Jacobi and it is the mathematical matrix equivalent of a derivative, this being the derivative of a vector-valued function of a vector with respect to a vector [4].

In Robotics, it is vital to relate joint velocities to end effector linear and angular velocities in either world frame or end effector frame. This relationship is represented by the **Geometric Jacobian** and it depends on the manipulator configuration.

The goal of the differential kinematics is to find the relationship between the joint velocities and the end effector linear and angular velocities. To understand the Geometric Jacobian, the notation was adapted from [8] for a 6 DoF robot manipulator arm (similar to the one used during the project) (Eq. 2.5). The Jacobian matrix is represented by $J(q)$ and because it is referring a 6R robot, it takes the form of a matrix with 6 columns and 6 rows. $[\dot{x}, \dot{y}, \dot{z}]^T$ represents the linear velocities of the end effector, this meaning how fast the end effector goes on each direction. $[\omega_x, \omega_y, \omega_z]^T$ represents the angular velocity of the end effector, this meaning how fast the end effector is rotating around each axis. $[\dot{q}_1\ \dot{q}_2\ \dot{q}_3\ \dot{q}_4\ \dot{q}_5\ \dot{q}_6]^T$ represents the joint velocities when it is unclear if they are referring prismatic or revolute joints. In the case of revolute joints the matrix is normally replaced with the notation $[\dot{\theta}_1\ \dot{\theta}_2\ \dot{\theta}_3\ \dot{\theta}_4\ \dot{\theta}_5\ \dot{\theta}_6]^T$ (Eq. 2.5).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ w_x \\ w_y \\ w_z \end{bmatrix} = J(q) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix} \quad , \quad J(q) \in \mathbb{R}^{6\times 6} \qquad \text{(Eq. 2.5)}$$

Since the Jacobian has always six rows due to the velocity factors, having a six joints robot makes the matrix to have six columns. For that reason, the Jacobian for a 6 DoF manipulator robot is a square matrix that can be invertible, if it is not located at a singularity point (subject studied further ahead).

In order to compute a Jacobian matrix so that either joint velocities or end effector velocities can be calculated, it is important to proceed separately for the linear and angular velocities. The following information only regards revolute joints since no prismatic joints exist on the robot manipulator. If a certain Joint i is revolute, Table 2.3 shows how to calculate both linear and joint regarding entries on the matrix. This table denotes a different notation although it is adapted from [8].

Table 2.3 Linear and revolute joints entries

Linear components of the end effector velocity	$\boldsymbol{R_{i-1}^{0}} \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} * (\boldsymbol{d_n^0} - \boldsymbol{d_{i-1}^0})$
Rotational components of the end effector velocity	$R_{i-1}^{0} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Where

- R_{i-1}^{0} represents the Rotational matrix between the base frame of the robot's manipulator and the frame of joint $i-1$;
- d_n^0 represents the displacement between the origin of the base frame and the origin of frame n (n is equal to the number of joints);
- d_{i-1}^{0} represents the displacement between the origin of the base frame and the origin of frame i.

The values are original from the transformation matrix represented by Eq. 2.6, Eq. 2.7 and Eq.2.8 (notation from [4])

$$T_n^0 = \begin{bmatrix} R_n^0 & d_2^n \\ 0_{1x3} & 1 \end{bmatrix} \quad \text{(Eq. 2.6)}$$

$$\begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \\ 1 \end{pmatrix} = \begin{bmatrix} {}^{A}R_B & t \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \\ 1 \end{pmatrix} \quad \text{(Eq. 2.7)}$$

$$where \ {}^{A}T_B = \begin{bmatrix} {}^{A}R_B & t \\ 0_{1\times 3} & 1 \end{bmatrix} \quad \text{(Eq. 2.8)}$$

The opposite problem also exists, this being represented by calculating joint velocities through end effector velocities, also referred to as resolved-rate motion control by [4]. This problem is visible in Eq. 2.9, where $\dot{q}$ represent the joint velocities matrix and ν represents the end effector velocities matrix.

$$\dot{q} = J^{-1}(q)\nu \quad , \quad \dot{q} \in \mathbb{R}^{6\times 1} \cap J^{-1}(q) \in \mathbb{R}^{6\times 6} \quad \text{(Eq. 2.9)}$$

To solve this problem, one must study the inverse matrix. When discussing the invertibility of the Jacobian, it is important to refer only square matrices are invertible. Methods exist to calculate the "pseudo-inverse" of rectangular matrices. However, these are not going to be studied in this document. Since the manipulator Jacobian has always 6 rows due to the fact they represent both linear and angular velocity on all three axes, it will only take the form of a square matrix when there are six either prismatic or revolute joints and no singularities associated with it. For that reason, only Jacobian matrices corresponding to 6 DoF robot manipulators are possible to calculate an inverse for. These Jacobians are also known as fully-actuated [4].

Another important concept to introduce is the **Analytical Jacobian**. Although previous information expressed spatial velocity in terms of translational and angular velocity vectors, angular velocity is not the most intuitive concept. For the application of this project, it was chosen to consider rotational velocity in terms of **rates of change of XYZ Euler Angles**, where the angles are represented by $\Gamma = (\theta, \varphi, \psi)$ (Eq. 2.10).

$$R = R_x(\theta)R_y(\varphi)R_z(\psi) \quad \text{(Eq. 2.10)}$$

After some manipulation [4], the corresponding derivate or R is represented (Eq. 2 11).

$$\dot{R} = [\omega]_{\times} R \qquad \text{(Eq. 2.11)}$$

The result is a relationship between angular velocity of the end effector and the rate of change of XYZ Euler Angles, giving origin to a "new 3x3 Jacobian matrix" denoted by A (Eq. 2.12), where ω represents the angular velocity. The new A matrix is also invertible, as it is a 3x3 non-singular matrix.

$$\omega = A(\Gamma)\Gamma \quad , \quad A(\Gamma) \in \mathbb{R}^{3\times3} \qquad \text{(Eq. 2.12)}$$

From here the analytical Jacobian $J_a(q)$ is defined from the original manipulator Jacobian as shown in Equation 2.13.

$$J_a(q) = \begin{bmatrix} I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & A^{-1}(\Gamma) \end{bmatrix} J(q) \qquad \text{(Eq. 2.13)}$$

2.2.5 Kinematic Singularities

As seen before, the Jacobian is, in general, a function of the configuration q. When those configurations reach an instant when J is rank deficient, they are called **kinematic singularities** [8]. Singularities may represent different situations, those being configurations at which the mobility of the manipulator is reduced (e.g. when the position and/or orientation is not reachable for the end-effector). When the structure is at a singularity, infinite solutions for the inverse kinematics may exist, and in the neighbourhood of a singularity small velocities in the operational space can cause large velocities in the joint space. Also, kinematic singularities are independent of the choice of either a fixed frame or an end effector frame [7].

Different definitions complement each other: [Modern robotics] defines singularities as "*postures at which the robot's end effector loses the ability to move instantaneously in one or more directions*", and [4] refers that "*singularities occur when the robot is at maximum reach or when one or more axes become aligned resulting in the loss of degrees of freedom*".

From here, [8] classifies singularities as one of either:

- *Boundary singularities* that happen when the manipulator is either outstretched or retracted. It can be avoided if a condition is set for the robot configurations to make the TCP remain inside the manipulator's reachable workspace.
- *Internal singularities* that happen generally when two or more axes of motion of the manipulator become aligned (inside the reachable workspace) or else the attainment of specific end-effector configurations. When occurring, they constitute a serious problem.

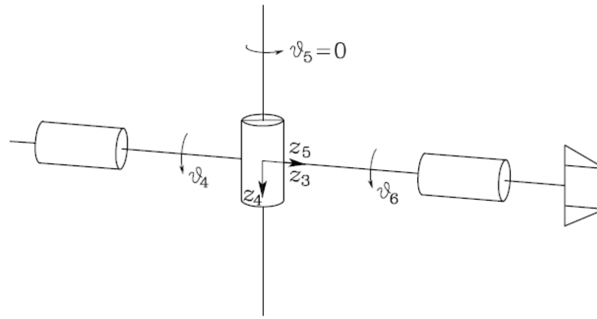

Figure 2.4 Manipulator with a spherical wrist at a wrist singularity [8]

When focusing on the Jacobian for identification, a singularity is expected when the determinant of the Jacobian is null, what is notated in Eq. 2.14, for a robot configuration q [4].

$$J(q)\,\det\big(J(q)\big) = 0 \qquad \text{(Eq. 2.14)}$$

On manipulators with spherical wrists, singularities are also divided between arm singularities (resulting from the motion of the first 3 or more links) and wrist singularities (resulting from the motion of the wrist joints), further explained in Table 2.4.

Table 2.4 Wrist and Arm Singularities

Wrist Singularities	On the structure of the robot visible in Figure 2.4, it is possible to imagine that motion caused by equal magnitude rotations about the positive directions on both $\boldsymbol{v_4}$ and $\boldsymbol{v_6}$ produce the same end effector rotation. This singularity is described in the joint space and can happen anywhere withing the manipulator's reachable workspace.
Arm Singularities	Unlike wrist singularities, arm singularities are well identified in the operational space and can be avoided while planning the trajectory for the end effector [8]. On Anthropomorphic arms, elbow singularities and shoulder singularities may be encountered.

It is possible to expect singularities upon faced with different scenarios. [7] fits them on five different events where it is possible to conclude the manipulator is at a singularity:

1. Two collinear revolute joint axes;
2. Three coplanar and parallel revolute joint axes;
3. Four revolute joint axes intersecting at a common point;
4. Four coplanar revolute joints;
5. Six revolute joints intersecting a common line.

2.3 Trajectory Generation

A trajectory is nothing more than the specification of the robot position as a function of time [7]. It may be strictly defined as for example when contouring an object edge or it may have a certain freedom to be obtained between two hypothetical points. As a function of time, it should sufficiently smooth simultaneously with respecting any given limits on joint velocities, accelerations or torques.

The planning of a trajectory consists of generating a time sequence of the values withdrawn from an interpolating function (that are studied ahead) of the desired trajectory [8]. The study of trajectory planning will be divided between *point-to-point* motion and *motion through a sequence of points*, while regarding **joint space trajectories**. Both aim to generate a time sequence of variables describing the end effector's position and orientation in time, while respecting imposed constraints.

Since the project requires C^2**continuity**, this meaning the curves are continuous simultaneously with its first and second derivates, solutions where this is guaranteed are to be further studied.

2.3.1 Tool Path Generation in Non-Planar Layers with FFF

Before digging deeper into trajectory planning, it is important to start by explaining and differentiating between path and trajectory, as these terms are often confused. In [8] a **path** is defined as *"the locus of points in the joint space, or in the operational space, which the manipulator has to follow in the execution of the assigned motion"*, as a **trajectory** by the other hand is defined as *"a path on which a timing law is specified, for instance in terms of velocities and/or accelerations at each point".*

In order to achieve the final goal of the project, the robot manipulator using the FFF device to create new parts must be given a trajectory after the slicing of the part is made, so it can successfully fill the designated layers. That way, the 6 DoF robot will be able to deposit material on complex surfaces. When integrated with Additive Manufacturing, the idea is to generate the sequences of positions and TCP orientations to form lines that cover the entire surface and its inside.

The orientation of the FFF extruder at each extruder Cartesian position is calculated in form of Euler Angles. The default orientation can be normal to the surface, however it must be known there is the possibility of either the nozzle or heating block hitting the part while printing on a concave surface. In order to prevent collisions and creating a smoother surface, the extruder tip is not always aligned with the surface normal and is instead calculated by computing the trajectory in the joint configuration space with the necessary constraints [10].

When waypoints are generated for the tool path, they do not take into consideration the constraints and parameters imposed by FFF with an articulated robot arm, as the relationship between tool configuration and robot configuration is hardly linear at all. For this reason, it is important to account for different factors, these being: robot reachability, collision avoidance and to meet TCP speed constraints [10]. **Robot reachability** is the most straightforward concept of the three, however it is of upmost importance, as points outside the robot workspace cannot be reached. The discussed

collision avoidance is somehow particular to the FFF application as in this case it is referred to collisions between extruder and print base. As inverse kinematics for 6 DoF robots has multiple solutions, reachability will not guarantee the robot will not collide with the part. In order to maintain uniform material deposition over the print base, the **extruder linear velocity** (that is governed by the robot joint angles velocity as seen before) should be maintained at all the time. It is important to guarantee the robot can provide and maintain the desired constant TCP velocities.

2.3.2 Splines: definition and properties

When trying to find a curve that best fits a set of data points, several options are presented as possible solutions. The first one that may come to mind is interpolating the points with a polynomial of a certain degree. In some cases, that solution becomes a sufficient approximation of reality, mainly when the set of data points are well distributed, this meaning points behave with a typical linear, quadratic or even cubic behaviour.

Trying to approximate sets of data points with higher degree polynomials may come to mind when trying to fit more delicate behaving curves. With the present case of defining a curve that is becoming a trajectory, very specific paths must be followed. These paths are hardly represented by higher order polynomials, as these tend to shoot very drastically for higher positive and negative values and are hard to control. Carl de Boor [11] even referred that "*If the function to be approximated is badly behaved anywhere in the interval of approximation, then the approximation is poor everywhere*", this meaning the approximation must be the best fit for every point that is to be approximated through the created polynomial. As splines are polynomials, the order of the polynomial defined splines must correspond to the requirements set while trying to minimize the order, as higher order polynomials tend to behave irregularly.

Visual confirmation of the information stated above may be found in Figure 2.5, comparing the function itself with a 16th polynomial approximation. The behaviour of the high order polynomial is accurate with the function itself for values that do not exceed values close to $+3$ and -3 within the x axis.

To achieve C^2continuity and smother paths for the FFF device to follow, splines are presented. These are known to respect the stated requirements, depending on the type discussed, as several types of splines are being further explained in this chapter.

A spline is defined by [12] simply as a *"piecewise polynomial parametric curve"*. Essentially, splines piece together several polynomials, and by defining certain boundary conditions to each one of them, certain properties like continuity may be assured. The polynomial in question can be of different degrees, however it is important to note the choosing of the polynomial degree will affect the continuity of the path itself and its derivatives. Continuity is guaranteed essentially by matching function and its derivatives at every point. Continuity for first derivatives is guaranteed by matching the slopes at every point and second derivative essentially operates the same way.

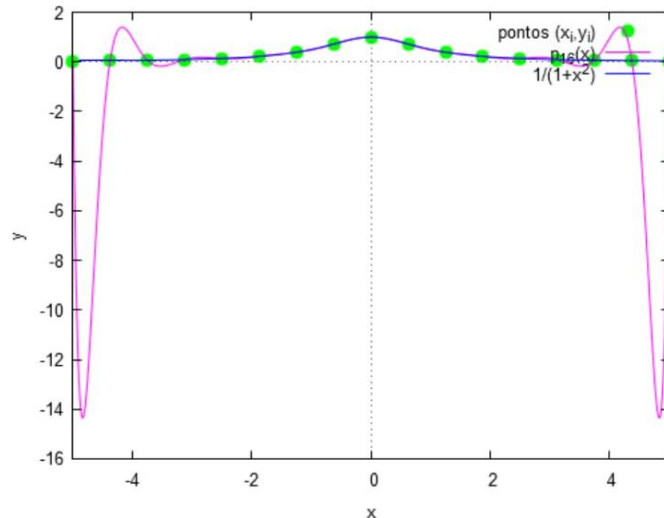

Figure 2.5 Function f(x)=1/(1+x^2) and the sixteenth degree polynomial approximation at the nodes x_k=-5+(10/16)k ,k=0,1,…,16 [13]

Because splines are composed by piecing together polynomials, they are defined along specific intervals of the parametric axis. If that parameter is time, the result is something like $S(t)$, where t represents the time that varies from 1 to n, knowing there are $n+1$ points where the spline is defined [13].

In (Eq. 2.15) S is interpolates the function f the nodes t_i , $i = 0,1,\ldots,n$ if:

$$S(t) = \begin{cases} S_0(t) \quad , & t_0 \leq t \leq t_1 \\ S_1(t) \quad , & t_1 \leq t \leq t_2 \\ \quad\quad\quad . \\ \quad\quad\quad . \\ \quad\quad\quad . \\ S_{n-1}(t) \quad , & t_{n-1} \leq t \leq t_n \end{cases} \tag{Eq. 2.15}$$

Where each $Si(t)$ is a polynomial of a certain degree that verifies the conditions below.

1. $S(t_k) = f(t_k)$, $k = 0,1,\ldots,n$;
2. $S_i(t)$, $i = 0,1,\ldots,n-1$ is a polynomial of a certain degree (3 for cubic splines, 5 for 5th order splines, etc.);
3. $S(t), S'(t)$ and $S''(t)$ and continuous on the interval $[t_0, t_n]$, when the polynomial is of sufficient degree to so allow it.

In order to verify the third condition, continuity must be imposed to both S and its derivatives. That may be done by matching the limits of $S(t), S'(t)$ and $S''(t)$ at each point t_i (Eq. 2.16).

$$\begin{cases} lim_{t\to t_i^-} S(t) = lim_{t\to t_i^+} S(t) \\ lim_{t\to t_i^-} S'(t) = lim_{t\to t_i^+} S'(t) \\ lim_{t\to t_i^-} S''(t) = lim_{t\to t_i^+} S''(t) \end{cases} \tag{Eq. 2.16}$$

2.3.3 Well known types of splines

Splines can be constructed through several methods that are divided in two different groups: through the **end points** or by creating **new control points**.

Defining a spline through the conditions on its end points may be called the "traditional" way, as one defines the polynomials by finding the border conditions for each polynomial and matching respectively with the ones right before and right after.

Splines can also be defined by newly created control points. The theory behind the definition of the spline itself remains, only the method change. One of the most famous methods while using control points are the **B-Splines**. These have the characteristic of not passing through any of their control points, and for that reason they would not be the first choice when it comes to defining a path from a set of data points. In Figure 2.6, a seven segment B-Spline may be observed, as it stands out the the control points not being part of the curve itself.

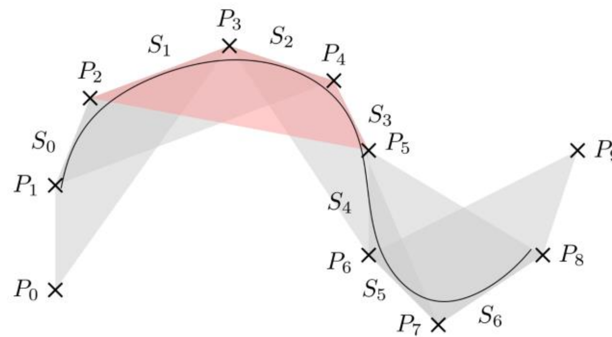

Figure 2.6 B-Spline consisting of seven segments [12]

2.3.4 Quintic polynomial splines

For the purpose of continuity on both functions, first and second derivatives, 5th degree polynomials are studied in particular for later application on splines. A typical 5th degree polynomial as the following aspect (Eq. 2.17).

$$p(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f \quad \text{(Eq. 2.17)}$$

In order to calculate the 6 unknows on the equation $\{a, b, c, d, e, f\}$, there must bethe same number of border conditions. In the case of the present challenge of trajectory calculation, the parameter to be used is time. Six conditions may be defined, these being position, velocity and acceleration at the beginning and end of each polynomial.

Formally, conditions are presented as follows (Eq. 2.18, Eq. 2.19, Eq.2.20).

$$p(t) = at^5 + bt^4 + ct^3 + dt^2 + et + f \quad \text{(Eq. 2.18)}$$

$$p'(t) = \frac{d\,p(t)}{dt} = 5at^4 + 4bt^3 + 3ct^2 + 2dt + e \quad \text{(Eq. 2.19)}$$

$$p''(t) = \frac{d^2\,p(t)}{dt^2} = 20at^3 + 12bt^2 + 6ct + 2d \qquad \text{(Eq. 2.20)}$$

By matching polynomial and derivatives at the two time instants delimiting the interval of time under study, it is possible to calculate the constant required to build the splines

2.4 Concluding Remarks

From the bibliographic research carried out for this work, some considerations can be considered for the experimental development:

- *G-Code* commands may be commands singly analysed and several information may be withdrawn from a *G-Code* file.
- Additive manufacturing requires special attention to several details when projecting trajectories that are to be used along this technology, and these must be considered.
- Robotic manipulators allow big flexibility of positions and orientations for the TCP, and for that reason are not to cause restrictions when defining polynomials defining position, velocities and orientations on the future trajectories.
- Singularities should not be an issue for the present methodology, as it gives its most attention to the TCP position for the robotic manipulator.
- Splines, and specially 5th degree polynomials offer great characteristics to approximate real life curves to mathematical expressions and are for that reason the optimal choice for the present objective.

3 Methodology Development

The following chapter presents the methodologies developed during the progress of this work. After a first description of the degrees of freedom that involve the generation of the nozzle trajectory, attention turns to point coordinates that are extracted to serve as data base for future developments. Following, a description is made concerning the problems detected after a first linear interpolation of the set of points resulted from the slicing of a geometry. After, the interpolation process is carried out while being divided into the outside an inside interpolation. Derivatives are calculated along with new points for the situations where what will be called as “sharp edges” exist. Time instants for each polynomial are normalized and velocity is described, while the maximum velocity is optimized within the identified constraints.

3.1 Degrees of freedom definition

A certain movement from one point to another is described by an array of conditions. With the variation of these several conditions, the movement is moulded. By controlling the path of a robot’s TCP as well as the orientation of its end effector, the goal of reaching a destination while following the desired path combining both translation and rotation is made possible by controlling the several degrees of freedom that constrain the movement. The result is a singular answer that specifies all the degrees of freedom at all time instants.

By using high order polynomials for the splines, a higher smoothness is imposed to the curve, ensuring both movements, rotation and extrusion speed have a smooth profile. Most importantly, 5th order polynomials will guarantee the second order derivatives of the equations are continuous at all their domain. As a result of using splines, continuity between any different polynomial is also ensured.

The degrees of freedom that directly affect the trajectory between two points among the ones withdrawn from the G-Code file are:

- Movement along the x axis.
- Movement along the y axis.
- Movement along the z axis.
- Rotation of the extrusion head on the plane that is tangent to the path.
- Material extrusion speed.

On the present document, the trajectories are to be described for planar slices of geometries, and for that reason only describe movement along the x and y axes. Later introduction of the z axis should not pose big constraints, as the generic slicer only makes the changing in layer increment the z coordinate value.

The rotation of the extrusion head is given by what is going to be called as *xy derivative*.

3.2 G Code points coordinates extraction

The generation of polynomials that creates the curve of the TCP's path itself is dependent from piece to piece, for natural reasons. An object may be created with any kind of computer aided design, the software choice being entirely up the user's preferences. When the design is finished, one typically exports the geometry as an *STL* file in order to later import it to the slicing software, however other file extensions are also used in replacement for *STL*. The generic slicing software will then generate G Code instructions that guide the FFF machine on creating the projected piece, with commands that control temperature, material feed rate and most importantly for this application, the positions along several time instants.

The goal is to be able to withdraw the coordinates of the points the nozzle will travel through, after these are generated by the slicer. These coordinates, among other factors, are later used to generate the polynomials that describe the trajectory curve, along with the velocities and accelerations at each one, on all three components (x, y, z) in space.

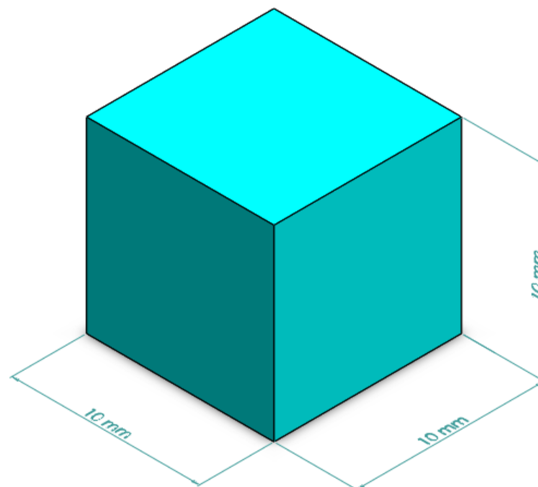

Figure 3.1 3D moduled cube with 10 mm edges

In order to pursue with the extraction of coordinates, a cube with 10 mm edges (Figure 3.1) was modelled for the sake of testing. This part was designed on *SOLIDWORKS* and exported as *STL*. The material of the part isn't relevant at this point, as this parameter is not taken into account and it suggested to be later parametrized so the results of this paper may be applied to different materials. The same happens with the nozzle diameter, as this will affect the layer thickness and therefore every z coordinate on any general FFF machine's reference frame. For this application, the slicing of the cube was made with the chosen diameter was 0.2 mm. After the sliced geometry is exported, work continues to extracting coordinates.

```
;LAYER:25
M106 S197.9
M204 S1000
M205 X10 Y10
;TYPE:WALL-INNER
;MESH:Cubo_10mm.STL
G1 F2700 X160.8 Y115.8 E94.20305
G1 X169.2 Y115.8 E94.28205
G1 X169.2 Y124.2 E94.36105
G1 X160.8 Y124.2 E94.44006
M204 S5000
M205 X30 Y30
G0 F9000 X160.5 Y124.5
M204 S1000
M205 X10 Y10
G1 F2700 X160.5 Y115.5 E94.52471
G1 X169.5 Y115.5 E94.60935
G1 X169.5 Y124.5 E94.694
G1 X160.5 Y124.5 E94.77865
M204 S5000
M205 X30 Y30
G0 F9000 X160.175 Y124.825
```

Figure 3.2 Fragment of G Code instructions for the 10 mm edge cube

When the G Code's instruction lines are analysed, distinct information may be encountered, regarding either temperature, nozzle diameters, printing sizes, among others. After the first lines where this information is found, the instructions start appearing on the form of commands starting typically with either the letter *G* or *M*. A piece of G code instructions for the referred cube may be found in Figure 3.2, for the case of the initial coordinates for layer number 25. The cube was sliced with the *CURA* software, using the *ULTIMAKER S5* as a printing base. Different FDM machines or slicing software's will naturally present different results in terms of G-Code generated, however the following model may be applied for any G-Code command sequence.

The different instructions relate to the fan turning on (*M106*), setting default accelerations (*M204*), other advanced settings (*M205*). The information that corresponds to the moving of the extrusion head may be found alongside the instructions *G0* (it is moving and not extruding material) and *G1* (it is moving and extruding material). Within the same layer, a *G0* command displays the x and y coordinates to where the nozzle will move to, as *G1* commands do the same alongside with displaying the rate of extrusion of material. Other commands that move the end effector may be found with specific geometries, like *G2* and *G3* for controlled arc moves among many others. For the simpler examples currently studied however, only *G0* and *G1* commands are looked after.

On a classic style slicer, the z coordinate is constant within the same layer, this meaning a changing in layer number will add the layer thickness to the previous value of z. Considering this and acknowledging the printed layer in G code is labelled as ";LAYER:0", the vertical component of a generic point i in a total of n point is given by (Eq. 3.1.

$$z_i = layer\ thickness + (layer\ number * layer\ thickness) \quad , \quad i \in \{0,1,\dots,n\} \qquad \text{(Eq. 3.1)}$$

Now it is known where the coordinates of the points are within the G Code, all is left is to create a program that selects and removes each one. Bearing in mind the following work, it was chosen to export these values as *.csv* files.

The program for this application with the cube was developed in *Python* language and it is displayed in Appendix A. When using the program, the user must insert the name of the file where the G Code is available and change the layer thickness for the desired one. The algorithm will a *.csv* file where the x, y, z coordinates for the n points covered by the extruder head are displayed.

Note that upon facing geometries with commands other than *G0* and *G1* after sliced, the fourth line of the code may be expanded to cover different commands by sampling replacing the set of numbers followed by the letter *G* compiled.

3.3 Linear interpolation of G code

Conventional slicers tend to behave differently when objects possess certain properties in geometry. One aspect that is worth looking in to are sharp edges. While working with articulated robot arms and FFF technology, this aspect will dictate if a part is in fact printable or not. If the nozzle encounters a sharp corner it may be unable to go through it. Sharp corners (or sharp edges), are assumed to be any at which the angle's module is lower than parameter defined as the sharp corner angle limit between two segments of the linear interpolation of a layer on three consecutive points. This angle is determined not as an accurately defined value, but as an assumption by default that may be changed depending on the case and geometry that are under study.

$$|Sharp\ Edge\ Angle\ | \leq Parameter$$

In order to have a first general impression of how the points generated by a specific slicer are spread in space, a three-dimensional visualization comes in hand. After visualizing one, conclusions can be made relating to aspects as points are dispersion along sharp edges, how spaced points are between each other in different sections of the part, among other aspects.

As a conventional slicer proceeds to the slicing along uniform layers on the front plane with constant z coordinate values, analysing a specific layer by itself is also beneficent. For objects like the discussed cube, the section is uniform along all the object verticality, as the same also reveals to be true for any constant section object. Sharp edges in two dimensions can then be thoroughly analysed.

In order to have a glimpse of the result on a fully printed cube, a three-dimensional plot of the linear interpolation of the points within G-Code was made and is visible in Figure 3.3. At a first glance, it is possible to notice that from the second layer until the top, the section is kept constant, even though it is rotated with a 90° angle around the z axis in order to provide the cube a higher structural stiffness.

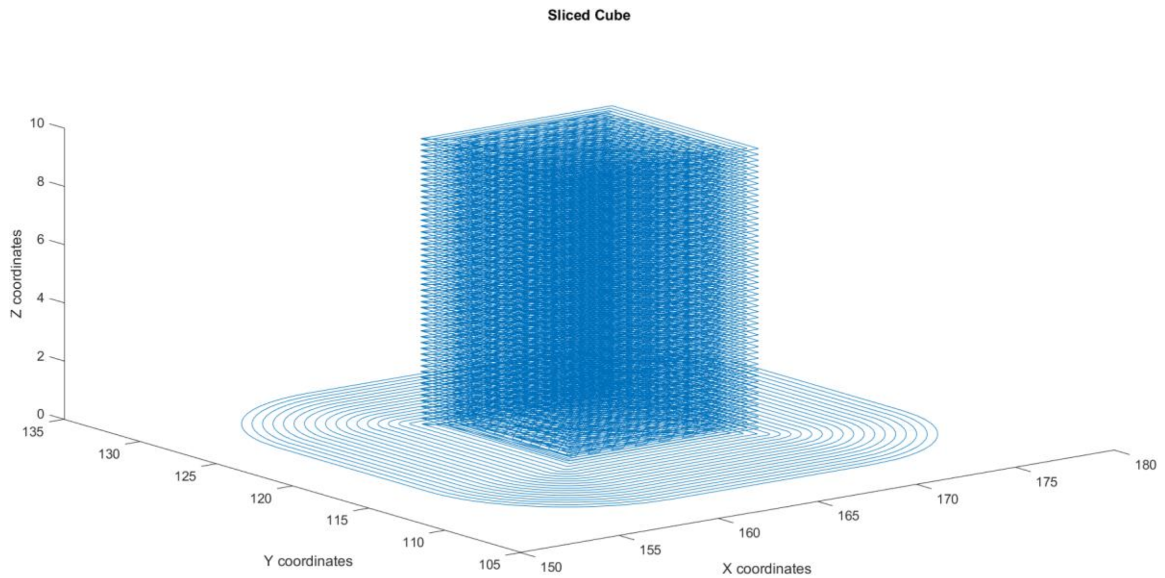

Figure 3.3 Sliced 10 mm edge cube

To analyze sharp edges closely, a single layer was extracted from the cube G-Code data points and its linear interpolation was plotted into a two-dimensional graph that is visible in Figure 3.4. It can be observed each layer of the object is composed by four contours of the shape (each decreasing in perimeter comparative to the previous) and a diagonal filling pattern for the interior of the shape. The transition lines between any diagonal in the filling pattern along with irregular lines that are observed besides the shapes mentioned before are the result of movements where the extrusion head is not extruding material (G0 commands) and are represented in the color red.

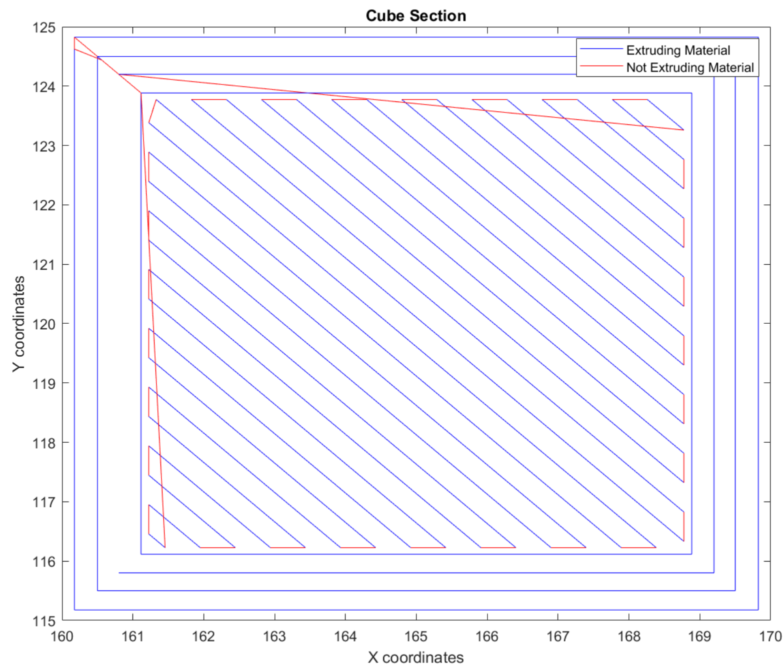

Figure 3.4 Cube section made visible in a single layer

Sharp edges that occur while the extrusion head is printing are visible on the corners of the outer layers defining the contours of the square. Angles of 90° between two lines in the linear interpolation are visible at every corner and are then classifying these as sharp corners, if the parameter defined is lower than this module value for the angle.

To understand the existence of sharp edges on other geometries, a similar example for linear interpolation on G-Code originated points was created for the cylinder in Figure 3.5. This object has a 10 mm diameter base with 10 mm height and it was also sliced with the same software as the cube, same nozzle diameter and printing specifications. The cylinder's section itself may be found in Figure 3.6. On this representation, red lines correspond to the ones where the material is not being extruded, or by other words, $G0$ commands. Blue lines by the other hand correspond to the instants where there is extrusion of material, or by other words, $G1$ commands.

Both cube and cylinder display 4 revolution with gradually lower perimeters creating the outer wall and a filling pattern inside. The differences are found on the inexistence of sharp edges caused by each circle being composed of small segments of a straight line, incrementing to the complete circle itself. As the angles between each line are smaller than angular parameter, it may be alleged no sharp corners are found within this geometry.

With the two examples, one may conclude that within the same layer, the existence of sharp edges is dependent on the chosen geometry of part to be extruded. Since the producing of parts within this project is not constrained to conventual layers parallel to the XY plane, avoiding sharp edges is possible.

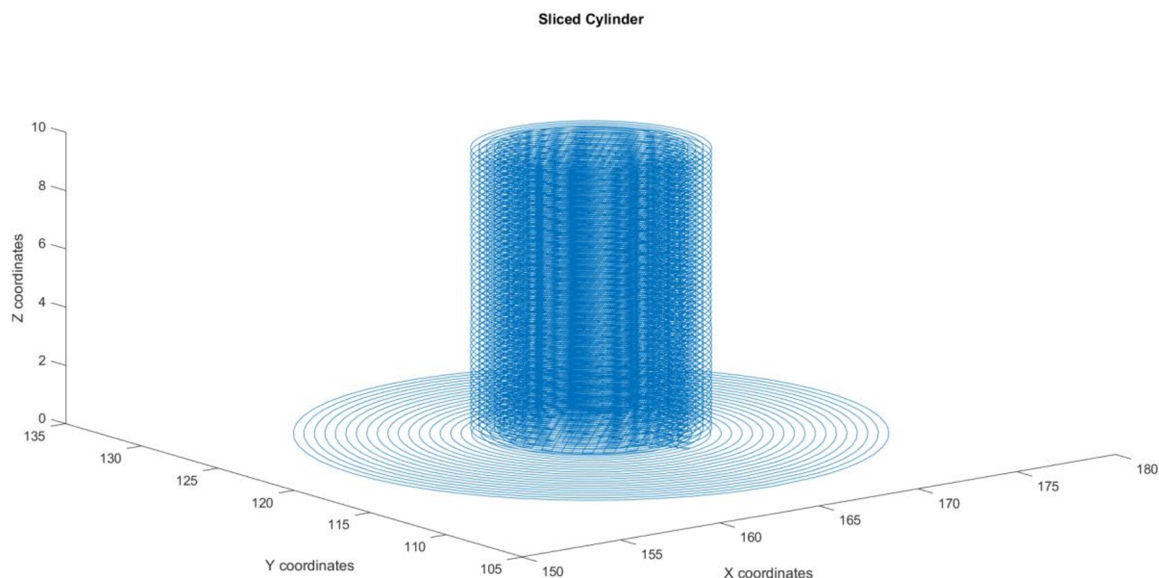

Figure 3.5 Sliced cylinder with 10 mm diameter and 10 mm height

Now that sharp edges are proved to exist, attention turns to solving this issue, as the robot is unable to perform turns like the ones discussed. For that matter, the path followed by the TCP goes to a smothering process.

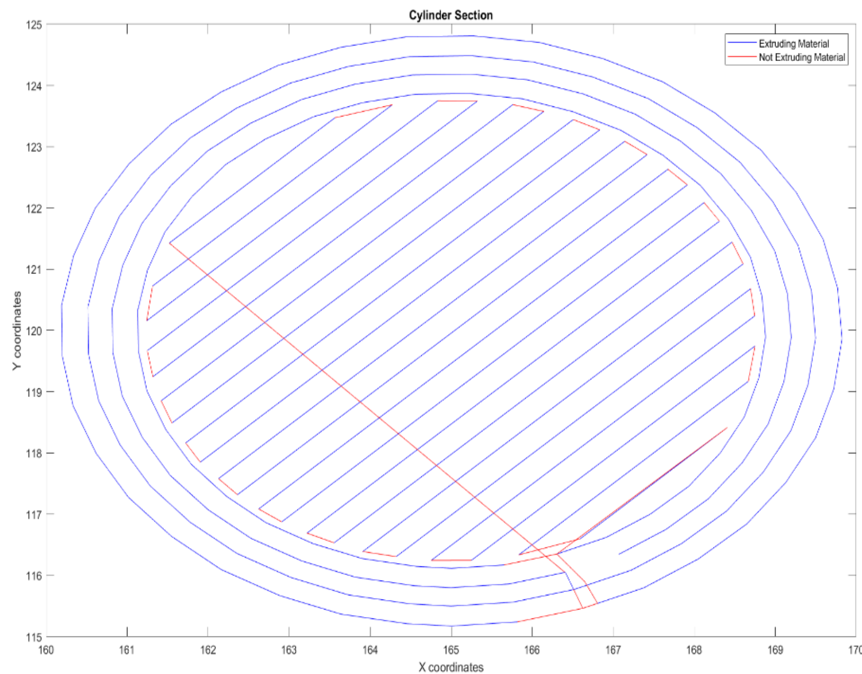

Figure 3.6 Cylinder section made visible in a single layer

While resolving the sharp edges problem, the printing resolution of the *G-Code* after slicing effect must be considered, as this is not controllable. On the examples above, the *Cura* software generates the list of point increments according to the followed standards of the software. This characteristic introduces new aspects to have in mind when applying to the future working method.

To have a clearer idea of the ideas that are being discusses, a representation of an octagon is visible in Figure 3.7. The octagon is the result of the linear interpolation of the points from a sliced cylinder's section, that is to suffer an FFF process to become a three-dimensional object. This example is merely illustrative and to be later compared with a generic section, as each of the regular octagon interior angles are valued at $135º$ and for that reason not considered a sharp edge itself.

The resulting lines from the linear interpolation can then be used to generate the circular section of the future cylinder while having two methods in mind, that are referred to as:

1. Inside interpolation.
2. Outside interpolation.

On **inside interpolation**, the product of the interpolation of the original octagon is an inscribed circle that touches the lines from the linear interpolation halfway between certain points i and $i + 1$. The finishing result is a circle that is smaller than the theoretical, and for this reason this method is only used when indispensable. This interpolation may need to be complemented with the addition of new points to the original G-Code coordinates list, for the cases where the length of two followed segments are disparate from one another. These cases are analyzed further ahead in this document.

On **outside interpolation** by the other hand, the product of the interpolation of the original octagon is a circumscribed circle that passes through the points taken from the *G-Code*. As seen

posteriorly in depth on this document, the polynomial is calculated accessing the direction of the derivatives of the tangent lines at the i-th point of a set of n points, calculated through point $i-1$ and point $i+1$, as $1 \leq i \leq n$.

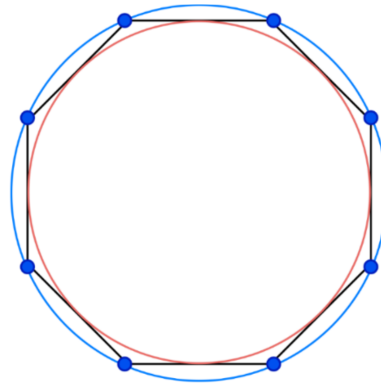

Figure 3.7 Section of a traditionally sliced cylinder with low resolution G-Code using inside and outside interpolation

The decision upon performing an inside or outside interpolation relies on the definition of the angle parameter between. The angle is measured between the two lines resulting from the linear interpolation that intersect at the point i. A vector $\overrightarrow{v_1}$ and a vector $\overrightarrow{v_2}$ are created following the linear interpolation segments, as displayed in Figure 3.8. The angle parameter is then calculated through the dot product of the two vectors and then converted to degrees. This calculation is made from Eq. 3.1 and Eq. 3.2.

$$\cos(\overrightarrow{v_1} \hat{} \overrightarrow{v_2}) = \frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{\|\overrightarrow{v_1}\| \cdot \|\overrightarrow{v_2}\|} \quad \text{(Eq. 3.1)}$$

$$angle_{in/out} = \cos^{-1}(\overrightarrow{v_1} \hat{} \overrightarrow{v_2}) \quad \text{(Eq. 3.2)}$$

When the $angle_{in/out}$ parameter is defined, the algorithm will verify what happens on every set of $\overrightarrow{v_1}$ and a vector $\overrightarrow{v_2}$ vectors, measuring the angle between them. If this value exceeds the chosen parameter angle (in degrees), the case will proceed for outside interpolation, else it proceeds for inside interpolation. Both processes are explained in upcoming chapters.

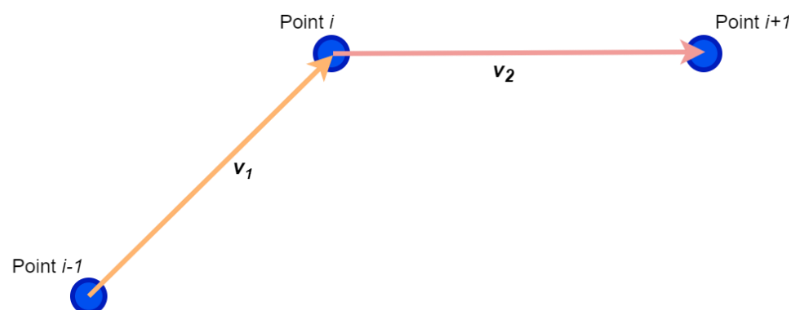

Figure 3.8 Angle calculation between two lines on the linear interpolation of G-Code

3.4 Extrusion effect: G0 and G1 commands

As understood by now, the study of a layer within a specific geometry relies on the definition of coordinates within the G-Code. These coordinates are always accompanied by the correspondent extrusion command. Even though several commands are known to exist (as explained on the *Literature Review* chapter), the study is carried out between the two most commonly found: *G0* and *G1*. It is of most importance to be able to differ whenever the extruder is moving and extruding material (*G1* command) or simply moving with no extrusion (*G0* command).

When the extruder faces a corner defined in two dimensions, the information regarding the existence of material extrusion at that time will determine if there is indeed a sharp corner to be considered by the following algorithms, and if it is to be eliminated following the proposed method on the upcoming chapters.

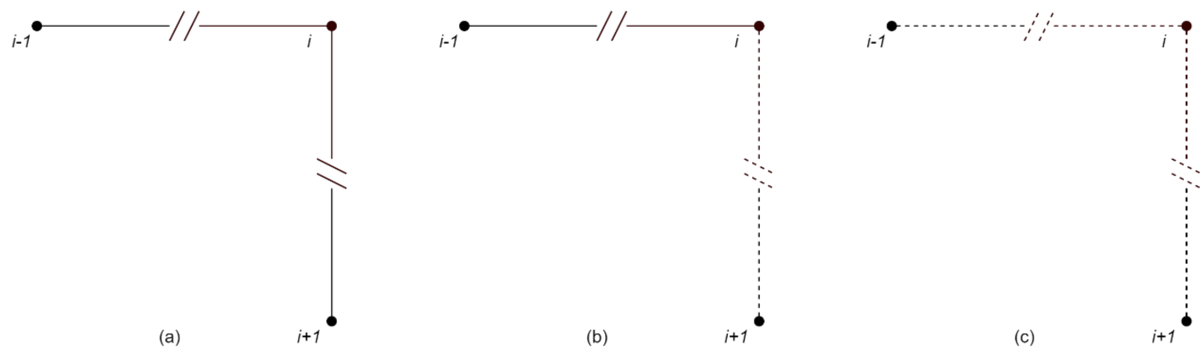

Figure 3.9 Extrusion on corners, where dashed lines represent *G0* commands and continuous lines define *G1* commands. (a) Corner defined between two *G1* commands (b) Corner defined between a *G1* command and a *G0* command (c) Corner defined between two *G0* commands

In order to be considered a sharp edge, a corner must be defined by two *G1* commands, as represented in Figure 3.9. Regarding the G-Code instructions, the two commands are represented along the coordinates of points i and $i+1$. If both these points correspond to *G1* commands it means material is being extruded, and the movement is limited by physical constraints that prevent certain trajectories. For that reason, if and there is indeed a sharp edge, it is to be considered. If one of the commands or both of them correspond to the inexistence of material extrusion, the sharp edge is not considered due to absence of impossibility of describing the correspondent sharp edges, for the cases where they actually exist.

The polynomial is to be fitted on every command correspondent coordinates where there is extrusion of material. For the cases where there is no extrusion between two different points (Figure 3.9 (c)), the point is excluded from the list of points to be interpolated, as there is no point on making the polynomial pass by those coordinates.

3.5 Outside Interpolation

As mentioned above, the study is to be carried out dividing and explaining both inside and outside interpolation.

When going through the list of *G-Code* coordinates and evidence in form of angles between the linear interpolation lines do not conduct to the inside interpolation, by comparison with the set angle parameter, outside interpolation surges as a solution.

For that reason, the so intitled **outside interpolation** algorithm process chain is fully carried out.

The presented method consists of defining the polynomial going through a certain point i by the direction of its derivate on the xy plane. The direction of the derivative provides the direction that the polynomial will follow on the generic point.

To better understand what the method proposed, a 5th degree polynomial p is defined in Eq. 3.3.

The particularity of p relies on it being defined by vectors of constants instead of singular scalar constants. These constant vectors are represented by their components in two dimensions, as for example $[ax\ ay]$ that corresponds to the vector $\vec{a}$ (Table 3.1). When using this notation, p describes the parametric curve on the two dimensions x and y between every point along the *G-Code* coordinates, with respect to the time parameter, t.

$$p = [ax \quad ay] \times t^5 + [bx \quad by] \times t^4 + [cx \quad cy] \times t^3 + [dx \quad dy] \times t^2 + [ex \quad ey] \times t + [fx \quad fy] \qquad \text{(Eq. 3.3)}$$

On Figure 3.10, a generic polynomial that was first introduced to describe how a function would behave when interpolating the points of an octagon is represented along the linear (in black) and polynomial interpolation (in blue) along the generic points $i-1$, i and $i+1$. The points referred are collected from the *G-Code* data.

Table 3.1 Constant vectors and their components

Constant Vectors	Vector Components
$\vec{a}$	$[ax\ ay]$
$\vec{b}$	$[bx \quad by]$
$\vec{c}$	$[cx \quad cy]$
$\vec{d}$	$[dx \quad dy]$
$\vec{e}$	$[ex \quad ey]$
$\vec{f}$	$[fx \quad fy]$

At point i, a green line is displayed representing the direction of the xy plane derivative direction. As it is possible to observe, the direction of the derivative at point i is the same as the direction of the 5th order polynomial at the same point. The hypostasis of the method is then proposed and the explanation regarding how to obtain broth the green line and blue curve is to be detailed.

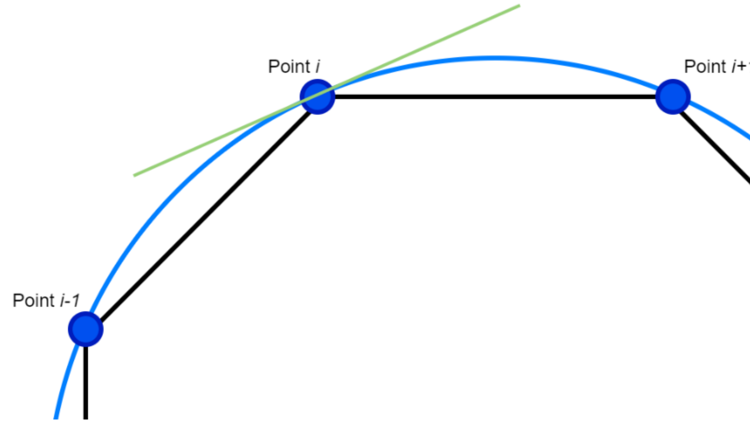

Figure 3.10 Polynomial following the direction of the xy derivative on outside interpolation

To achieve the full definition of the polynomial, all the vector constants that it is defined from must be determined. This is where the $\boldsymbol{xy}$ **derivates** are introduced, as a form of calculating the vector constants.

3.5.1 XY derivatives for outside interpolation

The goal of the outside interpolation algorithm is to calculate the input constants to then be used to define the polynomial p between two time instants. For this purpose, a first algorithm was created to create a list of xy derivatives directions at every point that is going to be user for the polynomial interpolation.

In order to understand how these derivatives are calculated, let Figure 3.11 be on focus for the following explanation.

The linear interpolation lines of Figure 3.10 were replaced by the generic vectors v_1 and v_2 are created from points $i-1$, i and $i+1$, and represented by:

$$v_1 = [x_i - x_{i-1} \quad y_i - y_{i-1}] \quad , \quad 1 < i < n \tag{Eq. 3.4}$$

$$v_2 = [x_{i+1} - x_i \quad y_{i+1} - y_i] \quad , \quad 1 < i < n \tag{Eq. 3.5}$$

Since the polynomial interpolation is to be made based on the same points as described in Figure 3.10 ($i-1$, i and $i+1$), and the vectors are defined for the same set of points, conclusions can be made. The direction of the xy plane derivative must follow the direction of the vector that results from the sum of vectors v_1 and v_2, as represented on Figure 3.11.

The xy derivative is then calculated through the normalized sum of vectors v_1 and v_2, or by other words, the normalization of vector v, as seen below.

$$v = \frac{v_1 + v_2}{\|v_1 + v_2\|} = \frac{[(x_i - x_{i-1}) + (x_{i+1} - x_i) \quad (y_i - y_{i-1}) + (y_{i+1} - y_i)]}{\|[(x_i - x_{i-1}) + (x_{i+1} - x_i) \quad (y_i - y_{i-1}) + (y_{i+1} - y_i)]\|} \qquad \text{(Eq. 3.6)}$$

$$v = \frac{[x_{i+1} - x_{i-1} \quad y_{i+1} - y_{i-1}]}{\|[x_{i+1} - x_{i-1} \quad y_{i+1} - y_{i-1}]\|} \qquad \text{(Eq. 3.7)}$$

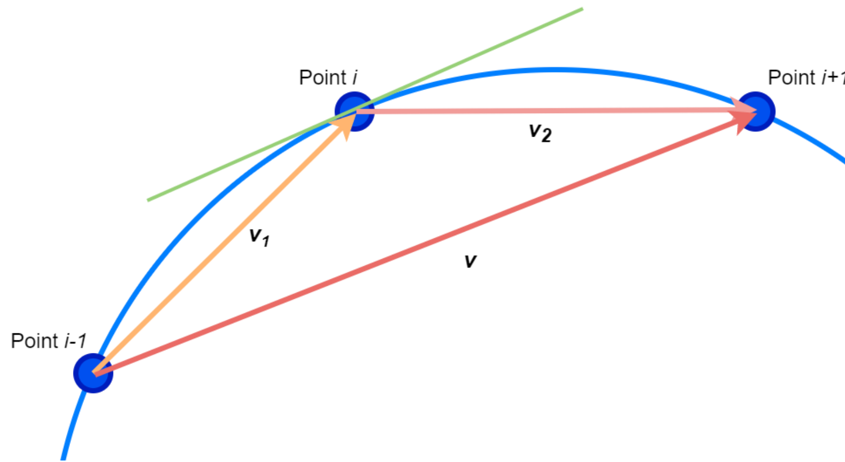

Figure 3.11 Method for calculating the direction of the xy derivative on outside interpolation

The created algorithm applies this method for every point along the set of coordinates originally from the *G-Code* file, normalizing the vector in order to obtain values that can be used to describe derivative orientation.

The first and last point of the data, however, are treated differently from the rest. The algorithm is not able to analyse point $i - 1$, when i is 1, and point $i + 1$, when i corresponds to the last point of the data ($i = n$). When $i = 1$, the direction of the derivative is the direction of the vector that connects the first and second points of the set. When $i = n$ by the other hand, the direction of the derivative is the direction of the vector that connects the penultimate and last points of the set.

The information regarding derivative direction calculation is summarized in Table 3.2.

Table 3.2 Direction of the xy derivative for the generic point on outside interpolation

Points analyzed	**Direction of the xy derivative**
$\boldsymbol{i = 1}$	$\frac{[x_2 - x_1 \quad y_2 - y_1]}{\|[x_2 - x_1 \quad y_2 - y_1]\|}$
$\boldsymbol{1 < i < n}$	$\frac{[x_{i+1} - x_{i-1} \quad y_{i+1} - y_{i-1}]}{\|[x_{i+1} - x_{i-1} \quad y_{i+1} - y_{i-1}]\|}$
$\boldsymbol{i = n}$	$\frac{[x_n - x_{n-1} \quad y_n - y_{n-1}]}{\|[x_n - x_{n-1} \quad y_n - y_{n-1}]\|}$

The data calculated through running the algorithm is stored inside a *.csv* file for later use when it comes to generate the trajectory polynomials.

3.5.2 Polynomial interpolation

As mentioned along this chapter, the outside polynomial interpolation is to occur whenever the inside interpolation requirements are not fulfilled. For this reason, the first step of the algorithm is to assess the truthfulness of this condition, by the same method used within the inside interpolation. On the new scenario, and on opposition to the case before, all the situations where the inside interpolation is not applicable are selected.

The definition of the polynomials is made through border conditions on both polynomial, its first derivate and second derivative. For each of these, equalities are made in order to establish a number of enough border conditions. For the case of 5th degree polynomials within the two dimensional space, this number is six.

$$number\ of\ necessary\ border\ conditions = 6$$

As border conditions are to be applied to the generation of a 5th order polynomial representing the trajectory of the nozzle during the FFF process, it can be said it's derivative will describe the velocity from which the trajectory is gone through, and finally its second derivative concerns the acceleration at every point along that curve. For that reason, border conditions evaluate:

- Position of the extreme point on the polynomial itself.
- Velocity on the extreme point by the polynomial first derivative.
- Acceleration on the extreme point by the polynomial second derivative

The necessary data for the definition of the border conditions is available from previous steps within this document, except for the second derivative conditions, by other words, concerning acceleration at the extreme points.

To assure continuity between polynomials that are defined between two time instants connected to respectively two positions, **acceleration is defined as being zero at all extreme points** within the interval that is under study.

To be able to better understand the algorithm that generates the several polynomials, Table 3.3 was created. The expressions are based on applying equation p to the two time instants t_1 and t_2. As it was chosen to use a time normalization method, all the coefficients are calculated for the time interval $[0,1]$, as the process behind normalization is explained on a subchapter ahead.

The border conditions are to be equalled to the expressions within the same row, creating six equations. This method then allows the calculation of the six constant vectors: $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, $\vec{e}$ and $\vec{f}$.

Firstly, the polynomial is specified for the time instant, as the border condition corresponds to the coordinates of the point within the layer, in two dimensions. This is the source of the two first equations.

Table 3.3 Definition of border conditions for outside interpolation

Time instant	Expression	Border condition
$t = 0 = t_1$	$[ax \quad ay] \times t_1^5 + [bx \quad by] \times t_1^4 + [cx \quad cy] \times t_1^3 + [dx \quad dy] \times t_1^2 + [ex \quad ey] \times t_1 + [fx \quad fy]$	$[x_{i-1} \quad y_{i-1}]$
	$[ax \quad ay] \times 5t_1^4 + [bx \quad by] \times 4t_1^3 + [cx \quad cy] \times 3t_1^2 + [dx \quad dy] \times 2t_1 + [ex \quad ey]$	$k_1 \times dxy_{i-1}$
	$[ax \quad ay] \times 20t_1^3 + [bx \quad by] \times 12t_1^2 + [cx \quad cy] \times 6t_1 + [dx \quad dy]$	$[0 \quad 0]$
$t = 0 = t_2$	$[ax \quad ay] \times t_2^5 + [bx \quad by] \times t_2^4 + [cx \quad cy] \times t_2^3 + [dx \quad dy] \times t_2^2 + [ex \quad ey] \times t_2 + [fx \quad fy]$	$[x_i \quad y_i]$
	$[ax \quad ay] \times 5t_2^4 + [bx \quad by] \times 4t_2^3 + [cx \quad cy] \times 3t_2^2 + [dx \quad dy] \times 2t_2 + [ex \quad ey]$	$k_2 \times dxy_i$
	$[ax \quad ay] \times 20t_2^3 + [bx \quad by] \times 12t_2^2 + [cx \quad cy] \times 6t_2 + [dx \quad dy]$	$[0 \quad 0]$

The first derivative border conditions are represented by $k_1 \times dxy_{i-1}$ and $k_2 \times dxy_i$ for respectively t_{i-1} and t_i. Here, two scalar constants ($k_1$and k_2) are multiplied to the direction of the xy derivative. These parameters must be equal in order to ensure symmetry on the polynomial, and they are manually defined in order to generate the best approximation possible. The constants $k_1$and k_2 are parametrical and are used to amplify or reduce what is referred to as the *strength of the derivative*. The derivative itself is represented by the vectors dxy_{i-1} and dxy_i for the two extreme points of the generic polynomial, and it is obtained from the previously explained algorithm.

$$k_1 = k_2 \,, \qquad 0 < \{k_1, k_2\} < 1 \qquad \text{(Eq. 3.8)}$$

Finally, the second derivative of the polynomials has the null two dimensional vectors as its border condition. The acceleration is set to be null on the extreme points of every polynomial because abrupt variations on the acceleration are undesired when trying to create a smooth trajectory. Since the acceleration is null on the borders of the polynomials, it is guaranteed its variation will always be relatively low upon the entire path.

The set of 6 equations that are withdrawn from Table 3.3 are then solved by the algorithm. The result are 6 constant vectors that are exported to an *.csv* from which is possible to build the polynomial to create every piece of the future curve, respecting the conditions stated for splines and polynomials in general.

3.6 Inside Interpolation

On this sub-chapter, the method for using inside interpolation is carried out along its explanation, as well as other aspects that interfere with the process simultaneously.

Before any interpolation may be made, attention must be directed throughout the data, having in mind the conditions imposed by *FFF* processes. The point coordinates that were extracted from the G-Code file may not be fully ready to use as interpolation data.

3.6.1 Data points analysis

When an extruder encounters a sharp corner along its path, it is usually unable to perform the exact corner as firstly described, being forced to detour from the original path. The new scenario allows the nozzle to perform the sharp corner, however deviations from the finished projected part tend to occur. Figure 3.12 displays an example of how a typical 90º corner within the same layer would typically be gone through.

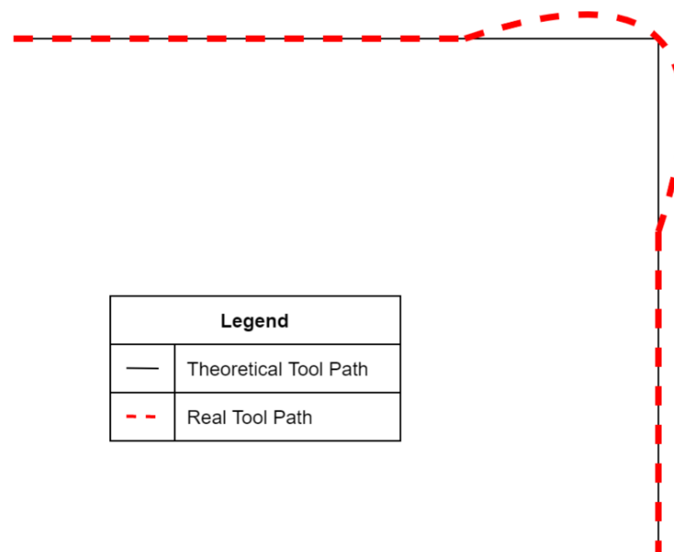

Figure 3.12 Conventional tool path compared with the theorical path when encountering a sharp corner

The problem mentioned above needs to be solved so that minimum impact is caused on the finished printed part. For that reason, and for the effect of inside interpolation, the chosen method is to replace the points that cause these sharp edges, extract the new list and make the inside or outside interpolation according to each circumstance.

For the case when a sharp edge stands alone withing many non-sharp edges, this method involves adding two extra points right before and after every point causing the sharp edge, where the new path is going to be created from. Both points are coincident with the lines from the linear interpolation of the cloud of arguments. The point causing the sharp edge is then excluded from the

real path to be followed, that is now possible to be run through the robot as the new angle forms a corner that is not as sharp as before.

The distance from each one to the sharp edge is said to be symmetric, as its module varies with the robot's maximum speed that is by itself linked with the maximum speed material can be deposited. The radius of the corners will then depend on the velocities that robot can achieve, with lower curvature radius corners requiring higher velocities and accelerations by the robot itself.

To develop a methodology that allow the analysing of each scenario, an algorithm was created via a *MATLAB* program. The theory behind the method and program itself is then explained so the thinking sequence may be followed.

The idea is to select three points within the set of n points that were exported from the *G-Code* file of a certain geometry. The second point corresponds to the one on the border, where the value of the angle is measured between the two lines generated from the line interpolation. These two lines, as it is seen in Figure 3.13, connect the first to the second point and the second to the third points, respectively. Here, the lines are represented by its projection on the XY plane, as happens on any uniform layer with constant z coordinate.

For simplicity purpose, the experiment is made for points within the same layer, this meaning the z coordinate will not affect the process as all the points have the same value for it.

Start by considering there are three points between the n points ($1 \leq i \leq n$) to be analyzed (represented in Figure 3.13), these being:

- **Point $\boldsymbol{i-1}$**, the point before the sharp edge with coordinates $(p0x\ ,p0y)$;
- **Point $\boldsymbol{i}$**, the point causing the sharp edge with coordinates $(p1x\ ,p1y)$;
- **Point $\boldsymbol{i+1}$**, the point after the sharp edge with coordinates $(p2x\ ,p2y)$;

The lines connecting the points are distinguished by its length in Figure 3.13, as the line with length d_1 connects point $i-1$ to point i and the line with length d_2 connects point i to point $i+1$. As said before, if the angle's module between the red and green line is lower than a parameter that is to be set, then it is considered a sharp edge. For this effect, the algorithm's goal is firstly to determine where this is happening from a set of points when linear interpolation is considered.

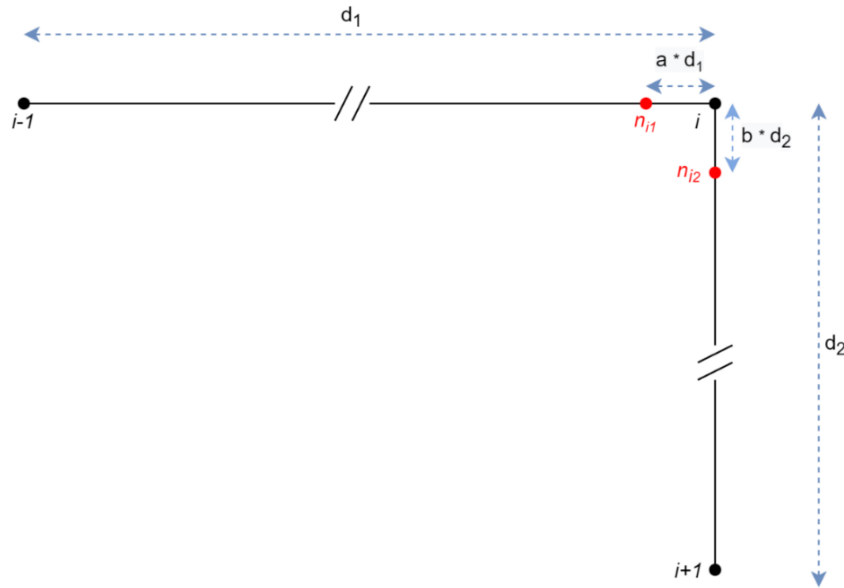

Figure 3.13 Points to be generated in the event of one single sharp edge

To determine if there is indeed a sharp corner, the algorithm calculates the angle between the two lines resulting from the linear interpolation that intersect at the point i. A vector u is created from the line right before and a vector v based on the line right after. The angle is then calculated through the dot product of the two vectors and then converted to degrees. This calculation is made from Equation 3.9.

$$\cos(\vec{u}\hat{\ }\vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|} \quad \text{(Eq. 3.9)}$$

For each one of the n points, if the result of the calculation above returns a value for which the border point is not part of a sharp edge, the algorithm ends. By the other hand, if there is effectively the existence of the sharp edge, and at the same time also qualifies for inside interpolation, the algorithm proceeds.

Now sharp edges are identified, the step that follows is to create the new points that are to replace point i.

The new points coexist on the original interpolation lines, right before and after the point causing the sharp edge situation. The distance from between the original and the newly created positions is set by defining a percentage of the total length of each line, bearing in mind for symmetrically purposes the distance module must be equal between each new point and the sharp edge. On this step, two new points are created.

- **Point n_{i1}**, on which the distances to the original point $i - 1$ and point i rely on the definition of a parameter a. This parameter varies from 0 to 1 and defines the profile of the new path by defining the position of the new coordinate.

- **Point $\boldsymbol{n_{i2}}$**, on which the distances to the original point i and point $i+1$ rely on the definition of a parameter b. This parameter varies from 0 to 1 and defines the profile of the new path by defining the position of the new coordinate.

Mathematically, the new coordinates are represented as follows, bearing in mind i varies from 2 to the number of original data points.

$$\begin{cases} n_{i1x} = (i-1)_x(a) + i_x(1-a) \quad , \quad 0 < a \leq 1 \\ n_{i1y} = (i-1)_y(a) + i_y(1-a) \quad , \quad 0 < a \leq 1 \end{cases} \quad \text{(Eq. 3.10)}$$

As the same occurs with point n_{i2}.

$$\begin{cases} n_{i2x} = i_x(1-b) + (i+1)_x(b) \quad , \quad 0 < b \leq 1 \\ n_{i2y} = i_y(1-b) + (i+1)_y(b) \quad , \quad 0 < b \leq 1 \end{cases} \quad \text{(Eq. 3.11)}$$

While satisfying the symmetrically condition:

$$a \times d_1 = b \times d_2 \quad \text{(Eq. 3.12)}$$

After the two new points are created, they are added to the previous list obtained from the original *G-Code* file. The points are inserted in the correct order as seen graphically, replacing the original point i by following the order in Eq. 3.13, Eq. 3.14 and Eq. 3.15.

$$x = \left[x(1{:}\,i-1)\,, \quad n_{i1_x}, \quad n_{i2_x}, \quad x(i+1end)\right] \quad \text{(Eq. 3.13)}$$

$$y = \left[y(1{:}\,i-1), \quad n_{i1_y}, \quad n_{i2_y}\,, \quad y(i+1{:}\,end)\right] \quad \text{(Eq. 3.14)}$$

$$z = [z(1{:}\,i-1), \quad z(i)\,, \quad z(i+1), \quad z(i+1{:}\,end)] \quad \text{(Eq. 3.15)}$$

For the cases when there are two followed sharp edges, as it is seen several times when analysing the infill pattern of the cube's section in Figure 3.14. For these cases, and possibly others depending on specifications of the object to be modelled, the algorithm must adapt. While adapting, a balance is sought between the path being described with the shortest set of new points while the finishing result does not lack on definition quality.

On the cases where there are two followed sharp edges connected by a short line in length while compared with the respective predecessor and successor, the new points generation is made differently from the case before.

To avoid creating additional points that would break the symmetrically on a sharp edge scenario by introducing extra points on smaller linear interpolation lines, an exceptional case is introduced. Here, the maximum length d_2 is parameterized so that this measurement defines a maximum length of a segment of line that qualifies for this method.

When defined, segments with length d_2 or lower are divided in two new segments when multiplied by the new parameter b, defining the position of point n_{i2}. Along the following demonstrations, and to be later explained, this parameter is assumed to be 0.5, guarantying symmetricity when dividing the original segment.

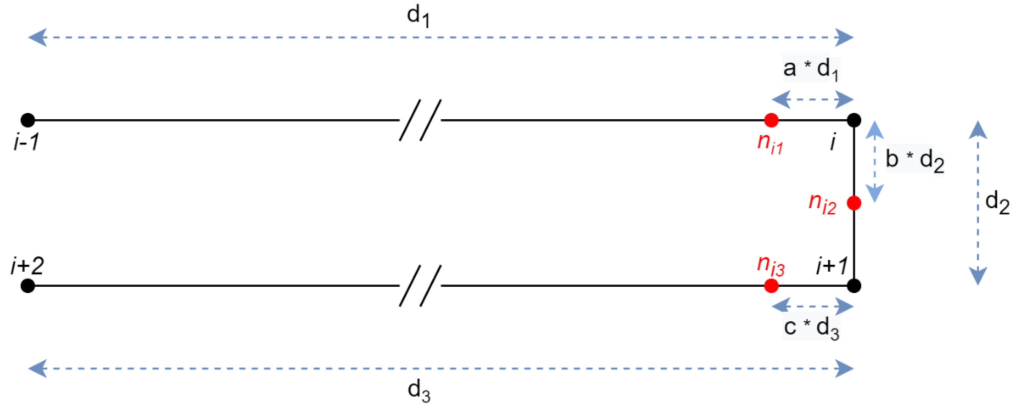

Figure 3.14 Points to be generated in the event of two followed sharp edges

The new points are calculated through the following expressions, bearing in mind i varies from 2 to the number of original data points.

$$\begin{cases} n_{i1x} = (i-1)_x(a) + i_x(1-a) \quad , \quad 0 < a \leq 1 \\ n_{i1y} = (i-1)_y(a) + i_y(1-a) \quad , \quad 0 < a \leq 1 \end{cases} \qquad \text{(Eq. 3.16)}$$

$$\begin{cases} n_{i2x} = i_x(1-b) + (i+1)_x(b) \quad , \quad 0 < b \leq 1 \\ n_{i2y} = i_y(1-b) + (i+1)_y(b) \quad , \quad 0 < b \leq 1 \end{cases} \qquad \text{(Eq. 3.17)}$$

$$\begin{cases} n_{i3x} = (i+1)_x(1-c) + (i+2)_x(c) \quad , \quad 0 < c \leq 1 \\ n_{i3y} = (i+1)_y(1-c) + (i+2)_y(c) \quad , \quad 0 < c \leq 1 \end{cases} \qquad \text{(Eq. 3.18)}$$

While satisfying the symmetrically condition:

$$a \times d_1 = b \times d_2 = c \times d_3 \qquad \text{(Eq. 3.19)}$$

If $b = 0.5$, the equations above are simplified to:

$$a = \frac{d_2}{2 \times d_1} \qquad \text{(Eq. 3.20)}$$

$$c = \frac{d_2}{2 \times d_3} \qquad \text{(Eq. 3.21)}$$

The list of points is updated to include the new additions, as $1 \leq i \leq n$.

$$x = [x(1{:}i-1), \quad n_{i1_x}, \quad n_{i2_x}, \quad n_{i3_x}, \quad x(i+2{:}end)] \qquad \text{(Eq. 3.22)}$$

$$y = [y(1{:}i-1), \quad n_{i1_y}, \quad n_{i2_y}, \quad n_{i3_y}, \quad y(i+2{:}end)] \qquad \text{(Eq. 3.23)}$$

$$z = [z(1{:}i-1), \quad z(i), \quad z(i), \quad z(i+1), \quad z(i+2{:}end)] \qquad \text{(Eq. 3.24)}$$

On the cases where three new points are created (Figure 3.14), replacing two points from original data, it is important to ensure the newly created points are not run through the algorithm once again. For this reason, a condition is inserted on the piece of algorithm concerning the three new

points creation. This condition ensures the algorithm will skip the iteration afterwards to the creation of the three points, after adding them to the final point list to use on interpolation.

3.6.2 XY Derivatives for inside interpolation

As it was seen while studying outside interpolation, the requirements to generate the necessary polynomial coefficients are based on end effector position coordinates and the direction of the curve derivative at the same point, among other factors concerning the optimization of the process.

For the case of inside interpolation, the process of obtaining the *XY* derivatives (as referred to before) is on all its form identical to the one used during outside interpolation, exemplified by Figure 3.15. The result is the derivative direction for the generic points n_{i1}, n_{i2} for the cases where there are two new points and the extra point n_{i3} when a third point is also created.

For the points that were not lately created, the direction of the derivative is unaltered and follows the information detailed on Table 3.4. On these cases however, as the points are fitted through the outside interpolation, their coefficients are calculated only on the last chapter. For that reason, the following explanation concerns only the inside interpolation chapter, or by other words, the 5th degree interpolation that is applicable on the new points.

The major difference when focusing on inside interpolation are the inputs that are used, these being the point coordinate list. As explained on the subchapter above, the sharp edges forced the replacement and addition of several points to the original *G-Code* list. For this reason, the previous list of XY derivatives directions that was created specifically for outside interpolation is no longer valid as it only concerns the coordinates used for outside interpolation.

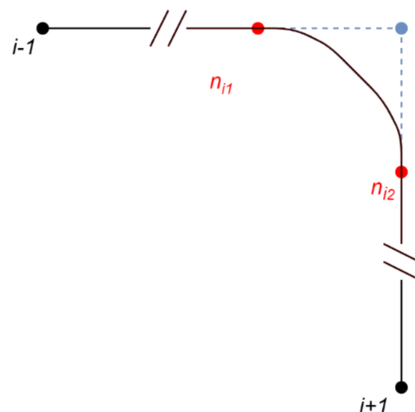

Figure 3.15 Representation of a generic polynomial following the xy derivative direction on points n_{i1} and n_{i2}

As the construction of the polynomial for the inside interpolation requires the values of the *XY* derivatives direction in order to be implemented, this is calculated through an algorithm that contemplates two different cases: when a sharp edge introduces **two new points** and when a sharp edge introduces **three new points**. The two cases that are now introduced are represented mathematically on Table 3.4.

Please consider the points the generic points $i-1$, i, $i+1$, n_{i1}, n_{i2} and n_{i3} in Figure 3.15. If the two new points case is faced, only n_{i1} and n_{i2} is relevant. For the case of three new points, all n_{i1}, n_{i2} and n_{i3} are considered. The *XY* derivatives on points n_{i1}, n_{i2} and n_{i3} have the following directions:

- On point n_{i1}, direction of the normalised vector connecting points n_{i1} and i.
- On point n_{i2}, direction of the normalised vector connecting points i and n_{i2}.
- On point n_{i3}, direction of the normalised vector connecting points $i+1$ and n_{i3}.

Table 3.4 Direction of the xy derivative for the generic point on inside interpolation

Alternative Case	New point	XY Derivative Direction
2 new points	n_{i1}	$\frac{[i_x - n_{i1_x} \quad i_y - n_{i1_y}]}{\lVert[i_x - n_{i1_x} \quad i_y - n_{i1_y}]\rVert}$
	n_{i2}	$\frac{[n_{i1_x} - i_x \quad n_{i1_y} - i_y]}{\lVert[n_{i1_x} - i_x \quad n_{i1_y} - i_y]\rVert}$
3 new points	n_{i1}	$\frac{[i_x - n_{i1_x} \quad i_y - n_{i1_y}]}{\lVert[i_x - n_{i1_x} \quad i_y - n_{i1_y}]\rVert}$
	n_{i2}	$\frac{[n_{i1_x} - i_x \quad n_{i1_y} - i_y]}{\lVert[n_{i1_x} - i_x \quad n_{i1_y} - i_y]\rVert}$
	n_{i3}	$\frac{[n_{i3_x} - (i+1)_x \quad n_{i3_y} - (i+1)_y]}{\lVert[n_{i3_x} - (i+1)_x \quad n_{i3_y} - (i+1)_y]\rVert}$

The algorithm may then be executed having the new list of point coordinates as input, and the new data is again stored inside a *.csv* file for later use when it comes to generate the trajectory polynomials.

3.6.3 Polynomial Interpolation

The process used for inside interpolation is in all its form similar to the one used for outside interpolation. The major characteristics that differ the two of them are the inputs (new coordinates, inside interpolation derivative directions) and the new constants $k_3$and k_3 to define the "*strength of the derivative"* at each time limit interval, for every polynomial. In order to ensure continuity between polynomials that are defined between two time instants connected to respectively two positions, **acceleration is again defined as being zero at all extreme points** within the interval that is under study.

The expressions behind the algorithm that generates the several polynomials may be found in Table 3.5, where $i \in [1\,, n]$ and n represent the number of created polynomials. This method again allows the calculation of the six constant vectors: $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, $\vec{e}$ and $\vec{f}$ (Table 3.1).

The polynomial is specified for the time instant, as the border condition corresponds to the coordinates of the point within the layer, in two dimensions, creating the two first equations.

Table 3.5 Definition of border conditions for inside interpolation

Time instant	Expression	Border condition
$\boldsymbol{t = 0 = t_1}$	$[ax \quad ay] \times t_1^5 + [bx \quad by] \times t_1^4 + [cx \quad cy] \times t_1^3 + [dx \quad dy] \times t_1^2 + [ex \quad ey] \times t_1 + [fx \quad fy]$	$[x_{i-1} \quad y_{i-1}]$
	$[ax \quad ay] \times 5t_1^4 + [bx \quad by] \times 4t_1^3 + [cx \quad cy] \times 3t_1^2 + [dx \quad dy] \times 2t_1 + [ex \quad ey]$	$k_3 \times dxy_{i-1}$
	$[ax \quad ay] \times 20t_1^3 + [bx \quad by] \times 12t_1^2 + [cx \quad cy] \times 6t_1 + [dx \quad dy]$	$[0 \quad 0]$
$\boldsymbol{t = 1 = t_2}$	$[ax \quad ay] \times t_2^5 + [bx \quad by] \times t_2^4 + [cx \quad cy] \times t_2^3 + [dx \quad dy] \times t_2^2 + [ex \quad ey] \times t_2 + [fx \quad fy]$	$[x_i \quad y_i]$
	$[ax \quad ay] \times 5t_2^4 + [bx \quad by] \times 4t_2^3 + [cx \quad cy] \times 3t_2^2 + [dx \quad dy] \times 2t_2 + [ex \quad ey]$	$k_4 \times dxy_i$
	$[ax \quad ay] \times 20t_2^3 + [bx \quad by] \times 12t_2^2 + [cx \quad cy] \times 6t_2 + [dx \quad dy]$	$[0 \quad 0]$

The first derivative border conditions are represented by $k_3 \times dxy_{i-1}$ and $k_4 \times dxy_i$ for respectively t_{i-1} and t_i, where dxy_{i-1} and dxy_i are the derivative direction calculated for the inside interpolation. The two scalar constants $k_3$and k_4 must be equal in order to ensure symmetry on the polynomial, as they are multiplied to the direction of the xy derivative.

$$k_3 = k_4 , \qquad 0 < \{k_3, k_4\} < 1 \qquad \text{(Eq. 3.25)}$$

As it was chosen to use a time normalization method, all the coefficients are calculated for the time interval $[0,1]$, as the process behind normalization is explained on the next subchapter.

The second derivative of the polynomials has again the null two dimensional vector as its border condition. As happened with outside interpolation, since the acceleration is null on the borders of the polynomials, it is guaranteed the variation will always be relatively low upon the entire path.

On the cases where it exists a set of two points that goes under an outside interpolation and the followed set of two points goes under an inside interpolation, the derivative direction will not be constant at the common point. However, this lack on derivative direction continuity does not affect the continuity of the polynomial itself, as it can be observed graphically on the next chapter concerning results analysis.

3.7 Time instants and time normalization

When studying trajectories, a piecewise polynomial is defined for an array of time instants comprehended between the lower and upper limit. Geometrically, each limit corresponds to the extreme point of the polynomial. The amount of time necessary to go through each polynomial is different on generic cases, and it is calculated from the constant linear velocity module and the length module of the polynomial itself. The problem resides in being necessary to know the polynomial length module in order to obtain the polynomial itself, as this is an iterative process.

In order to simplify the problem by avoiding iteration processes, it is assumed a new parameter $\boldsymbol{TLP}$ (True Length Percentage) containing the percentage value that compares the length of the polynomial between two points with the linear segment length ($\boldsymbol{LSL}$) between the two points themselves, represented by d_1 in Figure 3.16 (i and $i+1$, $i \in [1\,,n]$ and n represent the number of created polynomials). With the help of Figure 3.16 it is possible to imagine the theoretical basis of the chosen procedure. As the only available information at the time of calculating the polynomial coefficients is the length of the linear segment connecting to generic points i and $i+1$, $\boldsymbol{TLP}$ serves as a sort of approximation factor that compares the linear interpolation length with the future 5th degree polynomial line integral (Eq. 3.26).

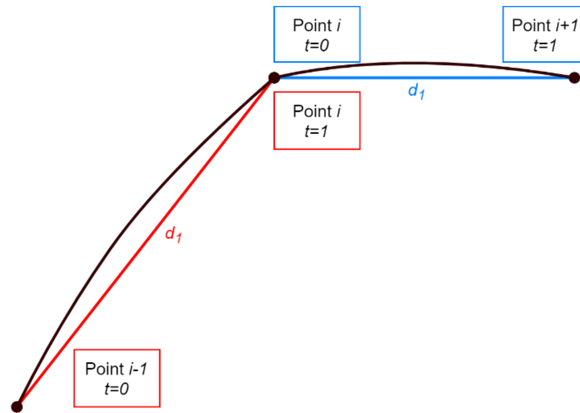

Figure 3.16 Generic time normalization for two different polynomials

$$LSL_{\,i\to i+1} = TLP \times Polynomial\ Length_{i\to i+1} \quad , \qquad TLP \in [0{,}1] \qquad \text{(Eq. 3.26)}$$

Where LSL is defined by the following expression where x and y represent the point coordinates used for the interpolation:

$$LSL_{\,i\to i+1} = \sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2} \qquad \text{(Eq. 3.27)}$$

The calculation of the time instant increments may then be calculated by creating a mathematical relationship (Eq. 3.28) between the following, where $i \in [1\,,n]$ and n represent the number of created polynomials:

- The previous time instant increment ($t_{\,i-1}$)

- Maximum extrusion speed, that is further explained on the following sub-chapter (ve).
- Linear distance between two followed points (LSL).
- True length percentage parameter (TLP).

$$t_i = t_{i-1} + \frac{LSL_i}{(ve \times TLP)} \quad \text{(Eq. 3.28)}$$

The time parameter used for defining polynomials goes under a normalization process (Eq.3.29) on the defined time interval. $\boldsymbol{t_{norm}}$ is created and it may be used to rightly define polynomials.

$$t_{norm} = \frac{t - t_{i-1}}{t_i - t_{i-1}} \quad \text{(Eq. 3.29)}$$

3.8 Extrusion speed and linear velocity

The extrusion speed is a sensible degree of freedom when defining the trajectory, as it is directly linked with the constant linear velocity along the polynomial. Since the deposition of material on each layer must be constant along the planar surface, the module of both velocities ought to be constant along the movement. If the nozzle moves faster than the material deposition rate, the surface layer will not be vertically uniform in the zones where deposition exists.

For simplicity purposes, it is assumed the extrusion speed has infinite acceleration. On that scenario, as the extrusion velocity module is constant, it is presumed the velocity goes from zero to the maximum value instantaneously.

For the reasons announced, the method proposes to calculate the polynomial maximum linear velocity and to guarantee it is never higher than the extruder maximum extrusion speed. This way, the polynomial velocity can be optimized while never exceeding the maximum extrusion speed. The control of the maximum linear velocity of the nozzle is made by controlling the length of the sharp edge reconstructed corners. When calculating the polynomial coefficients, the maximum velocity is then also determined using the method explained below, and this value is compared with the maximum extrusion speed. If it exceeds the referred quantity, the sharp edges of the model are once again redefined to accommodate for greater distance between newly created points, on the corners needing special attention.

$$maximum\ polynomial\ linear\ velocity\ \ module < maximum\ extrusion\ speed$$

If the maximum polynomial linear velocity module is represented by v_{max} and the maximum extrusion speed is represented es_{max}, the formula above is rearranged to the equation in Eq.3.30.

$$v_{max} < es_{max} \quad \text{(Eq. 3.30)}$$

Naturally, it is mandatory to calculate the maximum polynomial velocity so that the condition above can be verified. That velocity is calculated for every different inside and outside interpolated polynomial, though a new algorithm for which the following mathematical expressions apply. For

each polynomial, the calculation of the velocity is made by equalling the second derivative of the polynomial, corresponding to the instantaneously acceleration, to zero. The result obtained by solving the created equation are the time instants where the acceleration is null, this meaning it is either a maximum velocity corresponding time instant or the result of previously defined conditions, like for example the border conditions for each polynomial. These ensure that the acceleration at every extreme polynomial point is null, and for this reason these points must be discarded when calculating the polynomial maximum velocity, that is, the maximum of the polynomial second derivative.

It is also important to mention that the extrusion speed value to compare the results is only comparable with the module of the polynomial velocity, with both x and y components (Eq.3.31).

$$\|v(t)\| = \sqrt{\left(\frac{dp_x}{dt}\right)^2 + \left(\frac{dp_y}{dt}\right)^2}, \qquad t \in [t_{i-1}, t_i] \tag{Eq. 3.31}$$

Where $\|v(t)\|$ corresponds to a generic polynomial instantaneously velocity module (polynomial first derivative), p_x corresponds to the x component polynomial and p_y corresponds to the y component polynomial. The acceleration expression $a(t)$ (Eq. 3.32) naturally emerges by performing not the first but second derivative on each component polynomial within the specified time interval for the polynomial in question. As the current objective is to calculate the time instant correspondent to the maximum velocity module. $a(t)$ consist of a squared third degree expression, this meaning that when the expression is equalled to zero (Eq. 3.33), the result of the equation are six different time instants where the acceleration is null. Since $a(t)$ now corresponds to a six degree polynomial instead of a three degree, three of zeros do not actually exist, and they are denominated *fake zeros*.

$$a(t) = \frac{d^2p_x}{dt^2} + \frac{d^2p_y}{dt^2}, \qquad t \in [t_{i-1}, t_i] \tag{Eq. 3.32}$$

$$a(t) = 0, \qquad t \in [t_{i-1}, t_i] \tag{Eq. 3.33}$$

Since the data used for the creation of polynomials is originated from the process chain bases on the *G-Code* file for each geometry, certain inconsistencies were found when trying to calculate the maximum velocities. These inconsistencies were outlined in the form of sequences of *G-Code* points shortly followed by each other when comparing to the generic commonly found distance between two *G-Code* points. The result of the existence of these inconsistencies reflects on the calculation of the maximum linear velocity module, as this value tends to increase drastically if the segment length is drastically shorter than average. In order to prevent the drastic variation of values that do not correspond to real life faced scenarios, a velocity limit (v_L) parameter is created and related with the maximum extrusion speed ($esmax$) (Eq. 3.34). This value is correlated with the maximum extrusion speed, as linear velocities can never exceed this value. Even though every value beyond the maximum extrusion speed is not to be accepted, the velocity limit accepts the value for comparison. Once the user realizes this value exceeds the maximum extrusion speed, the parameters involved on the linear velocity calculation must be adjusted, in order to guarantee a valid result.

$$v_L = 2.5 \times es_{max} \quad \text{(Eq. 3.34)}$$

As the time instant correspondent to the maximum velocity module is calculated, the algorithm proceeds to calculate the maximum velocity module, as this value can now be compared with the maximum extrusion velocity.

Graphically, the visualization of the maximum polynomial module velocity can be made, and the value can be approximately calculated.

3.9 Maximum linear velocity module optimiziation

As it is inherent to any subject related to additive manufacturing, the time that is necessary to fully construct a certain geometry is vital. If a certain FFF machine extruder is built to withstand a high extrusion speed limit, and if all the other printing conditions are assured, the operator of the machine naturally wants to exploit the device to its maximum capabilities, and thereby reducing the process time. As explained before, the extrusion speed must be equal to the linear velocity module of the nozzle, and for that reason, if extrusion speeds are to be maximized, linear velocity must follow that tendency.

A maximum linear velocity module optimization algorithm is proposed, which relies on the manipulation of the TLP parameter defined in Eq. 3.26. As the variation of TLP affects the length of the future polynomial and thereby affects the maximum velocity this polynomial acquires, the maximization of TLP while not exceeding the maximum extrusion speed (es_{max}) thus leads to the optimization of v_{max}.

The objective function is then defined as the minimization of the squared average difference between es_{max} and v_{max} (Eq. 3.35). To ensure the stability of the process, and thereby ensuring the nozzle is not moving at the theoretical maximum linear velocity module, a *safety factor* of 90% is multiplied to the es_{max}, reducing the chances of v_{max} exceeding the *safe* limit velocity module.

$$obj\ (TLP) = \left(\frac{0.9 \times es_{max} - v_{max}(TLP)}{2}\right)^2, \qquad t \in [t_{i-1}, t_i] \quad \text{(Eq. 3.35)}$$

After the optimal TLP is calculated, the algorithm responsible for creating the polynomial coefficients must be executed once more to obtain the optimal polynomial definition. The new v_{max} value is also once more calculated with the optimal TLP parameter.

4 Analysis and Discussion

4.1 Introduction

This chapters gathers the results of the tests conducted with the newly created algorithms for the several steps described previously, having its final goal as the curve description.

The purpose of developing these case studies is to be able to clearly observe how the chosen parameters affect each determined section geometry and compare results. Additionally, the variation of these same parameters naturally produces different results. The optimization of the polynomials and its parameters is then responsible for creating the most similar results to the original 3D geometry, bearing in mind the conditions that were set during the present report.

In this chapter, two case studies are used to compare different results: a **cylinder case study** and a **cube case study**. The two geometries are considered pertinent to exemplify though since, although relatively simple, they imply different concerns due to severe differences in shape outline. The cube original perimeter composed by straight lines with 90º angles, as the cylinder original perimeter is defined though a circle. Although these parts suffer a decomposition of their segments on the slicing software (converting curved lines on small linear segments), the differences between the two are still noticeable and deserve distinction, as these examples may serve to predict how the algorithm will perform using future models.

On the following explanations, the two different geometries represent the extreme scenarios of the cases the algorithm is to be applied on. Few sharp edges exist on the cylinder *G-Code*, as it is now analysed. By the other hand, the cube geometry is built by several sharp corners connecting long straight lines on its outline. Some experts on FFF even proclaim certain geometries like the cube or similar type bodies are not ideally manufactured through this process, as sharp edges pose a problem on defining corners correctly, and the approximations can cause considerable deviations from the projected part.

The algorithms created to achieve the results proposed by the methodology introduced follow the methodological program sequence in Figure 4.1. This sequence must be strictly followed to obtain the right sequence of outputs and inputs that allow the user to reach the desired results. Each of the routines is available as an appendix to this report, and the main results are now introduced, and the main singularities present are highlighted.

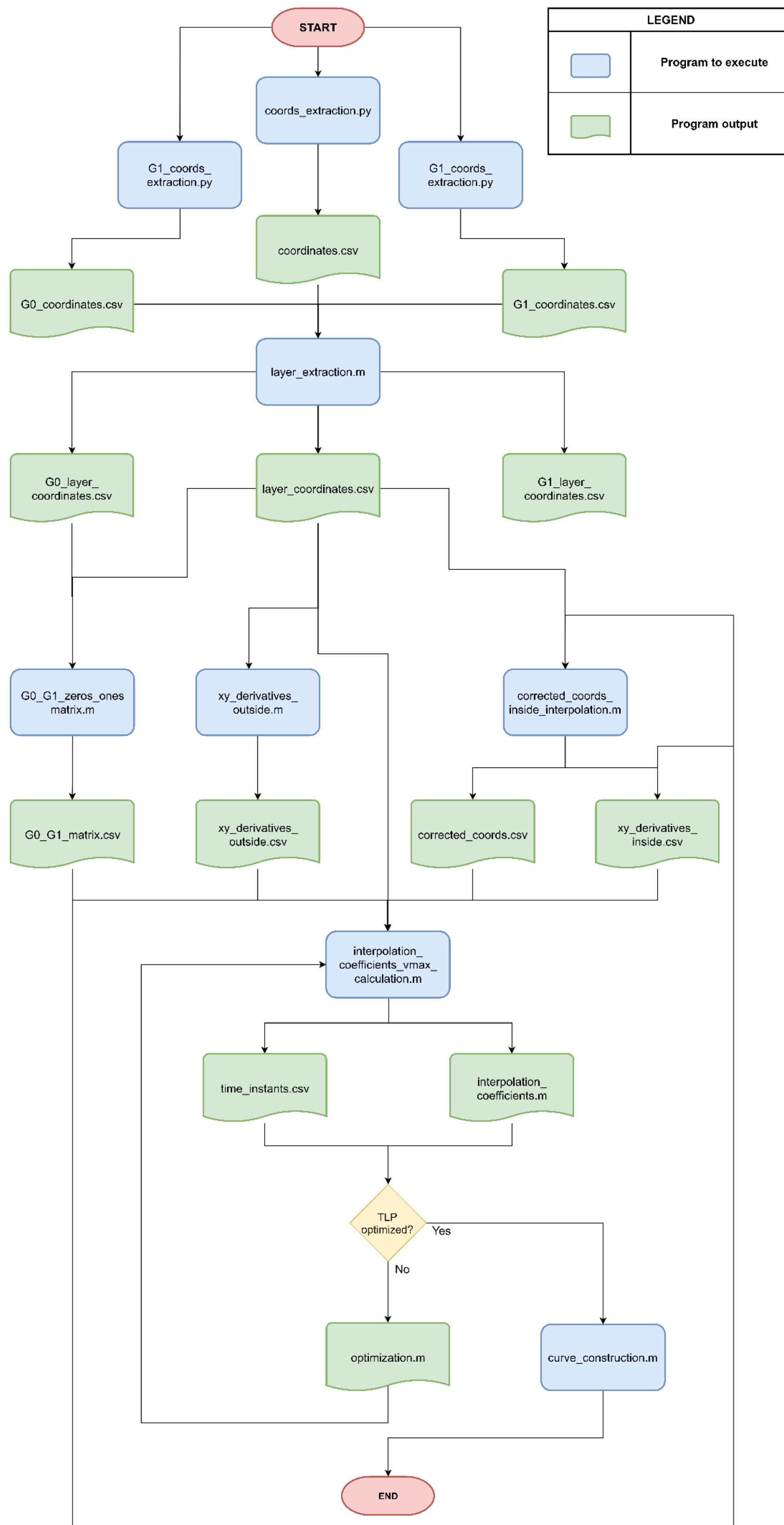

Figure 4.1 Methodological programe sequence

4.2 Case Study Test Results: Cylinder

The case study processes the event chain described on the methodology, and therefore the starting point is an $.STL$ file with the cylinder body. As explained above, the geometry is then sliced using the *Cura* software. The used pre-set slicer is originally from the *Ultimaker S5* along with the complementary default printing parameters in *Cura*. The process is carried out by obtaining the coordinates for every point and the linear interpolation of the results after the layer coordinates are extracted from the entire set of coordinates is visible in Figure 4.2. By taking closer attention at the linear interpolation, it is possible to identify sharp edges and the overall desired shape of the cylinder layer, even if it is only represented through 1st degree polynomials and not 5th degree ones.

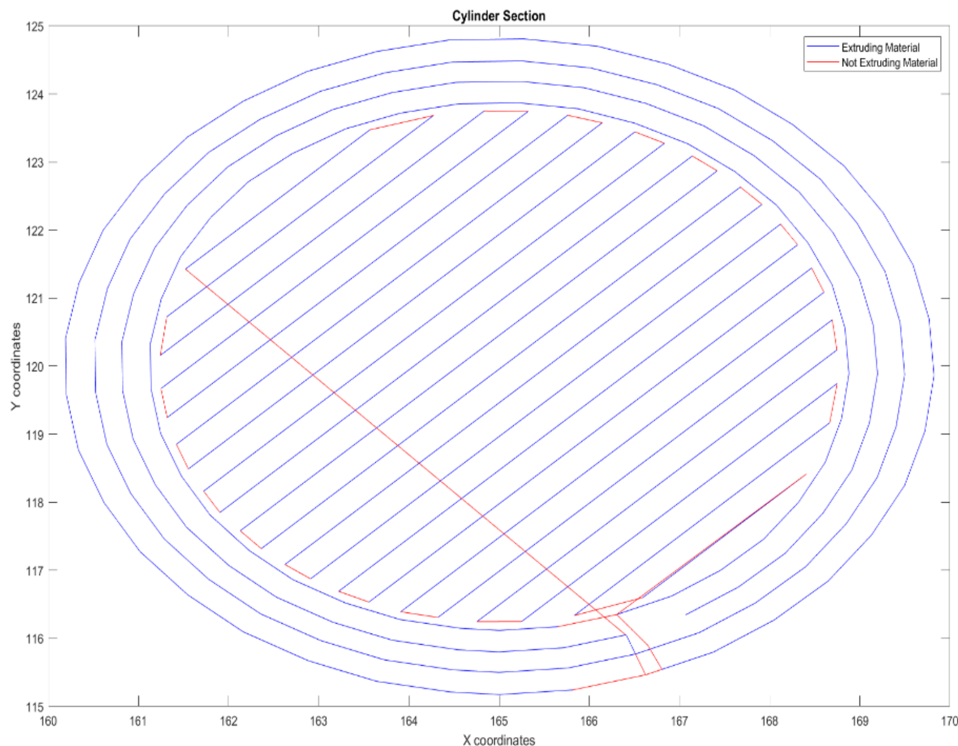

Figure 4.2 Linear interpolation of the results from the algorithm in "layer_extraction.m" on the cylinder case study

After extracting coordinates, the focus turns to three different tasks:

1. "*G0_G1_zeros_ones_matrix.m*". Creating a matrix where *G0* and *G1* commands are distinguished for every coordinate in the form of zeros and ones: if the move command corresponds to a *G1* command, then the matrix attributes the value 1. If not, it represents a G0 commands and the algorithm attributes the value 0.
2. "*xy_derivatives_outside.m*". Derivative directions are calculated using the method explained on the methodology and exported for later use on the calculation of the polynomial coefficients.

3. “*corrected_coords_inside_interp.m*”. The layer coordinates are assessed based on the existence of extrusion, and if so, on the existence of sharp edges. On the cases where there is indeed the existence of sharp edges, the described method for these situations is processed and the new points replace the previous coordinate causing a sharp edge. As these cases are to go under an inside interpolation process, the inside interpolation derivatives directions are calculated and exported in order to later use on the calculation of polynomial coefficients.

For the awareness of the present case study, the parameters used by the set of algorithms are presented in Table 4.1.

Table 4.1 Defined parameters for the cylinder case study

PARAMETER	VALUE
es_{max} [mm/s]	20
$angle_{in/out}$ [°]	100
Sharp edge angle [°]	100
a	0.1
b	0.5
c	0.1
k_1	0.4
k_2	.4
k_3	0.15
k_4	0.15
TLP (before optimization)	0.4
v_L [mm/s]	2.5 x 20 =50

The previously calculated data along with the set of parameters are now used by the “*interpolation_coefficients_vmax_calculation.m*” program. From here, various data is obtained:

- a list of polynomial coefficients is extracted.
- a list of time intervals correspondent to the domain for each polynomial is extracted. This is firstly calculated using the non-optimized TLP parameter.
- The maximum linear polynomial velocity (v_{max}) is calculated, also using the non-optimized TLP parameter.

With the set of parameters used, the result obtained for v_{max} while using the non-optimized TLP parameter of 0.4 is made (Eq.4.1).

$$before\ optimization: v_{max} = 15.0308\ \mathrm{mm/s} \qquad \text{(Eq. 4.1)}$$

This result is also visible in Figure 4.3, where the polynomial velocity module variation is represented along the time, before the optimization process takes place. It is important to note that for the purpose of this experiment, the theoretical speed is constant and always equal to the maximum values represented Figure 4.3 for each polynomial, as acceleration is said to be infinite. Thus, the real velocity variation for each polynomial is actually represented by a horizontal line with the maximum velocity for each polynomial. However, Figure 4.3 describes the peak values for each polynomial, as each one is defined on one different consecutive colour.

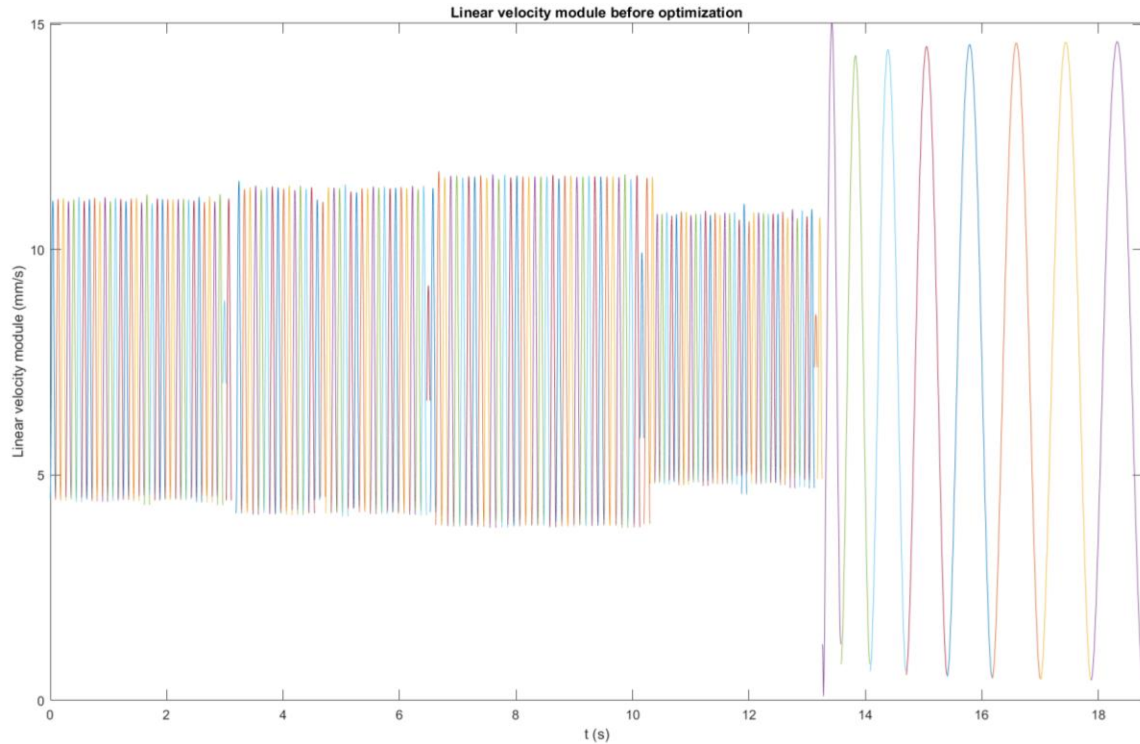

Figure 4.3 Linear polynomial velocity module variation before optimization

Continuing following the process chain in Figure 4.1, the focus now turned to optimizing the possible maximum velocity. As explained before, the optimization process minimizes the difference between v_{max} and the maximum extrusion speed when multiplied with the 90% safety factor ($v_{\max objective}$) (Eq.4.2).

$$v_{\max objective} = v_e \times 0.9 = 20 \times 0.9 = 18\ \mathrm{mm/s} \qquad \text{(Eq. 4.2)}$$

After *optimization.m* is executed, the program then calculates the optimized TLP parameter based on the applicable objective function (Eq. 4.3).

$$\mathrm{TLP} = 0.4790 \qquad \text{(Eq. 4.3)}$$

The new value can now be used to re-calculate the polynomial coefficients along with the final maximum linear velocity module. The v_{max} determination is graphically confirmed when representing the linear velocity module variation when using the optimized TLP value as the replacement parameter (Figure 4.4). The algorithm also calculates the new v_{max} mathematically as it did before optimization, obtaining the new result (Eq. 4.4).

$$v_{\max optimized} = 17.994 \text{ mm/s} \qquad \text{(Eq. 4.1)}$$

Even though this result is not precisely equal to $v_{\max objective}$, the error associated with its calculation is approximately 0.03%, which makes the result valid.

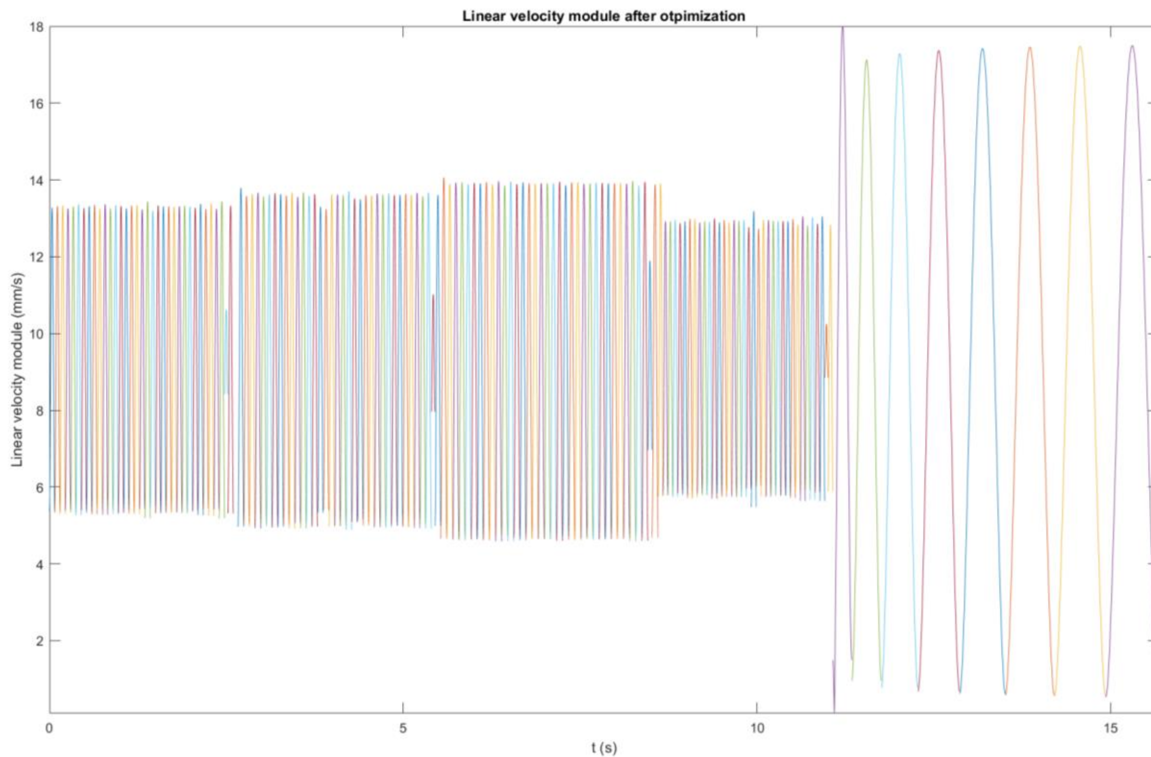

Figure 4.4 Linear polynomial velocity module variation after optimization

The polynomial coefficients also go under re-calculation, and the resulting polynomials built from the calculations within each polynomial domain are graphically represented in Figure 4.5. It is noticeable the set of contours are apparently missing one final polynomial closing the perimeter of the outer cylinder layers, however the results obtained are presented. The inconsistency is likely due to errors on the original *G-Code* file distinction between G0 and G1 commands and are to be corrected when considering future work.

Due to the geometry of the cylinder, sharp edges are not commonly found on the definition of the outer layers, however Figure 4.6 exemplifies an example where a sharp edge was detected and corrected according to the defined method. It can be noted the creation of two new points on what was previously a sharp edge, and the inside interpolation process that is used to define the corner.

The description of the rotation of the extrusion head along the movement is given by the files concerning either outside and inside derivatives on the correspondent points where there is either outside or inside interpolation.

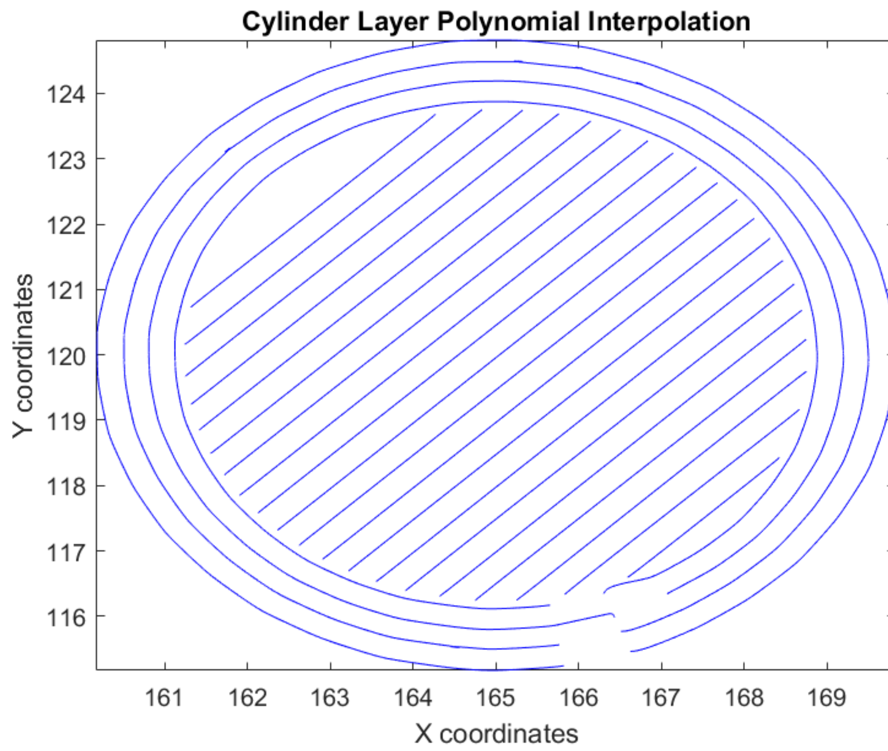

Figure 4.5 Polynomial interpolation representation for the cylinder layer

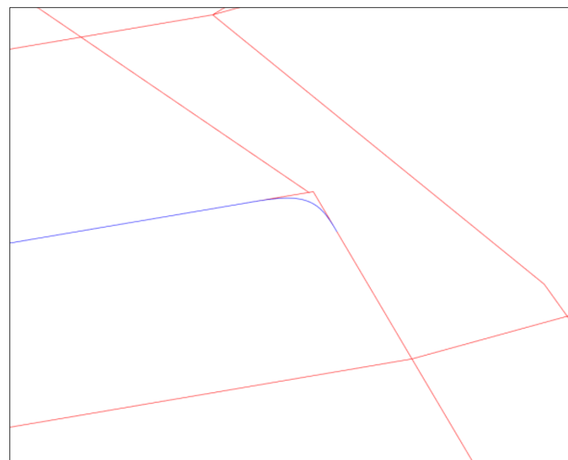

Figure 4.6 Polynomial interpolation sharp edge detail for the cylinder layer

4.3 Case Study Test Results: Cube

The second case study that is found relevant to analyse is the cube geometry and its layer section. The cube printing settings are the same as the cylinder, as is the FFF device. The cube itself had 10 mm edges and was represented in Chapter 3 (Figure 3.1). The part was centred on the FFF device, and that is the reason for the coordinate values presented. As said before, each layer presents several sharp edges that are determinant for the definition of the outside contours of the future 3D object. For that reason, some parameters must be adjusted to allow for a good corner definition.

The linear interpolation of a generic cube layer is presented in Figure 4.7.

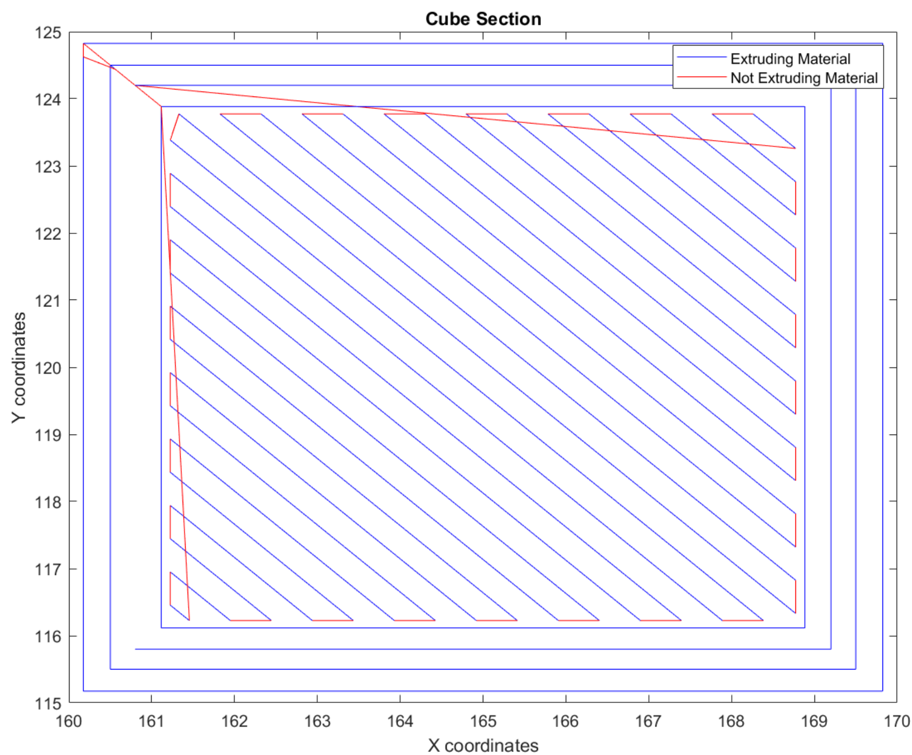

Figure 4.7 Linear interpolation of the results from the algorithm in "layer_extraction.m" on the cylinder case study

When defining the parameters for the cube case study (Table 4.2), most of the ones applied to the cylinder case study endured to this example. The a and c value were newly defined: these values will essentially represent the limits of the sharp corners inside interpolation, and must be minimized in order to approximate the reproduced geometry to the original model, while eliminating the sharp edge itself.

For the cube case study, although the methodology used is equal, the algorithm used previously on the cylinder case study was redesigned and the adapted to fit the geometry requirements due to the increased amount of sharp edges. The new versions concern the alterations stated bellow:

- The outside xy derivative directions are now calculated along the inside xy derivative direction on the "*corrected_coords_inside_interp.m*" (Appendix J) program. For that reason, "outside_derivatives.m" is no longer used for calculation of the outside derivatives.
- *"interpolation_coefficients_vmax_calculation.m"* (Appendix K) was adapted to use the methodology with a different syntax within the program.

Table 4.2 Defined parameters for the cube case study

PARAMETER	VALUE
$\boldsymbol{es_{max}}$ $[\mathbf{mm/s}]$	20
$\boldsymbol{angle_{in/out}}$ [°]	100
Sharp edge angle [°]	100
$\boldsymbol{a}$	0.01
$\boldsymbol{b}$	0.5
$\boldsymbol{c}$	0.01
$\boldsymbol{k_1}$	0.4
$\boldsymbol{k_2}$	0.4
$\boldsymbol{k_3}$	0.15
$\boldsymbol{k_4}$	0.15
TLP (before optimization)	0.4
$\boldsymbol{v_L}$ $[\mathbf{mm/s}]$	2.5 x 20 = 50

The interpolation is made and, as before, the algorithm calculates the velocity for the predefined TLP value before the optimization process takes place. This result for the used TLP of 0.4 is represented in Eq. 4.5. The graphical confirmation for the result returned by the algorithm is displayed in Figure 4.8.

$$before\ optimization: v_{max} = 15.0004\ \mathrm{mm/s} \quad \text{(Eq. 4.5)}$$

Subsequently, the "*optimization.m*" (Appendix L) program is executed, and the result is the TLP value present in Eq. 4.6. Please note this function was naturally updated to accommodate the new interpolation and maximum velocity module calculation program. It is also important to refer the same safety factor of 90% was once again used to obtain the optimized values.

$$\mathrm{TLP} = 0.4800 \quad \text{(Eq. 4.6)}$$

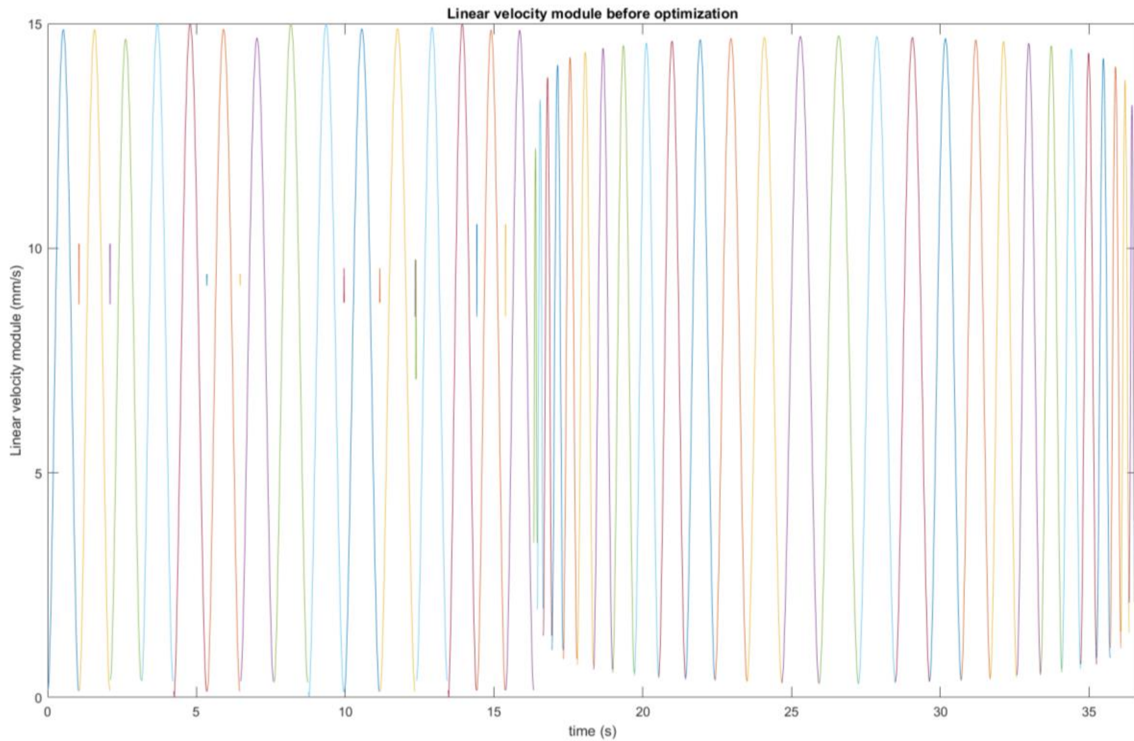

Figure 4.8 Cube linear polynomial velocity module variation before optimization

The polynomial interpolation is once again performed to re-calculate the final polynomials coefficients along with the maximum velocity (Eq. 4.7). Graphically, the polynomial interpolation result may be observed in Figure 4.9.

$$after\ optimization{:}\ v_{max} = 18.0005\ \mathrm{mm}/s \quad \text{(Eq. 4.7)}$$

The resulting polynomial interpolation for a generic layer is displayed in Figure 4.10. The corners of the square contours defining the shape were previously represented by sharp edges and are now represented by 5th degree polynomials withing the correspondent function domain.

Once again, irregularities were found concerning the correct correspondence between coordinates and $G0/G1$ commands. That is then the reason found to fundament the irregularities presented when observing the left wall outside contours of the cube layer. Exceptions were made for the infill of the layer, as it is known the pattern will follow straight lines, meaning there is an interlayer of $G0$ and $G1$ commands.

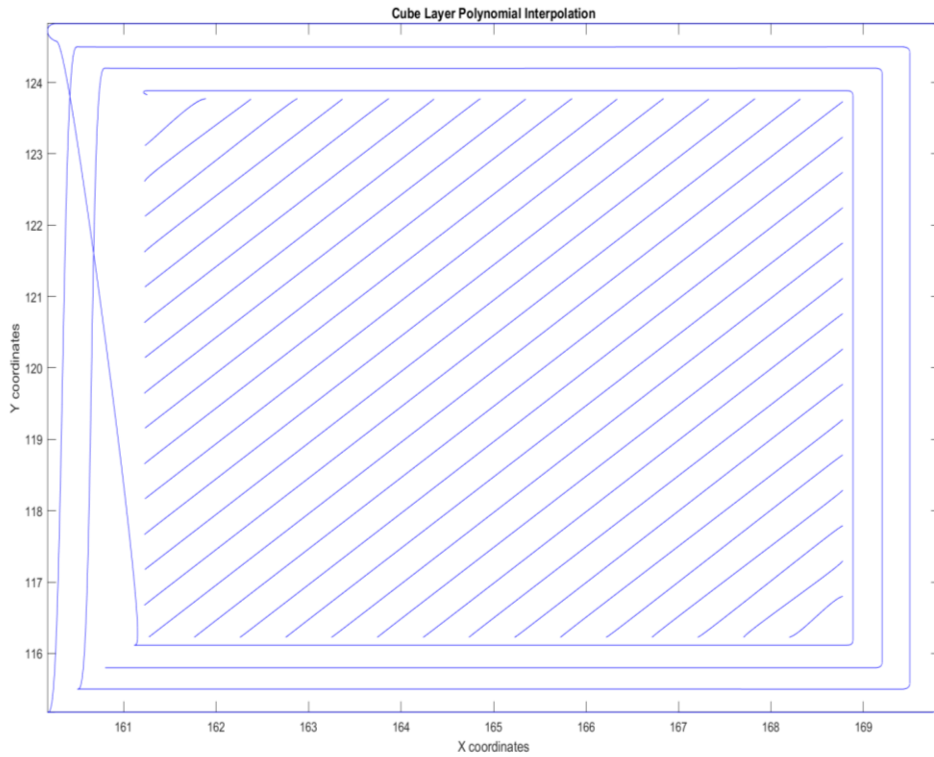

Figure 4.9 Polynomial interpolation representation for the cube layer

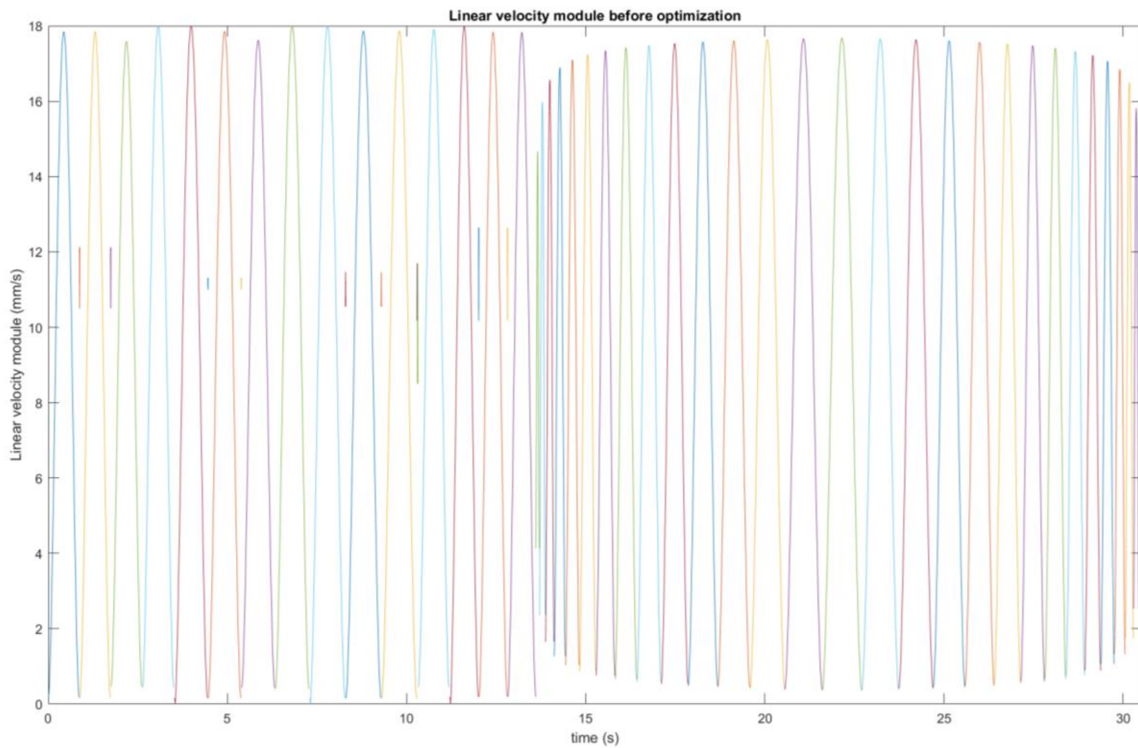

Figure 4.10 Cube linear polynomial velocity module variation after optimization

4.4 Concluding Remarks

The two case studies display similarities between them on aspects regarding the handling of sharp edges and generation of polynomials with measurement of the velocity module. As expected, different geometries emphasize different characteristics. The cylinder case study displays outside interpolation associated with curves with closer details while the cube case study highlights how straight lines are defined while 5th degree polynomials. By the other hand, the cube case study gives greater attention to the handling of sharp edges and later inside interpolation, along with the linkage between inside and outside interpolation sections.

Regarding maximum speed module calculation and optimization, the results are satisfactory on both case studies are they are maximized according to the maximum extrusion speed while accounting for the set safety factor.

5 Conclusions and Future Work

5.1 Conclusions and contributions

A new methodology, based on the literature review performed for this work, was developed, explained, and substantiated with two case studies.

The objectives undertaken that lead to the creation of the present dissertation were overall successfully achieved with the proposed methodology, however with certain irregularities that are now highlighted. The applicability of the method has shown to be successful on the cases where there is continuous extrusion that is uninterrupted by $G0$ commands withing the original *G-Code* files. For the cases where these conditions cannot be verified, inconsistencies were found concerning correspondence between what is or not extruded. Although the algorithm is programmed to identify if a section defined by its final point is correspondent to whether or not there is extrusion, by confirming if the coordinate is preceded by a $G0$ or $G1$ command, the encountered irregularities are still noticeable.

Sharp edges that were firstly identified on generic geometries layers are successfully replaced with the sets of either two or three new points and identified as piecewise polynomials requiring an inside interpolation, as it can be seen on the two case studies presented. The calculation of the derivative directions along with the rearranged set of data points and the information concerning the existence or inexistence of extrusion allows the interpolation to be made through inside and outside interpolation processes. As seen on the methodology and proved by the analysis of the case studies, the main differentiating aspect between the two kinds of interpolation is the direction of the derivatives that is used for the definition of the polynomials.

The calculation of the maximum velocity and further optimization of the TLP parameter method was validated by the case studies with unsignificant errors after optimization, meaning the safety factor imposed can also be adjusted and even removed if exceptions so demand.

Both case studies of the cylinder and cube reveal that outside and inside interpolation methods are applied correctly and this result can be graphically visualized. The coefficients exported by the polynomial generation along with the instantaneous velocity module calculation are the main product resulted from this research to be used further ahead on the “Optimal Non-Planar Trajectory Generation for Additive Manufacturing” investigation. The execution of the programs appended to the document grants any user the obtention of the desired results.

5.2 Suggestions for Future Work

The future work concerning the present thematic is essentially represented by the continuous development of the “Optimal Non-Planar Trajectory Generation for Additive Manufacturing” project

this investigation was conducted for. With the calculations of the trajectories within a slice of a generic object made, the next step of the project can be made towards completion.

Regarding the thematic discussed on the present report, future work can be represented by the adding of the z coordinate to the defined trajectories. Besides this, the control of the material deposition rate can also be made to adjust the TCP velocity so that the extrusion is not constricted to constant material deposition (constant extrusion speed). Inevitably, issues concerning the applicability of the proposed method to generic geometries can also be improved so that parameters that need definition are allowed to decrease and, consequently, the automation of the process increases as less inputs are required.

References

[1] I. Gibson, D. W. Rosen, and B. Stucker, *Additive manufacturing technologies: Rapid prototyping to direct digital manufacturing*. Springer US, 2010.

[2] J. Bromberger and R. Kelly, “Additive manufacturing: A long-term game changer for manufacturers | McKinsey.” https://www.mckinsey.com/business-functions/operations/our-insights/additive-manufacturing-a-long-term-game-changer-for-manufacturers (accessed Jan. 16, 2020).

[3] “G-code - RepRap.” https://reprap.org/wiki/G-code (accessed Nov. 03, 2020).

[4] P. Corke, *Robotics, Vision and Control*, vol. 73. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[5] G. Onwubolu, *Mechatronics*. Elsevier Ltd, 2005.

[6] S. Cubero, *Industrial Robotics: Theory, Modelling and Control*. Pro Literatur Verlag, Germany / ARS, Austria, 2012.

[7] K. M. Lynch, *Modern Robotics. Mechanics, Planning, and Control*, no. May. 2017.

[8] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, “Robotics: Modelling, planning and control,” in *Advanced Textbooks in Control and Signal Processing*, no. 9781846286414, 2009.

[9] M. Cakir and E. Butun, “An educational tool for 6-DoF industrial robots with quaternion algebra,” *Comput. Appl. Eng. Educ.*, vol. 15, no. 2, 2007, doi: 10.1002/cae.20104.

[10] A. V. Shembekar, Y. J. Yoon, A. Kanyuck, and S. K. Gupta, “Generating robot trajectories for conformal three-dimensional printing using nonplanar layers,” *J. Comput. Inf. Sci. Eng.*, vol. 19, no. 3, Sep. 2019, doi: 10.1115/1.4043013.

[11] J. R. and C. de Boor, “A Practical Guide to Splines.,” *Math. Comput.*, vol. 34, no. 149, p. 325, Jan. 1980, doi: 10.2307/2006241.

[12] C. Sprunk, “Planning motion trajectories for mobile robots using splines,” *Univ. Freibg.*, no. October, 2008.

[13] A. Oliveira, “Apontamentos de Cálculo Numérico D - Chapter 2.” FCT-UNL, Lisbon, p. 90, 2019.

Appendix A

File: coords_extraction.py

```
import re
import csv
fp = open("cylinder.txt","r")
regex = re.compile('^[G][0-1]')
regex2 = re.compile('^;LAYER:')
line= fp.readline()
with open('coordinates.csv', 'w', newline='') as csvfile:
    fieldnames=['X','Y','Z']
    thewriter = csv.DictWriter(csvfile, fieldnames=fieldnames)
    z = 0;
    x= y = 0;
    while line:
        line= fp.readline()
        if regex2.match(line):
            thickness=0.2
            elements2 = line.split()
            for element in elements2:
                if regex2.match(element):
                    z= thickness + (int(element[7:])*thickness)
        if regex.match(line):
            elements = line.split()
            x_regex = re.compile('^X')
            y_regex = re.compile('^Y')
            has_x_or_y = False;
            for element in elements:
                if x_regex.match(element):
                    x=element[1:]
                    has_x_or_y = True;
                if y_regex.match(element):
                    y=element[1:]
                    has_x_or_y = True;
            if(has_x_or_y):
                    thewriter.writerow({'X':x, 'Y':y, 'Z': z})
```

Appendix B

File: G0_coords_extraction.py

```
import re
import csv
fp = open("cylinder.txt","r")
regex = re.compile('^[G][0]')
regex2 = re.compile('^;LAYER:')
line= fp.readline()
with open('G0_coordinates.csv', 'w', newline='') as csvfile:
    fieldnames=['X','Y','Z']
    thewriter = csv.DictWriter(csvfile, fieldnames=fieldnames)
    z = 0;
    x= y = 0;
    while line:
        line= fp.readline()
        if regex2.match(line):
            thickness=0.2
            elements2 = line.split()
            for element in elements2:
                if regex2.match(element):
                    z= thickness + (int(element[7:])*thickness)
        if regex.match(line):
            elements = line.split()
            x_regex = re.compile('^X')
            y_regex = re.compile('^Y')
            has_x_or_y = False;
            for element in elements:
                if x_regex.match(element):
                    x=element[1:]
                    has_x_or_y = True;
                if y_regex.match(element):
                    y=element[1:]
                    has_x_or_y = True;
            if(has_x_or_y):
                 thewriter.writerow({'X':x, 'Y':y, 'Z': z})
```

Appendix C

File: layer_extraction.m

Import data

Please comment/uncomment the chosen line for each scenario results

```
data=csvread('coordinates.csv');
% data=csvread('G0_coordinates.csv');
% data=csvread('G1_coordinates.csv');
```

Create array for the chosen layer

```
layer=[];
```

Separation Routine

```
for i=1:length(data)
    if data(i,3)==1
        layer=[layer; data(i,1),data(i,2)];
    end
end
```

Output

Please uncomment the chosen line for each scenario results

```
writematrix(layer,'layer_coordinates.csv')
% writematrix(layer,'G0_layer_coordinates.csv')
% writematrix(layer,'G1_layer_coordinates.csv')
```

Appendix D

File: G0_G1_zeros_ones_matrix.m

Import data

```
clc
clear all
dataG0=csvread('G0_layer_coordinates.csv');
data=csvread('layer_coordinates.csv');
x=data(:,1);
y=data(:,2);
xG0=dataG0(:,1);
yG0=dataG0(:,2);
```

Create array for the matrix

```
M=ones(length(x),1);
```

Matrix definition Routine

```
for p=1:length(x)
    for pi=1:length(xG0)
        if xG0(pi)==x(p)
            if yG0(pi)==y(p)
                M(p)=0;
            end
        end
    end
end
```

Output

```
writematrix(M,'G0_G1_matrix.csv')
```

Appendix E

File: xy_derivatives_outside.m

Input Reading

```
clc,clear
close all;
data=csvread('layer_coordinates.csv');
x=data(:,1).';
y=data(:,2).';
```

Creation of the derivative storage array

```
dxy=zeros(length(x),2);
```

Derivative Calculation Routine

```
for i=1:length(x)
    if i==1
        p0x=x(i);
        p0y=y(i);
        p1x=x(i+1);
        p1y=y(i+1);
        v2 = [p1x,p1y] - [p0x,p0y];
        v=v2/norm(v2);
        dxy(i, :) = v;
    elseif i==length(x)
        p0x=x(length(x)-1);
        p0y=y(length(x)-1);
        p1x=x(length(x));
        p1y=y(length(x));
        v1 = [p1x,p1y] - [p0x,p0y];
        v=v1/norm(v1);
        dxy(i, :) = v;
    else
        p0x=x(i-1);
        p0y=y(i-1);
        p1x=x(i);
        p1y=y(i);
        p2x=x(i+1);
        p2y=y(i+1);
        v1 = [p1x,p1y] - [p0x,p0y];
        v2 = [p2x,p2y] - [p1x,p1y];
        v=(v1+v2)/norm(v1+v2);
        dxy(i, :) = v;
    end
end
```

Export Data

```
writematrix(dxy,'xy_derivatives_outside.csv')
```

Appendix F

File: corrected_coords_inside_interp.m

IMPORT DATA

```
clc,clear
close all;
data=csvread('layer_coordinates.csv');
x=data(:,1).';
y=data(:,2).';
%EXTRUSION ON/OFF
G0_G1_data=csvread('G0_G1_matrix.csv');
%
%ORIGINAL COORDINATES
original_x=data(:,1).';
original_y=data(:,2).';
%
```

PARAMETERS

```
a=0.1;
b=0.5;
c=0.1;
length_percentage=0.1;
in_out_angle=100; %INSIDE/OUTSIDE INTERPOLATION ANGLE LIMIT
sharp_edge_angle=100;
```

FUTURE STORAGE

```
%NEW COORDINATES LIST
xn=[];
yn=[];
%
%DERIVATIVES
dxy=[];
%
iterations_to_skip=0;
num_new_points=0;
%
```

MAIN ROUTINE

```
for i=1:length(x)-2
    if iterations_to_skip > 0
        iterations_to_skip = iterations_to_skip - 1;
        continue
    else
        if i==1
            xn=[x(1)];
            yn=[y(1)];
            p1x=x(i);
```

```
            p1y=y(i);
            p2x=x(i+1);
            p2y=y(i+1);
            v2 = [p2x,p2y] - [p1x,p1y];
            v=v2/norm(v2);
            dxy= [dxy(1:end);v];
            G0_G1_update=[G0_G1_data(1)];
            continue
        elseif G0_G1_data(i-1)==1
            if G0_G1_data(i)==1
                %G1 FOLLOWED BY G1
                p0x=x(i-1);
                p0y=y(i-1);
                p1x=x(i);
                p1y=y(i);
                p2x=x(i+1);
                p2y=y(i+1);
                pax=x(i+2);
                pay=y(i+2);
                v1 = ([p1x,p1y] - [p0x,p0y]);
                v2 = ([p2x,p2y] - [p1x,p1y]);
                CosTheta = max(min(dot(v1,v2)/(norm(v1)*norm(v2)),1),-1);
                angle = 180-abs(real(acosd(CosTheta)));
                if abs(angle)<in_out_angle
                    %INSIDE INTERPOLATION
                    %Length of line before measurement
                    length_lb= sqrt((p1x-p0x)^2+(p1y-p0y)^2);
                    %Length of present line measurement
                    length_la= sqrt((p2x-p1x)^2+(p2y-p1y)^2);
                    %Lenght of line after measurement
                    length_la2= sqrt((pax-p2x)^2+(pay-p2y)^2);
                    if length_la<=length_percentage*length_lb
                        a=(length_la*b)/length_lb;
                        pn1x=(p0x*a + p1x*(1-a));%NEW POINT 1
                        pn1y=(p0y*a + p1y*(1-a));
                        pn2x=(p1x*(1-b) + p2x*b);%NEW POINT 2
                        pn2y=(p1y*(1-b) + p2y*b);
                        %
                        %Vector after
                        va = ([pax,pay] - [p2x,p2y])/norm([pax,pay] - [p2x,p2y]);
                        CosTheta2 = max(min(dot(v2,va)/(norm(v2)*norm(va)),1),-1);
                        angle2 = real(acosd(CosTheta2));
                        if abs(angle2)<in_out_angle
                            %THIRD NEW POINT
                            %NEW COORDINATES
                            num_new_points=num_new_points+3;
                            c=(length_la*b)/length_la2;
                            pn3x=(p2x*(1-c) + pax*c);%NEW POINT 3
                            pn3y=(p2y*(1-c) + pay*c);
                            xn = [xn(1:end), pn1x , pn2x , pn3x];
                            yn = [yn(1:end), pn1y , pn2y , pn3y];
                            iterations_to_skip=1;
                            %DERIVATIVES FOR THE NEW POINTS
                            d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                            d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                            d3=([pn3x,pn3y] - [p2x,p2y])/norm([pn3x,pn3y] - [p2x,p2y]);
                            dxy= [dxy(1:end,:);d1;d2;d3];
                        else
```

```
                    %TWO NEW POINTS
                    %NEW COORDINATES
                    num_new_points=num_new_points+2;
                    xn = [xn(1:end) , pn1x , pn2x];
                    yn = [yn(1:end) , pn1y , pn2y];
                    %DERIVATIVES
                    d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                    d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                    dxy= [dxy(1:end,:);d1;d2];
                end
            else
                %TWO NEW POINTS
                %NEW COORDINATES
                c=(length_lb*a)/length_la;
                num_new_points=num_new_points+2;
                pn1x=(p0x*a + p1x*(1-a));%NEW POINT 1
                pn1y=(p0y*a + p1y*(1-a));
                pn2x=(p1x*(1-c) + p2x*c);%NEW POINT 2
                pn2y=(p1y*(1-c) + p2y*c);
                xn = [xn(1:end) , pn1x , pn2x];
                yn = [yn(1:end) , pn1y , pn2y];
                %DERIVATIVE
                d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                dxy= [dxy(1:end,:);d1;d2];
            end
        else
            %NO SHARP EDGE
            xn = [xn(1:end) , x(i)];
            yn = [yn(1:end) , y(i)];
            %DERIVATIVE
            v=v2/norm(v2);
            dxy= [dxy(1:end,:);v];
            plot(xn,yn,'r.-')
            drawnow
        end
    else
        %G1 FOLLOWED BY G1
        xn = [xn(1:end) , x(i)];
        yn = [yn(1:end) , y(i)];
        %DERIVATIVE
        v=v2/norm(v2);
        dxy= [dxy(1:end,:);v];
        plot(xn,yn,'r.-')
        drawnow
    end
elseif G0_G1_data(i-1)==0
    if G0_G1_data(i)==1
    %G0 FOLLOWED BY G1 (no sharp edge)
    xn = [xn(1:end) , x(i)];
    yn = [yn(1:end) , y(i)];
    %DERIVATIVE
    v=v2/norm(v2);
    dxy= [dxy(1:end,:);v];
    plot(xn,yn,'r.-')
    drawnow
    else
        continue
```

```
                end
            end
        end
    end
```

EXPORT OUTPUT

```
plot(xn,yn,'r.-')
M=[xn;yn].';
writematrix(M,'corrected_coords.csv')
writematrix(dxy,'xy_derivatives_inside.csv')
```

Appendix G

File: Interpolation_coefficients_vmax_calculation.m - CYLINDER CASE STUDY

IMPORT DATA

```
clc,clear
close all;
%
%CORRECTED COORDINATES FOR INSIDE INTERPOLATION
%
data=csvread('corrected_coords.csv');
x=data(:,1).';
y=data(:,2).';
%
%EXTRUSION ON/OFF
G0_G1_data=csvread('G0_G1_matrix.csv');
%
%ORIGINAL DATA
%
data_original=csvread('layer_coordinates.csv');
x_o=data_original(:,1).';
y_o=data_original(:,2).';
%
%STABLE DIFFERENCES BETWEEN ORIGINAL AND CORRECTED COORDINATES
%
x_diff=setdiff(x,x_o,'stable');
y_diff=setdiff(y,y_o,'stable');
%
%OUTSIDE DERIVATIVES
%
ddata=csvread('xy_derivatives_outside.csv');
xd_o=ddata(:,1).';
yd_o=ddata(:,2).';
%INSIDE DERIVATIVES
%
ddata=csvread('xy_derivatives_inside.csv');
xd=ddata(:,1).';
yd=ddata(:,2).';
%
```

PARAMETERS DEFINITION

```
in_out_angle=100;
sharp_edge_angle=100;
v_max_extrusion=20; %MAXIMUM EXTRUSION SPEED [mm/s]
true_length_percentage=0.4; %TLP PARAMETER
vL=50; %VELOCITY LIMIT PARAMETER
%
%K VALUES
K1=0.40;
K2=0.40;
K3=0.15;
```

```
K4=0.15;
%
```

POLYNOMIAL DEFINITION

```
syms t ax ay bx by cx cy dx dy ex ey fx fy
syms tt %Ture time variable
p = [ax ay]*t^5 + [bx by]*t^4 + [cx cy]*t^3 + [dx dy]*t^2 + [ex ey]*t + [fx fy];
dp = diff(p,t); %Polynomial 1st derivative
d2p = diff(p,t,2); %polynomial 2nd derivative
%
t=0;%First time instant
num_sharp_edges=0;
empty_vector=zeros(1,12);
%
tt=0;%True time count
length_seg=0;
```

FUTURE STORAGE

```
tt_list=[0];
length_list=[];
poly_coef=[];
v_max_total=[];
%
```

MAIN ROUTINE

```
i2=2;
for i=2:length(x_o)-1
    if G0_G1_data(i)==1
        p0x=x_o(i-1);
        p0y=y_o(i-1);
        p1x=x_o(i);
        p1y=y_o(i);
        p2x=x_o(i+1);
        p2y=y_o(i+1);
        v1 = [p1x,p1y] - [p0x,p0y];
        v2 = [p2x,p2y] - [p1x,p1y];
        CosTheta = max(min(dot(v1,v2)/(norm(v1)*norm(v2)),1),-1);
        angle = 180-abs(real(acosd(CosTheta)));
        if abs(angle)<in_out_angle
            if G0_G1_data(i-1)==1
                %INSIDE INTERPOLATION
                while i2<=length(x)-2
                    if x(i2)==x_diff(1)
                        if y(i2)==y_diff(1)
                            %Derivative directions
                            v=[xd(i2) yd(i2)];
                            vb=[xd(i2-1) yd(i2-1)];
                            %
                            t=0;
                            eq1=(subs(p) == [x(i2-1) y(i2-1)]);
                            eq2=(subs(dp) == K3*vb);
```

```
                        eq3=(subs(d2p) == [0 0]);
                        %
                        tt1=tt; %True time instant 1
                        length_seg= sqrt((x(i2)-x(i2-1))^2+(y(i2)-y(i2-1))^2);
                        tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
                        tt_list=[tt_list(1:end),tt];
                        length_list=[length_list(1:end),length_seg];
                        tt2=tt; %True time instant 2
                        %
                        t=1;
                        eq4=(subs(p) == [x(i2) y(i2)]);
                        eq5=(subs(dp) == K4*v);
                        eq6=(subs(d2p) == [0 0]);
                        %
                        sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                        %
                        axSol=sol.ax;
                        aySol=sol.ay;
                        bxSol=sol.bx;
                        bySol=sol.by;
                        cxSol=sol.cx;
                        cySol=sol.cy;
                        dxSol=sol.dx;
                        dySol=sol.dy;
                        exSol=sol.ex;
                        eySol=sol.ey;
                        fxSol=sol.fx;
                        fySol=sol.fy;
                        %
                        pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol
cySol]*tt^3 + [dxSol dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];

                        new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol
bySol cySol dySol eySol fySol]);
                        poly_coef=[poly_coef(1:end,:); new_coefs];
                        %
                        x_diff=[x_diff(2:end)]; %Remove different point from original
list
                        y_diff=[y_diff(2:end)];
                        i2=i2+1;
                        %CALCULATE MAXIMUM VELOCITY
                        syms time
                        tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                        %
                        x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                        y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                        vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                        atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                        max_t=vpasolve(atotal==0,tt1);
                        v_max_i=(subs(vtotal,max_t));
                        v_max_i_real=double(real(v_max_i(5)));
                        if v_max_i_real > vL
                            continue
                        end
                        v_max_total=[v_max_total(1:end);double(v_max_i_real)];
```

```
                        v_max=max(v_max_total);
                        %
                        if isempty(x_diff)
                            break
                        end
                    else
                        i2=i2+1;
                    end
                else
                    i2=i2+1;
                end
            end
        else
            %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
            %Derivative directions
            v=[xd_o(i) yd_o(i)];
            vb=[xd_o(i-1) yd_o(i-1)];
            %
            t=0;
            eq1=(subs(p) == [p0x p0y]);
            eq2=(subs(dp) == K1*vb);
            eq3=(subs(d2p) == [0 0]);
            %
            %LENGTH OF THE SEGMENT
            tt1=tt;%True time instant 1
            length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
            tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
            tt_list=[tt_list(1:end),tt];
            length_list=[length_list(1:end),length_seg];
            tt2=tt;%True time instant 2
            %
            t=1;
            eq4=(subs(p) == [p1x p1y]);
            eq5=(subs(dp) == K2*v);
            eq6=(subs(d2p) == [0 0]);
            %
            sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
            %
            axSol=sol.ax;
            aySol=sol.ay;
            bxSol=sol.bx;
            bySol=sol.by;
            cxSol=sol.cx;
            cySol=sol.cy;
            dxSol=sol.dx;
            dySol=sol.dy;
            exSol=sol.ex;
            eySol=sol.ey;
            fxSol=sol.fx;
            fySol=sol.fy;
            %
            pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
            %
            new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol
dySol eySol fySol]);
            poly_coef=[poly_coef(1:end,:); new_coefs];
```

```
                %CALCULATE MAXIMUM VELOCITY
                syms time
                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                %
                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                max_t=vpasolve(atotal==0,tt1);
                v_max_i=(subs(vtotal,max_t));
                v_max_i_real=double(real(v_max_i(5)));
                if v_max_i_real > vL
                    continue
                end
                v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                v_max=max(v_max_total);
                %
            end
        else
            %REGULAR OUTSIDE INTERPOLATION
            %Derivative directions
            v=[xd_o(i) yd_o(i)];
            vb=[xd_o(i-1) yd_o(i-1)];
            t=0;
            eq1=(subs(p) == [p0x p0y]);
            eq2=(subs(dp) == K1*vb);
            eq3=(subs(d2p) == [0 0]);
            %
            tt1=tt;%True time instant 1
            length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
            tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
            tt_list=[tt_list(1:end),tt];
            length_list=[length_list(1:end),length_seg];
            tt2=tt;%True time instant 2
            %
            t=1;
            eq4=(subs(p) == [p1x p1y]);
            eq5=(subs(dp) == K2*v);
            eq6=(subs(d2p) == [0 0]);
            %
            sol = solve([eq1,eq2,eq3,eq4,eq5,eq6], [ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
            %
            axSol=sol.ax;
            aySol=sol.ay;
            bxSol=sol.bx;
            bySol=sol.by;
            cxSol=sol.cx;
            cySol=sol.cy;
            dxSol=sol.dx;
            dySol=sol.dy;
            exSol=sol.ex;
            eySol=sol.ey;
            fxSol=sol.fx;
            fySol=sol.fy;
            %
            pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
```

```
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
            %
            new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol dySol
eySol fySol]);
            poly_coef=[poly_coef(1:end,:); new_coefs];
            %CALCULATE MAXIMUM VELOCITY
            syms time
            tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
            %
            x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 + dxSol*tt_sym.^2 +
exSol*tt_sym + fxSol;
            y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 + dySol*tt_sym.^2 +
eySol*tt_sym + fySol;
            vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
            atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
            max_t=vpasolve(atotal==0,tt1);
            v_max_i=(subs(vtotal,max_t));
            v_max_i_real=double(real(v_max_i(5)));
            if v_max_i_real > vL %Eliminate false positives
                continue
            end
            v_max_total=[v_max_total(1:end);double(v_max_i_real)];
            v_max=max(v_max_total);
            fplot(vtotal,[tt1,tt2]);
            hold on
        end
    end
end
```

EXPORT OUTPUTS

```
display(v_max)
writematrix(poly_coef,'interpolation_coefficients.csv')
writematrix(tt_list,'time_instants.csv')
writematrix(length_list,'length_list.csv')
```

Appendix H

File: optimization.m - CYLINDER CASE STUDY

INTRO

```
clc,clear
close all;
vemax=20;
%
```

INITIAL GUESSES

```
tlp_guess=0.4;
x0=[tlp_guess];
%
```

CALL THE SOLVER

```
opt_tlp=fmincon(@objective,x0,[],[],[],[],[],[],[],[])
vmax_opt=calcvmax(opt_tlp)
%
```

VMAX CALCULATION

```
function vmax_to_opt=calcvmax(tlp)
    %CORRECTED COORDINATES FOR INSIDE INTERPOLATION
    %
    data=csvread('corrected_coords.csv');
    x=data(:,1).';
    y=data(:,2).';
    %
    %EXTRUSION ON/OFF
    G0_G1_data=csvread('G0_G1_matrix.csv');
    %
    %ORIGINAL DATA
    %
    data_original=csvread('layer_coordinates.csv');
    x_o=data_original(:,1).';
    y_o=data_original(:,2).';
    %
    %STABLE DIFFERENCES BETWEEN ORIGINAL AND CORRECTED COORDINATES
    %
    x_diff=setdiff(x,x_o,'stable');
    y_diff=setdiff(y,y_o,'stable');
    %
    %OUTSIDE DERIVATIVES
    %
    ddata=csvread('xy_derivatives_outside.csv');
    xd_o=ddata(:,1).';
    yd_o=ddata(:,2).';
    %INSIDE DERIVATIVES
```

```
%
ddata=csvread('xy_derivatives_inside.csv');
xd=ddata(:,1).';
yd=ddata(:,2).';
%
% PARAMETERS DEFINITION
in_out_angle=100;
sharp_edge_angle=100;
v_max_extrusion=20; %MAXIMUM EXTRUSION SPEED [mm/s]
vL=50; %VELOCITY LIMIT PARAMETER
%
%K VALUES
K1=0.40;
K2=0.40;
K3=0.15;
K4=0.15;
%
% POLYNOMIAL DEFINITION
syms t ax ay bx by cx cy dx dy ex ey fx fy
syms tt %Ture time variable
p = [ax ay]*t^5 + [bx by]*t^4 + [cx cy]*t^3 + [dx dy]*t^2 + [ex ey]*t + [fx fy];
dp = diff(p,t); %Polynomial 1st derivative
d2p = diff(p,t,2); %polynomial 2nd derivative
%
t=0;%First time instant
num_sharp_edges=0;
empty_vector=zeros(1,12);
%
tt=0;%True time count
length_seg=0;
% FUTURE STORAGE
tt_list=[0];
length_list=[];
v_max_total=[];
%
% MAIN ROUTINE
i2=2;
for i=2:length(x_o)-1
    if G0_G1_data(i)==1
        p0x=x_o(i-1);
        p0y=y_o(i-1);
        p1x=x_o(i);
        p1y=y_o(i);
        p2x=x_o(i+1);
        p2y=y_o(i+1);
        v1 = [p1x,p1y] - [p0x,p0y];
        v2 = [p2x,p2y] - [p1x,p1y];
        CosTheta = max(min(dot(v1,v2)/(norm(v1)*norm(v2)),1),-1);
        angle = 180-abs(real(acosd(CosTheta)));
        if abs(angle)<in_out_angle
            if G0_G1_data(i-1)==1
                %INSIDE INTERPOLATION
                while i2<=length(x)-2
                    if x(i2)==x_diff(1)
                        if y(i2)==y_diff(1)
                            %Derivative directions
                            v=[xd(i2) yd(i2)];
                            vb=[xd(i2-1) yd(i2-1)];
```

```
                                %
                                t=0;
                                eq1=(subs(p) == [x(i2-1) y(i2-1)]);
                                eq2=(subs(dp) == K3*vb);
                                eq3=(subs(d2p) == [0 0]);
                                %
                                tt1=tt; %True time instant 1
                                length_seg= sqrt((x(i2)-x(i2-1))^2+(y(i2)-y(i2-1))^2);
                                tt=tt+length_seg/(v_max_extrusion*tlp);
                                tt_list=[tt_list(1:end),tt];
                                length_list=[length_list(1:end),length_seg];
                                tt2=tt; %True time instant 2
                                %
                                t=1;
                                eq4=(subs(p) == [x(i2) y(i2)]);
                                eq5=(subs(dp) == K4*v);
                                eq6=(subs(d2p) == [0 0]);
                                %
                                sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                                %
                                axSol=sol.ax;
                                aySol=sol.ay;
                                bxSol=sol.bx;
                                bySol=sol.by;
                                cxSol=sol.cx;
                                cySol=sol.cy;
                                dxSol=sol.dx;
                                dySol=sol.dy;
                                exSol=sol.ex;
                                eySol=sol.ey;
                                fxSol=sol.fx;
                                fySol=sol.fy;
                                %
                                x_diff=[x_diff(2:end)]; %Remove different point from
original list
                                y_diff=[y_diff(2:end)];
                                i2=i2+1;
                                %CALCULATE MAXIMUM VELOCITY
                                syms time
                                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                                %
                                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3
+ dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3
+ dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                                atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                                max_t=vpasolve(atotal==0,tt1);
                                v_max_i=(subs(vtotal,max_t));
                                v_max_i_real=double(real(v_max_i(5)));
                                if v_max_i_real > vL
                                    continue
                                end
                                v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                                v_max=max(v_max_total);
                                %
                                if isempty(x_diff)
```

```
                                break
                            end
                        else
                            i2=i2+1;
                        end
                    else
                        i2=i2+1;
                    end
                end
            else
                %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
                %Derivative directions
                v=[xd_o(i) yd_o(i)];
                vb=[xd_o(i-1) yd_o(i-1)];
                %
                t=0;
                eq1=(subs(p) == [p0x p0y]);
                eq2=(subs(dp) == K1*vb);
                eq3=(subs(d2p) == [0 0]);
                %
                %LENGTH OF THE SEGMENT
                tt1=tt;%True time instant 1
                length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
                tt=tt+length_seg/(v_max_extrusion*tlp);
                tt_list=[tt_list(1:end),tt];
                length_list=[length_list(1:end),length_seg];
                tt2=tt;%True time instant 2
                %
                t=1;
                eq4=(subs(p) == [p1x p1y]);
                eq5=(subs(dp) == K2*v);
                eq6=(subs(d2p) == [0 0]);
                %
                sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                %
                axSol=sol.ax;
                aySol=sol.ay;
                bxSol=sol.bx;
                bySol=sol.by;
                cxSol=sol.cx;
                cySol=sol.cy;
                dxSol=sol.dx;
                dySol=sol.dy;
                exSol=sol.ex;
                eySol=sol.ey;
                fxSol=sol.fx;
                fySol=sol.fy;
                %
                %CALCULATE MAXIMUM VELOCITY
                syms time
                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                %
                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
```

```
                    atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                    max_t=vpasolve(atotal==0,tt1);
                    v_max_i=(subs(vtotal,max_t));
                    v_max_i_real=double(real(v_max_i(5)));
                    if v_max_i_real > vL
                        continue
                    end
                    v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                    v_max=max(v_max_total);
                    %
                end
            else
                %REGULAR OUTSIDE INTERPOLATION
                %Derivative directions
                v=[xd_o(i) yd_o(i)];
                vb=[xd_o(i-1) yd_o(i-1)];
                t=0;
                eq1=(subs(p) == [p0x p0y]);
                eq2=(subs(dp) == K1*vb);
                eq3=(subs(d2p) == [0 0]);
                %
                tt1=tt;%True time instant 1
                length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
                tt=tt+length_seg/(v_max_extrusion*tlp);
                tt_list=[tt_list(1:end),tt];
                length_list=[length_list(1:end),length_seg];
                tt2=tt;%True time instant 2
                %
                t=1;
                eq4=(subs(p) == [p1x p1y]);
                eq5=(subs(dp) == K2*v);
                eq6=(subs(d2p) == [0 0]);
                %
                sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                %
                axSol=sol.ax;
                aySol=sol.ay;
                bxSol=sol.bx;
                bySol=sol.by;
                cxSol=sol.cx;
                cySol=sol.cy;
                dxSol=sol.dx;
                dySol=sol.dy;
                exSol=sol.ex;
                eySol=sol.ey;
                fxSol=sol.fx;
                fySol=sol.fy;
                %CALCULATE MAXIMUM VELOCITY
                syms time
                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                %
                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
```

```
                max_t=vpasolve(atotal==0,tt1);
                v_max_i=(subs(vtotal,max_t));
                v_max_i_real=double(real(v_max_i(5)));
                if v_max_i_real > vL %Eliminate false positives
                    continue
                end
                v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                v_max=max(v_max_total);
            end
        end
    end
    % OUTPUTS
    vmax_to_opt=v_max;
end
```

OBJECTIVE FUNCTION

```
function obj=objective(tlp)
    vemax=20;
    obj=((0.5*(vemax*0.9-calcvmax(tlp)))^2);
end
```

Appendix I

File: curve_construction.m

IMPORT DATA

```
clc,clear
close all;
data=csvread('interpolation_coefficients.csv');
tt_list=csvread('time_instants.csv');
syms t
```

TO COMPARE WITH THE ORIGINAL DATA

```
data_original=csvread('layer_coordinates.csv');
x_o=data_original(:,1).';
y_o=data_original(:,2).';
plot(x_o,y_o,'r')
hold on
%
```

ROUTINE

```
for i=1:length(data)
    ax=data(i,1).';
    bx=data(i,2).';
    cx=data(i,3).';
    dx=data(i,4).';
    ex=data(i,5).';
    fx=data(i,6).';
    %
    ay=data(i,7).';
    by=data(i,8).';
    cy=data(i,9).';
    dy=data(i,10).';
    ey=data(i,11).';
    fy=data(i,12).';
    %
    tt1=tt_list(i);%time instant 1
    tt2=tt_list(i+1);%time instant 2
    tt=(t-tt1)/(tt2-tt1); %normalized time instans
    %
    x= ax*tt.^5 + bx*tt.^4 + cx*tt.^3 + dx*tt.^2 + ex*tt + fx;
    y= ay*tt.^5 + by*tt.^4 + cy*tt.^3 + dy*tt.^2 + ey*tt + fy;
    fplot(x,y,[tt1 tt2],'b')
    xlabel('X coordinates');
    ylabel('Y coordinates');
    title('Cylinder Layer Polynomial Interpolation');
    drawnow
    hold on
    %
end
```

Appendix J

File: corrected_coords_inside_interp.m – CUBE CASE STUDY

IMPORT DATA

```
clc,clear
close all;
data=csvread('layer_coordinates.csv');
x=data(:,1).';
y=data(:,2).';
%EXTRUSION ON/OFF
G0_G1_data=csvread('G0_G1_matrix.csv');
% G0_G1_data=ones(85);%PARA TESTE - APAGAR DEPOIS
%
%ORIGINAL COORDINATES
original_x=data(:,1).';
original_y=data(:,2).';
%
```

PARAMETERS

```
a=0.01;
b=0.5;
c=0.01;
length_percentage=0.1;
in_out_angle=100; %INSIDE/OUTSIDE INTERPOLATION ANGLE LIMIT
sharp_edge_angle=100;
```

FUTURE STORAGE

```
%NEW COORDINATES LIST
xn=[];
yn=[];
diff=[];
%
%DERIVATIVES
dxy=[];
%
iterations_to_skip=0;
num_new_points=0;
%
```

MAIN ROUTINE

```
for i=1:length(x)-2
    if iterations_to_skip > 0
        iterations_to_skip = iterations_to_skip - 1;
        continue
    else
        if i==1
            xn=[x(1)];
```

```
        yn=[y(1)];
        p1x=x(i);
        p1y=y(i);
        p2x=x(i+1);
        p2y=y(i+1);
        v2 = [p2x,p2y] - [p1x,p1y];
        v=v2/norm(v2);
        dxy= [dxy(1:end);v];
        G0_G1_update=[1];
        continue
    elseif G0_G1_data(i-1)==0
        if G0_G1_data(i)==1
            %G0 FOLLOWED BY G1 (no sharp edge)
            xn = [xn(1:end) , x(i)];
            yn = [yn(1:end) , y(i)];
            %DERIVATIVE
            v1 = [p1x,p1y] - [p0x,p0y];
            v2 = [p2x,p2y] - [p1x,p1y];
            v=(v1+v2)/norm(v1+v2);
            dxy= [dxy(1:end,:);v];
            plot(xn,yn,'r.-')
            drawnow
            G0_G1_update=[G0_G1_update(1:end),1];
        else
            continue
        end
    else
        p0x=x(i-1);
        p0y=y(i-1);
        p1x=x(i);
        p1y=y(i);
        p2x=x(i+1);
        p2y=y(i+1);
        pax=x(i+2);
        pay=y(i+2);
        v1 = ([p1x,p1y] - [p0x,p0y]);
        v2 = ([p2x,p2y] - [p1x,p1y]);
        CosTheta = max(min(dot(v1,v2)/(norm(v1)*norm(v2)),1),-1);
        angle = 180-abs(real(acosd(CosTheta)));
        if abs(angle)<in_out_angle
            %INSIDE INTERPOLATION
            %Length of line before measurement
            length_lb= sqrt((p1x-p0x)^2+(p1y-p0y)^2);
            %Length of present line measurement
            length_la= sqrt((p2x-p1x)^2+(p2y-p1y)^2);
            %Lenght of line after measurement
            length_la2= sqrt((pax-p2x)^2+(pay-p2y)^2);
            if length_la<=length_percentage*length_lb
                a=(length_la*b)/length_lb;
                pn1x=(p0x*a + p1x*(1-a));%NEW POINT 1
                pn1y=(p0y*a + p1y*(1-a));
                pn2x=(p1x*(1-b) + p2x*b);%NEW POINT 2
                pn2y=(p1y*(1-b) + p2y*b);
                %
                %Vector after
                va = ([pax,pay] - [p2x,p2y])/norm([pax,pay] - [p2x,p2y]);
                CosTheta2 = max(min(dot(v2,va)/(norm(v2)*norm(va)),1),-1);
                angle2 = real(acosd(CosTheta2));
```

```
                if abs(angle2)<in_out_angle
                    %THIRD NEW POINT
                    %NEW COORDINATES
                    num_new_points=num_new_points+3;
                    c=(length_la*b)/length_la2;
                    pn3x=(p2x*(1-c) + pax*c);%NEW POINT 3
                    pn3y=(p2y*(1-c) + pay*c);
                    xn = [xn(1:end), pn1x , pn2x , pn3x];
                    yn = [yn(1:end), pn1y , pn2y , pn3y];
                    diff=[diff(1:end,:);pn1x,pn1y;pn2x,pn2y;pn3x,pn3y];
                    iterations_to_skip=1;
                    %DERIVATIVES FOR THE NEW POINTS
                    d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                    d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                    d3=([pn3x,pn3y] - [p2x,p2y])/norm([pn3x,pn3y] - [p2x,p2y]);
                    dxy= [dxy(1:end,:);d1;d2;d3];
                    plot(xn,yn,'r.-')
                    drawnow
                    G0_G1_update=[G0_G1_update(1:end),1,1,1];
                else
                    %TWO NEW POINTS
                    %NEW COORDINATES
                    num_new_points=num_new_points+2;
                    xn = [xn(1:end) , pn1x , pn2x];
                    yn = [yn(1:end) , pn1y , pn2y];
                    diff=[diff(1:end,:);pn1x,pn1y;pn2x,pn2y];
                    %DERIVATIVES
                    d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                    d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                    dxy= [dxy(1:end,:);d1;d2];
                    plot(xn,yn,'r.-')
                    drawnow
                    G0_G1_update=[G0_G1_update(1:end),1,1];
                end
            else
                %TWO NEW POINTS
                %NEW COORDINATES
                c=(length_lb*a)/length_la;
                num_new_points=num_new_points+2;
                pn1x=(p0x*a + p1x*(1-a));%NEW POINT 1
                pn1y=(p0y*a + p1y*(1-a));
                pn2x=(p1x*(1-c) + p2x*c);%NEW POINT 2
                pn2y=(p1y*(1-c) + p2y*c);
                xn = [xn(1:end) , pn1x , pn2x];
                yn = [yn(1:end) , pn1y , pn2y];
                diff=[diff(1:end,:);pn1x,pn1y;pn2x,pn2y];
                %DERIVATIVE
                d1=([p1x,p1y] - [pn1x,pn1y])/norm([p1x,p1y] - [pn1x,pn1y]);
                d2=([pn2x,pn2y] - [p1x,p1y])/norm([pn2x,pn2y] - [p1x,p1y]);
                dxy= [dxy(1:end,:);d1;d2];
                plot(xn,yn,'r.-')
                drawnow
                G0_G1_update=[G0_G1_update(1:end),1,1];
            end
        else
            %NO SHARP EDGE
            xn = [xn(1:end) , x(i)];
            yn = [yn(1:end) , y(i)];
```

```
                %DERIVATIVE
                v1 = [p1x,p1y] - [p0x,p0y];
                v2 = [p2x,p2y] - [p1x,p1y];
                v=(v1+v2)/norm(v1+v2);
                dxy= [dxy(1:end,:);v];
                plot(xn,yn,'r.-')
                drawnow
                G0_G1_update=[G0_G1_update(1:end),1];
            end
        end
    end
end
```

EXPORT OUTPUT

```
plot(original_x,original_y,'b.-')
hold on
plot(xn,yn,'r.-')
M=[xn;yn].';
writematrix(M,'corrected_coords.csv')
writematrix(G0_G1_update,'G0_G1_matrix_updated.csv');
writematrix(dxy,'xy_derivatives_inside.csv')
writematrix(diff,'diff_coords.csv');
```

Appendix K

File: Interpolation_coefficients_vmax_calculation.m – CUBE CASE STUDY

IMPORT DATA

```
clc,clear
close all;
%
%CORRECTED COORDINATES FOR INSIDE INTERPOLATION
%
data=csvread('corrected_coords.csv');
x=data(:,1).';
y=data(:,2).';
%
%EXTRUSION ON/OFF
G0_G1_data=csvread('G0_G1_matrix.csv');
G0_G1_data=csvread('G0_G1_matrix_updated.csv');
% G0_G1_data=ones(85);%PARA TESTE - APAGAR DEPOIS
%
%ORIGINAL DATA
%
data_original=csvread('layer_coordinates.csv');
x_o=data_original(:,1).';
y_o=data_original(:,2).';
%
%STABLE DIFFERENCES BETWEEN ORIGINAL AND CORRECTED COORDINATES
%
% data_diff=setdiff(data,data_original,'stable');
x_diff=setdiff(x,x_o,'stable');
y_diff=setdiff(y,y_o,'stable');
diff_coord=csvread('diff_coords.csv');
%
%OUTSIDE DERIVATIVES
%
ddata=csvread('xy_derivatives_outside.csv');
xd_o=ddata(:,1).';
yd_o=ddata(:,2).';
%INSIDE DERIVATIVES
%
ddata=csvread('xy_derivatives_inside.csv');
xd=ddata(:,1).';
yd=ddata(:,2).';
%
```

PARAMETERS DEFINITION

```
in_out_angle=100;
sharp_edge_angle=100;
v_max_extrusion=20; %MAXIMUM EXTRUSION SPEED [mm/s]
true_length_percentage=0.40; %TLP PARAMETER
vL=50; %VELOCITY LIMIT PARAMETER
%
%K VALUES
```

```
K1=0.40;
K2=0.40;
K3=0.15;
K4=0.15;
%
```

POLYNOMIAL DEFINITION

```
syms t ax ay bx by cx cy dx dy ex ey fx fy
syms tt %Ture time variable
p = [ax ay]*t^5 + [bx by]*t^4 + [cx cy]*t^3 + [dx dy]*t^2 + [ex ey]*t + [fx fy];
dp = diff(p,t); %Polynomial 1st derivative
d2p = diff(p,t,2); %polynomial 2nd derivative
%
t=0;%First time instant
num_sharp_edges=0;
empty_vector=zeros(1,12);
%
tt=0;%True time count
length_seg=0;
```

FUTURE STORAGE

```
tt_list=[0];
length_list=[];
poly_coef=[];
v_max_total=[];
%
```

MAIN ROUTINE

```
i2=2;
for i=2:length(x)-1
    p0x=x(i-1);
    p0y=y(i-1);
    p1x=x(i);
    p1y=y(i);
    p2x=x(i+1);
    p2y=y(i+1);
    v1 = [p1x,p1y] - [p0x,p0y];
    v2 = [p2x,p2y] - [p1x,p1y];
    % Manipulate infill result - only applicable on this CASE study
    if i==27
        continue
    end
    if (27<i)
        length_lb= sqrt((p1x-p0x)^2+(p1y-p0y)^2);
        if (0.75>length_lb)
            continue
        end
    end
    if G0_G1_data(i)==0;
        continue
    end
```

```
    % Continue
    if isempty(diff_coord)
        %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
        %Derivative directions
        v=[xd(i) yd(i)];
        vb=[xd(i-1) yd(i-1)];
        %
        t=0;
        eq1=(subs(p) == [p0x p0y]);
        eq2=(subs(dp) == K1*vb);
        eq3=(subs(d2p) == [0 0]);
        %
        %LENGTH OF THE SEGMENT
        tt1=tt;%True time instant 1
        length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
        tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
        tt_list=[tt_list(1:end),tt];
        length_list=[length_list(1:end),length_seg];
        tt2=tt;%True time instant 2
        %
        t=1;
        eq4=(subs(p) == [p1x p1y]);
        eq5=(subs(dp) == K2*v);
        eq6=(subs(d2p) == [0 0]);
        %
        sol = solve([eq1,eq2,eq3,eq4,eq5,eq6], [ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
        %
        axSol=sol.ax;
        aySol=sol.ay;
        bxSol=sol.bx;
        bySol=sol.by;
        cxSol=sol.cx;
        cySol=sol.cy;
        dxSol=sol.dx;
        dySol=sol.dy;
        exSol=sol.ex;
        eySol=sol.ey;
        fxSol=sol.fx;
        fySol=sol.fy;
        %
        pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
        %
        new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol dySol eySol
fySol]);
        poly_coef=[poly_coef(1:end,:); new_coefs];
        %CALCULATE MAXIMUM VELOCITY
        syms time
        tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
        %
        x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 + dxSol*tt_sym.^2 +
exSol*tt_sym + fxSol;
        y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 + dySol*tt_sym.^2 +
eySol*tt_sym + fySol;
        vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
        atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
        max_t=vpasolve(atotal==0,tt1);
        v_max_i=(subs(vtotal,max_t));
```

```
        v_max_i_real=double(real(v_max_i(5)));
        if v_max_i_real > vL
            continue
        end
        v_max_total=[v_max_total(1:end);double(v_max_i_real)];
        v_max=max(v_max_total);
        fplot(vtotal,[tt1,tt2]);
        hold on
        %
    else
        if [x(i),y(i)]==diff_coord(1,:)
            v=[xd(i) yd(i)];
            vb=[xd(i-1) yd(i-1)];
            %
            t=0;
            eq1=(subs(p) == [x(i-1) y(i-1)]);
            eq2=(subs(dp) == K3*vb);
            eq3=(subs(d2p) == [0 0]);
            %
            tt1=tt; %True time instant 1
            length_seg= sqrt((x(i)-x(i-1))^2+(y(i)-y(i-1))^2);
            tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
            tt_list=[tt_list(1:end),tt];
            length_list=[length_list(1:end),length_seg];
            tt2=tt; %True time instant 2
            %
            t=1;
            eq4=(subs(p) == [x(i) y(i)]);
            eq5=(subs(dp) == K4*v);
            eq6=(subs(d2p) == [0 0]);
            %
            sol = solve([eq1,eq2,eq3,eq4,eq5,eq6], [ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
            %
            axSol=sol.ax;
            aySol=sol.ay;
            bxSol=sol.bx;
            bySol=sol.by;
            cxSol=sol.cx;
            cySol=sol.cy;
            dxSol=sol.dx;
            dySol=sol.dy;
            exSol=sol.ex;
            eySol=sol.ey;
            fxSol=sol.fx;
            fySol=sol.fy;
            %
            pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];

            new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol dySol
eySol fySol]);
            poly_coef=[poly_coef(1:end,:); new_coefs];
            %
            diff_coord=[diff_coord(2:end,:)];
            %CALCULATE MAXIMUM VELOCITY
            syms time
            tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
            %
```

```
            x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 + dxSol*tt_sym.^2 +
exSol*tt_sym + fxSol;
            y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 + dySol*tt_sym.^2 +
eySol*tt_sym + fySol;
            vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
            atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
            max_t=vpasolve(atotal==0,tt1);
            v_max_i=(subs(vtotal,max_t));
            v_max_i_real=double(real(v_max_i(5)));
            if v_max_i_real > vL
                continue
            end
            v_max_total=[v_max_total(1:end);double(v_max_i_real)];
            v_max=max(v_max_total);
            fplot(vtotal,[tt1,tt2]);
            hold on
        else
            %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
            %Derivative directions
            v=[xd(i) yd(i)];
            vb=[xd(i-1) yd(i-1)];
            %
            t=0;
            eq1=(subs(p) == [p0x p0y]);
            eq2=(subs(dp) == K1*vb);
            eq3=(subs(d2p) == [0 0]);
            %
            %LENGTH OF THE SEGMENT
            tt1=tt;%True time instant 1
            length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
            tt=tt+length_seg/(v_max_extrusion*true_length_percentage);
            tt_list=[tt_list(1:end),tt];
            length_list=[length_list(1:end),length_seg];
            tt2=tt;%True time instant 2
            %
            t=1;
            eq4=(subs(p) == [p1x p1y]);
            eq5=(subs(dp) == K2*v);
            eq6=(subs(d2p) == [0 0]);
            %
            sol = solve([eq1,eq2,eq3,eq4,eq5,eq6], [ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
            %
            axSol=sol.ax;
            aySol=sol.ay;
            bxSol=sol.bx;
            bySol=sol.by;
            cxSol=sol.cx;
            cySol=sol.cy;
            dxSol=sol.dx;
            dySol=sol.dy;
            exSol=sol.ex;
            eySol=sol.ey;
            fxSol=sol.fx;
            fySol=sol.fy;
            %
            pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
            %
```

```
            new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol dySol
eySol fySol]);
            poly_coef=[poly_coef(1:end,:); new_coefs];
            %CALCULATE MAXIMUM VELOCITY
            syms time
            tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
            %
            x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 + dxSol*tt_sym.^2 +
exSol*tt_sym + fxSol;
            y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 + dySol*tt_sym.^2 +
eySol*tt_sym + fySol;
            vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
            atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
            max_t=vpasolve(atotal==0,tt1);
            v_max_i=(subs(vtotal,max_t));
            v_max_i_real=double(real(v_max_i(5)));
            if v_max_i_real > vL
                continue
            end
            v_max_total=[v_max_total(1:end);double(v_max_i_real)];
            v_max=max(v_max_total);
            fplot(vtotal,[tt1,tt2]);
            hold on
            %
        end
    end
end
```

EXPORT OUTPUTS

```
xlabel('time (s)');
ylabel('Linear velocity module (mm/s)');
title('Linear velocity module before optimization');
display(v_max)
writematrix(poly_coef,'interpolation_coefficients.csv')
writematrix(tt_list,'time_instants.csv')
```

Appendix L

File: optimization.m - CUBER CASE STUDY

INTRODUCTION

```
clc,clear
close all;
vemax=20;
%
```

INITIAL GUESSES

```
tlp_guess=0.4;
x0=[tlp_guess];
%
```

CALL THE SOLVER

```
opt_tlp=fmincon(@objective,x0,[],[],[],[],[],[],[],[])
vmax_opt=calcvmax(opt_tlp)
%
```

VMAX CALCULATION

```
function vmax_to_opt=calcvmax(tlp)
```

```
    %CORRECTED COORDINATES FOR INSIDE INTERPOLATION
    %
    data=csvread('corrected_coords.csv');
    x=data(:,1).';
    y=data(:,2).';
    %
    %EXTRUSION ON/OFF
    G0_G1_data=csvread('G0_G1_matrix.csv');
    G0_G1_data=csvread('G0_G1_matrix_updated.csv');
    % G0_G1_data=ones(85);%PARA TESTE - APAGAR DEPOIS
    %
    %ORIGINAL DATA
    %
    data_original=csvread('layer_coordinates.csv');
    x_o=data_original(:,1).';
    y_o=data_original(:,2).';
    %
    %STABLE DIFFERENCES BETWEEN ORIGINAL AND CORRECTED COORDINATES
    %
    % data_diff=setdiff(data,data_original,'stable');
    x_diff=setdiff(x,x_o,'stable');
    y_diff=setdiff(y,y_o,'stable');
    diff_coord=csvread('diff_coords.csv');
    %
    %OUTSIDE DERIVATIVES
```

```
%
ddata=csvread('xy_derivatives_outside.csv');
xd_o=ddata(:,1).';
yd_o=ddata(:,2).';
%INSIDE DERIVATIVES
%
ddata=csvread('xy_derivatives_inside.csv');
xd=ddata(:,1).';
yd=ddata(:,2).';
%
```

```
in_out_angle=100;
sharp_edge_angle=100;
v_max_extrusion=20; %MAXIMUM EXTRUSION SPEED [mm/s]
vL=50; %VELOCITY LIMIT PARAMETER
%
%K VALUES
K1=0.40;
K2=0.40;
K3=0.15;
K4=0.15;
%
```

```
syms t ax ay bx by cx cy dx dy ex ey fx fy
syms tt %Ture time variable
p = [ax ay]*t^5 + [bx by]*t^4 + [cx cy]*t^3 + [dx dy]*t^2 + [ex ey]*t + [fx fy];
dp = diff(p,t); %Polynomial 1st derivative
d2p = diff(p,t,2); %polynomial 2nd derivative
%
t=0;%First time instant
num_sharp_edges=0;
empty_vector=zeros(1,12);
%
tt=0;%True time count
length_seg=0;
```

```
tt_list=[0];
length_list=[];
poly_coef=[];
v_max_total=[];
i2=2;
for i=2:length(x)-1
    p0x=x(i-1);
    p0y=y(i-1);
    p1x=x(i);
    p1y=y(i);
    p2x=x(i+1);
    p2y=y(i+1);
    v1 = [p1x,p1y] - [p0x,p0y];
    v2 = [p2x,p2y] - [p1x,p1y];
    % Manipulate infill result - only applicable on this example
    if (27<i)
        length_lb= sqrt((p1x-p0x)^2+(p1y-p0y)^2);
        if (0.75>length_lb)
            continue
        end
    end
    if G0_G1_data(i)==0;
        continue
```

```
        end
        % Continue
        if isempty(diff_coord)
            %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
            %Derivative directions
            v=[xd(i) yd(i)];
            vb=[xd(i-1) yd(i-1)];
            %
            t=0;
            eq1=(subs(p) == [p0x p0y]);
            eq2=(subs(dp) == K1*vb);
            eq3=(subs(d2p) == [0 0]);
            %
            %LENGTH OF THE SEGMENT
            tt1=tt;%True time instant 1
            length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
            tt=tt+length_seg/(v_max_extrusion*tlp);
            tt_list=[tt_list(1:end),tt];
            length_list=[length_list(1:end),length_seg];
            tt2=tt;%True time instant 2
            %
            t=1;
            eq4=(subs(p) == [p1x p1y]);
            eq5=(subs(dp) == K2*v);
            eq6=(subs(d2p) == [0 0]);
            %
            sol = solve([eq1,eq2,eq3,eq4,eq5,eq6], [ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
            %
            axSol=sol.ax;
            aySol=sol.ay;
            bxSol=sol.bx;
            bySol=sol.by;
            cxSol=sol.cx;
            cySol=sol.cy;
            dxSol=sol.dx;
            dySol=sol.dy;
            exSol=sol.ex;
            eySol=sol.ey;
            fxSol=sol.fx;
            fySol=sol.fy;
            %
            pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
            %
            new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol dySol
eySol fySol]);
            poly_coef=[poly_coef(1:end,:); new_coefs];
            %CALCULATE MAXIMUM VELOCITY
            syms time
            tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
            %
            x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 + dxSol*tt_sym.^2 +
exSol*tt_sym + fxSol;
            y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 + dySol*tt_sym.^2 +
eySol*tt_sym + fySol;
            vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
            atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
            max_t=vpasolve(atotal==0,tt1);
```

```
            v_max_i=(subs(vtotal,max_t));
            v_max_i_real=double(real(v_max_i(5)));
            if v_max_i_real > vL
                continue
            end
            v_max_total=[v_max_total(1:end);double(v_max_i_real)];
            v_max=max(v_max_total);
            %
        else
            if [x(i),y(i)]==diff_coord(1,:)
                v=[xd(i) yd(i)];
                vb=[xd(i-1) yd(i-1)];
                %
                t=0;
                eq1=(subs(p) == [x(i-1) y(i-1)]);
                eq2=(subs(dp) == K3*vb);
                eq3=(subs(d2p) == [0 0]);
                %
                tt1=tt; %True time instant 1
                length_seg= sqrt((x(i)-x(i-1))^2+(y(i)-y(i-1))^2);
                tt=tt+length_seg/(v_max_extrusion*tlp);
                tt_list=[tt_list(1:end),tt];
                length_list=[length_list(1:end),length_seg];
                tt2=tt; %True time instant 2
                %
                t=1;
                eq4=(subs(p) == [x(i) y(i)]);
                eq5=(subs(dp) == K4*v);
                eq6=(subs(d2p) == [0 0]);
                %
                sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                %
                axSol=sol.ax;
                aySol=sol.ay;
                bxSol=sol.bx;
                bySol=sol.by;
                cxSol=sol.cx;
                cySol=sol.cy;
                dxSol=sol.dx;
                dySol=sol.dy;
                exSol=sol.ex;
                eySol=sol.ey;
                fxSol=sol.fx;
                fySol=sol.fy;
                %
                pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];

                new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol
dySol eySol fySol]);
                poly_coef=[poly_coef(1:end,:); new_coefs];
                %
                diff_coord=[diff_coord(2:end,:)];
                %CALCULATE MAXIMUM VELOCITY
                syms time
                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                %
```

```
                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                max_t=vpasolve(atotal==0,tt1);
                v_max_i=(subs(vtotal,max_t));
                v_max_i_real=double(real(v_max_i(5)));
                if v_max_i_real > vL
                    continue
                end
                v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                v_max=max(v_max_total);
                %
                %         if isempty(diff_coord)
                %             continue
                %         end
            else
                %NO SHARP EDGE BUT OUTSIDE INTERPOLATION
                %Derivative directions
                v=[xd(i) yd(i)];
                vb=[xd(i-1) yd(i-1)];
                %
                t=0;
                eq1=(subs(p) == [p0x p0y]);
                eq2=(subs(dp) == K1*vb);
                eq3=(subs(d2p) == [0 0]);
                %
                %LENGTH OF THE SEGMENT
                tt1=tt;%True time instant 1
                length_seg = sqrt((p1x-p0x)^2+(p1y-p0y)^2);
                tt=tt+length_seg/(v_max_extrusion*tlp);
                tt_list=[tt_list(1:end),tt];
                length_list=[length_list(1:end),length_seg];
                tt2=tt;%True time instant 2
                %
                t=1;
                eq4=(subs(p) == [p1x p1y]);
                eq5=(subs(dp) == K2*v);
                eq6=(subs(d2p) == [0 0]);
                %
                sol = solve([eq1,eq2,eq3,eq4,eq5,eq6],
[ax,ay,bx,by,cx,cy,dx,dy,ex,ey,fx,fy]);
                %
                axSol=sol.ax;
                aySol=sol.ay;
                bxSol=sol.bx;
                bySol=sol.by;
                cxSol=sol.cx;
                cySol=sol.cy;
                dxSol=sol.dx;
                dySol=sol.dy;
                exSol=sol.ex;
                eySol=sol.ey;
                fxSol=sol.fx;
                fySol=sol.fy;
                %
```

```
                pp= [axSol aySol]*tt^5 + [bxSol bySol]*tt^4 + [cxSol cySol]*tt^3 + [dxSol
dySol]*tt^2 + [exSol eySol]*tt + [fxSol fySol];
                %
                new_coefs=double([axSol bxSol cxSol dxSol exSol fxSol aySol bySol cySol
dySol eySol fySol]);
                poly_coef=[poly_coef(1:end,:); new_coefs];
                %CALCULATE MAXIMUM VELOCITY
                syms time
                tt_sym=(time-tt1)/(tt2-tt1); %Normalized time instans
                %
                x_pol= axSol*tt_sym.^5 + bxSol*tt_sym.^4 + cxSol*tt_sym.^3 +
dxSol*tt_sym.^2 + exSol*tt_sym + fxSol;
                y_pol= aySol*tt_sym.^5 + bySol*tt_sym.^4 + cySol*tt_sym.^3 +
dySol*tt_sym.^2 + eySol*tt_sym + fySol;
                vtotal=sqrt((diff(x_pol,time)).^2+(diff(y_pol,time)).^2);
                atotal=((diff(x_pol,2)).^2+(diff(y_pol,2)).^2);
                max_t=vpasolve(atotal==0,tt1);
                v_max_i=(subs(vtotal,max_t));
                v_max_i_real=double(real(v_max_i(5)));
                if v_max_i_real > vL
                    continue
                end
                v_max_total=[v_max_total(1:end);double(v_max_i_real)];
                v_max=max(v_max_total);
                %
            end
        end
    end
    % OUTPUTS
    vmax_to_opt=v_max;

end
```

OBJECTIVE FUNCTION

```
function obj=objective(tlp)
    vemax=20;
    obj=((0.5*(vemax*0.9-calcvmax(tlp)))^2);
end
```