A NOVEL DESIGN AND FABRICATION PARADIGM FOR 3D PRINTERS: LIPRO

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

VAHID HASELTALAB

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

SEPTEMBER 2018

Approval of the thesis:

A NOVEL DESIGN AND FABRICATION PARADIGM FOR 3D PRINTERS: LIPRO

submitted by **VAHID HASELTALAB** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ____________

Prof. Dr. M. A. Sahir Arıkan
Head of Department, **Mechanical Engineering** ____________

Assist. Prof. Dr. Ulaş Yaman
Supervisor, **Mechanical Engineering Department, METU** ____________

Assoc. Prof. Dr. Melik Dölen
Co-supervisor, **Mechanical Engineering Department, METU** ____________

Examining Committee Members:

Prof. Dr. Serkan Dağ
Mechanical Engineering Department, METU ____________

Assist. Prof. Dr. Ulaş Yaman
Mechanical Engineering Department, METU ____________

Assist. Prof. Dr. Evren Yasa
Mechanical Engineering Department,
Eskisehir Osmangazi University ____________

Assist. Prof. Dr. Orkun Özşahin
Mechanical Engineering Department, METU ____________

Assist. Prof. Dr. Sezer Özerinç
Mechanical Engineering Department, METU ____________

Date: 06.09.2018

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Vahid Haseltalab

Signature :

ABSTRACT

A NOVEL DESIGN AND FABRICATION PARADIGM FOR 3D PRINTERS: LIPRO

Haseltalab, Vahid
M.S., Department of Mechanical Engineering
Supervisor : Assist. Prof. Dr. Ulaş Yaman
Co-Supervisor : Assoc. Prof. Dr. Melik Dölen

September 2018, 103 pages

This study starts with highlighting some disadvantages of the conventional design and fabrication pipelines of Additive Manufacturing (AM) processes. In order to overcome the major drawbacks of the conventional pipeline, a novel design and fabrication pipeline called as LIPRO is proposed and it is implemented on two different AM processes to illuminate its effectiveness on alleviating the disadvantages of the conventional approaches. A single board computer is used to realize the method on AM machines. The structure of the LIPRO based on different types of functions is explained and the developed Python scripts are appended to the thesis. By employing this method, some sample parts are fabricated with two AM processes; namely Fused Deposition Modeling and Digital Light Processing. The details of this implementation are elaborated and the advantages are discussed throughout the thesis.

Keywords: Curve Offset Generation, Additive Manufacturing, Command Generation,

Fused Deposition Modeling, Digital Light Processing.

ÖZ

3B YAZICILAR İÇİN YENİ BİR TASARIM VE ÜRETİM AKIŞI: LIPRO

Haseltalab, Vahid
Yüksek Lisans, Makina Mühendisliği Bölümü
Tez Yöneticisi : Dr. Öğr. Üyesi Ulaş Yaman
Ortak Tez Yöneticisi : Doç. Dr. Melik Dölen

Eylül 2018 , 103 sayfa

Bu çalışma, Eklemeli Üretim (EÜ) yöntemlerinde kullanılan geleneksel tasarım ve üretim akışının sorunlarını tartışarak başlamaktadır. Geleneksel akışın sahip olduğu dezavantajların üstesinden gelmek amacıyla LIPRO isimli yeni bir tasarım ve üretim akışı önerilmiş ve başarımını göstermek amacıyla iki farklı EÜ yöntemi üzerinde gerçekleştirilmiştir. Tek kartlı bir bilgisayar kullanılarak gerçekleştirilen yöntem 3B yazıcılar üzerinde sınanmıştır. Birçok farklı fonksiyona dayanan LIPRO'un yapısı açıklanmış ve geliştirilen Python kodları tez içerisinde sunulmuştur. Önerilen yöntem iki farklı EÜ yöntemi, Eriyik Yığma Modelleme ve Sayısal Işık İşleme, kullanılarak çeşitli parçalar üretilmiştir. Uygulanan yöntemin detayları tez içerisinde açıklanmış ve avantajları tartışılmıştır.

Anahtar Kelimeler: Eğri Ofset Üretimi, Eklemeli Üretim, Komut Üretimi, Eriyik

Yığma Modelleme, Sayısal Işık İşleme.

Dedicated with love and thanks to my family, who have given me so much along the way.

ACKNOWLEDGMENTS

Hereby I would like to express my deep gratitude to my supervisor Assist. Prof. Dr. Ulaş Yaman, for his generous support and guidance throughout my studies. His constant encouragements have given me the confidence to complete this dissertation. I also would like to thank Assoc. Prof. Dr. Melik Dölen for conveying his invaluable knowledge that helped me at some points of this thesis.

The author also would thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for granting the scholarship with the code 116E882.

I owe my sincere gratitude to my family who gives me strength and assurance throughout my studies. I would like to acknowledge the invaluable assistance and counseling of my deepest friend Dr. Reza Zeinali especially at the starting of my study. I also appreciate the support of my friends Armin Taghipour, Sardar Vayghannezhad, Bahar Garehpapagh, Rasul Tarvirdilu Asl, Ramin Rouzbar and appreciate their intimate friendships during these years.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS

L	Length of base curve
N	Number of points on perimeter of a curve
P	Base curve
p_k	The k^{th} index of set P
q	A point on a curve offset
r	Offset distance
S	Set of polygons
u	Direction of a vector
α	Minimum angle of the curvature
δ	Maximum distance between two successive points
ϵ	Error tolerance band for generating curve offset
θ	Rotation angle

ABBREVIATION

ABS	Acrylonitrile Butadiene Styrene
AM	Additive Manufacturing
AMOBS	Advanced Morphological Operation on Boundary Set
CAD	Computer Aided Design
CBS	Creation of Boundary Set
CCO	Creation of Curve Offsets
CCW	CounterClockWise
CLIP	Continuous Liquid Interface Production
CW	ClockWise
DDM	Direct Digital Manufacturing
DLP	Digital Light Processing
DMD	Digital Micromirror Device
FDM	Fused Deposition Modeling
GPIO	General-Purpose Input/Output
IMOBS	Improved Morphological Operation on Boundary Set
MFO	Merging the Fragmented Offsets
NURBS	Non-Uniform Rational B-Spline
OM	Operation Management
PIL	Python Imaging Library
PLA	Polylactic Acid
STL	Standard Tessellation Language

CHAPTER 1

INTRODUCTION

1.1 Motivation

In the last three decades, advancements in Additive Manufacturing (AM) undoubtedly have had a huge impact on the manufacturing technology. Despite the massive advantages of these processes in comparison to the traditional ones, there are still numerous complications in their manufacturing pipeline. The conventional plan for fabricating with AM processes requires an STL (Standard Tessellation Language) file format which is obtained from Computer Aided Design (CAD) software. The STL file is later sliced in Computer Aided Manufacturing (CAM) software and most of the manufacturing parameters must be fixed in this environment before the commands are transferred to the machine. This arrangement brings some limitations in the AM technologies which draws attention of researchers. These drawbacks are mostly in the prefabrication time including the design of the CAD model, conversion to the STL file format from the original CAD file format, slicing, setting the fabrication parameters and generating the G-codes. In some complicated artifacts having high resolutions, these issues become very significant. For instance, in Continuous Liquid Interface Production process known as CLIP, in order to produce a single hearing aid whose STL file has more than 1 million triangles, it takes 10 hours to prepare the model for printing whereas the fabrication process only takes 10 minutes (Kwok et al. [1]). Another major problem of the conventional pipeline is that after the conversion of the design model to an STL file, the additional information of the model (such as material, color etc.) will no longer exist. If the design artifact happens to be changed during the fabrication process, the operation must be aborted and all of the process planning must be exerted to the new design. These factors were the motivation of the proposed study by [2, 3]. Their focus was to come up with a solution that can alleviate the terms described and this dissertation will present an implementation of the corresponding works.

1.2 Scope of the Thesis

LIPRO, a design and fabrication pipeline proposed for AM processes, is an abbreviation of List Processing. Since the structure of this method provided in Python programming language and lists are one the major Python data type employed for storing data in each process of this study, the name LIPRO is chosen to describe this fabrication paradigm. To appreciate this new command generation and its components, a background information along with the recent developments on each stage of fabrication pipeline is presented in Chapter 2. Furthermore, Chapter 3 will provide the concept of the proposed approach and it will demonstrate each particular operation involved in this technique. To assess the performance and benefits of LIPRO, Chapter 4 and 5 are presented. In these Chapters, an implementation of this method is applied on an Fused Deposition Modeling (FDM) and a Digital Light Processing (DLP) printer by utilizing Python programming language and through a comparison with conventional approaches, advantages of LIPRO is highlighted. Finally, the last Chapter will recap the experimental work discussed in the earlier Chapters and in the end, the implemented Python scripts is delivered.

1.3 Limitations of the Study

Like every study, this one also has some restrictions which can be improved in the future. The following context highlights some of these limitations.

- Currently, there is no Graphical User Interface (GUI) for the LIPRO that enables the users to communicate with the design in a comfortable manner.
- There is no function available in the LIPRO that handles support structures at the moment, but this feature can be included in the future.

- At this moment, the proposed paradigm only uses contour parallel tool-path for filling each layer.
- LIPRO is unable to adjust the orientation of the design artifact.
- The current slicing method presented in the LIPRO is not efficient enough in terms of execution time.

CHAPTER 2

BACKGROUND

2.1 Introduction

As mentioned in the introductory chapter, there is a basis procedure for almost all AM processes from designing to the fabrication of the product. Throughout the years, many studies provided to amend the limitations of the existing process pipelines. Similar to these studies, the thesis has also the same objective and in order to demonstrate the details of the proposed approach, the prior works are summarized in this section. This literature survey starts with the importance of additive manufacturing and the changes it provides in today's manufacturing organizations (Section 2.2). Then, some recent researches regarding fabrication pipeline of AM processes are evaluated. Since some steps in the pipeline are playing significant roles, their background are delivered in more details in the study. Finally, in Sections 2.7 and 2.8 the fundamentals of the two AM processes which are employed in this dissertation are described briefly.

2.2 Direct Digital Manufacturing

Direct Digital Manufacturing (DDM) or Direct Additive Manufacturing is the process of using AM techniques for producing ready-to-use products and it has been evolving in the recent years considerably. Nowadays, demands for fabricating customized products with lower cost and better quality have encouraged researchers to develop AM processes as a solution for this challenge. David Bak [4] discussed the benefits of rapid manufacturing and how it can improve the industrial economy. In another work, Holmstrom et al. [5] discussed the implications of DDM and how it may change the operations management (OM). They claimed that by utilizing DDM, many of these OM principles may alter or even become unnecessary including job-shop problems, supply chain management and batch sizing. Chen et al. [6] highlighted some of DDM merits by comparing it with other paradigms like craft production, mass production and mass customization. Petrovic et al. [7] also conducted some case studies to present the application of AM in various industries and they emphasized the privilege of using AM processes through these test cases.

Mechanical properties of metal products were always a major concern in AM industry. However, in the last two decades, a significant amount of development was established in this area which enhanced the DDM capabilities. Some of these direct methods consist of Selective Laser Melting (SLM), Laser Metal Deposition (LMD) and Electron Beam Melting (EBM) are analyzed in [8] in terms of the qualities of the final products. In this study, the mechanical and physical properties of the metal parts are evaluated in comparison with traditional processes. Results of this comparison stated that the products manufactured by AM processes have better properties. The only constraint is the property's dependency on the process selection. In another study, Mari Koike et al. [9] investigated the mechanical properties of titanium alloy products which are manufactured by EBM for the purpose of dental applications. The examination of metal parts from AM processes is also presented in [10–13]. In each of these studies, the properties of the final specimen was evaluated regarding their fabrication purpose and some suggestions were proposed to increase their functionality.

2.3 Design and Fabrication Pipelines

Regarding fabrication pipeline, although it is a privilege to use AM processes rather than traditional methods, their pipeline still require a great deal of careful consideration [14]. Figure 2.1 indicates the base configuration of pipeline in AM. This is the general form and there are some other details which may vary according to the processes including part orientation and support structures. The figure represents the conventional pipeline and it is still the practical method of most of the 3D printers.

Even though it has some limitations, which were explained in the previous Chapter, there is no efficient pipeline introduced which can overcome all the problems up to now. However, there are some effective researches conducted recently for each step of the AM pipeline that can remodel this format in the future. One of the restrictions exist in conventional pipeline is about the thickness of the last layer which sometimes is less than the machine resolution. For this purpose, Telea-Jalba [15] proposed a method using voxel-based representations to detect these regions. Rolland-Nevier [16] also utilized a shape diameter function to estimate the corresponding thickness of the artifacts. To overcome this challenge, Wang-Chen [17] suggested an approach to increase the thickness of the 3D objects so that the layer will be printed. Another constraint is due to the volumes of the machines. They may fabricate parts up to a specific size. For this case, Luo et al. [18] came up with an algorithm to separate the parts into pieces (printable with the machine) and then reassemble the components to form the original part. Triangular tessellation (STL file conversion) is another factor affecting the accuracy and robustness of the printed products. There are some common errors happen when a CAD model converts to STL file format including missing triangles or flipped triangles. These errors cannot be observed visually but they interfere the slicing process and make it unreliable. That is why, some studies attempt to provide techniques to repair ([19–21]) and improve the meshes for gaining better quality based on the application of the final product.

Orientation of the part is a crucial parameter since it affects the amount of support structures, the execution time, the final quality of the surfaces and its practicality. Earlier studies discussed the suitable direction of the parts [23–27], but in the later ones, they focused on optimizing the orientation based on the specific concerns of the fabrication. For instance, some works intended to reduce the amounts of support structures [28–30], some others considered the accuracy and the surface quality as their basis [31–36] and lastly, functionality was at the center of attention while considering the part orientation [37, 38]. One of the main concepts of process planning in AM technology is devoted to the generation of support structures. In many processes, they are needed to avoid the material falls. On the other hand, they can increase the cost of manufacturing considerably. For this reason, some researchers offered to change the original design to decrease (Kailun Hu et al. [39]) the amount or even eliminate (Reiner-Lefebvre [40]) the support structures. Generally, the pro-

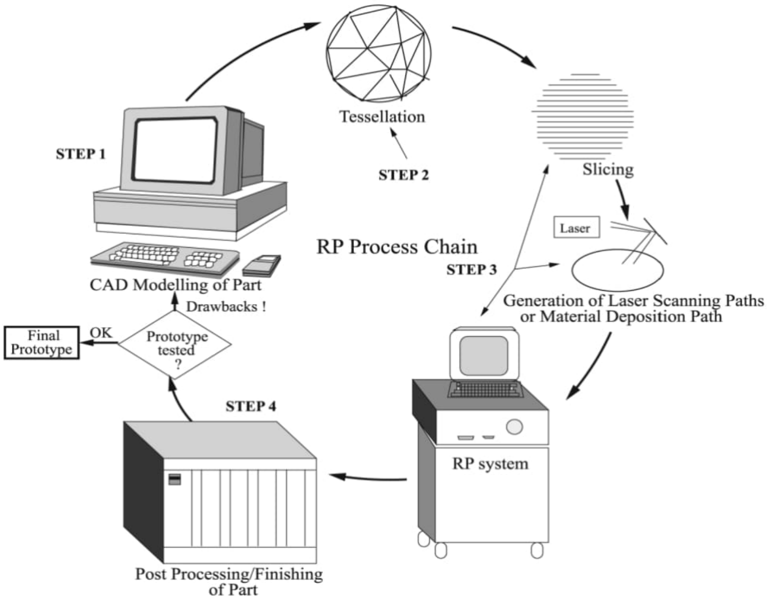


Figure 2.1: AM fabrication pipeline [22]

cess of building supports can be divided into two steps. First, the corresponding areas and surfaces must be identified and then the actions for constructing such supports can be applied. In order to recognize these regions, Kirschman et al. [41] analyzed the slopes of the mesh faces. Some others proposed methods using two consecutive sliced layers and a Boolean comparison to detect the exposed regions [42, 43]. Once these areas are discovered, the support structures can be produced considering a few factors including the use of material and print time. For instance, [44, 45] provided methods for FDM type of 3D printers that decrease the volume of support structures for greater efficiency. Another concern which has an influence on surface quality and cost is the removal of supports. They must be taken away and there should be very low amount of residuals to reduce the post-processing efforts. Preideman-Brosch [46] proposed soluble material for support structures to alleviate these leftovers. Hildreth et al. [47] inspired by this approach and offered a similar technique in the printing of stainless steel. An alternative algorithm was offered by Zhang et al. [36] to adjust the direction of the parts such that the supports attach to the parts with their least possible sections. This eases the removal of the support structures. One more significant parameter of the fabrication pipeline is slicing. The algorithm utilized for slicing must give precise closed contours for each layer and should be executed in the shortest possible time. Some algorithms are slicing the model uniformly and some others are doing it in an adaptive manner, meaning that they are capable of modifying the layer height throughout the part [48, 49]. The adaptive methods can reduce the building time effectively, but they are not capable of dealing with complications within a layer which may result in the staircase effect. For this issue, Tyberg-Bohn [50] offered a locally adaptive slicing algorithm in which the 3D model is split into segments and each part is sliced separately. Further studies employed this technique to divide a part into interior and exterior segments. They sliced them independently in a way that the interior portions utilize thicker layers since they are not visible when the part is fabricated [51, 52]. Another impressible case of study regarding the slicing is about the type of input geometry. Whether the geometry is presented by triangular meshes and ray representations (ray-reps) which is referred to indirect slicing, or using the original CAD model without any conversion to slice directly (direct slicing). Slicing of the triangle meshes is known as the most common technique and is of two strategies. The first one is focused on each triangle individually and detects every slicing plane which crosses it [53–55]. The second algorithm discovers all the triangles located in each slicing plane and the contour of that is created from their intersections [56–58]. Ray-reps is a method for representing solid geometries which were developed earlier by Hook [59] and utilized in many studies to acquire the sliced layers of a 3D model [60–63]. In addition to indirect slicing, some works dedicated for direct slicing and implicit slicing (lately) which are explained in Sections 2.4 and 2.5, respectively.

After the model is sliced, tool-path for each layer must be generated. This operation has also a great impact on the fabrication cost and surface quality. Therefore, providing an effective algorithm to fill up each layer requires a considerable attention in terms of the path continuity, geometry, type of pattern and its performance. Zhao et al. [64] tried to come up with a strategy to alleviate the number of disconnections in an entire layer in FDM process. Jin et al. [65] focused on the geometry and explained how sharp corners can increase the execution time. They provided an optimization approach to smooth the geometry of the tool-paths without losing the accuracy. Many other approaches attempt to enhance the performance of the fabrication process by reducing the non-printing time of the nozzle in FDM machines [66–70]. Some other studies conducted to target specific applications. For instance, the pipeline for having multiple-materials in a fabricated object was studied in several cases [71–73].

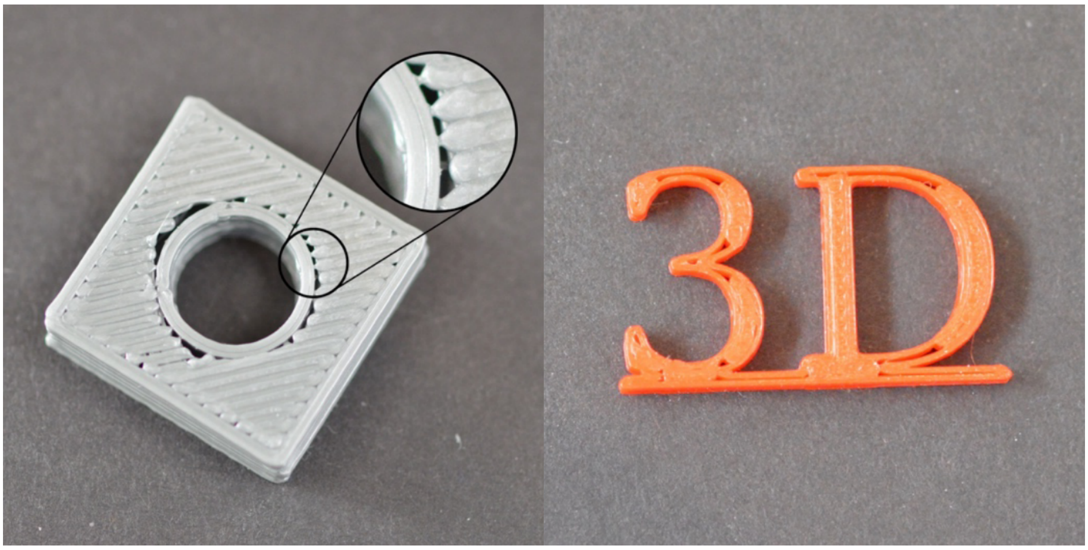
2.4 Direct Slicing

In the conventional approach of AM processes, STL is widely used for representing the designed models. This approach has some drawbacks that motivated the researchers to turn into direct slicing. The first reason is about the precision of the STL file which greatly depends on the number of triangles. Therefore, if high accuracy is required, then a wide storage room must be provided. Another motivation is due to the loss of properties from the conversion and also some slicing failures may occur because of the patch errors. Direct slicing was offered as a solution to these dilemmas. In this method, there is no conversion of the CAD model to STL file format and the model is sliced directly. An implementation of direct slicing on a commercial CAD software (Parasolid) was provided by [74] to acknowledge the benefits of direct slicing. Through this technique, it was asserted that the sliced data can be manipulated. Area deviation ratio was utilized by [75] to obtain an adaptive slicing algorithm on AutoCAD software. Here, uniform and adaptive algorithms were compared and the merits of the second method were highlighted. Another study [76] used PowerSHAPE to slice the models directly. As a result, some Bezier curves and lines were established which can be utilized for AM machinery. Further research conducted by Hayasi-Asiabanpour [77], was aimed to generate machine paths. On the base of their techniques, they used a direct slicing algorithm written in Visual Basic programming language and applied inside Autodesk Inventor as a solid modeler. The biggest advantage of this method is that it can be adaptable to different AM processes. Ma et al. [78] employed non-uniform rational B-spline to achieve this purpose. Their study contained a provision of adaptive slicing algorithm with and without a selective hatching strategy. Both methods demonstrated efficient outputs in terms of decreasing the build time while retaining the surface accuracy. Starly et al. [79], first developed a direct slicing method on standard STEP files which are presented by NURBS and then through some test cases they compared the algorithm with the conventional approach utilizing STL files. The results indicated a better accuracy of direct slicing for freeform models and shorter amount of file sizes. In order to combine rapid prototyping with reverse engineering Qiu et al. [80] got benefited from point clouds to propose a direct slicing process. They managed to improve the efficiency of the slicing for complicated shapes, using topological information of the point cloud. The

NURBS-based surfaces are directly employed in Sikder et al. [81] research to present an adaptive slicing algorithm. The intention of this study was to make the slicing process more efficient by optimizing the texture errors. In another work, Sasaki et al. [82] put forward an adaptive slicing approach of the geometric models which are formed by trivariate B-spline functions. Recently Feng et al. [83] took advantage of T-spline surfaces to develop a direct slicing algorithm. In accordance with this study, although the presented method expressed great deals of usefulness for free-form surfaces, it is not efficient enough for models with regular shapes.

2.5 Implicit Slicing

In today's AM industries, most of the commercial slicing algorithms create tool-paths explicitly without considering the geometrical attributes of the parts including sharp corners, thin walls and round profiles. As a result of these properties, the final part is fabricated by having some serious voids inside which may yield to fractures and failures.



(a) (b)

Figure 2.2: The voids caused by a) Round profiles and b) Thin walls, [84]

This issue was the basis of a new methodology named as implicit slicing in which the functional tool-path patterns are specified for each particular sliced layer based on their geometries. The purpose is to rectify or eliminate the aforementioned voids.

In addition, some studies also considered the mechanical properties when generating these patterns. For instance, Adams-Turner [85] performed a tensile test on some specimens with different infill patterns to highlight their influence on the mechanical properties. Steuben et al. [86] provided an implicit slicing method in terms of evaluating tool-paths for each layer. They further examined the effect of infill patterns on the performance of different models regarding their stress and strain distributions and compared them with the conventional explicit methods. The final tests of their study indicated a great improvement in terms of structural analysis of the specimens.
Although the implicit slicing brings great advantages to AM machinery, there is still room for progress. Since this method utilizes mathematical functions to model the artifacts, currently it is not applicable for organics parts having complicated geometric features.

2.6 Curve Offset

The parallel of a curve, also known as a curve offset in computer-aided design, is defined as locating a curve at a constant distance from the basis curve with any shape and it is one of the major concepts in geometry with many applications in different areas especially in mechanical engineering. There are three domains which get benefits mostly from curve offsets including tool-path patterns in pocketing and motion planning in robotics as well as additive manufacturing (figure 2.3). One of the main concerns in the field of robotics is motion planning, it means dividing the movement trajectory into separated tracks so that it satisfies constraints which are represented as obstacles. In order to plan the route, offsets of the obstacles in workspace must be carried out to avoid the risk of serious injury from the impact between the robot and these barriers. A robot path needs to be smooth with a shortest possible way to its destination, hence a considerable amount of studies was conducted to present various approaches for this problem [87]. In pocket milling, which is a machining process, there is a cutting tool for removing material from workpiece according to a specific path. Whether it is a rough operation or a finishing one, there are some tool-path patterns like the direction-parallel path or contour-parallel path which utilize offset curve algorithms. These patterns are selected based on the workpiece geometry or other properties of the machining process. but they extremely affect the surface quality and cost of the machining operation. Therefore, choosing the proper one may be slightly troublesome. Additive manufacturing (AM) which lately becomes an explicit requirement in manufacturing industry utilizes curve offsets widely in its processes. The process basically begins by defining the shape of a model with the help of CAD software. Based on the analysis of this geometrical information the process planning will be carried out in order to convert CAD representation of the desired component into a finished part. Process planning is composed of four principal parts including, defining the orientation, support structures, slicing and tool-path generation. In process planning of an AM process, a reduction in product launch time to the final part was always researchers concern intensively. Moreover, some other parameters including geometric accuracy, build efficiency etc. need to be considered. The major application of curve offsets in additive manufacturing processes is for generating tool-path patterns. Path planning is not limited only to AM but also utilized in NC machining, generating hollow shell and robotics. Therefore, some of those path patterns are applicable for path planning in AM processes. For instance, the most desirable patterns, utilized in the FDM process are contour and parallel zigzag paths which are employed distinctly or together as hybrid path[88]. Here, the focus is on the contour paths to demonstrate the importance of offset curve and compare the capability of each algorithm. Since tool-path affects the manufacturing efficiency and the accuracy of the finishing part, choosing a proper offsetting algorithm is imperative. Although these algorithms have been extensively studied recently, there is still a lot of room for progress.

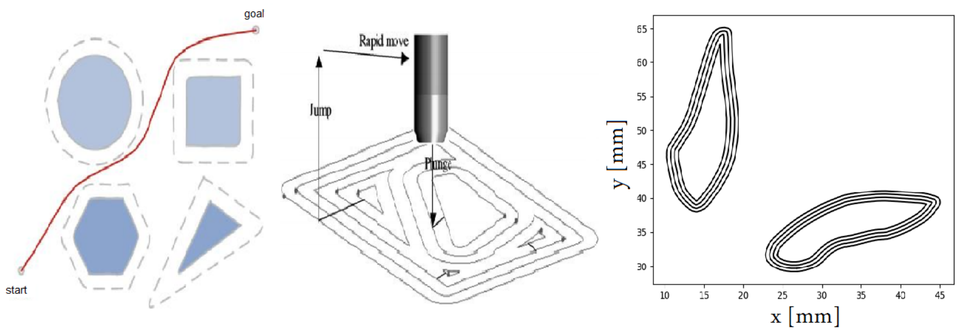


Figure 2.3: From left to right: Curve offsets in a) path planning of Robotics, b) tool-path generation of Pocket milling, c) tool-path patterns of Additive Manufacturing

E. Lee [89] presented a new toolpath technique for finishing processes in high-speed machining and he mentioned it as a spiral topology toolpath. In this method, he utilized an approximation to compute curve offsets. Here the base curve is represented in a parametric form as C(t) and the corresponding curve offsets are also evaluated parametrically as,

$$C^0(t) = C(t) + l(t)N(t) \tag{2.1}$$

where $N(t)$ is unit normal vector and at t the offset distance can be evaluated as $l(t)$. The output of Equation 2.1 is a set of points which can be interpolated as line sectors. Some of these offset lines may intersect with each other and they should be eliminated from the set, therefore, there must be an algorithm to recognize these invalid segments and the solution is to check the direction of each offset loop based on the direction of the base. If the base curve has a counter-clockwise direction then all the clockwise loops must be removed. Another method used for generating contour-parallel path conducted by Zhiwei et al.[90] and it is very suitable for creating offsets of curves with islands. In order to have a result, three steps must be applied to the given data. These steps are, the islands bridging, creation of offset curves and removing of invalid loops. Here the input has two distinct sets which represents outer profile and islands. In the first process, Delaunay triangulation is utilized to connect islands together and also with the outer profile so that the base curve presents uniformly as a single set. In the second step, in order to generate the offset curves, two algorithms are developed. The first one uses three successive points of the base and finds the bisector of the corresponding two lines. Then by having the offset distance, offset point is obtained. But there are some situations that bisectors intersect with each other, then the second algorithm needs to be applied. It defines a stuck circle with a radius of r (offset distance) and for four consecutive points, the algorithm checks whether this circle exists or not. The center of the stuck circle is defined as an offset point. Finally, in the last step, all of the self-intersections are recognized and the invalid loops are removed. Because of the connections between islands and outer profile, there are some areas left without any offset curves. So this can be one of the disadvantages of this method. On the other hand, since all the stages in this method show a linear trend in time with respect to the number of base points, therefore it has a linear time complexity. A pairwise offset technique was provided by Choi-Park[91] which utilizes closed 2D

Point Sequence curves (PS-curves) as input. In this method, a pairwise interference detection (PWID) test is applied before constructing the offset curves. In the test, all the invalid loops which will be made by self-intersection of offset curves, are detected and removed from PS curve. Then the raw offset curve is created and local interfering ranges and pair-wise self intersections are detected and removed. Finally, the remaining parts of the raw offset curve form the result offset curve. The biggest advantage of this algorithm can be the linear time complexity $O(n)$ and the drawback is that it cannot be applied to base curves with islands. Another approach obtained by Lee et al.[92] creates offset curves in four steps. First, according to the bisectors of each two lines of the base curve, the offset points are evaluated. The direction and position of each offset line are checked to define its validity. After that, all of the offset lines gathered to construct the raw offset curve without local invalid loops. The radius of the raw offset curves is checked and global invalid loops are eliminated so that in the end, the final offset curves remained. Yang et al. [93] introduced an effective offset algorithm for generating tool-path in additive manufacturing processes. The process begins with finding the direction of the base boundary. Then for every three vertices on the base, the inward and outward offset points are computed using Equations 2.2 and 2.3 [93],

$$x = \frac{(x_{i+1} - x_i)r}{x_i y_{i+1} - y_i^2} \quad , y = \frac{(y_{i+1} - y_i)r}{x_i y_{i+1} - y_i^2} \tag{2.2}$$

$$x = -\frac{(x_{i+1} - x_i)r}{x_i y_{i+1} - y_i^2} \quad , y = -\frac{(y_{i+1} - y_i)r}{x_i y_{i+1} - y_i^2} \tag{2.3}$$

After generating the raw offset curve, self-intersections must be identified and eliminated from the set. The algorithm checks the curve offset, line by line to see if there is an intersection between non-adjacent lines and provide a set of intersection points. The order of these points must preserve according to the initial direction of the base curve. From induction, it can be proved that the amount of closed polygon obtained from this raw offset curve is $m + 1$, in which m is the number of self-intersection points. By traveling in the same direction as the base, if a closed loop is located on the right side of the joint (self-intersection point) then the corresponding polygon

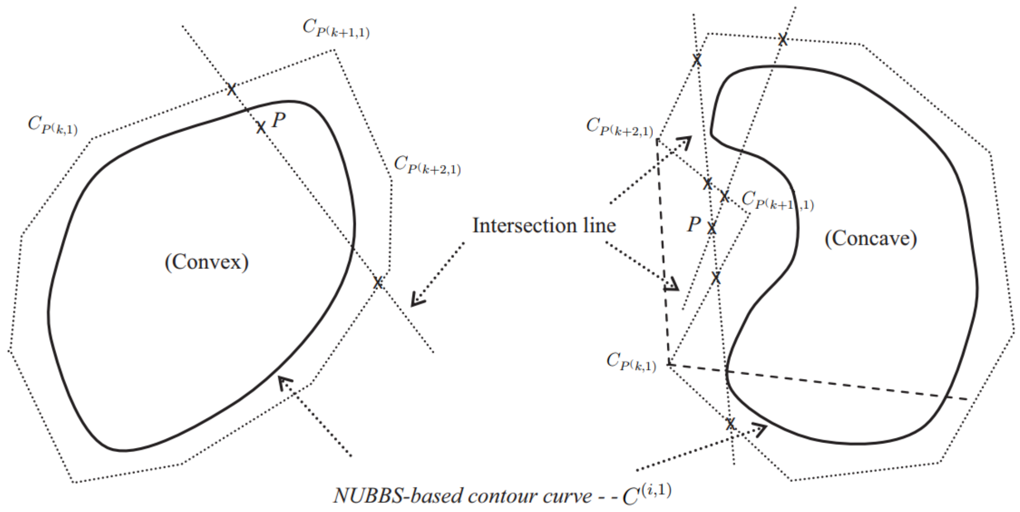


Figure 2.4: Two types of closed curves [94]

must be removed. Another algorithm provided by Jin et al.[94] to create curve offsets is using NURBS (Non-Uniform Rational B-Spline). In this method, the boundary curves need to be categorized into two types firstly. Based on the box created by connecting the control points of the boundary curve and passing some intersection lines the type of the curve can be identified. As it is evident in Figure 2.4, if the line intersects with more than two points from the box, then the boundary is recognized as the concave curve. Otherwise, it is considered to be convex. For Type 1, a center point (D) is utilized for obtaining the offset curves in k^{th} layer through Equation 2.4 [94]. In this equation, k represents the number of layers and j is the number of offset curves in a single layer.

$$C_{P(k,j)} = D + \alpha(C_{P(k,1)} - D) \tag{2.4}$$

Here, α is a factor for producing the new control points which is related to the offset radius and the amount of overlapping between the two paths (Figure 2.5).

If the curve is in Type 2 (concave), it must be separated into multiple convex curves. Thus, the concave points are identified and they will be connected to other control points to turn the curve into two or more convex curves with their own center points (D_i). Then for each curve, the algorithm used in Type 1 will be applied to obtain the offsets.

In another study presented by Dolen-Yaman [95], morphological operations were em-

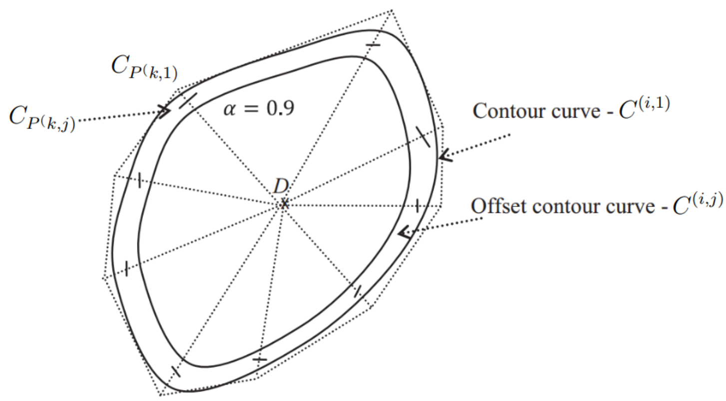


Figure 2.5: Generating offset curve for Type 1 [94]

ployed to offer four offset techniques with the aim of generating tool-paths for 2.5D machining. The first one utilized a tracing system on binary images to create offsets and in the result, it was known as its high memory cost. The second method intended to solve this issue by using boundary sets, but instead it demonstrated inefficient outcomes in terms of time complexity. Therefore, the third technique was proposed to overcome this problem and it successfully lowered the time complexity with the help of a grid search. Finally, the last one produces offset curves using polygon operations. These methods evaluated in terms of time complexity, memory complexity and the accuracy of their geometry and in conclusion, the third one which is known as Improved Morphological Operations on Boundary Sets (IMOBS), found to be more eligible to be used in CNC machining applications. Furthermore, they introduced an additional approach in [96] named as Advanced Morphological Operations on Boundary Sets (AMOBS) which was aimed to enhance the geometrical accuracy of the IMOBS using a gradient algorithm.

2.7 Digital Light Processing

DLP is a mask projection process which depends on the functionality of Digital Micromirror Device (DMD) systems (Dudley et al. [97]). The DMD was then equipped with electrical supports to create DLP technology and used in AM. This technology utilizes bitmap images as a tool to solidify the UV photopolymerised resin for manufacturing parts. This resin is reactive to light and they can be cured using a light source like a projector. Figure 2.6 indicates a comprehensive view of the entire process and its components. At first, a 3D model is sliced into layers and bitmap images are created from these slices. Then the images are sent to the machine and ready to be projected. In this process, resin is kept in a vat and over the vat a build table is situated to hold the fabricated parts. In order to print each layer, the build table is located away from the vat surface with a distance of one layer thickness so that the resin can fill this gap. Afterwards, the corresponding image is projected from the bottom to the table which hardens the material. In some commercial machines, recoating is required to prepare the surface for the next layer. Then, the table rises one layer, resin is exposed to the light again and the process continues consecutively to the end. This method is known as a fast process owing to the fact that the entire layer is created at once. Another advantage is its remarkable resolution and the accuracy of the end products. This operation requires support structures and in some cases post-curing.

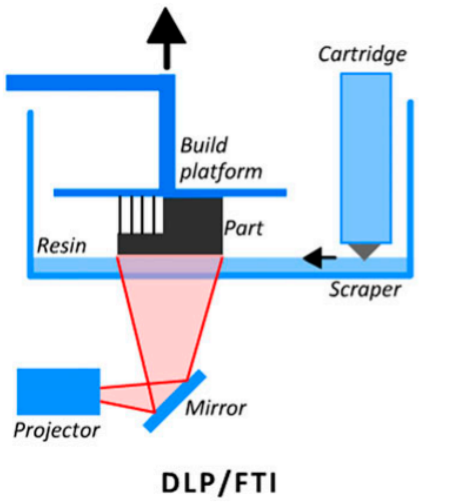


Figure 2.6: DLP Process [98]

2.8 Fused Deposition Modeling

FDM is the most common AM process which is based on material extrusion systems. In extrusion-based operations, the material is stored in a reservoir and fed to the nozzle with a controllable speed which results in a continuous flow of materials onto the table. Since the extruded material is in a semisolid state, they cohere with the previous layer immediately after leaving the nozzle. By keeping the temperature of the nozzle at a constant level, the state of the material can be controlled. Each layer can be produced by having a machine able to scan the horizontal plane. Then, the build plate brings down for one layer thickness so that the next layer can be manufactured. The cross-sections are made successively until the entire part is fabricated (Figure 2.7). In this process, ABS and PLA are the most popular materials, but recently some other materials are used with this operation for the special purposes. Some studies even employed soluble materials for support structures in FDM which was mentioned in the previous section.

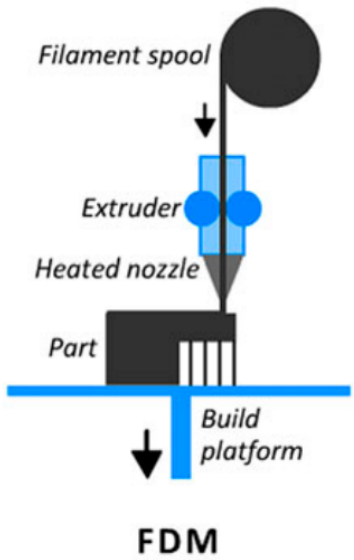


Figure 2.7: FDM process [98]

The quality of the final parts highly depends on the material properties and the building parameters. Ahn et al. [99] analyzed some of these parameters in the FDM fabrication process. This method has low resolution in comparison to other processes. Therefore, it may need some post processing. Another drawback is that it is not a fast AM process.

CHAPTER 3

LIST PROCESSING LANGUAGE AND ITS PIPELINE

3.1 Introduction

As was mentioned in Chapter 1 of the dissertation, the conventional pipeline of AM processes has some limitations and in order to make these drawbacks less severe, LIPRO is provided which was proposed earlier by [2, 3]. LIPRO is a new command generation paradigm, aimed to produce motion trajectories for various production machinery. The main data employed for generating these trajectories are base curves and as it is illustrated in Figure 3.1, they can be provided based on the imported input data by using three different procedures. Sometimes the information about the CAD models are provided using 3D scanning or from slicing. Since the size of such data might be huge, it is more sensible to compress them before they can be transferred into the LIPRO. Therefore, a decompression function is provided to access the basis curve of a particular layer and represent it in a readable format for further operations. Next method employs a function to create simple polygons. Here, in order to define the design artifacts, some parameters related to its geometry are required. Furthermore, if a CAD model representation or an STL file format is given, the LIPRO is also capable of preparing the base curve of each layer using a slicing function.

After each basis curve is provided, it can be reshaped to be a more complicated curve by utilizing two functions including polygon operations and homogeneous transformation. Then, the pathway of the fabrication for each layer is generated by using curve offsets. If there are more than two separate basis curves, a path sequencing function is executed on them to speed up the fabrication process. In the final stage, commands for each layer based on the existing paths are generated. However, the

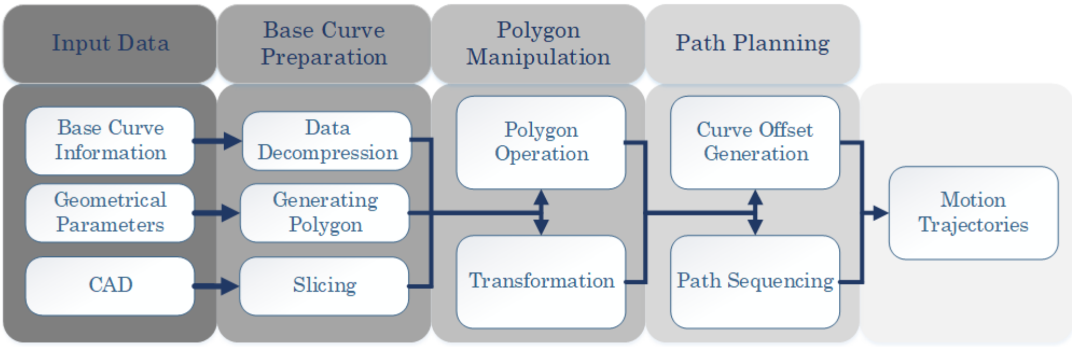


Figure 3.1: Structure of LIPRO

same commands are not operational for both FDM and DLP printers. Since FDM is a point-wise process, its operative information must contain the details about the coordinates of the points so that it can be utilized for generating the corresponding G-code commands. On the other hand, DLP is layer-wised and its path related information is obtained in the form of a bitmap image. In addition, G-codes are not applicable for DLP printers and separate commands are introduced to control the fabrication process. As a result, a decision must be made at the beginning of the process so that the LIPRO can comprehend the correct procedure to follow. Table 3.1 is representing the process of printing Stanford Bunny with a DLP printer. Also the main functions written in Python for both DLP and FDM printers are provided in Table 3.2 and 3.3 respectively.

Table 3.1: The process of choosing printer

```python
import LIPRO
### Define your printer ###
Printer = 'DLP'
### Load Data and parameters ###
data = load('StanfordBunny')
### Printing ###
if Printer == 'FDM':
    LIPRO.FDMmain(data)
elif Printer == 'DLP':
    LIPRO.DLPmain(data)
else:
    print('Your printer is not defined by LIPRO!')
```

Table 3.2: DLP main function

```
import LIPRO as lp
dc, step, s = lp.InitialS()          ### The initial measures
lp.wait(s)                           ### Wait for operator
lp.DLPPrint(s,dc,step,data)          ### Start the print
lp.FinalS(dc, step, s)               ### The end settings
```

Table 3.3: FDM main function

```
import LIPRO as lp
import time
### Connection to Arduino
s = lp.connection('/dev/ttyACM0', 250000, 5)
time.sleep(5)
lp.strt_confg(s)     ### Initiate the start settings
lp.FDMPrint(s,data)
lp.end_confg(s)      ### Initiate end settings
lp.closing(s)        ### Terminate the connection
```

3.2 Generating Polygons

In accordance with the designed artifacts, sometimes the 3D model is not complicated and can be created using the provided functions in the LIPRO (*gpc*). With the help of this function, various types of polygons can be generated. Basically, this operation can divide a circle into equal segments and delivers the coordinates of the points representing these segments. Figure 3.2 presents this operation by creating a quadrilateral shape. Therefore, by putting the number of points (number of sides), radius of the polygon and the starting angle of the first point, polygons with different types can be generated. Figure 3.3 illustrates some of the polygons created by this function.

3.3 Data Decompression

In some cases, along with information regarding layer numbers, layer thickness and other fabrication parameters, base curve (number of points) data must be loaded into the LIPRO. The size of the data might be large and it results in some issues in terms

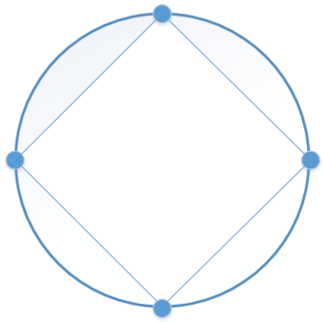


Figure 3.2: The process of generating polygon

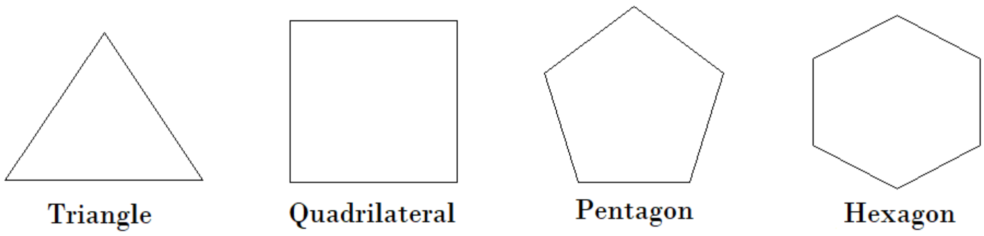


Figure 3.3: Different types of polygons generated by gpc

of the use of memory and data transfer. Therefore, it is more efficient to store the compressed version of the original data in the LIPRO and use a decompression operation when it is required. There are some approaches including zip and gzip, but like it was offered in [3], the ΔY method can be more effective for motion trajectories.

3.4 Slicing

To enhance the capability of the LIPRO, a slicing operation is provided so that the CAD models with the file format of STL can be fabricated using this method. The main concept of this slicing method is that the algorithm searches for all the triangles that are passing through each slicing plane and it finds the intersection points. This process employs three points from each facet, generates three lines from these coordinates and checks if the Z value for the slice plan is in the range of the line. Then it obtains a 2D coordinates of a point located on the line based on Z value. After the intersection points are found, since they may not be in a correct sequence, additional operation for ordering the points are required. A summary of this approach is described in Algorithm 3.1. In order to find the triangles that are in contact with the slicing plane, since the number of triangles is extremely large, a grid search must be employed to speed up this process. Also, this special function applied to ordering part to handle all of the intersection points in a shorter time. For better comprehension, a sample (Stanford Bunny) is sliced using this method with a layer thickness of 0.06 mm and the layer with the number 1166 is demonstrated in Figure 3.4.

Algorithm 3.1

```
procedure SLICING(M)
    Input: STL file format representing triangular facets
    Input: Parameter Z as slice plane
    Output: List of coordinates (Q) representing the base curve
    Find all triangular facets near slicing plane with a constant search radius and store them in a list as T
    for each facet in T do
        Get the vertices of the facet and store three lines from those vertices in L
        for line in L do
            Check if there is any intersection between the line and the slicing plane
            Store these intersections into Q
    Order the intersection points inside Q
```

3.5 Polygon Operations

This is another function that enables the LIPRO create more complex 2D curves from simple polygons using a polygon clipping technique. The operations provided in this method are composed of union, difference and intersection. Since in this study Python programming language is chosen to represent the LIPRO, many Python libraries capable of performing these operations can be employed including Clipper and Shapely. A single union operation is illustrated in Figure 3.5.

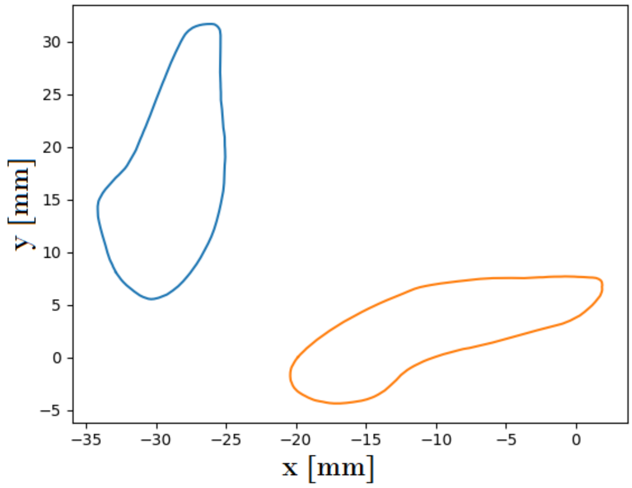
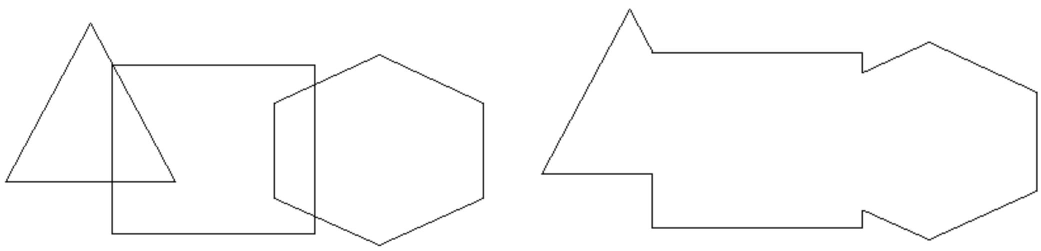


Figure 3.4: A sliced layer of Stanford Bunny



(a) (b)

Figure 3.5: Union operation. a) raw polygons, b) after union applied

3.6 Transformations

Another function utilized for building more sophisticated 2D shapes is described here. It applies a homogeneous transformation on the base curves including translation (t_x and t_y) and rotation (θ). Equation 3.1[3] displays this operation that can both translate and rotate at the same time by taking the amount of translations (t_x and t_y) and the rotation angle (θ). This equation is applied on each points of the base curve with the coordinates of $P_i = (x_i, y_i)$ and it is capable of providing the transformed points as

$P_i' = (x_i', y_i')$.

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{3.1}$$

3.7 Path Sequence

There are some cases that a few models are printed beside each other simultaneously. The algorithm is generating the motion trajectories of these closed curves for each layer. In order to reduce the time and make the printing process more efficient, the order of these closed curves in each layer must be optimized. Therefore, the sequence of multiple paths are analyzed within this function based on the shortest distance between them. To illuminate this concept, Algorithm 3.2 is clarifying this approach and further details are explained in [3].

Algorithm 3.2

```
procedure PATH SEQUENCING(S)
    Input: A list of basis curves (S)
    Output: A list of ordered curves (S')
    Let S' be a set of basis curves ordered
    Let P1 be the first curve of the set S
    for curve in S do
        Check euclidean distance of points in P1 with points in curve
        Add P1 as the next curve to S'
        Set P1 to the closest curve
```

3.8 Curve Offset Generation

In this Section, the two offset algorithms conducted by Yaman and Dolen [96] are studied. These algorithms are unique because of their capability for offsetting both self-intersection curves and islands. Here, these methods are implemented in Python and the capability and the performance of each are compared with another offset method which exists in one of the Python libraries. The two methods were briefly introduced in the previous Chapter and they are known as IMOBS and AMOBS. Their structure is almost the same and they are basically created by three common steps including creation of boundary sets (CBS), invalid points removal and creation of curve offsets (CCO). In addition, IMOBS has an extra phase for merging the fragmented offsets (MFO) at the end of CCO. In the following context, the details for each method are explained.

3.8.1 Improved Morphological Operations on Boundary Sets (IMOBS)

In this method, the initial step (CBS) has two equations for generating the offset points. One of them is based on the curvature of the initial curve to create only two boundary points for each base point and the other is to make a set of circular boundary points around each base point. The decision between these two methods will be done by a user-defined parameter. For instance, if the curvature between three points in the base is more than the threshold, then the first method is utilized to create offset points otherwise, the circular method is carried out. After creating the raw offset points, those invalid points must be eliminated with the help of a hash table localizing the search for each boundary point. In order to determine whether the curve offset points are valid, the method checks the point with all the base points close to it. A circular search with a radius of 2r is performed to find neighboring base points. Next step is to order the final sets (CCO) according to the direction of the base curve. To order all the disjoint sets, IMOBS goes through the boundary points and for each point, the nearest one is found. In the end, the remaining disjoint sets need to be merged together (MFO) to create the sets of external and internal offset curves. Figure 3.6 briefly displays each section along with its outcome.

3.8.2 Advanced Morphological Operations on Boundary Sets (AMOBS)

The second part of IMOBS (CBS) can participate in AMOBS only in specific cases if the distance between the particular point and its previous point is less than the

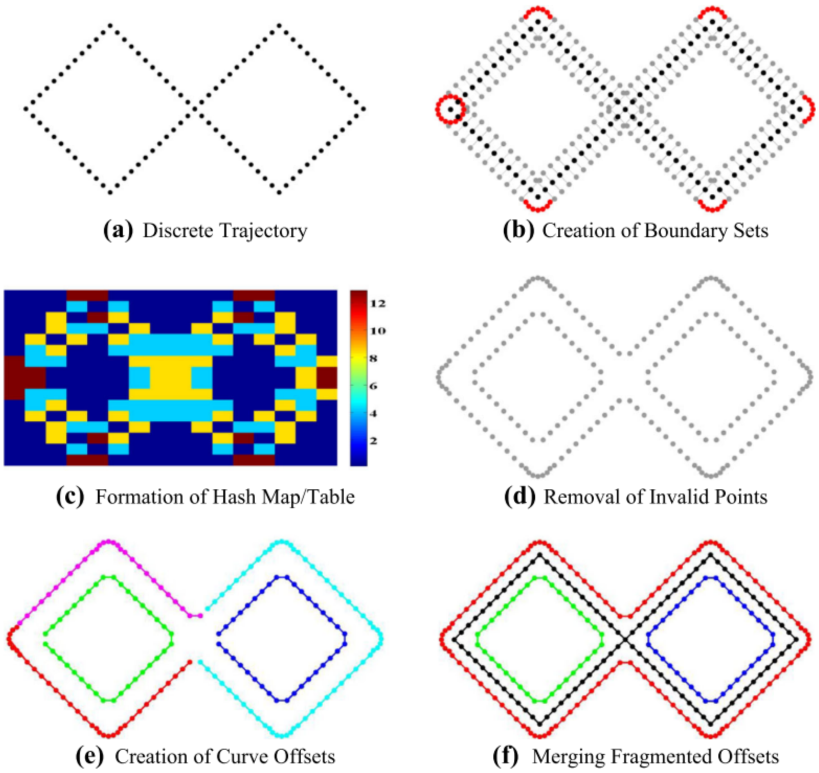


Figure 3.6: Summary of IMOBS [96]

parameter δ [96]. The CBS part of AMOBS relies on the gradient vectors around each base point (Figure 3.7). After obtaining the boundary points, invalid points must be defined. For this purpose, the algorithm checks the Euclidian distance between each base point and the boundary points around it. If the distance is less than r, then it will be known as invalid and will be removed from the set. To speed up the process, a grid search is applied to localize the search for finding the invalid points. The CCO of AMOBS also uses the same grid search in the previous phase to find the order of offset points. Here, along with examining the Euclidean distance between vertices, a gradient algorithm of the offset curve is added with a weight factor to overcome some of the errors occurring in the sharp corners and increase the accuracy.

3.8.3 Shapely

Shapely is library written in Python which basically utilized in geographic information systems (GIS) and has some functions for manipulating the geometrical features

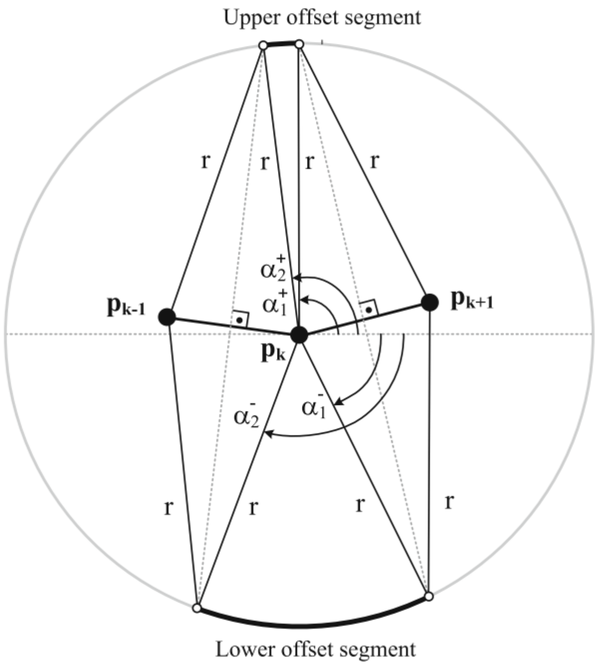


Figure 3.7: Using gradient vector around each base to create boundary points [96]

in a plane. There is a buffer function for offsetting a base curve which is employed in this study. This function can be applied not only to polylines but also to points and polygons. The buffer technique [100] goes through each base point and creates buffer offset vertices around them. Then as a compound of circular arcs and lines centered on the points of the convex hull, the buffer regions are formed. From this region, the exterior part is the final offset curve which can be used for the purpose of this study.

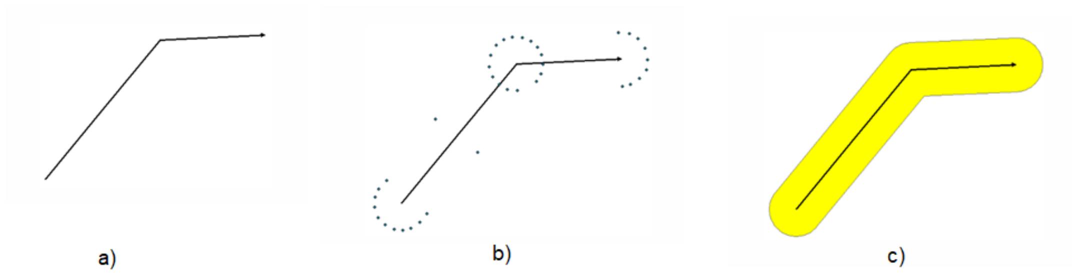


Figure 3.8: The process of Buffer around a line. a) base line, b) offset points generated, c) buffer derived [101]

3.8.4 Parameter Adjustment

Through the implementation of IMOBS and AMOBS, it is observed that there are some parameters affecting the running time and accuracy of the final curve offset. Hence, in order to maximize the performance of these two algorithms, some measures need to be carried out for adjusting the parameters. The two influential parameters affecting the outcomes of IMOBS are the number of points on the base curve (P) and the number of points on a circle in circular approach (N). In addition, there exists a parameter called resolution (ffi) which specifies the method needs to be used for creating boundary points. The following relations can be used for this purpose,

$$P = \frac{L}{\sqrt{8r\epsilon}} \tag{3.2}$$

$$N = \frac{14r\pi}{\delta} \tag{3.3}$$

where ϵ is an error tolerance band and L describes the length of the base curve. In order to find a relation for the resolution (ϵ [0,1]), some tests performed and by considering the accuracy and the time complexity of the outcomes. It is concluded that the most effective way is to pick this number as one since reducing this amount leads to an increment in the number of base points (P) which results in an increase in execution time. In the case of AMOBS, there are two extra parameters which are necessary for gradient algorithm [96], Equation 3.4. First one is the weight factor ($w \in$ [0,1]) and the other is n_Q, which is the number of points required for computing the gradient of the offset curve.

$$U = \min_{m,j} \left\{ \left\| s^*_{m,j} - q \right\|_2 \left[\frac{w}{\delta} + \frac{1-w}{u.(s^*_{m,j} - q)} \right] \right\} \tag{3.4}$$

To find a relation between them, the base curve must be analyzed since the curvature of offset curves follow the same trend in their basis curve. While testing some trajectories, results proved that for noisy curves large number of n_Q with $w < 0.75$ yields to an incorrect order of offset points resulting in errors in the final offset curves (this fact was also mentioned in Yaman and Dolen [96]). Therefore, by having a set of curvatures of all base points and considering worse case, the minimum value of this set must be picked for computing both parameters (Equation 3.5 and 3.6). These minimum values show the locations in which the curve loses its smoothness.

$$n_Q = P(0.0025\alpha^2 + 0.0005) \tag{3.5}$$

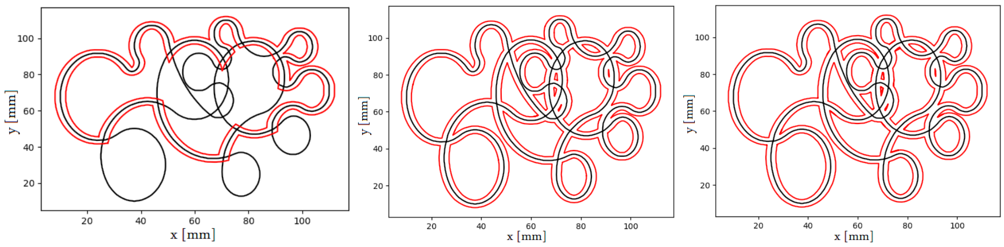
$$w = 0.7\sqrt{1-\alpha^2} + 0.3 \tag{3.6}$$

where α is the minimum angle exists in the curvature of the base curve.

3.8.5 Comparisons and Discussions

To demonstrate the capability of these methods, the same test cases used in [95] are examined. Among these samples, two of them (Doodle and Rabbit) are more controversial since Doodle is presenting a self-intersecting base curve and Rabbit which shows a basis curve with islands. For the Doodle case, an offset radius of $1mm$ and for the Rabbit, $r = 0.5mm$ is utilized. Initially, the three algorithms applied on the Doodle case (Figure 3.9) and the following results achieved. As it is clear in Figure 3.9a, Shapely is not capable of evaluating the offsets of the curve having self-intersections. Although IMOBS (Figure 3.9b) and AMOBS (Figure 3.9c) perform offsetting, their accuracies are not the same. In Figure 3.10a, a closed view of the Doodle case is shown for AMOBS which displays the precision of it. But in the case of IMOBS (Figure 3.10b), there are some fragments in the corners. This is due to the fact that in the CCO of IMOBS, the ordering of the offset points is not carried out well. That is why a gradient algorithm is added to the CCO of AMOBS to overcome this problem. For this model, IMOBS completes the offset curves in $t = 1.29$ seconds, but it takes longer for AMOBS with $t = 4.468$ seconds to fulfill the task. This difference in execution time is again because of the gradient algorithm used in the CCO of AMOBS. Despite its effective performance on the accuracy, since it utilizes a grid search, it increases the running time for a large amount of points in the base curve.

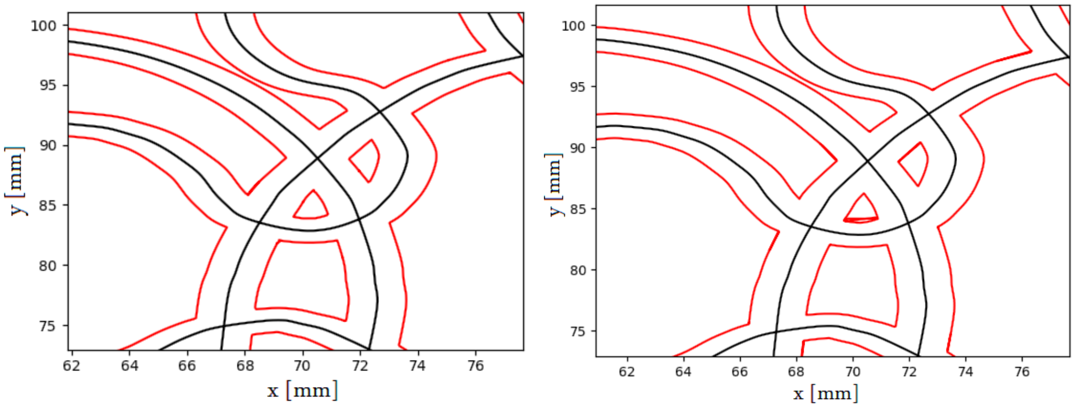
In the case of managing the islands, the Rabbit case is presented in Figure 3.11. The



(a) (b) (c)

Figure 3.9: Test results of the Doodle with (a) Shapely, (b) IMOBS and (c) AMOBS

offset curves in all of these three methods are achievable. In terms of accuracy, in addition to the fact explained in the Doodle case for AMOBS and IMOBS, it seems that Shapely can provide proper offset curves.



(a) (b)

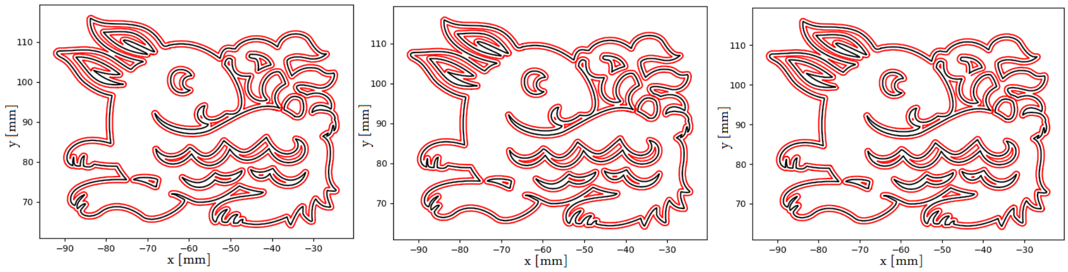
Figure 3.10: Accuracy of offset curves in (a) AMOBS and (b) IMOBS

The running time results are available in Table 3.4 for each test cases. These results indicate a big difference between Shapely and the other two methods in the case of execution time showing the major advantage of this algorithm.

Table 3.4: Execution times (sec) for two test cases

Test cases	**IMOBS**	**AMOBS**	**Shapely**
Doodle	1.29	4.4687	0.0312
Rabbit	0.75	3.5156	0.0312

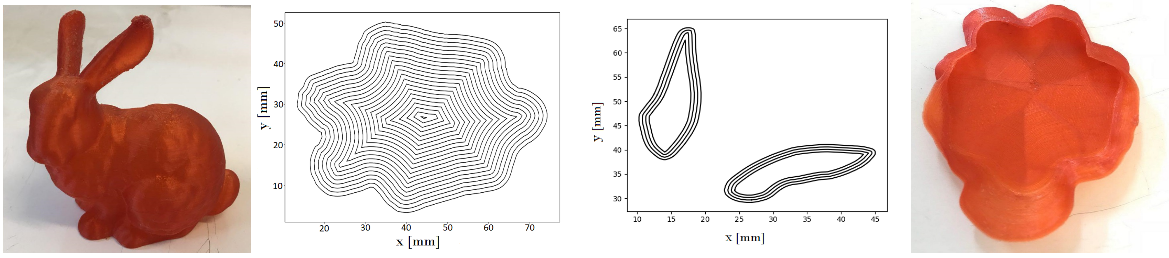
In another experiment, in order to understand the capability of the corresponding techniques to create the tool-path patterns, a model (Bunny) is presented in Figure 3.12. This model provided in STL file format is sliced with a layer thickness of 60



(a) (b) (c)

Figure 3.11: Test results for rabbit in (a) Shapely, (b) IMOBS and (c) AMOBS

µm. For this model, ten layers of the bottom are filled entirely with an offset distance of $0.35mm$ as illustrated in Figure 3.12b and the remaining layers will be created as a wall with three offset curves in each layer (Figure 3.12c). The G-code file is obtained from these paths and this file can be executed on an FDM machine to fabricate the model (Figure 3.12a). The printed bottom layers are also illustrated in Figure 3.12d.



(a) (b) (c) (d)

Figure 3.12: Tool path patterns of the layers. (a) The model Bunny after printing, (b) Paths of a bottom layer, (c) Paths of an upper layer and (d) The bottom layers printed

After implementing the three algorithms, results in Table 3.5 are obtained. As it is observed, there is a huge difference in the first row of the table between Shapely and the other two. That is due to the fact that in Shapely, the buffering technique will simplify curve segments on the basis. It means that Shapely performs the offset process with a lower amount of input data and also in the end, it presents the result in an optimum number of offset points. But in the case of AMOBS and IMOBS, the number of input data must be picked high in order to complete the offset curves and the corresponding data at the end of their algorithm is also high. This is due to the fact that the distribution of the points in the basis curve for IMOBS and AMOBS is the same because their procedure for presenting the initial curve is based on the fixed parameter δ. However for generating the G-codes since the results of IMOBS and AMOBS contains a huge amount of data, an extra operation involved to remove the redundant 2D coordinates. This function is applied on each curve offsets and the amount of deviation for every three consecutive points is evaluated. If this amount does not exceed 1 μm (deviation threshold), the corresponding point recognized as redundant and the program will remove it. Therefore, after simplification on outputs of IMOBS and AMOBS, the G-codes file sizes are provided in Table 3.5 to avoid having issues in terms of memory usage and data transmission.

Table 3.5: Results of three methods for bunny

Test cases	**IMOBS**	**AMOBS**	**Shapely**
Running time (sec)	750.28	1135.62	8.6562
G-code file size (MB)	36	42	67.8

The comparison between the offset methods through some test cases demonstrate the following facts.

- Shapely cannot be used for the base curves having self-intersections.
- All of the algorithms are capable of handling the base curves with islands. In the case of accuracy, AMOBS shows great preciseness and also Shapely can bring a proper offset curve but since it simplifies the base curve, the accuracy would not be that great.
- Using a huge amount of data in IMOBS and AMOBS increases the running time.
- Shapely can perform the offsetting in a very short amount of time, which makes it highly efficient for huge tasks like generating tool-path patterns for AM.

In order to make IMOBS and AMOBS applicable for generating tool-path patterns, some modifications are needed. Since they need to adjust some parameters for each test case, a new algorithm should be developed having only one parameter (P) to be specified. This algorithm must be functional for the base curves with few amount of input points. Hence, a densification method is required for presenting the base curve accurately, but in a lower amount of points. Due to these results, in the following Chapters, shapely is employed to fulfill the task of generating offsets.

3.9 Controller Board

In order to implement this new paradigm, a single-board computer called Raspberry Pi 3 is utilized (Figure 3.13). The reasons behind choosing this board are its high capability of computing, high performance, convenient connectivity, proper power management and low cost. One of the main advantages of this board is having various types of connectivity including 4 USB ports, HDMI, bluetooth and built-in wireless. These provide easy communication with the 3D printers. Additionally, the row of general-purpose input/output (GPIO) pins near the top edge make it capable to be connectted with other electronic devices like buttons and LEDs. Some extra information about its hardware and physical specifications are availablle in [102].

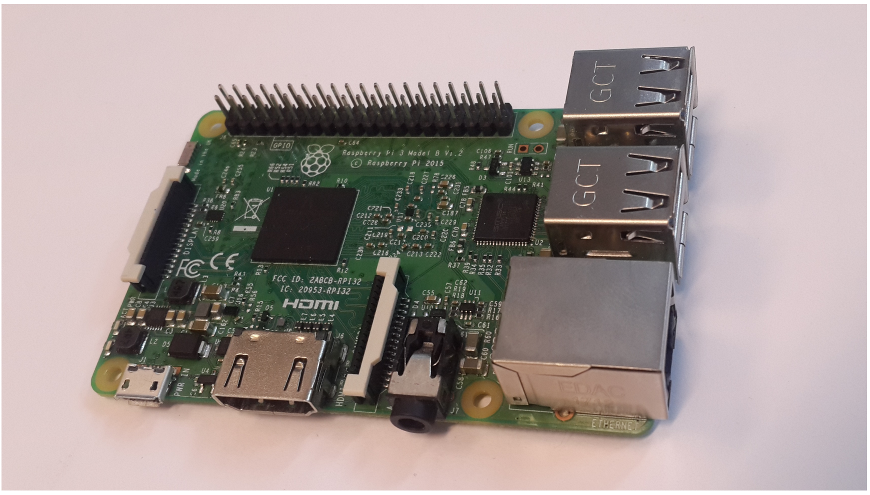


Figure 3.13: Raspberry Pi 3 model B

CHAPTER 4

IMPLEMENTATION OF THE LIPRO ON AN FDM PRINTER

4.1 Introduction

In this Chapter of the thesis, instructions of the LIPRO for printing 3D models on an FDM type of printer are presented. Before demonstrating the procedure, important machine specifications such as material descriptions, extruder parameters, etc. must be denoted. The two most common materials used in FDM process are Acrylonitrile Butadiene Styrene (ABS) and PolyLactic Acid (PLA). A comparison is provided in Table 4.1 and more details can be found in [103].

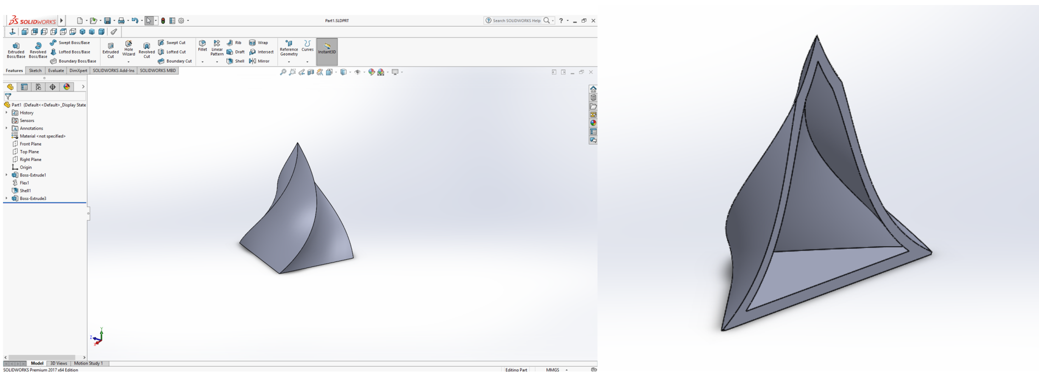
Table 4.1: Material properties of ABS and PLA [103]

Properties	**ABS**	**PLA**
Tensile Strength	27 MPa	37 MPa
Elongation	3.5 - 50%	6%
Flexural Modulus	2.1 - 7.6 GPa	4 GPa
Density	1.0 - 1.4 g/cm3	1.3 g/cm3
Melting Point	N/A (amorphous)	173 °C
Biodegradable	No	Yes, under the correct conditions
Glass Transition Temperature	105 °C	60 °C
Common Products	LEGO, electronic housings	Cups, plastic bags, cutlery

For the implementation of the LIPRO, Ultimaker 2 Go is utilized as a commercial FDM machine with specifications displayed in [104]. The only type of filament can be used on this machine is PLA. Arduino Mega 2560 is the main board responsible for interpreting G-codes to machine instructions and controlling the details of the fabrication. Since in this implementation LIPRO is performed on Raspberry Pi 3, a serial communication must be carried out between the Raspberry Pi and the Arduino Mega. The following sections denote the details of the fabrication process both in conventional way and the LIPRO. At the end, fabricated parts are presented and the fundamental differences of the LIPRO and the conventional approach are elaborated.

4.2 Main Steps of the Conventional FDM Printing

The initial step to fabricate by this method is to design an artifact in CAD software. Figure 4.1a is displaying this task performed in solid modeling CAD software named as SolidWorks and the cross section of this 3D model is presented in Figure 4.1b. In order to design this model in SolidWorks, firstly a simple pyramid is created using the commands called Extrude and Draft. Then, the resulting shape is twisted based on a desired angle with the help of the command Flex. Afterward, Shell is utilized to obtain a void inside the model and finally, the bottom face is filled using an Extrude command.



(a) (b)

Figure 4.1: Design in a CAD software

In order to import this model to the corresponding CAM software, it must first be converted to the STL file format. This task is carried out by SolidWorks itself, simply by saving as an STL file. Then Cura, which is an open source 3D printer slicing application, is utilized to read the STL file. Since the Ultimaker manufacturing company is using this application exclusively, the additional prefabrication operations can simply be set by this application. The most important task is to slice the model, therefore the slicing layer thickness must be defined. In conjunction with slicing, the tool-path patterns are also generated within the same software. Figure 4.2 shows the Graphical User Interface (GUI) of this CAM software. In addition to aforementioned tasks,

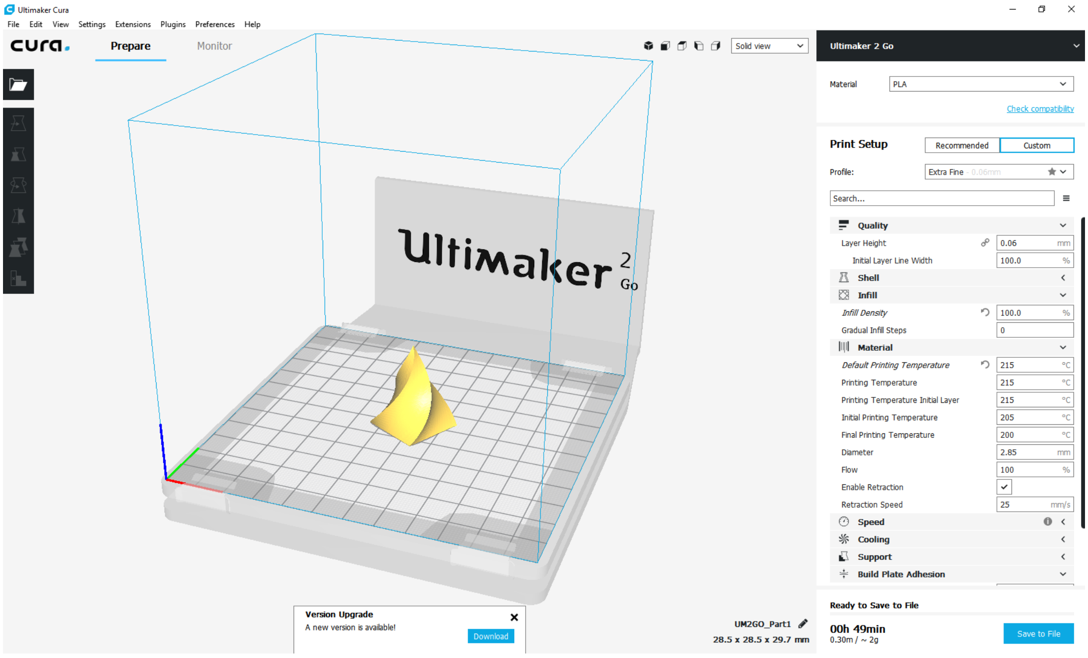


Figure 4.2: Slicing in Cura

several machine parameters can also be set in this environment including nozzle temperature, infill density, printing speed, adhesion type, support structure settings, etc. Finally, the corresponding G-codes are transferred to the 3D printer using an SD card and the process of fabrication can be initialized utilizing the user interface on the 3D printer.

4.3 Printing Scheme of the LIPRO

In this section, the part designed in the previous section is to be fabricated utilizing the LIPRO. In order to prepare the base curves, polygons are generated inside the LIPRO. Therefore, the design characteristics must be defined including the type of base polygon (number of polygon sides), radius of the base polygon, the height and its angle for each layer (if a twisted shape is required). These values are given to the main function (Algorithm 4.1) as input arguments so that at each layer the corresponding polygon is generated accordingly. This method can only be applied to the 3D models whose cross-sections can be constructed using the polygon functions available inside the LIPRO. Later, some printing parameters must be devoted containing nozzle diameter (offset distance), layer thickness and the starting position of the nozzle in the

Algorithm 4.1

```
procedure MainAlgorithm
    Input: Design parameters including height, polygon type and radius
    Output: A set of operations for printing
    Let B be the printing parameters
    Initialize serial connection
    InitialMeasure
    Printing(B,height,type, radius)
    FinalMeasure
    Close serial connection
```

z-direction. Then a serial connection between the Raspberry Pi and the machine board (Arduino mega) is established so that through this connection the G-code commands can be conveyed. In order to obtain this connection, knowing the USB port address (for the Raspberry Pi) and the baud rate is essential. Then, the operation performs some tasks that are vital to prepare the machine for printing (Algorithm 4.2). In this part of the scheme, the printer heats up the nozzle to 210 Celsius degrees which is the appropriate temperature for the thermoplastic filament to be extruded. After that, the nozzle and the bed are moved to their home locations so that the three axes can use absolute coordinates to stop in the accurate positions. Afterward, the program raises the bed to the below of the nozzle and extrudes some filaments to test if it is operational. Since the amount of extrusion is essential for the main process, it must set the amount of extruded material to zero before the printing starts. Finally, the side fans are turned on and the machine is prepared for printing the workpiece.

Algorithm 4.2

```
function InitialMeasure
    Output: A set of operations to prepare the machine for printing
    Set nozzle temperature to 210 °C
    Bring nozzle and bed to the home positions
    Bring up the bed close to the nozzle
    Extrude some filaments
    Set the amount of the extrusion to zero
    Turn the fan on
```

The printing function, presented by Algorithm 4.3, is started with a conditional loop. Here, the early value of the Z axis is set to 0.3 mm which the initial distance between the nozzle and the build table. The total number of layers can be obtained from Equation 4.1. Hence, the program iterates through some functions to establish the printing process. First, it creates the base polygon using arguments including radius, number of sides and its primary point angle. Then, in accordance with the number of offsets required for that layer (N), the program calls the offset function (Section 3.8.3) to generate the desired tool-path.

Algorithm 4.3

```
function PRINTING
    Input: Printing parameters (B), height, polygon type and radius
    Output: A set of operations for printing process
    Get the initial nozzle height Z from printing parameters B
    while Z is less than height do
        Generate POLYGON(type,radius)
        Generate OFFSET(N)
        Create GCODE(Z,B) and set it as L
        SENDGCODES(L) to machine line by line
        Z = Z + layer thickness
```

$$\text{Number of Layers} = \left\lfloor \frac{\text{Height}}{\text{Layer Thickness}} \right\rfloor \tag{4.1}$$

Afterward, having the tool-path, by using Algorithm 4.4 the G-code commands are created. In this algorithm, the amount of extrusion ratio is obtained based on the nozzle diameter and the layer thickness. This rate can be extracted from the G-codes provided by the conventional approach (Cura) by observing the G1 codes from one point to the next one. Equation 4.2 demonstrates this value which is obtained from the material used (E) between two consecutive points divided by the distance between them.

$$\text{Extrusion Rate} = \frac{E_{p_2} - E_{p_1}}{\|p_2 - p_1\|_2} \tag{4.2}$$

Algorithm 4.4

```
function GCODES(Z,B)
    Input: List of coordinates
    Input: Z value of the layer and printing parameters (B)
    Output: List of the strings that represents G-code
    Let extrusion ratio be 0.12
    for each offset in the list do
        Retract filament
        Go to the first point of the offset list with the correct height (Z)
        Extrude to cover the retraction
        for each point in offset list do
            Set the extruder position
            Write G1 code for each coordinates using the new extruder position
```

Therefore, the program loops through each offset polygon and each point inside the polygon. Then by employing extruder position and proper feed rate, the G1 commands are generated. The position of the extruder is determined using the equation below,

$$\text{Extruder Position} = \text{Extruder Position} + (\text{Distance} \times \text{Extrusion Ratio}) \tag{4.3}$$

where, distance is the length between that particular point from the iteration with the next point in the list. As it is clear, the extruder length is an incremental value and at the end of the printing, it presents how much filament was utilized in millimeters. After the G-codes are acquired, they must be sent to the machine to be executed. This task can be done from the serial connection between the Raspberry Pi and the Arduino Mega (machine mainboard). Through this connection, the strings can be dispatched line by line (Algorithm 4.5). Since the Arduino has a limited amount of buffer, all lines cannot be sent at once. Therefore, the machine must confirm that the action is done and then it can receive new commands. After each task, the buffer must be cleared so that other commands can be accepted.

After the printing is completed, like the initial settings, some measures must be ap-

Algorithm 4.5

```
function SENDGCODES(L)
    Input: List of G-code strings as L
    Output: Operation to send G-code line strings
    for each line in L do
        Send the line to the machine
        while Task is not finished do
            Check if the task is finished
        Clear the buffer
```

plied before terminating the serial connection. Algorithm 4.6 lists the details of this operation.

Algorithm 4.6

```
function FINALMEASURE
    Output: A set of operations after the printing process
    Retract the filament
    Bring nozzle and bed to the home positions
    Set nozzle temperature to 20 °C
    Turn off the side fans
```

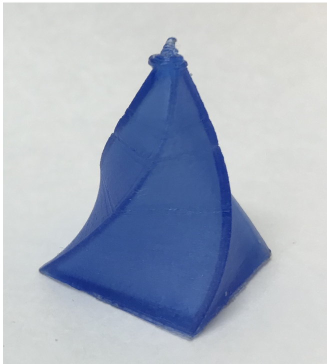
4.4 Test Prints

Following the aforementioned instructions, the same model provided in Figure 4.1a is printed by Ultimaker 2 Go using both methods. These prints are produced by the fabrication parameters revealed in Table 4.2 and the result of these processes are illustrated in Figure 4.3. In this Figure, the undesirable quality of the peak is due to the lacking of sufficient time for material to solidify.

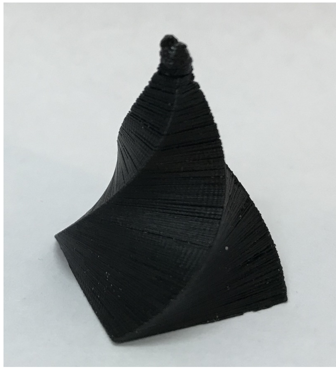
Although using the LIPRO and the conventional approach yield the same output, it is worth mentioning some of the details which differentiated these methods. As can be observed from Table 4.3, there are some extra stages in the first method to prepare the G-codes including designing the CAD model and converting it to STL file format. In addition, CAM software (Cura) must be employed to carry out the remaining op-

Table 4.2: Print Parameters

Material	**PLA**
Filament Diameter	2.85 mm
Nozzle Diameter	0.4 mm
Layer Thickness	60 μm
Feed-rate Speed	1800 mm/min
Rapid Movement Speed	6000 mm/min
Infill Density	100 %
Shell Thickness	1.2 mm



(a)



(b)

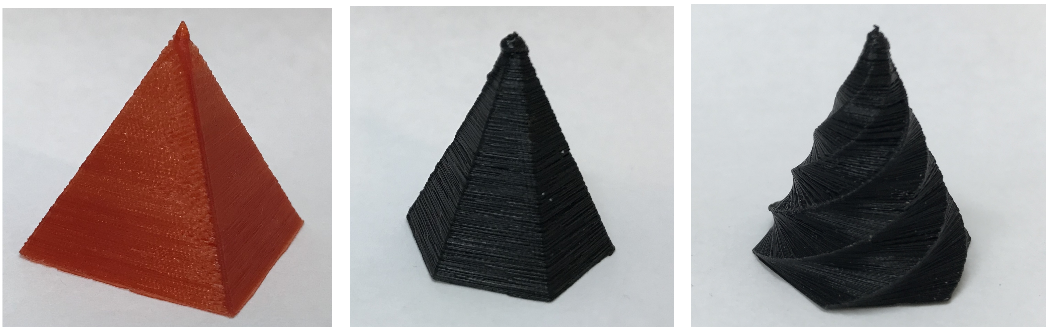
Figure 4.3: A sample printed in FDM machine using, a) Conventional method, b) LIPRO

erations. Whereas the second approach summarizes these pre-fabrication processes inside each layer of printing which results in a huge reduction of memory usage. Additionally, the third row of the Table is demonstrating this fact by comparing the size of the machine codes. The second column of this row is indicating the size of Python scripts being used for this fabrication process. Here, the LIPRO only stores one layer of G-codes and it transfers each line at a time to the FDM machine. Another distinction, which is stated in Chapter 1, is the ability of the second approach to change the design while it is in the middle of printing. In the conventional method, if any modification on the design of the artifact is required, the process of the fabrication must be aborted and all of the operations described in Section 4.2 should be executed again. Whereas, the LIPRO can pause the printing, reform the parameters inside the scripts for the new design and continue the printing from last index. Along with the new design, it also has the ability to resume the fabrication process with the new print parameters assigned in Table 4.2. For instance, it is possible to use different nozzle

Table 4.3: Comparison table, the sizes are in KB

	Conventional Approach	**LIPRO**
Size of CAD	244	—
Size of STL File	378	—
Size of Machine Code	3079 (G-code file)	8 (Python Scripts)

diameter, layer thickness, feed-rate speed and wall thickness. Figure 4.4 indicates other specimens fabricated by the LIPRO. The only difference between Figure 4.4a



(a) (b) (c)

Figure 4.4: Additional test parts printed by the LIPRO

and 4.4b is the number of sides employed in polygon function which is selected as 4 for the first model and 6 for the second one. Figure 4.4c also presents a hexagon for its basis curve like Figure 4.4b, but their main distinction is defined as the utilization of transformation function (rotation) within the right of the Figure.

CHAPTER 5

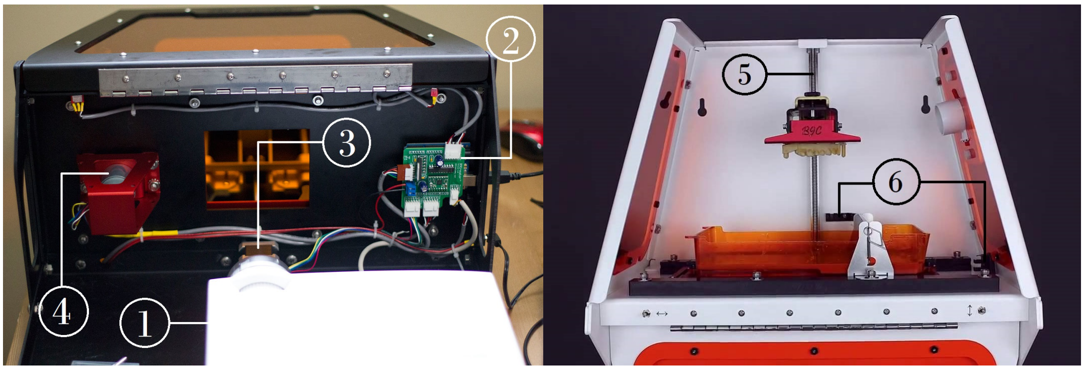
IMPLEMENTATION OF THE LIPRO ON A DLP PRINTER

5.1 Introduction

Digital Light Processing (DLP) is an Additive Manufacturing process which uses Photopolymer as material to manufacture desired parts. In this process, with the help of a projector, images (bitmaps) are projected onto a surface of the resin and the entire layer is cured at a time. Since it is a layer-wise process, speed is considered as one of its advantages as well as high resolution. An example commercial product is demonstrated in Figure 5.1 from B9Creator, which is founded by Kickstarter company in 2012 and it is the subject of the study.

Figure 5.1: B9Creator V1.2 [105]

Figure 5.2 presents main components of the DLP printer which are responsible for



(a) (b)

Figure 5.2: Major components of B9Creator V1.2 described in Table 5.1

actuating the machine, and their specifications are listed in Table 5.1. These parts are essential for controlling the printing process and their relations and duties are described below.

The projector is specifically modified for the purpose of printing and as mentioned before its task is to cure the resin of each layer based on the input bitmaps. In order to have a particular resolution, three parameters, including position, zoom and focus of the projector, play a critical role. The projector, the actuators and all the sensors are connected to a circuit board which is the brain of the system (Part 2). This circuit board is composed of an Arduino UNO along with a Motor Shield and it uses a 12-volt power supply and a USB port for communicating with the main computer (Host PC running the software of the printer).

Table 5.1: DLP Part Specifications

No.	Part Name	Descriptions	Quantity
1	Projector	Modified D912HD Projector	1
2	Printed Circuit Board	Arduino UNO along with B9Creator motor shield	1
3	Stepper Motor	1402HS050A stepper motor with 4 lead bi-polar and 1.8 degree	1
4	DC Motor	X axis DC gear motor with encoder, 131:1 eatio	1
5	Lead Screw	2 mm pitch lead screw along with anti-backlash nut	1
6	Optical Sensors	home positioning optical sensors, type EE-SX4009-P10	2

Next part is a lead screw (Part 5) which is combined with a stepper motor (Part 3), and together they provide movement of the Z axis. The screw has a 2 mm pitch per each revolution and since the stepper motor has an accuracy of 1.8 degrees, it results in 10 micrometer precision for the Z axis. Part 4 is showing a DC motor which is

utilized for the motion of the X axis. This travel can be limited by an optical limit switch. These sensors (Part 6) are used for stopping the motors when they reach the home positions.
Similar to the previous Chapter, the following sections clarify the fabrication procedure of the DLP process, both in conventional way and the LIPRO.

5.2 Conventional Approach

The usual printing procedure requires a CAD model in STL file format. Similar to Section 4.2 a designed artifact is provided in CAD software (SolidWorks) and it is converted to STL format. The remaining measures are taken by a commercial software created by B9Creations company which is particularly designed for B9Creator printer as its main menu illustrated in Figure 5.3. This software is capable of calibrating both build plate and projector, but as can be observed from the Figure, the main functions required to fulfill the printing task is of three parts.

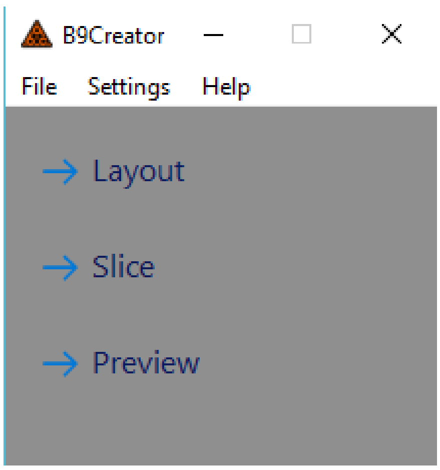


Figure 5.3: The main menu of B9Creator commercial software

To prepare the 3D model before slicing, Layout is employed in which some operations including position adjustment, orientation, scaling and generation of support structures are carried out (Figure 5.4). After saving the model, it is sliced using the second selection from the main menu. As it is illustrated in Figure 5.5, the main option available for this task is the layer thickness. Along with slicing, the bitmap images are generated with this operation. Since the size of these images is very large, they must be compressed before storing.

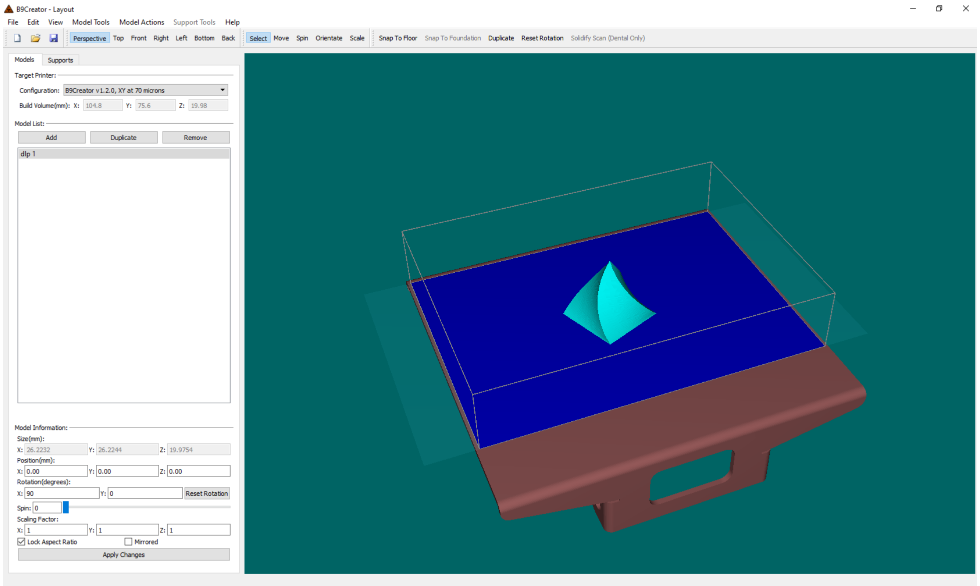


Figure 5.4: Adjusting the orientation of 3D model

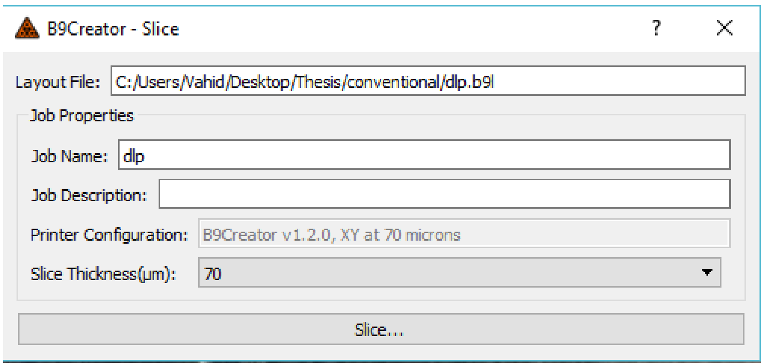
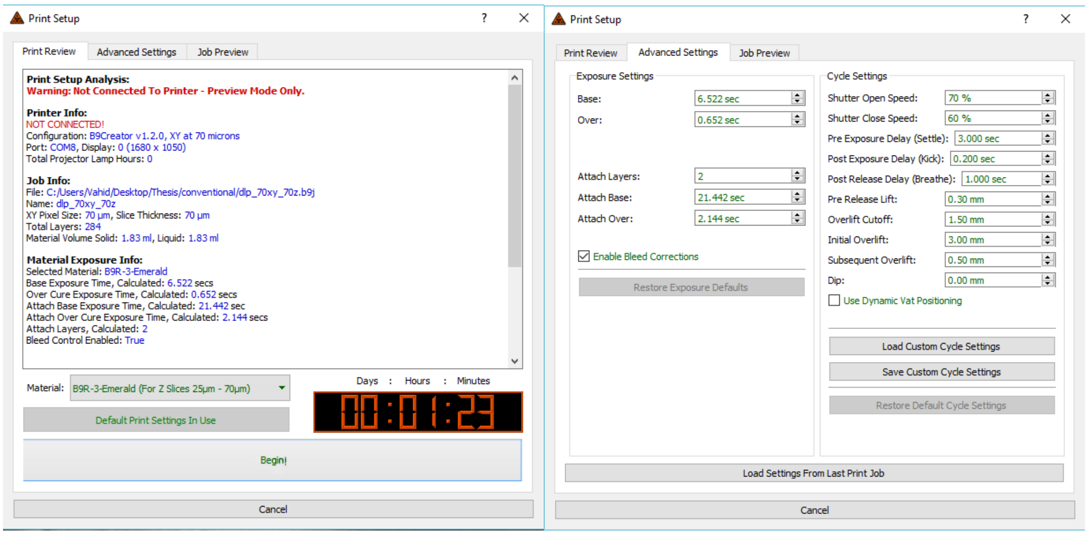


Figure 5.5: Slicing and compressing

Finally, by using Preview the compressed images are imported and print setup rises to indicate the settings required for the fabrication process (Figure 5.6a). These parameters whose values are highly effective on the quality of the end product can be modified using Advanced Setting tab located at the top bar (Figure 5.6b).

Some of these influential parameters are composed of the exposure time for both initial layers and remaining ones, number of attached layers (whose exposure time is more than other layers), delay time before exposing light, delay time after exposing, the amount of build table lift after exposing and the amount of lift for initial layers. Afterwards, by selecting Begin in the print setup part, the software asks for some instructions. These tasks which are represented in Figure 5.7 are essential to prepare



(a) (b)

Figure 5.6: Print settings

the DLP printer for the fabrication process.

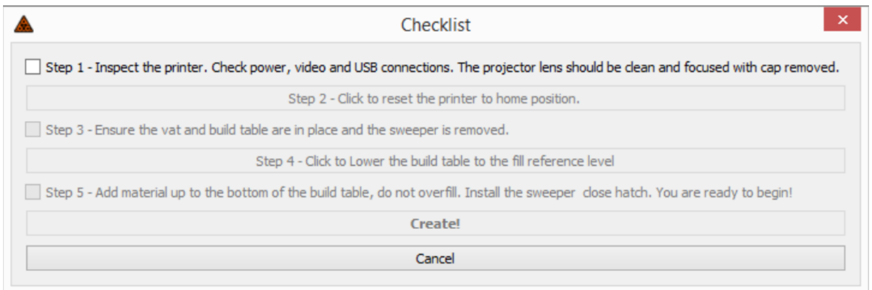


Figure 5.7: Machine preparation

5.3 Proposed Approach (The LIPRO)

In this Section, a Raspberry Pi computer is employed to control the actuators and receive singnals from the sensors of the B9Creator. For this purpose, a motor shield (Adafruit DC and Stepper Motor HAT for Raspberry Pi) is attached to the Raspberry Pi so that the PWM (Pulse Width Modulation) capability of the Raspberry Pi can be extended to control the DC and the stepper motor of the printer at the same time with the help of an extra power supply. Hence, the previous configuration of the B9Creator is changed by eliminating the main board and those connections and replacing it

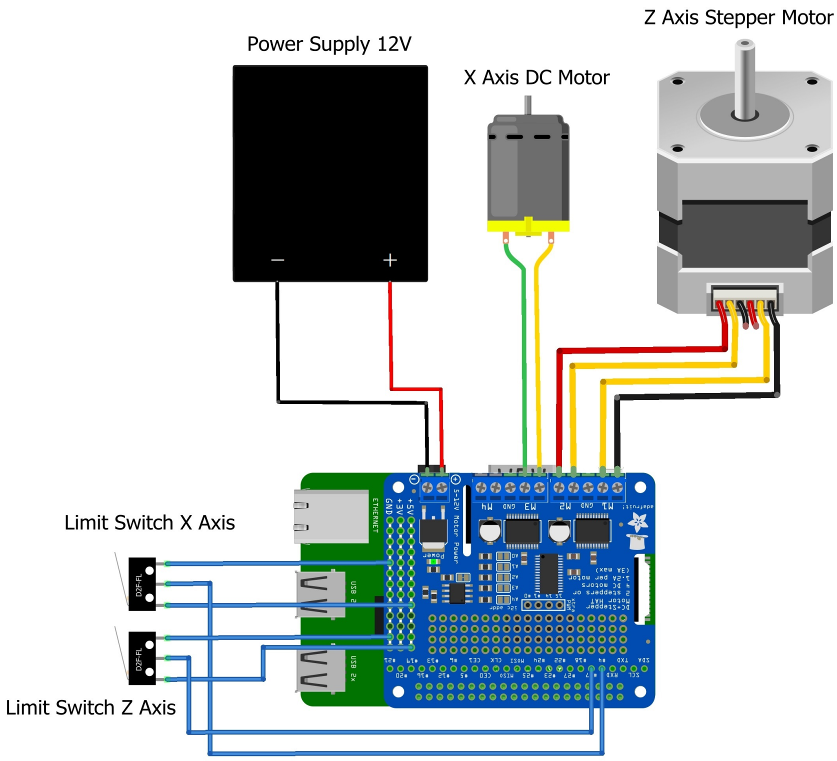


Figure 5.8: Motor shield connections

with the Raspberry Pi together with its motor shield and new connections. In this configuration, Raspberry Pi is connected directly to the projector (using an HDMI cable) and the motor shield (using GPIO pins). For the rest of the parts, the contact is indirectly through the motor shield (Figure 5.8). As can be observed from the Figure, the two groups of coils of the stepper motor are connected to the terminal blocks of M1 and M2, the DC motor uses terminal M3 and for the limit switches the connections are different. Each switch requires 5 Volt to operate. Thus, two wires reach the ground and 5 Volt pins, and the other which can give a signal (high or low) is attached to the one of the GPIO pins of the Raspberry Pi. Some of those pins are available on the motor shield. Here, pin number 4 and 17 are utilized as input pins for the limit switches.

5.3.1 Creating Bitmap Images

Prior to the print procedure, since each layer projects a mask, the process of generating bitmap images must be clarified. The running time of this process is of a great

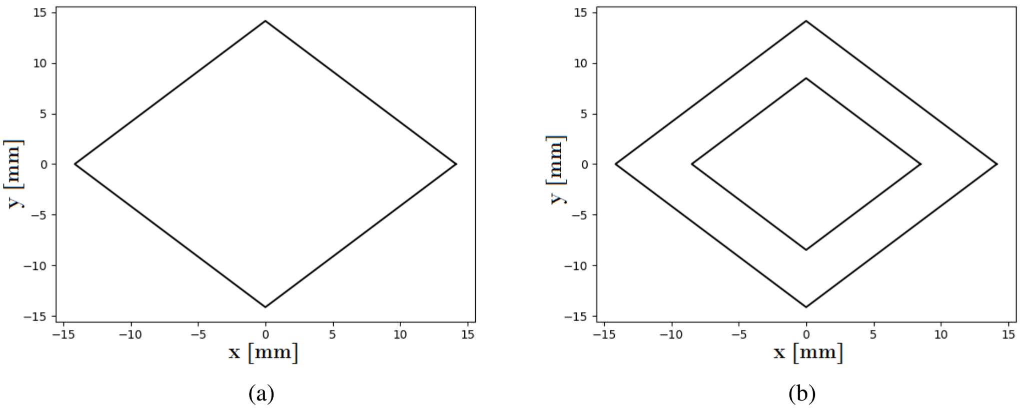


Figure 5.9: Polygon sample utilized in this process

importance and affects the printing time directly. Therefore, in this Section two algorithms to generate images are provided and the performance of each is analyzed on Raspberry Pi.

Firstly, the original 2D shape is created and the (x,y) coordinates are extracted. For this purpose, gpc function which is explained in Section 3.2 is employed. Figure 5.9 displays one of these types, without the offsets (5.9a) and with an offset (5.9b). The function utilized for generating the curve offsets is elaborated in Section 3.8. A summary of the entire operation is illustrated in Algorithm 5.1. After polygon is created, each polygon is scaled to the new screen dimensions since the projector of the B9Creator has a resolution of 1920×1080. If the projector and the printing parameters are set for printing with layer thickness of 70µm, then the size of the screen is 104×75.6 ($Width \times Height$). In order to maintain the aspect ratio of the polygon, their coordinates are multiplied by a constant amount which is obtained from the below equation:

$$\text{Ratio} = min\left(\frac{\text{MaxWidth}}{\text{Width}}, \frac{\text{MaxHeight}}{\text{Height}}\right) \tag{5.1}$$

where MaxWidth and MaxHeight stand for the projector resolution 1920×1080, respectively. Now a linear translation from the center of the polygon to the center of the projector screen locates the 2D shape at the center.

5.3.1.1 Image Generation Algorithms

After having the polygon scaled and centered on the new screen, bitmap image can be created using Python Imaging Library (PIL) within two methods. The first one which is described in Algorithm 5.2, starts with setting the screen in accordance with the projector resolution. Since this configuration follows the Cartesian coordinate system, and the corresponding polygon is also modified and scaled to be fit in this system, each pixel must be checked whether it is inside or outside of the polygon. This operation requires two indices provided by two for loops to represent a pixel which is known as a point. The script checks the point and if it is inside the polygon or outside the offset polygon, it sets the pixel's value to white by employing its RGB code (255,255,255). Therefore, all the points within the polygon boundary are checked and the bitmap image is generated.

In the second approach, an array containing 1080 rows and 1920 columns is created. Each value of this array is set to be a 1×3 vector and each element is set to be zero. Here, the algorithm only goes through each row which is started from minimum value of the polygon to its maximum. For each row, a line is formed from the starting point of the row to the end of the row. Then, the script checks if the line is colliding with both the original polygon and the offset polygon. Afterwards, from the intersection points it can identify the exposed region that must be filled with the white color. After all the rows are covered, the corresponding array can be converted to a pixel map by a single operation.

Algorithm 5.1

```
procedure IMAGE
    Input: Design parameters including polygon type and radius
    Output: Bitmap image
    CREATEPOLYGON(type,radius) and set it as P
    SCALE(P)
    CENTER(P)
    Generate BITMAP(P) image
```

Algorithm 5.2

```
procedure BITMAP(P)
    Input: A list of coordinates representing polygon
    Output: A pixel map as bitmap image
    Set the resolution and create pixel map
    Take the minimum and maximum boundaries of the polygon
    for i =min width of polygon to max width do
        for j =min height of polygon to max height do
            if pixel[i,j] is inside original polygon and outside the offset then
                Set value of the pixel as (255,255,255)
```

5.3.1.2 Comparison and Result

Both approaches are implemented on a test case illustrated in Figure 3.2 and the results prove the capability of both for generating bitmap images accurately (Figure 5.10). The difference in their performance is characterized by the execution time. From the examination on these methods, it is observed that the first algorithm utilized a slower time complexity $O(n^2)$ since it is checking each pixel at a time. On the other hand, the second approach instead of examining each pixel, it analyzes the entire row of pixels and performs the task with a time complexity of $O(n)$.

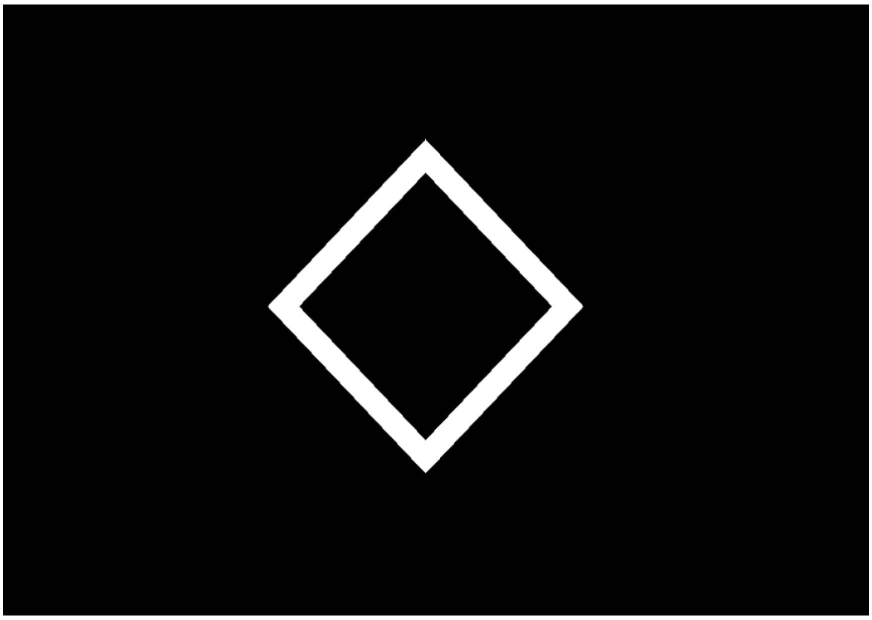


Figure 5.10: Generated bitmap image

Algorithm 5.3

```
procedure BITMAP(P)
    Input: A list of coordinates representing polygon
    Output: A pixel map as bitmap image
    Define a 2D array and assign black RGB code for each element
    Take the minimum and maximum boundaries of the polygon
    for each row of the array do
        Set a line from starting and ending point of this row
        if line intersects the original polygon then
            if line intersects the offset polygon then
                Find the intersection points for original polygon and its offset
                Set the exposed elements to (255,255,255) from intersection points
            else
                Find intersection points for original polygon
                Set the exposed elements of array to (255,255,255)
    Create pixel map based on the final array
```

To further evaluate the image generation algorithms, these two methods are run on Raspberry Pi 3 (with the processor of Quad Core 1.2GHz Broadcom BCM2837 64bit CPU) and their results are presented in Table 5.2. Here, three polygons with different sizes and having an offset of 2 mm are assessed to demonstrate the influence of the time complexity on the outcome. The Table confirms the effect of polygon size on the execution time especially on the first algorithm which is more extreme. In conclusion, to be able to print on the DLP printer without losing time, method number two is appeared to be efficient and is employed in the next parts of the thesis.

Table 5.2: Running time (sec) for each polygon size

	20×20	**30×30**	**40×40**
First Method	11.66	19.06	26.50
Second Method	1.18	1.65	2.15

Algorithm 5.4

procedure BTCALIBRATION
    Output: A set of operations for calibrating the table
    Move build table and vat to their home positions
    Move build table by 5 cm down
    Wait for command
    Move Bed to the home position

5.3.2 Printing

In this Section, the algorithm for controlling the printing process is going to be described using pseudocodes (Python codes can be found in the Appendices). Similar to the conventional approach, before dealing with the printing process, several calibrations must be exerted. Therefore, Algorithm 5.4 is performing this operation in a simple manner. Before executing this function, the build table must be mounted and it is essential to loose the four screws on its both side so that the surface of the build table can touch the vat without harming it. This function initially moves the table to the home position by calling another function named as BedHomePosition which will be described later. Since the distance between the home position and the vat is a constant, a 5 cm movement will assure the table to touch the vat. After the movement ceases, the operator can tighten the four screws and by pressing Ctrl+C the bed will rise to the home position again. Also, for calibrating the projector, Algorithm 5.5 is provided which is only capable of calibrating the projector sharpness. By running this function, a black screen appears and the vat travels to the left side so that the projector light can enter the upper side of the machine. Then, the calibration image is projected on the upper surface of the vat and the script waits for the operator to adjust the sharpness. Afterward, by pressing Esc from keyboard, the vat travels to the right side and the corresponding screen is closed. For better comprehension, the readers are referred to watch the B9Creator instructions for calibrating the projector and the build table.

The printing operation is performed in three stages including initial setting, printing process and final setting (Algorithm 5.6). In the first step, before turning on the projector, the vat travels to its home position (Xhome) in order to prevent the projector

Algorithm 5.5

```
procedure PROJCALIBRATION
    Output: A set of operations for adjusting the sharpness of the projector
    Turn screen into black
    Move vat to the left side
    Display the calibration image
    Wait for command
    Move vat to the right side
    Close the black screen
```

light from entering the vat. Then, a black screen needs to come up so that it prevents projector to cure the resin. Then, the projector can be turned on. While the projector is warming up, the bed is sent to its home position and then brought down for 5 cm to prepare the machine for printing (Algorithm 5.7).

Algorithm 5.6

```
procedure PRINTING
    Input: Design parameters including height, polygon type and redius
    Output: A set of instructions for printing
    INITIALSETTING
    Wait for operator to start printing
    PRINTINGPROCESS(height,type,radius)
    FINALSETTING
```

After execution of the initial settings, the program ceases for operator to fill the vat with photo polymer resin and connect the sweeper. Later, by pressing Esc the printing process begins. The function employed for moving the build table to its home location (Algorithm 5.8) explicitly presents how a limit switch is utilized for sending bed to its home location.

Here, pin number 17 of the Raspberry Pi is used as an input to get signals from the optical limit switch. The stepper motor must be set and prepared to work by allocating its number of steps and speed. In this function, the initial pulse from the sensor declares the location of the bed. If the sensor detects one (variable A is detected), it is inferred that the red lever connected to the bed is engaging with the sensor and the

Algorithm 5.7

```
function InitialSetting
    Output: A set of operations
    Define both motors
    Initiate a black screen
    Turn on the projector
    XHome
    Brings down the bed for 50 mm
```

Algorithm 5.8

```
function BedHomePosition
    Output: A set of operations to stop the bed in its home position
    Setup GPIO pin 17 as an input
    Let A be the signal comes from pin 17
    if A not detected then
        while A not detected do
            Let A be the signal comes from pin 17
            Turn stepper motor for one step forward (down)
    else
        while A is detected do
            Let A be the signal comes from pin 17
            Turn stepper motor for one step backward (up)
```

bed motion must be upward. Therefore, a while loop is employed to turn the stepper motor until the against signal is detected. For each loop, the stepper motor is turned for one step and also the amount of voltage in pin 17 is read. Immediately after the program detects this amount against its initial value, it breaks the loop and the stepper motor stops turning. Consequently, for each initial position of the bed, this function is able to lead it to its home location.

For sending vat to its home, Algorithm 5.9 is utilized. The general idea is similar to the previous one which uses an optical limit switch for taking a signal. The function starts with setting up the pin number 4 as an input for the sensor. Then, the DC motor is prepared to operate at a specific speed. Afterward, inside a while loop, the DC motor rotates backward until the red sheet beneath the vat reaches to the sensor. The high pulse coming from the sensor breaks the loop and the program ceases the DC motor in its home position.

Algorithm 5.9

```
function XHOME
    Output: A set of operations to stop the Vat in its home position
    Setup GPIO pin 4 as an input
    Define DC motor and its parameters
    Let A be the signal comes from pin 4
    while A is detected do
        Let A be the signal comes from pin 4
        Turn DC motor backward
    Stop DC motor
```

The printing process which is provided by Algorithm 5.10 takes the design parameters as an input. Based on the number of layers provided from the geometric parameter (height) and the amount of resolution (layer thickness), the script loops through each layer and performs the following operations. At the beginning of each loop, the screen color must be set to black to avoid curing resin more than their exposure time. Through the function KeyboardEvent, the script can pause or abort the fabrication process just by pressing P and Esc on the keyboard, respectively. For initial layers, in order to have better attachment between printed part and the build plate, the time for curing and the amount of lifting must be set differently. Therefore, the if statements presented in this Algorithm checks the indices and set these parameters. Then, the bed rises based on the amount defined for lifting. In this function, recoat is referred to the motion of the vat in which it first travels to the home position and then comes back to the left side. By this movement, the sweeper can clean the vat surface leading to better attachment of the next layer. Afterward, the bed pulls down with one layer thickness away from the vat surface. The corresponding image is generated based on the index and the geometrical parameters (details of this operation is demonstrated in Section 5.3.1). Eventually, the bitmap image is displayed in accordance with the exposing time allocated in the previous lines. This process continues till the last layer and the 3D model is manufactured.

Algorithm 5.10

```
function PRINTINGPROCESS
    Input: Design parameters including height, polygon type and redius
    Output: A set of operations for printing
    Let N be the number of layers obtained from parameter height
    Set the initial count to zero
    while counter is less than N do
        Update the screen to black
        KEYBOARDEVENT
        if counter is less than 2 then
            Set the initial exposure time
            Set the initial overlift
        else
            Set the sequential exposure time
            Set the sequential overlift
        Move up the bed with the amount defined for overlift
        Recoat
        Lower down the bed for (overlift - resolution)
        Generate IMAGE(type,radius)
        Display image for the amount defined in exposure time
        counter = counter + 1
```

After the printing is done, the stepper motor lifts the bed up to some extent that the operator can extract the bed without colliding with the vat. Then, the vat moves to the left and the projector is turned off. Finally, the script closes the black screen and disables the motors (Algorithm 5.11). Now, the bed can be separated and the operator can remove the workpiece.

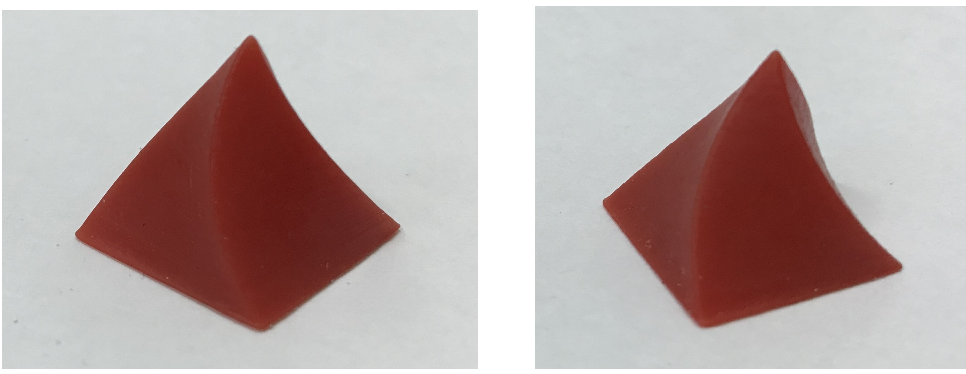
5.4 Results and Discussions

Similar to the previous Chapter, both aforementioned methods are employed to produce samples on the B9Creator printer. They utilized the same printing parameters demonstrated in Figure 5.6a to fulfill the task. Despite the same quality observed in

Algorithm 5.11

```
function FINALSETTING
    Output: A set of operations after printing
    Bring up the bed
    Move vat to the right side
    Turn off the projector
    Wait for the projector to be turned off
    Close the black screen
    Disable the motors
```

Figure 5.11, there exist some distinctions elaborated in Table 5.3 which can prove the capability of the LIPRO for reducing memory usage. As it is realized from the Table, the four steps require in conventional method for printing are gathered in a single process with 19 KB of size. Additionally, another term which is recognized as the main advantage of the LIPRO is the ability to pause the printing process, modify the designed artifact and print parameters, and continue printing with the updated design. As mentioned before, in order to accomplish this task in the conventional approach, the entire operations stated in Table 5.3 must be executed from the start.



(a) (b)

Figure 5.11: Printed samples by a) Conventional approach, b) LIPRO

The design changes can be exerted in the LIPRO without any complications. To appreciate this concept, Figure 5.12 is presented to indicate the flexibility of this new paradigm. There is only a single line of Python code excluded from the print function of Figure 5.11b so that Figure 5.12a can eventuate. That particular line defines the radius of the polygon in accordance with the index of that layer and by removing this

Table 5.3: Comparison table, the sizes are in KB

	Conventional Approach	**LIPRO**
Size of CAD	162	—
Size of STL	223	—
Size of B9Layout	1	—
Size of B9Job	246	19 (Size of Python scripts)

line, the radius is considered as constant throughout the entire fabrication process. In addition, Figure 5.12c is demonstrating the design change during printing process. While printing this specimen, the process paused for two times and different design plan applied for the remaining process.

(a) (b) (c)

Figure 5.12: Other specimens printed by LIPRO

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this dissertation, a new design and fabrication pipeline for 3D printers is introduced and its operational algorithms along with necessary Python scripts for implementing on FDM and DLP printers are delivered. In the first Chapter, the motivation and reasons behind this study are illuminated and the related literature is reviewed in the second Chapter. After that, structure of the proposed pipeline (LIPRO) is described in details and the process of generating motion trajectories is evaluated in three different approaches which represents the flexibility of the paradigm. Afterward, the instructions and the algorithms for implementing on FDM and DLP printers are provided in Chapter 4 and 5, respectively. At the end of these Chapters, some test cases printed by the LIPRO are demonstrated and through a discussion, the conventional approach and the LIPRO are compared. As a result of this analysis, the advantages of fabrication using the LIPRO are eventuated. A brief recapitulation of the main contributions provided in the chapter are presented in the following paragraphs.

In the second Chapter of the thesis, the significant role of Additive Manufacturing in today's manufacturing industry is discussed. Then, the conventional AM pipeline is described and the background for some parts of the proposed pipeline which are crucial for this study was assessed exclusively. In addition, the details of two AM processes which are the subject of the experiments in this dissertation are presented.

The third chapter introduces the LIPRO as a new design and fabrication pipeline for AM machinery and discusses how this paradigm is able to alleviate the drawbacks of the standard AM pipelines mentioned before. Within the structure of the LIPRO, three course of actions are assigned for different input data to provide the base curve in each layer. Later, some operations for modifying the designed geometry and the functions for generating the final motion trajectories are presented. After the general overview of the LIPRO is described, the functions for each stage are discussed in details. Since curve offsetting plays an essential role in the last stage and the accuracy of the end product depends on them significantly, more research is invested to this section. In order to supply an efficient offset algorithm for the LIPRO, three curve offset methods investigated and through some experiments, Shapely is recognized as a more qualified approach.

Chapter 4 demonstrates a direct set of commands and algorithms to fabricate a 3D model on an FDM printer by employing both the conventional method and the LIPRO. By utilizing a printed sample, the contrasts between these methods are examined. This comparison proved the new paradigm utility to overcome the limitations emphasized in Chapter 1. More specimens are printed by the LIPRO to demonstrate the flexibility of this approach for fabricating various artifacts by simply changing few parameters and lines.

Similarly Chapter 5 is dedicated to describe the course of actions required for fabricating on a DLP printer using both the standard method and the LIPRO. Within the implementation of the new paradigm, additional instructions for controlling the machine components (actuators and sensors) are devoted. Since generating bitmap images is the most crucial operation affecting the efficiency of the printing process considerably, two algorithms are developed for this purpose. The performances of these algorithms are examined in terms of the execution time and as a result, one of them is chosen to be employed in the LIPRO. In a similar manner to the prior Chapter, the differences in both printing methods are discussed and the final results once again support the capabilities of the LIPRO to resolve the difficulties mentioned in the first Chapter.

6.2 Future Works

Although the presented experimental works succeeded to accomplish the studies assigned within the scope of this dissertation, there are still a lot of research arising to improve this new design and fabrication pipeline.

Before fabricating each layer, the slicing algorithm provided in this study is only slicing the particular layer and if this function enriched with the best quality programming algorithms, it can reduce the overall fabrication time significantly. The current slicing function still requires some improvement in terms of execution time even though, it is able to slice a model with or without islands accurately.

Despite the fact that Python is a more comprehensible programming language, it is still difficult for users to employ functions in a correct sequence for a particular printing task. Therefore, a graphical user interface (GUI) needs to be developed that can perform properly on various 3D printers.

For the overhanging parts within the layer which are not supported by the prior layer, additional materials are required beneath those parts to ensure an accurate print. At the moment, the presented paradigm in this study is lacking this function and it is expected to be provided in the future.

Since the orientation of a fabricated part has a great influence on the quality of the end product and the amount of support structures, its existence is demanded. Therefore, while providing a GUI for the LIPRO this fact must be considered and a proper interface can be provided.

The tool-path pattern utilized in this study is composed of only parallel contours. However, some other studies should be conducted to evaluate the efficiency of other patterns or mixed patterns in the performance of the final product. Hence, the functionality of the LIPRO can be increased if it equips with various infill patterns.

REFERENCES

[1] T.-H. Kwok, H. Ye, Y. Chen, C. Zhou, and W. Xu, "Mass Customization: Reuse of Digital Slicing for Additive Manufacturing," *Journal of Computing and Information Science in Engineering*, vol. 17, p. 021009, 2 2017.

[2] U. Yaman and M. Dolen, "A command generation approach for desktop fused filament fabrication 3D printers," *IECON Proceedings (Industrial Electronics Conference)*, pp. 4588–4593, 2016.

[3] U. Yaman and M. Dolen, "A novel command generation paradigm for production machine systems," *Robotics and Computer-Integrated Manufacturing*, vol. 51, pp. 25–36, 6 2018.

[4] D. Bak, "The rapid prototyping technologies," *Rapid Prototyp. J.*, vol. 23, no. 4, p. 26, 2003.

[5] J. Holmström, M. Holweg, S. H. Khajavi, and J. Partanen, "The direct digital manufacturing (r)evolution: definition of a research agenda," *Oper. Manag. Res.*, 2016.

[6] D. Chen, S. Heyer, S. Ibbotson, K. Salonitis, J. G. Steingrímsson, and S. Thiede, "Direct digital manufacturing: definition, evolution, and sustainability implications," *J. Clean. Prod.*, vol. 107, pp. 615–625, 11 2015.

[7] V. Petrovic, J. Vicente, H. Gonzalez, O. J. Ferrando, J. Delgado Gordillo, J. Ramón, B. Puchades, . Luis, P. Griñan, O. J. Jorda´ferrando, J. Ramo´n, R. B. Puchades, L. Portole´s, and P. Grinãn, "International Journal of Production Research Additive layered manufacturing: sectors of industrial application shown through case studies Additive layered manufacturing: sectors of industrial application shown through case studies," *Int. J. Prod. Res.*, vol. 49, no. 4, pp. 1061–1079, 2011.

[8] F. I. Azam, A. M. Abdul Rani, K. Altaf, T. V. Rao, and H. A. Zaharin, "An In-Depth Review on Direct Additive Manufacturing of Metals," in *IOP Conf. Ser. Mater. Sci. Eng.*, 2018.

[9] M. Koike, K. Martinez, L. Guo, G. Chahine, R. Kovacevic, and T. Okabe, "Evaluation of titanium alloy fabricated using electron beam melting system for dental applications," *J. Mater. Process. Technol.*, vol. 211, pp. 1400–1408, 8 2011.

[10] L. E. Murr, S. M. Gaytan, D. A. Ramirez, E. Martinez, J. Hernandez, K. N. Amato, P. W. Shindo, F. R. Medina, and R. B. Wicker, "Metal Fabrication by Additive Manufacturing Using Laser and Electron Beam Melting Technologies," *J. Mater. Sci. Technol.*, vol. 28, pp. 1–14, 1 2012.

[11] A. Gebhardt, F.-M. Schmidt, J.-S. Hötter, W. Sokalla, and P. Sokalla, "Additive Manufacturing by selective laser melting the realizer desktop machine and its application for the dental industry," *Phys. Procedia*, vol. 5, pp. 543–549, 1 2010.

[12] W. E. Frazier, "Direct Digital Manufacturing of Metallic Components: Vision and Roadmap,"

[13] J. Yu, M. Rombouts, and G. Maes, "Cracking behavior and mechanical properties of austenitic stainless steel parts produced by laser metal deposition," *Mater. Des.*, vol. 45, pp. 228–235, 3 2013.

[14] S. Yang and Y. F. Zhao, "Additive manufacturing-enabled design theory and methodology: a critical review,"

[15] A. Telea and A. Jalba, "Voxel-Based Assessment of Printability of 3D Shapes," tech. rep.

[16] X. Rolland-Nevière, G. Doërr, and P. Alliez, "Robust diameter-based thickness estimation of 3D objects," *Graphical Models*, vol. 75, pp. 279–296, 11 2013.

[17] C. C. L. Wang and Y. Chen, "Thickening freeform surfaces for solid fabrication," Tech. Rep. 6, 2013.

[18] L. Luo, I. Baran, and S. Rusinkiewicz, "Chopper : Partitioning models into 3D-printable parts The MIT Faculty has made this article openly available . Please share how this access benefits you . Your story matters . Citation Association for Computing Machinery (ACM) Author ' s final manuscrip," 2016.

[19] T. Ju, "Fixing geometric errors on polygonal models: A survey," Tech. Rep. 1, 2009.

[20] M. Attene, "Polygon Mesh Repairing: An Application Perspective," vol. 45, 2013.

[21] M. Campen and L. Kobbelt, "Exact and Robust (Self-)Intersections for Polygonal Meshes," *Computer Graphics Forum*, vol. 29, pp. 397–406, 5 2010.

[22] Sushant Negi, S. D. Sharma, and R. Kumar, "Basics, applications and future of additive manufacturing technologies : A review," no. March, 2013.

[23] D. C. Thompson and R. H. C. Research, "Optimizing Part Quality with Orientation," tech. rep.

[24] D. Frank and G. Fadel, "Expert system-based selection of the preferred direction of build for rapid prototyping processes," tech. rep., 1995.

[25] W. Cheng, J. Y. H. Fuh, A. Y. C. Nee, Y. S. Wong, H. T. Loh, and T. Miyazawa, "Rapid Prototyping Journal Multi-objective optimization of part-building orientation in stereolithography Article information," tech. rep.

[26] P.-T. Lan, S.-Y. Chou, L.-L. Chen, and D. Gemmill, "Determining fabrication orientations for rapid prototyping with Stereolithography apparatus," *Computer-Aided Design*, vol. 29, pp. 53–62, 1 1997.

[27] J. H˜ar and K. Lee, "The Development of a CAD Environment to Determine the Preferred Build-up Direction for Layered Manufacturing," tech. rep., 1998.

[28] B. Ezair, F. Massarwi, and G. Elber, "Orientation analysis of 3D objects toward minimal support volume in 3D-printing," *Computers & Graphics*, vol. 51, pp. 117–124, 10 2015.

[29] R. Khardekar and S. McMains, "Fast Layered Manufacturing Support Volume Computation on GPUs," in *Volume 1: 32nd Design Automation Conference, Parts A and B*, vol. 2006, pp. 993–1002, ASME, 1 2006.

[30] H. D. Morgan, J. A. Cherry, S. Jonnalagadda, D. Ewing, J. Sienz, and H. D. Morgan hdmorgan, "Part orientation optimisation for the additive layer manufacture of metal components," *Int J Adv Manuf Technol*, vol. 86, pp. 1679–1687, 2016.

[31] P. Delfs, M. Tows, and H.-J. Schmid, "Optimized build orientation of additive manufactured parts for improved surface quality and build time," *Additive Manufacturing*, vol. 12, pp. 314–320, 10 2016.

[32] S. Masood, W. Rattanawong, and P. Iovenitti, "A generic algorithm for a best part orientation system for complex parts in rapid prototyping," *Journal of Materials Processing Technology*, vol. 139, pp. 110–116, 8 2003.

[33] S. H. Masood, W. Rattanawong, and P. Iovenitti, "Part Build Orientations Based on Volumetric Error in Fused Deposition Modelling," tech. rep., 2000.

[34] K. Hildebrand, B. Bickel, and M. Alexa, "Orthogonal slicing for additive manufacturing," *Computers & Graphics*, vol. 37, pp. 669–675, 10 2013.

[35] D. Ahn, H. Kim, and S. Lee, "Fabrication direction optimization to minimize post-machining in layered manufacturing," *International Journal of Machine Tools and Manufacture*, vol. 47, pp. 593–606, 3 2007.

[36] X. Zhang, X. Le, A. Panotopoulou, E. Whiting, and C. C. Wang, "Perceptual Models of Preference in 3D Printing Direction," *ACM Trans. Graph*, vol. 34, 2015.

[37] E. Ulu, E. Korkmaz, K. Yay, O. B. Ozdoganlar, and L. B. Kara, "Enhancing the Structural Performance of Additively Manufactured Objects Through Build Orientation Optimization," *Journal of Mechanical Design*, vol. 137, p. 111410, 11 2015.

[38] N. Umetani and R. Schmidt, "Cross-sectional Structural Analysis for 3D Printing Optimization," tech. rep.

[39] K. Hu, S. Jin, and C. C. Wang, "Support slimming for single material based additive manufacturing," *Computer-Aided Design*, vol. 65, pp. 1–10, 8 2015.

[40] T. Reiner and S. Lefebvre, "Interactive Modeling of Support-free Shapes for Fabrication," 5 2016.

[41] C. F. Kirschman, C. C. Jara-Almonte, and A. Bagchi, "Computer Aided Design of Support Structures for Stereolithographic Components," tech. rep.

[42] D. R. Smalley and B. Park, "United States Patent (19) 11 Patent Number: 5,854,748," tech. rep.

[43] K. Chalasani, L. Jones, and U. Larry Roscoe, "Support Generation for Fused Deposition Modeling," tech. rep.

[44] H. Xiaomao, Y. E. Chunsheng, X. Huang, C. Ye, J. Mo, and H. Liu, "Slice Data Based Support Generation Algorithm for Fused Deposition Modeling," *Tsinghua Science and Technology*, vol. 14, pp. 223–228, 6 2009.

[45] E. K. Heide, "Method for generating and building support structures with deposition-based digital manufacturing systems," tech. rep., 2010.

[46] A. L. B. William R. Priedeman, Jr., "Soluble material and process for three-dimensional modeling," 2004.

[47] O. J. Hildreth, A. R. Nassar, K. R. Chasse, and T. W. Simpson, "Dissolvable Metal Supports for 3D Direct Metal Printing,"

[48] E. Sabourin, S. A. Houser, and J. H. Bøhn, "Adaptive slicing using stepwise uniform refinement," *Rapid Prototyp. J.*, vol. 2, no. 4, pp. 20–26, 1996.

[49] R. L. Hope, R. N. Roth, and P. A. Jacobs, "Adaptive slicing using stepwise uniform refinement," *Rapid Prototyping Journal*, vol. 3, no. 3, pp. 274–288, 1997.

[50] J. Tyberg and J. H. Bøhn, "Local adaptive slicing," Tech. Rep. 3, 1998.

[51] E. Sabourin, S. A. Houser, and J. H. Bøhn, "Accurate exterior, fast interior layered manufacturing," Tech. Rep. 2, 1997.

[52] K. Mani, P. Kulkarni, and D. Dutta, "Region-based adaptive slicing," *Computer-Aided Design*, vol. 31, pp. 317–333, 4 1999.

[53] C. F. Kirschman and C. C. Jara-Almonte, “A Parallel Slicing Algorithm for Solid Freeform Fabrication Processes,” tech. rep.

[54] K. Tata, G. Fadel, A. Bagchi, and N. Aziz, “Efficient slicing for layered manufacturing,” *Rapid Prototyping Journal*, vol. 4, no. 4, pp. 151–167, 1998.

[55] Y.-S. Liao and Y.-Y. Chiu, “A New Slicing Procedure for Rapid Prototyping Systems,” tech. rep., 2001.

[56] S. J. Rock and M. J. Wozny, “Generating Topological Informa*ion from a "Bucket of Facets",” tech. rep., 1992.

[57] S. J. Rock and i. Vvozny, “Utilizing Topological Information to Increase Scan Vector Generation Efficiency,” tech. rep.

[58] S. Mcmains and C. Sequin, “A Coherent Sweep Plane Slicer for Layered Manufacturing,” tech. rep., 1999.

[59] T. Van Hook, “Real-time shaded NC milling display,” Tech. Rep. 4.

[60] W. Zhang, X. Peng, M. C. Leu, and W. Zhang, “A Novel Contour Generation Algorithm for Surface Reconstruction From Dexel Data,” *Journal of Computing and Information Science in Engineering*, vol. 7, p. 203, 9 2007.

[61] K. Yuksek, W. Zhang, B. I. Ridzalski, and M. C. Leu, “A new contour reconstruction approach from dexel data in virtual sculpting,” *3rd International Conference on Geometric Modeling and Imaging: Modern Techniques and Applications, GMAI*, pp. 82–86, 2008.

[62] P. Huang, C. C. L. Wang, and Y. Chen, “Intersection-Free and Topologically Faithful Slicing of Implicit Solid,” *Journal of Computing and Information Science in Engineering*, vol. 13, p. 021009, 4 2013.

[63] P. Huang, C. C. L. Wang, and Y. Chen, “Algorithms for Layered Manufacturing in Image Space,” in *Advances in Computers and Information in Engineering Research, Volume 1*, ASME Press.

[64] H. Zhao, F. Gu, Q.-X. Huang, J. Garcia, Y. Chen, C. Tu, B. Benes, H. Zhang, D. Cohen-Or, and B. Chen, “Connected Fermat Spirals for Layered Fabrication,”

[65] Y.-a. Jin, Y. He, J.-z. Fu, W.-f. Gan, and Z.-w. Lin, “Optimization of tool-path generation for material extrusion-based additive manufacturing technology,” *Additive Manufacturing*, vol. 1-4, pp. 32–47, 10 2014.

[66] Y. W. Y. Weidong, “Optimal path planning in Rapid Prototyping based on genetic algorithm,” *2009 Chinese Control and Decision Conference*, pp. 5068–5072, 2009.

[67] P. K. Wah, K. G. Murty, A. Joneja, and L. C. Chiu, "Tool path optimization in layered manufacturing," *IIE Transactions*, vol. 34, pp. 335–347, 4 2002.

[68] K. Y. Fok, N. Ganganath, C. T. Cheng, and C. K. Tse, "A 3D printing path optimizer based on Christofides algorithm," *2016 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2016*, pp. 9–10, 2016.

[69] N. Ganganath, C.-t. Cheng, K.-y. Fok, and C. K. Tse, "Trajectory Planning for 3D Printing : A Revisit to Traveling Salesman Problem," vol. 2, pp. 287–290, 2016.

[70] K. Castelino, R. D'souza, and P. K. Wright, "Tool-path Optimization for Minimizing Airtime during Machining," tech. rep.

[71] L. E. Weiss, R. Merz, F. B. Prinz, G. Neplotnik, P. Padmanabhan, L. Schultz, and K. Ramaswami, "Shape Deposition Manufacturing of Heterogeneous Structures," *Journal of Manufacturing Systems*, 1997.

[72] K.-H. Shin, H. Natu, D. Dutta, and J. Mazumder, "A method for the design and fabrication of heterogeneous objects," *Materials & Design*, vol. 24, pp. 339–353, 8 2003.

[73] S.-M. Park, R. H. Crawford, and J. J. Beaman, "Volumetric Multi-Texturing for Functionally Gradient Material Representation," 2001.

[74] R. Jamieson and H. Hacker, "Direct slicing of CAD models for rapid prototyping," *Rapid Prototyp. J.*, vol. 1, no. 2, pp. 4–12, 1995.

[75] Z. Zhao and Z. Luc, "International Journal of Production Research Adaptive direct slicing of the solid model for rapid prototyping," 2010.

[76] X. Chen, C. Wang, X. Ye, Y. Xiao, and S. Huang, "Direct Slicing from PowerSHAPE Models for Rapid Prototyping," *Int J Adv Manuf Technol*, vol. 17, pp. 543–547, 2001.

[77] M. T. Hayasi and B. Asiabanpour, "Machine path generation using direct slicing from design-by-feature solid model for rapid prototyping,"

[78] W. Ma, W.-C. But, and P. He, "NURBS-based adaptive slicing for efficient rapid prototyping," *Comput. Des.*, vol. 36, pp. 1309–1325, 11 2004.

[79] B. Starly, A. Lau, W. Sun, W. Lau, and T. Bradbury, "Direct slicing of STEP based NURBS models for layered manufacturing," *Comput. Des.*, vol. 37, pp. 387–397, 4 2005.

[80] Y. Qiu, X. Zhou, and X. Qian, "Direct slicing of cloud data with guaranteed topology for rapid prototyping,"

[81] S. Sikder, A. Barari, and H. A. Kishawy, “Global adaptive slicing of NURBS based sculptured surface for minimum texture error in rapid prototyping,” Tech. Rep. 6, 2015.

[82] Y. Sasaki, M. Takezawa, S. Kim, H. Kawaharada, and T. Maekawa, “Adaptive direct slicing of volumetric attribute data represented by trivariate B-spline functions,” *Int J Adv Manuf Technol*, vol. 91, pp. 1791–1807, 2017.

[83] J. Feng, J. Fu, Z. Lin, C. Shang, and B. Li, “Rapid Prototyping Journal Direct slicing of T-spline surfaces for additive manufacturing Article information,” tech. rep.

[84] D. W. Adams and C. J. Turner, “Implicit slicing method for additive manufacturing processes,” tech. rep.

[85] D. Adams and C. J. Turner, “An implicit slicing method for additive manufacturing processes, Virtual and Physical Prototyping,” vol. 13, no. 1, pp. 2–7, 2018.

[86] J. C. Steuben, A. P. Iliopoulos, and J. G. Michopoulos, “Implicit slicing for functionally tailored additive manufacturing,” *Computer-Aided Design*, vol. 77, pp. 107–119, 8 2016.

[87] T. Lozano-Perez and P. A. O’Donnell, “Parallel robot motion planning,” in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, 1991.

[88] Y.-a. Jin, Y. He, G.-h. Xue, and J.-z. Fu, “A parallel-based path generation method for fused deposition modeling,” *The International Journal of Advanced Manufacturing Technology*, vol. 77, no. 5-8, pp. 927–937, 2015.

[89] E. Lee, “Contour offset approach to spiral toolpath generation with constant scallop height,” *CAD Computer Aided Design*, vol. 35, no. 6, pp. 511–518, 2003.

[90] Z. Lin, J. Fu, Y. He, and W. Gan, “A robust 2D point-sequence curve offset algorithm with multiple islands for contour-parallel tool path,” *CAD Computer Aided Design*, vol. 45, no. 3, pp. 657–670, 2013.

[91] B. K. Choi and S. C. Park, “A pair-wise offset algorithm for 2D point-sequence curve,” *Computer-Aided Design*, vol. 31, no. 12, pp. 735–745, 1999.

[92] C. S. Lee, T. T. Phan, and D. S. Kim, “2D curve offset algorithm for pockets with Islands using a vertex offset,” *International Journal of Precision Engineering and Manufacturing*, vol. 10, no. 2, pp. 127–135, 2009.

[93] Y. Yang, H. Loh, J. Fuh, and Y. Wang, “Equidistant path generation for improving scanning efficiency in layered manufacturing,” *Rapid Prototyping Journal*, vol. 8, no. 1, pp. 30–37, 2002.

[94] G. Q. Jin, W. D. Li, L. Gao, and K. Popplewell, “A hybrid and adaptive tool-path generation approach of rapid prototyping and manufacturing for biomedical models,” *Computers in Industry*, vol. 64, no. 3, pp. 336–349, 2013.

[95] M. Dolen and U. Yaman, “New morphological methods to generate two-dimensional curve offsets,” *International Journal of Advanced Manufacturing Technology*, vol. 71, no. 9-12, pp. 1687–1700, 2014.

[96] U. Yaman and M. Dolen, “A gradient-based morphological method to produce planar curve offsets,” *International Journal of Advanced Manufacturing Technology*, vol. 80, no. 1-4, pp. 255–274, 2015.

[97] D. Dudley, W. M. Duncan, and J. Slaughter, “Emerging digital micromirror device (DMD) applications,” no. January 2003, p. 14, 2003.

[98] J. Gardan, “Additive manufacturing technologies: state of the art and trends,” *International Journal of Production Research*, vol. 54, no. 10, pp. 3118–3132, 2016.

[99] S.-H. Ahn, M. Montero, D. Odell, S. Roundy, and P. K. Wright, “Rapid Prototyping Journal Anisotropic material properties of fused deposition modeling ABS Article information,” tech. rep.

[100] S. Bhatia, V. Vira, D. Choksi, and P. Venkatachalam, “Geo-spatial Information Science An algorithm for generating geometric buffers for vector feature layers,” *Online) Journal*, pp. 1993–5153, 2013.

[101] “How Buffer (Analysis) works—Help | ArcGIS Desktop.”

[102] “Raspberry Pi 3 Model B Product Description | www.rs-components.com.”

[103] “PLA vs. ABS: What’s the difference? | 3D Hubs.”

[104] “Ultimaker 2 Go Factsheet | Ultimaker.com.”

[105] Creations and LLC, “B9Creator v1.2 Hardware Assembly Manual,” tech. rep., 2014.

APPENDIX A

OFFSET ALGORITHMS

The Python scripts for offset algorithms compared in Section 3.8 is presented in appendix A. From these methods, since Shapely was chosen to be employed in the LIPRO, its Python scripts are delivered in appendix B.

A.1 IMOBS

```
import numpy as np
import itertools
from scipy import spatial

def IMOBS(c,y,r):
    ### optimizing parameters
    def Iopt_prmt(B,eps,r):
        delta1=[]
        for i in range(len(B)):
            delta1.append(abs(B[i]-B[i-1]))
        L=sum(delta1)
        delta=np.sqrt(1*eps*r)
        P=int(L/delta)
        N=int(round(14*np.pi*r/delta,-1))
        if P<=0:
            P=10
        print(P);print(N)
        return P,N,L
    #Increasingpoint function
    def ssample(X,Y,P):
        s=np.insert(np.cumsum(np.absolute(np.add(np.diff(X)
        ,1j*np.diff(Y)))),0,0)
        if P<1:P=np.ceil(s[-1]/P)
        n=len(X);delta=s[-1]/P
        xs=np.zeros((P+1,1));ys=np.zeros((P+1,1))
        xs[0]=X[0];ys[0]=Y[0]
        for i in range(1,P+1):
```

```
            sr=i*delta;j=np.sum(s<sr)
            if j!=n:
                u=(sr-s[j-1])/(s[j]-s[j-1])
                dx=X[j]-X[j-1];dy=Y[j]-Y[j-1]
                xs[i]=X[j-1]+u*dx;ys[i]=Y[j-1]+u*dy
            else: xs[i]=X[-1];ys[i]=Y[-1]
        return xs,ys
    #The function clear will remove empty lists from S
    def clear(A):
        n=len(A);s=0;k=0
        for i in range(n):
            s+=int(not A[i])
        W=[[0 for i in range(1)]for j in range(n-s)]
        for i in range(n):
            if A[i]:
                W[k]=A[i]
                k+=1
        return W
    #MFO phase
    def mfo(Qi,mfo_r):
        Qc=[]
        while Qi:
            Q=Qi[0]
            del(Qi[0])
            b=Q[0]
            e=Q[-1]
            delta=0
            while delta<mfo_r and Qi:
                dl11=[];dl21=[];dl31=[];dl41=[]
                for j in range(len(Qi)):
                    dl11.append(abs(b-Qi[j][0]))
                    dl21.append(abs(b-Qi[j][-1]))
                    dl31.append(abs(e-Qi[j][0]))
                    dl41.append(abs(e-Qi[j][-1]))
                dl1=min(dl11);idx1=dl11.index(dl1)
                dl2=min(dl21);idx2=dl21.index(dl2)
                dl3=min(dl31);idx3=dl31.index(dl3)
                dl4=min(dl41);idx4=dl41.index(dl4)
                delta=min(dl1,dl2,dl3,dl4)
                if delta>mfo_r:
                    Qc.append(Q)
                else:
                    if delta==dl1:
                        Q=Qi[idx1][::-1]+Q
                        del(Qi[idx1])
                        b=Q[0]
                    elif delta==dl2:
                        Q=Qi[idx2]+Q
                        del(Qi[idx2])
                        b=Q[0]
```

```
                elif delta==dl3:
                    Q.extend(Qi[idx3])
                    del(Qi[idx3])
                    e=Q[-1]
                elif delta==dl4:
                    Q.extend(Qi[idx4][::-1])
                    del(Qi[idx4])
                    e=Q[-1]
        if len(Qi)==0:
            Qc.append(Q)
    return Qc
def IMOBS_offset(x,y,r):
    eps=0.05 # error tolerance band
    B = [x+y*1j for x,y in zip(x,y)]
    res = 1;mfo_r = 1.8*r
    P,N,L = Iopt_prmt(B,eps,r)
    X,Y=ssample(np.array(x),np.array(y),P)
    y1=1j*Y
    x=list(itertools.chain(*X.tolist()))
    y=list(itertools.chain(*y1.tolist()))
    ### CBS of IMOBS)
    S=[[]for c in range(len(x))]
    #making T0 for Tk and eqn7
    T0=(r*np.exp(1j*np.linspace(0,2*np.pi,num=N+1)))
    S[0]=np.add(np.add(x[0],y[0]),T0).tolist()
    for k in range(1,len(x)-1):
        #angle of two consecutive points for a specific k
        ukp=(x[k+1]-x[k]+y[k+1]-y[k])/abs(x[k+1]-x[k]+y[k+1]-y[k])
        ukn=(x[k]-x[k-1]+y[k]-y[k-1])/abs(x[k]-x[k-1]+y[k]-y[k-1])
        xkp=ukp.real;ykp=ukp.imag
        xkn=ukn.real;ykn=ukn.imag
        chi=(xkp*xkn)+(ykp*ykn)
        #Tk will be:
        Tk=np.add(np.add(x[k],y[k]),T0)
        Tk=Tk.tolist()
        #condition to see whether to use eqn6 or eqn7
        if chi>res: #from eqn6
            xs=x[k]-r*ykp
            ys=y[k]+r*xkp*1j
            S[k].append(xs+ys)
            xs=x[k]+r*ykp
            ys=y[k]-r*xkp*1j
            S[k].append(xs+ys)
        else: #from eqn7
            for i in range(len(Tk)):
                if abs(Tk[i]-(x[k+1]+y[k+1]))>r and abs(
                        Tk[i]-(x[k-1]+y[k-1]))>r:
                    S[k].append(Tk[i])
    alpha=[[]for h in range(len(S))]
    #Grid search and removing invalid points
```

```
        B=[x+y for x,y in zip(x,y)]
        S0=[[]for h in range(len(S))]
        A=np.column_stack((X,Y))
        A=A.tolist()
        T=spatial.cKDTree(A)
        for k in range(int(len(A))):
            idx=T.query_ball_point(A[k],2*r)
            alpha[k]=idx
            for i in range(len(S[k])):
                for j in range(len(idx)):
                    if abs(S[k][i]-B[idx[j]])<r:
                        S0[k].append(S[k][i])
            S[k]=[a for a in S[k] if a not in S0[k]]
        #creation of curve offsets
        S=clear(S)
        Qi=[]
        Q=[]
        while S:
            Qi.append(Q)
            Q=[]
            for k in range(len(S)):
                qmin=0
                while qmin<1.4*r and S[k]:
                    if not Q:
                        q=S[0][0]
                    qabs=[];qlist=[]
                    for j in range(len(S[k])):
                        qabs.append(abs(S[k][j]-q))
                        qlist.append(S[k][j])
                    qmin=min(qabs)
                    idx=qabs.index(qmin)
                    qq=qlist[idx]
                    if qmin<1.4*r:
                        Q.append(q)
                        q=qq
                        del(S[k][idx])
            S=clear(S)
            if len(S)==0:
                Qi.append(Q)
        del(Qi[0])
        Qc = mfo(Qi,mfo_r)
        return Qc
    X1 = [];Y1 = [];j = 0
    B = [x+y*1j for x,y in zip(x,y)]
    for i in range(len(B)-1):
        if abs(B[i+1]-B[i]) > 2*r:
            X1.append(x[j:i+1]);Y1.append(y[j:i+1])
            j = i+1
    X1.append(x[j:]);Y1.append(y[j:])
    Qcc = []
```

```
    for i in range(len(X1)):
        Qcc.append(IMOBS_offset(X1[i],Y1[i],r))
    return Qcc
```

A.2 AMOBS

```
import numpy as np
import itertools
from scipy import spatial
import math

def AMOBS(x,y,r):
    ### this function find the two parameters based on the curvature
    def weight(B,P,L):
        h = []
        for k in range(1,len(B)-1):
            if abs(B[k+1]-B[k]) != 0 and abs(B[k]-B[k-1]) != 0:
                ukp=(B[k+1]-B[k])/abs(B[k+1]-B[k])
                ukn=(B[k]-B[k-1])/abs(B[k]-B[k-1])
                xkp=ukp.real;ykp=ukp.imag
                xkn=ukn.real;ykn=ukn.imag
                cost=(xkp*xkn)+(ykp*ykn)
                h.append(abs(cost))
        g = min(h)
        #avg = sum(h)/len(h)
        nq = int((0.0025*g**2+0.0005)*P)
        if nq == 0:
            nq = 5
        if g == 0:
            w = 1
        else: w = 0.7*math.sqrt(1-g**2)+0.8
        print(w);print(nq)
        return w,nq
    def ssample(X,Y,p):
        s=np.insert(np.cumsum(np.absolute(
                np.add(np.diff(X),1j*np.diff(Y)))),0,0)
        if p<1:p=np.ceil(s[-1]/p)
        n=len(X);delta=s[-1]/p
        xs=np.zeros((p+1,1));ys=np.zeros((p+1,1))
        xs[0]=X[0];ys[0]=Y[0]
        for i in range(1,p+1):
            sr=i*delta;j=np.sum(s<sr)
            if j!=n:
                u=(sr-s[j-1])/(s[j]-s[j-1])
                dx=X[j]-X[j-1];dy=Y[j]-Y[j-1]
                xs[i]=X[j-1]+u*dx;ys[i]=Y[j-1]+u*dy
```

```
            else: xs[i]=X[-1];ys[i]=Y[-1]
        return xs,ys
    #This function will remove empty lists from S
    def clear(A):
        n=len(A);s=0;k=0
        for i in range(n):
            s+=int(not A[i])
        W=[[0 for i in range(1)]for j in range(n-s)]
        for i in range(n):
            if A[i]:
                W[k]=A[i]
                k+=1
        return W
    ### finding direction of a set of points
    def drctn(Q,nq):
        cost = 0; sint = 0
        cos = 0 ; sin = 0
        for m in range(nq-1):
            cos = (Q[m+1].real - Q[m].real)/(abs(Q[m+1]-Q[m]))
            sin = (Q[m+1].imag - Q[m].imag)/(abs(Q[m+1]-Q[m]))
            cost += cos
            sint += sin
        return cost/nq, sint/nq
    ### length of a curve
    def length(B):
        delta2 = []
        for i in range(len(B)):
            delta2.append(abs(B[i]-B[i-1]))
        return sum(delta2)
    def number_of_P(x,y,B,eps,r):
        mxx = max(x);mnx = min(x);mxy = max(y);mny = min(y)
        Sf = (mxx-mnx)*(mxy-mny)
        delta2 = []
        for i in range(len(B)):
            delta2.append(abs(B[i]-B[i-1]))
        L=sum(delta2)
        cr = (0.1-L/Sf)*(2*r)+0.2
        if cr<=0: cr = 0.08
        delta3=np.sqrt(cr*eps*r)
        P=int(L/delta3)
        return P,L
    #MFO phase
    def mfo(Qi,mfo_r):
        Qc=[]
        while Qi:
            Q=Qi[0]
            del(Qi[0])
            b=Q[0]
            e=Q[-1]
            delta=0
```

```
        while delta<mfo_r and Qi:
            dl11=[];dl21=[];dl31=[];dl41=[]
            for j in range(len(Qi)):
                dl11.append(abs(b-Qi[j][0]))
                dl21.append(abs(b-Qi[j][-1]))
                dl31.append(abs(e-Qi[j][0]))
                dl41.append(abs(e-Qi[j][-1]))
            dl1=min(dl11);idx1=dl11.index(dl1)
            dl2=min(dl21);idx2=dl21.index(dl2)
            dl3=min(dl31);idx3=dl31.index(dl3)
            dl4=min(dl41);idx4=dl41.index(dl4)
            delta=min(dl1,dl2,dl3,dl4)
            if delta>mfo_r:
                Qc.append(Q)
            else:
                if delta==dl1:
                    Q=Qi[idx1][::-1]+Q
                    del(Qi[idx1])
                    b=Q[0]
                elif delta==dl2:
                    Q=Qi[idx2]+Q
                    del(Qi[idx2])
                    b=Q[0]
                elif delta==dl3:
                    Q.extend(Qi[idx3])
                    del(Qi[idx3])
                    e=Q[-1]
                elif delta==dl4:
                    Q.extend(Qi[idx4][::-1])
                    del(Qi[idx4])
                    e=Q[-1]
        if len(Qi)==0:
            Qc.append(Q)
    return Qc
def AMOBS_offset(x,y,N=20,r=1):
    eps=0.05 # error tolerance band
    B = [x+y*1j for x,y in zip(x,y)]
    P,L = number_of_P(x,y,B,eps,r)
    w,nq = weight(B,P,L)
    X,Y=ssample(np.array(x),np.array(y),P)
    x = list(itertools.chain(*X.tolist()))
    y = list(itertools.chain(*Y.tolist()))
    B = [x+y*1j for x,y in zip(x,y)]
    if w != 1:
        w,nq = weight(B,P,L)
    delta = max((np.absolute(np.add(np.diff(x),
                                    1j*np.diff(y)))).tolist())
    S = [[] for _ in range(len(x))]
    for k in range(len(x)-1):
        if abs(B[k]-B[k-1]) <= delta:
```

```
            if x[k-1] <= x[k] and x[k] <= x[k+1]:
                if (x[k+1]-x[k])==0:
                    if (y[k+1]-y[k]) < 0:
                        atan1 = -math.pi/2
                    else: atan1 = math.pi/2
                else: atan1 = math.atan((y[k+1]-y[k])/
                                        (x[k+1]-x[k]))
                if (x[k]-x[k-1])==0:
                    if (y[k]-y[k-1]) < 0:
                        atan2 = -math.pi/2
                    else:
                        atan2 = math.pi/2
                else: atan2 = math.atan((y[k]-y[k-1])/
                                        (x[k]-x[k-1]))
                aplus1 = atan1+math.acos(abs(B[k+1]-
                                             B[k])/(2*r))
                amines1 = atan1-math.acos(abs(B[k+1]-
                                              B[k])/(2*r))
                aplus2 = math.pi+atan2-
                math.acos(abs(B[k]-B[k-1])/(2*r))
                amines2 = -math.pi+atan2+
                math.acos(abs(B[k]-B[k-1])/(2*r))
            elif x[k-1]-x[k] < 0 and x[k+1]-x[k] < 0:
                atan1 = math.pi +
                math.atan((y[k+1]-y[k])/(x[k+1]-x[k]))
                atan2 = math.pi +
                math.atan((y[k]-y[k-1])/(x[k]-x[k-1]))
                aplus1 = atan1+
                math.acos(abs(B[k+1]-B[k])/(2*r))
                amines1 = atan1-
                math.acos(abs(B[k+1]-B[k])/(2*r))
                aplus2 = atan2-
                math.acos(abs(B[k]-B[k-1])/(2*r))
                amines2 = (-math.pi*2)+atan2+math.acos(
                        abs(B[k]-B[k-1])/(2*r))
            elif x[k-1]-x[k] > 0 and x[k+1]-x[k] > 0:
                atan1 = math.atan((y[k+1]-
                                   y[k])/(x[k+1]-x[k]))
                atan2 = math.atan((y[k]-y[k-1])/
                                  (x[k]-x[k-1]))
                aplus1 = atan1+math.acos(abs(B[k+1]-
                                             B[k])/(2*r))
                amines1 = atan1-math.acos(abs(B[k+1]-
                                              B[k])/(2*r))
                aplus2 = atan2-math.acos(abs(B[k]-
                                             B[k-1])/(2*r))
                amines2 = (-math.pi*2) + atan2+math.acos(
                        abs(B[k]-B[k-1])/(2*r))
            else:
                if (x[k-1]-x[k])==0:
```

```
                if (y[k-1]-y[k]) < 0:
                    atan1 = -math.pi/2
                else: atan1 = math.pi/2
            else: atan1 = math.atan((y[k-1]-y[k])/
                                    (x[k-1]-x[k]))
            if (x[k]-x[k+1])==0:
                if (y[k]-y[k+1]) < 0:
                    atan2 = -math.pi/2
                else:
                    atan2 = math.pi/2
            else: atan2 = math.atan((y[k]-y[k+1])/
                                    (x[k]-x[k+1]))
            aplus1 = atan1+math.acos(abs(B[k-1]-
                                         B[k])/(2*r))
            amines1 = atan1-math.acos(abs(B[k-1]-
                                          B[k])/(2*r))
            aplus2 = math.pi+atan2-
            math.acos(abs(B[k]-B[k+1])/(2*r))
            amines2 = -math.pi+atan2+
            math.acos(abs(B[k]-B[k+1])/(2*r))
        A = [];nplus=0;nmines=0
        if (aplus2 - aplus1) < 0:
            pass
        else:
            if abs(aplus2 - aplus1) <= (delta/(1*r)):
                A.append((aplus1+aplus2)/2)
            else:
                nplus = math.ceil(r*abs(aplus2-
                                        aplus1)/delta)
                dphiplus = ((aplus2-aplus1)-
                            (delta/r))/(nplus)
                i = 0
                while i < nplus:
                    i += 1
                    aplus = aplus1+(delta/(10*r))+
                                   (i-1)*dphiplus
                    A.append(aplus)
        if -(amines2 - amines1) < 0:
            pass
        else:
            if abs(amines2 - amines1) <= (delta/(1*r)):
                A.append((amines1+amines2)/2)
            else:
                nmines = math.ceil(r*abs(
                        amines2-amines1)/delta)
                dphimines = (-(amines2-amines1)-
                             (delta/r))/(nmines)
                i = 0
                while i < nmines:
                    i += 1
```

```
                            amines = amines1-(
                                    delta/(10*r))-(i-1)*dphimines
                            A.append(amines)
                for a in A:
                    S[k].append(((x[k]+r*
                     math.cos(a))+1j*(y[k]+r*math.sin(a))))
            else:
                T0=(r*np.exp(1j*np.linspace(0,2*np.pi,num=N+1)))
                Tk=np.add(np.add(x[k],1j*y[k]),T0)
                Tk=Tk.tolist()
                for i in range(len(Tk)):
                    S[k].append(Tk[i])
        ### removing the invalid points
        A = np.column_stack((X,Y))
        T = spatial.cKDTree(A)
        for k in range(int(len(A))):
            idx=T.query_ball_point(A[k],2*r)
            for a in idx:
                for m in S[a]:
                    if abs(m-B[k])<r:
                        S[a].remove(m)
        x1 = x
        y1 = y
        for i in range(len(S)):
            if not S[i]:
                x1[i] = []
                y1[i] = []
        S = clear(S)
        x1 = clear(x1)
        y1 = clear(y1)
        ### CCO of AMOBS
        Qi = []
        Q = [[]]
        A = np.column_stack((np.array(x1),np.array(y1)))
        T = spatial.cKDTree(A)
        z = [9]
        while z:
            q = 0
            z = clear(S)
            Qi.append(Q)
            Q=[]
            if not Q:
                for k in range(len(S)):
                    if S[k]:
                        q = S[k][0]
                        del(S[k][0])
                        break
                if q == 0:
                    break
            qmin = 0
```

```
            while qmin < r:
                idx=T.query_ball_point(A[k],2*r)
                qmin = 2*10**15
                light = False
                if len(Q) >= nq and w != 1:
                    u1, u2 = drctn(Q[-nq:],nq)
                    for a in idx:
                        for b in S[a]:
                            if b != q:
                                e1 = b.real - q.real
                                e2 = b.imag - q.imag
                                dt = (u1*e1+u2*e2)
                                if dt == 0:
                                    b1 = (w/delta)+
                                         ((1-w)/0.00001)
                                else:
                                    b1 = (w/delta)+((1-w)/dt)
                                if abs(b-q)*b1 < qmin:
                                    qmin = abs(b-q)*b1
                                    qq = b
                                    k = a
                                    light = True
                    if light:
                        qmin = abs(qq-q)
                else:
                    for a in idx:
                        for b in S[a]:
                            if b != q:
                                if abs(b-q) < qmin:
                                    qmin = abs(b-q)
                                    qq = b
                                    k = a
                if qmin < r:
                    if q not in Q:
                        Q.append(q)
                    q = qq
                    S[k].remove(q)
        del(Qi[0])
        Qi = clear(Qi)
        mfo_r=1.9*r
        Qc = mfo(Qi,mfo_r)
        for a in Qc:
            if abs(a[-1]-a[0]) < 1.5*r:
                a.append(a[0])
        return Qc
    X1 = [];Y1 = [];j = 0
    B = [x+y*1j for x,y in zip(x,y)]
    for i in range(len(B)-1):
        if abs(B[i+1]-B[i]) > 2*r:
            X1.append(x[j:i+1]);Y1.append(y[j:i+1])
```

```
            j = i+1
    X1.append(x[j:]);Y1.append(y[j:])
    Qcc = []
    for i in range(len(X1)):
        Qcc.append(AMOBS_offset(X1[i],Y1[i],N,r))
    return Qcc
```

A.3 Simplification

```
from numpy.linalg import norm
import numpy as np
### Find the prependicular distance between point 3 and line generated
### from point 1 and 2
def deviation(p1,p2,p3):
    return  norm(np.cross(np.subtract(p2,p1),
                          np.subtract(p3,p1)))/norm(np.subtract(p2,p1))
### Filter the points based on their deviations
### Inputs: x and y coordinates and the amount of deviation threshold
def Filter(x,y,e):
    iddx = []
    for i in range(1,len(x)-1):
        if deviation([x[i-1],y[i-1]],[x[i+1],y[i+1]],[x[i],y[i]]) < e:
            iddx.append(i)
    for a in iddx[::-1]:
        del(x[a]);del(y[a])
    return x,y           ### Returns simplified x,y coordinates
```

APPENDIX B

The Python scripts for the LIPRO is presented here as a library. It is composed of the functions described as the structure of the LIPRO in Chapter 3 and operations required to implement on the FDM and DLP printers.

B.1 LIPRO Library

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 15:51:26 2018
@author: V.Haseltalab
"""
import math
import shapely.geometry as sg
import numpy as np
import serial
import pygame,os
from Adafruit_MotorHAT import Adafruit_MotorHAT,
Adafruit_DCMotor, Adafruit_StepperMotor
import time
import atexit
import RPi.GPIO as gp
from PIL import Image
from stl import mesh
from scipy import spatial
import matplotlib.pyplot as plt

### Generating polygon
### Input: radius of the polygon, Number of points in polygon
### and the starting angle of the polygon
def gpc(r,N,angle=0):
    out = [0]*2
    angle = np.pi*angle/180
    circle = r * np.exp(1j * (np.linspace(
```

```
            0, 2 * math.pi, N + 1) + angle))
    out[0] = circle.real; out[1] = circle.imag;
    np.put(out[0],[-1],out[0][0]); np.put(out[1],[-1],out[1][0])
    return out
### Generate offset polygons
### Input arguments are x,y coordinates of the base curve,
### number of offsets and offset distance
def offset(x,y,N,d):
    ### providing (x,y) tuples together
    if any(isinstance(el, list) for el in x):
        poly=[[] for j in range(len(x))]
        B=[[] for _ in range(len(x))]
        for i in range(len(x)):
            B[i].extend([(x,y) for x,y in zip(x[i],y[i])])
    else:
        poly = []
        B = [(x,y) for x,y in zip(x,y)]
    X = [];Y = []
    ### creating seperated polygons
    if any(isinstance(el, list) for el in x):
        for i in range(len(poly)):
            poly[i]=sg.Polygon(B[i])
        for k in range(len(poly)):
            for j in range(N):
                s = poly[k].buffer(-d*j)
                if isinstance(s,sg.multipolygon.MultiPolygon):
                    for i in range(len(s)):
                        x1,y1 = s[i].exterior.coords.xy
                        X.append(x1);Y.append(y1)
                else:
                    s1 = np.array(s.exterior)
                    if s1.any():
                        x1, y1 = s.exterior.coords.xy
                        X.append(x1);Y.append(y1)
    else:
        poly=sg.Polygon(B)
        for i in range(N):
            s = poly.buffer(-d*i)
            if isinstance(s,sg.multipolygon.MultiPolygon):
                for i in range(len(s)):
                    x1,y1 = s[i].exterior.coords.xy
                    X.append(x1);Y.append(y1)
            else:
                s1 = np.array(s.exterior)
                if s1.any():
                    x1,y1 = s.exterior.coords.xy
                    X.append(x1);Y.append(y1)
    return X,Y
### Slicing an STL file format. It requires STL address,
### layer thickness and the height of slice plane
```

```
def slicing(addr,thickness = 0.07,z = 1):
    sb = 0.5 ## search bound
    def vertices(a):  ### Input argument a as address of the stl file
        # importing ths stl file
        msh = mesh.Mesh.from_file('%s'%a)
        zvalues = [] ### stores the z value of a vertex
        ### categorizing the vertices into a list based on their faces
        vrt = [[] for s in range(len(msh))]
        for i in range(len(msh)):
            p1 = (msh[i][:3]).tolist()
            p2 = (msh[i][3:6]).tolist()
            p3 = (msh[i][6:]).tolist()
            zvalues.append(p1[-1]);zvalues.append(p2[-1])
            zvalues.append(p3[-1])
            vrt[i] = [p1,p2,p3]
        return vrt,zvalues
        ### Returns a list of vertices along with their z values

    ### finds intersection coordinates between a point and a line
    ### p1 and p2 are two points generating a line
    ### and z is the slice plan presented as a point
    def eqn(p1,p2,z):
        ### the ratio that can apply to distance of x and y
        t = (z-p1[2]) / (p2[2]-p1[2])
        return [p1[0] + (p2[0]-p1[0])*t , p1[1] + (p2[1]-p1[1])*t]
    ### returns coordinates of the intersection point

    ### checks whether the z plane is crossing through the line
    def checkline(zl,z):
        if z < max(zl) and z > min(zl):
            return True
        else: return False

    ### finds intersection coordinates between
    ### a plane and a triangular facet
    def trintersct(l,z):
        inlst = []
        for i in range(3):
            pt1 = l[i];pt2 = l[i-1]
            zl = [pt1[2],pt2[2]]
            if checkline(zl,z):
                p = eqn(pt1,pt2,z)
                inlst.append(p)
        return inlst
    #The function clear will remove empty lists from S
    def clear(A):
        n=len(A);s=0;k=0
        for i in range(n):
            s+=int(not A[i])
        W=[[0 for i in range(1)]for j in range(n-s)]
```

```
        for i in range(n):
            if A[i]:
                W[k]=A[i]
                k+=1
        return W
    ### Ordering the points in a correct sequence
    def order(L,r):
        l = [[] for _ in range(5)]
        i = 0
        p = L[0]
        del(L[0])
        T1 = spatial.cKDTree(L)
        l[i].append(p)
        while L:
            ds, idx = T1.query(p)
            pp = L[idx]
            if ds > r:
                i += 1
            l[i].append(pp)
            del(L[idx])
            if not L:
                break
            T1 = spatial.cKDTree(L)
            p = pp
        l = clear(l)
        for a in l:
            a.append(a[0])
        return l
    vrt, zvalues = vertices(addr)
    one = [1 for f in range(len(zvalues))]
    A=np.column_stack((np.array(zvalues),np.array(one)))
    T = spatial.cKDTree(A.tolist())
    fnum = []
    Q = []
    planept = [z,1]
    idx=T.query_ball_point(planept,sb)
    for a in idx:
        fidx = int(a/3)
        if fidx not in fnum:
            fnum.append(fidx)
            Q.extend(trintersct(vrt[fidx],z))
    if Q:
        l = order(Q,4)
        return l
    else:
        print("No intersections!")
        return []

#### Functions specifically used in FDM printer

```

```
### Generating G-code
### Input: two lists of x,y coordinates and nozzle height Z
### Returns a list of strings as G-codes
def u2g(X,Y,Z):
    main = []
    extruderRatio = 0.12  #0.008
    extruderPos = 0
    offsetX = 40
    offsetY = 40
    for j in range(len(X)):
        main.append("G1 F6000 E%.3f\n" % (extruderPos - 2))
        main.append("\nG0 F3600 X%.3f Y%.3f Z%.3f\n" % (
                X[j][0] + offsetX, Y[j][0] + offsetY, Z))
        main.append("G1 F6000 E%.3f\n" % (extruderPos))
        for i in range(len(X[j])-1):
            distance = math.sqrt(math.pow(X[j][i] - X[j][i+1],2) +
                                 math.pow(Y[j][i] - Y[j][i+1],2))
            extruderPos += distance * extruderRatio
            main.append("G1 F1800 X%.3f Y%.3f E%.3f\n" % (
                    X[j][i+1]  + offsetX,
                     Y[j][i+1] + offsetY, extruderPos))

    main.append("G92 E0.0\n")
    return (main)

### removing curves inside to maintain the wall
### Input arguments are the x,y coordinates
def rem_ins(x,y):
    if any(isinstance(el, list) for el in x) and len(x)>1:
        poly=[[] for j in range(len(x))]
        B=[[] for _ in range(len(x))]
        pt = [[] for _ in range(len(x))]; g = []
        for j in range(len(x)):
            B[j].extend([(x,y) for x,y in zip(x[j],y[j])])
        for k in range(len(poly)):
            poly[k]=sg.Polygon(B[k])
            pt[k] = sg.Point(B[k][0])
        for n in range(len(pt)):
            for m in range(len(poly)):
                if pt[n].within(poly[m]):
                    g = m
        if isinstance(g,int):
            x = x[g]; y = y[g]
    return [x,y]

### Connect with Arduino
### Inputs: Port address, required buad rate and timeout
### Delivers a variable sr defining the connection port
def connection (com, baudrate, tout):
    # Open serial port
```

```
    sr = serial.Serial('%s'%com, baudrate, timeout = tout)
    return sr
### Wait until operation is over and reset the buffer
def readsg(s):
    srs = 'no signals'.encode()
    while srs != 'ok'.encode():
        srs = s.readline().strip()
    s.reset_input_buffer()
### Initial measures
def strt_confg(s):
    s.write(('M109 S210'+'\n').encode())
    print(type(s))
    s.write(('G28'+'\n').encode())
    readsg(s)
    s.write(('G0 F3600 X-10.0 Y11.800 Z20.0'+'\n').encode())
    readsg(s)
    s.write(('G92 E0.0'+'\n').encode())
    readsg(s)
    s.write(('G1 F100 E25'+'\n').encode())
    readsg(s)
    s.write(('G92 E0.0'+'\n').encode())
    readsg(s)
    s.write(('M106 S255'+'\n').encode())
    readsg(s)
### Turn on the side fans
def fan_on(s):
    s.write(('M106 S255'+'\n').encode())
    print(('M106 S255'+'\n').encode())
    readsg(s)
def fan_off(s):
    s.write(('M106 S0'+'\n').encode())
    readsg(s)
### Apply end settings
def end_confg(s):
    s.write(('G1 F6000 E-10.0'+'\n').encode())
    readsg(s)
    s.write(('G28'+'\n').encode())
    readsg(s)
    s.write(('M104 S0.0'+'\n').encode())
    readsg(s)
    s.write(('M106 S0'+'\n').encode())
    readsg(s)
### Terminate the connection
def closing(s):
    s.close()
### Retract the filament
def retract(s):
    s.write(('G1 F6000 E-10.0'+'\n').encode())
### Send G-codes line by line to the machine
### l is representing the list of G-codes
```

```
def txt(l,s):
    srv = 'no signals'.encode()
    for e in l:
        s.write(e.encode())
        while srv != 'ok'.encode():
            srv = s.readline().strip()
        s.reset_input_buffer()
        srv = 'no signals'.encode()
### Main function of printing process
def FDMPrint(radius,height,s):
    t = 0.06 ## Layer thickness
    d = 0.35 ## Offset distance
    angle = 0.3 ## twisting angle in each layer
    diff = t*(radius)/height ### Requires for pyramid
    Z = 0.3 ## initial position of the nozzle
    P = int(height/t) ### Number of layers
    N = 4 ## Polygon type - Number of sides
    n = int(radius/d) ## Number of offsets in each layer
    for i in range(5):  ### Print the bottom layers
        print('Layer Number: %d'%(i+1))
        ### Generate the base polygon
        a = gpc(radius-(diff*i),N,i*angle)
        x = a[0].tolist();y = a[1].tolist()
        x1, y1 = offset(x,y,n,d) ### Generate the offsets
        gcode = u2g(x1,y1,Z) ### Get G-codes from trajectories
        txt(gcode,s)  ### Send G-codes line by line to machine
        Z += t  ### Add one layer thickness
    n = 3
    for i in range(5,P+1):   ### Pring the upper layers
        print('Layer Number: %d'%(i+1))
        a = gpc(radius-(diff*i),N,i*angle)
        if int((radius-(diff*i))/d) < 3:
            n = int((radius-(diff*i))/d)
        x = a[0].tolist();y = a[1].tolist()
        x1, y1 = offset(x,y,n,d)
        gcode = u2g(x1,y1,Z)
        txt(gcode,s)
        Z += t
    ### Lower down the bed
    s.write(('G0 F3600 Z%.3f\n'%(Z+20)).encode())
    readsg(s)

#### Functions specifically used in DLP printer

### Provide the black screen and return as a variable
def Screen():
    os.environ["SDL_VIDEO_CENTERED"] = "1"
    pygame.init()
    #screen = pygame.display.set_mode((1500,800))
    screen = pygame.display.set_mode((0,0),pygame.FULLSCREEN)
```

```python
    pygame.display.set_caption('DLP Print')
    pygame.mouse.set_visible(0)
    screen.fill((0,0,0))
    return screen
### Define DC motor and Stepper motor and return both of them
def MotorStart():
    ### create a default object
    mh = Adafruit_MotorHAT(addr=0x70)
    ### No changes to I2C address or frequency
    mhstep = Adafruit_MotorHAT()
    ### recommended for auto-disabling motors on shutdown!
    def turnOffMotors():
        mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)
    atexit.register(turnOffMotors)
    myMotor = mh.getMotor(3)
    ### 200 steps/rev, motor port #1
    myStepper = mhstep.getStepper(200, 1)
    return myMotor,myStepper
### Motion of vat to right side, Input argument is DC motor
def XRmove(myMotor):
    gp.setmode(gp.BCM)
    gp.setwarnings(False)
    gp.setup(4,gp.IN,pull_up_down=gp.PUD_UP)
    ### set the speed to start, from 0 (off) to 255 (max speed)
    myMotor.setSpeed(50)
    myMotor.run(Adafruit_MotorHAT.FORWARD);
    ### turn on motor
    myMotor.run(Adafruit_MotorHAT.RELEASE);
    myMotor.run(Adafruit_MotorHAT.FORWARD)
    running = True
    try:
        while running:
            A = gp.input(4)
            myMotor.setSpeed(70)
            if A:
                running = False
    except KeyboardInterrupt:
        myMotor.run(Adafruit_MotorHAT.RELEASE)
        gp.cleanup()
    myMotor.run(Adafruit_MotorHAT.RELEASE)
### Motion of vat to left side. Inputs: DC motor and its time
def XLmove(myMotor,xt=1.4):
    myMotor.setSpeed(50)
    myMotor.run(Adafruit_MotorHAT.FORWARD);
    ### turn on motor
    myMotor.run(Adafruit_MotorHAT.RELEASE);
    myMotor.run(Adafruit_MotorHAT.BACKWARD)
```

```
        myMotor.setSpeed(70)
        time.sleep(xt)
        myMotor.run(Adafruit_MotorHAT.RELEASE)
    ### Brings the build table to home position. Input: Stepper motor
    def Zhome(myStepper):
        gp.setmode(gp.BCM)
        gp.setup(17,gp.IN,pull_up_down=gp.PUD_UP)
        myStepper.setSpeed(2000)
        A = gp.input(17)
        if not A:
            running = True
            try:
                while running:
                    A = gp.input(17)
                    myStepper.step(10, Adafruit_MotorHAT.FORWARD,
                                   Adafruit_MotorHAT.INTERLEAVE)
                    if A:
                        running = False
            except KeyboardInterrupt:
                running = False
        else:
            running = True
            try:
                while running:
                    A = gp.input(17)
                    myStepper.step(10, Adafruit_MotorHAT.BACKWARD,
                                   Adafruit_MotorHAT.INTERLEAVE)
                    if not A:
                        running = False
            except KeyboardInterrupt:
                running = False
    ### Move up the build table for defined amount
    ### Inputs are Stepper motor and amount of rotation
    def Z_up(stepmtr,amount):
        stepmtr.setSpeed(2000)             # 30 RPM
        stepmtr.step(amount, Adafruit_MotorHAT.BACKWARD,
                     Adafruit_MotorHAT.SINGLE)
    ### Move down the build table for defined amount
    def Z_down(myStepper,amount):
        myStepper.setSpeed(2000)   ### Set speed
        myStepper.step(amount, Adafruit_MotorHAT.FORWARD,
                       Adafruit_MotorHAT.SINGLE)
    ### The start settings for printing in DLP printer
    def InitialS():
        dc,step = MotorStart()     ### Define motors
        s = Screen()               ### Initiate black screen
        XRmove(dc)             ### Move vat to right
        Zhome(step)            ### Move build table to home position
        wait(s)                    ### Wait for operator
        Z_down(step,5080-300)  ### Lower down the build table
```

```
    return dc, step, s ### Returns DC, stepper motor and screen
### The final settings after printing
def FinalS(dc, step, s):
    Z_up(step,7000)           ### move up the build table
    XRmove(dc)    ### To avoid entering projector light into vat
    wait(s)       ### Wait for operator to turn off the projector
    ScreenOff(s)              ### Terminate the screen

### Calibrate the projector
def CalbProjector():
    os.environ["SDL_VIDEO_CENTERED"] = "1"
    pygame.init()
    #s = pygame.display.set_mode((800,800))
    s = pygame.display.set_mode((0,0),pygame.FULLSCREEN)
    pygame.display.set_caption('DLP Print')
    pygame.mouse.set_visible(0)
    s.fill((0,0,0))
    ### Get the image address
    Img = pygame.image.load(r"/home/pi/122.bmp")
    s.blit(Img, (-100,-60))  ### Display the image
    pygame.display.update() # Update the screen
    pygame.display.flip()
    Clock = pygame.time.Clock()
    running = True
    while running: ### Check for events (press Esc to exit)
        Clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN and
            event.key == pygame.K_ESCAPE:
                running = False
                break
    s.fill((0,0,0))
    pygame.display.update() # Update the screen to black
    pygame.display.flip()
    return s  ### returns the screen
### Build table calibration
def BuildTableCal(myStepper):
    myStepper.setSpeed(1000)             # 30 RPM
    c = 0
    running = True
    try:
        while running:
            myStepper.step(20, Adafruit_MotorHAT.FORWARD,
                               Adafruit_MotorHAT.SINGLE)
            c += 20
            if c > 5080:
                running = False
    except KeyboardInterrupt:
```

```python
            running = False
            print(c)
### Wait for operator command (Esc)
def wait(s):
    s.fill((0,0,0))
    pygame.display.update()
    pygame.display.flip()
    running = True
    try:
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                if event.type == pygame.KEYDOWN
                and event.key == pygame.K_ESCAPE:
                    running = False
                    break
    except KeyboardInterrupt:
        print(running)
    print("Done")
### Wait for operator command (Ctrl+C)
def wait2():
    w = 0
    try:
        while True:
            w += 1
            time.sleep(2)
    except KeyboardInterrupt:
        print(w)
### Check if the point is inside a single polygon
def Inside(a,p1):
    if a.within(p1) or a.touches(p1):
        return True
    else:
        return False
### Check if a point is between two polygons
### p1 is the outerior polygon and p2 is interior
def Inside2(a,p1,p2):
    if (a.within(p1) or a.touches(p1)) and not a.within(p2):
        return True
    else:
        return False
### Locate the polygon into center of the screen
### x,y presents the polygon,
### width and length are the resolution of projector
def center(x,y,width,length):
    B = [(x2,y2) for x2,y2 in zip(x,y)]
    poly = sg.Polygon(B)
    minx,miny,maxx,maxy = poly.bounds
    dis = ((width/2)-(maxx+minx)/2,(length/2)-(maxy+miny)/2)
```

```
    x = (np.array(x) + dis[0]).tolist()
    y = (np.array(y) + dis[1]).tolist()
    return x,y        ### return the new list of coordinates
### Scale the polygon
def scale(x,y):
    x1 = [];y1 = []
    for i in range(len(x)):
        x1.append(x[i]*1080/80)
        y1.append(y[i]*1080/75.6)
    return x1,y1     ### return the new list of coordinates
### Get new coordinates
def NewCoord(x,y):
    x,y = scale(x,y)
    x,y = center(x,y,1920,1080)
    return x,y
### Generate bitmap image.
### Inputs: Length of polygon, starting angle
### if offset is requires withoff must be True, offset distance
def image(L,angle,withoff = False, offdis = 2):
    width = 1920
    height = 1080
    array = np.zeros([height, width, 3], dtype=np.uint8)
    array[:,:] = [0, 0, 0] ### defines a black pixel
    if withoff: ### if Offset requires
        r = L/math.sqrt(2)
        b = gpc(r,4,angle)
        x = b[0].tolist();y = b[1].tolist()
        B = [(x1,y1) for x1,y1 in zip(x,y)]
        poly = sg.Polygon(B) ### Base polygon
        ### Offset polygon
        offset = list(poly.buffer(-offdis).exterior.coords)
        x1 = [b[0] for b in offset]
        y1 = [b[1] for b in offset]
        x,y = NewCoord(x,y) ### new coordinates of base polygon
        x1,y1 = NewCoord(x1,y1)
        B1 = [(x1,y1) for x1,y1 in zip(x,y)]
        B2 = [(x2,y2) for x2,y2 in zip(x1,y1)]
        poly1 = sg.Polygon(B1) ### Define bsae polygon in shapely
        poly2 = sg.Polygon(B2)
        ### Get the boundaries of the base
        minx,miny,maxx,maxy = poly1.bounds
        ### for every Row of screen:
        for i in range(int(miny),int(maxy)):
           ### Generate a line to cover the row
           l = sg.LineString([(int(minx-3),i),(int(maxx+3),i)])
           ### Check if the line intersects with base
           if l.intersects(poly1):
               ### Get intersection
               a = list(l.intersection(poly1).coords)
               ### Check with offset polygon
```

```
               if l.intersects(poly2):
                   a2 = list(l.intersection(poly2).coords)
                    ### Fill the exposed region with white
                   array[i,int(a[0][0]):int(a2[0][0])].fill(255)
                   array[i,int(a2[1][0]):int(a[1][0])].fill(255)
               else:
                   a = list(l.intersection(poly1).coords)
                   array[i,int(a[0][0]):int(a[1][0])].fill(255)
    else:
        r = L/math.sqrt(2)
        b = gpc(r,4,angle)
        x = b[0].tolist();y = b[1].tolist()
        B = [(x1,y1) for x1,y1 in zip(x,y)]
        poly = sg.Polygon(B)
        x,y = NewCoord(x,y)
        B1 = [(x1,y1) for x1,y1 in zip(x,y)]
        poly1 = sg.Polygon(B1)
        minx,miny,maxx,maxy = poly1.bounds
        for i in range(int(miny)-1,int(maxy)+1):
            l = sg.LineString([(int(minx-3),i),(int(maxx+3),i)])
            if l.intersects(poly1):
                a = list(l.intersection(poly1).coords)
                array[i,int(a[0][0]):int(a[1][0])].fill(255)
    return array
### Get images based on indices of the design model
def Pyimage(i,L,diff,offdis,H):
    L = 2*(10-diff*i) ### Must be used if pyramid requires
    num = int(H/0.07)
    if i == 0:
        array = image(L,0,False,2)
        img = Image.fromarray(array)
        data = img.tobytes()
        Img = pygame.image.fromstring(data,img.size,img.mode)
    elif i < 29 or i > num-29: ### No offset for these layers
        array = image(L,0.2*(i+1),False,2)
        img = Image.fromarray(array)
        data = img.tobytes()
        Img = pygame.image.fromstring(data,img.size,img.mode)
    else:
        array = image(L,0.2*(i+1),True,2)
        img = Image.fromarray(array)
        data = img.tobytes()
        Img = pygame.image.fromstring(data,img.size,img.mode)
    return Img            ### returns the bitmap image

### Printing process
### Inputs: screen, DC and stepper motors,
### length, height and offset distance
def DLPPrint(s,dc,myStepper,L,H,offdis):
    lt = 0.07 ## layer thickness
```

```
    diff = lt*L/(2*H) ## radius difference in each layer
    num = int(H/lt) ## number of layers
    Clock = pygame.time.Clock()
    myStepper.setSpeed(2000)    ### Set Stepper motor speed
    running = True
    while running:
        Clock.tick(60)
        s.fill((0,0,0))
        pygame.display.flip()
        for i in range(num):   ### i indicates the layer number
            print(i)   ### Display layer number
            xt = 1.4   ### Time used for moving DC motor
            if i < 3: ct = 21.182 ### Exposure time
            else: ct = 6.637
            s.fill((0,0,0))
            pygame.display.update() # Update the screen to black
            pygame.display.flip()
            time.sleep(0.5)
            ### For initial layers settings are different
            if i < 3 and i != 0:
                Z_up(myStepper,30)
                time.sleep(1)
                XRmove(dc)
                time.sleep(0.5)
                Z_up(myStepper,270)
                Img = Pyimage(i,L,diff,offdis,H)   ### Get image
                XLmove(dc,xt)
                time.sleep(0.5)
                Z_down(myStepper,293)
                time.sleep(1)
            elif i == 0:
                Img = Pyimage(i,L,diff,offdis,H)
                XLmove(dc,xt)
                time.sleep(1)
                Z_down(myStepper,293)
            else:
                Z_up(myStepper,50)
                time.sleep(1)
                XRmove(dc)
                time.sleep(0.5)
                Img = Pyimage(i,L,diff,offdis,H)
                XLmove(dc,xt)
                time.sleep(1)
                Z_down(myStepper,43)
            ### Check for Keyboard events
            for event in pygame.event.get():
                pause = False
                if event.type == pygame.QUIT:
                    running = False
                if event.type == pygame.KEYDOWN and
```

```
                event.key == pygame.K_ESCAPE:
                    running = False
                    break
                if event.type == pygame.KEYDOWN and
                event.key == pygame.K_p:
                    pause = True
                while pause:
                    for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                            pause = False
                            running = False
                        if event.type == pygame.KEYDOWN and
                        event.key == pygame.K_ESCAPE:
                            pause = False
                            running = False
                        if event.type == pygame.KEYDOWN and
                        event.key == pygame.K_p:
                            pause = False
            if not running:
                break
            if i == (num-1): running = False
            s.blit(Img, (0,0))
            ### Update the screen to show image
            pygame.display.update()
            pygame.display.flip()
            time.sleep(ct)   ### Cure with this interval
    s.fill((0,0,0))
    pygame.display.update() ### Update the screen to black
    pygame.display.flip()
### Terminate the black screen
def ScreenOff(s):
    pygame.quit()
```

B.2 Build Table Calibration Main Function

```
import LIPRO as lp
dc,step = lp.MotorStart()    ### Start motors
lp.Zhome(step)               ### Move build table to home position
lp.XLmove(dc,1.4)            ### Move vat to left side
lp.wait2()                   ### Wait for operator
lp.BuildTableCal(step)       ### Build table calibration function
lp.wait2()
lp.Z_up(step,5000)           ### Move the build table up 5 cm
```