• U C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE
ENGENHARIA MECÂNICA

Implementation of a Simulation System for Additive Task Experiments

Submitted in Partial Fulfilment of the Requirements for the Degree of Master in Mechanical Engineering in the speciality of Production and Project

Realização de um Sistema de Simulação de Tarefas de Produção Aditivas

Author

Filipe Monteiro Ribeiro

Advisor[s]

Professor Doutor Joaquim Norberto Cardoso Pires da Silva
Investigador Científico Amin Shahrestani Azar

Jury

President	**Professor Doutor Altino de Jesus Roque Loureiro** **Professor Associado c/ Agregação da Universidade de Coimbra**
Vowel[s]	**Professor Doutor António Fernando Macedo Ribeiro** **Professor Associado c/ Agregação da Universidade do Minho** **Professor Doutor Carlos Xavier Pais Viegas** **Investigador Auxiliar da Universidade de Coimbra**
Advisor	**Professor Doutor Joaquim Norberto Cardoso Pires da Silva** **Professor Associado c/ Agregação da Universidade de Coimbra** **Investigador científico Amin Shahrestani Azar** **CEO na empresa Söhner Kunststofftechnik Gmbh**

Coimbra, Setembro, 2018

“All Models Are Wrong, but Some Are Useful”

(Box, George, 1980)

Aos meus pais

ACKNOWLEDGEMENTS

Without the help of great people, the realization of this thesis would not be possible. Firstly, I want to start by saying that I am grateful for all the support I had from my thesis coordinator, professor J. Norberto Pires. Your knowledge, as well as the way you face new challenges, is an inspiration for always trying to do more and better.

I also want to thank Albert Nubiola and Amin Azar for all the help you gave me during this thesis, by answering some important questions that helped me reach the final solution.

Special thanks for the Kivy development core that helped me every time I was stuck in a problem, however, I have to highlight the fundamental contribution of Dominik since he had the dedication to teach a lot about Python. I just can say that without your contribution, this thesis would not be finished.

To my research's partners, I just want to say that you're the best colleagues I've ever worked and that I will never forget the six months we spent together. All the moments were amazing. João, Diana and Diogo thanks for all the support and motivation you gave me, but also sorry for the promised cake that never came.

To my girlfriend, Maria, I want to say thank you for all the dedication and love you have always shown. You were tremendously important in this thesis because you were always present to listen to my problems, to motivate me or even trying to help me solve the problems. You are unique and special.

Finally, to my parents, I want to say that they are the best of the best, and I have no words to describe all the things they have done for me. Without them, these five years in the University would never be possible. I Respect, admire and love them.

Abstract

Additive manufacturing (AM) technologies have always been a curious field of research and development which, nowadays, have transformed the production lines of many industries. Hence, the adoption of new production's scenarios led to the necessity of a full automation and control of all the processes through the combination of fields such as robotics and computer programming. Therefore, this thesis describes a simple solution for the implementation of a simulation system for additive tasks experiments in a robot working station, which are controlled through a system control application (SCA). As demonstrated, the emulation of the AM tasks was executed by creating a robot working station in *RoboDK*, which is responsible to generate all the additive tasks, automatically, by interpreting *Gcode* generated in *Slic3r*. Posteriorly, all the system control application (SCA) was fully developed in *Python*, whose final result was a graphical user interface (GUI) that is able to control, by using simple commands, the AM tasks generated in *RoboDK*. As an extra feature, *Slic3r* was embedded in the SCA to enable the generation of *GCode* automatically, without being necessary using the *Slic3r's* user interface. To sum up, this thesis adds new value for this researching field, because it demonstrates how it is possible to simulate and control additive manufacturing tasks into a robot working station, by using conventional working tools of this research area.

Keywords Additive manufacturing simulation, RoboDK, Slic3r, Python, Hyroman, Robotics.

Resumo

As tecnologias de produção aditiva sempre foram uma área de pesquisa e desenvolvimento entusiasmante e que, hoje em dia, têm revolucionado as linhas de produção de muitas indústrias. Consequentemente, as adoções de novos cenários de produção induziram a necessidade de um controlo e automação de todos os processos, através da combinação de grandes áreas como, por exemplo, a robótica e a programação. Assim sendo, esta tese descreve uma solução simples para a realização de um sistema de simulação de tarefas de produção aditiva, numa estação de trabalho robotizada que é controlada através de um sistema de controlo. Como demonstrado, a simulação das tarefas aditivas foi realizada através da criação de uma estação de trabalho robotizada em *RoboDK*, que é responsável por gerar todas as tarefas aditivas, automaticamente, através da interpretação de *Gcode* gerado em *Slic3r*. Posteriormente, o sistema de controlo foi totalmente implementado em *Python*, cujo resultado final foi uma interface gráfica capaz de controlar, através de simples comandos, todas as tarefas aditivas geradas em RoboDK. Como funcionalidade extra, o programa Slic3r foi embebido no sistema de controlo para permitir a geração de Gcode automaticamente, sem ser necessário recorrer à interface de utilização do mesmo programa. Em suma, esta tese constitui uma mais-valia, por demonstrar como é possível simular e controlar, virtualmente, tarefas de produção aditiva robotizadas, através da utilização de ferramentas convencionais desta área de investigação.

Palavras-chave: Simulação de processos aditivos, RoboDK, Python, Slic3r, Hyroman, Robótica.

Contents

LIST OF FIGURES

LIST OF TABLES

SYMBOLOGY AND ACRONYMS

Acronyms

ASTM – American Society for Testing and Materials

HYROMAN – Hybrid Robotics Manufacturing

AM – Additive Manufacturing

SCA – System Control Application

TCP/IP – Transport Control Protocol/Internet Protocol

GUI- Graphical User Interface

OS – Operative System

1. INTRODUCTION

Engineering has always been described as a practical faculty, which is specialized in the resolution of real problems, through an effective, creative and innovative style. It is due to engineering that the world lived 3 industrial revolutions and lives now the fourth, which is strongly based on concepts such as, automation, robotics and additive manufacturing, which are, at the same time, keywords of this thesis.

The industrial robotics was always faced, by society, as a two-sided coin. In one side, some people shouted approval about their implementation, because of the fact that the robots were a precious help in the execution of the heavy works, but also, because the products became cheaper since the robotization provoked a decrease of the production's costs. However, on the other hand, some parts of the society, mainly composed of operators, came out against the industrial robots, because their position in the factory was strongly threatened. This threat was the responsible for several riots, which were planned and executed by operators, such as, the sabotage of woollen looms, in some of the biggest companies in England.

Besides of being controversial, the industrial robotics has never stopped growing. Nowadays it is referred like a field in permanent evolution, highly creative and ambitious, and that always looking forward to new challenges and horizons. A good example, of this characteristics, is the theme of this thesis, which is part of a major initiative called HYROMAN.

The HYROMAN initiative (consortium) was born due to the necessity of improving all the steps of part's production, by grouping all the production tasks into the same working cell. This is done by using additive manufacturing technologies, whose objectives of implementation, is the decreasing of energy and prime-materials costs, as well as the environmental impacts of the whole process. Therefore, the theme of this thesis, which is the ***"Implementation of a simulation system for additive manufacturing tasks"***, tries to find out the answers for the objectives previously mentioned, by developing a system control application which is able to control and execute all the additive

manufacturing tasks. This system will be developed by a wide range of software which gives the necessary tools to solve this problem.

To sum up, during this thesis, all the steps that were taken in order to solve the problem, will be explained, in detail, through a detailed narrative.

1.1. Motivation

Nowadays, the industry is permanently searching for new ways to subsist, since it is necessary to develop new solutions and formulas for classical challenges. This industry's paradigm happens in virtue of previous methods of production, which are based on the usage of a machine for each operation, has started to become outdated due to, for example, economic and environmental reasons.

The adoption of new solutions can be achieved by using non-conventional working scenarios filled with machines, which are capable of performing a wide range of operations. The main goal of applying this concept is to decrease the production costs, through the increase of the system's efficiency, by saving resources such as prime-materials or energy. However, in spite of this scenario being based on economization, it is important to keep in mind that quality must never be affected.

One initiative that can be used as an example of what was previously described is the HYROMAN consortium which led to the development of this thesis. The HYROMAN is a European initiative which aim is creating a robot working station able to combine additive, subtractive and transformative manufacturing in order to minimize the production's time, cost and waste. This is intended to be achieved by creating a system control application capable of performing all the tasks previously mentioned. However, due to time limitations, this thesis only focus on the development of a system control application to perform AM tasks.

The reason why this subject was chosen for this thesis, is not only because robotics and additive manufacturing are two areas constantly growing, but also due to the personal belief that a thesis must be a sample of challenges that an engineer will face during his career, since most of the times, the solution to a problem will not reflect the standard solutions presented during classes.

As it was mentioned, the HYROMAN initiative is a combination of three manufacturing processes that are expected to produce a significant impact on industrial variables such as (Pires, 2017) and (Pires & Azar, 2018):

- **Production time:** It is estimated a decreasing, at least, of 20% of this variable since the working's operations are optimized, and fully integrated into only one cell.
- **Production costs:** It is estimated a decreasing, at least, of 25% on this variable since all the production processes are integrated into the same working cell.
- **Resources costs:** It is estimated a decreasing, at least, of 50% on this variable, due to the technologies and architecture used in this process, in which it is important to highlight the usage of an additive manufacturing process, as well as the smaller travelling distance between operations.

Any project must start with the creation of its foundations, which in this case is the development of an AM robot's station, which is responsible to simulate the AM tasks, and the system control application responsible to control the AM robot's station. Therefore, the concepts previously mentioned were converted into reality, by mixing a wide range of software, such as, a software capable of simulate all the work performed in the working cell, or even, a high level programming language to develop all the system control application, which is responsible for controlling all the working cell's operation. Furthermore, more details will be discussed.

In addition, it is important to clarify that by performing these tasks, this thesis is considered a huge contribution to the HYROMAN initiative since a big part of all the system control will be fully developed and tested.

1.2. Objectives

The objectives of this thesis can be briefly described in the next topics:

- Designing and implementing a virtual working cell in *RoboDK*.
- Generating the AM tasks with *Slic3r* and *RoboDK*.
- Development of a system control application in a high-level computer language (*Python*).

1.3. Chapter's organization

To finish, a description of the thesis's structure and content will be presented next. This thesis will be divided into 5 chapters, whose content is briefly presented below:

- **Chapter 1** – In this chapter, it is revealed the thesis's motivation, as well as a reflection regarding the state of art.
- **Chapter 2** – In this chapter, all the software is described, in order to understand the motivation for its use.
- **Chapter 3** – In this chapter, the virtual additive manufacturing process, which was made in a robot simulation software, will be fully described in order to be understood how the environment was simulated.
- **Chapter 4** - This chapter will describe how it was used a High-level computer programming language was used to develop the system control application, and also to explain which tasks can be performed with this system.
- **Chapter 5-** In this chapter, a brief reflection about this work's future will be done to understand what can be improved.

1.4. State of art

Additive manufacturing, AM, is the official industry standard term, accordingly to the American Society for Testing and Materials, ASTM, to describe all the technologies used to build 3D objects, by the simple process of adding layer upon layer of material until reaching the desired shape, (ASTM F42, 2018).

AM is a field of investigation that has grown a lot during the last years, especially because of all the booming related with the 3D Printing, but also because of the industrial interest on the subject. However, it is important to clarify that AM is not only the 3D printing made by specialized printers, which are commonly used to build parts made with polymers, such as acrylonitrile butadiene styrene (ABS) or polylactic acid (PLA). In order to support the last affirmation, in 2010, the ASTM committee, (ASTM F42, 2017) published a set of standards which describe all the 7 types of AM technologies, which are described in the following image.

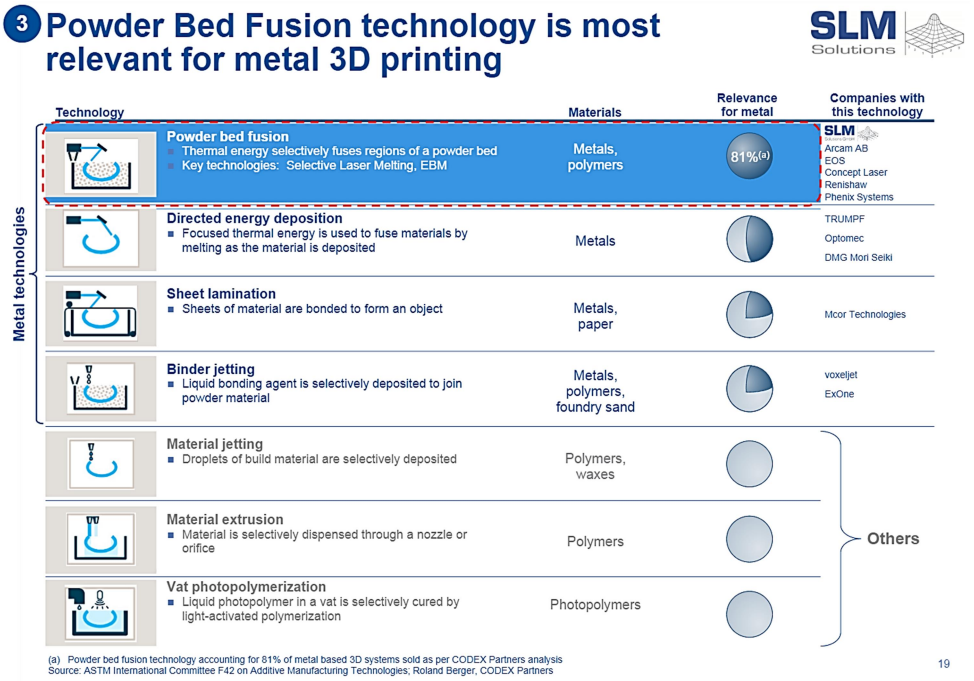


Figure 1.1 - 7 types of AM technologies according to ASTM. Image from ("SLM Solutions Group-Company presentation," 2014)

As in the conventional technologies, each AM technology uses specifics materials, since the genetics of the process does not allow a full material coverage. Accordingly, with the study conducted by (Bourell et al., 2009), the materials used in AM technologies can be grouped into two categories, each is the homogenous and heterogeneous materials, respectively. In addition, some studies about the practical application of these materials have been done, such as the ones in the following table.

Homogenous materials	**Important studies about applications which use AM technologies**
• **Polymers, such as epoxies and thermoplastics.**	(Odom et al., 2017)
• **Natural materials, such as living tissues, paper/adhesive, starch**	(Melchels et al., 2012) ,
• **Metals such as pre-alloying**	(Tan et al., 2017)
• **Ceramics such as glasses, cement**	(Guo & Leu, 2013)
Heterogeneous materials	**Important studies about applications which use AM technologies**
• **Polymeric matrix**	(Tekinalp et al., 2014)
• **Metallic matrix**	(Murr et al., 2012)
• **Ceramic matrix**	(Eckel et al., 2016)

Table 1.1 - Important AM projects in each type of material

Accordingly with (Bourell et al., 2009) and (Ålgårdh et al., 2017), in the future, it will be possible to see AM technologies in industries such as aerospace, military, automotive and motorsport, electronics, biomedical, jewellery, collectables, dentistry, food, education and toys. However, it is important to understand why industries will start changing the production's technologies mainly because of factors, such as (Berman, 2012):

- 95% - 98% of the waste material can be recycled in 3-D printing.
- The ability to economically build custom products in limited production runs.
- The ability to share designs and outsource manufacturing.
- The speed and ease of designing and modifying products.
- No need for costly tools, moods, or punches
- No scrap, milling, or sanding requirements
- Automated manufacturing

Another important study, which supports the idea that AM technologies are the future, is the one conducted by Espacenet, (European Patent Office, 2017). Accordingly with this study, the number of patented projects regarding AM are facing an exponential growth since a few years back, a behaviour that is expected to keep its tendency justified by the increase of investment from big companies and countries, that are inserted in the program Industry 4.0.

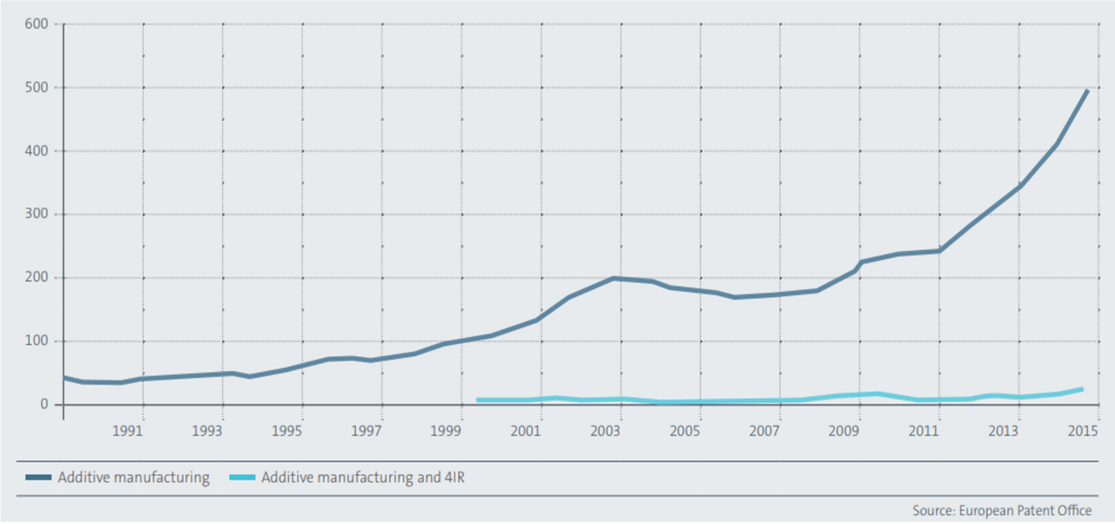


Figure 1.2 - Graphic showing the exponential growth in AM patents. (European Patent Office, 2017)

AM allows overcoming many boundaries carried out by conventional manufacturing technologies such as the production's limitations of components with

complex geometries/shape, and the excessive waste of material due to excessive wall thickness or the type of the technological process. The study conducted by (Ålgårdh et al., 2017) shows how the components with the specifications previously referred can be optimized by using an AM process.

Figure 1.3 - Design for AM resulted in subpart elimination and weight reduction. (Ålgårdh et al., 2017)

However, it is important to notice that in an industrial environment, all of these AM technologies are only possible due to the integration of another area, which is the industrial robotics.

Industrial robot manipulators have been allied with AM technologies due to the fact that they are machines with a huge potential on this field, since they have natural characteristics that make them enabled to perform AM tasks such as the ability to perform repetitive tasks, a high reliability and performance, easy to program and control, and, the ability to print components with a significant size, something that is not possible by using a common 3D printer.

This recent study conducted by (Danielsen Evjemo et al., 2018) shows some projects that would be impossible to perform in a traditional 3D printer, due to physical limitations, since they only have 3 degrees of freedom, DoF, which, consequently, only allows translations along X, Y or Z. Otherwise, a robot manipulator has at least 6DoF, which, in addition, allows the execution of 3 rotational movements. This 3 extra DoF make the difference in a 3D printing because it allows keeping the extruder nozzle correctly oriented during the movements.

AM technologies are being adopted by many companies, and one good example of that is the ADDere System, developed Midwest Engineered Systems Inc., MWES, that has been using a laser additive manufacturer to create complex metal parts, which are impossible to manufacturing via conventional technologies. Accordingly to

company's white paper, (MWES: ADDere System, 2018), it is used a laser light composed by photons, that intercepts a metal wire that melts it, instantly, in a melting pool. An extraordinary point, about this operation, is that it can be executed at room temperature, instead of other processes such as the Electron Beam, which allows a reduction of costs by not being necessary using a vacuum environment.

Figure 1.4 - a) Robot from MWES performing AM tasks. b) Propeller made with AM technologies. Source: (Anandan, 2017)

An important statement was made by the president/founder of MWES, Scott Woida, in the robotic industries association' journal, in an article about the robotic additive Manufacturing, were Woida said that "we are getting properties similar to casting, closer to forged". (Anandan, 2017), which is a great achievement because this proves that AM technologies can perform as good as the conventional ones.

Some important projects about this subject have been developed in the last years, such as:

- *3D Concrete Printing*: A project that uses additive manufacturing processes to produce full-scale constructions and architectural components, mainly walls, bridges, houses, etc., always large-scale components preferentially made of cement. This work has been developed by the Eindhoven University of Technology. (Bos et al., 2016)

Figure 1.5 - AM process using cement. (Bos et al., 2016)

- ***MX3D Bridge (2015 -2018):*** A project which aim is proving that is possible to build strong, complex and gracious structures in steel, by a combination of AM technologies and manipulator robots.

Tim Geurtjens, chief technology officer MX3D, describes the project in this way: *"What distinguishes our technology from traditional 3D printing methods is that we work according to the 'Printing Outside the box' principle. By printing with 6-axis industrial robots, we are no longer limited to a square box in which everything happens. Printing a functional, life-size bridge is, of course, the ideal way to demonstrate the endless possibilities of this technique"*. (Mings, 2017)

Figure 1.6 – MX3D's bridge made from AM process. Source: https://www.engineering.com/3DPrinting/3DPrintingArticles/ArticleID/17038/Additive-Construction-From-the-3D-Printed-House-to-the-3D-Printed-High-Rise.aspx

To sum up, the AM technologies are becoming the future and cannot be ignored, because the world needs to search for new ways of production, due to all the problems it has faced, such as the climate changes and lack of resources. It is impossible to ensure that by implementing these technologies the problem will be entirely solved, but it is undeniable that it will help to reach the solution.

2. SOFTWARE

As previously mentioned, the aim of this thesis is to simulate an AM process fully controlled by a system control application, SCA. So, in order to achieve those objectives, it was used specific software for both parts. In addition, it is important referring that the software used was preferentially open-source/soft-software to be not be restricted to any brand in order to build up a more generic solution.

In the first part, the AM simulation was developed by using *RoboDK* and *Slic3r*, while the last one was entirely developed in *Python* to create the SCA

An introduction of all these programs and why they were used will be given, in detail, during the next topics.

2.1. RoboDK

RoboDK is an industrial robot software that was developed by the PhD graduate Albert Nubiola, which has the possibility to do **offline/online robot programming,** and also **robot simulation** (Nubiola, 2015).

The reasons, why this software was used, are summarized in the next topics:

1. It was necessary to perform a virtual simulation of the process, and that is only possible by using software which has the ability to do offline robot programming.
2. By using *RoboDK* is possible to use more than 200 robots, from a wide range of brands. This is a fundamental point due to the fact that the AM simulation should be flexible and adaptable, something that is not possible to reach by imposing the robot's brand.
3. *RoboDK* is prepared to perform AM projects by interpreting *GCode*, which will be used to generate the components.
4. *RoboDK* is prepared to be controlled by a SCA, once it is fully embedded in programming languages, such as *Python*, *C#* or *MatLab*.

5. *RoboDK* is a new robot programming, which was never tested by the people involved in this thesis. Therefore, this reason also had an impact on the robot software's decision, since it was necessary to test it to make conclusions about its performance and reliability.

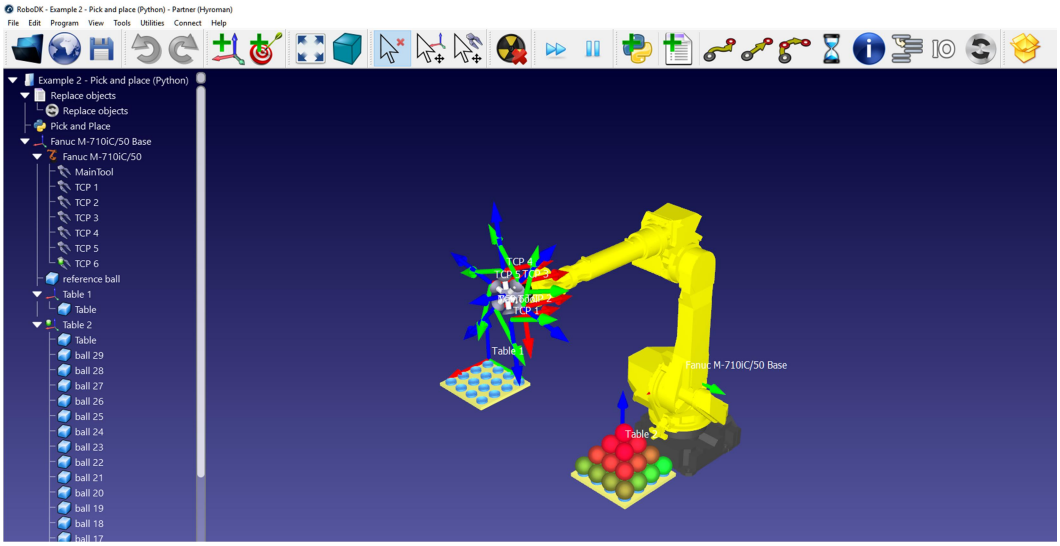


Figure 2.1 - RoboDK user interface

2.2. Slic3r

Slic3r is an open source software launched in 2011, within the *RepRap community*, a community which motivation is making self-replicating machines freely available for the benefits of everyone. Following the rules of this community, Slic3r is used to convert 3D models into printing instructions by generating *GCode,* which is a programming language used to create numeric instructions that make a machine moving along x, y, and z.

Slic3R was used, in this thesis, because, it is a recognized open-source software, which uses essential and accurate parameters to accomplish a great additive manufacturing process. It uses as input *CAD files*, which are developed in the most known *CAD software – Autodesk Inventor*, generating *GCode* as output, which is very important in the robot path generation. The last but not the least reason, it is the fact that *Slic3r* can be fully embedded into the SCA.

It is also important to refer, that even being a software used for 3d printer's projects, this is not a limitation on the AM simulation since the robot will simulate a fused

deposition material process, FDM, which is performed by using 3DoF, as in a 3D Printer. This was done with the objective of simplifying the development.

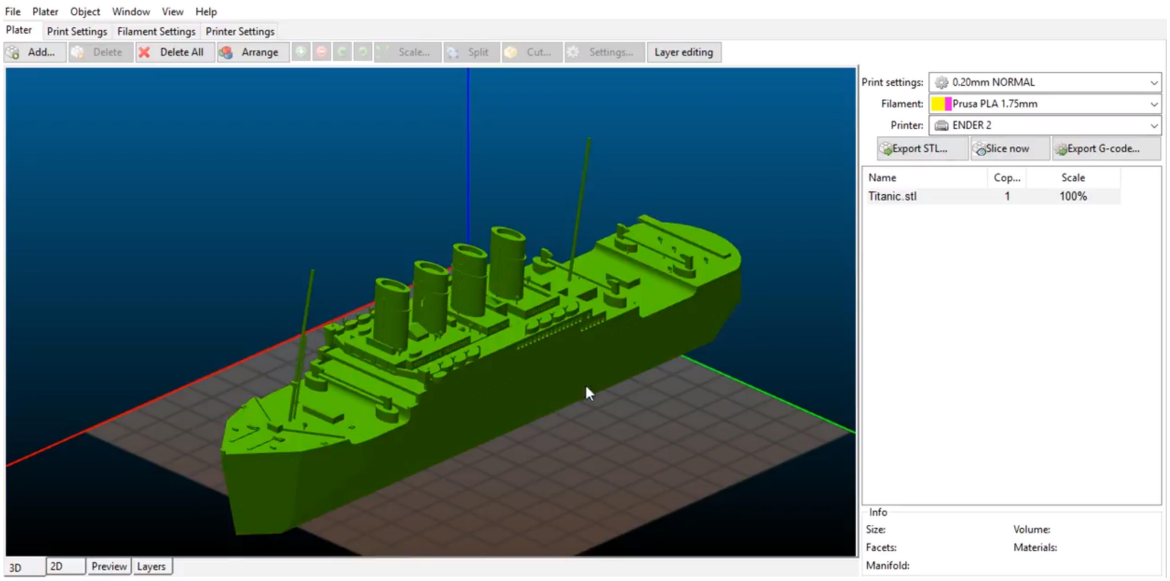


Figure 2.2 - Slic3r's User Interface

2.3. Python

Python is an **interpreted high-level programming language**, created by Guido van Rossum in 1991, and designed to work in a wide range of domains **(General purpose programming language)**. An interpreted language means that python execute instructions directly and freely, without previously compiling a program into machine-language instructions, while the high-level programming concept is related to the independency *Python* has from the computer details.

Python is known for its versatility, for being a fast development environment, an open-source software, and also a "clean" and "clear" programming language, as said in PEP 20, The Zen of Python, *"Beautiful is better than ugly, explicit is better than implicit, simple is better than complex".* (Peters, 2004).

In this thesis it was used *Python 3.6*, whose reasons will be presented next:

1. It is the programming language recommend by *RoboDK* on the basis of this software were written in *Python. RoboDK* has a specific library that allows controlling all the robot station operations, which is called *Robolink*. This library is used for offline/online programming and simulation.

2. Accordingly with Stack Overflow, Python is one of the most used programming languages in the world, and it is also the one with the higher growth rate. (StackOverFlow, 2018)
3. Python has a wide range of libraries such as *Asyncio, threading, subprocess and Kivy,* which allow the creation of a complete and complex SCA.

2.4. Pycharm

PyCharm is an integrated development environment (IDE), used in computer programming, optimized to work with *Python*. This software has been developed by *JetBrains* and provides a lot of advantages like (Pycharm, 2018)

- **Smart Code Navigation** - which gives the possibility to easily jump to any class, file, symbol, etc.
- **Intelligent Code Editor** - something very important due to the ability to detect spelling errors, indentation errors and wrong use of variables.
- **Debugging and Testing** – this feature simplifies the working flow once it is easier to manipulate and write code when compared with the traditional Python IDLE.
- **Virtual environments and one-click libraries** – this feature simplify the process of creating a virtual environment (venv) for each project, and the installation of libraries is much simpler because it is only necessary an online search to automatically install them, without using wheels in the shell.

Pycharm was used in this thesis because it has features and tools to support the SCA, in order to streamline this process, and, also due to the fact that it was built to specifically work with Python.

In the next chapter, it will be presented everything related to the AM simulation.

3. ADDITIVE MANUFACTURING SIMULATION

One of the goals of this thesis is the creation of an additive manufacturing simulation. This chapter will reflect all the work that was done, in order to achieve the previous goal.

The chapter will be divided into two parts, which will be focused on the development of the robot off-line program, as well as in the generation of *Gcode*, by using *RoboDK* and *Slic3r*, respectively.

3.1. Robot offline programming

The robot offline program is an essential part of this thesis since it allows the creation of a virtual simulation, which is responsible to emulate the real working cell environment, as well as all the AM tasks. Therefore, the advantages of using this way of programming are tremendous, since (Nubiola, 2015), (Mountaqin, 2015):

- There is not any breaking in the production flow.
- It is safer because there is not any risk of damage to the working cell/human operator since all the operations are simulated virtually.
- All the simulated processes are replicated in the real working cell because the virtual cell must behave exactly as the real one, otherwise, the model would be invalid.

After this contextualization, the next sections will present all the working cell architecture, and also all the robot programs that were generated to simulate the AM process.

3.1.1. Working cell architecture

Before any consideration about this topic, it is important to refer that there is not any previous design of the working cell. Due to that, the following working cell was

designed carefully since, in the future, it has to be possible of being replicated in the real conditions.

3.1.2. Working cell description

The working cell needs to be designed considering the following aspects:

1. The robot must move freely, without any restriction or obstacle, otherwise, it will stop and fail the operation.
2. The robot must have a tool, in order to simulate the deposition of the melting material.
3. All the parts generated must be built in an appropriated place such as a bed plate.

Therefore, the working cell was designed considering the mentioned aspects, whose final solution consists of a table which will support a robot equipped with an extruder and a bed plate. The robot used was an ***ABB IRB140***, which is available in the *RoboDK* library, as well as the extruder and the table used. In addition, the plate was built in the *Autodesk Inventor* and has the following dimensions: 250x250x10 mm.

3.1.3. Working cell considerations

In *RoboDk*, the working station should always follow a hierarchy, in terms of creation.

This hierarchy is important since it is necessary to have a reference frame which allows the objects and tools to be correctly orientated and positioned. Therefore, this is a delicate subject, since it is the only way to ensure that all the paths and operations will keep the same reference frame, during operation, in order to avoid problems such as singularities or wrong tool orientation.

Hence, the solution for these problems is to define a reference frame that never changes its origin, as well as to simply control all the movements and positions of the robot flange, during simulation. Therefore, the robot base was adopted as a reference frame, due to the fact that all the motions and operations will be executed by the robot.

The hierarchy construction rule implies that all the other frames are dependent on the reference frame, as can be seen in the next image.

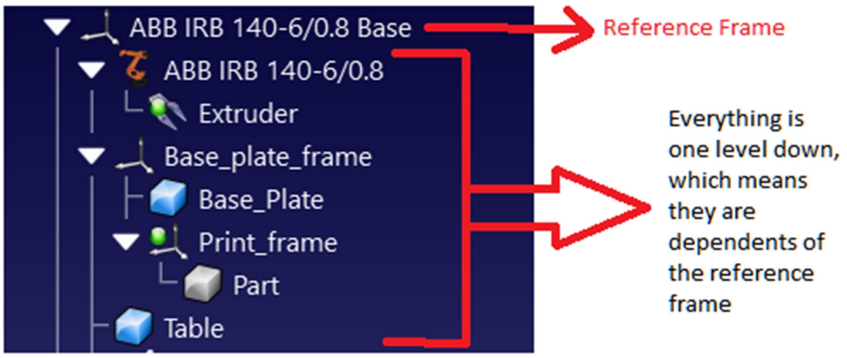


Figure 3.1 - This figure shows how the reference frame and all the dependencies.

3.1.3.1. Working cell design

In terms of vectorial positions $(x, y, z, \theta_1, \theta_2, \theta_3)$, the next table resumes the position of all the objects. The position of the extruder is not mentioned, since it is attached in the robot flange.

Object	Position
Robot base	(0,0,0,0,0,0)
Table	(310,0,0,0,0,0)
Plate	(600,0,0,0,0,0)

Table 3.1 - Robot and objects position in the working cell

The final result can be seen in the next image.

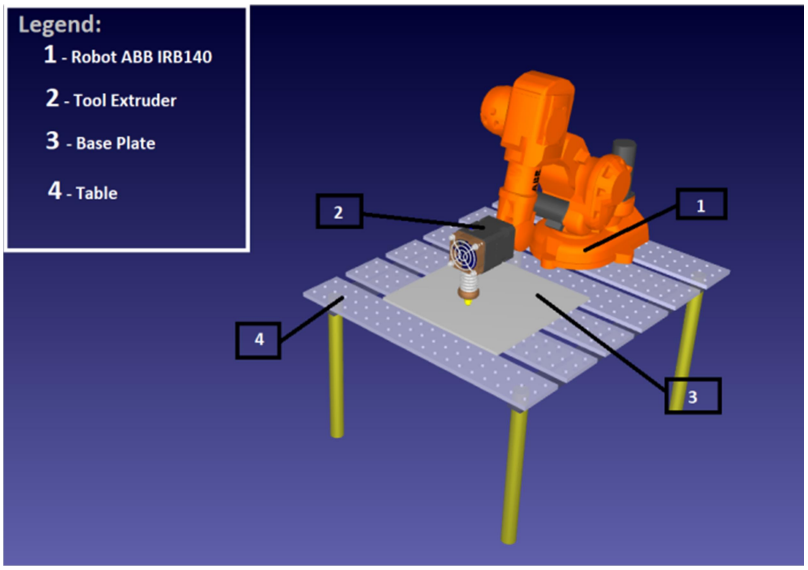


Figure 3.2 - Working cell design

3.1.4. Robot Programs

To perform an AM simulation, it is necessary to generate robot programs in *RoboDK* which are capable of replicating all the AM tasks. This is an important step because otherwise, the robot would not be even able to move.

Therefore, the *RoboDK* programs, which can be created **manually** or **automatically,** are responsible to compile information about the simulation, such as the **robot flange velocities**, **paths**, **types of movements** and to control the **active tool operation**. In this thesis, the programs were created manually and automatically, as it will be explained in the next topics.

3.1.4.1. Manual programming

In RoboDK, it is possible to do manual programming, which is similar to *teaching by showing*, (Lozano-Perez, 1983) by defining targets or operations by hand.

Targets are the space coordinates of each position and orientation occupied by the robot flange which, posteriorly, will be interpolated into a **linear, joint or circular movement**. In addition, operations such as defining the robot flange velocity or activating the material deposition simulation are also defined by hand.

Manual programming was used in this thesis, in order to define the next AM operations:

- Approaching the robot flange, next to the plate's surface, to perform the AM simulation.
- Returning the robot to a home position, after the AM simulation

Both operations are performed by joint movements, due to the fact that they are intended to be fast. However, further details are presented in the next table:

Program	Velocity	Movement type	Precision
Return Home	150 mm/s	Move Joint	Fine
Approach	150 mm/s	Move Joint	Fine

Table 3.2- Specifications of the manual programs used

The next image shows two targets, which were used to return the robot to the home position, after the AM simulation by a joint movement.

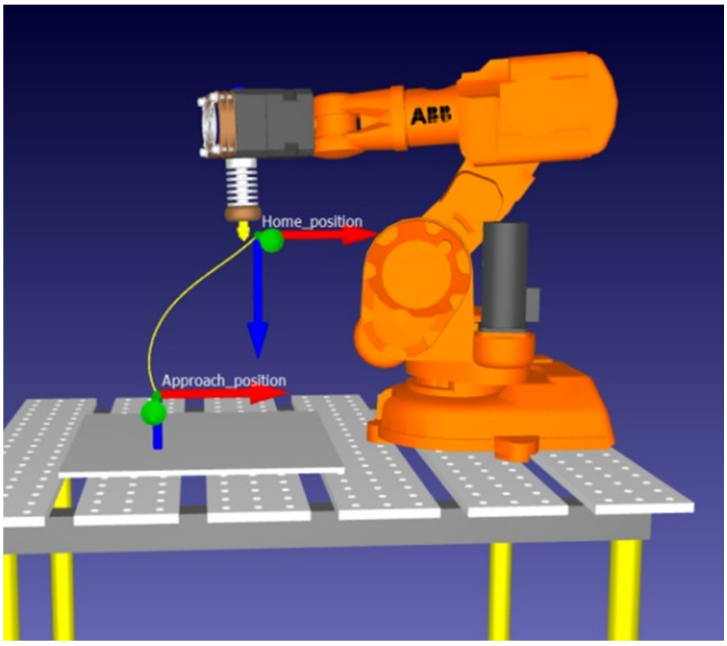


Figure 3.3 - Robot flange approaching the plate surface through a joint movement.

3.1.4.2. Automatic Programming

RoboDK has the ability to generate automatic programs by using the *RoboDK* machine projects.

In this thesis, this type of programming was used to generate the robot program that is responsible to build parts by using the project called "*Milling Project*". This project online requires a *GCode* file to generate the paths, the extruder function, which is responsible to control the extruder flow and the robot flange velocities. In addition, it was used a *Python script,* that is executed every time this robot program is called, in order to simulate the material deposition.

The next image shows the milling project user interface and also, the *tool paths* that will be executed to simulate the AM process.

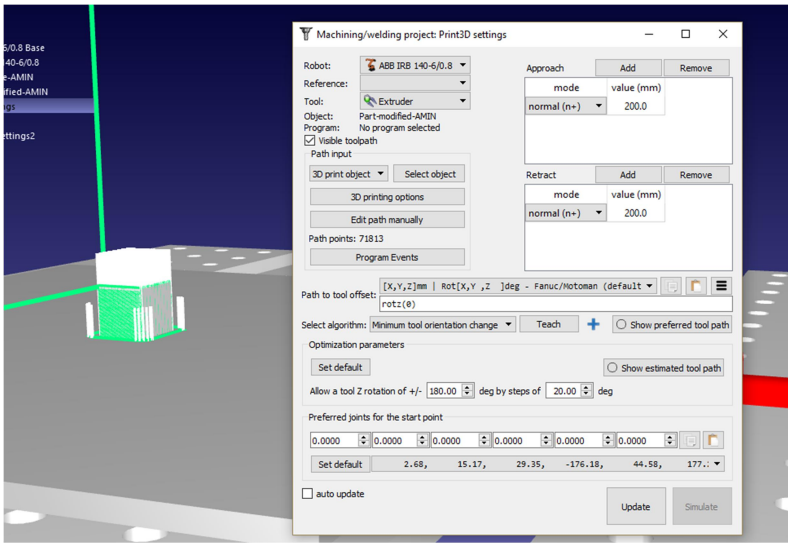


Figure 3.4- Milling project user interface and the tool paths generated.

At the end of this operation, all the programs were grouped in a final program, whose name is AM simulation, which is nothing more than sequential calls of the robot programs, in a predefined order. In summary, the AM simulation contains the following robot programs, whose order of presentation is the same of execution:

Approach – This is a manual robot program approaches the robot next to the plate´s surface to prepare the material deposition.

3D – This is an automatic robot program which contains all the information related to the part's generation.

Filament On/Off – This is a Python's program which simulates the deposition of material.

Return Home – This is a manual robot program responsible for moving the robot to a home position, in the final of the AM simulation.

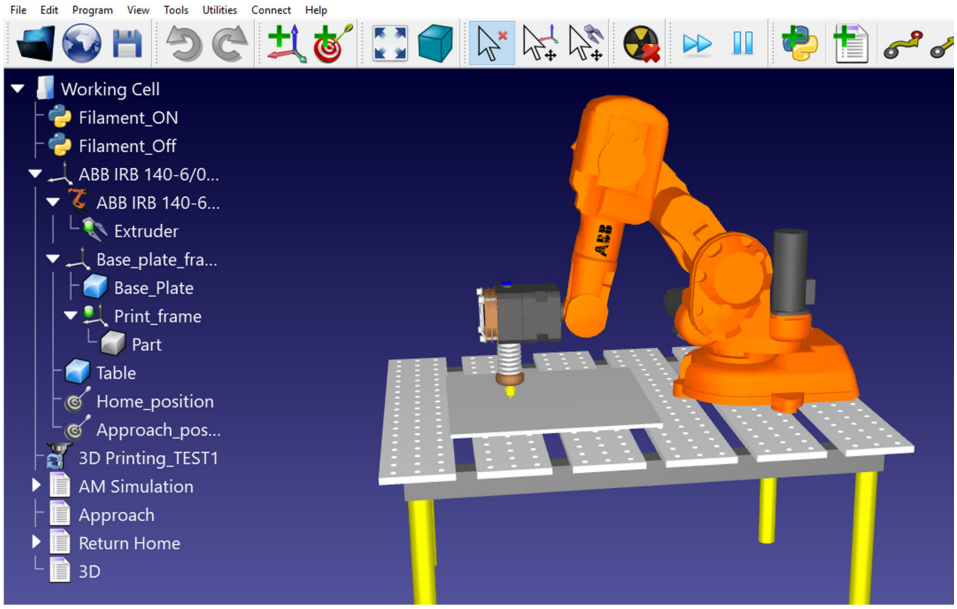


Figure 3.5- The Final look of the working cell, which is ready to perform an AM simulation

3.2. Slic3r

As previously mentioned, *Slic3r* was used in this thesis to generate *GCode* through the configuration of 3 essential settings: **Printer Settings, Print Settings and Filament Settings** (Gary, 2013):

It is important to notice that, since it is not defined any AM technology in the *HYROMAN* initiative, until the date of realization of this thesis, it was decided to use a *Slic3r's* profile from *Lulzbot*, which is a company specialized in FDM solutions. The *Slic3r's* profile used was the "**TAZ PLA Profile – Medium PLA 0,35 mm Nozzle",** (Lolzbot Company, n.d.)

However, it is important to give some basic notions about the key settings of each parameter, in order to understand all the extruder's behaviours during the AM simulation.

3.2.1. Print Settings

The print settings are responsible to control all the parameters related with the layer adjustments such as:

a) **Layer Height:** The layer height is responsible to establish the thickness of each layer.

b) **Perimeters:** The perimeters represent the number of threads that will form a wall in the part, which is usually identified as *a **vertical shell.*** They are easy to be identified during the AM simulation since they are the first thing to be printed in each layer. This parameter is an important feature due to its influence on the wall's thickness, where a higher number of perimeters will be associated with a thicker wall.

c) **Infill –** This parameter has the responsibility to establish the infill's pattern and percentage. The infill's pattern is the geometric configuration used to fill the gap, that is limited by the perimeters, while the infill's percentage determines the density of the part, which varies from 0% to 100%. It is important to notice that the bottom and the top of the part will always be fully filled.

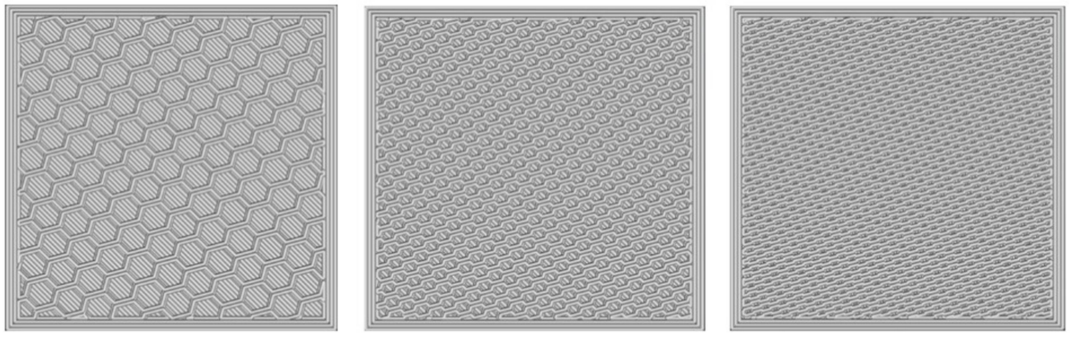


Figure 3.6 - Part sliced with the HoneyComb pattern with 3 different infill's percentages, such as: a) Density: 25%; b) Density: 50%; c) Density: 75%

d) **Skirt**- This parameter is responsible to ensure that the nozzle is cleaned and ready to extrude material, by doing a small printing near the first perimeter.

e) **Support Material** – This feature is important for parts that have some overhanging details where it is necessary to have a structure capable of sustaining it, otherwise there is a risk of deformation or break during printing.

f) **Speed** – This parameter is extremely important since it is responsible to adjust all the speed executions of the following printing operations: Perimeters, infill and support material. However, it is also responsible to define the Travel Speed, a non-printing operation.

3.2.2. Filament Settings

The filament settings are responsible to control all the parameters related with the material used in the AM simulation, such as:

- **Filament diameter**
- **Temperature:** This parameter corresponds to the extruder and bed temperature, and it is possible to select a wide range of temperatures for each feature, accordingly to some properties such as the flow viscosity and melting point.

3.2.3. Printer settings

The printer settings are responsible to control all the parameters related with the printing environment, such as:

- **Nozzle diameter**
- **Bed shape and size**

All the key parameters of the print settings are easily identified during the AM simulation, as it is visible in APPENDIX A.

Even out of this thesis's subject, it is important to remember that by changing the key parameters' values, the mechanical properties/characteristic, such as, for example, tensile strength, surface finish, ductility, porosity, etc., will be changed, like it is suggested by many articles, such as (Singh Bual & Kumar, 2014), (Hong et al., 2016), (Ferrandiz et al., 2016) and (Sukindar et al., 2017).

4. SYSTEM CONTROL APPLICATION

In this thesis, the system control application, SCA, is a mechanism that enables the interaction between a user and the virtual/real robot station.

The importance of a SCA is related to the fact that it allows a user to control all the operations through a "friendly" application, App. The App is described as "friendly" due to the fact that is controlled by a user interface that is intended to be intuitive and logical. These characteristics are achieved by creating an App that has a simple design and clear commands, in order to allow a universal usage, independently of the user's experience.

As previously mentioned, the SCA was entirely developed in Python, in order to understand all the SCA functionalities. Hence, all the libraries will be contextualized and explained accordingly to its objective of usage as well as the way they were combined in order to build a complete App. To finish this brief introduction, the four fundamental parts of the SCA will be soon described:

- Asynchronous *TCP/IP* - Server/client.
- Robot Station Control.
- Generation of *Gcode*.
- Graphical user interface, *GUI*.

4.1. Asynchronous TCP/IP - Server/client

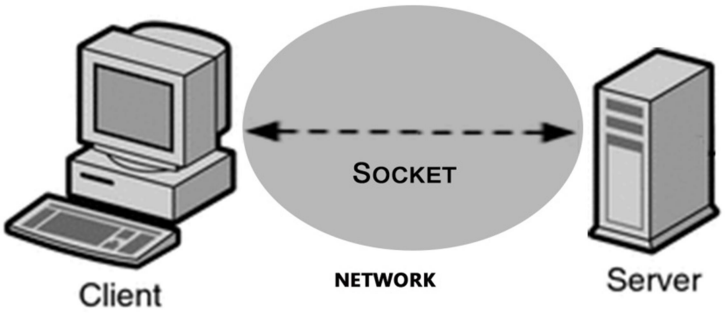
4.1.1. Important considerations

First of all, it is fundamental to understand that the robot and the computer are two devices that do not know how to communicate/share information with each other since they do not have implemented a native protocol capable of being interpreted by both. Due to this reason, it was implemented a *transport control protocol*, *TCP*, combined with the *internet protocol*, *IP*, also known as *TCP/IP*, in order to enable the communication.

The development of the SCA began with the creation of an ***asynchronous server and client***, in order to the required SCA tasks in the AM simulation.

In the computing world, a server and a client are part of a software architecture, whose objective is the inter-communication between each program, whereby the ***clients*** **always sent requests** while the ***server*** **always responds the requests sen**t, as well as, it is responsible to **execute/schedule the client's request**, (Oluwatosin, 2014). Hence, these two programs can be seen like a real-life symbiosis, due to the fact that they need each other to survive and be useful, since if one fails the other will not be able to do anything.

In the SCA, the clients represent all the operations requested by a user, while the server is the program responsible to receive the requests, and then, accessing the AM robot's station to active the operation requested, by the user, through ***sockets,*** that are responsible to establish the connecting endpoints between both sides of connection, (Xue & Zhu, 2009)



.

Figure 4.1 - Representation of a socket

However, it is important to ensure that the system is asynchronous, otherwise, it is impossible to perform multiple tasks simultaneously, such as request position, extruder flow, bed temperature, etc., due to the fact that the system blocks while it is performing an operation, only available at the end of it.

The usage of asynchronous programming enables the system to interleave tasks, by suspending and returning through I/O control. This is called a ***concurrent programming***, where all the tasks can start, run and complete at overlapping time periods, whose biggest advantage is the fact the system is always responsive and independent of the end of each operation to progress (Ghezzi, 1985).

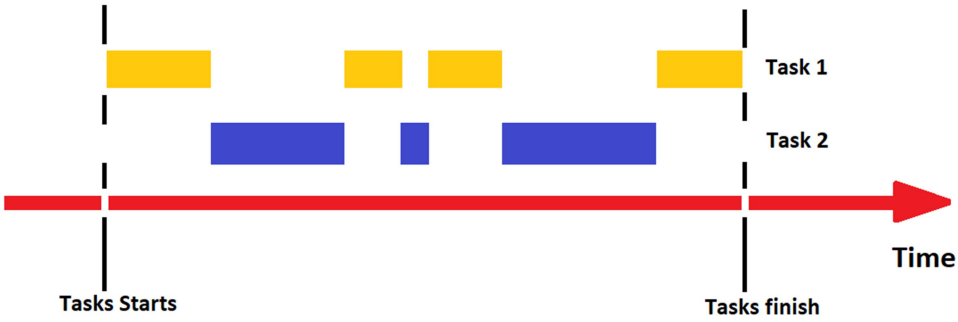


Figure 4.2 - Illustration of how the tasks are interleaved in a concurrent programming.

The implementation of an asynchronous *TCP/IP* server/client was done in *Python*, as it was mentioned before, by using the *Asyncio's library*, (van Rossum, 2012).

Before explaining how the server/client was made it is important to understand the basic concepts of *Asyncio*, which are:

- **Event loop** – The event loop is the brain of the asynchronous program. It is responsible to take care of all the program execution by launching and schedule the operations. If the event loop fails all the operation will instantly fail.
- **Coroutines** – Coroutines can be described as special functions which work like generators, however, which do not iterate as a pure one. Instead of that, a coroutine has the ability to ***await*** for another operation, also known as ***cooperative multitasking***, by suspending its operation, without loss of information as a regular function. This is done by releasing the control of the operation to the event loop which will decide what will do next. All the coroutines need to be wrapped in ***tasks***.
- **Tasks** – They are responsible to schedule and wrap coroutines on the event loop, to perform the desired operations, with a promise that they will always return a result. This is the reason why tasks are identified on Python's PEP 3156 as a subclass of ***Future,*** which is an object that encapsulates a result or an error, since when a coroutine finishes there is always a task's result. An important consideration about tasks is they

need to be scheduled before launching the event loop since they are not thread safe, however, this is only a problem when it is necessary to access objects outside the event loop.

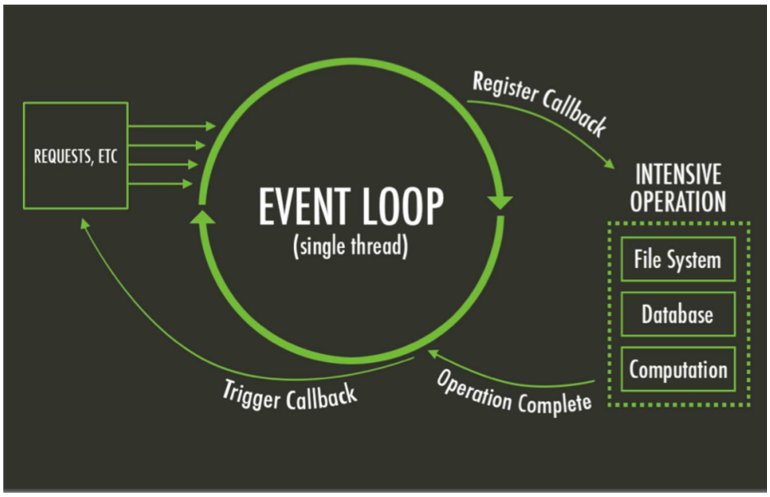


Figure 4.3 - This image shows the working flow of an asynchronous program made with Asyncio. Source: https://bit.ly/2LSLg1W

Asyncio is not a simpler library and it is very hard to understand all the concepts behind this, however, there are some rules that must be followed in order to prevent errors, such as, (Diaz, 2016):

- Coroutines always implement tasks.
- Coroutines ***await*** other coroutines
- Event-loop schedules concurrent tasks
- Tasks must not block
- Awaiting facilitates context switches

After this important introduction about *Asyncio*, it is going to be explained how the server/client was implemented.

4.1.2. Server Implementation

In the SCA, like it was previously mentioned, the *server* is responsible to activate all the functionalities requested by the *clients* in the AM simulation, and also to send back to them the operation result. In addition, the *server* needs to have the ability to connect with multiples clients, because the user can request multiple services during

operation. This behaviour can provoke problems, during execution, such as App breaks or loss of information.

These problems exist due to the fact that the ***synchronous operations*** always block during execution, and as a consequence, they do not allow any new operation until the end of the previous. If this type of operations were adopted in the SCA, the consequences would be an App extremely slow and inefficient and that only allows the execution of one task simultaneously.

As it was mentioned in the consideration's section, the *server's* implementation was done by following the *Asyncio* library, which has specific documentation about the creation of a TCP/IP server, (van Russom, 2012)

4.1.2.1. Asynchronous server creation

The first step in any *Asyncio* program is the creation of the event loop, which in this case should ***run forever()*** due to the fact that the server will always be ready to perform any operation or to receive a new client. The usage of a loop that will run forever means that it will only stop when a ***stop()*** method is invoked, however, it is fundamental to have in mind that after stopping the loop it is impossible to start it again.

After creating the loop it was necessary to create a coroutine which could be performed by a task. The coroutine is responsible ***to return*** a tuple with two objects, which are a **StreamReader**, the object that interprets the message received by the client, and a ***StreamWriter***, the object that sends information to the client, respectively. One important characteristic about this coroutine is the fact that it only executes once a new connection is established since each client's connection requires a new socket, otherwise, the App would block, or even mix or miss the information.

The final step was to ***fire*** the loop to start the server, making it able to receive the clients' requests. The main code of the server, whose parts of creation were explained above, can be seen in the next image. Each code's method is commented to make it

understandable.

```
import asyncio

class Server:

        def__init__(self):
                "This class is used to create the server"
                self.loop = asyncio.get_event_loop()
                self.loop.run_until_complete(self.coro) #runs everytime there is a client
                self.run_forever()

        async def _coro(self):
                "Asyncio Coroutine that defines the TCP/IP protocol to start the server"
                await asyncio.start_server(self.handle_echo, 'localhost, port =8888, loop= self.loop) #ensure
server is created

        async def handle_echo(self, reader, writer):
                "Asyncio coroutine which creates the StreamReader and StreamWriter
                self.reader = reader
                self.writer = writer

        del decisions(self):
                "Method used to decode the receives message and interact with the AM simulation"
                pass

if __name__ = '__main__':
        Server()
```

Figure 4.4 - Server code snippet

4.1.3. Client

A client is a program which makes requests in a server, as already mentioned. In terms of the principle of working, the client is very similar to a server since both are used to communicate. In addition, the differences are mainly concentrated on the client's lifetime, due to the fact the client is supposed to be created to inquire the server about something, and, after the work has finished, leaving the system to preserve computer resources.

4.1.3.1. Asynchronous Client creation

As mentioned, the client is not supposed to live forever as the server, for the reason that it will lead to a ***high consumption of the computer resources***, which affects parameters such as ***connection velocity***, or, even ***tasks and app performance,*** problems

easily avoided if the client is killed at the end of each ***socket.*** This means that the client will ***run until complete*** whereas the server will ***run forever***.

It is important understanding that if the server is asynchronous the client also needs to be, otherwise the code will block until the end of the client's work, due to the lack of ***concurrent tasks***. The client was created with the Asyncio library, which has specific documentation about a TCP/IP client. (van Russom, 2012)

Such as in the server, the first thing to be created was the ***event loop,*** which always needs to be prepared to generate a new client in order to request something to the server, as also to kill it every time the work has finished. Due to this feature, the loop will not ***run forever,*** like in the server, but instead of that, it will ***run until complete***.

After creating the loop, the next step was enabling the client's generation without being necessary to wait until the end of the previous one, whose objective is the elimination of the App's blocking risk, and also to allow more than a request, simultaneously. The solution to these problems was the creation of a task that will ***run coroutine threadsafe.*** By running the coroutine thread safe, the loop will be able to perform the creation of a new client, because it will prepare the loop to start a coroutine, which was not scheduled before firing the loop, and that also has initial running conditions, without breaking the loop.

The source's code of the client, whose parts of creation were explained above, can be seen in the next image. Each code's method is commented to make it understandable.

```
import asyncio

class Client:

        def__init__(self):
                "This class is used to create the server"
                self.loop = asyncio.get_event_loop()
                self.message = None

        async def create_cleinte(self):
                "Asyncio Coroutine that defines the TCP/IP protocol to create a client"
                reader, writer = await asyncio.open_connection('localhost, port =8888, loop= self.loop)
#creates the client as also the StreamReader and the StreamWriter
                self.reader = reader
                self.writer = writer
                await asyncio.ensure_future(self.connection) #waits until the end of the client

        async def connection(self):
                "Asyncio coroutine to send and receive information"
                self.writer,write(self.message) #Writes on server
                server_message = await self.reader.read(100) #Receives the server's message
                await self.writer.drain()

        def new_request(self, message):
                "this method creates a new client
                self.message = message.encode()
                asyncio.run_coroutine_threadsafe(coro = self.create_client(), loop = self.loop)

if __name__ = '__main__':
        client = Client()
        client.new_request(message = "Hello World")
```

Figure 4.5 - Client Code Snippet

To finish this subject, a schema to illustrate how the system control application works will be present in APPENDIX B.

4.2. Robot Station Control

The reason why a program which controls the robot station was created is linked with the fact that the server needs to perform tasks in the robot station, however, the server does not know how to do it, since the source's code only knows how to accept clients and manage requests.

Therefore, the solution to this problem is the implementation of a code, which will be called every time the server receives a request related to the robot station. By doing this, the server is able to perform the all the station's tasks, as also, to translate, and send back to the user, all the information received from the robot station.

This control was entirely made in *Python*, as already referred, by using the *RoboDK* library, (Robodk, n.d.), since all the functionalities are 100% compatible and workable.

The implementation of all the mentioned concepts was done in two main parts, which are, respectively, the ***station recognition*** and the ***station tasks***. Both parts are methods of the ***Subclass*** **ValidateRobodk(Robolink).**

In ***Python***, a ***subclass*** is defined as a child of a superclass, *see Python's PEP 252 and 253*, which in this case is the superclass ***Robolink***. The reason for the creation is the fact that a subclass ***inherits all attributes and behaviour methods from the rooted class***. By doing this, the class ***ValidateRobodk(Robolink)***, releases control to its "parent" every time it is necessary to perform an operation related with the robot station, whose biggest advantage is more efficiency and performance.

4.2.1. Robot station recognition

The station recognition is a class's method which is responsible to detect all the objects in the robot station, such as the robot, the working tool, the working frame and, finally, the final robot program, that has all the movements executed by the robot, during the printing simulation. The recognition of the robot station is very important since the SCA must have information about all the robot station's objects, otherwise, they will be impossible to control from the App, due to lack of object's information in the server. A snippet view of the station recognition can be seen in the next image.

```
from robolink import *

class ValidateRobodk(Robolink):
    """RobotDk Class that makes possible manage robotDK, for this we use the Parent
Class RoboLink"""

    def __init__(self):
        self.Render(True)
        self.ShowRoboDK()
        self.robot = self.Item('', ITEM_TYPE_ROBOT) # This detects the robot
        if not self.robot.Valid():
            raise Exception('No robot selected or available')
```

Figure 4.6- Code snippet showing the robot station Detection

4.2.2. Robot Station Tasks

On the other hand, the station's tasks, which are a set of class's methods, are responsible to interact with the robot station, in order to perform all the user's requests, which are:

- ***Get_position():*** A method responsible to get the real position of the robot flange.
- ***Go_home():*** A method that sends the robot to a predefined home position.
- ***Start_printing():*** A method that starts the robot simulation.
- ***Pause_printing():*** A method that pauses the robot simulation.
- ***Stop_printing():*** A method that stops the robot simulation.
- ***Check_colisions():*** A method that checks if there is any collision.
- ***Calibrate_robot():*** A method that performs an automatic robot calibration.
- ***Station_ready():*** A method which verifies if the station is ready to perform.
- ***Simulation_time():*** A method that calculates the simulation time during the simulation

A snippet view of the code, which has all the methods referred above, can be seen in the next image.

```
from robolink import *

class ValidateRobodk(Robolink):
    """RobotDk Class that makes possible manage robotDK, for this we use the Parent
Class RoboLink"""

    def get_position(self):
        """This is a method that gets the real position of the robot flange"""
        position = Pose_2_TxyzRxyz(self.robot.Pose())
        position_final = [round(position[elem], 2) for elem in range(len(position))]
        return position_final[:3]

    def go_home(self):
        """This is a method that sends robot to home position"""
        Robolink().RunCode('Return Home', True)
        return str("Robot at Home position")
```

Figure 4.7- Code Snippet showing the robot station task

The previous methods represent all the clients' requests which are performed by *RoboDK* and launched by the Server. As it has been mentioned, this class, specialized on the AM system control, is extremely important to enable the system to collect information about all the robot station status, at real time, but as also to launch the required user's operation. The next section will discuss the ability to the SCA generate Gcode.

4.3. Gcode Generation

The SCA must able to automatically generate *Gcode* in order to be possible the creation of an AM simulation on the working cell, easily, however, under some conditions.

Accordingly to *Slic3r's library*, (Gary, 2013), this is a software written in *Perl* and *C++*, that is possible to be embedded in another language, by launching the *Slic3r's execution* on the command prompt, through the introduction of commands lines, which are available in the *Slic3r's documentation*. Therefore, by launching the execution of *Slic3r* in the *Python's shell*, it is possible to generate *Gcode*.

An important consideration about the generated Gcode, is that there is not any difference between the *Gcode* generated from the command prompt when compared with the *Gcode* generated in the *Slic3r's* user interface. If this condition was not verified the command prompt could not be used to perform this operation.

To sum up, all the implementation was done in *Python*, by using the subprocess's library, since it is necessary to execute command lines on the shell.

4.4. G Code Embedding

The automatization of the *Gcode's* generation started with the creation of a class, whose name is ***Slic3r(),*** that initializes, ***__init__()***, with the creation of objects that represent each *slic3rs' command word,* that is used in the *Gcode's* generation.

The initialization of the class started in this way because each slic3r's operation requires a specific command line, which differs a lot from one operation to another. Due to this reason, the command line cannot be written only in one variable, since it is very hard manipulating it.

```
class Slic3r:

    def __init__(self, stl_file, ini_file):
        '''This variables allows us to slice and communicate between processes'''
        self.stl_file = stl_file
        self.gcode = None
        self.gcode_name = None
        self.file_name = None
        self.link = None
        self.txt_file = None
        self.cmd = r'C:\RoboDK\Other\Slic3r\slic3r-console.exe'
        self.command = None
        self.load = r'--load'
        self.output = r'--output'
        self.save = r'--save'
        self.info = r'--info'
        self.ini_settings = ini_file
        self.command_list = []
        self.__write = None
        self.generate_gcode()
        self.file_properties()
```

Figure 4.8- Code snippet of Gcode init method

After defining all the necessaries commands to generate Gcode from the command prompt, it was created a method called ***slic3r_shlex(),*** whose basis came from the native python ***Class Shlex.*** The objective of this method is to write prompt commands automatically, in order to be executed in the shell, without lexical errors. A code snippet from this method can be seen below.

```
def my_shlex(self):
    """Method to make the cmd line in order to properly execute the command"""

    cmd = self.cmd
    for i in range(len(self.command_list)):
        cmd += ' ' + self.command_list[i]
        i += 1
    return cmd
```

Figure 4.9- Code snippet of the *Slic3r*'s *shlex* method

After the command line has been manipulated, it is ready to be executed on the shell. This was done by calling the private method _command_line(), which is responsible to open a *Pipe* by invoking the method Popen(), which method's description is well

documented in the *subprocess's* library, (Python, n.d.). The method created works under the following principle:

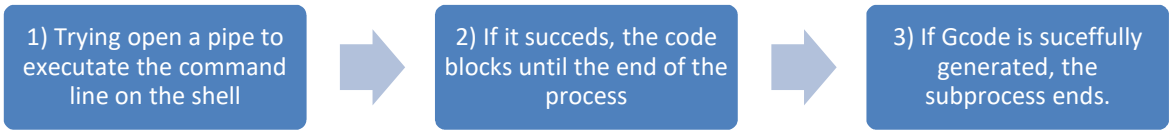


Figure 4.10- ChartFlow about the *Gcode*'s creation

```
def _command_line(self,cmd, std):
    """Method to create the subprocess in order to create the connection with the
command line"""

    try:
        process = Popen(cmd, stdin=PIPE, stdout= std, stderr=PIPE,
universal_newlines=True) #opens the process
    except:
        print('Something happened please contact assistance')
    finally:
        print('Executing command, please wait...')
        process.wait() # blocks until finished
        print('Successfully done! Check slic3rs destiny path to see the GCode ')
        self.command_list.clear()
```

Figure 4.11- Code snippet to open the subprocess and run the command line on the shell

After generating *Gcode*, another *subprocess* is created, in order to write in the slic3r's command prompt a command line that analyses the part sliced to give to the user important specifications about her. These specifications are saved in a notepad file. In addition, the previous file is complemented with information related to all the *Slic3r*'s configurations, which were used to generate the part.

```
def file_properties(self):
    """Method that returns a .TXT file with the file properties"""

    self.file_name = self.gcode_name + '_Properties'
    self.link = r'C:\RoboDK\Other\Slic3r\%s.txt' % self.file_name
    self.txt_file = open(self.link, 'w')
    self.command_list.extend((self.stl_file, self.info))
    self.__write = self.my_shlex()
    print('\nthe self.write result is:', self.__write)
    self._command_line(self.__write, self.txt_file)
    self.txt_file.close()
    self.slic3r_properties()
```

Figure 4.12 - Code snippet to show the subprocess that creates a file with information about the sliced part

4.5. Graphical User Interface- GUI

A **graphical user interface**, also known as **GUI**, allows any user to figure out, by themselves, which operations can be performed in the system since they are usually very intuitive and restricted to simple actions, such as, for example, point and click. A great example of what is a GUI is the Android system.

In the system control application, the GUI was developed to control the robot station, by using simple interactions with the App, such as clicking buttons or dragging bars, but, as also to generate *GCode* intuitively. The building and designing of the GUI were done in *Python*, by using the *Kivy library*.

The reasons why it was used *Kivy* is because this library allows the creation of beautiful interfaces, much easier and faster when compared with *TkInter*, the native *Python's* GUI developer allied with flexibility, since it is compatible with a wide sort of operating systems, OS, such as *IOS*, *Android* and *Windows*.

Before giving more details about the GUI development and functionalities, it is important to give a small explanation about the most important *Kivy's* concepts, which in some way are similar with the *Asyncio's* library, in order to understand the purpose of some actions, which were taken during programming.

4.5.1. Kivy's Considerations

First of all, it is important to notice that the heart of an application built with *Kivy* is the main loop, like in the *Asyncio Library*.

The main loop is responsible to invoke *callbacks* at every iteration, during the whole application's lifetime, which starts when the app is opened and that ends when it is closed. As in the *Asyncio Event Loop*, the main loop is not prepared to schedule any callback susceptible to block during operation, otherwise, the loop will break and the interface will freeze, since the essential operations, which keep the loop alive, cannot be performed. (Kivy, 2012)

One of the most important classes of *Kivy* is the ***event dispatcher***, which is responsible to register and dispatch all the user input events. In addition, a *user input even*t occurs every time the user interacts with any user interface's element, which is also known as *widgets*.

This is such an important point since the *event dispatcher* generates *events* that will be handled by the main loop, in order to specify which callback must run to perform the required action. The following image shows how events are handled in the Kivy framework.

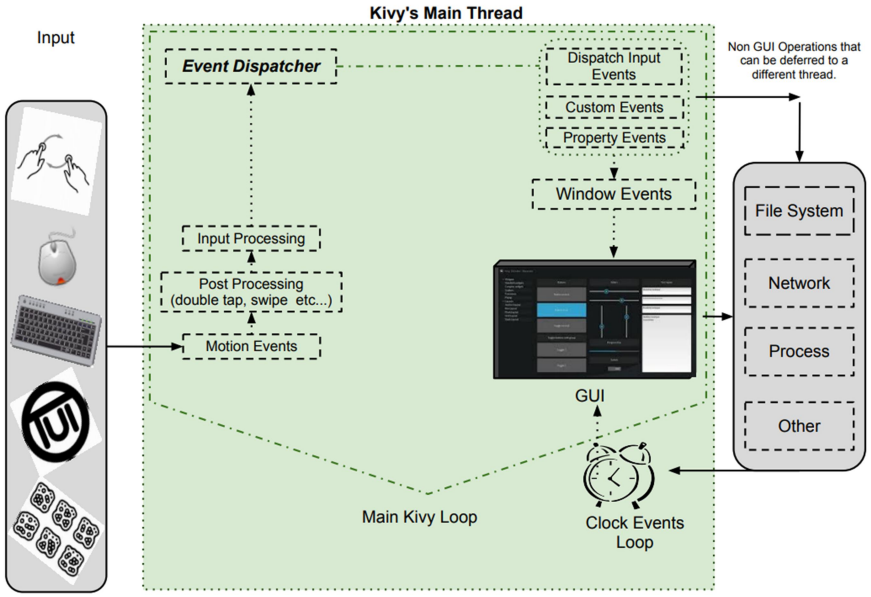


Figure 4.13– Event handling in the Kivy Framework. (Kivy, 2012)

The last important consideration about *Kivy* is related with the programming language that is used to design the GUI since *Kivy* has its own "*design language*" which is called ***Kivy language***, also known as ***KV Language***.

In terms of coding style, the KV Language uses the *Cascade Style Sheet, CSS*, which is used for programming in *HTML*. In addition, the files generated by this programming language are known as *Kivy files*, which extension is ".kv".

The *Kv language* is not obligatory to be used since the user interface can be created in *Python* by using special syntax, however, for a complex App it is highly recommended due to the following advantages:

- Widgets are created in a declarative way and follow a tree construction, which makes them easy to relate and modify.
- Widget properties and Callbacks are easier to bind because there are defined all the widgets behaviours, which lead to an easier interaction between the Event Dispatcher and the User input events.
- There is a separation of the two main parts of the GUI, which are the logic that is implemented in *Python*, and the design that is implemented in the *KV File*.

After this introduction, it is going to be explained how all the subjects, which were mentioned in this chapter were incorporated, and also how the GUI was created. The GUI creation is divided into two parts, where the first one is related to the AM simulation controlling and the last one with the generation of *GCode*.

4.5.2. Graphical User Interface's development

The first steps to develop the GUI were the ***creation of the Kivy's loop***, and of the ***KV file.*** This was done simultaneously because otherwise it is not possible to visualize the building of the GUI.

The *Kivy*'s loop is responsible to load the *Kivy* file automatically, if the *Kivy* file's name is the same as the class that is responsible to generate the *Kivy* loop, in the *Python* program. However, it is important to always keep in mind the fact, that the *Kivy*'s

loop is responsible to update everything related with the App, for example, the widgets status/information or even the generation of the callbacks to create the clients.

```
class InterfaceApp(App):

    def build(self):
        "Method used to build the loop and the User interface"
        return GenerateSlide()

if __name__ == '__main__':
    InterfaceApp().run()
```

Figure 4.14 - Code snippet of kivy class responsible to generate the loop

In terms of design, the GUI was projected by following the considerations mentioned on this chapter's introduction, and due to that, it was only used straightforward commands such as buttons, selectors and bars, which only require one touch to interact with the ***EventDispatcher.***

Kivy is based on iterative operations, which implies that the main loop must not have any blocking operation, otherwise, the app will automatically crash. This point is fundamental to explain the next step of the App's development which is related to the embedding of the server and the client in the Kivy's app.

As it was explained previously, the server and the client are asynchronous programs that require an ***Asyncio Event loop*** to be executed, that must not be suspended in order to keep the server and the client alive. This characteristic represents a serious problem, because it turns Kivy and Asyncio into incompatible libraries, due to the fact that they do not allow the execution of simultaneous loopings.

The reason why this happens is related with the fact that both loops operate in the same thread, hence, one of the loops will be suspended during the execution of the other, leading to a lack of resources that are necessary to keep both sides alive. Therefore, the solution for this problem is launching the ***Asyncio's event loop*** in a ***new thread***, by using the ***multithreading library***, which allows parallel execution without breaking, as mentioned on PEP 371 (Noller & Oudkerk, 2008). In addition, the Kivy's loop will be launched in a lower-level thread, which represents the ***mainthread*** of execution.

However, the threads need to communicate with each other, otherwise, it would be impossible to control the AM simulation since the *mainthread* only triggers the callbacks that will imply a client's generation as well as the respective message, in the upper-level thread.

Regarding this situation, a ***worker*** was created to be the intermediate between each thread-lever. Its working principle is based on a ***callback*** that is triggered everytime the ***EventDispatcher*** creates an ***event***. After the worker has been created it will send a message, which contains the operation to be executed, to the *Asyncio's thread*. In **APPENDIX C** it is possible to see the flow chart that illustrates what was described.

```
class InterfaceApp(App):

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.worker = self.event_loop_worker = EventLoopWorker() #Worker
Creation
        self.ini_file = None
        self.stl_file = None

    def create_request(self, *args):
        self.worker.creates_connection(*args) #Worker orders a new client.
```

Figure 4.15 - Code snippet showing the worker's creation method

However, the opposite working flow must happen, since the GUI must be updated, accordingly with was requested by the user, in the server. This reverse operation is not performed by the worker, but through a method kivy_update_app that uses a special decorator, @mainthread, in order to return all the desired results to the *Kivy* loop as well as to update the widgets.

```
async def _update_app(self, message):
    '''Used to Update all the information related with the App'''

    @mainthread #Decorator to return to the mainthread
    def kivy_update_status(message):

      if self.request == 'position':

App.get_running_app().root.get_screen('main').ids.alert_message.text = str(')
#This line updates a widget information
          position = message.decode()
          position = position.split()
          print('What I will update at my label', position)
```

Figure 4.16 - Code snippet which shows the method that returns information to the mainthread.

Finally, the last step in the GUI development was to enable the app to generate Gcode. This feature was also done by launching another thread, since the Gcode program blocks during operation when it is required to waits() until the generation of Gcode.

```
class Slic3rCodeInvoke(EventDispatcher):
  def __init__(self,stl_file, ini_file,**kwargs):
    super().__init__(**kwargs)
    self.teste = None
    self.stl_file = stl_file
    self.ini_file= ini_file

    self.create_thread()

  def create_thread(self):
    "This method creates a subprocess to run Slic3r"
    try:
      threading.Thread(target=SLIC3R.Slic3r(self.stl_file, self.ini_file),
daemon=True).start()
    except:
      print('Hello World')
    finally:
      print('I canot kill it')
```

Figure 4.17 - Code snippet showing the Slic3r launching in a thread

To finish, a complete view and description of the GUI can be seen in the APPENDIX D. In the next chapter, the conclusions of this thesis will be presented.

4.5.3. App protections

In order to increase the reliability of the App, it was implemented some App's protections, such as:

- It is not possible to repeat the client's request is already active by another client. This is important because it avoids strange behaviours in the working cell during execution.
- App has a method called decisions() which are intended to verify and ensure that the clients are closed after they have done its work, in order to keep the app fast, as well as to save computer resources.
- During the generation of *Gcode*, the user is only allowed to search for the files in the specific folder for this operation and they only show the available files to be used as input. The program also checks if the files are valid or not, showing a message every time that is not possible to generate *Gcode* due to wrong input files.

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

Additive manufacturing technologies have been seeing as one of the most enthusiastic research areas, due to the fact that they are appointed to be the future of the production processes. However, there is still a long way to go until these technologies reach their maturity, since they only started to arouse the curiosity of the industry a few years ago. Therefore, the purpose of this thesis was contributing for the growth of the additive manufacturing field through the realization of a system control application capable of executing/simulating the additive manufacturing tasks, which are performed in a robot manipulator.

As demonstrated, the creation of the additive manufacturing environment was only possible by using software that allows the creation of a robot working station and its main operations, where it is important to highlight the ability to simulate the additive manufacturing tasks. These objectives were fully achieved by using RoboDk and Slic3r's software whose final result was a part completely printed, as can be seen in the next image.

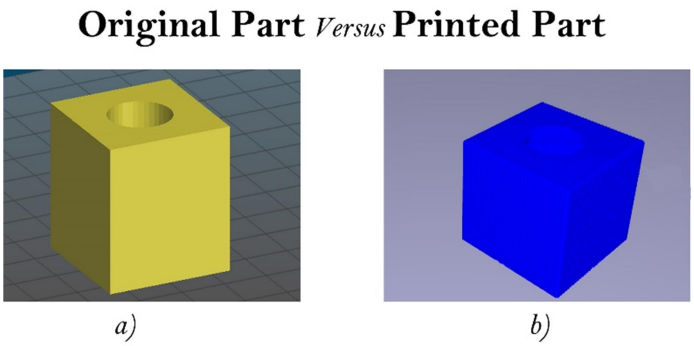


Figure 5.1 – This image shows the part specimen before and after simulation, where a) Part in the Slic3r's software to generate Gcode. b) Part printed in the robot working station by using the Gcode generated in Slic3r.

In spite of the additive manufacturing simulation was working correctly, it was necessary to develop a system control application capable of controlling the whole simulation through simple commands performed by a user.

As described in this thesis, the development of the system control application was entirely made in *Python*, by using the official *Python's* libraries. The final result was an App capable of controlling all the operations of the Additive manufacturing simulation, through a server/client socket connection.

In addition, the system control application is also capable of generating Gcode, automatically, by a simple operation as clicking a button. However, the goal of this feature, which was the creation of an automatic additive manufacturing simulation, was not accomplished due to technical issues related to the *RoboDK Python*'s library.

To sum up, the objective of this thesis was fully accomplished due to the fact that was developed a system capable of controlling an additive manufacturing simulation, by only performing simple commands in a graphical user interface. In addition, this thesis represents a positive contribution to the additive manufacturing world since it was shown how a system control application can be developed to execute additive manufacturing tasks.

5.2. Future Work

During the realization of this thesis a lot of questions were raised, however, they could not be answered due to the short period of time to find them out. All these questions are related to aspects that can be improved in the system control application, as well as in the additive manufacturing simulation since there is never a perfect solution.

As demonstrated, the additive manufacturing simulation is capable of generating all the printing movements automatically, and also to visualize the extrusion process by calling a *Python*'s program responsible to simulate the deposition of material. However, the first aspect has space to be improved due to the following reason:

The mathematical algorithm, used by RoboDK to generate the additive manufacturing simulation, assumes that the robot manipulator only has 3Dof, like if it was a 3D printer, which means that the orientation of the robot is not considered. This simplification restricts the system since it impossible to print parts that require different orientation on the robot's tool, such as in parts with eccentrics geometries, as it is mentioned by (Danielsen Evjemo et al., 2018). In addition, the previous mathematical algorithm also as the problem that it only creates linear type movements, which leads to the

generation of programs with thousands of movements. Hence, in a future work, it is necessary to research how these problems can be solved.

On the other hand, there are also problems related with the system control application since it is not possible to control the following parameters: extruder flow, extruder feed and the extruder and bed temperature, however, the control of these parameters only have influence in a real simulation. So, it is suggested as a future work, the implementation of a microcontroller, such as Arduino, to control all the mentioned parameters.

Finally, this thesis only focuses on the virtual simulation of an additive manufacturing process, so, hence, the next level of this work is to apply the developments done in this thesis in a real working system, in order to establish a test for additive manufacturing.

BIBLIOGRAPHY

Ålgårdh, J., Strondl, A., Karlsson, S., Farre, S., Joshi, S., Andersson, J., … Ågren, J. (2017). State-of-the-art for additive manufacturing of metals, (June), 98. Retrieved from https://www.swerea.se/en/news/state-of-the-art-for-additive-manufacturing-of-metals-report-from-ramp-up

Anandan, T. (2017). Robotics Industry Insights - Building the Future with Robotic Additive Manufacting. Retrieved September 2, 2018, from https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Building-the-Future-with-Robotic-Additive-Manufacturing/content_id/6860

ASTM F42/ISO TC 261. (2017). ASTM F42/ISO TC 261 Develops Additive Manufacturing Standards. Retrieved from https://www.astm.org/COMMIT/F42_AMStandardsStructureAndPrimer.pdf

ASTM F42. (2018). The Global Leader in Additive Manufacturing Standards. Retrieved from https://www.astm.org/ABOUT/OverviewsforWeb2014/Additive-Manufacturing.pdf

Berman, B. (2012). 3-D printing: The new industrial revolution. *Business Horizons*, *55*(2), 155–162. https://doi.org/10.1016/j.bushor.2011.11.003

Bos, F., Wolfs, R., Ahmed, Z., & Salet, T. (2016). Additive manufacturing of concrete in construction: potentials and challenges of 3D concrete printing. *Virtual and Physical Prototyping*, *11*(3), 209–225. https://doi.org/10.1080/17452759.2016.1209867

Bourell, D. L. (The U. of T. at A.), Leu, M. C. (Missouri U. of S. and T.), & Rosen, D. W. (Georgia I. of T.). (2009). Identifying the Future of Freeform Processing 2009. *Rapid Prototyping Journal*, 92. https://doi.org/10.1108/13552549910295514

Danielsen Evjemo, L., Moe, S., Gravdahl, J. T., Roulet-Dubonnet, O., Gellein, L. T., & Brøtan, V. (2018). Additive manufacturing by robot manipulator: An overview of the state-of-the-art and proof-of-concept results. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, (September), 1–8. https://doi.org/10.1109/ETFA.2017.8247617

Diaz, Y. (2016). AsyncIO for the Working Python Developer – Hacker Noon. Retrieved September 4, 2018, from https://hackernoon.com/asyncio-for-the-working-python-developer-5c468e6e2e8e

Eckel, Z. C., Zhou, C., Martin, J. H., Jacobsen, A. J., Carter, W. B., & Schaedler, T. A. (2016). Additive manufacturing of polymer-derived ceramics. *Science*, *351*(6268), 58–62. https://doi.org/10.1126/science.aad2688

European Patent Office. (2017). Patents and the Fourth Industrial Revolution, (December).

Fernandez-Vicente, M., Calle, W., Ferrandiz, S., & Conejero, A. (2016). Effect of Infill Parameters on Tensile Mechanical Behavior in Desktop 3D Printing. *3D Printing and Additive Manufacturing*, *3*(3), 183–192. https://doi.org/10.1089/3dp.2015.0036

Gary, H. (2013). Slic3r User Manual. Retrieved from manual.slic3r.org/expert-mode/infill

Ghezzi, C. (1985). Concurrency in programming languages: A survey. *Parallel Computing*, *2*(3), 229–241. https://doi.org/10.1016/0167-8191(85)90005-5

Guo, N., & Leu, M. C. (2013). Additive manufacturing: Technology, applications and research needs. *Frontiers of Mechanical Engineering*, *8*(3), 215–243. https://doi.org/10.1007/s11465-013-0248-8

Hong, M.-H., Min, B., & Kwon, T.-Y. (2016). The Influence of Process Parameters on the Surface Roughness of a 3D-Printed Co–Cr Dental Alloy Produced via Selective Laser Melting. *Applied Sciences*, *6*(12), 401. https://doi.org/10.3390/app6120401

Kivy. (2012). *Kivy Documentation*. Retrieved from https://media.readthedocs.org/pdf/kivy/latest/kivy.pdf

Lolzbot Company. (n.d.). Medium PLA 0.35mm Nozzle. Retrieved September 3, 2018, from http://devel.lulzbot.com/TAZ/3.1/software/2014Q1/slic3r/config/medium_PLA_no-support_pt35nzl_pt22layer/medium_PLA_no-support_pt35nzl_pt22layer.ini

Lozano-Perez, T. (1983). Robot programming. *Proceedings of the IEEE*, *71*(7), 821–841. https://doi.org/10.1109/PROC.1983.12681

Melchels, F. P. W., Domingos, M. A. N., Klein, T. J., Malda, J., Bartolo, P. J., & Hutmacher, D. W. (2012). Additive manufacturing of tissues and organs. *Progress in Polymer Science*, *37*(8), 1079–1104. https://doi.org/10.1016/j.progpolymsci.2011.11.007

Mings, J. (2017). MX3D is 3D Printing the Metal Structures. Retrieved September 2, 2018, from https://www.solidsmack.com/fabrication/mx3d-is-3d-printing-the-metal-structures-of-the-future/

Mountaqin, A. (2015). Offline programming software for industrial robots from RoboDK offers hundreds of virtual industrial robots from top robotics companies. *Robotics & Automations News*, (Offline programming software for industrial robots). Retrieved from https://roboticsandautomationnews.com/2015/07/14/offline-programming-software-robodk-offers-hundreds-of-virtual-industrial-robots-from-top-robotics-companies/540/

Murr, L. E., Gaytan, S. M., Ramirez, D. A., Martinez, E., Hernandez, J., Amato, K. N., … Wicker, R. B. (2012). Metal Fabrication by Additive Manufacturing Using Laser and Electron Beam Melting Technologies. *Journal of Materials Science and Technology*, *28*(1), 1–14. https://doi.org/10.1016/S1005-0302(12)60016-4

Noller, J., & Oudkerk, R. (2008). PEP 371 -- Addition of the multiprocessing package to the standard library | Python.org. Retrieved September 5, 2018, from https://www.python.org/dev/peps/pep-0371/

Nubiola, A. (2015). The future of robot off-line programming | CoRo Blog. Retrieved September 3, 2018, from http://coro.etsmtl.ca/blog/?p=529

Odom, M. G. B., Sweeney, C. B., Parviz, D., Sill, L. P., Saed, M. A., & Green, M. J. (2017). Rapid curing and additive manufacturing of thermoset systems using scanning microwave heating of carbon nanotube/epoxy composites. *Carbon*, *120*, 447–453. https://doi.org/10.1016/j.carbon.2017.05.063

Oluwatosin, H. S. (2014). Client-Server Model. *IOSR Journal of Computer Engineering*, *16*(1), 57–71. https://doi.org/10.9790/0661-16195771

Peters, T. (2004). PEP 20 -- The Zen of Python. Retrieved September 3, 2018, from https://www.python.org/dev/peps/pep-0020/

Pires, J. N. (2017). HYROMAN – Hybrid robotic additive HYROMAN – Hybrid robotic additive manufacturing platform for agile production of large multi-metal components, (January). https://doi.org/10.13140/RG.2.2.22065.38244

Pires, J. N., & Azar, A. S. (2018). Advances in robotics for additive/hybrid manufacturing: robot control, speech interface and path planning. *Industrial Robot*, *45*(3), 311–327. https://doi.org/10.1108/IR-01-2018-0017

Pycharm. (2018). Features - PyCharm. Retrieved September 10, 2018, from https://www.jetbrains.com/pycharm/features/

Python. (n.d.). 17.5. subprocess — Subprocess management — Python 3.4.9 documentation. Retrieved September 4, 2018, from https://docs.python.org/3.4/library/subprocess.html#popen-objects

Robodk. (n.d.). 2. robolink Module — RoboDK API Documentation. Retrieved September 4, 2018, from https://robodk.com/doc/en/PythonAPI/robolink.html#id1

Singh Bual, G., & Kumar, P. (2014). Methods to Improve Surface Finish of Parts Produced by Fused Deposition Modeling. *Manufacturing Science and Technology*, *2*(3), 51–55. https://doi.org/10.13189/mst.2014.020301

SLM Solutions Group – a leader in metal based 3D printing Company presentation. (2014), (April).

StackOverFloW. (2018). Stack Overflow Developer Survey 2018. Retrieved September 3, 2018, from https://insights.stackoverflow.com/survey/2018/#technology

Sukindar, N. A., Kharul, M., Mohd, A., Baharudin, B. T. H. T., Nor, C., Jaafar, A., … Ismail, S. (2017). Optimization of the Parameters for Surface Quality of the Open-source 3D Printing, *3*(1), 33–43.

Systems, M. E. (n.d.). *Mwes: system*.

Tan, X. P., Tan, Y. J., Chow, C. S. L., Tor, S. B., & Yeong, W. Y. (2017). Metallic powder-bed based 3D printing of cellular scaffolds for orthopaedic implants: A state-of-the-art review on manufacturing, topological design, mechanical properties and biocompatibility. *Materials Science and Engineering C*, *76*, 1328–1343. https://doi.org/10.1016/j.msec.2017.02.094

Tekinalp, H. L., Kunc, V., Velez-Garcia, G. M., Duty, C. E., Love, L. J., Naskar, A. K., … Ozcan, S. (2014). Highly oriented carbon fiber-polymer composites via additive manufacturing. *Composites Science and Technology*, *105*, 144–150.

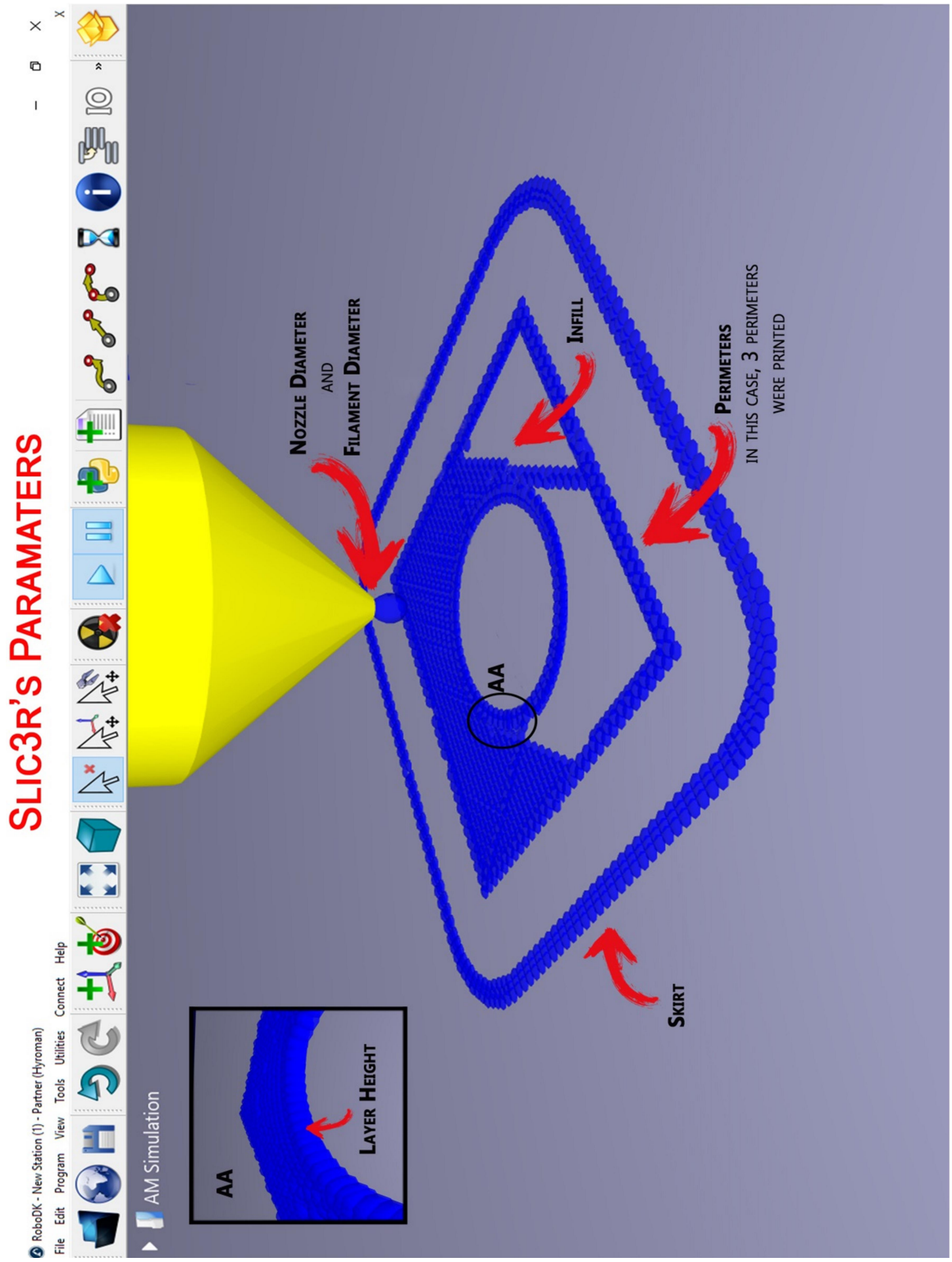
https://doi.org/10.1016/j.compscitech.2014.10.009

van Rossum, G. (2012). PEP 3156 -- Asynchronous IO Support Rebooted: the "asyncio" Module | Python.org. Retrieved September 4, 2018, from https://www.python.org/dev/peps/pep-3156/

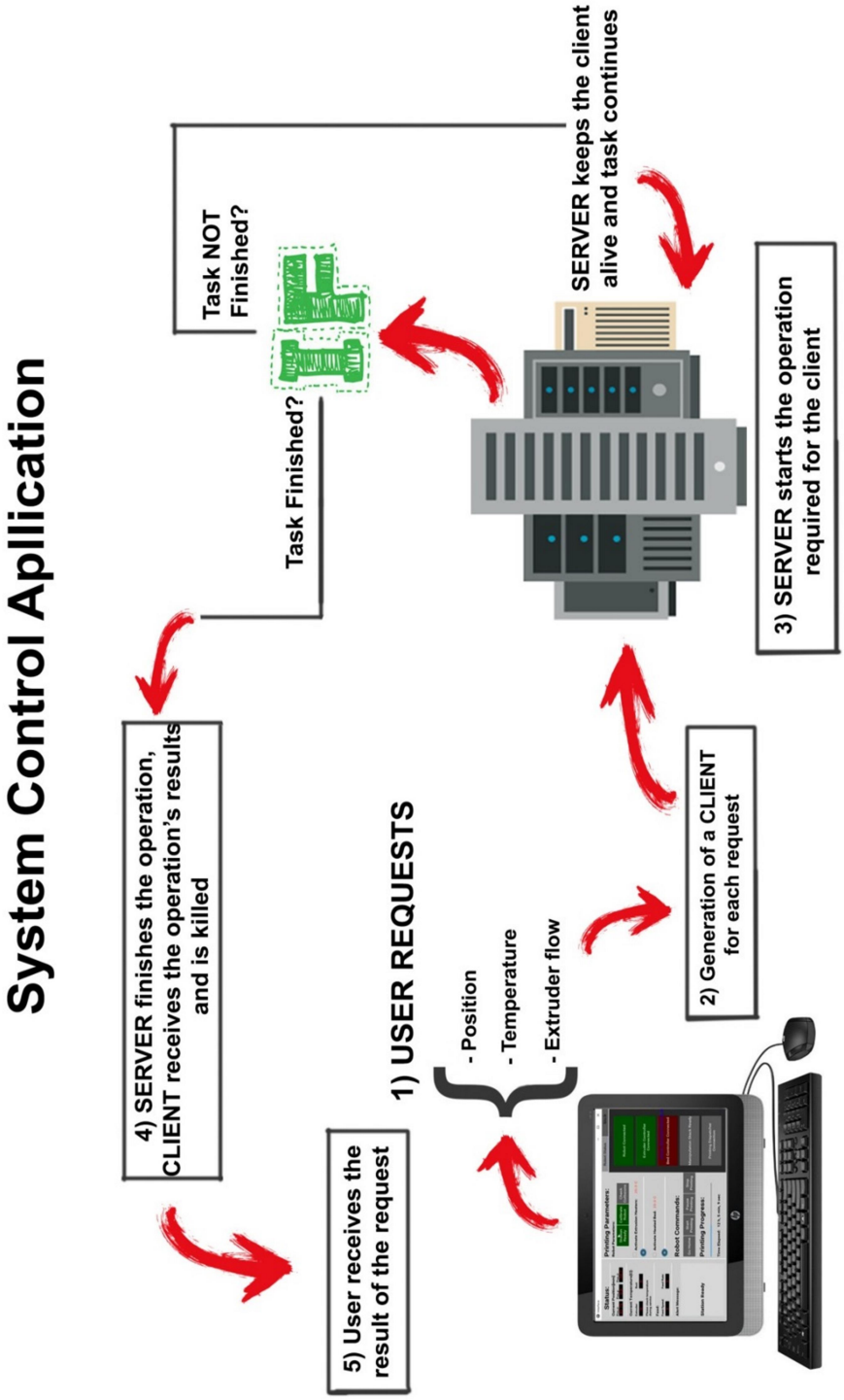
van Russom, G. (2012). 19.5.4. Transports and protocols (callback based API) — Python 3.7.0 documentation. Retrieved September 4, 2018, from https://docs.python.org/3/library/asyncio-protocol.html#tcp-echo-server-protocol

Xue, M., & Zhu, C. (2009). The socket programming and software design for communication based on client/server. *Proceedings of the 2009 Pacific-Asia Conference on Circuits, Communications and System, PACCS 2009*, 775–777. https://doi.org/10.1109/PACCS.2009.89

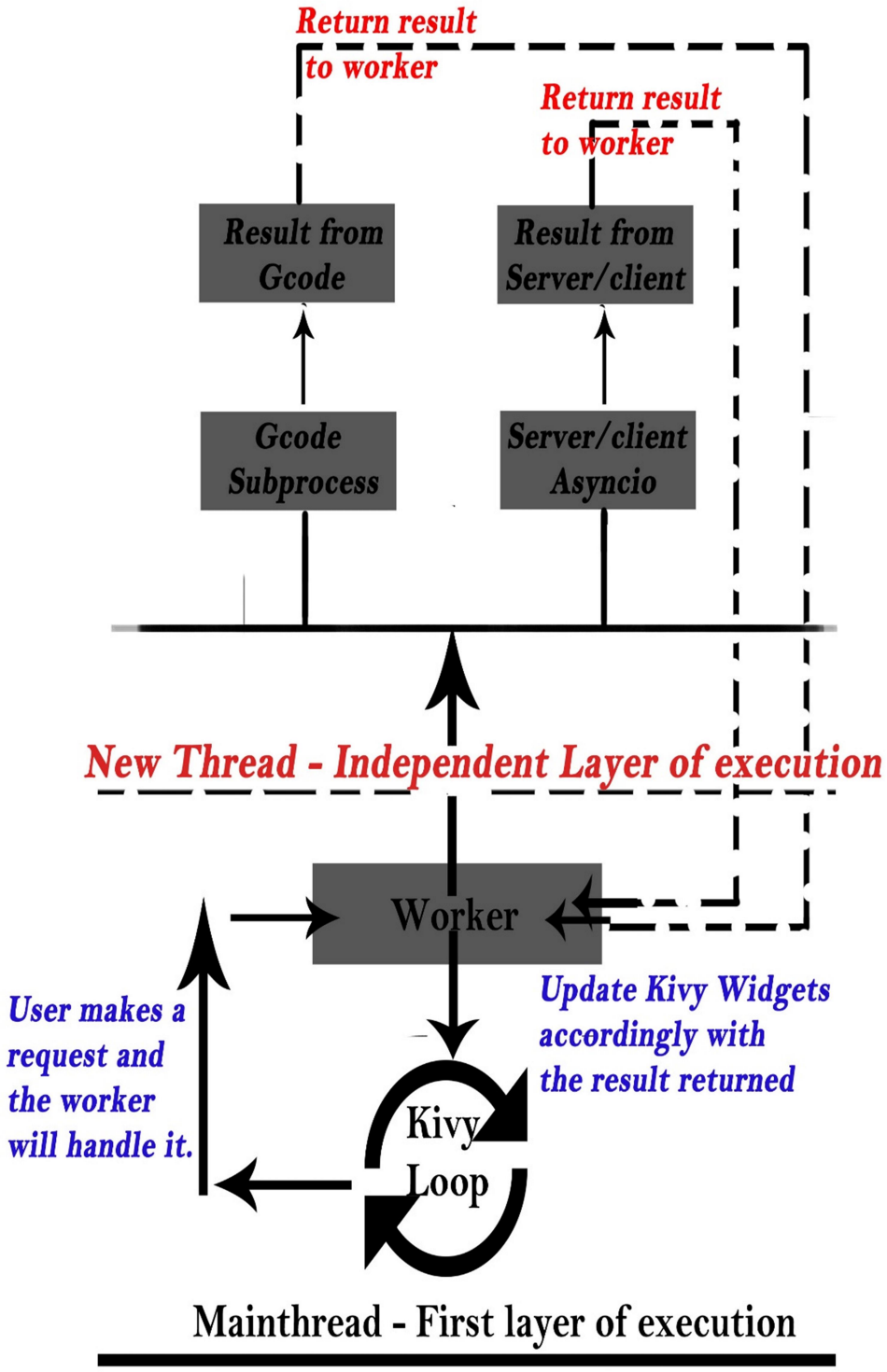
APPENDIX A


SLIC3R'S PARAMATERS
RoboDK - New Station (1) - Partner (Hyroman)
File Edit Program View Tools Utilities Connect Help
AM Simulation
AA
LAYER HEIGHT
NOZZLE DIAMETER
AND
FILAMENT DIAMETER
INFILL
PERIMETERS
IN THIS CASE, 3 PERIMETERS
WERE PRINTED
AA
SKIRT

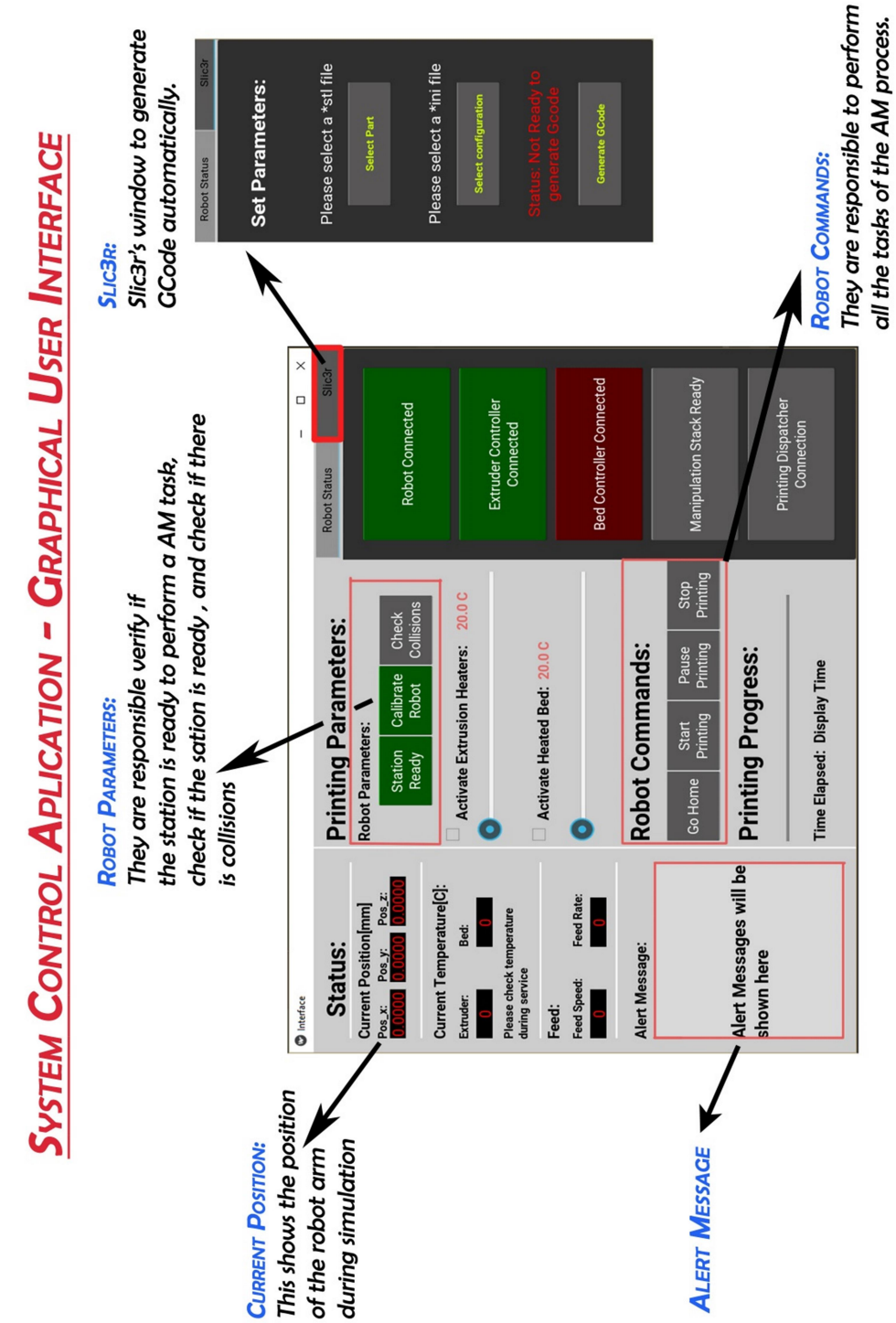
APPENDIX B


System Control Apllication
1) USER REQUESTS
- Position
- Temperature
- Extruder flow
2) Generation of a CLIENT for each request
3) SERVER starts the operation required for the client
SERVER keeps the client alive and task continues
Task NOT Finished?
Task Finished?
4) SERVER finishes the operation, CLIENT receives the operation's results and is killed
5) User receives the result of the request

APPENDIX C


Return result to worker
Return result to worker
Result from Gcode
Result from Server/client
Gcode Subprocess
Server/client Asyncio
New Thread - Independent Layer of execution
Worker
User makes a request and the worker will handle it.
Update Kivy Widgets accordingly with the result returned
Kivy Loop
Mainthread - First layer of execution

APPENDIX D


SYSTEM CONTROL APLICATION – GRAPHICAL USER INTERFACE
ROBOT PARAMETERS:
They are responsible verify if the station is ready to perform a AM task, check if the sation is ready , and check if there is collisions
SLIC3R:
Slic3r's window to generate GCode automatically.
CURRENT POSITION:
This shows the position of the robot arm during simulation
ALERT MESSAGE
ROBOT COMMANDS:
They are responsible to perform all the tasks of the AM process.
Interface
Status:
Current Position[mm]
Pos_x: 0.0000
Pos_y: 0.0000
Pos_z: 0.0000
Current Temperature[C]:
Extruder:
Bed:
Please check temperature during service
Feed:
Feed Speed:
Feed Rate:
Alert Message:
Alert Messages will be shown here
Printing Parameters:
Robot Parameters:
Station Ready
Calibrate Robot
Check Collisions
Activate Extrusion Heaters: 20.0 C
Activate Heated Bed: 20.0 C
Robot Commands:
Go Home
Start Printing
Pause Printing
Stop Printing
Printing Progress:
Time Elapsed: Display Time
Robot Status
Slic3r
Robot Connected
Extruder Controller Connected
Bed Controller Connected
Manipulation Stack Ready
Printing Dispatcher Connection
Set Parameters:
Please select a *stl file
Select Part
Please select a *ini file
Select configuration
Status: Not Ready to generate Gcode
Generate GCode