University of Edinburgh

School of Engineering

MECHANICAL ENGINEERING

BEng Mechanical Engineering Project 4

Final Report:

Robotic Printing of Continuous Carbon Fibre Composites: Printer Head Design and Path Planning

Student:

Yulin Wang

Yulin WANG (S1889038)

BEng (Hons) Mechanical Engineering

Supervisor:

Dr Dongmin YANG

Senior Lecturer

April 29th, 2020

PERSONAL STATEMENT

This is a novel project with rarely any existing attempt on a similar method being found, which was suggested by my supervisor and I extensively elaborated upon. My supervisor provided the idea of attaching the 3D printer on a robotic arm to print 3D continuous fibres as well as a substantial amount of guidance and support from the literature review to software and operating system to be used.

In terms of the theoretical work involved, one thing that I was intensively involved in is building the ROS workspace and using Python to simulate the robotic arm trajectory in Gazebo (Robot simulation software). Based on the tutorials online and some existing courses teaching how to use Gazebo, I spend paramount time working on understanding existing python scripts and rewriting the one incorporating my ideas to fit the requirements of the project. Further to this, some mechanical work has also been done including designing the printer head using CAD, this procedure requires quite a lot of effort from understanding the existing design, figuring out necessary components to model each part in CAD and assembling them. The last part of the project is about controlling the printer head using Arduino, Megatronics, specifically, although due to the limited time and some inevitable issues during the project, I was not given a chance to practically test the performance of the Arduino board in the laboratory.

Therefore, this project was processed smoothly to accomplish the objectives with a little adjustment of the determined plan at the beginning due to some undesirable situations. However, I committed that I have spared no efforts doing whatever can be done in limited time. The tasks in the project vary from mechanical design to electronics controlling and further to software simulation and Python programming, which not only gave me a systemic training across different disciplines but also required an inter-discipline knowledge. I have to admit that the time spent on electronics and Gazebo simulation has significantly exceeded my expectations. Arduino Board was not delivered until week eight, and the Gazebo tutorials online sometimes are difficult to understand, but from this experience, I have also learnt to always prepare a plan B and assume the worst case beforehand.

I, Yulin Wang, declare this project is of my own work, and where external sources have been used, they have been correctly cited.

Yulin Wang

April 22nd, 2020

SUMMARY

Title: Robotic Printing of Continuous Carbon Fibre Composites: Printer Head Design and Path Planning

Author: Yulin WANG (S1889038)

Date: April 20th, 2020

3D printing, also known as rapid manufacturing, has been widely used from aerospace, artificial organs, bespoke human implants to glass frame manufacturing and children's toys thanks to their high efficiency, better quality, unlimited geometry, cost-effectiveness and short manufacturing time. Hitherto, many investigations have been done in order to further explore the potential of 3D printing, and one of the hottest topics so far is to 3D print continuous fibres. Nowadays, Fused Deposition Modelling (FDM) is considered to be the most widely used printing method in fabricating prototypes and functional components. Conventionally, FDM is only used in desktop 3D printers, which have high controllability but coupled with high spatial and dimensional restrictions. In order to tackle this restriction, two robotic manufacturing methods – Automated Tape Placement and Automated Fibre Placement have been proposed. By attaching 3D printer heads on robot arms, larger products can be printed. With the aid of the robot arm, 3D printer heads could also be more agile, manoeuvrable and capable to achieve sophisticated geometries. Nevertheless, ATP and AFP typically use relatively fragile and compliant thermoplastic feeding materials that might lead to poor quality of the product, therefore, to solve this problem, continuous carbon fibre is thought as a good alternative due to its high strength and stiffness. However, given the different mechanical properties of carbon fibres from traditional materials, robotic carbon fibre printing adds several complexities including printer head configuration, manufacturing process, robotic controlling, etc. So far, there is no available mature software that can simulate the path of a robotic arm with taking material properties into account. Therefore, this area could potentially be quite promising and worth investigating.

Overall, the project focuses on the a) new 3D printer head design; b) understanding the Arduino circuit board for 3D printer and c) trajectory manipulation using simulation software after mounting the printer head to a viable robotic arm. This report gives a detailed literature survey coupled with my critical understanding, project timeline management, all the technical work done throughout the entire project and a discussion of orientation for the future.

CONTENTS

SUMMARY OF RESOURCES

Item	*Cost*
Total Money Spent	£ 115.00
Money Spent on Component Ordering	£ 86.00
Total Technician Hour used	0 Hours
Other Resources Used	Online ROS Tutorial – $36 ~ Approx. £ 29.00

TABLE FOR WORD COUNT BY CHAPTER

Chapter No.	*Name of Chapter*	*Word Count*
1	INTRODUCTION	701
2	LITERATURE REVIEW	1454
3	MECHANICAL DEISNG OF PRINTER HEAD	1002
4	PANDA ROBOT ARM CONFIGURATION AND SETTING UP	2012
5	PYTHON-ROBODK & PYTHON-GAZEBO INTERFACE	1589
6	OVERVIEW OF GAZEBO SYSTEM	838
7	ELECTRICAL COMPONENT – ARDUINO BOARD	762
8	PROJECT MANAGEMENT	308
9	DISCUSSION	428
10	CONCLUDION	479
Total		9582

Acronyms & Abbreviations

Polylactic Acid	**ABS**
Automated Fibre Placement	**AFP**
Additive Manufacturing	**AM**
Automated Tape Placement	**ATP**
Computer-Aided Design	**CAD**
Digital Light Processing	**DLP**
Damped Least Square	**DLS**
Degree of Freedom	**DoF**
Electronic Beam Melting	**EBM**
Fused Deposition Modelling	**FDM**
Integrated Circuit	**IC**
Integrated Development Environment	**IDE**
Partial, Integration, Differentiation	**PID**
Acrylonitrile Butadiene Styrene	**PLA**
Rapid Manufacturing	**RM**
Robot Operation System	**ROS**
Stereolithography	**SLA**

1. Chapter 1: INTRODUCTION

Robotic manufacturing is considered to be one of the fastest-growing industrial sectors in recent years. Firstly originated in 1937, the earliest known industrial robot was invented by Griffith Bill [1], then about 18 years later, after George Devol teaming up with Joseph Engel Berger, the very first industrial robot firm was established and put into service at General Motors plant in 1961 [2], and two images below schematically reflects the robot configuration, shown in Figure.1 and Figure.2.

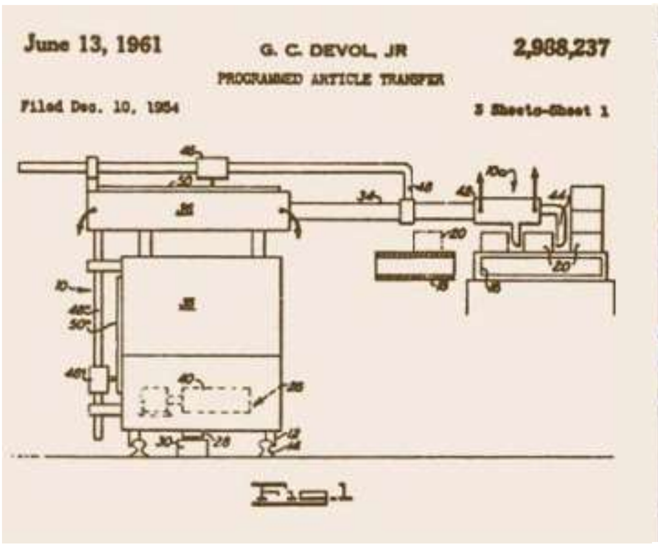


Figure 1 – Earliest Industrial Robot [1]

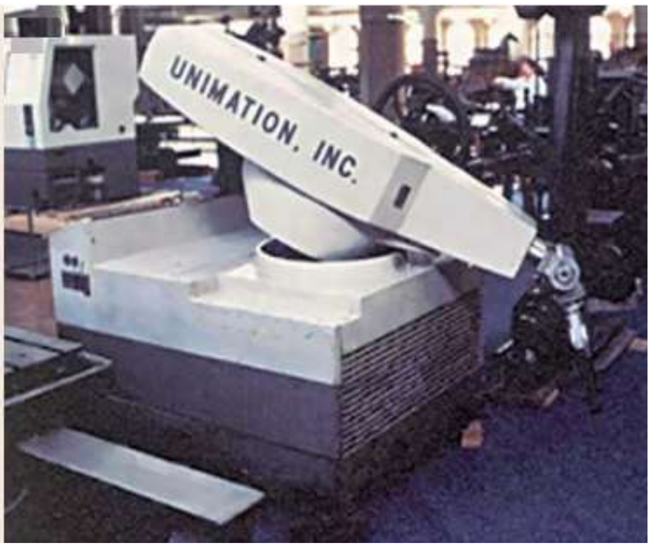


Figure 2 – Robot from first robot company [2]

Depending on various criteria, robotic manufacturing can be classified into many different categories including a) Type of movement; b) Application; c) Configuration and d) Ability to be collaborative with humans or other autonomous devices.

Despite that many robotic manufacturing techniques mentioned above are available, most of them are still in the formative stage. Amongst all of those robotic manufacturing technologies, one of the flourish fields nowadays is Robotic 3D printing. Thanks to some traits of robotic arms, high Degree of Freedom (DoF), for example, robotic 3D printing has been considered as an extension of traditional 3D printing due to many benefits in the field of large-scale manufacturing and fast construction of highly customised features without human intervention.

In terms of 3D printing technique, Fused Deposition Modelling (FDM) is currently known as the most commonly used 3D printing method, and Automated Fibre Placement (AFP) or Automated Tape Laying (ATP) are two pervasive automated processing techniques using robotic arms to place materials, these two methods have shown a great promise in automated manufacturing and a paramount similarity to FDM and robotic 3D fibre printing technique [3]. However, although AFP and ATP have presented great benefits in improving efficiency in many aspects such as composite fabrication and material coating, when it comes to manufacturing geometrically complicated 3D structures, these two methods have relatively lame performances due to the poor mechanical properties, for instance, the fragility of its feeding materials. Therefore, in order to overcome the gap and improve the performance, one novel idea proposed in recent years is to mimic the working principle of AFP and ATP whilst adding a few bundles of continuous fibre into the feeding materials to boost the mechanical properties.

1.1 Aims and Objectives

Noticing the gaps of AFP and ATP mentioned above, in order to enhance the quality, strength and stiffness of printed products so that they can fit the mechanical and structural requirements of complicated spatial 3D geometries and provide enough strength, this project will focus on proposing a 3D printer head of using continuous fibre as feeding material and make simulations first on RobotDK-Python and later on Gazebo-Python Interface to mimic the trajectory of a panda robotic arm. However, unlike typical thermoplastic materials, it is not easy to stop printing simply by stop heating the materials because fibres do not melt, therefore, an innovative 3D printer head design should be considered. To ensure that the entire project is finished in a timely and costly way, some main steps have been made to cover project tasks like trajectory design of the robotic arm (main task), understanding the control Arduino board of 3D printer head control and designing the printer head.

Typically, those aims mentioned above can be further subdivided into six objectives

- Get the state-of-the-art FDM printing, ADP and ATP working principles
- Printer Head Design
- Arduino Board Configuration and Arduino Path Planning Control
- Robotic Arm configuration selection
- Movement of robotic arm simulation in software Gazebo-Python Interface in Ubuntu
- Gazebo and ROS overview

1.2 Hardware and Software Description

In this Project, a Panda robotic arm and the embedded circuit board Megatronics v3.3 have been used as the main hardware. In terms of Software, as aforementioned, RoboDK was initially applied. However, due to the fact that RoboDK is not free with only thirty days of trial. Later, Gazebo was implemented to visualise the trajectory of the panda robotic arm, and throughout the entire procedure, Python was used as the main programming language to control the robot movement.

2. Chapter 2: LITERATURE REVIEW

In this section, four aspects – 3D printing, robotic arm manufacturing, Automated Composite Manufacturing (ACM) including ATP and AFP and robotic path planning have been reviewed. Literature review of these four aspects mainly focuses on the state of the art, currently available methods, advantages and disadvantages of existing approaches.

2.1 Introduction of 3D Printing and Additive Manufacturing

Firstly introduced and proposed in 1981 by Hideo Kodama [4], Additive Manufacturing (AM) is a process for 3D fabrication by putting materials layer by layer to accomplish a wide range of structures and complicated geometries. During the past three decades, 3D printing has experienced a prosperous development, by the mid-2000s, the democratisation of 3D manufacturing has grasped the public's attention. Till today, due to the advantages such as less energy usage, fewer wastes, high customisation, high accessibility, 3D printing has been used in a wide range of applications from aerospace, healthcare to architecture construction and affordable houses prototyping. Figure.4 shows a house manufactured by 3D printing techniques, and Figure.5 uses 3D printing to print out some tissues.

Figure 3 – 3D Printed Architecture [5]

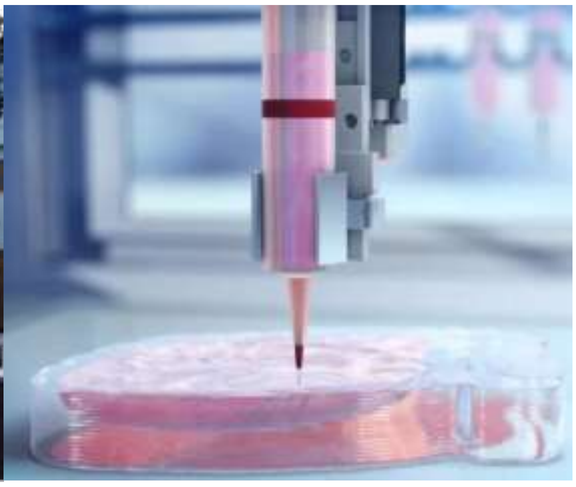


Figure 4 – 3D Printed Artificial Tissue [5].

There are two main steps in the 3D printing process – 3D CAD modelling and printing. Figure.6 shows the detailed procedure of 3D printing all the way through CAD modelling to 3D object printing. In terms of 3D CAD modelling, designers will firstly use CAD software to make a model of the object, then export those CAD files to an STL file for 3D printing. Finally, 3D printers will print out the objects. However, as for the printing process, one thing worth noting is that depending on different printing orientations, sizes of objects and amount of supporting materials, the time required for 3D printing is different. Figure.7 and Figure.8 below have shown the estimated time depending on different orientations.

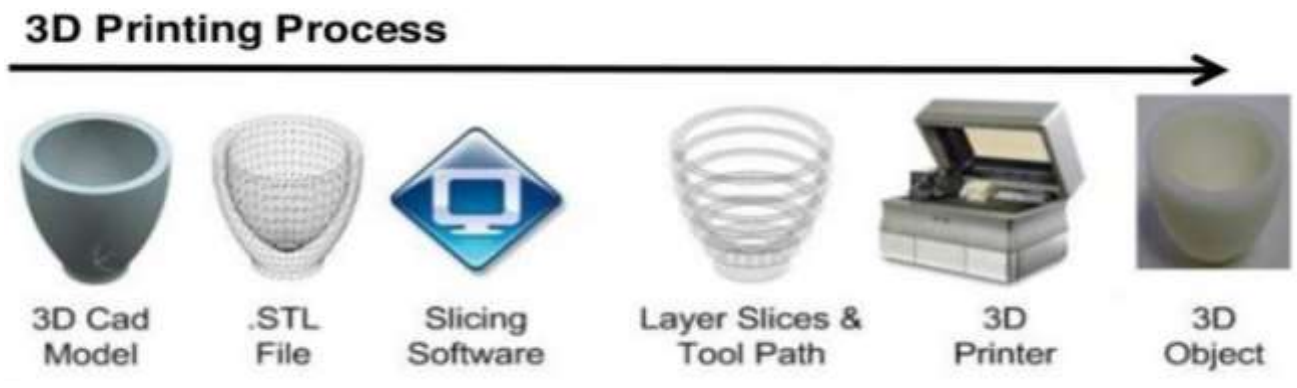


Figure 5 – Process of 3D Printing [6]

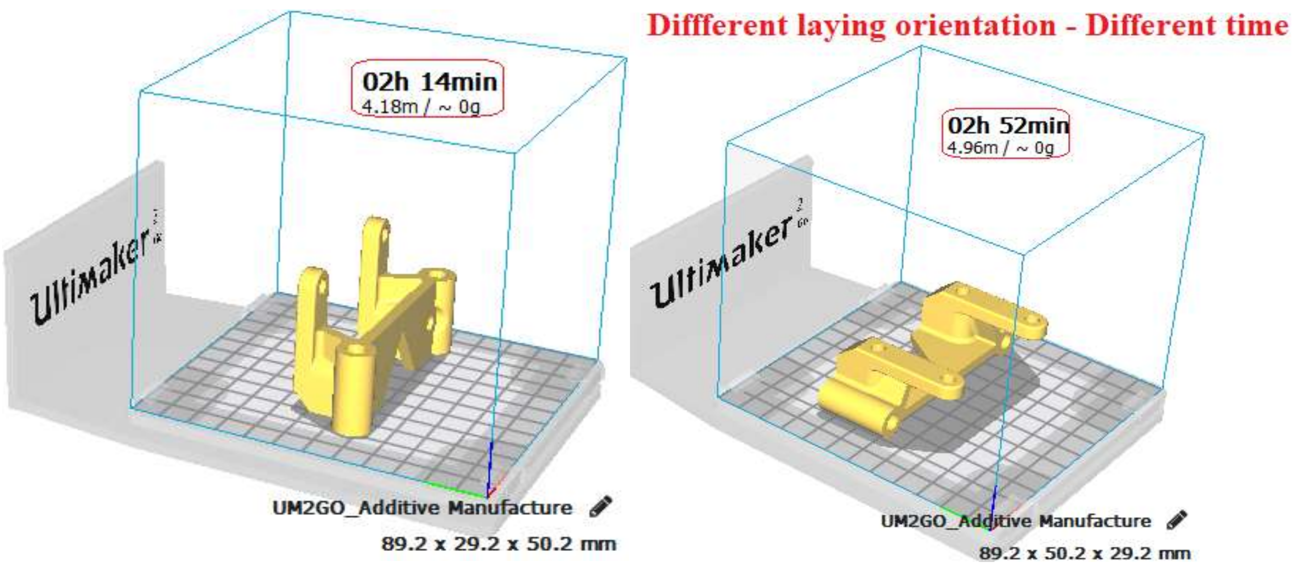


Figure 6, Figure 7 – Different time required depending on different printing orientation

Depending on different objects to be printed out, printing materials will vary a lot. So far, a big domain of materials have been used in 3D printing industry such as Polylactic Acid (PLA), Acrylonitrile Butadiene Styrene (ABS), organic human tissues and even metals [7], and one recent upcoming material is carbon fibre because carbon fibre has presented a series of distinctively advantageous material properties such as high strength and stiffness.

As mentioned before, there are many possible materials for 3D printing, and similarly, thanks to the continuous efforts made by researchers, many feasible 3D printing techniques have also been introduced including stereolithography (SLD), Fused Deposition Modelling (FDM), Inkjet Printing and Digital Light Processing (DLP). Amongst all of them, FDM is currently one of the most widely used 3D modelling techniques. In its printing process, filaments are firstly melted inside of a heated nozzle, then printed out into 2D contours and eventually solidified and finished when the nozzle stops heating. Then by piling up those layers, a 3D geometry can be constructed. Conventionally, FDM uses benchtop FDM 3D printer due to its ease of operation, low price and energy consumption. However, it is undeniable that using desktop printers will lead to a few restrictions. As traditional 3D printers only support translational movements of nozzles, multi-plane layering and complicated spatial structures are not achievable by using them. Further to this, the typical 3D printers are small which significantly constrains the size of the products. Moreover, due to the working principle of benchtop FDM 3D printers, only those thermoplastic materials could be used as feeding materials as they can melt and solidify quickly.

2.2 Robotic Manufacturing

In the 21st century, robots have been more and more widely used in manufacturing. Robots have presented many superiorities in the manufacturing industry including high efficiency from raw materials to ultimate product packing, continuous production with 24/7, high flexibility that can be customised to perform complex functions, cost-effectiveness and high accuracy. Robots have been applied in various fields including vehicle body assembling, rocket painting, surface coating and press tending, and there is still a huge tendency of technology of robot manufacturing.

2.3 State-of-the-art ACM – ATP and AFP

Automated Composites Manufacturing (ACM) is a typical fibre specialising robotic manufacturing technique using robots to place composite materials and build a structural layer at a time, shown in Figure.9. ATP and AFP are two commercially proven and widely used methods of ACM. With the help of ATP and AFP, manufacturers can explore those previously unavailable possibilities of composite construction, curving in an oven or autoclave [8], for example, and significantly reduce material consummation and labour cost [9]. One application of AFP is to construct highly bespoke layers of materials as each ply can be placed by pre-programmed robots at different angles to best carry the required loads. Thanks to this trait, AFP has already been implemented and practised in many areas such as rocket surface painting (Figure.10) and turbine blades [10] [11] [12].

However, despite many benefits, ATP and AFP still have a few non-negligible deficiencies. Depending on material properties, consolidation force, lay-up speed, curing and melting temperature [13] [14], there are three different common flaws using ATP or AFP, which Boundary Coverage (Figure.11), Twist (Figure.12) and Loose Tow (Figure.13), respectively. These flaws have significantly declined the quality of the painting process. By continuum investigations, scientists have found out that the main reason of all those flaws is that the commonly used materials are typically not strong and stiff enough, and the fragility often brings a series of secondary problems, coating fracture and thereby making inner materials directly exposed to the outside environment, for instance. This exposure will further lead to some catastrophic issues such as parts broken or materials deformation.

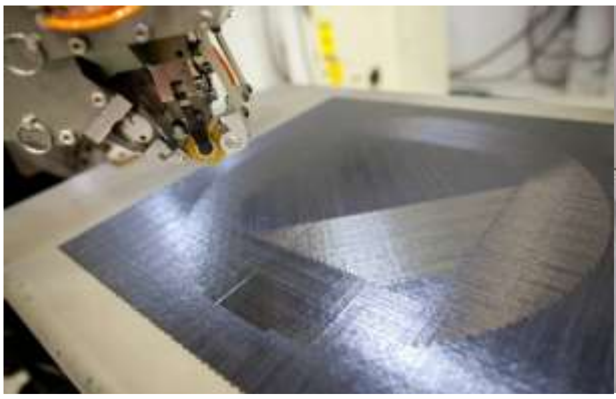


Figure 8 – ACM printing on plane surface [9]

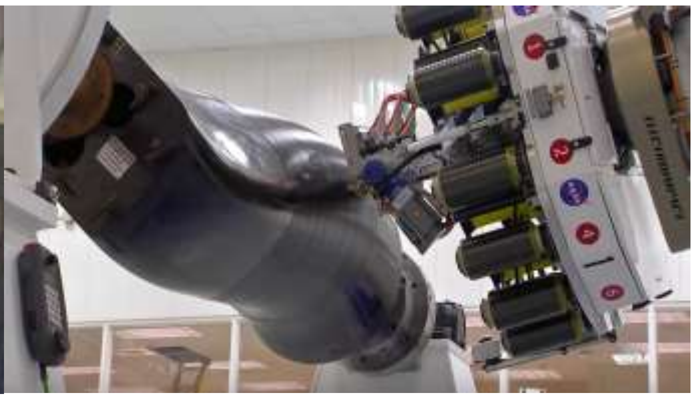


Figure 9 – ACM techniques used for rocket coating [15]

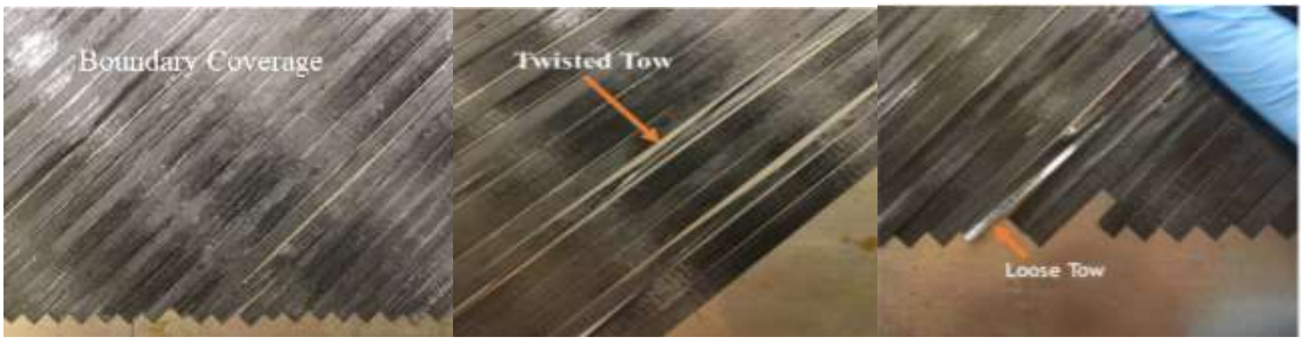


Figure 10, Figure 11, Figure 12 – 3 common flaws of using AFP and ATP [13] [14]

Therefore, in order to solve this material issue, one novel method proposed recently is to engrave some bundles of continuous carbon fibre into the plastic matrices of conventional thermoplastic materials, namely, the continuous fibres will be coated with a plastic matrix of thermoplastic materials. Nevertheless, as the mechanical properties of carbon fibre are different from that of thermoplastic materials, some adjustments on the printer head will be required to fit the mechanical properties of carbon fibres.

2.4 Robotic Control and Path Planning Review

Although industrial automation has dramatically improved efficiencies, for further ensuring the best quality of products, maximum capacity and productivity of manufacturing plants in limited time, the control of robots is also considered to be crucial. As mentioned above, using robotic arms for 3D printing continuous fibre brings a number of benefits compared to traditional 3D printing methods and ATP/AFP techniques. However, this new technique also concomitantly leads to a few problems. One problem is the requirement of achieving delicate robotic arm control to ensure the desired path planning. For a long time, the control of robotic arms is considered as a tricky problem because of a) the highly non-linear motion caused by high DoF, typically five to seven on a robot arm; b) the danger of colliding with the robot itself or other co-working robots or human being; and c) physical restrictions, the maximum range of operation angle, for instance. Therefore, to solve these problems, some researches and methods have been done so far including using border based description of obstacle [16] [17], inverse kinematics [18], etc. to compute the possible trajectories to either obtain minimum operational time and energy consummation or shortest distance [19]. Each of those existent methods has represented a valid point to improve robotic arm trajectory. A typical path planning method for a robotic arm is based on the context of using robots to grasp and transport an object in which spatial analysis can be conducted to compute its motion trajectory whereas in this project, although the robotic arm is mainly used for 3D printing, the motion of robotic arm will dominantly stay on a 2D surface for simplicity and will be extended into 3D later. Furthermore, as there are limited chances to conduct experiments, the simulation can only stay on theory.

One of the pervasive methods is inverse kinematics, and there are two sub-categories of inverse kinematics, namely, numerical and analytical methods. Depending on the different robot types and various required minimised factors, many techniques could be applied including the Newton-Raphson approach [20], the steepest descent approach [21] and the Damped Least-Square (DLS) approach [22] [23]. In the Gazebo simulation, the software automatically uses Inverse Kinematics to calculate the best trajectory so that users do not have to calculate themselves.

3. Chapter 3: MECHANICAL DESIGN OF PRINTER HEAD

In this section, the main components consisted of the mechanical assembly aforementioned will be explained in detail, and the software control section will be explained in the next section.

3.1 MECHANICAL PART – PRINTER HEAD DESIGN

The first task that has been done is the mechanical design of the printer head using Autodesk Inventor (CAD). It is time-wasting and unnecessary to design a completely new printer head with every single part to be designed manually as there are many factors to consider. Considering that there are already quite a few existing printer heads in the market, the strategy of designing a printer head is to mimic the components that are available and existent so far because this can not only reduce a significant amount of work but also makes it easier for further practical assembling by simply buying some existing components and assembling them with those self-designed and self-fabricated parts. In this case, Mark Forge Two (Figure.14) has been chosen as the design reference.

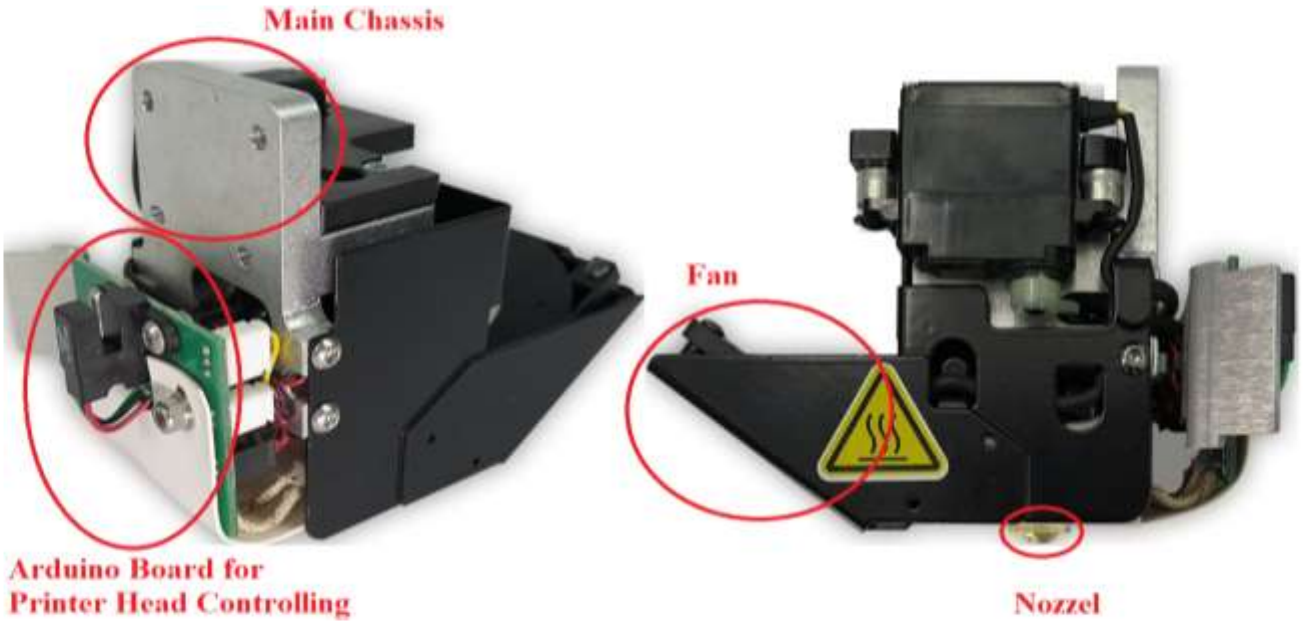
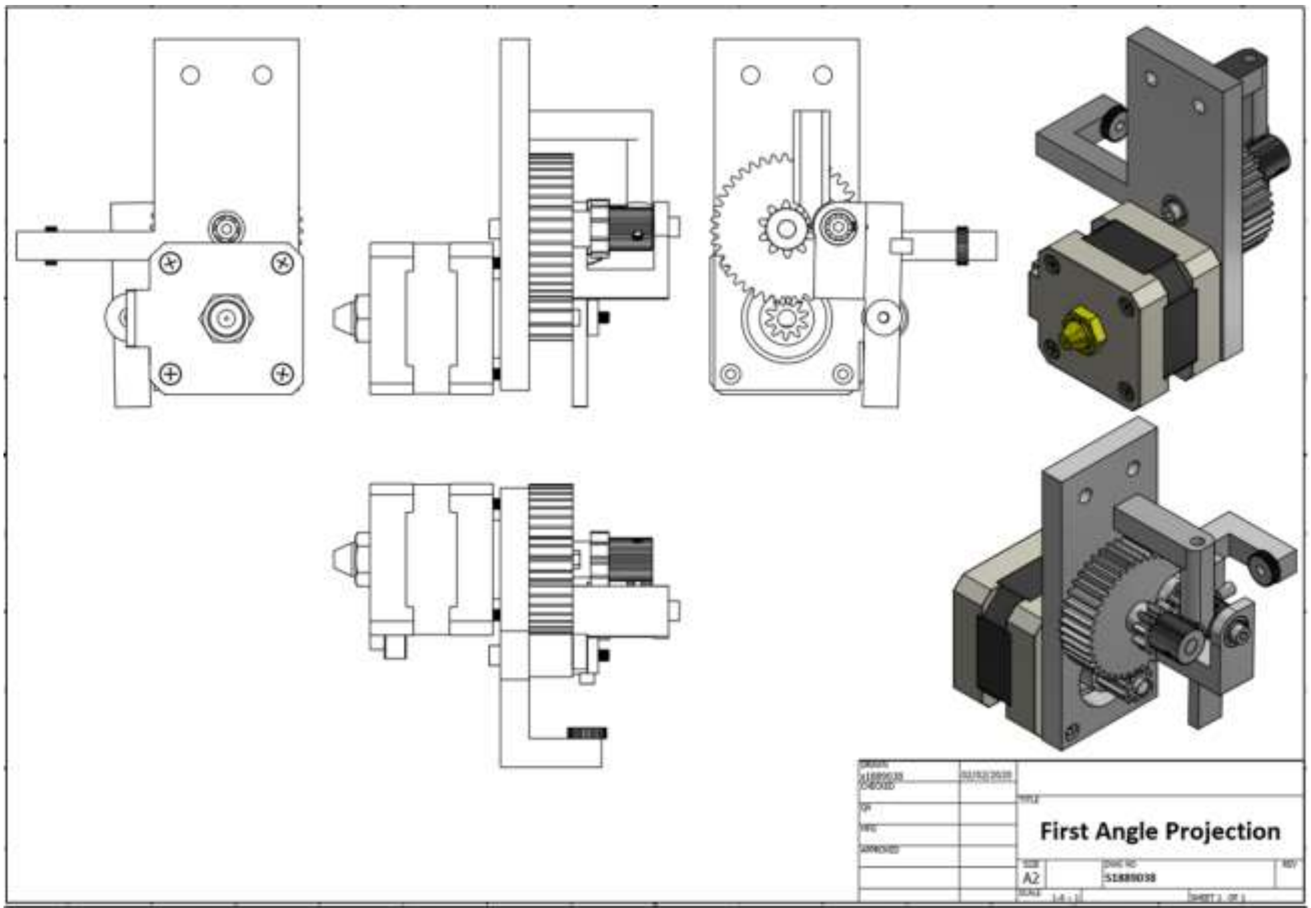


Figure 13 - Mark Forge Two Printer Head [24]

Based on the Mark Forge Two printer head and available components of assembling a 3D printer head, a new mechanical design has been made. Figure.16 below gives the first view and third view projection of the 3D printer head assembly, and Figure.17 shows the explosion view of the 3D printer head. In the sections below, each component of the printer head will be elaborated.



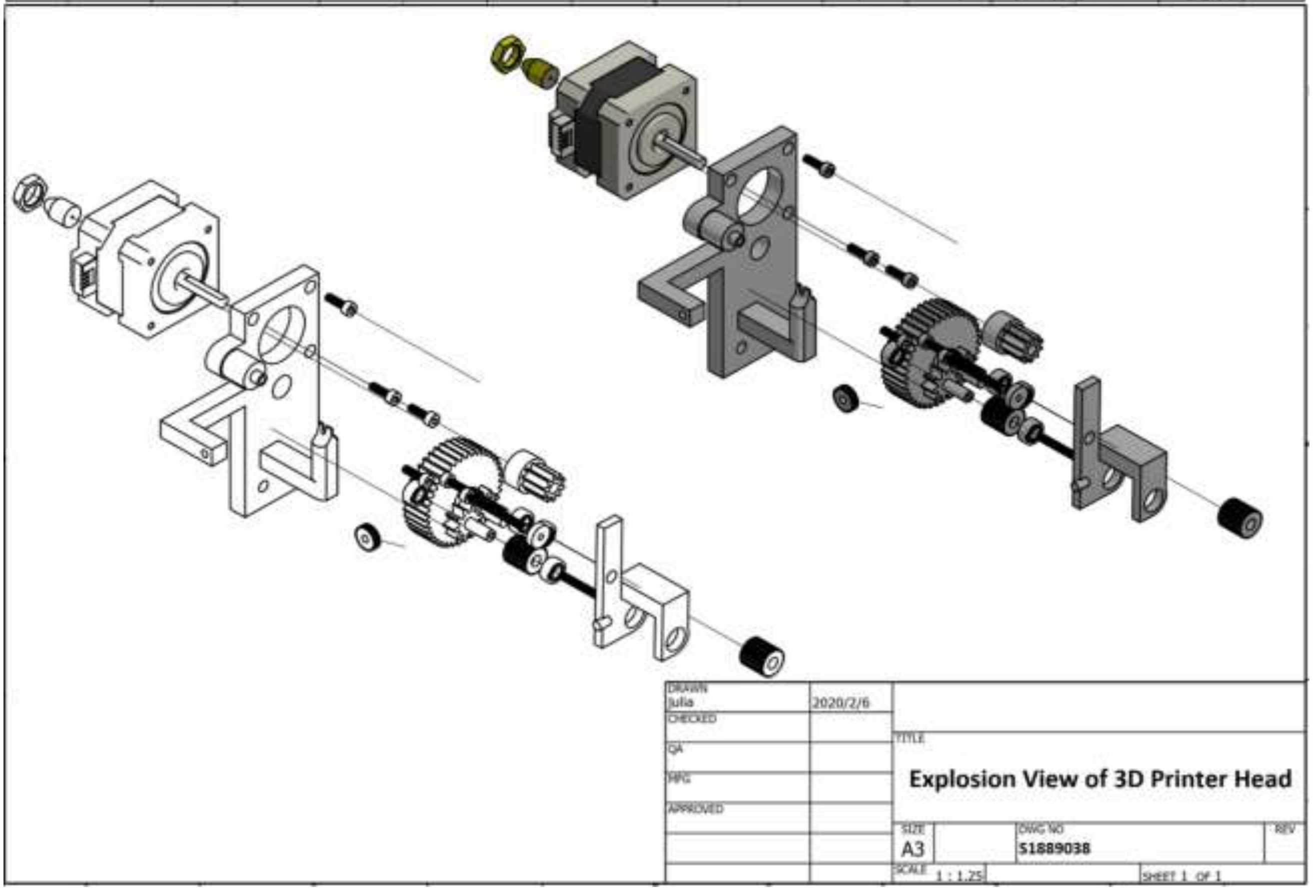


Figure 14 (Upper) – the 1st and 3rd angle projection of printer head; Figure 15 (Lower) – Explosion View of Printer Head Design

3.2 3D PRINTER HEAD DESIGN IN DETAIL

As shown in Figure.15, there are a considerable number of parts, from self-designed components, such as idler arm and idler gear to standard components including gears, nuts, washers and bolts. However, as for the printer head design, six of them are considered to be critical including printer head nozzle set, idler arm, idler gear, main chassis, main drive gear and sync gear. Figures below have shown each part, and the function of each component will be clarified below.

3.2.1 Printer Head Nozzle Set Overview

First and foremost, as far as the printer head nozzle set shown in Figure.16, it contains a servo motor to generate force, a hot end to heat up materials, a nozzle and several gears to transmit the force and drive the feeding filament to be extruded out. In terms of the servo motor, unlike regular DC motors that rotate continuously under the given power, the servo motor has a position feedback mechanism to control its motion and final position. This trait allows it to have the capability to control position and speed very precisely and fit the requirement of 3D printing in a better way. The design of servo motor was based on an existing product (Figure.17 [25]). However, this design in only a mechanical design excluding the inner electrical functional parts. The hot end is also one of the most important parts of the 3D printer as it is where the plastic is melted and extruded in layers. In the project, as the diameter of the filament is 0.375 mm, the diameter of the nozzle should be around 0.380$\pm$ 0.005mm to ensure that it fits the filament. Further to this, another critical factor of hot end is the material, considering that 3D printer will work under high temperature, it is important to make sure that the nozzle can endure the maximum possible temperature, which is typically around 220 degree Celsius [26].

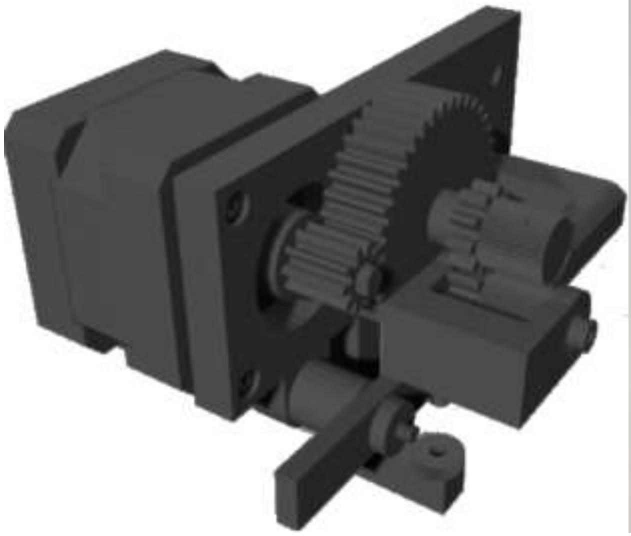


Figure 16 – Self-Designed Printer head nozzle part

Figure 17 – An existing servo motor [25]

3.2.2 Idler Gear and Idler Arm

The next two crucial components are idler gear and idler arm, shown in Figure.19 and Figure.20. Idler is a spring-loaded wheel that pushes the filament down through the nozzle [27]. In order to make sure that the material filaments go through the hot end and the nozzle in a moderate speed, namely, neither too quickly nor to slow, most printer heads have an idler arm and ideal gear to adjust the tension on the idler so that the idler does not squeeze the materials filament too hard or too little.

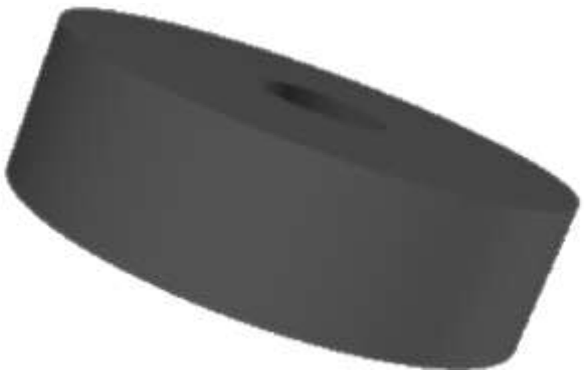


Figure 18 – Standard Part – Idler Gear

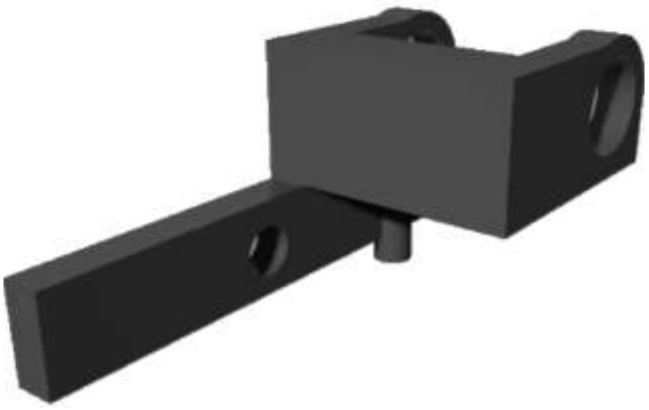


Figure 19 – Self-Designed Part – Idler Arm

3.2.3 Main Chassis Design

Another critical part is the main chassis, the CAD design has been shown in Figure.21 with a real device shown in Figure.22. The function of chassis is to hold everything on the printer head together, and early printer heads had chassis made out of plywood, however, nowadays, most of chassis are made of steel metal or aluminium beams. Generally, the more rigid the chassis is, the more precise the printer's movement will be as every component will be held very tightly so that the position and coordinate of each component can be calculated.

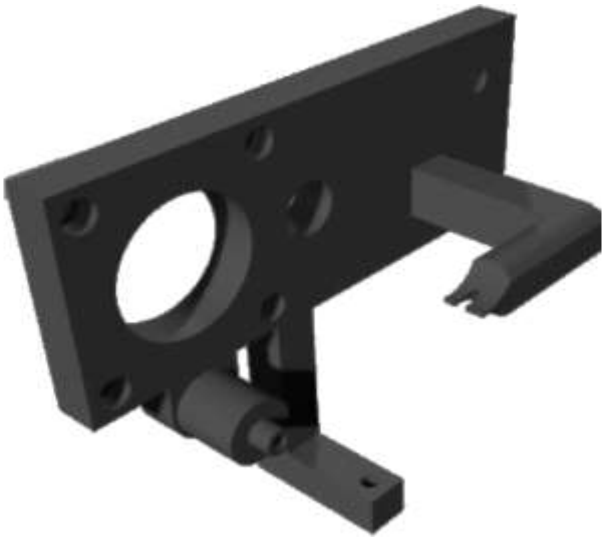


Figure 20 – Self-designed Chassis in CAD

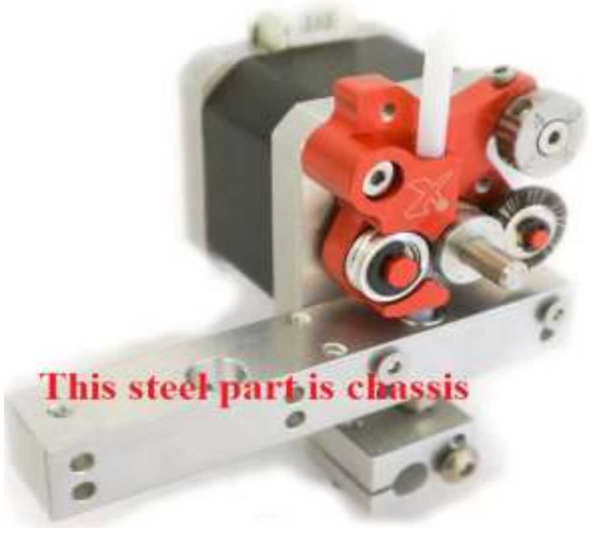


Figure 21 – Chassis from an existing 3D printer [28]

3.2.4 Gears and Transmission Components

The last two mechanical components are the main drive gear and sync gear, see Figure.23 and Figure.24. In principle, the main gears will be driven by the servo motor, and the sync gear will automatically be driven by the main gear to transmit force to extrude material filaments. Explicitly, when the servo motor spins, the spinning of the shaft connected to the servo motor will drive the main gear to spin. As the main gear is connected to the sync gear, the sync gear will also automatically be driven with the same angular velocity as the servo motor, and thereby squeeze and extrude the filament to print out. One thing worth noting here is that this servo motor is used for driving gears to extrude feeding materials. In addition to this, another servo motor will also be required to drive the cutter to cut off the fibre filament. Those two gears are not designed by the author but grabbed from

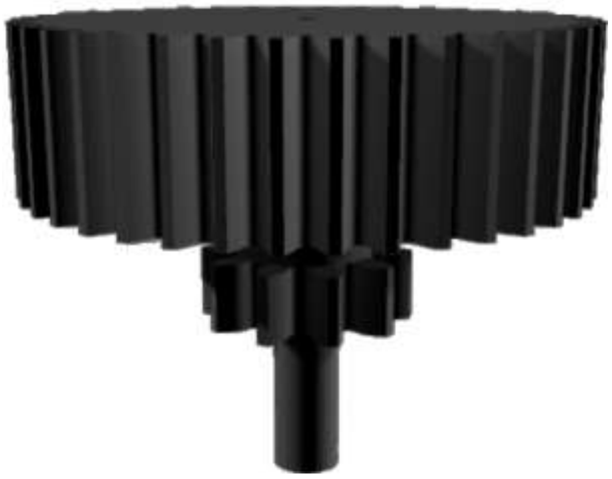


Figure 22 – Standard Part – Main gear, driven by servo motor

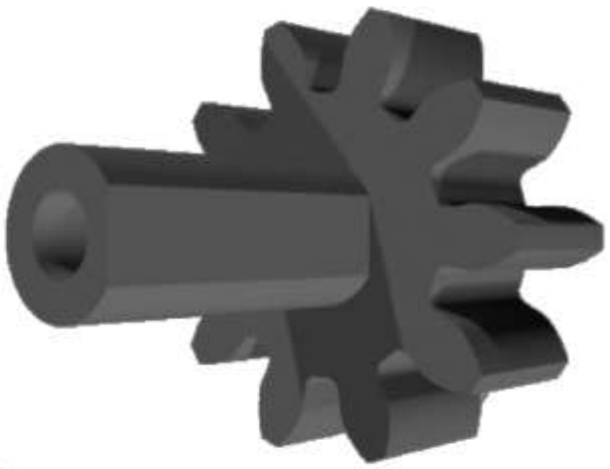


Figure 23 – Standard Part – Sync gear, driven by main gear

4. CHAPTER 4: PANDA ROBOT ARM CONFIGURATION AND SETTING UP

4.1 ROBOTIC ARM CONFIGURATION

In this section, some software preparation such as system installation and set up before the actual robotic arm trajectory simulation has been finished from the initial robotic arm workspace and environment construction to the robot configuration design and further to manual control and simulate the trajectory of the robotic arm.

4.1.1 Robotic Arm Control Procedure Literature Review

The first thing that has been done is to figure out the workflow of the entire robot arm system for 3D printing so that readers can have a full picture, as shown in Figure.25.

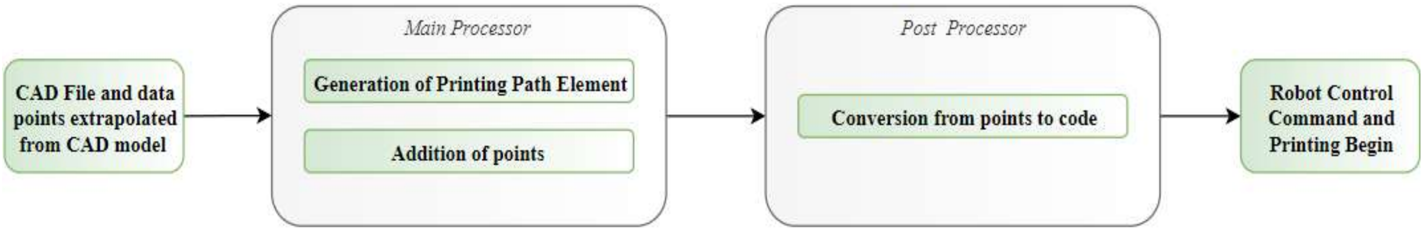

CAD File and data points extrapolated from CAD model
Main Processor
Generation of Printing Path Element
Addition of points
Post Processor
Conversion from points to code
Robot Control Command and Printing Begin

Figure 24 - Workflow of Robotic Arm from CAD to Path Planning

Similar to the workflow of using traditional desktop 3D printers, in order to use a robot arm to accomplish 3D printing, the first step is to build a 3D CAD model. Then, the robotic arm controller, also known as the main processor, mentioned in Figure.25, will automatically convert those data points extrapolated from CAD model into control command and generate corresponding codes and trajectories in post-processor. Further to this, the robotic arm will be manipulated and driven by the auto-generated code and print out the desired 2D contour shapes. However, as no software currently is available to automatically generate the robot path itself, some human intervention will be required, by which it means, the codes of robot trajectory controlling should be written by the designer to manipulate the robotic arm trajectory.

4.1.2 Robot Arm Configuration and Technical Specifications

In this project, a panda robot will be applied in the simulation to keep consistency with the real robotic arm at the School of Informatics, shown in Figure.27. An exact same panda robot has been applied in Gazebo shown in Figure.26.

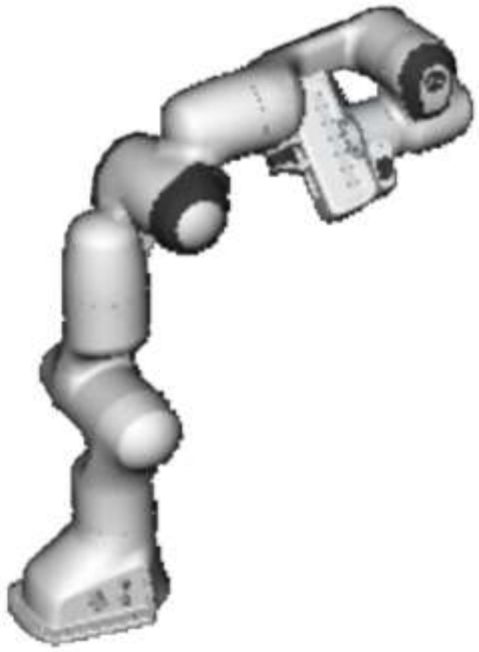


Figure 25 - Panda Robot Arm in Gazebo

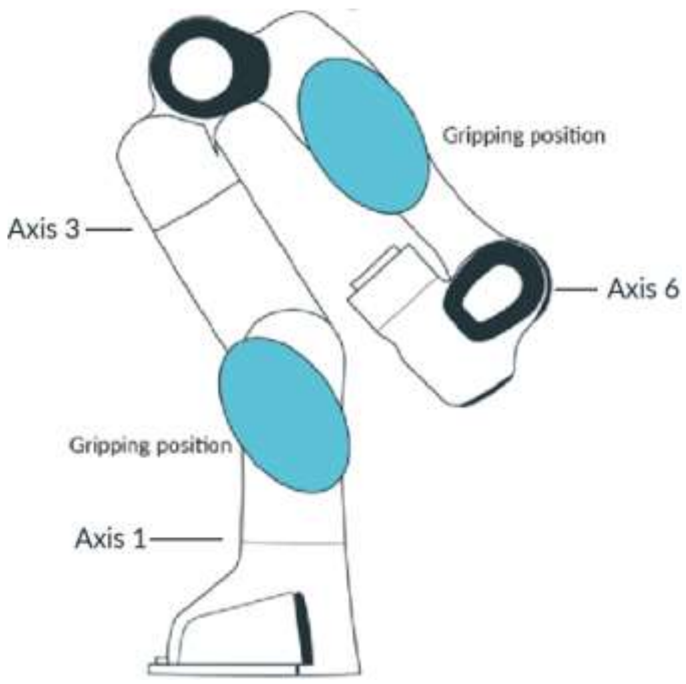


Figure 26 - Schematic View of Panda Robot

However, before designing the trajectory, one thing that has been done is to check out the technical specifications to understand the parameters such as the number of DoF, payload, joint position limits, joint velocity limits, interaction, self-collision range, weight, etc.

Specifically, the technical parameters mentioned above have been shown in table one below [29].

Meanwhile, Figure 28 and Figure 29 below have shown the maximum operating distances of this panda robot.

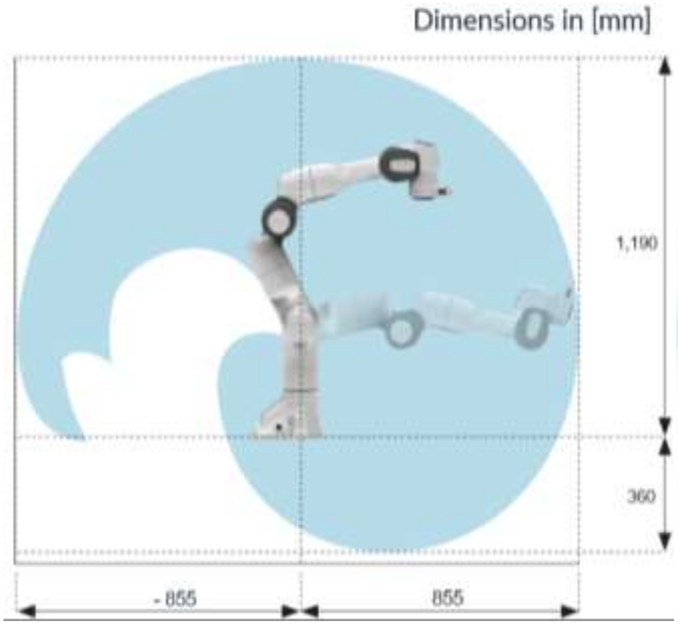


Figure 27 - Front view of the operating range [29];

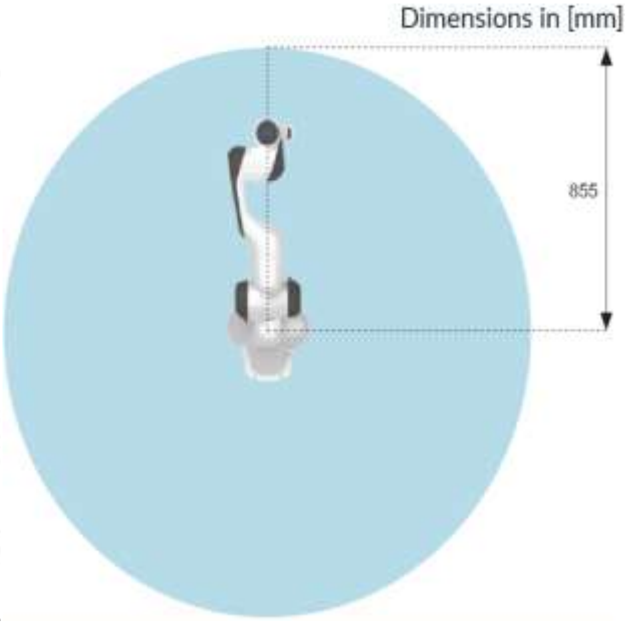


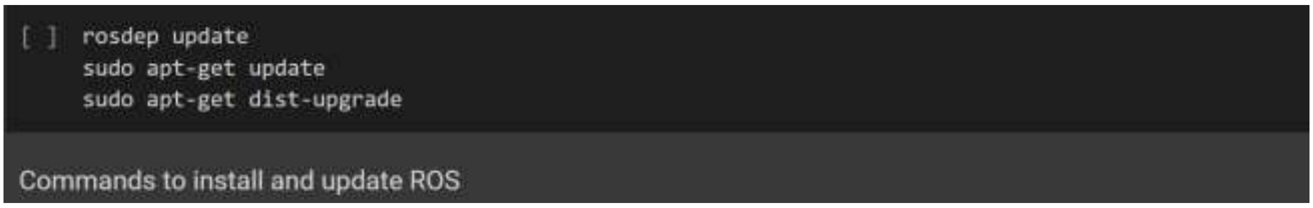
Figure 28 - Top view of the operating range [29]

Table 1 - Technical Specifications of Panda Robot Arm [29]

ROBOT ARM	**Degree of Freedom**	**7 DoF**
	Payload	**3 Kg**
	Sensitivity	**Torque sensors in all 7 axes**
	Maximum Reach	**855 mm**
	Joint Position Limits [°]	**A1: -166 ~ 166 A2: -101 ~ 101 A3: -166 ~ 166 A4: -176 ~ -4 A5: -166 ~ 166 A6: -1 ~ 215 A7: -166 ~ 166**
	Joint Velocity Limits [°/s]	**A1: 150 A2: 150 A3: 150 A4: 150 A5: 180 A6: 180 A7: 180**
	Weight	**~ 18 Kg**
CONTROL	**Weight**	**~ 7 Kg**
	Power Consumption	**Max ~ 600 W Average ~ 300 W**
	Ambient Temperature	**+15°C to 25°C (typical) +5°C to + 45°C (extended)**
GRASPER HAND	**Parallel Gripper**	**Exchangeable Fingers**
	Grasping Force	**Continuous force: 70 N Maximum force: 140 N**
	Travel Speed	**80 mm (50 mm/s per finger)**
	Weight	**~ 0.7 Kg**
DESK	**Platform**	**Via modern web browser**
	Programming	**Visual & intuitive, dialog-based**

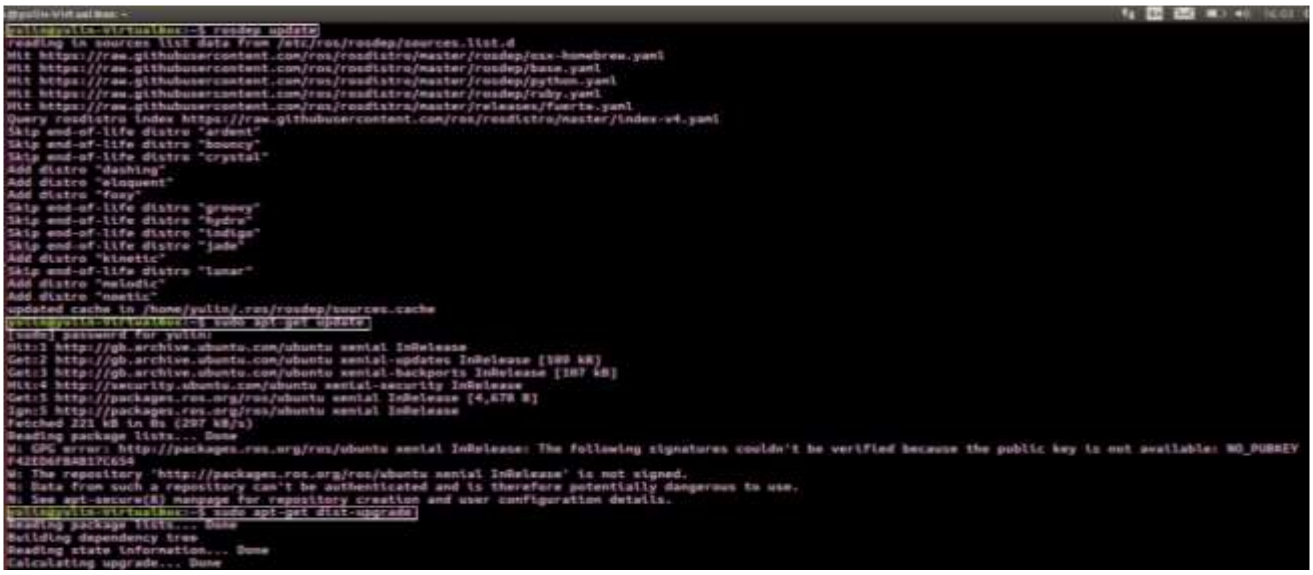
4.2 ROS KINETIC INSTALLATION, CATKIN WORKSPACE CREATE AND SETUP

After understanding the basic structures and configurations of the panda robotic arm, the next step is computer simulations. First and foremost, a ROS system should be installed before simulation. ROS is an open-source, meta-operating robot-oriented system that can provide tools, libraries for obtaining, building, writing and running code across multiple computers [30]. In order to use ROS system, the first step is to install and update the ROS to the latest version using terminal in Ubuntu (Linux) system using the following three-line commands, as shown in Figure.30.



```
[ ]  rosdep update
     sudo apt-get update
     sudo apt-get dist-upgrade
```

Commands to install and update ROS



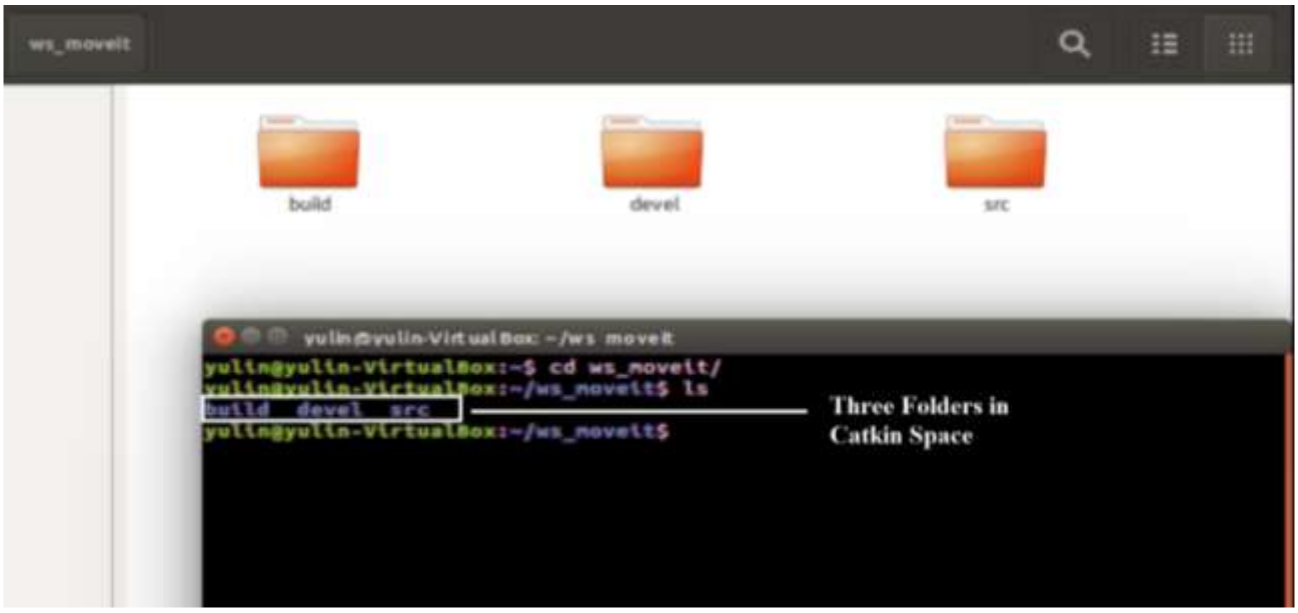
```
yulin@yulin-VirtualBox:~$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/osx-homebrew.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/base.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/python.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/ruby.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/releases/fuerte.yaml
Query rosdistro index https://raw.githubusercontent.com/ros/rosdistro/master/index-v4.yaml
Skip end-of-life distro "ardent"
Skip end-of-life distro "bouncy"
Skip end-of-life distro "crystal"
Add distro "dashing"
Add distro "eloquent"
Add distro "foxy"
Skip end-of-life distro "groovy"
Skip end-of-life distro "hydro"
Skip end-of-life distro "indigo"
Skip end-of-life distro "jade"
Add distro "kinetic"
Skip end-of-life distro "lunar"
Add distro "melodic"
Add distro "noetic"
updated cache in /home/yulin/.ros/rosdep/sources.cache
yulin@yulin-VirtualBox:~$ sudo apt-get update
[sudo] password for yulin:
Hit:1 http://gb.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://gb.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:3 http://gb.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Hit:4 http://security.ubuntu.com/ubuntu xenial-security InRelease
Get:5 http://packages.ros.org/ros/ubuntu xenial InRelease [4,678 B]
Ign:5 http://packages.ros.org/ros/ubuntu xenial InRelease
Fetched 221 kB in 0s (297 kB/s)
Reading package lists... Done
W: GPG error: http://packages.ros.org/ros/ubuntu xenial InRelease: The following signatures couldn't be verified because the public key is not available: NO_PUBKEY
F42ED6FBAB17C654
W: The repository 'http://packages.ros.org/ros/ubuntu xenial InRelease' is not signed.
N: Data from such a repository can't be authenticated and is therefore potentially dangerous to use.
N: See apt-secure(8) manpage for repository creation and user configuration details.
yulin@yulin-VirtualBox:~$ sudo apt-get dist-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
```

Figure 29 - ROS Update Commands - see three lines circled

After updating the ROS system to the latest version, the next step is to create a Catkin Workspace because the robot can only be simulated in the virtual workspace. Catkin workspace is a folder where catkin packages (some packages used for robotic controlling) can be built and applied to control the robotic arm [31]. Generally, a Catkin workspace contains three folders, which are 'build', 'development (devel)' and 'source (src)' separately, as shown in Figure.31.

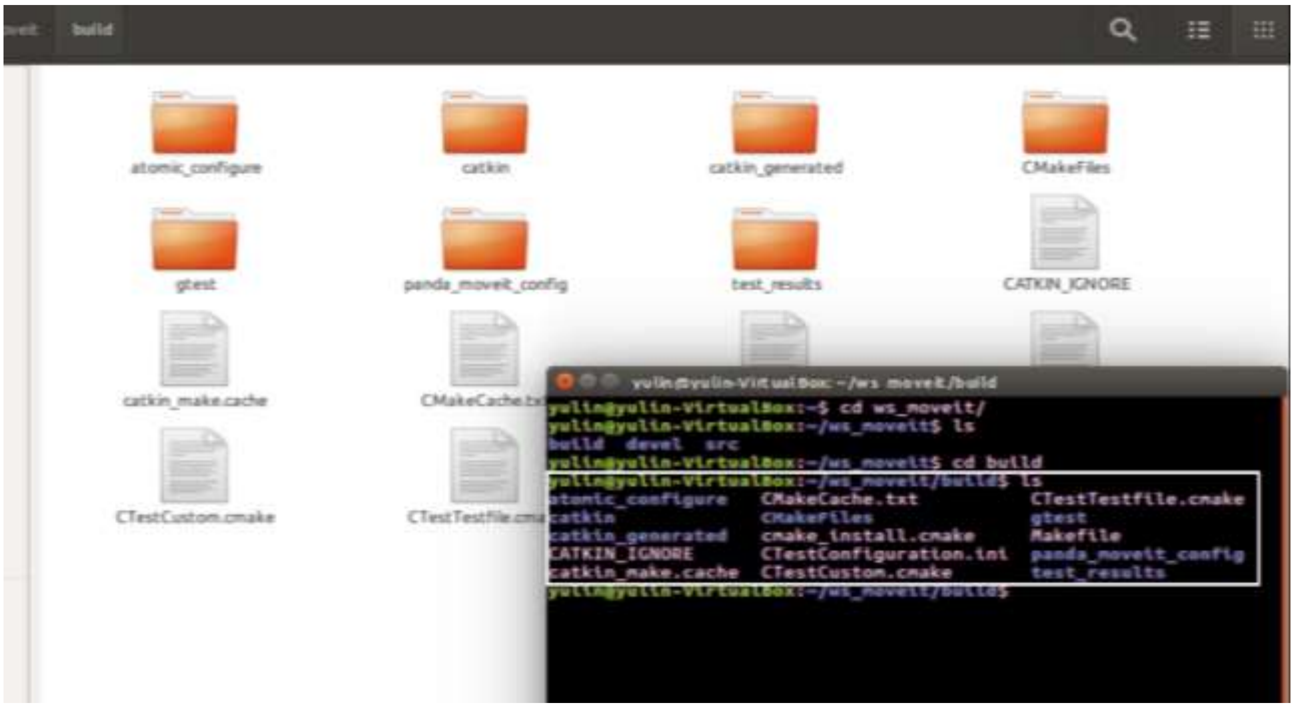
In terms of 'build' space (Figure.32), it is the major place where CMake is invoked to build the Catkin packages, namely, the 'build' folder is like a factory where a number of useful packages will be fabricated for further use. The 'development (devel)' space (Figure.33) is where built targets are placed prior to being installed. The way targets are organized in the devel space is the same as their layout when they are installed. This provides a useful testing and development environment which does not require invoking the installation step [31].

Finally, shown in Figure.34, the last folder is source space (src), which contains the source code of Catkin packages. Designers can extract or clone source codes from this workspace, and each folder within the source space contains one or more Catkin space. One thing that should be noted is that all this folder should always remain unchanged during configuring, building and installing procedure.


ws_moveit
build
devel
src
yulin@yulin-VirtualBox: ~/ws_moveit
Three Folders in Catkin Space

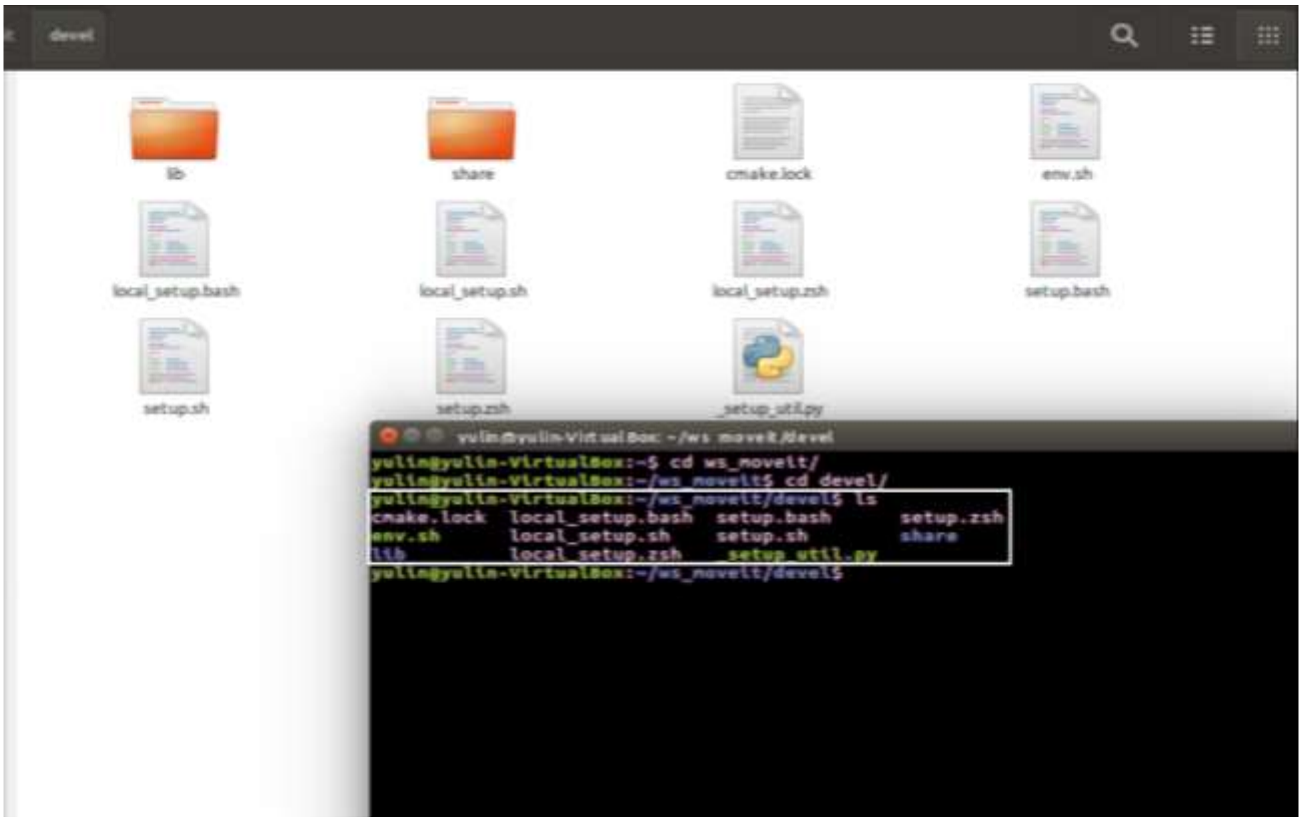
```
yulin@yulin-VirtualBox:~$ cd ws_moveit/
yulin@yulin-VirtualBox:~/ws_moveit$ ls
build  devel  src
yulin@yulin-VirtualBox:~/ws_moveit$
```

Figure 30 - Three important folders in Catkin Space



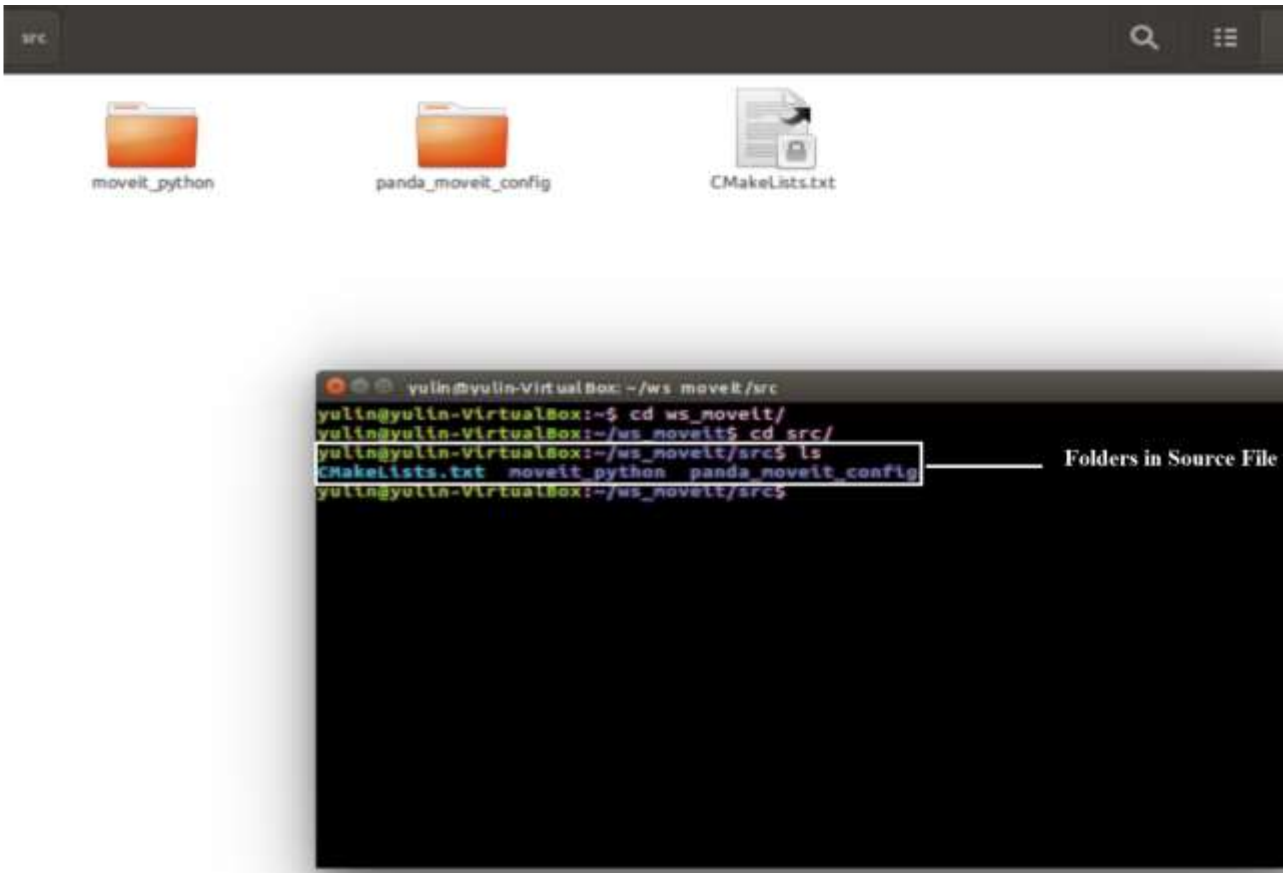
```
yulin@yulin-VirtualBox:~$ cd ws_moveit/
yulin@yulin-VirtualBox:~/ws_moveit$ ls
build  devel  src
yulin@yulin-VirtualBox:~/ws_moveit$ cd build
yulin@yulin-VirtualBox:~/ws_moveit/build$ ls
atomic_configure   CMakeCache.txt            CTestTestfile.cmake
catkin             CMakeFiles                gtest
catkin_generated   cmake_install.cmake       Makefile
CATKIN_IGNORE      CTestConfiguration.ini    panda_moveit_config
catkin_make.cache  CTestCustom.cmake         test_results
yulin@yulin-VirtualBox:~/ws_moveit/build$
```

Figure 31 - Folders in Build Space



```
yulin@yulin-VirtualBox:~$ cd ws_moveit/
yulin@yulin-VirtualBox:~/ws_moveit$ cd devel/
yulin@yulin-VirtualBox:~/ws_moveit/devel$ ls
cmake.lock  local_setup.bash  setup.bash      setup.zsh
env.sh      local_setup.sh    setup.sh        share
lib         local_setup.zsh   _setup_util.py
yulin@yulin-VirtualBox:~/ws_moveit/devel$
```

Figure 32 – Folders in Devel Space

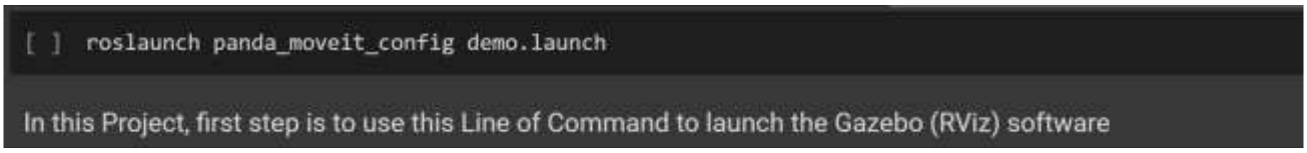


```
yulin@yulin-VirtualBox:~$ cd ws_moveit/
yulin@yulin-VirtualBox:~/ws_moveit$ cd src/
yulin@yulin-VirtualBox:~/ws_moveit/src$ ls
CMakeLists.txt  moveit_python  panda_moveit_config
yulin@yulin-VirtualBox:~/ws_moveit/src$
```

Figure 33 - Folders in Source (src) space

4.3 RViz ROBOT CONFIGURATION SETUP

After the completion of ROS installation and Catkin Workspace construction, the next thing to do is to use RViz software to personalise the configuration of the robotic arm. RViz is the primary visualizer in ROS and a useful tool to personalise and debug your robot [32]. By launching the software RViz using the command in Figure.35, a panda robot arm and a few display settings can be seen (Figure.36).



```
[ ]  roslaunch panda_moveit_config demo.launch
```

In this Project, first step is to use this Line of Command to launch the Gazebo (RViz) software

Figure 34 - using command line in Ubuntu Shell to launch RViz

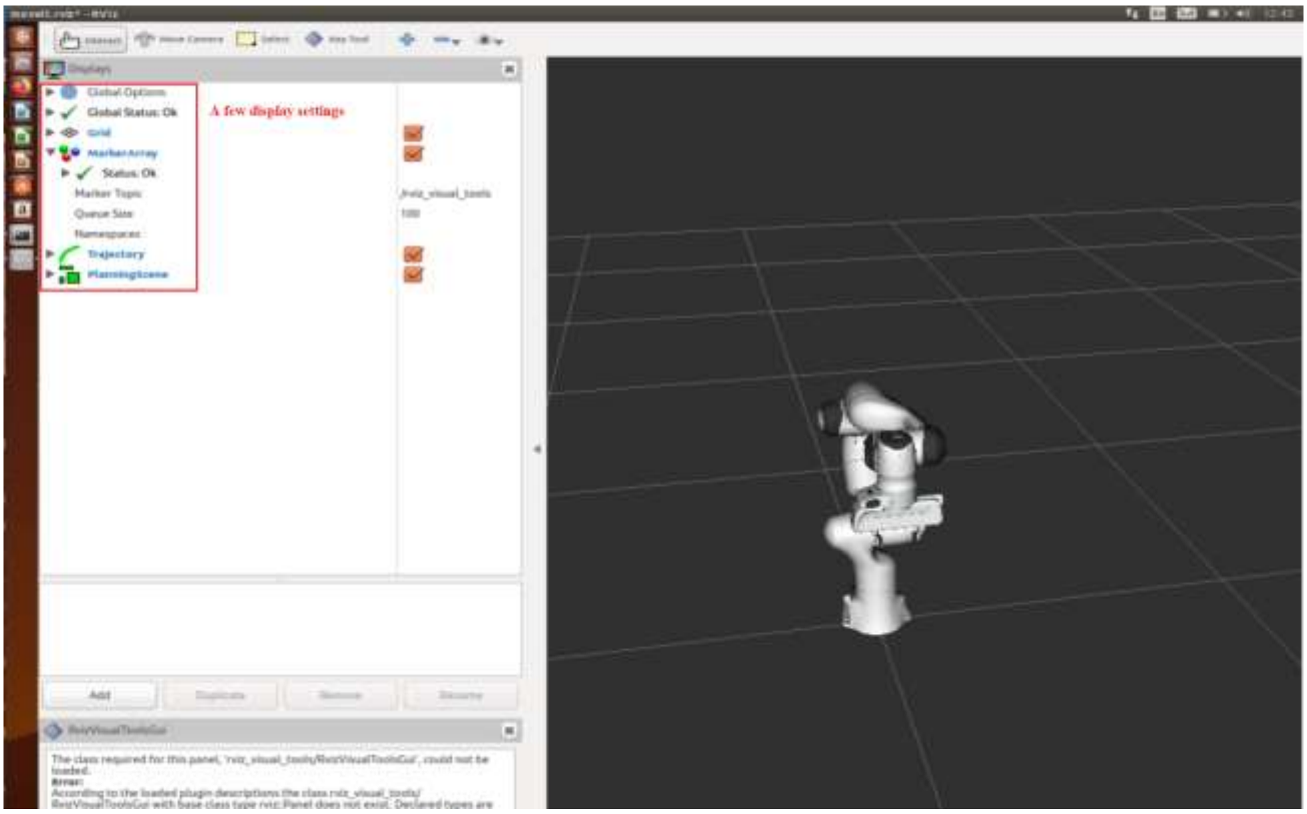
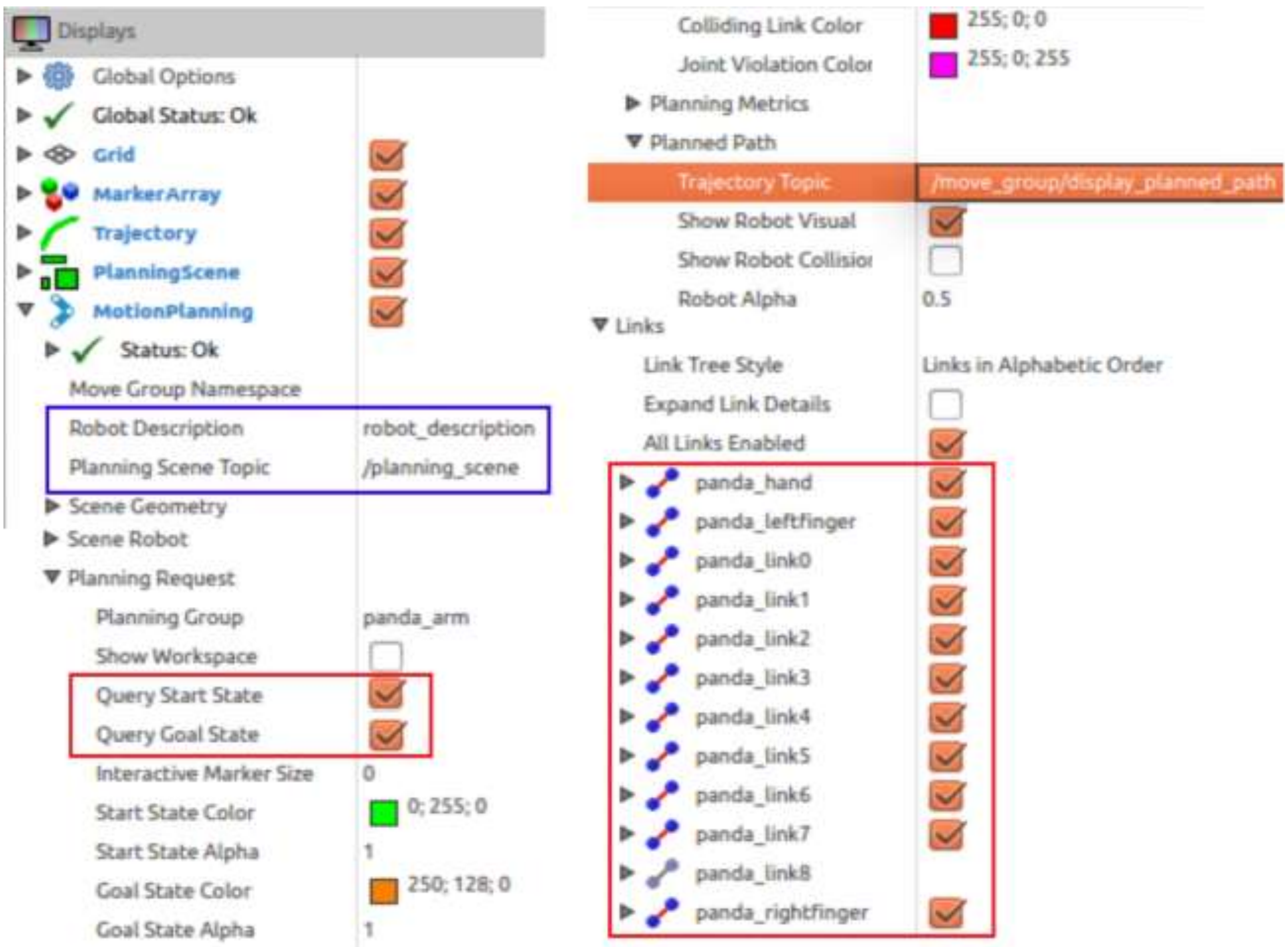


Figure 35 - RViz Robot Arm setting interface including panda robot arm and relevant display settings

To manually interact with the robot, a few settings are required in advance. As shown in Figure.37, the robot description and the planning scene are set automatically as default. Under the Planning Request tab, in order to visualise both the starting point and endpoint, the Query Start State and Query Goal State boxes should be checked. Further to this, under the ‘Links’ tab, it can be seen that there are a few links including panda_hand, panda_leftfinger, panda_link0, panda_link1, panda_rightfinger, etc. Each link connects each joint, namely, each degree of freedom of the robotic arm corresponds to one link. The pre-setting of those links will be explained below.



Displays	
Global Options	
Global Status: Ok	
Grid	✓
MarkerArray	✓
Trajectory	✓
PlanningScene	✓
MotionPlanning	✓
Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	/planning_scene
Scene Geometry	
Scene Robot	
Planning Request	
Planning Group	panda_arm
Show Workspace	☐
Query Start State	✓
Query Goal State	✓
Interactive Marker Size	0
Start State Color	🟩 0; 255; 0
Start State Alpha	1
Goal State Color	🟧 250; 128; 0
Goal State Alpha	1

Colliding Link Color	🟥 255; 0; 0
Joint Violation Color	🟪 255; 0; 255
Planning Metrics	
Planned Path	
Trajectory Topic	/move_group/display_planned_path
Show Robot Visual	✓
Show Robot Collision	☐
Robot Alpha	0.5
Links	
Link Tree Style	Links in Alphabetic Order
Expand Link Details	☐
All Links Enabled	✓
panda_hand	✓
panda_leftfinger	✓
panda_link0	✓
panda_link1	✓
panda_link2	✓
panda_link3	✓
panda_link4	✓
panda_link5	✓
panda_link6	✓
panda_link7	✓
panda_link8	
panda_rightfinger	✓

Figure 36 - RViz Display Settings

As mentioned above, each DoF corresponds to a link, and to help readers understand what each link represents, several figures have been attached on the next page (Figure.38).

According to the technical specification table, the panda robot has seven DoFs, which in Figure.38 corresponds to panda_link1 to panda_link8, and each DoF is located on the robot arm has been shown as well. However, one thing to be noted here is that panda_link8 is the same as panda_hand, which is why there seems to be one extra link shown in Figure.38 but in reality, it still has seven DoFs rather than eight.

Moreover, in this pre-setting, a gripper hand has been used in replace of the 3D printer head as there is no 3D printer head available. It can be observed that the gripper itself also adds two extra DoFs to the entire system, which is panda_leftfinger and panda_rightfinger, respectively. However, in the project, since a desktop 3D printer head will be mounted, there should be no additional DoFs as obviously, the printer head cannot move. In this case, in order to keep consistent with the real condition, in the simulation, those two extra DoFs should be defunctionalized.

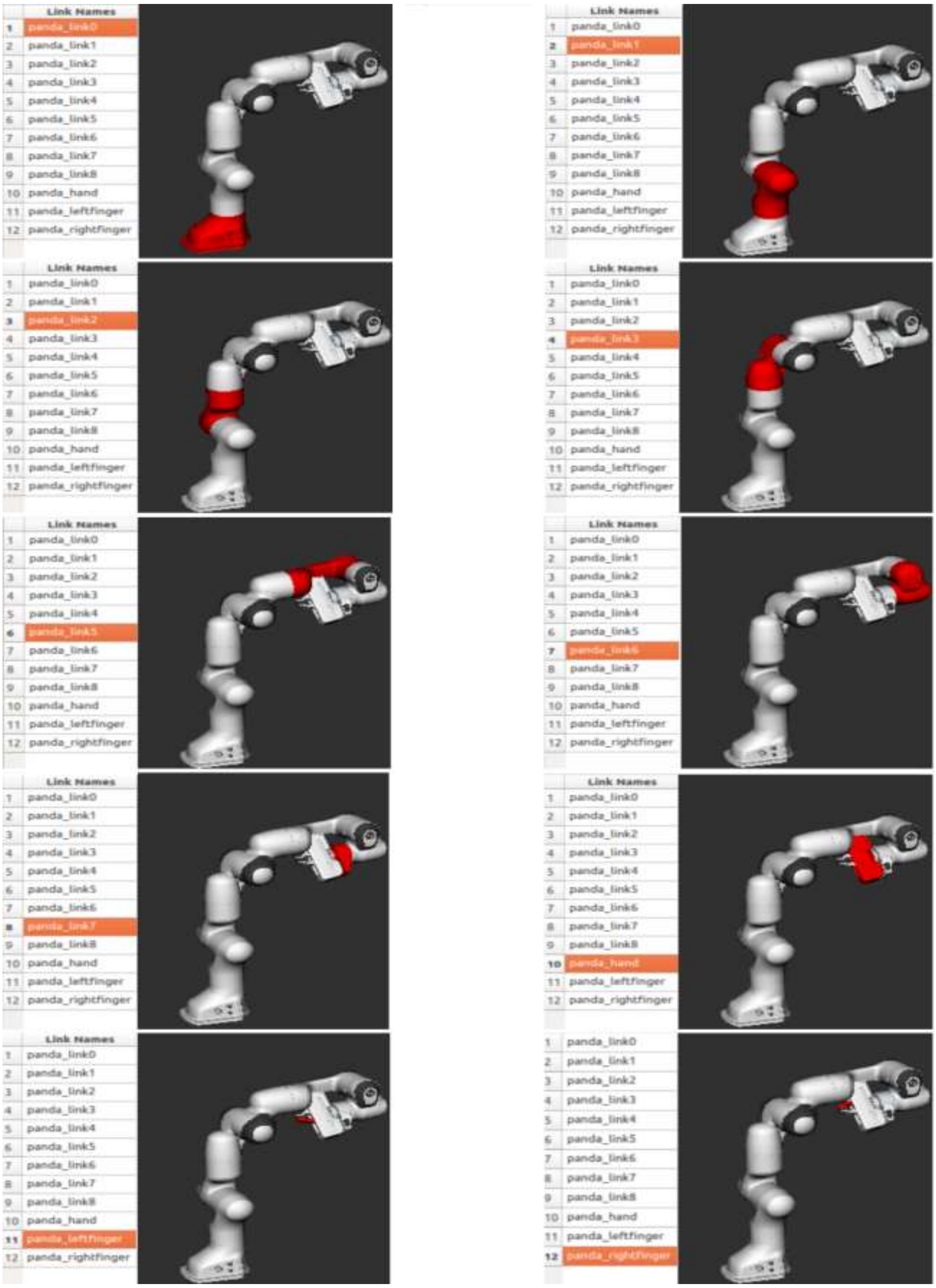


Figure 37 - Different Links & Different DoFs

4.4 ROS DATA FLOW OF CONTROLLERS

In the previous two parts, some basic installations and configurations setting up have already been introduced. In this part, a brief introduction of how the data flows over the entire controller will be given. As the complete and detailed data flow over the controller is way too complicated, it is not possible and necessary to explain every single definition of this part under the limited time of this project, therefore, a simple flowchart below (Figure.39) shows a concise description of data transmission from controller manager and controller to hardware interface (which is not available in our project) and further to the real robot (not available in the project either).

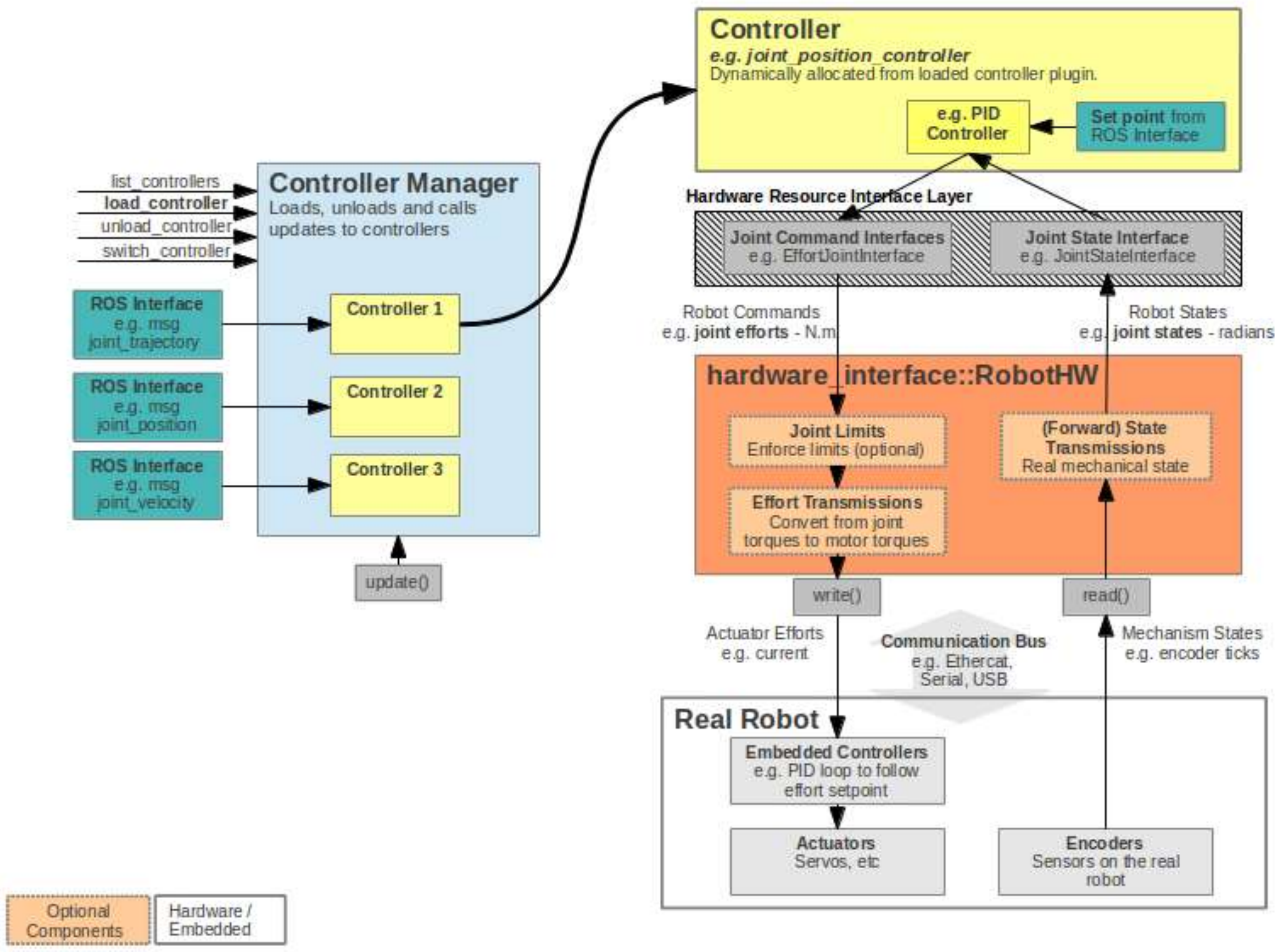

Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded

Figure 38 - ROS Control Data Flow [33]

As shown above, a ROS control package includes controller interfaces (ROS interface in the figure), controller managers, transmissions and hardware interfaces. Its working principle is to take the data extrapolated from robot's actuator's encoder of desired state (the position that the robot will move to in the next step) as inputs, then it uses a generic control loop feedback mechanism, typically PID controller to control the outputs and send them to actuators to control the robotic arm by moving each joint and links so that the robot can move towards the desired position.

4.5 MANUAL INTERACTION WITH THE PANDA

After setting up the configurations of panda robot, the next step is to interact with panda to have a taste of the motion planning. As shown in Figure.40, the initial status of the robot (green) and the final status of the robot (orange) have been defined manually. One thing worth noting is that collision should be avoided when setting up the initial and final positions of the robotic arm and to do this, the 'Collision-Aware IK" box should be checked, as shown in Figure.41, and Figure.42 shows the robot collision situation.

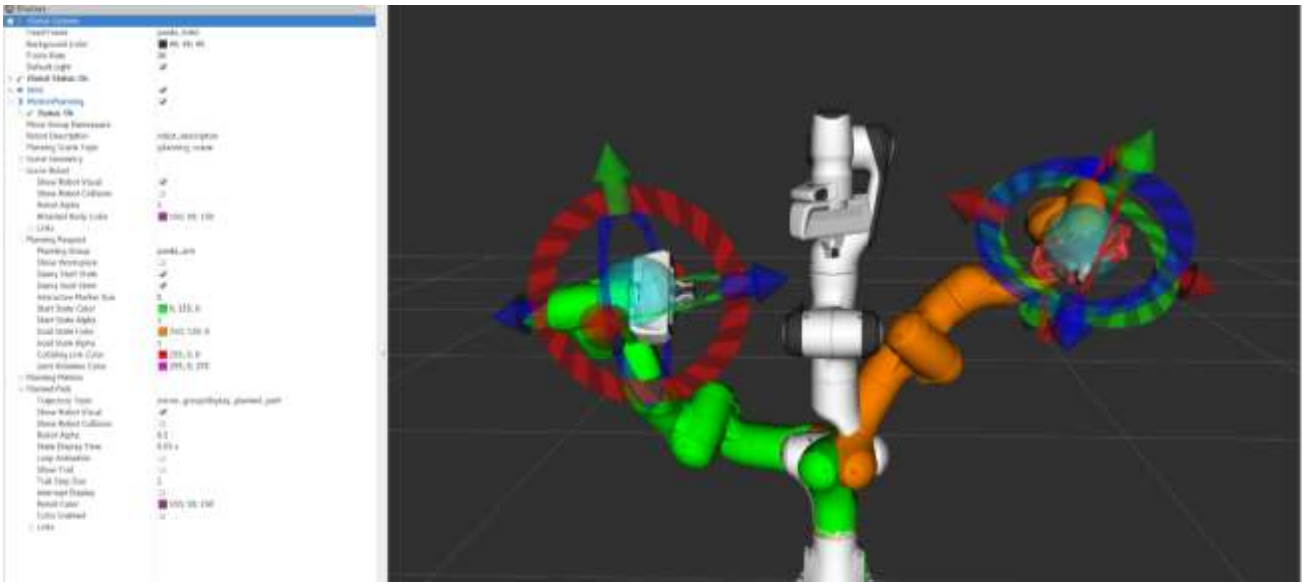
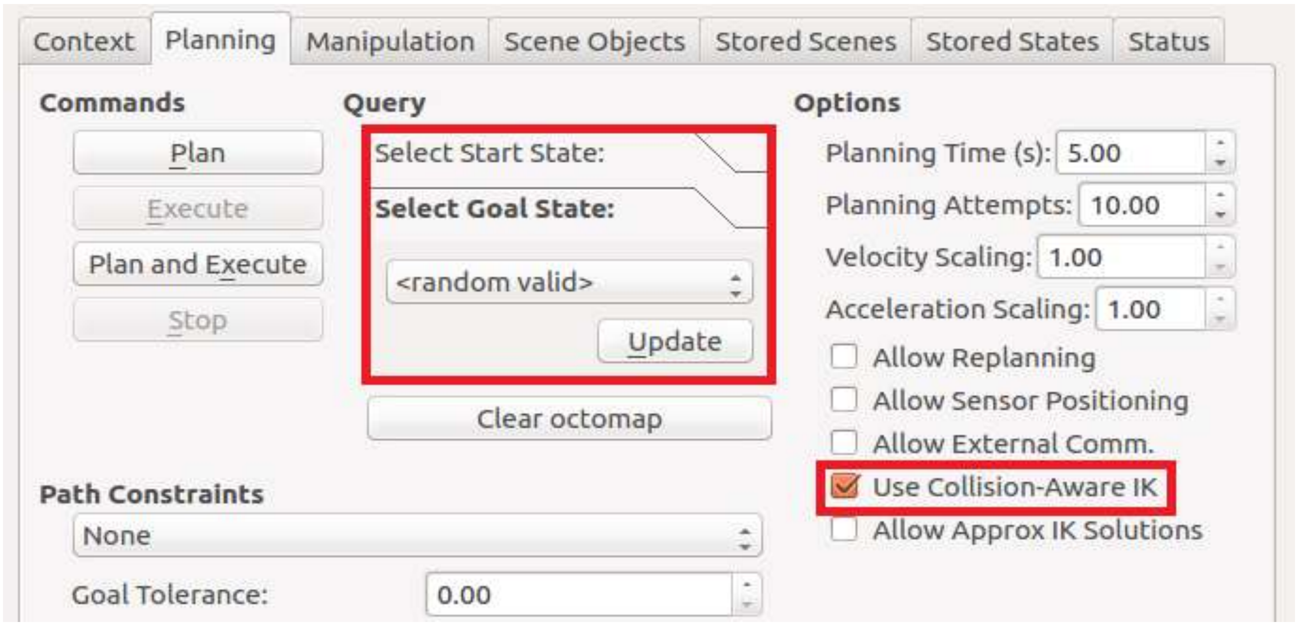


Figure 39 - Manually define the initial state (green) and final state (orange)



Select Start State:
Select Goal State: <random valid>
Planning Time (s): 5.00
Planning Attempts: 10.00
Velocity Scaling: 1.00
Acceleration Scaling: 1.00
☐ Allow Replanning
☐ Allow Sensor Positioning
☐ Allow External Comm.
☑ Use Collision-Aware IK
☐ Allow Approx IK Solutions
Path Constraints: None
Goal Tolerance: 0.00

Figure 40 - Initial and Final Position Setup

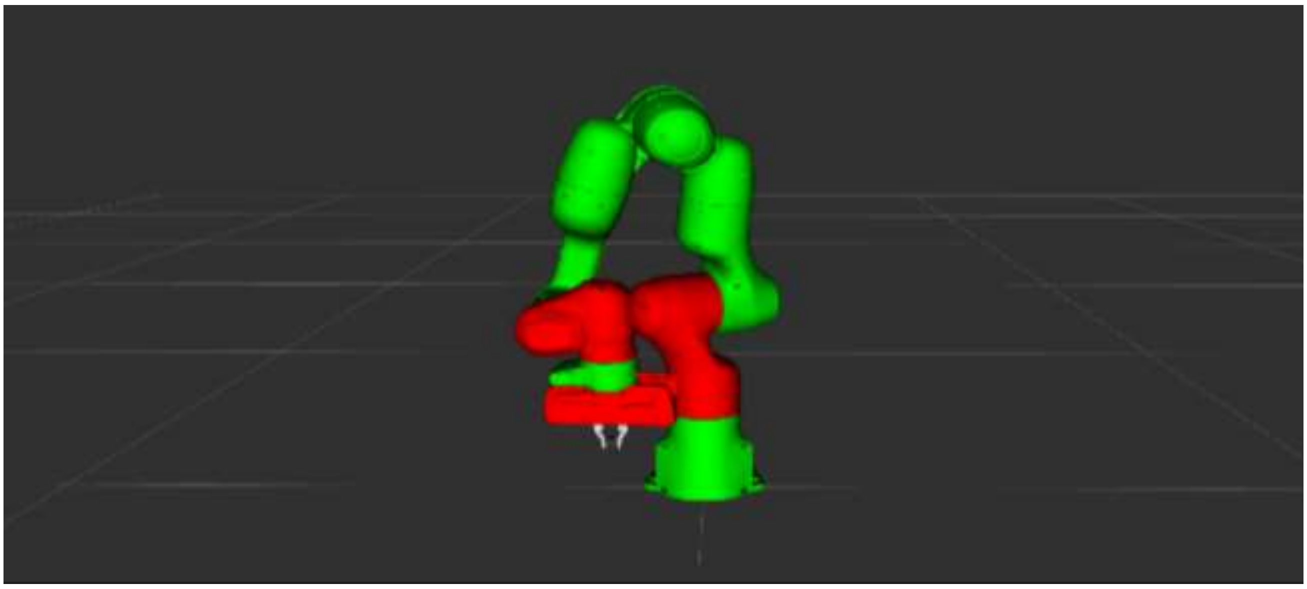


Figure 41 - Robot Arm Self-Collision (Red Part)

One significant advantage of using Gazebo is that the Inverse Kinematics algorithms have been packaged and embedded inside of this software already. On this occasion, users only have to define the initial and final coordinates, and the Gazebo itself will automatically use Inverse Kinematics algorithms to make calculations and compute an optimised trajectory to control each DoF move accordingly. In order to have a more illustrative view of how the Gazebo plans the path and how the trajectory looks like, two simulations have been conducted in Figure.42 under the condition of setting different initial positions, final positions and state display time. It can be seen that the path will be automatically generated by Gazebo without human intervention, and the state display time can be tuned by users to mimic the real situation.

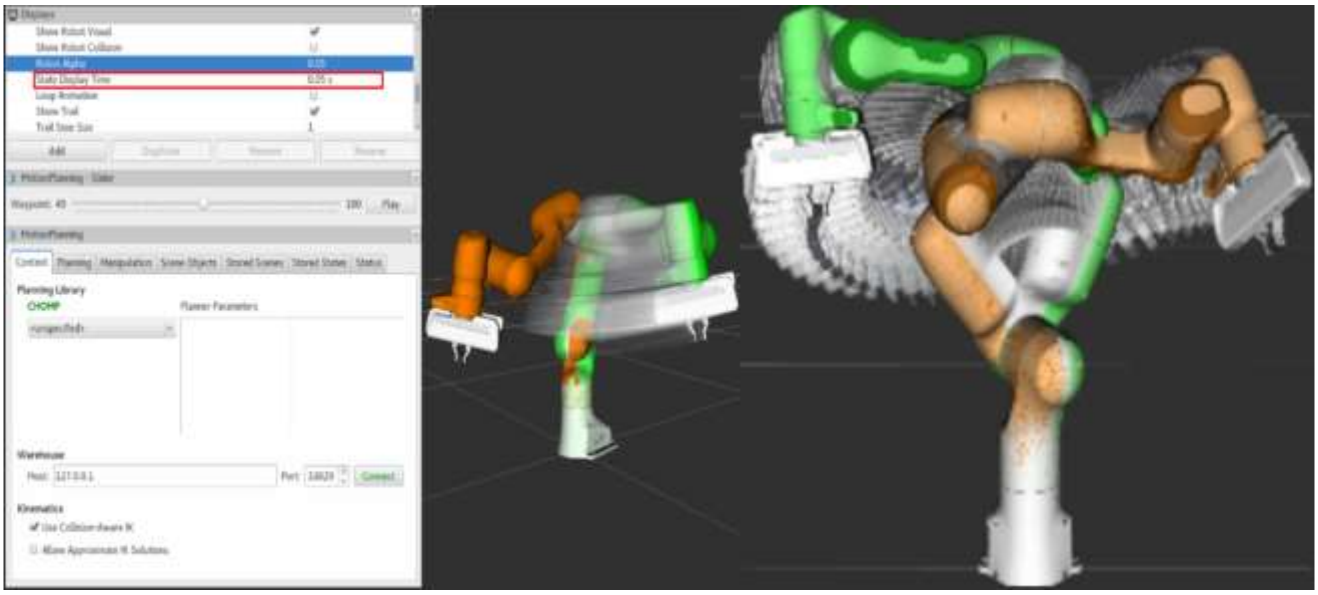


Figure 42 - Left: Trajectory Simulation with higher movement speed; Right: Trajectory simulation with lower movement speed

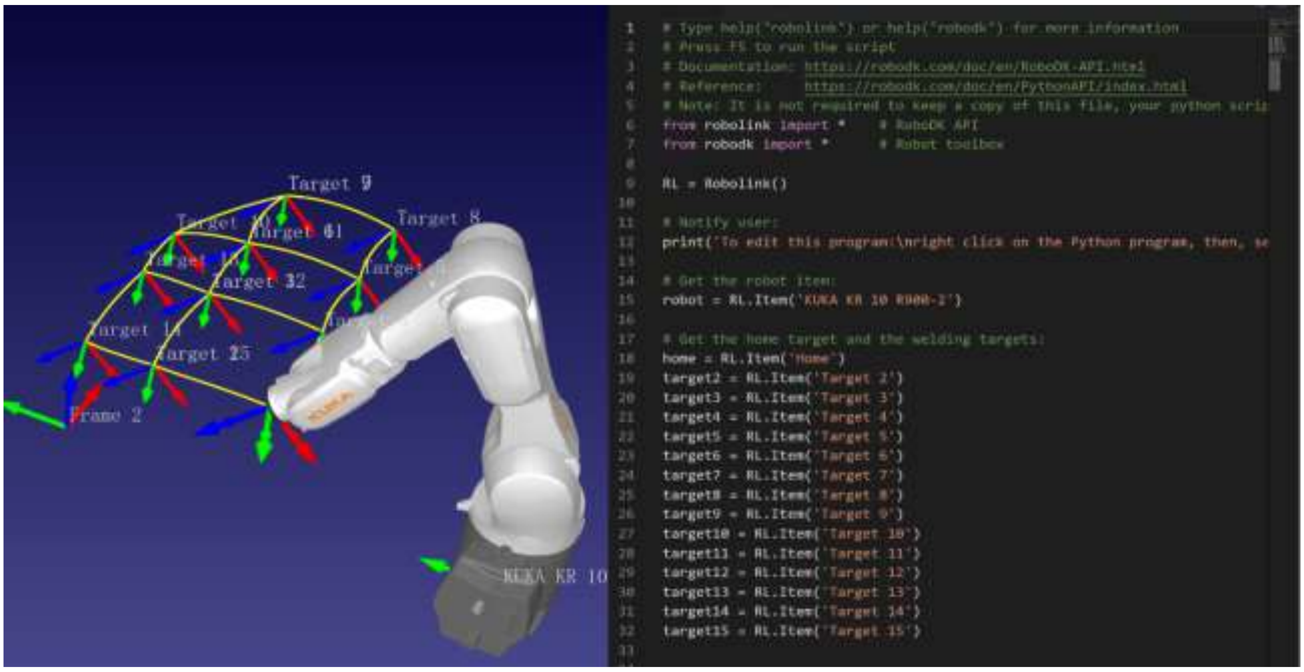
5. CHAPTER 5: PYTHON-ROBODK & PYTHON-GAZEBO INTERFACE

According to the previous section, the configuration of Gazebo has been explained in detail, along with which the manual control has also been introduced. However, as the manual control method is still considered to be primitive and cannot fully obtain automation, therefore, to overcome this barrier and make the controlling procedure fully automated, Python scripts have been applied in this chapter to control the robot trajectory.

5.1 PATH PLANNING IN PYTHON-ROBODK INTERFACE

Initially, RoboDK was implemented due to its simplicity of using and a highly compact GUI. However, one thing that was later noticed is that in RoboDK robotic arm library, there is no panda robot available, therefore, a KUKA robotic arm with seven DoFs has been selected as an alternative as it presents a wide range of similarities as the panda robot.

In order to visualise the trajectory of the robot, Python-RoboDK interfaced was applied, namely, a Python script will be run in RoboDK simulate the path, as shown in Figure.43.



```
# Type help("robolink") or help("robodk") for more information
# Press F5 to run the script
# Documentation: https://robodk.com/doc/en/RoboDK-API.html
# Reference:     https://robodk.com/doc/en/PythonAPI/index.html
# Note: It is not required to keep a copy of this file, your python scri
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox

RL = Robolink()

# Notify user:
print('To edit this program:\nright click on the Python program, then, se

# Get the robot item:
robot = RL.Item('KUKA KR 10 R900-2')

# Get the home target and the welding targets:
home = RL.Item('Home')
target2 = RL.Item('Target 2')
target3 = RL.Item('Target 3')
target4 = RL.Item('Target 4')
target5 = RL.Item('Target 5')
target6 = RL.Item('Target 6')
target7 = RL.Item('Target 7')
target8 = RL.Item('Target 8')
target9 = RL.Item('Target 9')
target10 = RL.Item('Target 10')
target11 = RL.Item('Target 11')
target12 = RL.Item('Target 12')
target13 = RL.Item('Target 13')
target14 = RL.Item('Target 14')
target15 = RL.Item('Target 15')
```

Figure 43 – Python-RoboDK Interface to visualise and simulate the trajectory

In terms of controlling and making simulations in RoboDK, there are typically four steps including robot selecting, end effectors defining, robot programs (Python) writing and. Figure.44 has shown the flowchart of using RoboDK.

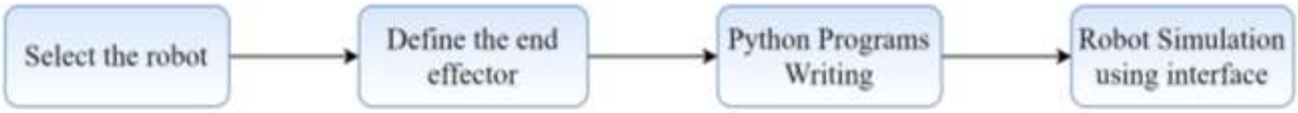
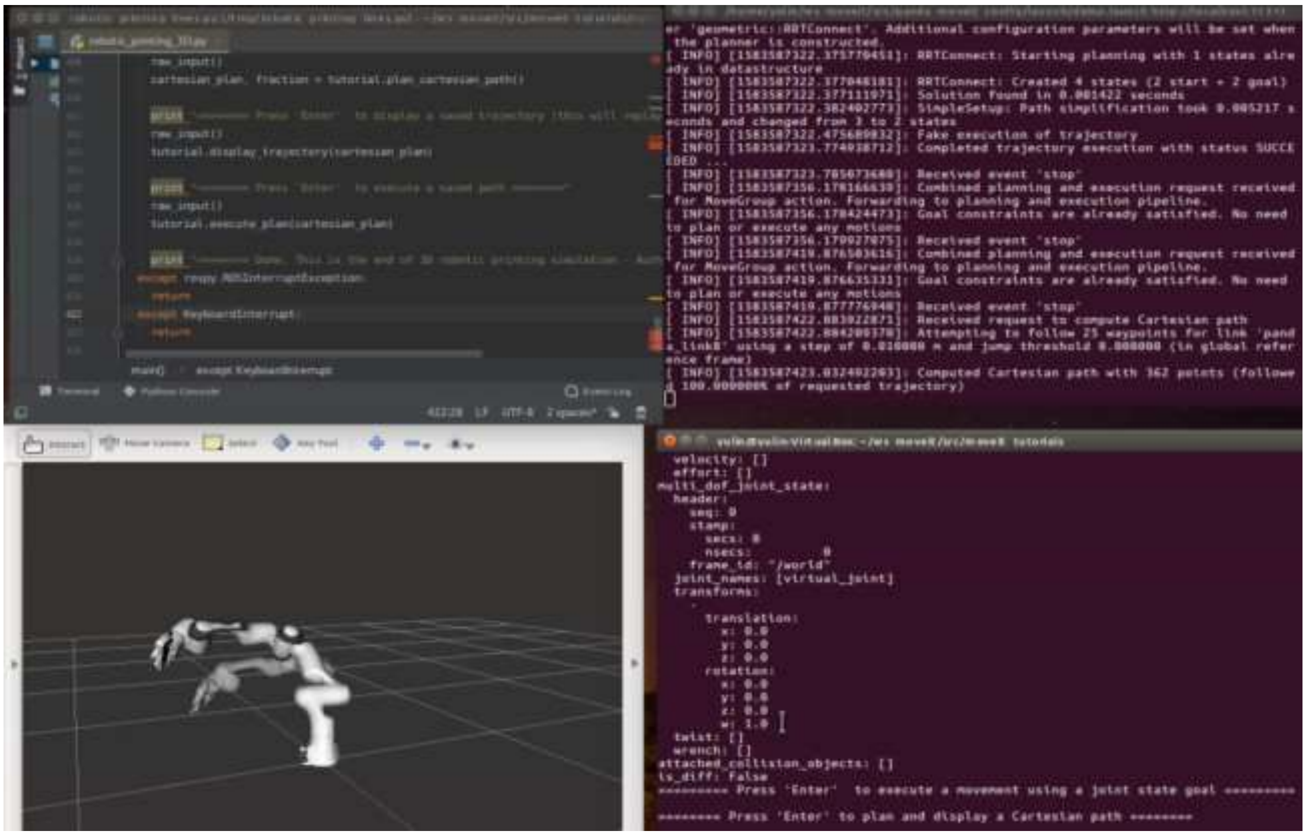


Figure 44 - Flowchart of using RoboDK

5.2 PATH PLANNING IN PYTHON-GAZEBO INTERFACE

Despite a number of advantages of using the Python-RoboDK interface, there are also quite a few drawbacks such as the insufficiency of robot options in the library (no panda robot available), limited time of free trial (seven days) and the extremely high price of the full version ($3,495 for eternity). Therefore, after balancing the benefits and drawbacks, RoboDK was eventually not considered as the best software for simulation. Later, as recommended by a professor at the School of Informatics, The University of Edinburgh, Python-Gazebo interface is used to simulate the trajectory. In the following section, the Python script will be elaborated to help readers understand the programming idea. First and foremost, recall the original project target and intention, a robot arm was used aiming to extend the operational range of conventional 3D printer, therefore, in order to achieve this target, two simple cases have been conducted in this project, one of which is in 2D surface and the other one is to print a 3D geometry, specifically, a pyramid shape. As shown in Figure.45, 46, 47, 48, Python script was applied to control the robot arm to print a 3D pyramid geometry.

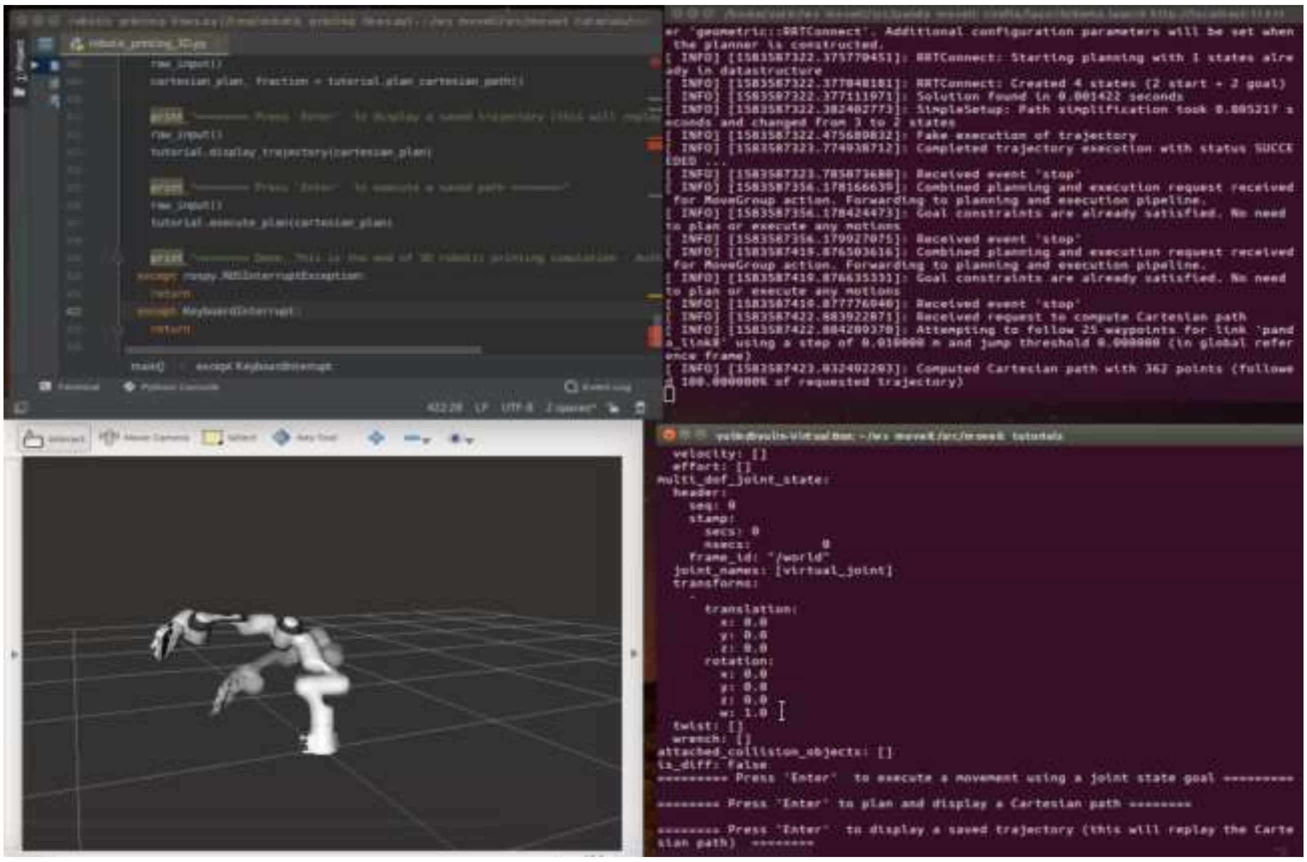


```
        raw_input()
        cartesian_plan, fraction = tutorial.plan_cartesian_path()

        print "========== Press `Enter` to display a saved trajectory (this will replay
        raw_input()
        tutorial.display_trajectory(cartesian_plan)

        print "========== Press `Enter` to execute a saved path =========="
        raw_input()
        tutorial.execute_plan(cartesian_plan)

        print "========== Done. This is the end of 3D robotic printing simulation - Auth
    except rospy.ROSInterruptException:
        return
    except KeyboardInterrupt:
        return
```

```
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583587322.375770451]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ INFO] [1583587322.377048181]: RRTConnect: Created 4 states (2 start + 2 goal)
[ INFO] [1583587322.377111971]: Solution found in 0.001422 seconds
[ INFO] [1583587322.382402773]: SimpleSetup: Path simplification took 0.005217 s
econds and changed from 3 to 2 states
[ INFO] [1583587322.475689832]: Fake execution of trajectory
[ INFO] [1583587323.774938712]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583587323.785673680]: Received event 'stop'
[ INFO] [1583587356.178166630]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587356.178424473]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587356.179927875]: Received event 'stop'
[ INFO] [1583587419.876503616]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587419.876635331]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587419.877776948]: Received event 'stop'
[ INFO] [1583587422.883922871]: Received request to compute Cartesian path
[ INFO] [1583587422.884200378]: Attempting to follow 25 waypoints for link 'pand
a_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refer
ence frame)
[ INFO] [1583587423.832402203]: Computed Cartesian path with 362 points (followe
d 100.000000% of requested trajectory)
```

```
  velocity: []
  effort: []
multi_dof_joint_state:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: "/world"
  joint_names: [virtual_joint]
  transforms:
    -
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  twist: []
  wrench: []
attached_collision_objects: []
is_diff: False
========== Press `Enter` to execute a movement using a joint state goal ==========

========= Press `Enter` to plan and display a Cartesian path =========
```

Figure 45 - Status one of Pyramid Geometry case study

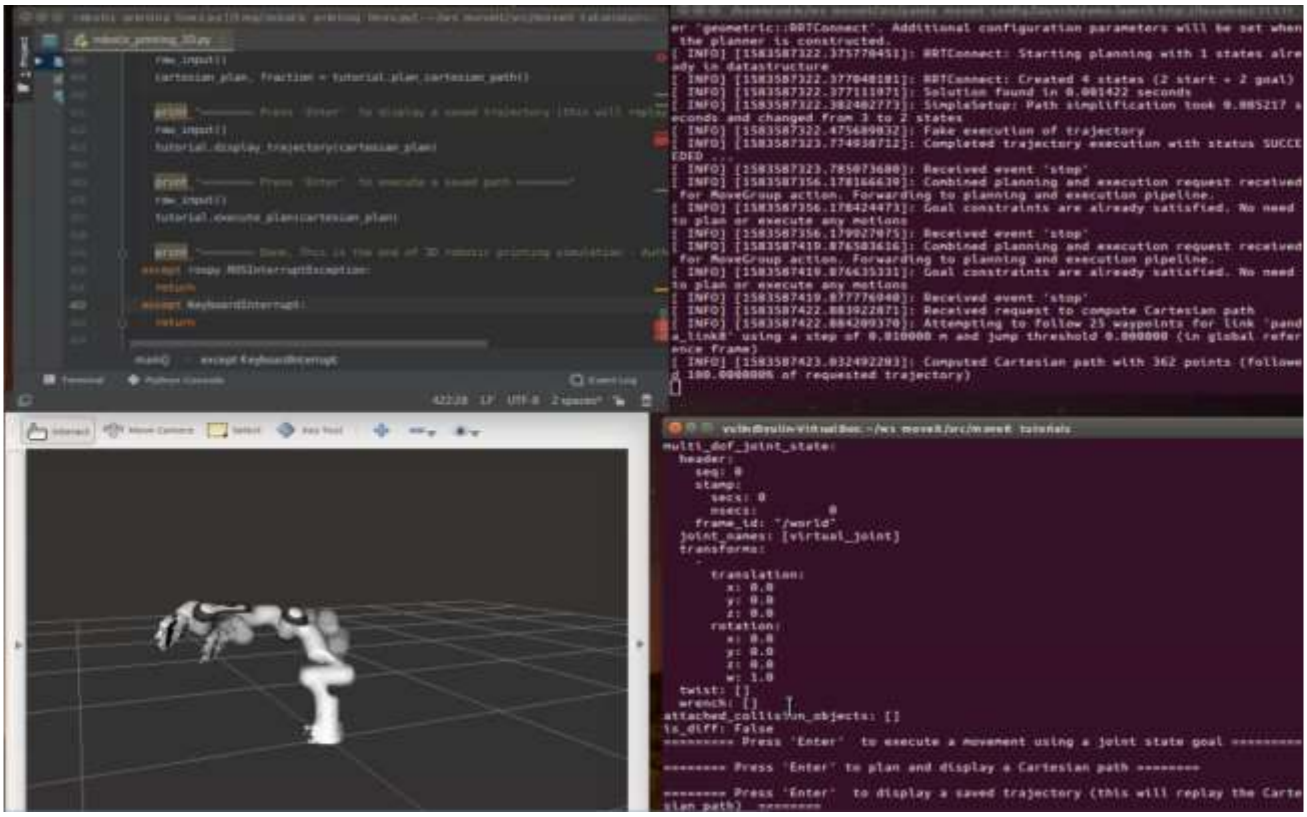


```
            raw_input()
            cartesian_plan, fraction = tutorial.plan_cartesian_path()

            print "========== Press `Enter` to display a saved trajectory (this will repla
            raw_input()
            tutorial.display_trajectory(cartesian_plan)

            print "========== Press `Enter` to execute a saved path =========="
            raw_input()
            tutorial.execute_plan(cartesian_plan)

            print "========== Done. This is the end of 3D robotic printing simulation - Auth
        except rospy.ROSInterruptException:
            return
        except KeyboardInterrupt:
            return

main() > except KeyboardInterrupt
```

```
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583587322.375770451]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ INFO] [1583587322.377048181]: RRTConnect: Created 4 states (2 start + 2 goal)
[ INFO] [1583587322.377111971]: Solution found in 0.001422 seconds
[ INFO] [1583587322.382402773]: SimpleSetup: Path simplification took 0.005217 s
econds and changed from 3 to 2 states
[ INFO] [1583587322.475689832]: Fake execution of trajectory
[ INFO] [1583587323.774938712]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583587323.785073680]: Received event 'stop'
[ INFO] [1583587356.178166639]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587356.178424473]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587356.179927075]: Received event 'stop'
[ INFO] [1583587419.876503616]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587419.876635331]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587419.877776940]: Received event 'stop'
[ INFO] [1583587422.883922871]: Received request to compute Cartesian path
[ INFO] [1583587422.884209370]: Attempting to follow 25 waypoints for link 'pand
a_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refer
ence frame)
[ INFO] [1583587423.032492203]: Computed Cartesian path with 362 points (followe
d 100.000000% of requested trajectory)
```

```
  velocity: []
  effort: []
multi_dof_joint_state:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: "/world"
  joint_names: [virtual_joint]
  transforms:
    -
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  twist: []
  wrench: []
attached_collision_objects: []
is_diff: False
========== Press 'Enter'  to execute a movement using a joint state goal ==========

========== Press 'Enter' to plan and display a Cartesian path ==========

========== Press 'Enter'  to display a saved trajectory (this will replay the Carte
sian path) ==========
```

Figure 46 - Status two of Pyramid Geometry case study

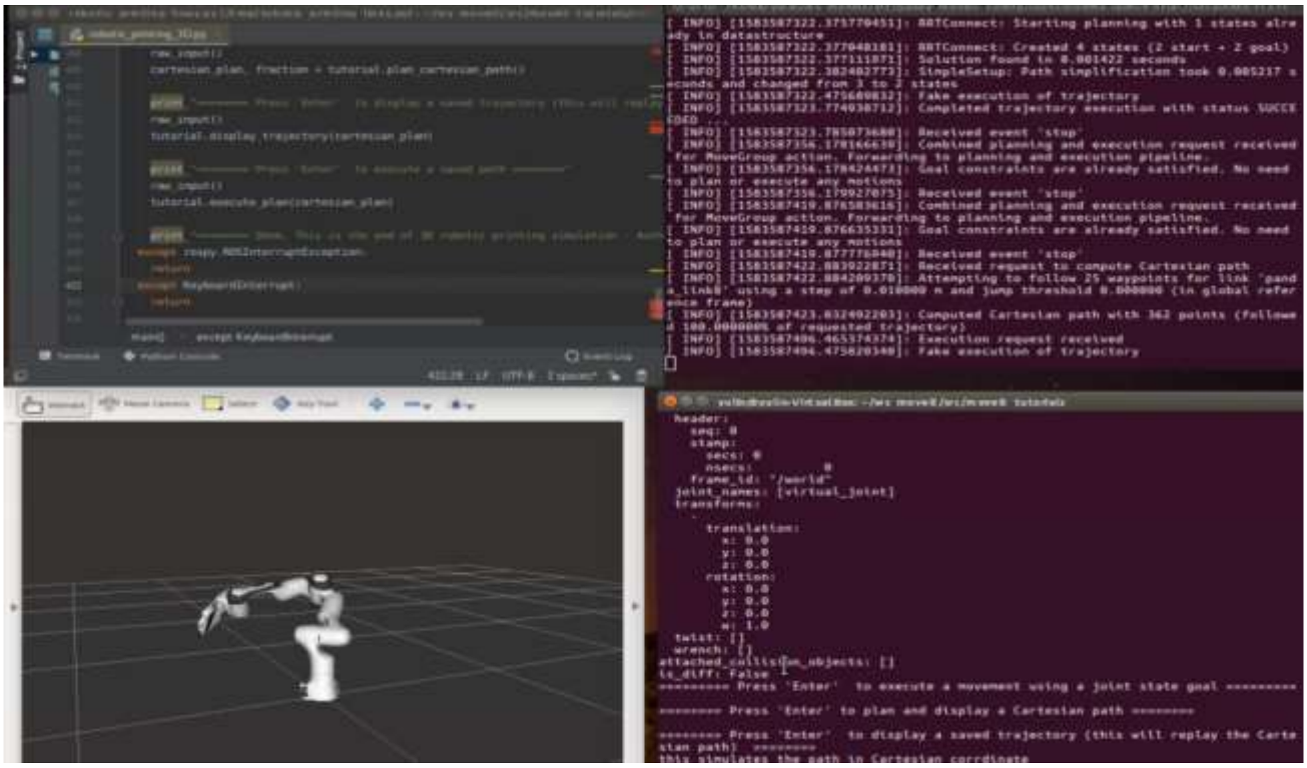


```
            raw_input()
            cartesian_plan, fraction = tutorial.plan_cartesian_path()

            print "========== Press `Enter` to display a saved trajectory (this will repla
            raw_input()
            tutorial.display_trajectory(cartesian_plan)

            print "========== Press `Enter` to execute a saved path =========="
            raw_input()
            tutorial.execute_plan(cartesian_plan)

            print "========== Done. This is the end of 3D robotic printing simulation - Auth
        except rospy.ROSInterruptException:
            return
        except KeyboardInterrupt:
            return

main() > except KeyboardInterrupt
```

```
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583587322.375770451]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ INFO] [1583587322.377048181]: RRTConnect: Created 4 states (2 start + 2 goal)
[ INFO] [1583587322.377111971]: Solution found in 0.001422 seconds
[ INFO] [1583587322.382402773]: SimpleSetup: Path simplification took 0.005217 s
econds and changed from 3 to 2 states
[ INFO] [1583587322.475689832]: Fake execution of trajectory
[ INFO] [1583587323.774938712]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583587323.785073680]: Received event 'stop'
[ INFO] [1583587356.178166639]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587356.178424473]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587356.179927075]: Received event 'stop'
[ INFO] [1583587419.876503616]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587419.876635331]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587419.877776940]: Received event 'stop'
[ INFO] [1583587422.883922871]: Received request to compute Cartesian path
[ INFO] [1583587422.884209370]: Attempting to follow 25 waypoints for link 'pand
a_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refer
ence frame)
[ INFO] [1583587423.032492203]: Computed Cartesian path with 362 points (followe
d 100.000000% of requested trajectory)
```

```
multi_dof_joint_state:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: "/world"
  joint_names: [virtual_joint]
  transforms:
    -
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  twist: []
  wrench: []
attached_collision_objects: []
is_diff: False
========== Press 'Enter'  to execute a movement using a joint state goal ==========

========== Press 'Enter' to plan and display a Cartesian path ==========

========== Press 'Enter'  to display a saved trajectory (this will replay the Carte
sian path) ==========
```

Figure 47 - Status three of Pyramid Geometry case study



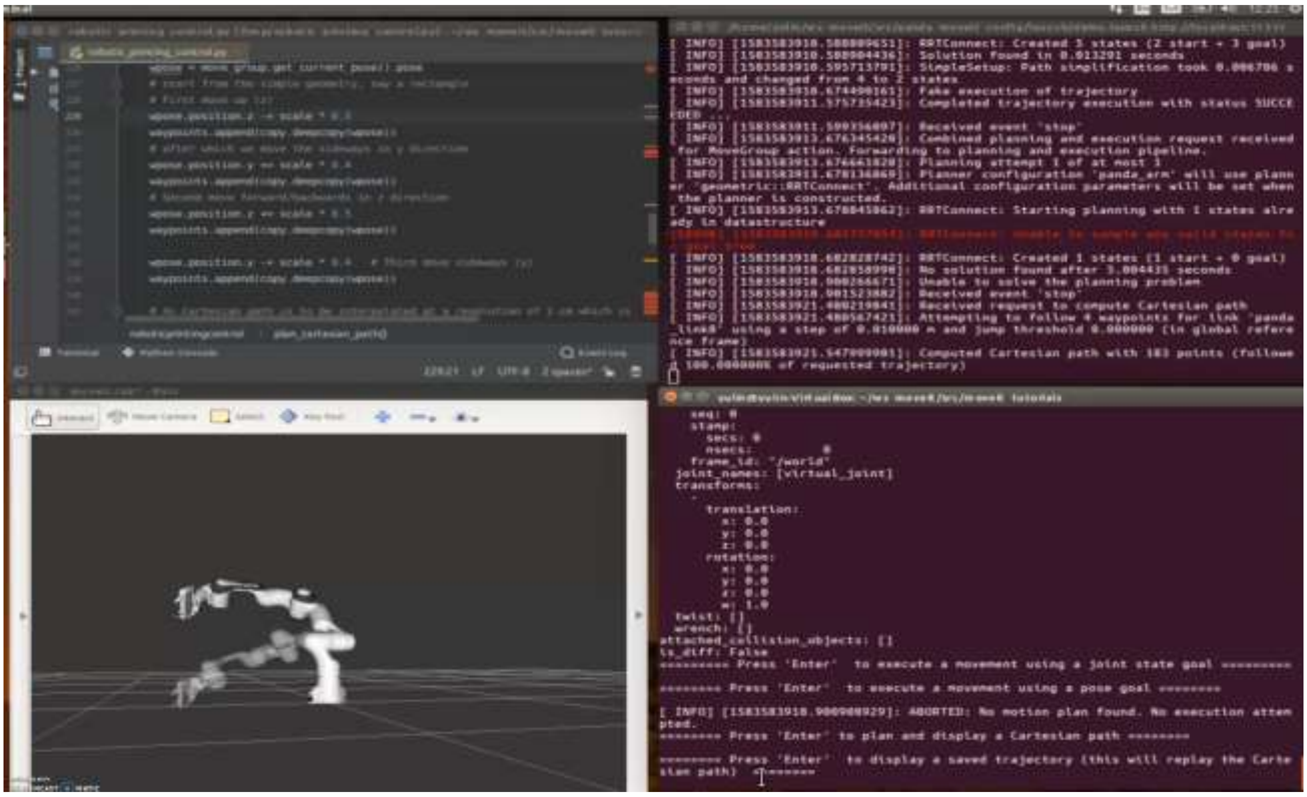
```
    raw_input()
    cartesian_plan, fraction = tutorial.plan_cartesian_path()

    print "======== Press `Enter` to display a saved trajectory (this will replay
    raw_input()
    tutorial.display_trajectory(cartesian_plan)

    print "======== Press `Enter` to execute a saved path ========"
    raw_input()
    tutorial.execute_plan(cartesian_plan)

    print "======== Done, This is the end of 3D robotic printing simulation
  except rospy.ROSInterruptException:
    return
  except KeyboardInterrupt:
    return
```

```
[ INFO] [1583587322.375770451]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ INFO] [1583587322.377048181]: RRTConnect: Created 4 states (2 start + 2 goal)
[ INFO] [1583587322.377111871]: Solution found in 0.001422 seconds
[ INFO] [1583587322.382402773]: SimpleSetup: Path simplification took 0.005217 s
econds and changed from 3 to 2 states
[ INFO] [1583587322.475600832]: Fake execution of trajectory
[ INFO] [1583587323.774930712]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583587323.785073680]: Received event 'stop'
[ INFO] [1583587356.178166630]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587356.178424473]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587356.179927075]: Received event 'stop'
[ INFO] [1583587419.876583616]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583587419.876635331]: Goal constraints are already satisfied. No need
to plan or execute any motions
[ INFO] [1583587419.877776040]: Received event 'stop'
[ INFO] [1583587422.883922871]: Received request to compute Cartesian path
[ INFO] [1583587422.884209370]: Attempting to follow 25 waypoints for link 'pand
a_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refer
ence frame)
[ INFO] [1583587423.832492203]: Computed Cartesian path with 362 points (followe
d 100.000000% of requested trajectory)
[ INFO] [1583587406.465374374]: Execution request received
[ INFO] [1583587406.475820340]: Fake execution of trajectory
```

```
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: "/world"
  joint_names: [virtual_joint]
  transforms:
    -
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  twist: []
  wrench: []
attached_collision_objects: []
is_diff: False
========== Press `Enter`  to execute a movement using a joint state goal ==========

======== Press `Enter` to plan and display a Cartesian path ========

======== Press `Enter`  to display a saved trajectory (this will replay the Carte
sian path) ========
this simulates the path in Cartesian corrdinate
```

Figure 48- Status four (Planning Path execution) of Pyramid Geometry case study

The four figures above have briefly showed the robot trajectory when attempting to print a 3D pyramid geometry. Similarly, the other four figures below showed the trajectory of the second case study – 2D shape printing.

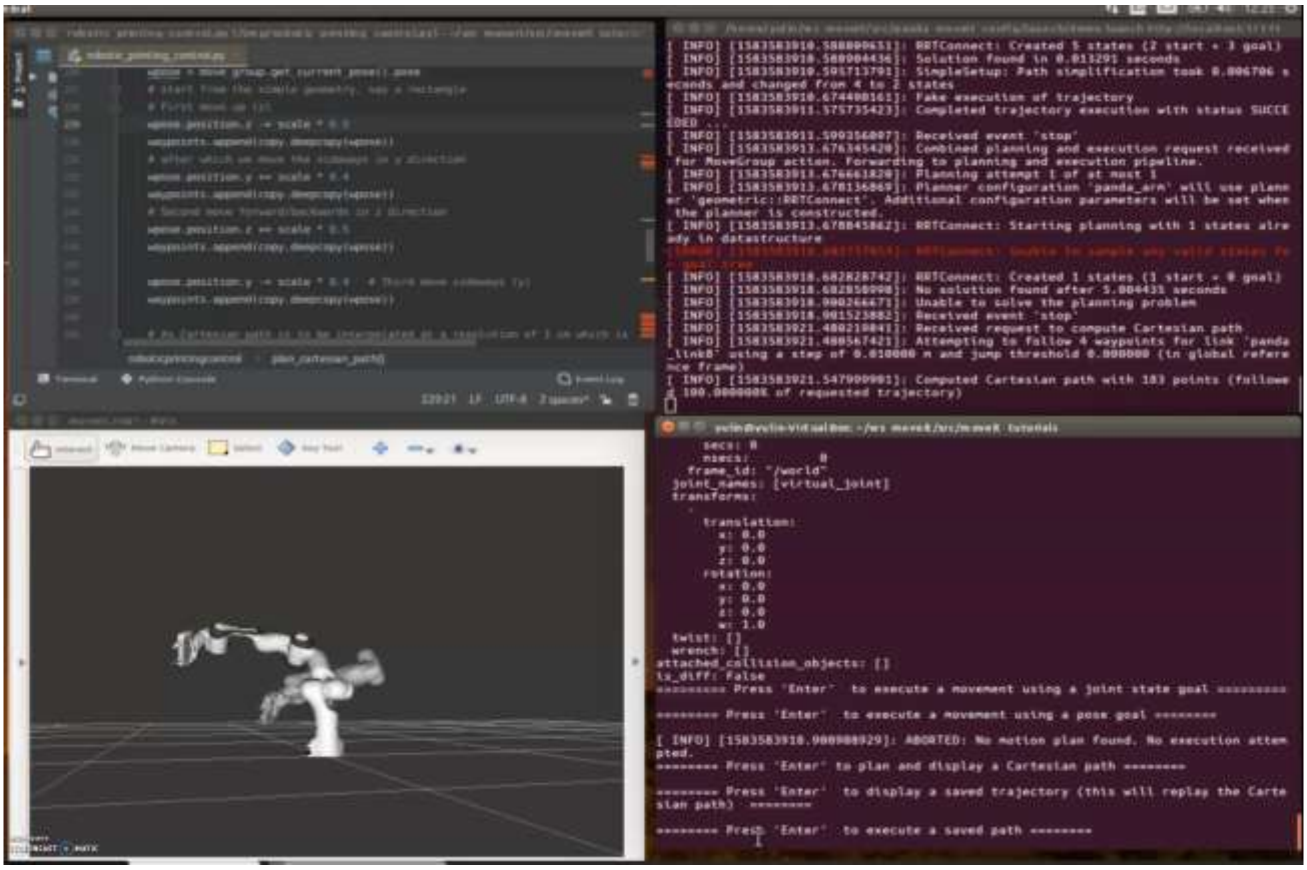


```
    wpose = move_group.get_current_pose().pose
    # start from the simple geometry, say a rectangle
    # First move up (z)
    wpose.position.z -= scale * 0.3
    waypoints.append(copy.deepcopy(wpose))
    # after which we move the sideways in y direction
    wpose.position.y -= scale * 0.4
    waypoints.append(copy.deepcopy(wpose))
    # Second move forward/backwards in z direction
    wpose.position.z += scale * 0.3
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.y -= scale * 0.4  # Third move sideways (y)
    waypoints.append(copy.deepcopy(wpose))
```

```
[ INFO] [1583583910.588009651]: RRTConnect: Created 5 states (2 start + 3 goal)
[ INFO] [1583583910.588904436]: Solution found in 0.013291 seconds
[ INFO] [1583583910.595713791]: SimpleSetup: Path simplification took 0.006706 s
econds and changed from 4 to 2 states
[ INFO] [1583583910.674490161]: Fake execution of trajectory
[ INFO] [1583583911.575735423]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583583911.599356097]: Received event 'stop'
[ INFO] [1583583913.676345420]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583583913.676661820]: Planning attempt 1 of at most 1
[ INFO] [1583583913.678136869]: Planner configuration 'panda_arm' will use plann
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583583913.678845862]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[illegible]
[ INFO] [1583583918.682828742]: RRTConnect: Created 1 states (1 start + 0 goal)
[ INFO] [1583583918.682858998]: No solution found after 5.004435 seconds
[ INFO] [1583583918.900266671]: Unable to solve the planning problem
[ INFO] [1583583918.901523882]: Received event 'stop'
[ INFO] [1583583921.480219841]: Received request to compute Cartesian path
[ INFO] [1583583921.480567421]: Attempting to follow 4 waypoints for link 'panda
_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refere
nce frame)
[ INFO] [1583583921.547009981]: Computed Cartesian path with 183 points (followe
d 100.000000% of requested trajectory)
```

```
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: "/world"
  joint_names: [virtual_joint]
  transforms:
    -
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  twist: []
  wrench: []
attached_collision_objects: []
is_diff: False
========== Press `Enter`  to execute a movement using a joint state goal ==========

======== Press `Enter`  to execute a movement using a pose goal ========

[ INFO] [1583583918.900900929]: ABORTED: No motion plan found. No execution atten
pted.
======== Press `Enter` to plan and display a Cartesian path ========

======== Press `Enter`  to display a saved trajectory (this will replay the Carte
sian path) ========
```

Figure 49 - Status One of 2D shape printing

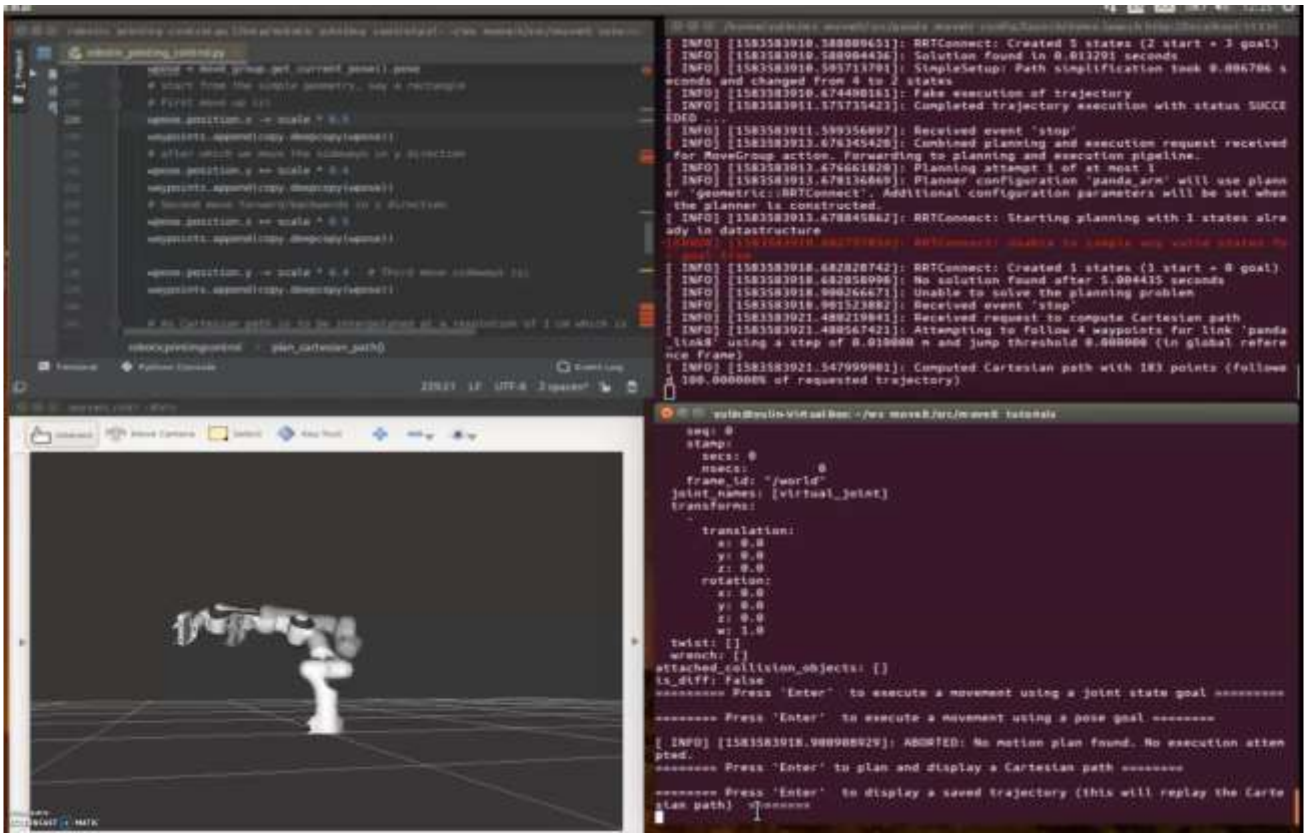


```
wpose = move_group.get_current_pose().pose
# start from the simple geometry, say a rectangle
# First move up (z)
wpose.position.z -= scale * 0.5
waypoints.append(copy.deepcopy(wpose))
# after which we move the sideways in y direction
wpose.position.y += scale * 0.4
waypoints.append(copy.deepcopy(wpose))
# Second move forward/backwards in z direction
wpose.position.z += scale * 0.5
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.4  # Third move sideways (y)
waypoints.append(copy.deepcopy(wpose))

# As Cartesian path is to be interpolated at a resolution of 1 cm which is
```

```
[ INFO] [1583583910.588009651]: RRTConnect: Created 5 states (2 start + 3 goal)
[ INFO] [1583583910.589004436]: Solution found in 0.013291 seconds
[ INFO] [1583583910.595713791]: SimpleSetup: Path simplification took 0.006706 s
econds and changed from 4 to 2 states
[ INFO] [1583583910.674400161]: Fake execution of trajectory
[ INFO] [1583583911.575735423]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583583911.599356007]: Received event 'stop'
[ INFO] [1583583913.676345420]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583583913.676661820]: Planning attempt 1 of at most 1
[ INFO] [1583583913.678136869]: Planner configuration 'panda_arm' will use plann
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583583913.678845862]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ERROR] [illegible]: RRTConnect: Unable to sample any valid states fo
r goal tree
[ INFO] [1583583918.682828742]: RRTConnect: Created 1 states (1 start + 0 goal)
[ INFO] [1583583918.682850998]: No solution found after 5.004435 seconds
[ INFO] [1583583918.900266671]: Unable to solve the planning problem
[ INFO] [1583583918.901523882]: Received event 'stop'
[ INFO] [1583583921.480219841]: Received request to compute Cartesian path
[ INFO] [1583583921.480567421]: Attempting to follow 4 waypoints for link 'panda
_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refere
nce frame)
[ INFO] [1583583921.547999981]: Computed Cartesian path with 183 points (followe
d 100.000000% of requested trajectory)
```

```
    secs: 0
    nsecs:         0
  frame_id: "/world"
joint_names: [virtual_joint]
transforms:
  -
    translation:
      x: 0.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
twist: []
wrench: []
attached_collision_objects: []
is_diff: False
========== Press 'Enter'  to execute a movement using a joint state goal ==========

======== Press 'Enter'  to execute a movement using a pose goal ========

[ INFO] [1583583918.900908929]: ABORTED: No motion plan found. No execution attem
pted.
======== Press 'Enter' to plan and display a Cartesian path ========

======== Press 'Enter'  to display a saved trajectory (this will replay the Carte
sian path) ========

======== Press 'Enter'  to execute a saved path ========
```

Figure 50- Status Two of 2D shape printing

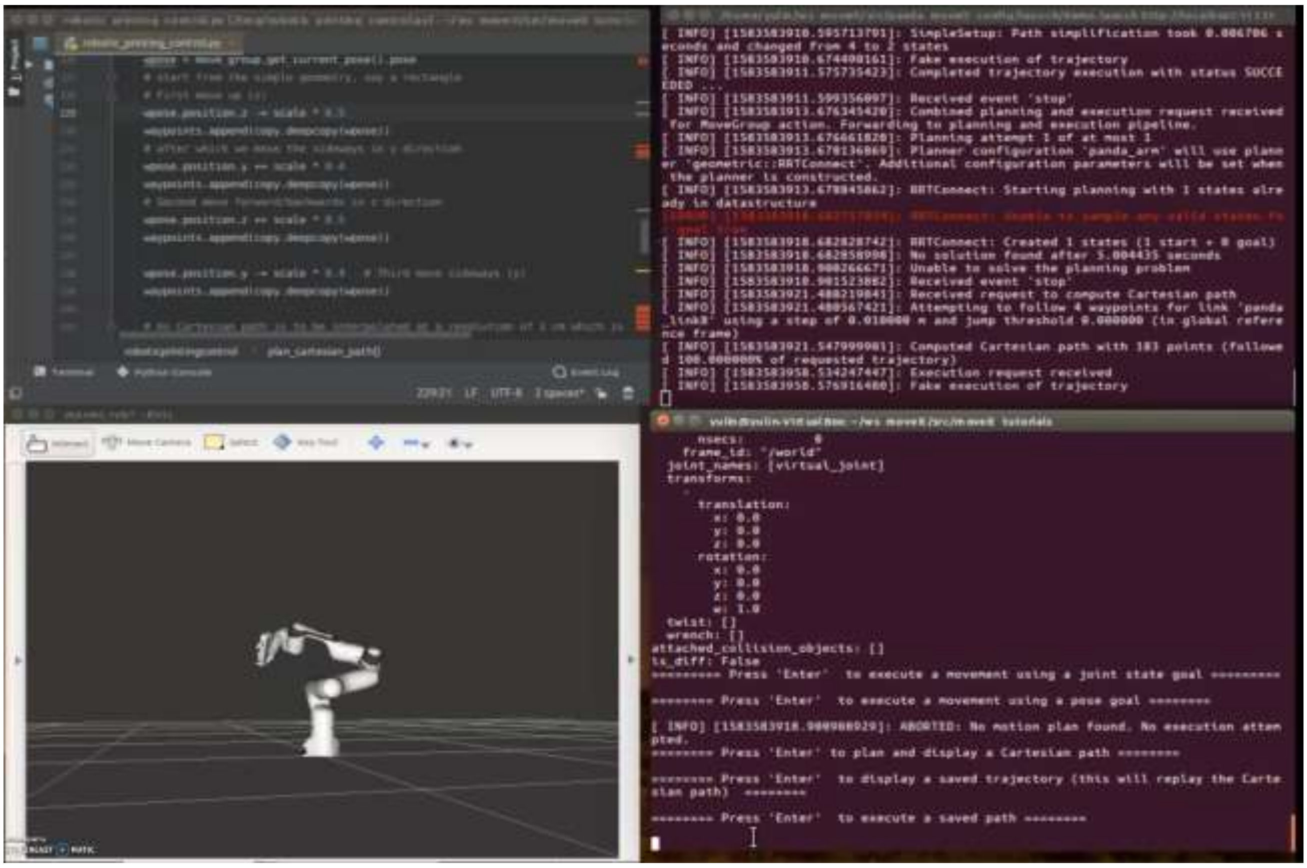


```
wpose = move_group.get_current_pose().pose
# start from the simple geometry, say a rectangle
# First move up (z)
wpose.position.z -= scale * 0.5
waypoints.append(copy.deepcopy(wpose))
# after which we move the sideways in y direction
wpose.position.y += scale * 0.4
waypoints.append(copy.deepcopy(wpose))
# Second move forward/backwards in z direction
wpose.position.z += scale * 0.5
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.4  # Third move sideways (y)
waypoints.append(copy.deepcopy(wpose))

# As Cartesian path is to be interpolated at a resolution of 1 cm which is
```

```
[ INFO] [1583583910.588009651]: RRTConnect: Created 5 states (2 start + 3 goal)
[ INFO] [1583583910.589004436]: Solution found in 0.013291 seconds
[ INFO] [1583583910.595713791]: SimpleSetup: Path simplification took 0.006706 s
econds and changed from 4 to 2 states
[ INFO] [1583583910.674400161]: Fake execution of trajectory
[ INFO] [1583583911.575735423]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583583911.599356007]: Received event 'stop'
[ INFO] [1583583913.676345420]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583583913.676661820]: Planning attempt 1 of at most 1
[ INFO] [1583583913.678136869]: Planner configuration 'panda_arm' will use plann
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583583913.678845862]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[ERROR] [illegible]: RRTConnect: Unable to sample any valid states fo
r goal tree
[ INFO] [1583583918.682828742]: RRTConnect: Created 1 states (1 start + 0 goal)
[ INFO] [1583583918.682850998]: No solution found after 5.004435 seconds
[ INFO] [1583583918.900266671]: Unable to solve the planning problem
[ INFO] [1583583918.901523882]: Received event 'stop'
[ INFO] [1583583921.480219841]: Received request to compute Cartesian path
[ INFO] [1583583921.480567421]: Attempting to follow 4 waypoints for link 'panda
_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refere
nce frame)
[ INFO] [1583583921.547999981]: Computed Cartesian path with 183 points (followe
d 100.000000% of requested trajectory)
```

```
  seq: 0
  stamp:
    secs: 0
    nsecs:         0
  frame_id: "/world"
joint_names: [virtual_joint]
transforms:
  -
    translation:
      x: 0.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
twist: []
wrench: []
attached_collision_objects: []
is_diff: False
========== Press 'Enter'  to execute a movement using a joint state goal ==========

======== Press 'Enter'  to execute a movement using a pose goal ========

[ INFO] [1583583918.900908929]: ABORTED: No motion plan found. No execution attem
pted.
======== Press 'Enter' to plan and display a Cartesian path ========

======== Press 'Enter'  to display a saved trajectory (this will replay the Carte
sian path) ========
```

Figure 51- Status Three of 2D shape printing



```
wpose = move_group.get_current_pose().pose
# start from the simple geometry, say a rectangle
# First move up (z)
wpose.position.z -= scale * 0.3
waypoints.append(copy.deepcopy(wpose))
# after which we move the sideways in y direction
wpose.position.y += scale * 0.4
waypoints.append(copy.deepcopy(wpose))
# Second move forward/backwards in z direction
wpose.position.z += scale * 0.3
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.4  # Third move sideways (y)
waypoints.append(copy.deepcopy(wpose))

# We Cartesian path is to be interpolated at a resolution of 1 cm which is
```

```
[ INFO] [1583583910.595713701]: SimpleSetup: Path simplification took 0.006706 s
econds and changed from 4 to 2 states
[ INFO] [1583583910.674400161]: Fake execution of trajectory
[ INFO] [1583583911.575735423]: Completed trajectory execution with status SUCCE
EDED ...
[ INFO] [1583583911.599356097]: Received event 'stop'
[ INFO] [1583583913.676345420]: Combined planning and execution request received
 for MoveGroup action. Forwarding to planning and execution pipeline.
[ INFO] [1583583913.676661820]: Planning attempt 1 of at most 1
[ INFO] [1583583913.678136860]: Planner configuration 'panda_arm' will use plann
er 'geometric::RRTConnect'. Additional configuration parameters will be set when
 the planner is constructed.
[ INFO] [1583583913.678845862]: RRTConnect: Starting planning with 1 states alre
ady in datastructure
[illegible]
[ INFO] [1583583918.682828742]: RRTConnect: Created 1 states (1 start + 0 goal)
[ INFO] [1583583918.682858998]: No solution found after 5.004435 seconds
[ INFO] [1583583918.900266671]: Unable to solve the planning problem
[ INFO] [1583583918.901523882]: Received event 'stop'
[ INFO] [1583583921.480219841]: Received request to compute Cartesian path
[ INFO] [1583583921.480567421]: Attempting to follow 4 waypoints for link 'panda
_link8' using a step of 0.010000 m and jump threshold 0.000000 (in global refere
nce frame)
[ INFO] [1583583921.547999981]: Computed Cartesian path with 103 points (followe
d 100.000000% of requested trajectory)
[ INFO] [1583583958.534247447]: Execution request received
[ INFO] [1583583958.576916480]: Fake execution of trajectory
```

```
    nsecs:         0
  frame_id: "/world"
joint_names: [virtual_joint]
transforms:
  -
    translation:
      x: 0.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
twist: []
wrench: []
attached_collision_objects: []
is_diff: False
========== Press 'Enter'  to execute a movement using a joint state goal ==========

======== Press 'Enter'  to execute a movement using a pose goal ========

[ INFO] [1583583918.900988029]: ABORTED: No motion plan found. No execution atte
mpted.
======== Press 'Enter' to plan and display a Cartesian path ========

======== Press 'Enter'  to display a saved trajectory (this will replay the Carte
sian path)  ========

======== Press 'Enter'  to execute a saved path ========
```

Figure 52 - Status Four of 2D shape printing

However, as this entire procedure is a dynamic one, it is still too abstract to visualise it by only observing some fragments as those figures above are just some screenshots, therefore, in the supplementary file, two video recordings have been attached showing the dynamic trajectories to help readers have a better understanding of how the robotic arm moves.

5.3 OPEN SOURCE PYTHON SCRIPT FLOWCHART AND EXPLANATION

As aforementioned, in order to ensure that the robotic arm trajectory can be automatically controlled, Python was used here to interact with Gazebo. Alternatively, C++ can also be used, however, as Python is becoming more and more popular in recent years, in this project, Python was chosen as the main programming language. Holistically speaking, the code consists of one class and eight functions, that are 'roboticprintingcontrol' class, all_close function, go_to_joint_state function, go_to_pose_goal function, plan_cartesian_path function, display_trajectory function, execute_plan function and main function, respectively. To make readers understand the idea of the entire code, a flow chart along with some text explanations has been attached below in Figure.54.

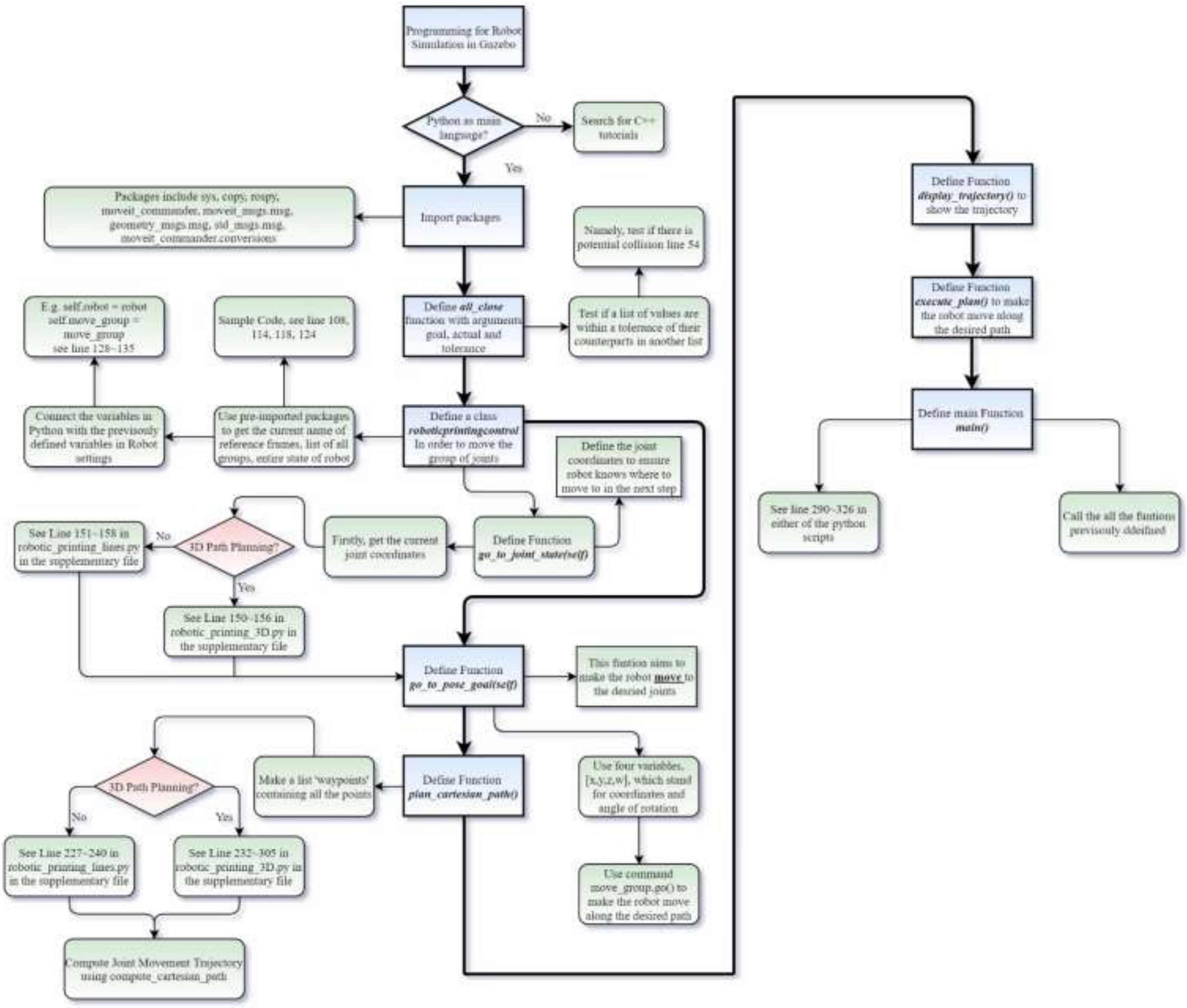

Programming for Robot Simulation in Gazebo
Python as main language?
No
Search for C++ tutorials
Yes
Import packages
Packages include sys, copy, rospy, moveit_commander, moveit_msgs.msg, geometry_msgs.msg, std_msgs.msg, moveit_commander.conversions
Define all_close function with arguments goal, actual and tolerance
Test if a list of values are within a tolerance of their counterparts in another list
Namely, test if there is potential collision line 54
Define a class roboticprintingcontrol In order to move the group of joints
Use pre-imported packages to get the current name of reference frames, list of all groups, entire state of robot
Sample Code, see line 108, 114, 118, 124
Connect the variables in Python with the previsouly defined variables in Robot settings
E.g. self.robot = robot self.move_group = move_group see line 128~135
Define the joint coordinates to ensure robot knows where to move to in the next step
Define Function go_to_joint_state(self)
Firstly, get the current joint coordinates
3D Path Planning?
No
See Line 151~158 in robotic_printing_lines.py in the supplementary file
Yes
See Line 150~156 in robotic_printing_3D.py in the supplementary file
Define Function go_to_pose_goal(self)
This funtion aims to make the robot move to the desried joints
Use four variables, [x,y,z,w], which stand for coordinates and angle of rotation
Use command move_group.go() to make the robot move along the desired path
Define Function plan_cartesian_path()
Make a list 'waypoints' containing all the points
3D Path Planning?
No
Yes
See Line 227~240 in robotic_printing_lines.py in the supplementary file
See Line 232~305 in robotic_peinting_3D.py in the supplementary file
Compute Joint Movement Trajectory using compute_cartesian_path
Define Function display_trajectory() to show the trajectory
Define Function execute_plan() to make the robot move along the desired path
Define main Function main()
See line 290~326 in either of the python scripts
Call the all the funtions pervisouly ddeifned

Figure 53 - Flowchart of Python Script

Shown in Figure.54, Python opted as the programming language because it presents one significant benefit. In general, when it comes to building more complicated applications with MoveIt, developers are required to dig into Python API because it can help developers to directly skip many of the ROS service/action layers that will result in much faster performance and thereby save a lot of time. Although this might not seem that important in this project because of the small scale of the project, when designers tend to simulate a more complicated trajectory, the amount of time to be saved could be quite large.

Furthermore, in Figure.54, the boxes in blue represent the core workflow, which includes functions and class definitions. Each green box on the branch gives corresponding explanations and codes for the core steps.

5.4 OVERVIEW OF HOW GAZEBO INTERFACE WORKS

After introducing a great amount of information about using Gazebo, from ROS system installation, setting up, manual control, Python-Gazebo interface to control and the simulation results, it is also important to understand what different types of plugin interfaces that Gazebo can provide.

Gazebo offers lots of plugin interfaces so that users can override different parts of the framework and apply new concepts without having to fully understand and manipulating the core algorithms of the framework. Typically, Gazebo takes the robot model and joint states as two inputs and sends out data values to robot actuators as an output. Once receiving the input arguments, some embedded/plugin interfaces will be implemented depending on which plugin interface users would like to use, and the result data values will be sent to robot actuators to control the robot to move. Gazebo provides a wide variety of plugin interfaces including high-level capacity plugins, kinematics plugins, collision plugins, collision detection plugins, controller manager plugins, planning plugins, octomap updater plugins and planning request adapter plugins [34]. In order to let users have a more explicit picture, a flowchart has been made in Figure.55 to help readers understand the wide variety of plugin interfaces Gazebo contains.

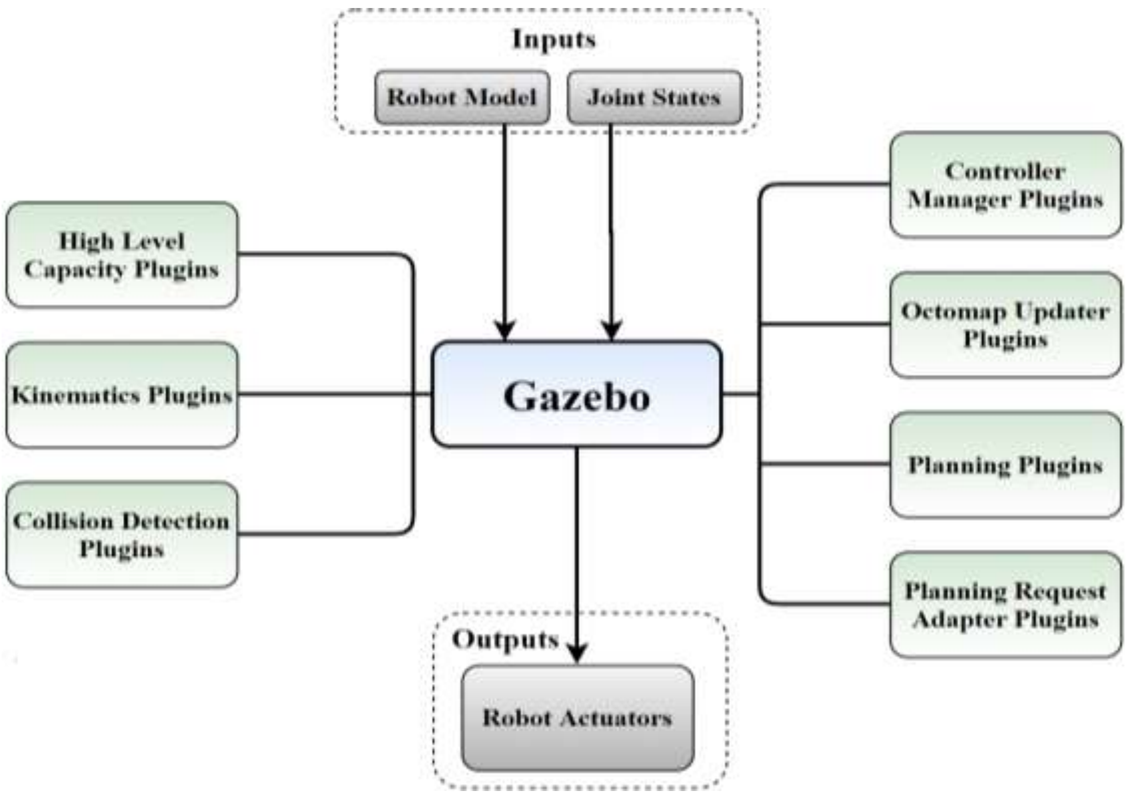


Figure 54 – Different Plugin Interfaces, In/Outputs in Gazebo

6. CHAPTER 6: OVERVIEW OF HOW GAZEBO SYSTEM WORKS

In the previous chapters, a fundamental introduction of Gazebo from installation, setting up to plugin interfaces have been given to readers. In this chapter, a bigger picture, namely, an overview of how the entire Gazebo system works will be explained.

6.1 GAZEBO SYSTEM ARCHITECTURE

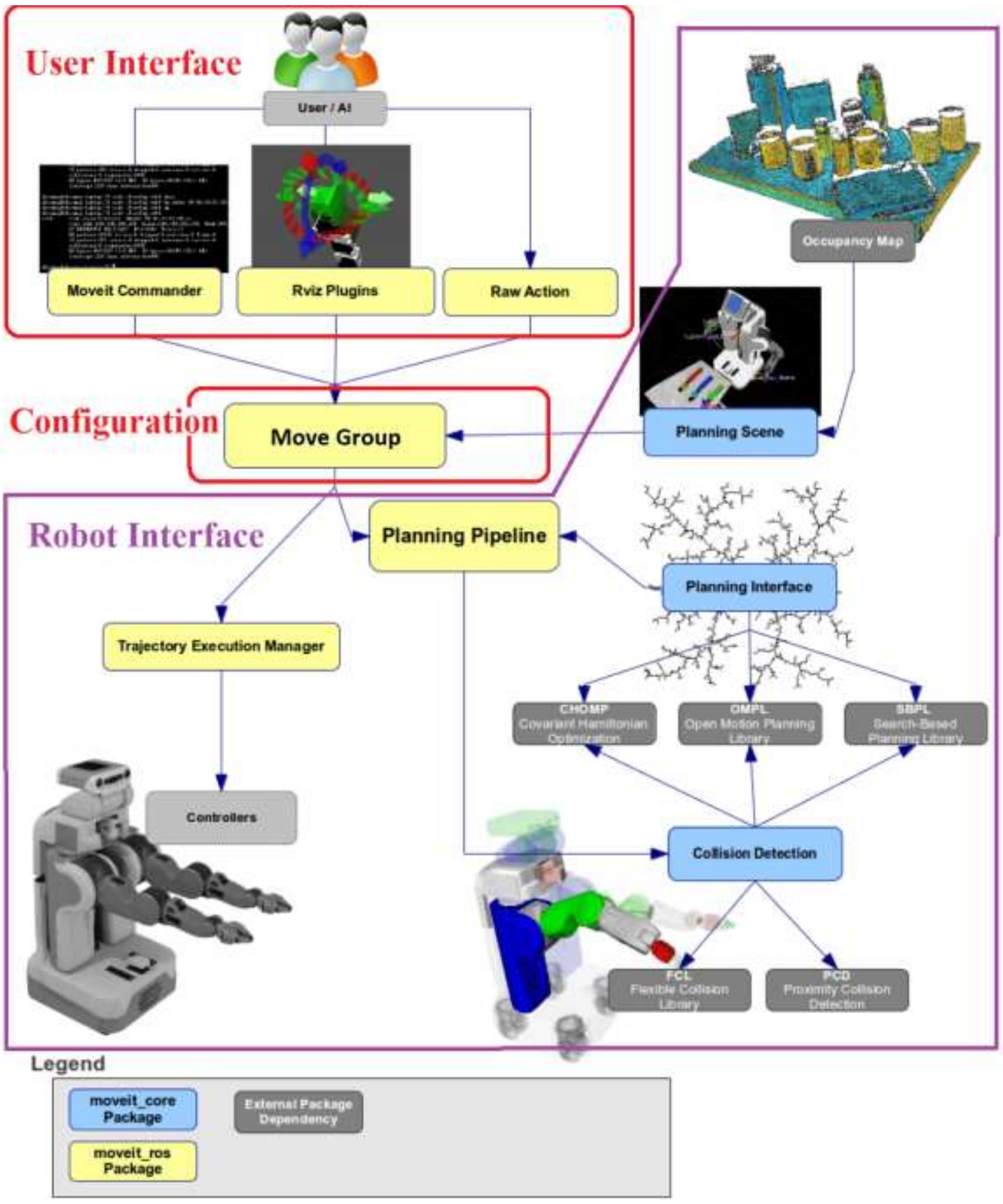


Figure 55 - Diagram showing the pipeline of Gazebo working flow [35]

Figure.56 shows the system architecture for the primary node Move Group provided by Gazebo. This Move Group node serves as an 'integrator', namely, to pull all the individual components together to provide a set of ROS actions and services for users to use. This might sound a bit abstract to understand. However, once you split the system architecture into three components, things become a lot easier to understand. To understand this in a better way, Figure.57 has also been attached below illustrating how the 'integrator' works and what are the in/outputs of this node.

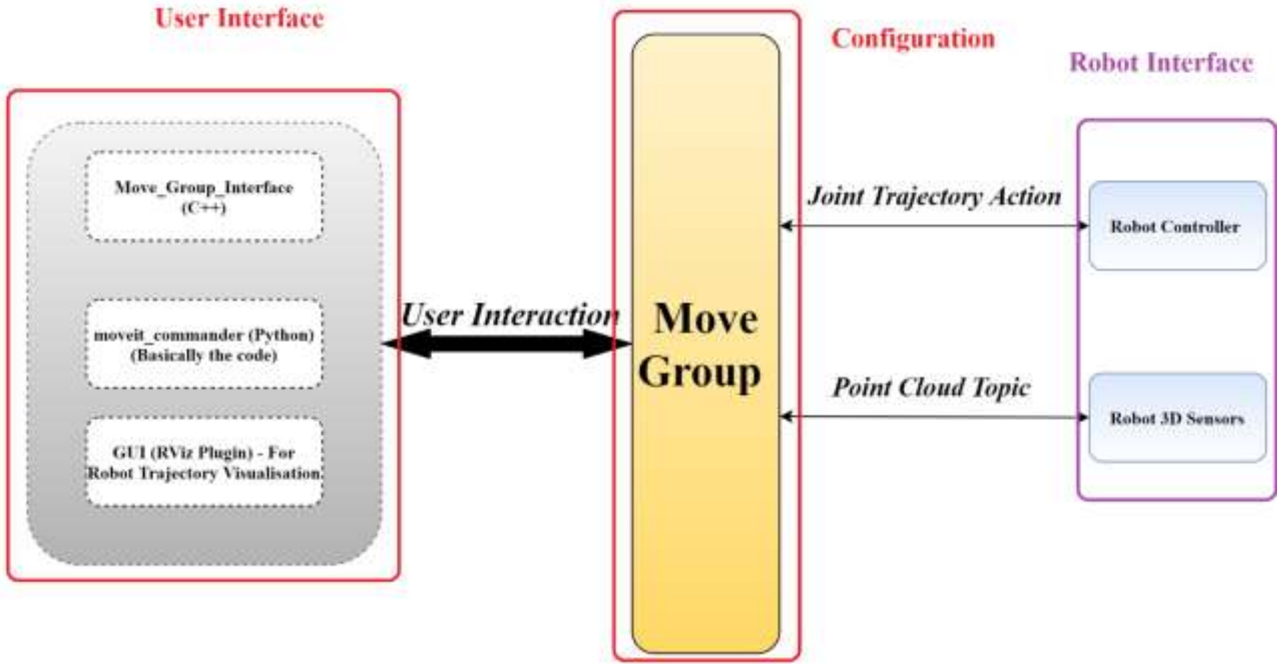


Figure 56: (Left to Right) User Interface - Move Group Node (aka Configuration) - Robot Actuator (aka Robot Interface)

It is noticeable that there are three components in Figure.57, and they are User Interface, Configuration and Robot Interface, respectively. Those three components also correspond to the three parts in Figure.56. the In terms of the User Interface, three ways have been listed for users to access the actions which are a) In C++, using the move_group_interface package that provides a C++ interface to Move Group; b) In Python, using the moveit_commander (The language used in this project); c) Through a GUI Interface, by using RViz, this method corresponds to the manual trajectory simulation mentioned in the previous Chapter.

As far as the second component – Configuration, it usually requires to take in some specific parameters of robots such as joint limits, kinematics, motion planning, perception and other information. Configuration files for these components are automatically generated by the MoveIt setup assistant and stored in the config directory of the corresponding MoveIt config package for the robot [35].

Finally, for the last part – robot interface, the ROS interface firstly communicates with the robot to get current state information (e.g. positions of the joints), get point clouds and other sensors data from the robot 3D sensors, and then ROS will transmit those data values to the robot controller to actuate the robotic arm.

6.2 MOTION PLANNING

As aforementioned in the Gazebo Interface overview, in order to control or design a motion planning trajectory, users will have to work with motion planners through a plugin interface. This allows Gazebo to communicate with and have access to different motion planners from multiple libraries which makes Gazebo easily extensible. The interface to the motion planners is through a ROS Action or service (offered by the Move Group node) [35]. To satisfy a designer’s target, typically to move an arm to a different location or generate a new pose, a few constraints should be checked beforehand. In Gazebo, there are five inbuilt kinematic constraints including a) position constraints, as in to restrict the position of a link to lie within a region of space; b) orientation constraints to restrict the orientation of a link to lie within the specified roll, pitch or yaw limits, visibility constraints to restrict a point on a link to a line within the visibility cone for a particular sensor; d) joint constraints to restrict a joint to be located between two values and e) constraints specified by users [36]. Therefore, those five restrictions should be taken into account and checked before designers start planning the robot motion.

After setting up all the restrictions, the Move Group node will automatically generate the desired trajectory using the Inverse Kinematics algorithm (typically but not necessarily to be true all the time) to fit the requirement of your motion planning request. This trajectory will actuate the arm and group of joints to the pre-set locations. One thing worth noting here is that the result is an optimised trajectory rather than some random path because Gazebo or specifically Move Group node will use the desired maximum velocities and accelerations to generate an optimised trajectory that obeys velocity and accelerations limits at the joint level. The entire motion planning process has been illustrated in Figure.58.

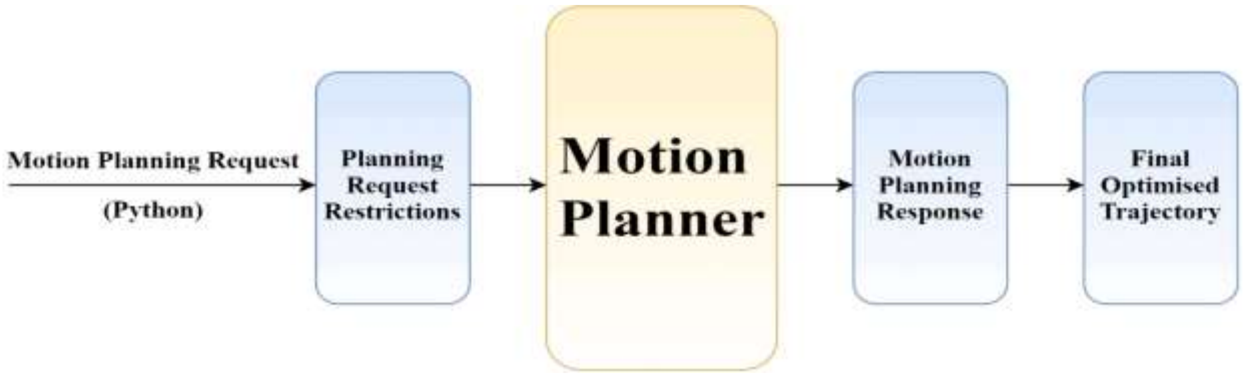


Figure 57 - Motion Planning Process From initial request setting to final optimised trajectory generation

7. CHAPTER 7: ELECTRICAL COMPONENT – ARDUINO BOARD

As is known that robot control requires the combination of mechanical knowledge, software simulation and electrical control. In previous chapters, the first two parts have been explained, and in chapter seven, some basic information about the Arduino board will be given. Due to the limited time and devices, the practical coding test on Arduino has not been conducted, therefore, this chapter is dominantly a literature review-based research and an induction of the Arduino Board.

7.1 ARDUINO BOARD CHOICE

Recall the target of this project, a 3D printer head will be attached to the robot arm to print out continuous fibres, and in this case, an Arduino board will be required to control the 3D printer head. Different from the traditional 3D printers that usually take thermoplastic materials as main feeding materials, this 3d printer uses continuous fibres as the main material. One thing that is very different between continuous carbon fibres and traditional thermoplastic materials is that continuous carbon fibre does not have thermoplastic properties, which means that an extra cutter will be needed to cut off the fibre once it stops printing. In order to fit this special trait, the Arduino board should contain several spare connectors for cutter (servo motor, in fact) connection. Under this special condition, a MEGATRONICS V3.3 has been selected as the desired circuit board as it contains a spare connector for servo connection, as shown in Figure.59.

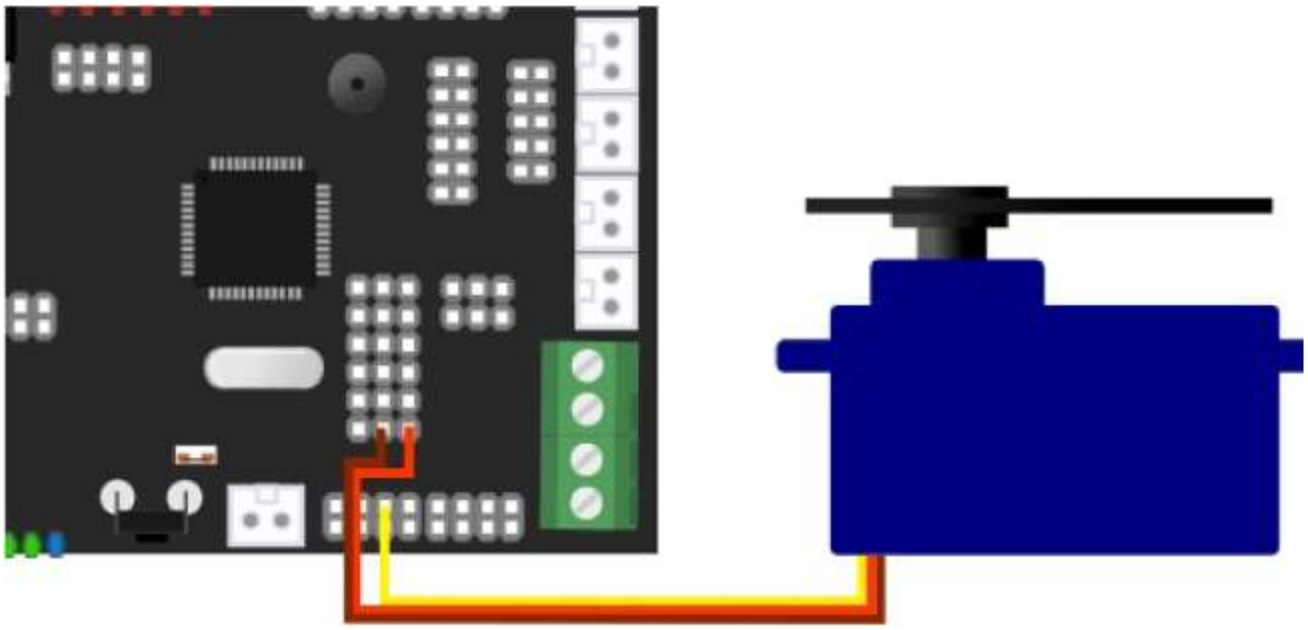


Figure 58 - Megatronics supports a Servo motor for fibre cutting

7.2 ARDUINO BOARD OVERVIEW

Similar to typical circuit boards for 3D printers, Megatronics is also based on many famous open-source 3D printing-oriented products including Arduino Mega, RAMPS, SD Ramps. It combines all major features of these boards into a single board solution for more reliable 3D-printing.

Megatronics has an Atmega2560 processor with 256 KB memory, running at 16MHz. The board can be connected to a PC using a normal USB cable. It will register as FTDI FT232R device. The board is compatible with Arduino Mega 2560 and will therefore be easily programmed from the Arduino IDE [37]. Further to this, the operating voltage for Megatronics is 5V, and the DC per I/O pin is 40 mA. Table two below is an overview of the technical specifications of this board, and Figure.60 below shows the configurations of this product.

Table 2 - Technical Specifications of Arduino Board Megatronics [37]

Microcontroller	**Atmega 2560-16AU**
Operating Voltage Electronics	5V
Operating Voltage	12-24V (15A heated bed, 7A electronics)
DC Current per I/O Pin	40mA
High Clock Speed	16Mhz

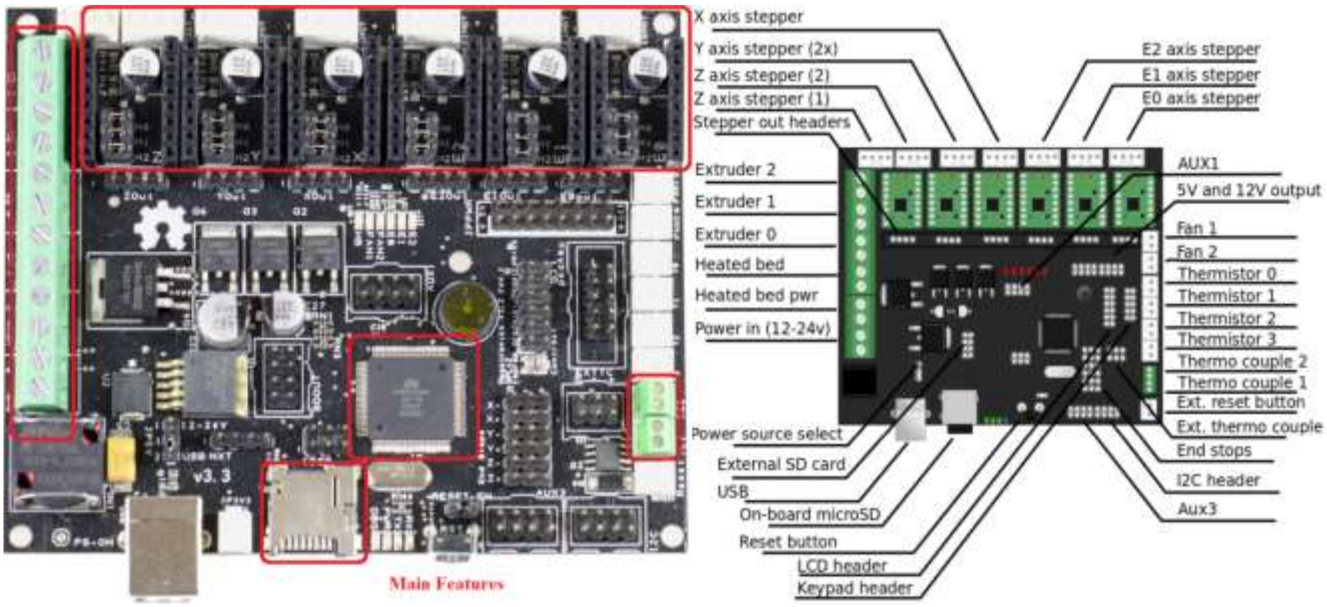


Figure 59 - Arduino Board Megatronics [37]; Left: Real Board Right: Schematic drawing showing the outputs of each connector

As illustrated in Figure.60, there are five major features on the circuit board including SD card, Atmega 2560 Processor, thermocouple, sic MOSFETS and stepper drivers. The details of those main features are shown in Table three below.

Table 3 - Megatronics Major Features [37]

	CPU: Atmega2560 **Atmega2560 processor with 256 kB memory, running at 16Mhz**
	Thermocouple Onboard support for connecting two thermocouples
	SD Card Autonomous printing from Micro SD card onboard or external SD card, using the external SD card PCB module. Now with SD card detection pin
	Six MOSFETs The board has 3 regular MOSFETs (25A), two 1A MOSFETs (fans) and one MOSFET for the heated bed (IRLS3034PBF) to support many needs.
	Up to 6 stepper drivers Compatible with RAMPS, 6 slots for stepper drivers (X, Y, Z, E2, E1, E0, see Figure.59). Modularized to make replacement easy for damaged drivers.

It can be seen there are a lot of connectors on Megatronics to control different components on the printer head such as thermocouples, thermistor, fan, step motors, extruder heads. Therefore, another very important aspect is to figure out what output each connector corresponds to. To understand this, a schematic view of Megatronics in Figure.61 has been attached showing the corresponding outputs of the connectors in Megatronics.

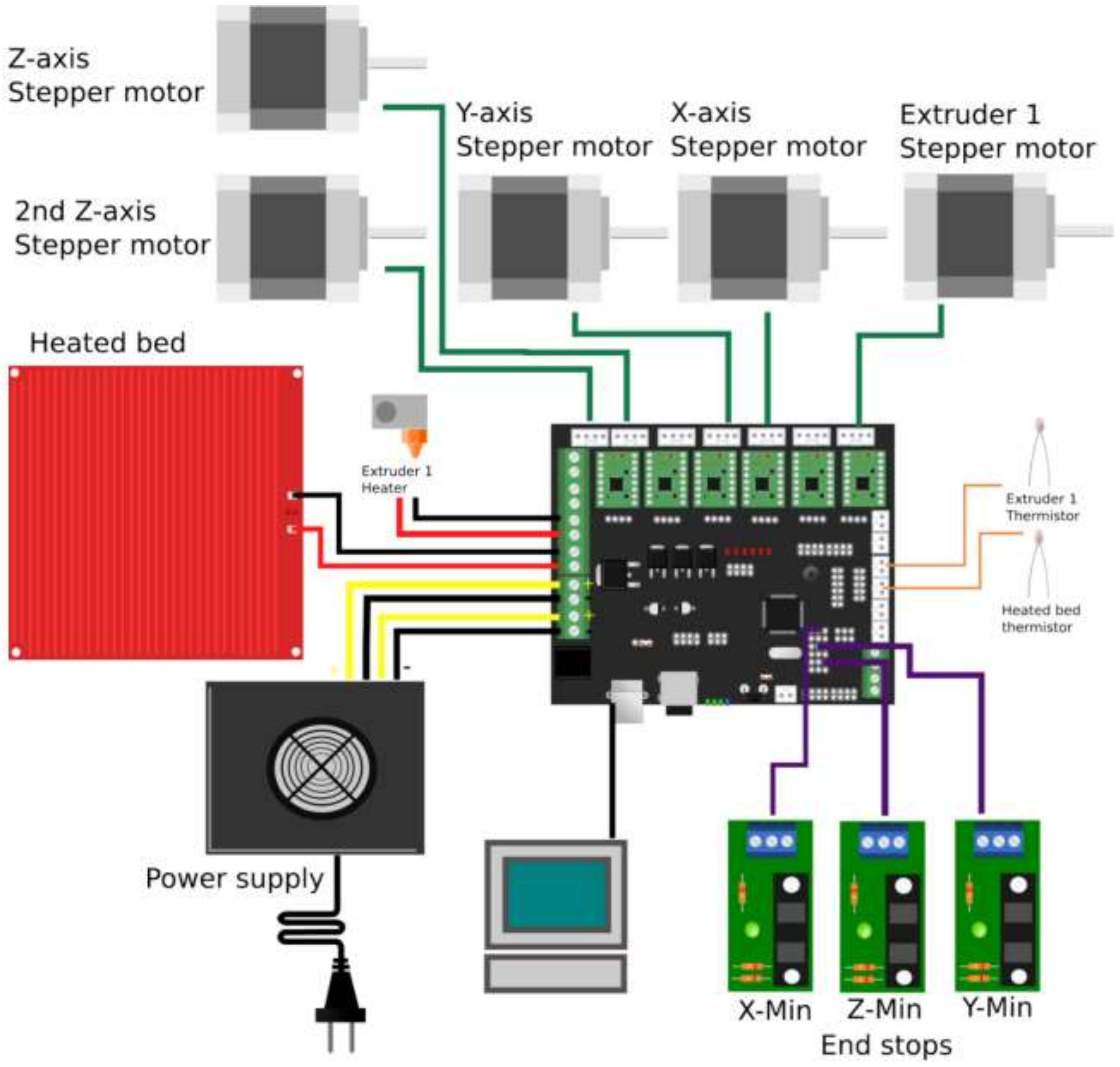


Figure 60 - The outputs of the Arduino Board Megatronics [37]

8. CHAPTER 8: PROJECT MANAGEMENT

Overall, the project began with proposing a novel 3D printing method, which is to combine the traditional 3D printer head with a robotic arm to construct products with more complicated structures. This project can mainly be split into three components, which are Mechanical part – 3D printer head design, Software part – Gazebo Simulation and Electrical part – Arduino Board understanding.

In terms of the entire project, a literature review was initially done to research and understand state-of-the-art current robotic 3D printing technologies, ATP, AFP techniques, in particular and robotic path planning control methods (ROS Gazebo Software). In terms of the software part, originally, the core target was to practically test the robot at Bayes Centre. However, due to the limited time and resources, instead of doing the practical test, two simulation cases (2D and 3D trajectory (Pyramid shape) planning and simulation) were set as the core aims of the project. To achieve this, a Python script was written, which later turned out to interact with Gazebo to achieve the desired trajectory successfully. Further to this, as far as the mechanical and electrical part, a 3D printer head assembly has been designed and the basic in/outputs and configurations of the Arduino Circuit Board have also been investigated.

A Gantt chart (Figure.62) has been made to illustrate the timeline, one thing that should be noted is that this Gantt Chart is not completely the same as the one in the interim report because of some undesired conditions, therefore, some adjustments have been made after the submission of the interim report.

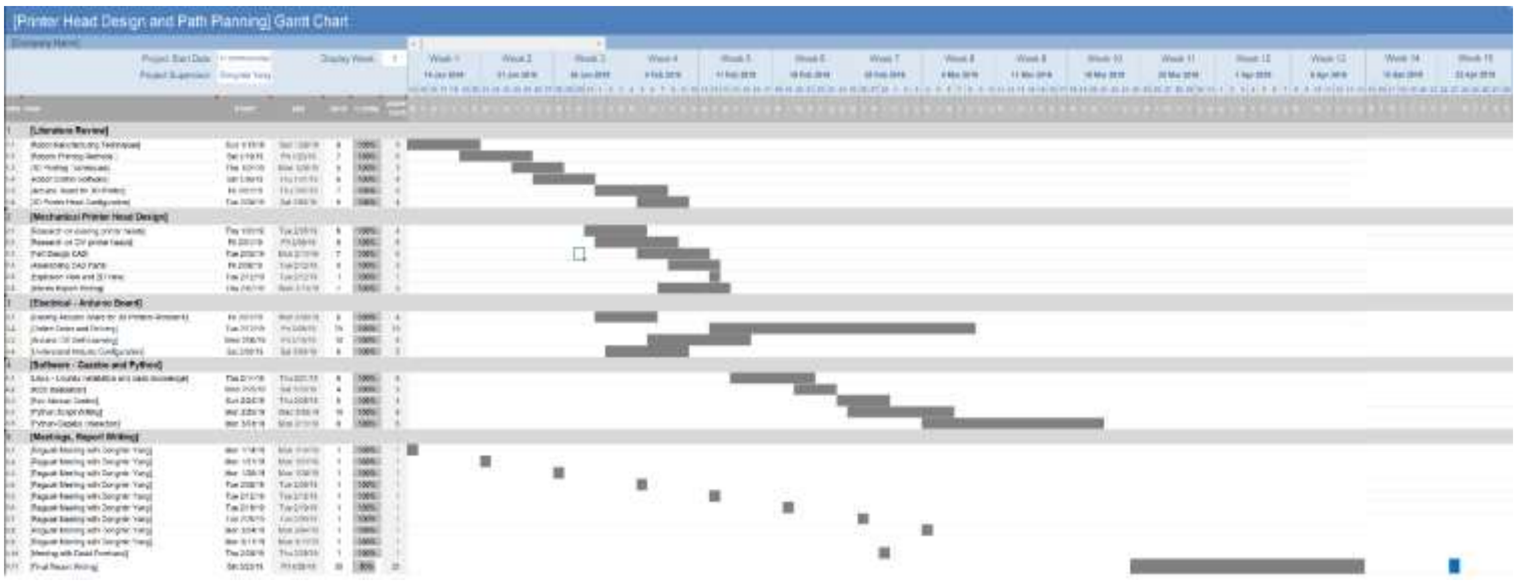


Figure 62 - Gantt chart during the entire project

9. CHAPTER 9: DISCUSSION

9.1 LIMITATIONS AND FURTHER WORK

Throughout the entire project, a few reasonable results have been obtained including mechanical design, understanding of electronics configuration and software simulation. However, there are still some deficiencies and imperfections in the methodology, design and results.

First and foremost, as the time and device are limited, there was no chance for practical testing, therefore, all the designs and simulations still stay on theory, which means that there will potentially be many issues coming up once it is applied in reality. Therefore, for further work, one thing that is urgently required to do is to put the theoretical design into practice and make a trail experiment by manufacturing the printer head and testing the performance on the robot arm.

Furthermore, the lack of prior experience with Gazebo, ROS and Linux (Ubuntu) also results in confusion and incompetency when doing robot arm simulations. In terms of further development, by further practicing, simulation has finally been successfully achieved with the aid of a large number of open-source external resources.

For the electrical section, the main limitation is the amount of time was consumed during delivery, which results in a very short time to code and test.

9.2 EXPLANATION OF RESULTS

According to the targets set at the beginning, the majority of core aims including the simulations of two cases, printer head CAD design, have been achieved despite that Arduino coding and testing was not finished in time due to some inevitable situations. However, one main deficiency that goes beyond expectation is the mechanical cutter design. During the later process of designing cutter, it was found out that the cutter was difficult to fit in the pre-designed printer head as the pre-designed printer head was already very compact. This aspect, to some extent, is a flaw as the cutter was not carefully considered at the beginning when designing the printer head. Moreover, as far as the cutter itself, the way to achieve the force transmission of the cutter is also a tricky problem, namely, how to use a servo motor to control the cutter at the right time to generate adequate force and thereby cut off the continuous carbon fibre filament still requires further thinking and investigations.

10. CHAPTER 10: CONCLUSION

Throughout the entire project, the majority of aims and objectives proposed at the beginning have been achieved. Many factors guided me down to the accomplishment of the aims and eventual project success including abundant background literature review, sufficient external resources, software simulation and engineering design. In this section, an overview will be given to summarise and conclude all the work that has been done in order to achieve each target.

According to chapter 1, there were in six objectives in total including:

1. Get the state-of-the-art FDM printing, ADP and ATP working principles
2. Printer Head Design
3. Arduino Board Configuration and Arduino Path Planning Control
4. Robotic Arm configuration understanding
5. Movement of robotic arm simulation in software Gazebo-Python Interface in Ubuntu
6. Gazebo and ROS overview

In terms of objective 1, it has been mentioned in Chapter 2 – Literature Review, giving a clear and detailed explanation of how conventional 3D printing works.

In Chapter 6 – Overview of ROS, the holistic workflow such as how Gazebo interface works and how Gazebo manages to convert data from users to robot path planning has been illustrated with the aid of many flowcharts.

In order to meet objectives 4 and 5, two Chapters – Chapter 4 and Chapter 5 were used. In Chapter 4, it started with ROS installation and setting up, then, a series of critical parameters (key configurations) such as joints and links will be defined to make preparations for the manual simulation is RViz (Objective 4). After finishing those preparations, in Chapter 5 – Gazebo&Python Interface, Python script was incorporated and connected to the Gazebo to achieve full automation, and two case studies have also been successfully simulated.

Moreover, Chapter 2 – Mechanical Design, a 3D printer head has been assembled based on pre-designing different parts of the 3D printer head. One thing that should be noticed is that not all of the parts were manually designed as there are already a large number of existing components such as gears, bears, bolts and nuts. The successful CAD design of the printer meets objective three.

Finally, as far as objective 3, it is worth noting that the other half of the objective, specifically, Arduino Board Configuration, has been achieved. Arduino path planning control was not realised due to two main reasons. The first reason is that the ordering and delivery were very time-consuming, and by the time the Megatronics board arrived, there were only approximate four weeks left. Moreover, due to the global undesired situation, the university was shut down in a relatively short time, which means that the project was terminated all of a sudden leaving no time at all for coding or testing.

ACKNOWLEDGEMENT

The author would like to express his profound gratitude to the project supervisor, Dr. Dongmin YANG, who not only inspired the author with the project topic and provided illuminating guidance till the end, but also encouraged the author constantly and mentored him to put through some of the hardest moment during the project. Then are external helpers including Ph.D. students Haoqi ZHANG whom the author wants to thank. Their treasure suggestions on mechanical design as well as assistance in Gazebo configuration have made a significant contribution to the project. Last but not least, appreciation should also be given to the second marker of this project, Dr. David Forehand, for his helpful advice provided during the interim assessment.

REFERENCES

[1] M. Davids, “A Brief History of Robots in Manufacturing,” ROBOTIQ, 17 July 2017. [Online]. Available: https://blog.robotiq.com/a-brief-history-of-robots-in-manufacturing. [Accessed 19 January 2020].

[2] K. N. J. N. P. Martin Hägele, “Field and Service Robotics,” in *Industrial Robtics*, J. Norberto Pires, 2014.

[3] M. R. *. I. a. M. Sinapius, “A Novel Approach: Combination of Automated Fiber Placement (AFP) and Additive Layer Manufacturing (ALM),” *Composite Science,* vol. 2, no. 10.3390/jcs2030042, p. 9, 2018.

[4] D. GOLDBERG, “History of 3D Printing: It’s Older Than You Are (That Is, If You’re Under 30),” Autodesk, 13 April 2018. [Online]. Available: https://www.autodesk.com/redshift/history-of-3d-printing/. [Accessed 19 January 2020].

[5] T. Ooi, “All3DP,” All3DP, 04 02 2019. [Online]. Available: https://all3dp.com/2/greatest-3d-printing-applications/. [Accessed 30 01 2020].

[6] N. Shahrubudina and R. R. T.C. Lee, “An Overview on 3D Printing Technology: Technological, Materials and Applications,” in *2nd International Conference on Sustainable Materials Processing and Manufacturing*, Batu Pahat, Johor, Malaysia, 2019.

[7] A. Chen, “Top 10 Materials Used For Industrial 3D Printing,” Industries Co-operative Ltd Innovative metal Solution , 12 June 2018. [Online]. Available: https://www.cmac.com.au/blog/top-10-materials-used-industrial-3d-printing. [Accessed 09 January 2020].

[8] S. Béland, “High performance thermoplastic resins and their composites,” *Noyes Data Corp,* 1990.

[9] G. J. a. D. Z. August, “Recent development in automated fibre placement of Thermoplastic composites,” *SAMPE,* vol. 50, pp. 30-37, 2014.

[10] C. librett, “3D Demonstrators Designed for Bigger, Lighter Auto and Aerospace Parts,” Startasys, 24 August 2016. [Online]. Available: http://blog.stratasys.com/2016/08/24/infinite-build-robotic-composite-3d-demonstrator/. [Accessed 19 January 2020].

[11] M. A. N. a. D. P. A.H. Akbarzadeh, “Structural Analysis and Optimization of Moderately-Thick Fiber Steered Plates with Embedded Manufacturing Defects,” *he Second International Symposium on Automated Composites Manufacturing ,* pp. 23-24, April 2015.

[12] L. L. M. E. G. D. a. A. S. M. Barile, “Advanced AFP Applications on Multifunctional Composites,” *The Second International Symposium on Automated Composites Manufacturing ,* pp. 23-24, April 2015.

[13] A. R. B. a. K. K.Senthil, “Defects in composite structures: Its effects and prediction methods–A comprehensive review,” *Composite Structures,* vol. 106, no. 12, pp. 139-149, 2013.

[14] T. S. R. S. a. H. M. E.-D. Z. Qureshi, “n situ consolidation of thermoplastic prepreg tape using automated tape placement technology: Potential and possibilities,” *Composites Part B: Engineering,* vol. 66, pp. 255-267, 2014.

[15] NASA, “NASA,” NASA, 04 08 2015. [Online]. Available: https://www.nasa.gov/centers/marshall/news/news/releases/2015/robotic-manufacturing-system-will-build-biggest-composite-rocket-parts-ever-made.html. [Accessed 30 01 2020].

[16] F. C. a. P. Bidaud, “Closed form solutions for inverse kinematics approximation of general 6R manipulators,” *Mechanism and Machine Theory,* vol. 39, no. 3, pp. 323-338, 2004.

[17] S. Števo, I. Sekaj and M. Dekan, “Optimization of Robotic Arm Trajectory Using Genetic Algorithm,” in

Proceedings of the 19th World Congress - The International Federation of Automatic Control, Cape Town, South Africa, 2014.

[18] J. K. R. D. T. A. Dominik Bertram, "An Integrated Approach to Inverse Kinematics and Path Planning for Redundant Manipulators," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida, 2006.

[19] P. B. M. a. D. K. A. P. Kalra, "An evolutionary approach for solving the multimodal inverse kinematics problem of industrial robots," *Mechanism and Machine Theory,* vol. 41, no. 10, pp. 1213-1229, 2006.

[20] O. Khatib, "A unified approach for motion and force control of," *Journal of Robotics and Automation,* vol. 3, no. 1, p. 43–53, 1987.

[21] W. W. a. H. Elliott, "A computational technique for inverse kinematics," in *Proceedings of the 23rd IEEE Conference on Decision and Control, IEEE*, Las Vegas, Nev, USA, December 1984.

[22] B. S. a. O. E. S. Chiaverini, "Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator," *IEEE Transactions on Control Systems Technology,* vol. 2, no. 2, p. 123–134, 1994.

[23] J. Z. B. L. a. B. L. W. Xu, "Singularity analysis and avoidance for robot manipulators with nonspherical wrists," *IEEE Transactions on Industrial Electronics,,* vol. 63, no. 1, p. 2016, 277–290.

[24] "Markforged Mark Two Print Head," CREATED ADDITIVE MANUFACTURING SOLUTIONS PROVIDER, [Online]. Available: https://www.creat3d.shop/markforged-materials/mark-two-print-head.html. [Accessed 19 03 2020].

[25] M. P. Head, "Markforged 3D Print Stronger," Markforged 3D Print Stronger, 2018. [Online]. Available: https://www.mark3d.com/en/product/spareparts-for-markforged-3d-printers/markforged-print-head-for-mark-two-onyx-nylon/. [Accessed 13 03 2020].

[26] MY3DCONCEPTS.COM, 15 05 2018. [Online]. Available: http://my3dconcepts.com/explore/main-components-of-desktop-3d-printers/. [Accessed 16 03 2020].

[27] T. Anderson, "How Does a 3D Printer Work?," MatterHackers, [Online]. Available: https://www.matterhackers.com/articles/anatomy-of-a-3d-printer. [Accessed 21 03 2020].

[28] "Flexion Extruder Kit - Dual i3 (right side only) (5121SKU002)," SPOOL3D, 2020. [Online]. Available: https://spool3d.ca/flexion-extruder-kit-dual-i3-right-side-only-5121sku002/. [Accessed 19 03 2020].

[29] Panda, "SPECIFICATION Technical specification," October 2018. [Online]. [Accessed 21 03 2020].

[30] "ROS/Instruction," ROS.org, [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 20 03 2020].

[31] Ros.org, "catkin/workspaces," [Online]. Available: http://wiki.ros.org/catkin/workspaces. [Accessed 20 03 2020].

[32] ROS, "ROS Moveit! Quickstart in RViz," [Online]. Available: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html . [Accessed 22 03 2020].

[33] Descartes, "MoveIt," MoveIt, 24 06 2013. [Online]. Available: https://moveit.ros.org/documentation/related_projects/. [Accessed 30 03 2020].

[34] "Plugin Interfaces," MoveIt, [Online]. Available: https://moveit.ros.org/documentation/plugins/#plannermanager. [Accessed 31 03 2020].

[35] MoveIt, “Concepts,” [Online]. Available: https://moveit.ros.org/documentation/concepts/. [Accessed 31 03 2020].

[36] “Developers’ Concepts,” MoveIt, [Online]. Available: https://moveit.ros.org/documentation/concepts/developer_concepts/. [Accessed 01 04 2020].

[37] B. Meijer, “Megatronics Datasheet,” 10 03 2020. [Online]. Available: https://reprapworld.co.uk/datasheets/datasheet%20megatronicsv33.pdf. [Accessed 01 04 2020].

[38] K. Company, “Additive manufacturing and industrial 3D printing,” KUKA , 2019. [Online]. Available: https://www.kuka.com/en-gb/products/process-technologies/2018/12/3d-printing. [Accessed 19 January 2020].

[39] E. Online, “E3D Online,” E3D Online, 2017. [Online]. Available: https://e3d-online.com/titan-aero. [Accessed 06 02 2020].

APPENDICES

CONTENT

Robotic Path Planning Simulation Python Script in RoboDK

```
# Type help("robolink") or help("robodk") for more information
# Press F5 to run the script
# Documentation: https://robodk.com/doc/en/RoboDK-API.html
# Reference:     https://robodk.com/doc/en/PythonAPI/index.html
# Note: It is not required to keep a copy of this file, your python script is saved with the station
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox

RL = Robolink()

# Notify user:
print('To edit this program:\nright click on the Python program, then, select "Edit Python script"')

# Get the robot item:
robot = RL.Item('K UKA KR 10 R900-2')

# Get the home target and the welding targets:
home = RL.Item('Home')
target2 = RL.Item('Target 2')
target3 = RL.Item('Target 3')
target4 = RL.Item('Target 4')
target5 = RL.Item('Target 5')
target6 = RL.Item('Target 6')
target7 = RL.Item('Target 7')
target8 = RL.Item('Target 8')
target9 = RL.Item('Target 9')
target10 = RL.Item('Target 10')
target11 = RL.Item('Target 11')
target12 = RL.Item('Target 12')
target13 = RL.Item('Target 13')
target14 = RL.Item('Target 14')
target15 = RL.Item('Target 15')

#   Get the pose of the target (4x4 matrix):
poseref = target2.Pose()

# Move the robot to home, then to the centre:
robot.MoveJ(home)
robot.MoveJ(target2)
robot.MoveJ(target3)
robot.MoveJ(target4)
robot.MoveJ(target5)
robot.MoveJ(target6)
robot.MoveJ(target7)
robot.MoveJ(target8)
robot.MoveJ(target9)
robot.MoveJ(target10)
robot.MoveJ(target11)
robot.MoveJ(target12)
robot.MoveJ(target13)
robot.MoveJ(target14)
robot.MoveJ(target15)
robot.MoveJ(home)
robot.MoveJ(target4)
robot.MoveJ(target3)
robot.MoveJ(target2)
robot.MoveJ(target14)
robot.MoveJ(target13)
robot.MoveJ(target10)
robot.MoveJ(target6)
robot.MoveJ(target3)
robot.MoveJ(target4)
robot.MoveJ(target5)
robot.MoveJ(target8)
robot.MoveJ(target9)
```

Gazebo 3D Path Planning (Pyramid Geometry)

```
#!/usr/bin/env python

""" Script Writer: Yulin Wang | UUN: S1889038 """
""" Purpose of code: Simulate the trajectory of robotic arm, thereby getting to know Gazebo """

"""

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials provided
    with the distribution.

3. Neither the name of SRI International nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission

"""

""" --------- Step One - Install ROS, Prepare RViz and Moveit Interfaces will be using --------- """

""" To use the Python MoveIt interfaces, 'moveit_commander' _ namespace will be imported first
    This namespace provides us with a 'MoveGroupCommander' _ class, a 'PlanningSceneInterface' _ class,
    and a 'RobotCommander'_ class. More on these below."""
""" --------- Step Two - Import all the required packages and some other messages that will be used """
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
```

```
def all_close(goal, actual, tolerance):
  """
  Convenience method for testing if a list of values are within a tolerance of their counterparts in another list
  @param: goal       A list of floats, a Pose or a PoseStamped
  @param: actual     A list of floats, a Pose or a PoseStamped
  @param: tolerance  A float
  @returns: bool
  """
  all_equal = True
  if type(goal) is list:
    for index in range(len(goal)):
      if abs(actual[index] - goal[index]) > tolerance:
        return False

  elif type(goal) is geometry_msgs.msg.PoseStamped:
    return all_close(goal.pose, actual.pose, tolerance)

  elif type(goal) is geometry_msgs.msg.Pose:
    return all_close(pose_to_list(goal), pose_to_list(actual), tolerance)

  return True


class roboticprintingcontrol(object):
  """ Move the group of joints using python """
  def __init__(self):
    super(roboticprintingcontrol, self).__init__()

    # First initialize 'moveit_commander' and a 'rospy' node:
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('robotic_printing_control', anonymous=True)

    # Instantiate a "RobotCommander" object. Provides information such as the robot's
    # kinematic model and the robot's current joint states, this object is outer-level
    # robot
    robot = moveit_commander.RobotCommander()

    # Instantiate a "PlanningSceneInterface". This provides a remote interface
    # for getting, setting, and updating the robot's internal understanding of
    # the surrounding world:
    scene = moveit_commander.PlanningSceneInterface()

    # Instantiate a "MoveGroupCommander". This object is an interface to a planning
    # group (group of joints). In this project, the robot that we will use is panda
    # therefore, set the group's name to be "panda_arm". If in some certain cases,
    # we are using a different robot, change this value to the name of your robot
    # arm planning group.

    # This interface can be used to plan and execute motions:
    group_name = "panda_arm"
    move_group = moveit_commander.MoveGroupCommander(group_name)

    # Create a "DisplayTrajectory" ROS publisher which is used to visualise
    # trajectories in RVIZ:
    display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
                                                   moveit_msgs.msg.DisplayTrajectory,
                                                   queue_size=20)

    """ --------------------- Getting Basic Information ------------------------- """
    """
    One thing worth noting is that in my system, the version of python is
    2.7, therefore, when print things, ignore parenthesis, otherwise, do
    not forget to add them back
    """
    # Get the name of the reference frame for panda robot
    planning_frame = move_group.get_planning_frame()
    print "============ Planning frame: %s" % planning_frame

    # Print the name of the end-effector link for this group, in this project
    # the end effector is supposed to be a 3D printer head although it has not
    # been completely designed yet
    end_effector = move_group.get_end_effector_link()
    print "============ End effector link: %s" % end_effector

    # Get a list of all the groups in the robot:
    group_names = robot.get_group_names()
    print "============ Available Planning Groups:", robot.get_group_names()

    # For debugging purposes, print the entire state of the
    # robot:
    print "============ Printing robot state"
    print robot.get_current_state()


    # Misc variables
    self.box_name = ''
    self.robot = robot
    self.scene = scene
```

```
    self.move_group = move_group
    self.display_trajectory_publisher = display_trajectory_publisher
    self.planning_frame = planning_frame
    self.eef_link = end_effector
    self.group_names = group_names


def go_to_joint_state(self):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group


    """ ------------------------ Planning to a Joint Goal -------------------------"""
    # The Panda's zero configuration is at a singularity
    # <https://www.quora.com/Robotics-What-is-meant-by-kinematic-singularity>"
    # so the first thing to do is move it to a slightly better configuration.
    # We can get the joint values from the group and adjust some of the values:
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = -pi/4
    joint_goal[2] = -pi/2
    joint_goal[3] = -1.69
    joint_goal[4] = 0
    joint_goal[5] = 2.43
    joint_goal[6] = 0.8

    # Call go command with some pre-set joint values, poses, or without any
    # parameters if you have already set the pose or joint target for the group
    move_group.go(joint_goal, wait=True)

    # Calling 'stop()' ensures that there is no residual movement
    move_group.stop()

    # For testing:
    current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)

# Define a class named as move_group

def go_to_pose_goal(self):
    move_group = self.move_group
```

```
        """ -------------------------- Planning to a Pose Goal ------------------------ """
        # Plan a motion for this group to a desired pose for the
        # end-effector - printing head
        # For any three-dimensional space, it requires a 6-degree os freedom
        # of specification, to uniquely identify a pose, namely, in this case, position
        # x, y and z as well as orientation including roll, pitch and yaw

        # In order to make it easier for computers to understanad, Quaternions for
        # rotation is applied using four numbers of vectors [x y z w],
        # http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/
        pose_goal = geometry_msgs.msg.Pose()
        pose_goal.orientation.w = 1.0      # Angle of rotation
        # Vectors representing axis of rotation
        pose_goal.position.x = 0.28
        pose_goal.position.y = -0.7
        pose_goal.position.z = 1.0

        move_group.set_pose_target(pose_goal)

        ## Now, we call the planner to compute the plan and execute it.
        plan = move_group.go(wait=True)
        # Calling 'stop()' to ensure that there is no residual movement.
        move_group.stop()
        # It is always good to clear your targets after planning with poses.
        move_group.clear_pose_targets()

        # Similar to before, the following code is for testing:
        # we use the class variable rather than the copied state variable
        current_pose = self.move_group.get_current_pose().pose
        return all_close(pose_goal, current_pose, 0.01)


# Most important Section
# Plan a Cartesian path by specifying a list of
# waypoints for the end-effector to go through

 #
    def plan_cartesian_path(self, scale=1):
        move_group = self.move_group

        # Cartesian Paths
        # Plan a Cartesian path directly by specifying a list of waypoints
        # for the end-effector to go through. If executing interactively in a
        # Python shell, set scale = 1.0.
# Build A list called waypoints to contain the points of robotic arm
waypoints = []

# Now we define a list containing all the target positional points
# the first target point is to move z axis first, and in this case,
# we get the initial position of the robotic arm
wpose = move_group.get_current_pose().pose
# In this section, the case study that we focus is a pyramid geometry
# First move up (z) to the initial point
wpose.position.z -= scale * 0.3
waypoints.append(copy.deepcopy(wpose))
# after which we move the sideways in y direction, thereafter, x direction
""" ------ print the base layer --------- """
 wpose.position.y += scale * 0.25
waypoints.append(copy.deepcopy(wpose))

wpose.position.x -= scale * 0.25
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.25
waypoints.append(copy.deepcopy(wpose))

wpose.position.x += scale * 0.20
waypoints.append(copy.deepcopy(wpose))

""" -------- print the second layer from the base layer   ---------- """
 wpose.position.z += scale * 0.05
waypoints.append(copy.deepcopy(wpose))

wpose.position.x -= scale * 0.2
waypoints.append(copy.deepcopy(wpose))

wpose.position.y += scale * 0.2
waypoints.append(copy.deepcopy(wpose))

wpose.position.x += scale * 0.2
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.16
waypoints.append(copy.deepcopy(wpose))

""" -------- print the second layer from the base layer   ---------- """
 wpose.position.z += scale * 0.05
```

```
waypoints.append(copy.deepcopy(wpose))

wpose.position.x -= scale * 0.16
waypoints.append(copy.deepcopy(wpose))

wpose.position.y += scale * 0.16
waypoints.append(copy.deepcopy(wpose))

wpose.position.x += scale * 0.16
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.12
waypoints.append(copy.deepcopy(wpose))

""" -------------- Print the third layer ---------------- """
 wpose.position.z += scale * 0.05
waypoints.append(copy.deepcopy(wpose))

wpose.position.x -= scale * 0.12
waypoints.append(copy.deepcopy(wpose))

wpose.position.y += scale * 0.12
waypoints.append(copy.deepcopy(wpose))

wpose.position.x += scale * 0.12
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.08
waypoints.append(copy.deepcopy(wpose))

""" -------------- Print the fourth layer ---------------- """
 wpose.position.z += scale * 0.05
waypoints.append(copy.deepcopy(wpose))

wpose.position.x -= scale * 0.08
waypoints.append(copy.deepcopy(wpose))

wpose.position.y += scale * 0.08
waypoints.append(copy.deepcopy(wpose))

wpose.position.x += scale * 0.08
waypoints.append(copy.deepcopy(wpose))

wpose.position.y -= scale * 0.04
```

```
    waypoints.append(copy.deepcopy(wpose))



    # As Cartesian path is to be interpolated at a resolution of 1 cm which is
    # why we will specify 0.01 as the end_effector_translation in Cartesian
    # translation.

    # compute_cartesian_path is used to compute the joint trajectory automatically
    (plan, fraction) = move_group.compute_cartesian_path(
                                    waypoints,       # waypoints to follow
                                    0.01,            # end_effector_step
                                    0.0)             # jump_threshold

    # Note: This is just the planning procedure rather than
    # asking move_group to actually move the robot yet

    return plan, fraction


  def display_trajectory(self, plan):

    robot = self.robot
    display_trajectory_publisher = self.display_trajectory_publisher

    # Displaying a Trajectory
    # Now, ask RViz to visualize a plan (aka trajectory). But the
    # group.plan() method does this automatically so this is not that useful
    # here (it just displays the same trajectory again):

    # A 'DisplayTrajectory' msg has two primary fields, trajectory_start and trajectory.
    # We populate the trajectory_start with our current robot state to copy over
    # any AttachedCollisionObjects and add our plan to the trajectory.
    # Display trajectories
    display_trajectory = moveit_msgs.msg.DisplayTrajectory()
    display_trajectory.trajectory_start = robot.get_current_state()
    display_trajectory.trajectory.append(plan)

    display_trajectory_publisher.publish(display_trajectory);


  def execute_plan(self, plan):

      move_group = self.move_group
      move_group.execute(plan, wait=True)

    # Note that the robot's current joint state must be within some tolerance of the
    # first waypoint in the 'RobotTrajectory', and otherwise 'execute()' will fail



  def wait_for_state_update(self, box_is_known=False, box_is_attached=False, timeout=4):

    box_name = self.box_name
    scene = self.scene

    # If the Python node dies before publishing a collision object update message, the message
    # could get lost and the box will not appear. To ensure that the updates are made, we wait
    # until we see the changes reflected in the 'get_attached_objects()' and 'get_known_object_names()' lists.

    start = rospy.get_time()
    seconds = rospy.get_time()
    while (seconds - start < timeout) and not rospy.is_shutdown():
      # Test if the box is in attached objects
      attached_objects = scene.get_attached_objects([box_name])
      is_attached = len(attached_objects.keys()) > 0

      # Test if the box is in the scene.
      # Note that attaching the box will remove it from known_objects
      is_known = box_name in scene.get_known_object_names()

      # Test if we are in the expected state
      if (box_is_attached == is_attached) and (box_is_known == is_known):
        return True

      # Sleep so that we give other threads time on the processor
      rospy.sleep(0.1)
      seconds = rospy.get_time()

    # If we exited the while loop without returning then we timed out
    return False


"""-------------------------------------------------------------------------------------------"""

def main():
  try:
    print "-------------------------------------------------------"
    print "----------------------------------------------------------"
    print "Welcome to the MoveIt MoveGroup Python Interface - Yulin's BEng Project"
    print "----------------------------------------------------------"
    print "Press Ctrl-D to exit at any time"
    print "----------------------------------------------------------"
    print "========= Press 'Enter' to begin the simulation by setting up the moveit_commander =========="
    raw_input()
    tutorial = roboticprintingcontrol()

    print "========= Press 'Enter'   to execute a movement using a joint state goal ========="
    raw_input()
    tutorial.go_to_joint_state()

    print "======== Press 'Enter' to plan and display a Cartesian path ========"
    raw_input()
    cartesian_plan, fraction = tutorial.plan_cartesian_path()

    print "======== Press 'Enter'   to display a saved trajectory (this will replay the Cartesian path)
========"
    raw_input()
    tutorial.display_trajectory(cartesian_plan)

    print "======== Press 'Enter'   to execute a saved path ========"
    raw_input()
    tutorial.execute_plan(cartesian_plan)

    #print "======== Press 'Enter' to attach an end effector to help illustrate trajectory ========"
    #raw_input()
    #tutorial.attach_box()

    #print "======== Press 'Enter' to execute a path with attached end-effector ========"
    #raw_input()
    #cartesian_plan, fraction = tutorial.plan_cartesian_path(scale=-1)
    #tutorial.execute_plan(cartesian_plan)

    #print "======== Press 'Enter' to detach the end-effector from the robot ========"
    #raw_input()
    #tutorial.detach_box()

    #print "======== Press 'Enter' to remove the end-effector "
    #print "======== Press 'Enter' to remove the end-effector from the planning scene ========"
    #raw_input()
    #tutorial.remove_box()
    print "======== Done, This is the end of 3D robotic printing simulation - Author: Yulin Wang !! ========"
  except rospy.ROSInterruptException:
    return
  except KeyboardInterrupt:
    return

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

Gazebo 2D Shape Path Planning

```
#!/usr/bin/env python

""" Script Writer: Yulin Wang | UUN: S1889038 """
""" - this script controls the robot to draw a curve in 2D"""
""" Purpose of code: Simulate the trajectory of robotic arm, thereby getting to know Gazebo """

"""

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials provided
    with the distribution.

3. Neither the name of SRI International nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission

"""

""" --------- Step One - Install ROS, Prepare RViz and Moveit Interfaces will be using --------- """

""" To use the Python MoveIt interfaces, 'moveit_commander' _ namespace will be imported first
    This namespace provides us with a 'MoveGroupCommander' _ class, a 'PlanningSceneInterface' _ class,
    and a 'RobotCommander'_ class. More on these below."""
""" --------- Step Two - Import all the required packages and some other messages that will be used """
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
```

```
def all_close(goal, actual, tolerance):
  """
  Convenience method for testing if a list of values are within a tolerance of their counterparts in another list
  @param: goal        A list of floats, a Pose or a PoseStamped
  @param: actual      A list of floats, a Pose or a PoseStamped
  @param: tolerance   A float
  @returns: bool
  """
  all_equal = True
  if type(goal) is list:
    for index in range(len(goal)):
      if abs(actual[index] - goal[index]) > tolerance:
        return False

  elif type(goal) is geometry_msgs.msg.PoseStamped:
    return all_close(goal.pose, actual.pose, tolerance)

  elif type(goal) is geometry_msgs.msg.Pose:
    return all_close(pose_to_list(goal), pose_to_list(actual), tolerance)

  return True


class roboticprintingcontrol(object):
  """ Move the group of joints using python """
  def __init__(self):
    super(roboticprintingcontrol, self).__init__()

    # First initialize 'moveit_commander' and a 'rospy' node:
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('robotic_printing_control', anonymous=True)

    # Instantiate a "RobotCommander" object. Provides information such as the robot's
    # kinematic model and the robot's current joint states, this object is outer-level
    # robot
    robot = moveit_commander.RobotCommander()

    # Instantiate a "PlanningSceneInterface". This provides a remote interface
    # for getting, setting, and updating the robot's internal understanding of
    # the surrounding world:
    scene = moveit_commander.PlanningSceneInterface()
```

```
# Instantiate a "MoveGroupCommander". This object is an interface to a planning
# group (group of joints). In this project, the robot that we will use is panda
# therefore, set the group's name to be "panda_arm". If in some certain cases,
# we are using a different robot, change this value to the name of your robot
# arm planning group.

# This interface can be used to plan and execute motions:
group_name = "panda_arm"
 move_group = moveit_commander.MoveGroupCommander(group_name)

# Create a "DisplayTrajectory" ROS publisher which is used to visualise
# trajectories in RVIZ:
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
                                               moveit_msgs.msg.DisplayTrajectory,
                                               queue_size=20)

""" --------------------- Getting Basic Information ------------------------- """
"""
One thing worth noting is that in my system, the version of python is
2.7, therefore, when print things, ignore parenthesis, otherwise, do
not forget to add them back
"""
# Get the name of the reference frame for panda robot
planning_frame = move_group.get_planning_frame()
print "============ Planning frame: %s" % planning_frame

# Print the name of the end-effector link for this group, in this project
# the end effector is supposed to be a 3D printer head although it has not
# been completely designed yet
end_effector = move_group.get_end_effector_link()
print "============ End effector link: %s" % end_effector

# Get a list of all the groups in the robot:
group_names = robot.get_group_names()
print "============ Available Planning Groups:", robot.get_group_names()

# For debugging purposes, print the entire state of the
# robot:
print "============ Printing robot state"
 print robot.get_current_state()


# Misc variables
self.box_name = ''
```

```
    self.robot = robot
    self.scene = scene
    self.move_group = move_group
    self.display_trajectory_publisher = display_trajectory_publisher
    self.planning_frame = planning_frame
    self.eef_link = end_effector
    self.group_names = group_names


def go_to_joint_state(self):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    move_group = self.move_group


    """ ------------------------------- Planning to a Joint Goal ------------------------------ """
    # The Panda's zero configuration is at a singularity
    # <https://www.quora.com/Robotics-What-is-meant-by-kinematic-singularity>"
    # so the first thing to do is move it to a slightly better configuration.
    # We can get the joint values from the group and adjust some of the values:
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = -pi/4
    joint_goal[2] = -pi/2
    joint_goal[3] = -1.69
    joint_goal[4] = 0
    joint_goal[5] = 2.43
    joint_goal[6] = 0.8

    # Call go command with some pre-set joint values, poses, or without any
    # parameters if you have already set the pose or joint target for the group
    move_group.go(joint_goal, wait=True)

    # Calling 'stop()' ensures that there is no residual movement
    move_group.stop()

    # For testing:
    current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)

# Define a class named as move_group

def go_to_pose_goal(self):
        move_group = self.move_group

        """ ------------------------------ Planning to a Pose Goal ----------------------------- """
        # Plan a motion for this group to a desired pose for the
        # end-effector - printing head
        # For any three-dimensional space, it requires a 6-degree os freedom
        # of specification, to uniquely identify a pose, namely, in this case, position
        # x, y and z as well as orientation including roll, pitch and yaw

        # In order to make it easier for computers to understanad, Quaternions for
        # rotation is applied using four numbers of vectors [x y z w],
        # http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/
        pose_goal = geometry_msgs.msg.Pose()
        pose_goal.orientation.w = 1.0      # Angle of rotation
        # Vectors representing axis of rotation
        pose_goal.position.x = 0.28
        pose_goal.position.y = -0.7
        pose_goal.position.z = 1.0

        move_group.set_pose_target(pose_goal)

        ## Now, we call the planner to compute the plan and execute it.
        plan = move_group.go(wait=True)
        # Calling 'stop()' to ensure that there is no residual movement.
        move_group.stop()
        # It is always good to clear your targets after planning with poses.
        move_group.clear_pose_targets()

        # Similar to before, the following code is for testing:
        # we use the class variable rather than the copied state variable
        current_pose = self.move_group.get_current_pose().pose
        return all_close(pose_goal, current_pose, 0.01)


# Most important Section
# Plan a Cartesian path by specifying a list of
# waypoints for the end-effector to go through

  #
    def plan_cartesian_path(self, scale=1):
        move_group = self.move_group

        # Cartesian Paths
        # Plan a Cartesian path directly by specifying a list of waypoints
    # for the end-effector to go through. If executing interactively in a
    # Python shell, set scale = 1.0.

    # Build A list called waypoints to contain the points of robotic arm
    waypoints = []

    # Now we define a list containing all the target positional points
    # the first target point is to move z axis first, and in this case,
    # we get the initial position of the robotic arm
    wpose = move_group.get_current_pose().pose
    # start from the simple geometry, say a rectangle
    # First move up (z)
    wpose.position.z -= scale * 0.5
    waypoints.append(copy.deepcopy(wpose))
    # after which we move the sideways in y direction
    wpose.position.y += scale * 0.4
    waypoints.append(copy.deepcopy(wpose))
    # Second move forward/backwards in z direction
    wpose.position.z += scale * 0.5
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.y -= scale * 0.4    # Third move sideways (y)
    waypoints.append(copy.deepcopy(wpose))

    # As Cartesian path is to be interpolated at a resolution of 1 cm which is
    # why we will specify 0.01 as the end_effector_translation in Cartesian
    # translation.

    # compute_cartesian_path is used to compute the joint trajectory automatically
    (plan, fraction) = move_group.compute_cartesian_path(
                                       waypoints,     # waypoints to follow
                                       0.01,          # end_effector_step
                                       0.0)           # jump_threshold

    # Note: This is just the planning procedure rather than
    # asking move_group to actually move the robot yet

    return plan, fraction


def display_trajectory(self, plan):

    robot = self.robot
    display_trajectory_publisher = self.display_trajectory_publisher
    # Displaying a Trajectory
    # Now, ask RViz to visualize a plan (aka trajectory). But the
    # group.plan() method does this automatically so this is not that useful
    # here (it just displays the same trajectory again):

    # A 'DisplayTrajectory' msg has two primary fields, trajectory_start and trajectory.
    # We populate the trajectory_start with our current robot state to copy over
    # any AttachedCollisionObjects and add our plan to the trajectory.
    # Display trajectories
    display_trajectory = moveit_msgs.msg.DisplayTrajectory()
    display_trajectory.trajectory_start = robot.get_current_state()
    display_trajectory.trajectory.append(plan)

    display_trajectory_publisher.publish(display_trajectory);


  def execute_plan(self, plan):

      move_group = self.move_group
      move_group.execute(plan, wait=True)

    # Note that the robot's current joint state must be within some tolerance of the
    # first waypoint in the 'RobotTrajectory', and otherwise 'execute()' will fail



  def wait_for_state_update(self, box_is_known=False, box_is_attached=False, timeout=4):

    box_name = self.box_name
    scene = self.scene

    # If the Python node dies before publishing a collision object update message, the message
    # could get lost and the box will not appear. To ensure that the updates are made, we wait
    # until we see the changes reflected in the 'get_attached_objects()' and 'get_known_object_names()' lists.

    start = rospy.get_time()
    seconds = rospy.get_time()
    while (seconds - start < timeout) and not rospy.is_shutdown():
      # Test if the box is in attached objects
      attached_objects = scene.get_attached_objects([box_name])
      is_attached = len(attached_objects.keys()) > 0

      # Test if the box is in the scene.
```

```
        # Note that attaching the box will remove it from known_objects
        is_known = box_name in scene.get_known_object_names()

        # Test if we are in the expected state
        if (box_is_attached == is_attached) and (box_is_known == is_known):
          return True

        # Sleep so that we give other threads time on the processor
        rospy.sleep(0.1)
        seconds = rospy.get_time()

      # If we exited the while loop without returning then we timed out
      return False

"""-----------------------------------------------------------------------------------------------"""

def main():
  try:
    print "----------------------------------------------------------"
    print "----------------------------------------------------------"
    print "Welcome to the MoveIt MoveGroup Python Interface - Yulin's BEng Project"
    print "----------------------------------------------------------"
    print "Press Ctrl-D to exit at any time"
    print "----------------------------------------------------------"
    print "========= Press 'Enter' to begin the simulation by setting up the moveit_commander =========="
    raw_input()
    tutorial = roboticprintingcontrol()

    print "========= Press 'Enter'   to execute a movement using a joint state goal ========="
    raw_input()
    tutorial.go_to_joint_state()

    print "======== Press 'Enter'   to execute a movement using a pose goal ========"
    raw_input()
    tutorial.go_to_pose_goal()

    print "======== Press 'Enter' to plan and display a Cartesian path ========"
    raw_input()
    cartesian_plan, fraction = tutorial.plan_cartesian_path()

    print "======== Press 'Enter'   to display a saved trajectory (this will replay the Cartesian path)
========"
    raw_input()
    tutorial.display_trajectory(cartesian_plan)
```

```
      print "======== Press 'Enter'   to execute a saved path ========"
       raw_input()
      tutorial.execute_plan(cartesian_plan)

      print "======== Done, Nail it!! ========"
    except rospy.ROSInterruptException:
      return
    except KeyboardInterrupt:
      return

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

School of Engineering – Incident Management

Project Status Declaration

This form is to be used in unforeseen circumstances necessitating the immediate cessation of practical project work during semester. It acts as a record of the current status of practical work (whether it be laboratory based, computational, or fieldwork). Due to circumstances, ***no further practical work is to be continued, regardless of the type of work or current status***. This ensures equality of opportunity for all students, regardless of the type of work being undertaken.

The form must be completed during a meeting with the student, and verified by the student, supervisor, and either the thesis examiner, or a second supervisor. **Any practical work beyond that stated in this form will not be considered in the final project assessment.**

A copy of the signed form must be included in the final project submission.

Name: Yulin Wang ____________________________ Student number: S1889038_______________

Work completed

All items of wholly, or partially completed work must be listed, indicating the percentage completion for each task. Reference can be made to an attached project plan if appropriate. **Please take care to provide a full detailed list of all work done.**

1. Literature Review of current state-of-the-art robotic printing techniques, 3D printer head design, robotic manufacturing technologies and robotic control software (initially use RoboDk but later changed to Gazebo due to the suggestion from another professor in school of informatics) – **Percentage of initial determined target completion – 100%**

2. Robot trajectory simulation in Gazebo including robot configuration setup, using GUI to control the robot path and use Python-Gazebo to control trajectory. **– Percentage of initial determined target completion – 100%**

3. Full CAD assembly design based on currently existing design of 3D printer head without Cutter – Percentage of initial determined target completion **– 90% (the cutter has not and will not be designed and fitted into the printer head in my project)**

4. Arduino Board configurations, for example, in/outputs and understand the configuration what each output connects to such as temperature control, heat control, etc. – Percentage of initial determined target completion **– 50% (did not finish the coding and practically testing part)**

5. Project management – make and update a plan on a weekly basis using Gantt Chart. Keep progressing my project and regular meeting with supervisors. **– Percentage of initial determined target completion – 100%**

Work not commenced

Any items of outstanding work that have not been started should be listed here.

1. Arduino board programming and practical testing (lack of wire to connect the Arduino Board to my PC which blocks me from coding and testing on Arduino IDE software)

2. The cutter design of 3D printer head due to the high complexity of fitting the cutter into the printer head

3. Deeper analysis, namely, adding an end effector on the tip of robotic arm to further visualise the printer head trajectory, will be applied on my robotic arm simulation – Date of finish: March 21st, 2020 (For this part, it is not quite necessary and was not in the originally determined plan, that is why in the previous section, the percentage of completion of Gazebo simulation is considered to be 100% finished)

4. Also, in terms of the CAD design, the critical components in the assembly will be explained with an introduction of their functions, and if possible, a cutter design will be fitted into the CAD (Optional) – March 26th , 2020

Plans for completing project submission

State revised plans for producing the final project submission in the absence of any additional practical work beyond that already listed. For example, this may include literature-based research, or more in-depth analysis of results already obtained. Dates for completion of each element should be given.

The majority of the project is based on computer, and most of (≥90%) the initial determined targets from both technical and project management point of view including robotic arm control in Gazebo, design of 3D printer head without cutter using CAD, Gantt Chart and understanding the Adruino Board configuration have been finished on computers before the release of the contingency plan.

In the remaining days, the majority of attention will be put on warping up and incorporating the information into my report.

Although it might not be possible for me to code on Arduino and practically test the performance of the Arduino board, I will try to write a literature-based paragraph of the Arduino board in my final report in replace of the originally-planned actual programming on Arduino board.
As far as the electronics part, as aforementioned, a literature-based description about the Arduino board configuration will be given in the report – March 30th , 2020
Finally, I will be starting to write my final report at the beginning of April although I have started a little bit already, and I believe the remaining days should be very enough for me to finish the report in a timely way without compromising too many qualities despite of this situation.

Declaration

To the best of our knowledge, this form is an accurate record of the project status and revised completion plans on March 19th, 2020 ______________ (date)

Student: Yulin Wang____(Signature)

Supervisor: _________ [illegible] ____________________________________

Second sup./Thesis examiner: David I. M. Forehand 19/03/2020

Covid-19 Impact Statement

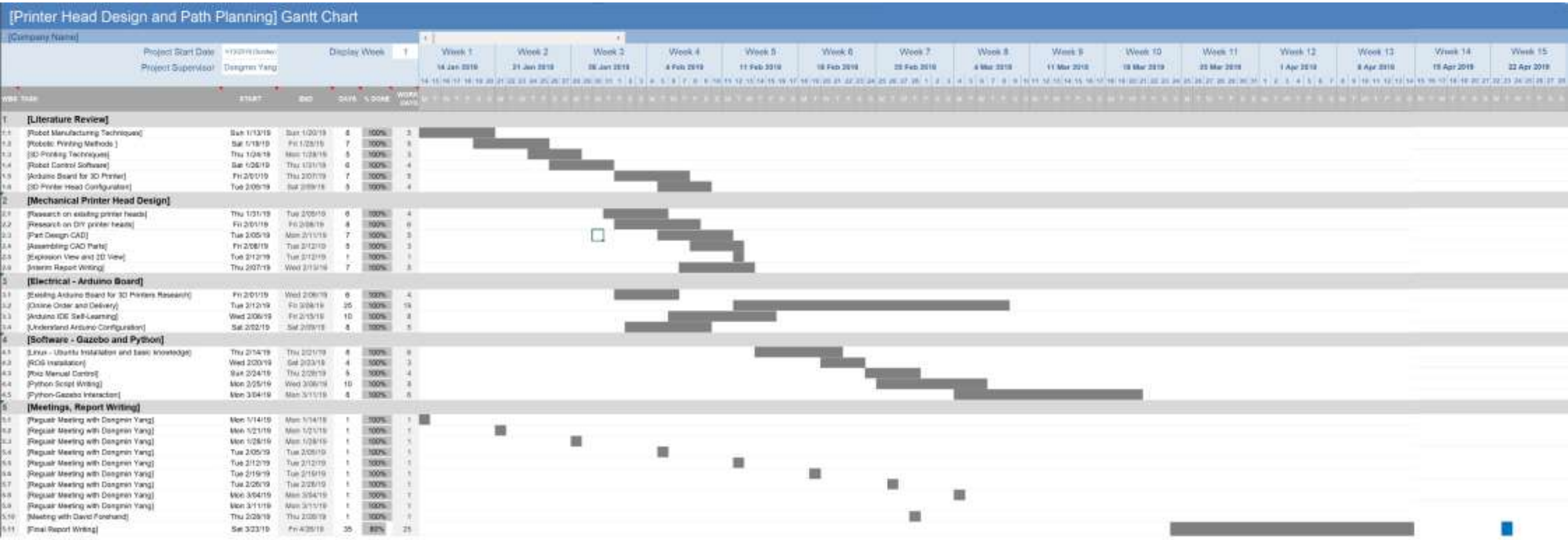
Due to the current covid-19 issue, there are two main aspects of my project that have been strongly impacted.

First and foremost, in terms of the technical impacts, the practical lab work (Arduino board electrical test) was not able to do. Further to this, as the project was terminated all of a sudden, it leaves me barely any time to finish the software section, particularly in adding end-effectors to help visualise the trajectory.

Besides the technical impacts, the other inevitable impact was project management – Gantt Chart. As the project was stopped in a relatively short period of time, the original project management plan has to be modified, which concomitantly leads to a different project timeline from the one set initially.

Student: Yulin Wang

Date: April 28, 2020



[Printer Head Design and Path Planning] Gantt Chart

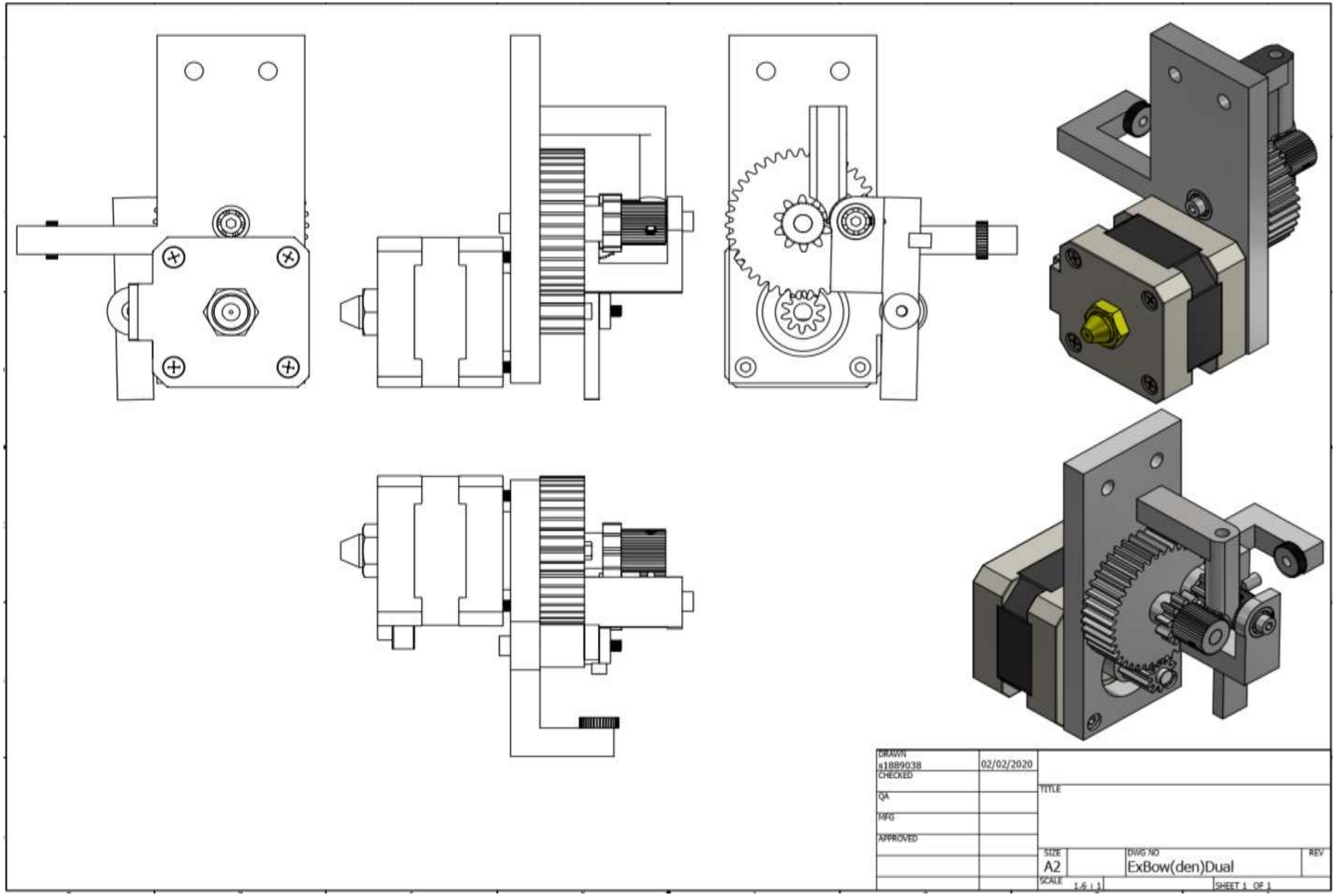
[Company Name]

Project Start Date: 1/13/2019 (Sunday) Display Week: 1

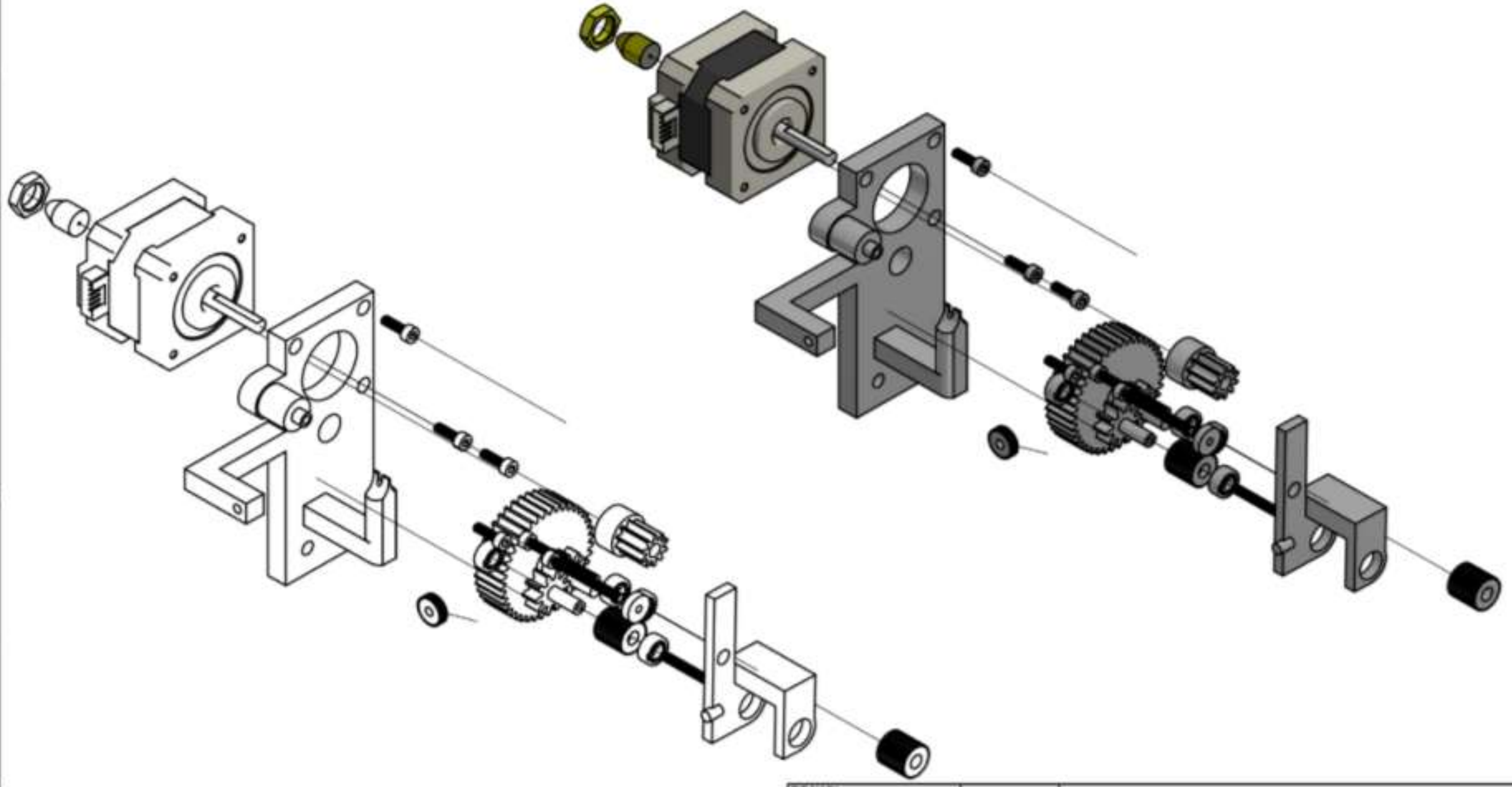
Project Supervisor: Dongmin Yang

Week 1 14 Jan 2019 | Week 2 21 Jan 2019 | Week 3 28 Jan 2019 | Week 4 4 Feb 2019 | Week 5 11 Feb 2019 | Week 6 18 Feb 2019 | Week 7 25 Feb 2019 | Week 8 4 Mar 2019 | Week 9 11 Mar 2019 | Week 10 18 Mar 2019 | Week 11 25 Mar 2019 | Week 12 1 Apr 2019 | Week 13 8 Apr 2019 | Week 14 15 Apr 2019 | Week 15 22 Apr 2019

WBS	TASK	START	END	DAYS	% DONE	WORK DAYS
1	**[Literature Review]**					
1.1	[Robot Manufacturing Techniques]	Sun 1/13/19	Sun 1/20/19	8	100%	5
1.2	[Robotic Printing Methods]	Sat 1/19/19	Fri 1/25/19	7	100%	5
1.3	[3D Printing Techniques]	Thu 1/24/19	Mon 1/28/19	5	100%	3
1.4	[Robot Control Software]	Sat 1/26/19	Thu 1/31/19	6	100%	4
1.5	[Arduino Board for 3D Printer]	Fri 2/01/19	Thu 2/07/19	7	100%	5
1.6	[3D Printer Head Configuration]	Tue 2/05/19	Sat 2/09/19	5	100%	4
2	**[Mechanical Printer Head Design]**					
2.1	[Research on existing printer heads]	Thu 1/31/19	Tue 2/05/19	6	100%	4
2.2	[Research on DIY printer heads]	Fri 2/01/19	Fri 2/08/19	8	100%	6
2.3	[Part Design CAD]	Tue 2/05/19	Mon 2/11/19	7	100%	5
2.4	[Assembling CAD Parts]	Fri 2/08/19	Tue 2/12/19	5	100%	3
2.5	[Explosion View and 2D View]	Tue 2/12/19	Tue 2/12/19	1	100%	1
2.6	[Interim Report Writing]	Thu 2/07/19	Wed 2/13/19	7	100%	5
3	**[Electrical - Arduino Board]**					
3.1	[Existing Arduino Board for 3D Printers Research]	Fri 2/01/19	Wed 2/06/19	6	100%	4
3.2	[Online Order and Delivery]	Tue 2/12/19	Fri 3/08/19	25	100%	19
3.3	[Arduino IDE Self-Learning]	Wed 2/06/19	Fri 2/15/19	10	100%	8
3.4	[Understand Arduino Configuration]	Sat 2/02/19	Sat 2/09/19	8	100%	5
4	**[Software - Gazebo and Python]**					
4.1	[Linux - Ubuntu Installation and basic knowledge]	Thu 2/14/19	Thu 2/21/19	8	100%	6
4.2	[ROS Installation]	Wed 2/20/19	Sat 2/23/19	4	100%	3
4.3	[Rviz Manual Control]	Sun 2/24/19	Thu 2/28/19	5	100%	4
4.4	[Python Script Writing]	Mon 2/25/19	Wed 3/06/19	10	100%	8
4.5	[Python-Gazebo Interaction]	Mon 3/04/19	Mon 3/11/19	8	100%	6
5	**[Meetings, Report Writing]**					
5.1	[Reguair Meeting with Dongmin Yang]	Mon 1/14/19	Mon 1/14/19	1	100%	1
5.2	[Reguair Meeting with Dongmin Yang]	Mon 1/21/19	Mon 1/21/19	1	100%	1
5.3	[Reguair Meeting with Dongmin Yang]	Mon 1/28/19	Mon 1/28/19	1	100%	1
5.4	[Reguair Meeting with Dongmin Yang]	Tue 2/05/19	Tue 2/05/19	1	100%	1
5.5	[Reguair Meeting with Dongmin Yang]	Tue 2/12/19	Tue 2/12/19	1	100%	1
5.6	[Reguair Meeting with Dongmin Yang]	Tue 2/19/19	Tue 2/19/19	1	100%	1
5.7	[Reguair Meeting with Dongmin Yang]	Tue 2/26/19	Tue 2/26/19	1	100%	1
5.8	[Reguair Meeting with Dongmin Yang]	Mon 3/04/19	Mon 3/04/19	1	100%	1
5.9	[Reguair Meeting with Dongmin Yang]	Mon 3/11/19	Mon 3/11/19	1	100%	1
5.10	[Meeting with David Forehand]	Thu 2/28/19	Thu 2/28/19	1	100%	1
5.11	[Final Report Writing]	Sat 3/23/19	Fri 4/26/19	35	80%	25



DRAWN s1889038	02/02/2020	
CHECKED		TITLE
QA		
MFG		
APPROVED		
		SIZE A2 · DWG NO ExBow(den)Dual · REV
		SCALE 1.6 : 1 · SHEET 1 OF 1



DRAWN julia	2020/2/6			
CHECKED		TITLE		
QA				
MFG				
APPROVED				
		SIZE A3	DWG NO ExBow(den)Dual	REV
		SCALE 1 : 1.25		SHEET 1 OF 1