NTNU

Norwegian University of
Science and Technology

Path Correction for 3D Printing by Robotic Manipulator

Nora Leiva Garcia

Embedded Computing Systems
Submission date: July 2018
Supervisor: Jan Tommy Gravdahl, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Summary

Technology is an essential part of industry nowadays, thanks to which there are advances in quality of the product, efficiency of energy or safety of human capital (among others). Today, we are part of the fourth industrial revolution, where mass production is being replaced by customized production and Additive Manufacturing (group of technologies that build 3D objects by adding material layer-by-layer) and Robotics (an interdisciplinay branch of engineering and science that deals with the design, construction, operation and use of robots) are becoming leading technologies.

The primary objective of this work is to correct the path of a 6 Degrees Of Freedom (DOF) robotic arm with 3D printing purposes. Improving the error detection in real time can make the usage of robotic arms in 3D printing grow noticeably, as a result of making them as efficient and useful in industry as possible (which is the main motivation of this research).

A piece of code will be developed in RAPID (programming language used in ABB robots) in order to correct two important errors that occur while printing. One of the errors comes linked to the use of materials that expand or contract after being deposited (insulation foam for example), as this change in size affect to the final dimensions of the product, and more specifically to the height of each layer printed. The other is the path deviation in straight lines, where an iterative correction (ILC control method) layer-by-layer is needed until the good coordinates are reached.

To test the effectiveness of the code, different experiments will be carried out in Robotstudio (ABB software where code can be tested using a Virtual Controller and a replica of the real robotic station) as well as in the IRB140 (real robot at the laboratory). During the experiments, different data will be obtained and exported to a text file. After that, the information will be reorganized in EXCEL and used in MATLAB to obtain some graphs. At the end, the performance of the code will be evaluated by studying the graphs and comparing the results obtained in Robotstudio with those obtained when using the robot IRB140.

Once the experiments have been done and all the information obtained from them has been analyzed, a discussion will be done on whether the errors have been corrected or not as well as on further work that could be done on this area.

The results show that the errors have been corrected after applying the code developed in this project; and moreover, the influence of the difference variables has been tested to be the one expected, except for one case.

Preface

This master thesis is submitted as part of the requirements for the Erasmus+ program at the Norwegian University of Science and Technology (NTNU), in order to fulfil the demands of the Master's degree in Industrial Engineering at the University of the Basque Country. It has been written in cooperation with the department of Engineering Cybernetics and NTNU, together with the Professor Jan Tommy Gravdahl and in the field of 3D printing by robot.

First of all, I would like to thank my supervisor Professor Jan Tommy Gravdahl for giving me the opportunity to do the thesis with him, as well as for all the guidance and support throughout the semester. I would also like to thank my co-supervisor Linn Danielsen Evjemo, a PhD graduate from the Department of Engineering Cybernetics, for the time spent sharing her knowledge on 3D printing by robotic manipulators with me.

Apart from that, I would like to show my gratitude to the NTNU in general, for the warm welcome; and to the department of Engineering Cybernetics in particular, for allowing me to use all its facilities (such as the laboratory) and resources (giving me a personal computer with all the software (SW) I needed).

A special mention goes to my family, for their support and unconditional motivation, even being thousands of kilometres away one from each other. Finally, I would like to thank the people I have met during my stay in Norway, for making it exceptional.

Nora Leiva Garcia

July 2018

Table of Contents

List of Tables

List of Figures

Abbreviations

NTNU	=	Norwegian University of Science and Technology
IoT	=	Internet of Things
AM	=	Additive Manufacturing
ILC	=	Iterative Learning Control
CS	=	Coordinate System
TCP	=	Tool Center Point
SW	=	Software

Chapter 1

Introduction

This chapter is devoted to the introduction of the general information of this thesis report and it is divided as follows:

- **Background information:** A little presentation of the different technologies that surround this research.
- **Motivation:** Explanation of the different reasons that made this research standout from others when choosing the topic of the master thesis.
- **Objectives:** Presentation of the different objectives of this master thesis.
- **Related work:** Introduce to the reader what has already been done in connection to this topic.
- **Report structure:** Little summary of each of the chapters of the report.

1.1 Background information

Technology is an essential part of industry nowadays, thanks to which there are advances in quality of the product, efficiency of energy or safety of human capital (among others). The world is now entering a new era, called Industry 4.0, or the fourth industrial revolution. Going back in history, the first industrial revolution arrived at the end of the 18th century with steam engines and the first machines, introducing the mechanization in some of the processes. A century later, mass production was established thanks to electricity, resulting in the second industrial revolution. The third industrial revolution came at the end of the 20th century, with the emergence of computers and the first steps into automation (Technology by which a process or procedure is performed without human assistance [5]).

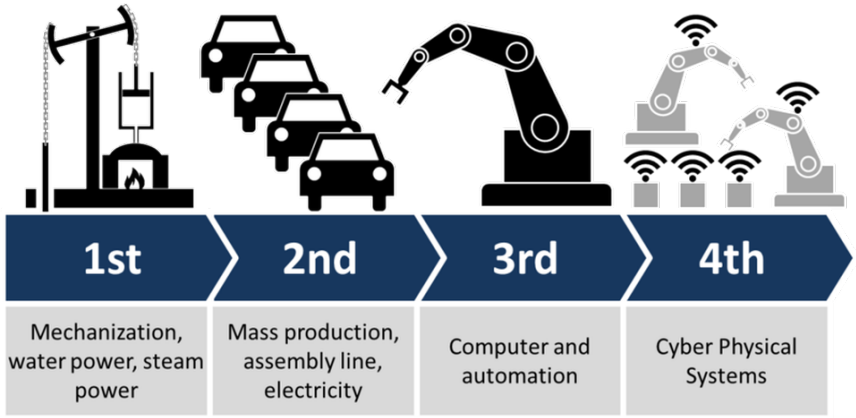


Figure 1.1: The four Industrial Revolutions(by Cristph Roser at AllAboutLean.com)

Today, we are part of the fourth industrial revolution or Industry 4.0, where mass production is being replaced by customized production. This industrial revolution can be defined as

- "A new level of organization and control over the entire value chain of the life cycle of products; geared towards increasingly individualized customer requirements" [6].
- "The integration of a multiplicity of technologies and agents for the common goal of improving the efficiency and responsiveness of a production system" [7].

and it is based on six important technologies: Internet of Things (IoT), Additive Manufacturing (AM), Cyberphysical systems, Machine Learning, Robotics and Sensorization.

There is a nonstop evolution of these technologies, making the knowledge and use of them indispensable for any company that wants to be at the top. Proof of this is the talking that the topic generates. Bernard Marr, for example, in his article named ”What Everyone Must Know About Industry 4.0” [8] says that ”Early adopters will be rewarded for their courage jumping into this new technology, and those who avoid change risk becoming irrelevant and left behind”.

In this part of the document, two of those new technologies will be studied more in detail due to the close relation of them with the topic in research, the AM and Robotics:

1.1.1 Additive manufacturing

Additive manufacturing, the group of technologies that build 3D objects by adding material layer-by-layer, has been a top research topic since 1987, when stereolithography (Figure 1.2) was first introduced to the market.
From there on, other processes like VAT photopolymerisation, material jetting, material extrusion (Figure 1.3), binder jetting (Figure 1.4), powder bed fusion, directed energy deposition or sheet lamination (Figure 1.5) have been developed.

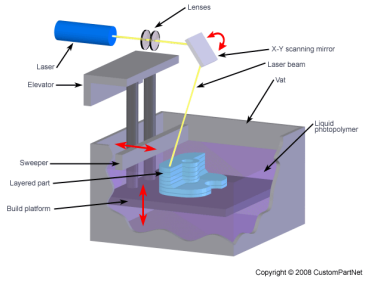


Figure 1.2: SLA

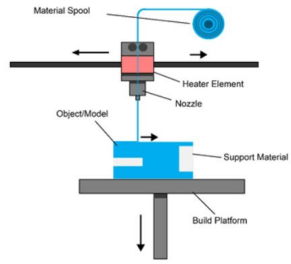


Figure 1.3: Material extrusion

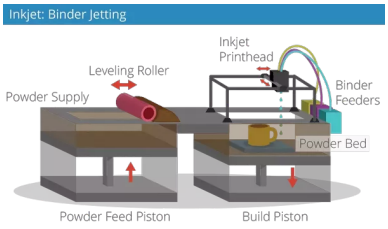


Figure 1.4: Binder jetting

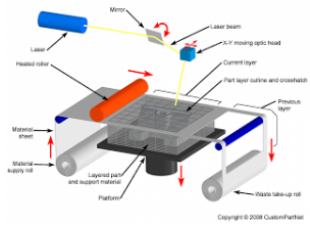


Figure 1.5: Sheet lamination

Nowadays, 3D printing (which is based on material extrusion) is entering the market with great power, not only because of the improvements it brings to industry, but also due to the diffusion of open source software. This is a software development technique based on open collaboration that has created a huge collective on internet, with people sharing all their knowledge in the area. As a result, it is now possible for anyone with a basic knowledge on the topic to have their own 3D printer at home, at a reasonable price.

This technology is becoming so important in industry thanks to its advantages compared to other manufacturing processes. First of all, using an AM technology instead of a traditional one, supposes a reduction in waste material due to the minimization of post-processing and lack of mould, among other things.
Another relevant improvement is the simplification and affordability for customization. This kind of technology decreases the gap between the small-scale and mass production, making it possible to print a customized object for each client, without increasing the cost of production. Other significant advances that worth mentioning are the storage minimization (Objects are usually printed under demand, whose downside is the time needed for it) and Bioprinting (3D fabrication technology used to precisely dispense cell-laden biomaterials for the construction of complex 3D functional living tissues or artificial organs.[9]).

But it also has some disadvantages that need to be studied and improved, in order to make this technology even better. One of the biggest problem is that of legal-rights. As it has been mentioned before most of this technology is open source, which eases the availability of the code to everyone. Its drawback is that it is really difficult to actually know if the conditions of the copyrights are being fulfilled or not.
Another shortcoming is the limited size of the objects to be printed, property defined by the 3D printer being used. The bigger the printing space, the bigger the printer and so the bigger the space needed for all the technology.
Lastly, the late error detection is a big weakness of these kind of technologies, usually having to dispose of the object after an error has happened. Indeed, this will be topic of research later on this document.

1.1.2 Robotics

The other area of interest is Robotics, more precisely robotic arms. The following information is based on [10] and will be studied more in detail in Chapter 2 on Robotic manipulators.

A robot is anything that operates with some degree of autonomy, usually under computer control. It is usual to refer to the term robot (or robot manipulator) as just its mechanical part, while it is just a component. In fact, a robotic system consists of the following parts: Mechanical arm, external power source, end effector, external and internal sensors and computer interface.

A typical robotic arm is made up of seven metal segments (mechanical arm), linked one to another by six joints. The computer controls the robot by rotating individual step motors connected to the joints. This kind of motor is a DC motor that works in discrete steps (one step at a time), not with a continuous rotation. It converts electrical energy into mechanical shaft rotation, rotating a different angle depending on the voltage applied.[11]. Thanks to the step motors, the computer can move the arm very precisely, being able to repeat the exact same movement once and over again. Its job is to move the end effector from one place to the other.

Why 6DOF?

Normal robotic arms have 6DOF; resembling a human arm. The general definition of the term DOF is the amount of pieces of information required for doing a calculation. This term is one of the most important in Kinematics, where the number of DOF of a mechanism is related to the number of variables required to determine its position in space.

In robotics (area of interest for this report), the number of DOF is that of the single-axis rotational joints the robot has. So, saying that a robotic manipulator has 6DOF, is the same as that it has freedom of movement in the three-dimensional space, or that there are six ways in which the body can move.
When working with robotic manipulators, being able to make a relationship between the position of the actuator and the configuration of the manipulator itself is an important task and this is reached by defining those 6DOF as follows:

- Three of them will be used for translation (in X, Y and Z directions).
- The other three for orientation (defining pitch, yaw and roll rotations).
- A 7th DOF is sometimes added; extension. Even if robotic manipulators try to be similar to human arms and humans can not extend their arm, it is an added characteristic.

This movements are described in a more graphical way in the following picture (Figure 1.6)

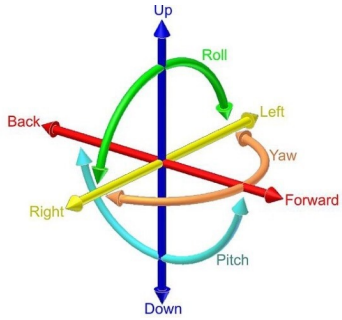


Figure 1.6: 6 DOF movements, taken from [1]

It has been said that each new DOF on a system adds flexibility and mobility to it, why do most robotic arms only have 6 DOF then?
By studying a 7 DOF mechanism it can be concluded that the addition of that extra DOF does not add any new mobility feature to it; being then completely redundant. It has been mentioned before that 6 DOF is the minimum needed to reach a volume of space from every angle. Adding the 7th (or even more) just makes the robot kinematically redundant, meaning that the robot will be able to reach the same point in more than one way.

Having said that, robots in general have more than 6 DOF, and even if it seems contradictory to what it has just been said, here is the explanation to it. A robotic system is usually composed of several simpler modules. When the DOF of a robotic system is mentioned, the sum of the DOF of each module is done. So, even if each module does not have more than 6 DOF, the resulting system does; which does not mean that it is kinematically redundant.

1.2 Motivation

Industry 4.0 is the industrial situation nowadays, and AM and Robotics are two of the leading technologies. Being able to merge both into what can be called Robotic 3D printing is a huge progress achieved by the fourth industrial revolution. With this technology, different shortcomings of AM can be overcome, being the printing space the most significant one, since a robotic arm can cover a bigger printing space than a normal 3D printer while occupying a smaller site.
This is a topic of research of the phd student Linn Danielsen Evjemo, from the Department of Engineering Cybernetics, Faculty of Information Technology and Electrical Engineering at NTNU; as it can be seen in the already published article ”Additive manufacturing by robot manipulator: overview of the state-of-the-art and proof-of-concept results” [12].

Industrial robots, as said before, are nowadays used in a wide range of applications such as welding, painting... all of them being repetitive operations. Taking this into account, the use of this type of robots (robotic manipulators) in 3D printing (also a repetitive operation) is lately increasing. They have many advantages over CNC machines (e.g. higher adaptability to customized production, larger workspace...) which could make choosing them the better option if precision problems were addressed. But the efficiency of this type of robots in 3D printing is not comparable to CNC machines yet.

One of the areas of improvement in the way to increase effectiveness, comes from one of the biggest problems of 3D printing; that is, the waste material produced due to defective objects. In order to overcome this problem, it is necessary to be able to detect, as soon as possible, when the printing is not correct. To detect those errors, Jørgen Jackwitz is developing an image comparison software (See section 1.4 for more information on this research). If an error is detected, there would be another important job to carry out: trying to correct the path or the error so that it does not reach the point where the object needs to be discarded.
Once this problem is addressed, the industry of Robotic 3D printing could reach the top in terms of what is demanded nowadays.

1.3 Objectives

In this research, the goal is to be able to correct the path of a 6 Degrees Of Freedom (DOF) robotic arm with 3D printing purposes. To reach that, there are a few simpler tasks that have to be fulfilled.
Below, a list of the different steps of this thesis is presented:

- Description and analysis of the IRB140 (Obtain a model).
- Research of the Iterative Learning Control (ILC) method.
- Investigation of the different type of errors that can occur while printing with a robotic arm.
- Implementation of a control method for each of the errors.
- Conduction of experiments in RobotStudio; with Virtual Controller (VC).
- Conduction of experiments in the real robot (IRB140); with the Real Controller (IRC5).
- Comparison of results obtained from experiments on SW and on real robot.
- Discussion on the results.

If time: conduct experiments on the real robot already with material deposition. Alternatively, design appropriate disturbances, in order to simulate the deviations that the robot could suffer due to that deposition of material.

1.4 Related work

This thesis is part of a bigger project carried out by the PhD student Linn Danielsen Evjemo. The overall goal of her project is to 3D print with a robot. In this report, the focus is on the path correction techniques, but other thesis have already been carried out in cooperation with her and the supervisor Professor Jan Tommy Gravdahl. Those thesis are the following ones:

1. **An Additive Manufacturing Path Generation Method Based on CAD Models for Robot Manipulators** by Ingrid Onstein. This master student worked on moving from CAD-files to a generated path for a 6 DOF robot manipulator.

2. **Visual Feedback for Large Scale Additive Manufacturing Process** by Jørgen Jackwitz. He worked on visual feedback from a build process using a 3 camera.

Apart from that, it is also worth mentioning the research already done in relation to path correction SW. Although path correction techniques are not very widespread on AM yet, this kind of software has already been used in other industries.

Directional Drilling is one of them, as stated in paper ”Automated Geometric Path Correction in Directional Drilling” [13]. The objective of that research was ”to define algorithms to detect a well going off from a planned path and to find the most efficient corrective path to enable downhole directional automation”. For that, a path-correction software containing 12 different algorithms is described and tested using a prototype.

Another example of path correction is the one stated on paper ”Laser based inspection and path correction system for automatic sewing apparatus” [14]. The problem studied there is that of leather and textiles being flexible materials that change size and position before and during sewing, resulting in a no match of actual and predetermined sewing path.For that, the form and size of each fabric piece is checked using a laser technique, scanned edge point data sets are collected and then, a software is used in order to recalculate the stitch path, through two different methods.

Last, it is worth mentioning two robot path correction methods that have been under development lately.
One of them is based on a vision system [15][16]. The paper ”Robot path correction using stereo vision system” [15], presents the integration of a hybrid vision system with an

industrial robotic arm, for welding purposes. In this case, it is stated that ”vision systems can be used in order to identify the flanges position and edges and provide a feedback to the robot controller indicating the necessary offset to ensure that the welding spot be performed within the desired area”. The other paper, ”On-line correction of robots path based on computer vision” presents a control loop, with all connection types and devices needed to control the path of a robotic arm. For that, the path error was obtained through an image processing algorithm and that data was then sent to a PI controller.
The other method uses 3D ultrasound sensors [17]. In this paper, called ”Method and system for robot end effector path correction using 3-D ultrasound sensors”, the correction of coordinates is done by applying some translations and rotations (by homogeneous transformations) from data obtained from 3D ultrasound sensors. Although not being used for AM, they are a good base for the development of this thesis, as they can be modified and implemented here. For that reason, this papers will be taken into consideration and studied deeper in the following sections of the report.

1.5 Report structure

In this section there is a general explanation of each of the chapters found all along this report. Apart from this, the report also contains a summary, a preface, different indexes, a list with the abbreviations, the bibliography and all the appendices that contain extra information needed to complement the report.

Chapter 2 - Theory
This chapter will be devoted to the theoretical background of this thesis report. There are some basic concepts from two research areas that the reader has to know in order to follow the document; Robotic manipulators on the one hand, and ILC on the other. These are the two sections of these chapter.

Chapter 3 - IRB140
In this chapter, the theory introduced on section 2.1 will be applied to the IRB140. This way, this robotic manipulator will be modeled. This part is important in order to know how does the robot work, as well as all its characteristics and limits.

Chapter 4 - Experiment
This chapter will contain all the important information regarding the experiments carried out in the SW RobotStudio as well as at the laboratory with the IRB140. The errors under study will be introduced and the approach to correct them too. Apart from that, all the steps that have to be followed in order to user RobotStudio as well as the IRB140 will also be explained here. At the end of the chapter, the code developed will be exposed and explained.

Chapter 5 - Results
This is the part of the report where all the results obtained will be shown and analyzed. For that, first of all, the method used to collect the data will be explained. After, a summary of the experiments will be done, followed by the analysis with data obtained from RobotStudio and from IRB140.

Chapter 6 - Conclusion and Future Work
In this last chapter, the general conclusions obtained from the development of the thesis will be exposed. Apart from that, little suggestions for further work will also be mentioned here.

Chapter 2

Theory

This chapter provides all the necessary theoretical background. It is divided into two main parts; Robotic manipulators and ILC control method.

The first part covers all the necessary principles and methods about robotic manipulators used along the thesis; and it is divided as follows:

- **History:** A brief summary with the most remarkable events in the history of robots. At the end, it is described the situation of them nowadays.
- **Definitions:** Different definitions of the term robotic manipulators are introduced here.
- **Parts of a Robotic Manipulator:** Explanation of the different parts that compose a robotic manipulator.
- **Applications:** Typical applications of this kind of robots are mentioned here.
- **Forward Kinematics:** Explanation of the procedure to obtain the forward kinematics of a robot, along with the necessary equations for obtaining the mathematical model of it.
- **Robot programming methods:** Summary of existing programming methods.

The second part, introduces the reader into the ILC control method in a general and simple way, as it will not be used in this case.

2.1 Robotic Manipulators

2.1.1 History

There are a lot of moments in history that stand out for events related to human-invented automation and that have been indispensable for the development of what is now known as a robot.
The Industrial Revolution (second half of XVIII century), for example, linked to the increase in the importance in topics like mathematics or engineering at that time, helped on the way to actual robotics. During the 19th century, development continued in the same direction, with steam-powered machines being used in factories to increase work loads and precision. But the scientific progress made in the 20th century in the field of robotics is remarkably greater than in previous years, with machines capable of assembling other machines or even robots that could be mistaken for human beings.

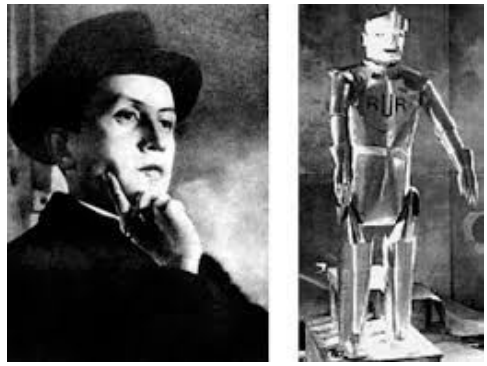


Figure 2.1: Karel Capek and his image about robots, taken from [2]

The term Robot (as we know it nowadays) comes from the Czech word ”robota”, which means ”servitude”, deriving from ”rabu” that means ”slave”. So, it can be said that the word robot, in its origins, meant something like ”forced labour or work”. This concept was introduced in our vocabulary in 1921 by Karel Capek, on his satiric novel ”Rosmus Universal Robots”. Here, he described a robot as a machine that substitutes a human being developing tasks without resting and that rebel against their human masters (See Figure 2.1). From there on, almost any mechanical system with movement was called robot. The next mention of this word can be found in the short story ”Runaround” by Russian-born American science-fiction writer Isaac Asimov in 1942 where he described the robots as

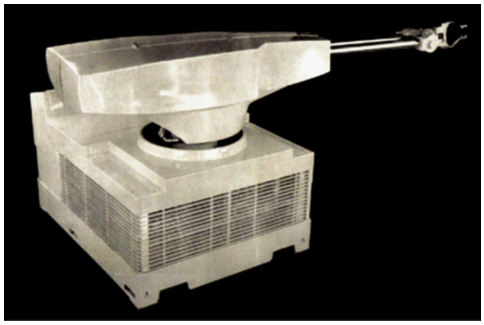


Figure 2.2: Unimate robotic arm. Image source

helpful servants. He proposed three ”Laws of Robotics” [18]:

”A robot may not injure a human being or, through inaction, allow a human being to come to harm.” (First Law)

”A robot must obey orders given it by human beings except where such orders would conflict with the First Law.” (Second Law)

”A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.” (Third Law)

Until this point in history, everything had just been theory. The first ”modern” robot was created by George C. Devol (inventor from Louisville) in the 1950s. He designed ”Unimate” (See Figure 2.2), a reprogrammable manipulator that started to work in 1961. This years of delay were due to the fact that Devol was not able to sell his product and had to give his patent to Joseph Engelberger, who modified the robot into a more industrial version. For the success he had on this, Engelberger is known as ”The Father of Robotics”. Although at the beginning it was seen more as a curiosity, it soon became a normal device in the industrial manufacturing sector.

At that point those machines were programmed to perform a task without any interaction with the real world. That is, they were programmed to perform repetitive tasks. In contrast to that, industrial robots have nowadays and progressively become more and more sophisticated, reaching the point where they are able to interact with the environment. For that, a lot of programming is required in the robot; becoming the coding an extremely important part of robotics.

Finally, and looking at the future, it is indubitable that robotics will be in continuous development and that being up-to-date in this area will be indispensable for any company wanting to be a leader in industry. An interesting prediction of future in robot programming was done by Samuel Bouchard in his article about ”The Evolution of the Robotic Teach Pendant” [19]; where he stated:

> ”The ultimate programming will probably morph into autonomous learning by the robot using sensor information and experience, maybe even experience of other robots.”

2.1.2 Definitions

A robot is a reprogrammable machine that is capable of movement in the completion of a task. They use special coding that differentiates them from other machines and machine tools, such as CNC. There is a wide range of definitions of this term, from the simplest one to the one that contains all the details. Starting with the simplest, and more visual one:

"A computer controlled industrial manipulator of the type: (See Figure 3.1)" [10].

Figure 2.3: Robot IRB 140 from ABB

Another definition is the one adopted by the Robot Institute of America (RIA), called also the industrial definition and probably the commonly most accepted one:

"A multifunctional, reprogrammable manipulator designed to move material, parts, tools or specialized devices based on programmable movements for the execution of a variety of tasks."

The last definition that will be presented here is the modified version of the previous one and it is the one found in the International Organization for Standardization (ISO):

"An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications."

2.1.3 Parts of a Robotic Manipulator

A robot is anything that operates with some degree of autonomy, usually under computer control. It is usual to refer to the term robot (or robot manipulator) as just its mechanical part, while it is just a component. In fact, a robotic system consists of the following parts:

1. Mechanical arm (Robotic manipulator)
   This gathers all the mechanical parts of the robot, those that without the controller are just simple materials. The robotic arm is built to resemble a human arm and it is composed of an assembly of rigid links and flexible joints.
   - Links: the rigid sections that make up the mechanism. (chest, upper arm, fore arm)
   - Joints: the connection between two links. (shoulder, elbow, wrist)
2. End-of-arm tooling (End effector)
   The End effector or end-of-arm tooling (EOAT) is the part connected to the end of the robotic arm (to the wrist), the one interacting with the environment and because of that, known as the most essential robot peripheral. Depending on the task the robot has to develop, its structure and nature of programming and hardware will be different. For basic jobs it is common to use off-the-self components, as they are cheaper and reduce the preparation time. For special jobs instead, the end effectors are custom made,, that is, each of them is designed and produced with the specific characteristics and measurements of the task to develop.
   Some of the usually used end effectors are listed below.
   - Grippers (most common; for pick and place operations)
   - Brushes
   - Magnets
   - Sanders
   - Screw drivers
   - Material removal tools (e.g. cutting tools or drills)
   - Spray guns
   - Vacuum cups
   - Welding guns
   - Tool changers
3. External and internal sensors
   The sensors make it possible to have exchange of information between the robot and the environment. They send different measurements from the environment to

the robot, which is an essential task for the robot to work properly. There are three types of piece of information that can be sent:

- Different type of measurements (e.g. force-torque sensors are used to measure the force and the torque in part insertion or assembly operations)
- Security or error warnings (e.g. anti-collision sensors prevent damaging the robot tooling or the object that is being manipulated)
- Real time information about the task itself (e.g. installing cameras close to the robot make it possible to compare the theoretical with the practical movement)

4. External power source
   At the moment, it is quite usual to use batteries as a source of energy. The design of a robot powered by a battery should take into account factors such as security, cycle life and weight. Another source used quite often is the generator, e.g., internal combustion motor. However, drawbacks like its mechanical complexity, the need of gas or heat dissipation and its weight result in a decrease of its use.

5. Computer interface (robot controller)
   This can go from simple discrete I/O to industrial communication protocols

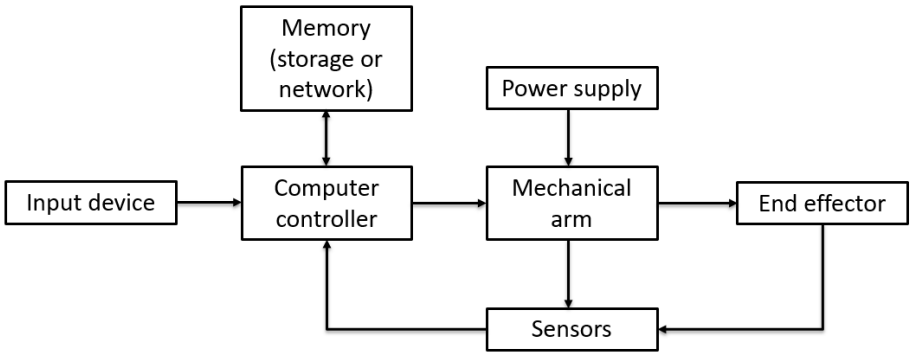


Figure 2.4: Components of a robotic system

2.1.4 Applications

From the scientific point of view, the robotic manipulators are a great area of research, making it possible to put into practice all their theory. In the past, robots were seen as a curiosity, but they soon began to have a great acceptance in industry, being now in a point where they are an indispensable part for automatizing industrial processes and, therefore, becoming a normal tool in the industrial manufacturing arsenal. They are used in a wide variety of industries thanks to their robust resistance capabilities, precision function or its flexibility to automatize and adapt to working environments. Apart from that, their use introduce advantages such as cost reduction, increase in productivity, product quality improvement or reduction in environmental issues.

Currently, robots are used extensively in industry, being an indispensable element in a great part of manufacturing processes. The industrial robot is multifunctional, that is, in theory, it can be used for an unlimited number of applications. In practice, however, it has been proved that its use is optimal only in certain processes.
The most common applications are:

- Welding
- Painting
- Assembly
- Pick and place
- Packaging and labeling
- Palletizing
- Product inspection
- Testing
- ...

Although it is usual to link the usage of this type of robots to mass production, as programming and technology improve it is becoming more and more common to use them in jobs/tasks that in the past have been too dangerous or impossible for humans to achieve, or when the quality of the results is very exigent. An example of this is that robots are being launched into space to complete the next stages of extraterrestial and extrasolar research. Nowadays, robots have fund place in spheres like toys and entertainment, military weapons, search and reserve assistants, and many others.

2.1.5 Forward kinematics

The term kinematics, in robotics, involves the study of motion for a robot manipulator, without looking at forces and torques that affect motion. It is usually divided into forward and inverse kinematics. Here, the forward kinematics of a robotic manipulator will be introduced.

Calculating the Forward Kinematics of a robot is an important step when studying a robotic manipulator, as they are used to obtain the position and orientation of the end-effector for a given set of values of the joint variables by using the kinematics equations. It is not topic of this thesis to explain the theory or get into a detailed explanation on how this kinematics equations are obtained, that is why, only the most relevant concepts will be covered here; those thought as indispensable for applying the forward kinematics to the IRB140, as done in Section 3.2. The material for this section is taken from [10]; where further reading on the topic can be done.

Kinematic chain

A kinematic chain is a group of rigid bodies (also called links) connected together by joints. A simple example of it is the robot manipulator, defined as a simple chain formed by rigid bodies connected in series.

The joints can be simple (revolute or prismatic) or more complex (ball or socket); but from now on and to make calculations easier, only kinematic chains with simple joints will be studied, and thus, the following assumption will always be true:
"Each joint has a single degree-of-freedom and, in consequence, the action of each of them will be described by only one parameter. (See equation 2.1)"

$$q_i = \begin{cases} \theta_i & \text{if joint i is revolute} \\ d_i & \text{if joint i is prismatic} \end{cases} \tag{2.1}$$

Apart from that, there is some basic nomenclature that is important to understand and keep in mind when following the steps and equations that are described in the next section. (Section 2.1.5 on DH convention)

- A robot manipulator composed of n joints will have n+1 links.
- The joints are numbered from 1 to n and the links from 0 to n.
- Joint i connects link i-1 to link i, and it is fixed with respect to link i-1.
- Each link has a coordinate frame rigidly attached. (Coordinate frame $o_i x_i y_i z_i$ is attached to link i)
- The inertial frame (i=0) is the one attached to the robot base.
- The homogeneous transformation matrix A_i:
  - Gives information about position and orientation of $o_i x_i y_i z_i$ with respect to $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$.
$$A_i = \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{2.2}$$
  - According to the assumption made before, it is only function of a variable, q_i. (Equation 2.1)
- By developing the following multiplication, the position and orientation of the end effector with respect to the inertial frame is calculated: (It contains the total rotation matrix R_n^0 and the total length o_n^0)

$$H = T_n^0 = A_1(q_1) \cdots A_n(q_n) = \begin{bmatrix} R_n^0 & o_n^0 \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.3}$$

In summary, there are two main steps to follow to determine the forward kinematics of a robot manipulator:

1. Set the coordinate frame for each joint
2. Fill in the DH parameter table
3. Calculate each $A_i(q_i)$
4. Multiply them together to obtain H

But, even if it seems an easy task, it has been proved that developing the kinematic analysis of an n-link manipulator can be quite complicated. In order to simplify the analysis as much as possible and to try to make it universal, conventions such as The Denavit-Hartenberg (DH) have been introduced over the years. In the following section (Section 2.1.5, the DH convention will be covered.

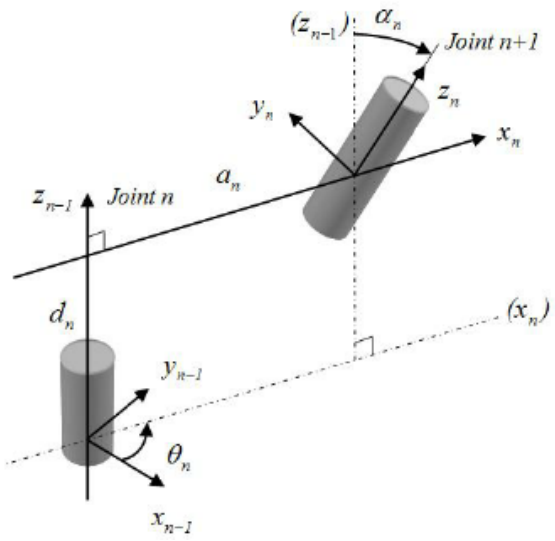


Figure 2.5: DH frame assignment, taken from [3]

DH convention

This is a systematic approach to the calculation of the homogeneous transformation matrix (Equation 2.3) of a kinematic chain. In general, any homogeneous transformation matrix has six relevant numbers (e.g. three that specify the position and three Euler angles). In the DH convention instead, only four parameters are required, but for that, a clever choice of the origin and coordinate axes has to be made.
All the procedure on how to define the coordinate systems of each joint is described on p72-74 of [10]. There are two important assumptions that need to be fulfilled when making the choice of each coordinate frame, so that equations stated on the DH convention (shown below) can be used. This assumptions assure existence and uniqueness of the homogeneous transformation matrix for the coordinate frames chosen, and are stated as:

(DH1) The axis x_1 is perpendicular to the axis z_0.
(DH2) The axis x_1 intersects the axis z_0.

When using this convention, frames are assigned as shown in Figure 2.5 and each A_i matrix is calculated by the following basic transformations:

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \tag{2.4}$$

Where the four parameters are commonly known as:

- a_i: Link length
- d_i: Link offset
- α_i: Link twist
- θ_i: Joint angle

By remembering the assumption from 2.1.5; it can easily be deduced that three of the four variables will be constant, so that each matrix A_i is only function of one variable.

Being the following the expressions for each of those matrices that make up A_i

$$Rot_{z,\theta_i} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & 0 \\ sin\theta_i & cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Trans_{x,a_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans_{z,d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Rot_{z,\theta_i} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & cos\theta_i & -sin\theta_i & 0 \\ 0 & sin\theta_i & cos\theta_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The result of making the product of the above equations in the order described in Equation 2.4 is the following:

$$A_i = T_i^{i-1} = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_{i,i+1} & sin\theta_i sin\alpha_{i,i+1} & a_{i,i+1} cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_{i,i+1} & -cos\theta_i sin\alpha_{i,i+1} & a_{i,i+1} sin\theta_i \\ 0 & sin\alpha_{i,i+1} & cos\alpha_{i,i+1} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

2.1.6 Robot programming methods

There are three different methods to program any kind of robotic arm. Below, a description of each of them will be made, along with the advantages and disadvantages of each of the methods.

Offline programming

This method consists of using a virtual controller (see Section 3.4.2) to program the robot, that is, simulate and program all the paths or routines the robot has to repeat and, once finished, send the code to the real robot. This method has a wide variety of uses.
First of all, it is used in robotics research when advanced control algorithms are implemented. This way, the correct operation of the code can be ensured before trying it in the real robot.
Apart from that, it is also used in industry, as it is the best way to increase the cost effectiveness in robotic systems. The most relevant advantages of this method are the following ones:

- Downtime reduction (e.g. by the ability to test many different approaches to the same problem)
- Efficiency increase
- Easy to use (It is very intuitive and usually not deep programming skills are needed)
- Increase in productivity (i.e. it allows formation, programming and optimization of programs without altering/stopping the production)
- Risk reduction (e.g. by using tools such as collision detection, breakdowns can be detected and corrected)

But there are also some disadvantages, compared to the other methods:

- Not %100 accuracy is obtained
- Software errors (i.e. Instead of just focusing on the code or process, the software might give some errors too, which would result in a waste of time; fixing the software itself, as well as a waste of resources; as a person would need to be there to fix it)

Teaching pendant

This is nowadays the most popular method, where the industrial robot is controlled remotely using a teaching pendant (control box for programming the options of a robot). With this method, the robot is controlled step by step (what is also known as point-to-point movement), that is, one command is sent at a time. The positive points when using this method are:

- Familiarity among technicians (A high percentage of industrial robots nowadays come with a teach pendant)
- Precise positioning

And, on the contrary, the main drawbacks are:

- Training required (Even if it is familiar for technicians, other people that are not used to working with robots will need training to learn how to use it and program with it)
- Operations halted while programming (i.e. The robot has to be put into ”teach mode” while it is been programmed, which forces to stop the rest of operations)

Teaching by demonstration

When using this method, the operator stores each position in the robot computer. A lot of robots nowadays have incorporated this method, as it is easy for operators to get started immediately using the robot with their applications. The advantages are the following:

- Quicker
- No need of knowledge in programming concepts

And on the contrary the main disadvantages of this method are that it does not reduce the downtime and that positioning is not so straightforward.

2.2 Iterative Learning Control

There are many control techniques and methodologies that can be applied to the control manipulators. The particular control method chosen as well as the manner in which it is implemented can have a significant impact on the performance of the manipulator and consequently on the range of its possible applications.

In this section, theory on the control method *Iterative Learning Control (ILC)* will be introduced. When thinking about the thesis report, it was first thought to develop a literature review on different control schemes, so that all of them could be studied. But in the end, just the ILC will be studied, as this is the one that best fits this type of technologies.

The concept of ILC describes a technique used for improving the response (compensate the error) of a system that operates repetitively. All systems working in a repetitive mode will suffer the same overshoot, rise time, settling time, and steady-state error in each repetition. ILC tries to improve this output by updating the control signal until finding the input that converges the output error into successful results [20]. One of the most relevant motivations for the development of this concept is the industrial robot, due to the fact that it is usually designed and used to repeat the same task once and again, in an industrial process [21].

It is long since this concept has been in use, even before it was officially known by its name, Iterative Learning Control. The first use of this idea can be dated in 1967 (and patented in 1971), when ”Learning control of actuator in control systems” was published. This paper studied the idea of storing a ”command signal” in memory and then updating it in an iterative way using the error between the actual response and the desired response [22]. When searching for the exact concept ILC, it was first introduced in 1978 by Uchiyama, but he published it in Japanese, so it was not until 1984 that it became an active research area. This happened with the publication of ”Bettering operation of robots by learning”, by Arimoto, Kawamura and Miyazaki [23]. It was published in English and stated a method that used a simple iteration rule that autonomously generated a present actuator input better than the previous one [23].

There are some assumptions taken into account in ILC that make a clear distinction of this method with the standard tracking problem [24]. At t=T (one period/repetition):

- The state x of the system is reset to the same initial condition x0, that is x(T) = x0.
- After resetting, the system is again required to track the same reference signal r(t).

The standard ILC scheme is the following:

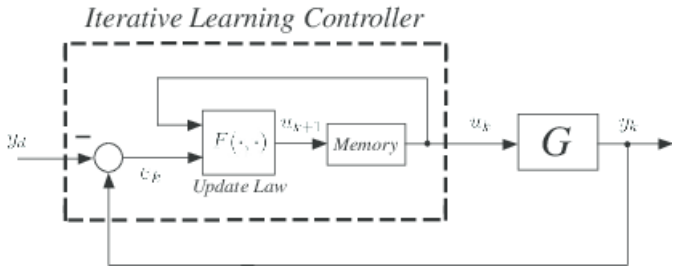


Figure 2.6: Standard ILC schematic, taken from [4]

Chapter 3

IRB140

In this chapter, the hardware used for the experiments will be introduced. In this case, the robot manipulator chosen is the IRB140 from ABB (Figure 3.1); robot available in the laboratory at the Department of Engineering Cybernetics, Faculty of Information Technology and Electrical Engineering at NTNU.
This chapter applies the theory from section 2.1 on the robot IRB140. The parts of this chapter are the following:

- **Characteristics:** Important characteristics of the IRB140; such as different versions available in the market or its measurements.
- **Forward kinematics:** Developing the DH coordinate frame representation and its corresponding parameters table, as well as the homogeneous transformations.
- **Inverse kinematics:** Derivation of equations of motion
- **Robot programming methods:** Application of the three methods described on Section 2.1.6 to IRB140.

Figure 3.1: IRB140

3.1 Characteristics

The IRB140 is a 6DOF robotic manipulator from the wide catalogue of industrial robots that ABB offers. This company is a leader in industrial robotics, having a wide variety of robots for different applications and describes itself as:

”ABB is a pioneering technology leader that works closely with utilities, industry, transportation and infrastructure customers to write the future of industrial digitalization and realize value.” [25]

The manipulator IRB140 has 6 links (one for each DOF mentioned before); with all revolute joints; it can handle up to 6kg and with the working area seen in the following picture (Figure 3.2).

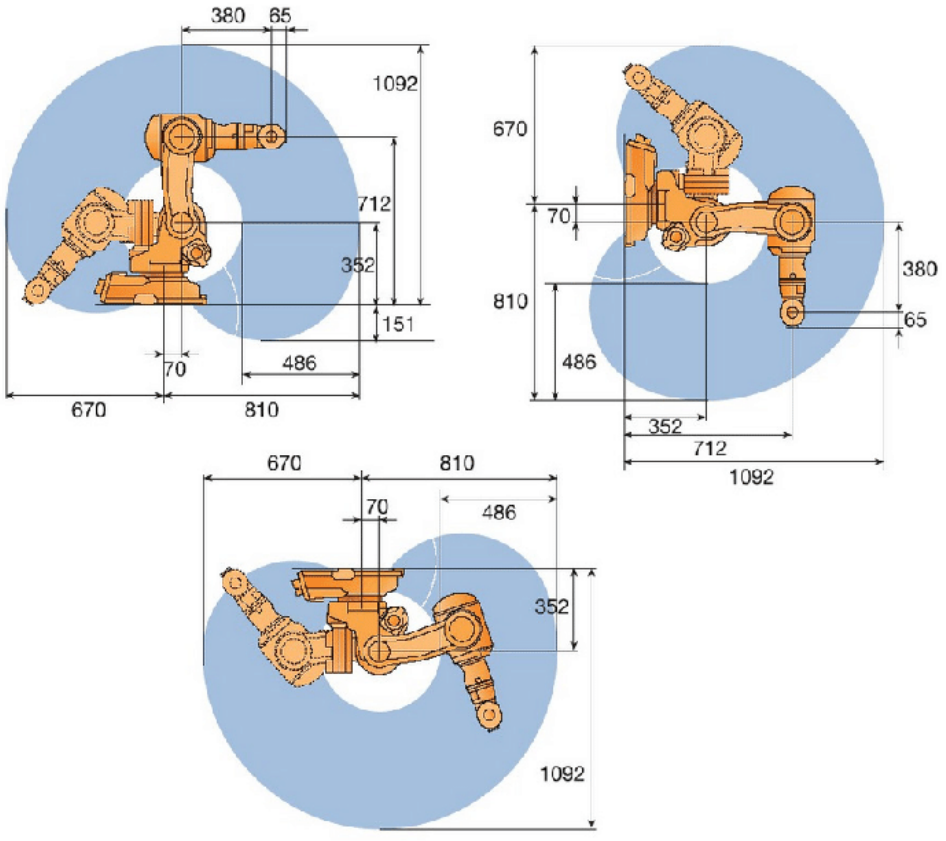


Figure 3.2: IRB140 workspace (up-left: Floor mounted; up-right: wall mounted; down: suspended)

It is very used in industry thanks to its flexibility and versatility when it comes to the position of mounting it, as it can be floor mounted, suspended or wall mounted with any angle, what makes any line of production possible (See Figure 3.2 ”IRB140 workspace” for an image of the three options).
There are some factors that are key to ensure the reliability and the security of the robot. One of them is its robust design, with completely integrated cables and being well-suited for functioning in exigent environments (it has all its mechanic arms protected with IP67 protection). Apart from that its collision detection function with complete path retraction is also an important feature of this robot. It is also important to say that it outstands for its precision, being able to provide a uniform quality thanks to its repeatability of the path.

It can be found in different versions:

- Standard
- Clean Room
- Foundry Plus
- Washdown

In the following table (Table 3.1), a summary of the specifications and robot motion range and speed of the different joints is exposed.

Table 3.1: Summary of IRB140 robot information

Robot Specification		**Robot Motion Range**		**Robot Motion Speed**	
Axes:	6	J1	$\pm 360°$	J1	200°/s
Payload:	6Kg	J2	$\pm 200°$	J2	200°/s
H-Reach:	810 mm	J3	$\pm 260°$	J3	260°/s
Repeatability:	$\pm 0.03mm$	J4	$\pm 280°$	J4	360°/s
Robot mass:	98 Kg	J5	$\pm 240°$	J5	360°/s
Structure:	Articulated	J6	$\pm 800°$	J6	450°/s
Mounting:	See Figure 4.2				

For more information on the robot, see the manual [26] or the data sheet [27] (also found in Appendix A)

3.2 Forward kinematics

As mentioned before in section 2.1.5 , deriving the forward kinematics means obtaining the position and orientation of the TCP. For that, the DH convention will be followed in this case.

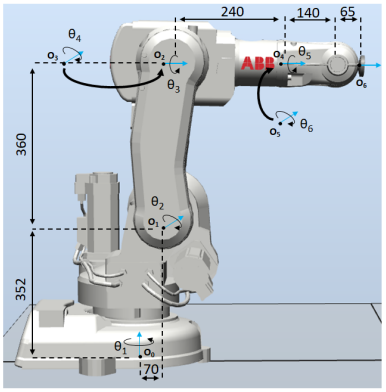


Figure 3.3: Model of IRB140 with dimensions (in mm) and joint variables

3.2.1 DH frame assignment

The first step is to draw a model of the manipulator, with a frame attached to each of the joints. Each of the frames has been set following the DH convention mentioned before in section 2.1.5. The result of applying it, is the one of Figure 3.4.

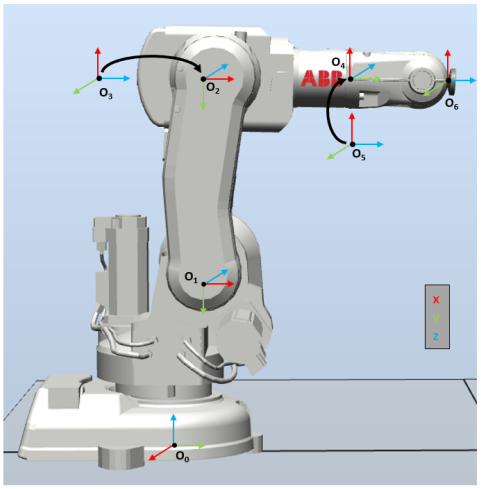


Figure 3.4: DH frame assignment of the IRB140

3.2.2 DH parameter table

With that coordinate frame assignment, the following step is to create the DH parameters table. This can be found in table 3.2. This parameters will be used later when deriving the homogeneous transformation matrices in the next step.

Table 3.2: DH parameters table

Links	a	alpha	d	theta
1	70	-pi/2	352	$\theta 1^*$
2	360	0	0	$\theta 2^*$
3	0	-pi/2	0	$\theta 3^*$
4	0	pi/2	380	$\theta 4^*$
5	0	-pi/2	0	$\theta 5^*$
6	0	0	65	$\theta 6^*$

3.2.3 Homogeneous transformation

The next step is to obtain the homogeneous transformations. As the robotic manipulator has 6 DOF and one homogeneous transformation matrix is needed for each of them, MATLAB has been used for obtaining them (The code used has been attached in Appendix B). This way, when developing the homogeneous transformation of the robot, the product among the matrices has been obtained directly from MATLAB, without having to perform it by hand (Reducing that way the calculation time and error in a notable way). No multiplication among them has been done, because it was not needed for this thesis report and would only be a big matrix with a lot of numbers and variables. If the homogeneous transformation of the system is needed, a multiplication of all the matrices is enough.

$$A_1 = \begin{bmatrix} cos\theta_1 & 0 & -sin\theta_1 & 70cos\theta_1 \\ sin\theta_1 & 0 & cos\theta_1 & 70sin\theta_1 \\ 0 & -1 & 0 & 352 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$A_2 = \begin{bmatrix} cos\theta_2 & -sin\theta_2 & 0 & 360cos\theta_2 \\ sin\theta_2 & cos\theta_2 & 0 & 360sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$A_3 = \begin{bmatrix} cos\theta_3 & 0 & -sin\theta_3 & 0 \\ sin\theta_3 & 0 & cos\theta_3 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$A_4 = \begin{bmatrix} cos\theta_4 & 0 & sin\theta_4 & 0 \\ sin\theta_4 & 0 & -cos\theta_4 & 0 \\ 0 & 1 & 0 & 380 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$A_5 = \begin{bmatrix} cos\theta_5 & 0 & -sin\theta_5 & 0 \\ sin\theta_5 & 0 & cos\theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$A_6 = \begin{bmatrix} cos\theta_6 & -sin\theta_6 & 0 & 0 \\ sin\theta_6 & cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 65 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

3.2.4 From wrist to tool

It is known by theory that the tool does not increase the DOF of the robotic manipulator, so it is an easy task to find its position and orientation by a rotation and translation from the last joint (being the four DH parameters known constants). The end effector used in the experiments with material deposition is a glue gun. But, as it will be seen later, no experiment will be done in the end with material deposition. That being the case, the mathematical model will be the one obtained in the previous part (Section 3.2.3 where no end effector was used.

3.3 Robot programming methods

The IRB140 robot gives the option of programming with either of the three methods described on Section 2.1.6.

3.3.1 Offline programming

The software RobotStudio (See Section 3.4) is the offline programming software created by ABB with the purpose of making it easier for the user (the clients that have bought their product) to program different processes by using the Virtual Robot Technology (VRT).

This is the method chosen for programming the different experiments and control schemes developed along with the thesis. This way, all the code done will be tested and simulated on the software, trying it on the real robot just after a successful simulation has been reached.

3.3.2 Teaching pendant

The IRB140 robot at the laboratory is equipped with a teaching pendant, as it can be seen in the following picture (Figure 3.5).

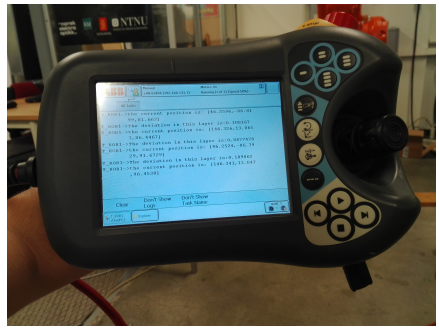


Figure 3.5: Teaching pendant available on laboratory

But, in this thesis, this machine will not be used for programming purposes but for controlling the robot once the program has been developed and tried in RobotStudio or, in some cases, to make some little changes on the code while simulation is going on.

3.3.3 Teaching by demonstration

The robot IRB140 available at the laboratory has also the option of programming by this technique, but it will not be considered for the experiments in this thesis. The reason for that is that even if it is possible with this method to store positions, a control scheme can not be programmed this way. That being the situation, it is not worth using this technique just for storing the positions or targets of the path, and then, programming the rest by means of one of the other two possibilities (Offline programming or Teaching pendant).

3.4 RobotStudio

RobotStudio is an offline programming software (See section 2.1.6 for more information on programming methods) that allows creating, programming and simulating ABB industrial robot cells and stations in a practical and easy way. It is known for being a very realistic virtual environment that allows simulating in a very precise manner the real application/process. Its main characteristics are:

- Automatic creation of any type of station.
- Ability to import any kind of 3D geometry or model, of any format.
- Programming and kinematic simulation of the stations.
- Easiness of robotic cell design and creation.
- Ability to export to the real station the results obtained.

The programming language used in RobotStudio is RAPID, which makes it possible to change or add code to the one created automatically after a CAD design. This SW contains a huge library with different manipulators , tools or controllers, but, apart from that, it is also possible to create new objects or tools (similar to a CAD SW).

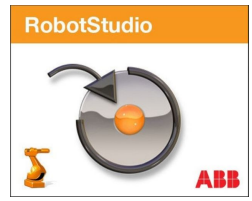


Figure 3.6: RobotSudio logo

3.4.1 Concepts and nomenclature

- Objective: Position or coordinate that the robot must reach and composed of
  - a position (x,y,z)
  - an orientation (rx,ry,rz)
  - a configuration (In which form the robot must reach the objective, defined by the quaternions)

- Trajectory: Instruction sequence of movements towards the objective. An instruction is composed of a reference to an objective, movement data and a reference to the tool and work-object.
- Coordinate system (CS):
  - World CS: It represents the totality of the station or robot cell. It is used as a base reference for all the other coordinate systems.
  - Robot base CS: Each robot in a station has a coordinate system attached to its own base, which is used as a reference for its objectives and trajectories.
  - Tool centre CS: Coordinate system situated at the center of the tool.
  - Work object CS: It is situated at the physic work piece.
- Robot configurations: The controller can give more than one configuration option for reaching the same objective. Each configuration has its distinctive quaternions, making it possible to distinguish among different configurations.

  Quaternions Another way to represent an orientation and a rotation in a 3-dimensional space. This method is a lot simpler compared to the Euler angles and more efficient and stable numerically compared to the rotation matrices.

3.4.2 Virtual Controller(VC)

This is the tool used by Robotstudio for running the robots as they can run real systems as well as virtual systems (for testing and evaluation). It uses the same SW as the real system, in order to obtain the closest results to those in reality as possible.

3.4.3 Steps to implement a robotic system

1. Choose a robot manipulator, with the tool and workspace required for the application.
2. Set the manipulator on the same position on the virtual workspace than that in the reality.

3. Create targets and join them in the desired order to create paths for the robot to follow. Each target needs a robot configuration, as there is usually more than one way in which the robot can reach that target.

4. Change RAPID code to meet all your requirements.

5. Using the virtual controller and the virtual Flexpendant, simulate the code to search for errors. (It is easier and time and resource saving to check it on the computer, before transferring the codeto the real controller and try it on the real robot)

Chapter 4

Experimental method

It is known that additive manufacturing requires the deposition of material layer by layer, having to put material on previously accumulated one. This brings up some important issues that have to be analyzed in detail and solved somehow, so that the final product needs as little post-processing as possible; that is, obtaining an object as close to the final version as possible.
Being this an important research area nowadays, this chapter will be devoted to the experimental part of the thesis; that is, the study and correction of two different type of errors that can occur while printing with a robotic arm. For that, a control scheme will be introduced for each of the errors. This chapter will be organized as follows:

- **Introduction to the experiment:** Explanation of the basics of the experiment, as well as introduction of the assumptions taken for it.
- **Introduction to errors:** General description of each of the errors accompanied with the approach chosen used to correct each of it.
- **Getting started with RobotStudio:** Steps that have to be followed in RobotStudio to fulfill different tasks; such as setting the station.
- **Getting started with IRB140:** Steps that have to be followed at the laboratory to start the robot or to launch an execution (among other tasks)
- **RAPID code:** Explanation of the objective of each of the procedures found on the code developed.

4.1 Introduction to the experiment

The main idea of the experiment is to develop a piece of code in RAPID (programming language used in RobotStudio) to implement a control method in order to correct the two errors under consideration in this thesis report. This code will be debugged while simulating it with the VC IRC5 in RobotStudio. Once its effectiveness has been proved there, the code will be ready to be launched to the real controller and to be checked on the real robot IRB140.

The assumptions taken all over the research are the ones shown in the list below. The experiment is thought to see the effectiveness of the code when correcting the two errors described later, that is why, it will be as simpler as possible.

1. **Limited to straight lines:** The code will be thought for correcting only paths following straight lines. This way, the correction of path deviation will be done in an easier way, as it will be described later in this chapter.

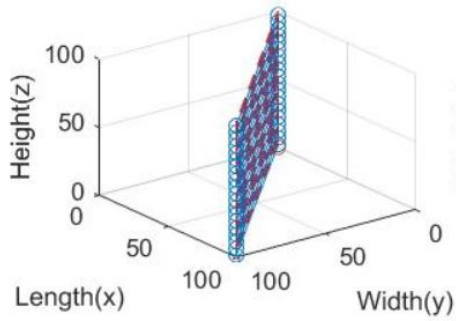


Figure 4.1: 3D view of theoretical path in experiments

2. **Constant deposition of material:** Even if no experiments with material deposition have been carried out in the end, this is important to take into account when developing the code. The glue gun used at the laboratory is started and stopped manually introducing a period of imprecision in the deposition of the material. In order to have a constant deposition, this starting and stopping of the gun will be done before and after the start and stop of the code consequently. This way, the material deposition can be taken as constant all over the execution of the code, as well as the initial and final diameters of the filament.

Taking into account all the assumptions mentioned above, the path selected for the experiments is a straight wall, as the one shown in Figure 4.1.

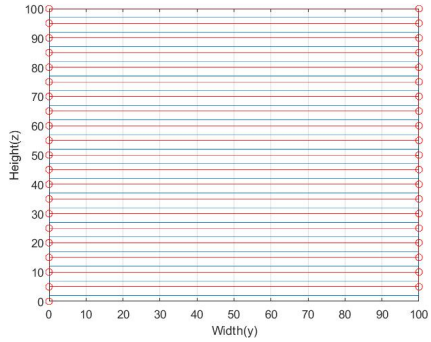


Figure 4.2: Front view of theoretical path in experiments

Apart from that, the material selected for the experiments will be the insulating foam. This choice was done in order to face the problem of expansion or contraction (expansion in this case) of materials. And even no experiments with material deposition will be done at the laboratory for this thesis in the end, all the code will be developed on the thought of using that material later on.

4.2 Introduction to errors

As mentioned in previous parts, two errors will be examined and corrected in this thesis. In the following pages, a general description of the cause of each of the errors and the approach to correct each of them will be done.

4.2.1 Correction of the height

The first error that will be studied and corrected is the height of each layer.

General description

As the 3D printing industrial world increases and develops, so does the variety of materials that can be used. Each material has its own properties and some of them will be crucial for the behaviour and result of the printing. Because of that, it is important to know beforehand which are the properties that have to be taken into account, and what is the influence of each of them in the result of the printing.

One of the properties that is important to take into account is the change in volume that some materials suffer from. This change can be due to thermal expansion/contraction (where the change in volume results from a change in temperature; in concrete for example) or due to a chemical reaction (where the change comes after a mix of two or more materials; in insulation foam for example). Being the origin one or the other, this is an issue that needs to be addressed and overcome before starting to print, as the diameter of the filament will change in time and consequently, so will have to do the height of each layer.
This change in volume can happen either as an expansion or as a contraction, but to simplify the writing and understanding of this section, only expansion will be studied, as the material used in the experiments performed at the laboratory will suffer from expansion, not from contraction. If a material that suffers from contraction is to be used, the same code would be valid; with just a few changes.

In this cases, a compromise has to be taken between the printing velocity and the change in height for each layer. When the material suffers from expansion, the printing velocity is usually chosen to be quite slow, so that the change in volume happens before

the next layer has to be deposited. This way, by knowing the expansion percentage, the layer width can be pre-calculated and set as a constant. In contrast, if the printing velocity is not slow enough, the change in volume will be happening while the next layer is printed, colliding one layer with the other in the cases of expansion, and not having a good contact between layers in case of contraction.
Apart from that, this will also affect the final dimensions (height in this case, the other dimensions will be covered later in this chapter) of the object printed, having to calculate the amount of layers to be printed taking into account the width of each layer after the expansion has happened.

Correction approach

There are two main dimensions that are of real importance when working with materials that expand after some time. One of them is the initial diameter of the filament used and the other is the final diameter of that same filament. That is why, the important thing to do at the beginning of each execution is to ask the user to enter those dimensions.

After that, the first layer will be deposited with a height of that of the filament before any expansion happens (initial diameter of the filament) and with a slow enough printing velocity so that the material has time to expand before next layer is printed.

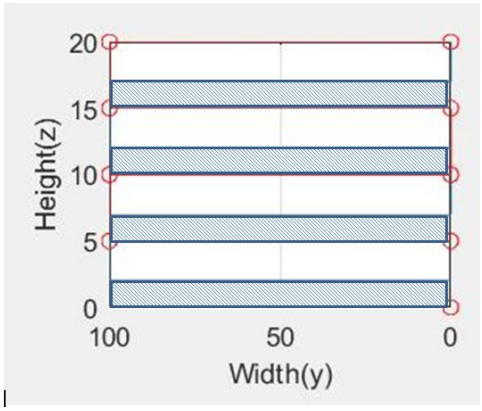


Figure 4.3: Frontal view of the path followed by the robot, with initial diameter of the filament in blue and final one until red line

In the following layers, the method will be the same. Once that expansion has happened, the robot will go up the same amount of mm as that of the final diameter of the

filament. This is done like this in order to count the expanded part and to have enough space to print the following layer with the initial diameter of the filament. This is seen in a more graphical way in Figure 4.3 and in a mathematical way in equations 4.1 and 4.2.

$$\text{Increase of z} = Z_{expansion} + D_{initial} \tag{4.1}$$

$$Z_{expansion} = D_{final} - D_{initial} \tag{4.2}$$

This will be done by updating the z component of each target in each iteration, as it can be seen in the procedure *update_height* described on section 4.5.2. Later on, when visual feedback is used in the code, the update of the height will be done by checking the expansion of each layer with the cameras, to obtain this way a lot more precise results.

4.2.2 Correction of path deviation in straight lines

The other error that will be analyzed more in detail is the deviation of the path in straight lines.

General description

Even if industrial robotic arms have a high precision, depending on the diameter of the filament deposited that precision is not enough. That is why, it is important to check the deviation and to correct it in case it is not inside the requirements.

In this case, the problem that will be studied is the path deviation in straight lines. This simplification is done in order to make it easier and simpler to develop a first version of the software and, once the software is working, make the necessary changes for it to cover other kind of paths.

Correction approach

In this case, an iterative process will be used in order to correct the path deviation. The use of RobotStudio and RAPID language limits in a big way the use of common control methods. After making a research on them (Section 2.2) it was concluded that no known

method would be used, thinking that it is simpler but effective in the same way to work just on RobotStudio rather than having to link it to Matlab, for example. This will be commented later on Section 6.2, where all suggestions for further work will be done.

Instead, an easy iterative method will be programmed with RAPID language in RobotStudio. For that, there are some issues that have to be addressed first of all:

- **Slope limit when 3D printing:** A lot of research has already been done in this area, and it is known that the limit of the slope is set in 45 degrees with the vertical plane, as shown in Figure 4.4. If that limit wants to be exceeded, some kind of support will be needed, adding wasted material to the production of the object.

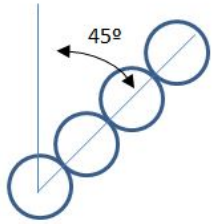


Figure 4.4: Limit of slope (with zoom)

  In order to have the path corrected in the least amount of steps possible, this limit of 45 degrees will be used.

- **Deviations defined by random numbers:** The development of this project in parallel with the one on visual feedback with cameras (more information on section 1.4) has made the use of that feedback impossible in terms of timing. To solve that problem, the deviations in the different directions have been defined by some random numbers, different for each of the executions. These random numbers have been created using a function found on the internet; the one shown on Section 4.5.11. Each time that function is called, a random number between 0 and 1 is outputted. In this case, the maximum deviation has been set to 20mm in each direction, giving a maximum deviation of $\sqrt{2} * 20$ in straight line.

Taking that into account, the procedure used is the following one:
First, the deviation in each direction will be calculated (with *calculate_error* procedure described on section 4.5.3; each of the variables can be seen graphically in Figure 4.5):

$$\Delta_y := |practice_y - theory_y| \tag{4.3}$$

$$\Delta_x := |practice_x - theory_x| \tag{4.4}$$

$$dev_i := \sqrt{\Delta_x^2 + \Delta_y^2} \tag{4.5}$$

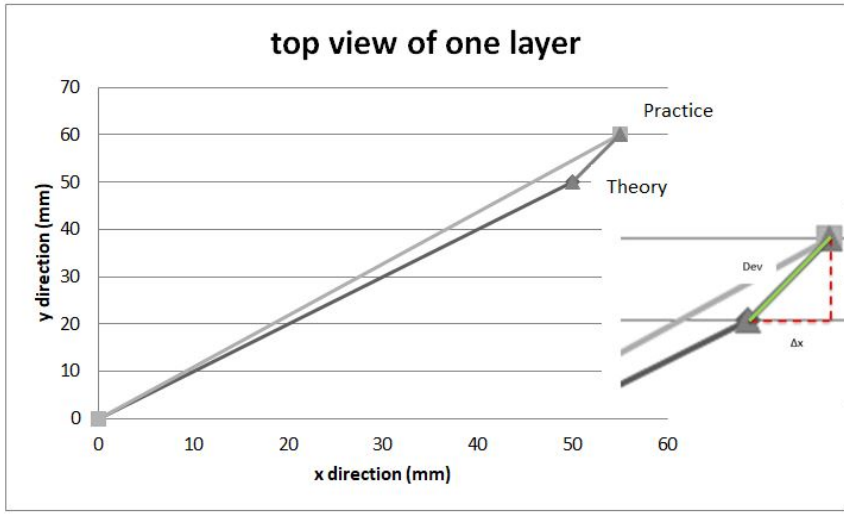


Figure 4.5: Top view of a layer with zoom in deviation

After, the target will be updated. For that, the radius of the final diameter of the filament will be subtracted to the actual position of the robot, as it can be seen on the following picture (Figure 4.6). The subtraction is first calculated in direction of the deviation and then it is projected on the x and y directions (with $alpha = 45$), as data has to be introduced separately.
The following equations (Equations 4.4 - 4.6) show mathematically what has been explained until here:

$$\text{In straight line : } subtraction = r_{final} \tag{4.6}$$

$$\text{In x direction : } subtraction_x = r_{final} * Cos(\alpha) \tag{4.7}$$

$$\text{In y direction : } subtraction_y = r_{final} * Sin(\alpha) \tag{4.8}$$

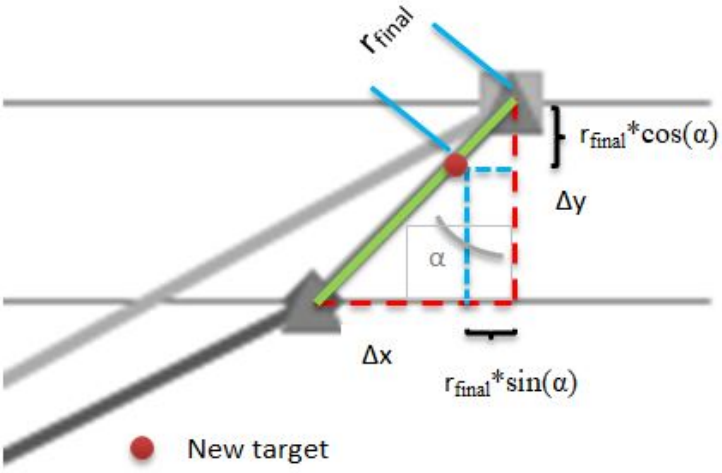


Figure 4.6: Zoom of deviation from top view with target updated

When the deviation is smaller than the final radius of the filament, the values will be directly updated to those theoretical ones (Equation 4.9), finishing here the iterative process.

$$\begin{cases} target_x = theoretical_x \\ target_y = theoretical_y \end{cases} \tag{4.9}$$

4.3 Getting started with RobotStudio

One important task is to learn how to use RobotStudio, as this will be the SW used along all the experiments (All information related to RobotStudio can be found in section **??**). A lot of tutorials of great relevance and help in this step of the procedure can be found in YouTube. That is why, in this case, YouTube has been used in order to get the basic background and knowledge of this SW, along with the manuals found in the page of ABB [25].

There are three important tasks that need to be fulfilled in RobotStudio to carry out the experiments; in the following pages, each of the tasks will be explained in detail, together with the steps that have to be followed in each of them.

Set the station

First, in order to obtain good results, it is important to set from the beginning the station as it is at the laboratory. This way, the results obtained here and at the laboratory will be based on the same coordinate frames and it will be possible to compare them without any changes.

Here are the main steps that have to be followed in order to set the station in RobotStudio. For a more detailed description, see [28] (Spanish tutorial, but content in English can also be found in YouTube).

1. Open RobotStudio
2. Click on *Empty station* (You will find a working area as the one shown in Figure 4.7)
3. Add a robot:
    - Click on *ABB Library* (upper-left side of the screen; this is one of advantages of this SW, as it comes with a wide library, where all type of ABB robots are ready to be launched to the station.)
    - Select the same robot you have at the laboratory (IRB140 in this case, you will end up with what is seen on Figure 4.8)
    - Set its position (The same as that at the laboratory)

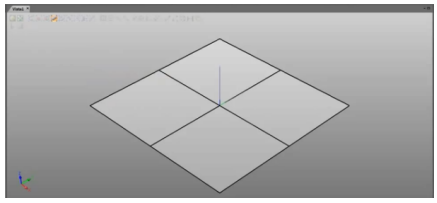


Figure 4.7: Empty Station in RobotStudio

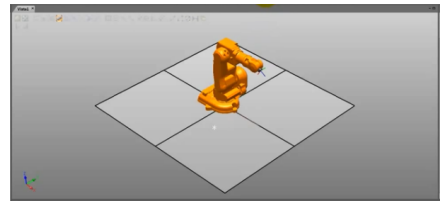


Figure 4.8: IRB140 on station

Until this point, what is shown in Figure 4.8 is only a piece of iron (mechanical arm), with no functioning capacity. For obtaining a proper robotic arm, the controller has to be added, by following some steps:

4. Click on *Robot System* → *From design*. (There are several options there, but just keep selecting Next until the end)

5. The controller will be started and a box will appear in the lower-right part of the screen. After some time, this box will be green-coloured with the text ”Controller status 1/1” inside it; meaning that the controller is on. (There is no need to put the controller, physical box, on the station, but the one used in here is the IRC5)

   From this point on, different type of movements are available, being the controller the one calculating the movement of each of the joints of the robot.

6. Add a tool:

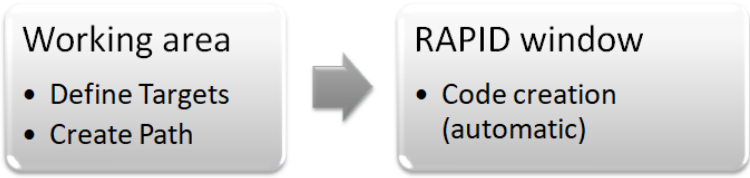
   - Click on *Import Library* → *Equipment*
   - Search for *Tools* and select the tool you will be using (In this case the first experiments will be done without any tool, in order to make all calculations easier. After that, when material deposition is used, the tool will have to be inserted to the robot on RobotStudio)

7. Save the station: Click on *File* → *Save as*; and save the station on the folder of your choice.

Create the path

Once the robot is set (mechanical part as well as its controller), it is time to create the path the robot will be following. There are two options to choose from at this point (that follow just the opposite order).

On the one hand, the path can be directly created on the working area. For this, different targets will be defined, and then, a path will be created by telling the robot the order in which those targets have to be reached.
This option has several advantages over the other one. It is a really visual approach and apart from that, once the path has been defined, the RAPID code is automatically created. After that, there is the option of changing or modifying the code in the RAPID window on RobotStudio. Combining the advantages just mentioned, this is an attractive approach for beginners to learn. For this reason, this option has been used at the beginning to get to know the SW and the programming language RAPID.



On the other hand, the path can also be created after a code programmed in RAPID language. In this case, it is first required to go to the RAPID window at RobotStudio and develop the code for the path. Once the code is finished, it can be simulated on the robot, through the VC (explained in the next task).
This approach is more difficult than the other one, as it requires that the person behind the computer knows how to program. But that difficulty is clearly compensated by the advantages that come linked to the self-creation of the code:
It is easier to read and modify a self-created code than that of another person. In this case, this is good for:

- Including procedures to extract data to a txt file.
- Changing coordinates of the path for different experiments.

Taking this into account, the second approach will be used in general as it results in more opportunities for extracting data and in an easier development and understanding of the code.

Simulate with VC

The last task to carry out in RobotStudio is to check if the code generated works as expected or not; for that, the VC is used (All information regarding the VC in RobotStudio is in section 3.4.2).
The steps that have to be followed to start a simulation are listed below:

1. Change window to *Controller*. See Figure 4.9
2. Open the *Control Panel*. See Figure 4.10 with explanation of each of the buttons there.

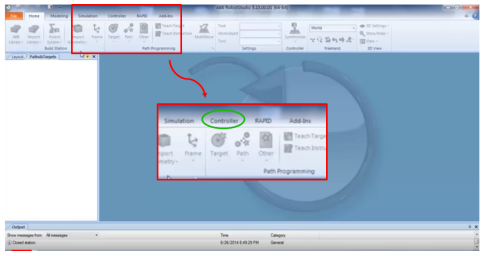


Figure 4.9: Controller button on RobotStudio

Figure 4.10: Control panel on RobotStudio

3. Open the *FlexPendant* (by clicking on FlexPendant). See Figure 4.11.

Figure 4.11: FlexPendant on RobotStudio

4. Click on *ABB* and the menu on Figure 4.12 will appear on the screen.

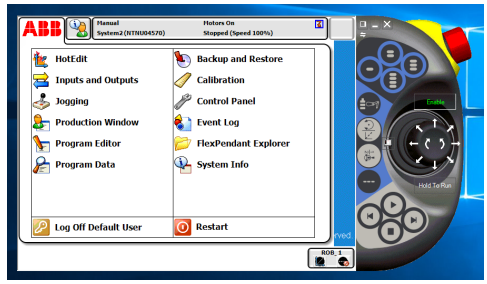


Figure 4.12: FlexPendant menu on RobotStudio

5. From here on, same steps as those explained in *Launch an execution* from section 4.4.

4.4 Getting started with IRB140

The robot used for the experiments is the IRB140 of ABB found at the robot laboratory at the department of Engineering Cybernetics at NTNU. All the information regarding this robot has already been introduced in Chapter 3, where its characteristics have been mentioned, and a mathematical model of it has been obtained.

In this case, as it was done previously with RobotStudio, there are also some important tasks that need to be done to start with the execution of the code on the IRB140.

Start the robot

First of all, there are some steps that have to be followed in order to start the robot in a correct way. All this steps are described in detail below:

1. Unlock the padlock: The robot has a code padlock on the start-up switch as a security measure.
2. Change the start-up switch to *ON* (Everything will be turned on in that moment).
3. Wait until the FlexPendant is started, see Figure 4.13 to see the welcome window of the FlexPendant.

Figure 4.13: Welcome window of FlexPendant at laboratory

4. Start of robot complete.

Load a module

After starting the robot, the module containing all the code has to be loaded to the FlexPendant. Here are the steps to follow in order to have the module loaded:

1. Insert the USB in the FlexPendant.

2. Click on *ABB* on the upper-left part of the screen, to open the menu seen in Figure 4.14.

Figure 4.14: Window with main menu of FlexPendant at laboratory

3. Click on *Program Editor* → *Modules*.

4. Locate the module saved on the USB (extension of the file must be .mod) and click on *Load*.

5. The module will be loaded on the FlexPendant.

IMPORTANT: Each time some part of the code is changed on the computer, the module has to be loaded again into the FlexPendant. Same steps as stated here will be followed for that.

Launch an execution

Once the previous steps have been done, it is time to start an execution. For that, the following steps have to be followed:

1. Security measure: There is a button at the right-hand side of the FlexPendant that needs to be pressed at all times during the execution so that the robot works. If this button is unpressed, the robot will stop.
2. Click on *Start* button. (Remember to maintain the security button pressed at all times for the robot to work)
3. The execution will start (Stay tuned to the screen of the FlexPendant and introduce all data asked there).

4.5 RAPID code

As mentioned in the previous section, a code has been developed in RAPID programming language for carrying out the experiments on this work. The entire code is shown in Appendix D.
But in order to get a deeper understanding of the code, this section of the report will be devoted to the explanation of the purpose of each of the procedures found on the code, together with the parameters used as an input as well as those outputted from each of them.

4.5.1 get_data

```
1 PROC get_data()
2     TPReadNum height, "Which is the height of the structure
  to build?";
3     TPReadNum D_initial_filament, "Which is the diameter of
  the filament without expansion?";
4     TPReadNum D_final_filament, "Which is the diameter of
  the filament once it has expanded?";
5     num_layers := height DIV D_final_filament;
6     TPWrite "Execution starts.";
7 ENDPROC
```

Information

- **Description:** Get from the user all the information needed for that experiment (initial and final diameter of the filament and the height of the wall) and calculate the number of layers needed to reach that height.
- **Input parameter:** None.
- **Return:**
  - height: variable of type num containing the height of the wall in mm.
  - D_initial_filament: variable of type num containing the initial diameter of the filament in mm.

- D_final_filament: variable of type num containing the final diameter of the filament in mm.
- num_layers: variable of type num containing the number of layers needed to obtaining an object of required height.

4.5.2 update_height

```
1 PROC update_height(num layer)
2     z_update := D_initial_filament + D_final_filament*(layer-1);
3     p10.trans.z := z_update;
4     p0.trans.z := z_update;
5 ENDPROC
```

Information

- **Description:** Calculate the value in z direction for next layer and update the value for both ends of the wall.
- **Input parameter:**
  - layer: variable of type num containing the number of the layer for that iteration. This is used to calculate the new z position.
- **Return:**
  - z_update: variable of type num containing the new value for z coordinate.
  - p10.trans.z: variable of type num inside the variable p10 of type robtarget, containing the new value for the z coordinate.
  - p0.trans.z: variable of type num inside the variable p0 of type robtarget, containing the new value for the z coordinate.

4.5.3 calculate_error

```
1 PROC calculate_error(num i)
2     delta_y := Abs(p10.trans.y - p10_theoretical.trans.y);
3     delta_x := Abs(p10.trans.x - p10_theoretical.trans.x);
4     dev{i} := Sqrt(Pow(delta_x,2) + Pow(delta_y,2));
```

```
5       TPWrite "The deviation in this layer is:" \Num:=dev{i};
6   ENDPROC
```

Information

- **Description:** Calculate different parameters of the error in path deviation.
- **Input parameter:**
  - i: variable of type num containing the number of the layer for that iteration. This is used to know the position where the deviation has to be saved.
- **Return:**
  - delta_y: variable of type num containing the path deviation in y direction.
  - delta_x: variable of type num containing the path deviation in x direction.
  - dev{i}: variable (array) of type num containing the path deviation in straight line.

4.5.4 check_deviation

```
1   PROC check_deviation(num layer)
2       IF dev{layer} > (D_final_filament/2) THEN
3           TPWrite "The new p10 is:"\Pos:=p10.trans;
4           alpha := ATan(delta_y/delta_x);
5           p10 := Offs(p10, -D_final_filament*Cos(alpha)/2,
    D_final_filament*Sin(alpha)/2,0);
6           TPWrite "The new p10 is:"\Pos:=p10.trans;
7       ELSE
8           p10 := p10_theoretical;
9       ENDIF
10  ENDPROC
```

Information

- **Description:** Check if deviation calculated in *calculate_error* is bigger than the radius of the filament (see WHICH SECTIONNNNN for detailed explanation). In case it is, calculate the new coordinates for that point. If smaller, equal the coordinates of that point to those of its theoretical position.

- **Input parameter:**
  - layer: variable of type num containing the number of the layer for that iteration. This is used to know the deviation that has to be checked each time.
- **Return:**
  - alpha: variable of type num containing the angle formed by the path deviation with the x direction in degrees.
  - p10: variable of type robtarget containing the new coordinates for that point.

4.5.5 Path_0_10

```
1  PROC Path_0_10(num layer)
2       update_height(layer);
3       MoveL p0,v100,fine,tool0\WObj:=wobj0;
4       p0 := Crobt (\Tool:=tool0 \WObj:=wobj0);
5       z_array{layer} := p0.trans.z;
6       height_error{layer} := z_array{layer} - z_update;
7       TPWrite "the current position is: "\Pos:=p0.trans;
8       MoveL p10,v100,fine,tool0\WObj:=wobj0;
9       p10 := Crobt (\Tool:=tool0 \WObj:=wobj0);
10 ENDPROC
```

Information

- **Description:** Tell the robot where to move and get feedback from it on where has he actually moved, in order to calculate the error. This is half of the path, shown in Figure **??** in red.
- **Input parameter:**
  - layer: variable of type num containing the number of the layer for that iteration. This is used to know in which position of the array save the z position as well as the height error in that case.
- **Return:**
  - z_array$\{layer\}$: variable (array) of type num containing the height in mm for each iteration.

- height_error{$layer$}: variable (array) of type num containing the height error in mm for each iteration.

4.5.6 Path_10_0

```
1 PROC Path_10_0(num layer)
2     update_height(layer);
3     MoveL p10,v100,fine,tool0\WObj:=wobj0;
4     p10 := Crobt (\Tool:=tool0 \WObj:=wobj0);
5     z_array{layer} := p10.trans.z;
6     height_error{layer} := z_array{layer} - z_update;
7     TPWrite "the current position is: "\Pos:=p10.trans;
8     MoveL p0,v100,fine,tool0\WObj:=wobj0;
9 ENDPROC
```

Information

- **Description:** Tell the robot where to move and get feedback from it on where has he actually moved, in order to calculate the error. This is the other half of the path, shown in Figure **??** in blue MISSING THE FIGURE!!.
- **Input parameter:**
  - layer: variable of type num containing the number of the layer for that iteration. This is used to know in which position of the array save the z position as well as the height error in that case.
- **Return:**
  - z_array{$layer$}: variable (array) of type num containing the height in mm for each iteration.
  - height_error{$layer$}: variable (array) of type num containing the height error in mm for each iteration.

4.5.7 open_file

```
1  PROC open_file()
2      Open "LOCATION", executionfile \Append;
3      Write executionfile, "Execution started.";
4      Write executionfile, " Total Height (mm): "\Num:=height;
5      Write executionfile, " Initial diameter of the filament
   (mm): "\Num:=D_initial_filament;
6      Write executionfile, " Final diameter of the filament
   (mm): "\Num:=D_final_filament;
7      Write executionfile, " Number of layers: "\Num:=num_layers;
8      Write executionfile, " Random deviation created in this
   execution: ";
9      Write executionfile, "   x : "\Num:= delta_x;
10     Write executionfile, "   y : "\Num:= delta_y;
11     Write executionfile, "Iteration : Path deviation :
   Length(x) : Length error(Δx) "\NoNewLine ;
12     Write executionfile, ": Width(y) : Width error(Δy):
   Height(z) : Height error(Δz)";
13 ENDPROC
```

Information

- **Description:** Open a .txt file and write some general information about the experiment. The location of the text file is different if the code is used in RobotStudio or in the robot at the laboratory, so the code must be changed in each situation. In the case of this project, the locations used are the following ones:
  - RobotStudio: LOCATION → d:/execution.txt (USB)
  - Laboratory: LOCATION → /hd0a/noratest/execution.txt (FlexPendant memory)

  It is also important to write the term *Append* at the end of the *Open* instruction. This way, if the file does not exist it will create it at the location mentioned before. But, in the case the file already exists, it will continue adding data at the end of the file. This way, more than one execution can be stored in the same text file.

- **Input parameter:** None.

- **Return:**
  - execution.txt: Text file with general information that will be filled with data later during the execution.

4.5.8 write_to_file

```
1  PROC write_to_file(num i)
2      Write executionfile, ""\Num:=i\NoNewLine;
3      Write executionfile, " : "\Num:=dev{i}\NoNewLine;
4      Write executionfile, " : "\Num:=(p10.trans.x-home.trans.x)
   \NoNewLine;
5      Write executionfile, " : "\Num:=delta_x\NoNewLine;
6      Write executionfile, " : "\Num:=(p10.trans.y-home.trans.y)
   \NoNewLine;
7      Write executionfile, " : "\Num:=delta_y\NoNewLine;
8      Write executionfile, " : "\Num:=z_array{i}\NoNewLine;
9      Write executionfile, " : "\Num:=height_error{i};
10 ENDPROC
```

Information

- **Description:** Fill the text file with data obtained from each iteration (layer).
- **Input parameter:**
  - i: Variable of type num containing the number of the layer for that iteration. This is used to know the position of the different arrays containing data that has to be checked each time.
- **Return:**
  - execution.txt: Text file filled with data until iteration number *i*.

4.5.9 close_file

```
1  PROC close_file()
2      Write executionfile, "Execution finished.";
```

```
3       Close executionfile;
4   ENDPROC
```

Information

- **Description:** Close the .txt file once the execution has finished.
- **Input parameter:** None.
- **Return:**
  - execution.txt: Text file already filled will all data from that execution of the code.

4.5.10 initialize_variables

```
1   PROC initialize_variables()
2       delta_y := RANDOM() * 20;
3       delta_x := RANDOM() * 20;
4       open_file;
5       p10_theoretical := Offs(home,100,100,0);
6       p10 := Offs(p10_theoretical,delta_x,-delta_y,0);
7       p0 := home;
8       iteration := 2;
9       FOR i FROM 1 TO (num_layers+1) DO
10          dev{i} := 0;
11          z_array{i} := 0;
12          height_error{i} := 0;
13      ENDFOR
14  ENDPROC
```

Information

- **Description:** Initialization of all variables. The piece of code inside the *FOR* command is used to set all variables to 0. This is an important step to do in order to have all arrays free of values each time a new experiment is done.
  Imagine the case where in a first experiment the number of layers is 21 and in the following experiment it decreases to 20 (due to a change in the requirements of that

new experiment). When exporting the data, it might happen that the 21st entry of each array is filled with data from the first experiment and not from the second (as there is no new information for that 21st entry). Being this the case, when the data is processed and used in a graph, an important error will be introduced and so the conclusions obtained from there are not going to be valid.

- **Input parameter:** None.
- **Return:** All variables have already been introduced previously.

4.5.11 RANDOM (Taken from the internet)

```
1  !Function to generate random numbers for deviation taken from:
2  !https://forums.RobotStudio.com/discussion/3742/random-value-needed
3  FUNC NUM RANDOM()
4      VAR num val:=1;
5      VAR pos pos_current;
6      pos_current := CPos(\Tool:=tool0 \WObj:=wobj0);
7      !1 seeding, Seed by position
8      val := val*(Abs(ROUND((pos_current.x*100) \Dec:=0)) +1);
   !+1 to avoid mul by 0 scenario
9      val := 789 + val MOD 1000;
   !777 is just a bogus valu, since the mod might give 0
10     !Feel free to seed with y,z, more
11     !2. seeding, Seed by time
12     val := val * (GetTime(\Sec)+1);
13     val := val * (GetTime(\Min)+1);
14     !3 seeding, increment callcount and handle large nuber
15     callCount := callCount +1;
16     IF callCount > 1000 THEN
17         callCount := 234;
18     ENDIF
19     !finally a division to get something interesting large
   float number
20     val := val / callCount;
21     !get a value between 0 and 1 using COS ( math)
22     RETURN (0.5 + COS(val)/2);
```

```
23 ENDFUNC
```

Information

- **Description:** Creation of a random number ranging from 0 to 1. This function has been obtained from the internet.
- **Input parameter:** None.
- **Return:**
  - RANDOM(): Variable of type num containing the value of the random number created.

4.5.12 main

```
1  PROC main()
2      TPErase;
3      get_data;
4      MoveL home,v100,fine,tool0\WObj:=wobj0;
5      initialize_variables;
6      FOR i FROM 1 TO (num_layers + 1) DO
7          IF (i MOD 2) = 1 THEN
8              Path_0_10(i);
9          ELSE
10             Path_10_0(i);
11         ENDIF
12         calculate_error(i);
13         write_to_file(i);
14         check_deviation(i);
15     ENDFOR
16     finish_point:=CRobT(\Tool:=tool0 \WObj:=wobj0);
17     finish_point:=Offs(finish_point, 0, 0, 50);
18     MoveL finish_point,v100,fine,tool0\WObj:=wobj0;
19     close_file;
20 ENDPROC
```

Information

- **Description:** Call all the other procedures in the right order.
- **Input parameter:** None.
- **Return:** None.

Chapter 5

Results

This chapter will be devoted to show the different results that have been obtained from the experiments explained in Chapter 4. Apart from that, those results will be analyzed and discussed to see whether the code developed has worked or not.
For that, this chapter will be organized as follows:

- **Data collection:** Steps followed to gather and process all data obtained from the robot.
- **Summary of experiments:** Tables summarizing all the important features of each of the experiments.
- **Results from RobotStudio:** Analysis and comments of the results obtained in RobotStudio.
- **Results from IRB140:** Comparison of results obtained in the laboratory on the IRB140 with those obtained in RobotStudio.

5.1 Data collection

This first part of Chapter 5 will be used to introduce all the steps that have been followed in order to collect data from the robot as well as to process it afterwards.

5.1.1 Gather data in a text file

In order to analyze the behaviour of the robot the first step is to collect data from it. For that, RAPID gives the option to export data to a text file, as it can be seen in the procedures *open_file, close_file and write_to_file* described in section 4.5.

The first step to do so is to open the file with the command ***Open***. After that, the command ***Write*** is used each time new data wants to be stored. Finally, ***Close*** is used to close the file in that execution.

It is important to store the data in a way such that it is possible to export it to excel later as a table. In this case, the symbol ":" has been used each time a new column was needed. Having said that, Figure 5.1 shows the structure of the file, without all data from iterations.

```
Execution started.
 Total Height (mm) :    100
 Initial diameter of the filament (mm) : 2
 Final diameter of the filament (mm) : 5
 Number of layers : 20
 Random deviation created in this execution
   x :  2,8986
   y :  0,000144243
Iteration : Path deviation : Length(x) : Length error(Δx) :  Width(y) : Width error(Δy) : Height(z) : Height error(Δz)
1
2
.
.
.

Execution finished.
```

Figure 5.1: Structure of text file without data

5.1.2 Import the text file to excel

The information on the text file is not well organized, and thus, it is difficult to analyze and compare the results from different executions. To make the results more visual, all the text files have been imported to excel. Below, the different stages of the process followed have been listed:

1. Open Excel.

2. Click on *File* → *Open* (A dialog box will open).

3. Select *Text Files* from the options on the lower-right part of the dialog box.

4. Locate and open the text file you want to open (Another dialog box will open).

5. Follow all the steps on the dialog box: (All sequence shown in Figure 5.2)

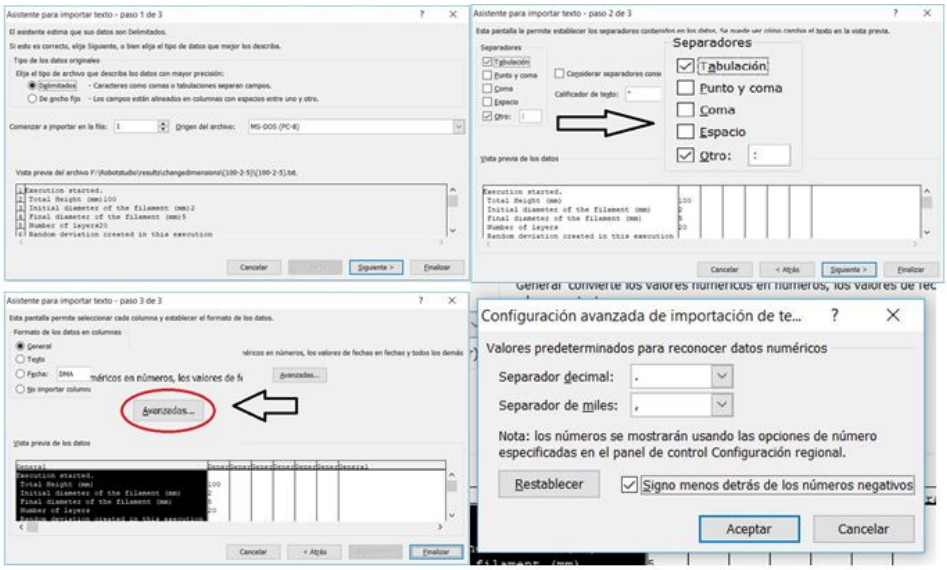


Figure 5.2: Sequence to import data in the correct way

6. Click on *Finish* (All data will be imported).

Once that process has been followed, all data will already be available in excel files, and its appearance will be the one shown in Figure 5.3

Execution started.							
Total Height (mm)	100						
Initial diameter of the filament (mm)	2						
Final diameter of the filament (mm)	5						
Number of layers	20						
Random deviation created in this execution							
x	2,8986						
y	0,000144243						
Iteration	Path deviation	Length(x)	Length error(Δx)	Width(y)	Width error(Δy)	Height(z)	Height error(Δz)
1	2,79448	102,791	2,79071	99,855	0,145035	1,75446	-0,245536
2	0,30658	100,305	0,305481	99,9741	0,0259323	6,81412	-0,185878
3	0,165985	99,903	0,0970459	99,8653	0,134659	11,8403	-0,159747
4	0,0630397	99,9376	0,062439	99,9913	0,00868225	16,8101	-0,189865
5	0,186347	99,8885	0,111511	99,8507	0,1493	21,8403	-0,159742
6	0,061719	99,9388	0,0611572	99,9917	0,00830841	26,8115	-0,188543
7	0,173307	99,899	0,100952	99,8591	0,140869	31,8412	-0,15877
8	0,0758657	99,9248	0,0751953	99,9899	0,0100632	36,7122	-0,287796
9	0,161806	99,9034	0,0965576	99,8702	0,129837	41,8412	-0,158752
10	0,0731214	99,9275	0,0725098	99,9906	0,00943756	46,7161	-0,283943
11	0,148961	99,9144	0,0856323	99,8781	0,121887	51,7504	-0,24958
12	0,0713699	99,9292	0,0708008	99,991	0,00899506	56,7178	-0,282181
13	0,220027	99,866	0,133972	99,8255	0,174538	61,7494	-0,250591
14	0,0696788	99,9308	0,0691528	99,9915	0,00854492	66,7196	-0,280418
15	0,165457	99,9036	0,0963745	99,8655	0,134491	71,7499	-0,250137
16	0,0679912	99,9325	0,0675049	99,9919	0,00811768	76,7213	-0,278748
17	0,204244	99,8798	0,120239	99,8349	0,1651	81,7503	-0,24968
18	0,0654407	99,935	0,0650024	99,9924	0,00756073	86,7248	-0,275238
19	0,164964	99,9061	0,0938721	99,8643	0,135651	91,7508	-0,249222
20	0,0637565	99,9366	0,0633545	99,9929	0,00714874	96,7264	-0,273598
21	0,135377	99,9246	0,0753784	99,8876	0,11245	101,75	-0,250015
Execution finished.							

Figure 5.3: Structure of excel file

As a lot of executions have been performed, the following excel files have been created:

- ChangeDimensions.xlsx → Excel file containing all data from different executions where dimensions of the material as well as of the final object are different. Inside this file, the following pages can be found:
  - (100-2-5)
  - (200-1-2)
  - (300-1-2)
- ChangeVelocity.xlsx → Excel file containing all data from different executions where printing velocity is different. Inside this file, the following pages can be found:
  - (v10)
  - (v100)
  - (v1000)

5.1.3 Make graphs using Matlab

The last step is to make all necessary graphs using matlab. Excel is also a good tool for making all the graphs, but in this case, matlab has been chosen to do this part just for a matter of personal preference.

Graphs of 3D views

In order to see the path deviation in the different directions and be able to comment and compare later the results from the different executions, a 3D reproduction of the path followed by the robot, as well as its views have been plotted. Apart from that, the theoretical path has also been shown in the graphs (path in red), to be able to see the error and how it was corrected in the end. The piece of code used for obtaining these graphs is shown in Appendix B (*Matlab Code → Graphs with object in 3D and its views*).

By changing the name of the file, the name of the page and the piece of data, the different graphs shown in Appendix C have been obtained. In Figure 5.4 an example is shown.

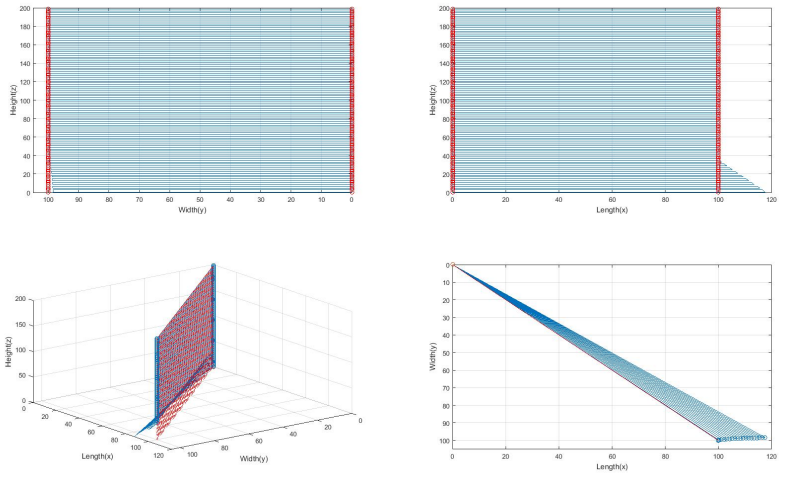


Figure 5.4: Example of graphs with object in 3D and its views

Graphs of deviation

This second set of graphs show the different deviations, the path deviation in straight line on the one hand, and the height deviation on the other. The piece of code used for obtaining these graphs is shown in Appendix B (*Matlab Code → Graphs with path and height deviations*).

By changing the different data as mentioned in the previous part, the different graphs shown in Appendix C have been obtained. In Figure 5.5 and example is shown.

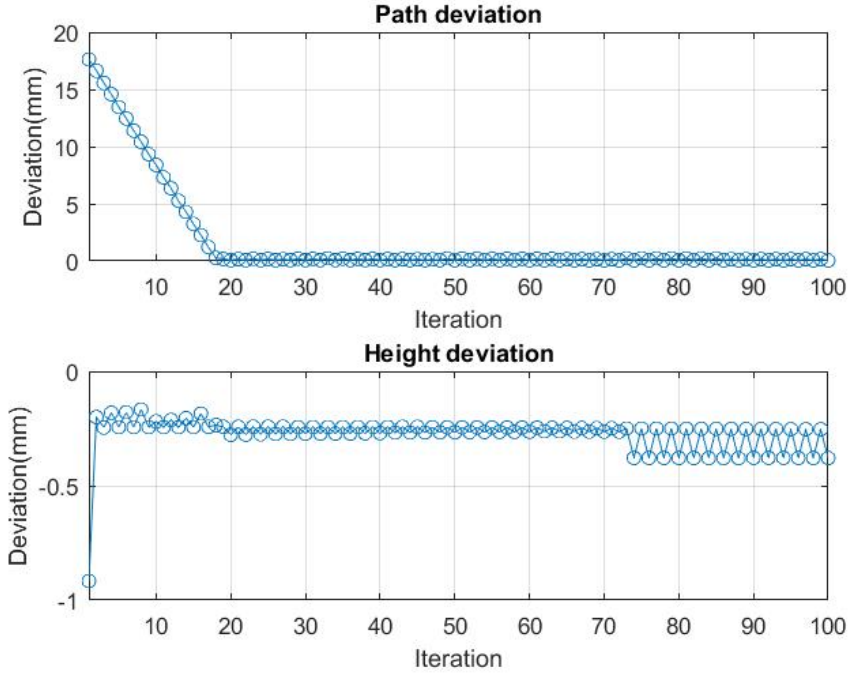


Figure 5.5: Example of graphs with path and height deviations

5.2 Summary of experiments

Due to the big amount of experiments performed, the most important features of each of them have been gathered in the following tables (Table 5.1 and Table 5.2 for experiments executed on RobotStudio and Table 5.3 for those on IRB140). This way, each of the experiments will be named after the group name followed by the number of that experiment ((v100).3 for example) and the reader will be able to easily locate the experiment mentioned or analyzed when commenting the results in the following sections of this chapter. Apart from that, all graphs from all experiments are shown in Appendix C.

Table 5.1: Summary of experiments performed with change of dimensions and constant velocity on RobotStudio

Change of dimensions		Constant velocity					
Group name	**Number of experiment**	**Printing velocity**	**Height of object**	**Initial diameter**	**Final diameter**	**Random deviation**	
						x	**y**
(100-2-5)	1	v100	100	2	5	2,8986	0,000144243
	2					10,9582	8,02209
	3					1,08989	13,0901
	4					14,8172	3,30855
(200-1-2)	1		200	1	2	10,9582	19,9452
	2					7,23166	2,12819
	3					19,4154	17,071
	4					17,6604	1,49782
(300-1-2)	1		300	1	2	19,7237	9,99988
	2					7,24378	0,46411
	3					13,0901	12,5657
	4					0,659393	0,321151

Table 5.2: Summary of experiments performed with change of velocity and constant dimensions on RobotStudio

Constant dimensions					Change of velocity		
Group name	**Number of experiment**	**Printing velocity**	**Height of object**	**Initial diameter**	**Final diameter**	**Random deviation**	
						x	**y**
(v10)	1	v10	100	2	5	2,92853	10,0011
	2					20	5,00039
	3					17,6607	5,00042
	4					19,7814	0,21849
(v100)	1	v100				2,8986	0,000144243
	2					10,9582	8,02209
	3					1,08989	13,0901
	4					14,8172	3,30855
(v1000)	1	v1000				18,0905	13,0912
	2					19,1354	5,7418
	3					0,340787	2,72551
	4					1,14133	6,11764

Table 5.3: Summary of experiments performed on IRB140

Group name	Number of experiment	Printing velocity	Height of object	Initial diameter	Final diameter	Random deviation	
						x	y
Constant dimensions		**Change of velocity**					
(100-2-5)	2	v100	100	2	5	10,9582	8,02209
(200-1-2)	1		200	1	2	10,9582	19,9452
(300-1-2)	1		300	1	2	19,7237	9,99988
Constant dimensions		**Change of velocity**					
(v10)	1	v10	100	2	5	2,92853	10,0011
(v100)	2	v100				10,9582	8,02209
(v1000)	2	v1000				19,1354	5,7418

5.3 Results from RobotStudio

In this first method of executing the code, the focus will be first on the good performance of the code; and once that is verified, on the effect it has the change of different variables on the performance of the robot.

5.3.1 Checking good performance of the code

Before executing the different experiments and commenting data obtained from there, the good performance of the code was checked. For that, different executions were launched on RobotStudio and these times the focus was on the following things:

First of all, programming errors were analyzed. Some time was spent so that no error was found on the code when compiling it. Here, syntax and semantic errors were found mostly. The way to correct them and obtain a code with no compilation error was different depending the type of error.
The syntax errors were the easiest to correct due to the big amount of information available on the internet with regard to RAPID language. This way a simple search of the instruction with the error on the internet was enough to find the cause of the error.
On the contrary, the logical errors (errors where the specification is not respected) were the most difficult ones. To correct this, a deep understanding of the code is needed as well as the ability to follow the change of indexes while execution is going on.

Once the compilation was completed without errors, it was time to confirm that the function of the code was correct. That is, that the robot was moving to the targets in a good way, as well as in the good order. This was checked in a superficial way, visually, as no data was recovered in these trials. To see so, the first set of graphs of each of the execution was analyzed. These graphs can be found on Appendix C, but below, in figures 5.6 and 5.7 two examples of it are shown. There, one can see how the path required is recovered after some time, once the iterative process for correcting the error has concluded.

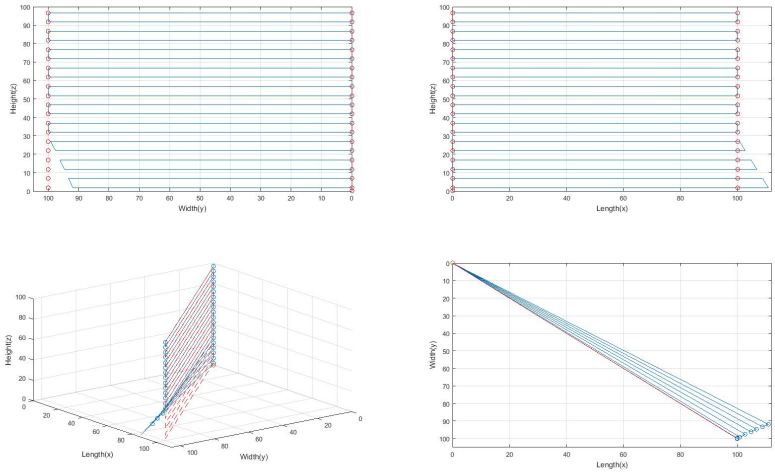


Figure 5.6: Graphs of the object in 3D and its views for experiment (100-1-2).2

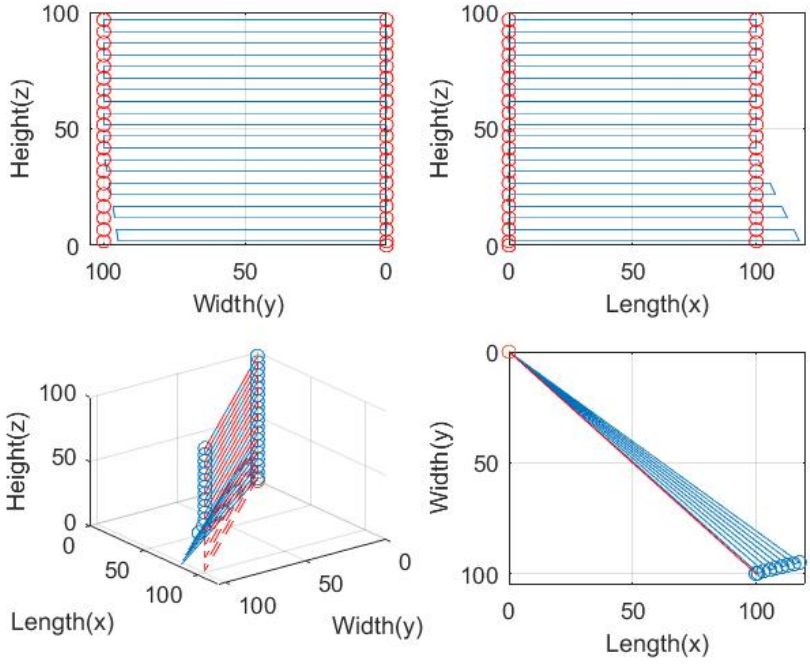


Figure 5.7: Graph of height deviation for experiment (v10).3

5.3.2 Effect of change of dimensions

There are three dimensions that were changed along the experiments to see the effect of them. The first one was the height of the final object and the other two were the initial and final diameters of the filament.
First of all, two experiments will be analyzed, each of them with a different final height of the object but same initial and final diameters of the filament. The following table (Table 5.4) is a summary of the important characteristics and data of these experiments chosen, those that will be used to see the effect of the final height of the object in the height deviation. (Experiments with similar deviation in straight line will be chosen, so that these random numbers do not influence on what is being studied)

Table 5.4: Summary of experiments performed under same velocities and similar random numbers but different dimensions

Experiment name	Random deviation		Deviation in straight line	Maximum height deviation
	x	y		
(200-1-2).1	10,9582	19,9452	22,75726588	0.379303
(300-1-2).1	19,7237	9,99988	22,1138405	1.53729

Looking at the maximum deviation in both experiments it can be concluded that the height of the final object has a big impact on the height deviation, suffering this last variable an important increase when the height is increased. The cause of this can be explained in two different ways. First, the robot introduces a new error in each iteration, which makes the total error to increase. If the object is taller, then more iterations are needed to print the object, and more iterations end up causing a bigger error. Second, the height correction approach suggested and implemented in this thesis report is completely theoretical, as no feedback data from the robot is used when updating it. By using feedback from the robot to update the height, this error could also be minimized, or even eradicated.

Apart from that, it is interesting to mention how all experiments follow the same pattern in terms of height deviation. This can be seen in the graphs shown below (Figure 5.8 and Figure 5.9), as well as in those in Appendix C (first part called *Change of dimensions*; second set of graphs for each execution).

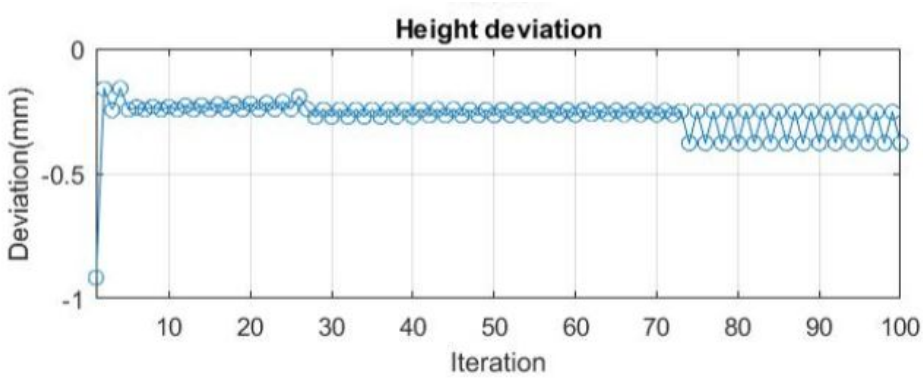


Figure 5.8: Graph of height deviation for experiment (200-1-2).1

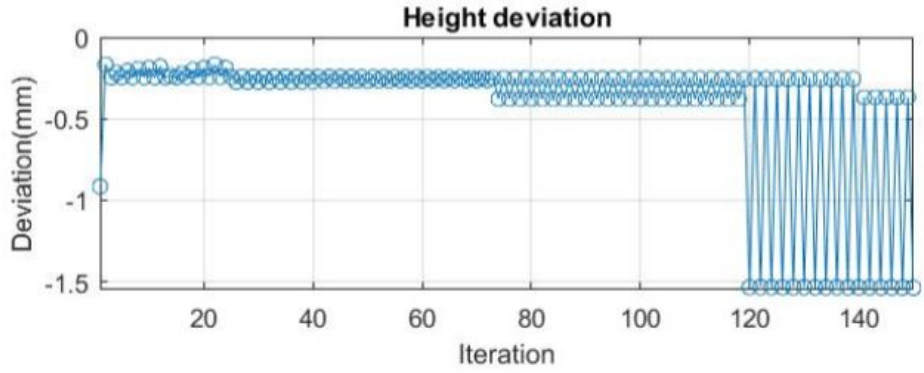


Figure 5.9: Graph of height deviation for experiment (300-1-2).1

The change of the initial and final diameter of the filament did not have any noticeable effect on any of the experiments performed. However, it is advisable to make some trial experiments every time when new dimensions for these diameters are to be used.

5.3.3 Effect of change of velocity

It has already been mentioned before that the printing velocity has a direct effect on the behaviour of the robot as well as on that of the material. Apart from that, it is also known that when working with materials that expand the printing velocity can not be very high, as the material of a layer would not have time to expand before next layer was deposited and then, some material would be deposited in collision with the previous one.

Moreover, faster movements should suffer more from disturbances and thus, the deviation should have a proportional behaviour with the velocity (increasing when the printing velocity is increased).

To see if all those assumptions are true or not, one experiment will be analyzed for each of the different printing velocities chosen (three in total). The experiments have been chosen to have as similar as possible the deviation, so that the results are not affected by that. The following table (Table 5.5) summarizes the key features of these experiments.

Table 5.5: Summary of experiments performed under same dimensions and similar random numbers but different velocities

Experiment name	Random deviation		Deviation in straight line	Average height deviation
	x	y		
(v10).3	17,6607	5,00039	18,35495913	0,264853095
(v100).4	14,8172	3,30855	15,18209205	0,225063286
(v1000).2	19,1354	5,7418	19,98883	0,191194524

In contrast to what it was thought, surprisingly, the height deviation gets bigger when the printing velocity gets smaller. Even if the table above (Table 5.5) just one experiment is shown for each of the velocities, this fact has been analyzed in all rest of the experiments, obtaining the same results. No theoretical explanation has been found for this.
But, even if faster velocities give smaller deviations, this higher velocities will cause collision of material from one layer to the other, as no time for expansion is let.

Looking at the graphs in Figures 5.10 - 5.12 another conclusion can be obtained. When the printing velocity is (v10), the slowest, the height deviation gets constant (even though quite big). On the contrary, when the printing velocity is the biggest (v1000), it is not constant at any time, and it gets more and more irregular and changing with the increase of the iterations.
It is thought to be easier to handle a higher but constant deviation, rather than a smaller but irregular one. In the first case, some offset could be introduced at the beginning of the printing in order to get rid of that error.

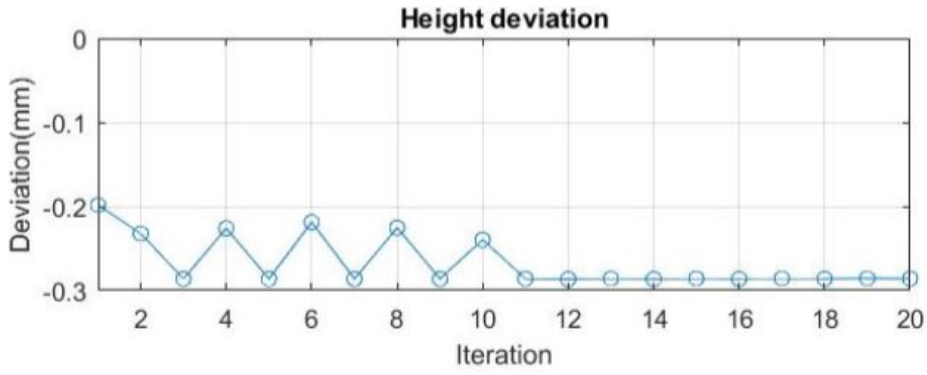


Figure 5.10: Graph of height deviation for experiment (v10).3

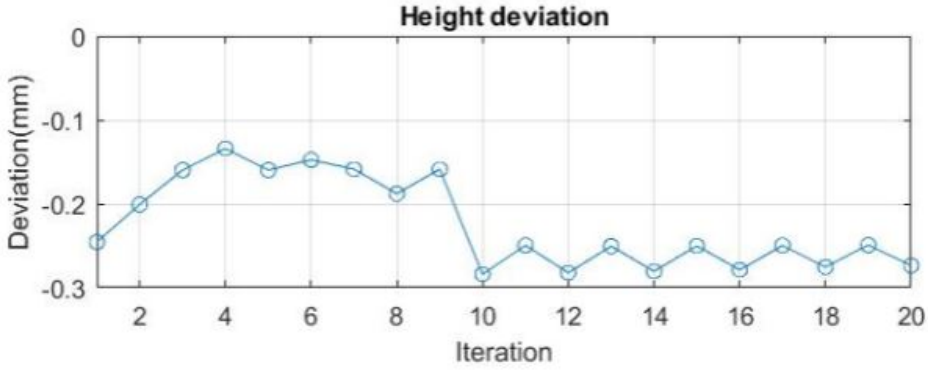


Figure 5.11: Graph of height deviation for experiment (v100).4

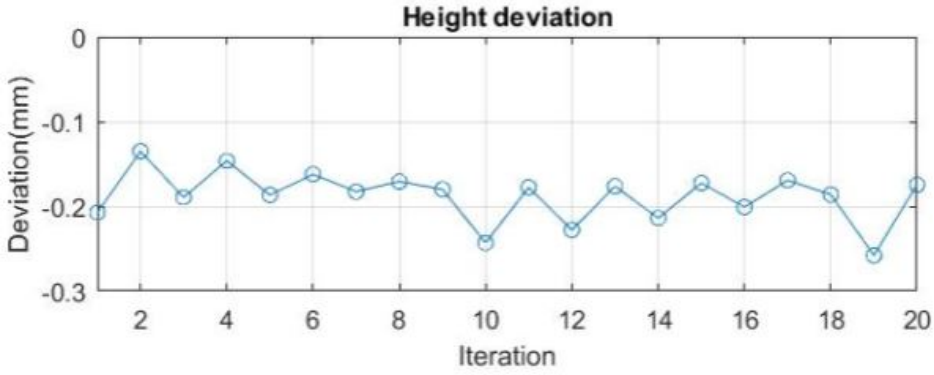


Figure 5.12: Graph of height deviation for experiment (v1000).2

5.3.4 Effect of change of deviation (Random numbers)

In this last case, the effect of the deviation will be analyzed. It has already been said that it has been impossible to use the visual feedback from the cameras in this report, due to the parallel development of both projects. Instead, some random numbers have been used in each execution to create that path deviation, as explained in section 4.2.2.

To see the effect of this random numbers, four different experiments will be analyzed, all of them with the same dimensions and with the same printing velocity, that is, under the same conditions. In this case, the executions chosen are those done with dimensions (100-2-5) and with (v100) printing velocity.

In the following table (Table 5.6, a summary with the different important features of each of the experiments are shown. Some of the data has been directly extracted from the excel tables, and other characteristics have been or calculated or obtained visually from the graphs.
The Random deviations in x and y directions have been obtained from the excel files. The deviation in straight line has been obtained in a rough way form the graphs, by watching the path deviation at Iteration=0; but it has also been calculated mathematically as shown in Equation 4.5. Last, the number iterations needed to regain the theoretical path has been obtained visually from the graphs (Looking at Figures 5.13 - 5.16 when the path deviation is less than the final radius of the filament, that is 2,5mm in this case).

Table 5.6: Summary of experiments performed under same conditions but different random numbers

Experiment name	Random deviation		Deviation in straight line	Iterations to correct path
	x	y		
(100-2-5).1	2,8986	0,000144243	2,898600004	3
(100-2-5).2	10,9582	8,02209	13,580744	7
(100-2-5).3	1,08989	13,0901	13,1353941	7
(100-2-5).4	14,8172	3,30855	15,1809205	7

Some conclusions can be obtained from the table shown above, as well as from the graphs of path deviation shown in the next Figures 5.13 - 5.16.

First of all, and the most important one, the number of iterations needed to correct the path is proportional to the path deviation calculated in straight line, not to the deviations in x and y directions. This means that the code works the same way in both x and y directions and that no matter if the path deviation is bigger in one or the other, when the deviation in straight line is similar, the correction will be done in the same number of iterations.

Apart from that, it is also worth mentioning how the correction of the path is done linearly. This can be seen in the Figures 5.13 - 5.16, where linear decrements of the deviations are shown. Linked to this, those graphs also show that once the deviation is less than the final radius of the filament, it is not maintained constant. Instead, it increases and decreases in a repetitive way (Figure 5.13 is the most representative for this case, as it has the smallest deviation and so the scale is bigger). This will be understood a lot better in next section (Section 5.3.5).

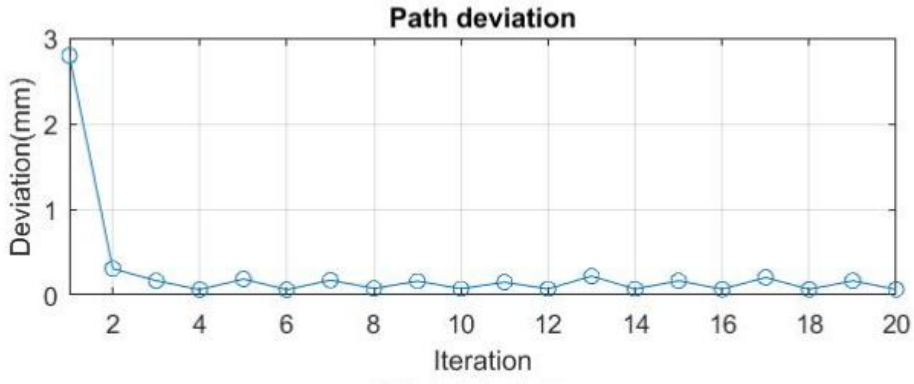


Figure 5.13: Graph of path deviation for experiment (100-2-5).1

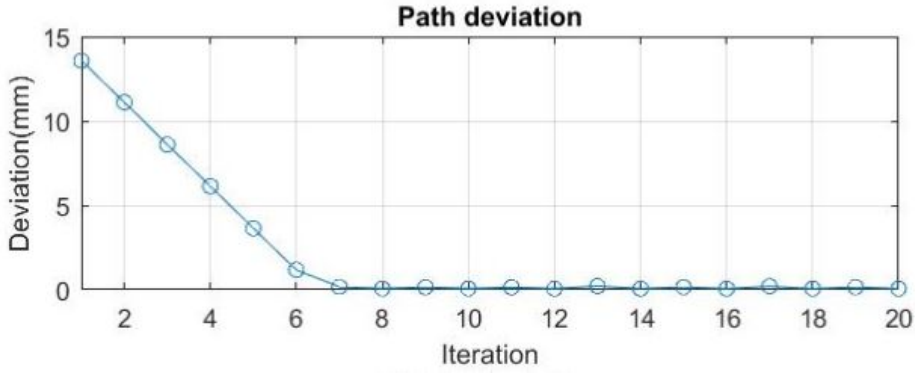


Figure 5.14: Graph of path deviation for experiment (100-2-5).2

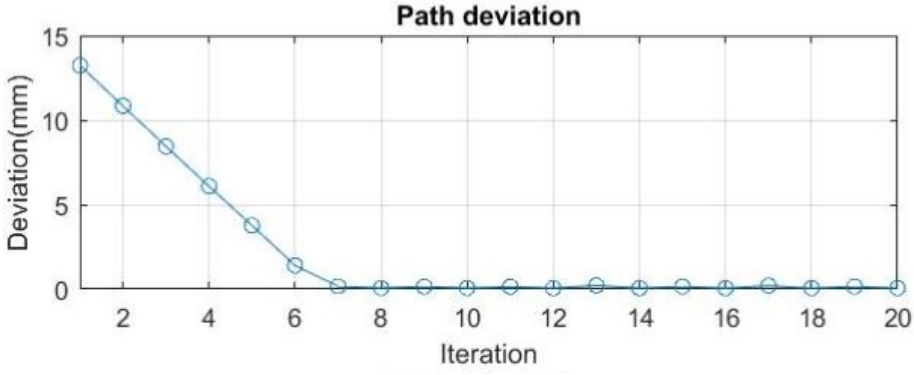


Figure 5.15: Graph of path deviation for experiment (100-2-5).3

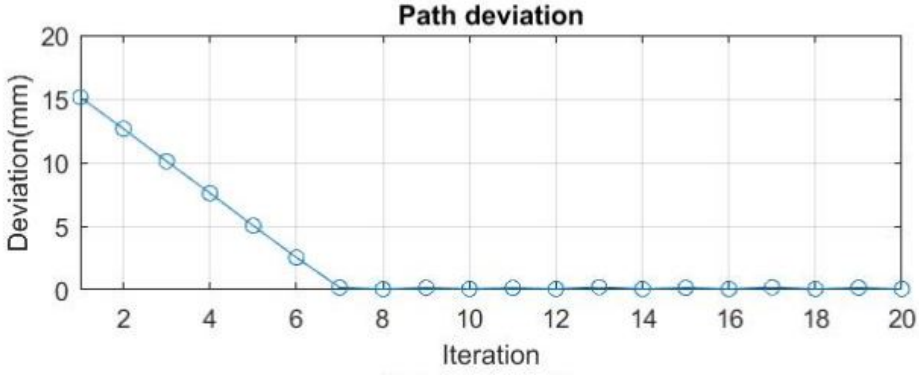


Figure 5.16: Graph of path deviation for experiment (100-2-5).4

5.3.5 Effect of use of *fine* in *MoveL* instructions

There are several options when determining how a position will be terminated, and this is defined in the ***MoveL*** instruction by the argument *zonedata*. In all the executions performed for this thesis report, ***fine*** has been used as the argument for zonedata (as it can be seen in all the ***MoveL*** instructions found on the code in Appendix D). This choice was done for two reasons:

- The robot must reach the target position before continuing with the program execution. This way, the data taken would be the correct one each time, without reading the position of the robot before reaching the target.
- The target will be reached in an exact way, that is, the robot will not change its direction until the exact point is reached.

But when analyzing the data obtained from all the executions, it has been seen that the data read and extracted to the text files was never the exact position where the robot was asked to go to. But instead, it was always a few μm smaller and surprisingly, it followed the same pattern with almost the same numbers for executions performed under the same conditions. This is numerically shown in the following two tables (Figures 5.17 and 5.18).

Theoretical printing height	Experiment	
	(100-2-5).1	(100-2-5).2
2	1,75446	1,75446
7	6,81412	6,80265
12	11,8403	11,8403
17	16,8101	16,8139
22	21,8403	21,8403
27	26,8115	26,8322

Figure 5.17: Theoretical and experimental heights (part1)

Theoretical printing height	Experiment	
	(200-1-2).1	(200-1-2).2
1	0,0834789	0,0834789
3	2,83968	2,77557
5	4,75446	4,75446
7	6,84008	6,79039
9	8,75581	8,75581
11	10,7655	10,8055

Figure 5.18: Theoretical and experimental heights (part2)

Searching on the internet, the definition of the *zonedata* argument was found starting on page 1232 of the *Technical reference manual* [29], from where the following was obtained:

finep
fine point
Data type: bool Defines whether the movement is to terminate as a stop point (fine point) or as a fly-by point.
TRUE: The movement terminates as a stop point, and the program execution will not continue until robot reaches the stop point. The remaining components in the zone data are not used.
FALSE: The movement terminates as a fly-by point, and the program execution continues about 100 ms before the robot reaches the zone.

From there, it can be concluded that even when using *fine*, it is not assured that the robot will reach the exact position, as it can terminate as a fly-by point and thus, continue 100 ms before reaching the exact zone.

No excel file has been included in the Appendix, due to the big amount of data that is stored there and because that data has already been processed and shown in the graphs, that are thought to be more visual than the excel tables (Instead, all excel files have been uploaded as a zip file). But for this case, the data from some of the excel files has been used, in order to show numerically the errors.

5.4 Results from IRB140

In this part of the report, results from IRB140 will be compared with those obtained in RobotStudio. For that, some experiments performed in RobotStudio were exactly replicated at the laboratory. In order to be able to compare the results, both executions had to be done under the same conditions, meaning that same velocity and same printing dimensions should be used on both experiments. Not all experiments performed in RobotStudio were done in IRB140, as it was thought that enough data was gathered for the comparison after one experiment with each change of variable was done.

5.4.1 Change of height of object

In this first part of the section, the results obtained and discussed on section 5.3.2 will be compared to those obtained when performing the same experiments (same velocity and dimensions, as well as random numbers) on the IRB140 at the laboratory. For that, first of all, the Table 5.7 will be done, in order to see the important results from both experiments performed on RobotStudio and on IRB140.

Table 5.7: Comparison of experiments on RobotStudio and on IRB140 with different dimensions

Experiment name	Deviation in straight line	Maximum height deviation	
		RobotStudio	IRB140
(200-1-2).1	22,75726588	0.379303	1,10933
(300-1-2).1	22,1138405	1.53729	1,1111

When analyzing the effect of the final height of the object in section 5.3.2, it was concluded that the maximum height deviation was obtained when performing the experiment with the biggest height. With the IRB140, however, it can be seen how the maximum height deviations are similar in magnitude for both final heights of the object. That is, in reality, the height of the object does not influence the height deviation in such a big way when comparing it with the influence it has on the SW RobotStudio.

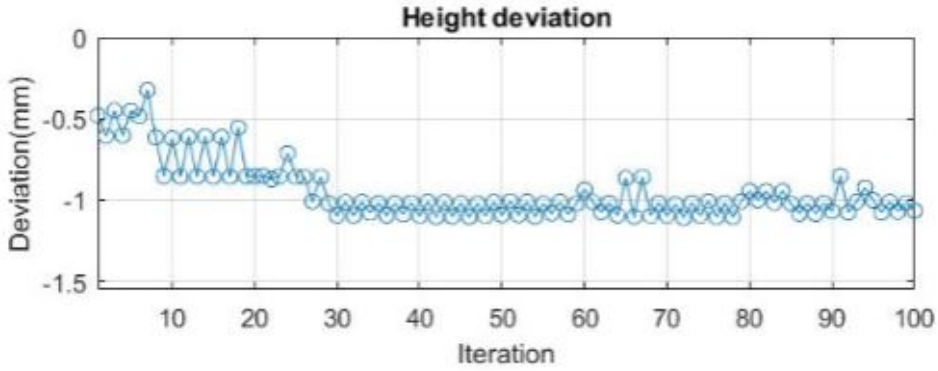


Figure 5.19: Graph of height deviation for experiment (200-1-2).1 in IRB140

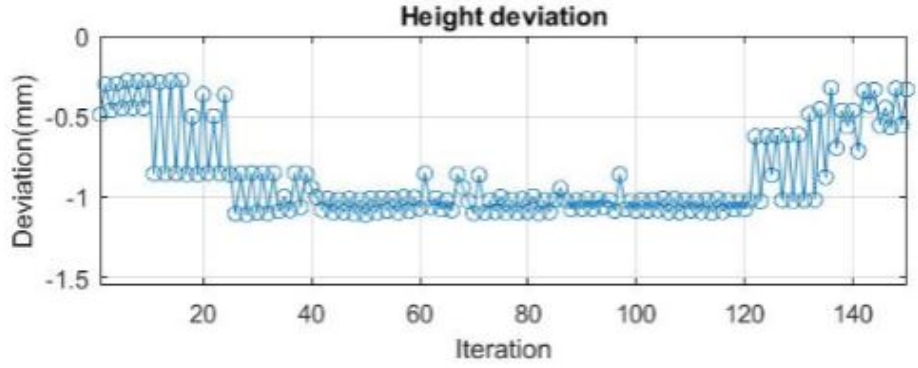


Figure 5.20: Graph of height deviation for experiment (300-1-2).1 in IRB140

By analyzing the graphs above (Figures 5.19 and 5.20) one can see how the height deviation follows the same pattern, but the experiment with a bigger height has, in the end, enough time (or iterations) in order to minimize that height deviation, whereas the one with the smaller height, finishes the print with almost the maximum value of that height deviation.

5.4.2 Change of velocity

Secondly, a comparison between the results obtained on Robotstudio and those obtained on the IRB140 at the laboratory on experiments with different velocities will be done. As

in the previous section, first of all, a talbe will be developed (Table 5.8) in order to show all relevant characteristics from both type of experiments.

Table 5.8: Comparison of experiments on RobotStudio and on IRB140 with different velocities

Experiment name	Deviation in straight line	Average height deviation	
		RobotStudio	IRB140
(v10).1	10,42105029	0,262181762	0,093032533
(v100).2	13,5807244	0,233093333	0,74145381
(v1000).2	19,98883	0,191194524	0,295603338

From the table above (Table 5.8) it can be concluded that in reality (with the IRB140 at the laboratory), the height deviation is smaller when the printing velocity is the smallest. This is what is was thought at the beginning, as mentioned in section 5.3.3, but did not happen in the experiments performed on RobotStudio. On the contrary, this has been corroborated in reality, and thus, being able to say that small printing velocities should be used for this cases, as they five the smallest height deviation apart from letting enough time for the material to expand after being deposited.

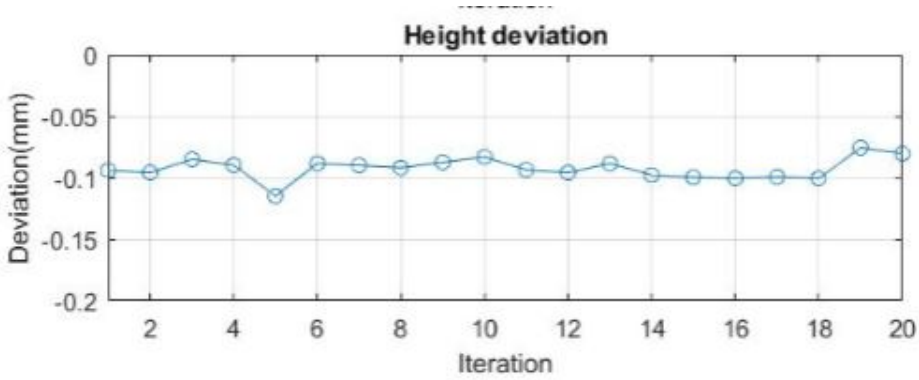


Figure 5.21: Graph of height deviation for experiment (v10).1

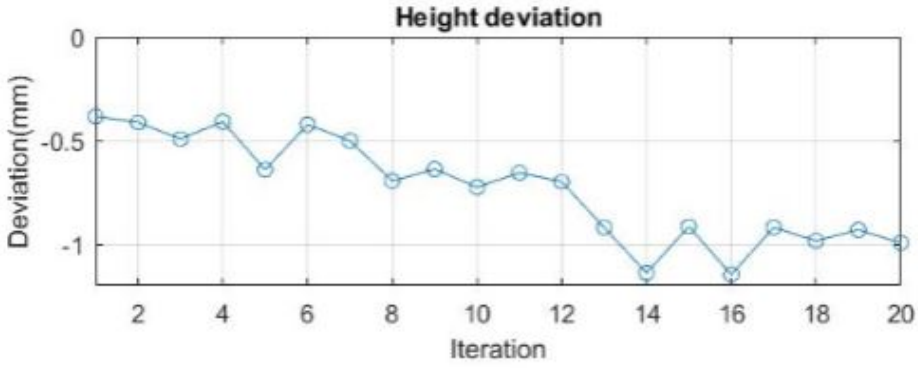


Figure 5.22: Graph of height deviation for experiment (v100).2

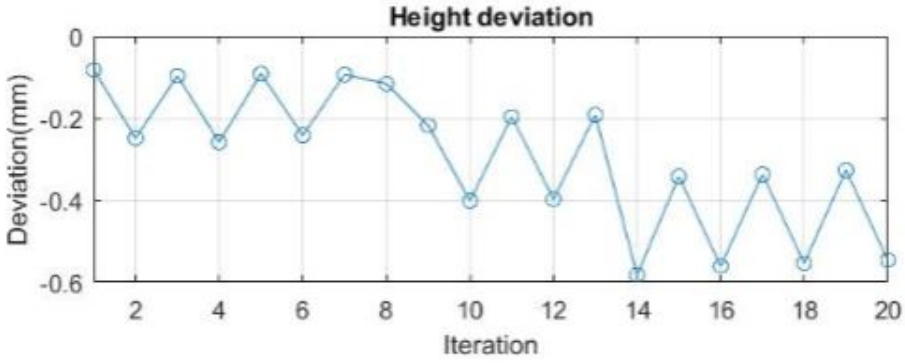


Figure 5.23: Graph of height deviation for experiment (v1000).2

Apart from that, to finish with this analysis, it is worth mentioning that the smallest printing velocity has the most constant height deviation all along the execution (As seen in graphs from Figure 5.21 - 5.23). This, as said in the case of Robotstudio, is the best case that one can face, as it can be easily corrected by inserting an offset to the z direction at the beginning of the printing. The other two cases instead, suffer from irregular height deviations (that additionally increase with iterations) making the correction of them a lot more difficult and challenging.

Chapter 6

Conclusion and further work

This chapter is used to summarize the work and results described along the report. There are two main sections:

- **Conclusion:** Summary to relate the steps followed and the results with the problem definition and goal of the project.
- **Further work:** Suggestions on steps that could be done to continue with this research area.

6.1 Conclusion

In this thesis the work has consisted of conducting several experiments on RobotStudio and at the laboratory using an IRB140 manipulator, by applying a code developed in order to correct two errors that happen quite often when 3D printing.
At this point it is worth remembering why is the use of robotic manipulators increasing in industry. One of the benefits compared to normal printers is the one linked to the reach of the robotic arm, as it increases the printing space. In conventional 3D printers, the size of the printer limits the size of the product, while with a robotic arm, the size of the product is not limited in such a way and it could be increased even further by mounting the robot in a moving platform.

For this project, a piece of code was developed where the robot was asked to follow a path that would create a straight wall when material deposition was used. This experiment was set as simple as possible, in order to focus on the correction of the errors under study, and so avoid other type of influences.

Focusing on the first error, the height deviation, different conclusions have been obtained for the SW RobotStudio and for the robot IRB140 at the laboratory.
On the experiments carried out at the laboratory, it has been seen how a slow printing velocity decreases this error noticeably while the final height of the object does not have a real influence on it.
The conclusions from experiments on RobotStudio, instead, have been quite contradictory to those obtained at the laboratory. Here, while the final height of the object had a great impact on the height deviation (being this last parameter a lot smaller when the height was smaller too), the printing velocity affected just the opposite way as in the experiments performed at the laboratory. That is, smaller height deviations where found when faster velocities where used.

Going on to the second error, the path deviation, a type of ILC control method was applied in order to correct it (as it was an iterative method that used information and data from previous iterations to update the variables). As it has been said before, no known algorithm was used, but instead, a personal one was developed.
The most relevant conclusion obtained is that the number of iterations needed to correct the path is proportional to the error in straight line, without mattering the influence of each of the x and y directions on it.

A weak part of the project is the fact that there was no time left in order to perform experiments with material deposition. This topic is left as a suggestion for further work on the next section. Even if these experiments could have been really interesting to execute and analyze, it was thought that first, the correction of the errors should be successfully completed before trying the code with material deposition. That is why, more time was spent on this experiments, and in the end, the correction of the errors was reached, as it has been mentioned during the report.

To conclude, the objectives of this thesis were accomplished. The two errors under study happened to be corrected in quite a high percentage after the application of the piece of code developed. Even with that good results, some improvements and other research areas on the same topic are suggested in the following section.

6.2 Further work

In this section, different suggestions for further work will be presented. Some of them were not developed in this thesis because there was unfortunately no time left and others were thought of later on.

6.2.1 Correction of other potential errors

The first suggestion is to complete the code so that it corrects other potential errors. This thesis report has been focused on correcting two errors, but there are others that can also be studied and corrected; such as:

- Constant extrusion of filament: This could be easily done by automatizing the extrusor.

6.2.2 Using information from cameras

In the first chapter of the report it was explained how another master student had worked on obtaining visual feedback in order to monitor when the printed path and the theoretical one did not coincide. As this project has been done in parallel with the one just mentioned, it has been impossible to use that data (the deviation in the different directions for example) as input to the error variables on the code developed here (random numbers have been used instead). Knowing that, this is the second suggestion for further work.

There are two main issues when facing this project work. First of all, how to enter the data needed into the variables in RAPID code. Data could be stored in an excel file and then read whenever needed, but this way could not be useful in this case, due to timing problems. And this is linked to the second issue, the synchronization. Data from the cameras will be taken in a continuous process (new data will be stored at each new iteration or layer), and it will have to be processed and used in that same continuous process.

6.2.3 Experiments with material deposition

Even if different experiments were carried out using the robot IRB140 from ABB at the laboratory at the Department of Engineering Cybernetics at NTNU, no material deposition was used in any of them. This is the last step to follow once the errors have been corrected; that is why, it is suggested that experiments with material deposition are carried out once the previous two suggestions are done. It is also advisable to use different materials, ranging from the simplest ones to those that suffer from contraction or expansion.

6.2.4 Extension of code to other type of paths

This research has been done only for paths following straight lines, but this is not enough for the real demands on industry, where objects are composed of curved trajectories too. Once the code works well on paths following straight lines and when executions have been carried out with material deposition, it will be time to extend that code to other type of trajectories too. This would be a big advance on this area, that is why it is also suggested to be carried out, once the previous ones have been done.

6.2.5 Implementation of control methods mentioned in section 2.2

Another topic of research could be the implementation of known control methods like ILC or RC. This have already been studied and explained in section 2.2, but in the end none of them was used in order to correct any of the errors from this research project. That being the case, it would be interesting to try to use any of them and compare results obtained there with those of this project.

Bibliography

[1] Horia Ionescu. 6 degrees of freedom, June 2010.

[2] CCEB. Where do robots come from?

[3] Jelena Z. Vidakovic, Mihailo P. Lazarevic, V. M. Kvrgic, Zorana Z. Dancuo, and Goran Z. Ferenc. Advanced quaternion forward kinematics algorithm including overview of different methods for robot kinematics. 2014.

[4] Rafael Quintanilla and J.T. Wen. Passivity based iterative learning control for mechanical systems subject to dry friction. In *Proceedings of the American Control Conference*, pages 4573–7578, July 2008.

[5] M.P. Groover. *Fundamentals of Modern Manufacturing: Materials, Processes, and Systems*. John Wiley & Sons, 2010.

[6] Saurabh Vaidya, Prashant Ambad, and Santosh Bhosle. Industry 4.0 a glimpse. *Procedia Manufacturing*, 20:233 – 238, 2018. 2nd International Conference on Materials, Manufacturing and Design Engineering (iCMMD2017), 11-12 December 2017, MIT Aurangabad, Maharashtra, INDIA.

[7] H. Ahuett-Garza and T. Kurfess. A brief discussion on the trends of habilitating technologies for industry 4.0 and smart manufacturing. *Manufacturing Letters*, 2018.

[8] Bernard Marr. What everyone must know about industry 4.0. *Forbes*, June 2016.

[9] Christian Mandrycky, Zongjie Wang, Keekyoung Kim, and Deok-Ho Kim. 3d bioprinting for engineering complex tissues. *Biotechnology Advances*, 34(4):422 – 434, 2016.

[10] Mark W. Spong, Seth. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons, Hoboken, NJ, 2006.

[11] techopedia. What does stepper motor mean?

[12] L. Danielsen Evjemo, S. Moe, J. T. Gravdahl, O. Roulet-Dubonnet, L. T. Gellein, and V. Brøtan. Additive manufacturing by robot manipulator: An overview of the state-of-the-art and proof-of-concept results. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sept 2017.

[13] Deep R. Joshi and Robello Samuel. Automated geometric path correction in directional drilling, October 2017.

[14] AJ Crispin, B Pokric, M Rankov, D Reedman, and GE Taylor. Laser based inspection and path correction system for automatic sewing apparatus. *WIT Transactions on Engineering Sciences*, 16, 1970.

[15] G. Michalos, S. Makris, A. Eytan, S. Matthaiakis, and G. Chryssolouris. Robot path correction using stereo vision system. *Procedia CIRP*, 3:352 – 357, 2012. 45th CIRP Conference on Manufacturing Systems 2012.

[16] S Selingerova, T Kubela, A Pochyly, and V Singule. On-line correction of robots path based on computer vision. *Engineering Mechanics*, page 110, 2012.

[17] Y. Veryba, L. Kourtch, B. Verigo, and V. Murashko. Method and system for robot end effector path correction using 3-d ultrasound sensors. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*, volume 2, pages 1240–1243 vol.2, 2000.

[18] The list of lists. Isaac asimov's "three laws of robotics".

[19] Samuel Bouchard. The evolution of the robotic teach pendant. *RobotiQ*, April 2011.

[20] Thomas Daun. Iterative learning control for milling with industrial robots in advanced manufacturing, 2014. Student Paper.

[21] Francis J. Doyle III Youqing Wang, Furong Gao. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, (19):1589–1600, December 2008.

[22] Mikael Norrlf. Iterative learning control – analysis, design, and experiments. Technical report, 2000.

[23] S. Arimoto, S. Kawamura, and F. Miyazaki. Bettering operation of dynamic systems by learning: A new control theory for servomechanism or mechatronics systems. In *The 23rd IEEE Conference on Decision and Control*, pages 1064–1069, Dec 1984.

[24] D.H. Owens and J. Htnen. Iterative learning control an optimization paradigm. *Annual Reviews in Control*, 29(1):57 – 70, 2005.

[25] ABB. Main page of abb.

[26] ABB. *Product specification: IRB 140*, October 2017.

[27] ABB. *Data sheet: IRB 140*, January 2010.

[28] alonsoprofe. Curso de robotstudio. tema 1.

[29] ABB Robotics. Technical reference manual: Rapid instructions, functions and data types.

Appendix A

IRB140 Data Sheet

IRB 140

Industrial Robot

Small, Powerful and Fast

Compact, powerful IRB 140 industrial robot. Six axis multipurpose robot comprising IRB 140 manipulator and S4Cplus industrial robot controller. Handles payload of 5 kg, with long reach (810 mm) of axis 5, optional floor, wall and suspended mounting. Available as Standard, Foundry and Clean Room versions, all mechanical arms completely IP67 protected, making IRB 140 easy to integrate in and suitable for a variety of applications. Uniquely extended radius of working area due to bend-back mechanism of upper arm, axis 1 rotation of 360 degrees and flexible mounting capabilities.

The compact, robust design with integrated cabling adds to overall flexibility. The Collision Detection option with full path retraction makes robot reliable and safe.

Using IRB 140T, cycle-times are considerably reduced where axis 1 and 2 predominantly are used.

Reduction between 15-20% are possible using pure axis 1 and 2 movements.

This faster version is well suited for packing applications and guided operations together with PickMaster.

IRB Foundry Plus versions are suitable for operating in extreme foundry environments. In addition to the IP67 protection, excellent surface treatment makes the robot high pressure steam washable. The white-finish Clean-Room versions meets Clean-Room 10 regulations, making it especially suited for environments with stringent cleanliness standards.

The S4Cplus controller has the electronics for controlling the robot manipulator, external axes and peripheral equipment. S4Cplus also contains system software with all basic functions for operating and programming, including two built-in Ethernet channels with 100 Mbit/s capacity. This ensures a significant increase in computing power as well as improved controller monitoring and supervision.

Main Applications

- Arc welding
- Cleaning/Spraying
- Material handling
- Deburring
- Assembly
- Machine tending
- Packing

IRB 140 Industrial Robot

Technical Data

Specification

Robot versions	Handling capacity	Reach of 5th axis	Remarks
IRB 140	5 kg	810 mm	
IRB 140F	5 kg	810 mm	Foundry protection
IRB 140CR	5 kg	810 mm	Clean Room
IRB 140T	5 kg	810 mm	

Supplementary load (on upper arm aft. wrist)	
on upper arm	1 kg
on wrist	0.5 kg
Number of axes	
Robot manipulator	6
External devices	6
Integrated signal supply	12 signals on upper arm
Integrated air supply Max.	8 bar on upper arm

Performance

Position repeatability	0.03 mm (average result from ISO test)
Axis movement	
Axis	Working range
1, C Rotation	360°
2, B Arm	200°
3, A Arm	280°
4, D Wrist	Unlimited (400° default)
5, E Bend	240°
6, P Turn	Unlimited (800° default)
Max. TCP velocity	2.5 m/s
Max. TCP acceleration	20 m/s²
Acceleration time 0-1 m/s	0.15 sec

Velocity

Axis no.	IRB 140	IRB 140T
1	200°/s	250°/s
2	200°/s	250°/s
3	260°/s	260°/s
4	360°/s	360°/s
5	360°/s	360°/s
6	450°/s	450°/s

Cycle Time

	IRB 140	IRB 140T
5 kg Picking side cycle 25 x 300 x 25 mm	0,85s	0,77s

Electrical Connections

Supply voltage	200–600 V, 50/60 Hz
Rated power	
Transformer rating	4.5 kVA

Physical

Robot mounting	Any angle
Dimensions	
Robot base	400 x 450 mm
Robot controller H x W x D	950 x 800 x 620 mm
Weight	
Robot manipulator	98 kg
Robot controller	250 kg

Environment

Ambient temperature	
Robot manipulator	5 – 45°C
Robot controller	5 – 52°C
Relative humidity	Max. 95%
Degree of protection, Manipulator	IP67
Foundry/Clean Room	High pressure steam washable
Clean Room	Class 10
Controller	Air cooled
Computer system	Totally enclosed
Noise level	Max. 70 dB (A)
Safety	Double circuits with supervision, emergency stops and safety functions, 3-position enable device
Emission	EMC/EMI-shielded

Data and dimensions may be changed without notice.

Working range and load diagram

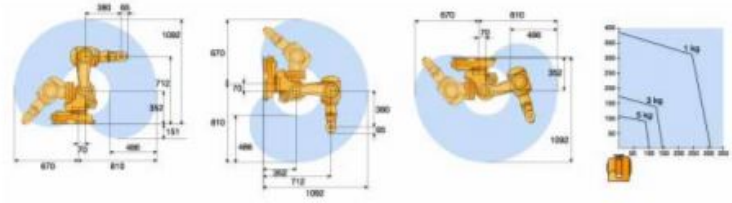


ABB Limited
Plot Nos. 5 & 6, II Phase, Peenya Industrial Area
Bangalore 560 058
Tel: +91-80-2294 9507 – 08
Fax: +91-80-2839 6537

www.abb.co.in

Appendix B

Matlab code

This Appendix contains the pieces of code for obtaining the following:

- Forward kinematics of the IRB140
- Graphs with object in 3D and its views
- Graphs with path and height deviations

Forward kinematics of the IRB140

```
%How many joints does the manipulator have?
disp('The number of joints of this manipulator is: ');
n = 6
%Which is the type of the manipulator? (1=revolute; 0=prismatic)
joint_type = [1, 1, 1, 1, 1, 1];
%Variables: (all revolute so theta(i))
q = sym('q',[n 1],'real');

%DH parameters: (order: a, alpha, d, theta)
dh_parameters = [ 70 , -pi/2 , 352 , q(1);
                 360 ,     0 ,   0 , q(2);
                   0 , -pi/2 ,   0 , q(3);
                   0 ,  pi/2 , 380 , q(4);
                   0 , -pi/2 ,   0 , q(5);
                   0 ,     0 ,  65 , q(6)];
%create the Ai matrices:
T0 = eye(4);
T = cell(1,n);
for i = 1:6
    a = dh_parameters (i,1);
    alpha = dh_parameters (i,2);
    d = dh_parameters (i,3);
    theta = dh_parameters (i,4);
    A = [cos(theta) , -sin(theta)*cos(alpha) ,  sin(theta)*sin(alpha) , a*cos(theta);
         sin(theta) ,  cos(theta)*cos(alpha) , -cos(theta)*sin(alpha) , a*sin(theta);
              0     ,       sin(alpha)       ,       cos(alpha)      ,       d     ;
              0     ,           0            ,           0           ,       1     ]
    T{i} = T0 * A;
    T0 = T{i};
end
```

Graphs with object in 3D and its views

```
x_array = xlsread('changevelocity.xlsx', '(v10)','C10:C29');
y_array = xlsread('changevelocity.xlsx', '(v10)','E10:E29');
z_array = xlsread('changevelocity.xlsx', '(v10)','G10:G29');
size_array = size(x_array);
x = zeros([(size_array*2) 1]);
x_theory = zeros([(size_array*2) 1]);
y = zeros([(size_array*2) 1]);
y_theory = zeros([(size_array*2) 1]);
z = zeros([(size_array*2) 1]);
x(1) = 0;
y(1) = 0;
z(1) = z_array (1);

for i = 1:size_array
    if mod(i,2) == 1
        x(2*i-1) = 0;
        x(2*i) = x_array(i);
        x_theory(2*i-1) = 0;
        x_theory(2*i) = 100;
        y(2*i-1) = 0;
        y(2*i) = y_array(i);
        y_theory(2*i-1) = 0;
        y_theory(2*i) = 100;
        z(2*i-1) = z_array(i);
        z(2*i) = z_array(i);
    else
        x(2*i-1) = x_array(i);
        x(2*i) = 0;
        x_theory(2*i-1) = 100;
        x_theory(2*i) = 0;
        y(2*i-1) = y_array(i);
        y(2*i) = 0;
        y_theory(2*i-1) = 100;
        y_theory(2*i) = 0;
        z(2*i-1) = z_array(i);
        z(2*i) = z_array(i);
    end
end

subplot(2,2,1)
plot(y,z)
hold on
plot(y_theory,z, 'ro')
hold off
xlabel('Width(y)')
```

```
46 ylabel('Height(z)')
47 grid on
48 xlim([0 105])
49 ylim([0 100])
50 set(gca,'Xdir','reverse')
51 subplot(2,2,2)
52 plot(x,z)
53 hold on
54 plot(x_theory,z, 'ro')
55 hold off
56 xlabel('Length(x)')
57 ylabel('Height(z)')
58 grid on
59 xlim([0 105])
60 ylim([0 100])
61 subplot(2,2,3)
62 plot3(x,y,z,'-o')
63 hold on
64 plot3(y_theory,x_theory,z, '--r')
65 hold off
66 xlabel('Width(y)')
67 ylabel('Length(x)')
68 zlabel('Height(z)')
69 grid on
70 set(gca,'Xdir','reverse','Ydir','reverse')
71 xlim([0 105])
72 ylim([0 105])
73 zlim([0 100])
74 subplot(2,2,4)
75 plot(x,y,'-o')
76 hold on
77 plot(x_theory,y_theory, '--r')
78 hold off
79 xlabel('Length(x)')
80 ylabel('Width(y)')
81 grid on
82 axis ij
83 xlim([0 105])
84 ylim([0 105])
```

Graphs with path and height deviations

```
pathdev = xlsread('changevelocity.xlsx', '(v1000)','B103:B122');
iteration = xlsread('changevelocity.xlsx', '(v1000)','A103:A122');
heightdev = xlsread('changevelocity.xlsx', '(v1000)','H103:H122');

subplot(2,1,1)
plot(iteration,pathdev, '-o')
grid on
xlabel('Iteration')
ylabel('Deviation(mm)')
xlim([1 20])
title('Path deviation');

subplot(2,1,2)
plot(iteration,heightdev, '-o')
grid on
xlabel('Iteration')
ylabel('Deviation(mm)')
xlim([1 20])
ylim([-0.3 0])
title('Height deviation');
```

Appendix C

Graphics

(Graphs have been ordered following the tables in Section 5.2, and each of them named the following way:

General graphs:*(Group name).(Number of experiment)*

Graphs on deviations: *(Group name).(Number of experiment + dev)*

Change of dimensions

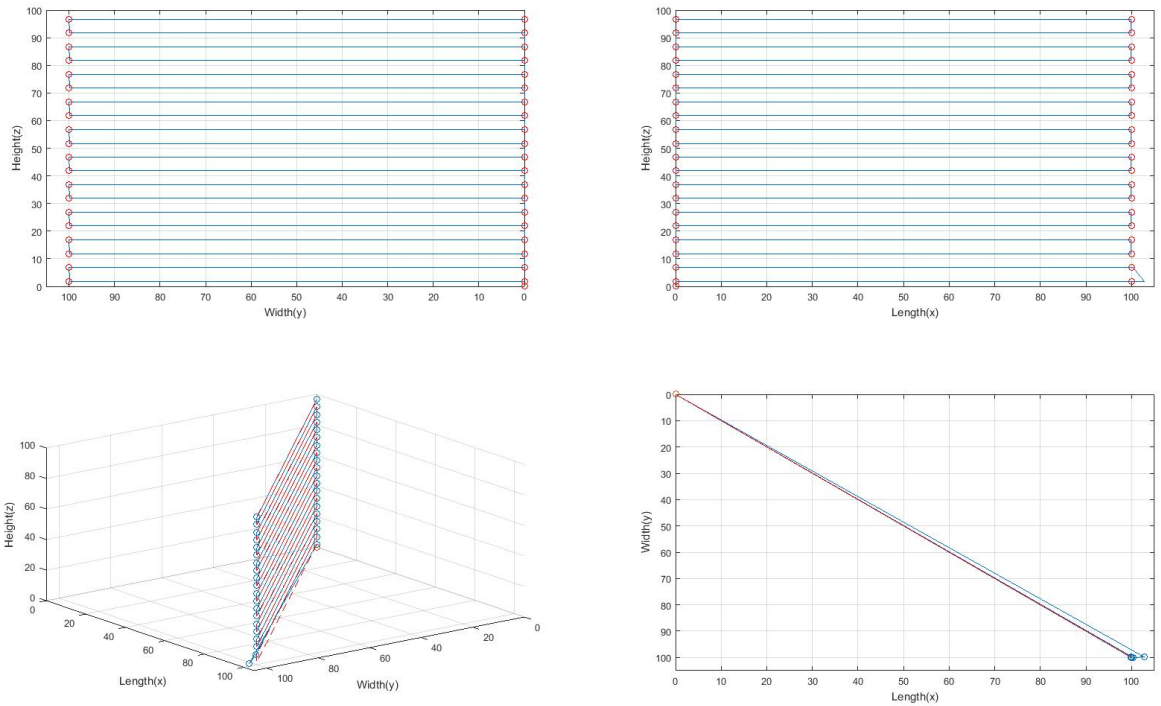


Figure C.1: (100-2-5).1

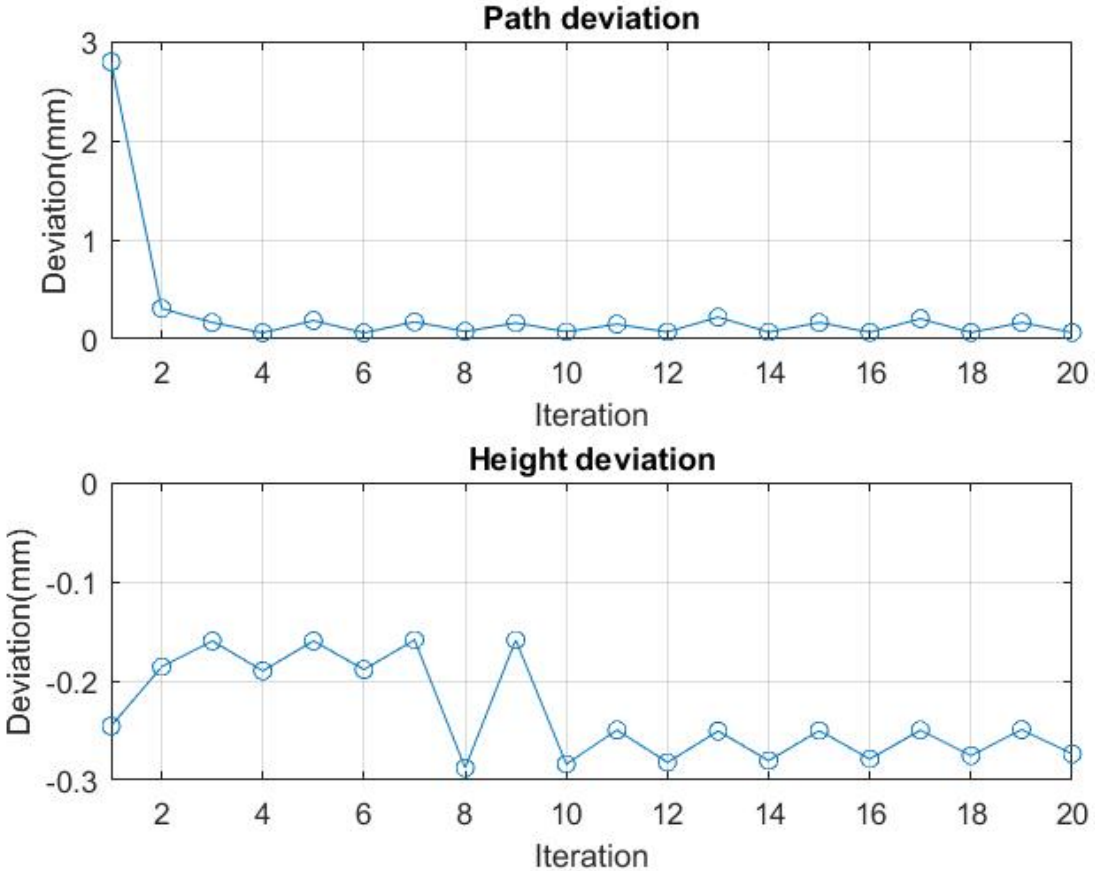


Figure C.2: (100-2-5).1dev

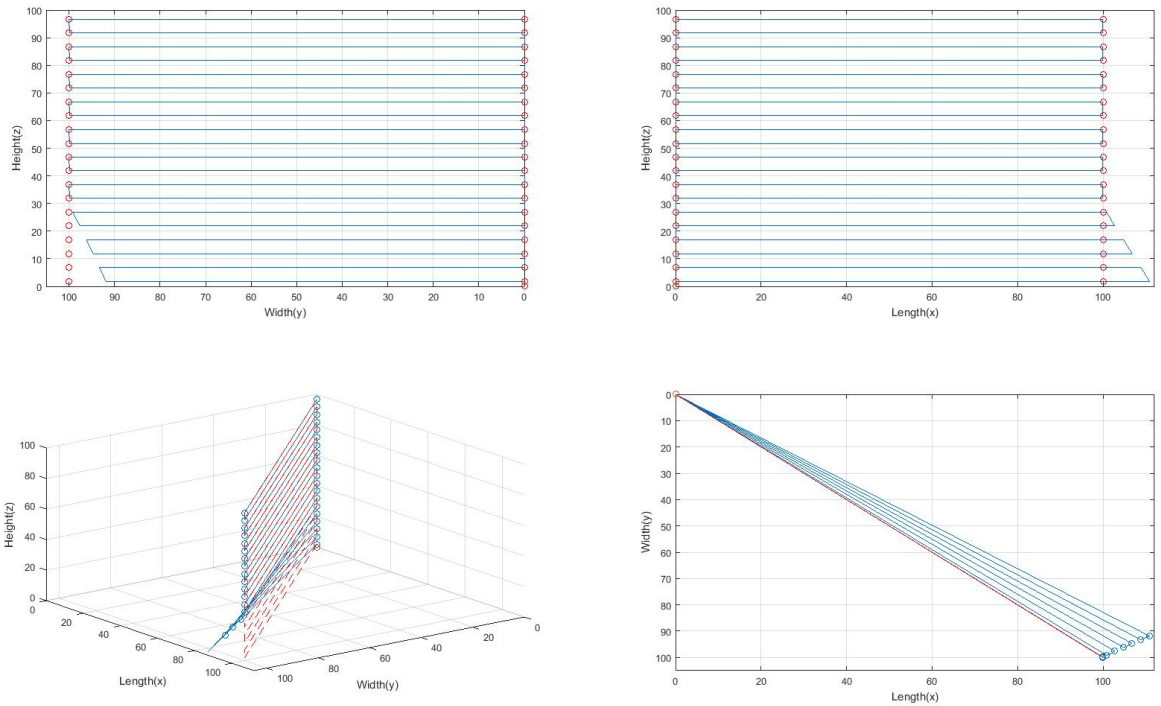


Figure C.3: (100-2-5).2

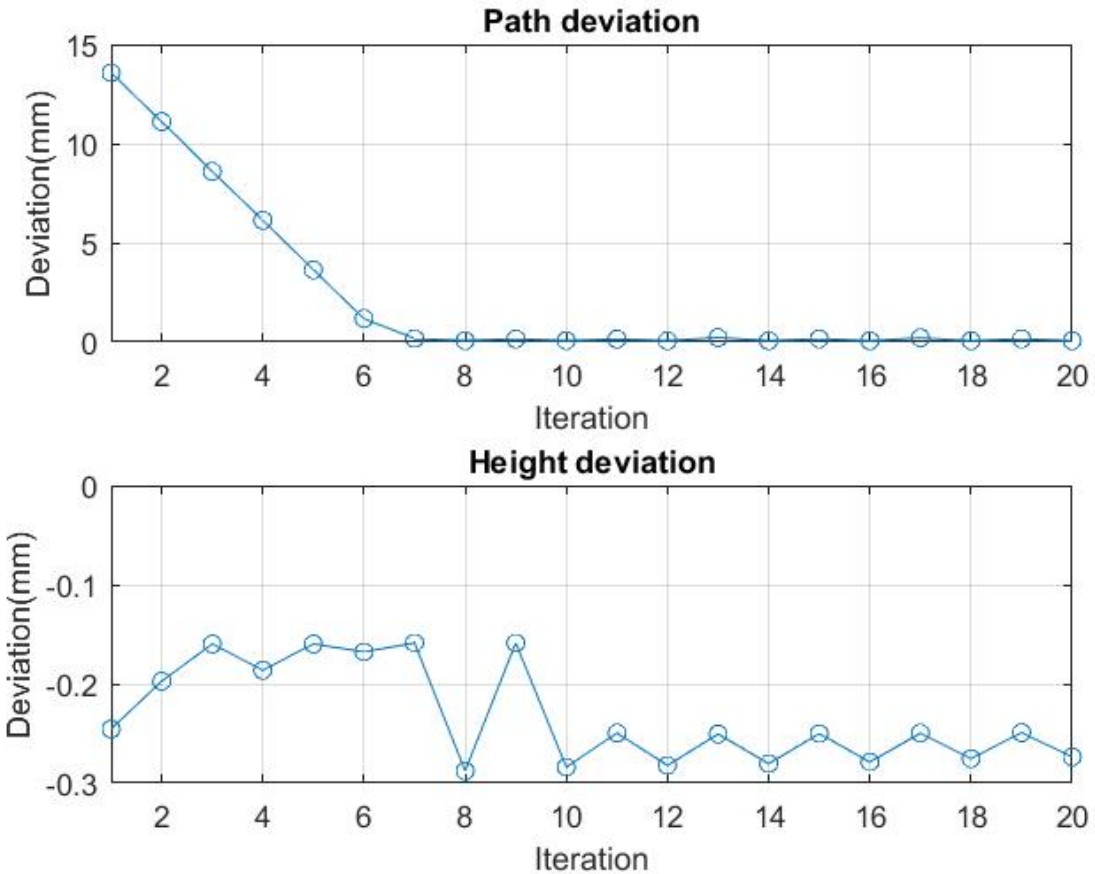


Figure C.4: (100-2-5).2dev

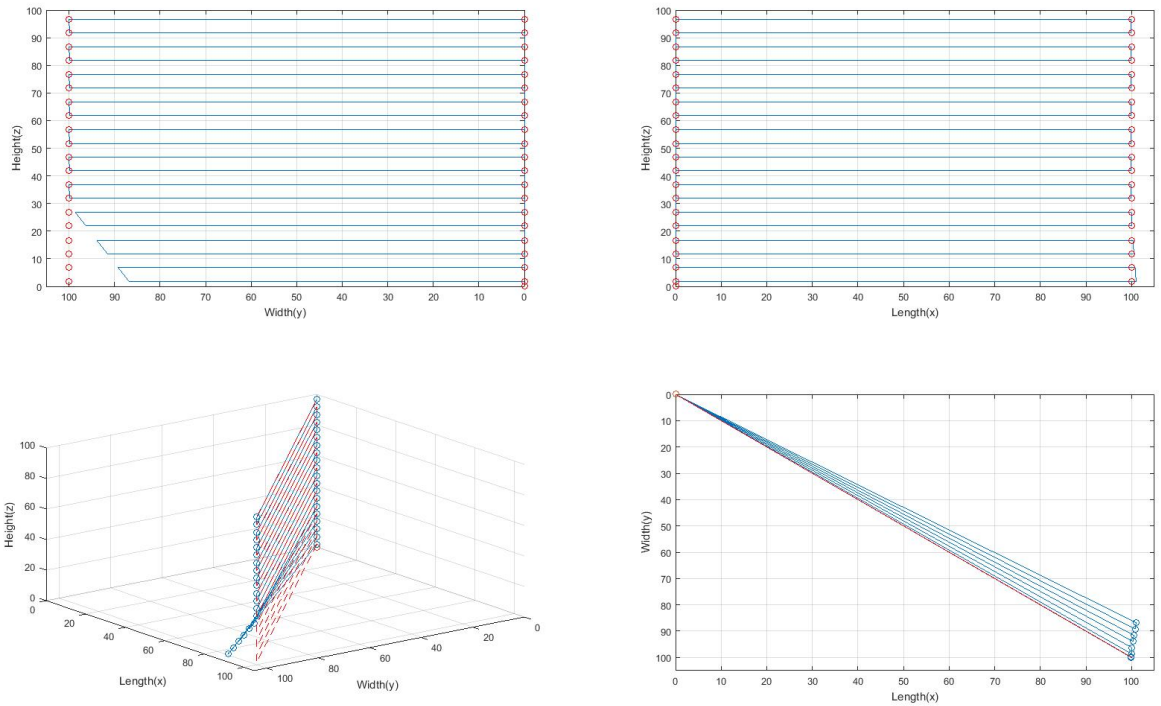


Figure C.5: (100-2-5).3

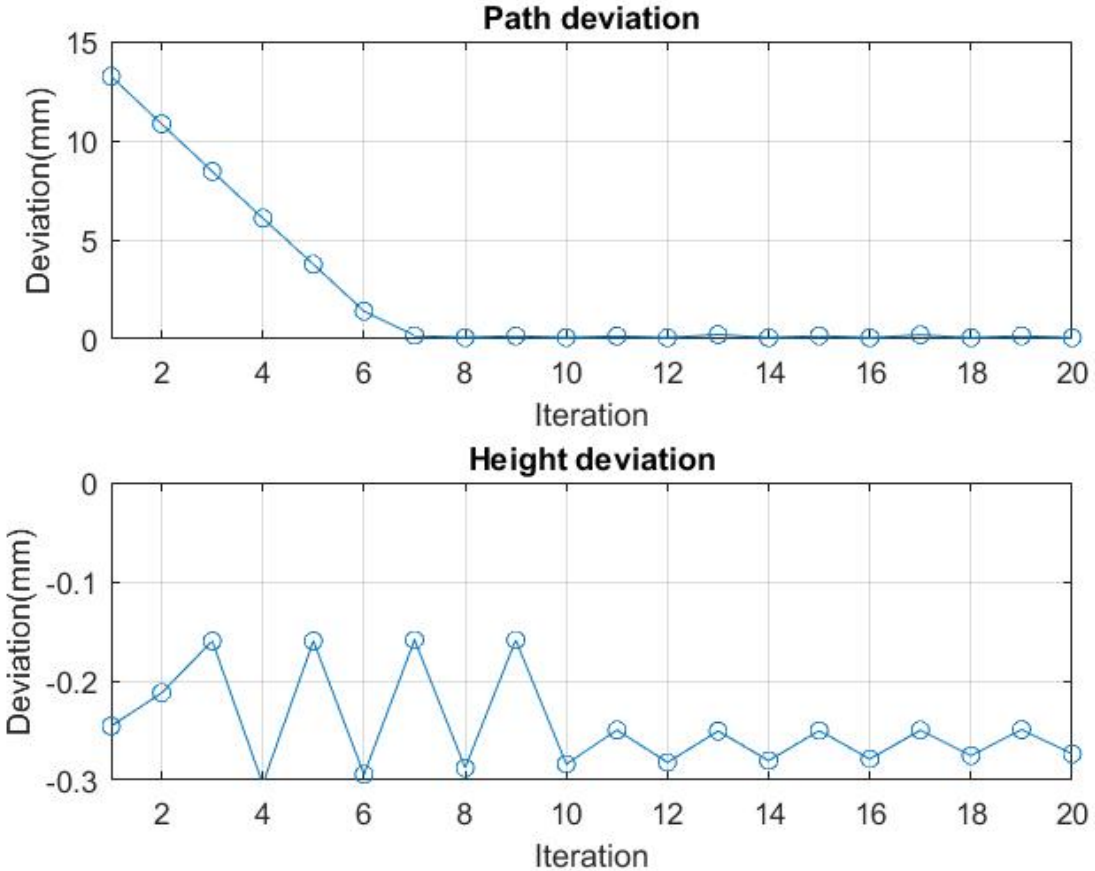


Figure C.6: (100-2-5).3dev

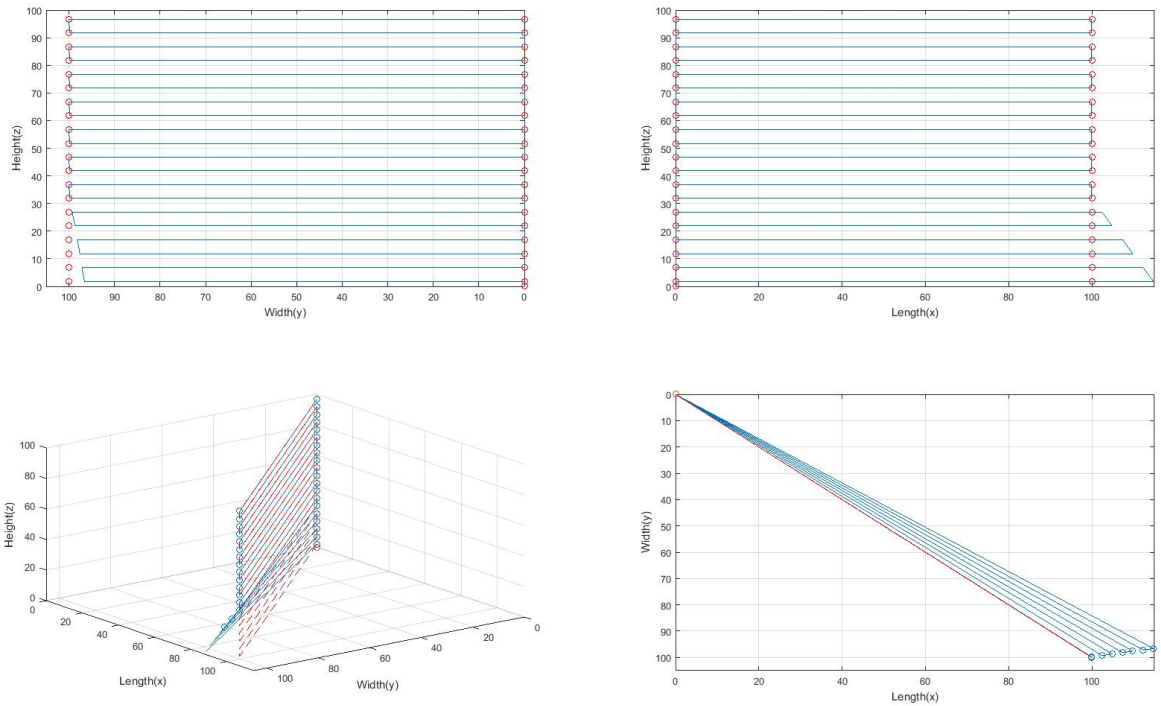


Figure C.7: (100-2-5).4

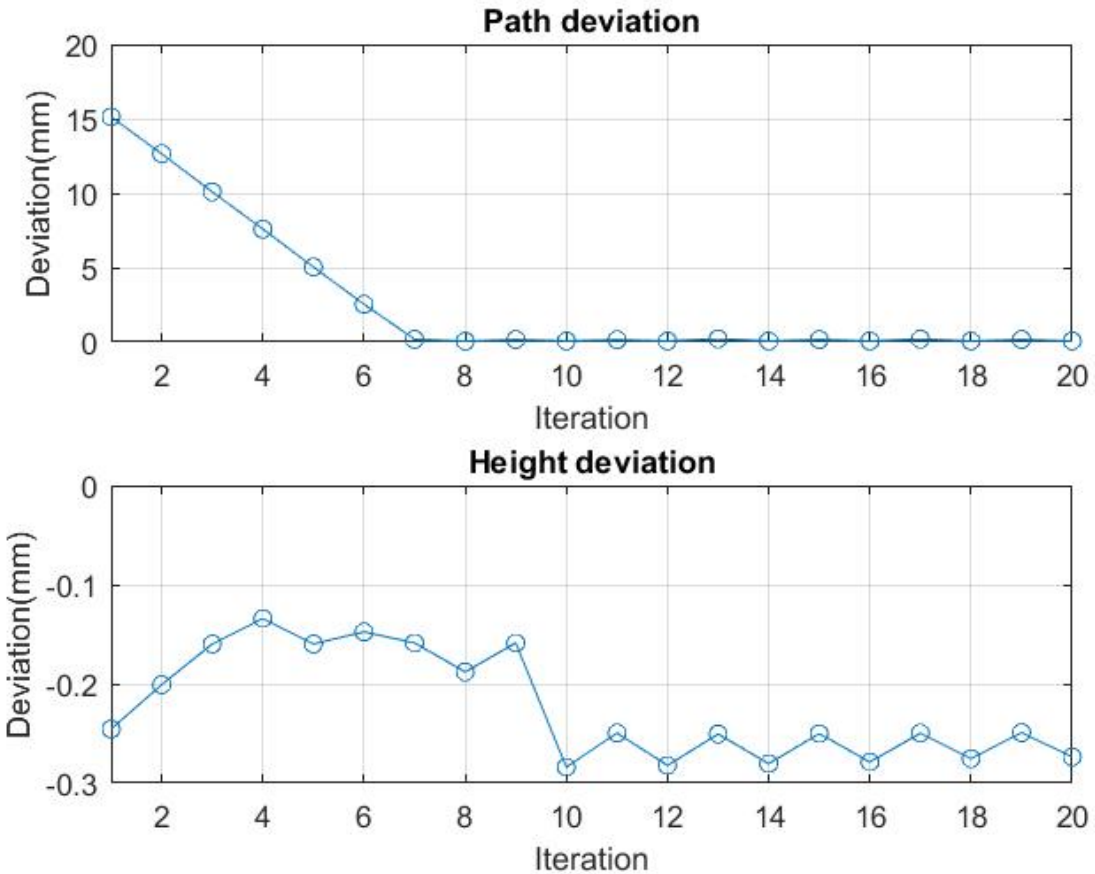


Figure C.8: (100-2-5).4dev

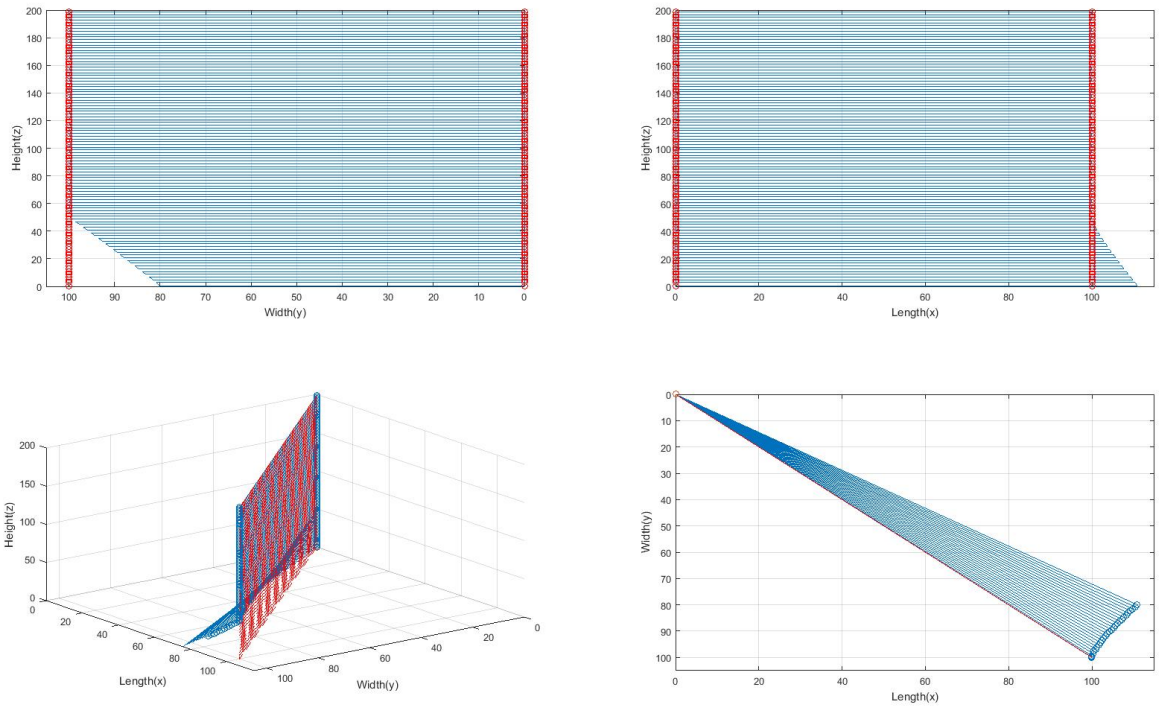


Figure C.9: (200-1-2).1

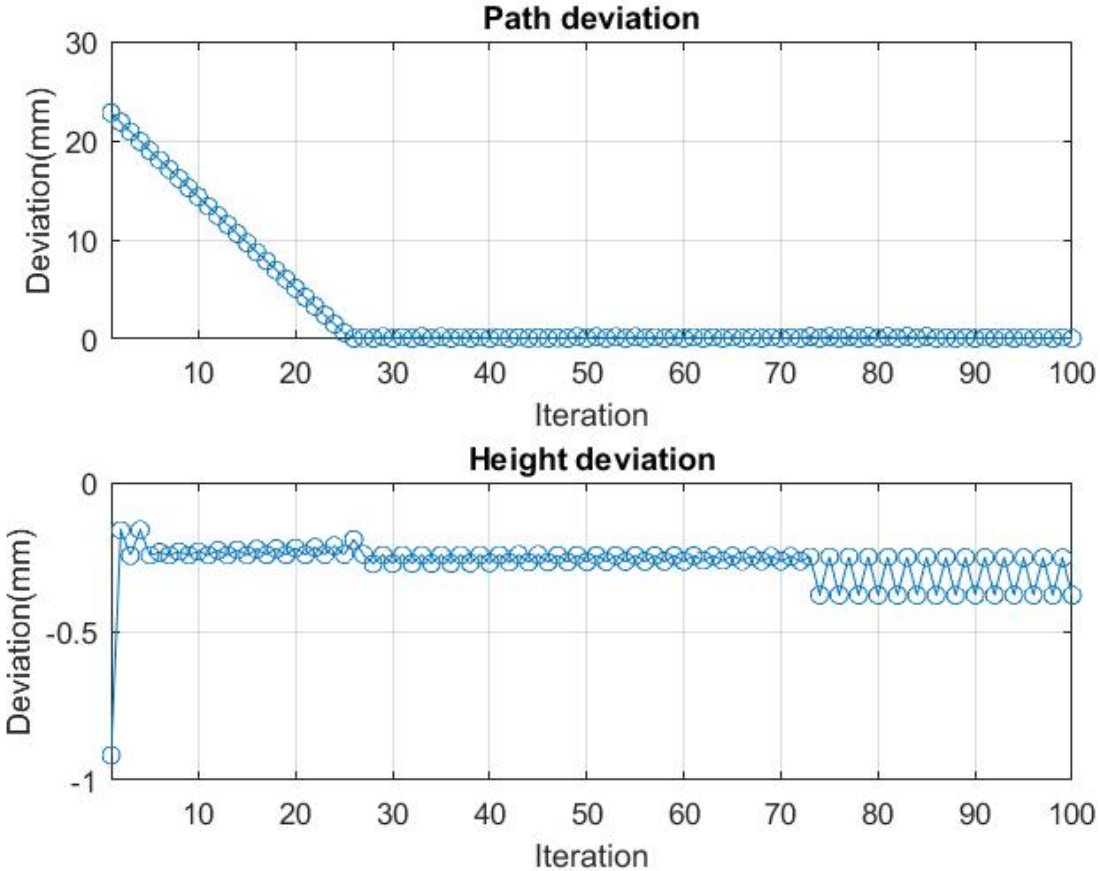


Figure C.10: (200-1-2).1dev

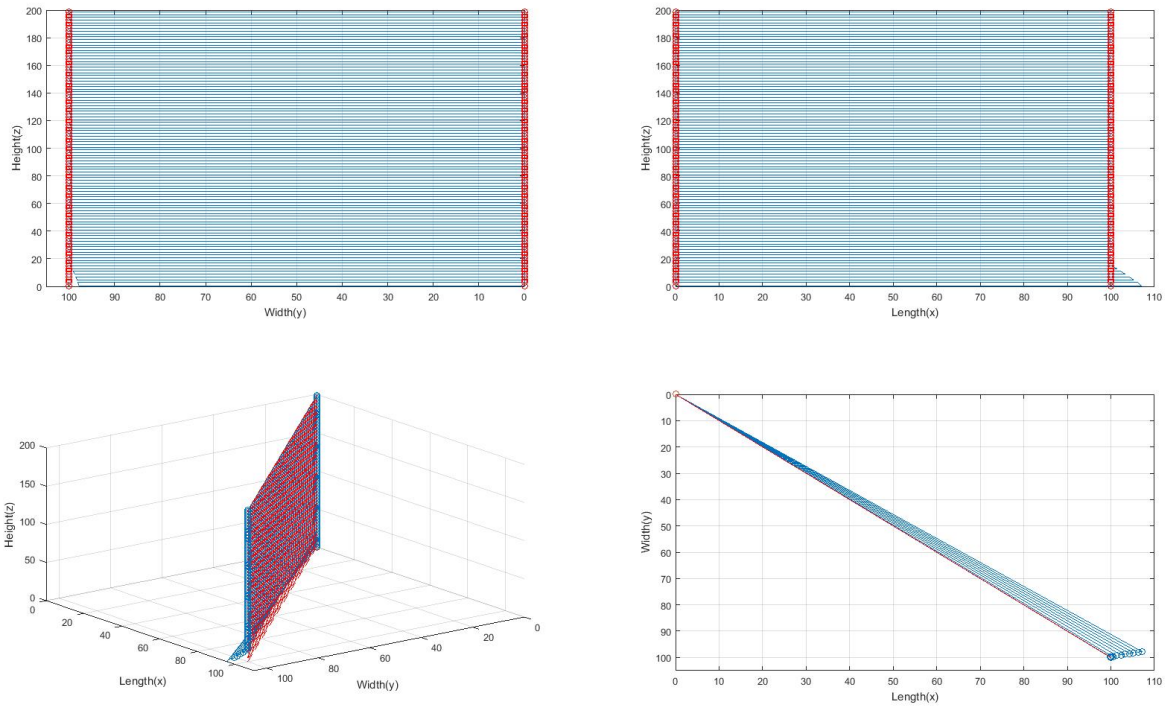


Figure C.11: (200-1-2).2

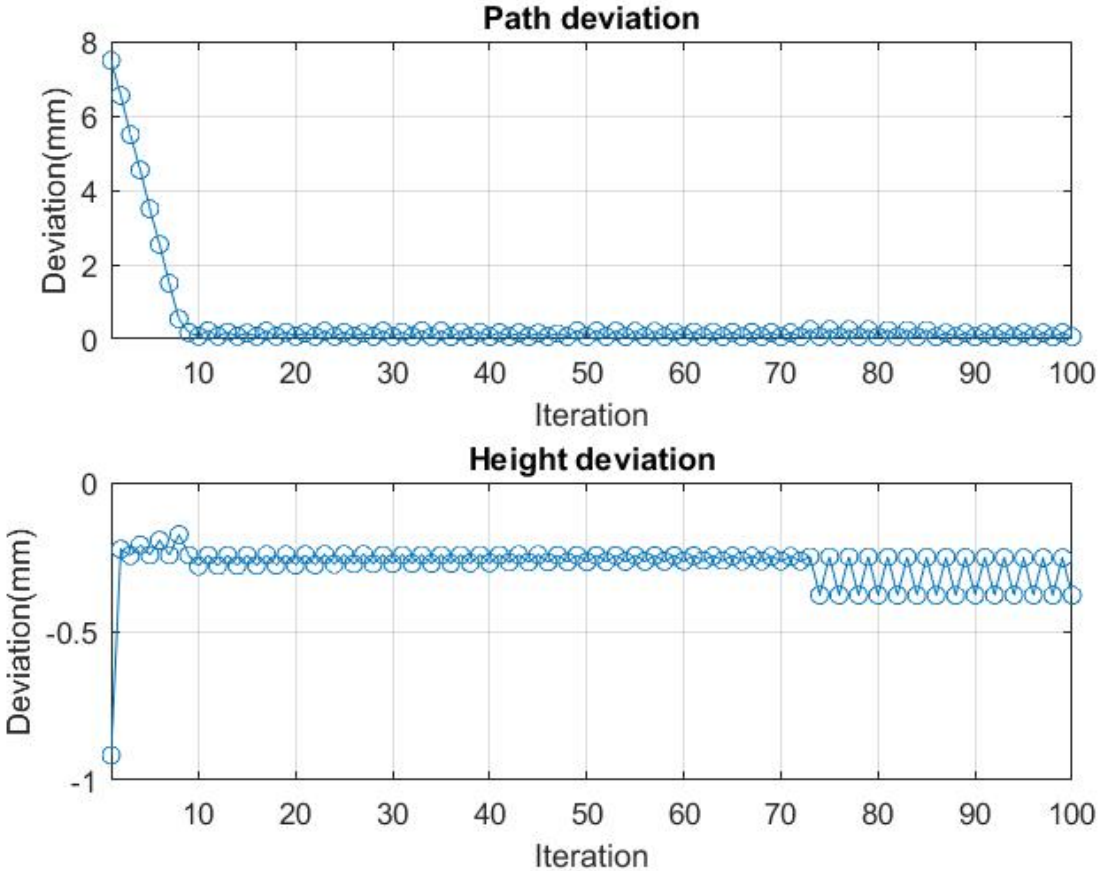


Figure C.12: (200-1-2).2dev

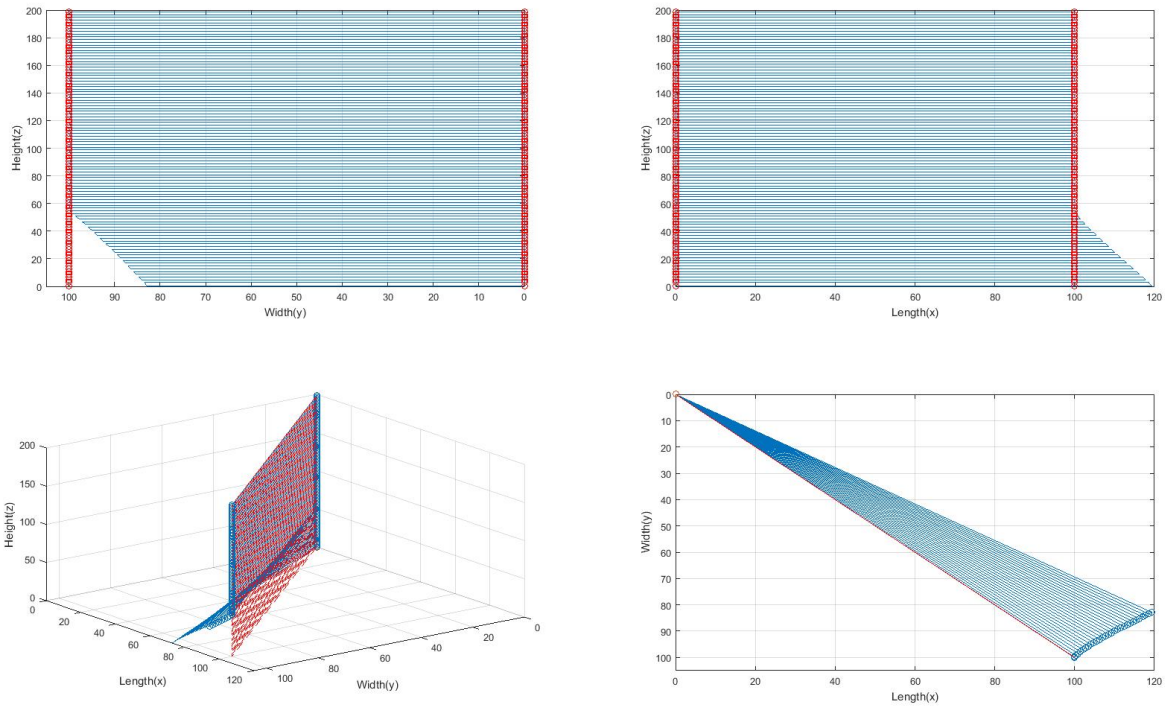


Figure C.13: (200-1-2).3

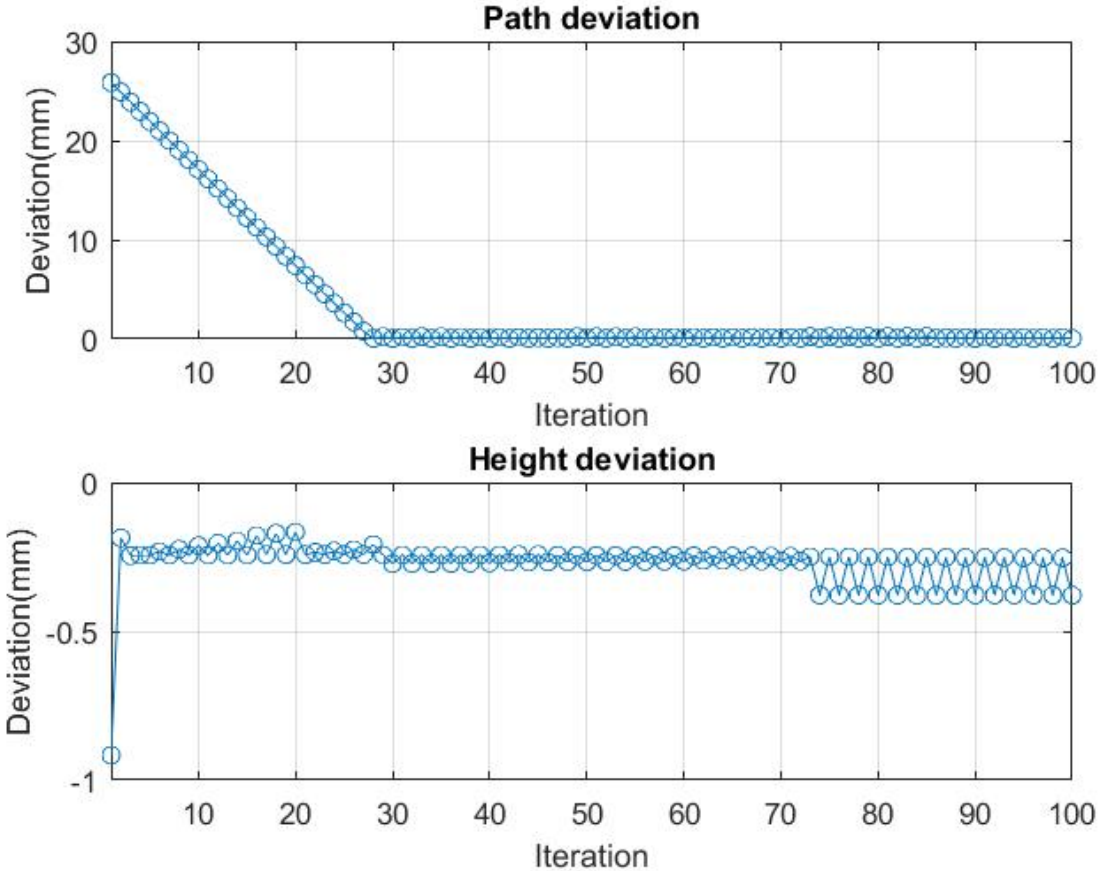


Figure C.14: (200-1-2).3dev

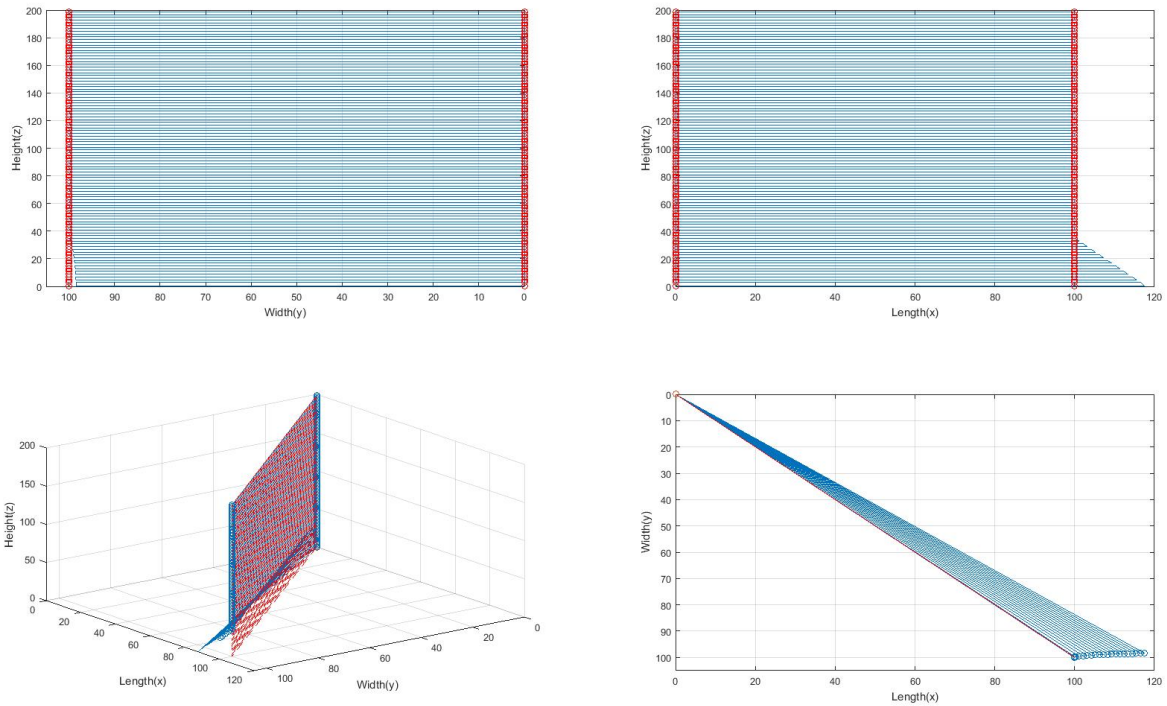


Figure C.15: (200-1-2).4

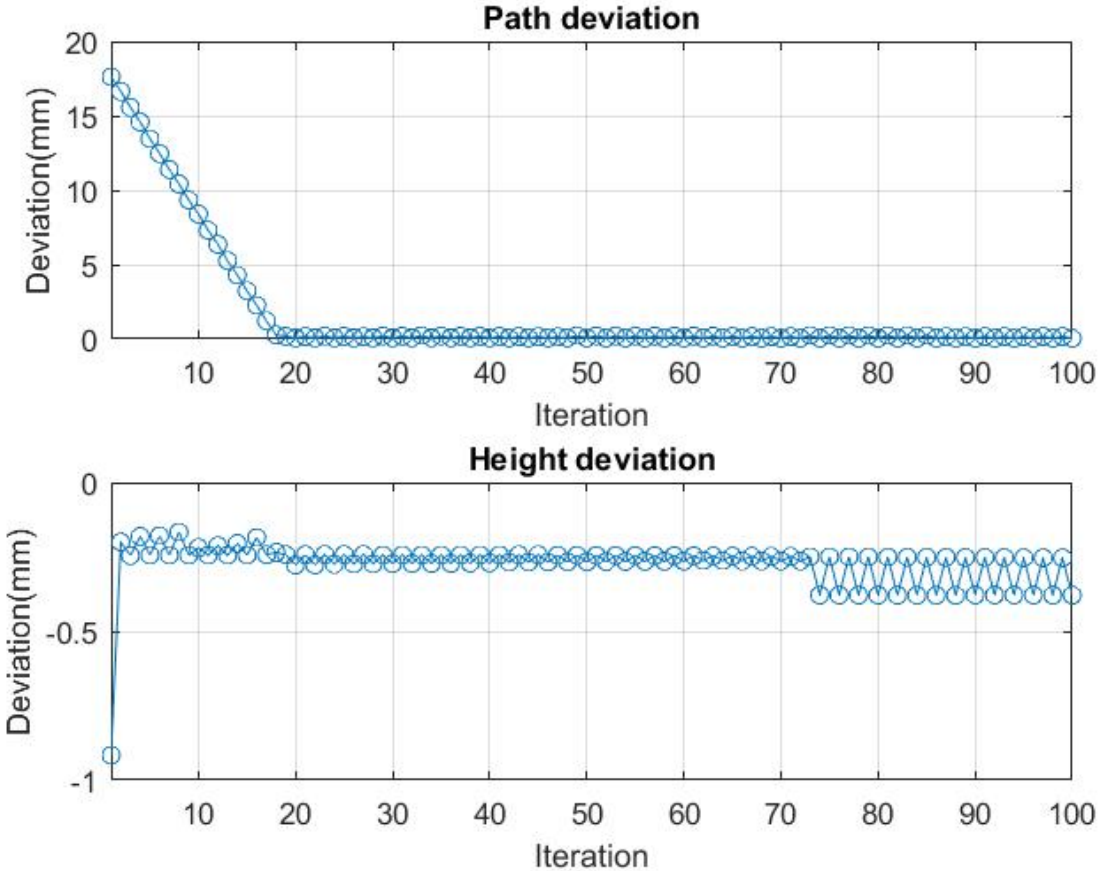


Figure C.16: (200-1-2).4dev

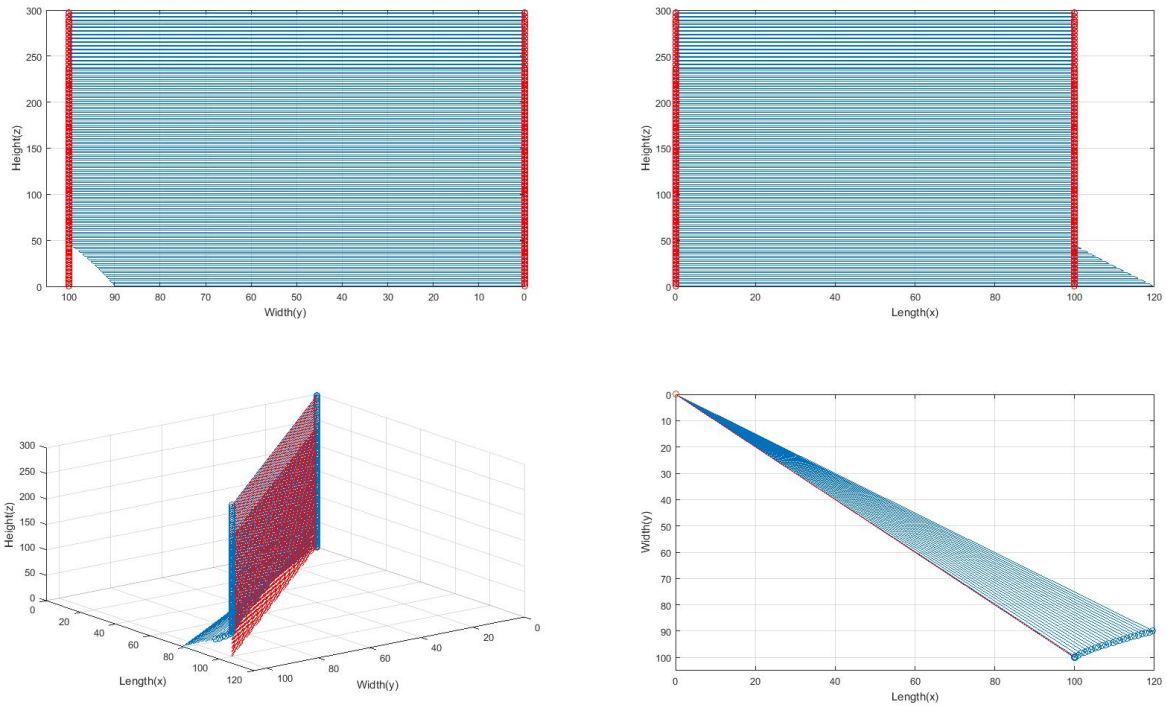


Figure C.17: (300-1-2).1

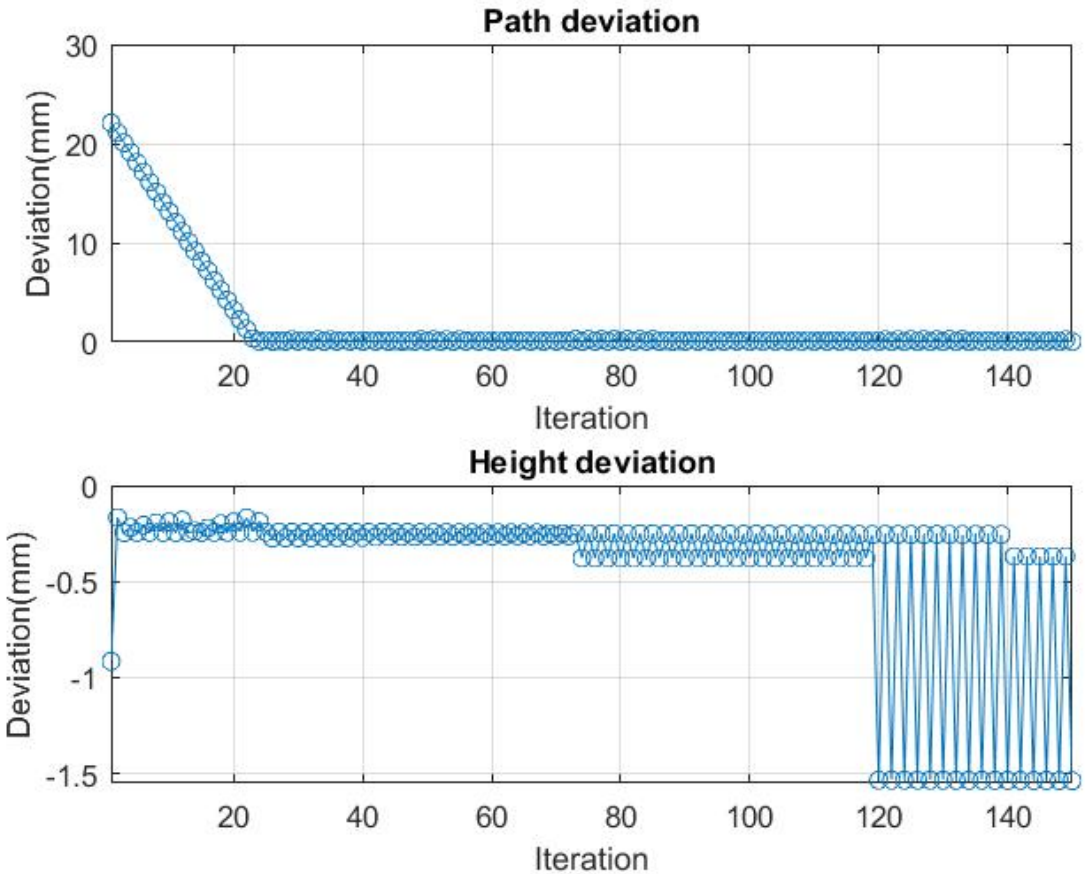


Figure C.18: (300-1-2).1dev

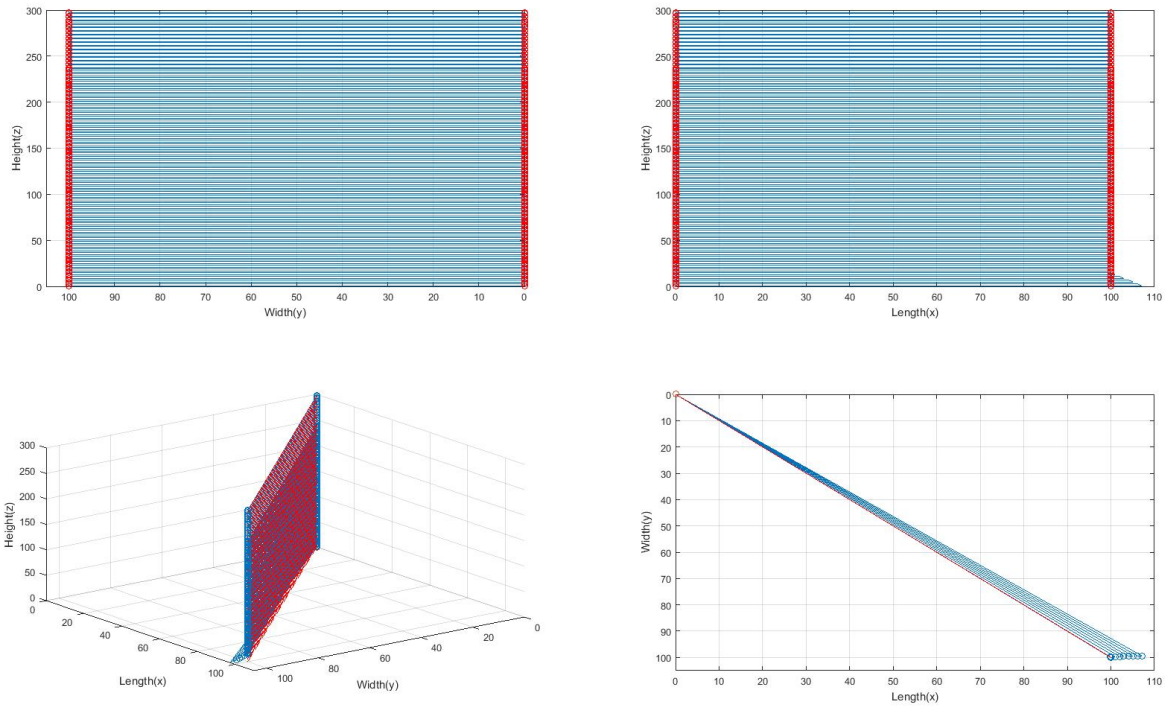


Figure C.19: (300-1-2).2

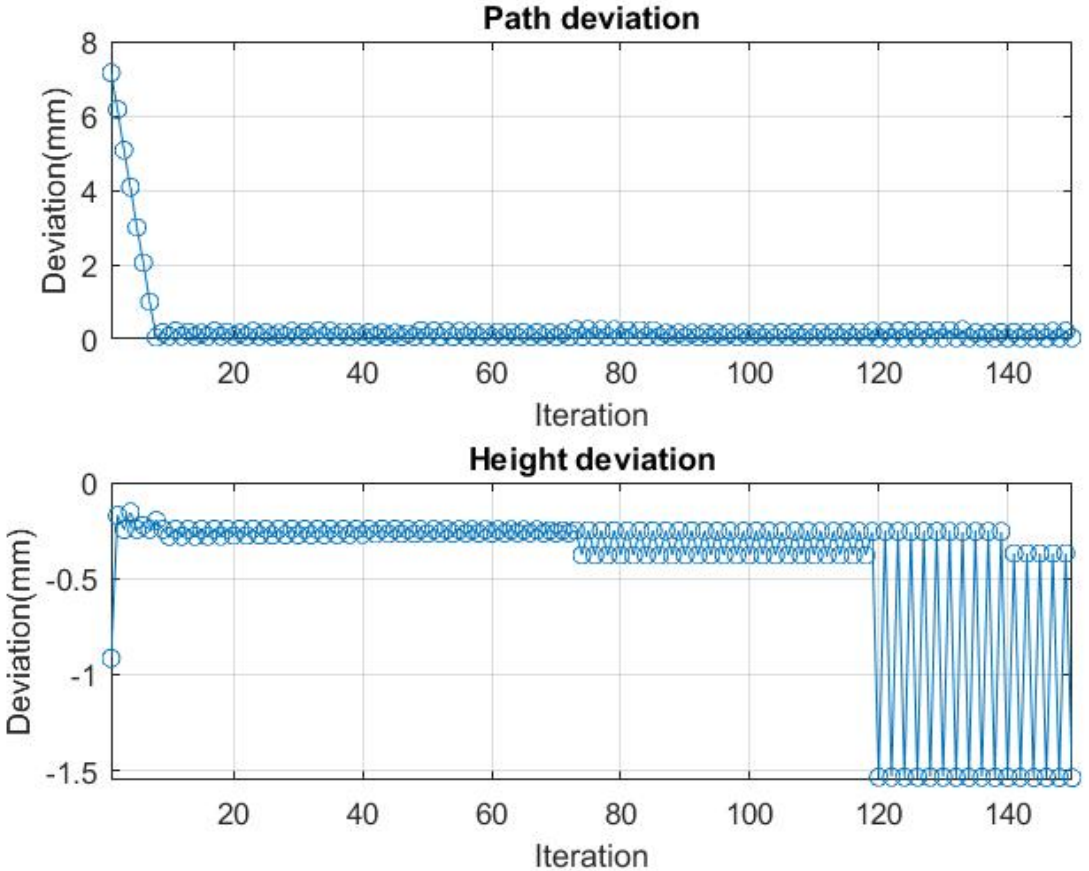


Figure C.20: (300-1-2).2dev

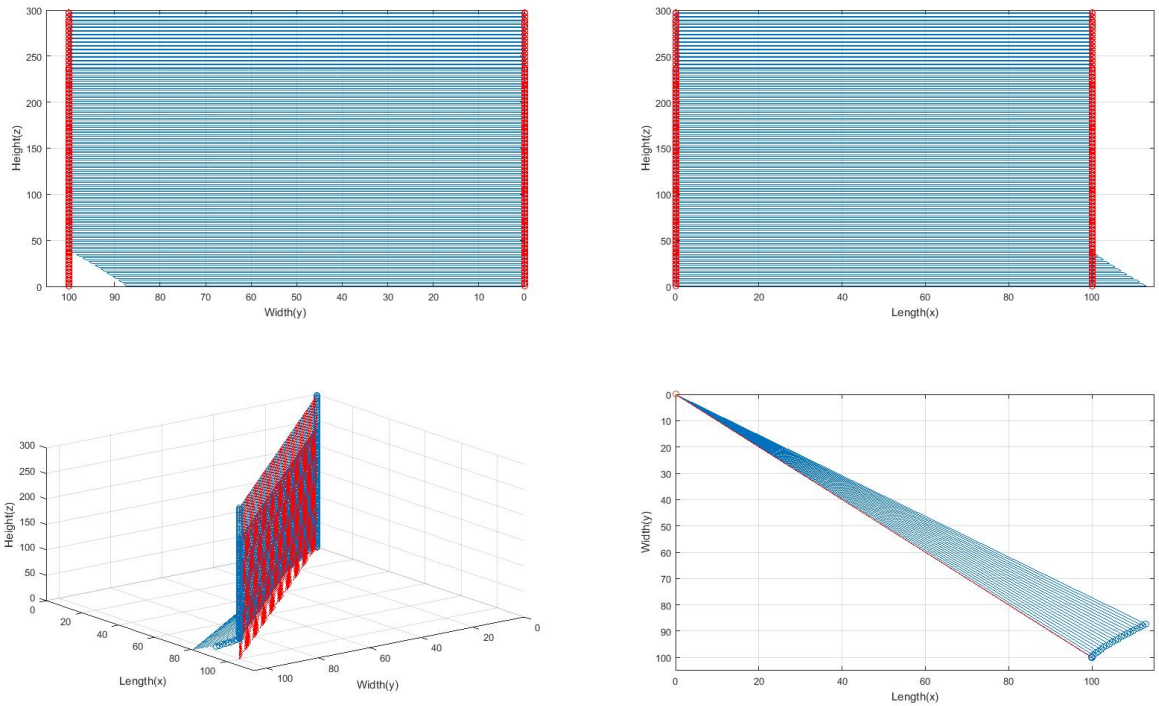


Figure C.21: (300-1-2).3

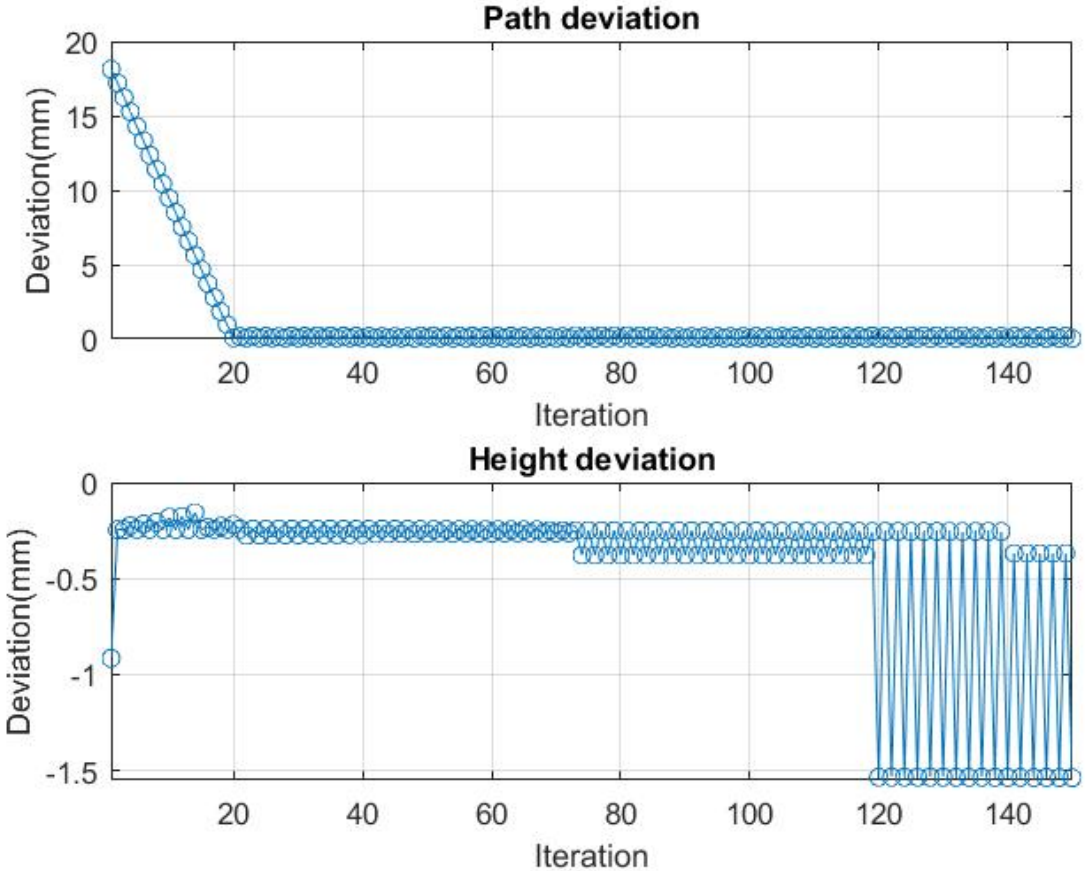


Figure C.22: (300-1-2).3dev

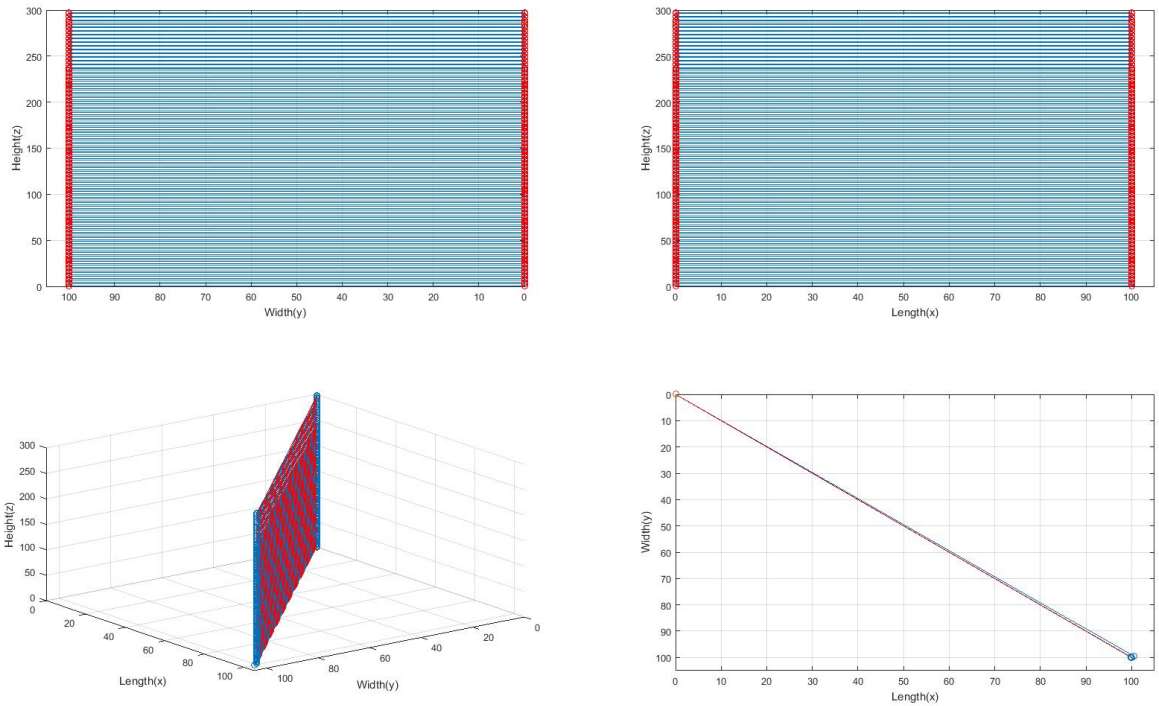


Figure C.23: (300-1-2).4

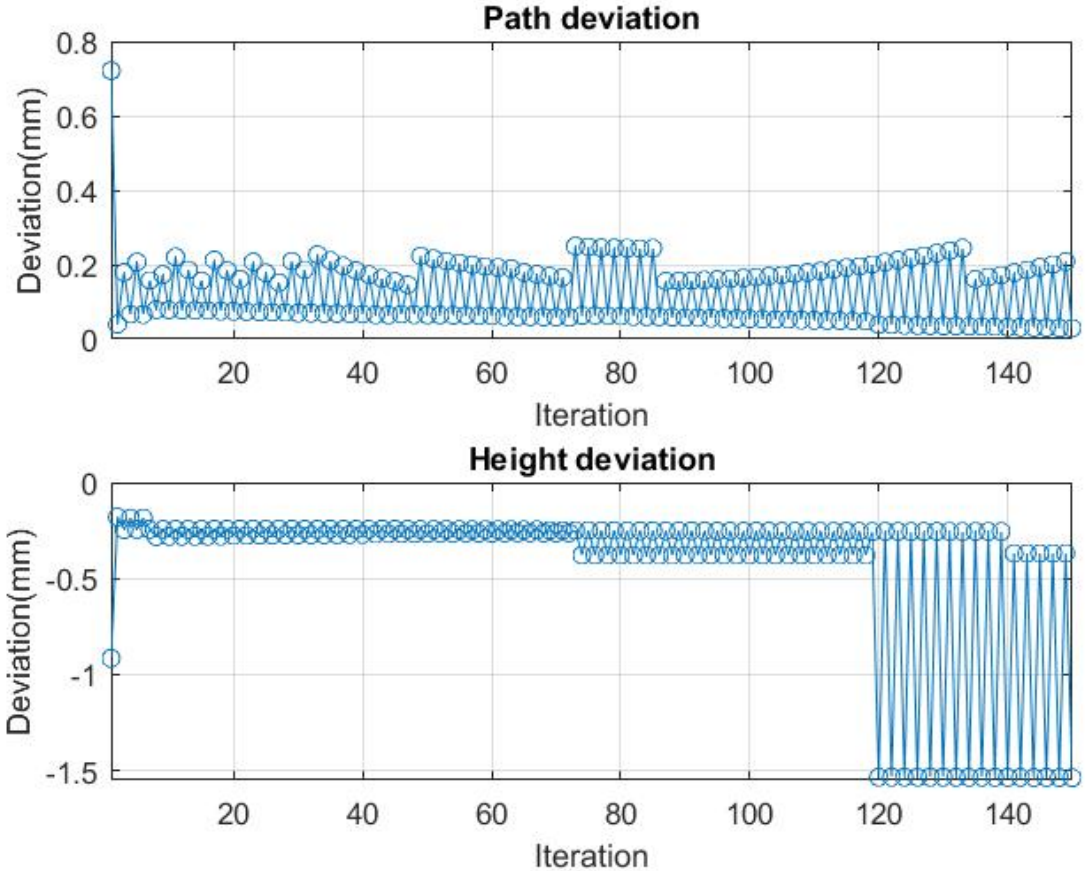


Figure C.24: (300-1-2).4dev

Change of velocity

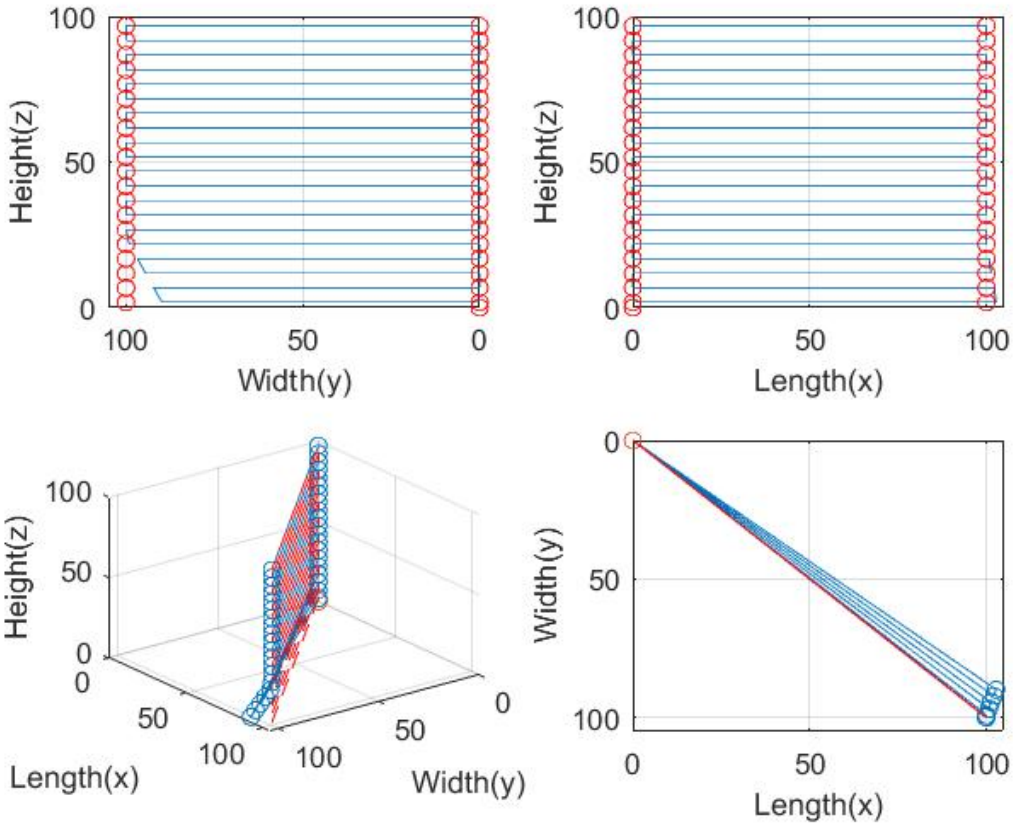


Figure C.25: (v10).1

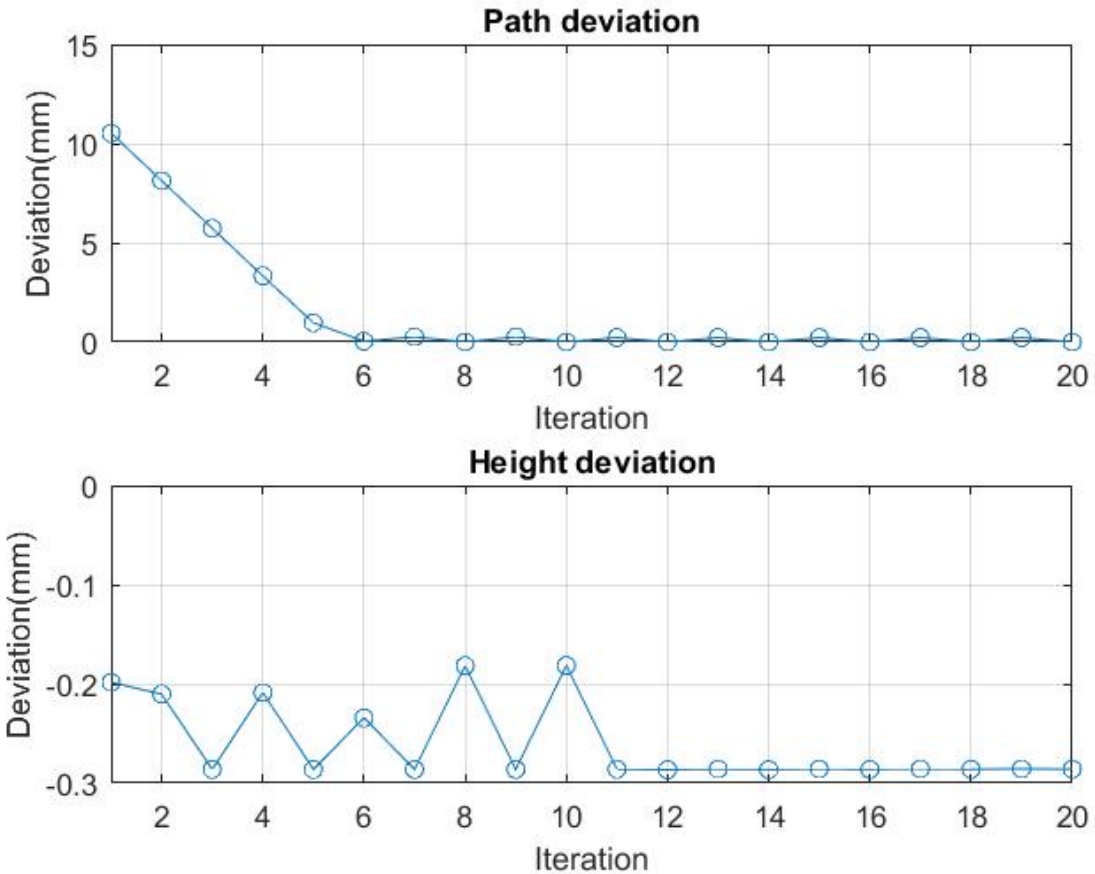


Figure C.26: (v10).1dev

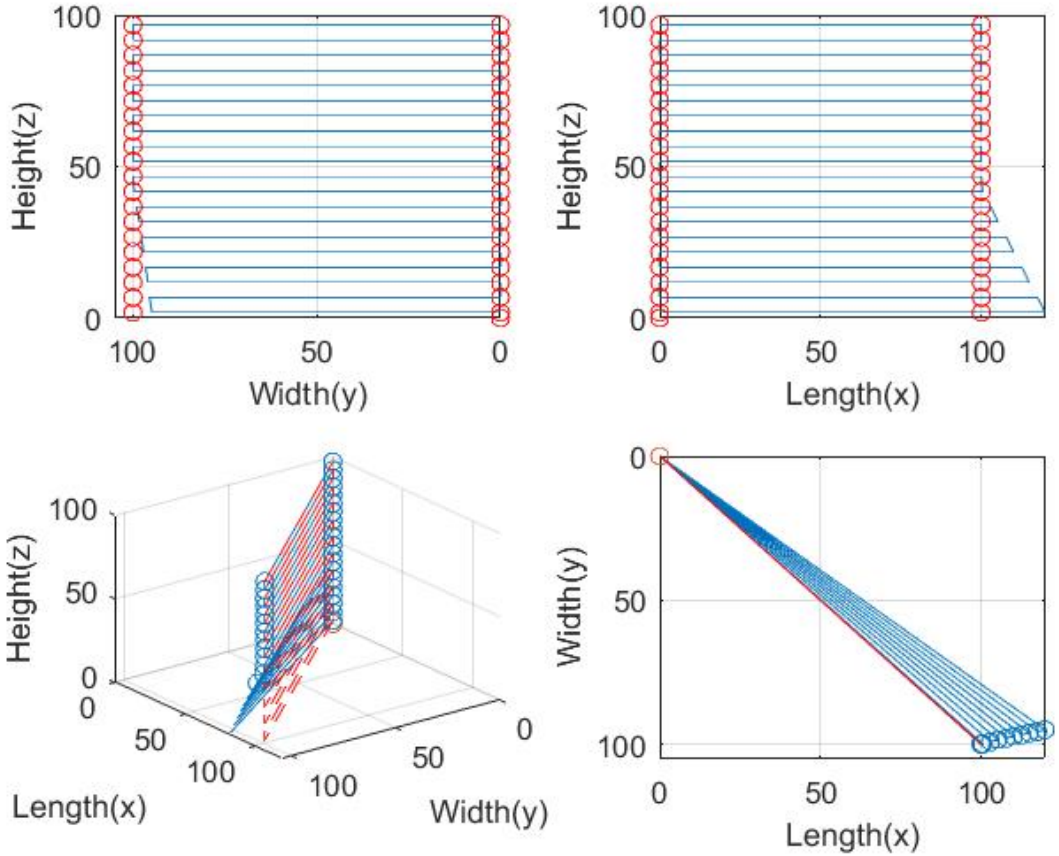


Figure C.27: (v10).2

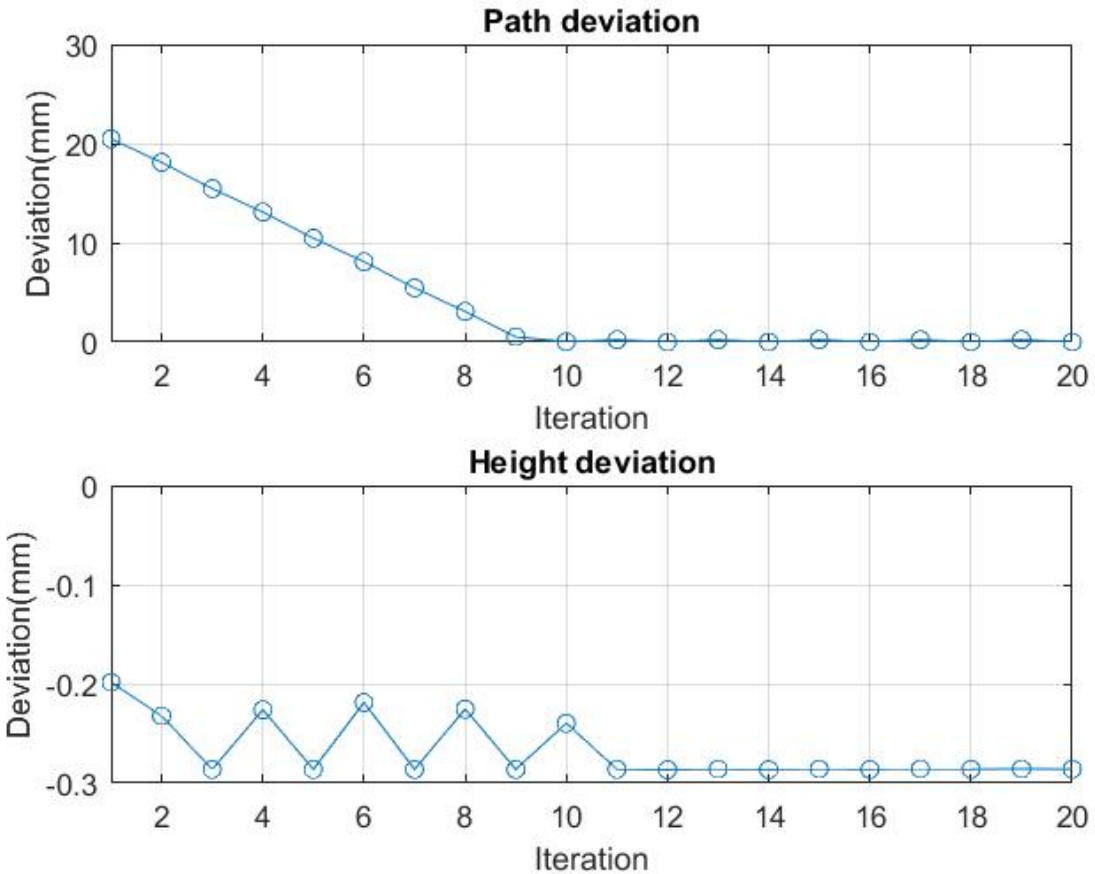


Figure C.28: (v10).2dev

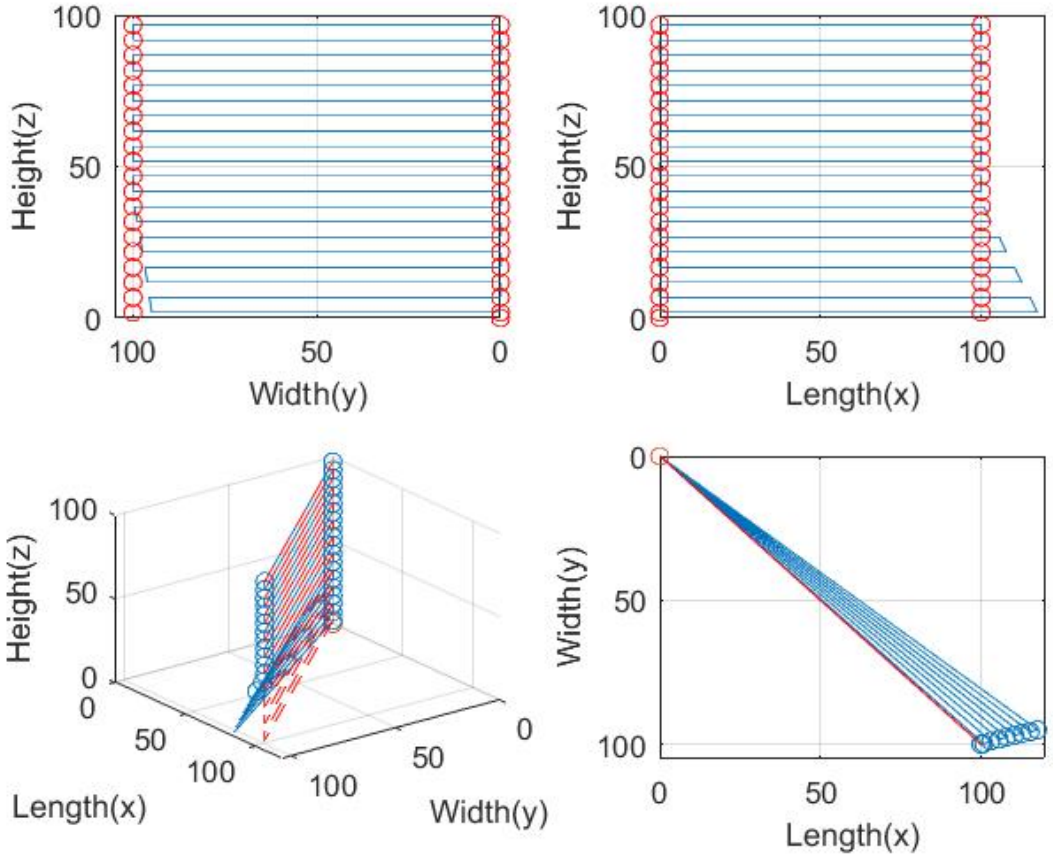


Figure C.29: (v10).3

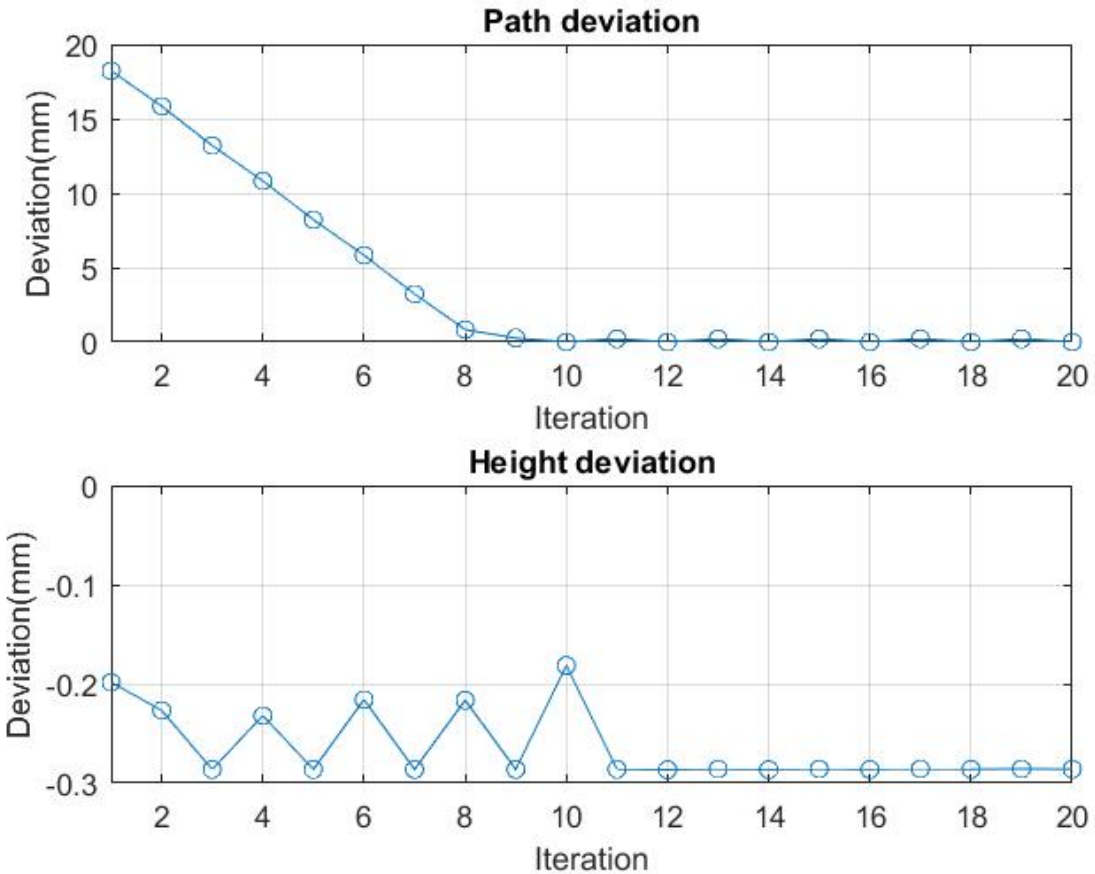


Figure C.30: (v10).3dev

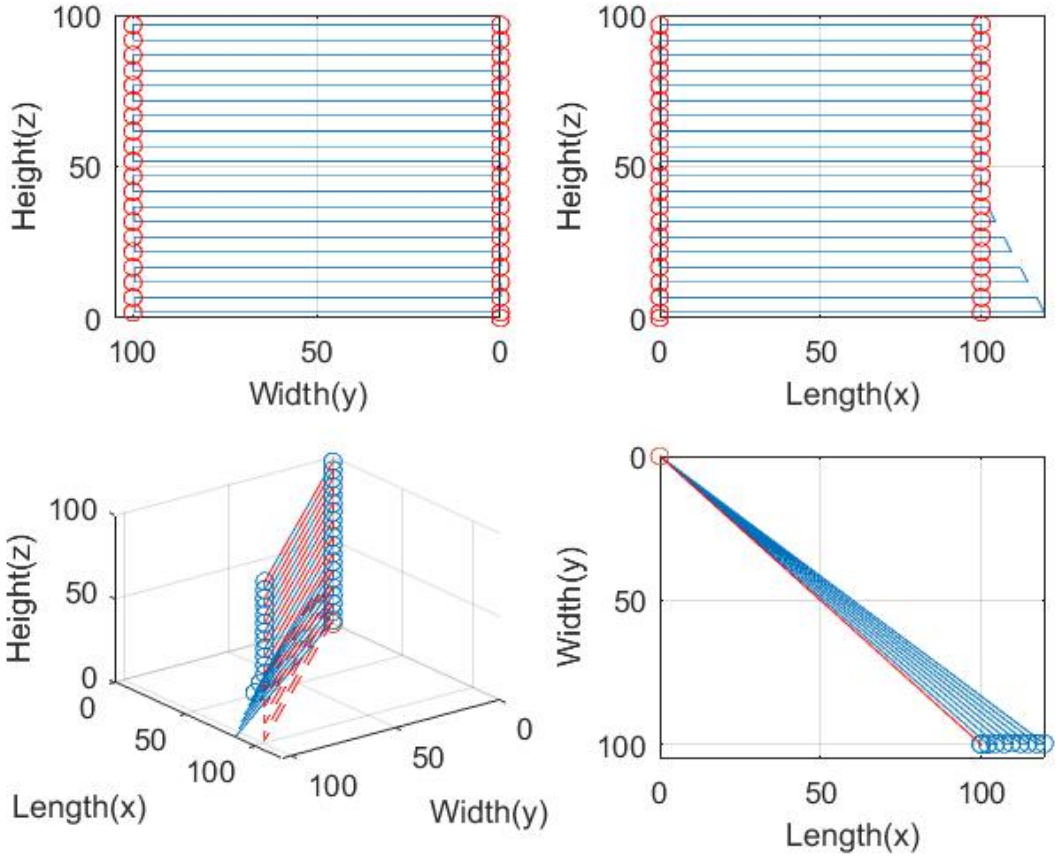


Figure C.31: (v10).4

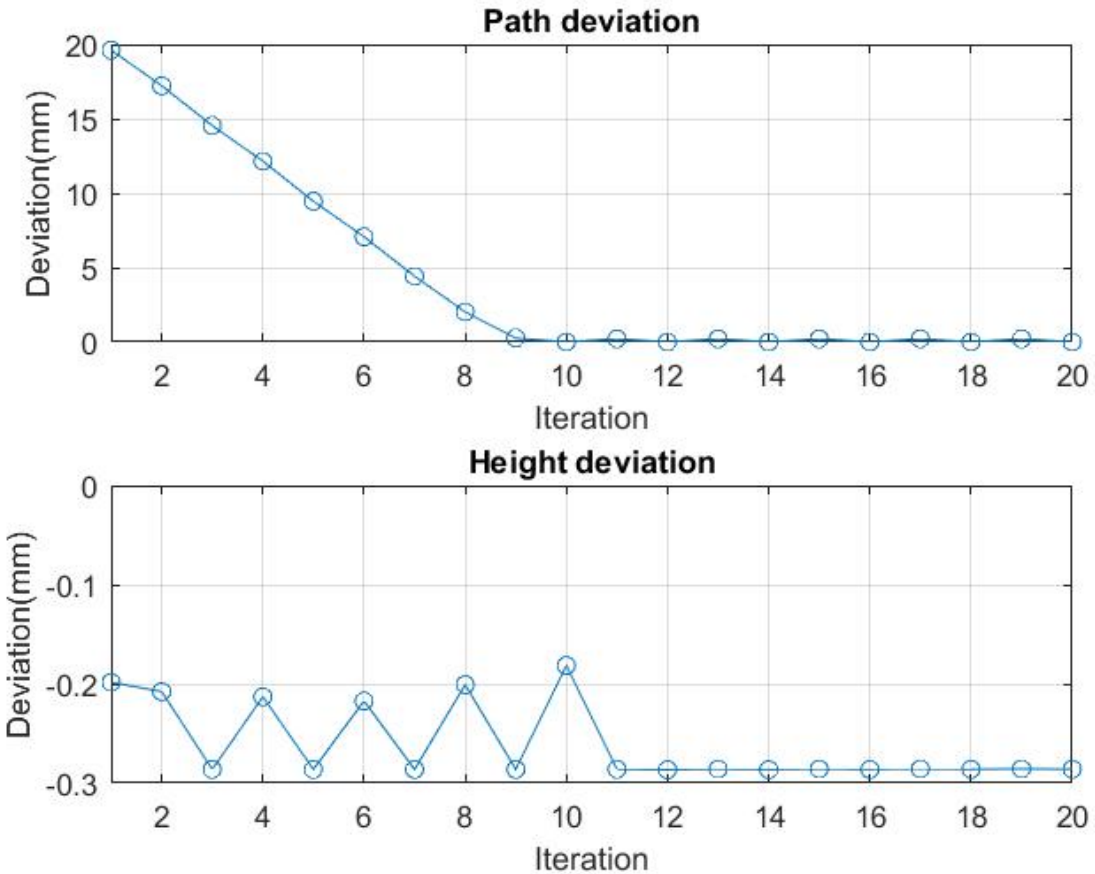


Figure C.32: (v10).4dev

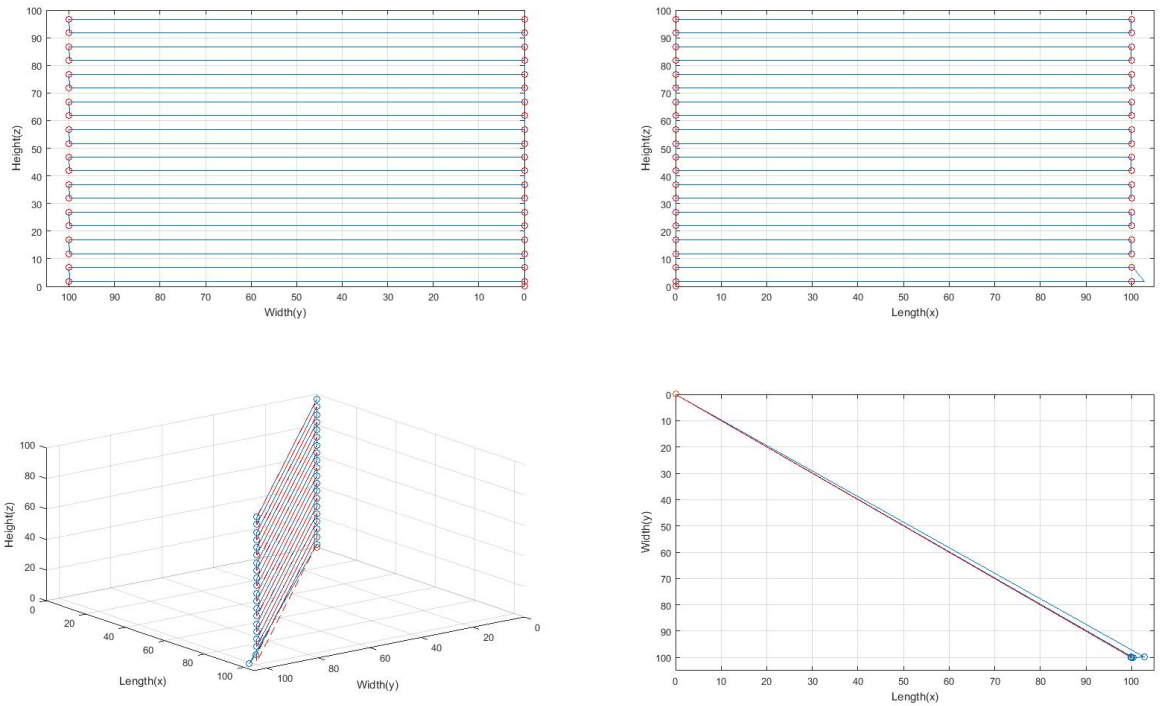


Figure C.33: (v100).1

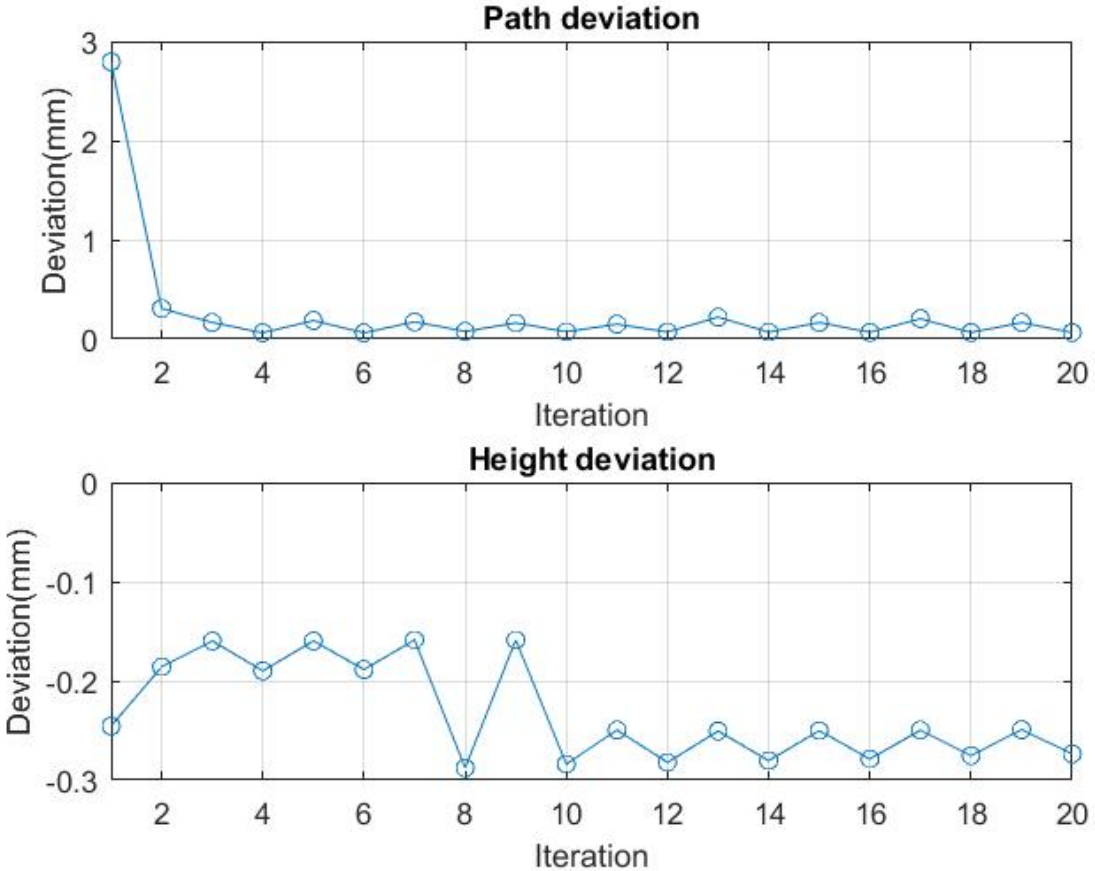


Figure C.34: (v100).1dev

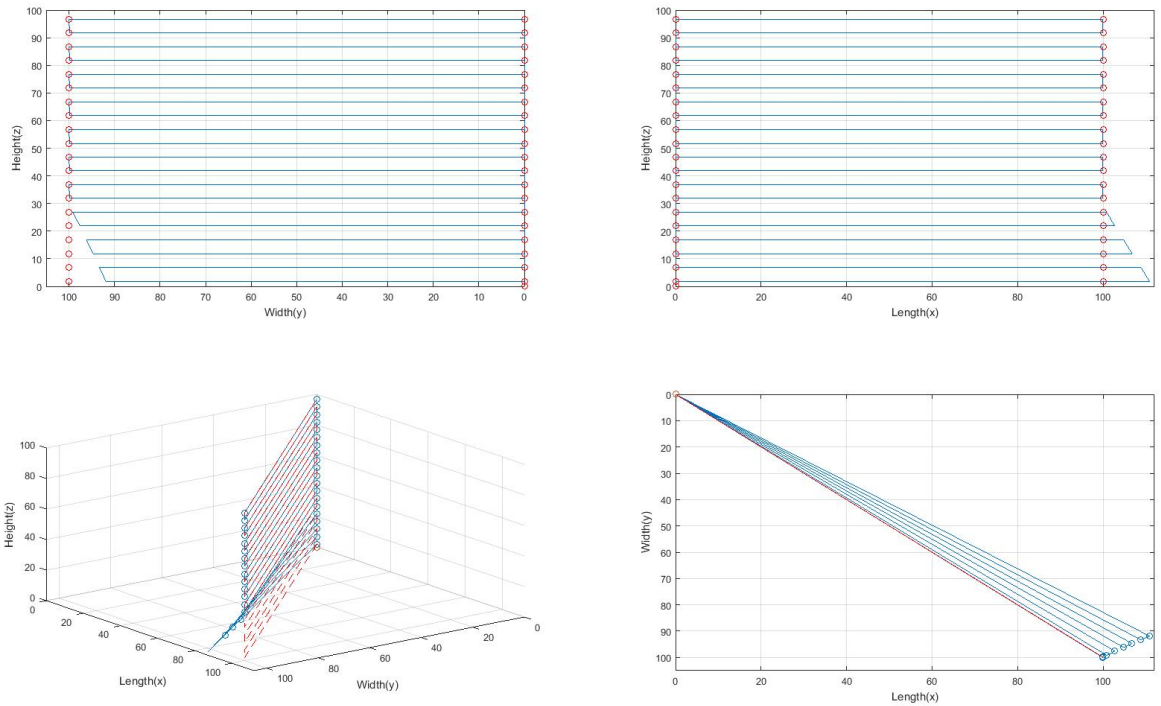


Figure C.35: (v100).2

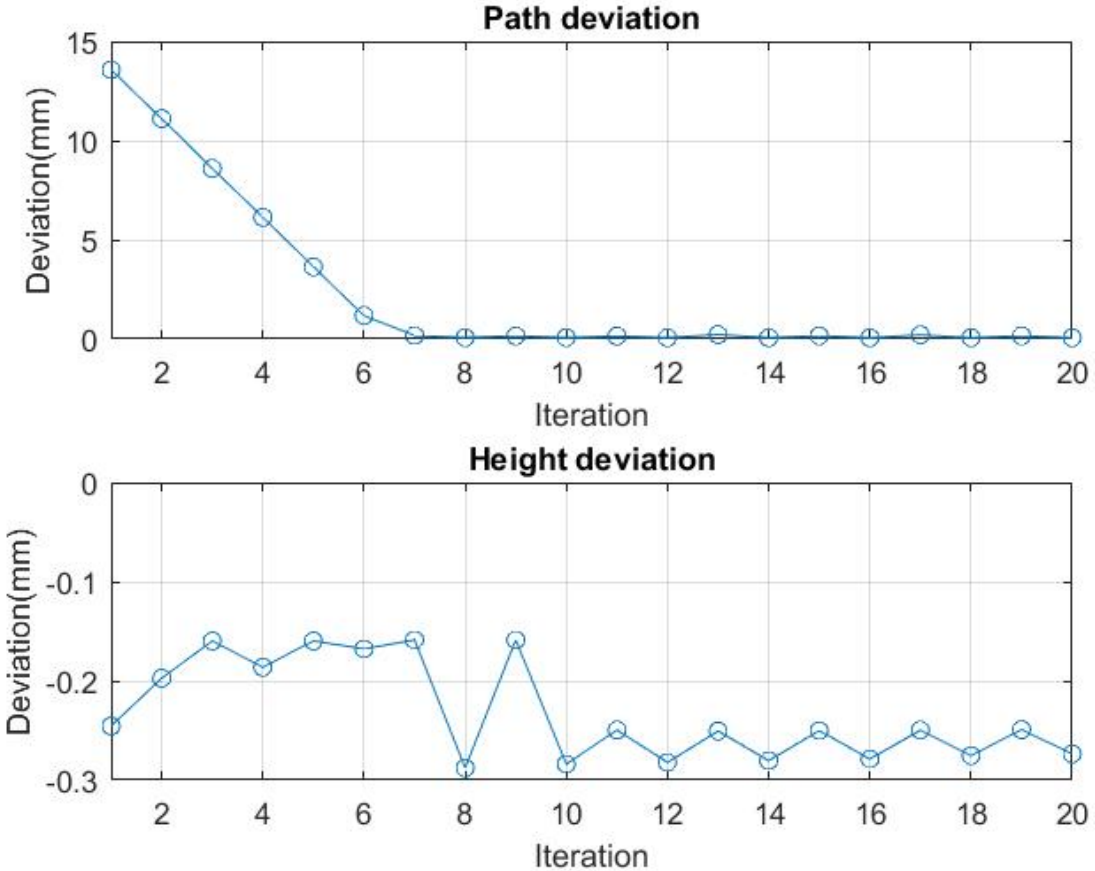


Figure C.36: (v100).2dev

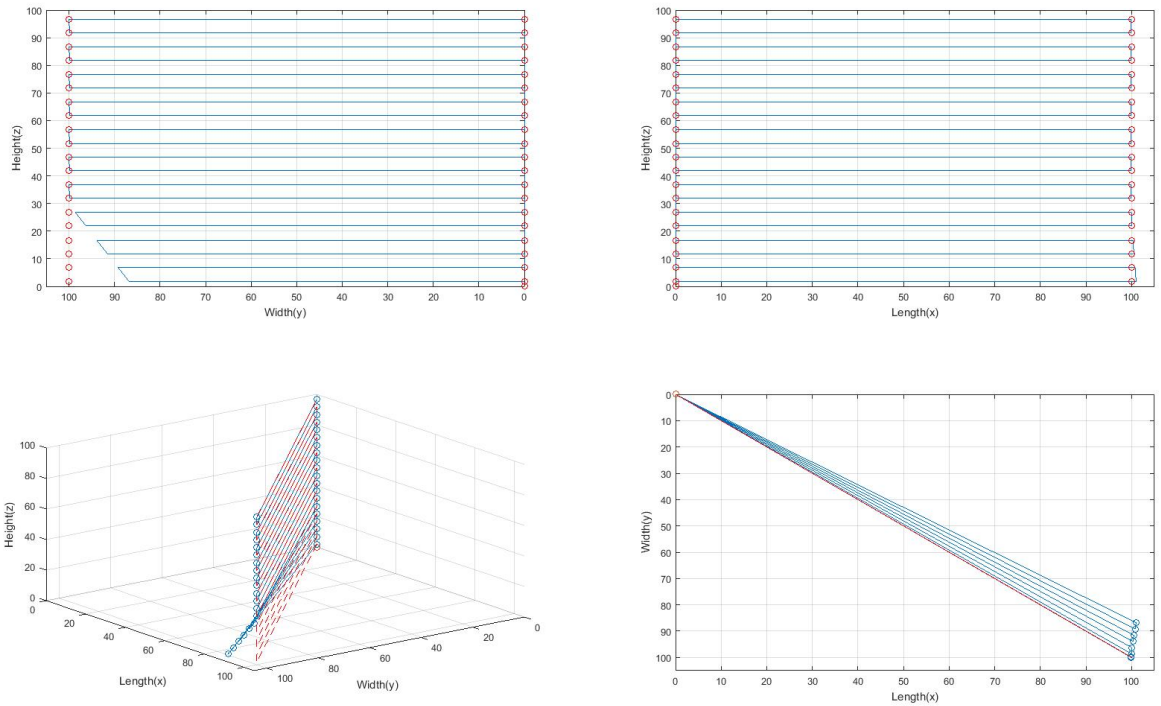


Figure C.37: (v100).3

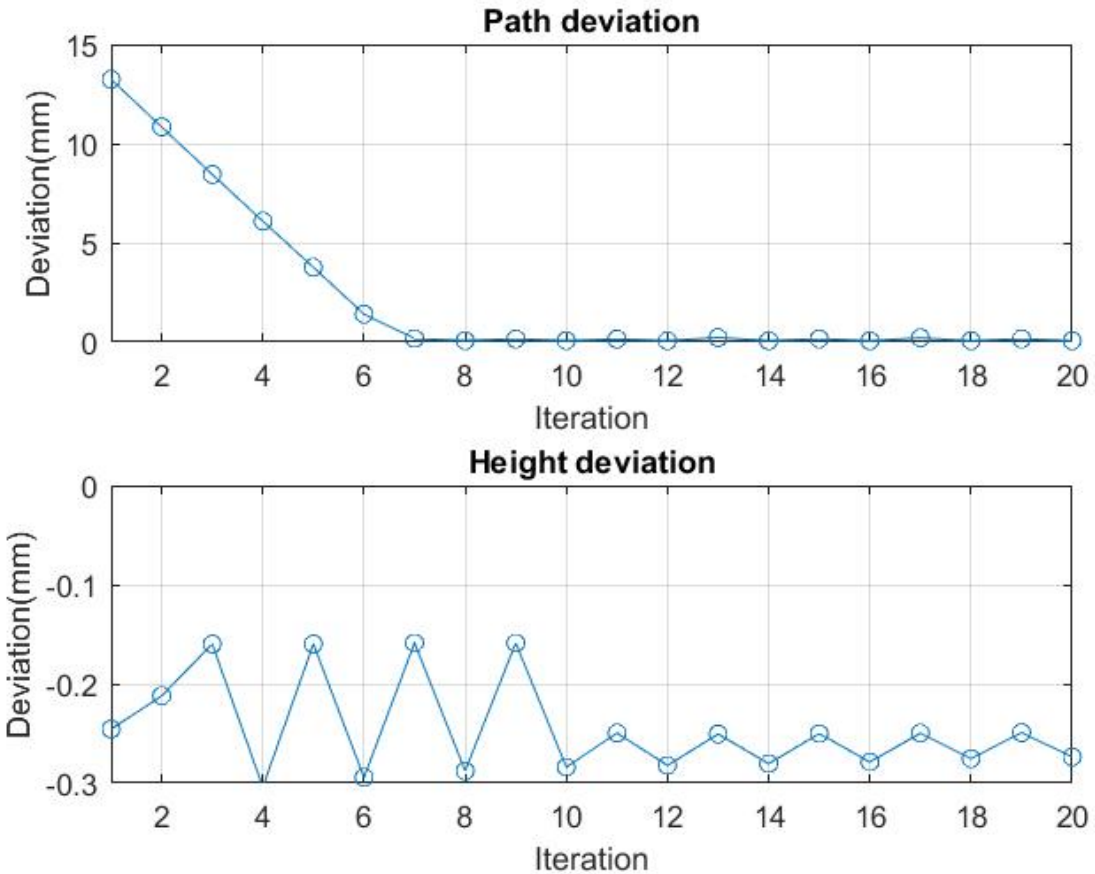


Figure C.38: (v100).3dev

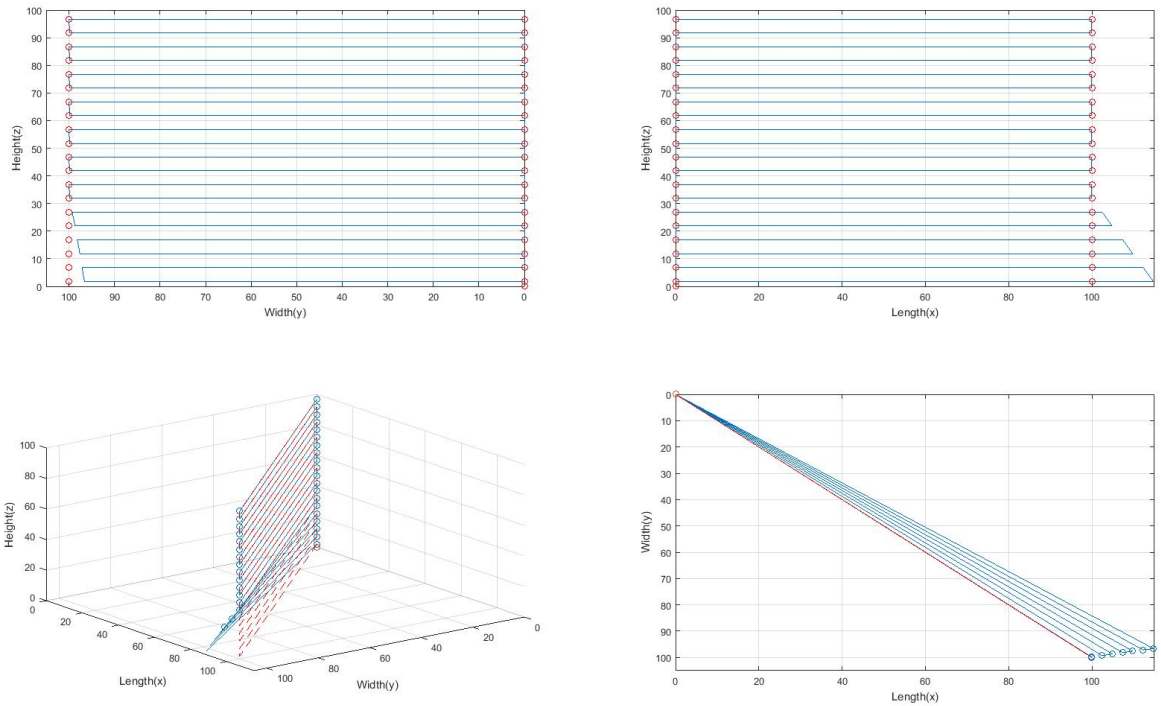


Figure C.39: (v100).4

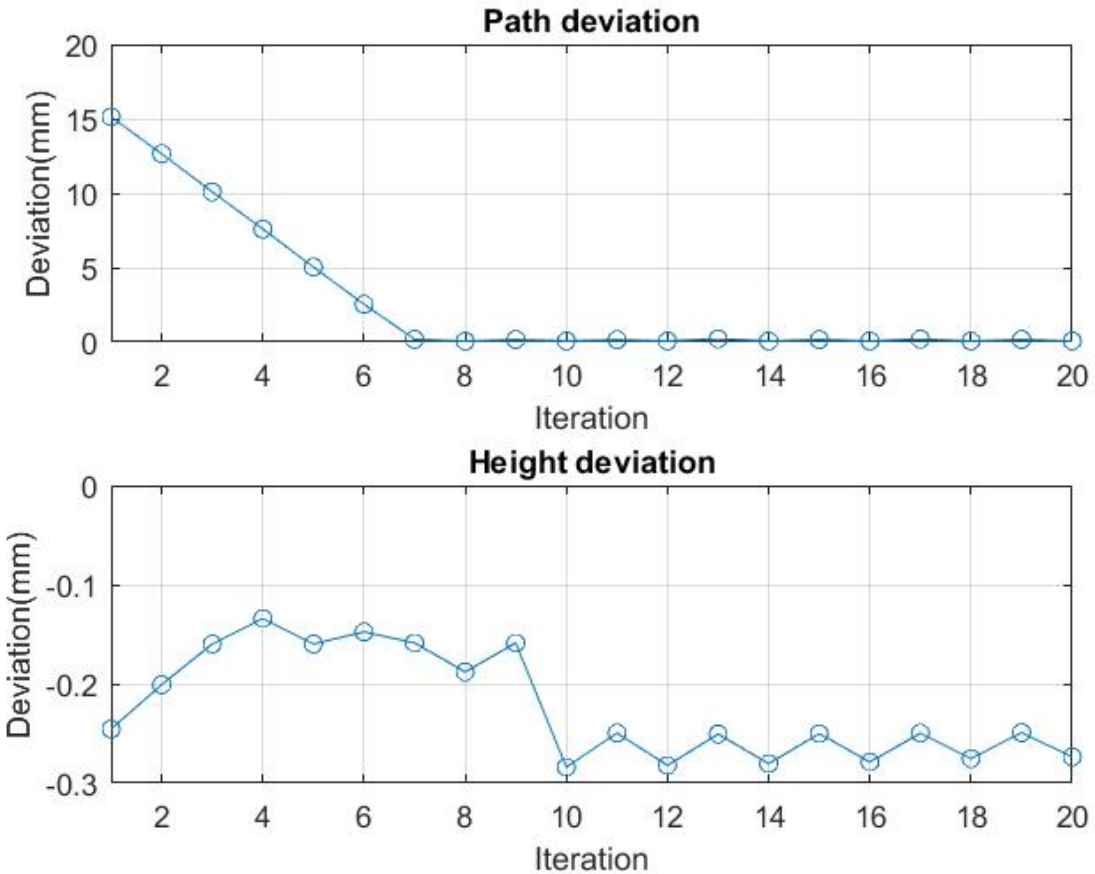


Figure C.40: (v100).4dev

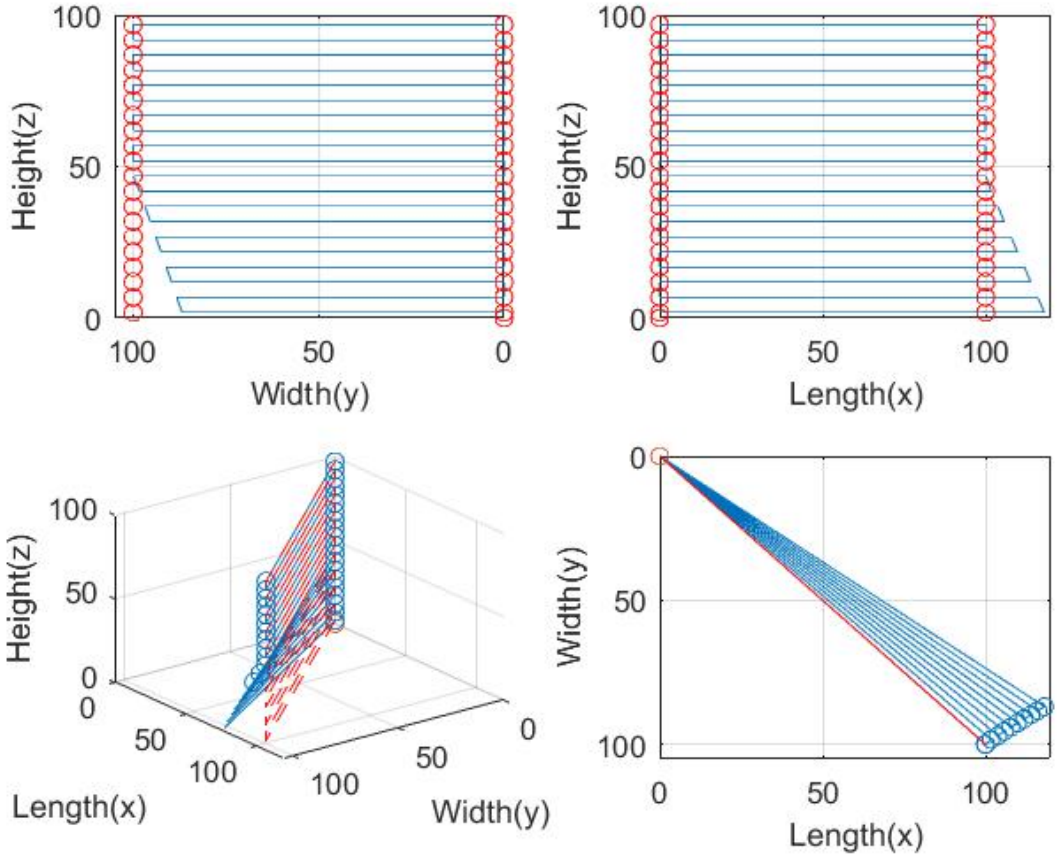


Figure C.41: (v1000).1

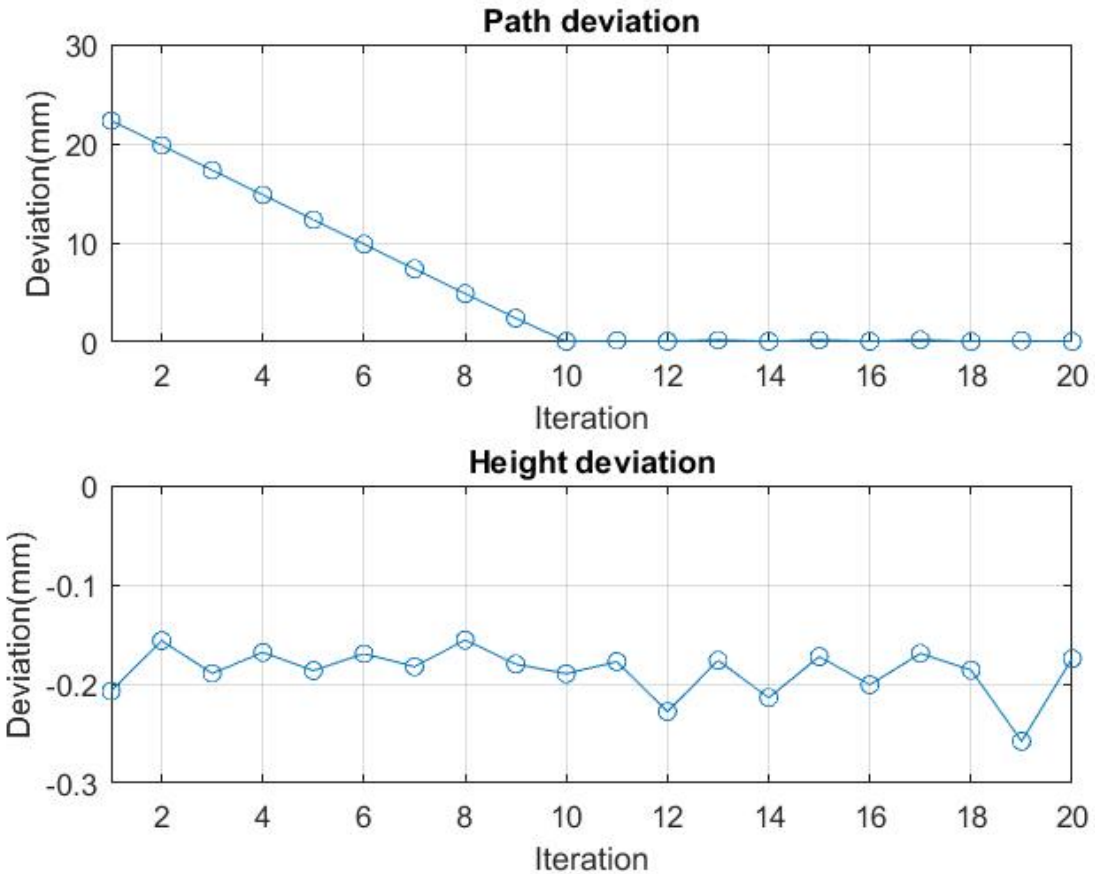


Figure C.42: (v1000).1dev

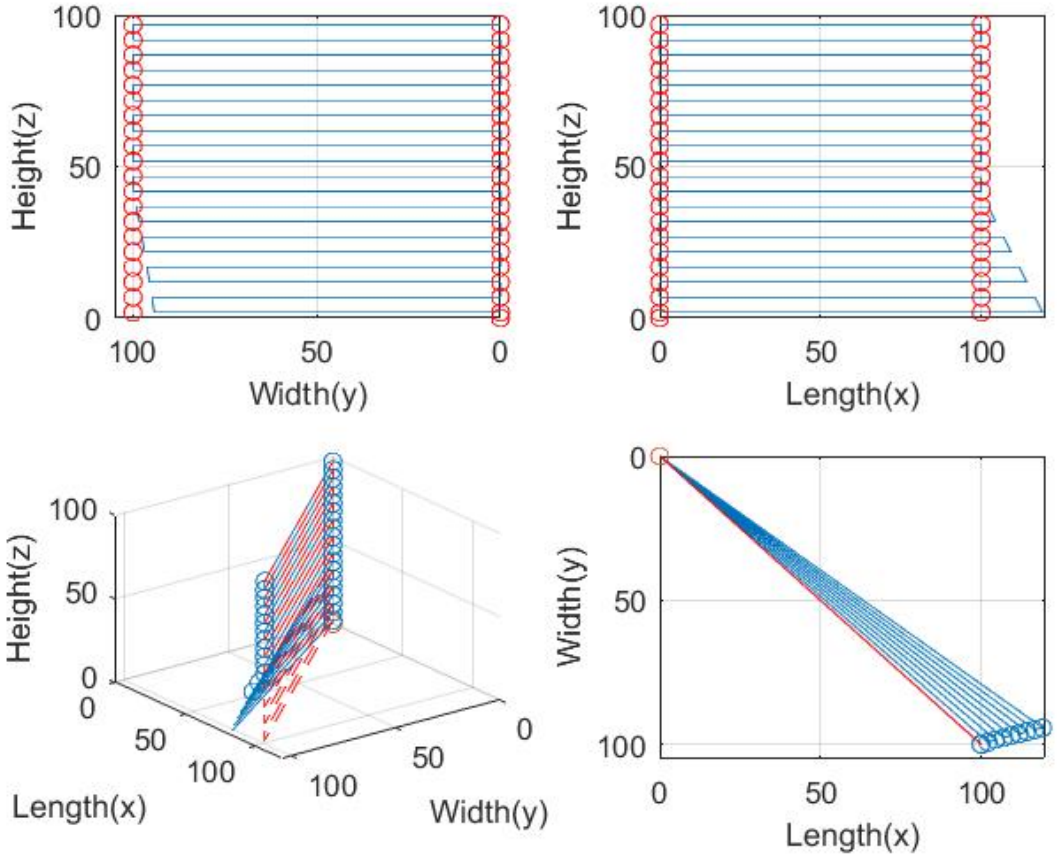


Figure C.43: (v1000).2

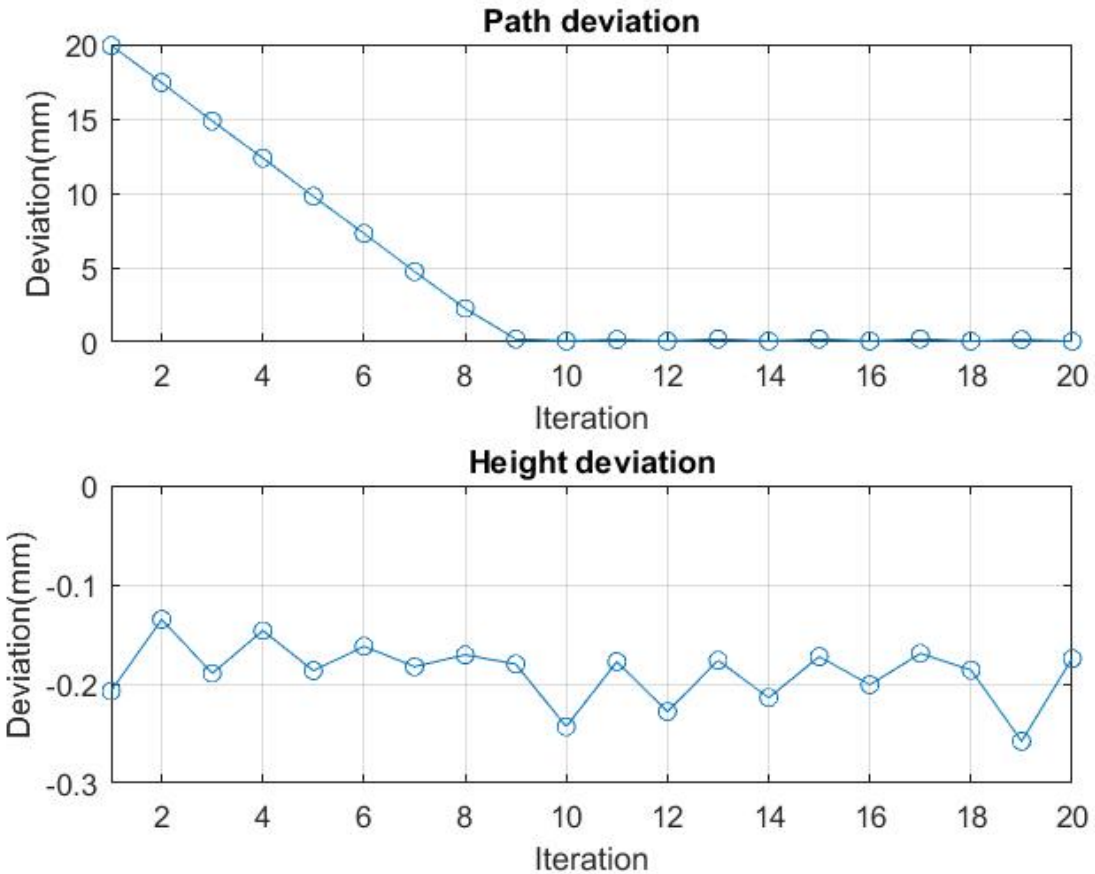


Figure C.44: (v1000).2dev

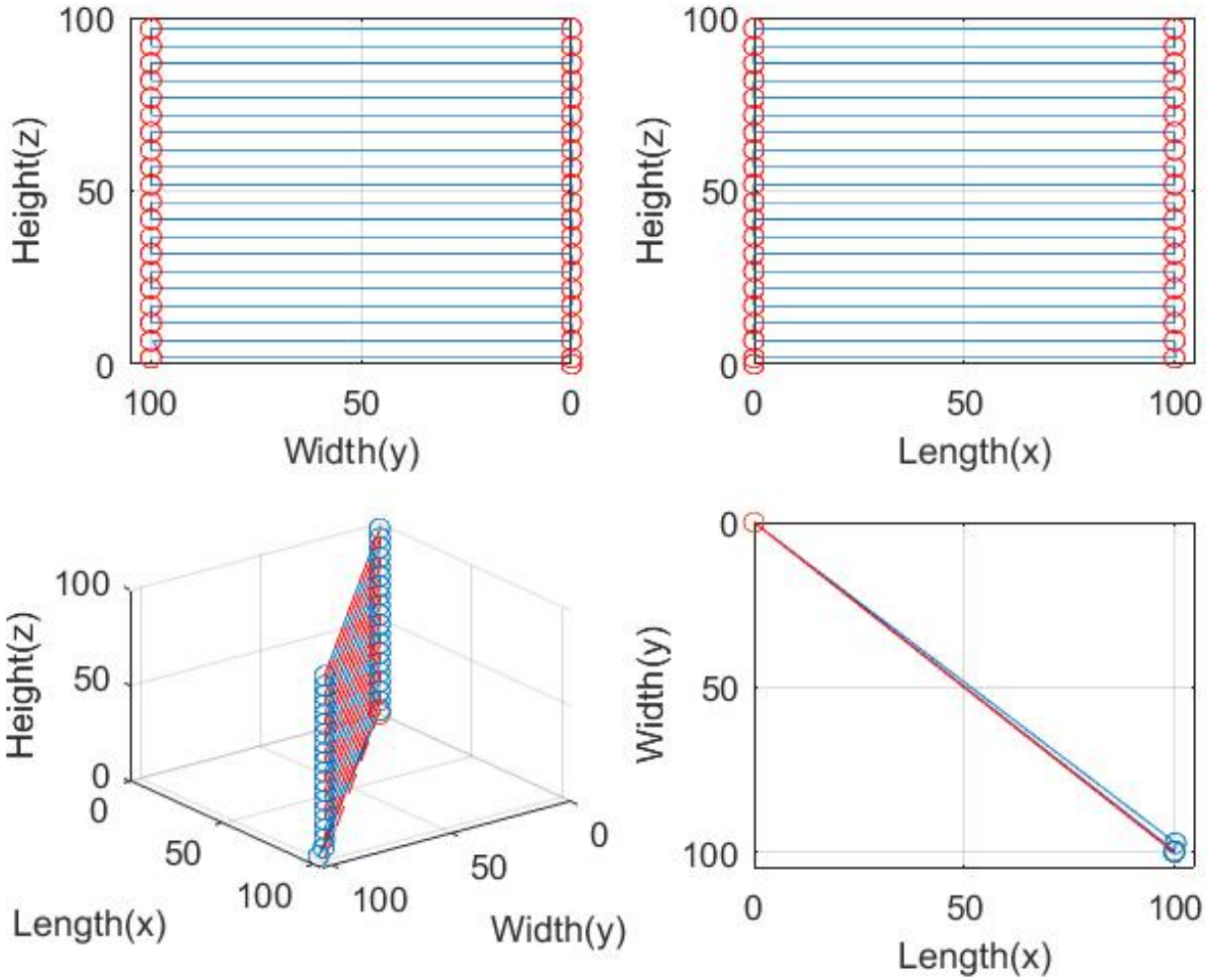


Figure C.45: (v1000).3

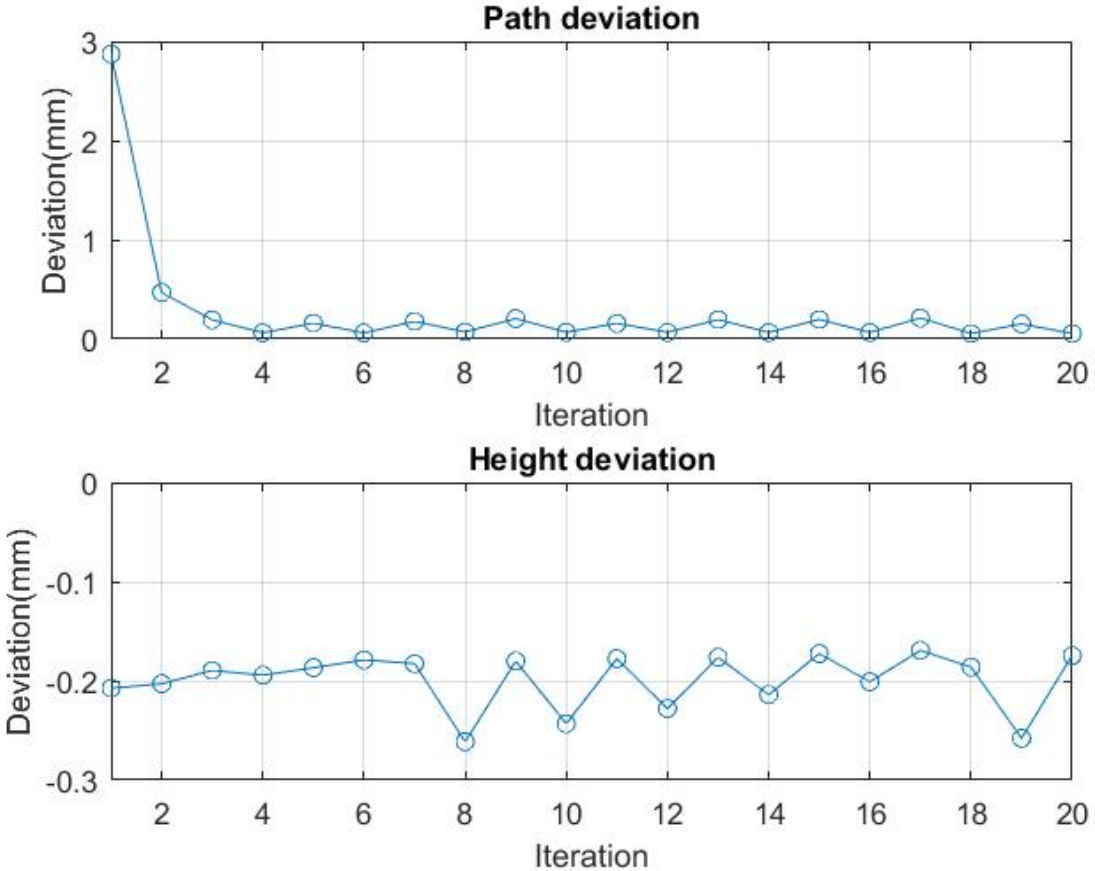


Figure C.46: (v1000).3dev

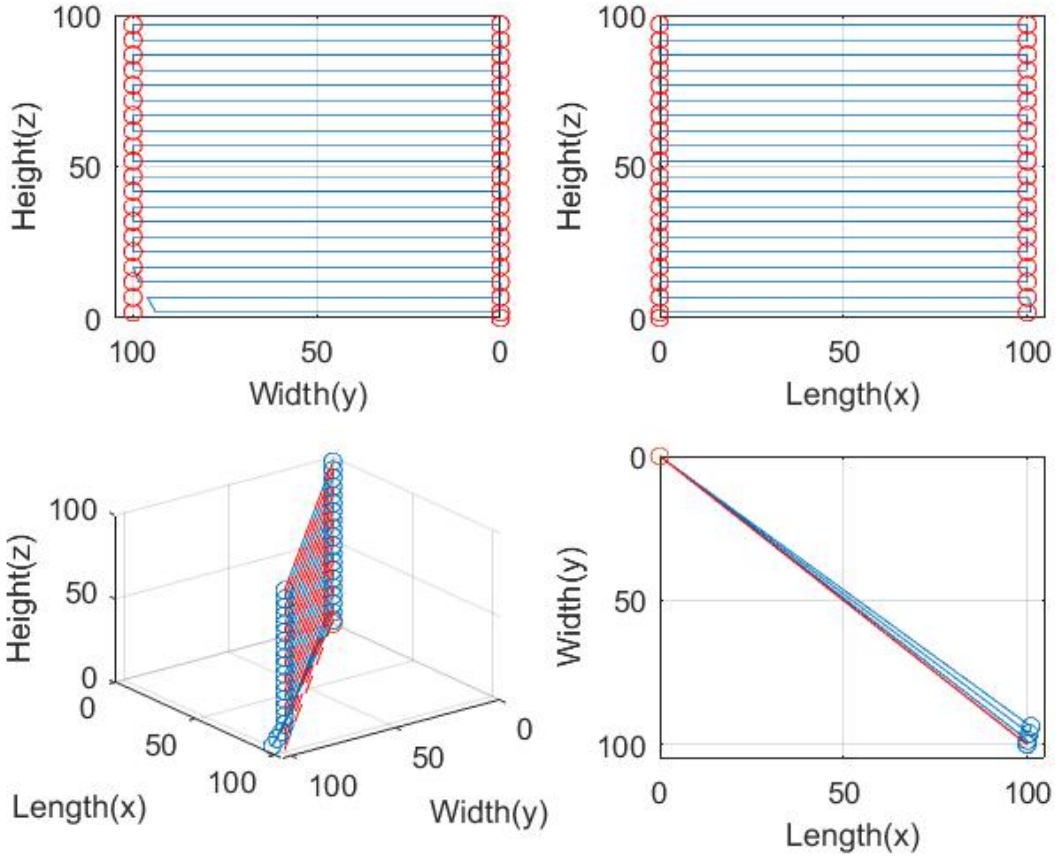


Figure C.47: (v1000).4

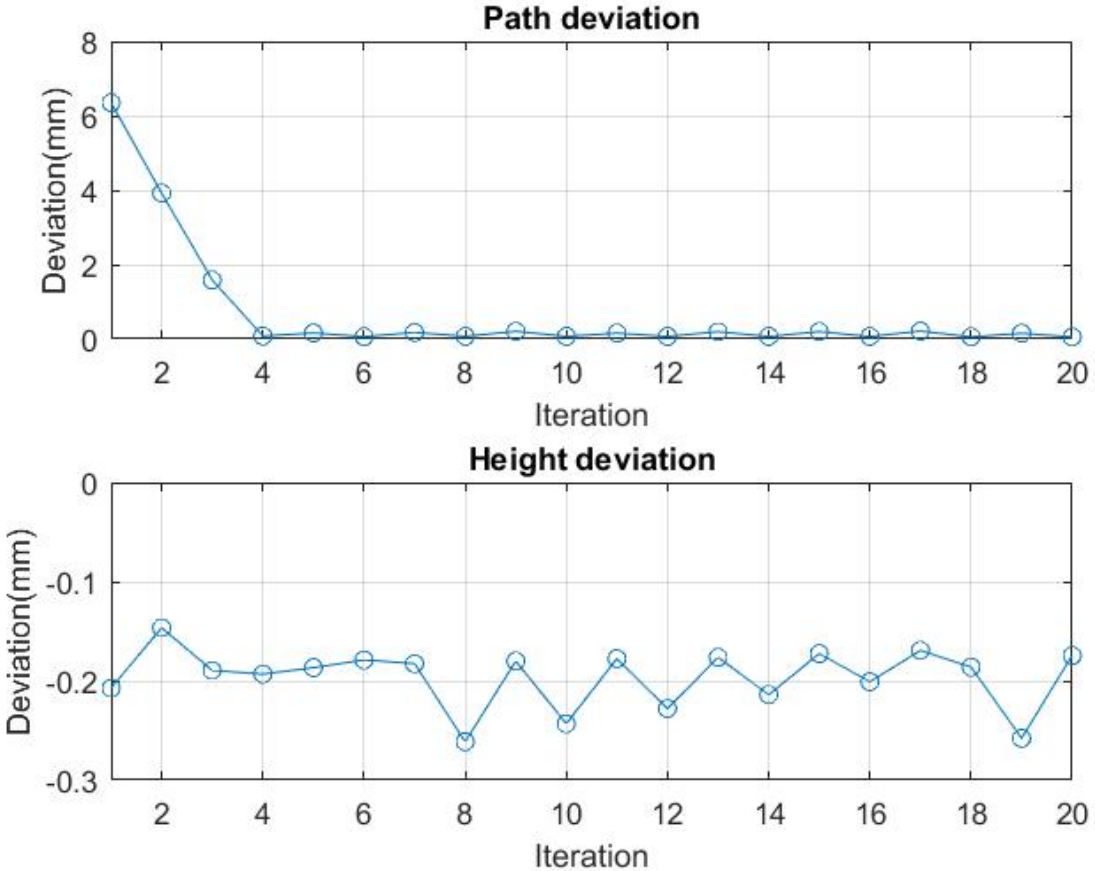


Figure C.48: (v1000).4dev

IRB140

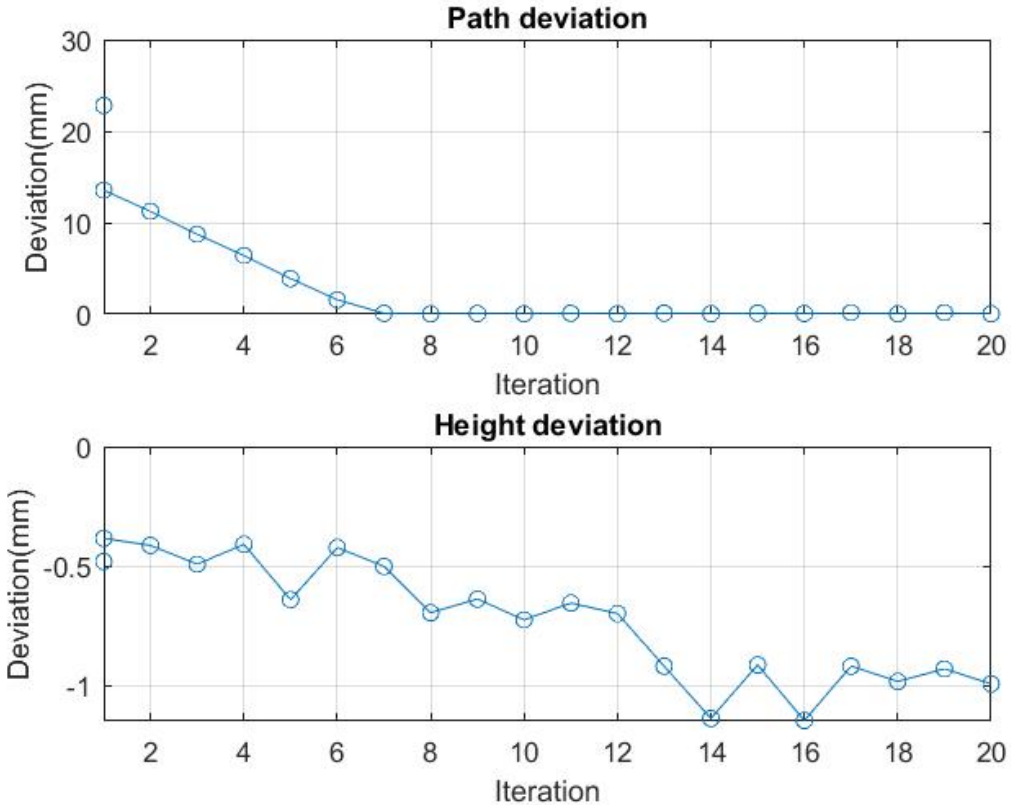


Figure C.49: (100-2-5).2dev in IRB140

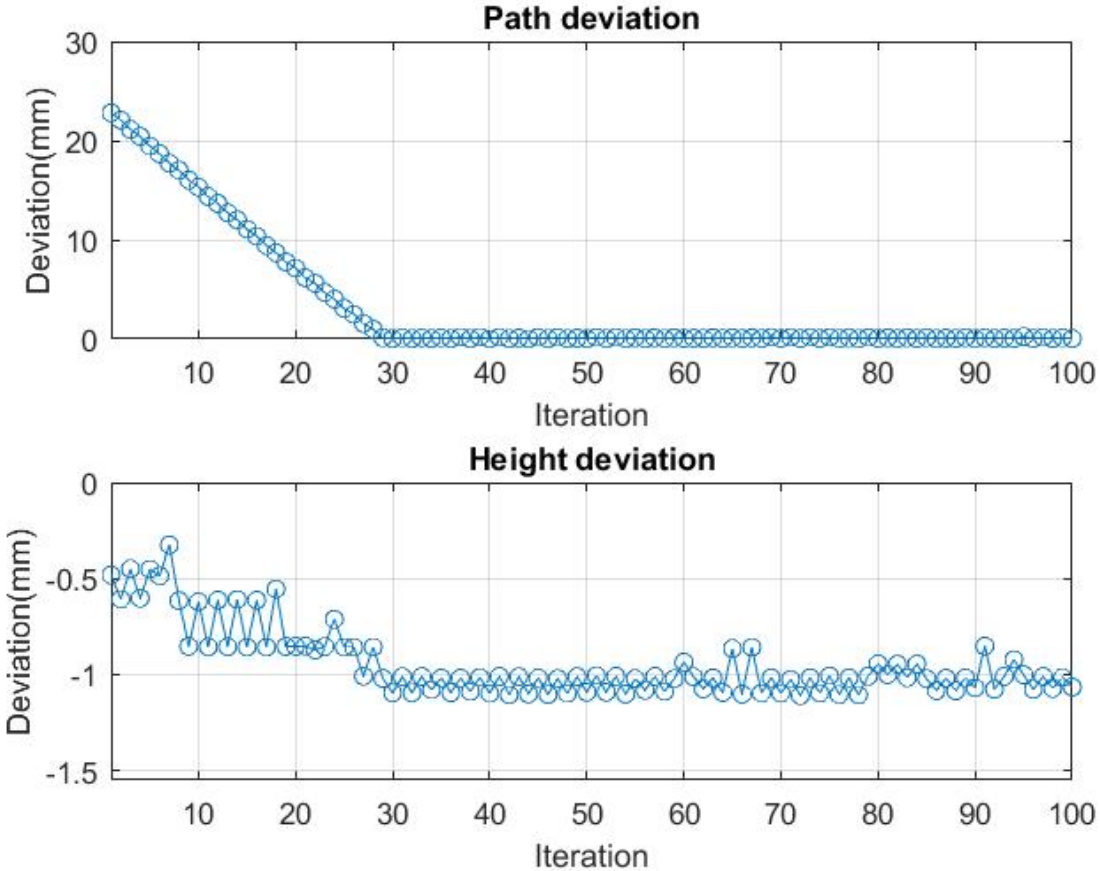


Figure C.50: (200-1-2).1dev in IRB140

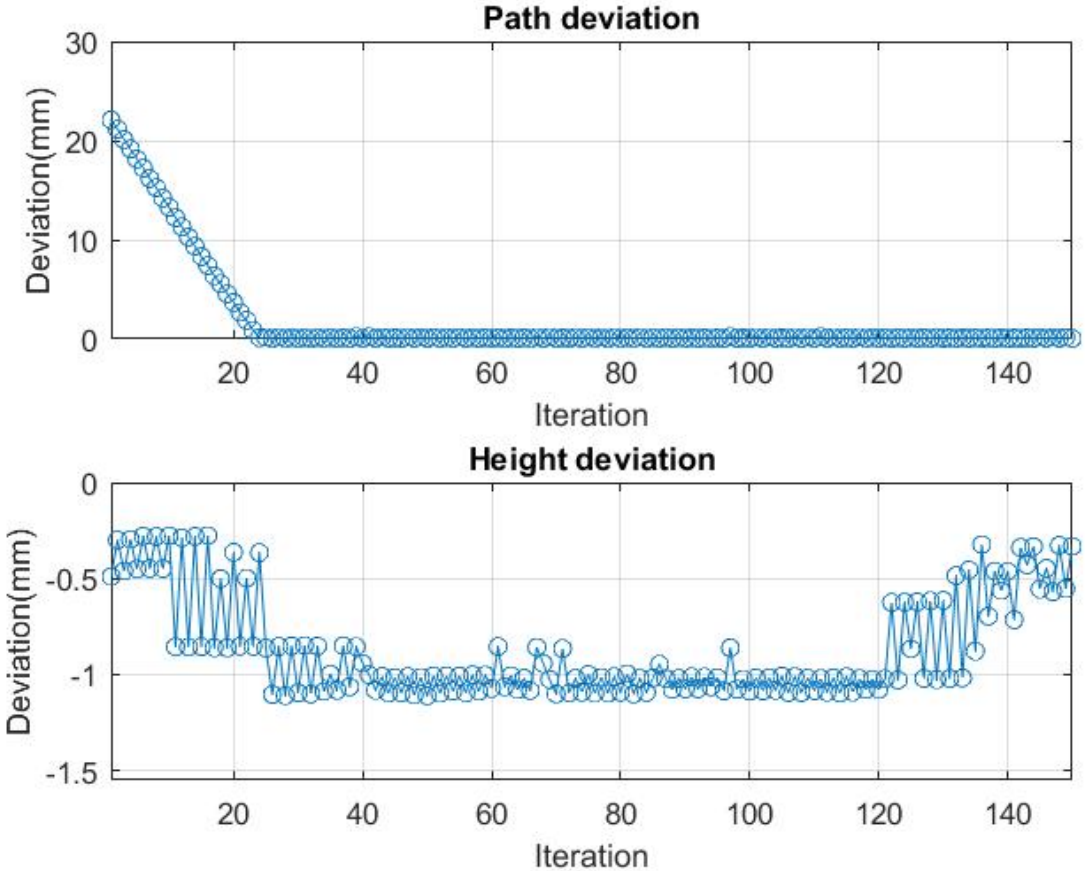


Figure C.51: (300-1-2).1dev in IRB140

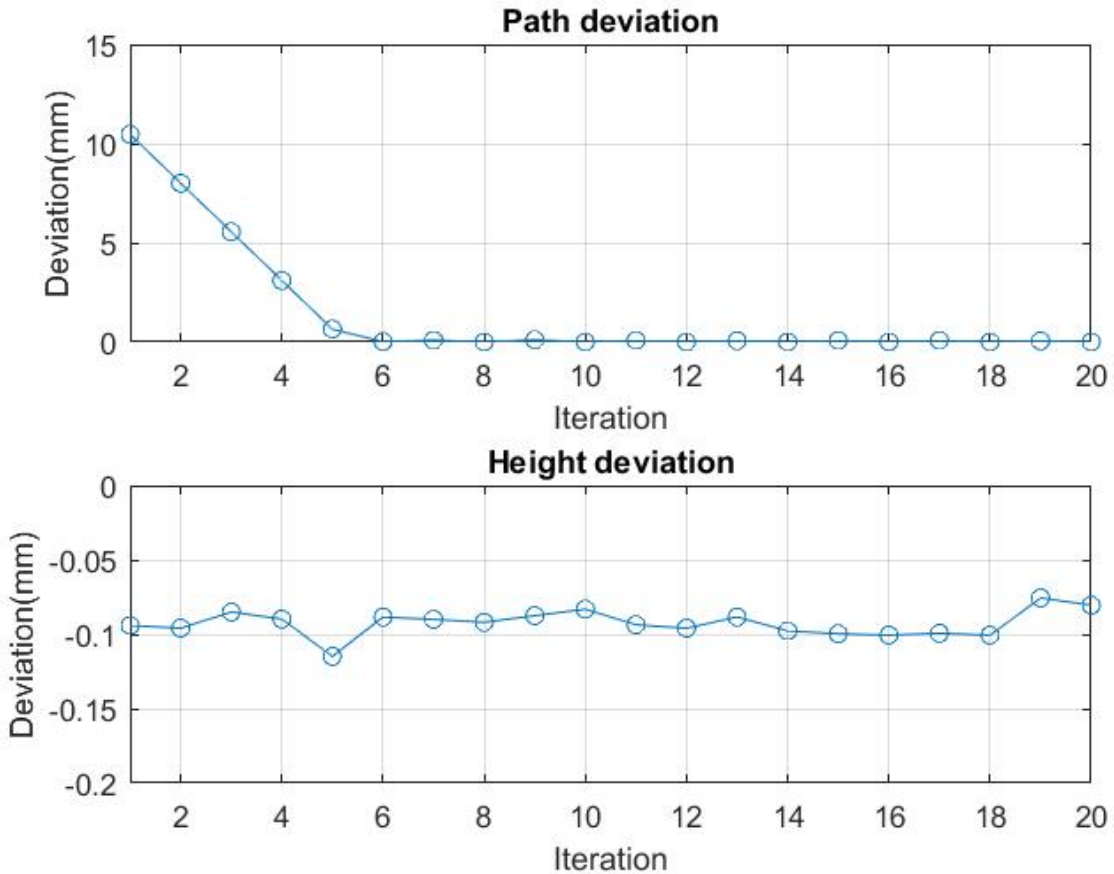


Figure C.52: (v10).1dev in IRB140

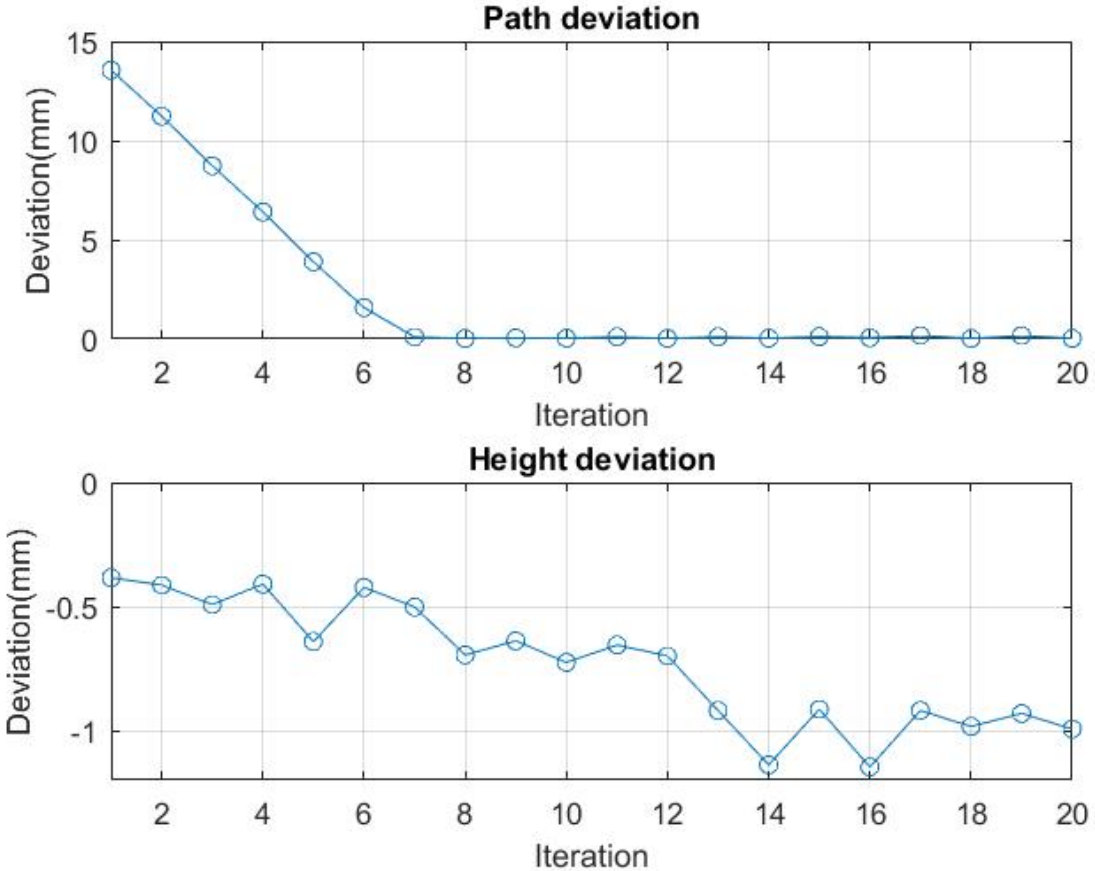


Figure C.53: (v100).2dev in IRB140

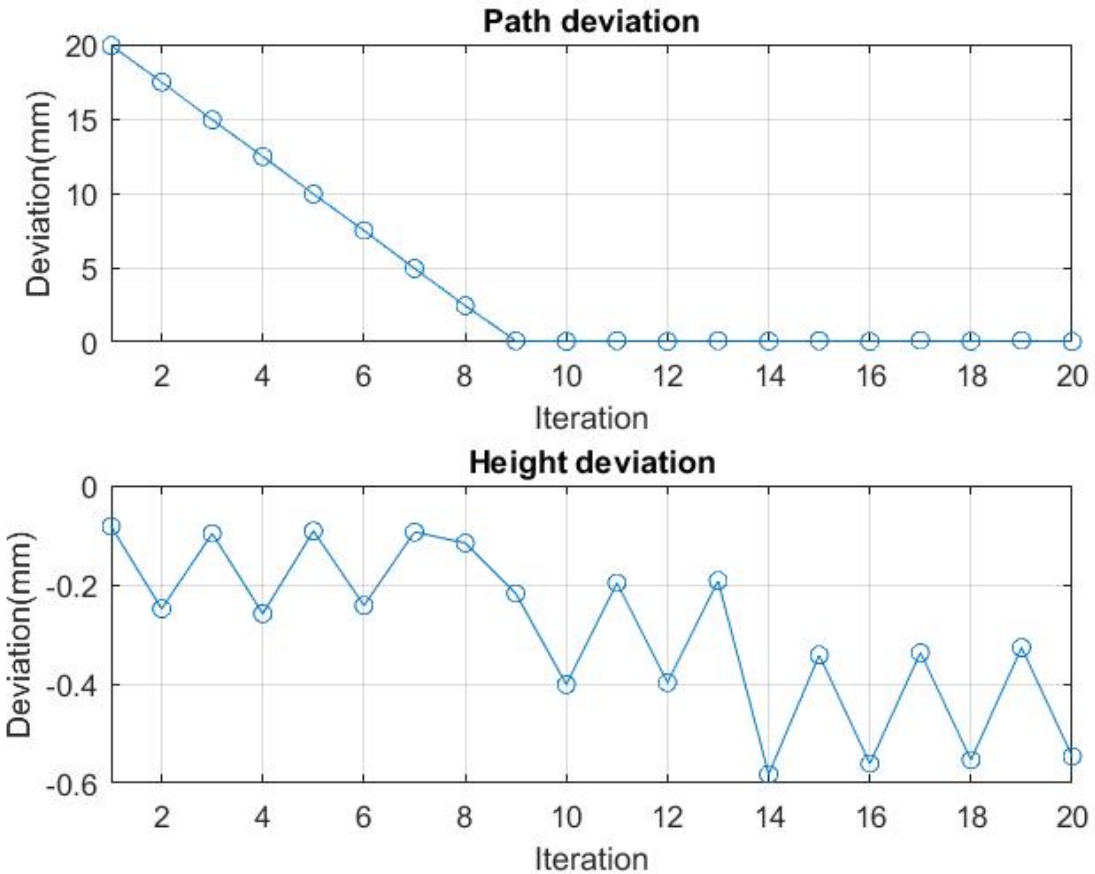


Figure C.54: (v1000).2dev in IRB140

Appendix D

RAPID Code

```
MODULE straigth_line
    VAR robtarget home := [[532.304,0,0],[3.3478E-07,-8.05985E-08,-1,-2.34526E-
08],[0,2,2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget p0;
    VAR robtarget p10;
    VAR robtarget p10_theoretical;
    VAR robtarget finish_point;
    VAR num z_update;
    VAR num iteration;
    VAR num dev{300};
    VAR num z_array{300};
    VAR num height_error{300};
    VAR num layer;
    VAR num height;
    VAR num D_final_filament; !final diameter of the filament, once expanded
    VAR num D_initial_filament; !initial diameter of the filament, without expansion
    VAR num num_layers;
    VAR iodev executionfile;
    VAR num callCount;
    VAR num delta_x;
    VAR num delta_y;
    VAR num alpha;

    PROC get_data()
        TPReadnum height, "Which is the height of the structure to build?";
        TPReadNum D_initial_filament, "Which is the diameter of the filament without
expansion?";
        TPreadnum D_final_filament, "Which is the diameter of the filament once it has
expanded?";
        num_layers := height DIV D_final_filament;
        TPWrite "Execution starts.";
    ENDPROC

    PROC calculate_error(num i)
        delta_y := Abs(p10.trans.y - p10_theoretical.trans.y);
        delta_x := Abs(p10.trans.x - p10_theoretical.trans.x);
        dev{i} := Sqrt(Pow(delta_x,2) + Pow(delta_y,2));
        TPWrite "The deviation in this layer is:" \Num:=dev{i};
    ENDPROC

    PROC check_deviation(num layer)
        IF dev{layer} > (D_final_filament/2) THEN
            TPWrite "The new p10 is:"\Pos:=p10.trans;
            alpha := ATan(delta_y/delta_x);
            p10 := Offs(p10, -D_final_filament*Cos(alpha)/2, D_final_filament*Sin(alpha)/2,0);
            TPWrite "The new p10 is:"\Pos:=p10.trans;
        ELSE
            p10 := p10_theoretical;
        ENDIF
    ENDPROC
```

```
    PROC update_height(num layer)
        z_update := D_initial_filament + D_final_filament*(layer - 1); !First layer must be
deposited D_filament above the starting point (surface).
        p10.trans.z := z_update;
        p0.trans.z := z_update;
    ENDPROC

    PROC Path_0_10(num layer)
        update_height(layer);
        MoveL p0,v100,fine,tool0\WObj:=wobj0;
        p0 := Crobt (\Tool:=tool0 \WObj:=wobj0);
        z_array{layer} := p0.trans.z;
        height_error{layer} := z_array{layer} - z_update;
        TPWrite "the current position is: "\Pos:=p0.trans;
        MoveL p10,v100,fine,tool0\WObj:=wobj0;
        p10 := Crobt (\Tool:=tool0 \WObj:=wobj0);
    ENDPROC

    PROC Path_10_0(num layer)
        update_height(layer);
        MoveL p10,v100,fine,tool0\WObj:=wobj0;
        p10 := Crobt (\Tool:=tool0 \WObj:=wobj0);
        z_array{layer} := p10.trans.z;
        height_error{layer} := z_array{layer} - z_update;
        TPWrite "the current position is: "\Pos:=p10.trans;
        MoveL p0,v100,fine,tool0\WObj:=wobj0;
    ENDPROC


    !Function to generate random numbers for deviation
    !taken from: https://forums.robotstudio.com/discussion/3742/random-value-needed
    FUNC NUM RANDOM()
        VAR num val:=1;
        VAR pos pos_current;
        pos_current := CPos(\Tool:=tool0 \WObj:=wobj0);

        !1 seeding, Seed by position
        val := val*(Abs(ROUND((pos_current.x*100) \Dec:=0)) +1); !+1 to avoid mul by 0
scenario
        val := 789 + val MOD 1000; !777 is just a bogus valu, since the mod might give 0

        !Feel free to seed with y,z, more

        !2. seeding, Seed by time
        val := val * (GetTime(\Sec)+1);
        val := val * (GetTime(\Min)+1);

        !3 seeding, increment callcount and handle large nuber
        callCount := callCount +1;
```

```
        IF callCount > 1000 THEN
            callCount := 234;
        ENDIF

        !finally a division to get something interesting large float number
        val := val / callCount;

        !get a value between 0 and 1 using COS ( math)
        RETURN (0.5 + COS(val)/2);
    ENDFUNC

    PROC open_file()
        Open "d:/execution.txt", executionfile \Append;
        Write executionfile, "Execution started.";
        Write executionfile, " Total Height (mm): "\Num:=height;
        Write executionfile, " Initial diameter of the filament (mm): "\Num:=D_initial_filament;
        Write executionfile, " Final diameter of the filament (mm): "\Num:=D_final_filament;
        Write executionfile, " Number of layers: "\Num:=num_layers;
        Write executionfile, " Random deviation created in this execution: ";
        Write executionfile, "    x : "\Num:= delta_x;
        Write executionfile, "    y : "\Num:= delta_y;
        Write executionfile, "Iteration : Path deviation : Length(x) : Length error(?x)
"\NoNewLine ;
        Write executionfile, ": Width(y) : Width error(?y): Height(z) : Height error(?z)";
    ENDPROC

    PROC close_file()
        Write executionfile, "Execution finished.";
        Close executionfile;
    ENDPROC


    PROC initialize_variables()
        delta_y := RANDOM() * 20;
        delta_x := RANDOM() * 20;
        open_file;
        p10_theoretical := Offs(home,100,100,0);
        p10 := Offs(p10_theoretical,delta_x,-delta_y,0);
        p0 := home;
        iteration := 2;
        FOR i FROM 1 TO (num_layers+1) DO
            dev{i} := 0;
            z_array{i} := 0;
            height_error{i} := 0;
        ENDFOR
    ENDPROC

    PROC write_to_file(num i)
        Write executionfile, ""\Num:=i\NoNewLine;
        Write executionfile, " : "\Num:=dev{i}\NoNewLine;
```

```
        Write executionfile, " : "\Num:=(p10.trans.x-home.trans.x)\NoNewLine;
        Write executionfile, " : "\Num:=delta_x\NoNewLine;
        Write executionfile, " : "\Num:=(p10.trans.y-home.trans.y)\NoNewLine;
        Write executionfile, " : "\Num:=delta_y\NoNewLine;
        Write executionfile, " : "\Num:=z_array{i}\NoNewLine;
        Write executionfile, " : "\Num:=height_error{i};
    ENDPROC

    PROC main()
        TPErase;
        get_data;
        MoveL home,v100,fine,tool0\WObj:=wobj0;
        initialize_variables;
        FOR i FROM 1 TO (num_layers + 1) DO
            IF (i MOD 2) = 1 THEN
                Path_0_10(i);
            ELSE
                Path_10_0(i);
            ENDIF
            calculate_error(i);
            write_to_file(i);
            check_deviation(i);
        ENDFOR
        finish_point:=CRobT(\Tool:=tool0 \WObj:=wobj0);
        finish_point:=Offs(finish_point, 0, 0, 50);
        MoveL finish_point,v100,fine,tool0\WObj:=wobj0;
        close_file;
    ENDPROC
ENDMODULE
```