Surface Scanning and Path Planning for Non-Planar 3D Printing

Diplomarbeit

Liang Pan

Betreuer
Ass.Prof. Dipl.-Ing. Dr.mont. Gerhard Rath
O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary

Montanuniversität Leoben
Institut für Automation

October 2014

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

AFFIDAVIT

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Datum/Date

Unterschrift/Signature

Abstract

This thesis describes a new approach for achieving a non-planar 3D printing with an industrial manipulator. In recent years 3D printing is becoming a promising new industry. It is applied in more and more areas. Simultaneously, with the improvement of production efficiency, the cost for industrial robots is decreasing. Our motivation is to combine the two technologies, and to develop a method using a robot to expand the practical range of 3D printing. In this work firstly the basic concepts about robot kinematic modelling are introduced. Then the essential hard- and software equipment that is chosen to solve the problem is described. A method is presented using a laser distance sensor mounted on a robot to scan an arbitrary surface. Levenberg-Marquardt method is applied to find a least-mean-square approximation and to reconstruct the non-planar surface in real-time during the motion of the robot, which carries also a 3D printer head. For testing the algorithms a virtual reality simulation model of the robot was used. Finally the solution is successfully implemented in a Matlab/Simulink environment that controls the robot drives in real-time. With this work the feasibility of a non-planar 3D printing with an industrial robot is proved. A practicable approach consisting of surface scanning and printing is successfully applied on a real robot. At last some tracking error considerations are given.

Zusammenfassung

Diese Diplomarbeit beschreibt einen neuen Ansatz für das 3D-Drucken auf einer gekrümmten Oberfläche mit einem industriellen Manipulator. In den letzten Jahren wird das 3D-Drucken mehr und mehr zu einer vielversprechenden neuen Industrie und wird in immer mehr Anwendungsbereichen eingesetzt. Gleichzeitig sind die Kosten für Industrieroboter auf Grund gestiegener Produktivität reduziert worden. Die Motivation für diese Arbeit ist, die beiden Technologien zu kombinieren, um mit Hilfe eines Roboters neue Anwendungsbereiche des 3D-Druckens zu ermöglichen. Zu Beginn wird in die Grundkenntnisse über Roboter-Kinematik eingeführt. Es werden die verwendete Ausrüstung und die Software-Werkzeuge beschrieben. Eine Methode wird vorgestellt, um durch einen auf dem Roboter montierten Lasersensor eine beliebige Oberfläche abzutasten. Die Levenberg-Marquardt Methode wird verwendet, um aus den gemessenen Daten Parameter für eine Approximation der Oberfläche zu finden. Für die Bewegung eines auf dem Roboter montierten 3D-Druckkopfes über diese Oberfläche wird diese in Echtzeit rekonstruiert. Vor den Versuchen am realen Roboter wird diese Methode zuerst mit einem Simulationsmodell getestet. Schließlich wird das Programm in einer Matlab/Simulink-Umgebung implementiert, das die Antriebe des Roboters in Echtzeit steuert. Durch diese Arbeit wurde die Möglichkeit des 3D-Druckens auf einer gekrümmten Fläche geschaffen. Die notwendigen Programme, um eine Oberfläche abzutasten und danach darauf zu drucken, wurden erfolgreich auf einem realen Roboter implementiert. Zum Schluss werden Betrachtungen über die Genauigkeit der Bahnbewegung gemacht.

Contents

Chapter 1

Introduction

In this chapter an overview about the 3D printing technology is included and it explains the problem which 3D printing faces under the new applications requirements. And it also presents the proposal of this thesis to fulfill the applications requirements.

1.1 3D Printing Technology

1.1.1 Development of the Technology

The 3D printing, also known as additive layer manufacturing, is nowadays a very popular technology. It was developed in 1980s by Chuck Hull, he is also known as the inventor of 3D printing technology. In 1984 Chuck Hull invented a process known as stereolithography, the first commercial rapid prototyping technology. And he also developed the STL file format[1], which is widely accepted by 3D printing software. At first this technology was used to cure the photopolymers. With the development of this technology, today it is already applied to variety of materials and in many practical fields, such as architecture, construction, industrial design, automotive industry, aerospace, military, engineering, and even biological technology (human tissue replacement).

1.1.2 Description of the Process

3D Printing is a kind of rapid prototyping technology. It's a manufacturing process for the rapid production of three dimensional object directly from computer models. A solid object is built on a layer-by-layer basis, through a series of cross-sectional slices. The roughly description of this process is as below [1].

- A computer model of the desired object is created, and a slicing algorithm draws detailed information for every layer.

[1] STL (STereoLithography) is a file format native to the stereolithography CAD software created by 3D Systems.

- Using a technology similar to ink-jet printing, for printing each layer, a thin distribution of material powder (or liquid sometimes) spread over a basis surface. Then a selective application of a binder material is used to harden and join each layer in the specified cross-section pattern.
- This layer-by-layer process repeats until the object is completed. Following a heat treatment, unbound powder is removed, leaving the fabricated object.

1.1.3 Characteristics of 3D Printing

Advantages According to the characteristics of this manufacturing process, it can produce a object almost with any arbitrary desired shape. Compared with the other rapid prototyping technologies 3D printing has the following advantages [2].

- It can economically build custom products in small quantities. No need for costly tools or milling or sanding equipments.
- It has the ability to recycle waste material.
- 3D printer can seamlessly integrate with computer assisted design (CAD) software and other digital files. It's easily to share the design.
- It's possible more rapid and easier to design and modify the products.

Current limitations This technology also has some current limitations [2], such as:

- relatively higher costs for large production compared with other technologies,
- relatively few choice for materials, colors and surface finishes,
- relatively lower precision compared to other technologies,
- limited strength and resistance to heat and moisture, and color stability.

1.1.4 3D Printer

An industry grade 3D printer is capable to be applied to more diversity of materials, such as metal or biological material, but in general it's also very expensive. Nowadays more and more companies are developing affordable desktop 3D printer for laboratory using and even the home consumer, the so-called consumer grade 3D printer. A consumer grade 3D printer generally consists of following parts [3]:

frame ... holds all printer parts together, and gives the printer its stiffness;

constructions of x-axis, y-axis and z-axis ... are driven by motors and move together to enable the printing nozzle to move in the workspace;

print plate ... provides a working plane for printing process;

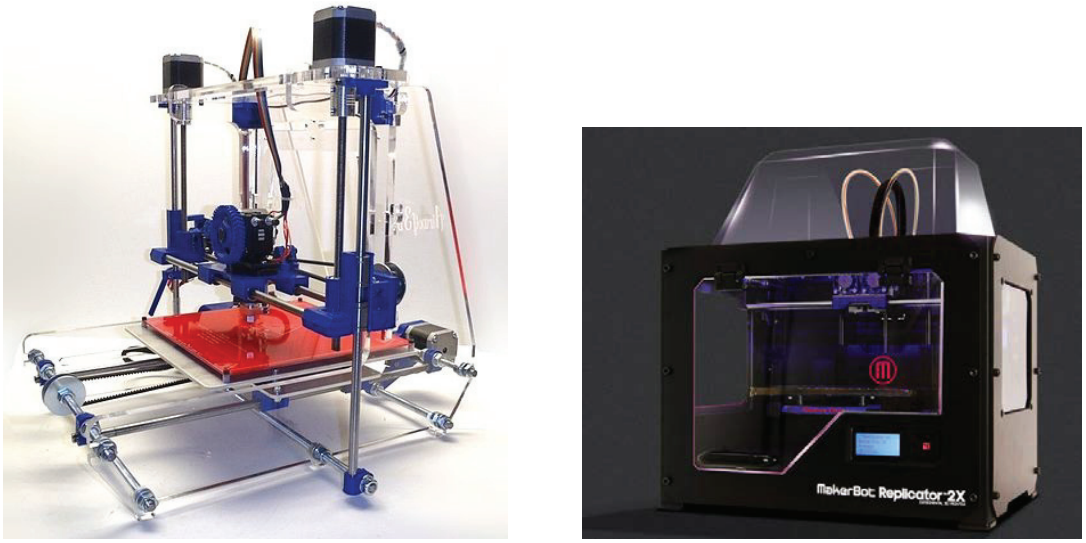
extruder ... consists of two parts, a cold top part that feeds the plastic filament and a hot bottom part that melts and extrudes the plastic;

electronics board ... controls the printing process;

stepper motors and motor controllers ... drive three axes and extruder;

end stop ... is the homing position for the extruder.

In Figure (1.1) are two typical consumer grade 3D printers seen.



(a) Airwolf 3D AW3D v.4 (b) MakerBot Replicator 2X

Figure 1.1: Consumer grade 3D printers

Through the introduction above it can be seen, that the workspace of the desktop 3D printers is generally restricted by the size of frame and the moving pattern of the extruder.

1.2 New Requirements and Problems

With the expansion of the scope of applications, 3D printing is facing some new requirements. For example, in a prototype production for synthetic substance, the 3D printer is expected to achieve a curved surface printing. Out of consideration of the material strength and the function, this object is expected to be directly printed on a working surface with the same curvature as the prototype.

Through the introduction above, it's obvious that those desktop 3D printers support only the translation movement of the extruder along the printing plane. For a curved surface printing, it can only fulfill with the additive of sequential printed two-dimensional layers.

1.3 New Concept

For a 6 DOF industrial robot, it's capable to accomplish almost any curve movement in its workspace. And its workspace is significantly larger than a desktop 3D printer. With the decrease of the cost for an industrial manipulator, it's already affordable for a laboratory or small company. Therefore a new concept is proposed, to combine the industrial robot with 3D printing extruder.

The proposal is so, that a printing block is mounted on the robot's end effector. And this block can fulfill the function of an extruder. Such a combination makes full use of the agility of a 6 DOF robot. So it can expand the working range of the 3D printer and endue the 3D printer more application possibilities. In the next several chapters the attempt to implement this concept will be detailed described.

Chapter 2

Experimental Environment and Design Procedure

This chapter introduces generally the experimental environment. This environment contains the manipulator Stäubli RX60, a 3D printer block and the software application for simulation and controlling the manipulator. In addition the design of experimental procedure is generally described.

2.1 Robot Stäubli RX60

In the practical experiment of this thesis the Robot Stäubli RX60 is used. The RX60 is a medium payload robot from the manufacturer **Stäubli**. It features a highly precise articulated arm with 6 DOF (degrees of freedom) for optimum flexibility, see Figure (2.1). And a spherical work envelope allows maximum utilization of the workspace. Compared with a normal desktop 3D printer the workspace of this robot is obviously larger. This feature of the robot ensure that 3D printing can be complied in a more flexible working rang. The technical data of robot RX60 are shown as Table (2.1). More detailed size and accuracy explanation see appendix.

Axes	1	2	3	4	5	6
Working range (°)	320	255	269	540	230	540
Working range division (°)	A +/-160	B +/-127,5	C +/-134,5	D +/-270	E +120,5,-109,5	F +/-270
Nominal velocity ($^\circ/s$)	287	287	319	410	320	700
Angular resolution ($^\circ \cdot 10^{-3}$)	0,724	0,724	0,806	1,177	0,879	2,747
Maximum linear speed (m/s)	1.8					

Table 2.1: Technical data of RX 60

Figure 2.1: Image of the Stäubli RX60

2.2 3D Printer Block

For this experiment a specific designed tool block, 3D printer block, is built. This tool block consists of a printer holding frame, an extruder and a laser distance sensor. It is mounted on the end effector of the robot. Figure (2.2) shows a schematic sketch about the mounting position of the block.

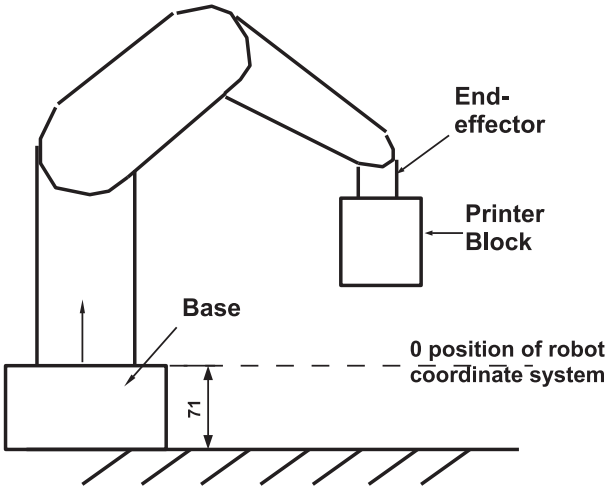


Figure 2.2: Schematic drawing of the mounting position of 3D printer block

The Extruder contains these components, a material heating device, a cooling ventilator, a material feeder as well as the printer nozzle. And they are assembled on the printer holding frame, see Figure (2.3). In addition, the laser sensor is mounted on the other side of the frame. And the Figure (2.4) shows a top view of the assembled 3D printer block.

2.3 Laser Distance Sensor

As we mentioned before, a laser distance sensor is also mounted on the printer holding frame, see Figure (2.5). It is used to scan a surface model, and the scanning process will be introduced in chapter 4. This laser distance sensor is from the manufacturer **Mel Microelectronic Ltd.**, with model **'M5L/10'**. It has the following characteristics:

- Big invisible light spot,
- consistent measuring values,
- and qualified for textiles with consistent colors.

The technical data of this sensor are as shown in Table (2.2).

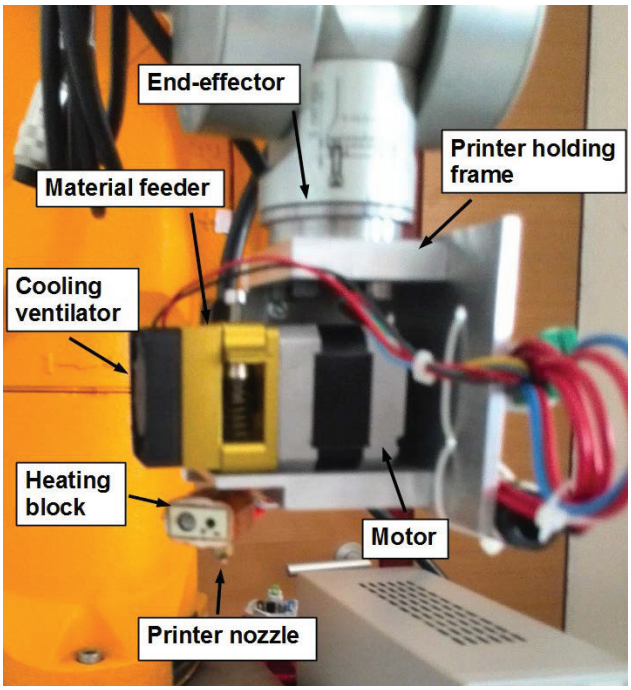


Figure 2.3: Components of the 3D printer block

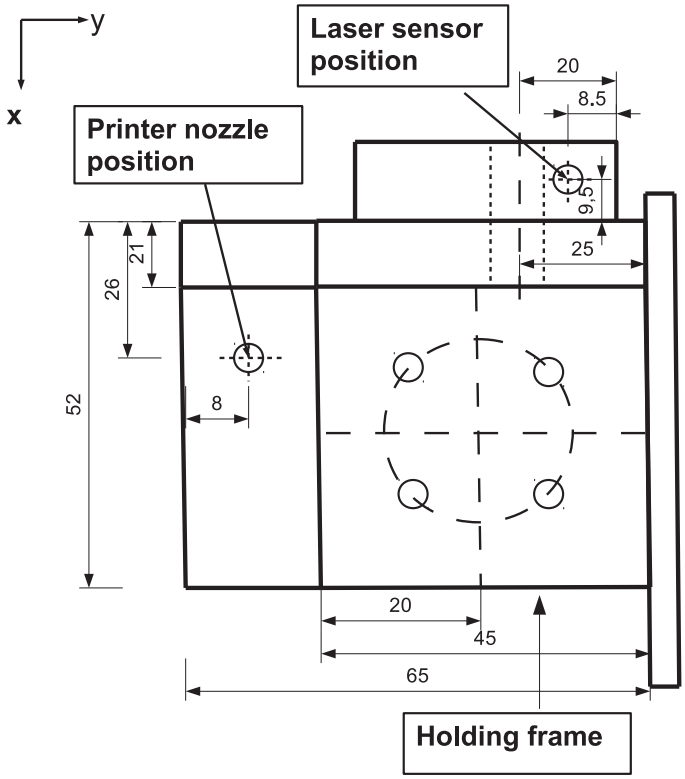


Figure 2.4: Top view of the printer block

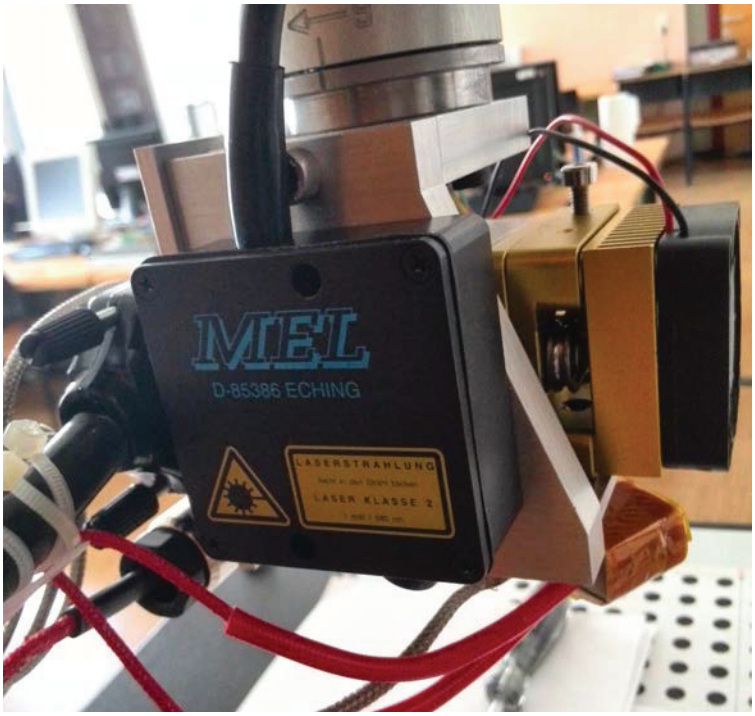


Figure 2.5: Laser sensor mounted on end effector

Type	Measuring Range (mm)	Reference Distance (mm)	Linearity Error (μm)	Resolution (μm)	Light Spot Diameter (mm)
M5L/10	$\pm$5	45	30	3	0,6

Table 2.2: Technical data of laser sensor

2.4 Manipulator Control

In this experiment several means are available to control the manipulator. The manual control with control panel and two software applications, B&R robotic project and Matlab Simulink.

2.4.1 Control Panel

The manufacturer of Stäubli RX60 robot provides a control panel. Via this control panel the robot can be intuitively and continuously driven to any desired position in the defined workspace. But the manual operation can not ensure the accuracy of the reached position. And of course this control method is not programmable. But for some tasks such as homing position setting it's still usable and necessary.

2.4.2 B&R Robotic Project

The B&R (Bernecker & Rainer) Automation Studio is a very friendly software development environment, and is suitable for a variety of automation control projects. The B&R robotic project is a for robot control established project under this environment. It consists of two main parts. The first part is the Motion-Handling block, which is installed on a industrial computer. It deals with the path planning and controls the movement of the six axes of the manipulator. The second part is the Human-Machine-Interface block, and it runs on a PC (personal computer). It provides the user a graphical control interface, and also provides user a simulation environment with a visualized robot. This simulation enabled a pre-test of the robot control software before it is implemented on the real robot. Figure (2.6) shows the software interface of the simulation environment of the robotic project.

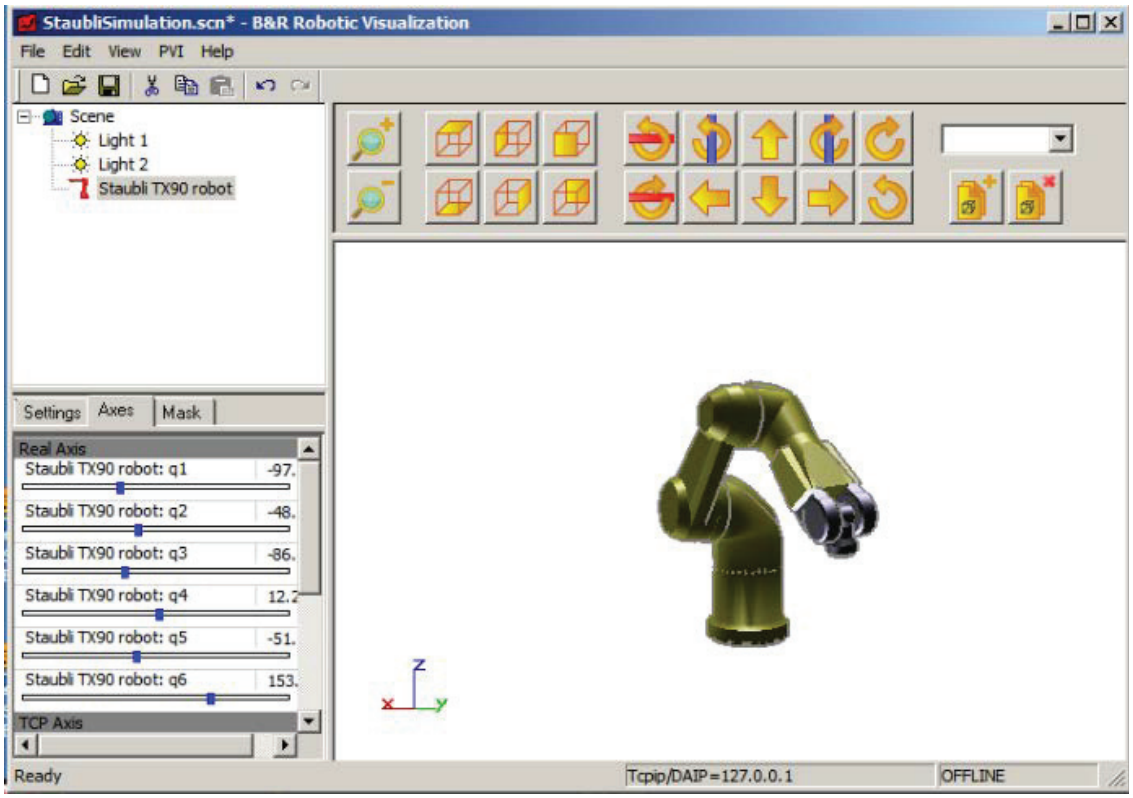


Figure 2.6: B&R robot project environment

2.4.3 G-code Programming

G-code, which has many variants, is the common name for the most widely used numerical control (NC) programming language. It is used mainly in computer-aided manufacturing for controlling automated machine tools. The G-code programming language has some specific commands and grammars, these will not be introduced in this thesis.

Using the G-code file is a programmable controlling method. That means, we can select several

points, use the G-code programming language to describe the absolute coordinates of these points or describe the relative position between each two points. All these codes are saved in one file, then the file presents a specific motion path for the manipulator. This procedure is so-called path planning. The path planning file can be imported through the B&R robotic project to drive the robot to achieve a movement along the preset path.

2.4.4 MATLAB®, Simulink® Programming

Using **MATLAB**® function is another programmable control method. **Simulink**® is a very useful product family from the **MathWorks** company. It's a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems.

Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with Matlab, enabling the user to incorporate Matlab algorithms into models and export simulation results to Matlab for further analysis [4]. For us it's a more friendly programming interface. Furthermore Simulink provides us a rich choices of the Control Toolbox, that enable us through a Simulink function to control, configure, and transfer data with the real manipulator over UDP(User Datagram Protocol).

In the experiment a Simulink simulation program is built, seen in Figure (2.7). The program consists of four blocks: the space mouse control block, the path planing block, the forward and inverse kinematics block and a 3D virtual reality environment. The space mouse control complies a similar function as the robot's control panel. It enables a intuitively and continuously drive of the robot. The path planing block accepts importing a preset motion path file for the robot. And this block processes these coordinates into usable translation and orientation vectors and delivers further to kinematics block. In the 3D virtual environment a virtual robot simulated Stäubli RX60 is assembled. Also a tool block and a surface object are in the virtual environment included. This 3D virtual environment provides us the possibility to observe the robot's motion in a pre-test. More description about the establishing of the 3D virtual reality environment refers to Li's thesis [5].

2.5 Experimental Design Procedure

With those hardware and software environment a experimental procedure is designed. This procedure contains the following steps:

1. surface scanning,
2. surface rebuilding,
3. programming,
4. simulation in Simulink,
5. simulation in B&R robotic project,

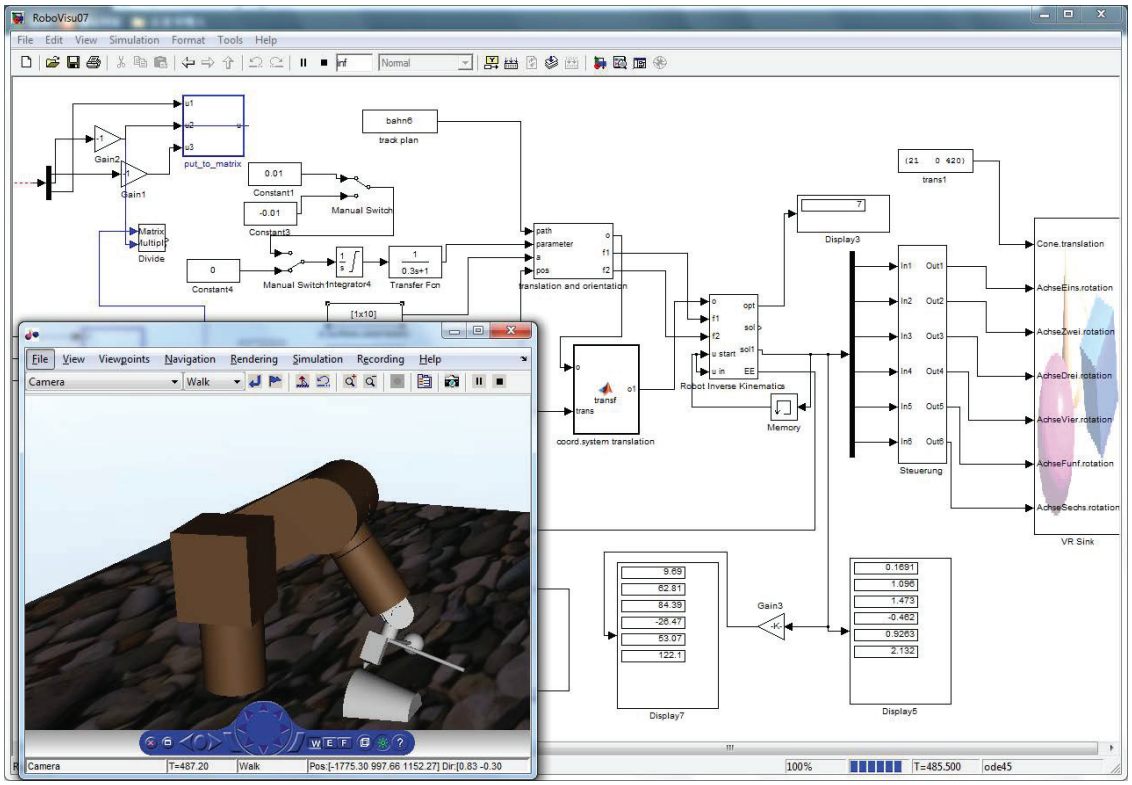


Figure 2.7: Simulink simulation program

6. robot operation.

At first the laser distance sensor is used to scan a surface object. Then with the measurement data an approximated surface will be by a mathematical tool in Matlab function rebuilt. By third step the translation and orientation vectors setting for inverse kinematics will be programmed with Matlab function. Then the algorithm will be verified in Simulink simulation environment. In addition a simulation in B&R environment will be implemented. Finally the operation on real robot will be executed.

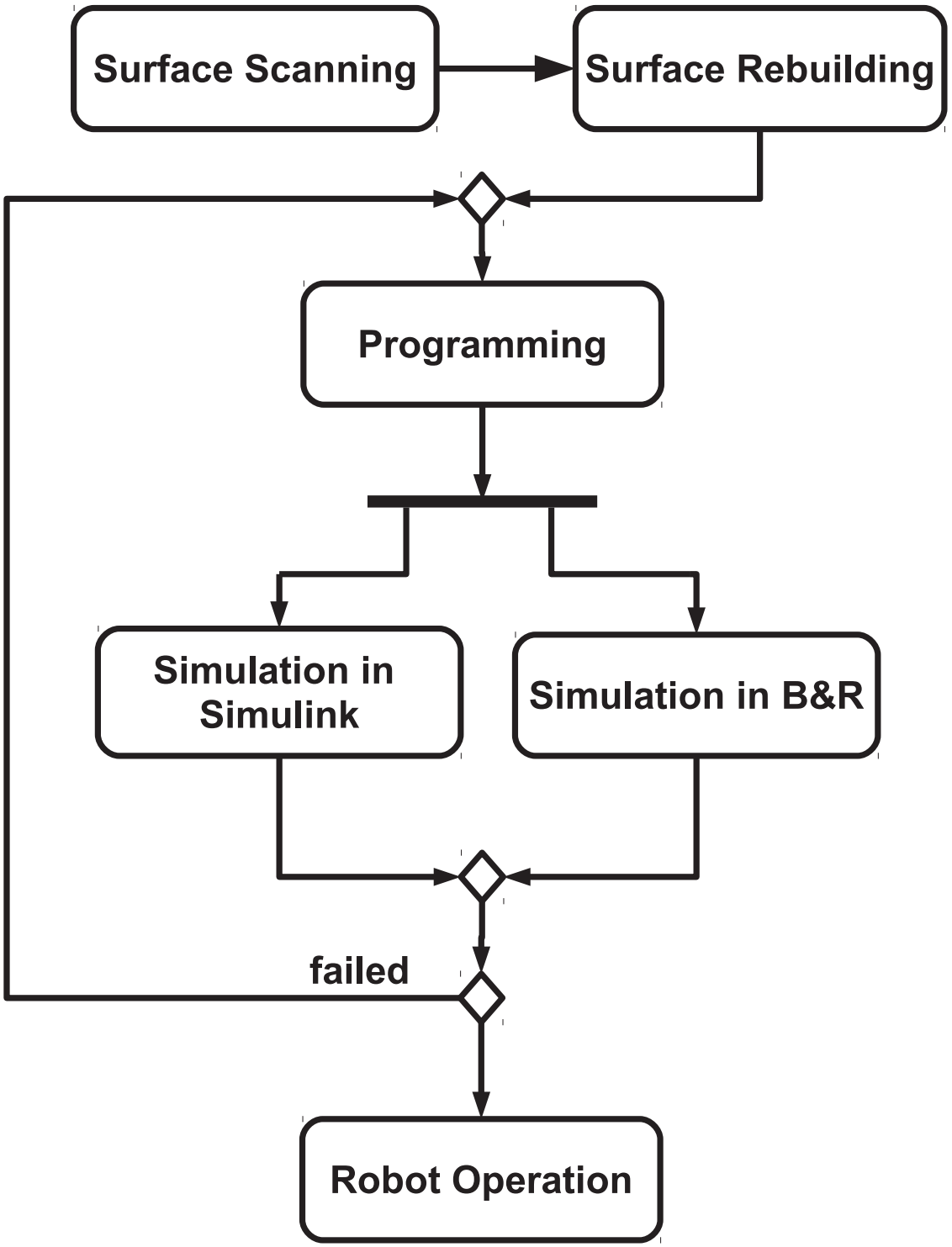


Figure 2.8: Experiment workflow

Chapter 3

Robot Kinematic Model

The purpose of this chapter is to introduce the widely applied robot kinematic modeling method, Denavit-Hartenberg (D-H) Model. And the Denavit-Hartenberg parameters setting based on the manipulator Stäubli RX60 will be also introduced. In addition a brief introduction about Hayati Model is concluded. Furthermore the forward and inverse kinematics concepts will be lead into this thesis. The robot kinematic model is a important foundation for the whole experiment.

3.1 Denavit-Hartenberg Modeling

A serial manipulator usually consists of a group of rigid bodies (are also typically known as links), which are connected together by joints. In most industrial robots, each link is connected to two other members. Therefore, each link has two joints. The Denavit-Hartenberg Modeling is widely used to describe the kinematic structure of the link in terms of physical link parameters [6]. A figurative description may refer to Figure (3.1). The following conventions indicate how to determine the D-H parameters:

a_i ... link's length, the length of the common normal of the two joints

α_i ... twist angle between joint i and joint $i+1$

d_i ... distance between link $i-1$ and link i along the common joint i

θ_i ... rotation angle of link i about joint i

We can tell, that among the four D-H parameters $(a_i, \alpha_i, d_i, \theta_i)$, two parameters (a_i, α_i) describe the link shape, and the other two (d_i, θ_i) describe the relative positions of two neighboring links.

Of cause by determining the D-H parameters it also have to be taken into account, that the establishment of each coordinate system Σ_i on each joint i. There are a set of rules about locating the origin O_i for Σ_i and choosing the axes x_i, y_i ,z_i. These rules are not discussed here, the detailed description please see the reference [6–8].

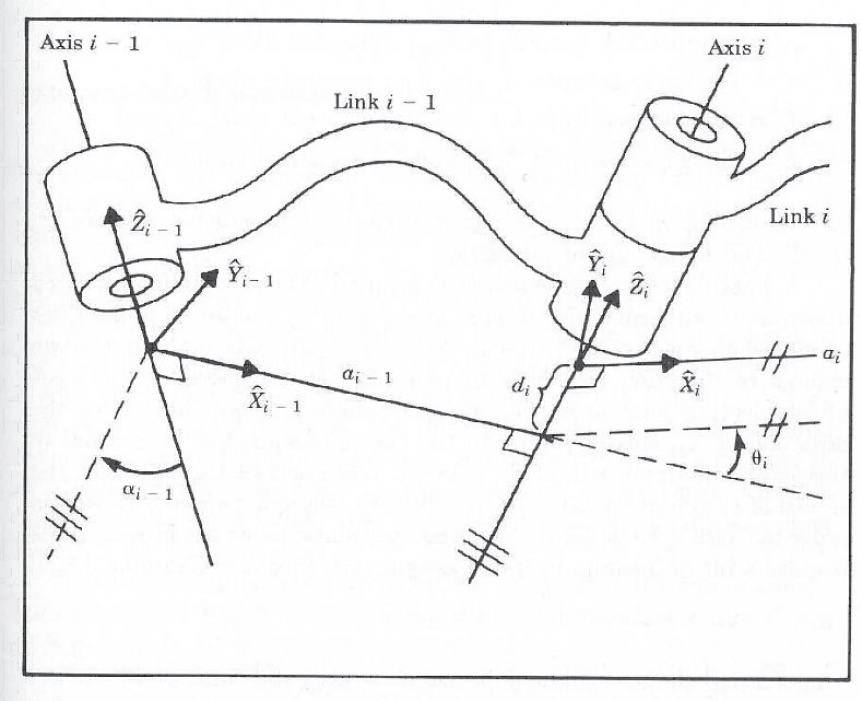


Figure 3.1: D-H modeling convention

The D-H Modeling convention of a link can be considered as a transformation A_i, which presents the transformation from coordinate system Σ_i to Σ_{i+1} in terms of the four parameters. That means:

$$A_i = Rot(z, \theta_i)Trans(0, 0, d_i)Trans(a_i, 0, 0)Rot(x, \alpha_i), \tag{3.1}$$

We can write this equation in a matrix form:

$$\mathbf{A_i} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{R_z}(\theta_i) \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{d_i} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{a_i} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{R_x}(\alpha_i) \end{bmatrix}. \tag{3.2}$$

Where $\mathbf{R_z}$ means the rotation matrix about axis z, and $\mathbf{R_x}$ means the rotation matrix about axis x [7]. If we sit the rotation matrix in this equation, then we get:

$$\mathbf{A_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_i) & -\sin(\theta_i) & 0 \\ 0 & \sin(\theta_i) & \cos(\theta_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_i & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha_i) & -\sin(\alpha_i) \\ d_i & 0 & \sin(\alpha_i) & \cos(\alpha_i) \end{bmatrix}. \tag{3.3}$$

For a robot with n joints, there are n transformation matrices from $\mathbf{A_1}$ to $\mathbf{A_n}$.

3.2 D-H Parameters Based on Stäubli RX60

Robot **Stäubli RX60** was used in the experiment. A roughly representation about this manipulator has been in chapter 2 included. Following the D-H convention the local coordinates systems are

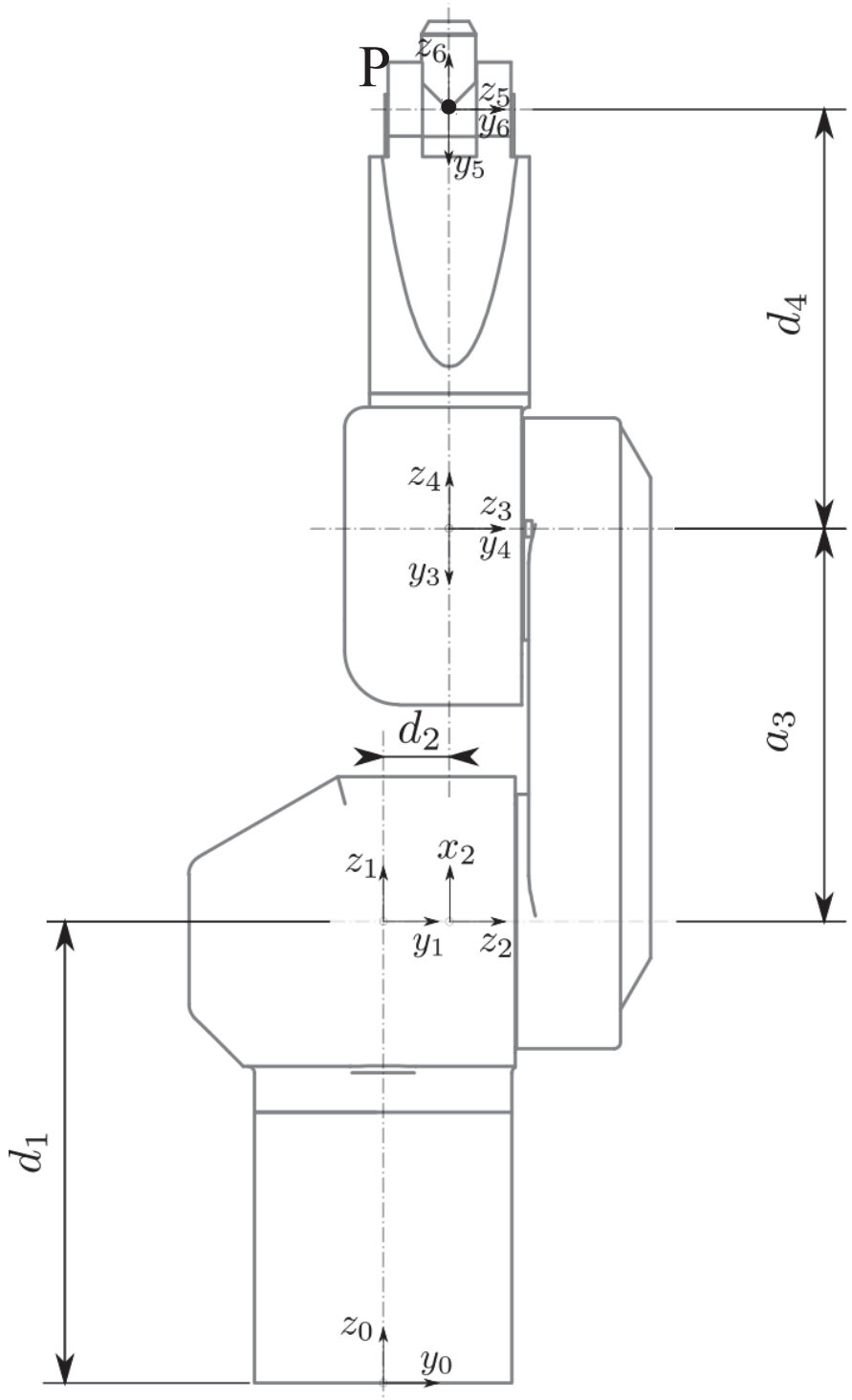


Figure 3.2: Robot Stäubli RX60

Joints	a_i	α_i	d_i	θ_i
1	0	0	d_1	θ_1
2	0	-90	d_2	θ_2
3	a_3	0	0	θ_3
4	0	90	d_4	θ_4
5	0	-90	0	θ_5
6	0	90	0	θ_6

Table 3.1: D-H parameters for Stäubli RX60

for each joint set, then we get the D-H parameters for robot Stäubli RX60 shown as Table (3.1). A figurative presentation is shown in Figure (3.2).

More specific explanation about setting these parameters is informed in Kollment's Thesis [8].

3.3 Hayati's Modified D-H Modeling

Many alternative kinematic models or modified forms of D-H model have been proposed. For example Hayati has proposed a modified D-H model. As in his work introduced [9], most of the current industrial or research manipulators consistent at least one set of consecutive parallel joints. By Stäubli RX60 the joint 2 and joint 3 are two consecutive parallel joints. When two joints are parallel or near parallel, following the D-H convention we have $\theta_i = 0$, $d_i = 0$, $a_i = L$, $\alpha_i = 0$. Assume that due to manufacturing tolerances the joint z_i is misaligned by a small angle β as shown in Figure (3.3). Now the two joints are regarded as two axes with an intersection at a very far away point. Therefor the D-H parameters can be totally different. The small error in the alignment of joint z_i causes a large error in D-H parameters.

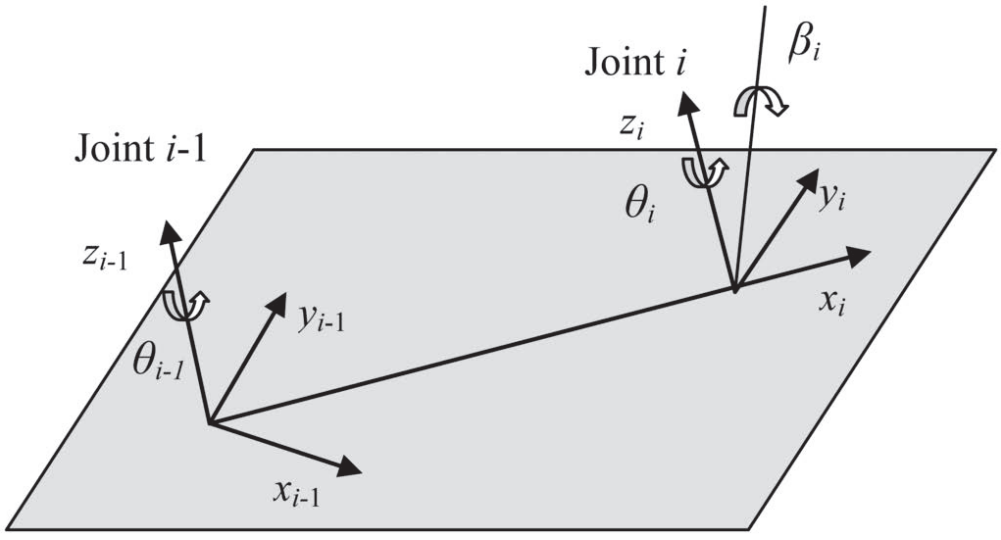


Figure 3.3: Schematic drawing of two consecutive parallel joints

So Hayati introduced us in his work a modified D-H model to overcome this problem [9]. He put

the angle β in as the fifth parameter, when the consecutive parallel joints situation exists. Through the following four steps he obtained the link coordinate frame.

1. Pass a plane perpendicular to the axis z_{i-1}. The intersection of this plane with joint i is O_i.
2. Rotate the frame $i-1$ about z_{i-1} to align the axis x_{i-1} with the line connecting the points O_{i-1} and O_i.
3. Translate the origin of the last frame to O_i.
4. Perform two rotations about the resulting frame's x and y axes to align its z axis with that of joint i

And he got the transformation as Equation (3.4),

$$A_i = Rot(z, \theta_i)Trans(x', d_i)Rot(x'', \alpha_i)Rot(y'', \beta). \tag{3.4}$$

By the other joints, they maintain to follow the D-H convention. So we call this Hayati's modeling as a modified D-H modeling. In Hayati's article, it was proved that this modeling method improved the absolute positioning accuracy of serial robots. In spite of the advantage of Hayati's modeling, still D-H modeling was used as the kinematic model in this thesis.

3.4 Forward and Inverse Kinematics

3.4.1 Forward Kinematics

Forward kinematic, also known as direct kinematic, is a mathematical approach to calculate the position of the end effector through giving the D-H parameters in. For Stäubli RX60, the robot with six joints, the complete transformation matrix from robot's base coordinates system to the end effector's coordinates system should be like Equation (3.5) built,

$$\mathbf{T} = \mathbf{A_1 A_2 A_3 A_4 A_5 A_6}, \tag{3.5}$$

matrix $\mathbf{A_i}$ is the transformation matrix from coordinate system Σ_i to Σ_{i+1} in Equation (3.3) introduced. We set the robot's base coordinates system as frame Σ_0, and the end effector's coordinates system as frame Σ_6. When the global coordinates $\mathbf{X}$ of the origin of Σ_0 is given, then we can through the transformation matrix get the global coordinates $\mathbf{X}^*$ of the end effector as,

$$\mathbf{X}^* = \mathbf{T} \cdot \mathbf{X}. \tag{3.6}$$

From the equation above we can tell, that the forward kinematic is easier to implement. But it make just little significance for the practical applications. Because it's a large work to calculate every rotation angle for each movement.

3.4.2 Inverse Kinematics

In most cases, we have only the desired coordinates of the end effector, so a mathematical method is needed to inversely calculate the possible rotation angles for six axes. That is the so-called inverse kinematics.

Because transformation matrix for the forward kinematics exists already, it's easy to understand, that it exists for sure also an algebraic approach to fulfill the inverse kinematics. This algebraic approach will be accomplished through splitting the transformation matrix and trigonometric operations. We will not discuss further about this method, since we didn't use it in the experiment. The detailed derivation and explanation see the reference [8, 10].

In the experiment a geometric approach was used to solve this inverse kinematics problem. As it is previously introduced, the robot Stäubli RX60 consists of six joints. For each movement to reach a particular spatial position, it needs the series rotations of six axes. And the whole movement of the manipulator could be also split into two sub-motions:

translation ... is the movement which is achieved through the rotation of axes 1, 2 and 3;

orientation ... is the movement which is completed through the rotation of axes 4, 5 and 6.

The rotation of axes 1, 2 and 3 determines the global coordinates of the intersection of axes 4, 5, and 6. This intersection point is in Figure (3.2) as point **P** indicated. Conversely when the desired coordinates of **P** is given and it's within defined workspace, besides the geometric parameters of the robot arms are already known, the rotation angles for axes 1, 2 and 3 are by trigonometric functions solvable. Because we can see this motion as taking point **P** from the origin to the desired position, so it is called as translation.

Joints 4, 5 and 6 intersect by point **P**, this configuration imitates a human's wrist action. The combination of rotation of these tree joints decides in which direction the end effector points. Due to this feature this motion is called as orientation. The inverse kinematics solving is similar to it by sub-motion of translation. A desired pointing direction for end effector should be provided, it's determined by the task for robot. And this direction can be presented as a vector. With this vector, as long as this configuration is physically possible, then the rotation angles are solvable.

Specific introduction about the solving procedures see Kollment's thesis [8]. And the calculating about the coordinates of point **P** and the direction vector will be in chapter 5 specifically discussed.

Chapter 4

Surface Scanning and Reconstruction

In this chapter it will be detailed described, how the essential data are from an observed surface object obtained in the experiment. And a useful mathematic method, Levenberg-Marquardt method, is introduced, with the mathematical tool which is provided by Matlab an approximated surface is practically reconstructed.

4.1 Surface Scanning With a Laser Distance Sensor

For describing a certain surface in the global coordinate system, first of all we need the coordinates of several points on this surface. To obtain these coordinates, a laser distance sensor is used, which is already in chapter 2 introduced. By using the space mouse we can drive the robot roughly to any chosen points. And the laser spot can be also used to indicate those selected points.

For each point a set of x, y and z-coordinates is read from the manipulator's controller. But they are just the coordinates of the point **P**. Through the introduction in chapter 2, and referred to Figure (2.4), it can be confirmed that between the read coordinates and the coordinates of the laser spot exists offset. Therefore a coordinates conversion is needed. The x and y-coordinates of point **P** are set as (x, y), and the x and y-coordinates of laser spot are set as (x^*, y^*). Then the conversion is presented as below:

$$x^* = x - 45.5, \tag{4.1}$$

$$y^* = y + 11.5. \tag{4.2}$$

The sensor works not just as a perfect laser pointer, and it can provide the real z-coordinate of a surface point at the same time. The reference distance between the laser head and the indicated point is 45 mm. The sensor exports a voltage value, which can be read from a voltmeter. That shows the distance bias about ± 5 mm around 45 mm, see Figure (4.1). Through this voltmeter reading we get the real distance from laser apparatus to the measured surface. Combination of this value and the z-coordinate read from controller, here means the z-coordinate of point **P**, presents the real z-coordinate of laser spot. The real z-coordinate conversion is presented in Equation (4.3),

$$z^* = z - 57 - (45 - V \cdot (-5/10)),^{1} \tag{4.3}$$

[1] This equation presents the translation of z-coordinate, based on Figure (4.1) and Figure (2.2)

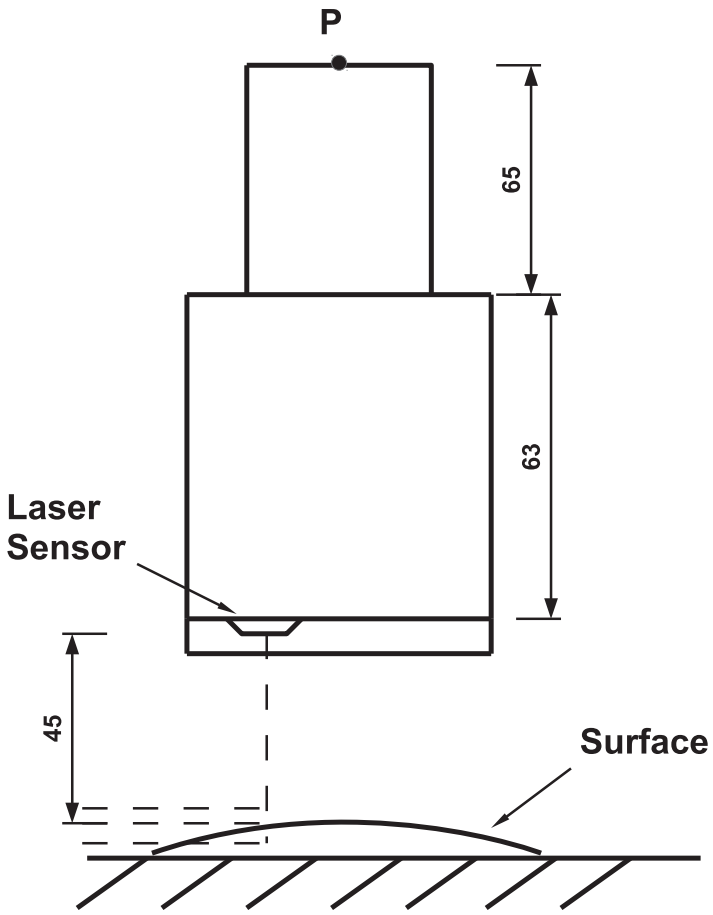


Figure 4.1: Schematic drawing of the position of laser sensor

where V means voltage value. The z^* presents the real z-coordinate. Since the surface model could not be a perfectly smooth surface, the variations of z-coordinates in experimental data are totally reasonable.

4.2 The Levenberg-Marquardt Method

With those measured coordinates from the surface, an approximated surface equation can be rebuilt with a certain mathematical method. First of all we take a quick look at *least squares problem*. The *least squares problems* arise when fitting a function to a set of measured data points by minimizing the sum of the squares of the errors between the data points and the function [11]. That means, given a fitting model $M(\mathbf{x}, t_i)$, and this model depends on the parameters $\mathbf{x} = [x_1, \ldots, x_n]$. For any choice of $\mathbf{x}$ we can compute the residuals,

$$f_i(\mathbf{x}) = y_i - M(\mathbf{x}, t_i)\,, \qquad i = 1, \ldots, m. \tag{4.4}$$

So the least squares problem can be defined as the following equation:

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{x}))^2, \tag{4.5}$$

where $m \geq n$, and n is the number of parameters. We assume that there exists an $\mathbf{x}^*$, it is the local minimizer for Equation (4.5). Find the $\mathbf{x}^*$, then the least squares problem is solved [12].

For a function of a linear line, this method is already sufficient, but for a nonlinear function, such as a quadric surface, we need some new methods. According to the introduction from Madsen et al, all methods for nonlinear optimization are iterative. That means, from a starting point x_0 the method produces a series of vectors $\mathbf{x_1}$, $\mathbf{x_2}$,..., which should converge to the minimizer $\mathbf{x}^*$ for the given function. For each iterate, one step from the previous one to the next, it consists two procedures:

1. Find a descent direction h_d, and
2. find a step length giving a good decrease in the function value.

Here it's not going to go into more details about the two procedures. The detailed explanation refer to the Madsen's article [12].

Madsen introduced us a useful nonlinear least squares problems method in his work, the *Levenberg-Marquardt method*. The step h_{lm} is defined by the following equation,

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})h_{lm} = -\mathbf{J}^T\mathbf{f}\,, \qquad \mu \geq 0, \tag{4.6}$$

here $\mathbf{f} = \mathbf{f}(\mathbf{x})$ is the given function and $\mathbf{J} = \mathbf{J}(\mathbf{x})$ is its Jacobian matrix. Furthermore μ is a damping parameter, and it should have the following effects:

1. For all $\mu > 0$, the coefficient matrix is positive definite, that ensures the h_{lm} is a descent direction.
2. For a large value of μ we get

$$h_{lm} \simeq -\frac{1}{\mu}\mathbf{J}^T\mathbf{f} = -\frac{1}{\mu}\mathbf{F}'(\mathbf{x}), \tag{4.7}$$

   it likes a *gradient descent method*, the sum of the squared errors is reduced by updating the parameters in the direction of the greatest reduction.
3. For a very small value of μ, then $h_{lm} \simeq h_{gn}$. It likes a *Gauss-newton method*. When $\mathbf{x}$ is close to $\mathbf{x}^*$, then we get locally quadratic final convergence [12].

Now we can tell that, the *Levenberg-Marquardt method* acts more like a *gradient-descent method* when the parameters are far from their optimal value, and acts more like the *Gauss-Newton method* when the parameters are close to their optimal value. It's actually a combination of the other two minimization methods [11]. More detailed description see the reference.

Fortunately the Matlab Software has contained the *Levenberg-Marquardt method* tool, the function *'lsqcurvefit'*. So we don't have to be anxious about finding the descent direction or step length. The function code is shown as below,

$$x = lsqcurvefit(FUN, x_0, XDATA, YDATA).$$

It starts at $\mathbf{x}_0$ and finds coefficients $\mathbf{x}$ to best fit the nonlinear functions in **FUN** to the data **YDATA**. Where **XDATA** are the variants of **FUN**. More specific explanation about the using of this function please see the Matlab help file. The following sections present how to achieve the surface fitting by using the *lsqcurvefit* function.

4.3 The Plane Fitting

Before doing a surface fitting, a plane fitting is a simple start to check the feasibility of algorithm and the scanning method, although it's not a nonlinear problem. A general plane is defined as following equation:

$$Ax + By + Cz + D = 0. \tag{4.8}$$

And it can be rewritten as this:

$$z = a_1 x + a_2 y + a_3, \tag{4.9}$$

where $a_1 = -\frac{A}{C}, \quad a_2 = -\frac{B}{C}, \quad a_3 = -\frac{D}{C}$. Then this function is set as a fitting model, and there are three parameters in this function. That means at least 4 sets of measurement data are needed for solving this equations system. In this experiment six points were picked, and we obtained six sets of coordinates. Measurement data are as shown in Table (4.1).

Points	1	2	3	4	5	6
x (mm)	488,6	447,0	411,0	411,0	449,3	487,9
y (mm)	57,39	57,39	57,4	20,44	20,44	20,44
z (mm)	114,2	114,2	110,5	110,5	110,5	110,5
voltage (v)	9,2	8,55	3,55	1,28	3,22	1,26

Table 4.1: Measuring data for a plane

By application of the function *lsqcurvefit* x and y-coordinates are set as **XDATA**. And as already known in Equation (4.3), the real z-coordinate of laser spot, z^*, is set as **YDATA**. The vector $a_0 = [1, 1, 10]$ is as starting value given in. And the fitting results are shown as below:

```
a =

    0.0055   -0.0033    5.1232
```

So the plane's equation as results of the fitting function is '$z = 0.0055x - 0.0033y + 5.1232$', and the objective function should be '$z = 0$'. From the comparison of the two equations above we can tell, that *lsqcurvefit function* does its job quite well. Matlab source codes are shown as below.

```
function [a] = planefit(coord)
  zv = coord(:,4);
  zz = coord(:,3)+71-(45 - zv*(-5/10))-63-65;
  % height correction for laser head

  x = [coord(:,1),coord(:,2)];

  f = @(a,x)a(3) + a(2)*x(:,2) + a(1)*x(:,1);
  ydata = zz;
  a0 = [1 1 10];
  [a,resnorm,residual,exitflag,output,lambda,jacobian]
    = lsqcurvefit(f,a0,x,ydata);
```

4.4 The Quadric Surface Fitting

The surface fitting should follow the same principle. The main point is how to define the surface equation and the parameters. According to the surface object which was used in the experiment, it is close to a cone form. A cone form can be classified as a quadric. In mathematics, a quadric, or quadric surface, is any n-dimensional hypersurface in (n+1)-dimensional space defined as the locus of zeros of a quadratic polynomial [13]. In coordinates $\{x_1, x_2, \ldots, x_{n+1}\}$, the general quadric is defined by the algebraic equation:

$$\sum_{i,j=1}^{D+1} Q_{ij} x_i x_j + \sum_{i,j=1}^{D+1} P_i x_i + R = 0. \tag{4.10}$$

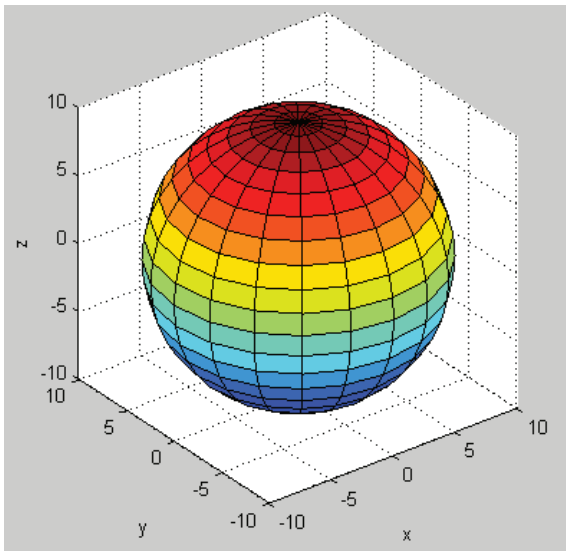
Then in the three-dimensional space the general equation of a quadric should be:

$$Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J = 0. \tag{4.11}$$

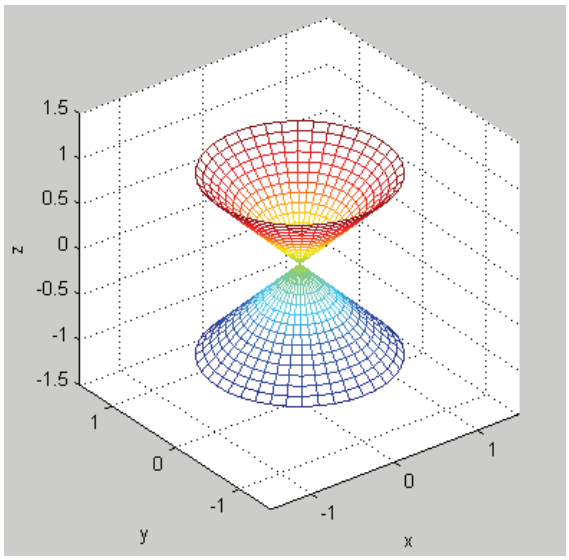
There are some normal forms of quadric in three-dimensional space shown as following:

- Sphere: $\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{a^2} = 1$;
- Circular Cone: $\frac{x^2}{a^2} + \frac{y^2}{a^2} - \frac{z^2}{b^2} = 0$;
- Circular Cylinder: $\frac{x^2}{a^2} + \frac{y^2}{a^2} = 1$;
- Hyperbolic Paraboloid: $\frac{x^2}{a^2} - \frac{y^2}{b^2} - z = 0$.

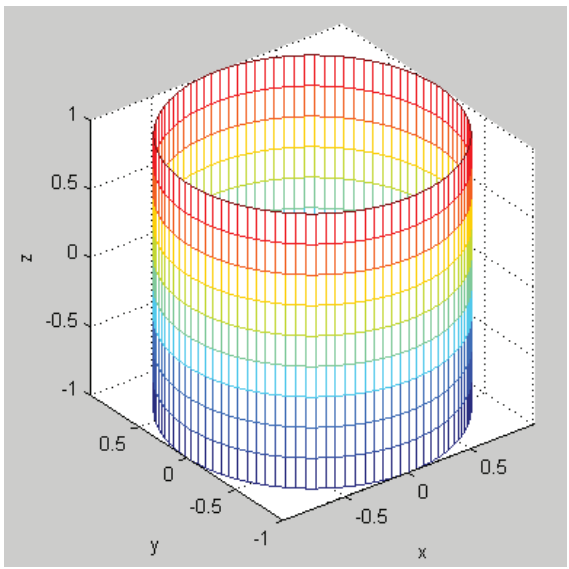
Figure (4.2) shows the Matalb output of these four quadric surfaces.



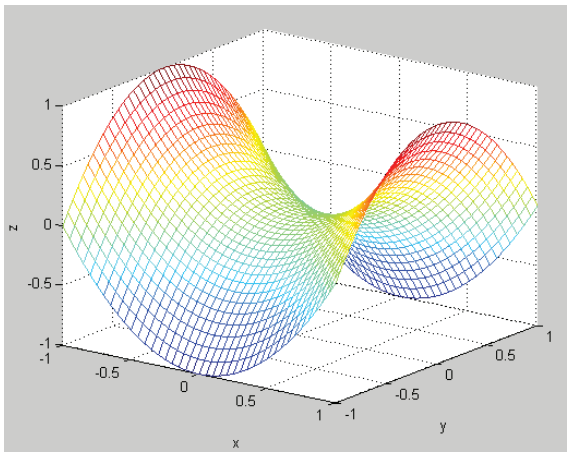
(a) Sphere



(b) Circular Cone



(c) Circular Cylinder



(d) Hyperbolic Paraboloid

Figure 4.2: Some normal forms of quadric in three-dimensional space

As introduced before, in this experiment the surface object to be scanned is close to a cone surface, see Figure (4.3). So the objective function is set on the basis of a circular cone. By measuring the dimensional characteristics of the object and its position in the robot coordinate system, the parameters of the objective function could be determined, and the function is presented in Equation (4.12). And Figure (4.4) demonstrates the simulation result of this circular cone in Matlab.

$$0,047x^2 - y^2 - z^2 - 70x + 130y + 22344 = 0 \tag{4.12}$$

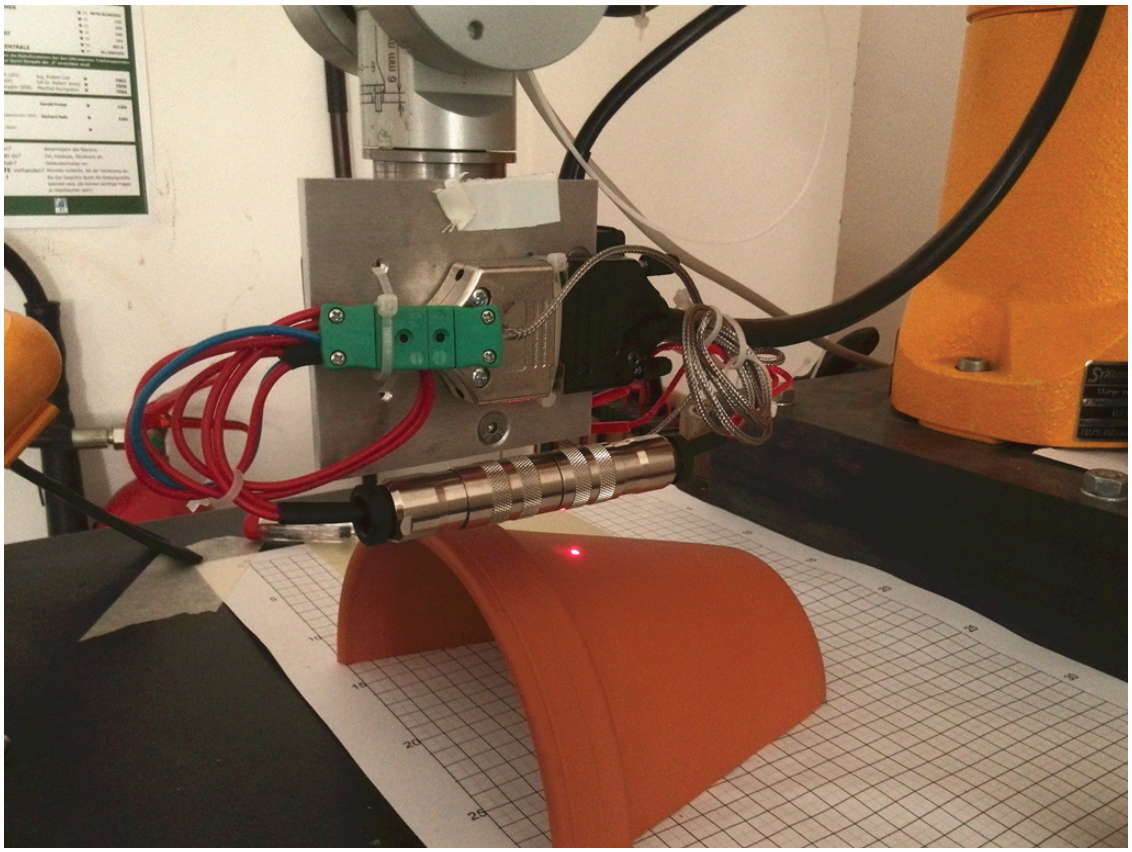


Figure 4.3: Scanning of a cone surface

Basically speaking, through a rewrite of this equation the nonlinear function for the *lsqcurvefit* can be already settled. But for a further application and a better versatility, a more general expression of z is needed. As previously mentioned, Equation (4.11) is the general expression of a quadric surface, so the z expression derived from this equation should fit for all quadric models. Here the quadratic formula is used to solve this problem. Equation (4.11) could be as following rewritten:

$$Cz^2 + (Ey + Fx + I)z + (Ax^2 + By^2 + Dxy + Gx + Hy + J) = 0. \tag{4.13}$$

If we set,
$a = C$,
$b = Ey + Fx + I$,
$c = Ax^2 + By^2 + Dxy + Gx + Hy + J$,
then according to quadratic formula we get the expression of z shown as below,

$$z = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{4.14}$$

We took Equation (4.14) as the nonlinear function for the *lsqcurvefit* and set appropriate starting values of the parameters based on Equation (4.12), the surface fitting can be done. Because there are total ten parameters to solve, a total of 25 points were picked from the surface object. These 25 points located in an area around 50 by 50 mm. The 25 sets of coordinates can ensure the equations be able to be solved. The measurement data are seen in Appendix.

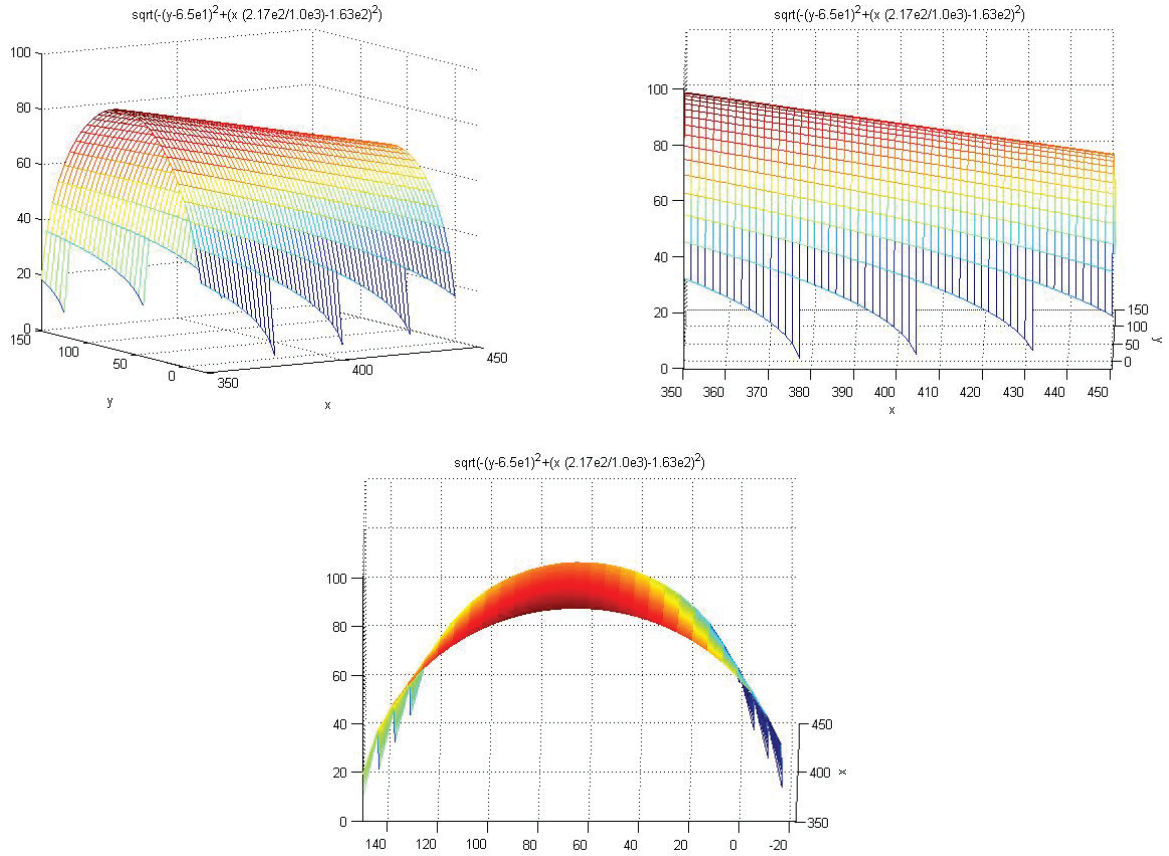


Figure 4.4: Simulation of the cone surface in Matlab

But in the practical experiment it's found out that determining ten parameters is still a large work for the Matlab function. The equation system is solvable, and the minimization is able to be found. But an approximation with ten parameters still leads to a big deviation from the ideal values. That reduces the precision of the rebuild quadric surface. Therefor the following two steps are taken to solve this problem:

1. The maximal iteration's number was modified from the default value '1000' to '1500'.
2. The DOF of the equation is restricted. Since it's already known that the desired surface equation is a circular cone, through comparing the circular cone's equation and the general quadric equation, four parameters of the ten should be zeros. So we can restrict these four parameters within a small range around '0'. This can be done by setting an upper and a lower boundary for the parameters in *lqscurvefit*'s option.

After these steps the fitted surface equation is already very near the objective function. The Matlab source code is seen below.

```
function [a,x,y,cov,J,sigma,resnorm,residual] = identQuadric2(coord)

v = coord(:,5); % Voltmeter reading from laser measure instrument

%z value calculating, bases on the original labor data 'dataSet'
y = coord(:,4)+70-(45 - v*(-5/10))-63-65;

x = [coord(:,2)-45.5, coord(:,3)+11.5];

% Starting value setting bases on the original cone ('dataSet')
% function: a*x^2 + b*y^2 + c*z^2 + g*x +h*y + j
a0 = [0.04  -1  -1  0 0 0  -75  100  0  2.7e4];

% surface fitting's boundary setting for the original cone
lb = [-2 -2 -5  -0.5 -0.5 -0.5  -105   100   -0.5  1e4];
ub = [ 2  0  0   0.5  0.5  0.5  -45    150   1.5   6e4];

% increasing the max evaluation's limit from
% default '1000' to '1500'
options = optimset('MaxFunEvals',1500);
[a,resnorm,residual,exitflag,output,lambda,jacobian]
 = lsqcurvefit(@fquadric, a0,x, y,lb,ub,options);
J = jacobian;
cov = inv(J'*J);
sigma = sqrt(diag(cov));
z = fquadric(a, x);

%rewriting the x,y,z data into the same size 5*5 matrics
```

```
xx = x(:,1);
X = reshape(xx,5,5);
yy = x(:,2);
Y = reshape(yy,5,5);
Z = reshape(z,5,5);

surf(X, Y, Z)

hold on
plot3(x(:,1), x(:,2), y, 'o' )

hold off
```

The Matlab operation's results are shown as following, and Figure (4.5) also shows the rebuilt surface. The surface fitting is done.

```
a=[0.044843283119378, -1.099931905262076, -1.061442712789104,
   0.038787388696698, -0.039591515413684, 0.035412367741600,
   -73.216117468512660, 1.200000060888018e+02, 0.021540558901372,
   2.329189141280282e+04]
```

It is proved by the experiment, that this method completes very well the requirements about rebuilding a desired quadric surface. And this method has certain versatility for the other forms of quadric surface. But by using this method, the choosing of starting value for *lsqcurvefit* should be cautious. Because the solving of z, that includes a square root operation, can be easily led to complex results by inappropriate variables setting. Those complex results make the *lsqcurvefit* no solution. So the starting value choosing should be not too far from the ideal value.

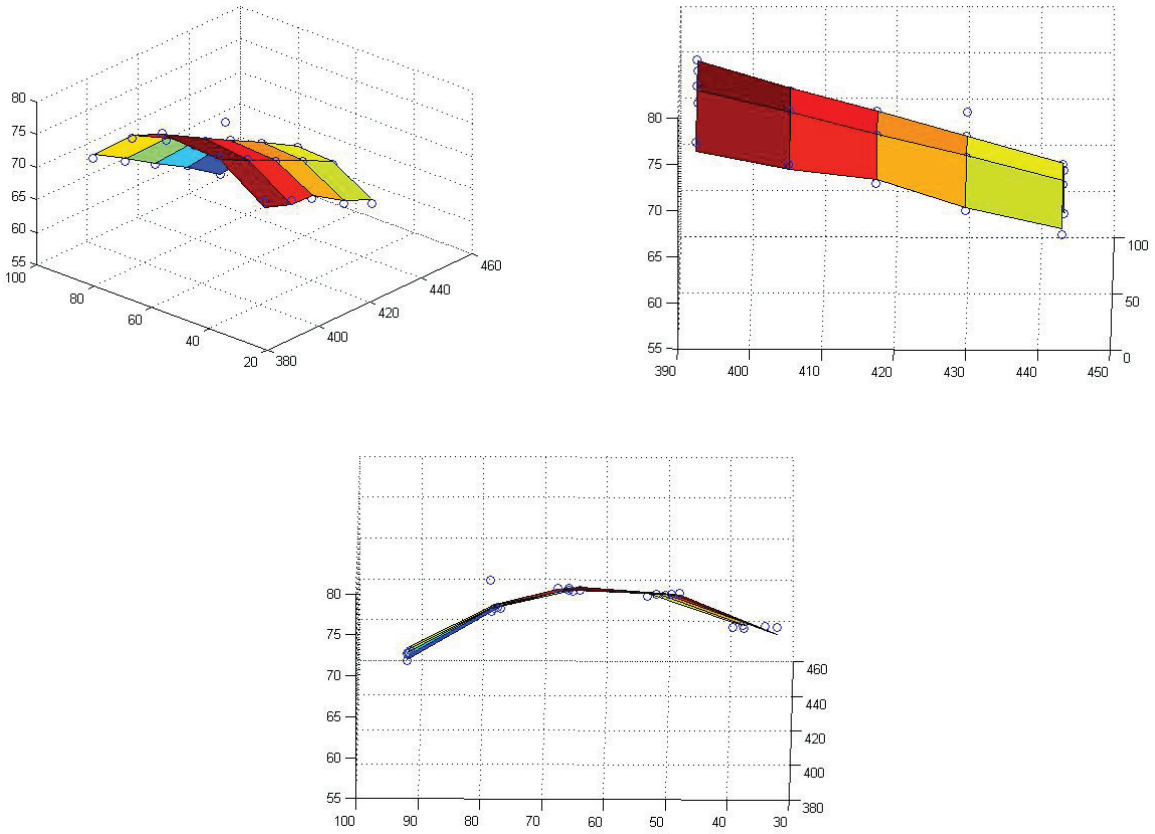


Figure 4.5: Surface fitting's output by Matlab

Chapter 5

Path Planning and Implementation

In this chapter the path planning concept of this thesis will be explained, and the algorithm of orientation and translation computing for the manipulator will be introduced. This algorithm will be implemented in a Simulink program.

5.1 The Purpose of Path Planning

Through the introduction in chapter 4, an approximated surface is already available for the experiment. If we attempt to print this rebuilt surface, we need the robot accomplish two function. One is to make sure the printer nozzle follow a motion path, which can accurately restitute the approximated surface. The other is to make sure the nozzle always point perpendicularly towards the surface, because that's the optimal approaching angle for the 3D printer nozzle.

At first the motion path of the nozzle should match those points coordinates on the fitted surface. Through the completed work in chapter 4, these coordinate are informed. As in chapter 3 already introduced, the inverse kinematics calculating is split into two procedures, translation and orientation. Then two end conditions are needed, the position of point **P** and a direction vector. In this thesis the path planning doesn't mean only the picking of path coordinates. It also considers the computing of apposite translation coordinates and orientation vector.

5.2 Implementing the Path Planning on Stäubli RX60

G-code Importing a G-code programming file is one method to implement the path planning for robot Stäubli RX60. G-code programming is already in chapter 2 introduced. Although the G-code programming language has already provided some functions to achieve a circle movement or a curve movement, or a movement along an arbitrary spline, the radius for the curve must be known in the first place. And it's difficult to suit a movement with the real-time changes in the curvature. So this method for the experiment is not practical.

Matlab function The another way to implement the path planning, as we in chapter 2 already introduced, is using a Simulink function.

In the Simulink program, which developed for the simulation, tree variables o, f_1 and f_2 are defined. The variable o delivers a vector containing the translation coordinates to the inverse kinematics block. The by inverse kinematics block calculated rotation angles are through UDP transferred back to the robot. With those angles the robot is driven to the desired position. How to get the value for o will be introduced in the later section.

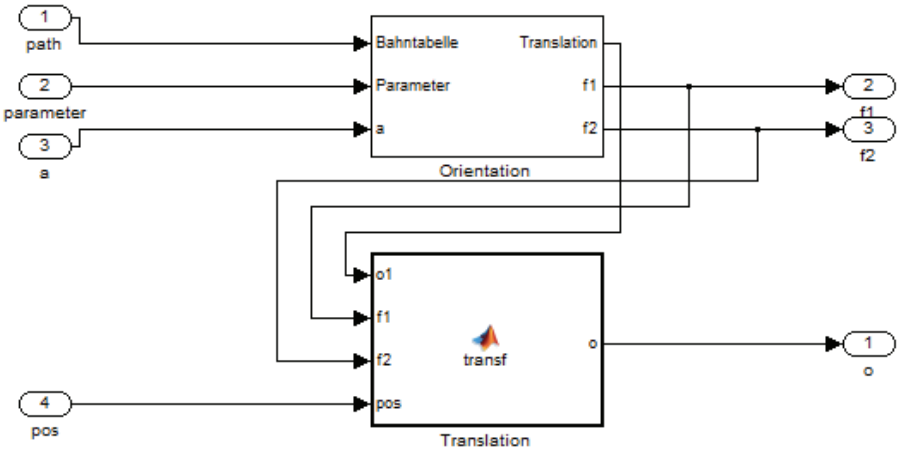


Figure 5.1: Computing the robot pose in Simulink program

The other two variables, f_1 and f_2, contain two vectors. Here we name the two vectors as $\vec{f_1}$ and $\vec{f_2}$. They are used to implement the orientation. In the following sections the detailed introduction about setting these variables will be presented.

5.3 Implementation of the Orientation Vector

According to the introduction about the geometric characteristics and mounting position of the 3D printer block in chapter 2, it's known that the nozzle's axis is parallel to the end effector's central axis. Therefor the orientation problem for printer nozzle is obviously the same problem for robot orientation.

We set a local coordinate system for the end effector, and name it as $\sum_6$, and $\vec{x}, \vec{y}, \vec{z}$ are three unit vectors for x, y and z-axes in $\sum_6$. In our Simulink function vector $\vec{f_1}$ is set to overlap on the unit vector $\vec{x}$. That means they point the same direction, but $\vec{f_1}$ can be a scalar vector of $\vec{x}$. And similarly the vector $\vec{f_2}$ overlaps on unit vector $\vec{y}$, it points the same direction as $\vec{y}$ with a scalar size.

So according to the definition above, if we set $\vec{f_3} = \vec{f_1} \times \vec{f_2}$, then we can get, that the vector $\vec{f_3}$ should point the same direction as $\vec{z}$ and with a scalar size, schematic figure is shown as following.

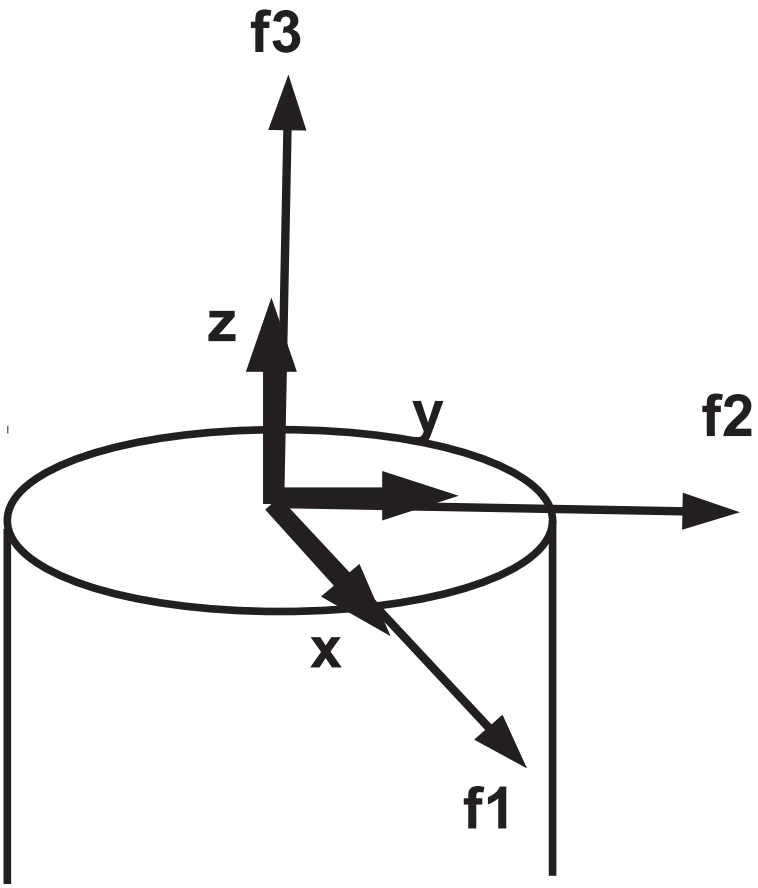


Figure 5.2: Schematic drawing of vectors $\vec{f_1}$, $\vec{f_2}$ and $\vec{f_3}$

Theoretically speaking, when the vectors $\vec{f_1}$ and $\vec{f_2}$ are setting up, the orientation of the robot end effector is done. Obviously through setting up appropriate $\vec{f_1}$ and $\vec{f_2}$ can let $\vec{f_3}$ point the inverse direction as the surface normal. This setting makes the nozzle vertical to the surface. In the next section it will be detailed introduced, how to find the surface normal.

5.4 Computing the Normal of the Surface

In the three-dimensional case, a surface normal to the surface at a point Q, is a vector that is perpendicular to the tangent plane to that surface at Q. Such a tangent plane can be determined by two tangent lines, because of the geometric theorem that two intersecting straight lines can establish a plane. So through the partial differential of variable x of this surface equation, one tangent line can be determined. And the other one is through partial differential in direction y determined. If we get a function,

$$F(x, y, z) = 0, \tag{5.1}$$

and the variable z can also be presented as a equation of variables x and y,

$$z = f(x, y), \tag{5.2}$$

then we get a rewrite of Equation (5.1) like this,

$$F(x, y, f(x, y)) = 0. \tag{5.3}$$

Based on the chain rule, the expression for partial derivation in direction x of Equation (5.3) is shown as following:

$$\frac{\partial F}{\partial x} + \frac{\partial F}{\partial z} \cdot \frac{\partial f}{\partial x} = 0. \tag{5.4}$$

According to our definition $z = f(x, y)$, so Equation (5.4) will be rewritten as this,

$$\frac{\partial F}{\partial x} + \frac{\partial F}{\partial z} \cdot \frac{\partial z}{\partial x} = 0. \tag{5.5}$$

Naturally we also get,

$$\frac{\partial z}{\partial x} = -\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial z}}. \tag{5.6}$$

The tangent line in direction x should be a line in XOZ plane, and $\frac{\partial z}{\partial x}$ shows the slope of this tangent line. As we in chapter 4 already introduced, the quadric surface function can be presented in a expression of z. Now with the surface function and Equation (5.6) we can calculate the tangent line in direction x at any point on this quadric surface. And similarly we can get the slope of tangent line in direction y at the same point,

$$\frac{\partial z}{\partial y} = -\frac{\frac{\partial F}{\partial y}}{\frac{\partial F}{\partial z}}. \tag{5.7}$$

The next step, we set the equation for a general quadric surface as in chapter 4 mentioned Equation (4.11). According to the description above, now we get

$$\frac{\partial F}{\partial x} = 2Ax + Dy + Fz + G, \tag{5.8}$$

$$\frac{\partial F}{\partial y} = 2By + Dx + Ez + H, \tag{5.9}$$

$$\frac{\partial F}{\partial z} = 2Cz + Ey + Fx + I. \tag{5.10}$$

And now the slopes for two tangent lines are in following expression presented,

$$\frac{\partial z}{\partial x} = -\frac{2Ax + Dy + Fz + G}{2Cz + Ey + Fx + I}, \tag{5.11}$$

$$\frac{\partial z}{\partial y} = -\frac{2By + Dx + Ez + H}{2Cz + Ey + Fx + I}. \tag{5.12}$$

Now two tangent lines are available, the tangent plane is naturally identified, and also the normal of the surface at each point is identified.

In the practical application, we need not to accurately calculate the normalized vector[1] of each normal, only the direction of the normal is what we are interested. That means, we can set a vector, that is in the same direction as normal, but with arbitrary scalar size of the normal. This characteristic offers us a great freedom to set the vectors.

We set two vectors $\vec{a}$ and $\vec{b}$ as the following,

$$\vec{a} = (1, 0, \frac{\partial z}{\partial x}), \tag{5.13}$$

$$\vec{b} = (0, 1, \frac{\partial z}{\partial y}). \tag{5.14}$$

Then we can tell, that $\vec{a}$ is a vector in XOZ plane, and points the same direction as the tangent line which we've mentioned. And $\vec{b}$ lies in YOZ plane pointing the same direction as the other tangent line. And now we let,

$$\vec{f_2} = \vec{a} = (1, 0, \frac{\partial z}{\partial x}), \tag{5.15}$$

$$\vec{f_1} = \vec{b} = (0, 1, \frac{\partial z}{\partial y}). \tag{5.16}$$

Schematic drawing is seen in Figure (5.3). Now vector $\vec{f_3}$ obviously points the inverse direction as the surface normal. These expressions of $\vec{f_1}$ and $\vec{f_2}$ will be in a function block in the Simulink program file written. Then along the surface we get at each point a pair of the vectors with real-time calculated values. They are delivered to the kinematics block for accomplishing the orientation.

5.5 Implementation of the Translation Coordinates

In the experiment, what we are informed are the coordinates of the desired surface. Those coordinates are computed through surface fittings function. By introduction in the section above, orientations vector is already known. The position of the nozzle should undoubtedly match the coordinates of the surface points. But the vector $\vec{o}$ delivers the position of the point **P**, that point has been in chapter 3 introduced. This means, that a deviation exists between what we know and what the robot should know. The task is to find out the convention between them.

From the Figure (2.4) we can tell, that due to the mounting position of the 3D-printer's, the central axis of nozzle doesn't concentric with the central axis of end effector. The relative horizontal position offset is shown as the Figure (5.4).

Due to the own length of the end effector, the printer holding frame and the printer nozzle, there exists of cause also a vertical offset between nozzle and the point P. The Figure (5.5) shows a schematic drawing about the vertical offset. In the drawing an additional height offset valued '1 mm' is set. That's for the little interval between nozzle and the surface.

So according to Figure (5.5) the vertical offset can be calculate with the following equation,

$$z^* = z + 1 + 30 + (75 - 7) + 65. \tag{5.17}$$

[1]Here the normalized vector means the unit vector, the vector in same direction but with norm length 1

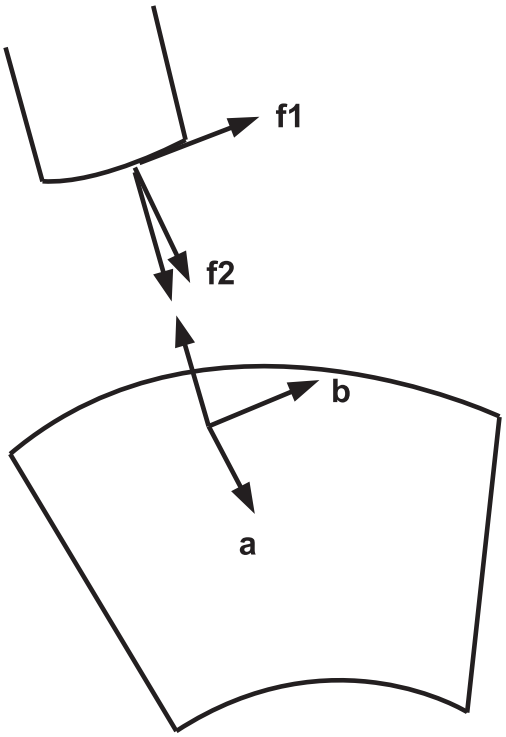


Figure 5.3: Schematic drawing of the orientation

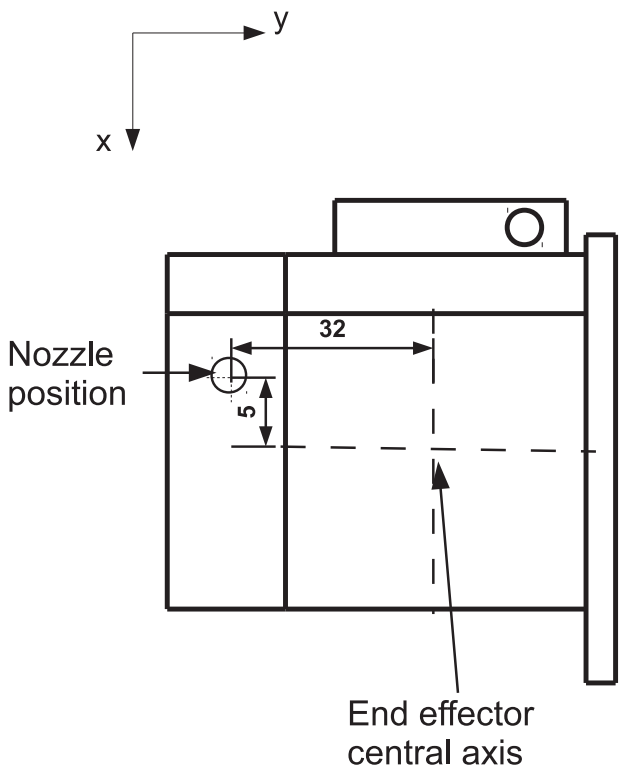


Figure 5.4: Horizontal offset between nozzle and central axis

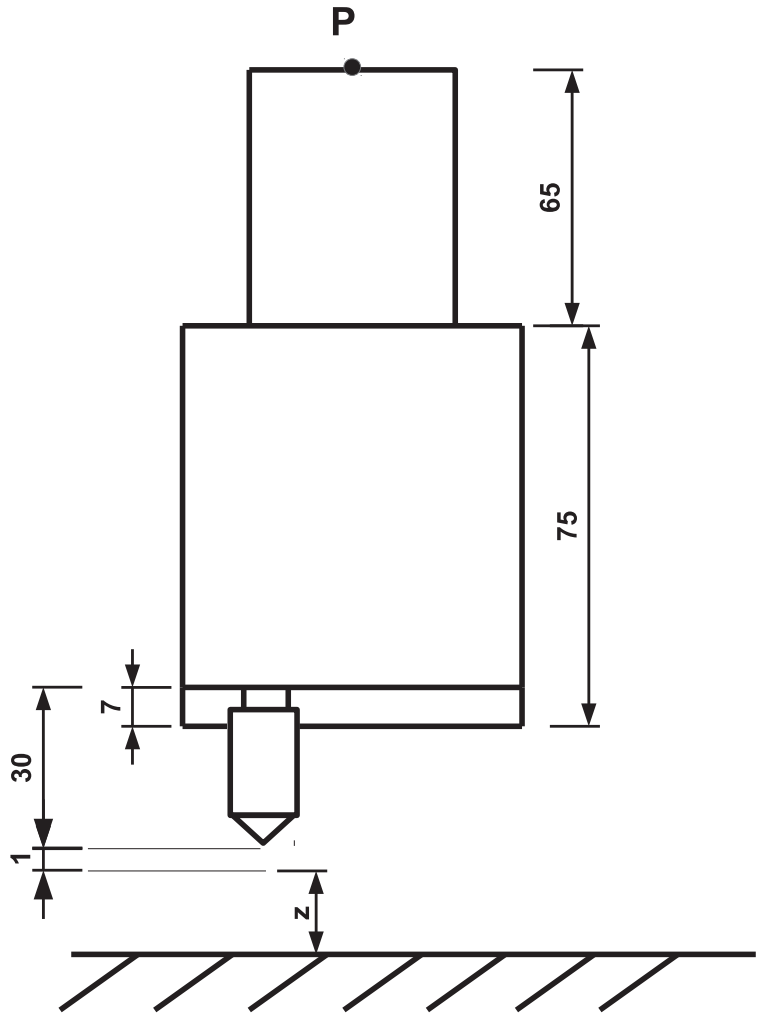


Figure 5.5: Vertical offset between nozzle and point **P**

And the whole translation relation between the nozzle peak and the point **P** could be presented in following equations. If we set the coordinate of nozzle peak as (x, y, z), and the coordinate of point **P** as (x^*, y^*, z^*), then we get,

$$x^* = x + 5, \tag{5.18}$$
$$y^* = y + 32, \tag{5.19}$$
$$z^* = z + 164. \tag{5.20}$$

Starting from the desired coordinates, along the direction of orientation's vector, with the relative horizontal offset and vertical offset, we can calculate the coordinate of $\vec{o}$.

The relative position between these two points is fixed during the robot motion. But the global coordinates of them changes in time. For more intuitively understanding the computation of the coordinates, based on their relative position, the two points are imagined as two vertices on the diagonal of a cuboid, seen in Figure (5.6).

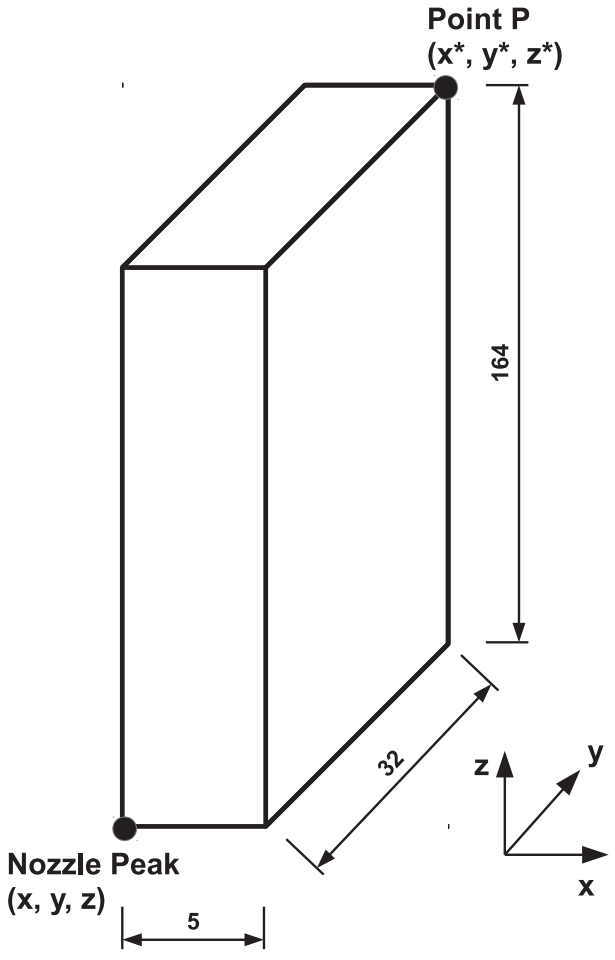


Figure 5.6: Relative position between nozzle peak and point **P**

Now the point where the nozzle peak locates is set as point **O**. And with the point **O** as origin point we build a local coordinate system. The orientation's vector is already known. We name

it as vector $\vec{n}$ in this coordinate system. Through rotating this rigid cuboid about tree axes x, y and z, in accordance with a certain order, we can let the cuboid stay in the same direction as the vector $\vec{n}$ points. Then the mathematical expression about this rotation sequence is the coordinates conversion between nozzle peak and $\mathbf{P}$.

At first the rotation matrices are going to introduce here as basics. As we have in chapter 3 already mentioned, we can use the trigonometric to build the rotation matrices. They are shown as below: $\mathbf{R_x}(\mathbf{u_1})$, the rotation matrix about axis x with angle u_1,

$$\mathbf{R_x}(\mathbf{u_1}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(u_1) & -\sin(u_1) \\ 0 & \sin(u_1) & \cos(u_1) \end{bmatrix}; \tag{5.21}$$

$\mathbf{R_y}(\mathbf{u_2})$, the rotation matrix about axis y with angle u_2,

$$\mathbf{R_y}(\mathbf{u_2}) = \begin{bmatrix} \cos(u_2) & 0 & \sin(u_2) \\ 0 & 1 & 0 \\ -\sin(u_2) & 0 & \cos(u_2) \end{bmatrix}; \tag{5.22}$$

and $\mathbf{R_z}(\mathbf{u_3})$, the rotation matrix about axis z with angle u_3,

$$\mathbf{R_z}(\mathbf{u_3}) = \begin{bmatrix} \cos(u_3) & -\sin(u_3) & 0 \\ \sin(u_3) & \cos(u_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.23}$$

In the experiment we followed such a rotation sequence, that first of all the rigid rotated about axis z with $-90°$, then it rotated about axis y with angle u_2, at last it rotated about axis x with angle u_1. A schematic explanation is seen in Figure (5.7).

Following the sequence above each rotation angle is in geometry solvable. The rotation angle in clockwise is set as negative, and the rotation angle in anticlockwise as positive. The first rotation angle about axis z is already known as $-90°$.

The second rotation angle u_2 about axis y will be solved with help of the vector dot product rule. The vector dot product rule says,

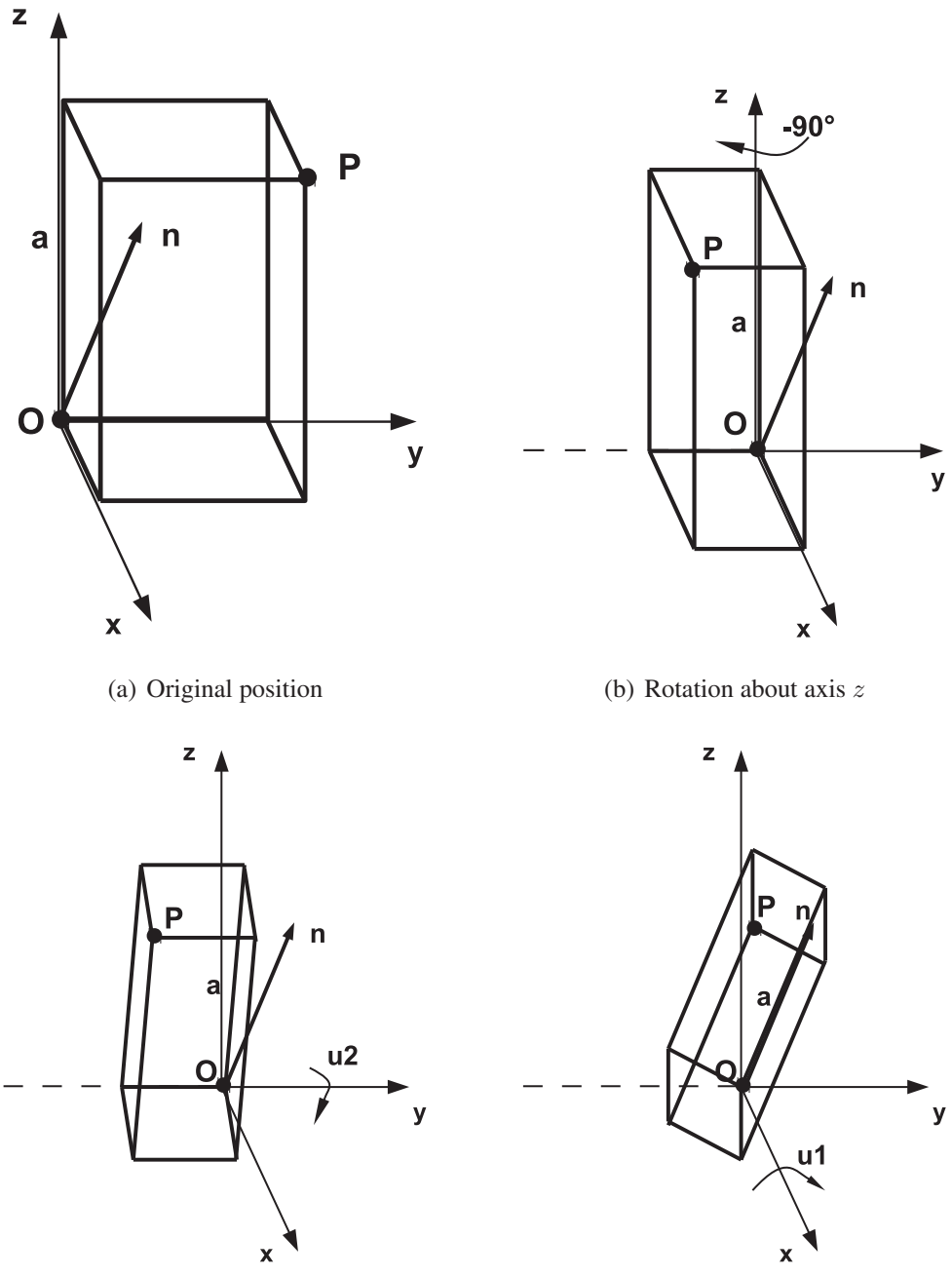
$$\vec{a} \cdot \vec{b} = \parallel \vec{a} \parallel \parallel \vec{b} \parallel \cos\theta \qquad \theta \in (0, 180°) \tag{5.24}$$

The geometric meaning of this equation is, that θ is the angle between $\vec{a}$ and $\vec{b}$. With two known vectors the angle is of cause able to be computed. Vector $\vec{n}$ is already known as orientation's vector. We set the unit vector of x-axis as $\vec{x}$, then $\vec{x} = (1, 0, 0)$. So according to the dot product rule and combined with the rotation direction's setting, we get the expression for angle u_2 as following,

$$u_2 = 90° - \arccos(\frac{\vec{n} \cdot \vec{x}}{\parallel \vec{n} \parallel \parallel \vec{x} \parallel}). \tag{5.25}$$

By calculating the third rotation angle u_1 about x-axis we also used the vector dot product rule. We set the unit vector of axis y as $\vec{y}$, then $\vec{y} = (0, 1, 0)$. For the calculation we set additionally a vector $\vec{n_2}$, and $\vec{n_2} = \vec{n} \times \vec{y}$. Then the expression for angle u_1 is as following,

$$u_1 = \arccos(\frac{\vec{n_2} \cdot \vec{y}}{\parallel \vec{n_2} \parallel \parallel \vec{y} \parallel}). \tag{5.26}$$



(a) Original position

(b) Rotation about axis z

(c) Rotation about axis y

(d) Rotation about axis x

Figure 5.7: Rotation sequence

Now followed the rotation sequence, and with the corresponding rotation angles, the entire rotation matrix could be built,

$$\mathbf{R} = \mathbf{R_x}(\mathbf{u_1}) \cdot \mathbf{R_y}(\mathbf{u_2}) \cdot \mathbf{R_z}(-\mathbf{90}^\circ). \tag{5.27}$$

Then the coordinates of point **P** in this local coordinate system after a series of rotation should be,

$$\mathbf{P} = \mathbf{R} \cdot \begin{bmatrix} 5 \\ 32 \\ 164 \end{bmatrix}. \tag{5.28}$$

Of cause the coordinates of **P** in the global coordinate system are required. If the coordinates of origin **O** in the global system is known as $\mathbf{o}$, then the global coordinates of **P** can be presented as,

$$\mathbf{P}^* = \mathbf{o} + \mathbf{R} \cdot \begin{bmatrix} 5 \\ 32 \\ 164 \end{bmatrix}. \tag{5.29}$$

By this solution, the consistence of the rotation sequence should be drawn attention. It's understandable, that the translation could be equally effective when it follows another rotation sequence. But when the sequence is changed, the calculation of rotation angles should be also reformed and suit itself to the new sequence. Exactly based on this reason, the sequence is at first to be determined, then the rotation angles. The Matlab source codes are shown as below.

```
function o  = transf(o1,f1,f2,pos)
%#codegen

ff = cross(f1,f2);
f = -ff;
e1 = [1,0,0]';
e2 = [0,1,0]';

% Rotating sequence, first about e2 , then about e1
if (f(3)>=0)
   u2 = 90 - acosd(dot(f,e1)/norm(f));
else
   if (f(1)>=0)
      u2 = 90 + acosd(dot(f,e1)/norm(f));
   else
      u2 = 270 - acosd(dot(f,e1)/norm(f));
   end
end

n2 = cross(f,e1);
if (f(3)>=0)
  if (f(2)<=0) % When ny in 3rd and 4th quadrant
   u1 = acosd(dot(n2,e2)/norm(n2));
```

```
  else
   u1 = -acosd(dot(n2,e2)/norm(n2));
  end
else
  if (f(2)>=0)
   u1 = 180 - acosd(dot(n2,e2)/norm(n2));
  else
   u1 = acosd(dot(n2,e2)/norm(n2)) - 180;
  end
end

% Building the rotation matrix
Rz = [cosd(-90), -sind(-90), 0;
      sind(-90),  cosd(-90), 0;
      0     ,  0     , 1];

c2 = cosd(u2);
s2 = sind(u2);
Ry = [c2, 0, s2;
      0,  1,  0;
      -s2, 0,  c2];

c1 = cosd(u1);
s1 = sind(u1);
Rx = [1, 0,   0;
      0, c1, -s1;
      0, s1,  c1];

% The rotation sequence must be first about axis-y,
% then aboutaxis-x
R = Rx*Ry*Rz;

o = R*pos + o1;
```

Chapter 6

Test Runs and Possibility of Improving the Accuracy

In this chapter the experimental results are analyzed. The possibility about reducing the pose error is presented. An overview about robot calibration is led into this thesis. This work completed some meaningful theoretical preparation for the further accuracy enhancing of this 3D printing method. And also a simple method to determine the joints error is introduced.

6.1 Test Runs

After completing the orientation and translation computation, several test runs were executed. For the test run such a motion path is planed. Several points from the outermost boundary of the scanned surface are picked out. And they are sequentially ordered to form the motion path. At first the test run was executed in Simulink virtual world simulation, then in B&R simulation, that includes the simulation of the motor drive controllers. Finally it was run on the real robot. Figure (6.1) shows the robot pose during a test run.

6.2 Analysis of Experimental Results

In the test runs the printer nozzle has maintained the correct orientation, and the motion tracks were generally according to the right slopes and curvatures of the surface object. These test runs verified the series of work accomplished before. But there are still light deviations between the motion track and the real surface possible. The reasons, that cause the deviation, might be found in two aspects.

1. One reason might be the measuring deviation by surface scanning, that causes the fitted surface deviating from the real surface.

2. Another reason could be the inaccuracy of the robot's gears, that causes the inaccuracy of the reaching position for robot's end effector.

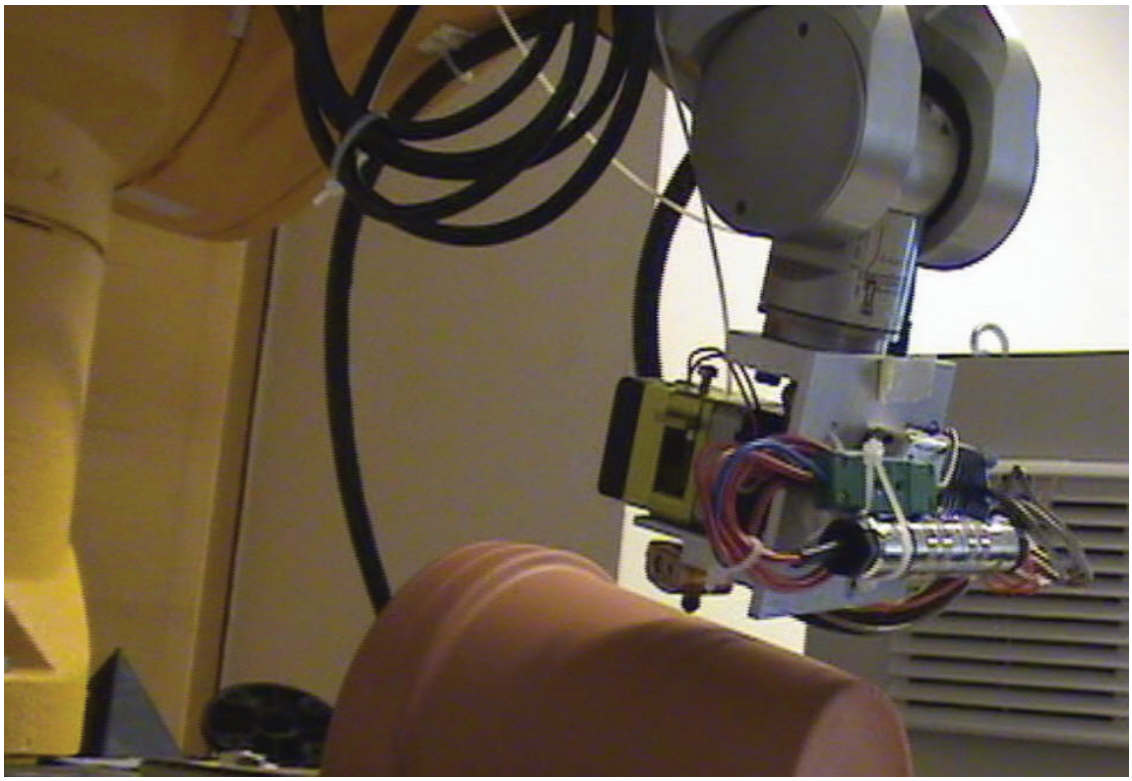


Figure 6.1: Robot pose in test run

As Klimchik presented in his work, there are two approaches to improve the identification accuracy [14],

- optimization of the manipulator measurement configurations and
- enhancing the objective function, minimized impact of the measurement noise.

Assuming that the laser sensor instrument will not be replaced, there exists another possibility to improve the accuracy. Robot calibration process could be led into this experiment. It will be introduced in next section.

6.3 Overview of Robot Calibration

Robot calibration is a process by which robot positioning accuracy can be improved by modifying the positioning software instead of changing the design of robot or its control system [15]. Here the accuracy doesn't mean resolution or repeatability. Accuracy is ability of the robot to move to a pose defined in the workspace [16]. Most of the research works have concentrated on the kinematic model-based calibration [17–19]. Of course there is also non-kinematic calibration. In non-kinematic calibration the robot's joints or links are no longer considered as perfect rigid. In those cases more non-geometrical error sources such as temperature or backslash have been considered. Therefor the non-kinematic calibration is more complicated.

Kinematic calibration is considered as global calibration method [15]. It consists of four sequence steps:

- Modeling
- Measurement
- Identification
- Compensation or Correction

These four steps will be generally introduced below.

6.3.1 Modeling

Modeling is to find a mathematical description of the geometry and motion of the manipulator. It fulfils the conversion of the definition of poses in workspace into joint space. In chapter 3 the most famous modeling conversion, D-H modeling is already described. There are also some other well known modeling methods such as Hayati's model and Mooring's Zero-reference model [20].

6.3.2 Measurement

The goal of the second step is to determine the position of either end effector or tool of the robot. The actual measured positions will be compared with the predicted positions to obtain the inaccuracy data. By many research works different methods and different measuring devices have been used.

6.3.3 Identification

By identification, the kinematic errors are identified, and an error model is established. And by applying numerical methods such as least square algorithm or Levenberg-Marquardt algorithm the inaccuracy will be minimized. Here a nearer introduction about establishing an error model follows.

As introduced in chapter 3 Equation (3.1), A_i is the transformation from coordinate system Σ_i to Σ_{i+1}. And a_i, α_i, d_i and θ_i are the four parameters. If small parameter errors Δa_i, $\Delta\alpha_i$, Δd_i and $\Delta\theta_i$ happen to the four kinematic parameters, then under error propagation rule [21] the error of transformation matrix can be presented as

$$\Delta A_i = \frac{\partial A_i}{\partial a_i}\Delta a_i + \frac{\partial A_i}{\partial \alpha_i}\Delta\alpha_i + \frac{\partial A_i}{\partial d_i}\Delta d_i + \frac{\partial A_i}{\partial \theta_i}\Delta\theta_i. \tag{6.1}$$

Also based on Equation (3.5) the error of whole transformation matrix $\mathbf{T}$ can be written as [22]

$$\Delta\mathbf{T} = \frac{\partial \mathbf{T}}{\partial a_1}\Delta a_1 + \frac{\partial \mathbf{T}}{\partial \alpha_1}\Delta\alpha_1 + \frac{\partial \mathbf{T}}{\partial d_1}\Delta d_1 + \frac{\partial \mathbf{T}}{\partial \theta_1}\Delta\theta_1 + \ldots + \frac{\partial \mathbf{T}}{\partial a_6}\Delta a_6 + \frac{\partial \mathbf{T}}{\partial \alpha_6}\Delta\alpha_6 + \frac{\partial \mathbf{T}}{\partial d_6}\Delta d_6 + \frac{\partial \mathbf{T}}{\partial \theta_6}\Delta\theta_6, \tag{6.2}$$

where

$$\frac{\partial \mathbf{T}}{\partial a_i} = \mathbf{A_1} \cdot \ldots \cdot \frac{\partial \mathbf{A_i}}{\partial a_i} \cdot \ldots \cdot \mathbf{A_6} \tag{6.3}$$

$$\frac{\partial \mathbf{T}}{\partial \alpha_i} = \mathbf{A_1} \cdot \ldots \cdot \frac{\partial \mathbf{A_i}}{\partial \alpha_i} \cdot \ldots \cdot \mathbf{A_6} \tag{6.4}$$

$$\frac{\partial \mathbf{T}}{\partial d_i} = \mathbf{A_1} \cdot \ldots \cdot \frac{\partial \mathbf{A_i}}{\partial d_i} \cdot \ldots \cdot \mathbf{A_6} \tag{6.5}$$

$$\frac{\partial \mathbf{T}}{\partial \theta_i} = \mathbf{A_1} \cdot \ldots \cdot \frac{\partial \mathbf{A_i}}{\partial \theta_i} \cdot \ldots \cdot \mathbf{A_6} \tag{6.6}$$

According to Gong's introduction in his article [22], with Equation (6.2) the final positional and orientational changes due to parameter errors can be calculated. Rewriting this function, it can be presented in a more compact form

$$\Delta \mathbf{X} = \mathbf{J} \Delta \mathbf{P}, \tag{6.7}$$

where
$\Delta \mathbf{X}$ presents total positional and orientational errors of the end effector, including geometric error, compliance error and thermal error.
$\Delta \mathbf{P}$ presents the total parameters errors.
$\mathbf{J}$ is the Jacobian matrix. Until here the error synthesis model is established.

6.3.4 Compensation or Correction

In this phase, using the information obtained from the previous steps, a new and accurate kinematic model is established. This new model should be a modification from the nominal model accomplished by joints compensation. Through implementation of the new model the manipulator performance for reaching a desired position will be improved.

Generally calibration can enhance the robot positioning accuracy. It can be predicted, that the robot can follow the approximated surface more accurately.

6.4 Simple Measurement of Joints Errors

Normally speaking by a classic robot calibration process, the robot end effector with a calibration tool is controlled to reach a number of points in the workspace. By each position, the coordinates and the robot pose are measured and recorded. The error model is normally a huge matrix expression with a dozen parameters and parameter errors.

In this thesis just a simple measuring method is proposed to determine the joints errors. Through the introduction about D-H modeling in chapter 3, it is known that the model consists of geometrical parameters and rotation angles of joints. These rotation angles determine the pose of the robot, therefor the measurement errors of the rotation angles are very important for the accuracy of the robot.

The basic principle of this measuring method is using horizontal distance and vertical offsets to indirectly measure the rotated angles. Figure (6.2) shows the conversion relation between angle and offsets.

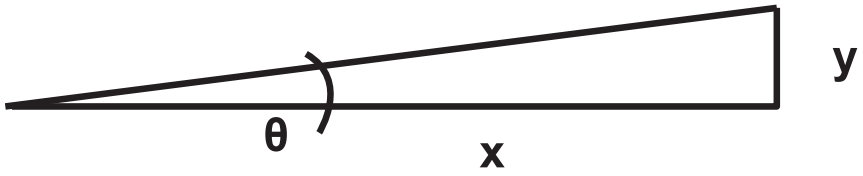


Figure 6.2: Conversion relation between angle and offsets

According to the schematic drawing, when the rotated angle is sufficiently small, the angle in degrees can be calculated by Equation (6.8).

$$\theta = \tan\theta = \frac{y}{x}\frac{180}{\pi} \tag{6.8}$$

To implement this measuring method, the following measure instruments were used:

- A laser pointer is mounted on the 3D printing block or on the end effector to point the position, see Figure (6.3). The mounting position should be able to sense the rotated angle.
- A scale was set on the opposite wall, shown as in Figure (6.4). The position of laser spot before rotation and after rotation will be marked. The vertical offset is designated as y.
- Furthermore a distance measurement instrument was applied to determine the distance between laser emission point and the wall, see Figure (6.5). This distance is labeled as x.

The measurement results are shown in Table (6.1).

Axes	nominal $\theta(^\circ)$	$x(mm)$	$y(mm)$	measured $\theta(^\circ)$
1	0,5	5474,15	48	0.5024
2	1	8048	143	1.0179
3	1	8048	137	0.9752
4	1	5436	84	0.8853
5	1	7464	100	0.7676
6	1	5431	92	0.9705

Table 6.1: Identification of angle deviations of six joints

In Table (6.1) the 'nominal θ' represents the angle value entered to the robot controller. And the 'measured θ' represents the values calculated from the measurements. The measurement error of x is ± 1 mm, which is specified in the data sheet of the instrument. And the measurement error of y is estimated with ± 0.5 mm, determined by the size of the laser spot and the recognition capability of the human eye.

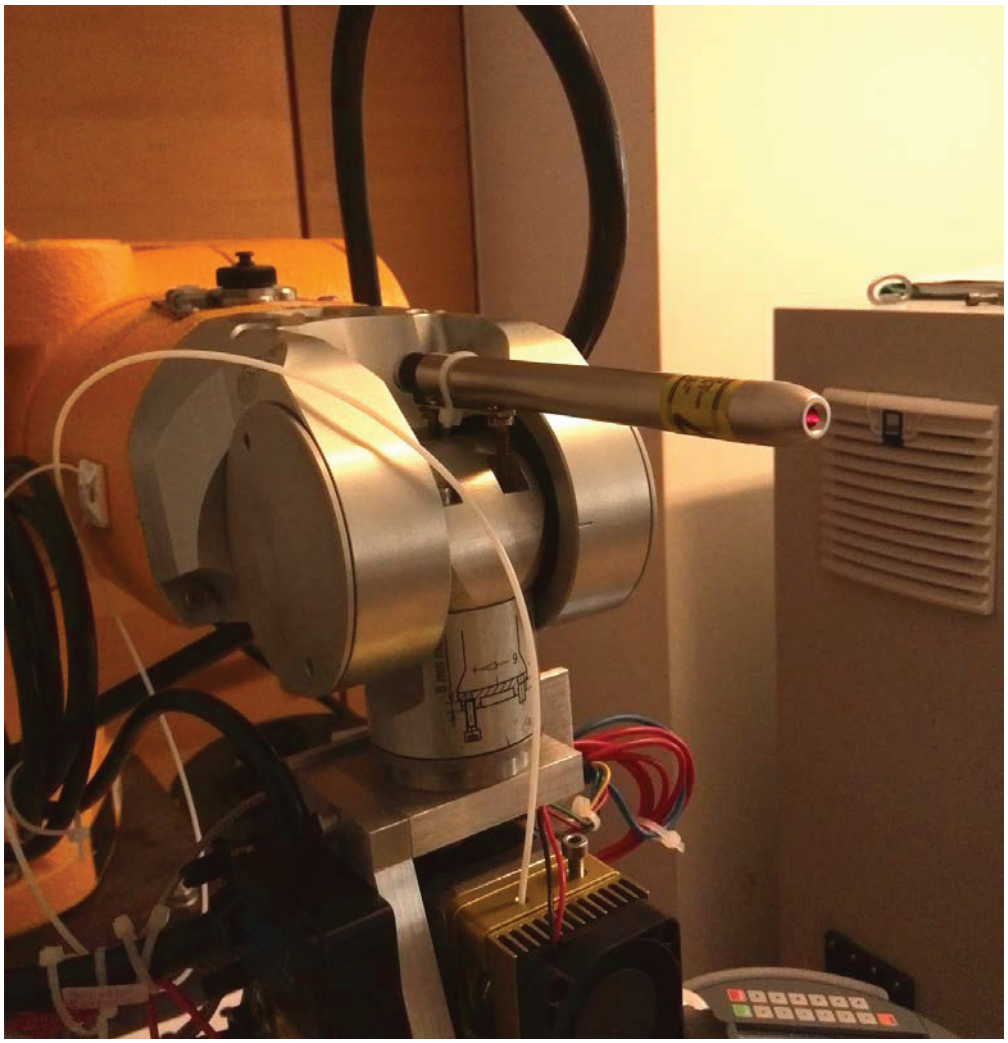


Figure 6.3: Laser pointer mounted on end effector

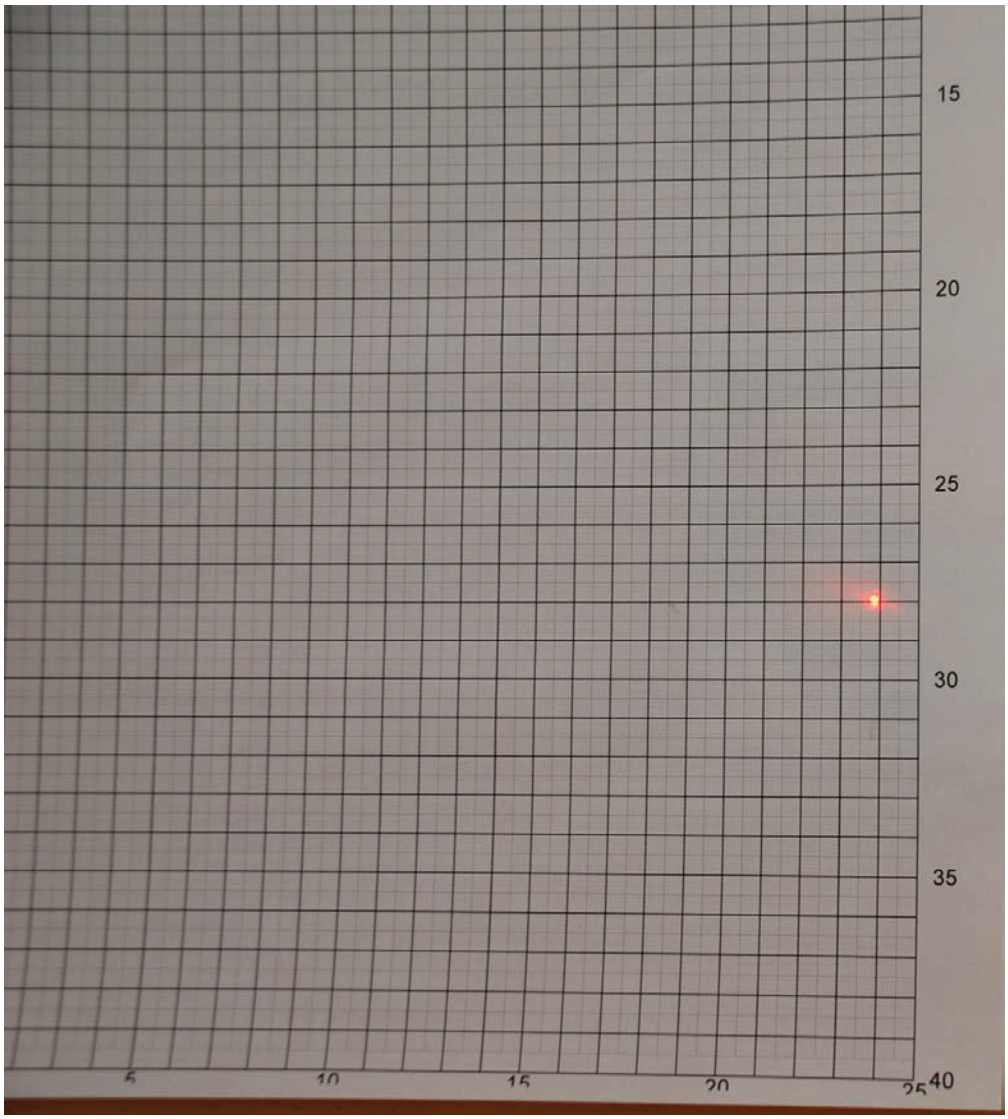


Figure 6.4: Scale for measuring vertical offset

Figure 6.5: Distance measurement instrument

According to error propagation rule and Equation (6.8), the rotation angle's error model expression follows with

$$\Delta\theta = (|-\frac{y}{x^2}||\Delta x| + |\frac{1}{x}||\Delta y|)\frac{180}{\pi}, \tag{6.9}$$

where Δx and Δy are the absolute errors of measurement. Table (6.2) shows the angles' error results according to Equation (6.9).

Axes	measured $\theta(^\circ)$	$\Delta\theta(^\circ)$
1	0.5024	0.0053
2	1.0179	0.0037
3	0.9752	0.0037
4	0.8853	0.0054
5	0.7676	0.0039
6	0.9705	0.0055

Table 6.2: Computing results of angles' errors

Equation (6.8) could be considered as the error model of this simple measurement. The error minimizing work will not be further discussed in this thesis. And also the compensation work is not included in our scope. Here it just proposed a simple model to understand the calibration process and completed some elementary work for calibration.

Chapter 7

Conclusion

This thesis introduced a new proposal about combining an industrial robot and a 3D printer extruder. This concept was expected to expand the application area of 3D printing compared to consumer grade 3D printers. And it was also expected to enable the printer extruder to achieve printing on non-planar surfaces.

A measuring method was presented, applying a laser displacement sensor to scan a surface. With the Levenber-Marquardt method, which is implemented as a mathematical tool in Matlab, the scanned surface was approximated with a quadric surface. The applicability of this method for surface fitting task is validated. It also was proved that the resolution of the laser sensor is suitable for the scanning task. The approximation result was used in the next step for path planning.

In this thesis the robot kinematic modeling was based on Denavit-Hartenberg model. Around this kinematics model a simulation program in Simulink was built. Besides the kinematics block, the function contains a path planning block as well. The function of that block is computing the translation and orientation variables and delivering them to the kinematics block. Through the analysis of requirement of 3D printing and the kinematics model as well as the geometric characteristics of the hardware, an algorithm about computing the translation and orientation variables was developed. The algorithm was successively verified in Simulink simulation environment and in the B&R simulation environment as well. Based on the coordinates of approximated surface, a motion path for reconstructing the surface is planed. The robot pose for 3D printing is adjusted appropriately. And with a Simulink program the robot is controlled in order to accomplish this motion path. The motion path was also proved to reconstruct the surface object very well.

For a more accurate surface printing, implementing the robot calibration process is an option. In this thesis the theoretical preparation for calibration is also included.

Appendix A

Measurement Data

A.1 Measurement Data of Surface Model

Points	x	y	z	voltage (v)
1	391	50.69	168.3	-8.54
2	390.9	39.25	173	-7.65
3	390.9	25.89	175.2	-7.7
4	390.9	9.221	175.2	-6.47
5	390.9	-5.085	175.2	0.72
6	403	-5.085	175.2	6.8
7	404.5	10.17	175.2	-0.6
8	404.4	25.98	175.2	-1.63
9	404.4	39.56	175.2	3.13
10	404.5	51.96	172.7	7.3
11	417.1	51.7	170.1	8.6
12	417.1	38.97	170.1	-1.64
13	417.6	24.62	170.1	-6.25
14	417.6	9.64	170.1	-4.74
15	417.6	-3.09	170	2.36
16	429.3	-3.1	169.9	8.02
17	431.2	10.87	169.8	0.8
18	431.2	25.79	169.8	-0.57
19	431.2	38.37	169.8	3.7
20	431.3	51.51	166.3	7.17
21	443.9	49.26	164.7	8.15
22	445.4	38.49	164.7	-0.25
23	445.4	26.07	164.7	-4.86
24	445.4	12.04	164.7	-3.56
25	445.4	-0.7254	164.6	3.73

Appendix B

Technical Data of Stäubli RX60

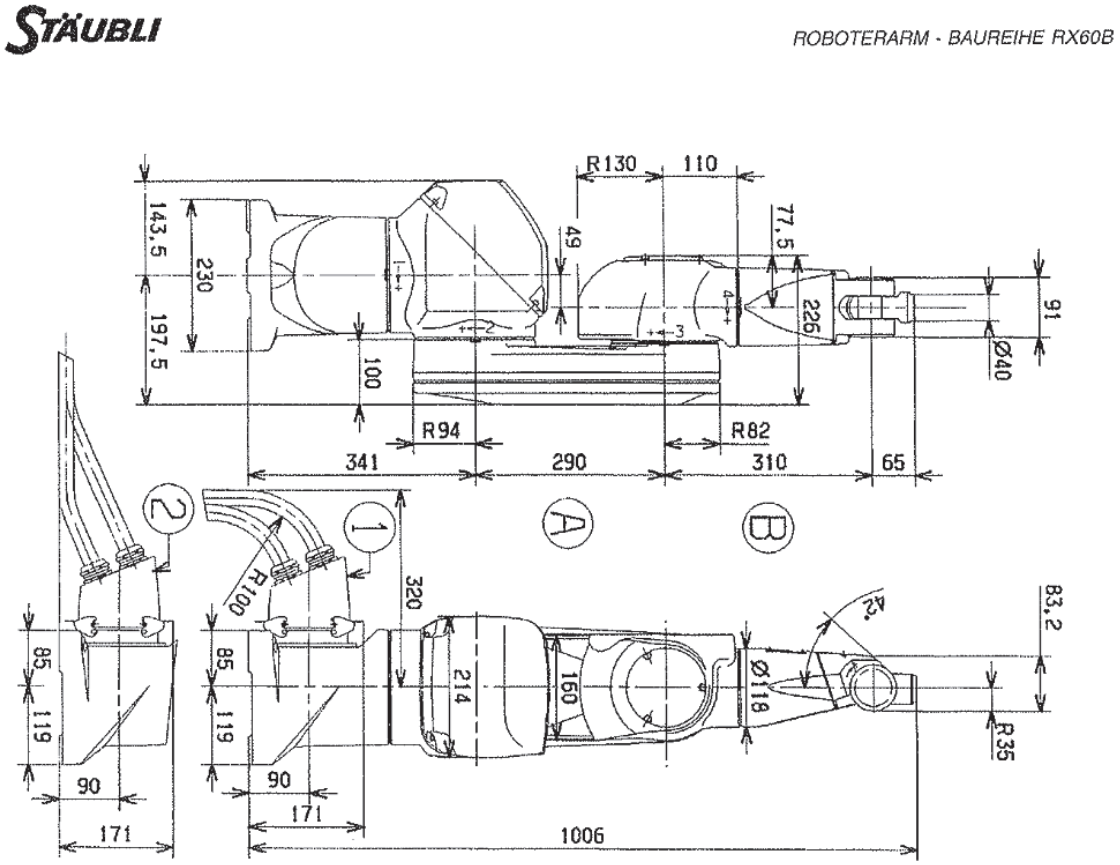


Figure B.1: Dimensions of Stäubli RX60

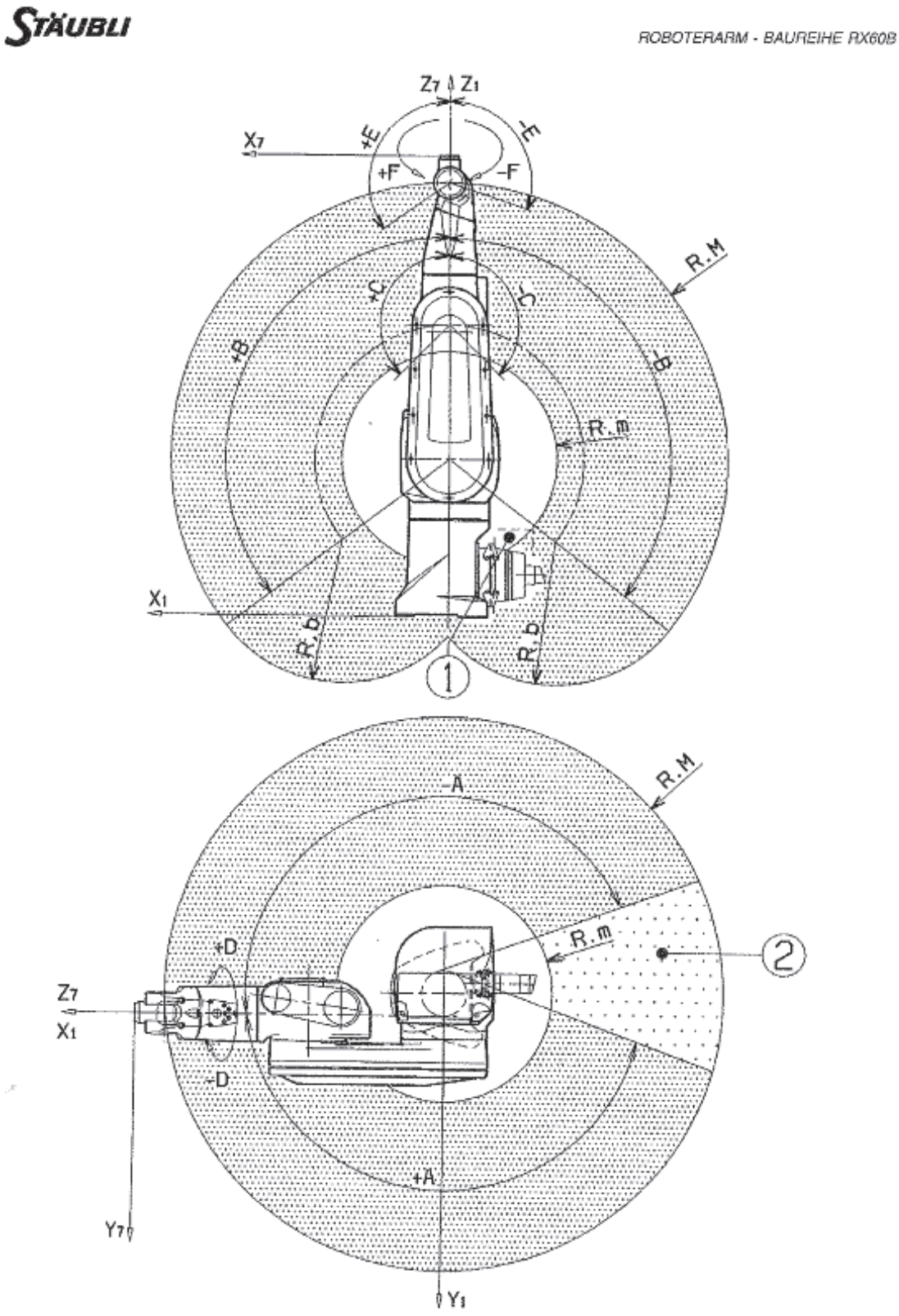


Figure B.2: Workspace of Stäubli RX60

Appendix C

Source Code and Simulink Program

C.1 Plane Fitting Function

```
function [u1,u2,a,n,jacobian] = planefit(coord)
zv = coord(:,4);
zz = coord(:,3)+71-(45 - zv*(-5/10))-63-65;
% height correction for laser head

x = [coord(:,1),coord(:,2)];

f = @(a,x)a(3) + a(2)*x(:,2) + a(1)*x(:,1);
ydata = zz;
a0 = [1 1 10];
[a,resnorm,residual,exitflag,output,lambda,jacobian]
 = lsqcurvefit(f,a0,x,ydata);
```

C.2 Quadric Surface Fitting Function

```
function [a,x,y,cov,J,sigma,resnorm,residual] = identQuadric2(coord)

v = coord(:,5); % Voltmeter reading from laser measure instrument

%z value calculating, bases on the original labor data 'dataSet'
y = coord(:,4)+71-(45 - v*(-5/10))-63-65;

x = [coord(:,2)-45.5, coord(:,3)+11.5];

% Starting value setting bases on the original cone ('dataSet')
% function: a*x^2 + b*y^2 + c*z^2 + g*x +h*y + j
%a0 = [0.04  -1  -1  0 0 0  -75  100  0  2.7e4];
a0 = [0.04  -1  -1  0 0 0  -75  40  0  2.5e4];

% surface fitting's boundary setting for the original cone
lb = [-2 -2 -2  -0.04 -0.04 -0.04  -90   30   -0.04  1e4];
ub = [ 2  0  0   0.04  0.04  0.04  -50   50   0.04   3e4];

% increasing the max evaluation's limit from defult '1000' to '1500'
options = optimset('MaxFunEvals',1500);
[a,resnorm,residual,exitflag,output,lambda,jacobian]
  = lsqcurvefit(@fquadric, a0,x, y,lb,ub,options);
J = jacobian;
cov = inv(J'*J);
sigma = sqrt(diag(cov));
z = fquadric(a, x);

%rewriting the x,y,z data into the same size 5*5 matrics
xx = x(:,1);
X = reshape(xx,5,5);
yy = x(:,2);
Y = reshape(yy,5,5);
Z = reshape(z,5,5);

surf(X, Y, Z)

hold on
plot3(x(:,1), x(:,2), y, 'o' )

hold off
```

C.3 Z-coordinate Computing Function in Simulink

```
function Z  = zCalculation(x,y,a)
%#codegen
aa = a(3);
bb = a(5)*y + a(6)*x + a(9);
cc = a(7)*x + a(1)*x*x + a(2)*y*y + a(4)*x*y + a(8)*y + a(10);

z1 = (-bb + sqrt( bb.*bb - 4*aa.*cc ))./(2*aa) ;
z2 = (-bb - sqrt( bb.*bb - 4*aa.*cc ))./(2*aa) ;
 if (z1 >= z2)   %insure that we always get the bigger value from
     Z = z1;     %the two roots, cause it should be the upper half of the
 else
     Z = z2;
 end
```

C.4 Orientation Vectors Setting Function in Simulink

```
function [f1,f2]  = orientation(x,y,z,a)
%#codegen

% the general quadric function
% F = ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0
% F's partial derivative of x,y and z, imported from mapel
Fx = 2*a(1)*x + a(4)*y + a(6)*z + a(7);
Fy = 2*a(2)*y + a(4)*x + a(5)*z + a(8);
Fz = 2*a(3)*z + a(5)*y + a(6)*x + a(9);

% the partial derivative of x and y for z
dzx = -Fx/Fz;
dzy = -Fy/Fz;

%e1 = [1,0,0];
%e2 = [0,1,0];

f2 = [1,0,dzx];
f1 = [0,1,dzy]; % let the axis with ball in this direction
```

C.5 Translation Vector Setting Function in Simulink

```
function o  = transf(o1,f1,f2,pos)
%#codegen

ff = cross(f1,f2);
f = -ff;
e1 = [1,0,0]';
e2 = [0,1,0]';

% Rotating sequence, first about e2 , then about e1
if (f(3)>=0)
   u2 = 90 - acosd(dot(f,e1)/norm(f));
else
   if (f(1)>=0)
      u2 = 90 + acosd(dot(f,e1)/norm(f));
   else
      u2 = 270 - acosd(dot(f,e1)/norm(f));
   end
end

n2 = cross(f,e1);
if (f(3)>=0)
  if (f(2)<=0) % When ny in 3rd and 4th quadrant
   u1 = acosd(dot(n2,e2)/norm(n2));
  else
   u1 = -acosd(dot(n2,e2)/norm(n2));
  end
else
  if (f(2)>=0)
   u1 = 180 - acosd(dot(n2,e2)/norm(n2));
  else
   u1 = acosd(dot(n2,e2)/norm(n2)) - 180;
  end
end

% Building the rotation matrix
Rz = [cosd(-90), -sind(-90), 0;
      sind(-90),  cosd(-90), 0;
      0     ,  0     , 1];

c2 = cosd(u2);
s2 = sind(u2);
Ry = [c2, 0, s2;
      0,  1,  0;
```

```
      -s2, 0,  c2];

c1 = cosd(u1);
s1 = sind(u1);
Rx = [1, 0,   0;
      0, c1, -s1;
      0, s1,  c1];

R = Rx*Ry*Rz;
% The rotation sequence must be first about axis-y,then about
% axis-x
o = R*pos + o1;
```

C.6 Simulink Simulation Program

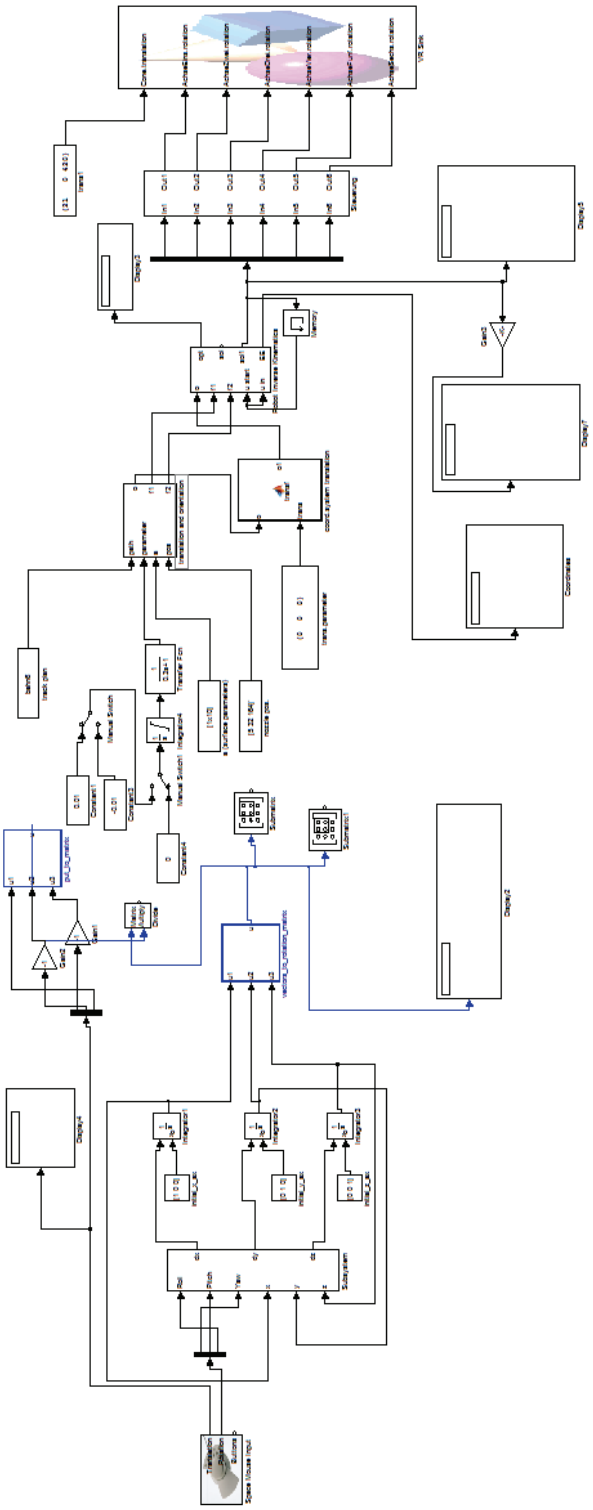


Figure C.1: Simulink Program

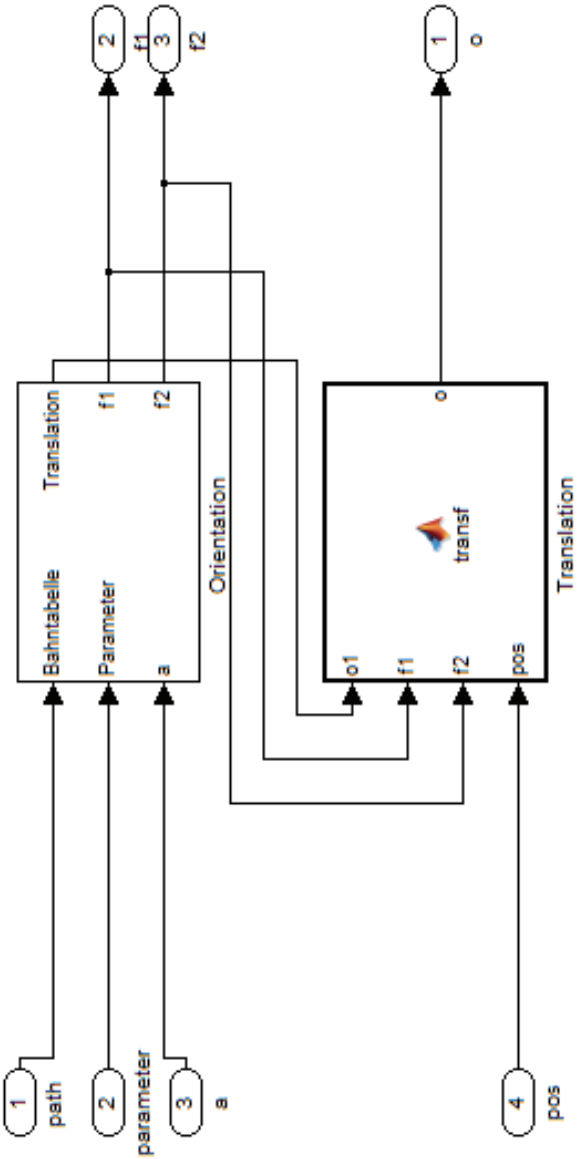


Figure C.2: Pose Computing Block

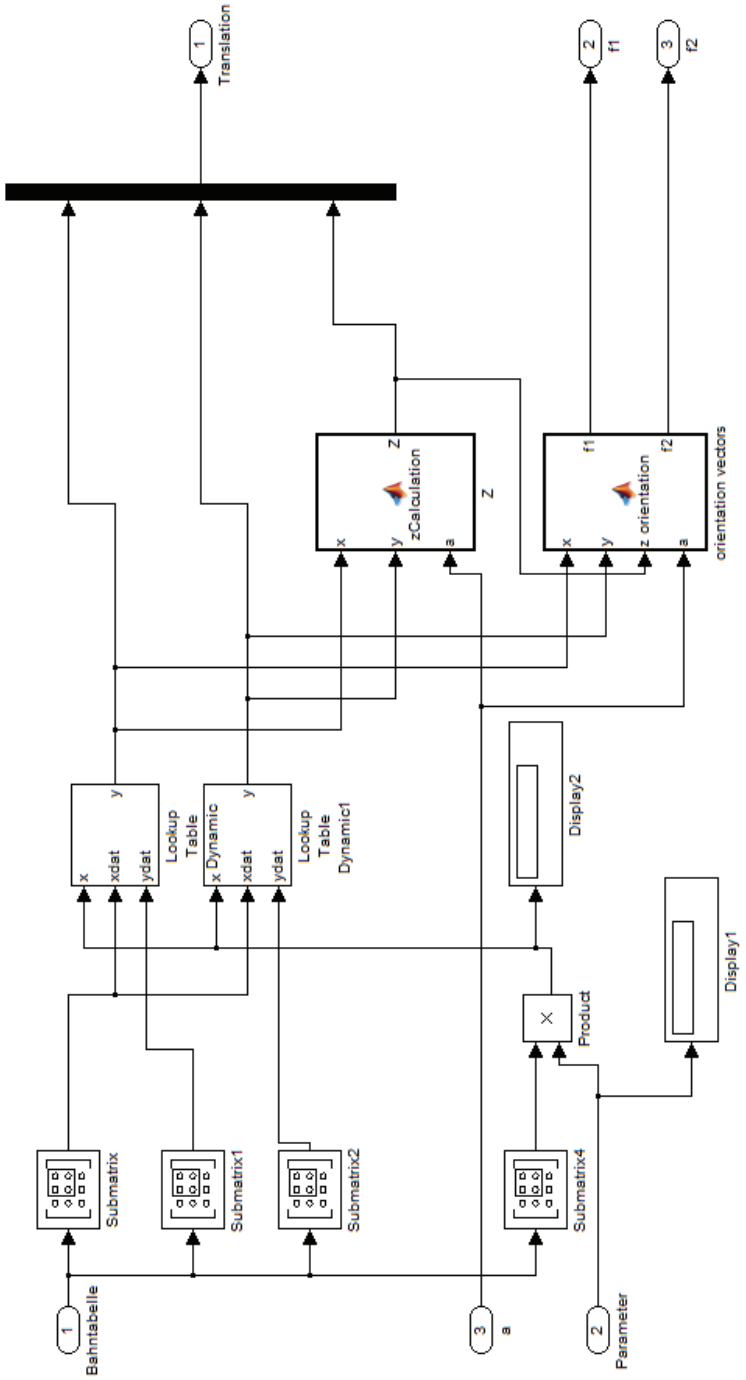


Figure C.3: Orientation Computing Block

List of Figures

List of Tables

Bibliography

[1] E. Sachs, M. Cima, and J. Cornie. Three-dimensional printing: Rapid tooling and prototypes directly from a cad model. *CIRP Annals - Manufacturing Technology*, 39(1):201 – 204, 1990.

[2] B. Berman. 3-d printing: The new industrial revolution. *Business Horizons*, 55(2):155 – 162, 2012.

[3] S. Devijver. Building your own 3d printer. http://reprapbook.appspot.com.

[4] The Math Works. Matlab r2012a product help. http://www.mathworks.com.

[5] Y. Li. 6 dof motion control of a robot-driven camera and its application in virtual scenes. Master's thesis, Institute for Automation, Montanuniversity Leoben, 2011.

[6] H. Zhuang and Z.S. Roth. *Camera-aided robot calibration*. CRC press, 1996.

[7] A. Gfrerrer. Kinematik und robotik. Institute for Geometry, Technical University of Graz, 2008.

[8] W. Kollment. Design, implementation and construction of a controller for a 6-dof serial robot. Master's thesis, Institute for Automation, Montanuniversity Leoben, 2014.

[9] S. Hayati and M. Mirmirani. Improving the absolute positioning accuracy of robot manipulators. *Journal of Robotic Systems*, 2(4):397–413, 1985.

[10] H. Sachs M. Husty, A. Karger and W. Steinhilper. *Kinematik und Robotik*. Springer-Verlag, 1997.

[11] H.P. Gavin. The levenberg-marquardt method for nonlinear least squares curve-fitting problems. Department of Civil and Environmental Engineering, Duke University, 2013.

[12] K. Madsen, H.B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.). Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.

[13] S. Levy. *Geometry Formulas and Facts*, volume 16 Quadric. CRC Press, 1995.

[14] S. Caro B. Furet A. Klimchik, Y. Wu and A. Pashkevich. Advanced robot calibration using partial pose measurements. In *Methods and Models in Automation and Robotics (MMAR), 2013 18th International Conference on*, pages 264–269, 2013.

[15] L.Zh. Fan D.Y. Yu A.Y. Elatta, P.G. Li and F. Luo. An overview of robot calibration. *Information Technology Journal*, 3:74–78, 2004.

[16] Z.S. Roth B.W. Mooring and M.R. Driels. *Fundamentals of manipulator calibration*. Wiley-interscience, 1991.

[17] Sh.N. Yang R.B. He, Y.J. Zhao and Sh.Z. Yang. Kinematic-parameter identification for serial-robot calibration based on poe formula. *Robotics, IEEE Transactions on*, 26(3):411–423, 2010.

[18] M. Becquet J.M. Renders, E. Rossignol and R. Hanus. Kinematic calibration and geometrical parameter identification for robots. *Robotics and Automation, IEEE Transactions on*, 7(6):721–732, 1991.

[19] W. Veitschegger and C.H. Wu. Robot accuracy analysis based on kinematics. *Robotics and Automation, IEEE Journal of*, 2(3):171–179, 1986.

[20] B.W. Mooring and G.R. Tang. An improved method for identifying the kinematic parameters in a six axis robot. In *Proceedings of the 1983 ASME Computers in Engineering Conference*, pages 79–84, 1984.

[21] H.H. Ku. Notes on the use of propagation of error formulars. *JOURNAL OF RESEARCH of the National Bureau of Standards - C. Engineering and Instrumentation*, 70C:263 – 273, 1966.

[22] Ch.H. Gong, J.X. Yuan, and J. Ni. Nongeometric error identification and compensation for robotic system by inverse calibration. *International Journal of Machine Tools and Manufacture*, 40(14):2119 – 2137, 2000.