CVMT 2010
Aalborg University: Department of Electronic System

How to improve the laser scanning of shiny surfaces which can give unwanted reflections.

3D laser scanner for painting robot

Cao-long HUYNH

10th semester 2010

CVMT 2010
Aalborg University: Department of Electronic System

Title:

How can we improve the laser scanning of shiny surfaces which can give unwanted reflections?

Period:

February 1st 2010 to Jun 3th 2010

Project Group:

CVMT2010 gr 1025

Group Member:

Cao-long HUYNH

Supervisor:

Thomas Moeslund

Report: 62

Pages: 64

Finished: Jun 1st 2010

Abstract:

The main purpose of this paper is to propose a solution to remove errors due to spurious reflections which can occur during the scanning of shiny surfaces. As glass objects and semi transparent objects, shiny surfaces are one of the main problems in 3D scanners based on triangulation. Different techniques which deal with the same problem are investigated. One of these methods was chosen in order to be implemented and tested. This technique uses a 3D data cloud density. The data cloud density of a 3D point is all the points that remain in the close vicinity of that point. Furthermore, the basic 3D laser scanner system is described in order to understand the theory around the data cloud density method.

An implementation and performance tests were done to prove the relevance of this method in the detection and the suppression of errors in cloud of points created by 3D laser scanners. There is no doubt about the advantage this method has to find false points in a set of measured points. However, the velocity of this technique can be problematic in certain area.

A better implementation has to be found to conclude if this method can be used in industrial area or not.

Preface

This report is the result of a 10th semester Computer Vision and Graphics project at Aalborg University in the spring 2010, running from the 1st of February to the 3rd of June. The project is proposed by INROPA, and the solution is developed to fit the requirements set forth by the firm.

The intended audience is not expected to have any prior knowledge about Computer Vision or Image processing. The target audiences for this report are people interested in 3D scanner laser and their current limits.

The report is divided into six parts: Pre-analysis, Analysis, Implementation, Test, Project closure and Appendix.

If the reader is interested in the implementation of the solution, a brief description of the code is found in part III. The source code of the implementation is moreover enclosed on the CD-ROM.

A full bibliography is located in the end of the report. The citations are referenced using the surname of the first author followed by the publishing year, e.g. [J. Chen et *al*, 1998].

Cao-long HUYNH

Table of content

1. Thanks

First of all I would like to thank the Aalborg Universitet which provided all infrastructures that I needed to work on this project. I would like to thank all my professors too for giving me all the necessary knowledge to understand and handle this project.

I would like to personally thank Thomas Moeslund for his precious advises during this project and for the time he devoted to me throughout the whole semester, knowing how to answer all my questions.

I also would like to thank Kåre Astorp Nissum, Troels Hessner Larsen and the INROPA company for their constant help and support.

Part I

Preliminary Analysis

2. Pre analysis

In this part, the problem statement is defined, taking into account INROPA requirements. The 3-D scanner of INROPA is quickly described in order to understand better how to answer the problem statement. Then one method is chosen upon several methods briefly described at the end of this part. This method will be described in the Analysis part.

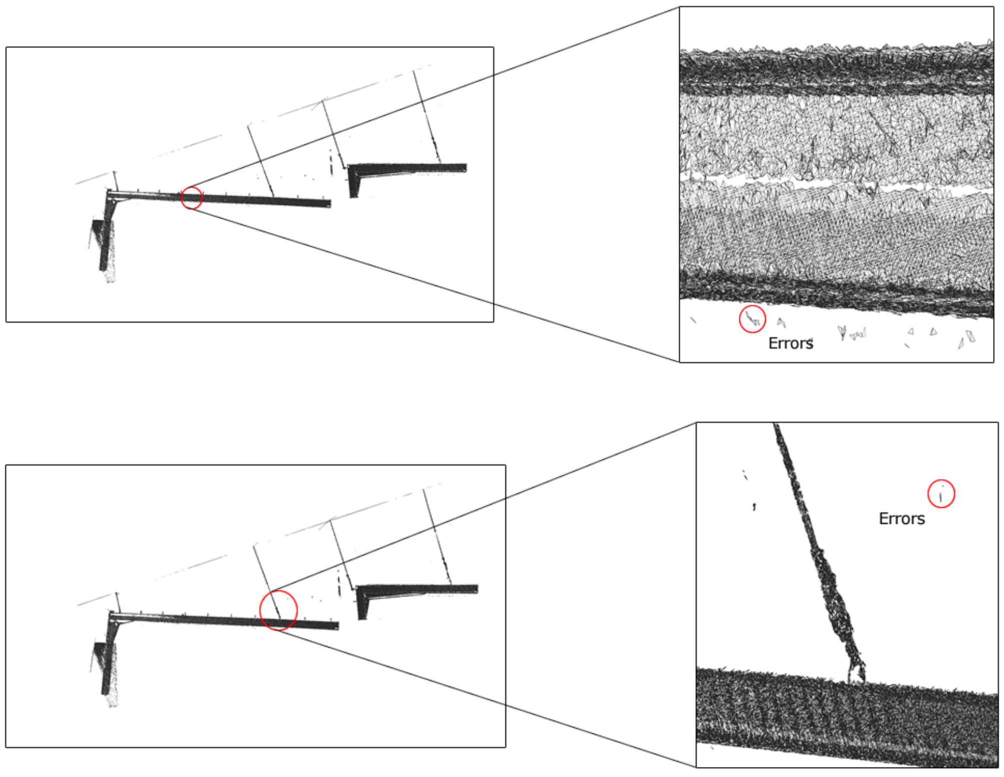
2.1. Introduction

Companies all over the world are using painting robots to maintain competiveness. Nowadays, since the price of industrial paint robots decreased drastically from their invention, almost all general industries can afford them [Wikipedia, 2007]. However, programming a robotized painting line to new objects implies stopping the production line. Moreover, the process is time-consuming and not without a cost for the industries [A. Pichler *et al*, 2004]. To make this process automatic, the painting robot must be aware of the following characteristics of the object to paint: the shape, the surface structure and the orientation. Nowadays, this can be done with the technology of laser scanning. In this way, INROPA developed a system using a laser scanner to create a 3D model of the object to paint in order to program painting robots. This system which is a triangulation-based sensors system will be described in the section 2.2 of this paper [K. A. Nissum et *al*, 2005].

Most of object surfaces in industry are made of polished metal or plastic. These materials are likely to reflect laser light in a specular way. The cameras used as sensors can interpret those specular components or noisy reflections as the main signal. In this case, the system will return false range values and then the range image would contain several spurious peaks and blobs which don't belong to the object [E. Trucco et *al*, 1994]. INROPA, in a perspective of improving the robustness of 3-D measuring in general, wants to optimize their laser scanning system by removing the potential errors due to specular reflections.

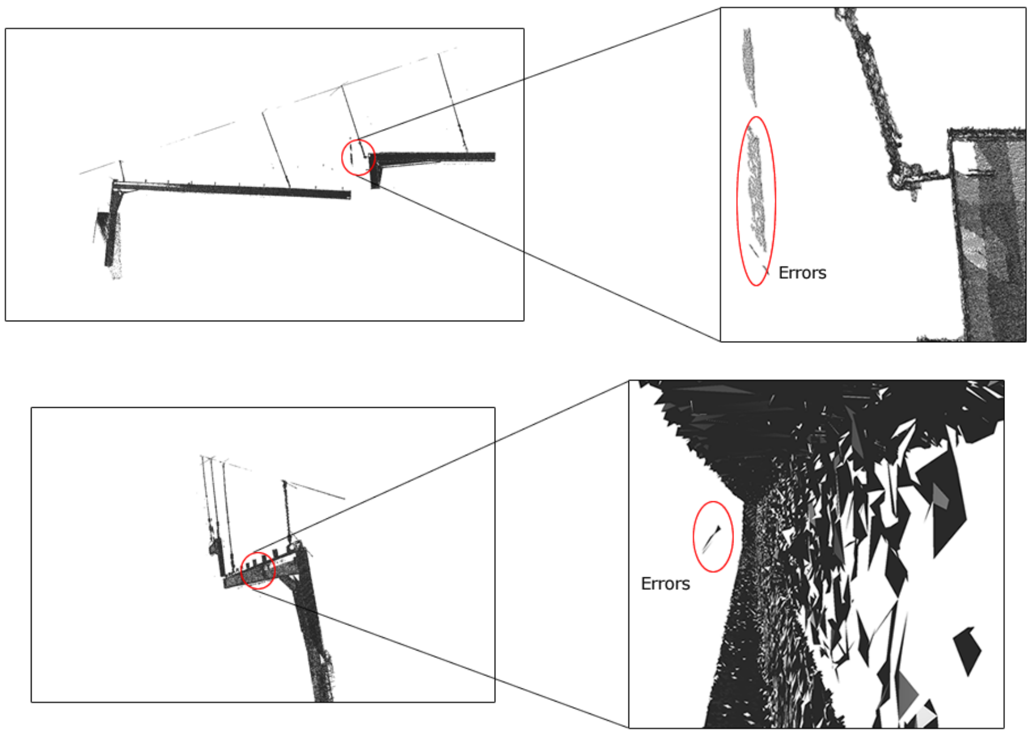
2.1.1. Problem statement

The laser scanning system of INROPA is based on the triangulation of a laser line light. As the system is used in industry, it often scans shiny and specular surface objects. Those kinds of objects can give spurious errors in the 3-D mesh produced by the INROPA software. Figure 2.1(a) and figure 2.1(b) show two types of errors from INROPA mesh reconstruction. The first type of error is a single point, more or less isolated. The second type of error is a blob of points. These second errors are mostly far from the mesh itself.



(a)

***Figure 2.1**: **(a)** Dot type errors from spurious reflections seen on a mesh file of INROPA.*



(b)

***Figure 2.1**: **(b)** blob type errors from spurious reflections seen on a mesh file of INROPA.*

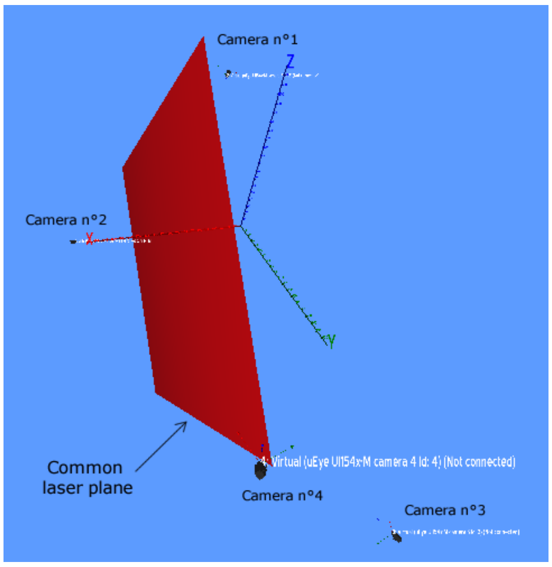
Easily detected by a human-being, those spurious peaks and blobs constituted a major problem for an automatic system. Indeed, the automatic teaching system cannot make the difference between an error and the real surface of the object. The INROPA system has to be entirely automatic and it has to reduce the programming of painting robot from several hours to only a few minutes. The software-based solution has to remove all errors, and this, in less than 30 seconds. It must have the possibility to be integrated in the software of INROPA which is coded in C/C++, OpenCV and DirectX [K. A. Nissum et *al*, 2010] [K. A. Nissum et *al*, 2005].

2.2. INROPA 3-D scanner description

INROPA (Intelligent Robot Painting) developed a 3D scanner based on the triangulation of a laser light. The solution includes a hardware set up and a software which utilizes computer vision techniques to create a 3D CAD model of the scanned object. This system is briefly described in this part to understand the context in which this project is.

2.2.1. Hardware

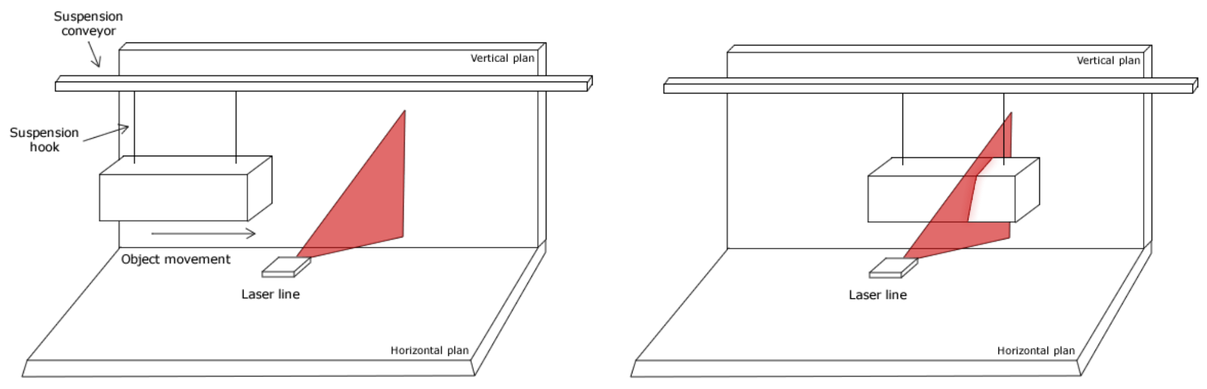
The hardware used by INROPA is a 3D laser scanner where the sensors are cameras and the light sources are laser lines. The complete set up is composed by 4 lasers lines and 4 cameras. Each one is link and calibrate depending on each laser line. The laser are calibrated and positioned in order to have their laser plane superposed which simplifies the calculation of the laser line plane location (figure 2.2(a)). By doing that, the laser light hits all the surface of the object to be scanned. Each camera is calibrated depending on each respective laser. Therefore, the whole object is viewed and scanned without moving either cameras or light sources.



(a)

***Figure 2.2**: **(a)** Calibration of the 4 cameras in order to have a common laser plane.*

In this system, the object is moving and the optical set up is fixed. Figure 2.2(b) illustrates it. The object scanned is the parallelepiped. It is transported using two suspension hooks attached to a suspension conveyor. It is translated horizontally to the laser scan line [K. A. Nissum et *al*, 2010]. Figure 2.2(c) shows how the object is hit by the laser line.



(b) (c)

Figure 2.2: ***(b)*** *Set up of the laser scanner: the object is moving horizontally to the laser scan line* ***(c)*** *the object is hit by the laser scan line.*

The object does not stop during the whole process of scanning and the software computes all ranges of points on the fly. This software is briefly described in the following section.

2.2.2. Software

The software of INROPA is coded in C/C++ and follows these steps:

- Images acquisition
- Images segmentation
- Triangulation
- Mesh construction

Theoretically, a video is captured and process automatically by the software, then all images are segmented. Range of all points is found by processing the triangulation step on all images which gives, as output, a cloud of points. This cloud of points is finally processed and the mesh of the object is stored in an STL file.

The process below is the most simple but it requires a computer which can store huge amounts of data. Indeed, video sequences are often in high resolution. They are therefore, quite heavy. Clouds of points computed from industrial objects which can easily reach millions of points, are also space consuming [K. A. Nissum et *al*, 2010].

Thus, the system doesn't store, neither video sequences, nor clouds of points. Each image is directly processed and the range of all points is triangulated into mesh on the fly.

2.3. Problem identification

Errors encounter in meshes are due to unwanted reflections. In this part, brief reflection rules are enounced to have a better understanding of the phenomena that provoke spurious peaks and blobs in the mesh. First, two different types of reflection are explained. Then, geometrical issues are described to show how the laser light can be reflected uncontrolled by the surface of an object.

2.3.1. Brief optical assessments on reflection

Shiny surfaces as polished metal or plastic reflect light in a mixed diffuse and specular way. A diffuse reflection can be encounter when the light strikes a matte or rough surface. The ray bounces off in all directions. The energy of the incident ray is thereby shared between all the diffused rays. Figure 2.3(a) shows how the light is reflected on diffused surface.

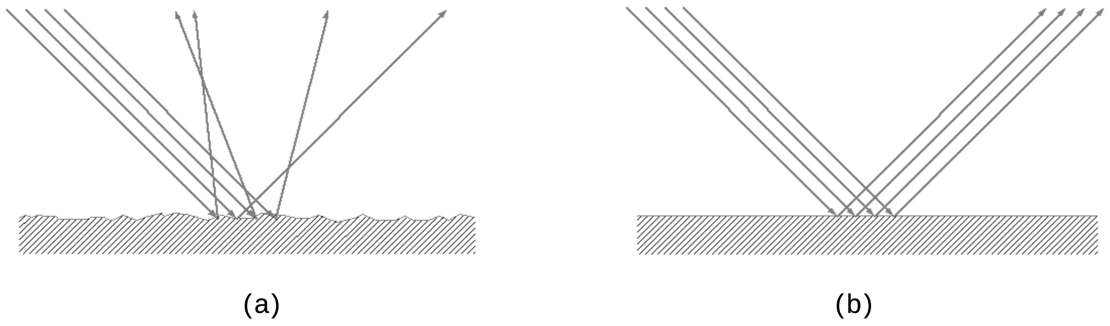
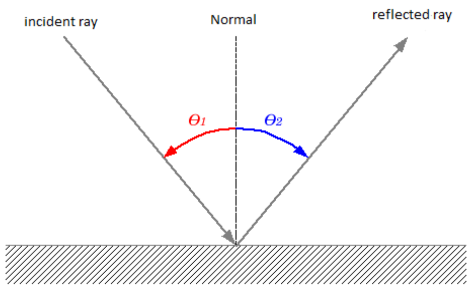


Figure 2.3: (a) *Representation of a diffused reflection.* ***(b)*** *Representation of a specular reflection.*

In opposite of that, the reflection is called specular when the incident ray gives a unique reflected ray. This phenomenon follows the 3 laws of reflection:

- The incident ray, the reflected ray and the normal to the reflection are in the same plane which is called the plane of incidence
- The angle between the incident ray and the normal is equal to the angle of the reflected ray and the normal (figure 2.3(c)).
- The light path is reversible

Two examples of these reflections: the screen of a cinema is a diffused surface and a planar mirror is a specular surface [Wikipedia, 2010].



(c)

***Figure 2.3**: **(c)** 2^{nd} law of reflection: θ_1 the angle between the incident ray and the normal θ_2 the angle between the reflected ray and the normal.*

The main difference of the specular and the diffuse reflection is the repartition of the energy between only one or multiple reflected rays. A polished metal or plastic object is considered as mixed diffuse and specular surface: most of the energy is given to a small amount of rays which are reflected to the same direction. The rest of the energy is then spread to others rays which diffuse the light (figure 2.3(d,e,f)) [I. Ihrke et *al*, 2008].

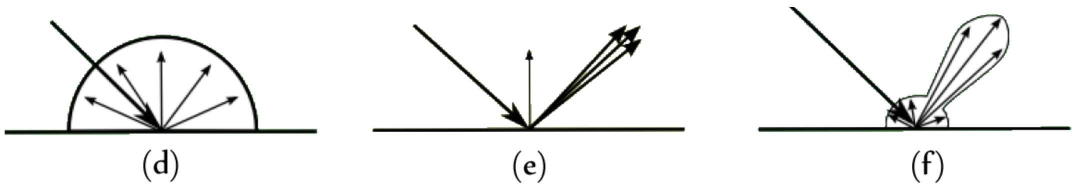
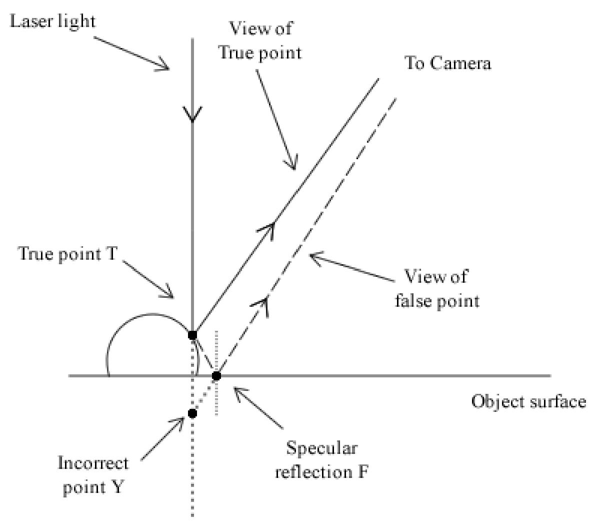


***Figure 2.3: (d)** Diffusal reflection, **(e)** Nearly ideal specular reflection, **(f)** Mixed diffuse and specualr reflection.*

2.3.2. Spurious reflections on a laser scanner

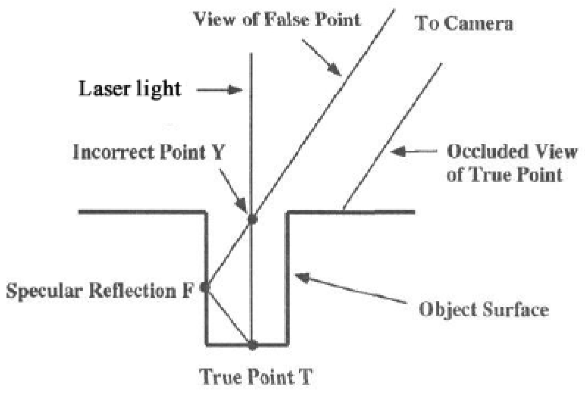
Even if industrial objects have often mixed diffuse and specular reflection surfaces, 3D laser scanner are nowadays enough robust to create a usable 3D mesh. However, problems of unwanted reflections can appear with some specific but common situation. Objects having concave, convex geometry are susceptible to give multiple reflections. In that case, sensors can be easily confused by noisy reflections [D. J. Svetkoff, 2000]. Figure 2.3(g) shows how a concave geometry in an object surface can produce some secondary reflections. Since the object is considered as mixed diffuse and specular reflection surface, the first specular reflection gives the view of the true point whereas, the second reflection which comes from one diffused ray of the laser light gives a false point Y.



(g)

***Figure 2.3**: **(g)** Concave geometry which can cause false range recording.*

Moreover, objects surfaces presenting holes and sharp edges can also be susceptible to the occurrence of spurious reflections since those geometries can produce total or partial occlusion [E. Trucco et *al*, 1994]. Figure 2.3(h) shows a rectangular hole in an object surface. The laser light hits the true point T inside the hole and is reflected. Two reflections can potentially arrive to the sensor: the 1st one, directly from the point T, and the 2nd one which is the result of a second reflection on one of the edge of the hole. The direct reflection from the true point T is occluded by the hole. Thus, the sensor only gets the second reflection from the reflection on the edge of the hole, on point F. Since all observed points must belong to the laser plane, the system records the incorrect point Y instead of the true point T.



(h)

***Figure 2.3**: **(h)** Rectangle hole on an object surface which can cause false range recording.*

2.4. Filtering methods description

In this section, four different methods to filter spurious reflections are briefly described. Each method works during different step of the scanning process seen in the section 2.2. These are presented in order to choose the most relevant techniques for the project which is then analyzed in the Part II.

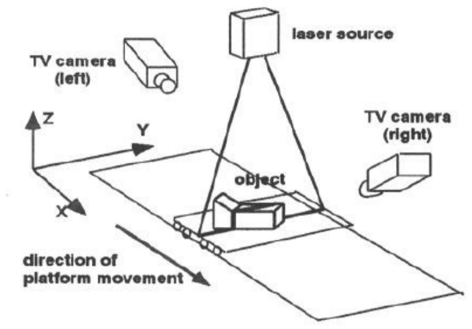
2.4.1. Hardware filter

Two methods are described in this part. They both use new hardware setups that have to be added to the basic laser range scanning.

Direct calibration method

This first method is set up before the acquisition of the image and is based on the fact that unwanted reflections occur depending on camera position. So the main idea is to put more than one camera for one laser source light. By doing that, two images can be produced and compared to find incorrect points [E. Trucco et *al*, 1994].

Figure 2.4(a) shows what can be the setup of this method. Two cameras are positioned to the left and the right side of the object. After calibrating them to have the same origin reference, the two cameras can produce two theoretically identical range images.



(a)

***Figure 2.4**: **(a)** Set up for the direct hardware filtering.*

To find the false range values, cloud of points from both left and right cameras must be compared. A true point must satisfy the following constraints:

- Since the source of light is not moving, the laser can only hit a surface once. So, for a set of one light source and two cameras, if more than one point belonging to the same laser plane are find in the cloud of point, all the points are deleted.
- If a point is observed by both cameras, then the range values $Z_L(t)$ and $Z_R(t)$ given respectively by the left and the right cameras must be equal considering a margin of error τ. More over the local surface normal $\vec{n}_p(t)$ of the point must be the same for both cameras considering a margin of error τ_2.

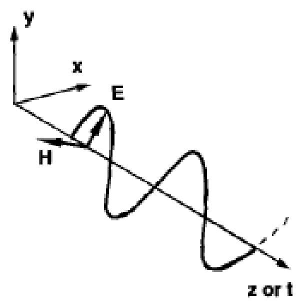
- If a point is only recorded on data cloud of one camera, it means, it exists another true point recorded by the other camera which could obscure the view of this point. Hence, any single point only existing on one cloud of points which are not obscured by another point from the other cloud of points are removed.

The benefits of this method are that most of the errors can be removed easily since if any point has different range from the right cloud of point to the left cloud of point, then they are removed. Moreover, the set up of two cameras for one source of light can help to record a full range image from all angles without additional source of light. The drawback is that this method requires calibration of two cameras. Besides, the processing of two clouds of points can be time consuming because industrial object meshes are usually made of millions of points.

Polarization filtering

This method uses a polarized laser light and measure the polarization state of the reflected light to distinguish the true laser line and the spurious reflections. Since this polarization value changes for each reflection on the object, this method can separate the true laser line and spurious reflections [J. Clark et *al*, 1996].

To have a better understanding of this method, a really brief recall of basic facts about light polarization is enounced. A ray of light is composed of the electric field waveform $\vec{E}$(t) and the magnetic field waveform $\vec{H}$(t). Those two waveforms are orthogonally to each other and they form a plan called the plane wave (figure 2.4(b)). Considering only the electric field waveform $\vec{E}$(t) to represent the light, this vector can be decomposed into two components E_x(t) and E_y(t). In an unpolarized light, E_x(t) and E_y(t) are non-deterministic which means that $\vec{E}$(t) can't be described except statistically. Whereas in a polarized light, E_x(t) and E_y(t) oscillate in a deterministic way, in phase or not, which leads to a linear or elliptical polarization.

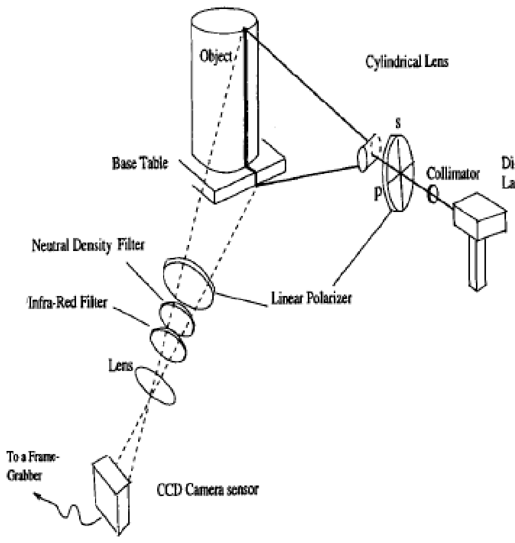


(b)

***Figure 2.4**: **(b)** Ray of light with the electric field and magnetic field waveform $\vec{E}(t)$ and $\vec{H}(t)$ in the case of a linear polarization.*

The technique consists in polarized linearly the laser source light and find the changes in all reflections gathered by the camera (figure 2.4(c)). To find the changes, the reflections pass through a linear polarizer before hitting the CCD captor of the camera. By doing this, the intensity of the resulting polarized light is a function of $(\theta - \beta)$ called the transmitted radiance sinusoid (TRS) where θ is the orientation of the incident polarized light and β, the orientation of the linear polarizer. The system calculates the TRS for each pixel of the laser line reflection and compares values between them [J. Clark et *al*, 1996].

However, even if polarization status between the main reflection and the unwanted reflection are different, the polarization value of the main reflection is not constant and changes according to the shape of the object. To predict the polarization of the main reflection, the system is based on the Fresnel reflectance model. Since the orientations of unwanted reflections are very different from the orientation of the primary reflection, the system can identify the true laser line reflection [J. Clark et *al*, 1996].



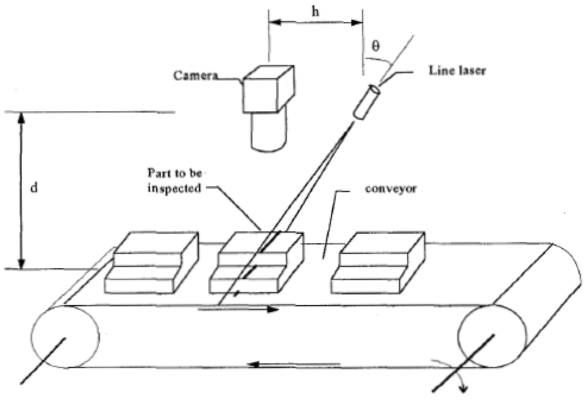
(c)

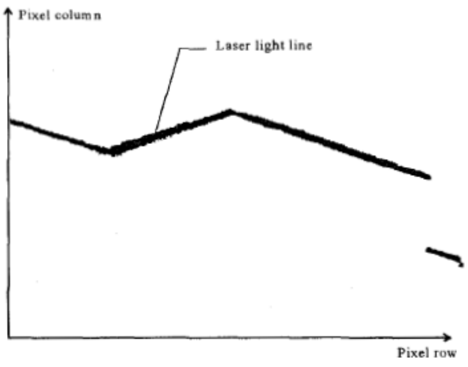
Figure 2.4: ***(c)*** *Set up of the polarization filter method.*

The benefits of this method are that the system not only rejects all inconsistent range values but identifies the true laser line projected on the object. This allows the process to work directly on the direct captured image without waiting a rending of the cloud of points. The drawbacks are that the setting up of filter and the calibration of them can be time consuming and hard to do it automatically. This method depends also on the accuracy of the Fresnel reflectance model which is applicable only for smooth surface which means that the result depends mostly on the physical properties of the scanned object and not the geometry.

2.4.2. Segmentation filtering

In this method, there is no need to change the basic set up of the 3D laser scanner. It is software based and the algorithm works on the segmentation part. The first step of this method consists on finding the center of the center of the peak of the reflected laser line. Considering the set up figure 2.4(d), which is a basic 3D laser scanner, after the segmentation step, the resulting image at the time *t* looks like the figure 2.4(e). There are no spurious reflections on this sample and the laser line can be easily distinguished.

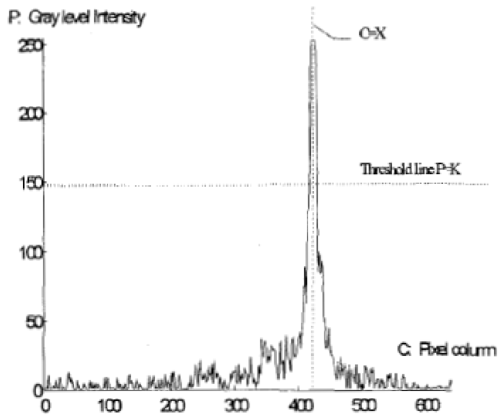




(d) (e)

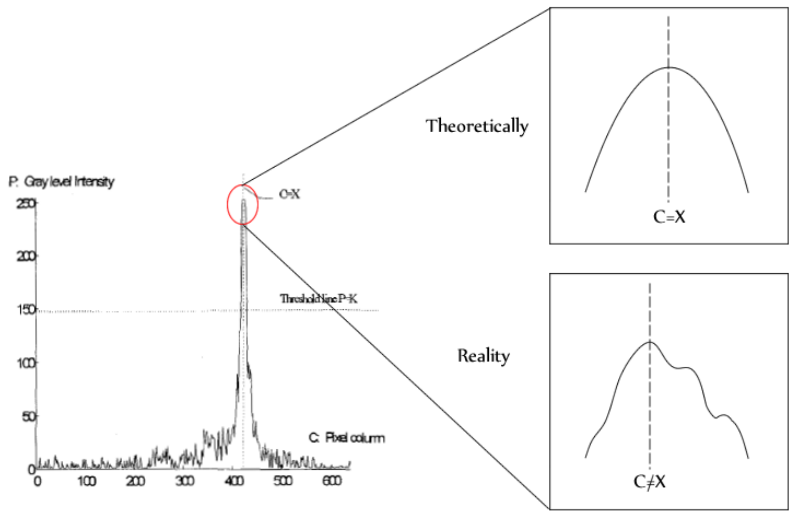
***Figure 2.4**: **(d)** Basic laser scanner **(e)** segmented image at the time t.*

In calculating the 3D coordinates, a histogram of the gray level intensity for each pixel in a single column is made (figure 2.4(f)). The idea is to find the center of the maximum peak which represents the center of the laser line. Theoretically, a laser beam can be considerate as a punctual source of light or at least the distribution of the light is a Gaussian one. Therefore, the center of the maximum peak is the center of the laser line. However, the distribution is not exactly a Gaussian one because of several factors, for instance multiple reflections can cause that (figure 2.4(g)) [J. Chen et *al*, 1998].



(f)

***Figure 2.4**: **(f)** Gray level for each pixel in a single column of the segmented image.*



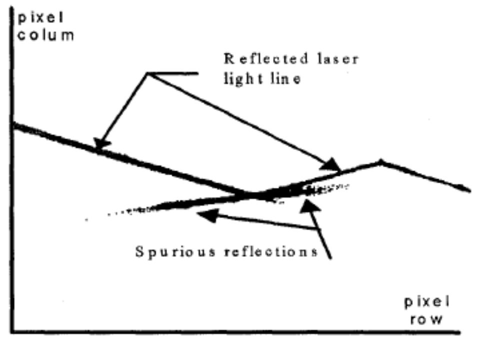
(g)

***Figure 2.4**: **(g)** Non Gaussian distribution.*

Thus the first step of the method is to run a new algorithm, more efficient, to determine the center of the center of the peak of the reflected laser line. To simplify the calculation, a threshold is chosen P=K (figure 2.4(f)).

This algorithm is based on finding the pixel which divides the total integral area *S* by two. First, the algorithm finds the point (X_i,Y_i) where $S_{Li} < \frac{S}{2}$. S_{Li} is the integral area from the left side of (X_i,Y_i). Then, it finds the point (X_{i+1},Y_{i+1}) which is the second next pixel from (X_i,Y_i) and where $S_{Li+1} > \frac{S}{2}$. The pixel (X,Y) between (X_i,Y_i) and (X_{i+1},Y_{i+1}) is then the center of the center of the peak of the reflected laser line. With this first step, the accuracy of the scanning is improved by 2.5 [J. Chen et *al*, 1998].

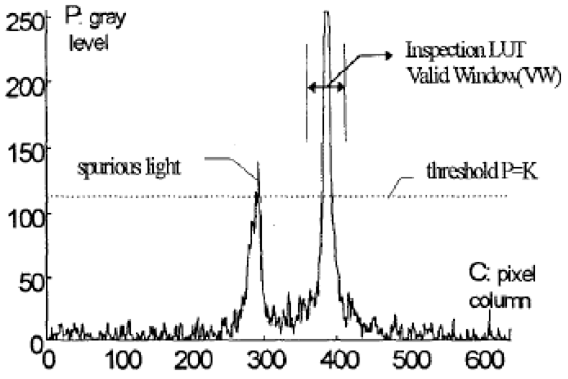
The second step is to remove spurious reflections. Figure 2.4 (h) shows the resulting image at the time *t* with spurious reflections. Then, a histogram of the gray level intensity for each pixel in a single column is made (figure 2.4(i)). In this case, two distinct peaks can be seen. According to reflection assessment of part 2.3.1, the lowest peak comes from spurious light since the intensity of the light from secondary reflections is lower than from the main reflection. The first algorithm cannot now be applicable since it is based on the total integral area *S* [J. Chen et *al*, 1998].



(h)

***Figure 2.4**: **(h)** segmented image at the time t with spurious reflections.*

Two solutions can be then considered: adapt the threshold P=K according to the gray level of the spurious reflection or only use the algorithm on the region near the maximum peak. The first solution cannot be used since the gray level of the spurious light cannot be characterized by a deterministic way. The region near the maximum peak is called the Valid Window (VW) thereafter [J. Chen et *al*, 1998].



(i)

***Figure 2.4**: **(i)** Gray level for each pixel in a single column of the segmented image with spurious reflections.*

So the whole method starts with the search of the maximum peak to determine the VW. Then, the algorithm to determine the center of the center of the peak of the reflected laser line is run as seen before.

The benefits of this method are that there is no need to modify the setup of the laser scanner. Moreover, the algorithm is working only on the VW which means that the number of processes is relatively small for each column. The drawbacks are the undeterministic gray level of the spurious reflection light. Thus, there is a possibility that the system doesn't delete all the errors due to spurious reflections. Besides, since the algorithm is run for each column of each image of the video, this method can be very time consuming and process consuming too.

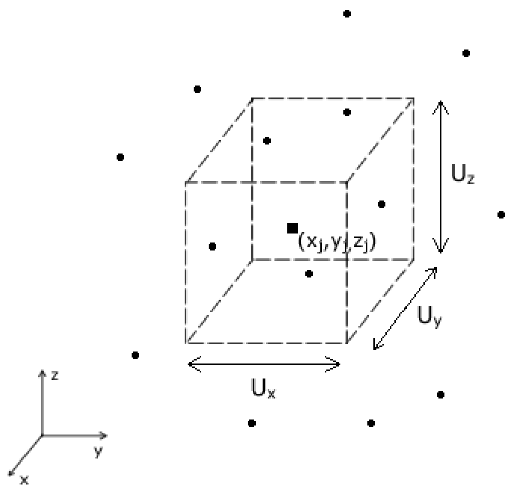
2.4.3. Mesh filtering

The last method uses the 3D data cloud density. The algorithm is run after the recording of all the 3D points and before the tessellation and the creation of the final 3D mesh. The density of a 3D point, according to [D. Yang et *al*, 1999], is the number of measured points in the close vicinity of that point. Considering the 3D point $P_j(x_j,y_j,z_j)$, the data cloud density of P_j is all the 3D points restrained inside a parallelepiped [Ux,Uy,Uz] where $P_j(x_j,y_j,z_j)$ is the center of the parallelepiped (figure 2.4(j)).

For the point $P_j(x_j,y_j,z_j)$:

The data cloud density of P_j is: $$DS_j = \left\{ P_i(x_i,y_i,z_i) \middle| \begin{array}{c} (x_j - U_x) < x_i < (x_j + U_x) \\ (y_j - U_y) < y_i < (y_j + U_y) \\ (z_j - U_z) < z_i < (z_j + U_z) \end{array} \right\}$$

The density of $P_j(x_j,y_j,z_j)$ is then: $\text{Density}_j = \sum(P_i \in DS_j)$



(h)

***Figure 2.4**: **(h)** Graphical representation of the data cloud density of a 3D point* $P_j(x_j,y_j,z_j)$.

The errors due to unwanted reflections are points or blobs of points more or less far from the object as we identified them in the problem statement part 2.1.1. Depending on the thresholds U_x, U_y and U_z, a spurious point's density value is lower than the real points' one. With this assertion, errors from unwanted reflection are easy to detect and delete.

The benefits of this method are the fact that the algorithm is run after the capture of all the 3D points which means it can be run only once. It is also an only-software based solution and thus, it is not affected by the calibration, the characteristic of hardware used and the setup of the 3D laser scanner since all the distances computed are relative to the point $P_j(x_j,y_j,z_j)$. The drawback is the fact that the algorithm needs to compute all the cloud of points which, in the case of an industrial object, can be very heavy: around 2 millions of points. Therefore, the method can be very time consuming.

2.5. System requirements

In the previous chapter, several techniques are described according to the initial requirements from INROPA. In this section, one technique is chosen for further study and implementation following the new inherent requirements.

The first two methods are hardware based techniques. Even if they can provide a really good accuracy concerning the detection of spurious reflections errors, both need the set up of new hardware on INROPA 3D laser scanner. First, the price of the solution can be considered to be too expensive, secondly, partial or all INROPA system have to be modified to be adapted to the new setup.

Two methods can therefore be considered: the segmentation and the mesh filtering since both are only-software based. Moreover, they can be both implemented in C/C++ and then easily integrate in INROPA software. However, these methods can be really time consuming considering the size of an industrial object.

Comparing the errors identified in the problem statement part 2.1.1, the mesh filtering can provide a better result and a more robust solution than the segmentation filtering. Anyway, the system must be capable of the following:

- Computing a file .xyz of at least 2 millions points.
- Finding all errors and delete them without destroying the shape of the mesh.
- Creating a new file .xyz of the cloud of points without errors
- Finishing all process in less than 30 seconds.

To have a better understanding on how the mesh filtering method can fulfill these requirements, the basic 3D laser scanner process and methods to achieve each requirement are described and analyzed in detail in the following part.

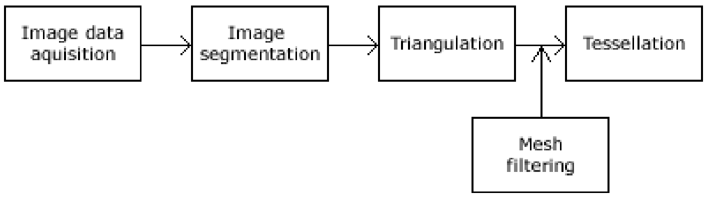
Part II

Analysis

3. Analysis

In this part, steps to create a 3D model are described in order to understand how the mesh filtering method can work to delete the errors from spurious reflection. Camera calibration and camera setup are not analyzed since it is not the point of this paper to deal with hardware setup.

As said below, before the steps to create a 3D model illustrated figure 3(a), camera calibration and camera setup steps are needed because the system must determine an origin to calculate the range of 3D points. Moreover, many parameters as the shape of the scanned object or distortion from the camera can interfere into the image data acquisition. These variations can be controlled in the camera setup.



(a)

***Figure 3**: **(a)** Steps to create a 3D model including the filtering step to delete errors from unwanted reflection.*

Concerning the construction of 3D CAD model steps, first, the direct image is captured in the Image data acquisition step from the cameras. Then, the image preprocessing approach to determine the scan line location in the Image segmentation step is then described. Those two steps can be defined as the acquisition and the identification parts.

After getting raw data from cameras, the system needs to compute them in order to determine range of each 3D point and create the final 3D model. Those parts are named triangulation and tessellation or mesh construction and can be defined as the processing and the creation parts.

Between the triangulation and the tessellation, the chosen method to remove unwanted reflections errors is applied in the Mesh filtering step as shown in figure 3(a). This method is described in detail in the last part of the analysis and will be implemented in the following chapter.

3.1. Three-dimensions model creation

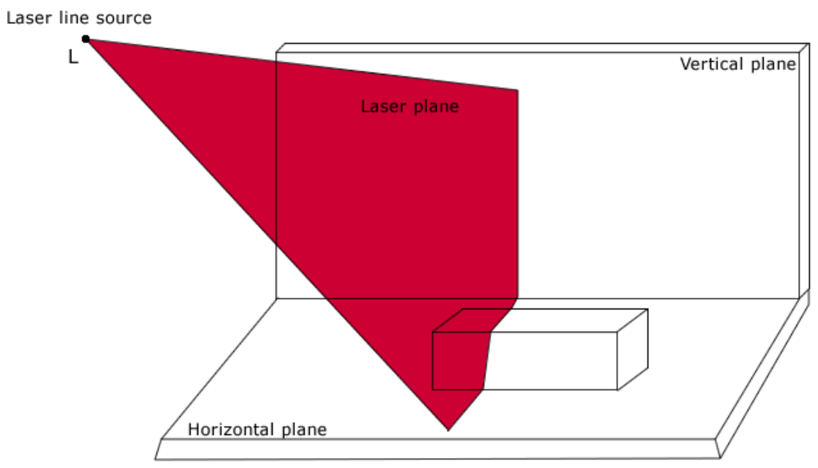
In this section, the creation of a 3D model is described step by step to understand how the mesh filtering method can work to delete the errors from spurious reflection. These steps include the Image segmentation step, the triangulation and the tessellation (figure 3(a)). The image data acquisition is not described in this paper since this step only depends on the software choice.

3.1.1. Image segmentation

In a 3D laser scanner, the first part is to determine the scan line location from the raw image given by the camera. There are two options concerning the scanning of an object by a laser scanner: the object can be stationary and the laser line plan is moving or the opposite. In both case, the system has to be able to define three distinct zones which are, the vertical plane, the horizontal plane and the object. Those planes need to be identified in order to know where is the laser line and therefore, to be able to triangulate the position of 3D points.

Identification of the laser plan

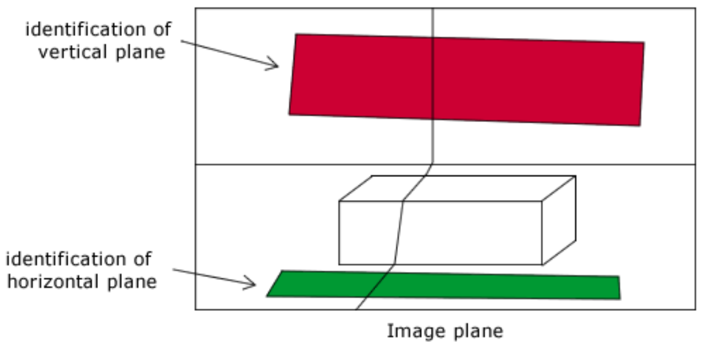
On a 3D scene, the different planes can be easily distinguished. However, the system has to define the three planes on the 2D images given by cameras. Figure 3.1(a) shows the different planes from a basic setup of a laser line scanner. The laser line source which creates the laser plane hits the scene. This creates two unchanged lines, one on the vertical plane and the other one on the horizontal plane.



(a)

***Figure 3.1**: **(a)** Representation of the different plane on a laser scanner scene.*

The projected laser line can be defined in three areas: the vertical area, the horizontal area and the object view. When those areas are detected, the orientation and the direction of the laser line reflected on the vertical area and on the horizontal area have to be detected. This operation will permit to estimate the orientation of the scan plan and then to find the range of points via the triangulation step. Since the intersection of the laser plane with the vertical plane and with the horizontal plane stay unchanged as long as the object doesn't pass through, the system can define both planes and characteristics of intersection lines on a 2D image. In figure 3.1(b), the area in red corresponds to the vertical plane and the green one corresponds to the horizontal plane. Those two parts will remains unchanged all along the scanning of the object. The area in between these two planes is assumed to contain the object.


identification of vertical plane
identification of horizontal plane
Image plane

(b)

***Figure 3.1**: **(b)** Identification of the horizontal and vertical plane on the 2D output image.*

After finding characteristics of the reflected laser line on vertical and horizontal plane, the system can isolate the object reflected laser line. In the following part, basic edge detection is explained. Then, the practical method to isolate the laser line is described, and a preprocessing technique to find the coordinate of the laser line reflected on the object is explained.

Edge detection principle

In this part, the general edge detection principle is described and then applies on the segmented image of the laser line previously obtained.

The edge detection is based on the detection of pixels corresponding to an abrupt changing of the intensity in a gray level image. The main method to find these pixels is to find maxima of the first-order derivative expression or the local directional maxima of the gradient magnitude. For a numeric image, each pixel is associated with a value between 0 to 255. In calculus, the first-order derivate is a measure of how a function changes as its input changes. In this case, the first-order derivate is simplified because the edge detection only makes a comparison between the derivates without taking in account their value. Thus, the first-order derivate for a pixel is the difference between the values of both adjacent pixels. Figure 3.1(c) shows a grid of pixels *a* to *i*.

a	b	c
d	e	f
g	h	i

(c)

***Figure 3.1**: **(c)** Pixel grid containing 9 pixels: a to i.*

The first-order derivates of the pixel *e* are:

- D_{xe} = f - d
- D_{ye} = b - h

In this view, three main methods can be described: the Prewitt edge detection, the Sobel edge detection and the Canny edge detection [Wikipedia Edge detection, 2010].

The Prewitt edge detection is the most basic filter used to detect edge from a numeric picture. This technique utilizes two kernels 1x3 and 3x1: one horizontal h_x = $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ and one vertical h_y = $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ with gives to table Gx and Gy. These tables are then combined to give a common table G, filled with the first-order derivate of each pixel of the image, where the system can find maxima and therefore, edges pixels.

The Sobel edge detection is more efficient than the Prewitt model. It uses two kernels of 3x3 instead of kernels of 1x3 and 3x1: one horizontal h_x = $\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$ and one vertical h_y = $\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$. The convolution of the vertical and the horizontal kernels with the source image give two tables G_x and G_y. With these data, the gradient's direction is:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

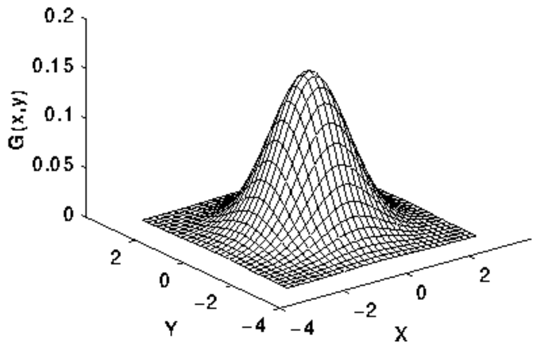
The main problem of these two methods is that they are very sensible to the noise. The Canny edge detection is an evolution of the Sobel edge detection. Indeed, it uses the same kernel as the Sobel edge detection but before using it, the Canny edge detection uses a Gaussian filter to remove noises. Then a hysteresis thresholding is applied to refine the edge tracing.

A 2D isotropic Gaussian filter has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where σ, is the standard deviation of the distribution. Figure 3.1(d) shows a 2D Gaussian distribution with σ = 1. This is the representation of the following 5x5 Gaussian filter *B* with σ = 1:

$$B = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$



(d)

***Figure 3.1**: **(d)** 2D Gaussian distribution with σ = 1.*

Usually, the size of the kernel filter is smaller than the filtered picture and the bigger the filter is, the less it is sensible to noise.

After removing noises, the Canny edge detection finds the intensity gradient of the image with the same operator as the Prewitt and the Sobel edge detection and detects the edges by keeping maxima of gradient intensity values. However, the only intensity is not sufficient to decide whether a point corresponds to an edge or not. To be more accurate, the system has to compute the gradient's direction as in the Sobel edge detection:

$$\theta = \arctan(\frac{G_y}{G_x})$$

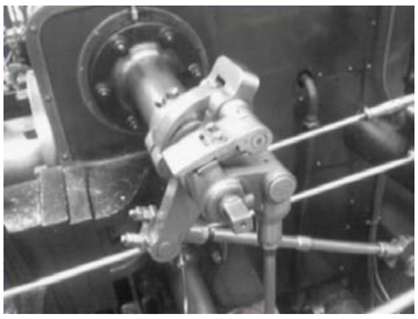
With the gradient's direction, maxima can be characterized as local or not. For instance, if θ = 0° then, the point will be considered to be on the edge is its intensity is greater than the intensity of the adjacent north and south pixels.

Finally, a hysteresis thresholding is applied on the resulting image. This requires two thresholds: one high and one low.

For all pixels, if his intensity gradient is:

- Lower than the low threshold, the pixel is rejected
- Higher than the high threshold, the pixel is considered as it belongs to an edge
- Between the two threshold, the pixel is considered as it belongs to an edge only if it is connected to one pixel already valid

At the end of the process, the output of this method is a binary image where there are pixels which belong to an edge and the others [Wikipedia Edge detection, 2010]. For example, you can see figure 3.1(e) the input image of a valve and the result after a Canny edge detection figure 3.1(f).


(e)

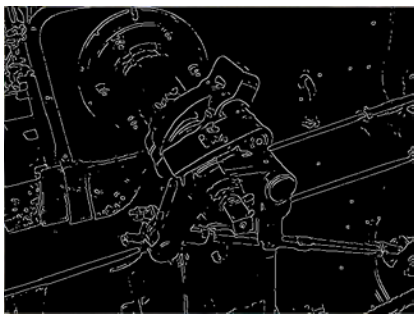

(f)

***Figure 3.1**: **(e)** Input image before the edge detection **(f)** resulting image from a Canny edge detection.*

In theory, edge detection can be applied to detect the laser line reflection but this is a really complex process. A more appropriate method is described in the following section.

Laser line location method

In the analysis, the object scanned is a metal beam with the same shape as the one shown figure 3.1(g).

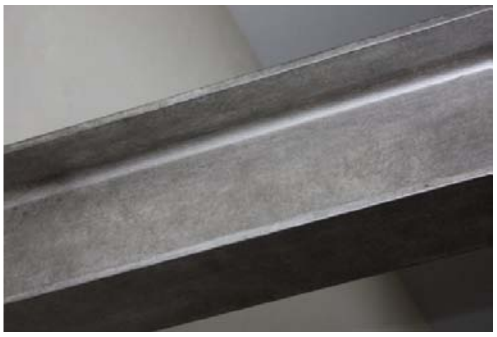

(g)

***Figure 3.1**: **(g)** Shape of the metal beam used in the analysis.*

The method consists on the comparison between two images at a time. The first image is captured without the object: only the background with the reflected laser line on the vertical and horizontal plane (figure 3.1(h)). This image is also used to define the laser plane location. The second image is the one where the system must locate the laser line edge (figure 3.1(i)). The subtraction of both images isolates the laser line reflected on the object (figure 3.1(j)). Then segmentation is applied with a binary transfer function using a threshold to remove noise and to prepare the image for the calculation of the laser line location [K. A. Nissum et *al*, 2005].

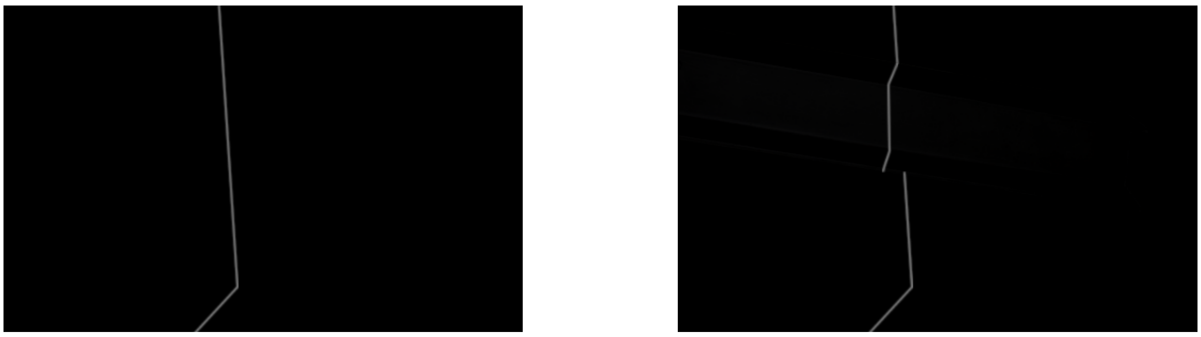

(h) (i)

***Figure 3.1**: **(h)** Image of the background **(i)** Image to be segmented.*

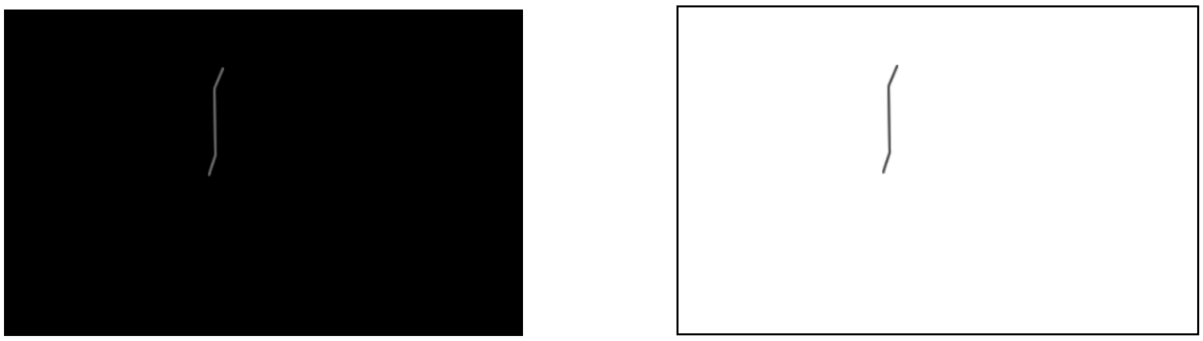

(j) (k)

***Figure 3.1**: **(j)** Subtraction between (h) and (i) **(k)** Segmented image ready for the edge detection.*

At this step, the laser line location can be found using the segmented image figure 3.1(k). Since there are only two type of pixel, black or white, which compose the segmented image, the calculation of the location can be done by using a kernel. Figure 3.1(l) shows three different kernels. Pixel in gray with the value of 0 is the one which is examined. The first one is a 1x2 kernel which considers any black pixel which has the left pixel in white as it belongs to an edge. In the three examples, all pixels scanned belong to an edge but it is obvious that this kernel only find pixel from the left side of edges. The second one is a 1x3 kernel. This one is a little more advanced than the first but still only detects pixels from the left side of edges. The third kernel is a 3x3 mask with the following restrictions: only one of the pixels 4 and 6 in the kernel must be black, and at least half of the 8-connected pixels must be black. This kernel is more advanced compare to the first ones but all three kernels are only usable for locating non horizontal scan lines since restrictions are put on the horizontal connected pixels. Other kernels can be used but with restrictions on the vertical connected pixels [K. A. Nissum et *al*, 2005].

This last method is rapid and simple and gives a usable location of the laser line but it depends on the setup and the quality of the segmentation. Another method can be perform on the segmented image and was described in the 2.4.2 part using the centroid algorithm.

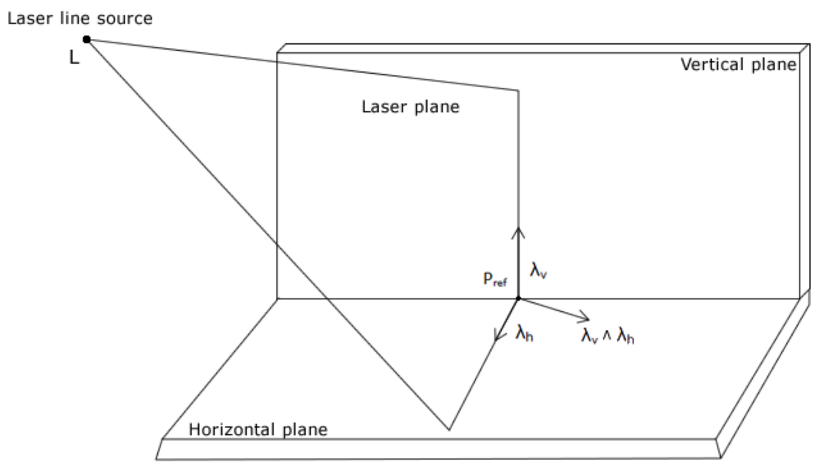
3.1.2. Triangulation

After segmented the image, the system has to determine the coordinates of points from the scan line reflected on the object. These coordinates have to be identified according to the reference coordinates from the camera calibration. To do that, the scan plane has to be estimated and the light source calibrated. Both of these two steps are liked. That means if the scan plane is successfully estimate, the light source is, in the same time, calibrated. One method to estimate the scan plane is described in the beginning of this section. It is then, followed by the description of the principles of geometric triangulation.

Scan plan estimation

In chapter 3.1.1, the method to identify the vertical plane and the horizontal plane is enounced. The scan lines corresponding to both horizontal and vertical plan can therefore be found through the image plane figure 2.4(h) using an appropriate kernel. Two points are then picked from each of the scan lines in order to transform them into 3D vectors λ_h and λ_v. This process is done with the world coordinate found with the calibration of the camera.

To estimate the scan plane, the system needs a direction and a reference point. The cross product of λ_h and λ_v gives the normal vector of the plane. The intersection of λ_h and λ_v gives the reference point P_{ref} (figure 3.1(l)) [K. A. Nissum et *al*, 2005].

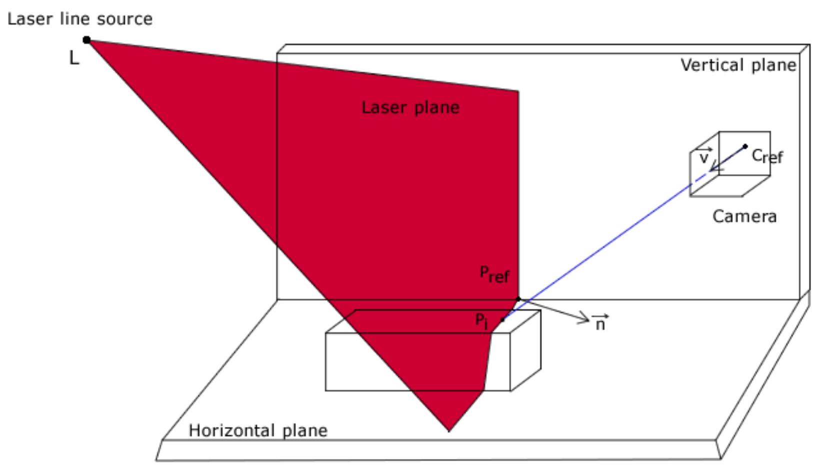


(l)

***Figure 3.1**: **(l)** Characterization of the laser scan plane.*

Optical triangulation

The 3D laser scanner is based on optical triangulation. The goal of this step is to find the coordinates of a given pixel Pi, intersection between the scan plane and the ray from the center of the camera. Figure 3.1(m) shows the intersection between the ray from the center of the camera and the scan plane.



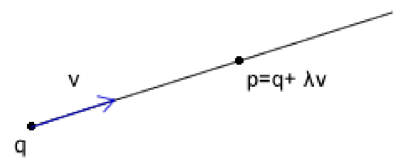
(m)

***Figure 3.1**: **(m)** Intersection between the ray from the center of the camera and the scan plane.*

A ray in 3-dimensions coordinates is represented as following:

$$p = q + \lambda v$$

Where p is a given point which lies on the ray, q is the reference point, λ is a positive parameter and v is the direction of the ray (figure 3.1(n)).



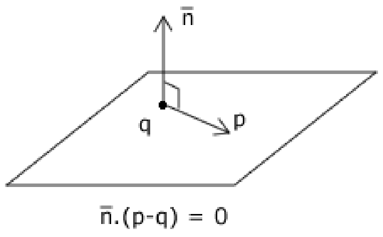
(n)

***Figure 3.1**: **(n)** Representation of a line in 3D coordinates.*

A plane in 3-dimensions coordinates is represented as following:

$$\bar{n}.(p - q) = 0$$

Where p is a given point which lies on the plane, q is the reference point and $\bar{n}$ the normal vector of the scan plane (figure 3.1(o)).



(o)

***Figure 3.1**: **(o)** Representation of a plane in 3D coordinates.*

Using the notification of the figure 3.1(m), the equation of the laser plane is:

$$P = \{p \mid 0 = \bar{n}.(p - p_{ref})\}$$

The equation of the ray from the center of the camera is:

$$L = \{p \mid p = c_{ref} + \lambda v\}$$

The intersection of these two occurs when:

$$\bar{n}.\left(c_{ref} + \lambda v - p_{ref}\right) = 0$$

And the solution of this equation for λ is:

$$\lambda = \frac{\bar{n}.(p_{ref} - c_{ref})}{\bar{n}v}$$

With that parameter, the intersection p_i is then equal to:

$$p_i = c_{ref} + \frac{\bar{n}.(p_{ref} - c_{ref})}{\bar{n}}$$

The normal vector $\bar{n}$ and p_{ref} are known from the scan plan estimation and c_{ref} is known from the calibration of the camera. Thus, it is easy to calculate p_i [D. Lanman et *al*, 2009].

In INROPA system, the laser and the optical set up is stationary which means that all points computed with the optical triangulation method will remains in the same plan. To correct that, each point has to be translated for each scanned image. This translation depends on the frame rate and the velocity of the conveyor. Indeed, the distance between each image, and so between each point in the direction of the conveyor movement is equal to:

$$D = \frac{v_c}{fr}$$

Where *fr* is the frame rate and v_c is the velocity of the conveyor.

Starting for the first point, each next point has to be separated from the last point of the distance D. So if the first frame in which the scan plane intersects the object is taken as the origin, for a given point p_j of the jth frame, the translation is equal to:

$$D_j = \frac{v_c}{fr}.j$$

The direction of the translation depends of the direction of the conveyor. Since this movement is a straight horizontal line, it is easy to find the direction of the translation [K. A. Nissum et *al*, 2005].

3.1.3. Mesh construction

The last step is to link all the points from the cloud of points to make a mesh and create the 3D object. This is called the tessellation. One of the most fundamental structures in computational geometry is the Delaunay triangulation. In this part, the theory of Delaunay is presented, followed by the practical application of this method.

Theory of Delaunay triangulation

The definition of the Delaunay triangulation of a set of points is a triangulation where no point of the set of points is inside the circumcircle of any triangle of the triangulation [Wikipedia D. triangulation, 2010]. Unlike others method such as the adjacent pixel tessellation, the Delaunay triangulation is possible on sets of points which are not in a fixed grid. It is based on the Voronei diagram.

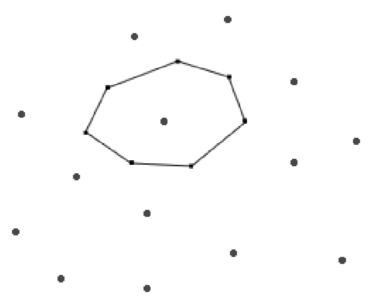
The Voronei diagram is defined as the union of all Voronei polygons. Considering *S* a set of *N* points randomly positioned on the plan *P* (figure 3.1(p)). A Voronei polygon *Vor(p)* associate to a point *p* of *S* is all points in the region of the point *p* which are closer to the point *p* than to any other point in *S*:

$$Vor(p) = \{x \in P \mid \forall\, q \in S, d(x,p) \leq d(x,q)\}$$

The Voronei polygon is illustrated figure 3.1(q) and the Voronei diagram is illustrated figure 3.1(r).

(p)

***Figure 3.1**: **(p)** S Set of N points.*


(q)

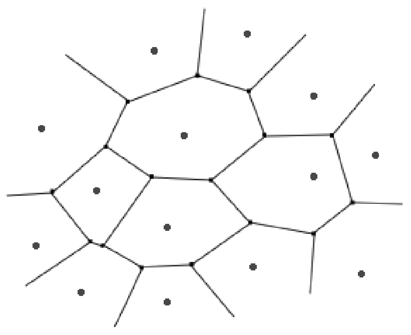

(r)

***Figure 3.1**: **(q)** Voronei polygon **(r)** Associated Voronei diagram.*

The Voronei diagram follows three properties:

- Each vertex of the Voronei diagram is the intersection of exactly three Voronei edges.
- For each vertex, the circumcircle of the associated triangle doesn't contain any other point of the set of points.
- Each Voronei edge is shared by exactly two points.

Each Voronei polygon is then associated to one vertex of the Delaunay triangulation. Delaunay vertexes are connected only if their associated Voronei polygons are neighbors. The Delaunay triangulation is the straight-line dual of the Voronei diagram (figure 3.1(s)).

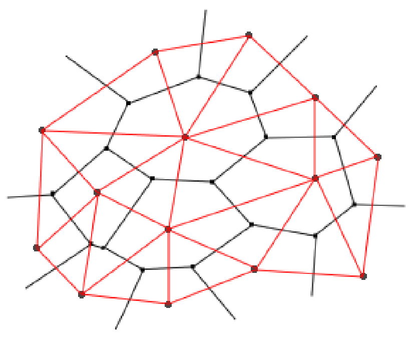

(s)

***Figure 3.1**: **(s)** In red, the straight-line dual of the Voronei diagram (r).*

To the end, the triangulation of Delaunay is unique; it is a complete triangulation and has the following property: the circumcircle of each triangle doesn't contain any other point [K. Chakib, 1999].

Practical approach of the Delaunay triangulation

Two practical methods are described in this part. Both of them are based on Voronei diagram properties.

The first method uses a FIFO list to determine the order of the creation of triangles. Figure 3.1(t) depicted the basic process to create a triangle from the set of points. The first edge of the triangle is created by connecting the two points which lie closest to one another. The third point has to follow two properties of a Delaunay triangle:

- The three points has to lie on the same circle: the circumcircle of the triangle.
- No other points than the three vertexes have to be inside the circumcircle of the Delaunay triangle.

The first step is to find a point that lies on the same circle as the two first points of the first edge (figure 3.1(t)(iii)). Then, the algorithm determines if there is one co-circular point inside the range of the first circle. If there is, this point become the new third point of the Delaunay triangle (figure 3.1(t)(iv)). This loop is running until no other closer co-circular point is found. Then the triangle is made (figure 3.1(t)(vi)).

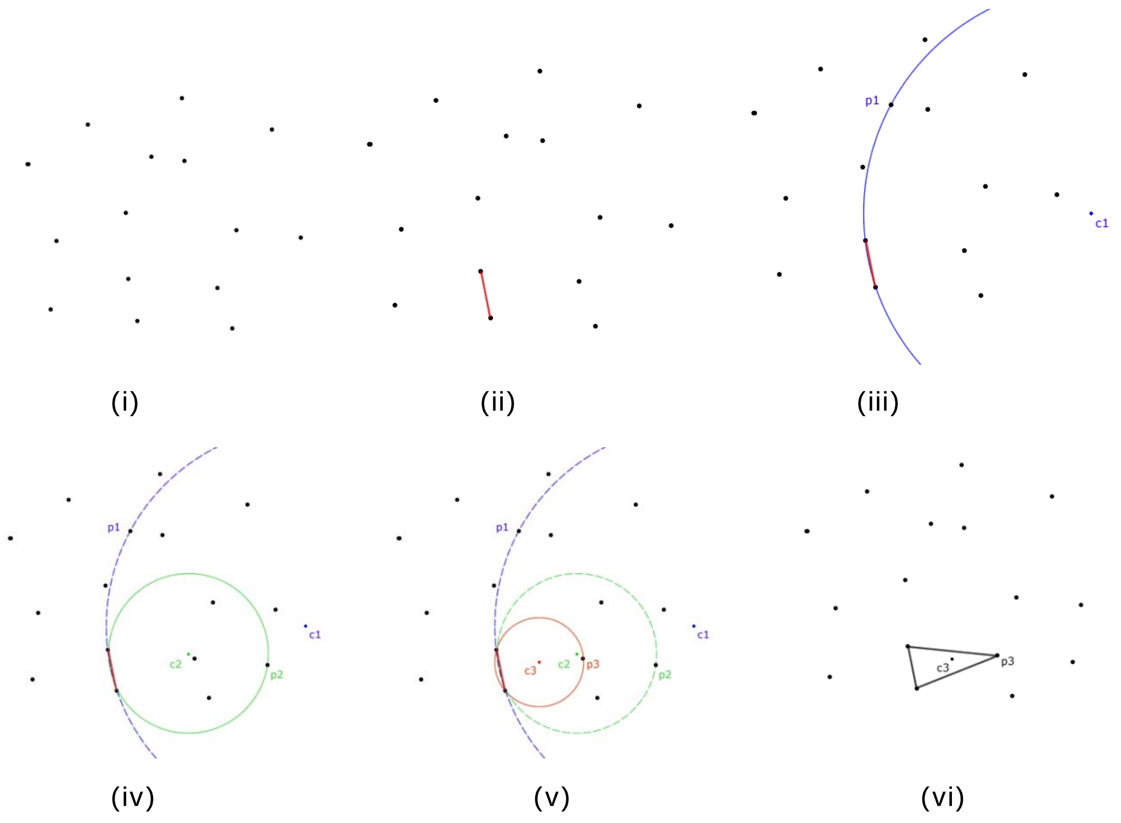
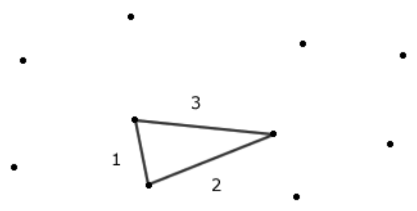


Figure 3.1*:* ***(t)*** *Creation of the first triangle in a set of points. (i) is the set of points, the first edge is made (ii), then the first third point is found (iii). A second third point is found and replaces the first one (iv). The loop continues till there is no other closer co-circular point (v). Then the triangle is made (vi).*

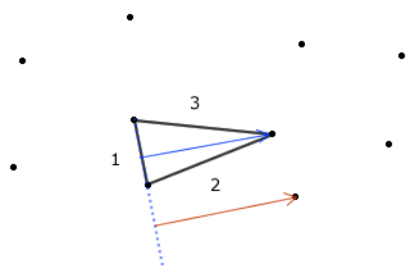
Now that the first triangle is created, the three edges of this triangle become the starting point of the creation of the others triangle. This method uses a First-In-First-Out (FIFO) list to determine which edge is used to create the next triangle. Figure 3.1(u) shows the order given to edges of the first triangle.



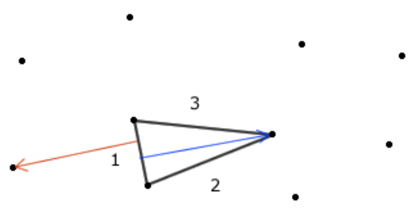
(u)

***Figure 3.1**: **(u)** Order given to edges of the first triangle.*

To create the other triangle, the process to create the first triangle is used again. However, in order to avoid that the edges of the next triangle cross the first ones another process has to be add. Figure 3.1(v)(i) shows a potential third point for the next triangle. The vector from the starting edge to this point is then compared to the vector from the first triangle. If both vectors point in the same direction, then, the potential third point is rejected. The case that the potential third point is accepted is pictured figure 3.1(v)(ii).



(i)



(ii)

***Figure 3.1**: **(v)** (i) the potential point pointed by the red vector is rejected since both red and blue vectors are pointing in the same direction. (ii) shows the case where the potential point is accepted.*

After the creation of the second triangle, the first edge is deleted from the FIFO list and the second edge becomes the first one. However, there are two particular cases:

- When the creation of the next triangle edges includes one already created edge
- When no acceptable points exist

In the first case, illustrated figure 3.1(w), the creation of the triangle has to connect only two points. Indeed, only one edge is needed to complete the triangle. As a consequence, the already created edge has to be deleted from the FIFO list in order to not created duplicate triangles. Therefore, every time a triangle is created, it is required to read the whole FIFO list and detect the already made connections.

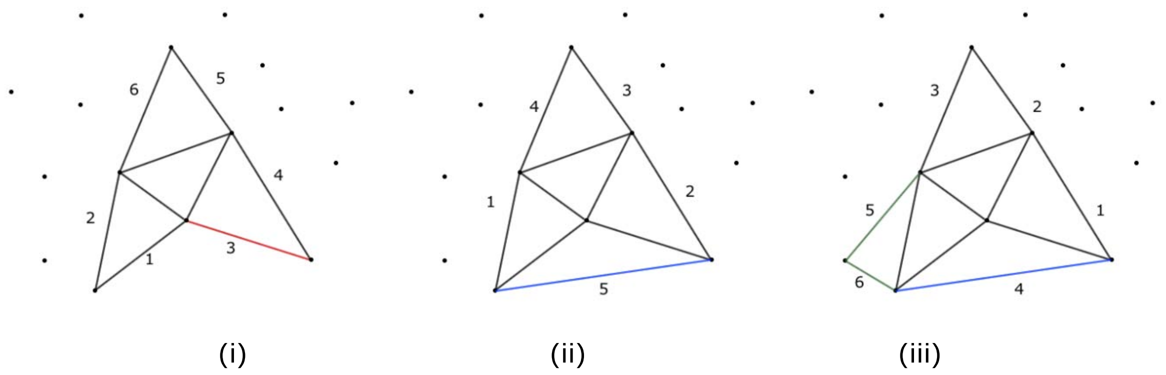


***Figure 3.1**: **(w)** (i) Edge processing (ii) Existing edge (iii) new edge processing*

In the second case, if there is no acceptable point for the current edge, for example, near the border of the cloud of points, the edge is removed from the FIFO list without the creation of any new edges.

When the FIFO is empty all edges have been processed and the Delaunay triangulation of the cloud of points is complete [Preparata & Shamos, 1985].

The second method transforms the 3-dimensional problem in a 2-dimensional problem. Indeed, instead of creating triangle using a FIFO list to sort them, this second method scan in only one direction the points from the set of points. In figure 3.1(x), the green line represents the scanning line. When the scanning line encounters a point, the system tries to connect this point with the others points already scanned. To do this, it uses the same process as the precedent method to create the first triangle. Figure 3.1(x)(iii) depicts the case of a rejected edge creation. The edge which links the point *b* to the point *c* is accepted since it does not create a triangle. However, the edge which links the point *c* to the point *a* is rejected. The purple circle is the circumcirlce of the *abc* triangle. The point *d* which is inside the purple circle is also co-circular with the points *a* and *b*, and it is closer than the point *c*. According to the properties of a Delaunay triangle, the point *c* is rejected as the third vertex of the Delaunay triangle .

When the scan line reaches the end of the cloud of points all the points are connected and the mesh is complete.

The two methods described above uses two different approaches. The first creates triangles starting from one edge and finding the appropriate third vertex to complete the triangle. The second starts from the third vertex and try to find the appropriate edge. Since the Delaunay triangulation is unique, the two methods give at the end the same 3D mesh. As it is not the main focus of this project to find the best way to create a 3D mesh, there is no comparison between the two methods.

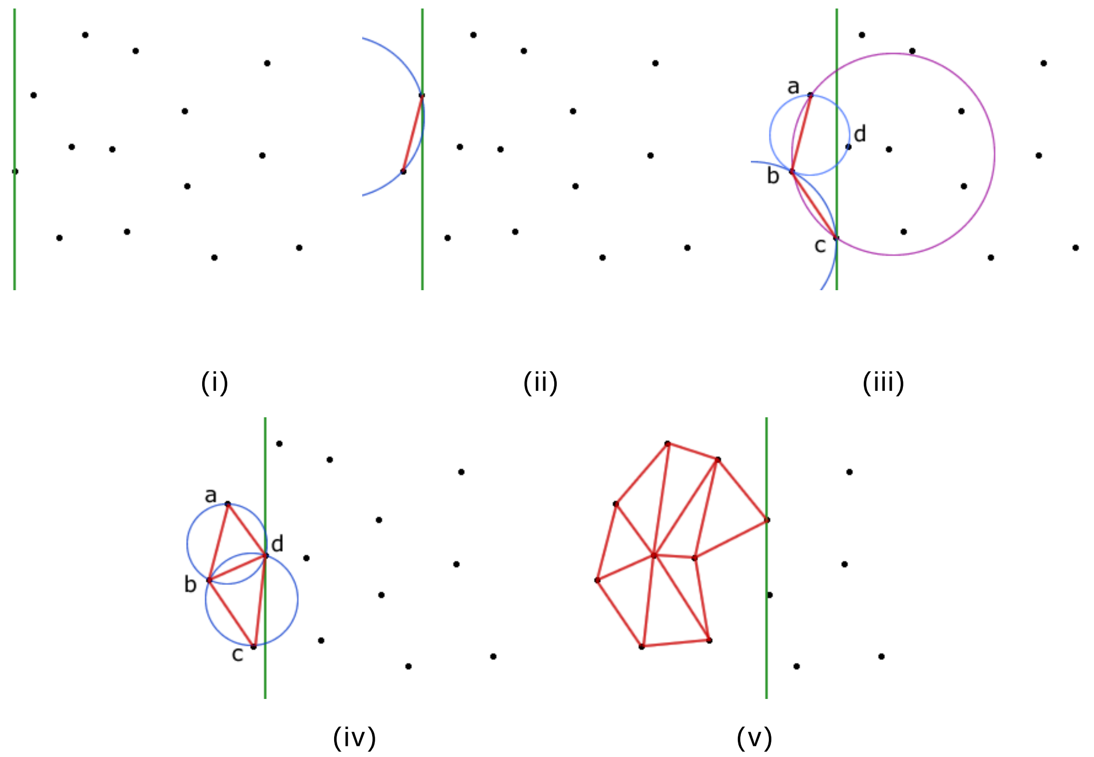


***Figure 3.1**: **(x)** (i) The scan line in green encounter the first point (ii) The first edge is created (iii) the creation of the abc triangle is rejected (iv) creation of Delaunay triangles (v) the process continue till the end of the cloud of points.*

3.2. Mesh filtering analysis

This part is the analysis of the method chosen in the part 2.5 of this paper. The filtering occurs between the triangulation and the tessellation part, when the creation of the cloud of points is complete.

As seen in the previous part 2.4.3, the method uses the density of 3D points to find errors due to unwanted reflections. Considering the point $P_j(x_j,y_j,z_j)$ which belong to the set of points *S*. The data cloud density of P_j is:

$$DS_j = \left\{ P_i(x_i, y_i, z_i) \left| \begin{pmatrix} (x_j - U_x) < x_i < (x_j + U_x) \\ (y_j - U_y) < y_i < (y_j + U_y) \\ (z_j - U_z) < z_i < (z_j + U_z) \end{pmatrix} \right. \right\}$$

Where U_x, U_y, U_z are known fixed variables.

And the density of P_j is:

$$Density_j = \sum (Pj \in DS_j)$$

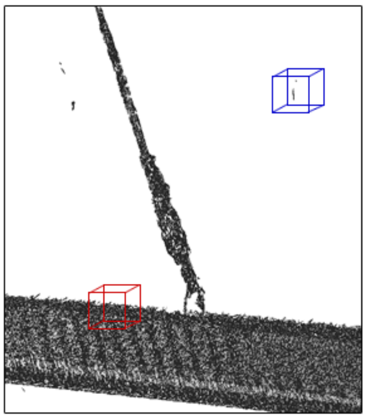
Since false 3D points are assumed to have a 3D density lower than the real 3D points, the right cloud of points have the following equation:

$$S = \{ Pj \,|Density_j > Density_{errors} \}$$

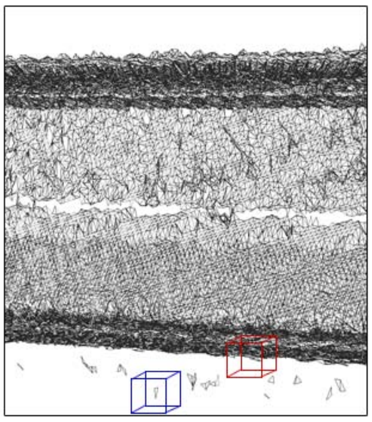
Where $Density_{errors}$ is a variable fixed at the beginning.

[J. Chen et *al*, 1998]

Figure 3.2(a) shows how the algorithm can be used in INROPA system. The errors far from the edges of the object are easily identified by the algorithm because there density is very low. However, the threshold fixed by the variable $Density_{errors}$ is not the only one which has to be adapted to the system. Indeed, the data cloud density depends on three different thresholds: U_x, U_y, U_z. These variables determine also the 3D density of a point since it is related to the data cloud density.



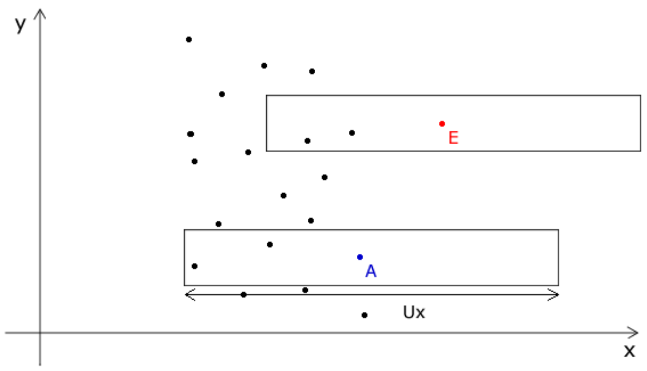
(i)



(ii)

***Figure 3.2**: **(a)** The cubes represent the 3D kernel used by the 3D data cloud density method, the blue one scans an error, the red one scans a real point (i) errors far from the edge of the 3D object. (ii) errors near the edge of the 3D object.*

To explain that, the figure 3.2(b) takes the problem in two dimensions. If the threshold U_x is too wide compare to the average distance between two points on the x-axis, the density of errors will be the same as real points. This is depicted figure 3.2(b); the error point *E* has the same density as the point *A*, then the algorithm considers the point *E* as a real point. The same problem happens if the threshold U_y is too wide and it can be also extended in three dimensions with the threshold U_z.



(b)

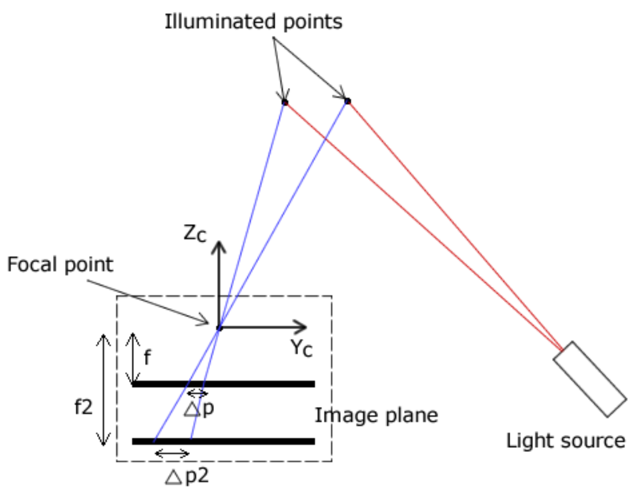
***Figure 3.2**: **(b)** U_x is too wide. $Density_E$ = $Density_A$ = 3.*

As seen before, the threshold U_x, U_y, U_z have to be calibrated. The variable U_x depends on the distance between 3D points on the x-axis. Hence, the calibration of U_x depends on the velocity of the conveyor and the frame rate of the camera:

$$U_x = \lambda \frac{v_c}{fr}$$

Where V_c is the velocity of the conveyor, *fr* is the frame rate of the camera and λ an independent fixed variable.

The threshold U_y and U_z depends on the focal length of the camera. Figure 3.2(c) shows how this parameter can change the distance between 3D points on the y-axis. Considering *f* the focal length of the camera and *Δp* the distance between the two illuminate points in the image plane. When the focal length *f* increase, *Δp* increase as well. This figure can be extended to the z-axis since during the 3D scanning, the object only translate on the x-axis.



(c)

***Figure 3.2**: **(c)** Influence of the focal length on the distance Δp between two illuminate points in the image plane.*

Characteristics of hardware are needed to calibrate those thresholds U_x, U_y, U_z. For the INROPA system, the distance between two points on the x-axis is 10mm. The distance between two points on the y-axis and z-axis is 1cm [K. A. Nissum et *al*, 2010].

Part III

Implementation

4. Implementation

In this part the implementation of the filtering method described in part 3.2 is explained. First of all the design of the implementation is enounced and the choice of the technology used is justified. Then, the first prototype is explained following by a memory test which leads to an improvement of the first code. The second code is the final implementation provided in this paper.

4.1. Design

The existing system of INROPA is coded in C/C++. One of the requirements for this project is that the implementation has to be compatible with the current system. The choice to implement the method in C is then a logic choice. Moreover, the software has to be able to compute a cloud of millions points. Using a low-level programming language is more relevant since one of the success criteria is to have a low-time consuming software. The following part describes the algorithm used on the implementation.

The input for the implementation is a .xyz file. The format of the xyz file used is as follows:

```
x-coord1 y-coord1 z-coord1
x-coord2 y-coord1 z-coord2
...
x-coordn y-coordn z-coordn
```

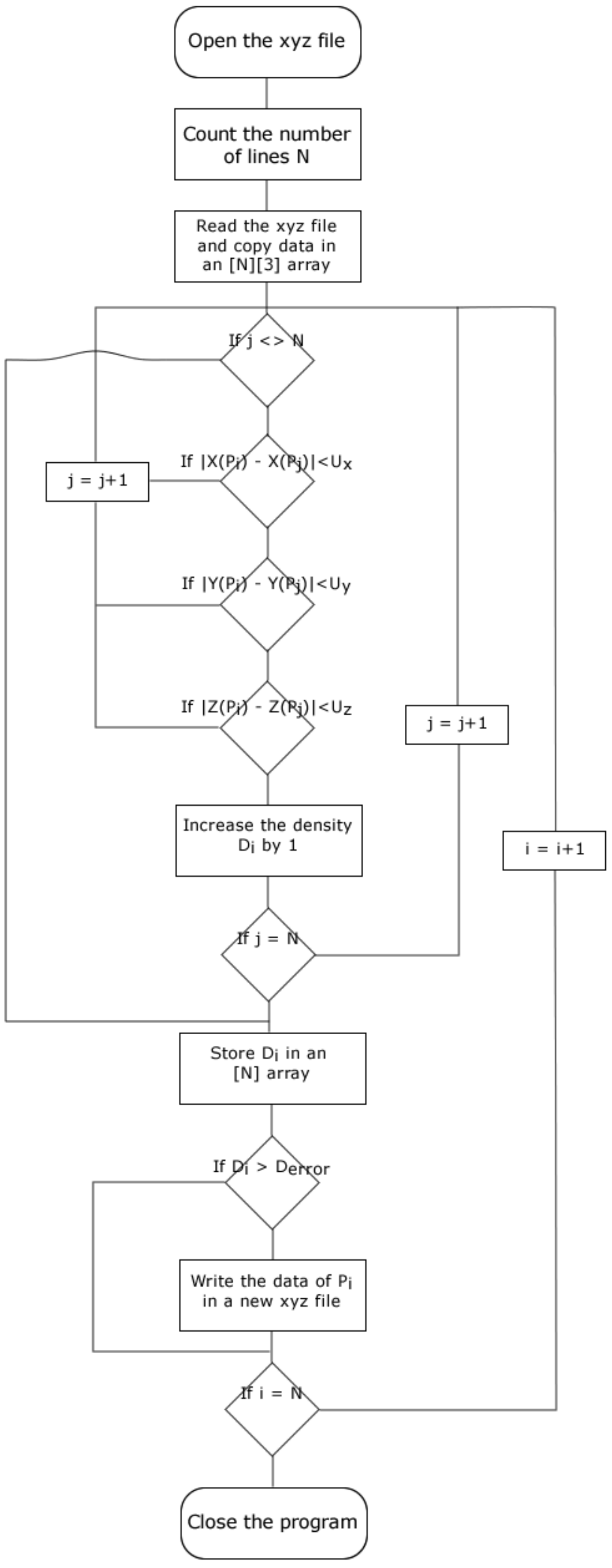
Each line of the file corresponds to one point of the cloud of points. The simpler is the input file, the quicker is the processing of the file. The output file is also a .xyz file which can be triangulated into a 3D mesh.

Flowchart of the algorithm

Figure 4.1(a) is the flowchart of the algorithm used in the implementation of the 3D data cloud density method. The first step is to read the xyz file. After storing all data into an array, each coordinates of each point are compared to each other in order to find the density of each point. Density values are then store into an array to be process after. If the density of a point is inferior to the threshold $Density_{error}$, the point is rejected and not copied into the new xyz file.

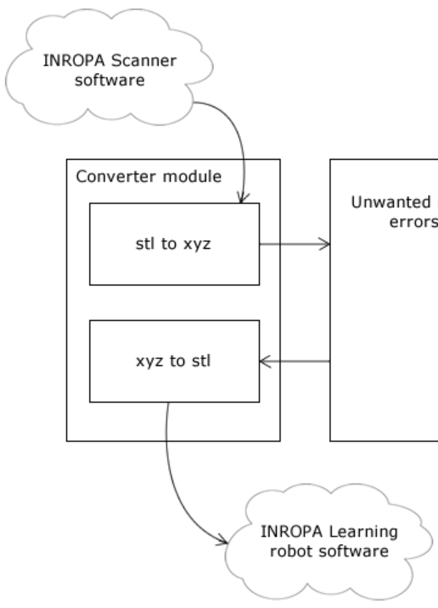
Integration of the implementation

According to [K. A. Nissum et *al*, 2010], INROPA software doesn't store the cloud of points and makes the triangulation of 3D points on the fly. The implementation made in this project only computes xyz file and returns a xyz file as well. A converter is then needed to change the stl file output of INROPA software into a xyz file. Then, a converter xyz into stl is also needed to give back a stl file to INROPA software. Figure 4.1(b) shows an overview of the integration.



(a)

***Figure 4.1**: **(a)** Flow chart of the algorithm.*

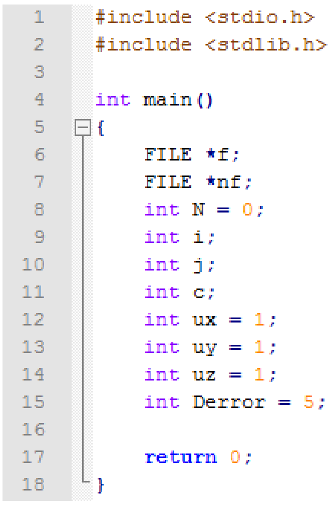

INROPA Scanner software
Converter module
stl to xyz
xyz to stl
Unwanted reflections errors filter
INROPA Learning robot software

(b)

***Figure 4.1**: **(b)** Overview of the implementation integration in INROPA system.*

4.2. 1st implementation

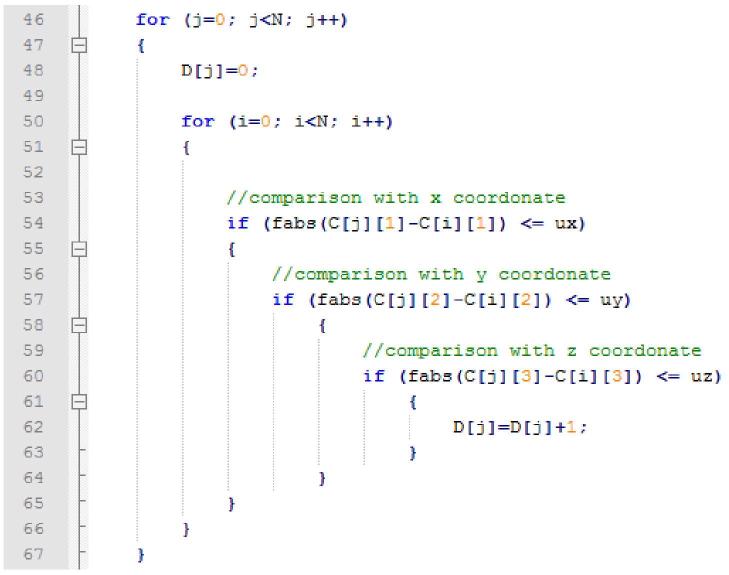
The first implementation follows the flow chart depicted figure 4.1(a). The header of the implementation is as follows:



```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *f;
    FILE *nf;
    int N = 0;
    int i;
    int j;
    int c;
    int ux = 1;
    int uy = 1;
    int uz = 1;
    int Derror = 5;

    return 0;
}
```

In the declaration of variables, ux, uy and uz represent the three thresholds from the data cloud density. Those variables have to be calibrated following the rules enounced in the part 3.2. The variable Derror have to be calibrated depending of the minimum density of real points.

The core of the implementation is the comparison method. This method compares each point coordinates to another point coordinates as following:



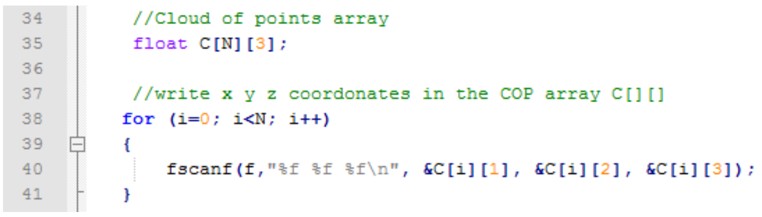
```
for (j=0; j<N; j++)
{
    D[j]=0;

    for (i=0; i<N; i++)
    {

        //comparison with x coordonate
        if (fabs(C[j][1]-C[i][1]) <= ux)
        {
            //comparison with y coordonate
            if (fabs(C[j][2]-C[i][2]) <= uy)
            {
                //comparison with z coordonate
                if (fabs(C[j][3]-C[i][3]) <= uz)
                {
                    D[j]=D[j]+1;
                }
            }
        }
    }
}
```

Comparisons are made between absolute values since the calculation is made on Euclidean distance. First, coordinates on x-axis are compared. If the Euclidean distance is superior that the threshold U_x, then, there is no need to compare the others coordinates. The same reasoning is made for the y coordinates. If the comparison point is considerate to be in the neighborhood of the compared point, the density of the compared point is increase by one. All density values are stored in a N dimension array.

4.3. Memory test

The first implementation gives good results in term of errors deletion. However, since the implementation is coded in C, the program is limited in memory by the stack address space. Indeed, the memory test reveals that the first implementation cannot compute a file of more than 130284 lines.

This is due to the creation of the static array of the cloud of points:

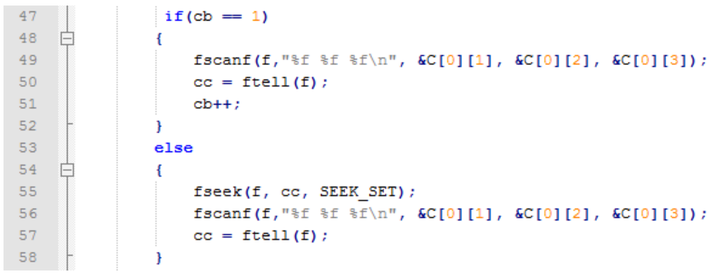


```
 //Cloud of points array
 float C[N][3];

 //write x y z coordonates in the COP array C[][]
for (i=0; i<N; i++)
{
    fscanf(f,"%f %f %f\n", &C[i][1], &C[i][2], &C[i][3]);
}
```

The stack address space cannot create an array of more than 130284 rows because the buffer space is limited. One solution of that problem is to read the file each time a new point is compared. This approach is made in the second implementation described in the following part.

4.4. 2nd implementation

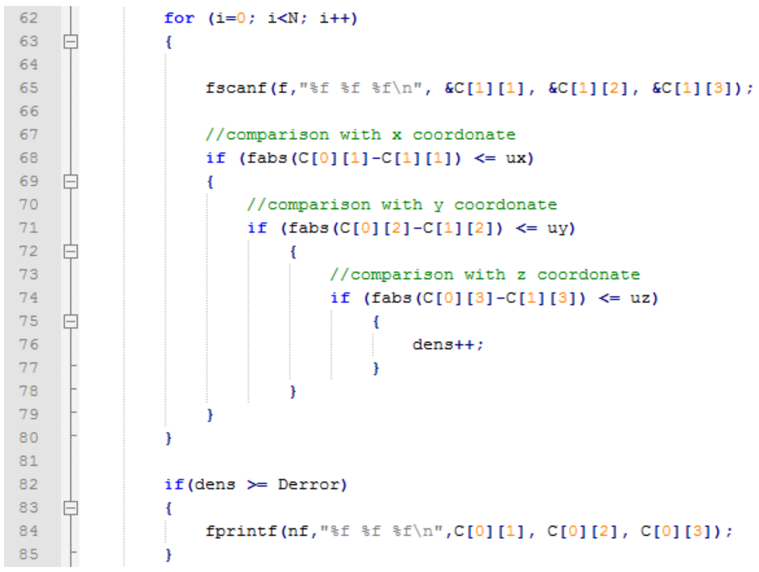
Unlike the first implementation which creates N dimensions arrays, the second implementation only create a two dimensions array which will be mentioned in the following as the comparison array. The first row of this array contains data of the compared point and the second row contains data of the comparison point. Each time the comparison loop starts again, the first row of the array has to be updated with data of the next point. To do that, the program needs to remember the number of the last line read. In C, the notion of line in a file doesn't exist. To read a file line by line, the function *fseek()* is used:



```
if(cb == 1)
{
    fscanf(f,"%f %f %f\n", &C[0][1], &C[0][2], &C[0][3]);
    cc = ftell(f);
    cb++;
}
else
{
    fseek(f, cc, SEEK_SET);
    fscanf(f,"%f %f %f\n", &C[0][1], &C[0][2], &C[0][3]);
    cc = ftell(f);
}
```

The function *fseek(File * stream, lont int offset, int origin)* sets the position indicator associated with the *stream* to a new position defined by adding *offset* to a reference position specified by *origin*. Each time the comparison loop starts, the function *fseek()* is launched and positions the reader cursor at the beginning of the next line to read. Then the function *fscanf()* read the line and update the first row of the comparison array. The function *ftell()* is used to remember the position of the reader cursor in order to read the next line during the forthcoming comparison loop.

Another difference between the first and the second implementation is the fact that density values are not anymore store in an N dimensions array. Instead, the density value of the compared point is stored in a simple variable. At the end of each comparison loop, the density value is directly compared to the density threshold:



```
for (i=0; i<N; i++)
{

    fscanf(f,"%f %f %f\n", &C[1][1], &C[1][2], &C[1][3]);

    //comparison with x coordonate
    if (fabs(C[0][1]-C[1][1]) <= ux)
    {
        //comparison with y coordonate
        if (fabs(C[0][2]-C[1][2]) <= uy)
            {
                //comparison with z coordonate
                if (fabs(C[0][3]-C[1][3]) <= uz)
                    {
                        dens++;
                    }
            }
    }
}

if(dens >= Derror)
{
    fprintf(nf,"%f %f %f\n",C[0][1], C[0][2], C[0][3]);
}
```

Since in this implementation, there isn't creation of N dimensions array, the problem of memory is solved.

Part IV

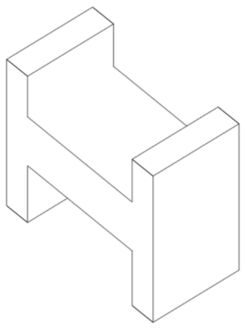
Test

5. Tests

In this part, several tests are performed in order to determine if the implementation can answer the problematic enounced in the problem statement part 2.1.1. To make the tests, models have to be generated. In the first sections, the implementations of testing tools are described. Then, two tests are performed in order to know if the implementation reaches the success criteria.

5.1. Implantation of the model generator

In order to perform tests, a model of a real cloud of points needs to be generated. Figure 5.1(a) shows how the model has to be, according to [K. A. Nissum et *al*, 2010]. This is the representation of a metallic beam. Since metallic beam are made of shiny material like polished metal, it is relevant to create a model from this object. Moreover the shape of these objects is susceptible to give spurious reflections.

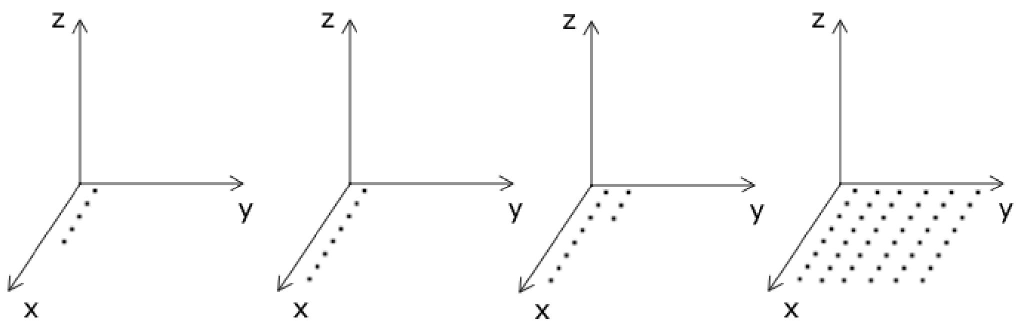


(a)

***Figure 5.1**: **(a)** Shape of the model.*

5.1.1. Point generator

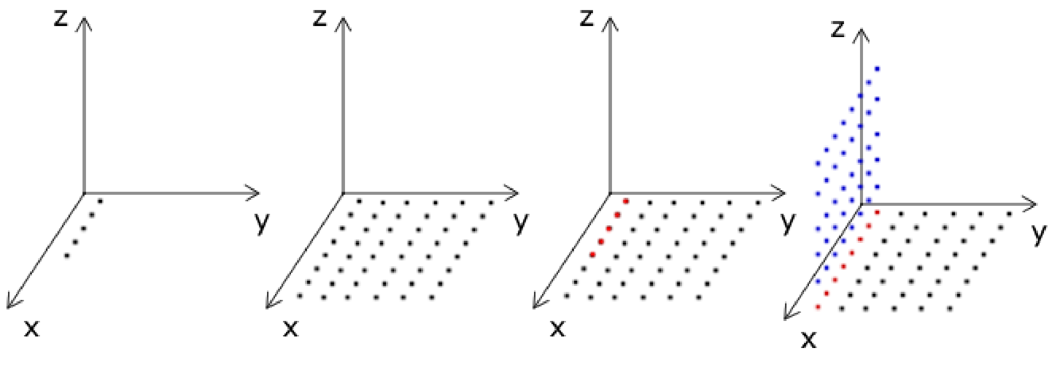
The first part of the implementation is the point generator. Each face is decomposed into several lines which are generated one after the other. Figure 5.1(b) show how a face is generated.



(b)

***Figure 5.1**: **(b)** Generation of a face by the point generator. Each picture is taken at a different time of the creation of a face.*

Each face is generated separately without taking account the points created by the other faces. Therefore, the point generator will create duplicate points which will distort the 3D data cloud density. Figure 5.1(c) shows an example of this phenomenon. The first face is generated on the xy plane. Then the second face is generated on the xz plane. The red points represent the duplicate points.

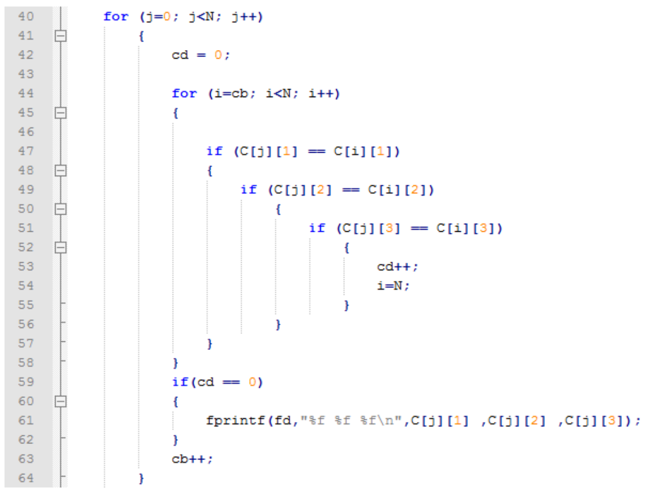


(c)

***Figure 5.1**: **(c)** The creation of two faces which generate duplicate points (in red).*

5.1.2. Deletion of duplicates

To generate a workable, the model generator must delete all duplicate points. To do that, a simple algorithm is used to compare all the points with each other:



```
for (j=0; j<N; j++)
    {
        cd = 0;

        for (i=cb; i<N; i++)
        {

            if (C[j][1] == C[i][1])
            {
                if (C[j][2] == C[i][2])
                {
                    if (C[j][3] == C[i][3])
                    {
                        cd++;
                        i=N;
                    }
                }
            }
        }
        if(cd == 0)
        {
            fprintf(fd,"%f %f %f\n",C[j][1] ,C[j][2] ,C[j][3]);
        }
        cb++;
    }
```

When a point is detected as a duplicate of another one, this point is not written in the new xyz file and the algorithm starts to compare the next point. Moreover, to optimize the algorithm, the point number one is compared to all the other N points. Then the point number two is compared to all other N points minus the first point and so on. So the point number n is compared to N-(n-1) next points.

5.2. Implementation of the converter stl to xyz

To confirm that our implementation work in real situation, the implementation of a converter stl file to xyz file is necessary. When storing a stl file, there are two fundamentally different ways to do it; as ASCII or binary files. The binary stl file describes each triangle with twelve 32-bit-floating point numbers whereas the ASCII stl file described each triangle twelve floating point numbers. In order to extract coordinates of points, an ASCII stl file is used.

The structure of ASCII stl file is as following:

```
solid name
 facet normal ni nj nk
   outer loop
     vertex v1x v1y v1z
     vertex v2x v2y v2z
     vertex v3x v3y v3z
   endloop
 endfacet
//repeat for all facets
endsolid name
```

Each triangle is defined by a normal vector and three vertexes. To convert a stl file into a xyz file, the first step is extract the vertex coordinates. Then, since each vertex is shared by three triangles, the converter needs to delete all duplicate points.

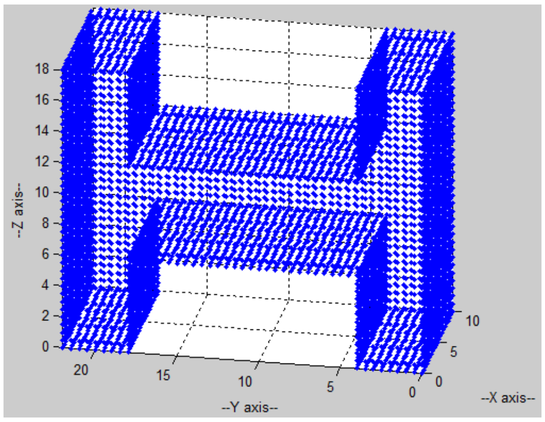
To read the file, the function fgetc() is used. When the program finds the words vertex, it starts to write the coordinates of the vertex in a xyz file.

The next step is to delete the duplicate points. Theoretically, the program to delete duplicate points described in the previous 5.1.2 can manage to do it in the resulting xyz file. However, a real stl file usually contains millions of faces which mean even more vertex coordinates. The time of computing all those points with the previous program can be a real problem. Thus, the following Linux/UNIX command line can be use:

sort file.xyz | unic > newfile.xyz

5.3. Velocity test

To do the velocity test, several models have to be created. Indeed, the velocity of the implementation depends directly on the number of points of the cloud of points. All models have the same shape, only the number of points is different. Figure 5.3(a) shows the first model with 2000 points.



(a)

***Figure 5.3**: **(a)** Testing model with 2000 points.*

5.3.1. Test from model

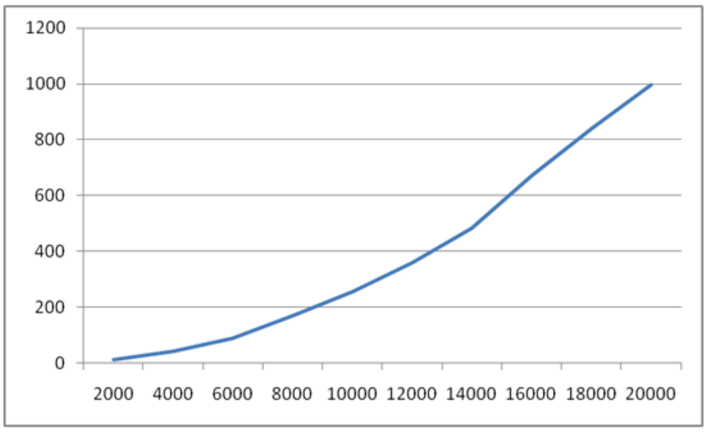
The velocity test consists on running the implementation with several models. For each model, the number of points is changed, starting from 2000 points to 20000 points, each time increasing the number of point by 2000. Then, each time the number of points increases by 10000 until 100000. Finally, the number increases by 100000 each time until 1000000.

The success criterion for this test is to be able to compute a cloud of 1000000 points in less than 30s.

Results

The results for the first series of models is presented figure 5.3(b). It is obvious that the implementation is far too slow. Therefore, the success criterion of time consuming is not reach with this implementation.

Number of points of the model	Time of processing in s
2000	10.107
4000	40
6000	89.504
8000	168.8
10000	254.56
12000	359.9
14000	484
16000	670.756
18000	840.5
20000	998.188



(b)

***Figure 5.3**: **(b)** Results of the time processing of the 1st series of models.*

Improvement in the implementation

The first velocity test shows that the implementation is far too slow to be workable. This problem comes from the fact that for each comparison, the program needs to read the xyz file. Another solution to avoid the memory errors from the stack address space is to use the dynamic allocation of memory. Then, the only limits of memory come from the hardware of the computer which runs the program.

The implementation is the same as the first implementation described in part 4.2, but instead of creating fixed N dimensions arrays, all arrays are dynamically allocated via the function *malloc()*:

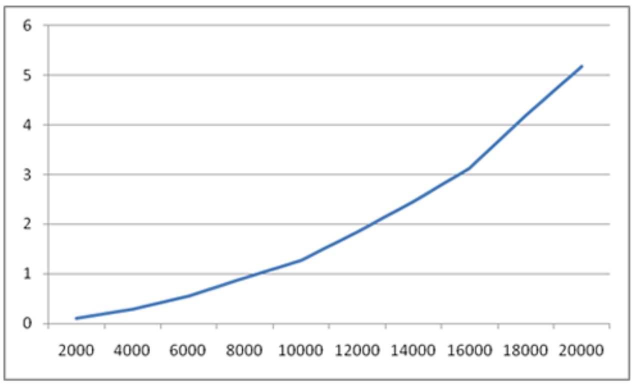
```
//Cloud of points array
float** C = (float**) malloc(N * sizeof(float*));

for(i=0; i<N; i++)
{
    C[i] = (float*) malloc(3 * sizeof(float));
}
```

The velocity test is then remade with the new implementation.

Results

The results for the first series of models is presented figure 5.3(c). Compare to the first implementation, the time of processing decrease by 99%. For cloud of points of less than 20000 points, the implementation reaches the success criterion of time consuming.

Number of points of the model	Time of processing in s
2000	0.0993
4000	0.285
6000	0.548
8000	0.915
10000	1.273
12000	1.845
14000	2.461
16000	3.131
18000	4.2
20000	5.186

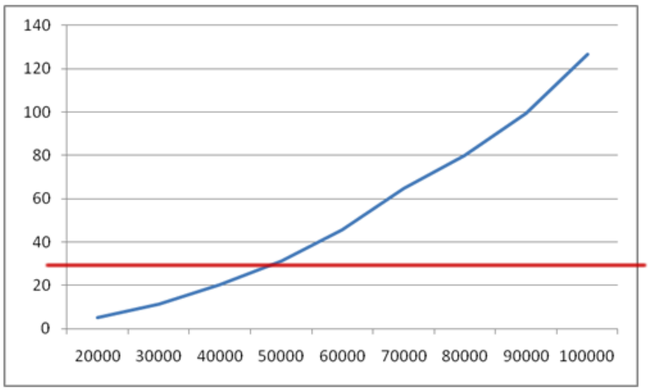


(c)

***Figure 5.3**: **(c)** Results of the time processing of the 1st series of models.*

The results for the second series of models is presented figure 5.3(d). The time of processing is still decent: to process a cloud of 100000 points, the implementation needs around two minutes. However, the success criterion of time consuming is reached only if the cloud of points is composed by less than 50000 points.

Number of points of the model	Time of processing in s
20000	5.186
30000	11.416
40000	20.187
50000	31.302
60000	45.751
70000	64.754
80000	79.748
90000	99.301
100000	126.525

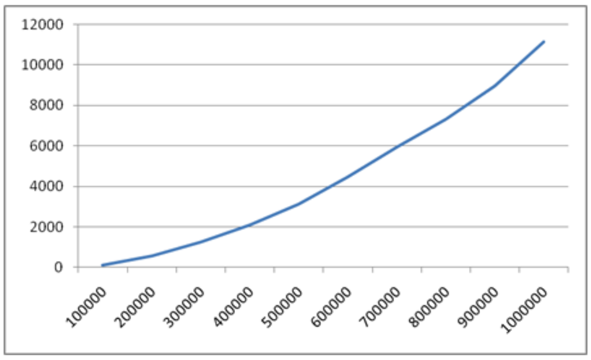


(d)

***Figure 5.3**: **(d)** Results of the time processing of the 2nd series of models.*

To have an idea how the time consuming grows depending on the number of points of the model, the results for the second series of models is presented figure 5.3(e). For a cloud of one million points, the time processing exceeds the 30 seconds success criterion.

Number of points of the model	Time of processing in s
100000	126.525
200000	559.097
300000	1241.309
400000	2109.48
500000	3123.178
600000	4471.509
700000	5932.73
800000	7307.46
900000	8960.302
1000000	11135.455

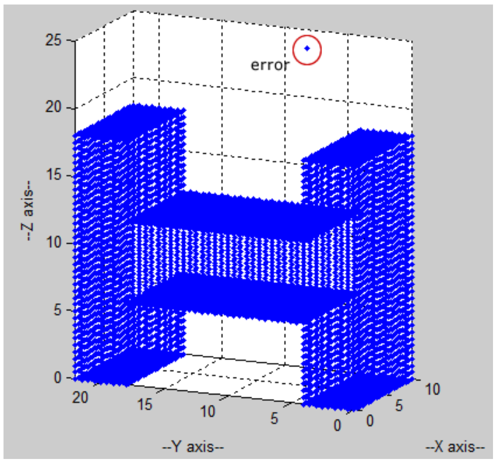


(e)

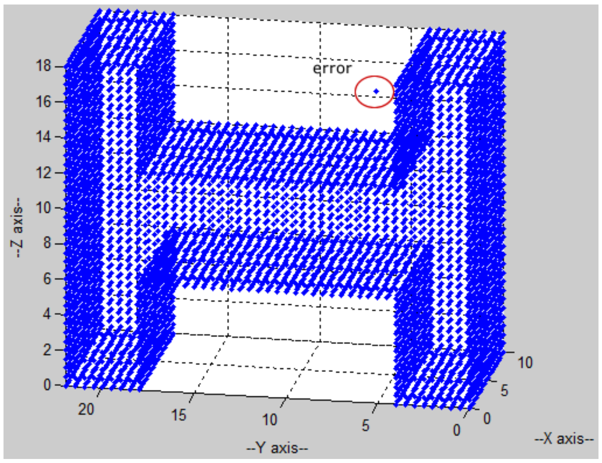
***Figure 5.3**: **(e)** Results of the time processing of the 3rd series of models.*

5.4. Quality test

The quality test consists on processing 4 different models and finding if the implementation detects and deletes the errors. Each of these models has one kind of errors presented in part 2.1.1. Figure 5.4(a) and figure 5.4(b) depict the 4 models used in these tests. Each point is separated of 0.5 units from each others.

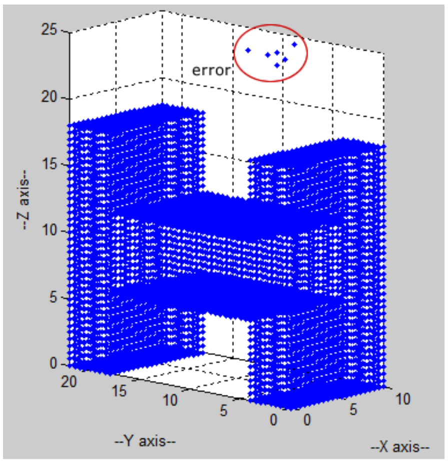


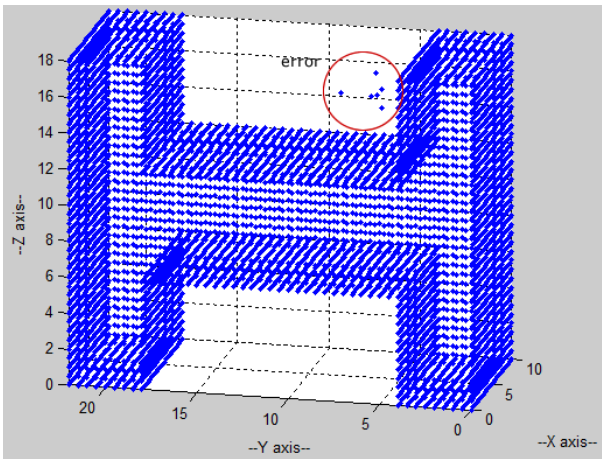
(i)



(ii)

***Figure 5.4**: **(a)** (i) the first model: the error is a single point far from the edge of the object (ii) the second model: the error is a single point near to the edge of the object.*





(i) (ii)

***Figure 5.4**: **(b)** (i) the third model: the error is a blob far from the edge of the object (ii) the forth model: the error is a blob near to the edge of the object.*

During the test, the thresholds u_x, u_y, u_z of the data cloud density is varied five times; {1, 2, 5, 10}. Each time, the threshold $Density_{error}$ is varied from 8 to 2.

5.4.1. Test from model

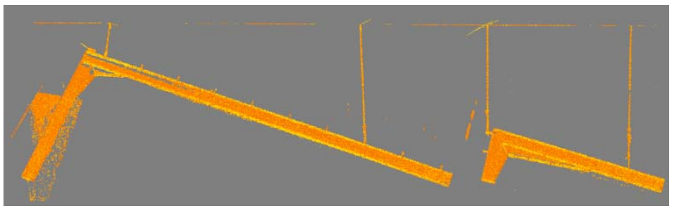
The implementation is run 8 times for each type of error. The following table summarizes the results of the 32 tests where “Ok” means that all the errors are deleted and “No” means that none of the errors are deleted:

	Model n°1	Model n°2	Model n°3	Model n°4
$U_x = U_y = U_z = 1$ $Density_{error} = 8$	Ok	Ok	Ok	Ok
$U_x = U_y = U_z = 1$ $Density_{error} = 2$	Ok	Half: points closest each others	Half: points closest each others	Ok
$U_x = U_y = U_z = 2$ $Density_{error} = 8$	Ok	Ok	Half: points closest to the edge	No
$U_x = U_y = U_z = 2$ $Density_{error} = 2$	Ok	-1: the farthest from the blog	-1: the farthest from the blog and the edge	No
$U_x = U_y = U_z = 5$ $Density_{error} = 8$	Ok	Ok	No	No
$U_x = U_y = U_z = 5$ $Density_{error} = 2$	Ok	No	No	No
$U_x = U_y = U_z = 10$ $Density_{error} = 8$	Ok	No	No	No
$U_x = U_y = U_z = 10$ $Density_{error} = 2$	Ok	No	No	No

The test is relevant for the model n°1, in any circumstances. Obviously, if the range of the 3D kernel is as big as the distance the error lies from the object, the result would be different. For an isolated blob, the $\text{density}_{\text{error}}$ must not be equal or inferior to the number of points which constitute the blob. Otherwise, there is a risk that the error is not detected since points from the blob increase their density each other. For the models number 3 and 4, the smaller 3D kernel range is, the better the detection will be.

5.4.2. Test on INROPA file

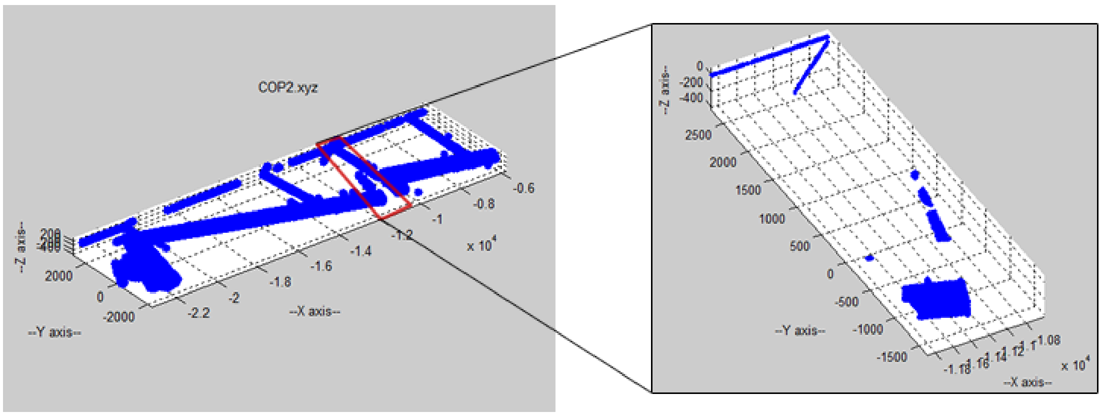
The last step of the test is to compute INROPA file to confirm that the implementation work on a real file. The stl file of INROPA has more than 1.2 millions of points. In order to decrease the time of processing, only a chosen part of the cloud of points will be process. Figure 5.4(c) shows a representation of the stl file of INROPA.



(c)

***Figure 5.4**: **(c)** Representation of the stl file of INROPA.*

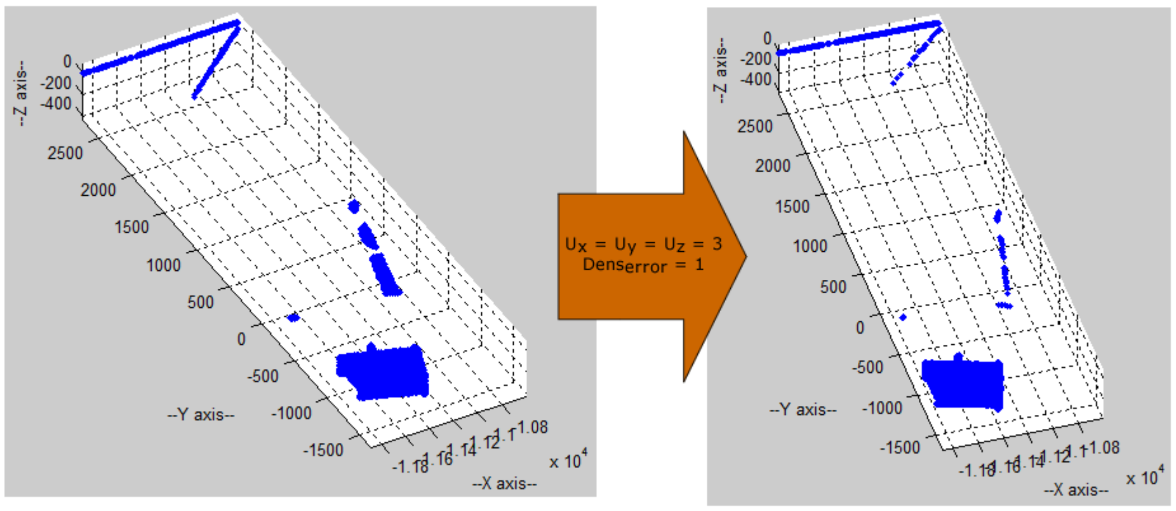
Using the converter stl to xyz implemented on the part 5.2, the respective xyz file is created. Figure 5.4(d) shows the representation of this cloud of points via Matlab. The red area represents the part of the cloud of points which will be process. Both blob error and single point error are present inside this area.



(d)

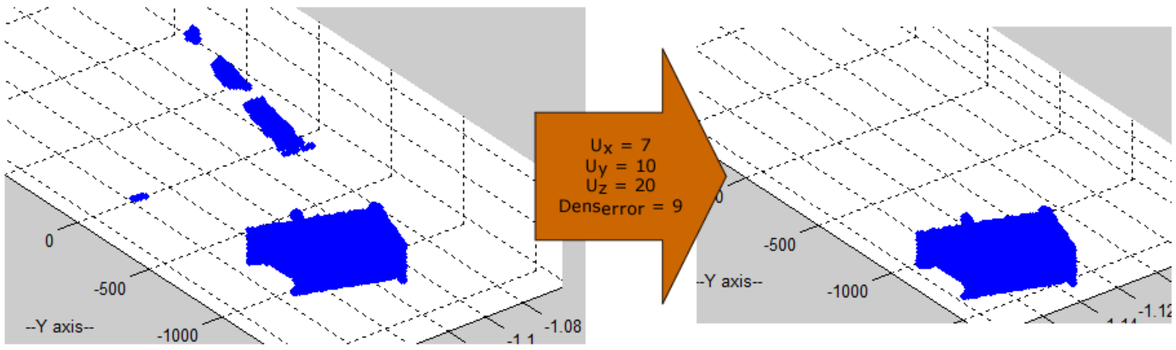
***Figure 5.4**: **(d)** Selection of the test area of INROPA file.*

There is 38657 points in the selected area. The process of this set of points last 13.492 s. To find the best thresholds, several tests were performed. Figure 5.4(e) shows one of the several tests. One of the most efficient thresholds setting is: U_x = 7, U_y = 10, U_x = 20, $Dens_{error}$ = 9 (figure 5.4(f)).



(e)

***Figure 5.4**: **(e)** One of the several test to find the right thresholds settings.*



(f)

***Figure 5.4**: **(f)** Efficient thresholds setting.*

Part V

Project closure

6. Discussion

This part contains a discussion of the problem areas which have been faced throughout the project. The results of the test chapter are then discussed to give a direction for further development.

This project has been proposed by the company INROPA who has specific initial requirements. The solution had to be software based since the modification of the hardware could be problematic. The choice of the system to analyze was therefore restricted to two methods. The segmentation filtering and the 3D data cloud density method. The INROPA system captures and process images from camera on the fly which means that it is difficult to have raw video data to test and implement the segmentation filtering method. This leads inevitably to the choice of the 3D data cloud density method.

Implementation evaluation

To implement a method properly, raw data to process are needed. The system of INROPA creates a mesh from 3D points on the fly. Thus, the raw cloud of points is not accessible. However, the system stores the mesh construction in an STL file. The ASCII STL file from INROPA contains the coordinates of more than 2 millions of triangles and weight 714 megabytes. Free visualization software doesn't have the possibility to read an ASCII STL file of 2 millions. The initial implementation used a generated cloud of points to test the efficiency of the program. Hence, a new implementation was needed since a memory error occurs. The next implementation was then too slow to perform the testing part. Finally, a third implementation had to be made. The one used in the testing part.

Test evaluation

During the testing part, the idea was to create model to perform different tests since the raw stl file was assumed to be inoperable. The creation of a generator of cloud of point was needed. After the implementation of the converter stl to xyz done, news requirements arise. Indeed, during the velocity test, it occurs that the velocity law is not a linear function: the time of processing increases very fast when the number of points increases. Even optimized, the implementation is still too slow considering the requirements of INROPA.

Concerning the quality tests, since the implementation takes more than 3 hours to process the real cloud of point from INROPA, it was necessary to partition the cloud of point in order to compute them.

Future development

The main improvement of the current implementation is the time of processing. Indeed, this is the only success criterion not reached. The logic of comparison needs to be revised. Another way to improve it would be to automatically cluster huge cloud of point into smaller one. Indeed, the velocity law is not linear; processing multiple small clouds of points would decrease the total time of processing.

Moreover an automatic way to calibrate thresholds can be found. It would make the system more polyvalent and more robust.

7. Conclusion

The main purpose of the project was to implement a method to remove errors from spurious reflection due to uncontrolled reflection on shiny surface. This project was proposed by the company INROPA which has a desire to find a software based solution to avoid errors from spurious reflection. Through a preliminary analysis of four eventual solutions, the 3D data cloud density method has been chosen considering the requirements of INROPA.

The creation of a tree-dimensions model can be divided into four main steps: the image data acquisition, the image segmentation, the triangulation and the mesh construction. Since the solution has to be run between the triangulation and the mesh construction, the steps to create a tree-dimensions model are described in the analysis part to have a better understanding of the process.

The main idea of the 3D data cloud density method is to find the number of nearby other 3D points. The most basic way to store a cloud of point is to use the xyz file format. Therefore, the implementation has to be able to read a xyz file and to compare all points between each others. In the implementation part prototype are made. Those prototypes are then tested.

Based on the tests performed, it is concluded that the 3D data cloud density method is an efficient technique to find errors from spurious reflections. The velocity of the technique depends essentially on the implementation. However, for an industrial usage, the implementation created in this paper has to be improved in order to decrease the processing time.

8. Bibliography

[Wikipedia, 2007] Paint robot article. WWW. http://en.wikipedia.org/wiki/Paint_robot

[A. Pichler *et al*, 2004] A. Pichler, H. Bauer, C. Eberst, C. Heindl, J. Minichberger (2004). Towards more Agility in Robot Painting through 3D Object Recognition. IPROM Conference 2004.

[K. A. Nissum et *al*, 2005] K. A. Nissum, T. H. Larsen (2005). 3-D laser scanner for painting robots. AAU report 2005.

[E. Trucco et *al*, 1994] E. Trucco, R. B. Fisher, A. W. Fitzgibbon (1994). Direct Calibration and Data consistency in 3-D Laser Scanning. Proceedings of the conference on British machine vision (vol. 2) 1994.

[K. A. Nissum et *al*, 2010] K. A. Nissum, T. H. Larsen (2010). Interview. INROPA 2010.

[Wikipedia, 2010] Reflection (physics). WWW. http://en.wikipedia.org/wiki/Reflection_(physics)

[I. Ihrke et *al*, 2008] I. Ihrke, K. Kutulakos, H. P. A. Lensch, M. Magnor, W. Heidrich (2008). State of Art in Transparent and Specular Object Reconstruction. The Eurographics Association 2008.

[D. J. Svetkoff, 2000] D. J. Svetkoff, D. B. T. Kilgus (2000). Method and system for suppressing unwanted reflections in an optical system. U. S. Patent n°6028671.

[J. Clark et *al*, 1996] J.Clark, E. Trucco, L. B. Wolff (1996). Using light polarization in laser scanning. Image and Vision computing 15, pp 107-117.

[J. Chen et *al*, 1998] J. Chen, D. Yang, H. Zhou(1998). Avoiding spurious reflection from shiny surfaces on a 3D real-time machine vision inspection system. IEEE instrumentation and Measurment Technology Conference 1998.

[D. Yang et *al*, 1999] D. Yang, J. Chen, H. Zhou, S. Buckley (1999). Two practical ways to avoid spurious reflections from shiny surfaces on a 3D machine vision inspection system. SPIE Vol. 3652.

[Wikipedia Edge detection, 2010] Edge detection article. WWW. http://en.wikipedia.org/wiki/Edge_detection

[D. Lanman et *al*, 2009] D. Lanman, G. Taubin (2009).The mathematics of 3D triangulation. WWW. http://mesh.brown.edu/byo3d/

[Wikipedia D. triangulation, 2010] Delaunay triangulation article. WWW. http://en.wikipedia.org/wiki/Delaunay_triangulation

[K. Chakib, 1999] K. Chakib (1999). Triangulation de Delaunay. WWW. http://www.kaddour.com/chap4/chap4.htm

[Preparata & Shamos, 1985] F. P. Preparata, M. I. Shamos (1985). Computational Geometry an Introduction. Upson Hall. ISBN: 0-387-96131-3

[J. D. Boissonat & M. Yvinec, 2007] J. D. Boissonat, M. Yvinec (2007). De la géométrie algorithmique au calcul géométrique. WWW.

http://www-sop.inria.fr/geometrica/courses/

[Wikipedia xyz, 2010] XYZ file format article. WWW. http://en.wikipedia.org/wiki/.xyz

Part VI

Appendix

9. Appendix A: The enclosed CD-ROM

The enclosed CD-ROM has the following contents:

1. Implementation/
   The source code of the final implementation developed in this project.

2. Report/
   The final project report in portable document format (pdf)

3. Application/
   The source code of the .xyz viewer in Matlab