Attitude Determination Using Imaging Lidar

Camille Decoust
Embry-Riddle Aeronautical University - Daytona Beach

Embry Riddle Aeronautical University

Aerospace Engineering Department

Master of Science Thesis

Attitude Determination Using Imaging Lidar

by

Camille Decoust

Daytona Beach, 2008

ATTITUDE DETERMINATION USING IMAGING LIDAR

by

Camille Decoust

This thesis was prepared under the direction of the candidate's thesis committee chairman, Dr. Bogdan Udrea, Department of Aerospace Engineering, and has been approved by the members of her thesis committee. It was submitted to the Aerospace Department and was accepted in partial fulfillment of the requirements for the degree of Master of Aerospace Engineering.

THESIS COMMITTEE:

[illegible] Dec. 9, 2008

Bogdan Udrea, chairman

[illegible]

Frederique Drullion, Member

[illegible]

Yechiel Crispin, Member

[illegible] 12/10/08

Department Chair, Aerospace Engineering Date

[illegible] 12/10/08

Associate Provost for Academics, James Cunningham Date

Acknowledgements

The author wishes to express special thanks to her Thesis Advisor, Dr. Bogdan Udrea, whose constant encouragement, helpful counsel and practical suggestions were crucial to the successful outcome of this thesis. Appreciation is also due to Drs. Frederique Drullion and Yechiel Crispin, Thesis Committee Members, for their assistance in preparing this manuscript.

This statement of acknowledgment would be incomplete without an expression of sincere appreciation and gratitude to both the author's friends, especially Julien Thivend and Jeremy Hart and family for providing support and encouragement needed to complete the task.

ABSTRACT

Author: Camille Decoust

Title: Attitude Determination Using Imaging Lidar

Institution: Embry Riddle Aeronautical University

Degree: Master of Science in Aerospace Engineering

Year: 2008

The purpose of this study is to determine the attitude of an out of control object using a new technology called lidar (Light Ranging and Detection). As the number of spacecraft continues to grow, it is paramount to introduce a new type of autonomous on-orbit satellite inspection and repair involving docking. Traditional space vision technology is based on video systems. This method is limited by the necessity of operating when the target is illuminated by the sunlight or using its own source of illumination. The use of laser imaging technology offers an elegant solution to these challenges. This approach allows the collection of range data, while scanning the lidar field-of-view together with the transmitted laser beam across the required solid angle. A lidar simulator was implemented to generate point clouds of digital 3D models. This thesis describes methods that can be used to detect features such as edges, boundaries, surfaces and corners in the point cloud. From those features it was possible to define a reference frame and associate it to the object. Observing the evolution of this body frame, the changes in orientation can be deduced in the direction cosine matrix form. It was desired to retrieve angular rates in Euler angle form but since the conversion from rotation matrix to Euler is not a bijection, no satisfying results were obtained. The results are therefore expressed in terms of rotation matrix. It was found that depending on the orientation of the spacecraft the accuracy of the results varied. The results indicate that filtering of the direction cosine matrices might yield good data for determining attitude rates.

Contents

List of Tables

List of Figures

Symbols and Abbreviations

Abbreviation	Description	Definition
ADM-Aeolus	Atmospheric Dynamics Mission	
ALADIN	Atmospheric Laser Doppler Instrument	
ALHATA	Autonomous Landing and Hazard Avoidance Technology	
ATLID	ATmopsheric LIDar	
ATV	Automated Transfer Vehicle	
az	Azimuth	
CAD	Computer Aided Design	
CALIPSO	Cloud-Aerosol Lidar and Infrared Pathfinder Satellite Observation	
CSA	Canadian Space Agency	
NRC-CNRC	National Research Council Canada	
DCM	Direction Cosine Matrix	
deg	Degree	
DIAL	Differential Absorption Lidar	
el.	Elevation	
ESA	European Space Agency	
FOV	Field Of View	
GLAS	Geoscience Laser Altimeter System	

Abbreviation	Description	Definition
GNC	Guidance Navigation and Control	
HR	High Resolution	
HTV	Japanese Transfer Vehicle	
ICESat	Ice Cloud and Elevation Satellite	
ISS	International Space Station	
JAXA	Japanese Space Agency	
LASSO	LCS Algorithms for Spacecraft Servicing On-orbit	
LCS	Laser Camera System	
LITE	Lidar In-space Technology Experiment	
LS	Least Squares	
MGS	Mars Global Surveyor	
MOLA	Mars Orbiter Laser Altimeter	
MSR	Mars Sample Return	
NASA	National Aeronautics and Space Administration	
NN	Nearest Neighbor	
OBSS	Orbiter Boom Sensor System	
RF	Reference Frame	
RF_L	Lidar Reference Frame	
RVS	Rendez-vous and Docking Sensor	
STDP	Space Technologies Development Program	
STL	Stereolithography	
SVD	Single Value Decomposition	
TOF	Time Of Flight	
WALES	Water vApour Lidar Experiment in Space	

Introduction

Hundreds of orbiting spacecraft provide a broad variety of services, including global communication and meteorological monitoring. Among these satellites are the International Space Station, Hubble Telescope as well as large number of military satellites. As their number will certainly continue to grow, it will be paramount to introduce a new type of autonomous on-orbit satellite inspection and repair including rendezvous and docking. Precise autonomous performance of space maneuvers, especially within close proximity to a target satellite, requires a novel type of robotic vision system. Such a system will provide reliability and high accuracy of all data required to navigate the seeker spacecraft towards the target satellite. Traditional space vision technology is based on using various types of video systems. This methodology is limited by the necessity of operating when the target is illuminated by the sunlight or using its own source of illumination. Both of these approaches have serious shortcomings:

1. The sunlight limits the operation only to those periods, when the sunlight is present,

2. The use of the hand-made illuminator, even a well collimated source, limits the range to the target spacecraft

3. Finally, both of theses techniques quite often lead to image distortions that prohibit or make the navigation extremely difficult and risky.

Compared with passive optical and active radar/microwave instruments, lidar systems produce substantially more accurate and precise data without reliance on natural light sources and with much greater spatial resolution. It is quite common to use a laser ranging device along with the imaging video system. Laser radar technology has been demonstrated for about four decades, since the laser source was invented. This technique is based on the use of a pulsed laser beam aligned with the optical receiver-telescope, which collects the laser photons reflected back from the target on the system's detector. By using time-of-flight calculations, the range between the laser source and the target could be accurately derived. However, the image distortions will still make the required close proximity maneuvering quite risky. The use of laser imaging technology offers quite elegant solution to these challenges and ESA identified it as "the most sustainable technology for future exploration mission" This approach allows the collection of range data, while scanning the lidar field-of-view together with the transmitted laser beam across the required solid angle. Each individual laser pulse provides accurate range information, while an extremely accurate knowledge of the scanning optics position provides with accurate position information. In addition, each point of the point cloud contains the return signal intensity data. Point cloud is the term use to define a set of vertices in a three-dimensional coordinate system. So far imaging lidar (Light Detection and Ranging) has been widely used and tested for earth atmosphere studies and terrain mapping. The key earth missions in lidar development are described in Chapter 1. The impressive accuracy of this technology makes it a relevant candidate for guidance navigation and control (GNC) applications. The European Automated Transfer Vehicle (ATV) and many studies use lidar for tracking, rendezvous and docking but all of them use targets (retro-reflectors) on the tracked spacecraft or assume its shape known. A literature survey describes those studies in the background section (1).

In this thesis, it is desired to find the orientation of an unknown spacecraft with no retro-reflector (or marker) using lidar technology only. Since old spacecraft or space debris might not have such markers, and may be spinning out of control this study is essential for on-orbit maintenance and debris recovery/removal. The scenario is the following: a chaser equipped with an imaging lidar follows a target at a certain constant distance "taking pictures" or "snapshots" of the target at different instant, close to each other in time. Each snapshot results in a point cloud that is processed in order to retrieve features from the object (edge, boundaries, surfaces, corners). Those features, especially the corners, are then used to define a frame attached to the object. Comparing the frames from each point cloud allows retrieving the orientation as a function of time. The whole algorithm is implemented in MATLAB. The first step, described in Chapter 2, was to implement a lidar simulator with a model as an input and a point cloud as an output. Chapter 3 explains the processing of the point cloud. Finally, the definition of the reference frame as well as the overall results are presented in Chapter 4. The concept of this thesis being so new, little attention was paid to computation efficiency except in the lidar simulator part for which an extensive optimization of processing time was conducted. The focus was set on finding creative simple concepts. In the eventuality of the pursuing of the research, an optimization of the functions could would have to be performed.

Chapter 1

Background: Literature Survey

Until now lidar has been mostly used for geophysical observations. NASA has been a major investigator in lidar technology and applications from the 1960s starting with the development of ground-based satellite laser ranging systems for studying crustal dynamics and plate tectonics [1]. The first part of the section describes key steps in lidar development and testing over different Earth observation missions. The second part presents the current studies aiming at introducing lidar as an integral part of the GNC system for tracking, docking, safe landing and collision avoidance.

1.1 Earth Observation Experiences with Lidars

In the 1970s NASA put together groups to study the capabilities of lidar on satellite platforms. Because of the heavy-weight and high-power requirements for these early lidars, the obvious platforms for demonstrating lidar's capabilities were Spacelab and the Shuttle.

1.1.1 Lidar In-space Technology Experiment (LITE)

After some delays in its development, primarily due to the Shuttle Challenger mishap, the Shuttle Discovery flight of LITE took place for 11 days in September 1994 [16]. LITE is a three-wavelength backscatter lidar developed by NASA Langley Research Center. The goals of the LITE mission were to validate key lidar technologies for space-borne applications, to explore the applications of space lidar, and to gain operational experience which will benefit the development of future systems on free-flying satellite platforms. This flight was truly a pathfinder mission for future space lidars, and ushered in a new era of remote sensing from planetary orbit. It showed the science community the exceedingly important data that a space born lidar can provide [13]. LITE operated for 53 hours demonstrating the ability of lidar to probe between clouds and penetrate through optically thin clouds with high horizontal resolution, high sensitivity to aerosol measurements, and an excellent discrimination against noise because of laser spectral purity.

However until then, technology did not allow long duration mission. Long-lifetime, laser power efficiency, cooling and weight issues had to be solved if lidars were to fly for long-duration on Earth-orbiting spacecraft. In the late 1980s and 1990s diode-pumped and long-lived ND-YAG lasers, light-weight optics and structures, changed significantly the feasibility for lidar flights.

1.1.2 Mars Orbiter Laser Altimeter (MOLA)

In November 1996, the Mars Global Surveyor (MGS) mission was launched with MOLA aboard. This time of flight laser scanner was designed to map the Martian global topography and measure the height of water and carbon dioxide clouds [20]. MOLA was built by NASA Goddard Space Flight Center.

1.1.3 Geoscience Laser Altimeter System (GLAS)

GLAS is the first lidar instrument for continuous global observations of Earth [21]. From aboard the Ice Cloud and Elevation Satellite (ICESat) spacecraft, it makes atmospheric observations, including measuring ice-sheet topography, cloud and atmospheric properties. GLAS was successfully launched aboard the ICESat in 2003.

1.1.4 Cloud Aerosol Lidar and Infrared Pathfinder Satellite Observation (CALIPSO)

CALIPSO combines an active lidar instrument with passive infrared and visible imagers to probe the vertical structure and properties of thin clouds and aerosols over the globe. CALIPSO was built by Ball Aerospace and launched in 2006 on the CloudSat satellite. CALIPSO is a joint U.S. (NASA) and French (Centre National d'Etudes Spatiales/CNES) satellite mission with an expected 3 year lifetime. The lidar is designed to scan the atmosphere with green and infrared laser light and detect backscatter from clouds and aerosols.

1.1.5 Phoenix

The Mars mission Phoenix launched in August 2007 carried a meteorological station built by the Canadian Space Agency based on lidar technology. Together, Optech and MD Robotics (Canada) designed and built the meteorological lidar system for the 2007 NASA Phoenix Mars mission. Optech is the world leader in the development, manufacture and marketing of advanced laser-based survey instruments.

1.2 Future Projects

ESA planned various missions of Earth observation based on lidar technology for the coming years.

1.2.1 Atmospheric Dynamics Mission (ADM-Aeolus, 2009

As part as its Living Planet program, ADM-Aeolus will make novel advances in global wind-profile observation and will provide much-needed information to improve weather forecasting. It will carry a highly sophisticated instrument called ALADIN (Atmospheric Laser Doppler Instrument) to measure wind velocity with unequaled accuracy. The instrument emits short and high-energy pulses towards the atmosphere and analyses the Doppler shift of the backscattered signal for different altitudes. Recently the laser diodes, which are the core components of the mission's instrument, have successfully passed their long-lifetime test.

1.2.2 Water vApour Lidar Experiment in Space (WALES), 2010

The WALES mission will provide accurate profiles of water vapor contents. It consists of a single satellite in Sun-synchronous dawn-dusk orbit carrying a Differential Absorption Lidar (DIAL).

1.2.3 EarthCARE, 2013

EarthCARE is a joint European-Japanese mission addressing the need for a better understanding of the interactions between cloud, radiative and aerosol processes that play a role in climate regulation. It will use a high spectral resolution ATmopsheric LIDar (ATLID).

The lidars used for those three missions were developed by LIDAR Technologies

Ltd. which specializes in the design, manufacture and testing of lidar instrumentation.

Now that lidar is a well known and established technology for Earth atmosphere observation and terrain mapping, researchers and agencies are trying to integrate it as part as the navigation system on spacecraft. It is thought to be an efficient instrument for rendezvous and docking maneuvers, planetary and small body mapping, hazard avoidance and precision landing.

1.3 Lidar for Navigation Instrumentation

Lidars have many desirable characteristics and advantages: high spatial resolution, independence from lighting conditions, it avoids problems of scaling by measuring directly the range. A lidar sensor acquires thousands of range measurements of the target in its field of view and generates three-dimensional maps of the scanned object in sensor reference frame (RF). The lidar hardware has to be coupled with on-board autonomous software that can extract ”intelligent” data from the raw data so that higher-level forms of observables can be used at lower bandwidth and data rates in the on-board GNC system.

1.3.1 Neptec Design Group Ltd.

Neptec Design Group Ltd. is actively researching on imaging sensors. The development of the Laser Camera System (LCS), was Neptec's first entry into the world of 3D data acquisition. Focused on the development of a 3D tracking capability, the LCS was developed with a flexible two-axis steering that allows standard raster scanning for imaging and custom scan patterns for tracking targets. This capability has since become one of the significant advantages of the basic LCS design and is a fundamental feature of the new generation of Neptec scanners [11]. LCS flew on board Space Shuttle Discovery in 2001 on $STS - 105$. After the Columbia accident, LCS was chosen to be part of the Orbiter Boom Sensor System (OBSS) that will perform inspection of the Space Shuttle Thermal Protection System before re-entry. In 2005, Neptec revealed an interesting application of their Laser Camera Systems (LCS) called LASSO (LCS Algorithms for Spacecraft Servicing On-orbit) [17]. The project was funded by the Canadian Space Agency (CSA) under the Space Technologies Development Program (STDP). The 3D LASSO system is designed to perform real-time tracking and 6 degree of freedom pose estimation of target spacecraft from sparse and noisy 3D data and the shape of the spacecraft . The approach is compatible with any sensor capable of providing 3D data. The algorithms have been successfully tested with Neptec's LCS in a variety of test scenarios.

Still under development, Tridar is Neptec's newest promising lidar based sensor [5]. The TriDAR is a hybrid scanner combining the best features of the space qualified, near field LCS (based on triangulation) with a long range lidar system. Unlike pure lidar systems the TriDAR operates at distances ranging from 0.5 meters to over 2000 meters without sacrificing speed or precision at either end of the range. Neptec's 3D Automated Rendezvous and Docking Sensor system is based on two Neptec innovations: the TriDAR 3D sensor and Neptec's Intelligent 3D (3Di) software toolkit.

1.3.2 Jena-Optronik GmBH

Jena-Optronik developed the Rendezvous and Docking Sensor (RVS) for ESA's ATV and JAXA's HTV for docking with the ISS. An RVS prototype had already been successfully demonstrated in orbit during two campaigns of the Space Shuttles STS-84 and STS-86 docking to the MIR space station in 1997.

This year, the first ATV named "Jules Verne" has approached the International Space Station with a successful docking maneuver based on RVS. RVS uses the time of flight

(TOF) principle, operates at ranges from 1 to 1000 m and estimates the pose by tracking retro-reflectors on the ISS during the approach. Using targets imposes restrictions on the rendezvous and docking operations, and approach trajectories, and may require control over the target spacecraft. In some cases it is necessary to approach the target spacecraft from any direction or perform a fly around operation using visual feedback. The use of optical targets also introduces a failure mode when one or more targets may not be detectable due to damage. Jena-Optronik also designed a Lunar Landing 3D lidar [3].

1.3.3 Lidar based navigation algorithm

Researchers in Canada managed to remove the need of target but their algorithm requires the model of the spacecraft for pose estimation. Piotr Jasiobedzki et Al. have set a model and algorithm for autonomous rendezvous and docking using lidar. They developed a system that uses scanning lidar to estimate the pose of a spacecraft from which the shape is known. The main purpose of such study is servicing failed spacecraft. The fact that the spacecraft to be serviced has to be known is a big limit to the algorithm [15]. Canada through The National Research Council Canada, universities and Optech Inc. is very active in the field of lidar.

1.3.4 LIDAR Technologies Ltd.

In addition to atmospheric lidar, Technologies Ltd. is developing an imaging lidar for Landing on Mars. This is part of the Aurora program [7] for the Mars Sample Return (MSR) mission. Lidar has been identified as one of the most sustainable technologies for exploration [2]. Sample Return Mission is a complex mission which calls for five spacecraft: an Earth/Mars transfer stage, a Mars orbiter, a descent module, an ascent module and an Earth re-entry vehicle. Lidar is considered for hazard mapping, to improve landing accuracy and rendezvous maneuvers.

1.3.5 NASA

The future crew exploration vehicle, which is to replace the space shuttle and be used for a crewed mission to the moon, will most likely rely on a lidar sensor for its rendezvous and docking maneuvers. The lidar technique is being considered for providing critical distance, approach velocity, and relative orientation of the docking port during the rendezvous and docking maneuver. The precision and frequent update rate offered by the lidar could be key for mating the vehicle with the International Space Station and, in the case of the human mission to the moon, for mating the lunar crew module with the Earth re-entry vehicle that will be awaiting it in the moon orbit.

Currently, NASA is actively advancing the lidar technology for future lunar landing missions through its Autonomous Landing and Hazard Avoidance Technology (ALHAT) project. This program is developing three-dimensional imaging and Doppler velocity lidar technologies as part of the landing GNC system. The lidar sensors being developed under ALHAT will enable safe soft-landing of large robotic, cargo and crewed vehicles with a high degree of precision at the designated landing site under any lighting conditions.

Chapter 2

Lidar Simulator

2.1 Lidar Architecture

This section describes an imaging **L**ight **D**etection and **R**anging (lidar) system and its functional simulator. The lidar architecture section presents the basic principles of the lidar, together with a notional architecture. The simulation section presents the functional simulator of the lidar and the results of the simulated tests. Note that in equations, bold variables are vectors.

2.1.1 Principles

An imaging lidar is an electro-optical instrument employed to obtain 3D data from an object placed in its field of view. It usually consists of a laser source, a scanning mechanism, a detector and its associated focusing optics. Like the similar radar technology, which uses radio waves instead of light, the range to an object is determined by measuring the time delay or time of flight (TOF) between transmission of a pulse and detection of the reflected signal:

$$TOF = t_d - t_e = \frac{2D}{c} \tag{2.1}$$

Where:

D	=	Distance traveled
t_d	=	Detection time
t_e	=	Emission time
c	=	Speed of light in medium

Lidar can be classified by the relative position between the emitter and the receiver. A monostatic lidar is a system in which its transmitter and detector are in the almost same location. A near-monostatic lidar is defined as a system which is close to a monostatic system and the assumption of $r_{emitter} \approx r_{receiver}$ is valid [10]. To derive spatial data from the target object the direction of the pulse is modulated in two directions with the help of a scan mechanism, as illustrated in Figure 2.1. This scan mechanism consists of an azimuth scan mirror, an elevation scan mirror, and their associated electronics. Each scan mirror oscillates about its longitudinal axis a certain number of degrees. The projection of the path of the pulses on a plane perpendicular to the longitudinal axis of the laser source (x_L) describes a scan pattern. The pattern shown in Figure 2.1 is the so called "TV scan pattern" where the spot travels from left to right and top to bottom. The angular position of each mirror at the time of the firing of the laser pulse is measured by a shaft encoder and thus the direction of the pulse can be determined. For each of the firings of the laser pulse the direction and the distance to the target object are employed to generate a point cloud. The point cloud represents the three dimensional position vector (x, y, z) of each of the reflected pulses in a coordinate frame attached to the lidar.

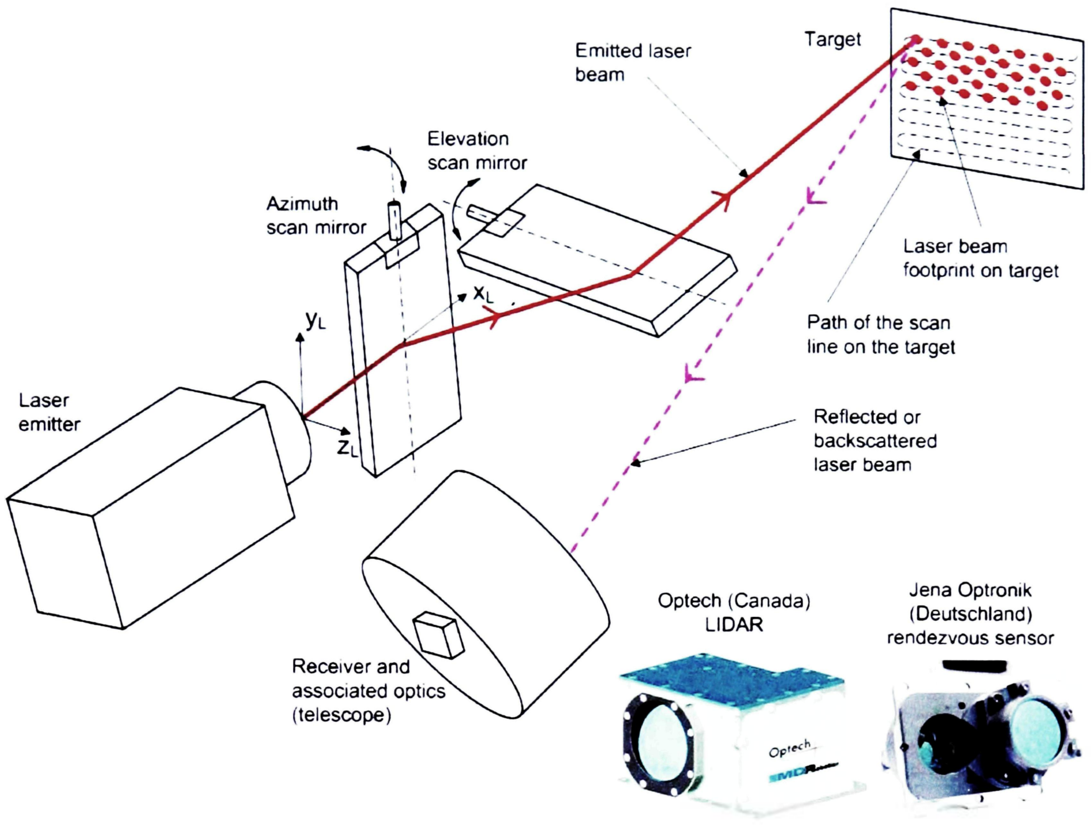

Emitted laser beam
Target
Elevation scan mirror
Azimuth scan mirror
Laser beam footprint on target
x_L
y_L
z_L
Path of the scan line on the target
Laser emitter
Reflected or backscattered laser beam
Optech (Canada) LIDAR
Jena Optronik (Deutschland) rendezvous sensor
Receiver and associated optics (telescope)

Figure 2.1: Lidar architecture. Example of hardware.

In addition to the range the lidar can also determine the intensity of the reflected pulse. In this study a variable has been included in the code for the intensity but not used. It is recommended that it is used later should someone continue the work.

2.1.2 Quality of measurement

The following parameters do NOT affect measurements:

- Day or night: laser radar is an "active illumination" technique that, unlike photography, does not depend on ambient illumination. It works during the day or at night.
- Target's angle of repose: laser measurements can be made to targets at any angle.

- Background noise and radiation: the laser is not affected by background noise.
- Temperature variations: laser measurements are based on the speed of light and are unaffected by temperature variations. However the electronic might operate only in a certain range.
- Vessel pressure and off-Gas layers: the laser is unaffected by pressure or vacuum variations, or off-gas layers.

The following parameters can affect measurements:

- Dust and vapor: laser measurements can be weakened by interacting with dust and vapor particles, which scatter the laser beam and the signal returning from the target. This principle is used for Earth atmosphere study presented in section 1.
- Sunlight and reflections and angle of measurement: a strong sunlight reflection off a highly reflective target may "saturate" a receiver, producing an invalid or less accurate reading. However, laser measurements are not usually affected by other reflections.
- Reflectivity of the object: highly reflective objects may saturate some laser detectors, while the return signal from low-reflectivity objects may occasionally be too weak to register as valid.

2.2 Lidar Simulator

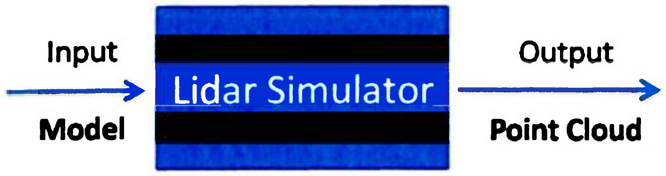


Figure 2.2: Block diagram of the lidar simulator.

The first step in creating a lidar simulator was to have a model of a spacecraft so the simulator could be tested. The model constitute the input of the system, its creation and manipulation are explained in the first section.

The two primary functions of a lidar are:

1. Produce and emit laser rays in defined directions to scan a field of view,

2. Compute TOF for each ray and deduce the distance to the scanned object.

The simulator must produce rays and find the distance between the origin of the ray and its intersection with the model. Those two functions are described in the two next sections. A ray generator was implement to create rays and store their directions. The intersection computation are performed by a simple ray tracer. The output of the simulator is a 3D point cloud.

2.2.1 General overview of the simulator

The diagram in Figure 2.3 shows the general organization of the lidar simulator. Each function is described in detail in the next sections.

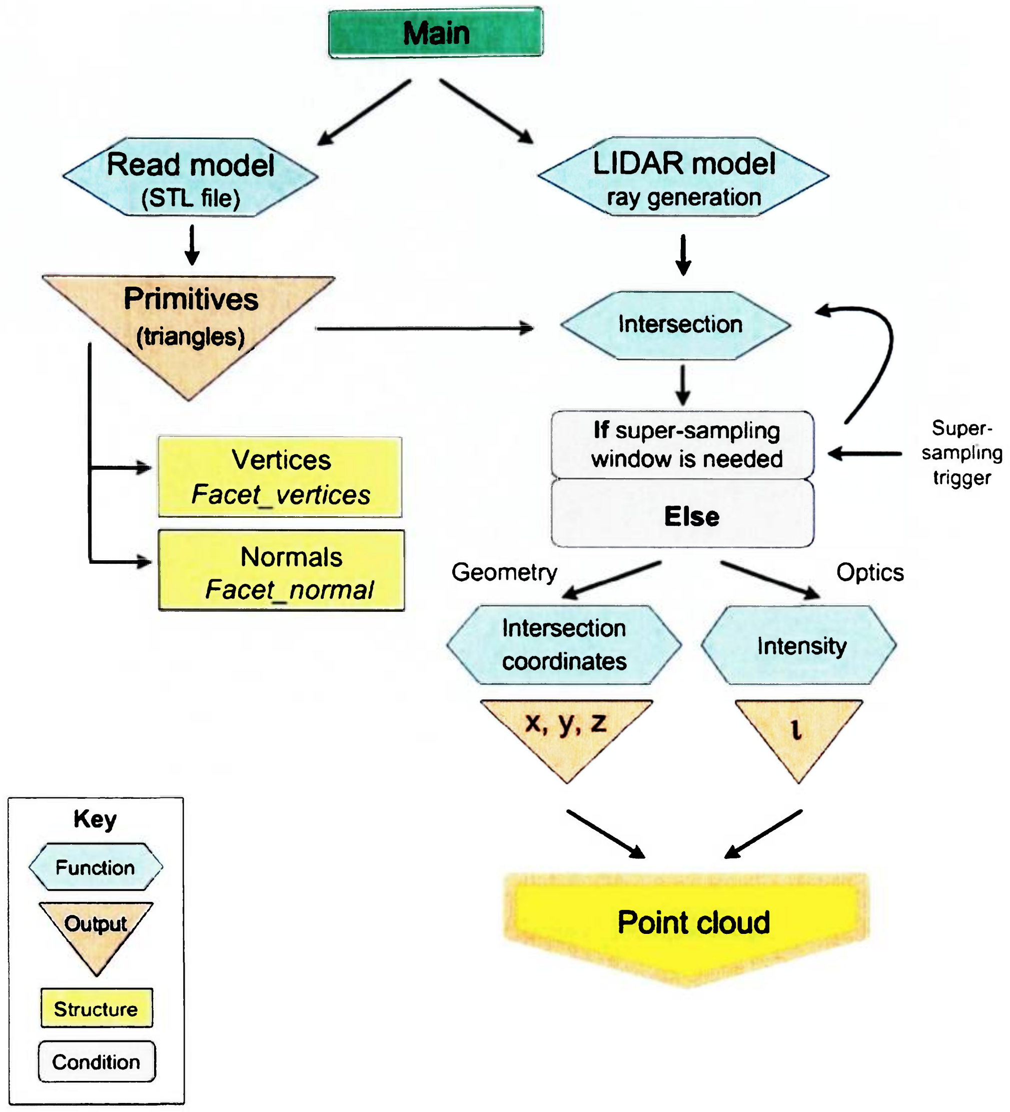


Figure 2.3: General structure of the lidar simulator.

2.2.2 Model

CATIA modeling

The models used for this study were made with CATIA V5. It is widely and internationally used, reliable, and relatively easy. For simplicity purpose the models were centered at (0, 0, 0) and later moved and rotated in a separate file. A simple cube as well as simplified spacecraft were designed (Figure 2.4).

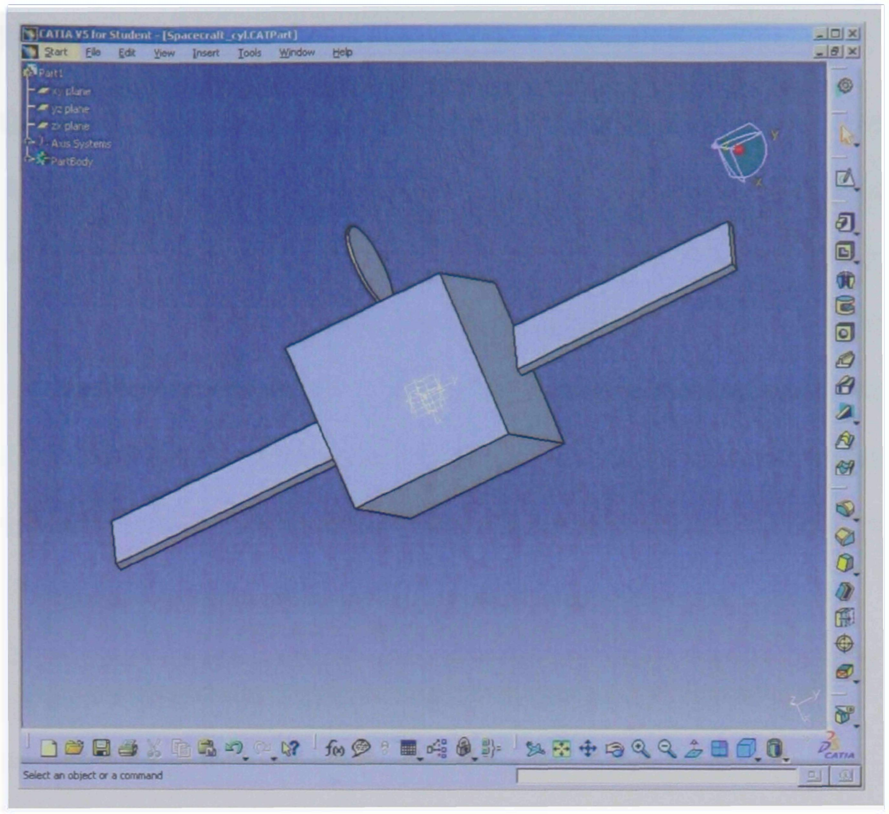

CATIA V5 for Student - [Spacecraft_cyl.CATPart]
Start File Edit View Insert Tools Window Help
Part1
xy plane
yz plane
zx plane
Axis Systems
PartBody
Select an object or a command

Figure 2.4: CATIA model of a simplified spacecraft.

A drawing of the model with all the dimensions can be found in Appendix A. The models are saved as *.STL* files. The format is described in next section.

STL format

STL is a file format native to the stereo lithography CAD software created by 3D Systems. An STL file is a triangular representation of a 3D surface geometry. The surface is tessellated logically into a set of oriented triangles (facets). Each facet is described by the unit outward normal and three points listed in counterclockwise order representing the vertices of the triangle. While the aspect ratio and orientation of individual facets is governed by the surface curvature, the size of the facets is driven by the tolerance controlling the quality of the surface representation in terms of the distance of the facets from the surface. The format of an STL file is given in Appendix C.

The next section describes the MATLAB routine that reads the model STL files.

STL reader

The STL reader function opens the STL file, counts the number of lines, deduces the number of facets and vertices, read each facet, stores its vertices and normal and closes the STL file.

Figure 2.5 shows the facets the spacecraft modeled in CATIA. This file is read by the STL reader and plotted using the command *patch* which allows to plot polygons given N vertices (three vertices give a triangle).

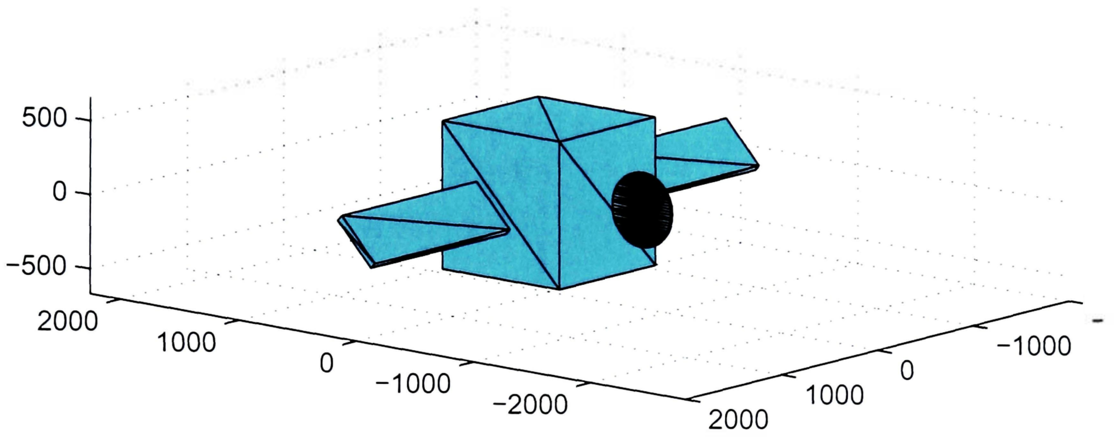


Figure 2.5: MATLAB rendering of the STL file of the spacecraft mode.

A reference frame is attached to the lidar (RF_L), the lidar being its origin. The models from CATIA are centered at zero meaning they are centered at origin of RF_L. In order for the object to appear in the lidar FOV, the model must be translated. Also since the goal is to retrieve attitude of the model it must be rotate. A separate function translates and rotates the target.

Orientation of the model

After the STL file is read, each facet is defined by three points and a normal stored in two different arrays (3D for the vertices, 2D for the normals). In the $f\_target\_rot\_trans$ function, the vertices of each facet are first rotated using a rotation matrix, and then translated. The user chooses the desired rotation and translation in the main command file. The input rotation is input in terms of Euler angle (α, β, γ) according to the (3, 2, 1) convention for simplicity. It is immediately converted to rotation matrix to perform the rotation of the object. The conversion Euler-Rotation matrix (inertial to object frame convention) is done as follow:

$$R_z = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}, R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\gamma) & cos(\gamma) \end{bmatrix}$$

Then, each point of the model is multiplied by the following rotation matrix:

$$\mathbf{P_{rotated}} = R_x * [R_y * (R_z * \mathbf{P_{initial}})]$$

The translation parameter is fed as an input to the function under the form of a vector with three components: translation in x-direction, translation in y-direction and finally translation in z-direction. As an example, let's consider a cube with a side of 1 m initially centered at (0, 0, 0). It desired to translate it 1.5 m along the x-axis and 1 m along y-axis (*translation* = [1500 1000 0] (mm)) and rotate it 45 degrees about z-axis (*rotation* = [45 0 0]).

In Figure 2.6, the green cube is the original model centered at (0, 0, 0). The yellow cube is the result of the translation and rotation. The new center is [1500 1000 0] due to the translation. The new orientation is [45 0 0] due to the rotation.

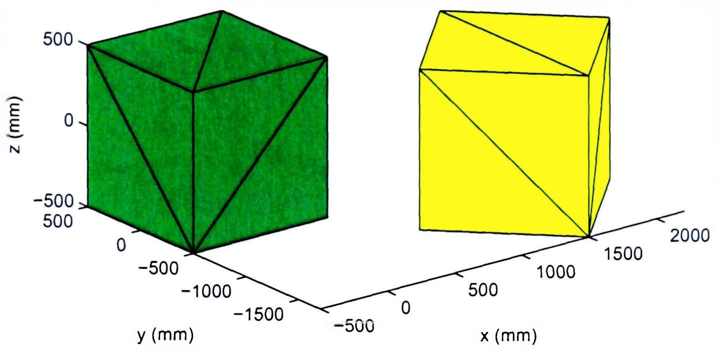


Figure 2.6: Illustration of translation and rotation for a cube.

2.2.3 Ray generator

Simulating a lidar means generating rays. A ray is a thin, straight line used to model a beam of light. It can be seen as a long thread that starts at an origin and is extended in a direction. The point (0, 0, 0) is taken as the origin, in this study the lidar beam. The rays are expressed in RF_L. Ray directions are generated by sweeping a field of view (FOV) through a double loop. The index i represents the elevation, the index j represents the azimuth.

As can be seen on Figure 2.7, the elevation is the angle in the (Y_L, Z_L) plane, the azimuth is the angle in the (X_L, Z_L) plane. The projections of the directions are given by:

$$d_x = \cos(\theta_E)\ \cos(\theta_A)$$
$$d_y = \sin(\theta_E)$$
$$d_z = \cos(\theta_E)\ \sin(\theta_A)$$

where:

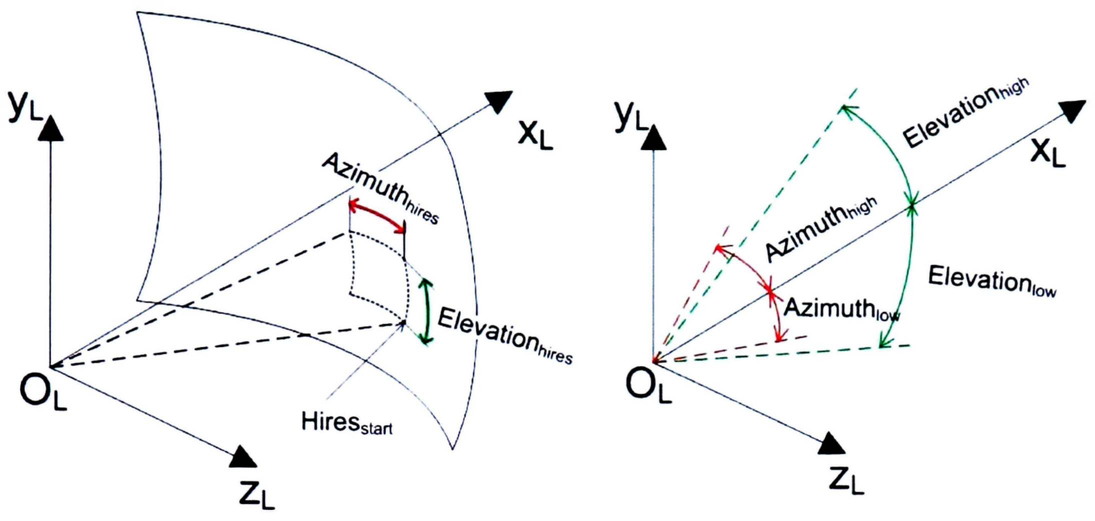

y_L
x_L
z_L
O_L
$\text{Azimuth}_{\text{hires}}$
$\text{Elevation}_{\text{hires}}$
$\text{Hires}_{\text{start}}$
y_L
x_L
z_L
O_L
$\text{Elevation}_{\text{high}}$
$\text{Azimuth}_{\text{high}}$
$\text{Elevation}_{\text{low}}$
$\text{Azimuth}_{\text{low}}$

Figure 2.7: Lidar simulator reference frame setting

d_x, d_y, d_z	=	Projections of the unit vector along the ray on RF_L
θ_E	=	Elevation angle
θ_A	=	Elevation angle

The coordinates thereby computed are stored in a 3D array.

In some cases it is desirable to have higher resolution on parts of a model to see details. This is done by introducing **H**igh **R**esolution (HR) windows. A high resolution window is an area of the FOV where the density of rays is higher. Dealing with HR windows is slightly more complicated than regular uniform resolution. For each HR window, the resolution is defined as a multiplier coefficient. If this coefficient is 2, the window will contain twice as much rays as in the rest of the FOV. The window itself is defined by a starting point and a span (both in azimuth and elevation). The code adjusts the input starting point so it matches an existing direction and from this direction, the span is swept in the same way the FOV is in the situation of basic resolution. When there is various HR windows, each window is stored in separated arrays grouped in a structure.

To illustrate this section, an example with the following characteristics was studied:

FOV = 50 × 50

Resolution in azimuth = 5

Resolution in elevation = 10

Number of high resolution windows = 2

Multiplier coefficient = 2

		Window 1	Window 2
El	Start	20	5
	Span	14	8
Az	Start	25	1
	Span	5	6

Table 2.1: High resolution windows parameters.

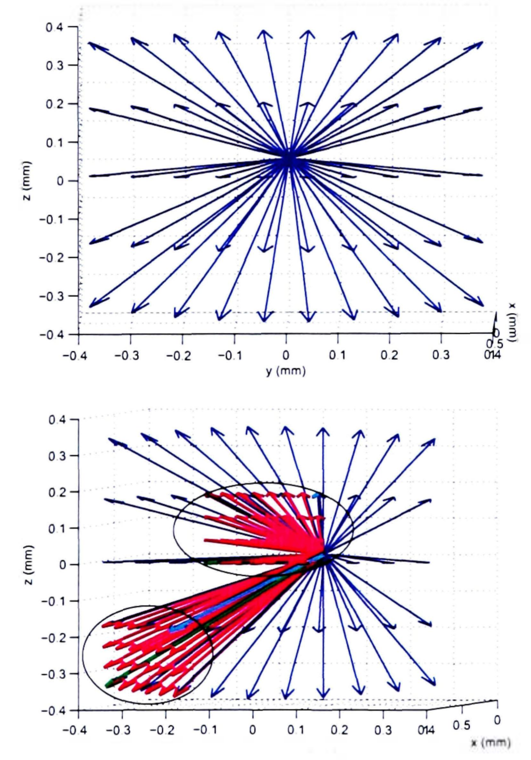


Figure 2.8: Ray generator results without (top) and with two high resolution windows(bottom).

2.2.4 Ray tracer

The output of a real lidar is the distance between points of an object and the laser source for each beam. In ray casting the visible surfaces of objects (parts of a scene that are visible to the camera) are found by casting rays of light from the light source to the scene and finding the closest intersecting objects. Ray tracing is an extension of ray casting in that it also describes what the visible surface looks like. Ray casting was used in the first version of the study presented in this thesis. However a variable named *Intensity* was added to the output distances for future improvement of the algorithm. The simulator hereby designed outputs the coordinates of the intersection between rays and the model triangular facets. Those coordinates are expressed in the lidar reference frame.

Choices and assumptions

As a reminder, the surfaces of the models are meshed with triangles when output from CATIA software as STL files. Those triangles are called primitives. A primitive is a basic shape easily defined and interpreted by computer.

Choice 1: The ray tracer will operate with only one type of primitive (triangle). Since most of the meshing software offer the possibility to mesh with triangle, this choice is not restrictive.

Choice 2: Only simple operations on primitive such as read specification of primitive (three points and a normal) and compute the intersection of the primitive with a ray are performed in the ray tracing code. The translation and rotation of the model are made at the model level in order to gain modularity, and computation time.

Choice 3: No shadow, reflection, texture mapping, color, and diffusion have been considered because it was not necessary at this stage of the project. Shadow and reflection are not relevant since the objective is to track one object at the time in space.

The simulated lidar is supposed to work in space where diffusion is not too much of a problem.

Principles

Time wise, ray tracing is a very heavy process. If n is the number of facets of the model, each ray has to be cast n times. For instance, if we have 200 facets, and 40 rays (which is not much for a real system), it means there is 40*200 = 8000 intersections to compute. Since many of those intersections do not exist or are not valid, tests are performed to eliminate them instead of wasting time in determining their inexistent intersection.
The tests implemented in this ray tracer are the following:

1. Is the ray intersecting the plane to which belongs the facet? If the ray is parallel to the facet then there is no need to continue.

2. Is the ray intersecting the plane behind the origin? If the spacecraft is behind the lidar, the lidar will not see the spacecraft. The simulator should not compute an intersection that is behind the origin.

3. Does the ray intersect various facets? In the case of a volume, the ray will intersect the front and the back of the model. In the case of a spacecraft, the body of the spacecraft might hide part of a solar panel, which is therefore not seen. In those cases it is desired to keep only the closest intersection.

The two main steps in ray tracing are:

1. Find intersection between plane containing facet and ray

2. Determine if this intersection point falls inside the current facet

Ray-plane intersection [6], [8]

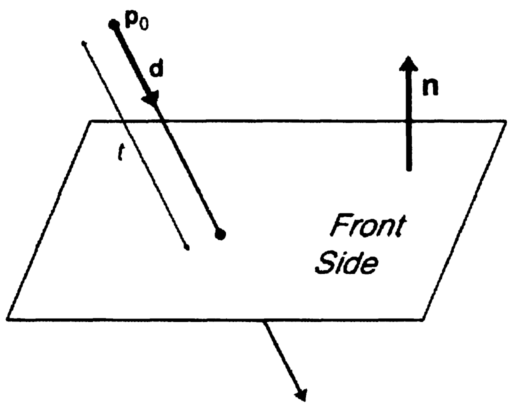


Figure 2.9: Intersection of a ray and a plane in 3D [6]

Expressing the ray with parametric description gives::

$$Ray\ :\ \mathbf{r}(t)\ =\ \mathbf{O}\ +\ \mathbf{D}\,t \qquad (2.2)$$

$$Plane\ :\ \mathbf{P}\cdot\mathbf{N}\ +d\ =\ 0 \qquad (2.3)$$

where:

$\mathbf{r}(t)$	=	Any point on the ray
$\mathbf{O}$	=	Origin of the ray
$\mathbf{N}$	=	Normal of the plane
$\mathbf{d}$	=	A parameter defining the plane such that $d = -V_0 \cdot \mathbf{N}$
V_0	=	Coordinates of a point belonging to the facet
$\mathbf{D}$	=	Direction of the ray

The parameter d is calculated by doing the dot product between a point ($\mathbf{V}_0$) and its normal. It was chosen to take the center of mass of each facet for $\mathbf{V}_0$. The evaluation of the parameter t corresponding to the intersection point can be obtained by substituting equation 2.2 into **P** (equation 2.3) :

$$t = -\frac{d + \mathbf{N}\cdot\mathbf{O}}{\mathbf{N}\cdot\mathbf{D}} \tag{2.4}$$

If the denominator is equal to zero, the ray and the plane are parallel. If it is positive, the ray and the normal are in the same direction; since the normal points outward, the ray intersects the surface through its back. This happens in the case of a volume, the ray enters the volume with a valid intersection but exits it with an invalid one. If $t \leqslant 0$, the intersection is behind the plane, and it is rejected as well.

Ray-triangle intersection

If a ray-plane intersection is found, the ray tracer proceed to step two: the intersection with the primitive. It determines if the point of intersection between the ray and plane falls inside the current facet or not. The *dominant axis method* was used. Assume three vertices V_1, V_2 and V_3 from a triangle. In the barycentric space, a point P is given by:

$$\overrightarrow{V_1P} = \alpha\overrightarrow{V_1V_2} + \beta\overrightarrow{V_1V_3} \tag{2.5}$$

Any point in the plane of a triangle can be expressed as the weighted average of the vertices of the triangle. The weights (α and β) are known as barycentric coordinates. The barycentric coordinates of a point inside a triangle will be in the range [0, 1]. Any point outside the triangle will have at least one negative coordinate. P is in the triangle if and only if: $\alpha \geq 0, \beta \geq 0$ and $\alpha + \beta \leq 1$

The following analysis is explained for one triangle, note that the process is repeated for each facet of the model.

Equation 2.5 has three components. To reduce the system, it is desired to project the triangle onto one of the primary plane (OX_LY_L), (OY_LZ_L) or (OZ_LX_L) as 2D treatments are cheaper and faster. If the triangle is perpendicular to one of these planes, its projection

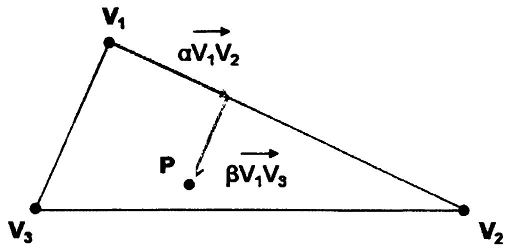

V_1
$\alpha\overrightarrow{V_1V_2}$
P
$\beta\overrightarrow{V_1V_3}$
V_3
V_2

Figure 2.10: Barycentric coordinates of a point in a triangle. [8]

will be a single line. To avoid this problem, and make sure the projections are as large as possible, the dominant axis of the normal vector is found. The plane perpendicular to that axis is used for the projection. For example if the normal to a plane is (0, -5, 3) then y is the largest coordinate and the triangle is projected onto the XZ plane, X and Z being the dominant axis. By using the dominant axis method the projection with the greatest projected area is obtained, resulting in the best precision for the rest of the calculations. A separate function was written to find those axes.

For greater detail on the theory behind this method and how it is implemented, refer to the code of reference [8].

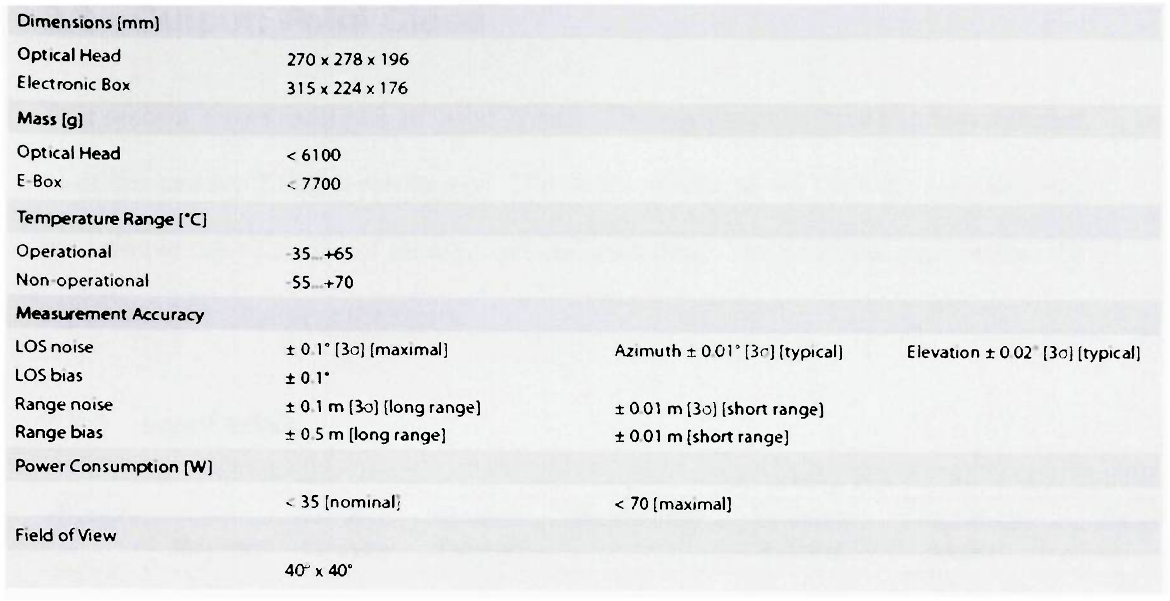
Algorithm: pseudo code

This pseudo code summarizes the ray tracing ideas described earlier.

```
For each elevation
   For each azimuth
      For each facet
         Compute Vd
         If Vd < 0 : there exists a valid intersection
            Compute the plane parameter d
            Compute parameter t (distance origin - intersection)
            If t > 0 (the intersection is on the front side)
               Compute the coordinates of the intersection point
               Find dominant axis
               Compute alpha
               If alpha > or =  0
                  Compute beta
                  If beta > or = 0 and alpha + beta < OR = 1
                     If no closer intersection has been found
                        Store intersection in 'lidar_out' array
                     EndIf
                  EndIf
               EndIf
            EndIf
         EndIf
      EndFor
   EndFor
EndFor
```

Noise

The ray tracing code allows to introduce errors in measurement of the range (t) and the direction of the rays (elevation and azimuth). A Gaussian noise based on Jena-Optronik RVS Lidar specifications was implemented (Figure 2.11).



Dimensions [mm]			
Optical Head	270 x 278 x 196		
Electronic Box	315 x 224 x 176		
Mass [g]			
Optical Head	< 6100		
E-Box	< 7700		
Temperature Range [°C]			
Operational	-35...+65		
Non-operational	-55...+70		
Measurement Accuracy			
LOS noise	± 0.1° [3σ] [maximal]	Azimuth ± 0.01° [3σ] [typical]	Elevation ± 0.02° [3σ] [typical]
LOS bias	± 0.1°		
Range noise	± 0.1 m [3σ] [long range]	± 0.01 m [3σ] [short range]	
Range bias	± 0.5 m [long range]	± 0.01 m [short range]	
Power Consumption [W]			
	< 35 [nominal]	< 70 [maximal]	
Field of View			
	40° x 40°		

Figure 2.11: Jena-Optronik RVS specifications. [14]

The MATLAB function *randn* was used to generate this noise. *randn* generates normally distributed random numbers.

$$R_N = R_I + \frac{0.1}{3} \times k \tag{2.6}$$

where:

R_N	=	Range with noise
R_I	=	Ideal range
k	=	Random number

The output of *randn* is multiplied by the standard deviation (0.1), and added to the desired mean (range with no error). Since the variance was given at three sigma in the specifications, the standard deviation is divided by three. In addition to the range noise, the ray directions in azimuth and elevation are not perfect. Noise was added in the same fashion in the ray generator.

2.3 Output: Point Cloud

This section shows examples of point clouds. The spacecraft is shown in two different positions and for different resolutions. The divers options of the software are illustrated on a simple cube for ease of reading and understanding. The first table summarizes the inputs used for obtaining the figures.

2.3.1 Input table

Figure	FOV	Az. Res.	El. Res.	HR Window	Rotation	Translation	Noise
	Degree	Degree	Degree		Degree	Meter	
2.12	4×4	301	301	No	(6, 10, 0)	(500, 0, 0)	No
2.13	4×4	501	501	No	(45, 20, 0)	(500, 0, 0)	No
2.14	40×40	70	70	No	(45, 20, 0)	(3, 0, 0)	No
2.15	40×40	70	70	Yes	(45, 20, 0)	(3, 0, 0)	No
2.17	40×40	70	70	No	(45, 20, 0)	(3, 0, 0)	Yes (Az/El)
2.16	40×40	70	70	No	(45, 20, 0)	(3, 0, 0)	Yes (Range)

Table 2.2: Ray tracer inputs used to obtain the next figures.

2.3.2 Figures

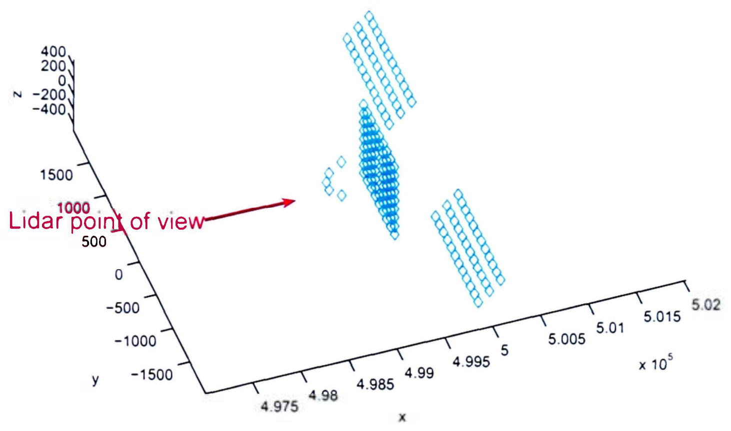


Figure 2.12: Rendering of the spacecraft at low resolution.

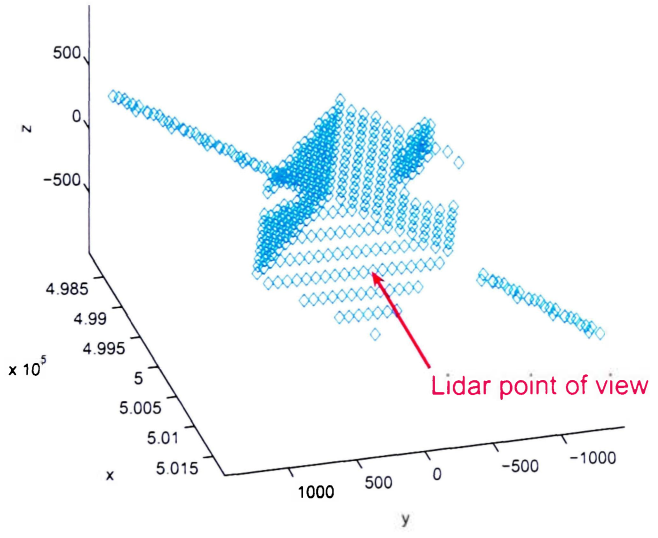


Figure 2.13: Rendering of the spacecraft oriented (45, 20, 0).

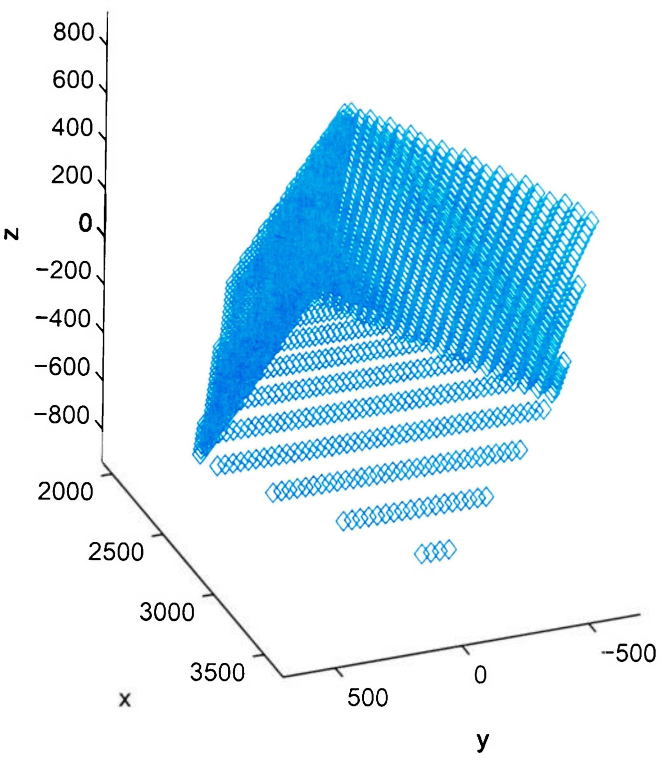


Figure 2.14: Rendering of the reference cube.

High resolution window: The high resolution window starts at 5.5 degrees in azimuth and 9 degrees in elevation. The span in azimuth is 12.5 degrees, the span in elevation is 16 degrees.

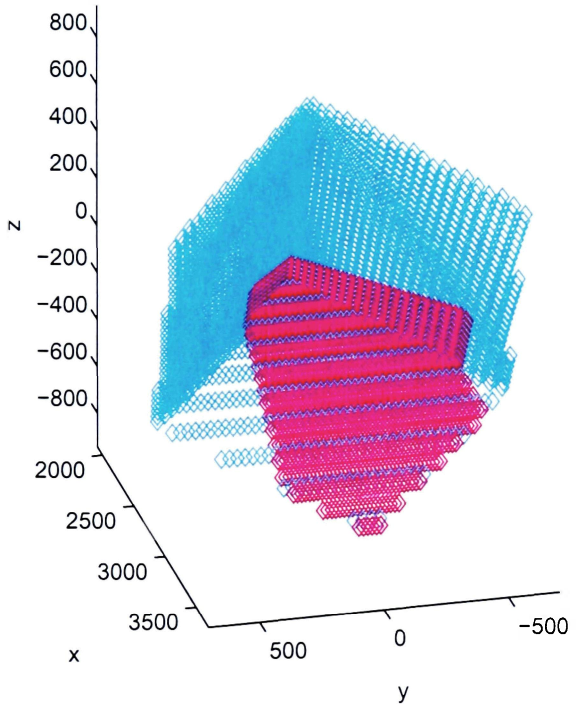


Figure 2.15: Cube with one high resolution window.

Error in range measurement: The error in the range measurement is added in the ray tracing part when the parameter t is calculated. Referring to the RVS specifications (Figure 2.11), a (3σ) 0.01 meter noise was added to the computed t. As seen on Figure

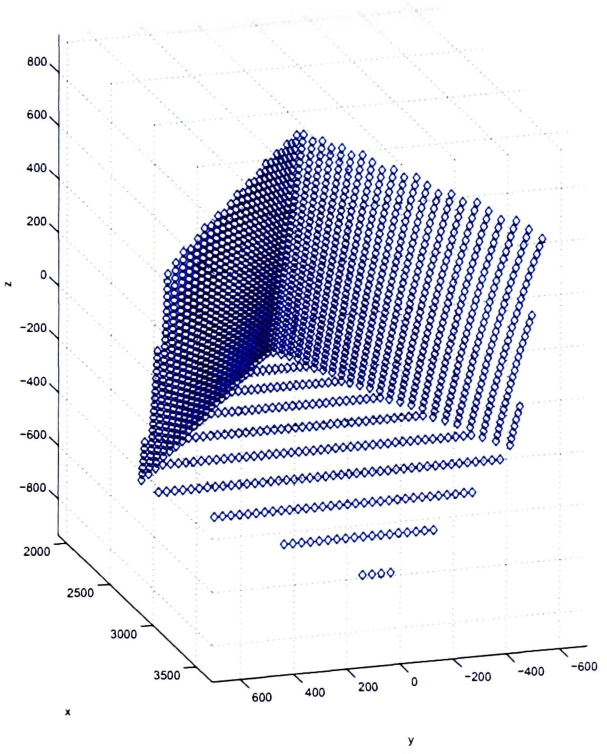


Figure 2.16: Cube with noise in range measurement.

2.16, adding an error in the measurement of the range does not produce visible change in the results. The parameter t corresponds to the distance between the origin and the facet-ray intersection point. This distance is in the order of the distance lidar-spacecraft. An error of 0.01 m is very small relatively to t. This error is not of great concern especially when operating in long range. If the project was to be continued, further investigation should determine the impact of such error in short range operations.

Errors on azimuth and elevation Errors in azimuth and elevation are introduced in the ray generator. The error value at 3σ is $+/-0.01^{\circ}$ for azimuth and $+/-0.02^{\circ}$ for elevation (Refer to Figure 2.11).

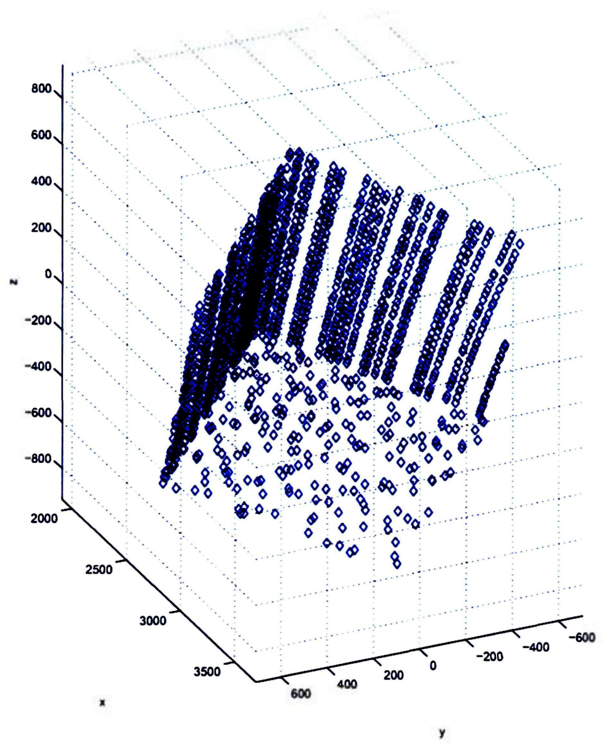


Figure 2.17: Cube with noise in azimuth and elevation measurement.

These errors have more impact on the point cloud but the overall shape of the object is still clear.

Errors and high resolution window: The following figure is obtained with noisy range measurement as well as noisy azimuth and elevation. A high resolution is added.The high resolution window has the same characteristics as previously.

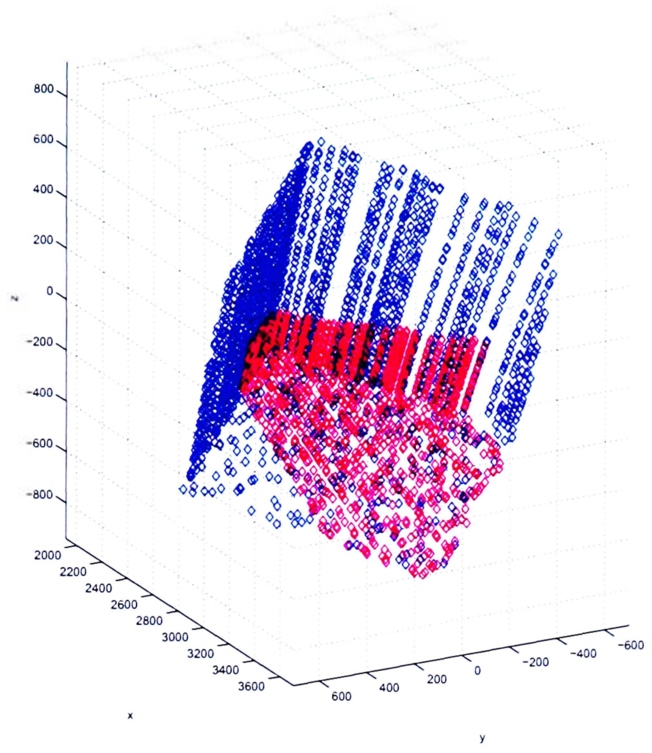


Figure 2.18: Noise in range, azimuth, elevation and high resolution window.

2.3.3 Conclusion

The different concepts behind the model creation and its processing to obtain a point cloud have been described in details, some more information can be found in the pseudo codes in Appendix C. The processing steps tackled in this chapter can be summarized in the following sequence:

1. Read model

2. Rotate and translate the model

3. Generate ray directions for a defined FOV and resolution

4. Ray tracing

The ray tracer has been thoroughly tested, on different shapes, at different distances, for different orientations and resolutions... The whole sequence described above was put into a loop so that simulations for different orientations of the same object could be run automatically. The output point clouds are saved into a folder.

Next chapters describes how features can be retrieved from 3D point clouds.

Chapter 3

Point Cloud Processing

This part has been implemented commonly by the author and Dr. Bodgan Udrea, her advisor.

The output of the ray tracer gives a 3D point cloud with valid and invalid points. Each ray has either a valid intersection with the model, or no/invalid intersection. The valid points are labeled with a flag equal to 1, the invalid ones have a 0 flag. The first step in processing the point cloud is to retrieve only valid points. Then it is necessary to simplify the data set. This Chapter starts with the description of a very useful MATLAB function called *Nearestneighbor* widely used throughout the study, follows an explanation of features detection such as edges, boundaries and surfaces.

It is important to define two terms: *edges* are at the intersection of two visible/detected surfaces. *Boundaries* are the points at the extremity of the scanned object that are the intersection of one visible and one hidden surfaces. Each step of the process is illustrated with the cube model for simplicity. An example of the spacecraft point cloud processing is given at the end of the chapter.

3.1 Nearest Neighbor Routine

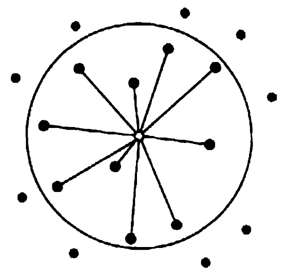


Figure 3.1: Nearest neighbor illustration.

The nearest neighbor function used in the present study was written by Richard Brown (Copyright 2006) and downloaded from the Mathwork File Exchange. It finds the nearest neighbors by Euclidean distance to a set of points of interest from a set of candidate points as illustrated in Figure 3.1. The points of interest are specified as a matrix of points. The *nearestneighbour* function can be used to search for k nearest neighbors, or neighbors within some distance (or both). A more detailed description of the function can be found in appendix D.1.

3.2 Edge Detection Routine

The edges are found using the surface variation method described by Pauly in "Efficient Simplification of Point Sampled Surfaces". The method will be explained in this section, for more details refer to [12] and [4]. An eigenvalue analysis of the covariance matrix of a local neighborhood is performed. Covariance analysis is often a starting point in classification of point clouds. It is performed by determining the covariance matrix for a local neighborhood surrounding the point of interest referred to as index point.

3.2.1 Covariance matrix

Covariance is a measure of how much two variables change together:

$$cov(x, y) = E((x - \mu)(y - \nu))$$

where μ and ν are the expected values for the random variables x and y, noted respectively $E(x) = \mu$ and $E(y) = \nu$.

The covariance matrix is a matrix of covariances between elements of a vector. It is the natural generalization to higher dimensions of the concept of the covariance of a scalar-valued random variable.

If entries in the column vector $X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix}$ are random variables, each with finite variance, then the covariance matrix is:

$$\Sigma = COV(X) = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 \end{bmatrix}$$

The diagonal terms are the variance whereas the other terms are covariance.

$$\sigma_{ij} = \text{cov}(X_i, X_j) = (X_i - \mu_i)(X_j - \mu_j)$$

where $\mu_i = \text{E}(X_i)$ is the expected value of the i^{th} entry in the vector X.

This is equivalent to: $\Sigma = \text{E}\left[(\mathbf{X} - \text{E}[\mathbf{X}])(\mathbf{X} - \text{E}[\mathbf{X}])^{\text{T}}\right]$

In our case, the covariance matrix is defined for a sample point **P** and its neighborhood N (in the sense described in previous section 3.1) is given by:

$$C = \underbrace{\begin{bmatrix} P_{i_1} - \bar{P} \\ \cdots \\ P_{i_k} - \bar{P} \end{bmatrix}^T}_{3 \times k} \cdot \underbrace{\begin{bmatrix} P_{i_1} - \bar{P} \\ \cdots \\ P_{i_k} - \bar{P} \end{bmatrix}}_{k \times 3} \tag{3.1}$$

where $\bar{P}$ is the centroid of the neighbors P_{i_j} of point P_i.

Since C is a 3×3 symmetric, positive semi-definite matrix, all eigenvalues λ_j are real-valued and the eigenvectors $\mathbf{v_j}$ form an orthogonal frame, corresponding to the principal components of the point set defined by N. The λ_j measure the variation of the P_i, $i \in N$, along the direction of the corresponding eigenvectors. The total variation, i.e. the sum of squared distances of the P_i from their center of gravity is given by:

$$\Sigma|P_i - \bar{P}|^2 = \lambda_0 + \lambda_1 + \lambda_2,\ i \in N \tag{3.2}$$

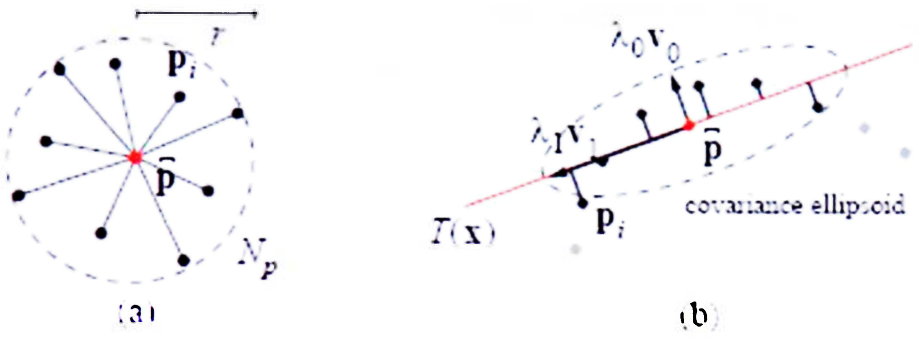


Figure 3.2: (a) Local neighborhood, (b) Covariance analysis. [12]

Assuming $\lambda_0 \leq \lambda_1 \leq \lambda_2$, it follows that the plane $(T(x) : (\mathbf{x} - \bar{\mathbf{P}}) \cdot \mathbf{v_0})$ through $\bar{\mathbf{P}}$ minimizes the sum of squared distances to the neighbors of **P**. Thus $\mathbf{v_0}$ approximates the surface normal $\mathbf{n_p}$ at **P**, or in other words, $\mathbf{v_1}$ and $\mathbf{v_2}$ span the tangent plane at **P**. The smallest eigenvalues λ_0 describes the variation along the surface normal (associated to the eigenvector $\mathbf{v_0}$). λ_0 estimates how much the points deviate from the tangent plane. The surface deviation at point **P** in a neighborhood of size N is defined as:

$$\sigma_N(P) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{3.3}$$

Note that if $\sigma_N(P) = 0$ then all the points lie in the plane.

3.2.2 Implementation

The function *find_edge_points* processes the valid points of the lidar simulator data and output for each point, the surface variance and the normal to the surface, plus a label. The default label is set at default (-1) and becomes 0 if the point is on a smooth surface, 1 if it is on an edge. It starts by calculating the surface variation of a neighborhood of a certain size. The neighborhood is then grown with a certain increment until the change in the surface variation is larger than a threshold. The routine then decreases the size of the neighborhood by the increment and starts incrementing it by one point at a time. It stops at the point where the surface variation threshold is exceeded and it labels the point as an edge point. The pseudo code can be found in appendix D.2. The covariance matrice computation and the eigenvalue analysis are done in a separate function for modularity. It computes the matrix (MATLAB*cov*), find the eigenvalues (MATLAB *eig*), find the smallest eigenvalue, calculate the surface variance and the normal to the surface (the eigenvector associated with the smallest eigenvalue).

3.2.3 Influence of input parameters

Threshold of the surface variance value

If the surface variance calculated exceeds this threshold, there is an edge point in the neighborhood. If it smaller then all the neighbors belong to a smooth surface. The threshold must be small enough to have acceptable accuracy. On Figure 3.3 the light blue dots are the lidar raw data, the dark blue one are the detected edge points.

The simulations are performed with a threshold of 10^{-5} For bigger values, many edge points are labeled as unknown (label = -1). Having smaller values does not change the results as can be seen in Figure 3.3. Generally, this threshold is difficult to obtain when the density of the neighborhood is not consistent throughout the point cloud. This

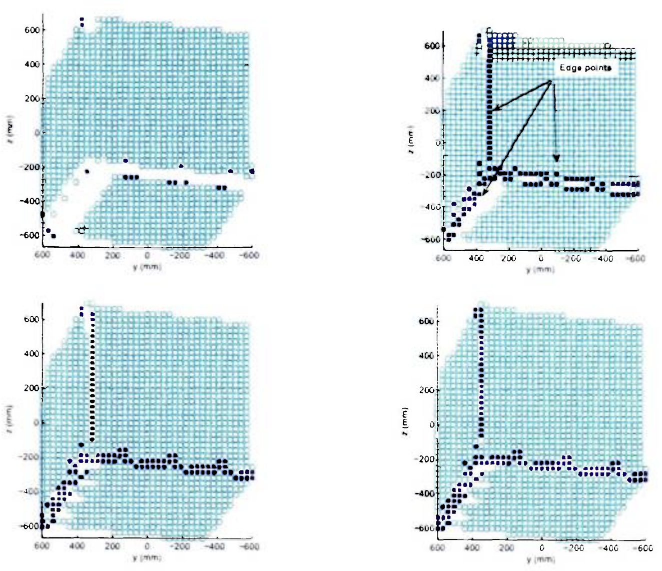


Figure 3.3: Edge detection for different thresholds: 10^{-1}, 10^{-2}, $\mathbf{10^{-5}}$, 10^{-10}. From left to right and top to bottom

is the case in this study since error in range and angle measurement are present. Work done in reference [10] allows approximating a theoretical threshold value for each point based on the scanners origin and attributes. This is out of the scope of this thesis.

Influence of the "maximum size of neighborhood"

The more neighbors, the more time the simulation takes with no obvious gain in accuracy as observed on Figure 3.4.

Maximum Size of Neighborhood	Simulation Time
5	4.5
10	6.2
16	7.5
50	27.8

Table 3.1: Simulation time for different size of neighborhood

In general, the size of the neighborhood should be chosen based on the resolution of the scan and the level of detail. This way, there is sufficient sample to perform the calculations but small details are not smoothed over.

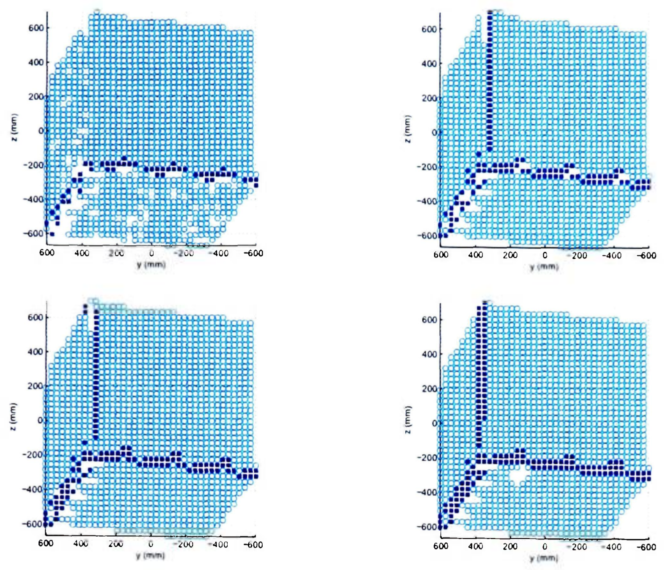


Figure 3.4: Edge detection for different maximum number of neighbors: 5, 10, **16**, 50. From left to right and top to bottom

3.3 Boundary Detection

Boundary points are those points in the point cloud which have no neighbor. i.e., the points around which the lidar has detected an invalid return. Those points are initially labeled as surface points however it is possible to distinguish them because of a different distribution of the points in the neighborhood. If the index point lies on the boundary, when the neighborhood is projected onto the the local best fit plane, the distribution of the neighborhood takes an elliptical shape whereas an interior point has a more circular distribution. This can also be seen in the difference of the two largest eigenvalues since they represent the variance in the principal directions on this plane. A small difference will represent an interior point, and a large difference represents a boundary point [4]. Since the distribution of the point cloud is not consistent, this method was thought to be too sensitive. A simpler and more robust technique is to examine the position of the index point relative to the neighborhood centroid. If the difference is large, the index point is close to a boundary and on one side there is no neighbors. This can be implemented by defining a confidence region around the centroid and test to see if the projected index point is outside this region. For this, the eigenvalues and a χ^2- test are used in the following equation:

$$\frac{e_{1_i}{}^2}{\lambda_1} + \frac{e_{2_i}{}^2}{\lambda_2} \leq \chi^2 \tag{3.4}$$

Where

λ_1 and λ_2	=	Two largest eigenvalues
e_{1_i} and e_{2_i}	=	Project coordinate system described below

$$\mathbf{e}_{1_i} = \mathbf{u}_1 \cdot (\mathbf{P}_i - \bar{\mathbf{P}}) \tag{3.5}$$

$$\mathbf{e}_{2_i} = \mathbf{u}_2 \cdot (\mathbf{P}_i - \bar{\mathbf{P}}) \tag{3.6}$$

$\mathbf{u}_1$ and $\mathbf{u}_2$ are the eigenvectors associated with the two largest eigen values. $\mathbf{P}_i$ is the index point, $\bar{\mathbf{P}}$ is the centroid of the neighborhood.

Pearson's χ^2- test is the original and most widely-used χ^2- test. It is used to assess two types of comparison: tests of goodness of fit and tests of independence. In this study it is used for the first type. A test of goodness of fit establishes whether or not an observed frequency distribution differs from a theoretical distribution.

3.3.1 Implementation

The χ^2- test steps requires a threshold and is implemented as follow:

1. Compute $\mathbf{e}_{1_i}$ and $\mathbf{e}_{2_i}$ according to Formula 3.5 and 3.6.

2. Compute χ as follow:

$$\chi = \sqrt{\frac{\mathbf{e}_{1_i} \cdot \mathbf{e}_{1_i}}{\lambda_1} + \frac{\mathbf{e}_{2_i} \cdot \mathbf{e}_{2_i}}{\lambda_2}} \tag{3.7}$$

3. Compare χ with a threshold.

   If χ is bigger than the threshold, the index point is a boundary point

   Otherwise it is a surface point

After the boundary points have been identified the functions passes one more time through the cloud to perform a clean-up. If a boundary point has an edge point as its nearest neighbor the label of that boundary point is reset to default (-1). This tends the eliminate the boundary point that show in between the edge points in regions with low point density. As will be seen in the next chapter, the situation arises for some orientations of the spacecraft but is not an issue at this stage of the project.

3.3.2 Influence of input parameters

The smaller the threshold the more boundary points are detected. Time is not affected. The red points are the detected boundary points.

Threshold of the χ parameter

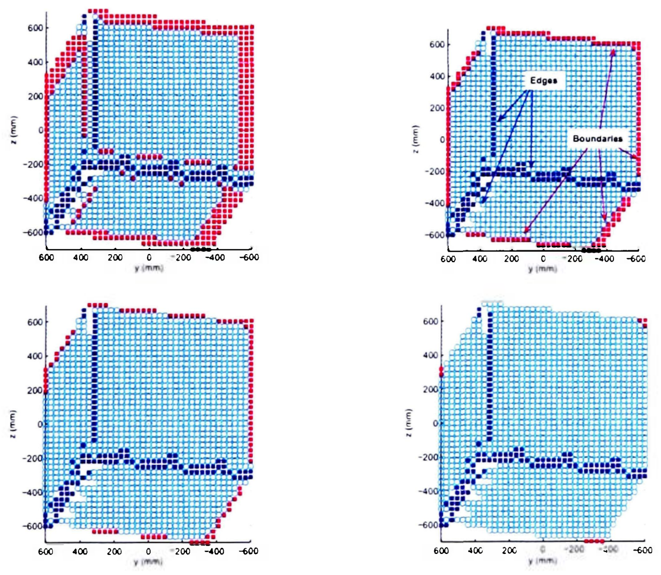


Figure 3.5: Boundary detection for different χ threshold: 0.1, **0.6**, 1, 1.6. From left to right and top to bottom

Influence of the "maximum size of neighborhood"

Figure 3.5 shows boundaries detected with a maximum neighborhood size of *25*. This is the value used throughout the study since bigger value increases the simulation time and outputs too many boundary points as seen on Figure 3.6. In the other hand, if the neighborhood is too small, some boundary points are missing and accuracy decreases.

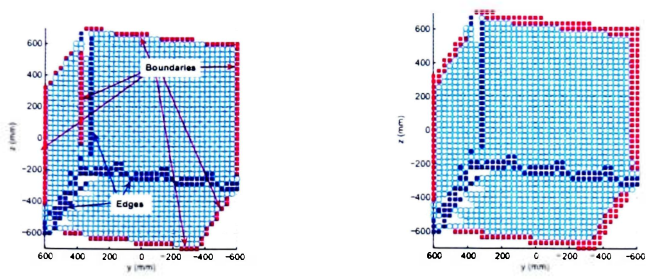


Figure 3.6: Boundary detection for different maximum neighborhood size: 10, 35.

3.4 Surface Detection and Labeling

So far the points are labeled as follow: the default value is *-1*, *0* for smooth surfaces points, *1* for edge points and *2* for boundary points.

This section deals with the identification of the different surfaces. Given the normal to the faces at each surface point it is possible to determine the points belonging to the same surface. A tolerance for the direction of normal and a minimum number of points which can define a surface are defined. If the total number of points with the same normal is less than this minimum, the points receive a special label meaning it was not recognized as edge, nor boundary, nor surface point. The segmentation is done in two steps:

1. Region growing: It starts by selecting an arbitrary points that has been classified as on a smooth surface. A neighborhood is progressively built around that point

until a boundary or edge point is met.

2. Clustering: The purpose of this second loop over the faces defined in step 1 is to separate surfaces that have the same normal but are not one surface. For example, for a spacecraft that has two symmetrical solar arrays, the points on the panels have the same normal but do not belong to the same part. The MATLAB function *Clusterdata* groups points into clusters. Since this function is used in the next chapter as well, a brief description is given here. For further details refer to appendix D.5.

MATLAB *clusterdata* function

Clustering is the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait often proximity according to some defined distance measure . Data clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis and bioinformatics [19]. MATLAB *Clusterdata(X)* command allows to group points from a point cloud into clusters. The function accepts many parameters as input:

- In the present study the data clustering algorithm is **hierarchical** meaning it finds successive clusters using previously established clusters.

- An important step in any clustering is to select a distance measure, which will determine how the similarity of two elements is calculated. This will influence the shape of the clusters, as some elements may be close to one another according to one distance and further away according to another. The **Euclidean** distance is used.

- Finally, instead of defining a distance threshold below which two points are considered being part of the same cluster, a **maximum number of clusters** was used. This maximum number of cluster must at least be the number of expected surfaces (3 for the cube, 6 for the spacecraft).

3.4.1 Influence of surface normal tolerance

If the surface normal tolerance is too small many surfaces are detected. If it is too big surfaces might be merged. This parameter really depends on the type of structure observed. For the cube a large tolerance is fine because the surfaces have very different normals. The best results are obtained for variable tolerances between 10^{-1} to 10^{-3} depending on the density of the point cloud. The study was performed with a tolerance of 10^{-3} which is the best value form most of the orientations of the cube and spacecraft.

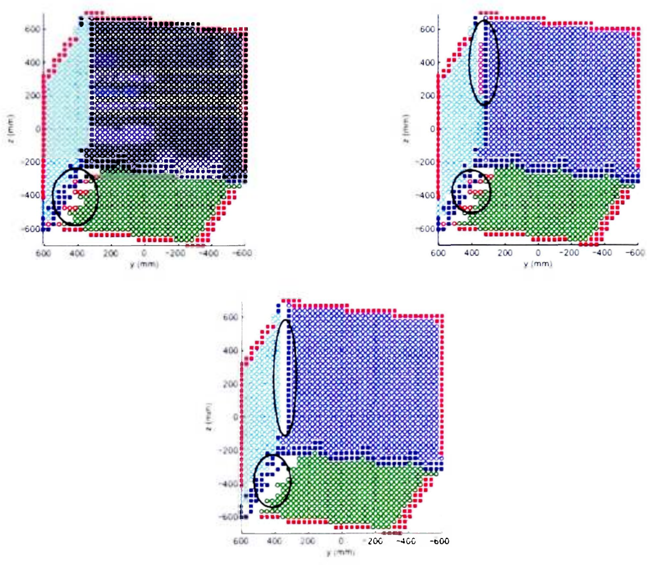


Figure 3.7: Surface segmentation for different surface normal tolerances: $\mathbf{10^{-1}}$, 10^{-3},10^{-5}.

From left to right and top to bottom

The number of maximum cluster does not have any influence on the results.

3.5 Summary and Results

3.5.1 General results

Each step of the point cloud processing was illustrated by the cube model throughout the chapter. In this section the results on the spacecraft are presented. The choice for parameters are summarized in table 3.2. A picture illustrates the results.

	Parameter	**Value**
Ray tracing		
	FOV	2×2
	Resolution	551 in El. and 551 in Az.
	HR windows	No
	Translation of model	(500, 0, 0) meter
	Rotation	Variable
	Error	Variable
Point cloud processing		
	Threshold surface variance	10^{-5}
	Threshold χ^2 test	0.8
	Surface normal tolerance	10^{-3}
	Maximum neighborhood for edge detection	16
	Maximum neighborhood for χ^2 - test	25
	Maximum number of clusters	10

Table 3.2: Summary of optimal parameters

As seen on Figure 3.8 the model of the antenna which is a thin cylinder has points recognized as boundary and some recognized as edge points. Further work could be done for improving detection of shapes without edge. It took about five minutes to perform the ray tracing. The higher the resolution the most accurate the results. Depending on the model used, the various parameters described in previous section can be adapted. Their initialization at the beginning of each command file is easy.

3.5.2 Influence of the resolution

For readability the influence of the resolution is illustrated on the cube. The point cloud processing results does depends slightly on the resolution as can be seen on next figures. For orientation with big visible surfaces both resolutions are fine and give very accurate results. For the case where some of the surfaces are barely visible the processing of the point cloud does not give very good results. That could be improved by having more robust code or by filtering the snapshot used for attitude determination purpose. The higher resolution on the figures is 1001×1001 and the lower one is 501×501.

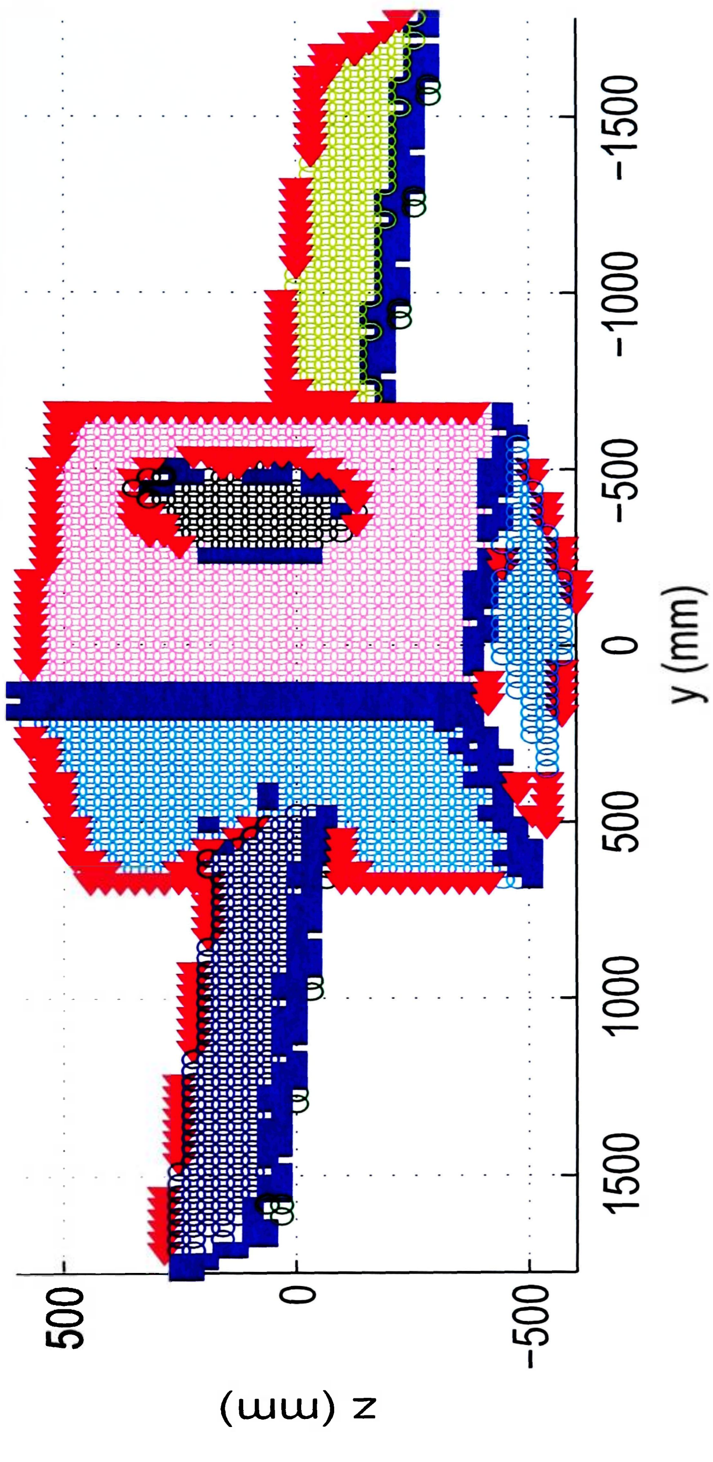
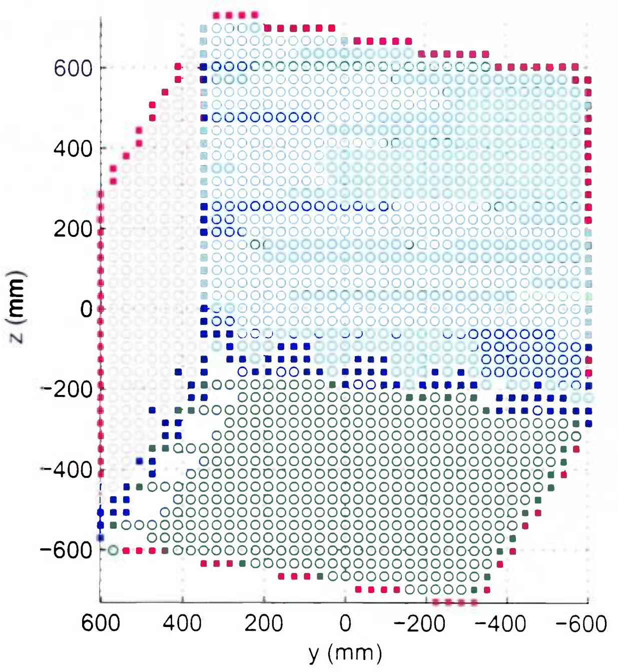


Figure 3.8: Spacecraft point cloud processing

The magenta triangles are the boundary points, the blue squares are the edge points, the empty circles are surface points.



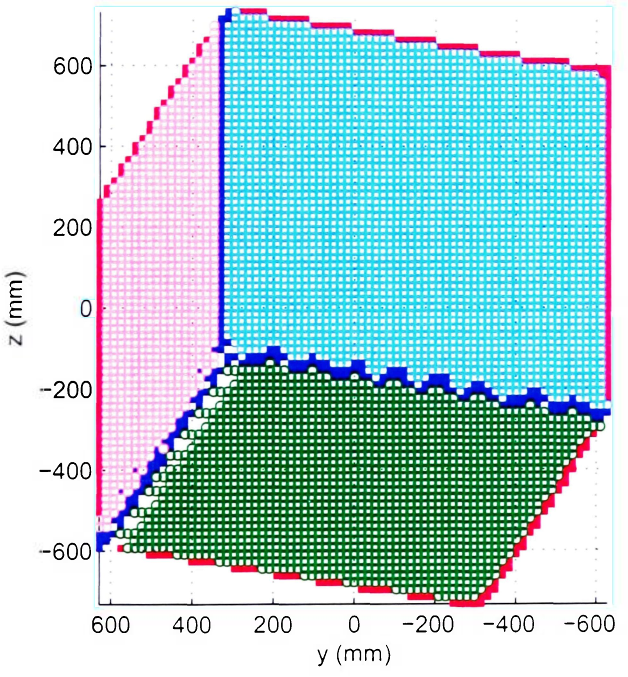
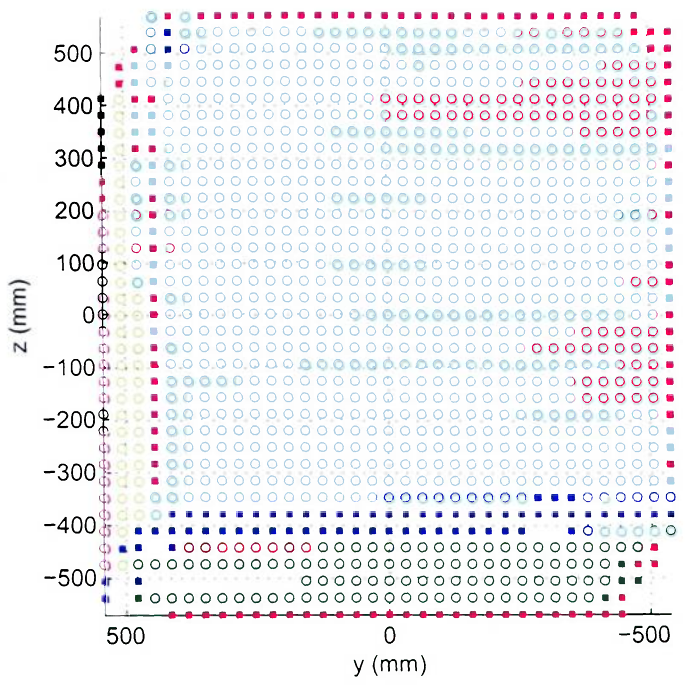


Figure 3.9: Low resolution versus high resolution when all the surfaces are well visible.



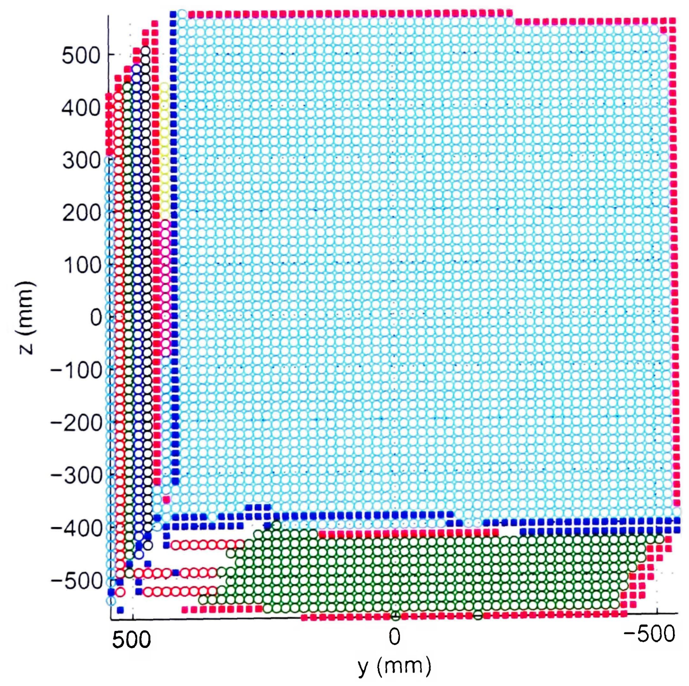


Figure 3.10: Low resolution versus high resolution when two of the surfaces are barely visible.

Chapter 4

Attitude Determination

The previous chapter describes the process of finding edges, boundaries and surfaces from the point cloud. This chapter explains how from these features it is possible to attach a pertinent reference frame (RF) and recover the attitude of the spacecraft. This reference frame must be defined from special features that are common to each snapshot in other words, feature that are fixed on the body.

A first attempt was made by enclosing each detected surface in a minimum volume bounding ellipsoid and define a RF based on the centers of the biggest ellipsoids. The projections of the ellipsoids were also study without convincing results. It was deduced that bounding ellipsoids were not accurate enough and some secondary features had to be retrieved. This process being complex, the study was performed on a cube instead of the whole spacecraft. However, the algorithm is readily applicable to parallelepiped, that are widely used as bounding boxes. A bounding box is a cuboid, or in 2-D, a parallelogram, fully containing an object.

It was finally found that a RF based on corners gives acceptable results. Two types of corners are detected for each snapshot: (1) A 'corner' located at the intersection of three surfaces, (2) Three 'boundary corners' at the extremity of the edges. From those second level features, and using a best fitting sphere, the center of rotation of the object can be found. Based on the center of rotation, the corner and one boundary corner, a RF is defined. Recall that a snapshot is a lidar picture taken at a certain time for a certain orientation. It is assumed that the lidar is fast enough so that the object does not move during the scanning. Comparing the RF of the different snapshots, the attitude is retrieved.

4.1 Second Level Feature Detection

Definitions: Edges are at the intersection of two visible/detected surfaces. Boundaries are the points at the extremity of the scanned object that are the intersection of one visible and one hidden surfaces.

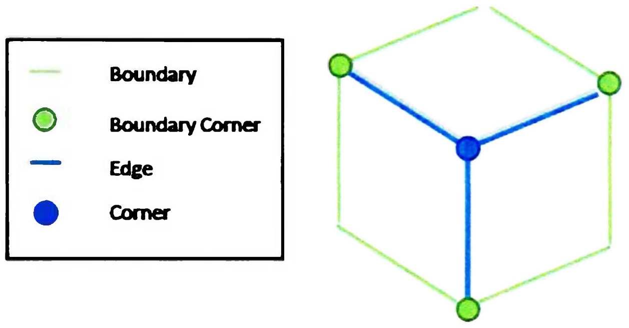


Figure 4.1: Definition corner and boundary corners

Figure 4.1 clarifies the terms *corner* and *boundary corners* that will be used throughout the chapter.

4.1.1 Corner detection

The function *find_corner* finds a corner defined by the intersection of three surfaces. Because the studied object is a cube, the case where the intersection of the three planes is a point is the only possibility. The body of the spacecraft are usually closed box therefore it is always possible to find corners at intersection of planes.

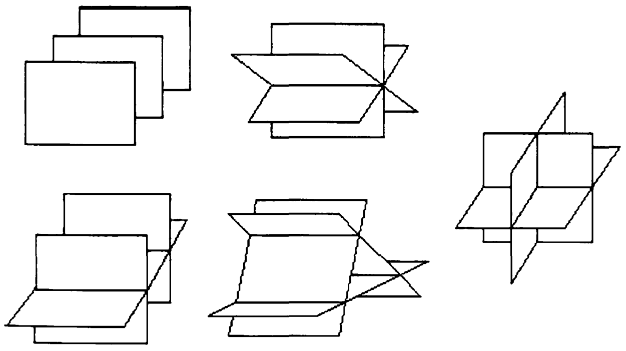


Figure 4.2: Different cases of intersection between three planes.

Principle

Each plane is described by an equation of the following type:

$$\mathbf{n_i} \cdot \mathbf{P_i} = d_i \ , \ i \in \{1, \ 2 \ 3\} \tag{4.1}$$

If $X = (x_1, \ x_2, \ x_3)$ then the intersection point can be obtained by solving the system:

$$\begin{cases} \mathbf{n_1} \cdot \mathbf{X} = d_1 \\ \mathbf{n_2} \cdot \mathbf{X} = d_2 \\ \mathbf{n_3} \cdot \mathbf{X} = d_3 \end{cases} \tag{4.2}$$

Note that $\mathbf{n_1} \cdot (\mathbf{n_2} \times \mathbf{n_3}) = 0$ ensures a unique intersection point.

The system of three equations can be solved by using the Cramer rule, a Gaussian elimination algorithm or as suggested by Goldman [9]:

$$X = \frac{d_1(\mathbf{n_2} \times \mathbf{n_3}) + d_2(\mathbf{n_3} \times \mathbf{n_1}) + d_3(\mathbf{n_1} \times \mathbf{n_2})}{\mathbf{n_1} \cdot (\mathbf{n_2} \times \mathbf{n_3})} \tag{4.3}$$

This last solution is the easiest to implement and works well.

Implementation

The algorithm is implemented in a separate function and starts by counting the number of surfaces detected, then sorts them according to their number of points. The three largest faces are kept. From each of the three remaining surfaces, an arbitrary point is picked, its normal is retrieved in order to define the planes whose intersection is sought. Recall that the normal was computed during the edge search and stored along with the variance in a structure. The d_i parameter is computed according to the equation 4.1 with the selected point and its associated normal. The intersection is calculated using equation 4.3.

Results

The error between the expected value and the computed corner was calculated for different orientations. Without noise, the error is very small, of the order of 10^{-10}. Including noise in the measurements (refer to paragraph on noise 2.2.4) the error is slightly bigger but remains below 10^{-7}. Figure 4.3 shows that the algorithm works well even in the case where two of the surfaces are not well defined. The calculated corner appears in green.

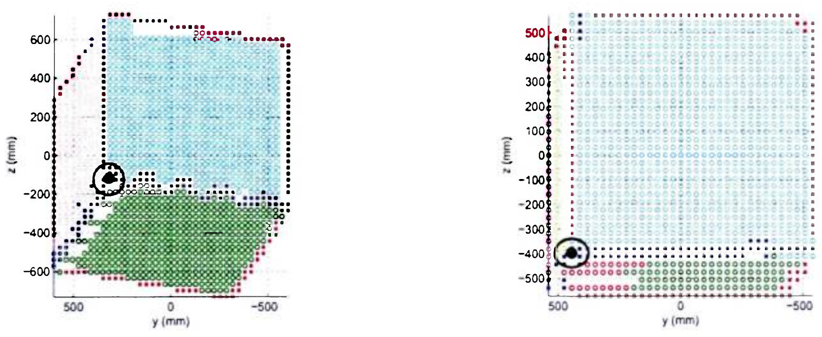


Figure 4.3: Corner detection results.

On the left, the three surfaces are well defined, on the right is shown the case where two of the surfaces are barely visible

4.1.2 Edge labeling

This section describes the function that segments the edges in order to label them. The three surfaces of the object containing the greater number of points are found in the same fashion as in the corner detection function. The function in this case however is extremely sensitive to the selection of the arbitrary point. For some orientations a point would work perfect, for some others, the picked point was not defining the surfaces correctly. As a result, one edge would not be labeled. To remedy to this problem quickly but efficiently, a small test at the end of the function was implemented. It checks how many edges are labeled, if there is only two, the non labeled point are labeled with the missing label. It also counts how many points belong to each labeled edge. If it is below three, it is assumed that some points were missed and they are labeled with the label counting less than three points. The three largest surfaces are arranged in pairs ((1, 2), (2, 3), (3, 1)). The intersection between each set of two surfaces is found. The resulting line is described in a parametric form by a point and a direction:

$$\mathbf{P}(t) = \mathbf{P}_0 + \mathbf{D} \cdot t \tag{4.4}$$

where:

$\mathbf{P}(t)$	=	Any point on the line
$\mathbf{P}_0$	=	Known point belonging to the line
$\mathbf{D}$	=	Direction of the line
t	=	Variable

For each of the line of intersection a loop over all the edge points is performed to determine the distance between a current edge point and the current line.

Assuming that P_p is the base of the perpendicular dropped from P to L, then the vector $\overrightarrow{P_1 P_p}$ is the projection of the vector $\overrightarrow{P_1 P}$ onto $\vec{D}$, as shown in Figure 4.4. The distance $d(P, L)$ from an arbitrary point P to a line L given by a parametric equation is:

$$Distance = | \mathbf{w} - (\mathbf{w} \quad \mathbf{D}) \quad \mathbf{D} | \tag{4.5}$$

where:

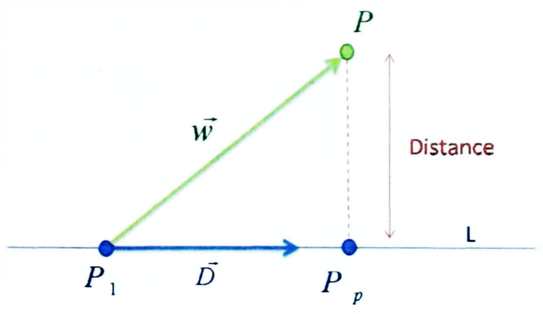


Figure 4.4: Notation for the line-point distance computation.

w	=	Vector $\overrightarrow{P_1 P}$
D	=	Direction of the line
t	=	Variable

For details on the derivation refer to [18]. This calculated distance is compared with a threshold. If the distance is below that threshold then the edge point belong to the studied edge. A threshold too low implies missed edge points and edges becoming incomplete. If it is too high, points from other boundaries are picked up. The pseudo code can be found in appendix D.7. Note that the exact same process was applied to boundary points since they sometimes appear near the edges as can be seen on Figure 4.3.

Results

Results appear on Figure 4.5, the green dot is the corner. Note that the labeling is not done in the same order from orientation to next orientation (the colors of the edges are not consistent from snapshot to snapshot). This makes difficult the tracking of an edge. It is necessary to know which boundary corner is used when fixing a reference frame on the object. This is done by clustering the boundary corners (explained in next section).

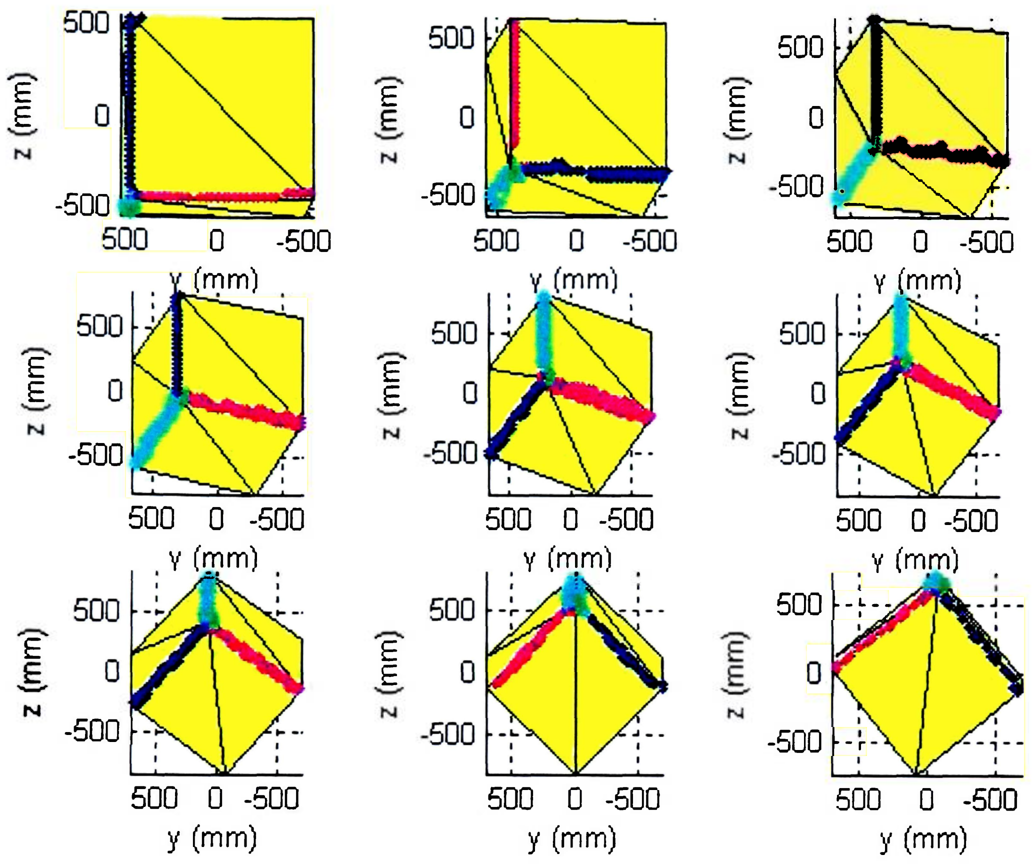


Figure 4.5: Results of the edge labeling.

4.1.3 Boundary corner

For each labeled edge point, the distance between the corner and the current point is computed. The furthest point from the corner is the boundary corner.

4.1.4 Errors in corner and boundary corner detection

Figure 4.6 shows the theoretical position of the corners and boundary corners as well as their measured positions for the range of orientation expressed in Euler form: (3 to 45, 5 to 75, 0) in degrees. The orientation (0, 0 ,0) is not considered since there is only one faces visible (no corner).

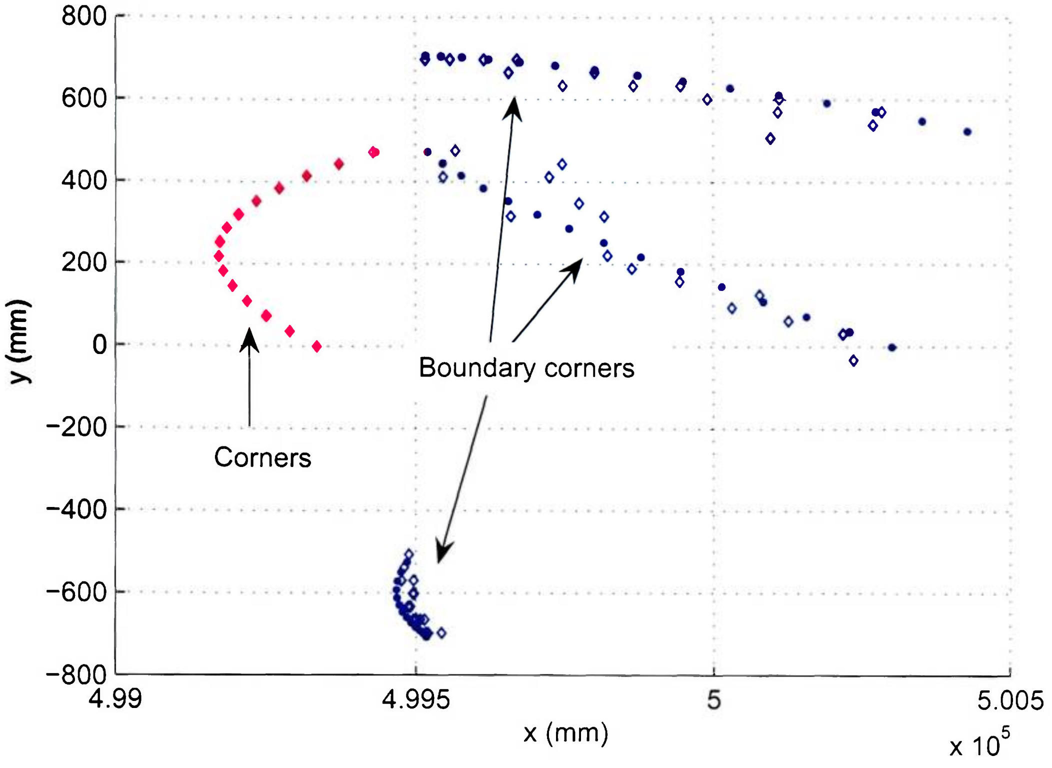


Figure 4.6: Corners: theoretical and measured positions.

The filled circles are the theoretical positions (magenta for corner, blue for boundary corners). The empty diamonds are the measured positions (same color code).

The error computations described in this section were done using the pose (6, 10, 0). It is one of the less accurate since only one face is well visible. It shows that even in the worst case, the error is very tolerable. Table 4.1 expresses the absolute value of error between the measured corner and the theoretical position of that corner in different way. The first column shows the difference in coordinates in cm:

$$Error = \vec{r_{th}} - \vec{r_m} \tag{4.6}$$

where:

$\vec{r_{th}}$ = Theoretical position of corners

$\vec{r_m}$ = Measured position of corners

The second column expresses this error as a percentage error (the computation is performed component by component):

$$Error\ \% = \frac{r^i_{th} - x^i_m}{r^i_{th}} \times 100,\ i \in \{1,2,3\} \tag{4.7}$$

The third column is the distance between the position of the measured corner and the position of the theoretical point. It is expressed in cm.

$$D = \left\| \overrightarrow{r_{th}} - \overrightarrow{r_m} \right\| \tag{4.8}$$

		Difference of coordinates	**% Error**	**Distance**
		(cm)	%	(cm)
Equation to refer to		4.6	4.7	4.8
	x	1.10^{-5}	10^{-8}	
Corner	y	-8.10^{-5}	-2.10^{-4}	8.10^{-6}
	z	-1.10^{-6}	4.10^{-6}	
	x	−0.05	-1.10^{-4}	
Boundary corner 1	y	3.28	7.40	0.37
	z	1.71	2.91	
	x	8.35	0.02	
Boundary corner 2	y	0.98	1.78	1.04
	z	−6.17	10.8	
	x	−0.50	-1.10^{-3}	
Boundary corner 3	y	−1.06	1.93	0.37
	z	−3.47	8.37	

Table 4.1: Error in corner measurement for orientation (6, 10, 0).

t	Orientation		
	Z	Y	X
1	3	5	0
2	6	10	0
3	9	15	0
4	12	20	0
5	15	25	0
6	18	30	0
7	21	35	0
8	24	40	0
9	27	45	0
10	30	50	0
11	33	55	0
12	36	60	0
13	39	65	0
14	42	70	0
15	45	75	0

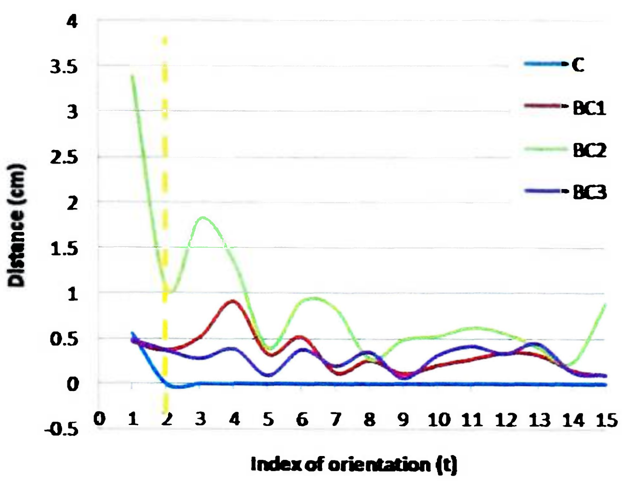


Figure 4.7: Distances between theoretical and measured corners as a function of the orientation

C = Corner, BC = Boundary Corner. Computations were done for 15 different attitudes.

Figure 4.7 shows the evolution of the error as a function of the orientation. The dashed line represents the orientation used to compute the errors described in table 4.1. The corner stabilizes at zero indicating a very accurate restitution of the position. BC1 and BC3 do not suffer from the changes in orientation. It is easily understood while looking at Figure 4.5. BC1 and BC3 are the extremities of the dark blue and pink edges appearing on the three last snapshots (bottom ones). BC2 is the extremity of the light blue edge on the same last three snapshots, as it disappears from the field of view, the error increases. BC2 is the less accurate boundary corner.

4.2 Determination of the Center of Rotation

The center of rotation is determined by fitting a sphere to the corners found for different poses. The center of the sphere is the center of rotation. The study is performed for the orientations showed in Figure 4.7 given in Euler (3,2,1) convention. The corners are used because their measurements are more accurate than boundary corners.

4.2.1 Principle of best fitting sphere

From a set of N points, it desired to find a sphere defined by its radius and center that go through as much point as possible. The equation of a sphere in 3D is given by:

$$r^2 = (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 \tag{4.9}$$

where

x, y, z	=	Coordinates of points on the surface of the sphere
x_c, y_c, z_c	=	Coordinates of the center of the sphere
r	=	Radius of the sphere

The best fitting sphere can be found from a minimum set of four points since there are four unknowns, the three components of the center as well as the radius. The non linear system 4.10 has to be solved:

$$r^2 = (x_i - x_c)^2 + (y_i - y_c)^2 + (z_i - z_c)^2, \; i \in \{1, 2, 3, 4\} \tag{4.10}$$

where $p_i = (x_i,\ y_i\ z_i)$ is the i^{th} point used for the fitting process. The system can be generalized to N points with N equations. It becomes overdetermined and an optimization is required. The Least Squares (LS) method is used. LS is a common method for fitting data. The best fit in the LS sense is that instance of the model for which the sum of squared residuals has its least value, a residual being the difference between an observed value and the value given by the model.

In order to do so, the system is re-written in terms of norm:

$$\begin{aligned} r^2 &= (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 \\ r^2 &= \|P - C\|^2 \end{aligned} \tag{4.11}$$

Recalling the Euclidean definition of the norm:

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} \tag{4.12}$$

We can write:

$$||P\text{-}C||^2 = \sum_{i=1}^{n} |P_i - C|^2 \tag{4.13}$$

Which can be expanded and rearranged:

$$r^2 = \Sigma\, P_i^2 - \Sigma\, 2P_iC_i + \Sigma\, C_i^2 \ ,\ i \in \{1,\ 2\ 3\}$$

$$\underbrace{r^2 - \Sigma\, C_i^2}_{R} + \Sigma\, 2P_i\, C_i = \Sigma\, P_i^2 = ||P_i^2|| \tag{4.14}$$

For N points, the last equation can be expressed as a matrix equality:

$$\underbrace{\begin{bmatrix} 2x_1 & 2y_1 & 2z_1 & 1 \\ 2x_2 & 2y_2 & 2z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 2x_N & 2y_N & 2z_N & 1 \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} x_c \\ y_c \\ z_c \\ R \end{bmatrix}}_{\text{X}} = \begin{bmatrix} x_1^2 + y_1^2 + z_1^2 \\ x_2^2 + y_2^2 + z_2^2 \\ \vdots \\ x_N^2 + y_N^2 + z_N^2 \end{bmatrix} \tag{4.15}$$

Since the goal is to solve for X, matrix A must be full rank to ensure the existence of the inverse.

4.2.2 Results of best fitting sphere

Table 4.2 shows the coordinates of the center of the fitted sphere for 4, 8 and 12 corner points (corresponding to different orientations). The theoretical center is at [500000 0 0] in mm. The radius is the half diagonal of the cube, corresponding to $\sqrt{3} \times \frac{Side}{2} = \sqrt{3} \times 500 = 866.025\ mm$.

A comparison between a sphere fitting only corners and a sphere fitting the corners plus the boundary corners is presented in table 4.3.

The more points, the more accurate the center of rotation, and radius. The radius is given for information purpose because it is never used in the following development.

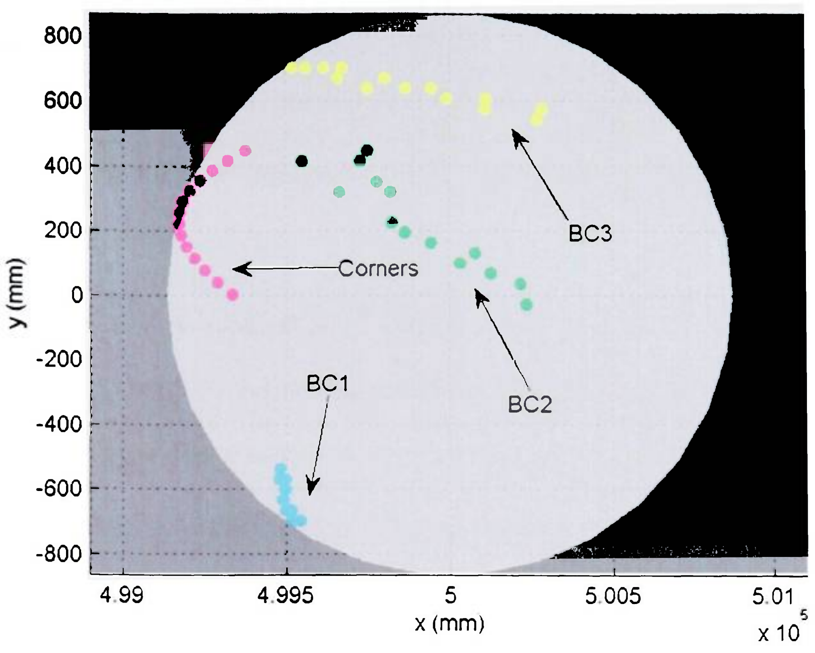


Figure 4.8: Sphere fitting the corners.

The magenta dots (corners) are the fitted corners, the other dots are the different boundary corners.

Number of points considered	N	4	8	12
Center	x	499999.898	499999.999	500000.002
	y	1.08	0.11	0.03
	z	0.41	0.05	0.01
Radius	r	865.59	865.00	866.02

Table 4.2: Center of rotation coordinates (mm). Sphere fitting corners ONLY

The important data is the center of rotation. The figures are given in mm. The biggest error is on y and has a magnitude of 1 mm in the 'only corners' case and 34 mm on z in the 'corners plus boundary corners' case.

This method is therefore very accurate as long as no 'bad corner' is taken into account. 'Bad corners' are corners computed for orientation of the spacecraft where the visibility of certain faces is limited. In this case the position of the corner is less accurate leading to bad fitting. For instance, the orientation (3, 5, 0) was not taken into account in the simulation because the error on the corner location was important. If this orientation

Number of point considered	N	**4**	**8**	**12**
Center	x	499944.746	499954.445	499964,481
	y	12.48	7.81	1.92
	z	-34.32	-20.71	-8.09
Radius	r	806.16	818.81	827.27

Table 4.3: Center of rotation coordinates (mm). Sphere fitting corners AND boundary corners.

was taken into account in addition to 12 other corners, the error is of the order of the cube size itself! If boundary corners are added to this set of points then the result becomes acceptable since the error is 46 mm on x, 3 mm on y, and 52 mm on z. In practise a filter can be designed to remove the views of the spacecraft that do not show enough of certain surfaces.

In conclusion, the center of rotation is determined by fitting a sphere to the corners only since it gives better results. This center is used to define an object attached reference frame as described in the next section.

4.3 Choice of a Reference Frame

The objective of this section is to describe a way to retrieve the orientation of the scanned object without any assumption on the shape except for the features that have been detected. The steps taken so far are:

1. Built a model
2. Given a (3, 2, 1) Euler angle sequence converted to a rotation matrix, rotate the model.

$\left.\right\}$ Modeling part

3. Scan the model with ray tracer
4. Process the point cloud obtained from ray tracing

From the information gathered through this process, it is desired to retrieve the orientation. It was hoped to find the Euler angle so error was readily available and understandable. However since the transformation from rotation matrix to Euler angle is not bijective, it was not possible. The angular displacement between two consecutive orientations (described by rotation matrices) are calculated and compared with the rotation used in modeling part to rotate the model.

4.3.1 Advantages and drawbacks of rotation matrices

Advantages

Matrix form is a very explicit form of representing orientation. This explicit nature provides some benefits.

- Rotation of vectors is immediately available
- Standard format used by graphics APIs
- Concatenation of multiple angular displacements
- Matrix inversion. When an angular displacement is represented in matrix form, it is possible to compute the opposite angular displacement using matrix inversion. Note that since rotation matrices are orthogonal, this computation is a trivial matter of transposing the matrix.

Drawbacks

- Matrices take more memory than other techniques because nine components have to be stored instead of three for Euler Angle and four for quaternions
- Difficult for humans to use, they are not intuitive

As a reminder, Figure 4.9 shows the direction cosines of the x_o axis.

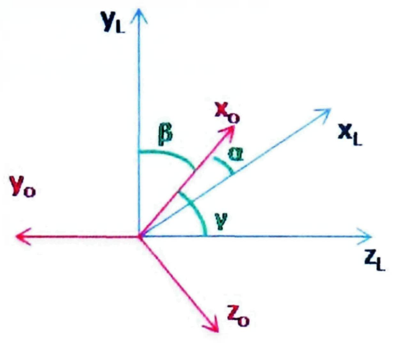


Figure 4.9: Recall on direction cosine matrix

4.3.2 Definition of the reference frame (RF)

The corners and the center of rotation are determined very accurately, therefore those points are the first candidates for defining a RF. A third point is needed. It has to be a boundary corner. The definition of the RF attached to the object is thus defined with the center of rotation (A), the corner (B) and one boundary corner (C). The origin of the frame is the center of rotation. The x axis is the unit vector of $\overrightarrow{AB}$. The z axis is perpendicular to the (ABC) plane. Y axis simply complete the right handed frame. Refer to Figure 4.10. Note that since the studied object is a cube, the boundary corner falls on the y axis but it is not necessarily the case. The third point, here the boundary corner is used to define a plane.

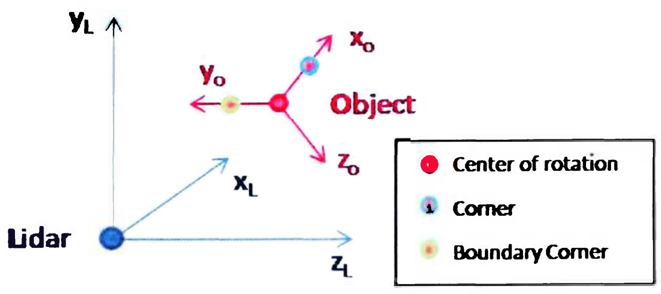


Figure 4.10: Reference frames of the lidar and the object

The computation of the axis of the RF is done in terms of component of unit vectors which actually gives the direction cosines.

4.4 Results

For each snapshot, a RF is defined and stored as a direction cosine matrix. Since the rotation matrix used to rotate the initial model and the direction cosine matrix (DCM) defined from corners are not built the same way, it is not possible to compare them directly. The comparison is done on the change in orientation. For each two consecutive snapshots a transition matrix M_T is calculated:

$$M_T = DCM_{t+\Delta t} \times DCM_t^T$$

where DCM_t is the DCM at orientation t and $DCM_{t+\Delta t}$ is the new DCM for the orientation in the sequence. The transition matrices are calculated for the theoretical case and for the measured case and compared. Note that the computed transition matrices are also normalized DCMs. DCMs can be represented as rotations of RFs as shown on Figure 4.11.

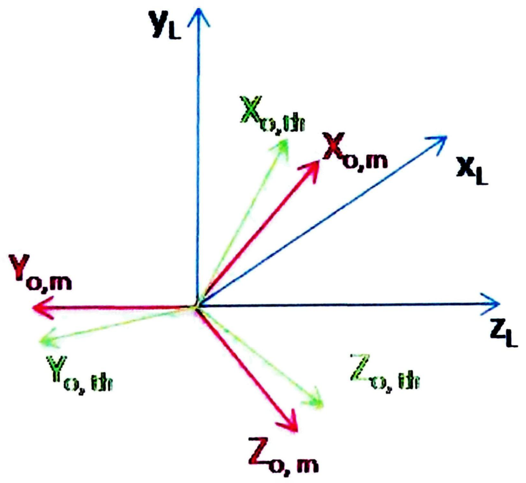


Figure 4.11: Theoretical and experimental reference frames to be compared

Figure 4.11 shows the expected transition matrix (O, $X_{o,th}$, $Y_{o,th}$, $Z_{o,th}$) and the experimental one (O, $X_{o,m}$, $Y_{o,m}$, $Z_{o,m}$). Since the transition matrices are normalized, the projection of one axis onto another gives the angle between them. This is how the error of the whole process is expressed.

In the following sections, two different sequences of orientations are studied. For each case, a figure shows the corners and the fitting sphere as well as the different orientations of the sequence. Then a table summarizes the angle of error between the theoretical and measured RF shown on Figure 4.11. Since the object reference frame is defined using only one boundary corner, results are presented for each of the three boundary corners.

First sequence of snapshots

t	Orientation		
	Z	Y	X
1	6	10	0
2	9	15	0
3	12	20	0
4	15	25	0
5	18	30	0
6	21	35	0
7	24	40	0
8	27	45	0
9	30	50	0
10	33	55	0
11	36	60	0
12	39	65	0
13	42	70	0
14	45	75	0

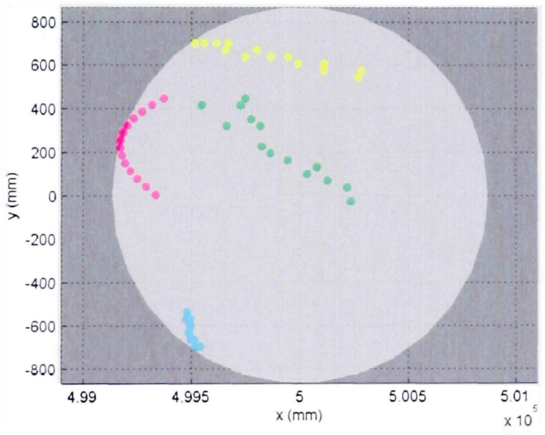
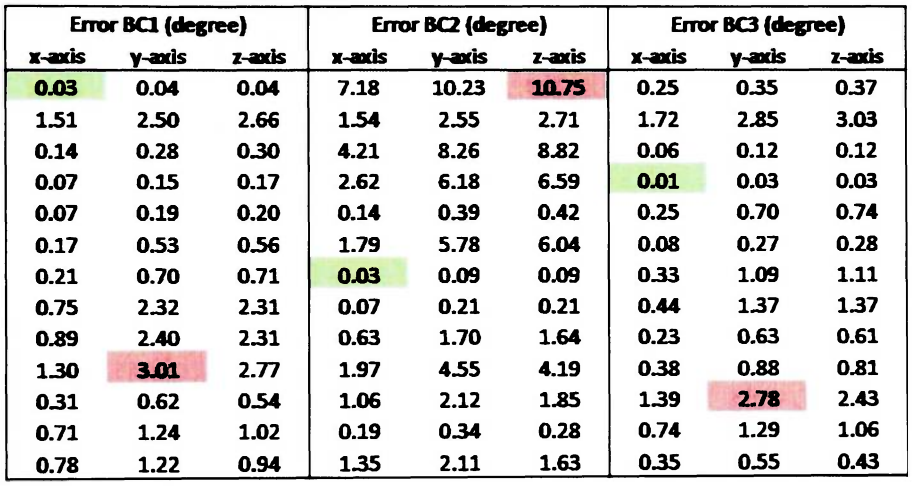


Figure 4.12: Orientation used for simulation (left) and plot of the corners (right)

Figure 4.12 shows the correspondence between the shot number parameter (t) and the orientation of the object as well as a plot of the boundary corners BC1 is in yellow, BC2 in green and BC3 in cyan. Figure 4.14 shows the error angle between the expected direction cosine matrix and the measured one. The values are summarized in Figure

4.13. The biggest errors are encountered when the object RF is defined based on BC2. In this case the results are off by 10.75 °. When the RF is based on BC1 or BC3 the error decreases to 3 °. An error of 3 ° is significant since the rotation of the model around z axis is done with a step of 3. However, if this step is doubled (6 °) the error does not increase but remains with a maximum of 3 °. These results indicate that a filtering of the rotation matrices based on a simple 'goodness' of orientation can be performed. The filtering criterion can be a threshold on the variation of the rotation angle between two consecutive shots. Another filtering method can consist in averaging the angles between all the shots and rejecting the values larger than the average.



Error BC1 (degree)			Error BC2 (degree)			Error BC3 (degree)		
x-axis	y-axis	z-axis	x-axis	y-axis	z-axis	x-axis	y-axis	z-axis
0.03	0.04	0.04	7.18	10.23	10.75	0.25	0.35	0.37
1.51	2.50	2.66	1.54	2.55	2.71	1.72	2.85	3.03
0.14	0.28	0.30	4.21	8.26	8.82	0.06	0.12	0.12
0.07	0.15	0.17	2.62	6.18	6.59	0.01	0.03	0.03
0.07	0.19	0.20	0.14	0.39	0.42	0.25	0.70	0.74
0.17	0.53	0.56	1.79	5.78	6.04	0.08	0.27	0.28
0.21	0.70	0.71	0.03	0.09	0.09	0.33	1.09	1.11
0.75	2.32	2.31	0.07	0.21	0.21	0.44	1.37	1.37
0.89	2.40	2.31	0.63	1.70	1.64	0.23	0.63	0.61
1.30	3.01	2.77	1.97	4.55	4.19	0.38	0.88	0.81
0.31	0.62	0.54	1.06	2.12	1.85	1.39	2.78	2.43
0.71	1.24	1.02	0.19	0.34	0.28	0.74	1.29	1.06
0.78	1.22	0.94	1.35	2.11	1.63	0.35	0.55	0.43

Figure 4.13: Angle error for BC1, BC2 and BC3 for each orientation

Another way to visualize results is done in Figure 4.14. They show that the error on the x axis is smaller than for on the two other axis.

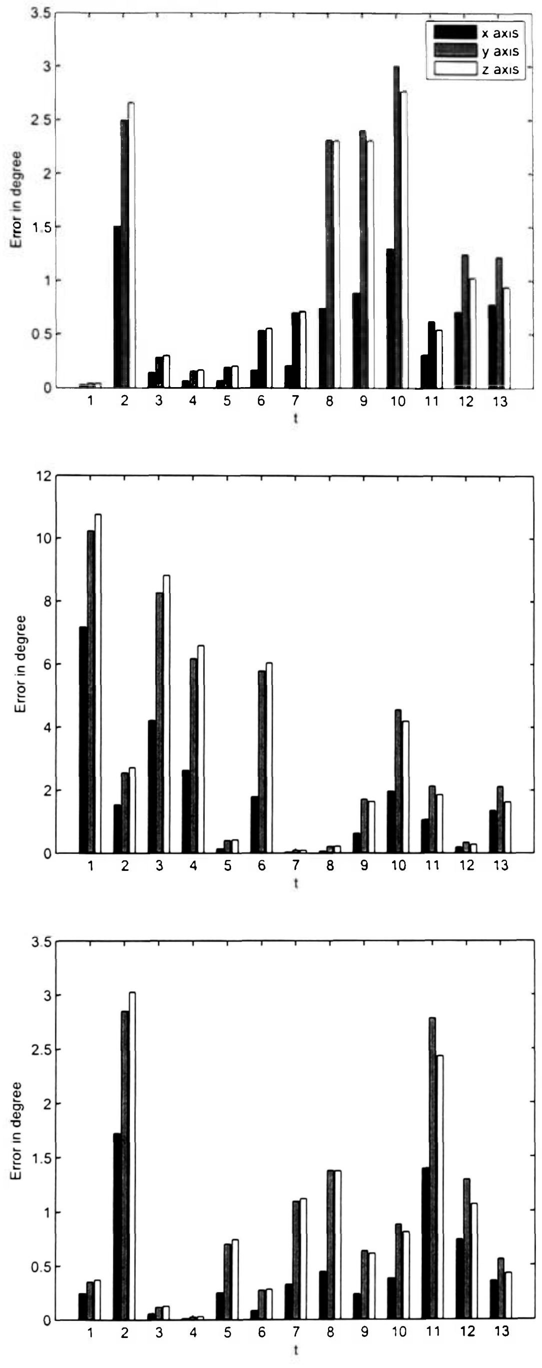
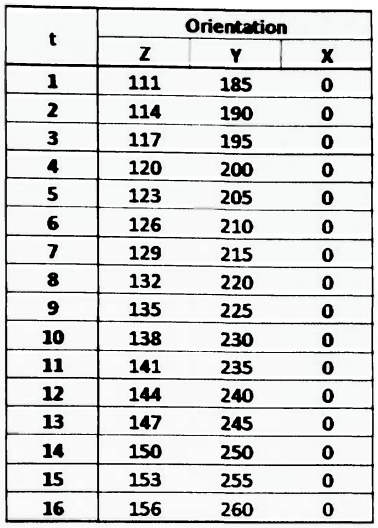


Figure 4.14: Error in angle between axis of RF_{th} and RF_m
Using BC1 (top), BC2 (middle) and BC3(bottom)

Second sequence of snapshots



t	Orientation		
	Z	Y	X
1	111	185	0
2	114	190	0
3	117	195	0
4	120	200	0
5	123	205	0
6	126	210	0
7	129	215	0
8	132	220	0
9	135	225	0
10	138	230	0
11	141	235	0
12	144	240	0
13	147	245	0
14	150	250	0
15	153	255	0
16	156	260	0

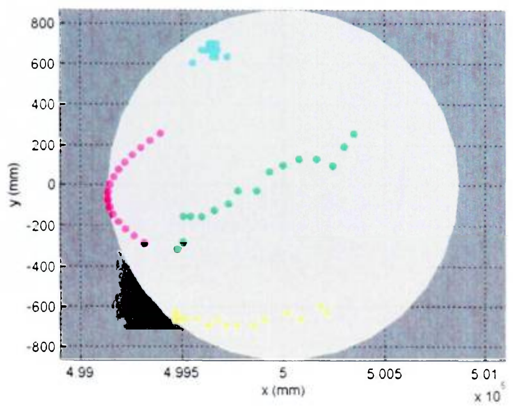
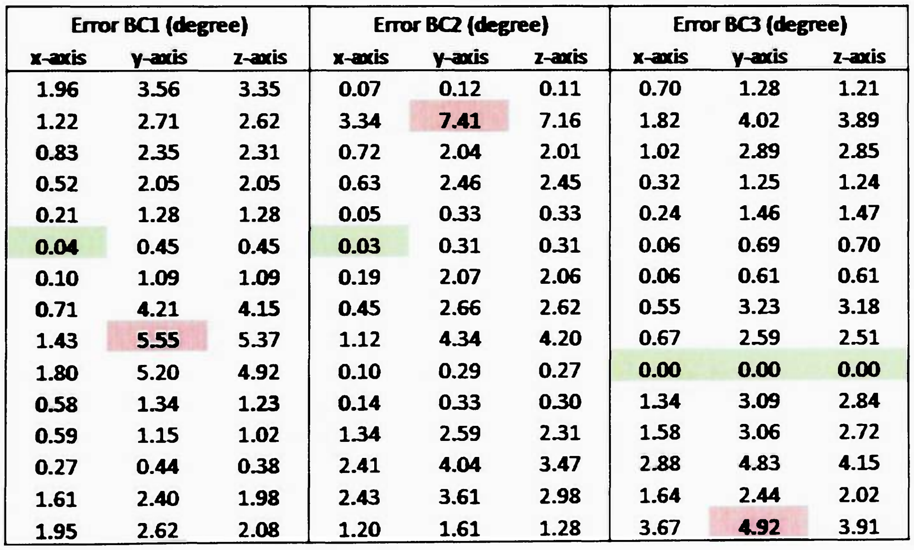


Figure 4.15: Orientation used for simulation and plot of the Corners



Error BC1 (degree)			Error BC2 (degree)			Error BC3 (degree)		
x-axis	y-axis	z-axis	x-axis	y-axis	z-axis	x-axis	y-axis	z-axis
1.96	3.56	3.35	0.07	0.12	0.11	0.70	1.28	1.21
1.22	2.71	2.62	3.34	7.41	7.16	1.82	4.02	3.89
0.83	2.35	2.31	0.72	2.04	2.01	1.02	2.89	2.85
0.52	2.05	2.05	0.63	2.46	2.45	0.32	1.25	1.24
0.21	1.28	1.28	0.05	0.33	0.33	0.24	1.46	1.47
0.04	0.45	0.45	0.03	0.31	0.31	0.06	0.69	0.70
0.10	1.09	1.09	0.19	2.07	2.06	0.06	0.61	0.61
0.71	4.21	4.15	0.45	2.66	2.62	0.55	3.23	3.18
1.43	5.55	5.37	1.12	4.34	4.20	0.67	2.59	2.51
1.80	5.20	4.92	0.10	0.29	0.27	0.00	0.00	0.00
0.58	1.34	1.23	0.14	0.33	0.30	1.34	3.09	2.84
0.59	1.15	1.02	1.34	2.59	2.31	1.58	3.06	2.72
0.27	0.44	0.38	2.41	4.04	3.47	2.88	4.83	4.15
1.61	2.40	1.98	2.43	3.61	2.98	1.64	2.44	2.02
1.95	2.62	2.08	1.20	1.61	1.28	3.67	4.92	3.91

Figure 4.16: Angle error for BC1, BC2 and BC3

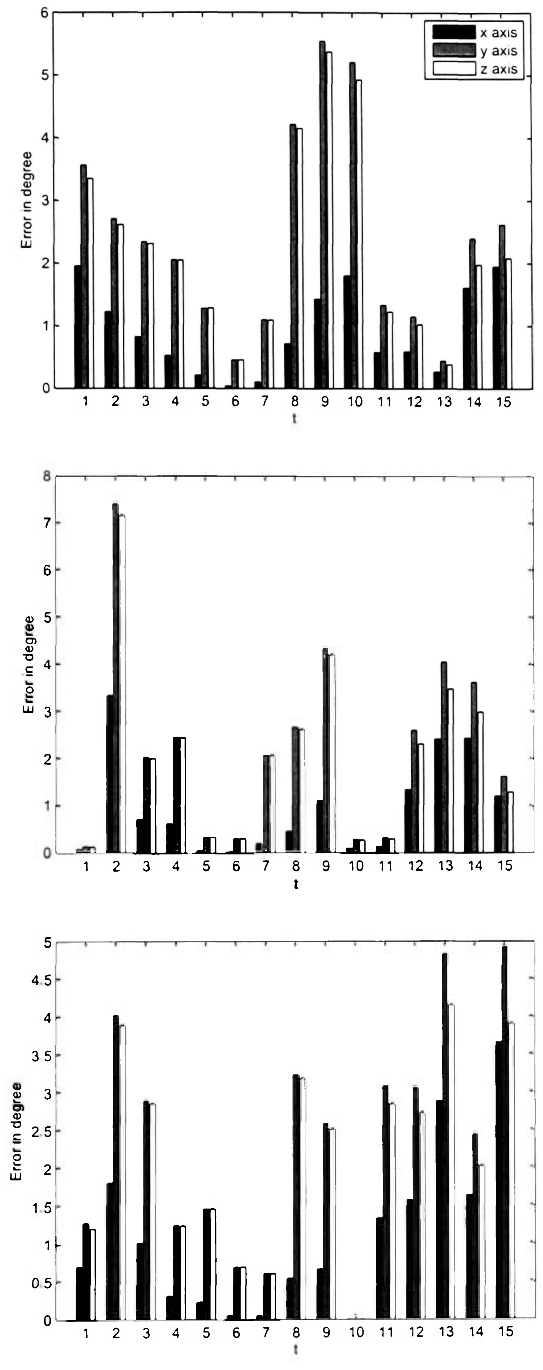


Figure 4.17: Error in angle between axis of RF_{th} and RF_m

Using BC1 (top), BC2 (middle) and BC3(bottom)

Conclusion

In quest of using lidar for navigation, this thesis has showed a procedure to retrieve elements of the attitude of a tumbling, out of control object from a lidar point cloud. The high cost of lidar devices made impossible the acquisition of one. To overcome this difficulty, a lidar simulator was written. Its main components are (1) a ray direction generator and (2) an algorithm that computes range data based on ray tracing techniques. The lidar was implemented to be able to reproduce errors in measurements however the results presented in the present report were obtained without those errors. The simulator outputs a complex point cloud that needs to be simplified. The processing of this set of points mainly consists in extracting features. Covariance matrix and eigenvalue analysis is performed to retrieve edges. A Chi-squared test in the sense of test of goodness of fit allows to detect boundary points (points that do not have neighbors on one side, they constitute the limit of what the lidar can see). Finally the surfaces are created using a region growing method associated to clustering. Simple geometry was used to find corners (at the intersection of three surfaces) and boundary corners (at the extremity of edges, one per snapshot). These features, especially the corners (at the intersection of three surfaces, three per snapshot) were found accurately, less than 0.0001% error. The scenario studied to perform the attitude analysis consists in a chaser equipped with a lidar following an out of control target while taking pictures (or snapshots) at regular time intervals. A sequence of 15 snapshots was considered in the study. Further work would include determining the optimal number of snapshots. The first step in attitude retrieval is finding the center of rotation. It is performed fitting a sphere to the 15 corners detected for the 15 different orientations. This center of rotation is determined very accurately, less than 1% error. Based on the corners, one boundary corner and the center of rotation, a reference frame is defined for each orientation of the sequence and expressed as a direction cosine matrix. The changes in orientation in the theoretical case (rotation of the model) and experimental (body fixed frame based on the features) case are compared. A direct comparison is not possible because the rotation matrix used to rotate the model and the direction cosine matrix representing the orientation of the object are not built the same way. Finding a more direct way to express the result and quantify the error is desirable and should be studied in future research on the topic. The accuracy of the results depends on the accuracy of the boundary corner position used to build the reference frame. It was noticed that the three boundary corners are alternatively accurately determined. The algorithm could be improved by implementing a filter that selects the most accurate boundary corner and define the reference frame from it.

Recommendations This work constitutes a starting point for a robust navigation algorithm based on imaging lidar. A few recommendations are now proposed. The errors found in the change of orientation are due to the inaccurate determination of the boundary corners. The use of high resolution windows would improve edge detection resulting in better results for those corners. Also the whole feature detection part could be facilitated by the use of the intensity parameter implemented in the lidar simulator but not used in the present study. The expression of the results needs to be more intuitive, for instance it would be desirable to obtain the angular rates in terms of Euler angles.The algorithm shall also be tested with various object shapes.

Bibliography

[1] F. Einaudi G. K. Schwemmer B.M. Gentry J.B. Abshire. Lidar past, present, and future in nasa's earth and space science programs. 2004. [cited at p 5]

[2] A. Allen. Lidar based gnc for automatic rendezvous and safe landing. Technical report, MDA Space Missions for ESA, Juillet 2005. [cited at p 11]

[3] U. Soppa B. Moebius, M. Pfennigbauer. Miniaturized 3d lidar for lunar landing. June 2008. [cited at p 11]

[4] D. Belton and D. D. Litchi. Classification and segmentation of terrestrial laser scanner point clouds using local variance information. *International Archives of Photogrammetry*, 36:44 to 49, 2006. [cited at p 42, 47]

[5] S. Ruel C. English S. Zhu C. Smith and I. Christie. Tridar: A hybrid sensor for exploiting the complementary nature of triangulation and lidar technologies. In *ISAIRAS*, September 2005. [cited at p 10]

[6] F. Dunn and I. Parberry. *3D Math Primer for Graphics and Game Development*. Wordware, 2002. [cited at p viii, 28]

[7] ESA. Aurora exploration programme. Website, 2006. http://www.esa.int/esaMI/Aurora/SEM1U7A5QCE_0.html. [cited at p 11]

[8] A. S. Glassner. *An Introduction to Ray Tracing*. Morgan Kaufmann, 1989. [cited at p viii, 28, 30]

[9] R. Goldman. *Intersection of three planes*. Graphics gems. Academic Press Professional, Inc., San Diego, CA, USA, 1990. [cited at p 61]

[10] D. Belton K. Bae and D. D. Lichti. A framework for position uncertainty of unorganized three dimensional point clouds from near mono static laser scanners using covariance analysis. 2005. [cited at p 14, 46]

[11] Neptec Design Group Ltd. Neptec space systems. Neptec Website, 2007. http://www.neptec.com/Neptec_LCS.html. [cited at p 10]

[12] L. P. Kobbelt M. Pauly, M. Gross. Efficient simplification of point-sampled surfaces. IEEE Computer Society, 2002. [cited at p ix, 42, 44]

[13] M. P. McCormick. A review of spaceborne lidar and a look to the future. San Antonio, Texas, January 2007. AMS 3rd Symposium on Lidar Atmospheric Applications. [cited at p 6]

[14] U. Meck. Jena-optronik gnc sensor. http://www.jenaoptronik.com. [cited at p viii, 32]

[15] M. Umasuthan P. Jasiobedzki, S. Se; T. Pan and M. Greenspan. Autonomous satellite rendezvous and docking using lidar and model based vision. In SPIE, editor, *Spaceborne Sensors II*, pages 54–65, 2005. [cited at p 11]

[16] K. A. Powell. Lite. NASA Web site, April 1998. http://www-lite.larc.nasa.gov/. [cited at p 6]

[17] M. Anctil S. Ruel, C. English and P. Church. Lasso: Real time pose estimation from 3d data for autonomous satellite servicing. *ESA*, August 2005. [cited at p 10]

[18] D. Sunday. About lines and distance of a point to a line. Web site, 2006. http://softsurfer.com/Archive/algorithm_0102/algorithm_0102.htm. [cited at p 64]

[19] Wikipedia. Data clustering. Web site, November 2008. http://en.wikipedia.org/wiki/Data_clustering. [cited at p 51]

[20] Dr. D. R. Williams. Mars global surveyor. NASA Web site, December 2004. http://nssdc.gsfc.nasa.gov/planetary/marsurv.html. [cited at p 6]

[21] H. J. Zwally. Glas. NASA Web site. http://icesat.gsfc.nasa.gov/glasinstrument.php. [cited at p. 7]

Appendices

Appendix A

Dimension of the CATIA model of the spacecraft

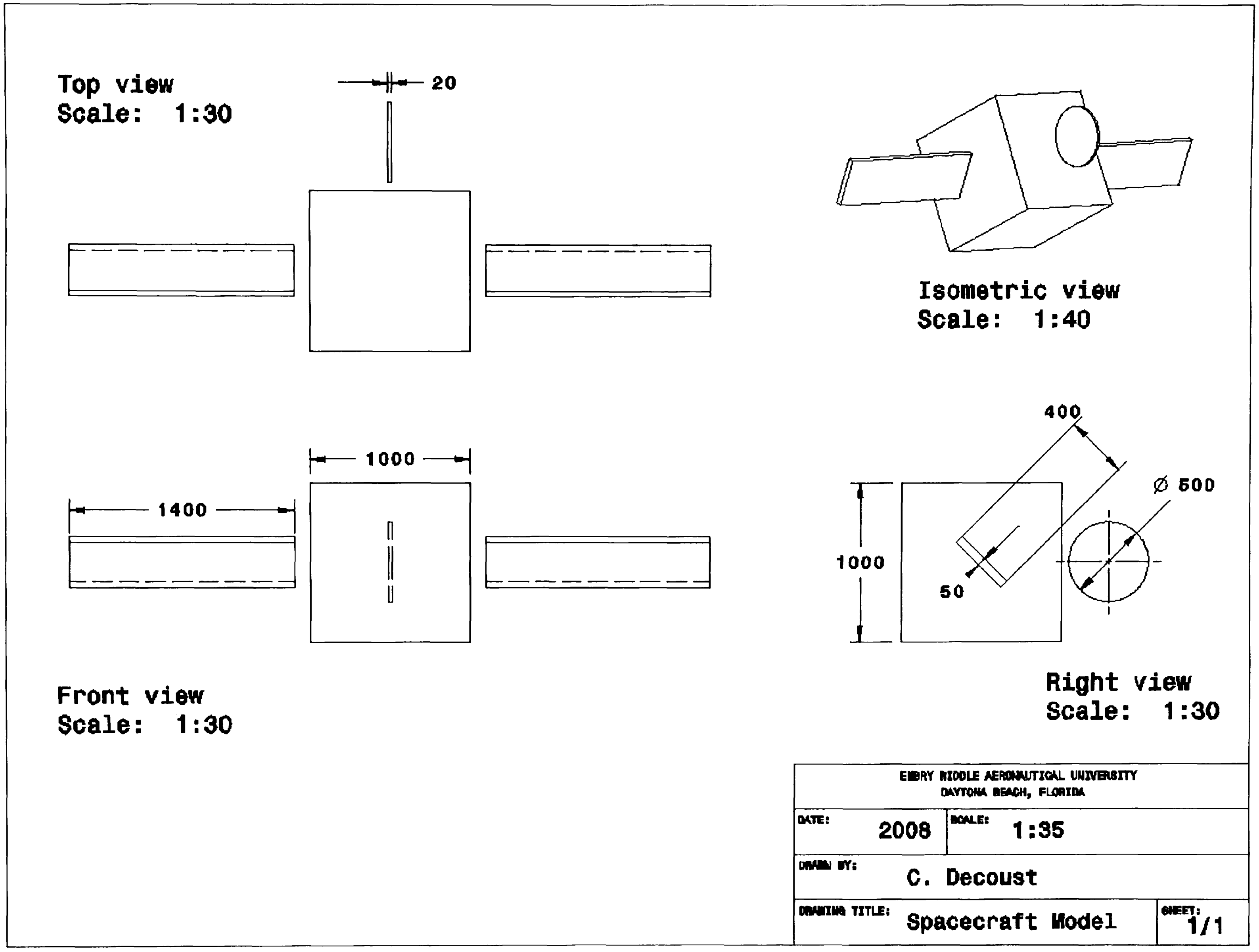


Figure A.1: Catia Model Drawing

Appendix B

STL Reader

B.1 STL Format

STL files describe a facet as follow:

```
facet normal  0   0   1
outer loop
      vertex   5   5   0
      vertex  -5   5   0
      vertex   0  -5   0
endloop
endfacet
```

B.2 Test of the STL Reader

The STL reader has been stress tested with a complex shape as presented on the figure:

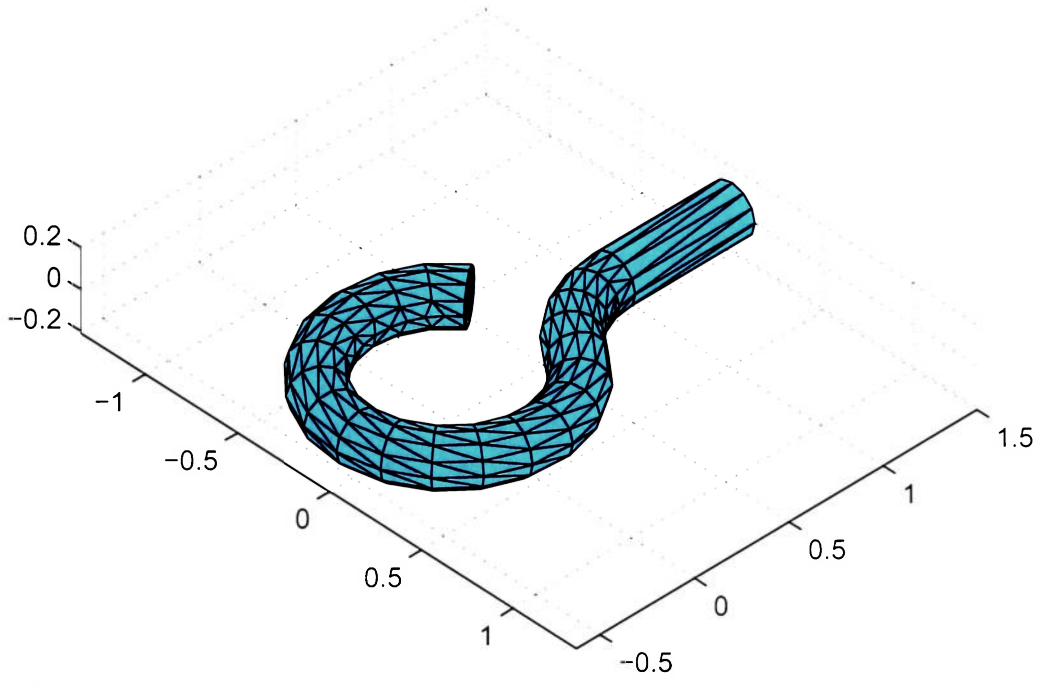


Figure B.1: STL reader tested on a complex shape.

Appendix C

Ray Tracer Pseudo code

C.1 STL_reader

```
Open Stl file

Count the number of lines to deduce the number of facet

Read the data form files

    Look for 'facet\_normal'

    Store the coordinates of the facet normal

    Extract the vertex position from strings

Close file
```

C.2 Ray_gen

```
For regular rays

   From the input define a step size for the directions of the rays

   Generate ray direction and store them

For HR windows

   Find indexes (El. and Az.) of the hires window start at the
```

```
     closest existing ray
Find indexes (El. and Az.) of the hires window end at the
     closest existing ray
If the required window corner lies out of the lidar FoV then
     make the hires window corner the lidar FoV corner
Generate the ray directions
```

Appendix D

Point Cloud Processing Pseudo Code

D.1 Nearestneighbour function Description

```
function [idx, tri] = nearestneighbour(varargin)


NEARESTNEIGHBOUR    find nearest neighbors
   IDX = NEARESTNEIGHBOUR(X) finds the nearest neighbor by Euclidean
   distance to each point (column) in X from X. X is a matrix with points
   as columns. IDX is a vector of indices into X, such that X(:, IDX) are
   the nearest neighbors to X. e.g. the nearest neighbor to X(:, 2) is
   X(:, IDX(2))

   IDX = NEARESTNEIGHBOUR(P, X) finds the nearest neighbor by Euclidean
   distance to each point in P from X. P and X are both matrices with the
   same number of rows, and points are the columns of the matrices. Output
```

```
is a vector of indices into X such that X(:, IDX) are the nearest
neighbors to P


IDX = NEARESTNEIGHBOUR(I, X) where I is a logical vector or vector of
indices, and X has at least two rows, finds the nearest neighbor in X
to each of the points X(:, I).
I must be a row vector to distinguish it from a single point.
If X has only one row, the first input is treated as a set of 1D points
rather than a vector of indices


IDX = NEARESTNEIGHBOUR(..., Property, Value)
Calls NEARESTNEIGHBOUR with the indicated parameters set. Property
names can be supplied as just the first letters of the property name if
this is unambiguous, e.g. NEARESTNEIGHBOUR(..., 'num', 5) is equivalent
to NEARESTNEIGHBOUR(..., 'NumberOfNeighbours', 5). Properties are case
insensitive, and are as follows:
  Property:                      Value:
  ---------                      ------
    NumberOfNeighbours           natural number, default 1
      NEARESTNEIGHBOUR(..., 'NumberOfNeighbours', K) finds the closest
      K points in ascending order to each point, rather than the
      closest point. If Radius is specified and there are not
      sufficient numbers, fewer than K neighbors may be returned

    Radius                       positive, default +inf
```

```
        NEARESTNEIGHBOUR(..., 'Radius', R) finds neighbors within
        radius R. If NumberOfNeighbours is not set, it will find all
        neighbors within R, otherwise it will find at most
        NumberOfNeighbours. The IDX matrix is padded with zeros if not
        all points have the same number of neighbors returned. Note
        that specifying a radius means that the Delaunay method will
        not be used.

    DelaunayMode                {'on', 'off', |'auto'|}
        DelaunayMode being set to 'on' means NEARESTNEIGHBOUR uses the
        a Delaunay triangulation with dsearchn to find the points, if
        possible. Setting it to 'auto' means NEARESTNEIGHBOUR decides
        whether to use the triangulation, based on efficiency. Note
        that the Delaunay triangulation will not be used if a radius
        is specified.

    Triangulation               Valid triangulation produced by
                                delaunay or delaunayn
        If a triangulation is supplied, NEARESTNEIGHBOUR will attempt
        to use it (in conjunction with dsearchn) to find the
        neighbors.

[IDX, TRI] = NEARESTNEIGHBOUR( ... )
If the Delaunay Triangulation is used, TRI is the triangulation of X'.
Otherwise, TRI is an empty matrix
```

```
Copyright 2006 Richard Brown. This code may be freely used and
distributed, so long as it maintains this copyright line
```

D.2 Find_edge_points

```
For each point of the point cloud
   Find nearest neighbors of current point
   Compute the surface variance
   If the surface variance is > threshold
      Decrease neighborhood size (16 to 2)
      Decrease increment to 1
   Else
      Set increment to coarse (16)
   While number of neighbors < number of neighbor max
      Find nearest neighbors
      Compute the surface variance
      If surface variance > threshold
         If increment is already fine
            Label the furthest point with 1 as it is an edge point
            Label all the other points with 0 as it is a surface
         Else
            Reset the label of the last increase
   Take one step back (number of neighbor - increment of neighbor number)
   Set increment of neighbor number to 1
   Increment number of neighbor
   If number of neighbors > number of neighbor max
      Labels all the points as surfaces (0)
```

D.3 Find_bndry_points

```
For each point of the point cloud
   If the current point in not an edge point
      Find nearest neighbor of current point
      Compute centroid of the neighborhood
      Compute covariance matrix of the neighborhood
      Compute Eigenvectors and values of the covariance matrix
      Perform Chi squared test
      If the results of the test > threshold
         Label the point as a boundary
% Clean up the boundary points
Get the boundary points
For each boundary point
   Find nearest neighbor of current point
   If nearest neighbor is an edge point
      Set the boundary point as an edge point
```

D.4 Surf_segm

```
For each point of the cloud
   If current point is on a smooth surface (label = 0)
      Compute absolute value of the components of the normal
      Find and count the points with similar normal direction
```

```
         (taking into account the tolerance)
      If the number of point found is lower than the minimum allowed
         Give a special label to those points
      Else
          Label point of the surface with a number (surface count)
          Increment surface count
%Further segment the surface using Matlab Cluster Data
For each detected/labeled surface
   Find all the points belonging to the surface
   Group the points in x clusters
   Find the smallest cluster
   Discard the cluster whose size is below a defined threshold
   If more than one cluster has been found for one surface
     (as defined in first part of code) then change the label of
the points
```

D.5 Clusterdata function

```
T = clusterdata(X, cutoff) uses the pdist, linkage, and cluster  functions
to construct clusters from data X. X is an m-by-n matrix,treated as m
 observations of n variables.
Cutoff is a threshold for cutting the hierarchical tree generated by linkage
into clusters.
When 0 < cutoff < 2, clusterdata forms clusters when inconsistent values
 are greater than cutoff (see the inconsistent function).
When cutoff is an integer and cutoff > 2, then clusterdata
```

```
 interprets cutoff as the maximum number of clusters to keep
 in the hierarchical tree generated by linkage.
The output T is a vector of size m containing a cluster
number for each observation.
T = clusterdata(X,cutoff) is the same as
     Y = pdist(X,'euclid');
     Z = linkage(Y,'single');
     T = cluster(Z,'cutoff',cutoff);


T = clusterdata(X,'param1',val1,'param2',val2,...)
provides more control over the clustering through a set of
parameter/value pairs.
Valid parameters are
     'distance'   Any of the distance metric names allowed by pdist
         (follow the 'minkowski' option by the value of the exponent p)
     'linkage'    Any of the linkage methods allowed by the linkage
         function
     'cutoff'     Cutoff for inconsistent or distance measure
     'maxclust'   Maximum number of clusters to form
     'criterion'  Either 'inconsistent' or 'distance'
     'depth'      Depth for computing inconsistent values
```

D.6 Find_corner

```
Find the number of surfaces
For each surface
```

```
   Find the number of point belonging to this surface
Find the three biggest surfaces (with most points)
Retrieve a point and its normal from each of the three surfaces
Compute the d parameter for the three planes (surfaces)
Compute the intersection of the three planes.
```

D.7 Label_edge

```
Find the number of surfaces
For each pair of surfaces
   Find the number of point belonging to this surface
Find the three biggest surfaces (with most points)
Retrieve a point and its normal from each of the three surfaces
For each line of intersection
   For each of the two intersecting surfaces
      Retrieve a point and its normal
   Compute the intersection of the two surfaces
   For each edge point
      Compute distance between current point and current
          intersection line
      If the distance is below threshold, the point belongs
          to the current intersection. New label.
```