Image-Based Navigation in Space

Project Report
Group 622

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

Department of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø
http://es.aau.dk

AALBORG UNIVERSITY

STUDENT REPORT

Title:
Image Based Navigation in Space

Theme:
Informatics

Project Period:
Spring Semester 2014

Project Group:
622

Participant:
Arthur Paul Schmitt
Panajot Rizo
Vincent Pierre William Renaudat
Ramzi Jiryes

Supervisors:
Thomas B. Moeslund
Chris Bahnsen

Copies: 4

Date of Completion:
May 27, 2014

Synopsis:

AAUSAT's orientation needs to be more accurate, so image based navigation was used in this project in order to add more redundancy to the Attitude Determination and Control System of the satellite. The image based navigation is based on taking picture of Earth at night in order to retrieve at which angles the satellite was when the picture was taken.
A simulation of the satellite's orbit around earth was implemented, along with all the rotation that the satellite does. An image is taken from the simulated satellite and compared with a set of reference images (also taken from the simulated satellite), in order to find its exact orientation.
Template matching with sum of absolute differences was used for the comparison between the simulated satellite image and the reference images. Lossless Compression and decompression modules were implemented in order to efficiently retrieve information from the satellite, such as the image.

Contents

Preface

This report was written by four electronic engineering students as a 6th semester project. The project was proposed by Thomas Moeslund and Jesper Larsen.
Prerequisites for reading this report are a basic understanding of numeric algebra. Knowledge of basic programming in C or C++ is advised although this report does contain little code.
A CD is attached to the report. It contains the report in PDF format, as well as the source code for the image compression, image decompression, reference model and image processing.
Thanks to our supervisors Thomas Moeslund and Chris Bahnsen, as well as Jesper Larsen, for the time they've invested in this project and their valuable feedback.

Aalborg University, May 27, 2014

Arthur Paul Schmitt
<aschm13@student.aau.dk>

Panajot Rizo
<prizo13@student.aau.dk>

Vincent Pierre William Renaudat
<vrenau13@student.aau.dk>

Ramzi Issam Said Jiryes
<rjirye13@student.aau.dk>

Chapter 1

Introduction

The history of developing satellites at Aalborg university (AAU) began with assisting the building of the first Danish satellite, the Orsted satellite which was launched in February 1999 [1]. Since then, three other satellites developed by students are in orbit : AAU-CUBESAT, AAUSATII and AAUSAT3. AAUSAT4 and AAUSAT5 are currently in development [2] and this project is part of the development of AAUSAT5.

AAU-CUBESAT was launched in 2001, the satellite's purpose was to take pictures of Earth, especially Denmark, using an on-board camera. The connection worked well but battery issues meant that only simple telemetry data was downloaded and no pictures were taken.

AAUSATII is the second cubesat and was launched in 2008 (the development started in 2003). Its primary missions were to establish one-way and a two-way communication. The purpose was also to test an Attitude Determination and Control System (ADCS) and to measure gamma ray bursts from outer space. The communication links, both one-way and two-way, were established with the satellite, but some problems with tumbling made it difficult to exchange large data packets. Finally, for unknown reasons, no gamma rays had been measured.

AAUSAT3 was launched the 25^{th} of February 2013 and is still operational. Its primary mission is to test two AIS receiver $AAUSAT3FM$ and EM ready for shipping. The main test area is around Greenland due to the low density of ships but [3] shows that is working fine with the rest of the world. The secondary missions were to test how well the ADCS can control a Low Earth Orbit (LEO) cubesat, to take picture of Earth for publicity, to test the GPS receiver in space and finally to test the FPGA's resilience toward the harsh conditions in space.

AAUSAT4 is still in development at AAU and will have for mission to test a modernised version of the AIS receiver which has shown very good performances with AAUSAT3 and an upgrade of the ADCS system.

The main innovation on AAUSAT5 will be the integration of an imaging system using a camera taking picture of Earth to improve the accuracy of the satellite navigation.

In order to understand why an imaging system is required to improve the precision of the navigation, knowledge about ADCS and its limitation is demanded.

The ADCS system is a major spacecraft system handling spacecraft control such as actuator and spacecraft stabilization[4]. Concerning AAUSAT5, the stabilization is made using a 3-axis control where the three axis are called the Pitch, the Roll and the Yaw which are the rotation angles around respectively the X,Y and Z axis' of the spacecraft. The actuators

of the satellite are magnetic torque rods, which give the satellite a precision between 1 and 10 degrees and are used in LEO, in order to interact with the Earth's magnetic field. This is relevant since the satellite will orbit at an altitude of 600km. The satellite is also using sensors that are part of the ADCS like a magnetometer, a sun sensor and a rate gyros sensor. However, those sensors are not error-free and might produce some errors due to noise and disturbances such as aerodynamic drag, magnetic torque and sun radiation.

Those sensors errors imply that the actual angles (Pitch, Roll, Yaw) of the satellite are not exactly the ones measured by the sensors. To improve the precision and correct the determination errors, an imaging system is used to add redundancy to the system and provide a stable solution.

The use of imaging system is widely used to have a very accurate knowledge of the orientation of a spacecraft [5]. For instance, spacecraft observing stars (Hubble telescope), or traveling and orbiting other planet, satellite, stars or asteroids, i.e. Galileo orbiting Jupiter [6] and Rosetta, a comet orbiter/lander [7].

This project brings a solution to the sensor issue by using a camera pointing toward Earth. The camera takes photos of Earth at night and the image processing program tries to recognize noticeable patterns on the surface (city lights, coasts, mountains,...). Once the image has been matched with the reference, the program finds the angles' errors and sends its correction to the ADCS system. In the meantime the picture is sent to Earth.

This project is about implementing the program that receives the picture from the camera and the data from the sensor, then the program processes the picture and sends a compressed copy to the communication module in order to forward it to Earth. It also sends the orientation corrections to the ADCS system.

Chapter 2

Navigation in space theory

This chapter gives presents the background of the project. This chapter is divided in two parts. The first one is about how a spacecraft is located in orbit and how a spacecraft can modify its orbit. The second part is a description of the different sensors that make AAUSAT able to navigate and control its attitude while orbiting. It also explains the need for this project.

2.1 Space coordinates

2.1.1 The celestial sphere

When it's about to describe the locations of an object in the sky, we do not use the Earth reference location (longitude and latitude). The used reference is called the celestial sphere. The celestial sphere has an infinite radius and the center of the celestial body (planet, star, satellite,...) is the center of the celestial sphere. There are some noticeable location on the celestial sphere which are :

- The Zenith, which is the point on the celestial sphere directly overhead for an observer.
- The Nadir, which is the direction opposite to the zenith.
- The Meridian, which is an imaginary arc passing through the celestial poles and the zenith

Those elements are also depicted in the figure 2.1

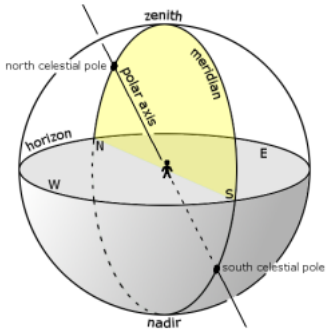

Figure 2.1: Representation of the zenith, nadir and a meridian on a celestial sphere [8]

2.1.2 Coordinates systems

There are several coordinate systems used to locate an object in the celestial sphere. The system depicted below is the international reference system and the one used by AAUSATs.

International Celestial Reference System (ICRS).

First of all, the ICRS [9], is the fundamental celestial reference system and is composed of two coordinates, the declination (DEC) and the right ascension (RA). The DEC is the celestial sphere equivalent of latitude. A positive DEC means North and a negative one means South. The RA is the celestial sphere equivalent of the longitude. It can be expressed in degrees but it is more commonly expressed in hours, minutes and seconds; where one hour of RA is equal to 15 degrees of sky rotation. This comes from the fact that 360 degrees are equal to 24 hours. Finally the celestial equator makes an angle of 23.4 degrees with the ecliptic plane for the Earth. Figure 2.2 shows an example of how DEC and RA work.

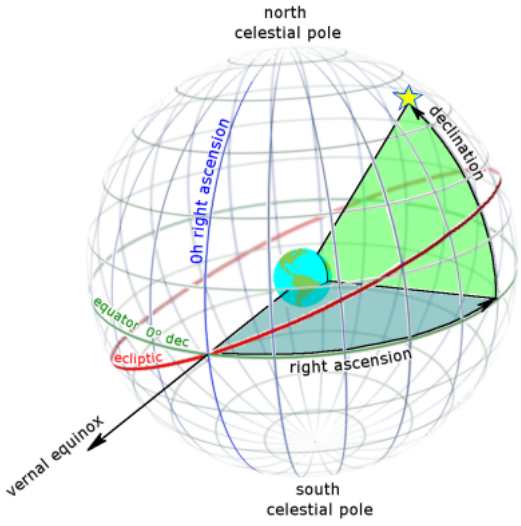

Figure 2.2: Declination and Right Ascension of an object on the celestial sphere.[10]

2.1.3 Apoapsis and Periapsis

The apoapsis and periapsis are noticeable points of the orbit of an object orbiting around a star, a planet, a satellite, ... The apoapsis is the farthest point of the orbit from the center of the celestial body while the periapsis is the closest (i.e. figure 2.3). On Earth, those points are called the apogee and the perigee.

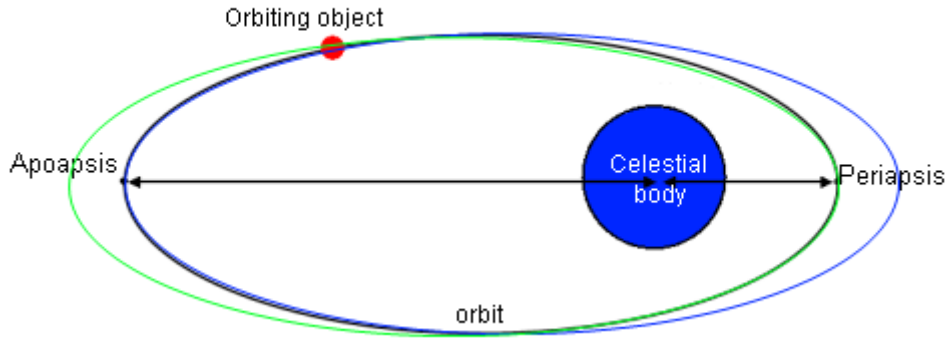

Figure 2.3: Schematic representation of the apoapsis and periapsis and their properties on an object's orbit.

The apoapsis and periapsis also have useful properties for space navigation :

- If the spacecraft applies more energy at the apoapsis, the periapsis' altitude is raised. This property is pictured in blue in figure 2.3.
- If the spacecraft applies more energy at the periapsis, the apoapsis' altitude is raised. This property is pictured in green in figure 2.3

The opposite effect is obtained by removing energy at apoapsis or periapsis. The most common way for a spacecraft to apply or remove energy is the thrust.

When the spacecraft applies or remove energy at the apoapsis or periapsis, in order to raise or lower the apoapsis or periapsis altitude, the energy is applied in the opposite direction of where the spacecraft is heading to and removed in the direction the spacecraft is coming from. For example a spacecraft wants to raise its apoapsis. When it is at its periapsis, it will apply a thrust in order to gain speed while still following its orbit path. When the wanted apoapsis is reached (the spacecraft is still at the periapsis), the thrust stops and the spacecraft is orbiting in its new orbit. This manoeuver raises the altitude of the apoapsis of the spacecraft's orbit. Now the spacecraft is at its apoapsis and wants to lower its periapsis, so it has to remove energy. In order to do so, it will apply a thrust in the direction the spacecraft is heading to, in order to slow the spacecraft and so lower the periapsis. When the wanted altitude is reached, the thrust is stopped.

In the case of AAUSAT, the orbit is almost circular at an altitude of 600km. We then neglect the difference between the apogee and the perigee and assume that the satellite is orbiting circularly at 600km from the sea level.

2.2 Spacecraft navigation

This section describes the different modules used by AAUSAT to navigate and control its position and orientation while orbiting. As explained in the Introduction (1), the ADCS is the module managing the controls of the satellite by using the navigation modules. These navigation modules are :

- a magnetometer
- a sun sensor
- a gyro rate sensor

2.2.1 Magnetometer

A magnetometer is a tool used to measure the strength and in some cases, the direction of the magnetic field at a point in space or to measure the magnetization of a magnetic material [11].The magnetometers can be used on spacecraft and satellites for attitude sensing. The magnetometer used for AAUSATs senses Earth's magnetic field and with the sensed data the ADCS corrects the attitude of the satellite. Even though Earth magnetic field is well known 2.4 noise and disturbance can be induced by the satellite itself or cosmic rays. The attitude control of the satellite must then be handled by more than one sensor.

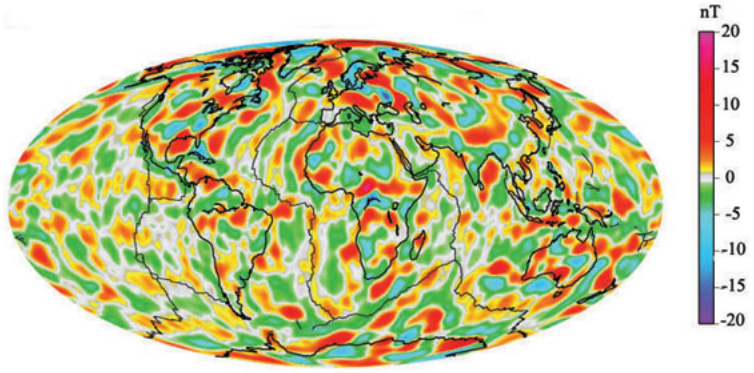

Figure 2.4: Lithospheric Magnetic Anomalies. nT = nanoteslas. The color bar indicates areas with positive and negative magnetic fields. [12]

2.2.2 Sun Sensor

Sun sensors are devices that sense the direction to the sun. Sun sensors are one of the most common attitude determination systems , these sensors determine the satellite's orientation relatively to the sun by measuring the amount of light or shadow on them. [13]

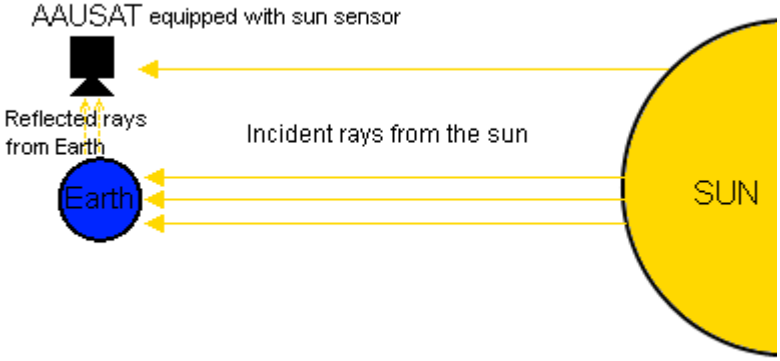

Figure 2.5: Illustration of where the data collected from the sun sensor come from

The amount of light captured by the sensor creates a current that allows the ADCS to know what's the attitude of the satellite. However, as shown in figure 2.5 some noise can be induced due to Earth reflection or shadows from other spacecraft. This noise makes the sun sensor have some imprecisions.

2.2.3 Gyro rate sensor

A gyro rate sensor is a device that senses angular velocity [14]. In another words, a gyro sensor can sense rotational motion and sense changes in its orientation. Therefore, in order for AAUSAT to be more accurate in its rotations, a gyro sensor is used. In order to sense the angular velocity, the gyro sensors use the Coriolis force (i.e. appendix A). The gyro rate sensor, also called angular rate sensor, is measuring the angular velocity along the three axis of the spacecraft : X, Y and Z. The three possible rotations around those axis are called Pitch, Roll and Yaw. In most cases, The roll is the rotation around the X axis, the Yaw is around the Z axis and the pitch is around the Y axis. AAUSAT's ADCS is using the gyro rate sensor to gain accuracy when changing attitude (rotating for instance)and to help maintain a stable state when the attitude should not change. Even though the accuracy is increased, an imprecision domain remains when the exact orientation of the satellite is wanted.

2.2.4 The Imprecision domain

The imprecision domain means that for a particular range of angles, it is impossible to say precisely if the angles given by the three sensors are the actual angles of the satellite. This is due to noise and interferences at the sensors and original sensor inaccuracy. The noise and interferences caused by the sensors are the main responsible for those imprecisions. For the magnetometer, they are caused by the magnetic field of the component of the satellite, indeed, AAUSAT being a cubesat, it is not possible to place the magnetometers far from all the other devices. The magnetic fields created by other satellite, the sun, and other celestial bodies in the solar system also have a small impact on the precision of the magnetometer. for example, the ADCS receives a message from the magnetometer saying that the satellite should rotate 2 degrees to be perfectly positioned toward the center of Earth (X-axis going straight to the center of earth). But because of the noise, the magnetometer said 2 degrees instead of 1.4 degrees and so the ADCS will think the satellite is looking straight to the center of Earth, whereas it is looking at some other point. The problem is the same with the sun sensor but the noise and interferences are not caused by magnetic fields but by reflected light from Earth and by objects between the sun and AAUSAT (a satellite for instance) creating a shadow. To counter imprecisions induced by the magnetometer and the sun sensor, which are, according to Jesper A. Larsen $\pm$ 5 degrees on each rotation axis (Pitch,Roll and Yaw) an additional system is needed.

2.2.5 Optical system

Spacecraft equipped with imaging instruments, such as a camera, can use them to observe the spacecraft's destination planet or other celestial body (satellite, asteroid,...) against a known background star field. These images are called opnav images. The observations are carefully planned and uplinked in advance as part of the command sequence development process. Opnav images are downlinked. [5] As said in chapter 1 this system is already widely used. For example, the Hubble telescope needs to look at the exact same position in the universe sometimes for several month in order to receive enough light to see how was the universe billion of years ago, in its first years. To stay in the same position, the telescope is using a lot of sensors, in particular an optical navigation system using a star field to correct the imprecisions of the other sensors. Thank to this system the Hubble telescope is able to show us how the universe was at the very beginning and how it has evolved and still evolves. Figure 2.6 is the longest exposure picture taken by the Hubble telescope.

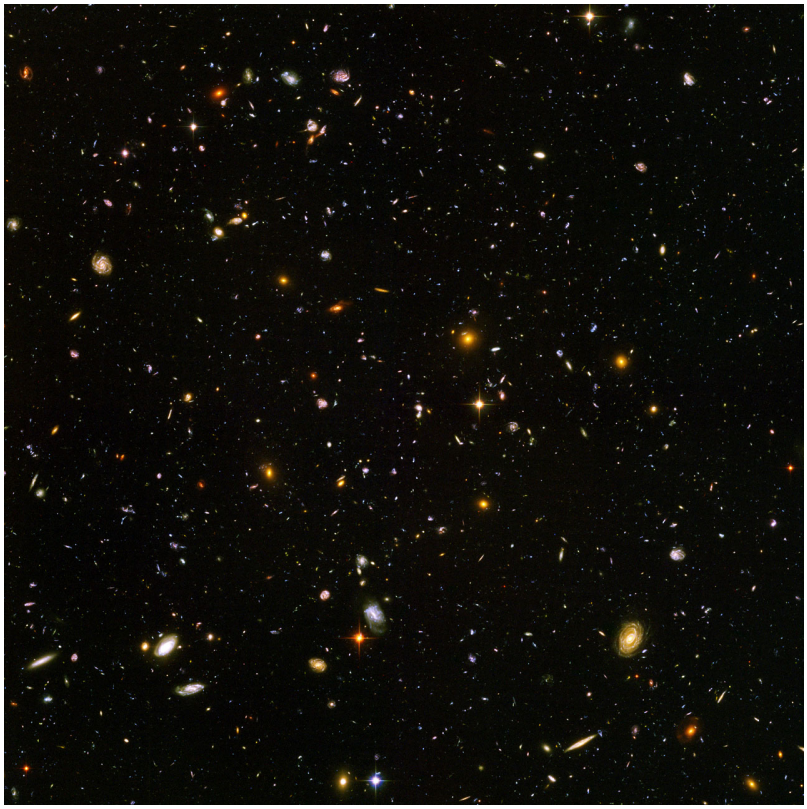

Figure 2.6: Fraction of the Hubble Ultra Deep Field picture showing the earliest galaxies of the universe [15]

In the case of this project, the opnavs are not pictures of a star field but picture of the Earth at night.Then to add redundancy to the ADCS and reduce the imprecision of the attitude determination, a reference model of Earth surface at night is used in order to determine the attitude of the spacecraft by comparing the opnav to the city's light patterns. This project is about implementing the image processing part of the imaging system of the satellite.

Chapter 3

Scenario and Requirements

3.1 Scenario description

The next generation of AAUSAT is going to use on-board camera in order to improve ADCS's accuracy and thus allow the satellite to have a precise knowledge of its attitude. The goal of this project is to find the best approximation of the actual attitude of the satellite using image processing on a taken picture from the camera. Since the satellite has not been built yet, it is impossible to work with real images of Earth. In order to counter this issue, a reference model of Earth is built and a simulated opnav is taken from this reference model (a screen-shot for example). Then a simulated camera is positioned at a given position around the reference model of Earth. This position corresponds to the same position the real satellite would have been in if the picture was taken by the satellite. In this case, the coordinates correspond to where the simulated opnav was taken. Once the camera is positioned, it will start rotating to cover the whole the imprecision domain of the satellite. For each angle incrementation, a reference picture is taken by the camera. This reference pictures and the simulated opnavs are compared using image processing. If a match is found, then the program sends the determined attitude to the ADCS module of the satellite, if not the program continue to go through all the angles. If there hadn't been a match when all the angles had been processed, then the simulated opnav (the one from the satellite in real life) is saved and an error message is sent to the satellite.

Another part of this project is to handle the data compression and decompression in order to be able to send ,in an effective way, data from the satellite to Earth. This data is composed of the picture taken by the satellite's camera (which is the simulated opnav for the purpose of this project), the position of the satellite in the ICRS system (see 2.1.2) and a time stamp. The data are compressed on-board, then they are sent to the communication module which send them to Earth. When they arrived, they are decompressed and ready to be used in order to determine the true attitude of the satellite.

The following flowchart (figure 3.1) summarizes how the different part of the project interact with each other.

In figure 3.1, the different parts of the project are divided in modules. There are four modules, the Compression module, the Decompression module, the Reference module and the Image Processing module. Each of those module are composed by processes that represent the core methods of the corresponding module. The process called "Simulate opnav" is not part of any module and is here to simulate the picture which would have been taken from the camera and send it to the compression module. Then the simulated opnav and the

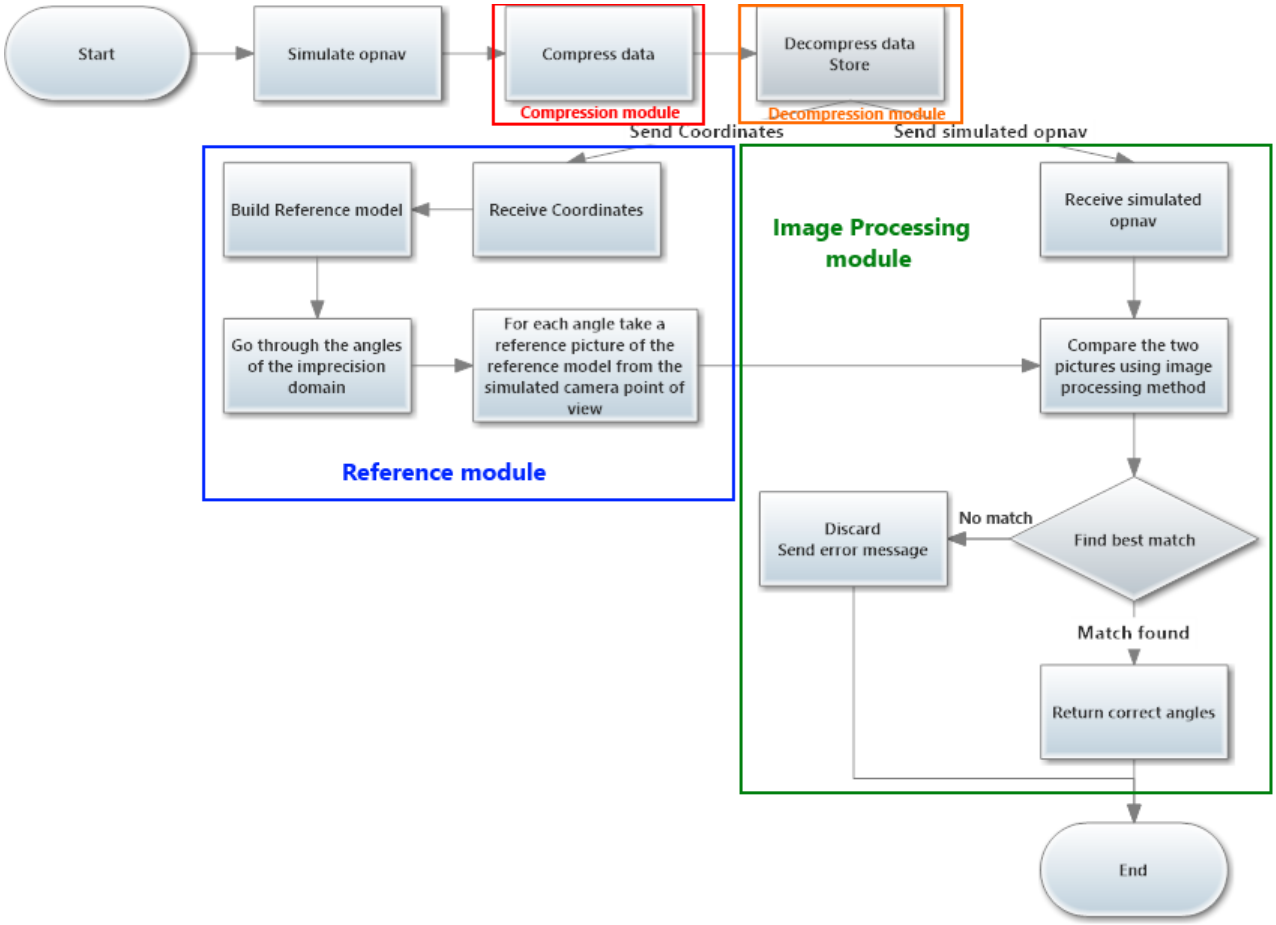

Figure 3.1: Flowchart of the project, each process is described inside the small rectangles.

coordinates of where it was taken are compressed in the compression module. The compressed data is sent to the decompression module where the simulated opnav and the coordinates are decompressed. The coordinates are sent to the reference module and the simulated opnav is sent to the image processing module. In the reference module, The coordinates are received and translated into Cartesian system, The reference model is built and the simulated camera positioned at the received coordinates. Then the camera rotates along its axis in order to cover the imprecision domain and for each angle it provides a reference image. When all the angles of the imprecision domain have been processed, all the reference images are sent to the image processing module for comparison. In the image processing module, the simulated opnav and the reference images from the reference module are received. Then the simulated opnav is compared with all the reference images using image processing method. Finally, the best match between the simulated opnav and a reference image is kept and if it is good enough, the angles with which the matched reference picture was taken are returned to the user. If no satisfactory matches are found, the simulated opnav is discarded and an error message is returned.

To test the overall system, a test of the implementation of the flowchart in figure 3.1 will be done using noiseless simulated opnav and noisy simulated opnav. The test will consist

of compressing and decompressing the simulated opnav and then compared the simulated opnav with the reference images from the reference module.

3.2 System Requirements

In this section the general requirements for the project will be elaborated. The system requirements are split up into the different subsystems being Compression module, Decompression module, Reference Model, and the Image Processing module.

1. **Compression module**
   (a) Compression module must receive image from satellite *(section 4.1)*.
   (b) Compression must be lossless *(section 4.1)*.
   (c) Compressed image file must be smaller than original raw image *(section 4.1)*.
   (d) Send compressed data to the decompression module *(section 4.1)*.

2. **Decompression module**
   (a) Decompression module must receive image from compression module *(section 4.2)*.
   (b) Decompression module must save the decompressed data as an image *(section 4.2)*.
   (c) Decompression module must send the decompressed data to the image processing module *(section 4.2)*.

3. **Reference Model**
   (a) Must be able to generate and load a reference model of the simulated Earth object *(section 4.3)*.
   (b) Must simulate the correct position of the satellite(camera) on the model *(section 4.3)*.
   (c) Must simulate the imprecision domain *(section 4.3)*.
   (d) While simulating the imprecision domain 2.2.4, the camera must provide a reference image *(section 4.3)*.

4. **Image Processing module**
   (a) Must receive image from decompression module *(section 4.4)*.
   (b) Must receive the reference images from reference module *(section 4.4)*.

(c) The Image Processing module must compare two images and provide a measure on whether or not they match *(section 4.4)*.

(d) Must find the best match from several images *(section 4.4)*.

5. **Overall System**

(a) All the modules mentioned above must communicate with one another and react to one another.

These are the requirements for the whole system and based on these requirements the design phase of the system can be further looked into in chapter 4.

Chapter 4

System Design

The project scenario and the requirements set in Chapter 3 depict how the project should globally work and within which limitations. This chapter gives a more in depth approach by depicting precisely how each module of the project is designed. The four modules of the system are listed below :

- Compression Module
- Decompression Module
- Reference Model
- Image Processing Module

Every module has its own functions. This chapter contains description of all the functions used in the implementation of each of the modules along with the parameters and arguments used. The modules are explained following the chronological order in which they are called in the main system. This order can be seen in figure 3.1.

4.1 Compression Module

The first module to be called is the compression module. This module is an on-board module and cannot be otherwise. However, for the purpose of the project and because of the conditions (the satellite doesn't exist yet), The compression module is processing on Earth. If the conditions were ideal (the satellite is orbiting and the module is on-board), the role of the compression module would be to obtain data from the satellite. Those data would be composed of the picture just taken by the camera and the position of the satellite in the celestial sphere which would be given by the ADCS module. Its main requirement is to compress the data without losing any information. Figure 4.1 depicts the use case of the compression module in ideal conditions. The camera module and the ADCS module are providing data to compress to the compression module. The camera provides the opnav and the ADCS provides the current coordinates of the satellite on the celestial sphere (RA-DEC, i.e. section 2.1.2. Those data are then compressed and sent to the decompression module on Earth via a communication module. Because the Compression module is processing on the satellite, the compression program has to be optimized to use as little memory and computing power as possible.

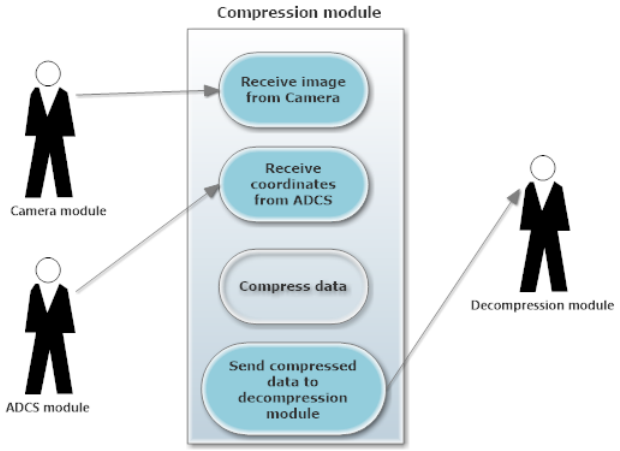

Figure 4.1: Use case diagram of the Compression Module on ideal conditions

However, the conditions of development of the project are not the ideal ones and it is thus required to simulate the communication between the satellite and the compression module. The use case in figure 4.2 shows the simulated design of the communication between the satellite and the project. The satellite is replaced by the user and the user provides a simulated opnav which is the picture taken from the reference model (see section 3.1)and provides the coordinates on celestial sphere of where the simulated opnav was taken.

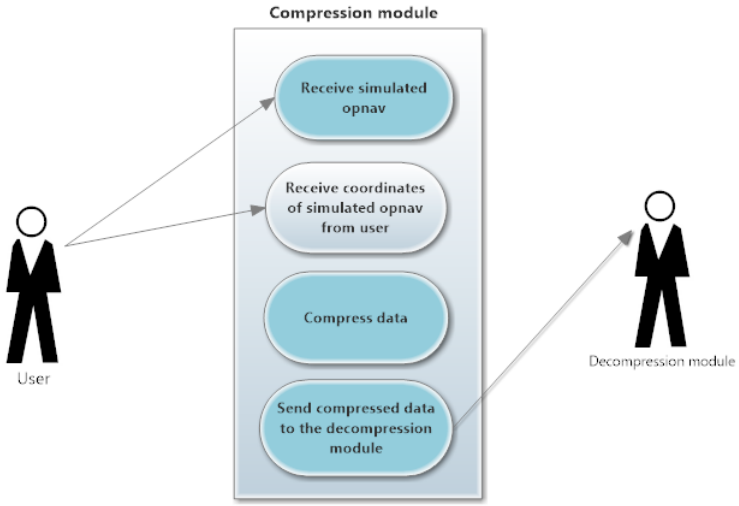

Figure 4.2: Use case diagram of the Compression module with simulated satellite

From this point, it will be considered that the project works with a simulated satellite, so the opnav is referred as a simulated opnav and the coordinates in RA-DEC are the ones where the simulated opnav had been taken on the celestial sphere.

4.2 Decompression Module

The Decompression module, which operates on Earth has the requirement to decompress the received data from the compression module in order for the other modules to process them. Once the data decompressed, the simulated opnav is sent to the image processing module and and the coordinates on the celestial sphere are sent to the reference module. In the meantime, the simulated opnav is stored on the computer so it can still be available if anything unexpected should happen or if someone wants to use or look at the opnav for other purposes than knowing what are the correct angles of the satellite. The use case depicted in figure 4.3 shows the input and output of the decompression module and its functions.

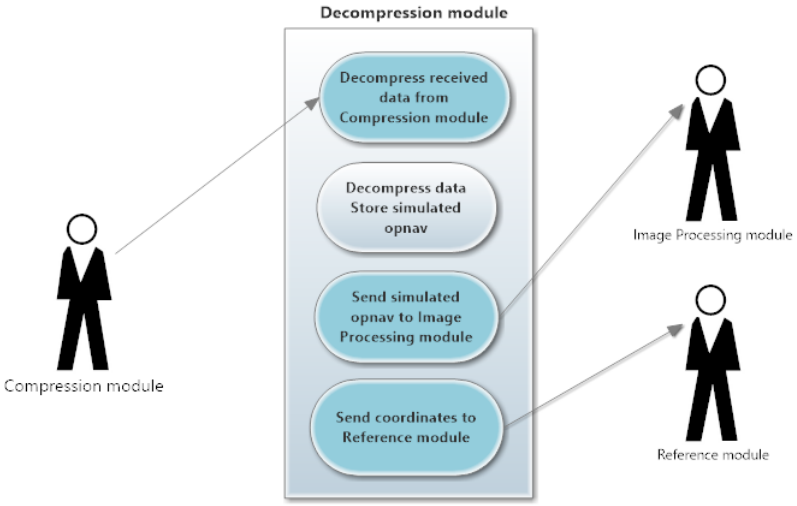

Figure 4.3: Use Case Diagram of the De Compression Module

4.3 Reference Module

The main requirements of the reference module are that it has to build a reference model in order to generate reference images for each angle of the imprecision domain and then send those reference images to the image processing module for comparison. The reference model is a 3D representation of Earth at night with city lights patterns. The coordinates of the simulated satellite (RA-DEC) are received from the decompression module and are translated into Cartesian coordinates in the reference model in order to position a simulated camera that should be able to take picture of the 3D model of Earth. This simulated camera should also be able to perform a rotation on itself along two axis, those rotations are the pitch and the yaw. The roll is performed later directly on the simulated opnavs. The reason why the three rotation are not performed at the same time is to make the design and the implementation simpler, regardless of the programming language. For each pitch and yaw, a reference image of what the simulated camera sees is saved and transferred to the image processing module. The use case in figure 4.4 shows the input/output and processes of the reference module.

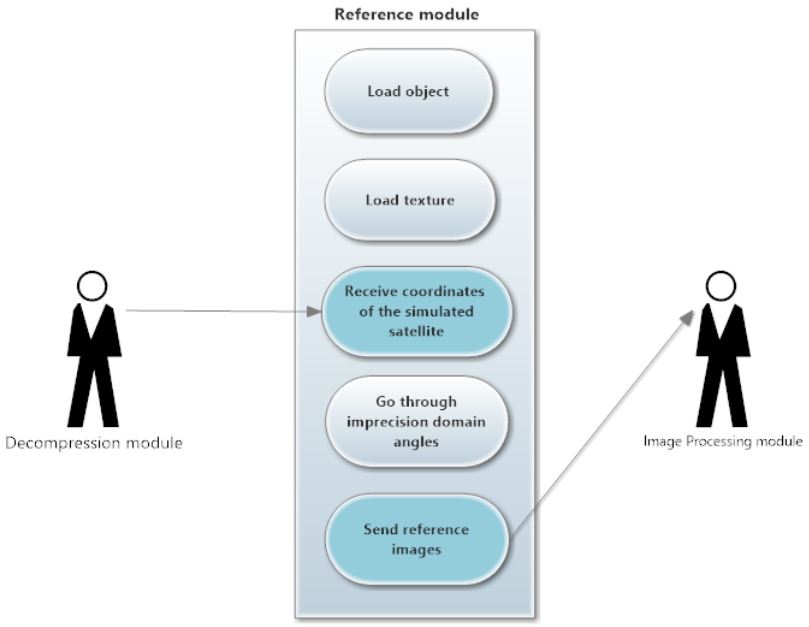

Figure 4.4: Use Case Diagram of the Reference Module

4.4 Image Processing Module

As said before, the image processing module receives the simulated opnav from the decompression module and the reference images from the reference module. The reference images are stored in a 3-dimensional array. Each dimension is related to one rotation angle (Pitch, Roll, Yaw). The pitch and the yaw had already been processed in the reference module, and to determine the roll, each reference image is rotated and every new reference image resulting from the rotation is stored in the array accordingly to its pitch and yaw. Now that all the reference images are created, they are process one by one using template matching with the simulated opnav. Every template matching gives a result and the best match is kept as being the actual attitude of the satellite. The corrected attitude (or angles) is then returned to the user.

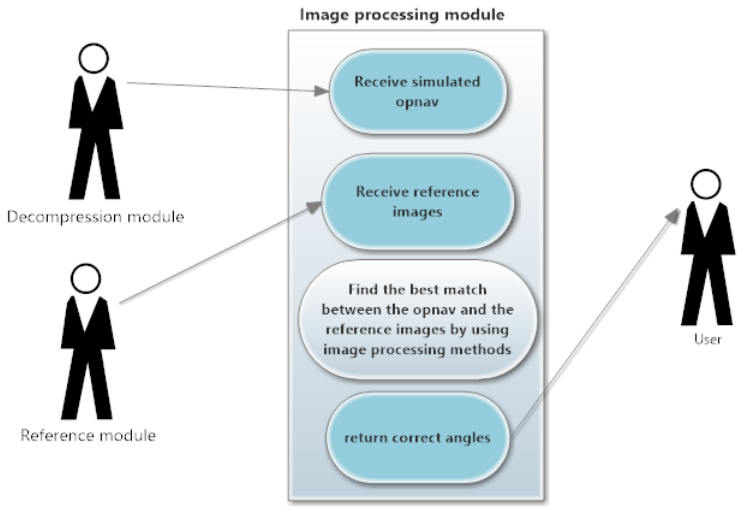

Figure 4.5: Use Case Diagram of the Image Processing Module

4.5 Domain model

The domain model of the project pictured in figure 4.6 is another representation of the project. Here is shown how the different modules interact with each other and what are the involved functions. This representation gives a global visibility of all the project and how it is designed.

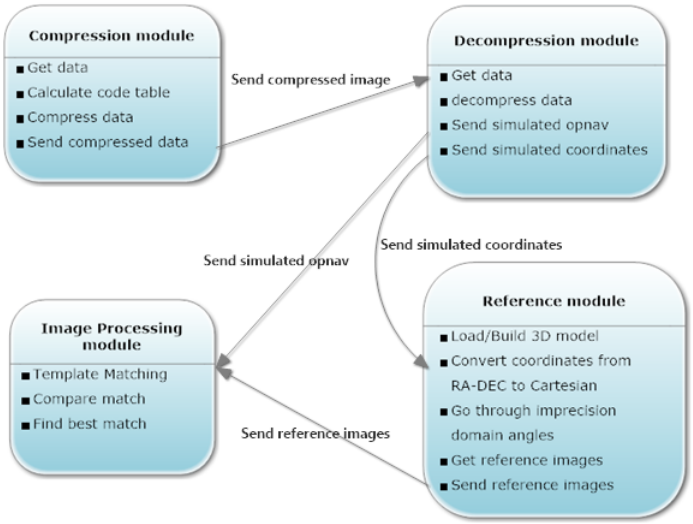

Figure 4.6: Domain model of the system

Chapter 5

Data compression and decompression

5.1 Data Compression Theory

A picture taken by the satellite camera, needs to be sent back to Earth. Due to limited transmission bandwidth, it is preferable to reduce the size of the image file, which makes compressing it an interesting possibility. The image quality is very important for the image processing that will follow. There are many image compression methods available to satisfy this need, which are split into 2 main categories, lossless and lossy. Lossy methods do not allow you to retrieve all the information from the compressed file, some of it is lost. This is useful when the information loss is not noticeable to the human eye in the decompressed image, so the losses are a sort of trimming of the usable information. Here, the image processing program will benefit from more information as it will increase the reliability and certainty of its decisions. For example, template matching with more information reduces the error rate because there are more pixels to compare. Lossless compression, on the other hand, as its name indicates, will provide an exact replica of the original after it has been in turn compressed and decompressed. There is no loss in data or information.

5.1.1 Lossless Compression

Entropy

Entropy is used to measure the information content of a source. It can be determined through both the source's statistical or predictive properties. Shannon's source coding theorem dictates that the optimal code length for a symbol is

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{5.1}$$

where p is the probability of the input symbol. H gives us the optimal code length in bits per symbol [16]. The compression rate is based on the entropy, given by

$$Compression\% = \frac{bits/symbol - calculated\ entropy}{bits/symbol} \tag{5.2}$$

The lower the calculated entropy, the higher the compression ratio, and vice versa. The entropy can also be compared to the actual code length after compression to see how effective it is.

Entropy encoding consists of designating a prefix-free code to each unique symbol. The compression is done by replacing each symbol by its output codeword [17]. The codewords are determined according to the probability of the symbol appearing. The more common the symbol, the shorter the codeword.

The probability of the input symbol can be determined through 2 approaches. In the first case, both the compressor and decompressor have a table where each symbol has a corresponding code. This could be used for compressing text for example, when you know that for a particular language, certain letters are more common than others. This way, the decoding table doesn't have to be sent, which makes the compressed file smaller. Another way to do it is to calculate the frequency of each symbol first, proceed with the compression and then send the coding table along with the compressed image. This method is more robust as it is not susceptible to unexpected variations in symbols frequencies which would render the compression ineffective [18].

Huffman Coding

Huffman encoding is one of the most popular entropy methods, because it yields the smallest possible number of code symbols per source code. The first step is to list the different symbols in order of their probabilities. Afterwards, the two symbols of the lowest probability are combined into one, of a probability equal to the sum of the two of the symbols making it. This procedure, called source reduction, is done until there are only 2 symbols left.

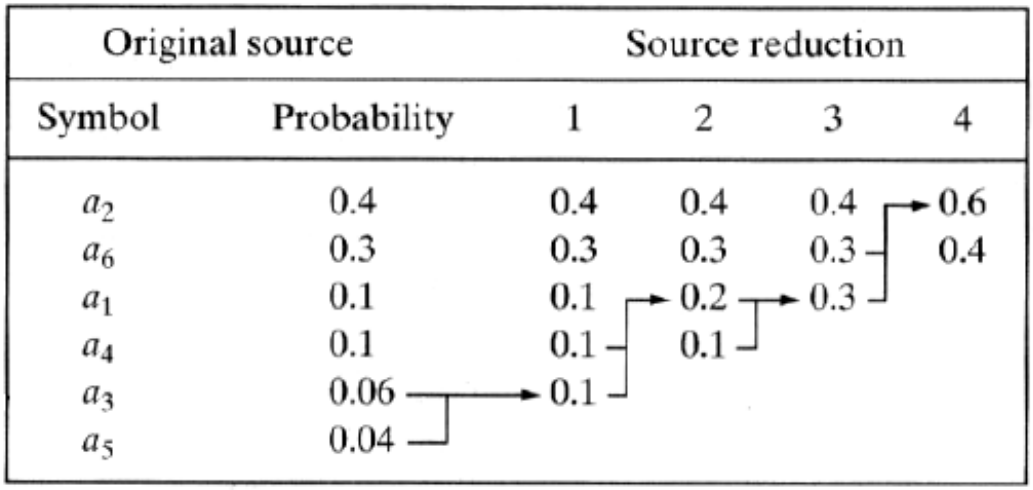

Figure 5.1: Huffman Encoding Part 1

Figure 5.1 illustrates how the process is started.
Now, each reduced source has to be replaced by a code. Figure 5.2 illustrates how the codes for each symbol are determined. The two final compound symbols are attributed the values of 0 and 1. For each reduced source, a digit is added. In the end, the original symbols are replaced by their respective Huffman codes, and a table of these codes is kept for decoding

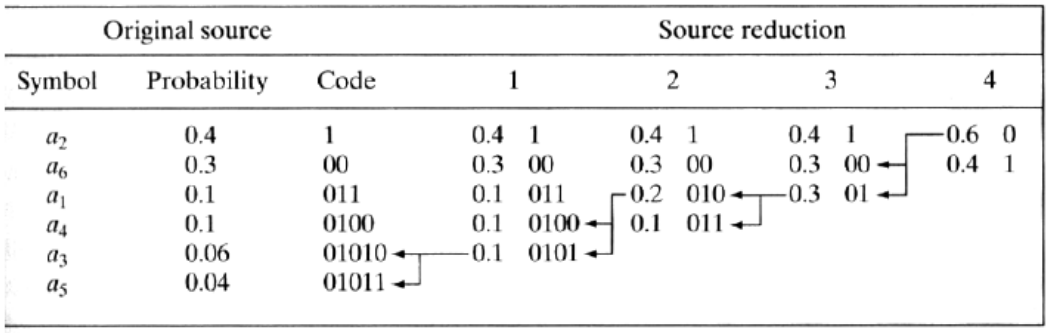

Figure 5.2: Huffman Encoding Part 2

purposes.
Huffman encoding attributes the shortest codes to the most frequent symbols so that they take the least amount of memory.

$$Input \begin{bmatrix} 0 & 1 & 8 \\ 8 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

$$Output \begin{bmatrix} 0 & 110 & 111 \\ 111 & 0 & 0 \\ 10 & 0 & 10 \end{bmatrix}$$

Symbol	Code
0	0
1	110
2	10
8	111

Table 5.1: Huffman code table

If the original matrix is made up of 8 bit unsigned integers, like in a grayscale image, then the average bits/symbol is of 8 of course. For the Huffman encoded matrix, the average code length can be calculated by summing the symbol lengths multiplied by their probability or frequency:

$$L_{avg} = \frac{4}{9} \cdot 1 + \frac{2}{9} \cdot 2 + \frac{2}{9} \cdot 3 + \frac{1}{9} \cdot 3 = \frac{17}{9} = 1.8 bits/symbol \tag{5.3}$$

The Huffman encoding is over 4 times more efficient in this case

Using the formula from 5.1, the entropy is

$$H(X) = -(\frac{4}{9}\log_2\frac{4}{9} + 2 \cdot \frac{2}{9}\log_2\frac{2}{9} \cdot + \frac{1}{9}\log_2\frac{1}{9}) = 1.67 bits/symbol \tag{5.4}$$

which is close to the actual average length.

Run Length Encoding

This method takes advantage of sequences of repetitive data. A run of data is stored as a value and a count. The easiest way to understand this is through an example:
Input:
BBBBWWWBBWBWWBWBBBBBBBWWWWWWW
Output:
4B3W2B1W1B2W1B1W7B7W

This could be implemented in the project, depending on the typical image that is to be transmitted. For example, if the picture is taken at night, then there could be a lot of black surrounding the illuminated cities. Run length coding would greatly compress this area of data. However, it could prove inefficient in the illuminated areas, which are small and isolated.

Arithmetic Coding

Arithmetic coding is another form of entropy coding, which essentially uses the probabilities of the input symbols to compute a floating point, or a fraction [19]. It starts with a probability line 0-1 and assigns a range in this line to each symbol. The higher probability symbols are granted a greater range. Table 5.2 is an example of 3 symbols and their ranges compared

Symbol	Probability	Range
a	2	[0, 0.5)
b	1	[0.5, 0.75)
c	1	[0.75, 1)

Table 5.2: Arithmetic Coding Symbols and Properties

to their probability. Once the ranges have been determined, the coding can begin using the following algorithm [20]:

- Low=0
- High=1
- Loop through all the desired symbols:
  - Range = High - Low
  - High = Low + Range * High Range of the symbol being coded
  - Low = Low + Range * Low Range of the symbol being coded

Here, *Range* is tracking the range for the next symbol, and *High* and *Low* specify the output number.
The decoding is done as follows. The first symbol is found by determining the range of the floating point, and then

- Range = High Range of the symbol - Low Range of the symbol
- Number = Number - Low Range of the symbol
- Number = Number / Range

5.1.2 Algorithmic complexity

Huffman: (linear-logarithmic)

$$O(n \log(n)) \tag{5.5}$$

Run Length: (linear)

$$O(n) \tag{5.6}$$

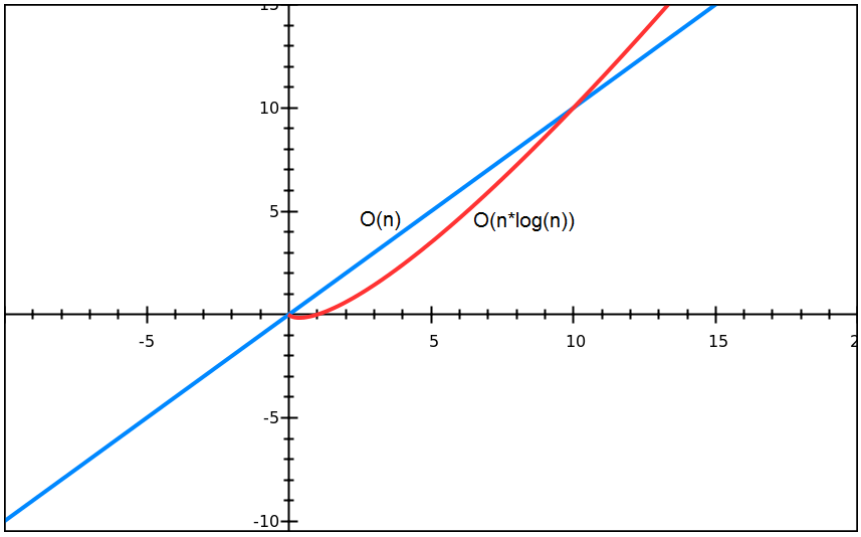

Figure 5.3: Algorithmic Complexity for Huffman Coding and RLE

Run Length coding is not a good option for grayscale image compression because there are too many possible values (256). RLE is better suited for binary images (each pixel is only black or white). Although there appear to be large areas of black in the simulated opnav, after looking at the gray values, the color fades, meaning that adjacent pixels are of a different value, although similar to each other.
Arithmetic coding and Huffman coding present themselves as viable options for image compression. Both have a high compression ratio and are adaptive to the different symbol probabilities. Arithmetic coding can achieve a slightly higher compression than Huffman [19] for images with until 1 million pixels (around 1%, depending on the image), but a much higher compression for images with 4 million. However, the satellite images have a resolution of 236 x 234, meaning that the difference is negligible between the two. The two algorithms differ in their computing power demands. Arithmetic coding takes over twice as long to compress an image [19]. As the satellite's computing power is limited, the performance issue trumps the slight compression advantage that Arithmetic encoding.
Huffman coding is therefore the best solution for the satellite image compression. It can be easily implemented in C and C++ as it mainly consists of building a binary tree. It can also adapt the symbol codes to tailor each different image to achieve a high compression ratio.

5.2 Image compression and decompression

5.2.1 Image Compression Module

The image compression module can be split into several steps depicted by figure 5.4

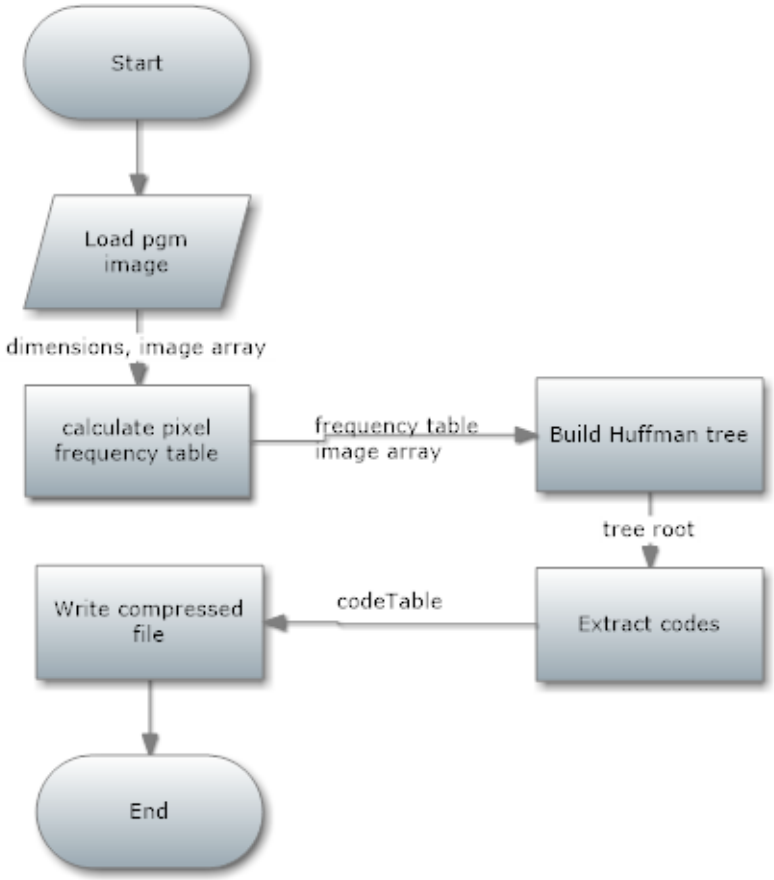

Figure 5.4: Image compression order of events

The first function loads the simulated opnav data. Each gray value frequency is calculated next, and sent with the image array to a function that builds the Huffman tree. Once the tree has been reduced, the codes for each gray value are extracted and the compressed file writing can begin. The following subsections describe how each process is carried out.

Load .pgm image

To simulate the incoming image captured by the satellite camera, the function first loads a portable gray map (.pgm) file into the structure depicted in listing 5.1.

```
1 typedef struct pgm { //pgm structure
2   int w; //width
3   int h; //height
4   int max; //maximum color component
5   int** pData; //two dimensional array of gray values
6 } pgm;
```

Listing 5.1: pgm file structure

The PGM format is the lowest common denominator grayscale file format [21]. The length of the header illustrates this fact as there are only 3 properties of the image loaded into it. The width and height of the image, provided by the header, will be used to determine the number of elements accessible through *pData*. The maximum color component will always be 255 since the image is in grayscale format.

Calculate pixel frequency

A histogram for the image is stored in the array *frequency[256]*, where *frequency[i]* is equal to the frequency of the color component i in the image. The frequency is calculated by simply going through each value of pixel array and incrementing the respective frequency. This is one of the key components for implementing the Huffman algorithm.

Create and fill Huffman tree

```
1  // A Huffman tree node
2  struct MinHeapNode
3  {
4      int data;  // Color component
5      unsigned freq;  // Frequency of color component
6      struct MinHeapNode *left, *right; // Node children
7  };
8
9  // A Min Heap:  Collection of Huffman tree nodes
10 struct MinHeap
11 {
12     unsigned size;     // Current size of min heap
13     unsigned capacity;   // capacity of min heap
14     struct MinHeapNode **array;  // Pointer to array of node pointers
15 };
```

Listing 5.2: Huffman tree and node structures

Listing 5.2 shows the 2 core elements in building a Huffman tree. The first, *MinHeapNode* is a binary tree node. Each node of the tree will have a unique color component ranging between 0 and 255, and a frequency, calculated previously. *MinHeap* is the tree structure, essentially pointing towards each node [22].

The construction of the Huffman tree is carried out as depicted in the following flowchart:

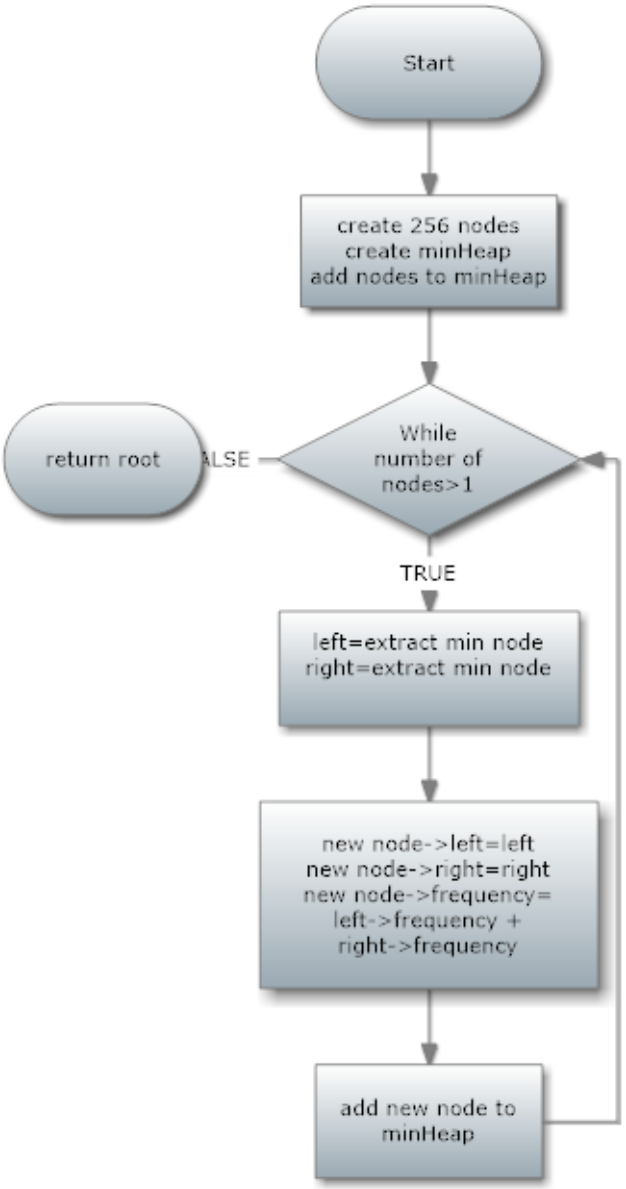

Figure 5.5: Flowchart of the Huffman algorithm implementation

At first, 256 nodes are created and added to the node collection. At this point, they are all accessible through the minHeap structure, but they are not aligned in any particular order. The source reduction is begun by extracting the two nodes with the lowest frequency from the minHeap. They are still in memory, only they are no longer in the minHeap. These two nodes will become the children of a new node, and the sum of their frequencies will be attributed to it. This new node will then be added to the minHeap, and the process will continue until there is only 1 node left. The last one is the root of the tree, which can be used to access any node.

Extract codes from tree

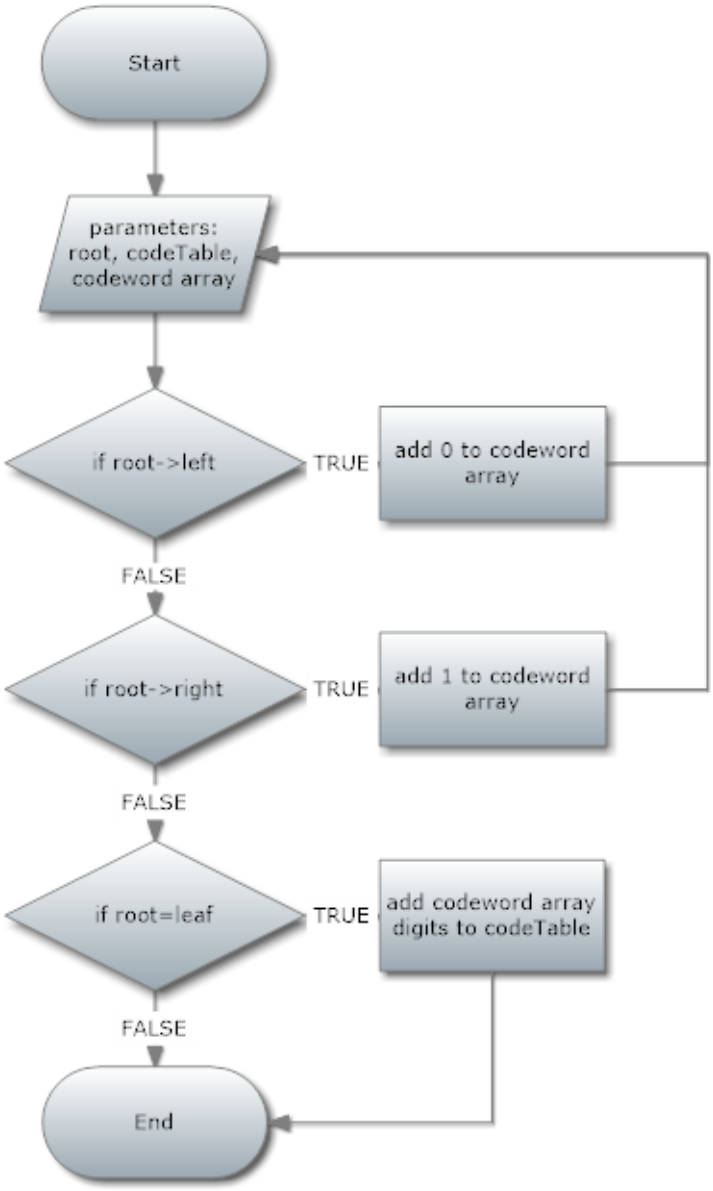

Figure 5.6: Flowchart of the Extract Codes function

Once the tree is built, the Huffman codes can be retrieved from it. The flowchart 5.6 illustrates how the process is executed. The entirety of the tree is read thanks to a recursive function. Starting at the root, the function checks if the node has a left child. If so, the prefix 0 is added to an array of single digit integers, which is collecting each digit of the nodes code word. The root becomes the left child of the initial root, and the function is called again. This continues until a leaf is reached, at which point the function can put together the code

word from the array, and get the node color component.
Each code word is saved in an *unsigned long long int* array, as it can be up to 20 digits long [23]. This space is necessary as a code word can be up to 18 digits. Furthermore, each code is saved in the table as 1's and 2's instead of 0's and 1's, for example 1112212 instead of 0001101. Any code starting with a 0 will be automatically truncated otherwise, resulting in corrupt codes.

A grayscale image pixel has 256 possible values. One way to encode such values would be by attributing a binary number to each one, 0 going to the most prevalent value and 11111111 to the least used one. However, since there is no 1 bit data type, these values will have to be written one after the other into a binary file. At this point, there will be no way to distinguish two adjacent values.
Example:

Code a: 101

Code b: 1010

Code c: 0

Value in file: 1010, can be b or a-c.

An important property of these code words is that none of them is a prefix to another [22]. That means that by reading the bits 1 by 1, it is possible to determine whether the end of a word is reached or not. There is therefore no need to use a sort of flag, or a way to separate the values. This would have been demanding had maximum 8 bit words been used. Either the length of each pixel value would have to be kept in parallel, using even more memory than the code itself, or a flag value would be inserted between two words, which would double the memory usage. Furthermore, it would be necessary to fill a byte with zeros if the code word is less than 8 bits. Without flags, two code words can be inserted in a single or series of bytes easily.

Gray value	frequency	code
0	120	1
51	10	0001
119	13	011
187	13	010
255	12	001

Table 5.3: Example image properties

Table 5.3 shows the codes of an example image. A random series of these gray values in the compressed file would resemble table 5.4: When reading bit by bit from left to right, the

Bit array:	...	10001101	00010100	00101111	...
Image array:	...	0 - 51 - 0	187 - 255 - 187	51 - 119 - 0 - 0	...

Table 5.4: Encoded values in a bit array

different code words are distinct and cannot be confused with one another.

Write compressed file

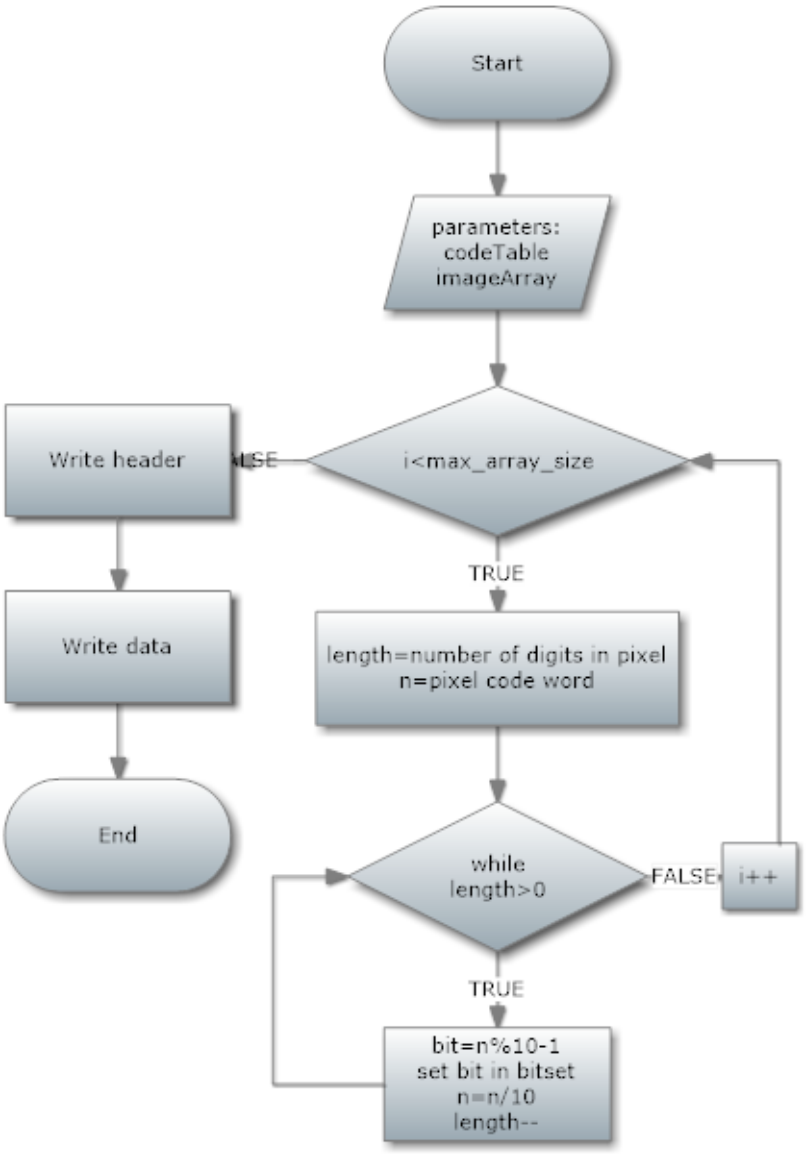

Figure 5.7: Flowchart of the compression function

Figure 5.7 how the compression is done. First, a header is written to the file, containing the dimensions of the image, the code table for decompression, and the total length in bits of the image data. The total length can be calculated using the following equation:

$$L = \sum_{n=0}^{256} frequency[n] \cdot codelength[n] \tag{5.7}$$

An array of unsigned chars of size equal to L is then allocated and used as a bit-set to host all of the image data. Each bit in the array can be manipulated thanks to the following macros in listing 5.3

```
1 #define BITMASK(b) (1 << ((b) % CHAR_BIT))
2 #define BITSLOT(b) ((b) / CHAR_BIT)
3 #define BITNSLOTS(nb) ((nb + CHAR_BIT - 1) / CHAR_BIT)
4 #define BITCLEAR(a, b) ((a)[BITSLOT(b)] &= ~BITMASK(b))
5 #define BITSET(a, b) ((a)[BITSLOT(b)] |= BITMASK(b))
6 #define BITTEST(a, b) ((a)[BITSLOT(b)] & BITMASK(b))
```

Listing 5.3: Bit handling macros

BITSET enables setting any bit to 1, and *BITTEST* returns the value of any bit in a bit-set, the former being used in the compression module, and the latter in the decompression. *CHAR_BIT* is number of bits in the char data type, which is 8 here.
The image array elements are picked treated one by one. For each value, the corresponding code word is first found in the code table. It's lowest digit is isolated, and added to the bit-set as a 0 or 1. The process continues until the array has been fully copied into codewords, and then the file is saved. This way, each code has been entered backwards into the bit array. Therefore, it will have to be read starting from the end by the decompression module. Otherwise, the encoded values will not be recognizable.

5.2.2 Image Decompression Module

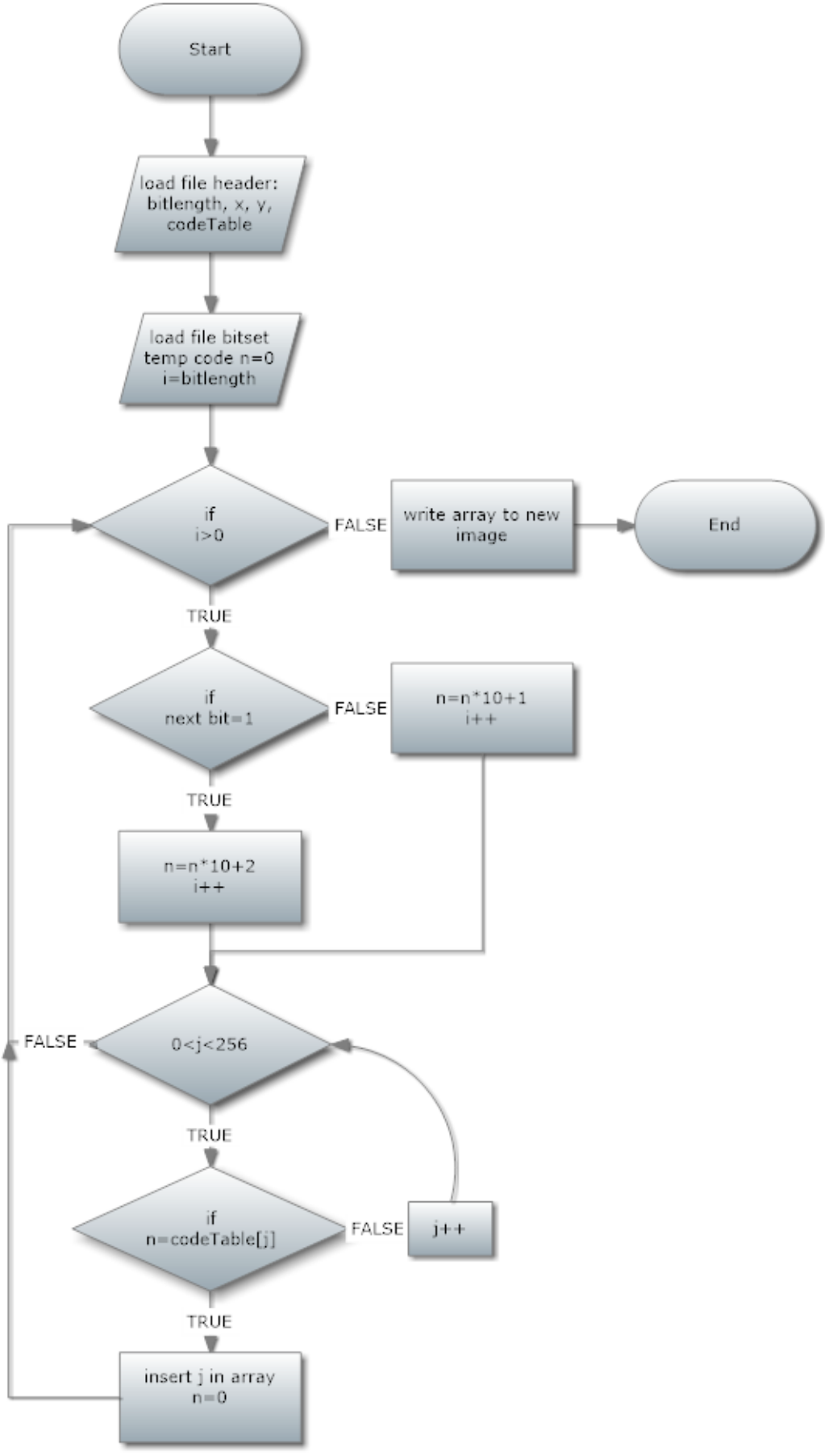

Figure 5.8: Flowchart of the decompression function

As illustrated in figure 5.8 the image decompression is done by first reading the compressed file's header, in order to know the dimensions to allocate a bit-set to copy that of the file. The bits are read one by one, starting from the end. Each bit is added to a temporary codeword which is compared to the values in the code table. 1 bit is added until a match is found, and then the temporary value is set back to 0. The correct gray value is thus found and copied to a new array which will be written to a PGM file to be compared with the original image and sent to the image processing module.

5.2.3 Demonstration

This subsection will show the compression and decompression on a simulated opnav. The code compresses the image, loads the compressed file, decompresses it and then writes it to a new image.

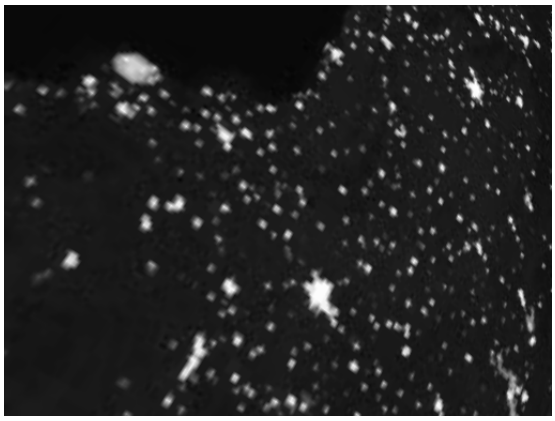

Figure 5.9: Simulated opnav example

Figure 5.9 is a simulated opnav image.

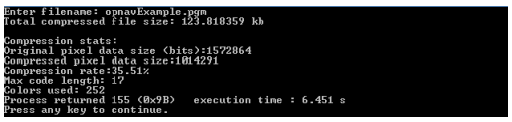

Figure 5.10: Simulated opnav example

Figure 5.10 is the output console displaying compression statistics. The total compressed file size includes the file header containing the code table, as well as as the image array. The pixel data is the number of bits used to code the entire image array. It is the most revealing statistic as it is the data that the compression algorithm is focused on.

Chapter 6

Reference Model

6.1 Introduction

In order to acquire the angles of the satellite (pitch, roll, yaw), when its at a known position, the simulated opnav is compared with a set of reference images until a match is found. The reference images will hold all the necessary information (angles, position, etc..), once a match is found, the information is used to get the desired angles of the satellite at the moment that the simulated opnav was taken.

The most feasible solution to obtain the reference images is to create a 3D model of Earth and simulate the process of an orbiting satellite by using a camera. The camera can be positioned anywhere around the 3D object, and can take pictures of the object (Earth) in order to simulate what is viewed by the satellite. The system should be able to create the reference images dynamically once the satellite requests its attitude (or the three angles).

To get the correct reference images, the satellite's position has to be known in order to simulate it's position. To do that, the satellite has to provide the system with its spherical coordinates (RA-DEC) (see chapter 2). These coordinates are then converted into Cartesian coordinates in order to find the right position for the camera on the 3D model, it will be discussed further in section 6.4.

The position of the satellite is now known, so when it sends a request, the camera will be simulate the satellite's position. The camera now will rotate in the three dimensional space (roll, pitch, and yaw) while facing the object, and takes a picture of the model of Earth at each angle, also it will store the associated information of where the picture of the model was taken. Once the system has the reference images for the desired position of the satellite, the simulated opnav is then compared to the reference model. To compare, Template Matching (explained in section 7.4) is used. If a match is found, the system will return the satellite's attitude at the moment the picture was taken.

6.2 3D Model

3D models represent 3D objects using a collection of points in 3D space, connected by various geometric entities, it allows items that appeared flat to the human eye to be displayed in a form that allows for various dimensions to be represented.[24] These dimensions include width, depth, and height. 3D models have the advantage over 2D images, because they create a more complete picture of the object. It can also graphically simplify complicated concepts

and convey complex inter-relationships, which are difficult to visualise.[25] Concepts and ideas, which cannot easily be represented in words or even through illustrations, can be easily created and viewed from different angles. Not only complicated concepts can be simplified, but 3D modelling can also help in creating an event that is too expensive to create, which is the main reason of its use in this project.

A 3D model of Earth is used in this project in order to simulate the orbiting satellite around it. Having a 3D object representing Earth helps in getting reference pictures that otherwise would've needed a real satellite in space, in the exact altitude and orbit, in order to get the needed reference pictures.

6.2.1 Earth 3D Model

In order to simulate earth's view at night from the satellite, two main things were needed:

- A model of earth
- A texture (2D picture that represents earth at night)
- A uvmap of the 2D picture onto the 3D object

A geosphere shape was chosen to represent earth's model, figure 6.1 shows how it looks before mapping the 2D picture on it.

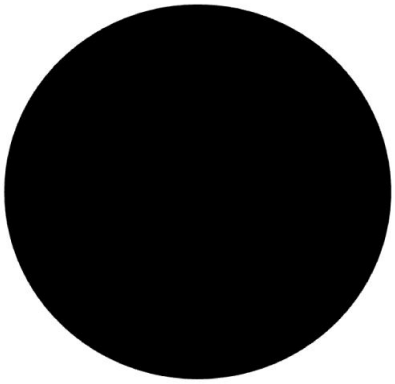

Figure 6.1: The 3D object that represents Earth

A 2D picture 6.2 was chosen to represent Earth's model.

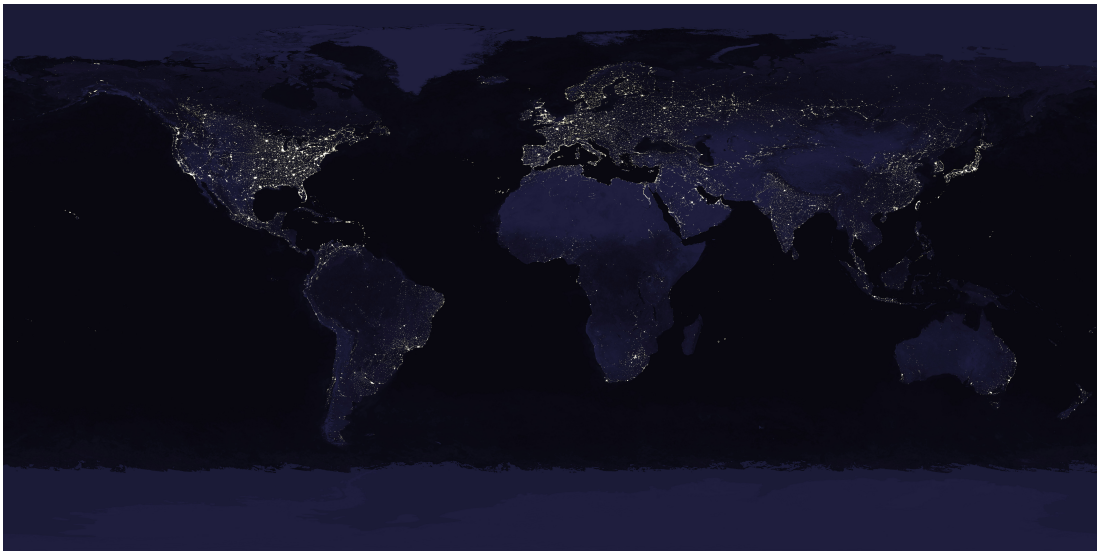

Figure 6.2: The 2D image that represents Earth at night [26]

In order to combine the 2D image in figure 6.2 with the 3D object, UV mapping is used.

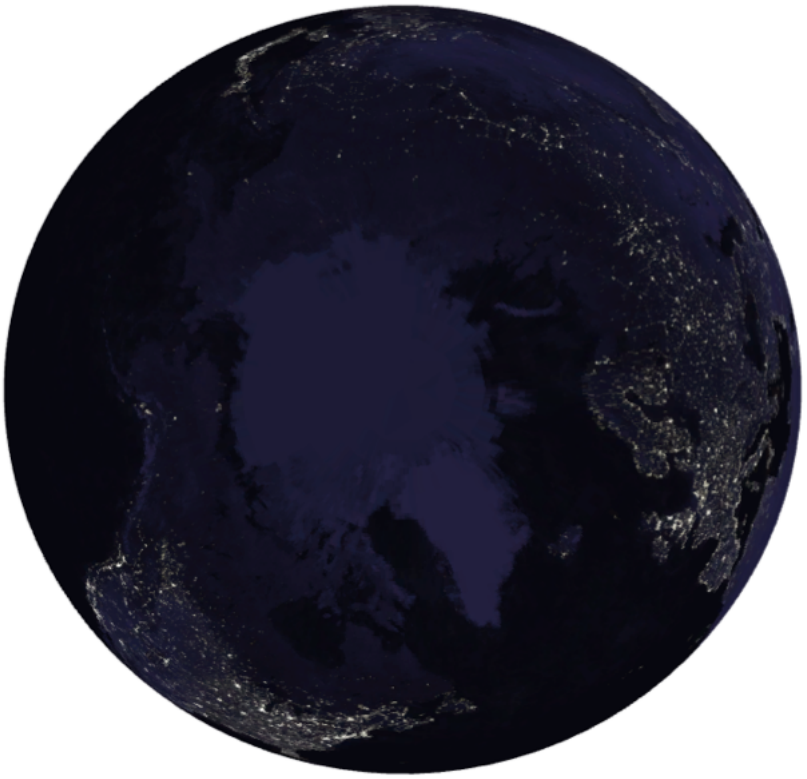

Figure 6.3: The 3D model with the texture

UV Mapping

This process projects a 2D image (texture) onto a 3D object. UVs exist to define a two-dimensional texture coordinate system, called UV texture space. UV texture space uses the letters U and V to indicate the axes in 2D. UV texture space facilitates the placement of image texture maps on a 3D surface. UV is the alternative to XY, it only maps into a texture space rather than into the geometric space of the object.[27]

A 3D object is made up of a polygon mesh, using a 3D modeling software (Cinema 4D, LightWave 3D, etc..), and these polygons are painted with colors in order to depict the uv-map (2D image). It is called a uv-map, but it is essentially a normal 2D image. The uv-mapping process involves assigning pixels in the image to surface mappings on the polygon, usually done by copying a triangle shaped piece of the image map and pasting it onto a triangle on the 3D object.[28]

UVs are essential in that they provide the connection between the surface mesh and how the image texture gets mapped onto the surface mesh. That is, UVs act as marker points that control which points (pixels) on the texture map correspond to which points (vertices) on the mesh.

UV mapping

The UV Mapping process at its simplest requires three steps:

- Unwrapping the mesh
- Creating the texture
- Applying the texture

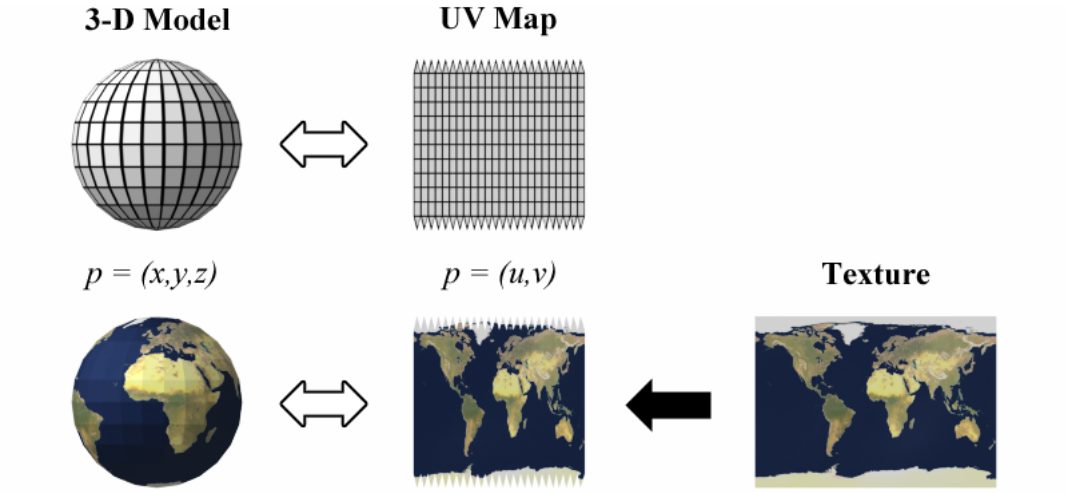

Figure 6.4: Example of the process of UV Mapping [29]

6.3 OpenGL

OpenGL or "Open Graphics Library" is a software interface to graphics hardware. It consists of functions and procedures that allow the developer to produce high-quality graphical images, specifically colour images of 3D objects. To the developer, OpenGL is a set of functions that allow the specification of geometric objects in two or three dimensions.[30]

Developers do not need to license the OpenGL API. If an application developer wants to use the OpenGL API, the developer needs to obtain copies of a linkable OpenGL library for a particular hardware device or machine. Those OpenGL libraries may be bundled with the development and/or run-time options.

6.3.1 Using OpenGL

In order to simulate the orbit of the satellite and what the satellite's camera's view is, a virtual camera from OpenGL is used and placed on specific coordinates while taking pictures.

The first step was creating a 3-Dimensional space and loading the 3D object. When the object is loaded, a uv-map is also accompanied with it, in order to get the full 3D object with the chosen texture.

Loading object and texture

```
78
79          // Load the texture
80
81      GLuint Texture = loadDDS("EarthMap3.dds");
82
83          // Get a handle for our "myTextureSampler" uniform
84      GLuint TextureID  = glGetUniformLocation(programID, "myTextureSampler");
85
86          // Read our .obj file
87          std::vector<glm::vec3> vertices;
88          std::vector<glm::vec2> uvs;
89          std::vector<glm::vec3> normals; // Won't be used at the moment.
90      bool res = loadOBJ("earth.obj", vertices, uvs, normals);
```

Listing 6.1: loading the uv-map and the 3D object from earth.cpp

By observing the code 6.1, loading the texture, (or in other words, uv-map), is done on line 80 by using the function **loadDDS()**. This function has one argument which is the image path of the uv-map, and in this case it has to be **DirectDraw Surface** container file format (with extension DDS) in order for it to work.

The same thing also goes to loading the object, but another function is used on line 90, loadOBJ(). This function has 4 arguments, the first is the path of the object, the other 3 vectors to hold the object's values. The function can read files of type **object file** (or .obj).

6.4 Simulation of satellite view

6.4.1 Roll, Pitch and Yaw

Before explaining how the simulation of the satellite view is processed, a more in depth understanding on how an object rotates is needed. Let's consider an object, defined by a 3D right-handed coordinate system located, for instance, at its center of gravity. The object can rotate independently around the three axis. Most of the time, the rotation angle around the

X-axis is called the roll, around the Z-axis is called the yaw and around the Y-axis is called the pitch. This is summarized in figure 6.5.

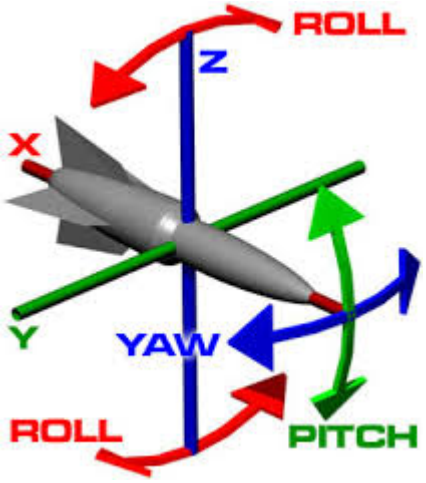

Figure 6.5: scheme showing what are the pitch, roll and yaw [31]

Another way to determine which angle corresponds to which axis is to consider a direction vector, a up vector and their cross product. The direction vector is a vector coming from the center of gravity of the object to the front of the object and thus giving the direction of the object if the object is going straight forward. The up vector is vector going upward from the center of gravity to the top of the object. For instance the up vector of a camera, if the camera is properly held is going straight upward and we get that the up vector is equal to $(0, 0, 1)$ (assuming that the Z-axis is the one going up). Now a 45 degree rotation is applied around the X-axis (a roll) the up vector is still the vector going from the center of gravity to the top of the object but now its coordinates are $(0, 1, 1)$ because of the 45° rotation. [32]. The cross product of the direction vector and the up vector gives the third axis. Concerning the rotations around the axis, the roll is around the direction vector, the yaw is around the up vector and the pitch is around the cross product of the direction vector and the up vector. Figure 6.6 depict this representation of the coordinate system with the pitch, roll, yaw rotations.

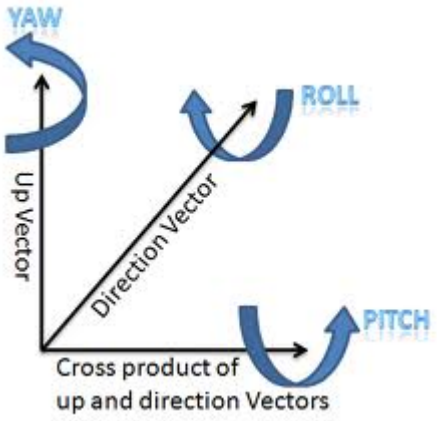

Figure 6.6: another representation of the right handed coordinate system with the axis rotations [33]

In this project, the right-handed coordinate system of the camera has been defined as follow :

- The direction vector is the X axis of the camera
- The up vector is the Y axis of the camera
- The cross product is the Z axis of the camera

6.4.2 Positioning the simulated satellite

In order to simulate the satellite with openGL, the openGL camera is used. This simulated camera must be positioned at the exact same place as the one the satellite would have been. In order to do so, a scaling coefficient and a coordinates conversion must be done.

Scaling factor

The first step is to position the simulated camera at same distance as the satellite would have been. In order to do that, the distance between the center of the 3D model and the camera has to be the scaled distance between the center of Earth and the distance the satellite would orbit once built and launched. In real life, the satellite is suppose to orbit in Low Earth Orbit (LEO) at $600 * 10^3 m$ from the ground, and the Earth has a radius of $6371 * 10^3 m$. The sphere of the 3D model is a 1 meter radius sphere. In order to know the distance of the camera from the center of the 3D model, a scaling factor k is calculated. formula 6.1 gives the relation :

$$k = \frac{R_{3Dmodel}}{R_{Earth}} = \frac{1}{6371 * 10^3} \tag{6.1}$$

where $R_{3Dmodel}$ is the radius of the 3D model and R_{Earth} is the radius of Earth The scaled distance between the center of the 3D model and the camera is thus :

$$d_{scaled} = k * (R_{Earth} + 600 * 10^3) = \frac{6971}{6371} = 1.09 \tag{6.2}$$

Using the result of equation 6.2, the camera is positioned at $r_{3Dmodel} * d_{scaled} = 1.09$ meters from the center of the 3D model.

Unit	circle fraction value	Radian value
hour	$\frac{1}{24}$ circle	$\frac{\pi}{12}$ rad
minute	$\frac{1}{1440}$ circle	$\frac{\pi}{720}$ rad
second	$\frac{1}{86400}$ circle	$\frac{\pi}{43200}$ rad

Table 6.1: table showing the unit conversion between hours, minutes and second and radian

Coordinates conversion

As said in chapter 2, the satellite is using the RA-DEC coordinates system which is spherical coordinate system. Since the satellite is not created yet, this project uses a simulated satellite as depicted in chapter 3. This simulated satellite is behaving exactly as the real satellite would have behaved. To be able to work properly, the implementation needs to have the position of the simulated satellite in Cartesian coordinates system. Before actually doing the conversion from RA-DEC coordinates system to Cartesian coordinates system, the declination and the right ascension need to be expressed in radian. For the declination, the conversion is a classic degree to radian conversion as depicted in equation 6.3

$$\phi[rad] = \phi[deg] * \frac{\pi}{180} \tag{6.3}$$

Then, as explained in chapter 2, the right ascension is an angle expressed in hours, minutes, seconds (hh:mm:ss) and the conversion to radian is given by table 6.1

The next step is the conversion in Cartesian coordinates, to do so, the changing reference system formulas 6.5,6.6 and 6.6 are applied :

$$X = r * \cos(dec - \pi/2) * \sin(ra) \tag{6.4}$$
$$Y = r * sin(dec - \pi/2) \tag{6.5}$$
$$Z = r * \cos(dec - \pi/2) * \cos(ra) \tag{6.6}$$

Where :

- X, Y, Z are the Cartesian coordinates
- r is the distance from the center of the 3D model to the simulated camera
- dec is the declination in radian
- ra is the right ascension in radian

The simulated camera can now be positioned in the 3D model reference and figure 6.7 picture the scaled positioning for a random declination and right ascension.

Figure 6.7: Scaled representation of the position of the simulated camera in the 3D simulation

The simulated camera can be positioned anywhere around the sphere by specifying the declination and right ascension as inputs. On the other hand, the simulated camera cannot move (along an orbit path for instance), it can only rotate.

6.4.3 Simulation of the satellite's angles

The method used to rotate the camera in order to simulate the attitude of the satellite is depicted as follow : Initially, the simulated camera (point S on figure 6.8) is positioned at constant coordinates $(posX, posY and posZ)$ In a Cartesian reference system where the center of the simulated Earth (Coordinates (0,0,0)) is the origin of the referece system. The simulated camera is looking straight toward the center of the simulated Earth, making an initial angle of 45 degrees with the reference plan, this initial angle is called θ_0 and is constant. Then a rotation is applied to the simulated camera, placing it toward the point it should look at. The angle created by the rotation is called θ_r and represent the simulated error. This simulated error is a variable that goes through all the different values of the pitch, if rotating around the z-axis, or the Yaw if rotating around the y-axis. By changing the orientation of the camera, the point where the camera is looking toward is also changed. This point is called Z_r and represents the point where the simulated camera looks directly at after a rotation of θ_r. The necessity to translate a movement in angles to a position on a plan comes from the use of the OpenGL function "lookAt". Indeed, this function is using the point where the camera looks at as one of its parameters. So the movement in angle is translated into the position the camera should look at on the plan. This position is then used by the "lookAt" function and allows the camera to sweep the zone corresponding to the imprecision domain.

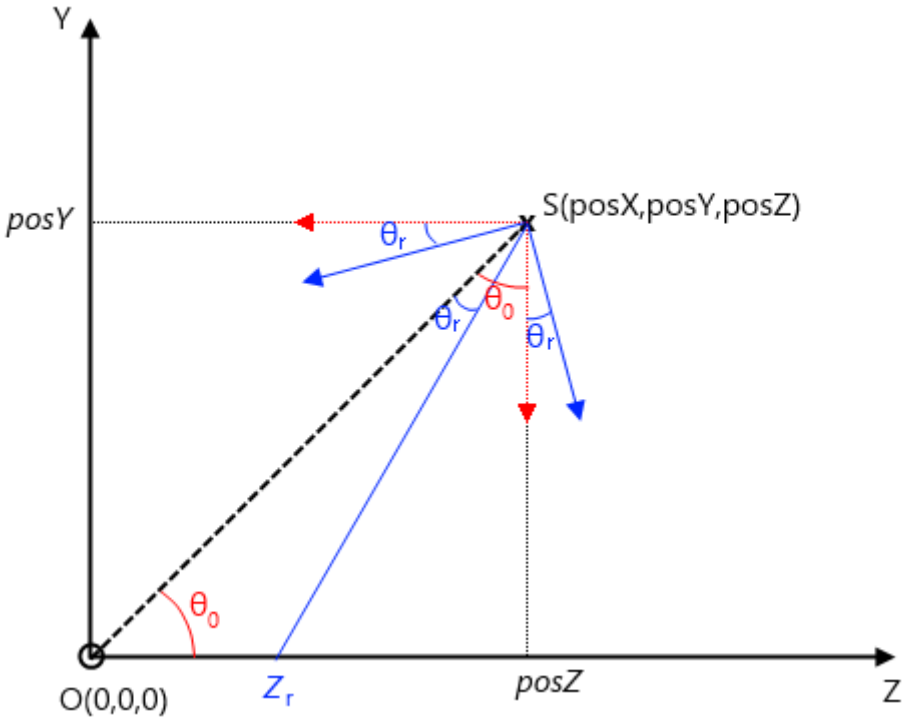

Figure 6.8: scheme depicting the simulation of how to simulate the angles of the satellite. In red is displayed the initial situation and in blue the rotated situation

Using this scheme and trigonometry it is now possible to mathematically determine the position where the camera should face. The following equation 6.7 gives the position of the point the camera is looking toward.

$$z = -posy * \tan(\frac{\pi}{4} - \theta_r) - posz \tag{6.7}$$

The same equation is used to have the y coordinates if the camera is looking toward the Y axis. as shown on equation 6.8

$$y = -posz * \tan(\frac{\pi}{4} - \theta_r) - posy \tag{6.8}$$

The reason the attitude determination is only performed on a plan is because, here, only the yaw and the pitch (respectively y and z in the program) need to be simulated. The roll (along the x axis of the camera) is being taken care of in the template matching module when the image is rotated.

The implementation of the this simulation of the satellite angles is depicted through three flowcharts, which are figures 6.9 ,6.10 and 6.11. The first flowchart, displayed in figure 6.9 shows the loops allowing the simulation to process all the angles of the Pitch-Yaw plan. It also shows in which order are called the two core functions of the simulation that are the determination of the rotation vector and the camera rotation with the generation of the reference images. It is important to note that what is called rotation vector is in fact a translation vector that induces a rotation of the simulated camera but it is called a rotation

vector because here the simulated camera is considered as the main actor of the process and so for more clarity, the phenomenon is described from the camera point of view (which is a rotation on itself).

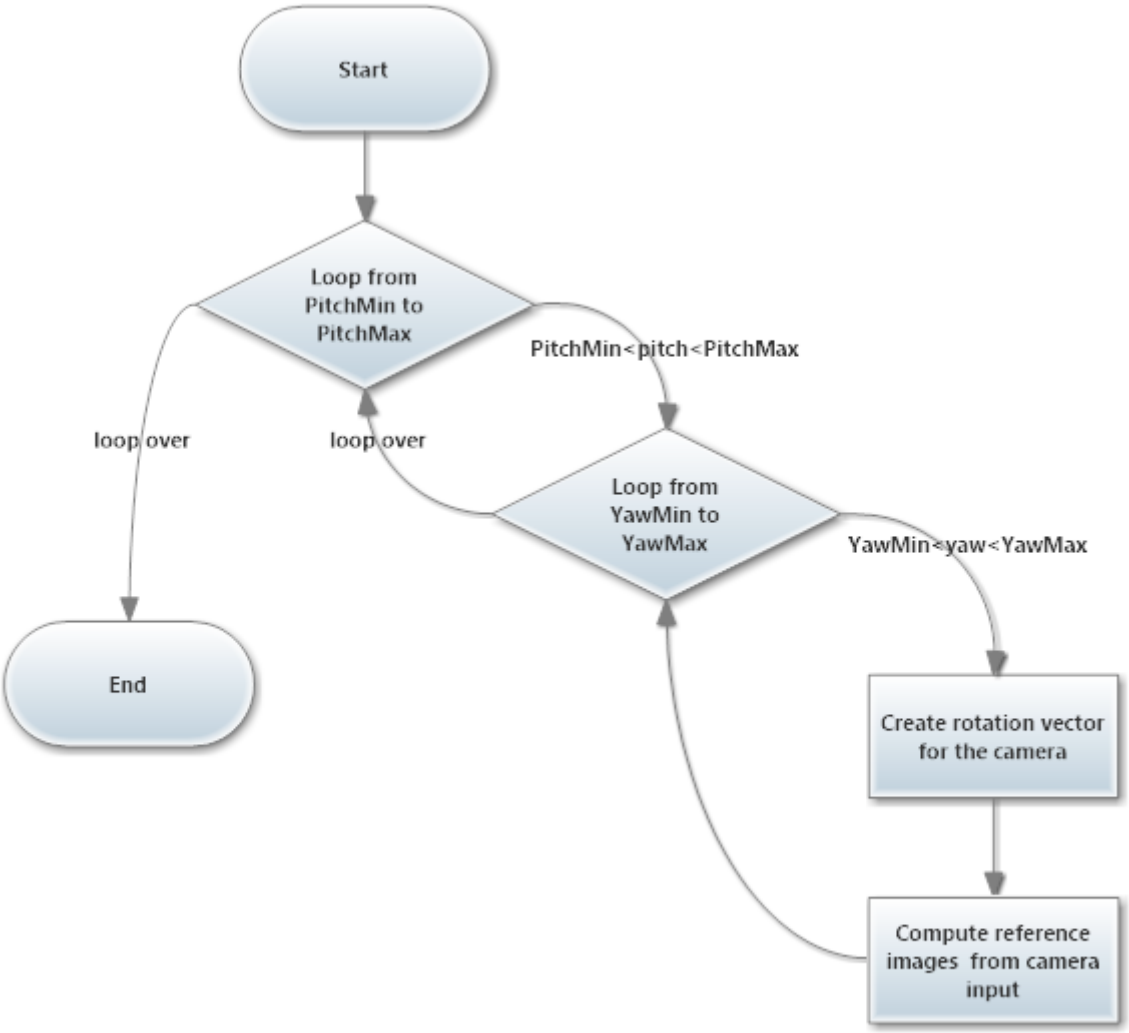

Figure 6.9: Flowchart of the context of the call of the core functions

The next flowchart (figure 6.10) depicts how the camera rotation vector is built, using the equations 6.8 and 6.7. The parameter axis is for the function to know on which axis the equation has to be applied. A new call of the function is necessary to compute on another axis. The other parameter is the wanted error angle, θ_r in figure 6.8. Then the variables are the three coordinates of the vector used to translate the point the satellite should look at. First the angle (θ_r) is converted to radians in order to be used by trigonometric functions. Then a switch and the selected axis is made. If the point the satellite should look at is to

be on the Y axis of the reference system, then the equation 6.8 is applied. If the point the simulated camera should look at is to be on the z axis, then equation 6.7 is applied. When the switch is over, the vector of coordinates (x,y,z) is created. It is important to note that this function only gives one component of the vector. For example if a translation of the point the simulated camera should look at is wanted along both Y and Z axis, the method creating the vector has to be called two times; one time with Y as a parameter and the pitch angle and one time with z as parameter and the yaw angle.

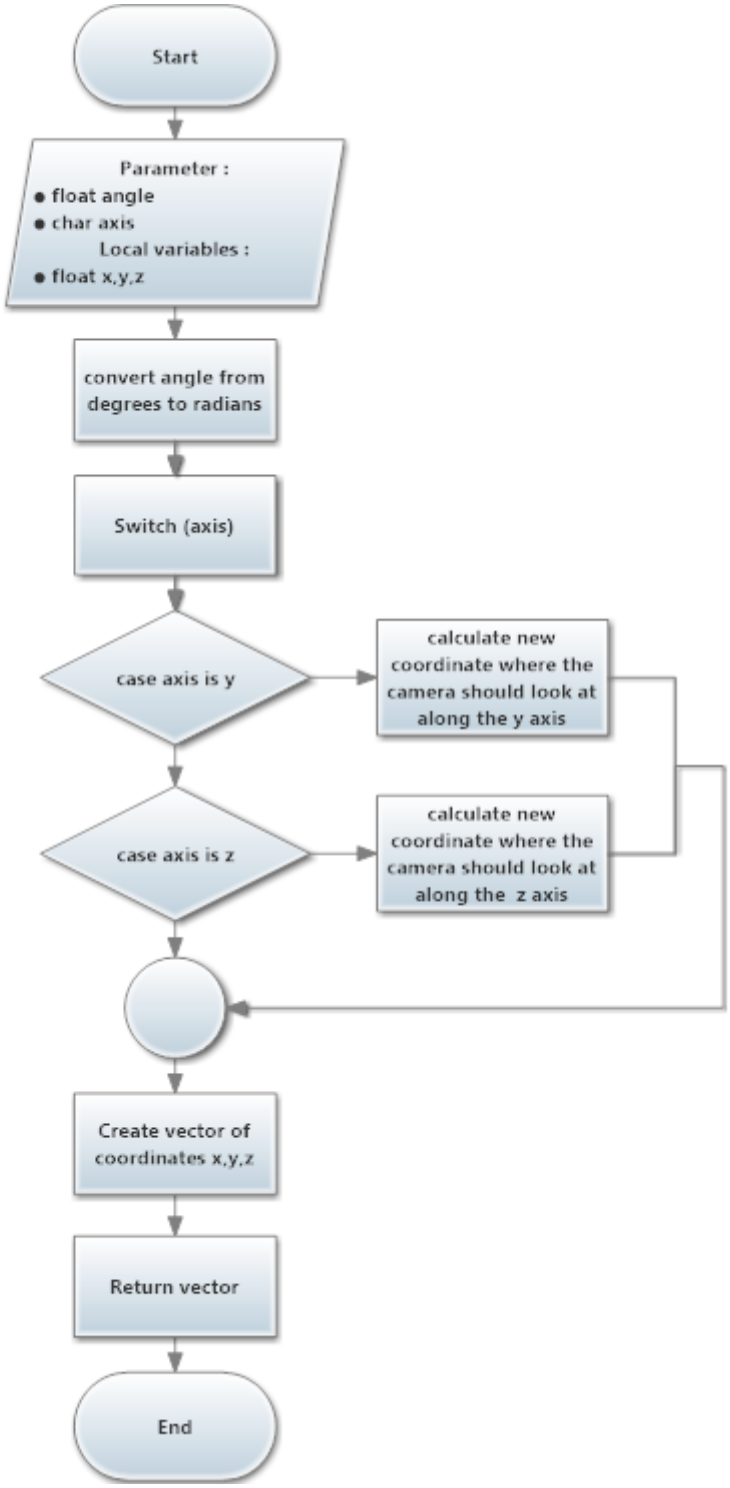

Figure 6.10: Flowchart of the creation of the rotation vector for the camera

The last flowchart, in figure 6.11 shows how the rotation vector obtained from function described in figure 6.10 is used to simulate the rotation of the simulated camera by changing the coordinates of the point it should look at. It is also shown the moment when the reference image corresponding to the processed simulated angle is generated. The rotation of the simulated camera is made by using the glm::lookAt function which come from an extended openGL free library. This function takes three parameters which are three $3x1$ vectors and correspond to:

- The position of the simulated camera
- The point where the simulated camera is looking straight at
- The up vector of the camera

The position of the camera is given by the conversion from RA-DEC to Cartesian explained in subsection 6.4.2. The purpose of the up vector is explained in subsection 6.4.1. And the point where the satellite is looking straight at is as stated in figure 6.11 the vector of the center of the reference system, which is the center of Earth and has for coordinates $(0, 0, 0)$, plus the rotation vectors obtained from the function described in figure 6.10. This addition of three vectors gives the point were the simulated camera should look at. The simulated camera is thus rotating to look in the right direction. Then a reference image is created using a projection matrix on the 3D model.

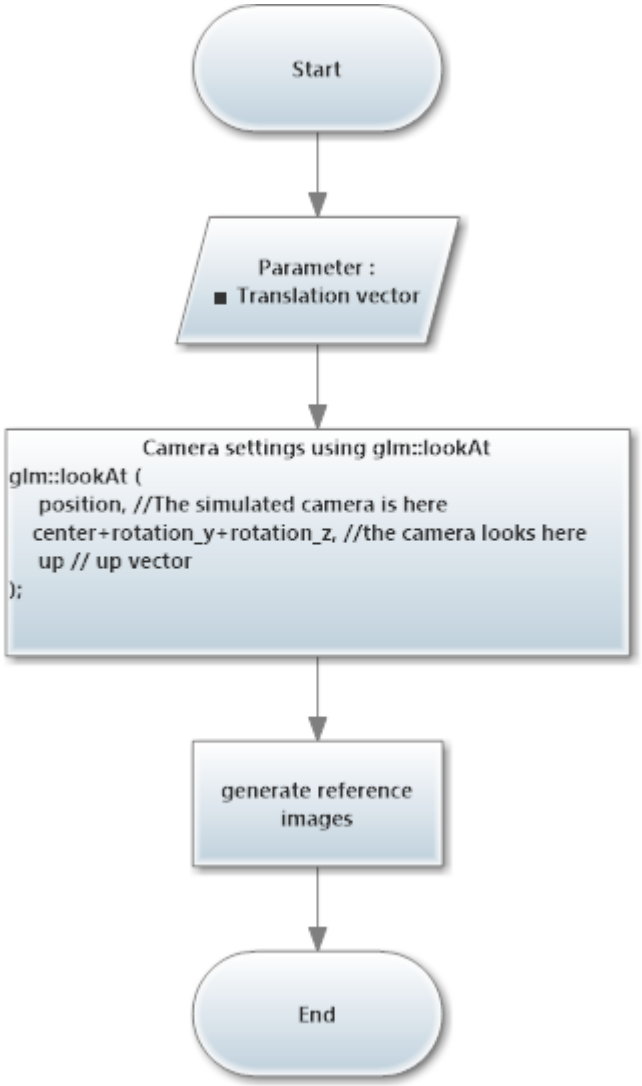

Figure 6.11: Flowchart showing how the simulated rotation is made and the moment the reference images are generated

6.5 Reference Module Demonstration

The reference module now is able to load the 3D model of Earth and create reference images by going through the imprecision domain. This demonstration will show how the reference module works.

The reference module first creates an OpenGL window, as stated in section 6.2.1, and loads the 3D object (Earth.obj) and all its required files (texture and shaders).

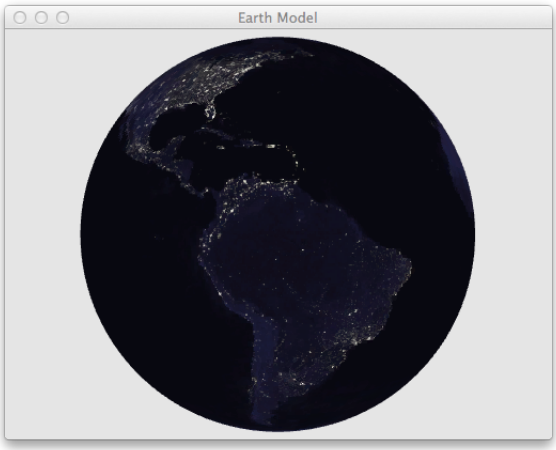

Figure 6.12: The 3D object that represents Earth is loaded

After the model is loaded, the OpenGL camera is then positioned by the coordinates of the simulated satellite on the celestial sphere (RA, DEC) and by the simulated radius as depicted, respectively, in subsections 2.1.2 and section 6.4.

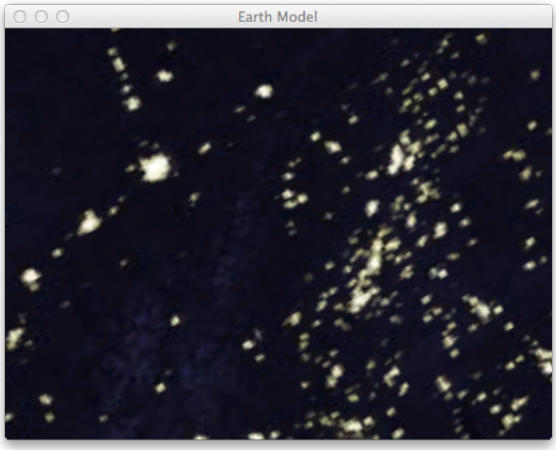

Figure 6.13: Camera is positioned (simulating satellite)

The camera (simulated satellite) is now positioned where it should be, and now it goes through the imprecision domain (see section 6.4) in order to get the reference images. Every time it goes through an angle it will read the pixels of the window (takes a picture of the model), convert it to grayscale and store it in an array of Mat as explained in subsection 7.2.2. Also, the reference module is responsible for simulating the opnav (as the satellite is currently unavailable), so while it is going through the angles, it will choose a random angle to take a picture of and save it as simulated opnav. When the imprecision domain is simulated, the array of reference images is then sent to the image processing module along with the simulated opnav, to find the correct match by comparing them together using template matching as explained in section 7.4.

Figure 6.14: Simulated opnav is taken from the model

This concludes the reference module chapter.

Chapter 7

Image Processing

7.1 Introduction

Image processing is a form of signal processing for which the input is an image (photograph), while the output can be either an image, or a set of characteristics describing the image after processing. Image processing mostly refers to digital images, and is processed as a 2D picture by a computer. In order to process images, they have to be first converted into a digital form. The digital form of an image is a 2D matrix, each element of the matrix contains one or more values, depending on the type of image (RGB, Grayscale, etc...). Each value(s) represents a pixel in the digital image.

Raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement and display, information extraction.

Image processing basically includes the following three steps:

- Importing the image with optical scanner or by digital photography (camera).
- Analysing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

7.2 Image

An image is nothing more than a two-dimensional matrix. It is defined by a mathematical function f(x,y) where x and y are the vertical and horizontal co-ordinates. The value of the function is the pixel at that point of the image.[34]

Since capturing an image from a camera is a physical process. The sunlight is used as a source of energy. A sensor array is used for the acquisition of the image. The amount of light reflected by that object is sensed by the sensors, and a continuous voltage signal is generated by the amount of sensed data. In order to create a digital image, the data is converted into a digital form. The result is a two dimensional array or matrix of numbers (pixel values) which are nothing but a digital image.

7.2.1 Image Representation

An image is represented by a set of pixels, for an RGB (red, green, and blue or coloured) image, each pixel holds three values, each of these values represent a colour. When they're displayed, the RGB colours will mix to generate the required colour. The red, green and blue values of each pixel are used to calculate the intensity of that pixel in order to visually represent the image as a whole to the human eye.

An example of how pixels are arranged:

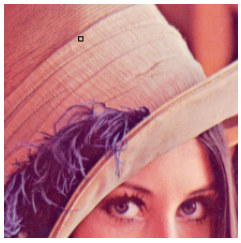

Figure 7.1: This is a normal RGB image, the black square represents a set of pixels

The image above 7.1 is an RGB image, which as stated before, means that each pixel will have three values. The image below will show the pixels inside the square.

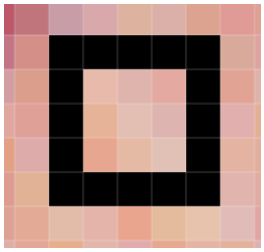

Figure 7.2: Shows a set of pixels

Now lets take a look on how these pixels will be represented for a computer.

$$\begin{bmatrix} 229 \cdot 186 \cdot 169 & 221 \cdot 180 \cdot 171 & 225 \cdot 169 \cdot 158 \\ 228 \cdot 180 \cdot 150 & 225 \cdot 190 \cdot 177 & 220 \cdot 180 \cdot 176 \\ 228 \cdot 167 \cdot 140 & 227 \cdot 186 \cdot 162 & 224 \cdot 193 \cdot 180 \end{bmatrix}$$

As it is seen in matrix 7.2.1, each element in the matrix comprises one pixel. Each value in a pixel is one of the three colours (RGB), so if we take the first element we can see that textbf229 is Red, textbf186 is Green, and textbf169 is Blue. Which if combined will produce a colour that can be shown in figure 7.1. Also, as each of the colour can be represented by only 8 bits, the maximum value of each colour can be 255 2^8.

For this system's purposes, only one value from each pixel is needed in order to compare with another image. To do that, it has been decided that *grayscale images* will be used.

Grayscale images

A grayscale image is an image that displays the intensity of each pixel, and is displayed in shades of grey (black and white). Black is at the weakest intensity (0) and white is at the strongest (255). As stated before, each pixel in an RGB image contains three values, this is not true for grayscale images. Grayscale images have only one value for each pixel, and this value represents a shade of grey.

The picture taken by the camera is RGB, so in order to convert from RGB to Grayscale, an OpenCV (see subsection 7.2.2) function is used *cvtColor()*, this function will take two matrices (source and destination) and the type of the conversion, in this case *CV_BGR2GRAY*[35], this type of conversion is chosen because OpenGL saves the image's pixel values as BGR instead of the normal RGB (see section 6.2.1). In order to convert, the function will take each pixel, it will then use a formula to calculate the new **value**. The new value is comprised of the three "older" values (RGB).

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Figure 7.3: The formula used to convert RGB to Grayscale

The ratio of the colours is (in this case) decided by OpenCV 7.2.2, but basically what it means is that, 0.299 of the red value, 0.587 of the green value, and 0.114 of the blue value is taken and added together to compute the new value (luminance). How the ratio is chosen is different for different types of conversions, but basically the formula used by OpenCV is the Luminance formula as seen in figure 7.3. There are different types of formulas (even luminance has different types), each type has different ratios, but basically the same way of calculating the luminance. Different ratios are used in order to emphasise the physiological aspects, as the human eyeball is most sensitive to green light, less to red and least to blue. [35]

7.2.2 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions that focus on real-time image processing. It is available for Windows, Mac, Android, and iOS. Has C/C++, Java, and Python interfaces. The library allows developers to import, manipulate, process images with ease.

OpenCV has a modular structure. The main modules of OpenCV are listed as follow:

- **Core:** This is the basic module of OpenCV. It includes basic data structures (e.g.- Mat data structure) and basic image processing functions. This module is also extensively used by other modules like highgui, etc.

- **highgui:** This module provides simple user interface capabilities, several image and video codecs, image and video capturing capabilities, manipulating things such as image windows, handling track bars and mouse events. If you want more advanced UI capabilities, you have to use UI frameworks like Qt, WinForms, etc.
- **imgproc:** This module includes basic image processing algorithms including image filtering, image transformations, colour space conversions and etc.
- **ivideo:** This is a video analysis module which includes object tracking algorithms, background subtraction algorithms and etc.
- **objdetect:** This includes object detection and recognition algorithms for standard objects.

[36]

OpenCV usage

In this project, OpenCV was not extensively used, as all the image processing functions were done from scratch. Although OpenCV's *Mat* was used to store the images. Also, another function that was used from OpenCV was *cvtcolor*, cvtcolor is the function responsible from converting RGB (or BGR) to grayscale images. OpenCV was good for testing as it quite easily helps with displaying the images using the function *imshow*, which just takes a Mat data structure as input.

7.3 Noise distortion

The noise, for an image, is unwanted information which deteriorates the image quality[37]. The noise is not part of the original image and is the consequence of processes applied to the image. The noise creates a distortion of the image by changing the value of some pixels. For this project, two types of noise will be tested : Salt and Pepper noise and the Shear distortion. Both of them are part of a sub-module whose purpose is to give noisy images in order to test the robustness of the final implementation.

7.3.1 Salt and pepper noise

The salt and pepper noise is a noise characterized by a certain amount of pixels in an image being white or back (hence the name of the noise).[38]. The salt and pepper noise is usually caused by malfunctioning camera's sensor cells, memory cell failure or synchronization error. Hence, this adding a controlled quantity of this noise to the simulated opnav is a good way to test the robustness of the system if the on-board camera would come to be defective. As mentioned before, only two values are possible for the distorted pixel, those value are called a and b. The probability of each a and b is less than 0.1, t avoid having a dominant noise on the image. For an 8bits/pixel image, the value for pepper noise is close to 0 and the value for salt is close to 255. [37]. The diagram in figure 7.4 shows the probability density function (pdf) described in equation 7.3.1

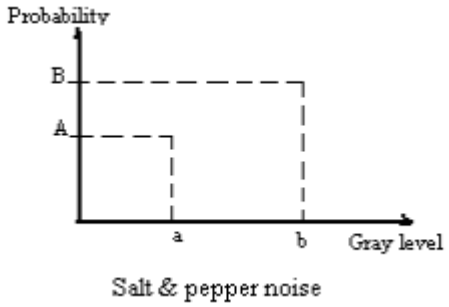

Figure 7.4: Pdf of the salt and pepper noise [37]

$$PDF_{SaltandPepper} = \begin{cases} A & \text{for } p = a\text{"pepper"} \\ B & \text{for } p = b\text{"salt"} \end{cases}$$

In figure 7.5 are represented a noiseless image which represent he simulated opnav and the same simulated opnav with salt and pepper noise distortion added.

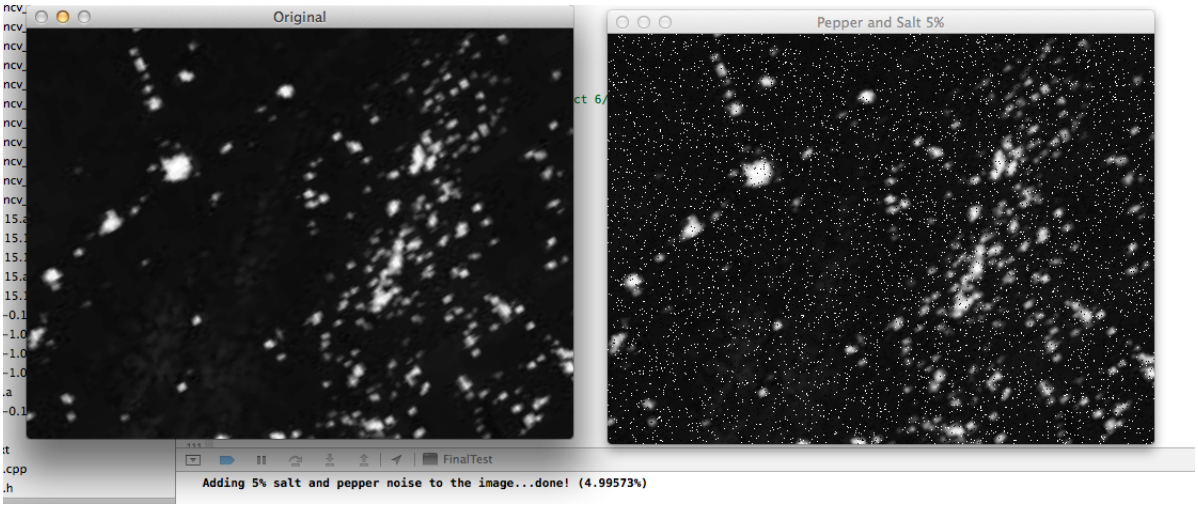

Figure 7.5: Example of a noiseless simulated opnav (left) and the same simulated opnav with a 5% salt and pepper noise

7.3.2 Shear distortion

A shear distortion is a distortion that affect the angles of the object. It is mostly used when discussing oblique projection. [39]. An axis aligned shear provides a shift in one or two axes proportional to the component of a third axis. A z-shear on a square is pictured in figure 7.6

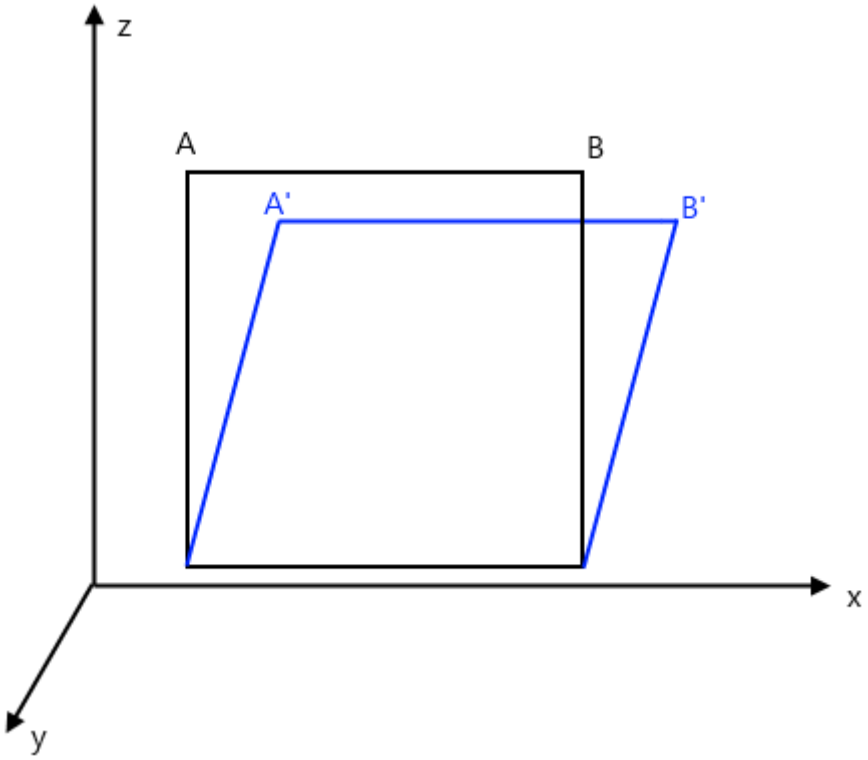

Figure 7.6: z-shear on square. The original square is in black colour and the sheares square is in blue

Two examples of shear distortion are showed on figures 7.8 and 7.9, respectively a x-shear and a y-shear. The original image is the simulated opnav in figure 7.7.

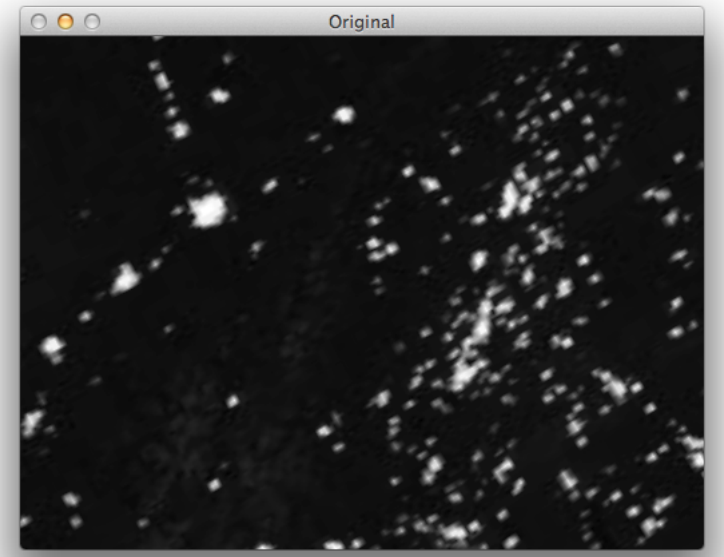

Figure 7.7: original simulated opnav, to be sheared

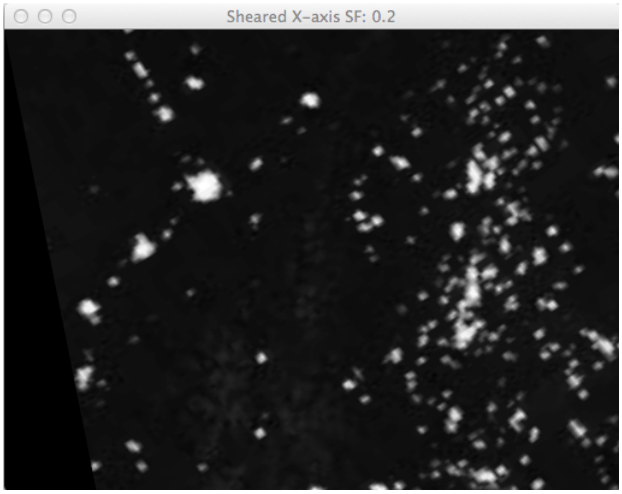

Figure 7.8: x-shear on original simulated opnav

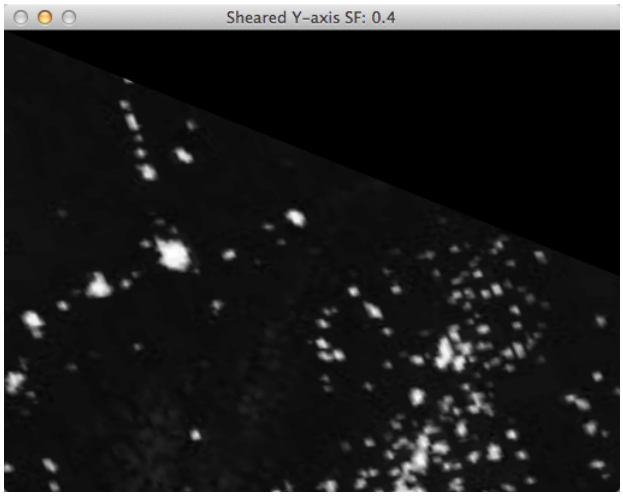

Figure 7.9: y-shear on original simulated opnav

7.4 Template Matching

7.4.1 Introduction

Template Matching is a high-level machine vision technique [40] which is able to identify parts on an image that actually match a predefined template. Many important computer vision tasks can be solved with template matching techniques such as

- Object detection/recognition
- Object comparison
- Content-based image retrieval
- Facial recognition

On the other hand Template Matching itself depends on:

1. Physics (imaging)
2. Probability and statistics
3. Signal processing

There are advanced template matching algorithms available that allow to find occurrences of the template regardless of their orientation and local brightness. Template Matching techniques are straightforward to use in comparison with other methods and also resilient, which in conclusion makes them one of the most popular methods of object localization.

7.4.2 How Template Matching works

In order the use one of the Template Matching techniques [41], a reference image of an object (template image) together with an image to be inspected (input image) must be provided. The next step is to identify all input image locations at which the object from the template image is present. Depending on the case we may choose to identify or not the rotated or scaled occurrences.

Lets take into account a demonstration for example of a naive Template Matching method, which is insufficient for real-life applications, but illustrates the core concept from which the actual Template Matching algorithms stem from. Afterwards the method is enhanced and extended in advanced Grayscale-based Matching and Edge-based Matching routines.

Lets make the search in a straightforward way, we will position the template image 7.10 over the input image 7.11 at every possible location, by computing every time some numeric measure of similarity between the template and the image segment it currently overlaps with.

Figure 7.10: Template Image

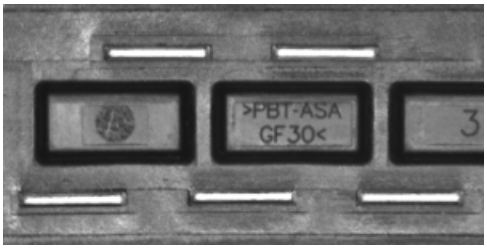

Figure 7.11: Input Image

Lastly the positions that yield the best similarity measures are identified as the probable template occurrences. At this point it must decide which parts of the template correlation image are good enough to be considered actual matches as shown in figure 7.12. Usually at this step the best matches are identified by finding the positions that (simultaneously) represent the template correlation :

- Stronger than some predefined threshold value (i.e stronger that 0.5).
- Locally maximal (stronger than the template correlation in the neighboring pixels).

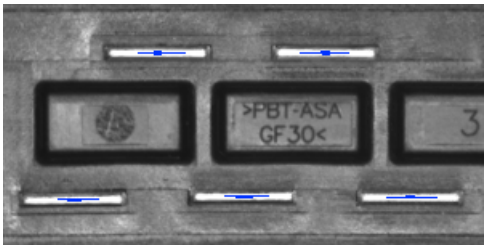

Figure 7.12: Template matching result, areas of template correlation above 0,75. [42]

Templates can either be synthetically generated (model based) or sampled from images as shown in figure 7.10.

In conclusion template matching [41] is a technique used to categorize objects. The goal is to find occurrences of this template in a larger image (input image) that is, you want to find matches of this template in the image. Template matching techniques compare portions of images against one another and compute the similarity between those images at each pixel.

7.4.3 Image Correlation

One of the sub-problems that may come up is finding the similarity of the template image and the overlapped segment of the input image, which is the same as calculating a similarity measure of two images of equal dimensions.

The numeric measure of image similarity is called image correlation. So correlation is a measure of the degree to which two variables agree behaviour. Those two variables represent the pixel values of the two images, template and input image.

The template matching technique, especially in the two dimensional field, as mentioned above has many applications in object tracking, image compression, stereo correspondence, and other computer vision applications. There are several matching methods [43] used as the measure for similarity such as :

- Normalized Cross Correlation (NCC)

$$\frac{\sum_{(i,j)\in W} I_1(i,j).I_2(x+i,y+j)}{\sqrt{\sum_{(i,j)\in W} I_1^2(i,j) \sum_{(i,j)\in W} I_2^2(x+i,y+j)}} \tag{7.1}$$

- Sum of Absolute Differences (SAD)

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i,y+j)| \tag{7.2}$$

- Sum of Square Differences (SSD)

$$\sum_{(i,j)\in W} (I_1(i,j) - I_2(x+i,y+j))^2 \tag{7.3}$$

- Sum of Hamming Distances (SHD)

$$\sum_{(i,j)\in W} I_1(i,j) bitwise XOR I_2(x+i,y+j) \tag{7.4}$$

- Zero-mean Sum of Absolute Differences (ZSAD)

$$\sum_{(i,j)\in W} |I_1(i,j) - \overline{I}_1(i,j) - I_2(x+i,y+j) + \overline{I}_2(x+i,y+j)| \tag{7.5}$$

Sum of Absolute Differences (SAD) is one of the simplest of the similarity measures and works by subtracting pixels within a square neighbourhood between the template image and the input image followed by the aggregation of absolute differences within the square window, and optimization with the winner-take-all (WTA) strategy. Furthermore if the left and right images exactly match, the resultant will be zero.

On the other hand, In Sum of Squared Differences (SSD), the differences are squared and aggregated within a square window before being optimized by WTA strategy. This measure has a higher computational complexity compared to the SAD algorithm as it consists of numerous multiplication operations.

In contrast, Normalized Cross Correlation is even more complex to both SAD and SSD algorithms as it involves several multiplication, division and square root operations.

Lastly, Sum of Hamming Distances is usually used for matching census-transformed images by computing bitwise-XOR of the values in between images, within a square window. Afterwards a bit-counting operation takes place which results in the final Hamming distance score.

In conclusion, in this project the Sum of Absolute Differences (SAD) technique (it will thoroughly explained in section 7.4.4) was used as the measure for similarity since it is the simplest of the similarity measures. Due to its simplicity, it is considered fast and is effectively the simplest similarity measure that takes into account every pixel within a square.
These matching methods [43] among others have been adopted in several applications such as pattern recognition and video compression. In addition, template matching has also been used in various applications, for example, extraction of container identity codes and image segmentation.

7.4.4 Sum of Absolute Differences (SAD)

The sum of absolute differences (SAD) is an algorithm for measuring the similarity between images [44]. The way it works is by taking the absolute difference between each pixel in the original image and the corresponding pixel in the template being used for comparison. These differences are summed to create a simple metric of image similarity. The sum of absolute differences as mentioned above may be used for a variety of purposes, such as object recognition, comparison etc. . .

In order to get a more detailed view of the sum of absolute differences an example will be explained below. In this case an input image is used together with a template image and the representation in pixels of each one of them accordingly is shown in pictures 7.13 and 7.14.

$$\begin{bmatrix} 2 & 6 & 4 & 7 & 7 \\ 1 & 0 & 3 & 2 & 9 \\ 5 & 4 & 8 & 6 & 4 \end{bmatrix}$$

Figure 7.13: Pixel representation of the input image

The sum of absolute differences is used to identify which part of the input image is most similar to the template image. In this example, the template image is 3 by 3 pixels in size, while the input image is 3 by 5 pixels in size. Each pixel is represented by a single integer from 0 to 9 as it can be seen below.

$$\begin{bmatrix} 1 & 3 & 4 \\ 2 & 0 & 5 \\ 5 & 8 & 9 \end{bmatrix}$$

Figure 7.14: Pixel representation of the template image

In this case there are three unique locations within the input image where the template may fit: the left side of the image, the center of the image, and the right side of the image. To find out the SAD values, the absolute value of the difference between each corresponding

$$\begin{bmatrix} 1 & 3 & 0 \\ 1 & 0 & 2 \\ 0 & 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 5 & 1 & 3 \\ 2 & 3 & 3 \\ 1 & 0 & 3 \end{bmatrix} \quad \begin{bmatrix} 3 & 4 & 3 \\ 1 & 2 & 4 \\ 3 & 2 & 5 \end{bmatrix}$$

(a) Left **(b)** Center **(c)** Right

Figure 7.15: Results of all the absolute differences for each pixel

pair of pixels is used: the difference between 2 and 2 is 0, 4 and 1 is 3, 7 and 8 is 1, and so forth.

Calculating the values of all the absolute differences for each pixel, for the three possible template locations, gives the following:

For each of these three absolute differences results, the 9 absolute differences are added together, giving SAD values of 20, 25, and 17, respectively. From these SAD values, it can be seen that the right side of the input image is the most similar to the template image, because it has the lowest sum of absolute differences as compared to the other two locations.

7.4.5 Implementation

The image processing module is responsible for comparing the reference images with the original satellite image, and providing a measure on whether they match or not. If one of the reference images match, then that image's information (angles, position), is returned to the satellite (or this case the user).

Figure 7.16 is the flowchart describing the various steps of the image processing module.

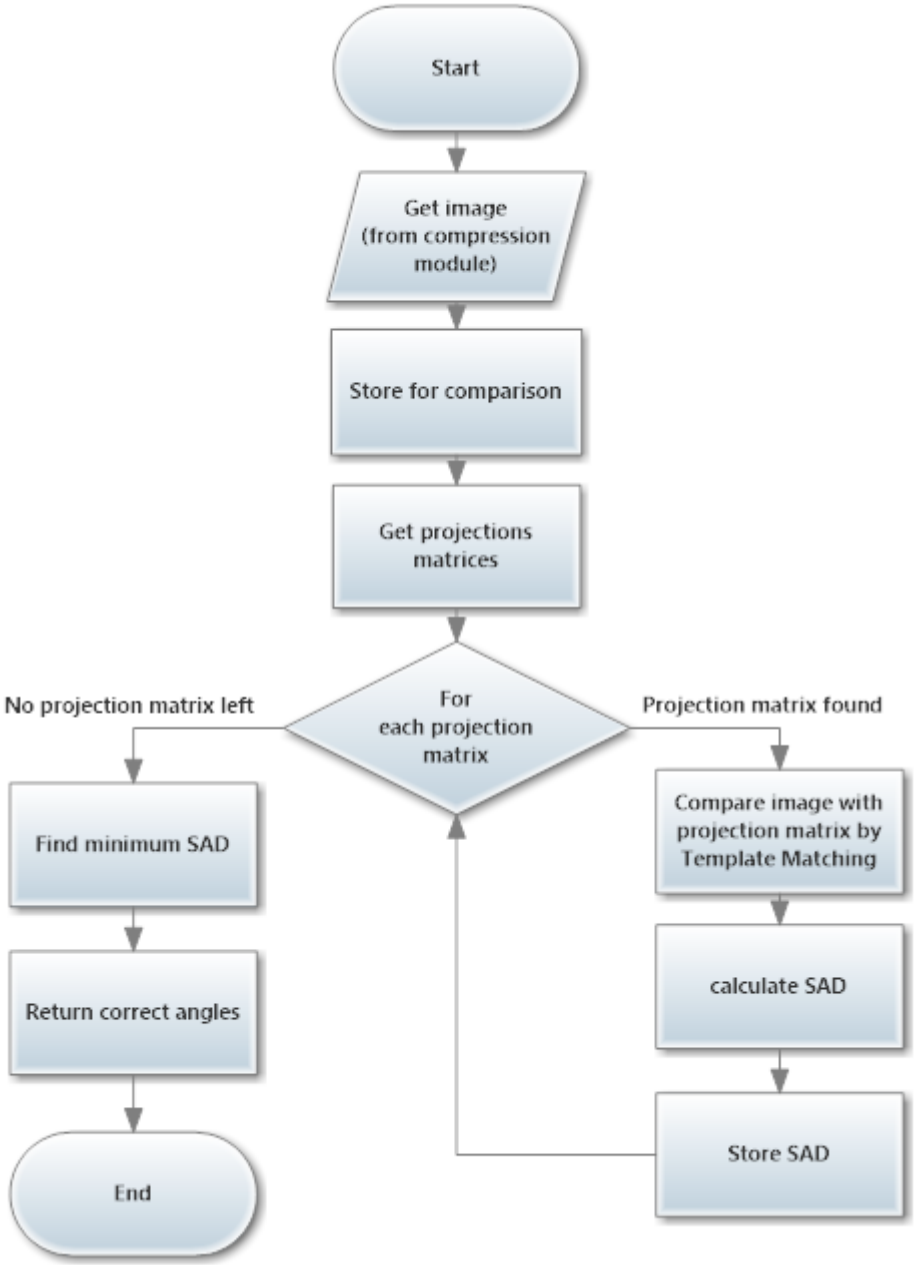

Figure 7.16: Flowchart describing the Image Processing Module

As seen above, the first step of the flowchart is receiving the image from the decompression module, as the satellite compresses the image and then sends it down to the decompression module where it receives and decompresses it. After decompressing, the image is sent to the image processing module where it's stored for comparison.

Now all it needs is something to compare it to, this is where the reference module comes in. The reference module as said in chapter 6, is responsible for simulating the satellite's

orbit as a camera in a 3D space with a model of Earth also loaded as a 3D object. This camera will then take pictures while going through all possible angles (roll, pitch, yaw) while placed in the same position as the satellite was.

After receiving the image, it will then start comparing all the reference images with the original satellite image. The comparison is achieved using template matching 7.4. Template matching will find out how much the SAD (sum of absolute difference) is for every reference image. After going through all the reference images, it will then find the image with the least SAD (usually 0 if 100% match), and returns that reference image's angles to the satellite (or in this case the reference model).

```
21 int TemplateMatch(Mat img, Mat templArr[], int size)
22 {
23
24
25     position p1;
26
27     Point matchLoc;
28     Mat img_display;
29     Mat result;
30
31
32     double VALUE_MAX = 100000;       // SAD max
33     double minSAD = VALUE_MAX;
34     double SAD = 0.0;          // SAD: Sum of absolute difference
35     double imgArrSAD[size];
36
37
38     img.copyTo( img_display );
39     img.copyTo(result);
40
41     for(int k=0;k<size;k++){                  /// goes through templArr
42
43
44         minSAD = VALUE_MAX;
45         SAD = 0.0;
46
47         p1.bestCol = 0;
48         p1.bestRow = 0;
49         p1.bestSAD = 0;
50         p1.changed = false;
51
52         for(int x=0;x<= (img.rows - templArr[k].rows);x++){
53             for(int y=0;y<=(img.cols - templArr[k].cols);y++){
54
55                 SAD = 0.0;
56
57                 for(int j=0;j< templArr[k].cols ; j++){
58                     for(int i=0;i<templArr[k].rows;i++){
59
60                         double pimg = img.at<uchar>(x+i,y+j);
61                         double ptempl = templArr[k].at<uchar>(i,j);
62
63
64                         SAD += abs(pimg-ptempl);
65                     }
66                 }
67
68                 if(minSAD>SAD){
69                     p1.changed = true;
70                     minSAD = SAD;
```

```
71                          //minSAD position
72                          p1.bestRow = x;
73                          p1.bestCol = y;
74                          p1.bestSAD = minSAD;
75                          // std::cout << "Best Row: " << p1.bestRow << "\nBest
                                Column: "<<p1.bestCol<<"\nBest SAD: "<<p1.bestSAD;
76                      }
77                  }
78
79              }
80
81
82              matchLoc.x = p1.bestCol;
83              matchLoc.y = p1.bestRow;
84
85
86              if(p1.changed){
87                  //    imshow( result_window, result );
88                  imgArrSAD[k]= p1.bestSAD;
89
90              }else
91                  imgArrSAD[k]= -1;     // No match found
92
93
94
95          }
96
97          double min = VALUE_MAX;
98          int index = -1;
99          for(int f=0;f<size;f++){
100             if(imgArrSAD[f] <= min && imgArrSAD[f] != -1){
101                 min = imgArrSAD[f];
102                 index = f;
103             }
104         }
105
106
107         return index;
108 }
```

Listing 7.1: Template Matching code using SAD (image.cpp)

Above is shown the implementation of the template matching code used in the Image Processing module for the comparison between the simulated opnav and the reference images.By observing the code in 7.1 the function called TemplateMatch takes as input an image which is the simulated opnav, an array of images called templArr which contains all the reference images from which we want to find a match and an integer called size that which corresponds to the size of the array and is defined by the accuracy and the imprecision domain. In this function a structure was used which contains the position of the match if there is any found. At the same time another variable called $VALUE_MAX$ was needed which is defined to 100000 and represents the maximum SAD acceptable meaning that if an SAD is found bigger than the $VALUE_MAX$ it is not considered as a good match, so as lower the $VALUE_MAX$ is the more accurate the match is. Afterwards there is another array created called imgArrSAD which keeps track of the SAD values for each of the reference images.

Here the template matching takes place using the SAD algorithm (Sum of Absolute Differences). Firstly the code goes through all the rows and columns one by one and calculates the difference of the pixels in every block between the simulated opnav and the reference image and afterwards the absolute values of the differences are found. At these point all the

absolute values are added together and an SAD value is found. Then the SAD value found is compared with the minSAD which is predefined and if the SAD value is smaller than the minSAD, the image which corresponds to that SAD value is considered a match and the position of the match is stored in the structure.
So now the same calculations described above happen for all the reference images remaining in the array templArr and all of the SAD found as mentioned above are stored into the imgArrSAD array.

Finally the last step of this algorithm is to find the best match between the reference images and this is achieved by going through the imgArrSAD array and finding the lowest SAD value which corresponds to the best match found. After the best match is found, the index of that image that holds the lowest SAD value in the imgArrSAD array, is returned and that is because the array that holds the reference image's information (angles) has the same index.

7.5 Template Matching Conclusion

The Image Processing module now is able to receive the simulated OpNav together with the reference images from the Reference module. So now that the Image Processing module has the reference images and the simulated OpNav the next step is to perform the template matching between the simulated OpNav and the reference images. This is achieved by using the SAD algorithm (Sum of Absolute Differences) between the simulated OpNav and each one of the reference images separately.

Finally after the SAD values are found for each one of the reference images the last step is to find the lowest SAD value between them and the reference image that corresponds to the lowest SAD value is considered the best possible match. Now that the reference image with the lowest SAD value is found, the Image Processing module will return that reference image's angles to the satellite (or in this case the reference model).
This concludes the Image Processing module chapter.

Chapter 8

Acceptance Testing

8.1 Image Compression and Decompression Testing

Tests on simulated opnav images were performed to evaluate the performance of the compression and decompression modules. The system requirements, in section3.2, dictate that the compression must be lossless, and that the compressed file must be smaller than the original. To test this, the compression and decompression are done consecutively, and the two images are compared afterwords to see if they are identical.

Test setup and procedure

To perform this test, a simulated opnav is needed. The test is carried out as follows:

1 The simulated opnav is loaded

2 The simulated opnav is compressed

3 The compressed file is loaded

4 The compressed file is decompressed

5 The decompressed image and original image arrays are compared

Test results

Figure 8.1 is the output console of one of the tests.

```
Enter filename: opnav2.pgm
Total compressed file size: 84.614258 kb

Compression stats:
Original pixel data size (bits):1592352
Compressed pixel data size:693135
Compression rate:56.47%
Max code length: 17
Colors used: 250
Decompression Complete
There were 0 different pixels between the original and decompressed file.
Process returned 74 (0x4A)   execution time : 3.735 s
Press any key to continue.
```

Figure 8.1: Output console of a compression and decompression test

	opnav 1	opnav 2	opnav 3
number of colors used	247	250	247
longest code word (bits)	17	17	17
pgm file size (kb)	194	194	194
compressed file size	112	84	108
compression %	42.3	56.7	44.3

Figure 8.2: Compressed image properties

The yielded results of the tests, performed on 3 images, are documented in figure 8.2. The compression was lossless each time, proving that the compression and decompression modules function properly amongst themselves. The compression ratio is also satisfactory, as it ranges between 40% and 60% for the simulated opnavs.

The images all have a different compression %, which can be explained by looking at their histograms:

Figure 8.3: opnav histogram

Figure 8.3 is the histogram of one of the opnavs. It's distribution is similar to a comb distribution, as it has several very high peaks, and the rest of the gray value frequencies are considerably lower. These peaks are what contribute the most to the success of the compression, as the compression algorithm relies on redundancies.

8.2 Reference Model testing

8.2.1 Loading Earth model

Test #1 Setup

The test shows how the model of Earth is loaded using openGL 6.2.1, once the model is loaded, then the camera is positioned correctly using the radius, RA, and DEC 2.1.2. It will first load the 3D object which is a geo-sphere, then load the texture and will uv-map it on the object. 6.1

- A 3D object of shape geo-sphere
- A texture of type DirectDraw (.dds) 6.1
- OpenGL library included

Test #1 Procedure

1. The 3D object is loaded
2. The texture is loaded

Figure 8.4: Test #1 - Earth model is loaded

Test #1 Result

It can be shown from figure 8.4 that the Earth model is correctly loaded with the texture used, in this case Nighlights [26] was used to show how Earth looks like at night.

8.2.2 Showing angles test

Test #2 Setup

This test shows all the angles that the reference model takes reference images at. The imprecision domain is from -5 till 5 degrees in all the angles and the accuracy is 0.5. So the reference images will be taken every 0.5 from -5 till 5. This means that it will have 8000

different reference images that are sent to the image processing module to be compared with the original image.

- A reference model
- A computer running the reference model code which goes through all the angles

Test #2 Procedure

1. The reference model is loaded
2. The reference model will go through all the angles that are in the imprecision domain with the accuracy provided
3. It will store the correct angles along with the reference images
4. prints out all the angles and where they are stored

Index	ROLL	PITCH	YAW
0	0.000000	-5.000000	-5.000000
1	0.000000	-5.000000	-4.500000
2	0.000000	-5.000000	-4.000000
3	0.000000	-5.000000	-3.500000
4	0.000000	-5.000000	-3.000000
5	0.000000	-5.000000	-2.500000
6	0.000000	-5.000000	-2.000000
7	0.000000	-5.000000	-1.500000
8	0.000000	-5.000000	-1.000000
9	0.000000	-5.000000	-0.500000
10	0.000000	-5.000000	0.000000
11	0.000000	-5.000000	0.500000
12	0.000000	-5.000000	1.000000
13	0.000000	-5.000000	1.500000
14	0.000000	-5.000000	2.000000
15	0.000000	-5.000000	2.500000
16	0.000000	-5.000000	3.000000
17	0.000000	-5.000000	3.500000
18	0.000000	-5.000000	4.000000

Figure 8.5: Test #2 - Shows the beginning of the imprecision domain

3403	-4.500000	-1.500000	4.000000
3404	-4.000000	-1.500000	4.000000
3405	-3.500000	-1.500000	4.000000
3406	-3.000000	-1.500000	4.000000
3407	-2.500000	-1.500000	4.000000
3408	-2.000000	-1.500000	4.000000
3409	-1.500000	-1.500000	4.000000
3410	-1.000000	-1.500000	4.000000
3411	-0.500000	-1.500000	4.000000
3412	0.500000	-1.500000	4.000000
3413	1.000000	-1.500000	4.000000
3414	1.500000	-1.500000	4.000000
3415	2.000000	-1.500000	4.000000
3416	2.500000	-1.500000	4.000000
3417	3.000000	-1.500000	4.000000
3418	3.500000	-1.500000	4.000000
3419	4.000000	-1.500000	4.000000
3420	4.500000	-1.500000	4.000000
3421	-5.000000	-1.500000	4.500000

Figure 8.6: Test #2 - Shows how the imprecision domain goes on

7980	4.500000	4.500000	4.000000
7981	-5.000000	4.500000	4.500000
7982	-4.500000	4.500000	4.500000
7983	-4.000000	4.500000	4.500000
7984	-3.500000	4.500000	4.500000
7985	-3.000000	4.500000	4.500000
7986	-2.500000	4.500000	4.500000
7987	-2.000000	4.500000	4.500000
7988	-1.500000	4.500000	4.500000
7989	-1.000000	4.500000	4.500000
7990	-0.500000	4.500000	4.500000
7991	0.500000	4.500000	4.500000
7992	1.000000	4.500000	4.500000
7993	1.500000	4.500000	4.500000
7994	2.000000	4.500000	4.500000
7995	2.500000	4.500000	4.500000
7996	3.000000	4.500000	4.500000
7997	3.500000	4.500000	4.500000
7998	4.000000	4.500000	4.500000
7999	4.500000	4.500000	4.500000

Figure 8.7: Test #2 - Shows where the imprecision domain ends

Test #2 Result

As seen in the figures 8.5, 8.6, and 8.7 it can be shown that the reference model goes through all the angles in the imprecision domain with the correct accuracy which 0.5. It can also be shown that there are 8000 reference images, this is because there are 20 angles between -5 to 5, and there are three angles needed, so 20 * 20 * 20 is equal to 8000.

8.2.3 Getting reference image and converting to grayscale test

Test #3 Setup

This test is about the Reference Model module which goes through all the angles (roll, pitch ,yaw) and afterwards the reference images of the angles are generated, they are also converted to grayscale before being sent to the image processing module.

The following things are needed to carry out the test of the Reference Model module requirements 1.c.

- A reference model.
- A computer running the reference model code which goes through all the angles and finds the reference images.

Test #3 Procedure

1. The reference model is loaded.
2. A search is performed by the reference model module in order to find the correct angles (roll, pitch, yaw).
3. The reference images of the angles are converted into grayscale and then generated.

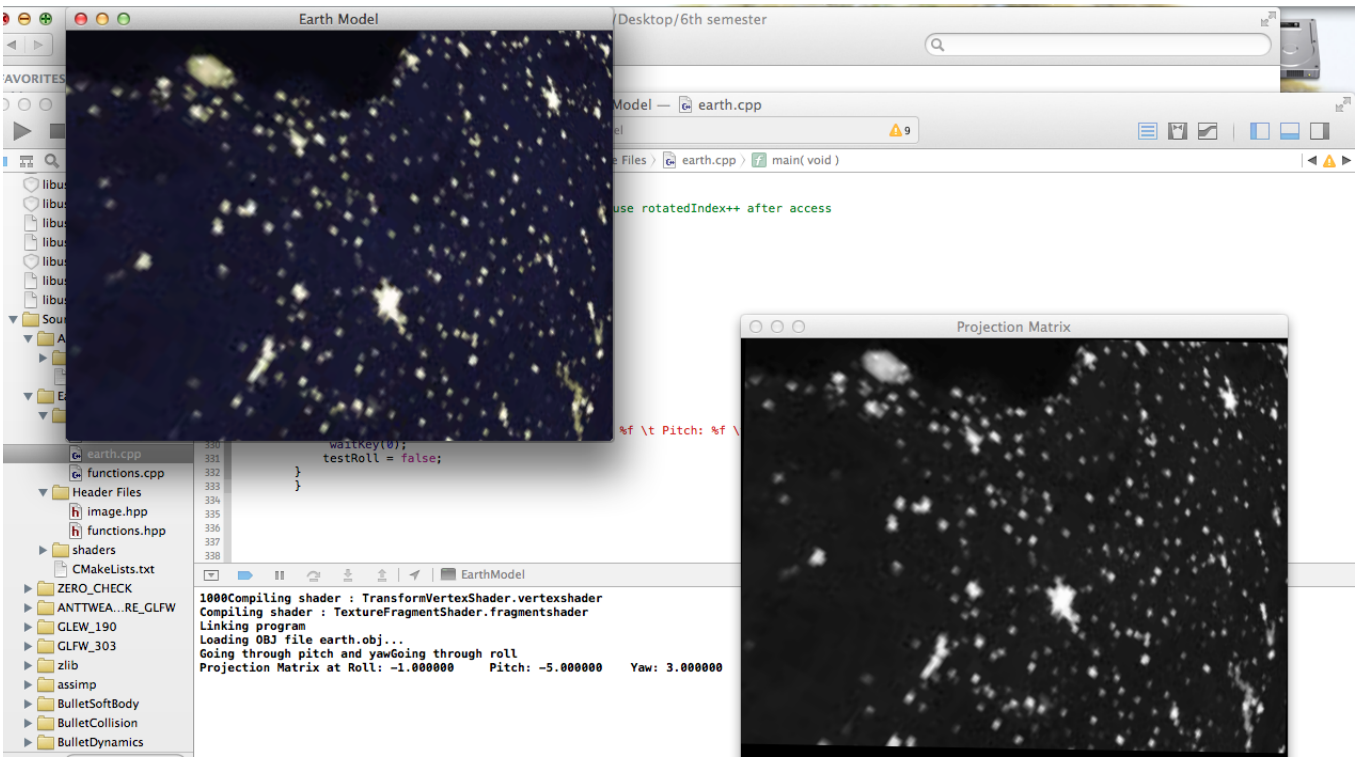

Figure 8.8: Test #3 - Projection Matrix.

Test #3 Result

As mentioned above, the Reference Model module goes through all the angles and for each angle a reference image is generated. Afterwards the reference images are converted into grayscale and then sent to the Image processing module. As seen in picture 8.8 in the top left corner, the reference model is loaded, and on the other side there is a reference image that for this test is generated at -1, -5 ,3 (roll, pitch and yaw respectively).

8.2.4 Imprecision domain test

Test #4 Setup

This test is about simulating the imprecision domain and how the implementation of the project is going through the different points of the imprecision domain by changing the angle of the simulated camera. This test is to fulfil the requirement saying that the program must be able to simulate the imprecision domain of the satellite and to verify if the formulas stated in subsection 6.4.3.

Test #4 Procedure

To be sure the formulas set in subsection 6.4.3 are covering the imprecision domain, a *matlab* script reproducing what's happening in the code is written. Figure 8.9 shows the script. The double for loops simulate the different angles of the imprecision domain (± 5 degrees). The designed result is a rectangle of points where each point shows he location of where the simulated camera is looking straight at. The distance between two points should also always be the same. The position of the simulated camera on the y axis and the z axis is randomly chosen because the obtained result should be the same regardless the position of the simulated camera. If a rectangle verifying what was stated just above is obtained, then the formulas are correct and accurate. If something else than a rectangle is obtained, that means that the formulas are not describing the imprecision domain and another approach needs to be taken.

```
%%%%%%%%%%% Imprecision Domain test %%%%%%%%%
posy = -2;      % Random y and z coordinates
posz = 1;
k = 1;          % Increment for the vectors
% double for loop, processing the Pitch and Yaw
for i=-5:0.5:5
    for j = -5:0.5:5
        theta=i*pi/180; % translation in radian for the pitch
        phi= j*pi/180;  % translation in radian for the yaw
        y(k)=-posy*tan((pi/4)-theta)-posz;  % Obtention of the position
        z(k)=-posz*tan((pi/4)-phi)-posy;    % the camera should look at

        k=k+1;
    end
end

  plot(y,z,'x')         % Plotting the vectors
```

Figure 8.9: Matlab script of the simulation of the imprecision domain

Test #4 Result

The obtained result is showed in figure 8.10 and the result is a rectangle of point displayed as expected. This test proves that the implementation of the project is correctly simulating

the imprecision domain. That means that the process of going through the different pitches and yaws is accurate.

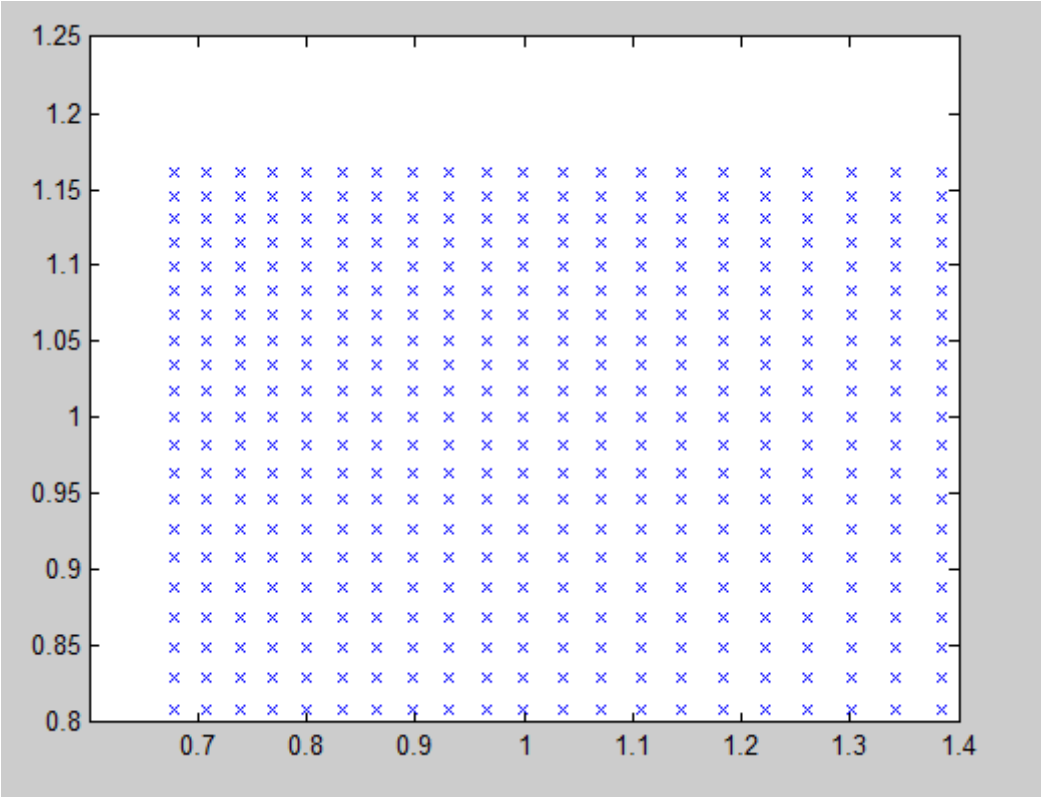

Figure 8.10: result of the simulation of imprecision domain test

8.3 Image Processing module testing

In this section the requirements of the Image Processing module (see 3.2) will be tested. This is to make sure that the Image Processing module fulfills the requirements for the module and to make sure that the module is working correctly.

Here the setup for each test for the Image Processing module requirements will be set up.

There are two individual tests of the Image Processing module. These will be listed below and then when a test setup is added it will reference to the specific requirement. These are all found in ref the requirements.

1. Find a match between two images (using template matching technique, SAD).
2. Find a match from several images (using template matching technique, SAD).
3. Receive reference images the Reference model.

Test #1 setup

This test is setup to compare and find a match between two images. The Image Processing module must be able to find a match between a template and an image using the template matching technique. In this case, the sum of absolute differences (SAD) will be tested, as it was the one that was implemented. The following things are needed to carry out this test from Image Processing module requirements.

- An image.
- A template image.
- A computer running the template matching code using the SAD technique (Sum of Absolute Differences).

Test #1 procedure

1. The input image is loaded.
2. The template image is loaded.
3. Template matching starts to compare each pixel of the two images in order to find a match.
4. When the template matching finishes, it then calculates the SAD.

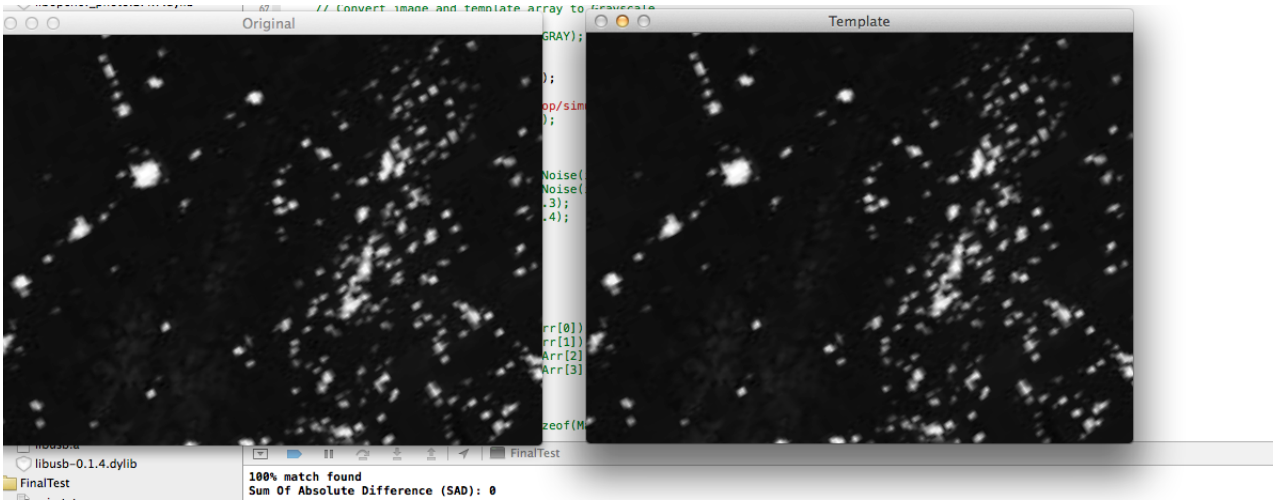

Figure 8.11: Test #1 - Template Matching between two images.

Test #1 result

As seen in figure 8.11 there is the template image and the original image (input image). After template matching with SAD is performed, it returns a match which in this case was 100% and also the Image Processing module returns the value found for the SAD which as it can be seen is 0. By taking into account that the match is 100% and the SAD value is 0 this means that the two images are the same. This test is approved.

Test #2 setup

This test is for finding a match from several images. The Image Processing module must be able to to find a match from several images by using template matching. The sum of absolute differences will be tested as it is the technique used to find a match if there is any. In this test, the case where no match is found will be tested.
The following things are needed to carry out the test of Image Processing module requirements.

- An array of template images (reference images).
- An image
- A computer running the template matching code using the SAD technique (Sum of Absolute Differences).

Test #2 procedure

1. The input image is loaded.
2. The template images are loaded.
3. Template matching starts to compare each template image with the original image, by comparing their pixels.
4. When the template matching finishes, it then calculates the SAD for each of the template images.
5. The image with the least SAD is the best match found.

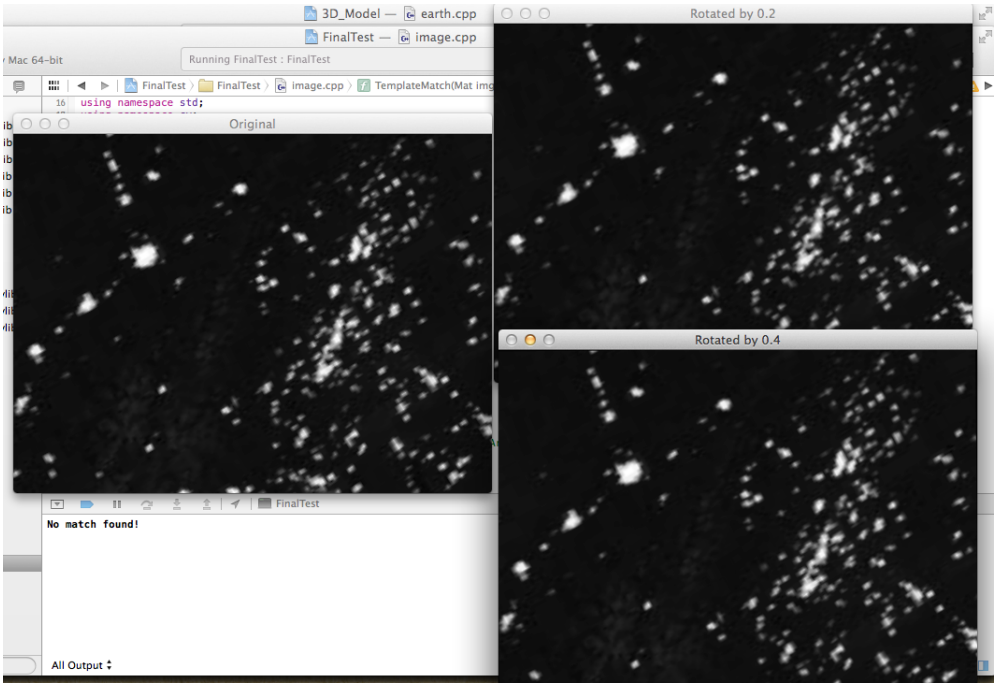

Figure 8.12: Test #2 - Template Matching between several images that are rotated by 0.2

Test #2 result

As seen in figure 8.12 the template images and the original image(input image) are loaded. The templates in this case are rotated by 0.2, 0.4, respectively. The template matching takes place by using SAD (Sum of Absolute Differences) in each of the template images, every time a template image is finished, its SAD is stored. In this test, it returns "no match was found", this is because the images are rotated by 0.2, so the SAD must be larger than maxSAD 7.4.4. As mentioned before, maxSAD is in charge of the accuracy of the match, so with no match found at 0.2 rotation, means that the template matching is accurate enough.

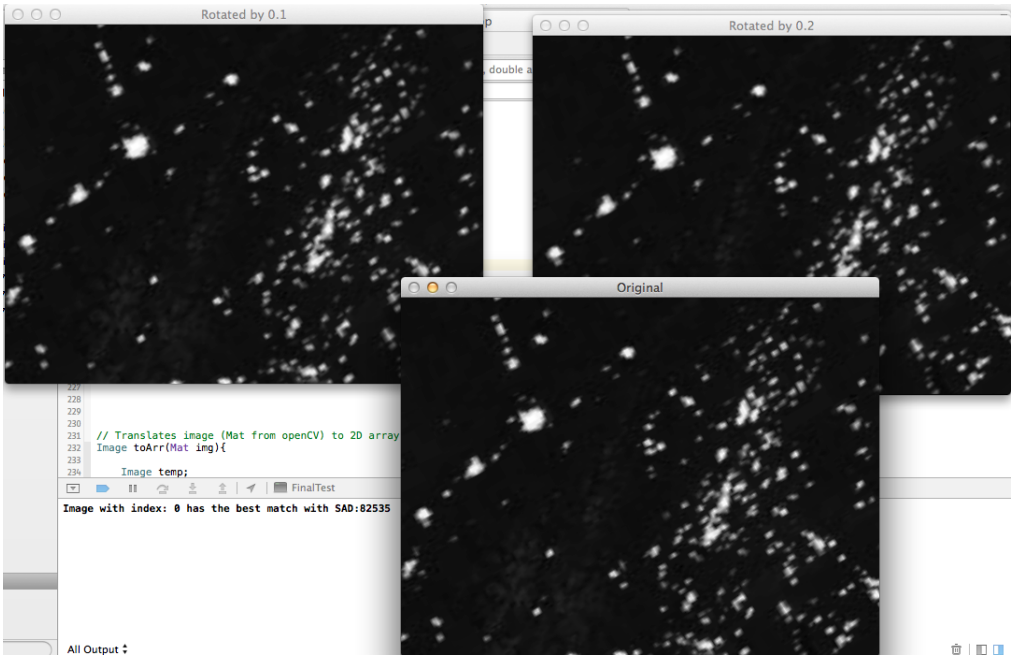

Figure 8.13: Test #2 - Template Matching between several images are rotated by 0.1

In figure 8.13 the templates are rotated by 0.1, 0.2 respectively. The template matching takes place by using SAD (Sum of Absolute Differences) in each of the templates so every time a template image its finished, its SAD is stored. In this case it finds a match on the first template which is rotated by 0.1 since the SAD is less than the maxSAD, but its not a complete match as the SAD is not 0.

8.4 Integrated System Testing

8.4.1 Noise test (Salt and Pepper noise)

Test #1 Setup

This test is set up to compare and find a match between several images by firstly adding noise in the template images(salt and pepper noise). The search image is the simulated OpNav itself and the template images that will be used for comparison, are formed by adding noise to the simulated opnav. This test will show whether the Image Processing module can find a match between a search image and several templates with noise, using the template matching technique (SAD-Sum of Absolute Differences). The following things are needed to carry out this test :

- The simulated opnav.
- The template images with noise added(salt and pepper noise).
- A computer running the template matching code using the SAD technique (Sum of Absolute Differences).

Test #1 Procedure

1. The simulated opnav is loaded.
2. The template images with noise added(salt and pepper noise) are loaded.
3. Template matching starts to compare each pixel of the several images in order to find a match.
4. When the template matching finishes, it then calculates the SAD.

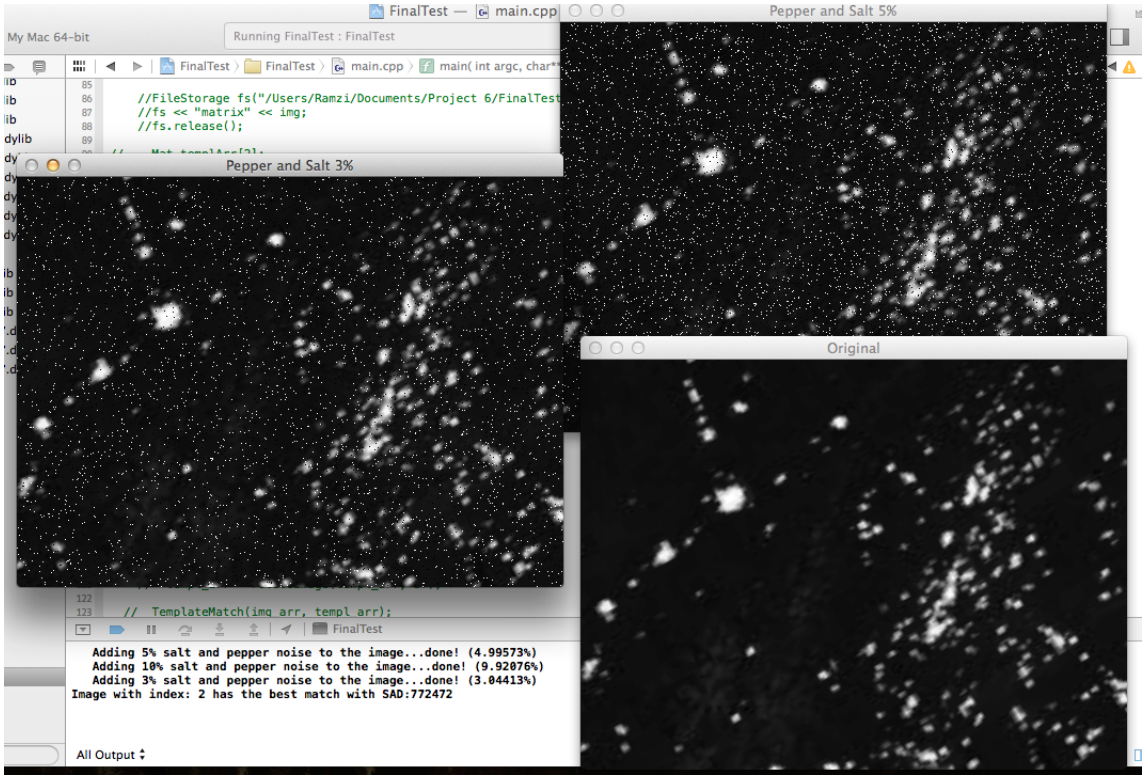

Figure 8.14: Test #1 - Result of Template matching comparing the simulated OpNav to images with noise.

Test #1 Result

It can be shown from figure 8.14 that in the first two templates the noise added is 3% and 5% respectively. After the Sum of Absolute Differences compares each one of the templates to the simulated opnav it finds an SAD value for each one of them. At this point the template with the lowest SAD is chosen as the best possible match and in this test the template with the added noise of 3% is considered as the best possible match.

8.4.2 Applying shear transformation test

Test #2 Setup

This test is setup to compare and find a match between the search image (simulated opnav) and several template images. In the template images is applied the shear transformation by the X-axis and Y-axis respectively. This test will show whether the Image Processing module can find a match between a search image(simulated opnav) and several template images where shear transformation was applied, using the template matching technique (SAD-Sum of Absolute Differences). The following things are needed to carry out this test :

- The simulated opnav.
- Template images where shear transformation was applied by X-axis and Y-axis respectively.

- A computer running the template matching code using the SAD technique (Sum of Absolute Differences).

Test #2 Procedure

1. The simulated opnav is loaded.
2. The template images where shear transformation is performed are loaded.
3. Template matching starts to compare each pixel of the several images in order to find a match.
4. When the template matching finishes, it then calculates the SAD.

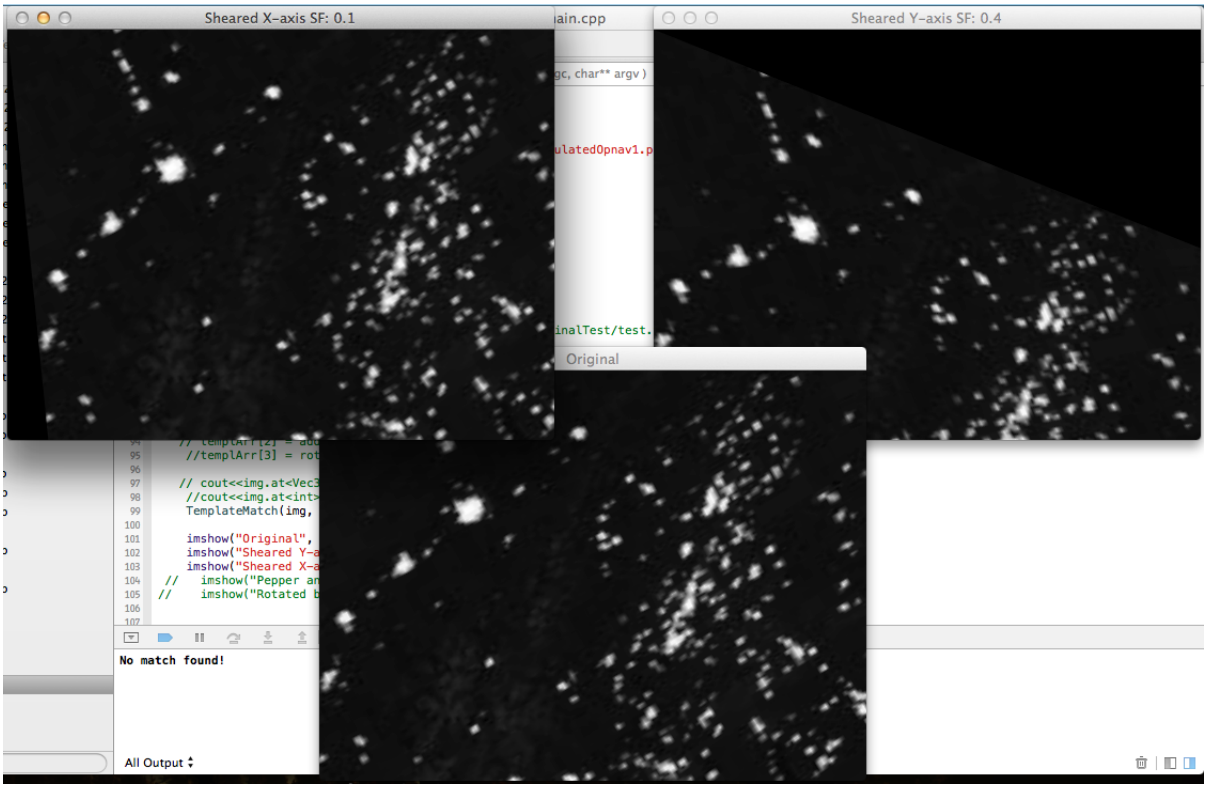

Figure 8.15: Test #1 - Result of Template matching comparing the simulated opnav to template images where shear transformation was performed.

Test #2 Result

It can be shown from figure 8.15 that in the first two templates the shear transformation is performed by the X-axis with a shear factor of 0.1 and the Y-axis with a shear factor of 0.4 respectively. After the Sum of Absolute Differences compares each one of the templates to the simulated opnav it finds an SAD value for each one of them. At this point the template with the lowest SAD is chosen as the best possible match. In this test because of the shear transformation, the Image Processing module was not able to find any match in any of the template images.

Chapter 9

Conclusion

In conclusion, the objectives of the project were achieved, and all the requirements 3.2 were fulfilled. The main objective of the project was to return the attitude of AAUSAT for a known position. The system simulates a model of Earth and the satellite orbiting it in order to get reference images. The satellite was simulated as an OpenGL camera, to collect the reference images.

The reference images were found by, first positioning the camera where the satellite requested its attitude, and then going through all possible angles within the imprecision model with a certain accuracy, taking a picture at each angle. Using the reference images, template matching finds the correct match of the simulated opnav, thus finding the correct attitude of the satellite. Also, to realistically simulate the satellite, some noise distortion was added to the simulated opnav (salt and pepper, shear distortion).

Another objective achieved by this project, was compressing data efficiently from the satellite, and decompressing without losing any information. Compression was used to be able to have an efficient communication link between the spacecraft and the Earth station.

The requirements were tested and fulfilled in chapter 8, they prove the system to be accurate. All the tests were made using a simulated opnav.

Chapter 10

Perspectives

This section explain what further development can be done to the product of the project

The first improvement of the product is to replace the picture of Earth currently used for the 3D model by a more detailed picture of the planet. By doing this, the simulation will give better results if it is used with picture from the satellite. However, those pictures are owned by state or companies and are very expensive.

Then, another improvement can be to improve the robustness of the system by adding noise handling in order to simulate and handle clouds for instance. By doing that, the goal would be to be able to have a successful match using template matching, and then give very accurate and reliable simulation of what could happen in reality. Other source of noise could be for example a spacecraft or a art of a spacecraft passing in front of the camera at the exact moment when the picture is taken, even though the probability is extremely low, it is not zero or an event on Earth making light fluctuations as blackouts, fireworks or a nuclear explosion, for instance.

Currently, the simulation has been developed with the assumption that the satellite would take a picture and then send it to Earth for processing. Another solution can be that everything could be processed on-board, gaining processing time because the compression, sending and decompression of the opnav would be discarded (at least it is not mandatory and can be done after). This gained time gives more accuracy to the returned angles because the satellite wouldn't have time to move as far as it would have if the process was on Earth. In order to have an on-board processing, the image processing code has to be optimized to be fast and to use as little memory as possible. The ultimate goal being making the imaging system a real-time system.

An example of another application for this imaging system using a 3D model can be a security system where the camera can move in a known environment (on a robot for instance). The camera takes pictures, processes them and if the match is good then there is nothing to report, if not, an alert can be triggered to notify the user that something is not normal. In this case, the camera and the processing can use night vision or infra-red vision to give more reliability to the security system.

Bibliography

[1] K F Jensen-K Vinther. Attitude determination and control system for aausat3, 2010. URL http://projekter.aau.dk/projekter/files/32312420/thesis_10gr1035.pdf.

[2] AAUSAT team. Aausat main page, 2014. URL http://www.space.aau.dk/aausat3/.

[3] AAUSAT team. Aausat main page, 2013. URL http://www.space.aau.dk/aausat3/index.php?n=Main.AAUSAT3MissionDescription.

[4] Olivier L. de Weck. Attitude determination and control (adcs), 2001. URL http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-851-satellite-engineering-fall-2003/lecture-notes/l9acs.pdf.

[5] jpl NASA. Imaging system, 2013. URL http://www2.jpl.nasa.gov/basics/bsf12-1.php.

[6] Carol Polanskey. Instrument host information, 1999. URL http://starbase.jpl.nasa.gov/go-a-nims-3-tube-v1.0/go_1117/catalog/insthost.cat.

[7] ESA. Osiris—the optical, spectroscopic and infrared remote imaging system for the rosetta orbiter, 1998. URL http://www.sciencedirect.com/science/article/pii/S0273117797009435.

[8] Tfr000. Meridian on celestial sphere, . URL http://en.wikipedia.org/wiki/FileMeridian_on_celestial_sphere.png.

[9] jpl NASA. Celestial sphere, 2013. URL http://www2.jpl.nasa.gov/basics/bsf2-2.php.

[10] Tfr000. Ra and dec on celestial sphere, . URL http://en.wikipedia.org/wiki/FileRa_and_dec_on_celestial_sphere.png.

[11] J Jankowski-C Sucksdorff. Guide for magnetic measurements and observatory practice, 1996. URL http://iugg.org/IAGA/iaga_pages/pdf/IAGA-Guide-Observatories.pdf.

[12] Terrence Sabaka. Lithospheric magnetic anomalies, 2004. URL http://www.nasa.gov/centers/goddard/images/content/95391main_globe_m.jpg.

[13] Uval dd Dr Anna B. Heller (b) Yonatan Winetraub (a), San Bitan. Attitude determination – advanced sun sensors for pico-satellites. URL http://www.agi.com/downloads/corporate/partners/edu/advancedSunSensorProject.pdf.

[14] EPSON. Gyro sensors - how they work and what's ahead. URL http://www5.epsondevice.com/en/sensing_device/gyroportal/about.html.

[15] ESA. Hubble's deepest view ever of the universe unveils earliest galaxies. URL http://www.spacetelescope.org/news/heic0406/.

[16] . URL http://ac.els-cdn.com/S1047320383710254/1-s2.0-S1047320383710254-main.pdf?_tid=f3a3bad2-aa87-11e3-%b2fe-00000aab0f6b&acdnat=1394698843_4fce4e58ac974654da8de7017a02341c.

[17] . URL http://ee.lamar.edu/gleb/dip/08-2%20-%20Image%20compression.pdf.

[18] Gleb V. Tcheslavski. URL http://www.binaryessence.com/dct/en000083.htm.

[19] Asadollah Shahbahrami. URL http://arxiv.org/ftp/arxiv/papers/1109/1109.0216.pdf.

[20] Arturo Campos. URL http://www.arturocampos.com/ac_arithmetic.html.

[21] . URL http://netpbm.sourceforge.net/doc/pgm.html.

[22] . URL http://www.programminglogic.com/implementing-huffman-coding-in-c/.

[23] . URL http://msdn.microsoft.com/en-us/library/s3f49ktz.aspx.

[24] . URL http://3d.about.com/od/3d-101-The-Basics/a/3d-Defined-What-Is-3d.htm.

[25] . URL http://www.saillabstechnology.com/3d-modeling-is-an-important-process-for-making-a
html.

[26] . URL http://www.nasa.gov/mission_pages/NPP/news/earth-at-night.html.

[27] T Mullen. *Mastering Blender*. Wiley Publishing, Inc, 1st ed edition, 2009.

[28] K.L. Murdock. *3ds Max 2009 Bible*. Wiley Publishing, Inc, 1st ed edition, 2008.

[29] . URL http://en.wikipedia.org/wiki/File:UVMapping.png.

[30] . URL http://www.sgi.com/products/software/opengl/license.html.

[31] Project rho, . URL http://www.projectrho.com/public_html/rocket/controldeck.php.

[32] codeHXR. what exactly is the up vector in opengl function, 2013. URL http://stackoverflow.com/questions/10635947/what-exactly-is-the-up-vector-in-opengls-lookat-function.

[33] mohamanag. Yaw, pitch, and roll, 2010. URL http://axiom3d.net/forums/viewtopic.php?f=1t=831.

[34] . URL http://www.tutorialspoint.com/dip/image_processing_introduction.htm.

[35] . URL http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html.

[36] . URL http://opencv-srf.blogspot.dk/2010/09/what-is-opencv.html.

[37] . URL http://users.utcluj.ro/~rdanescu/PI-L10e.pdf.

[38] . URL www.imm.dtu.dk/~pcha/HNO/ChallF.pdf.

[39] J.M. Van Verth L.M. Bishop. *Essential Mathematics for Games and Interactive Applications A programmer guide*. Morgan Kauffman, 2 edition, 2008.

[40] . URL https://www.adaptive-vision.com/pl/dane_techniczne/dokumentacja/3.2/machine_vision_guide/TemplateMatching.html.

[41] . URL http://www.matdat.life.ku.dk/ia/sessions/session10-4up.pdf.

[42] Adoptive Vision. Template matching, correlation, 2010. URL https://www.adaptive-vision.com/pl/dane_techniczne/dokumentacja/3.2/machine_vision_guide/TemplateMatching.html.

[43] siddhant ahuja. Similarity measure techniques, 2010. URL https://siddhantahuja.wordpress.com/tag/sum-of-absolute-differences-sad/.

[44] Iain Richardson. *Video Coding for Next-generation Multimedia*. Wiley Publishing, Inc, 1 edition, 2003.

[45] Hubi. Illustration of the coriolis force, 2003. URL http://en.wikipedia.org/wiki/File:Corioliskraftanimation.gif.

[46] samwsm1. Coriolis effect, 2008. URL https://www.youtube.com/watch?v=mcPs_OdQOYU.

Appendix A

Appendix A

A.1 The Coriolis effect

This section is an introduction to the Coriolis effect in order to give a better understanding of what was stated in subsection 2.2.3. The Coriolis effect is a deflection of moving objects in a rotating reference. [45] [46]. An example of the Coriolis effect can be given as follow : A ball is thrown from one point to another on a disc. The disc is not rotating and the distance between the two points is the diameter of the disc. The ball is moving in straight line from one point to another as stated in figure A.1. The trajectory of the ball thrown by A to B in figure A.1 is represented by a red arrow.

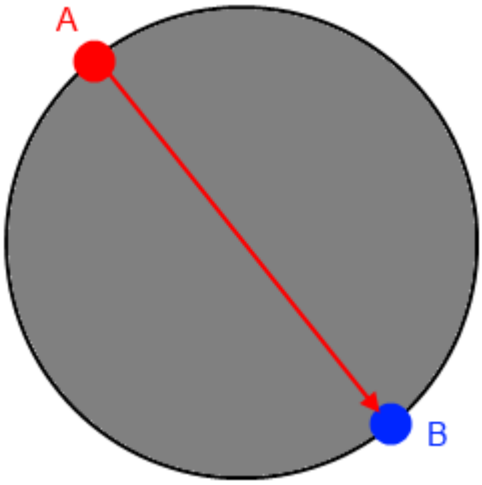

Figure A.1: Initial state of the Coriolis effect example where the ball is throw and the disc doesn't rotate.

Now, the disc rotates counter-clockwise and A throws the ball to B. The ball is still going straight but because of the rotation, the trajectory seems to be a curve going right. If the rotation was clockwise, the trajectory would seem to be a curve to the left. Figure A.2 shows the phenomenon with point A being at the center of the disc and B on the edge.

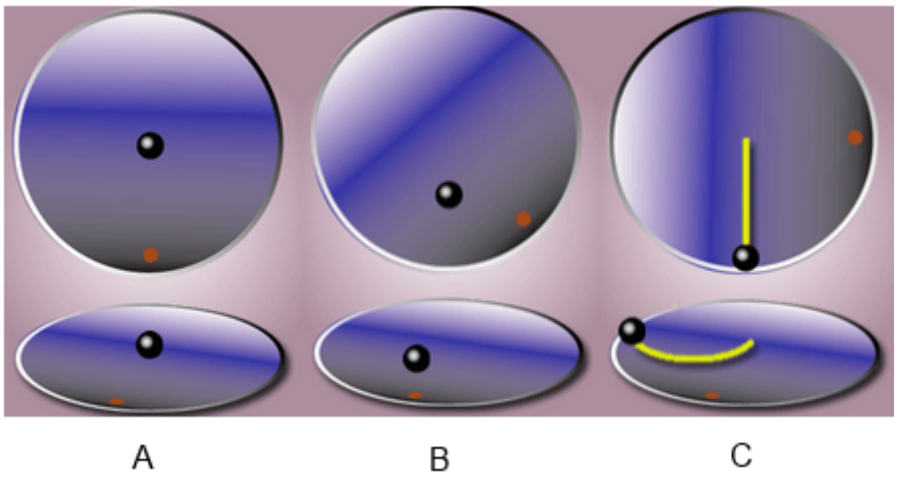

Figure A.2: The experience depicting the Coriolis effect

- A. Before the ball is being thrown, the ball is at the middle of the disc at point A
- B. The ball is thrown as the disc rotates, if the reference is the disc, the ball seem to go straight (circles at the top), if the reference is the ground on which the disc is rotating (circles at the bottom) then the ball describes a curve
- C. The yellow lines show the trajectory of the ball in both references