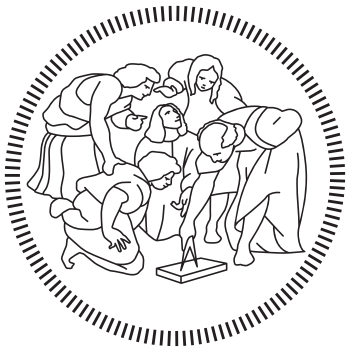
Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Spaziale



Analysis and validation of spaceborne synthetic imagery using a Vision-Based pose initialization algorithm for non-cooperative spacecrafts

Advisor: Prof. Paolo LUNGHI
Co-Advisor: Dr. Eng. Aureliano RIVOLTA

Thesis by:
Francescodario CUZZOCREA Matr. 885016

Academic Year 2019–2020

A chi mi vuole bene...

Acknowledgments

My immense gratitude is owed to Jacopo Guarneri and Ilaria Cannizzaro, for their technical and moreover moral support during this long journey I spent at D-Orbit, working on this project. Without them, without their support and their truely kind words for me, I strongly believe that I would never made it to the end. Thanks to my dad and my brother, who always supported me during these long (10!) years I spent in the University. And also thanks for understanding my absence from home, during the days I spent in the library to study for exams. I think other people wouldn't gave me all the opportunities they gave me during my life. A truely heartfelt tanks goes to Flavio, a precious and invaluable friend. People like you are rare to find, and I am more than sure that if I hadn't met you on my path, I would never have reached this goal.
And also thanks to Stefano and Davide for all the days passed in the library to study between one "accademic break" and another. I think only you two guys have this concept of talking of University stuffs also during a break !
A special thanks goes also to Aureliano, Federico, Mattia, Umberto and Trevis which took care of me during the years passed at Politecnico di Milano, especially when I was in pain because some girl refused to go out with me.
Thanks to Viviana and Andrea, which accompanied me through the bachelor degree, and kindly supported me when I was sad and I was thinking of giving up.
Thanks to Alfonso and Benedetto for all the study done together during the master degree and for the mutual push to keep going trough our very rough exams.
A special thanks is owed to Amro, Anas, Antonio, Emilio, Mayra and all the PoliMI Bovisa library peeps for cheering me up during the writing of this thesis.
Thanks also to professor Paolo Mantegazza for being such an inspiring person for me and for letting me understand what a true engineer is.
Lastly but not leastly, my gratitude is owed to Cecilia and her family who had to live with my frustration and doubts in the past years.

Francescodario Cuzzocrea, March 12, 2021, Busto Arisizio

Abstract

Nowadays...

Sommario

Host Company

This R&D project was developed at D-Orbit. D-Orbit is a New Space company with solutions convering the entire life-cycle of a space mission, including mission analysus and design, engineering, manufacturing, integration, testing, launch, mission control and end of life decommisioning. The company's competitive advantage is in the versatility of its launch and deployement services that can me tailored to the customer's needs, from the launch procurement of a single satellite using standard deployment strategies to the precise deployment of a full constrallation with ION satellite Carrier, a satellite dispenser developed and operated by D-Orbit. ION Satellite Carrier can host any combination of CubeSat with a total volume of up to 48U and release them individually into distinct orbital slots, enabling delplyment schemes previously unavailable to satellite with no indipendent propulsion. Committed to pursuin business modeal that are profitable, friendly for the environment, and socially beneficia, D-Orbit is the first certified B-Corp space company in the world. Headquartered in Como, Italy, D-Orbit has subsidiaries in Lisbon, Portugal, Harwell, UK, and Washington DC, USA.

Contents

Acknowledgments **I**

Abstract **III**

Sommario **V**

Host Company **VII**

List of figures **XI**

List of tables **XIII**

Introduction **3**
Motivation 3
Problem Statement 4
Structure of the Thesis 6

1 State-of-the-art and Limitations **7**
1.1 Synthetic Image Generation for Spaceborne Applications 7
1.1.1 Professional Solutions 7
1.1.2 Low-Cost Solutions 8
1.2 Spaceborne Close-Proximity Relative Navigation 9
1.2.1 Pose estimation sensors 9
1.2.2 Pose estimation tecniques 9

2 Synthetic image generation **11**
2.1 Mathematical Preliminaries 11
2.1.1 Reference Frames 11
2.2 Image generation 13
2.2.1 Ray Tracing 13
2.2.2 Environment Modeling 15
2.2.3 Tango Spacecraft Modeling 23
2.2.4 Camera Modeling 28
2.2.5 Noise modeling 33

2.3 MATLAB integration 34
2.4 Conclusions 34

3 The SVD architecture for pose initialization 37
3.1 Mathematical Preliminaries 37
3.1.1 Pinhole Camera Model 37
3.1.2 Image derivatives 39
3.1.3 The Hough Transform 40
3.1.4 Perspective-n-Point problem 41
3.2 Feature-based pose estimation implementation: the SVD algorithm 41
3.2.1 General Architecture 42
3.2.2 Image processing subsystem 43
3.2.3 Model Matching and Pose Determination 61
3.3 Conclusions 61

4 Results 63

Conclusions 65

A First appendix: Attitude and Orbit Simulator 71
A.1 Keplerian Motion 71
A.2 Dynamics 72
A.2.1 Euler Equations 72
A.3 Direct Cosine Matrix 73
A.3.1 Environmental Disturbances 73

B Second appendix: Random Rotation Matrix Generation 77

List of Figures

1 Schematic representation of the pose estimation problem using a monocular image [38] 5

2.1 Geocentric Inertial Frame (GCI) frame 12
2.2 Raytracing: from the observer to the light source [10] 14
2.3 Sphere manipulation with POV-Ray [20] 15
2.4 Initially chosen texture of the Earth 16
2.5 Comparison of synthetic Earth image with Apollo 11's picture . . 16
2.6 True Color Earth Mercator Image [1] 17
2.7 Landmask Mercator Image 18
2.8 Comparing images with no differential treatment between land and ocean and with differential treatment 18
2.9 Two-color mercator image of Earth's cloud layer 19
2.10 Earth's representation using cloud shell 20
2.11 Earth with the atmosphere layer 21
2.12 Comparison between true and final rendered Earth image 22
2.13 Tango S/C 3-D model 24
2.14 Tango S/C 3-D model in Blender 25
2.15 Add custom POV code into Blender 25
2.16 Texture used to model MLI 26
2.17 Texture used to model solar panels 26
2.18 Tango S/C rendered using POV-Ray Blender add-on 27
2.19 POV-Ray coordinate system 28
2.20 Left Handed Coordinate System and Right Handed Coordinate System 29
2.21 Reference Frame Rotations 30
2.22 Tango S/C rendered using table 2.3 parameters 31
2.23 POV-Ray default perspective camera 32
2.24 Comparison between image without noise and image with speckle and Gaussian white noise 33
2.25 Final result 36

3.1 The pin-hole camera model [39] 38
3.2 Polar representation of a straight line [9] 40

3.3 Schematic representation of the pose estimation problem using a monocular image [38] 42
3.4 The SVD architecture for solving the pose estimation problem [38] 43
3.5 The image processing subsystem [38] 44
3.6 Different steps of the WGE 45
3.7 Result of the ROI detection procedure 46
3.8 Improving the ROI detection procedure 46
3.9 Calculation of a coarse relative position solution using the WGE technique [38] 47
3.10 Merging two truncated edges [38] 50
3.11 ρ and θ diagram 51
3.12 Results obtained applying edge detection and Hough transform on the two streams (1) 52
3.13 Results obtained applying edge detection and Hough transform on the two streams (2) 53
3.14 Case where no merging was required 53
3.15 Results of the merging process (1) 55
3.16 Results of the merging process (2) 56
3.17 Results of the merging process (3) 57
3.18 High-level feature groups [38] 59
3.19 Full CAD model and reduced one 60
3.20 Case of succesful pose initialization 61

List of Tables

2.1 Optical parameters of Earth's different layers 22
2.2 Optical parameters of S/C different parts 28
2.3 Parameters of the camera used to capture the SPEED images [26] 30
2.4 Parameters used to generate the reference orbit [2] 35

List of Symbols

α Aperture angle

λ_{Hough} Maximum gap between two points to be considered on the same line segment

$A.R.$ Aspect Ratio

$\mathbf{A_{CN}}$ Orientation of the camera frame with respect to the inertial frame

$\mathbf{A_{TC}}$ Orientation of target principal axes with respect to the camera frame

$\mathbf{A_{TN}}$ Orientation of the target principal axes with respect to the inertial frame

L_C S/C characteristic length

$L_{min,Hough}$ Expected minimum lenght of a line segment

N_u Number of horizontal pixels

N_v Number of vertical pixels

d_u Horizontal pixel length

d_v Vertical pixel length

f_x Orizontal focal length

f_y Vertical focal length

$\mathbf{t_C}$ Target center of mass location with respect to camera frame

Abbreviated Terms

2-D Two-Dimensional

3-D Three-Dimensional

ADR Active Debris Removal

CCD Charge Couple Device

CDF Cumulative Distribution Function

CG Center of Mass

CV Computer-Vision

DCM Direction Cosine Matrix

EO Electro-Optical

FF Formation Fliying

FLOSS Free, Libre and Open Source Software

GCI Geocentric Inertial Frame

LIDaR Light Detection and Ranging

LoC Lines of Code

NR Netwon-Raphson

OOS On-Orbit Servicing

P-n-P Perspective-n-Point

PDF Probability Distribution Function

POV-Ray The Persistence Of Vision Raytracer

R&D Research and Development

ROI Region of Interest

S&H Sobel and Hough

S/C Spacecraft

SDL Scene Description Language

STL Stereolitographic

SVD Sharma-Ventura-D'Amico

WGE Weak Gradient Eliminator

Introduction

"All models are wrong, but some are useful."

George Box

Motivation

Close range proximity operations between Spacecraft (S/C)s has been studied and discussed by space agencies and private companies since the early stages of space exploration, dating back to the Apollo program [27]. Since then we can find a wide range of missions where close-range proximity operations are in, like Formation Fliying (FF) [3] [5], On-Orbit Servicing (OOS) [35] [44] [40] [17] and Active Debris Removal (ADR) [8] [4]. Most of those missions were possible thanks to the presence of on-board crew or to the cooperativeness between S/Cs. A target space object is deemed cooperative if it is built to provide information suitable for the estimation of its distance and orientation in space with respect to the chaser S/C. Also, it can be be actively or passively cooperative depending on whether it interacts with a dedicated radio-link with the chaser S/C or not [31]. As regard to the new generation of space robotics missions such as debris removal and OOS, proximity operations and docking are key-enabling capabilities for either repair, refuel or deorbit end-of-life and nonfunctional S/Cs [42]. The main challenge when performing close-range navigation in actual OOS and ADR removal missions however is when the target S/C may be uncooperative. This implies that the target S/C may not be equipped with an active communication link or identifiable markers such as light-emitting diodes or corner cube reflectors to help with computing the relative position and attitude of the active S/C (chaser) with respect to a uncooperative target space object [36]. Another important aspect, which comes out when dealing with uncooperative targets, is that debris or operating S/Cs to be serviced may have suffered physical damages as well as optical degradation of their surfaces due to the prolonged exposure to the space environment, thus appearing different than expected [31]. Thus, when operating in close-proximity the attitude and the motion of the target S/C must be estimated in

autonomy by exploiting the sensors available on the servicer S/C. With regards to the technological aspects, Electro-Optical (EO) sensors have been identified as the best option for relative navigation in the foredescribed scenario [31] [33]. Either active Light Detection and Ranging (LIDAR) systems or passive monocular and stereo cameras can be used. The selection of the navigation sensor must consider the resources available on board in terms of mass, electrical and processing power, on one side, the mission scenario and the costs to be sustained for design and development of the satellite system, on the other side [8] [33]. As stated in [37] monocular vision navigation has been identified as an enabling technology for present and future FF and OOS missions (namely PROBA-3 by ESA [7], PRISMA by OHB Sweden [14]). Monocular navigation on such missions relies on finding an estimate of the initial pose of the space resident object with respect to the camera, based on a minimum number of features from a 3D computer model and a single 2D image [37]. In contrast to other state-of-the-art systems based on Light Detection and Ranging (LIDaR) or stero camera sensors, monocular navigation ensures rapid pose determination while offering some advantages such as lower hardware complexity, cost, weight and power consumption, possibility to be simultaneously used for supervised applications and a much larger operational range, not limited by the size of the platform [38] [42] [33]. However, the benefit of lower hardware complexity trades off with increased algorithmic complexity since a monocular sensor cannot provide direct direct three-dimensional (3D) measurements about the target. Moreover, monocular sensors can be less robust to adverse illumination conditions typical of the space environment [43] (e.g., saturation under direct Sun illumination, or absence of light during eclipse) [33]. The increasing challenges of space exploration and moreover the urgent need for debris removal to free slots in orbit and to avoid unwanted collisions is what mainly motivate this work. The capability of being able to develop and test pose determination algorithm in fact will be a key factor for the overall success of new generation missions. Thus, this work is motivated from the need to give to the space community a reliable and extendable Free, Libre and Open Source Software (FLOSS) solution for simulating spaceborne imagery for Computer-Vision (CV) alghoritms needs.

Problem Statement

As will be better explained in the following chapters, in order to verify the goodness of a given image dataset, the most meaningful way it is to apply on it a CV algorithm on it. In this work so, we will validate the generated dataset by implementing the Sharma-Ventura-D'Amico (SVD) monocular pose initialization algorithm. As stated in [12] and [38], the pose initialization problem consist of determining the position of the Center of Mass (CG) $\mathbf{t_C}$ and the orientation of the principal axes $\mathbf{A_{TC}}$ of a non-cooperative target S/C (no markers or other

specific supportive means) with respect to the camera frame **C** from a single Two-Dimensional (2-D) image, given its Three-Dimensional (3-D) geometrical representation (referred as map or model). If we assume that the 3-D model of the target S/C is defined in a body fixed coordinate system **T**, and it is aligned with the target's principal axes with its origin at the CG, the orientation is then defined by the Direction Cosine Matrix (DCM) $\mathbf{A_{TC}}$, which represents the trasformation from the coordinate system of **T** to **C**.

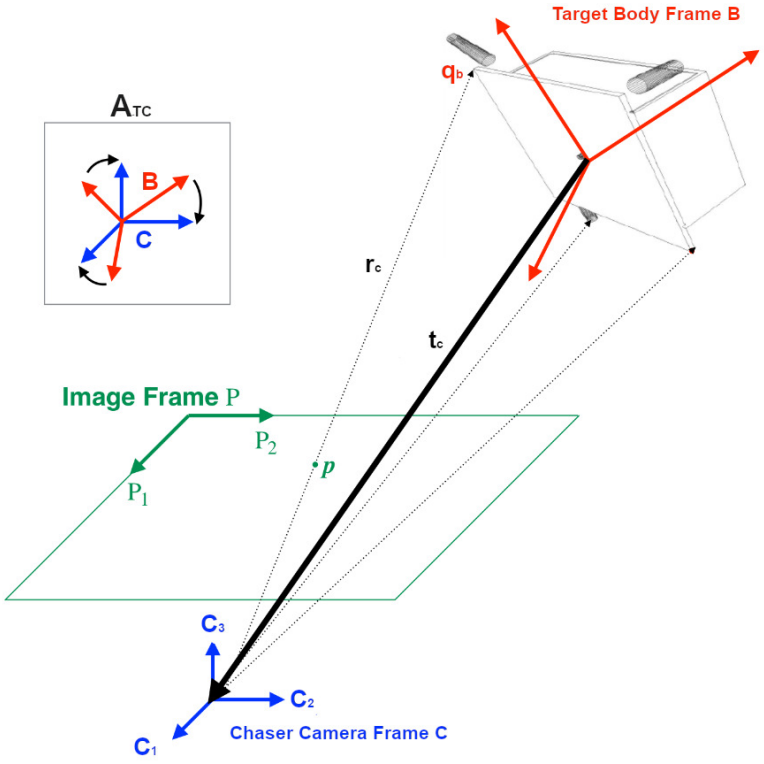


Figure 1: Schematic representation of the pose estimation problem using a monocular image [38]

A generic image point $p = [u, v]^T$ can be related to the corresponding $\mathbf{q_B}$ point of the 3-D model by means of the unknown pose ($\mathbf{t_C}$ $\mathbf{A_{TC}}$) according to the following 2-D - 3-D true perspective equations:

$$\mathbf{r_C} = [x_C \quad y_C \quad z_C]^T = \mathbf{A_{TC}}\ \mathbf{q_B} + \mathbf{t_C}\,, \tag{1}$$

$$\mathbf{p} = \left[\frac{x_C}{z_C} f_x + C_x, \frac{y_C}{z_C} f_y + C_y\right]\,, \tag{2}$$

where f_x and f_y are the respectively the orizontal and the vertical focal lenght and $[C_x, C_y]$ are the coordinates of the center of the image.

As discussed in [12], we can build several considerations on top of those two coupled equations :

- the relationship between image points and pose parameter is hightly non linear and the problem of retrieving the 3-D points from the 2-D image can have infinite solution in under-constrained situations [16];
- the correspondence between $\mathbf{q_B}$ and $\mathbf{p}$ is not knon a priori;
- the image coordinates need to be corrected for pixel's non-quadratism and lens distortion before using the foreametioned equations.

The pose estimation system thus needs the be capable of extracting the client features from the available image (image processing) to obtain measurements, namely $\mathbf{p}$. The extracted features then needs to be matched to the correspondent elements of a client model (model matching), namely $\mathbf{q_B}$. The unknown pose then has to be estimated based on the available measurements (pose estimation). The architecture proposed in [38] solve the pose estimation problem by coupling the Weak Gradient Eliminator (WGE) technique with the Sobel edge detection to perform the image processing, uses feature synthesis to reduce the searh space for model matching and combines a P-n-P solver with the Netwon-Raphson (NR) method for pose estimation.

Structure of the Thesis

Chapter 1

State-of-the-art and Limitations

"We are just an advanced breed of monkeys on a minor planet of a very average star. But we can understand the Universe. That makes us something very special."

Stephen Hawking

1.1 Synthetic Image Generation for Spaceborne Applications

1.1.1 Professional Solutions

ESA PANGU

PANGU is a software developed in order to create synthetic planetary surface images, as much representative as possible, to aid the development of vision-based algorithms.[32]
PANGU is a nice software which is ready to use. It's use is permitted for free for users working on an ESA project, while non-ESA users have to contact STAR-Dundee to purchase PANGU with technical support.
PANGU can also be integrated with proprietary or OSS simulation tools and it gives the possibility to correctly simulate a space camera in all its aspects (so focal lengths, and other relevant parameters of the image). It renders the images using OpenGL and it can use GPU cores to accelerate the rendering.

Airbus Surrender

SurRender is a software developed by Airbus Defense and Space. The software handles various space objects such as planets, asteroids, satellites and spacecraft. It is capable of accommodating solar system-sized scenes without precision loss,

and optimizes the ray tracing process to explicitly target objects. It can operate in real time mode to be coupled with proprietary or OSS simulation tools and it gives the possibility to have an Hardware in The Loop simulation to test the responsiveness of the image processing subsystem. It gives the possibility to correctly simulate a space camera in all its aspects (so focal lengths, and other relevant parameters of the image). It parallelized and so can be ran on cloud platform to accelerate rendering times.

1.1.2 Low-Cost Solutions

SPEED dataset: image generation using OpenGL

URSO dataset: image generation using Unreal Engine 4

POV-Ray

POV-Ray it is an opensource ray-tracing tool. It does not offer a GUI for modeling objects, like Blender, but it can be used as Blender rendering engine to be able to have a 3D modeling environment to model our objects.
POV-Ray has also been used by a different number of people doing research work in the space field to generate images, for example it has been used under the ESA LunarSim study to render images of lunar surfaces. It can also be extended to correctly simulate images of spacecraft. It is a powerful software which let us define surfaces and materials relevant properties such as reflectivity, diffraction, specularity and brillance. Those parameters can be fine tuned to obtain an image as realistic as possible, under certain limits.
POV-Ray can be scripted in order to be used in conjunction with other softwares. Although does not let the user to add some sort of disturbance or noise to the generated images, those disturbances may be added by using some third party software such as MATLAB thanks to POV-Ray's ease of scriptability.
Up to a certain point, it also let us define some basic characteristic of the camera, such as the focal lengths, although it does not let us correctly simulate some other effects such as lens distorsions (which again, can be added using a third party software such as MATLAB).
POV-Ray major drawback are reported in [23], and are:

- Only a Lambertian reflectance model is possible;
- A uniform surface albedo is used and realistic albedo values cannot used;
- The extended illumination source (sun) can be modelled only as an array of point light sources or as an area light, which is a suboptimal solution;
- Background lighting (starlight) is cannnot modelled;

- Earthshine cannot be easily modelled;
- POV-Ray can produceshigh quality images but rendering is slow as many minutes (sometimes hours) are required to produce a single image;

Despite those limitations POV-Ray can been used to produce synthetic space imagery with an acceptable degree of accuracy for CV algorithm training.

1.2 Spaceborne Close-Proximity Relative Navigation

1.2.1 Pose estimation sensors

1.2.2 Pose estimation tecniques

Chapter 2

Synthetic image generation

"In math, you're either right or you're wrong."

Katherine Johnson

The availability of a tool capable of rendering batch of images of the target client S/C is of vital importance for the implementation and testing of any CV algorithm. In this chapter the reader will firstly be presented with a brief mathematical introduction. After that, the proposed solution to generate synthetic space-borne imagery will be presented and analyzed.

2.1 Mathematical Preliminaries

2.1.1 Reference Frames

In order to be able to correctly analyze to problem of image generation, five reference frames are of interest:

- Geocentric Inertial Frame (GCI)
- chaser body-fixed frame
- camera frame
- target model frame
- target body-fixed frame

The GCI frame has its origin at the center of mass of the Earth. This frame has a linear acceleration because of the Earth's circular orbit about the Sun, which however can be neglected for attitude analysis. It's axes are aligned with the mean

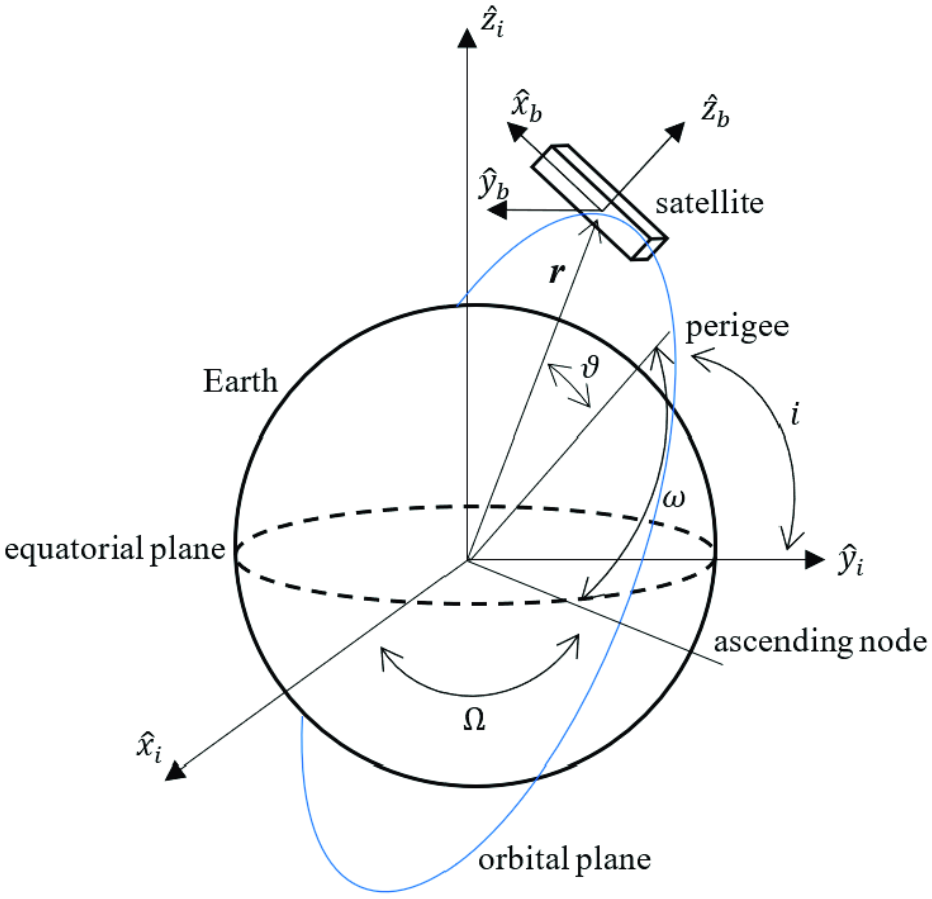


Figure 2.1: Geocentric Inertial Frame (GCI) frame

North pole and the mean vernal equinox at some epoch.

The chaser and target body-fixed frames have their origins at the CG of both, respectively, and their axes are usually aligned with their respective principal axes. The origin of the camera frame instead coincides with the center of perspective projection. The target model frame is fixed with respect to the body frame and often coincides with it. For what concerns this work, we will consider the camera frame to be coincident with the chaser body-fixed frame, and the target model frame to be coincident with the target body-fixed frame. The transformation from one reference frame to another can be uniquely defined by means of a translation vector and a rotation matrix. The translation vector represents the relative positions between the two different frames origins and the rotation matrix is determined by a sequence of three rotations about three different linearly independent axis by three angles, according to the Euler's angle representation.

2.2 Image generation

The use of artificial images gives a complete control over the scene. As stated in [29], the generated dataset should be as complete as possible in terms of metals and terrain features and illumination conditions. A lack of realism in image generation can lead to incoherent results, not representative of real operative conditions and thus can lead to wrong results in terms of CV algorithm tuning. A particular care is then required in the image generation process. For the purpose of this work, a new procedure for the generation of realistic images, representative of a dataset taken by a monocular navigation camera during a close-proximity approach to targeg S/C has been developed. Unsing the illustrated procedure, the user is able to create synthetic images of a target S/C given it's STL model, by fine tuning all the properties of the materials composing the target S/C. Since there could be also cases in which the Earth can be behind the target S/C, the developed tool is also able to simulate Earth's presence at any given location. The tool is also able to the simulate the the athmosphere of the Earth and the cloud layer [20]. The developed tool couples a well known open source raytracer (POV-Ray) with MATLAB in order to archive a good degree of realism in the generated images.

2.2.1 Ray Tracing

What is ray tracing

Ray tracing is a rendering technique used for generating artificial images which relies on the concept of controlling the path of view lines which starts from the observer camera and ends to generic virtual objects and thus calculating the color of the object. What is really of crucial importance for our particular use case is that ray tracing is capable of re-creating some of nature's optical effects

through transparent and opaque surfaces as reflection and refraction, scattering, and dispersion phenomena (such as chromatic aberration). Due to this, ray tracing techniques can generate artificial images with a really high degree of photorealism. When the ray tracer renders the scene, a ray of light is traced for every pixel of the camera. Typically, each ray must be tested for intersection with some subset of the object in the scene. Once the ray has intercepted an object, the ray tracing algorithm will estimate the amount and the type of incoming light at the point of intersection, examine the material properties declared by the programmer and by combining those information will calculate the final color that should be attributed to the pixel. Several illumination algorithms and reflective or translucent materials may also require more rays to be re-cast into the scene in order to do the necessary computations. At first glance may seem cumbersome to start the ray from the observer towards the object rather than casting rays to the camera (as is in reality). However, doing so speeds up a lot the computation time, as most of the light rays present in a scene may never reach the eye of the observer, and so, all the time spent for tracing those would be useless. After either a maximum number of reflections or a ray traveling a certain distance without intersection, the ray ceases to travel and the pixel's value is updated.

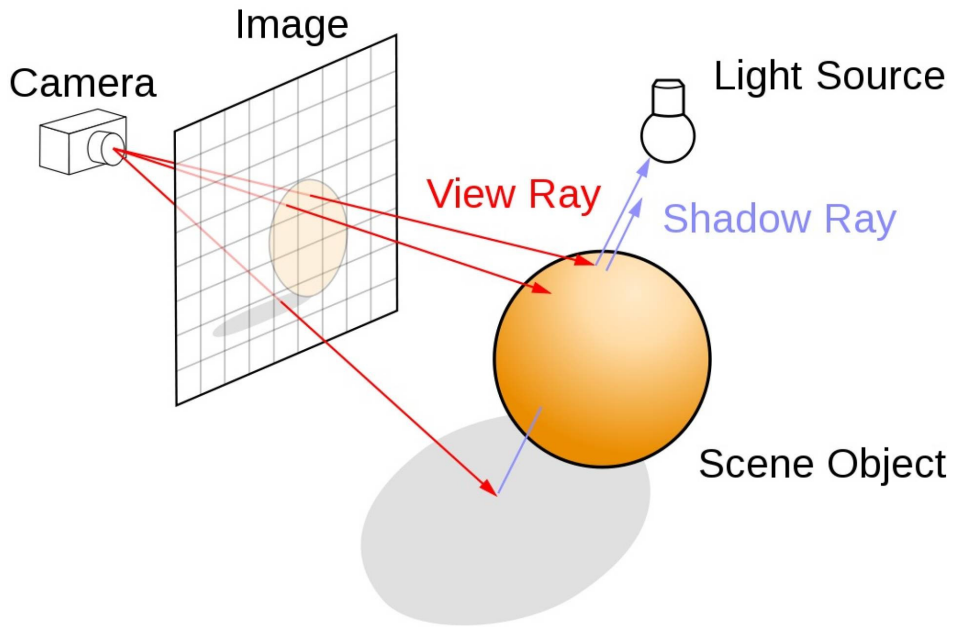


Figure 2.2: Raytracing: from the observer to the light source [10]

For more information regarding how ray tracing works, and ray tracing techniques see [25].

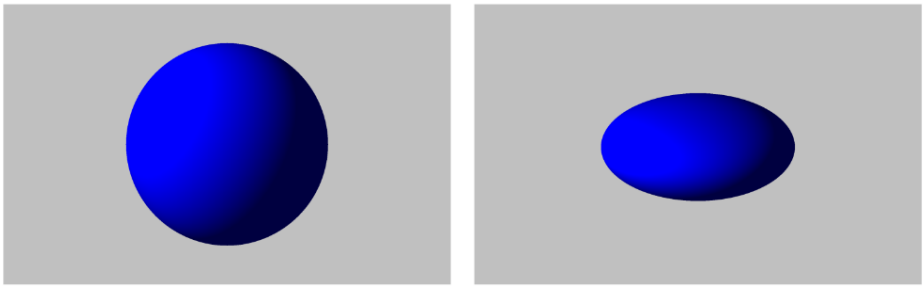
Ray tracing with POV-Ray

POV-Ray gives the programmer several options to customize both the look and feel and the optical properties of the represented objects and the medium that light passes trough. POV-Ray offers the programmer the choice to use some predefined geometric shapes, like spheres or cubes; another option instead is to define surfaces of the objects trough meshes. This capability has been used widely by this project to model the spacecraft object. Any surface can then be characterized by its own optical properties. Every object can have a color specified by an RGB triplet or can be wrapped with a texture. The light source is not really an object. Light sources have no visible shape of their own. They are just points or areas which emit light and can be tuned to accomplish the wanted result, for example we can tune the intensity of the light and the light color by specifying the RGB triplet; any atmospheric effect like opaque gas presence (like smoke or clouds) and its rlative optical distortions can be modeled as well.

2.2.2 Environment Modeling

Earth Modeling

Earth modeling has been worked out in collaboration with Jacopo Guarneri. Here will follow a brief review of how Earth has been modeled taken from [20]. For modeling the Earth, the POV-Ray **sphere** object has been used, in conjunction with the **scale** feature, which allows to make the sphere become an ellipsoid.



(a) Normal sphere (b) Differently scaled sphere

Figure 2.3: Sphere manipulation with POV-Ray [20]

Once the 3-D object is created, the most natural choice one can think of is to wrap a 2-D texture of the Earth (2.4) on it, and this indeed is what has been done, using the high resolution (8K) texture of the Earth.

In 2.5 we can see a comparison of the synthetically generated image and an actual true image of the Earth as seen from the Apollo 11. From a first glance, used texture gives a good representation of what we should see when we look of a picture of the Earth taken from the space.

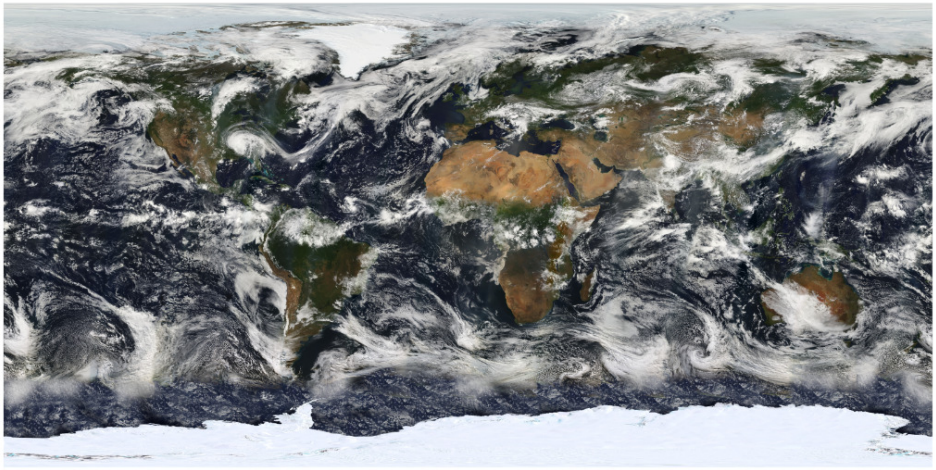
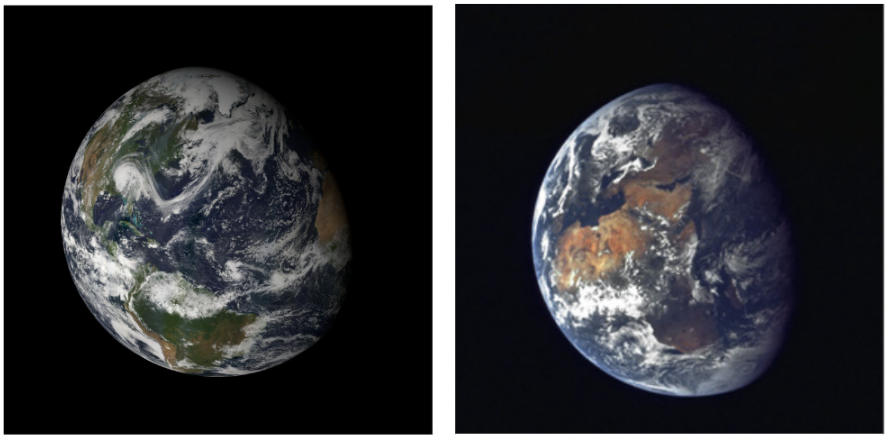


Figure 2.4: Initially chosen texture of the Earth



(a) POV-Ray representation of the Earth (b) Apollo 11's picture of Earth

Figure 2.5: Comparison of synthetic Earth image with Apollo 11's picture

Despite the relatively good result, this way of modeling the Earth still haves some issues :

- since for all pictures we use the same texture, the clouds are sticked to their position on top of the surface of the Earth;
- there is no way to define different optical properties for the terrain and the seas;
- there is no way to try to simulate diffusion of sunlight in the atmosphere;

In the following, those issues will be addressed.

Cloud Layer

The fixed coupling of Earth surface and clouds is big problem for what concerns CV algorithms development, which should be trained to work in several different conditions, so, having the clouds always on the same region of the Earth despite the orientation is not acceptable. The solution adopted in both this work and [20] relies on decoupling the Earth terrain layer and the cloud layer, specifying different optical properties for each layer, specify clouds relative rotation with respect to the terrain and coupling back everything together. Furthermore, oceans have been splitted too, in order to be able to set different optical properties for the seas. For the terrain layer, a a true-color mercator image of the Earth has been used as a base, in this way every piece of the surface could possibly be visible in any picture.

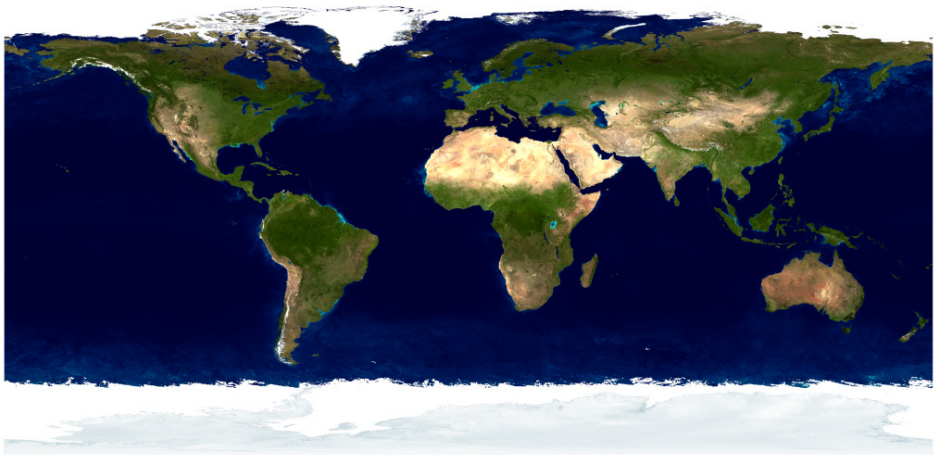


Figure 2.6: True Color Earth Mercator Image [1]

As said before, in order to be able to use different optical properties for water and terrains, a separate two-color mercator-projection image of the Earth where the water is black and the land is white has been used.

In 2.8 the result of differentiating the optical properties for terrains and oceans is shown. Despite the fact that it may seems that there is only a small distinction

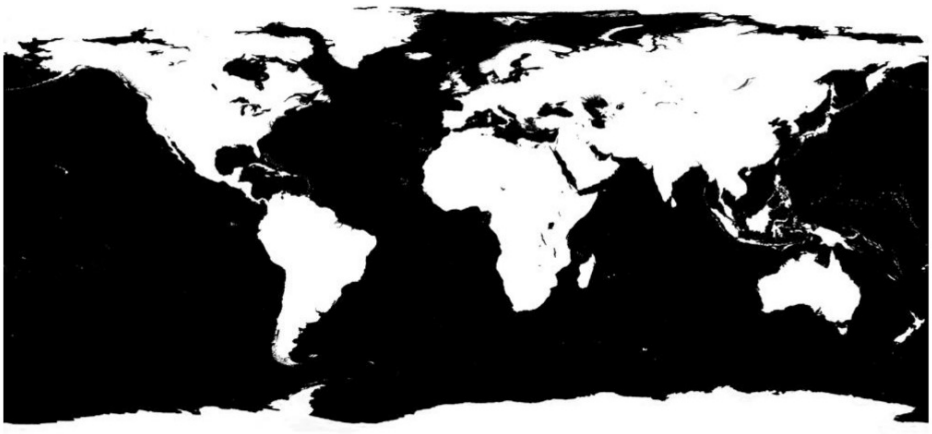
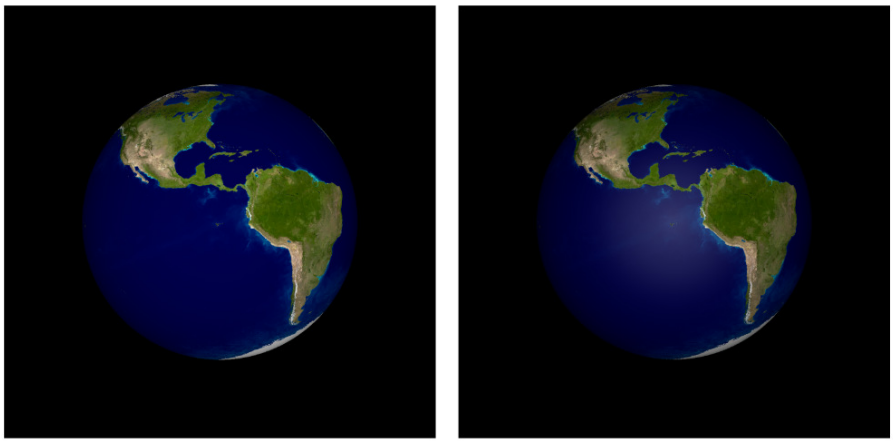


Figure 2.7: Landmask Mercator Image

between the two images (in particular, shinier oceans and darker forests), is still enough the make the Earth to seem more photorealistic, and it leave some space for further tuning.



(a) No differentiation of parameters (b) Differentiation of parameters

Figure 2.8: Comparing images with no differential treatment between land and ocean and with differential treatment

The cloud layer is added on top of the cloudless surface and thanks to that, it can rotate with respect to the Earth by a prescribed angle set by the programmer. The cloud layer texture and the shape of the clouds itself always remains the same, but this can be partially mitigated by using different cloud textures.

The cloud texture used for this work (2.9) is not actually directly printed on the surface of the earth, but rather is extruded on a shell built around the sphere which defines the Earth, which has an inner radius equal to $R_{in} = 1.001 \cdot R_{Earth}$ and an outer radius equal to $R_{out} = 1.0002 \cdot R_{Earth}$. Those values have been found

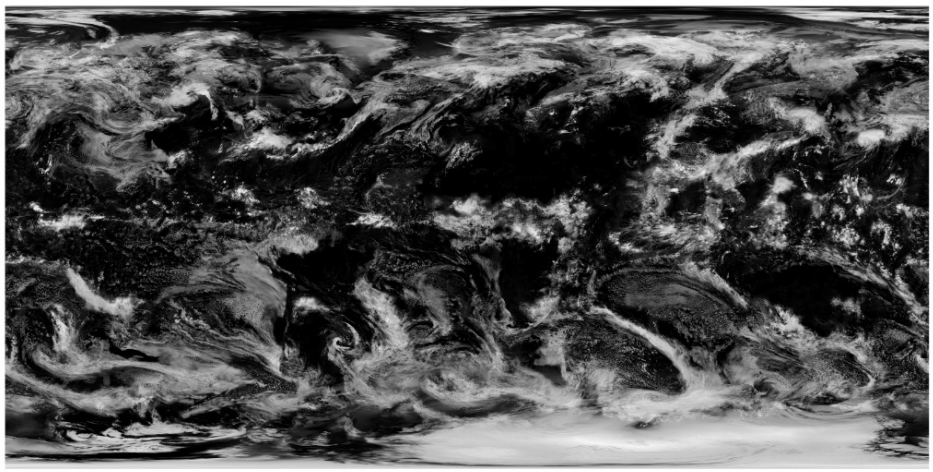


Figure 2.9: Two-color mercator image of Earth's cloud layer

by taking as a reference the fact that low Earth clouds ranges from an altitude of 600 m to 15 000 m [11]. Although higher or thicker clouds can be modeled, this will require longer rendering times. Of course, also for the cloud layer is possible to set custom optical properties, in order to make the clouds partially transparent, so that when there isn't a dense cloud area, the terrain behind is still visible under the white blanket.
The result of adding the cloud layer can be seen in 2.10

Cloud Layer
Recreating the characteristic atmosphere presence around the Earth is of crucial importance for developing a CV algorithm for space applications, as it is needed in order to train the algorithm itself to work into a real-case scenario. The atmosphere's presence in fact would enlarge the aspect of the Earth in the image, and furthermore would stress the edge detection algorithms. To recreate the atmospheric shine effect, a strategy which is similar to the one adopted to model the cloud layer has been used. In fact, it has been modeled as a shell with a certain thickness of a transparent material with some scattering properties. For what concerns both this project and what has been done in [20], the gaseous layer was made only 25 km thick. Despite the fact that the atmosphere should be visible up to many more kilometers, the computational load introduced by rendering a much high atmosphere is not negligible on a standard PC hardware, and so the image generation time would grow up exponentially, making the task of compiling a data-set of thousands of images very time consuming. In figure 2.11 can be seen the final result of adding the atmospheric model.

In figure 2.12 instead is possible to view the artificially generated next to a real Earth picture taken by NASA's Suomi NPP on January 4, 2012. It can be seen that, despite the fact that the real image isn't perfectly reproduced (because

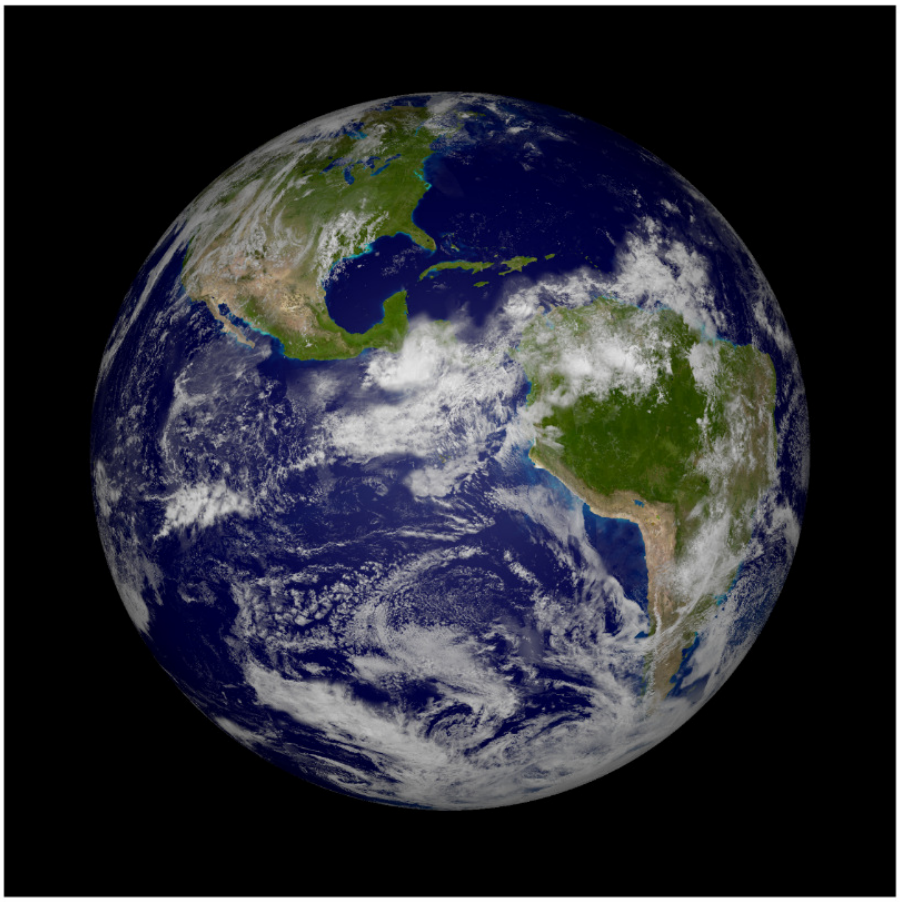


Figure 2.10: Earth's representation using cloud shell

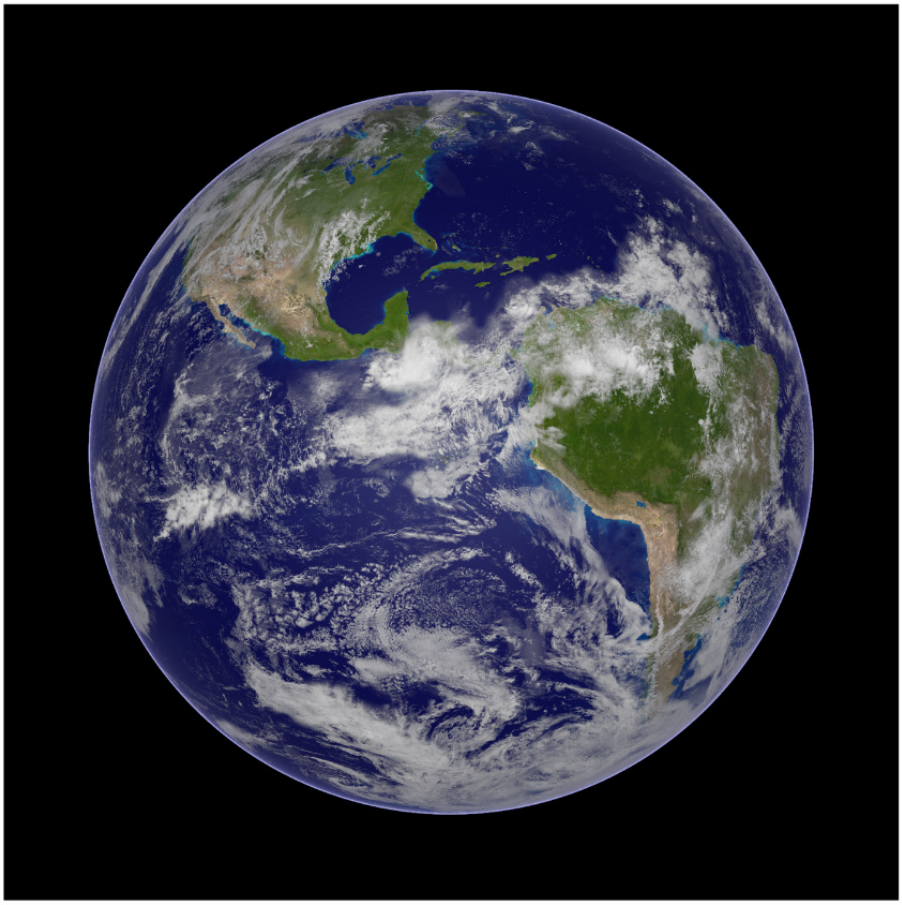
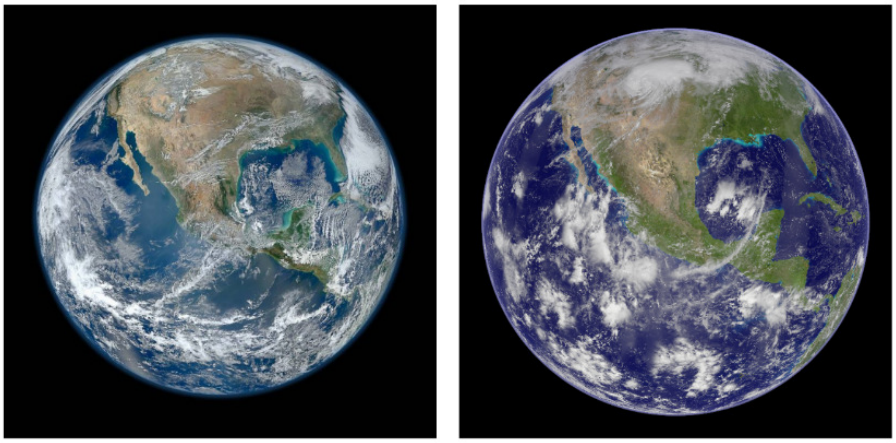


Figure 2.11: Earth with the atmosphere layer

some other factors should be known in order to be taken into account, such as exposure time), the synthetic image still provide an high degree of similarity with the real one.



(a) True Earth image (b) Synthetic Earth image

Figure 2.12: Comparison between true and final rendered Earth image

In table 2.1 are briefly resumed the values used to model the various layer of the Earth.

	Terrain	Oceans	Clouds	Atmosphere
Ambient	0.001	0.001	0.001	0.0001
Roughness	0.05	0.1	0.005	0.5
Brillance	1	1	1	1
Diffuse	0.85	0.85	1	0.6
Reflection	0	0.04, 0.25	0	0
Specular	0	0.1	0	0

Table 2.1: Optical parameters of Earth's different layers

Light Modeling

For an object to show up into the scene, it must be illuminated. There are two ways to illuminate an object with POV-Ray:

- use a standard light source
- use ambient light

The light source is controlled by the keyword **light_source**, which in turn accepts several modifiers. Light sources in POV-Rayhave no visible shape of their

own. They are just points or areas which emit light.
The ambient light instead is controlled by the keyword **ambient** added to the **finish** modifier of an object, and it is used to simulate the light inside a shadowed area. We can think of ambient light like a kind of light that is scattered everywhere in the room. It bounces all over the place and manages to light objects up a bit even where no light is directly shining. In our particular case, we can use the **ambient** option to simulate the illumation of the object due to spurious light sources (such as stars) or reflection from other bodies (like the Moon or other planets). In order to model the lightning condition of a true solar system, the light source has been modeled to resemble as much as possible the light emitted by the Sun. The solution adopted in this project and in [20] relies upon modeling the sun as an area light source through the option **area_light**. This allows to create a cluster of point-like light sources distributed on a disc (simulated by adding the circular option to the area light source) which has radius equal to the radius of the Sun, and placed at the exact distance which the Sun has from the Earth in the GCI frame. In order to cope with the fact that in reality the Sun is equal to a sphere of light and not a disc, the option orient has been used. When using orient, every object in the POV-Ray world would see the Sun's disc as oriented toward it, from any position around it. Furthermore, the option **jitter** has been used, which tells the ray-tracer to slightly move the position of each light in the area light to eliminate any shadow banding that may occur.

2.2.3 Tango Spacecraft Modeling

3D Model of the Spacecraft

The problem of modeling rather complex shapes with POV-Ray, such as can be a spacecraft, it was one of the most long and painful one during this work, which also required to patch POV-Ray source code, to prevent the ray-tracer to made assumption that aren't true. Obviously, the path of building the S/C using POV-Ray primitives was really not feasible, as many different S/C can have different shapes, and modeling them by using simpler shapes it is simply not possible. Furthermore, usually detaled 3-D CAD models of S/C are available which can be easily exported into STL format, so the focus has been putted into trying to understand how to translate those 3-D CAD models directly into POV-Ray code. Several open source and closed source software have been coupled together in order to produce trough POV-Ray a render of a given STL model. The procedure will be detailed in the following paragraphs. The Tango spacecraft has been modeled using the dimensions specified in [38], which will be here reported for ease of reading. The solar panel is represented by a polygon of 570 mm x 759 mm, while the spacecraft body instead is represented by a convex polyhedron of 560 mm x 550 mm x 300 mm. The radio frequency antennas length instead is of 204 mm. The origin of the CAD model is located in correspondence of the CG.

Using the aforementioned dimensions, the Tango spacecraft has firstly been reproduced using Autodesk Inventor. The result of the modeling procedure is shown in figure 2.13.

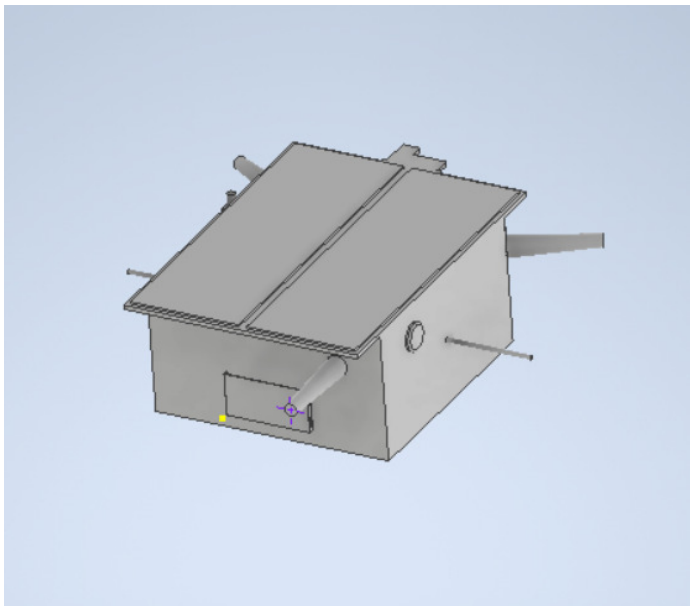


Figure 2.13: Tango S/C 3-D model

The 3-D CAD model can now be exported in STL format to be manipulated through other software.

Blender

Once the 3-D model of the Tango S/C was available, the most challenging task has been the one of rendering the S/C itself using POV-Ray at attitudes imposed by the user. The first step for archiving that goal is to import the STL model of the S/C into Blender. By exploiting the POV-Ray render add-on for Blender, it is possible to render any given STL file imported file using POV-Ray as rendering engine. The POV-Ray add-on will optionally save the generated SDL code used to render the scene. The generated code however, will threat the entire 3-D model as a whole, generating one giant POV-Ray mesh2 object, which is something which cannot be easily managed. Just as an example, would be impossible to assign to the different parts of the S/C different optical parameters or textures. So, to workaround this limitation, it is a good practice to first split the different surfaces of the 3-D model in different different children objects (or children surfaces), and only after render the scene in order to get the POV code. In that way, the POV-Ray render add-on for Blender will generate a mesh2 object for each children object created in Blender. This will enable us to set different material properties for each children surface or to apply different textures to different surfaces. For

the purpose of this work, the original STL model has been subdivided into thirty children object, each one with its own optical parameters, as can be seen from figure 2.14 .

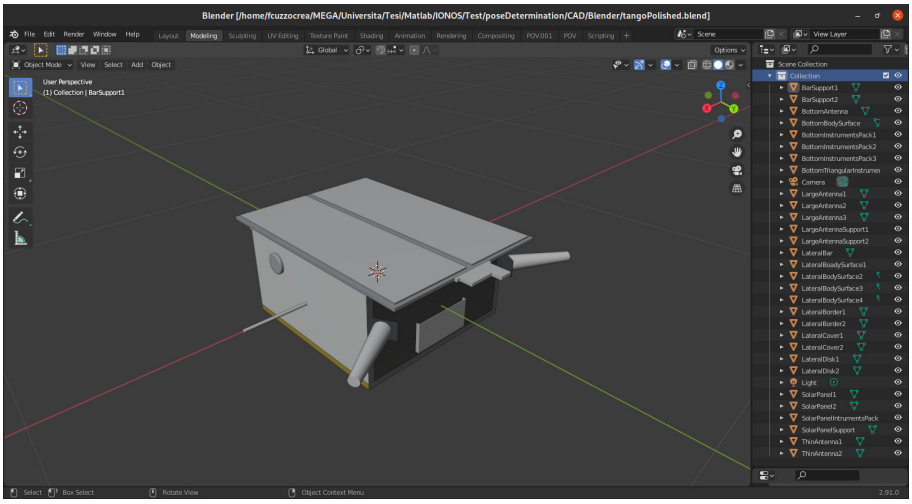


Figure 2.14: Tango S/C 3-D model in Blender

The Blender POV-Ray add-on also let the user to inject custom POV code (figure 2.15) into the auto-generated one, which can be useful especially for adding textures, for example to the solar panels, as it has been done into this project.

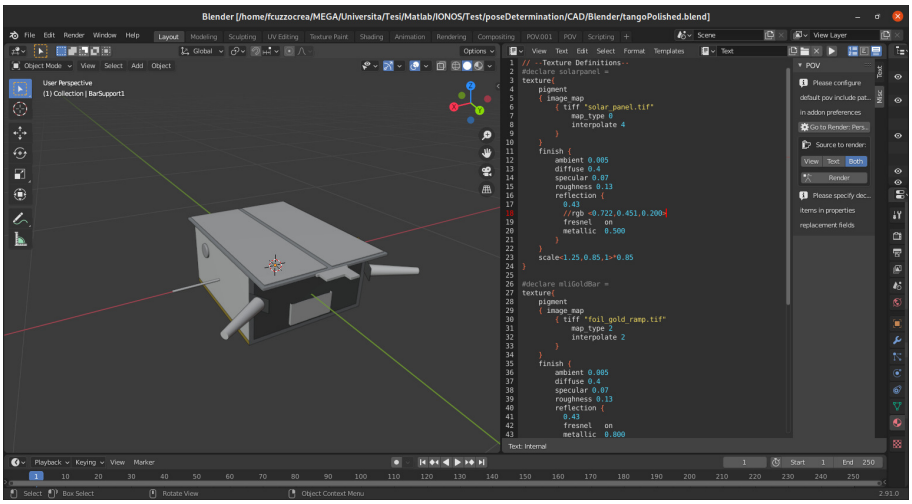


Figure 2.15: Add custom POV code into Blender

The end results is showed in figure 2.18

Figure 2.16: Texture used to model MLI

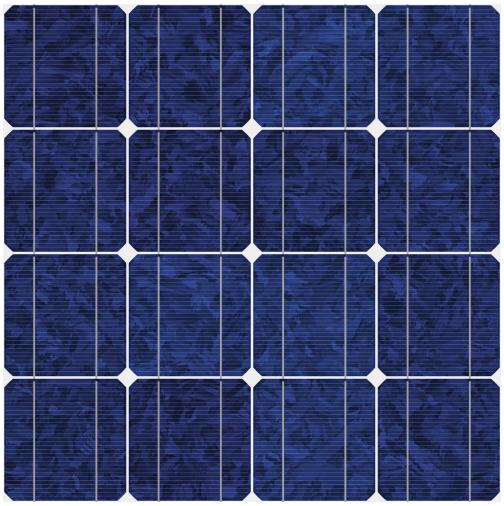


Figure 2.17: Texture used to model solar panels

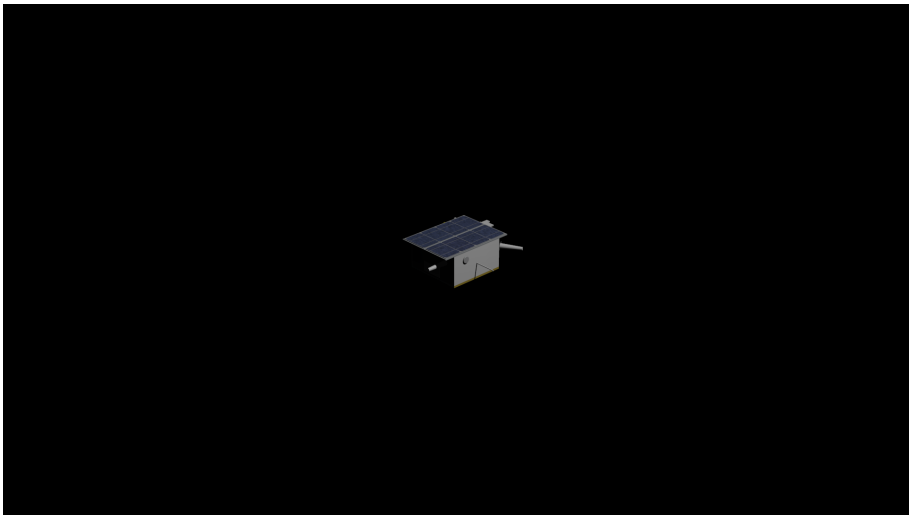


Figure 2.18: Tango S/C rendered using POV-Ray Blender add-on

POV-Ray

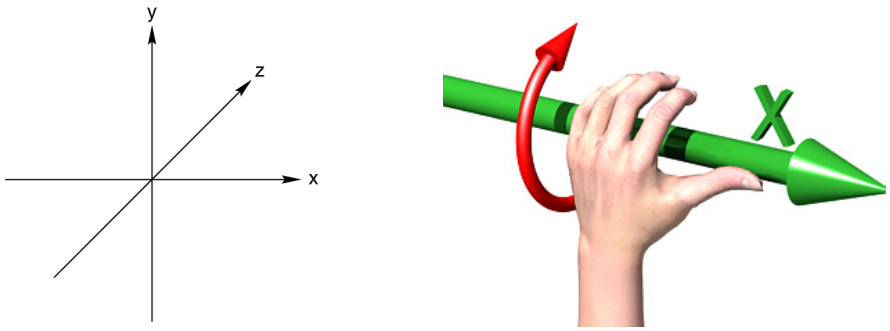
The major issue of the code generated from the POV-Ray Blender add-on is that is composed of several separated mesh2 objects which makes almost impossible to rotate the whole object by imposing a given attitude matrix, since one is supposed to rotate all the mesh2 objects by hand.
In order to workaround this limitation, from all the mesh2 objects which have been generated from POV-Ray Blender add-on are merged into one single **merge** object. The **merge** POV-Ray operation allows to bind two or more shapes into a single entity that can be manipulated as a single object, which is exactly what we want. The new object created by the merge operation can be scaled, translated and rotated as a single shape. The entire merge can share a single texture and optical parameters but each object contained in the union may also have its own texture and optical parameters, which will override any texture statements in the parent object. So, all the mesh2 objects which describes the S/C surfaces are merged into a single big (21K LoC) **spacecraft merge** object. To ease the usage of the **merge** object, a POV-Ray include file it is created, with the sole purpose of containing the **spacecraft merge** object. The include file is read in as if it were inserted at that point in the file. Using include is almost the same as cutting and pasting the entire contents of this file into the scene. This allow to define the **spacecraft merge** once and call it from any other POV file just like any other predefined object is called, and so, it is possible to manipulate its position and orientation in a much more easier way by just using the **matrix** keyword. The **matrix** keyword allows to specify directly the orientation of the **spacecraft** object, $\mathbf{A_{TN}}$, and its location with respect to POV-Ray "inertial" world. In table 2.2 are briefly resumed the optical parameters used to model the different part of the S/C.

	Solar Panels	Antennas	Main Body
Ambient	0.0	0.25	0.25
Roughness	0.13	0.0005	0.0005
Brillance	-	3.15	3.15
Diffuse	0.3	0.95	0.99
Reflection	0.23, 0.5	0.65, 1	0.65, 1
Specular	0.04	0.96	0.96
Phong	-	0.43	0.43
Phong Size	-	25	25

Table 2.2: Optical parameters of S/C different parts

2.2.4 Camera Modeling

POV-Ray let the programmer simulate a perspective pinhole camera (more details about the pinhole camera will be given in section 3.1.1). In POV-Ray's camera environment, the programmer can set all relevant camera parameters, which will be then used to simulate the camera trough which the scene will be rendered. The most meaningful parameters which can be imposed are the location of the camera itself, the direction of the boresight axis (where is the camera looking at), the view angle and the direction of the camera reference frame.
The major issue which has been faced when modeling the camera in POV-Ray is the fact that the program defaults to a left handed coordinate system to describe the scene, while all other software (like Autodesk Inventor, Blender) are using a right handed coordinate system.

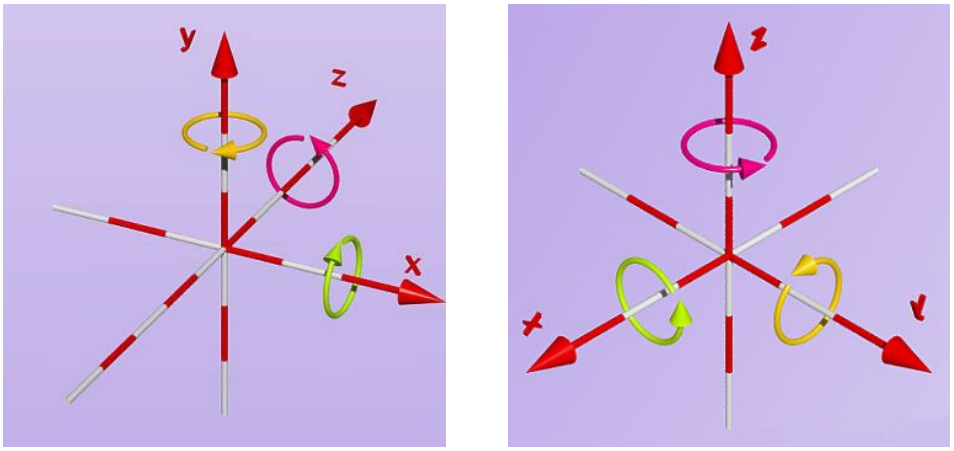


(a) POV-Ray axis directions (b) Left hand rule

Figure 2.19: POV-Ray coordinate system

Moreover, the right handed coordinate system is used also to model the orbit that the spacecraft will follow and the spacecraft attitude from Euler equations. It is possible to trick POV-Ray to behave like it using a right handed coordinate system by acting on the **right** vector of the camera environment. The camera

right vector describes the direction to the right of the camera, so, in practicte, tells POV-Ray where the right side of the screen is. Therefore, the sign of the x component of the **right** vector can be used to determine the handedness of the coordinate system in use.



(a) Left Handed Coordinate System (b) Right Handed Coordinate System

Figure 2.20: Left Handed Coordinate System and Right Handed Coordinate System

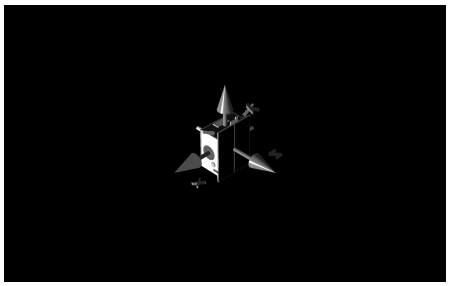
By default POV-Ray use a positive x value in the **right** vector. This means that the right side of the screen is aligned with the +x-direction. By using a negative x value in the **right** vector instead the right side of the screen will be aligned to the -x-direction, and the coordinate system will be right handed. Doing only that however left us having the y axis as the one pointing upward and the z axis as the one point in the direction which goes outside the screen. To have a more comfortable, which also uses the same axes of the GCI reference frame, we can flip y and z axis by overriding the **sky** vector. By default, in fact, POV-Ray uses <0,1,0> as **sky** vector. By redefining it as <0,0,1> POV-Ray will roll the camera until the top of the camera is in line with the **sky** vector, giving us the desired reference frame.

In order to rightly simulate a real camera, the POV-Ray camera aperture angle has been computed by assuming the intrinsic properties of a real camera, a Point Grey Grasshopper 3, equipped with a Xenoplan 1.9/17 mm lens, which is the same camera employed to capture the SPEED dataset [26].

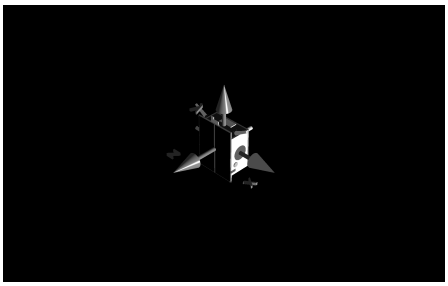
By assuming a square CCD sensor with square pixels we can obtain:

$$A.R. = \frac{N_u}{N_v}\,, \tag{2.1}$$

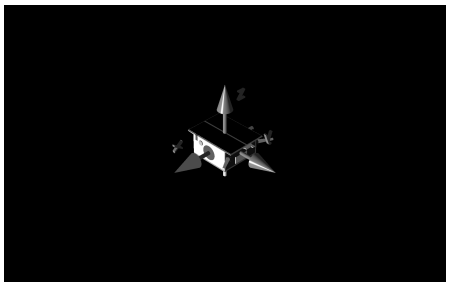
$$CCD_{size} = d_u \cdot N_u\,, \tag{2.2}$$



(a) POV-Ray standard coordinate system



(b) Result after imposing negative x to the **right** vector



(c) Result after imposing <0,0,1> to the **sky** vector

Figure 2.21: Reference Frame Rotations

Parameter	Description	Value
N_u	Number of horizontal pixels	1920
N_v	Number of vertical pixels	1200
f_x	Horizontal focal length	17.6 mm
f_y	Vertical focal length	17.6 mm
d_u	Horizontal pixel length	5.86×10^{-3} mm
d_v	Vertical pixel length	5.86×10^{-3} mm

Table 2.3: Parameters of the camera used to capture the SPEED images [26]

$$\alpha = 2 \cdot \arctan \frac{CCD_{size}}{2 \cdot f_x}, \tag{2.3}$$

where $A.R.$ is the Aspect Ratio of the picture, which is passed as first component **right** vector (with the negative sign, as described before), and α is the aperture angle which will be input in POV-Ray. Using the parameters detailed in table 2.3 we can compute:

- $A.R. = 1.6$
- $\alpha = 35.4515°$

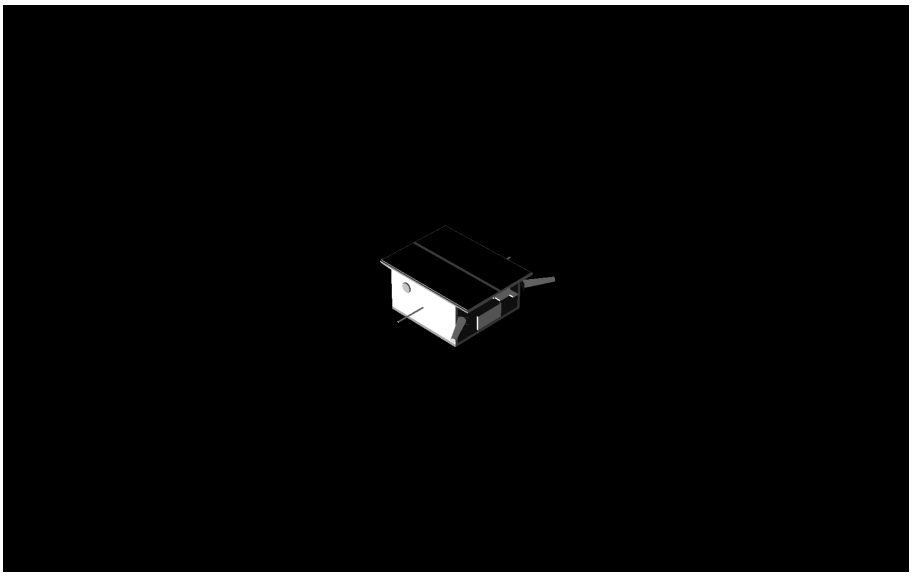


Figure 2.22: Tango S/C rendered using table 2.3 parameters

Camera Attitude

Being able to correctly know the imposed camera attitude at the moment of image generation is of crucial importance, especially when developing images to test pose determination algorithms. Using the **look_at** keyword we specify the direction of the camera's boresight axis (so, in practice, where the camera is looking at, as the keyword itself suggest). Therfore, by imposing the **look_at** vector, we are implicitly imposing the camera attitude with respect to what is looking at. For what concerns this work, it is assumed that the camera always looks at the center of the target S/C. By imposing **sky** and **right** vectors we impose the initial direction which POV-Ray uses for orienting the camera. By imposing the **look_at** vector POV-Ray will first roll the camera until the top of the camera is in line with the sky vector. Then it uses the **sky** vector as the axis of rotation left or right and then to tilt up or down in line with the **sky** until until it lines up with

the **look_at** point. Then tilts straight up or down until the **look_at** point is met exactly.

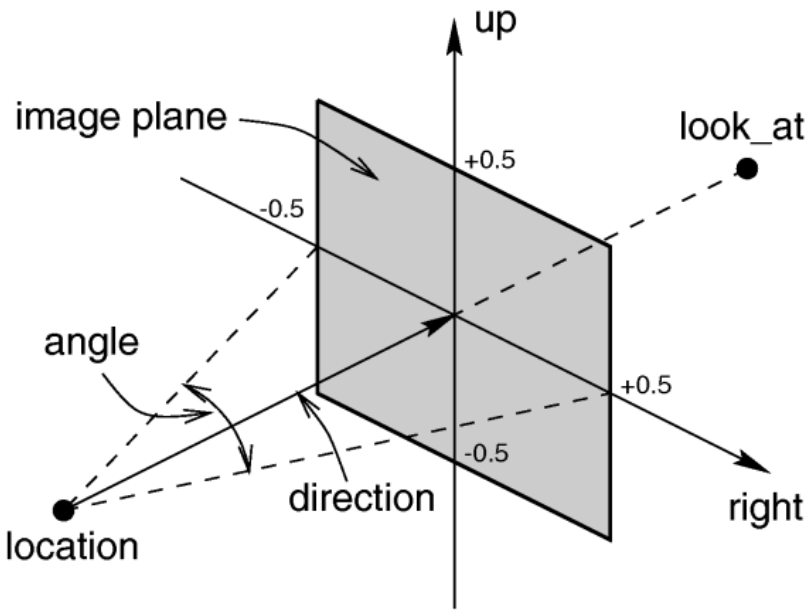


Figure 2.23: POV-Ray default perspective camera

So, by knowing how POV-Ray rotates the camera to point to the the **look_at** point, we can compute the actual attitude matrix of the camera with respect to the "inertial" POV-Ray world by exploiting simple linear algebra rules:

- the z-direction, which is the one going straight out from the camera, can be computed by simply making the difference between the **look_at** vector and the camera **location** vector;
- the x direction, the one going right on the image plane, can be found by the cross product between the z-direction and the **sky** vector
- the y direction, which is the one going downward in the image plane, can be found by performing the cross product between the z-direction and y-direction

$$\hat{\mathbf{z}} = \frac{\mathbf{look\_at} - \mathbf{location}}{\|\mathbf{look\_at} - \mathbf{location}\|}, \tag{2.4}$$

$$\hat{\mathbf{x}} = \frac{\hat{\mathbf{z}} \wedge \mathbf{sky}}{\|\hat{\mathbf{z}} \wedge \mathbf{sky}\|}, \tag{2.5}$$

$$\hat{\mathbf{y}} = \frac{\hat{\mathbf{z}} \wedge \hat{\mathbf{x}}}{\|\hat{\mathbf{z}} \wedge \hat{\mathbf{x}}\|} \,. \tag{2.6}$$

So the attitude of the camera with respect an the inertial POV-Ray frame, $\mathbf{A_{CN}}$, can be simply defined as :

$$\mathbf{A_{CN}} \triangleq [\hat{\mathbf{x}}, \hat{\mathbf{y}} \,. \hat{\mathbf{z}}] \tag{2.7}$$

At this point, the full pose ($\mathbf{A_{TC}}$, $\mathbf{t_C}$) of the camera with respect to the S/C is defined by:

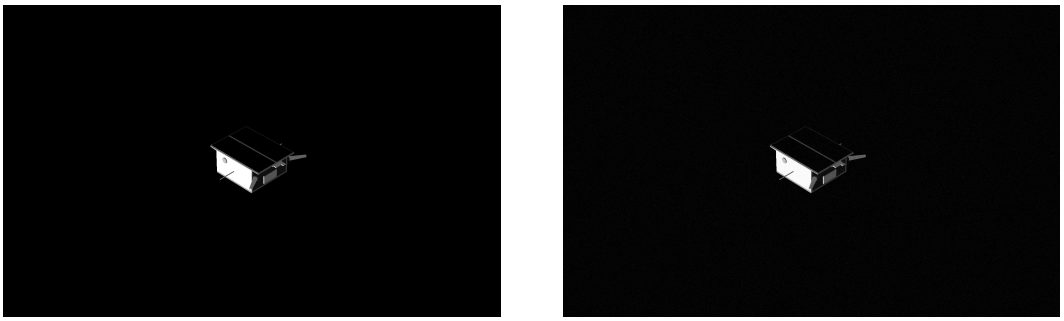
$$\mathbf{A_{TC}} = \mathbf{A_{CN}} \mathbf{A_{TN}}' \,, \tag{2.8}$$

$$\mathbf{t_C} = -(\mathbf{look\_at} - \mathbf{location}) \mathbf{A_{TN}}' \,. \tag{2.9}$$

At this point, the pose imposed to the camera is completely defined.

2.2.5 Noise modeling

Finally, for augmenting the "realness" of an image generated though POV-Ray, the image needs to be post-processed using MATLAB. The image so is input in MATLAB and two different noises are applied through the imnoise command : speckle noise ($\sigma^2 = 0.004$) and Gaussian white noise ($\sigma^2 = 0.003$). The final result can be seen in image 2.24.



(a) Image without noise added

(b) Image wit speckle and Gaussian white noise

Figure 2.24: Comparison between image without noise and image with speckle and Gaussian white noise

2.3 MATLAB integration

Since it would be highly unpractical to hand edit every time all the parameters to render the S/C in different position and attitudes as well as Earth's different rotations, a MATLAB toolbox has been developed which allows the user to automatically generate .pov SDL files, and it renders them trough POV-Ray automatically. The implemented toolbox takes as input N_u, N_v, f_x, f_y, d_u, d_v to compute intrinsic camera parameters to set POV-Ray camera. Then, it offers the user the possibility to generate a sequence of random attitudes, given the orbital parameters of a reference orbit. If inertial properties of the target S/C are available, instead, it can compute the full uncontrolled dynamics (so to speak, the dynamics of the S/C when only subjected to the disturbances torques acting on the reference orbit) of the S/C trough a Simulink model at each time instant. Details about how random attitude matrices are computed and how the Space environment has been modeled in Simulink are give in A and B, respectively. For each time instant for which the attitudes of the target S/C have been computed, the relative position of the camera with respect to the target S/C is computed by selecting a random point inside a sphere having a 20 m radius centered in the target position, and the imposed pose is computed. A random Earth's rotation is computed as well. All the properties of the scene are stored in a custom .att file, which can be passed in a second time to retrieve the absolute position and orientation of both the camera and the target S/C and their pose. A .pov SDL file is written for each time instant for which the attitudes of the target S/C have been computed and POV-Ray is called to render each image in the background, from within MATLAB. Rightly after an image has been rendered, it is post-processed by MATLAB in order to add the noise, as described in 2.2.5.

2.4 Conclusions

It must be noted by the reader, that a custom POV-Ray version is required in order to correctly replicate what has been done in this project. The issue originates by the fact that at the beginning of this project it has been decided that 1 unit in the POV-Ray world is exactly to 1 km to ease the modeling of the solar system. Thus, the Earth is placed at the origin of the POV-Ray world (consistently with the GCI frame)and it has a 6378 km radius, and the light source, the Sun, is placed 1.4723×10^8 km far from the Earth, and has a radius of 696 340 km. The S/C instead is placed on an orbit characterized by the orbital parameters specified in table 2.4. The inertial properties of the target instead are computed by assuming a mass of 50 kg and using the dimension given in 2.2.3.

The issue originates from the fact that what is being represented is a scene having an extremely small object (the S/C) in front of a giant object the Earth, which is impacted by light which is generated from a very far distance (the Sun).

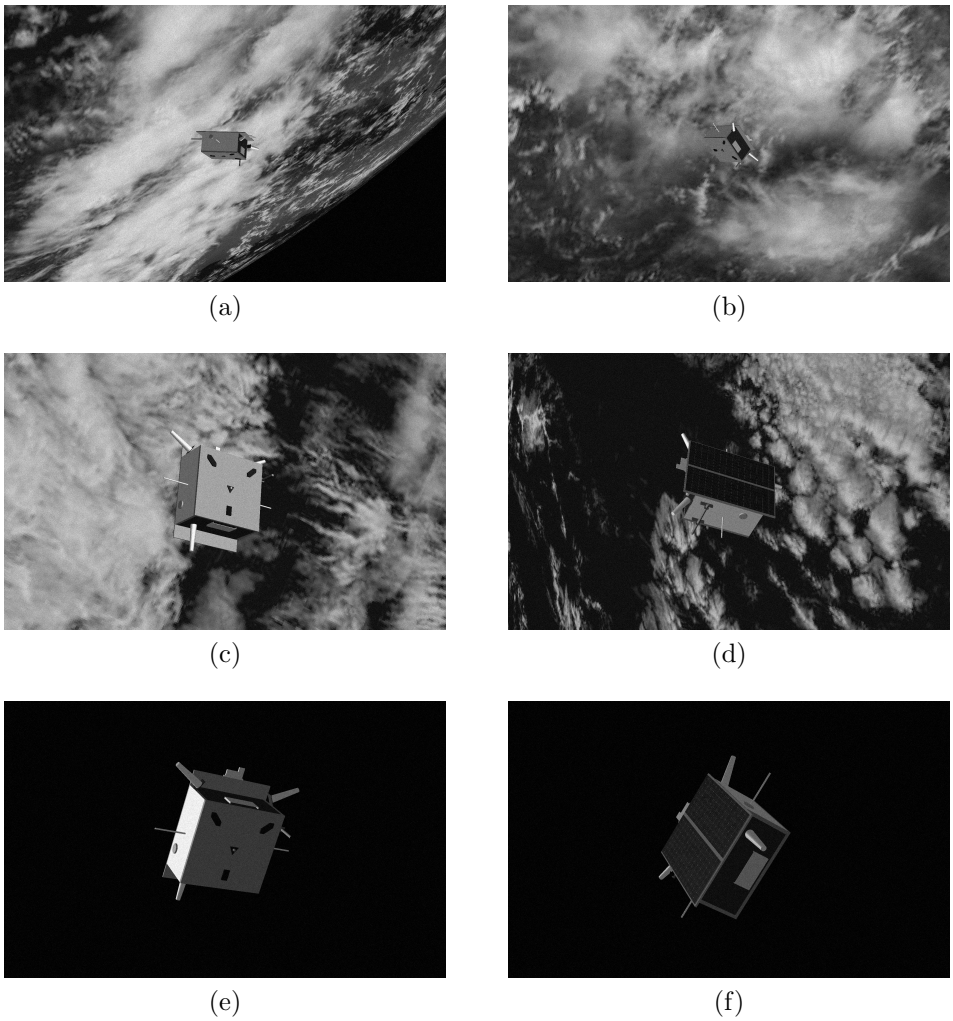
Orbital Parameters	Symbol	Values
Apogee	r_a	7178 km
Perigee	r_p	7101 km
Semimajor axis	a	7133 km
Inclination	i	98.28°
Pericenter anomaly	ω	0°
True anomaly	θ	0°
Eccentricity	e	0.0045°

Table 2.4: Parameters used to generate the reference orbit [2]

Due to the exquisite peculiarity of this kind of scene, in some corner cases a "bounding issue" can be triggered, causing the S/C to be only partially redendered in the scene, or, in worst case scenario, not being rendered at all. Directly citing the POV-Ray documentation, in order to speed up the ray-object intersection tests POV-Ray uses a variety of spatial sub-division systems. The primary system uses a hierarchy of nested bounding boxes. This system compartmentalizes all finite objects in a scene into invisible rectangular boxes that are arranged in a tree-like hierarchy. Before testing the objects within the bounding boxes the tree is descended and only those objects are tested whose bounds are hit by a ray. This can greatly improve rendering speed. However, during the development of this project turned out that POV-Ray automatic bounding is not perfect. There are situations where a perfect automatic bounding is very hard to calculate, like the one faced in this project. Unfortunately, POV-Ray developers removed the possibility of turning off bounding control in POV-Ray, thus it is necessary to compile a custom POV-Ray version which introduces back the command line switch **-MB** which allows the user to turn off bounding control. According to the GPL license POV-Ray is shipped with, the patch has been made freely available [1].

The output of the toolbox which has been described trough this chapter can be observed in figure 2.25.

[1] `https://github.com/fcuzzocrea/povray/commit/2aecdfb20eef3ed3b6a5698392a9c91fa3f1afcd`



(a) (b) (c) (d) (e) (f)

Figure 2.25: Final result

Chapter 3

The SVD architecture for pose initialization

"Prediction is very difficult, especially if it's about the future."

Niels Bohr

Performing a robust pose initialization from a 2-D image captured by a monocular camera in space is a very though task, particularly due to the fact that illumination conditions in space may vary during a single orbit, and therefore may cause inaccurate and unreliable features detection. Moreover, the Earth presence in the background introduces a major complexity into distinguishing the target S/C shape from the Earth surface. The SVD pose initialization architecture aims at solving the problem of pose initialization by employing a single 2-D image taken by a monocular camera. In this chapter first the reader will be proposed with a brief Computer-Vision (CV) mathematical concerning some common Computer-Vision (CV) models and problems. After that, a detailed description of the SVD architecture implementation which has been carried out in this work will be given.

3.1 Mathematical Preliminaries

3.1.1 Pinhole Camera Model

The pinhole camera model is camera model which is widely used in Computer-Vision (CV). Despite being a relatively simple model, it is still accurate enough for the vast majority of applications. The pinhole camera model is used to describe the mathematical relationship which exists between the coordinates of a point in the 3-D world and its projection onto the image plane. In the pinhole camera model, the light passes trough a single point, the camera center, before being projected onto

the image plane, and no lenses are used to focus light, so geometrical distortions or blurring are not modeled in the pinhole camera model.

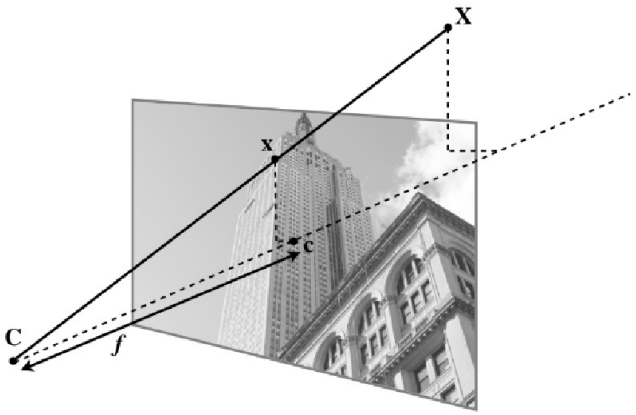


Figure 3.1: The pin-hole camera model [39]

By referring to 3.1, a 3-D point $\mathbf{X}$ is projected to an image point $\mathbf{x}$ (both expressed in homogeneous coordinates) as :

$$\lambda \mathbf{x} = P\mathbf{X}\,,$$

where P is a 3-by-4 matrix called camera matrix and λ is a scalar number representing the inverse depth of the 3-D point. As a result, the 3-D point $\mathbf{X}$ is characterized by four elements, $X = [X, Y, Z, W]$ in homogeneous coordinates. Generally, the camera matrix P can be decomposed as :

$$P = K[R \mid \mathbf{t}]\,,$$

where R is the rotation matrix which describes the orientation of the camera, $\mathbf{t}$ is a 3-D translation vector which describes the position of the camera center and K is the intrinsic camera calibration matrix. The camera calibration matrix basically describes the projection properties of the camera and can be written as :

$$K = \begin{bmatrix} \alpha f & s & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}\,,$$

where α is the aspect ratio of the sensor's pixels, f is the focal length, s is the skew and c_x, c_y are the coordinates of the optical center (or also called the principal point) of the image. Usually the principal point of the image can be approximated with half the height and half the width of the image. The skew is used only if the pixel array in the sensor is skewed. In most cases is safe to assume $s = 0$. By defining :

$$f_x = \alpha f_y$$

and by neglecting s we can write the calibration matrix in the most common form :

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

If we make the assumption of square pixel, then $\alpha = 1$ and $f_x = f_y = f$, and so we can write the camera calibration matrix as :

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

More information about camera model and camera calibration matrix can be found in [21] and [34].

3.1.2 Image derivatives

For a grayscale image, the image intensity changes over the image itself can be described by using the x and y derivatives, I_x and I_y of the graylevel image I. Once defined the image derivatives, it is possible to define another quantity, the image gradient, which is the vector:

$$|\nabla I| = \sqrt{I_x^2 + I_y^2},$$

which describes how strong the image intensity change. Image derivatives can be computed using a discrete approximations, which are implemented as convolutions:

$$I_x = I * D_x \quad , \quad I_y = I * D_y,$$

where D_x, D_y are the kernel of the derivative and $*$ is the 2-D convolutional operation. Two common choices for D_x, D_y are either the so called Prewitt filters :

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad D_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

or the so called Sobel filters:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix},$$

3.1.3 The Hough Transform

The Hough transform is a method often used in Computer-Vision (CV) for finding shapes in images. It wors by using a voting procedure in the parameters space of the shapes. The most common use of the Hough transform is to find lines in images. Consider the straight line equation, in its polar form,

$$\rho = x\cos\theta + y\sin\theta\,,$$

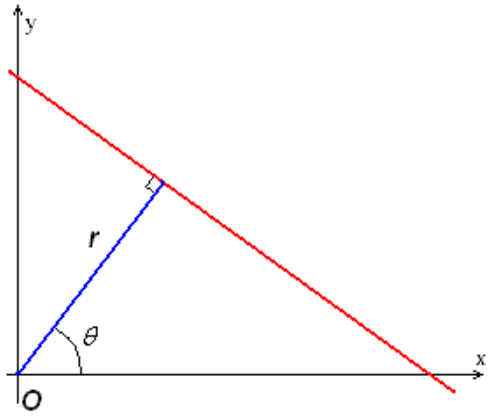


Figure 3.2: Polar representation of a straight line [9]

where ρ obviously is the distance from the origin of the reference system to the closest point on the straight line, while θ is the angle between the x axis and the line connecting the origin with that closest point. The linear Hough transform algorithm estimates the two parameters that define the straight line. The transform space has two dimensions, and any point in the transform space is used as an accumulator (a bi-dimensional vector) to detect a line described by the aforementioned polar equation of the straight line. Every point in the detected edges in the image contributes to the accumulators. Generally, the dimension of the accumulator is equal to the number of unknown parameters, which in this case are two, ρ and θ. For each pixel and is neighborhood, the algorithm which computes the Hough transform first try to detect if the specific pixel which is being analyzed lie on a line. If so, computes ρ and θ of that line, and then look for the accumulator's bin that the parameters fall into, and increment the value of that bin. By searching the boxes with highest values, typically by looking at local maxima in the accumulator space, it is possible to identify the the most likely lines. The final result of the Hough transform will be a 2-D matrix, were one dimension will be represented by θ and the other one by ρ. Each element of that matrix will have a value equal to the sum of the pixels that are positioned on the line represented by the parameters ρ, θ. So the element with the highest value indicates the straight line that is most represented in the input image [24].

More information about the how the Hough transform works can be retrieved in [15] and [22].

3.1.4 Perspective-n-Point problem

The Perspective-n-Point (P-n-P) point is the problem of determining the position and the orientation of a calibrated camera given its intrinsic parameters and a set of correspondences between a given set of 3-D points and their respective 2-D projections in the image [28]. The perspective projection for a standard pinhole camera that has been introduced in section 3.1.1 leads to the following equation (in homogeneous coordinates) for the model:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

were r_{ij} and t_i are the components of the rotation matrix and the translation vector which are being calculated, respectively. Several methods exists for solving the P-n-P problem, the two most common of which are the P3P method and the eP-n-P method. More information about their implementations can be found on [19] [41] and [28].

3.2 Feature-based pose estimation implementation: the SVD algorithm

The images generated trough the toolbox presented in Chapter 2 will now be analyzed using the robust monocular vision-based pose initilization algorithm proposed by S. Sharma, J. Ventura and S. D'Amico in [38]. As briefly explained in the introduction, the problem of pose initialization consists in computing the rotation matrix,$\mathbf{A_{TC}}$ and the translation vector, $\mathbf{t_C}$ that describes the transformation between the camera frame, C, and the target body frame, B. Given a generic image point, $p = [u, v]^T$, we can relate the corresponding $\mathbf{q_B}$ point of the 3-D model by employing the standard pinhole camera model briefly reviewed in section 3.1.1:

$$\mathbf{r_C} = [x_C \quad y_C \quad z_C]^T = \mathbf{A_{TC}}\, \mathbf{q_B} + \mathbf{t_C}\,, \tag{3.1}$$

$$\mathbf{p} = \left[\frac{x_C}{z_C} f_x + C_x, \frac{y_C}{z_C} f_y + C_y \right], \tag{3.2}$$

where f_x and f_y are the respectively the horizontal and the vertical focal length and $[C_x,\ C_y]$ are the coordinates of the center of the image. Moreover, it is assumed - without loss of generality - that the direction C_3 of the camera frame

is pointed along the boresight of the camera and the directions C_1 and C_2 are aligned with the image frame defined by $P = (P_1,\ P_2)$

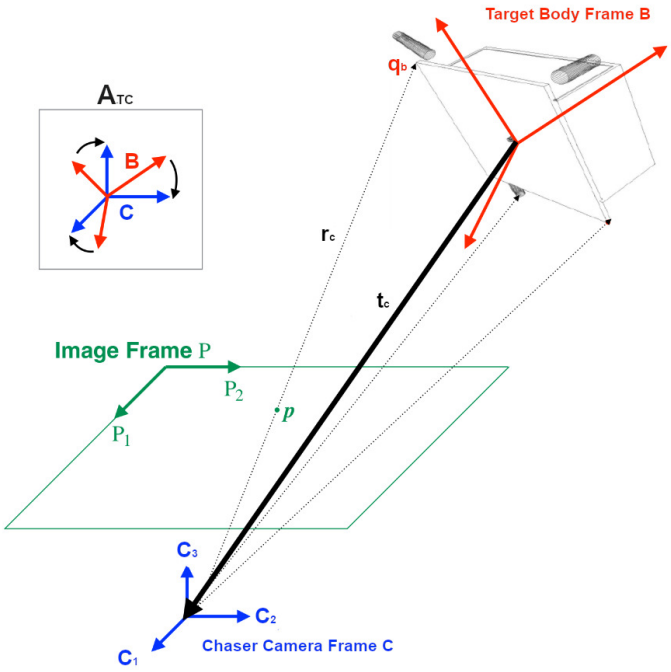


Figure 3.3: Schematic representation of the pose estimation problem using a monocular image [38]

To be able to uniquely solve this set of equation are needed at least six correspondences between the image points and the model points [16]. The challenging part of the problem is to be able to correctly detect the most meaningful points in the 2-D image. Usually this can be archived by employing edge detection techniques, such as the Canny edge detector [6] followed by the Hough transform [15]. This approach however may be biased it applied dierctly to the image due to the fact that these algorithm are gradient based and are not able to correctly distinguish the foreground from the background and also requires the definition of numerous hyperparameters which are difficult to tune for broad applicability because the imaged scene and the illumination conditions are constantly changing throughout the orbit [38].

3.2.1 General Architecture

In [38] is proposed a novel architecture to solve the initial pose of a non cooperative client S/C, whose pipeline is depicted in figure 3.11.

Its key features are (directly citing [38]) :

- the fusion of the WGE technique with the more traditional Sobel edge detector and the Hough algorithm to perform the feature detection;

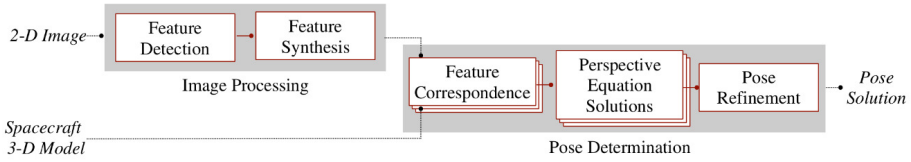


Figure 3.4: The SVD architecture for solving the pose estimation problem [38]

- the use of feature synthesis to reduce the search space for feature correspondences between the features extracted from the images and the 3-D model;
- the combination of the eP-n-P solver and the Netwon-Raphson (NR) method for the final pose determination.

As the reader can see from the schematic represented in figure 3.11, the pipeline is composed from two different subsystems:

- the image processing subsystem, which receive as an input a 2-D image, distinguishing the S/C from the image, extract its edge features and groups them into geometric groups;

the pose determination subsystems, which acceps as input the previously feature groups detected by the image processing subsystem and the 3-D model. It then pairs the 2-D and 3-D geometric groups to formulate multiple correspondence hypotesis. For each hypothesis formulated, the endpoints of the line segments forming the geometric groups are used to solve equation (3.1) and equation (3.2). The five best solution then are iteratively refined by employing the NR method.

two are the most meaningful innovation the SVD architecture introduces:

- the usage of the WGE technique in order to distinguish the target S/C from the background and for enhancing the output of the edge detection procedure bu providing a robust identification of the small as well as large edges of the S/C;
- the usage of feature groups detected in the image and in the 3-D model to solve the feature correspondence problem, which also allows the SVD architecture to be more computationally efficient.

3.2.2 Image processing subsystem

The taks of the image processing subsystem is to extract the most meaningful features from the 2-D image a of the target S/C. To effectively and rapidly detect the target S/C edges - even in presence of the Earth in the background - an hybrid image processing subsystem is proposed in [38]. By fusing together the WGE technique and classical state-of-the-art edge detection techniques the

architecture proposed by Sharma *Et al.* is capable to provide a robust and efficient identification of the rue edges of the S/C. In particular, the WGE technique is able to identify the a Region of Interest (ROI) in a more accurate and robust way with respect to state-of-art techniques. The ROI is detection not only makes the entire subsystem robust against the background, but also allows an automated selection of the hyperparameters needed for the Hough transform, which will be used to detect both small and large features of the S/C, in contrast with state-of-the-art algorithms which rely on a single set of manually tuned hyperparameters. The flow of the image processing subsystem is illustrated in figure 3.9.

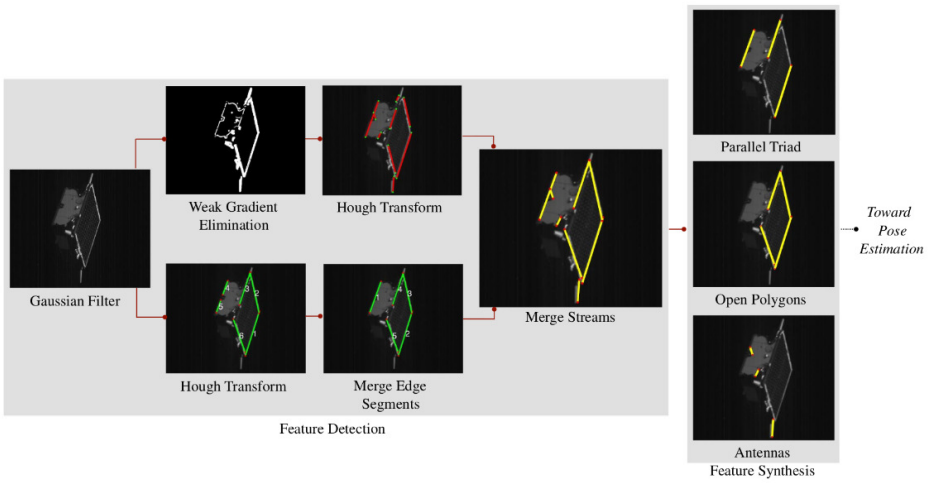


Figure 3.5: The image processing subsystem [38]

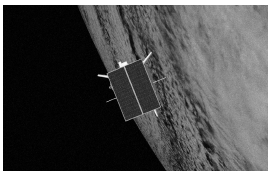
Feature Detection

The first sub-block of the pose estimation technique proposed in [38] is the feature detection subsystem, whose aim is to identify, in the input image, a set of segments that corresponds to the true edges of the target S/C. The raw input image - assumed to be rectified - is imported in MATLAB and is converted to the uint8 data type trough the command im2uint8 . After that, A Gaussian filter is applied in order to the decrease the magnitude of image noise. The Gaussian filtering can be carried out by using the MATLAB function imgaussfilt , which let the user set the desired filtering depth. For what concerns this work, the images generated in chapter 2, before being fed to the image processing subsystem have been filtered by using a standard deviation $\sigma_X = 1.15$. During this work, has been observed that the choice of the σ_X value is very important for the success of the subsequent steps, as a wrong value can highly impact the effectiveness of the WGE technique. Once the image has been prefiltered, is fed as input to the WGE block, for the computation of the target's ROI. The first step consist in computing the magnitude of the image gradient, $|\nabla I|$, using the Prewitt filter for computing

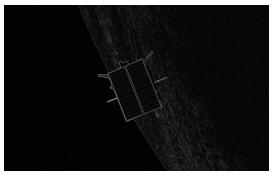
the image gradient, as showed in section 3.1.2. The S/C can be detected in the image by observing that, under most illumination conditions, in correspondence of the target S/C edges the gradient variation is more pronounced with respect to the background, regardless of the fact that the background is empty or the Earth is behind. Indeed, to detect the spacecraft, the gradient distribution is normalized using the MATLAB mat2gray function, sorted into 100 uniformly distributed bins and fitted by and exponential Probability Distribution Function (PDF), described by the equation :

$$y = \frac{1}{\mu} e^{-\frac{x}{\mu}} .$$

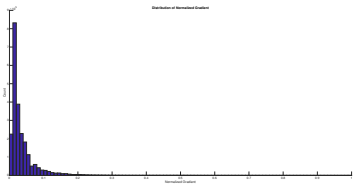
By observing the result histogram, it is clearly possible to see that most of the gradient intensities are weak and corresponds to the feature in the background or on the spacecraft surface. The weak gradient pixel locations can be classified by tresholding the PDF fit to the gradient histogram. The original authors suggest to use a value of .0.99 to treshold the PDF, however, for what concerns this work, all the pixel which corresponds to a cumulative distribution inferior to 0.996 are classified as weak and set to zero instead. Once the weak gradients are set to zero, only the most prominent features of the S/C are present in the image, as shown in figure 3.6.



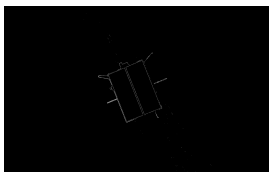
(a) Original image



(b) Gradient detection of original image



(c) Histogram of normalized gradient values



(d) Output filtered image gradients

Figure 3.6: Different steps of the WGE

By computing the Cumulative Distribution Function (CDF) of the vertical and horizontal gradient of the filtered image obtained by the WGE, we can determine the coordinates of a ROI where the features that yield the strongest intensity variations are located. Then, assuming that the filtered image gradient is normally

distributed, the ROI coordinates can be found by limiting each of the previously computed CDFs between 0.025 and 0.975 in order to enclose the central 95% of the normal distribution.

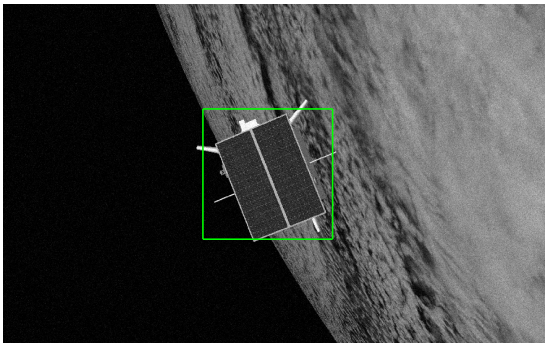
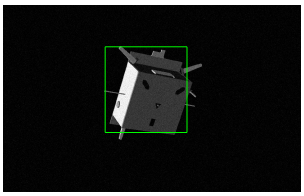
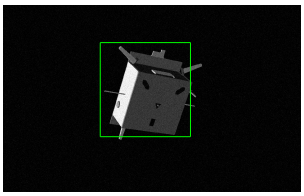


Figure 3.7: Result of the ROI detection procedure

As noted in [18], for images having a composite background the ROI detection is limited by the presence of interference, so a smarter choice for selecting the bounding box limits is to treshold the CDFs between 0.005 and 0.95. Therefore, only the central 90% of the normal distribution is selected. Restricting the range however negatively affect the ROI detection procedure for images were the background is empty. This problem can be solved by taking into account an additional constant, equal to the 5% of the mean edge length of the ROI for enlarging the ROI boundaries.



(a) ROI selected by considering only the central 90% of the normal distribution



(b) ROI enlarged by adding the 5% of the mean edge length of the previously ROI

Figure 3.8: Improving the ROI detection procedure

One of the innovative features of the WGE technique is that, once the ROI is correctly determined, it is possible to compute a coarse estimation of the the relative position and line of sight even before the pose determination subsystem. By defining the diagonal length of the ROI, l_{ROI}, as:

$$l_{ROI} = \sqrt{{ROI_{width}}^2 + {ROI_{height}}^2}\,, \tag{3.3}$$

a coarse estimation of the range to the target S/C from the origin of the camera frame is given by :

$$\|t_C\|_2 = \frac{((f_x+f_y)/2)L_C}{l_{ROI}}\,, \tag{3.4}$$

were L_C denotes the diagonal characteristic length of the S/C 3-D model. By knowing the coordinates of the center of the image, $[C_x, C_y]$ and the coordinates of the ROI center, $[B_x, B_y]$, it is possible to compute the azimuth and elevation angles (α, β) from the origin of the camera frame, C, to the origin of the body-fixed coordinate system, B :

$$\alpha = \arctan\frac{B_x - C_x}{f_x}\,, \tag{3.5}$$

$$\beta = \arctan\frac{B_y - C_y}{f_y}\,. \tag{3.6}$$

Once (α, β) are known, the coarse relative position solution is given by :

$$t_C = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \|t_C\|_2 \end{bmatrix}\,. \tag{3.7}$$

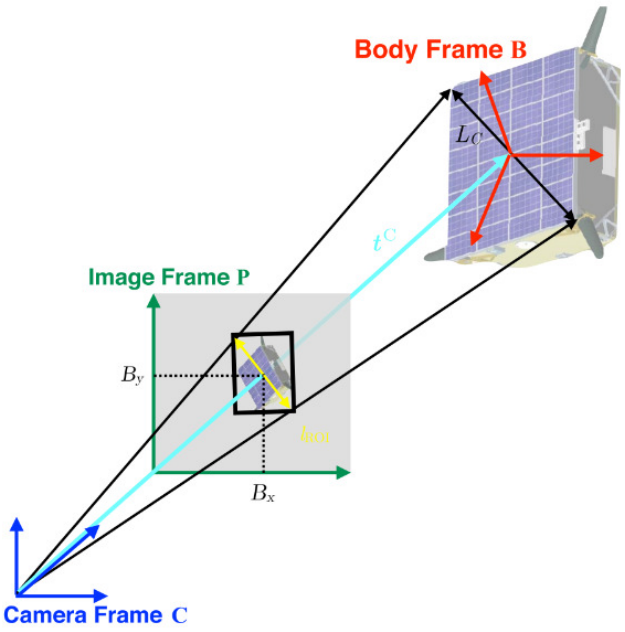


Figure 3.9: Calculation of a coarse relative position solution using the WGE technique [38]

In order to extract the features from the input image, a first edge detection process is applied to the tresholded image by means of the Hough transform. As briefly introduced in section 3.1.3, when applied to the problem of detecting straight lines into an image, the Hough transform performs a shape identification through a voting procedure in the paramter space (ρ, θ). When computing the Hough transform with MATLAB some hyperparameters are required, which for instance are :

- the desired resolution of ρ and θ,
- the maximum number of peaks to identify in the Hough transform matrix,
- the expected minimum length of the the line segment, $L_{min,Hough}$,
- procedure : the expected minimum length of the the line segment, λ_{Hough} and the maximum gap between two points to be considered in the same line segment, λ_{Hough}.

The geometrical constraints imposed to the Hough transform are dictated by the choice of $L_{min,Hough}$ and λ_{Hough}. Fixing a value of ($L_{min,Hough}$, λ_{Hough}) for all the images will result into the inability of the algorithm to adapt to different distances between the camera and the target S/C. In fact, as the camera moves closer and closer to the target S/C, the latter will fill larger and larger portions of the image, and the expected lengths of its edges in the image plane are expected to grow, so a recomputation of the geometrical hyperparameters would be needed. However, it is reasonable to hypothesize that the size of the detected ROI bounding box will grow too, so, in [38] is proposed to scale the geometrical hyperparameters by using the information about the ROI diagonal length :

$$L_{min,Hough} = \kappa_1 l_{ROI} \qquad \lambda_{Hough} = \kappa_2 l_{ROI} \, ,$$

were κ_1 and κ_2 are two scalars which can be empirically estimated by performing several test simulations on ground.

The idea proposed in [38] is to tune the values of κ_1 and κ_2 in order to only extract short line segments from the tresholded image.

A second stream of features is then obtained by directly appling the Sobel operator, which has been shown in section 3.1.2, to the unfiltered, rectified image, using the MATLAB function edge. The Sobel operator is applied by imposing an intensity treshold equal to 0.04. As proposed in [18], in order to eliminate the pixel chunks belonging to smaller reflective elements from the output of edge, it is possible to filter the output by using the MATLAB function bwareaopen. Once defined a minimum pixel number P, bwareaopen removes all connected components (objects) that have fewer than P pixels from the input binary image. In contrast with what done in [18], which sets $P = 10$ for all images, in this work

the P value is computed adaptively for each image exploiting the information about the ROI diagonal length :

$$P = \lfloor \eta \; l_{ROI}$$

where $\lfloor$ is the commonly used symbol for the **ceil** operator and η is a multiplicative constants which can be tuned empirically. After that, the Hough transform is applied again on the output produced by the Sobel operator in order to extract line segments corresponding to the silhouette of the large components of the target S/C. The knowledge the ROI location, obtained through the WGE technique, is now exploited to automatically reject any line segment for which the midpoint lies outside the ROI itself. Also for computing the Sobel and Hough (S&H) stream of features, in [38] is advised to scale the geometrical hyperparameters needed to compute the Hough lines by using the information about the ROI diagonal length :

$$L_{min,Hough} = \kappa_3 l_{ROI} \qquad \lambda_{Hough} = \kappa_4 l_{ROI} \, ,$$

where again, κ_3 and κ_4 are two multiplicative constants which can be fixed in an offline phase before the mission. The idea proposed in [38] is to tune the values of κ_3 and κ_4 in order to extract large line segments from the image.

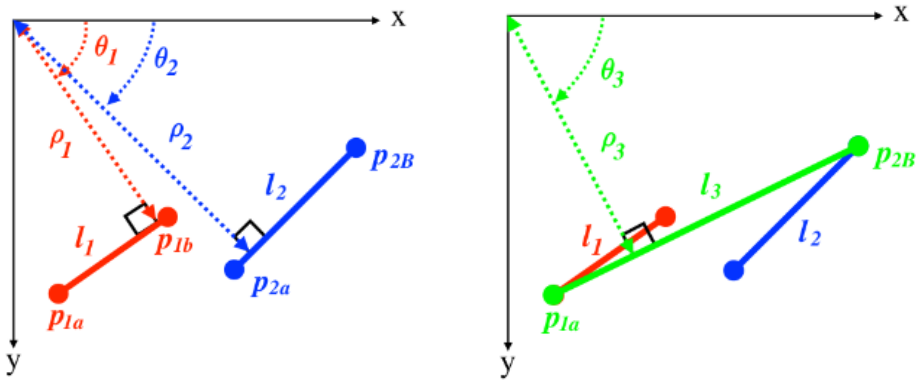
Merging Edges

The output of the Hough transform often results in multiple truncated edges, especially when applied to the image processed by the WGE. Since fragmentation can increase uncertainty and increase the computational cost of feature matching and pose solving [18], a section of the SVD architecture is dedicated to find and merge line segments which may corresponds to the same true line segment into a single line segment. To check whether two line segments can be considered similar and merged into one single edge, it is possible to define some geometrical checks on the basis of the parameters ρ and θ, which are defined for each Hough line found using the MATLAB function hough. Considering the two line segments represented in 3.10a[1], it is possible to write their polar representation in the Hough space as:

$$l_1\rho_1 = x\cos(\theta_1) + y\sin(\theta_1)\,, \tag{3.8}$$

$$l_2\rho_2 = x\cos(\theta_2) + y\sin(\theta_2)\,. \tag{3.9}$$

[1]Note that the segments are represented in the image coordinate system, where the positive x axis points to the right, the positive y axis points downward and the positive z axis points toward the direction given by $\hat{\mathbf{z}} = \frac{\hat{\mathbf{x}} \wedge \hat{\mathbf{y}}}{\|\hat{\mathbf{x}} \wedge \hat{\mathbf{y}}\|}$.



(a) Original edge (b) Output merged edge

Figure 3.10: Merging two truncated edges [38]

The conditions of similarity proposed by [38] are then:

$$|\theta_1 - \theta_2| \leqslant \theta_{tresh} \,, \tag{3.10}$$

$$|\rho_1 - \rho_2| \leqslant \rho_{tresh} \,, \tag{3.11}$$

where θ_{tresh} and ρ_{tresh} are set to be equal to the resolution of θ and ρ in the Hough space. In [18] is proposed to scale the value of ρ_{tresh} using the information about the ROI diagonal length in order to improve the identification of the spacecraft true edges :

$$\rho_{tresh} = \nu d_{ROI} \,, \tag{3.12}$$

where ν is a multiplicative constant which can be set to be equal to the resolution of ρ in the Hough space. In this work, this second approach has been preferred. Furthermore, it is imposed that the Euclidean distance between the farthest pair of endpoints of the two line segments must be less than d_{tresh} :

$$d_{p_{1a}-p_{2b}} \leqslant d_{tresh} \,, \tag{3.13}$$

where d_{tresh} is computed for each image as half of the mean length of the detected segments :

$$d_{tresh} = \frac{1}{2}\overline{(l_1, l_2, \ldots, l_n)} \,. \tag{3.14}$$

If the similarity conditions are met, then the two line segments are replaced with the line l_3, which is the line segment which joins the two farthest endpoints of l_1 and l_2 :

$$l_3\rho_3 = x\cos(\theta_3) + y\sin(\theta_3) \,. \tag{3.15}$$

The Hough parameters which defines l_3 can be computed as follows[2]. First, the angle between the positive x-axis and the joined segment is retrieved[3] :

$$\alpha = 90° - \arctan\left(\frac{p_{1a,x} - p_{2b,x}}{p_{1a,y} - p_{2b,y}}\right), \tag{3.16}$$

from which it follows the value of θ_3

$$\theta_3 = \alpha - 90° . \tag{3.17}$$

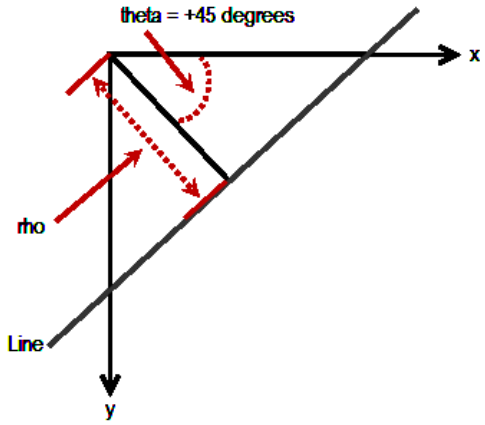


Figure 3.11: ρ and θ diagram

Now, to find ρ_3 it is sufficient to compute the intersection between the line which passes trough the origin and is also perpendicular to the joined segment :

$$m = \tan(\alpha) \tag{3.18}$$

$$x = \frac{m(p_{1a,x} - p_{1a,y})}{1/m + m}, \tag{3.19}$$

$$y = -\frac{1}{m}x, \tag{3.20}$$

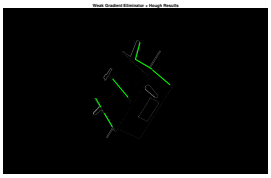
$$\rho_3 = x\cos(\theta_3) + y\sin(\theta_3). \tag{3.21}$$

For what concerns the S&H stream instead, the edge function is applied by using a treshold of 0.4. After that, the Hough transform is applied, and a dedicated function removes all the detected lines which midpoints are located outside the

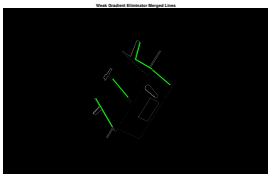
[2]Note that the computation is done in the image coordinate system previously defined

[3]Note that it is necessary to subtract 90° from the computed angle to take into account that in the image coordinate system negative the quadrant is the upper while the positive one is the lower

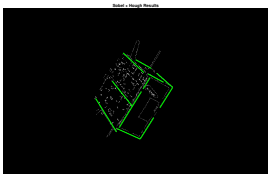
ROI. In figures 3.12, 3.13, 3.14 are presented some preliminary results of the edge merging procedure applied to images produced using the toolbox presented in 2. The resolution of ρ and θ imposed to the Hough transform are set as $0.001 d_{ROI}$ px, 5° over the range $[-90\ 89]$ with a maximum number of 5 peaks over a thresold of 0.1 for the WGE stream and $0.001 d_{ROI}$ px, 0.1° over the range $[-90\ 89]$ with a maximum number of 10 peaks over a threshold of 0.1 for the S&H stream.



(a) WGE output



(b) WGE output after merging truncated edges



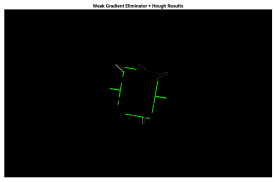
(c) S&H output

Figure 3.12: Results obtained applying edge detection and Hough transform on the two streams (1)

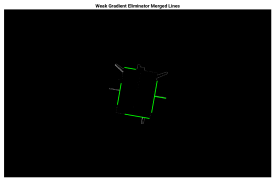
Merge Streams

The final step of the feature detection procedure consist into merging the two streams of features obtained by using both the WGE and S&H. The aim of combining the two different streams is to identify different elements of the S/C silhouette. As stated in [38], the uniqueness check archived in this phase resolves the issue of detecting repeated edges as encountered in previous works. In contrast with what proposed in [38], which suggests to detect and merge pairs of close and similar line segments separately for the two streams and merge them after, here this job is accomplished in one step. The line segments of both streams are collected together and scanned by a loop which decides what to do with the current pair being analyzed on the basis of some geometrical conditions which are set. For example, pairs of close and similar line segments are detected and merged using conditions similar to the one imposed in section 3.2.2:

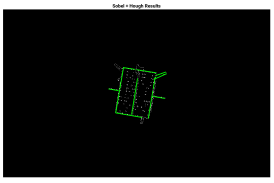
$$|\theta_1 - \theta_2| \leqslant \tilde{\theta}_{tresh}\ , \tag{3.22}$$



(a) WGE output

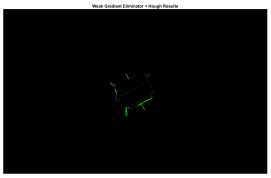


(b) WGE output after merging truncated edges

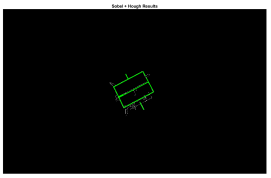


(c) S&H output

Figure 3.13: Results obtained applying edge detection and Hough transform on the two streams (2)



(a) WGE output



(b) S&H output

Figure 3.14: Case where no merging was required

$$|\rho_1 - \rho_2| \leqslant \tilde{\rho}_{tresh}\,, \tag{3.23}$$

where $\tilde{\rho}_{tresh}$ can be expressed as a function of the ROI diagonal lenght trough the multiplicative constant $\tilde{\nu}$ [18]:

$$\tilde{\rho}_{tresh} = \tilde{\nu} d_{ROI}\,. \tag{3.24}$$

Furthermore, it is imposed that the Euclidean distance between the midpoints must be less than half the length of the shorter line. With reference to figure 3.10 :

$$d_{p_{1b}-p_{2a}} \leqslant \tilde{d}_{tresh}\,, \tag{3.25}$$

where $\tilde{d}_{tresh}$ is computed for each pair being considered as half the length of the longer line segment :

$$\tilde{d}_{tresh} = \frac{1}{2} max(l_1, l_2)\,. \tag{3.26}$$

If the $\tilde{d}_{tresh}$ check is failed, the loop does two more checks before passing to the next pair. The first check controls if it is being considered a case were are present two long lines which intersect. The intersection can be computed by exploiting linear algebra. Considering two lines, l_1 (x_1, x_2,y_1,y_2) and l_2 (x_3, x_4,y_3,y_4) is possible to compute the following determinants :

$$dt_1 = det\left(\begin{bmatrix}1 & 1 & 1\\ x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ x_1 & x_2 & x_4\\ y_1 & y_2 & y_4\end{bmatrix}\right) \tag{3.27}$$

$$dt_2 = det\left(\begin{bmatrix}1 & 1 & 1\\ x_1 & x_3 & x_4\\ y_1 & y_3 & y_4\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ x_2 & x_3 & x_4\\ y_2 & y_3 & y_4\end{bmatrix}\right) \tag{3.28}$$

at this point, if $dt_1 \leqslant 0$ and $dt_2 \leqslant 0$ the two lines intersect, otherwise they do not. If they do intersect, then only the line segment composed by joining the farthest endpoints is retained if the following condition is met:

$$\epsilon = \frac{min(l_1, l_2)}{max(l_1, l_2)} > 0.25\,. \tag{3.29}$$

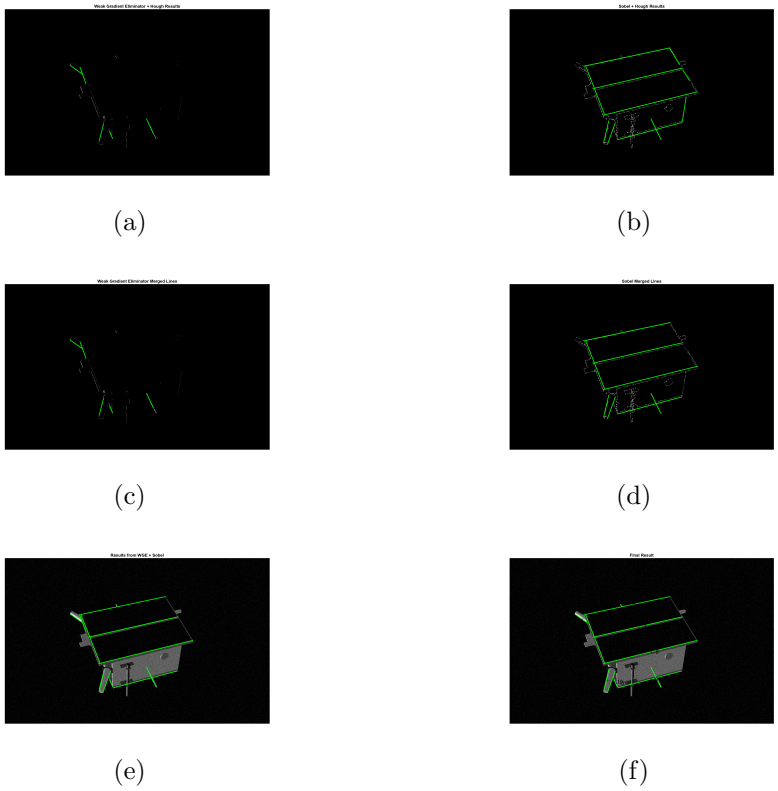
The second check instead controls if it is being considered a case were are present two long lines which are almost parallel. The parallelism is computed by exploiting the information about ρ :

$$\chi = 1 - \frac{min(\rho_1, \rho_2)}{max(\rho_1, \rho_2)} \tag{3.30}$$

if χ is less than 0.005 then the two lines are considered parallel and only the line segment composed by joining the farthest endpoints is retained if:

$$\epsilon = \frac{min(l_1, l_2)}{max(l_1, l_2)} > 0.25\,. \tag{3.31}$$

Figures 3.15, 3.16, 3.17 shows the results of the merge streams procedure when applied to the images generated with the toolbox described in chapter 2.

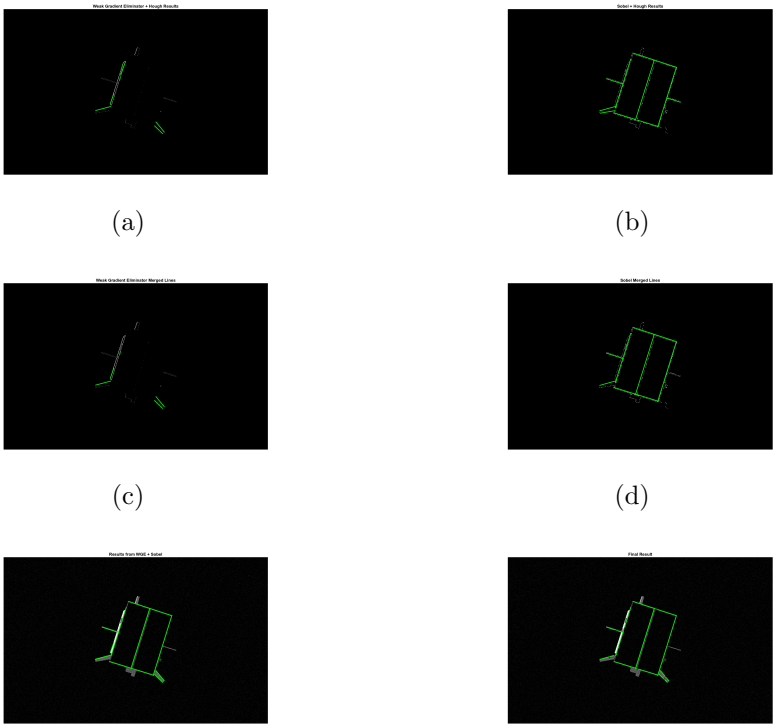


(a) (b) (c) (d) (e) (f)

Figure 3.15: Results of the merging process (1)

Feature Synthesis and Perceptual Grouping

One of the innovation introduced in the SVD algorithm is what by the original authors has been defined as "Feature Synthesys", which allows to organize the simple segments output from the merge of the WGE and S&H streams into into high-level geometrical groups called "Perceptual Groups" in order to reduce the search space of the correspondence problem. Being able to correctly define perceptual groups is of crucial importance to have a successful pose initialization. As stated in [16], to uniquely solve the P-n-P problem are required at least six correspondences between model and image points. The number of possible

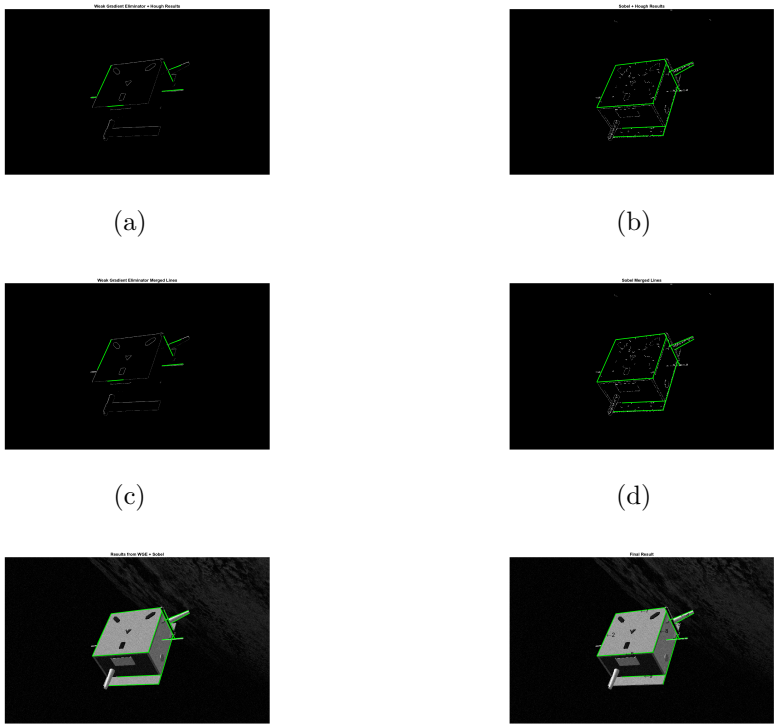


(a) (b)

(c) (d)

(e) (f)

Figure 3.16: Results of the merging process (2)



(a) (b)

(c) (d)

(e) (f)

Figure 3.17: Results of the merging process (3)

correspondences, so, the number of hypothetical pose solution given n points in the image and m points in the 3-D model therefore can be expressed as [38]:

$$\binom{m}{6}\binom{n}{6}6!\,.$$

The idea presented by Sharma *et al.* so is to reduce the solutions' search space by just using a small set of high level feature groups instead of using a large number of feature point. For example, it is sufficiently safe to say that four image points may belong to a polygonal feature such as a solar panel, then a unique solution can be found by just using four correspondences. The feature synthesis implementation proposed by Sharma *et al.* groups the features into five high-level perceptual groups, which, in order of increasing complexity are :

- parallel pairs;
- proximity pairs;
- parallel triads;
- proximity triads;
- closed polygonal tetrads;

moreover, antennas are treated as a separate feature group. As done for the merging procedure, the perceptual groups are recognized by examining several geometrical constraints.

Antennas are detected by selecting among all line segments only the one for which the following condition is met :

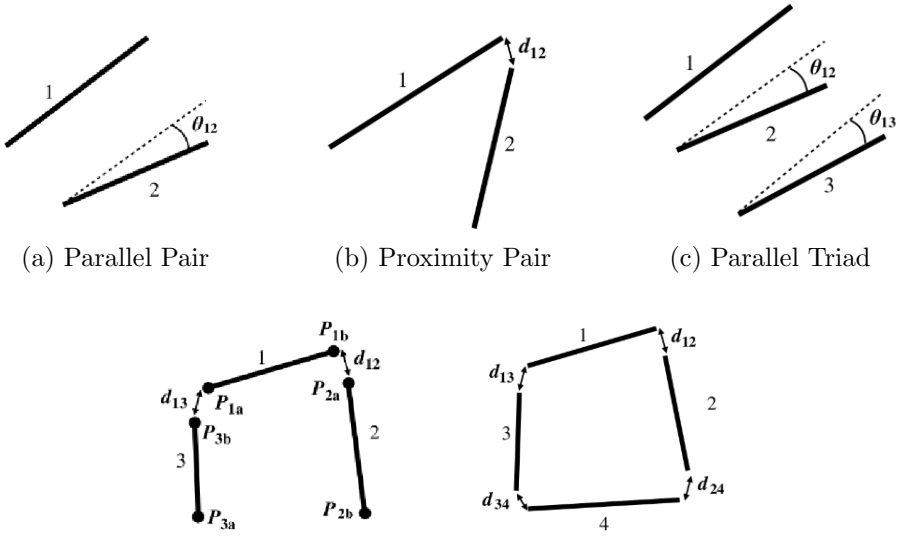
$$l_1 \leqslant \tau d_{ROI}\,, \tag{3.32}$$

where τ is a multiplicative constant, which [38] suggest to take equal to $1/3$. Before checking if a line can be inserted into an high level feature group, is always checked if it cannot be characterized as antenna. With reference to figure 3.18a, parallelism is enforced by checking the the angular difference between the two line segments:

$$\theta_{12} = |\theta_1 - \theta_2| \leqslant \theta_{max}\,, \tag{3.33}$$

However, as suggested in [18], to avoid detecting spurious parallel pairs it is useful to introduce two more geometric constraints, which are the minimum radial distance and the minimum ratio between the lines' length :

$$\rho_{12} = |\rho_1 - \rho_2| \geq \rho_{min}\,, \tag{3.34}$$



(a) Parallel Pair (b) Proximity Pair (c) Parallel Triad

(d) Proximity Triad (e) Closed Polygonal Tetrad

Figure 3.18: High-level feature groups [38]

$$\epsilon_{12} = \frac{min(l_1, l_2)}{max(l_1, l_2)} \geq \epsilon_{min} \,. \tag{3.35}$$

Proximity pairs instead are detected by checking the distance between their closest endpoints. With reference to figure 3.18b, the geometric condition checked is:

$$d_{12} \leqslant d_{max} \,, \tag{3.36}$$

where d_{max} can be adaptively computed in function of the ROI diagonal length trough a multiplicative constant:

$$d_{max} = \xi d_{ROI} \,. \tag{3.37}$$

Furthermore, in order to prevent duplicated edges to be grouped as proximal pair, in [18] is proposed a supplementary constraint on the minimum angular difference between the two line segments being considered:

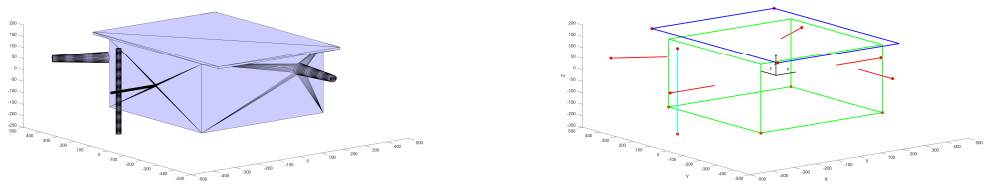
$$\theta_{12} = |\theta_1 - \theta_2| \geq \theta_{min} \,. \tag{3.38}$$

Parallel triads can be detected by selecting among all parallel pairs, the ones which shares a line segment. Similarly, proximity triads are detected by scanning

all the proximity pairs and extracting the ones which shares a line segment and satisfy the following geometric condition:

$$(P_{1a} - P_{3a})(P_{1b} - P_{3b}) > 0\,. \tag{3.39}$$

Finally, polygonal tetrads that have two segments in common are classified as closed polygonal tetrads. Similarly, the perceptual grouping is applied to a reduced CAD model of the target S/C. As proposed in [38], the CAD model introduced in 2 is reduced to only contain a low number of features. The number of features present in the wireframe reduced modules must be tuned in order to reduce all possible pose ambiguities and to reduce the number of feature correspondence hypotheses to speed up the pose determination process. The origin of the body frame is located in correspondence of the CG of the S/C.



(a) Original CAD model (b) Reduced CAD model

Figure 3.19: Full CAD model and reduced one

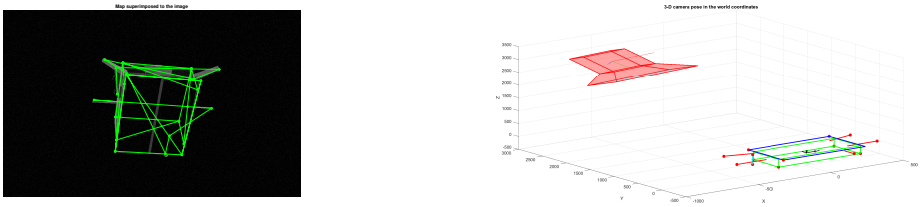
Once the CAD model is input in MATLAB and reduced, the same feature groups can be extracted from the 3-D model by applying 3-D geometry definitions instead of 2-D ones. For example, parallelism can be defined using the unit vectors constructed using end-points of the lines. Similarly, proximal pairs can be defined based on the smallest possible distance between the end points of the line segments, etc. From the 3-D model shown in figure 3.19 the following high level features have been detected :

- 18 parallel pairs;
- 16 proximity pairs;
- 12 parallel triads;
- 11 proximity triads;
- 2 closed polygonal tetrads;

in addition to the five antennas and the lateral bar which have been placed on the S/C.

3.2.3 Model Matching and Pose Determination

Once all feature groups are extracted from the 3-D model and the 2-D image,



(a) (b)

Figure 3.20: Case of succesful pose initialization

3.3 Conclusions

Chapter 4

Results

"Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program."

Linus Torvalds

Conclusions

"If you didn't get angry and mad and frustrated, that means you don't care about the end result, and are doing something wrong."

Greg Kroah-Hartman

Bibliography

[1] Blue marble. https://visibleearth.nasa.gov/. Accessed: 13-02-2021.

[2] The prisma mission. https://space.skyrocket.de/doc_sdat/prisma.htm. Accessed: 13-02-2021.

[3] F. H. Bauer, J. O. Bristow, J. R. Carpenter, J. L. Garrison, K. R. Hartman, T. Lee, A. C. Long, D. Kelbel, V. Lu, J. P. How, F. Busse, P. Axelrad, and M. Moreau. Enabling spacecraft formation flying in any earth orbit through spaceborne gps and enhanced autonomy technologies. *Space Technology*, 20(4):175–185, 2001.

[4] Christophe Bonnal, Jean-Marc Ruault, and Marie-Christine Desjean. Active debris removal: Recent progress and current trends. *Acta Astronautica*, 85:51–60, apr 2013.

[5] Kyle Alfriend Srinivas Vadali Pini Gurfil Jonathan How Louis Breger. *Spacecraft Formation Flying*. Elsevier, 2009.

[6] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, jun 1986.

[7] M. Casti, A. Bemporad, S. Fineschi, G. Capobianco, D. Loreggia, V. Noce, F. Landini, C. Thizy, D. Galano, and R. Rougeot. PROBA-3 formation-flying metrology: algorithms for the shadow position sensor system. In Nikos Karafolas, Zoran Sodnik, and Bruno Cugny, editors, *International Conference on Space Optics — ICSO 2018*. SPIE, July 2019.

[8] Xavier Clerc and Ingo Retat. Astrium vision on space debris removal. In *Proceeding of the 63rd International Astronautical Congress (IAC 2012), Napoli, Italy*, volume 15, 2012.

[9] Wikimedia Commons. The hough transform. https://upload.wikimedia.org/wikipedia/commons/e/e6/R_theta_line.GIF/. Accessed: 13-02-2021.

[10] Wikimedia Commons. Ray tracing. https://upload.wikimedia.org/wikipedia/commons/8/83/Ray_trace_diagram.svg/. Accessed: 13-02-2021.

[11] Wikipedia Contributors. Nimbostratus cloud. https://en.wikipedia.org/wiki/Nimbostratus_cloud/. Accessed: 13-02-2021.

[12] Simone D', N.A. Amico, Mathias Benn, and John L. Jørgensen. Pose estimation of an uncooperative spacecraft from actual space imagery. *International Journal of Space Science and Engineering*, 2(2):171, 2014.

[13] Jeremy Davis. Mathematical modeling of earth's magnetic field. Technical report, Virginia Tech, Blacksburg, VA 24061, 5 2014.

[14] Marco D'Errico, editor. *Distributed Space Missions for Earth System Monitoring*. Springer New York, 2013.

[15] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972.

[16] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[17] Angel Flores-Abad, Ou Ma, Khanh Pham, and Steve Ulrich. A review of space robotics technologies for on-orbit servicing. *Progress in Aerospace Sciences*, 68:1–26, jul 2014.

[18] Pierdomenico Fracchiolla. Analysis and varlidation of a vision-based pose initialization algorithm for non-cooperatie spacecrafts. Master's thesis, Università degli Studi di Padova, Padova, 2019.

[19] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):930–943, aug 2003.

[20] Jacopo Guarneri. Autonomous optical navigation and attitude estimation system tested on artificially generated images. Master's thesis, Politecnico di Milano, via La Masa 34, Milano, Italia, 7 2020.

[21] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition, 2003.

[22] P V.C. Hough. Method and means for recognizing complex patterns. 12 1962.

[23] SM Parkes I. Martin, M. Dunstan. Planet and asteroid natural scene generation utility final report. Technical Report UoD-PANGU-Final, University of Dundee, Nethergate, Dundee DD1 4HN, Regno Unito, apr 2003.

[24] J. Jensen. Hough Transform for Straight Lines. Technical report.

[25] Morgan Kaufmann. *An Introduction to Ray Tracing*. Elsevier, 1989.

[26] M. Kisantal, S. Sharma, T. Ha Park, D. Izzo, M. Märtens, and S. D'Amico. Satellite pose estimation challenge: Dataset, competition design and results. *CoRR*, abs/1911.02050, 2019.

[27] Robert D. Langley. Apollo Experience Report - The Docking System. 1972.

[28] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. 81(2):155–166, feb 2009.

[29] Paolo Lunghi. *Hazard Detection and Avoidance Systems for Autonomous Planetary Landing*. PhD thesis, Politecnico di Milano, via La Masa 34, Milano, Italia, 1 2018.

[30] F. Landis Markley and John L. Crassidis. *Fundamentals of Spacecraft Attitude Determination and Control*. Springer New York, 2014.

[31] Roberto Opromolla, Giancarmine Fasano, Giancarlo Rufino, and Michele Grassi. A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations. *Progress in Aerospace Sciences*, 93:53–72, aug 2017.

[32] S.M. Parkes, I. Martin, M. Dunstan, and D. Matthews. *Planet Surface Simulation with PANGU*.

[33] Vincenzo Pesce, Roberto Opromolla, Salvatore Sarno, Michèle Lavagna, and Michele Grassi. Autonomous relative navigation around uncooperative spacecraft based on a single camera. *Aerospace Science and Technology*, 84:1070–1080, jan 2019.

[34] Marc Pollefeys, Luc Van Gool, Maarten Vergauwen, Frank Verbiest, Kurt Cornelis, Jan Tops, and Reinhard Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, sep 2004.

[35] Aureliano Rivolta. *Guidance Navigation Control and Robotics for On Orbit Servicing*. PhD thesis, Politecnico di Milano, via La Masa 34, Milano, Italia, 1 2019.

[36] Sumant Sharma. *Pose Estimation of Uncooperative Spacecraft Using Monocular Vision and Deep Learning*. PhD thesis, Stanford University, 450 Serra Mall, Stanford, CA 94305, USA, 9 2019.

[37] Sumant Sharma and Simone D'Amico. Comparative assessment of techniques for initial pose estimation using monocular vision. *Acta Astronautica*, 123:435–445, jun 2016.

[38] Sumant Sharma, Jacopo Ventura, and Simone D'Amico. Robust model-based monocular pose initialization for noncooperative spacecraft rendezvous. *Journal of Spacecraft and Rockets*, 55(6):1414–1429, nov 2018.

[39] Jan Erick Solem. *Programming computer vision with Python*. O'Reilly, Beijing; Cambridge; Sebastopol [etc.], 2012.

[40] A. Tatsch, N. Fitz-Coy, and S. Gladun. *On-Orbit Servicing: A Brief Survey*, pages 21–23, 2006.

[41] P.H.S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, apr 2000.

[42] Jacopo Ventura. *Autonomous Proximity Operations for Noncooperative Space Target*. PhD thesis, Technical University of Munich, Arcisstrabe 21, Munich, Bavaria, 80333, Germany, 12 2016.

[43] Renato Volpe, Giovanni B. Palmerini, and Christian Circi. Preliminary analysis of visual navigation performance in close formation flying. In *2017 IEEE Aerospace Conference*. IEEE, mar 2017.

[44] Douglas Zimpfer, Peter Kachmar, and Seamus Tuohy. Autonomous rendezvous, capture and in-space assembly:past, present and future. In *1st Space Exploration Conference: Continuing the Voyage of Discovery*. American Institute of Aeronautics and Astronautics, jan 2005.

Appendix A

First appendix: Attitude and Orbit Simulator

In this appendix the kinematic and dynamic representation used for the simulation of the attitude of the target spacefrat will briefly be descripted.

A.1 Keplerian Motion

The dynamics of the CG of a rigid body is given by Newton equation of motion stating that the variation in time of the linear mentum is equal to the external forces applied to the body, with respect to an inertial reference frame. Thus :

$$\frac{d}{dt}(m\mathbf{v}) = \sum_{i=1}^{n} \mathbf{f_i} \tag{A.1}$$

where m is the mass of the body, $\mathbf{v}$ its velocity and $\mathbf{f_i}$ the forces applied to the system. Then, for a rigid body orbiting around a single massive body, the main force to take into account is the gravitational pull that can be modelled accordingly to Newton law of gravitation as follows :

$$m\ddot{\mathbf{r}} = -\frac{\mu m}{\|\mathbf{r}\|^3}\mathbf{r} + \mathbf{f} \tag{A.2}$$

where μ is the gravitational constant of the main attractor, $\mathbf{r}$ the position vector of the orbiting body computed from the main attractor CG and $\mathbf{f}$ the sum of other forces applied to the system. The underlying assumption is to consider the main attractor to be still or moving in linear constant motion, which is never the case. Such assumptions holds well enough for many problem of interest. Considering null or negligible other forces it follows that the system motion is central, thus the angular momentum is conserved.

Such quantity, for this system, is defined as follows

$$\mathbf{h} = \mathbf{r} \wedge \mathbf{r} \tag{A.3}$$

where $\mathbf{v}$ is the velocity, derivative of the position vector $\mathbf{r}$ expressed in the said reference frame.
Crossing A.2 with the angular momentum and considering that the gravitational field is conservative, it is possible to determine the position of the orbiting body in time through six parameters that represent the analytical solution of the restricted two body problem. For this research the influence of other massive bodies is not taken into account as for many of the applications here considered can be analysed considering satellites to be small bodies close to the planet.
The most used set of parameters are often referred to as Keplerian parameters, however different parametrization are possible. Of these six constant two represent the shape of the orbit, which must be a conic according to Kepler's studies and Newton's formulation, three identify the orientation of the orbit in a 3-D space and the last one links the position along the orbital with time. The shape is identified by the semi-major axis a of the conic and the eccentricity e, restricted to be between zero and one for closed orbits, being zero for circular orbits. In the reference frame with z axis aligned with angular momentum and x axis aligned with the minimum orbital distance (eccentricity vector) the position vector can be written as follows

$$\mathbf{r_{orb}} = \frac{\frac{\|h\|^2}{\mu}}{1 + ecos\theta} \begin{Bmatrix} cos\theta \\ sin\theta \\ 0 \end{Bmatrix} = \frac{a(1-e^2)}{1 + ecos\theta} \begin{Bmatrix} cos\theta \\ sin\theta \\ 0 \end{Bmatrix} \tag{A.4}$$

Then the position vector $\mathbf{r_{orb}}$ in this frame can be translated back into the inertial $\mathbf{r}$ by using three sequential rotations in the order z - x - z according to the values of the other three parameters: argument of pericenter ω , inclination i and right ascension of the ascending node Ω.

A.2 Dynamics

A.2.1 Euler Equations

The dynamics of a rigid body which is tumbling into space can be modeled by mean of the well known Euler equations, having in mind that such equation is expressed in a non-inertial reference frame attached to the body frame :

$$\omega_{\mathbf{B}} = (\mathbf{I_B})^{-1}[\mathbf{L_B} - \omega_{\mathbf{b}} \wedge (\mathbf{I_B}\omega_{\mathbf{b}}) \tag{A.5}$$

where ω_B is the angular velocity vector of the spacecraft in body frame, I_B is the inertia tensor of the spacecraft in body frame and L_B are the external torques acting on the spacecraft due to external disturbances and control torques.

A.3 Direct Cosine Matrix

The direct cosine matrix, or attitude matrix, gives the transformation of a vector from a reference frame N to another reference frame O :

$$\mathbf{r_O} = \mathbf{A_{ON}} \mathbf{r_N} \tag{A.6}$$

So, if we consider the spacecraft body frame and the inertial frame, then we can write the relation between the two frames as :

$$\mathbf{r_B} = \mathbf{A_{B/N}} \mathbf{r_N} \tag{A.7}$$

As it is demonstrated in Ref. [30], we can express the time dependence of the attitude matrix by writing the rotation from the body frame to the inertial frame as :

$$\dot{A}_{B/N} = -[\omega_{B/N}\times] A_{B/N} \tag{A.8}$$

where $[\omega_{B/N}\times]$ is the skew-symmetric cross product matrix containing the components of the angular velocity vector :

$$[\omega\times] = \begin{bmatrix} 0 & -\omega_{B/N_3} & \omega_{B/N_2} \\ \omega_{B/N_3} & 0 & -\omega_{B/N_1} \\ -\omega_{B/N_2} & \omega_{B/N_1} & 0 \end{bmatrix}$$

Thus, by integrating this relation, we can get full attitude at every iteration.

A.3.1 Environmental Disturbances

The external disturbances acting on spacecraft placed in a LEO orbit are basically four and are due to :

- Gravity Gradient
- Magnetic Field
- Solar Radiation Pressure
- Aerodynamic Drag

Details about how those torques have been mathematically modeled will be given in the following sections.

Gravity Gradient

Any non-symmetrical rigid body in a gravity field is subject to a gravity-gradient torque. If we consider a rigid spacecraft, the torque due to the gravity gradient about the spacecraft's center of mass can be modeled as :

$$\mathbf{L_{gg}} = \frac{3\ mu}{r^3}\mathbf{n} \wedge (\mathbf{I}\ \mathbf{n}) \tag{A.9}$$

where μ is the Eart's gravitational constant, $\mathbf{r}$ is the radius vector from the center of the Earth, thus $r \equiv \|\mathbf{r}\|$, $\mathbf{I}$ is the inertia tensor in body frame and $\mathbf{n}$ is the body frame representation of a nadir-pointing unit vector.
Further details about the model can be found in reference [30].

Earth's Magnetic Field

The torque generated by a magnetic dipole $\mathbf{m}$ in a magnetic field $\mathbf{B}$ is

$$\mathbf{L_{mag}} = \mathbf{m} \wedge \mathbf{B} \tag{A.10}$$

where $\mathbf{B}$ is the magnetic field in the body frame. The most basic source of a magnetic dipole is a current loop. A current of I amperes flowing in a planar loop of area A produces a dipole moment of magnitude m=IA in the direction normal to the plane of the loop and satisfying a right-hand rule. When $\mathbf{m}$ is in Am^2 and the magnetic field is specified in Tesla, Eq. A.10 gives the torque in Nm. If there are N turns of wire in the loop, the dipole moment has magnitude m=NIA (such as the case of a magnetic torquer). To model $\mathbf{B}$ either the full IGRF model or the simple dipole model can be used.
For what concerns this work, the full IGRF model truncated to the 10th order has been used. Further details about the model can be found in reference [13]

Solar Radiation Pressure

Solar radiation pressure acting on the surfaces of the spacecraft causes a disturbance torque that in general, cannot be neglected for orbits higher than 800 km, so it has been taken into account in this work. The SRP torque is zero zero when the spacecraft is in the shadow of the Earth or any other body, of course. To take into account the effect of solar radiation pressure on the spacecraft, the spacecraft's surface can be modeled as a collection of N flat plates of area S_i, outward normal $\mathbf{n_b}$ in the body coordinate frame, specular reflection coefficient ρ_s, diffuse reflection coefficient ρ_d and absorption coefficient ρ_a; those coefficients must sum to unity. For what concerns this work, only ρ_s and ρ_d have been considered, since all the absorbed radiation is emitted as thermal radiation, although not necessarily at the same time or from the same surface as its absorption. For an accurate modeling of thermal radiation we must also known the absolute temerature and the emissivity

of each surface. We can define the spacecraft-to-Sun unit vector in the spacecraft's body frame as :

$$\mathbf{s_b} = \mathbf{A_{B/N}} \mathbf{s_i} \tag{A.11}$$

where $\mathbf{A_{B/N}}$ is the attitude matrix and $\mathbf{s_i}$ is the spacecraft-to-Sun unit vector in the GCI frame. We can define the angle between the Sun vector and the normal exiting from the normal to the i-th plate as :

$$cos(\theta^i_{SRP}) = \mathbf{n}^i_B \cdot \mathbf{s_b} \tag{A.12}$$

The SRP force on the i-th plate can be expressed as :

$$\mathbf{F}_{SRP} = -P_{Sun} A_i \left[2 \left(\frac{\rho^i_d}{3} + \rho^i_s cos(\theta^i_{SRP}) \right) \mathbf{n}^i_B + (1 - \rho_s) \mathbf{s_b} \right] max(cos(\theta^i_{SRP}), 0) \tag{A.13}$$

where P_{Sun} is the solar radiation pressure. The Solar radiation pressure torque acting on the spacecraft is then given by :

$$\mathbf{L}^i_{SRP} = \sum_{i=1}^{n} \mathbf{r}_i \times \mathbf{F}^i_{SRP} \tag{A.14}$$

where $\mathbf{r}_i$ is the vector from the spacecraft center of mass to the centre of pressure of the SRP on the i-th face. In this formulation we are not considering the albedo radiation coming from the Earth and from the Moon. Further details on how the solar radiation pressure, the spacecraft-to-Sun unit vector and the eclipse condition have been modeled can be found in reference [30].

Aerodynamic Drag

Aerodynamic torque is generally computed by modeling the spacecraft as a collection of N flat plates of area A_i and outward normal unit vector $\mathbf{n_B}$ expressed in the spacecraft body-fixed coordinate system. The torque depends on the velocity of the spacecraft relative to the atmosphere, which is not simply the velocity of the spacecraft in the GCI frame, because the atmosphere is not stationary in that frame. The most common assumption is that the atmosphere co-rotates with the Earth. The relative velocity in the GCI frame is then given by :

$$\mathbf{v_{relI}} = \mathbf{v_I} + [\omega_{\oplus} \times] \mathbf{r_I} \tag{A.15}$$

where $\mathbf{r_I}$ and $\mathbf{v_I}$ are the position and velocity of the spacecraft expressed in the GCI frame. The Earth's angular velocity vector is $\omega_{\oplus} = \omega_{\oplus}[001]'$ with $\omega_{\oplus} =$

0.000 072 921 158 553 rad/s. The velocity in the body frame is the computed as :

$$\mathbf{v_{relB}} = \mathbf{A_{B/N}} \begin{bmatrix} \dot{x} + \omega_{\overset{+}{\circlearrowleft}} y \\ \dot{y} - \omega_{\overset{+}{\circlearrowleft}} x \\ \dot{x} \end{bmatrix} \tag{A.16}$$

The inclination of the i-th plate WRT the relative velocity is given by :

$$cos(\theta^i_{aero}) = \frac{\mathbf{n}^i_B \cdot \mathbf{v}_{rel\ B}}{\|\mathbf{v}_{rel}\|} \tag{A.17}$$

The aerodynamic force on i-th plate in the flat plate model is

$$\mathbf{F}^i_{aero} = -\frac{1}{2}\rho C_D \|\mathbf{v}_{rel}\| \mathbf{v}_{relB}\ S_i\ max(cos(\theta^i_{aero}), 0) \tag{A.18}$$

where ρ is the atmospheric density, and C_D is the drag coefficient. ρ can be modeled by mean of the well known exponential decaying model for the atmospheric density :

$$\rho = \rho_0 e^{(-(h-h_0)/H)} \tag{A.19}$$

where ρ_0 and h_0 are reference density and height, respectively, h is the height above the ellipsoid and H is the scale height, which is the fractional change in density with eight. The value of ρ_0, h_0 and H changes with h. The values used to perform the simulation are the one given in [30]. The actual torque due to the aerodynamic drag can be computed as :

$$\mathbf{L}^i_{aero} = \sum_{i=1}^{n} \mathbf{r}_i \times \mathbf{F}^i_{aero} \tag{A.20}$$

where n is the number of faces and $\mathbf{r}_i$ is the vector from the spacecraft center of mass to the center of pressure on the i th face.

Appendix B

Second appendix: Random Rotation Matrix Generation