POLITECNICO MILANO 1863

School of industrial and information engineering

Department of Aerospace Science and Technologies - DAER
Master of Science in Space Engineering

Monocular feature-based navigation in proximity of unknown space objects with regions of interest effective extraction

Marco Pittoni
898986

Supervisor: Prof. M. Lavagna
Cosupervisor: M. Piccinin

A.Y. 2019/2020

Sommario Le recenti evoluzioni delle missioni spaziali hanno richiesto continui sviluppi di tecnologie sempre più all'avanguardia. Obiettivi complessi come la navigazione in formazione o la manutenzione orbitale attualmente ricercati necessitano di strumenti e metodi di navigazione evoluti, in grado di garantire prestazioni elevate e sicurezza nelle operazioni.
Il campo di ricerca che si occupa specificamente di navigazione ottica è quello della computer vision. Esso nasce dalla volontà di capire meglio il sistema visivo umano e poggia le sue basi su vari ambiti tecnologici come il processo di segnali, di immagini, statistica, metodi di ottimizzazione etc. Uno dei processi più comuni con cui la navigazione ottica viene condotta è spesso definito V-SLAM, ad indicare la localizzazione di un robot in un ambiente sconosciuto e la mappatura di quest'ultimo. Inoltre grazie all'introduzione di nuove tecnologie nell'ambito dell'AI, machine learning in particolare, nuovi metodi sono emersi in questi ultimi anni ed un loro possibile contributo viene investigato in questo lavoro.
La seguente tesi esplora quindi il processo V-SLAM adottando una architettura monoculare applicata al settore spaziale. Un software di generazione immagini è stato utilizzato per ottenere immagini di un satellite non noto e non cooperativo. Un metodo per la navigazione relativa basato su features è stato implementato e test preliminari sono stati effettuati con risultati soddisfacenti. Inoltre il problema della stima della Region of Interest in immagini affette dalla presenza della Terra è stato affrontato utilizzando strumenti di intelligenza artificiale. Una Convolutional Neural Network è stata addestrata su un nuovo dataset contenente immagini e informazioni dell'assetto e posizione relativi di due satelliti. La CNN offre performance positive e, testata anche su immagini di satelliti mai visti prima, conferma la possibilità di queste network di generalizzare ad oggetti sconosciuti.

Keywords: Stima di posizione e assetto, Visione monoculare, Convolutional Neural Networks, Operazioni di prossimità, V-SLAM

Abstract. The recent evolution of space missions has required continuous development of cutting-edge technologies. Complex objectives such as formation flying or orbital maintenance currently sought-after necessitate of advanced navigation tools and methods, capable of guaranteeing performance and safety during operations.
The research field that deals with optical navigation is the one of computer vision. It was born from the desire to better understand the human visual system and it is built on various technological fields such as signal and image processing, statistics, optimization methods etc. One of the most common processes by which visual navigation is performed is often referred to as V-SLAM, designating the estimation of the location of a robot in an unknown environment and the mapping of the latter. Furthermore, thanks to the development of new AI technologies, machine learning in particular, new methods have been introduced in recent years and their possible contribution is investigated in this work.
The following thesis explores the V-SLAM process by adopting a monocular architecture applied to the space sector. An image generator software has been used to obtain images of an unknown and uncooperative satellite. A feature based method for relative navigation has been implemented and preliminary tests have been performed with satisfying results. Moreover, the problem of estimating the Region of Interest in images affected by Earth presence is addressed by means of artificial intelligence tools. A Convolutional Neural Network has been trained on a new data set containing satellite images and information of relative attitude and position. The CNN offers positive performance and, tested also on images of never seen before satellites, confirms the capability of these networks to generalize to previously unknown objects.

Keywords: Pose Estimation, Monocular vision, Convolutional Neural Networks, Proximity Operations, V-SLAM

Table of Contents

1 Introduction 1
1.1 Relative navigation in space 2
1.2 Visual relative navigation methods 4
1.3 Machine Learning 7
1.4 Rendering for space applications 9
1.5 Thesis overview and contributions 10
2 Relative dynamics 12
2.1 Reference Frames 12
2.2 System of equations for nonlinear relative dynamics 14
2.3 System of equations for coupled translational-rotational relative dynamics 16
2.3.1 Relative rotational model 16
2.3.2 Relative translational model 18
2.4 Numerical Simulations 20
3 Projective geometry 25
3.1 Geometric Transformations 26
3.1.1 Translation 26
3.1.2 Euler transformation 28
3.1.3 Similarity transformation 28
3.1.4 Affine transformation 29
3.1.5 Projective transformation 29
3.2 Camera model and image formation 31
3.3 Ray casting 34
3.4 Analogies to photography and optics 35
4 Computer Vision 38
4.1 Feature Detection 38
4.2 Feature Matching and Tracking 40
4.2.1 Feature Matching 40
4.2.2 Feature Tracking 42
4.3 Epipolar geometry 44
4.4 Fundamental and essential matrix estimation 45
4.5 Extrinsic parameters from essential matrix 47
4.6 Triangulation 48
4.7 Perspective n-Point Problem 50

4.8 Bundle Adjustment 51
5 Deep Learning 53
5.1 Fully Connected Neural Networks 54
5.2 Backpropagation 56
5.3 Convolutional Neural Networks 61
6 Architecture for relative spacecraft navigation 64
6.1 V-SLAM for Visual Navigation 65
6.2 CNN for ROI Estimation 66
6.3 Modeling of realistic spacecraft scenarios 67
6.3.1 Image generation algorithm 68
6.3.2 Render examples 71
7 V-SLAM Implementation and results 74
7.1 Preliminary simulation implementation 74
7.2 Simulation implementation 78
7.2.1 Map initialization test 78
7.3 Results 81
7.3.1 Test case 1 82
7.3.2 Test case 2 86
7.3.3 Test case 3 89
7.3.4 Test case 4 92
7.3.5 Test case 5 95
8 CNN ROI Design and results 97
8.1 Dataset generation and description 97
8.2 CNN Training and Architecture Tradeoff 100
8.3 CNN Analysis and results for design scenario 103
8.4 CNN Testing 110
8.4.1 Off-nominal scenarios 110
8.4.2 Noise and Blur 116
8.5 V-SLAM with CNN ROI results 118
9 Conclusion 120
9.1 Future Work 121
9.1.1 V-SLAM method 121
9.1.2 CNN ROI estimation 121
A Singular value decomposition 124
A.1 SVD for least squares problems 124
A.2 SVD for overdetermined systems 125
B Unconstrained Numerical Optimization 126
B.1 Newton method 126
B.2 Line search methods 127

B.3 Trust region methods 128
C Nonlinear least square methods 130
C.1 Gauss-Newton method 131
C.2 Levenberg-Marquardt method 131
D Vespa 3D model development 132
E CNN Testing dataset overview 136
E.1 Dragon, Earth & Space 137
E.2 Dragon without solar array, Earth & Space 139
E.3 Vespa, Earth & Space 141
E.4 MarCO, Earth & Space 143
E.5 EnviSat, Earth & Space 145
E.6 MarCO, close Moon 147
E.7 Dragon, close Moon 149
E.8 Dragon, far Moon 151
E.9 Dragon, far Comet 153
E.10 Dragon, close Comet 155
E.11 Dragon, Earth & Space, Noise 157
E.12 Dragon, Earth & Space, Blur 158
References 162

List of Figures

1.1 Rosetta instruments overview [1] 3
1.2 ORB-SLAM pipeline [2] 6
2.1 Reference frames 13
2.2 Chief and Deputy spacecraft 14
2.3 Chief and Deputy spacecraft with their reference frames 16
2.4 Chief and Deputy spacecraft 19
2.5 Orbital dynamics 21
2.6 Attitude and position propagation 22
2.7 Comparison of different feature positions 22
2.8 Comparison of deputy CM position with CW and NL models 23
2.9 Differences of deputy CM position, CW and NL models for $e_c = 0$ & $e_c = 0.05$ 23
3.1 Parallel railway tracks converging at infinity 25
3.2 Translation example 26
3.3 Scaled homogeneous vectors to inhomogeneous one 27
3.4 Euler transformation example 28
3.5 Similarity transformation example 28
3.6 Affinity transformation example 29
3.7 Projective transformation examples, infinite horizon line in blue 30
3.8 Pinhole camera model [3] 31
3.9 Camera translation and rotation [3] 32
3.10 Camera model examples 33
3.11 Dragon spacecraft rendered images 34
3.12 Ray triangle intersection example 35
3.13 Converging lens representation [4] 36
3.14 Camera field of view 37
4.1 Example of minimum eigenvalue detector 39
4.2 Statistics for feature detection 39
4.3 Example of feature matching 41
4.4 Statistics for feature matching 42
4.5 Example of KLT tracking with Min. Eig. feature points 43
4.6 Example of KLT traking 44
4.7 Two view geometry representation 44
4.8 Camera ambiguity in reconstruction [5] 48

5.1 Biological and artificial neuron [6] . . . 53
5.2 Example of neuron connections . . . 53
5.3 Fully connected ANN example . . . 55
5.4 Notation for ANN . . . 57
5.5 LeNet-5 architecture [7] . . . 61
5.6 Example of convolution without kernel flipping [8] . . . 63
6.1 Relative spacecraft navigation architecture . . . 64
6.2 Typical V-SLAM pipeline . . . 65
6.3 Example of false positive tracks . . . 67
6.4 Examples of 3D map generation with and without CNN ROI . . . 67
6.5 Ray tracing basic principle [9] . . . 68
6.6 Blender and Matlab outputs example . . . 70
6.7 Render example of Dragon spacecraft with Earth in background . . . 71
6.8 Render examples of Dragon spacecraft with Earth in background . . . 72
6.9 Comparison between rendered image on top and real spacecraft image [10] . . . 73
7.1 Reference trajectory plot . . . 74
7.2 Feature detection examples . . . 75
7.3 Feature tracking examples . . . 76
7.4 Results for preliminary simulation . . . 77
7.5 Angle between views and uncertainty on 3D points [3] . . . 79
7.6 Results for 3D points accuracy test . . . 80
7.7 Reference and estimated trajectories . . . 82
7.8 Examples of feature detection and tracking . . . 83
7.9 Actual and estimated 3D points . . . 83
7.10 Test case 1 results . . . 84
7.11 Translation error for different feature detectors . . . 85
7.12 Reference and estimated trajectories . . . 86
7.13 Examples of feature detection and tracking . . . 87
7.14 Actual and estimated 3D points . . . 87
7.15 Test case 2 results . . . 88
7.16 Reference and estimated trajectories . . . 89
7.17 Examples of feature detection and tracking . . . 90
7.18 Actual and estimated 3D points . . . 90
7.19 Test case 3 results . . . 91
7.20 Reference and estimated trajectories . . . 92
7.21 Examples of feature detection and tracking . . . 93
7.22 Actual and estimated 3D points . . . 93
7.23 Test case 4 results . . . 94

7.24 Reference and estimated trajectories . . . 95
7.25 Actual and estimated 3D points . . . 96
7.26 Test case 5 results . . . 96
8.1 Dataset generation environment . . . 97
8.2 Examples of dataset images . . . 98
8.3 Bounding box generation process . . . 99
8.4 CNN training report . . . 102
8.5 CNN final architecture . . . 102
8.6 CNN execution times . . . 103
8.7 CNN Results examples . . . 104
8.8 Example of filtering after first convolutional layer . . . 105
8.9 Example of filtering after second convolutional layer . . . 106
8.10 Example of filtering after third convolutional layer . . . 106
8.11 Example of filtering after ninth convolutional layer . . . 106
8.12 Example of filtering after ninth convolutional layer . . . 107
8.13 Example of learned patters in the seventh convolutional layer . . . 107
8.14 CNN Results examples, previously unseen spacecraft [11] . . . 108
8.15 CNN Results examples, previously unseen cubesats [11] . . . 109
8.16 Dragon error data and normal distribution fitting . . . 113
8.17 Example of random test images, Earth cases, estimation in yellow . . 114
8.18 Example of random test images, Moon and Rosetta comet, estimation in yellow . . . 115
8.19 Noise tests, Dragon with Earth in background . . . 116
8.20 Blur tests, Dragon with Earth in background . . . 117
8.21 Frame examples, with and without CNN ROI estimation . . . 118
8.22 Results for V-SLAM algorithm with and without CNN ROI . . . 119
A.1 SVD illustration [12] . . . 124
D.1 Vespa, source images . . . 133
D.2 Examples of Vespa initial renderings . . . 134
D.3 Examples of Vespa renderings . . . 135
E.1 Dragon, Earth & Space, 1 . . . 137
E.2 Dragon, Earth & Space, 2 . . . 138
E.3 Dragon without solar array, Earth & Space, 1 . . . 139
E.4 Dragon without solar array, Earth & Space, 2 . . . 140
E.5 Vespa, Earth & Space, 1 . . . 141
E.6 Vespa, Earth & Space, 2 . . . 142
E.7 MarCO, Earth & Space, 1 . . . 143
E.8 MarCO, Earth & Space, 2 . . . 144
E.9 EnviSat, Earth & Space, 1 . . . 145

E.10 EnviSat, Earth & Space, 2 146
E.11 MarCO, close Moon, 1 147
E.12 MarCO, close Moon, 2 148
E.13 Dragon, close Moon, 1 149
E.14 Dragon, close Moon, 2 150
E.15 Dragon, far Moon, 1 151
E.16 Dragon, far Moon, 2 152
E.17 Dragon, far Rosetta comet, 1 153
E.18 Dragon, far Rosetta comet, 2 154
E.19 Dragon, close Rosetta comet, 1 155
E.20 Dragon, close Rosetta comet, 2 156
E.21 Dragon, Earth & Space, Noise 157
E.22 Dragon, Earth & Space, Blur 158

List of Tables

1 Orbital simulation data 20
2 Relative translation simulation data 21
3 Relative attitude simulation data 21
4 CNN architecture tradeoff 101
5 List of test datasets 111
6 Statistic data for tests 111
7 Insights on RMSE variations 112
8 Parameters of the normal distributions for the tests 113
9 Noise tests 116
10 Blur tests 117

List of Acronyms

V-SLAM	Visual Simultaneous Localization And Mapping
SfM	Structure from Motion
VO	Visual Odometry
fps	Frames per second
ROI	Region Of Interest
CPU	Central Processing Unit
GPU	Graphic Processing Unit
RT	Ray Tracing
ANN	Artificial Neural Network
FC	Fully Connected
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
DL	Deep Learning

1 Introduction

Relative navigation between two space objects can be expressed as the problem of estimating the relative parameters, such as the position and orientation, of one object with respect to the other. Understanding the surrounding environment and how to navigate it is a vital process for any entity, it being a satellite imaging space debris, a rover exploring the surface of an unknown planet or a bird migrating to warmer regions. Satellites of course are not an exception and they too need some form of localization procedures. Even though this can happen remotely, the need of having on-board awareness and autonomy is an interesting and emerging challenge especially when time critical decision must be made, as during formation flying or proximity operations, and appropriate solutions should be sought.

Two satellites orbiting together can be categorized into two cooperative objects, which can communicate to each other, or two uncooperative and unrelated objects. In the first case, where the two objects are cooperative, a telecommunication link can be established between the two to share the appropriate pose information. In this case, each satellite can be tasked to reconstruct its state and share it. Laser or camera based sensors are also a common choice and a mandatory one when one object might not be cooperative or known. A LiDAR, Light Detection and Ranging, is essentially a radar that exploits light, a laser, to understand the range of a target object. Through spinning and tilting of the laser beam, a full scan of the environment can be performed too. These solutions are generally very expensive but offer high degree of precision and can be applied to relative navigation with respect to another satellite or planetary body [13]. Cameras on the other hand provide simple and low cost solutions with a penalty in the accuracy of the results. By implementing vision-based sensors on board, resources that would be needed for a LiDAR solution could then be redirected towards mission objectives. A camera can be used in a mono configuration, which could be very appealing for cubesats application, or in stereo configuration in order to retrieve the depth of the scene. A camera and laser beam can be exploited to indirectly obtain the scene depth too.

Autonomy is another aspect of space systems which can grant major improvement in the current generation of space missions and its effect would be beneficial also to many future missions. An autonomous system must be self-directed towards its goals and self-sufficient, being able to operate without any external inputs [14]. In the space sector an increase in autonomous capabilities can affect many different fields that span from spacecraft formation flying up to rover ground operations. Close relative navigation and proximity operations must be performed safely and precise poses knowledge is fundamental. Docking, satellite inspection, refueling and maintenance are all possible applications that could benefit from this novel approach. Moreover relative navigation of unknown and uncooperative objects can be applied to asteroid navigation and mining or space debris removal, which are emerging fields in recent years. Introduction of autonomy in space system would provide a great boost to its capabilities, however its implementation should be handled with high attention as the risk involved in this process must not be underestimated.

1.1 Relative navigation in space

In this section a brief review of important past space missions developed with different forms of relative navigation is provided. The increased interest in autonomous proximity operations has indeed given birth to many technology demonstration missions. At first missions with cooperative objects are discussed, followed by spacecraft operating in unknown environments.

The first mission for demonstration of rendezvous and docking capabilities, the Japanese ETS-7, was launched in 1997. Proximity operations were carried out automatically and with remote control through a camera-aided robotic arm [15] and in orbit refuelling was also investigated successfully. NASA launched in 2005 the XSS-11 spacecraft, meant to conduct proximity operations with its carrier, the second stage of the Minotaur I rocket. The XSS-11 was equipped with both a LiDAR and a camera to intercept that stage and also other expended stages close to its orbit [16]. The DART spacecraft, also launched in 2005, was intended to perform a 24 hours mission with no human intervention which included several rendezvous and proximity navigation, but unfortunately ended with a soft collision with its target due to spacecraft malfunctions [17]. Moreover docking with the international space station, the ISS, has become an important problem to solve and in 2009 a LiDAR based solution has been introduced by the Canadian Space Agency and NASA. An all-in-one solution, called TriDAR (Triangulation & Light Detection and Ranging Automated Rendezvous & Docking) is an automated rendezvous and docking sensor composed by a LiDAR and a thermal imager. The shape of the target is assumed to be known and is compared to the estimated one to retrieve important parameters like the relative range and position between the spacecraft and the ISS, up to 1 km distance. Three Space Shuttle missions and Cygnus spacecraft adopted this technology [18]. Also Dragon spacecraft uses a similar sensor asset called DragonEye. Relative GPS navigation is used up to 750 m, after which the ISS is tracked using the LiDAR, DragonEye, and thermal cameras [19].

For what concerns relative navigation around unknown space objects, the main and most recent missions include OSIRIS-REx, Hayabusa 2 and Rosetta. In order to have a broader view of cameras characteristics in these spacecraft, few parameters

are listed too.
Rosetta consisted of an orbiter and a lander for the first ever study of a comet, 67P/Churyumov-Gerasimenko [20]. Launched in 2004, Rosetta was equipped with one redundant navigation camera called NAVCAM [21] for AOCS only and a pair of cameras for optical, spectroscopic and infrared imaging called OSIRIS [22]. The NavCam provided imaging data for optical navigation of the spacecraft around the comet. It had a CCD sensor resolution of 1024×1024 pixels and the optics had an effective focal length of 152.5 mm with an aperture of 30 or 70 mm mode-dependent.

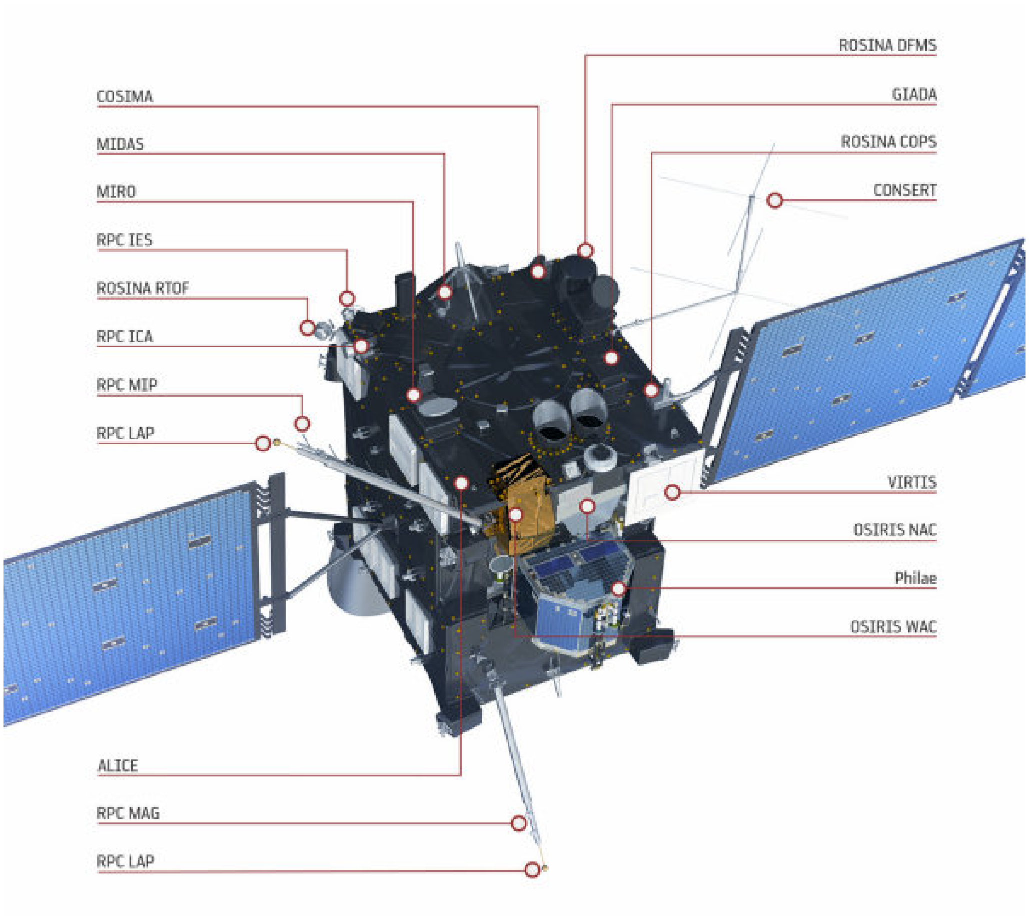


Fig. 1.1: Rosetta instruments overview [1]

OSIRIS was a scientific payload instead that comprised two cameras: a high resolution Narrow Angle Camera, NAC, and a Wide Angle Camera, WAC. Both of them had CCD detectors of 2048 x 2048 pixels. The NAC was developed to produce high resolution images of the comet, its focal length was of 717 mm and the aperture was of 90 mm, resulting in a moderately fast system with f/8. The WAC instead was mainly used to study the intensity of gas emissions and dust-scattered sunlight and it had a focal ratio of f/5.6 with a focal length of around 140 mm.

A more recent mission similar to Rosetta is Hayabusa 2. It is an asteroid sample-return mission by JAXA (Japanese Space Exploration Agency) developed to study Asteroid 1999 JU3 and is provided with an optical navigation system which includes three Optical Navigation Cameras and a LiDAR [23]. There are two wide angle cameras onboard the spacecraft, called ONC-W1 and ONC-W2, and a telescopic one called ONC-T. All of them use CCD detectors of size 1024 by 1024 pixels and the telescopic camera has a focal length of 121 millimeters and an aperture diameter of 15 millimeters.

Finally OSIRIS-REx, launched in 2016, is a sample and return mission from NASA targeting the asteroid Bennu. A redundant LiDAR is used for ranging to the surface and two NavCams are used to support the navigation during mission operations and in particular during the sampling operation. NavCam images track star-fields and landmarks on Bennu to determine the spacecraft position [24].

As stated before, these missions generally use a mix of laser-based and vision-based sensor for relative navigation. Specifically, visual navigation for Rosetta was essential during the whole the mission duration. In the far approach navigation, images coming from the cameras were used to understand the centroid position of the comet and perform various optical measurements, including the comet rotational period [25]. Then in the following phases of comet characterization and mapping a landmark-based visual navigation was performed on ground. This process, manual at first and automatic later on, allowed to predict the spacecraft position and orientation along with a very detailed shape model of the comet [26]. Limitations however present in the system, such as the amount of data that can be sent due to the link budget, could be solved by researching and developing more autonomous navigation systems which could also expand the fan of possibilities for proximity operations. One exciting mission that is due to launch in 2024 and will extensively use visual techniques for navigation is the HERA mission. It will visit in 2026 the Didymos pair of near-Earth asteroids after NASA DART mission and will be equipped with two cameras, a thermal imager and a LiDAR as a baseline [27]. Line-of-sight navigation will be performed at far range from thermal and visual images processing, while feature based navigation will be added when closer to the asteroids system [28].

1.2 Visual relative navigation methods

In this next section vision based navigation methods are discussed. Visual navigation methods have their primary building blocks in computer vision, which is an interdisciplinary field tasked to understand and extract useful information starting from images only. This has proven to be a very complex challenge and it still is, as O. Faugeras wrote: “making a computer see was something that leading experts in the field of Artificial Intelligence thought to be at the level of difficulty of a summer student's project back in the sixties. Forty years later the task is still unsolved and seems formidable" [3]. Closely connected to the understanding of how the human visual system works, computer vision is now a continuous developing field with many different practical applications.

The problem of computing the trajectory and 3D structure of a scene is often referred to as V-SLAM, standing for Visual Simultaneous Localization and Mapping. It is a subcategory of the SfM problem, the structure from motion problem, and there are mainly two branches that attempt at solving the online, real time version of V-SLAM. One category exploits the use of filtering techniques and the most popular filters lie in the Kalman filter family (KF, EKF, ...). They constitute a tool that continuously predicts and updates the state of a dynamical system in time recursively. They are very light in terms of memory and fast, making them ideal for real time problems and embedded systems for space applications [29] [30].
On the other hand, keyframe based methods derive their popularity on the optimisation approach of global bundle adjustment. Bundle adjustment is an optimization algorithm targeted for structure and motion refinement and is often too expensive to be executed real time. For this reason a windowed version of it might be used on a subset of the available frames or keyframe only optimization can be exploited. Keyframes are selected from the available frames and are mainly meant to reduce the computational load and enable real time performances. A very popular algorithm in this category is the ORB-SLAM one [2], which consists in a complex workflow capable of automatic map initialization, loop detection and closure, re-localization. Moreover the tracking and local mapping steps are split in parallel threads, enhancing even further the time execution performances. An investigation between these two categories has been performed by Strasdat et al. in [31], which concluded that keyframe-based techniques with bundle adjustment are superior with respect to filtering ones for the same computational cost while filter techniques might be beneficial if a small processing budget is available. This last consideration is an usual condition for the space domain, however as the sector continues to grow, autonomous systems are introduced and private companies start to emerge, the problem of development of new and more powerful processors for space applications will need to be tackled.

With respect to V-SLAM for space applications, solutions include the use of ORB-SLAM for spacecraft non-cooperative rendezvous image sequences, such as in [32]. ORB-SLAM performs accurately even in a very different scenario with respect to the robotic field ones. The authors however do point out that pose reconstruction is

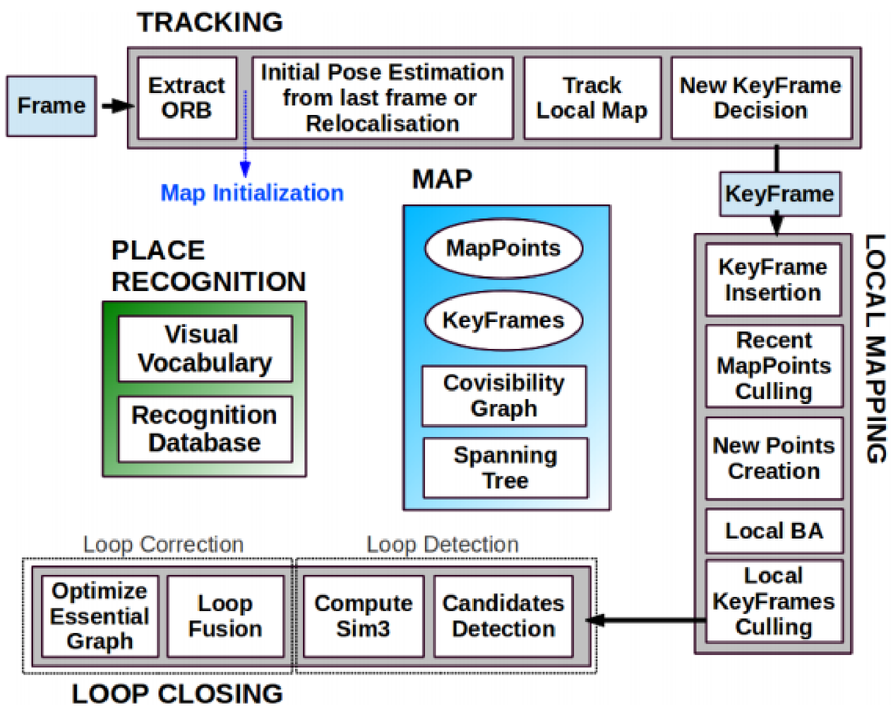


Fig. 1.2: ORB-SLAM pipeline [2]

lost when too many features disappear from the images, but the algorithm is always able to recover the trajectory once it revisits previously known map features.

Finally a subcategory of the V-SLAM problem is the visual odometry one. Visual odometry, or VO, is not generally tasked at reconstructing the 3D scene structure but only in recovering the trajectory. As such different methodologies are used and it might be useful mainly in situations where it would be impossible or improbable to revisit a particular 3D scene point twice, such as landing trajectories or rover ground operations.

One feature that might be needed in vision-based algorithms, depending on the application, is the ROI, region of interest, estimation. It can occur that not every part of an image should be processed in a V-SLAM framework, as it can happen for a spacecraft imaged with the Earth in the background. The different motion of the target spacecraft and the background can be very dangerous for the mapping and trajectory estimation process and a ROI extraction can help handling these cases. ROI algorithms were categorized by Lin et al. [33] into feature based algorithms, which use feature points to generate a ROI [34], and target based ones, which instead use higher level information, the target shape and structure, to compute the ROI [35] [36]. Feature based algorithms however do not provide accurate results as not all target feature points exhibit the same optical behaviour, while target based algorithms have been limited by the available computing power. Target based approaches generally exploit template matching or the learning of the known target shape, which can be conducted for example on support vector machines [37]. Thanks to the increasing computational resources, advanced machine learning techniques in recent years were explored, which are able to learn not only the final target shape but also its features, patterns and parts, and show very good performance in ROI estimation [38]. Machine learning, one of the research areas in computer vision, will be discussed next.

1.3 Machine Learning

In this section a description of machine learning is presented along with state of the art applications for space domain.
Machine learning is a field of artificial intelligence in which an agent learns how to perform a certain task through leaning. A formal definition is given by Tom Mitchell, professor at the Carnegie Mellon University [39]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". The task has one or more target variables that the model should be able to reconstruct when an input, or predictor variable, is given. AI instead, or artificial intelligence can be defined as "the capability of computer systems to perform tasks that normally require human intelligence" and can be used to increment system autonomy. Machine learning, robotics, computer vision, natural language processing all fall in the AI category [14].

Machine learning is generally distinguished in three major branches:

– Supervised learning: the model at first learns to associate an input to a given output through a training phase; it will then be able to predict target variables when a new input is shown.

– Unsupervised learning: the model is shown unlabelled inputs and is asked to recover unknown patterns or features by itself.

– Reinforcement learning: software agents take actions in an environment to explore or exploit it and learn from the outcome of their choices. Agents are generally required to maximize a cumulative reward.

Artificial neural networks, ANNs, are one of the tools that have been developed in the context of machine learning. They are present in everyone of the previous categories and mimic a very complex function that maps a given input to its output. They need at first to learn what prediction is expected when a given input is shown: this

process is referred to as training and requires a very large amount of data to be accomplished correctly. Next ANNs can be deployed by giving data as input and retrieving the output estimation.
Another way through which ANNs can be trained is through transfer learning, where a previously trained ANN is retrained on a different set of target data. This is done after having removed some of the last hidden layers and output layer of the network. The main concept behind transfer learning is that hidden layers of a network learn features to be recognized step by step. The first layers learn simple patterns and features, while the deeper ones learn more complex patterns. Thus by keeping the initial layers, a pretrained neural network can be specialized through further training to recognize different target variables.

For what concerns space applications, machine learning is best suited to automate various tasks and indeed it has found its role in performing several manual operations. For example many machine learning tools have been developed in order to retrieve information in Earth imaging data, from automated ice-charting [40] to ship detection [41]. For what concerns the space engineering sector, deep reinforcement learning has recently been used for the autonomous imaging and mapping of small bodies [42] [43]. The authors mention how interesting it would be to plan trajectories for very complex missions in autonomy with simultaneous attention to the mapping capabilities. A clear path towards neural network verification and validation is however needed before being able to deploy these tools in real world scenarios. Moreover various ESA challenges for space applications have taken place from 2015 by the Advanced Concepts Team [44]. The team mission is "to monitor and perform research on advanced space concepts and technologies, preparing ESA for any disruptive change to come" and, starting from the end of 2018, all the proposed challenges are developed with a machine learning approach. MISR, multi image super resolution, has been applied in the PROBA-V Super Resolution challenge to obtain a high definition image of a scene from low resolution ones. The winning team, with the DeepSUM network, used a complex CNN based architecture [45] to tackle the problem and obtain invariance to temporal and brightness variations in the network. In 2019 a challenge to estimate the pose of a known uncooperative satellite from the Prisma mission took place. It poses its foundation on the work by Sharma and D'Amico [46] who proposed a CNN based approach to retrieve the pose of a known and uncooperative spacecraft. A state of the art ROI neural network, Faster R-CNN [47], is trained to gain information about the 2D bounding box and then a hybrid classification/regression CNN obtains the pose of Tango spacecraft from custom generated images. The image dataset was implemented into the ESA challenge, which was later won by a team from the University of Adelaide. Their solution leverages on the regression of 2D positions of some interest points of Tango by a trained neural network. Once those selected points are known, they are matched to a 3D model of the spacecraft to again obtain the relative pose [48]. Finally, at the end of 2019, the spacecraft collision avoidance challenge took place, which provided users with a real world dataset of timed collision risk of ESA satellites. Naive forecast methods proved to be excellent at this task and machine learning techniques can provide additional accuracy by improving upon those methods [49].

Up to now, to the author's knowledge no current mission exploit machine learning in a substantial way. A clear road-map has been established for future development, showing the interest in machine learning research for space applications [50]. Many fields are discussed, from machine learning-based processing for star trackers to on board autonomy, failure prognostics and detection, etc, demonstrating how revolutionary machine learning could be for space exploration in the future.

1.4 Rendering for space applications

In this final section a brief survey of rendering softwares is provided. Accurate space imaging software is an important tool while dealing with visual relative navigation techniques: indeed it can be an useful instrument to deploy in combination with an experimental GNC facility, thanks to its fast prototyping capabilities. Moreover, thanks to the increasing improvements on graphic processors, it is often very efficient to run rendering engines on such GPUs which, when clustered in a server, can provide a huge amount of computational power.

Some softwares for space imaging already exist, such as PANGU [51], Planet and Asteroid Natural Scene Generation Utility, created by the Univerisity of Dundee in Scotland with support from ESA. It can generate camera and LiDAR images of different planetary bodies and spacecraft to test vision-guided navigation, guidance and landing systems. The software is licensed and to be used free of charges in ESA projects only.

Another similar software has been developed by Airbus Defence & Space. It is named SurRender [52] and it utilizes a ray tracing engine to generate images specifically for space scenes such as planetary approach, landing and in orbit rendezvous. The software is free to use upon request.
Both softwares include various effects induced by the space environment in which the camera is placed, in a quantitative manner. Examples can be camera lens distortion and internal light scattering, motion blur, electronic noises and rolling shutter.

In this thesis Blender [53] is used in order to get synthetic images of orbiting spacecraft. Blender is a free and open source 3D creation suite very popular in the entertainment industry, often used in animation, film production and 3D modeling. To the author's knowledge no major space-related tools have been produced in Blender, even if it is a standard referenced instrument for satellite and asteroids 3D modeling [54] [55]. A similar software called Unreal Engine 4 was also used to generate images of a spacecraft orbiting the Earth [56].

1.5 Thesis overview and contributions

The aim of this thesis is to explore the topic of relative navigation between two unknown and uncooperative space objects in the V-SLAM formulation and to study the possible use of machine learning methods in this research area. In order to do so, Blender is used to develop realistic space images of artificial satellites with the possibility of changing the target satellite model or the natural background. A V-SLAM like method has been implemented in Matlab from the ground up, using preexisting functions only for classical computer vision algorithms. Then, after understanding what kind of applications machine learning could be suited for, region of interest estimation has been chosen and examined. It is indeed important that features are detected for the target satellite only and are not derived from the background, where the Earth could be present, and a ROI solution can help mitigate this problem. While it would be possible to use standard techniques to code a ROI estimator, it would be incredibly difficult to set all the rules to find the proper bounding box around the target. As such, machine learning techniques are used, where a neural network learns the rules and patterns by itself from experience obtained through training. A dataset of around 10000 valid images has been produced in Blender with corresponding pose data in order to reconstruct the ground truth bounding box and a CNN has been designed to tackle this problem on a predefined scenario. Finally the CNN is tested in off design conditions to understand its behaviour in untrained scenes. Indeed CNN testing is an important aspect that often is not performed when dealing with known scenarios for predefined problems, however understanding the flexibility and robustness of an algorithm is essential for practical applications.
In summary, a new image rendering framework has been developed to aid V-SLAM and CNN ROI simulations. Often in literature actual spacecraft images are used for validation of V-SLAM pipelines, however they are small in number, with a small time duration and are affected by the ground trajectory estimation error (ex via GPS), which is not present when using an analytical framework. The V-SLAM application has been built from ground up and tested with synthetic space images, showing promising results. ROI detection for a target satellite has been studied with a machine learning approach, a research field in the early stages for space engineering applications. Instead of exploiting transfer learning, the typical but costly solution for state of the art networks applied in the space domain for the first time

in late 2018 [46], a novel CNN is trained with a less demanding supervised learning pipeline. Insights on off-nominal behaviour, to the author's knowledge not discussed in literature works, are obtained by testing the CNN on various off-design scenarios.

With respect to the thesis structure, in Section 2 an introduction to relative dynamics is present. Used as a theory framework, it mainly investigates the system of equations for nonlinear translational-rotational relative dynamics with simulations. In Section 3 projective geometry theory is discussed. This includes the basics for homogeneous transformations and the image formation through a pinhole camera model. Section 4 introduces computer vision theory and methods, starting from feature extraction, feature tracking and epipolar geometry and going through important numerical methods like the fundamental matrix estimation, triangulation and Perspective-n-Point. Essential machine learning theory is presented in Section 5 while in Section 6 the proposed architecture for relative spacecraft navigation is shown. A summary of the V-SLAM method and the CNN ROI estimation is given in order to provide an general overview of the work performed in the next chapters. The rendering framework, based on Blender and able to generate spacecraft images along with relevant pose data, is also introduced. This has been used to obtain pictures and ground truth data for the V-SLAM algorithm and for the CNN training process. The V-SLAM application to space objects is reported in Section 7 along with examples. A novel CNN design for region of interest estimation is then outlined in Section 8 along with off design testing and sensitivity to blur/noise assessment.
For what concerns the appendix, mathematical tools for computer vision and machine learning are discussed in Appendix A, Appendix B and Appendix C. A brief description of Vespa 3D model generation, a payload adapter for VEGA launcher, is described in Appendix D while a broad overview of the CNN testing datasets is present in Appendix E.

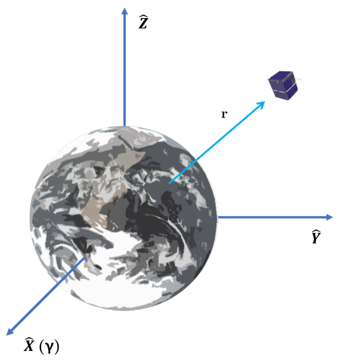
2 Relative dynamics

In order to have a consistent framework to develop this thesis, relative dynamics has been investigated first. Relative dynamics is important for visual navigation problems as any computer vision algorithm should reconstruct it from images as its goal. In the following sections the nonlinear relative dynamics system of equations and the coupled translational-rotational relative dynamics system of equations are derived. The latter is implemented and simulated in Matlab, with results shown in Section 2.4 along with the linear Clohessy Wiltshire model.

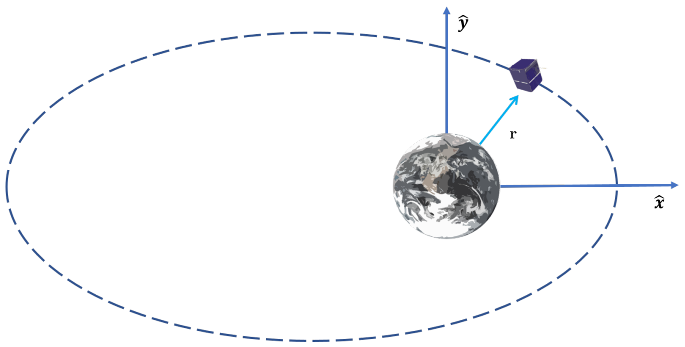
2.1 Reference Frames

Some standard reference frames are used through this chapter. They will be written as apices of the quantities they refer to.

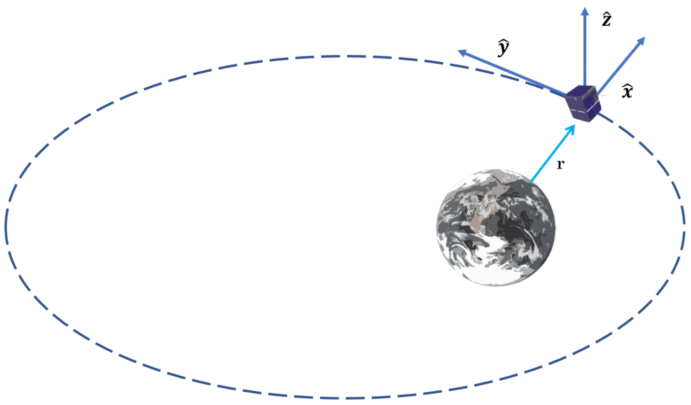
- ECI, Earth Centered Inertial reference frame (2.1a) $[I]$: inertial reference frame where $\hat{\boldsymbol{X}}$ points towards the vernal equinox and forms with $\hat{\boldsymbol{Y}}$ the equatorial plane. $\hat{\boldsymbol{Z}}$ is positive towards the North Pole.
- Perifocal reference frame (2.1b) $[P]$: $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ denote the orbital plane, with $\hat{\boldsymbol{x}}$ pointing towards the periapsis of the orbit.
- LVLH, Local Vertical, Local Horizontal reference frame (2.1c) $[L]$: $\hat{\boldsymbol{x}}$ is directed as the radial vector, $\hat{\boldsymbol{y}}$ lies in the orbital plane and $\hat{\boldsymbol{z}}$ completes the frame.
- Rotating reference frame (2.1d) $[R]$ used to write equations in polar coordinates, the vector $\hat{\boldsymbol{r}}$ goes outwards as $\boldsymbol{r}$ vector.



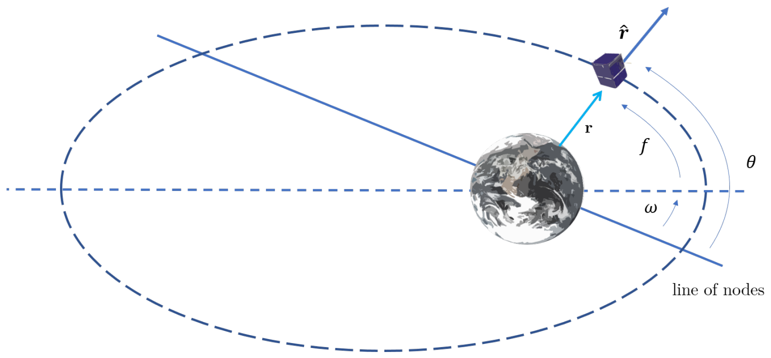
(a) ECI reference frame



(b) Perifocal reference frame



(c) LVLH reference frame



(d) Rotating reference frame

Fig. 2.1: Reference frames

2.2 System of equations for nonlinear relative dynamics

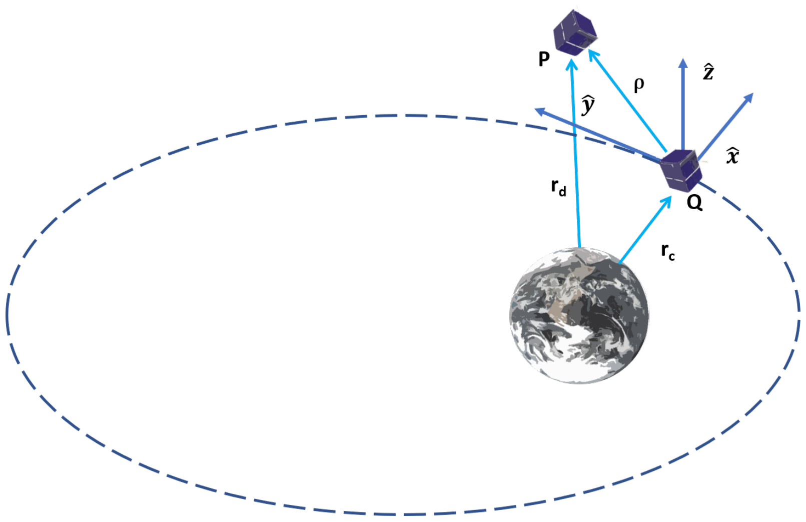


Fig. 2.2: Chief and Deputy spacecraft

The two-body problem equations of motion in an inertial reference frame are given by:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} \tag{2.1}$$

And by specializing them for the chief and deputy become:

$$\ddot{\mathbf{r}}_c = -\frac{\mu}{r_c^3}\mathbf{r}_c \tag{2.2}$$

$$\ddot{\mathbf{r}}_d = -\frac{\mu}{r_d^3}\mathbf{r}_d \tag{2.3}$$

where $\mathbf{r}$ denotes the position vector and μ is the planetary gravitational constant. The relative position of the deputy relative to the chief, as can be seen in Fig. 2.3, is:

$$\boldsymbol{\rho} = \mathbf{r}_c - \mathbf{r}_d \tag{2.4}$$

Subtracting Eq. (2.2) and Eq. (2.3) and rewriting the position of deputy with respect to the chief as $\mathbf{r}_d = \mathbf{r}_c + \boldsymbol{\rho}$, results in:

$$\ddot{\boldsymbol{\rho}} = -\frac{\mu}{||\mathbf{r}_c + \boldsymbol{\rho}||^3}(\mathbf{r}_c + \boldsymbol{\rho}) + \frac{\mu}{r_c^3}\mathbf{r}_c \tag{2.5}$$

Eq. (2.5) is written with respect to an inertial frame reference frame, however it would be useful to have it written in the LVLH frame. Since this last frame is not an inertial one by definition, the relative acceleration $\ddot{\boldsymbol{\rho}}$ written in LVLH frame can be recovered as [57]:

$$\boldsymbol{a}_a = \boldsymbol{a}_r + \boldsymbol{a}_\tau + \boldsymbol{a}_c \tag{2.6}$$

Where $\boldsymbol{a}_a$ is the absolute acceleration of the deputy, so measured in I frame, $\boldsymbol{a}_r$ is the relative acceleration of the deputy (relative to the chief), $\ddot{\boldsymbol{\rho}}$, measured in L frame, $\boldsymbol{a}_\tau = \boldsymbol{a}_Q + \dot{\boldsymbol{\omega}} \times \boldsymbol{QP} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{QP})$ is the entrained acceleration and $\boldsymbol{a}_c = 2\boldsymbol{\omega} \times \boldsymbol{v}_r$ is the Coriolis acceleration. $\boldsymbol{\omega}$ is the relative angular velocity of the two frames, the difference between the angular velocity of the chief LVLH frame and the ECI one. $\boldsymbol{v}_r$ is the relative velocity (of the deputy relative to the chief, $\dot{\boldsymbol{\rho}}$), so measured in L frame, while Q is the origin of the chief reference frame and P denotes the position of the deputy as in Fig. 2.3. Rewriting these terms yields:

$$\boldsymbol{a}_a - \boldsymbol{a}_Q = \boldsymbol{a}_r + \dot{\boldsymbol{\omega}} \times \boldsymbol{QP} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{QP}) + 2\boldsymbol{\omega} \times \boldsymbol{v}_r \tag{2.7}$$

since $\boldsymbol{a}_a - \boldsymbol{a}_Q$ is the relative acceleration of point P with respect to Q measured in the inertial frame, Eq. (2.7) can be rewritten as:

$$\frac{d^{2\,I}\boldsymbol{\rho}}{dt^2} = \frac{d^{2\,L}\boldsymbol{\rho}}{dt^2} + \frac{d\,{}^I\boldsymbol{\omega}^L}{dt} \times \boldsymbol{\rho} + {}^I\boldsymbol{\omega}^L \times ({}^I\boldsymbol{\omega}^L \times \boldsymbol{\rho}) + 2{}^I\boldsymbol{\omega}^L \times \frac{d^L\boldsymbol{\rho}}{dt} \tag{2.8}$$

Where the notation ${}^I\boldsymbol{\omega}^L = \boldsymbol{\omega}_L - \boldsymbol{\omega}_I$ denotes the two reference frames as well ($\boldsymbol{\omega}_I$ is null). Since the derivative of ${}^I\boldsymbol{\omega}^L$ is the same measured from the inertial I or the non-inertial L reference frames, its derivative doesn't have any apexes, differently from the ones of $\boldsymbol{\rho}$. By substituting Eq. (2.5) and quantities in L frame ${}^I\boldsymbol{\omega}^L = [0{,}0, \dot{\theta}_c]^T$, $\boldsymbol{r}_c = [r_c{,}0{,}0]^T$, $\rho = [x, y, z]^T$ into Eq. (2.8) the system of nonlinear equations for relative motion is obtained [58]:

$$\begin{cases} \ddot{x} - 2\dot{f}_c\dot{y} - \ddot{f}_c y - \dot{f}_c^2 x = -\dfrac{\mu\,(r_c + x)}{\left[(r_c + x)^2 + y^2 + z^2\right]^{\frac{3}{2}}} + \dfrac{\mu}{r_c^2} \\ \ddot{y} + 2\dot{f}_c\dot{x} + \ddot{f}_c x - \dot{f}_c^2 y = -\dfrac{\mu y}{\left[(r_c + x)^2 + y^2 + z^2\right]^{\frac{3}{2}}} \\ \ddot{z} = -\dfrac{\mu z}{\left[(r_c + x)^2 + y^2 + z^2\right]^{\frac{3}{2}}} \end{cases} \tag{2.9}$$

The equation for absolute motion of the chief completes this set, namely Eq. (2.2). In order to have f_c (chief true anomaly) and r_c explicitly, that equation is solved in R frame where $\boldsymbol{r}_c = r_c\hat{\boldsymbol{r}}$. By derivating twice this last expression, where $\hat{\boldsymbol{r}} = \hat{\boldsymbol{r}}(t)$, one gets:

$$\begin{cases} \ddot{r}_c = r_c \dot{f}_c^2 - \dfrac{\mu}{r_c^2} \\ \ddot{f}_c = -\dfrac{2\dot{r}_c \dot{f}_c}{r_c} \end{cases} \tag{2.10}$$

Eqs. (2.9) and Eqs. (2.10) form a system of five second order differential equations which can be reduced in order into ten ODEs and integrated.

2.3 System of equations for coupled translational-rotational relative dynamics

When two spacecraft are in close proximity, they cannot be modeled anymore as single points, as their dimensions might be comparable in order of magnitude to their relative distance. It is thus important to develop a model that can take into account both the relative dynamic and the attitude dynamics of chief and deputy. Each spacecraft will then have its own LVLH frame, labelled as L_c and L_d, and its own Body frame, B_c and B_d. For this model, it is assumed that the satellites are Earth pointing, so that $L_c \equiv B_c$ and $L_d \equiv B_d$.

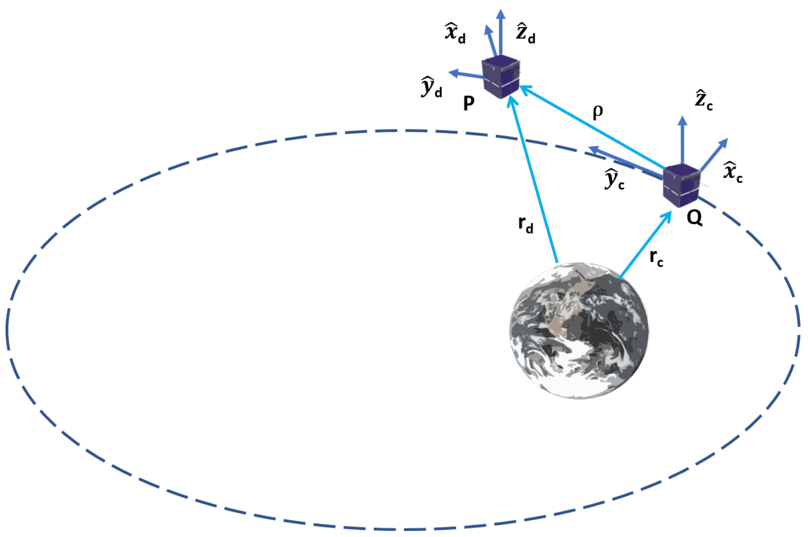


Fig. 2.3: Chief and Deputy spacecraft with their reference frames

2.3.1 Relative rotational model

First define $\boldsymbol{\omega}_c$ as the chief angular velocity, $\boldsymbol{\omega}_d$ as the deputy angular velocity and

$$\boldsymbol{\omega} = \boldsymbol{\omega}_d - \boldsymbol{\omega}_c \tag{2.11}$$

as the relative angular velocity in a given reference frame. Define ζ the eigenangle of rotation and $\boldsymbol{e}$ the eigenaxis of rotation which uniquely parametrize the Euler rotation from L_c to L_d. The relative quaternion vector $\boldsymbol{\beta}$ is then:

$$\begin{cases} \beta_1 = e_1 sin(\frac{\zeta}{2}) \\ \beta_2 = e_2 sin(\frac{\zeta}{2}) \\ \beta_3 = e_3 sin(\frac{\zeta}{2}) \\ \beta_0 = cos(\frac{\zeta}{2}) \end{cases} \tag{2.12}$$

The rotation matrix from L_c to L_d expressed in terms of quaternions becomes:

$$\boldsymbol{D}(\boldsymbol{\beta}) = \begin{bmatrix} \beta_1^2 - \beta_2^2 - \beta_3^2 + \beta_0^2 & 2\left(\beta_1\beta_2 - \beta_3\beta_0\right) & 2\left(\beta_1\beta_3 + \beta_2\beta_0\right) \\ 2\left(\beta_1\beta_2 + \beta_3\beta_0\right) & -\beta_1^2 + \beta_2^2 - \beta_3^2 + \beta_0^2 & 2\left(\beta_2\beta_3 - \beta_1\beta_0\right) \\ 2\left(\beta_1\beta_3 - \beta_2\beta_0\right) & 2\left(\beta_2\beta_3 + \beta_1\beta_0\right) & -\beta_1^2 - \beta_2^2 + \beta_3^2 + \beta_0^2 \end{bmatrix} \tag{2.13}$$

and the quaternion kinematic equation of motion is :

$$\dot{\boldsymbol{\beta}} = \frac{1}{2}\boldsymbol{Q}(\boldsymbol{\beta})\boldsymbol{\omega}|_{L_d} \tag{2.14}$$

with:

$$\boldsymbol{Q}(\boldsymbol{\beta}) = \begin{bmatrix} -\beta_1 & -\beta_2 & -\beta_3 \\ \beta_0 & -\beta_3 & \beta_2 \\ \beta_3 & \beta_0 & -\beta_1 \\ -\beta_2 & \beta_1 & \beta_0 \end{bmatrix} \tag{2.15}$$

and:

$$\boldsymbol{\omega}|_{L_d} = \boldsymbol{D}(\boldsymbol{\beta})^T \boldsymbol{\omega}|_{L_c} \tag{2.16}$$

To derive the attitude dynamics of the deputy relative to the chief, express Eq. (2.11) in L_c and derive it in I:

$$\boldsymbol{\omega}|_{L_c} = \boldsymbol{\omega}_d|_{L_c} - \boldsymbol{\omega}_c|_{L_c} = \boldsymbol{D}(\boldsymbol{\beta})\boldsymbol{\omega}_d|_{L_d} - \boldsymbol{\omega}_c|_{L_c} \tag{2.17}$$

$$\left.\frac{d^I\,\boldsymbol{\omega}}{dt}\right|_{L_c} = \left.\frac{d^I\,\boldsymbol{\omega}_d}{dt}\right|_{L_c} - \left.\frac{d^I\,\boldsymbol{\omega}_c}{dt}\right|_{L_c} = \boldsymbol{D}(\boldsymbol{\beta})\left.\frac{d^I\,\boldsymbol{\omega}_d}{dt}\right|_{L_d} - \left.\frac{d^I\,\boldsymbol{\omega}_c}{dt}\right|_{L_c} \tag{2.18}$$

The derivative in I of $\boldsymbol{\omega}|_{L_c}$ can be obtained in another way too. The derivation rule from an inertial I to a non-inertial V frame of a generic vector $\boldsymbol{c}$ is [57]:

$$\frac{d^I\boldsymbol{c}}{dt} = \frac{d^V\boldsymbol{c}}{dt} +^I \boldsymbol{\omega}^V \times \boldsymbol{c} \tag{2.19}$$

and by exploiting it on $\boldsymbol{\omega}$ and expressing the resulting equation in L_c:

$$\left.\frac{d^I\boldsymbol{\omega}}{dt}\right|_{L_c} = \left.\frac{d^{L_c}\boldsymbol{\omega}}{dt}\right|_{L_c} + \boldsymbol{\omega}_c|_{L_c}\times\boldsymbol{\omega}|_{L_c} \tag{2.20}$$

Comparing Eq. (2.18) and Eq. (2.20) gets:

$$\left.\frac{d^{L_c}\boldsymbol{\omega}}{dt}\right|_{L_c} = \boldsymbol{D}(\boldsymbol{\beta})\left.\frac{d^I\,\boldsymbol{\omega}_d}{dt}\right|_{L_d} - \left.\frac{d^I\,\boldsymbol{\omega}_c}{dt}\right|_{L_c} - \boldsymbol{\omega}_c|_{L_c}\times\boldsymbol{\omega}|_{L_c} \tag{2.21}$$

In order to drop the deputy angular velocity in the previous equation, it is possible to write Euler's equation for the deputy and chief angular velocity (by applying Eq. (2.19) to the angular momentums of the chief and deputy $\boldsymbol{I}_c\boldsymbol{\omega}_c$ and $\boldsymbol{I}_d\boldsymbol{\omega}_d$):

$$\boldsymbol{I}_c\left.\frac{d^I\boldsymbol{\omega}_c}{dt}\right|_{L_c} = \boldsymbol{I}_c\left.\frac{d^{L_c}\boldsymbol{\omega}_c}{dt}\right|_{L_c} + \boldsymbol{\omega}_c|_{L_c}\times\boldsymbol{I}_c\boldsymbol{\omega}_c|_{L_c} = \boldsymbol{N}_c \tag{2.22}$$

$$\boldsymbol{I}_d\left.\frac{d^I\boldsymbol{\omega}_d}{dt}\right|_{L_d} = \boldsymbol{I}_d\left.\frac{d^{L_d}\boldsymbol{\omega}_d}{dt}\right|_{L_d} + \boldsymbol{\omega}_d|_{L_d}\times\boldsymbol{I}_d\boldsymbol{\omega}_d|_{L_d} = \boldsymbol{N}_d \tag{2.23}$$

where $\boldsymbol{N}$ denotes the external torque applied on the spacecraft, if any. Moreover, since the chief spacecraft is in an LVLH frame, $\boldsymbol{\omega}_c$ can be written as $\boldsymbol{\omega}_c = [0{,}0, \dot{f}_c]^T$. Finally, substituting Eq. (2.17), Eq. (2.22) , Eq. (2.23) into Eq. (2.21) and multiplying by $\boldsymbol{I}_c$ yields the equation of relative attitude dynamics written in terms of chief/relative angular velocities:

$$\begin{aligned}\boldsymbol{I}_c\left.\frac{d^{L_c}\boldsymbol{\omega}}{dt}\right|_{L_c} =& \boldsymbol{I}_c\boldsymbol{D}(\boldsymbol{\beta})\boldsymbol{I}_d^{-1}\{\boldsymbol{N}_d - \boldsymbol{D}(\boldsymbol{\beta})^T(\boldsymbol{\omega}|_{L_c}+\boldsymbol{\omega}_c|_{L_c})\times\boldsymbol{I}_d\boldsymbol{D}(\boldsymbol{\beta})^T(\boldsymbol{\omega}|_{L_c}+\boldsymbol{\omega}_c|_{L_c})\}\\ &- \boldsymbol{I}_c\boldsymbol{\omega}_c|_{L_c}\times\boldsymbol{\omega}|_{L_c} - \{\boldsymbol{N}_c - \boldsymbol{\omega}_c|_{L_c}\times\boldsymbol{I}_c\boldsymbol{\omega}_c|_{L_c}\}\end{aligned} \tag{2.24}$$

Eq. (2.14) and Eq. (2.24) fully describe the relative attitude behaviour. Next, the relative translational motion is investigated.

2.3.2 Relative translational model

For what concerns relative translational motion, feature points on the two spacecraft must be taken into account. Define P_d^i a generic point on the deputy and P_c^j a generic point on the chief. Denote the distance from the CMs as $\boldsymbol{P}_d^i|_{L_d}$ and $\boldsymbol{P}_c^j|_{L_c}$, $\boldsymbol{\rho}_{ij}$ being their distance.

The velocity of the deputy feature point in L_c is obtained as [57]:

$$\frac{d^{L_c}\boldsymbol{P}_d^i}{dt} = \frac{d^{L_d}\boldsymbol{P}_d^i}{dt} + \frac{d^{L_c}\boldsymbol{\rho}}{dt} + \boldsymbol{\omega}\times\boldsymbol{P}_d^i \tag{2.25}$$

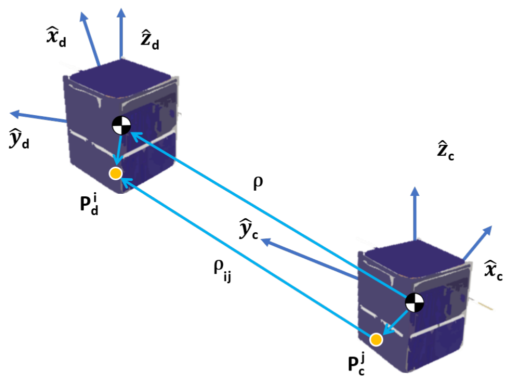


Fig. 2.4: Chief and Deputy spacecraft

and the acceleration is, by using Eq. (2.6):

$$\frac{d^{2\,L_c}\boldsymbol{P}_d^i}{dt^2} = \frac{d^{2\,L_d}\boldsymbol{P}_d^i}{dt^2} + \frac{d^{2\,L_c}\boldsymbol{\rho}}{dt^2} + \frac{d\boldsymbol{\omega}}{dt} \times \boldsymbol{P}_d^i + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{P}_d^i) + 2\boldsymbol{\omega} \times \frac{d^{L_d}\boldsymbol{P}_d^i}{dt} \tag{2.26}$$

Since the deputy is a rigid body, the derivatives in L_d of $\boldsymbol{P}_d^i$ are null; the last two equations become:

$$\frac{d^{L_c}\boldsymbol{P}_d^i}{dt} = \frac{d^{L_c}\boldsymbol{\rho}}{dt} + \boldsymbol{\omega} \times \boldsymbol{P}_d^i \tag{2.27}$$

$$\frac{d^{2\,L_c}\boldsymbol{P}_d^i}{dt^2} = \frac{d^{2\,L_c}\boldsymbol{\rho}}{dt^2} + \frac{d\boldsymbol{\omega}}{dt} \times \boldsymbol{P}_d^i + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{P}_d^i) \tag{2.28}$$

The relative distance and derivatives in L_c are obtained, from Fig. 2.4, as:

$$\boldsymbol{\rho}_{ij}|_{L_c} = \boldsymbol{P}_d^i|_{L_c} - \boldsymbol{P}_c^j|_{L_c} \tag{2.29}$$

$$\left.\frac{d^{L_c}\boldsymbol{\rho}_{ij}}{dt}\right|_{L_c} = \left.\frac{d^{L_c}\boldsymbol{P}_d^i}{dt}\right|_{L_c} + \left.\frac{d^{L_c}\boldsymbol{P}_c^j}{dt}\right|_{L_c} \tag{2.30}$$

$$\left.\frac{d^{2\,L_c}\boldsymbol{\rho}_{ij}}{dt^2}\right|_{L_c} = \left.\frac{d^{2\,L_c}\boldsymbol{P}_d^i}{dt^2}\right|_{L_c} + \left.\frac{d^{2\,L_c}\boldsymbol{P}_c^j}{dt^2}\right|_{L_c} \tag{2.31}$$

or by substituting Eq. (2.27) and Eq. (2.28) into Eq. (2.30) and Eq. (2.31) respectively:

$$\boldsymbol{\rho}_{ij}|_{L_c} = \boldsymbol{P}_d^i|_{L_c} - \boldsymbol{P}_c^j|_{L_c} \tag{2.32}$$

$$\left.\frac{d^{L_c}\boldsymbol{\rho}_{ij}}{dt}\right|_{L_c} = \left.\frac{d^{L_c}\boldsymbol{\rho}}{dt}\right|_{L_c} + \boldsymbol{\omega} \times \boldsymbol{P}_d^i + \left.\frac{d^{L_c}\boldsymbol{P}_c^j}{dt}\right|_{L_c} \tag{2.33}$$

$$\left.\frac{d^{2\,L_c}\boldsymbol{\rho}_{ij}}{dt^2}\right|_{L_c} = \left.\frac{d^{2\,L_c}\boldsymbol{\rho}}{dt^2}\right|_{L_c} + \frac{d\boldsymbol{\omega}}{dt} \times \boldsymbol{P}_d^i + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{P}_d^i) + \left.\frac{d^{2\,L_c}\boldsymbol{P}_c^j}{dt^2}\right|_{L_c} \tag{2.34}$$

Denoting $\boldsymbol{\rho} = [x, y, z]^T$, $\boldsymbol{\rho}_{ij} = [x_{ij}, y_{ij}, z_{ij}]^T$, $\boldsymbol{P}_d^i = [P_{x_d}^i, P_{y_d}^i, P_{z_d}^i]^T$, $\boldsymbol{P}_c^j = [P_{x_c}^j, P_{y_c}^j, P_{z_c}^j]^T$, $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$ and substituting this last three equations into Eq. (2.9) gets the following system for relative motion dynamics [58]:

$$
\begin{cases}
\ddot{x}_{ij} - \left[\omega_y\left(\omega_x P^i_{y_d} - \omega_y P^i_{x_d}\right) - \omega_z\left(\omega_z P^i_{x_d} - \omega_x P^i_{z_d}\right)\right] - \dot{\omega}_y P^i_{z_d} + \dot{\omega}_z P^i_{y_d} \\
\quad - 2\dot{f}_c\left[y_{ij} - \left(\omega_z P^i_{x_d} - \omega_x P^i_{z_d}\right)\right] - \ddot{f}_c\left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right) - f_c^2\left(x_{ij} - P^i_{x_d} + P^j_{x_c}\right) \\
\quad = \dfrac{-\mu\left(r_c + x_{ij} - P^i_{x_d} + P^j_{x_c}\right)}{\left[\left(r_c + x_{ij} - P^i_{x_d} + P^j_{x_c}\right)^2 + \left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right)^2 + \left(z_{ij} - P^i_{z_d} + P^j_{z_c}\right)^2\right]^{\frac{3}{2}}} + \dfrac{\mu}{r_c^2} \\
\ddot{y}_{ij} - \left[\omega_z\left(\omega_y P^i_{z_d} - \omega_z P^i_{y_d}\right) - \omega_x\left(\omega_x P^i_{y_d} - \omega_y P^i_{x_d}\right)\right] - \dot{\omega}_z P^i_{x_d} + \dot{\omega}_x P^i_{z_d} \\
\quad + 2\dot{f}_c\left[x_{ij} - \left(\omega_y P^i_{z_d} - \omega_z P^i_{y_d}\right)\right] + \ddot{f}_c\left(x_{ij} - P^i_{x_d} - P^j_{x_c}\right) - \dot{f}_c^2\left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right) \\
\quad = \dfrac{-\mu\left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right)}{\left[\left(r_c + x_{ij} - P^i_{x_d} + P^j_{x_c}\right)^2 + \left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right)^2 + \left(z_{ij} - P^i_{z_d} + P^j_{z_c}\right)^2\right]^{\frac{3}{2}}} \\
\ddot{z}_{ij} - \left[\omega_x\left(\omega_z P^i_{x_d} - \omega_x P^i_{z_d}\right) - \omega_y\left(\omega_y P^i_{z_d} - \omega_z P^i_{y_d}\right)\right] - \dot{\omega}_x P^i_{y_d} + \dot{\omega}_y P^i_{x_d} \\
\quad = \dfrac{-\mu\left(z_{ij} - P^i_{z_d} + P^j_{z_c}\right)}{\left[\left(r_c + x_{ij} - P^i_{x_d} + P^j_{x_c}\right)^2 + \left(y_{ij} - P^i_{y_d} + P^j_{y_c}\right)^2 + \left(z_{ij} - P^i_{z_d} + P^j_{z_c}\right)^2\right]^{\frac{3}{2}}}
\end{cases}
\tag{2.35}
$$

The system of differential equations is then composed by Eq. (2.14), Eq. (2.24) and Eq. (2.35) along with Eq. (2.10). There are a total of 17 ODEs to be integrated: six come from $\ddot{\boldsymbol{\rho}}_{ij}$, two from $\ddot{f}_c$, two from $\ddot{r}_c$, three from the relative omega $\dot{\boldsymbol{\omega}}|_{L_c}$ and four from quaternions kinematics $\dot{\boldsymbol{\beta}}$.

2.4 Numerical Simulations

In this section a numerical simulation and validation is presented for the models previously discussed. The nonlinear translational-rotational coupled (NL) model is integrated while Clohessy-Wiltshire (CW) model is used for validation. CW model is the linearization of Eq. (2.9), it holds whenever chief orbit is circular and the relative distance $\boldsymbol{\rho}$ is much smaller with respect to the orbital radius. The following data have been used:

Table 1: Orbital simulation data

a_c [km]	e_c [-]	i_c [deg]	Ω_c [deg]	ω_c [deg]	f_c [deg]
7170	0.05	15	0	340	20

The inertia matrix of the two spacecraft has been assumed diagonal with $I_x = 500$ [kg m^2], $I_y = 500$ [kg m^2] and $I_z = 500$ [kg m^2]. The integration time is one orbital period. Finally the selected feature points are $\boldsymbol{P}_c = (0\ 0\ 0)$ [m] and the set $\boldsymbol{P}_d =$

Table 2: Relative translation simulation data

$x_{00}(t_0)$ [m]	$y_{00}(t_0)$ [m]	$z_{00}(t_0)$ [m]	$\dot{x}_{00}(t_0)$ [m]	$\dot{y}_{00}(t_0)$ [m]	$\dot{z}_{00}(t_0)$ [m]
25	25	50	0	-0.0555	0

Table 3: Relative attitude simulation data

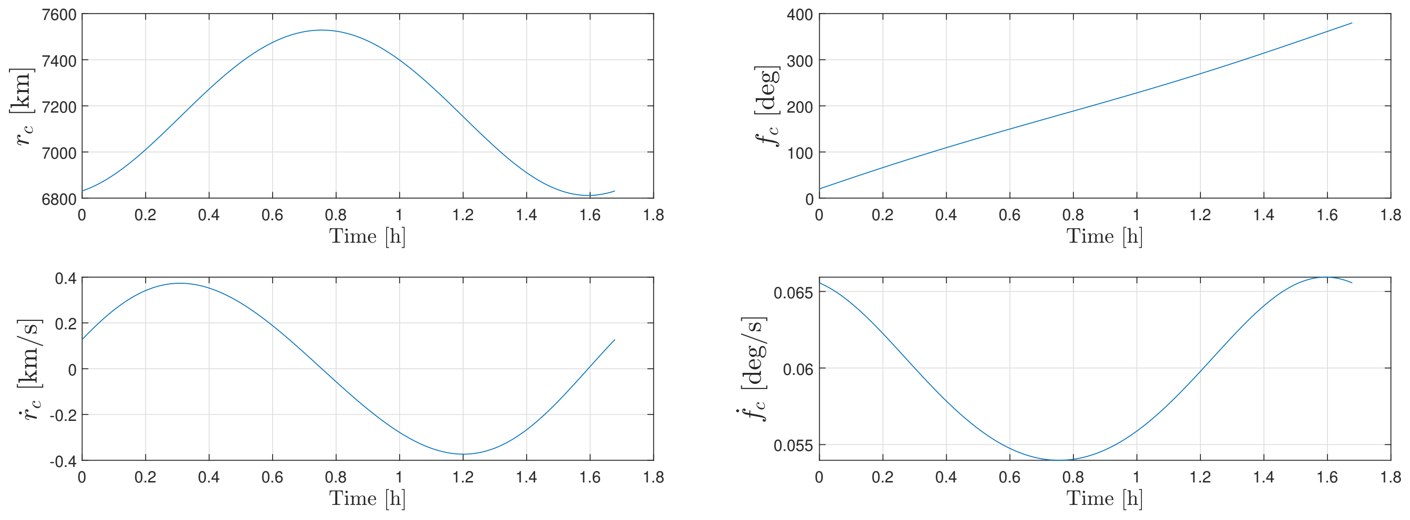
$\omega_x(t_0)$ [°/s]	$\omega_y(t_0)$ [°/s]	$\omega_z(t_0)$ [°/s]	β_0	β_1	β_2	β_3
0.006	0.006	0.12	1	0	0	0

[(0 0 0),(1.5 1.5 0),(-1.5 -1.5 0)] [m]. The ICs for points different from the CM are computed as [59]:

$$\boldsymbol{\rho}_{ij}(t_0) = \boldsymbol{\rho}(t_0) + \boldsymbol{P}_d^i \tag{2.36}$$

$$\frac{d^{L_c}\boldsymbol{\rho}_{ij}(t_0)}{dt} = \frac{d^{L_c}\boldsymbol{\rho}(t_0)}{dt} + \boldsymbol{\omega}(t_0) \times \boldsymbol{P}_d^i \tag{2.37}$$

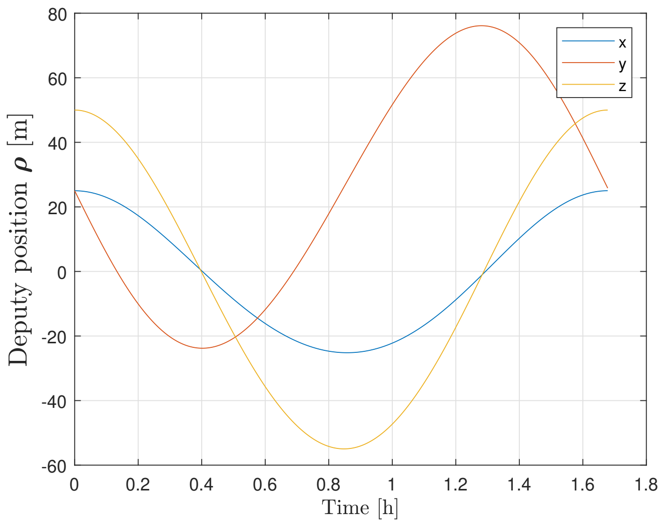
In Fig. 2.5 the orbital description of the chief spacecraft is obtained by integration of Eq. (2.10) and its behaviour is coherent with the propagation of an elliptical orbit. Fig. 2.6 shows instead the propagation of $\boldsymbol{\rho}$, so deputy CM in L_c in terms of relative position and relative attitude quantities.



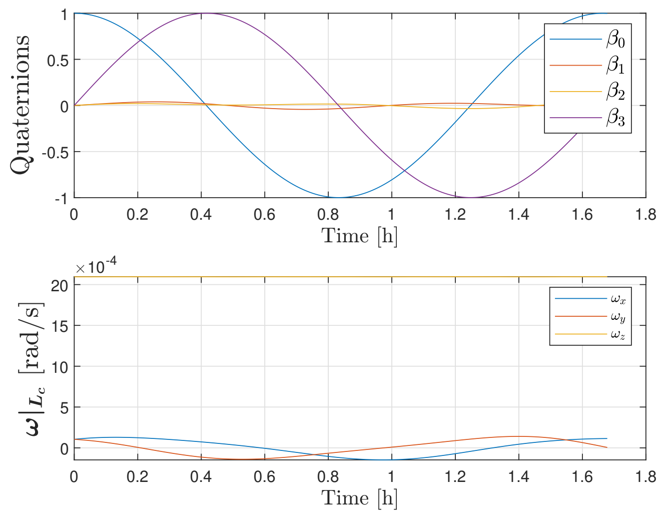
(a) Radius and derivative

(b) True anomaly and derivative

Fig. 2.5: Orbital dynamics



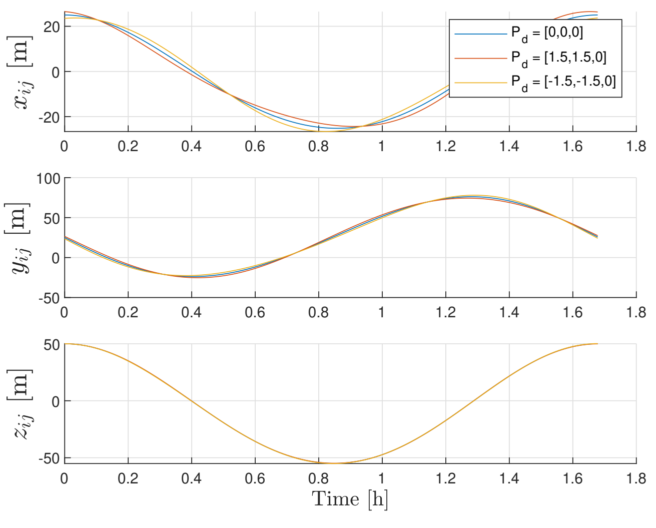
(a) Deputy position in L_c



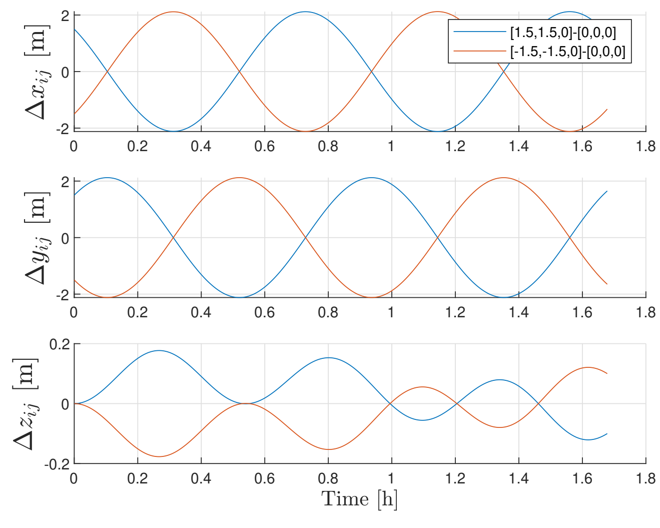
(b) Relative quaternions and angular velocity

Fig. 2.6: Attitude and position propagation

Fig. 2.7a shows the relative position of the three deputy feature points and denotes how relative rotation can affect relative translation through the kinematic coupling. Fig. 2.7b shows the difference between the propagation of $\boldsymbol{P}_d = (0\ 0\ 0)$ and the other two points. It's possible to see how the motion of those feature points can be recovered by applying a sinusoidal motion aroud the CM of the deputy.

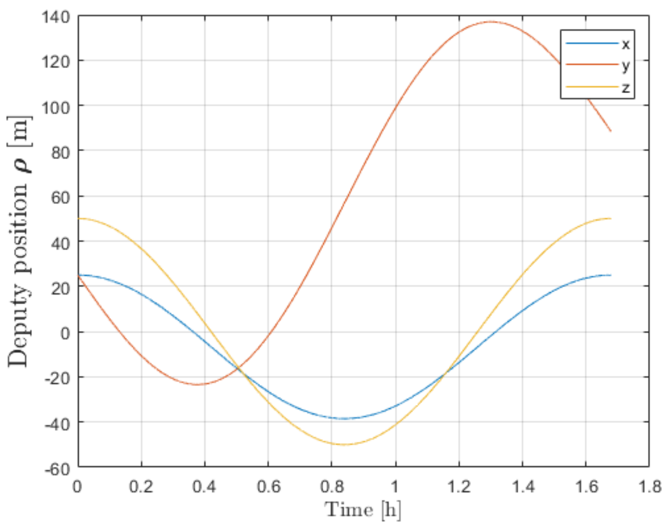


(a) Deputy feature positions in L_c

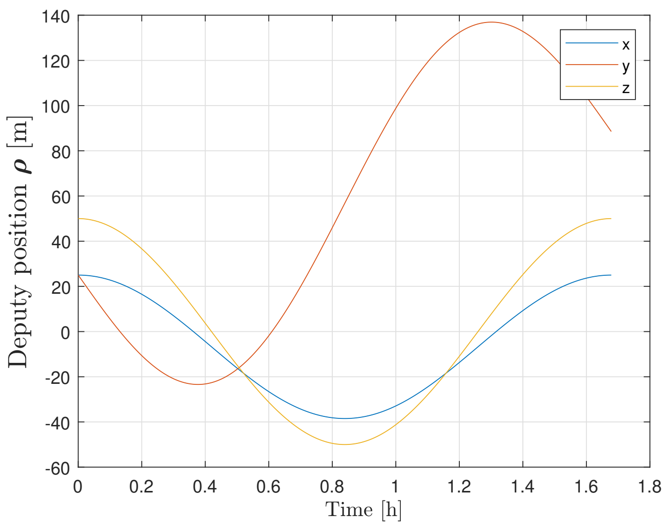


(b) Deputy feature positions difference in L_c

Fig. 2.7: Comparison of different feature positions

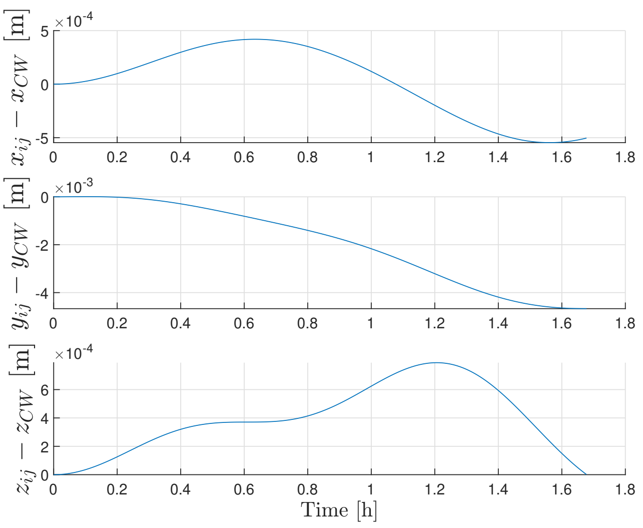


(a) $\boldsymbol{\rho}$ propagation with the NL model

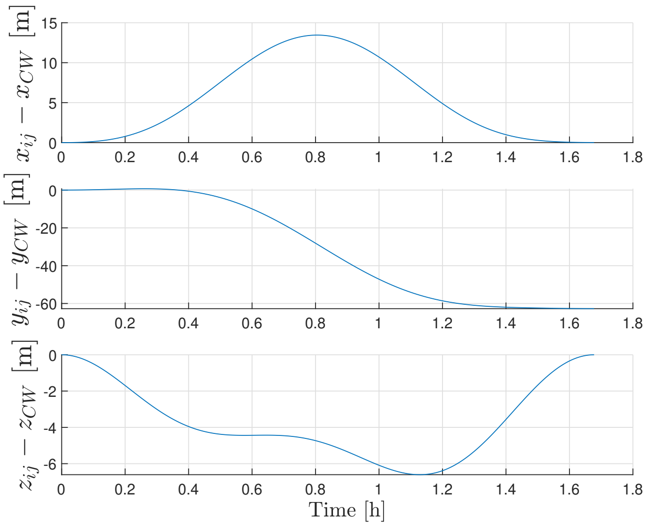


(b) $\boldsymbol{\rho}$ propagation with CW model

Fig. 2.8: Comparison of deputy CM position with CW and NL models



(a) Difference between CW and NL, $P_d = (0,0,0)$ and $e_c = 0$



(b) Difference between CW and NL, $P_d = (0,0,0)$ and $e_c = 0.05$

Fig. 2.9: Differences of deputy CM position, CW and NL models for $e_c = 0$ & $e_c = 0.05$

In Fig. 2.8 the position of the deputy CM is presented. In Fig. 2.8a the NL model is used with e_c is set to zero. In Fig. 2.8b, integration with CW model is presented.

Finally, in Fig. 2.9a the difference between the NL propagation with zero eccentricity is plotted against the CW one. Fig. 2.9b shows the NL propagation with $e_c = 0.05$ against the CW one. The CW model is extremely close to the NL one when e_c=0, however is not able to precisely describe the behaviour of the deputy CM in the second case. This error would be even greater if the point to be described were features points different from the CM of the deputy.

The subject of relative dynamics was the first one to be thoroughly explored during the course of this thesis. Even though these models and simulations have not been used later on, they provide a valuable theory framework in which to shape and understand further topics. Most rendering software adopt the same quantities definition as in these relative dynamics models and this theory background can prove to be very useful. Since those software allow for the definition of trajectory of an object before imaging, such models could be exploited in the future to automate the trajectory rendering process.
Moreover relative motion models are fundamental when a filter based solution to the V-SLAM problem is sought. In this case a motion model is directly embedded into the selected filter generating a simple and efficient estimation procedure [30].

3 Projective geometry

In the following chapter important topics needed for a visual navigation system, such as geometrical transformations and camera model, are discussed. The context in which cameras operate and pictures are taken cannot be fully explained through the classical Euclidean geometry. For instance, two parallel lines will never intersect together in $\mathbf{R}^3$. However this clashes with the perception of the world that we live in: for example when looking at railway tracks in the distance as in Fig. 3.1, they do appear to converge and meet at an infinite distance while being parallel. A way to understand and model how 2D images of 3D objects are formed comes from projective geometry.

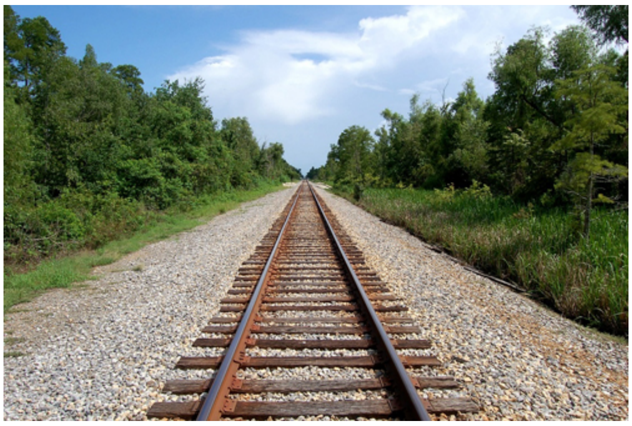


Fig. 3.1: Parallel railway tracks converging at infinity

In 2D projective geometry a particular notation for coordinates is used, called homogeneous. A general 2D point $\boldsymbol{x} = (x, y)^T$ is written in homogeneous coordinates as $\tilde{\boldsymbol{x}} = (x_1, x_2, x_3)^T$. In order to recover the inhomogeneous representation it is necessary to divide by the last element x_3, thus getting the point $\boldsymbol{x} = (x_1/x_3, x_2/x_3)^T$. Moreover, if $x_3 = 0$, $\tilde{\boldsymbol{x}}$ becomes a point at infinity, where lines, that in $\boldsymbol{R}^2$ or $\boldsymbol{R}^3$ would be parallel, meet.
Lines can be written as homogeneous vectors too, where homogeneous indicates that scaling that particular vector won't change what it represents. A line $ax + by + c = 0$ can be rewritten in a vector form as $\tilde{\boldsymbol{l}} = (a, b, c)^T$. This is an homogeneous form too since $\tilde{\boldsymbol{l}}_1 = k(a, b, c)^T$ is representing the same line as $\tilde{\boldsymbol{l}}$. It can be quickly demonstrated that a point $\tilde{\boldsymbol{x}} = (x_1, x_2, x_3)^T$ lies on a line $\tilde{\boldsymbol{l}} = (a, b, c)^T$ if $\tilde{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = 0$. By applying this property, it holds that two parallel line $\tilde{\boldsymbol{l}}_1 = (a, b, c)^T$ and $\tilde{\boldsymbol{l}}_2 = (a, b, c')^T$ do indeed meet at the same point at infinity $\tilde{\boldsymbol{x}} = (x_1, x_2, 0)^T$.

3.1 Geometric Transformations

A discussion on the expression of 2D geometric transformations with homogeneous coordinates is reported below. These transformations can be generalized from 2D to 3D with minor efforts and are an essential step for further projective geometry topics.

3.1.1 Translation

The easiest form of geometric transformation is a simple translation. It can be expressed as:

$$\bar{\boldsymbol{x}}' = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \bar{\boldsymbol{x}} \tag{3.1}$$

where the notation $\bar{\boldsymbol{x}}$ indicates an augmented vector, which is an homogeneous vector with the scale term $x_3 = 1$. $\boldsymbol{I}$ is the 2×2 identity matrix and $\boldsymbol{t}$ is the translation vector. The expression for $\bar{\boldsymbol{x}}'$ is then:

$$\bar{x}'_1 = \bar{x}_1 + t_1 \tag{3.2}$$

$$\bar{x}'_2 = \bar{x}_2 + t_2 \tag{3.3}$$

$$\bar{x}'_3 = 1 \tag{3.4}$$

A simple translation is shown in Fig. 3.5.

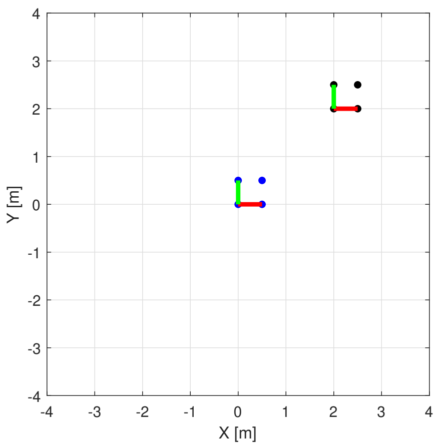


Fig. 3.2: Translation example

Note that, if vector $\bar{\boldsymbol{x}}$ was scaled by a factor k, so that $x_3 \neq 1$, its expression would be $\tilde{\boldsymbol{x}} = (k\bar{x}_1, k\bar{x}_2, k)^T = (\tilde{x}_1, \tilde{x}_2, \tilde{x}_3)$. The vector $\tilde{\boldsymbol{x}}'$ is then:

$$\tilde{x}'_1 = \tilde{x}_1 + t_1\tilde{x}_3 \tag{3.5}$$

$$\tilde{x}'_2 = \tilde{x}_2 + t_2\tilde{x}_3 \tag{3.6}$$

$$\tilde{x}'_3 = \tilde{x}_3 \tag{3.7}$$

The resulting vector $\tilde{\boldsymbol{x}}'$ is different from the previous vector $\bar{\boldsymbol{x}}'$, as it is just a scaled version of it. When trying to recover the inhomogeneous point however, the result will be the same:

$$x'_1 = \frac{\tilde{x}'_1}{\tilde{x}'_3} = \frac{\tilde{x}_1 + t_1\tilde{x}_3}{\tilde{x}_3} = \frac{k\bar{x}_1 + t_1k\bar{x}_3}{k} = \frac{\bar{x}'_1}{\bar{x}'_3} = \bar{x}_1 + t_1 \tag{3.8}$$

$$x'_2 = \frac{\tilde{x}'_2}{\tilde{x}'_3} = \frac{\tilde{x}_2 + t_2\tilde{x}_3}{\tilde{x}_3} = \frac{k\bar{x}_2 + t_2k\bar{x}_3}{k} = \frac{\bar{x}'_2}{\bar{x}'_3} = \bar{x}_2 + t_2 \tag{3.9}$$

Indeed in $\boldsymbol{P}^2$, two vectors differing only from scale will be projected in the same $\boldsymbol{P}^2$ point, as shown in Fig. 3.3.

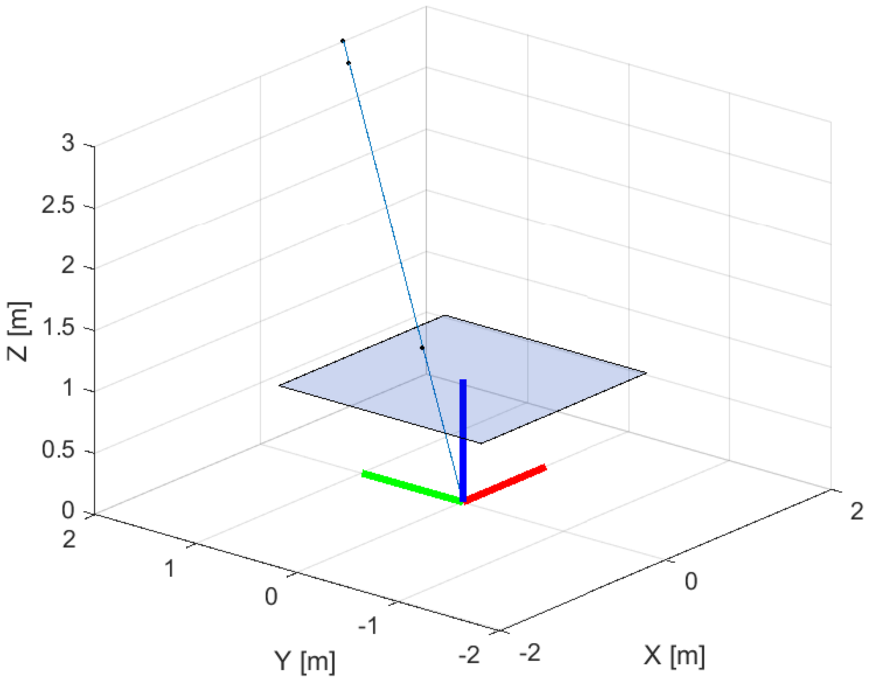


Fig. 3.3: Scaled homogeneous vectors to inhomogeneous one

3.1.2 Euler transformation

Euler transformation, also referred to as rigid body transformation, encodes a translation and a rotation in its transformation matrix. It can be expressed as:

$$\bar{\boldsymbol{x}}' = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \bar{\boldsymbol{x}} \tag{3.10}$$

where $\boldsymbol{R}$ is the rotation matrix and $\boldsymbol{t}$ is the translation vector.

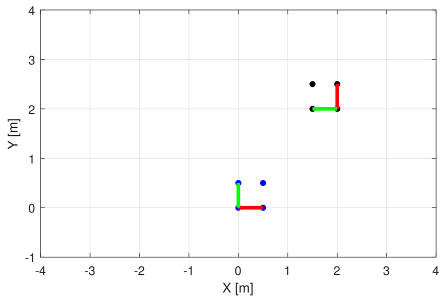


Fig. 3.4: Euler transformation example

3.1.3 Similarity transformation

The similarity transformation is an Euler transformation with the addition that space is equally scaled in all directions. It can be expressed as:

$$\bar{\boldsymbol{x}}' = \begin{pmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \bar{\boldsymbol{x}} \tag{3.11}$$

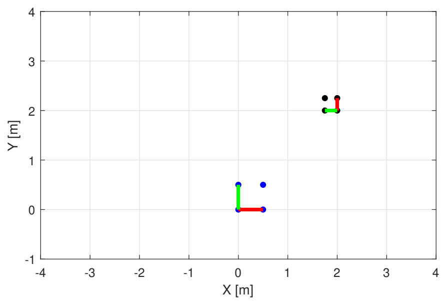


Fig. 3.5: Similarity transformation example

3.1.4 Affine transformation

An affine transformation adds up to the previous ones by including bi-directional shear terms in the transformation matrix. It is written as:

$$\bar{\boldsymbol{x}}' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \bar{\boldsymbol{x}} \tag{3.12}$$

and deforms space coordinates as seen in Fig. 3.6.

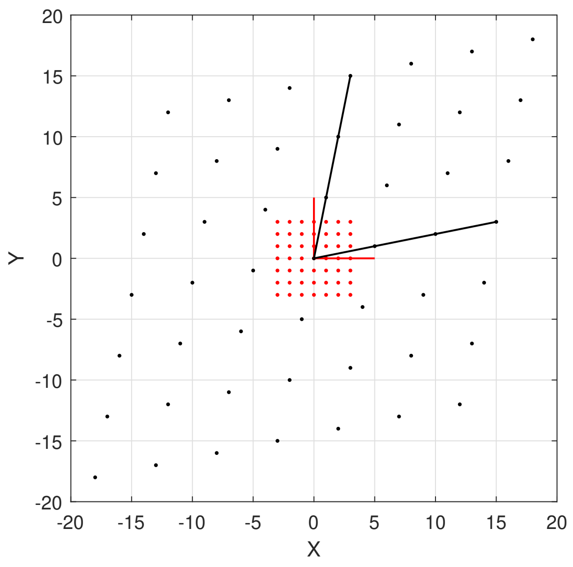


Fig. 3.6: Affinity transformation example

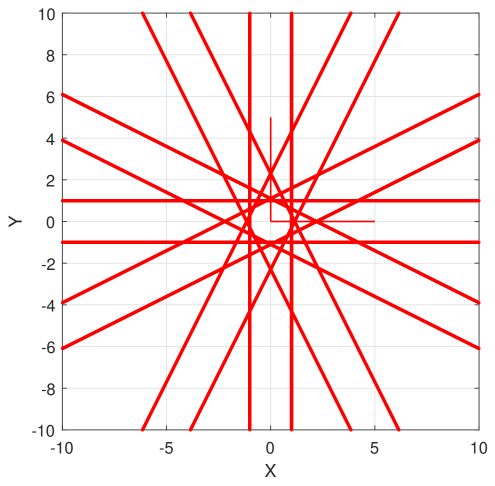
3.1.5 Projective transformation

A projective transformation is the most important one within projective geometry. Also referred to as homography, it maps points in one $\boldsymbol{R}^3$ plane to another one, or, in other terms, points in one image to another one. It is defined up to scale, meaning that it has only 8 degrees of freedom, since homogeneous coordinates are used:

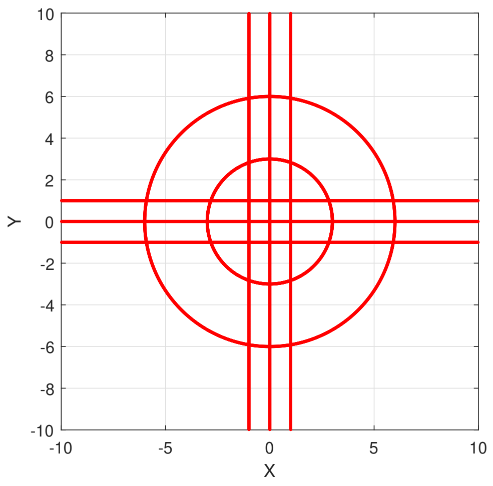
$$\tilde{\boldsymbol{x}}' = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tilde{\boldsymbol{x}} = \boldsymbol{H}\tilde{\boldsymbol{x}} \tag{3.13}$$

After this transformations, parallel lines will not be parallel anymore. Instead, parallel lines will converge to a unique ideal point, or point at infinity. The set of ideal points forms the ideal line. Example of this transformation are shown in Fig. 3.7.

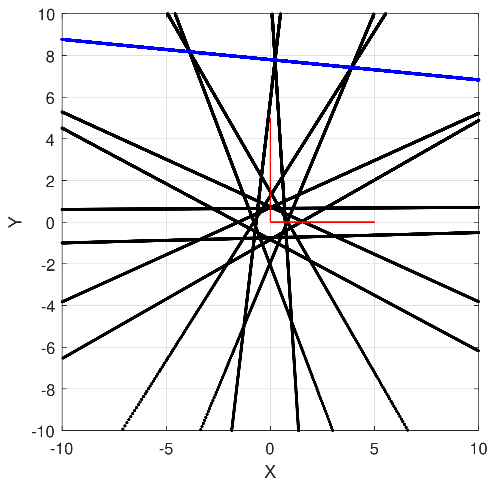
Formally 3D geometric transformation are identical to 2D ones. The only major difference is the definition of the rotation matrix $\boldsymbol{R}$, which can be defined either through euler angles or quaternions.



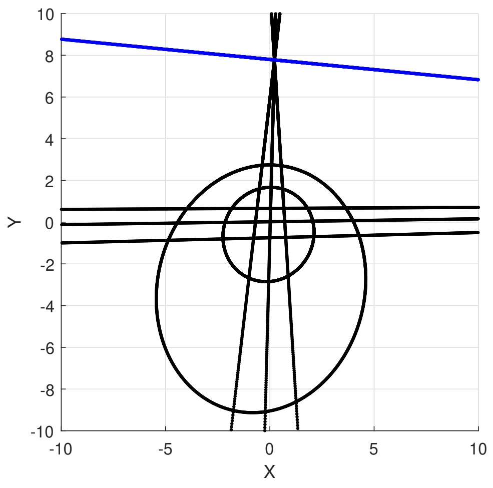
(a) Example 1, before transformation



(b) Example 2, before transformation



(c) Example 1, after transformation



(d) Example 2, after transformation

Fig. 3.7: Projective transformation examples, infinite horizon line in blue

3.2 Camera model and image formation

The simplest model for a camera is the pinhole camera model. Light rays converge in the camera center and points are projected onto the image plane (called also virtual image plane to differentiate it from the one of Fig. 3.13) at distance f from the center. The principal axis of the camera encounters the image plane in point p, the principal point. A generic point in space $\mathbf{X} = (X, Y, Z)^T$ is mapped into the $\mathbf{P}^2$ image plane to $\mathbf{x} = (fX/Z, fY/Z)^T$ as shown in Fig. 3.8.

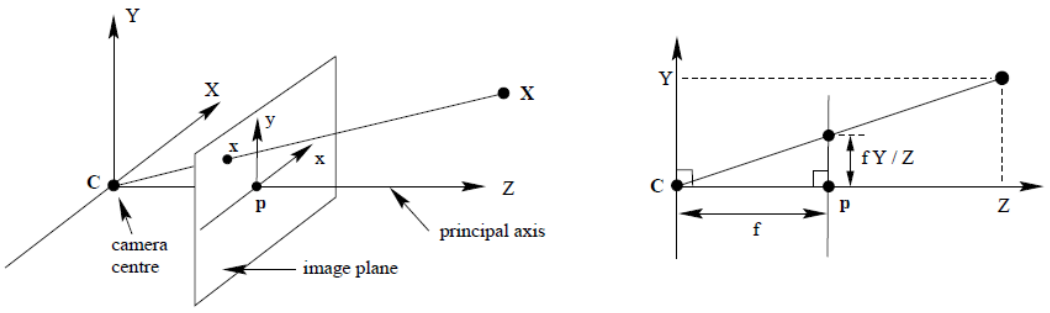


Fig. 3.8: Pinhole camera model [3]

In homogeneous coordinates, that mapping goes then from point $\mathbf{X} = (X, Y, Z, 1)^T$ to $\mathbf{x} = (fX, fY, Z)^T$ and can be easily written as a matrix:

$$\begin{pmatrix} f\mathrm{X} \\ f\mathrm{Y} \\ \mathrm{Z} \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix} \tag{3.14}$$

or:

$$\mathbf{x} = \boldsymbol{PX} \tag{3.15}$$

where $\boldsymbol{P}$ is the 3×4 the camera matrix. The axis of the image plane generally is not centered in the principal point, so two additional translational degrees of freedom for the image plane origin can be introduced in $\boldsymbol{P}$ as:

$$\mathbf{x} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix} \tag{3.16}$$

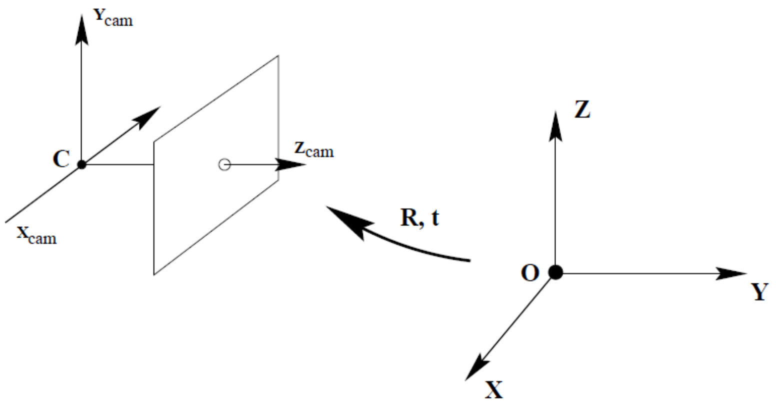


Fig. 3.9: Camera translation and rotation [3]

The previous expression for $\boldsymbol{P}$ matrix holds only if the camera reference frame and the world one are coincident. In order to take into account possible rotation and translation of the camera coordinate frame with respect to the world coordinate frame, an Euler transformation can be exploited as follows:

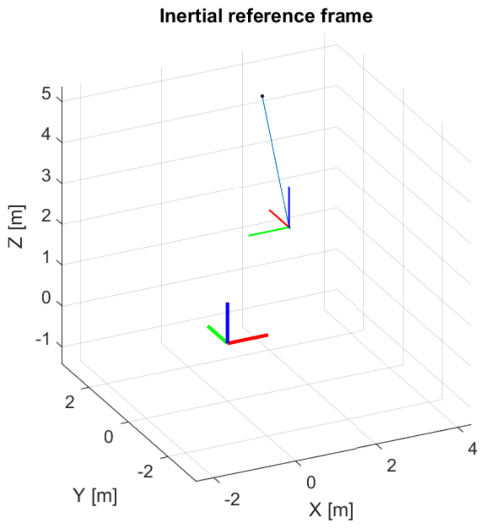
$$\boldsymbol{X}|_{camera} = \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RC} \\ 0 & 1 \end{bmatrix} \boldsymbol{X}|_{world} \tag{3.17}$$

Where $\boldsymbol{R}$ is the rotation matrix from world to camera frame and $\boldsymbol{C}$ is the inhomogeneous vector of the camera center in world coordinate frame. Denoting with $\boldsymbol{t} = -\boldsymbol{RC}$ the world center in camera frame the mapping becomes [3]:

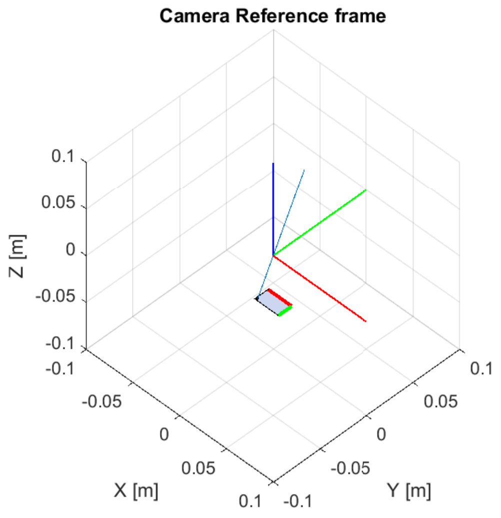
$$\mathbf{x} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} [\boldsymbol{R}|\boldsymbol{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{X}|_{world} = \boldsymbol{PX}|_{world} \tag{3.18}$$

$\boldsymbol{K}$ is referred to as the calibration matrix and takes into account intrinsic parameters of the camera. It describes the projective mapping from 3D camera frame to 2D image plane. The other matrix $[\boldsymbol{R}|\boldsymbol{t}]$ describes instead a 3D to 3D rigid transformation and for this reason it embeds extrinsic parameters only. For the pinhole camera there are 9 DOFs in total - 3 for rotation, 3 for translation, 2 for image axis translation, 1 for focal distance f.

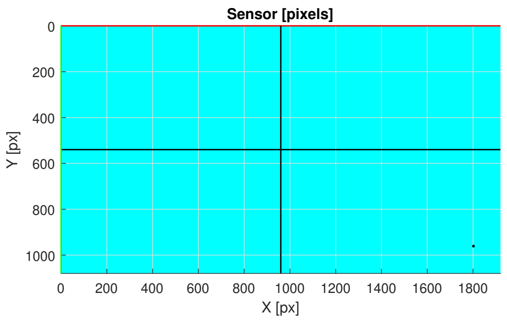
In Fig. 3.10 an example of a point projection on a generic sensor is shown. Moreover note that the parameters of the intrinsic matrix $\boldsymbol{K}$ must all have the same unit, pixels. For this reason f needs to be converted in pixels by multiplying it by the



(a) World reference frame



(b) Camera frame, image plane behind camera center

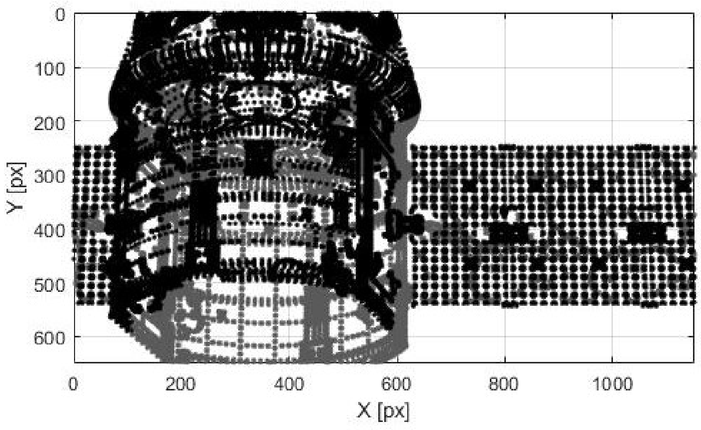


(c) Sensor reference frame

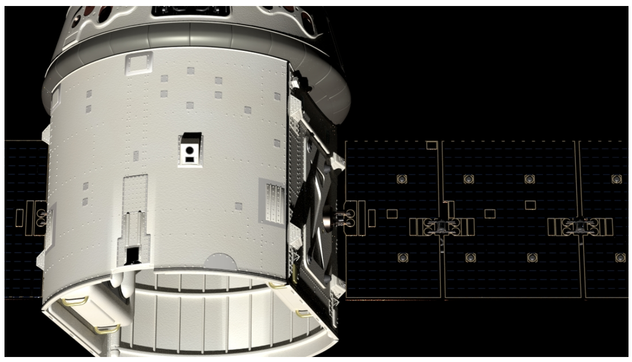
Fig. 3.10: Camera model examples

pixel density ($\frac{px}{mm}$) assuming square pixels while p_x, p_y can be taken as half image resolution in x and y directions.

In order to check the validity of the code developed with the theoretical notions discussed above, the image of a Dragon spacecraft has been simulated in Matlab environment using the pinhole camera model and in parallel has been rendered in Blender as in Fig. 3.11. Blender is a rendering software that has been exploited in this thesis to obtain realistic spacecraft images along with a list of important simulation parameters, such as the satellite position and attitude. More information is available in Section 6.3. In Fig. 3.11a single dots composing the mesh are shown and the visual comparison of the two pictures gives satisfying results. The spacecraft is distant roughly 30 meters and the camera has a focal length $f = 150$ [mm]. The sensor size is of 36×20.25 [mm] with an image resolution of 1152 by 648 pixels and standard 16:9 proportions. Camera parameters have been selected mainly by previous heritage as discussed for past missions in Section 1.1. Another criteria for the image resolution selection was the rendering time duration. It is indeed necessary that images are rendered in a reasonable time in Blender. This point will be crucial in the next sections, especially in Section 8.1, where the rendering software will be tasked with thousands of images to reproduce.



(a) Pinhole camera model image



(b) Blender rendered image

Fig. 3.11: Dragon spacecraft rendered images

3.3 Ray casting

Often throughout this thesis the operation of reconstructing the structure of the 3D scene given some 2D points is necessary. This is very useful for example to understand if some estimated scene 3D points are computed accurately with respect to the true 3D points. Indeed later in Section 7 the estimated structure of the scene is compared to the true one through this method. Ray casting acts by first casting rays from the camera center. Rays pass through the sensor 2D points and are then propagated in space. The intersection between these rays and every object in the scene must be verified in order to retrieve the corresponding 3D intersection point. The intersection operation is processed through an algorithm proposed by Möller and Trumbore (1997). More in-depth information and the original code can be found in [60].

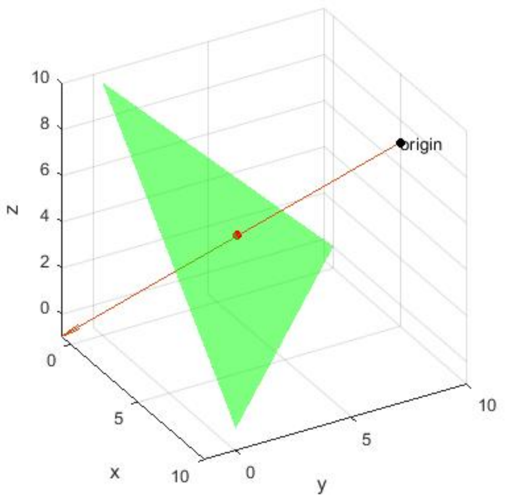


Fig. 3.12: Ray triangle intersection example

Ray casting can become quickly expensive as the number of mesh triangles increases in the scene structure. Some improvements can be introduced by checking intersection only with triangles whose normal is opposite to the ray direction. In this way a lot of unnecessary intersections are avoided. Another important improvement has been implemented by parallelizing the process. The effect of parallelization is enhanced on multi-core CPUs and multiple workers are deployed that check randomly-chosen ray/triangles intersections. The 3D point reconstruction can still take a large amount of time to process, depending on the triangle mesh number, going from some seconds to some minutes, however this method is not part of on-board visual navigation algorithms as it is used for validation purposes.

3.4 Analogies to photography and optics

Most imaging software use a mixture of terminology that comes from projective geometry theory, photography and optics theory. For this reason parameters of a camera related to this fields are mentioned and described next.

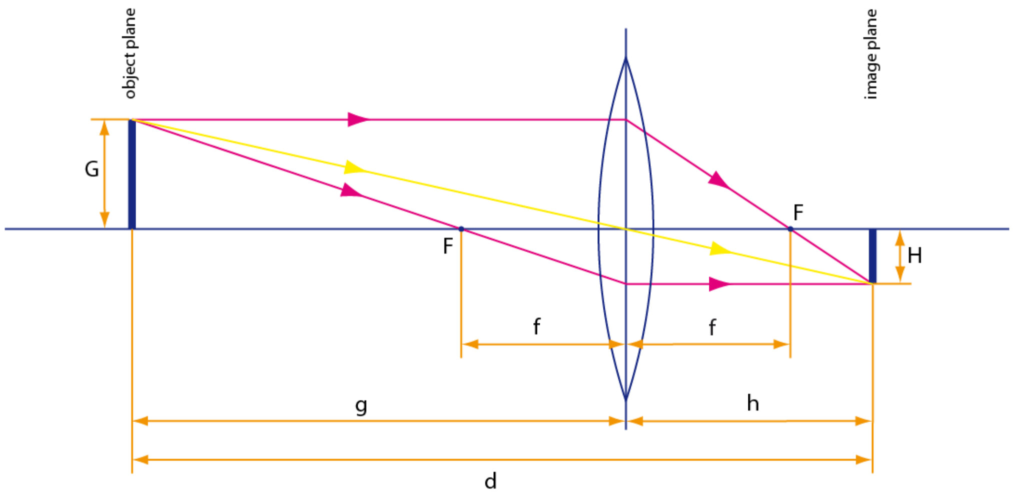


Fig. 3.13: Converging lens representation [4]

First of all, a camera lens can be generally approximated as a converging lens which collects light rays to the focus. The lens equation reads [4]:

$$\frac{1}{f} = \frac{1}{g} + \frac{1}{h} \tag{3.19}$$

Where related quantities are shown in Fig. 3.13. The camera focal length f, as seen before, is the distance between the camera center and the image plane. For real cameras, that translates as the distance between the lens of the camera and the imaging sensor. In Fig. 3.8 the image plane is shown along Z axis, however the imaging sensor is actually placed behind the camera center along negative Z axis. Focus distance d is:

$$d = g + h \tag{3.20}$$

where g and h must be such that the lens equation is satisfied. From the focal length an important parameter can be computed, the FOV or field of view. It is the angular extent of a given scene that is imaged by a camera, as in Fig. 3.14.

FOV can be computed as:

$$FOV = 2\arctan\frac{d_{sensor}}{2f} \tag{3.21}$$

where d_{sensor} is the size of the imaging sensor, generally expressed in millimeters. From Eq. (3.21) it results that FOV depends on the camera focal length too. The bigger the focal length, the narrower the FOV.

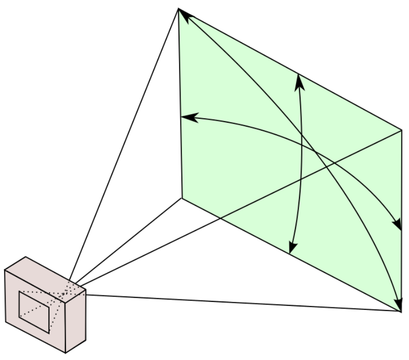


Fig. 3.14: Camera field of view

Another quantity of interest is the f-number. It is defined as the ratio between the focal lenght f and the aperture a, the size of the lens opening, as:

$$f_{number} = \frac{f}{a} \tag{3.22}$$

This parameter is very important as it is an absolute measure of how much light enters the camera lens. Indeed aperture by itself is not enough. It is true that a big aperture will gather more light with respect to a smaller one, especially against a pinhole camera, however also the spherical rectangle view of the camera must be taken into account, by means of the FOV or the focal length. For example, a lens with $f = 100\,[mm]$ and $a = 25\,[mm]$ will gather as much light as a $f = 200\,[mm]$ and $a = 50\,[mm]$ one, despite the second having double the aperture of the first, since their f numbers are the same, f/4.

In summary, in this chapter important theory topics of projective geometry have been discussed, which will be extensively applied in the V-SLAM and DL, deep learning, sections. Geometric transformations are useful to better understand computer vision algorithms and the Euler one in particular is essential for changes in coordinate systems. Moreover the image formation and ray casting method are the building blocks for ground truth estimation of the DL and V-SLAM algorithm respectively.

4 Computer Vision

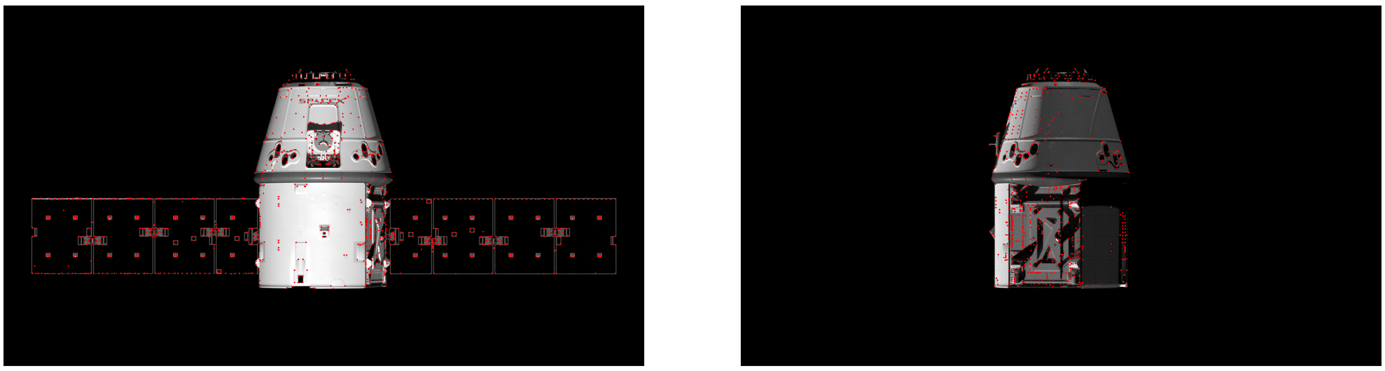
Computer vision is the engineering field that seeks to understand and extract various types of information from images. Methods and algorithms used in V-SLAM pipelines are presented in this chapter. V-SLAM, which stands for Visual Simultaneous Localization and Mapping, is the problem of computing the trajectory and 3D structure of a scene from visual data. The general structure of a V-SLAM algorithm starts from an initial 3D map computation which is later used for navigation. In order to do so, features, special points of an object which are easy to recognize, can be exploited. They are extracted and matched/tracked so that two images with a set of point tracks between them are obtained. Two view geometry, or epipolar geometry, is then applied to compute the rotation and translation of the camera in time: an important matrix, called fundamental/essential matrix, is estimated from the point tracks and is decomposed into the relative rotation matrix and translation vector. Triangulation allows to compute the initial map, which is later navigated by continuously tracking features and solving the PnP, Perspective n-Point, problem. Finally an optimization method called Bundle Adjustment is presented. Each of these topics is discussed and analyzed individually in the following sections.

4.1 Feature Detection

The first step in a V-SLAM algorithm is information retrieval from the input images. The main approaches to this problem rely on feature detectors, but other solutions exists too. Direct methods for example aim to use information from every pixel in the images, while feature detectors only use accurately selected points. More complex structures can be found too, such as lines and edges, however the increased complexity can result in an higher computational cost and lower accuracy. For this reason, feature detectors have been preferred.

Features are special points in an image which stand out among the other ones and are easy to relocate in different images of the same scene. Feature detectors need to be able to locate them accurately without being affected by external manipulation. An ideal detector should be invariant to scale and rotation changes, to noise, to illumination changes, to affine or projective transformations and still be computationally efficient. Several feature detectors have been developed in literature with different properties between each other. Corner detectors have been prioritized due to the kind of object, an artificial one, that is studied. Among those ones there are the minimum eigenvalue algorithm detector, Harris detector, FAST, BRISK and ORB. It is out of the scope of this thesis to discuss each one of them, however a review can be found in [61].

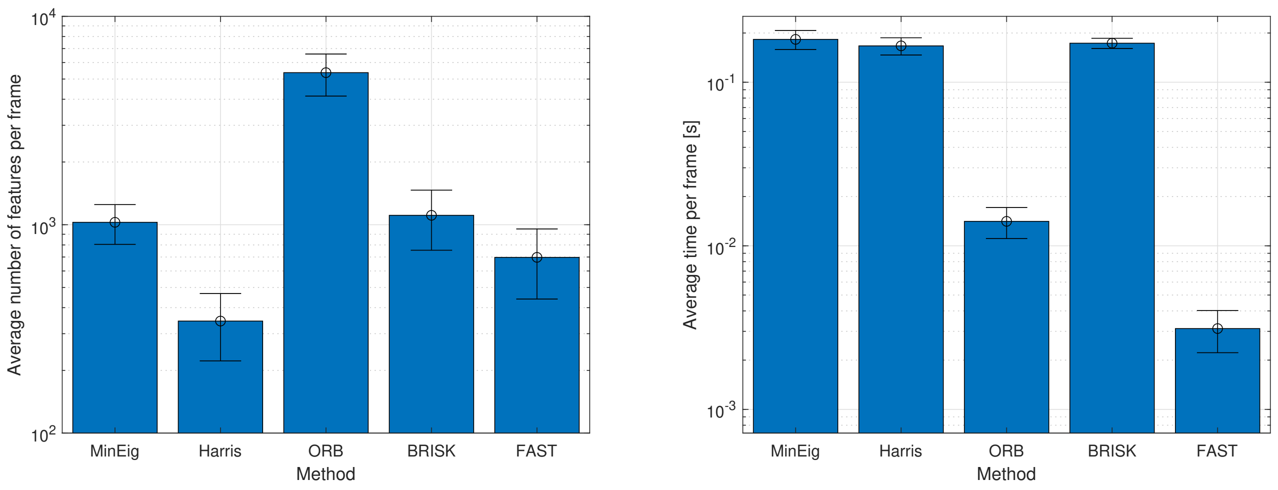
Statistical characteristics of the detectors have been explored by applying them over a number of synthetic spacecraft images depicting a rotation of 90 degrees around a deputy satellite. The first and last frames are shown below.



(a) First frame (b) Last frame

Fig. 4.1: Example of minimum eigenvalue detector

In Fig. 4.2 the mean and the standard deviation of the number of features and execution time are reported for every detector. All the investigations performed in this thesis are executed on a Ryzen R7 3700X and a RTX 2070 Super. The average number of features for all the algorithms is fairly consistent along the frames while the execution time is noticeably smaller for FAST and ORB detector.



(a) Average feature number per frame (b) Average execution time per frame

Fig. 4.2: Statistics for feature detection

4.2 Feature Matching and Tracking

After finding good features in an image, the subsequent step is to try to find them in other frames. Two approaches are commonly used: feature matching or feature tracking.

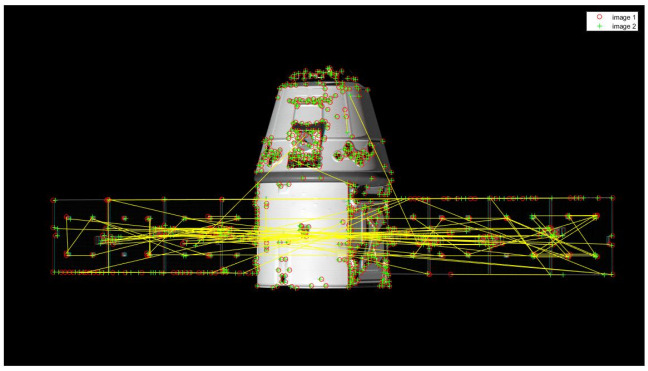
4.2.1 Feature Matching

Feature matching is the process through which descriptors are extracted for each detected feature in two different frames and are then compared. The ones that are most closely related will then get matched to create a track. A high number of descriptors are available in literature and their purpose is to compress key information of a feature point generally into vectors.

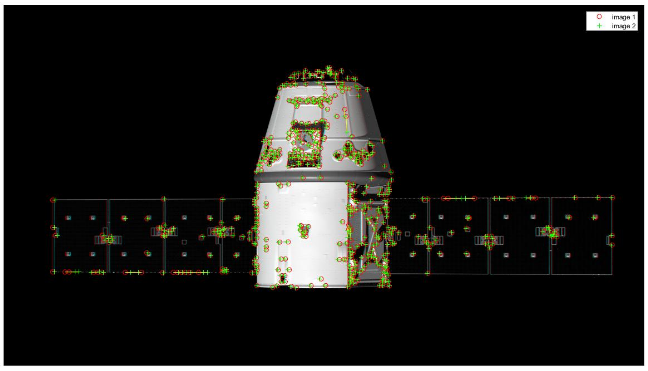
In Fig. 4.3a an example of feature matching is presented. Features detected through the minimum eigenvalues algorithm are found in two frames and then matched. The result is not accurate as a lot of false tracks are present, due to the fact that many points can be very similar to each others, and they must be removed. For this purpose a popular algorithm that is used is RANSAC [62].

RANSAC, which stands for RANdom Sample Consensus, is a simple and efficient algorithm for outliers detection. It starts by randomly selecting a subset of the input data and fitting it with a model, that could be for example a least square linear regression. A threshold is then defined to compare the remaining input data with respect to the model. Some will be marked as inliers while other ones will be outliers. This procedure is repeated up to a number of iterations, after which the model with the most number of inliers is selected between all the computed ones. RANSAC is implemented in Matlab but a similar algorithm called MSAC has been used for outlier removal. The main difference between MSAC and RANSAC is that the former chooses the best model by scoring inliers based on their fitness to the model, while the latter simply uses the cardinality of the inliers group. The authors conclude that this simple modification grants a modest to high benefit to the method with insignificant added computational cost [63].

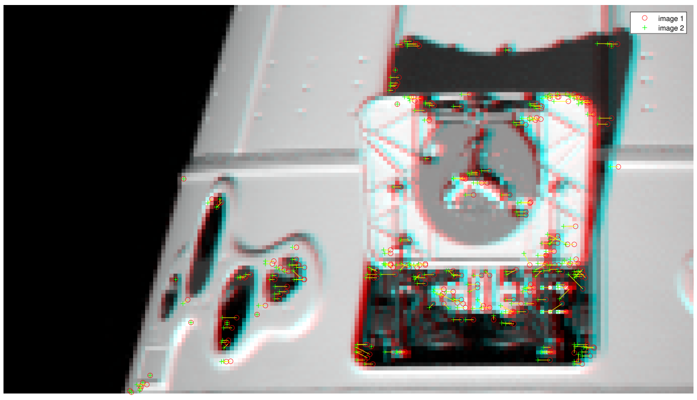
In Fig. 4.3 an example of this process is provided. Features are extracted in two different frames and then matched. The outlier removal algorithm, MSAC, is then used to detect false matching. When looking at Fig. 4.3b, even though the quality of the matching seems to be good at first, various tracks are not coherent with the spatial movement of the object when zoomed as in Fig. 4.3c. Further processing would be needed to remove the inaccurate ones as they must not appear when estimating the fundamental or essential matrix. Errors that would appear during the matching process would be later amplified and threaten the integrity of the entire SLAM algorithm.



(a) Frame after MinEig feature matching



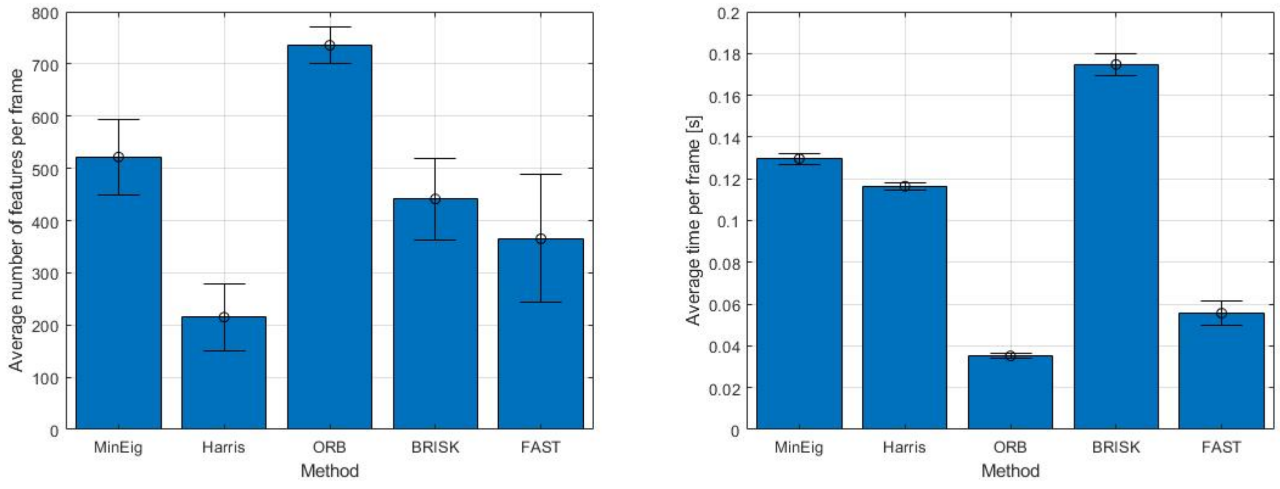
(b) Frame after MSAC



(c) Zoom of ORB feature matching after MSAC

Fig. 4.3: Example of feature matching

Finally in Fig. 4.4 feature matching with outlier removal for the same image sequence used in Section 4.1 is shown. Feature detection has been capped at 1000 features per image and most of them are lost during frames. The average execution time is roughly between 0.04 and 0.18 seconds as in Fig. 4.4b. Due to the high variability in the feature matching step, this process has not been deemed accurate enough to be used for space applications. For this reason feature tracking has been investigated further and is presented in the next section.



(a) Average feature number per frame (b) Average execution time per frame

Fig. 4.4: Statistics for feature matching

4.2.2 Feature Tracking

Feature tracking is another possibility that can be used for finding 2D correspondences between points in images. One of the characteristics of feature tracking is that it is generally faster than feature matching and is a popular choice for online applications. In particular in this thesis the KLT tracker has been used [64].

This tracker can be derived by Lucas-Kanade image alignment method. An image can be aligned with another one by first defining a cost function as:

$$\boldsymbol{E}(\boldsymbol{u}) = \sum_i \left(f_1(\boldsymbol{x}_i + \boldsymbol{u}) - f_0(\boldsymbol{x}_i)\right)^2 \tag{4.1}$$

where $f(\boldsymbol{x})$ is a generic image function, $\boldsymbol{x}$ is a vector containing the two x and y pixel coordinates of every pixel and $\boldsymbol{u}$ is the optical flow, a parameter that allows to shift the first image by $(u, v)^T$. By finding the optimal value for $\boldsymbol{u}$, the cost function will be minimized.

A fast solution of the minimization of $\boldsymbol{E}$ is needed for online applications. It can be found by linearizing the cost function around an initial guess of $\boldsymbol{u}$ as:

$$\boldsymbol{E}(\boldsymbol{u} + \Delta\boldsymbol{u}) = \sum_i \left(f_1(\boldsymbol{x}_i + \boldsymbol{u} + \Delta\boldsymbol{u}) - f_0(\boldsymbol{x}_i)\right)^2 \tag{4.2}$$

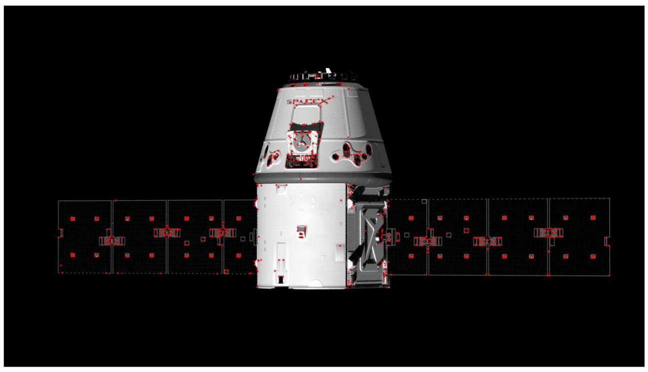
and by developing the Taylor series expansion up to the first order:

$$\boldsymbol{E}(\boldsymbol{u} + \Delta \boldsymbol{u}) = \sum_i \left(f_1(\boldsymbol{x}_i + \boldsymbol{u}) + \nabla f_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta \boldsymbol{u} - f_0(\boldsymbol{x}_i)\right)^2 \tag{4.3}$$

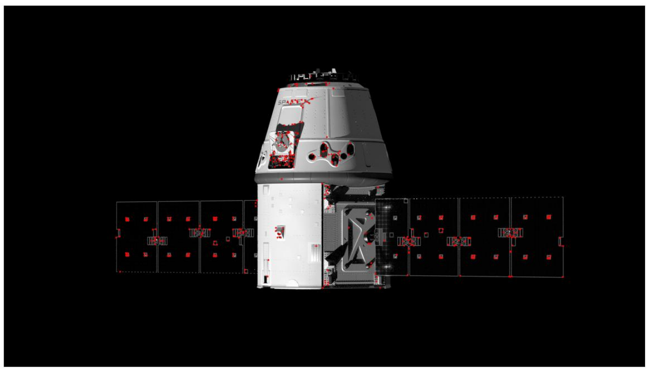
The goal of minimizing $\boldsymbol{E}(\boldsymbol{u} + \Delta \boldsymbol{u})$ can be reached by imposing $\frac{\partial \boldsymbol{E}(\boldsymbol{u}+\Delta \boldsymbol{u})}{\partial \Delta u} = 0$. This leads to a linear system in the form $\boldsymbol{A}\Delta \boldsymbol{u} = -\boldsymbol{b}$ where $\boldsymbol{A}$ is the discrete structure tensor of f_1 and $\boldsymbol{b}$ is composed by the image derivatives in space and time [65]. In this way a very fast tracking algorithm is obtained. Due to the linearization approximation, KLT tracker assumes that pixel intensity values do not change rapidly between frames and pixel motion is small.

This basic KLT tracker can be made more accurate and robust at the cost of increasing complexity. For example a pyramidal implementation can be exploited to make feature tracking invariant to the scale of the features themselves [66]. Moreover an iterative and windowed method to deal with feature tracking is generally adopted, and a different mathematical approach can be derived to describe feature motions different than simple translation too [67].

In Fig. 4.5 an example of tracks found by the KLT tracker are shown. The image sequence is still the same as for the previous paragraphs, with 91 images linearly covering angles from 0 to 90 deg. The tracking for all the image set is shown in Fig. 4.6. Different feature detectors are used in the first frame to detect a maximum of 1000 points, after which KLT is used to track them. As expected some features are lost while moving around the target spacecraft as in Fig. 4.6a. Execution times shown in Fig. 4.6b are very fast, in the order of milliseconds; feature points detected with Harris appear to be the fastest, however they are the fewest too.

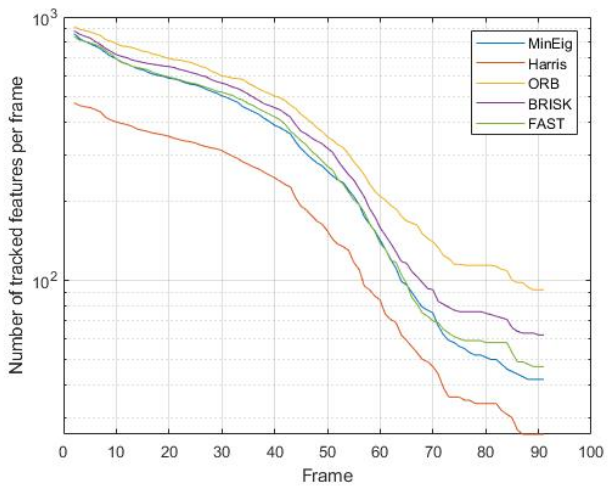


(a) 20 degrees view

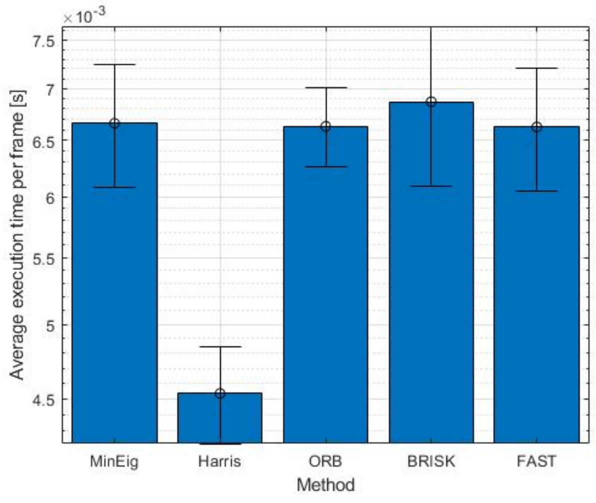


(b) 40 degrees view

Fig. 4.5: Example of KLT tracking with Min. Eig. feature points



(a) Tracked feature number per frame



(b) Average time per frame

Fig. 4.6: Example of KLT traking

4.3 Epipolar geometry

The term epipolar geometry is used to refer to the intrinsic projective geometry properties that arise between two views. When a camera images a point $\boldsymbol{X}$ in world frame, it is projected in the image plane to $\boldsymbol{x}_1$ as seen in Section 3.2. In Fig. 4.7 $\boldsymbol{O}_1$, $\boldsymbol{O}_2$ are the camera centers, $\boldsymbol{B}$ is the baseline while $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ are the epipoles, the projection of the camera centers onto the image planes. The epipolar lines are a family of lines that pass through one epipole, so that if the camera moves to another position in space or a second camera takes an image of the same point $\boldsymbol{X}$, the projection $\boldsymbol{x}_2$ is constrained to be fixed on the epipolar line $\boldsymbol{l}_2$.

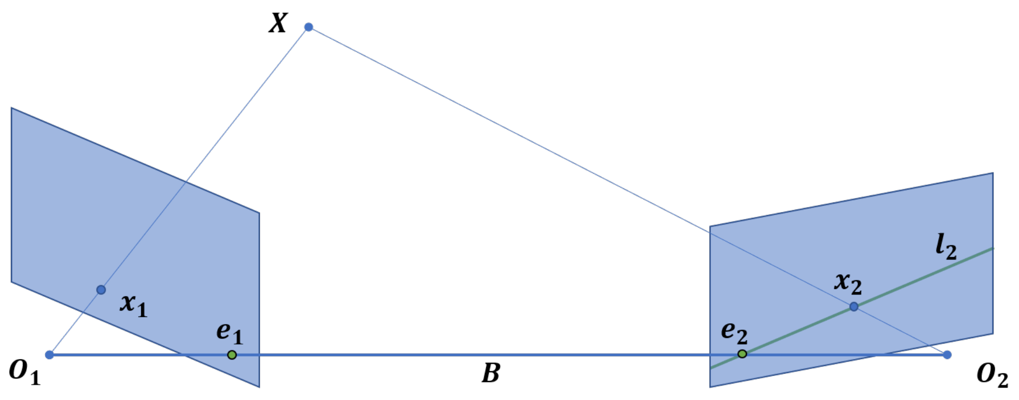


Fig. 4.7: Two view geometry representation

This constraint can be expressed in mathematical formulation through a matrix $\boldsymbol{E}$, called the essential matrix, which maps points in one image to the corresponding epipolar lines in the other one:

$$\boldsymbol{E}\hat{\boldsymbol{x}_1}^T = \boldsymbol{l}_2 \tag{4.4}$$

Where the $\hat{\boldsymbol{x}} = \boldsymbol{K}^{-1}\boldsymbol{x}$ notation refers to normalized coordinates, in which a camera matrix $\boldsymbol{P}$, such that $\hat{\boldsymbol{x}} = \boldsymbol{P}, \boldsymbol{X}$ becomes $\boldsymbol{K}^{-1}\boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}] = \boldsymbol{I}[\boldsymbol{R}|\boldsymbol{t}]$. Moreover, since it must hold that $\hat{\boldsymbol{x}_1}^T\boldsymbol{l}_1 = \hat{\boldsymbol{x}_2}^T\boldsymbol{l}_2 = 0$, the epipolar constraint is formulated as:

$$\hat{\boldsymbol{x}_2}^T\boldsymbol{E}\hat{\boldsymbol{x}_1} = 0 \tag{4.5}$$

The demonstration for this last equation leverages on various geometric properties of the elements of Fig. 4.7, and in particular the matrix $\boldsymbol{E}$ is defined as:

$$\boldsymbol{E} = \boldsymbol{R}[\boldsymbol{t}]_\times \tag{4.6}$$

Where $\boldsymbol{R}$ is the rotation matrix from camera frame 2 to 1 and $\boldsymbol{t}$ is the translation vector of frame 2 written in frame 1. This expression can be used to recover analytically the expression of $\boldsymbol{E}$ given the two camera poses as well as to retrieve the extrinsic parameters of the relative camera pose if an estimate of $\boldsymbol{E}$ is available.

The generalization of the essential matrix is the fundamental matrix $\boldsymbol{F}$. The main difference between the two is that $\boldsymbol{F}$ estimation does not require calibrated cameras and thus a priori knowledge of $\boldsymbol{K}$. It relates to the essential matrix as $\boldsymbol{E} = \boldsymbol{K}^T\boldsymbol{F}\boldsymbol{K}$. For this reason $\boldsymbol{F}$ has more degrees of freedom with respect to $\boldsymbol{E}$, a total of 7 degrees of freedom against 5 for $\boldsymbol{E}$. The previous equations for the essential matrix hold too for the fundamental matrix using non-normalized points: for example the epipolar constraint can be written as $\boldsymbol{x}_2{}^T\boldsymbol{F}\boldsymbol{x}_1 = 0$.

4.4 Fundamental and essential matrix estimation

The fundamental matrix $\boldsymbol{F}$ and essential matrix $\boldsymbol{E}$ can be estimated from a set of point correspondences in two images. Due to the fact that the essential matrix has a lower number of degrees of freedom with respect to the fundamental one, it can also be extracted from a lower number of correspondences with respect to $\boldsymbol{F}$ even though the algorithm complexity increases.

The simplest algorithm for $\boldsymbol{F}$ estimation is the 8 points normalized algorithm. By recalling that, given two set of corresponding points, it must hold for each one of them that $\boldsymbol{x}_2^T \boldsymbol{F} \boldsymbol{x}_1 = 0$, one linear equation in $\boldsymbol{F}$ components can be obtained per points pair. By rearranging the set of equations in a linear system it is possible to obtain [3]:

$$\boldsymbol{Af} = \begin{bmatrix} x_2 x_1 & x_2 y_1 & x_2 & y_2 x_1 & y_2 y_1 & y_2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_2^n x_1^n & x_2^n y_1^n & x_2^n & y_2^n x_1^n & y_2^n y_1^n & y_2^n & x_1^n & y_1^n & 1 \end{bmatrix} \boldsymbol{f} = 0 \tag{4.7}$$

where $\boldsymbol{f}$ contains the components of $\boldsymbol{F}$ and $\boldsymbol{n}$ is the index for the n-th correspondence. Unfortunately when points couples are not exact, which can happen during feature tracking or matching, a least square solution must be sought for that system. By referring to Appendix A, $||\boldsymbol{Af}||$ subjected to $||\boldsymbol{f}||= 1$ can easily be solved through an SV decomposition of $\boldsymbol{F}$. In this case the solution $\boldsymbol{f}$ will be the last column of the right singular vector matrix $\boldsymbol{V}$. This procedure will not however guarantee that $det(\boldsymbol{F}) = 0$, which comes from the mathematical definition of $\boldsymbol{F}$, and is the condition responsible for having all the epipolar lines meet in the epipole. Indeed an epipole $\boldsymbol{e}_1$ belongs to any epipolar line, so that $\boldsymbol{e}_1^T \boldsymbol{l}_1 = 0$. By exploiting the fact that $\boldsymbol{l}_1 = \boldsymbol{F}\boldsymbol{x}_2$ the previous expression becomes $(\boldsymbol{e}_1^T \boldsymbol{F})\boldsymbol{x}_2 = 0 \; \forall \boldsymbol{x}_2$ or $\boldsymbol{e}_1^T \boldsymbol{F} = 0$. This condition only holds if one eigenvalue of $\boldsymbol{F}$ is zero, or equivalently if $det(\boldsymbol{F}) = 0$. It is possible to demonstrate that, by imposing to zero the last singular value of $\boldsymbol{F}$, the Frobenius norm $||\boldsymbol{F} - \boldsymbol{F}'||$ with $det(\boldsymbol{F}') = 0$ is minimized.
Finally the normalization step at the beginning of the algorithm is fundamental to retrieve good results, and it refers to the choice of the coordinates in the image, since the algorithm is sensible to its changes. Points are first translated so that their centroid is at the origin and scaled so that their average distance from the origin is equal to $\sqrt{2}$.

In general the 8 point algorithm is accurate when points correspondences are very precise, which might not always happen when noise is present too. In this case, noise is assumed to be Gaussian and a maximum likelihood method is adopted. The cost function to be minimized is defined as the squared sum of the reprojection errors in both images:

$$\sum_n d(\boldsymbol{x}_1^n, \hat{\boldsymbol{x}}_1^n)^2 + d(\boldsymbol{x}_2^n, \hat{\boldsymbol{x}}_2^n)^2 \tag{4.8}$$

where d defines the geometric distance. $\boldsymbol{x}_1^n, \boldsymbol{x}_2^n$ are the measured correspondences generally found through feature matching or tracking while $\hat{\boldsymbol{x}}_1^n, \hat{\boldsymbol{x}}_2^n$ are the estimated true correspondences that satisfy exactly the epipolar constraint $\hat{\boldsymbol{x}}_1^{n,T}\boldsymbol{F}\hat{\boldsymbol{x}}_2^n = 0$. In this algorithm, called the Gold Standard method, an initial estimate of $\hat{\boldsymbol{F}}$ is obtained through the previously shown linear algorithm. From it, the two camera matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ are computed and 3D points $\hat{\boldsymbol{X}}$ can be obtained through triangulation. The 2D points correspondences that satisfies exactly the epipolar constraint are $\hat{\boldsymbol{x}}_1^n = \boldsymbol{P}_1\hat{\boldsymbol{X}}_1^n$ and $\hat{\boldsymbol{x}}_2^n = \boldsymbol{P}_2\hat{\boldsymbol{X}}_2^n$. The cost function can then be minimized through a Levenberg-Marquardt algorithm as in Appendix C thanks to its fast execution times.

4.5 Extrinsic parameters from essential matrix

The essential matrix $\boldsymbol{E}$ embeds the information about the relative translation and rotation of two different camera poses. It is thus possible to reversely extract the relative extrinsic parameters when an estimation of it is available. However the camera matrix can be recovered up to scale, as the essential matrix is an homogeneous one. It can be demonstrated that, for a SV decomposition up to scale of $\boldsymbol{E} = \boldsymbol{U}diag(1{,}1{,}0)\boldsymbol{V}^T$ and $\boldsymbol{P}_1 = [\boldsymbol{I}|\boldsymbol{0}]$ four possible choices for $\boldsymbol{P}_2$ are present [3].

The translation vector can be recovered by solving the linear system $\hat{\boldsymbol{t}}^T\boldsymbol{E} = 0$, since by definition $\hat{\boldsymbol{t}}^T[\hat{\boldsymbol{t}}]_\times = 0$. The solution will then be the last column of $\boldsymbol{U}$ and is up to scale, so that the sign is not known too. With respect to the rotation matrix, it is possible to demonstrate that $[\hat{\boldsymbol{t}}]_\times$ can be decomposed as $\boldsymbol{U}diag(1{,}1{,}0)\boldsymbol{W}\boldsymbol{U}^T$ or equally $\boldsymbol{U}diag(1{,}1{,}0)\boldsymbol{W}^T\boldsymbol{U}^T$ with:

$$\boldsymbol{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Finally since $\boldsymbol{E} = [\boldsymbol{t}]_\times\boldsymbol{R} = \boldsymbol{U}\boldsymbol{Z}\boldsymbol{U}^T\boldsymbol{R}$, the expression for $\boldsymbol{R}$ can be:

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T \quad \text{or} \quad \boldsymbol{R} = \boldsymbol{U}\boldsymbol{W}^T\boldsymbol{V}^T \tag{4.10}$$

with:

$$\boldsymbol{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.11}$$

Since the scale of $\boldsymbol{E}$ is not defined, the sign in not set too and both $\boldsymbol{R}$ expressions are valid. With two possibilities for the translation vector and two for the rotation matrix a total of four possible solutions are retrieved and shown in Fig. 4.8. It is

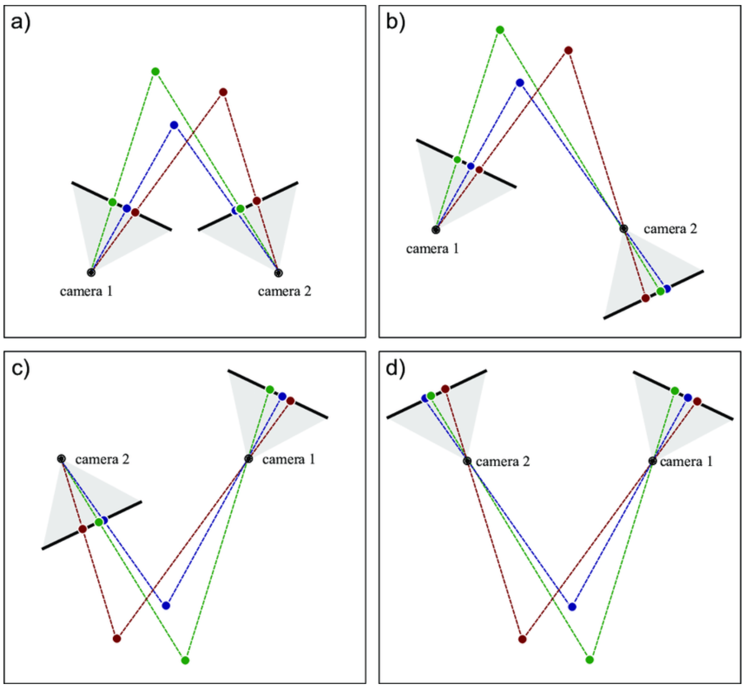


Fig. 4.8: Camera ambiguity in reconstruction [5]

trivial to understand which is the correct one by imposing that the 3D points must lay in front of both cameras.

4.6 Triangulation

The aim of triangulation methods is to estimate the position of the world point $\boldsymbol{X}$ given two image points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and the respective camera matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. In real applications image points are affected by noise and will not satisfy exactly geometrical relations, so that optimal algorithms are needed.

The easiest methods for triangulation are the linear ones, which are not optimal. The system of equations is obtained by expanding the expressions $\boldsymbol{x}_1 = \boldsymbol{P}_1\boldsymbol{X}$ and $\boldsymbol{x}_2 = \boldsymbol{P}_2\boldsymbol{X}$. The corresponding system is $\boldsymbol{AX} = 0$ with $\boldsymbol{A}$ equal to:

$$\boldsymbol{A} = \begin{bmatrix} x_1\boldsymbol{P}_1^{3T} - \boldsymbol{P}_1^{1T} \\ y_1\boldsymbol{P}_1^{3T} - \boldsymbol{P}_1^{2T} \\ x_2\boldsymbol{P}_2^{3T} - \boldsymbol{P}_2^{1T} \\ y_2\boldsymbol{P}_2^{3T} - \boldsymbol{P}_2^{2T} \end{bmatrix} \tag{4.12}$$

where the notation $\boldsymbol{P}^n$ indicates the rows of $\boldsymbol{P}$. If the system is solved through SVD, then the method is referred as DLT. The solution is the singular vector corresponding to the singular value of $\boldsymbol{A}$.

Triangulation algorithm however must take into account that the pair of image points will be affected by noise and geometric relations will not be exactly solved, so that the projected rays in 3D world will not meet. A cost function is then defined as:

$$C(\boldsymbol{x}_1, \boldsymbol{x}_2) = d(\boldsymbol{x}_1, \hat{\boldsymbol{x}}_1)^2 + d(\boldsymbol{x}_2, \hat{\boldsymbol{x}}_2)^2 \tag{4.13}$$

with $\hat{\boldsymbol{x}}$ indicating the exact image point and $d(\boldsymbol{x}, \hat{\boldsymbol{x}})$ the distance. By denoting with $\boldsymbol{l}_1$ the epipolar line in which $\hat{\boldsymbol{x}}_1$ lies, the squared distance $d(\boldsymbol{x}, \hat{\boldsymbol{x}})$ can also be written as $d(\boldsymbol{x}_1, \boldsymbol{l}_1)$. $d(\boldsymbol{x}_1, \boldsymbol{l}_1)$ represents the perpendicular distance of point $\boldsymbol{x}$ to the unknown epipolar line $\boldsymbol{l}$. The cost function is rewritten by parametrizing the epipolar lines families with parameter t as:

$$C(\boldsymbol{x}_1, \boldsymbol{x}_2, t) = d(\boldsymbol{x}_1, \boldsymbol{l}_1(t))^2 + d(\boldsymbol{x}_2, \boldsymbol{l}_2(t))^2 \tag{4.14}$$

This cost function can be minimized using numerical methods such as Levenberg Marquardt as in Appendix C, however the problem can be reduced to the solution of a 6-th order polynomial. A traslation is performed to place points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in the image origins $(0,0,1)^T$ and epipoles in $(1,0,f_1)$ and $(1,0,f_2)$. The fundamental matrix can be parametrized as a function of variables $\boldsymbol{F} = \boldsymbol{F}(a, b, c, d, f_1, f_2)$. An epipolar line $\boldsymbol{l}(t)$ passes through the epipole of the corresponding image and point $(0, t, 1)$. The distance from the origin to the line $\boldsymbol{l}_1$ is then:

$$d(\boldsymbol{x}_1, \boldsymbol{l}_1(t))^2 = \frac{t^2}{1 + (tf)^2} \tag{4.15}$$

The epipolar line $\boldsymbol{l}_2$ is computed as $\boldsymbol{l}_2(t) = \boldsymbol{F}(0, t, 1)^T$ and is equal to:

$$\boldsymbol{l}_2(t) = (-f_2(ct + d), at + b, ct + d)^T \tag{4.16}$$

so that the perpendicular distance from the origin becomes:

$$d\left(\boldsymbol{x}_2, \boldsymbol{l}_2(t)\right)^2 = \frac{(ct + d)^2}{(at + b)^2 + f_2^2(ct + d)^2} \tag{4.17}$$

and the cost function becomes:

$$C(t) = \frac{t^2}{1 + (tf)^2} + \frac{(ct + d)^2}{(at + b)^2 + f_2^2(ct + d)^2} \tag{4.18}$$

There are a total of six roots which can be computed numerically. After finding the optimum value for t, the corresponding epipolar lines can be found and so also the points $\hat{\boldsymbol{x}}_1$ and $\hat{\boldsymbol{x}}_2$. These points can be given as input to the linear triangulation algorithm to compute more accurately the 3D points in world frame.

4.7 Perspective n-Point Problem

The perspective n-point problem, or PnP, is the problem of estimating the camera matrix of two view geometry when the 3D points and corresponding 2D points in both images are given as well as camera parameters.

The simplest form of the PnP problem is expressed by expanding the relation $\boldsymbol{x} = \boldsymbol{PX}$:

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{4.19}$$

and casting it into a linear system. This is obtained by taking the fractions $\tilde{x}/\tilde{w}$ and $\tilde{y}/\tilde{w}$ in Eq. (4.19) and isolating the m_{ij} components as:

$$\begin{pmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y \end{pmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ \dots \end{bmatrix} \tag{4.20}$$

By adding at least $n > 6$ correspondences the linear system $\boldsymbol{Am} = 0$ can be solved through SV decomposition methods as detailed in Appendix A [68].

Other formulation of the PnP problem are available that solve it in different ways. When $n = 3$, the P3P problem is formulated in its minimal form. This formulation is provided in Matlab and is joined with RANSAC too for outlier removal. There are up to four solutions that come from the linear system:

$$\begin{cases} Y^2 + Z^2 - YZp - b^2 = 0 \\ Z^2 + X^2 - XZq - c^2 = 0 \\ X^2 + Y^2 - XYr - a^2 = 0 \end{cases} \tag{4.21}$$

An extensive description of the problem is provided in [69]. Another formulation can be adopted when $n > 3$ which has been named EPnP, which stands for efficient PnP.

The first step consists in generating four control points $\boldsymbol{c}$ expressed as homogeneous vectors that are used as a basis to express all other 3D world points. By using α_{ij} as weights, the linear combination is written as:

$$\boldsymbol{X}_i|_{world} = \sum_{j=1}^{4} \alpha_{ij} \boldsymbol{c}_j|_{world} \tag{4.22}$$

The projection onto the image sensor of the 3D points is given by:

$$\boldsymbol{x}_i = \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij} \boldsymbol{c}_j|_{camera} \tag{4.23}$$

with $\boldsymbol{K}$ expressing the calibration matrix. The same steps as in Eq. (4.19) and Eq. (4.20) can be taken to generate a linear system in the form $\boldsymbol{Ax} = 0$ by exploiting the n 3D world points, where $\boldsymbol{x}$ contains the coordinates of the control points expressed in camera frame. Further details are available in [70]. A nonlinear refinement is also performed to optimize the position of the control points and their corresponding image points. Finally the $\boldsymbol{R}$ and $\boldsymbol{T}$ matrices that minimize the reprojection error of the world reference points and the corresponding image points are computed.

4.8 Bundle Adjustment

Bundle adjustment is an optimization technique generally found at the last steps of a computer vision algorithm. It aims to refine the model of the structure of the scene, 3D points, and the motion of the camera in terms of relative poses. Camera optical parameters can be optimized too, but in this thesis it is assumed that cameras are already known and calibrated.
As detailed in [3] any 3D point is projected onto an image frame by solving the equation $\boldsymbol{x} = \boldsymbol{PX}$, where $\boldsymbol{x}$ is the 2D point, $\boldsymbol{X}$ is the 3D world one and $\boldsymbol{P}$ is the camera matrix. This equation in presence of noise is never solved exactly and estimated quantities needs to be used instead:

$$\hat{\boldsymbol{x}} = \hat{\boldsymbol{P}}\hat{\boldsymbol{X}} \tag{4.24}$$

By assuming the noise to be Gaussian the goal is to estimate the three parameters above such that Eq. (4.24) is solved exactly and the reprojection error between the estimated 2D point and the real 2D point is minimized. This needs to happen for every reconstructed 3D point and for every camera poses that is available. By denoting with the index i the i-th camera view and j the j-th point, Eq. (4.24) becomes:

$$\hat{\boldsymbol{x}}_j^i = \hat{\boldsymbol{P}}^i \hat{\boldsymbol{X}}_j \tag{4.25}$$

while the equation for the reprojection error is written as:

$$\min_{\hat{\boldsymbol{P}}^i, \hat{\boldsymbol{X}}_j} \sum_{i,j} d(\hat{\boldsymbol{P}}^i \hat{\boldsymbol{X}}_j, \boldsymbol{x}_j^i) \tag{4.26}$$

with d denoting the geometric distance between the estimated and true 2D points. The problem is generally solved through Levenberg–Marquardt method thanks to its implementation being fast to execute. However, global bundle adjustment can quickly become a huge minimization problem and is generally not implemented in real time simulations. Indeed, since a full camera matrix requires 11 parameters at minimum and a 3D point requires 3 coordinates, 3j+11i parameters are needed in total. When using Levenberg–Marquardt method (3j+11)×(3j+11) matrices must be factored or inverted. Important processing power should be available on board, which is generally not the case for space applications, so that the problem of bundle adjustment is reduced in dimension. One simple solution consists in applying the method to a moving window between frames or to carefully selected frames in the image sequence, called key-frames. Moreover bundle adjustment method can be selectively used to optimize structure only, the 3d points in the scene, or motion only, the camera parameters, trajectory included, to provide further customization options.

5 Deep Learning

In this chapter theory background regarding Fully Connected Neural Network and Convolutional Neural Network is presented. In particular the backpropagation algorithm is closely discussed.
Deep learning is a sub-field of machine learning that exploits multiple layers neural networks to extract progressively complex and elaborate features along the network from the input. Deep learning and artificial neural networks, ANNs, were at first inspired by the information processing of the brain neural network and its basic elements, neurons.

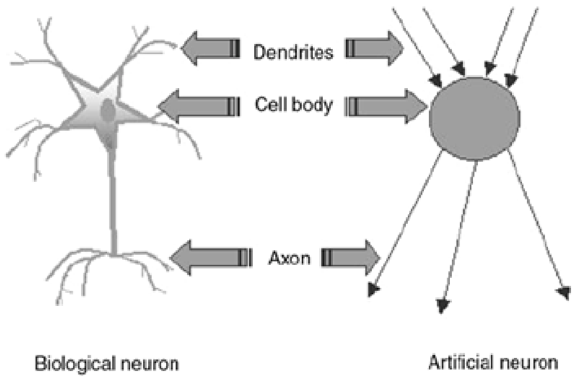


Fig. 5.1: Biological and artificial neuron [6]

The term “deep” is referred to the capability of such networks to extract profound features in the input data when the number of hidden layers increases. For example the detection of such features could begin from identification of lines in an image, then corners, basic geometrical figures up to letters or complex objects.

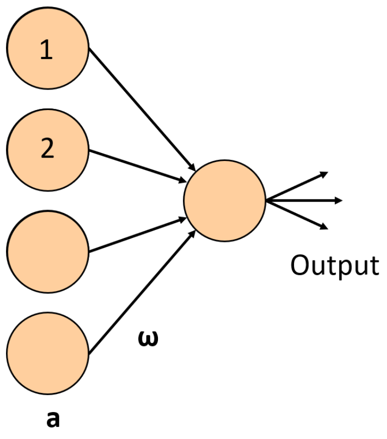


Fig. 5.2: Example of neuron connections

Fig. 5.2 shows the simplest connections that a neuron can have. The input to the neuron is the linear combination of all the j-th connections coming in plus a bias term. Its output can be written as:

$$output = f(\boldsymbol{\omega} \cdot \boldsymbol{a} + b) \tag{5.1}$$

where $\boldsymbol{a}$ is the vector of outputs $[a_1, a_2, ...]^T$ coming from the preceding neurons, $\boldsymbol{\omega}$ is the vector of weights $[\omega_1, \omega_2, ...]^T$ associated to the connections and b is a bias. f is called activation function and determines if and how strongly the neuron fires. Moreover f is necessary in order for the ANN to be able to model nonlinearity in the given problem. The most common activation functions are ReLu, or rectified linear unit, for regression problems and softmax for classification problems with k-th categories, usually in the last layer:

$$f_{ReLu}(x) = max(0, x) \tag{5.2}$$

$$f_{softmax}(x_j) = \frac{e_j^x}{\sum_j e_j^x} \tag{5.3}$$

There are many different types of ANNs, the simplest one being the fully connected one. These generally receive a vector of values as input and output another vector. When the input is composed of images CNNs, or Convolutional Neural Networks, can be used. The hidden layers are composed by a large amount of filters which span the input image at different scales. In the following sections details and algorithms about these two types of networks is discussed.

5.1 Fully Connected Neural Networks

Neurons in fully connected ANNs are organized in layers, where each layer contains a certain number of cells. Every layer is connected with the previous and the subsequent layers as shown in Fig. 5.3. Each neuron of a given layer receives information coming from the previous layer and streams it forward after some operations. The first and last layers have connections on one side only.

In order to understand 'how good' an ANN performs, a loss function is defined: mean squared error, MSE, is adopted to this purpose. When an input $\boldsymbol{x}$ is presented to the network, it travels through the various layers until an output $\hat{\boldsymbol{y}}$ is produced. The loss function with respect to the true target output $\boldsymbol{y}$ is:

$$C(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{b}) = \frac{1}{2n} \sum_{x=1}^{n} ||\boldsymbol{y}(\boldsymbol{x}) - \hat{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{b})||^2 \tag{5.4}$$

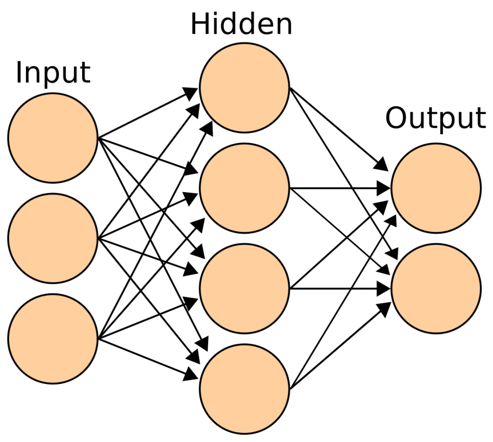


Fig. 5.3: Fully connected ANN example

Where n is the number of training samples, $\boldsymbol{\omega}$ and $\boldsymbol{b}$ are the vectors containing all the weights and biases of the ANN. MSE computes the differences between the target and predicted outputs for every training sample and then takes an average over those inputs.

The objective becomes then the minimization of the error as defined statistically by $C(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{b})$ by acting on weights and biases of the network. This can be achieved by using the methods described in Appendix B and the gradient descent method can be used thanks to its simplicity. Indeed using algorithms which compute the second derivatives can quickly become problematic: for example in an ANN with one million parameters the algorithm would have to compute a matrix with $\frac{1}{2}\text{million}^2$ elements, which can become a short stopper in terms of available computing resources.

Recalling Eq. (B.6) and using the direction of the gradient it is possible to write the updates that must be made on the k-th weight l-th and bias as:

$$\omega_{k+1} = \omega_k - \eta \frac{\partial C}{\partial \omega_k} \tag{5.5}$$

$$b_{l+1} = b_l - \eta \frac{\partial C}{\partial b_l} \tag{5.6}$$

Where η is the learning rate and defines how big of a step in the gradient direction is taken at every update. The main problem with this approach is that the loss function $C(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{b})$ should be evaluated for all the training samples n before making an adjustment to the parameters of the previous equations through its gradient, which must be computed too. Again, generally due to the size of the ANN and its weights and biases in particular, duration of training and hardware requirements can scale up very quickly, leading to failure during training due to lack of memory.

To avoid this problem, SGD or stochastic gradient descent is introduced. A way to estimate the gradient is provided by the following equation:

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{x_j} \tag{5.7}$$

where ∇C_x is the gradient of C computed for one training input only and the sum goes over m randomly selected inputs. The set composed by those random inputs is called a minibatch. This is an approximation of the gradient over some samples but, if the minibatch is statistically coherent with the training set, this method will find a low-cost way to decrease the cost function. Moreover the next update will also start from a vantage point with a lower cost function. Eq. (5.5) and Eq. (5.6) become:

$$\omega_{k+1} = \omega_k - \eta \left(\frac{1}{m}\sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial \omega_k} \right) \tag{5.8}$$

$$b_{l+1} = b_l - \eta \left(\frac{1}{m}\sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial b_l} \right) \tag{5.9}$$

Once a minibatch is used, another one is generated with different samples for the next iteration. When all inputs are used, an "epoch" of training is concluded.

5.2 Backpropagation

In this chapter backpropagation algorithm is discussed. This method allows to compute the gradient of $C(\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{b})$ and is used along with SGD to train a neural network. The description will follow the notation and structure of [71].

First a_j^l is defined as the activation output of j-th neuron in layer l. The weight connecting k-th neuron in layer $l-1$ to j-th neuron in layer l is ω_{jk}^l while the bias of neuron j-th in layer l is b_j^l. As in Eq. (5.1), the expression for a_j^l is:

$$a_j^l = f\left(\sum_k \omega_{jk}^l a_k^{l-1} + b_j^l \right) \tag{5.10}$$

or, in matrix form:

$$\boldsymbol{a}^l = f(\boldsymbol{\Omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l) \tag{5.11}$$

with $\boldsymbol{a}^l$ activation output vector for layer l, $\boldsymbol{b}^l$ bias vector for layer l and $\boldsymbol{\Omega}^l$ matrix of weights with entries ω_{jk}^l. For convenience define also z_j^l the weighted input as:

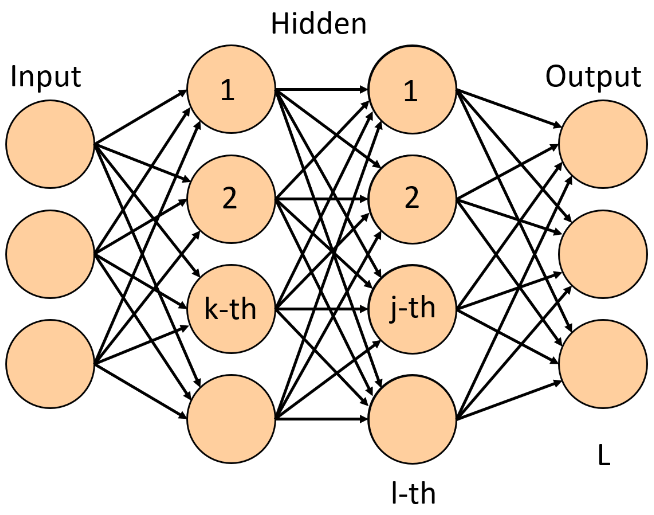


Fig. 5.4: Notation for ANN

$$z_j^l = \sum_k \omega_{jk}^l a_k^{l-1} + b_j^l \tag{5.12}$$

or:

$$\boldsymbol{z}^l = \boldsymbol{\Omega}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \tag{5.13}$$

so that $a_j^l = f(z_j^l)$ or $\boldsymbol{a}^l = f(\boldsymbol{z}^l)$. The main objective of backpropagation is, given the cost function C_x of one sample, to trace back the error through the network and compute $\partial C_x / \partial w_{jk}^l$ and $\partial C_x / \partial b_j^l$ for any weight and bias in the ANN in order to update them with Eq. (5.8) and Eq. (5.9). To this purpose define the error of the j-th neuron as:

$$\delta_j^l = \frac{\partial C_x}{\partial z_j^l} \tag{5.14}$$

The reasoning behind this definition is that when δ_j^l is high in value, it means that the corresponding neuron can still be updated in order to reduce the value of the cost function. If instead δ_j^l is close to zero, the j-th neuron already contributes to lower C_x as much as possible. The objective of the following discussion is to find a way to compute δ_j^L and to go backwards in the network computing every other δ_k^{l-1}. Finally from these information gradients can be extrapolated.

The error in the output layer δ_j^L can be rewritten starting by its definition:

$$\delta_j^L = \frac{\partial C_x}{\partial z_j^L} \tag{5.15}$$

Since $C_x = C_x(a_j^L)$, the chain rule can be applied as:

$$\delta_j^L = \frac{\partial C_x}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C_x}{\partial a_j^L}\frac{\partial f(z_j^L)}{\partial z_j^L} \tag{5.16}$$

and so:

$$\delta_j^L = \frac{\partial C_x}{\partial a_j^L} f'(z_j^L) \tag{5.17}$$

or, in matrix form, denoting with $\odot$ the component-wise product or Hadamard product:

$$\boldsymbol{\delta}^L = \nabla_a C_x \odot f'(\boldsymbol{z}^L) \tag{5.18}$$

The error in any other hidden layer δ_k^{l-1} is, by definition:

$$\delta_k^{l-1} = \frac{\partial C_x}{\partial z_k^{l-1}} \tag{5.19}$$

Since $C_x = C_x(z_1^l(z_k^{l-1}), z_2^l(z_k^{l-1}), ...)$, using the chain rule:

$$\delta_k^{l-1} = \sum_j \frac{\partial C_x}{\partial z_j^l}\frac{\partial z_j^l}{\partial z_k^{l-1}} \tag{5.20}$$

which becomes:

$$\delta_k^{l-1} = \sum_j \delta_j^l \, \omega_{jk}^l \, f'(z_k^{l-1}) \tag{5.21}$$

or:

$$\boldsymbol{\delta}^l = ((\boldsymbol{\Omega}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot f'(\boldsymbol{z}^l) \tag{5.22}$$

The gradient of the cost function with respect to biases, the partial derivative of C_x with respect to b_j^l, is easily obtainable considering that $C_x = C_x(z_j^l(b_j^l))$:

$$\frac{\partial C_x}{\partial b_j^l} = \frac{\partial C_x}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C_x}{\partial z_j^l} 1 \tag{5.23}$$

and so:

$$\frac{\partial C_x}{\partial b_j^l} = \delta_j^l \tag{5.24}$$

Finally, considering that $C_x = C_x(z_j^l(\omega_j^l))$, the gradient of the cost function with respect to weights:

$$\frac{\partial C_x}{\partial \omega_{jk}^l} = \frac{\partial C_x}{\partial z_j^l}\frac{\partial z_j^l}{\partial \omega_{jk}^l} = \frac{\partial C_x}{\partial z_j^l}\, a_k^{l-1} \tag{5.25}$$

so:

$$\frac{\partial C_x}{\partial \omega_{jk}^l} = \delta_j^l\, a_j^{l-1} \tag{5.26}$$

The backpropagation algorithm can now be built from the equations developed above. As seen in previous sections backpropagation allows to compute the cost function gradients from an input sample. This input is propagated forward through the network up to the last layer with Eq. (5.11) and Eq. (5.13). Then the error is computed by going backwards along the network through Eq. (5.18) and Eq. (5.22) in order to compute the corrections with Eq. (5.24) and Eq. (5.26). Then weights and biases values can be updated using SGD procedure similarly to Eq. (5.8) and Eq. (5.9) as:

$$\hat{\omega}_{jk}^l = \omega_{jk}^l - \eta\left(\frac{1}{m}\sum_{x=1}^{m}\frac{\partial C_x}{\partial \omega_{jk}^l}\right) \tag{5.27}$$

$$\hat{b}_j^l = b_j^l - \eta\left(\frac{1}{m}\sum_{x=1}^{m}\frac{\partial C_x}{\partial b_j^l}\right) \tag{5.28}$$

With $\hat{\omega}$ and $\hat{b}$ denoting updated values. Pseudocode of this method is shown below.

Algorithm 1: SGD with Backprop pseudocode

```
for epochs = 1:$N_E$ do
    generate minibatches;
    for minibatches = 1:$N_m$ do
        for one input sample $x$ in minibatch do
            supply input values to Input Layer;
            for layer = 2:$L$ do
                % propagate forwards;
                compute $\boldsymbol{z}^l$ (5.13);
                compute $\boldsymbol{a}^l$ (5.11);
            end
            compute output error $\delta^L$ (5.18);
            for layer = $L$-1:2 do
                % propagate backwards;
                compute $\boldsymbol{\delta}^l$ (5.22);
            end
            compute $\partial C_x/\partial \omega^l_{jk}$ (5.26) $\forall \omega^l_{jk}$;
            compute $\partial C_x/\partial b^l_j$ (5.24) $\forall b^l_j$;
        end
        update all $\omega^l_{jk}$ (5.27);
        update all $b^l_j$ (5.28);
    end
end
```

5.3 Convolutional Neural Networks

Convolutional Neural Networks are specialized kind of networks that perform the convolution operation inside their layers instead of simple matrix multiplication, which is usually performed only in their last layers [8].

The history of CNN is a very recent one, having its fundations in the Neocognitron, introduced by Kunihiko Fukushima in 1980 [72]. Its contribution to shift invariance of the NN to shapes in the input layer is the precursor of convolutional neural layers. In 1998 one of the most iconic CNN was built, called LeNet-5 [7]. It was a CNN able to recognize digits from 32x32 images and, while its architecture was not very complex with respect to modern standards, it obtained excellent performances, with a 1% error rate.

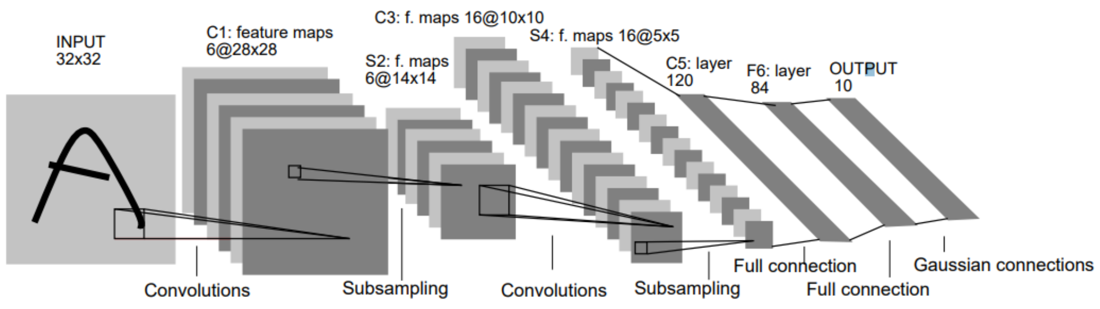


Fig. 5.5: LeNet-5 architecture [7]

In early 2010s GPUs started being used as a tool to train neural networks, which resulted in a great boost in the training time duration. GPUs indeed have thousands of computing cores while a CPU might have only few of those, as they are built with different purposes in mind. This enabled CNNs to achieve state of the art performances on image recognition challenges and in 2012 AlexNet was published [73]. It contained a total of eight layers, the first five being convolutional ones and the rest fully connected ones for a total of 61 million parameters. It achieved high performances on the ImageNet dataset and was overcomed only after three years by a 100 layers deep CNN by Microsoft. Currently neural networks can be built using hundreds of layers and tens of millions of parameters and with complex architectures. Another kind of layer, called the recurrent layer, can be exploited to provide a feedback effect inside the nested layers and has proven to be very useful for state of the art neural networks [74]. Even though CNNs are a very recent tool and still an open research field, they have already contributed to technology advancements in the space sector. In Earth imaging data analysis most applications, from automated ice-charting [40] to ship detection [41], are built with CNN based architectures. MISR, multi image super resolution, has been implemented in space context too [45] to enhance image resolution, while object pose estimation from a single image is another strong research field applied successfully to a known satellite [46].

From a mathematical point of view, the 1D convolution operation is defined as:

$$y(t) = \int x(a)w(t-a)da \tag{5.29}$$

and in discrete domain, the one in which CNNs operate, it becomes:

$$y(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{5.30}$$

1D convolution is especially useful with LTI systems. Any signal can be decomposed into a weighted sum of shifted delta functions, exactly as it can be decomposed as an infinite sum of sines and cosines for a Fourier decomposition. The output of any LTI system then can be defined as a weighted sum of shifted impulse responses. In CNNs however the input are images, so 2D convolution needs to be defined:

$$Y(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{5.31}$$

where I is the image matrix and K is the kernel, or filter of the operation. Since convolution is commutative, a simpler implementation that is kernel focused is given by:

$$Y(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{5.32}$$

However the kernel or the image in this operation needs to be flipped [75] and this adds unnecessary complexity to machine learning codes. Convolution is redefined as:

$$Y(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{5.33}$$

which is actually the cross-correlation operation. This contrast of notation is not important for CNN implementation as, if the kernel is flipped, then the CNN will learn flipped weights as well. The use of “convolution” terminology has been interpreted as a way to bond CNNs to traditional signal processing and image filtering techniques.

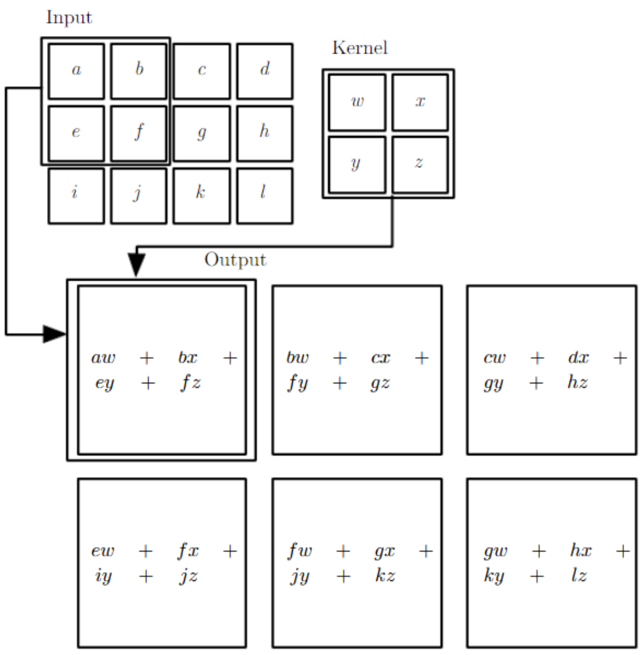


Fig. 5.6: Example of convolution without kernel flipping [8]

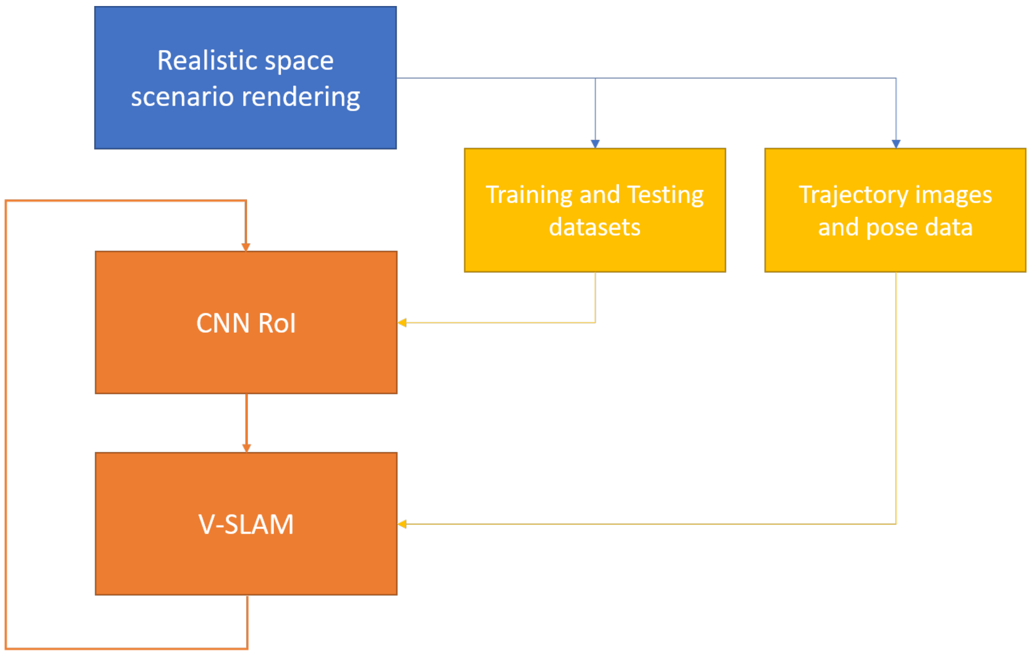
Various parameters of a kernel are tunable in CNNs, such as their size in width and height and the stride, which is correlated to how many times the filter will act on a single image. The kernel entries are instead learned while training.

A generally unwanted effect that happens when doing a convolution operation on images is that the output size will be different with respect to the input size and the reduction of dimension depends on the filter size. To compensate for this effect zero-padding can be used, where the input image is padded with zeros until the output size matches the input one. In between CNNs layers an operation called Pooling can be performed, which reduces the image size by a predefined amount. It is important that a CNN is scale invariant, in the sense that features at low scale are detected as well as the same features at larger scales. For this reason, pooling layers are usually exploited, which resemble the pyramidal formulation of the KLT algorithm. Another advantage of pooling in between layers is that images become sequentially smaller, reducing the computational requirements by reducing the feature map sizes. Other tools can be used in between network layers, which include batch normalization and dropout layers, in order to stabilize the learning process and allow faster training.

Once a CNN layer has been activated, a trainable bias can be added to the numerical output and an activation function can be used to add nonlinearity capabilities to the network as done previously for the Fully Connected Neural Networks. The resulting feature maps, the output of the convolutional layer, is passed forward to the next layers until the output layer is reached.

6 Architecture for relative spacecraft navigation

In the following chapter the framework for relative spacecraft visual navigation proposed in this thesis is provided. In Section 6.1 the V-SLAM problem, the problem of the relative spacecraft trajectory and 3D structure of a scene computation, is briefly discussed and used to asses relative navigation capabilities. A CNN is also deployed, as discussed in Section 6.2, to restrict the feature search region only to the bounding box of the target satellite to remove as many false features as possible. In order to generate images and pose data for both the V-SLAM algorithm and the neural network training, validation and testing datasets, Blender, a rendering software, has been used. The details of its implementation are presented in Section 6.3. The proposed architecture is shown in Fig. 6.1, with the navigation blocks denoted in orange.



Realistic space scenario rendering
Training and Testing datasets
Trajectory images and pose data
CNN RoI
V-SLAM

Fig. 6.1: Relative spacecraft navigation architecture

The choice of this architecture is mainly based on a tradeoff between complexity/cost and accuracy. Thanks to continuous development in rendering software the ability to model real scenarios is constantly increasing. Software cannot completely replace a dedicated GNC laboratory, however it is still a valid solution and its cost is definitely irrelevant with respect to the laboratory one. It is an even more compelling solution when considering that it allows for fast prototyping, with tests that can be carried quickly even if they use completely different target and background models. Thousands of tests can be performed in a relatively small amount of time and any kind of trajectory can be defined. Numerous quantities of interest can be extrapolated, which would otherwise need dedicated sensors in a laboratory environment. With respect to the navigation algorithm, V-SLAM represents the state of the art for relative navigation. Filtering techniques have not been used, even if they are very efficient and computationally inexpensive, as they do not provide the same level of accuracy overall with respect to optimization ones, as discussed in Section 1.2. CNNs represent the state of the art as well for image processing and machine learning, a field of research still in its initial stages. False feature detection is important to ensure a proper behaviour of the V-SLAM algorithm and few methods to face this problem have been developed, as discussed in Section 6.2. Each one has its strengths and drawbacks, but a trained CNN would offer the best performance with insignificant computational time.

6.1 V-SLAM for Visual Navigation

The problem of computing the trajectory and 3D structure of a scene is generally referred to as SLAM. In order to settle in the most inclusive situation, the one of an inspector spacecraft visiting an unknown and uncooperative object, no hardware, such as telecommunication antennas or markers, is considered available on the target. A LiDAR is excluded due to its high mass and power requirements and a single mono camera is chosen over a stereo one thanks to its simplicity and better applicability on cubesats scenarios.

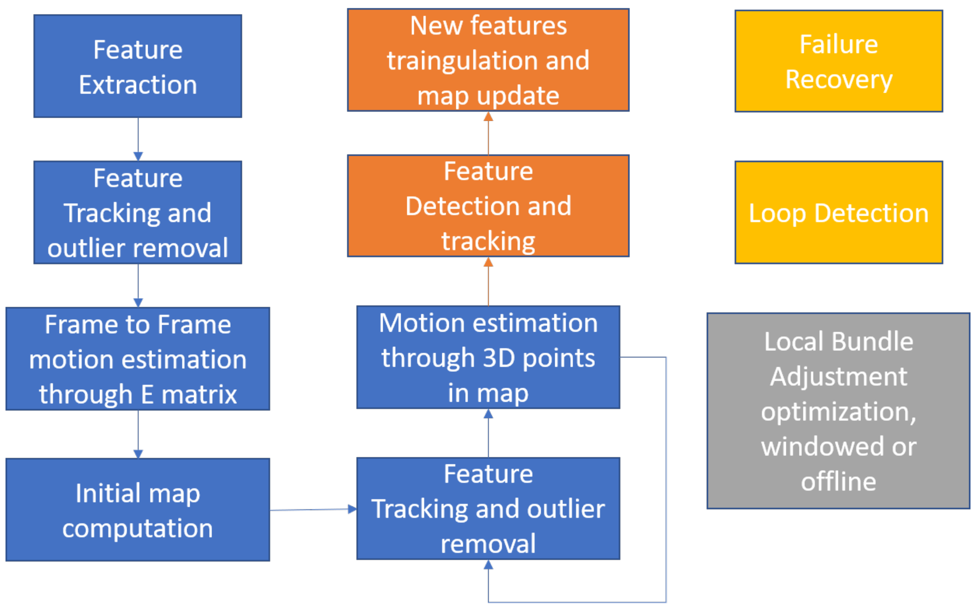

Feature Extraction
Feature Tracking and outlier removal
Frame to Frame motion estimation through E matrix
Initial map computation
New features traingulation and map update
Feature Detection and tracking
Motion estimation through 3D points in map
Feature Tracking and outlier removal
Failure Recovery
Loop Detection
Local Bundle Adjustment optimization, windowed or offline

Fig. 6.2: Typical V-SLAM pipeline

As such visual SLAM is considered and in Fig. 6.2 its typical feature based implementation is shown. It is composed of an initialization step, in the left column, in which features are extracted and tracked between two frames. If any, outliers can be removed through RANSAC/MSAC and the essential matrix estimation can take place. From it, roto-translation properties of the camera motion are retrieved and an initial map is computed via triangulation. The map can then be used to navigate in the scene by continuously solving a PnP problem with features that are tracked between frames. When some predefined conditions are met, new feature points can be extracted in a given frame, 3D points are computed and added to the map. Often V-SLAM methods have also a loop detection process that improves map consistency over time and reduces the uncertainty in the pose estimation, which otherwise may grow indefinitely. In image-to-image loop closing popular approaches require the use of a visual dictionary, or bag of words, in order to compare images in an efficient way, which represents a non trivial challenge for space applications. Local optimization is also performed to improve the map and trajectory consistency through bundle adjustment. Global bundle adjustment is too expensive to run in real time, so other solutions should be used such as a windowed version or a key-frame based one. In Section 7 two versions of a V-SLAM navigation algorithm are designed by increasing complexity. Bundle adjustment is discussed in the last test scenario.

6.2 CNN for ROI Estimation

V-SLAM algorithms have to deal with challenges typical of the space environment, such as feature absence due to unavoidable lack of texture. The presence in camera frame of unwanted objects like the Earth or natural celestial bodies can also become problematic for the navigation algorithm. In Fig. 6.3 and Fig. 6.4 false point detection is shown, where points not belonging to the target are incorrectly triangulated and propagated in successive frames. This occurrence can be very dangerous for navigation as it can jeopardize the trajectory estimation. Some approaches that can mitigate this effect are the use of RANSAC/MSAC to remove outliers, but care must be taken not to exclude real target points, which is not a trivial task [76]. Point life algorithms can also be used, where a point is triangulated and added to the map only if it appears in a minimum number of frames. This approach however prevents instant use of feature information and does not guarantee that all false features are eliminated. In this thesis another approach is presented, consisting in the estimation of the region of interest through CNNs. In this way a bounding box region is generated around the target satellite for which the network has been trained. Dataset generation, CNN training and architecture are discussed in Section 8 along with CNN testing in different off-nominal conditions.

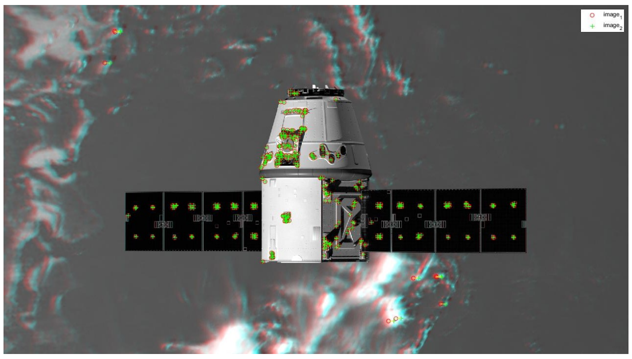
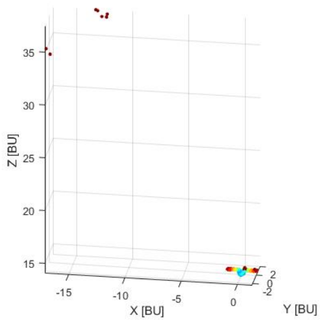
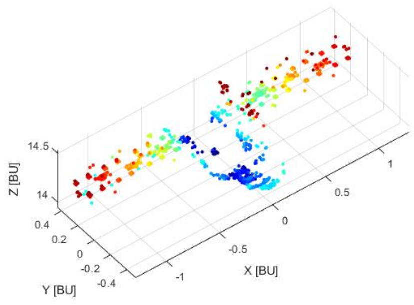


Fig. 6.3: Example of false positive tracks



(a) 3D map with outliers



(b) 3D map with CNN ROI

Fig. 6.4: Examples of 3D map generation with and without CNN ROI

6.3 Modeling of realistic spacecraft scenarios

In order to obtain realistic spacecraft images along with relative or absolute pose data of the target and inspector spacecraft, Blender, a free and open source 3D creation suite, has been used, denoted by the blue block in Fig. 6.1. Synthetic images of two orbiting spacecraft and the relative pose data for V-SLAM testing, used in Section 7, have been simulated entirely in Blender. Moreover images for the deep learning training, validation and testing datasets have been generated as well with this tool, along with pose data for bounding box generation, as detailed in Section 8. Further description of the algorithm for dataset generation is provided in the next section. Some of the used 3D models have been sourced online and are available under non commercial use. Links to original authors are provided in [77] [78] [79] [80] [81].

The main Blender graphic engine is called Cycles and is a path tracer engine. Path tracers are an evolution of ray tracers and are built with photorealism as their goal. The main idea behind ray tracing algorithms is the rendering of an image by following light paths casted from the sensor as rays until they hit a light source. Bounces and reflections off an object can split a single ray into multiple ones which need to be followed too. In complex scenes this becomes a short stopper, as the number of rays can become so huge that memory and computational power can be insufficient. For this reason a Monte Carlo method is adopted, in which multiple rays are casted from the same image sensor pixel into the scene and are followed without splitting them. When a ray should be splitted, for example due to a diffuse reflection, one random path is instead chosen and followed. This technique is the path tracing one and allows to perform a trade off between cost and quality of the final result. Quality is penalized when a low number of Monte Carlo iterations is adopted, as noise will be present, while cost in terms of computational resources and time is lowered.

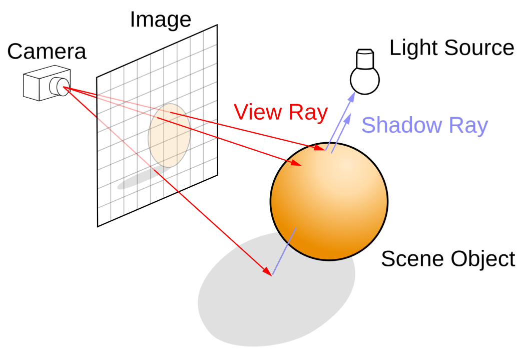


Fig. 6.5: Ray tracing basic principle [9]

6.3.1 Image generation algorithm

The algorithm that has been written for space images generation was initially conceived as a general purpose one, so that its scope of interest could be narrowed later on during this thesis. From it, two categories of scripts have been written. The first one, for image generation when a spacecraft trajectory is imposed around an object, is used in Section 7 in order to provide image data and ground truth data to the V-SLAM algorithm. Thanks to the in-built animation capabilities, the trajectory definition is imposed in Blender and the script stores the pose data in an appropriate database. The second type of scripts are more involved and have been written for the deep learning dataset generation used in Section 8. The pseudo-algorithm that has been written for deep learning image generation is briefly described below.

Algorithm 2: Random image generator

```
for $image$ = $1{:}N_I$ do
    generate random $\boldsymbol{x}_d|_I$, $\boldsymbol{\rho}|_I$;
    compute $\boldsymbol{x}_c|_I = \boldsymbol{x}_d|_I - \boldsymbol{\rho}|_I$;
    generate random $\boldsymbol{q}_d$;
    impose track constraint of chief versus deputy;
    impose small random perturbation on $\boldsymbol{q}_c$;
    generate random Sun position not in camera view;
    render image;
    write on csv/json format: image name, $\boldsymbol{r}_c|_I$, $\boldsymbol{r}_d|_I$, $\boldsymbol{\rho}|_I$, $\boldsymbol{q}_c$, $\boldsymbol{q}_d$;
end
```

At first the deputy spacecraft is generated in a random position. The relative distance is set, constrained between two appropriately chosen boundaries, and the chief spacecraft position can be computed. Attitude of the deputy is randomized, while the attitude of the chief is imposed in order to point to the other spacecraft. A random perturbation is added to the chief attitude too. The Sun position is randomized but constrained not to appear in camera frame. Finally the image can be rendered and parameters are saved in a csv and json files for redundancy.

With respect to the scenarios developed for the two categories, in both of them the inspector satellite is equipped with a full frame, 1152 by 658 [px] and 150 [mm] focal length camera as discussed in Section 3.2. The distance between the target and inspector spacecraft is set, for the V-SLAM scripts category, in order to have most of the target spacecraft in view with no pointing errors. As such their distance in the beginning of the trajectories is roughly 80 [m], which varies during the motion. For the CNN scripts category instead the orbit around the Earth is in LEO at 1500 [km]. The excursion of the distance from the target and the inspector spacecraft is randomly chosen between 100 [m] up to 600 [m]. These values have been chosen to have most of the target satellite in view or to have a recognizable shape at high distances when the Dragon spacecraft model is used. During CNN testing other models are employed and the exact distance values vary, but the reasoning for the threshold selection is the same.

Unfortunately, due to the characteristic lengths of the different objects present in the scene, a compromise in terms of scale was needed in order to fit everything together. Indeed the radius of the Earth is of about 6.370 km while a typical spacecraft can be few meters tall. That is a difference in characteristic lengths of about

1.000.000:1 which makes small meshes unstable and will yield distorted renders. For what concerns absolute distances scaling was introduced by a simple proportion:

$$x_{real} = \frac{r\text{♁}_{real}}{r\text{♁}_{synth}} x_{synth} \tag{6.1}$$

Where ♁ is the symbol for the Earth, x is a generic distance measured in the real world and in Blender world. With respect to relative distances and proportions between the deputy and chief spacecrafts another scaling equation has been used. The proportion is similar to the previous one:

$$\boldsymbol{\rho}_{real} = \frac{l_{real}}{l_{synth}} \boldsymbol{\rho}_{synth} \tag{6.2}$$

where l denotes the characteristic length of the spacecraft, for example its height, in real and synthetic worlds. A comparison between results obtained in Blender and Matlab softwares is reported in Fig. 6.6 for generic deputy/chief poses set. The difference in the convention used for the camera reference frame is visible, with Blender pointing the Z axis away from the target.

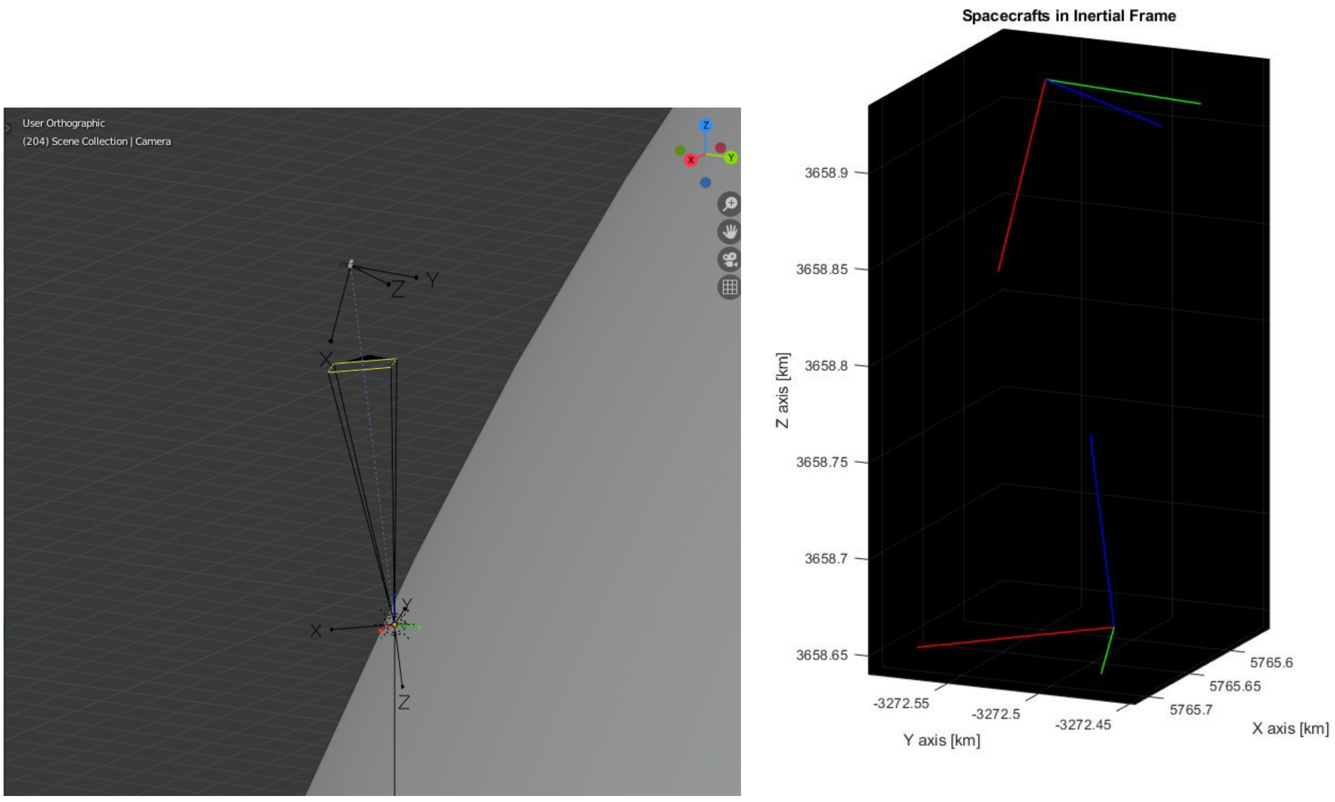


Fig. 6.6: Blender and Matlab outputs example

6.3.2 Render examples

Some examples of renders are presented below at 1152 by 648 pixels resolution. One of the major problems that has been encountered during the render development was the computational time required. Initially renders could take several tens of minutes to compute, and this would have definitely become a short stopper whenever a big dataset of images was needed. However, thanks to the recent improvements to ray tracing technologies, NVIDIA [82] has released in previous years new GPUs optimized for ray tracing and machine learning operations. Moreover, very recently a new implementation of a ray tracing engine on RT cores present in RTX graphic cards, called OptiX, has been introduced into Blender. This new tools have been able to reduce the computational time from tens to few minutes. Following renders were obtained with an RTX 2070 Super coupled with a Ryzen R7 3700X processor.

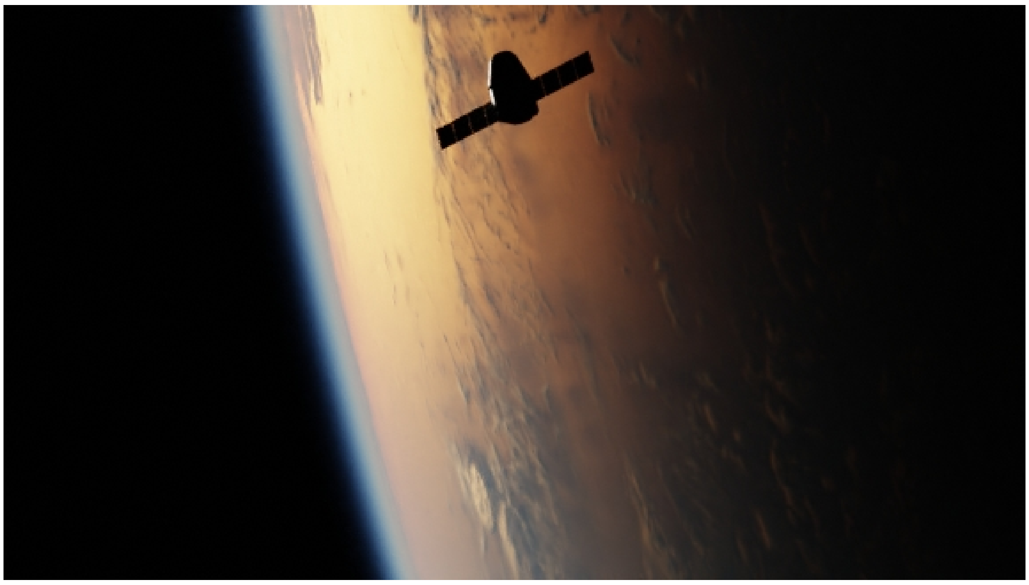


Fig. 6.7: Render example of Dragon spacecraft with Earth in background

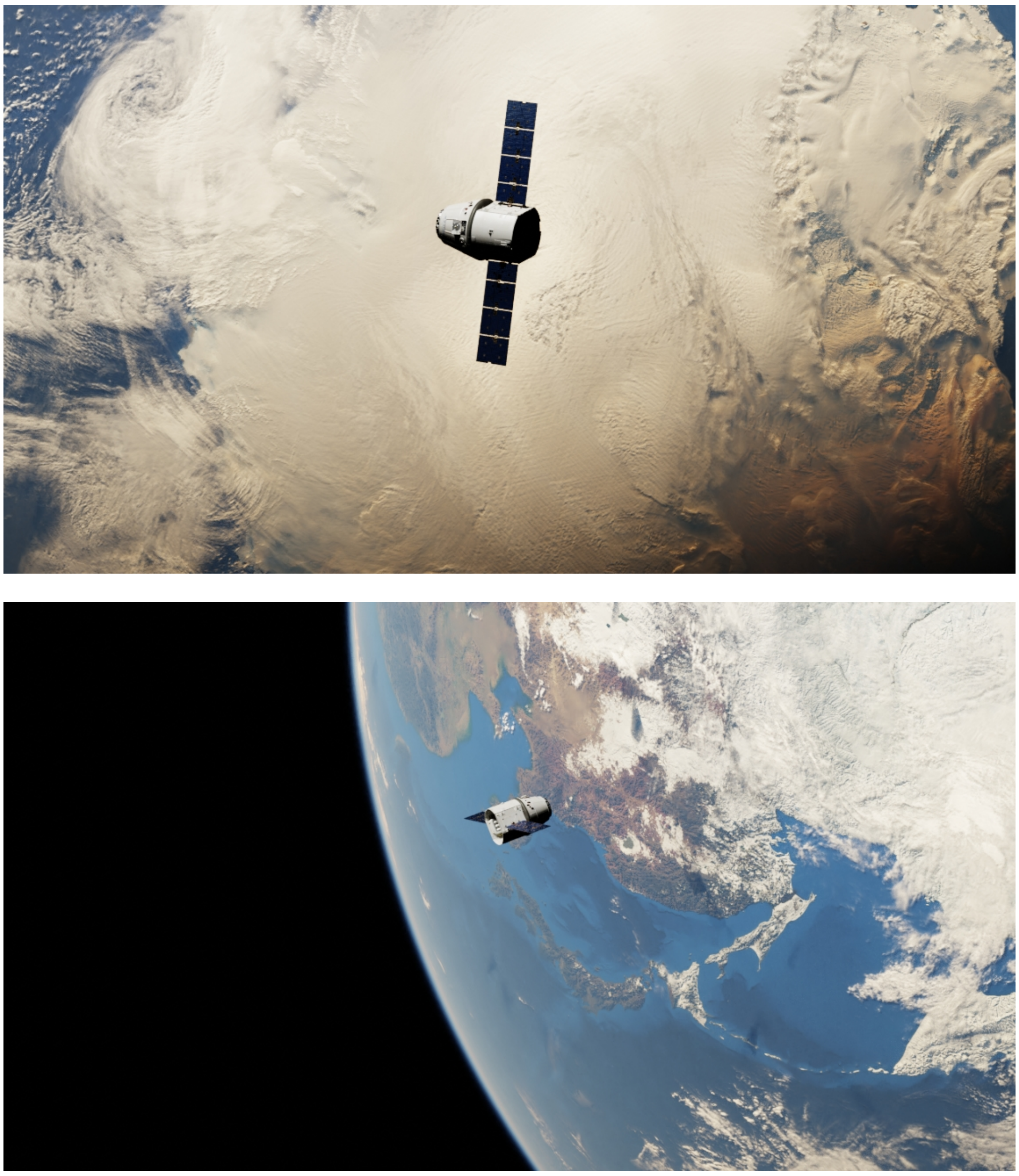


Fig. 6.8: Render examples of Dragon spacecraft with Earth in background

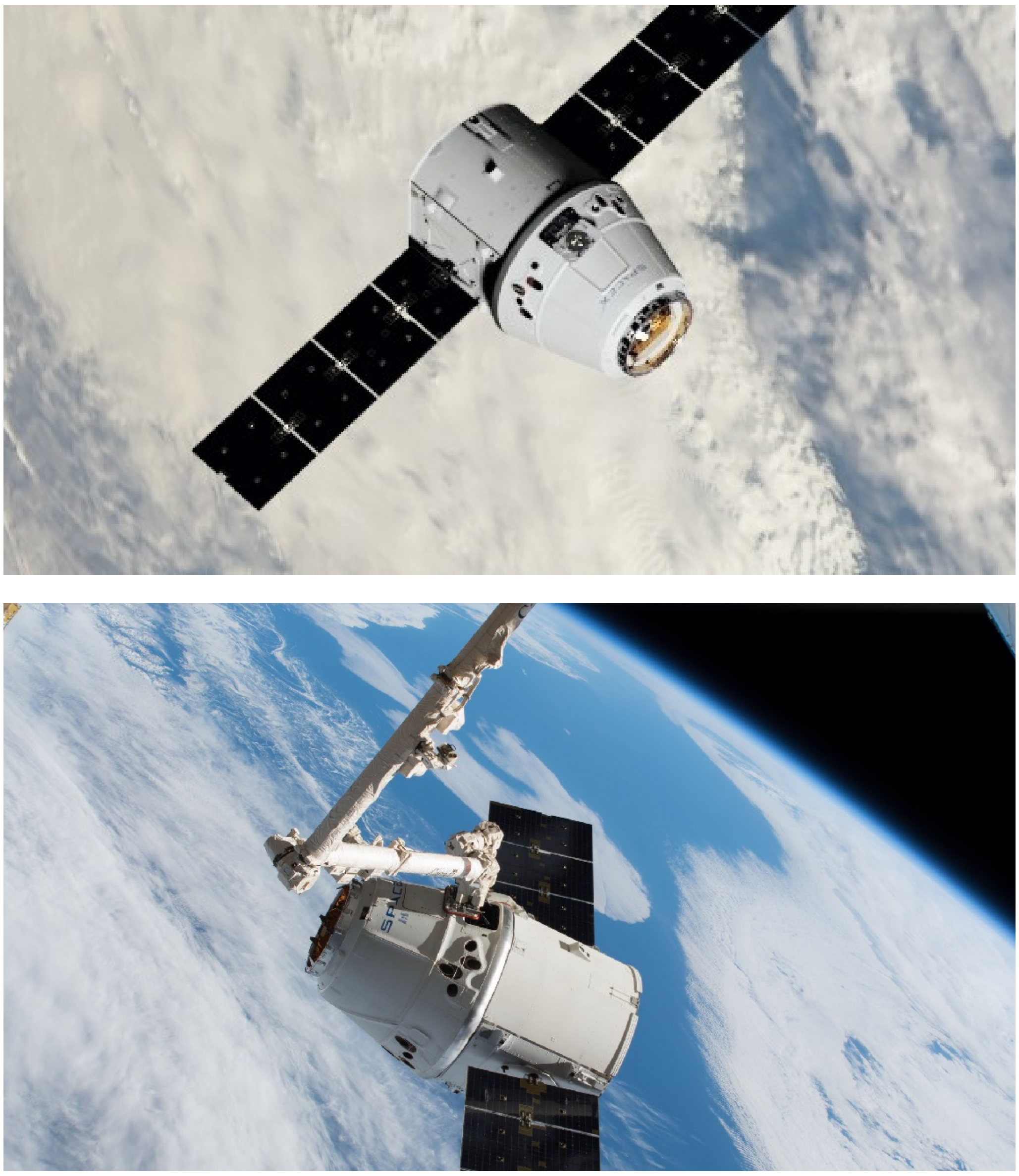


Fig. 6.9: Comparison between rendered image on top and real spacecraft image [10]

7 V-SLAM Implementation and results

In this chapter a V-SLAM like navigation algorithm is presented and analyzed. The algorithm is based on concepts shown in previous chapters and has been developed at first with no synthetic images in the loop. This implementation is discussed in Section 7.1 and is used to asses the model capability to recover the relative pose estimation in one of its simplest form. Then, in Section 7.2, the algorithm using images rendered in Blender is shown.

7.1 Preliminary simulation implementation

In this preliminary simulation a validation study was performed to better understand the method behaviour without introducing image complexity and to a ensure solid foundations for the algorithm discussed later in Section 7.2. In Fig. 7.1 the reference trajectory is shown along with the attitude of the chief spacecraft. The camera that is used is assumed to have a focal length of 150 [mm] and the resolution which is set at 1152x648 pixels is now assumed infinite. This parameters allow to have the target spacecraft occupy most of the image sensor at a distance of 75 [m]; normalized units are used, with each BU, Blender Unit, corresponding to 3 [m].

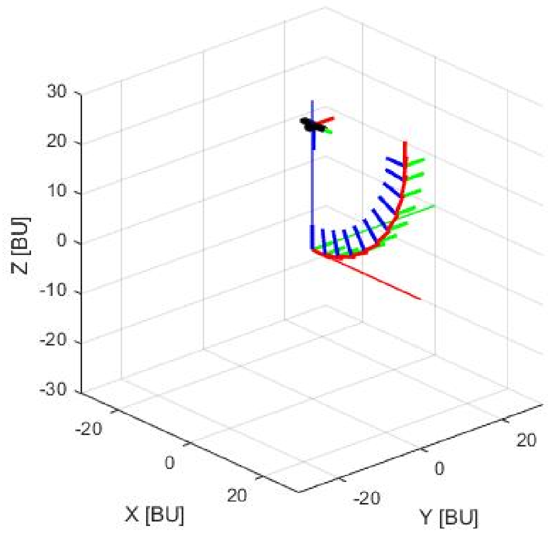
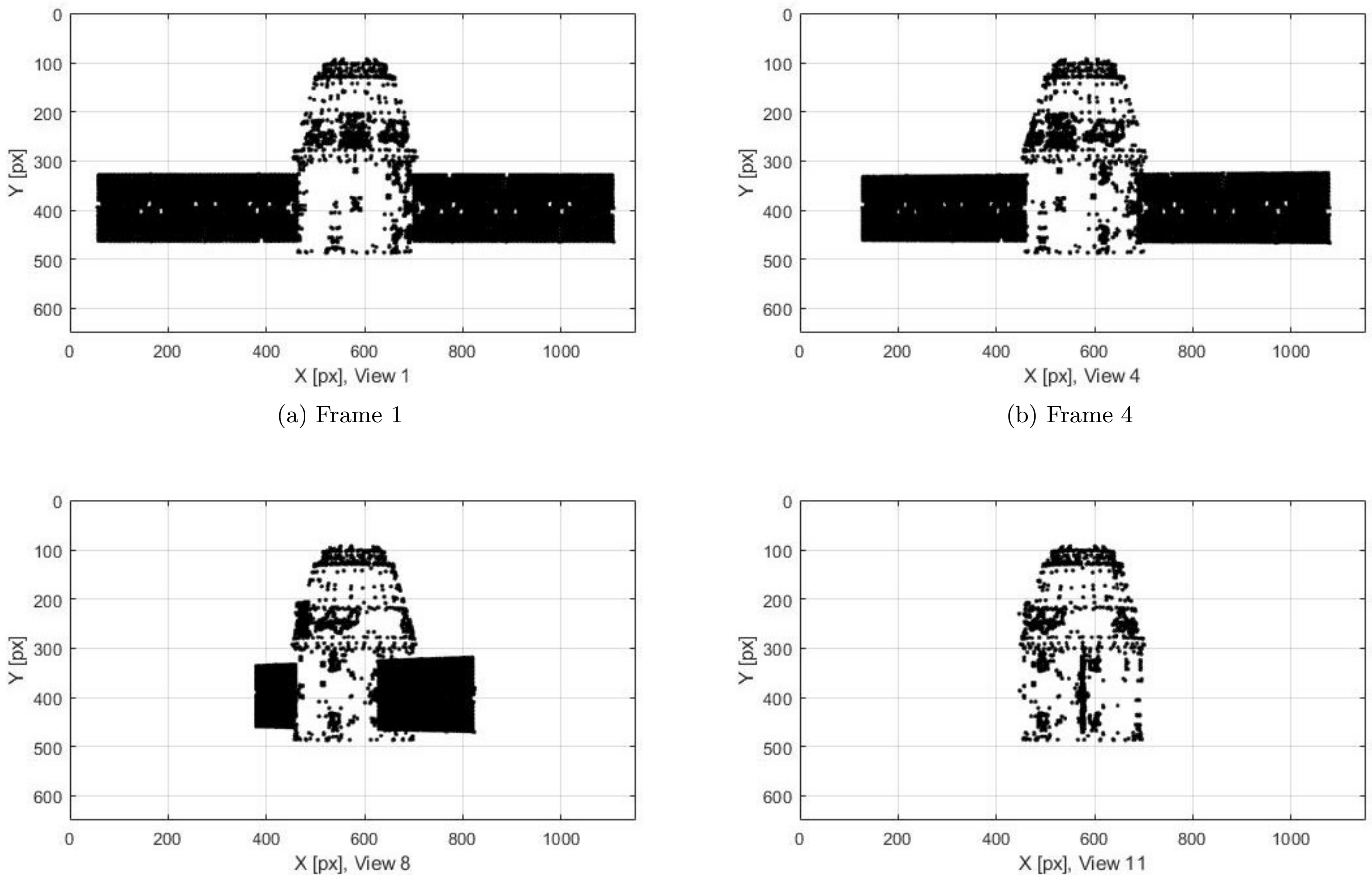


Fig. 7.1: Reference trajectory plot

In Fig. 7.2 mesh points are plotted. Since it is assumed that the sensor has infinite resolution, each feature is retrieved exactly. Mesh points in sensor frame are computed using a rasterization-like method: 3D mesh points are casted onto the sensors as rays and are kept only if they do not intersect any other mesh triangles. The intersection method [83], which is shown previously in Fig. 3.12, can quickly become CPU intensive when a fine mesh is used and parallelization over multiple cores has been introduced to reduce the computation time.



(a) Frame 1

(b) Frame 4

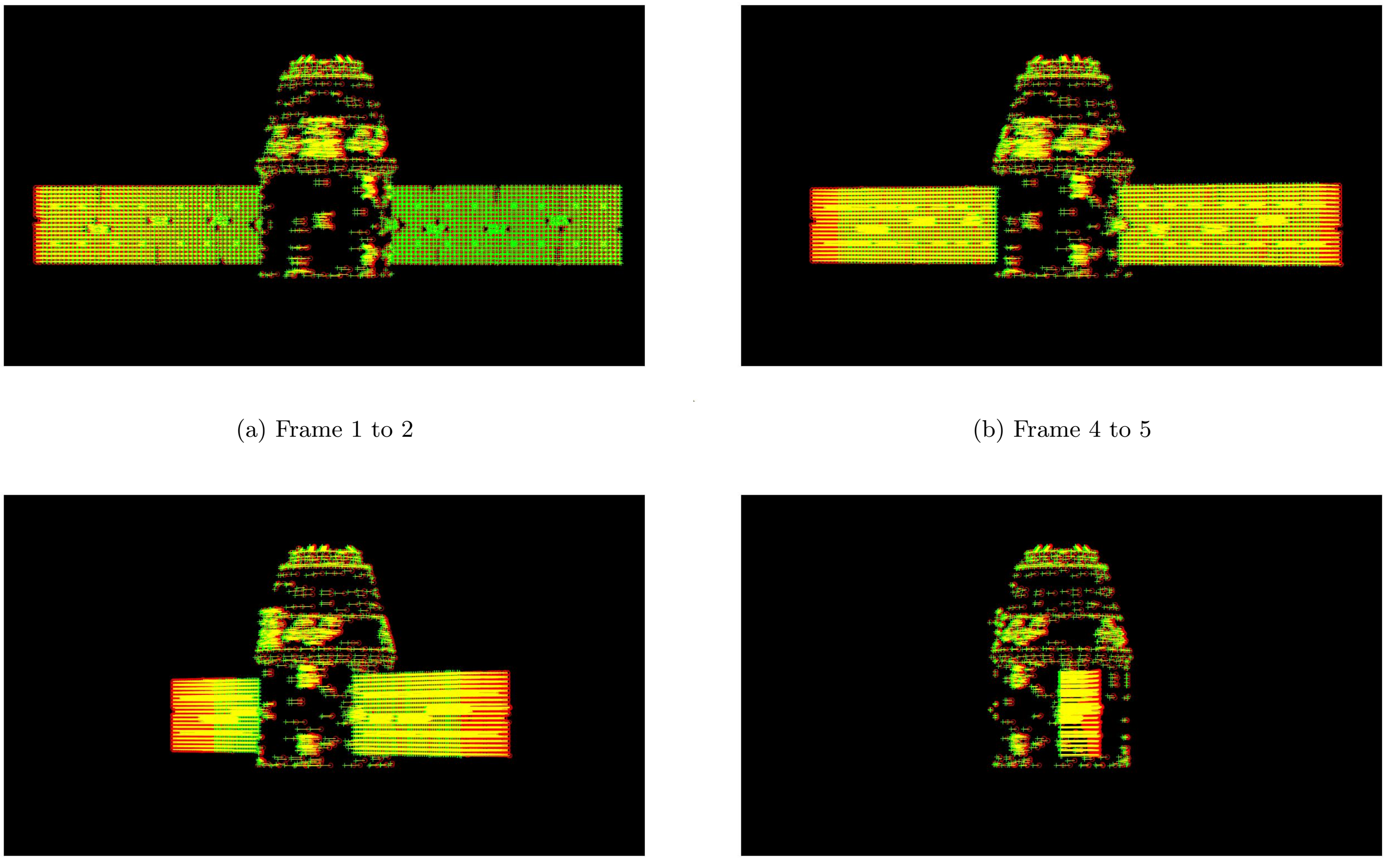
(c) Frame 8

(d) Frame 11

Fig. 7.2: Feature detection examples

Once the feature points are available, they can be tracked between frames as in Fig. 7.3. Since this is done analytically, the tracks are computed exactly. Following the sequence of operations detailed before, the algorithm is divided into an initialization part and a main body one. In the initialization part the first two frames are used and tracks are obtained. Essential matrix can be estimated and the extrinsic parameters relative to the camera motion are obtained. Then 3D point triangulation can take place and an initial map of the environment, the target satellite, is built. Only some tracks are converted into 3D points and are selected to reduce track management computational load. After the initialization is performed, the PnP problem is continuously solved to provide information on subsequent relative poses.

A monocular solution however is not able to retrieve the scale of the scene, which analytically can be explained by the fact that homogeneous vectors are used and the essential matrix is defined up to scale. To compare the estimated results with the ground truth ones scale needs to be reconstructed. This is done by using a ray-casting like method: 2D image points in the initialization step are casted back into the scene where the target spacecraft mesh is present. In this way the actual position of the triangulated 3D points is recovered. The barycenter of the estimated and actual triangulated 3D points is reconstructed through a RANSAC-like method and the relative scale is finally obtained.


(a) Frame 1 to 2

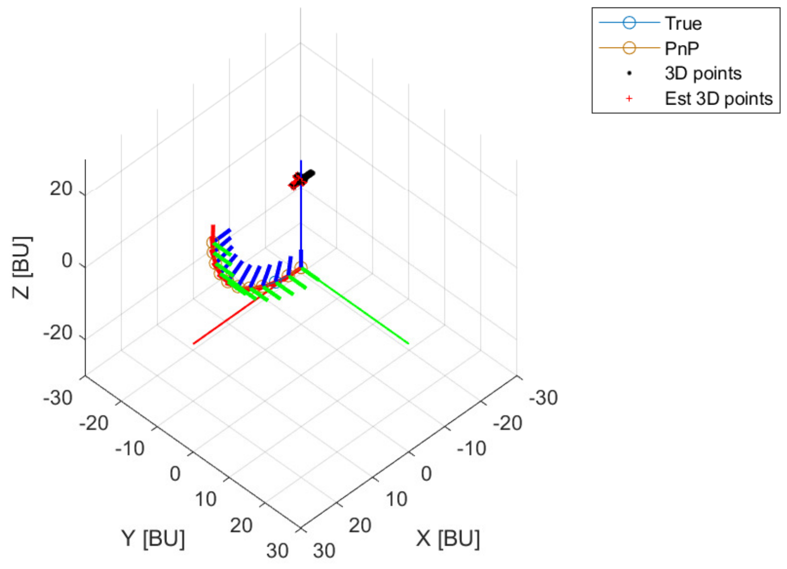
(b) Frame 4 to 5

(c) Frame 7 to 8

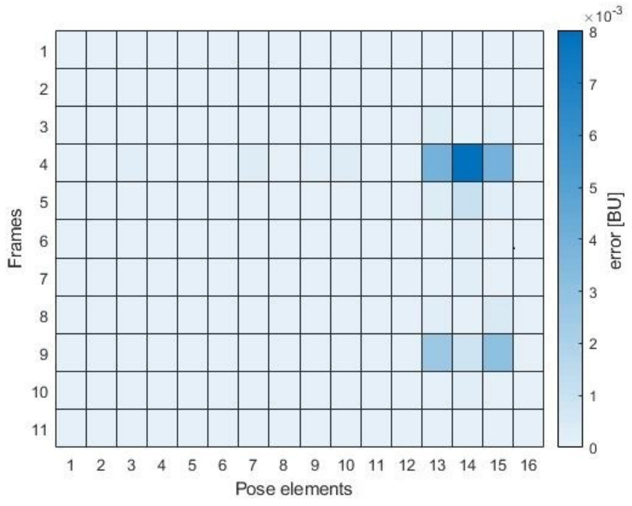
(d) Frame 10 to 11

Fig. 7.3: Feature tracking examples

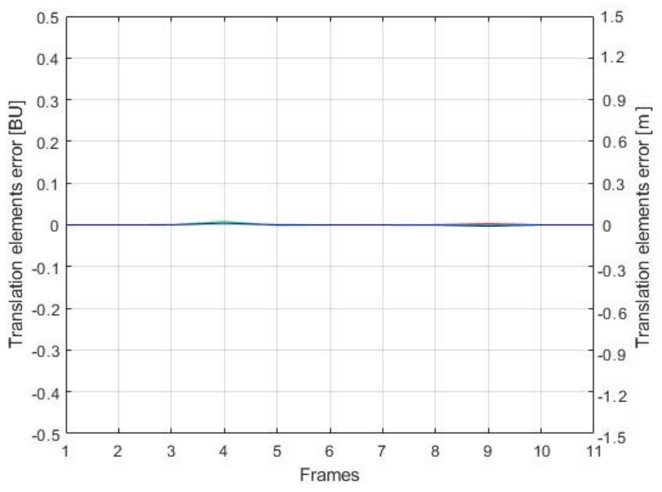
In Fig. 7.4 results for this simulation are shown. As expected, since features are detected and tracked perfectly as the sensor is assumed to have infinite resolution, the error in pose determination, shown in Fig. 7.4b, is very low. The 4x4 pose matrix error is shown in the heat plot: it is composed by the 3x3 rotation matrix in the upper left part and 3x1 translation vector in the upper right part. Elements from 1 to 11 except 4 and 8 are the rotation matrix components while elements from 13 to 15 are the translation components. Translation error only is reported in Fig. 7.4c.



(a) Estimated and actual trajectory with 3D points



(b) Pose matrix elements and error



(c) Translation error elements

Fig. 7.4: Results for preliminary simulation

7.2 Simulation implementation

In this section the algorithm in Section 7.1 is further developed to accept images in the loop. Feature detection and tracking must take place, as well as a proper track management method. Since feature tracking is not an ideal process anymore, error will be introduced into the estimation procedure. Another source of error is the resolution of the image sensor: by discretizing the image into pixels information is inevitably lost.
It is assumed that the Sun is never present in the camera frame. This requirement is necessary to preserve the functionality of the relative estimation algorithm, however it also becomes a constraint on other subsystems, Mission Analysis in particular. Non-nominal cases in which the requirement is violated must however be taken into account too and a failure recovery routine should still be implemented.
In Section 7.2.1 the initialization step is further studied to understand how to properly generate the map during the initialization step. Then from Section 7.3.1 tests of the V-SLAM algorithm are reported.

7.2.1 Map initialization test

V-SLAM algorithm behaviour depends heavily on the initialization step, the process in which the map is generated for the first time. Since that map is used for navigation in the subsequent steps, it is important to obtain an accurate estimation of its 3D points. For this reason a study on the accuracy reconstruction on 3D points has been performed by using as a variable the separation angle between the views. In Fig. 7.5 the effect of the difference in viewing angle is shown: very low difference in the separation angle will result in failure of the method as the uncertainty for the 3D points location is too high. Moreover docking or landing on an asteroid in a straight trajectory with respect to the target does not allow any perspective view on the target object, so that the map will not be correctly generated. Other approaches must be used, such as a prior investigation of the object with consequent map initialization.
However high difference in viewing angle is also problematic in terms of feature tracking. It is generally assumed for feature trackers that features do not change in appearance by large amounts, as small perturbation hypothesis is introduced. Features tracked between largely different frames will inevitably be small in number and might lead to poor accuracy in the pose estimation. For this reason, a trade off must be made to understand how the two mentioned effects combine together.

An algorithm that computes the error in 3D points reconstruction has been developed. The first frame used in the process is always the same while the other one is chosen so that the angle between the views increases. The image sequence used

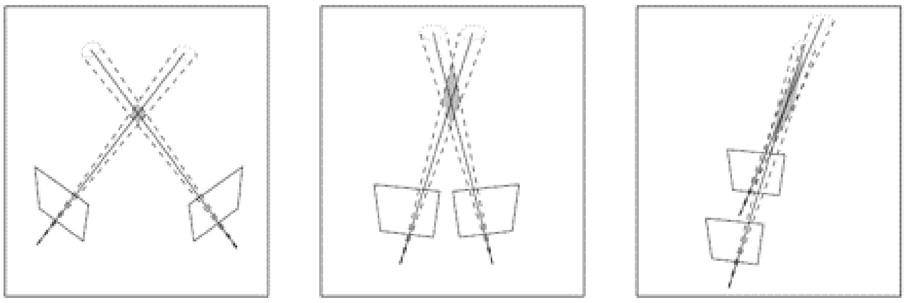
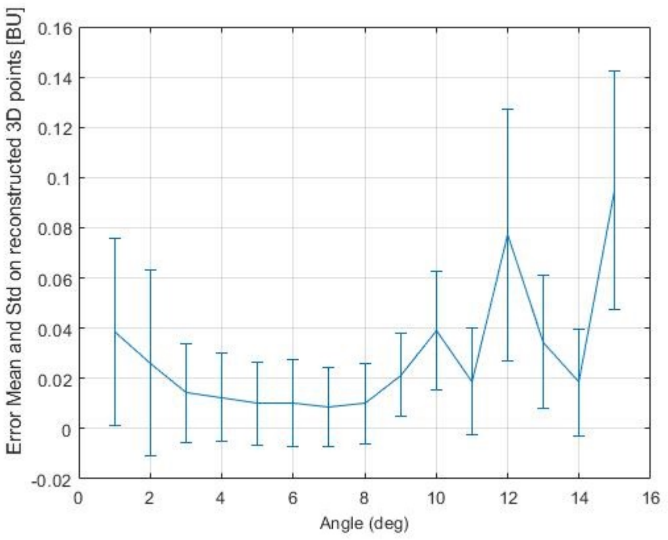


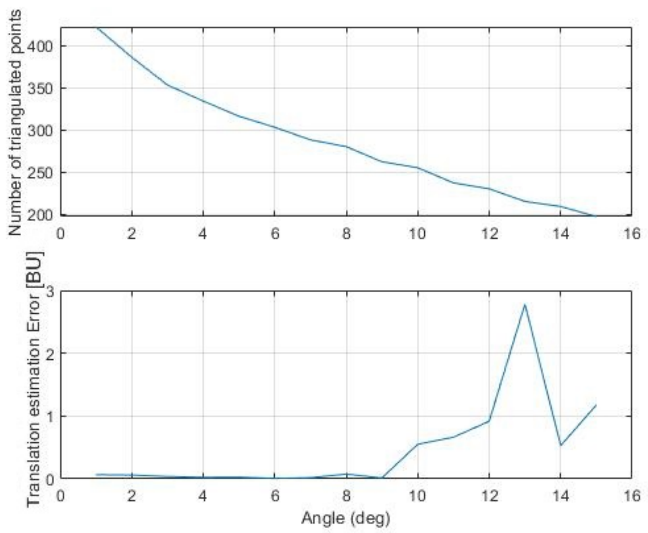
Fig. 7.5: Angle between views and uncertainty on 3D points [3]

is the same as in Section 4.1. The initialization step is performed starting with the Essential matrix estimation, its decomposition into rotation and translations parts and then 3D points triangulation. Finally a ray-casting like algorithm is used to compute the real position of the 3D points that correspond to the estimated ones.

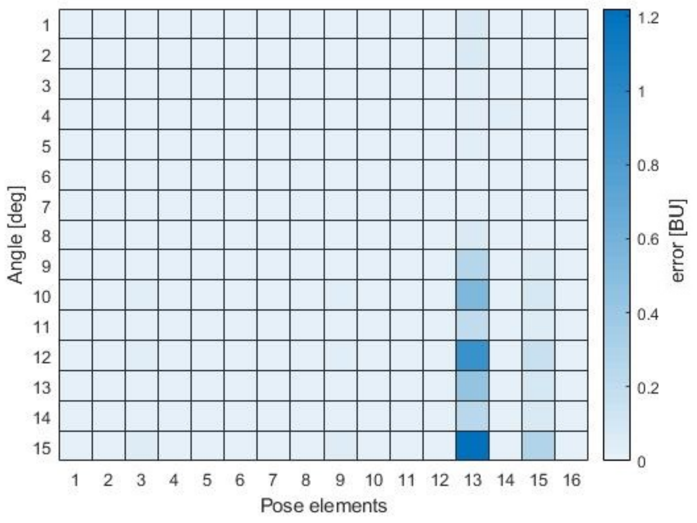
Fig. 7.6 shows the results for angles ranging from 1 to 15 [deg]. Fig. 7.6a in particular denotes how the error mean on the 3D points reconstruction varies with the viewing angle. Both small separation angle between views and relatively high ones are not ideal for reconstructing world points and a plateau can be found for some values of the viewing angle. As shown in Fig. 7.6b, even though very small viewing angles do not produce a high error in the translation estimation, the error associated to the 3D point reconstruction is high and might be problematic in the subsequent trajectory estimation. Moreover the larger the viewing angle, the more difficult it is for the tracker to find point tracks.



(a) Estimated 3D points accuracy



(b) Triangulated points number and translation error



(c) Pose matrix elements and error

Fig. 7.6: Results for 3D points accuracy test

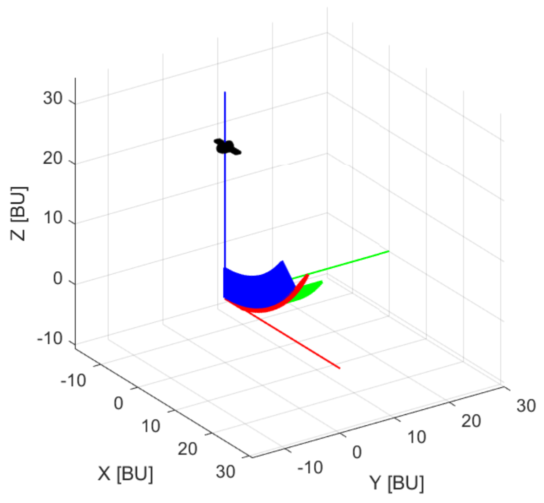
7.3 Results

In the following subsections test cases of the V-SLAM algorithm are reported. The initialization step is performed by tracking points between the first frame and the next ones until a proper separation angle is achieved. If this condition is not met, then another condition to force the algorithm to start based on the number of previous frames can be introduced.
Images are obtained through the image generation software by imposing a specific trajectory in space without considering pointing errors possibly arising from the chief spacecraft. The target attitude is not commanded and, in case of a rotating one, the camera exposure must be fast enough not to cause blurring. Images and relative pose data is transferred to Matlab where the V-SLAM algorithm takes place. Features are detected through the minimum eigenvalues algorithm and tracked through the KLT algorithm. In order to reduce the accumulating tracking error of the KLT tracker, its bidirectional error can be imposed to be low enough which, depending on the wanted precision, has been set to $10^{-1}/5 \cdot 10^{-3}$ [px]. Tracks are saved in a data-store which needs to be managed carefully due to the different validity vectors and indexes that need to be used. Features are extracted, already detected ones are removed and then the remaining ones are tracked for triangulation. Triangulation after the initial mapping is performed between two non consecutive images spaced by a few number of frames in order to achieve good 3D point reconstruction and is executed when tracked 3D points become too low. Camera parameters are the same as in Section 3.2, with a focal length of $f = 150$ [mm], sensor size of 36×20.25 [mm] and image resolution of 1152 by 648 [px].

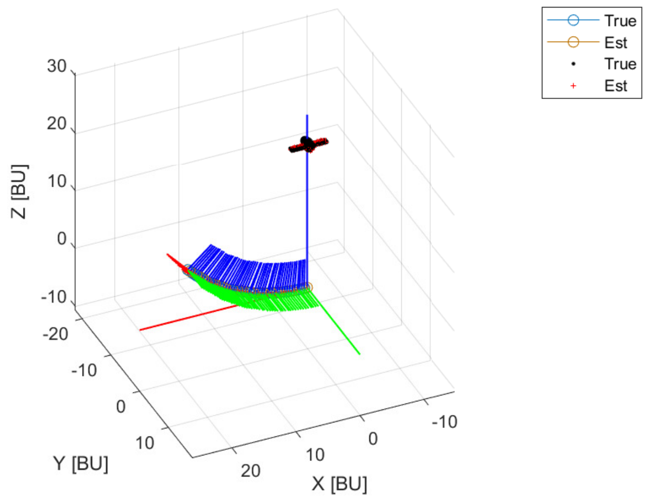
Test number 1 in Section 7.3.1 starts with a standard partial circular trajectory achieving good results. In test case 2 a closed trajectory is imposed while in test case 3 a rectilinear one is set, both of which can potentially be challenging for the algorithm. Finally in test case 4 a demanding natural scenario is introduced by means of the comet 67P/Churyumov-Gerasimenko while in test case 5 bundle adjustment optimization is examined. From these tests, performed with no a priori information about the target, good navigation performance is obtained. It emerges that proper initial mapping process must be ensured in order to correctly continue the navigation process. Good feature detection is fundamental as features are the only source of motion information and enough of them must be found in order to picture the overall motion. The tradeoff for the feature tracking error threshold is also a delicate process, necessary to ensure a good amount of tracked features versus their quality. Limitations are also present, such as the Sun appearance in view inducing sensor saturation or motion blur from a spinning spacecraft, which represent challenges to be addressed in the future.

7.3.1 Test case 1

In this first test a simple circular trajectory has been chosen, as in Fig. 7.7a. Images are processed as stated before (Fig. 7.8) and the estimated trajectory is shown in Fig. 7.7b. 3D points composing the map of the V-SLAM algorithm are presented in Fig. 7.9, showing that they are consistent with the actual points of satellite mesh. Finally translation and orientation errors are plotted in Fig. 7.10. The quaternion error is small and can be neglected, while the translation error is more noticeable. In percentages, it is about 1% of the range distance, which translates to a maximum of 0.6 [m] of error per component over a 75 [m] range. The execution time is in the order of tenth of a second at maximum, as in Fig. 7.10c, depending on whether new features are extracted or not. This results in a mean time per frame of 0.057 [s] and mean fps, frames per second, value of 17.6 frames.

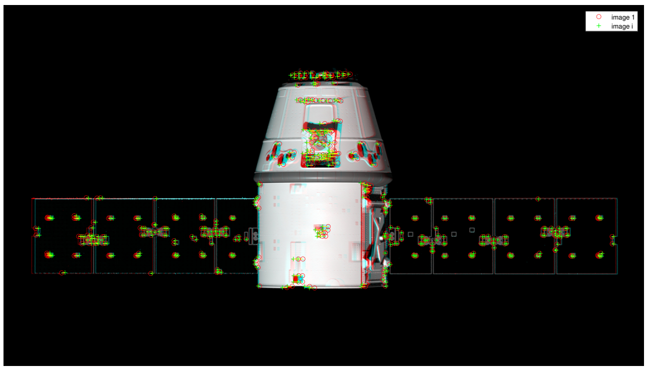


(a) Reference trajectory

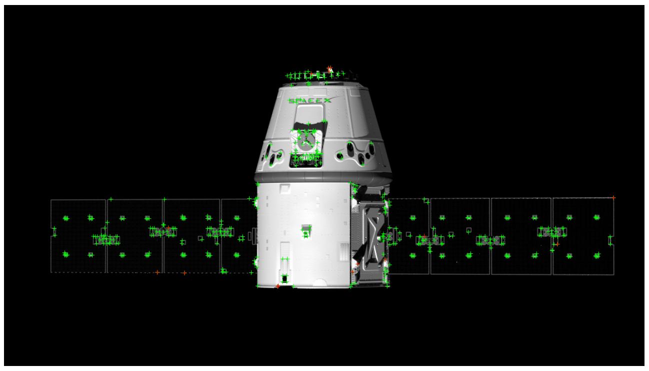


(b) Estimated trajectory

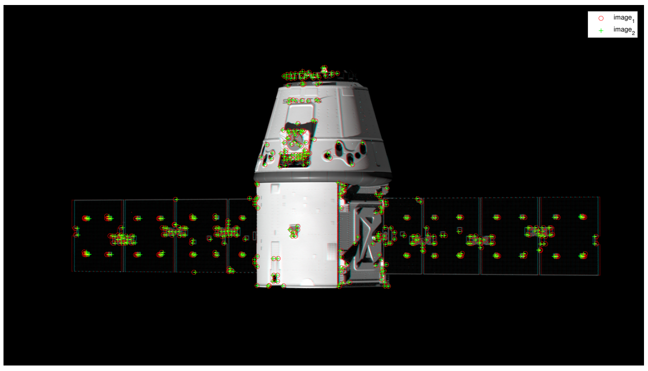
Fig. 7.7: Reference and estimated trajectories



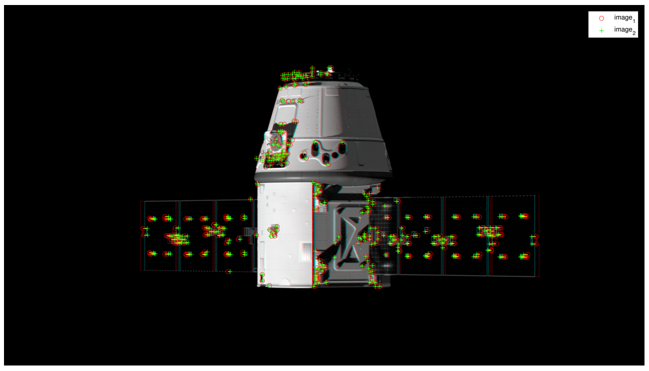
(a) Initialization tracking



(b) Detection of new points in red, previous points in green

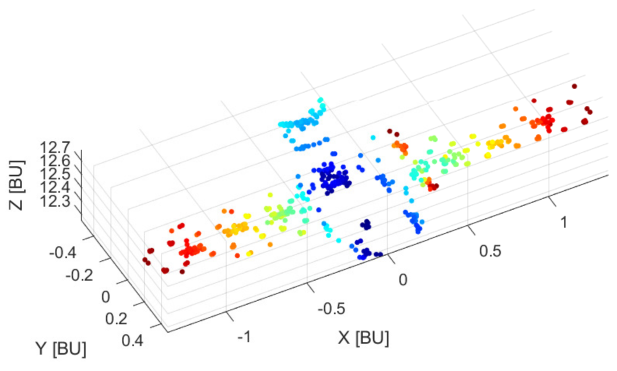


(c) Tracking example, frame 29 to 30

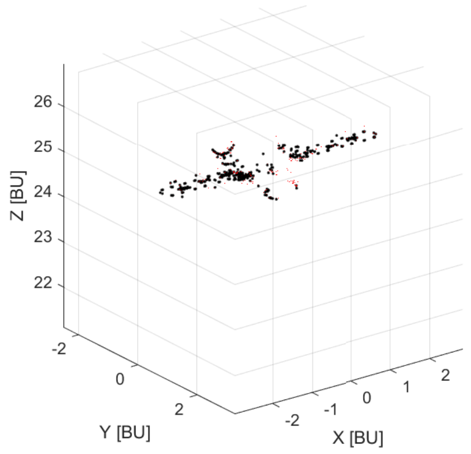


(d) Tracking example, frame 49 to 50

Fig. 7.8: Examples of feature detection and tracking

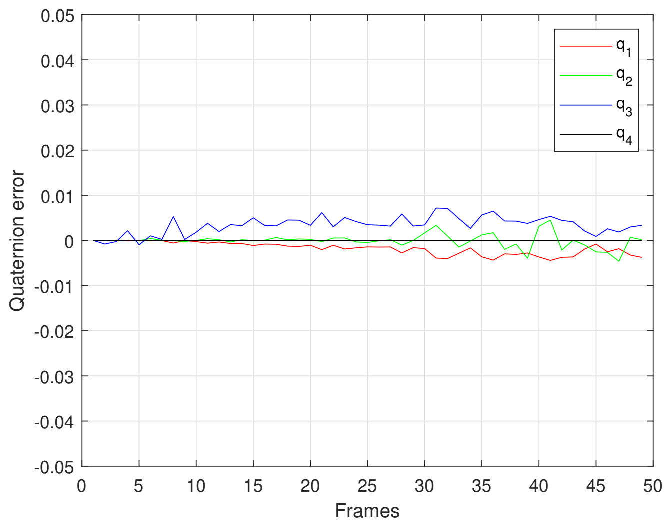


(a) 3D points reconstruction

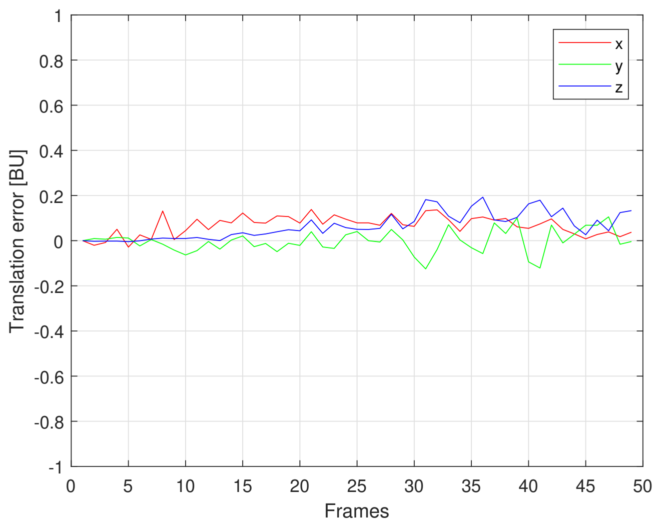


(b) Actual 3D points in black, estimated in red

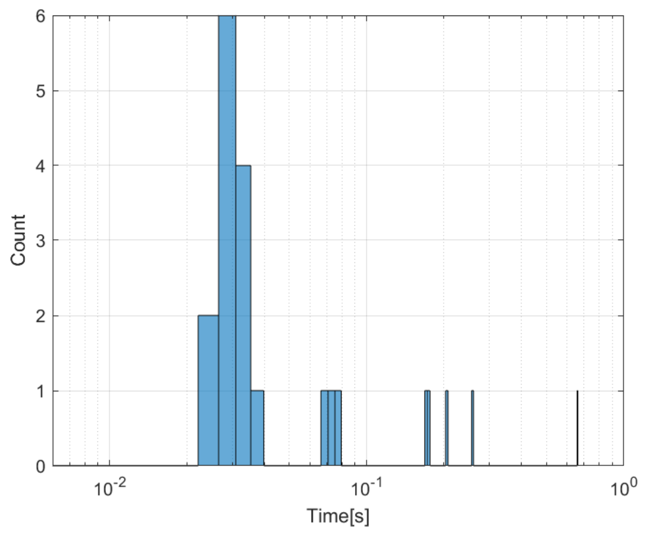
Fig. 7.9: Actual and estimated 3D points



(a) Quaternion error



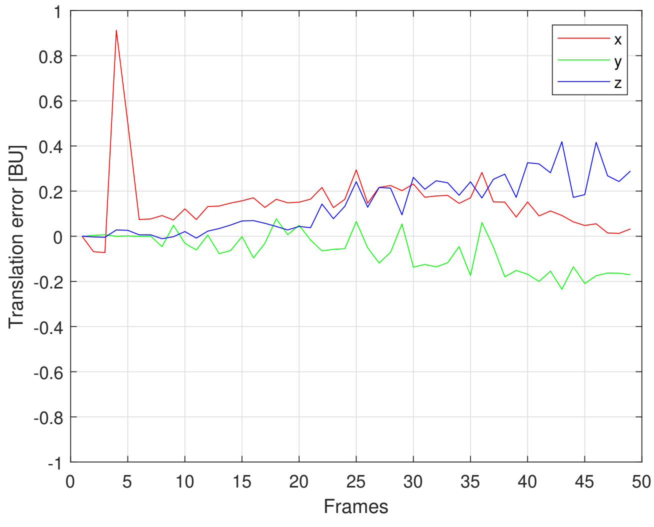
(b) Translation error



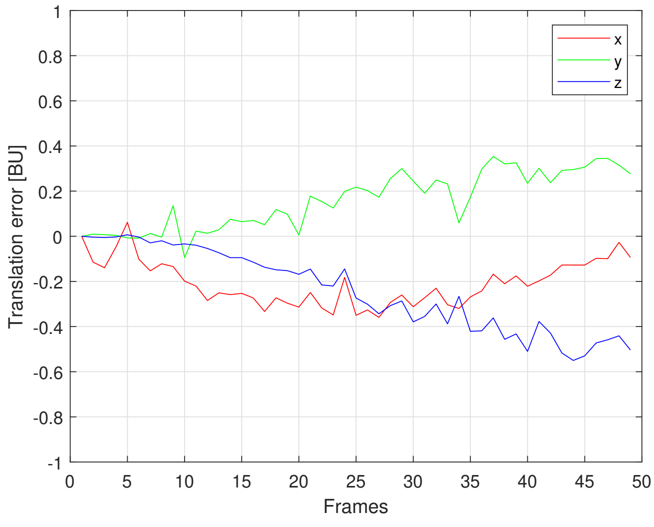
(c) Execution time histogram

Fig. 7.10: Test case 1 results

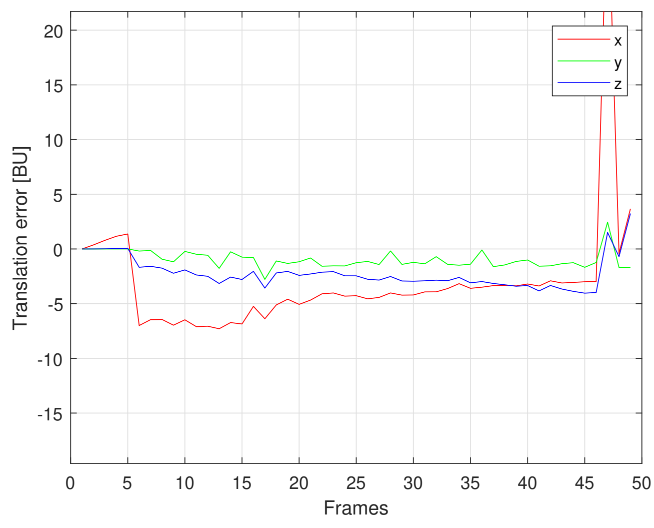
In the following figures the translation error for other feature types is reported. BRISK, FAST, ORB and Harris features are used and every simulation has been executed few times before extracting the translation error. All of them provide worse results with respect to the Minimum eigenvalues features and thus MinEig algorithm has been selected for feature extraction.



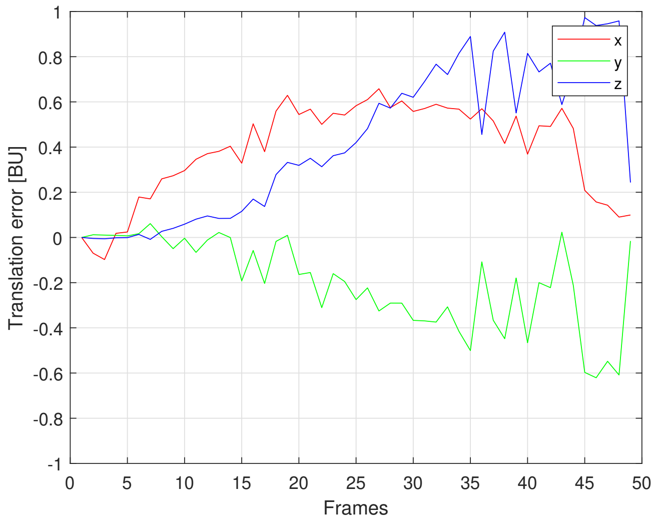
(a) BRISK feature detector



(b) FAST feature detector



(c) ORB feature detector

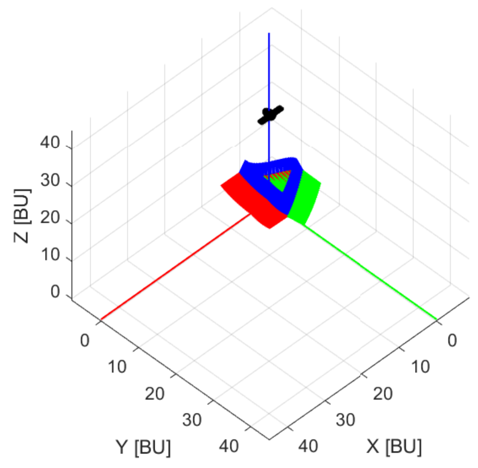


(d) Harris feature detector

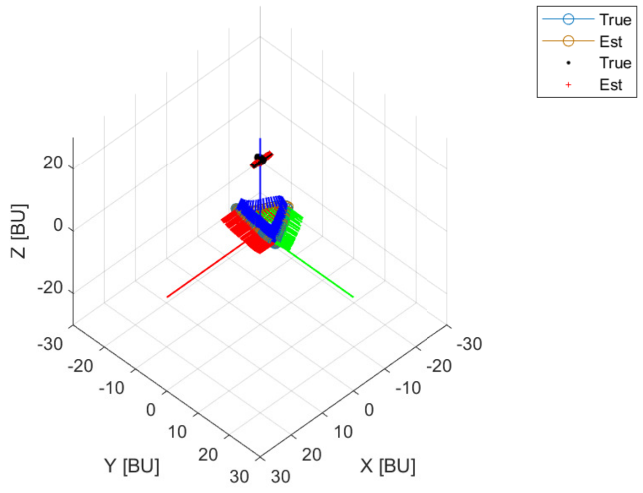
Fig. 7.11: Translation error for different feature detectors

7.3.2 Test case 2

In this second test a triangular shaped trajectory is used to check the consistency of the algorithm when returning to an already visited point. The reference trajectory is shown in Fig. 7.12a and the estimated one in Fig. 7.12b. Some images are provided in Fig. 7.13. 3D points composing the map are reported in Fig. 7.14 showing again no evident deviation from the true 3D points. Translation and orientation errors are plotted in Fig. 7.15, not being particularly different in magnitude from the previous example. Both of them tend to increase when moving away from the original position in which the map is generated, but decrease when the spacecraft is brought back to the starting point. The execution time is again in the order of tenth of a second, as in Fig. 7.15c, reaching a maximum of 0.3 [s].

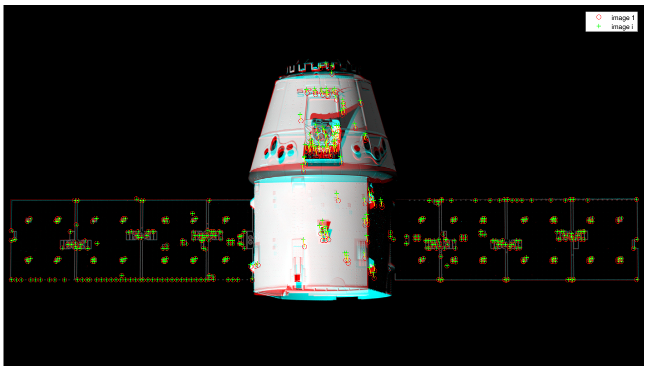


(a) Reference trajectory

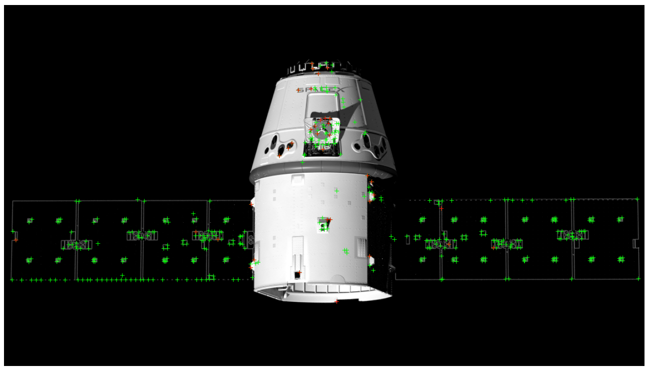


(b) Estimated trajectory

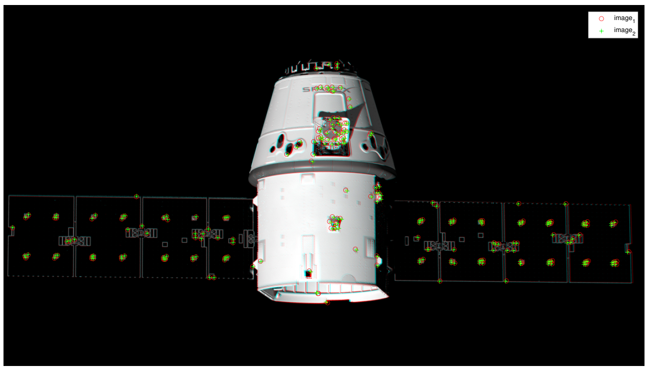
Fig. 7.12: Reference and estimated trajectories



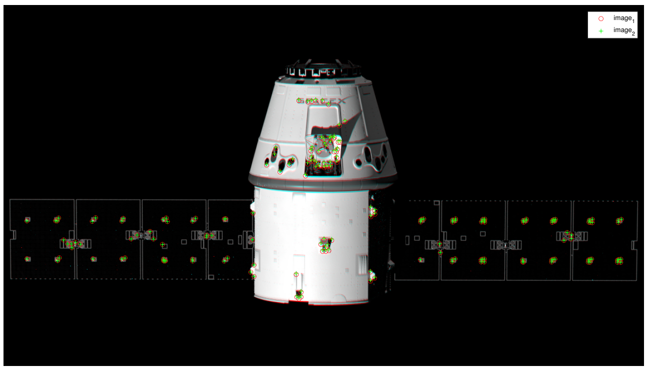
(a) Initialization tracking



(b) Detection of new points in red, previous points in green

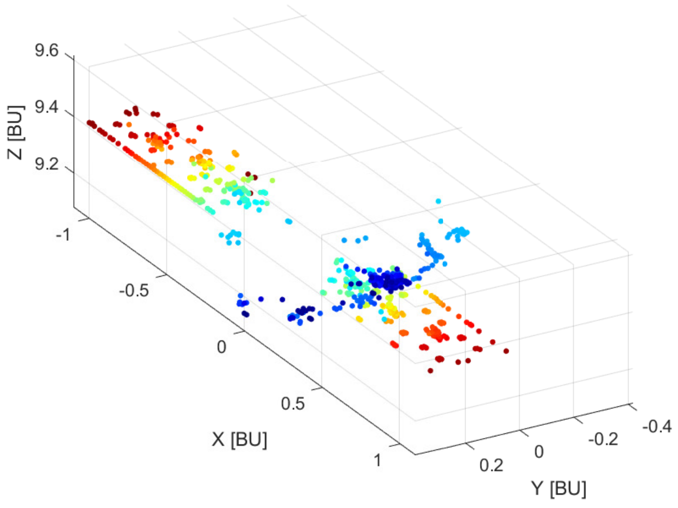


(c) Tracking example, frame 29 to 30

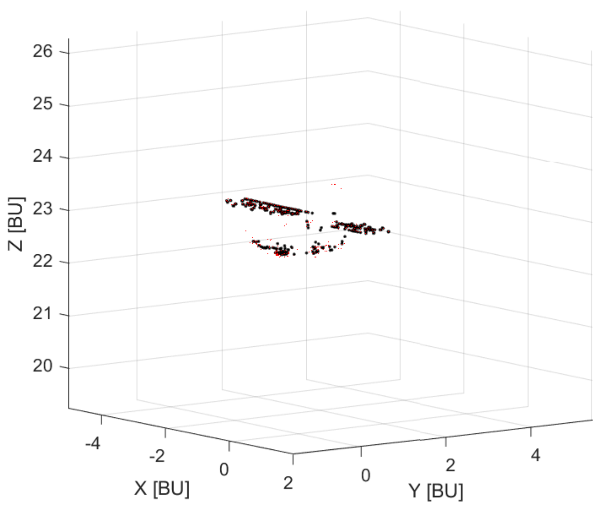


(d) Tracking example, frame 74 to 75

Fig. 7.13: Examples of feature detection and tracking

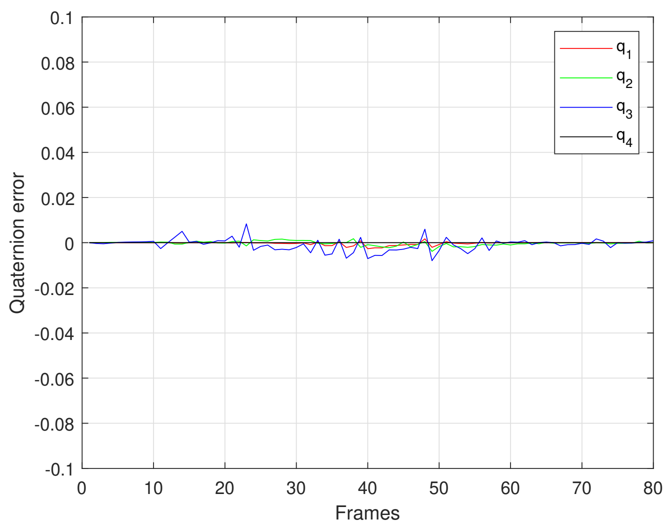


(a) 3D points reconstruction

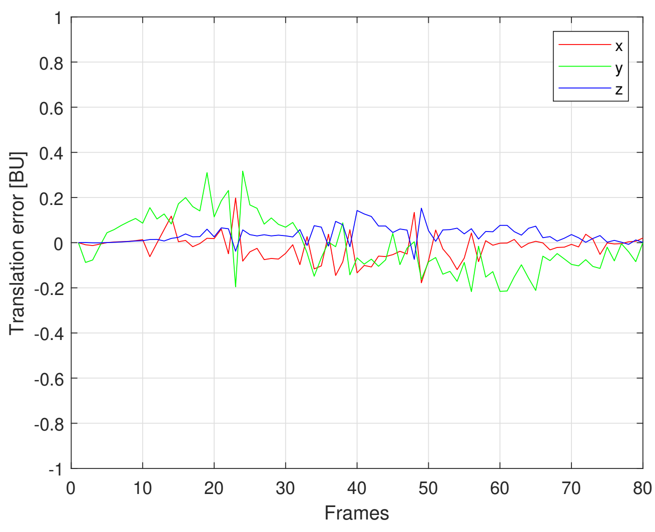


(b) Actual 3D points in black, estimated in red

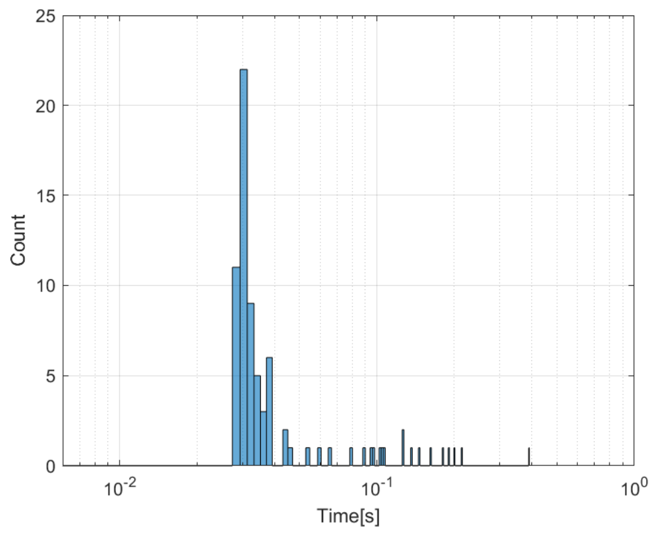
Fig. 7.14: Actual and estimated 3D points



(a) Quaternion error



(b) Translation error

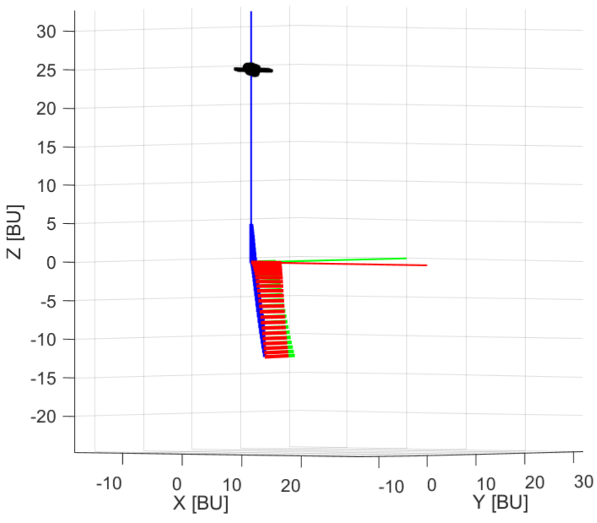


(c) Execution time histogram

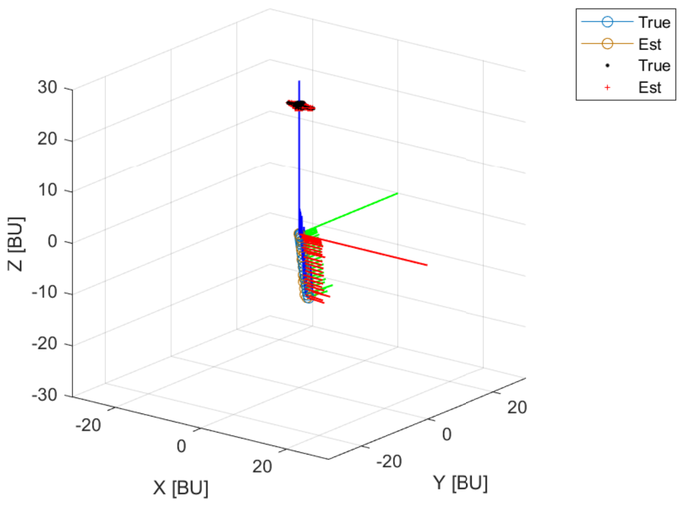
Fig. 7.15: Test case 2 results

7.3.3 Test case 3

The third test deals with a peculiarity of V-SLAM algorithm that affects specific trajectories. A fully rectilinear trajectory with no attitude change does not provide enough perspective information on the target object to build a consistent map. The trajectory, an almost straight departing one, is shown in Fig. 7.16a, while the estimated one is present in Fig. 7.16b. Images are again provided in Fig. 7.17. 3D points composing the map are reported in Fig. 7.18 and are not consistent with the actual 3D points. Even if translation and orientation errors are not extremely affected, navigation through the map in Fig. 7.18b will definitely jeopardize the future pose measurements. The initialization step must thus be provided with enough information to produce a correct triangulation and these kind of trajectories should be avoided.

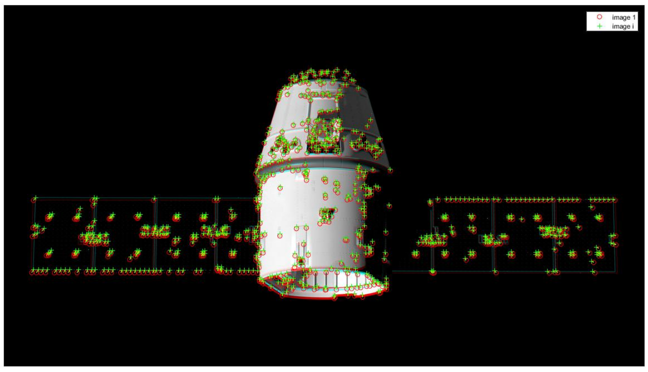


(a) Reference trajectory

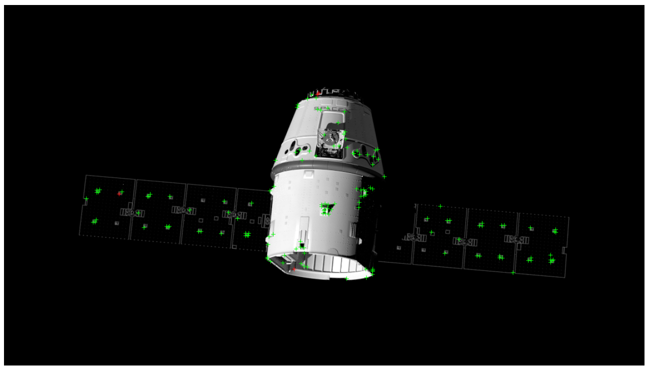


(b) Estimated trajectory

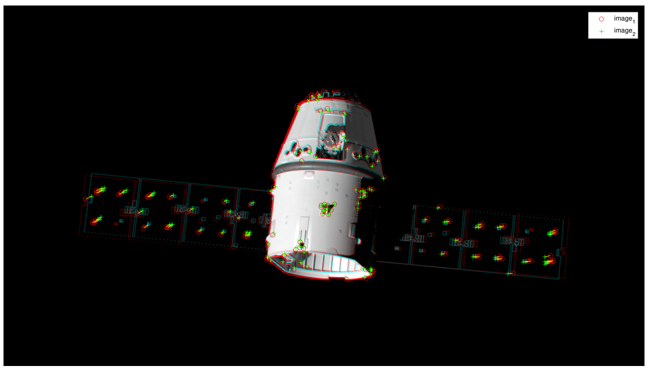
Fig. 7.16: Reference and estimated trajectories



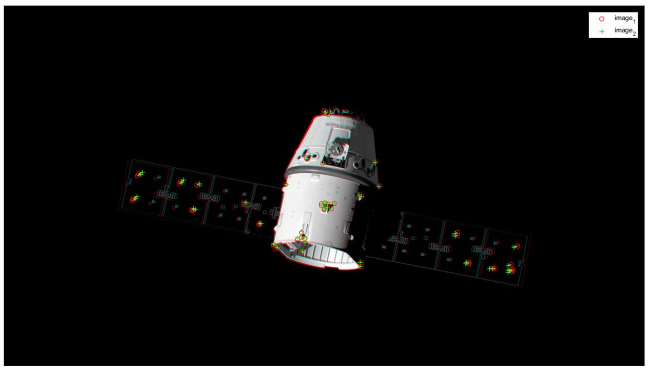
(a) Initialization tracking



(b) Detection of new points in red, previous points in green

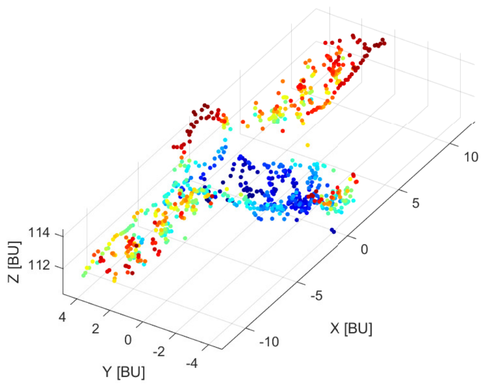


(c) Tracking example, frame 14 to 15

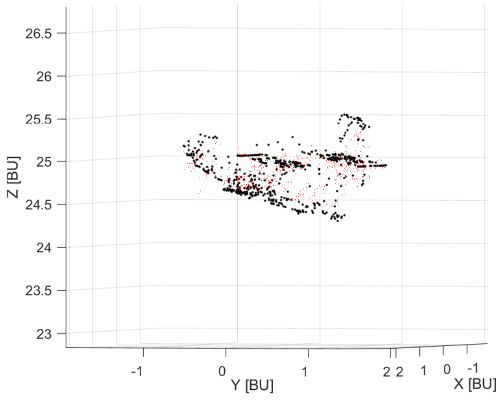


(d) Tracking example, frame 24 to 25

Fig. 7.17: Examples of feature detection and tracking

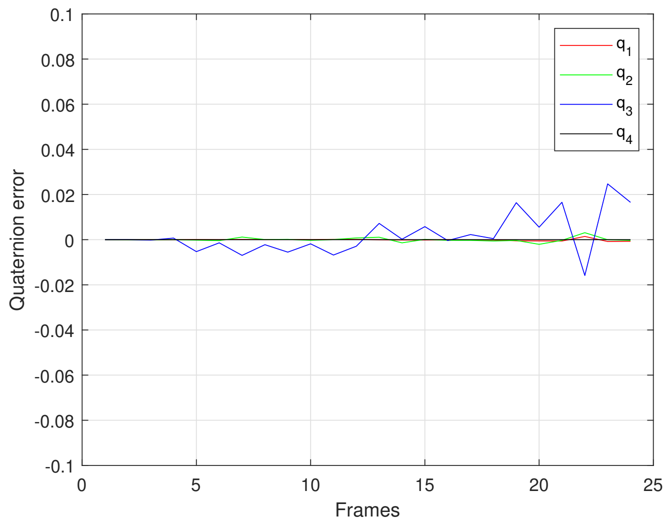


(a) 3D points reconstruction

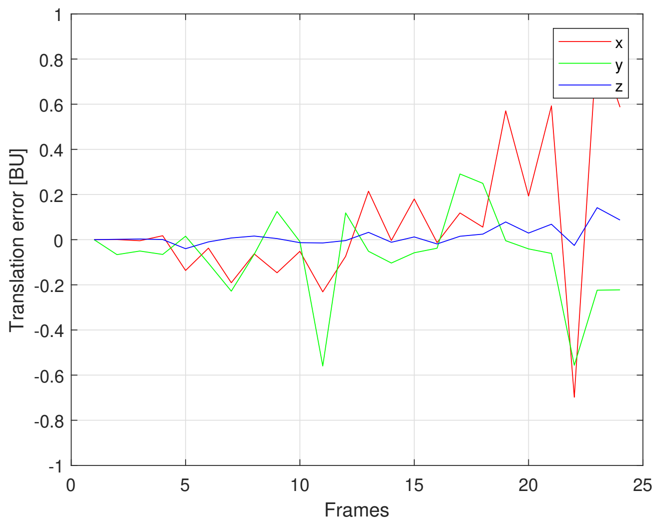


(b) Actual 3D points in black, estimated in red

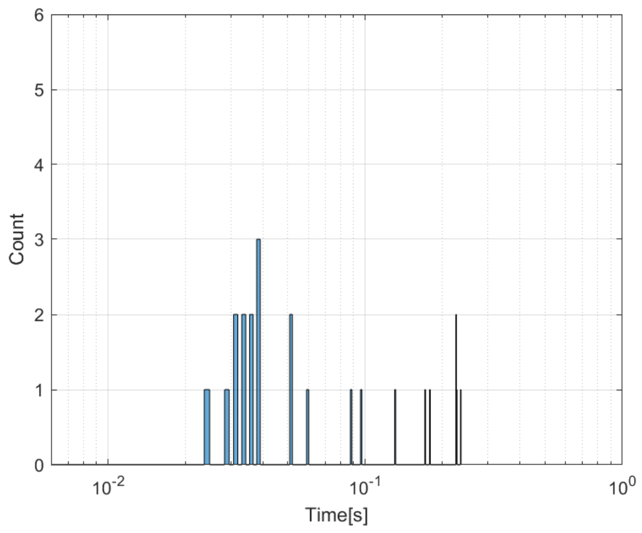
Fig. 7.18: Actual and estimated 3D points



(a) Quaternion error



(b) Translation error

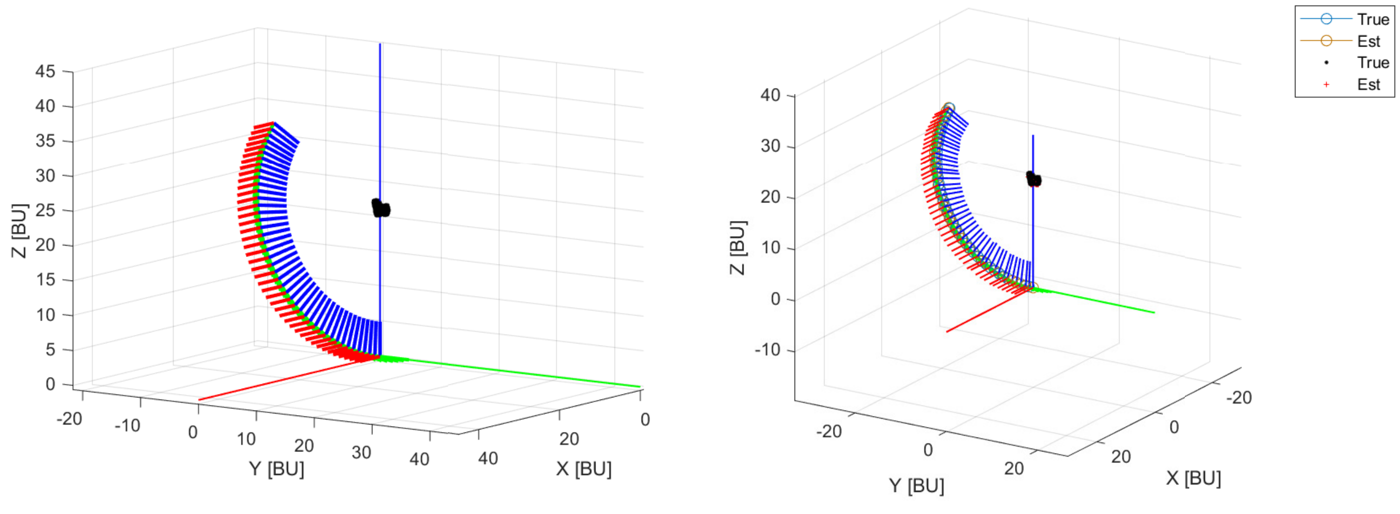


(c) Execution time histogram

Fig. 7.19: Test case 3 results

7.3.4 Test case 4

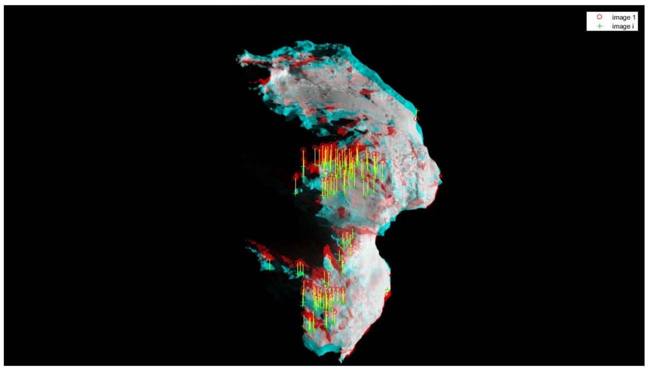
In this fourth test a circular trajectory is used with a different object as target. A natural scene is introduced by means of the comet 67P/Churyumov-Gerasimenko, with the shape model provided in [84]. The model does not include a texture, which has procedurally been generated in Blender to be as realistic as possible. The reference trajectory of about 32 [km] radius is shown in Fig. 7.20a, the estimated one in Fig. 7.20b and images used by the algorithm in Fig. 7.21. As it is possible to see from these last ones, a natural scene can provide denser region of features to be tracked thanks to the geological surface features. However some texture-less regions can also be present, as in the right side of Fig. 7.21d. This scenario is very challenging for the algorithm in terms of feature detection, which has been applied mainly to artificial objects, however it is still shows good results. 3D points composing the map are reported in Fig. 7.22 showing good accuracy in the reconstruction. Relative pose errors are plotted in Fig. 7.23. Translation error reach up to 0.4 [BU] per component, or 600 [m] which, depending on the wanted accuracy, might be too high. In that case bundle adjustment with loop closure or sensor fusion techniques, for example with an altimeter or IMU, should be sought. In this simulation new map points are added to the map as soon as possible, every third iteration, as feature points are more easily lost with respect to the other tests. This induces a better accuracy during navigation but also increases the number of frames that require execution times in the order of 0.1 [s]. The average fps value decreases to 5.4 frames per second.



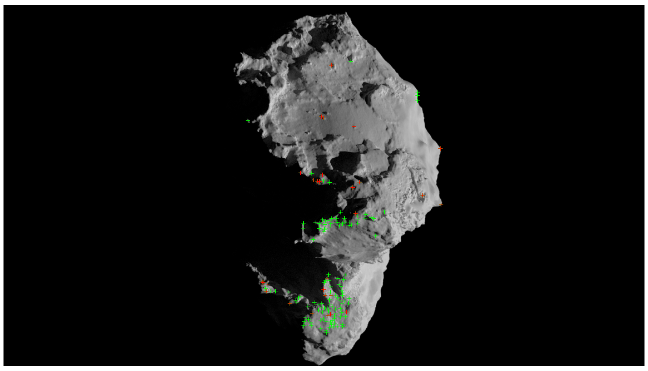
(a) Reference trajectory

(b) Estimated trajectory

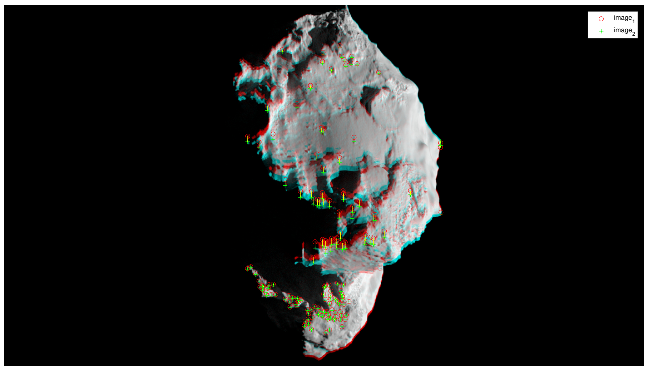
Fig. 7.20: Reference and estimated trajectories



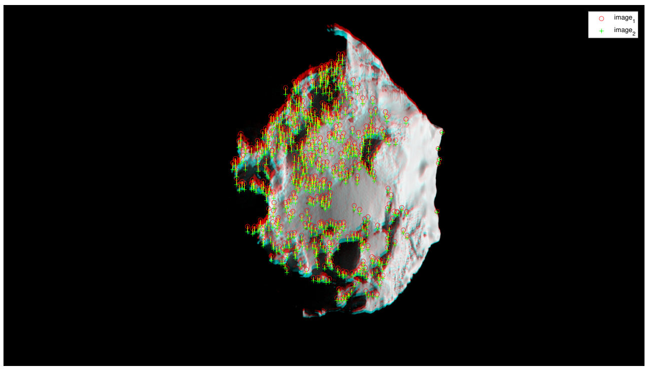
(a) Initialization tracking



(b) Detection of new points in red, previous points in green

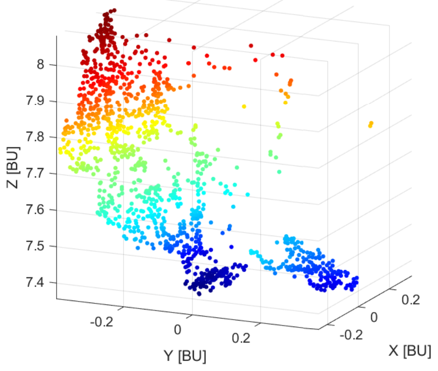


(c) Tracking example, frame 14 to 15

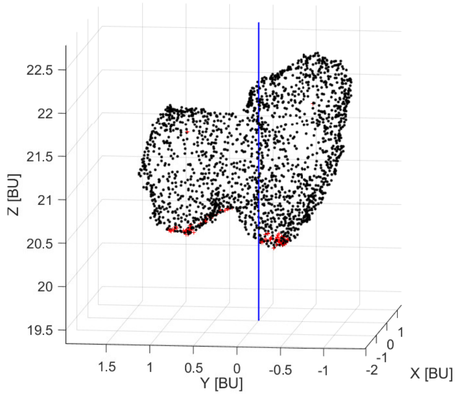


(d) Tracking example, frame 24 to 25

Fig. 7.21: Examples of feature detection and tracking

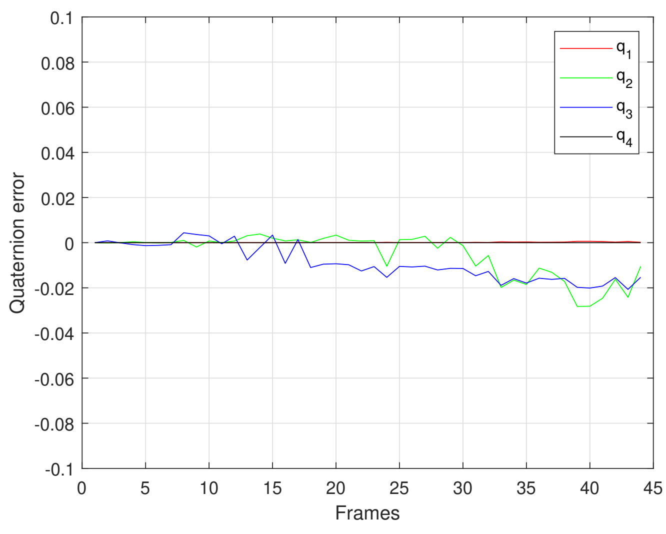


(a) 3D points reconstruction

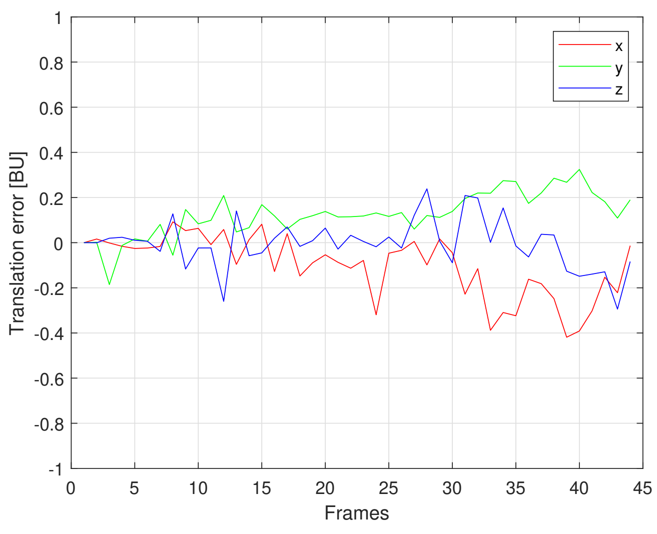


(b) Actual 3D points in black, estimated in red

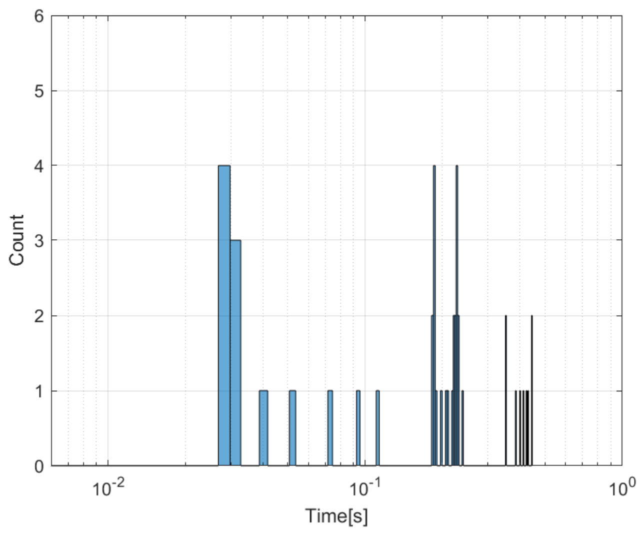
Fig. 7.22: Actual and estimated 3D points



(a) Quaternion error



(b) Translation error

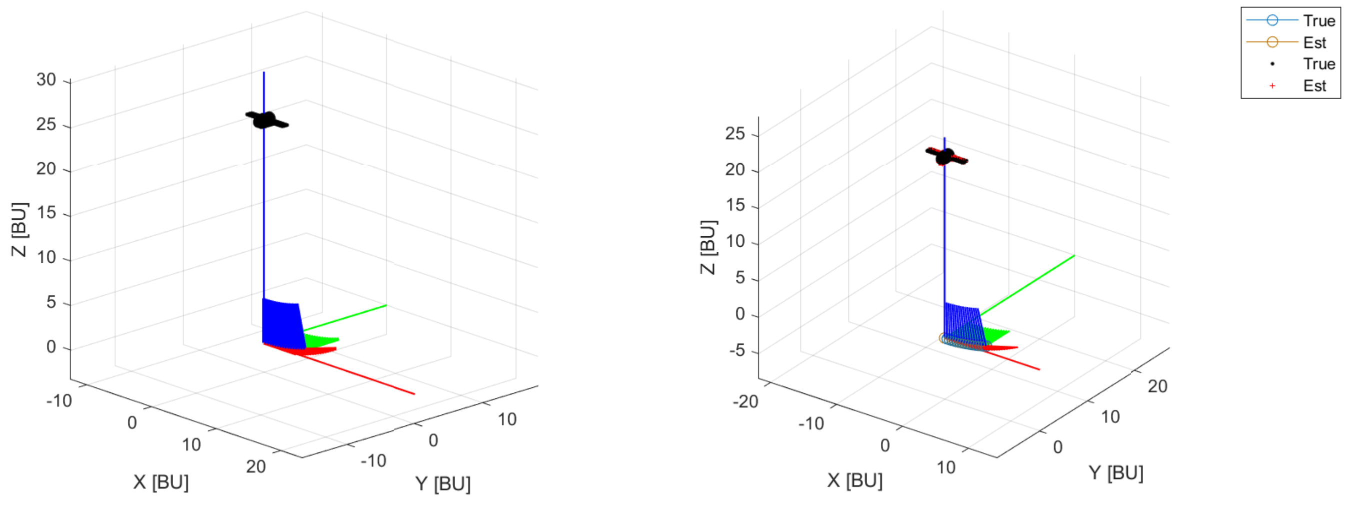


(c) Execution time histogram

Fig. 7.23: Test case 4 results

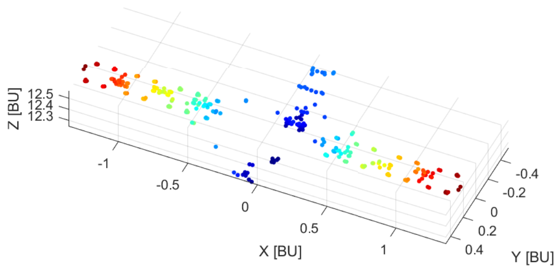
7.3.5 Test case 5

In this last test the effect of bundle adjustment on the initial part of the Test case 1 is studied. Bundle adjustment is run once at the end of the simulation and takes 0.35 seconds to run. The estimated trajectory, as in Fig. 7.24b, shows very good accuracy with respect to the reference one in Fig. 7.24a. 3D points are plotted in Fig. 7.25 while errors are presented in Fig. 7.26. Those last ones are lower in value with respect to the non-optimized version and show much better behaviour in terms of variability. Thanks to bundle adjustment a more coherent simulation can be obtained with a more refined map. Windowed or key-frame based bundle adjustment would prevent the computational cost of the simulation from exploding due to the large number of operations needed, but with future development in space-grade processors it will be an useful tool for space related scenarios.

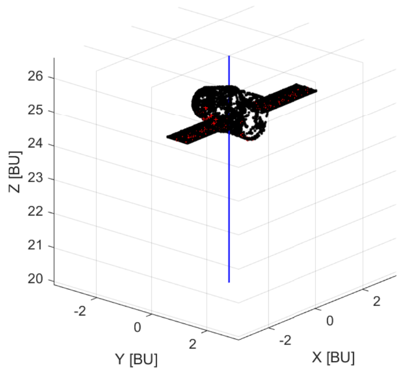


(a) Reference trajectory (b) Estimated trajectory

Fig. 7.24: Reference and estimated trajectories

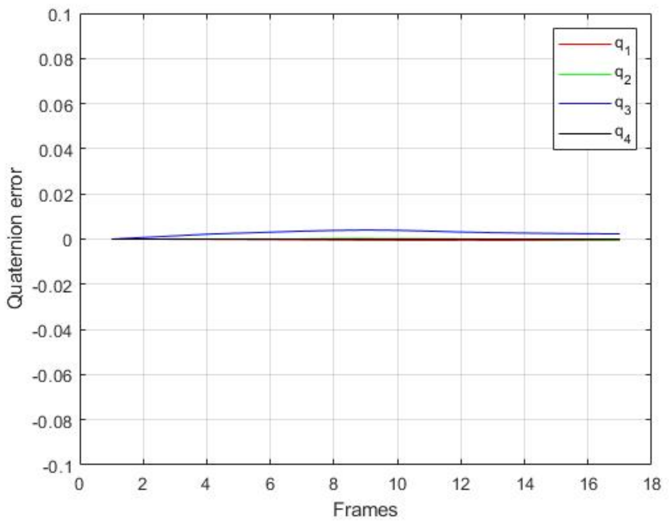


(a) 3D points reconstruction

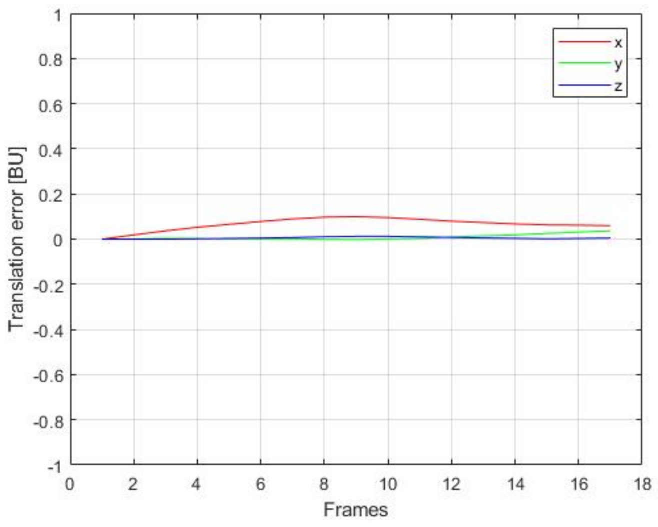


(b) Actual 3D points in black, estimated in red

Fig. 7.25: Actual and estimated 3D points



(a) Quaternion error



(b) Translation error

Fig. 7.26: Test case 5 results

8 CNN ROI Design and results

In this chapter the process of training and testing of the CNN for ROI estimation is presented. The training, validation and testing datasets are generated first for the reference design scenario. The neural network is trained on these dataset and a trade-off is performed to optimize the network performance through its architecture definition. The CNN is then analyzed in its nominal scenario and in off-design ones. The off-nominal conditions are obtained by varying the target object and the natural body in the background or by adding noise or blur. Finally an example of V-SLAM trajectory with CNN ROI estimation is provided.

8.1 Dataset generation and description

The first step in the implementation of a CNN structure is the generation of a large enough dataset for training. CNN need a lot of data in order to learn which features to detect and which one to leave out, and generally a large dataset implies greater performance of the network along with the added computational cost of the dataset generation itself and longer training time.
The dataset is generated, in this thesis, using Blender script detailed in Section 6.3. Its output is composed of an image folder containing 16000 images and a file containing the pose elements for target and inspector spacecraft written in inertial and relative frames. Examples of some images are reported in Fig. 8.2.

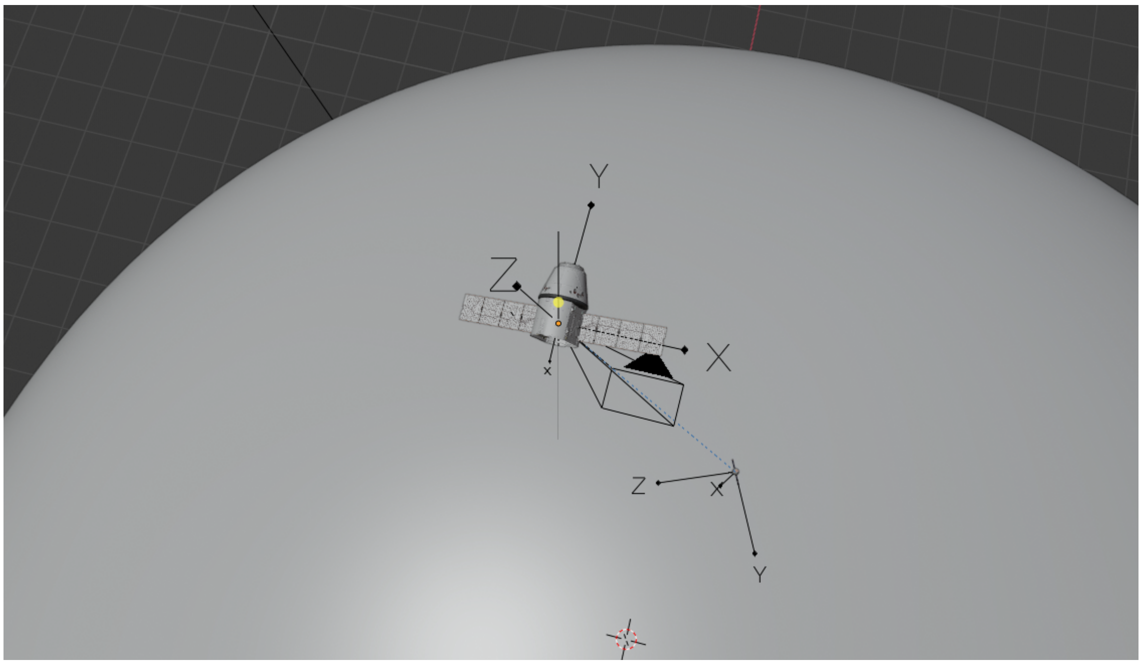
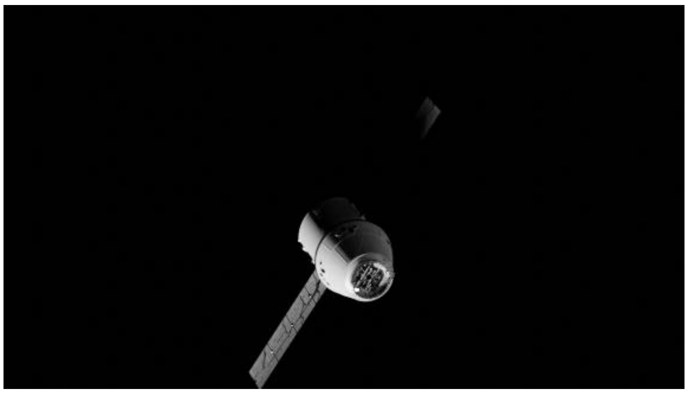
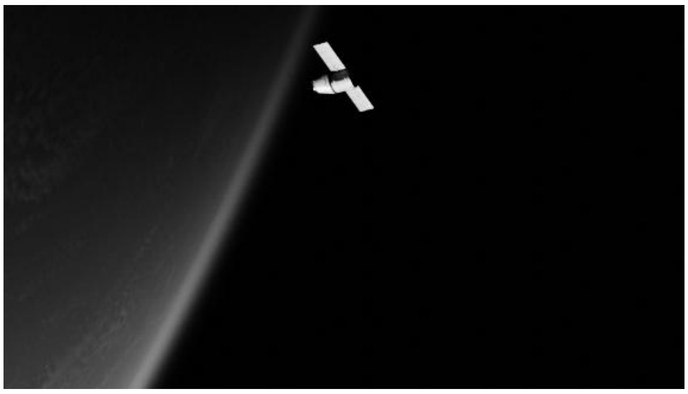
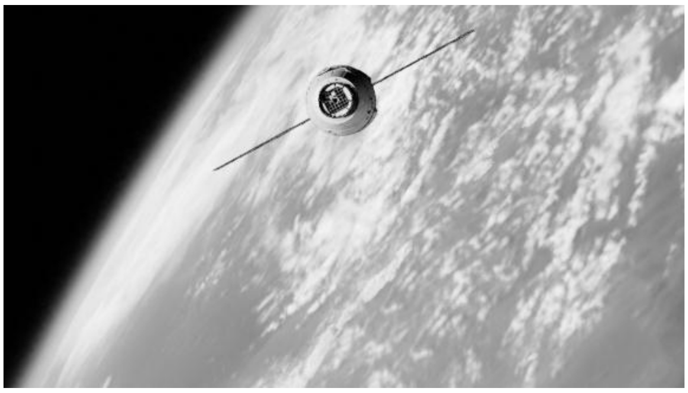


Fig. 8.1: Dataset generation environment

A tradeoff between image quality an memory available was necessary as the mini-batch size for 1152x648 resolution images was very small and leading to training times in the order of hours. The GPU card deployed for training, an RTX 2070 Super, is provided with 8 GB GDDR6 memory which would quickly fill up during training computations. The only viable solution was to reduce the memory requirements by converting images to half their size at 576x324. Images are also converted in BW from RGB to reduce again the dimension of the images from three to one.


(a) Example 1


(b) Example 2


(c) Example 3

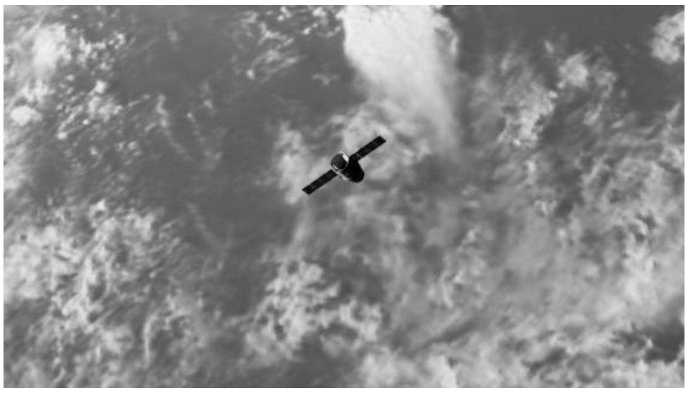
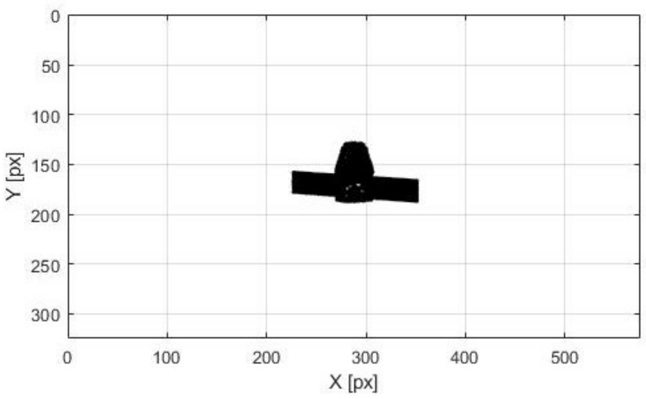

(d) Example 4

Fig. 8.2: Examples of dataset images

Outliers need to be dealt with too. Some images are indeed completely black due to the fact that the Sun and Earth can be in line with the spacecraft, so that the chief and deputy are in eclipse. Images with very low illumination are discarded too. About 40% of the images are considered not valid and discarded, leaving roughly 10000 images to train the model with. Of the remaining images about 25% of them contain the Earth in the background. As described in Section 6.3 a 1500 [km] LEO orbit has been selected. The absolute position of both spacecraft is randomly selected but the relative distance is constrained to range from 100 [m] to 600 [m]. After an initial tradeoff, the Sun position in frame has been selected to have a random azimuth and a minimum elevation angle of 90 [deg] in order to retain more images with better illumination quality. A random perturbation has been assigned to the camera to move the center of mass of the target spacecraft up to the image edges.

Once data is transferred to Matlab, the bounding box can be generated. The imaging scenario is recreated in Matlab using previously described projective geometry theory, with inertial and relative pose data acquired from a database generated in Blender. The coordinates of the target spacecraft on the sensor are then obtained. Two transformations are needed in order to retrieve the sensor image, one going from the 3D world reference frame to the 3D chief spacecraft frame and one from the 3D chief spacecraft frame to the sensor one. An example of this process is provided in Fig. 8.3. The bounding box has been parametrized by using the x and y coordinates of the top left corner of the bounding box and its width and height. An attempt to change the bounding box parameters to its center coordinates and its half width/height has been made, however results of the trained network were visually worse in quality with respect to the top-left corner parametrization. Moreover some considerations have been made in deciding which mesh points of the target objects are considered to be valid and can contribute to the bounding box definition. Indeed very dark pixel regions of the target satellite, of intensities lower than 20 units are not used to define the bounding box to prevent the CNN from learning the exact satellite shape.



(a) Pinhole camera model with Blender data results

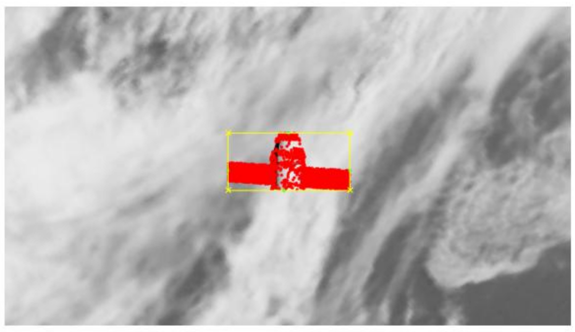

(b) Bounding box generation

Fig. 8.3: Bounding box generation process

8.2 CNN Training and Architecture Tradeoff

CNN architectures can be specified by changing many parameters that affect their performances. A continuous training and architecture tradeoff is thus needed to converge towards better and better results. The most important parameters that can be varied in a CNN are mainly the number of layers, the number of filters in each layer and the filter sizes. Other relevant parameters include for example the learning rate, the minibatch size and the train/test/validation splitting percentages. Architecture tradeoff is also necessary to avoid under/overfitting: a small neural network might not be able to recognize important patterns as its capability to learn is limited, denoting the underfitting condition. However a large neural network could also undergo the inverse problem, that its performance is so high on the training set that it will learn to recognize only specific patterns and won't be able to generalize well on unknown objects.

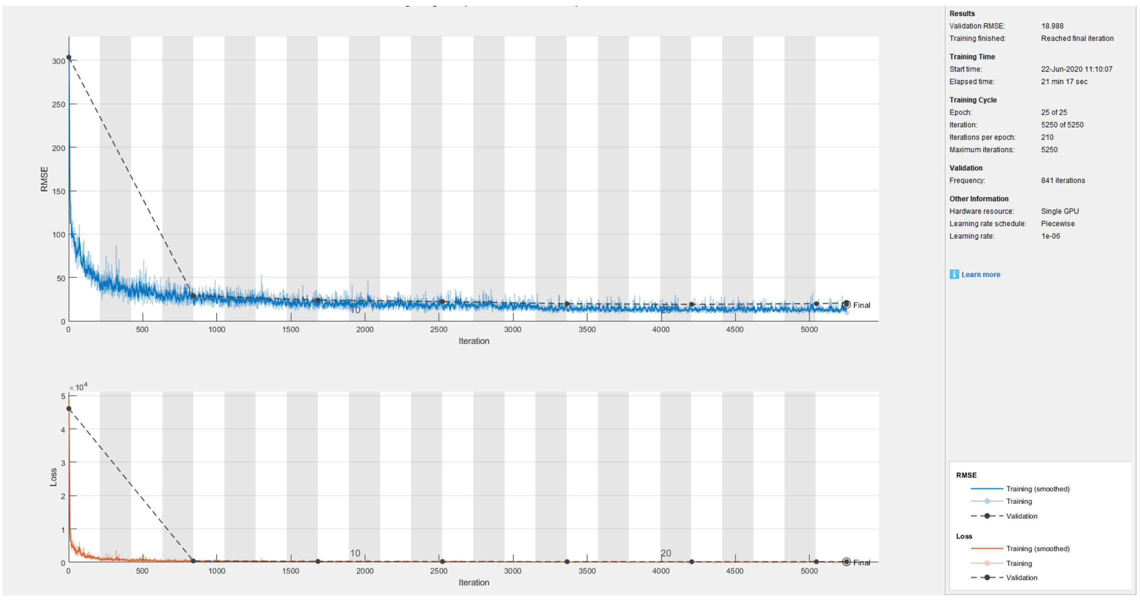
Various architectures have been proposed by varying the parameters mentioned before. The minibatch size of 32 frames has been used to reduce the training time, which is otherwise very long. Every training, performed on an R7 3700X and RTX 2070 super, typically lasts from 30 to 45 minutes depending on the architecture complexity. The layers have been continuously increased until no major improvements were obtained or until the training time became too expensive. In the architecture, the progressively increasing filter number has been used to enhance the network complexity, as done in past design, for example in [85]. A compromise has been performed to split the dataset in the training/validation and testing sets in order to retain more data in the training set. For this reason, the training set consisted of 70% of the total pictures in the dataset while the other two were both split at 15%. The learning rate was set at $1e-5$ and no major improvements have been registered by varying its value. 30 different architectures have been trained with varying level of accuracy in the results and a list of some selected networks is available in Table 4. RMSE is presented as testing (T) and validation (V). The minibatch column reports the size of the minibatch and LR reports the learning rate at the starting fifteen epochs and in the last ten epochs of the training. The "TrainTestVal" column shows the splitting of the dataset into the training, testing and validation ones. In the architecture column a summary of the convolutional layers layout is given. A single layer is denoted with its filter size followed, after a dot, with the filters number. In between any blocks of convolutional layers a max pooling operation is performed, while batch normalization and ReLu activation function is used after every convolutional layers. In the last column, BPR, black pixel removal, it is indicated if an algorithm to remove part of the bounding boxes containing only black pixels is used. This algorithm has been implemented to prevent the CNN from guessing the shape of the target object when no clear visual clue is available, which for example happens when a black solar panel is imaged with black space in the background.

Table 4: CNN architecture tradeoff

RMSE T [px]	RMSE V [px]	MiniBatch	LR	TrainTestVal [%]	Architecture	BPR
34.88	35.52	16	1e-5, 1e-6	70/15/15	3.16, 3.32, 3.64, FC4	-
26.82	27.25	24	1e-5, 1e-6	70/15/15	3.16, 3.32, 3.64, 3.128 FC4	-
20.16	20.24	24	1e-5, 1e-6	70/15/15	3.16, 3.32, 3.64, 3.128, 3.256 FC4	-
17.50	18.03	24	1e-5, 1e-6	70/15/15	3.16, 3.32, 3.64, 3.128, 3.256, 3.256 FC4	-
17.24	17.62	16	1e-5, 1e-6	70/15/15	2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	-
17.95	17.88	16	1e-5, 1e-6	70/15/15	4x3.16, 3x3.32, 2x3.64, 2x3.128, 3.192, 3.256 FC256 FC32 FC4	-
13.81	14.03	32	1e-5, 1e-6	70/15/15	2x3.8, 2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	-
10.89	9.49	32	1e-5, 1e-6	70/15/15	2x3.8, 2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	V
10.91	10.33	32	1e-6, 1e-7	70/15/15	2x3.8, 2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	V
9.15	10.10	32	1e-5, 1e-6	60/20/20	2x3.8, 2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	V
11.37	9.47	32	1e-5, 1e-6	70/20/10	2x3.8, 2x3.16, 2x3.32, 2x3.64, 2x3.128, 2x3.192, 2x3.256 FC4	V

The architecture that has been chosen is then the eight one, with a test RMSE of 10.89 and a validation RMSE of 9.49. Changing the learning rate or the dataset splitting percentages slightly impacts the network performance, and even though the second to last architecture shows marginally better results, a bigger training set percentage has been prioritized, in order to retain as many images during training as possible. CNN training, performed in Matlab, is presented in Fig. 8.4. In the report the RMSE in pixels is shown along with the loss function. They follow the same trend and find their way to a minimum point.

The selected architecture is also visually shown in Fig. 8.5. The pairs of convolutional layers are shown as boxes. To summarize, the input size is of 576x324, 14 hidden convolutional layers are present and the output layer consists of four fully connected neurons. About 1.9 million parameters are learned during the training phase, which takes around 30 minutes for this specific network.



Results

Validation RMSE:	18.988
Training finished:	Reached final iteration

Training Time

Start time:	22-Jun-2020 11:10:07
Elapsed time:	21 min 17 sec

Training Cycle

Epoch:	25 of 25
Iteration:	5250 of 5250
Iterations per epoch:	210
Maximum iterations:	5250

Validation

Frequency:	841 iterations

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Piecewise
Learning rate:	1e-06

Fig. 8.4: CNN training report

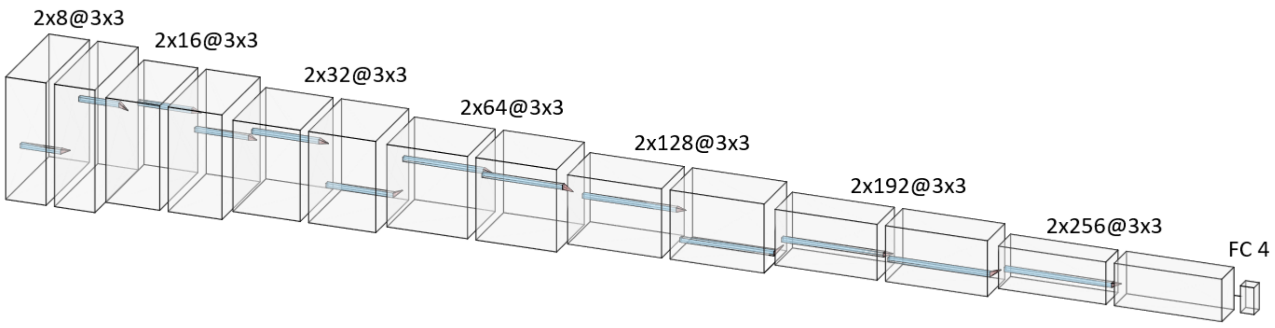


Fig. 8.5: CNN final architecture

8.3 CNN Analysis and results for design scenario

The previously described CNN performance parameter has been computed by evaluating the RMSE of the validation and testing data. It is defined as

$$RMSE = \sqrt{mean(SQE)} \tag{8.1}$$

Where the SQE is the SQuared validation/testing Error vector of all four outputs concatenated. The RMSE value in pixels, as presented before, is 9.49 for the validation error and 10.89 for the testing one. The prediction time is very fast, achieving state of the art performance in the order of units of milliseconds which is suitable to operate in real time applications. In Fig. 8.6 a set of 100 predictions are plotted against their respective execution times while in Fig. 8.7 some examples of the prediction on synthetic images are shown.

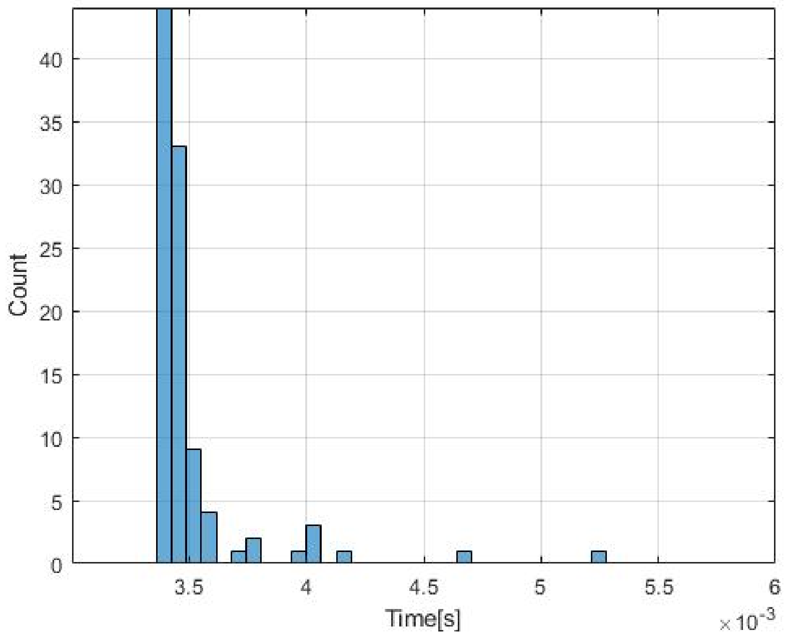
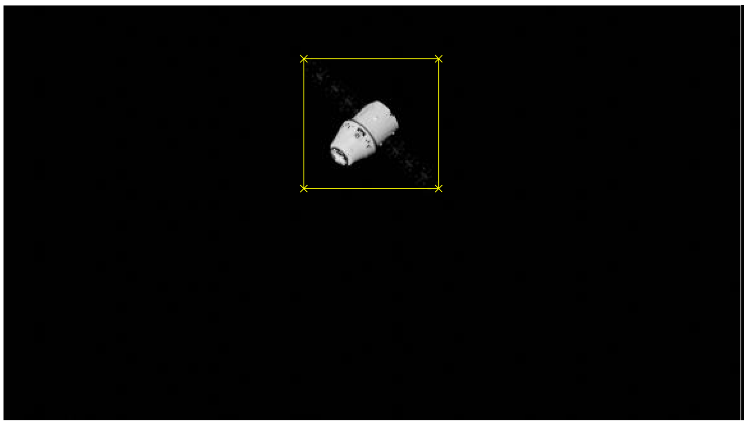
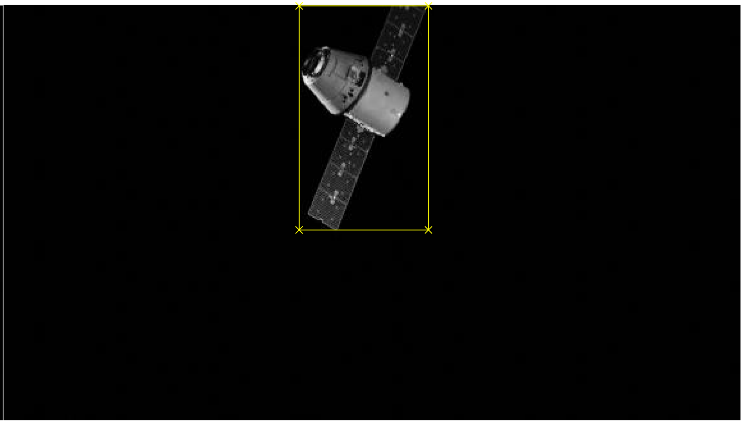
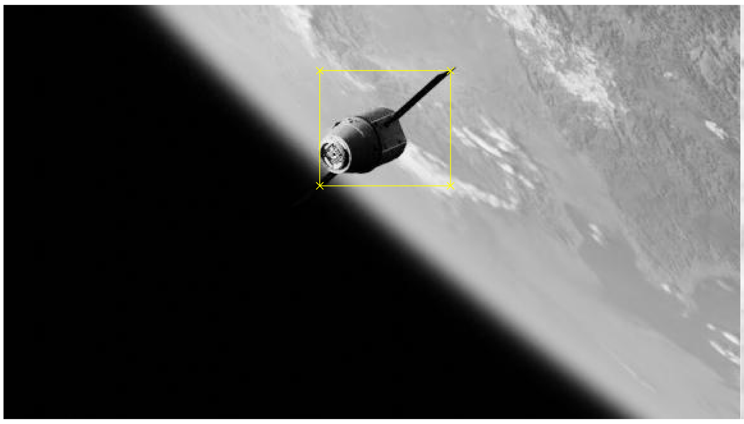
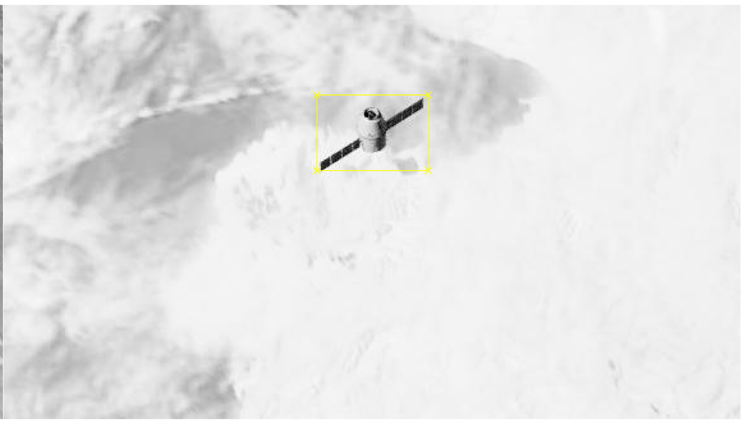


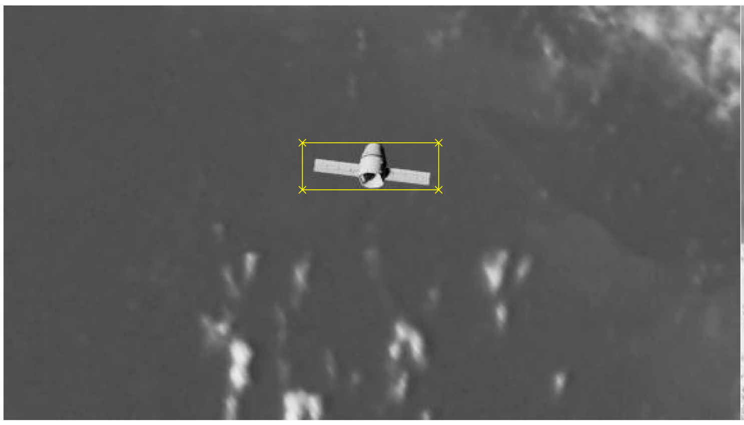
Fig. 8.6: CNN execution times


(a) Example 1


(b) Example 2


(c) Example 3


(d) Example 4


(e) Example 5

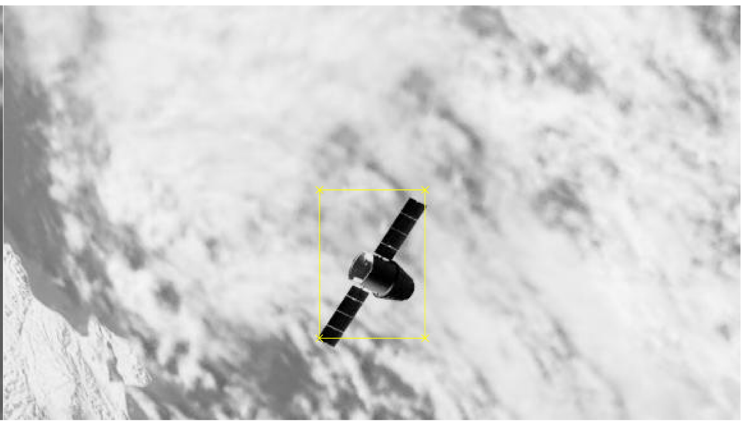

(f) Example 6

Fig. 8.7: CNN Results examples

A drawback of CCNs is that they are often referred to as “black boxes” since they can hardly be fully understood after training. This has definitely to do with the fact that there are millions of parameters that work together to filter learned patterns, however with layer by layer analysis some insights on the network can be obtained. For example, early layers in a network often recognize simple shapes like edges. The first layer of filters in the proposed CNN is strongly activated when black pixels are present (Fig. 8.8). From the author’s point of view this happens mainly to recognize if the Earth is present in the background or not. The second layer builds on the first one and some of its channels start to recognize edges, as in Fig. 8.9, while the third layer starts looking for contours too as in Fig. 8.10. Jumping to the ninth layer, the network is already able to isolate the artificial object from the Earth background as in Fig. 8.11 and Fig. 8.12. It is however true that it is difficult to properly interpret the deeper layers of the network. In Fig. 8.13 some of the features learned by the seventh convolutional layer are shown. They consist of complex wavy patterns that might be used to detect geographical and terrain features or seas and clouds patterns, however a very deep analysis would be needed to understand their precise role in the estimation process. Thanks to this intrinsic hierarchical approach in pattern recognition, an ANN learns to recognize the specific object only in the last layers and can still recognize learned patterns of a similar objects in the scene. As an example, prediction of real images on a Soyuz spacecraft and various Cubesats, spacecraft never seen before by the CNN, are shown in Fig. 8.14 and Fig. 8.15. Results are satisfactory, demonstrating the interesting capabilities that a CNN possesses and which are further explored in the next section.

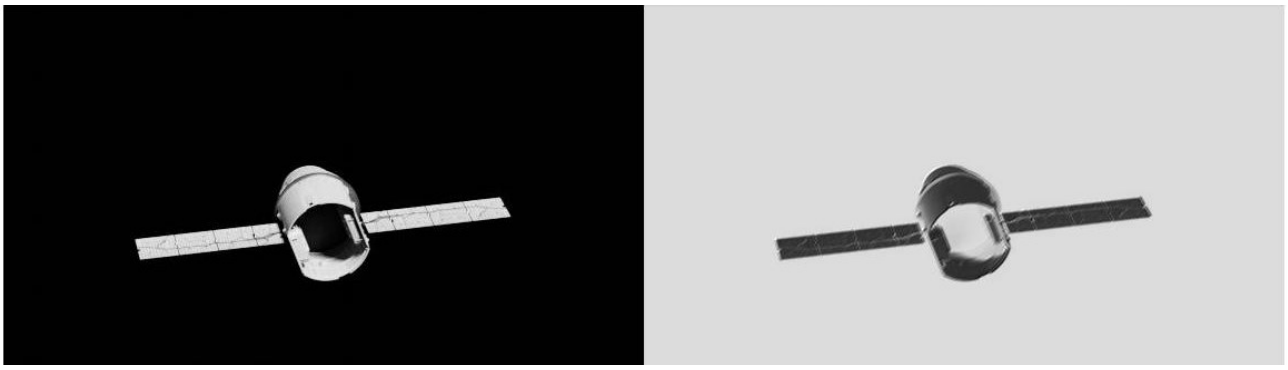


Fig. 8.8: Example of filtering after first convolutional layer

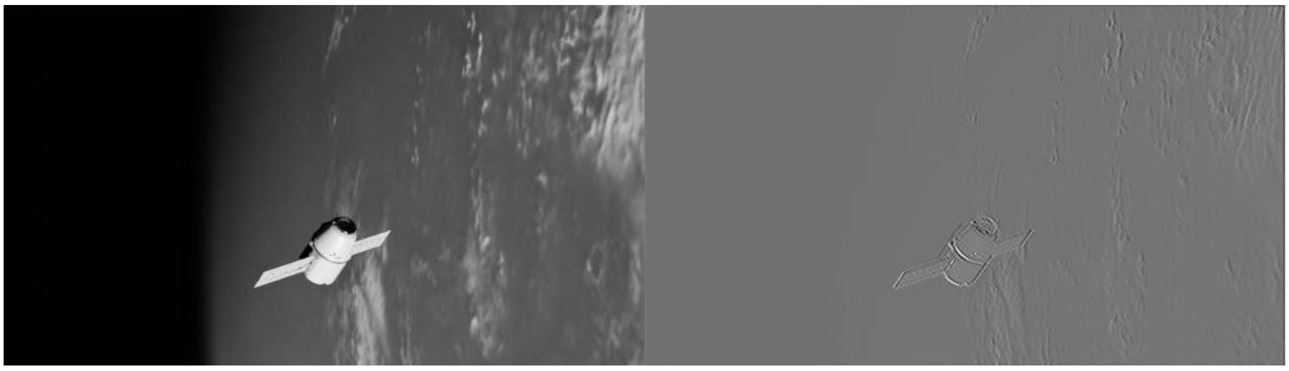


Fig. 8.9: Example of filtering after second convolutional layer

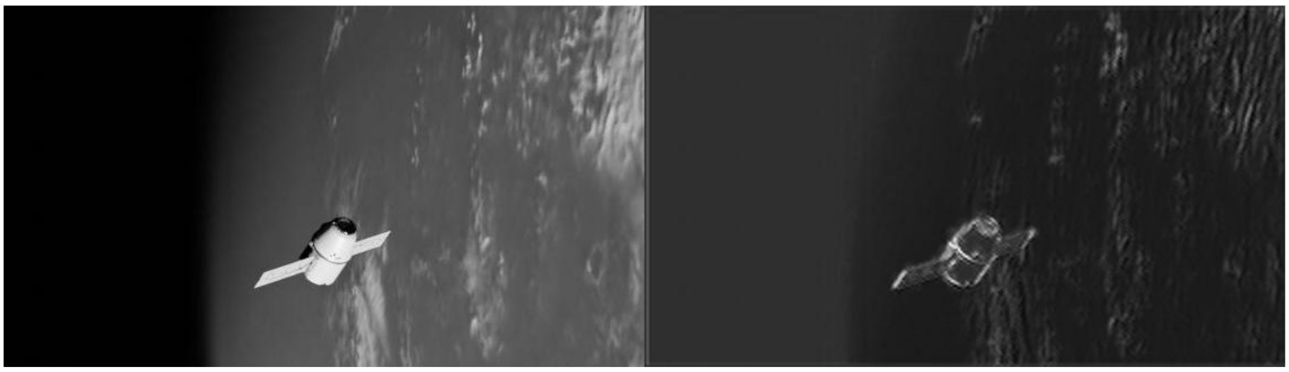


Fig. 8.10: Example of filtering after third convolutional layer

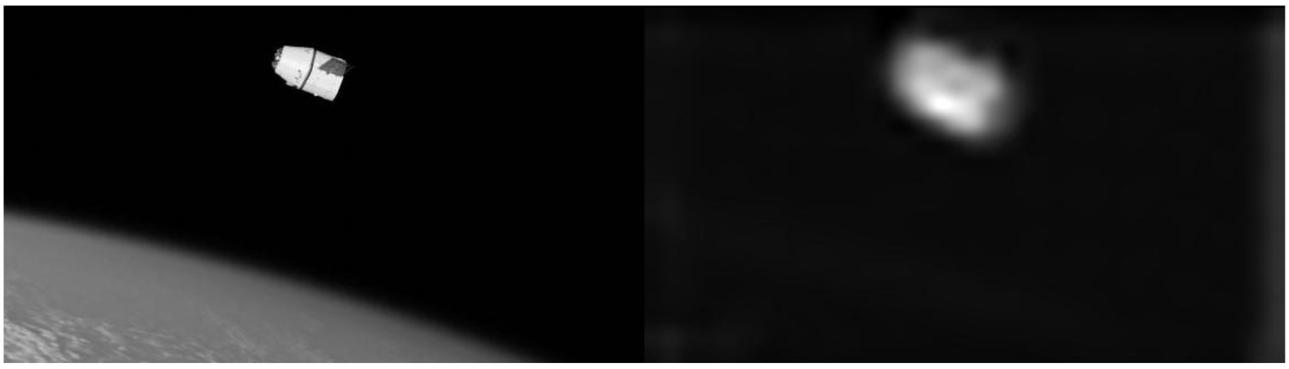


Fig. 8.11: Example of filtering after ninth convolutional layer

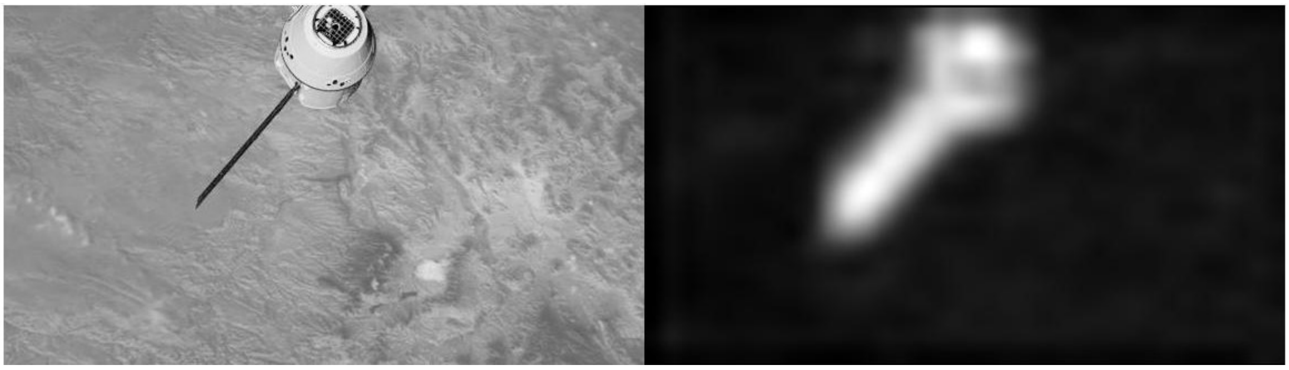


Fig. 8.12: Example of filtering after ninth convolutional layer

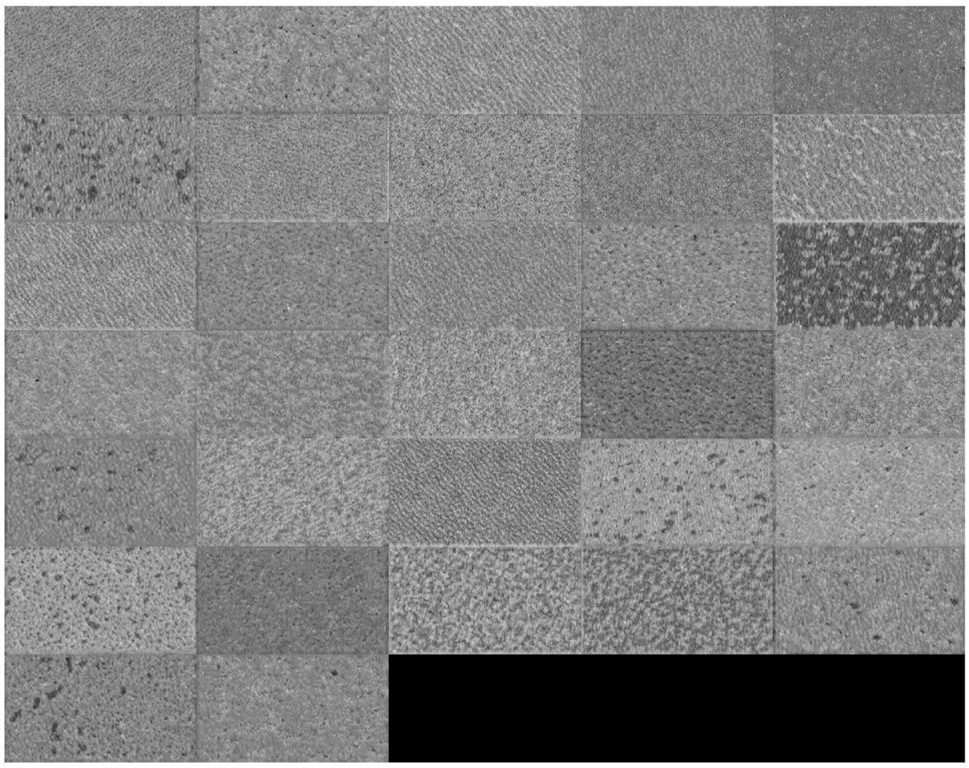
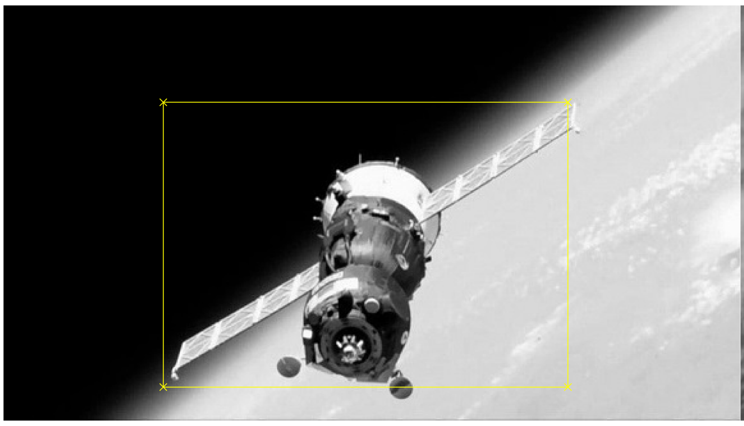
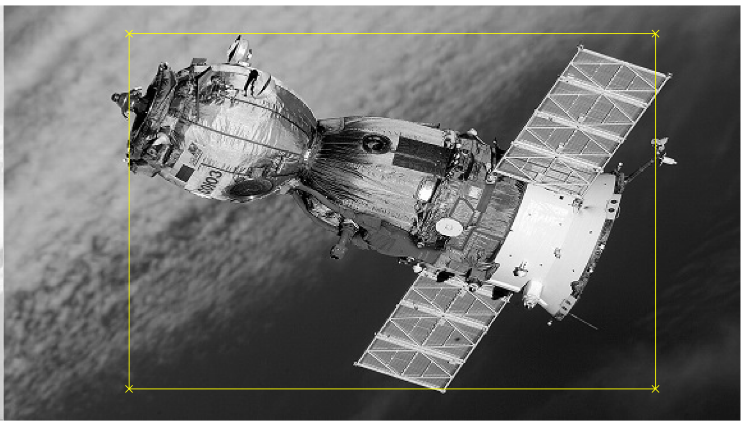


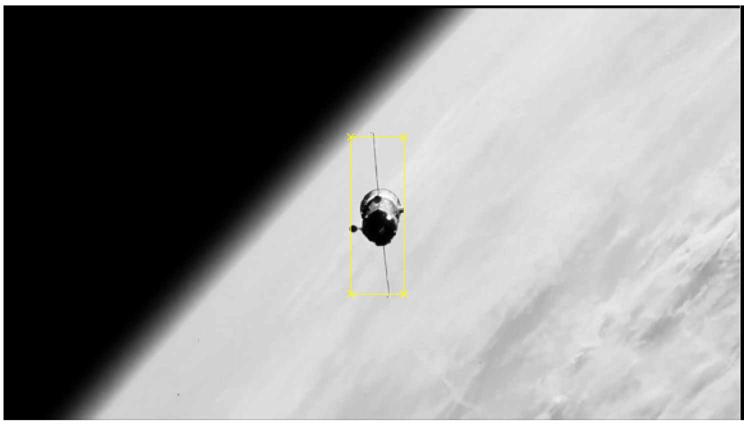
Fig. 8.13: Example of learned patters in the seventh convolutional layer



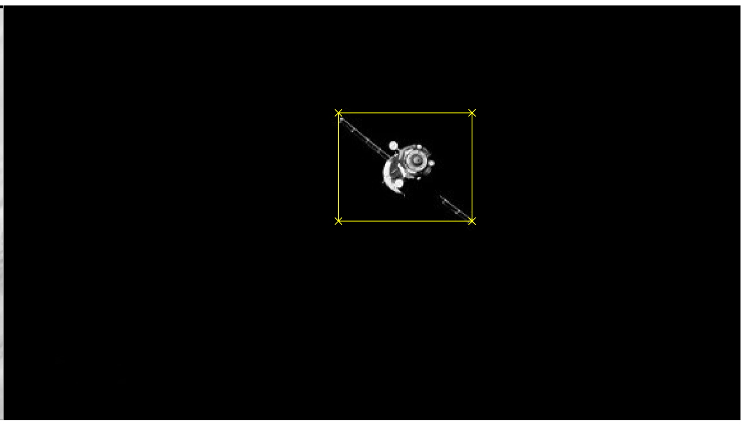
(a) Example 1



(b) Example 2

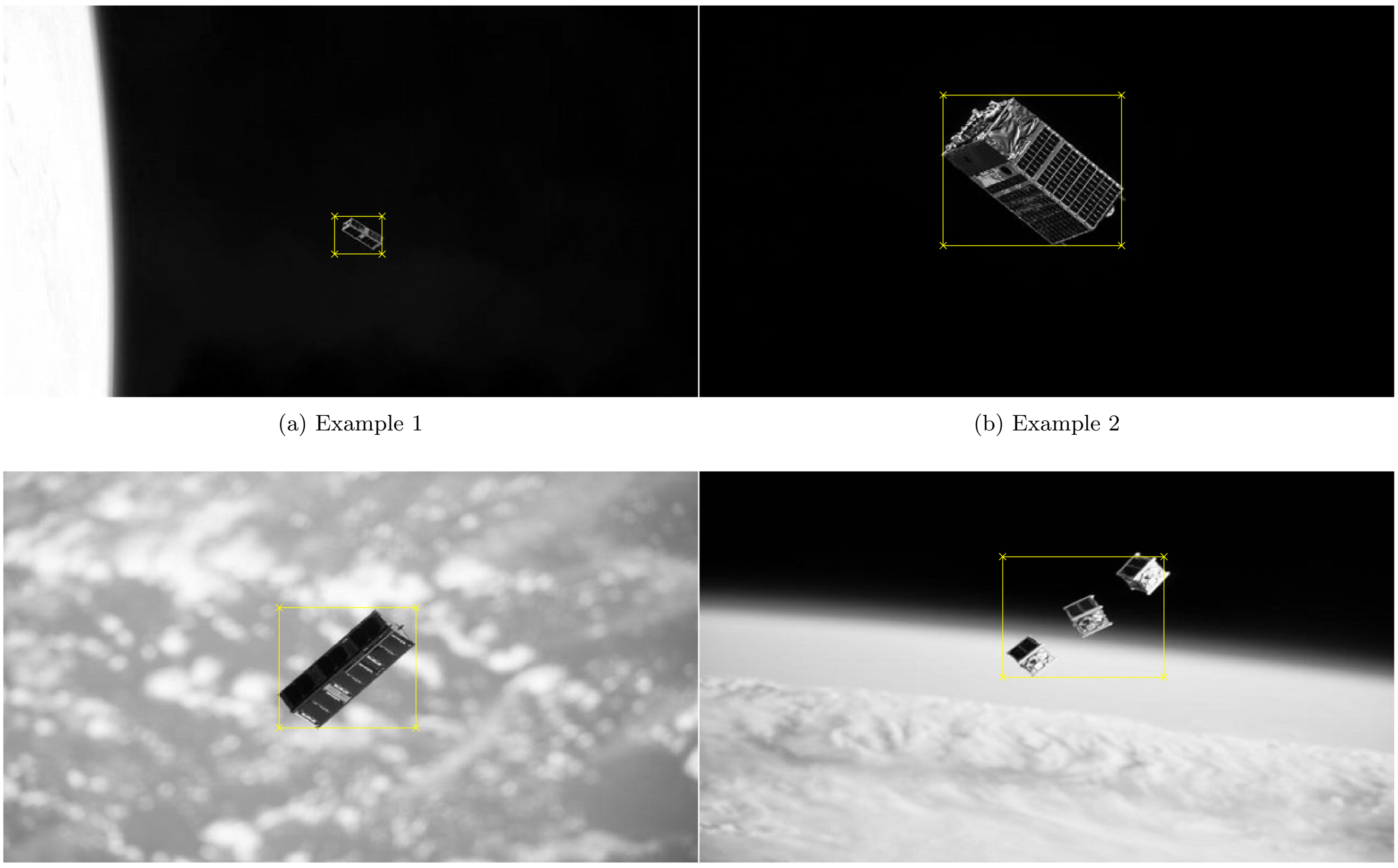


(c) Example 3



(d) Example 4

Fig. 8.14: CNN Results examples, previously unseen spacecraft [11]


(a) Example 1

(b) Example 2

(c) Example 3

(d) Example 4

Fig. 8.15: CNN Results examples, previously unseen cubesats [11]

8.4 CNN Testing

The CNN has been tested in various off-design scenarios in order to better understand its behaviour. These include tests with spacecrafts different than the design one such as MarCO, Vespa, Envisat etc and different natural backgrounds such as the Moon and comet 67P/Churyumov-Gerasimenko. Noise and blur analysis have been performed next. A broad overview of all the scenarios is available in Appendix E.

8.4.1 Off-nominal scenarios

Various off-nominal scenarios have been studied in order to understand the CNN behaviour in these types of situations. The target spacecraft has been changed to other satellites while still keeping the Earth in background. These are MarCO [86], a cubesat from NASA which joined the InSight mission to Mars, Vespa [87], a payload adapter of Vega launcher from ESA, and Envisat [88], the largest Earth observation satellite ever flown. Dragon spacecraft previously used is also reintroduced for comparison purposes, both in its original state and with a broken solar array. Moreover Dragon has been studied with Rosetta comet in the background, while Dragon and MarCO have been pictured with the Moon too.

Apart from Vespa, which 3D model has been developed from existing pictures of the adapter as in Appendix D, and Envisat, which model has been provided by the ASTRA research team at Department of Aerospace Sciences and Technology of Politecnico di Milano, all the other 3D models have been outsourced online. They can be found, with their original authors, in [77] [78] [79] [80] [81]. A list of the tests performed is shown in Table 5. Each test contains approximately 250 valid images and, when needed, the scene has been differentiated in “close” and “far”. That is to differentiate images in which the celestial body mostly occupies all the camera frame (close) and the ones in which the body is fully visible (far) and black space is present too. Images with the Earth in background have been developed with the same method used for CNN training, so that roughly 25% of the valid images contain the Earth in the background while the other ones are in black space. For what concerns the Moon and Rosetta comet, the celestial bodies have been placed in the line of sight of the chief vs deputy with some random variability in their exact position.

The test images have been fed to the network and the obtained results are shown in the following tables. In Table 6 the root mean squared error and the mean absolute error are presented, along with a score value. This last quantity has been arbitrarily defined as the amount of estimations that produce a maximum difference of the bounding box sides of at most 5% of the image height, approximately 16 pixels. Results show that the first two Dragon tests are mostly identical due to how the

Table 5: List of test datasets

OBJECT	SCENE
Dragon	Earth & Space
Dragon w/o SA	Earth & Space
Vespa	Earth & Space
Modified MarCO	Earth & Space
Envisat	Earth & Space
Modified MarCO	Moon close
Dragon	Moon, close
Dragon	Moon, far
Dragon	Comet, far
Dragon	Comet, close

neural network has been trained. Vespa and MarCO, two satellites that the network has never seen, are still identified with accuracy in more than 80% of the cases. Envisat falls lower than these last two satellites as its statistical parameters are worse. Indeed one problem that affects Envisat ROI estimation is that its solar panel is far from the main bus and sometimes it is not recognized correctly, especially when the back plate of the solar array is imaged, which can blend with the background, as shown in Fig. 8.17f. Some pictures of these test are presented below in Fig. 8.17 while a broader overview of all the test sets can be found in Appendix E.

Table 6: Statistic data for tests

OBJECT	SCENE	RMSE [px]	MAE [px]	SCORE [%]
Dragon	Earth & Space	7.42	4.50	90.5
Dragon w/o SA	Earth & Space	7.60	4.69	91.0
Vespa	Earth & Space	11.16	7.38	82.2
Modified MarCO	Earth & Space	11.23	6.99	81.4
Envisat	Earth & Space	14.48	7.91	78.0
Modified MarCO	Moon close	15.70	8.47	74.7
Dragon	Moon, close	14.84	8.89	70.6
Dragon	Moon, far	81.20	15.84	30.6
Dragon	Comet, far	91.91	71.15	< 0.1
Dragon	Comet, close	104.75	74.02	0.2

For what concerns the Moon and comet images, the CNN starts to struggle as it is shown different patterns and geographical features with respect to the training cases. It is still able to perform sufficiently in the close images of the Moon, both with Dragon and MarCO satellites. One interesting and unexpected result is that, even though the CNN has been trained with the Dragon spacecraft and the RMSE is lower in this case, MarCO test set outperforms the Dragon one in terms of score and MAE. One possible explanation to this behaviour is that Dragon can blend in more easily with the Moon background due to its white color. Moreover the backside of the solar panel, as happens for Envisat, is very difficult to spot in these kind of images, as in Fig. 8.18f, and results in worsening the average performance of this test scenario. Both cases however suffer from the bounding box "being pulled" towards regions with very high color gradient as in Fig. 8.18b, such as the terminator region or the passage from the bright moon to black space. During training with Earth scenarios, terminator regions are present, however the gradient from light to darkness is softer with respect to the sharp one of the Moon thanks to the atmosphere presence. This is especially true for the last three tests, in which the score is very low and shadows casted from the comet morphological features are not filtered correctly. Moreover the natural body is not recognized at all when fully imaged, as in Fig. 8.18d, as this condition never occurred during the training steps.

In Table 7 a breakdown of the RMSE is provided showing how the detection of the top left corner is performed more easily with respect to the height and width values in every set. This weakness of the bounding box definition is an hint to explore new types of parametrization and further improve it in future works. In the last two columns the RMSE for picture with black space or the Earth in background is shown. As expected the RMSE for images with the black space are easier to deal with by the CNN and thus their RMSE is lower with respect to the ones with the Earth. However, when using different spacecraft with respect to Dragon, the values start to converge. A part from Vespa, which has a very simple conical shape, the other spacecraft shown no difference in the space related and Earth related RMSE.

Table 7: Insights on RMSE variations

OBJECT	SCENE	RMSE x [px]	RMSE y [px]	RMSE h [px]	RMSE w [px]	RMSE S [px]	RMSE E [px]
Dragon	Earth & Space	4.97	5.87	8.75	9.21	6.16	10.42
Dragon w/o SA	Earth & Space	5.86	6.16	7.49	10.10	6.07	10.29
Vespa	Earth & Space	6.12	9.39	12.53	14.68	9.55	12.48
Modified MarCO	Earth & Space	6.17	10.46	13.94	12.77	11.94	10.47
Envisat	Earth & Space	12.75	10.10	15.59	18.22	14.32	14.95
Modified MarCO	Moon close	11.35	8.39	19.38	20.28	-	-
Dragon	Moon, close	10.18	9.59	18.88	18.16	-	-
Dragon	Moon, far	79.62	26.45	69.31	120.54	-	-
Dragon	Comet, far	82.17	40.05	104.68	120.29	-	-
Dragon	Comet, close	74.39	49.02	111.42	148.88	-	-

Finally in Table 8 some parameters regarding a normal distribution hypothesis are shown. The mean of the error is in most cases very low, and when this happens the RMSE and standard deviation coincide. The error data seems to be distributed normally, however fitting the data gives rise to a poor result. As it is possible to see in the last column of the table, at least 80% of the error values are contained inside the standard deviation region, so that only the remaining 20% error data are left outside of that region, which is not consistent with a Gaussian distribution. An example of Gaussian fitting in the case of Dragon with the Earth in background is shown in Fig. 8.16.

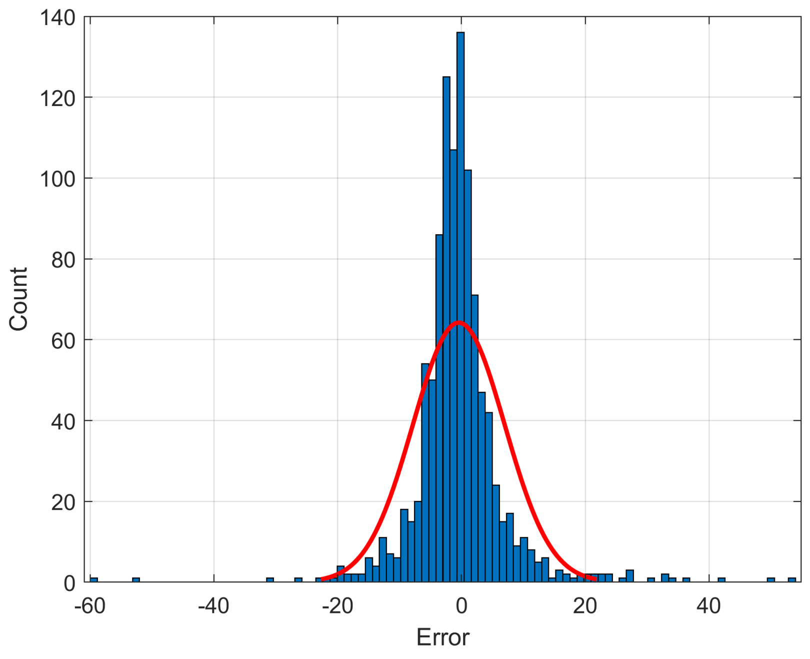
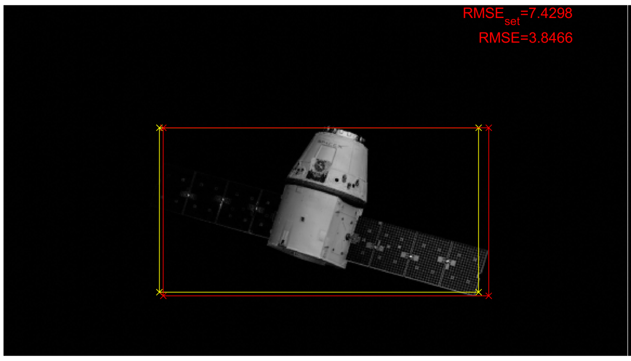


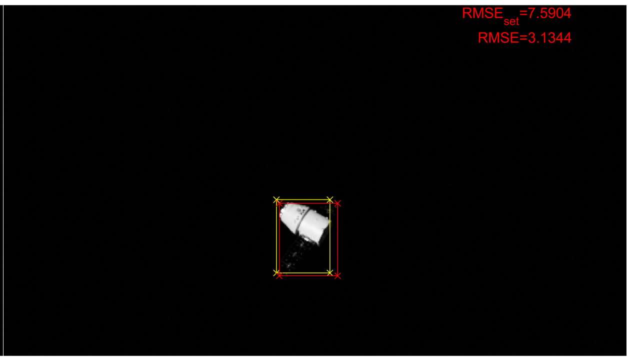
Fig. 8.16: Dragon error data and normal distribution fitting

Table 8: Parameters of the normal distributions for the tests

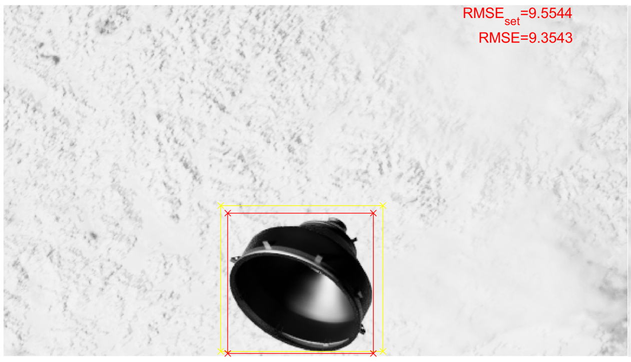
OBJECT	SCENE	Error STD [px]	Error mean [px]	Error elements < RMSE [%]	Error elements < STD [%]
Dragon	Earth & Space	7.42	-0.44	83.87	83.87
Dragon w/o SA	Earth & Space	7.57	-0.52	83.43	83.43
Vespa	Earth & Space	11.13	0.85	75.00	80.25
Modified MarCO	Earth & Space	11.21	-0.77	80.99	80.99
Envisat	Earth & Space	14.49	0.19	87.14	87.14
Modified MarCO	Moon close	15.29	3.61	86.83	86.53
Dragon	Moon, close	14.79	1.34	84.63	84.51
Dragon	Moon, far	79.77	50.03	74.66	74.66
Dragon	Comet, far	88.76	24.31	62.64	61.49
Dragon	Comet, close	100.41	30.17	69.72	67.96



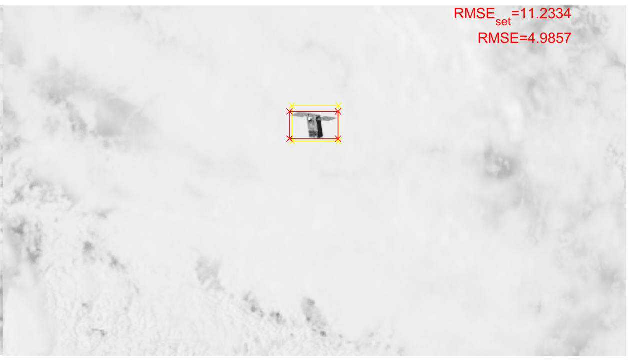
(a) Dragon



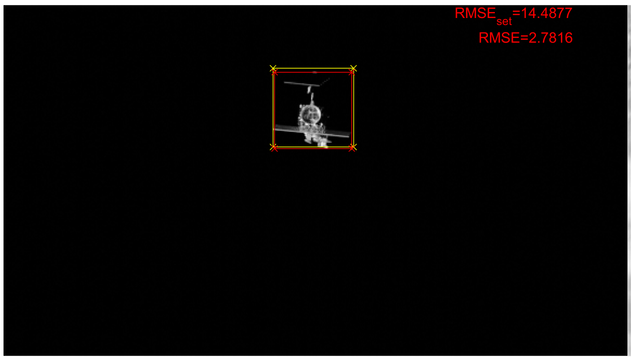
(b) Dragon w/o solar array



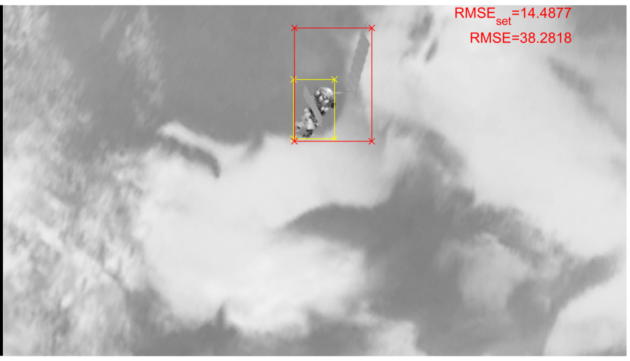
(c) Vespa



(d) MarCO

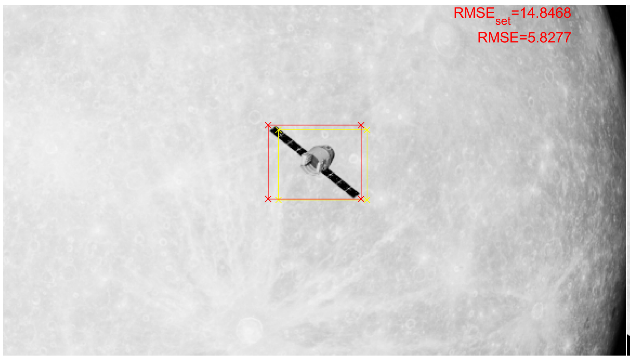


(e) Envisat

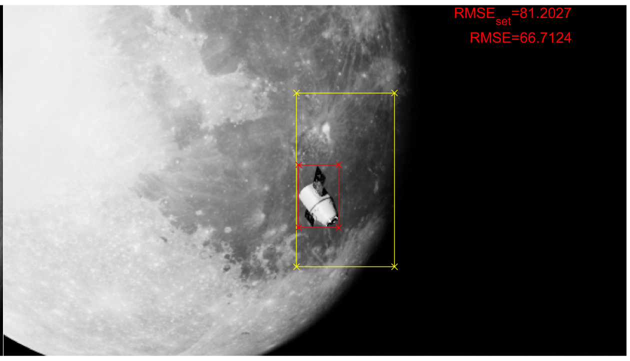


(f) Envisat, example of challenging recognition

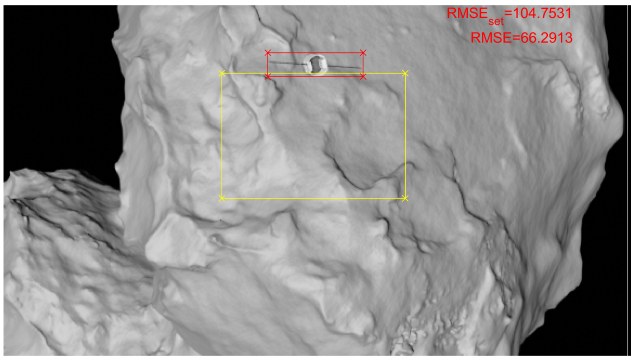
Fig. 8.17: Example of random test images, Earth cases, estimation in yellow



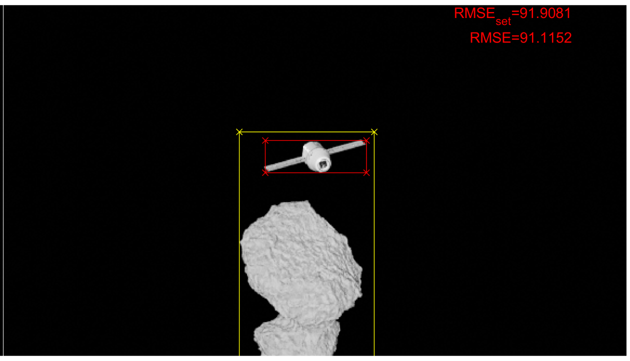
(a) Dragon, Moon, close



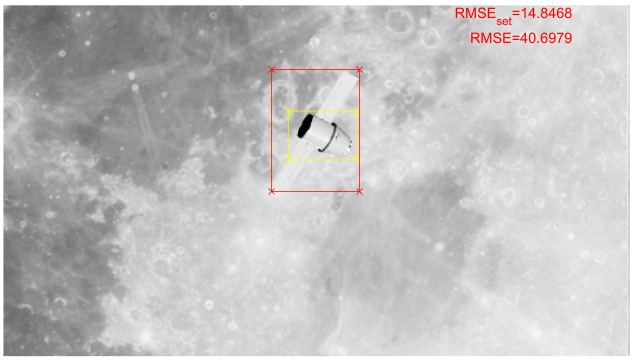
(b) Dragon, Moon, far



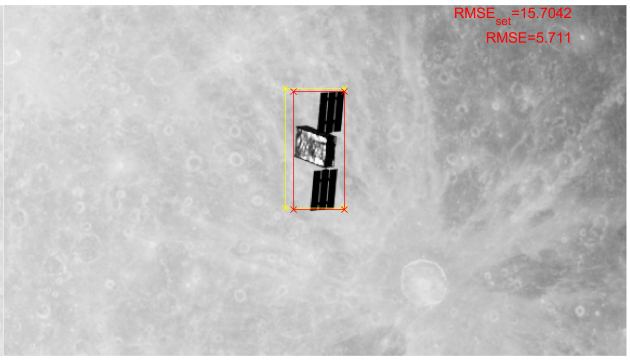
(c) Dragon, Comet, close



(d) Dragon, Comet, far



(e) Dragon, Moon, example of challenging recognition



(f) MarCO, Moon, close

Fig. 8.18: Example of random test images, Moon and Rosetta comet, estimation in yellow

8.4.2 Noise and Blur

Two other tests have been performed in order to asses the capability of the network to deal with noise and blur in the images, as those are two common causes of image degradation, for example due to thermal noise or camera vibrations. Noise is obtained through a Gaussian noise model while blur is performed through Gaussian filtering. The design scenario has been reused from previous test sets, specifically with Dragon and the Earth/space in background. For what concerns noise, its effects can become more and more detrimental on the CNN behaviour as the noise variance is increased. Intrinsic noise from the rending process is always present, however as the noise in the image increases, the network performance are slowly but gradually reduced, as can be noted in Table 9. When noisy images are expected, the training process should be performed again to ensure proper CNN behaviour, or at least some denoising algorithm should be used to preprocess images. The different noise levels are shown in Fig. 8.19 and full size images can be found in Appendix E.11.

Table 9: Noise tests

OBJECT	SCENE	Var noise	RMSE [px]
Dragon	Earth & Space	0.5e-6	7.43
Dragon	Earth & Space	1.0e-6	7.47
Dragon	Earth & Space	0.5e-5	8.12
Dragon	Earth & Space	1.0e-5	9.04
Dragon	Earth & Space	0.5e-4	14.79
Dragon	Earth & Space	1.0e-4	21.68
Dragon	Earth & Space	0.5e-3	120.95
Dragon	Earth & Space	1.0e-3	156.78
Dragon	Earth & Space	1.0e-2	171.66

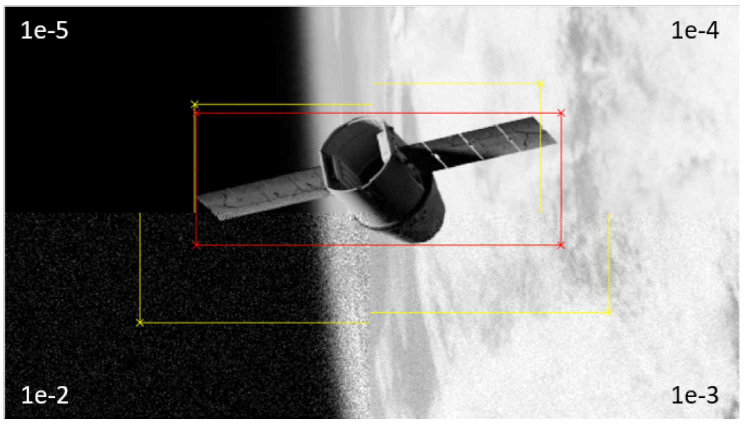


Fig. 8.19: Noise tests, Dragon with Earth in background

Finally, for what concerns blurring, small details and patterns can be lost during this process, which leads to worsening of the CNN performance. An interesting result is that small intensity blurring can be beneficial to the CNN estimations, as can be noted for small values of σ in Table 10. This low blurring effect has been interpreted as a denoising effect on the image which guarantees better estimations. However, as happens for the noise test, the higher the blurring intensity, the higher the RMSE and the higher the uncertainties in the estimations. Again a summary figure of the blurring tests is presented below in Fig. 8.20 while full size images are present in Appendix E.12.

Table 10: Blur tests

OBJECT	SCENE	σ blur	RMSE [px]
Dragon	Earth & Space	0.1	7.43
Dragon	Earth & Space	0.2	7.43
Dragon	Earth & Space	0.3	7.42
Dragon	Earth & Space	0.4	7.07
Dragon	Earth & Space	0.5	7.16
Dragon	Earth & Space	0.6	9.22
Dragon	Earth & Space	0.7	11.82
Dragon	Earth & Space	0.8	14.69
Dragon	Earth & Space	0.9	18.60
Dragon	Earth & Space	1.0	22.35
Dragon	Earth & Space	1.5	42.36
Dragon	Earth & Space	3.0	71.09

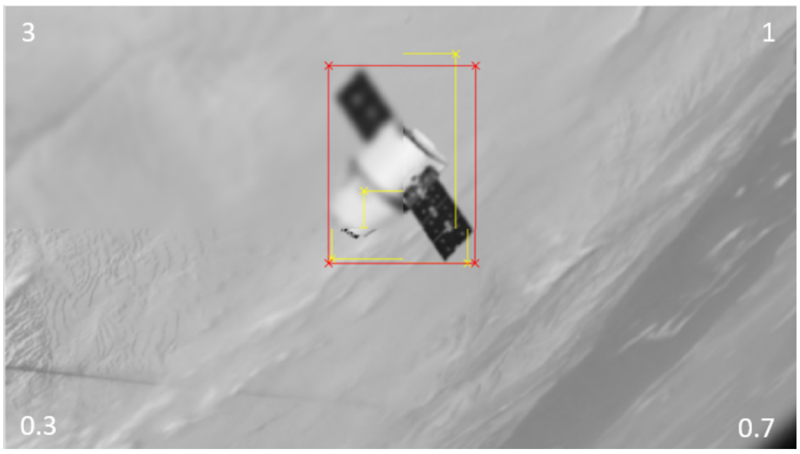
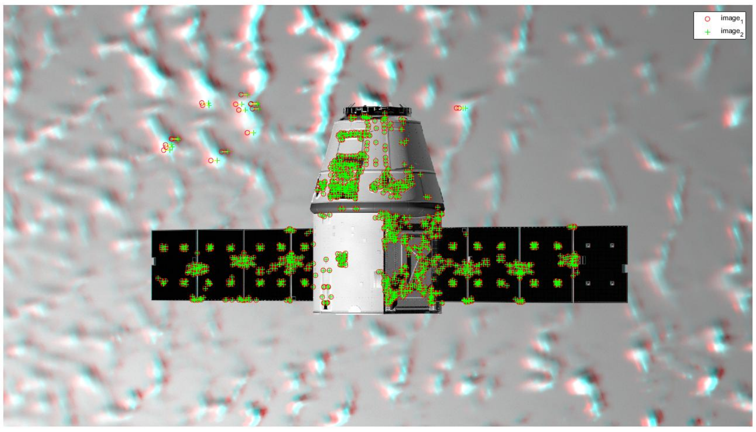


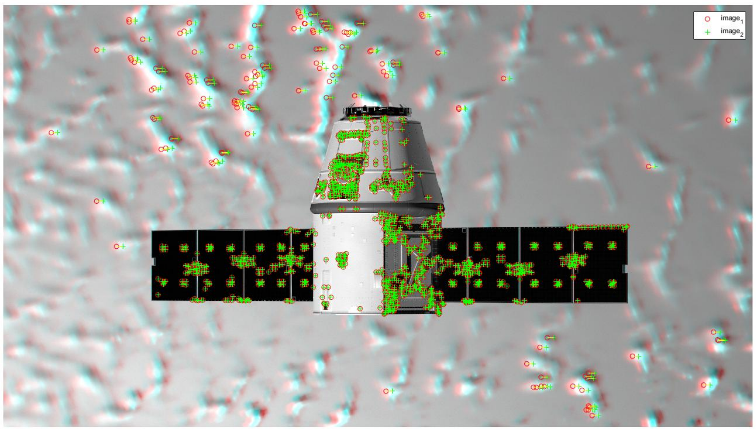
Fig. 8.20: Blur tests, Dragon with Earth in background

8.5 V-SLAM with CNN ROI results

In this last section a comparison between two identical V-SLAM test cases is provided, in which only one of them benefits of the CNN ROI extraction method. Features are thus not detected on the whole image but only in the estimated bounding box. The trajectory that has been used is the same as in the Test case 1 for the V-SLAM algorithm reported in Fig. 7.7a, while the background has been chosen to be an Earth one with visible and intense cloud patterns. This constitutes a challenging scenario as numerous false features are incorrectly detected an triangulated in the V-SLAM process as in Fig. 8.21.



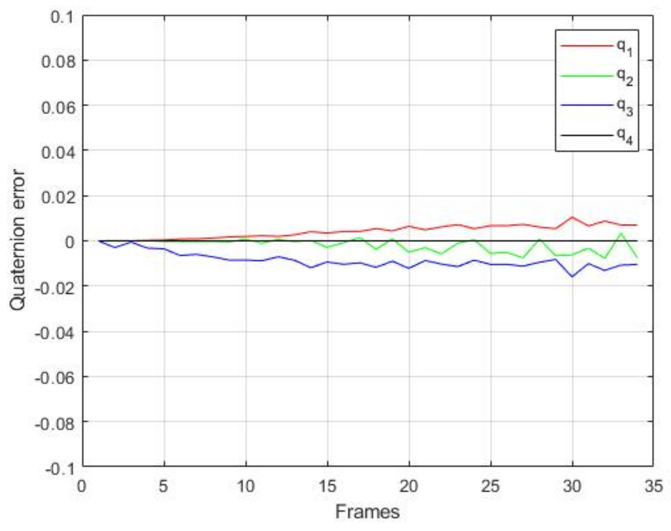
(a) Frame example, with CNN ROI estimation



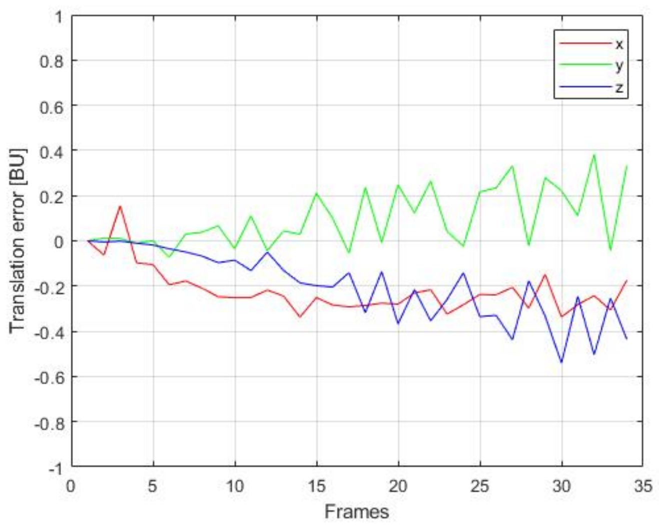
(b) Frame example, without CNN ROI estimation

Fig. 8.21: Frame examples, with and without CNN ROI estimation

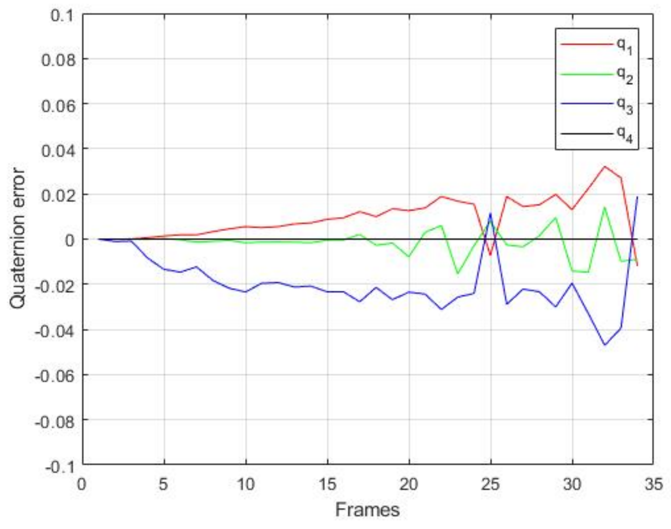
In particular in Fig. 8.21a it emerges how most of the false features are eliminated by the CNN bounding box estimation. This effect is also visible in the translation and quaternion errors as in Fig. 8.22. By comparing Fig. 8.22a and Fig. 8.22b with Fig. 8.22c and Fig. 8.22d the test case without the CNN is consistently worse in error quality with respect to the one with the CNN enabled. Due to the bounding box rectangular shape however some features are still present in the top part of the spacecraft even with CNN ROI estimation, as in Fig. 8.21a, which contributes to worsen the trajectory estimation of this case. In the future an image-to-image CNN could be developed in order to tackle this problem and align the error accuracy of Fig. 8.22a and Fig. 8.22b to the one obtained in Test case 1 in Section 7.3.1.



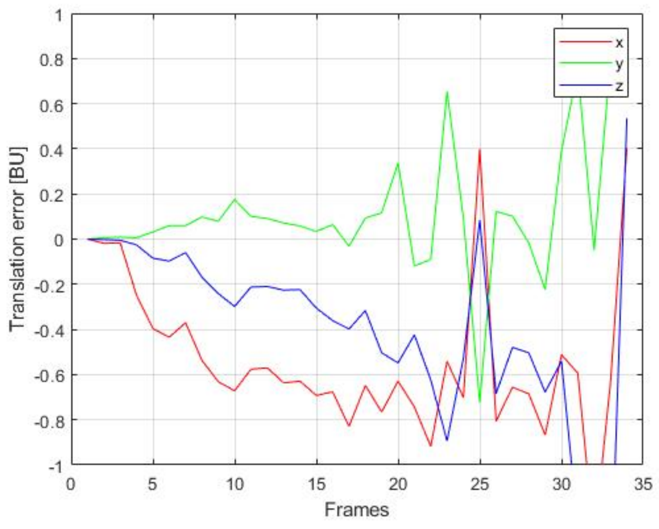
(a) Quaternion error with CNN ROI



(b) Translation error with CNN ROI



(c) Quaternion error without CNN ROI



(d) Translation error without CNN ROI

Fig. 8.22: Results for V-SLAM algorithm with and without CNN ROI

9 Conclusion

The present thesis has faced the problem of V-SLAM for space operations with machine learning applications. The most general situation has been assumed, with an inspector satellite investigating an unknown and uncooperative target, it being another satellite or a natural body. A simple and cheap sensor, a mono-camera, has been assumed as the only available instrument. This choice could be especially interesting for small satellites missions while its drawbacks, such as the saturation of the imaging sensor due to the Sun or eclipse periods, must be dealt with during mission design.

An image generation software has been introduced in order to produce high quality space images and orbital pose data. Images are used to study the implementation of a monocular V-SLAM method for relative navigation and to better understand its benefits and limitations. Simulations have been performed by varying the inspector and target relative trajectory with satisfactory results. Moreover the presence of unwanted objects in the camera frame has been considered, problem that has to be avoided as it can interfere heavily with the image capturing process. Detection of false features is a possibility when imaging Earth geographical patterns, but tracks must not be propagated throughout the V-SLAM process. Thanks to the recent developments in machine learning tools, the possible contributions of a CNN has been assessed. By training it on recognizing the bounding box of a particular satellite, it shows good performances also when real images of off-design satellites are given as input. Tests have been performed to understand the neural network behaviour with different scenarios, changing both the target spacecraft and the natural body in the background. The CNN behaves properly when the target satellite is swapped for another probe, however changing the planetary background to an asteroid or comet one gives rise to negative results.

AI and machine learning methods have been quickly developing in these few last decades, revolutionizing many engineering sectors, from robotics to computer vision. Moreover these technologies have shown great performance, excellent computational time and robustness to different scenarios and noise. As such these novel methods will definitely be able to impact the space domain too and revolutionize many current applications.

9.1 Future Work

Future development can be carried out to improve the performances of both the V-SLAM and the CNN applications.

9.1.1 V-SLAM method

Various modules can be added to the V-SLAM like algorithm presented in this thesis, such as loop detection and failure recovery. Loop closure is generally performed via a bag-of-words approach which compares current images to a pretrained visual dictionary. Extensive study could be performed on this topic, to understand whether this approach can be still valid for space applications. More insights on bundle adjustment could be seeked too, whether a windowed approach or a key-frame one is used. These two concepts would also enable the study of long trajectories with multiple intersections, in which the error would otherwise drift indefinitely.

The V-SLAM algorithm has been written from ground up, using existing functions only for computer vision algorithms. One of the most challenging problem in terms of code structure was the data structure for track management. A proper data structure is needed to save the tracks evolution in frames and provide them for tracking, triangulation, bundle adjustment, new feature tracking etc. A study to improve the data structure could be performed, or pre-existing track management code could instead be used, for example as in Matlab "viewSet" object.

Finally, an extensive testing with the tools denoted in this thesis could be performed on the V-SLAM algorithm. Various tests could be done by changing parameters, for example the focal length, noise, image resolution and illumination conditions with different target objects. It could be very interesting, and challenging as well, to perform a Monte Carlo simulation by varying these parameters and generating a huge amount of images and trajectory data to be analyzed. A powerful computer would probably be needed to generate and process this kind of data. In those simulations the feature detection process might be problematic when lots of features are lost and few features are detected. Artificial objects might have less texture with respect to natural ones and some form of failure recovery could be studied.

9.1.2 CNN ROI estimation

For what concerns the CNN design, a larger dataset would definitely be beneficial to the training process of the network. A starting point could be to enlarge by 5x the number of images in order to have the same dataset dimension as the ESA Pose Estimation challenge one. The dataset could also be enlarged with images containing different models for the target satellite in order to understand if any improvements can be obtained in this way.

It could also be interesting to change some parameters of the dataset to understand the effects on the network. For example the bounding box parametrization could be further explored as well as the criteria with which dark pixels belonging to the target satellite are discarded. Another approach that could result in even better estimation could be the use of image to image regression, where the CNN uses convolution and deconvolution operations to output an image with only the estimated satellite pixels highlighted, removing completely the need for the bounding box.

Cloud computing could also be explored, with services like Amazon AWS, Google Cloud or Microsoft Azure. These solutions would allow to use more powerful GPUs to perform faster training. High resolution images could also be exploited, that will otherwise overload a single desktop GPU, however these services are pay per use.

Finally testing in real scenarios with actual sensors and targets should be performed. Testing could be conducted on embedded systems or in a GNC facility, such as the one present at Politecnico di Milano with an asteriod and satellite mock-ups.

A Singular value decomposition

A matrix decomposition very useful for computer vision problems is the singular value decomposition. It can be seen as a generalization of the eigenvalue decomposition and can be applied to any square and rectangular matrix. A matrix $\boldsymbol{A}$ can be decomposed as [89]

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^H \tag{A.1}$$

where $\boldsymbol{U}$ is the n×n orthonormal left singular vectors matrix, $\boldsymbol{\Sigma}$ is the n×m diagonal singular values matrix and $\boldsymbol{V}^H$ is the hermitian transpose of the m×m orthonormal right singular vector matrix. The columns of $\boldsymbol{U}$ and $\boldsymbol{V}$, $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are the singular vectors while $\boldsymbol{\sigma}_i$ are the singular values. If the matrix $\boldsymbol{A}$ is square, symmetric and positive definite then the singular values decomposition and the eigendecomposition are the same.

The reasoning behind this decomposition is visually explained in Fig. A.1. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}^H$ represent rotations while $\boldsymbol{\Sigma}$ adds scaling in between the two transformations.

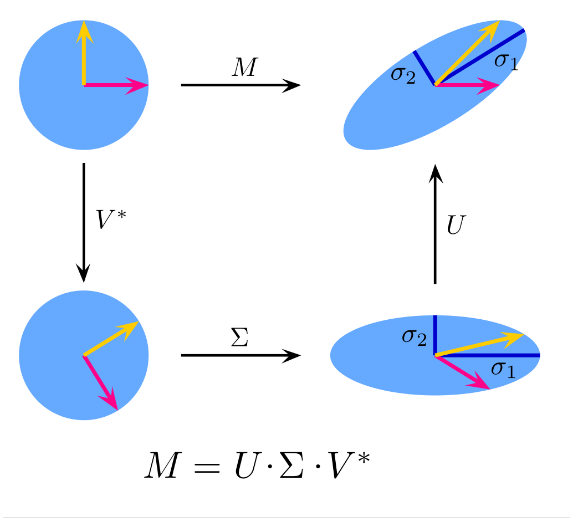


Fig. A.1: SVD illustration [12]

A.1 SVD for least squares problems

SVD is useful to solve total least squares minimization problems that often appear in computer vision. The problem can be expressed as finding the minimum of $||\boldsymbol{Ax}||^2$ with $||\boldsymbol{x}||^2 = 1$. Using the decomposition previously described and noticing that an initial rotation $\boldsymbol{U}$ or $\boldsymbol{V}^H$ does not change lengths, the formulation can be rewritten as $min||\boldsymbol{\Sigma V}^H\boldsymbol{x}||^2$ with $||\boldsymbol{V}^H\boldsymbol{x}||^2= 1$. Since $\boldsymbol{\Sigma}$ contains singular values in decreasing order, the solution becomes the singular vector of $\boldsymbol{V}$ corresponding to the lowest singular value, or $\boldsymbol{x} = \boldsymbol{V}[0, 0, ...,0,1]^T$ [90].

A.2 SVD for overdetermined systems

SVD can be used to solve overdetemined systems in the form of $\boldsymbol{Ax} = \boldsymbol{b}$ when $\boldsymbol{A}$ is a $m \times n$ matrix with $m > n$. First of all, the solution of that system can be rewritten as a $n \times n$ linear system as

$$\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b} \tag{A.2}$$

if $\boldsymbol{A}$ is a real matrix, then the hermitian transpose and the transpose coincide. Under this assumption, Eq. (A.2) becomes

$$\boldsymbol{V\Sigma}^T\boldsymbol{\Sigma V}^T\boldsymbol{x} = \boldsymbol{V\Sigma}^T\boldsymbol{U}^T\boldsymbol{b} \tag{A.3}$$

and the solution is

$$\boldsymbol{x} = \boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{b} \tag{A.4}$$

Where $(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}^T$ is the pseudoinverse of $\boldsymbol{\Sigma}$.

B Unconstrained Numerical Optimization

Optimization is the core of any machine learning algorithm. A cost function is generally defined and the objective is to minimize it. In this appendix section, most popular unconstrained optimization methods are discussed.

The general unconstrained optimization problem for a cost function $f(\boldsymbol{x})$ is to find

$$\min f(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \boldsymbol{R}^n \tag{B.1}$$

It is then necessary to find the conditions for which $\nabla f(\boldsymbol{x}) = 0$.

B.1 Newton method

The most popular method for minimizing a function is Newton method. It is used to obtain the zeros of a function but can be easily adapted to deal with minimization problems.
Given a vector valued function $g(\boldsymbol{x})$ it is possible to construct its first order approximation in the point $\boldsymbol{x}_k$ as

$$g(\boldsymbol{x}) = g(\boldsymbol{x}_k) + \boldsymbol{J}(g(\boldsymbol{x}_k))(\boldsymbol{x} - \boldsymbol{x}_k) \tag{B.2}$$

By requiring that $g(\boldsymbol{x}) = 0$, in order to obtain its zero(s), it results that

$$\boldsymbol{x} = \boldsymbol{x}_k - \boldsymbol{J}(g(\boldsymbol{x}_k))^{-1} g(\boldsymbol{x}_k) \tag{B.3}$$

and by rewriting Eq. (B.4) to be used in an iterative manner, the core equation of Newton method reads

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{J}(g(\boldsymbol{x}_k))^{-1} g(\boldsymbol{x}_k) \tag{B.4}$$

In order to apply Newton method in a minimization problem, it is sufficient to just impose the vector function to be $g(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$ so that

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}(f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x}_k) \tag{B.5}$$

In general matrices are not inverted but a linear system is solved. This method is simple, however it can quickly become computationally very expensive when the dimensions of $\boldsymbol{x}$ increase, as the gradient and the Hessian matrix must be computed at every iteration. Anyhow, there are other methods that can avoid using the information on the curvature of the function through the Hessian to reach the minimum and thus be computationally lighter. They are grouped in a family called line search methods.

B.2 Line search methods

Line search methods are iterative algorithms that aim to find the direction of steepest descent through the gradient information. They are written as [91]

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}(\boldsymbol{x}_k) \tag{B.6}$$

where $\boldsymbol{d}(\boldsymbol{x}_k)$ must be a descent direction. The directional derivative of $f(\boldsymbol{x}_k)$ along that direction must be negative and the descent direction must be zero when a stationary point is reached. So $\boldsymbol{d}(\boldsymbol{x}_k)$ must satisfy

$$\begin{aligned} &\boldsymbol{d}(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k) < 0 && \text{if } \ \nabla f(\boldsymbol{x}_k) \neq 0 \\ &\boldsymbol{d}(\boldsymbol{x}_k) = 0 && \text{if } \ \nabla f(\boldsymbol{x}_k) = 0 \end{aligned} \tag{B.7}$$

In Eq. (B.6) both α_k and $\boldsymbol{d}(\boldsymbol{x}_k)$ needs to be selected. For the descent direction, the most common ones are

– Newton directions: $\boldsymbol{d}(\boldsymbol{x}_k) = -\boldsymbol{H}(f(\boldsymbol{x}_k))^{-1}\nabla f(\boldsymbol{x}_k)$

– Quasi-Newton directions: $\boldsymbol{d}(\boldsymbol{x}_k) = -\boldsymbol{H}_k^{-1}\nabla f(\boldsymbol{x}_k)$

– Gradient directions: $\boldsymbol{d}(\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k)$

The gradient direction is the simplest one, where steps are taken just proportionally to the negative gradient, thus closing to the minimum at every iteration. Newton directions instead use also the information on the curvature to get to the minimum point through the fastest route. However, the Hessian matrix needs to be computed and the linear system solved which can be very heavy computationally. Quasi-Newton methods approximate the value of the Hessian matrix instead of using the analytic one.

For what concerns the step α_k, in principle it should be chosen such that the new point $\boldsymbol{x}_{k+1}$ of Eq. (B.6) is the minimum one $\forall \boldsymbol{x}_{k+1}(\alpha_k)$ given $\boldsymbol{d}(\boldsymbol{x}_k)$. This is a minimization problem in itself and may not be worth to solve it. Instead, α_k can be chosen such that $f(\boldsymbol{x}_{k+1})$ is arbitrary lower than $f(\boldsymbol{x}_k)$. Wolfe conditions roughly apply this principle and impose that α_k must satisfy

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) + \sigma\alpha_k \nabla_{\boldsymbol{d}} f(\boldsymbol{x}_k) \tag{B.8}$$

$$\nabla_{\boldsymbol{d}} f(\boldsymbol{x}_{k+1}) \geq \delta \nabla_{\boldsymbol{d}} f(\boldsymbol{x}_k) \tag{B.9}$$

where $\nabla_{\boldsymbol{d}} f(\boldsymbol{x}_k) = \boldsymbol{d}(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)$ is the directional derivative along $\boldsymbol{d}(\boldsymbol{x}_k)$ of $f(\boldsymbol{x}_k)$, $\nabla_{\boldsymbol{d}} f(\boldsymbol{x}_{k+1}) = \boldsymbol{d}(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_{k+1})$ is the one of $f(\boldsymbol{x}_{k+1})$ and $0 < \sigma < \delta < 1$ are parameters. The first condition ensures that the difference $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)$ is proportional to the directional derivative and is bigger when α_k increases. The second one instead ensures that the gradient along $\boldsymbol{d}$ of the iteration $k+1$ is lower in magnitude with respect to the previous iteration, thus reaching a stationary point for f. This guarantees that the steps are not too small when a function is almost stationary.

An iterative method can be adopted to get α_k. Starting from $\alpha_k = 1$, it can be gradually lowered in value until the first condition of Eq. (B.9) is met. The second condition will be automatically satisfied as the value of α_k will be the biggest allowable one.

B.3 Trust region methods

Trust region methods aim to solve the minimization problem choosing at the same time the descent direction and the step size. The search space at every step is constrained by a maximum radius δ_k which defines the trust region. At every iteration step a quadratic model with a Taylor expansion is fitted to the function in $\boldsymbol{x}_k$ as

$$f_k(\boldsymbol{s}) = f(\boldsymbol{x}_k) + \boldsymbol{s}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{s}^T \boldsymbol{H}_k \boldsymbol{s} \tag{B.10}$$

the step increment $\boldsymbol{s}_k$ is then the solution of the problem

$$\min f_k(\boldsymbol{s}) \quad \text{with } ||\boldsymbol{s}|| \leq \delta_k \tag{B.11}$$

and if the resulting step size is suitable, the iteration step becomes $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$, otherwise the trust region is modified and the problem is solved again.

If the minimum of the function $f(\boldsymbol{x})$ is inside the trust region, then $\boldsymbol{s}_k$ will be lower than δ_k and the Hessian $\boldsymbol{H}_k$ must be positive definite. In this case the problem is solved as with a quasi-Newton method, resulting in $\boldsymbol{s}_k = -\boldsymbol{H}_k^{-1}\nabla f(\boldsymbol{x}_k)$. Otherwise the constrained minimization problem of $f_k(\boldsymbol{s})$ with $||\boldsymbol{s}|| = \delta_k$ can be solved using the method of Lagrange multiplier. The Lagrangian $\boldsymbol{L}_k(\boldsymbol{s}, \lambda) = f_k(\boldsymbol{s}) - \lambda(||\boldsymbol{s}|| - \delta_k)$ is minimized by imposing that $\nabla_s \boldsymbol{L}_k(\boldsymbol{s}, \lambda) = 0$ and $\frac{\partial \boldsymbol{L}_k(\boldsymbol{s},\lambda)}{\partial \lambda} = 0$. Those expressions result in

$$(\boldsymbol{H}_k + \lambda_k \boldsymbol{I})\boldsymbol{s}_k = -\nabla f(\boldsymbol{x}_k) \quad \text{and} \quad ||\boldsymbol{s}_k|| - \delta_k = 0 \tag{B.12}$$

with $(\boldsymbol{H}_k + \lambda_k \boldsymbol{I})$ positive definite. Newton method can be used to compute the corrections on $\boldsymbol{s}_k$ and λ_k, generally through the solution of a linear system.

The criteria with which a step is considered accurate is if the actual change in f in $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ is similar to the estimated change of f_k. ρ_k is defined as

$$\rho_k = \frac{f(\boldsymbol{x}_k + \boldsymbol{s}_k) - f(\boldsymbol{x}_k)}{f_k(\boldsymbol{s}_k) - f_k(\mathbf{0})} \tag{B.13}$$

When ρ_k is close to one the approximation for f is good, otherwise the trust region needs to be shrinked.

C Nonlinear least square methods

Nonlinear least square fitting methods can be adopted to solve some computer vision tasks. Thanks to their implementation, that can be very efficient and fast, they can be adopted for real time applications too [3]. The following discussion will trail the guidelines of [91].

By denoting with $\boldsymbol{r}(\boldsymbol{x})$ the residual error vector, the objective is to find the minimum of $f(\boldsymbol{x})$ defined as

$$f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{r}(\boldsymbol{x})||^2 = \frac{1}{2}\sum_i r_i(\boldsymbol{x})^2 \tag{C.1}$$

This optimization can be achieved by methods detailed in Appendix B. The Jacobian $\boldsymbol{J_r}(\boldsymbol{x})$ of the vector function $\boldsymbol{r}(\boldsymbol{x})$ is by definition

$$[\boldsymbol{J_r}(\boldsymbol{x})]_{ij} = \frac{\partial r_i(\boldsymbol{x})}{\partial x_j} \tag{C.2}$$

while the expression for the components of the gradient of $f(\boldsymbol{x})$ is

$$[\nabla f(\boldsymbol{x})]_j = \frac{\partial f(\boldsymbol{x})}{\partial x_j} = \frac{\partial f(\boldsymbol{x})}{\partial r_i(\boldsymbol{x})}\frac{\partial r_i(\boldsymbol{x})}{\partial x_j} = \sum_i r_i(\boldsymbol{x})\frac{\partial r_i(\boldsymbol{x})}{\partial x_j} = \sum_i r_i(\boldsymbol{x})[\boldsymbol{J_r}(\boldsymbol{x})]_{ij} \tag{C.3}$$

By merging the two previous equation the relation between the gradient of $f(\boldsymbol{x})$ and the Jacobian of $\boldsymbol{r}(\boldsymbol{x})$ is

$$\nabla f(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x}) \tag{C.4}$$

Moreover the Hessian matrix $\boldsymbol{H}(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ is, by definition

$$\begin{gathered}[\nabla^2 f(\boldsymbol{x})]_{lm} = \frac{\partial^2 f(\boldsymbol{x})}{\partial x_l \partial x_m} = \frac{\partial}{\partial x_l}\left(\sum_i r_i(\boldsymbol{x})\frac{\partial r_i(\boldsymbol{x})}{\partial x_m}\right) = \\ \sum_i \frac{\partial r_i(\boldsymbol{x})}{\partial x_l}\frac{\partial r_i(\boldsymbol{x})}{\partial x_m} + \sum_i r_i(\boldsymbol{x})\frac{\partial^2 r_i(\boldsymbol{x})}{\partial x_l \partial x_m}\end{gathered} \tag{C.5}$$

Or, in matrix notation

$$\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{J_R}(\boldsymbol{x})^T\boldsymbol{J_R}(\boldsymbol{x}) + \boldsymbol{S}(\boldsymbol{x}) \tag{C.6}$$

C.1 Gauss-Newton method

By recalling that $\boldsymbol{H}(f(\boldsymbol{x}_k))\boldsymbol{d}(\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{\delta x}_k$, Gauss-Newton method is written as

$$[\boldsymbol{J_R}(\boldsymbol{x}_k)^T \boldsymbol{J_R}(\boldsymbol{x}_k)]\boldsymbol{\delta x}_k = -\boldsymbol{J_R}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k) \tag{C.7}$$

In the expression for the Hessian matrix, the component $\boldsymbol{S}(\boldsymbol{x})$ has been omitted as in several cases it has minor importance with respect to the Jacobians term [91]. Eq. (C.10) is a linear system in $\boldsymbol{\delta x}_k$ that can be solved through a singular value decomposition method or QR method.

C.2 Levenberg-Marquardt method

The Levenberg-Marquardt method is a trust region method that solves the constrained minimization problem

$$\min\ f_k(\boldsymbol{s}) \text{ with } ||\boldsymbol{s}|| \leq \delta_k \tag{C.8}$$

where f_k is the linear expansion of $\boldsymbol{r}(\boldsymbol{x})$ as in Eq. (C.1)

$$f_k(\boldsymbol{s}) = \frac{1}{2}||\boldsymbol{r}(\boldsymbol{x}_k) + \boldsymbol{J_R}(\boldsymbol{x}_k)\boldsymbol{s}||^2 \tag{C.9}$$

In particular the step direction update is searched by solving [92]

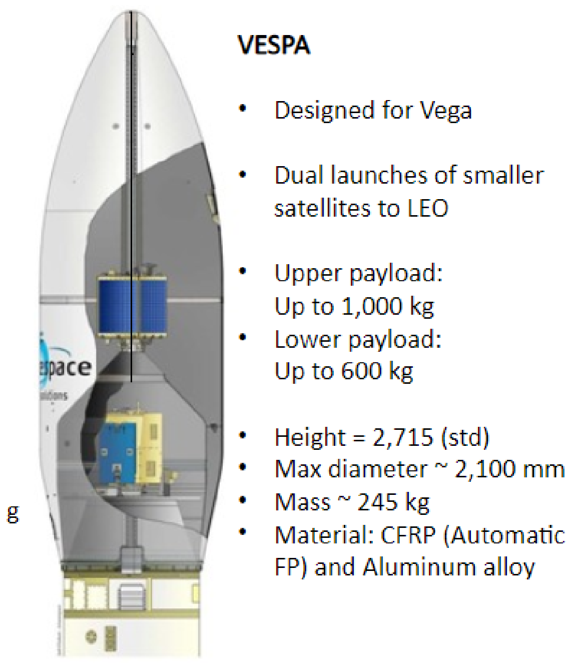
$$[\boldsymbol{J_R}(\boldsymbol{x}_k)^T \boldsymbol{J_R}(\boldsymbol{x}_k) + \lambda_k \boldsymbol{I}]\boldsymbol{\delta x}_k = -\boldsymbol{J_R}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k) \tag{C.10}$$

D Vespa 3D model development

During the CNN testing the development of a 3D model for Vespa payload adapter of Vega launcher has been accomplished. Vespa is a challenging object for a visual system as it is almost completely black and is an axisymmetric object. Moreover it is an important target for European space debris removal missions under the ESA project ADRIOS, Active Debris Removal/ In-Orbit Servicing. The model has been generated almost fully in Blender, apart from small details that have been modeled in Autodesk Inventor and then imported in Blender. The reference material has been provided by ASTRA research team at Department of Aerospace Sciences and Technology of Politecnico di Milano and some other pictures have been found online. The main two images used for Vespa development are shown in Fig. D.1. The first one, Fig. D.1a, has been used to understand the body dimensions. The broad values given, like the heigh and the diameter of Vespa, are compared to the pixels dimensions in the image to obtain a pixel to meter conversion. Then, by measuring in pixels the quantities of interest, their respective dimensions in meters can be obtained. The other picture, Fig. D.1b, was mainly used to obtain visual accuracy of the model.

Two procedural textures have been developed for Vespa. The first one, for the golden-coloured metallic material, is obtained with a noise node as displacement and it is used for the lower and upper parts of the adapter. The second one, for the pre-impregnated carbon fiber, is obtained with a checker node to give the characteristic carbon fiber stripes and a noise node displacement for the roughness.

Two models of Vespa were developed during this thesis. The first one, shown in Fig. D.2, has been the first one and has been used in the CNN tests. Later on, more details were added to obtain the model seen in Fig. D.3. These part are modeled in Inventor, as it is easier to generate them with an engineering software, and are then imported and textured in Blender.


VESPA

- Designed for Vega
- Dual launches of smaller satellites to LEO
- Upper payload: Up to 1,000 kg
- Lower payload: Up to 600 kg
- Height = 2,715 (std)
- Max diameter ~ 2,100 mm
- Mass ~ 245 kg
- Material: CFRP (Automatic FP) and Aluminum alloy

(a) Vespa, source image for dimensions [93]

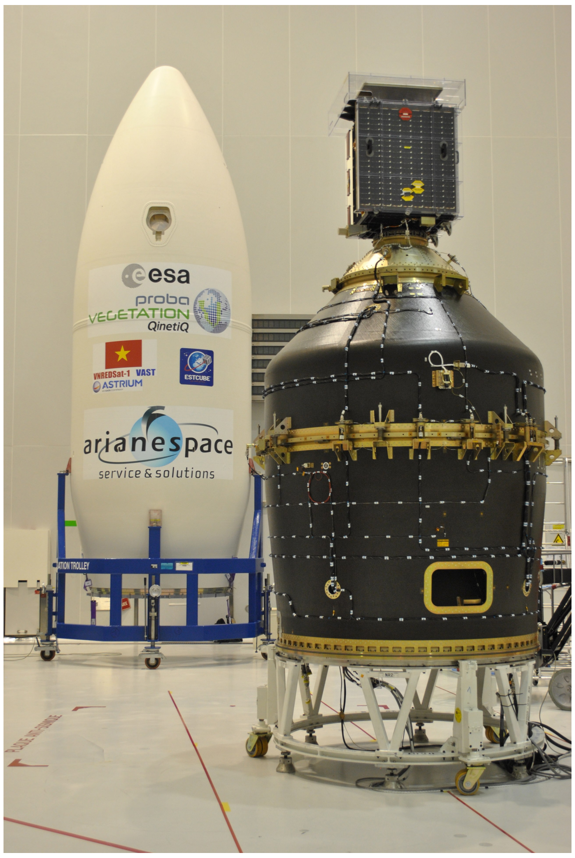
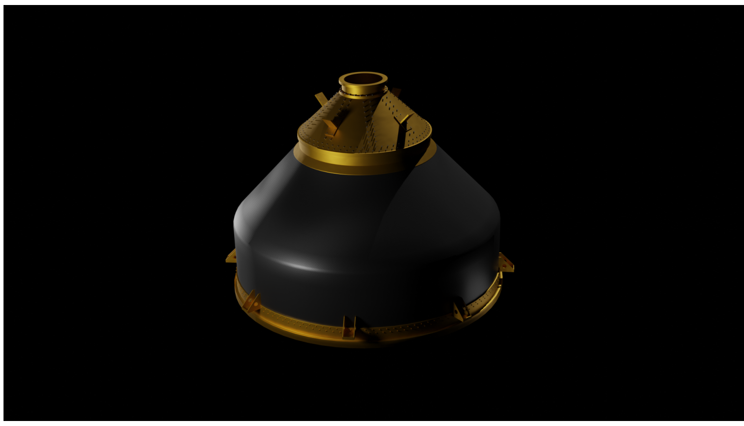
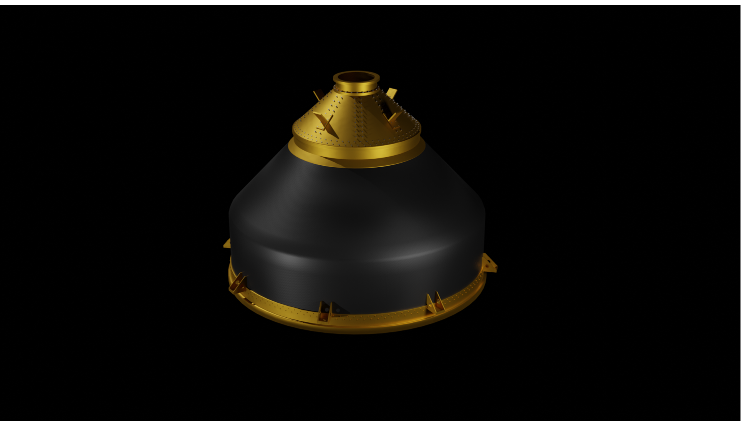

(b) Vespa, real image [94]

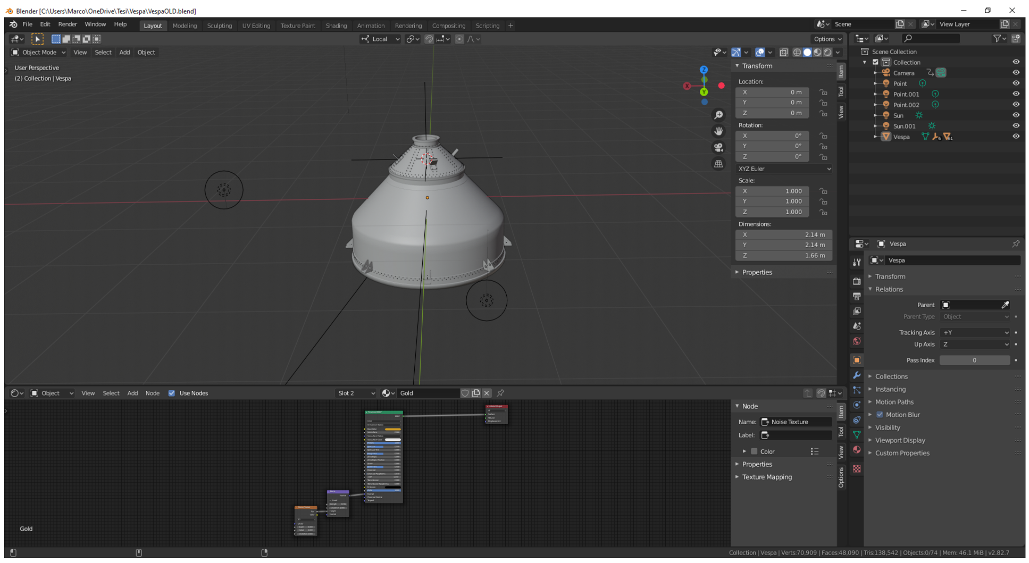
Fig. D.1: Vespa, source images



(a) Vespa rendering example, 1

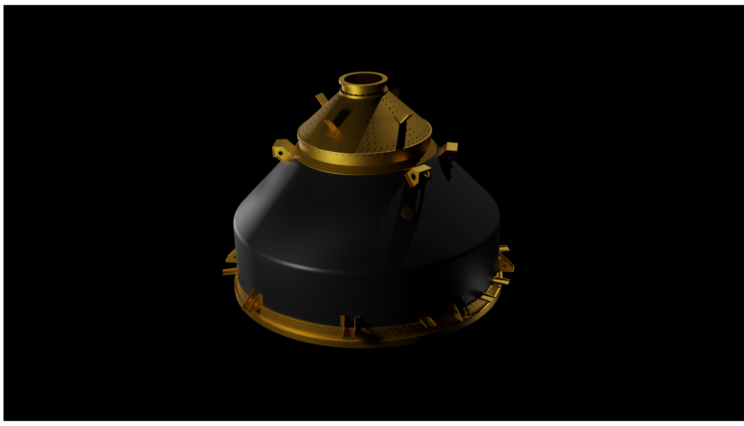


(b) Vespa rendering example, 2

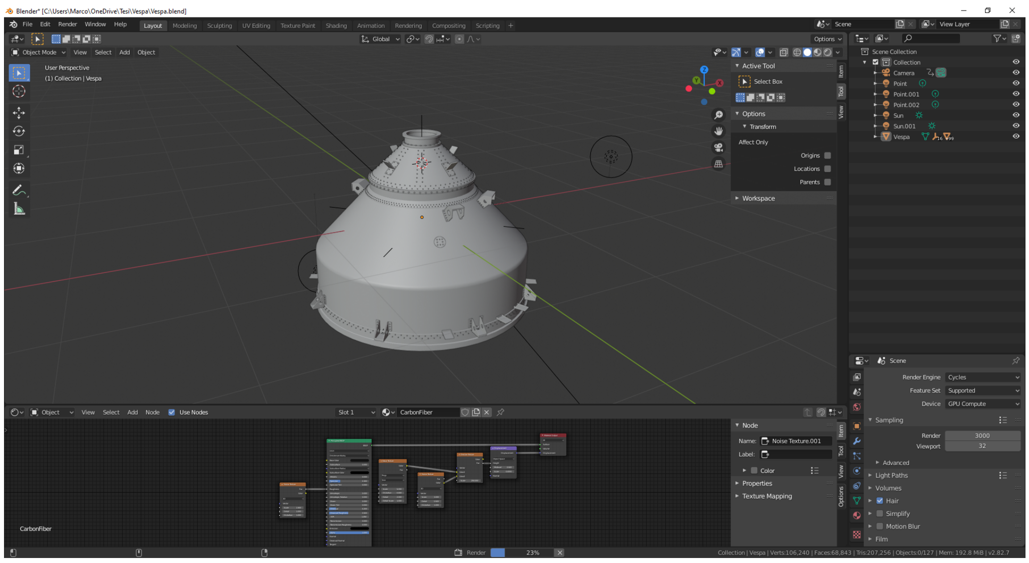


(c) Vespa in Blender environment

Fig. D.2: Examples of Vespa initial renderings


(a) Vespa rendering example, 1

(b) Vespa rendering example, 2


Render Engine: Cycles
Feature Set: Supported
Device: GPU Compute
Sampling
Render: 3000
Viewport: 32

Name: Noise Texture.001

Collection | Vespa | Verts:106,240 | Faces:68,843 | Tris:207,256 | Objects:0/127 | Mem: 192.8 MiB | v2.82.7

(c) Vespa in Blender environment

Fig. D.3: Examples of Vespa renderings

E CNN Testing dataset overview

In this section the CNN test datasets are reported. They are organized as follows:

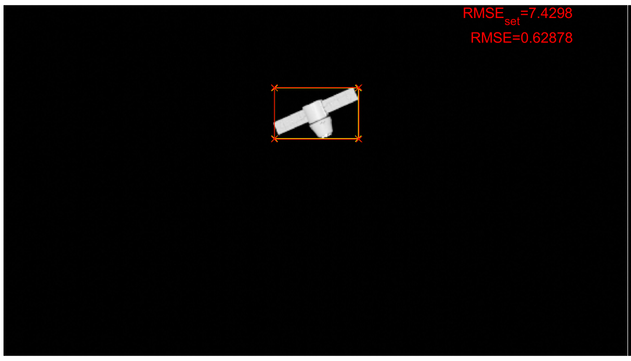
- Dragon, Earth and Space
- Dragon without solar array, Earth and Space
- Vespa, Earth and Space
- MarCO, Earth and Space
- Envisat, Earth and Space
- MarCO, close Moon
- Dragon, close Moon
- Dragon, far Moon
- Dragon, far Comet
- Dragon, close Comet
- Dragon, Earth and Space, Noise
- Dragon, Earth and Space, Blur

For every test the two best and the two worst estimations are reported. Four random examples along with the estimation closest to the dataset RMSE are shown as well. The error and RMSE histogram are presented lastly along with the error CDF, the cumulative density function.

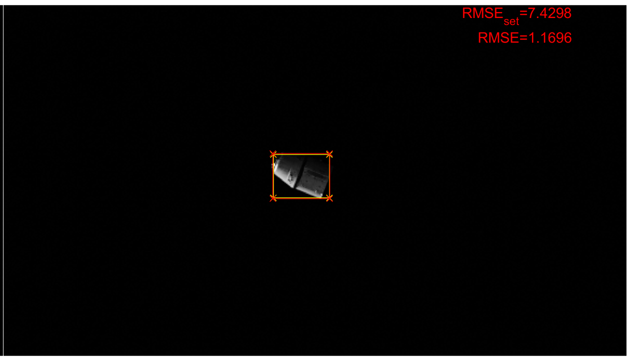
The network performs equally well on both Dragon and Dragon without solar array, Earth and Space cases. Vespa, MarCO and Envisat follow next, which are completely new satellites that the network has never seen during training. Thanks to the training step in which features are learned in the early layers, the CNN still recognizes them with good results. Performance starts dropping however when the background is changed too, adding another layer of difficulty. The CNN shows acceptable behaviour in the cases of close Moon for Dragon and MarCO spacecraft, however Dragon cases with far Moon and Rosetta comet are insufficient. Very strong color gradients, such as those found in the shadows due to the comet morphology or in the terminator region of both the Moon and the comet have not been encountered and trained for during the design phase of the CNN. Moreover images with the whole Moon or Rosetta comet in the background are often very erroneous since training was performed in LEO orbit, where the Earth was never fully imaged.

Finally CNN performance with Noise and Blur on the Dragon spacecraft case are shown, denoting moderate robustness to these unwanted effects.

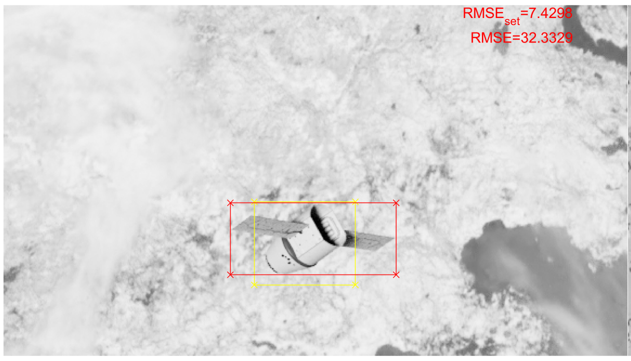
E.1 Dragon, Earth & Space



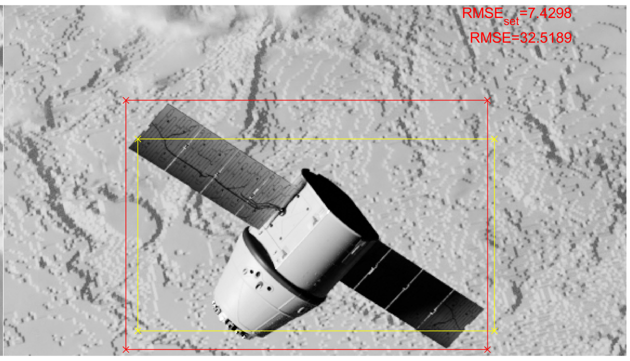
(a) Best estimation, 1



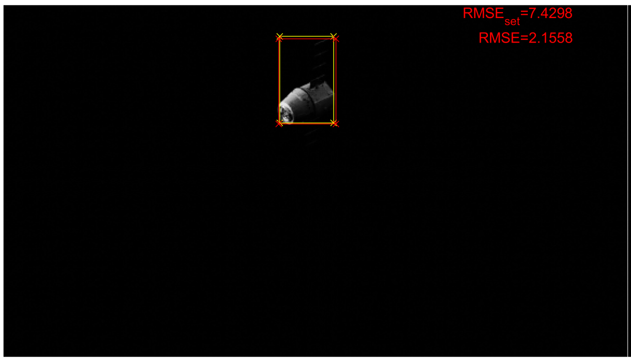
(b) Best estimation, 2



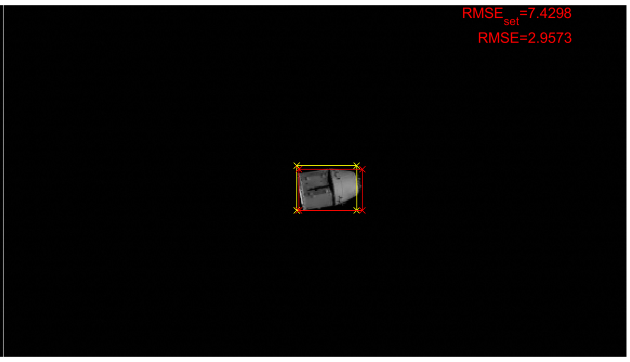
(c) Worst estimation, 1



(d) Worst estimation, 2

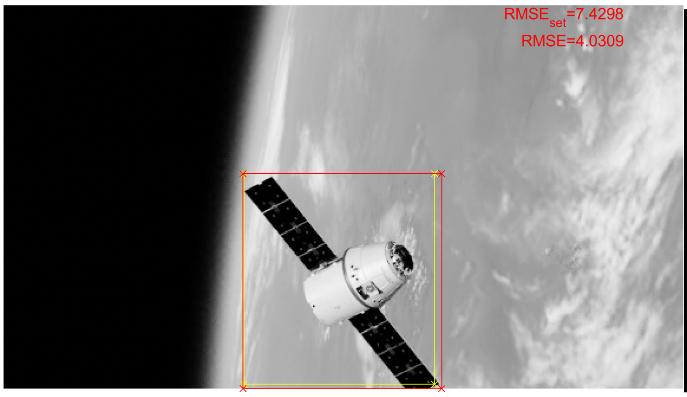


(e) Random example, 1

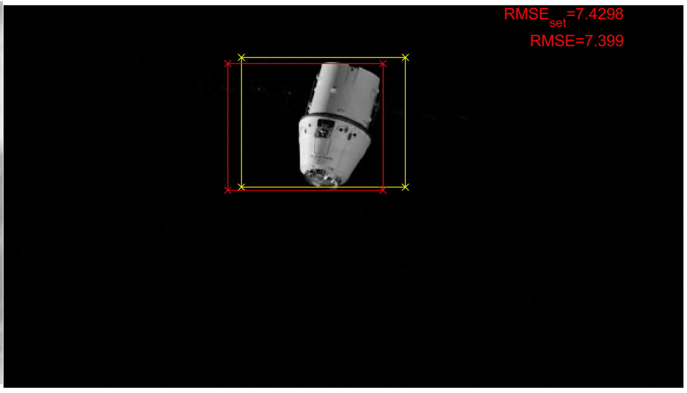


(f) Random example, 2

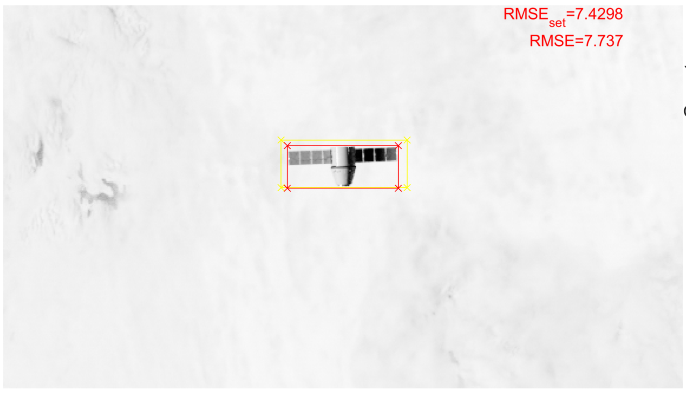
Fig. E.1: Dragon, Earth & Space, 1



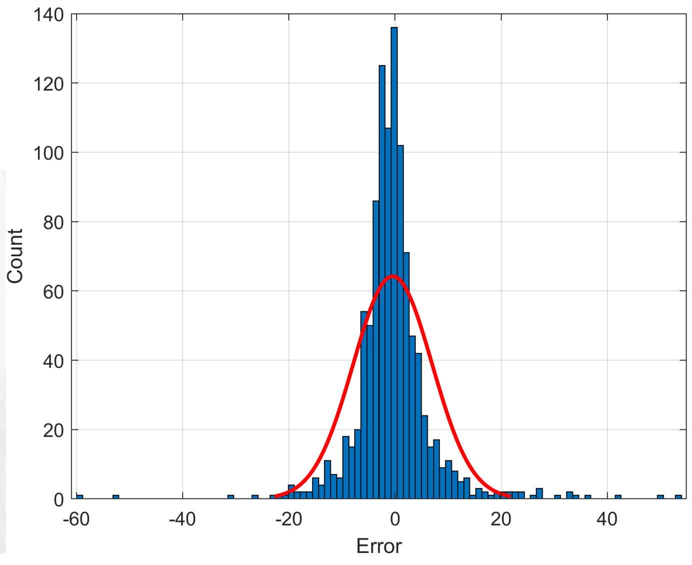
(a) Random example, 3



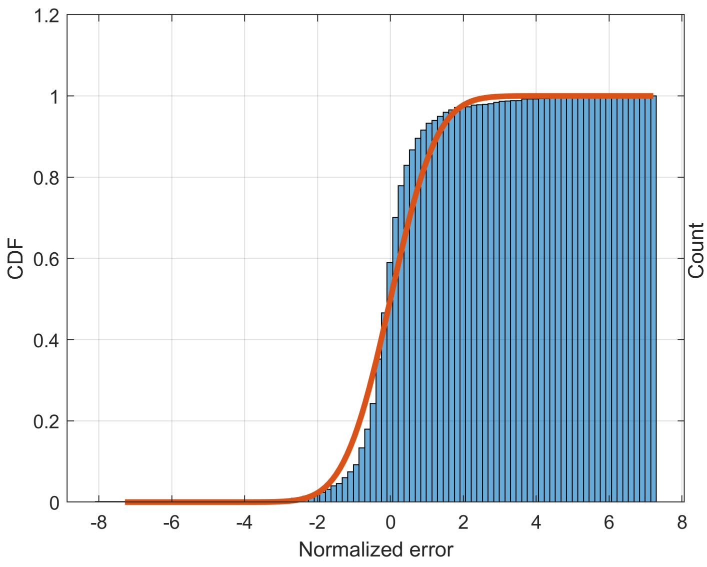
(b) Random example, 4



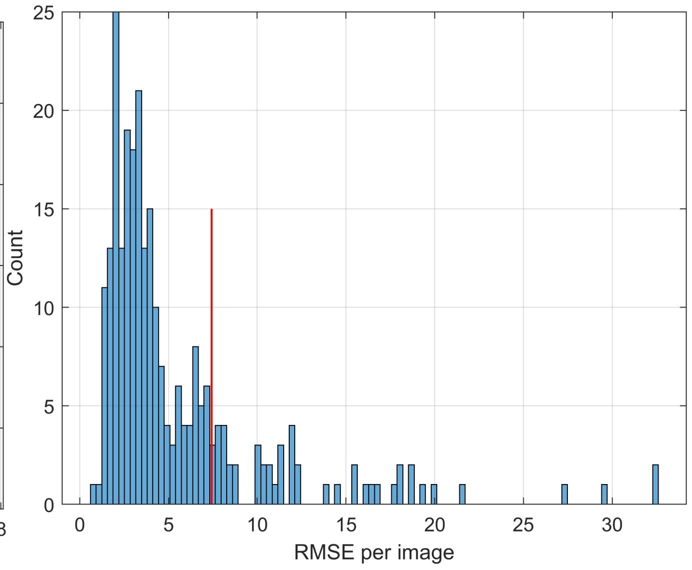
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



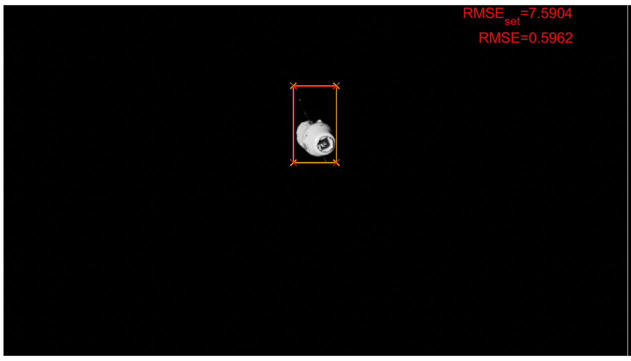
(e) CDF vs normal CDF in red



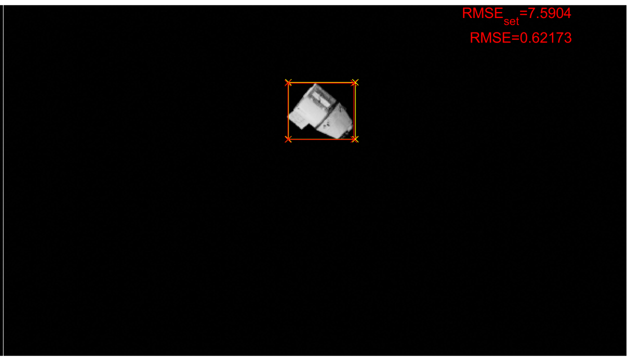
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.2: Dragon, Earth & Space, 2

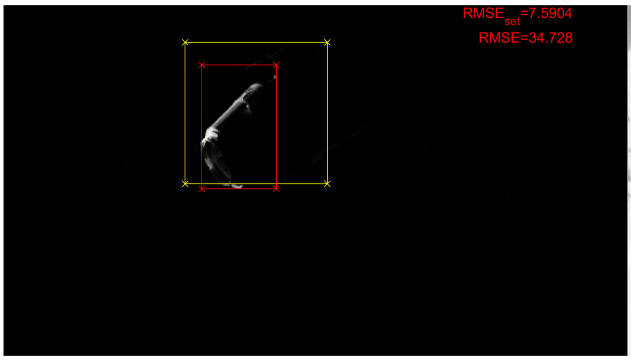
E.2 Dragon without solar array, Earth & Space



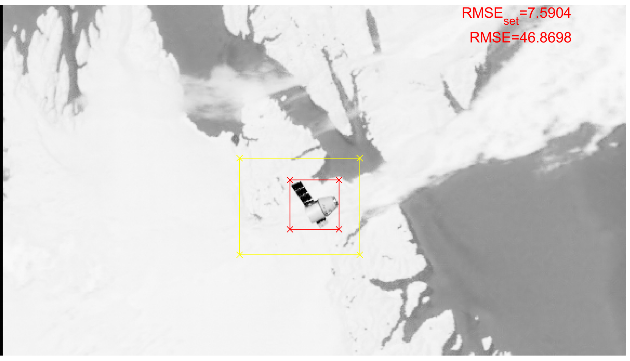
(a) Best estimation, 1



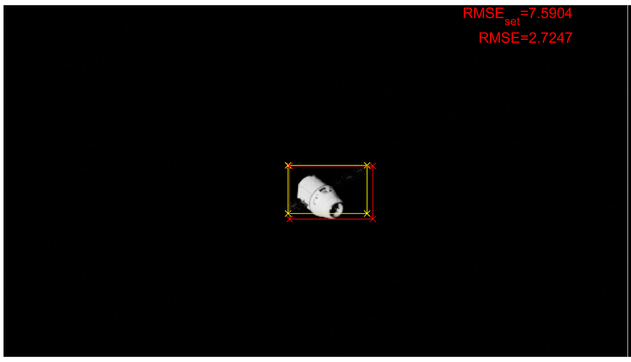
(b) Best estimation, 2



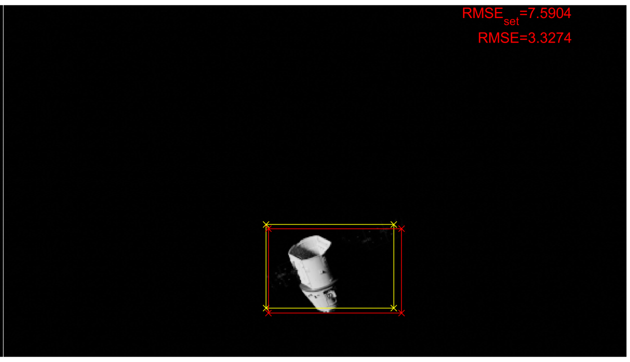
(c) Worst estimation, 1



(d) Worst estimation, 2

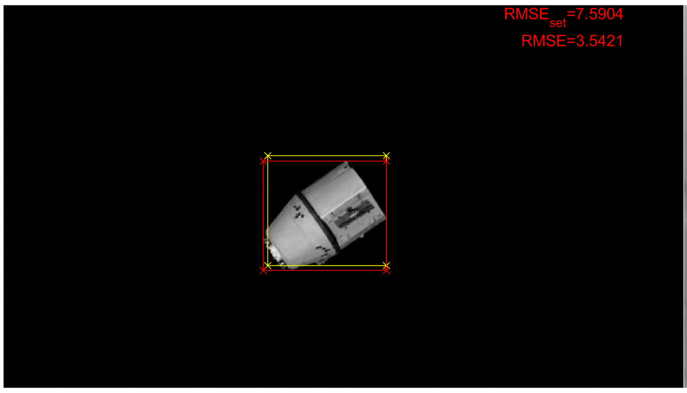


(e) Random example, 1

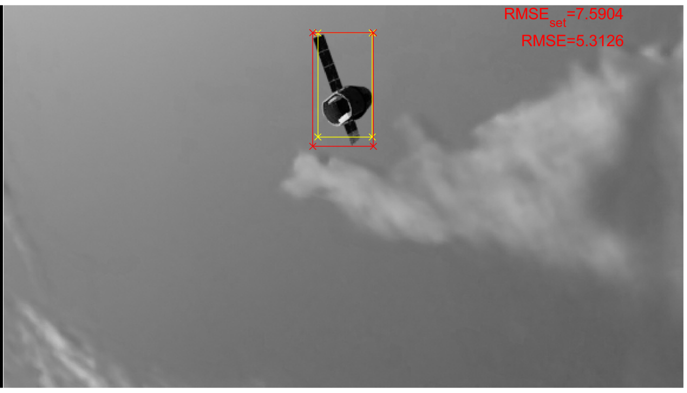


(f) Random example, 2

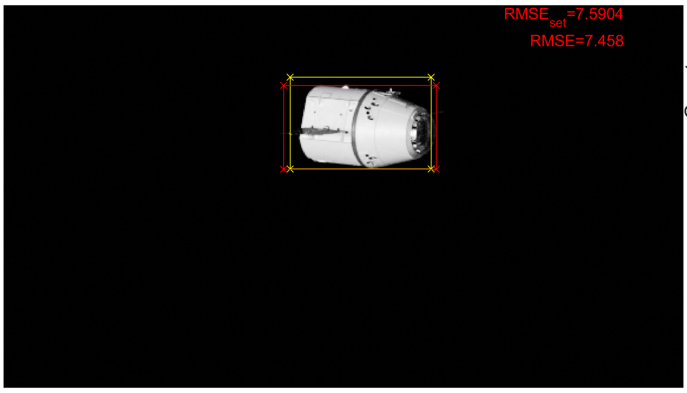
Fig. E.3: Dragon without solar array, Earth & Space, 1



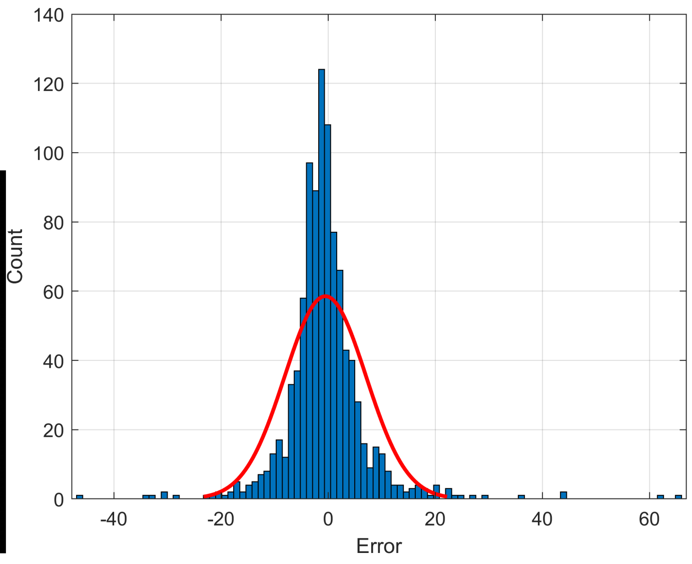
(a) Random example, 3



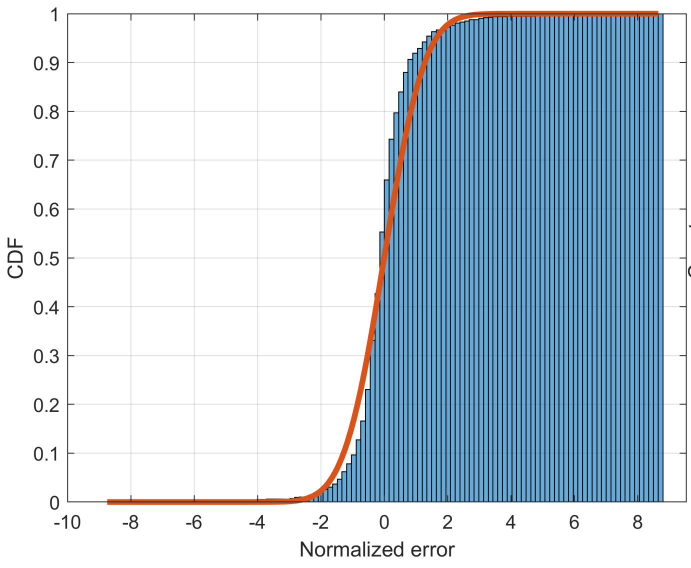
(b) Random example, 4



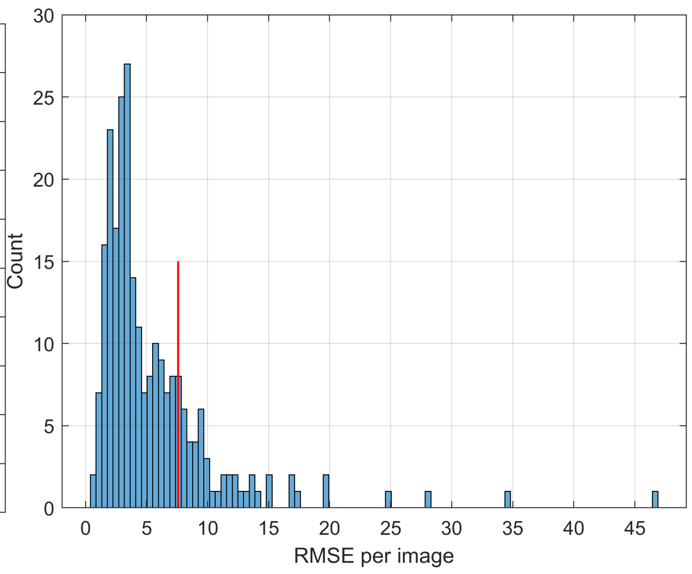
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



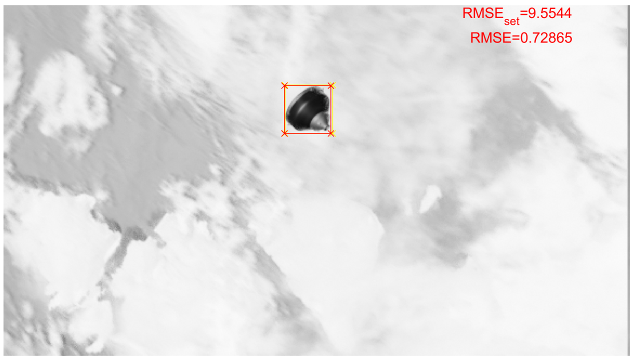
(e) CDF vs normal CDF in red



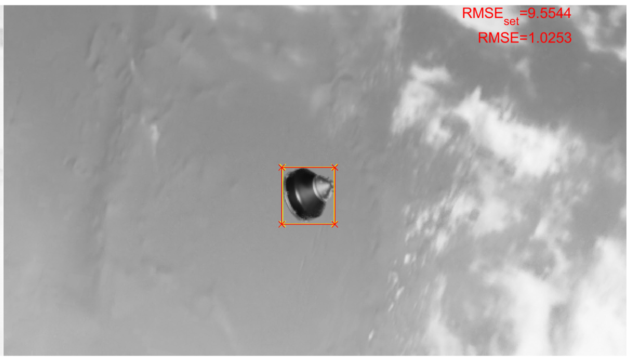
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.4: Dragon without solar array, Earth & Space, 2

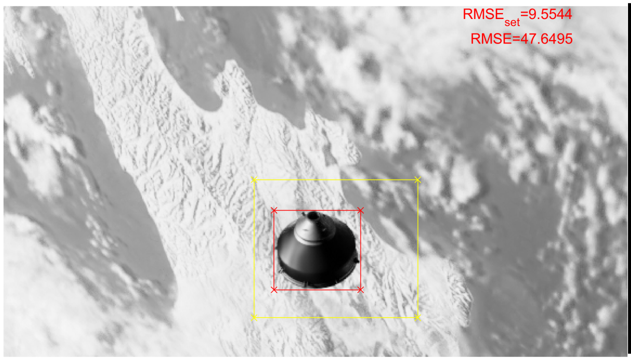
E.3 Vespa, Earth & Space



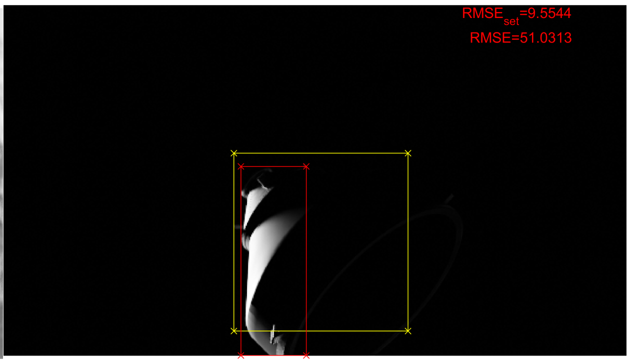
(a) Best estimation, 1



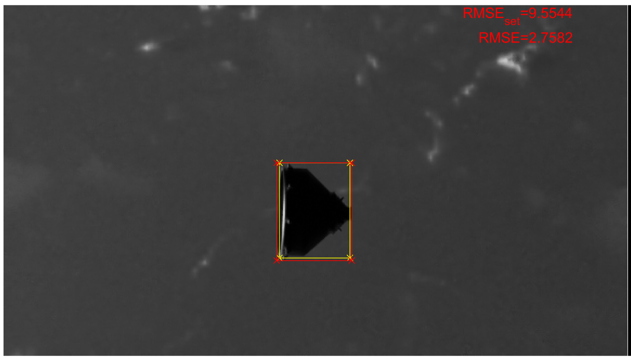
(b) Best estimation, 2



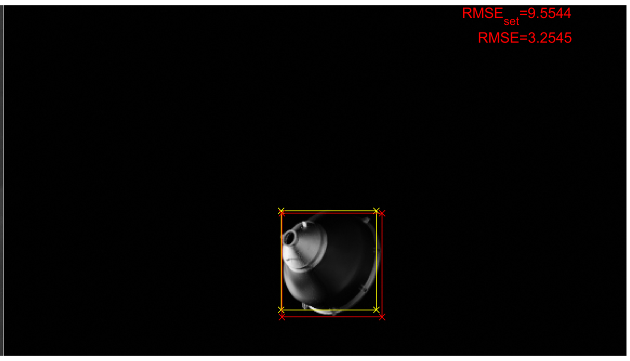
(c) Worst estimation, 1



(d) Worst estimation, 2

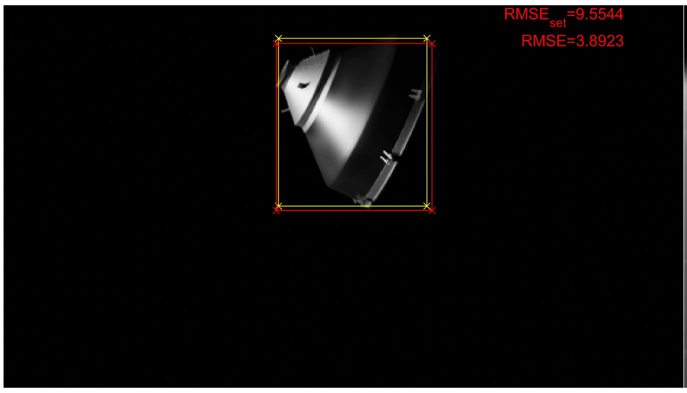


(e) Random example, 1

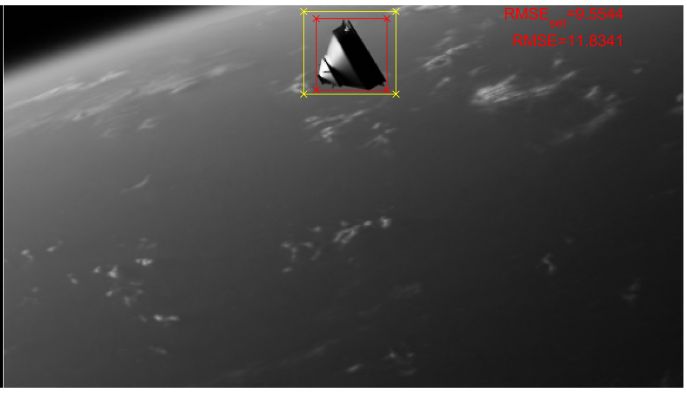


(f) Random example, 2

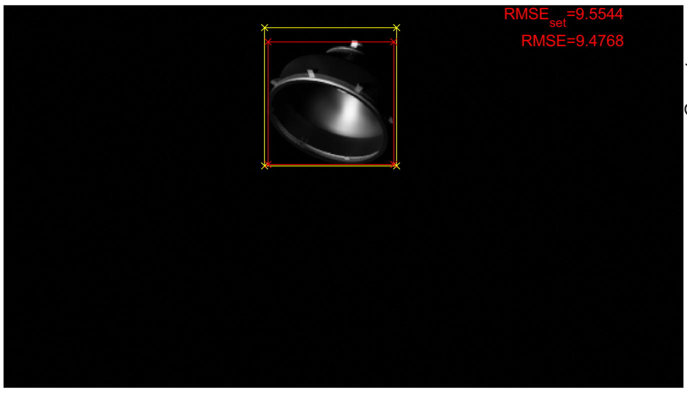
Fig. E.5: Vespa, Earth & Space, 1



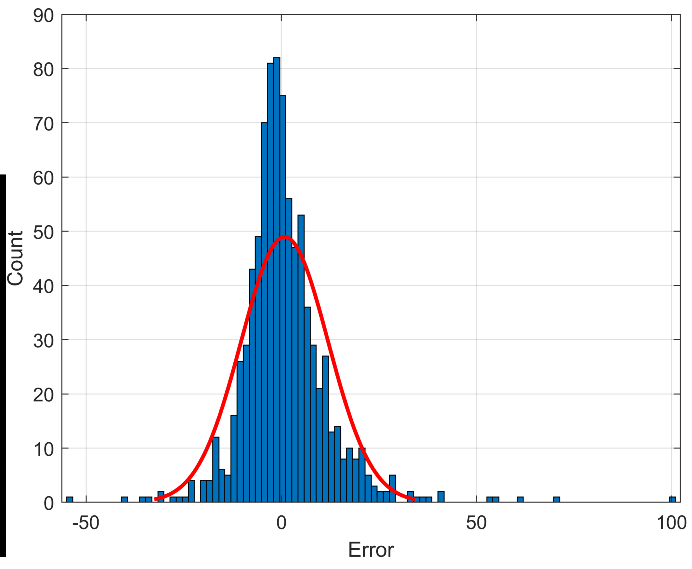
(a) Random example, 3



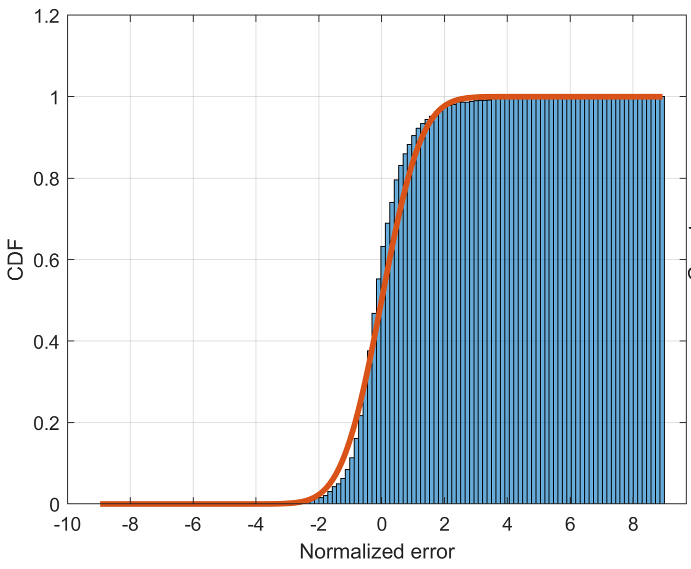
(b) Random example, 4



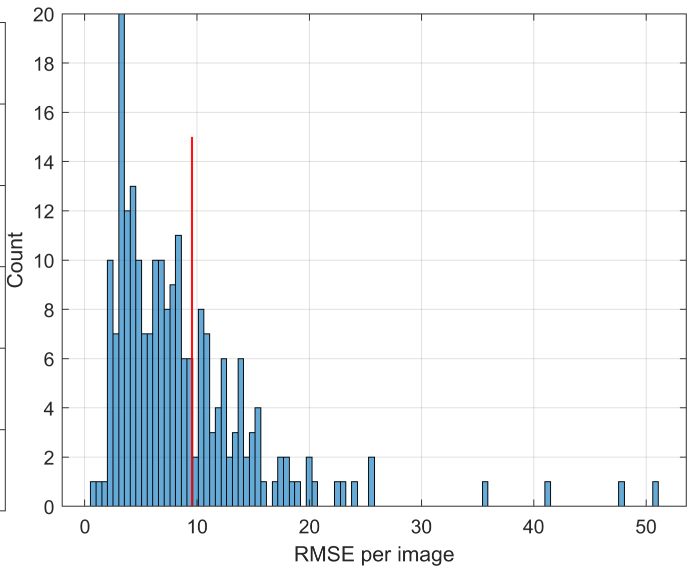
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



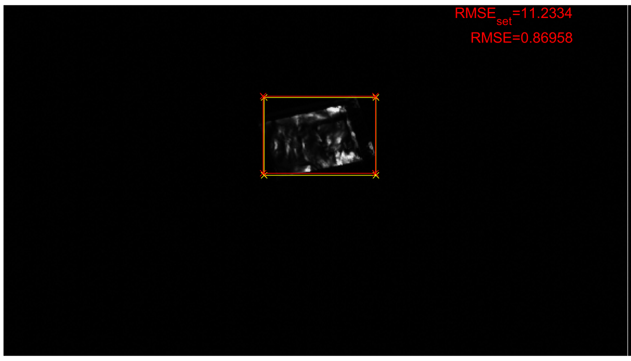
(e) CDF vs normal CDF in red



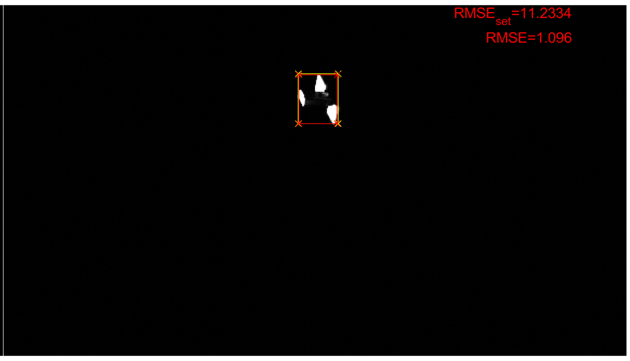
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.6: Vespa, Earth & Space, 2

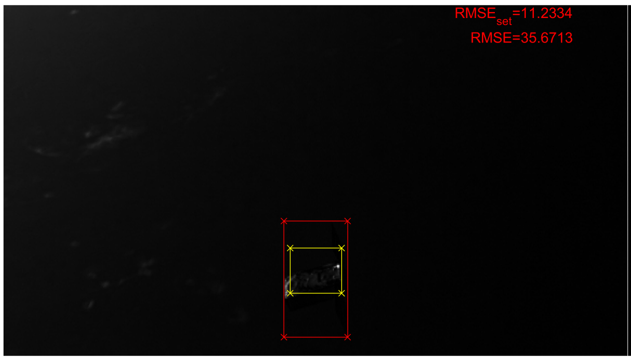
E.4 MarCO, Earth & Space



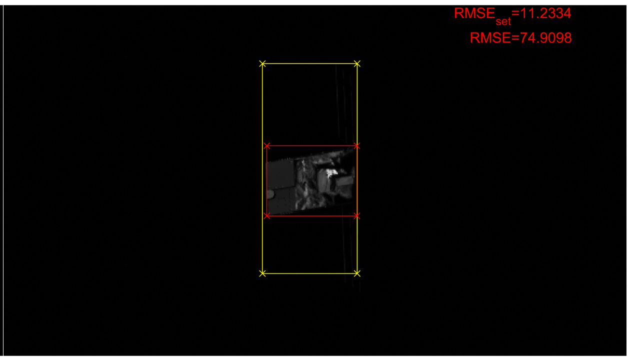
(a) Best estimation, 1



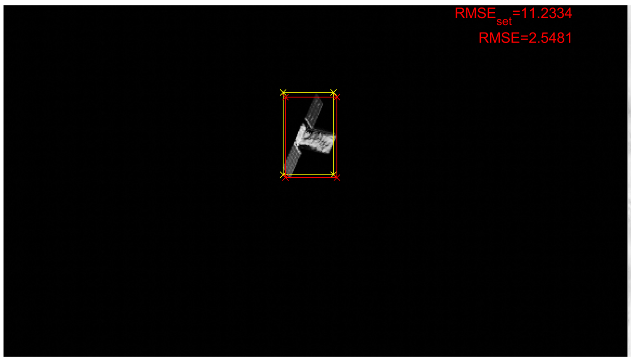
(b) Best estimation, 2



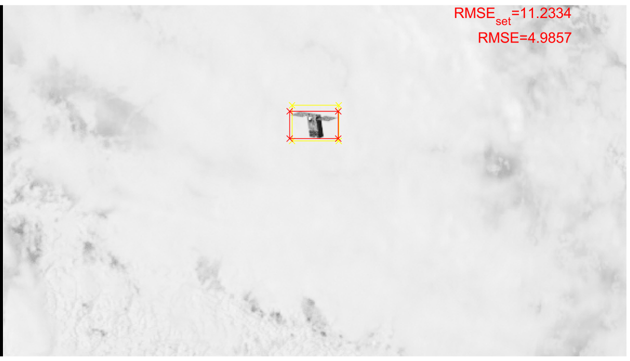
(c) Worst estimation, 1



(d) Worst estimation, 2

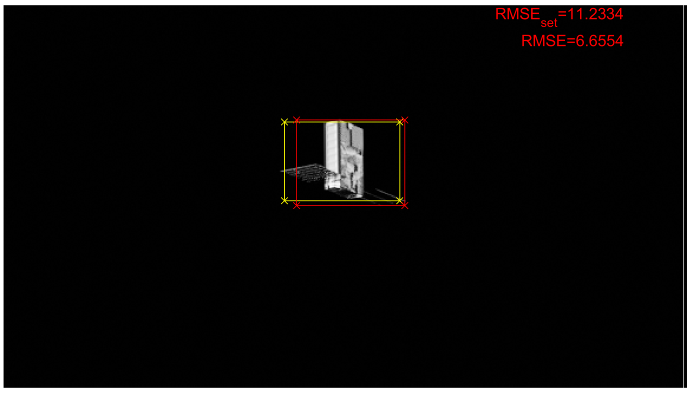


(e) Random example, 1

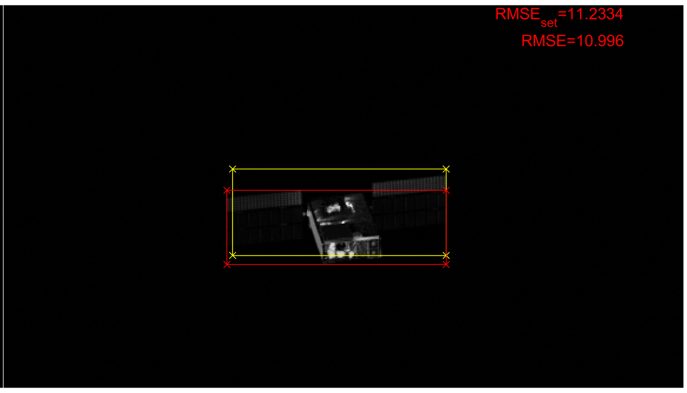


(f) Random example, 2

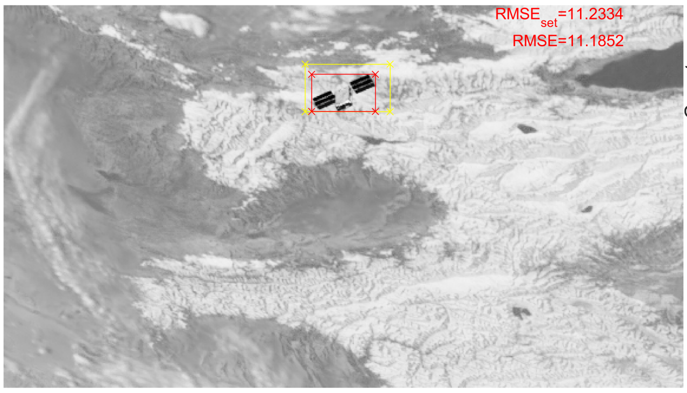
Fig. E.7: MarCO, Earth & Space, 1



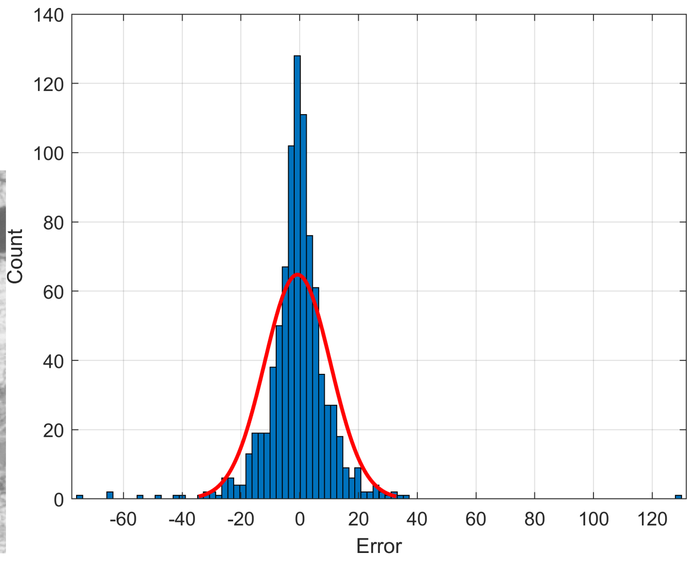
(a) Random example, 3



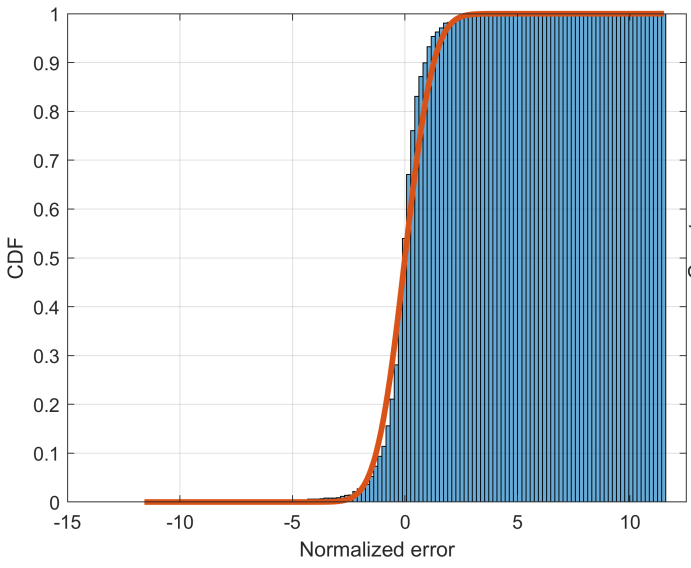
(b) Random example, 4



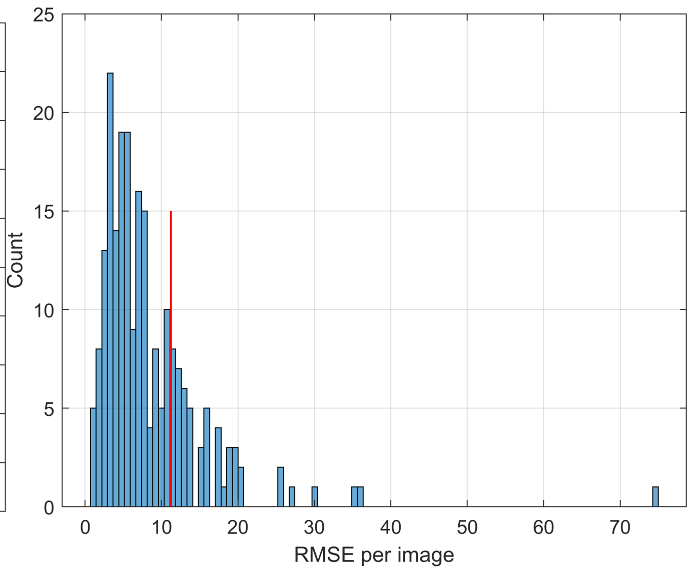
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



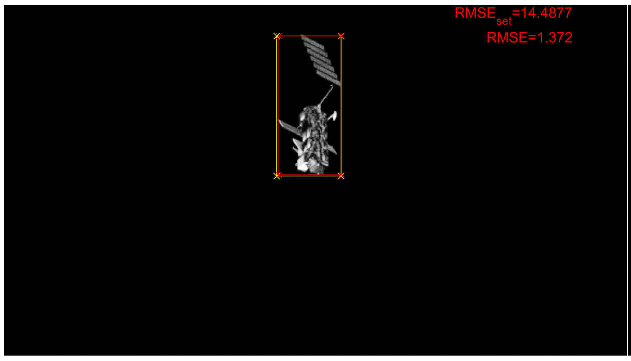
(e) CDF vs normal CDF in red



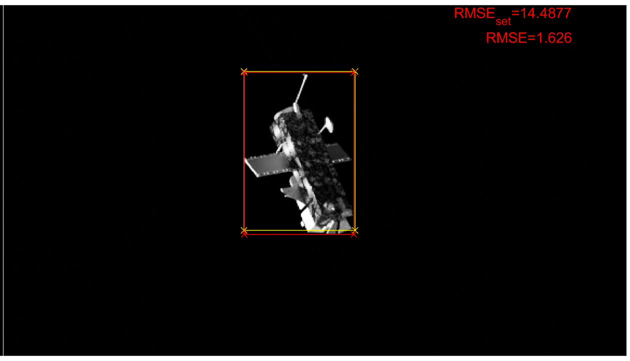
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.8: MarCO, Earth & Space, 2

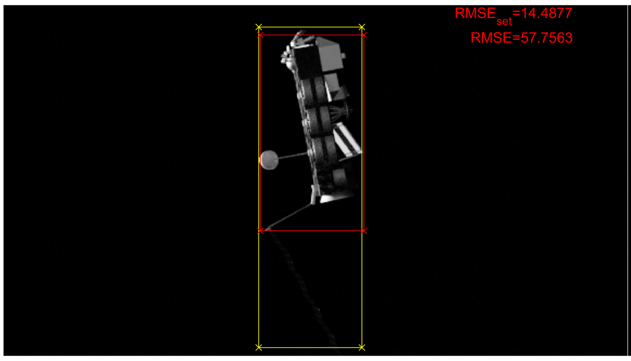
E.5 EnviSat, Earth & Space



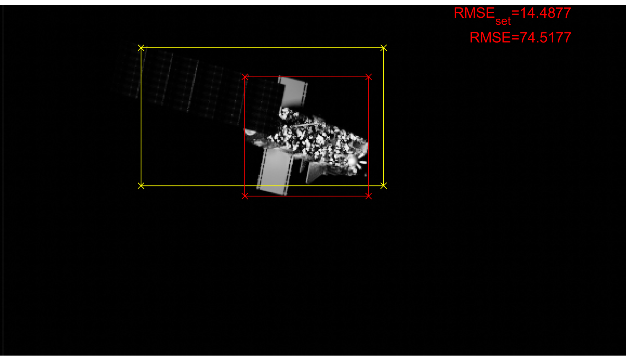
(a) Best estimation, 1



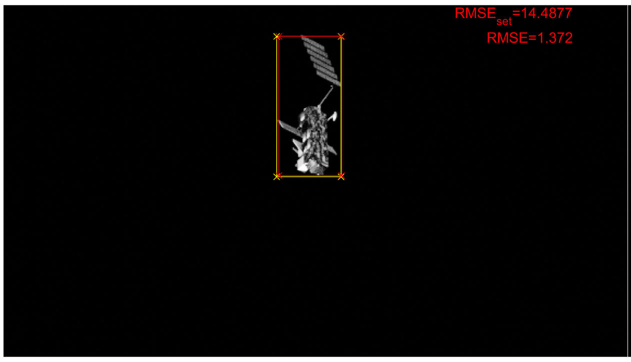
(b) Best estimation, 2



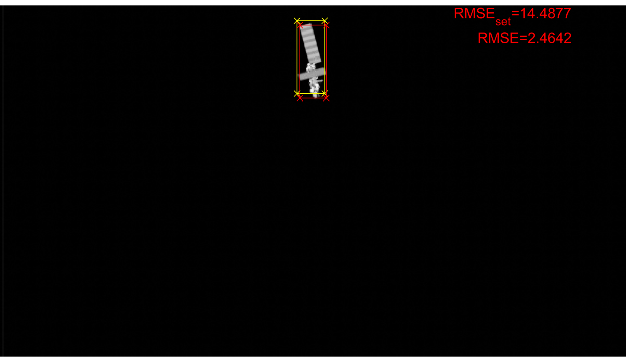
(c) Worst estimation, 1



(d) Worst estimation, 2

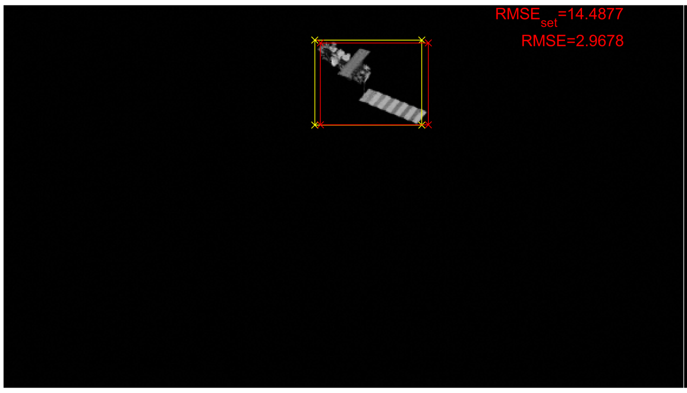


(e) Random example, 1

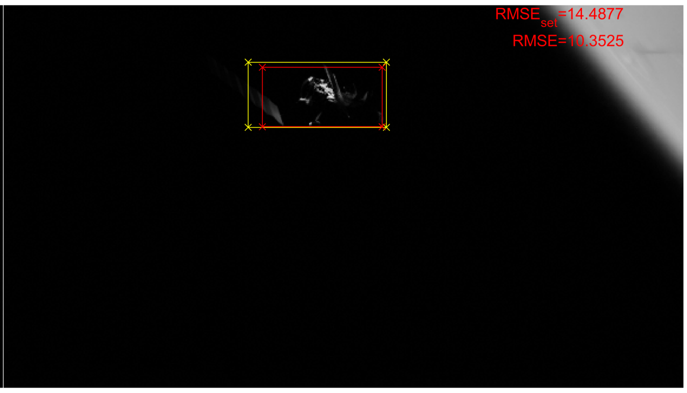


(f) Random example, 2

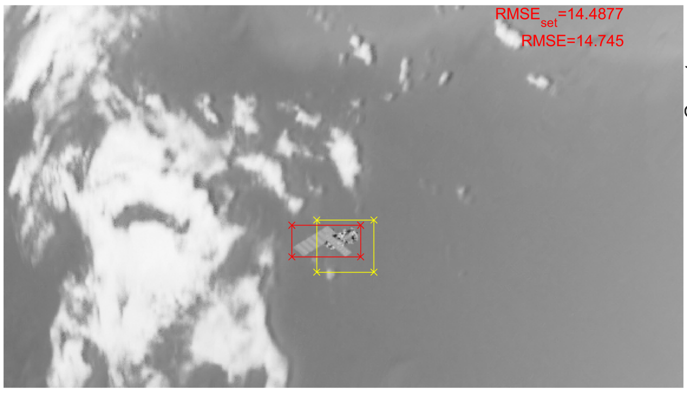
Fig. E.9: EnviSat, Earth & Space, 1



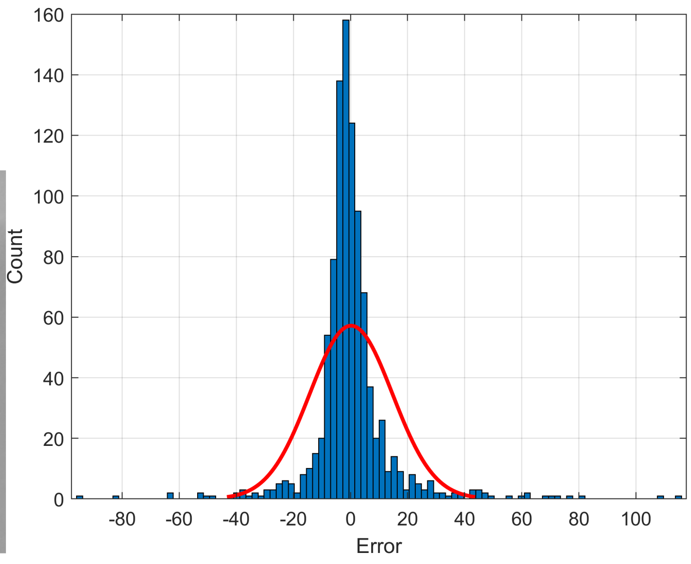
(a) Random example, 3



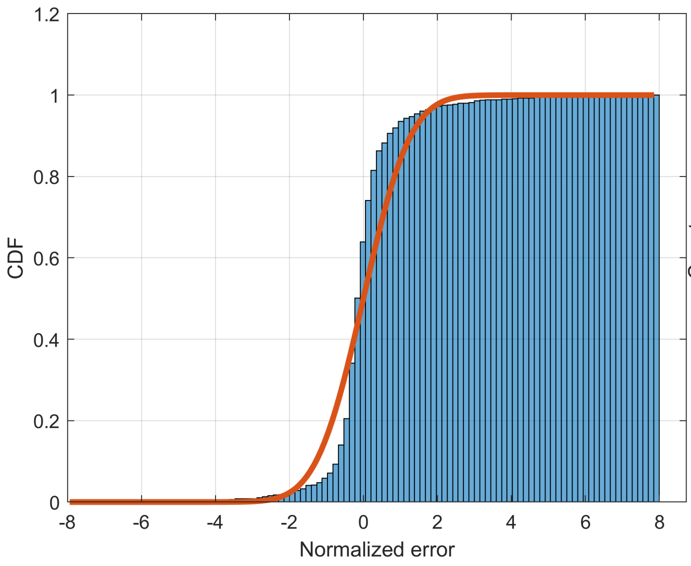
(b) Random example, 4



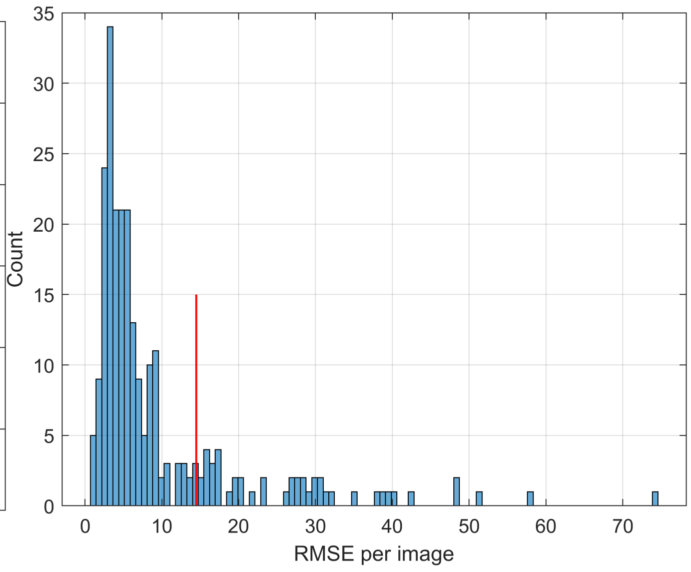
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



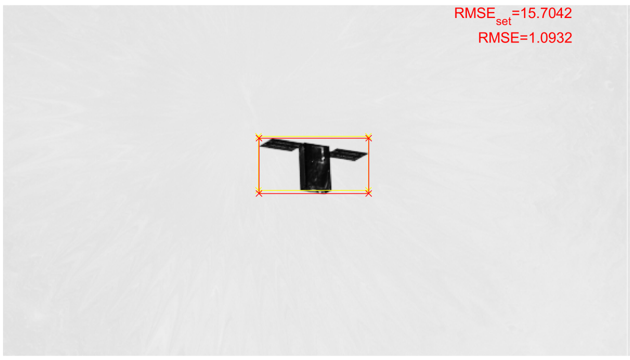
(e) CDF vs normal CDF in red



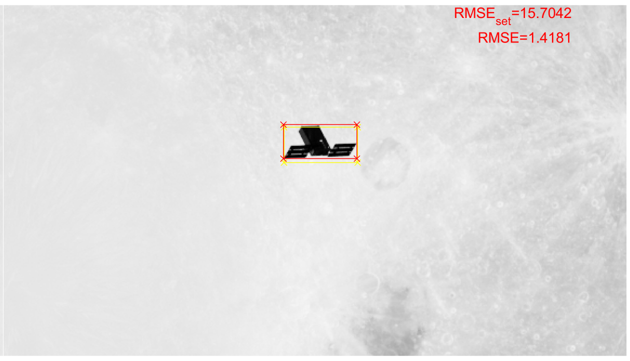
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.10: EnviSat, Earth & Space, 2

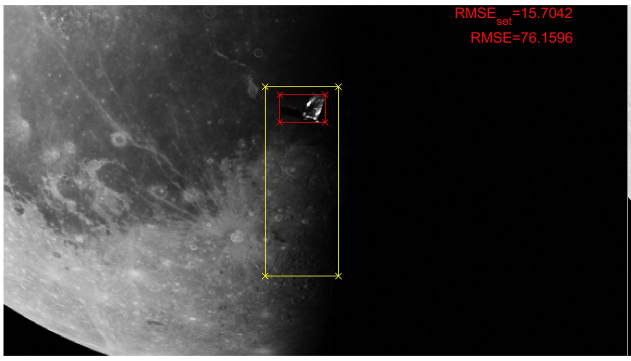
E.6 MarCO, close Moon



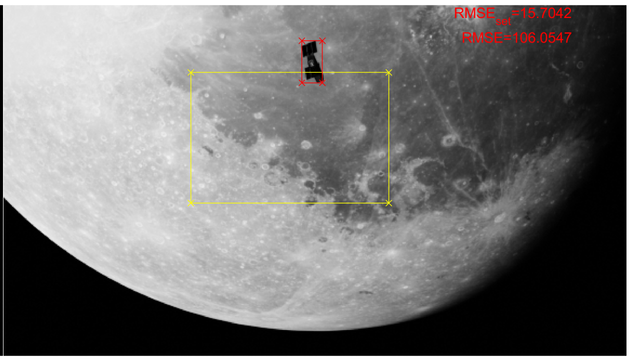
(a) Best estimation, 1



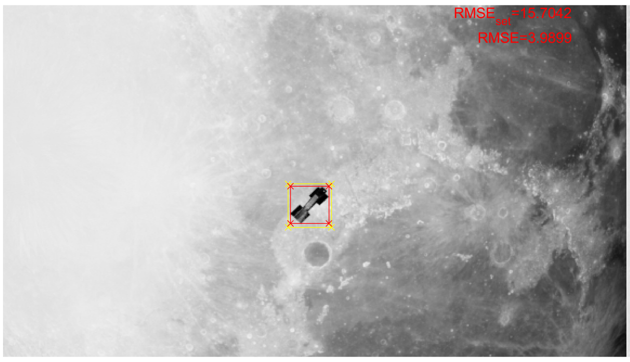
(b) Best estimation, 2



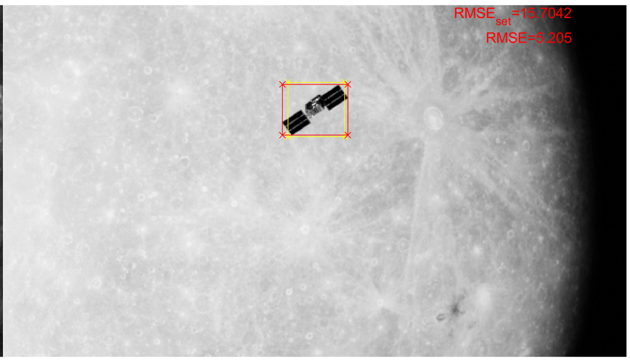
(c) Worst estimation, 1



(d) Worst estimation, 2

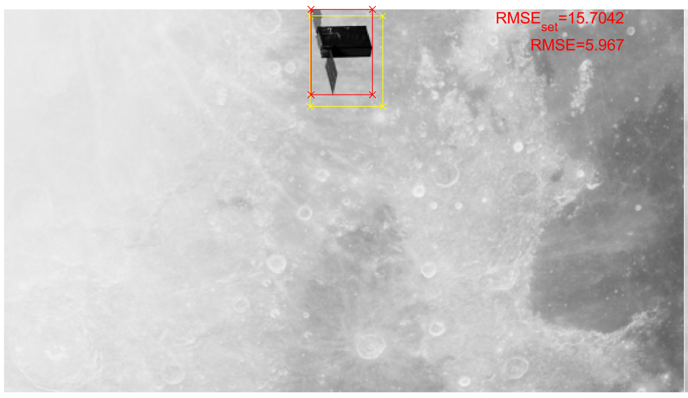


(e) Random example, 1

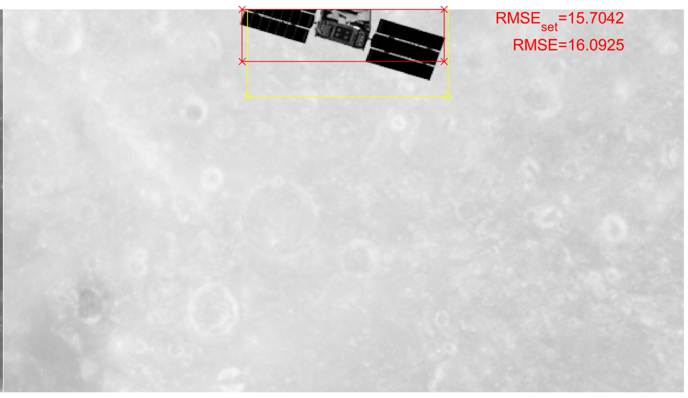


(f) Random example, 2

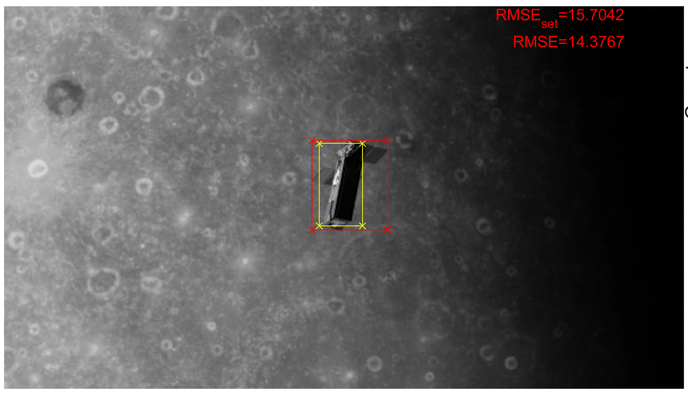
Fig. E.11: MarCO, close Moon, 1



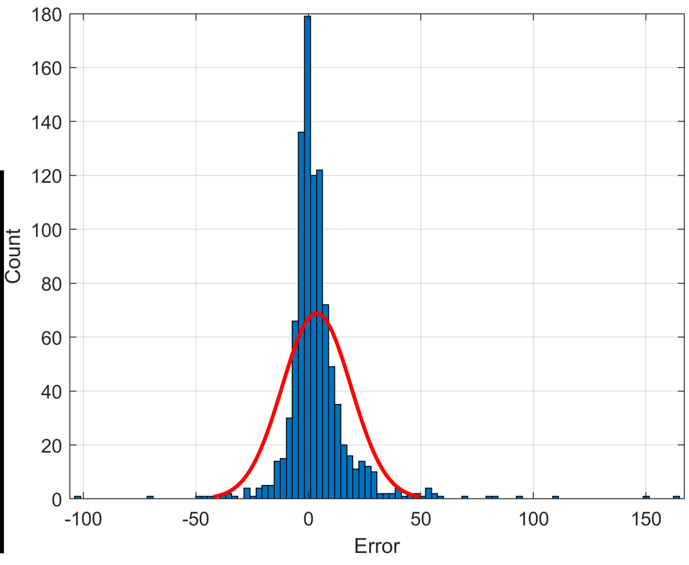
(a) Random example, 3



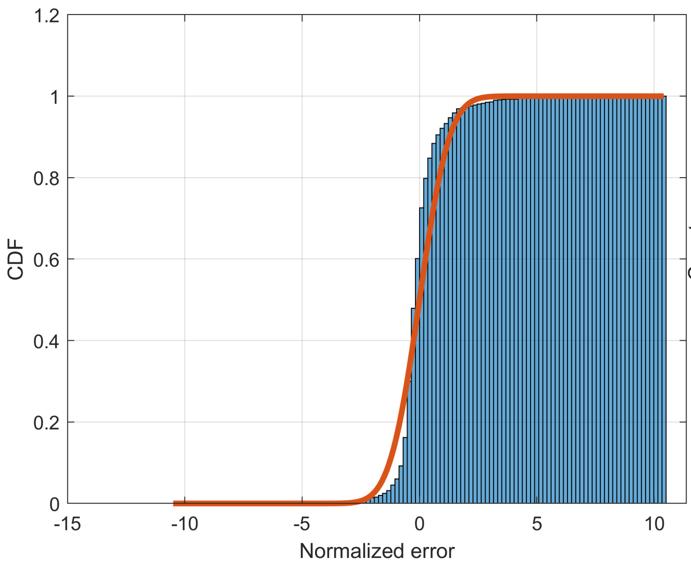
(b) Random example, 4



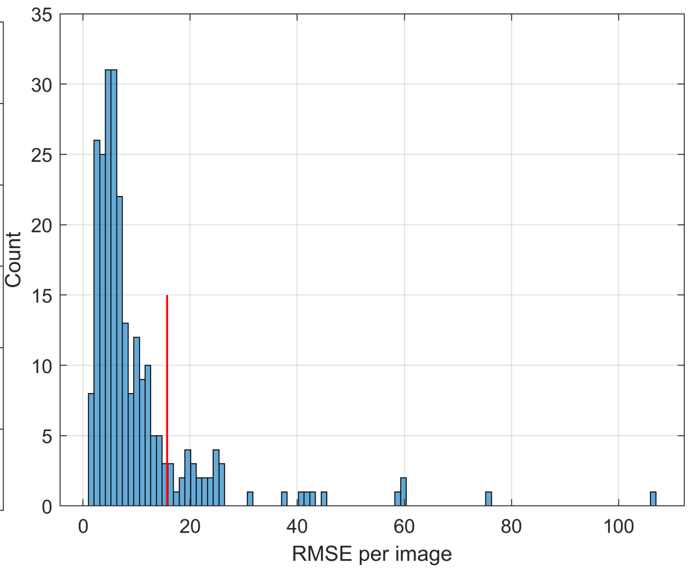
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



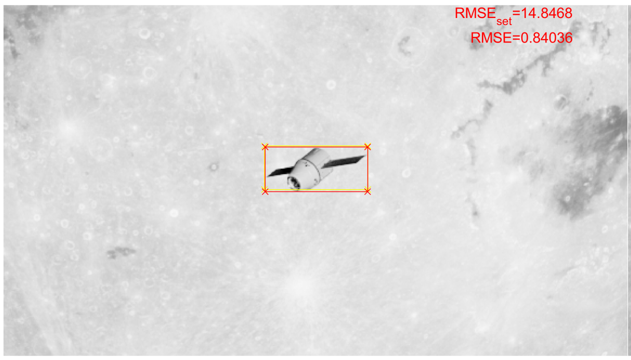
(e) CDF vs normal CDF in red



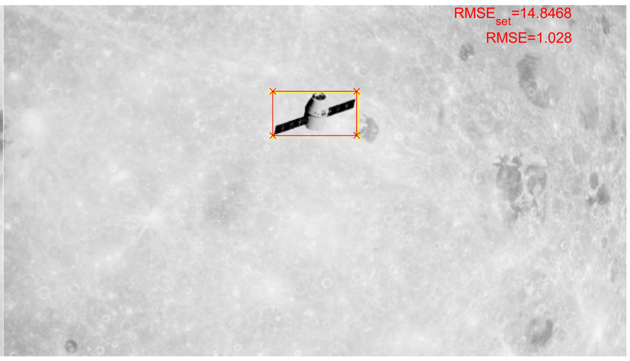
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.12: MarCO, close Moon, 2

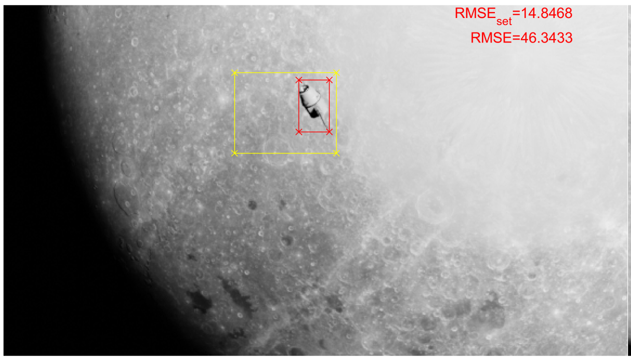
E.7 Dragon, close Moon



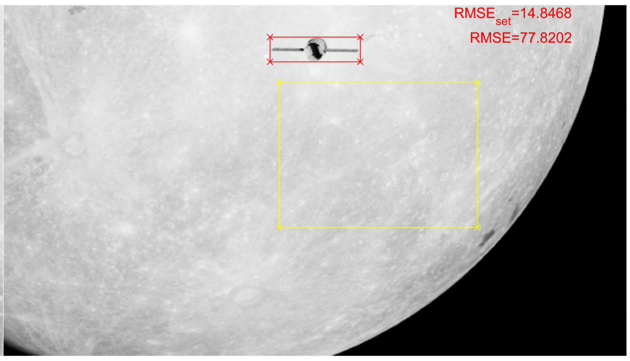
(a) Best estimation, 1



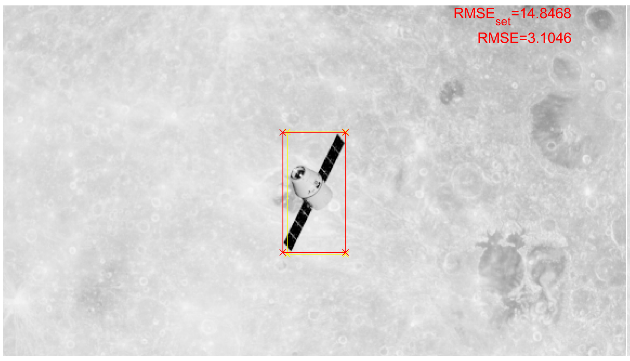
(b) Best estimation, 2



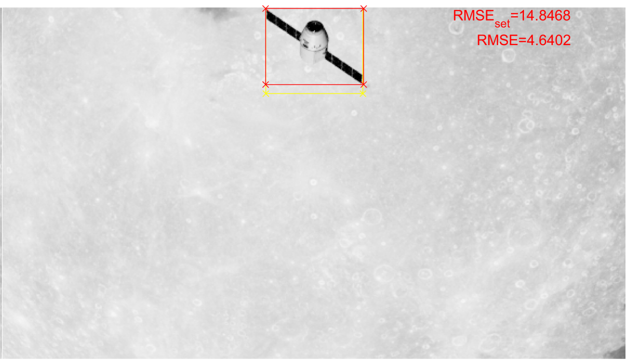
(c) Worst estimation, 1



(d) Worst estimation, 2

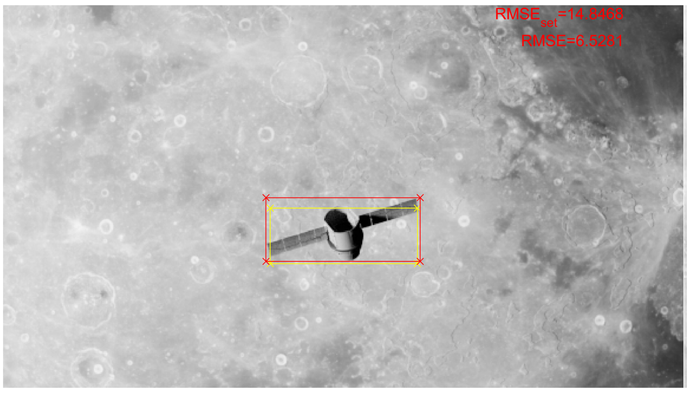


(e) Random example, 1



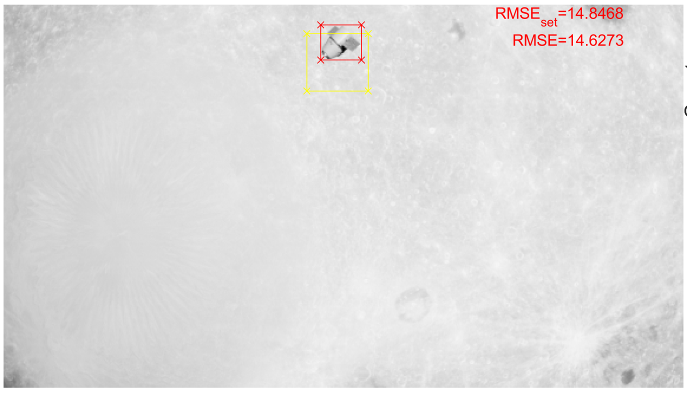
(f) Random example, 2

Fig. E.13: Dragon, close Moon, 1

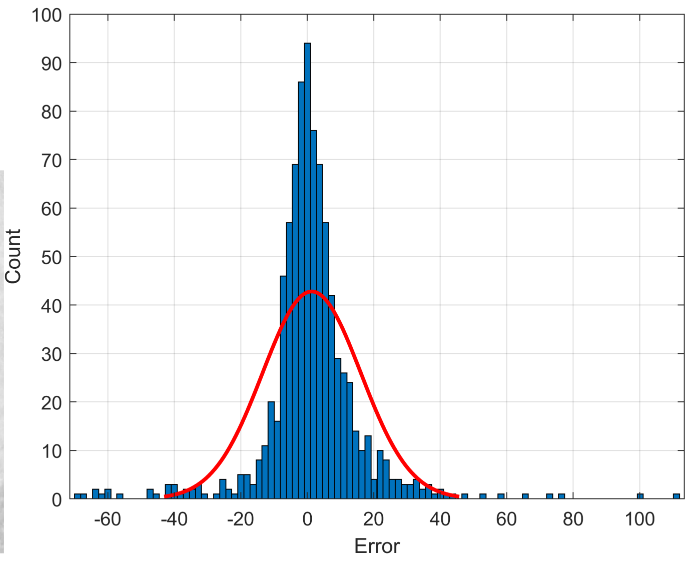


(a) Random example, 3

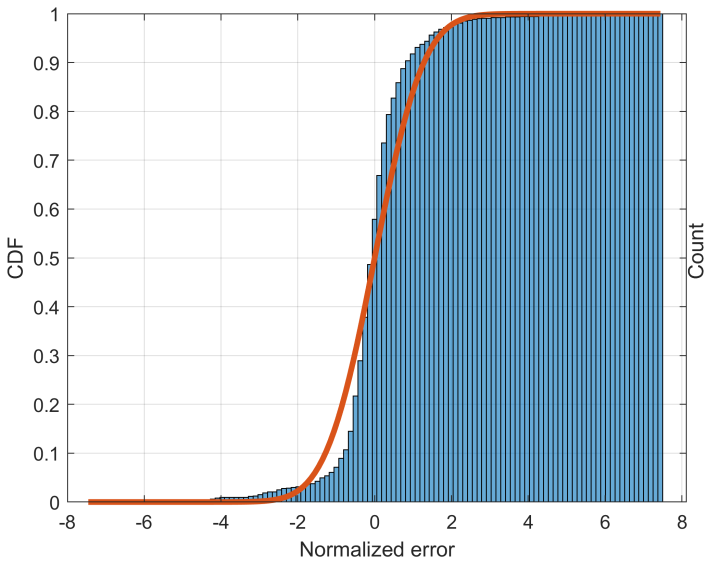
(b) Random example, 4



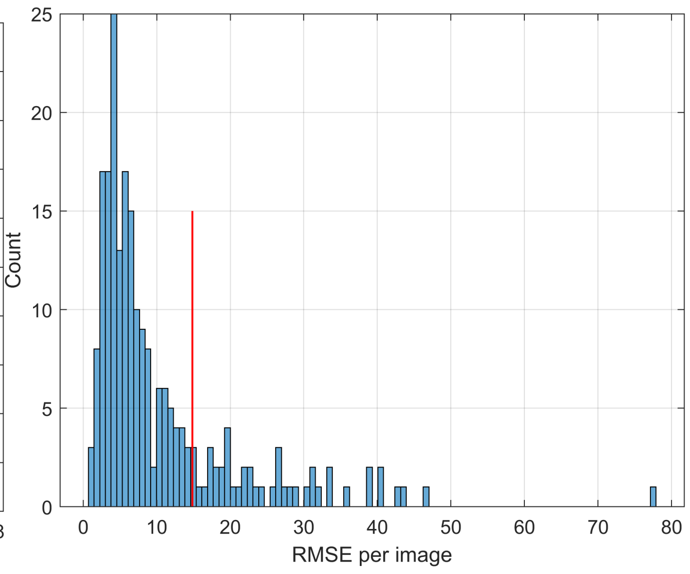
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



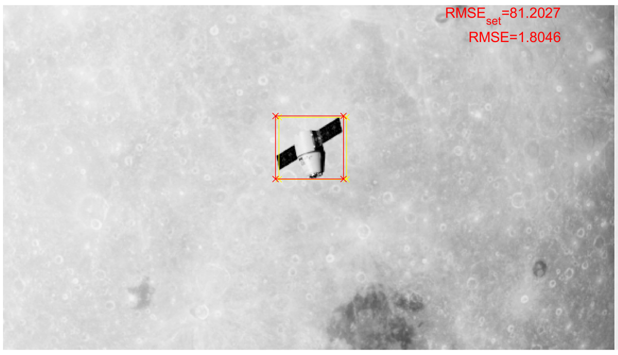
(e) CDF vs normal CDF in red



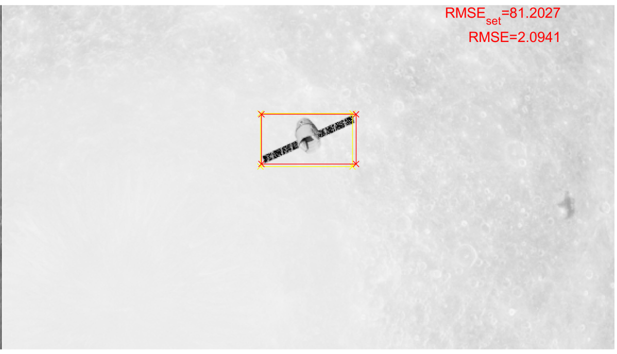
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.14: Dragon, close Moon, 2

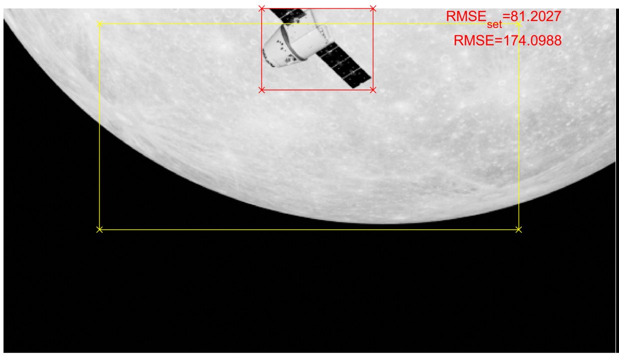
E.8 Dragon, far Moon



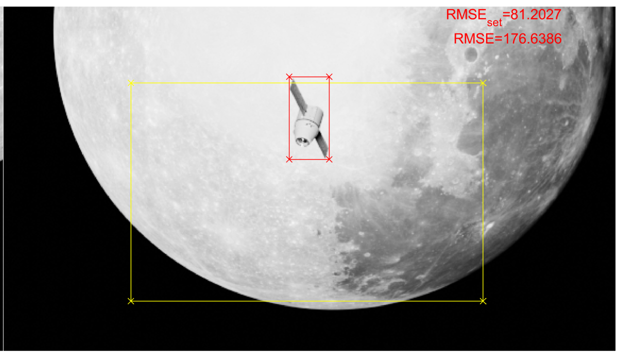
(a) Best estimation, 1



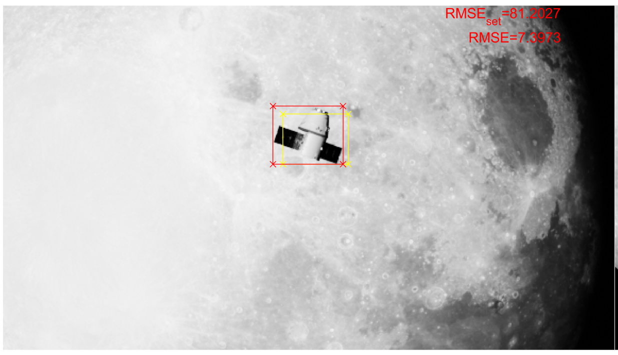
(b) Best estimation, 2



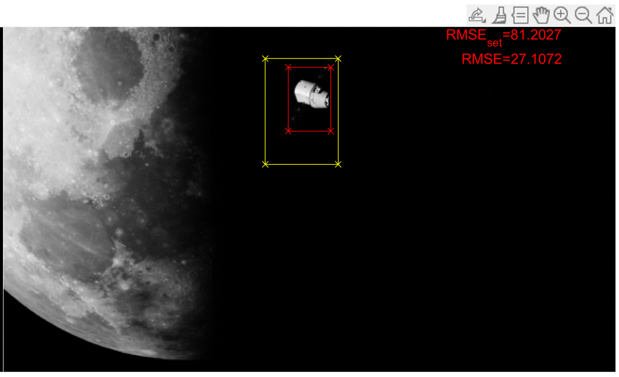
(c) Worst estimation, 1



(d) Worst estimation, 2

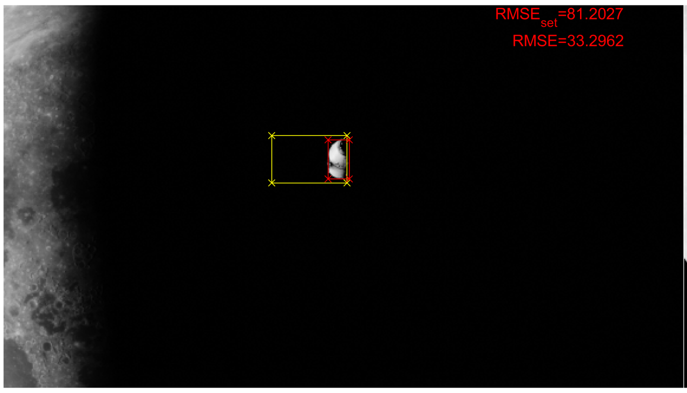


(e) Random example, 1

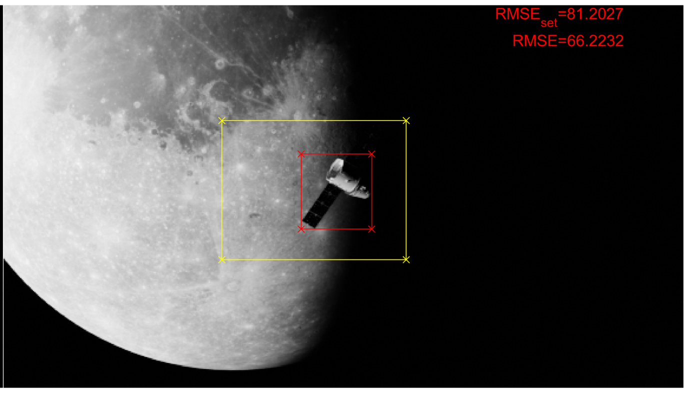


(f) Random example, 2

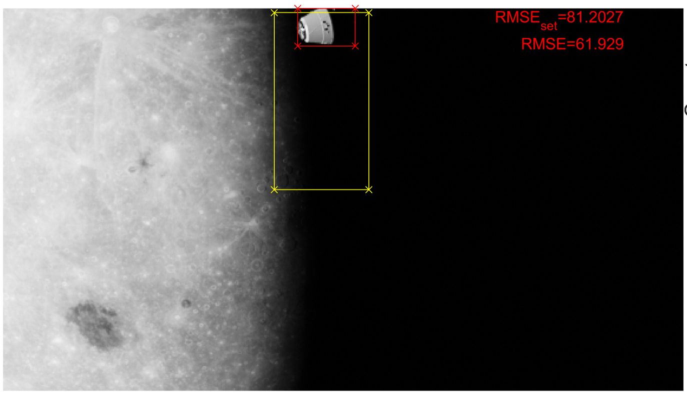
Fig. E.15: Dragon, far Moon, 1



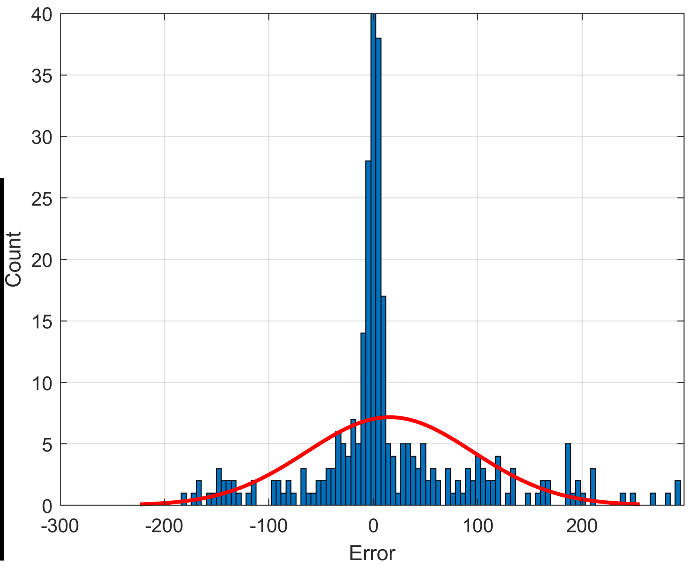
(a) Random example, 3



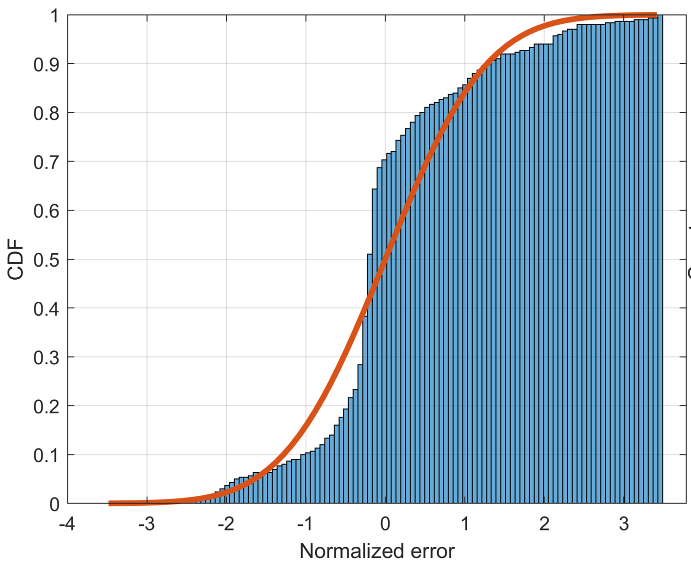
(b) Random example, 4



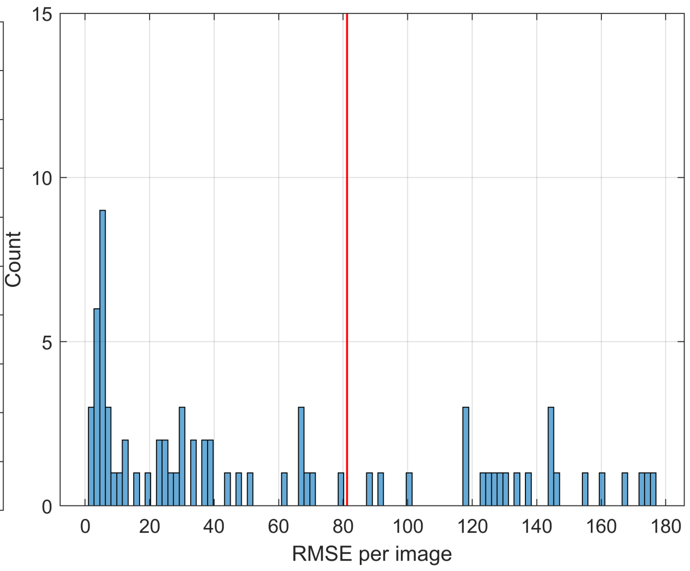
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



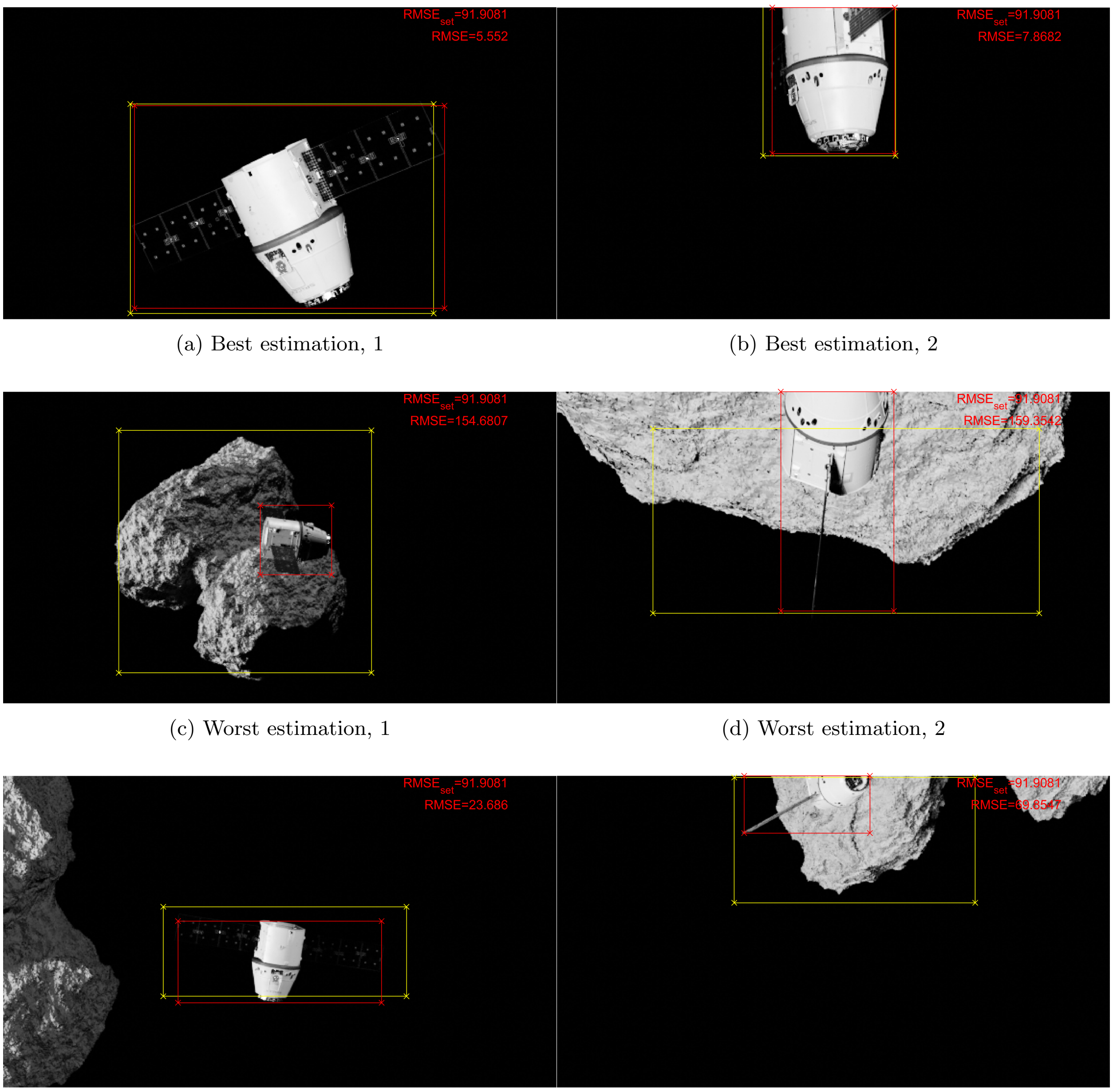
(e) CDF vs normal CDF in red



(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.16: Dragon, far Moon, 2

E.9 Dragon, far Comet



(a) Best estimation, 1

(b) Best estimation, 2

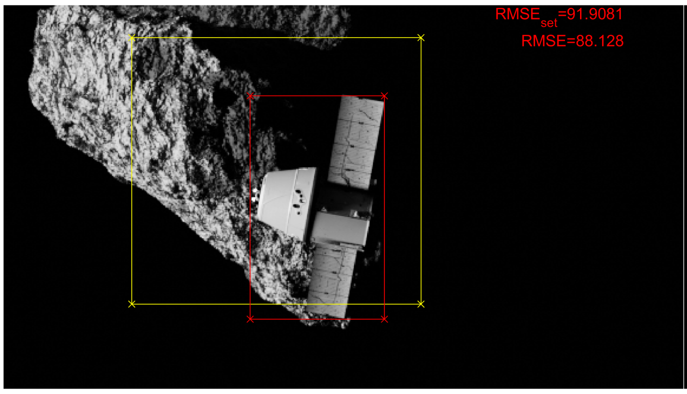
(c) Worst estimation, 1

(d) Worst estimation, 2

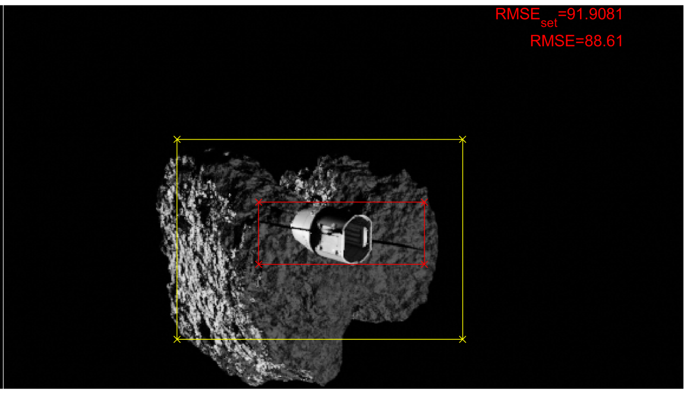
(e) Random example, 1

(f) Random example, 2

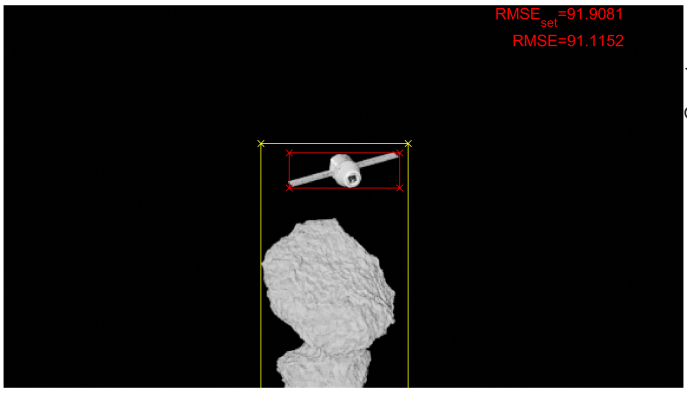
Fig. E.17: Dragon, far Rosetta comet, 1



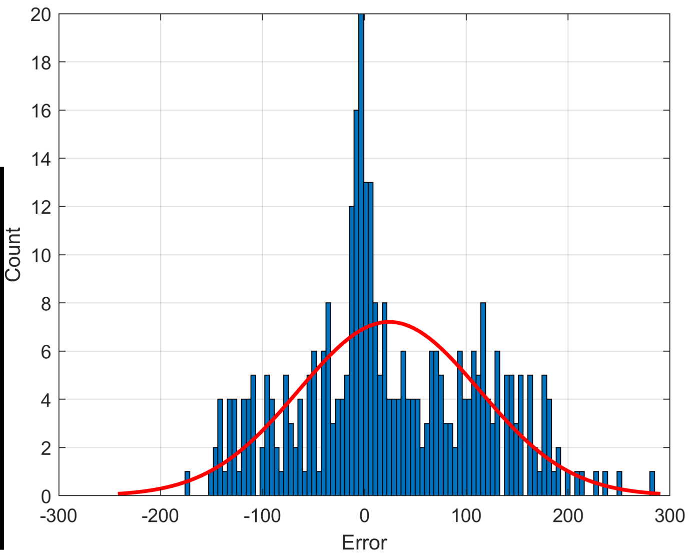
(a) Random example, 3



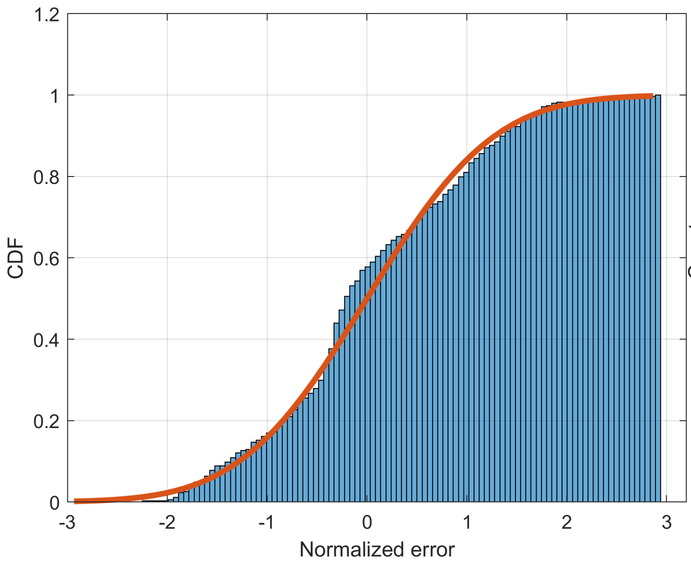
(b) Random example, 4



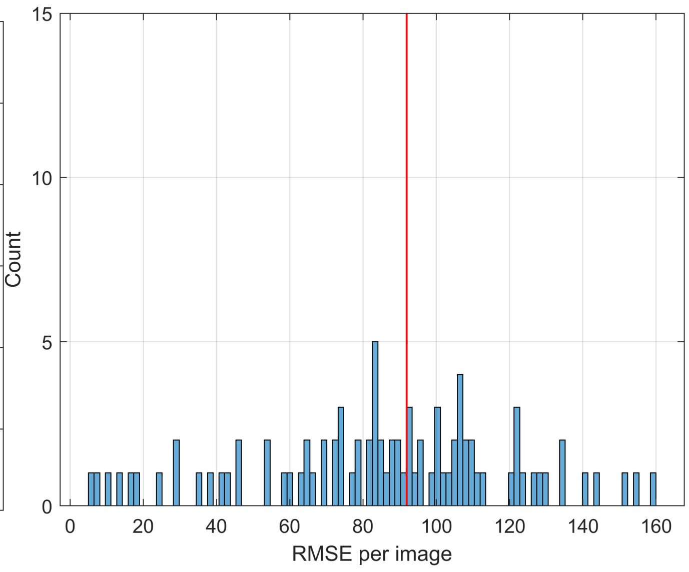
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



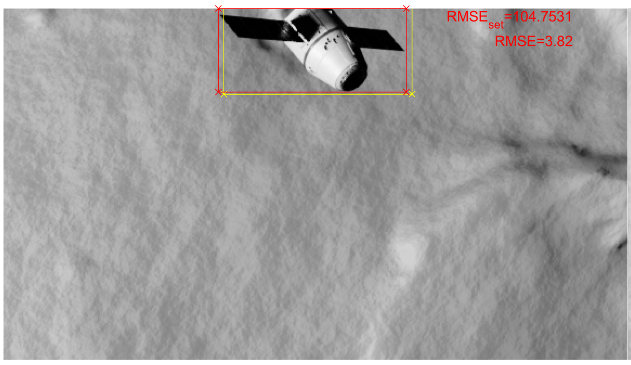
(e) CDF vs normal CDF in red



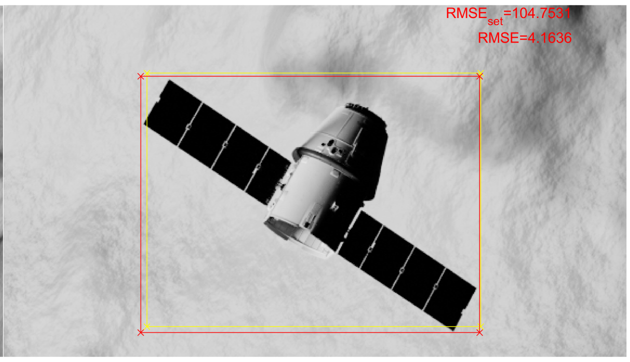
(f) i-th image RMSE histogram, dataset RMSE in red

Fig. E.18: Dragon, far Rosetta comet, 2

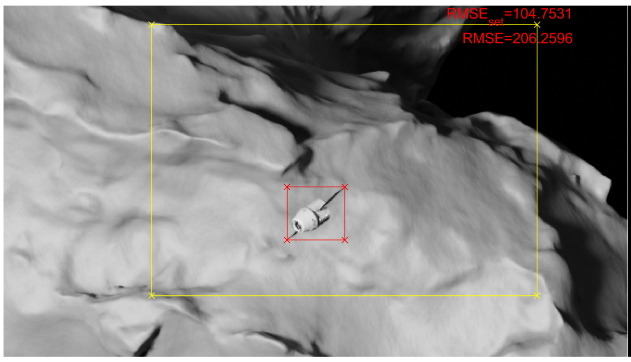
E.10 Dragon, close Comet



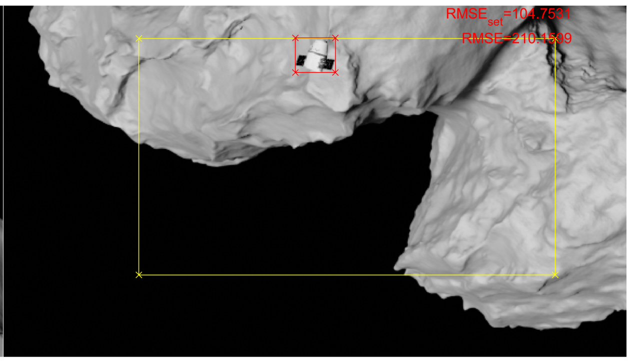
(a) Best estimation, 1



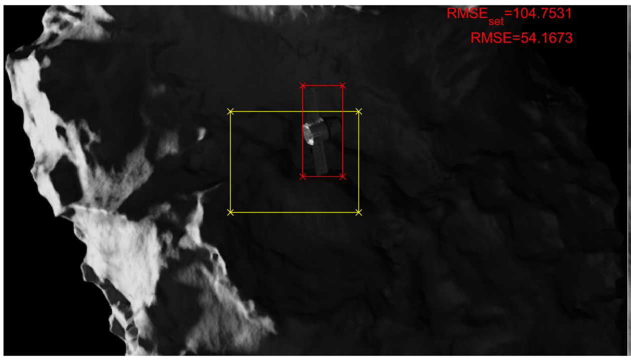
(b) Best estimation, 2



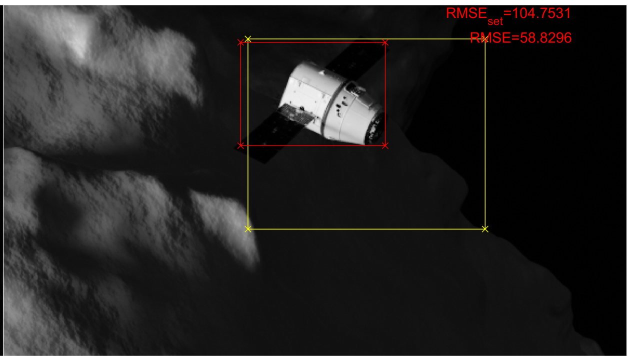
(c) Worst estimation, 1



(d) Worst estimation, 2

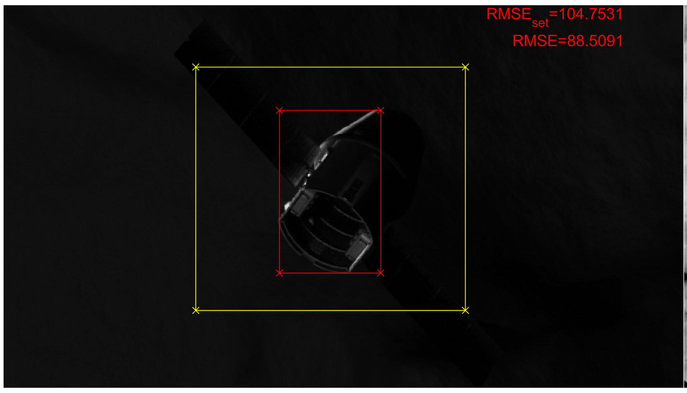


(e) Random example, 1

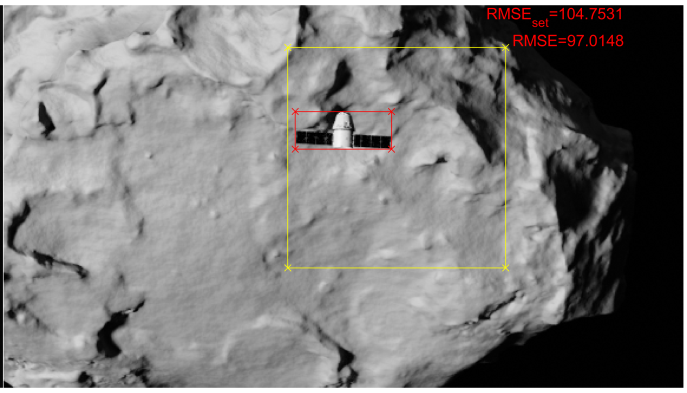


(f) Random example, 2

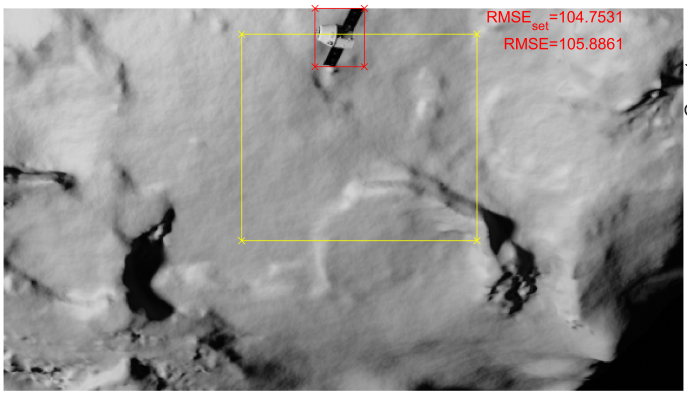
Fig. E.19: Dragon, close Rosetta comet, 1



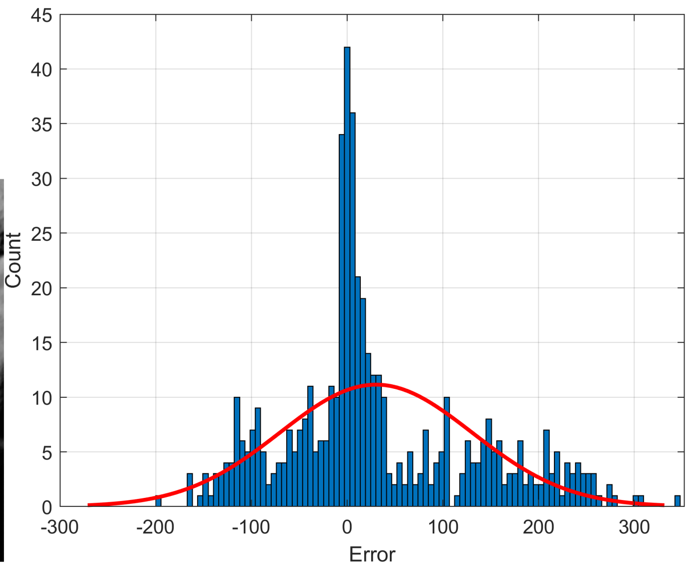
(a) Random example, 3



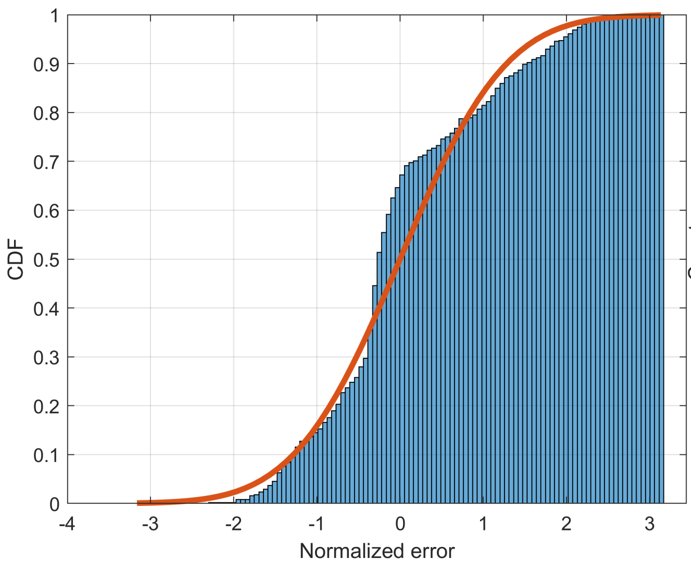
(b) Random example, 4



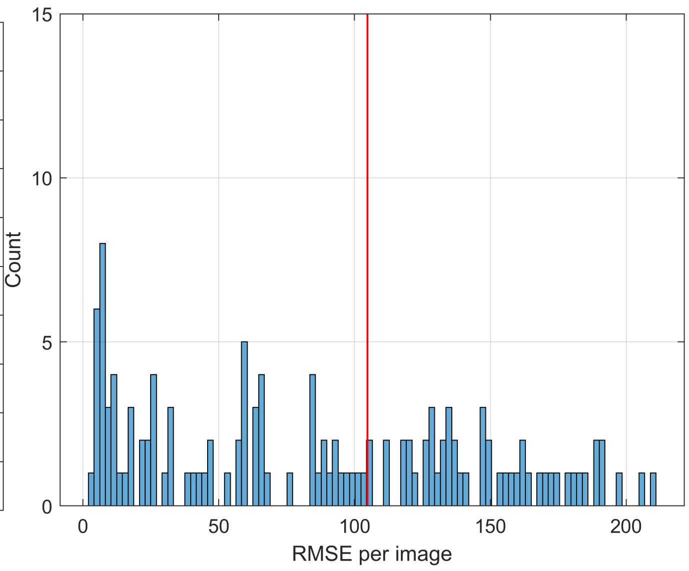
(c) Estimation close to RMSE example



(d) Error histogram vs fitting normal distribution in red



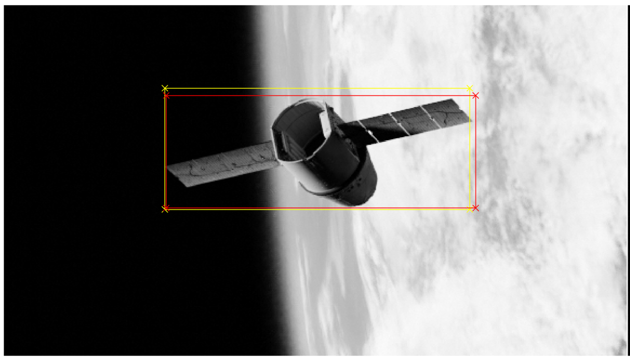
(e) CDF vs normal CDF in red

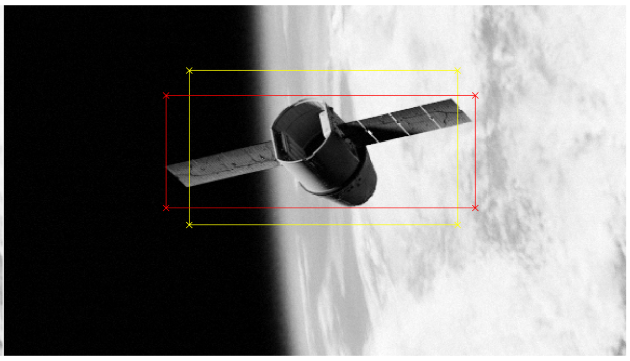


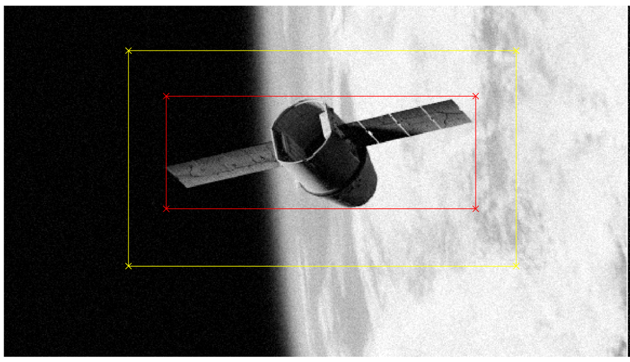
(f) i-th image RMSE histogram, dataset RMSE in red

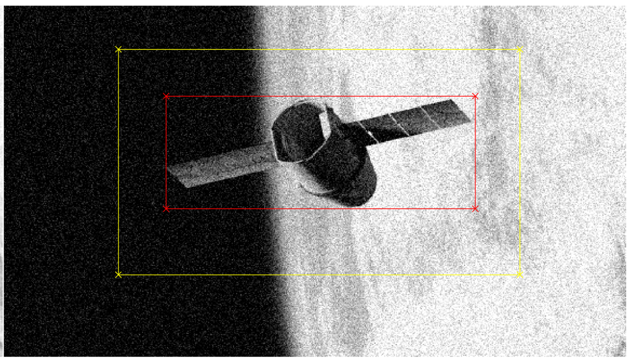
Fig. E.20: Dragon, close Rosetta comet, 2

E.11 Dragon, Earth & Space, Noise


(a) Noise example, Var $= 1e-5$


(b) Noise example, Var $= 1e-4$


(c) Noise example, Var $= 1e-3$


(d) Noise example, Var $= 1e-2$

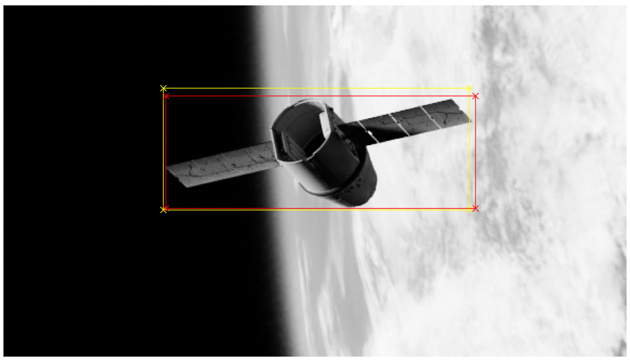
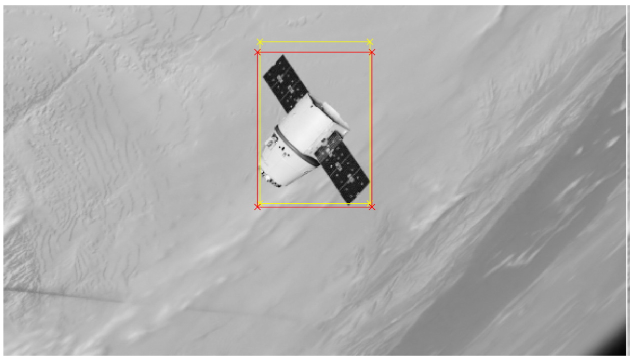
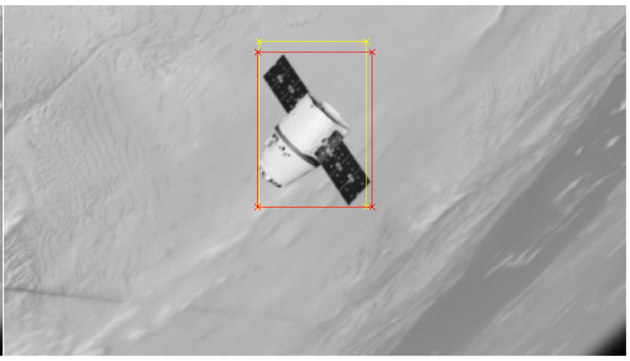
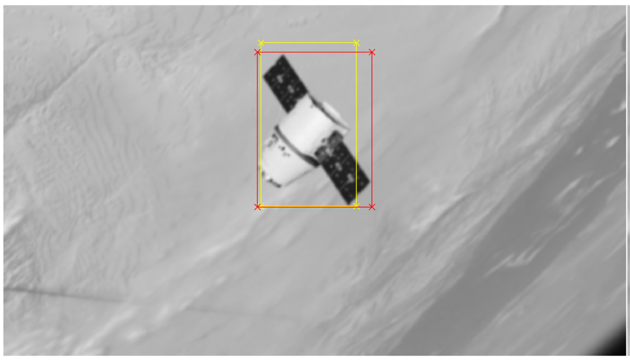

(e) Original picture

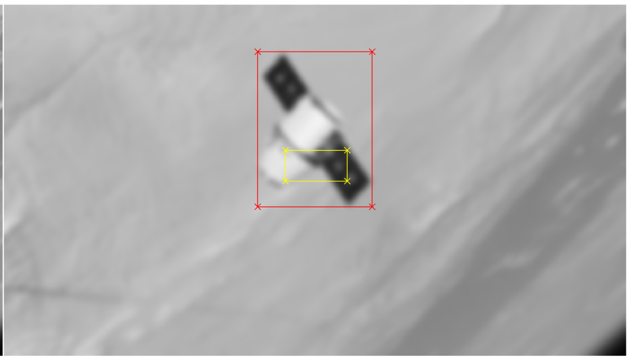
Fig. E.21: Dragon, Earth & Space, Noise

E.12 Dragon, Earth & Space, Blur


(a) Blur example, $\sigma = 0.3$


(b) Blur example, $\sigma = 0.7$


(c) Blur example, $\sigma = 1$


(d) Blur example, $\sigma = 3$

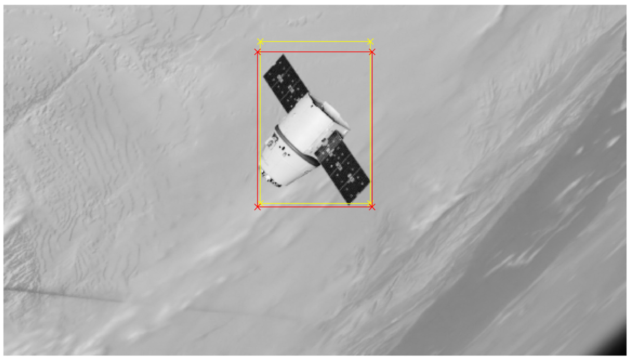

(e) Original picture

Fig. E.22: Dragon, Earth & Space, Blur

Acknowledgements

Vorrei iniziare prima di tutto ringraziando i miei genitori e mia sorella Francesca, che sono sempre stati presenti per supportarmi nei momenti importanti di questa esperienza universitaria. Il loro sostegno incondizionato è stato fondamentale per la mia crescita ed è anche grazie a loro che questa meta è stata raggiunta.

Un ringraziamento speciale alla Professoressa Michèle Lavagna, che mi ha dato l'opportunità di svolgere questa tesi ed allargare ed arricchire le mie conoscenze. La sua passione per l'ingegneria è sempre stata fonte di motivazione ed ispirazione anche prima di questo elaborato.

Vorrei ringraziare Margherita, che è sempre stata disponibile ad aiutarmi, revisionando spesso la mia tesi e dandomi indicazioni preziose anche per futuro. Il suo contributo è stato fondamentale per modellare questa tesi nella sua forma attuale.

Ringrazio anche il team ASTRA con cui ho condiviso molte esperienze, da quelle formative a quelle divulgative che mi hanno plasmato professionalmente ed anche caratterialmente. Un grazie a Stefano, che è stato sempre disponibile ad aiutarmi durante gli inizi di questa tesi.

In ultimo, ma non per importanza, vorrei ringraziare tutti i miei amici e parenti che mi hanno sostenuto ed accompagnato durante questi anni, dai compagni di corso, con cui ho condiviso tutti i momenti della vita universitaria, agli amici storici, con cui ho condiviso molti momenti di svago e tranquillità.

Marco Pittoni

References

1. Rosetta spacecraft, https://sci.esa.int/web/rosetta/-/53555-rosetta-instruments.
2. R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, ORB-SLAM: A versatile and accurate monocular slam system, IEEE Transactions on Robotics 31 (5) (2015) 1147–1163. doi:10.1109/tro.2015.2463671.
URL http://dx.doi.org/10.1109/TRO.2015.2463671
3. R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2004.
4. Lens diagram, https://www.kielia.de/photography/calculator/lens-equation/.
5. S. Prince, Computer vision : models, learning, and inference, Cambridge University Press, 2012.
6. Biological and artificial neuron illustration, https://www.researchgate.net/figure/Analogy-between-artificial-neuron-and-biological-neuron_fig1_7947079.
7. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
8. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
9. Ray tracing diagram, https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#/media/File:Ray_trace_diagram.svg.
10. Spacex images, https://twitter.com/spacex.
11. Nasa Image Galleries, https://www.nasa.gov/multimedia/imagegallery/index.html.
12. Svd visual illustration, https://en.wikipedia.org/wiki/Singular_value_decomposition#/media/File:Singular-Value-Decomposition.svg.
13. F. Amzajerdian, D. F. Pierrottet, G. D. Hines, L. B. Petway, B. W. Barnes, Doppler lidar sensor for precision navigation in GPS-deprived environment, in: M. D. Turner, G. W. Kamerman (Eds.), Laser Radar Technology and Applications XVIII, Vol. 8731, International Society for Optics and Photonics, SPIE, 2013, pp. 123 – 128. doi:10.1117/12.2018359.
URL https://doi.org/10.1117/12.2018359
14. T. Fong, Capability overview of nasa autonomous systems, https://www.nasa.gov/sites/default/files/atoms/files/nac_tie_aug2018_tfong_tagged.pdf.
15. Ets-7 mission, https://global.jaxa.jp/projects/sat/ets7/index.html.
16. Xss-11 mission, https://www.kirtland.af.mil/Portals/52/documents/AFD-111103-035.pdf?ver=2016-06-28-110256-797.
17. Dart mission, https://www.nasa.gov/mission_pages/dart/spacecraft/index.html.
18. Tridar, automating a better rendezvous in space, https://www.nasa.gov/mission_pages/station/research/news/b4h-3rd/it-automating-better-space-rendezvous/.
19. J. A. Christian, S. Cryan, A Survey of LIDAR Technology and its Use in Spacecraft Relative Navigation. arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2013-4641, doi:10.2514/6.2013-4641.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2013-4641
20. Rosetta instruments sheet, https://www.esa.int/Science_Exploration/Space_Science/Rosetta/Rosetta_Media_factsheet.
21. Rosetta navcam instrument, https://pdssbn.astro.umd.edu/catalogs/Rosetta/navcam_inst.cat.
22. Rosetta osiris instrument, https://www.mps.mpg.de/1979623/OSIRIS.
23. Hayabusa optical system, http://spaceflight101.com/spacecraft/hayabusa-2/.
24. Osiris-rex instruments, https://www.asteroidmission.org/objectives/instruments/.
25. R. P. de Santayana, M. Lauer, Optical measurements for Rosetta navigation near the comet, 2015.
26. F. Castellini, R. Santayana, D. Wokes, S. Kielbassa, Optical navigation for Rosetta operations near comet churyumov-gerasimenko, Advances in the Astronautical Sciences 150 (2014) 1619–1637.
27. P. A, G. M, F. M, G. J, C. I, HERA vision based gnc and autonomy, 2019.
28. J. Gil-Fernández, M. Casasco, I. Carnelli, P. Martino, M. Kueppers, HERA autonomous guidance, navigation and control experiments: Enabling better asteroid science future missions, 2019.

29. S. Segal, A. Carmi, P. Gurfil, Stereovision-based estimation of relative dynamics between noncooperative satellites: Theory and experiments, Control Systems Technology, IEEE Transactions on 22 (2014) 568–584. doi:10.1109/TCST.2013.2255288.
30. V. Pesce, Stereovision-based pose and inertia estimation for unknown and uncooperative space objects. URL https://www.politesi.polimi.it/handle/10589/107902
31. H. Strasdat, J. Montiel, A. Davison, Real-time monocular slam: Why filter?, 2010, pp. 2657–2664. doi:10.1109/ROBOT.2010.5509636.
32. M. Dor, P. Tsiotras, ORB-SLAM Applied to Spacecraft Non-Cooperative Rendezvous, 2018. doi:10.2514/6.2018-1963.
33. H. Lin, J. Si, G. P. Abousleman, region-of-interest detection and its application to image segmentation and compression, in: 2007 International Conference on Integration of Knowledge Intensive Multi-Agent Systems.
34. J. L. Solka, D. J. Marchette, B. C. Wallet, V. L. Irwin, G. W. Rogers, Identification of man-made regions in unmanned aerial vehicle imagery and videos, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8) (1998) 852–857.
35. A. Mohan, P. CP, T. Poggio, Example-Based Object Detection in Images by Components, Pattern Analysis and Machine Intelligence, IEEE Transactions on 23 (2001) 349 – 361. doi:10.1109/34.917571.
36. T. M. Stough, C. E. Brodley, Focusing attention on objects of interest using multiple matched filters, IEEE Transactions on Image Processing 10 (3) (2001) 419–426.
37. J. Kuklyte, K. McGuinness, R. Hebbalaguppe, C. Direkoglu, L. Gualano, N. E. O'Connor, Identification of moving objects in poor quality surveillance data, in: 2013 14th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), 2013, pp. 1–4.
38. M. Goyal, S. Hassanpour, M. H. Yap, Region of Interest Detection in Dermoscopic Images for Natural Data-augmentation (2018). arXiv:1807.10711.
39. T. M. Mitchell, Machine Learning, McGraw-Hill, 1997.
40. Copernicus Sentinel-1 and deep learning help advance sea ice information service, https://sentinel.esa.int/web/sentinel/home/-/journal_content/56/247904/3944958.
41. X. Yang, H. Sun, K. Fu, J. Yang, X. Sun, M. Yan, Z. Guo, Automatic Ship Detection of Remote Sensing Images from Google Earth in Complex Scenes Based on Multi-Scale Rotation Dense Feature Pyramid Networks, Remote Sensing 10 (2018) 132. doi:10.3390/rs10010132.
42. M. Piccinin, V. Pesce, S. Silvestrini, M. Lavagna, Smart Autonomous Imaging Plan For Small Bodies Efficient Mapping, 2019.
43. D. M. Chan, A. Agha-mohammadi, Autonomous Imaging and Mapping of Small Bodies Using Deep Reinforcement Learning, in: 2019 IEEE Aerospace Conference, 2019, pp. 1–12.
44. Esa kelvins, https://kelvins.esa.int/.
45. A. Bordone Molini, D. Valsesia, G. Fracastoro, E. Magli, DeepSUM: Deep Neural Network for Super-Resolution of Unregistered Multitemporal Images, IEEE Transactions on Geoscience and Remote Sensing 58 (5) (2020) 3644–3656. doi:10.1109/tgrs.2019.2959248.
URL http://dx.doi.org/10.1109/TGRS.2019.2959248
46. S. Sharma, C. Beierle, S. D'Amico, Pose Estimation for Non-Cooperative Spacecraft Rendezvous Using Convolutional Neural Networks (2018). arXiv:1809.07238.
47. S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (2015). arXiv:1506.01497.
48. satellite pose estimation with deep landmark regression and nonlinear pose refinement.
49. T. Uriot, D. Izzo, L. Simoes, R. Abay, N. Einecke, S. Rebhan, J. Martinez-Heras, F. Letizia, J. Siminski, K. Merz, Spacecraft Collision Avoidance Challenge: design and results of a machine learning competition (2020). arXiv:2008.03069.
50. Gstp Element 1 "Develop" Compendium 2019: Artificial Intelligence, http://emits.sso.esa.int/emits-doc/ESTEC/News/GSTPAICompedium2019.pdf.
51. Pangu software, https://www.star-dundee.com/products/pangu-planet-and-asteroid-natural-scene-generation-ut
#technical_specs.

52. Surrender software, https://www.airbus.com/space/space-exploration/SurRenderSoftware.html.
53. Blender project, https://www.blender.org/.
54. M. Pomerantz, A. Jain, S. Myint, Dspace: Real-time 3d visualization system for spacecraft dynamics simulation, 2009, pp. 237 – 245.
55. S. Weikert, A. Dobler, J. Teufel, S. Schäff, V. Zuccarelli, M. Jürgens, A. Wiegand, S. O. Erb, ASTOS 8.1-mission performance analysis , system concept analysis and other new features, 2016.
56. P. Proença, Y. Gao, Deep Learning for Spacecraft Pose Estimation from Photorealistic Rendering (07 2019).
57. P. Biscari, T. Ruggeri, G. Saccomandi, M. Vianello, Meccanica Razionale, Springer, 2013.
58. K. Alfriend, S. R. Vadali, P. Gurfil, J. How, L. Breger, Spacecraft Formation Flying Dynamics, control and navigation, Elsevier Astrodynamics - Butterworth-Heinemann, 2009.
59. S. Segal, P. Gurfil, Effect of kinematic rotation-translation coupling on relative spacecraft translational dynamics, Journal of Guidance, Control, and Dynamics 32 (3) (2009) 1045–1050. arXiv:https://doi.org/10.2514/1.39320, doi:10.2514/1.39320.
URL https://doi.org/10.2514/1.39320
60. J. P. Mena-Chalco, Ray/triangle intersection, https://www.mathworks.com/matlabcentral/fileexchange/25058-ray-triangle-intersection.
61. L. Losi, M. Lavagna, Visual Navigation for Autonomous Planetary Landing, 2016.
URL https://www.politesi.polimi.it/handle/10589/123100
62. M. A. Fischler, R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACMdoi:10.1145/358669.358692.
URL https://doi.org/10.1145/358669.358692
63. P. Torr, A. Zisserman, Mlesac: A new robust estimator with application to estimating image geometry, Computer Vision and Image Understanding 78 (2000) 138–156. doi:10.1006/cviu.1999.0832.
64. C. Tomasi, T. Kanade, Detection and tracking of point features, Tech. rep., International Journal of Computer Vision (1991).
65. J. Sturm, Lecture 5: Visual navigation for flying robots, https://www.youtube.com/watch?v=4eg0JtVT5Zc&list=PLTBdjV_4f-EKeki5ps2WHqJqyQvxls4ha&index=5.
66. J. Y. Bouguet, Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm, Tech. rep., Intel Corporation.
URL http://robots.stanford.edu/cs223b04/algo_affine_tracking.pdf
67. J. K. Suhr, Kanade-lucas-tomasi feature tracker, https://web.yonsei.ac.kr/jksuhr/articles/Kanade-Lucas-Tomasi%20Tracker.pdf.
68. J. Sturm, Lecture 6: Visual navigation for flying robots, https://www.youtube.com/watch?v=HuzCOMAlo0w.
69. Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, Hang-Fei Cheng, Complete solution classification for the perspective-three-point problem, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (8) (2003) 930–943.
70. V. Lepetit, F. Moreno-Noguer, P. Fua, Epnp: An accurate o(n) solution to the pnp problem, International Journal of Computer Vision 81. doi:10.1007/s11263-008-0152-6.
71. M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
URL http://neuralnetworksanddeeplearning.com
72. K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.
73. A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
74. A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi, A survey of the recent architectures of deep convolutional neural networks, Artificial Intelligence Reviewdoi:10.1007/s10462-020-09825-6.
URL http://dx.doi.org/10.1007/s10462-020-09825-6

75. S. H. Ahn, 2d convolution, http://www.songho.ca/dsp/convolution/convolution2d_example.html.
76. S. J. Kelly, A monocular slam method to estimate relative pose during satellite proximity operations.
77. C. Kuhn, Dragon v1 model, https://www.blendswap.com/blend/16155.
78. MarCO 3D Model, https://solarsystem.nasa.gov/resources/2373/marco-3d-model/.
79. 3D model of Rosetta's Comet 67P C-G, http://open.esa.int/rosetta-3d-model/.
80. F. Lasse, Photorealistic Earth model, https://gumroad.com/l/MwdT.
81. CGI Moon Kit, https://svs.gsfc.nasa.gov/4720.
82. Nvidia ray tracing technology, https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/.
83. J. P. Mena-Chalco, Ray/triangle intersection, https://www.mathworks.com/matlabcentral/fileexchange/25058-ray-triangle-intersection.
84. 67P/Churyumov-Gerasimenko comet shape model, http://open.esa.int/rosetta-3d-model/.
85. J. Llamas, P. Lerones, R. Medina, E. Zalama, J. Gómez-García-Bermejo, Classification of architectural heritage images using deep learning techniques, Applied Sciences 7 (2017) 992. doi:10.3390/app7100992.
86. Mars Cube One, https://www.jpl.nasa.gov/cubesat/missions/marco.php.
87. VV02 – Vega uses Vespa, http://www.esa.int/Enabling_Support/Space_Transportation/Launch_vehicles/VV02_Vega_uses_Vespa.
88. Envisat, https://www.esa.int/Enabling_Support/Operations/Envisat.
89. Singular values and svd, https://www.mathworks.com/help/matlab/math/singular-values.html.
90. K. Kitani, Svd for total least squares, http://www.cs.cmu.edu/~16385/s17/Slides/11.5_SVD.pdf.
91. A. Quarteroni, F. Saleri, P. Gervasio, Calcolo scientifico, Springer, 2012.
92. LM algorithm, https://mathworld.wolfram.com/Levenberg-MarquardtMethod.html.
93. Vespa source image for dimensions, https://www.arianespace.com/mission-update/ready-for-liftoff-vega-is-authorized-for-its-may-3-flight-from-the-spaceport-2/.
94. Vega user manual, https://www.arianespace.com/wp-content/uploads/2018/07/Vega-C-user-manual-Issue-0-Revision-0_20180705.pdf.
95. Feature descriptors, Matlab©, https://it.mathworks.com/help/vision/ref/extractfeatures.html.
96. D. DeTone, T. Malisiewicz, A. Rabinovich, Deep image homography estimation.